



《51 单片机 C 语言快速上手》

提示: 本教程共有 48 题动手实验, 36 题自我练习。而且全部程序都在 SSH_51MCU 实验板中调试通过。而第五章的第五节与第六节有部分的截图是出自数据手册。

第一章 学习单片机的前奏

1. 如何学习单片机（初学者必读）:

什么是单片机? 既然你已经正在看这份教程, 想必你已经有一定的了解, 但是如果让你去编写一个单片机程序你能胜任吗! 很多想学单片机的人问我的第一句话就是怎样才能学好单片机? 现在有好多的电子爱好者都想踏进单片机的大门, 但是总不知如何入手, 对于这个问题在这里我就以我是如何开始学习单片机, 如何入门, 给大家讲讲。希望大家在学习过程中有一个明确的方向。

学习单片机最好是从 51 系列的开始学起, 因为它的资料比较多, 其技术在内国非常的成熟, 用的人很多, 市场也很大。可以说现在大学学习单片机的教程都是从 51 单片机入手的。单片机是一门实践性非常高的学科。老是看书是绝对不行的, 书看多了只会理论, 真来动手的时候就不知如何下手。我记得我读大学的那一年老师曾这样讲过: 单片机是一门理论与实践比例为 2 比 8 的学科。但是看书还是必要的。因为我们要从书上大概了解一下单片机的各个功能寄存器, 那为什么要了解功能寄存器啊? 因为我们要用编写软件去控制单片机的各个功能寄存器。那目的为什么啊? 最简单的说法就是控制单片机的引脚什么时候输出高电平! 什么时候输出高电平! 从而去控制我们的电路系统。其实, 如果只是光靠看书而没有动手实践, 不要说是学会什么单片机了, 就是只看书本的前几十页你就会想睡觉了, 因为老是看书你会觉得很闷! 唯有不停的动手才会引起你的兴趣! 至于看书, 只需大至了解单片机各功能寄存器是如何操作的? 能实现什么样的功能? 在看完第一编后你还没有明白, 第二编你可能还是不明白, 但这不要紧, 因为你还缺少实际的感观认识! 只要动手做起来你就会觉得: “原来是这么简单!”。所以本教程当中都是在理论的引导下以动手为主的。而且还在每一节实验完后配合大量的自我练习。从而引起你的不断思考, 提高你的动手能力和兴趣。

再系统地学习完本教程之后, 我相信你对单片机已经有了一个系统的认识, 因为关于单片机硬件与软件你已经入门了。我想有一句话大家应该听讲吧“万事开头难”, 既然前面最难的那一关你都已经走过来了, 那下面的路就好走了。记住, 在后面的时间你要不断地练习如何去写程序; 要充分利用实验板去做练习, 只有这样你才能不断的积累经验, 在此同时, 我建议大家少看电影, 少上 Q 聊天, 少玩游戏。多化些时间坐在电脑前学习调试程序, 开始的时候要从最简单的流水灯开始做实验, 当你成功将 8 只 LED 灯按照你的意愿点亮时, 你就会感觉到原来单片机是那么的有趣! 你不要把学习单片机认为是一种技术, 你不是在学习知识, 你是在玩。这样你慢慢就会上瘾了, 真的, 搞电子类的人真会上瘾的, 然后再去学习如何点亮数码管, 当你成功完成了这两项, 我可以同你讲, 你已经不能自拔了! 你肯定会自己跟自己说: 如果我这一辈子都能在工作上搞电子这一行就好啦!

在学习过程中，肯定会遇到这样或那样的问题，而这时你再去翻书，再去别人，当你把问题解之后，我相信你一辈子都会记得，知识必须用于现实生活中，解决实际问题，这样才能发挥它的作用。在最后我想同大家讲讲关于 C 语言与汇编语言的问题，在单片机编程当中，你可以用 C 语言也可以用汇编语言，但是我建议大家用 C 语言，因为 C 语可移植性高，易读懂，而且效率也非常的高，相对来讲汇编语言则难度较大，而且很罗嗦，尤其是遇到算法方面的问题时，根本就整个人变疯了。你可以一点汇编都不懂也没有关系，但是如果你一点 C 语言都不懂的话我相信你以后肯定会吃苦头。而且现在的单片机主频和 ROM 都在不断提高，足够装下你所用的任何代码。C 语言的资料又多又好找，如果以后想将程序移植到另外一款单片机当中去，只需改变相应的端口和寄存器就可以了。所以我劝大家还是用 C 语言比较好。

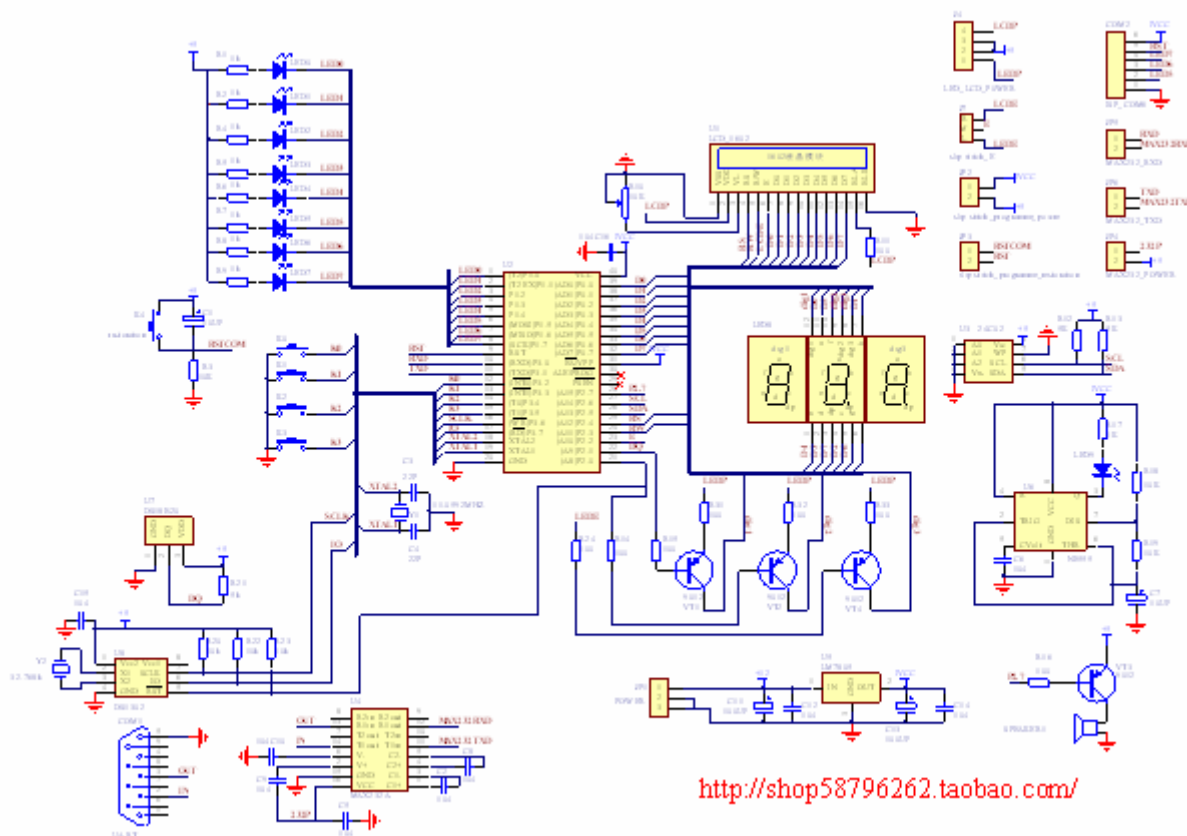
总的来说，不论是学习单片机还是做其它事情，只要能坚持到底，有不成功不放弃的念头和意志，就是已经成功了一半。下面我们不要废话了，让我们马上开始单片机世界漫游吧。

2. 单片机的学习工具：

对于初学者来讲，我们必须要多动手，所以一套功能齐全的工具是必不可少的。下面我们来介绍本教程所使用的 SSH_51MCU 型实验板及其一些相应的工具。



(SSH_51MCU 实验板，SST_51 仿真器，USB_ASP 下载线，配套工具)图 1-1



(SSH_51MCU 实验板电路图) 图 1-2

在图 1-1 中分别是SSH_51MCU实验板、SST_51 仿真器、USP_ASP下载线。而图 1-2 是SSH_51MCU 实板电路图。关于本教程学习工具的更多信息读者可以点击这里：

http://item.taobao.com/auction/item_detail-0db2-0aaa30452e89a2395d2c9d1ca5ab6cd9.htm

<http://shop58796262.taobao.com/>

3. 单片机初步：

特别提示：以下对 USB ASP 下载线进行介绍，这只是让初学者懂得如何操作使用，绝对没有侵犯知识产权的意图！其技术知识产权归原作者所有，望原作者理解。

在这里我们来熟悉一下 USP ASP 下载线的使用，也就是如何将编写的程序代码烧写到单片机中去，让其在电路中脱机运行。而关于 SSH_51MCU 实验板、SST_51 仿真器在随后的动手实验中会有详细的讲解。

(1)、安装 USB ASP 下载线的驱动。

当我们把下载线插到电脑的 USB 接口，即出来以下的图案（如图 1-3）



图 1-3

出现安装新硬件向导的时候我们选择“否，暂时不”，因为我们要手动安装驱动（如图 1-4）



图 1-4

再点击下一步，点浏览选择“windows 端驱动”文件夹的存放路径（驱动附光盘中）。如图 1-5。



图 1-5

再点击下一步进行安装，如图 1-6。



图 1-6

经过上面图 1-6 的界面之后会出现如图 1-7 的画面，此时 USB ASP 驱动程序安装成功。



图 1-7

点击完成之后，在电脑的右下角会出现如图 1-8 的图案，说明你的 USB ASP 下载线已经可以正常使用了。同时你还可以在电脑设备管理器中确认你的驱动是否成功安装。（如图 1-9）

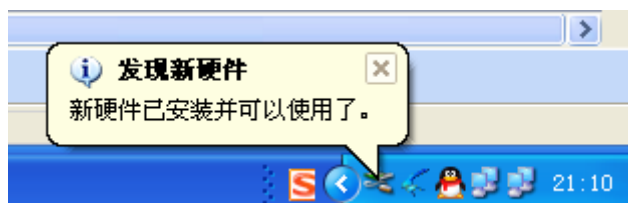


图 1-8



图 1-9

(2)、安装 USB_ASP 上位机。



双击

图标进行安装。如图 1-10

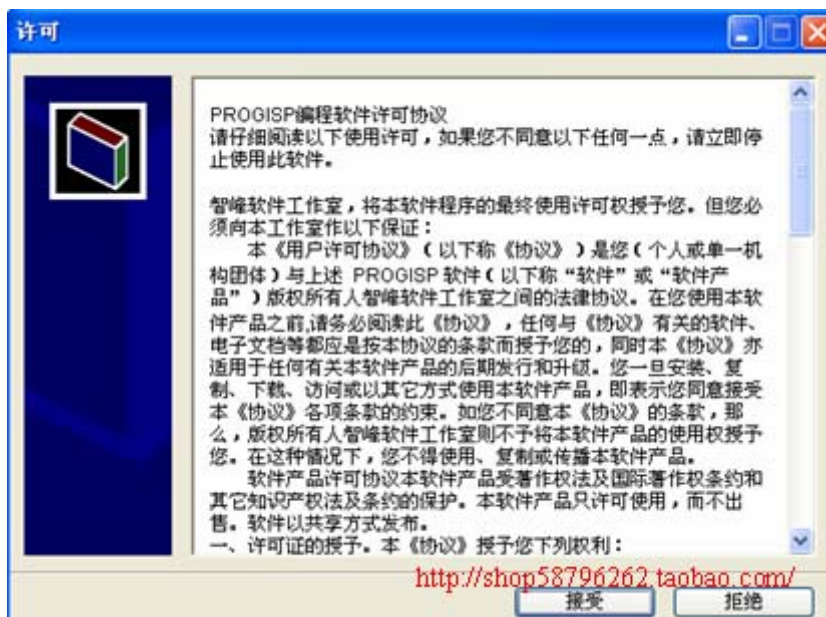


图 1-10



图 1-11

经过图 1-11 的安装之后，再弹出图 1-12 的界面，这就是 USB_ASP 的最终操作界面。



图 1-12

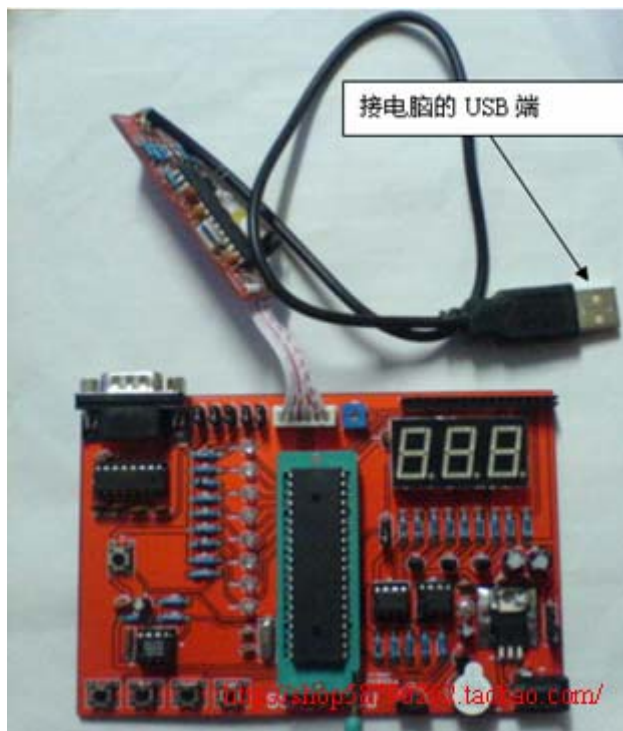
(3) 利用 USB_ASP 烧写程序

将下载线的另一端与 SSH_51MCU 实验板的 COM2 相接，正确把 AT89S52 单片机安装在实验板上，在安装时要注意引脚的对称，不能装反。

在这里有一项要特别注意：下载时实验板的供电有两种方式。

第一种：利用实验板的电源供电，这时必须要将 USB_ASP 下载线的 JP2 断开。

第二种：利用 USB_ASP 下载线进行供电，这时应接上下载线的 JP2，但是实验板就不能再接上电源供电。(注意：在做实验时不能用 USB 供电，这样有可能电流过大，烧坏 USB 端口)
在上面的两种方式中只能选其一，不能同时选择两种否则会有可能烧坏电脑的 USB。这点大家务必要注意。而无论选用上面的任可一种方式进行烧写程序时，都要将实验板的 JP2、JP3 断开。在本教程中我们就以 USB_ASP 下载线的电源作为供电方式。当把实验板与下载线正确连接之后，实验板的 LED9 不停在闪动，表示电源正常。如图 1-13



(下载线与 SSH_51MCU 实验板的正确接法)

图 1-13



图 1-14

在烧写之前我们可以先来确认一下 USB_ASP 下载线能否正确检测在实验板上的 AT89S52 芯片，在图 1-14 界面点击“芯片识别字”，有可能会出现一个信息提示对话框（如图 1-15A），其意思是：所

烧写芯片的识别字不对。在这一步可以点击“否(N)”跳过而不用理会它。但是如果出现如图 1-15B 的对话框则说明 USB_ASP 工作正常，但不能以所下载的单片通信，此时应检查单片机有没有与下载线正确连接上。

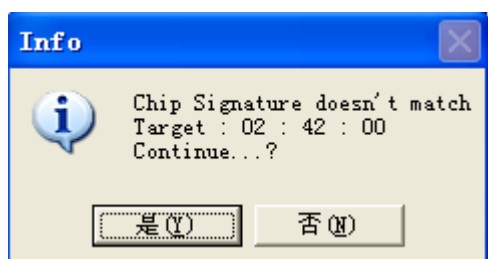


图 1-15A

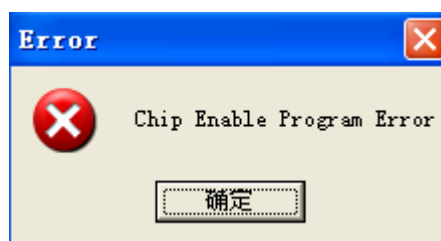


图 1-15B

跳过之后，我们可以点击 1-14 界面的“调入 Flash”装入我们要烧写的程序。在这里以第五章的“colligate_”为例。如图 1-16

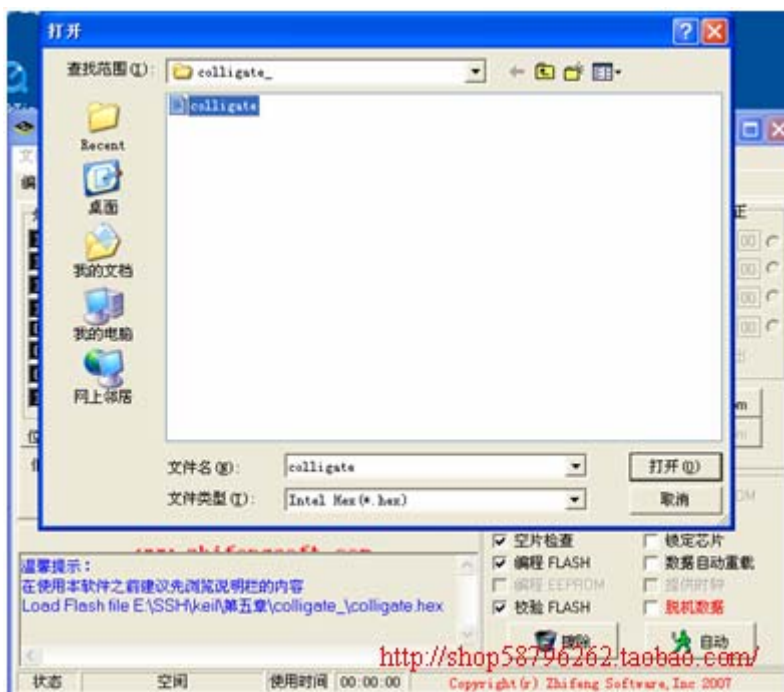


图 1-16

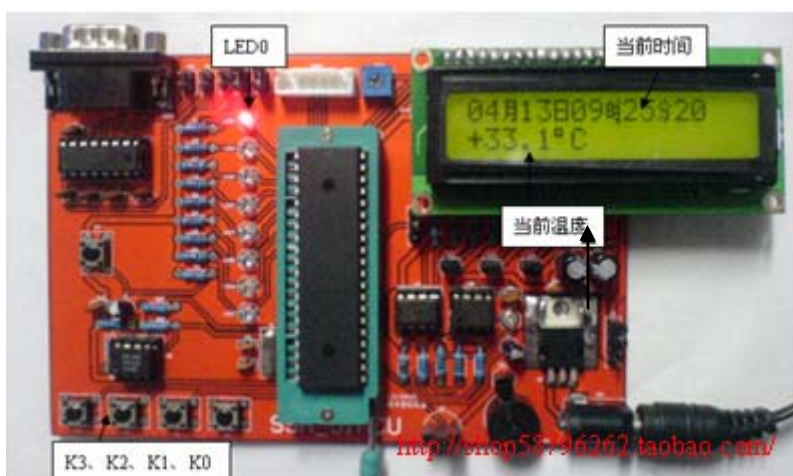
当正确调入烧写文件之后，我们可以点击 1-14 界面的“自动”将文件烧写到 AT89S52 单片机中去，当成功把文件烧写到单片机之后，会出来如图 1-17 的界面，表示文件烧写成功。



图 1-17

(4)、脱机运行

最后将下载线拔出，将 JP2、JP3 接上，把 J4 接到 LCDP 的一边，而 J7 接到 LCDE 的一边。最后通上电源。可以看到 LCD 液晶显示器显示时间与当前的温度，而 LED0 在按每秒一次不停地闪动。而且我们还可以按 K0、K1、K2、K3 等四个按键进行不同功能的设置。关于其操作，读者可以查看本教程的最一页，如下（实图 1）就是程序运行的最终效果图。



程序最终运行的后果图（实图 1）

是不是觉得要实现这个功能很高难度啊！其实你这样想就错了。此程序也是本教程的最后一个综合动手实验，当大家完整地学完这个教程之后，我相信你也能很轻松地写出来的。

第二章 详解 KEIL C51 软件的使用

单片机要运行，就必须将程序代码下载到程序存储器内部，但是在写进单片机之前要先将你写的程序转换成*.hex 或*.bin 的文件。不同系列的单片机都有不同的软件对其进行编译，而 keil Cx51

是德国开发的一个专为 51 系列单片机提供的软件开发平台，基本上现在的所有 51 系列内核的单片机都是使用 keil 来调试和编译，在上面编写好程序后，将其编译成*.hex 的十六进制烧写文件。然后再烧写到单片机系统中去。最后单片机就会按照你写程序的逻辑思维在电路上运行。

Keil Cx51 软件使用

下面以 keil Cx51 V7.04 版本为例，介绍如何使用 keil 的集成开发环境。

1 建立工程

一：双击桌面，启动 keil 如图 2-1 所示。

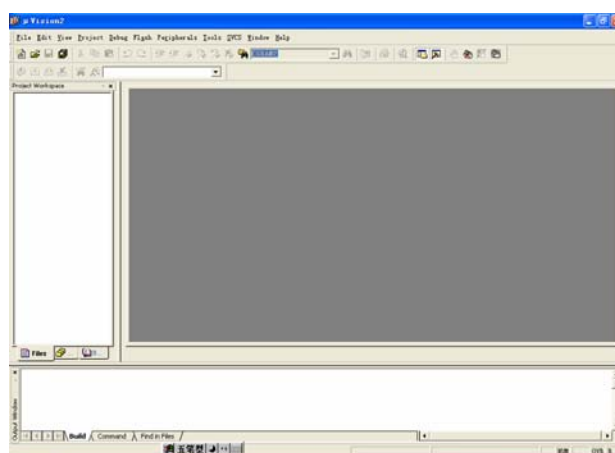


图 2-1

二：新建一个工程

(1) 点击“Project”菜单，选择下拉菜单的“New Project”，会弹出如图 2-2 的窗口，在文件名一栏中填入你想要的工程文件名，文件名是任由你决定的，但是一般是取带有特定意义的为文件名，这样比较容易管理和理解。在这里就取“text”，意为测试的意思。

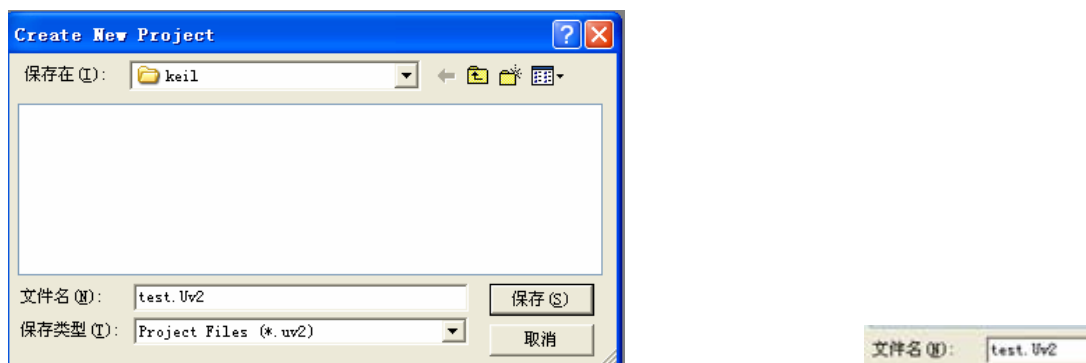


图 2-2

图 2-3

然后点击保存。文件的扩展名为*.uv2（如图 2-3），这是 keil 项目文件扩展名，以后我们直接双击打开这个文件就可以了。

(2) 点击保存之后会弹出如图 2-4 的窗口，要求选择芯片的型号，在这里我们选择“Atmel”内面的 AT89S52 如图 2-5 所示。

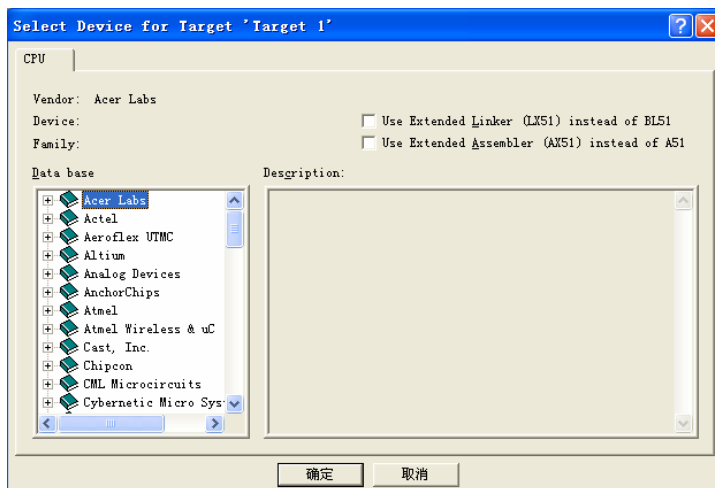


图 2-4

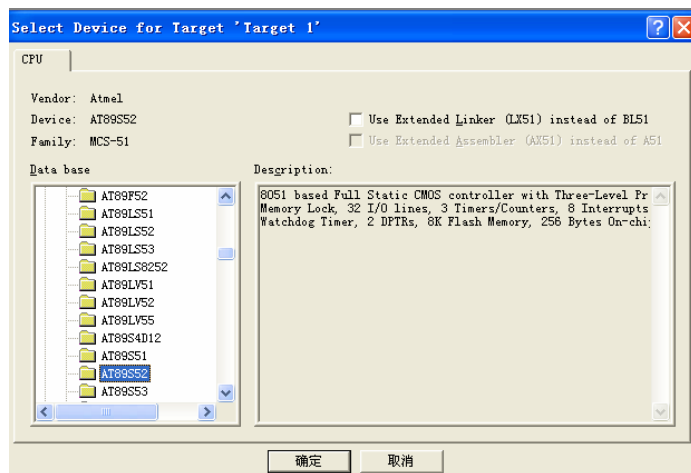


图 2-5

(3) 完成上一步之后点击确认。有可能会出图 2-6 的对话框，其意思是把标准 8051 的启动代码复制到本工程中去，你只需要“N（否）”就可以了。

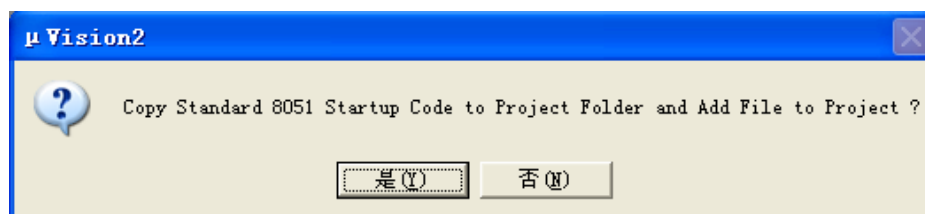


图 2-6

(4) 完成以上步骤之后我们就可以见到的 keil 界面如图 2-7

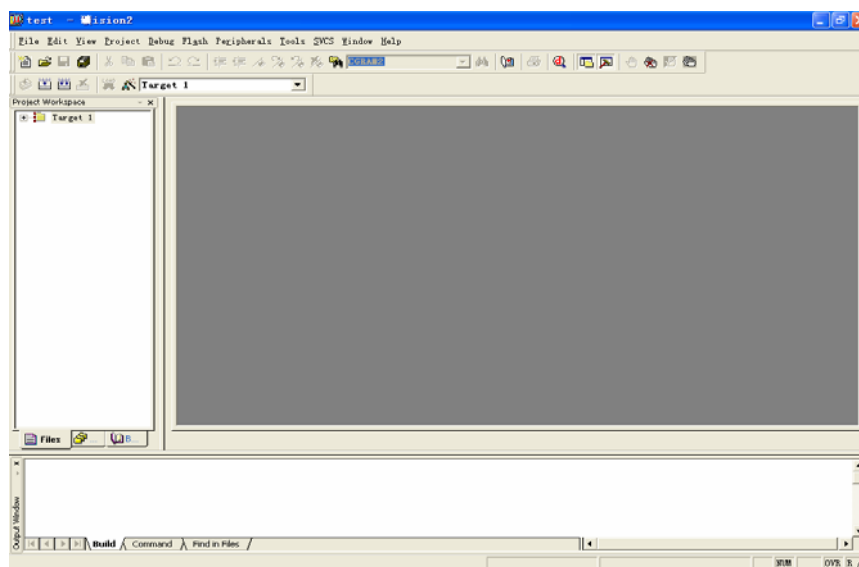


图 2-7

(5) 我们现在来编写第一个程序。点击“File”的下拉菜单中选择“New”的选项如图 3-8 所示。

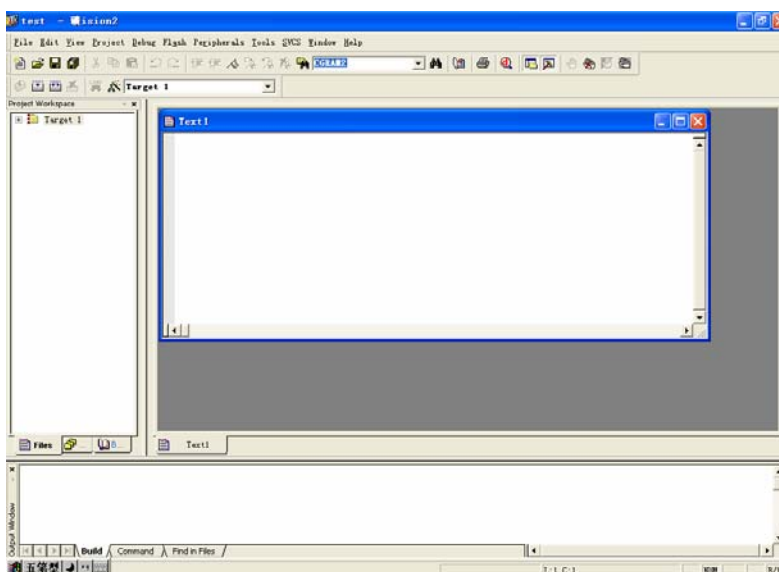


图 3-8

(6) 这时光标会在新建的“Text1”这个文本框中闪烁。其实现在已经可以编写程序了，但是笔者建议大家还是先将这个空白文件保存好之后再编写。点击“File”的下拉菜单中的“Save As”此时会弹出图 2-9 的对话框。在文件名中填入你想要的文件名，但是文件名最好是具有一定的代表意义，这样比较容易管理和理解。这里特别要注意的就是，如果我们用 C 语言来写程序的话，那么文件的扩展名一定要为 .c，但是如果我们用的是汇编语言来写程序，那么文件的扩展名一定要为 .asm，由于我们现在用 C 语言来编写程序，所以这里就取“test.c”，单击“保存”。原来的那个“Text1”的文件已经变成了我们刚才的那个“test.c”文件。见图 2-10 黑色箭头所指。

当你按上面的程序步骤完成之后，还是看见“Text1”没有变这“test.c”文件，那只是你的 keil 没有刷新，你把它最小化，然后再还原就可以了。

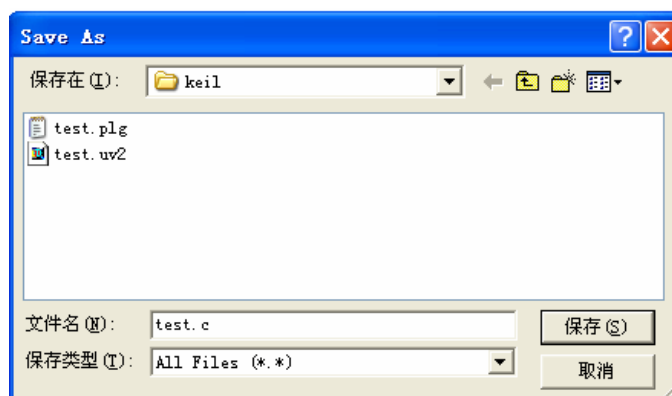


图 2-9

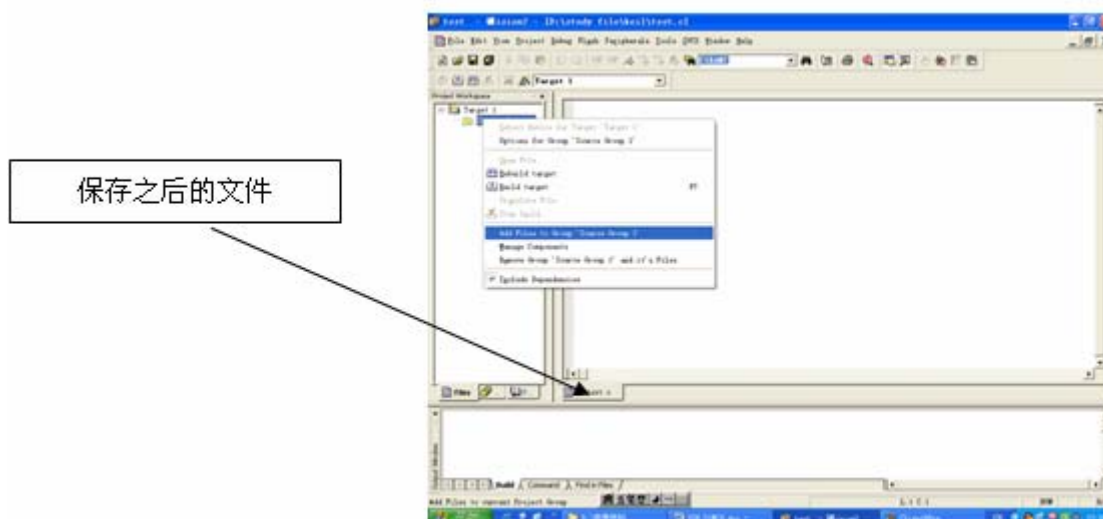


图 2-10

(7) 完成上一步之后，回到了图 2-10 的编辑界面，单击“Target”前面的“+”号，然后在“Source Group 1”上右击一下，选中“Add Files to Group ,Source Group 1”就会弹出图 2-11 的对话框，选择我们刚才建立的那个“test.c”的文件。

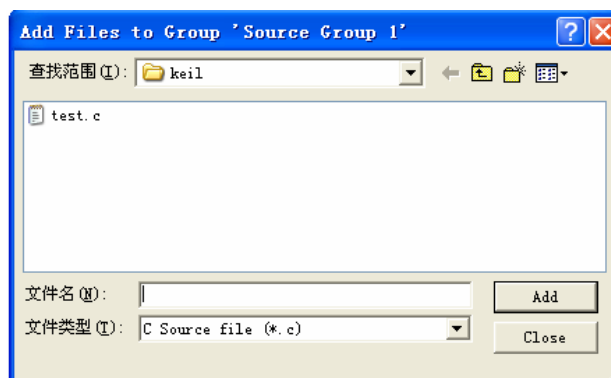


图 2-11

(8) 选择完之后，我们已经发现那刚才在图 2-10 右边的“Source Group 1”下面多了一个“test.c”，如图 2-12 所示。

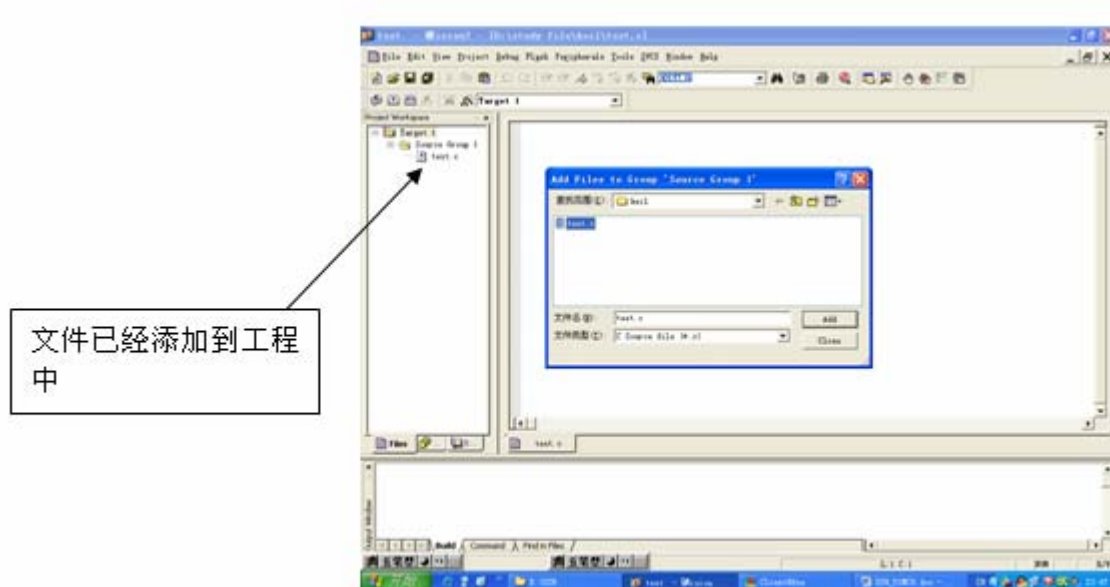


图 2-12

(9)将文件加入“Source Group 1”之后，增加文件的对话框并没有消失。这时其对话框还在等待添加其它的文件，如果你再单击“Add”，就会出现图 2-13 的对话框，其意思是提示用户所选的文件已经在列表中。这时点击“确定”返回到增加对话框，然后点击“Close”返回主界面。再在“Source Group 1”前面的“+”号中点击一下，你就会见到“test.c”这个文件已在工程当中。

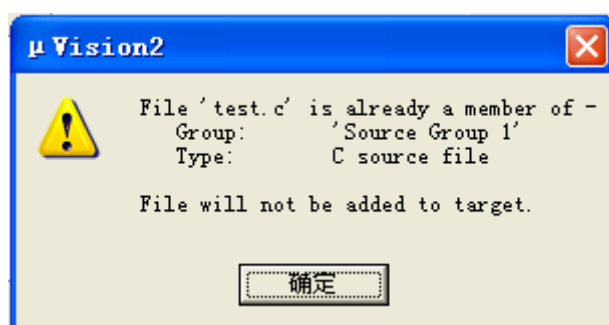


图 2-13

到这里一个完整的项目工程就建立完成了，下面让我们来学习如何调试程序。

```
#include<reg52.h>//包含所用单片机对应的头文件
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
```

```

        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
        {;}
    }
void main(void)
{
    while(1)
    {
        P1=0x00;// 点亮 P1 端口
        delay_ms(500);//把实际参数 500 传给 n，延时 500 毫秒，也就是 0.5 秒
        P1=0xff;// 熄灭 P1 端口
        delay_ms(500);//把实际参数 500 传给 n，延时 500 毫秒，也就是 0.5 秒
    }
}

```

上面是一个简单的 C 语言程序（程序附光盘中），只要将该程序的代码烧写到图 2-14 的电路中去，实验板就会实现“亮，延时 500 毫秒，灭，延时 500 毫秒”这样不断循环闪烁。2-14 是实验板的流水灯电路。对于上面的程序，如果我们是一个初学者，可能还有好多问题是不明白的，是吗？在这里我们暂时不用去管它，在以后节章中我们会详细去研究它，在这里只是让大家有一个初步的认识。

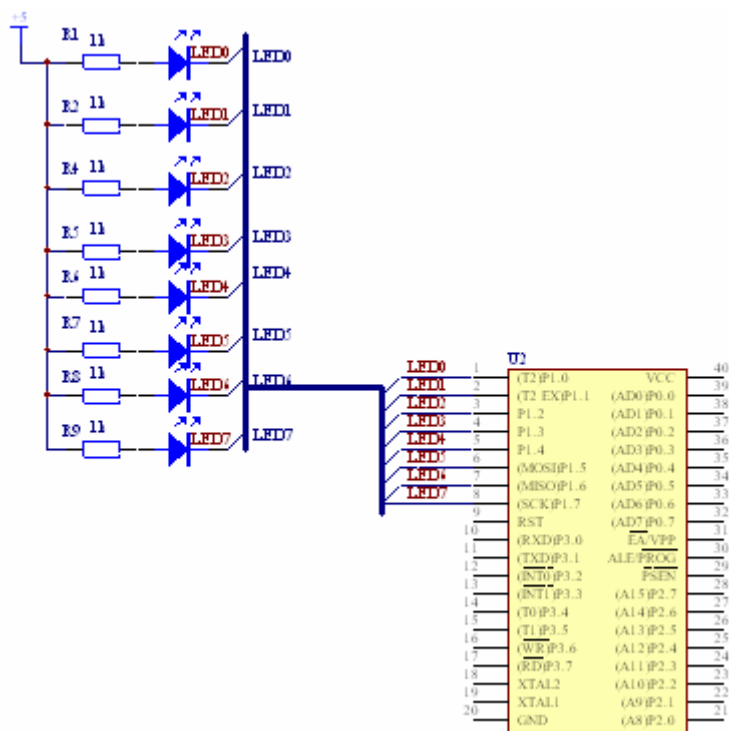


图 2-14

把程序装入 keil 之后的画面如图 2-15。

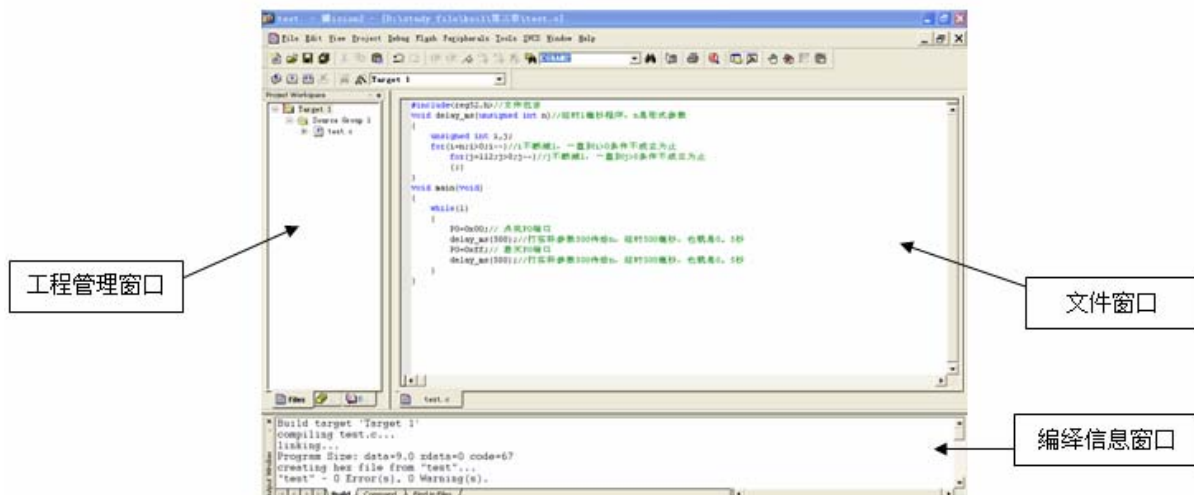



图 2-15

2 设置工程

(1) 在图 2-15 的画面中点击 ，会弹出如图 2-16 的对话框。其中有 10 个选择页。选择“Target”项，也就是图 2-16 的画面。

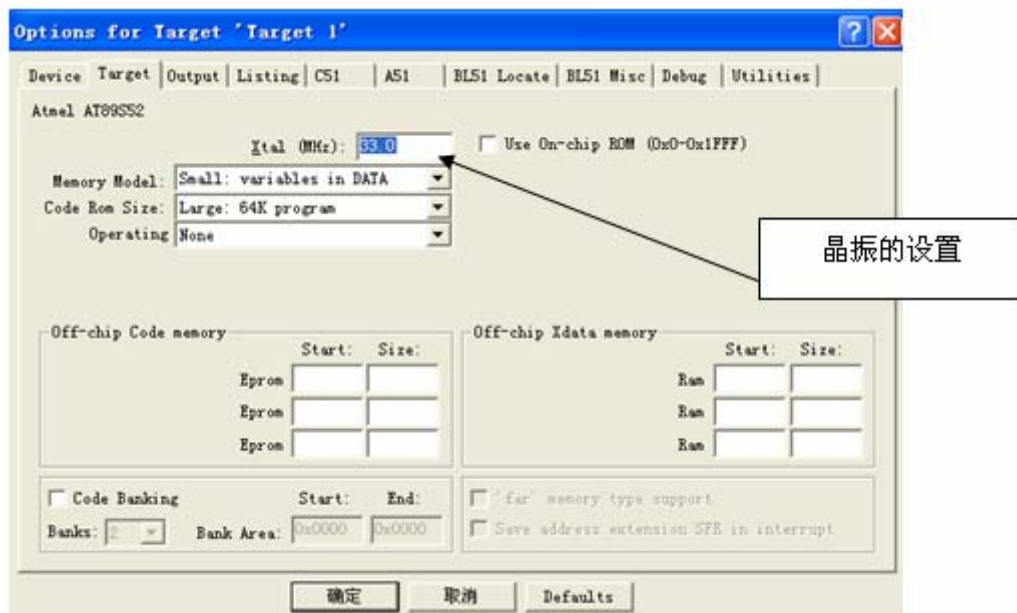


图 2-16

在图 2-16 中，箭头所指的是晶振的频率值，默认是所选单片机最高的可用频率值。该设置值与单片机最终在电路运行中的程序代码是无关的，这只是供我们在软件调试时，显示程序执行的时间（关于如何看时间，这个在后面章节中会详细讲解），一般设置为单片机运行的晶振值（实验板的晶振值为 11.0592MHZ），正确的设置可以显示单片机的实际运行时间，但是如果你在调试程序时不是很关心程序的运行时间，那你也可以不用理会它。

(2) 在图 2-16 的画面中点击“Output”页，会弹出如图 2-17 的对话框。

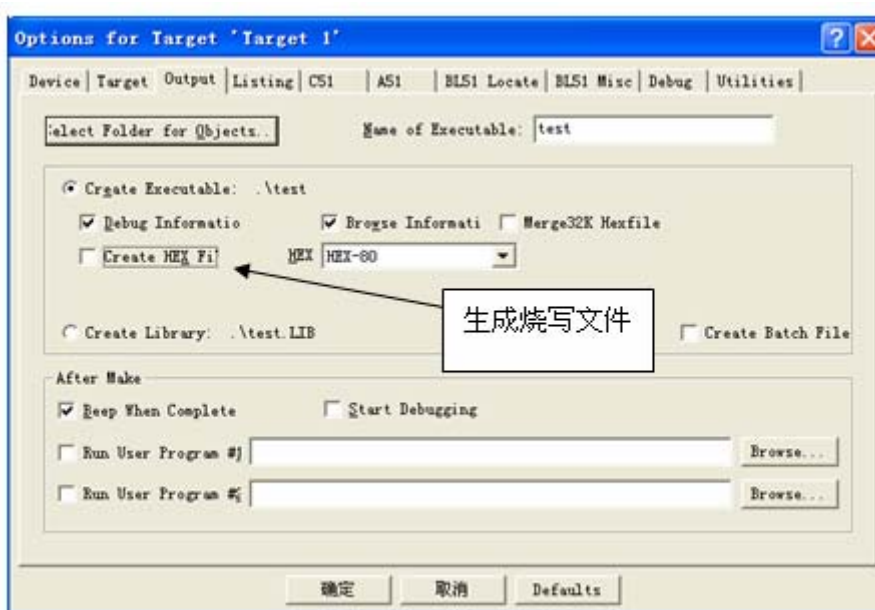


图 2-17

在图 2-17 的对话框中， **Create HEX File** 是生成烧写文件的选项，它的扩展名是“.hex”在默认的情况下是未被选中的。因为我们稍后要将程序烧写到实验板中去，所以在这里选中该项。在默认情况下这个“.hex”文件的存放路径与我们开始时建立项目工程的存放路径相同。只要将这个“.hex”的文件烧写到单片机里面去，其运行的结果与你想象中的一致，那你的电子产品就开发成功了。

(3) 在图 2-17 的画面中点击“Debug”页，会弹出如图 2-18 的对话框。

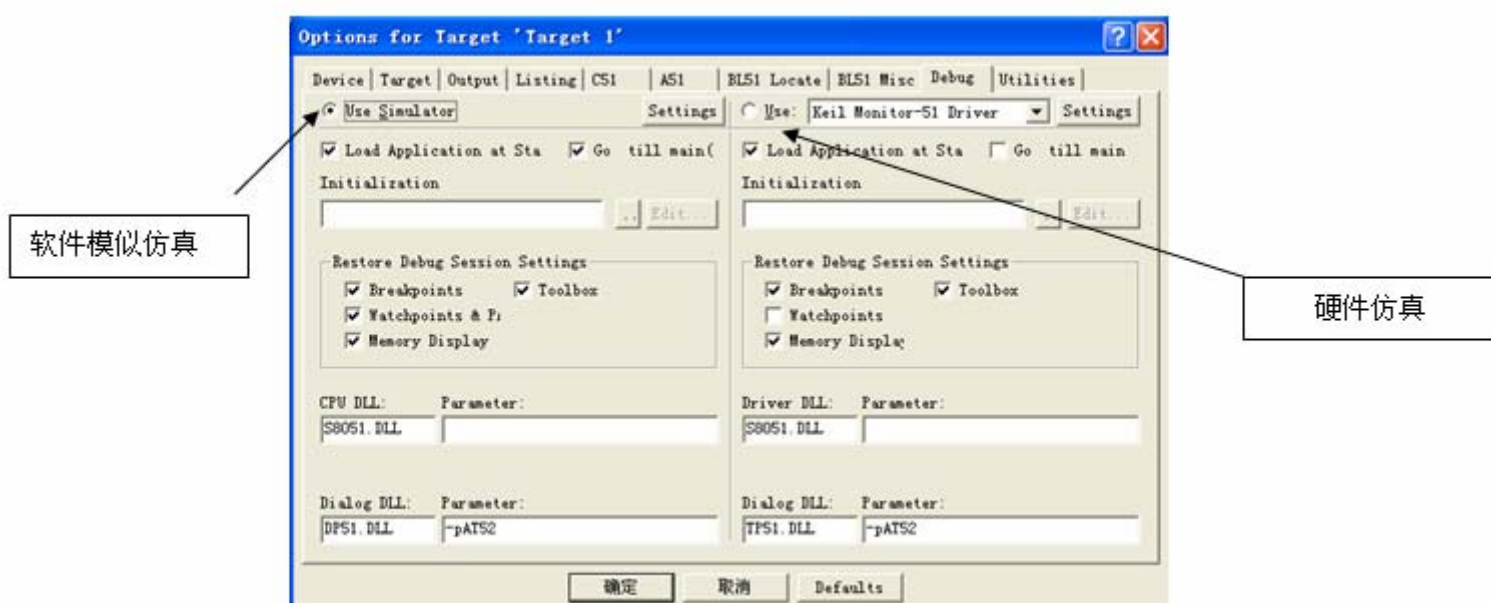


图 2-18


Keil 提供了两种调试方式。一种是“软件模拟仿真”；一种是“硬件仿真”。在软件模拟仿真的模式下完全不需要硬件的支持（如目标板，仿真器，开发板，实验板等），在这种模式下我们可以观察单片机内部的运行情况，如运行时间，寄存器的状态，变量状态等。一般非常简单的程序你就可以这样调试，但是如果较为复杂的程序，这样来调试就较为困难。


还有一种就是硬件仿真，它是利用仿真器与 keil 连接起来，在硬件中看到真实的运行情况，但是硬件仿真有一样是做不到的，那是看不到程序运行时间，这一点也只有软件仿真才能做到。


若果是一个专业的电子工程师，在一个真正项目开发情况下，一般除了看程序运行时间外，其它多般是用硬件仿真的，因为你写的程序是在真实的单片机中运行，而不是在 keil 软件中运行。在软件仿真的情况下有好多实质性的问题是不能发现的。keil 软件的设置我们只需要了解这么多就可以了，因为 keil 的绝大部分设置都是默认的，无需去设置。


3 编译与连接程序


在图 2-15 中我们已经把程序装 keil 中，现在我们就来进行编译与连接。在图 2-15 界面的左上角，

有三个按钮 。现在分别来讲解一下它们不同的作用。


 只用于编译当前文件中的语法错误，不对文件进行链接。

 编译链接按钮，用于对当前工程进行链接，如果当前工程已修改，keil 会对该文件进行编译。然后再链接以产生目标代码。

 重新编译按钮，每点击一次均会再编译链接一次，不管程序是否有改动，保证生产的目标代码是最新的！

 停止按钮，只要点击了上面三个的任意一个，停止按钮才会生效。

以上的那些按钮都可以在“project”菜单中找到。

在我们刚才建立的那个工程当中，按上面分析的那三个按钮意思，我们就按  个按钮吧！在“Build”窗口出现了一段如图 2-19 的编译信息。

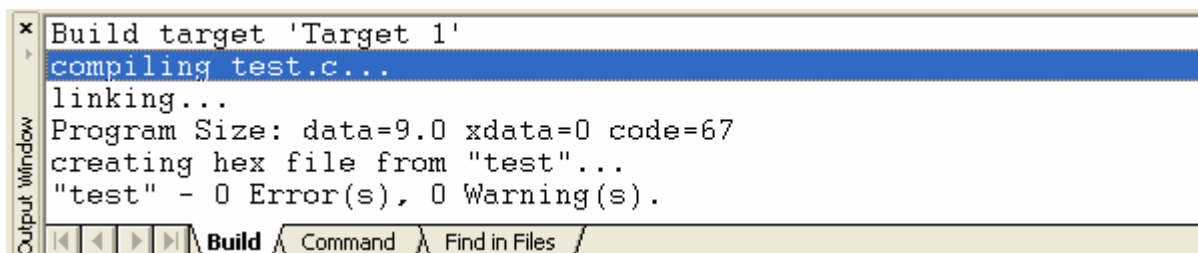



图 2-19

```
creating hex file from "test"...  
"test" - 0 Error(s), 0 Warning(s).
```

图 3-20

在图 2-19 的信息中，大家有没有留意到出现在图 2-20 的那些信息语句啊！它的大概意思是：“创建了 hex 文件来自 test 这个工程，在 test 这个工程中有 0 个错误和 0 个警告”。但是大家要知道，要想产生 hex 文件，我们一定要在图 3-17 的对话框中设置。大家不防试一下，如果在图 3-17 的“Output”选项页中不把  选中的，会是怎么样的呢！还有，如果程序中有语法错误，也可以在“Build”窗口中显示出来。现在我们就来做一个试验。我们刻意地把“P0=0x00;”这句程序语句改为“Po=0x00;”，就是把数字的“0”改为英文字母的“o”，按一下编译按钮，就会出现图 2-21 的信息。

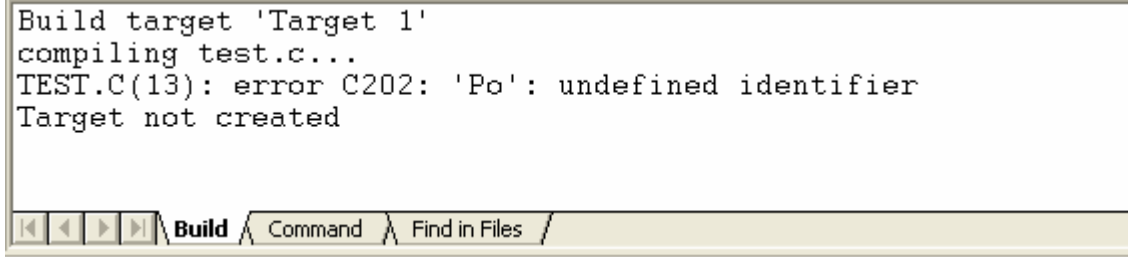



图 2-21

它的大概意思是“test.c 这个文件中的‘Po’为未明确的标识符，目标代码不能创建”，只要你用鼠标双击一下该行错误的信息，在“test.c”文件中就会出现一个蓝色的箭头，说明该行出现错误。但是如果文件中某程序语句后面少一个“;”号，双击图 2-21 中的错误信息栏，箭头会指向缺少“;”号的程序句的下一行，以上的两种错误在编写程序中也是经常出现的，希望大家注意。

4 调试程序

在编写程序的时候一般会出错两种错误，一种为程序的语法错误，而另一种是程序逻辑错误，当完成到出现图 2-20 信息栏的时候，只是说明你编写的程序没有语法错误，但是至于程序当中的逻辑错误是不能发现的，那就得靠我们对程序的反复调试来找出错误所在。调试是项目开发中一个非常重要的环节。下面我们重点来学习如何调试吧！当通过之前所讲的各个步骤出现图 2-20 的信息栏之后，点击  按钮当前的界面会发生变化，如图 2-22 所示。

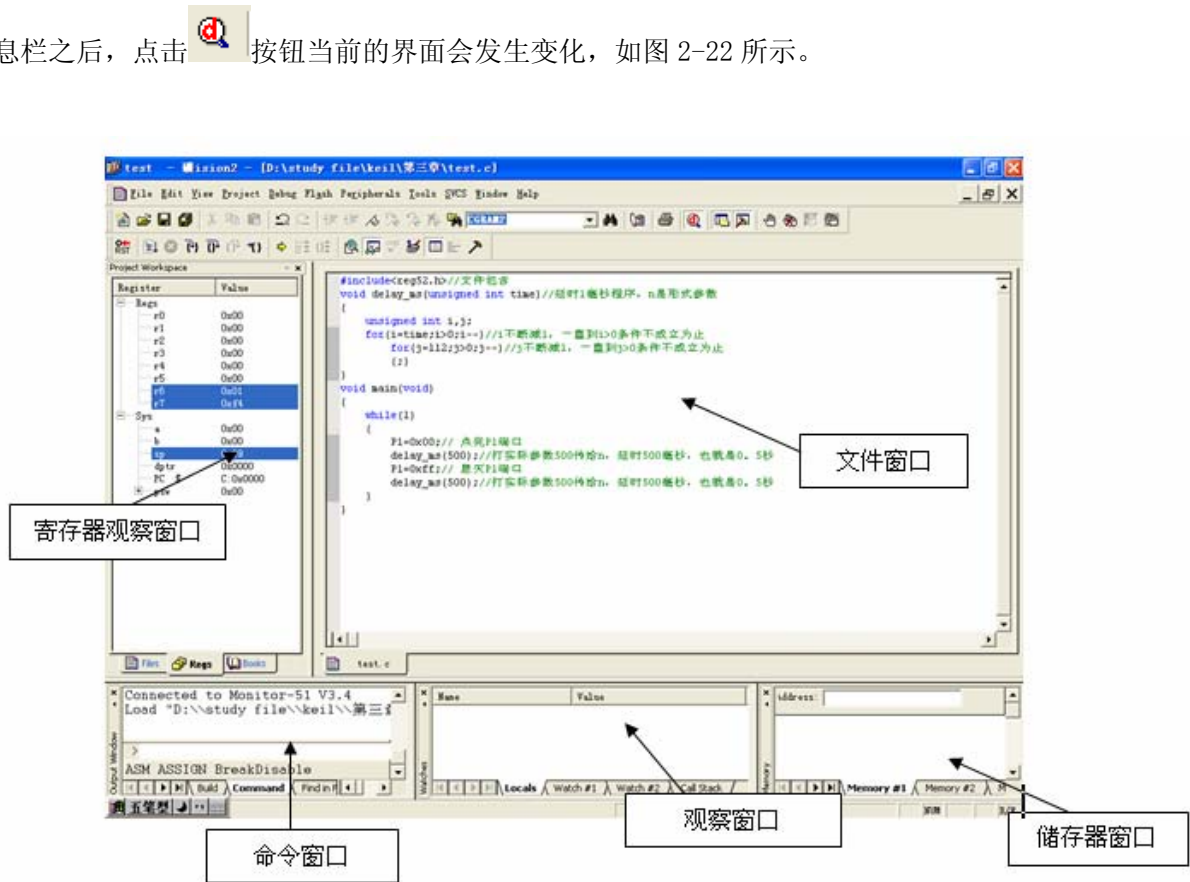


图 2-22



图 2-23

图 2-23 为 keil 的调试工具按钮，意思从左到右为，复位、全速运行、暂停、单步执行、过程单步执行、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。

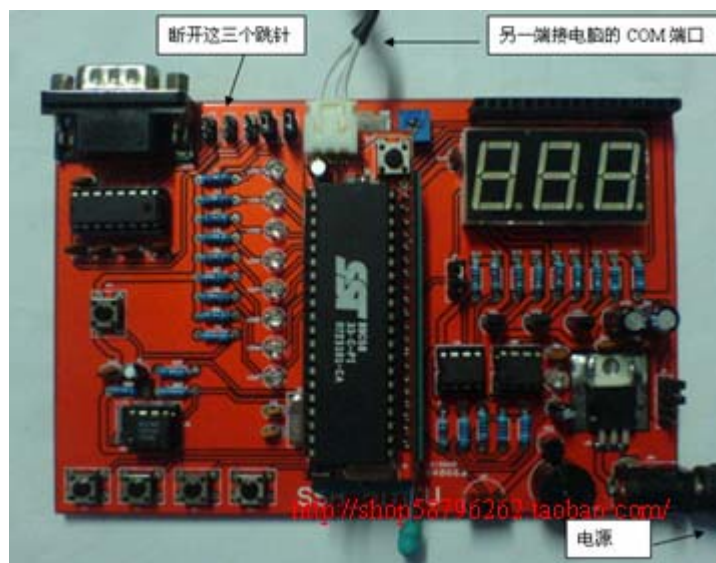
特别注意：所为的软件仿真是在没有真实硬件支持的前况下进行程序的调试，不过软件的模拟仿真与真实的硬件仿真是不完全相同的，其中最明显的就是时序，最为具体的表现就是程序的执行时间与所使用的计算机的配置性能有较大的关系，配置性能越高，执行的速度越快。

首先我们来区分单步执行与全速运行有什么不同，这绝对不能混淆的。第 1：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，中间是不会停止的，在这样的情况下程序执行的速度非常之快，即可以看到整个程序执行的最终效果是正确还是错误的，但是如果程序当中出现了逻辑性错误，我们就很难判断到底是那条程序语句出错。第 2：单步执行是指每次执行一条程序语句，执行完该条程序语句之后就会停止下来，等待命令再执行下一条程序语句，在这时我们就可以通过观察结果与程序员的逻辑思维是否相一致，如果不一致我们就分析错误到底出现在那里。下面我们利用实验板来试验图 2-23 中几个主要的按钮功能。

动手实验：（1）

实验目的：熟悉全速执行、单步执行、过程单步、断点调试、学习如何分析 keil 软件窗口使用。


注意：在利用 SST_51 仿真器进行程序调试时，一定要将实板的 JP4、JP5、JP6 跳针断开。否则有可能会因为两个串口芯片同时工作，导致仿真器不能与 Keil 正常通信。正确的操作方法如（实图 2）所示。



仿真器与实验板的正确接法（实图 2）

- 步骤：1. 把 SST_51 仿真器装上到 SSH_51MCU 实验板上，要特别注意引脚不能插反，否则会烧毁仿真器。而另一端的 DB9 孔插座与计算机的 COM 串口 DB9 针插座相接。
2. 将 SSH_51MCU 实验板上电源，些时会看见 LED9 在不停闪动，这样表示实验板通电工作正常。
3. 按照上面的方法新建一个工程，打开光盘中的第二章文件夹，“text.c”文件。
4. 在图 2-18 的界面中设置为硬件仿真。点击旁边的 **Settings** 按钮会弹出如图 2-25 的对

话框，因为 51 仿真器的晶振为 11.0592Mkz，所以波特率 **Baudrate: 38400** 选择 38400。

5. 点击  将实验板与 keil 连接。如果出现 2-24 的对话框，则说明实验板未能与电脑正常连接，这时应该检查仿真器的连接是否与你的设置相一致，还得检查 51 仿真器有没有完全复位。（如果没有完全复位，用手按下仿真器的复位按键片刻即可）

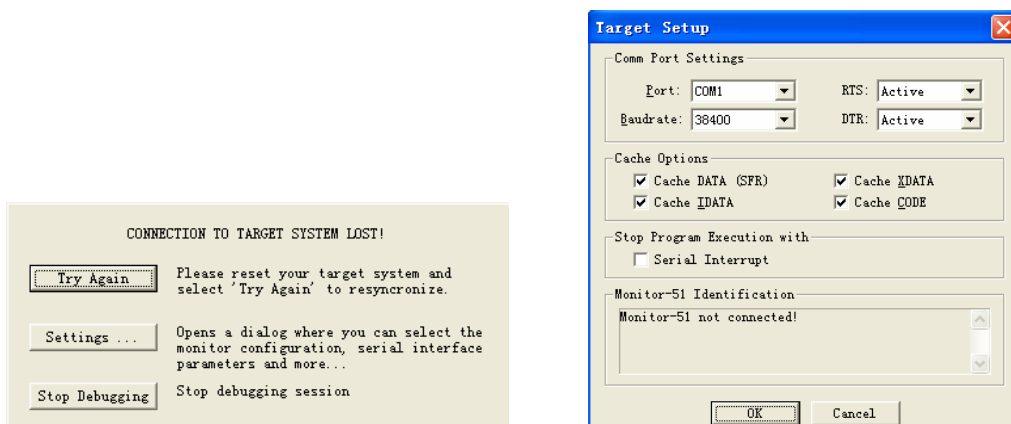


图 2-24

图 2-25

6. 通信正确后出来如图 2-26 的界面。

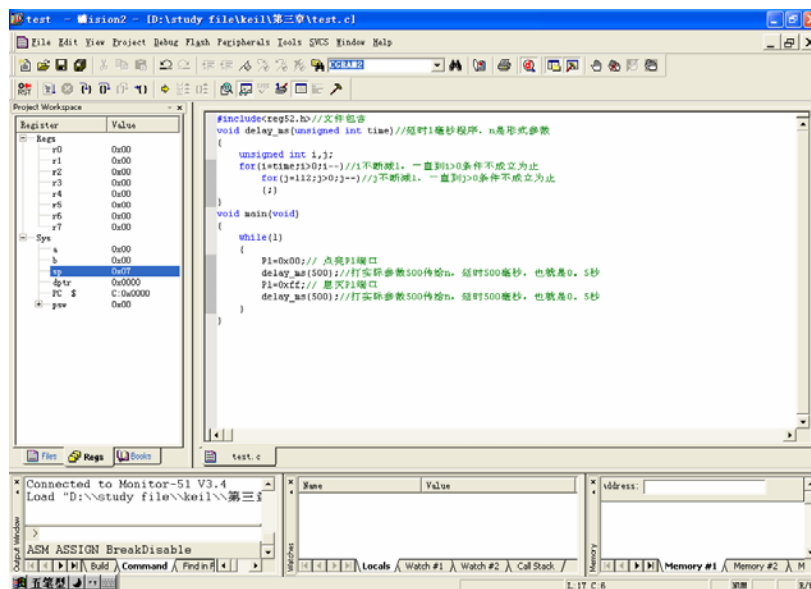


图 2-26

实验单步调试:






点击  出现一个黄色的箭头指向程序的第一句“P1=0x00;”，（注意：出现黄色箭头的快慢与电脑的配置有关，配置越高出现得越快，配置越低出现得越慢），再点击一下，实验板上单片机 P1 口的 8 只发光二极管点亮。这时 keil 等待你再一次输入命令，再点击一下箭头指向 delay_ms 延时函数的第一句“for(i=time;i>0;i--)”这时可以看到观察窗口如图 2-27 所示。你可以点击鼠标右键选择十进制数显示或十六进制数显示的切换。再点击一下  可以见到变量“j”的值减小 1，即执行该箭头所指程序行，然后箭头指向下一行，当点击 112 下的时候，“j”变为 0，变量“i”就会减 1，当变量“i”由 500 变为 0 的时候就会退出 delay_ms 函数。（可能有些读者会问，为什么要延时啊，其实好简单，因为单片机的运行速度太快了，是以微秒来计算的，所以必需加入适当延时人的肉眼才能看见。）









图 2-27

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比较困难的，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“`for(i=time;i>0;i--)`”的值不断减1，这一行程序执行56000多次来达到延时目的，如果用按 56000多次方法来执行完该程序行，这显然是不可取的。下面我们介绍另外一种比较灵的方法。


实验过程单步调试：

第一种方法：首先，当箭头执行到“`delay_ms(500);`”的时候，P1端口的8只发光二极管已经点亮，这时就不要按 单步执行按钮，而改按 过程单步执行按钮，箭头不是进入延时函数而是跳过延时函数指向“`P1=0xff;`”这一语句，用同样的方法可以不停地执行主程序的语句，而实验板的8只发光二极管也在不停地跟随闪动。

第二种方法，把光标定位于你想要执行到的程序行，然后用菜单（运行到当前行），即可全速执行完黄色箭头与光标之间的程序行。另外，在进入该子程序后，使用按钮（执行完当前子程序）命令，即可全速执行完调试光标所在的子函数。我们现在就用第二种方法调试一下程序，在第二个“`delay_ms(500);`”延时，用光标点击一下，把光标定位于该行，按（运行到当前行），这时实验板的8只发光二极管会点亮然后延时500毫秒再熄灭。

第三种方法，在开始调试的时候，按（过程单步执行）而不是（单步执行），程序也将单步执行，不同的是，执行到“`delay_ms(500);`”行时，按下键，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“`P1=0xff;`”。按这种方法进行调试实验板的发光二极管会跟随你的过程单步指令一闪一闪，灵活应用以上的几种方法，可以大大提高查错的效率。

实验全速执行调试：

在一开始的时候，就按（全速执行），实验板的8只发光二极管会一亮一灭不停地循环闪烁，这是你写程序的最终结果。


实验断点调试：

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，例如：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。这些条件往往是难以预



先设定的，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调试光标立即停止，可在此时观察有关变量值，以确定问题所在。下面分别讲解一下关于设置断点几个按钮。


 设置或移除断点，同时也可以鼠标在该行双击实现同样的功能；

 开启或暂停光标所在行的断点功能；

 暂停所有断点；

 清除所有断点的设置。

其实断点调试与 （运行到当前行）调试是一样的，只不过  运行到当前行之后，如果下次还想同样运行到当前行，那就要用光标从新点击。但是断点调试就不一样，当设置了一个断点之后，它会长时间存在的，除非程序员刻意把其移除。下面我们来试验一下断点是如何使用的。



用光标分别在“P1=0xff;”和“}”双击一下，出现两个红色的断点标志，然后再按  全按运行，出现了如图 2-28 的画面，你就会分别看到黄色光标在设置有断点的地方停下来，而实验板的发光二极管会跟随程序语句做起相应的动作。

```
#include<reg52.h>//文件包含
void delay_ms(unsigned int time)//延时1毫秒程序, n是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i不断减1, 一直到i>0条件不成立为止
        for(j=112;j>0;j--)//j不断减1, 一直到j>0条件不成立为止
            {;}
}
void main(void)
{
    while(1)
    {
        P1=0x00;// 点亮P1端口
        delay_ms(500);//打实际参数500传给n, 延时500毫秒, 也就是0.5秒
        P1=0xff;// 熄灭P1端口
        delay_ms(500);//打实际参数500传给n, 延时500毫秒, 也就是0.5秒
    }
}
```

图 2-28

下面我们的来介绍常用的观察窗口。图 2-29 为寄存器观察窗口，其中 r0~r7 是通用寄存器，而 a, b, sp, sp_max, dptr, pc, 是系统寄存器，这些寄存器在使用 C 语言编程时是比较少用到的，只是 sec 这一项比较常用到，它在软件仿真的时候可以显示程序执行的时间，下面我的来试验一下。

实验观察程序的执行时间：

（注意：这时应该选取软件仿真，同时应把图 2-16 的对话框中晶振设置与实验板的晶振相同），下面我们来观察一下“text.c”这个程序的“delay_ms(500);”延时函数是否延时 500 毫秒。展开“Regs”选项页，如图 2-29 所示。它记录了当前程序所执行流逝的时间。点击复位按钮，使程序复位，此时“sec”的值归零，按  单步执行，当执行到“delay_ms(500);”的时候，记录下执行到当前的时间为 0.00042426 秒，然后再按  跨过这个函数，些时程序的执行时间为 0.49360569 秒，两个数相减为 0.49318143 秒，约为 500 毫秒，所以程序的延时大致是正确的。大家要记住，在使用这一功能的时候一定要保证已经正确设置晶振的频率。

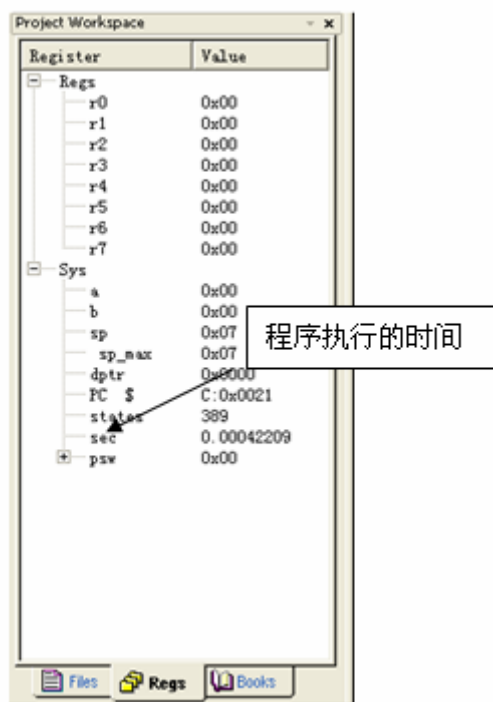


图 2-29

自我练习：

通过前面所讲的方法，自己建立一个工程，命名为“mytest”，同时也是打开“test.c”这个文件，利用软件仿真进行调试。

（详细练习教程以视频的形式附光盘中）

第三章 KEIL C51 C 语言

Keil c51 是专为 51 内核单片机设计的一个 C 语言编译器。Keil c51 支持 ANSI 标准的 C 语言。在标准 C 语言的基础上还扩展了一些关键字，使语法更加简洁，紧凑。在还没有开始讨论 Keil C51 之前，我们首先来讲解一下关于 C 语言中的标准输入输出函数和单片机的串行口通信，因为在本章内容中有大部分的动手实验和自我练习都是利用这两个函数与串行通信的配合来进行，以便提高读者的动手能力和理解。

(1) “printf” 函数（格式输出函数）

其语法格式为：`printf(格式控制, 输出列表);`

例如 1:

```
printf(“%c,%d”, x, y);
```

在上面的这个函数当中，格式控制就是“%c,%d”，它的作用是将输出的数据转换为指定的格式输出，格式的说明必须是以“%”字符开始，其中%c 表示以字符的格式输出；%d 则表示以十进制数的格式输出。而 x, y 就是输出列表，输出列表可以是表达式，整个函数的意思就是把输出列表的变量按格式控制中所指定的格式进行输出到计算机的终端设备（如串口调试器）。

例如 2:

```
void main(void)
{
    int x,y;
    x=520;
    y=130;
    printf(“%d”,x+y); //把 x 与 y 的值相加再按照十进制数的格式输出。
}
```

运行结果输出：650

(2) “scanf” 函数（格式输入函数）

其语法格式为：`scanf(格式控制, 地址列表);`

例如 3:

```
scanf(“%d%d%d”, &i, &j, &k);
```

上面的函数当中，“%d%d%d”表示按十进制数的格式输入数据，在“%d%d%d”之间不能有“，”逗号分隔，每输入一个数以回车键作为结束。&i、&j、&k 前面的“&”为取地址运算符，&i 表达 i 在内存中的地址。那么上面的函数意思为：在键盘中按照十进制数的格式输入 3 个数，然后按照 i、j、k 在内存中的地址将 i、j、k 的值存进去。

例如 4:

```
void main(void)
{
    int i, j, k;
    scanf("%d%d%d", &i, &j, &k); //从键盘中按十进制数输入 3 个数，存放在 i, j, k 的内存地
    //址当中.
    printf("%d, %d, %d", i, j, k); //把 i, j, k 的值按照十进制数的格式输出。
}
```

运行结果:

5↵77↵33↵

5, 77, 33

大家要注意: printf 函数与 scanf 函数是 C 语言当中的库函数, 在使用之前必须以 #include<stdio.h>的形式将头文件包含到当前文件当中。

最后我们还来简单介绍一下串行通信, 串行通信是指单片机与电脑之间的通信, 要实现通信就必须依靠波特率发生器产生波特率来为持时序, 从而实现数据的传送。

```
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
```

上面就是一个串行通信模块初始化函数, 只要初始化了串行通信模块, 单片机在串行通信时就会自动产生波特率来实现数据的传送。关于单片机的串行通信模块在第四章会有详细的讲解, 在这里大家只要大概了解一下就可以了。下面我们来详细讲解 51 单片机 C 语言。

第 1 节. C 语言的基本架构

1: 标准 C 语言的关键字 (表 3-1)

关键字	用途	说明
char	声明字符型变量或函数	

double	声明双精度变量或函数		
void	声明函数无返回值或无参数，声明无类型指针		
unsigned	声明无符号类型变量或函数		
struct	声明结构体变量或函数		
union	声明联合数据类型		
signed	声明有符号类型变量或函数		
short	声明短整型变量或函数		
long	声明长整型变量或函数		
int	声明整型变量或函数		
float	声明浮点型变量或函数		
enum	声明枚举类型		
sizeof	计算数据类型长度		
volatile	说明变量在程序执行中可被隐含地改变		
typedef	重新进行数据类型定义		
const	声明常量		
Static	声明静态变量		存储种类的说明
register	声明寄存器变量		
extern	声明外部变量		
auto	声明自动变量		
return	函数返回语句 返回一个值		程序语句
case	开关语句分支		
default	Switch 语句的失败选择项		
switch	开关语句		
goto	无条件跳转语句		
else	构成 if-----else 选择语句		
if	if 条件语句		
continue	结束当前循环，开始下一轮循环		
break	跳出当前循环体		
while	构成 while 和 do----while 循环语句		
do	循环语句的循环体		
for	for 循环语句		

Keil C 语言扩展的关键字 (3-2)

关键字	用途	说明
bit	声明位变量或位类型函数	声明位变量
sbit	声明可寻址变量	
sft	声明特殊功能寄存器	声明特殊功能寄存器
sfr16	声明 16 位特殊功能寄存器	
data	直接寻址内部数据存储器	存储种类的说明
bdata	可位寻址内部数据存储器	
idata	间接寻址内部数据存储器	
pdata	分页寻址内部数据存储器	

xdata	外部数据存储器	
code	程序存储器	
interrupt	定义一个中断函数	中断函数说明
reentrant	定义再入一个函数(递归)	再入一个函数说明
using	定义芯片的工作寄存器	寄存器组定义

注意：上面所提到的“sbit”与“bit”是两个不同的概念，“sbit”是定义位操作寄存器，而“bit”是 keil c51 中的一个数据类型，在使用中一定要注意，不能混淆。

第 2 节. C 语言规则

下面的程序是我们在第二章调试通过的程序，现在我们来分析这个程序的构成。（程序付光盘中）

```
#include<reg52.h>//包含所用单片机对应的头文件
void delay_ms(unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
    while(1)
    {
        P1=0x00;// 点亮 P1 端口
        delay_ms(500);//把实际参数 500 传给 n, 延时 500 毫秒, 也就是 0.5 秒
        P1=0xff;// 熄灭 P1 端口
        delay_ms(500);//把实际参数 500 传给 n, 延时 500 毫秒, 也就是 0.5 秒
    }
}
```

1. 标识符是用来定义源程序当中的某个对象名称，例如语句，数据类型，函数，变量，数组等。C 语言标识符只承认由字母，数字，下划线，而且还一定要以字母或者下划线开始；
2. 标识符的长度由系统决定，但至少前 8 个字符有效，字符超出的部分由系统省略；
3. 上面表 3-1 与表 3-2 所说的 C 语言关键字由系统保留不能用作标识符。
4. 声名标识符的时候，最好是选取具有一定代表意义的明词，例如延时 1 毫秒的延时函数就可以定义为“`delay_ms`”，尽量不要取名“aa” “bb” “tt” 等等没有特定意义的标识符，这样虽然没有违反 C 语言的规则，但是在程序里就很容易混淆，我们在编写程序的时候一定要养成良好的习惯。

5. C 语言区分大小写，例如定义一个延时函数的形式参数 `time`，但是如果程序当中再出现一个由大写字母定义的标识符 `TIME`，那么它们在程序当中是两个不同的标识符，是没有冲突的。
6. C 语言程序当中有且只有一个 `main` 函数，函数是构成 C 语言程序的单位。
7. 一个 C 语言程序，无论 `main` 函数的物理位置在那里，总是从 `main` 函数开始执行。
8. 书写 C 语言程序的时候，可以一行写多条程序，也可以一行只写一条程序，总的来讲，C 语言的书写是自由的，但是我们最好还是按照一定的格式来书写，这样在调试程序时比较容易，也比较容易读懂程序。
9. 每句程序语句后面一定要加分号，分号是 C 语言结构的一部分，如果缺少了就会语法出错。
10. 函数分两部分组成，在上面的程序当中 `void delay_ms(unsigned int time)` 是一个函数的首部，即函数头，而在括号里面的 `unsigned int` 是数据类型，它声明了 `time` 是一个无符号整型的变量。而下面的：

```
{  
    unsigned int i, j;  
    for(i=time; i>0; i--) // i 不断减 1，一直到 i>0 条件不成立为止  
        for(j=112; j>0; j--) // j 不断减 1，一直到 j>0 条件不成立为止  
            {;}  
}
```

是函数体，函数体里的语句一定要用“{ }”花括号括起来。这是函数的语法结构。

11. 注释，在程序中添加注释是为了能更加容易读懂和理解程序，keil 支持两种风格的注释方法“//”和“/*-----*/”。 “//”的意思是在其后面的全部引导为注释，而“/*-----*/”的意思是在“/*”开始，一直到遇到“*/”为止，在其里面的内容都被认为是注释。大家要注意，在编写程序的时候如果不小心删除了一个“*/”那么从“/*”开始的内容就全部被认为是注释，这一点大家要小心。

12. 文件包含，文件包含不是必要的。`#include<reg52.h>`就是一种文件包含，但是为了将程序更加人性化，模块化，所以引入了文件包含，它的意思是将扩展名为“.h”的 `reg52` 这个文件包含到当前文件当中。其中“`reg`”是标准的意思，“.h”是 `head` 的缩写为头文件的意思。下面就是 `reg52.h` 这个头文件。把鼠标放在 `#include<reg52.h>` 点击右键把它打开。(如图 3-1)

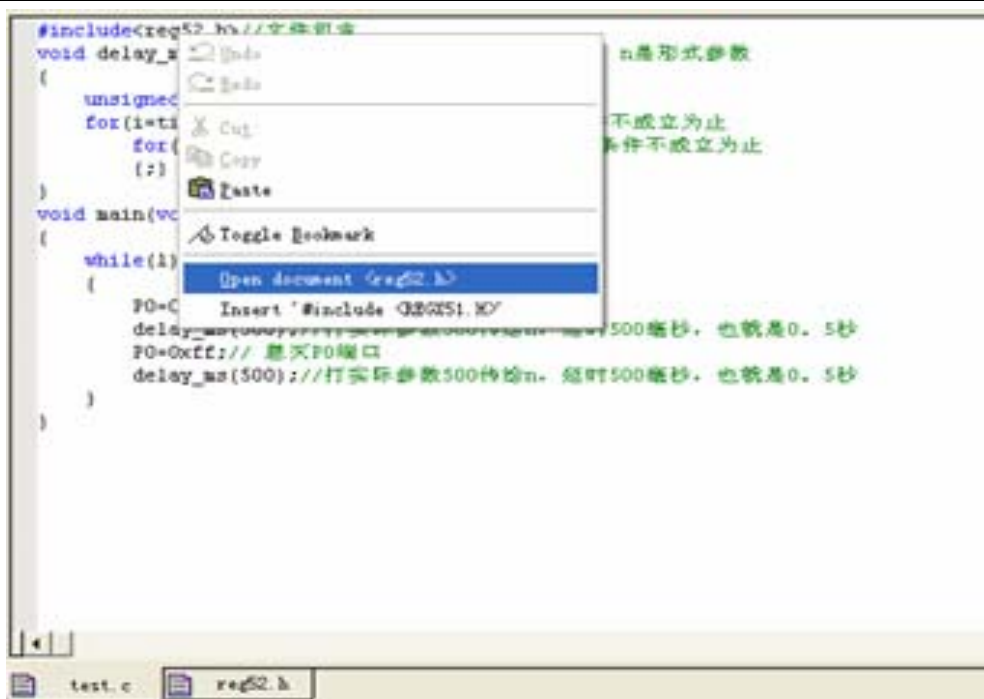


图 3-1

以下是打开后的头文件

```
/*-----*/
REG52.H

Header file for generic 80C52 and 80C32 microcontroller.
Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
All rights reserved.
-----*/

#ifndef __REG52_H__
#define __REG52_H__

/* BYTE Registers */
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;

/* 8052 Extensions */
sfr T2CON = 0xC8;
sfr RCAP2L = 0xCA;
sfr RCAP2H = 0xCB;
sfr TL2 = 0xCC;
sfr TH2 = 0xCD;

/* BIT Registers */
/* PSW */
sbit CY = PSW^7;
sbit AC = PSW^6;
sbit F0 = PSW^5;
sbit RS1 = PSW^4;
sbit RS0 = PSW^3;
sbit OV = PSW^2;
sbit P = PSW^0; //8052 only

/* TCON */
sbit TF1 = TCON^7;
sbit TR1 = TCON^6;
sbit TF0 = TCON^5;
sbit TR0 = TCON^4;
sbit IE1 = TCON^3;
sbit IT1 = TCON^2;
sbit IE0 = TCON^1;
sbit IT0 = TCON^0;

/* IE */
sbit EA = IE^7;
sbit ET2 = IE^5; //8052 only
sbit ES = IE^4;
sbit ET1 = IE^3;
```



```
sbit EX1    = IE^2;
sbit ET0    = IE^1;
sbit EX0    = IE^0;

/* IP */
sbit PT2    = IP^5;
sbit PS     = IP^4;
sbit PT1    = IP^3;
sbit PX1    = IP^2;
sbit PT0    = IP^1;
sbit PX0    = IP^0;

/* P3 */
sbit RD     = P3^7;
sbit WR     = P3^6;
sbit T1     = P3^5;
sbit T0     = P3^4;
sbit INT1   = P3^3;
sbit INT0   = P3^2;
sbit TXD    = P3^1;
sbit RXD    = P3^0;

/* SCON */
sbit SM0    = SCON^7;
sbit SM1    = SCON^6;
sbit SM2    = SCON^5;
sbit REN    = SCON^4;
sbit TB8    = SCON^3;
sbit RB8    = SCON^2;
sbit TI     = SCON^1;
sbit RI     = SCON^0;

/* P1 */
sbit T2EX   = P1^1; // 8052 only
sbit T2     = P1^0; // 8052 only

/* T2CON */
sbit TF2    = T2CON^7;
sbit EXF2   = T2CON^6;
sbit RCLK   = T2CON^5;
sbit TCLK   = T2CON^4;
sbit EXEN2  = T2CON^3;
sbit TR2    = T2CON^2;
sbit C_T2   = T2CON^1;
```

```
sbit CP_RL2 = T2CON^0;
```

```
#endif
```

第 3 节 C 语言数据类型

一. 数制的转换关系。在讨论数据类型之前，我们先来了解关于数制之间的关系。

1 二进制数

二进制是计算机运行的数制，它的数码取值只有“0”和“1”，由于二进制是比较难读懂，所以程序员不用下太多功夫去理解它，人与计算机之间的对话一般是用十进制和十六进制。

2. 十进制数

十进制数没有前缀。其数码为 0~9。

以下各数是合法的十进制整常数：

237 -568 65535 1627

以下各数不是合法的十进制整常数：

023 (不能以 0 开始) 23D (含有非十进制数码)

3. 十六进制整常数

十六进制整常数的前缀为 0X 或 0x。请注意这里的是“零”而不是字母“o”。其数码取值为 0~9, A~F 或 a~f。以下各数是合法的十六进制整常数：

0x2A (十进制为 42) 0xA0 (十进制为 160) 0xFFFF (十进制为 65535)

以下各数不是合法的十六进制整常数：

5A (无前缀 0x) 0x3H (含有非十六进制数码)

在程序中是根据前缀来区分各种进制数的。因此在书写常数时不要把前缀弄错造成结果不正确。下面我们来讲解一下不同数制之间的转换：

1、二进制与十六进制之间的转换

转换方法：四位的二进制代表一位的十六进制，只要用 8421 BCD 码代入去，如果为 0 的就是空位，如果为 1 的就保留 BCD 码值，这样最后把 BCD 码相加，就得出一个十六进制数。

例：1101 0010 代入 8421 就变为了 8401 0020，然后把四位的二进制数相加作为一位的十六进制数，这样就得到 8+4+0+1=D，而 0+0+2+0=2，那么 1101 0010 这个二进制数转换为十六进制数就是 0xD2。而十六进制转为二进制只是一个逆过程。

2 十进制转换为十六进制

转换方法：十六进制就是逢十六进 1，但我们只有 0~9 这十个数字，所以我们用 A、B、C、D、E、F 这五个字母来分别表示 10、11、12、13、14、15。字母不区分大小写。十六进制数的第 0 位的权值为 16 的 0 次方，第 1 位的权值为 16 的 1 次方，第 2 位的权值为 16 的 2 次方……所以，在第 N (N 从 0 开始) 位上，如果用 X 表示系数 (X 大于等于 0，并且 X 小于等于 15，即：F)，那么十六进制数的大小为 $X \times 16$ 的 N 次方。

例：假设将一个十六进制数 2AF5 转换成十进制，可以用公式直接计算 $5 \times 16^0 + F \times 16^1 + A \times 16^2 + 2 \times 16^3 = 10997$ 别忘了，在上面的计算中，A 表示 10，而 F 表示 15，而十进制转换成十六进制也是一个逆过程。

其实我们不用太过麻烦地去计算它们，在你的计算机中有一个非常好用的工具，在桌面点击左下角的：开始→程序→附件→计算器，就会弹出图 3-2 的计数器工具，基本上每一台计算机都会有这个计数器，因为它是 Windows 系统自带的，在查看这个菜单功能中还有多种型式供你选择。



图 3-2

二. keil C 语言的数据类型

Keil C 语言支持的数据类型包括**基本类型**，**指针类型**，其中基本类型又可按以下的来划分。

基本类型分为：位型 (bit)、字符型(char)、整型(int)、长整型(long)、浮点型(double)

(在标准的 C 语言中数据类型分为 char、int、short、long、float、double。而在 keil C 语言中 int 和 short 相同，而 float 和 double 相同。)

表 3-3

数据类型	名称	长度	值域
unsigned char	无符号字符型	单字节	0——255
signed char	有符号字符型	单字节	-128—— +127
unsigned int	无符号整型	双字节	0——65535
signed int	有符号整型	双字节	-32768—— +32767

unsigned long	无符号长整型	4 字节	0——4294967295
signed long	有符号长整型	4 字节	-2147483648——+2147483647
float	浮点型	4 字节	+ -1.175494E-38——+ -3.402823E+38
*	指针型	1—3 字节	对象地址
bit	位类型	位	0 或 1
sbit	可寻址位	位	0 或 1
sfr	特殊功能寄存器	单字节	0——255
sfr16	16 位特殊功能寄存器	双字节	0——65535

1. char (字符型)

一、字符型数是在 51 单片机编程中用得最多的一种数据类型，可以加上不同的修饰符，常用的有两种类型，分别为有符号类型与无符号类型。

Kile 系统默认为 signed char 有符号字符类型，字长为 1 字节共 8 位二进制数，数值范围是 -128—— +127。unsigned char 为无符号字符类型数说明，字长为 1 字节共 8 位二进制数，数值范围是 0——255。

二、以下是字符型变量的合法定义：

char a, b; 意思是：a、b 被定义为有符号字符型变量
 unsigned char c; 意思是：c 被定义为无符号字符型变量

2. int (整型)

一、int 同上面的 char 在单片机编程中用得同样是比较多，整型数可以加上不同的修饰符，整型数有以下两种类型。

Kile 系统默认为 signed int 有符号整型。字长为 2 字节共 16 位二进制数，数值范围是 -32768—— +32767。unsigned int 为无符号整型数。字长为 2 字节共 16 位二进制数，数值范围是 0——65535。

二、以下是整型变量的合法定义：

`int a, b;` 意思是: a、b 被定义为有符号整型变量

`unsigned int c;` 意思是: c 被定义为无符号整型变量

3. long(长整型)

一、long 由于字节比较长, 而单片机的内存空间比较小, 如果用了 long 会令到单片机的运速度变慢, 所以一般情况下不要用长整型数据。长整型数可以加上不同的修饰符, 长整型数有以下两种类型。

Kile 系统默认为 signed long 有符号长整型数。字长为 4 字节共 32 位二进制数, 数值范围是 $-2147483648 \sim 2147483647$ 。unsigned long 为无符号长整型数说明。字长为 4 字节共 32 位二进制数, 数值范围是 0——4294967295。

二、以下是长整型变量合法定义:

`long a, b;` 意思是: a、b 被定义为有符号长整型变量

`unsigned long c;` 意思是: c 被定义为无符号长整型变量

4. float(浮点型)

一、float 同上面的 long 的长度都是 4 位, 同样占用内存比较多, 而单片机的空间比较小, 如果用了 float 会令到单片机的运行速度变慢, 所以一般情况下不要用浮点型数据。keil C 中的浮点数字长为 4 个字节共 32 位二进制数, 数值范围是 $\pm 1.175494E-38$ —— $\pm 3.402823E+38$ 。说明: 浮点数均为有符号浮点数, 没有无符号浮点数。

二、以下是浮点型变量的合法定义:

`float a, f;` 意思是: a、f 被定义为浮点型变量

5. * (指针型)

指针型数据可以说是 C 语言中最难的, 也是最有用的一种数据类型, 也有人这样认为, 只有精通指针, 才能精通 C 语言。指针本身就是一个变量, 在这个变量中存放着指向另一个数据地址, 指针变量要占据一定的内存单元, 总的来讲“指针就是一个地址”。本教程后前会有专门详细介绍指针型的节章。

6. bit(位类型)

一、keil C 支持位操作（有某些单片机的编译器是不支持位操作的，就如现在新出 AVR 系列单片机的 ICCAVR 编译器）。bit 类型的数据值只可以是“0”或“1”。说明：bit 类型没有有符号与无符号之分。

二、以下是位类型变量合法定义：

```
bit a, f; 意思是：a、f 被定义为位类型变量
```

7. sbit（可寻址位）

一、sbit 是 keil C 为 51 内核单片机扩展的一个关键字，利用它可以定义可寻址位的对象。常用的定义方法是：**sbit 位变量名=特殊功能寄存器名^位置**。（当然如果不对其进行位定义，也是可以操作的，但是如果进行了位定义，在程序中就更加能容易读懂和操作）。在后面讨论单片机内部资源一章中会有详细的讲解。

二、以下是可寻址位合法定义：

```
sbit led=P0^0; 意思是：单片机的 P0.0 引脚被定义为位操作。
```

8. sfr(特殊功能寄存器)与 sfr 16(16 位特殊功能寄存器)

这两个也是 keil C 扩展的关键字，但是这两个特殊功能寄存器在 keil 中自带的头文件已经被定义，程序员不用对其太过操心。

注意：当我们在编写程序的时候，如果是要用到有符号数（例如：**signed char**）我们完全可以写成 **char**，因为 keil 系统把 **signed char** 默认为 **char**，也就是 **char** 已经代表了有符号数，如果把其写成 **signed char** 也没有错误，但是可以说是完全多余的。

动手实验（1）：

实验目的：理解 unsigned int 与 unsigned char 两者的不同。

实验内容：用 SSH_51MCU 实验板做以下的实验，下面的程序可以点亮 P1 端口的第一只二极管。先将下面延时程序中的数据类型改为 unsigned int，在实验板中观察延时时间的变化，然后再把延时程序中的数据类型改为 unsigned char，再在实验板中观察延时时间的变化。

```
#include<reg52.h>//包含所用单片机对应的头文件
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
```




```
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
    while(1)
    {
        P1=0xfe;// 点亮 P1 端口的第一只二极管
        delay_ms(1000);//把实际参数 1000 传给 n，延时 1000 毫秒，也就是 1 秒
        P1=0xff;// 熄灭 P1 端口的第一只二极管
        delay_ms(1000);//把实际参数 1000 传给 n，延时 1000 毫秒，也就是 1 秒
    }
}
```

步骤：1、打开光盘第 3 章/ unsi_si /unsi_si.Uv2 工程文件，对程序进行编译、链接、调试、产生 unsi_si.hex 烧写文件。

2、把 SST_51 仿真器装上到 SSH_51MCU 实验板上，要特别注意引脚不能插反，否则会烧毁仿真器。而另一端的 DB9 孔插座与计算机的 COM1 串口 DB9 针插座相接。

3、将 SSH_51MCU 实验板接上电源，会看见 LED9 在不停在闪动，这样表示实验板通电工作正常。

4、在图 2-18 的对话框中设置为硬件仿真。点击  按钮将波特率设置为 38400。

5、点击  全速运行观察实验板中二极管运行的两种不同延时后果。

我们由实验中可以理解到，当变量 time 为 unsigned int 类型的时候，对其赋 1000 数值，由于这个变量的值域为 0—65536 所以可以延时 1 秒。但是当 time 为 unsigned char 类型的时候，其值域为 0—255 所以即使我们赋初值 1000，也只能延时 0.255 秒

动手实验（2）：

实验目的：理解 unsigned char 与 signed char 两者的不同。

实验内容：用 SSH_51MCU 实验板做以下的实验，下面的程序以流水灯的形式点亮 P1 端口的 8 只二极管。先将下面 main 函数中 move 变量的数据类型改为 unsigned char，在实验板中观察流水灯的变化，然后再把其数据类型改为 signed char，再在实验板中观察其变化。


```
#include<reg52.h>//包含所用单片机对应的头文件
```

```
void delay_ms(unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
    unsigned char move;//move 为有符号字符型变量
    unsigned char loop;//loop 为有符号字符型变量
    while(1)
    {
        move=0x80;//对 move 赋初值 0x80,
        P1=move;//把 move 赋值于 P1 端口, 只点亮 LED7
        delay_ms(500);//延时 0.5 秒
        for(loop=0;loop<8;loop++)//P1 端循环 8 次
        {
            move>>=1;//move 右移一位
            P1=move;//把 move 赋值于 P1 端口
            delay_ms(500);//延时 0.5 秒
        }
    }
}
```

步骤: 1、打开光盘第 3 章/ unsi_ch /unsi_ch.Uv2 工程文件, 对程序进行编译、链接、

调试产生 unsi_ch.hex 烧写文件。

2、将 SST_51 仿真器正确装上到 SSH_51MCU 实验板上, 另一端与电脑相接, 把 SSH_51MCU 实验板接上电源。

3、在图 2-18 的对话框中设置为硬件仿真。点击  按钮将波特率设置为 38400。

4、点击  全速运行观察实验板中二极管运行的两种不同后果。

我们由实验中可以观察到, 当变量 move 为 unsigned char 类型的时候对其赋值 0x80, 由于 C 语言规定假如一个数为无符号数, 右移多少位只需在高位补“0”, 而左移就从低位补“0”。但是当 move 为 char 类型的时候对其赋 0x80, 由于 C 语言规定假如一个数是有符号数, 无论右移多少位都要确保其原来的符号位不变, 而左移则从低位补“0”。(关于左移和右移等运算符, 在随后的运算符与表达式一节章中会讨论到)

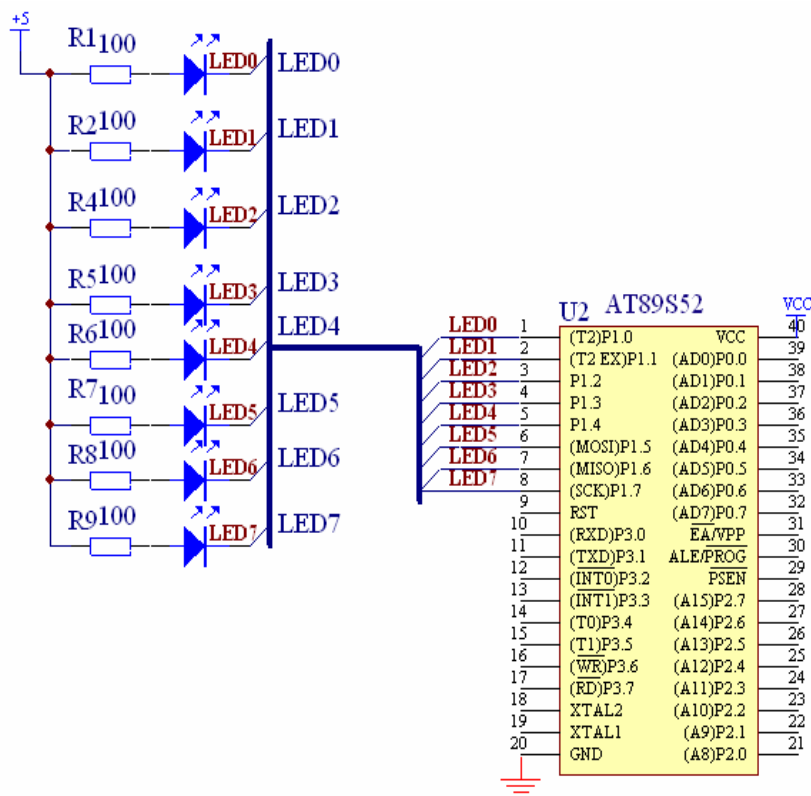


图 3-3

由图 3-3 的电路中可以看到，发光二极管的正极通过限流电阻接电源，而负极则接单片机的 P1 端口。只要端口赋低电平，电流就会从电源流向单片机，从而点亮二极管。大家可能会问，那我用高电平来点亮可以吗！其实当单片机为高电平作为输出的时候它输出电流很少，不足以驱动负载，所为单片机多以低电平有效（所有的 51 系列单片机，其上电默认为高电平，端口的值只有“0”和“1”）。AT89S52 是 8 位总线宽度的单片机，上电的时候默认为“1111 1111”如果我们想那一位为低电平，我们只要将其置“0”就可以了。例如：我要第一只二极管点亮“1111 1110”；我要第二只二极管点亮“1111 1101”；我要第八只二极管点亮“0111 1111”；如此类推。最后将其转换为十六进制数进行操作。

自我练习：

第一，自行编写一个程序，工程命名为“mybit”，利用“sbit”这种形式对 LED2 进行位定义。使其每 500 毫秒闪烁一次。

程序的设计思路：要完成这个练习我们可以参考动手实验（1）的程序。

- (1) 首先在程序的开始就用“sbit led=P1^2;”来进行位定义。
- (2) 在 main 函数当中，当要点亮二极管时可以用“led=0;”来实现，如果要熄灭则“led=1;”。而延时 500 毫秒可以用“delay_ms(500);”来实现。

第二，自行编写一个程序，工程命名为“myport”，利用十六进制数的赋值形式，对 P1 端口

LED5 和 LED7 闪烁点亮，时间为 500ms。

程序的设计思路：要设计这个程序，首先我们要了解图 3-3 的电路结构，根据前面的分析，当我们要点亮 LED5 与 LED7 时其 P1 端口二进制数为“0101 1111”，转换为十六进制数为“0x5f”，然后将此十六进制数写到 P1 端口去“P1=0x5f;”。而熄灭时二进制数为“1111 1111”，转换为十六进制为“0xff”。同时我们也可以参考动手实验（1）。

(以上练习答案附光盘中)

第 4 节. 常量与变量

一. 常量

我们先看看下面的一段代码

```
char i=50;

.....//表示有其它的源代码,

i=13;
```

上面的程序本来“i”的值是 50；但是在程序的运行过程中，由于出于某种需要，再给“i”赋予 13，所以“i”由原来的 50 变为了 13；而像上面这样，在程序中直接给出数值大小的量称为立即数，也称为常数。常数或代表固定不变值的名字称为常量。

1 字符型常量

字符型常量包括字符常量和字符变量。

1. 字符常量

字符常量是用单引号括起来的一个字符。例如：'a'、'b'、'='、'+'、'?' 都是合法的字符常量。在 C 语言中，字符常量有以下特点：

1. 字符常量只能用单引号括起来，不能用双引号或其它括号。
2. 字符常量只能是单个字符，不能是字符串(即多于一个的字符)。
3. 字符常量可以是任意的字符。但数字被定义为字符型之后就不能参与数值运算。如 '5' 和 5 是不同的，'5' 是字符常量，不能参与运算。

转义字符

转义字符是一种特殊的字符常量。转义字符以反斜线“\”开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。而字符串常量是由一对双引号括起来的字符序列。例如：“CHINA”、“C program: ”、“\$12.5” 等都是合法的字符串常量。字符

串常量和字符常量是不同的量。表 3-4 是转义字符及其含义。

表 3-4

转义字符	转义字符的意义
\n	回车换行
\t	横向跳到下一制表位置
\v	竖向跳格
\b	退格
\r	回车
\f	走纸换页
\\	反斜杠
\'	单引号符
\ddd	1~3 位八进制数所代表的字符
\xhh	1~2 位十六进制数所代表的字符

2 整型常量:

- (1) . 十六进制整常数，十六进制整常数的前缀为 0X 或 0x。其数码取值为 0~9、A~F 或 a~f。
- (2) . 十进制整常数，十进制整常数没有前缀。其数码为 0~9。

3. 实型常量(也称为浮点型)

实型常量也称为实数或者浮点数。在 C 语言中，实数只采用十进制。由数码 0~9 和小数点组成。例如：0.0、.25、5.789、0.13、5.0、300、-267.8230 等均为合法的实数。

在程序中是根据前缀来区分各种数制的。因此在书写常数时不要把前缀弄错造成结果不正确。

常量定义

常量在程序的运行过程中是不能被改变的，如果在程序中我们要写一个圆周率的程序，那么 3.14159 会频繁地用到，因为 3.14159 是一个固定不会被改变的值，为了避免在程序中频繁地写这个数，我们应该将其定义为常量。定义常量的两种方法。

一. 用宏来定义

其合法的格式为：`#define 宏名称 宏值`

例：`#define CIRCLE 3.14159`

二. 用常量关键字 const 来定义

其合法的格式为：`const 数据类型 常量名 = 常量值`

例：`const float CIRCLE = 3.14159;`

以上两种是常量合法的定义方法，在程序中凡是用到 3.14159 的语句我们就可以用 CIRCLE 来取代，在程序编译之后，CIRCLE 就成为了 3.14159。常量一经定义了就不能改变，否则会引起语法错误。为了区分常量与变量，使程序更加直观，在定义的时候多般用大写，而用宏来定义这种方法最为常用。

在编写程序的时候：unsigned int, unsigned char 是经常用到的，我们为了书写方便，也可以用宏来定义例如：

```
#define uint unsigned int

#define uchar unsigned char
```

在以后编写程序时我们就可以灵活地用“uchar”来代替“unsigned char”或“uint”来代替“unsigned int”。

二. 变量

1. 字符型

类型说明符为 char，在内存中占 1 个字节，其取值为基本字符常数，即单个字符。

2. 整型

类型说明符为 int，在内存中占 2 个字节，其取值为基本整型常数。

3. 实型变量

其类型说明符为 float 单精度说明符，在 keil C 语言中实型变量占 4 个字节（32 位）内存空间。实型变量说明的格式和书写规则与整型相同。

4. 位型（其值只有“0”或“1”）

请看下面变量 i 的声明

```
char i; //声明了一个字符型的变量 i
```

变量是在程序运行的过程中能任意地改变的数。我们在编写程序的时候，经常会遇到只需声明一个变量，无需对这个值进行赋值，因为这个变量的值是没有固定的，它的值是随机而变化的。但有些时候，出于某种需要对一个变量进行赋初值，那应该怎么办呢！为变量赋初值一般是用“=”，这里“=”不是数学中“等于”的意思，而是“赋值”的意思，下面我们来看一下。声明变量的合法格式：

[[存储类型](#)] [数据类型](#) [[存储器类型](#)] [变量名](#)

除了[数据类型](#)和[变量名](#)是必要之外，其它两个都是可选项。

例如：char x=0,y=8; //x、y 为字符型变量，x 赋值十进制数“0”，y 赋值十进制数“8”

例如：int a=0xff,r; //a、r 为整型变量，a 赋值十六进制数“0xff”，r 的值随机而定。

例如： `float x, y;` //x、y 为实型变量，值随机而定。

从上面的例子可以看出，在声明变量的时候，我们可以全部赋值，也可以只对特定需要的某个变量赋值，同时还可以不对其赋值，任其值随机而定。

其中变量可以分为全局变量和局部变量，请看下面的例子：

```
char t; // “t” 是全局变量，其作用在整个 “.c” 的文件中有效，但是如果一个函数里已
//经有一个以 “t” 命名的变量，则这个全局变量在此函数里无效。
```

```
void delay_ms()
{
    unsigned int i, j; // “i” 与 “j” 是局部变量，只在该函数体内有效
    unsigned int time; // “time” 是局部变量，只在该函数体内有效
}
void main(void)
{
    char k; // “k” 是局部变量，只在该函数体内有效
}
```

在定义变量时，应注意以下几点：

1. 允许在一个类型说明符后，说明多个相同类型的变量。各变量名之间用逗号间隔。类型说明符与变量名之间至少用一个空格间隔。
2. 最后一个变量名之后必须以 “;” 号结尾。
3. 变量说明必须放在变量使用之前。一般放在函数体的开头部分。

动手实验（3）：


实验目的：理解常量的用法。



实验内容：用 SSH_51MCU 实验板做以下的实验，下面的程序以常量的方式分别点亮 P1 端口的 8 只 LED。


```
#include<reg52.h> //包含所用单片机对应的头文件
void delay_ms(unsigned int time) //延时 1 毫秒程序，n 是形式参数
{
    unsigned int i, j;
    for(i=time; i>0; i--) //i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112; j>0; j--) //j 不断减 1，一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
```

```
while(1)
{
    P1=0xfe;      //点亮 LED0
    delay_ms(300); //延时 300 毫秒
    P1=0xfd;      //点亮 LED1
    delay_ms(300); //延时 300 毫秒
    P1=0xfb;      //点亮 LED2
    delay_ms(300); //延时 300 毫秒
    P1=0xf7;      //点亮 LED3
    delay_ms(300); //延时 300 毫秒
    P1=0xef;      //点亮 LED4
    delay_ms(300); //延时 300 毫秒
    P1=0xdf;      //点亮 LED5
    delay_ms(300); //延时 300 毫秒
    P1=0xbf;      //点亮 LED6
    delay_ms(300); //延时 300 毫秒
    P1=0x7f;      //点亮 LED7
    delay_ms(300); //延时 300 毫秒
}
}
```

步骤：1、打开光盘第 3 章/ constant / constant.Uv2 工程文件，对程序进行编译、链接、调试产生 constant.hex 烧写文件。

2、把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接，在图 2-18 的对话框中设置为硬件仿真，点击  按钮将波特率设置为 38400，将 SSH_51MCU 实验板接上电源。

3、用  和  进行调试，观察每一条程序语句在实验板中引起的变化，同时还在观察窗口中观察 P1 端口数值的变化。

(观察窗口的操作：用鼠标 点击两下 ，注意：是点击两下而不是双击。出现图 3-4 的对话界面，输入“P1”，就可以跟踪单步执行的每一条语句，从而看到其执行的结果)

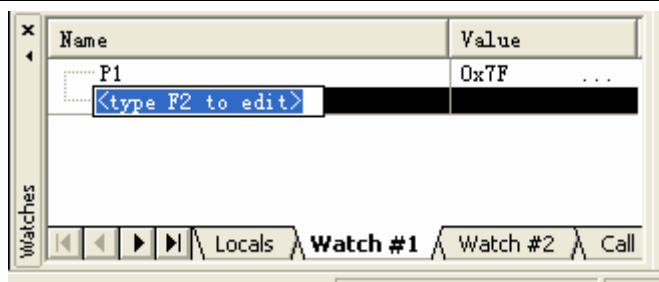


图 3-4

我们由实验中可以观察到，P1 口每延时 300 毫秒都会赋值一个立即数，立即数是一个常量，在程序中只可以作运算，而不能改变。

动手实验（4）：

实验目的：理解变量的用法。

实验内容：用 SSH_51MCU 实验板做以下的实验，下面的程序同上面动手实验（3）的程序所实现的功能完全一样，但是这个程序是以变量形式实现的。



```
#include<reg52.h>//包含所用单片机对应的头文件
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
    unsigned char move;//move 为有符号字符型变量
    unsigned char loop;//loop 为有符号字符型变量
    while(1)
    {
        move=0xfe;//对 move 赋初值立即数 0xfe，
        P1=move;//把 move 赋值于 P1 端口，只点亮 LED0
        delay_ms(300);//延时 0.3 秒
        for(loop=0;loop<8;loop++)//P1 端循环 8 次
        {

            move=(move<<1)|0x01;//move 左移一位
            P1=move;//把 move 赋值于 P1 端口
            delay_ms(300);//延时 0.3 秒
        }
    }
}
```

```
}  
}
```

步骤：1、打开光盘第 3 章/ change / change.Uv2 工程文件，对程序进行编译、链接、调试产生 change.hex 烧写文件。

2、把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接，在图 2-18 的对话框中设置为硬件仿真，点击 **Settings** 按钮将波特率设置为 38400，将 SSH_51MCU 实验板接上电源。

3、用  和  进行调试，观察每一条程序语句在实验板中引起的变化，同时还在观察窗口中观察 move 与 loop 两个变量数值的变化。

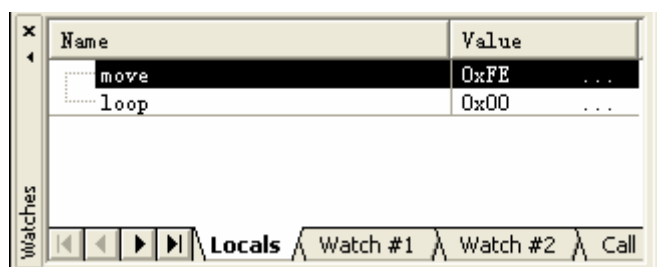


图 3-5

从图 3-5 的观察窗口我们可以看到，“loop” “move” 是两个变量，其值是没有固定的，在程序运行的时候，会不停地按实际的需要对其赋值。上面的程序中用到了一些运算符，在下一节中会讨论到，这个程序只要弄明白变量与常量的根本区别就可以了。

自我练习：

一. 自行编写一个程序，工程命名为“mychange”利用常量与变量组合运算的方式，以流水灯的形式点亮 P1 口的二极管。

程序设计思路：首先在程序开始时先点亮一只二极，然后再在循环语句当中与“P1=P1*2;”的运算来对端口进行点亮。在此语句当中，P1 可以看作是一个变量，其在程序的运行中不断在变化，而常数 2 是一个常量，它在程序的运行当中是不能被改变的，只能作为一种运算。这样 P1 口每一次都乘以 2，就等于左移一位。从而以流水灯的形式点亮 P1 口的二极管。因为单片机运行速度非常的快。所以每点亮一只二极管都要延时。

(以上练习答案附光盘中)

第 5 节 运算符和表达式

在编写程序的时候，要完成某一特定的算法，通常是离不开表达式的，在C语言中，表达式是最常用的对象。运算符按其表达式与运算符的关系可分为单目运算符，双目运算符与三目运算符。C语言的运算符不仅具有不同的优先级，而且还有一个特点，就是它的结合性。在表达式中，各运算量参与运算的先后顺序要遵守运算符优先级别的规定。C语言是一种表达式语言，其可读性非常之高。在现今的单片机编程当中，C语言已成为一种主流。

1. 算术运算符与表达式（用于各类数值运算）

表 3-5

+	加
-	减
*	乘
/	除
%	求余（或称求模运算）
++	自增
--	自减

- (1). 加法运算符“+”，应有两个量参与加法运算。 例如：a+b、5+2 等等。
- (2). 减法运算符（也可作负值运算符）“-”。 例如：-x、-5、7-4 等等。
- (3). 乘法运算符“*”。 例如：7*4、5*6 等等。
- (4). 除法运算符“/”。大家要记住“/”参与运算量均为整型数时，结果也为整型，舍去小数部分。例如：6/2=3、7/2=3
- (5). 求余运算符(求模运算符)“%”。求余运算的值为两数相除后的余数。如10除以3所得的余数1。求余运算符%要求参与运算的量均为整型。 例如：10%3的值为1。
- (6). 自增1运算符为“++”，其功能是使变量的值自增1。

自减1运算符为“--”，其功能是使变量的值自减1。

可有以下几种形式： ++i 意思是：i 自增1后再参与运算。

--i 意思是：i 自减1后再参与运算。

i++ 意思是：i 参与运算后，i 的值再自增1。

i-- 意思是：i 参与运算后，i 的值再自减1。

在理解和使用上容易出错的是 i++和++i。特别是当它们出现在较复杂的表达式或语句当中时。下面我们来看一个例子：

若 i=5;则执行 y=++i 时，先使 i 加1，即 i=i+1=6,再引用其结果，即 y=6. 运算结果为 i=6、y=6。

若 $i=5$; 则执行 $y=i++$ 时, 先引用 i 的值, 即 $y=5$, 再使 i 加 1, 即 $i=i+1=6$, 运算结果为 $i=6$ 、 $y=5$ 。

2. 关系运算符与表达式 (用于比较运算)

表 3-6

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

当两个表达式用关系运算符连接起来时就成为了关系表达式, 通常关系运算符是用来判别某个条件是否成立。当条件成立运算的结果为真; 而当条件不成立运算的结果为假。用关系运算符来运算的结果只有“0”和“1”两种。相对来讲关系运算符是最为容易理解的。

例如: `char a, b, c;`

`a=5; b=7;`

`c=(5<7)` // 因为 5 小于 7, $5<7$ 条件成立, 所为 $c=1$;

`c=(5>7)` // 因为 5 小于 7, $5>7$ 条件不成立, 所为 $c=0$;

`c=(5==7)` // 因为 5 小于 7, $5==7$ 条件不成立, 所为 $c=0$;

3. 逻辑运算符与表达式 (用于逻辑运算)

表 3-7

&&	逻辑与
	逻辑或
!	逻辑非

用逻辑运算符将关系表达式或逻辑量连接起来的的就是逻辑表达式。

逻辑与: **条件式 1 && 条件式 2** (两个条件为真时运算结果为真, 否则为假)

逻辑与运算, 当**条件式 1** 结果为真 (即非 0 值) 与**条件式 2** 结果为真时 (即非 0 值), 运算的结果为真。当**条件式 1** 结果为假 (0 值), 运算的结果为假, 意思是如果第一个条件式的结果为假时, 已经不用去判别条件式 2, 直接输出结果为假。

例如：char a ;

```
char i=9, j=3, k=5;
```

```
a=(i>j)&&(j>k)//运行的结果是：(i>j)为真，(j>k)为假，因此 a=0
```

逻辑或：**条件式 1** || **条件式 2**（两条表达式任其一为真时运算结果为真，当两者同是为假时结果为假）

逻辑或运算。当**条件式 1**的结果与**条件式 2**的结果任其一为真（即非 0 值），输出的结果为真；只有两个同时为假时（0 值），输出的结果才为假。

例如：char a ;

```
char i=9, j=3, k=5;
```

```
a=(i>j)|| (j>k)//运行的结果是：(i>j)为真，(j>k)为假，因此 a=1
```

逻辑非：**! 条件式**

逻辑非运算，把当前的**结果取反**，作为最终的运算结果。

例如：char a ;

```
char i=9, j=3;
```

```
a!=(i>j) //运行的结果是：(i>j)为真，因此 a=0
```

4. 位操作运算符与表达式（参与运算的量，按二进制数进行运算。）

表 3-8

&	按位相与
	按位相或
~	按位取反
^	按位异或
<<	左移
>>	右移

位运算符的作用是按位对变量进行运算，但是并不改变参与运算的值。大家要记住，浮点数是不能进行位操作的。位运算的语法格式为：**变量 1** **位运算符** **变量 2**

表 3-9 是位逻辑运算符的真值表，其中 a 是变量 1、 b 是变量 2。

表 3-9

a	b	$\sim a$	$\sim b$	a&b	a b	a^b
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

位运算符中的左移“<<”意思是把变量 a 的二进制位左移变量 b 指定的位数，其左边移出的位数弃丢。

例如：unsigned char a=0x22, 声明 a 为无符号数（二进制为 0010 0010），a<<2 进行左移后的值为 a=0x88（二进制为 1000 1000）。

又如：signed char b=0x52, 声明 b 为有符号数（二进制为 0101 0010），b<<4 进行左移后的值为 b=0x20（二进制为 0010 0000）。总结来讲，左移：不管是有符号数还是无符号数都是在相应的位补“0”。

位运算符中的右移“>>”意思是把变量 a 的二进制位右移变量 b 指定的位数。其右边移出的位数弃丢。

例如：unsigned char a=0x82 声明 a 为无符号数（二进制为 1000 0010），a>>2 进行右移后的值为 a=0x20（二进制为 0010 0000）。

又如：signed char b=0x82 声明 b 为有符号数（二进制为 1000 0010），b>>4 进行右移后的值为 b=0xf8（二进制为 1111 1000）。总结来讲，右移：如果是无符号数，都是在相应的位补“0”同左移的原理相同；但是如果为有符号数，则要在其左端补入原来数据的符号位（即保持原来数据符号的不变），右端移出的位数被弃丢。

5. 赋值运算符（用于赋值运算）

表 3-10

	赋值	简单赋值
=	赋值	简单赋值
+=	加法赋值	复合算术赋值
-=	减法赋值	
*=	乘法赋值	
/=	除法赋值	
%=	取余赋值	
&=	逻辑与赋值	复合位运算赋值
=	逻辑或赋值	
^=	逻辑异或赋值	
>>=	右移赋值	
<<=	左移赋值	

简单赋值运算符。简单赋值运算符记为“=”，这不是简单“等于”的意思，由“=”连接的式子称为赋值表达式。其一般语法格式为：**变量=表达式**

例如：x=a+b、y=a*5+6-2 意思是：a+b 的值赋给 x、a 乘以 5 加上 6 减 2 的值赋给 y。

关于简单运算符是非常容易理解的，在前面的实验我们已经大量使用过。如果在运算的表达式中，赋值运算符两边的数据类型不相同，系统将自动进行类型转换。即把赋值号右边的类型转换成左边的类型。具体规定如下：

1. 实型数赋予整型数，舍去小数部分。
2. 整型数赋予实型数，数值不变，但将以浮点数的形式存放，即增加小数部分(小数部分的值为 0)。
3. 字符型数赋予整型数，由于字符型为一个字节，而整型为二个字节。字符型数赋值于低位，高位则补“0”。
4. 整型数赋予字符型数，只把低八位赋予字符量，而高位则去丢。

下面我们来讨论一下关于复合赋值运算符。在赋值运算符“=”之前加上其它二目运算符可构成复合赋值符。其构成合法的复合赋值表达式为：**变量 双目运算符=表达式**。

例如：p+=7 等价于 p=p+7、i<<=j+6 等价于 i=x<<(j+6)、a*=b 等价于 a=a*b
复合赋值运算符这种写法，对初学者而言可能不习惯，但十分有利于编译器处理，能提高编译效率并产生质量较高的目标代码。

6 其它运算符与表达式

表 3-11

?:	用于条件求值运算符	条件运算符
,	用于把若干表达式组合成一个表达式	逗号运算符
*	用于取内容运算符	指针运算符
&	用于取地址运算符	
sizeof	用于计算数据类型所占的字节数	求字节数运算符
()	圆括号运算符	特殊运算符
[]	下标运算符	
->	指向结构体成员运算符	
.	结构体成员运算符	

- (1) C 语言中逗号“，”也是一种运算符，称为逗号运算符。其功能是把两个表达式连接起来组成一个表达式，称为逗号表达式。其合法的表达形式为：**表达式 1，表达式 2，表达式 3………表 达式 n**

在程序运行的时候，从左到右算出整个表达式的值，而整个表达式的值就是最右边**表达式 n** 的值。

- (2) “?:”是一个三目运算。其功能是把三个表达式连接起来成为一个表达式，合法的表达式形式是：

逻辑表达式? 表达式 1: 表达式 2

条件运算符的作用简单来说就是根据逻辑表达式的值来选择使用那个表达式的值。当逻辑表达式的值为真时（非 0 值），整个表达式的值为表达式 1 的值；当逻辑表达式的值为假时（0 值），整个表达式的值为表达式 2 的值。

例：如有 a=1, b=2，在程序当中比较两个值的大小，把最小的值放入 y 中, 程序可以这样写：

```
if(a<b)
y=a;
else
y=b;//这段程序目的是，假如 a<b，就把 b 的值赋给 y，否则就把 a 的值赋给 y，
```

上面的一段程序可以用条件运算符来代替：y=(a<b)? a:b

从上面的条件运算符可以看出，程序变得较为简洁，同时也变得较为难读懂，建议初学者少用。剩余的一些运算符在后面会有专门的章节来讨论。

7 运算符的优先级与结合性

表 3-12

优先级	操作符	功能	结合性
1 (最高)	()	改变优先级	从左至右
	[]	数组下标	
	—>	指向结构成员	
	.	结构体成员	
2	++ --	自增 1 自减 1	从右至左
	&	取地址	
	*	取内容	
	!	逻辑取反	
	~	按位取反	
	+ -	正数 负数	
	()	强制类型转换	
sizeof	计算内存字节数		
3	* / %	乘法 除法 求余	从左至右
4	+ -	加法 减法	
5	<< >>	左移位 右移位	
6	< <= > >=	小于 小于等于 大于 大于等于	
7	== !=	等于 不等于	
8	&	按位与	
9	^	按位异或	
10		按位或	
11	&&	逻辑与	
12		逻辑或	
13	?:	条件运算符	

14	= += -= *= /= %= &= ^= = <<= >>=	复合赋值运算符	从右至左
15 (最低)	,	逗号运算符	从左至右

从表 3-12 可知，C 语言中的运算符的运算优先级共分为 15 级。1 级最高，15 级最低。在表达式中，优先级较高的要比优先级较低的先进行运算。而在一个运算量两侧的运算符优先级相同时，则按运算符的结合性所规定的结合方向处理。C 语言中各运算符的结合性分为两种，即左结合性（自左至右）和右结合性（自右至左）。

例如：算术运算符的结合性是自左至右，即先左后右。

如有表达式 $x-y+z$ 则 y 应先与“-”号结合，执行 $x-y$ 运算，然后再执行 $+z$ 的运算。这种自左至右的结合方向就称为“左结合性”。

例如：而自右至左的结合方向称为“右结合性”。最典型的右结合性运算符是赋值运算符。

如有 $x=y=z$ 由于“=”是右结合性，应先执行 $y=z$ 再执行 $x=(y=z)$ 运算。

C 语言运算符当中有不少为右结合性，大家在写程序的时候应注意加以区别，以避免造成不必要的错误。另外在 C 语言中规定，在表达式后面加一个“;”号就成为了表达式语句，在前一章介绍 KEIL 软件的使用时已经同大家讲过，如果在表达句的后面少了一个“;”，那么在系统编译时就会把错误指向下一行的程序语句。同样，在程序中加入了全角符号，也会造成编译的错误。

动手实验（5）：

实验目的：理解算术运算符的用法。

实验内容：编写一个程序来计算梯形的面积，当输入梯形的上底，下底与高时，输出它面积。

```
#include<reg52.h>//包含所用单片机对应的头文件
#include<stdio.h>//包含输入输出函数的头文件
void uart(void)//串口初始化函数
{
    SCON=0x40;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
main()
{
    int up, down, high;                //定义梯形的，上底，下底，高
```

```
int result; // 面积
uart(); // 串行口初始化
while(1)
{
    printf("Please input trapezoid of up, descend and high\n");/*提示输入梯形的上底, 下底和高。*/
    scanf("%d%d%d", &up, &down, &high); //从键盘输入梯形的, 上底, 下底, 高
    result=(up+down)*high/2; // 计算梯形面积公式
    printf("result=%d\n", result); //输出梯形面积
}
}
```

注意：在利用实验板串口与电脑通信时，要先把 JP4、JP5、JP6 跳针连接上。

- 步骤：1. 打开光盘第 3 章/ operation / operation.Uv2 工程文件，对程序进行编译、链接、调试产生 operation.hex 烧写文件。
2. 把 operation.hex 烧写文件烧写到 AT8952 单片机当中，然后再安装到实验板上。
3. 利用串口线把 SSH_51MCU 实验板与电脑串口相接。
4. 打开串口调试器。（如图 3-6 所示，串口调试器附光盘中，读者也可以从网上下载）选择相应的端口号（默认为 COM1），选择波特率为 9600，不选取十六进制显示，不选取十六进制发送，然后打开串口。再将实验板接上电源，此时 AT89S52 单片机就会发送一串英文数据到调试器的接收区（如图 3-7 所示），意思是要求输入梯形的上底、下底和高。



图 3-6



图 3-7

5. 首先在发送区输入梯形的第一个数据“上底”（设为 8），再按回车键使 scanf 函数确认有数据输入，然后再按串口调试器的“手工发送”。此时在串口调试器就会接收到相应的数据。（如图 3-8）



图 3-8

6. 将发送区的数据删除，输入梯形的第二个数据“下底”（设为 20），再按回车键使 scanf 函数确认有数据输入，然后再按串口调试器的“手工发送”。此时在串口调试器就会接收到相应的数据。（如图 3-9）



图 3-9

7. 将发送区的数据删除，输入梯形的第三个数据“高”（设为 30），再按回车键使 scanf 函数确认有数据输入，然后再按串口调试器的“手工发送”。此时串口调试器就会接收到相应的数据。（如图 3-10）同时单片机会根据你输入梯形的三个数（上底、下底、高）而计算出梯形的面积并输出。从图 3-10 我们可以看到梯形的面积为 420。如果你再次按上面的方法输入，它就会再次计算。

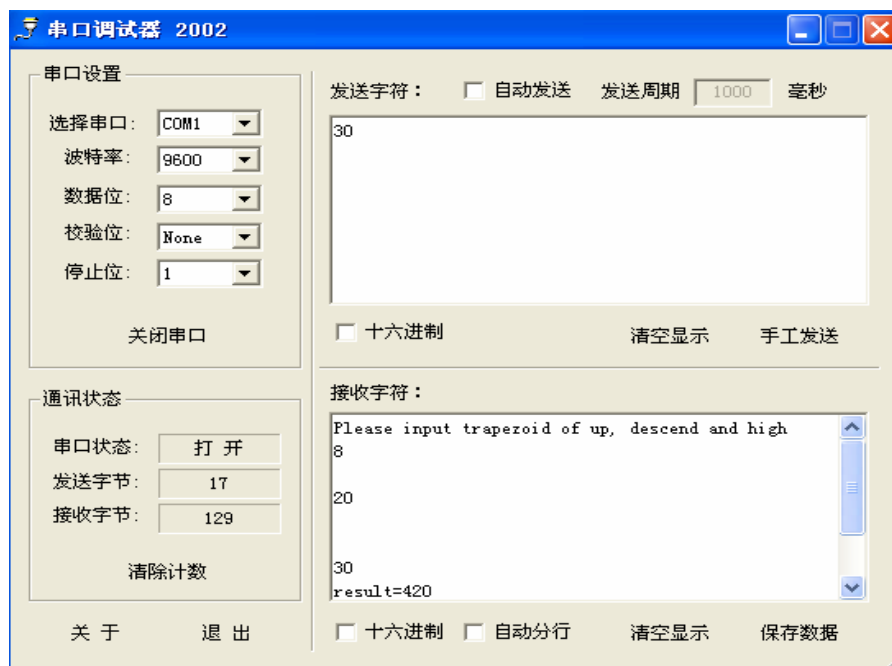


图 3-10

自我练习

- (1) 自行编写一个程序命名为“compare”利用“<<”、“>>”、“~”对 P1 端口的发光二极管以点亮一栈灯的方式先进行左移，再进行右移，最后集体闪动三次，时间为 300ms。

- 程序设计思路：
- (1) 进行单片机的头文件包含与建立一个延时函数。
 - (2) 先以 `P1=0xfe;` 这种方式点亮低位，然后用 `for` 循环语句循环执行 `P1=(P1<<1)|0x01` 八次，使其移动点亮 P1 端口的发光二极管。
 - (3) 以 `P1=0xf7;` 这种方式点亮高位，然后用 `for` 循环语句循环执行 `P1=(P1>>1)|0x80;`八次，使其移动点亮 P1 端口的发光二极管。
 - (4) 以 `P1=0xff;` 这种方式熄灭全部的 LED，然后以 `for` 循环语句循环执行 `P1=~P1;`三次，使其闪烁点亮 P1 端口的发光二极管。
 - (5) 在每一次 LED 二极管的点亮时延时 300ms。

最后关于循环语句读者可以先参考前面的程序。在以后的节章会讨论到。

(练习答案附光盘中)

第6节 C 语言的基本语句

一 选择性结构语句

1. if 语句

if 语句有三种结构形式，下面我们分别介绍。

1 语法格式：

```
if(条件表达式)
{
    语句
}
```

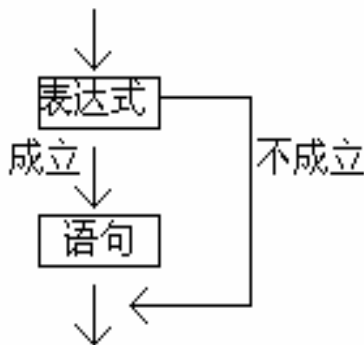


图 3-11

程序的执行过程为：假如条件表达式成立，就执行完花括号里面的语句再往下执行，否则就跳过花括号里面的语句而继续往下执行，如图 3-11 所示。

```
例 1: char a, b, c;  
      a=5;b=8;  
      if(a>b) c=a;  
      c=b;
```

因为 a 的值比 b 的值要小，所以条件不成立，则 C 的值为 8；

```
例 2: char a, b, c;  
      a=5;b=8;  
      if(a<b) c=a;  
      c=b;
```

例 2 中的 if 语句表达式将 “>” 改为 “<”。则同样的程序运行就有所改变，因为 a 的值比 b 的值要小，所以条件成立，则 C 的值先为 5，然后再往下执行最后为 8；

2 语法格式：

```
if(条件表达式)  
{  
    语句 1  
}  
else  
{  
    语句 2  
}
```

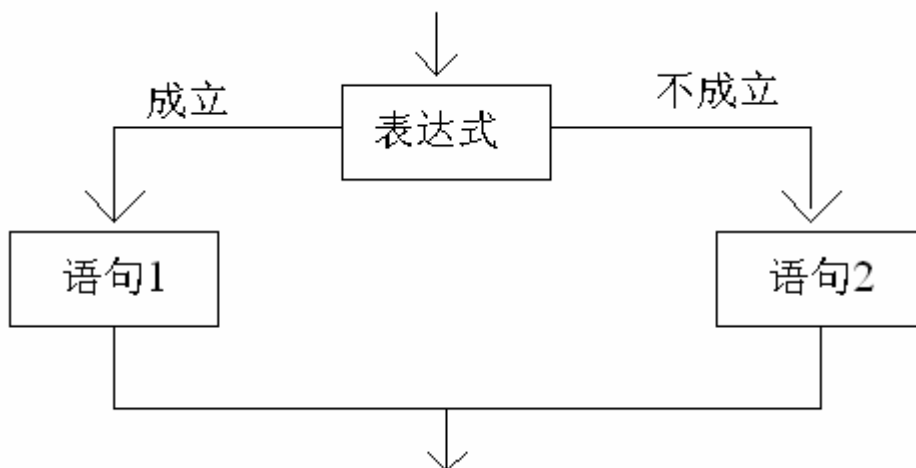


图 3-12

程序的执行过程为：假如条件表达式成立，就执行语句 1，否则就执行语句 2，如图 3-12 所示。

```
例 3 char a, b, c;  
      c=1;  
      if(a==b)  
          c++;  
      else  
          c--;
```

例 3 中程序的运行结构为：假如 a 的值等于 b 的值条件成立，c 则自加 1 结果为 2。否则，a 的值等于 b 的值条件不成立，c 则自减 1 结果为 0。

3 语法格式：

```
if(条件表达式 1)  
{  
    语句 1  
}  
else if(条件表达式 2)  
{  
    语句 2  
}  
else if(条件表达式 3)  
{  
    语句 3  
}  
else  
{  
    语句 4  
}
```

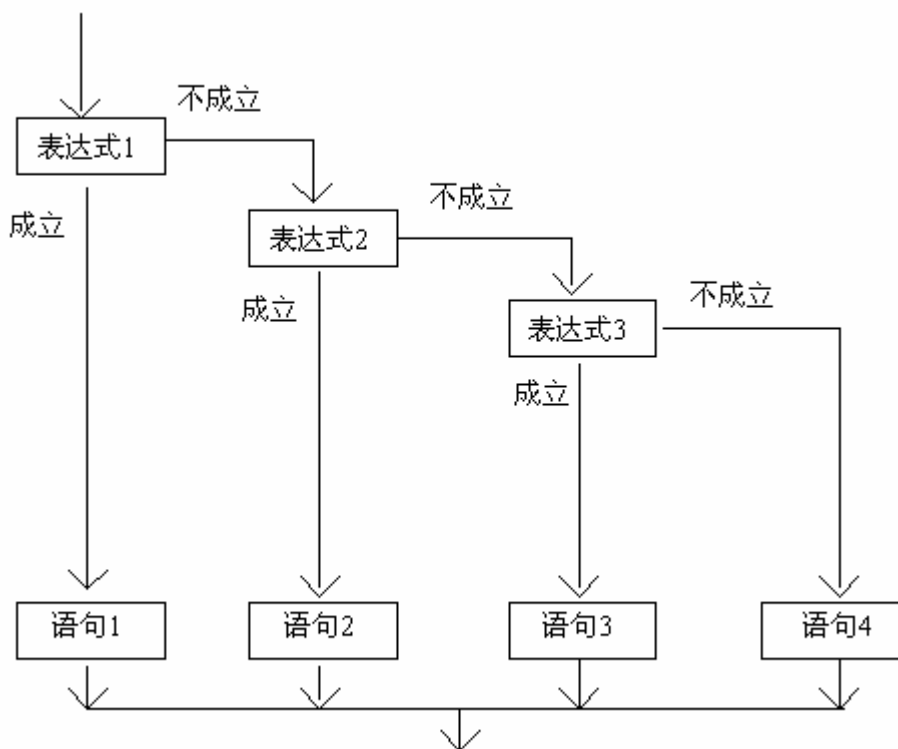


图 3-13

程序的执行过程为:程序运行时从上而下地对条件表达式进行判断,假如条件表达式成立,就执行相应的语句。当以上的表达式都不成立时,则执行 else 相应的语句,图 3-13 所示。

```
例 4 char i, j;
    if(i==0) {j=j+2;}
    else if(i==1) {j=j+3;}
    else if(i==2) {j=j+4;}
    else {j=j+5;}
```

假如 i 的值为 0 条件成立,则 j 的值为 j 加 2;假如 i 的值为 1 条件成立,则 j 的值为 j 加 3;假如 i 的值为 2 条件成立,则 j 的值为 j 加 4;当以上条件均不成立时,则 j 的值为 j 加 5。

动手实验 (6)

实验目的:学习 if 语句的使用。

实验内容:利用实验板的串行通信与电脑相接,从电脑键盘输入两个数,比较数值的大小,在串口调试器中输出一个最大的数。

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>//包含标准输入输出函数头文件
```

```
#define uint unsigned int
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
void main(void)
{
    uint x,y;
    uart();//初始化串口
    while(1)
    {
        printf("please! input two int number\n");
        scanf("%d%d",&x,&y);           //从键盘输入两个值。
        if(x>y)                         //比较 X 与 Y 的大小
            printf("the big number is %d\n",x);//假如 x 的值比 y 的值要大，输出 x 的
            值
        else
            printf("the big number is %d\n",y);//否则输出 y 的值
    }
}
```

步骤：1. 打开光盘第 3 章/ if / if.Uv2 工程文件，对程序进行编译、链接、

调试产生 if.hex 烧写文件。

2. 把 if.hex 烧写文件烧写到 AT8952 单片机当中，然后再安装到实验板上。

3. 利用串口线把 SSH_51MCU 实验板与电脑串口相接。

4. 打开串口调试器，选择相应的端口号（默认为 COM1），选择波特率为 9600，不选取十六进制显示，不选取十六进制发送，然后打开串口。再将实验板接上电源，此时 AT89S52 单片机就会发送一串英文数据到调试器的接收区（如图 3-14 所示），意思是要求输入两个整型数。



图 3-14

5. 从串口调试器的发送区输入第一个整型数（设为 78），然后按下回车键，再按串口调试器的手工发送按钮。同时串口调试器的接收区接收到相应的字符。（图 3-15 所示）



图 3-15

6. 删除了发送区之前输入的数据后，再从串口调试器的发送区输入第二个整型数（设为 51），然后按下回车键，再按串口调试器的手工发送按钮。同时串口调试器会接收到

“the big number is 78” 的信息，意思是你输入的两个数当中，最大的数为 78。（图 3-16 所示）

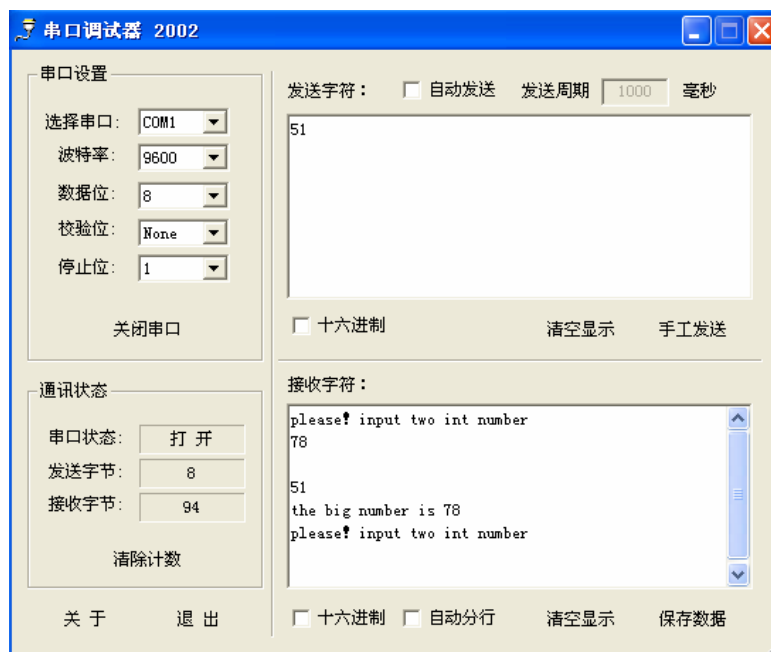


图 3-16

动手实验（7）

实验目的：学习 if----else if ----else 语句的使用。

实验内容：用实验板中的 K0、K1 利用 if----else if ----else 语句进行判断，假如 K0 按下了则 P1 口的高 4 位被点亮；假如 K1 被按下了则 P1 口的低 4 位被点亮，否则 P1 口全部被点亮。

```
#include<reg52.h>//文件包含
void main(void)
{
    while(1)
    {
        if((P3|0xfb)==0xfb)//判断是否 K0 被按下
            P1=0x0f;//点亮第一栈灯
        else if((P3|0xf7)==0xf7)//判断是否 K1 被按下
```



```
        P1=0xf0;//点亮第二栈灯
    else
        P1=0x00;//否则 P1 端口全部被点亮
    }
}
```

步骤：1. 打开光盘第 3 章/ if_else / if_else.Uv2 工程文件，对程序进行编译、链接、

调试产生 if_else.hex 烧写文件。

2. 把 if_else.hex 烧写文件烧写到 AT8952 单片机当中，然后再安装到实验板上。最后将实验板接上电源。

3. 分别按下 K0、K1 观察 P1 端口发光二极管的变化。

自我练习：

一. 利用 if 语句自行编写一个程序，工程名命名为“arrange”。从串口调试器的发送区任意输入三个整型数 x、y、z，在接收区按从小到大的顺序排列输出。

程序的设计思路：

(1) 首先判断 if (x>y) 则把 x 与 y 的值对调，因为 x 是 x 与 y 两者中最小；

(2) 再判断 if (x>z) 则把 x 与 z 的值对调，因为 x 是 x 与 z 两者中最小，也是三者中的最小。

(3) 然后判断 if (y>z) 则把 y 与 z 的值对调，因为 y 是 y 与 z 两者中最小，即三者中的第二。

(4) 最后用 printf 函数分别输出三者的值。

(答案附光盘中)

2. switch 语句

语法格式：

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

此开关语句与前面 if 语句 3 的执行过程是相似的。其执行过程是：计算表达式的值，然后与 case 后面的常量表达式相比较，假如相同的则执行其后的语句。表达式的值与所有 case 后面的常量表达式均不相同时，则执行 default 后面的语句。

```
例5   char a,b;
      switch(a)
      {
        case 1:b=a*2;
        case 2:b=a*3;
        case 3:b=a*4;
        case 4:b=a*5;
        case 5:b=a*6;
        default:b=a*7;
      }
```

大家有没有注意到 switch 语句与前面讲的 if 语句的语法格式有一点不同，就是 switch 语句的 case 常量表达式后面有一个“break”关键字。但是如果没有了“break”关键字会是什么样啊！就如例 5 所示，假如 a 的值为 2，那么我们想要的结果只是 a 的值与 3 相乘再赋予给 b，结果 b 的值为 6。但是程序的执行并不是这样，它会继续往下执行，直至到执行完“default:b=a*7;”为止。那么最后的结果为 14。显然这不是我们想要的运行结果。我们应该如例 6 那样在后面添加一个“break”关键字。

```
例6   char a,b;
      switch(a)
      {
        case 1:b=a*2;break;
        case 2:b=a*3;break;
        case 3:b=a*4;break;
        case 4:b=a*5;break;
        case 5:b=a*6;break;
        default:b=a*7;
      }
```

如例 6 所示，应特别注意。为了避免上述情况，C 语言提供了一种 break 语句，专用于跳出 switch 语句，break 语句只有关键字 break，没有参数。

动手实验（8）

实验目的：学习 switch 语句的使用。

实验内容：从串口调式器输入 0~7 中的任意值去控制实验板中 P1 端口对应的 LED，0 对应 LED0；1 对应 LED1；如此类推。如要输入的值不是 0~7 则提示“输入错误”。

```
#include<reg52.h>//包含所用单片机对应的头文件
```

```
#include<stdio.h>//包含输入输出函数的头文件
void uart(void)//串口初始化函数
{
    SCON=0x40;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
main()
{
    int sw_code;//声明一个中间变量
    uart(); //串口初始化
    printf("Please control LED\n");//提示"请控制LED"
    while(1)
    {
        scanf("%d",&sw_code);//从串口输入一个值
        switch(sw_code)
        {
            case 0://输入一个"0"
            {
                P1=0xfe;//点亮第一栈 LED0
                printf("LED0 is bright\n");//提示点亮的是那栈灯
                break;//退出
            }
            case 1://输入一个"1"
            {
                P1=0xfd;//点亮第一栈 LED1
                printf("LED1 is bright\n");
                break;
            }
            case 2://输入一个"2"
            {
                P1=0xfb;//点亮第一栈 LED2
                printf("LED2 is bright\n");
                break;
            }
            case 3://输入一个"3"
            {
                P1=0xf7;//点亮第一栈 LED3
                printf("LED3 is bright\n");
                break;
            }
        }
    }
}
```

```
}
case 4://输入一个"4"
{
    P1=0xef;//点亮第一栈 LED04
    printf("LED4 is bright\n");
    break;
}
case 5://输入一个"5"
{
    P1=0xdf;//点亮第一栈 LED5
    printf("LED5 is bright\n");
    break;
}
case 6://输入一个"6"
{
    P1=0xbf;//点亮第一栈 LED6
    printf("LED6 is bright\n");
    break;
}
case 7://输入一个"7"
{
    P1=0x7f;//点亮第一栈 LED7
    printf("LED7 is bright\n");
    break;
}
default: printf("input the error\n");//假如输入的不是 0~7, 提示输入
错误。
}
}
}
```

实验步骤:

1. 打开光盘第 3 章/ switch / switch.Uv2 工程文件, 对程序进行编译、链接、调试产生 switch.hex 烧写文件。
2. 把 switch.hex 烧写文件烧写到 AT8952 单片机当中, 然后再安装到实验板上。
3. 利用串口线把 SSH_51MCU 实验板与电脑串口相接。
4. 打开串口调试器, 选择相应的端口号 (默认为 COM1), 选择波特率为 9600, 不选取十六进制显示, 不选取十六进制发送, 然后打开串口。

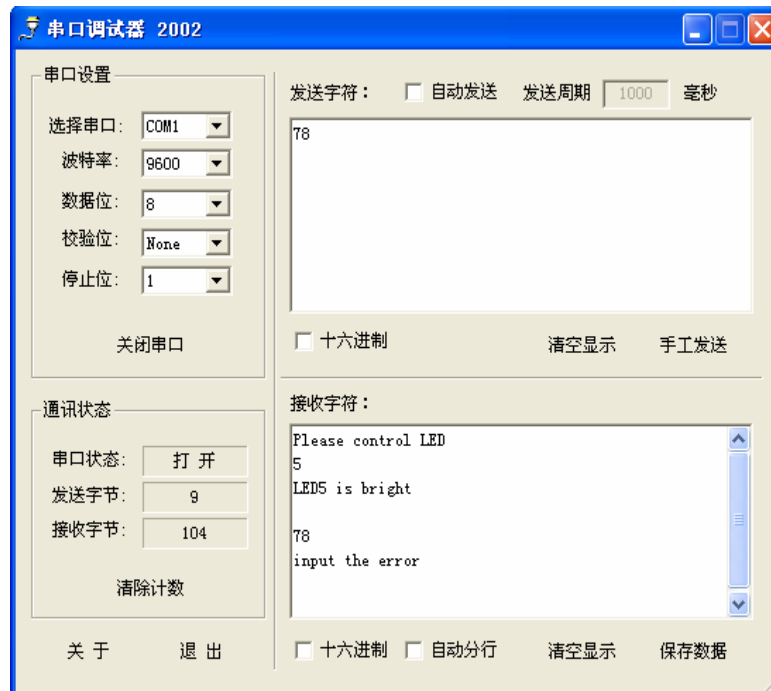


图 3-17

5. 再将实验板接上电源，此时 AT89S52 单片机就会发送一串英文数据到调试器的接收区，

意思是要求输入数据进行控制实验板中的 LED。假如从发送区中输入 0~7 中任意一个数，那个实验板就会点亮相应的 LED，同时还会在接收区提示那一只 LED 被点亮，但是如果输入的数不是 0~7 中的任意值，则会在接收区显示“input the error”的信息。（如图 3-17）

6. 试验完上面的步骤以后，再把程序中的“break”语句去丢，了解在 switch 语句当中如果没有了“break”语句会是什么情况。使自己加深理解。

自我练习：

一. 利用 switch 语句，自行编写一个程序，工程名命名为“myswitch”。任意按下实验板中的 4 个按键，将信息发送到串口调试器中，提示是哪一个按键被按下了。

程序的设计思路：首先定义一个延时程序，同时对串口进行初始化。声明一个中间变量，然后用 if 语句进行判断，当不同的按键被按下时，中间变量被赋予不同的值。最后可以用 switch 语句按中间变量的值进行相应的操作，在 case 中用 printf 函数发送数据到串口调试器进行提示。

（答案附光盘中）

二 循环控制语句

1. for 循环

语法格式:

```
for(表达式 1; 表达式 2; 表达式 3)
{
    循环体语句
}
```

在 for 语句中有三个表达式, 其中每个表达式在语法中担任不同的角色, 下面我们来看看:

表达式 1: 初始设定表达式。

表达式 2: 循环条件表达式。

表达式 3: 更新表达式。

所以 for 语句可以更巧妙地理解为, 初始设定表达式总是一个赋值语句, 它用来给循环控制变量赋初值; 循环条件表达式是一个关系表达式, 它决定何种情况退出循环; 更新表达式是按每循环一次后对 初始设定表达式的变量值进行更新, 当更新后的值使循环条件表达式不成立时, 则退出循环语句继续往下执行。这三个部分之间用“;”分开。下面的书写可以使大家更加容易理解。

```
for(初始设定表达式; 循环条件表达式; 更新表达式)
```

```
{
    循环体语句
}
```

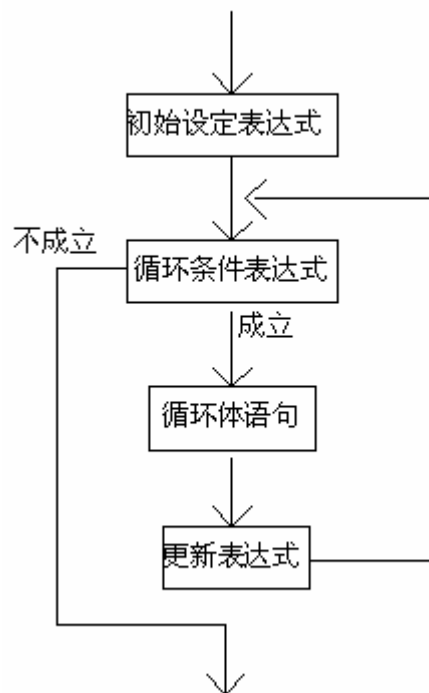


图 3-18

图 3-18 为 for 语句的执行结构流程图，下面我们来对其执行过程进行分析：

- (1) 先计算初始设定表达式。
- (2) 计算循环条件表达式。若其表达式条件成立，则执行 for 循环体语句；若其表达式条件不成立，则不进入 for 循环体语句，继续往下执行。
- (3) 计算更新表达式。
- (4) 转回上面第 (2) 步继续执行。
- (5) 循环结束，执行 for 语句下面的程序语句。

例 7

```
char i;           //声明变量 i
P1=0xfe;         //点亮第一只 led 管
for(i=0;i<8;i++)//for 循环语句
{
    P1<<=1;      //每循环一次点亮一只 led 管
}
```

例 7 的程序中，先是点亮 P1 口的第一只 led 管，然后就进入了循环语句，在这里我们可以利用上面的图 3-18 的流程图来进行分析。

- (1) 先将 i 赋予初值 0。
- (2) 计算 i<8 的条件是否成立。若 i<8 条件成立，则进入 for 循环体，执行“P1<<=1;”这条语句；若 i<8 条件不成立，则不进入 for 循环体语句，继续往下执行。
- (3) 计算 i++。
- (4) 转回上面第 (2) 步继续执行。
- (5) 循环结束，执行 for 语句下面的语句。

其实例 7 的 for 语句是一个典型的用法，但是在实际应用中 for 语句是一个非常灵活的结构性语句，因为 for 语句里面的“初始设定表达式”“循环条件表达式”“更新表达式”是可选项。下面我们就以例 7 为例来进行讨论。

```
(1) for(;i<8;i++)//for 循环语句
{
    P1<<=1;      //每循环一次点亮一只 led 管
}
```

在小括号的“初始设定表达式”被省略了，那么 i 的初值为多少呢！在 KEIL C 当中，若果声明了一个变量，但是未对其赋予任何值，则 KEIL 软件在进行编译的过程中会对其赋予 0 值。

```
(2) for (i=0; ;i++)//for 循环语句
{
    P1<<=1;      //每循环一次点亮一只 led 管
}
```

在小括号的“循环条件表达式”被省略了。因为有了循环条件表达式的判断，所以循环条件永远都是成立的。也就是说一旦进入了循环体，就永远都不会退出，不停地循环执行。

```
(3) for (i=0; i<8;)//for 循环语句
{
    P1<<=1;      //每循环一次点亮一只 led 管
}
```



```
    i++;        //更新表达式
}
```

在小括号的“更新表达式”被省略了。但是“更新表达式”改放在循环体里面，这样也是符合语法逻辑，当 i 小于 8 的条件不成立时，同样也能退出循环体。

```
(4) for (; ;) //for 循环语句
{
    P1<<=1;    //每循环一次点亮一只 led 管
}
```

在小括号里的三个表达式都被省略了。表示循环条件永远都会被成立，不会退出循环体。

以上的 for 语句的格式都是合法的，大家可以在编写程序的时候按实际需要而灵活运用。

动手实验（9）

实验目的：学习 for 语句的使用。

实验内容：利用实验板的 LED 管，用 for 循环语句进行花式表演。以下程序用到了多种的 for 语法格式，读者要在实验中认真地去理解它，以并灵活地运用到自己的程序算法当中去！



```
#include<reg52.h> //包含单片机对应的头文件
#define uchar unsigned char //宏定义
void delay_ms(unsigned int time) //延时函数
{
    unsigned int i, j;          //声明 i, j 为整型变量
    for(i=time; i>0; i--)      //将变量 time 赋予 i, 当 i 大于 0 的条件成立, 执行循环体, 然后 i 自减 1
        for(j=112; j>0; j--)  //将变量 112 赋予 j, 当 j 大于 0 的条件成立, 执行循环体, 然后 j 自减 1
            {;}               //空等待, 直到以上的循环条件结束, 从而实现延时。
}
void main(void)
{
    uchar i, j;                //声明 i, j 为字符型变量
    uchar move_1, move_2;      //声明 move_1, move_2 为字符型变量
    for(;;)                    //死循环, 永远不会退出循环体
    {
        i=0;                   //i 赋予 0
        for(; i<2;)            //假如 i 小于 2 执行循环体语句
        {
            P1=0x7e; delay_ms(200); //点亮 LED0。LED7
            P1=0xbd; delay_ms(200); //点亮 LED1。LED6
        }
    }
}
```

```
        P1=0xdb;delay_ms(200); //点亮 LED2。LED5
        P1=0xe7;delay_ms(200); //点亮 LED3。LED4
        i++; //i 自加 1，为退出循环体制造条件
    }
    for(i=0;i<2;i++) //i 赋予 0，当 i 小于 2 的条件成立，执行
    循环体，然后 i 自加 1
    {
        P1=0xe7;delay_ms(200); //点亮 LED3。LED4
        P1=0xdb;delay_ms(200); //点亮 LED2。LED5
        P1=0xbd;delay_ms(200); //点亮 LED1。LED6
        P1=0x7e;delay_ms(200); //点亮 LED0。LED7
    }
    move_1=0x7f; //move_1 赋值 0x7f
    for(i=0;i<8;i++) //i 赋予 0，当 i 小于 8 的条件成立，执行
    循环体，然后 i 自加 1
    {
        P1=move_1; //move_1 赋于 P1 端口
        move_1=(move_1>>1)|0x80; //move_1 右移一位，然后与 0x80 相或，
    高位总保持为 1
        delay_ms(300); //延时 300ms
        move_2=0xfe; //move_2 赋值 0xfe
        for(j=0;j<8;j++) //循环语句嵌套，意思是在循环语句中可以
    再有循环语句。
        {
            //i 赋予 0，当 i 小于 8 的条件成立，执行
    循环体，然后 i 自加 1
            P1=move_2; //move_2 赋于 P1 端口
            move_2=(move_2<<1)|0x01; //move_2 左移一位，然后与 0x01
    相或，低位总保持为 1
            delay_ms(100); //延时 100
        }
    }
}
```

实验步骤：1. 打开光盘第 3 章/ for / for.Uv2 工程文件，对程序进行编译、链接、调试产生 for.hex 烧写文件。

2. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

设置为硬件仿真。正确选择对应的 COM 端口，点击  按钮将波特率设置为 38400。

3、用  和  进行调试，观察 for 循环语句是如何执行的，在执行 for 循环语句的同时，还要观察变量窗口各变量的变化和实验板的 LED 灯的最终结果，从而理解 for 循环语句的用法。（如图 3-19 所示）

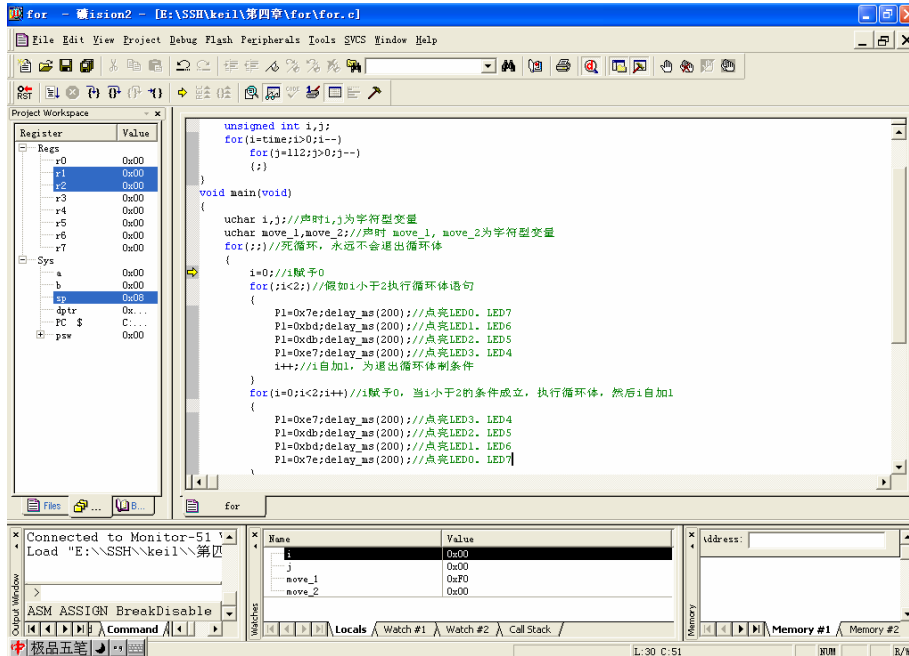


图 3-19

自我练习：

自行编写一个程序，将其命名为“myfor”。利用 for 循环语句先在串口中发送两个“AT89S52”的字符串，再发送三个“I need learn”的字符串，然后不断周而复始地循环。意思是：“AT89S52 单片机，我一定要学会”。

程序的设计思路：如果我们理解了上面动手实验（9）的程序，那么要设计这个程序就显得较为简单了。首先包含相应的头文件，编写一个延时函数和串口初始化函数，然后利用以下这种形式进行循环输出字符串：

for(初始设定表达式；循环条件表达式；更新表达式)

```

{
    循环体语句
}

```

当然啦，要实现不断周而复始的循环我们可以用 for(;;) 这种语法格式。

（以上练习答案符光盘中）

2. while 循环

1 格法格式：

while(条件表达式)

```

{

```

循环语句

}

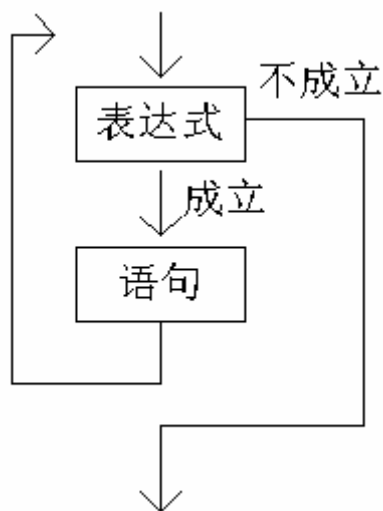


图 3-20

程序的执行过程是这样的：若果表达式的条件成立，则执行循环体语句；若果表达式的条件不成立，则跳过循环体语句继续往下执行。其程序的执行流程就如图 3-20 所示。

例 8

```
char i;
P1=0xfe; //点亮第一只 led 管
i=0; // 初始设定表达式
while(i<8) // 循环条件表达式
{
    P1<<=1; // 每循环一次点亮一只 led 管
    i++; //更新表达式
}
```

利用 while 语句可以实现与 for 语句完全相同的循环功能，就如例 8 所示，其实现与例 7 相同的功能。在执行 while 循环语句之前先让 i 赋予一个初值，在 while 中判断 $i < 8$ 的条件是否成立，当条件成立的情况下则进入循环体；否则就跳过循环体继续往下执行。但是大家有没有注意到，在循环体中有一句 $i++$ 的语句，它是一个更新表达式的语句，其作用就是为退出循环体制造条件。下面是 while 语句与 for 语句相替代的形式。

初始设定表达式

```
while(循环条件表达式)
{
    更新表达式
}
```

3. do_while 循环

2 格法格式:

```
do  
{  
    循环体语句  
}while(条件表达式);
```

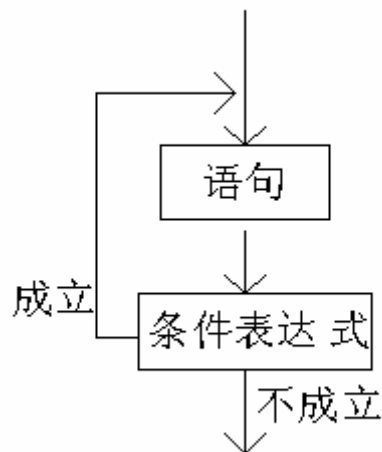


图 3-21

do_while 语句同样可以实现循环功能,但是与 while 循环的不同之处在于:do_while 先执行一次循环体中的语句,然后再判断条件表达式是否成立,若果成立,则继续循环;若果不成立,则退出循环。

所以, do-while 循环至少要执行循环体一次。(如图 3-21)

例 9:

```
i=0;// 初始设定表达式  
  
do  
  
{  
  
    printf("%d\n",i);//从计算机终端输出 5 个数  
  
    i++;//更新表达式  
  
}  
  
while(i<100); 循环条件表达式
```

例 9 中的程序，可以从计算机终端输出变量 i 的值。程序开始时给变量 i 赋一初值，然后执行一次循环语句，再对循环条件表达式进行判断。

动手实验（10）

实验目的：学习 while 循环语句的使用。

实验内容：本实验与前面所演示的 for 循环动手实验（9）实现的功能是完全相同的，但是这个实验则是利用 while 循环来实现。从中读者可以理解到 for 与 while 两个循环语句是完全可以互相代换的，在编写程序时应当灵活运用。

```
#include<reg52.h>//包含单片机对应的头文件
#define uchar unsigned char//宏定义
void delay_ms(unsigned int time)//延时函数
{
    unsigned int i,j;        //声明 i,j 为整型变量
    for(i=time;i>0;i--)      //将变量 time 赋予 i, 当 i 大于 0 的条件成立, 执行循环体,
    然后 i 自减 1
        for(j=112;j>0;j--)  //将变量 112 赋予 j, 当 j 大于 0 的条件成立, 执行循环体,
    然后 j 自减 1
            {;}              //空等待, 直到以上的循环条件结束, 从而实现延时。
}
void main(void)
{
    uchar i,j;                //声明 i,j 为字符型变量
    uchar move_1,move_2;     //声明 move_1, move_2 为字符型变量
    while(1)                  //死循环, 永远不会退出循环体
    {
        i=0;                  //i 赋予 0
        while(i<2)            //假如 i 小于 2 执行循环体语句
        {
            P1=0x7e;delay_ms(200); //点亮 LED0。LED7
            P1=0xbd;delay_ms(200); //点亮 LED1。LED6
            P1=0xdb;delay_ms(200); //点亮 LED2。LED5
            P1=0xe7;delay_ms(200); //点亮 LED3。LED4
            i++;                //i 自加 1, 为退出循环体制造条件
        }
        i=0;                  //i 赋予 0
        while(i<2)            //假如 i 小于 2 执行循环体语句
        {
            P1=0xe7;delay_ms(200); //点亮 LED3。LED4
            P1=0xdb;delay_ms(200); //点亮 LED2。LED5
            P1=0xbd;delay_ms(200); //点亮 LED1。LED6
            P1=0x7e;delay_ms(200); //点亮 LED0。LED7
```

```

        i++;                //i 自加 1，为退出循环体制造条件
    }
    move_1=0x7f;           //move_1 赋值 0x7f
    i=0;                   //i 赋予 0
    while(i<8)             //假如 i 小于 8 执行循环体语句
    {
        P1=move_1;         //move_1 赋于 P1 端口
        move_1=(move_1>>1)|0x80;//move_1 右移一位，然后与 0x80 相或，高位总保持
为 1

        delay_ms(300);     //延时 300ms
        move_2=0xfe;       //move_2 赋值 0xfe
        j=0;               //j 赋予 0
        while(j<8)         //假如 j 小于 8 执行循环体语句
        {
            P1=move_2;     //move_2 赋于 P1 端口
            move_2=(move_2<<1)|0x01;//move_2 左移一位，然后与 0x01 相或，低位总
保持为 1



            delay_ms(100); //延时 300ms
            j++;           //j 自加 1，为退出循环体制造条件
        }
        i++;               //i 自加 1，为退出循环体制造条件
    }
}
}

```

实验步骤：1. 打开光盘第 3 章/ while / while.Uv2 工程文件，对程序进行编译、链接、调试产生 while.hex 烧写文件。

2. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

设置为硬件仿真。正确选择对应的 COM 端口，点击  按钮将波特率设置为 38400。

3. 用  和  进行调试，观察 while 循环语句是如何执行的，在执行 while 循环语句的同时，还要观察变量窗口各变量的变化和实验板的 LED 灯的最终结果，从而理解 while 循环语句的用法。（如图 3-22 所示）

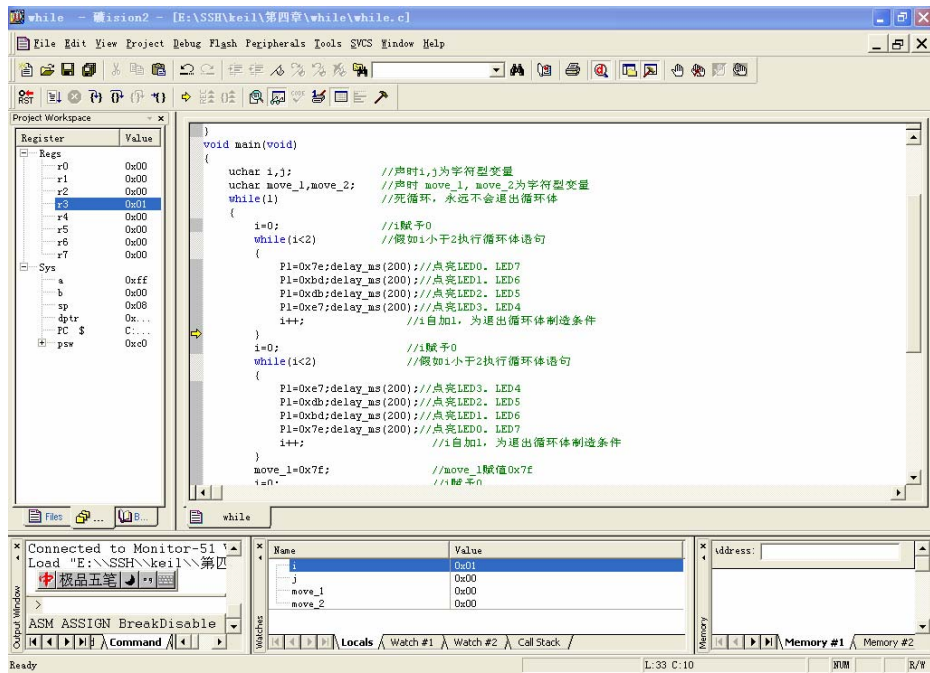


图 3-22

动手实验（11）

实验目的：实验理解 while 与 do—while 的不同。

实验内容：通过以下的实验，区分 while 与 do—while 两个循环语句在使用上有什么不同。以并更加深刻地去了解其结构。

程序一：

```

#include<reg52.h>        //包含单片机对应的头文件
#include<stdio.h>       //包含输入输出函数的头文件
#define uint unsigned int //宏定义
void uart(void)         //串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
void main(void)

```



```
{
uint sum=0,i;      //声明 sum,i 为整型变量, 并对 sum 赋初值 0
uart();          //串口初始化
while(1)
{
printf("please input i value\n");//提示输入 i 的值。
scanf("%d",&i);//从键盘输入 i 的值
while(i<20)      //假如 i 的值小于 20 则执行循环体语句。
{
sum=sum+1;      //sum 的值加 1 再存放于 sum 中
i++;           //i 自加 1, 为退出循环体制造条件
}
printf("%d\n",sum);//输出 sum 的值
sum=0;         //sum 的值清零
}
}
```

实验步骤: 1. 打开光盘第 3 章/ do_while / do_while.Uv2 工程文件, 对程序进行编译、链接、调试产生 do_while.hex 烧写文件。

2. 将串口线把实验板与电脑相接, 打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制的发送与接收、打开串口。把 do_while.hex 文件烧写到 AT89S52 单片机中去, 再安装到实验板中, 通上电源。

3. 分别从串口的发送区输入两个值:

输入: 1 ✓
输出: 19
输入: 25 ✓
输出: 0

如图 3-23 所示



图 3-23

程序二:

```
#include<reg52.h>          //包含单片机对应的头文件
#include<stdio.h>          //包含输入输出函数的头件
#define uint unsigned int //宏定义
void uart(void)           //串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
void main(void)
{
    uint sum=0, i;          //声明 sum, i 为整型变量, 并对 sum 赋初值 0
    uart();                 //串口初始化
    while(1)
    {
        printf("please input i value\n");//提示输入 i 的值。
        scanf("%d",&i);      //从键盘输入 i 的值
        do                  //总是先执行一次循环体
        {
```

```

sum=sum+1;      //sum 的值加 1 再存放于 sum 中
i++;           //i 自加 1，为退出循环体制造条件
}while(i<20);  //然后再判断 i 的值是否小于 20，如果是则执行循环体语句。
printf("%d\n",sum); //输出 sum 的值
sum=0;        //sum 的值清零
    }
}
    
```

实验步骤：1. 打开光盘第 3 章/ do_while_2 / do_while_2.Uv2 工程文件，对程序进行编译、链接、调试产生 do_while_2.hex 烧写文件。

2. 把 do_while_2.hex 文件烧写到 AT89S52 单片机中去，再安装到实验板中，通上电源。

3. 分别从串口的发送区输入两个值：

输入：1 ✓
 输出：19
 输入：25 ✓
 输出：1

如图 3-24 所示



图 3-24

分析结果:

```
while(i<20)                                do
{                                            {
    sum=sum+1;                               sum=sum+1;
    i++;                                     i++;
}                                            }while(i<20);
```

上面的两个小程序是实验中的程序，为了让大家更加容易理解，特别将其写了出来。下面我们就上面作过的实验再来分析。

第一：左边的 while 语句。在第一次输入“1”时，由于 i 的值小于 20，所以执行循环体里面的语句，最后输出的结果为“19”。当第二次输入“25”时，由于 i 的值大于 20，因此没有执行循环体的语句，最后输出的结果为“0”。

第二：右边的 do_while 语句。因为 do_while 语句无论条件如何总是先执行一次循环体的语句再去判断其结果。因此在第一次输入“1”时，由于 i 的值小于 20，最后输出的结果为“19”。但是当第二次输入“25”时，即使 i 的值大于 20，因为它总会执行一次循环体才会去判断其条件，因此最后输出的结果为“1”。

自我练习:

自行编写一个程序，将其命名为“mywhlie”。利用 whlie 循环语句先在串口中发送两个“AT89S52”的字符串，再发送三个“I need learn”的字符串，然后不断周而复始地循环，与上一节 for 循环的练习相同。

程序的设计思路：首先包含相应的头文件，编写一个延时函数和串口初始化函数，然后利用以下这种形式进行循环输出字符串：

初始设定表达式

```
while(条件表达式)
```

```
{
    循环语句
}
```

要实现不断周而复始的循环我们可以用 while(1)这种语法格式，表示条件永远成立。

(以上练习答案附光盘中)

4. goto 循环

语法格式:

goto 语句标号;

例 10

```
i=0;
```

```
LOOP:           //goto 语句的标号
    if(i<100)    //判断 i<100 的条件是否成立
    {
        printf("%d\n",i); //输出 i 的值
        i++;          //更新 i 的值
        goto LOOP;   //goto 循环语句
    }
```

goto 循环语句是一种无条件的循环语句。例 10 就是利用 goto 语句实现了循环的逻辑结构, goto LOOP 语句中的 LOOP 就是一个语句标号, 也是一个有效的标识符。这个标识符加上一个“:”构成完整的循环结构, 当程序执行 goto 语句后, 将跳转到该标号处并执行其后的语句。大家要注意一点, goto 循环语句只能在同一个函数内进行跳转, 不能从一个函数无条件地跳到另一个函数。些循环语句经常与 if 语句一起使用。当程序使用了 goto 语句之后会使程序的可读性大大降低, 所以在实际应用中建议大家少用。

5. break 和 continue 语句

1 break 语句

在前面讨论 switch 语句的时候, 已经讲过 break 可以退出 switch 语句, 如果没有 break 语句, 那么 switch 语句就永远地循环下去。而在循环控制语句当中, break 起到提前结束循环的作用。在循环语句中, break 语句常与 if 语句结合使用。我们来看下面的例子。

例 11

```
for(i=0;i<100;i++)
{
    if(i==50)
        break;
    printf("%d\n",i);
}
```

例 11 的程序当中本来 printf 应该输出 100 个值, 但是里面出现了一个 if 语句, 意思是: 假如 i 的值为 50 的条件成立, 则退出循环体, 那就提前结束了循环。在使用 break 前要注意两点: 第 1、break 在 else if 语句中不起作用; 第 2、当有多层循环语句嵌套的时候, break 语句只退出本层的循环, 如例 12 所示。

例 12

```
for(i=0;i<100;i++)
{
    for(y=0;y<100;y++)
    {
        if(y==50)
            break;
        printf("%d\n",y);
    }
}
```

```
}
```

上面的程序就是这样，当 y 的值为 50 的条件成立时，break 语句只可以退出本层的循环，最外层的循环不会因为 break 语句而提前退出，除非当 i<100 的条件不成立，否则会继续循环。

2 continue 语句

continue 语句只用在循环控制语句当中，其作用是跳过本次的循环，继续进行下一次的循环。continue 语句同样常与 if 语句相结合使用。

例 13

```
for(i=0;i<10;i++)
{
    if(i%2==0)
        continue;
    printf("%d\n",i);
}
```

例 13 的程序当中本来应该进行 10 次循环，从而输出从 0~9 的 10 个数。但是程序中出现了一句“continue;”，那么什么时候执行“continue;”语句啊！就是当 i 除以 2 的余数为 0 时，则跳过本次的循环。那么最后输出的值为 1、3、5、7、9。

动手实验（12）

实验目的：学习 break 语句的使用。

实验内容。在 while 循环体语句当中不断地使 LED0 以 100ms 闪动，但是当 K0 被按下之后则退出循环体，点亮 P1 口的全部 LED 灯 500ms 再熄灭。

```
#include<reg52.h>//头文件包含
#define uchar unsigned char
sbit LED=P1^0;//定义 P1.0 引脚进行位操
sbit K0=P3^2;//定义 P3.2 引脚进行位操
void delay_ms(unsigned int time)//延时函数
{
    unsigned int i,j;
    for(i=time;i>0;i--)
        for(j=112;j>0;j--)
            {;}
}
void main(void)
{
```

```
while(1)
{
    LED=~LED;//LED 不断按目前的状态取反
    delay_ms(100);//延时 100ms
    if(K0==0)//假如 k0 被按下了
        break;//退出当前循环体
}
P1=0x00;//点亮 P1 端口的 8 只二极管
delay_ms(500);//延时 500ms
P1=0xff;//熄灭 P1 端口的 8 只二极管
}
```

实验步骤：1、打开光盘第 3 章/ break/ break.Uv2 工程文件，对程序进行编译、链接、调试产生 break.hex 烧写文件。

2、把 51 仿真器插到实验板中，另一端与电脑相接，将相应的参数设置好。通上实验板电源。

3、分别按单步执行与全速执行进行调试，同时在实验板中观察其运行的结果。

实验分析：在这个程序当中，正常情况下程序是不会退出 while 语句的，因为 while 的判断条件为 1，表示条件永远成立，但是如果 K0 被按下了则执行 break 语句，从而退出循环体。也就是说：当执行 break 语句时，是无条件退出循环体的。

动手实验（13）：

实验目的：学习 continue 语句的使用。

实验内容。下面的程序是从串口调试器中输入一个值“n”，假如从“0”到“n”的数值除以 3 的结果等于 0 则跳过，否则输出其值。

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>//包含标准输入输出函数头文件
#define uint unsigned int
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
}
```

```
        TR1=1;
    }
    void main(void)
    {
        uint n;
        uint i;
        uart();//串口初始化
        while(1)
        {
            printf("please input number\n");//提示输入数值
            scanf("%d",&n);//从键盘输入一个数值
            for(i=0;i<n;i++)//i 赋值 0 当 i 小于 n 的条件成立时执行循环体然后 i 自加 1
            {
                if(i%3==0)//假如 i 除以 3 的值为 0，则跳过。
                    continue;
                printf("%d\n",i);//否则输出 i 的值。
            }
        }
    }
}
```

- 实验步骤：1. 打开光盘第 3 章/ continue / continue.Uv2 工程文件，对程序进行编译、链接、调试产生 continue.hex 烧写文件。
2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制发送与接收、打开串口。把 do_while.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
3. 从串口中输入一个值（设为 10），先按回车键再按手工发送。则单片机会将 1、2、4、5、7、8 这几个数发送到串口的接收区。如图 3-25

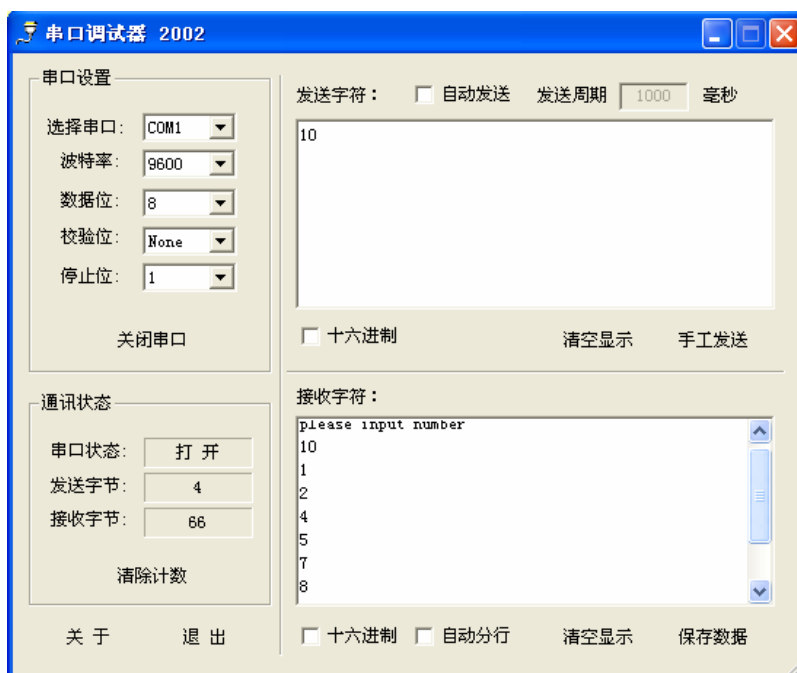


图 3-25

实验分析：从串口中输入一个数值是 10，最后从单片机发出来的数据只有 1、2、4、5、7、8 因为 0、3、6、9 这四个数除以 3 的值都为 0，所以被跳过了。

自我练习：

利用 break 语句自行编写一个程序，将其命名为“mybreak”。假如 K0 被按下了则点亮高 4 位 LED 管 500ms，否则不停地对低 4 位的 LED 以 100ms 延时时间进行闪烁。

程序的设计思路：此练习与前面的动手实验（12）的算法是一样的，读者可以参考从而来引导自己的思路。

（练习答案附光盘中）

第七节 数组

什么是数组？C 语言规定把具有相同数据类型的若干变量按有序的形式组织起来称为数组。在数组中的每一个成员称为数组元素。其中按数组元素类型的不同可以分为数值数组、字符数组、指针数组、结构数组。本章只要介绍数值数组与字符数组，至于指针数组与结构数组将在后面分别介绍。其中数值数组又可分为：一维数组，二维数组。下面我们将分别介绍。

一、一维数组

1. 一维数组的定义

语法格式：

类型说明符 数组名[常量表达式]

例 1: `int array[5];`

`int` 为类型说明符、`array` 为数组名、`5` 为常量表达式。那么例 1 的意思就是：定义了一个名为 `array`，数据类型为 `int` 的数组，其中数组含有 5 个元素，分别为 `array[0]`、`array[1]`、`array[2]`、`array[3]`、`array[4]`，而每一个数组元素的类型都为 `int` 型。注意，数组的元素是从 0 开始的，而不是从 1 开始的，即第 5 个元素为 `array[4]` 而不是 `array[5]`；而且数组名不能与变量名相同。

2. 一维数组的初始化

所谓初始化，就是在定义数组的同时给数组的元素赋予初值。下面是几种初始化的方式。

第一种方式：`int array[5]={1, 2, 3, 4, 5};`

在定义数组的同时并赋予初值。在花括号 `{}` 里面的数值就是元素 `array[0]`、`array[1]`、`array[2]`、`array[3]`、`array[4]` 的值。即 `array[0]=1`、`array[1]=2`、`array[2]=3`、`array[3]=4`、`array[4]=5`。

第二种方式：`int array[5]={1, 2};`

在花括号里只给需要的元素赋初值，而未被赋初值的元素在编译时由系统自动赋予“0”为初值。即 `array[0]=1`、`array[1]=2`、`array[2]=0`、`array[3]=0`、`array[4]=0`。

第三种方式：`int array[]={1, 2, 3, 4, 5};`

如果给每个元素都赋予了初值，那么在数组名中可以不给出数组元素的个数。上面的写法就等价于“`int array[5]={1, 2, 3, 4, 5};`”。

3. 一维数组的引用

例 2:

```
int array[5]={1, 2, 3, 4, 5}; //定义一个数组并初始化
char i;
for(i=0; i<5; i++)
{
    printf("%d\n", array[i]*2); //将数组每一个元素的值乘以 2 再输出。
}
```

引用数组元素时是通过数组名+[]+元素在数组中的位置来进行的。例 2 中的输出结果为 2、4、6、8、10。

二、二维数组

1. 语法格式:

类型说明符 数组名[常量表达式 1] [常量表达式 2]

在上面的语法格式当中常量表达式 1 表示第一维下标的长度, 常量表达式 2 表示第二维下标的长度。

例 3:

```
int array [3][4];
```

定义了一个 3 行 4 列, 数组名为 array, 类型为 int 的数组。该数组的下标变量共有 3×4 个, 即:

```
array [0][0], array [0][1], array [0][2], array [0][3]
```

```
array [1][0], array [1][1], array [1][2], array [1][3]
```

```
array [2][0], array [2][1], array [2][2], array [2][3]
```

二维数组在概念上是二维的, 即是说其下标在两个方向上变化, 下标变量在数组中的位置也处于一个平面之中, 而不是象一维数组只是一个向量。但是, 实际的硬件存储器却是连续编址的, 也就是说存储器单元是按一维线性排列的。其实在一维存储器中存放二维数组是按行排列, 即放完一行之后顺次放入第二行。如上面定义的二维数组即: 先存放 array[0] 行, 再存放 array[1] 行, 最后存放 array[2] 行。每行中有 4 个元素, 也是依次存放。因为数组 array 定义为 int 类型, int 为双字节的数据类型, 所以每个元素在内存中占两个字节的空間。

2. 二维数组的初始化

二维数组的初始化与一维数组的初始化是大同小异的, 只要掌握了一维数组, 那么二维数也很容易理解。

第一种方式: `int array [3][4]={{1, 2, 3, 4}, {5, 6, 7, 8, }, {9, 10, 11, 12}};`

在定义数组的同时并赋予初值。全部元素的初值括在一个花括号中, 其中每一行的元素又用一个花括号括起来, 其中间用 “, ” 分开。

第二种方式: `int array [3][4]={{1, 2}, {5, 6}};`

在花括号只给需要的元素赋初值, 而未被赋初值的元素在编译时由系统自动赋予 “0” 为初值。即 `array[0][0]=1`、`array[0][1]=2`、`array[1][0]=5`、`array[1][1]=6`, 其余未被赋值的全部为 “0”。相当于以下的赋值: `int array [3][4]={{1, 2, 0, 0}, {5, 6, 0, 0, }, {0, 0, 0, 0}};`

第三种方式: `int array[] [4]={{1, 2, 3}, {5, 6, 7, 8, }, {9, 10}}`

在一维数组中, 如果给每个元素都赋予了初值, 那么在数组名中可以不给出数组元素的个数。但是在二维数组中就只能省略行的个数, 而不能省略列的个数。那么上面的定义方式经系统编译之后得到的结果是: `int array[3] [4]={{1, 2, 3, 0}, {5, 6, 7, 8, }, {9, 10, 0, 0}};`

3. 一维数组的引用

例 4

```
int array [3][4]={{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}; //定义一个二维数组;
char i, j;
for(i=0; i<3; i++)
{
    for(j=0; j<4; j++)
    {
        printf("%d\n", array[i][j]); //输出二维数组的每一个元素
    }
}
```

二维数组的引用方法同一维数组的引用方法基本相同的，例 4 中就是定义了一个二维数组，然后用两个 for 循环语句将数组的每一个元素输出到计算机的终端。

三、字符数组

1. 语法格式

字符数组在使用上分为字符数组与字符串数组，其定义方式与前面讲的两种数值数组的定义方式是相同的。

例 5

```
char ch[4]; //字符数组或字符串数组的定义
```

字符串数组与字符数组的定义是相同的。例 5 定义了一个数组名为 ch，而类型为 char 的字符数组（或字符串数组），数组有 4 个元素；其次也可以用 int 数据类型来定义，如果用 int 数据类型来定义的话，那么字符常量就会占用两个字节。

2. 字符数组与字符串数组的初始化

```
char ch[10]={ 'c' , 'h' , 'i' , 'n' , 'a' } ; //字符数组初始化
```

```
char ch[10]= { "china" } ; //字符串数组初始化
```

字符数组与字符串数组的初始化方式同样可以在定义的同时进行，如上面所示。但是未被定义的元素会被系统编译的时候赋予空格字符“\0”。

字符与字符串的区别：用单引号 ‘ ’ 括起来的字符为字符的 ASCII 码值，而不是字符串。比如 ‘c’ 表示 c 的 ASCII 码值 99；而 “c” 表示一个字符串，由两个字符 c 和 \0 组成。

其中字符串同样可以用另外一种方式定义：

```
char ch[4]= "good" ; //这种定义的方式同样是合法的。
```

3. 字符数组的引用

例 7

```
char ch[10]; //定义一字符数组
while(1)
{
    scanf("%s", ch); //从键盘输入字符串于 ch 字符数组中。
    printf("%s\n", ch); //输出 ch 字符数组的元素。
}
```

动手实验（14）

实验目的：学习数组的使用。

实验内容：在程序当中声明一个整型数组并对其初始化为 10 个不相等的值。当 K2 被按下后在串口调试器中输出最大值。在这个实验中读者要学会如何声明一个数组，对其进行初始化，还要学会在程序当中如何对每一个元素进行引用。

```
#include<reg52.h> //包含 52 的头文件
#include<stdio.h> //包含标准输入输出头函数文件
#define uint unsigned int
sbit K2=P3^4; //定义位操作
void delay_ms(unsigned int time) //延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time; i>0; i--) //i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112; j>0; j--) //j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void uart(void) //串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
void main(void)
{
```

```
uint array[10]={89, 50, 784, 54, 33, 698, 2, 15, 4, 36}; //声明一个无符号整
型数组并初始化
uint i;
uint most;
uart(); //串口初始化
while(1)
{
    for(i=0;i<10;i++) //利用循环语句输出这个数组的 10 个元素
    {
        printf("%d ", array[i]);
    }
    printf("\n"); //把光标移向下一行
    printf("please press K2 to the work\n"); //提示按 K2 进行操作
    while(K2!=0); //等待 K2 被按下
    delay_ms(30); //延时消抖
    if(K2==0)
    {
        while(K2==0); //等待 K2 松开
    }
    /*以上这四句语句为对按键 K2 的操作，关于按键的操作在以后的章节会讨
论到*/
    most=array[0]; //把数组的第 0 号元素赋予 most
    for(i=0;i<10;i++) //循环 10 次，比较元素的值
    {
        if(array[i]>most) //假如 array 的第 i 号元素比 most 大
            most=array[i]; //把 array 的第 i 号元素赋予 most
    }
    printf("The most is %d\n", most); //输出最大值
}
}
```

- 实验步骤:
1. 打开光盘第 3 章/ array / array.Uv2 工程文件，对程序进行编译、链接、调试产生 array.hex 烧写文件。
 2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制发送与接收、打开串口。把 array.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
 3. 单片机会发出 10 个数值到串口的接收区，同时还提示请按下 K2 进行操作。当按下 K2 时单片机就会把最大的数值发送到串口的接收区如下图 3-26 所示。

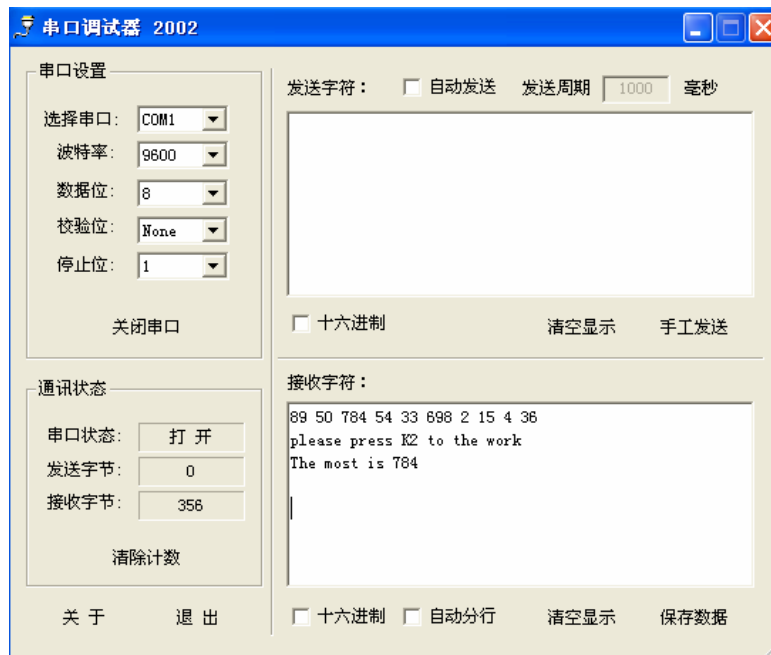


图 3-26

动手实验（15）：

实验目的：学习字符数组的使用。

实验内容：在程序当中声明一个字符数组并对其进行初始化。当 K2 被按下之后，在串口调试器输出这个字符数组的每一个元素，从而在接收区显示“AT89S52”。在这个实验过程中，我们要学会如何声明一个字符数组并对其进行初始化，同时还要掌握在程序的算法当中如何对字符数组元素的引用。

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>//包含标准输入输出头函数文件
#define uint unsigned int
sbit K2=P3^4;
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {;}
}
void uart(void)//串行口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
```

```
TMOD=0x20;
TH1=0xfd;
TL1=0xfd;
TI=1;
TR1=1;
}
void main(void)
{
    //声明一个字符型数组并初始化
    unsigned char array[7]={'A','T','8','9','S','5','2'};
    uint i;
    uart();//串口初始化
    while(1)
    {

        printf("please press K2 output the char\n");//提示按下K2 输出字符
        while(K2!=0);          //等待 k2 被按下
        delay_ms(30);
        if(K2==0)
        {
            while(K2==0);
/*以上这四句语句为对按键 K2 的操作，关于按键的操作在以后的节章会讨论到*/
            for(i=0;i<7;i++)
            {
                printf("%c",array[i]);//利用 for 循环输出字符
            }
            printf("\n");          //把光标指向下一行
        }
    }
}
```

- 实验步骤：
1. 打开光盘第 3 章/ char_array / char_array.Uv2 工程文件，对程序进行编译、链接、调试产生 char_array.hex 烧写文件。
 2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制发送与接收、打开串口。把 char_array.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
 3. 单片机会发出信息提示按下 K2 进行操作。当按下 K2 时串口调试器就会接收到“AT89S52”的字符，如下图 3-27 所示。

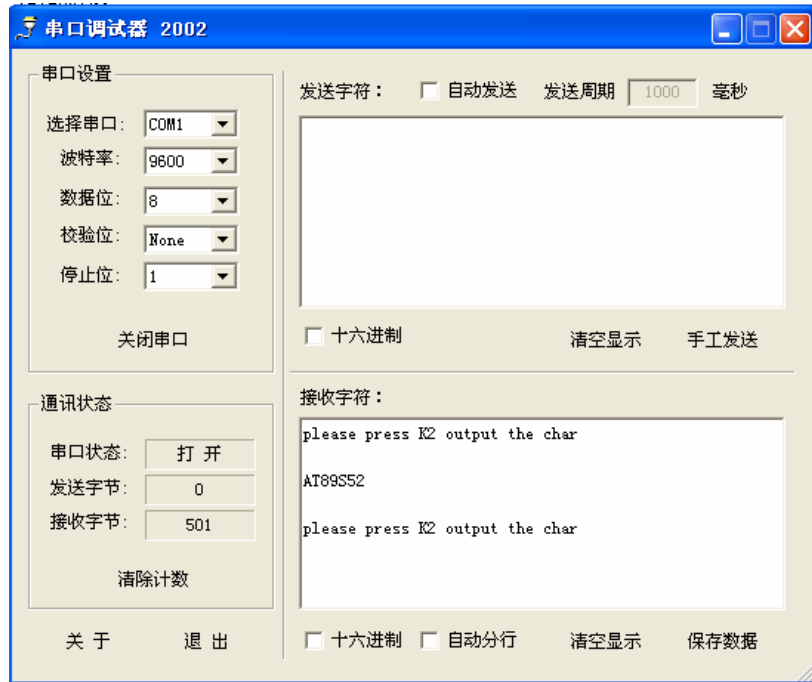


图 3-27

自我练习:

声明一个带有三个元素的整型数组，从键盘任意输入三个值存放于此数组中，利用 for 循环语句找出最大值从串口调试器中输出。工程名命名为“myarray”。

程序的设计思路：首先我们可以声明一个数组 array，再利用循环语句以 `scanf("%d",&array[i]);` 这种形式输入三个值，然后再在循环语句当中利用 if 语句比较数组每一个元素的大小最后输出最大值。当然，单片机的头文件和串口的初始化程序也是必不可少的。

第八节 函数

本节我们来讨论关于函数。一个实用的 C 语言程序总是由多个不同的函数来构成，其中每一个函数总扮演不同功能的角色。但是，无论一个 C 语言程序当中有多少个函数，程序总是从 main 函数开始执行起。而在 C 语言当中函数可以分为两类：一类为库函数（又称为标准库函数）；而另一类为用户自定义函数。

标准库函数：

Keil 编译器提供了丰富的库函数，库函数无需程序员自行去定义，只要包含相应的头文件即可任意地调用。在前面的节章当中我们也反复多次用到了系统库函数，例如前面用到的 printf、scanf 等等。在调用库函数之前，必须利用 include 命令包含相应的头文件。例如，在程序中用到 printf 或 scanf 函数时，则应用 include 命令包含 stdio.h 这个文件。下面是合法的书写格式：

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

include 命令必须以#include<头文件>这种形式进行，大家要注意 include 命令的后面不能加一个分号“；”，因为它不是一个 C 程序语句。打开 stdio.h 这个文件就可以看到 printf、scanf 这两个函数的原型，如下所示。

```
extern int printf (const char *, ...);
```

```
extern int scanf (const char *, ...);
```

用户自定义函数：

从函数定义的角度来看，大致可分为两类：

- 1) 无参数函数：在定义此类函数时是不带任何参数的，主调用函数和被调用函数之间不进行参数传送，调用此类函数目的只为完成某项特定的功能。
- 2) 有参数函数：在函数定义的同时并带有参数。主调用函数和被调用函数之间进行参数传送。当执行完被调用函数之后返回一个参数供主调用函数使用。

大家应当记住，在 C 语言中，所有的函数定义，包括主函数 main 在内，都是平行的。即在一个函数体里不能再重复定义一个函数。函数与函数之间可以互相调用，而且还可自己调用自己这种调用的方式称为递归调用。在任何一个 C 语言程序当中，都必须有一个 main 函数，而且也只能有一个 main 函数。main 函数称为主函数，它可以调用其它函数，而不能被其它函数调用。程序的执行是从主函数开始而最后也是从主函数程序结束。

一. 函数的定义

标准库函数是系统设计者事先已经将其存放在函数库当中，只要了解函数的头文件与函数的功能就可以直接调用它，而且库函数是面对所有用户的，不能满足每一个用户的特殊功能要求，所以在本节当中，我们只要讨论的是用户自定义函数。

无参数函数定义的语法格式：

```
返回类型标识符 函数名 ()  
{  
    声明部分  
    程序语句  
}
```

返回类型标识符是当函数被调用之后所返回的数据类型，这与前面所讲术的数据类型是相同的。但是如果不需要返回任可的数值也可以写为“void”，表示无类型数据。

函数名是由用户自定义的，而后面必须跟一个“（）”，如果是无参数函数多般在括号内边写入“void”表明是无参数的传递。

返回类型标识符与函数名称为函数头，函数头的下方必定是跟一个“{ }”。

“{}”内面就是**声明部分**与**程序语句**，大家要注意，声明部分指的是在当前函数体内所用的一些变量，声明部分一定要在程序语句之前，否则就会出现语法错误。

例 1

```
void delay(void)           //函数头
{
    unsigned int i, j;     //声明部分
    for(i=100;i>0;i--)    //程序语句
        for(j=112;j>0;j--)
            {;}
}
```

例 1 就是一个无参数函数的定义。其功能是以单片机AT89S52 用 12MHz晶振频率每调用一次可延时 100ms。

有参数函数定义的语法格式：

返回类型标识符 函数名(形式参数)

```
{
    声明部分
    程序语句
}
```

有参数函数定义与无参数函数的定义大致是相同的，区别在于无参数函数由于是没有数据的传递，所以在“（）”里可以停入关键字“void”。但是有参数函数定义因为是有数据的传递那么括号“（）”里应为数据的类型与所声明的变量，简称形式参数。其实返回类型标识符与形式参数的理解可以是一致的，只不过形式参数就是主调用函数传递给被调用函数的数值，而返回类型标识符就是被调用函数传递给主调用函数的数值。这是一种更为人性化的理解。下面我们以一个具体的例子来讨论。

例 2

```
int compare(int a, int b)//函数头，括号里为形式参数
{
    if(a>b)                //以下是程序语句
        return a;
    else
        return b;
}
```

例 2 就是一个有参数的函数，其中返回类型标识符与形式参数的数据类型都是 int 类型，而函数名为“compare”。功能就是比较 a 的值与 b 的值的的大小，然后返回一个最大值。关于如何调用函数与函数的返回值我们将在后续的节章中详细讨论到。

动手实验（16）

实验目的：学习如何声明一个函数

实验内容：下面的程序先是同时点亮 P1 口的 8 只 LED 管延时 500ms 再熄灭，然后再调用 move_bit 函数进行左移，逐位点亮二极管。注意理解 move_bit 函数是如何声明的，在仿真时进行单步调试，了解程序每一步是如何执行的。

```
#include<reg52.h>//包含 52 的头文件
#define uchar unsigned char
void delay_ms(unsigned int time)//延时函数
{
    unsigned int i, j;
    for(i=time;i>0;i--)
        for(j=112;j>0;j--)
            {;}
}
void move_bit(void)//移位函数
{
    uchar i;
    P1=0xfe;          //P1 端口赋值 0xfe
    delay_ms(100);    //延时 100ms
    for(i=0;i<8;i++)  //i 赋值 0，假如 i 小于 8 的条件成立，则执行循环体语句，i
    再自加 1
    {
        P1=(P1<<1)|0x01;//读 P1 口的值左移 1 位再与 0x01 相或保持低位为 1
        delay_ms(100); //延时 100ms
    }
}
void main(void)
{
    while(1)
    {
        P1=0x00;      //点亮 P1 口
        delay_ms(500); //延时 500ms
        P1=0xff;      //熄灭 P1 口
        move_bit();   //调用 move_bit 函数
    }
}
```

实验步骤：1、打开光盘第 3 章/ function/ function.Uv2 工程文件，对程序进行编译、链接、调试产生 function.hex 烧写文件。

2、把 51 仿真器插到实验板中另一端与电脑相接，将相应的参数设置好。再把实板通上电源。

3、分别按单步执行与全速执行进行调试，观察变量窗口与实验板 LED 的变化。（如图 3-28 所示）

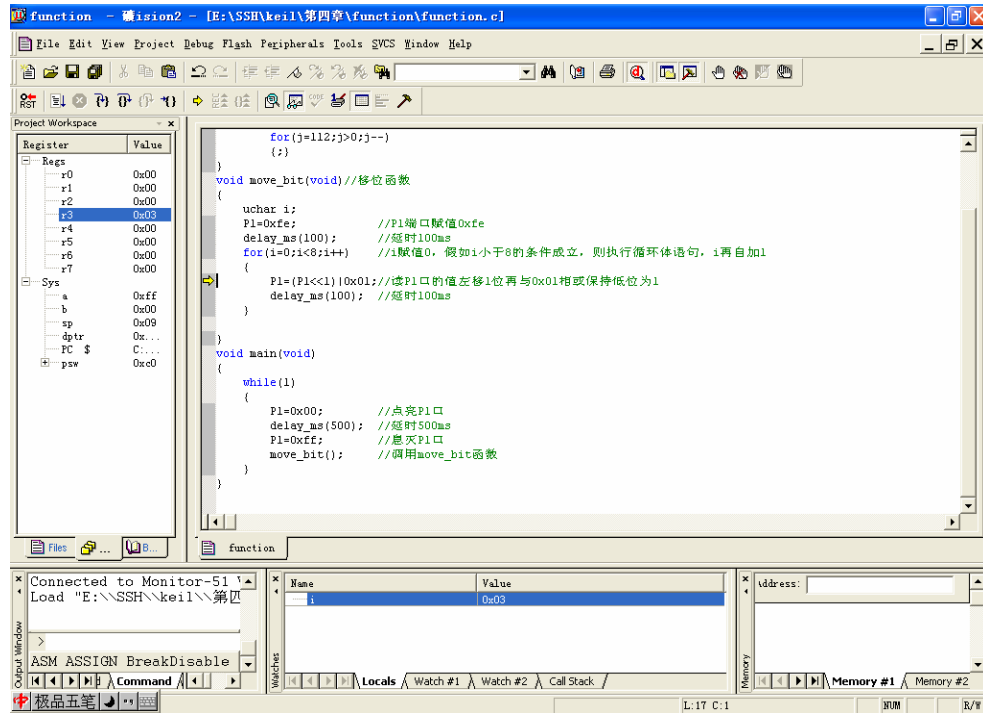


图 3-28

自我练习:

编写两个函数，一个是将 P1 口的 LED7 向 LED0 逐位增加点亮，另一个是将 P1 口的 LED7 向 LED0 逐位减小点亮，每点亮一只 LED 管延时 300ms。然后在 main 函数中调用这两个函数。将工程命名为“myfunction”。

程序的设计思路：（1）先写好头文件和延时函数。

（2）可以参考上面的动手实验（16）编写函数，在函数里声明一个有符号字符型变量将其赋初值 0x7f，因为我们在前面讨论过，当有符号数右移时要保持符号位的不变，所以利用“>>”就可以逐位增加点亮。

（3）用同样的方法在函数里声明一个有符号字符型变量将其赋初值 0x80，右移则可以逐位减小点亮 P1 口的 LED 灯。

（4）在 main 函数中用“while（1）”死循环语句不断调用这两个函数就可以了。

(练习答案附光盘中)

二. 函数的参数与返回值

函数的参数

在C语言当中函数的调用是经常用到的，主调用函数与被调用函数之间有数据传递的关系。数据的传递是双方面的，一定要有一个发送者与一个接收者才能实现数据的传送。其中实参就是扮演了发送者的角色，实参出现在主调用函数中，离开了主调用函数实参就不能使用；而形参就扮演了接收者的角色，形参要在被调用函数内定义，此定义的形参只在该函数中有效，离开了该函数则不能使用。下图 3-29 恰好可以体现在函数调用两者之间的关系。其中 x 为主调用函数的实参， y 为被调用函数的形参。首先，当主调用函数调用被调用函数的时候，将实参 x 的值传送给形参 y 。而当被调用函数被主调用函数调用完之后，因为实参与形参之间数据的传递是单向的，那么即使形参 y 的值发生了变化而实参 x 的值是不会改变的。

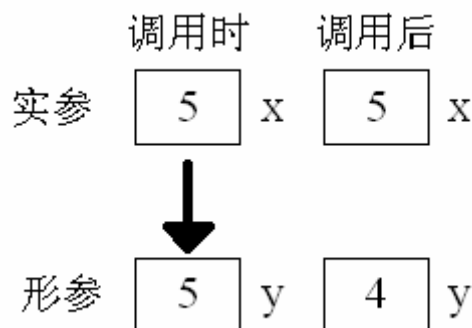


图 3-29

例 1:

```
int compare(int y)
{
    y--;          //自减 1
    printf("%d\n", y); //输出 y 的值
}

void main(void)
{
    int x;
    while(1)
    {
        scanf("%d",&x); //从键盘输入 x 的值
        compare(x);     //将实参 x 传送给形参 y
        printf("%d\n", x); // 输出 y 的值
    }
}
```

上面例 1 是一个较为简单的函数调用例子，但是正好体现了函数调用的概念。现在我们以图 3-29 为例结合上面的程序来讨论函数参数的传递过程。首先从主函数中声明一个变量，利用 `scanf` 函数从键盘输入一个值赋予 x ，假设从键盘输入的值为 5。然后在主函数中调用 `compare` 函数将实参 x 的值传送给形参 y ，那么此时形参 y 的值也为 5，进入了 `compare` 函数之后取 y 的值进行运算， y 的值自减 1，然后利用 `printf` 函数输出 y 的值，此时 y 的值应为 4。退出了 `compare` 函数之后返回到 `main` 函数，再执行 `printf` 函数输出 x 的值，此时 x 的值应为 5。从而可见两者之间的数据传送是单向的，实参的值不会因为形参的值改变而改变。

对于实参与形参的特点再有以下的几点补充:

- (1) 实参与形参在、类型、数量、顺序应保持一致,否则会在编译的时候出现警告或者程序运行的结果错误。
- (2) 被调函数的形参只有被调用的时候才会被分配内存空间,退出了函数之后,所分配的内存单元立即被释放。所以退出了函数之后形参就不能再使用。
- (3) 实参在调用前一定要有确定的值,因此在函数调用前必须先赋予实参一个确定的值。

函数的返回值

函数的返回值同前面讲的函数参数我们可以看作是同一个概念,只是传送的方向对调。由被调用函数扮演发送者,而主调用函数则扮演接收者。

语法格式:

`return` 表达式;

```
return a;
```

或者

```
return i+j;
```

以上的返回语句都是合法的,下面我们来看一个具体的例子。

例 2:

```
int account(int i,int j)
{
    int y;
    y=(i+2)*78+j;
    return y;
}
```

例 2 是一个带返回函数值的程序,关于函数值的返回大家应注意以下几点:

- (1) 函数的数据类型应该与所返回的表达式类型相一致,否则会自动进行类型转换。
- (2) 在函数中可以有多个 `return` 返回语句,但是每次调用只能有一个 `return` 语句被执行,因此每次调用只能返回一个值给主调用函数。
- (3) 一个函数允许没有 `return` 返回语句,那么当程序执行到函数最后一个“}”,则返回一个不确定的值。因此建议大家在一个函数不需要返回函数值的情况下,在返回数据类型里填写为“void”。

三、函数的调用

1. 函数调用的语法格式:

函数名（实际参数表）；

其中实际参数可以是变量、常量表达式，或函数等；但是如果被调用函数是无参数的，那么可以省略实际参数表，具体语法格式如下：

函数名（）；

下面来讨论几种常用的调用方式：

第 1 种，函数语句。当在函数的后面加上“；”就成为了函数语句：

```
例如：account（）；                    //调用无参数函数；
```

或者

```
printf(“%d\n”,result);    //调用有参数函数；
```

第 2 种，调用一个函数，使其返回一个函数值来参加某种特定的运算。这种情况下被调用函数一定要包含 return 返回语句。

```
例如：i=6*key(x,y)；
```

上面程序语句的意思就是调用 key 函数，然后返回一个函数值再与 6 进行运算，将运算的结果赋予变量 i。

第 3 种，将函数调用语句作为另一个函数的实参。

```
例如：i=show(a, count(x,y))；
```

先调用 count 函数返回的值再将其作为 show 函数的实参进行调用，最后的结果赋予变量 i。

前面介绍了函数调用的语法格式和几种常用的调用方式，但是在一个函数调用另一个函数之前，必须满足以下的条件：

- (1) 所调用的函数必须是已经被定义的函数。
- (2) 如果所调用的函数为一个库函数或者不在同一个文件的函数，那么一定要利用 #include 命令进行文件包含，把相应的头文件包含到当前文件来。在系统编译的时候就会把头文件的函数调到原程序当中从而产生代码。

```
例如：#include<stdio.h>//包含标准输入输出的头文件
```

```
#include<reg52.h>//包含 AT89S52 单片机寄存器的头文件
```


(3) 关于函数的原型。如果主调用函数定义在被调用函数之前，那么在主调用函数调用被调用函数之前应作函数原型的声明：

语句格式：

类型说明符 被调函数名(类型 形参, 类型 形参);

其实函数原型的声明非常简单，只是在原来定义函数的基础上在最后加一个“;”。如下面的例子所示，下面程序的第一，二行就是对函数 max, time 作出了声明。

例如：

```
char max(int a); //函数原型声明
char time(int b); //函数原型声明
void main(void)
{
}
char max(int a)
{
    .....
}
char time(int b)
{
    .....
}
```

但是如果主调用函数定义在被调用函数之后，那么在主调用函数调用被调用函数之前则不需要作函数原型的声明，因为 C 编译器在编译主调用函数之前，已经预先知道被调用函数的存在，并自行加以处理。

例如：

```
char max(int a)
{
    .....
}
char time(int b)
{
    .....
}
void main(void)
{
}
```

动手实验（17）：

实验目的：学习函数实参与形参之间数据的传递。

实验内容：以下实验的程序是从串口调试器中输入两个不相等值，利用函数调用的方式把数据传送到被调用函数中去，然后判断出其中的大者，再返回到主调用函数从串口输出。我们在这个实验当中要掌握函数是如何利用实参把数据传递到形参，再利用何种方式把被调用函数的值返回到主调用函数。

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>//包含标准输入输出函数头文件
#define uint unsigned int
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
uint compare(uint a,uint b)//此函数功能为判断两个数的大小
{
    if(a>b)//假如 a 大于 b
        return a;//返回 a 的值
    else
        return b;//否则返回 b 的值
}
void main(void)
{
    uint x,y;
    uart();//串口初始化
    while(1)
    {
        printf("please input two number\n");//提示输入两个数
        scanf("%d%d",&x,&y);//从键盘输入两个数
        //调用 compare 函数，将实参 x y 传送给形参 a b, 然后利用 printf 函数输出其返回值
        printf("the most number is %d\n",compare(x,y));
    }
}
```

- 实验步骤：1. 打开光盘第 3 章/ fun_compare /fun_compare.Uv2 工程文件，对程序进行编译、链接、调试产生 fun_compare.hex 烧写文件。
2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制的发送与接收、打开串口。把 fun_compare.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
3. 单片机会提示输入两个数。然后我们分别在串口的发送区输入两个不相等的数。当输入了两个数之后单片机就会把其中的大数输出到串口的接收区。如下图 3-30 所示。



图 3-30

自我练习：

参考上面的动手实验，从串口中输入两个值利用函数调用的方式把数据传递到被调用函数中去，然后将两个数值相加再利用函数值返回的形式在串口调试器中输出。工程名命名为“tun_transfer”。

程序的设计思路：

- (1) 首先在程序的开始要进行头文件的包含和在主函数中调用串口初始化函数。
- (2) 然后定义一个函数：返回类型标识符 函数名(形式参数)在函数里将两个值相加，利用 return 语句把值返回到主调用函数中。

- (3) 在主函数中声明两个变量，利用 scanf 函数中输入两个值放到这两个变量当中，然后利用函数调用的方式把实参传递到前面所定义的函数中去，再把返回值从 printf 函数中输出。

(练习答案附光盘中)

函数的自我调用——递归

一个函数在它的函数体内调用它自身称为递归调用，递归调用也可以理解为再入一个函数，其意思是在函数体内直接或间接调用其自身的一种函数。主调用函数同时又是被调用函数，递归调用是反复地调用其本身。而在 keil C 中利用关键字 reentrant 来定义再入一个函数的，下面是合法的语法格式：

返回类型标识符 函数名(形式参数) reentrant

在分析递归程序算法的结构之前，我们先来了解一下什么是阶乘。阶乘是指从 1 乘以 2 乘以 3 乘以 4 一直乘到所要求的数。例如所要求的数是 4，则阶乘就是 $1 \times 2 \times 3 \times 4$ ，得到的积是 24，24 就是 4 的阶乘。例如所要求的数是 6，则阶乘就是 $1 \times 2 \times 3 \times \dots \times 6$ ，得到的积是 720，720 就是 6 的阶乘。例如所要求的数是 n，则阶乘是 $1 \times 2 \times 3 \times \dots \times n$ 得到的积是 x，x 就是 n 的阶乘。

例如：

```
int fac(int n) reentrant//定义一个递归函数
```

```
{
    int result;
    if(n==0||n==1)
        result=1;           //假如这个是“0”或“1”，则结果为1。
    else if(n<0)
        printf("n<0 you input error");//假如这个数是小于“0”，此数无阶乘意义。
    else
        /*假如这个数是大于“0”的整数，则将其与相邻较小的数的阶乘相乘*/
        result=n*fac(n-1);
    return(result);//返回结果
}
void main(void)
{
    int n;
    int result;
    while(1)
    {
        printf("\ninput a inteager number:\n");//提示输入一个数
        scanf("%d",&n);           //从键盘输入一个数
        result=fac(n);           //调用阶乘函数
        printf("factorial=%d",result); //输出阶乘结果
    }
}
```

下面我们来分析上面的一个例程：

如果这个数小于零，则提示输入的数没有意义。如果不是一个整数，则将其向下舍入为相邻的整数。如果这个数为“0”或“1”，则其阶乘为1。如果这个数大于0，则将其与相邻较小的数的阶乘相乘。要计算任何大于0的数的阶乘，至少需要计算一个其相邻数的阶乘。用来实现这个功能的函数就是已经位于其中的函数；该函数在执行当前的这个数之前，必须调用它本身来计算相邻的较小数的阶乘。这就是一个递归。

在使用递归时大家应明确两点：

- (1) 递归就是在过程或函数里调用自身；
- (2) 在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

采用函数的递归调用可使程序的结构紧凑，但是递归调用要求采用再入函数，以便利用再入堆栈来保存有关的局部变量数据，从而要占据较大的内存空间，另一方面递归调用时对函数的处理也是比较慢的，因此一般情况下应该避免使用递归。递归和迭代（循环）是密切相关的；能用递归处理的算法也都可以采用迭代，反之亦然。确定的算法通常可以用几种方法实现，您只需选择最自然贴切的方法或者您觉得用起来最轻松的一种即可。

动手实验（18）

实验目的：学习函数的递归调用

实验内容：利用实验板做上面所讲的阶乘实验。

```
#include<reg52.h>//包含52的头文件
#include<stdio.h>//包含标准输入输出头函数文件
#define uint unsigned int
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
int fac(int n) reentrant//定义一个递归函数
{
    int result;
    if(n==0||n==1)
        result=1; //假如这个是“0”或“1”，则结果为1。
    else if(n<0)
        printf("n<0 you input error");//假如这个数是小于“0”，此数无阶乘意义。
    else
        /*假如这个数是大于“0”的整数，则将其与相邻较小的数的阶乘相乘*/
        result=n*fac(n-1);
    return(result);//返回结果
}
void main(void)
{
    int n;
    int result;
    uart();
    while(1)
    {
```

```
printf("\ninput a inteager number:\n"); //提示输入一个数
scanf("%d",&n); //从键盘输入一个数
result=fac(n); //调用阶乘函数
printf("factorial=%d",result); //输出阶乘结果
}
}
```

- 实验步骤：1. 打开光盘第3章/ recursion / recursion.Uv2 工程文件，对程序进行编译、链接、调试产生 recursion.hex 烧写文件。
2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为9600、不选取十六进制的发送与接收、打开串口。把 recursion.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
3. 从串口中输入一个数（设为4）单片机就会输出其阶乘的结果24；同样如果输入6那么单片机就会输出其结果720。如下图3-31所示。

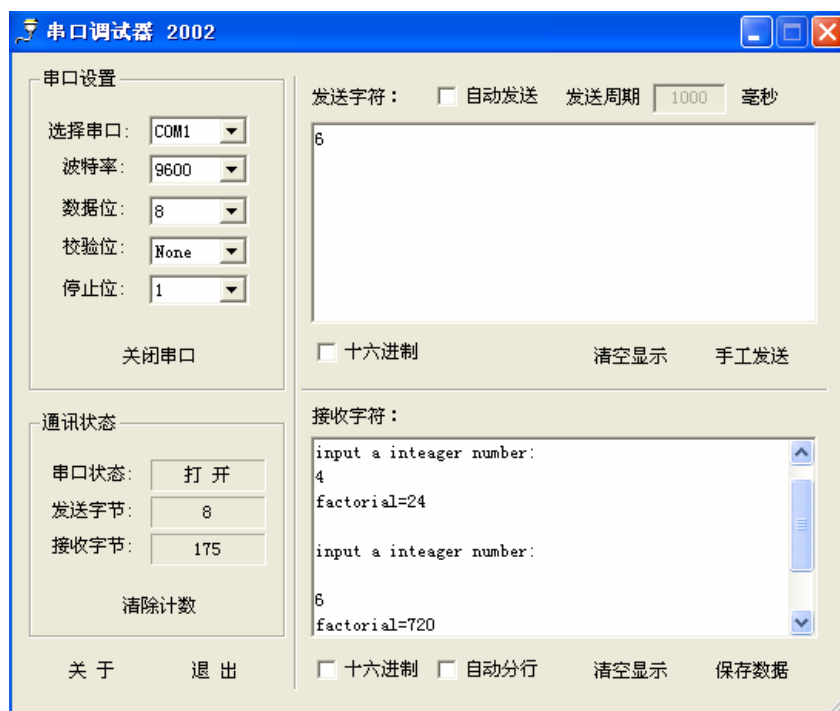


图 3-31

四. 函数变量的存储种类

变量的存储方式可分为“静态存储”和“动态存储”两种。

第一种，静态存储变量通常是在变量定义时就分配存储单元并一直保持不变，直至整个程序结束。全局变量即属于此类存储方式。

第二种，动态存储变量是在程序执行过程中，使用它时才分配存储单元，使用完毕立即释放。典型的例子是函数的形式参数，在函数定义时并不给形参分配存储单元，只是在函数被调用时，才给

以分配，调用函数完毕立即释放。如果一个函数被多次调用，则反复地分配、释放形参变量的存储单元。

从以上分析可知，静态存储变量是一直存在的，而动态存储变量则时而存在时而消失。我们又把这种由于变量存储方式不同而产生的特性称变量的生存期。生存期表示了变量存在的时间。生存期和作用域是从时间和空间这两个不同的角度来描述变量的特性，这两者既有联系，又有区别。一个变量究竟属于哪一种存储方式，并不能仅从其作用域来判断，还应有明确的存储类型说明。

下面我们来讨论关于C语言变量的存储类型说明，共分4种：

`auto`（自动变量）、`static`（静态变量）、`extern`（外部变量）、`register`（寄存器变量）

下面分别介绍以上四种存储类型：

一、`auto`（自动变量）

这种存储类型是C语言程序中使用最广泛的一种类型。C语言规定，在函数内，凡未加存储类型说明的变量均视为自动变量，也就是说自动变量可省去说明符`auto`。在前面各章的程序中所定义的变量凡未加存储类型说明符的都是自动变量。例如：

```
{  
  
    long x, y;  
  
    int a;  
    .....  
  
}
```

等价于：

```
{  
  
    auto long x, y;  
  
    auto int a;  
    .....  
  
}
```

自动变量具有以下特点：

1. 由于自动变量的作用域和生存期都局限于定义它的函数或复合语句内，因此不同的个体中允许使用同名的变量而不会混淆。即使在函数内定义的自动变量也可与该函数内部的复合语句中定义的自动变量同名。自动变量的作用域仅限于定义该变量的个体内。在函数中定义的自动变量，只在该函数内有效，在复合语句中定义的自动变量只在该复合语句中有效。

2. 自动变量属于动态存储方式，只有在使用它，即定义该变量的函数被调用时才给它分配存储单元，开始它的生存期。函数调用结束，释放存储单元，结束生存期。因此函数调用结束之后，自动变量的值不能保留。在复合语句中定义的自动变量，在退出复合语句后也不能再使用，否则将引起错误。

二. static (静态变量)

我们已经知道，自动变量在调用完函数之后就会释放变量，变量的值就会不存在。那么如果程序要求在函数调用完之后变量不会自动释放，在下次调用的时候还可以保留上次的变量值，那么应该如何是好啊？在这种情况下就要用 static 关键字来定义，如下面的程序所示。

```
void add(void)
{
    static int a=0;//声明一个静态变量
    int b=0;      //声明一个自动变量
    a=a+1;
    b=b+1;
    printf("a=%d\n",a);//输出 a 的值
    printf("b=%d\n",b);//输出 b 的值
}
main()
{
    char i;
    for(i=0;i<5;i++)
        add();
    while(1);
}
```

上面的程序当中，在 add 函数中分别定义了两个变量，一个为自动变量，另一个为静态变量，每一次调用 add 函数就会将两个值加 1 然后输出。在主程序中利用 for 循环调用 5 次 add 函数，结果可以看到，每一次调用 a 的值就会加 1 最后输出 5，但是变量 b 的值就每次调用输出都是 1。

从而我们可以得出，利用 static 定义的变量属于静态存储类别，在程序运行过程当中不会被释放。而且，静态变量只在程序被编译时赋一次初值，但自动变量就会每次调用都会赋初值；最后还有一点大家应该注意的，就是如果在定义静态变量时不给予赋初值，在系统编译时就会赋予“0”值，但是自动变量则不是，它会是一个不确定的值。

三. extern (外部变量)

```
void dec(void)
{
    extern x,y;//声明 x,y 是已经定义的外部变量
    printf("x=%d\n",x);
    printf("y=%d\n",y);
}
```



```
}  
int x=9,y=55;//声明 x,y 为全局变量  
main()  
{  
    dec();  
    while(1);  
}
```

就以上的程序，按我们前面学习过的知识，变量 x, y 是一个全局变量，其作用域是从定义处起到程序的结果，而在其前面的函数是不能使用它的，但是为什么上面的程序还是可以使用它呢？因为我们用了一个关键字 `extern` 将其从新定义为一个外部变量，意思就是告诉编译器：这个变量已经被定义了。那么 x, y 变量的作用域就被扩充了，就是从被 `extern` 定义处开始一直到程序的结尾。所以上面的程序在调用 `dec` 函数的时候输出 x, y 的值，就是 $x=9, y=55$ 。

四. register (寄存器变量)

当 CPU 要访问一个变量时，是要花费一定时间进行操作的。在程序中如果一个变量用得比较频繁时，那么访问它不是都要花费大量的时间吗！这样就大大降低了程序的执行效率。因此，为了提高程序的执行效率，C 语言允许把在程序当中用得比较频繁的变量定义为 `register` 寄存器变量，这样被定义的变量就可以存放在 CPU 寄存器当中，在访问时就不用花费大量时间，从而提高程序的执行效率。我们在使用时要注意，寄存器变量属于动态存储方式，只有局部的自动变量和形式参数才能定义为寄存器变量，而采用静态存储方式的变量则不能定为寄存器变量。

其实，现时绝大多数的编译器都进行了优化，包括 `keil`。就是在系统编译时凡是程序用得较为频繁的变量都被定义为 `register` 寄存器变量，所以程序员在编写程序时无必要再进行定义。

第九节 指针

指针是 C 语言中的一个重点，也是 C 语言的一个难点，曾经有许多人这样讲过，要精通指针的程序员才算真正懂得 C 语言。只要掌握指针，才能使程序变得更加简洁、紧凑、高效，在 C 语言当中指针可以谈得上是全部精华的所在。但是指针因为较为复杂，所以使程序的可读性大大降低。因此初学者在开始学习时可能会有一点不习惯、会时常出错误，因为这是一个学习的过程。但是不要灰心，只要我们在平时多思考，多上机，那么很快就可以掌握它了。

一. 关于指针

大家都知道计算机的内存是以字节为单位的一片连续的地址。而每一个地址都有自己的编号。就好比拟是旅馆的房号一样，旅馆的管理员每次寻找住客，就必定要先知道房间号；而计算机的 CPU 也是一样，每一次访问一个变量，就得先知其地址编号。如果没有了字节地址，系统就无法对内存进行管理。下面我们举例来讨论。

```
char a=5;
```

```
int b=2;
```

float c=3;

当在程序中声明了一个变量，如上面所示，那么在编译的时候编译器就会根据数据的类型给变量分配合适的内存空间。如果是字符型则分配一个字节空间，而整型则分配两个字节空间，当然啦，如果是实型数据就得分配四个字节的内存，如下面所示。

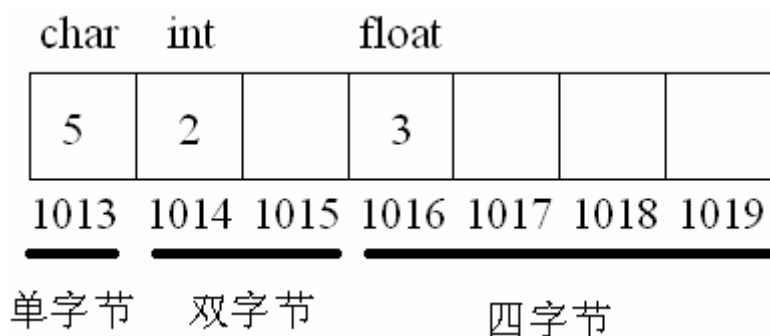


图 3-32

图 3-32 形象地反映了在程序当中声明的变量经过编译后在内存空间的存放情况。所谓变量地址就是变量在内存空间存放的首地址。那么好明显 a 的首地址为 1013，b 的首地址为 1014，而 c 的首地址为 1016。

在 C 语言中访问变量有两种方法：

- (1) 在程序当中只需要知道变量名，而不需理会变量在内存中的地址，其中变量与地址之间的联系由编译器来完成。当程序要访问某个具体的变量时，实质也是对变量在内存地址的编号来进行访问，这种按变量地址进行直接访问的方式称为之“直接存取”，前面我们所讨论的例题和所做的实验都是使用这种方式。如图 3-33 与图 3-32 相同，定义了一个字符变量 a，其中 a 被分配了一个字节单元内存，给变量 a 赋值数值 5。图中所示为直接操作的过程。无需理会内存地址。而是对变量名进行存取，如：a=5;而其它的工作则由编译器完成。

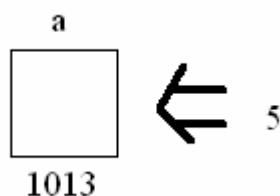


图 3-33

- (2) 按 C 语言规定，可以在程序中定义整型变量，实型变量，字符变量等，也可以定义这样一种特殊的变量，它是存放地址的。将“变量地址”存放在这个变量当中，当需要访问变量内容时，先找到存放“变量地址”的变量，从中取出“变量地址”，再取出“变量地址”的内容。如图 3-34

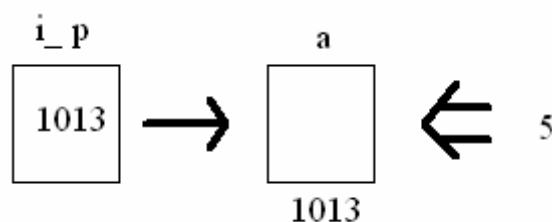


图 3-34

如图 3-34 所示，在定义了一个字符型变量以外，还另外定义了一个特殊变量“指向”变量 `a` 的内存地址，当需要操作的时候就利用取地址运算符“&”或取内容运算符“*”进行操作。如 `i_p=&a` 或 `*p`。

下面我们就指针来解释一下几个名词概念：

指向： 所谓指向变量是通过地址来体现的。`i_p` 中的值为 `1013`，而 `1013` 是变量 `a` 的地址，这样两者之间就建立了一种关系，即通过 `i_p` 可以知道变量 `a` 的地址，从而找到变量 `a` 的内存单元，所以图 3-34 中用箭头表示这种“指向”的关系。

指针： 在 C 语言中，将地址形象在称为“指针”。就是通过它能找到以它为地址的内存单元。例如：地址 `1013` 是变量 `a` 的指针。

指针变量： 假如用一个变量专门用来存放另一个变量的地址（即指针），则这个变量被称为指针变量。例如：上面讲术到的 `i_p` 就是一个指针变量。

大家要注意区分“指针”与“指针变量”的概念。如上图所说：可以说变量 `a` 的指针为 `1013`，而不能说 `a` 的指针变量为 `1013`。在此还应强调一点，当进行完变量、指针变量定义之后，如果对这些语句进行编译，那么 C 编译器就会给每一个变量和指针变量在内存中安排相应的内存单元。然而，这些单元地址除非使用特殊的调试软件，否则是看不见的。上面提到的 `1013`，`1014`，`1015` 等只是为了说明问题而列出来的。

二. 指针变量的定义与初始化

指针变量同前面所讲的一般变量也是一样的，在使用前必须先定义。

语法格式：

数据类型 *指针变量名；

例如：char *p;

int *ap;

int *p, *i;

以上都是合法的定义，指针变量的定义与一般的变量定义大致相同，只是在定义指针变量时在指针变量名前加一个“*”。

下面我们来讨论一下指针变量的初始化，也就是用赋值语句使指针变量指向另一个变量，下面是合法的语句。

例如：

```
int x=5,y=8,z=55;    //定义 x,y,z 为三个整型量

int *xp,*yp,*zp;     //定义 xp,yp,zp 为三个指针变量

xp=&x;                //xp 指向了 x

yp=&y;                //yp 指向了 y

zp=&z;                //zp 指向了 z
```

上面的例子当中分别定义了一般变量与指针变量，最后用取址运算符“&”对指针变量进行赋值。指针变量 xp 存储了变量 x 的地址，即 xp 指向了 x；指针变量 yp 存储了变量 y 的地址，即 yp 指向了 y；指针变量 zp 存储了变量 z 的地址，即 zp 指向了 z。

三. 指针的引用

请牢记，指针变量只能存放地址，绝对不能将非地址类的数据赋于一个指针变量，下面的赋值操作是非法的。

```
int *p;

p=46;
```

关于指针操作的两个重要的运算符，我们再来重复一下：

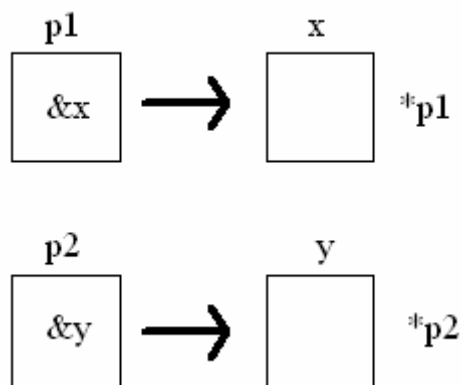
- (1) &：取地址运算符。
- (2) *：取内容运算符（或指针运算符）

假如：**&i** 表示变量 i 的地址；***p** 为指针变量 p 所指向的内容。下面我们用一个具体的例子来讨论关于指针的引用：

例 1：

```
int x=50,y=100;

int *p1,*p2;
```



```
p1=&x;

p2=&y;

printf("%d %d\n",*p1,*p2);

printf("%d %d\n",x,y);
```

图 3-35

下面我们来理解例 1 程序的思路。首先，程序的第 1 行与第 2 行分别定义了两个普通变量 x, y 和指针变量 $p1, p2$ 。 x, y 分别赋予初值 50、100，而指针变量 $p1, p2$ 只是规定它们指向一个整型变量，至于具体指向那一个整型变量还是要在程序中指明。程序的第 3 行第 4 行就是具体使 $p1$ 指向 $x, p2$ 指向 y ，那么 $p1$ 的值就是 $\&x$ 、 $p2$ 的值就是 $\&y$ 。最后在程序的第 5 行第 6 行分别输出 $p1$ 、 $p2$ 、 x 、 y 的值。（请参照图 3-35 来分析）

在以上的程序当中有三个问题要注意的（请参照图 3-35 来分析）：

第一个、在程序中出现了两次的 $*p1$ 、 $*p2$ 。第一次是在定义指针变量时出现的，其中“*”的意思是指所定义的变量为一个指针变量；而第二次是在 `printf` 输出函数的形参中出现的，其中“*”的意思是指 $p1$ 、 $p2$ 所指向的变量，也就是 x 、 y 。

第二个、是关于指针变量赋值语句的正确使用， $p1=\&x$ 、 $p2=\&y$ ；是指将 x 、 y 的地址分别赋给 $p1$ 、 $p2$ ，但是如果写成 $*p1=\&x$ 、 $*p2=\&y$ ；是错误的，因为 x 与 y 的地址是赋予给指针变量 $p1$ 与 $p2$ 而不是 $*p1$ 与 $*p2$ （即变量 x 与 y ）。

第三个、在编写程序时，要注意指针的指向。指针变量只能指向与自身类型相同的变量，例如：整型指针变量只能指向整型变量，而绝不能时而指向整型变量，时而指向字符变量。

动手实验（19）：

实验目的：学习利用指针作为函数的形式参数。

实验内容：此实验是将指针变量作为函数的参数，从主调用函数中利用指针变量把两个数据传递到被调用函数中去，而从被调用函数中将其两个数对调最后从主调用函数中按由大到小输出。

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>//包含标准输入输出函数头文件
#define uint unsigned int
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
```

```
TI=1;
TR1=1;
}
void swop( uint *p1,uint *p2)
{
//此函数功能是把 p1 p2 指向地址的值对换
//p1 p2 为指向无符号整型数的指针变量
    uint temp;
    temp=*p1;//把 p1 所指向地址的值赋于 temp
    *p1=*p2;//把 p2 所指向地址的值赋于 p1 所指向地址的值
    *p2=temp;//把 temp 赋于 p2 所指向地址的值
}
void main(void)
{
    uint x,y;
    uint *p_x,*p_y;
    uart();           //串口初始化
    p_x=&x;           //p_x 指向 x 的地址
    p_y=&y;           //p_y 指向 y 的地址
    while(1)
    {
        printf("please input two number\n"); //提示输入两个数
        scanf("%d%d",&x,&y);           //从键盘输入两个数
        if(*p_x<*p_y)                   //假如 p_x 指向地址的内容要比 p_y 的小
            swop(p_x,p_y);               //调用 swop 函数
        printf("seriation %d  %d\n",*p_x,*p_y);//输出 p_x p_y 指向地址的内容
    }
}
```

实验步骤：1. 打开光盘第 3 章/ pointer / pointer.Uv2 工程文件，对程序进行编译、链接、调试产生 pointer.hex 烧写文件。

2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制的发送与接收、打开串口。把 pointer.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
3. 从串口中分别输入 86 和 92，单片机就会按由大到小的顺序将两上数输出到接收区。如下图 3-36 所示。

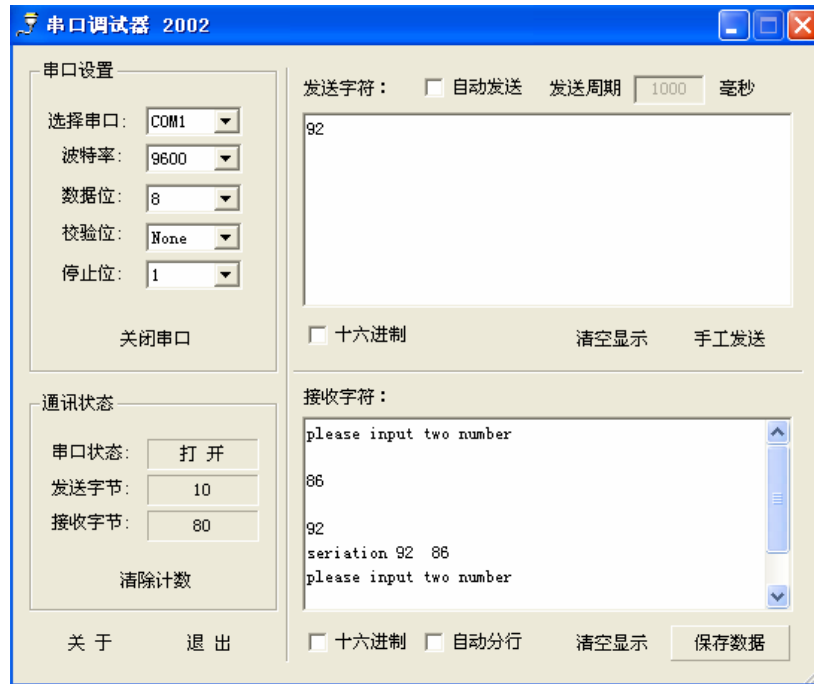


图 3-36

实验分析：经过上面的实验之后可能会有读者这样问，需然在 `swop` 函数中是把两个数对调了，但是为什么没有利用 `return` 语句返回函数的值，在主函数中输出 `*p_x` 与 `*p_y` 的值却是对调了。前面我们曾经讲过指针的值就是一个地址，下面我们根据这一点再来分析一下：

第一，在主函数中我们利用 `swop(p_x,p_y);`语句把 `p_x` 和 `p_y` 的地址赋给了 `p1` 与 `p2`。

第二，再在 `swop` 函数中利用 `*p1` 与 `*p2` 指向了地址的值（即 86 与 92）将其对调了。意思是地址是没有变的，但是地址的值已经变了。

第三，那么我们在主函数中利用 `*p_x,*p_y`，意思是输出地址的值那必然是对调了。大家要记住！在 C 语言当中我们只是利用指针来改变地址的内容，绝不会改变地址本身，因为内存地址是不可能被改变的。

自我练习：编写一个计算长方形的面积函数，从串口输入长方形的长与宽。利用指针作为函数的形式参数定义一个函数计算其面积，在主函数中调用其函数并利用 `printf` 函数输出结果，工程名命名为“`mypointer`”。

程序的设计思路：

本练习同上面动手实验是大同小异的，上面的动手实验是把两个数对调并没有返回一个值，但是这里我们可以试一下利用 `return` 语句将长方形的面积返回到主调用函数。

（练习答案附光盘中）

四. 指针与数组

声明一个变量，在编译时就会给与分配一个内存地址；同样，一个数组包含若干元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。所谓数组的指针是指数组的起始地址，数组元素的指针是数组元素的地址。一个数组在内存单元当中是连续存放的，数组名就是这块连续内存单元的首地址。

```
int max[8]={1,2,3,4,5,6,7,8};
```

```
int *a;
```

```
a=&max[0];
```

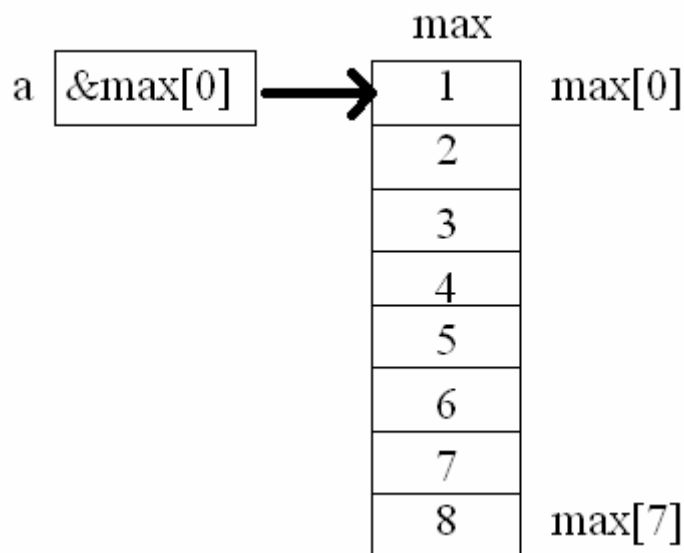


图 3-37

第一行为声明一个带有 8 个元素的数组；第二行为声明一个整型的指针变量，至于第三行则是指向了数组 `max` 的第 0 个元素，即 `max[0]`。那么这时 `a` 的值为 `max[0]` 的地址，因为数组元素在内存中是连续存放的，因此可以通过指针变量 `a` 来访问数组中有关变量的内容（如图 3-37）。`a`, `max`, `&max[0]` 均指向同一单元，它们是数组 `max` 的首地址，也是 0 号元素 `max[0]` 的首地址。

在 C 语言当中，数组名代表数组的首地址，即第 0 号元素的地址。因此以下语句是具有相同意义的：

```
a=&max[0];  
a=max;
```

我们也可以在定义的同时赋予其初值：

```
int *a=&max[0];  
或者  
int *a;  
a=&max[0];
```

下面我们来分析一个关于数组与指针的例子：

```
int team[10]={85,4,3,69,8,98,52,1,9,12};  
int temp,i,*p;           //定义两个普通变量和一个指针变量  
p=team;                 //指针变量 p 指向了数组的首地址  
temp=*p;                //把 p 所指向的地址的值赋予 temp  
for(i=0;i<9;i++)
```



```
{
    if(temp<*(p+i+1))    //假如 temp 的值比下一个地址的值要小
        temp=*(p+i+1); //将下一个地址的值赋于 temp
}
printf("the max number is %d\n",temp);//输出最大值
```

上面的程序是在 `team` 数组中找出一个最大值的程序算法，现在我们来分析程序的结构，程序的第一、二句是定义数组与变量；因为数组名代表了整个数组在内存当中的首地址，所以第三句将数组的首地址赋给了指针变量 `p`；第四句是将指针变量 `p` 所指向地址的值赋给了变量 `temp`（即 `team[0]` 的值）；第五句是一句循环语句，共循环 9 次以寻找出数组 `team` 中的最大元素；第六句是一句 `if` 选择语句，其作用是将数组中相近的两个数进行判断，选其最大值，在最后的 `printf` 函数中输出。

动手实验（20）：

实验目的：学习指针数组的使用。

实验内容：在理解上面讨论的关于数组与指针例子的前提下，下面我们来做一个实验，从串口中输入 5 个数存放于数组当中，声明一个指针变量指向该数组的首地址，再利用函数调用的操作方式找出其中的最大者，最后在主函数中输出，读者在这个实验中应撑握数组与指针是如何操作的。

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>//包含标准输入输出头函数文件
#define uint unsigned int
void uart(void)//串口初始化函数
{
    SCON=0x40;
    PCON=0;
    REN=1;
    TMOD=0x20;
    TH1=0xfd;
    TL1=0xfd;
    TI=1;
    TR1=1;
}
uint choose(uint *p)//此函数功能为找出并返回最大值
{
    uint i,temp;
    temp=*p;        //把 p 所指向的地址的值赋于 temp
    for(i=0;i<5;i++)
    {
        if(temp<*(p+i+1))    //假如 temp 的值比下一个地址的值要小
            temp=*(p+i+1); //将下一个地址的值赋于 temp
    }
}
```

```
        return temp;
    }
void main(void)
{
    uint i,team[5]; //声明了一个变量和数组
    uint *p;       //声明了一个指针变量
    uart();        //串口初始化
    p=team;        //指针变量 p 指向了数组的首地址
    while(1)
    {
        printf("please input five number\n");//提示输入 5 个数
        for(i=0;i<5;i++)
        {
            scanf("%d",p+i);//输入的数值存放于 p 所向的地址+i
        }
        printf("the max number is %d\n",choose(p));//调用 choose 函数得到返回值，在 printf
        函数中输出.
    }
}
```

- 实验步骤：1. 打开光盘第 3 章/ p_choose / p_choose.Uv2 工程文件，对程序进行编译、链接、调试产生 p_choose.hex 烧写文件。
2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制的发送与接收、打开串口。把 p_choose.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。
3. 从串口中分别输入五个数设为（45、 12 、 33、 14 、 12）然后单片机就会把其中的大者输出到串口的接收区。如下图 3-38 所示。

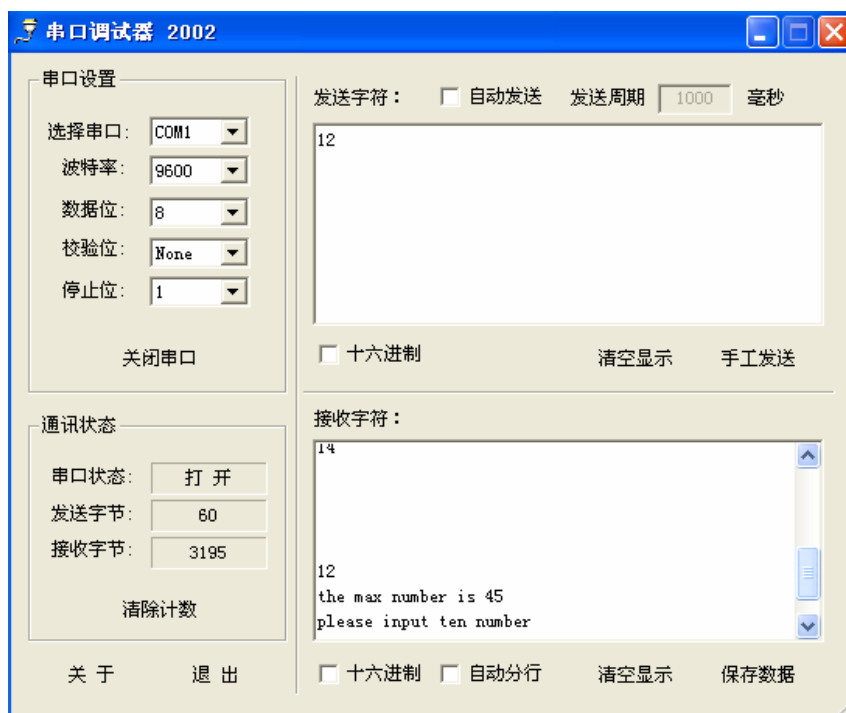


图 3-38

自我练习:

自行编写一个程序将其命名为“p_array”，在程序中定义一个带有 10 个元素的数组，当 K2 被按下之后，利用循环语句赋值 0~9，再将此数组的首地址传递到另一个函数中将其 10 个元素全部从串口中输出。

程序的设计思路：可以在程序中声明一个名为 array [10]的数组，再声明一个指针变量指向它，利用循环语句进行赋值例如：

```
for(i=0;i<10;i++)
{
    *p++=i;
}
```

在上面的程序当中，因为*的优先级比++要高，所以 P 先与*相结合之后再自加 1，所以*p++=i 的意思：先把 i 的值赋于 p 所指向地址的值，然后再使 p 的地址+1。完成赋值之后把数据传递到另一个函数中同样用循环语句输出即可。

(练习答案附光盘中)

第四节 AT89S52 单片机的内部资源

经过前一章关于单片机 C 语言的讨论，相信大家已经有了一定的了解。但是要想成为一个单片机程序员这样是远远不够的，还得要不断地组织你的逻辑思维，不停地动手进行程序的编写，从错误中得取经验。在此给大家介绍一本关于 C 语言的书《C 语言程序设计》，这本书是现今大学比较常用的 C 语言的教材，相信对大家是有用的。

第 1 节 单片机基本结构与引脚功能



图 4-1 (AT89S52 实物图)

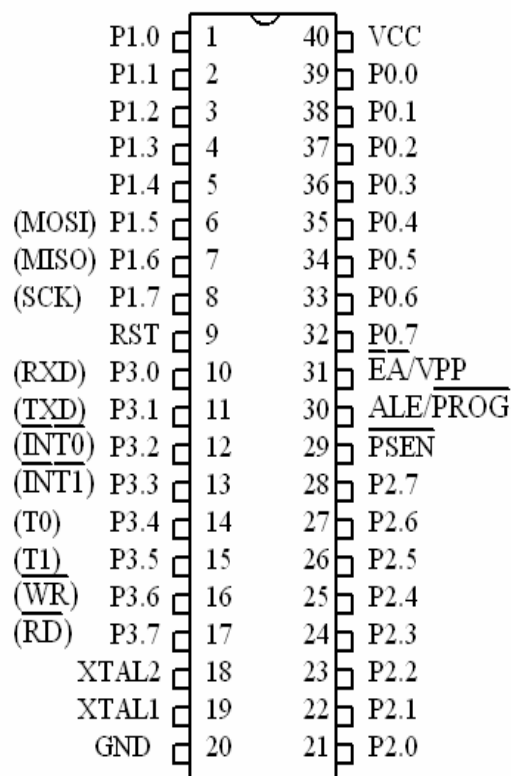


图 4-2 (AT89S52 引脚图)

图 4-1 为单片机的实物图，图 4-2 为其引脚功能图，而下面的图 4-3 为内部的结构原理图，从下面的图 4-3 我们可以看到，其实单片机内部非常的简单，就是那些控制应用所必需的基本内容集成在一个电路芯片里，若按功能划分，它由如下功能部件组成，微处理器，数据存储器，程序存储器，并行 I/O (P1、P2、P3、P4)、串行口、定时器/计数器、中断系统与特殊功能寄存器 (SFR)。它们都是通过芯片内的总线相连而成，在芯片内的各个功能可以看作是由不同的功能芯片所组成，所谓单片机编程，就是对 CPU 编写特定的程序来控制各个单元芯片。下面我们先来了解其引脚功能。

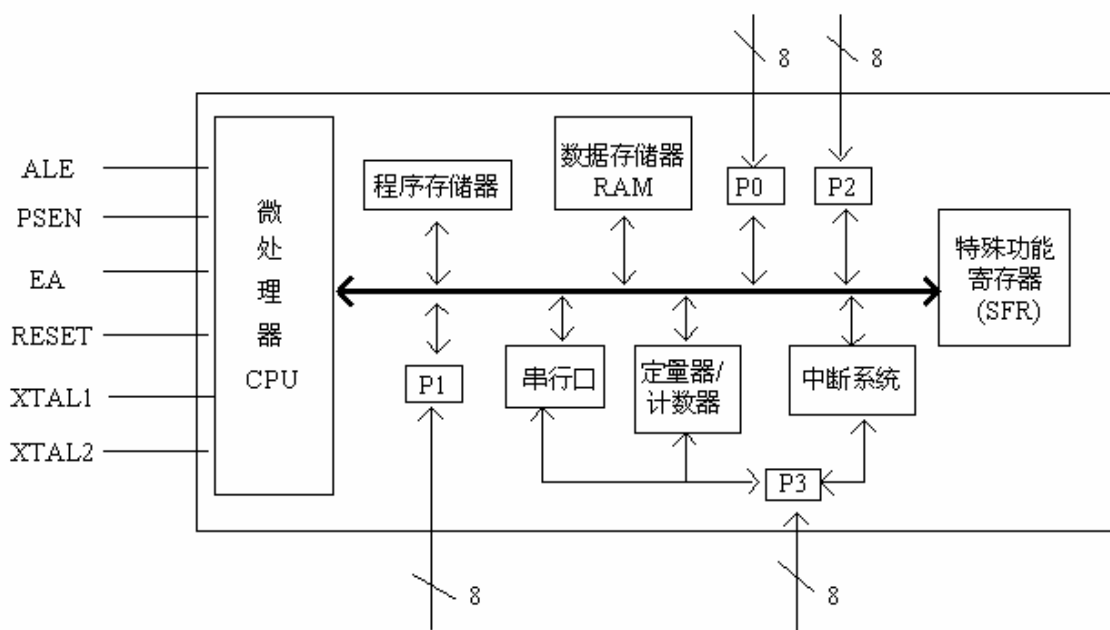


图 4-3

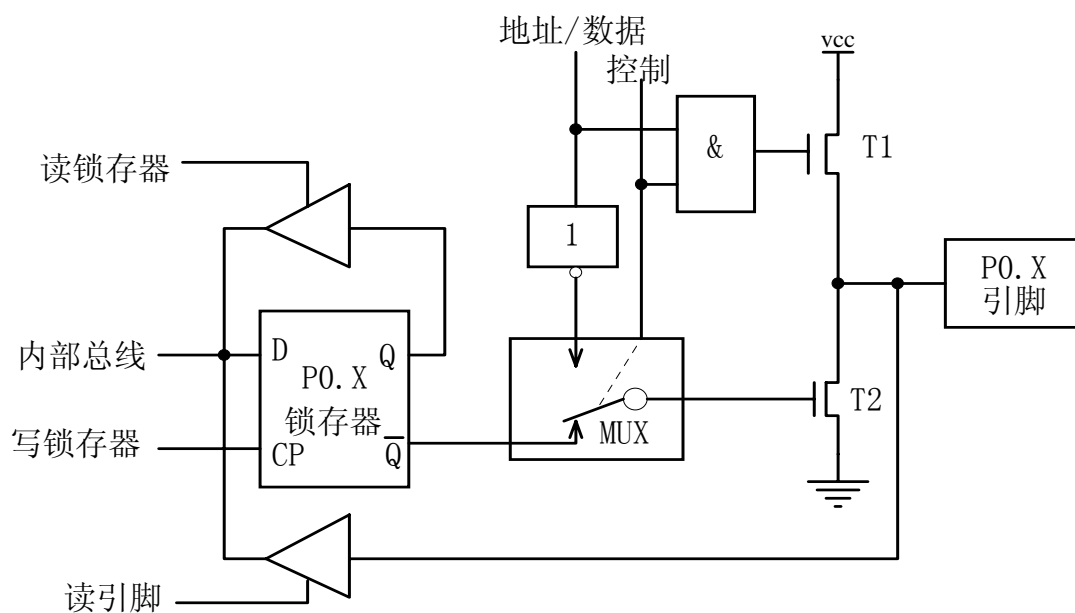


图 4-4

P0 口：（39 脚~32 脚）P0 口的字节地址为 0x80, 位地址为 0x80 至 0x87, P0 口的每个端口的功能逻辑电路是完全一样的。如图 4-4 可见，电路中包含有 1 个数据输出锁存器、2 个三态数据输入缓冲器、1 个数据输出的驱动电路和 1 个输出控制电路。下面我们来详细讨论其工作原理。

在运行时若对 P0 口进行写操作，由锁存器和驱动电路构成数据输出通路。由于通路中已有输出锁存器，因此数据输出时可以与外部设备直接连接，而不需再加数据锁存电路。考虑到 P0 口既可以作为通用的 I/O 口进行数据的输入输出，也可以作为单片机系统的地址/数据线使用。为此在 P0 口的电路中有一个多路转接电路 MUX。在控制信号的作用下，多路转接电路可以分别接通锁存器输出或地址/数据线。当作为通用的 I/O 口使用时，内部的控制信号为低电平，封锁与门将输出驱动电路的上拉场效应管（FET）截止，同时使多路转接电路 MUX 接通锁存器 Q 端的输出通路。当 P0 口进行一般的 I/O 输出时，由于输出电路是漏极开路电路，必须外接上拉电阻才能有高电平输出；当 P0 口进行一般的 I/O 输入时，必须先向电路中的锁存器写入“1”，使 FET 截止，以避免锁存器为“0”状态时对引脚读入的干扰。

在实际应用中，P0 口绝大多数情况下都是作为单片机系统的地址/数据线使用，这要比作一般 I/O 口应用简单。当输出地址或数据时，由内部发出控制信号，打开上面的与门，并使多路转接电路 MUX 处于内部地址/数据线与驱动场效应管栅极反相接通状态。这时的输出驱动电路由于上下两个 FET 处于反相，形成推拉式电路结构，使负载能力大为提高。而当输入数据时，数据信号则直接从引脚通过输入缓冲器进入内部总线。

P1 口：（1 脚~8 脚）P1 口的字节地址为 0x90, 位地址为 0x90 至 0x97, P1 口的逻辑电路与 P0 口的逻辑电路结构上有一点不同。比起 P0 口因为它只传送数据，所以不需要多路转接开关 MUX；而且 P1 口还多了一个上拉电阻，所以当 P1 口作为输出口使用时，已能对外部提供推拉电流负载，外电路无需再接上拉电阻。P1 口的每个端口功能逻辑电路是完全一样的。如图 4-5，而 P1 口的某些引脚还具备了第二功能，其每个功能如表 4-1 所示。

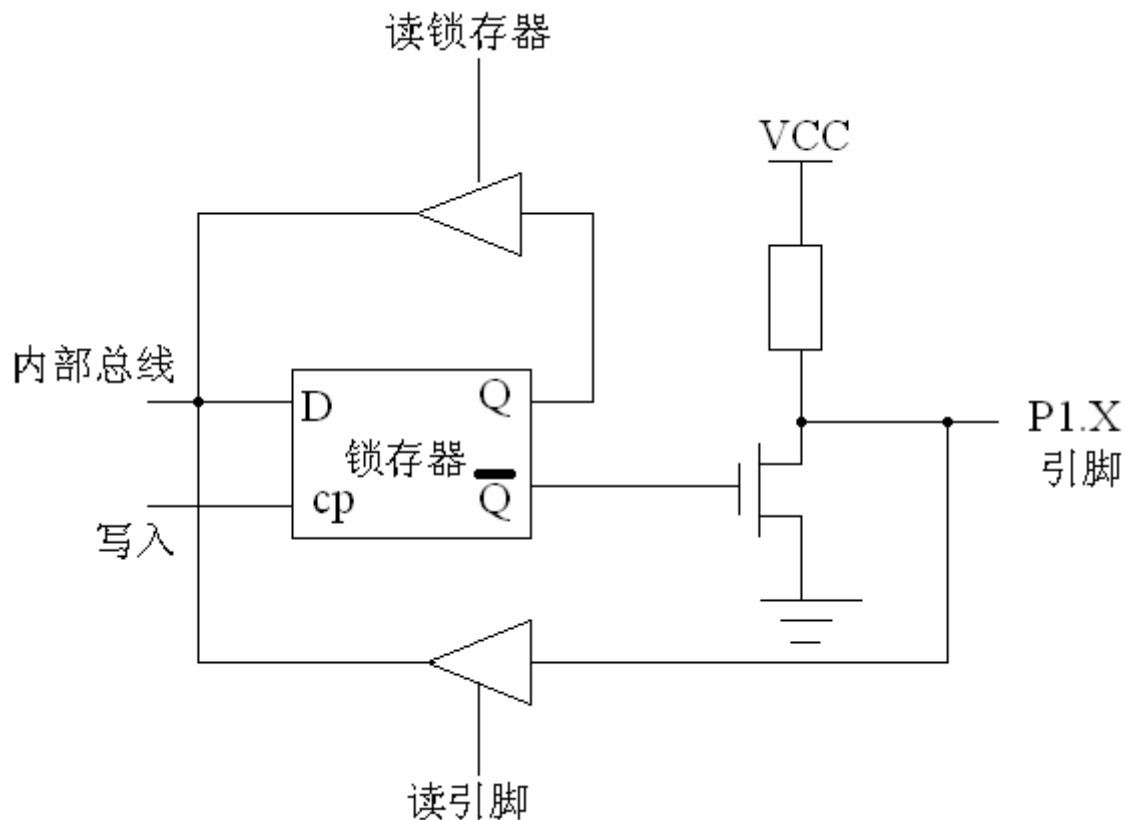


图 4-5

表 4-1

引脚号	第二功能
P1.5	MOSI (在系统编程用)
P1.6	MISO (在系统编程用)
P1.7	SCK (在系统编程用)

P2 口：（21 脚~28 脚）P2 口的字节地址为 0xa0, 位地址为 0xa0 至 0xa7。P2 口的每个端口功能逻辑电路是完全一样的。P2 口与 P0 口一样，在电路中有一个 MUX 多路开关，在作为通用的 I/O 口使用时，MUX 开关会转接到锁存器的 Q 端，其工作原理与 P0 类似。作为输出口使用时无需外接上接电阻。如图 4-6

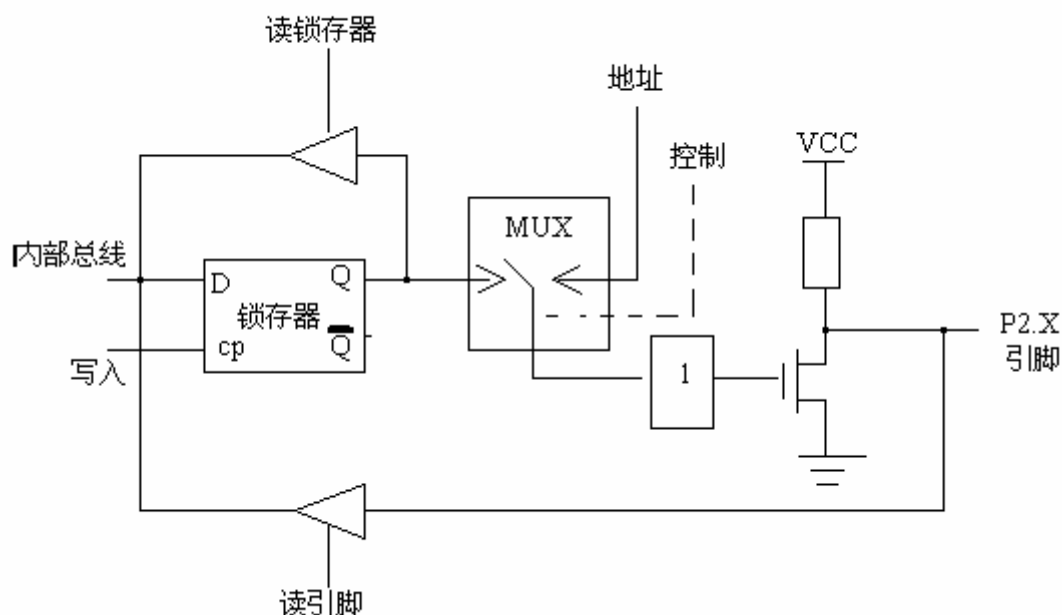


图 4-6

P3 口：（10 脚~17 脚）P3 口的字节地址为 0xb0, 位地址为 0xb0 至 0xb7。P3 口的逻辑电路如图 4-7，P3 端口为了满足电路系统的一些特殊功能的需要，增加了第二功能。从图 4-7 的逻辑电路中我可以清楚地见到，第二功能有输入与输出之分，下面我们分别来讨论这两个逻辑功能。

第一，先来讨论关于第二功能的输入引脚，端口线的输入端多加了一个缓冲器，第二功能的输入信号就从此端取得，而作为普通端口使用时，还是以三态缓冲器输出。

第二，再来讨论关于第二功能的输出引脚，当作为 I/O 使用时，片内的与非门开通，以维持从锁存器到输出端数据输出通路的畅通。当作为第二功能输出信号时，该位的锁存器置“1”，使与非门对第二功能信号的输出保持畅通，从而实现第二功能信号的输出。

在实际的应用中，用户只可能在普通的 I/O 端口与第二功能之间任选其一，也就是说，如果我们选取引脚作为第二功能那就不能再作为普通端口使用；但是如果选其作为普通端口那就再也不能选用第二功能；作为输出口使用时，无需外接上拉电阻，其用法与 P1 口相同。关于第二功能如表 4-2 所示，在后面会对第二功能作详细的介绍。

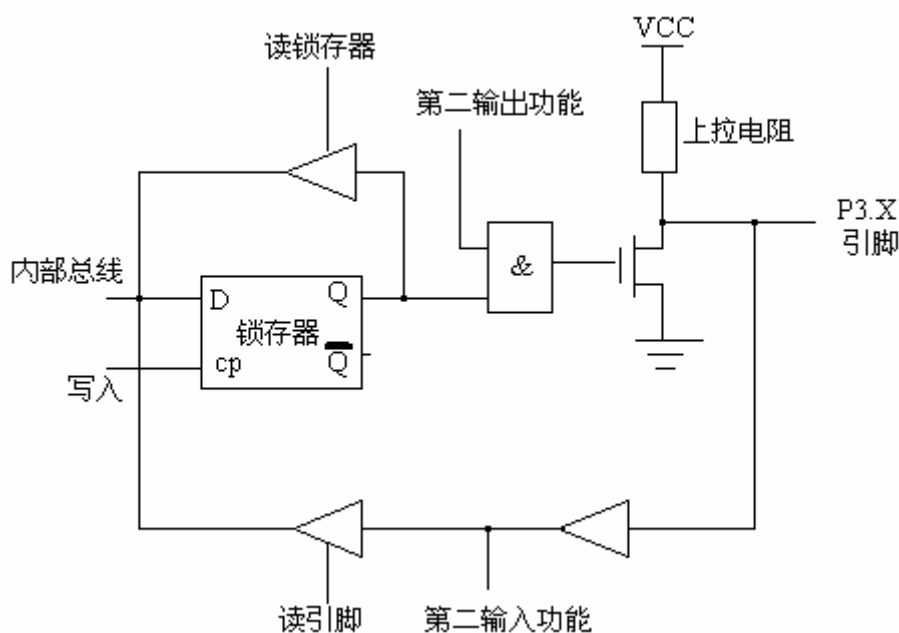


图 4-7

表 4-2

引脚号	第二功能
P3.0	RXD (串行数据输入)
P3.1	TXD (串行数据输出)
P3.2	INIT0 (外部中断 0)
P3.3	INIT1 (外部中断 1)
P3.4	T0 (定时器/计数器 0 外部输入)
P3.5	T1 (定时器/计数器 1 外部输入)
P3.6	WR (外部数据存储器写选通)
P3.7	RD (外部数据存储器读选通)

在较多的书籍中对端口的讲解非常的复杂，使读者看起来非常的心烦，在本节中我们将用实验来让读者去了解如何使用端口，这样比不停地讲理论要强得多，更加提高读者的学习兴趣。在做实验前，我们还来看一下除了普通端口和第二功能以外的其它引脚功能。

RST: (9 脚) 复位信号输入引脚，高电平有效。当单片机运行时，在此引脚上持续加上大于 2 个机器周期的高电平。就可以让单片机复位，从新于 0x00 的地址开始运行。图 4-8 是常用的复位电路。

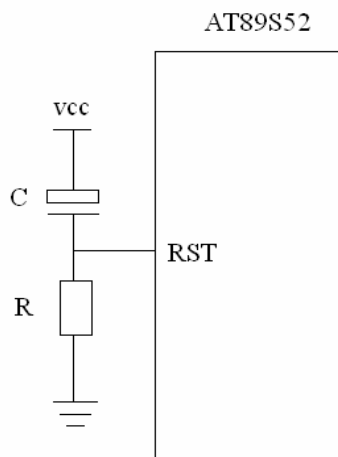


图 4-8

ALE/PROG: (30 脚)

ALE 当单片机正常运行的时候, 此引脚会不断输出脉冲信号, 此频率为时钟振荡频率的 1/6。如果想判断单片机芯片是否良好, 可以用示波器查看 **ALE** 端是否有正脉冲信号输出, 如果没有则单片机已经损坏。

PROG 为本引脚的第二功能, 在对片内 EPROM 型单片机 (例如 8751) 编程写入时, 此引脚作为编程脉冲输入端。

PSEN: (29 脚) 程序存储器允许输出控制端。在单片机访问外部程序存储器时, 此引脚输出的负脉冲作为读外部程序存储器的选通信号。在芯片上电运行的情况下, 用示波器测得如果此引脚有脉冲输出, 则证明单片机可以正常访问外部程序存储器, 否则将不能访问。

EA/VPP: (31 脚)

EA 是内外程序存储器的选择控制端, EA 端为高电平时选择内部程序存储器, 低电平时选择外部程序存储器。

VPP 为第二功能, 在 flash 编程期间, EA 也得接 VCC 。

VCC : (40 脚) 电源

GND: (20 脚) 地

XTAL1: (19 脚) 接外部晶振的一个引脚, 单片机就是根据晶振提供的时钟脉冲来执行每一条指令。

XTAL2: (18 脚) 接外部晶振的另一个引脚。(图 4-9 为常用的功能电路)

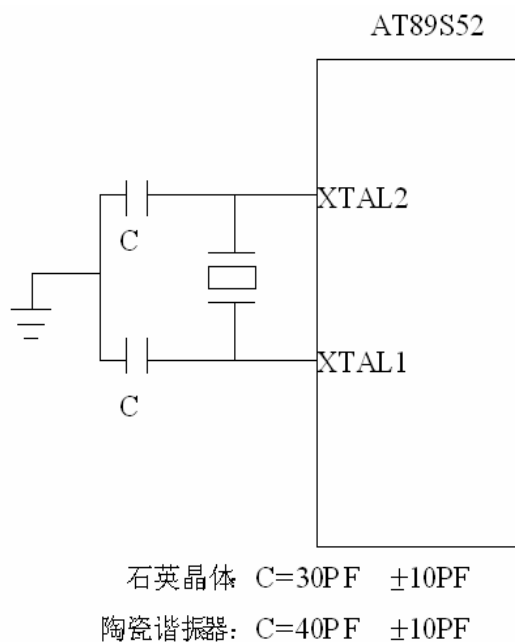


图 4-9

经过前面一系列的讲解，我们已经有了大致的了解。但这样还是不够的，我们必需通过大量的实验，来加强自己的动手能力。

动手实验（1）：

实验目的：了解端口的使用。


实验内容：用 SSH_51MCU 实验板做以下的实验，下面的程序非常的简单，但是它可以让我们很容易地去理解端口的使用。


```
sfr P1 = 0x90; //定义 P1 端口的字节地址
void main(void)
{
    P1=0x00; //对端口赋予低电平，点亮端口；
}
```

步骤：1. 打开光盘第 4 章/ PORT / PORT.Uv2 工程文件，对程序进行编译，链接，调试产生 PORT.hex 烧写文件。

2. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

3. 将 SSH_51MCU 实验板通上电源。

4. 在图 2-18 的对话框中设置为硬件仿真。点击  按钮将波特率设置为 38400。

5.用  进行调试，观察当黄色的光标执行到“P1=0x00;”的时候,在实验板中引起的变化。如图 4-10。

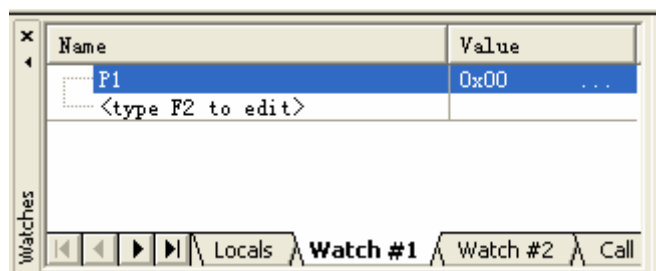


图 4-10

实验分析:

从上面的实验程序我们可以看出。当黄色的光标执行到“P1=0x00;”的时候，观察窗口中的“P1”由原来的 0xFF 变为了 0x00。但是在此之前，我们要留意到程序的开始有这么一条语句定义，“sfr P1 = 0x90; //定义 P1 端口的字节地址”，可能有些初学者会问，这是什么意思啊！在上面我们介绍端口的引脚功能时就已经讲过，P1 口的字节地址为 0x90, 位地址为 0x90 至 0x97，意思就是说，主要我们在程序中对其地址进行写数，它就会根据我们的数据进行相应的动作，“sfr”是 keil 扩充的关键字，其功能是声明一个特殊的功能寄存器，“sfr P1 = 0x90”这句语句当中，我们就是将“0x90”定义为一个特殊功能寄存器“P1”。也就是说对单片机端口的操作，就是对地址进行操作。当然啦，我们把“P1”改为“prot”可以吗！答案是可以的，无论你怎么改程序运行时只是对“0x90”操作，而不是对“port”进行操作。这样定义只是为了让程序有更高的可读性。下面我们就来把“P1”改为“port”试试结果是如何！程序如下。

```
sfr port = 0x90; //定义 port 端口的字节地址
void main(void)
{
    port = 0x00; //对端口赋予低电平，点亮端口;
}
```

对程序进行编译，链接，调试产生 PORT.hex 烧写文件。经过试验，我们在实验板上看到的结果跟原来的是一样。到了现在，你可能会想问一个题？为什么之前的程序，我们没有这样定义，又可以点亮二极管啊！如果你细心一点就会发现，其实是有定义的，在之前的程序当中有“#include<reg52.h>”文件包含，在这个文件里有一句这样的定义语句“sfr P1 = 0x90;”，所以我们可以直接用，而不用再次定义。

动手实验（2）：

实验目的：了解端口位定义的使用。

实验内容：用 SSH_51MCU 实验板做以下的实验。下面的类同上面动手实验（1），程序同样非常的简单，但是它可以让我们很容易地去理解位地址的使用。

```
sbit LED = 0x90^0;//对 P1 端口的第 0 位进行位定义
void main(void)
{
    LED=0;//对 P1 端口的第 0 位赋予低电平，点亮；
}
```

步骤：1. 打开光盘第 4 章/ PORT_BIT / PORT_BIT.Uv2 工程文件，对程序进行编译，链接，调试产生 PORT_BIT.hex 烧写文件。

3. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

2. 将 SSH_51MCU 实验板通上电源。

4. 在图 2-18 的对话框中设置为硬件仿真。点击 **Settings** 按钮将波特率设置为 38400。

5. 用 **F7** 进行调试，观察当黄色的光标执行到“LED=0;”的时候，在实验板中引起的变化。如图 4-11。

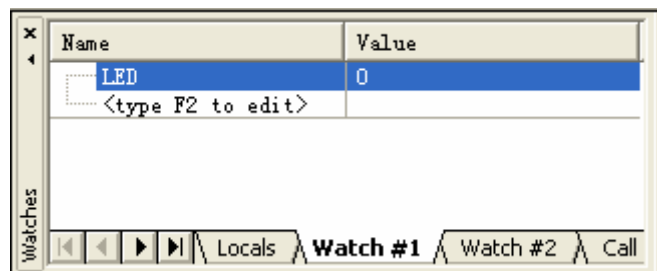


图 4-11

从上面的实验程序我们可以看出。当黄色的光标执行到“LED=0;”的时候，观察窗口中的“LED”由原来的 1 变为了 0。但是在此之前，我们同样是用“sbit LED = 0x90^0;”进行了定义。其功能与上面动手实验（1）完全一样的，只不过“sbit”是定义位操作的，而“sfr”是定义字节的。P1 口有位端口，如果想对某一端口进行位操作，我们可以通过相同的方法定义，如要对第 5 个端口进行位操作可以这样定义：“sbit LED = 0x90^5;” 试试结果是如何！程序如下。

```
sbit LED = 0x90^5;//对 P1 端口的第 5 位进行位定义
void main(void)
{
    LED=0;//对 P1 端口的第 5 位赋予低电平，点亮；
}
```

对程序进行编译，链接，调试产生 PORT.hex 烧写文件。经过试验，我们在实验板上看到的结果是点亮了第 5 只二极管。

自我练习：

1. 在实验板中的 P2.7 端口接了一个蜂鸣器，自行编写一个程序，工程名命名为“myprot”，利用“sfr”关键字定义 P2.7 端口，使其每 100 毫秒响一次，从而发出“嘟”“嘟”“嘟”的响声。

程序设计思路：

- (1) 如上面动手实验 (1) 用“sfr”定义 0xa0 地址。
- (2) 编写一个可以延时 100ms 的延时函数。
- (3) 每 100ms 改变接蜂鸣器的端口一次。
- (4) 不断死循环。

2. 自行编写一个程序，利用“sbit”关键字定义 P2.7 端口进行位操作，使其每 100 毫秒响一次，从而发出“嘟”“嘟”“嘟”的响声。

程序设计思路：

- (1) 如同动手实验 (2) 用“sbit”定义 0xa0 的第 7 个端口。
- (2) 编写一个可以延时 100ms 的延时函数。
- (3) 每 100ms 改变 P2.7 端口的状态一次。
- (4) 不断死循环。

以上的练习答案附光盘中。

第 2 节 中断系统

什么是“中断”？

中断的这个概念是很容易理解的，我们先来打个比喻。小强正在大厅看录象电视的时候，当门口有客人来访的时候，他会把电视暂停下来，去看一看是那位客人来访，把他请进门，然后再回到大厅，从刚才暂停的地方开始再看电视。但是在再次回来看电视的时候，妈妈在厨房做饭，要求小强帮手，那他只好再次把正在观看的电视暂停，去帮手做饭，在帮忙做完饭之后，再次回来从刚才暂停电视剧情中开始观看。在这整个过程中，放下当前的事，而去开门请客人，帮手做饭，这就叫中断。

单片机也是一样的，正常情况下 CPU 会执行主程序，但是如果有中断事件的发生，它就会把当前的事件保存起来，去执行中断程序，当执行完中断程序之后，再回来原来主程序的程序段中开始执行。图 4-12 为中断响应的流程。

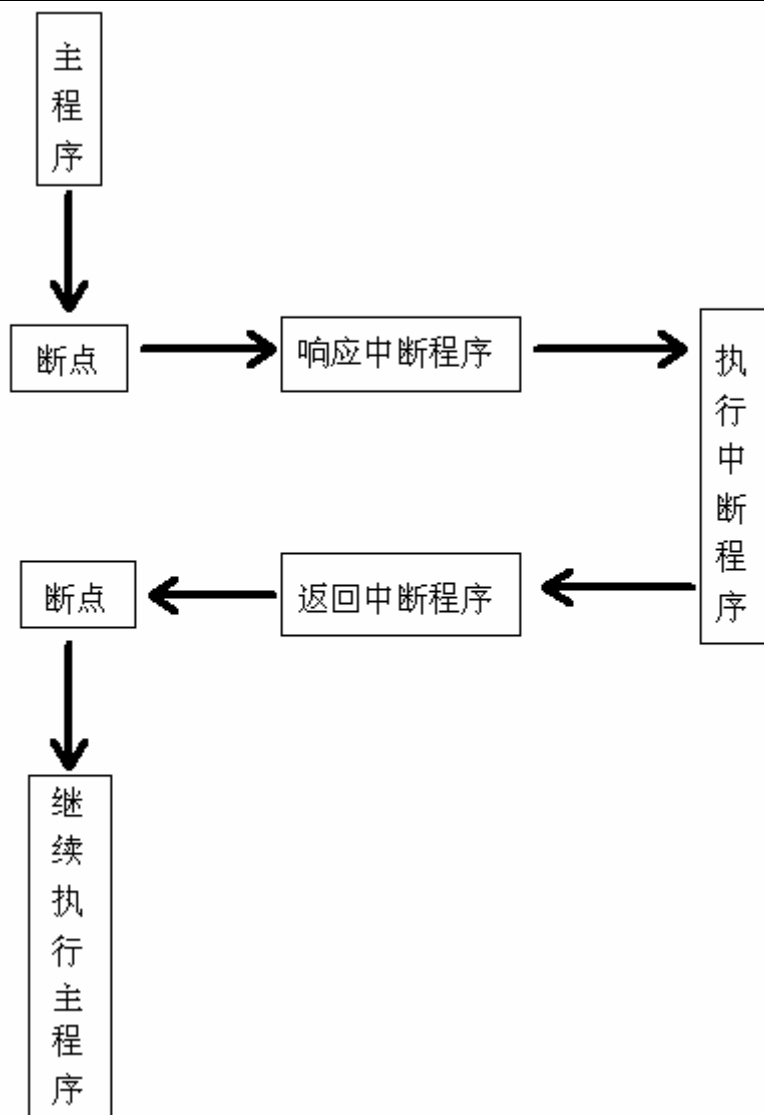


图 4-12 (中断流程)

在典型的 51 系列单片机中有 5 个中断源，各有各的入口地址。而且它们具有不同的优先级，所谓优先级就是当有两个或以上的中断源同时响应时，CPU 会先执行那一个中断源，此为优先级，具体如下表 4-3 所示。

表 4-3 (入口地址表)

中断号 n	优先级	中断源	中断入口地址
0	1(最高)	外部中断 0 (INT0 P3.2)	0003H
1	2	定时器 0	000BH
2	3	外部中断 1 (INT1 P3.3)	0013H
3	4	定时器 1	0018H
4	5 (最低)	串行口	0023H

在 AT89S52 单片机中，有 4 个寄存器是供用户对中断进行控制的，这 4 个寄存器分别是定时器控制寄存器 TCON，串行口控制寄存器 SCON，中断允许控制寄存器 IE，以及中断优先控制寄存器 IP。这 4 个寄存器都是属于专用寄存器，可完成中断请求标志寄存，中断允许管理和中断优先级的设定，由它们所构成的中断系统如图 4-13 所示。

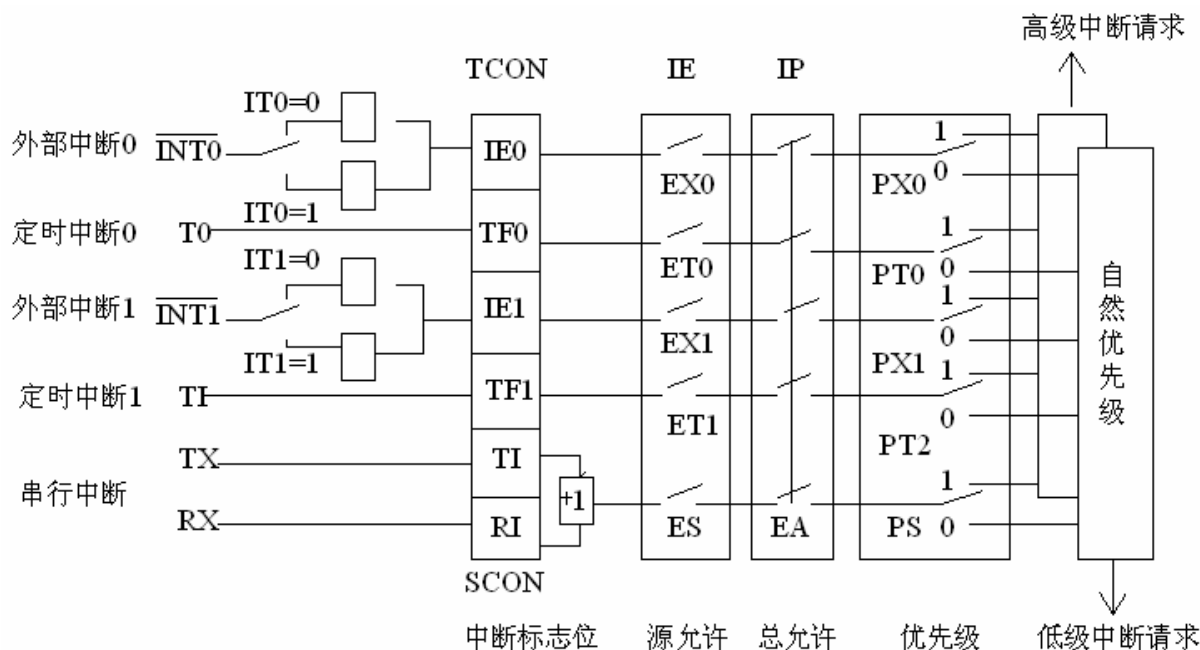


图 4-13

下面分别来介绍图 4-13 的各个控制功能。

1 定时器控制寄存器 TCON

位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
位名称	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF0 (TF1) 内部定时器/计数器 0 (定时器/计数器 1) 溢出中断标志位。

当片内定时器/计数器 0 (定时器/计数器 1) 计数溢出的时候，由单片机自动置 1，而当进入了中断服务程序之后再由单片机自动清 0。

IE0 (IE1) 外部中断请求标志位。

当 INT0 (或 INT1) 引脚出现有效的请求信号，此位由单片机置 1，而当进入了中断服务程序之后再由单片机自动清 0。

IT0 (IT1) 外部中断触发方式控制位。

IT0 (IT1) =1 脉冲触发方式，下降沿触发有效。

IT0 (IT1) =0 电平触发方式，低电平有效。

2 串行口控制寄存器 SCON

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位名称	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

TI 串行口发送中断标志位

当单片机串行口发送完一帧数据后，此位由单片机置 1，而当进入了中断服务程序之后是不会自动清 0 的，必需由用户在中断服务程序中用软件清 0。

RI 串行口接收中断标志位

当单片机串行口接收完一帧数据后，此位由单片机置 1，而当进入了中断服务程序之后是不会自动清 0 的，必需由用户在中断服务程序中用软件清 0。

3 中断允许控制寄存器 IE

位地址	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
位名称	EA	—	—	ES	ET1	EX1	ET0	EX0

EA 中断允许总控制位

EA=0 关闭总中断
EA=1 启动总中断，

当启动了总中断后，再由各中断源的中断允许控制位进行设置。

ES 串行中断允许控制位

ES=0 关闭串行中断
ES=1 启动串行中断

EX0 (EX1) 外部中断允许控制位

EX0 (EX1) =0 关闭外部中断 0 (外部中断 1)
EX0 (EX1) =1 启动外部中断 0 (外部中断 1)

ET0 (ET1) 定时中断允许控制位

ET0 (ET1) =0 关闭定时中断 0 (定时中断 1)
EX0 (EX1) =1 启动定时中断 0 (定时中断 1)

4 中断优先级控制寄存器 IP

位地址	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H
位名称	——	——	——	PS	PT1	PX1	PT0	PX0

PX0 (PX1) 外部中断 0 (外部中断 1) 优先级设定位

PX0 (PX1) = 1 外部中断 0 (外部中断 1) 定义为最高优先级中断。

PX0 (PX1) = 0 外部中断 0 (外部中断 1) 定义为最低优先级中断。

PT0 (PT1) 定时中断 0 (定时中断 1) 优先级设定位

PT0 (PT1) = 1 定时中断 0 (定时中断 1) 定义为最高优先级中断。

PT0 (PT1) = 0 定时中断 0 (定时中断 1) 定义为最低优先级中断。

PS 串行通信优先级设定位

PS=1 串行通信定义为最高优先级中断。

PS=0 串行通信定义为最低优先级中断。

如果在同时收到几个同一优先级的中断请求时，哪一个中断请求优先得到响应，取决于内部的查询顺序，其查询顺序如下。

外部中断 0 → 定时器 0 中断 → 外部中断 1 → 定时器中断 1 → 串行中断

Keil c 编译器支持在 C 语言中直接定义中断程序，因此减轻了用汇编语言开发中断程序的烦琐过程，以下是一个合法的中断函数定义。

函数类型 函数名 (形式参数表) interrupt n

在 interrupt 后面的 n 为中断号，请参照表 4-3 (入口地址表)

例如：以下是一个外部中断 1 的中断响应程序。

```
void INT_1(void) interrupt 2
{
    程序语句
}
```

当正确定义了中断程序之后，程序在运行时，若有中断信号的请求，其 CPU 就会自动进入中断程序执行其代码。在以下的几个章节中会介绍 AT89S52 中的各个中断资源。

第 3 节 外部中断 1

动手实验 (3):

实验目的：了解中断寄存器的初始化和如何定义一个中断程序。

实验内容：用 SSH_51MCU 实验板做以下的实验。在做以下实验的时候，读者要了解 CPU 什么时候才会进入中断程序。图 4-14 是实验板的电路，单片机默认为高电平，K1 的一端接 INT1 引脚，而另一端接电源地，当 K1 被按下的时候 INT1 引脚被拉为低，从而出现一个下降沿，在程序中我们初始化的时候已经设定为下降沿有效，也就是说当 INT1 引脚出现一个下降沿的时候，就会触发中断而进行中断程序。

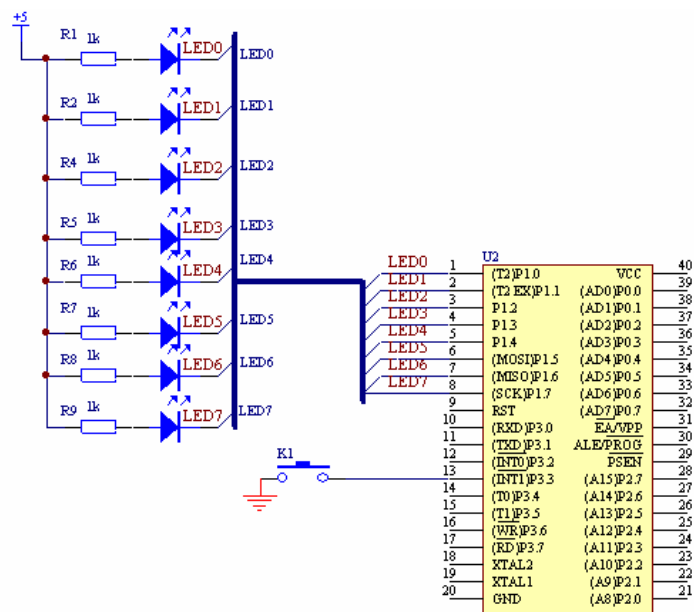



图 4-14

```
#include<reg52.h>//文件包含
void main(void)
{
//以下的三句语句为外部中断 1 的寄存器初始化
    IT1=1;//外部中断 1，脉冲触发方式，下降沿触发有效
    EX1=1;//启动外部中断 1
    EA=1;//启动总中断
    while(1);;//等待中断的发生
}
void INT_1(void) interrupt 2//中断程序
{
    EX1=0;//关闭外部中断 1
    P1=~P1;//读取 P1 口的状态，将其取反，再赋值于 P1 口
    while (P3&0x08==0x00) ;//等待按键松开
    EX1=1;//启动外部中断 1
}
}
```

步骤：1. 打开光盘第 4 章/ INT_1 / INT_1.Uv2 工程文件，对程序进行编译，链接，

调试产生 INT_1.hex 烧写文件。

2. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。
3. 将 SSH_51MCU 实验板通上电源。
4. 在图 2-18 的对话框中设置为硬件仿真。点击 **Settings** 按钮将波特率设置为 38400。
5. 按图 4-15 所示，得到了如图 4-16 的寄存器框图。

6. 用  进行调试，观察当黄色的光标在主程序中执行每一条语句的时候，同时在图 4-15 中寄存器对应的变化。

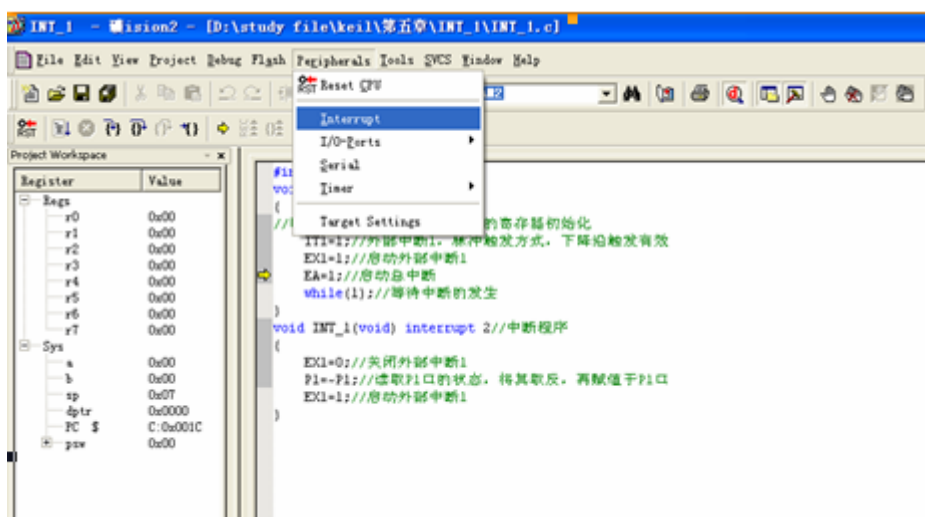


图 4-15

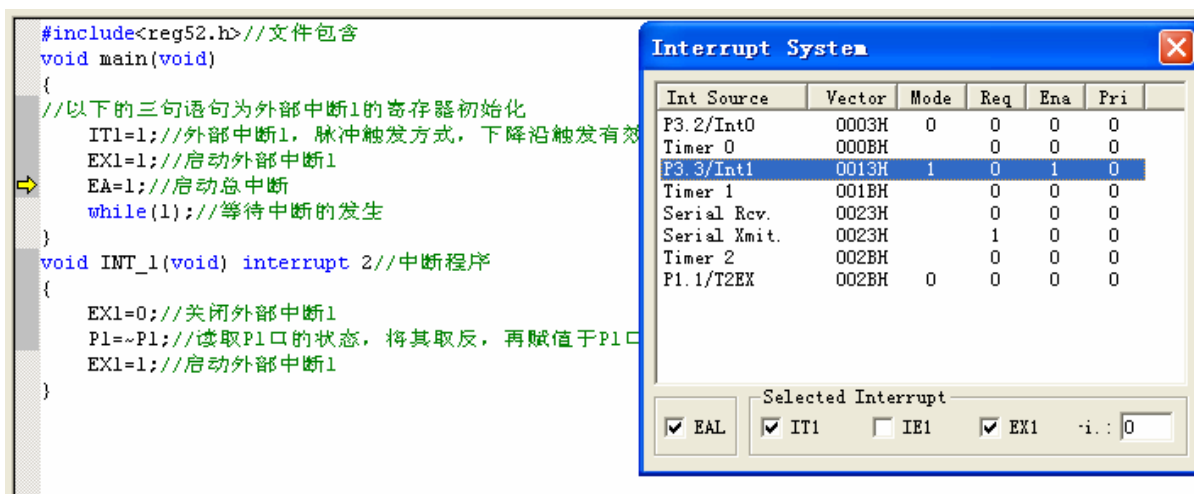



图 4-16

7. 按  全速执行，用手按实验板中的 K1 按键，当第一次被按下的时候，P1 口被点亮，当再一次被按下的时候，P1 口被熄灭。实验板会根据按键按下而不段的变化。从上面的实验程序我们可以看出。当单片机执行完三条初始化语句之后，就不停地在等待，等待中断信号的请求，从而执行中断程序，当执行完中断程序后，又返来主程序，继续等待，不断地循环。

自我练习：

自行编写一个程序，工程名命名为“myint_1”，单片机不停地在主程序中每 200 毫秒循环点亮 LED0，但是如果有中断信号的请求（即 K1 被按下），单片机就会进入中断程序点亮 LED1。而每一次按键被按下了则进入中断程序将原来的状态取反。

程序设计思路：

- (1) 包含所用单片机对应的头文件。
- (2) 利用关键字“sbit”定义两只二极管 LED0、LED1。
- (3) 定义一个延时函数。
- (4) 设置外部中断 1 为防冲触发方式下降沿有效，开启外部中断，开启总中断。
- (5) 在主程序中每 200ms 点亮 LED0。
- (6) 利用关键字“interrupt”定义一个中断函数。
- (7) 当进入中断程序之后关闭外部中断，取反 LED1，再开外部中断。

（以上的练习答案附光盘中）

第 4 节 外部中断 0

动手实验（4）：

实验目的：了解外部中断 0 寄存器的初始化和如何定义一个中断程序。

实验内容：在主程序中以流水灯的方式点亮二极管，当有中断信号的请求时，就进行中断程序，点亮 P1 口的所有 LED 灯。图 4-17 是实验板中的电路图，只要 K0 被按下就会出现一个低电平，从而出现脉冲的下降沿，触发 INTO 引脚，使 CPU 进入中断程序。

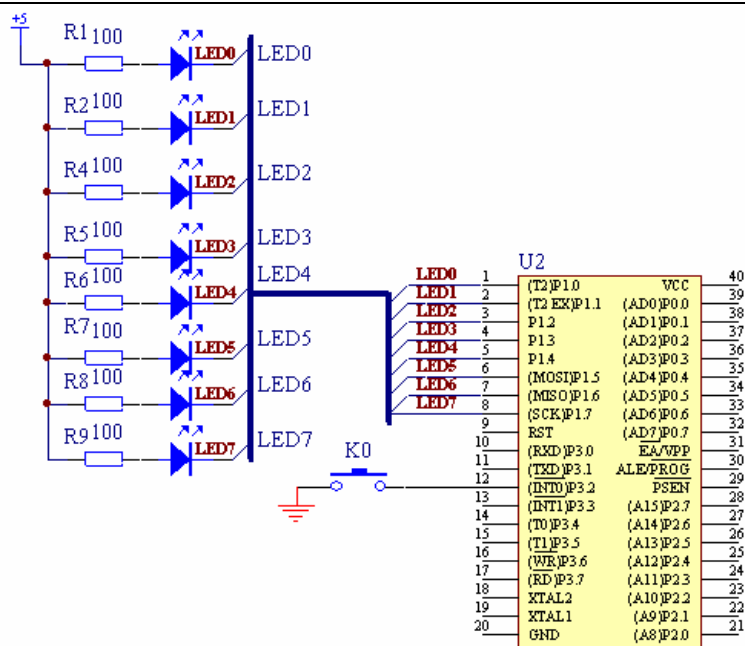


图 4-17

```

#include<reg52.h>//包含所用单片机对应的头文件
#define uchar unsigned char//宏定义
uchar temp;// 声明一个全局变量来暂存数据
void delay_ms(unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
    uchar loop;
    //以下的三句语句为外部中断 0 的寄存器初始化
    IT0=1;//脉冲触发方式, 下降沿触发有效
    EX0=1;//启动外部中断 0
    EA=1;//启动总中断

    while(1)//死循环, 等待中断的发生
    {
        P1=0xfe;//点亮第一只二极管
        delay_ms(200);//延时 200ms
        for(loop=0;loop<8;loop++)//循环 8 次
    }
}
    
```


```
        {  
            P1=(P1<<1)|0x01;//读 P1 口的数据，左移一位，并且保持第 1 位为 1  
            delay_ms(200);//延时 200ms  
        }  
  
    }  
}  
void INT_0(void) interrupt 0//中段程序  
{  
    EX0=0;//关闭外部中断 0  
    temp=P1;//进入中断程序之后，暂时存放 P1 口当前值。  
    P1=0x00;//点亮 P1 口的全部二极管  
    delay_ms(1000);//延时 1000ms  
    P1=0xff;//熄灭 P1 口的全部二极管  
    P1=temp;//将刚才暂存放的数据返还给 P1 口  
    EX0=1;//启动外部中断 0  
}
```


步骤：1、打开光盘第 4 章/ INT_0 / INT_0.Uv2 工程文件，对程序进行编译，链接，

调试产生 INT_0.hex 烧写文件。

2、把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

3、将 SSH_51MCU 实验板通上电源。

4、在图 2-18 的对话框中设置为硬件仿真。点击  按钮将波特率设置为 38400。

5、用  进行调试，观察当黄色的光标在主程序中执行每一条语句的时候，同时在图 4-18 中寄存器对应的变化。

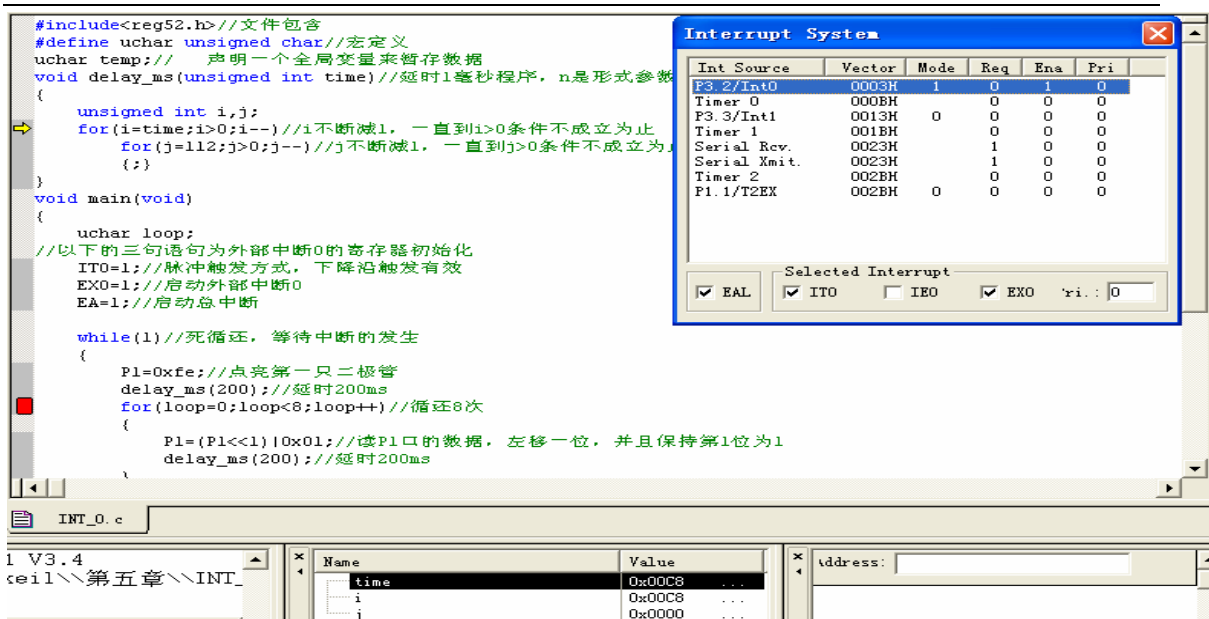



图 4-18

7. 按  全速执行，会看到 P1 口的二极管在不停地移动，当用手按下实验板中的 K0 按键，单片机会响应中断程序，进入中断程序之后会先将当前状态存放好，然后点亮 P1 口的 LED 灯，延时 1000MS 之后熄灭，再将暂存的数据返还给 P1 口，退出中断程序后继续移动 LED 灯。不断地循环。

自我练习：

自行编写一个程序，工程名命名为“myint_0_1”，在主程序中每 200ms 闪动八只二极管，当 K0 按下点亮低 4 位的 4 只二极管 1 秒，当 K1 按下点亮高 4 位的 4 只二极管 1 秒。

程序设计和思路：

- (1) 包含所用单片机对应的头文件。
- (2) 定义一个延时函数。
- (3) 外部中断 0 设为下降沿触发有效，开启外部中断 0，用同样的方法设置外部中断 1，最后开启总中断。
- (4) 主程序中每 200ms 闪动一次二极管。
- (5) 定义一个外部中断 0 的中断程序，每次进入中断程序就点亮低 4 位的 LED 1 秒。
- (6) 定义一个外部中断 1 的中断程序，每次进入中断程序就点亮高 4 位的 LED 1 秒。

(以上练习答案附光盘中)

第 5 节 定时器/计数器

什么是计数?

所谓计数是指对外部事件进行计数，外部事件的发生以输入脉冲的方式表示，因此计数功能的实质就对外来脉冲进行计数，AT89S52 单片机有 T0 (P3.4) 和 T1 (P3.5) 两个信号引脚，分别是这两个计数器的计数输入端。外部输入的脉冲在负跳变时有效，进行计数器加 1。

什么是定时?

定时器是通过计数器的计数来实现的，不过此时的计数脉冲来自单片机内部，因此定时器功能实质就是对单片机内部脉冲的计数，即每个机器周期产生一次计数脉冲，也就是每个机器周期计数器加 1。

AT89S52 单片机的定时器/计数器具有 4 种工作方式, 分别为: 方式 0、方式 1、方式 2、方式 3、只需通过寄存器的设置就可以方便地选择不同的工作方式。下面我们分别来讲解这 4 种不同的工作方式。

1 工作方式控制寄存器 TMOD

位号	D7	D6	D5	D4	D3	D2	D1	D0
符号	GATE	C/T	M1	M0	GATE	C/T	M1	M0

TMOD 的低半字节用来配置定时器/计数器 0，高半字节用来配置定时器/计数器 1。

GATE 门控位

GATE=1. 定时/计数器的运行受外部引脚输入电平的控制，即 INTO 控制 T0 运行，INT1 控制 T1 运行。

GATE=0. 定时/计数器的运行不受外部引脚输入电平的控制。

C/T 计数器模式和定时器模式选择位

C/T=1. 选择计数器模式，计数器对外部输入引脚 T0 (P3.4) 或 T1 (P3.5) 的外部脉冲计数。

C/T=0. 选择定时器模式。

M1 M0. 工作方式选择位

表 4-4 (定时/计数器方式选择)

M1 M0	工作方式	功能
0 0	工作方式 0	13 位计数器
0 1	工作方式 1	16 位计数器
1 0	工作方式 2	自动再装入 8 位计数器
1 1	工作方式 3	定时器 0: 分成两个 8 位计数器 定时器 1: 停止计数

定时/计数器的工作方式(下面我们以定时/计数器 0 来进行介绍)

1 工作方式 0

当 M1、M0 为 00 时，定时器/计数器工作于工作方式 0，工作方式 0 是 13 位计数器，其计数器由 TH0 全部 8 位和 TL0 的低 5 位构成，TL0 的高 3 位为未用，图 4-19 为工作方式 0 的逻辑框图。

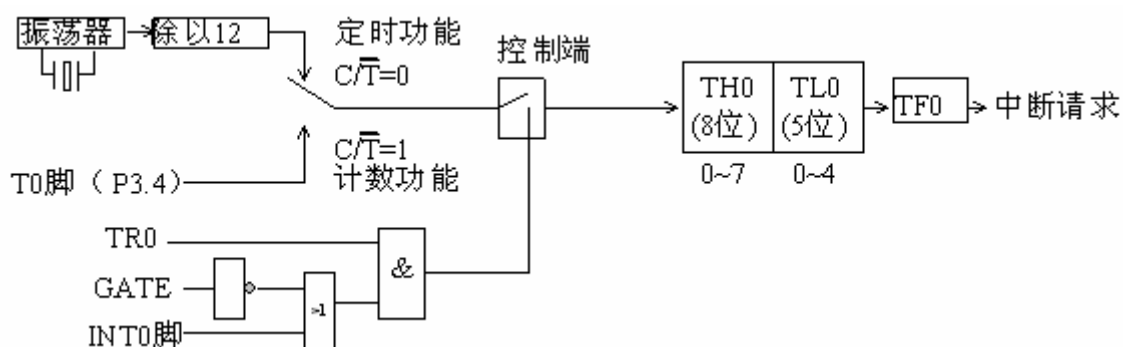


图 4-19 (工作方式 0)

工作方式 0 是 13 位计数器，其最大计数为“1 1111 1111 1111”=8192，也就是说，每次计数到 8192 都会产生溢出，置位 TF0。但是在实际应用中，经常有少于 8192 个计数值的要求。例如，在编写程序时要求计数满 1200 溢出中断，在这种情况下，计数就不应该从 0 开始了，而是应该从一个固定数值开始，那么这个数值是多少啊？上面我们要求 1200 溢出中断一次，那么只要用 8192-1200=6992，将 6992 作为初值赋给计数器，当计数器从 6992 开始计数，经过 1200 个计数脉冲，就到了 8192 产生了溢出。以下为工作方式 0 的计算公式。

$$\text{要定时时间} = (8192 - X) \times (12 \div \text{晶振频率})$$

定时时间的单位为：(微秒 us)

晶振频率的单位为：(MHZ)

下面我们用例子来理解这条公式，根据上面的分析，现在来计算定时 2ms 应该如何计算，由于实验板的晶振为 11.0592MHZ, 需要定时 2ms 也就是 2000us, 然后我们把其参数代入公式：

$$2000 = (8192 - X) \times (12 \div 11.0592)$$

结果 $X=6439$

十六进制 $X=0x18CC$

将这 13 位初值填入 TH0 和 TL0 中。注意，TL0 只用了低 5 位，高 3 位没有用到，填入 0。这时装入 TH0 和 TL0 的初值为如下：

1 1000 XXX0 1100

则 $TH0=0x18$ $TL0=0x0c$, 只要把这个初值赋给了定时器 0, 则定时器就每 2ms 溢出一次, 将计数溢出标志位置 1, 触发中断。大家要记住, 定时器工作方式 0 是没有自动重装功能的, 为了使下一次定时的时间不变, 需要每当定时器溢出之后, 要马上再赋初值给 TH0 和 TL0, 否则定时器就会从“0”开始计数, 这样就不准确了。

2 工作方式 1

当 M1、M0 为 01 时, 定时器/计数器工作于方式 1, 这时定时器/计数器的等效电路如图 4-20。

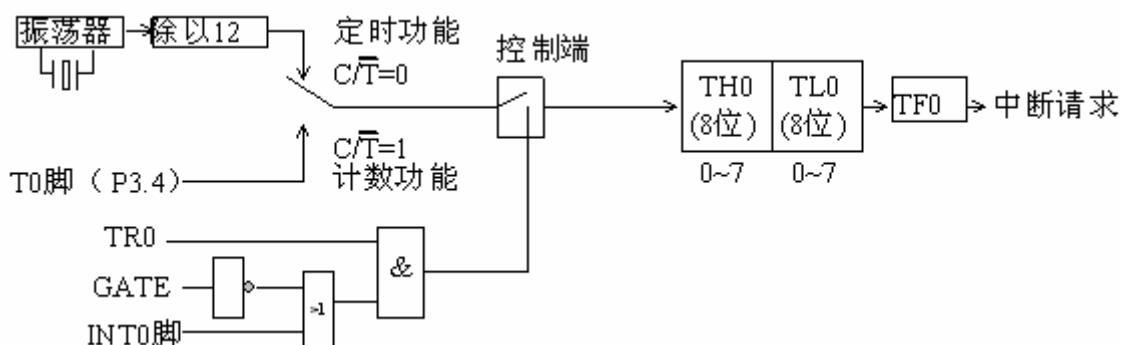


图 4-20

工作方式 1 与工作方式 0 的操作是完全相同的, 只是工作方式 1 是 16 位计数器, 而工作方式 0 是 13 位计数器。以下是工作方式 1 定时时间的计算公式。

$$\text{要定时时间} = (65536 - X) \times (12 \div \text{晶振频率})$$

因此每次计数到 65536 就会产生溢出, 去置位 TF0。

3 工作方式 2

工作方式 0 与工作方式 1 若用于循环计数, 每次计数到溢出的时候都必需在程序中利用软件重装定时的初值, 否则就会造成计算的不准确。但是在重装的同时必需花费一定的时间, 这样就会造成定

时时间有误差的情况，若果用于一般定时那是无关紧要的，但如果对定时要求非常严格的情况下，这样是不允许的。下面我们来介绍第三种定时工作方式，就是工作方式 2，下图 4-21 是工作方式 2 的逻辑功能框图。

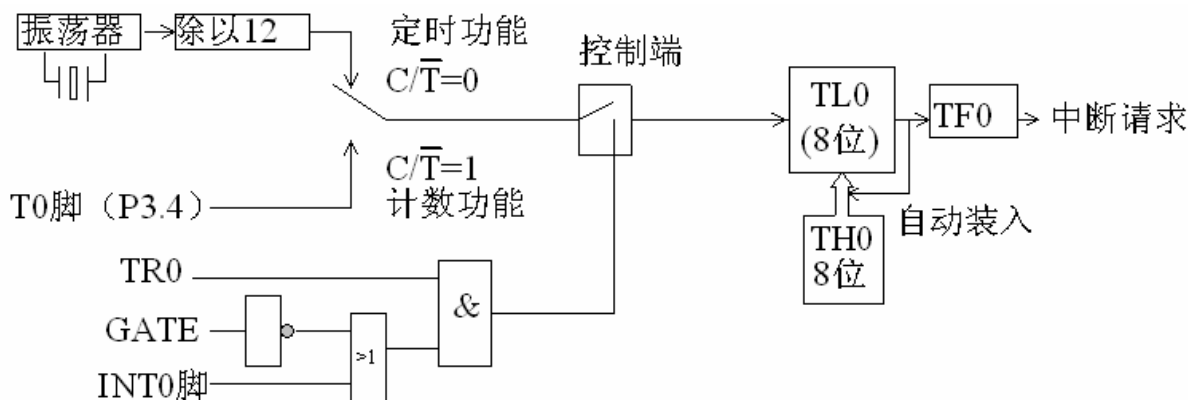


图 4-21

由图 4-21 可以清楚地看到，工作方式 2 与前面介绍的两种定时器唯一不同的就是定时器的低 8 位是用作定时的计数，当计数溢出的时候高 8 位就用作自动重装初值，赋值于低 8 位，因为是有硬件重装的功能，所以在每次计数溢出的时候，无需用户在程序当中利用软件去重装，这样就不但省去了程序中的重装指令，而且也有利于提高定时器的精确度。因为工作方式 2 只有 8 位数结构，所以计数十分有限，以下是工作方式 2 的定时计算公式：

$$\text{要定时时间} = (256 - X) \times (12 \div \text{晶振频率})$$

4 工作方式 3

工作方式 3 的结构较为特殊，只能用于定时器 0，如果强制用于定时器 1，就等同于 TR1=0，把定时器 1 关闭。

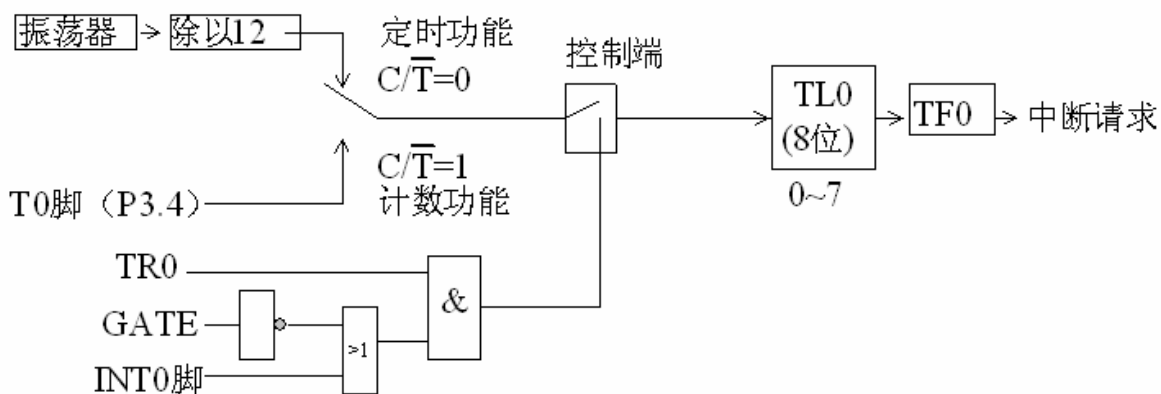


图 4-22

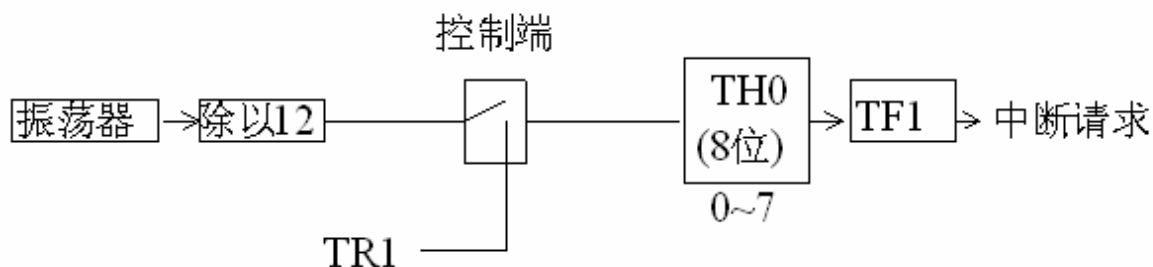


图 4-23

从图 4-22 与图 4-23 我们可以清楚地看到,在工作方式 3 的模式下它被拆分为两个独立的计数器 TL0 与 TH0。图 4-22 是拆分出来的 8 位定时/计数器,其使用跟前面介绍的几种工作方式是完全相同的,大家可以参考前面所讲的文段。

而图 4-23 只能用作简单的定时器使用。而且由于定时/计数器 0 的控制位已经被 TL0 占用,所以只好借用定时/计数器 1 的控制位 TR1 和 TF1。即以计数溢出去置位 TF1,而定时的启动和停止则受 TR1 所控制。以下为工作方式 3 的定时计算公式:

$$\text{定时时间} = (256 - X) \times (12 \div \text{晶振频率})$$

动手实验 (5)

实验目的:学会定时器工作方式 1 的使用。

实验内容:利用定时器 0 溢出中断的方式,每 5ms 中断一次,每中断 200 次点亮 P1 口的 8 只二极管,也就是每 1 秒点亮一次,由于实验板的晶振为 11.0592MHz,所以根据下面的计算公式,我们来计算定时的时间:

$$\text{定时时间} = (65536 - X) \times (12 \div \text{晶振频率})$$

$$5000 = (65536 - X) \times (12 \div 11.0592)$$

$$X = 60928$$

十六进制为 0xee00

把 0x00 赋给定时的低 8 位,而 0xee 就赋给定时的 8 位,程序如下。

```
#include<reg52.h>//包含所用单片机对应的头文件
#define uchar unsigned char
uchar time=200;
void main(void)
{
    TMOD=0X01;//设定于工作方式 1
    TH0=0xee;//给定时器赋 5000 微秒溢出初值
```

```
TL0=0x00;
ET0=1;//允许定时 0 中断
EA=1;//允许总中断
TR0=1;//启动定时器 0

while(1);//等待定时器的溢出


}
void TIME0(void) interrupt 1//中段程序
{
    TH0=0xee;//为了保证每次定时时间的准确，溢出后需从新赋值
    TL0=0x00;
    ET0=0;//关闭定时中断 0
    time--;//定时器每溢出一次变量的值减 1
    if(time==0)
    {
        P1=~P1;//取反 P1 口的全部二极管
        time=100;//变量 time 从新赋值
    }
    ET0=1;//启动定时中断 0
}
```


步骤：1、打开光盘第 4 章/ T_MODE_1 / T_MODE_1.uv2 工程文件，对程序进行编译，链接，

调试产生 T_MODE_1.uv2 烧写文件。

2、把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

3、将 SSH_51MCU 实验板通上电源。

4、在图 2-18 的对话框中设置为硬件仿真。点击  按钮将波特率设置为 38400。

5、在菜单 peripherals/timer/timer0 点击出，如图 4-24 的功能框，它是定时/计数器 0 的功能信息对话框，在里面可以清楚地看到定时器的运行情况。用  进行调试，观察当黄色的光标在主程序中执行每一条语句的时候，同时在图 4-24 中的信息对话框中对应的变化。

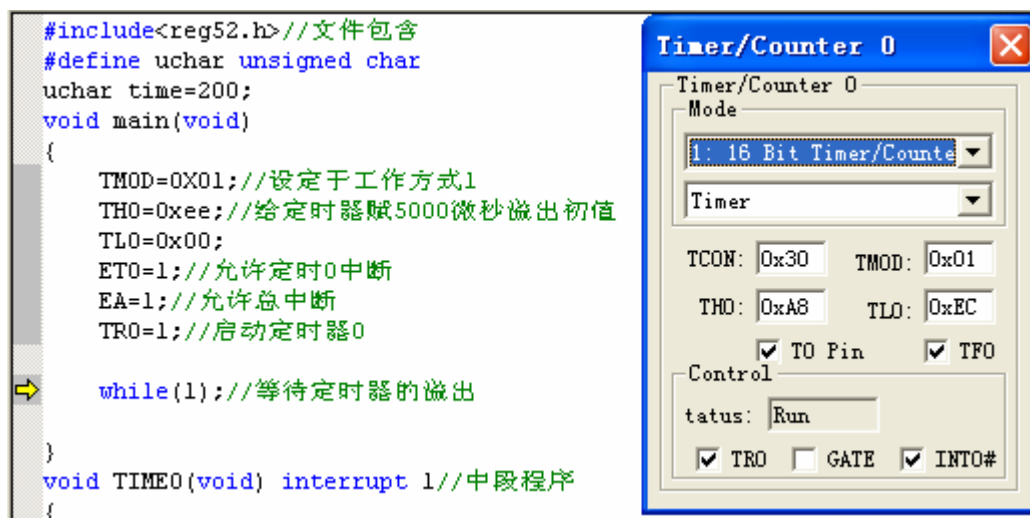




图 4-24

6. 按  全速执行，会看到 P1 口的二极管在不停在闪动，这是程序最终的运行结果，由于定时器不断溢出。导致不断地进入中断程序，从而引起 P1 口的变化。

如何测试溢出的时间是否正确：

当你利用公式计算出一个定时器初值时，到底这个定时器溢出的时间跟我们预计的是否一致呢！就上面的程序为例，我们来验证一下吧！

(1) 将工程设置为软件仿真。

(2) 把光标定于中断程序的第一句语句，同时点击  在此同时可以在寄存器观察窗口中观察到时间的变化。如图 4-25 所示。

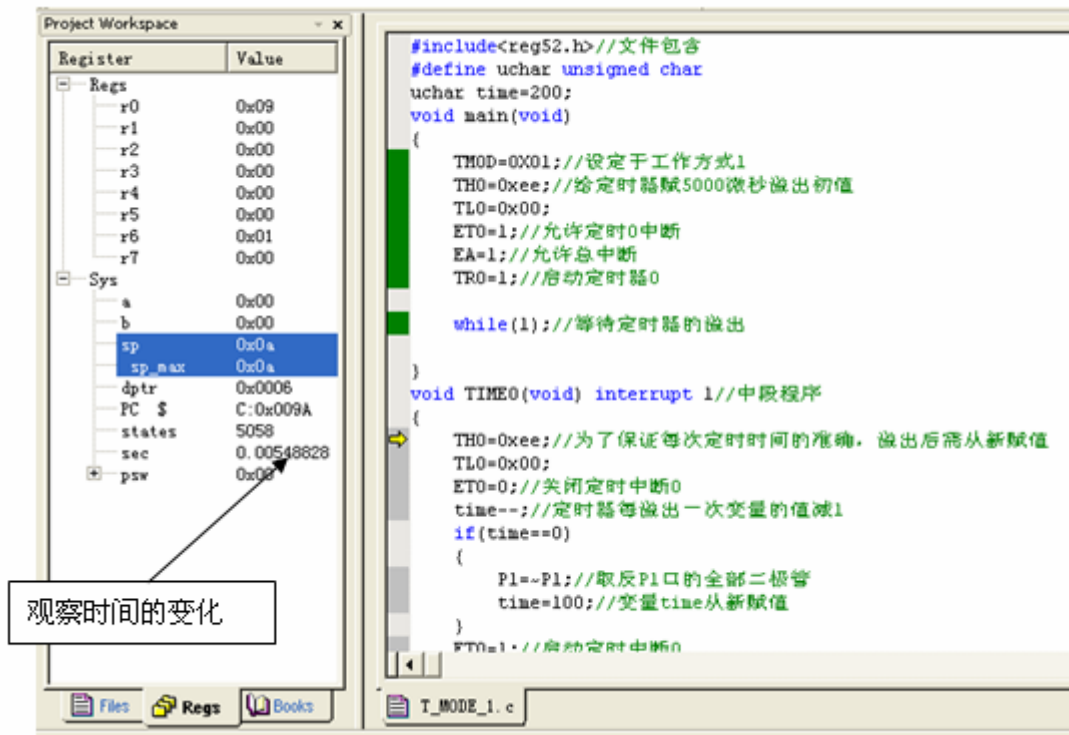


图 4-25

在前面所介绍的 4 种工作方式中，除了工作方式 2，其余的三种工作方式，中断服务程序都要进行定时初值的重装操作，因此，在定时器计数溢出到中断请求与重装初值的完成，其间总要一定时间的间隔，从而花费若干微秒的时间，这样就会导致定时时间的不准确。为了减少这种定时的误差，我们就得对重装的计数初值作适当的调整。一般情况下，使定时时间缩短 7 个至 10 个机器周期，对于本例，可由 5ms 调整为 4.99ms。

动手实验（6）

实验目的：学习 T0 计数器的使用。

实验内容：利用实验板做以下的实验。T0 每 200 微秒溢出一次，每溢出 2500 次（即 0.5 秒）点亮 LED0；T1 每 50 毫秒溢出一次，每溢出 20 次（即 1 秒）点亮 LED1。

```
#include <reg52.h> //文件包含
#include <stdio.h>
#define uchar unsigned char //宏定义
#define uint unsigned int //宏定义
sbit LED0=P1^0;
sbit LED1=P1^1;
uint count_1=2500; //每溢出一次为 200us, 2500 次合共 0.5 秒
uchar count_2=20; //每溢出一次为 50ms, 20 次合共 1 秒
void main(void)
{
```



```
/*定时器 T0 初始化*/
TMOD=0x12;//T1 使用工作方式 1, T0 使用工作方式 2
TH0=0x48;
TL0=0x48;//每 200us 溢出一次
EA=1;//开启全局中断
ET0=1;//允许计数中断
TR0=1;//开启计数器 T0

/*定时器 T1 初始化*/
TH1=0x4c;
TL1=0x00;//每 50ms 溢出一次
ET1=1;//允许计数中断
TR1=1;//开启计数器 T1
while(1);//等待计数溢出中断
}

void T0_INT(void) interrupt 1 //定时器 0 工作方式 2
{
    count_1--;//每次进入中断程序 count_1 变量减 1
    if(count_1==0)
    {
        count_1=2500;//假如 count_1 等于 0, 再将其赋值, 同时取反 LED0
        LED0=~LED0;
    }
}

void T1_INT(void) interrupt 3//定时器 1 工作方式 1
{
    /*除了工作方式 2 以外, 其它的三种工作方式在
    溢出后都要进行软件重装初值。因为工作方式 2 有
    硬件自动重装, 而其它的三种则没有。*/
    TH1=0x4c;
    TL1=0x00;//计数器赋初值
    count_2--;//每次进入中断程序 count_2 变量减 1
    if(count_2==0)
    {
        count_2=20;//假如 count_2 等于 0, 再将其赋值, 同时取反 LED1
        LED1=~LED1;
    }
}
```


步骤: 1. 打开光盘第 4 章/ COUNT / COUNT.uv2 工程文件, 对程序进行编译, 链接,

调试产生 COUNT.uv2 烧写文件。

2. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上, 另一端与电脑相接。

3. 将 SSH_51MCU 实验板通上电源。

4. 在图 2-18 的对话框中设置为硬件仿真。点击 **Settings** 按钮将波特率设置为 38400。

5. 在菜单 peripherals/timer/点击出，如图 4-27 的功能框，它是定时器/计数器 0 与定时器/计数器 1 的功能信息对话框，在里面可以清楚地看到定时器的运行情况。用  进行调试，观察当黄色光标在主程序中执行每一条语句时，在图 4-27 中的信息对话框中对应的变化。

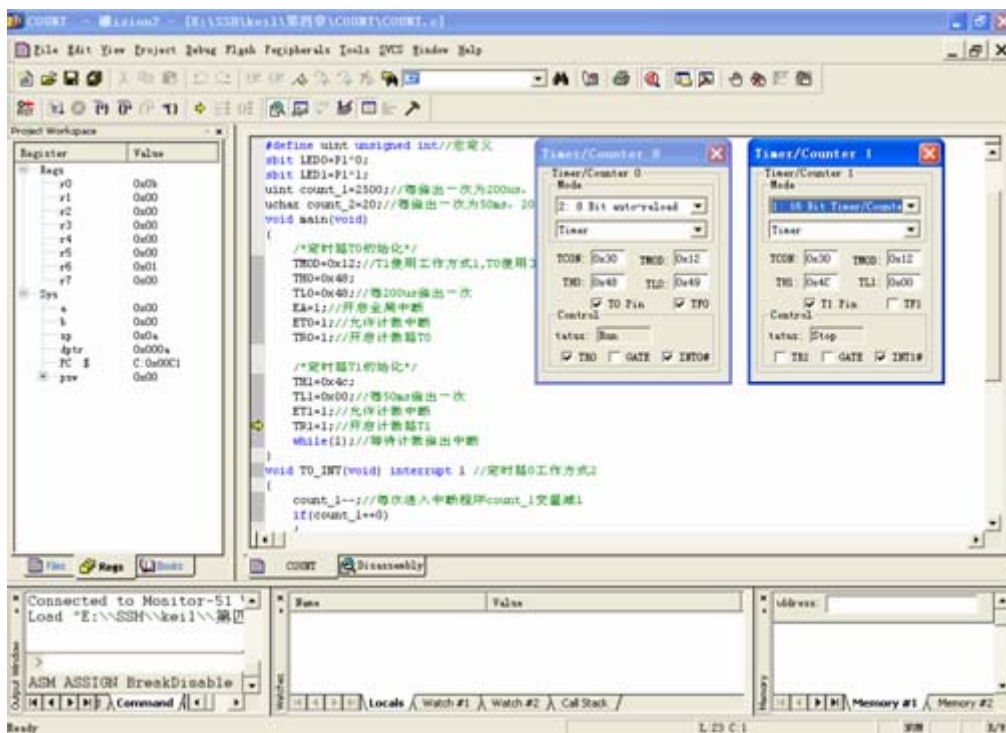


图 4-27

6. 按  全速执行，观赏实验板中 LED0、LED1 的变化。

自我练习

(1) 如图 4-30 所示为《八月桂花遍地开》的乐谱。自行建立一个工程，命名为“music”，编写其程序，利用定时器产生伴奏的音乐。其中图 4-29 为实验板的电路。

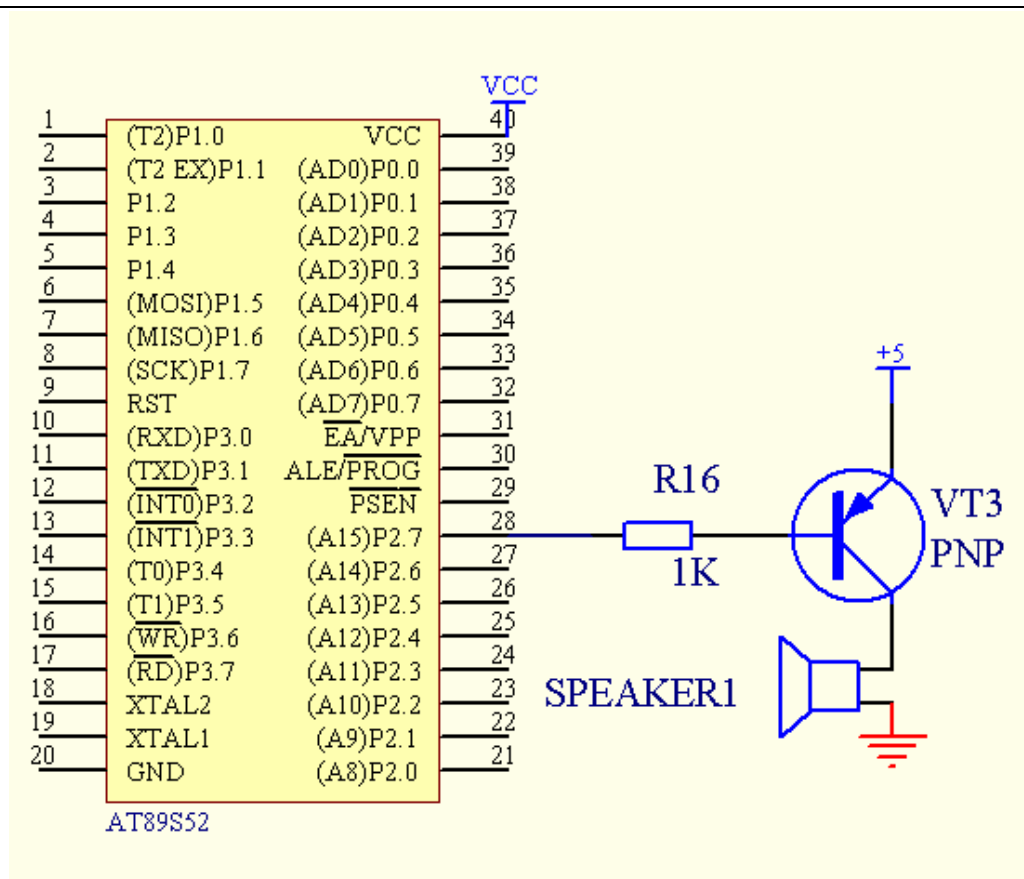


图 4-29 (实验板电路)

$\dot{1} \cdot \underline{65}$	$\underline{61561}$	$\underline{6561}$	5 -	$\underline{5 \cdot 612}$	$\underline{653}$	$\underline{5235}$	1 -
八 月	桂 花	遍 地	开,	鲜 红 的	旗 帜	竖 呀 竖 起	来,
$\underline{56153}$	$\underline{212}$	$\underline{56153}$	$\underline{212}$	$\underline{5 \cdot 612}$	$\underline{653}$	$\underline{5235}$	1 -
张 灯 又 结	彩 呀	张 灯 又 结	彩 呀,	光 辉	灿 烂	闪 出 新 世	界。
$\dot{1} \cdot \underline{65}$	$\underline{61561}$	$\underline{6561}$	5 -	$\underline{5 \cdot 612}$	$\underline{653}$	$\underline{5235}$	1 -
红 军	队 伍	真 威	风,	百 战	百 胜	最 英	勇。
$\underline{55653}$	$\underline{212}$	$\underline{55653}$	$\underline{212}$	$\underline{5 \cdot 612}$	$\underline{653}$	$\underline{5235}$	1 -
亲 爱 的 工 友	们 呀,	亲 爱 的 农 友	们 呀,	推 翻	旧 世 界	工 农 作 主	人。

图 4-30 (《八月桂花遍地开》乐谱)

在编写练习之前，我们先来讲解一下乐曲的原理。组成乐曲每个音符的音调（即频率值）与其持续的音长（即时间）是乐曲能连续演奏的两个基本要素，因此只要控制输出到喇叭的激励信号的频率高低和持续的时间，就可以使喇叭发出连续的音乐。关于音调的控制和音长的控制可以从表 4-5 中查得。

表 4-5

极低音	音名		#C0	D0	#D0	E0	F0	#F0	G0	#G0	A0	#A0	B0
	唱名		#1	2	#2	3	4	#4	5	#5	6	#6	7
	频率		35	37	39	41	44	46	49	52	55	58	61
超低音	音名	C1	#C1	D1	#D1	E1	F1	#F1	G1	#G1	A1	#A1	B1
	唱名	1	#1	2	#2	3	4	#4	5	#5	6	#6	7
	频率	65	69	73	78	82	87	93	98	104	110	117	##
低音	音名	C2	#C2	D2	#D2	E2	F2	#F2	G2	#G2	A2	#A2	B2
	唱名	1	#1	2	#2	3	4	#4	5	#5	6	#6	7
	频率	131	139	147	156	165	175	185	196	208	220	233	##
中音	音名	C3	#C3	D3	#D3	E3	F3	#F3	G3	#G3	A3	#A3	B3
	唱名	1	#1	2	#2	3	4	#4	5	#5	6	#6	7
	频率	262	277	294	311	330	349	370	392	415	440	466	##
高音	音名	C4	#C4	D4	#D4	E4	F4	#F4	G4	#G4	A4	#A4	B4
	唱名	1	#1	2	#2	3	4	#4	5	#5	6	#6	7
	频率	523	554	587	622	659	698	740	784	831	880	932	##
超高音	音名	C5	#C5	D5	#D5	E5	F5	#F5	G5	#G5	A5	#A5	B5
	唱名	1	#1	2	#2	3	4	#4	5	#5	6	#6	7
	频率	###	1108	1175	###	###	1397	1480	1568	1661	1760	###	##

节拍的控制可以利用定时器的中断进行产生。若果定时时间为 10 毫秒，现在以每拍 640 毫秒的节拍时间为例来说明，如果要产生 1 拍需要循环调用延时子程序为 64 次（64*10 毫秒），将其转换为十六进制数就是 0x40，那么节拍常数为 0x40；如此类推，若果要产生半拍需要调用延时子程序 32 次（32*10 毫秒），将其转换成十六进制数就是 0x20，其它节拍的算法也是一样道理，其中表 4-6 是节拍与调用延时子程序的关系，大家可以参考。在一组完整的音乐中，结束符和休止符是必不可少的，在此我们以 0x00 表示结束符，每当遇到 0x00 的地方就表示曲子的终结。而以 0xff 表示休止符，每当遇到 0xff 就作相应的停顿。

表 4-6

拍数	1/4 拍	半拍	3/4 拍	1 拍	1 又 1/2 拍	2 拍	4 拍
节拍常数	0x10	0x20	0x30	0x40	0x60	0x80	0x100

我们实验板的晶振为 11.0592MHZ, 现在用定时器 0 的工作方式 1 来产生节拍, 根据公式, 下面来计算定时 10 毫秒 (即 10000 微秒) 的计算过程

$$\text{定时时间} = (65536 - X) \times (12 \div \text{晶振频率})$$

$$10000 = (65536 - X) \times (12 \div 11.0592)$$

$$X = 56320$$

十六进制数为 $X = 0xDC00$

把 0x00 赋给定时器的低 8 位, 而 0xDC 就赋给定时器的高 8 位。

- 程序的设计与思路:
- (1) 包含所用单片机对应的头文件
 - (2) 对应图 4-30 的乐谱中建立一个数组, 数组的排列方式为“频率常数、节拍常数、频率常数、节拍常数”。在程序运行时不断选取节拍与频率进行演奏就可以了。
 - (3) 编写一个延时函数, 用作如果遇到休止符就停止一段时间
 - (4) 再编写一个延时函数, 用作频率的产生。
 - (5) 利用上面计算出的 10ms 定时中断, 用作节拍的产生。
 - (6) 在主程序中我们可以利用 if 语句来判断, 假如是休止符就调用函数程序来延时一段时间, 假如是结束符就从头开始唱, 假如不是休止符也不是结束符那就开启定时器来产生节拍和调用延时函数来产生频率。
- (2) 自行建立一个工程, 命名为“myone”编写其程序, 利用查询的方式每按下一次 K2, 计数器将 P1 端口的发光二极管进行顺序的递增点亮。当按到第 9 次的时候就全部熄灭, 又从零开始递增点亮, 按此方式不断循环。

- 程序的设计与思路:
- (1) 包含所用单片机对应的头文件
 - (2) 对 TMOD 进行设置, 使用计数器模式。
 - (3) 对计数器 T0 赋初值。
 - (4) 开启计数器, 等待计数器溢出。
 - (5) 利用开关语句进行相应的操作。

(以上的练习答案于光盘中)

第 6 节 串行数据通信

在介绍如何使用单片机的串行通信之前，我们先来了解几个关于串行通信的名词概念。

1. 什么是串行数据通讯与并行数据通信？

串行通信就是指将数据一位一位地按顺序传送，在传送的过程中只需要一根发送线和一根接收线，图 4-31 左边所示为串行数据通信的连接方式。

并行通信就是指将组成数据的各位不分先后同时传送。图 4-31 右边就是单片机与外部设备并行通信的连接方式。

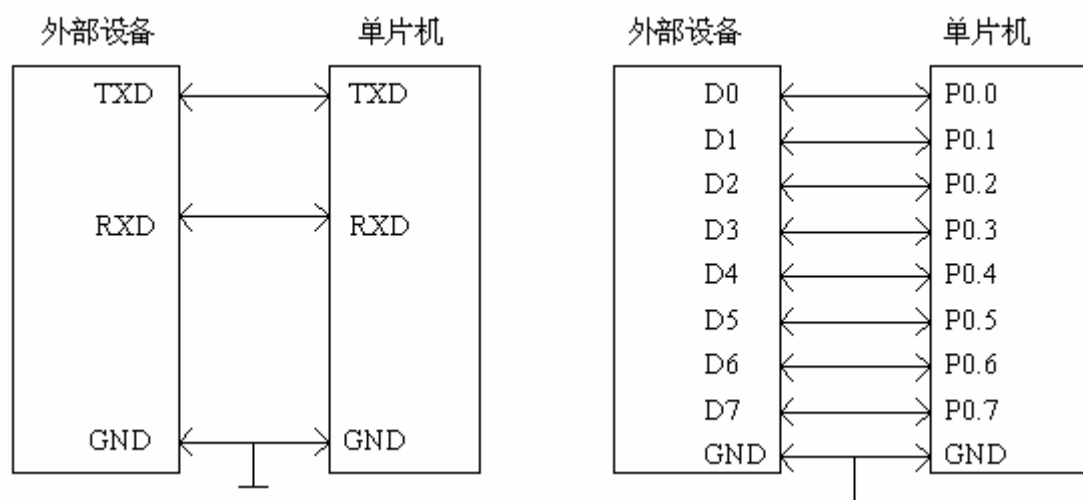


图 4-31（串行通信与并行通信）

2. 什么是奇偶校验？

串行数据在传输过程中，由于干扰可能引起数据的出错，例如：传输十六进制数‘0x55’，其各位为：0101 0101=0x55，但是由于干扰，可能使某些应该是“0”的位变为了“1”，这种情况，我们称为出现了“误码”。我们把如何发现传输中的错误，叫“检错”。发现错误后，如何消除错误，叫“纠错”。最简单的检错方法是“奇偶校验”，即在传送数据的各位之外，再传送 1 位奇/偶校验位。可采用奇校验或偶校验。

奇校验：所有传送的数位（含字节的各数位和校验位）中，“1”的个数为奇数，如：

1 0110 0101

0 0110 0001

偶校验：所有传送的数位（含字节的各数位和校验位）中，“1”的个数为偶数，如：

1 0100 0101

0 0100 0001

奇偶校验能够检测出数据传输过程中的部分误码（1 位误码能检出，2 位及 2 位以上误码不能检出），同时，它不能纠错。在发现错误后，只能要求重发。但由于其实现简单，仍得到了广泛应用。

3. 关于通信的传输速率（波特率）

串行通信的传输速率用波特率表示。波特率定义为：每秒发送二进制数码的位数，单位为“位/秒”，记作“波特”。

4. 串行通信的标准接口

标准串行通信的逻辑电平对地是对称的，与 TTL，MOS 逻辑电平完全不同。逻辑“0”电平规定为+5V 至+15V 之间，逻辑“1”电平为-5V 至-15V 之间，因此，标准串行通信接口与 TTL 电平连接必须经过电平的转换。目前比较常用的方法是直接选用 232 芯片，所以在实际应用中常把图 4-31 作出了图 4-32 的修改。也就是多加一个电平转换芯片。其中 DB-9 针插座为接到外部设备的。图 4-32 也是我们实验板的串口通信电路图。

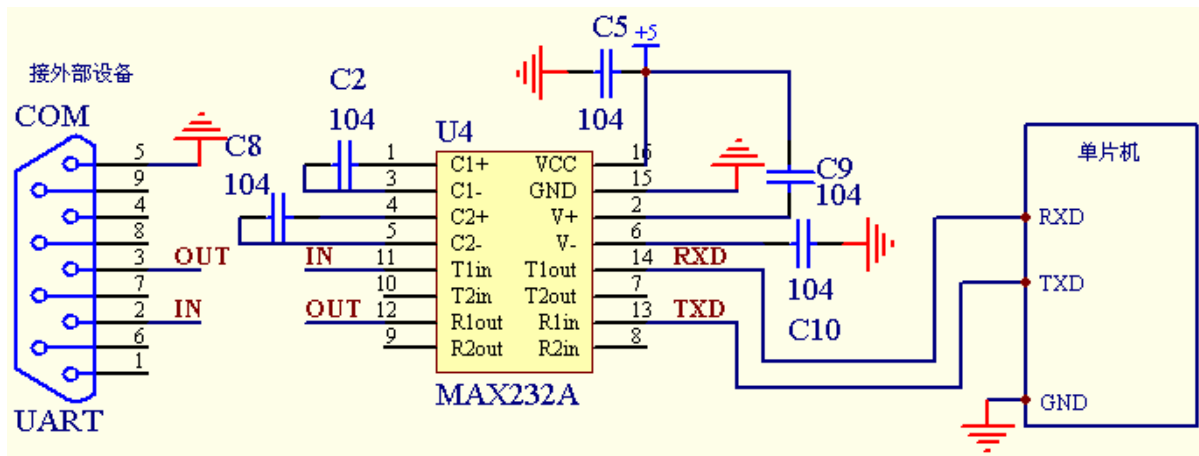


图 5-32（串行通信的实用电路）

5. 什么是帧？

每个被传送的字节数据由 4 部分组成：起始位，数据位，校验位和停止位，这 4 部分在通信中称为一帧。

6. 单片机串行口内部结构

经过以上的讲解相信大家对于串行通信有关概念有了初步的了解，下面我们再以单片机内部串行口的基本结构和相关的寄存器与如何使用它进行讨论。

如图 5-33 所示，图中共有两个串行口缓冲器（SBUF），一个是发送寄存器，一个是接收寄存器，以便单片机可以作双向通信。串行发送时，从片内总线向发送 SBUF 写入数据；串行接收时，从接收 SBUF 向片内总线读出数据。它们都是可寻址的寄存器，但因为发送与接收不能同时进行，所以给这两个寄存器赋以同一地址 0x99。

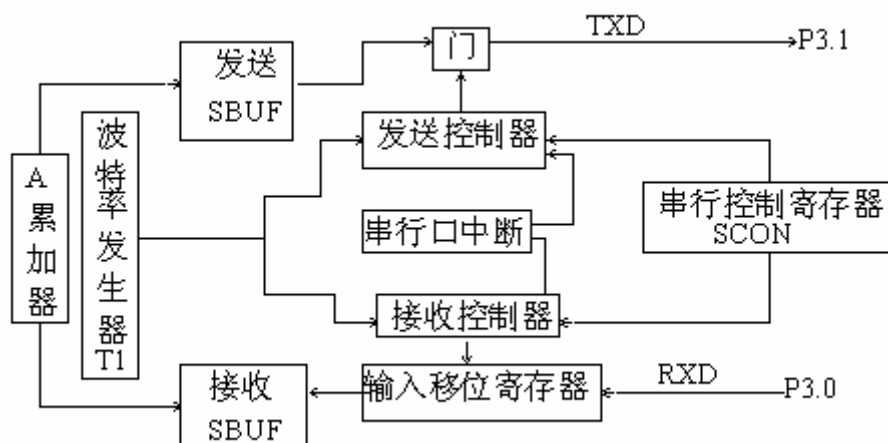


图 5-33（串行口内部结构）

而控制单片机进行串口通信的寄存器只有两个就是 SCON 与 PCON，其中有关于中断的控制则存放在 IE 寄存器中，下面我们将对这两个寄存器进行讨论。

7. 串行口控制寄存器 SCON

SCON 的地址为 98H，其中位地址为 98H 到 9FH。

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位名称	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 SM1 串行口工作方式选择位

其两者的组合如表 4-7 所示。

表 4-7

SM0	SM1	工作方式	功能	波特率
0	0	工作方式 0	8 位同步移位方式	晶振频率/12
0	1	工作方式 1	10 位 UART	可变
1	0	工作方式 2	11 位 UART	晶振频率/32 或晶振频率/64
1	1	工作方式 3	12 位 UART	可变

SM2 多机通信控制位

SM2=0 则接收到的第 9 位数据 (RB8) 无论是“0”还是“1”，都将接收到的数据装入 SBUF 中，在接收完数据后，产生中断请求，RI 置位。

SM2=1 则只有接收到的第 9 位数据 (RB8) 为“1”时，才将接收到的数据装入 SBUF 中，在接收完数据后，产生中断请求，RI 置位。若接收到的第 9 位数据 (RB8) 为“0”时，则接收到的前 8 位数据丢弃，且不产生中断请求。

REN 允许接收控制位

REN=1 允许接收数据

REN=0 禁止接收数据

TB8 发送第 9 位数据

当工作在工作方式 2 和工作方式 3 时，TB8 的内容是要发送的第 9 位数据，其值由程序员通过软件设置。在双机通信时，TB8 一般作为奇偶校验位使用，在多机通信中，常以 TB8 位的状态表示主机发送的是地址帧还是数据帧。

RB8 接收到的第 9 位数据

RB8 是接收到的第 9 位数据，当工作在工作方式 2 和工作方式 3 时，接收到的第 9 位数据存放在 TB8 中，它可能是约定的奇偶校验位，也可能是地址或数据的标志位。在工作方式 1 中，RB8 存放的是接收到停止位。而在工作方式 0 中，该位未用。

TI 发送中断标志位

当发送完一帧数据后，该位由单片机自动置 1，其状态位可供软件查询使用，也可作为中断的请求信号。大家要紧记，TI 必须由软件清 0。

RI 接收中断标志位

当接收完一帧数据后，该位由单片机自动置 1，其状态位可供软件查询使用，也可作为中断的请求信号。同样 RI 也必须由软件清 0。

8. 电源控制寄存器 PCON

PCON 地址为 0x87, 不能位寻址。

位号	D7	D6	D5	D4	D3	D2	D1	D0
位符号	SMOD	-	-	-	-	-	-	-

在电源控制寄存器 PCON 中，与串行口工作有关的仅有它的最高位 SMOD，SMOD 称为串行口的波特率倍增位。大家要注意：在程序当中不能将位 0~6 置 1，否则会出现古计不到的效果。

SMOD 串行口的波特率倍增位

SMOD=1 波特率加倍，当单片机复位后，SMOD 等于 0。

9. 中断允许控制寄存器 IE

位地址	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
位名称	EA	—	—	ES	ET1	EX1	ET0	EX0

EA 中断允许总控制位

EA=0 关闭总中断。

EA=1 启动总中断。

ES 串行中断允许控制位

ES=0 关闭串行口中断。

ES=1 启动串行口中断。

介绍完有关串行口的三个寄存器之后，我们再来讨论关于表 4-7 的 4 种工作方式。

10. 串行口的工作方式

1. 工作方式 0

当 SMO SM1 的组合为 0 0 时，单片机的串行口工作于工作方式 0。51 单片机串行口工作方式 0 实质是并行的工作方式。为同步移位输出输入，但是要实现工作方式 0 必须要借助于外接移位寄存器芯片，在实际的应用中，工作方式 0 常用于并行 I/O 口的扩展。如图 4-34 与 4-35 就是利用单片机的串行口工作方式 0 与芯片 74LS164 和 74LS165 进行 I/O 口的扩展。

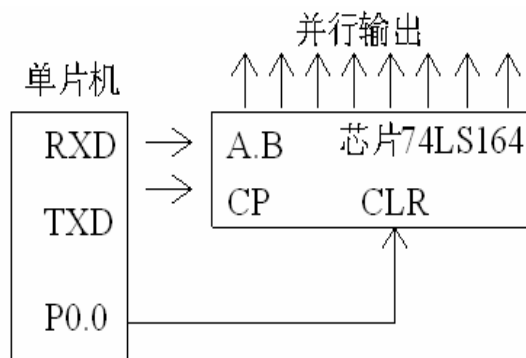


图 4-34 (并行输出)

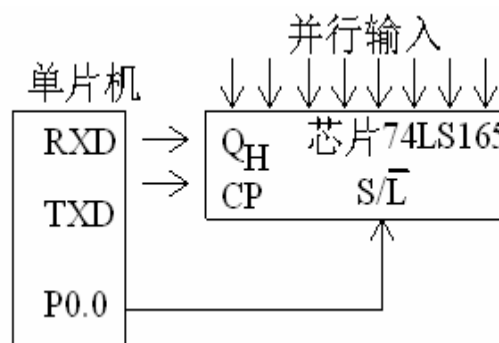


图 4-35 (并行输入)

工作方式 0 的数据传送是以 8 位数为一帧的，没有起始位和停止位，无论是接收数据或发送数据都是以低位在前，工作方式 0 的波特率是固定不变的，为 $\text{晶振频率} \div 12$ ，图 4-36 为工作方式 0 传送数据时的帧格式。

发送数据，在 TI=0 的时候，将一帧数据写进 SBUF，当发送完一帧数据之后，单片机自动将 TI 置 1。TI 必须由软件清 0。

接收数据，要想单片机接收数据 REN 必须置 1 (REN=1)，在 RI=0 的时候，当 SBUF 接收到一帧数据之后，单片机自动将 RI 置 1。RI 必须由软件清 0。

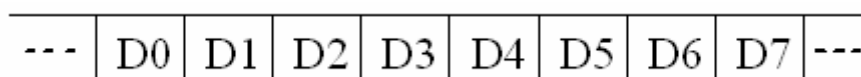


图 4-36 (工作方式 0 的帧格式)

2. 工作方式 1

当 SM0 SM1 的组合为 0 1 时，单片机的串行口工作于工作方式 1。工作方式 1 的数据帧格式由 1 个起始位，8 位数据位，和 1 位停止位组成，如图 4-37 所示。

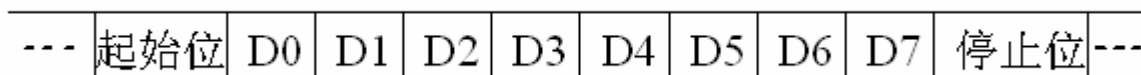


图 4-37 (工作方式 1 的帧格式)

当单片机工作于工作方式 1 时，串行口波特率是可变的 8 位通信接口，其公式如下：

$$\text{方式 1 波特率} = \frac{2^{\text{SMOD}}}{32} \times \text{定时器 T1 的溢出率}$$

经过公式的推导得出：

$$X = 256 - (2^{\text{SMOD}} \times \text{晶振频率}) \div (384 \times \text{波特率})$$

上式中，X 为定时器 T1 的计数初值，晶振频率计数单位为赫兹 (Hz)，SMOD 的数值可以从 PCON 寄存器中设置 (为 0 或 1)。现在我们来举例计算一下。假如现需要用到双机串行口通信，晶振频率为 11.0592MHz，波特率为 9600。那么根据 SMOD 值的不同可以得出下面两条式子：

当 SMOD=0 时

$$X=256- (2^{\text{SMOD}} \times \text{晶振频率}) \div (384 \times \text{波特率})$$

$$X=256- (2^0 \times 11059200) \div (384 \times 9600)$$

$$X=253$$

十六进制为 0xfd，即 TH1=TL1=0xfd

当 SMOD=1 时

$$X=256- (2^{\text{SMOD}} \times \text{晶振频率}) \div (384 \times \text{波特率})$$

$$X=256- (2^1 \times 11059200) \div (384 \times 9600)$$

$$X=250$$

十六进制为 0xfa，即 TH1=TL1=0xfa

从上面两个例子我们可以清楚在看到，当 SMOD=1 时恰好是 SMOD=0 的 1 倍。这就成为了波特率的倍增。

发送数据，在 TI=0 的时候，将一帧数据写进 SBUF，单片机自动在起始位清 0，随后是发送 8 位用户数据，在 8 位数据发送完后，单片机自动将停止位保持为 1。当发送完一帧数据之后，单片机自动将 TI 置 1。TI 必须由软件清 0。

接收数据，要想单片机接收数据 REN 必须置 1 (REN=1)。在 RI=0 的时候，SBUF 接收到由 1 到 0 的跳变开始接收数据，直到停止位送入 RB8 中，单片机自动将 RI 置 1。RI 必须由软件清 0。

3. 工作方式 2 和工作方式 3

工作方式 2 与工作方式 3 的帧格式是完全一样，如图 4-38 所示，而且与工作方式 1 在帧格式、发送和接收的情况基本相似。

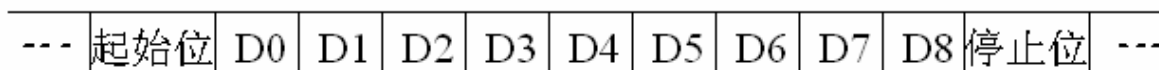


图 4-38 (工作方式 2、工作方式 3 的帧格式)

在帧格式方面只是在 8 位数据的基础上加多一个数据位第 9 位，而且第 9 位数据是可编程的，内容由用户决定。

发送数据，用户只需将第 9 位数据设定好，也就是 SCON 的 TB8 位。发送的过程与工作方式 1 基本相似，在发送到第 9 位数据时，单片机会自动从 SCON 的 TB8 位读取数据进行发送，无需用户进行干预。其它与工作方式 1 相同，当一帧数据发送完之后 TI 位置 1，需用软件清 0。

接收数据，与工作方式 1 是基本相同的，只是当接收到第 9 位数据单片机会自动将其存放在 SCON 的 TB8 位，也无需用户的干预。当完成一帧数据的接收之后，RI 置位，需用软件清 0。

工作方式 2 与工作方 3 唯一不相同的只是波特率：

工作方式 2 的波特率与 PCON 的 SMOD 位有关。当 SMOD=1 波特率为晶振频率的 $\frac{1}{32}$ ，当 SMOD=0

时波特率为晶振频率的 $\frac{1}{64}$ 。

工作方式 3 的波特率计算公式与工作方式 1 完全相同。

在实际应用中，将 T1 作为波特率发生器时，应禁止 T1 中断，以免产生不必要的中断而带来频率的误差。而 T1 一般采用定时工作方式 2。

动手实验（7）：

实验目的：学习串行通信的查询方式。

实验内容：从串行调试器中发送一个十六进制数 0x55 到单片机中，然后单片机就会发送一串 AT89S52 的字符到串行调试器的接收区。另外，在单片机上电瞬间为了测试系统的运行是否正常，特定闪动了一下 LED0。

```
#include<reg52.h>//包含 52 的头文件
#define uchar unsigned char
sbit LED=P1^0;
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {;}
}
void uart(void)//串行口初始化函数，晶振为 11.0592MHZ，波特率为 9600
{
    SCON=0x40;//选择工作方式 1
    PCON=0x00;//不选用波特率倍增
    REN=1;    //允许接收
    TI=0;    //清零发送标志位
```

```
    RI=0;    //清零接收标志位
    TMOD=0x20;//选用定时器 1 的工作方式 2
    TH1=0xfd; //对定时器 1 高位赋初值,
    TL1=0xfd; //对定时器 1 低位赋初值,
    TR1=1;    //开启定时器 1
}
void send(uchar _data)//串口发送函数
{
    SBUF=_data;//把数据装进串口缓冲寄存器
    while(TI==0);//等待数据发送完成
    TI=0;//清发送标志位
}
uchar incept(void)//串口接收函数
{
    uchar uart_data;
    while(RI==0); //等待接收数据
    RI=0;    //清接收标志位
    uart_data=SBUF; //读取串口缓冲寄存器的数据
    return uart_data;//返回从串口接收到的数据
}
void main(void)
{
    uchar ch[9]={'A','T','8','9','S','5','2',0x0d,0x0a,};//0x0d,0x0a 为回车换行的
    ASCII 代码
    uchar i;
    LED=0;//为了测试电路系统是否正常,在电路上电瞬间闪一下 led 管
    delay_ms(500);
    LED=1;
    uart();//初始化串行口
    while(1)
    {
        if(incept()==0x55)//判断从串行口接收到的数据是否为 0x55
        {
            for(i=0;i<9;i++)
            {
                send(ch[i]);//利用串行口发送数组 ch 的 9 个元素
            }
            delay_ms(1000);//延时 1 秒
        }
    }
}
```

实验步骤: 1. 打开光盘第 4 章/ uart / uart.Uv2 工程文件, 对程序进行编译、链接、调试产生 uart.hex 烧写文件。

- 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制接收，但是在这里要选取十六进制发送，最后打开串口。把 uart.hex 文件烧写到 AT89S52 单片机中，插到实验板中，通上电源。此时会看到实验板的 LED0 闪动一下，证明电路的运行正常。
- 从发送区输入一个十六进制数 0x55。在这里不用按回车键而是直接按“手工发送”。当单片机正确接收数据 0x55 时，就会把“AT89S52”的字符发送到调试器的接收区如下图 4-39 所示。



图 4-39

实验总结：大家应该也留意到，在上面的程序当中。发送数据与接收数据分成两个函数，这样有利于程序的可读性和调试。下面我们再来重温一下串口数据的发送与接收过程。如下面的发送函数，首先利用函数调用把要发送的数据传递过来，然后把数据装进串口缓冲寄存器 SBUF，在数据没有发送完毕之前 TI 值为 0，但是当数据发送完之后 TI 的值就会置 1，因此我们可以利用 while 循环语句来等待数据是否发送完成，最后再把 TI 清零，准备下次的的数据发送就可以了。

```
void send(uchar _data)//串口发送函数
{
    SBUF=_data;//把数据装进串口缓冲寄存器
    while(TI==0);//等待数据发送完成
    TI=0;//清发送标志位
}
```

下面我们再来分析一下串口数据的接收，如下面的函数，在接收方面我们可以用同样的方法利用 while 循环语句来等待数据的接收完成，当没有接收到一帧数据时 RI 的值为 0，假如 RI 的值被置 1 我们就可以读取 SBUF 缓冲寄存器的内容。最后利用 return 返回语句把接收到的数据返回即可，在接收完数据后同样要把 RI 清零以并下次再接收数据。

```
uchar incept(void)//串口接收函数
{
    uchar uart_data;
    while(RI==0); //等待接收数据
    RI=0;         //清接收标志位
    uart_data=SBUF; //读取串口缓冲寄存器的数据
    return uart_data;//返回从串口接收到的数据
}
```

动手实验（8）：

实验目的：学习串行数据通信的中断。

实验内容：我们利用此实验来学习单片机的串行中断，在主程序当中不断以延时 100ms 去闪亮 LED0，当接收到有数据时立即进入中断程序把数据发送到串口调试器的接收区。

```
#include<reg52.h>//包含 52 的头文件
#include<intrins.h>//_nop_ 函数头文件
#define uchar unsigned char
sbit LED=P1^0;
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {;}
}
void uart(void)//串口初始化函数，晶振为 11.0592MHZ，波特率为 9600
{
    SCON=0x41;//选择工作方式 1, 允许接收中断
    PCON=0x00;//不选用波特率倍增
    REN=1;    //充许接收
    RI=0;     //清零接收标志位
    TI=0;     //清零发送标志位
    ES=1;     //允许串行中断
    EA=1;     //允许总中断
    TMOD=0x20;// 选用定时器 1 的工作方式 2
    TH1=0xfd; //对定时器 1 高位赋初值
    TL1=0xfd; //对定时器 1 低位赋初值
    TR1=1;    //开启定时器 1
}
void main(void)
{
    uart();    //串口初始化
```



```
while(1)
{
    LED=~LED;//不断取反 LED，从而将其闪亮
    delay_ms(100); //延时 100ms
}
}
void uart_int(void) interrupt 4//串行中断函数
{
    uchar _data;//声明一个字符变量
    RI=0;      //清接收标志位
    ES=0;      //禁止串行中断
    _data=SBUF;//读取串行缓冲寄存器的数据
    _nop_ (); //空函数，可用作精确延时，11.0592 晶振可延时 0.00000109 秒
    _nop_ (); //因为是库函数所以应包含 intrins.h 头文件
    _nop_ ();
    SBUF=_data;//把接收到的数据再次发送出去
    while(TI==0); //等待发送完成
    TI=0;      //清发送标志位
    ES=1;      //允许串行中断
}
```

- 实验步骤：1. 打开光盘第 4 章/ uart_int/ uart_int.Uv2 工程文件，对程序进行编译、链接、调试产生 uart_int.hex 烧写文件。
2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、同时选取十六进制发送和接收，最后打开串口。把 uart_int.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。此时会看到实验板的 LED0 不停地闪动。
3. 从串口的发送区输入两个数（设为 AA 55），直接按“手工发送”以十六进制的形式发送到单片机中去，那么单片机就会把接收到的数据发送到串行调试器的接收区如图 4-40 所示。



图 4-40

实验总结：大家有没有注意到这个实验与上面的动手实验（7）有什么区别的。在动手实验（7）当中，如要想从串口中接收一帧数据那么就要不断地调用串口函数从而判断 RI 位有没有置 1。但如果程序当中还有其它任务要处理呢！那起不是增加了 CPU 的开销吗！但是如果利用串行中断就可以解决这个问题。单片机可以去处理其它任务，但是当串行口接收到数据时再去响应中断程序从而去处理相应的任务。就如这个实验一样，单片机不断在主程序中闪动 LED0，当接收到数据时再执行中断程序把数据发送出去。

动手实验（9）：

实验目的：区分 printf、scanf 函数与单片机串行通信的不同。

实验内容：分别用 printf 函数与单片机的串行通信写两个程序。其功能是对串口调试器发送一个字符“A”，实验后分析两个程序有何差异。

程序一：

```
#include<reg52.h>//包含 52 的头文件
#define uchar unsigned char
void uart(void)//串口初始化函数，晶振为 11.0592MHZ，波特率为 9600
{
    SCON=0x40;//选择工作方式 1
    PCON=0x00;//不选用波特率倍增
    REN=1;    //允许接收
    TI=0;    //清零发送标志位
    RI=0;    //清零接收标志位
```

```
TMOD=0x20;// 选用定时器 1 的工作方式 2
TH1=0xfd; //对定时器 1 高位赋初值
TL1=0xfd; //对定时器 1 低位赋初值
TR1=1;    //开启定时器 1
}
void send(uchar _data)//串口发送函数
{
    SBUF=_data;//把数据装进串口缓冲寄存器
    while(TI==0);//等待数据发送完成
    TI=0;//清发送标志位
}
void main(void)
{
    uart();//初始化串行口
    send('A');//利用串行口缓冲寄存器发送一个字符 A
    while(1);
}
```

实验步骤：1. 打开光盘第 4 章/ uart_sen/ uart_sen.Uv2 工程文件，对程序进行编译、链接、调试产生 uart_sen.hex 烧写文件。

2. 将串口线把实验板与电脑相接，打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制发送和接收，最后打开串口。把 uart_sen.hex 文件烧写到 AT89S52 单片机中去，插到实验板中，通上电源。此时单片机就会从串行口中发送一个字符 A 到串口调试器的接收区，如图 4-41 所示。



图 4-41

程序二:

```
#include<reg52.h>//包含 52 的头文件
#include<stdio.h>
void uart(void)//串口初始化函数,晶振为 11.0592MHZ,波特率为 9600
{
    SCON=0x40;//选择工作方式 1
    PCON=0x00;//不选用波特率倍增
    REN=1;      //允许接收
    TI=0;      //清零发送标志位
    RI=0;      //清零接收标志位
    TMOD=0x20;//选用定时器 1 的工作方式 2
    TH1=0xfd; //对定时器 1 高位赋初值,
    TL1=0xfd; //对定时器 1 低位赋初值,
    TR1=1;    //开启定时器 1
}
void main(void)
{
    uart();//串口初始化
    printf("A");//利用 printf 函数发送一个字符 A
    while(1);
}
```

实验步骤: 1. 打开光盘第 4 章/ _printf/ _printf.Uv2 工程文件,对程序进行编译、链接、调试产生_printf.hex 烧写文件。

2. 将串口线把实验板与电脑相接,打开串口调试软件正确选择端口、波特率为 9600、不选取十六进制发送和接收,最后打开串口。把_printf.hex 文件烧写到 AT89S52 单片机中去,插到实验板中,通上电源。此时与上面“程序一”的结果相同,单片机就会从串行口中发送一个字符 A 到串口调试器的接收区,如图 4-42 所示。



图 4-42

实验分析：从输出的结果来看两个程序是一样的，但是如果我们仔细看一下两个程序各生成的代码就可以知道两个程序是不相同的。如下图 4-43 与 4-44。我们知道单片机的 RAM（数据存储器）与 ROM（程序存储器）的空间是非常有限的，以 AT89S52 来讨论，其 RAM 为 256 个字节、ROM 为 8K 字节。而利用 printf 函数编写的程序占 RAM 就占 30.1、RAM 占 1098；我们再来看利用串行口实现的程序其 RAM 就占 9.0、RAM 占 57。所以在单片机电路系统中很少会中到 printf 函数进行通信，一般在不需要考虑到资源问题的情况下才用到它。第三章中我们在做实验时大量用到 printf、scanf 函数是为了让读者更人性化、更直观地去学习。

```
Build target 'Target 1'
compiling _printf.c...
linking...
Program Size: data=30.1 xdata=0 code=1098
creating hex file from "_printf"...
"_printf" - 0 Error(s), 0 Warning(s).
```

图 4-43（利用 printf 函数生成的代码）

```
Build target 'Target 1'
compiling uart_sen.c...
linking...
Program Size: data=9.0 xdata=0 code=57
creating hex file from "uart_sen"...
"uart_sen" - 0 Error(s), 0 Warning(s).
```

图 4-44（利用串行口生成的代码）

自我练习：

(1) 自行编写一个程序，其工程名命名为“cyc”。将波特率设置为 9600，利用单片机的串行通信每 300ms 对串口调试器发送 0x55、0xaa 的数据，不断地循环。

程序的设计思路：(1) . 包含相应的头文件。

(2) . 参考上面的实验，初始化串行口。

(3) . 利用查询的方式定义一个串行口发送函数。

(4) . 编写一个延时函数。

(5) . 在主程序中可以利用 while 循环语句每 300ms 调用一次上面的发送函数，从而将数据发送到串口调试器的接收区。

(2) 利用查询的方式，自行编写一个程序，其工程名命名为“sen_inc”。将波特率设置为 9600，利用串口调试器发送一个数据到单片机中，当单片机接收到数据之后马上发送到调试器的接收区。

程序的设计思路：(1) . 包含相应的头文件。

(2) . 参考上面的实验，初始化串行口。

(3) . 利用查询的方式定义一个串行口发送函数和接收函数。

(4) . 在主程序中可以利用 while 语句不断调用接收函数，当发现接收到数据之后马上将数据通过发送函数发送到串口调试器的接收区。

(以上练习附光盘中)

第五章 单片机的实用硬件接口技术

第一节 键盘的接口技术

在单片机系统中，按键是实现人机对话的一种重要手段，如对单片机进行数据输入，指令的操作等。下面我们将对单片机系统中应用到的按键检测进行讨论。

1. 按键的特性

按键的断开与闭合是由机械触点的动作所实现。由于机械的弹性作用在按键的断开与闭合过程中会产生抖动，当一个按键被按下到接触稳定，在此其间会因抖动而产生多个脉冲，而在断开的时候同样会存在此问题。就如图 5-1 所示！此抖动的脉冲会给单片机带来误判断，所以必顺消除。

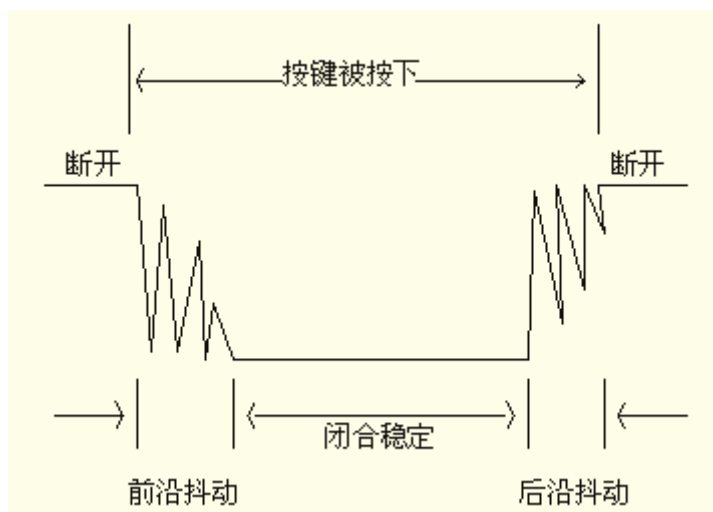


图 5-1（按键抖动的信号脉冲）

2. 消抖的处理方法

从图 5-1 可以清楚看到，从按键的断开至闭合稳定，再由按键的闭合至断开同样会产生一连串的抖动。而要消除这种抖动的脉冲可以从软件和硬件两方面入手。

硬件消抖：下面图 5-2 是一个双稳态消抖动的 RS 触发器，其作用是实现当 K 点与 B 点相接时，即按键被按下，输出 0；当 K 点与 A 点相接时，即按键松开，输出 1。从分析图 5-2 的双稳态 RS 触发器可以得知当没有与 A 或 B 点稳定相接的情况下，输出点 OUT 是不会从一个状态反转到另一个状态，从而消除了如图 5-1 因按键的弹性所产生的抖动。

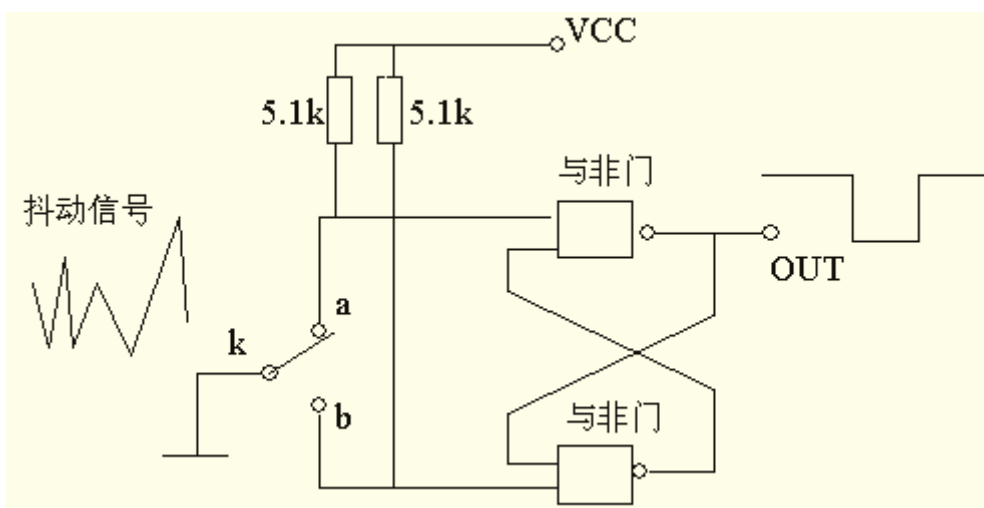


图 5-2(双稳态 RS 触发器)

软件消抖：当电路系统要检测的按键比较多时，如果用硬件来作为消抖的手段，就会显得较为烦琐，在这种情况下，我们就引入了另外一种消抖的手段，软件消抖。从按键断开到稳定闭合，此抖动的时间一般为 5ms~10ms。当单片机检测到有按键被按下的时候，就延时一段时

间，正常情况下延时 30ms 就较为合理,当延时完后再去判断刚才的状态是否仍然保持闭合。如果仍然保持闭合电平状态则确认真的有按键被按下。一般情况下，不对后沿抖动作处理。

3. 按键与单片机的连接方法

单片机与键盘的连接方式可分为独立式，编码式，串行口扩展式和矩阵式。其中较为常用的有独立式和矩阵式。当在电路系统中按键比较少时以用独立式，而按键比较多时可采用矩阵式。

独立式：

独立式按键，也就是每一个按键都与端口独立相接，每一个按键独占一根输入线，一根输入线的工作状态不会影响其它按键。独立式按键通过检测电平的状态可以很容易判断出有没有按键被按下。独立式的电路配置灵活，软件算法简单。因为要每一个按键占用一个端口数据线，当按键较多的时候就会大量浪费端口，而且电路变得复杂。所以独立式按键只适用在按键比较少或对单片机检测速度要求比较高的场合。图 5-3 为典型独立式按键的接法，图中的 8 个电阻为上拉电阻，作用是当按键被断开时确保端口为高电平，但是如果单片机端口内部含有上拉电阻的，那就不必要在外部加上拉电阻。独立式按键的软件操作方法：在程序运行时对每一个端口进行独立的判断，假如检测到某一端口有低电平，就延时一段时间，再去检测是否确定有低电平，假如还是有持续的低电平就确认真的有按键被按下，然后转到相应的程序进行处理。

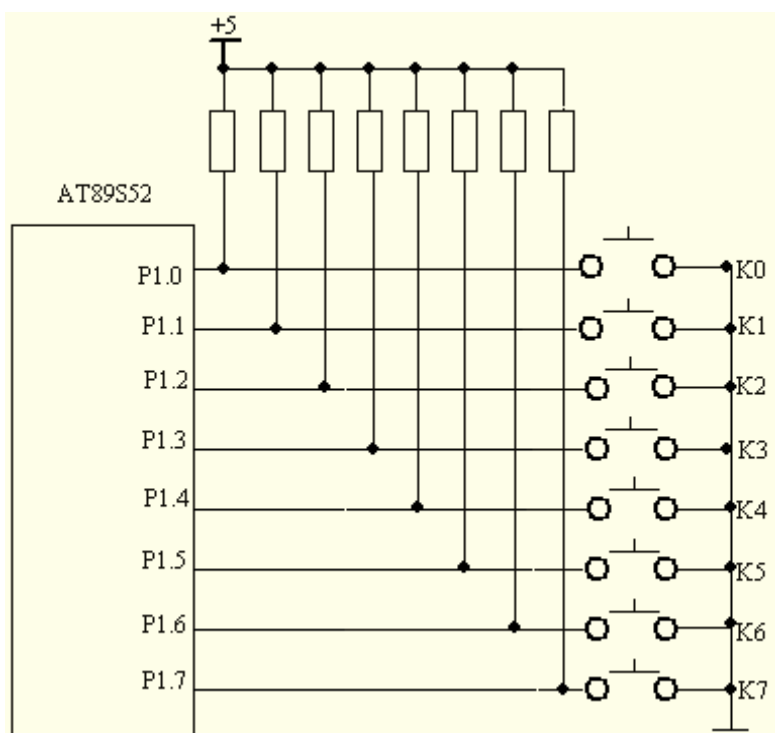


图 5-3（独立式按键的接法）

矩阵式：也称为行列式。当键盘中按键数量较多时 为了减少端口的占用通常将按键排列成矩阵形式如下图 5-4 所示，在矩阵式键盘中 每条水平线和垂直线在交叉处不直接连通而是通过一个按键加以连接，到底这样做是出意何种目的呢？ 大家看下面图 5-4 的电路图，单

片机的整个 8 位端口可以构成 $4 \times 4 = 16$ 个矩阵式按键，相比如图 5-3 的独立式按键接法多出了一倍，而且线数越多区别就越明显，假如再多加一条线就可以构成 20 个按键的键盘，但是独立式按键接法只能多出 1 个按键。由此可见，在需要的按键数量比较多时，采用矩阵法来连接键盘是非常合理的，矩阵式结构的键盘显然比独立式键盘复杂一些，单片机对其进行识别也要复杂一些，在图 5-3 中，行线通过电阻接电源作为输入（高 4 位，即 P1.4~P1.7）并将列线所接的单片机 4 个端口作为输出端（低 4 位，即 P1.0~P1.3）。因此当按键没有被按下时高 4 位全部是高电平代表无按键按下。列线逐一输出低电平，一旦有键被按下则输入线就会被拉低这样通过读入输入线的状态就可得知是否有键按下了。确定矩阵式键盘上任何一个键被按下通常采用行扫描法。行扫描法又称为逐行查询法它是一种最常用的多按键识别方法。因此，我们就以行扫描法为例介绍矩阵式键盘的工作原理。

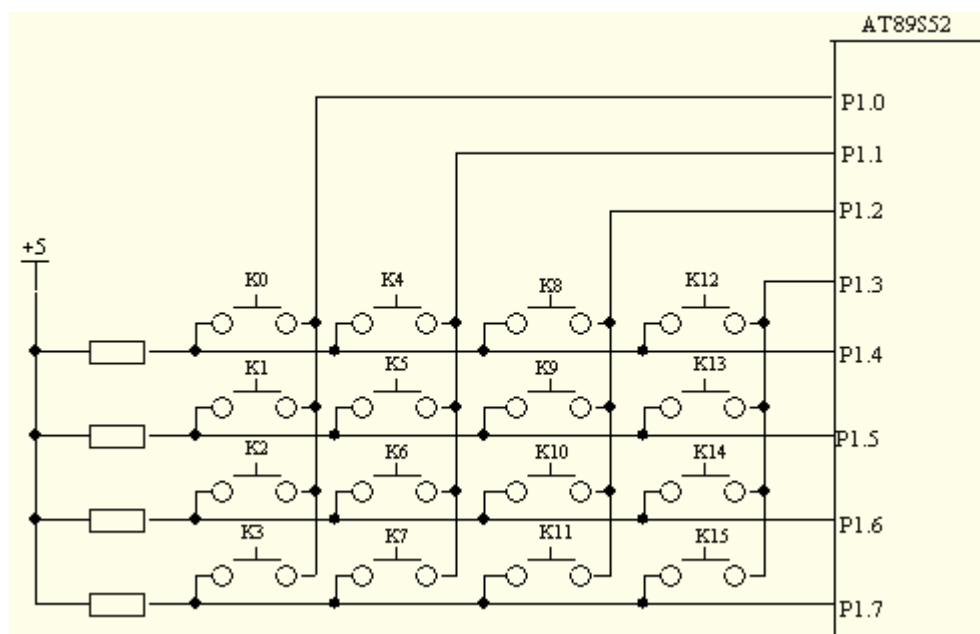


图 5-4（4*4 矩阵式按键的接法）

首先，判断键盘中是否有键按下。将全部列线（低 4 位，即 P1.0~P1.3）置低电平然后检测行线的状态（高 4 位，即 P1.4~P1.7），只要有一行的电平为低就延时一段时间以消除抖动，然后再次判断，假如依然为低电平，则表示键盘中真的有键被按下而且闭合的键位于低电平的 4 个按键之中任其一，若所有行线均为高电平则表示键盘中无键按下。其次，判断闭合键所在的具体位置。在确认有键按下后，即可进入确定具体闭合键的过程。其方法是：依次将列线置为低电平，即在置某一根列线为低电平时，其它列线为高电平。同时再逐行检测各行线的电平状态；若某行为低，则该行线与置为低电平的列线交叉处的按键就是闭合的按键。下面图 5-5 是 4*4 矩阵式按键接法的软件算法操作流程。

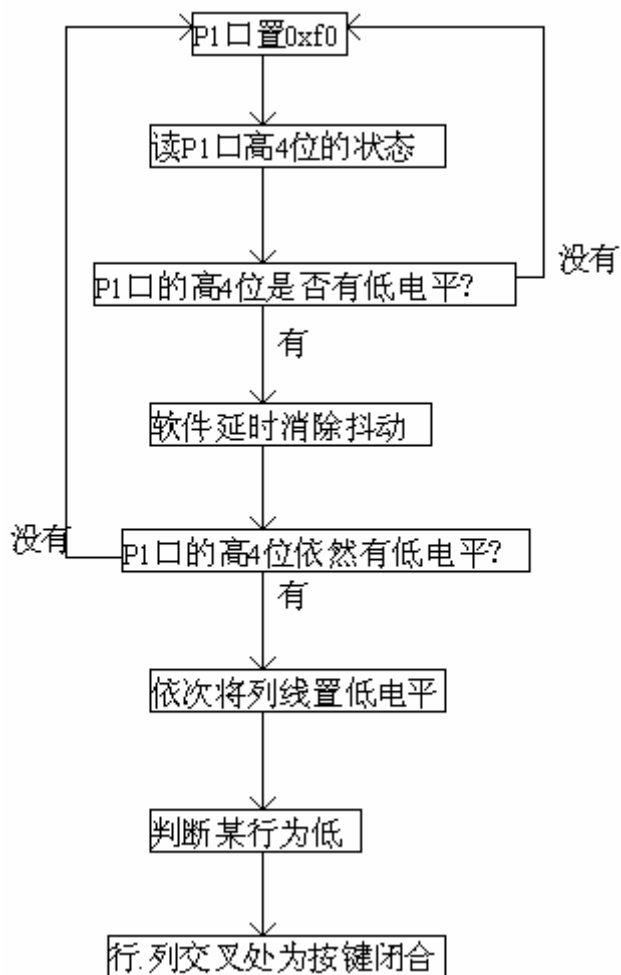


图 5-5 (矩阵式按键接法软件算法操作流程)

下面程序以图 5-5 的算法流程编写的，其电路如图 5-6，只是在图 5-5 的基础上多加了 P0 端口的 8 只 LED 灯。从键盘中检测到一个键值，然后将这个值写到 P0 口的 LED 灯中，利用 LED 灯把按键值显示出来，程序于光盘的第五章 matrix 文件夹中。

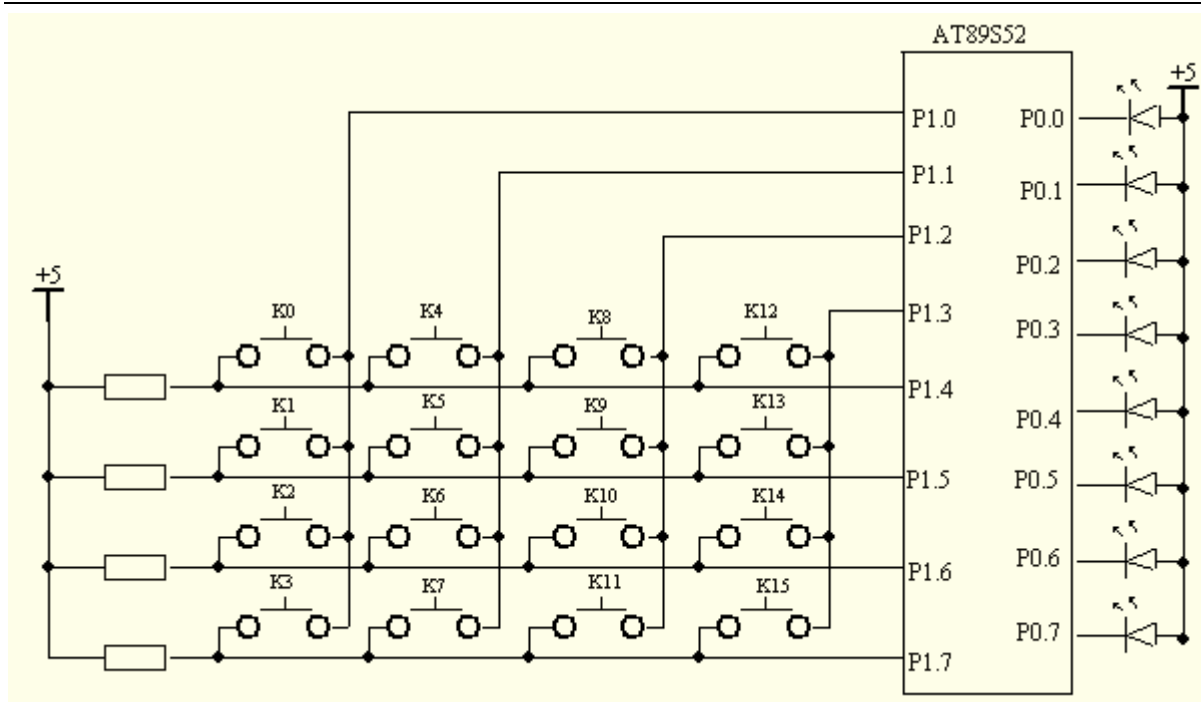


图 5-6

```
#include<reg52.h>
#define uchar unsigned char
#define uint unsigned int
void delay_ms(unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
uchar s_scan(void)
{/*按键扫描函数*/
    uchar scan_code, temp;
    P1=0xf0;//P1 口置 0xf0
    if((P1&0xf0)!=0xf0)//读高 4 位的状态, 判断是否有低电平
    {
        //假如有低电平, 表示有按键按下
        delay_ms(30);//延时消抖
        if((P1&0xf0)!=0xf0)//P1 口的高 4 位是否依然有低电平
        {
            scan_code=0xfe;
            while((scan_code&0x10)!=0)//逐行扫描
            {
                P1=scan_code;//开始扫描
                if((P1&0xf0)!=0xf0)//本行有按键按下
                {
```

```
        temp=(P1&0xf0)|0x0f;
        while((P1&0xf0)!=0xf0); //等待按键松开
        return ((~scan_code)+(~temp)); //返回按键码
    }
    else
        scan_code=(scan_code<<1)|0x01; //依次将列线置低
    }
}
return 0; //没有按键按下, 则返回 0
}
void main(void)
{
    uchar key_code;
    while(1)
    {
        key_code=s_scan(); //调用按键函数
        if(key_code) //假如有键被按下
        {
            P0=key_code; //将按键码输出到 P0 口的 LED 灯
        }
    }
}
```

动手实验 (1)

实验目的: 学习按键的使用

实验内容: 下面图 5-7 是实验板的电路图, 在 P3.2、 P3.3、 P3.4、 P3.5 各接一个开关, 虽然引脚是赋有第二功能, 但是如果在程序中不对第二功能进行开启, 其引脚就可当作普通端口使用。本实验以实现, 当 K0 按下 LED0 点 500MS, K1 按下 LED1 点 500MS, K2 按下 LED2 点 500MS, K3 按下 LED3 点 500MS。

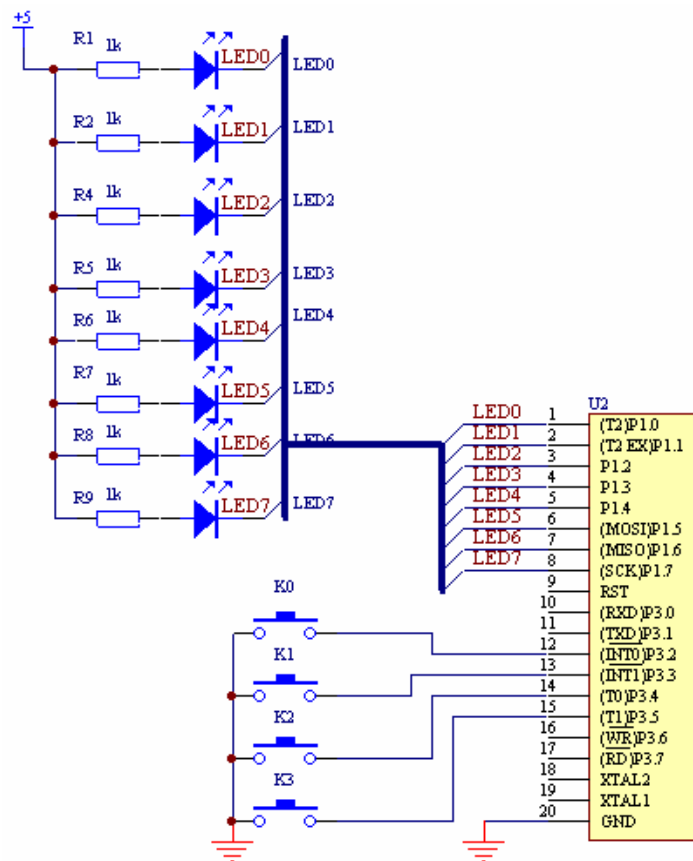


图 5-7

```
#include<reg52.h>//包含所用单片机对应的头文件
#define uchar unsigned char //宏定义
#define uint unsigned int
sbit led0=P1^0;//定义位操作
sbit led1=P1^1;
sbit led2=P1^2;
sbit led3=P1^3;
sbit K0=P3^2;
sbit K1=P3^3;
sbit K2=P3^4;
sbit K3=P3^5;
void delay_ms(unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time; i>0; i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112; j>0; j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void main(void)
{
```


```
while(1)//不断死循环
{
    if((P3&0x3c)!=0x3c)//判断是否有按键被按下
    {
        delay_ms(30); //延时消除抖动
        if((P3&0x3c)!=0x3c)//假如真的有按键被按下
        {
            if(K0==0)//K0 被按下
            {
                led0=0; //点亮 LED0
                delay_ms(500); //延时 500MS
                led0=1; //熄灭 LED0
            }
            if(K1==0)//K1 被按下
            {
                led1=0; //点亮 LED1
                delay_ms(500); //延时 500MS
                led1=1; //熄灭 LED1
            }
            if(K2==0)//K2 被按下
            {
                led2=0; //点亮 LED2
                delay_ms(500); //延时 500MS
                led2=1; //熄灭 LED2
            }
            if(K3==0)//K3 被按下
            {
                led3=0; //点亮 LED3
                delay_ms(500); //延时 500MS
                led3=1; //熄灭 LED3
            }
        }
    }
}
```

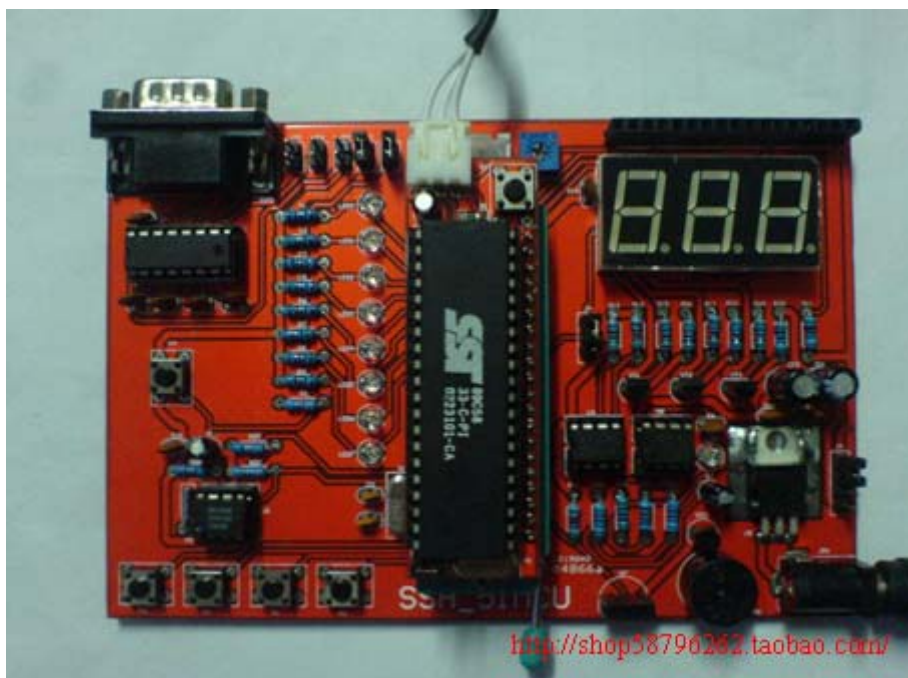
步骤：1、打开光盘第 5 章/switch / switch.uv2 工程文件，对程序进行编译，链接，

调试产生 switch.uv2 烧写文件。

2、把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接。

3、将 SSH_51MCU 实验板通上电源。

- 4、用  进行全速运行。然后分别用手去按实验板的 4 个按键，看其所实现的功能是否与程序的逻辑相一致。



自我练习：

上面动手实验的程序是在主程序中不断检测是否有按键按下，从而执行相应的操作。但是在实际应用中很少会这样做，多般的做法是键盘独立写成一个函数，在每次调用后返回一个按键码，从而得知那个按键被按下。这样做是为了程序可读性更高，更简洁和紧凑。

自行编写一个程序，功能命名为“myswitch”与实现动手实验（1）的同样功能，但是要编写一个按键函数，每次调用这个函数就可得知那个按键被按下。从而作出相应的操作。

程序设计思路：（1）同样要进行文件包含和位定义。

（2）建立一个按键函数，每次调用就可得知一个按键码，函数

可以用下面这种形式：

```
uchar key_code(void)//检测按键函数
```

```
{
```

在函数里面可以声明一个变量，如同动手实验（1）一样判断那个按键被按下，从而变量赋予不同的值，函数最后就返回这个值。

```
}
```

（3）在主函数里就不需要检测有没有键被按下，而是调用函数 `key_code()`；从而得出一个按键值，再作相应的操作。

(以上的练习答案于光盘中)

第二节 LED 数码管显示原理

除了键盘以外，LED 数码显示管与 LCD 液晶显示器在人机沟通方面也扮演了非常重要的角色，下面我们分别对其进行讨论。

一. LED 显示器的结构

LED(Light Emiting Diode)是发光二极管的缩写。LED 数码管里面有 8 只发光二极管，与实验板 P1 端口所接的二极管是相同的。分别记作 a、b、c、d、e、f、g、dp、其中 dp 为小数点，每一只发光二极管都有一根电极引到外部引脚上，而另外一只引脚就连接在一起同样也引到外部引脚上，记作公共端 (COM)，如图 5-8 所示，而图 5-9 为实物图，其中引脚的排列因不同的厂商而有所不同。

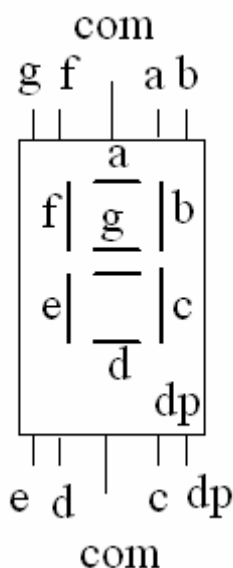


图 5-8 (数码管引脚图)



图 5-9 (数码管实物图)

市面上常用的 LED 数码管有两种，分为共阳极与共阴极。共阳极：当数码管里面的发光二极管的阳极接在一起作为公共引脚，在正常使用时此引脚接电源正极。当发光二极管的阴极接低电平时，发光二极管被点亮，从而相应的数码段显示(如图 5-10 所示)。而输入高电平的段则不能点亮。相反，共阴极：当数码管里面的发光二极管的阴极接在一起作为公共引脚，在正常使用时此引脚接电源负极。当发光二极管的阳极接高电平时，发光二极管被点亮，从而相应的数码段显示(如图 5-11 所示)，而输入低电平的段则不能点亮。

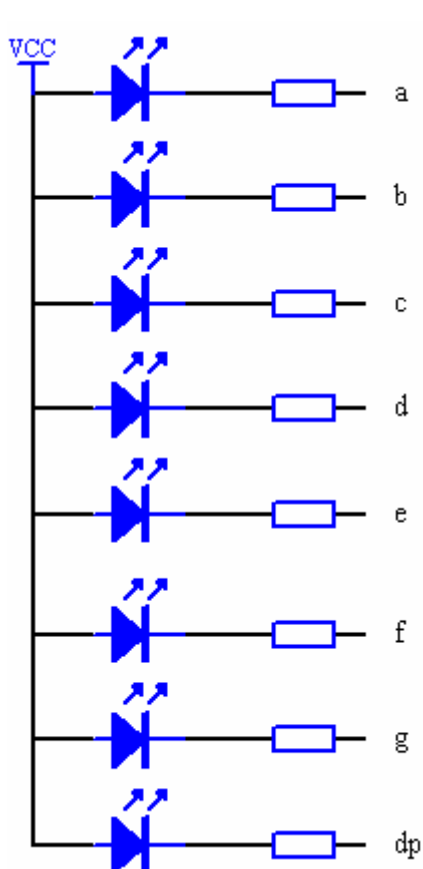


图 5-10 (共阳极)

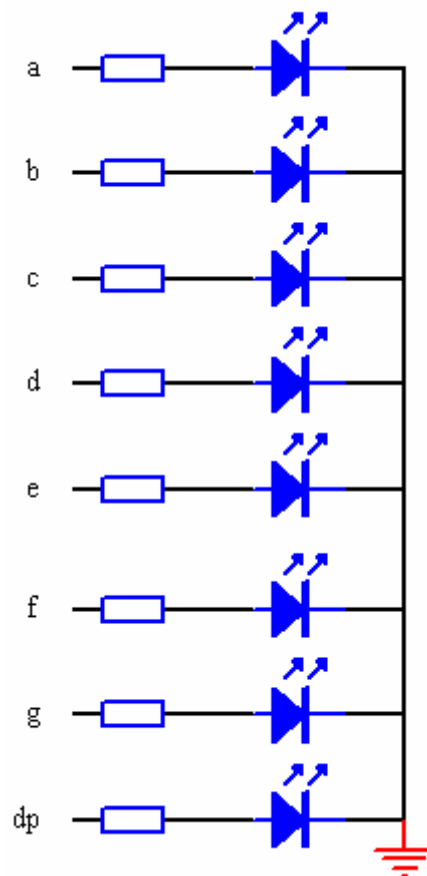


图 5-11 (共阴极)

二. LED 显示器的工作原理

下面图 5-12 为实验板其中一只数码管，而图 5-13 为数码管的内部接法，也就是前面所说的共阳极。当要其显示“1”时，只需置 B 与 C 为低电平，而其它的为高电平；当要显示“2”时，只需置 A、B、G、E、D 为低电平，而其它的为高电平；当要显示“8”时，就除了小数点以外全部为低电平；如此类推。

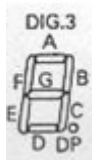


图 5-12

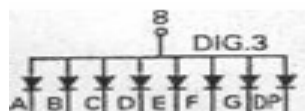


图 5-13

下面我们来系统地介绍一下在单片机应用电路中最为实用的 LED 数码管显示方法——“动态扫描”。什么是动态扫描？就是所要工作的若干个数码管轮流显示，只要轮流显示的速度足够快，每秒约 50 次以上，由于人眼的“视觉暂留”特性，看起来就像是连续显示，这样称为动态扫描。这种显示方式在数码管应用系统中应用得最为广泛。

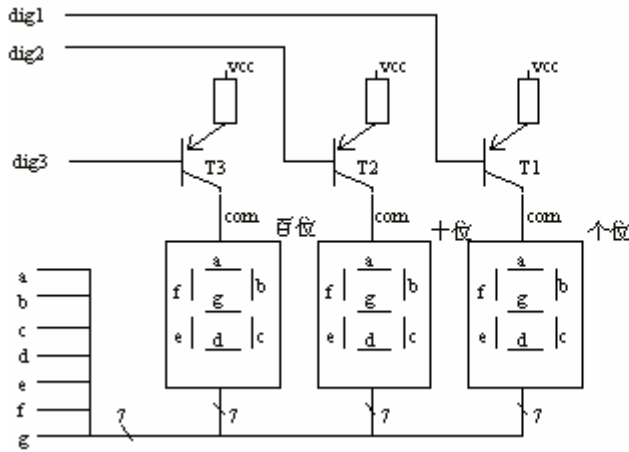


图 5-14 (动态扫描电路)

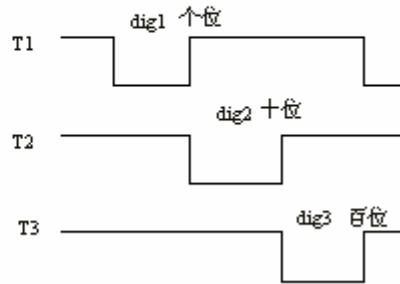


图 5-15 (动态扫描的工作时序)

图 5-14 为共阳极的动态扫电路，而 5-15 则为电路的工作时序。下面我们来分析一下动态扫描的工作原理。在电路中 T1、T2、T3 为用作开关作用，当 B 极 dig1、dig2、dig3 为低电平时导通，为高电平时截止，三个电阻为限流电阻。下面我们图 5-15 来理解一下动态扫描的工作原理。

第一：首先显示个位，在单片机中将 dig1 置低电平，而 dig2 与 dig3 置高电平，所以只有 T1 导通，而 T2 与 T3 则截止，同时在段码 a~g 中输出相应段码的低电平，那么在数码管中只有个位显示，而相应的十位与百位则没有显示。

第二：显示十位，在单片机中将 dig2 置低电平，而 dig1 与 dig3 置高电平，所以只有 T2 导通，而 T1 与 T3 则截止，同时在段码 a~g 中输出相应段码的低电平，那么在数码管中只有十位显示，而相应的个位与百位则没有显示。

第三：显示百位，在单片机中将 dig3 置低电平，而 dig1 与 dig2 置高电平，所以只有 T3 导通，而 T1 与 T2 则截止，同时在段码 a~g 中输出相应段码的低电平，那么在数码管中只有百位显示，而相应的个位与十位则没有显示。

这就是数码管动态扫描在单片机系统中的应用。下面我们通过实战来加深了解。

动手实验 (2)

实验目的：了解数码管的工作原理。

实验内容：下面的实验程序对了解 LED 数码管的工作原理是非常显著，利用实验板的三个数码管显示“1”“2”“3”图 5-16 为实验板的基本原理图（详细原理图附光盘中）。我们这次的任务是要点亮三只数码管，如果我们要 1 秒钟点亮 3 只数码管 50 次，那么一只数码管大概要点亮 6ms。

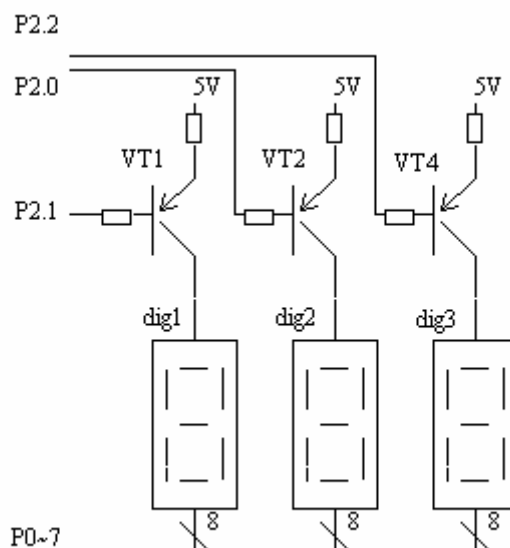


图 5-16

```

#include<reg52.h>//文件包含
#define uchar unsigned char
sbit led1=P2^1;//定义第一只数码管的控制线
sbit led2=P2^0;//定义第二只数码管的控制线
sbit led3=P2^2;//定义第三只数码管的控制线
#define dig1 led1=0;led2=1;led3=1;//只点亮第一只数码管
#define dig2 led1=1;led2=0;led3=1;//只点亮第二只数码管
#define dig3 led1=1;led2=1;led3=0;//只点亮第三只数码管
#define show P0//定义 P0 口为所显示段码的控制
void delay_ms(unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {};
}void main(void)
{
    uchar number[10]={0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x98}; //数码 0~9
    while(1)//不断循环
    {
        dig1          //只点亮第一只数码管
        show=number[1]; //显示 1
        delay_ms(6);   //延时 6ms
        dig2          //只点亮第一只数码管
        show=number[2]; //显示 2
        delay_ms(6);   //延时 6ms
        dig3          //只点亮第一只数码管
    }
}


```

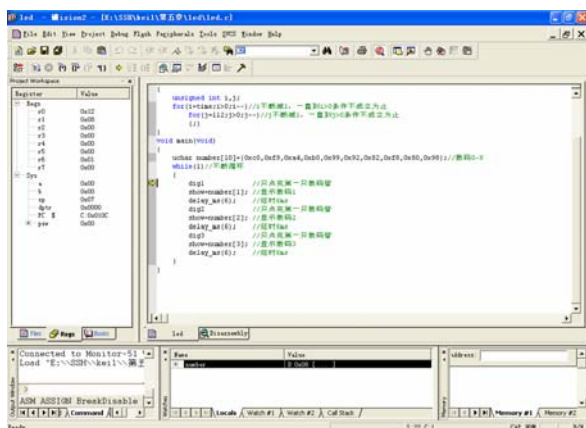
```

show=number[3]; //显示 3
delay_ms(6);    //延时 6ms
}
}
}

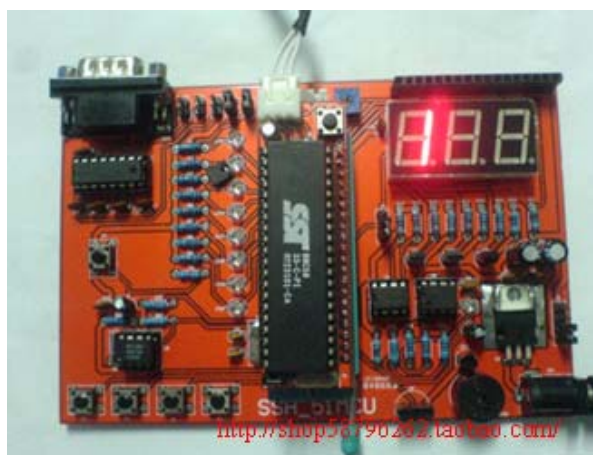
```

实验步骤:

1. 打开光盘第 5 章/ led / led. uv2 工程文件, 对程序进行编译、链接、调试产生 led. uv2 烧写文件。
2. 将实验板的 J4 短接到 LEDP 的一边, J7 短接到 LEDE 的一边。
3. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上, 另一端与电脑相接, 将 SSH_51MCU 实验板通上电源, 将工程设置为硬件仿真, 同时将波特率设置为 38400。
4.  对程序进行调试, 同时观察数码管显示的变化。
5. 下面 (实图 3) 为仿真时的 KEIL 界面, (实图 4) 为实验板的同步况情。



(实图 3)



(实图 4)

实验总结: 从上面的实验当中我们可以清楚地了解到数码管的工作原理, 但是实际应用中是不会在主函数中用死循环来点亮数码的。大家试想一下, 假如在主程序中不断用死循环来点亮数码管, 若果当单片机还要处理按键扫描, 数据的发送与接收等等那怎么办啊! 下面我们来介绍一种非常实用的数码管扫描技术。

动手实验 (3)

实验目的: 学习利用定时器来对 LED 数码管进行动态扫描。


实验内容: 利用定时器的定时中断来对 LED 数码管进行扫描, 实现上面同样的功能, 使数码管显示 “1” “2” “3”。

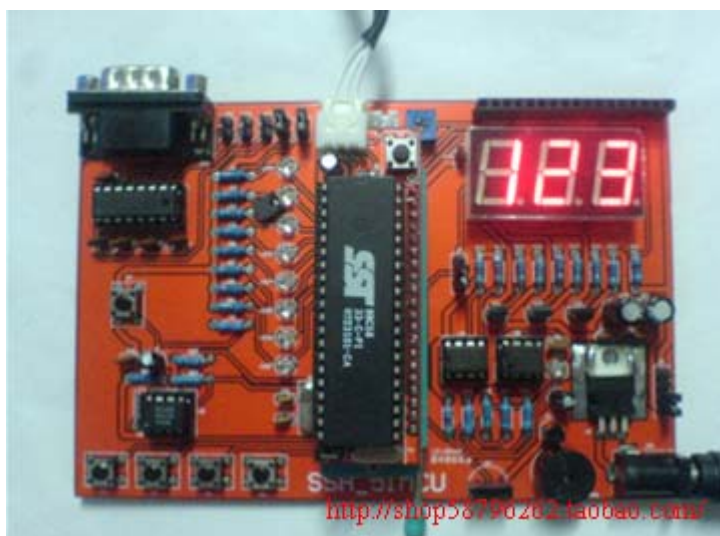
#include<reg52.h>//文件包含

```
#define uchar unsigned char
sbit led1=P2^1;//定义第一只数码管的控制线
sbit led2=P2^0;//定义第二只数码管的控制线
sbit led3=P2^2;//定义第三只数码管的控制线
#define dig1 led1=0;led2=1;led3=1;//只点亮第一只数码管
#define dig2 led1=1;led2=0;led3=1;//只点亮第二只数码管
#define dig3 led1=1;led2=1;led3=0;//只点亮第三只数码管
#define show P0//定义P0口为所显示段码
uchar number[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x98}; //数码0~9
uchar one; //个位
uchar ten; //十位
uchar hundred;//百位
void T0_init(void)
{
    TMOD=0x01;//选择工作方式1
    TL0=0x66;//每6ms中断一次
    TH0=0xea;
    TF0=0;//中断标志位清零
    ET0=1;//允许定时器0中断
    EA=1;//允许总中断
    TR0=1;//启动定时器
}
void main(void)
{
    T0_init(); //定时器0初始化
    one=number[3]; //个位显示3
    ten=number[2]; //十位显示2
    hundred=number[1]; //百位显示1
    while(1);
}
void time_display(void) interrupt 1
{
    static uchar show_bit=1;
    TL0=0x66;//定时器赋初始
    TH0=0xea;
    switch(show_bit)
    {
        case 1:
        {
            dig1 //只点亮第一只数码管
            show=hundred; //百位
            show_bit=2;//下次进中断程序点亮十位
            break;
        }
    }
}
```

```
case 2:
{
    dig2          //只点亮第二只数码管
    show=ten; //十位
    show_bit=3;//下次进中断程序点亮百位
    break;
}
case 3:
{
    dig3          //只点亮第三只数码管
    show=one; //个位
    show_bit=1;//下次进中断程序点亮个位
    break;
}
}
```

实验步骤:

1. 打开光盘第 5 章/ T_led / T_led.uv2 工程文件，对程序进行编译、链接、调试产生 T_led.uv2 烧写文件。
2. 将实验板的 J4 短接到 LEDP 的一边，J7 短接到 LEDE 的一边。
3. 把 SST_51 仿真器正确装上到 SSH_51MCU 实验板上，另一端与电脑相接，将 SSH_51MCU 实验板通上电源，将工程设置为硬件仿真，同时将波特率设置为 38400。
4.  对程序进行全速运行，同时观察数码管显示状态的变化。实验效果如下（实图 5）



（实图 5）

实验结果：本实验中利用定时器的定时中断对 LED 管进行点亮。这样单片机就可以在主函数中处理

其它的事情，如按键扫描，数据的发送与接收等。当定时计数溢出时单片机才去响应中断程序，点亮一次数码管再回到主程序，这样不断周而复始地循环。

动手实验（4）

实验目的：学习 LED 数码管的实用技术。

实验内容：用 LED 数码管记录下 K0 所按下的次数，即按一下 K0 增值一次曾在数码管中显示出来。

```
#include<reg52.h>//文件包含
#define uchar unsigned char
sbit led1=P2^1;//定义第一只数码管的控制线
sbit led2=P2^0;//定义第二只数码管的控制线
sbit led3=P2^2;//定义第三只数码管的控制线
sbit K0=P3^2;
#define dig1 led1=0;led2=1;led3=1;//只点亮第一只数码管
#define dig2 led1=1;led2=0;led3=1;//只点亮第二只数码管
#define dig3 led1=1;led2=1;led3=0;//只点亮第三只数码管
#define show P0//定义 P0 口为所显示段码
uchar number[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x98}; //数码 0~9
uchar one; //个位
uchar ten; //十位
uchar hundred;//百位
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {};
}
void T0_init(void)
{
    TMOD=0x01;//选择工作方式 1
    TL0=0x66;//每 6ms 中断一次
    TH0=0xea;
    TF0=0;//中断标志位清零
    ET0=1;//允许定时器 0 中断
    EA=1;//允许总中断
    TR0=1;//启动定时器
}
void main(void)
{
    uchar i,j,k;
    T0_init();//定时器 0 初始化
    i=0;//控制个位数码
    j=0;//控制十位数码
```

```
k=0;//控制百位数码
one=number[0];//上电时三位数码管都显示为 000
ten=number[0];
hundred=number[0];
while(1)
{
    if(K0==0)//判断是否有按键被按下
    {
        delay_ms(20);//延时消抖
        if(K0==0)//确认有按键被按下
        {
            while(K0==0);//等待按键松开
            i++;//个位递增
            one=number[i];//显示个位
            if(i==10)//假如个位为 10 立即向十位进 1
            {
                i=0;//个位再从 0 开始递增
                j++;//十位递增
                one=number[i];//显示个位
                ten=number[j]; //显示十位
                if(j==10)//假如十位为 10 立即向百位进 1
                {
                    j=0;//十位再从 0 开始递增
                    ten=number[j];//显示十位
                    k++;//百位递增
                    hundred=number[k]; //显示百位
                    if(k==10)//当计数为 999 时立即转为 000 从新开始递增
                    {
                        i=0;//三位数码管都显示为 000
                        j=0;
                        k=0;
                        one=number[0];
                        ten=number[0];
                        hundred=number[0];
                    }
                }
            }
        }
    }
}

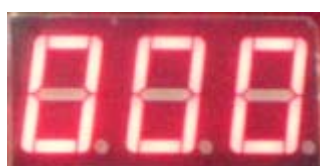
void time_display(void) interrupt 1
{
    static uchar show_bit=1;
```



```
TL0=0x66; //定时器赋初始
TH0=0xea;
switch(show_bit)
{
    case 1:
    {
        dig1          //只点亮第一只数码管
        show=hundred; //百位
        show_bit=2; //下次进中断程序点亮十位
        break;
    }
    case 2:
    {
        dig2          //只点亮第二只数码管
        show=ten; //十位
        show_bit=3; //下次进中断程序点亮百位
        break;
    }
    case 3:
    {
        dig3          //只点亮第三只数码管
        show=one; //个位
        show_bit=1; //下次进中断程序点亮个位
        break;
    }
}
}
```

实验步骤

1. 打开光盘第 5 章/ key_led / key_led.uv2 工程文件，对程序进行编译、链接、调试产生 key_led.hex 烧写文件。
2. 将实验板的 J4 短接到 LEDP 的一边，J7 短接到 LEDE 的一边。
3. 将烧写文件烧写到 AT89S52 单片机中去，正确安装到实验板上，接上电源。
4. 此时会见到数码管显示 000，当每按下一次 K0 数码管的值就会增 1。实验效果如下（实图 6）



(实图 6)

自我练习：自行编写一个程序，利用实验板的三个数码管显示从 000~999 每 300 毫秒增加一次，不断周而复始地循环。工程名命名为“myled”。

程序的设计思路：要实现本程序，我们可以在动手实验（4）的主程序中稍作修改就可以了，关于程序的其它可以完全不变。

- (1) 在主函数中声明三个变量 i、j、k、并使其初始化为 0。
- (2) 每 300ms 使 i 的值自加 1，当 i 的值为 10 时使 j 自加 1，当 j 为 10 时再使 k 自加 1，当计数到 999 时再把 i、j、k、三个变量赋值为 0 从新开始计数。
- (3) 计数的同时将 i、j、k、三个变量赋给全局变量的个位、十位、百位。

(练习的答案附光盘中)

第三节 LCD 液晶显示器的原理

LCD(Liquid Crystal Display)是液晶显示器的缩写。LCD 的应用十分广泛，简单如手表，闹钟上的液晶显示屏；一般办公设备如传真机，复印机，都是非常容易见到 LCD 的足迹。所以说：LCD 液晶显示器是学习单片机的一门必修课。

一般初学者由字符型 LCD 开始入手，到掌握如何控制字符型液晶再进一步学习控制图像那就容易得多了。LCD 字符型液晶显示模块是现今市面上最为常用的液晶显示器，它是一类专门用于显示字母、数字、符号等点阵式的液晶显示模块，其规格只要分为 16 字×1 行、16 字×2 行、20 字×2 行、40 字×2 行，这些规格的液晶所显示的字符和操作方法都是大同小异的。

本节我们要学习的就是 MD1602A 它的规格是属于 16 字×2 行（图 5-16 为其实物图），内嵌控制器为日本日立新华通讯社公司的 HD44780 芯片。一般字符 LCD 模块的控制器都是使用其芯片。



(LCD 液晶屏正面)



(LCD 液晶屏背面)

图 5-16 (LCD 字符液晶屏实物图)

要学习如何控制字符型 LCD 就必定要对其内部的控制芯片 HD44780 进行了解，下面我们就来对 HD44780 来进行讨论。

一. 引脚功能

表 5-1 就是 MD1602A 的接口引脚功能。

表 5-1

引脚号	符号	功能说明	引脚号	符号	功能说明
1	VSS	5V 电源地	9	D2	数据端口 2
2	VDD	5V 电源正极	10	D3	数据端口 3
3	VL	液晶显示偏压信号	11	D4	数据端口 4
4	RS	数据/命令选择端	12	D5	数据端口 6
5	R/W	读/写选择端	13	D6	数据端口 7
6	E	使能信号	14	D7	数据端口 8
7	D0	数据端口 0	15	BLA	背光源正极
8	D1	数据端口 1	16	BLK	背光源负极

在表 5-1 中

VL: 为液晶的对比度调整端，一般可以接一个 10K 可调电阻来根据不同对比度的调整。

RS: 为寄存器的选择，RS=1 选择数据寄存器，RS=0 选择指令寄存器。

R/W: 读写信号线，R/W=1 为读操作，R/W=0 为写操作。

E: 使能端，当 E 由高电平跳为低电平时，HD44780 芯片执行命令。

D0~D7: 8 位数据线。

二. 内部资源

(1) DDRAM (显示数据存储)

DDRAM 作用是用来存放 LCD 要显示的数据，只要将点阵字符图形的代码送入 DDRAM，内部的控制电路就会自动将数据传送到 LCD 显示屏上。如果我想在第一行的第 1 个位置显示字符“0”，那么只要把字符“0”的代码送到 DDRAM 的 0x80 地址中，在显示屏就会出现一个字符“0”，图 5-17 是存储器地址与实际显示字符的对应位置。

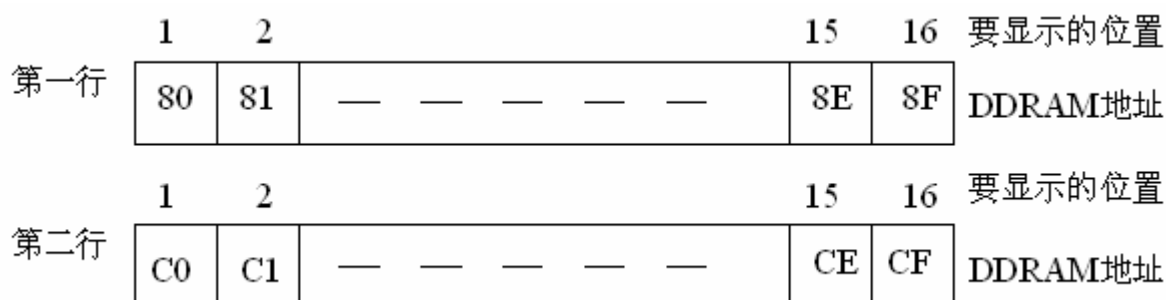


图 5-17

(2) CGROM (字符发生器)

HD44780 芯片内含一个 CGROM (字符发生器)，存储了 160 个不同的点阵字符图形，如数字、字母、日文等，如图 5-18 所示。

Upper 4bit Lower 4bit	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CB RAM (1)		0	1	2	3	4	5	6	7	8	9	A	B	C	D
0001	(2)		E	F	10	11	12	13	14	15	16	17	18	19	20	21
0010	(3)		22	23	24	25	26	27	28	29	30	31	32	33	34	35
0011	(4)		36	37	38	39	40	41	42	43	44	45	46	47	48	49
0100	(5)		50	51	52	53	54	55	56	57	58	59	60	61	62	63
0101	(6)		64	65	66	67	68	69	70	71	72	73	74	75	76	77
0110	(7)		78	79	80	81	82	83	84	85	86	87	88	89	90	91
0111	(8)		92	93	94	95	96	97	98	99	100	101	102	103	104	105
1000	(1)		106	107	108	109	110	111	112	113	114	115	116	117	118	119
1001	(2)		120	121	122	123	124	125	126	127	128	129	130	131	132	133
1010	(3)		134	135	136	137	138	139	140	141	142	143	144	145	146	147
1011	(4)		148	149	150	151	152	153	154	155	156	157	158	159	160	161
1100	(5)		162	163	164	165	166	167	168	169	170	171	172	173	174	175
1101	(6)		176	177	178	179	180	181	182	183	184	185	186	187	188	189
1110	(7)		190	191	192	193	194	195	196	197	198	199	200	201	202	203
1111	(8)		204	205	206	207	208	209	210	211	212	213	214	215	216	217

图 5-18

在图 5-18 中，行为高 4 位代码，列为低 4 位代码。如上图所说的要显示字符“0”，我们只要把字符“0”的高 4 位地址“0011”再加上低 4 位地址“0000”等于二进制码“0011 0000”，转换为十六进制为 0x30，即我们把代码 0x30 写到 DDRAM 中的 0x80 地址，就可以在第一行的第一个位置显示字符 0 了。

(3) CGRAM (字符发生器)

CGRAM 是用来储存设计者自行设计个性化字符造型代码的 RAM，共有 512bit (64 个字节)，一个 5×7 的字符体占用 8×8bit，因此 CGRAM 最多只能存放 8 个自定义字符。

(4) IR (指令寄存器)

指令寄存器负责存放单片机写给 HD44780 的指令，对 IR 的操作如下：

当 RS=0, R/W=0, E 引脚由 1 变为 0, 就会把 D0~D7 引脚上的数据送入指令寄存器 IR。

(5) DR (数据寄存器)

数据寄存器负责存放单片机写给 CGRAM 与 DDRAM 的数据或从 CGRAM 与 DDRAM 读出的数据。对 DR 的操作如下：

当 RS=1, R/W=1, E=1, HD44780 就会把数据送到 D0~D7 引脚上, 供单片机读取。

当 RS=1, R/W=0, E 引脚信号由 1 变为 0, HD44780 就会把 D0~D7 引脚上的数据存入 DR 数据寄存器。

(6) BF (忙碌标志位)

当 HD44780 内部正在处理数据时 BF=1; 当内部处于空闲状态时 BF=0。为什么要建立忙碌标志位, 那是因为单片机处理一条指令只需要化几微秒的时间, 但是 HD44780 则需要 40us~1.64ms 不等, 所以单片机每次对 LCD 进行操作时一定要检测忙碌位, 判断芯片内部是否在工作, 假如在工作则不会响应单片机的操作。

(7) (AC 地址计数器)

AC 是负责计算从 DDRAM, CGRAM 读出的地址, 或者计算写到 CGRAM, DDRAM 数据的地址, 当单片机对 CGRAM, DDRAM 进行操作时, AC 会依照单片机对 HD44780 的操作自动修改地址的计数值。

表 5-2 总结了 LCD1602 内嵌芯片 HD44780 的控制功能

表 5-2

RS	R/W	E	D7~D0	说明
0	1	1	数据输出	读 BF (忙碌标志) 与 AC 的值
1	0	下降沿	数据输入	写数据
0	0	下降沿	数据输入	写指令代码
1	1	1	数据输出	读数据

三. HD44780 的控制指令

LCD1602 的内部控制器 HD44780 共有 11 条指令, 我们下面分别来介绍。

1. 清屏

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

指令代码为: 0x01

执行此指令, HD44780 将 DDRAM 的数据全部写入“空白”的代码, 清除所显示的内容, 同时光标移到左上角。

2. 光标归位

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	—

指令代码为: 0x01, —表示可为 0 或 1。

执行此指令，AC 的值被清“0”，但是 DDRAM 的数据不变，光标移到左上角。

3. 输入方式设置

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

此指令用来设置字符、光标的移动方式详见表 5-3

表 5-3

控制位		指令代码	说明
I/D	S		
0	0	0x04	AC 的值减 1，光标左移 1 格，所显示的字符全部不动
0	1	0x05	AC 的值减 1，光标不动，所显示的字符全部右移 1 格
1	0	0x06	AC 的值加 1，光标右移 1 格，所显示的字符全部不动
1	1	0x07	AC 的值加 1，光标不动，所显示的字符全部左移 1 格

4. 控制显示的开与关

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

指令代码为：0x08~0x0f

D、C、B、三个位分别控制字符、光标和闪烁功能的开关。具体参考表 5-4

表 5-4

控制位	控制信号	功能	说明
D	D=1	开 LCD 显示	关闭显示仅字符不出现，而 DDRAM 的内容不变，这与清屏指令不同
	D=0	关 LCD 显示	
C	C=1	光标显示	光标的位置由地址指针计数器 AC 确定，并随变动而移动，当 AC 的值超出显示范围，光标将随之消失。
	C=0	光标消失	
B	B=1	光标闪烁	光标闪烁状态位。
	B=0	光标不闪烁	

5. 移动光标

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	S/C	R/L	—	—

执行该指令可使字符或光标向左或向右移动一个字符位，定时执行此指令可以使其平滑移动，如表 5-5。

表 5-5

控制位		指令代码	说明
S/C	R/L		
0	0	0x10	光标左移动
0	1	0x14	光标右移动

1	0	0x18	字符左移动
1	1	0x1c	字符右移动

6. 功能设置

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	N	F	0	0

关于功能指令的设置具体如表 5-6。

表 5-6

控制位	控制信号	功能	说明
DL	DL=0	接口总线为 4 位长度（紧 D7~D4 有效），8 位数据与指令代码按先高位后低位的方式分两次传送	LCD 与单片机接口形式。（即数据的传送方式）
	DL=1	接口总线为 8 位长度（即 D7~D0 有效）	
N	N=0	显示 1 行字符行	用于设置所显示字符的行数
	N=1	显示 2 行字符行	
F	F=0	5×7 字符体	用于设置所显示字符的字体
	F=1	5×10 字符体	

7. CGRAM 地址设置

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	A5	A4	A3	A2	A1	A0

此指令功能是将 6 位地址 **CGRAM** 写进地址计数器 AC，其后单片机对 HD44780 的操作就是对 CGRAM 的读/写操作。

8. DDRAM 地址设置

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	A6	A5	A4	A3	A2	A1	A0

此指令功能是将 7 位 DDRAM 地址写进地址计数器 AC，其后单片机对 HD44780 的操作就是对 DDRAM 的读/写操作。

9. 读 BF（忙碌标志）与 AC（地址计数器）的值

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

BF=0: HD44780 内部处于空闲状态，等待单片机对其进行读写操作。

BF=1: HD44780 内部正忙于处理数据，不接受单片机对其进行操作。

D0~D6: 最后写入 DDRAM 或 CGRAM 的当前地址值。

10. 写数据到 CGRAM 或 DDRAM

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0								

先设置 CGRAM 或 DDRAM 的地址，其后就可以把特定的字符代码写进 D0~D7，LCD 就可以将其显示出来。

11. 读 CGRAM 或 DDRAM 的数据

控制信号		指令代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	1								

先设置 CGRAM 或 DDRAM 的地址，其后就可以将 CGRAM 或 DDRAM 当前的数据读出。

经过上面对 MD1602A 内部芯片 HD44780 的讲解，相信大家已经有一定的认识，下面我们就以动手实验来加深对其了解。

动手实验（5）：

实验目的：使 LCD 单独显示一个字符。

实验内容：图 5-19 是实验板 LCD 与单片机的接口电路，现在我们就利用 LCD 的第一行的第一个位置显示一个字符“A”。

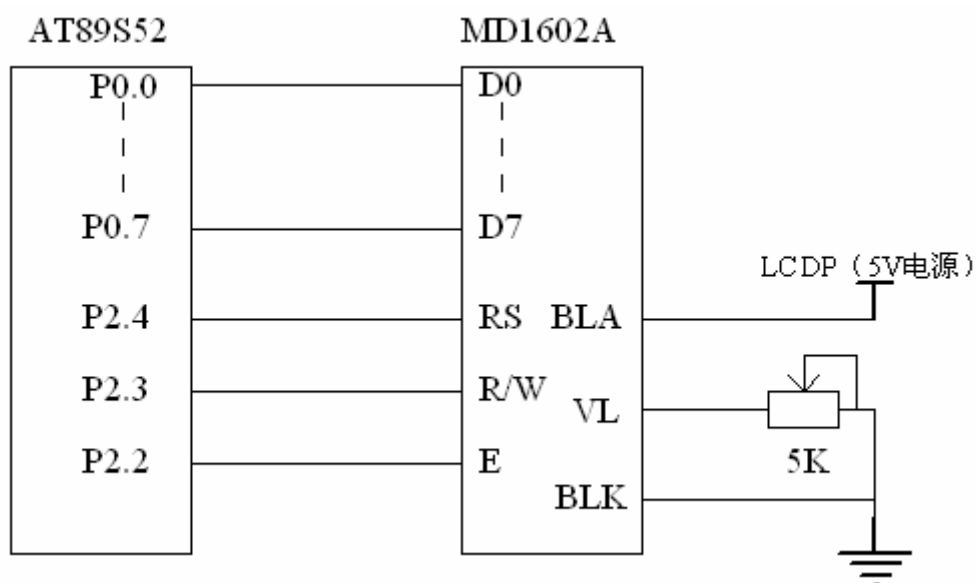


图 5-19

```
#include <reg52.h> //包含头文件
#include<intrins.h> //包含_nop_();库函数的头件
#define dataport P0 //定义 P0 为数据端口
#define uchar unsigned char
#define uint unsigned int
sbit RS=P2^4; //数据/命令选择引脚
sbit RW=P2^3; //读/写选择引脚
sbit E=P2^2; //使能信号
void rd_bf(void)
{ //检测 LCD 是否忙碌
    dataport=0xff; //数据端口先赋高电平
    RS=0; //选择指令寄存器
    RW=1; //读操作
    E=1; //E 为高电平
    _nop_(); //适当延时
    _nop_();
    while(dataport&0x80); //判断数据端口的 D7 是否为 1, 假如为 1 则等待
    E=0;
}
//写指令函数, dictate 为要写入的指令
wr_dictate(unsigned char dictate)
{
    rd_bf(); //每次写数据之前, 检测 LCD 是否忙碌
    RW=0; //写操作
    RS=0; //选择指令寄存器
    E=1; //E 为高电平
    _nop_(); //适当的延时, 目的是为了等待电平稳定
    _nop_();
    dataport=dictate; //把指令写入 IR
    E=0; //当 E 由高电平变为低电平时指令被写入寄存器
}
//写数据函数, dat 为要写入的数据
wr_data(unsigned char dat)
{
    rd_bf(); //每次写指令之前, 检测 LCD 是否忙碌
    RW=0; //写操作
    RS=1; //选择数据寄存器
    E=1; //E 为高电平
    _nop_(); //适当的延时, 目的是为了等待电平稳定
    _nop_();
    dataport=dat; //把数据写入 DR
    E=0; //当 E 由高电平变为低电平时数据被写入寄存器
```

```
}  
  
void init_lcd(void)  
{//初始化 LCD 函数  
    wr_dictate(0x38); //设置数据总线为 8 位, 字符字体为 5*7, 显示两行  
    wr_dictate(0x01); //清屏  
    wr_dictate(0x0c); //开显示,  
  
}  
void display_lcd(uchar address, uchar ch_data)  
{//形式参数 address 为要显示数据所在的地址, ch_data 为要显示的数据  
    wr_dictate(address); //写地址  
    wr_data(ch_data); //写数据  
}  
void main()  
{  
    init_lcd(); //初始化 LCD  
    //0x80 为第一行第一个位置的地址, 0x41 为字符“A”的代码。  
    display_lcd(0x80, 0x41);  
    while(1); //等待  
}
```

实验步骤

1. 打开光盘第 5 章/ show_A / show_A.uv2 工程文件, 对程序进行编译, 链接, 调试产生 show_A.hex 烧写文件。
2. 将实验板的 J4 短接到 LCDP 的一边, J7 短接到 LCDE 的一边。
3. 将 51 仿真器安装到实验板上, 对其进行仿真调试。
4. 将烧写文件烧写到 AT89S52 单片机中去, 再观察其结果。实验效果如 (实图 7)



(实图 7)

动手实验 (6)

实验目的：学习 LCD 如何使其显示一串字符。

实验内容：使 LCD 在第一行的第一个位置开始显示“AT89S52”的字样。

```
#include <reg52.h> //包含头文件
#include<intrins.h> //包含_nop_();库函数的头件
#define dataport P0 //定义 P0 为数据端口
#define uchar unsigned char
#define uint unsigned int
sbit RS=P2^4; //数据/命令选择引脚
sbit RW=P2^3; //读/写选择引脚
sbit E=P2^2; //使能信号
void rd_bf(void)
{ //检测 LCD 是否忙碌
    dataport=0xff; //数据端口先赋高电平
    RS=0; //选择指令寄存器
    RW=1; //读操作
    E=1; //E 为高电平
    _nop_(); //适当延时
    _nop_();
    while(dataport&0x80); //判断数据端口的 D7 是否为 1, 假如为 1 则等待
    E=0; //读取数据完成
}
//写指令函数, dictate 为要写入的指令
wr_dictate(unsigned char dictate)
{
    rd_bf(); //检测 LCD 是否忙碌
    RW=0; //写操作
    RS=0; //选择指令寄存器
    E=1; //E 为高电平
    _nop_(); //适当的延时, 目的是为了等待电平稳定
    _nop_();
    dataport=dictate; //把指令写入 IR
    E=0; //当 E 由高电平变为低电平时指令被写入寄存器
}
//写数据函数, dat 为要写入的数据
wr_data(unsigned char dat)
{
    rd_bf(); //检测 LCD 是否忙碌
    RW=0; //写操作
```

```
RS=1; //选择数据寄存器
E=1; //E 为高电平
_nop_(); //适当的延时, 目的是为了等待电平稳定
_nop_();
dataport=dat; //把数据写入 DR
E=0; //当 E 由高电平变为低电平时数据被写入寄存器
}

void init_lcd(void)
{ //初始化 LCD 函数
    wr_dictate(0x38); //设置数据总线为 8 位, 字符字体为 5*7, 显示两行
    wr_dictate(0x01); //清屏
    wr_dictate(0x0c); //开显示,
}

void display_lcd(uchar address, uchar ch_data)
{ //形式参数 address 为要显示数据所在的地址, ch_data 为要显示的数据
    wr_dictate(address); //写地址
    wr_data(ch_data); //写数据
}

void main()
{
    uchar ch[7]={0x41, 0x54, 0x38, 0x39, 0x53, 0x35, 0x32}; // “AT89S52” 的字符代码
    uchar i;
    init_lcd(); //初始化 LCD
    for(i=0; i<7; i++) //循环写数
        display_lcd(0x80+i, ch[i]); //从 lcd 第一行的第一个位置开始显示 AT89S52
    while(1);
}
```

实验步骤

1. 打开光盘第 5 章/ show_AT89S52 / show_AT89S52.uv2 工程文件, 对程序进行编译, 链接, 调试产生 show_AT89S52.hex 烧写文件。
2. 将实验板的 J4 短接到 LCDP 的一边, J7 短接到 LCDE 的一边。
3. 将 51 仿真器安装到实验板上, 对其进行仿真调试。
4. 将烧写文件烧写到 AT89S52 单片机中去, 再观察其结果。实验效果如 (实图 8)



(实图 8)

动手实验 (7)

实验目的：学习 LCD 的灵活使用。

实验内容：使 LCD 在第一行显示“HELLO!”这六个字符，而且不停地左右来回滚动！

```
#include <reg52.h> //包含头文件
#include<intrins.h> //包含_nop_();库函数的头件
#define dataport P0 //定义 P0 为数据端口
#define uchar unsigned char
#define uint unsigned int
sbit RS=P2^4; //数据/命令选择引脚
sbit RW=P2^3; //读/写选择引脚
sbit E=P2^2; //使能信号
void delay_ms(unsigned int time) //延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time; i>0; i--) //i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112; j>0; j--) //j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void rd_bf(void)
{//检测 LCD 是否忙碌
    dataport=0xff; //数据端口先赋高电平
    RS=0; //选择指令寄存器
    RW=1; //读操作
    E=1; //E 为高电平
    _nop_(); //适当延时
    _nop_();
    while(dataport&0x80); //判断数据端口的 D7 是否为 1, 假如为 1 则等待
    E=0; //读取数据完成
}
//写指令函数, dictate 为要写入的指令
wr_dictate(unsigned char dictate)
{
```

```
rd_bf();//检测 LCD 是否忙碌
RW=0;//写操作
RS=0;//选择指令寄存器
E=1;//E 为高电平
_nop_();//适当的延时, 目的是为了等待电平稳定
_nop_();
dataport=dictate;//把指令写入 IR
E=0;//当 E 由高电平变为低电平时指令被写入寄存器
}
//写数据函数, dat 为要写入的数据
wr_data(unsigned char dat)
{
    rd_bf();//检测 LCD 是否忙碌
    RW=0;    //写操作
    RS=1;    //选择数据寄存器
    E=1;     //E 为高电平
    _nop_();//适当的延时, 目的是为了等待电平稳定
    _nop_();
    dataport=dat;//把数据写入 DR
    E=0;     //当 E 由高电平变为低电平时数据被写入寄存器
}
void init_lcd(void)
{ //初始化 LCD 函数
    wr_dictate(0x38);//设置数据总线为 8 位, 字符字体为 5*7, 显示两行
    wr_dictate(0x01);//清屏
    wr_dictate(0x0c);//开显示,
}
void display_lcd(uchar address, uchar ch_data)
{ //形式参数 address 为要显示数据所在的地址, ch_data 为要显示的数据
    wr_dictate(address);//写地址
    wr_data(ch_data);//写数据
}
void main()
{
    uchar ch[6]={0x48, 0x45, 0x4c, 0x4c, 0x4f, 0x21};//“HELLO!”的字符代码
    uchar i;
    init_lcd();//初始化 LCD
    for(i=0;i<6;i++)//循环写数
        display_lcd(0x80+i, ch[i]);//从 lcd 第一行的第一个位开始显, HELLO!
    for(i=0;i<5;i++)
        wr_dictate(0x07);//把 AC 的值左移 5 位
    while(1)
    {
```

```
for(i=0;i<10;i++)//循环 10 次
{
    //每 500ms 右滚动一次
    wr_dictate(0x1c);
    delay_ms(500);
}
delay_ms(500);
for(i=0;i<10;i++)//循环 10 次
{
    //每 500ms 左滚动一次
    wr_dictate(0x18);
    delay_ms(500);
}
delay_ms(500);
}
}
```

实验步骤

1. 打开光盘第 5 章/ show_move / show_move.uv2 工程文件，对程序进行编译，链接，调试产生 show_move.hex 烧写文件。
2. 将实验板的 J4 短接到 LCDP 的一边，J7 短接到 LCDE 的一边。
3. 将 51 仿真器安装到实验板上，对其进行仿真调试。
4. 将烧写文件烧写到 AT89S52 单片机中去，再观察其结果。实验效果如（实图 9）



（实图 9）

自我练习：

- (1) 自行编写一个程序，使其显示一个“—>”箭头，这个箭头从第一行的第一个位置向右滚动，然后再出现在第二行的第一个位置继续向右滚动，当滚动完之后再从反方向移动，不断地周而复始的循环，将其工程名命名为“arrowhead”。

程序的设计思路：要完成此实验首先要理解图 5-17。我们可以在图 5-17 的第一个位置 0x80 中显示一个“—>”字符图型，再利用滚动指令每 500ms 右移

1 位,共循环 16 次;再在第二行的第一个位置 0xc0 中显示一个“—>”字符图型,同样利用滚动指令每 500ms 右移 1 位,共循环 16 次;第三步与第四步,是要显示一个“←”字符,但是要从 0xcf、0x8f 开始,然后左移,注意:在使用 LCD 之前应先初始化。

- (2) 编写一个程序,使其在第一行的第一个位置显示一个字符“0”,其后光标就在第二个位置不停在闪动,工程名为“bicker”。

程序的设计思路:本练习与前面的动手实验(5)基本相同,但要想光标不停地闪动,只要开启第 4 条指令,将 C=1、B=1,光标就会不停地闪动了。

- (3) 自行编写一个程序,利用自定义字符在 LCD 的第一行第一个位置显示一个“年”字,工程命名为“show_chinese”。

在做练习之前,我们首先了解一下如何对自定义字符进行操作。前面我们曾经讲过,HD44780 有两种字符发生器,一种是 CGROM,即已经固化好的字模库。主要我们写入正确的代码就可以显示出来,前面的几个实验我们就是选用 CGROM 的字符代码。另一种是 CGRAM,即随意定义的字符模库;HD44780 提供了 64B 的 CGRAM,地址是 0x00~0x3f。它可以生成 8 个 5×8 点阵的自定义字符或 5×11 点阵的自定义字符,由于 HD44780 仅使用一行的 5 位数据为字符点阵,所以作为 CGRAM 字模库,仅使用存储单元字节的低 5 位,而高 3 位虽然存在,但不作为字模数据的使用。HD44780 提供给 CGRAM 的字符字模代码为 0x00~0x07 或 0x08~0x0f。作为 5×8 的点阵字符的字模库,CGRAM 每 8B 为一个字符的字模数据,字符数据存储顺序是从上至下排列。每个字符代码都对应着 CGRAM 的 8 个单元,作为 5×11 点阵字符的字模库,CGRAM 每 16B 为一个字符的字符数据,其中前 11B 为字模数据存储单元,后 5B 与字模无关。字符代码与 CGRAM 地址的对应关系为如表 5-7。

表 5-7

5×8 点阵字符		5×11 点阵字符	
字符代码	CGRAM 地址	字符代码	CGRAM 地址
0x00(0x08)	0x00~0x07	0x00(0x08)	0x00~0x0f
0x01(0x09)	0x08~0x0f	0x01(0x09)	0x10~0x1f
0x02(0x0a)	0x10~0x17	0x02(0x0a)	0x20~0x2f
0x03(0x0b)	0x18~0x1f	0x03(0x0b)	0x30~0x3f
0x04(0x0c)	0x20~0x27		
0x05(0x0d)	0x28~0x2f		
0x06(0x0e)	0x30~0x37		
0x07(0x0f)	0x38~0x3f		

就如表 5-8 所示,如果我们要想在液晶屏上显示一个“年”字,那么我们只要将相应的点阵置“1”,而无需显示的则置“0”,最后把代码写进 DDRAM 中,那么 HD44780 就会自动到 CGRAM 中读取字符在液晶屏中显示出来。(表 5-8 是“年”字符及 CGRAM 地址与显示效果的对应关系)

表 5-8

显示屏的显示效果	CGRAM单元数据	CGRAM地址
■	0x08	0x00
■ ■ ■ ■	0x0F	0x01
■ ■ ■ ■	0x12	0x02
■ ■ ■ ■	0x0F	0x03
■ ■ ■ ■	0x0A	0x04
■ ■ ■ ■	0x1F	0x05
■ ■ ■ ■	0x02	0x06
■ ■ ■ ■	0x02	0x07

- 程序的设计思路：
- (1) 先定义一个数组，然后赋予上面“年”字的点阵字符数据，例如：

```
uchar year[8]={0x08, 0x0F, 0x12, 0x0F, 0x0A, 0x1F, 0x02, 0x02 };
```
 - (2) 在主程序的开始首先初始化 LCD。
 - (3) 先选取 CGRAM 操作指令，然后利用循环语句把上面 year 数组的每一个元素写进 CGRAM 的 0x00~0x07 地址。
 - (4) 再选取 DDRAM 操作指令，把代码 0x00 写到你要显示的位置上，那么液晶屏就会显示一个“年”字。

(练习答案附光盘中)

第四节 I2C 总线接口技术

什么是 I2C 总线？I2C (Inter-Integrated Circuit) 总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。也可以简单地理解为 I2C 是微控制器与外围芯片的一种通讯协议。在不同的书籍中，可能会称为 I2C，IIC，或者 I 平方 C，但是概念也是一样的，只是叫法不同。

一、I2C 总线特点

I2C 总线的优点非常多，其中最主要体现在 1：硬件结构上具有相同的接口界面；2：电路接口的简单性；3：软件操作的一致性。I2C 总线占用芯片的引脚非常的少，只需要两组信号作为通信的协议，一条为数据线 (SDA)，另一条为时钟线 (SCL)。因此减少了电路板的空间和芯片管脚的数量，所以降低了互联成本。总线的长度可高达 25 英尺，并且能够以 10Kbps 的最大传输速率支持 40 个组件。I2C 总线还具备了另一个优点，就是任何能够进行发送和接收数据的设备都可以成为主控机。当然，在任何时间点上只能允许有一个主控机。

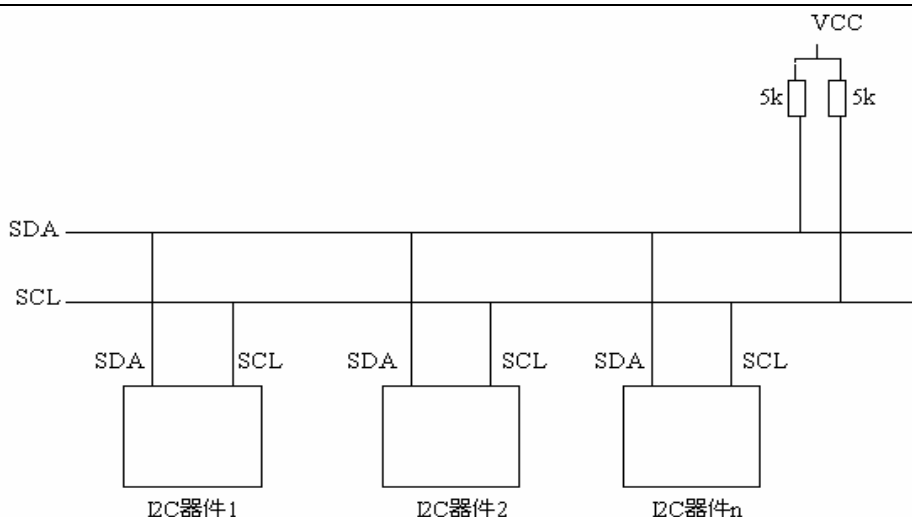


图 5-20 (总线连接图)

二、I2C 总线工作原理

图 5-20 为 I2C 总线的连接图。I2C 总线是由数据线 SDA 和时钟线 SCL 构成的串行总线，可发送和接收数据。在单片机与被控 IC 之间，最高传送速率 100kbps。各种 I2C 器件均并联在这条总线上，就像电话线网络一样不会互相冲突，要互相通信就必须拨通其电话号码，每一个 I2C 模块都有唯一地址。并接在 I2C 总线上的模块，既可以是主控器（或被控器），也可以是发送器（或接收器），这取决于它所要完成的功能。I2C 总线在传送数据过程中共有四种类型信号，它们分别是：起始信号、停止信号、应答信号与非应答信号。

三、I2C 总线数据的传送规则

起始信号：在 I2C 总线工作过程中，当 SCL 为高电平时，SDA 由高电平向低电平跳变，定义为起始信号，起始信号由主控机产生。如图 5-21 所示

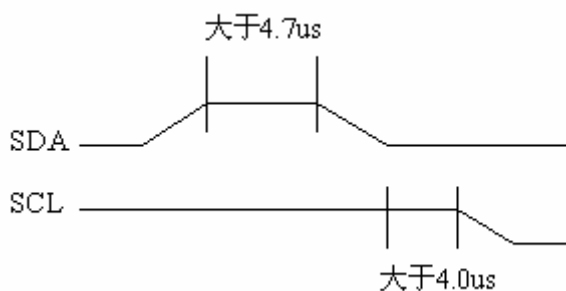


图 5-21 (开始信号)

停止信号：当 SCL 为高电平时，SDA 由低电平向高电平跳变，定义为停止信号，此信号也只能由主控机产生。如图 5-22 所示。

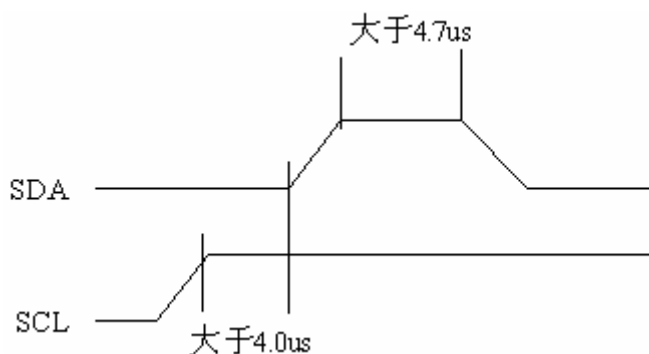


图 5-22 (停止信号)

应答信号：I2C 总线传送的每个字节为 8 位，受控的器件在接收到 8 位数据后，在第 9 个脉冲必须输出低电平作为应答信号，同时，要求主控器在第 9 个时钟脉冲位上释放 SDA 线，以便受控器发出应答信号，将 SDA 拉低，表示接收数据的应答（如图 5-23 所示）。若果在第 9 个脉冲收到受控器的非应答信号（如图 5-24 所示），则表示停止数据的发送或接收。

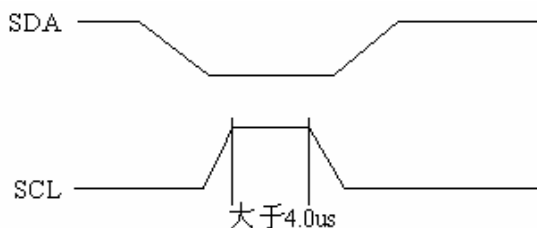
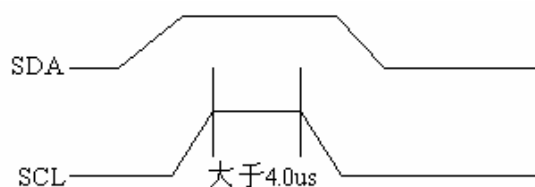


图 5-23 (应答信号)



5-24 (非应答信号)

其次，每启动一次总线，传输的字节数没有限制。主控件和受控器件都可以工作于接收和发送状态。总线必须由主器件控制，也就是说必须由主控器产生时钟信号、起始信号、停止信号。在时钟信号为高电平期间，数据线上的数据必须保持稳定，数据线上的数据状态仅在时钟为低电平的期间才能改变（如图 5-25），而当时钟线为高电平的期间，数据线状态的改变被用来表示起始和停止条件（如图 5-21 与 5-22 所示）。

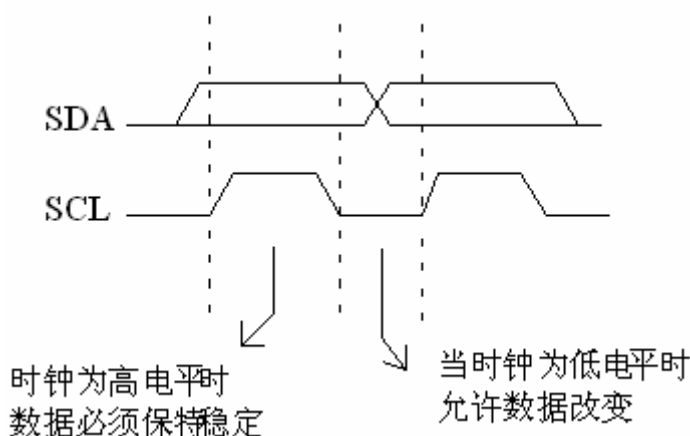


图 5-25 (数据的有效性)

图 5-26 为总线的完整时序，在这里有一点要加以说明的，当主控器接收数据时，在最后一个数据字节，必须发送一个非应答信号，使受控器释放数据线，以便主控器产生一个停止信号来终止总线的数据传送。

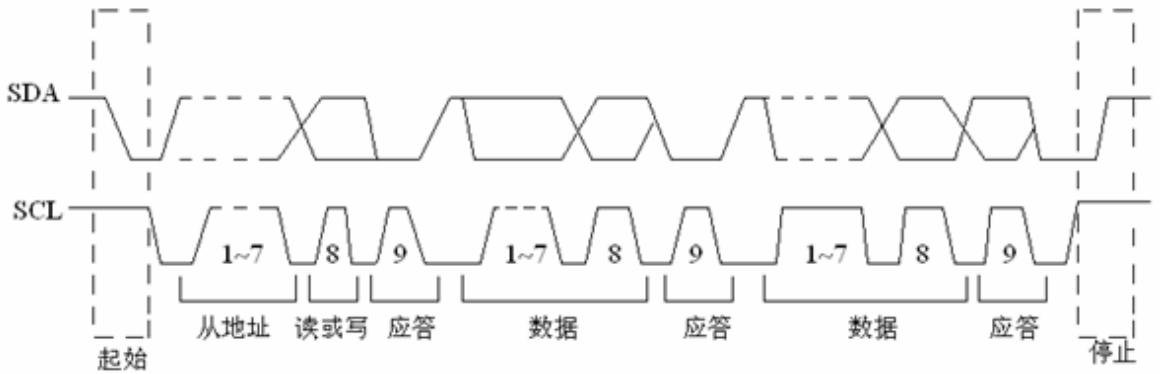


图 5-26 (总线的完整时序)

下面我们来看一下关于 I2C 总线的读操作与写操作：

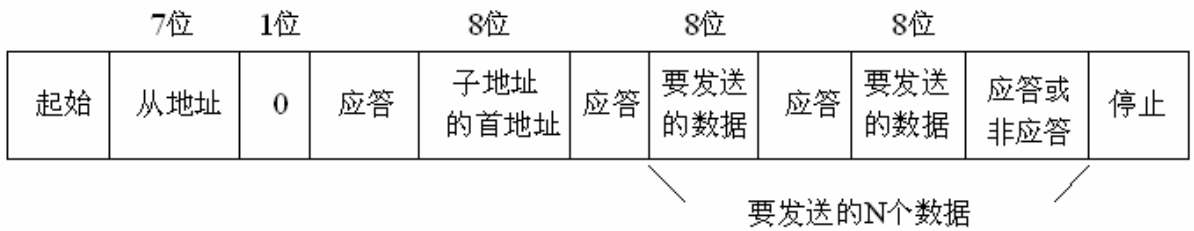


图 5-27 (总线写格式)

写操作就是主控器件向受控器件发送数据，如图 5-27 所示。首先，主控器会对总线发送起始信号，紧跟应该是第一个字节的 8 位数据，但是从地址只有 7 位，所谓从地址就是受控器的地址，而第 8 位是受控器约定的数据方向位，“0”为写，从图 5-26 中我们可以清楚地看到发送完一个 8 位数之后应该是一个受控器的应答信号。应答信号过后就是第二个字节的 8 位数据，这个数据一般是受控器件的寄存器地址，寄存器地址过后就是要发送的数据，当数据发送完后就是一个应答信号，每启动一次总线，传输的字节数没有限制，一个字节地址或数据过后的第 9 个脉冲是受控器件应答信号，当数据传送完之后由主控器发出停止信号来停止总线。

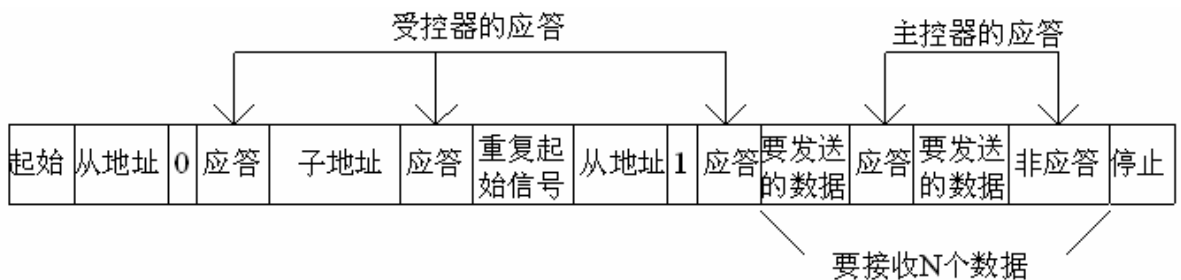


图 5-28 (总线读格式)

读操作指受控器件向主控制器件发送数，其总线的操作格式如图 5-28。首先，由主控制器发出起始信号，前两个传送的字节与写操作相同，但是到了第二个字节之后，就要从新启动总线，改变传送数据的方向，前面两个字节数据方向为写，即“0”；第二次启动总线后数据方向为读，即“1”；之后就是要接收的数据。从图 5-28 的写格式中我们可以看到有两种的应答信号。一种是受控器的，另一种是主控器的。前面三个字节的数据方向均指向受控器件，所以应答信号就由受控器发出。但是后面要接收的 N 个数据则是指向主控制器件，所以应答信号应由主控制器件发出，当 N 个数据接收完成之后，主控制器件应发出一个非应答信号，告知受控器件数据接收完成，不用再发送。最后的停止信号同样也是由主控制器发出。

四、支持 I2C 总线的器件

在当今市面上有部分的单片机是内置硬件 I2C 总线的，用户只需要设置好内部相关的寄存器就可以灵活地运用它。但是如果不内置硬件 I2C，我们在使用过程中可以用普通的 I/O 端口进行模拟。如实验板的 AT89S52 芯片，如果要用到 I2C 协议就得要用到软件模拟。

而在非单片机类的芯片当中，如时钟芯片 PCF8563，存储器 24Cxx 系列等都是使用 I2C 协议进行数据的操作，而我们实验板用的则是 24C02。下面我们先来介绍一下 24C02 的引脚功能和内部结构，然后再介绍如何利用单片机模拟 I2C 协议与 24C02 进行数据的传送。

五、I2C 器件 24C02

24C02 是一个 2K 串行 CMOS E²PROM，内部含有 256 个 8 位字节，该器件通过 I2C 总线接口进行操作。图 5-29 为其功能引脚的排列，而 5-30 为引脚的功能列表。

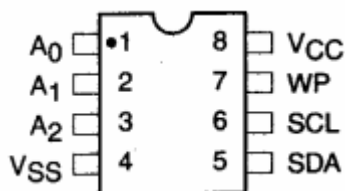


图 5-29 (24C_{xx}引脚排列)

管脚名称	功能
A0、 A1 、 A2	这三个引脚用于多个器件同时使用时设置区分器件地址，当这些引脚悬空时默认为 0。在同一总线中最多可同时使用 8 个 24C02 器件。如果总线只有一个 24C02 器件被寻址，这三个地址可悬或接地。
SDA	双向串行数据/地址管脚，用于器件所有数据的发送或接收，SDA 是一个开漏输出管脚。
SCL	串行时钟。串行时钟输入管脚用于产生器件所有数据发送或接收的时钟，这是一个输入管脚
	写保护引脚。当 WP 引脚连接到 VCC 时芯片里面的内容为只读内容而不能进行写操作。而当 WP 引脚连接到 VSS 时芯片里面的内容可进行正常

WP	的读/写操作。
VCC	+1.8V~6.0V 工作电压
VSS	地

图 5-30 (24C_{xx}引脚功能)

六、24C02 的从地址

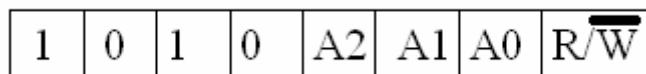


图 5-31 (24C02 从器件地址)

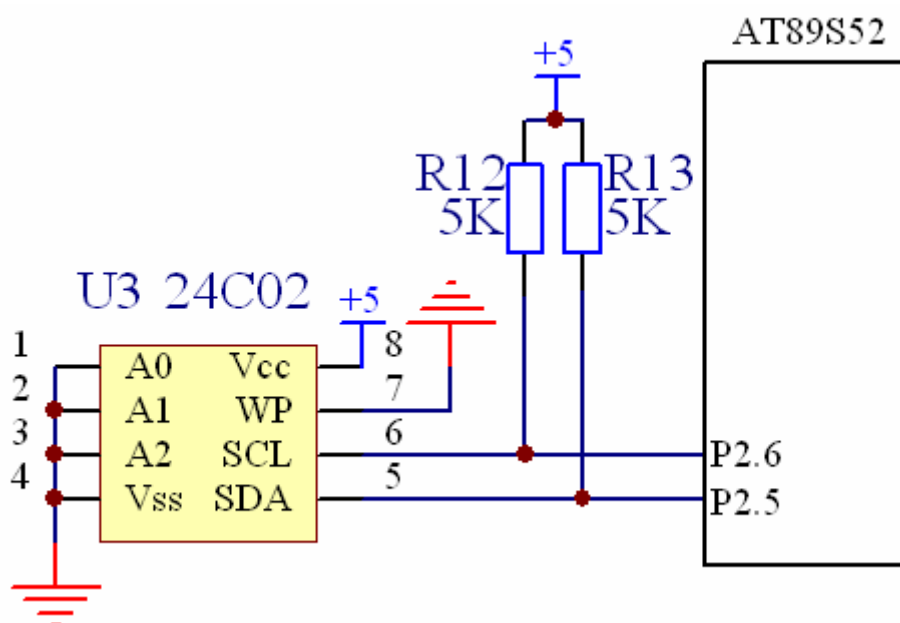


图 5-32 (实验电路)

从图 5-31 中我们可以看到从地址的高 4 位是固定的，而低 4 位是可根据 A0、A1、A2 引脚的接法与数据的方向来确定的，其中 R/W 为“0”时表示写，为“1”时表示读。而图 5-32 为实验板电路，A0、A1、A2 均接地，即为 0、0、0。所以在实验板中 24C02 器件的写从地址为 1010 0000（转为十六进制数为 0xa0），读从地址为 1010 0001（转为十六进制数为 0xa1）。

动手实验（8）：

实验目的：利用 I2C 总线协议，试验 24C02 断电数据保存的特性。

实验内容：下面有两个程序，其中程序一为把 0x55,0xaa 这两个数写进 24C02 器件中；然后把实验板断电一会儿后再利用程序二把 24C02 的数据读出来，同时发送到串口调试器中，试看数据在断电期间是否得到保存。

程序一：（附光盘的 i2c_W 文件夹中）

```
#include<reg52.h>
#include <intrins.h>
#define uchar unsigned char
#define uint unsigned int
#define Write 0xa0//写操作
#define Read 0xa1//读操作
sbit SCL=P2^6;//时钟引脚
sbit SDA=P2^5;//数据引脚
sbit led=P1^0;
void delay_ms(unsigned int time)//延时 1 毫秒程序， n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {};
}
void lay(void)//延时函数,约延时 10us
{
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
}
void Start_Cond(void)//启动总线
{
    SCL=0;
    lay();
    SCL=1;//发送起始条件时钟信号
    lay();
    SDA=1;//发送起始条件数据信号
    lay();//起始条件信号大于 4.7us
    SDA=0;
    lay();
    SCL=0;//锁住总线，准备发送数据
}
void Stop_Cond(void)//停止总线
{
```

```
    SDA=0;//发送结束条件数据信号
    SCL=0;
    lay();
    SCL=1;//发送结束条件时钟信号
    lay();//停止条件信号大于 4.7us
    SDA=1;
    lay();
    SDA=0;
}
void Ack(void)//应答信号
{
    SCL=0;//准备发送应答信号
    SDA=0;
    lay();
    SCL=1;//时钟信号保持高电平为大于 4.7us
    lay();
    SCL=0;//时钟线置低电平
    lay();
    SDA=1;//结束应答信号
}
void NoAck(void)//非应答信号
{
    SDA=0;//准备发送非应答信号
    SCL=0;
    lay();
    SDA=1;//时钟线与数据线为高电平，则为非应答信号
    SCL=1;
    lay();//保持高电平为大于 4.0us
    SCL=0;//结束非应答信号
    lay();
    SDA=0;
}
void Write8Bit(uchar Word)//向总线写入 word 变量的 8 位数据
{
    uchar i,temp;
    temp=Word;
    for(i=0;i<8;i++)
    {
```



```

        SCL=0;//当 SCL 为低电平时允许数据改变
        temp<<=1;//移位
        SDA=CY;//把移出的数据发送到总线上
        SCL=1;//当时钟线为 1 时，数据必顺保持稳定
    }
    SCL=0;//每发送 8 位数据要跟随一个时钟信号，等待从机作出应答
    lay();
    SCL=1;
}
uchar Read8Bit(void)
{
    uchar i,Result;
    Result=0;
    for(i=0;i<8;i++)
    {
        SCL=0;//时钟为低电平，准备接收数据
        lay();//时钟低电平大于 4.7us
        SCL=1;//当时钟线为高电平时，数据线上的数据有效
        Result=(Result<<1)|SDA;//从数据线读取数据
    }
    return Result;//返回从总线上读取到的数据
}
/*****
Read_Flash 函数，Array 为接收到数据所存放的地址，nAdd 为要发
送的器件子地址，nLen 为所读取数据的长度
*****/
void Read_Flash(uchar *Array,uchar nAdd,uchar nLen)//从总线读取多个字节函数
{
    Start_Cond();//启动总线
    Write8Bit(Write);//发送器件地址
    Write8Bit(nAdd);//发送器件子地址

    Start_Cond();//从新启动总线
    Write8Bit(Read);//读器件地址
    while(--nLen)
    {
        *Array=Read8Bit();//读 8 位数据
    }
}

```

```
        Array++;//Array 指向下一个要存放数据的地址
        Ack();//每接收到 8 位数据，跟随一个应答信号
    }
    *Array=Read8Bit();//最后再读多 8 位
    NoAck();//发送非应答信号，停止读取数据
    Stop_Cond();//停止总线

}
/*****
Write_Flash 函数，Array 为要发送数据的地址，nAdd 为要发
送的器件子地址，nLen 为所发送数据的长度
*****/
void Write_Flash(uchar *Array,uchar nAdd,uchar nLen)//发送多个字节到总线函数
{
    uchar i;
    Start_Cond();//启动总线
    Write8Bit(Write);//发送器件地址
    Write8Bit(nAdd);///发送器件子地址
    for(i=0;i<nLen;i++)
    {
        Write8Bit(*Array);//写 8 位数据到总线
        Array++;//Array 指向下一个要发送的数据
    }
    Stop_Cond();//结束总线
}

void main(void)
{
    uchar send_da[2];//存放要发送到 24c02 器件的数据
    send_da[0]=0x55;
    send_da[1]=0xaa;
    led=0;//点亮二极管
    /*从 0x00 的子地址写 send_da 数组的 2 个数据*/
    Write_Flash(send_da,0x00,2);
    delay_ms(1000);
    led=1;//当操作完成之后熄灭二极管
    while(1);
}
```

程序二：（附光盘的 i2c_R 文件夹中）

```
#include<reg52.h>
#include <intrins.h>
#define uchar unsigned char
#define uint unsigned int
#define Write 0xa0//写操作
#define Read 0xa1//读操作
sbit SCL=P2^6;//时钟引脚
sbit SDA=P2^5;//数据引脚
sbit led=P1^0;
void delay_ms(unsigned int time)//延时 1 毫秒程序，n 是形式参数
{
    unsigned int i,j;
    for(i=time;i>0;i--)//i 不断减 1，一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1，一直到 j>0 条件不成立为止
            {};
}
void lay(void)//延时函数,约延时 10us
{
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
}
void Start_Cond(void)//启动总线
{
    SCL=0;
    lay();
    SCL=1;//发送起始条件时钟信号
    lay();
    SDA=1;//发送起始条件数据信号
    lay();//起始条件信号大于 4。7us
    SDA=0;
    lay();
    SCL=0;//锁住总线，准备发送数据
}
void Stop_Cond(void)//停止总线
```

```
{
    SDA=0;//发送结束条件数据信号
    SCL=0;
    lay();
    SCL=1;//发送结束条件时钟信号
    lay();//停止条件信号大于 4。7us
    SDA=1;
    lay();
    SDA=0;
}
void Ack(void)//应答信号
{
    SCL=0;//准备发送应答信号
    SDA=0;
    lay();
    SCL=1;//时钟信号保持高电平为大于 4。7us
    lay();
    SCL=0;//时钟线置低电平
    lay();
    SDA=1;//结束应答信号
}
void NoAck(void)//非应答信号
{
    SDA=0;//准备发送非应答信号
    SCL=0;
    lay();
    SDA=1;//时钟线与数据线为高电平，则为非应答信号
    SCL=1;
    lay();//保持高电平为大于 4。0us
    SCL=0;//结束非应答信号
    lay();
    SDA=0;
}
void Write8Bit(uchar Word)//向总线写入 word 变量的 8 位数据
{
    uchar i,temp;
    temp=Word;
    for(i=0;i<8;i++)
```

```
{
    SCL=0;//当 SCL 为低电平时允许数据改变
    temp<<=1;//移位
    SDA=CY;//把移出的数据发送到总线上
    SCL=1;//当时钟线为 1 时，数据必顺保持稳定
}
    SCL=0;//每发送 8 位数据要跟随一个时钟信号，等待从机作出应答
    lay();
    SCL=1;
}
uchar Read8Bit(void)
{
    uchar i,Result;
    Result=0;
    for(i=0;i<8;i++)
    {
        SCL=0;//时钟为低电平，准备接收数据
        lay();//时钟低电平的同大于 4。7us
        SCL=1;//当时钟线为高电平时，数据线上的数据有效
        Result=(Result<<1)|SDA;//从数据线读取数据
    }
    return Result;//返回从总线上读取的数据
}
/*****
Read_Flash 函数，Array 为接收到数据所存放的地址，nAdd 为要发
送的器件子地址，nLen 为所读取数据的长度
*****/
void Read_Flash(uchar *Array,uchar nAdd,uchar nLen)//从总线读取多个字节函数
{
    Start_Cond();//启动总线
        Write8Bit(Write);//发送器件地址
    Write8Bit(nAdd);//发送器件子地址
    Start_Cond();//从新启动总线
    Write8Bit(Read);//读器件地址
    while(--nLen)
    {
        *Array=Read8Bit();//读 8 位数据
        Array++;//Array 指向下一个要存放数据的地址
    }
}
```

```
    Ack();//每接收到 8 位数据，跟随一个应答信号
}
*Array=Read8Bit();//最后再读多 8 位
NoAck();//发送非应答信号，停止读取数据
Stop_Cond();//停止总线

}
/*****
Write_Flash 函数，Array 为要发送数据的地址，nAdd 为要发
送的器件子地址，nLen 为所发送数据的长度
*****/
void Write_Flash(uchar *Array,uchar nAdd,uchar nLen)//发送多个字节到总线函数
{
    uchar i;
    Start_Cond();//启动总线
    Write8Bit(Write);//发送器件地址
    Write8Bit(nAdd);//发送器件子地址
    for(i=0;i<nLen;i++)
    {
        Write8Bit(*Array);//写 8 位数据到总线
        Array++;//Array 指向下一个要发送的数据
    }
    Stop_Cond();//结束总线
}
void uart(void)//串行口初始化函数，晶振为 11.0592MHZ，波特率为 9600
{
    SCON=0x40;//选择工作方式 1
    PCON=0x00;//不选用波特率倍增
    REN=1;    //允许接收
    TI=0;    //清零发送标志位
    RI=0;    //清零接收标志位
    TMOD=0x20;//先用定时器 1 的工作方式 2
    TH1=0xfd; //对定时器 1 高位赋初值，
    TL1=0xfd; //对定时器 1 低位赋初值，
    TR1=1;    //开启定时器 1
}
void send(uchar _data)//串口发送函数
{
```

```
SBUF=_data;//把数据装进串口缓冲寄存器
while(TI==0);//等待数据发送完成
TI=0;//清发送标志位
}
void main(void)
{
    uchar incept[2];//存放从 24c02 器件读到的数据
    uart();//串口初始化。
    led=0;
    /*从 0x00 的子地址读取两个数据存放到 incept 数组中*/
    Read_Flash(incept,0x00,2);
    send(incept[0]);//把 incept 数组发送到串口调试器中去
    send(incept[1]);
    delay_ms(1000);
    led=1;//当操作完成之后熄灭二极管
    while(1);
}
```

实验步骤:

1. 打开光盘第 5 章/ i2c_W / i2c_W.uv2 工程文件, 对程序进行编译, 链接, 调试产生 i2c_W.hex 烧写文件。将烧写文件烧写到 AT89S52 中去。通上电源, 当 LED 灯熄灭后表示写数据完成, 将实验板断电一会儿。(i2c_W 就是上面的程序一)
2. 打开光盘第 5 章/ i2c_R / i2c_R.uv2 工程文件, 对程序进行编译, 链接, 调试产生 i2c_R.hex 烧写文件。将烧写文件烧写到 AT89S52 中去。(i2c_R 就是上面的程序二)
3. 打开串口调试器, 选择波特率为 9600, 选择十六进制接收, 然后打开串口。
4. 把刚才烧有 i2c_R 程序的单片机插到实验板中去, 通上电源, 当 LED 灯熄灭之后, 就可以在串口调试器的接收区见到 0x55、0xaa 两个数据(如下图 5-32 所示)。

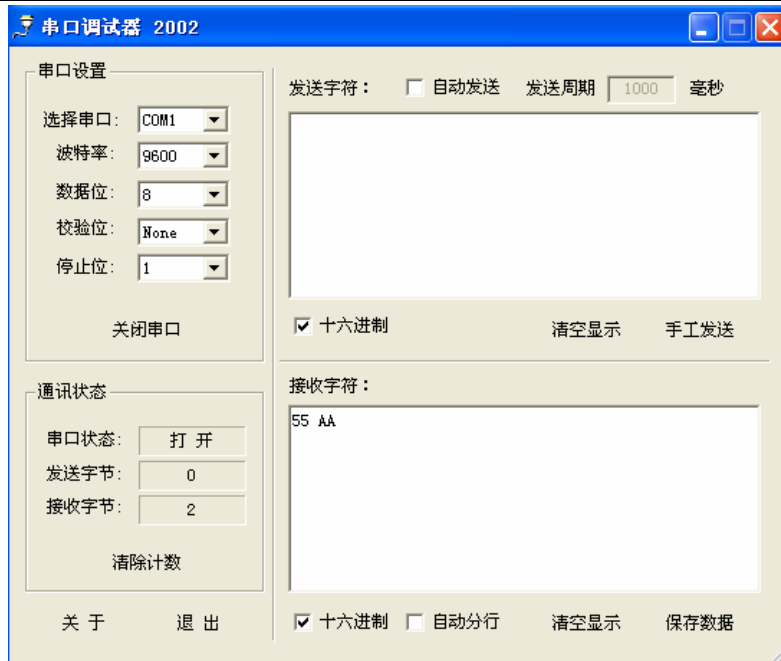


图 5-32

从上面动手实验我们可以清楚地了解到，我们把电路系统中的重要数据写进了 24C02 数据存储单元中去，即使断电，数据还是可以保存在器件当中。这样就可以避免了当电路系统断电后数据丢失的弊端。

自我练习：自行编写一个程序，工程名命名为“i2c”。在同一个程序当中，先从 0x00 的地址开始写 5 个数据“0x01,0x02,0x03,0x04,0x05”，然后再从 0x00 地址开始读 5 个数据，把读到的数据发送到串口调试器中去。从而测试 I2C 总线协议的读写是否成功。

程序的设计思路：1、在程序的一开始应定义两个数组，一个为存放从器件读到的数据：incept[5]；而另一个为存放要写到器件的数据：send_da[5]，同时存放写到器件的数组应初始化为：

“0x01,0x02,0x03,0x04,0x05”。

- 2、在没有读数据之前把 incept[5] 的五个元素利用串口函数发送到调试器。
- 3、利用 Write_Flash(send_da, 0x00, 5) 函数把数据写进 24C02 数据存储单元中去。
- 4、适当延时一段时间之后，再利用以下函数：

Read_Flash(incept, 0x00, 5); 把数据从 24C02 器件中读出，同时发送到串口调试器中去。对比 incept[5] 数组在读 24C02 的前后就可以得知数据是否读写成功。

(以上练习附光盘中)

第五节 实时时钟芯片 DS1302

DS1302 是美国 DALLAS 公司推出的一种带串行通信接口的实时时钟芯片，因为其既可以提供实时时钟，又可以把重要的数据存放在 RAM 中，再加上它的高性能、低功耗附加 31 字节静态 RAM，所以在智能化仪表及自动控制领域得到了广泛应用。（图 5-33 为其引脚分配图，表 5-10 是引脚功能图，图 5-34 为实验板的实用电路）

- 接口方式：采用 SPI 三线接口与 CPU 进行同步通信，并可采用突发方式一次传送多个字节的时钟数据和 RAM 数据。
- 实时时钟可提供秒、分、时、日、星期、月和年，一个月小于 31 天时可以自动调整，且具有闰年补偿功能。
- 工作电压在 2.5~5.5V。采用双电源供电（主电源和备用电源），Vcc1：主电源；Vcc2：备用电源。当 $V_{cc2} > V_{cc1} + 0.2V$ 时，由 Vcc2 向 DS1302 供电，当 $V_{cc2} < V_{cc1}$ 时，由 Vcc1 向 DS1302 供电。DS1302 还提供了对后备电源进行涓细电流充电的能力。（后备电源也可由大容量电容来替代）
- 31×8 个静态 RAM 可供使用。

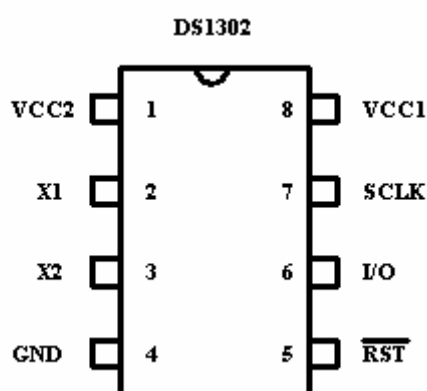


图 5-33 DS1302 的外部引脚功能

表 5-10

引脚号	引脚名称	说明
1	Vcc2	主电源
2、3	X1 X2	振荡源，外接 32.768K 晶振，为芯片提供计时脉冲。
4	GND	地线
5	RST	复位/片选线
6	I/O	串行数据输入/输出
7	SCLK	串行时钟引脚
8	Vcc1	后备电源

SCLK：串行时钟输入。

I/O：三线接口的双向数据线。

RST：输入信号，在读、写数据期间，必须为高。该引脚有两个功能：第一、RST 开始控制字访问移位寄存器的控制逻辑；其次，RST 提供结束单字节或多字节数据传输的方法。

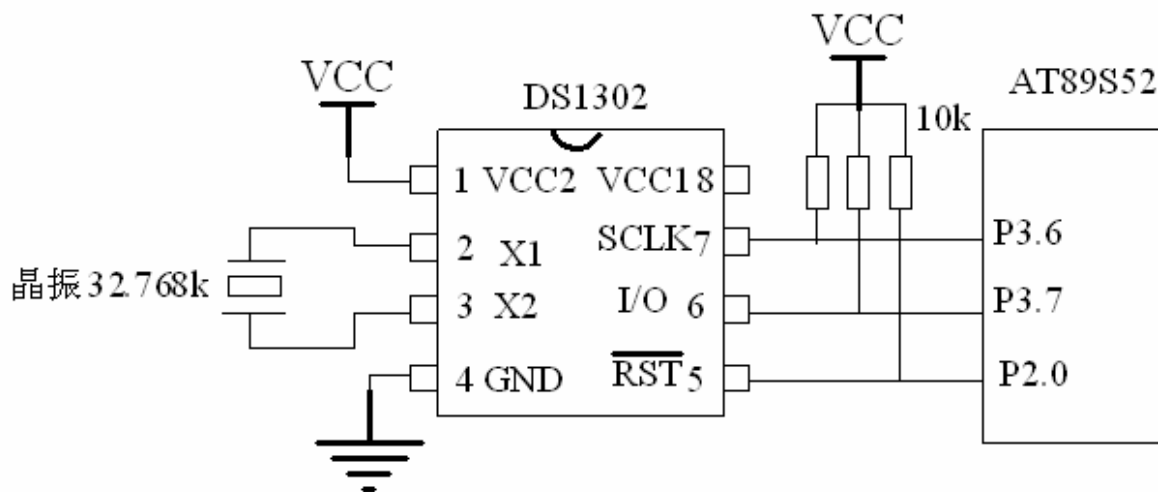


图 5-34 (实验板的实用电路 注：后备电源未使用)

一、DS1302 的控制字与读写时序

要想利用单片机对 DS1302 进行操作，那么要了解其读写时序与控制字是必要的，图 5-35 就是 DS1302 的控制字。

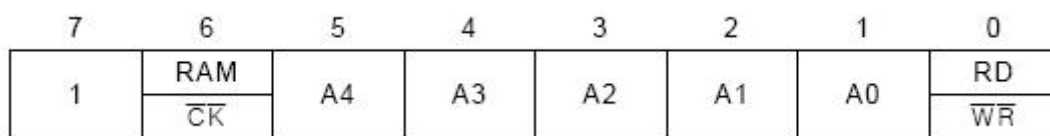


图 5-35 控制字 (即地址及命令字节)

- 位 7: (高位) 必须是逻辑 1, 如果它为 0, 则不能把数据写入到 DS1302 中。
- 位 6: 如果为 0, 则表示存取日历时钟数据, 为 1 表示存取 RAM 数据;
- 位 5 至位 1 (A4~A0): 指示操作单元的地址;
- 位 0: (低位) 如为 0, 表示要进行写操作, 为 1 表示进行读操作。

图 5-36、图 5-37 分别是单片机对 DS1302 的单字节读时序与单字节写时序。

控制字总是从最低位开始输出。在控制字指令输入后的下一个 SCLK 时钟的上升沿时, 数据被写入 DS1302, 数据输入从最低位 (0 位) 开始。同样, 在紧跟 8 位的控制字指令后的下一个 SCLK 脉冲的下降沿, 读出 DS1302 的数据, 读出的数据也是从最低位到最高位。

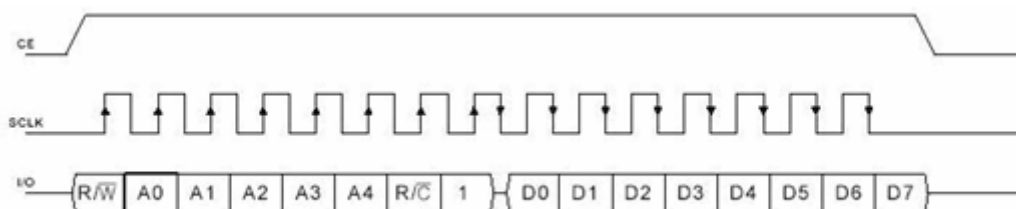


图 5-36 单字节读时序

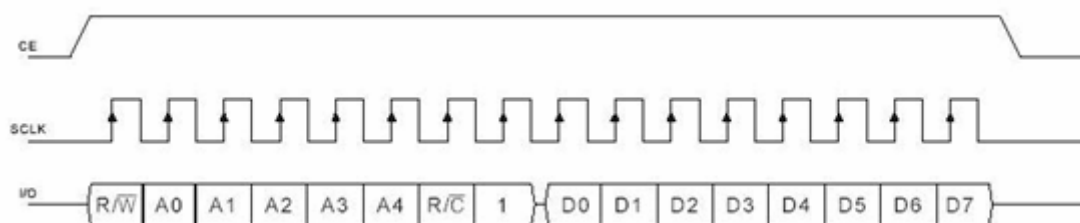


图 5-37 单字节写时序

DS1302 是通过 SPI 串行总线驱动方式，它不仅可以向寄存器写入控制字，还可以读取相应寄存器的数据。

二、关于 DS1302 的内部寄存器

(1) DS1302 有关日历、时间的寄存器共有 12 个，其中有 7 个寄存器（读时为 81h~8Dh，写时为 80h~8Ch），存放的数据格式为 BCD 码形式，如表 5-11 所示。

表 5-11 (DS1302 有关日历、时间的寄存器)

读寄存器	写寄存器	BIT7	DIT6	DIT5	DIT4	DIT3	DIT2	DIT1	DIT0	范围
0x81	0x80	CH	10 秒			秒				00-59
0x83	0x82		10 分			分				00-59
0x85	0x84	12/24	0	10	时	时				1-12 0-23
				AM/PM						
0x87	0x86	0	0	10 日		日				1-31
0x89	0x88	0	0	0	10 月	月				1-12
0x8b	0x8a	0	0	0	0	0	周日			1-7
0x8d	0x8c	10 年			年					00-99
0x8f	0x8e	WP	0	0	0	0	0	0	0	--

- 秒寄存器（81h、80h）的位 7 定义为时钟暂停标志位（CH）。当该位置为“1”时，时钟振荡器停止，DS1302 处于低功耗状态；当该位置为“0”时，时钟开始运行。
- 小时寄存器（85h、84h）的位 7 用于定义芯片在 12 小时与 24 小时之间的切换，当此位为“1”时，选择 12 小时模式，当此位为“0”时选择 24 小时。

在 12 小时模式时，位 5 为定义 PM 与 AM 的标志位，当此位为“1”时，表示 PM。当此位为“0”时表示选择 AM。在 24 小时模式时，位 5 是第二个 10 小时位。

- 控制寄存器（8Fh、8Eh）的位 7 是写保护标志位（WP），其它位 6~0 均置为 0。在任何情况下对时钟和 RAM 的写操作之前，WP 位必须为“0”。当 WP 位为“1”时，写保护位防止对任一寄存器的写操作。

(2) 下面我们来介绍 DS1302 有关 RAM 的地址

DS1302 中附加 31 字节静态 RAM 的地址如表 5-12 所示。

表 5-12

读地址	写地址	数据范围
0xC1	0xC0	0x00-0xFF
0xC3	0xC2	0x00-0xFF
0xC5	0xC4	0x00-0xFF
·	·	
·	·	

.	.	.
0xFD	0xFC	0x00-0xFF

(3) 关于 DS1302 的突发工作模式寄存器

我们在前面提及 DS1302 芯片有突发模式操作功能，所谓突发模式是指一次传送多个字节的时钟数据或 RAM 数据。突发模式寄存器如图表 5-13 所示。

表 5-13

工作模式	读寄存器	写寄存器
时钟突发模式寄存器	0xBF	0xBE
RAM 突发模式寄存器	0xFF	0xFE

下面我们再利用动手实验来对其芯片作进一步的了解。

动手实验 (9)

实验目的: 学习 DS1302 时钟芯片的使用。

实验内容: 利用 AT89S52 单片机每秒读取 DS1302 的时钟数据 1 次, 在 LCD 晶液屏中显示出来。DS1302 的初始化时间为: 4 月、13 日、9 时、24 分、45 秒。

```
#include<reg52.h>
#include<stdio.h>
#include <intrins.h>
#include"lcd.h"
#define uchar unsigned char
sbit led=P1^0;//led 灯定义
//位寻址寄存器定义
sbit ACC7 = ACC^7;
sbit ACC0 = ACC^0;
//管脚定义
sbit SCLK = P3^6; // DS1302 时钟信号 7 脚
sbit DIO= P3^7; // DS1302 数据信号 6 脚
sbit RST = P2^0; // DS1302 片选 5 脚
//寄存器宏定义
#define WRITE_SECOND 0x80//写秒
#define READ_SECOND 0x81//读秒
#define WRITE_MINUTE 0x82//写分钟
#define READ_MINUTE 0x83//读分钟
#define WRITE_HOUR 0x84//写小时
#define READ_HOUR 0x85//读小时
#define WRITE_DAY 0x86//写日
#define READ_DAY 0x87//读日
#define WRITE_MONTH 0x88//写月
#define READ_MONTH 0x89//读月
#define WRITE_PROTECT 0x8E//写保护
void DS1302writeByte(uchar _data) //向 DS1302 写入一个字节
{
    uchar i;
    ACC = _data;
    for(i=8; i>0; i--)
    {
```

```
DIO = ACC0;//写一位数据
SCLK = 1; //时钟信号
SCLK = 0;
ACC = ACC >> 1;//移位,准备好下次要写的数据。
}
}
uchar DS1302readByte(void)//向 DS1302 读取一个字节
{
    uchar i;
    for(i=8; i>0; i--)
    {
        ACC = ACC >>1;//移位,以便下次存放读出的数据
        ACC7 = DIO;//读 1 位数据
        SCLK = 1;//时钟信号
        SCLK = 0;
    }
    return(ACC);//返回读到的数据
}
void Write1302(uchar ucAddr, uchar ucDa)//写 DS1302 寄存器
{
    RST = 0;
    SCLK = 0;
    RST = 1;
    DS1302writeByte(ucAddr); // 地址, 命令
    DS1302writeByte(ucDa); // 写 1Byte 数据
    SCLK = 1;
    RST = 0;
}
uchar Read1302(uchar ucAddr) //读 DS1302 的寄存器内容
{
    uchar ucData;
    RST = 0;
    SCLK = 0;
    RST = 1;
    DS1302writeByte(ucAddr|0x01); // 地址, 命令
    ucData = DS1302readByte(); // 读 1Byte 数据
    SCLK = 1;
    RST = 0;
    return(ucData);
}
void get_time(uchar *time)
{
    *(time+4)=Read1302(READ_SECOND);//读取秒钟
    *(time+3)=Read1302(READ_MINUTE);//读取分钟
    *(time+2)=Read1302(READ_HOUR); //读取小时
    *(time+1)=Read1302(READ_DAY); //读取日
    *(time+0)=Read1302(READ_MONTH); //读取月
}
void Initial(void) //初始化 DS1302
{
    //初始化为 4 月, 13 日, 9 时, 24 分, 45 秒
```

```

Write1302 (WRITE_PROTECT, 0x00); //禁止写保护
Write1302 (WRITE_SECOND, 0x45); //秒位初始化
Write1302 (WRITE_MINUTE, 0x24); //分钟初始化
Write1302 (WRITE_HOUR, 0x09); //小时初始化
Write1302 (WRITE_DAY, 0x13); //日初始化
Write1302 (WRITE_MONTH, 0x04); //月初始化
Write1302 (WRITE_PROTECT, 0x80); //允许写保护
}
void delay_ms( unsigned int time)//延时 1 毫秒程序, n 是形式参数
{
    unsigned int i, j;
    for(i=time;i>0;i--)//i 不断减 1, 一直到 i>0 条件不成立为止
        for(j=112;j>0;j--)//j 不断减 1, 一直到 j>0 条件不成立为止
            {;}
}
void adopt_data(uchar n, uchar *show_data)//转换为 LCD 显示的数据
{
    *show_data=n>>4;//十位
    *(show_data+1)=n&0x0f;//个位
}
void main(void)
{
    uchar chinese[4][8]={
        0x0f, 0x09, 0x0f, 0x09, 0x0f, 0x09, 0x13, 0x00, //月
        0x1f, 0x11, 0x11, 0x1f, 0x11, 0x11, 0x1f, 0x00, //日
        0x01, 0x1d, 0x17, 0x1d, 0x17, 0x1d, 0x03, 0x01, //时
        0x04, 0x0a, 0x11, 0x0e, 0x02, 0x0a, 0x16, 0x00, //分
    };
    uchar munber[]={0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39}; //LCD 字符数码
    0~9
    uchar show_dat[2], time[5]; //时间的存放为月, 日, 时, 分, 秒
    uchar i, j;
    Initial(); //初始化 DS1302
    init_lcd(); //初始化 LCD
    wr_dictate(0x40); //对 CGRAM 进行操作
    for(j=0; j<4; j++)
    {
        for(i=0; i<8; i++)
        {
            wr_data(chinese[j][i]); //利用循环语句把点阵字符写进 CGRAM 中
        }
    }
    display_lcd(0x82, 0x00); //显示字符“月”
    display_lcd(0x85, 0x01); //显示字符“日”
    display_lcd(0x88, 0x02); //显示字符“时”
    display_lcd(0x8b, 0x03); //显示字符“分”
    while(1)
    {
        get_time(time); //获取 DS1302 的时间
        for(i=0, j=0; i<5; i++)
        {

```

```
adopt_data(time[i], show_dat); //转换为 LCD 显示的数据
display_lcd(0x80+j, munber[show_dat[0]]); //显示在液晶屏的相应位置
display_lcd(0x80+j+1, munber[show_dat[1]]);
j=j+3;
}
led=~led; //闪动 LED
delay_ms(1000); //大约 1 秒钟读一次
}
}
```

实验步骤：

1. 打开光盘第 5 章/ ds1302 / ds1302.uv2 工程文件，对程序进行编译、链接、调试产生 ds1302.hex 烧写文件。
2. 将实验板的 J4 短接到 LCDP 的一边，J7 短接到 LCDE 的一边。
3. 将烧写文件烧写到 AT89S52 单片机中去，正确安装到实验板上，接上电源。
4. 此时会见到液晶屏显示：4 月、13 日、9 时、24 分、45 秒，而且不断地走时。

实验效果如（实图 10）



（实图 10）

第六节 DS18B20 数字温度计

1. DS18B20 基本知识

DS18B20 数字温度计是 DALLAS 公司生产的 1-Wire，即单总线器件。具有线路简单，体积小，因此用它来组成一个测温系统，具有微型化，低功耗，线路简单的特点。每一个 DS18B20 器件中有一个唯一的序列号，因此同一条单线总线上可以挂接多个相同器件，十分方便。以下是 DS18B20 的产品特性。

- （1）、只要求一个端口即可实现通信。
- （2）、在 DS18B20 中的每个器件上都有独一无二的序列号。
- （3）、实际应用中不需要外部任何元器件即可实现测温。

- (4)、测量温度范围在 -55°C 到 $+125^{\circ}\text{C}$ 之间。
- (5)、数字温度计的分辨率用户可以从9位~12位选择。
- (6)、用户定义的、非易失性的温度报警设置，用户可自行设定报警的上下限温度。
- (7)、工作电压为3~5.5V。

如图 5-38 是器件的引脚功能。图 5-39 为器件的实用电路。

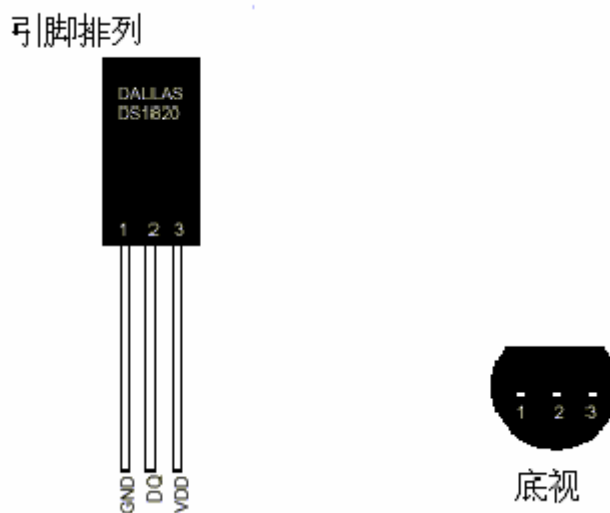


图 5-38

表 5-14

序号	名称	引脚功能描述
1	GND	电源地
2	DQ	数据输入/输出引脚。开漏单总线接口引脚。
3	VDD	电源正极引脚。工作电压为 3~5.5V。

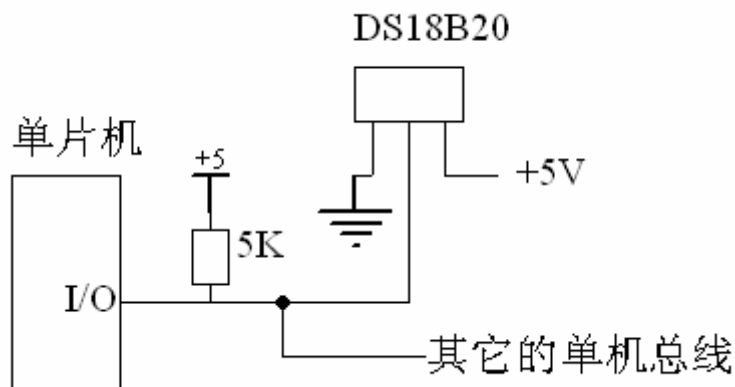


图 5-39 (DS18B20 的实用电路)

2. DS18B20 的使用方法

由于 DS18B20 采用的是 1-Wire 总线协议方式，而对 AT89S52 单片机来说，硬件上并不支持单总线协议，因此，我们必须采用软件的方法来模拟单总线的协议时序来完成对 DS18B20 芯片的访问。

由于 DS18B20 是在一根 I/O 线上读写数据，因此，对读写的数据位有着严格的时序要求。DS18B20 有严格的通信协议来保证各位数据传输的正确性和完整性。该协议定义了几种信号的时序：初始化时序、读时序、写时序。所有时序都是以主机作为主设备，单总线器件作为从设备。而每一次命令和数据的传输都是从主机主动启动写时序开始，如果要求 DS18B20 回送数据，在进行写命令后，主机需启动读时序完成数据接收。数据和命令的传输都是低位在先。

● DS18B20 的复位时序

在任何的情况下，要想与 DS18B20 通信必需以复位时序开始，在下图 5-40 就是一个初始化的复位时序。在图中我们可以清楚在见到，复位脉冲由主机开始生产大约 480us~960us，在等待大约 15us~60us，DS18B20 器件就会生产一个存在脉冲作为对单片机的应答，表明器件已经准备好接收或发送数据。

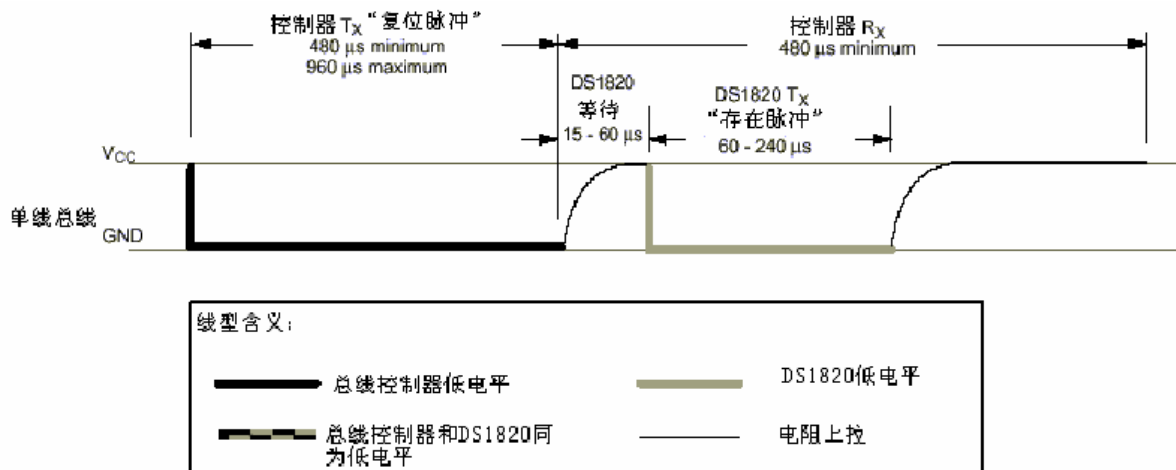


图 5-40（复位信号）

● DS18B20 的读时序

对于 DS18B20 的读时序分为读“0”时序和读“1”时序两个过程。当主机把数据线从高电平拉至低电平时，写时隙开始。数据线必须保持至少 1us；从 DS18B20 输出的数据在读时隙的下降沿出现后的 15us 内有效。因此，主机在读时隙开始必须停止把 I/O 引脚驱动为低电平 15us，以读取 I/O 脚状态。

对于 DS18B20 的读时隙是从主机把单总线拉低之后，在 15 us 之内就得释放单总线，以让 DS18B20 把数据传输到单总线上。DS18B20 在完成一个读时序过程，至少需要 60us 才能完成。如下图 4-41

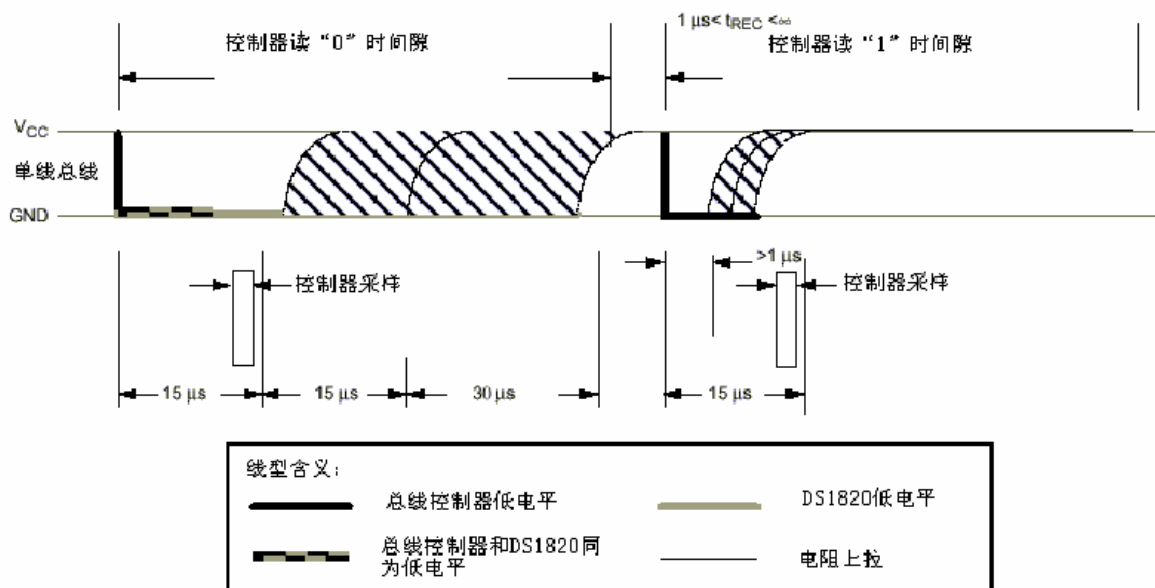


图 5-41 (读时序)

● DS18B20 的写时序

对于 DS18B20 的写时序仍然分为写“0”时序和写“1”时序两个过程。

对于 DS18B20 写“0”时序和写“1”时序的要求是不同的。当要写“0”时，单总线要被拉低至少 60us，保证 DS18B20 能够在 15us 到 45us 之间能够正确地采样 I/O 引脚上的“0”电平；当要写 1 时，单总线被拉低之后，在 15us 之内就得释放单总线。如下图 5-42 所示

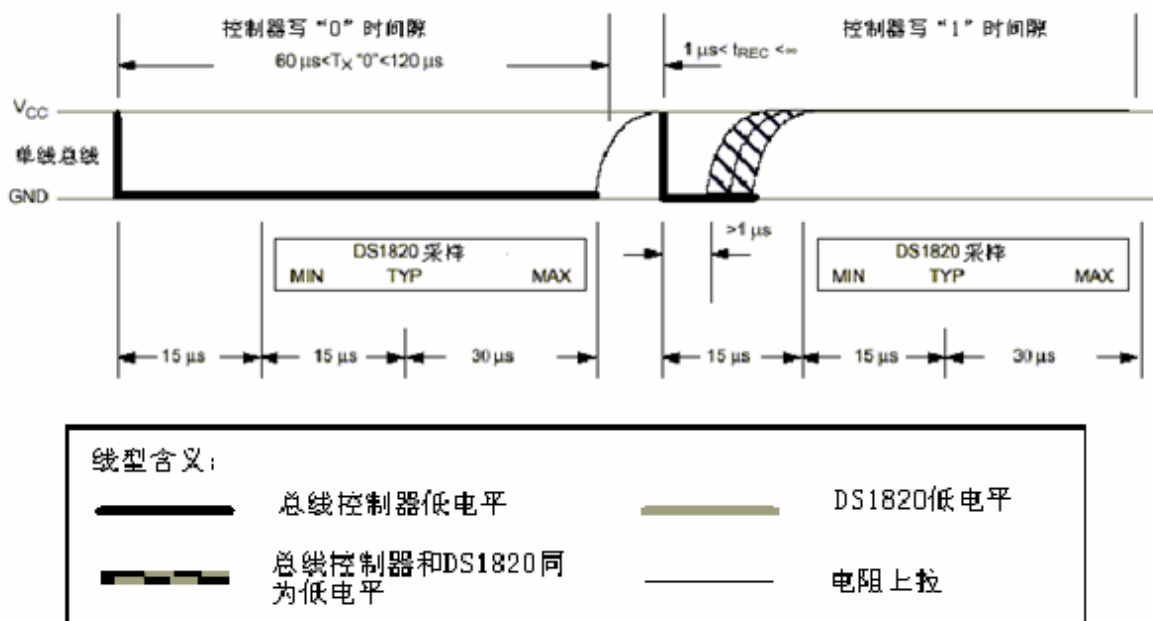


图 5-42 (写时序)

根据 DS18B20 的通讯协议，单片机(主机)控制 DS18B20 完成温度转换必须经过 3 个步骤：

- 每一次读写之前都要对 DS18B20 进行复位操作。

- 复位成功后发送一条 ROM 命令。
- 最后发送 RAM 命令。

这样才能对 DS18B20 进行预定的操作。复位要求主 CPU 将数据线下拉 500us，然后释放，当 DS18B20 收到信号后等待 16~60us 左右，后发出 60~240us 的存在低脉冲，主 CPU 收到此信号表示复位成功。在下面表 5-15 中是对 ROM 的操作命令，而表 5-16 是对寄存器的操作命令。

表 5-15（对 ROM 操作的命令码）

命令类型	约定命令	功能说明
Read Rom (读 ROM)	33H	此命令读取激光 ROM 中的 64 位，只能用于总线上单个 DS18B20 器件的情况，多个器件则会发生冲突。
Match Rom (匹配 ROM)	55H	此命令后跟 64 位 ROM 序列号，寻址多挂接总线上的对应 DS18B20。只有序列号完全匹配的 DS18B20 才能响应后面的内存操作命令，其它不匹配的将等待复位脉冲。此命令可用于挂接或者多挂接总线
Skip Rom (跳过 ROM)	CCH	此命令用于单挂接总线系统时，可以无须提供 64 位 ROM 序列
Search Rom (搜索 ROM)	FOH	主机调用此命令，通过一个排除法过程，可以识别出总线上所有器件的 ROM 序列号。
Alarm Search (警告搜索)	ECH	此命令流程和 Search Rom 命令相同，但是 DS18B20 只有在最近的一次温度测量时满足了警告触发条件，才会响应此命令。

表 5-16（对寄存器操作的命令码）

命令类型	约定命令	功能说明
Write Scratchpad (写暂存器)	4EH	此命令写暂存器中地址 2~地址 4 的 3 个字节（TH、TL 和配置寄存器）在发起复位脉冲之前，3 个字节都必顺要写。
Read Scratchpad (读暂存器)	BEH	此命令读取暂存器的内容，一直从第 0 个字节开始读到第 9 个字节。主机可以随时发起复位脉冲来停止此操作。
Copy Scratchpad (复制暂存器)	48H	此命令将暂存器中的内容复制到 E2RAM，以便将温度警告触发字节存入非易失内存。如果在此命令后主机产生读时隙，那么只要器件在进行复制就会输出 0，复制完成后再输出 1
Convert T (温度转换)	44H	此命令开始进行温度转换操作。如果在此命令后主机产生读时隙，只要器件在进行温度转换就会输出 0，转换完成后再输出 1。
Recall E2 (重调 E2 存储器)	B8H	将存储在 E2RAM 中的温度警告触发值和配置寄存器值重新拷贝到暂存器中。此重调操作在 DS18B20 加电时自动产生。
Read Power Suplly (读供电方式)	B4H	主机发起此命令后的每个读时隙，DS18B20 会发信号通知它的供电方式：0 为寄生电源方式；1 为外部供电方式。

在下图 5-43 中，分别列出 DS18B20 的寄存器分布。下面我们来分别讲解：

第 1 个字节的內容是温度的低 8 位。

第 2 个字节的內容是温度的高 8 位。

第 3 个字节的內容是高温限值。

第 4 个字节的內容是低温限值。

第 5 个字节是器件的配置寄存器。

第 6、7、8 为保留。

第 9 个字节的的内容是冗余校验字节。

寄存器内容	字节地址
温度值 (低位)	0
温度值 (高位)	1
高温限值 (TH)	2
低温限值 (TL)	3
配置寄存器	4
保留字节	5
保留字节	6
保留字节	7
CRC校验值	8

图 5-43 (寄存器的分布)

下面我们再针对第 5 个字节配置寄存器进行讨论。

表 5-17 (配置寄存器)

TM	R1	R0	1	1	1	1	1
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0

上表 5-17 已经列出了配置寄存器的位分配。其中位 7 为测试模式位，用于选择 DS18B20 是工作在测试模式或工作模式，出厂设置为“0”，一般情况下用户不要去改变它。而位 6、位 5 作为温度分辨率的设置，如图 5-18 所示。而位 4~位 0 固定为 1。

表 5-18 (温度值分辨率配置表)

R1	R0	分辨率	最大转换时间
0	0	9 位	93.75ms
0	1	10 位	187.50ms
1	0	11 位	375ms
1	1	12 位	750ms

3. DS18B20 的温度计算

下面我们再讨论关于 DS18B20 温度计算方法。温度转换后存放在寄存器的第 1、第 2 个字节内容中，DS18B20 的温度数据输出是以 16 位符号扩展的二进制补码格式表示。如下表 5-19

表 5-19 (温度数据格式)

位 15	位 14	位 13	位 12	位 11	位 10	位 9	位 8
符号位	符号位	符号位	符号位	符号位	2^6	2^5	2^4

位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

其中前 5 位是符号位，正温度则符号位为“0”，负温度则符号位为“1”。如果是正温度，只要将测到的数值乘以 0.0625 即可得到实际温度；若果是负温度，则把测到的数值按位取反后加 1 再乘以 0.0625 即可得到实际温度。大家要注意：DS18B20 在上电复位瞬间的温度值是+85° C。表 5-20 是温度的计算对照表。

表 5-20（温度值对照表）

温度	数据输出（二进制）	数据输出（十六进制）
+125° C	0000 0111 1101 0000	07D0H
+85° C	0000 0101 0101 0000	0550H
+25.0625° C	0000 0001 1001 0001	0191H
+10.125° C	0000 0000 1010 0000	00A2H
+0.5° C	0000 0000 0000 1000	0008H
0° C	0000 0000 0000 0000	0000H
-0.5° C	1111 1111 1111 1000	FFF8H
-10.125° C	1111 1111 0101 1110	FF5EH
-25.0625° C	1111 1110 0110 1111	FF6FH
-55° C	1111 1100 1001 0000	FC90H

动手实验（10）：

实验目的：学习 DS18B20 温度计的使用。

实验内容：利用 AT89S52 单片机模拟单总线协议读取 DS18B20 的温度值，并将温度数据在 LCD 液晶屏中显示出来。

```
#include <reg52.h> //包含头文件
#include "lcd.h" //包含 LCD 头文件
sbit DQ=P2^1; //18B20 温度数据引脚
sbit led=P1^0; //LED 灯
#define uchar unsigned char
#define uint unsigned int
uchar number[]={0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39}; //数字代码 0~9
uint temperature; //存放当前温度变量
uchar ind, ten, hun, decimal; //当前温度的百位，十位，个位，小数部分
void delay(uint n) //1ms
{
    uint i, j;
    for(i=n; i>0; i--)
        for(j=125; j>0; j--)
            {;}
}
void delay_18B20(uchar n) //DS18B20 转用延时函数
```

```
{
    while(n--);
}
void six_to_ten(void)//十六进制转换为十进制
{
    uchar kilo, ind, ten, hun;//千, 百, 十, 个
    hun=temperature%10;
    temperature=temperature/10;
    ten=temperature%10;
    temperature=temperature/10;
    ind=temperature%10;
    temperature=temperature/10;
    kilo=temperature%10;
    temperature=kilo*1000+ind*100+ten*10+hun;//温度已经转换为一个十进制数
}
void LCD_data(void)//将温度数据从 LCD 中显示出来
{
/*下面的 if 语句只要是判断是否需要显示百位,
假如百位是非 0 值则显示, 否则不用显示*/
    ind=temperature/1000;
    if(ind)
    { //寻找要显示的百位, 十位, 个位, 小数位
        temperature=temperature%1000;
        ten=temperature/100;
        temperature=temperature%100;
        hun=temperature/10;
        decimal=temperature%10;
        display_lcd(0xc1, number[ind]);//显示温度的百位
        display_lcd(0xc2, number[ten]);//显示温度的十位
        display_lcd(0xc3, number[hun]);//显示温度的个位
        display_lcd(0xc4, 0x2e);//显示小数点"."字符
        display_lcd(0xc5, number[decimal]);//显示小数部分
        display_lcd(0xc6, 0xdf);//显示一个"."
        display_lcd(0xc7, 0x43);//显示字符"C"
    }
    else
    { //寻找要显示的十位, 个位, 小数位
        ten=temperature/100;
        temperature=temperature%100;
        hun=temperature/10;
        decimal=temperature%10;
        display_lcd(0xc7, 0x20); //填入一个空字符
        display_lcd(0xc1, number[ten]); //显示温度的十位
        display_lcd(0xc2, number[hun]); //显示温度的个位
    }
}
```

```
        display_lcd(0xc3, 0x2e);           //显示小数点"."字符
        display_lcd(0xc4, number[decimal]); //显示小数部分
        display_lcd(0xc5, 0xdf);          //显示一个"."
        display_lcd(0xc6, 0x43);          //显示字符"C"
    }
}

void temperature_account(void)//温度处理函数
{
    if((temperature&0xf800)==0xf800)//负温度
    {
        temperature&=0x7ff;
        temperature=~temperature;//将数值取反
        display_lcd(0xc0, 0x2d); //在液晶中显示一个“-”字符
        six_to_ten();           //十六进制转换为十进制
        temperature=((temperature+1)*0.0625)*10;
    }
    else //正温度
    {
        display_lcd(0xc0, 0x2b); //在液晶中显示一个“+”字符
        six_to_ten();           //十六进制转换为十进制
        temperature=(temperature*0.0625)*10;
    }
    LCD_data();
}

void init_ds18b20(void)
{
    uchar x=0;
    DQ = 1;           //DQ 复位
    delay_18B20(8); //稍做延时
    DQ = 0;           //单片机将 DQ 拉低
    delay_18B20(40); //精确延时 大于 480us
    DQ = 1;           //拉高总线
    delay_18B20(7);
    x=DQ;           //稍做延时后 如果 x=0 则初始化成功 x=1 则初始化失败
    delay_18B20(10);
}

uchar read_18b20(void)//读取 ds18b20 的一个字节
{
    uchar loop, dat = 0;
    for (loop=8; loop>0; loop--)
    {
        DQ = 0; // 给脉冲信号
        dat>>=1;//移位,准备存放下一次数据
        DQ = 1; // 给脉冲信号
    }
}
```

```
        if(DQ)//读取数据
        dat|=0x80;//读取到的数据为 1
        delay_18B20(4);
    }
    return  dat;//返来读取到的数据
}
void write_18b20(uchar dat)//向 ds18b20 器件写一个字节
{
    uchar loop;
    for (loop=8; loop>0; loop--)
    {
        DQ = 0;
        DQ = dat&0x01;//向器件写一位数
        delay_18B20(5);
        DQ = 1;
        dat>>=1;//移位准备写下一个数据
    }
}
void Read_temperature(void)//读取当前温度函数
{
    uchar temperature_l=0;//存放当前温度的低位
    uchar temperature_h=0;//存放当前温度的高位

    init_ds18b20();          //在对器件操作之前先初始化
    write_18b20(0xCC);      // 跳过读序列号的操作
    write_18b20(0x44);      // 启动温度转换

    delay_18B20(100);       // 在读取转换温度之前应作适当的延时

    init_ds18b20();//初始化
    write_18b20(0xCC);      //跳过读序列号的操作
    write_18b20(0xBE);     //读取温度寄存器等（共可读 9 个寄存器） 前两个就是温度

    delay_18B20(50);

    temperature_l=read_18b20();          //读取当前温度值低位
    temperature_h=read_18b20();          //读取当前温度值高位
    /*将低位与高位组合为一个数方便处理*/
    temperature=temperature_h;
    temperature<<=8;
    temperature|=temperature_l;
    temperature_account();//调用温度处理函数
}
void main(void)
```



```
{  
  
    init_lcd();//初始化 LCD  
    while(1)  
    {  
        led=~led;//每 800ms 闪动 LED  
        Read_temperature();//读取当前温度  
        delay(800);  
    }  
}
```

实验步骤:

1. 打开光盘第 5 章/ DS18B20 / DS18B20.uv2 工程文件，对程序进行编译、链接、调试产生 DS18B20.hex 烧写文件。
2. 将实验板的 J4 短接到 LCDP 的一边，J7 短接到 LCDE 的一边。
3. 将 DS18B20 芯片按照实验板图形方向插到 U7 位置当中。
4. 将烧写文件烧写到 AT89S52 单片机中去，正确安装到实验板上，接上电源。
5. 此时 LCD 液晶屏就会显示当前温度。如果想测试 DS18B20 是否工作正常，可以用手触摸它，当触摸的同时 LCD 显示温度就会发生变化，从此证明温度检测系统是正常运行的。实验效果如（实图 11）



（实图 11）

第七节 综合实战——项目开发

利用实验板综合前面所学到的知识，开发一个项目，具体要实现以下的功能，硬件如图 5-43 所示。

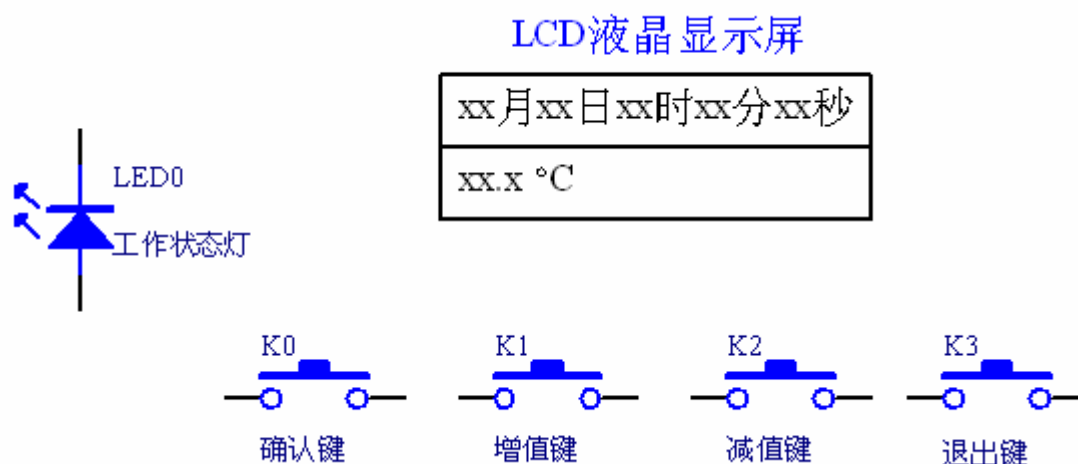
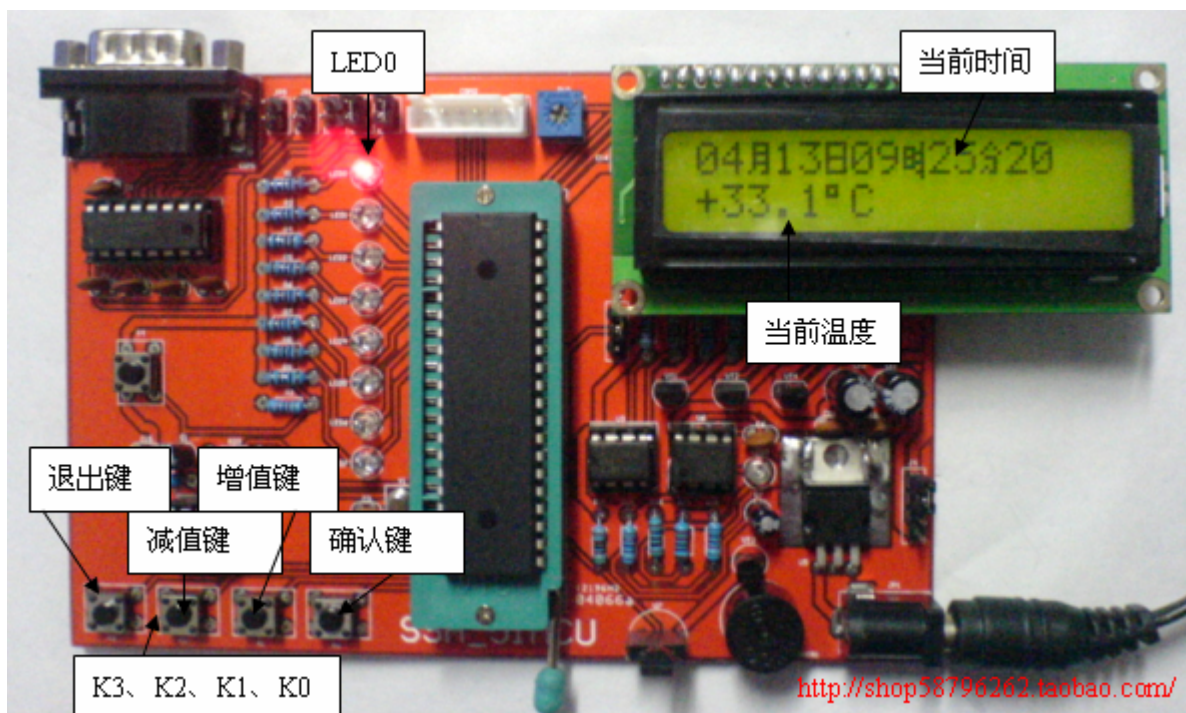


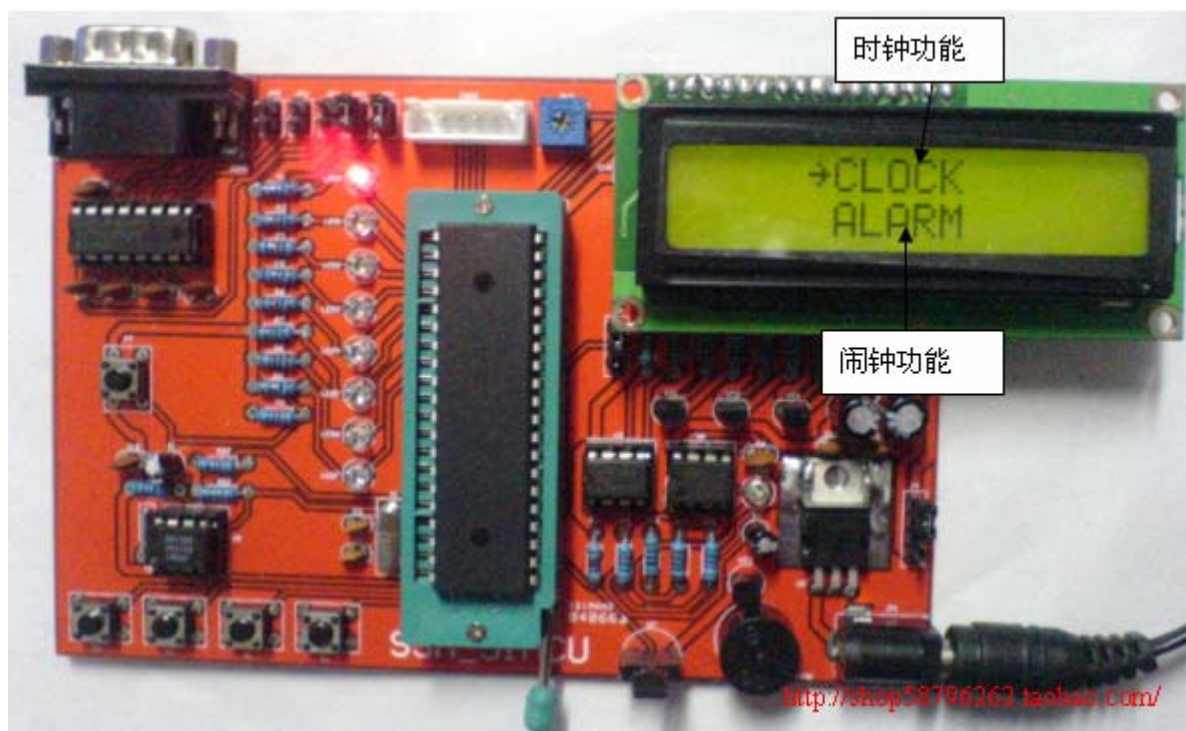
图 5-43（项目的整体硬件功能图）

其软件要实现的功能为：

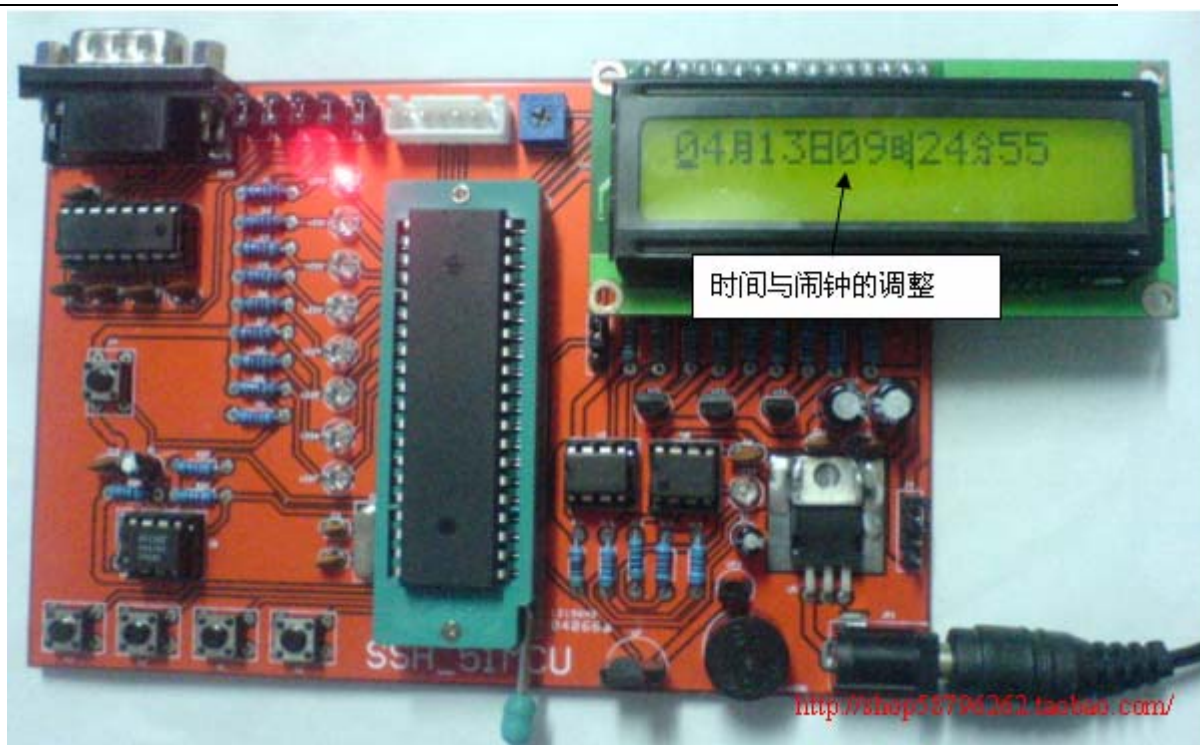
- 第（1）：LED0 是一枚工作状态灯。当处在正常工作状态下，此灯每一秒钟闪动一次。当进入功能设置的情况下，此灯长期点灯，表示正在进行功能的设置操作。
- 第（2）：LCD 的第一行显示日期和时间，其格式为：xx 月 xx 日 xx 时 xx 分 xx 秒。正常情况下在不停在走时。
LCD 的第二行显示当前温度。
- 第（3）：上面的 4 个按键分别定义为：确认键、增值键、减值键、退出键。
确认键：要对功能进行设定，总是以确认键开始。而在设置过程中，确认键也是对当前数据的确认。
增值键：用于对功能的选择和数据的增值调整。
减值键：用于对功能的选择和数据的减值调整。
退出键：当进入了功能设置状态，按退出键退出设置功能。
- 第（4）：利用按键可以对时间、闹钟进行设置。而且当闹钟成功设置后，在没有到达闹钟所设定的时间前即使是系统断了电，在下次系统上电正常运行情况下闹钟仍然有效，直至闹钟成功执行为止。
实验效果如下（实图 12. 13. 14. 15）



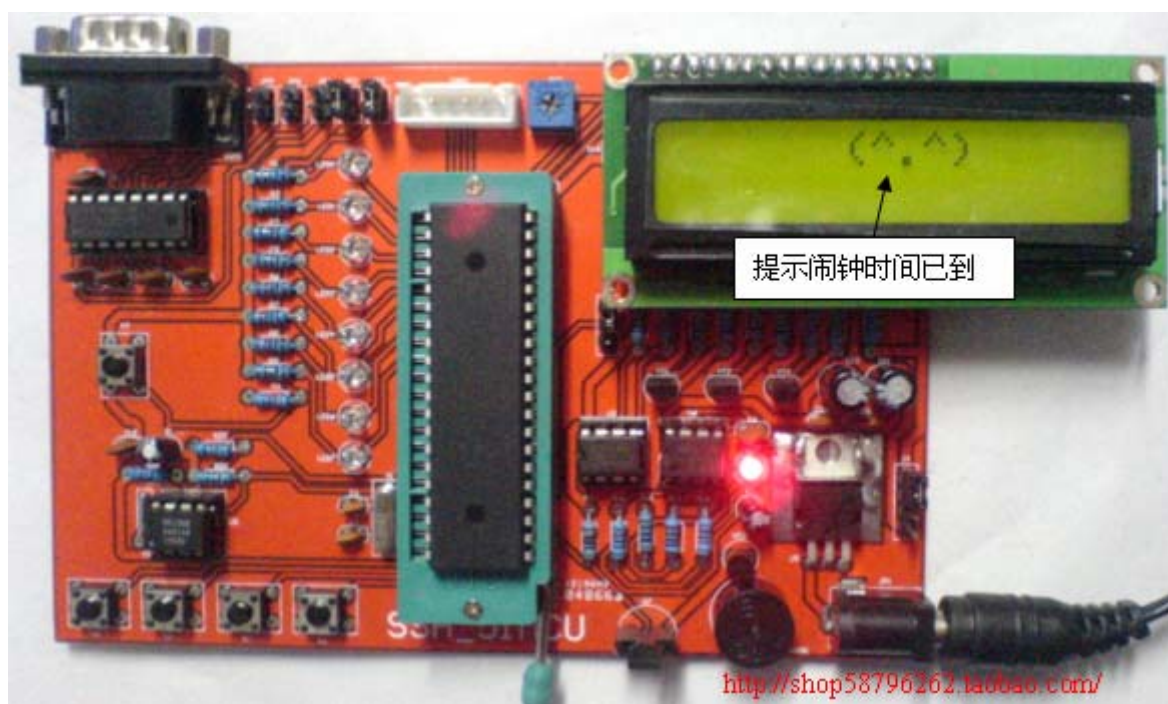
(实图 12)



(实图 13)



(实图 14)



(实图 15)

此项目与我们前面第一章：单片机初步，所实现的功能是完成相同的。

(程序附光盘的第五章/colligate_文件夹中)