

★校级精品课程

课程编号：102013

单片机原理与应用 实验指导书



石东新 温淑鸿 编写

2010.03.01 修订版

中国传媒大学·信息工程学院

2010

目 录

前 言.....	I
实 验 须 知.....	II
第一部分 基于 KEILC 的单片机编程实验.....	1
一、KEILC 介绍.....	1
二、KEILC 使用入门.....	3
实验 1-1 MCS-51 汇编语言指令系统（一）.....	14
实验 1-2 MCS-51 汇编语言指令系统（二）.....	19
实验 1-3 MCS-51 汇编语言指令系统（三）.....	23
实验 1-4 MCS-51 汇编语言程序设计（一）.....	26
实验 1-5 MCS-51 汇编语言程序设计（二）.....	31
实验 1-6 MCS-51 汇编语言程序设计（三）.....	33
实验 1-7 MCS-51 汇编语言程序设计（四）.....	35
实验 1-8 MCS-51 汇编语言程序设计（五）.....	38
第二部分 单片机系统的 PROTEUS 设计与仿真.....	40
一、PROTEUS 介绍.....	40
二、PROTEUS ISIS 入门.....	42
实验 2-1 发光二极管流水灯.....	57
实验 2-2 按键控制发光二极管发光.....	61
实验 2-3 定时器应用（一）.....	64
实验 2-4 定时器应用（二）.....	69
实验 2-5 定时器与中断的综合应用.....	72
实验 2-6 外中断应用.....	76
实验 2-7 串口通信.....	79
实验 2-8 外部数据存储扩展.....	88
实验 2-9 数码管显示实验.....	94
实验 2-10 键盘扫描.....	99
综合实验—电子时钟温度计.....	103
第三部分 PROTEUS 绘制 PCB 图.....	125
一、绘制电路原理图及调试仿真.....	126
二、加载网络表及元件封装.....	126
三、规划电路板并设置相关参数.....	127
四、元件布局及调整.....	128
五、布线及调整.....	130
六、输出并制作 PCB.....	132
第四部分 51 单片机的 C 语言编程.....	135
一、数据类型.....	135
二、存储类型.....	135
三、指针.....	137

四、中断	137
五、例程说明.....	138
附录 A：51 系列单片机指令系统.....	140
一、数据传送类指令.....	140
二、算术运算类指令.....	143
三、逻辑运算类指令.....	144
四、控制转移类指令.....	146
五、位操作类指令.....	148
附录 B：修订记录：	151

前 言

由于单片机具有高可靠性、超小型、低价格、容易产品化等特点，在仪器仪表智能化、实时工业控制、实时数据采集、智能终端、通信设备、导航系统、家用电器等控制应用领域，具有十分广泛的用途。目前国内单片机应用中，MCS-51 系列单片机仍然是一种主流单片机。为了配合《单片机原理与应用》课程的教学，我们结合 Proteus 仿真软件和自主的单片机开发平台，为学习 MCS-51 单片机的同学们编写了这本实验指导书（本实验指导书特为中国传媒大学编写）。

《单片机原理与应用》是一门实践性很强的课程，提高教学质量和实践开发能力的一个重要环节是上机实习和训练。无论是学习汇编语言程序设计，还是学习接口电路和外设与计算机的连接，或者软硬兼施地研制单片机应用系统，不通过勤动手是不能获得预期效果的。本实验指导书提供了近 20 个实验的指导性材料，有些实验有一定难度，可以选做。指导老师可以根据课时的安排和教学要求进行取舍。

本实验指导书由石东新、温淑鸿编写，并得到柴剑平、李真、周德阳、靳晓芳、金立标等老师的帮助。尤其是 05 级广电工孙正同学编写了第三部分“Proteus 绘制 PCB 图”的大部分内容，在这里表示感谢。

单片机的实验教学要向实际开发靠近，要向规范化的管理靠近，这都要求我们给出适应于教学的实验指导书。但是由于时间紧迫，加上编者学识有限，如有不妥之处，欢迎读者批评指正。

编 者

2008.12

实 验 须 知

1. 完成实验前要求的准备工作：认真对待老师布置的预习任务，阅读教科书和本实验指导书的相关章节，了解实验内容、目的、要求，按教师要求编写好实验中要求调试完成的程序；。

2. 硬件实验时，各种电源的电压和极性不能接错，严禁带电接线和接插元器件。通电前须经过严格检查确认方能通电。

3. 不准随意拨弄各种与实验无关的旋钮和开关，凡与本次实验无关的任何设备都禁止动用和摸弄，注意安全。

4. 严禁用手触摸实验系统印制电路板和元器件的引脚，防止静电击穿芯片。

5. 实验中若损坏仪器或元器件，应及时向指导教师报告，听候处理。

6. 在实验室内保持安静和卫生，不得在实验时内吃东西，随意走动和喧哗，集中精力完成实验。

7. 实验完成后，关掉电源，及时整理实验台桌面，保持环境整洁。

8. 按规定认真完成实验报告，对实验中出现的现象进行分析，在规定的时间内上交实验报告。

9. 凡实验或实验报告未能按规定完成者，不能参加本课程的考试。

第一部分 基于 KeilC 的单片机编程实验

一、KeilC 介绍

Keil C51 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统。Keil C51 软件提供丰富的库函数和功能强大的集成开发调试工具，Windows 标准界面。

Keil C51 μ Vision2 集成开发环境由 Keil Software, Inc/Keil Elektronik GmbH 开发，是基于 80C51 内核的微处理器软件开发平台，内嵌多种符合当前工业标准的开发工具，可以完成从工程建立到管理、编译、链接、目标代码的生成、软件仿真、硬件仿真等完整的开发流程尤其是 C 编译工具在产生代码的准确性和效率方面达到了较高的水平，而且可以附加灵活的控制选项，在开发大型项目时非常理想。Keil C51 集成开发环境的主要功能有以下几点：

- μ Vision2 for Windows：是一个集成开发环境，它将项目管理、源代码编辑和程序调试等组合在一个功能强大的环境中；
- C51 国际标准化 C 交叉编译器：从 C 源代码产生可重定位的目标模块；
- A51 宏汇编器：从 80C51 汇编源代码产生可重定位的目标模块；
- BL51 链接器/定位器：组合由 C51 和 A51 产生的可重定位的目标模块，生成绝对目标模块；
- LIB51 库管理器：从目标模块生成连接器可以使用的库文件；
- OH51 目标文件至 HEX 格式的转换器，从绝对目标模块生成 Intel Hex 文件；
- RTX-51 实时操作系统：简化了复杂的实时应用软件项目的设计。

这个工具套件是为专业软件开发人员设计的，但任何层次的编程人员都可以使用，并获得 80C51 单片机的绝大部分应用。

Keil Software 提供了一流的 80C51 系列开发工具软件，下面描述每个套件及其内容：

1. PK51 专业开发套件。PK51 专业开发套件提供了所有工具，适合专业开发人员建立和调试 80C51 系列微控制器的复杂嵌入式应用程序。专业开发套件可针对 80C51 及其所有派生系列进行配置使用。

2. DK51 开发套件。DK51 开发套件是 PK51 的精简版，它不包括 RTX51 Tiny 实时操作系统。开发套件可针对 80C51 及其所有派生系列进行配置使用。

3. CA51 编译器套件。如果开发者只需要一个 C 编译器而不需要调试系统，则 CA51 编译器套件就是最好的选择。CA51 编译器套件只包含 μ Vision2 IDE 集成开发环境，CA51 不提供 μ Vision2 调试器的功能。这个套件包括了要建立嵌入式应用的所有工具软件，可针对 80C51 及其所有派生系列进行配置使用。

4. A51 汇编器套件。A51 汇编器套件包括一个汇编器和创建嵌入式应用所需要的所有工具。它可针对 80C51 及其所有派生系列进行配置使用。

5. RTX51 实时操作系统 (FR51)。RTX51 实时操作系统是 80C51 系列微控制器的一个实时内核。RTX51 Full 提供 RTX51 Tiny 的所以功能和一些扩展功能，并且包括 CAN 通信协议接口子程序。

Keil C51 软件是众多单片机应用开发的优秀软件之一，它集编辑，编译，仿真于一体，支持汇编,PLM 语言和 C 语言的程序设计，界面友好，易学易用。与汇编相比，C 语言在功能上、结构性、可读性、可维护性上有明显的优势，因而易学易用。用过汇编语言后再使用 C 来开发，体会更加深刻。

Keil C51 软件是目前各种系列单片机 C 语言编程的流行开发环境，KeilC 本身的特长是高效率 C 语言编译和便捷的调试环境，但是 KeilC 对汇编语言也同样的支持，因此本课程的实验编程全部基于 KeilC 环境，为今后过渡到 C 语言的开发打下一个良好的基础。

二、KeilC 使用入门

1、创建一个工程

1) 单击桌面快捷键



或者单击“开始”->“所有程序”->“Keil uVision2”，如图 1-1。

运行后，出现如图 1-2 的屏幕

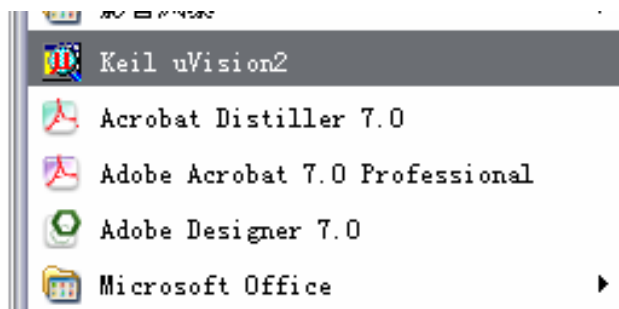


图 1-1



图 1-2

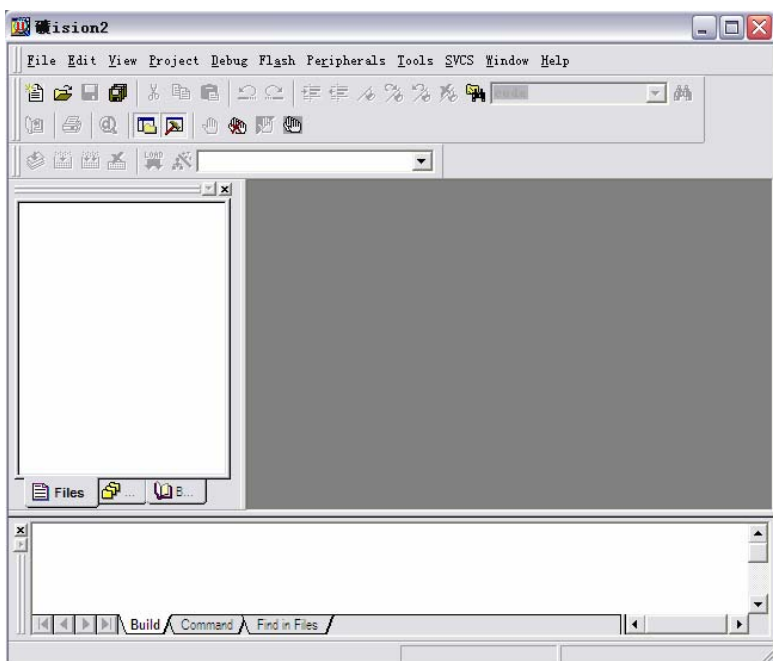


图 1-3

随后，程序运行的界面如图 1-3 所示。

2) 点击Project 菜单， 选择弹出的下拉式菜单中的New Project ， 如图1-4。接着弹出一个标准Windows 文件对话框， 如图1-5。

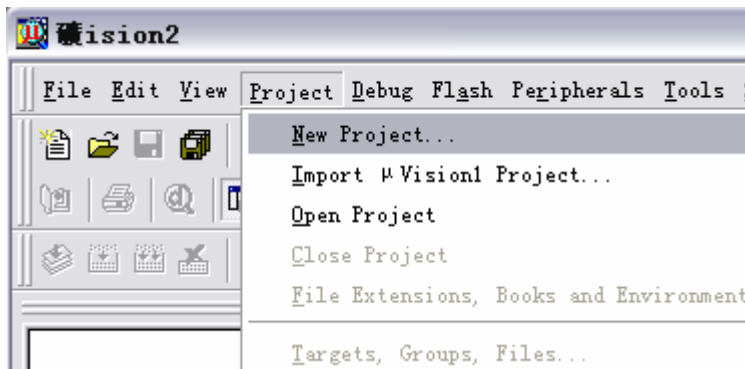


图 1-4

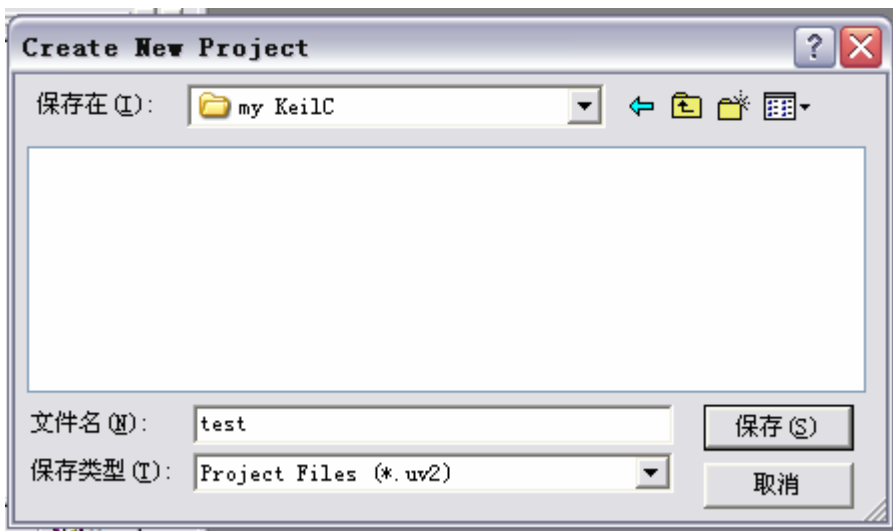


图 1-5

在“保存在”里选择一个合适的路径，或者创建一个合适文件夹，以方便将来编写的代码文件都保存在这里。图 1-5 创建了一个“my KeilC”的文件夹。然后在“文件名”中输入 C 程序项目名称，这里我们用“test”。“保存”后的文件扩展名为 uv2 ， 这是 KEIL uVision2 项目文件扩展名，以后我们可以直接点击此文件以打开先前做的项目。如图 1-5。

3) 接着，会弹出如图1-6所示的界面，选择所要的单片机，这里我们选择的是ATMEL 公司的AT89S52 。此时屏幕如图1-6 所示。图中右边有对AT89S52功能、特点简单的介绍。完成上面步骤后，我们就可以进行程序的编写了。

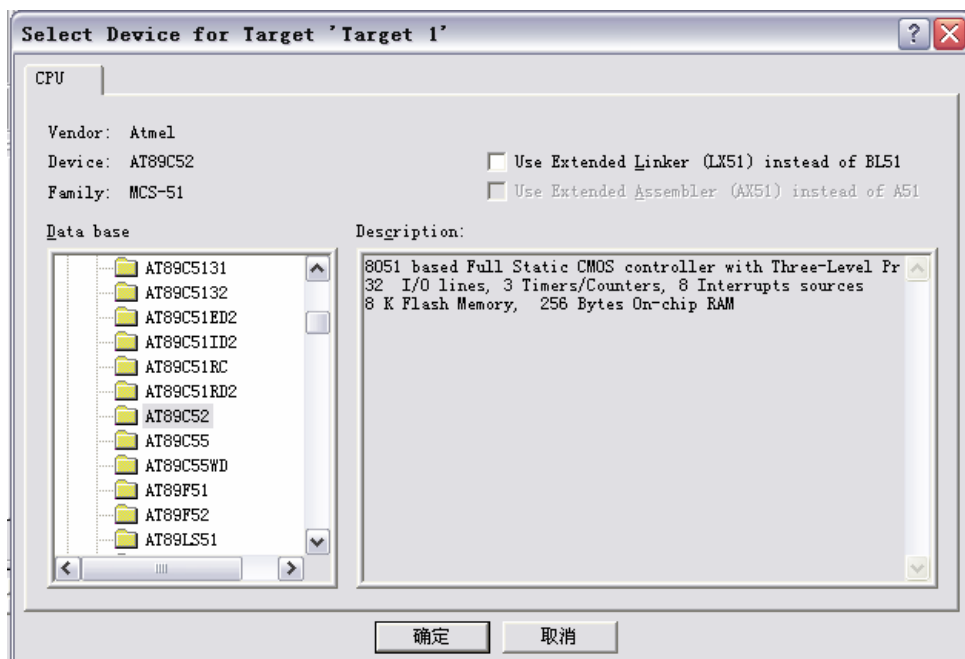


图 1-6



图 1-7

接着，完整 KeilC 环境就会出现，如图 1-8。

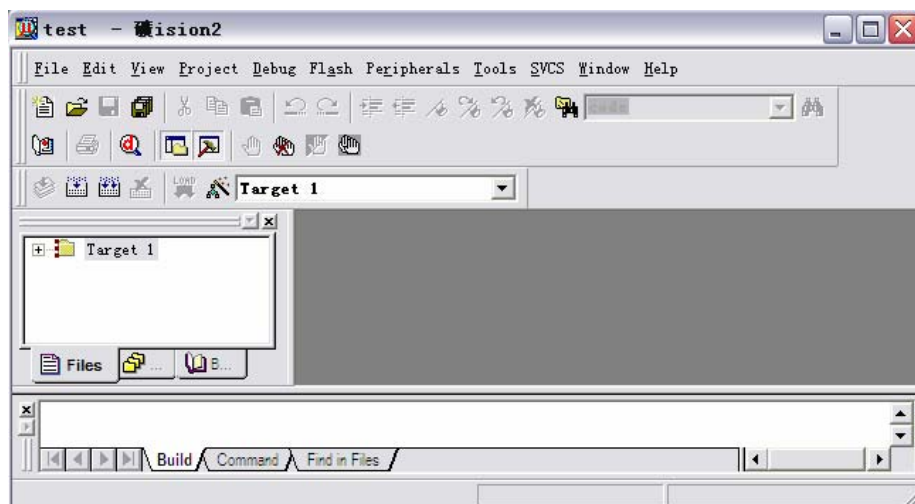



图 1-8

按“确定”，后，
会弹出如图 1-7
的对话框，选择
“否 (N)”。

下面我们继续进行。首先我们要在
项目中创建新的
程序文件或加入
旧程序文件。如果
没有现成的程序，
那么就要新建一
个程序文件。在

谢！

KEIL 中有一些程序的 Demo（范例程序），在这里我们以一个汇编程序为例介绍如何新建一个 ASM 程序以及如何添加到你的项目中。点击图 1-8 中工具栏中的  按钮（新建文件的快捷按钮）。在 1-8 中会出现一个新的文字编辑窗口，如图 1-9 所示这个操作也可以通过菜单 File New 或快捷键 Ctrl+N 来实现。

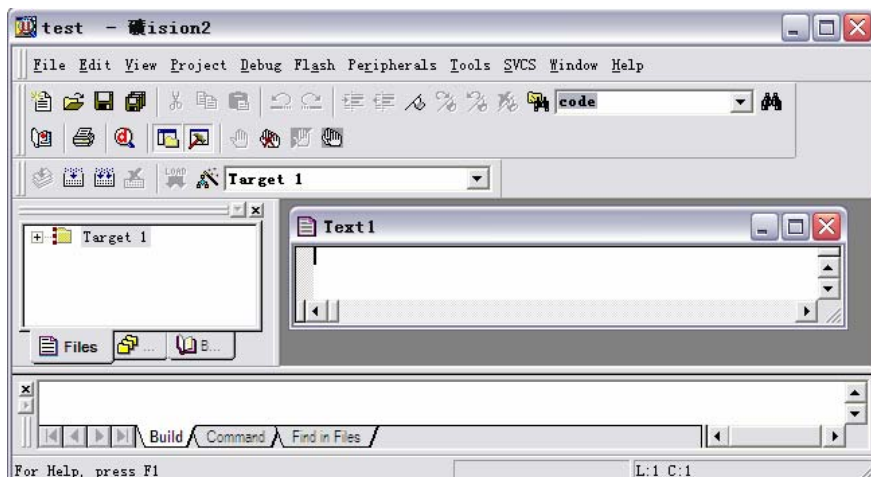



图 1-9

（4） 点击图1-9 中的 ，保存新建的程序，也可以用菜单File—Save 或快捷键 Ctrl+S 进行保存。因为是新文件所以保存时会弹出类似图1-5 的文件操作窗口，我们把第一个程序命名为test1.asm（手动添加后缀），保存在项目所在的目录中，这样KEIL 的asm语法检查会自动生效。如图 1-10 鼠标在屏幕左边的Source Group1 文件夹图标上右击弹出菜单，在这里可以在项目中增加减少文件等操作。我们选“Add File to Group ‘Source Group 1’” 弹出文件窗口1-11，在“文件类型（T）”选择如图所示的文件类型，则刚刚保存的文件 test1.asm就会出现,选中该文件,单击ADD 按钮，关闭文件窗，程序文件已加到项目

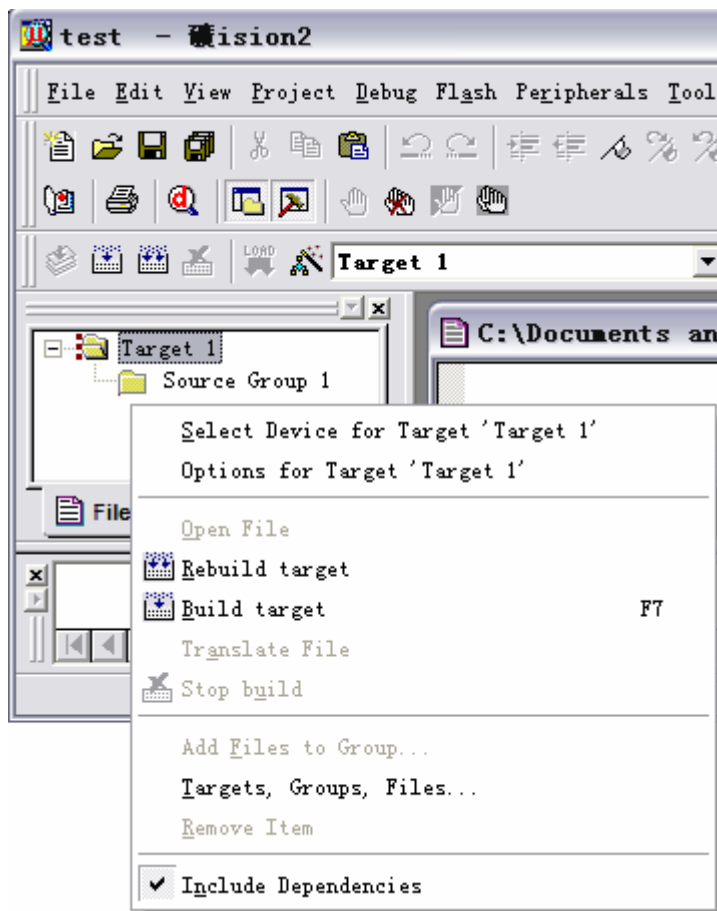


图 1-10 把文件加入到项目文件组中

中了。这时在Source Group1 文件夹图标左边出现了一个小+号，说明文件组中有了文件，点击它可以展开查看。

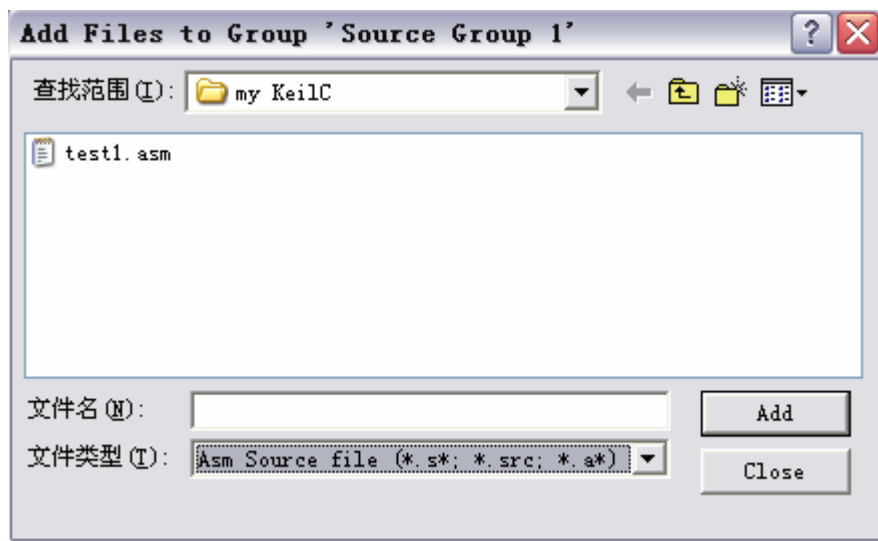


图 1-11

(5) 下面是一段程序，我们键入到test1.asm中：

```

;三字节无符号数加法（顺序结构）
ORG    0000H
LJMP   MAIN
ORG    0030H
MAIN:
MOV    R0, #50H      ;指向加数最低位
MOV    R1, #53H      ;指向另一加数最低位
MOV    A, @R0         ;取被加数
ADD    A, @R1         ;两数相加，带进位
MOV    @R0, A
INC    R0             ;修改地址
INC    R1
MOV    A, @R0         ;取被加数
ADDC   A, @R1         ;两数相加，带进位
MOV    @R0, A
INC    R0             ;修改地址
INC    R1
MOV    A, @R0         ;取被加数
ADDC   A, @R1         ;两数相加，带进位
MOV    @R0, A
MOV    00H, C
SJMP   $

```

END

这段代码实现了内部RAM区域存储的两个三字节数据进行加法的运算。我们先不管程序的语法和意思，先看看如何编译试运行。

(6) 我们先要根据需要进行一些配置。右键单击“Target 1”选项，弹出图1-12所示窗口，单击“Option for Targets ‘Target 1’”选项，弹出如图1-13对话框。这是一个十分重要的窗口，包括多个选项标签页，其中许多选项可以使用默认值，也可以根据自己需要进行调整。

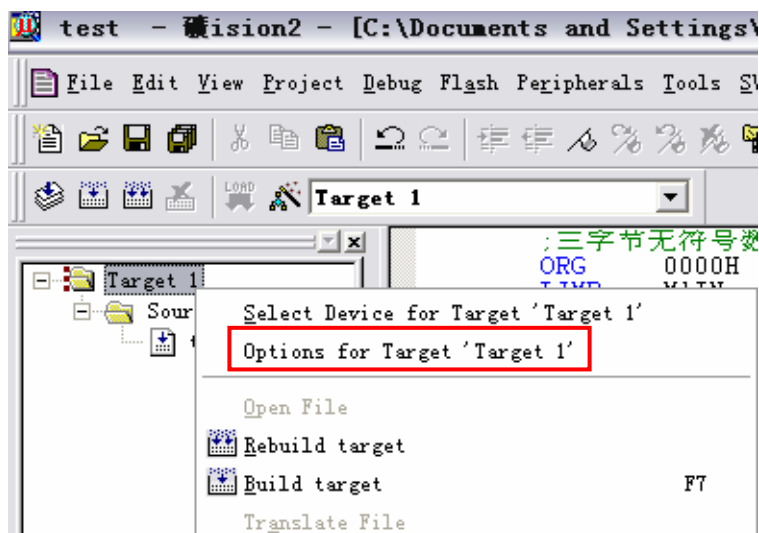


图 1-12

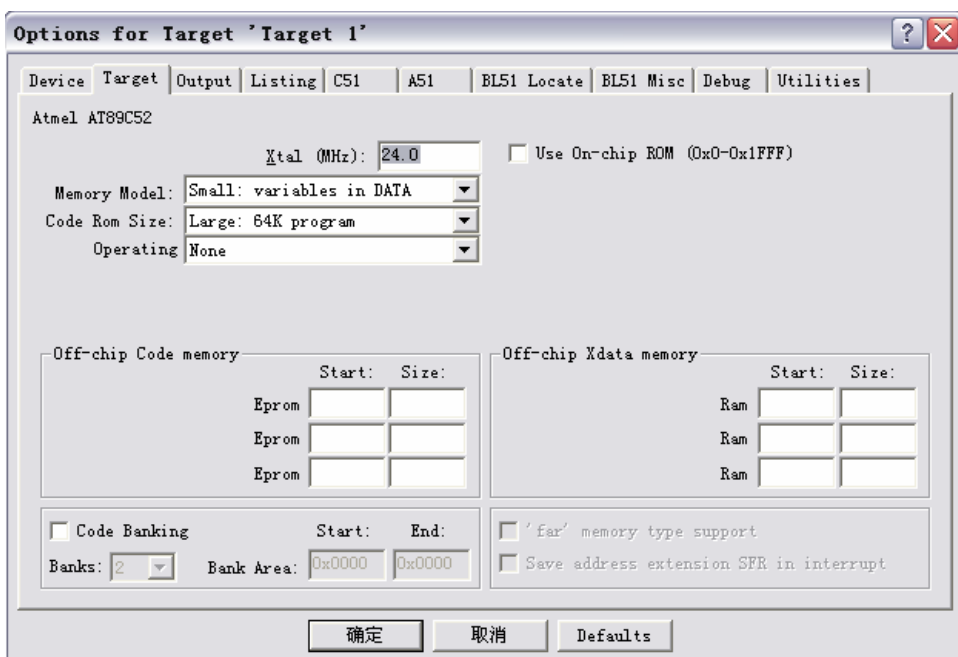


图 1-13

下面我们提供一组推荐配置方法，大家可以与下面各图一一对照进行配置。以下只对一些重要的选项进行了说明，其余选型的具体功能请大家参阅相关书籍。

Device 标签:

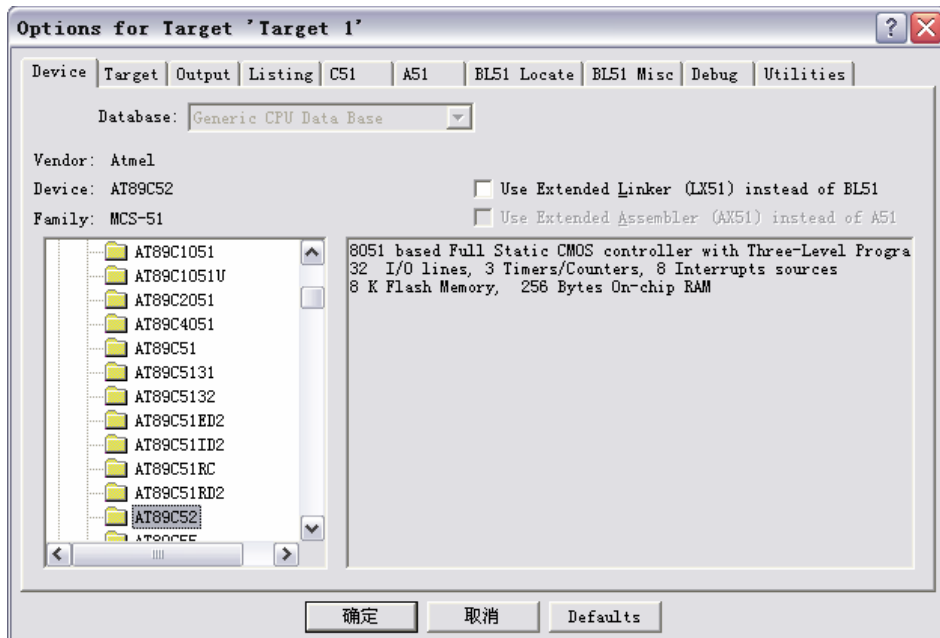


图 1-14

这一页我们在建立项目时曾经遇到过，在这里可以进行修改。

Target 标签:

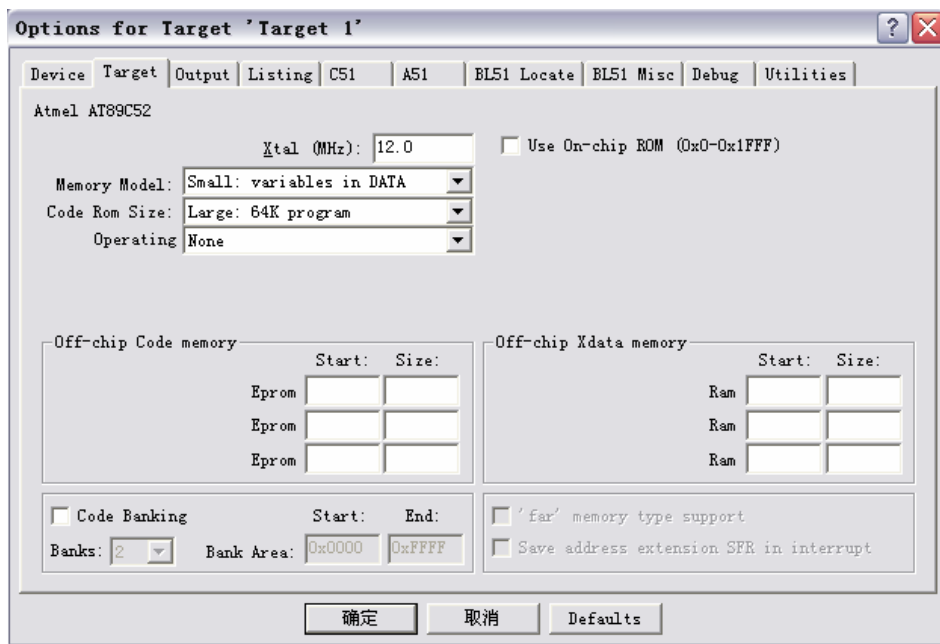


图 1-15

这里需要强调的是，晶振一般使用12MHz，因此这一页的Xtal（MHz）项要改为12.0（为了仿真与实际情况相吻合）。如果实际使用的是其他频率的晶振，这里要改为相应数字。

Output 标签:

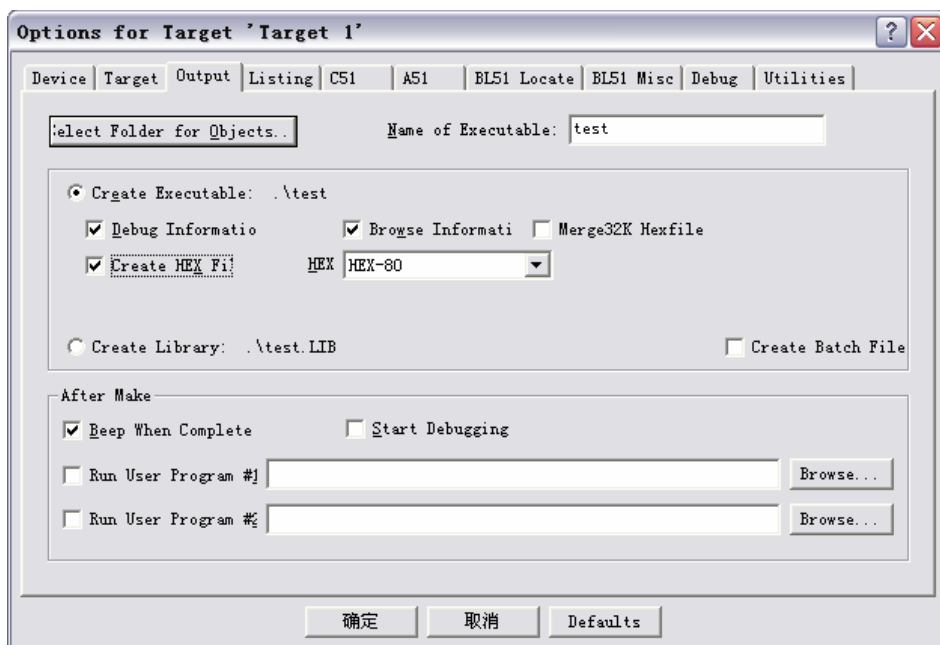


图 1-16

如果想把编译后的程序下载到单片机上进行验证，就要选中Create HEX File选项，这样系统在编译之后就会自动生成一个.hex文件（文件位置点击“Select Folder for Objects...”按钮选择），把这个文件下

载到单片机上，单片机就会按照程序进行相应操作。

Listing、C51、A51、BL51 Locate、BL51 Misc 标签全部使用默认选项。

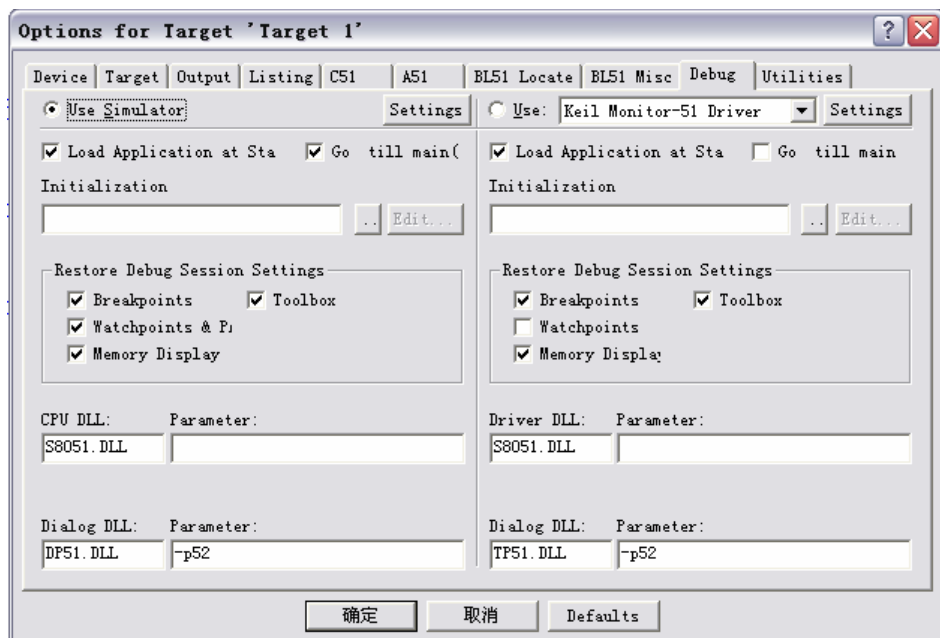


图 1-17













Debug 标签:

这一页也使用默认选项。可以看到这一页分左、右两部分，大家在使用软件仿真时要选中Use Simulator选项。

谢谢！

如果有硬件仿真器，则可以选中右边的选项来支持硬件仿真。考虑到硬件仿真器比较昂贵，因此大家一般是先进行软件仿真排除一般性错误，然后利用编程器把程序直接下载到单片机上进行调试。我们后面一部分的单片机硬件实验仍然使用软件仿真，但是使用目前比较先进的仿真软件Proteus，该软件对于KeilC来说，就相当于一个真实的硬件仿真。因此，在Proteus和KeilC联合使用时，KeilC仍然要在该页选中右边的选项来支持Proteus仿真，其具体使用方法，将在实验的第二部分中讲解。

2、编译与仿真：

(1) 设置完毕后，就可以编译调试程序了。如图2-1 所示，图中   都是编译按钮，不同是 是用于编译单个文件。 是编译当前项目，如果先前编译过一次之后文件没有做过编辑改动，这时再点击是不会再次重新编译的。 是重新编译，每点击一次均会再次编译链接一次，不管程序是否有改动。在 右边的是停止编译按钮，只有点击了   中的任一个，停止编译按钮才会生效。在图2-1下方的“Build”页中可以看到编译的错误信息和使用的系统资源情况等。 是开启\关闭调试模式的按钮，它也存在于菜单Debug--Start\Stop Debug Session，快捷键为Ctrl+F5。

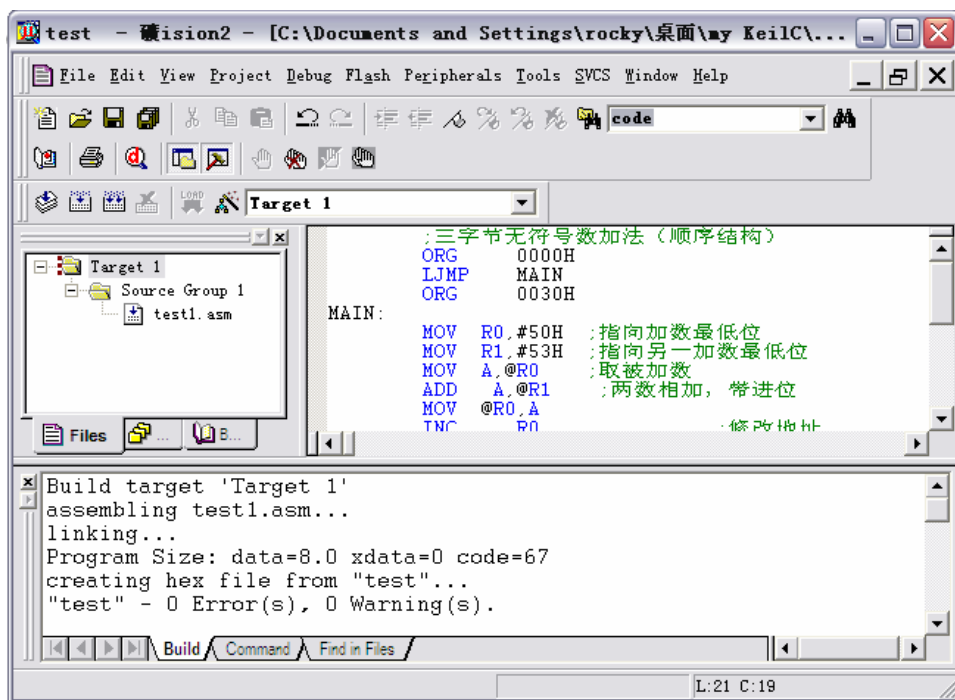













图 2-1

(2) 单击按钮后,就能进入调试模式,软件窗口样式大致如图2-2所示。图中为运行,当程序处于停止状态时才有效,为停止,程序处于运行状态时才有效。是复位,模拟芯片的复位,程序回到最开头处执行。按在汇编代码和机器代码之间(用C语言时,在C代码和汇编代码之间)进行切换。按钮实现程序进入到子程序中,如果不是子程序而是指令,则执行当前指令。按钮只运行一步,执行当前指令或者子程序的功能。按钮只有在进入子程序后才有效,单击该按钮,退出子程序的运行,到子程序的下一条指令处。按钮使程序运行到光标所指的位置。

最后我们要停止程序运行回到文件编辑模式中,只要先按停止按钮,再按按钮,我们才可以进行关闭KeilC软件等相关操作,否则KeilC软件无法退出关闭。

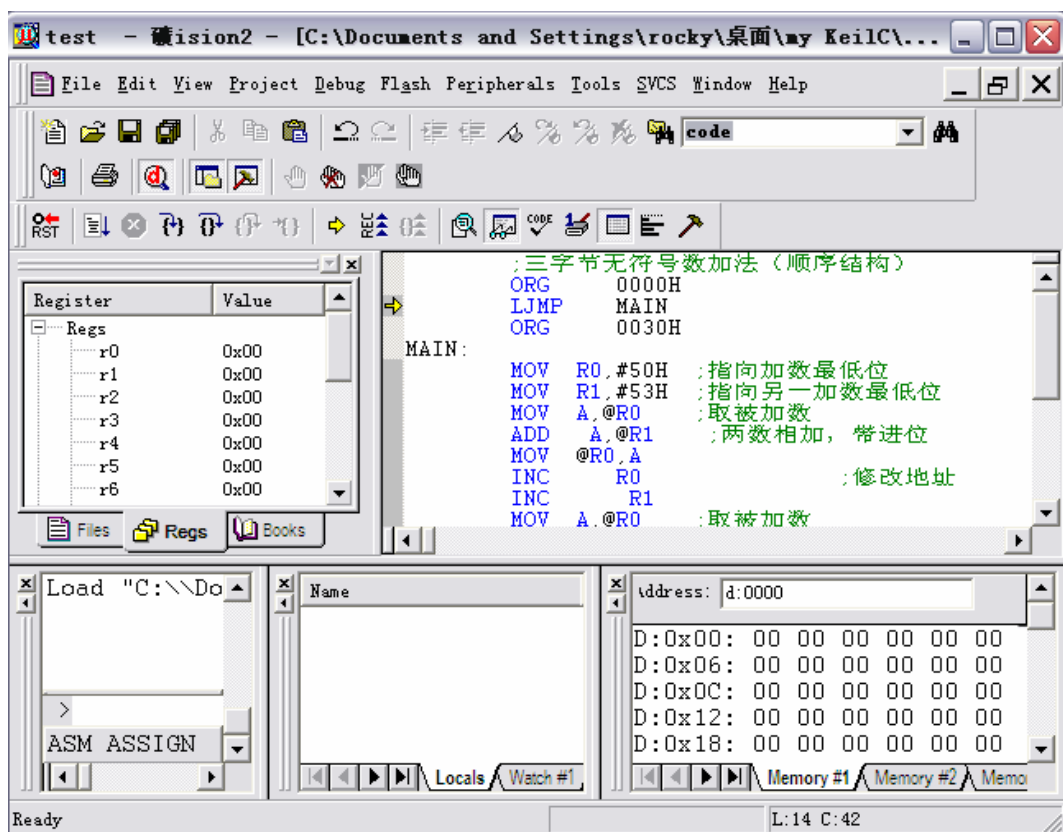


图2-2

(3) 在调试过程中,我们经常需要查看寄存器,内部数据存储器等的情况,KeilC提供了很好的查看窗口。在调试状态下,单击菜单栏里的“View”菜单,如图2-3所示。在非调试状态下,“View”菜单的下拉菜单里,很多选项是灰色禁用的。

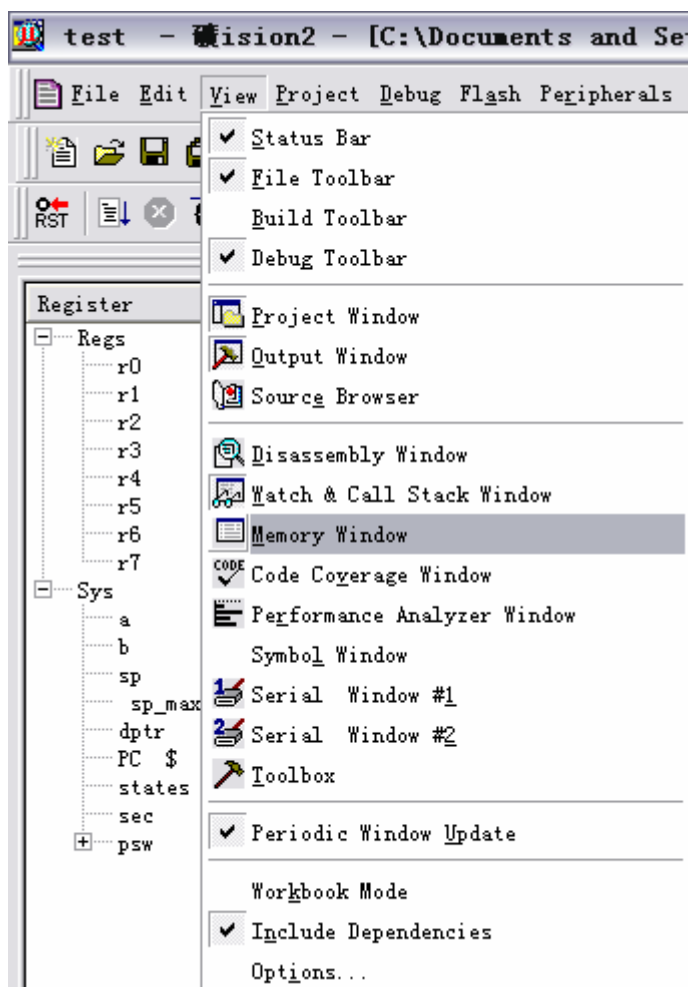










图2-3

这里的       按钮，在调试过程中经常起着非常重要的作用。单击  选项，可以弹出存储器查看窗口，我们可以在该窗口查看单片机所有内部存储器的资源。单击 ，我们可以查看关注的寄存器或存储器单元。其它选项的使用可以参考帮助，这里不再赘述。

实验 1-1 MCS-51 汇编语言指令系统（一）

【背景知识】

MCS-51 共有 7 种寻址方式：

1. 立即寻址：操作数就写在指令中，和操作码一起放在程序存储器中。把“#”号放在立即数前面，以表示该寻址方式为立即寻址，如 `MOV A, #20H`。

2. 寄存器寻址：操作数放在寄存器中，在指令中直接以寄存器的名来表示操作数地址。如 `MOV A, R0` 就属于寄存器寻址，即 `R0` 寄存器的内容送到累加器 `A` 中。

3. 直接寻址：操作数放在单片机的内部 RAM 某单元中，在指令中直接写出该单元的地址。如 `ADD A, 70H` 中的 `70H`。

4. 寄存器间接寻址：操作数放在 RAM 某个单元中，该单元的地址又放在寄存器 `R0` 或 `R1` 中。如果 RAM 的地址大于 256，则该地址存放在 16 位寄存器 `DPTR`（数据指针）中，此时在寄存器名前加@符号来表示这种间接寻址。如 `MOV A, @R0`。

5. 变址寻址：指定的变址寄存器的内容与指令中给出的偏移量相加，所得的结果作为操作数的地址。如 `MOVC A, @A+DPTR`

6. 相对寻址：由程序计数器中的基地址与指令中提供的偏移量相加，得到的为操作数的地址。如 `SJMP rel`

7. 位寻址：操作数是二进制中的某一位，其位地址出现在指令中。如 `SETB bit`

MCS51 的指令系统按功能分有：数据传送类、转移指令、算术运算类、逻辑运算类、和十进制指令及一些伪指令。

（1） 数据传送类指令（7 种助记符）

`MOV`（Move）：对内部数据寄存器 RAM 和特殊功能寄存器 SFR 的数据进行传送；

`MOVC`（Move Code）读取程序存储器数据表格的数据传送；

`MOVX`（Move External RAM）对外部 RAM 的数据传送；

`XCH`（Exchange）字节交换；

`XCHD`（Exchange low-order Digit）低半字节交换；

`PUSH`（Push onto Stack）入栈；

`POP`（Pop from Stack）出栈；

(2) 算术运算类指令（8种助记符）

ADD(Addition) 加法；

ADDC(Add with Carry) 带进位加法；

SUBB(Subtract with Borrow) 带借位减法；

DA(Decimal Adjust) 十进制调整；

INC(Increment) 加1；

DEC(Decrement) 减1；

MUL(Multiplication、Multiply) 乘法；

DIV(Division、Divide) 除法；

【实验内容】

对数据传送类指令和算术运算类指令每一条指令，进行执行，并注意观察相关寄存器如源操作数，目的操作数以及状态寄存器PSW的内容。

【实验目的】

1. 掌握数据传送类指令。

掌握MOV、MOVB、MOVW、XCH、XCHD、SWAP、PUSH和POP的应用，熟记各指令的功能及指令格式。

2. 掌握算术运算类指令。

掌握ADD、ADDC、MUL、DIV、SUBB、INC、DEC和DA的应用，熟记各指令功能及指令格式。

【实验要求】

1. 完成以下内容：

1) 假定外部数据存储器2000H单元的内容为80H，执行下列指令后，累加器A中的内容是（ ）：

MOV P2, #20H

MOV R0, #00H

MOVX A, @R0

注意：完成本实验，需完成题目的“假定”条件，将外部数据存储器2000H单元预先赋值，如何实现？

2) 假定 (SP) = 60H, (ACC) = 30H, (B) = 70H, 执行下列指令:

PUSH ACC

PUSH B

后, (SP) 的内容是 (), 61H 单元的内容是 (), 62H 单元的内容是 () .

注意: 完成本实验, 需完成题目的“假定”条件, 将 SP、ACC 和 B 中赋好值, 如何实现?

3) 假定 (A) = 85H, (R0) = 20H, (20H) = 0AFH, 执行下列指令:

ADD A, @R0

后, A 的内容是 (), CY 的内容是 (), AC 的内容是 (), OV 的内容是 ()。

注意: 完成本实验, 需完成题目的“假定”条件。

4) 假定 (A) = 0FFH, (R3) = 0FH, (30H) = 0F0H, (R0) = 40H, (40H) = 00H, 执行下列指令:

INC A

INC R3

INC 30H

INC @R0

后, A 的内容是 (), R3 的内容是 (), 30H 的内容是 (), 40H 的内容是 ()。

注意: 完成本实验, 需完成题目的“假定”条件。

5) 假定 (A) = 056H, (R5) = 67H, 执行下列指令:

ADD A, R5

DA A

后, A 的内容是 (), CY 的内容是 ()。

注意: 完成本实验, 需完成题目的“假定”条件。

6) 假定 (A) = 0FH, (R7) = 19H, (30H) = 00H, (R1) = 40H, (40H) = 0FFH, 执行下列指令:

DEC A

DEC R7

DEC 30H

DEC @R1

后，A的内容是（），R7的内容是（），30H的内容是（），40H的内容是（）。

注意：完成本实验，需完成题目的“假定”条件。

7) 假定 (A) = 0FH, (B) = 0A0H, , 执行下列指令:

MUL AB

后，A的内容是（），B的内容是（），AC的内容是（），OV的内容是（）。

注意：完成本实验，需完成题目的“假定”条件。

8) 假定 (A) = 0FBH, (B) = 12H, , 执行下列指令:

DIV AB

后，A的内容是（），B的内容是（），AC的内容是（），OV的内容是（）。

注意：完成本实验，需完成题目的“假定”条件。

9) 假定 (A) = 0C5H, 执行下列指令:

SWAP A

后，A的内容是（），AC的内容是（），OV的内容是（）。

注意：完成本实验，需完成题目的“假定”条件。

10) 完成书74页，第6题。

注意：完成本实验，需完成题目的“假定”条件。

11) 完成书75页，第4题。

注意：完成本实验，需完成题目的“假定”条件，m的大小自定。

2. 注意观察各个寄存器的状态变化。

【实验结果】

完成指导老师的要求，完成实验报告，详细记录实验结果。

实验 1-2 MCS-51 汇编语言指令系统（二）

【背景知识】

（1） 逻辑运算类指令（10 种助记符）

ANL(AND Logic) 逻辑与；

ORL(OR Logic) 逻辑或；

XRL(Exclusive-OR Logic) 逻辑异或；

CLR(Clear) 清零；

CPL(Complement) 取反；

RL(Rotate left) 循环左移；

RLC(Rotate Left throught the Carry flag) 带进位循环左移；

RR(Rotate Right) 循环右移；

RRC (Rotate Right throught the Carry flag) 带进位循环右移；

SWAP (Swap) 低 4 位与高 4 位交换；

（2） 控制转移类指令（17 种助记符）

ACALL (Absolute subroutine Call) 子程序绝对调用；

LCALL (Long subroutine Call) 子程序长调用；

RET (Return from subroutine) 子程序返回；

RETI (Return from Interruption) 中断返回；

SJMP (Short Jump) 短转移；

AJMP (Absolute Jump) 绝对转移；

LJMP (Long Jump) 长转移；

CJNE (Compare Jump if Not Equal)比较不相等则转移；

DJNZ (Decrement Jump if Not Zero)减 1 后不为 0 则转移；

JZ (Jump if Zero)结果为 0 则转移；

JNZ (Jump if Not Zero) 结果不为 0 则转移；

JC (Jump if the Carry flag is set)有进位则转移；

JNC (Jump if Not Carry)无进位则转移；

JB (Jump if the Bit is set) 位为 1 则转移;

JNB (Jump if the Bit is Not set) 位为 0 则转移;

JBC (Jump if the Bit is set and Clear the bit) 位为 1 则转移, 并清除该位;

NOP (No Operation) 空操作;

(3) 位操作指令 (1 种助记符)

SETB (Set Bit) 位置 1。

【实验内容】

对逻辑运算、移位指令、控制转移类指令和位操作类指令的每一条指令, 进行执行, 并注意观察相关寄存器及状态寄存器 PSW 的变化和运行效果。

【实验目的】

1. 掌握逻辑运算及移位类指令。

掌握 ANL、ORL、XRL、CLR、CPL、RR、RL、RRC、RLC 的应用, 熟记各指令的功能及指令格式。

2. 掌握控制转移类指令。

掌握 LJMP、AJMP、SJMP、JMP、JZ、JNZ、CJNE、DJNZ、LCALL、ACALL、RET、RETI 的应用, 熟记各指令的功能及指令格式。

3. 掌握位操作类指令。

掌握 MOV、SETB、CLR、ANL、ORL、JC、JNC、JB、JBC 的应用, 熟记各指令的功能及指令格式。

【实验要求】

1. 完成以下内容:

1) 假定 (A) = 83H, (R0) = 17H, (17H) = 34H, 执行下列指令:

ANL A, #17H

ORL 17H, A

XRL A, @R0

CPL A

后，A的内容是（ ）。写出每条指令执行后的目的操作数的内容。

注意：完成本实验，需完成题目的“假定”条件。

2)执行下列指令：

MOV 71H, #17H

MOV R0, #71H

MOV A, @R0

RL A

MOV R1, A

RL A

RL A

ADD A, R1

MOV @R0, A

后，实现的功能是（ ）。写出每条指令执行后的目的操作数的内容。

注意：完成本实验，需完成题目的“假定”条件。

2. 下列程序执行后，SP=? A=? B=?，解释每一条指令的作用，给最终执行结果的合理解释，写到作业本上，交上来。

```
ORG 0000H
LJMP MAIN
ORG 0030H
MAIN: MOV SP,#40H
      MOV A,#30H
      LCALL 0100H
      ADD A,#04H
      MOV B,A
      SJMP $
```

```
ORG 0100H
MOV DPTR,#0039H
PUSH DPL
PUSH DPH
RET
```

3) 按指导教师要求, 完成逻辑运算、移位指令、控制转移类指令和位操作类指令的训练。

3. 注意观察各个寄存器的状态变化。

【实验结果】

完成指导老师的要求, 完成实验报告, 详细记录实验结果。

实验 1-3 MCS-51 汇编语言指令系统（三）

【背景知识】

参考实验 1-1 和实验 1-2。

【实验内容】

对不同寻址方式进行比较，加以区分。更加明晰 51 单片机寻址方式的不同，得到的效果也不同。

【实验目的】

1. 对比存储器寻址和立即数寻址；
2. 对比寄存器寻址和寄存器间接寻址；
3. 对比外部 RAM 不同寻址方法；
4. 对比代码寻址和 RAM。

【实验要求】

1. 完成以下内容：
 - 1) 对比存储器寻址和立即数寻址，完成下面代码：

MOV 00H, #20H

A) MOV A, #00H

B) MOV A, 00H

MOV 30H, #50H

C) MOV A, #30H

D) MOV A, 30H

①写出每条指令执行后的目的操作数的内容，注意辨析指令的不同。

②说明A) 代码段和B) 代码段，C) 代码段和D) 代码段实现功能的异同。

- 2) 对比寄存器寻址和寄存器间接寻址，完成下面代码：

A) MOV 20H,#50H

MOV R0,#20H

B) MOV A, R0

MOV A,@R0

C) MOV 30H,#60H

MOV R1,#30H

D) MOV A, R1

MOV A,@R1

①写出每条指令执行后的目的操作数的内容，注意辨析指令的不同。

②说明A) 代码段和B) 代码段，C) 代码段和D) 代码段实现功能的异同。

3) 对比外部 RAM 不同寻址方法，完成下面代码：

A) MOV DPTR,#0000H

MOV A,#02H

MOVX @DPTR,A ;

MOV DPTR,#3000H

MOV A,#03H

MOVX @DPTR,A

B) MOV R0,#00H

MOVX A,@R0

MOV P2,#00H

MOVX A,@R0

MOV DPTR,#0000H

MOVX A,@DPTR

MOV P2,#30H

```
MOVX A,@R0
```

```
MOV DPTR,#3000H
```

```
MOVX A,@DPTR
```

①写出每条指令执行后的目的操作数的内容，注意辨析指令的不同；

②说明A) 代码段和B) 代码段实现功能的异同。

4) 对比代码寻址和RAM，完成下面代码：

A) MOV DPTR, #0800H

```
MOV A, #05H
```

```
MOVC A, @A+DPTR
```

B) MOV DPTR, #0805H

```
MOVX A, @DPTR
```

```
ORG 0800H
```

```
DB '1','2','3','4','5','6','7','8','9'
```

①写出每条指令执行后的目的操作数的内容，注意辨析指令的不同。

②说明A) 代码段和B) 代码段实现功能的异同。

2. 注意观察各个寄存器的状态变化。

【实验结果】

完成指导老师的要求，完成实验报告，详细记录实验结果。

实验 1-4 MCS-51 汇编语言程序设计（一）

【背景知识】

一、设计步骤

单片机汇编语言程序设计的基本步骤如下：

1) 题意分析：

熟悉并了解汇编语言指令的基本格式和主要特点，明确被控对象对软件的要求，设计出算法等。

2) 画出程序流程图：

编写较复杂的程序，画出程序流程图是十分必要的。程序流程图也称为程序框图，是根据控制流程设计的，它可以使程序清晰，结构合理，按照基本结构编写程序，便于调试。

3) 分配内存工作区及有关端口地址：

分配内存工作区，要根据程序区、数据区、暂存区、堆栈区等预计所占空间大小，对片内外存储区进行合理分配并确定每个区域的首地址，便于编程使用。

4) 编制汇编源程序。

5) 仿真调试程序。

6) 固化程序。

二、汇编语言格式

不同 CPU,其汇编语言指令集不同（硬件结构不同），但各种汇编语言的语法规则基本一样，具有类似的语句格式：

1.语句格式：

[标号：] 操作码 [操作数，] [；注释]

(1) 标号：

标号是由 1—8 个 ASCII 字符组成，起始字符必须是字母，其余字符可以是字母、数字或其它特定字符。

关键字不能作为标号，如指令助记符、伪指令记忆符以及寄存器的符号名称等。

标号后边必须跟以“:”,不能重复定义,大小写一样

语句只有在其它语句访问该条语句时,施加标号。

(2)操作码

操作码执行规定语句的操作。操作码以指令助记符或伪指令助记符表示。操作码是汇编指令中唯一不能空缺的部分。

(3)操作数

操作数用于给指令的操作提供数据或地址。在一条语句中,操作数可能是空白,也可能包括一项/二项/三项。各操作数之间以逗号分隔。MCS-51 的操作数有七种不同的寻址方式。

(4)注释

注释不属于语句的功能部分,它只是对语句的解释说明。只要用“;”开头,即表明后面为注释内容。

2.汇编语言伪指令:

伪指令的功能是向汇编程序发出指示信息,告诉编译程序如何完成汇编工作,其本身不属于指令系统,只是辅助汇编指令的执行。如用伪指令给程序分配一定的存储区、定义符号、指定暂存数据的 RAM 等。

1)ORG (ORiGin) 汇编起始命令

格式: ORG 16 位地址

如: ORG 0100H

START: MOV A,#20H

2)END 程序结束命令

格式: END

3)EQU(EQUate) 赋值命令

格式: 字符名称 EQU 赋值项(数据、表达式、字符串)

说明: 相当于 #define

- (1)字符名称必须以字母开头;
- (2)用字符表示的数据,汇编不能区分立即数和地址,在使用中确定;
- (3)若定义的是地址或寄存器,可作变量用。
- (4)可以把一个表达式赋给字符名称(表达式必须可求值)

4)DATA 数据地址赋值命令

格式: 字符名称 DATA 表达式(数据、表达式)

COUNT DATA 30H

COUNT DATA 30H+2

说明:

- (1) 主要用于定义变量或数据;
- (2) 格式中的表达式必须是确定的值;
- (3) 与 EQU 的区别:
 - ①DATA 中的表达式不能是字符串;
 - ②DATA 的作用仅是定义数据存储器 RAM 的地址,而 EQU 可以定义 ROM 或 RAM 的地址

5)DB(Define Byte) 定义字节命令

格式: DB 数据、数据表或字符串

说明:

- (1) 各项用逗号分开;
- (2) 字符串用单/双引号括起来;
- (3) 数据存入 ROM 指定(ORG)的空间。

6)DW (Define Word) 定义字命令

格式: DW 16 位数据、数据表

说明:

- (1) 各项用逗号分开,放字母时最多一次 2 个;
- (2) 先存放高 8 位,再存放低 8 位;

(3) 数据存入 ROM 指定的空间。

7) DS 定义存储空间命令

格式: DS 表达式 (常数)

说明: 汇编时, 从指定地址开始保留一定数量的存储空间。

如: ORG 1000H
 DS 08H
 DB 30H, 8AH

汇编后:

1000H~1007H 单元保留

(1008H) = 30H (1009H) = 8AH

8) BIT 定义位地址符号 (位变量) 命令

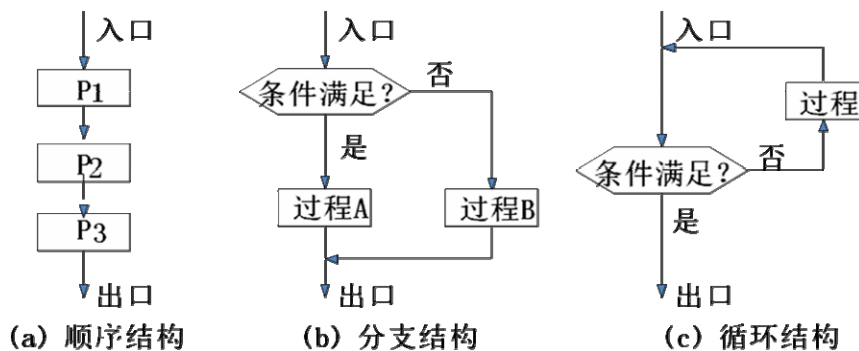
格式: 字符名 BIT 位地址

如:

A1 BIT P1.0
A2 BIT 02H
FLAGRUN BIT 08H
MOV C, A1
MOV A2, C

三、程序设计结构

设计编写程序时, 顺序结构、分支结构和循环结构:



汇编程序结构

顺序结构：无分支，无循环，指令逐条执行

【实验内容】

从程序结构及应用出发，通过汇编语言典型程序设计，进一步理解和掌握 MCS51 单片机的指令系统，并熟练掌握程序设计的方法和技巧。通过本主题实验的学习，应熟练掌握顺序程序、分支程序和查表程序的设计方法；掌握循环程序设计方法；掌握数制和码制转换程序的设计方法；掌握软件定时程序设计方法。

【实验目的】

1. 掌握汇编语言语句格式；
2. 掌握顺序结构程序编写；

【实验要求】

1. 完成以下内容：
 - 1) 编写书 79 页例程，将程序补完整，并进行调试，观察程序执行结果。
 - 2) 按指导老师要求完成其它题目。
2. 注意程序前后语句逻辑的衔接。

【实验结果】

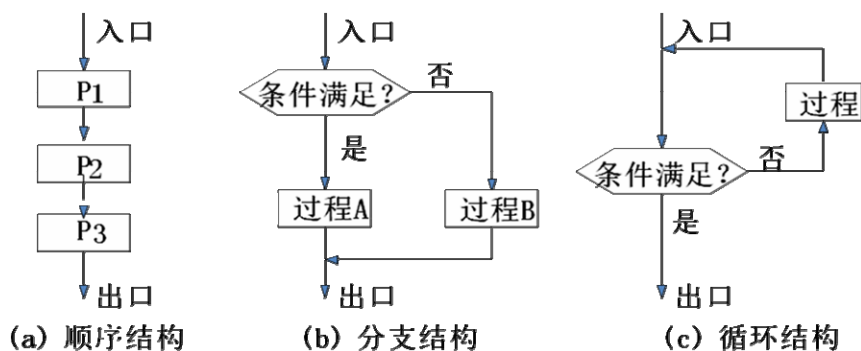
完成指导老师的要求，完成实验报告，详细记录实验结果。

实验 1-5 MCS-51 汇编语言程序设计（二）

【背景知识】

分支结构

设计编写程序时，顺序结构、分支结构和循环结构。



汇编程序结构

程序分支是通过条件转移指令实现的，即根据条件对程序的执行进行判断，满足条件则进行程序转移，不满足条件程序就顺序执行。

- (1) 无条件转移：LJMP, AJMP, SJMP
- (2) 条件转移：JZ、JNZ、CJNE 和 DJNZ 等。
- (3) 按位转移：JC、JNC、JB、JNB 和 JBC 等。

使用这些指令，可以完成为各种条件判断：

0、1、正、负、相等、不相等。

【实验内容】

从程序结构及应用出发，通过汇编语言典型程序设计，进一步理解和掌握 MCS51 单片机的指令系统，并熟练掌握程序设计的方法和技巧。通过本主题实验的学习，应熟练掌握顺序程序、分支程序和查表程序的设计方法；掌握循环程序设计方法；掌握数制和码制转换程序的设计方法；掌握软件定时程序设计方法。。

【实验目的】

1. 掌握单分支程序设计方法。
2. 掌握多分支程序设计方法。

【实验要求】

1. 完成以下内容：

1) 实现单分支程序设计。

编写书 80 页，“单分支结构举例”和“多重单分支结构举例”，将程序补完整，并进行调试，观察程序执行结果。

2) 实现多分支程序转移；

编写书 80 页，“多分支程序结构”的三种实现方法，将程序补完整，并进行调试，观察程序执行结果。

3) 按指导老师要求完成其它题目。

2. 注意程序前后语句逻辑的衔接。

【实验结果】

完成指导老师的要求，完成实验报告，详细记录实验结果。

实验 1-6 MCS-51 汇编语言程序设计（三）

【背景知识】

一、循环结构

循环初值、循环体、循环修改、循环控制、循环退出。

- 1、计数法：循环次数已知，DJNZ 指令
- 2、条件控制法：循环次数未知，条件转移指令
- 3、多重循环

二、子程序设计

ACALL addr11

LCALL addr16

RET

注意：入口参数 出口参数 避免主程序与子程序所用的寄存器冲突

【实验内容】

从程序结构及应用出发，通过汇编语言典型程序设计，进一步理解和掌握 MCS51 单片机的指令系统，并熟练掌握程序设计的方法和技巧。通过本主题实验的学习，应熟练掌握顺序程序、分支程序和查表程序的设计方法；掌握循环程序设计方法；掌握数制和码制转换程序的设计方法；掌握软件定时程序设计方法。

【实验目的】

1. 掌握循环程序设计方法。
2. 掌握算术运算程序。
不带符号的单字节加法；不带符号的多字节加法。
3. 掌握数制转换程序。
十六进制与ASCII码的相互转换

【实验要求】

1. 完成以下内容:

1) 掌握循环程序设计方法。

编写书 83 页,“循环程序结构”,将程序补完整,并进行调试,观察程序执行结果。

2) 实现算术运算程序设计;

编写书 84 页,“加减法运算”、“乘法运算”和“除法运算的例程”,将程序补完整,并进行调试,观察程序执行结果。

3) 实现数制转换程序设计;

编程实现 2 位 16 进制数转换为 ASCII 码和 ASCII 码转换为 16 进制数。

注意: 0...9 的 ASCII 码为“30H...39H”, A...F 的 ASCII 码为“41H...46H”。

4) 完成指导老师其它要求。

2. 注意程序前后语句逻辑的衔接,注意归纳总结程序设计方法技巧。

【实验结果】

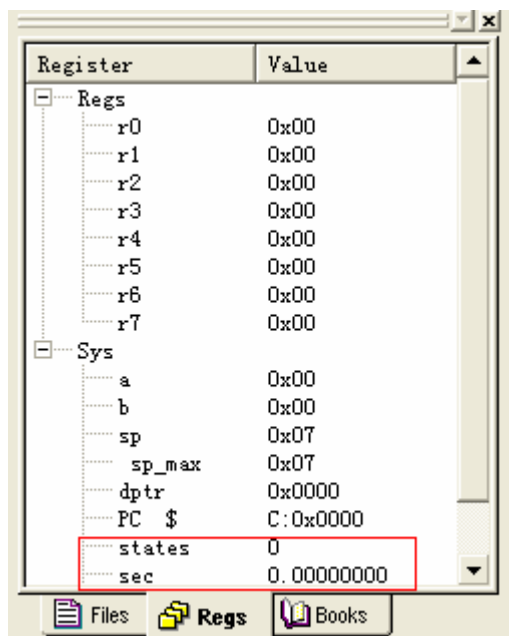
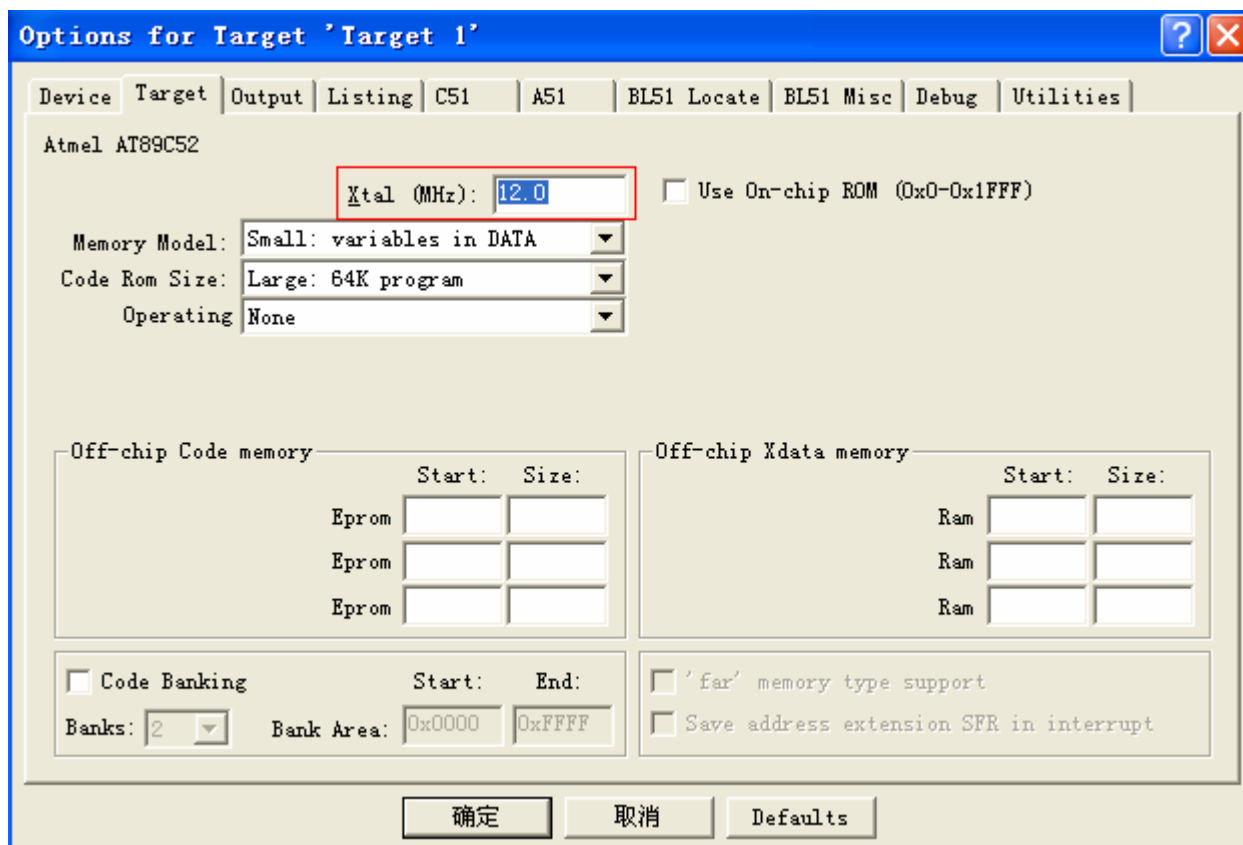
完成指导老师的要求,完成实验报告,详细记录实验结果。

实验 1-7 MCS-51 汇编语言程序设计（四）

【背景知识】

KeilC 调试软件延时方法：

1、首先打开“Options for Target”页面，在“Xtal (MHz)”中输入选择的晶振频率。



2、然后，借助“states”和“sec”这两个 KeilC 提供的变量帮助我们调试出精确地延时时间。如左图所示，特殊寄存器显示页面，即“Regs”页面中。

states 含义：其值（Value）指示了程序从开始到当前指令处所耗费的机器周期数。

sec 含义：其值（Value）指示了程序从开始到当前指令处所耗费的机器周期数。

其中： $\text{sec} = \text{states} \times \text{机器周期}$ 。

我们观察这两个变量，通过调整延时循环控制参数，可以得到需要的精确延时时间。

【实验内容】

从程序结构及应用出发，通过汇编语言典型程序设计，进一步理解和掌握 MCS51 单片机的指令系统，并熟练掌握程序设计的方法和技巧。通过本主题实验的学习，应熟练掌握顺序程序、分支程序和查表程序的设计方法；掌握循环程序设计方法；掌握数制和码制转换程序的设计方法；掌握软件定时程序设计方法。

【实验目的】

- 1、掌握软件定时程序设计方法。
- 2、掌握KeilC调试精确定时的方法。

【实验要求】

1. 将晶振频率设定为6MHz。
2. 完成以下内容，**将程序补全完整，并进行调试，观察程序执行结果：**
 - 1) 掌握软件定时程序设计方法，。

A、单循环定时程序

```
MOV R5, #TIME
```

```
LOOP: NOP
```

```
NOP
```

```
DJNZ R5,LOOP
```

◆ 计算本延时最短和最长延时时间。

◆ 按老师要求延时不同的时间，求 TIME 设定的具体数值。

B、较长时间定时程序

```
MOV R5, #TIME1
```

```
LOOP2 : MOV R4, #TIME2
```

```
LOOP1:  NOP
```

```
NOP
```

```
DJNZ R4,LOOP1
```

DJNZ R5,LOOP2

- ◆ 计算本延时最短和最长延时时间。
- ◆ 按老师要求延时不同的时间，求 TIME 设定的具体数值。

C、调整定时时间程序

对 A 和 B 进行适当修改，如增添“NOP”指令，对延时量做调整。

D、通过基本延时子程序产生不同的定时

要求定时的时间为 10s，编程实现精确的 1s 延时子程序 DELAY1S。

MOV R5, #10

LOOP: LCALL DELAY1S ;精确延时 1s

DJNZ R5,LOOP

2) 完成指导老师其它要求。

3. 注意程序前后语句逻辑的衔接，注意归纳总结程序设计方法技巧。

【实验结果】

完成指导老师的要求，完成实验报告，详细记录实验结果。

实验 1-8 MCS-51 汇编语言程序设计（五）

【背景知识】

冒泡排序：

冒泡排序就是排序方法中的最为基础和原始的一种。在各种程序语言中，虽然它们写法各异，但是原理是一样的。它排序的基本思想是：两两比较是排序数据的关键字，发现两个数据的次序相反时即进行交换，直到没有反序的记录为止。这个算法的名字由来是因为越小的元素会经由交换的动作慢慢“浮”到数列的顶端。冒泡排序对 n 个项目需要的比较次数约为 n^2 ，且可以原地排序。尽管这个算法是最简单了解和实用的排序算法之一，但它对于少数元素之外的数列排序效率不高。

冒泡排序算法的运作如下：

- 1、比较相邻的元素。如果第一个比第二个大，就交换他们两个。
- 2、对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
- 3、针对所有的元素重复以上的步骤，除了最后一个。
- 4、持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要交换。

【实验内容】

从程序结构及应用出发，通过汇编语言典型程序设计，进一步理解和掌握 MCS51 单片机的指令系统，并熟练掌握程序设计的方法和技巧。通过本主题实验的学习，应熟练掌握顺序程序、分支程序和查表程序的设计方法；掌握循环程序设计方法；掌握数制和码制转换程序的设计方法；掌握软件定时程序设计方法。

【实验目的】

1. 掌握查表程序方法。
掌握用 MOVC 指令查表的程序设计
2. 掌握数据排序程序方法。
掌握用冒泡法对数据进行升序或降序排序

【实验要求】

1. 完成以下内容：

1) 掌握查表程序设计。

编写90页例程序，将程序补完整，并进行调试，观察程序执行结果。

2) 掌握数据排序程序设计。

假定8个数连续存放在20H为首地址的内部RAM单元中，使用冒泡法进行升序排序编程。设R7为比较次序计数器，初始值为07H。TF0为冒泡过程中是否有数据交换的标志位，TF0=0表面无交换发生，TF0=1表明有交换发生。

编程实现上述功能，并进行调试，观察程序执行结果。

3) 完成指导老师其它要求。

2. 注意程序前后语句逻辑的衔接，注意归纳总结程序设计方法技巧。

【实验结果】

完成指导老师的要求，完成实验报告，详细记录实验结果。

第二部分 单片机系统的 Proteus 设计与仿真

一、PROTEUS 介绍

PROTEUS 是英国 Labcenter electronics 公司研发的 EDA 工具软件。PROTEUS 不仅是模拟电路、数字电路、模 / 数混合电路的设计与仿真平台，更是目前世界上最先进、最完整的多种型号单片机 (微控制器) 系统的设计与仿真平台。它真正实现了在计算机上完成从原理图设计、电路分析与仿真、单片机代码级调试与仿真、系统测试与功能验证到形成 PCB 的完整的电子设计、研发过程。PROTEUS 从 1989 年问世至今，经过了近 20 年的使用、发展和完善，功能越来越强，性能越来越好。PROTEUS 已在全球广泛使用。

PROTEUS 分为 PROTEUS VSM (Virtual System Modeling: 虚拟系统模型) 和 PROTEUS PCB 两大功能。简单来说，这两大功能由应用软件 **PROTEUS ARES** (Advanced Routing 安定 Editing Software : 高级布线编辑软件) 和 **PROTEUS ISIS** (Intelligent Schematic Input System: 智能原理图输入系统) 分别实现 (实际上功能略有交叉)，他们在 PROTEUS professional (专业版) 安装后都会出现在安装目录里。本实验仿真系统目前采用的是 PROTEUS professional 7.1SP2 版本。

PROTEUS VSM 能实现数字电路、模拟电路及数模混合电路的设计与仿真，特别是能实现单片机与外设的混合电路系统、软件系统的设计和仿真。后者是 PROTEUS 最具特色的革命性功能。在仿真过程中，用户可以用鼠标单击开关、键盘、电位计、可调电阻等动态外设模型，使单片机系统根据输入信号做出相应的响应，并将响应处理结果实时地显示在 LED、LCD 等动态显示器件上，实现了实时交互式仿真。整个过程与真实的软件、硬件调试过程相似。

PROTEUS PCB 设计系统是基于高性能网表的设计系统，组合了 ISIS 原理图捕捉和 ARES PCB 输出程序，构成一个强大的易于使用的设计 PCB 的工具包，能完成高效、高质的 PCB 设计。

Proteus 是一个完整的嵌入系统软、硬件设计仿真平台，它包括原理图输入系统 ISIS、带扩展的 ProSpice 混合模型仿真器、动态器件库、高级图形分析模块和处理器虚拟系统仿真模型 VSM。

Proteus VSM 的核心是 ProSpice，这是一个组合了 SPICE3F5 模拟仿真器核和基于快速


实践事件驱动的数字仿真器的混合仿真系统，SPICE 内核使用众多制造商提供的 SPICE 模型。Proteus VSM 包含大量的虚拟仪器、逻辑分析仪、函数发生器、数字信号图形发生器、时钟计数器、虚拟终端，以及简单的电压计和电流计。

Proteus VSM 最重要的特点是，它能把微处理器软件作用在处理器上，并和该处理器的任何模拟和数字器件协同仿真。仿真执行目标码就像在真正的单片机系统上运行，VSM CPU 模型能完整仿真 I/O 口、中断、定时器、通用外设口和其它 CPU 有关的外设，甚至能仿真多个处理器。

ISIS 是 Proteus 系统的核心，它是我们本次试验仿真的主要软件，完成原理图的设计并与 KeilC 集成开发环境配合，完成有关单片机的仿真试验。Proteus 还有很多深层次的功能，有待同学们去探索学习，限于课程安排和时间限制，实验不涉及 PROTEUS PCB 部分，即 PROTEUS ARES 软件的使用。Proteus ISIS 使用简单，易上手，希望同学们用好，用熟该软件，使得单片机的学习达到事半功倍的效果。

二、PROTEUS ISIS 入门

1、创建一个设计

1) 单击桌面快捷键。
或者，单击“开始”->“所有程序”
->“Proteus 7 Professional”->“ISIS
7 Professional”，如图 2-1。

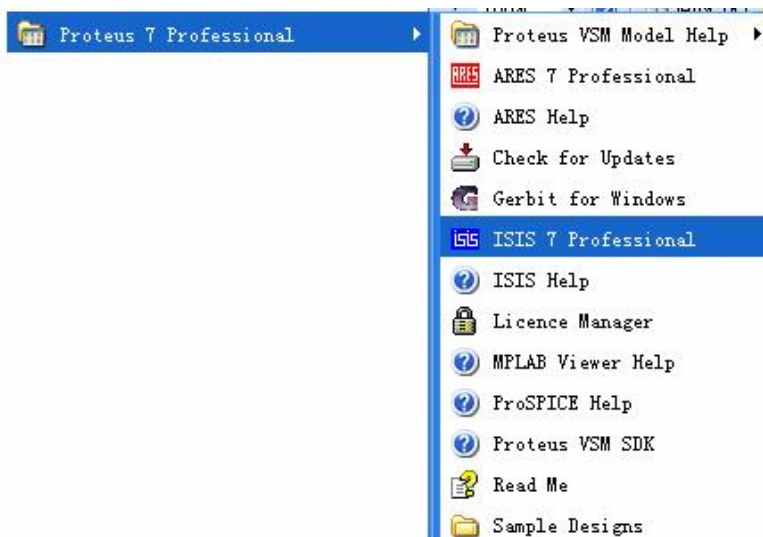


图 2-1

程序运行后的界面如图 2-2

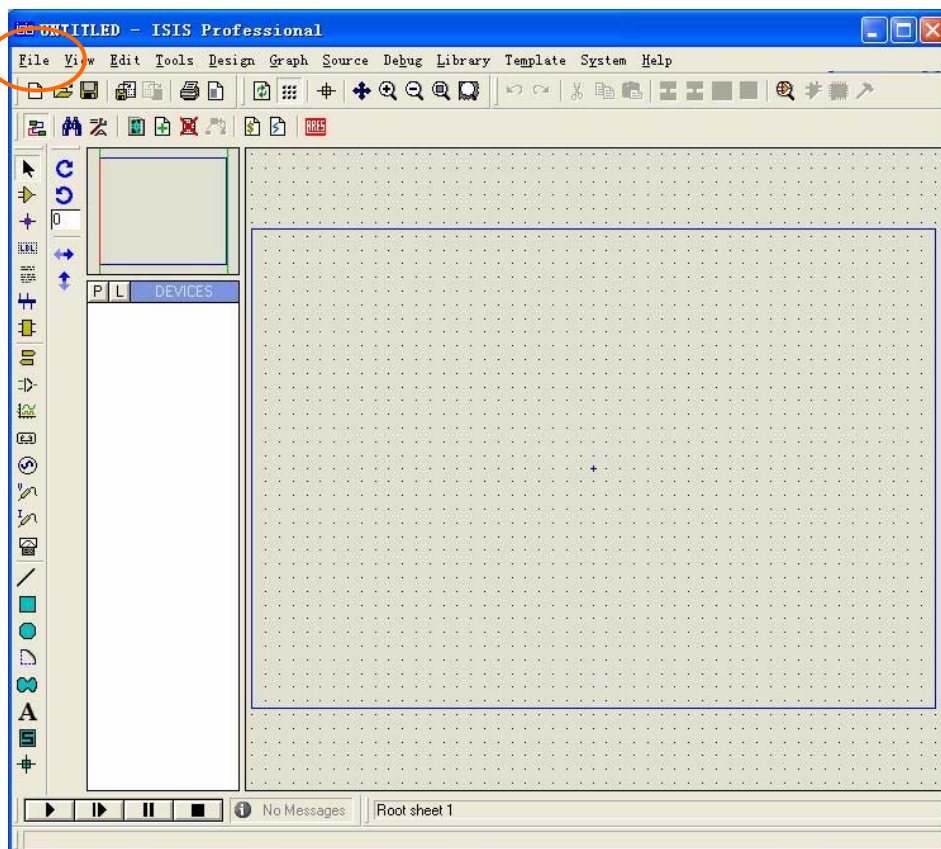


图 2-2

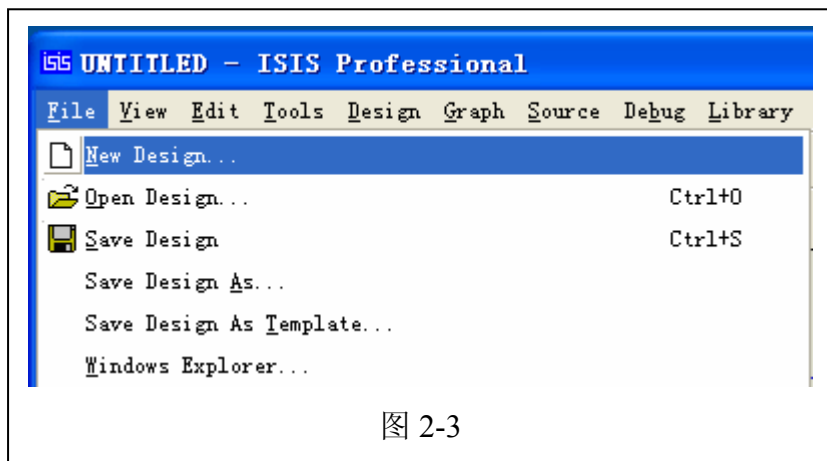


图 2-3

2) 单击菜单栏“FILE”，选择“New Design”。如图 2-3。

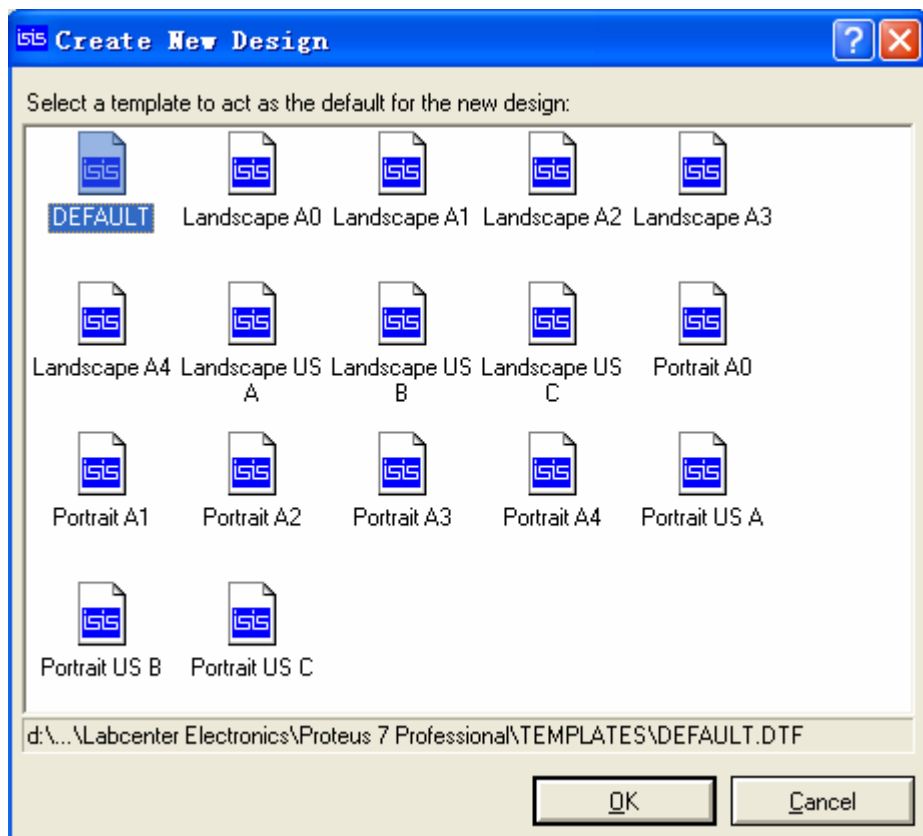



图 2-4

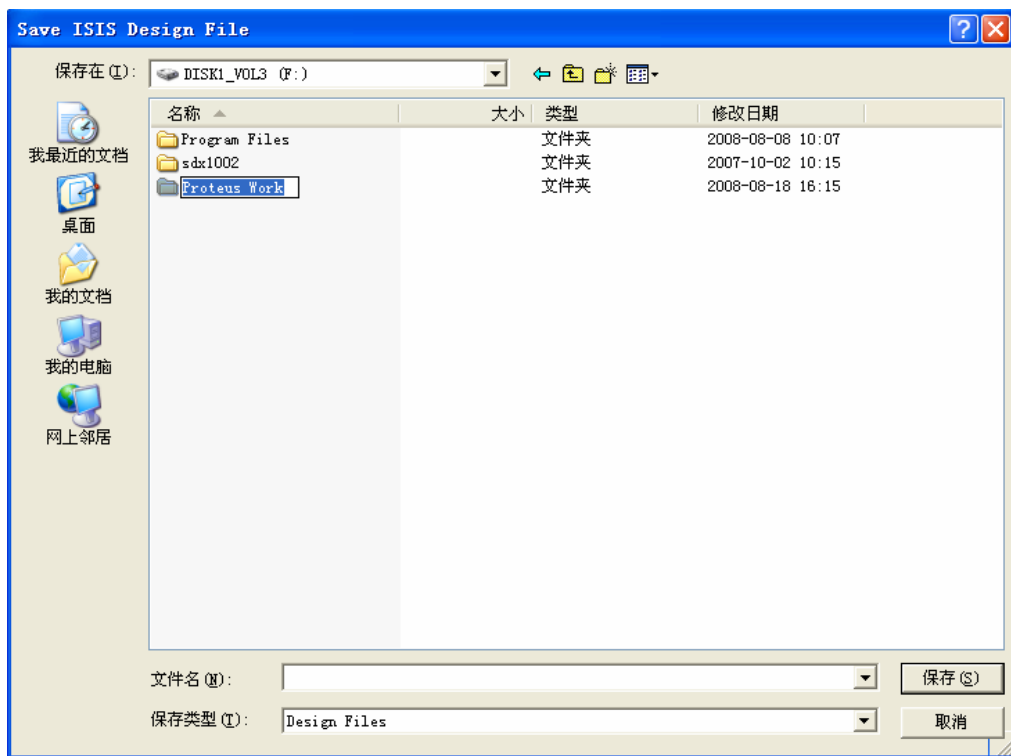
选中“DEFAULT”，即默认模板作为设计模板。

模板模式的选择主要是为原理图设计选定图纸的大小。系统默认方式大小为我们常用的“A4”大小。若要在以后更改设计图纸大小，可以单击菜单“System”->“Set Sheet Sizes”后弹出的对话框中进行设置。

随后，界面如图 2-2 所示，即实际上开始启动“ISIS 7 Professional”后，程序就已经处在这一步骤。

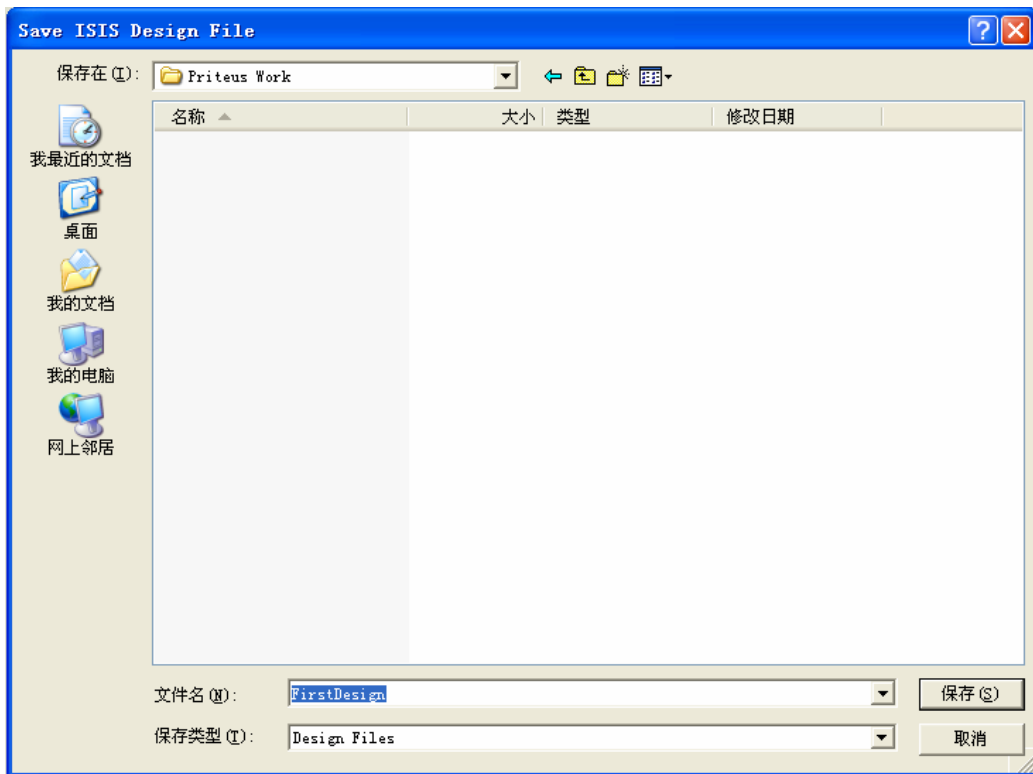
3) 保存设计，另起文件名

单击工具栏上图标，弹出下面对话框，



为便于管理，
选择合适的路
径，创建新文
件夹，这里创
建“Proteus
Work”文件夹
如图 2-5。

图 2-5



为设计起一个
名字，这里键
入
“FirstDesign
”，如图 2-6，
单击“保存”
按钮，保存的
新文件夹里。

图 2-6

文件保存成功，显示如图 2-7 界面，左上角标题栏显示保存文件名：“FirstDesign”。

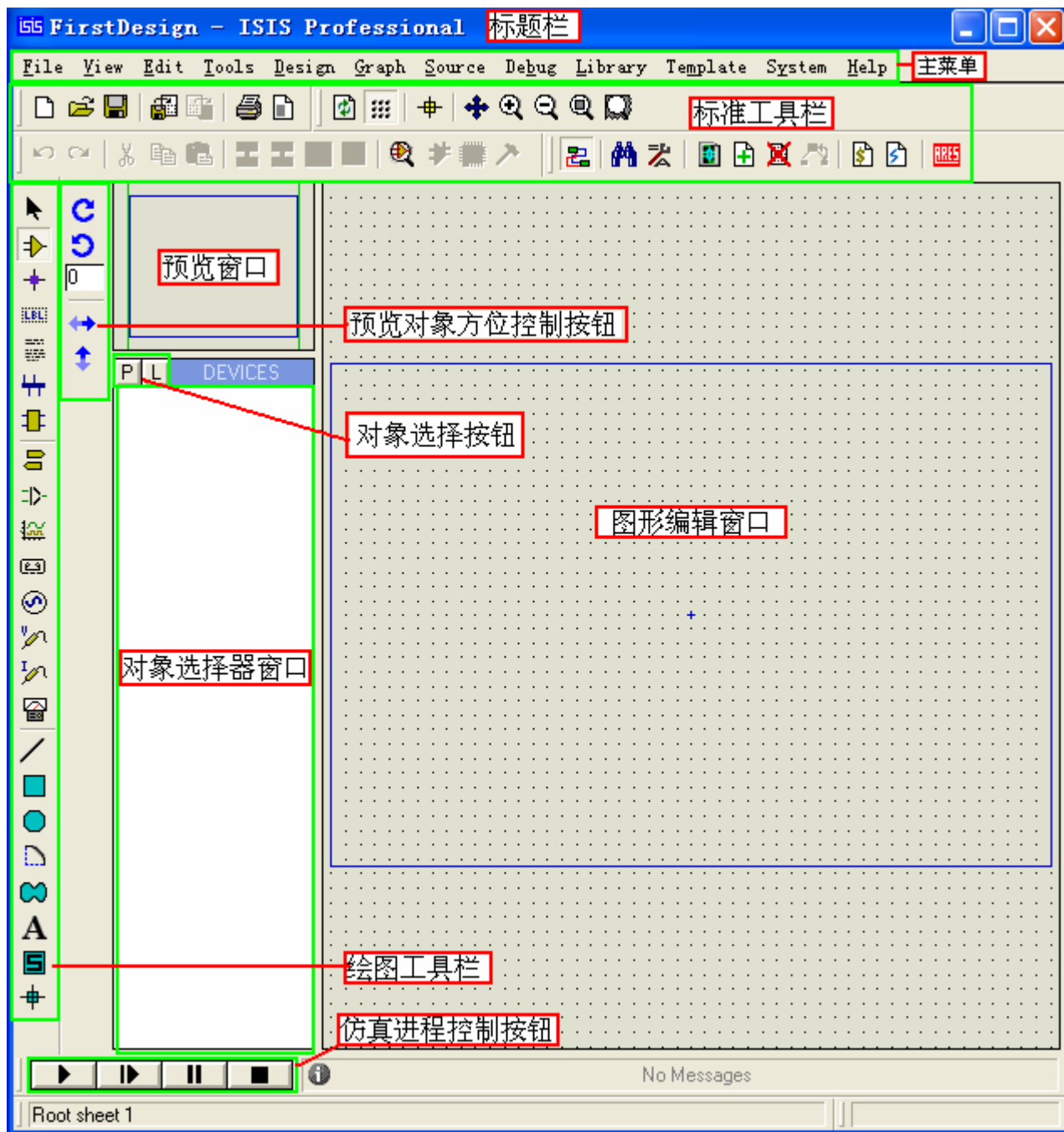



图 2-7

。

2、设计原理图：

1) 单击“绘图工具栏”里的按钮，再单击对象选择按钮“P”，出现如图 2-8 所示对话框。

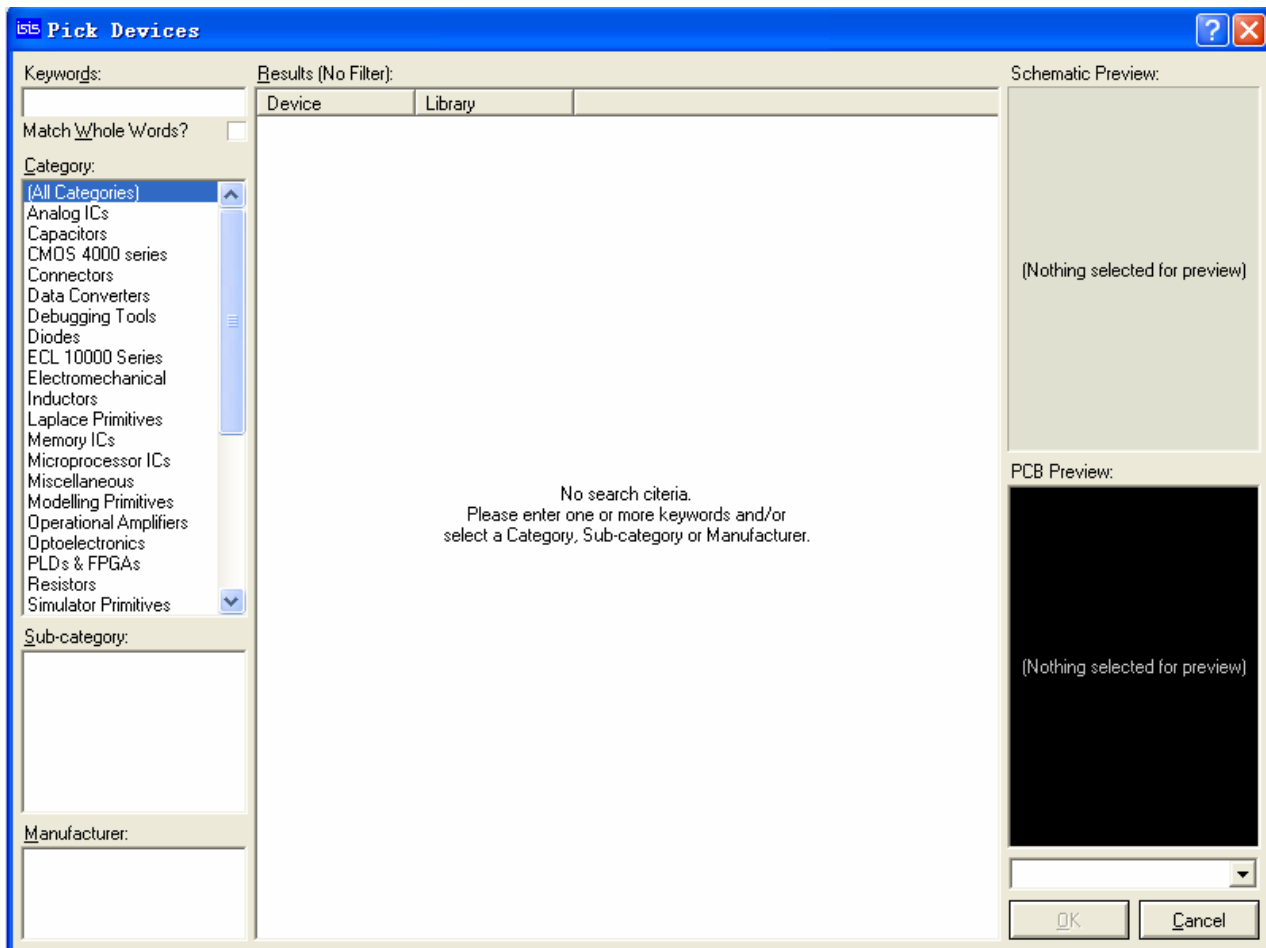


图 2-8

在“Keywords”里输入要放置的器件名称，例如，输入“8051”，则右侧会出现大量 8051 内核兼容的单片机供用户选择。又如要查找“47K 电阻”，可以输入“47K”，然后在列出的元器件列表中进一步选择需要的元器件型号。左侧“Category（目录）”下所列的是可供选择的元器件库，PROTEUS 系统中有符号库约 30 个元器件库。元器件库如图 2-9 所示。左边 Pick Devices 窗口所列的是 PROTEUS 库列表，右边是该列表的直译中文名列表。每个库又有许多模型，总共约 8000 个。可见 PROTEUS 库相当丰富。有关系统支持的库信息，请查

看 PROTEUS 安装路径下最新的 LIBRARY. PDF 文件。

(All Categories)	未指定
(Unspecified)	模拟集成电路
Analog ICs	电容
Capacitors	CMOS 4000 系列
CMOS 4000 series	接插件
Connectors	
Data Converters	数字转换器
Debugging Tools	调试工具
Diodes	二极管
ECL 10000 series	ECL 10000 系列
Electromechanical	电动机系列
Inductors	感应器
Laplace Primitives	Laplace 原型
Memory ICs	存储器集成电路
Microprocessor ICs	微处理器集成电路
Miscellaneous	其他
Modelling Primitives	模型原型
Operational Amplifiers	运放
Optoelectronics	光电器件
PLDs & FPGAs	PLD、FPGA
Resistors	电阻
Simulator Primitives	仿真原型
Speakers & Sounders	喇叭、音响
Switches & Relays	开关继电器
Switching Devices	开关
Thermionic Valves	热离子真空管
Transducers	传感器
Transistors	晶体管
TTL 74 series	74 系列
TTL 74ALS series	74ALS 系列
TTL 74AS series	74AS 系列
TTL 74F series	74F 系列
TTL 74HC series	74HC 系列
TTL 74HCT series	74HCT 系列
TTL 74LS series	74LS 系列
TTL 74S series	74S 系列

请注意元器件类库的含义，有助于迅速查找到需要的元器件。

图 2-9

2) 在“Keywords”中输入表 2-1 当中的英文关键词，对话框右侧“Result”栏中，选中查找到的跟关键词相符的元器件，并双击，会看到改关键词的元器件会自动出现在“对象选择器窗口”内。完成后，如图 2-10 所示。

表 2-1

单片机 AT89C51	发光二极管 LED-GREEN	瓷片电容 CAP	电解电容 CAP-ELEC
电阻 RES	上拉电阻 PULLUP	晶振 CRYSTAL	按钮 BUTTON

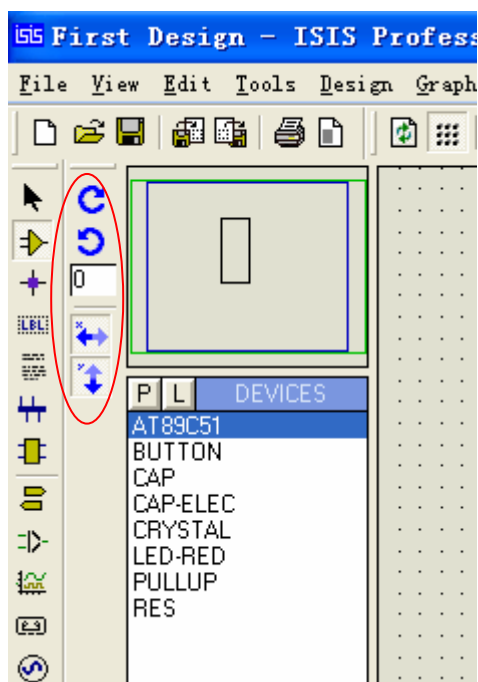




图 2-10

3) 单击 ISIS 对象选择器中的元器件名, 蓝色条出现在该元器件名上。把鼠标指针(以后简称指针)移到编辑区某位置后, 单击就可放置元器件于该位置, 每单击一次, 就放一个元器件。要移动元器件, 先单击使元器件处于选中状态(高亮度状态), 再按住鼠标左键(以后简称按住左键)拖功, 元器件就跟随指针移动, 松开鼠标即停止移动。单击“绘图工具栏”里的  按钮, 则退出放置器件状态。

4) 放置电源和地: 单击“绘图工具栏”里的  按钮, 对象选择器窗口出现如图 2-11 所示。选择 POWER 和 GROUND, 最后选择的元器件如图 2-12 所示。

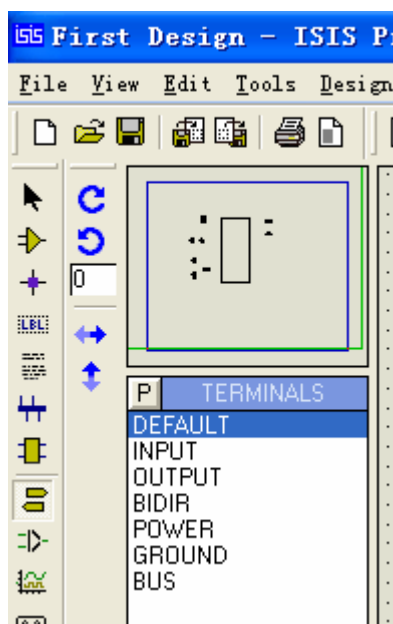


图 2-11

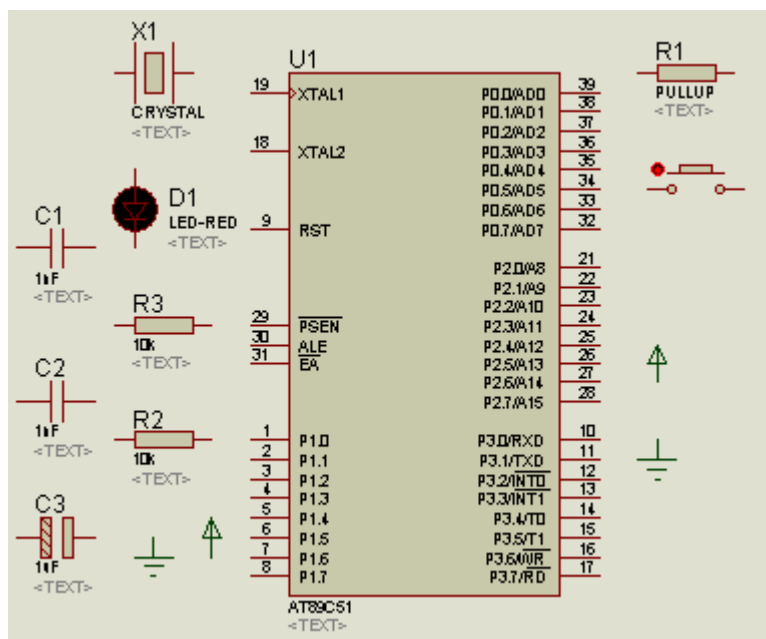
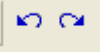


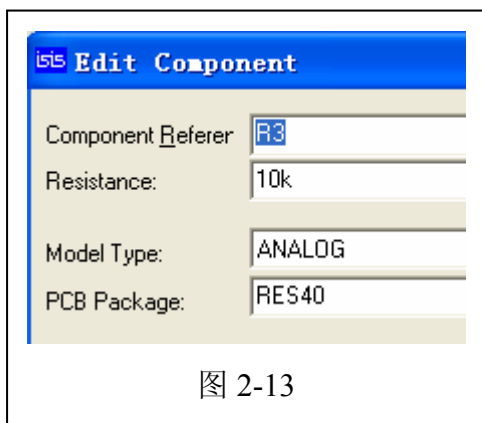
图 2-12

在放置元器件后，注意以下几点：

- 左击或右击鼠标都会使元器件处于选中状态，此时可以移动元器件。
- 鼠标右击还会弹出菜单，在菜单中可以对元器件进行旋转。元器件的旋转还可以使用数字键盘的“+”和“-”两个符号，前提是元器件在选中状态下。
- 鼠标左双击，弹出元器件属性对话框，可以对元器件的名称、值等属性进行修改。
- 鼠标右双击，删除元器件。
- 鼠标左击，选择一个矩形框，可以将多个元器件选中，可以进行块的移动、复制、删除等操作。

以上各点请同学们一一尝试，并熟练应用。注意使用  按钮，对错误操作进行修改。

5) 设置、修改元器件的属性。PROTEUS 库中的元器件都有相应的属性，鼠标左键双击该元器件，打开其属性窗口，这时可在属性窗口中设置、修改它的属性。例如，发光管的限流电阻 R3，双击打开其属性窗口如图 2-13 所示，已将电阻值 10k 修改为 300Ω。其他元器件属性值修改结果如图 2-14 所示。



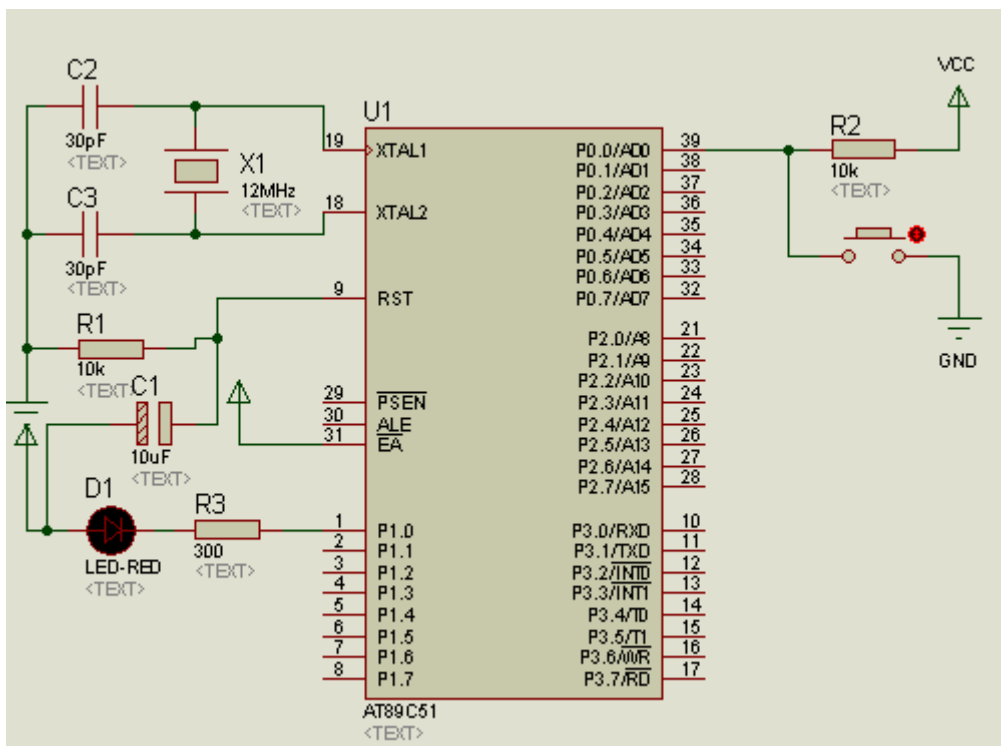



图 2-14

6) 电路图的布线，即元器件之间的连线，直接用鼠标在元器件各端点间两连接就行，位置和连线方向，可以适当调整。不合适的走线可以右键双击删除，再重新布线。如图 2-14 所示。

7) 电气检查。最后一步，设计电路完成后，单击电气检查按钮  即 (View Electrical report)，会出现检查结果窗口，如图 2-15 所示。窗口前面是一些文本信息，接着是电气检查结果列表，若有错，会有详细的说明。也可通过菜单操作“Tools->Electrical Rule Check...”，完成电气检测。

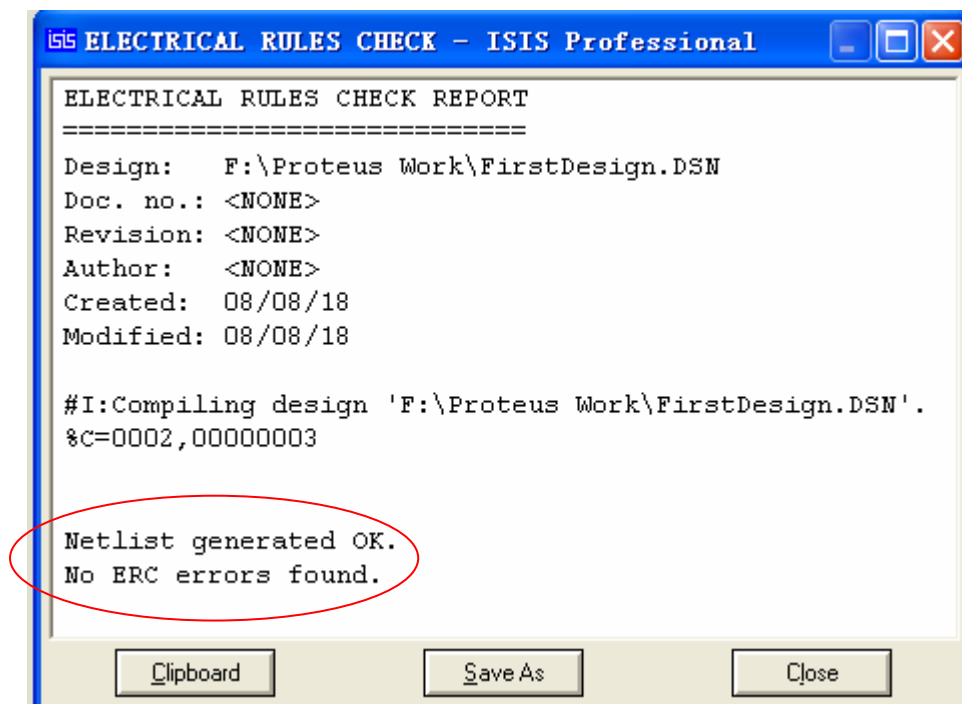


图 2-15

proteus 常用快捷键

F8: 全部显示 当前工作区全部显示

F6: 放大 以鼠标为中心放大

F7: 缩小 以鼠标为中心缩小

G: 栅格开关 栅格网格

Ctrl+F1: 栅格宽度 0.1mm 显示栅格为 0.1mm, 在 pcb 的时候很有用

F2: 栅格为 0.5mm 显示栅格为 0.5mm, 在 pcb 的时候很有用

F3: 栅格为 1mm 显示栅格为 1mm, 在 pcb 的时候很有用

F4: 栅格为 2.5mm 显示栅格为 2.5mm, 在 pcb 的时候很有用

Ctrl+s: 打开关闭磁吸 磁吸用于对准一些点的, 如引脚等等

x: 打开关闭定位坐标 显示一个大十字射线

m: 显示单位切换 mm 和 th 之间的单位切换, 在右下角显示

o: 重新设置原点 将鼠标指向的点设为原点

u: 撤销键

Pgdn: 改变图层

Pgup: 改变图层

Ctrl+Pgdn: 最底层

Ctrl+pgup: 最顶层

Ctrl+画线: 可以划曲线

R: 刷新

+ -: 旋转


F5: 重定位中心

3、KeilC 与 Proteus 连接调试

- 1) 安装Keil联调proteus驱动, 运行vdmagdi.exe文件(已经安装好)。
- 2) 进入KeilC μ Vision2开发集成环境, 创建一个新项目(Project), 并为该项目选定合适的单片机CPU器件(如: Atmel公司的AT89C51)。并为该项目加入源程序。

源程序如下:

```
ORG 0000H
SJMP 0030H
ORG 0030H
MAIN: MOV P1,#0
      MOV P0,#0FFH
LP:   JB P0.0,$
      JNB P0.0,$
      CPL P1.0
      SJMP LP
END
```

- 3) 单击“Project菜单/Options for Target”选项或者点击工具栏的“option for target”按钮, 弹出窗口, 点击“Debug”按钮, 出现如图2-16所示页面。

在出现的对话框里在右栏上部的下拉菜单里选中“Proteus VSM Monitor—51 Driver”。并且还要点击一下“Use”前面表明选中的小圆点。

再点击“Setting”按钮，出现如图2-17，确认并设置通信接口，在“Host”后面添上“127.0.0.1”，如果使用的不是同一台电脑，则需要在这里添上另一台电脑的IP地址(另一台电脑也应安装Proteus)。在“Port”后面添加“8000”。设置好的情形如图所示，点击“OK”按钮即可。最后将工程编译，进入调试状态，并运行。

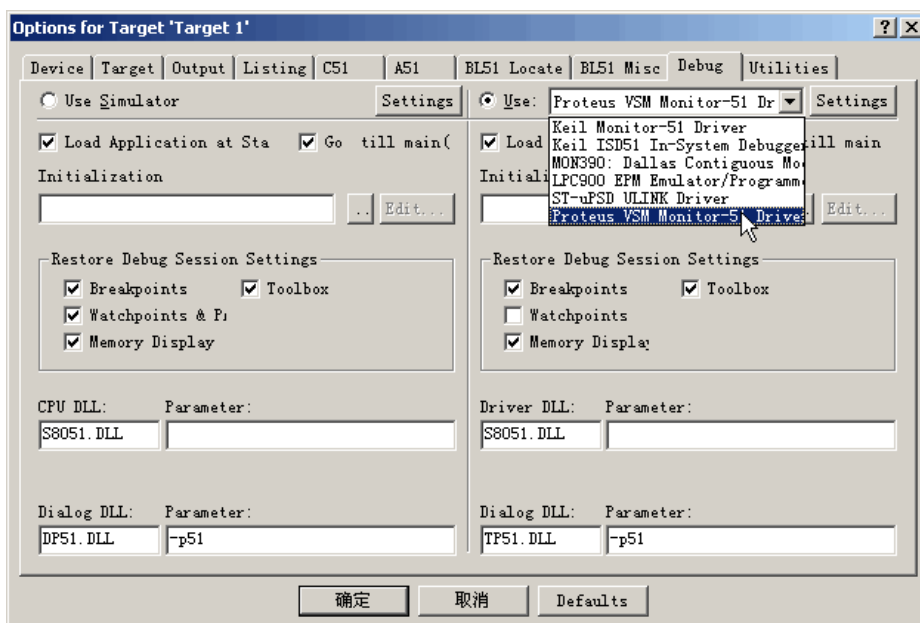


图2-16

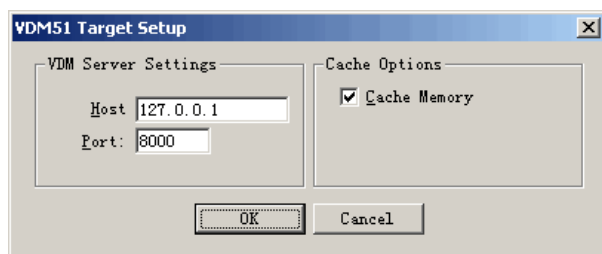


图2-17

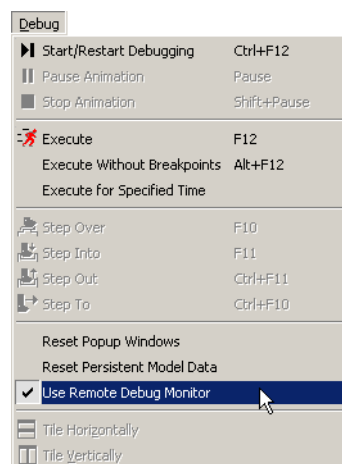
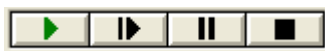


图2-18

4) Proteus的设置

进入Proteus的ISIS，鼠标左键点击菜单“Debug”，选中“use remote debug monitor”，如图2-18所示。此后，便可实现KeilC与Proteus连接调试。

5) KeilC与Proteus连接仿真调试

在 KeilC 中，单击全速运行，则 Proteus ISIS 中的仿真进程控制按钮  已经处于全速运行状态。原理图变成如图 2-19 所示，我们能清楚地观察到每一个引脚的电频变化，红色代表高电平，蓝色代表低电平，灰色代表悬空，如果连接线出现交叉等错误情况，就有可能出现黄色。图中 LED 灯是点亮状态。

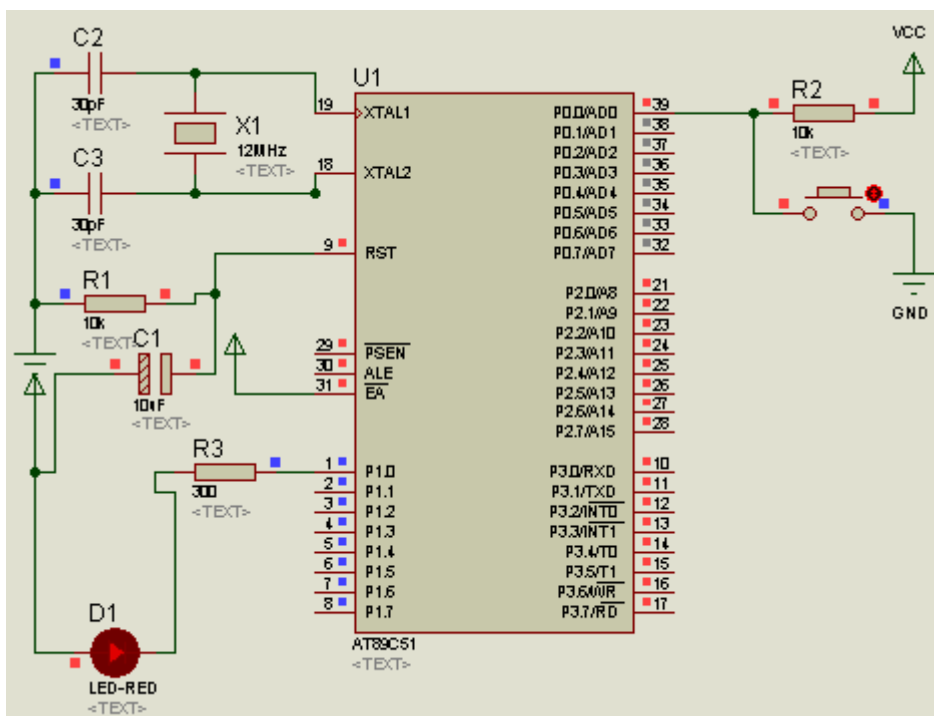



图 2-19

当我们单击一次按键，可以发现红色 LED 灯灭了，如图 2-20 所示。单击



上的  键，即可退出仿真状态。或者在 KeilC 下退出调试状态，亦可在 Proteus ISIS 中退出仿真状态。

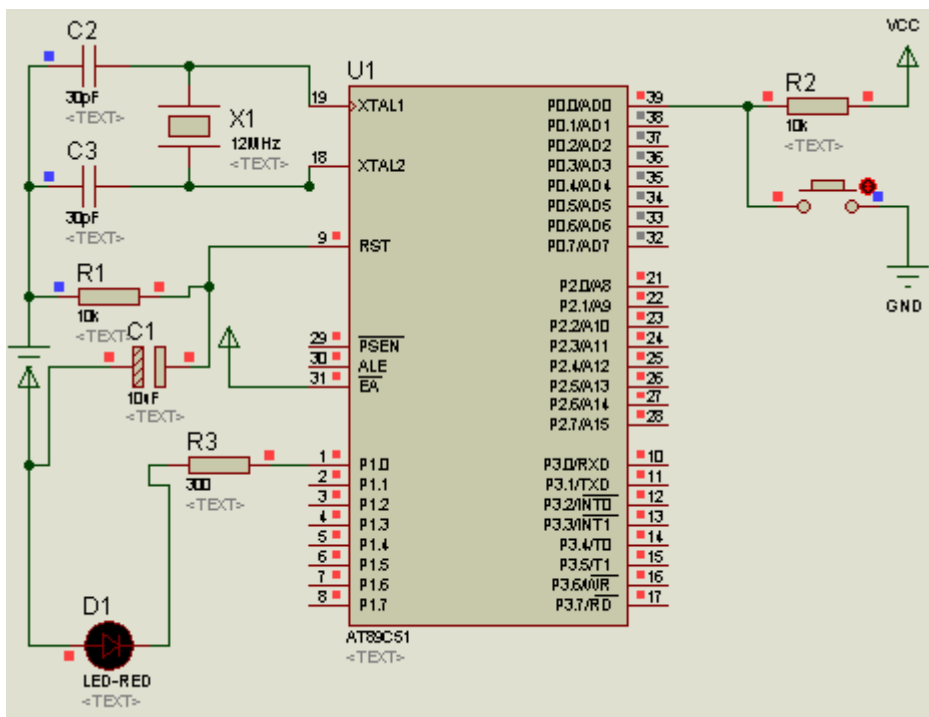



图 2-20

6) Proteus 独立仿真。如果单片机的程序已经编完测试通过，并以生成 hex 文件，就可以在 Proteus ISIS 下独立执行，观看运行效果。方法如下，左键双击单片机 AT89C51，出现该器件的属性窗口，在“Program File”一栏中，输入适用于该系统的 hex 文件。如图 2-21 所示。

然后，单击  键，单片机系统就开始处于运行状态，本例中，可以单击按键控制 LED 灯的亮灭。

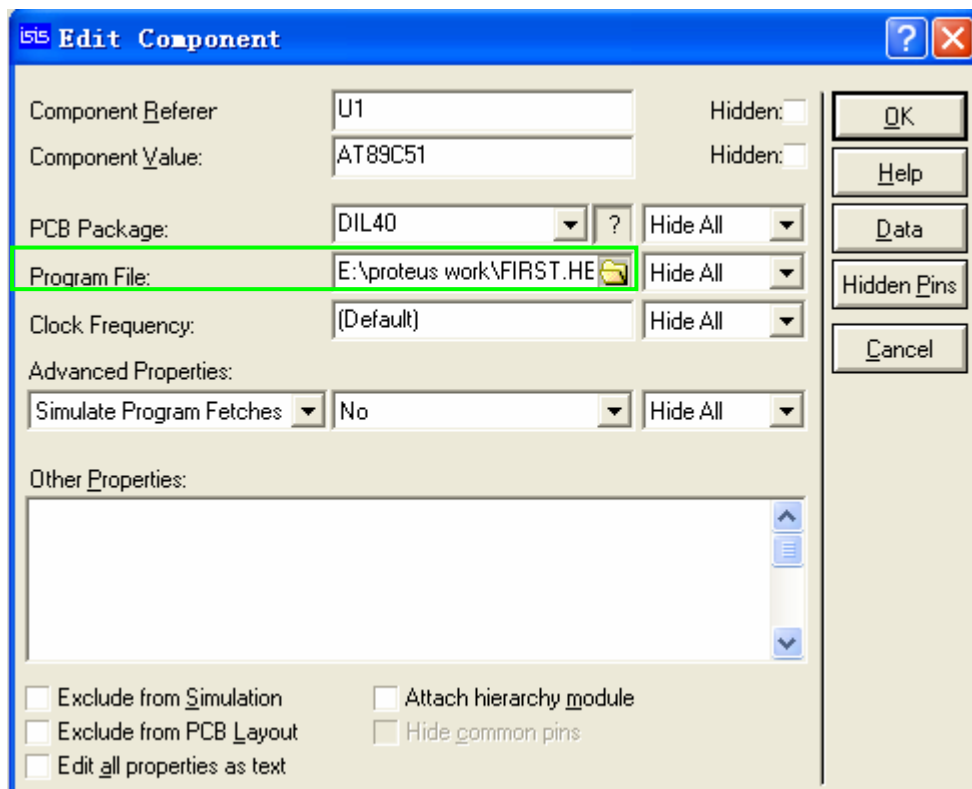


图 2-21

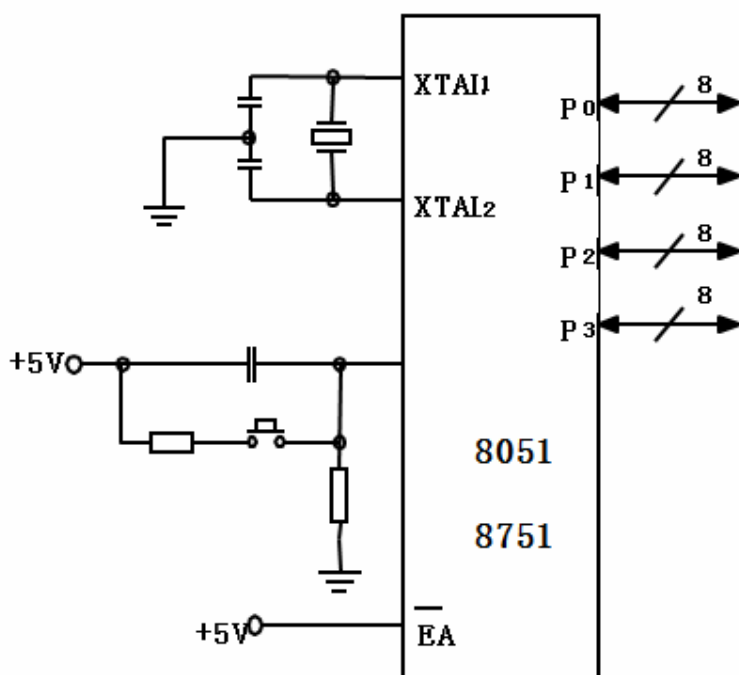
实验 2-1 发光二极管流水灯

【背景知识】

硬件实验背景知识的特点是软硬件结合，针对该次实验所用到的基础硬件原理和软件编写方法都要给出。

本实验涉及到三个知识点：单片机最小系统的构成、单片机 I/O 口的使用以及软件延时程序的编写。

1) 单片机最小系统由单片机芯片、时钟电路以及复位电路构成。



8051 最小应用系统

2) I/O 口线

P0 口：8 位双向 I/O 口。在访问外部存储器时，P0 口可用于分时传送低 8 位地址总线和 8 位数据总线。能驱动 8 个 LSTTL 门。

P1 口：8 位准双向 I/O 口（“准双向”是指该口内部有固定的上拉电阻）。能驱动 4 个 LSTTL 门。

P2 口：8 位准双向 I/O 口。在访问外部存储器时，P2 口可用于高 8 位地址总线。能驱动 4 个 LSTTL 门。

P3 口：8 位准双向 I/O 口。能驱动 4 个 LSTTL 门。P3 口还有第二功能。

P1 口作为输出口时与一般的双向口使用方法相同，即当 P1 口用为输入口时，必须先对它置“1”。若不先对它置“1”，读入的数据可能是不正确的。

3) 延时子程序的延时计算问题。例如：

```
DELAY:  MOV    R5, #20
DEL1:   MOV    R6, #100
DEL2:   MOV    R7, #250
        DJNZ   R7, $
        DJNZ   R6, DEL2
        DJNZ   R5, DEL1
        RET
```

>

【实验内容】

单片机具有基本的输入输出功能。通过 LED 可以直观的看到单片机输出的效果。在实际应用中，常常使用一、两个 LED 的输出，来表示程序运行的状态，便于判断系统是否运行正常。本次实验是使用单片机驱动 8 个 LED，实现左移、右移、闪烁等功能，一方面需要我们熟悉单片机的指令系统，另一方面，也可以让我们对单片机的应用有一个直观的了解。

【实验目的】

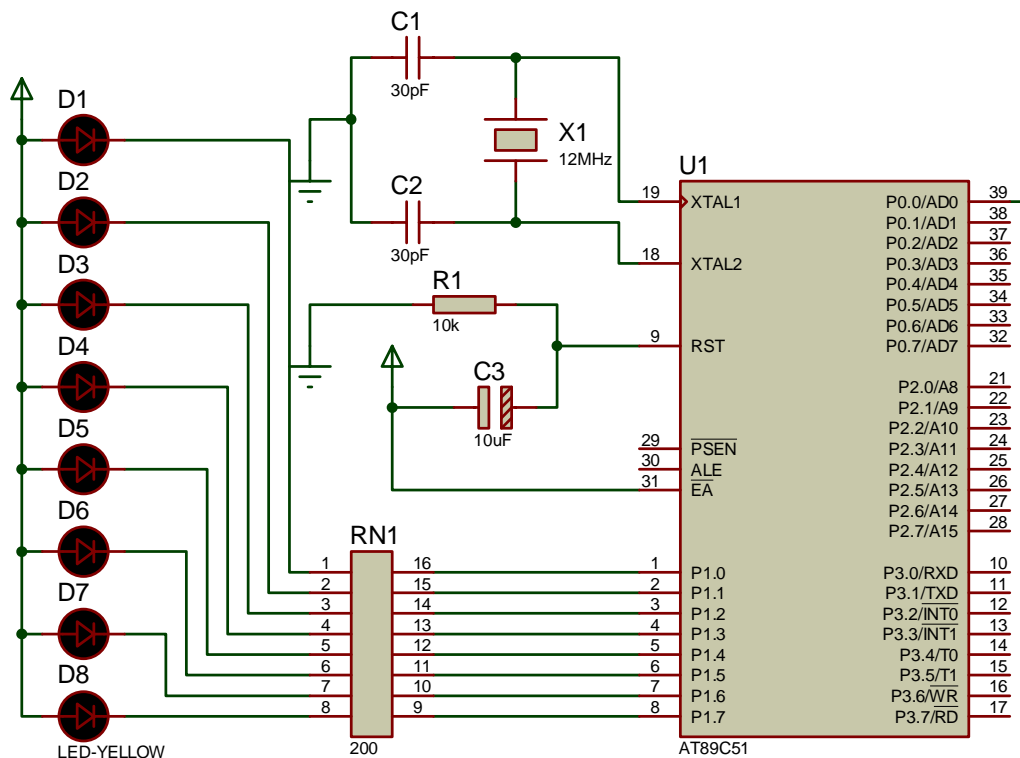
- 1、掌握 AT89C51 单片机 IO 口的输入输出。
- 2、掌握用查表方式实现 AT89C51 单片机 IO 口的控制。
- 3、熟练运用 Proteus 设计、仿真 AT89c51 系统。
- 4、掌握发光二极管的控制方法。

【实验要求】

- 1、通过 AT89C51 单片机控制 8 个发光二极管发光，实现暗点以大约 1Hz 频率由上到下循环移动。
- 2、通过 AT89C51 单片机控制 8 个发光二极管发光，循环实现由上到下移动 1 次，由下到上移动 1 次，闪烁 1 次（延时时间 0.2S）。
- 3、通过 AT89C51 单片机控制 8 个发光二极管发光，循环实现由上到下移动 2 次，由下到上移动 2 次，闪烁 2 次（延时时间 0.2S）。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，发光二极管流水灯实验装置电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
RX8（8 排阻）	Resistors	Resistor Packs
LED—YELLOW（黄色发光二极管）	Optoelectronics	LEDs
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--

- 2、放置元器件；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；

6、电气检查。

【源程序设计】

- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

实验 2-2 按键控制发光二极管发光

【背景知识】

略

【实验内容】

作为实际的单片机应用系统，基本的输入功能是不可或缺的。按键就是最常用的输入方式。操作人员通过按键或键盘输入数据或命令，实现简单的人机通信。本次实验，通过简单的按键来控制 LED 的亮灭，使同学们对按键的使用有一个基本的了解。

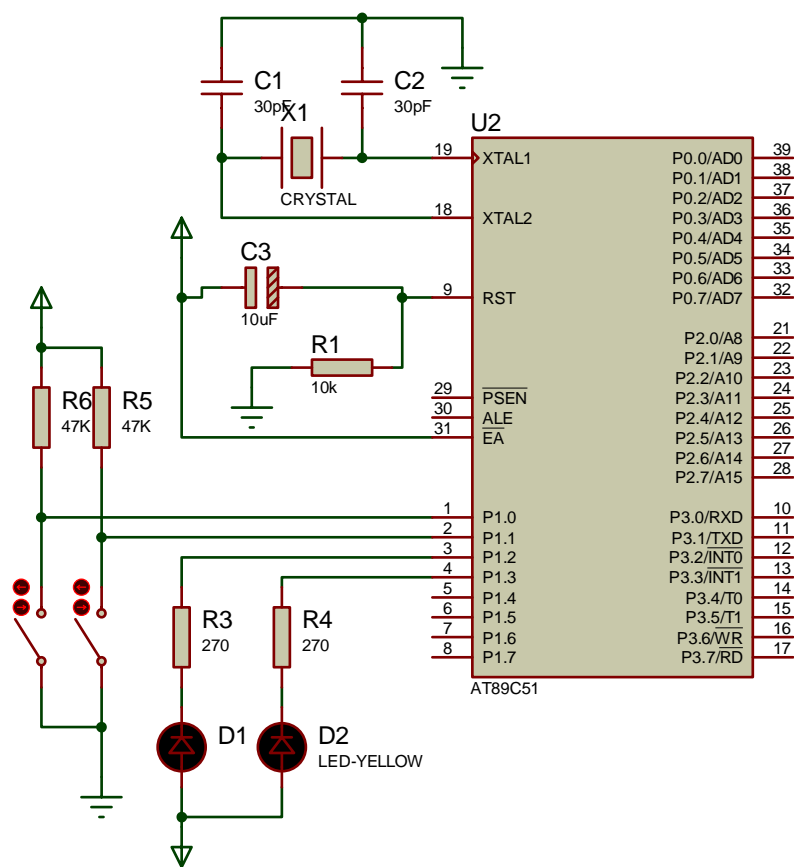
【实验目的】

- 1、掌握 AT89C51 单片机按键的设计和编程。
- 2、掌握 AT89C51 单片机按键控制任务的实现。
- 3、熟练运用 Proteus 设计、仿真 AT89c51 系统。
- 4、掌握按键控制发光二极管的方法。

【实验要求】

- 1、通过 AT89C51 单片机用按键实现发光二极管的同时亮灭、交替亮灭；
- 2、通过 AT89C51 单片机用按键实现发光二极管的同时闪烁、交替闪烁；
- 3、通过 AT89C51 单片机用一个按键控制 D1 的亮灭；用另一个按键控制 D2 的闪烁和熄灭。

【PROTEUS 电路设计】



电路原理图

在 ISIS 中进行电路图设计，本实验装置电路原理图如上图所示。

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
SWITCH（按键）	Switch&relays	Switches
LED-YELLOW（黄色发光二极管）	Optoelectronics	LEDs
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic

CRYSTAL（晶振）	Miscellaneous	--
-------------	---------------	----

- 2、放置元器件；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

实验 2-3 定时器应用（一）

【背景知识】

一、定时器/计数器的结构和功能

8051 单片机内有两个可编程 16 位定时器/计数器，常称为定时器 0 和定时器 1，简称为 T0 和 T1。都是 16 位加法计数器的结构，即 16 位定时器/计数器实质上是一个加法计数器。因此既可作定时器也可作计数器。常用于定时控制、延时、外部计数和检测等。

结构功能：

定时：对内部机器周期脉冲进行计数

计数脉冲来自单片机的内部。即每个机器周期产生一个计数脉冲使得计数器加 1，直至计满溢出。一个机器周期 = $12 \times$ 振荡周期，从开始计数到溢出这段时间就是“定时”时间。因此，若机器周期一定，计数初值越大，则定时越短。

b.计数：对外来脉冲进行计数。

计数脉冲来自单片机的外部的 T0(P3.4)和 T1(P3.5)两个引脚。外部输入的脉冲在出现从 1 到 0 的负跳变时有效，计数器进行加 1。计数方式下，单片机在每个机器周期的 S5P2 拍节时对外部计数脉冲进行采样。如果前一个机器周期采样为高电平，后一个机器周期采样为低电平，即为一个有效的计数脉冲。在下一机器周期的 S3P1 进行计数。即采样计数脉冲是在二个机器周期进行的。因此，计数脉冲的频率不能高于振荡脉冲频率的 1/24。

二、方式寄存器 TMOD 和控制寄存器 TCON

1.控制寄存器 TCON：（可以进行位寻址）

	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
TCON	TF ₁	TR ₁	TF ₀	TR ₀	IE ₁	IT	IE ₀	IT ₀

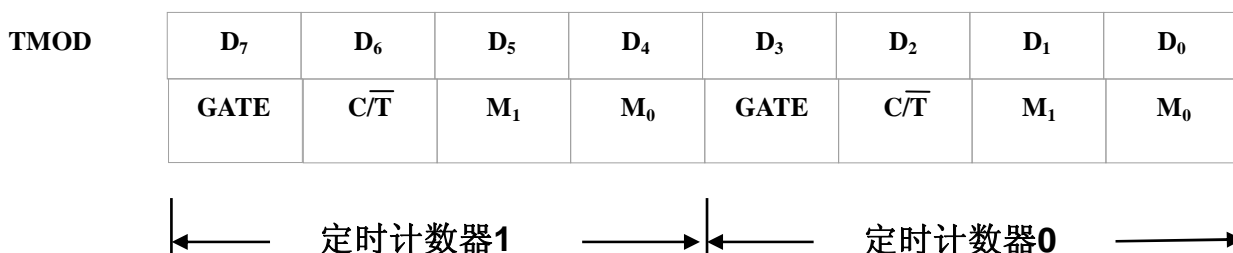
TF1（TCON.7, 8FH 位）----T1 溢出标志位。

TF0（TCON.5, 8DH 位）----T0 溢出标志位。

TR1（TCON.6, 8EH 位）----T1 运行控制位。0：关闭 T1；1：启动 T1 运行。

TR0（TCON.4, 8CH 位）----T0 运行控制位。0：关闭 T0；1：启动 T0 运行。

2. 方式寄存器 TMOD : (不能进行位寻址, 没有位地址)



C/T---定时器/计数器方式 选择位。0: 定时器; 1: 计数器。

GATE---外部门控位。

0: 不用外部门, 只将 TR0/TR1 置 1 来启动定时器;

1: 使用外部门, 外部请求信号 INT0/INT1 (高电平) 和 TR0/TR1 (置 1) 共同来启动定时器。

M1 M0 : 工作方式选择。 00: 方式 0 ; 01: 方式 1; 10: 方式 2 ; 11: 方式 3.

方式 0: 13 位计数器;

方式 1: 16 位计数器;

方式 2: 自动再装入 8 位计数器;

方式 3: 定时器 0: 分成 1 个 8 位的定时计数器和 1 个 8 位的计数器 (占用定时器 1 的 TR1 和 TF1); 定时器 1: 无此工作方式, 但可以停止其在方式 0、1、2 下的定时工作。

【实验内容】

定时是单片机系统非常重要且常用的功能，AT89C51 单片机具有较丰富的定时功能。本次实验是通过定时器实现 LED 的闪烁、亮灭，使同学们熟练掌握定时器的应用。

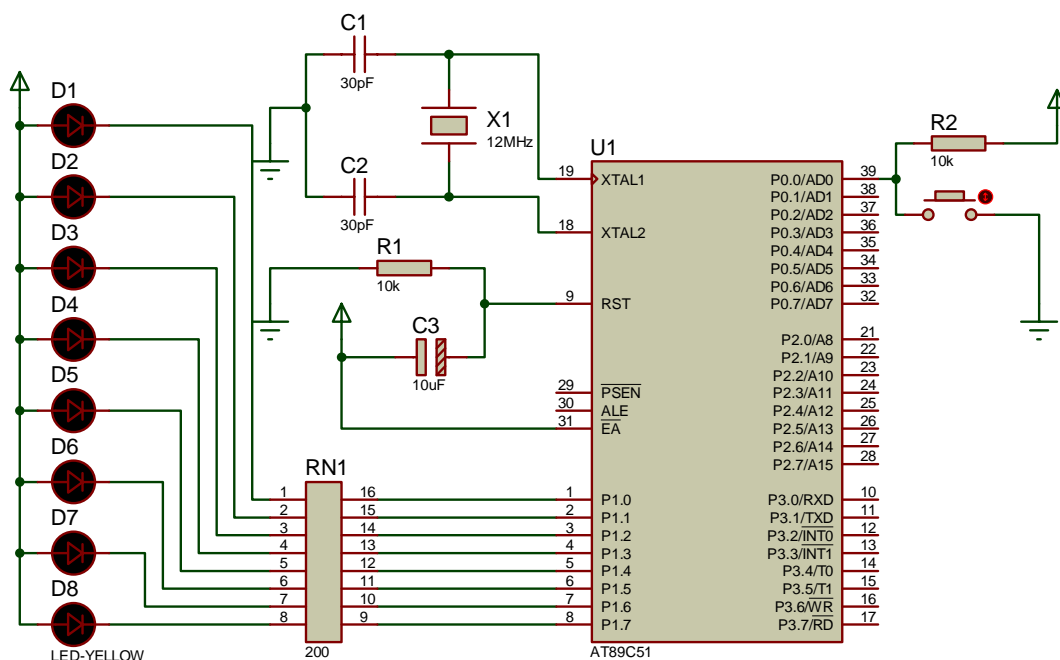
【实验目的】

- 1、理解 MCS—51 系列单片机定时器和计数器的概念和区别。
- 2、掌握和熟悉单片机定时计数器的初始化编程。
- 3、初步学会运用定时计数器进行单片机控制程序设计。

【实验要求】

- 1、通过 AT89C51 单片机用软件定时实现发光二极管的同时亮灭、交替亮灭；
- 2、通过 AT89C51 单片机用软件定时实现发光二极管的同时闪烁、交替闪烁；
- 3、通过 AT89C51 单片机用软件定时实现发光二极管的秒闪；
- 4、对 AT89C51 单片机的一个按键开关进行计数，计数十次，D1 的闪烁 1 次；计数 20 次，D2 循环闪烁，计数 30 次，D1、D2 熄灭，以此循环。

【PROTEUS 电路设计】



电路原理图

在

ISI
S
中进行电路图设

计，本实验装置电路原理图如下图所示。

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
BUTTON	Switches & Relays	Switches
RX8（8 排阻）	Resistors	Resistor Packs
LED—YELLOW（黄色发光二极管）	Optoelectronics	LEDs
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--

- 2、放置元器件；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

实验 2-4 定时器应用（二）

【背景知识】

略

【实验内容】

定时是单片机系统非常重要且常用的功能，AT89C51 单片机具有较丰富的定时功能。本次实验是定时器应用的第二次实验，目的的加深同学们对定时器的应用的熟练程度，另一方面，通过定时器/计数器的计数功能，实现对外来脉冲的计数功能，全面掌握定时/计数器的功能开发。

【实验目的】

- 1、理解 MCS—51 系列单片机定时器和计数器的概念和区别。
- 2、掌握和熟悉单片机定时计数器的初始化编程。
- 3、初步学会运用定时计数器进行单片机控制程序设计。

【实验要求】

- 1、以下定时计数程序设计必须用**查询方式**实现。
- 2、单片机晶振频率设定为 6MHz。
- 3、完成以下要求，通过虚拟示波器观看波形。

A、使用定时器 1 以方式 0 产生周期为 500us 的等宽方波连续脉冲，并由 P1.0 输出。

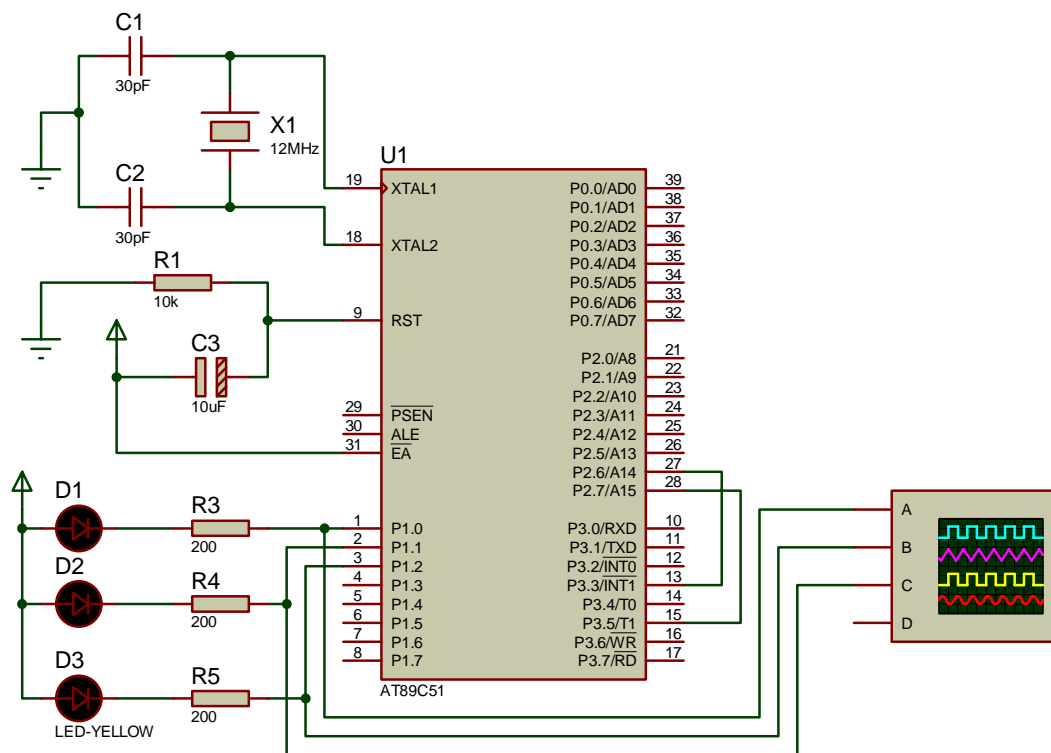
B、使用定时器 0 以方式 2 产生周期为 200us 的等宽方波连续脉冲，并由 P1.1 输出。

C、编程使 T0 工作于定时工作方式 0，产生 500us 等宽方波从 P2.7 输出。T1 工作于计数工作方式 2，其计数外部脉冲由 T0 产生，即 P2.7 引脚与 P3.5（T1 引脚）相连，T1 每计数 100 个，P1.2 取反输出一次。

D、编程使 T0 工作于定时工作方式 0，产生 500us 等宽方波从 P2.6 输出到 P3.3（INT1 脚），如图所示。T1 工作于定时工作方式 2，由 T1 来测量 P3.3 脚信号的正脉冲宽度，并通过示波器加以验证。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，本实验装置参考电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
RX8（8 排阻）	Resistors	Resistor Packs
LED—YELLOW（黄色发光二极管）	Optoelectronics	LEDs
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--

- 2、放置元器件；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

- 1、流程图；
- 2、在 KeilC 中进行源程序设计；
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

实验 2-5 定时器与中断的综合应用

【背景知识】

在计算机系统中，中断可以由各种硬件设备产生，以便请求服务或报告故障等。此外，中断也可由处理器自身产生，例如，程序错误或对操作系统的请求做出响应等。计算机的中断服务需求是以中断请求（Interrupt Request）的形式提出来的，不管是来自硬件的还是来自软件的中断请求，凡是中断请求的来源都统称为中断源。

80C51 的中断系统具有 5 个中断源，即 2 个外部中断、2 个定时器中断和 1 个串行中断。中断源及其对应的中断向量如下表所示：

80C51 的中断

中断名称	中断向量
外部中断 0	0030H
定时器 0 中断	000BH
外部中断	0013H
定时器 1 中断	001BH
串行发送/接收中断	0023H

1、当高、低优先级中断请求同时出现时，高优先级中断请求被响应。

2、如果统计的多个中断请求同时出现时，则按 CPU 查询次序确定哪个中断请求被响应。其查询次序为：外部中断 0→定时器 0→中断外部中断 1→定时器 1→中断串行中断。

由于串行中断的内容较为复杂，本文只较少外部中断和定时器中断。关于定时器中断的内容在上一小节的定时器实验中已经涉及到了，本次实验主要针对外部中断。中断存在可能只存在一个，也可能是多个同时存在。当多个中断同时存在的时候就要按照上面介绍的中断优先原则进行响应。下面就以实验为例介绍单片机的中断系统。

【实验内容】

定时器可以工作在两种模式下，查询方式和中断方式。中断方式下定时器的使用有着非常好的优势，在实际应用中常常采用。本次实验对定时功能采用中断的方式实现，并部分重

复了前面的实验内容。目的只有一个：加深同学们对单片机系统定时器应用的熟练掌握程度。

【实验目的】

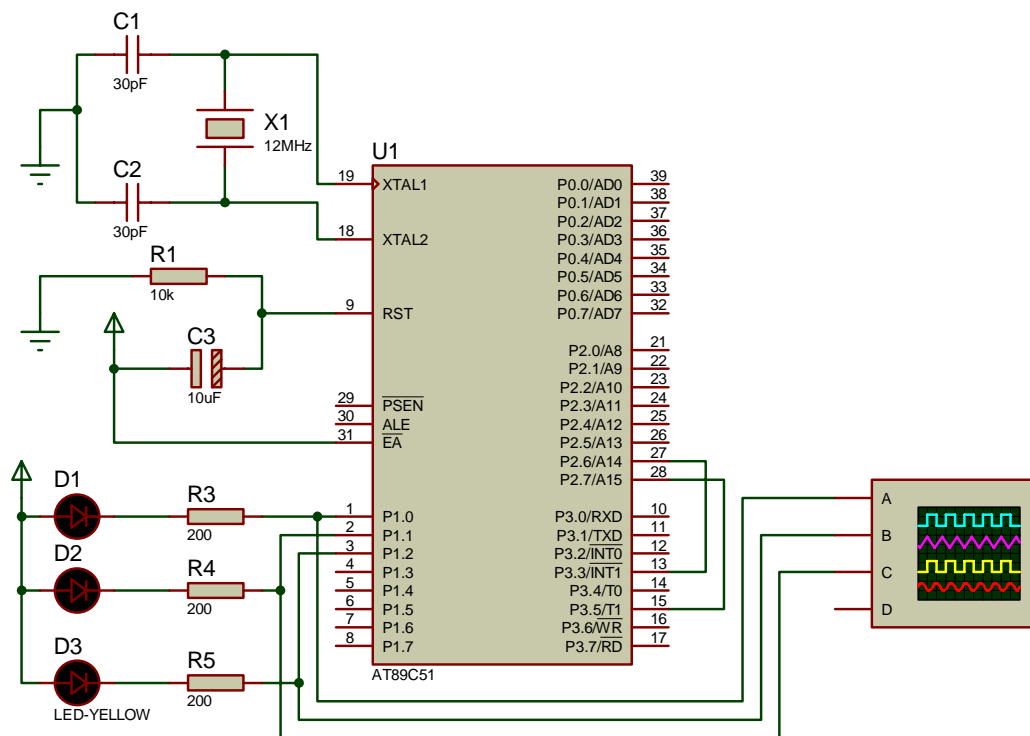
- 1、理解 MCS—51 系列单片机中断的概念。
- 2、掌握和熟悉单片机中断程序的初始化编程。
- 3、学会运用中断服务程序进行单片机控制程序设计。

【实验要求】

- 1、以下定时计数程序设计必须用**中断实现**。
- 2、单片机晶振频率设定为 6MHz。
- 3、完成以下要求，通过虚拟示波器观看波形。
 - A、使用定时器 1 以方式 0 产生周期为 500us 的等宽方波连续脉冲，并由 P1.0 输出。
 - B、使用定时器 0 以方式 2 产生周期为 200us 的等宽方波连续脉冲，并由 P1.1 输出。
 - C、编程使 T0 工作于定时工作方式 0，产生 500us 等宽方波从 P2.7 输出。T1 工作于计数工作方式 2，其计数外部脉冲由 T0 产生，即 P2.7 引脚与 P3.5（T1 引脚）相连，T1 每计数 100 个，P1.2 取反输出一次。
 - D、编程使 T0 工作于定时工作方式 0，产生 500us 等宽方波从 P2.6 输出到 P3.3（INT1 脚），如图所示。T1 工作于定时工作方式 2，由 T1 来测量 P3.3 脚信号的正脉冲宽度，并通过示波器加以验证。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，本实验装置电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
RX8（8 排阻）	Resistors	Resistor Packs
LED-YELLOW（黄色发光二极管）	Optoelectronics	LEDs
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--

2、放置元器件；

3、放置电源和地；

4、连线；

5、参照原理图进行元件属性设置；

6、电气检查。

【源程序设计】

1、流程图：

2、在 KeilC 中进行源程序设计：

3、编译、生成目标代码

【PROTUES 仿真】

1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；

2、仿真、调试代码

3、注意使用观察窗口

实验 2-6 外中断应用

【背景知识】

略

【实验内容】

MSC-51 是一个多中断源的单片机，有三类共五个中断源，分别是 2 个外部中断，2 个定时器中断和 1 个串行中断。外部中断是单片机应用系统经常使用的功能，本实验通过 P3.2 口连接一个按键，使用外部中断方式控制 LED 的亮灭等功能。同学们可以采用查询和中断两种方式来比较一下，采用查询和中断在编写程序上的不同以及优缺点。

【实验目的】

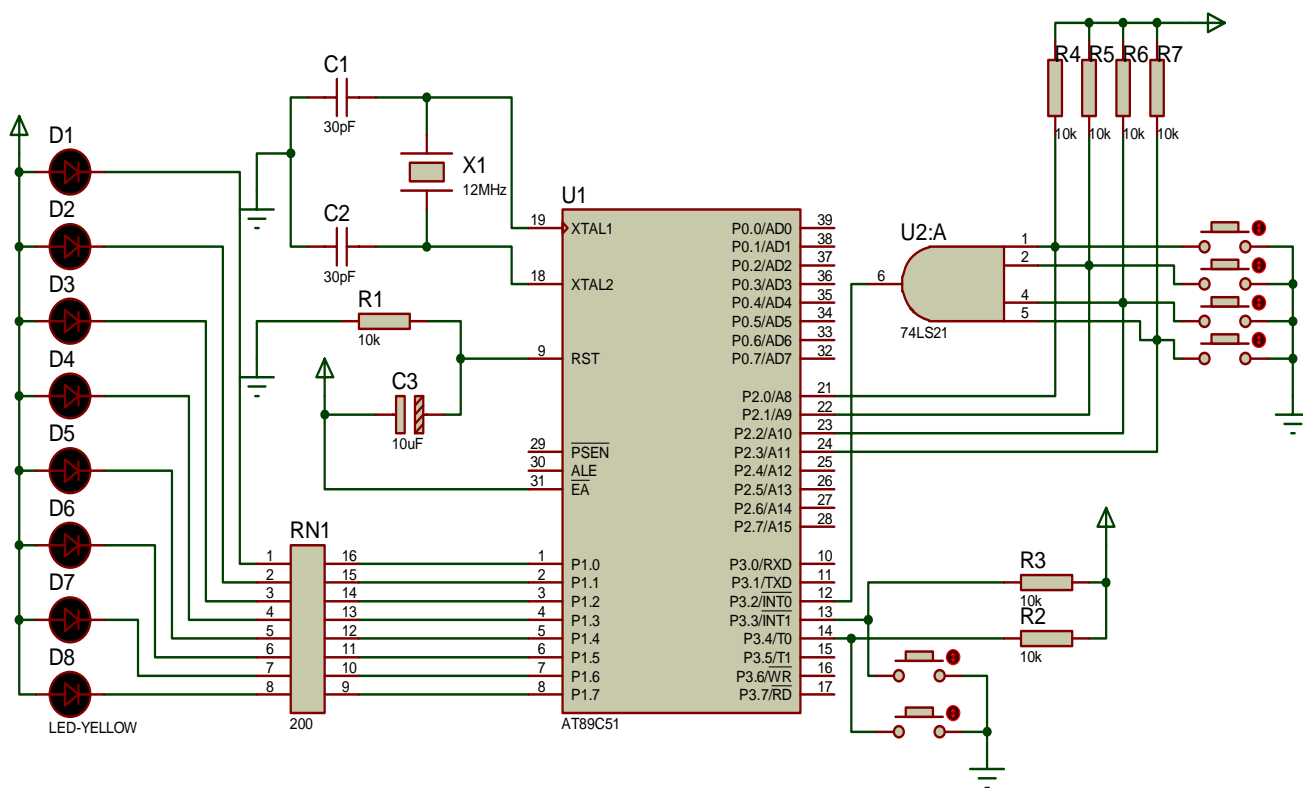
- 1、熟悉理解单片机 AT89C51 的中断系统组成；
- 2、了解掌握单片机系统中断的原理及使用方法。

【实验要求】

- 1、通过外部中断按键控制 LED 的亮灭；
- 2、通过外部按键（INT1 引脚链接）以中断方式控制 LED 的上移和下移。
- 3、通过外部按键（T0 引脚链接）以中断方式控制 LED 的上移和下移。
- 4、以查询方式重做 2。
- 5、以查询方式重做 3。
- 6、通过 INT0 引脚外界 4 输入与门扩展外部中断，编程实现扩展出的四个开关以中断方式，分别实现 LED 的上移、下移、亮和灭。
- 7、以查询方式重做 6。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，本实验装置电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT89C51 (单片机)	Microprocessor ICs	8051 Family
RES (电阻)	Resistors	Generic
CAP (电容)	Capacitors	Generic
CAP-ELEC (电解电容)	Capacitors	Generic
CRYSTAL (晶振)	Miscellaneous	--
LED-YELLOW (黄色发光二极管)	Optoelectronics	LEDs
BUTTON	Switches & Relays	Switches
74LS21	TTL 74LS series	Gates & Inverters

- 2、放置元器件，补全振荡电路，上电复位电路；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

实验 2-7 串口通信

【背景知识】

一、串口相关知识

8051 有一个可编程的全双工串行通信接口：

- 它可作 UART 用，
- 也可作同步移位寄存器，
- 其帧格式可有 8 位、10 位或 11 位，并能设置各种波特率。

1. 80C51 单片机的串行接口结构

80C51 单片机串行接口是一个可编程的全双工串行通信接口。它可用作异步通信方式（UART: universal asynchronous receiver and transmitter），与串行传送信息的外部设备相连接，或用于通过标准异步通信协议进行全双工的 8051 多机系统也能通过同步方式，使用 TTL 或 CMOS 移位寄存器来扩充 I/O 口。

8051 单片机通过管脚 RXD（P3.0，串行数据接收端）和管脚 TXD（P3.1，串行数据发送端）与外界通信。SBUF 是串行口缓冲寄存器，包括发送寄存器和接收寄存器。它们有相同名字和地址空间，但不会出现冲突，因为它们两个一个只能被 CPU 读出数据，一个只能被 CPU 写入数据。

2. 串行口的控制与状态寄存器

串行口通过两个特殊功能寄存器 SCON 和 PCON 来进行控制，分别介绍如下。

串行口控制寄存器 SCON（地址为 98H）：这个特殊功能寄存器包含串行口的工作方式选择位，接收发送控制位及串行口的状态标志，格式如下：

SCON 串行口标志位

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	SM3	SM4	SM5	SM6	SM7

SM2：多机通信控制位，1：多机通信；0：单机通信（方式 2、3）

REN：允许接收控制位，1：允许接收；0：禁止接收

TB8：发送的第 9 位数据（方式 2、3）。可为奇偶检验位（SM2=0），或多机通信特征位（1：表示地址；0：表示数据，SM2=1）

RB8: 接收到的第 9 位数据（方式 2、3）。

TI、RI: 分别表示发送和接收中断标志。无论查询还是中断方式都需软件编程清 0。

SM0 和 SM1 为串行口的工作方式选择位，详见下表：

串行口工作方式选择位

SM0	SM1	工作方式	说 明	波 特 率
0	0	方式0	同步移位寄存器	fosc/12
0	1	方式1	10位异步收发	由定时器1控制(可变)
1	0	方式2	11位异步收发	fosc/64 * 2 ^{SMOD}
1	1	方式3	11位异步收发	由定时器1控制(可变)

3. 串行口的工作方式

8051 单片机的全双工串行口可编程为 4 种工作方式，现分述如下：

方式 0 为移位寄存器输入/输出方式。可外接移位寄存器以扩展 I/O 口，也能外接同步输入/输出设备。8 位串行数据者是从 RXD 输入或输出，TXD 用来输出同步脉冲。输出串行数据从 RXD 管脚输出，TXD 管脚输出移位脉冲。CPU 将数据写入发送寄存器时，立即启动发送，将 8 位数据以 fosc/12 的固定波特率从 RXD 输出，低位在前，高位在后。发送完一帧数据后，发送中断标志 TI 由硬件置位。

输入当串行口以方式 0 接收时，先置位允许接收控制位 REN。此时，RXD 为串行数据输入端，TXD 仍为同步脉冲移位输出端。当 (RI) = 0 和 (REN) = 1 同时满足时，开始接收。当接收到第 8 位数据时，将数据移入接收寄存器，并由硬件置位 RI。

方式 1 为波特率可变的 10 位异步通信接口方式。发送或接收一帧信息，包括 1 个起始位 0，8 个数据位和 1 个停止位 1。输出当 CPU 执行一条指令将数据写入发送缓冲 SBUF 时，就启动发送。串行数据从 TXD 管脚输出，发送完一帧数据后，就由硬件置位 TI。输入在 (REN) = 1 时，串行口采样 RXD 管脚，当采样到 1 至 0 的跳变时，确认是开始位 0，就开始接收一帧数据。只有当 (RI) = 0 且停止位为 1 或者 (SM2) = 0 时，停止位才进入 RB8，8 位数据才能进入接收寄存器，并由硬件置位中断标志 RI；不然信息丢失。所以在方式 1 接收时，应先用软件清零 RI 和 SM2 标志。

方式 2 为固定波特率的 11 位 UART 方式。它比方式 1 增加了一位可编程为 1 或 0 的第 9 位数据。输出：发送的串行数据由 TXD 端输出一帧信息为 11 位，附加的第 9 位来自 SCON

寄存器的 TB8 位，用软件置位或复位。它可作为多机通信中地址/数据信息的标志位，也能作为数据的奇偶校验位。当 CPU 执行一条数据写入 SUBF 的指令时，就启动发送器发送。

发送一帧信息后，置位中断标志 TI。输入在 (REN)=1 时，串行口采样 RXD 管脚，当采样到 1 至 0 的跳变时，确认是开始位 0，就开始接收一帧数据。在接收到附加的第 9 位数据后，当 (RI)=0 或者 (SM2)=0 时，第 9 位数据才进入 RB8，8 位数据才能进入接收寄存器，并由硬件置位中断标志 RI；不然信息丢失。且不置位 RI。再过一位时间后，不管上述条件时否满足，接收电路即行复位，并重新检测 RXD 上从 1 到 0 的跳变。

方式 3 为波特率可变的 11 位 UART 方式。除波特率外，其余与方式 2 相同。

4. 波特率计算

方式 1、3 波特率 = $(2^{\text{SMOD}}/32) \times 1/\text{定时时间}$

$$\text{定时时间} = (2^n - X) \times T_{\text{机}} = (2^n - X) \times 12/f_{\text{osc}}$$

$$\text{因此有：方式 1、3 波特率} = \frac{2^{\text{smod}}}{32} \times \frac{f_{\text{osc}}}{12 \times (2^n - X)}$$

二、Proteus 下串口的使用

8051 通过引脚 RXD(P3.0, 串行数据接收端)和引脚 TXD(P3.1, 串行数据发送端)进行串口通信。51 系列单片机给出的信号为 TTL 电平：

- TTL 逻辑门输出高电平的允许范围为 2.4~5 V，其标称值为 3.6 V；
- 输出低电平的允许范围为 0~0.7 V，其标称值为 0.3 V。
- 0.7 V 与 2.4 V 之间的是非高非低的中间电平。

但是，标准串口通信采用标准的通信接口，本身需要具有一定的抗干扰能力，由于工业现场的情况往往很恶劣，因而要根据具体情况进行选择。通常有以下两种：

RS232C：一般场合，逻辑 1=-3~-15V，逻辑 0=+3~+15V。

RS422：共模信号比较强，接收 (V+) - (V-) ≥ 0.2V，表示信号“1”；

(V+) - (V-) ≤ -0.2V，表示信号“0”

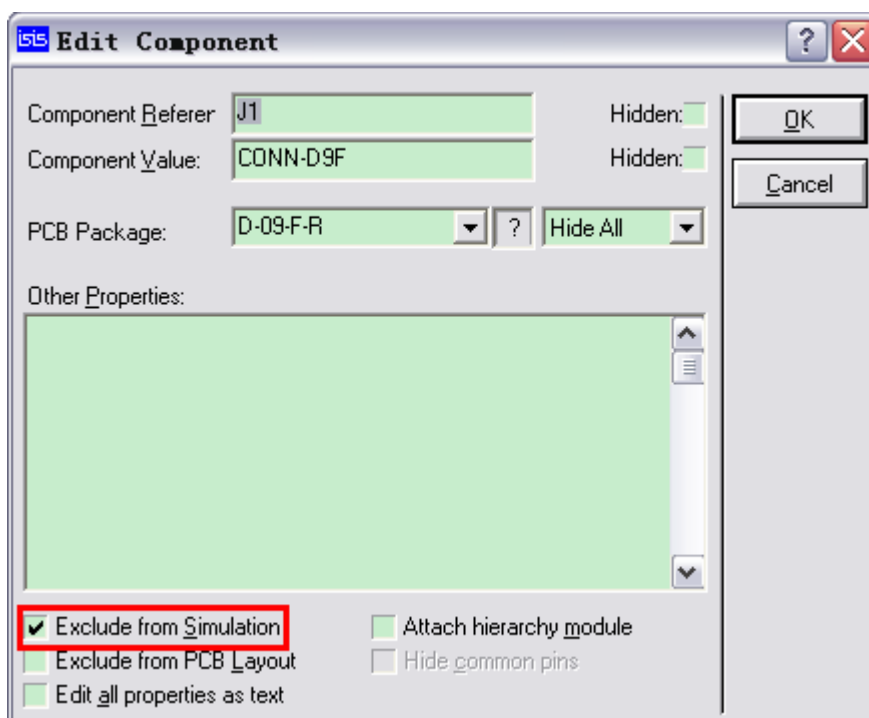
发送 (V+) - (V-) = 2~6V，表示信号“1”；

(V+) - (V-) = -2~-6V，表示信号“0”

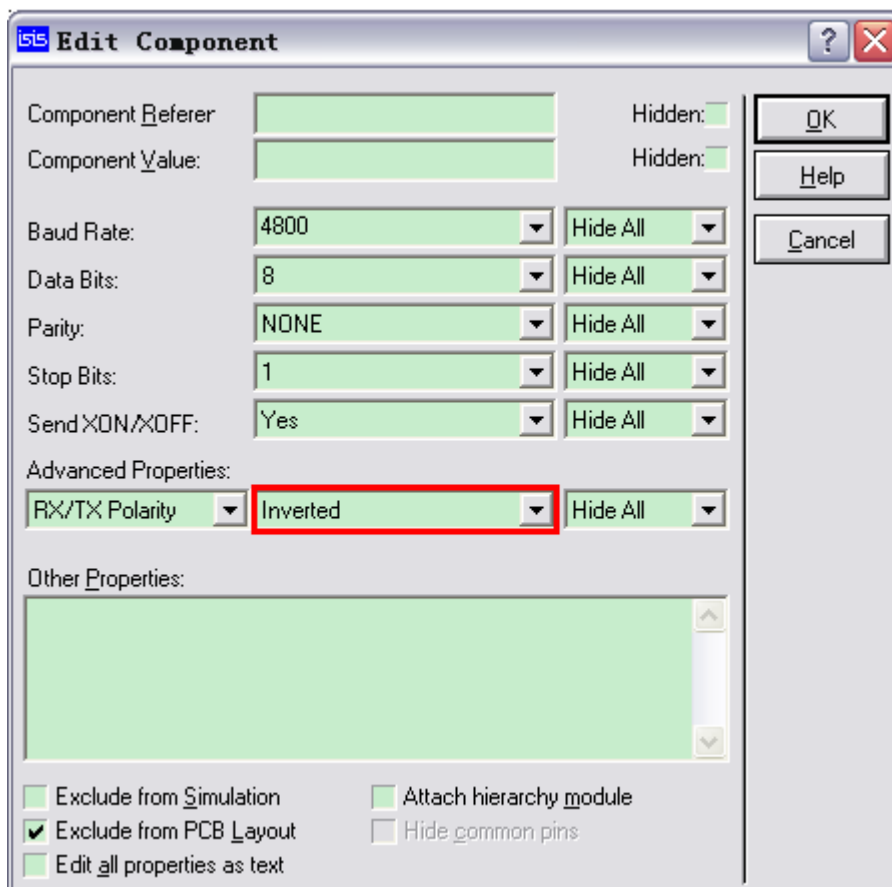
因此，在单片机串口不能直接和实际的通信串口直接相连，而是需要添加一个电平转换芯片。如本次实验“电路原理图”中给出的“MAX232”，就是进行 TTL 电平和 RS232 电平之间的转换。

本次实验中，原理图中除了单片机以外，还有 MAX232、虚拟终端以及串口接口 J1。

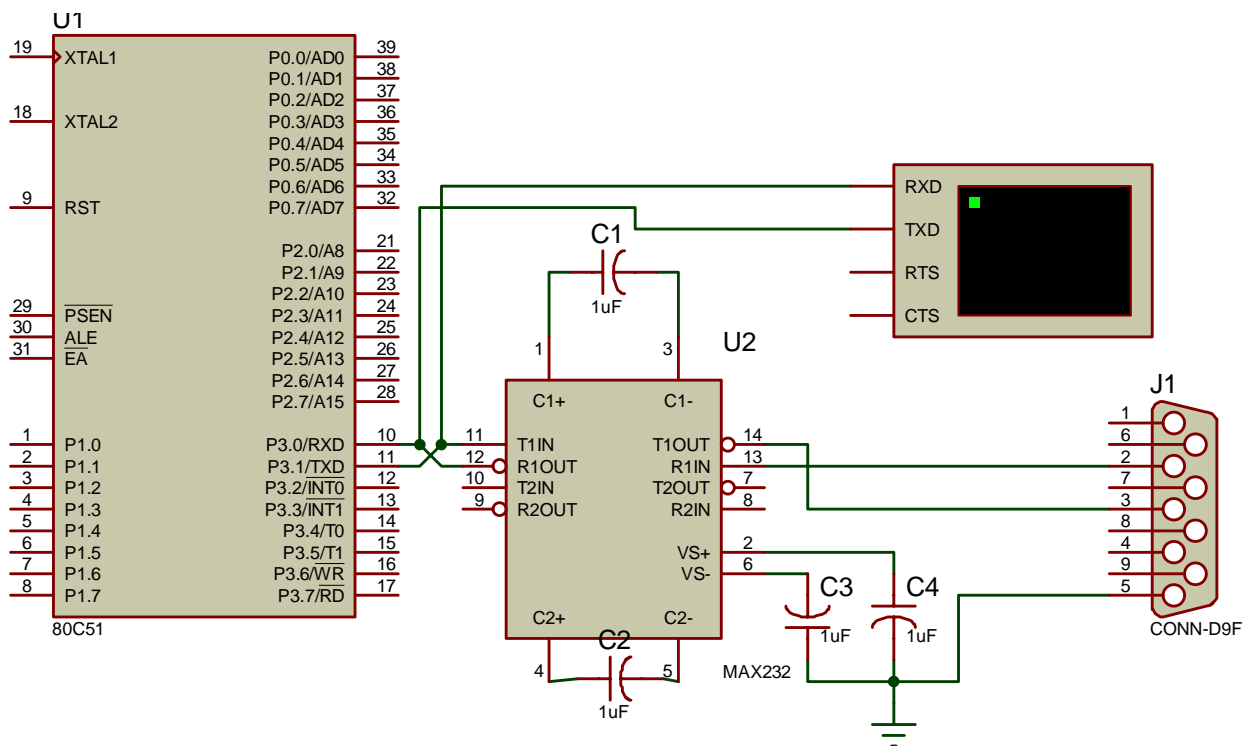
- 串口接口 J1 是仿真与计算机串口连接，由于本次实验不涉及 PC 的串口通信，因此在 J1 的属性中，将“Exclude from Simulation”勾选中，其含义是将此芯片在仿真中的功能禁止掉。

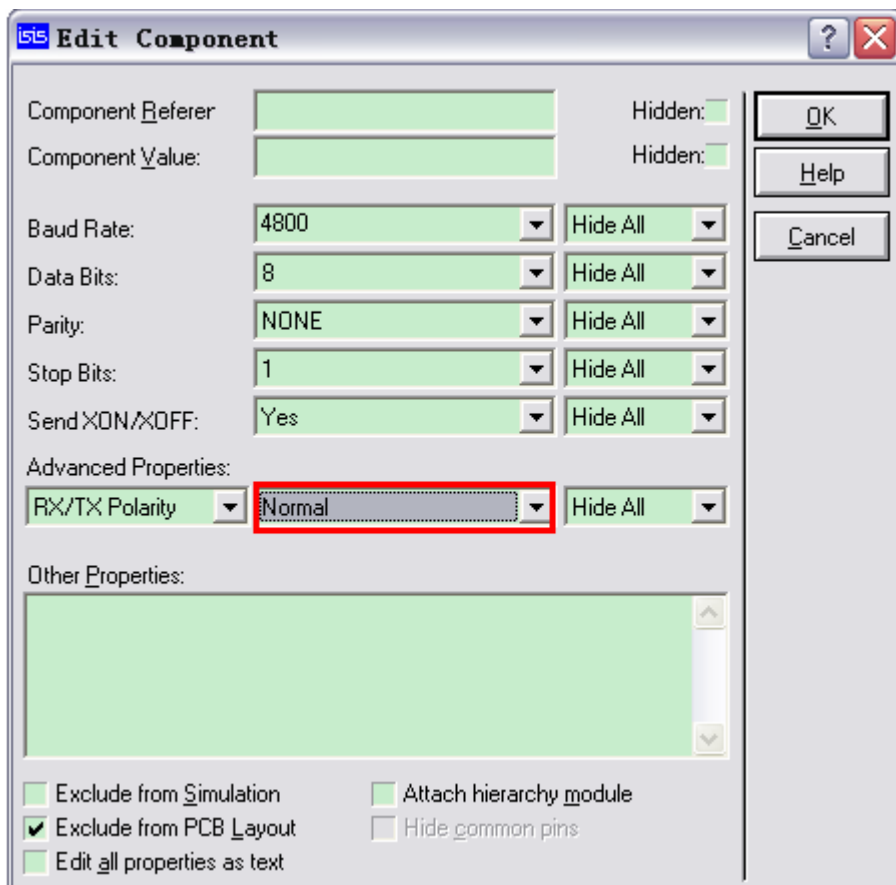


- 如果是按本次实验所给的连接方式，就需要在虚拟终端的属性中，将“RX/TX Polarity”的电平选择为“Inverted”。

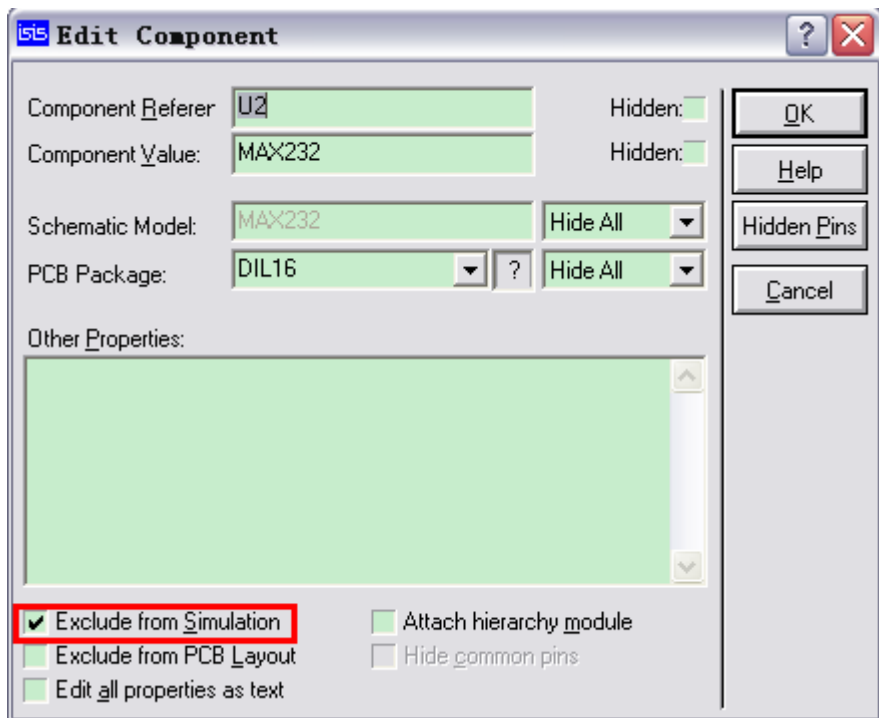


- 如果想要将单片机的串口与虚拟终端直接相连，如下图所示，就需要在虚拟终端的属性中，将“RX/TX Polarity”的电平选择为“Normal”。





另外，还需要在“MAX232”器件的属性中，将“Exclude from Simulation”勾选中，其含义是将此芯片在仿真中的功能禁止掉。



这样，串口就可以工作了。

【实验内容】

串口通信是现在仍在广泛应用的数据通信方式，因为其易于实现，比较稳定可靠。本次内容利用虚拟终端仿真单片机与 PC 机之间的串行通信。PC 机发送从键盘输入的数据，单片机接收后回发给 PC 机。同学们还可以扩展数码管的输出显示，将接收到的数据，显示到数码管上。另外，本次实验要求实现 2 两个单片机之间的串口通信，这部分内容要求同学们独立完成电路设计和代码的实现。

【实验目的】

- 1、理解 MCS—51 系列单片机中断的概念；
- 2、掌握和熟悉单片机中断程序的初始化编程；
- 3、学会运用中断服务程序进行单片机控制程序设计；
- 4、学会使用虚拟终端。

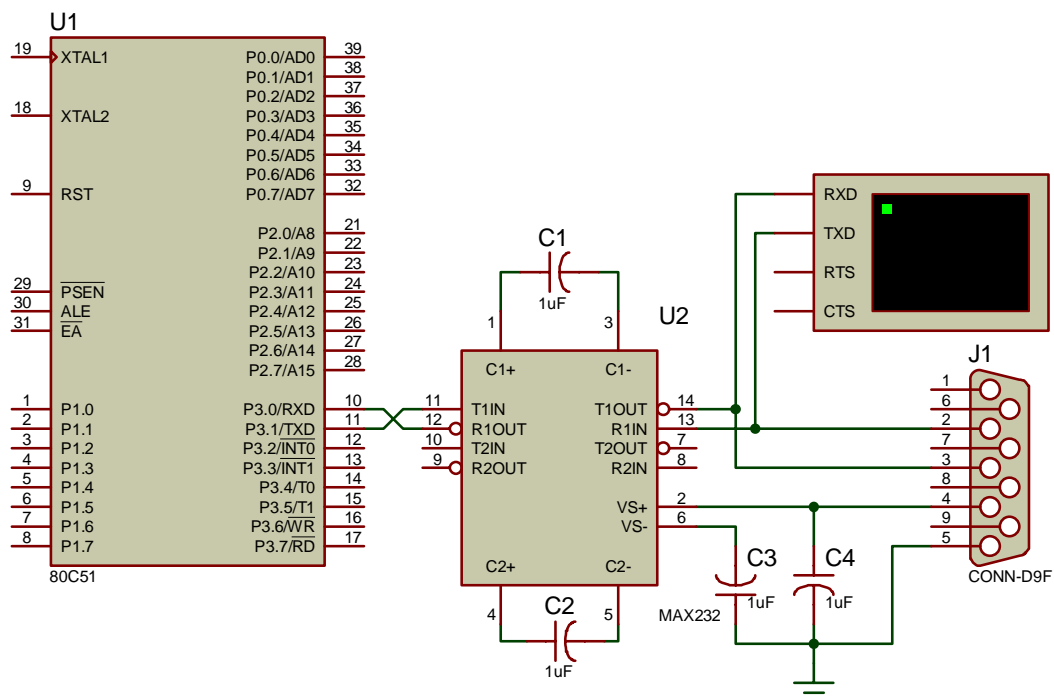
【实验要求】

- 1、认真阅读“【背景知识】”的“二、Proteus 下串口的使用”部分。
- 2、假定甲乙机以串行通信方式 1 进行数据通信，其波特率为 1200。由甲机发送数据，发送数据在 ROM 300H~31FH 单元中，但先发送数据块首末地址。由乙机接收，接收到的数据依次存入内部 RAM 的 50H，假设晶振 6MHz。

注意：请完善原理图，进行甲乙机的通信，切勿将发送和接收的连接弄错。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，本实验装置电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件, 进行第 2、3、4、5、6 步, 完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--
CONN-D9F（串口接口）	Connectors	D-Type
MAX232（串口电平转换芯片）	Microprocessor ICs	Peripherals

2、放置元器件，补全振荡电路，上电复位电路；

3、放置电源和地;

4、连线；

5、参照原理图进行元件属性设置:

6、电气检查。

【源程序设计】

- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】



- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

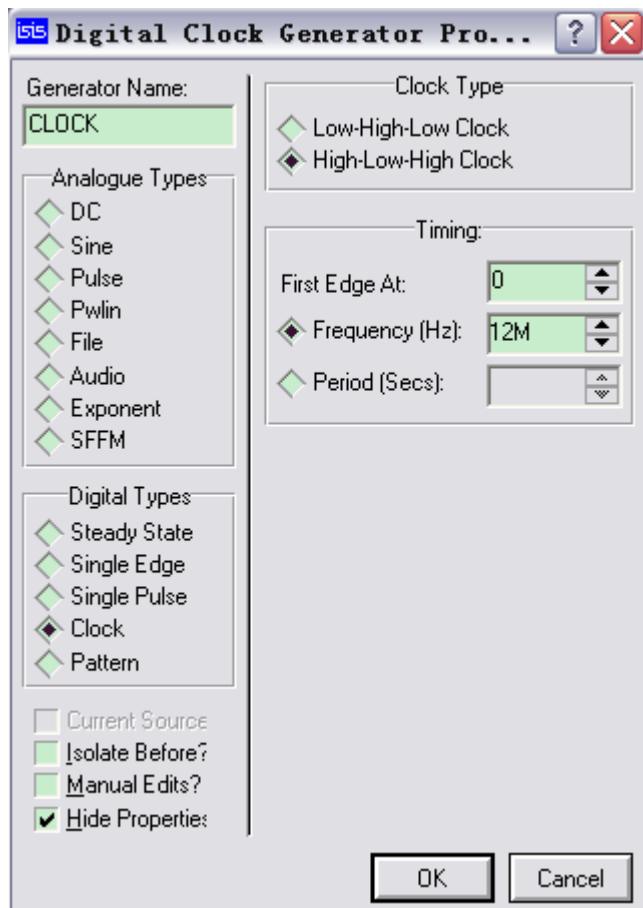
实验 2-8 外部数据存储器扩展


【背景知识】

一、Proteus 的波形图和信号源。


波形图不在仿真时，将程序的时序图显示出来，便于观察和打印。信号源可以产生不同频率、占空比的多种波形。

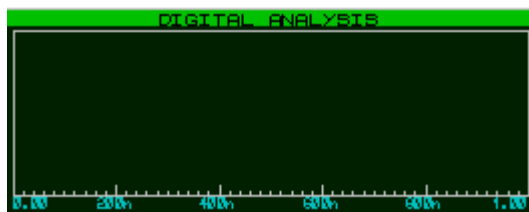
1、点击左侧工具栏中的，从右边的“GENERATORS”选中“DCLOCK”。单击图纸，出现控件，双击出现其属性页面，如下图，进行设置，将该信号源与单片机“X1”引脚相连。



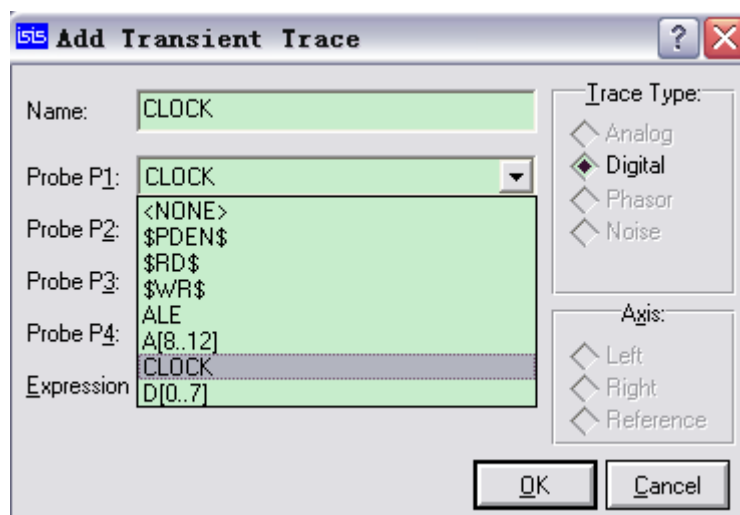
2、点击左侧工具栏中的，再左键单击想要检测的信号线，这时会有一个探针出现在信号线上。仿真时，该探针可以实时给出该信号线上的电压值。按“【PROTEUS 电路设计】”

所给原理图的要求施放探针，如需要，给探针命名。

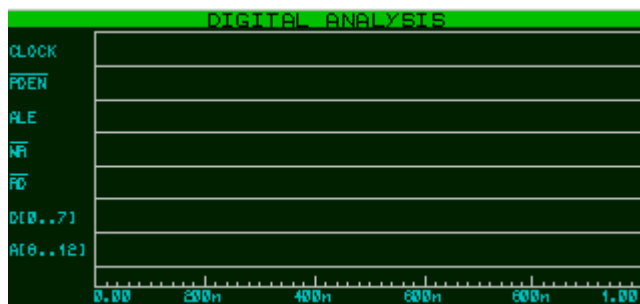
3、点击左侧工具栏中的，从右边的“GRAPHS”选中“DIGITAL（数字波形）”或者“MIXED(数字模拟混合波形)”。然后在图纸上空白处左键单击一点，并按住不放，拖出一个矩形窗口后，再释放按键。出现如下图所示图形窗口。



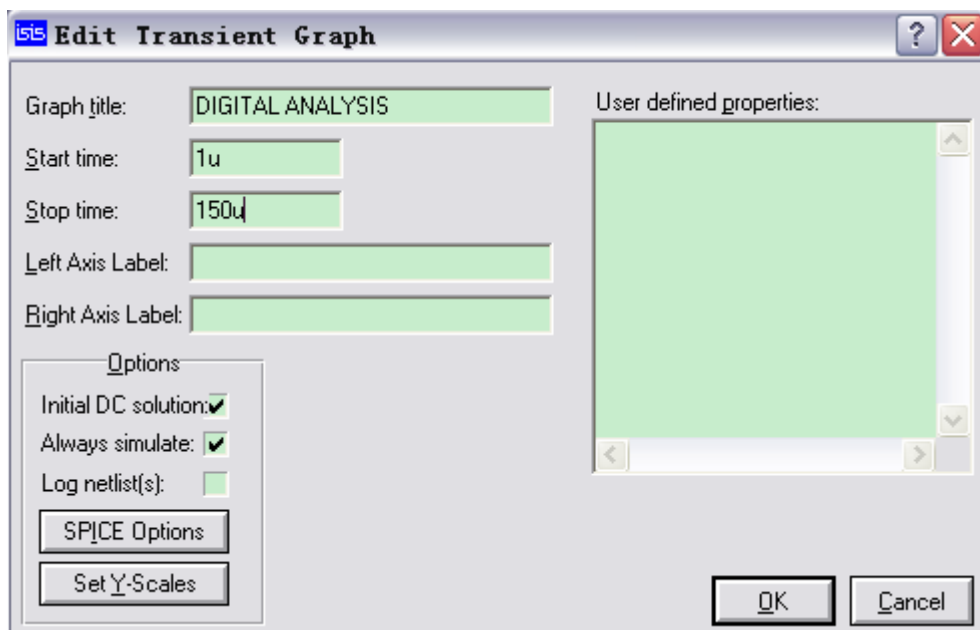
4、选择【Graph】→【Add Trace】，添加轨迹，或者也可以右键单击该图形窗口，在弹出菜单中选择“Add Trace”。在弹出的对话框中，选中 Probe P1 中的 CLOCK(电压探针加上后，所有探针的名称自动出现在这里)，单击“OK”，如下图所示。重复以上步骤，直到所有施放的探针都加到图形中为止。



5、最后，波形图显示如下图所示。



双击该波形图，弹出其属性页，如下图所示。可以重新命名标题。设定仿真的开始和结束时间，参数可参考下图。

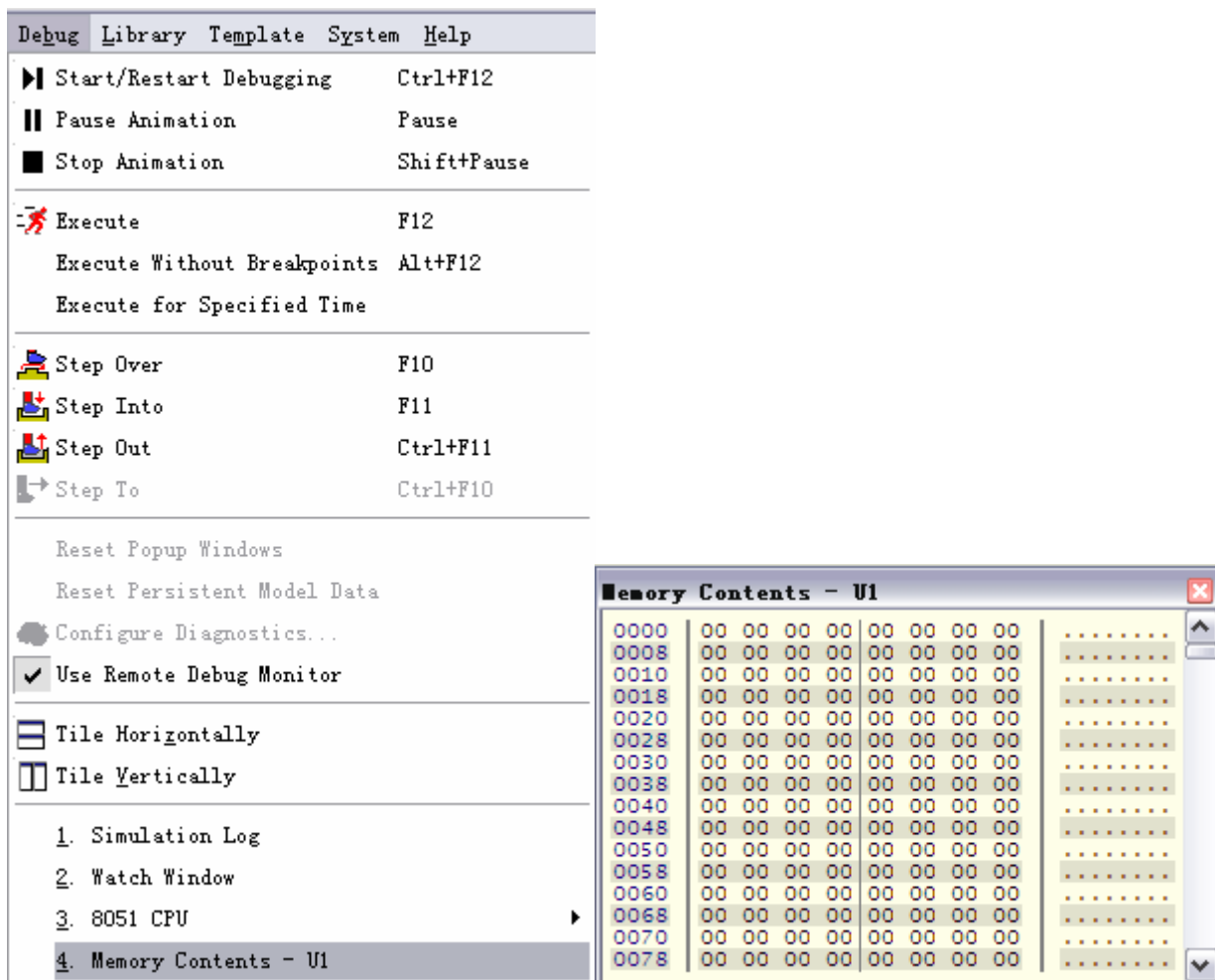


6、选择“Graph”→“Simulate Graph”或按空格键，生成波形。不需要运行仿真，只要执行此命令，所有探测点的波形就都自动生成了，如下图所示。这种波形不同于示波器显示的波形，它能够静态地保留在原理图中，供读者分析或随图形一起输出打印。当按下空格键后，它可以再次刷新生成。

7、左击波形图的绿色标题栏部分，或右键单击波形图，在弹出菜单中，选择“Maximize (Show Window)”，即可全屏显示波形，并可根据全屏显示的菜单更改波形及背景的颜色，拉动竖线，在左侧查看各时刻各观测点电平的高低等。

二、外部 RAM 存储器的观察

在仿真运行时，点击仿真条上的按键，将仿真暂停。选中菜单【Debug】->【Memory Contents-U1】(U1 为 RAM6264 的名称)，可以调出 RAM6264 的存储区域。



【实验内容】

用 SRAM6264 扩展单片机 AT89C51 的外部数据存储器，通过仿真窗口观察向 6264 写入数据的过程。

【实验目的】

- 1、掌握单片机 AT89C51 扩展外部数据存储器时的接口电路设计方法；
- 2、加深单片机对外部数据存储器进行读写过程的理解；
- 3、掌握波形图的仿真使用；
- 4、加深对地址映射区的理解。

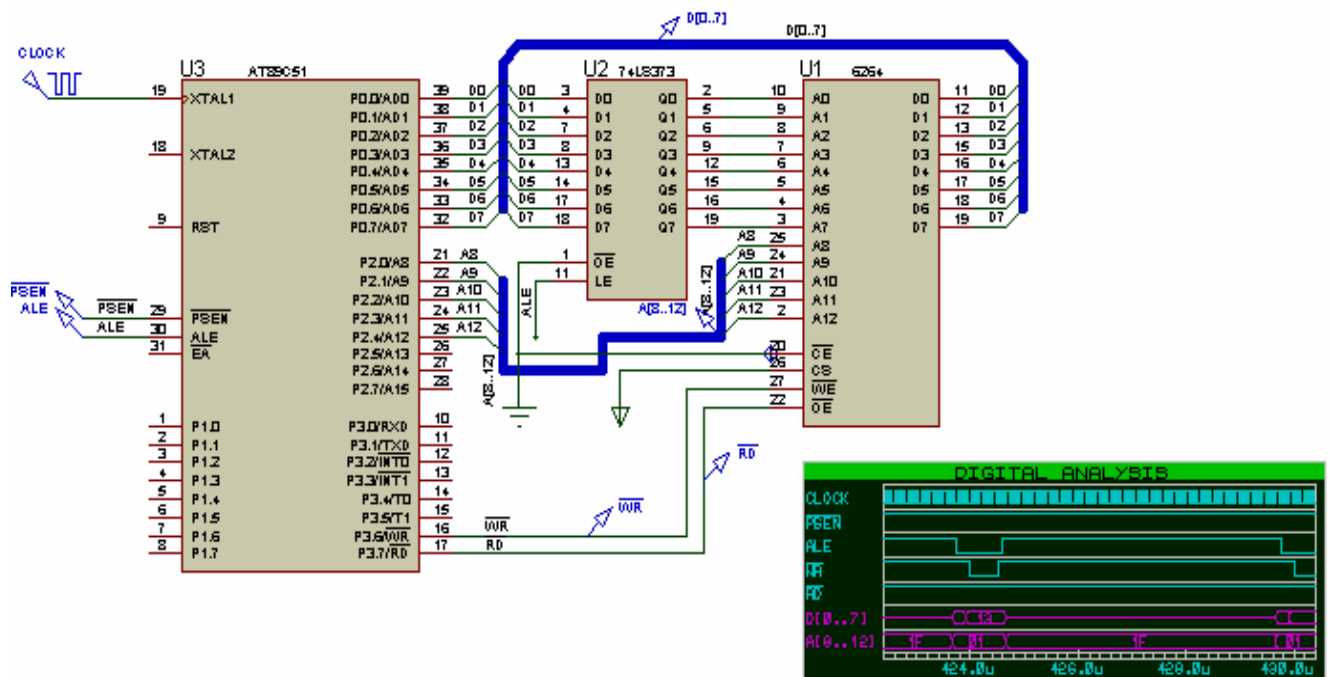
【实验要求】

- 1、按【PROTEUS 电路设计】完成电路设计；
- 2、实现向外部数据存储器 SRAM6264 周期性写入、读出数据；

- 3、使用图表仿真窗口，通过仿真窗口观察向 6264 写入、读出数据的过程；
- 4、从波形图上观察地址总线和数据总线的数据；
- 5、验证地址映射区的划分。注意阅读“【背景知识】”第二部分“外部 RAM 存储器的观察”。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，本实验装置电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--
74LS373	TTL 74LS Series	Flip Flops & Latches
6264	Memory ICs	Static RAM

2、放置元器件，补全振荡电路，上电复位电路；

3、放置电源和地；

4、连线；

5、参照原理图进行元件属性设置；

6、电气检查。

【源程序设计】

1、流程图：

2、在 KeilC 中进行源程序设计：

3、编译、生成目标代码

【PROTUES 仿真】

1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；

2、仿真、调试代码

3、注意使用观察窗口

实验 2-9 数码管显示实验

【背景知识】

LED 数码管是由发光二极管作为显示字段的数码型显示器件。七段 LED 显示器内部由七个条形发光二极管和一个小圆点发光二极管组成，根据二极管连接形式的不同分为共阳型和共阴型。

COM 为数码管的公共端，称为数码管的位，a、b、c、d、e、f、g、dp 称为数码管的段。LED 数码管的 g~a 七个发光二极管因加正电压而发亮，因加零电压而不以发亮，不同亮暗的组合就能形成不同的字形，这种组合称之为字形码，下面给出共阳极的字形码见下表：

8 段数码管十六进制数段码表

数字	共阳极段码	共阴极段码	数字	共阳极段码	共阴极段码
0	C0H	3FH	9	90H	6FH
1	F9H	06H	A	88H	77H
2	A4H	5BH	B	83H	7CH
3	B0H	4FH	C	C6H	39H
4	99H	66H	D	A1H	5EH
5	92H	6DH	E	86H	79H
6	82H	7DH	F	8EH	71H
7	F8H	07H	灭	FFH	00H
8	80H	7FH			

从共阳型数码管内部结构可以看出，它内部电路和我们前面做的流水灯电路有共同之处。只要将它的公共端接电源，每个段通过限流电阻分别接到单片机引脚上，就可以用单片机的 I/O 口输出要显示的信息，在数码管上显示。可见数码管的静态显示电路连接很简单。

LED 静态显示电路每个数码管都要一个 8 位的 I/O 口控制，则 MCS-51 单片机只能驱动 4 位静态 LED 数码管。LED 静态显示电路占用 I/O 口线较多，只适合数码管较少的场合。

【实验内容】

数码管是单片机系统常采用的显示器件。LED 有静态显示和动态显示两种方式。在多位数码管显示时，为了简化电路、降低成本，常采用动态显示，即将所有数码管位的段线并联在一起，由一个 8 为 I/O 控制；而共阴极（共阳极）公共端分别由相应的 I/O 线控制，实现

各位的分时选通。

【实验目的】

- 1、理解数码管的共阴极和共阳极显示原理；
- 2、掌握和熟悉数码管的动态显示；
- 3、学会利用单片机实现数码管动态显示程序设计。

【实验要求】

- 1、动态数码管显示程序设计，实现电路板上的 6 个数码管分别显示“123456”。
- 2、实现电路板上的 6 个数码管的显示内容循环左移或右移。

注意：所选数码管是共阴极还是共阳极！

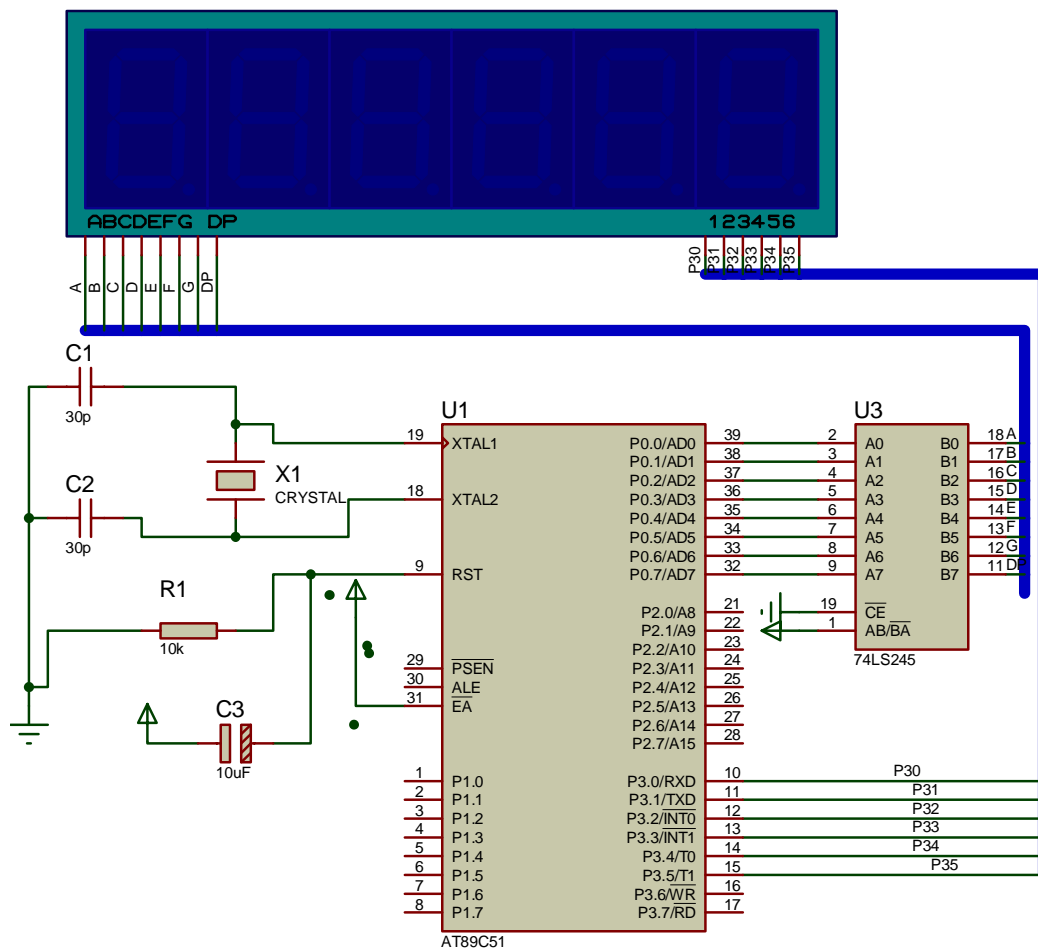
【PROTEUS 电路设计】

本实验的系统设计比较灵活，这里提供三种方案，请同学们自己分析下这几种方案有什么不同，在编写程序时要注意什么？请参看电路原理图 A、B 和 C。

- 1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图 A 或 B。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
74LS245	TTL 74LS Series	Transceivers
74LS273	TTL 74LS Series	Flip-Flops&Latches
74LS373	TTL 74LS Series	Flip Flops & Latches
7SEG-MPX6-CC-BLUE	Optoelectronics	7-Segments Displays
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--
NOT		



电路原理图 A

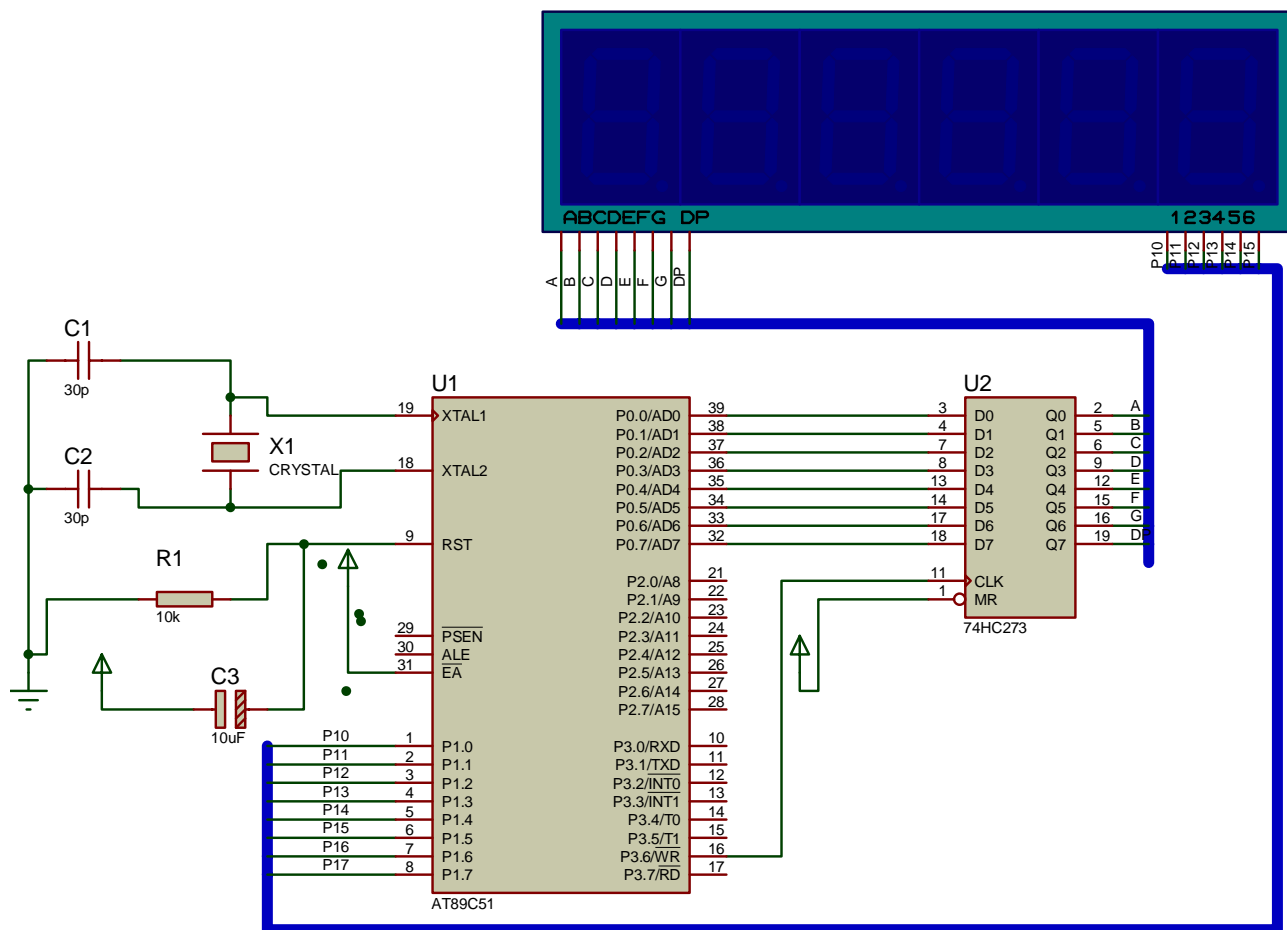
- 2、放置元器件，补全振荡电路，上电复位电路；；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

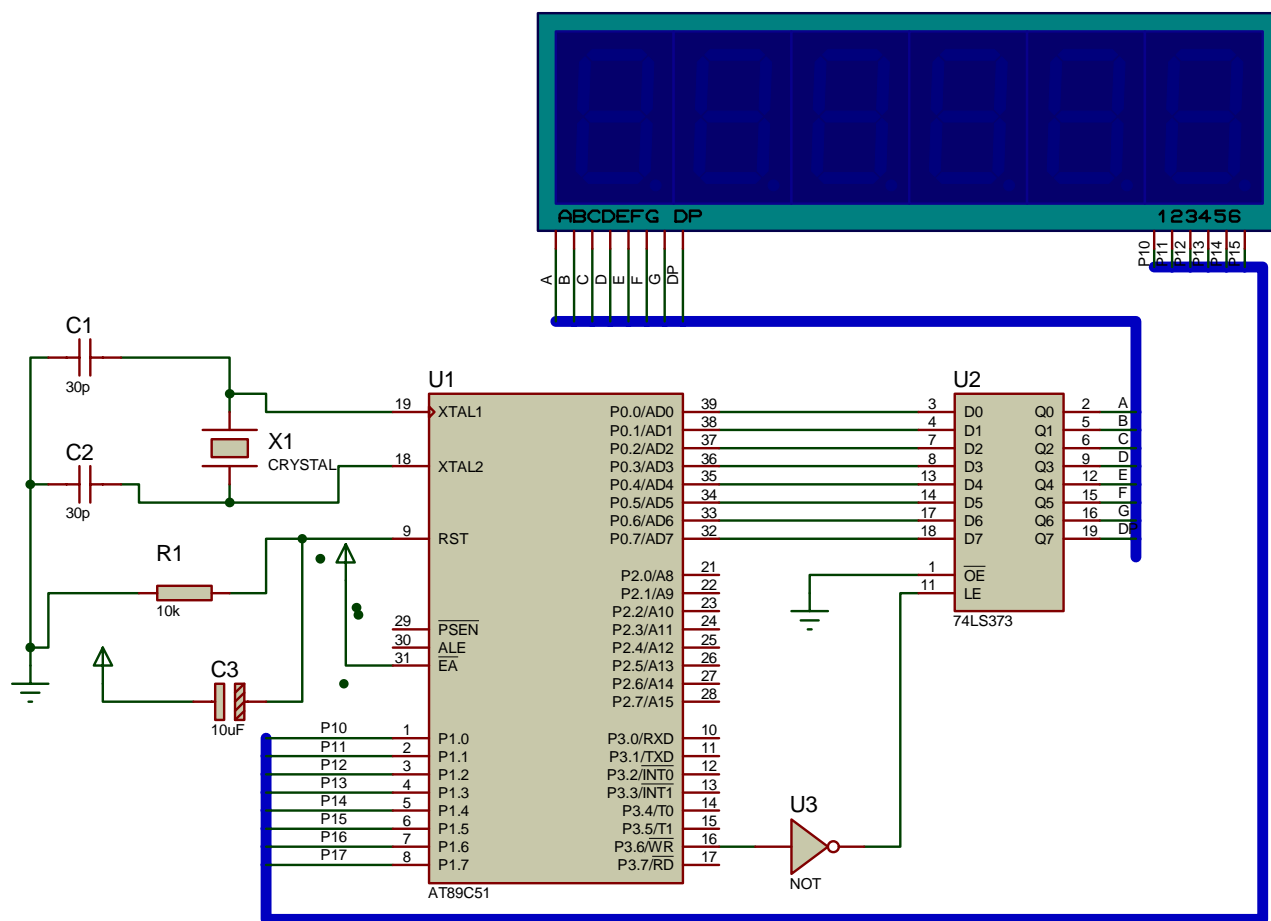
- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码;
- 2、仿真、调试代码
- 3、注意使用观察窗口



电路原理图 B



电路原理图 C

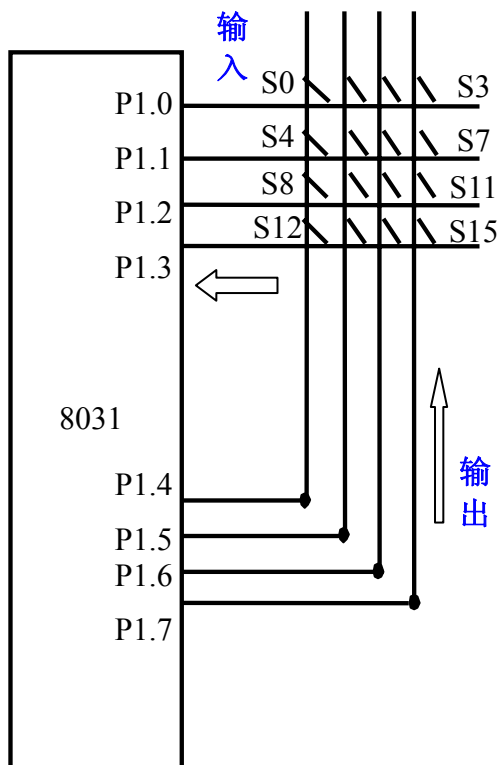
实验 2-10 键盘扫描

【背景知识】

矩阵式键盘，又称行列式键盘：键盘按行列构成矩阵模式，因此按键数较多。

一、按键的识别过程

- 1、键盘扫描，判定是否有“闭合键”；
- 2、按键识别，确认“闭合键”的行列位置；
- 3、产生键码，排除多键、串键（复按）及去抖动。



键盘扫描原理图

如上图所示，则键盘扫描过程如下：

- 1、P1 口输出“0F(F0)”；

P1 口低（高）4 位读入数据时：全“1”，表示无按键按下，否则，表示有按键按下。

- 2、进一步确认行列号：

P1 口输出“0F”，确认 P1.X 输入为“0”；

P1 口输出“F0”，确认 P1.Y 输入为“0”；

则按下键的坐标就是（X，Y）

二、按键抖动

抖动的原因：目前大部分按键或键盘都是利用机械触电的合、断作用。机械触点在闭合及断开瞬间由于弹性作用的影响，在闭合及断开瞬间均有抖动过程，从而使电压信号也出现抖动，抖动时间的长短与开关的机械特性有关，一般为 5~10ms。而单片机对键盘扫描一次仅需几百微妙。这样，将会对键盘扫描产生误判。为了保证单片机对按键闭合仅作一次输入处理，必须去除抖动的影响。

去抖动的方法：

- 1、外加硬件电路，用 RS 触发器或单稳态电路构成的去抖动电路，或键盘扩展专用芯片。
- 2、在检测按键按下时，执行约 10ms 的延时程序后（避开抖动前沿区域），再确认按键是否仍然保持闭合状态。

【实验内容】

16 键盘扩展，通过键盘扫描和去抖动，实现每一按键的精确定位。

【实验目的】

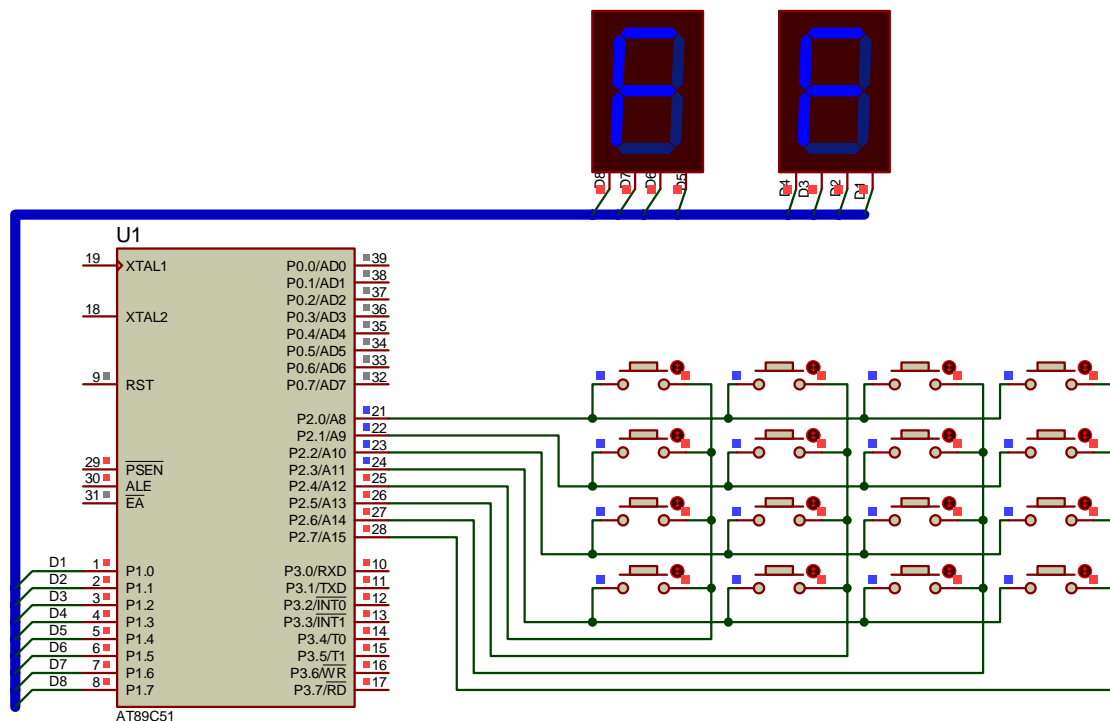
- 1、理解键盘扫描和去抖动的原理；
- 2、掌握键盘扫描实现方法；
- 3、掌握一种 16 键盘的实现方法。

【实验要求】

- 1、16 键盘程序设计，根据每一按键，实现不同的按键编号。
 - 2、按键初始显示“FF”。以后无按键按下时，显示上一次按键的编号。
- 注意、原理图中的数码管，提供的是 BCD 码。

【PROTEUS 电路设计】

在 ISIS 中进行电路图设计，本实验装置电路原理图如下图所示。



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
7SEG-BCD-BLUE(7 段-BCD--蓝色数码管)	Optoelectronics	7-Segments Displays
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--
BUTTON	Switches & Relays	Switches

- 2、放置元器件，补全振荡电路，上电复位电路；；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

- 1、流程图：
- 2、在 KeilC 中进行源程序设计：
- 3、编译、生成目标代码

【PROTUES 仿真】

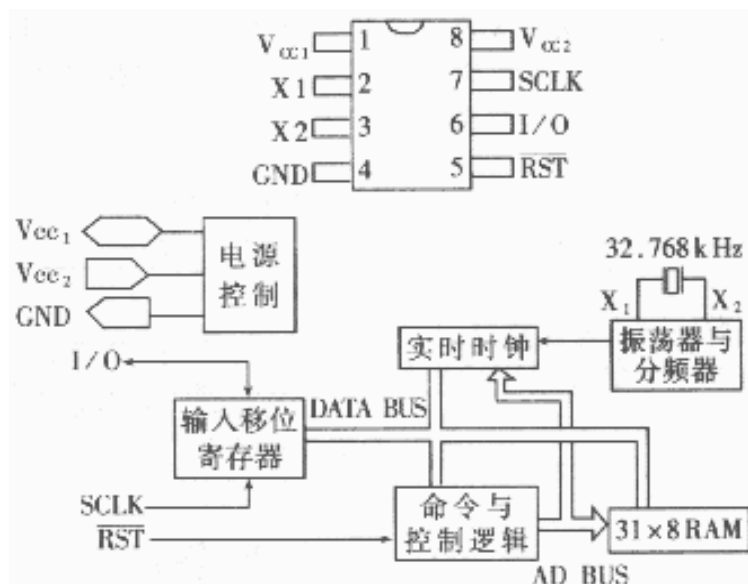
- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

综合实验一—电子时钟温度计

【背景知识】

一、时钟芯片 DS1302 介绍

DS1302 是美国 DALLAS 公司推出的一种高性能、低功耗、带 RAM 的实时时钟电路，它可以对年、月、日、周日、时、分、秒进行计时，具有闰年补偿功能，工作电压为 2.5V~5.5V。采用三线接口与 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信号或 RAM 数据。DS1302 内部有一个 31×8 的用于临时性存放数据的 RAM 寄存器。增加了主电源/后备电源双电源引脚，同时提供了对后备电源进行涓细电流充电的能力。



DS1302 管脚图及内部结构图

二、DS1302 的工作原理：

1. DS1302 引脚功能如下表所示：

管脚号	管脚名称	功 能
1	V _{CC2}	主电源
2、3	X1, X2	32.768 kHz 晶振接口
4	GND	接地
5	RST	复位兼片选端, 读/写操作时必为高电平
6	I/O	串行数据输入/输出
7	SCLK	串行时钟输入端, 是串行数据的同步信号
8	V _{CC1}	后备电源

2. 控制字格式

DS1302 控制字格式如下表所示。控制字的最高位 D7 必须是 1, 如果它为 0, 则不能把数据写入到 DS1302 中。如果 D6 为 0, 则表示存取日历时钟数据; 如果 D6 为 1, 则表示存取 RAM 数据。D5~D1 指示操作单元的地址。最低位 D0 为 0, 表示要进行写操作; D0 为 1, 表示进行读操作。**注意: 控制字节总是从最低位开始输出。**

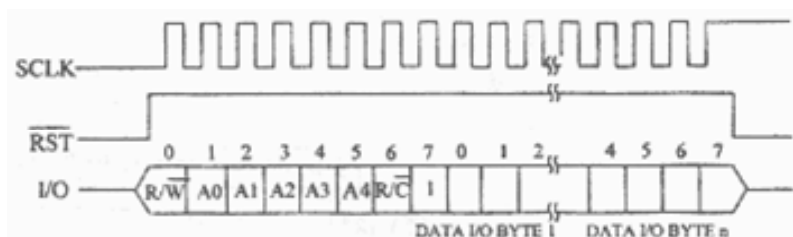
D7	D6	D5	D4	D3	D2	D1	D0
1	RAM/ $\overline{\text{CK}}$	A4	A3	A2	A1	A0	RD/ $\overline{\text{WR}}$

3. 复位和时钟控制

DS1302 通过把 RST 输入驱动置高电平来启动所有的数据传送。RST 输入有两种功能: 首先, RST 接通控制逻辑, 允许地址/命令序列送入移位寄存器; 其次, RST 提供了终止单字节或多字节数据的传送手段。当 RST 为高电平时, 所有数据传送被初始化, 允许对 DS1302 进行操作。如果在传送过程中置 RST 为低电平, 则会终止此数据传送, 并且 I/O 引脚变为高阻状态。上电运行时, 在 $V_{CC} \geq 2.5 \text{ V}$ 之前, RST 必须保持低电平。只有在 SCLK 为低电平时, 才能将 RST 置为高电平。

4. 数据的输入与输出

DS1302 写入是在控制字输入后的下一个 SCLK 时钟的上升沿时, 数据被写入 DS1302, 数据输入从低位即 D0 开始。同样, 在紧跟 8 位的控制字后的下一个 SCLK 脉冲的下降沿读出 DS1302 的数据, 读出数据的顺序为低位 D0~D7。



数据读写时序

5. DS1302 寄存器

DS1302 共有 12 个寄存器，其中有 7 个寄存器与日历、时钟相关，存放的数据位为 BCD 码形式。

此外，DS1302 还有年份寄存器、控制寄存器、充电寄存器、时钟突发寄存器及与 RAM 相关的寄存器等。时钟突发寄存器可一次性顺序读写除充电寄存器外的所有寄存器内容。

(DS1302 与 RAM 相关的寄存器分为两类，一类是单个 RAM 单元，共 31 个，每个单元组态为一个 8 位的字节，其命令控制字为 C0H~FDH，其中奇数为读操作，偶数为写操作；再一类为突发方式下的 RAM 寄存器，此方式下可一次性读写所有的 RAM 的 31 个字节，命令控制字为 FEH (写)、FFH (读)。)

寄存器名		命令字格式		取值范围	位 内 容							
		写操作	读操作		D7	D6	D5	D4	D3	D2	D1	D0
秒寄存器		80H	81H	00~59	CH	10SEC			SEC			
分寄存器		82H	83H	00~59	0	10MIN			MIN			
小时寄存器		84H	85H	01~12 或 11~23	12/24	0	10/AP	HR	HR			
日期寄存器		86H	87H	01~31	0	0	10DATA		DATA			
月份寄存器		88H	89H	01~12	0	0	0	10M	MONTH			
星期寄存器		8AH	8BH	01~07	0	0	0	0	0	DAY		
年份寄存器		8CH	8DH	00~99	10YEAR			YEAR				
写保护寄存器		8EH	8FH	00H/80H	WP	0						
涓流充电寄存器		90H	91H	—	TCS			DS		RS		
时钟突发寄存器		BEH	BFH	—	—							
RAM 突发寄存器		FEH	FFH	—	—							
RAM 寄存器	0	C0H	C1H	00~FFH	RAM 数据							
	⋮	⋮	⋮	00~FFH								
	30	FCH	FDH	00~FFH								

6. 启动 DS1302 顺序:

复位→关闭写保护→时钟脉冲串（突发）写入（设置初始时间，调整）→读取时间
 复位→时钟脉冲串（突发）→读取时间值

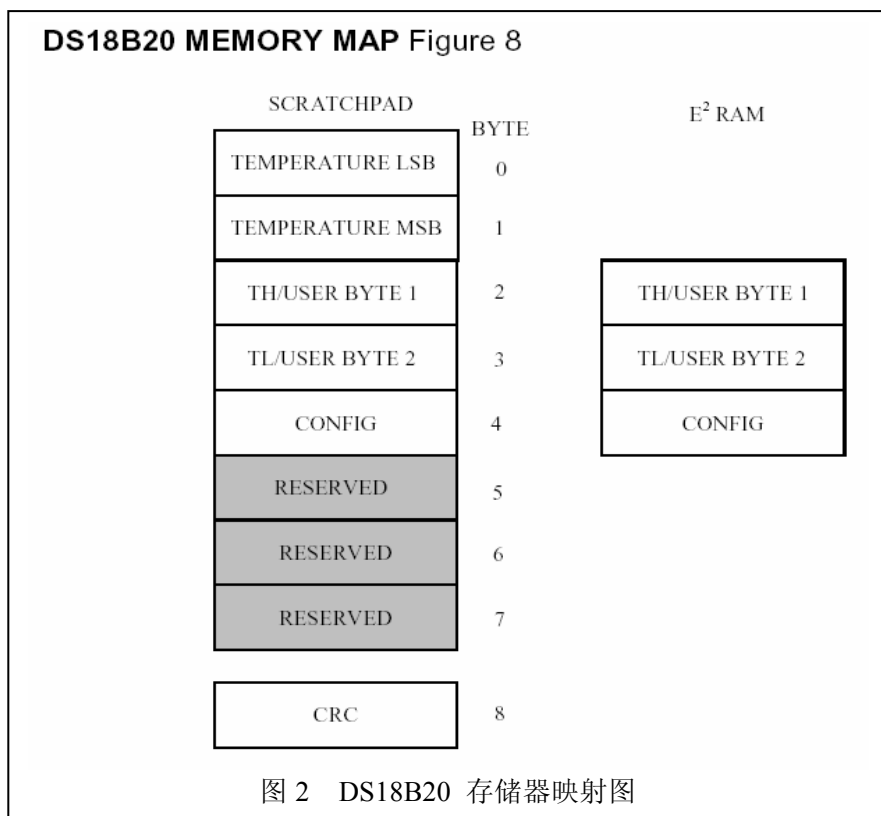
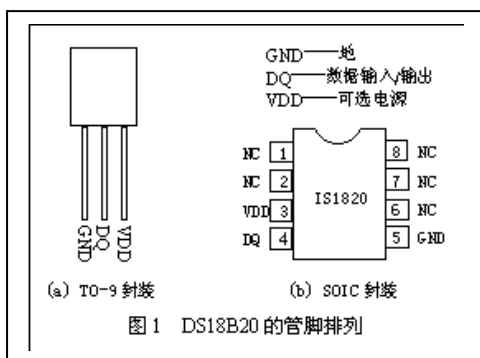
二、温度传感器芯片 DS18B20 介绍

DS18B20 是美国 DALLAS 公司最新推出的一种可组网数字式温度传感器，DS18B20 也能够直接读取被测物体的温度值。它体积小，电压适用范围宽（3V~5V），用户还可以通过编程实现 9~12 位的温度读数，即具有可调的温度分辨率。

DS18B20 只有一个数据输入/输出，属于单总线专用芯片之一。DS18B20 工作时被测温度值直接以“单总线”的数字方式传输，**低字节先传输**，大大提高了系统的抗干扰能力。其内部采用在板温度测量专利技术，测量范围为-55℃~+125℃，在-10℃~+85℃时，精度为±0.5℃。每个 DS18B20 在出厂时都已具有唯一的 64 位序列号，因此一条总线上可以同时挂接多个 DS18B20，而不会出现混乱现象，一般不超过 8 个。另外用户还可自设定非易失性温度报警上下限值 TH 和 TL（掉电后依然保存）。DS18B20 在完成温度变换后，所测温度值将自动与贮存在 TH 和 TL 内的触发值相比较，如果测温结果高于 TH 或低于 TL，DS18B20 内部的告警标志就会被置位，表示温值超出了测量范围，同时还有报警搜索命令识别出温度超限的 DS18B20。

1. DS18B20 的内部存储器结构

图 2 为 DS18B20 的内部存储器结构，它包括一个 8 字节暂存 RAM 和一个非易失性电可擦除（E²）RAM。



其中暂存存储器作用是在单线通信时确保数据的完整性，它包括 8 个字节，头两个字节表示测得的温度读数，数据格式如下：

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
MSB (单位: $^{\circ}\text{C}$)				LSB			
S	S	S	S	26	25	24	

S=1 时表示温度为负，S=0 时表示温度为正，其余低位以二进制补码形式表示，最低位为 1 时表示 0.0625°C 。温度/数字对应关系如表 1 所示。

表 1 DS18B20 温度/数据对应关系表

温度($^{\circ}\text{C}$)	输出的二进制码	对应的十六进制码
+125	0000 0111 1101 0000	07D0H
+85	0000 0101 0101 0000	0550H
+25.0625	0000 0001 1001 0001	0191H
10.125	0000 0000 1010 0010	00A2H
0.5	0000 0000 0000 1000	0008H
0	0000 0000 0000 0000	0000H
-0.5	1111 1111 1111 1000	FFF8H
-10.125	1111 1111 0101 1110	FF5EH
-25.0625	1111 1110 0110 1111	FF6FH
-55	1111 1100 1001 0000	FC90H

DS18B20 内部暂存存储器的第 5 个字节是结构寄存器，它主要用于确定温度值的数字转换分辨率。字节结构如下：

0	R_1	R_0	1	1	1	1	1
---	-------	-------	---	---	---	---	---

MSB

LSB

其中 R_1 、 R_0 用于设置分辨率，如表 2 所示。

表 2 DS18B20 分辨率设置表

R_1	R_0	温度分辨率	最大转换时间
0	0	9 位	93.75ms
0	1	10 位	187.5ms
1	0	11 位	375ms
1	1	12 位	750ms

2. DS18B20 的硬件连接（以 51 单片机为例）

DS18B20 与单片机的接口极其简单，只需将 DS18B20 的信号线与单片机的一位双向端口相连即可。DQ 与 P2.2 连接。

传感器初始值为+85.0℃的温度值。

3. 软件编程

（1）DS18B20 的工作过程

DS18B20 的工作遵循严格的单总线协议。

- a. 主机首先发一复位脉冲，使信号线上所有的 DS18B20 芯片都被复位。
- b. 接着发送 ROM 操作命令，使序列号编码匹配的 DS18B20 被激活，准备接收下面的内存访问命令。（略）
- c. 内存访问命令控制选中的 **DS18B20 的工作状态—》完成温度转换—》读取等工作**（单总线在 ROM 命令发送之前存储命令和控制命令不起作用）。

工作中系统对 DS18B20 的操作以 ROM 命令和存储器命令形式出现。

其中 ROM 操作命令均为 8 位长，命令代码分别为：

读 ROM (33H)；

匹配 ROM (55H)；

跳过 ROM (CCH)；

搜索 ROM (F0H)；

告警搜索 (ECH) 命令。

存储器操作命令为：

写暂存存储器 (4EH)；

读暂存存储器 (BEH)；

复制暂存存储器 (48H)；

温度变换 (44H)；

重新调出 EERAM (B8H)；

读电源供电方式 (B4H) 命令。

综上所述，驱动 DS18B20 的指令流程如下：

复位->跳过 ROM->温度变换->延时（供温度转换）->复位->跳过 ROM->读暂存器。

（2）工作时序

由于 DS18B20 的测温分辨率提高了（12 位），因此对时序及电特性参数要求较高，必须严格按照 DS18B20 的时序要求去操作。DS18B20 数据的读写也是由主机读写特定时间片来完成的，包括初始化、读时间片和写时间片。

主机控制 DS18B20 完成任何操作之前必须先初始化：

即主机发一复位脉冲（最短为 480μs 的低电平），接着主机释放总线进入接收状态，DS18B20 在检测到 I/O 引脚上的上升沿之后，等待 15~60μs 然后发出存在脉冲（60~240μs 的低电平）。参看时序图。

写时间片：

将数据线从高电平拉至低电平，产生写起始信号。在 $15\mu\text{s}$ 之内将所需写的位送到数据线上，在 $15\mu\text{s}$ 到 $60\mu\text{s}$ 之间对数据线进行采样，如果采样为高电平，就写 1，如果为低电平，写 0 就发生。每一个写时间槽必须是： $60\text{MS} < T < 120\text{MS}$ 。在开始另一个写周期前必须有 $1\mu\text{s}$ 以上的高电平恢复期。

参看时序图

读时间片：

主机将数据线从高电平拉至低电平 $1\mu\text{s}$ 以上，（再使数据线升为高电平，这不是协议中的内容，而是 51 准双向口的原因），从而产生读起始信号。主机在读时间片下降沿之后 $15\mu\text{s}$ 内完成读位。每个读周期最短的持续期为 $60\mu\text{s}$ ，各个读周期之间也必须有 $1\mu\text{s}$ 以上的高电平恢复期。参看时序图，如图 3，图 4。

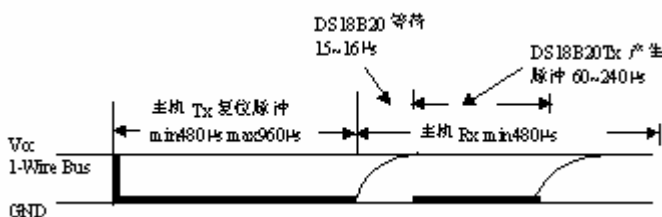


图 3 DS18B20 初始化时序

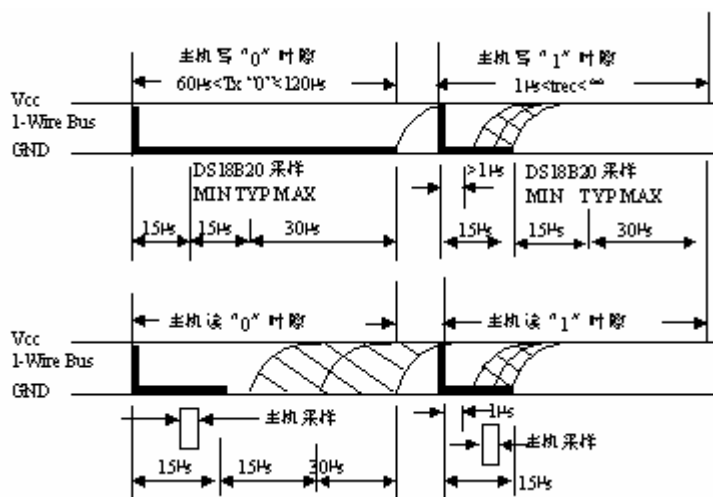


图 4 DS18B20 读写时序图

【实验内容】

时钟温度计是常见的单片机应用系统。但是一个功能完善，操作便捷的时钟温度计系统也不是很容易设计。本次实验是实现一个时钟系统的功能，同学们可以在完成基本功能的基

础上，充分发挥，制作出自己的时钟温度计系统。

【实验目的】

- 1、按要求独立设计 MCS—51 系统；
- 2、实现时钟基本功能；
- 3、实现温度测量功能；
- 4、掌握 Proteus 软件绘制 PCB 图。
- 5、熟练掌握汇编程序。

【实验要求】

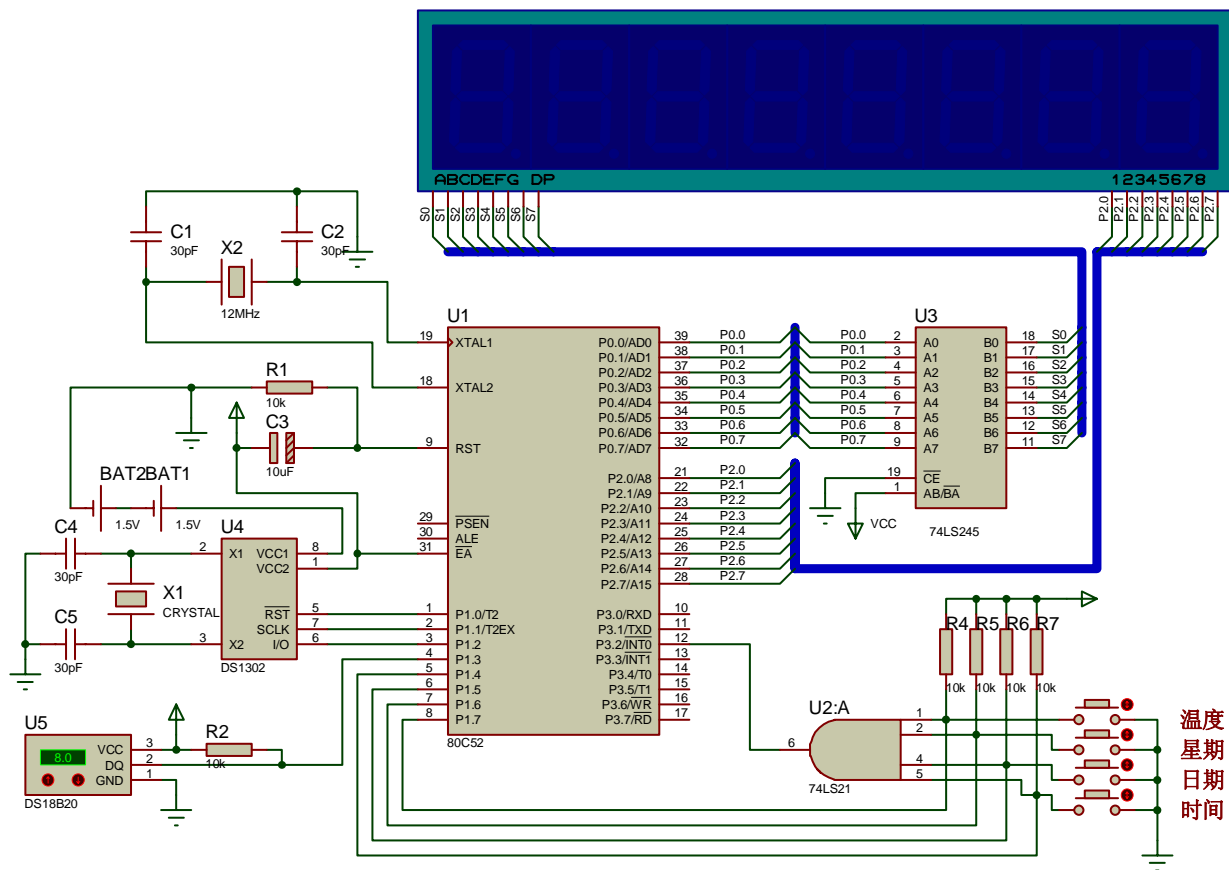
基本功能：

- 1、系统具有时间、日期、星期切换显示功能；
- 2、系统具有温度显示功能；

扩展功能：

- 1、系统具有秒表功能；
- 2、系统具有整点响铃报时功能；
- 3、系统具有定时响铃（加入蜂鸣器）、撤销功能；
- 4、系统具有温度摄氏华氏转换功能。

【PROTEUS 电路设计】



电路原理图

1、按照元件清单从 PROTEUS 库中选取元器件，进行第 2、3、4、5、6 步，完成原理图。

元件清单

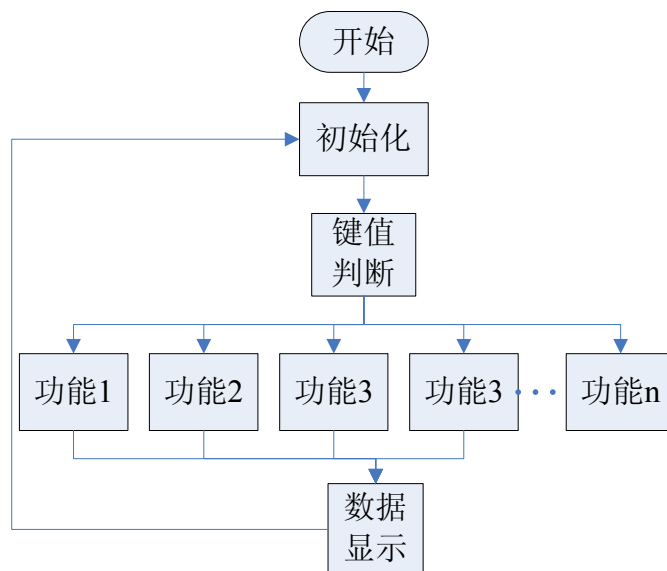
元件名称	所属类	所属子类
AT80C51（单片机）	Microprocessor ICs	8051 Family
RES（电阻）	Resistors	Generic
CAP（电容）	Capacitors	Generic
CAP-ELEC（电解电容）	Capacitors	Generic
CRYSTAL（晶振）	Miscellaneous	--
BUTTON	Switches & Relays	Switches

DS1302	Microprocessor ICs	Peripherals
DS18B20	Data Converters	Temperature Sensors
BUTTON	Switches & Relays	Switches
74LS21	TTL 74LS series	Gates & Inverters

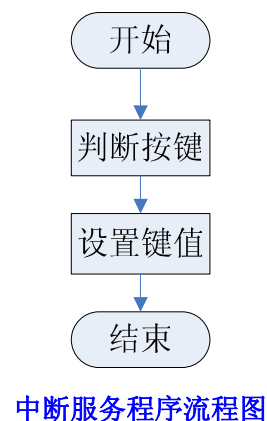
- 2、放置元器件；
- 3、放置电源和地；
- 4、连线；
- 5、参照原理图进行元件属性设置；
- 6、电气检查。

【源程序设计】

- 1、流程图：



主程序流程图



中断服务程序流程图

- 2、在 KeilC 中进行源程序设计；
- 3、编译、生成目标代码

【PROTUES 仿真】

- 1、在 AT89C51 属性页中加载 KeilC 中生成的目标代码；
- 2、仿真、调试代码
- 3、注意使用观察窗口

附：基本模块参考代码

```

;*****

;RAM 定义

;*****

SECOND    EQU    60H            ;时间存储首地址
KEY        EQU    67H            ;中断控制关键字


TEMPER_L   EQU    29H            ;18B20 低位存储地址
TEMPER_H   EQU    28H            ;18B20 低位存储地址
FLAG1      EQU    38H            ;18B20 数据有效判断位


BAI_BIT    EQU    33H            ;百位存储单
SHI_BIT    EQU    32H            ;十位存储单元
GE_BIT     EQU    31H            ;个位存储单元
T_DF       EQU    27H            ;暂存温度小数部分
T_INTEGER  EQU    30H            ;暂存温度整数部分


;*****

;引脚连接

;*****

LED_SHOW   EQU    P0            ;LED 显示数据口
T_RST      BIT    P1.0          ;实时时钟复位线引脚
T_CLK      BIT    P1.1          ;实时时钟时钟线引脚
T_IO       BIT    P1.2          ;实时时钟数据线引脚
DATAIN     BIT    P1.3          ;温度读写引脚
TIME_SHOW  BIT    P1.4          ;时间显示按钮
DAY_SHOW   BIT    P1.5          ;日期显示按钮
WEEK_SHOW  BIT    P1.6          ;星期显示按钮

```

```

TEM_SHOW  BIT    P1.7           ;温度显示按钮
POINT     BIT    P0.7           ;小数点

```

```

ORG    0000H

```

```

LJMP    MAIN

```

```

ORG    0003H           ;设置外部中断矢量地址

```

```

LJMP    INT             ;跳转到中断控制入口处

```

```

;*****

```

```

;主程序

```

```

;*****

```

```

MAIN:    MOV    SP , #90H       ;初始化

```

```

        MOV    TCON, #01H

```

```

        MOV    IE , #81H       ; 对中断进行初始化

```

```

        MOV    R0, #60H

```

```

        CALL   SET1302          ;启动 DS1302

```

```

        CALL   DELAY            ;延时

```

```

;*****

```

```

;按键中断程序

```

```

;*****

```

```

INT:     MOV    A, P1           ;读 P1 口数据

```

```

        ANL    A, #0F0H        ;取用 P1 口的高四位数据

```

```

SHIY1:   CJNE   A, #0E0H, NEXT1 ;P1. 4=0 时进入时间显示分支

```

```

        AJMP   TIME

```

```

NEXT1:   CJNE   A, #0D0H, NEXT2 ;P3. 4=0 时进入日期显示分支

```

```

        AJMP   DAY

```

```

NEXT2:   CJNE   A, #0B0H, NEXT3 ;P3. 1=0 时进入星期显示分支

```

```

        AJMP    WEEK
NEXT3:   CJNE    A, #70H, DONE           ;P3.0=0 时进入温度显示分支
        AJMP    TEM

TIME:    MOV     KEY, #00H              ;时间分支
        AJMP    DONE
DAY:     MOV     KEY, #01H              ;日期分支
        AJMP    DONE
WEEK:    MOV     KEY, #02H              ;星期分支
        AJMP    DONE
TEM:     MOV     KEY, #03H              ;温度分支
DONE:    RETI

;*****

;*****

;DS1302 初始化子程序

;*****

SET1302:

        CLR     T_RST
        CLR     T_CLK
        SETB    T_RST
        MOV     B,      #8EH           ;控制寄存器
        LCALL   RTINPUTBYTE
        MOV     B,      #00H           ;写操作前 WP=0
        LCALL   RTINPUTBYTE
        SETB    T_CLK
        CLR     T_RST

```



```

CLR    T_RST
CLR    T_CLK
SETB   T_RST
MOV     B,    #8EH           ;控制寄存器
LCALL  RTINPUTBYTE
MOV     B,    #80H           ;控制, WP=1, 写保护
LCALL  RTINPUTBYTE
SETB   T_CLK
CLR    T_RST
RET

```

```

;*****

```

```

;时间初始化

```

```

;*****

```

```

INIT_TIME:

```

```

MOV     R0,    #SECOND;
MOV     R7,    #8       ; 秒 分 时
MOV     R1,    #80H     ; 秒写地址

```

```

S13021:

```

```

CLR    T_RST
CLR    T_CLK
SETB   T_RST
MOV     B,    R1           ;写秒 分 时 地址
LCALL  RTINPUTBYTE
MOV     A,    @R0          ;写秒数据
MOV     B,    A

```

```

LCALL  RTINPUTBYTE
INC     R0
INC     R1
INC     R1
SETB    T_CLK
CLR     T_RST
DJNZ    R7,    S13021

```

```
;*****
```

```
;读时间子程序——GET1302
```

```
;*****
```

```
GET1302:
```

```

MOV     R0,    #SECOND;
MOV     R7,    #7
MOV     R1,    #81H    ;秒地址

```

```
G13021:
```

```

CLR     T_RST
CLR     T_CLK
SETB    T_RST
MOV     B,     R1    ;秒 分 时 日 月 星期 年 地址
LCALL   RTINPUTBYTE
LCALL   RTOUTPUTBYTE
MOV     @R0,   A    ;秒
INC     R0
INC     R1
INC     R1
SETB    T_CLK
CLR     T_RST
DJNZ    R7,    G13021

```

RET

;写 DS1302 一个字节

RTINPUTBYTE:

MOV R4, #8

INBIT1:

MOV A, B

RRC A

MOV B, A

MOV T_10, C

SETB T_CLK

CLR T_CLK

DJNZ R4, INBIT1

RET

;读 DS1302 一个字节

RTOUTPUTBYTE:

MOV R4, #8

OUTBIT1:

MOV C, T_10

RRC A

SETB T_CLK

CLR T_CLK

DJNZ R4, OUTBIT1

RET

```
;*****  
;DB18B20 子程序  
;*****  
;初始化 18B20  
;*****  
INIT_1820:  
        SETB    DATAIN  
        NOP  
        CLR     DATAIN  
        MOV     R1, #3  
TSR1:   MOV     R0, #107  
        DJNZ    R0, $  
        DJNZ    R1, TSR1  
        SETB    DATAIN  
        NOP  
        NOP  
        NOP  
        MOV     R0, #25H  
TSR2:   JNB     DATAIN, TSR3  
        DJNZ    R0, TSR2  
        CLR     FLAG1  
        SJMP    TSR7  
TSR3:   SETB    FLAG1  
        CLR     POINT  
        MOV     R0, #117  
TSR6:   DJNZ    R0, $  
TSR7:   SETB    DATAIN
```

RET

;*****

;读温度

;*****

GET_TEMPER:

SETB DATAIN

LCALL INIT_1820

JB FLAG1, TSS2

NOP

RET

TSS2:

MOV A, #0CCH

LCALL WRITE_1820

MOV A, #44H

LCALL WRITE_1820

LCALL DELAY1S

LCALL INIT_1820

MOV A, #0CCH

LCALL WRITE_1820

MOV A, #0BEH

LCALL WRITE_1820

LCALL READ_1820

RET

WRITE_1820:

MOV R2, #8 ;写入命令

CLR C

WR1:

```
CLR    DATAIN
MOV     R3, #6
DJNZ    R3, $
RRC     A
MOV     DATAIN, C
MOV     R3, #23
DJNZ    R3, $
SETB    DATAIN
NOP
DJNZ    R2, WR1
SETB    DATAIN
RET
```

;*****

;发送读温度命令

;*****

READ_1820:

```
MOV     R4, #2
MOV     R1, #29H
RE00:   MOV     R2, #8
RE01:   CLR     C
        SETB    DATAIN
        NOP
        NOP
        CLR     DATAIN
        NOP
        NOP
        NOP
```

```

        SETB    DATAIN
        MOV     R3, #9
RE10:    DJNZ    R3, RE10
        MOV     C, DATAIN
        MOV     R3, #23
RE20:    DJNZ    R3, RE20
        RRC     A
        DJNZ    R2, RE01
        MOV     @R1, A
        DEC     R1
        DJNZ    R4, RE00
        RET

```

```

;*****

```

```

;温度转化子程序

```

```

;*****

```

```

CVTTMP:    MOV     A, #0FH
           ANL     A, TEMPER_L
           MOV     T_DF, A           ;获得小数部分(4 位)
           MOV     A, TEMPER_L
           SWAP    A
           MOV     TEMPER_L, A

           MOV     A, TEMPER_H
           SWAP    A
           MOV     R0, #TEMPER_L
           XCHD    A , @R0

```

```

        MOV    T_INTEGER, A        ;获得整数部分(1 字节)
    AJMP    DISPLAY
    RET

```

```

DISPLAY:  MOV    A, T_INTEGER
          RLC    A
          JC     FU1
          SJMP   ZHENG

```

```

FU1:      MOV    A, T_INTEGER
          DEC    A
          CPL    A
          SJMP   ZHUANHUA

```

```

ZHENG:    MOV    A, T_INTEGER

```

```

ZHUANHUA: MOV    B, #10
          DIV    AB
          MOV    GE_BIT, B          ;个位存在 C_BIT
          MOV    B, #10
          DIV    AB
          MOV    BAI_BIT, A         ;百位存在 A_BIT
          MOV    SHI_BIT, B         ;十位存在 B_BIT
          RET

```

```

;*****

```

```

;延时子程序

```

```

;*****

```

```

DELAY:    MOV    R4, #03H

```

```

AA1:      MOV    R5, #0FFH

```



```
AA:      DJNZ    R5, AA
          DJNZ    R4, AA1
          RET
```

```
;*****
```

```
;延时子程序 1S
```

```
;*****
```

```
DELAY1S: MOV     R3, #1
AX2:      MOV     R4, #79H
AX1:      MOV     R5, #78H
          DJNZ    R5, $
          DJNZ    R4, AX1
          DJNZ    R3, AX2
          RET
```

```
;*****
```

```
TABLE:    DB      3FH, 06H, 5BH, 4FH, 66H   ;段码表   不带小数点
          DB      6DH, 7DH, 07H, 7FH, 6FH
```

```
;TABLE:DB  0BFH, 86H, 0DBH, 0CFH, 0E6H, 0EDH, 0FDH, 87H, 0FFH, 0EFH   ;带小数点共阴
          END
```

第三部分 Proteus 绘制 PCB 图

借助 KeilC 和 Proteus，我们可大大缩短单片机系统的开发周期，降低开发成本。当仿真调试成功后，我们还可以利用 Proteus 中的 ARES 进行 PCB 设计与制作。这一部分我们“实验 10 综合实验—电子时钟温度计”（如图 3-1 所示）为例，将该实验深入下去，利用 Proteus 制作 PCB。

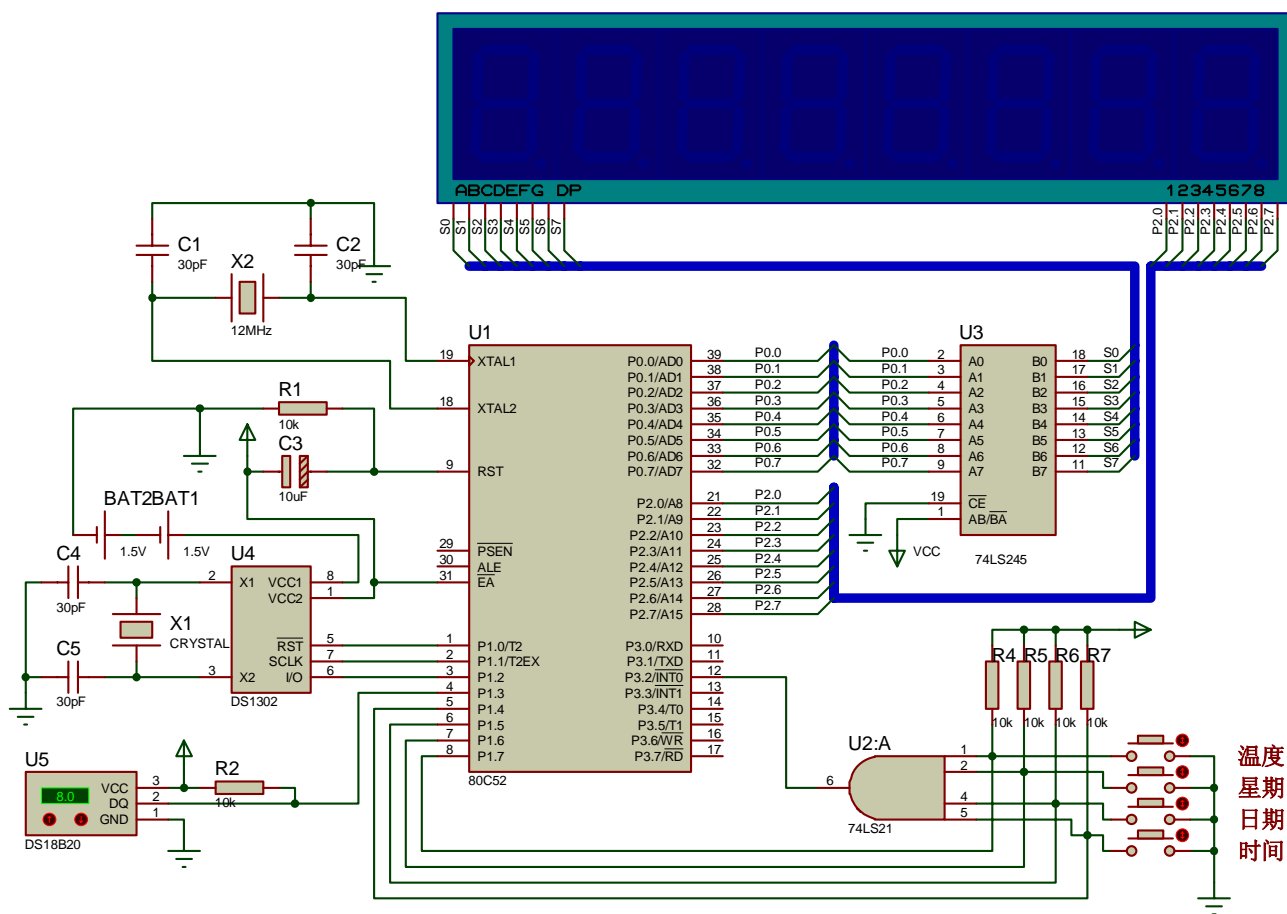


图 3-1 “综合实验-电子时钟温度计”实验原理图

用 Proteus 制作 PCB 通常包括以下一些步骤：

- 1、绘制电路原理图并仿真调试；
- 2、加载网络表及元件封装；
- 3、规划电路板并设置相关参数；
- 4、元件布局及调整；
- 5、布线并调整；
- 6、输出及制作 PCB。

一、绘制电路原理图及调试仿真

这点在前面的实验过程中已经多次涉及，这里不在赘述。但需要特别说明的是，Proteus 是一个很好的仿真软件，也正是这一点，在从原理图到 PCB 图的转换过程中容易出问题。原因就在于，及时单片机的外围电路有问题，如外部震荡电路、电路等有问题，仿真依然是可以进行的。这就要求我们在从原理图转向绘制 PCB 图时，必须要先检查原理图的完整性，这一点非常重要。否则，依照错误的 PCB 图制作的电路板但制造出来，就后悔莫及了。下面，我们就讲述一下，“综合实验-电子时钟温度计”原理图元件的加载。

二、加载网络表及元件封装

在 ISIS 7.1 Professional 界面中单击  按钮或执行工具菜单的网表到 ARES 命令，系统将自动打开 ARES 7.1 Professional 窗口（如图 3-2 所示），

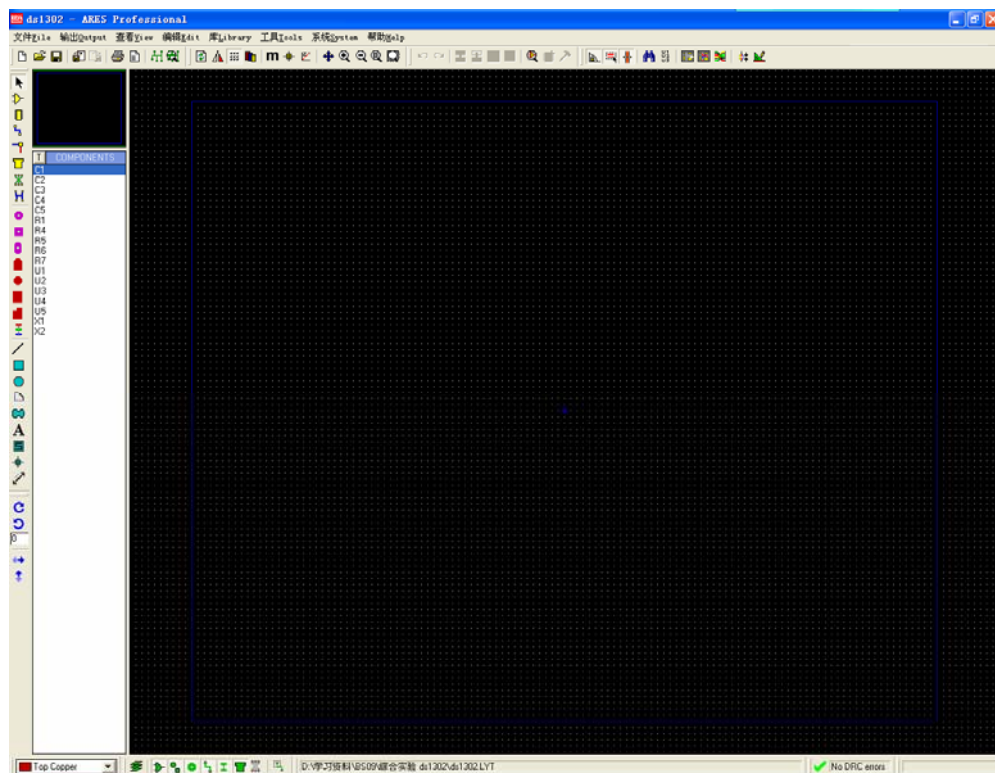


图 3-2 ARES 7 Professional 界面

在元件列表框中看到从 ISIS 7.1 Professional 中加载过来的所有原理图中包含的元件，如图 3-3 所示：

若原理图中的某些器件没有自动加载封装或者封装库中没有合适的封装，那么在加载网

络表时就会弹出一个要求选择封装的对话框，如图 3-4 所示。这时就需要根据具体的元件及其封装进行手动选择并加载。

由于综合实验中所有元件不需要封装，在这里我们不介绍元件封装的步骤。如果想进一步了解元件封装的过程可以查看相关资料。

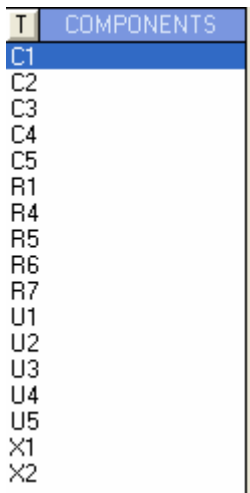


图 3-3 ARES 7 Professional 元件列表窗口

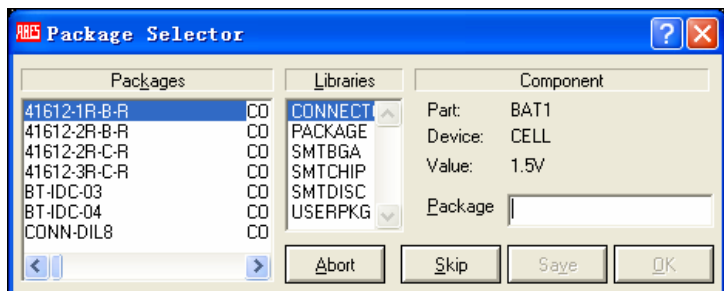



图 3-4 元件封装对话框

三、规划电路板并设置相关参数

1、规划电路板

在 ARES 7.1 Professional 窗口中选 2D 画图工具栏的  图标，在底部的电路层中（如图 3-5 所示）选中 Board Edge 层（黄色），即可以单击鼠标左键拖画出 PCB 板的边框了。

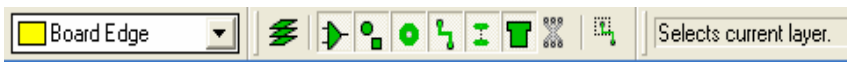



图 3-5 边框选择界面

在设置边框的时候可以通过  按钮或查看菜单中的源点 Origin 选项（快捷键为 O）来

设置边框的源点。边框的大小就是 PCB 板的大小，所以在画边框时应根据实际来测量所需大小，也可以再布置元件后调整。边框的绘制如图 3-6 所示：

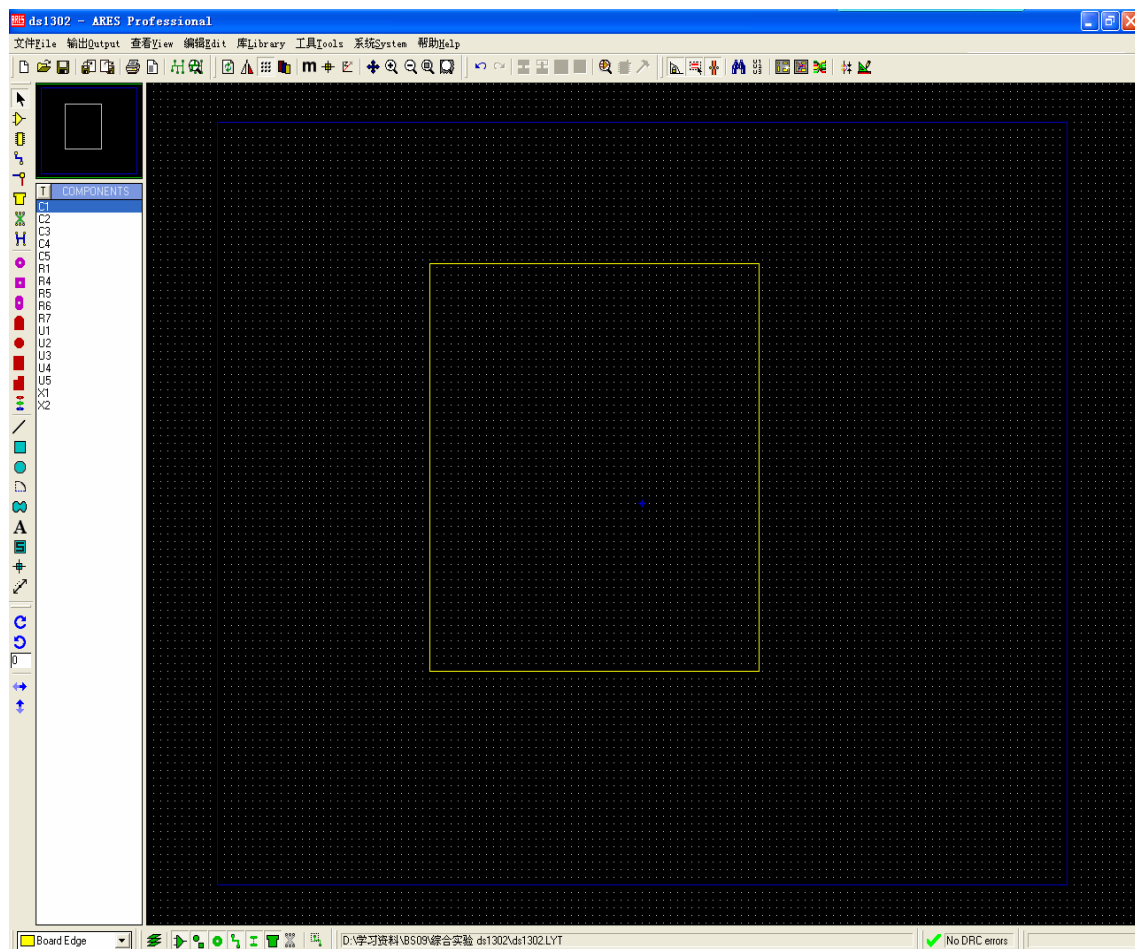


图 3-6 边框绘制界面


2. 设置电路板相关参数

PCB 板边框画好以后，就要设置电路板的相关参数。单击 System 中的 Set Default Rules 项，在弹出的对话框（如图 3-7 所示）中设置规则参数，有焊盘间距、线与焊盘间距、线与线间距等一些安全允许值。

四、元件布局及调整

1、元件布局

电路板的规则设计好以后，就可导入元件并布局。布局有自动布局和手动布局两种方式。

若采用自动布局方式，只要在界面的菜单栏中选中  项，弹出对话框（如图 3-8 所示）。在弹出的对话框中可以设置自动布局中各元件之间的距离和一些线宽等基本信息。单击 OK，

就自动把元件布局于 PCB 板中了。

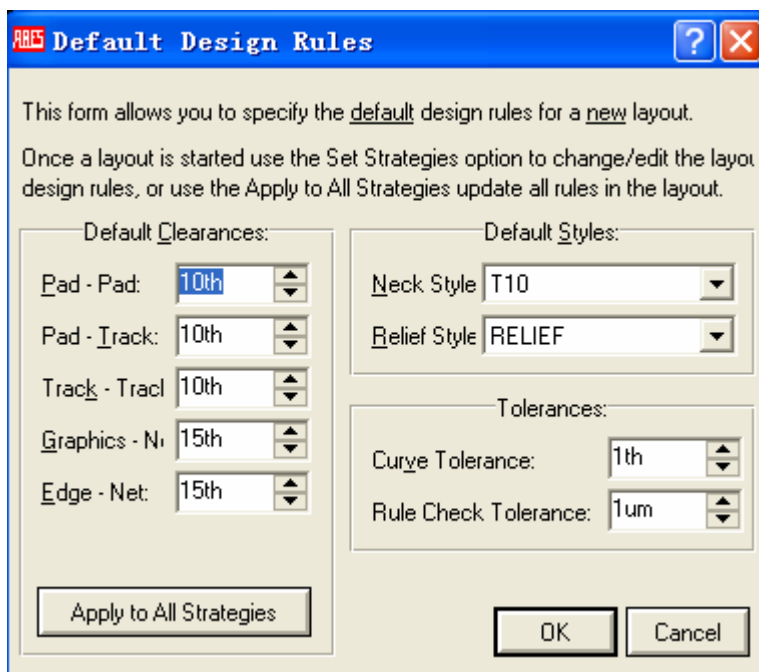


图 3-7 系统默认设置

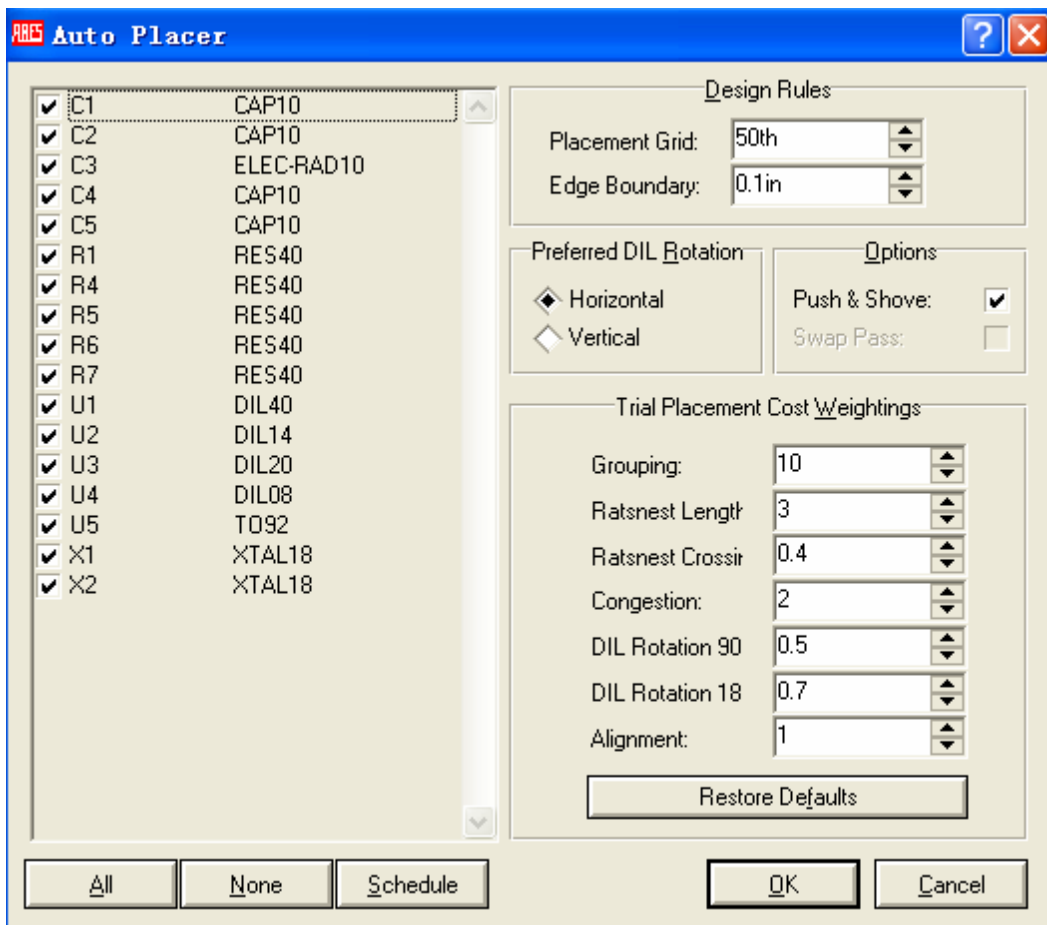



图 3-8 自动布局对话框

在自动布局对话框中左侧的元件选中为自动布局元件列表，可以通过 ALL 选项来完成所有元件自动布局，也可以自主选择哪些元件需要自动布局。一般情况下电阻类元件自动布局后会使布局美观整齐。右侧的选项为一些自动布局设置选项包括群组，元件间隔等。

而如果采用手动布局的方式，选中  项，选择所要布局的元件，在 PCB 板边框中适当位置单击左键，就可以把元件放入。

要想完成所有布局，一般情况下都要采用自动布局 and 手动布局相结合的方法。

2、元件调整

无论是自动布局还是手动布局，都需要对元件进行调整。主要是对元件的移动和翻转等操作。对元件的布局原则是：美观、便于布线、PCB 板尽可能小。PCB 的元件布局完成如图 3-9 所示。

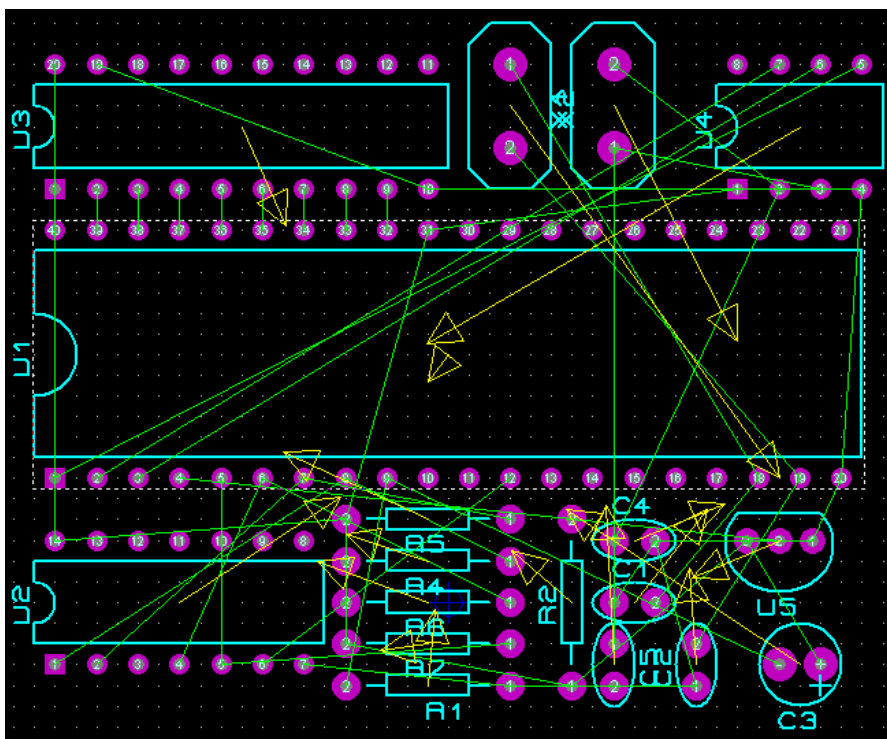



图 3-9 PCB 元件布局结果

五、布线及调整

和元件布局一样，PCB 的布线也是有自动布线和手动布线两种布线方式。一般，是先自动布线，然后手工修改，当然也可以直接手工布线。布线的原则是使效果美观。

点击按钮或工具菜单中自动布线 Auto Router 选项，将弹出如图 3-10 所示对话框，可以通过其中的选项设置线宽，所需自动布线元件和其他一些具体选项。单击 OK 即可对元件进行自动布线。

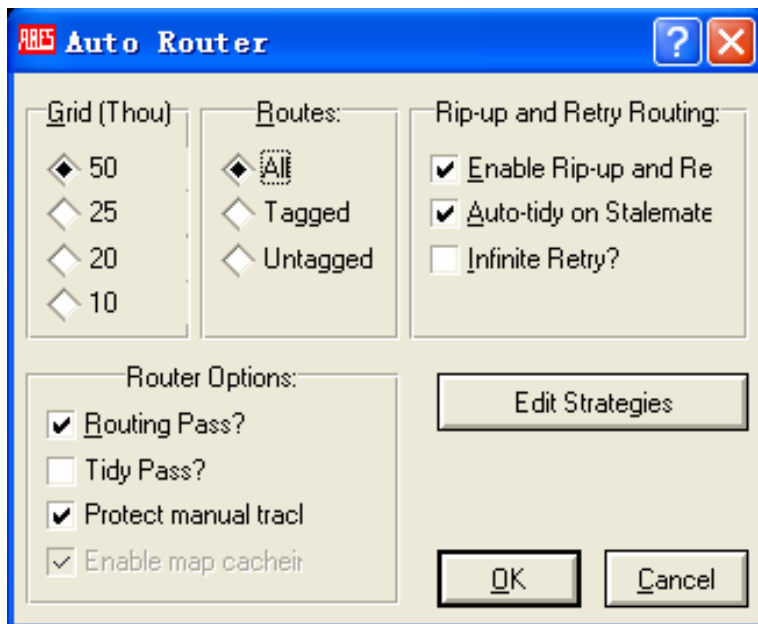





图 3-10 自动布线对话框

关于手动布线其中有两个选项十分重要，他们是位于 ARES 界面的左侧的连线按钮和拐点按钮.

使用连线按钮可以对自动布线后不合适的按钮进行修改，在布线的过程中，如果需要改变某一根线的大小，可以双击右键，选择 Trace Style 选项中的合适类型；要删除该线，则左键单击 Delete。

使用拐点按钮可以再 PCB 中加入拐点使得布线美观减少交叉。布线和布局完成后根据元件占用区间的大小调整边框的大小。布局完成图如图 3-11 所示：

布线完成后要进行铺铜操作，我们选择左侧按钮，沿着板的边框绘制一个矩形，这一矩形不能超出板的边框。在弹出的对话框（如图 3-12 所示）中：

选择 Net 为 VCC/VDD=POWER，Layer/Colour 为顶层红色，其他为默认。然后在用铺铜按钮绘制一个相同的矩形，在弹出的对话框中选择 Net 为 GND=POWER，Layer/Colour 为底层蓝色，其他为默认。这样就完成了铺铜工作。

铺铜完成后如图 3-13 所示：

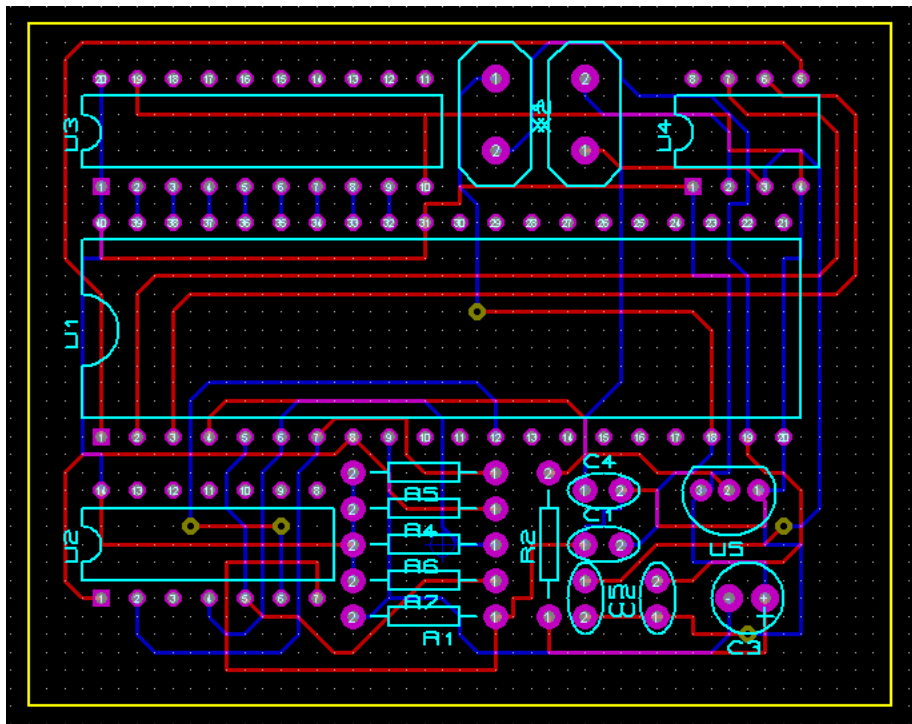


图 3-11 PCB 布局完成图

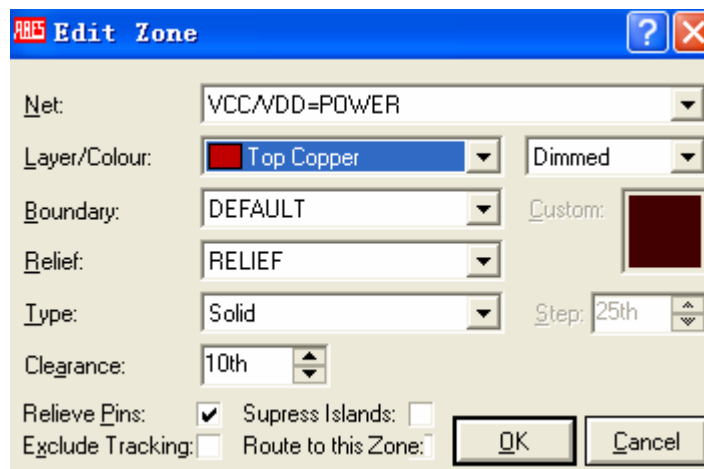


图 3-12 铺铜选项对话框

六、输出并制作 PCB

PCB 板完成以后，我们要进行输出操作。选择输出菜单下的“输出 Gerber 文件”选项。弹出的如图 3-14 所示对话框中选中。左侧为输出层选项，选中如图所示 3-14 的选项，其他默认即可。单击 OK 我们就可以生成 Gerber 文件。

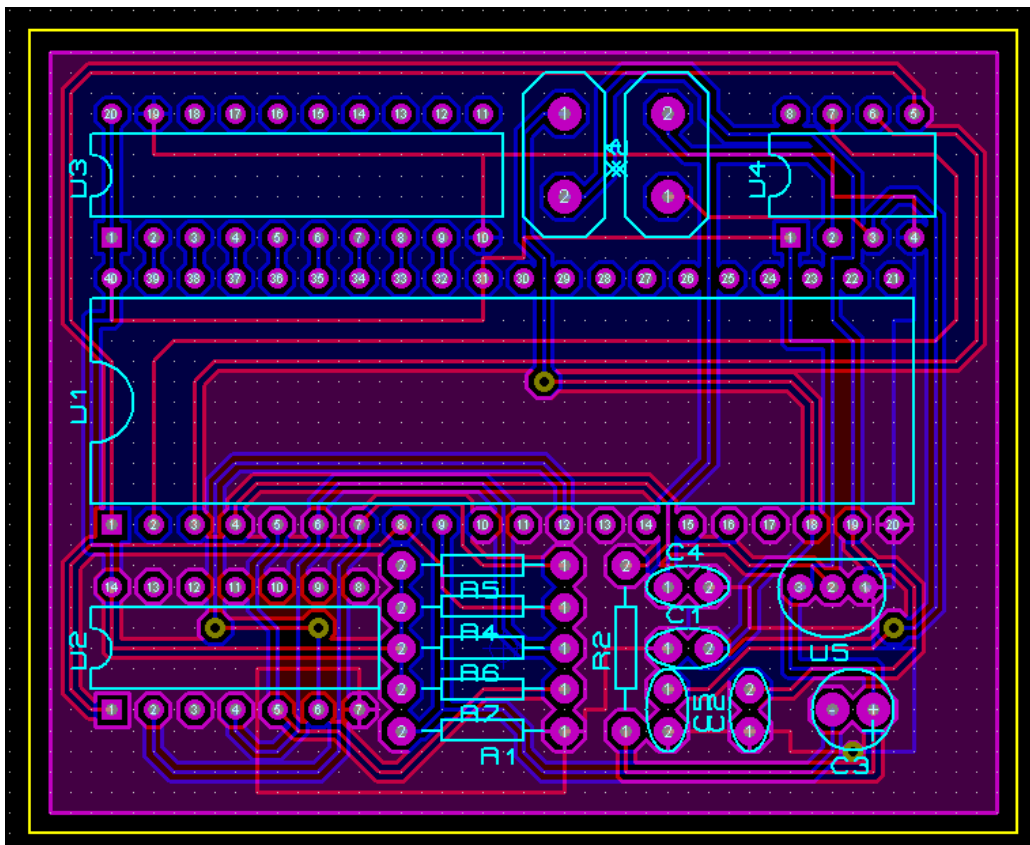


图 3-13 PCB 板完成图

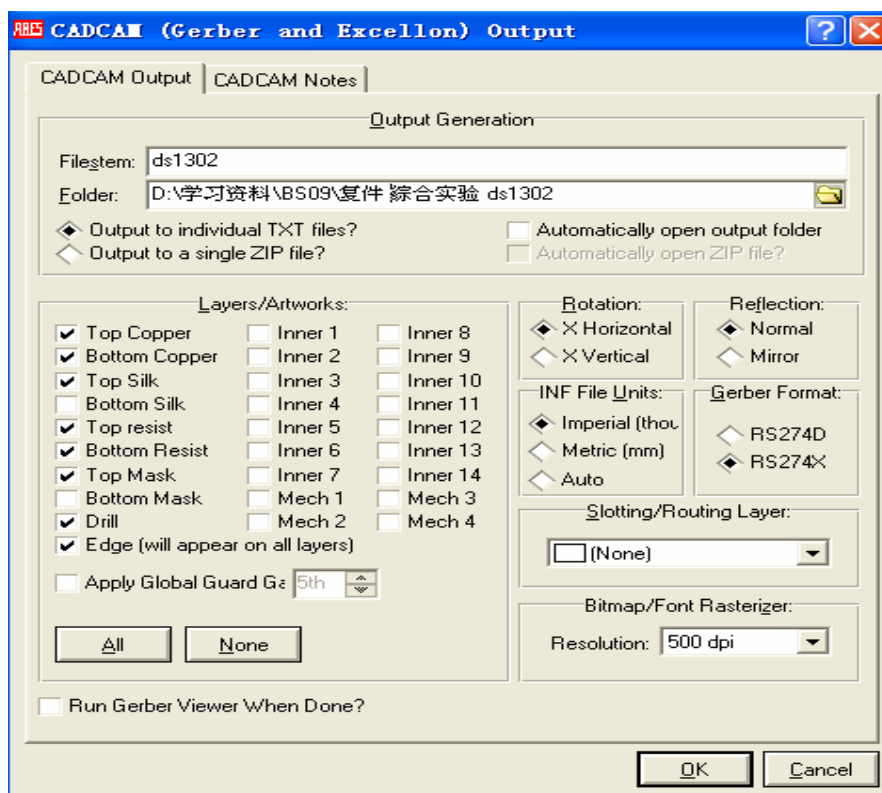


图 3-14 输出 CAD/CAM 文件选项对话框

使用这个文件我们就可以到 PCB 制作厂商进行电路板的生产。一般情况下都将制作的 Gerber 文件进行压缩打包处理。所以本文中课程也对生成的文件做成压缩文件。制作后的文件如图 3-15 所示,

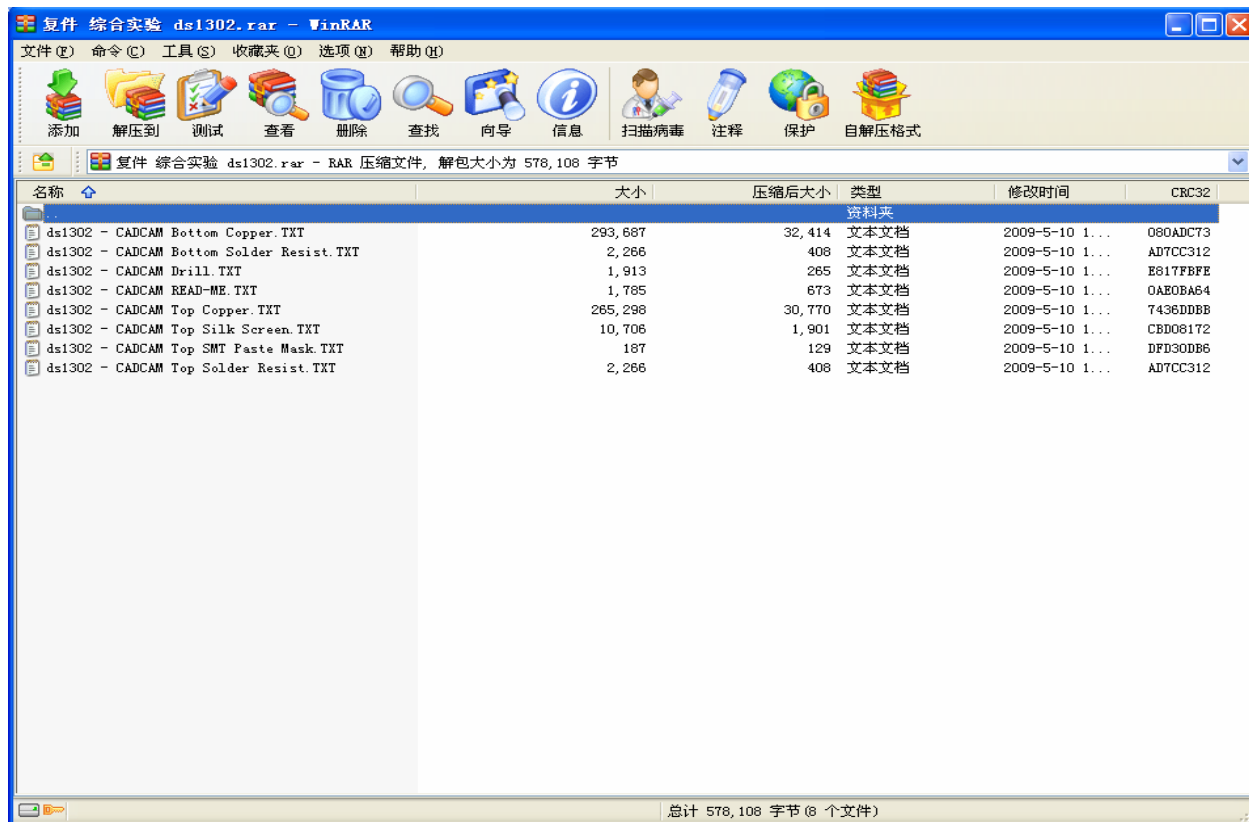


图 3-15 生成的 Gerber 文件压缩文件

这样, PCB 板就基本制作完成了。

第四部分 51 单片机的 C 语言编程

当设计一个小的嵌入式系统时一般我们都用汇编语言在很多工程中这是一个很好的方法，因为代码一般都不超过8K，而且都比较简单如果硬件工程师要同时设计软件和硬件经常会采用汇编语言来做程序。使用汇编的麻烦在于它的可读性和可维护性，特别当程序没有很好的标注的时候。代码的可重用性也比较低。如果使用C的话可以很好的解决这些问题^[3]。

用C编写的程序因为C语言很好的结构性和模块化更容易阅读和维护而且由于模块化用C语言编写的程序有很好的可移植性功能化的代码能够很方便的从一个工程移植到另一个工程从而减少了开发时间。

使用像C这样的语言，程序员不必十分熟悉处理器的运算过程。这意味着对新的处理器也能很快上手，不必知道处理器的具体内部结构，使得用C编写的程序比汇编程序有更好的可移植性。很多处理器支持C编译器

所有这些并不说明汇编语言就没了立足之地。很多系统特别是实时时钟系统都是用C 和汇编语言联合编程，对时钟要求很严格时使用汇编语言成了唯一的方法。除此之外包括硬件接口的操作都应该用C来编程。C的特点就是可以使你尽量少地对硬件进行操作，是一种功能性和结构性很强的语言。

下面简要讲解一下使用C语言对51单片机编程的相关知识点。

一、数据类型

KeilC有ANSI C的所有标准数据类型。除此之外，为了更加有利的使用8051的结构，还加入了一些特殊的数据类型。注意整型和长整型的符号位字节在最低的地址中。除了这些标准数据类型外，编译器在内部RAM的可位寻址区中还支持位数据类型，可像操作其它变量那样对位变量进行操作，但位数组和位指针是违法的。

二、存储类型

Keil允许使用者指定程序变量的存储区这使使用者可以控制存储区的使用编译器可识别以下存储区

1 DATA 段

对DATA 区的寻址是最快的。一般都把使用频率最高的变量放在DATA 区。由于空间有限，必须注意使用DATA区除了包含程序变量外，还包含了堆栈和寄存器组。

标准变量和用户自定义变量都可存储在DATA区中，只要不超过DATA区的范围因。为C51使用默认的寄存器组来传递参数你至少失去了8 个字节。另外要定义足够大的堆栈空间，当你的内部堆栈溢出的时候，你的程序会莫名其妙的复位。实际原因是8051系列微处理器没有硬件报错机制，堆栈溢出只能以这种方式表示出来。

表4-1 C语言在KeilC中的存储区描述

存储区	描述
DATA	RAM的低128个字节可在以个周期内直接寻址
BDATA	DATA区的16个字节的可位寻址区
IDATA	52子系列RAM区的高128字节必须采用间接寻址
PDATA	外部存储区的256个字节通过P0口的地址对其寻址。只用MOVX @Rn，需要两个指令周期
XDATA	外部存储区使用DPTR寻址
CODE	程序存储区使用DPTR寻址

2 BDATA 段

在DATA区的位寻址区可以定义变量，这个变量就可进行位寻址，并且声明位变量。这对状态寄存器来说是十分有用的。因为它需要单独的使用变量的每一位，不一定要用位变量名来引用位变量。

3 IDATA 段

IDATA段也可存放使用比较频繁的变量。使用寄存器作为指针进行寻址在寄存器中，设置8位地址进行间接寻址和外部存储器寻址比较。它的指令执行周期和代码长度都比较短。注意该存储区域特指52子系列高128字节。

4 PDATA 和 XDATA 段

在这两个段声明变量和在其它段的语法是一样的。PDATA段只有256个字节，而XDATA段可达65536个字节。这两个存储段都属于外部数据存储区域。

5 CODE 段

代码段的数据是不可改变的。8051的代码段不可重写一般代码段中可存放数据表跳转向量和状态表。对CODE段的访问和对XDATA段的访问的时间是一样的。代码段中的对象在编译的时候初始化，否则你就得不到你想要的值。

三、指针

80C51提供一个3字节的通用存储器指针。通用指针的头一个字节表明指针所指的存储区空间,另外两个字节存储16位偏移量。对于DATA、IDATA和PDATA段只需要8位偏移量。KeilC允许使用者规定指针指向的存储段,这种指针叫具体指针。使用具体指针的好处是节省了存储空间,编译器不用为存储器选择和决定正确的存储器操作指令产生代码。这样就使代码更加简短,但你必须保证指针不指向你所声明的存储区以外的地方,否则会产生错误,而且很难调试。指针类型表如表4-2所示。

表4-2 C语言控制80C51指针类型表

指针类型	大小
通用指针	3字节
XDATA指针	2字节
CODE	2字节
IDATA指针	1字节
DATA指针	1字节
PDATA指针	1字节

四、中断

8051的中断系统十分重要。C51使你能够用C来声明中断和编写中断服务程序。中断过程通过使用interrupt关键字和中断号(0到31)来实现。中断号告诉编译器中断程序的入口地址,中断号对应着IE寄存器中的使能位。换句话说IE寄存器中的0位对应着外部中断0,相应的外部中断0的中断号是0。表4-3反映了这种关系。

表4-3 C语言中的中断号与80C51中断源的对应关系

IE寄存器中的使能位和C中的中断号	中断源
0	外部中断0
1	定时器0中断
2	外部中断1
3	定时器1中断
4	串行中断
5	定时器2中断

一个中断过程并不一定带上所有参数。可以没有返回值,有了这些限制编译器不须要担心寄存器组参数的使用和对累加器状态寄存器, B寄存器, 数据指针和默认的寄存器的保护。只要他们在中断程序中被用到, 编译的时候会把他们入栈。在中断程序结束时将他们恢复。

中断程序的入口地址被编译器放在中断向量中。C51支持所有5个8051/8052标准中断从0到4和在8051系列中多达27个中断源。

五、例程说明

在介绍过如何使用C语言控制80C51后，我们用一个简单的例子来对上述过程进行熟悉。一个中断服务程序的例子如下：

```
#include <reg51.h>
#define RELOADVALH 0x3C
#define RELOADVALL 0xB0
extern unsigned int tick_count;
void timer0 (void) interrupt 1 using 0
{
    TR0=0; // 停止定时器0
    TH0=RELOADVALH; // 设定溢出时间50ms
    TL0=RELOADVALL;
    TR0=1; // 启动T0
    tick_count++; // 时间计数器加1
}
```

在进行编译后产生的汇编代码如下：

```
PUSH ACC
PUSH B
PUSH DPH
PUSH DPL
PUSH PSW
MOV PSW,#00H
CLR TR0
MOV TH0,#03CH
MOV TL0,#0B0H
SETB TR0
MOV DPTR,#tick_count+01H
MOVBX A,@DPTR
INC A
MOVBX @DPTR,A
JNZ ?C0002
```

```
MOV DPTR,#tick_count
MOVX A,@DPTR
INC A
MOVX @DPTR,A
MOV R3,#05H
MOV R2,#HIGH ?SC_0
MOV R1,#LOW ?SC_0
MOV DPTR,#tick_count
MOVX A,@DPTR
MOV R7,A
INC DPTR
MOVX A,@DPTR
MOV DPTR,#?_printf?BYTE+03H
XCH A,R7
MOVX @DPTR,A
INC DPTR
MOV A,R7
MOVX @DPTR,A
LCALL _printf
POP PSW
POP DPL
POP DPH
POP B
POP ACC
RETI
```

希望通过上述过程的讲解，可以对C语言在单片机技术中的应用有一个初步的认识。更具体的内容可以参考相关书目。只有通过大量的练习和具体系统功能的开发实现，才能更好的掌握单片机C语言的编程。

附录 A：51 系列单片机指令系统

一、数据传送类指令

数据传送类指令共 29 条

- 内部 RAM 间的数据传送指令（16 条）
- 访问外部 RAM 的数据传送指令（4 条）
- 访问 ROM 程序存储器的数据传送指令（2 条）
- 交换类指令（5 条）
- 堆栈指令（2 条）

通用格式为：MOV <目的操作数>，<源操作数>

传送指令的约定：从右向左传送数据，是将源操作数送到目的操作数。指令执行后，源操作数不变，目的操作数被源操作数取代。

数据传送类指令用到的助记符有 MOV、MOVB、MOVC、XCH、XCHD、SWAP、PUSH、POP 共 8 种。

<一>以 A 为目的操作数的指令（4 条）

MOV	A, Rn	; A ← (Rn)	如：MOV	A, R2
MOV	A, direct	; A ← (direct)	如：MOV	A, 30H
			MOV	A, 0E3H
MOV	A, @Ri	; A ← ((Ri))	如：MOV	A, @R0
MOV	A, #data	; A ← data	如：MOV	A, #36H
			MOV	A, #0D2H

<二>以 Rn 为目的操作数的指令（3 条）

MOV	Rn, A	; (Rn) ← (A)	如：MOV	R0, A
MOV	Rn, direct	; (Rn) ← (direct)	如：MOV	R3, 30H
MOV	Rn, #data	; (Rn) ← data	如：MOV	R7, #36H
			MOV	R1, #30
			MOV	R6, #01101100B

<三>以直接地址为目的操作数的指令（5 条）

MOV	direct, A	; direct ← (A)	如：MOV	30H, A
-----	-----------	----------------	-------	--------

MOV	direct, Rn	;	direct ← (Rn)	如: MOV	P1, R2
MOV	direct1, direct2	;	(direct1) ← (direct2)	如: MOV	38H, 60H
MOV	direct, @Ri	;	direct ← ((Ri))	如: MOV	TL0, @R1
MOV	direct, #data	;	direct ← data	如: MOV	58H, #36H

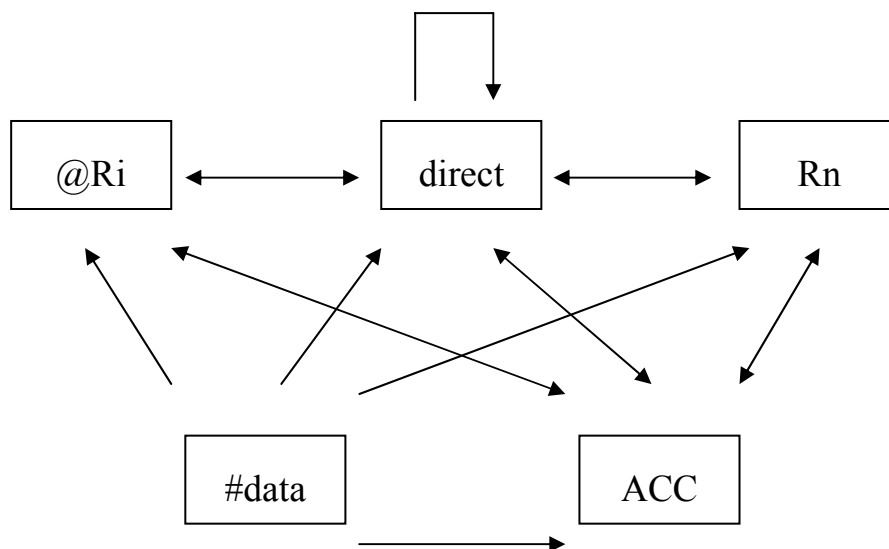
〈四〉以间接地址为目的操作数的指令（3 条）

MOV @Ri, A ; ((Ri)) ← (A)	如: MOV @R0, A
MOV @Ri, direct ; ((Ri)) ← (direct)	如: MOV @R1, 36H
	MOV @R0, SBUF
MOV @Ri, #data ; ((Ri)) ← data	如: MOV @R1, #48
	MOV @R0, #0D6H

MOV 指令在片内 RAM 的允许操作图

不允许的操作有:

$@Ri \leftarrow \rightarrow @Ri$	$Rn \leftarrow \rightarrow Rn$
$@Ri \leftarrow \rightarrow Rn$	立即数作为目的操作数



传送指令在片内储存器的操作功能

<五>十六位数据传送指令（1 条）

```
MOV    DPTR, #data16;    (DPH) ← dataH8 , (DPL) ← dataL8
如: MOV    DPTR, #2368H  ⇔  MOV    DPH, #23H
                                MOV    DPL, #68H
                                MOV    DPTR, #35326
```

<六>程序存储器数据传送指令（查表指令）（2条）

MOVC A, @A+DPTR ; $A \leftarrow ((A) + (DPTR))$

一般 DPTR 放表的首地址, A 放所查数据在表中的偏移; 查表范围为 64KB 空间, 称为远程查表。

MOVC A, @A+PC ; $A \leftarrow ((A) + (PC))$

PC 的值为将要执行的下一条指令的地址, A 放所查数据相对 PC 值的偏移; 查表范围为最大为 256B 空间, 称为近程查表。

注意: 内外存储器在逻辑上连续统一, 传送单向, 只读。

这类指令用于访问程序存储器中的数据表格。

<七>累加器 A 与片外 RAM 的数据传送指令 (4 条)

用间址方式

MOVX A, @Ri ; $A \leftarrow ((Ri))$, 且使/RD=0

MOVX A, @DPTR ; $A \leftarrow ((DPTR))$, 且使/RD=0

MOVX @Ri, A ; $((Ri)) \leftarrow (A)$, 且使/WR=0

MOVX @DPTR, A ; $((DPTR)) \leftarrow (A)$, 且使/WR=0

注意: 1、外部数据的传送只能使用 A

1、@DPTR 两条指令以 DPTR 为片外 RAM16 位地址指针, 寻址范围为 64KB 空间; P2 输出高 8 位地址, P0 口输出低 8 位地址和数据, 分时复用

2、@Ri 两条指令以 R0 或 R1 作低 8 位地址指针, 由 P0 口送出, 寻址范围为 256B 空间 (P2 口仍可作通用 I/O 口)。

<八>堆栈操作指令 (2 条)

PUSH direct ; 先 $(SP)+1 \rightarrow SP$, 后 $(direct) \rightarrow ((SP))$

POP direct ; 先 $((SP)) \rightarrow direct$, 后 $(SP)-1 \rightarrow SP$

如: PUSH 0E0H ; 实际是 $(SP)+1 \rightarrow SP, (A) \rightarrow ((SP))$

POP 05H ; 实际是 $((SP)) \rightarrow R5, (SP)-1 \rightarrow SP$

注意: 其操作数为直接地址, 不能用寄存器名。

POP ACC 正确

POP A 错误

<九>交换指令 (5 条)

1、字节交换指令

XCH A, Rn ; $(A) \leftrightarrow (Rn)$

XCH A, direct ; $(A) \leftrightarrow (direct)$

XCH A, @Ri ; $(A) \leftrightarrow ((Ri))$

2、半字节交换指令

(1) 二低半字节交换指令

XCHD A, @Ri ; $(A)_{0\sim3} \leftrightarrow ((Ri))_{0\sim3}$

(2) 累加器 A 高、低半字节交换指令

SWAP A ; $(A)_{0\sim3} \leftrightarrow (A)_{4\sim7}$

如: 设 $(A) = 36H$

SWAP A ; $(A) = 63H$

二、算数运算累指令

算术运算指令包括加、减、乘、除运算，针对 8 位无符号数；第一操作数一般为 A；一般影响 PSW 的标志位 CY、AC、OV 和 P。共 24 条指令，分成七个小类

<一> 不带进位加法指令（4 条）

```
ADD    A, Rn           ; (A) ← (A) + (Rn)
ADD    A, direct       ; (A) ← (A) + (direct)
ADD    A, @Ri          ; (A) ← (A) + ((Ri))
ADD    A, #data        ; (A) ← (A) + #data
```

注意：如果位 3 有进位，则半进位标志位 AC 置 1，否则清零。
 如果位 7 有进位，则进位标志位 CY 置 1，否则清零。
 如果两个数（看作有符号数时）相加溢出，则 OV 置 1，否则清零。
 如果两个同符号数相加的结果变号即溢出，则 OV 置 1，否则清零。
 两个同符号数相加的结果变号，则 OV 置 1，否则清零。

<二> 带进位加法指令（4 条）

```
ADDC   A, Rn           ; (A) ← (A) + (Rn) + CY
ADDC   A, direct       ; (A) ← (A) + (direct) + CY
ADDC   A, @Ri          ; (A) ← (A) + ((Ri)) + CY
ADDC   A, #data        ; (A) ← (A) + #data + CY
```

<三> 带借位减法指令（4 条）

```
SUBB   A, Rn           ; (A) ← (A) - CY - (Rn)
SUBB   A, direct       ; (A) ← (A) - CY - (direct)
SUBB   A, @Ri          ; (A) ← (A) - CY - ((Ri))
SUBB   A, #data        ; (A) ← (A) - CY - #data
```

注意：如果位 3 有借位，则半进位标志位 AC 置 1，否则清零。
 如果位 7 有借位，则进位标志位 CY 置 1，否则清零。
 如果两个数（看作有符号数时）相减溢出，则 OV 置 1，否则清零。
 如果两个异号数相减的结果与被减数符号不同即溢出，OV 置 1，否则清零。
 无不带借位减法指令，需要时，先执行一条 CLR C 指令既可。

<四> 加 1 指令（5 条）

```
INC    A               ; (A) ← (A) + 1
INC    Rn              ; (Rn) ← (Rn) + 1
INC    direct          ; (direct) ← (direct) + 1
INC    @Ri             ; ((Ri)) ← ((Ri)) + 1
```

INC DPTR ; (DPTR) \leftarrow (DPTR) + 1

说明: INC A 影响 P 外, 不影响 PSW 其它位(即标志位 CY、AC、OV)。

<五> 减 1 指令 (4 条)

DEC A ; (A) \leftarrow (A) - 1

DEC Rn ; (Rn) \leftarrow (Rn) - 1

DEC direct ; (direct) \leftarrow (direct) - 1

DEC @Ri ; ((Ri)) \leftarrow ((Ri)) - 1

DEC DPTR ; 无此指令

说明: DEC A 影响 P 外, 不影响 PSW 其它位(即标志位 CY、AC、OV)。

<六> 乘除法指令 (2 条, 单字节四周期指令)

1、乘法指令

MUL AB ; (A) \times (B) \rightarrow B15~8, A7~0

说明: (1) 为无符号乘法;

(2) 若结果的 B \neq 0, 则 OV=1, 若 B=0, 则 OV=0; CY 总是清零, AC 不影响, P 影响。

2、除法指令

DIV AB ; (A)/(B) 的商 \rightarrow A, 余数 \rightarrow B

说明: (1) 为无符号除法;

(2) 若除数 B=0, 则 OV=1, 若 B \neq 0, 则 OV=0; CY 总是清零, AC 不影响, P 影响。

<七> 十进制数调整指令 (1 条)

DA A ; 调整累加器内容为 BCD 码(二进制编码的十进制)

说明:

(1) 此指令跟在 ADD 或 ADDC 指令之后, 将 A 中的和调整为 BCD 码, 并且 ADD 或 ADDC 的两个操作数是 BCD 码, 即两个 BCD 码相加必须经过 DAA 指令后, 才能得到正确的 BCD 码的结果。

(2) 调整方法:

若(A0~3) > 9 或 AC=1, 则(A0~3) + 6 \rightarrow (A0~3);

若(A4~7) > 9 或 CY=1, 则(A4~7) + 6 \rightarrow (A4~7);

若(A4~7) = 9 且(A0~3) > 9, 则(A) + 66H \rightarrow (A);

(3) 对标志的影响: 若结果 A > 99H, 则 CY=1; 不影响 OV、AC, 影响 P。

执行 DA A 后, CPU 根据累加器的值, 以及 AC、CY 的状态, 来自动的对 A 进行修正, 不需人为的干预。

注意:

1、DA 指令不对减法进行修正

2、借助 CY, 可以实现多位 BCD 码的加法

三、逻辑运算类指令

逻辑运算指令包括与、或、异或、清除、求反、移位等操作。这类指令一般不影响标志位 CY、AC 和 OV。共 24 条指令, 分成五个小类。

<一> 逻辑“与”指令（6 条）

```
ANL    A, Rn           ; (A) ← (A) ∧ (Rn)
ANL    A, direct       ; (A) ← (A) ∧ (direct)
ANL    A, @Ri          ; (A) ← (A) ∧ ((Ri))
ANL    A, #data        ; (A) ← (A) ∧ #data
ANL    direct, A       ; (direct) ← (direct) ∧ (A)
ANL    direct, #data   ; (direct) ← (direct) ∧ #data
```

说明：

- (1) 目的操作数只能是 A 或者 direct;
- (2) 前 4 条指令仅影响标志位 P；后两条不影响标志位；
- (3) 与运算常用于使某些位清 0。

<二> 逻辑“或”指令（6 条）

```
ORL    A, Rn           ; (A) ← (A) ∨ (Rn)
ORL    A, direct       ; (A) ← (A) ∨ (direct)
ORL    A, @Ri          ; (A) ← (A) ∨ ((Ri))
ORL    A, #data        ; (A) ← (A) ∨ #data
ORL    direct, A       ; (direct) ← (direct) ∨ (A)
ORL    direct, #data   ; (direct) ← (direct) ∨ #data
```

说明：

- (1) 目的操作数只能是 A 或者 direct;
- (2) 前 4 条指令仅影响标志位 P；后两条不影响标志位。
- (3) 或运算常用于使某些位置 1。

<三> 逻辑“异或”指令（6 条）

```
XRL    A, Rn           ; (A) ← (A) ⊕ (Rn)
XRL    A, direct       ; (A) ← (A) ⊕ (direct)
XRL    A, @Ri          ; (A) ← (A) ⊕ ((Ri))
XRL    A, #data        ; (A) ← (A) ⊕ data
XRL    direct, A       ; (direct) ← (direct) ⊕ (A)
XRL    direct, #data   ; (direct) ← (direct) ⊕ #data
```

说明：

- (1) 目的操作数只能是 A 或者 direct;
- (2) 前 4 条指令仅影响标志位 P；后两条不影响标志位。
- (3) 用 1 异或使对应位取反，用 0 异或使对应位不变，异或运算常用于使某些位取反。

<四> 累加器 A 清 0 与取反指令（2 条）**1、累加器 A 清 0 指令**

```
CLR    A           ; A ← 0
```

说明：只影响标志位 P。

2、累加器 A 取反指令（按位取反）

```
CPL    A           ; (A) ← (/A) , 相当于 0FFH - A → A
```

说明：不影响标志位。

<五> 移位指令（4 条）

1、累加器 A 循环左移

RL A ;

2、累加器 A 循环右移

RR A ;

3、累加器 A 带进位位循环左移

RLC A ;

4、累加器 A 带进位位循环右移

RRC A ;

说明：

- (1) 各条指令每次只移动一位；
- (2) 左移一位相当于乘以 2；右移一位相当于除以 2；
- (3) 带进位位移动的影响标志位 CY 和 P。

四、控制转移类指令

控制转移类指令包括无条件转移、条件转移、子程序调用和返回指令等，共 17 条。只有比较转移指令影响标志。

<一> 无条件转移指令（4 条）

1、长转移指令

LJMP addr16 ; (PC) \leftarrow addr16

说明：转移范围：64KB 全程序空间任何单元。

如：LJMP NEXT

2、绝对转移指令

AJMP addr11 ; 先(PC) \leftarrow (PC)+2
; 再(PC10~0) \leftarrow addr11, (PC15~11) 不变

说明：(1) 两字节编码

(2) 该指令执行前 PC 值为下一条指令的首地址；

(3) 转移范围：含有下一条指令首地址的同一个 2KB 范围，即高 5 位地址相同；

3、相对转移（短转移）指令

SJMP rel ; 先(PC)+2 \rightarrow (PC), 后 (PC) +rel \rightarrow (PC)

说明：(1) 该指令执行前 PC 值为下一条指令的首地址；

(2) 转移范围：-128~+127；对应 rel 值为：00H~7FH (0~+127)、80H~FFH (-128~-1)；

4、间接转移指令

JMP @A+DPTR ; (A)+(DPTR)→(PC)

说明: (1) 具有多分枝转移功能, 即散转功能, 又叫散转指令;

(2) 转移范围：是以 DPTR 为首地址的 256B。

<二> 条件转移指令（8 条） 均为相对寻址方式。

1、累加器 A 为零（非零）转移指令

JZ rel ; 当 A=0 时, (PC)+2+rel→(PC) 转移; SJMP rel
; 当 A≠0 时, 顺序执行 (PC)+2→(PC)。

JNZ rel ; 当 A≠0 时, (PC)+2+rel→(PC) 转移; SJMP rel
; 当 A=0 时, 顺序执行 (PC)+2→(PC)。

2、比较转移指令

4 条, 均为三字节指令。一般形式为:

CJNE X1 (目的操作数), X2 (源操作数), rel

CJNE X1 , X2, rel
 ; 若 $X1 > X2$, 则 $(PC)+3+rel \rightarrow (PC)$, 且 $0 \rightarrow CY$;
 ; 若 $X1 < X2$, 则 $(PC)+3+rel \rightarrow (PC)$, 且 $1 \rightarrow CY$;
 ; 若 $X1 = X2$, 则顺序执行 $(PC)+3 \rightarrow (PC)$, 且 $0 \rightarrow CY$ 。

CJNE A, direct, rel
 ; 若 A>(direct) , 则(PC)+3+rel→ (PC) , 且 0→CY;
 ; 若 A<(direct) , 则(PC)+3+rel→ (PC) , 且 1→CY;
 ; 若 A=(direct) , 则顺序执行(PC)+3→ (PC) , 且 0→CY。

CJNE	A, #data, rel
	； 若 $A > \#data$, 则 $(PC) + 3 + rel \rightarrow (PC)$, 且 $0 \rightarrow CY$;
	； 若 $A < \#data$, 则 $(PC) + 3 + rel \rightarrow (PC)$, 且 $1 \rightarrow CY$;
	； 若 $A = \#data$, 则顺序执行 $(PC) + 3 \rightarrow (PC)$, 且 $0 \rightarrow CY$ 。

CJNE Rn, #data, rel
 ; 若 (Rn) > #data , 则(PC)+3+rel→PC, 且 0→CY;
 ; 若 (Rn) < #data , 则(PC)+3+rel→PC, 且 1→CY;
 ; 若 (Rn) = #data , 则顺序执行(PC)+3→PC , 且 0→CY。

CJNE @Ri, #data, rel
 ; 若((Ri)) >#data , 则(PC)+3+rel→PC, 且 0→CY;
 ; 若((Ri)) <#data , 则(PC)+3+rel→PC, 且 1→CY;
 ; 若((Ri)) =#data , 则顺序执行(PC)+3→PC , 且 0→CY。

3、循环转移指令

DJNZ	Rn, rel	; (Rn) -1 → Rn; ; 若(Rn)≠0, 则(PC)+2+rel → PC ; SJMP rel ; 若(Rn) = 0, 则结束循环, 顺序执行(PC)+2 → PC
DJNZ	direct, rel	; (direct) -1 → direct ; ; 若(direct)≠0, 则(PC)+3+rel → PC ; SJMP rel ; 若(direct) = 0, 则结束循环, 顺序执行(PC)+3 → PC

说明: Rn、direct 相当于控制循环的计数器。

<三> 子程序调用和返回指令（3 条）

1、短(绝对)调用指令

ACALL addr11 ; (PC)+2→(PC)
 ; (SP)+1→(SP), (PC_{0~7})→((SP))
 ; (SP)+1→(SP), (PC_{8~15})→((SP))
 ; addr11→(PC_{0~10}), (PC_{11~15} 不变) AJMP addr11

转移范围：含有下一条指令首地址 高 5 位的同一个 2KB 范围。

2、长调用指令

LCALL addr16 ; (PC)+3→(PC)
 ; (SP) +1→(SP), (PC 0~7)→ ((SP))
 ; (SP) +1→(SP), (PC 8~15)→ ((SP))
 ; addr16→(PC) LJMP addr16

说明： 转移范围：整个程序存储空间，64KB 范围。

3、返回指令

子程序返回指令

RET ; ((SP)) →PC 8~15 , (SP) -1→SP
 ; ((SP)) →PC 0~7 , (SP) -1→SP

中断服务程序返回指令

RETI ; ((SP)) →PC 8~15 , (SP) -1→SP
 ; ((SP)) →PC 0~7 , (SP) -1→SP
 ; 开放中断逻辑

空操作指令：

NOP ; (PC) +1→ (PC)

仍然取指令，并译码，但不做任何操作，只是为了耗费一个机器周期，作用：测试，延时

五、位操作类指令

位操作类指令包括位变量传送、逻辑运算、控制转移等指令，共 17 条，分成 4 个小类。只有部分指令影响 CY 标志。

位地址的表示方法：

- (1) 直接用位地址 如：D4H
- (2) 用特殊功能寄存器名加位数 如：PSW.4
- (3) 用 SFR 直接地址加位数 如：D0H.4
- (4) 用位名称 如：RS1
- (5) 用伪指令 bit 定义的有名字的位地址
 如：SUB.REG bit RS1, FLAGRUN bit 02H

<一> 位数据传送指令（2 条）

```
MOV    C, bit           ; (bit)→(C)
MOV    bit, C           ; (C)→(bit)
如: MOV    C, TR0
      MOV    08H, C
```

<二> 位修正指令（6 条）

1、位清 0 指令

```
CLR    C               ; 0→ (C)
CLR    bit             ; 0→ (bit)
如: CLR    TR0
```

2、位置 1 指令

```
SETB   C               ; 1→ (C)
SETB   bit             ; 1→(bit)
如: SETB   TR0
      SETB   06H
```

3、位取反指令

```
CPL    C               ; (/C) → (C)
CPL    bit             ; (/bit) → (bit)
如: CPL    TR0
      CPL    EA
```

<三> 位逻辑运算指令（4 条）

1、位逻辑“与”指令

```
ANL    C, bit          ; (C) ∧ (bit) → (C)
ANL    C, /bit         ; (C) ∧ (/bit) → (C)
如: ANL    C, P1.0
      ANL    C, /P1.2
```

2、位逻辑“或”指令

```
ORL    C, bit          ; (C) ∨ (bit) → (C)
ORL    C, /bit         ; (C) ∨ (/bit) → (C)
如: ORL    C, P1.0
      ORL    C, /P1.2
```

<四> 位条件转移类指令（5 条）

1、判断 C 值转移指令

```
JC      rel           ; (C) =1, 则 (PC) +2+rel→ ( PC ) SJMP rel
                        ; (C) =0, 则 (PC) +2→ ( PC ) 顺序向下执行
```

```
JNC     rel           ; (C) =0, 则 (PC) +2+rel→ ( PC ) SJMP rel
                        ; (C) =1, 则 (PC) +2→ ( PC ) 顺序向下执行
```

常与 CJNE 指令一起用

如: JC NEXT1
 JNC FIRST

CJNE A, direct, rel

 ; 若 $X1 > X2$, 则 $(PC)+3+rel \rightarrow (PC)$, 且 $0 \rightarrow CY$;
 ; 若 $X1 < X2$, 则 $(PC)+3+rel \rightarrow (PC)$, 且 $1 \rightarrow CY$;
 ; 若 $X1 = X2$, 则顺序执行 $(PC)+3 \rightarrow (PC)$, 且 $0 \rightarrow CY$ 。

2、判断位值转移指令

JB bit, rel ; 若 (bit) =1, 则 $(PC)+3+rel \rightarrow PC$ SJMP rel
 ; 若 (bit) =0, 则 $(PC)+3 \rightarrow (PC)$ 顺序向下执行
JNB bit, rel ; 若 (bit) =0, 则 $(PC)+3+rel \rightarrow PC$ SJMP rel
 ; 若 (bit) =1, 则 $(PC)+3 \rightarrow (PC)$ 顺序向下执行

如: JB 20H.3, NEXT1
 JNB TI, \$

3、判断位值并清 0 转移指令

JBC bit, rel ; 若 (bit)=1, 则 $(PC)+3+rel \rightarrow PC$, $0 \rightarrow bit$ SJMP rel
 ; 若 (bit) =0, 则 $(PC)+3 \rightarrow (PC)$ 则顺序向下执行

如: JBC F0, NEXT1

附录 B：修订记录：

- 2009-05-28： 1、重新整理，增加第三、第四部分；
2、补全了背景知识；
3、增加了附录 A：51 系列单片机指令系统。
- 2009-6-1： 1、修改实验 2-9 数码管显示实验；
2、增加实验 2-10 键盘扫描。
- 2010-3-1： 部分文字修改。