

# 单片机实验指导书



西安唐都科教仪器公司

Copyright Reserved 2006

## 版权声明

本书的版权归西安唐都科教仪器开发有限责任公司所有，保留一切权利。未经本公司书面许可，任何单位和个人不得擅自摘抄、复制本书的部分或全部内容，以任何形式传播。

西安唐都科教仪器开发有限责任公司，2006(C)，All right reserved.

单片机实验指导书

©版权所有 未经许可 严禁复制

技术支持邮箱: [service@tangdu.com](mailto:service@tangdu.com)

唐都公司网址: <http://www.tangdu.com>

# 目 录

<b>第 1 章 概述</b>	<b>1</b>
1.1 SST89E554RC 简介	1
1.2 实验项目	3
1.3 Keil C51 的安装	4
1.3.1 系统要求	4
1.3.2 软件安装	4
1.4 $\mu$ Vision2 集成开发环境	7
1.5 仿真调试与脱机运行间的切换方法	10
1.5.1 脱机运行	10
1.5.2 与 Keil C51 开发环境联机调试的方法	13
1.5.3 从 SoftICE 返回 IAP 引导程序的方法	14
<b>第 2 章 单片机原理实验</b>	<b>15</b>
2.1 系统认识实验	15
2.2 数码转换实验	22
2.3 运算程序设计实验	24
2.4 查表程序设计实验	28
2.5 数据排序实验	30
2.6 位操作实验	32
<b>第 3 章 单片机集成功能模块实验</b>	<b>33</b>
3.1 数字量输入输出实验	33
3.2 中断系统实验	35
3.3 定时/计数器实验	38
3.4 看门狗实验	42
3.5 低功耗实验	45
3.6 PCA 实验	48
3.7 串口通讯实验	53
3.8 SPI 总线实验	56
<b>第 4 章 单片机系统扩展实验</b>	<b>62</b>
4.1 静态存储器扩展实验	62
4.2 FLASH 存储器扩展实验	64
4.3 A/D 转换实验	68

4.4	D/A 转换实验 .....	71
4.5	键盘扫描及显示设计实验.....	73
4.6	电子发声设计实验.....	78
4.7	点阵 LED 显示设计实验.....	81
4.8	接触式 IC 卡读写实验.....	84
4.9	单总线数字温度传感器实验.....	89
4.10	字符型 LCD 显示设计实验.....	95
4.11	图形 LCD 显示设计实验 (选配) .....	103

## **第 5 章 单片机控制应用实验.....110**

5.1	步进电机实验 .....	110
5.2	直流电机 PWM 调速实验.....	112
5.3	温度闭环控制实验.....	114

## 第 1 章 概述

单片机进入我国已 20 多年了,随着科学技术的发展,计算机技术的普及,单片机已经成为工科院校的一门技术基础课。西安唐都科教仪器公司自成立以来一直致力于单片机教学实验平台的开发,并积累了大量的经验。从 2002 年开始,我公司又陆续推出了以 SST89E554RC 单片机为核心的一系列单片机教学实验平台,以满足不同用户的不同需要。

### 1.1 SST89E554RC 简介

SST89E554RC 具有在系统可编程 (ISP) 和在应用可编程 (IAP) 技术,该器件是 SST 公司推出的 8 位微控制器 FlashFlex51 家族中的一员,内置仿真程序,完全取代传统的硬件仿真器和编程器。这种先进的单片机将仿真系统 and 应用系统合二为一,大大降低了应用开发成本,极大地提高了研发效率。把单片机的仿真开发和应用设计提高到一个崭新的技术领域。SST89E554RC 具有如下特征:

- 与 8051 兼容,嵌入 SuperFlash 存储器
  - 软件完全兼容
  - 开发工具兼容
  - 引脚全兼容
- 工作电压 5V,工作时钟 0~40MHz
- 1Kbyte 内部 RAM
- 两块 SuperFlash EEPROM,主块 32Kbyte,从块 8Kbyte,扇区为 128Byte
- 有三个高电流驱动端口 (每个 16mA)
- 三个 16 位的定时器/计数器
- 全双工、增强型 UART
  - 帧错误检测
  - 自动地址识别
- 八个中断源,四级优先级
- 可编程看门狗定时器 (WDT)
- 可编程计数阵列 (PCA)
- 双 DPTR 寄存器
- 低 EMI 模式 (可禁止 ALE)
- SPI 串行接口
- 标准每周期 12 个时钟,器件提供选项可使速度倍增,达到每周期 6 个时钟
- 低功耗模式
  - 掉电模式,可由外部中断唤醒
  - 空闲模式

SST89E554RC 的功能框图如图 1-1-1 所示，外部引脚如图 1-1-2 所示。

SST89E554RC 的特殊功能寄存器如表 1-1-1 所列。

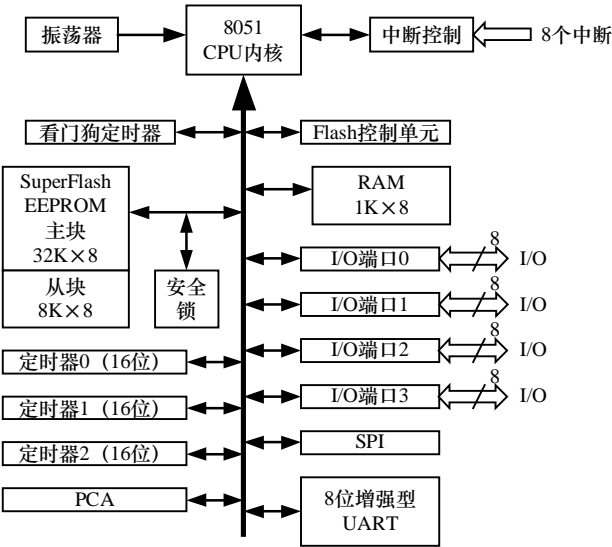


图 1-1-1 SST89E554RC 功能框图

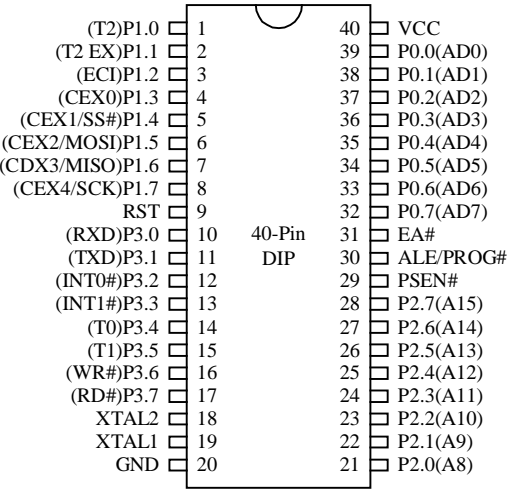


图 1-1-2 SST89E554RC 外部引脚图

表 1-1-1 SST89E554RC 特殊功能寄存器存储器映象

8 字节								
F8H	IPA <sup>1</sup>	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	CCAP4H	FFH
F0H	B <sup>1</sup>							IPAH
E8H	IEA <sup>1</sup>	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	CCAP4L	EFH
E0H	ACC <sup>1</sup>							E7H
D8H	CCON <sup>1</sup>	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	CCAPM4	DFH
D0H	PSW <sup>1</sup>							D7H
C8H	T2CON <sup>1</sup>	T2MOD	RCAP2L	RCAP2H	TL2	TH2		CFH
C0H	WDTC <sup>1</sup>							C7H
B8H	IP <sup>1</sup>	SADEN						BFH
B0H	P3 <sup>1</sup>	SFCF	SFCM	SFAL	SFAH	SFDT	SFST	IPH
A8H	IE <sup>1</sup>	SADDR	SPSR					AFH
A0H	P2 <sup>1</sup>		AUXR1					A7H
98H	SCON <sup>1</sup>	SBUF						9FH
90H	P1 <sup>1</sup>							97H
88H	TCON <sup>1</sup>	TMOD	TL0	TL1	TH0	TH1	AUXR	8FH
80H	P0 <sup>1</sup>	SP	DPL	DPH		WDTD	SPDR	PCON
								87H

注：1 表示该特殊功能寄存器可位寻址。

关于此单片机特有功能模块及寄存器可参看芯片数据手册或相应实验章节。

## 1.2 实验项目

本实验指导书包含以下单片机实验，某些实验仅适用于相应实验平台，具体可见实验中的实验设备要求，如果没有具体指明则表示该实验适用于所有设备。

### 1. 单片机原理实验

- (1) 系统认识实验
- (2) 数码转换实验
- (3) 运算程序设计实验
- (4) 查表程序设计实验
- (5) 数据排序实验
- (6) 位操作实验

### 2. 单片机集成功能模块实验

- (1) 数字量输入/输出实验
- (2) 中断系统实验
- (3) 定时器/计数器实验
- (4) 看门狗实验
- (5) 低功耗实验
- (6) PCA 实验
- (7) 串口通讯实验
- (8) SPI 总线实验 (TD-NMC+)

### 3. 单片机系统扩展实验

- (1) 静态存储器扩展实验
- (2) FLASH 扩展实验
- (3) A/D 转换实验
- (4) D/A 转换实验
- (5) 8255 键盘及数码显示实验
- (6) 电子发声实验
- (7) 点阵 LED 实验
- (8) 接触式 IC 卡读写实验 (TD-NMC+)
- (9) 单总线数字温度传感器实验 (TD-NMC+)
- (10) 字符型液晶实验 (TD-NMC+)
- (11) LCD 图形液晶实验 (选配)

### 4. 单片机控制应用实验

- (1) 步进电机实验 (选配)
- (2) 直流电机实验
- (3) 温度闭环控制实验

## 1.3 Keil C51 的安装

Keil C51  $\mu$ Vision2 集成开发环境是 Keil 公司开发的基于 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编译、链接、目标代码生成、软件仿真、硬件仿真等完整的开发流程。

### 1.3.1 系统要求

安装 Keil C51 集成开发软件，必须满足最小的软、硬件要求，以确保程序功能的正常。

- (1) Pentium、Pentium-II 或兼容处理器的 PC；
- (2) Windows98、Windows2000 或 Windows XP 操作系统；
- (3) 至少 16MB RAM；
- (4) 至少 20MB 硬盘空间。

### 1.3.2 软件安装

下面介绍如何安装 Keil  $\mu$ Vision2 集成开发环境。

(1) 进入 Keil C51 软件的 Setup 目录下，双击 SETUP.EXE 开始安装，这时会出现如图 1-3-1 所示的安装初始化界面。

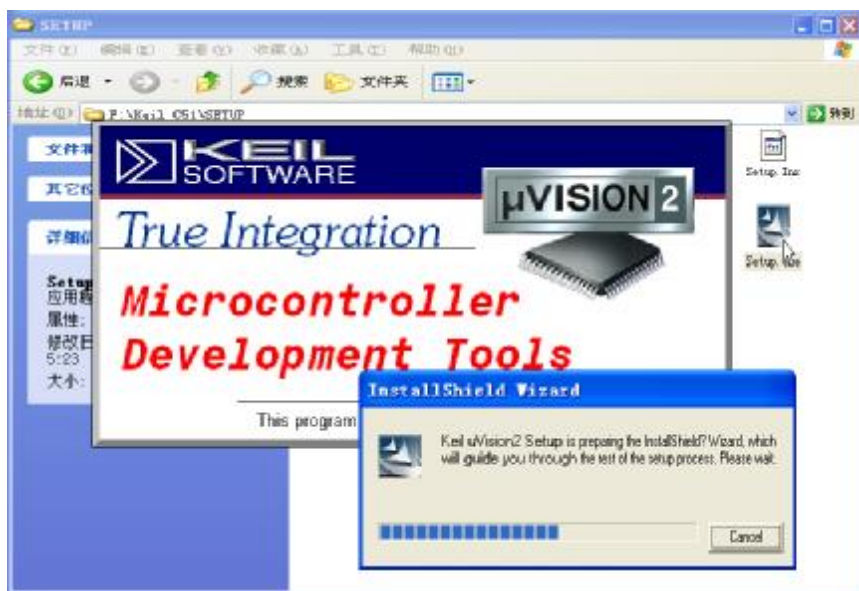


图 1-3-1 安装初始化

(2) 接下来会弹出安装向导对话框，如图 1-3-2 所示，询问此时是需要安装、修复更新或是卸载 Keil C51 软件。若是第一次安装该软件，请选择第一项 Install ... 安装软件。



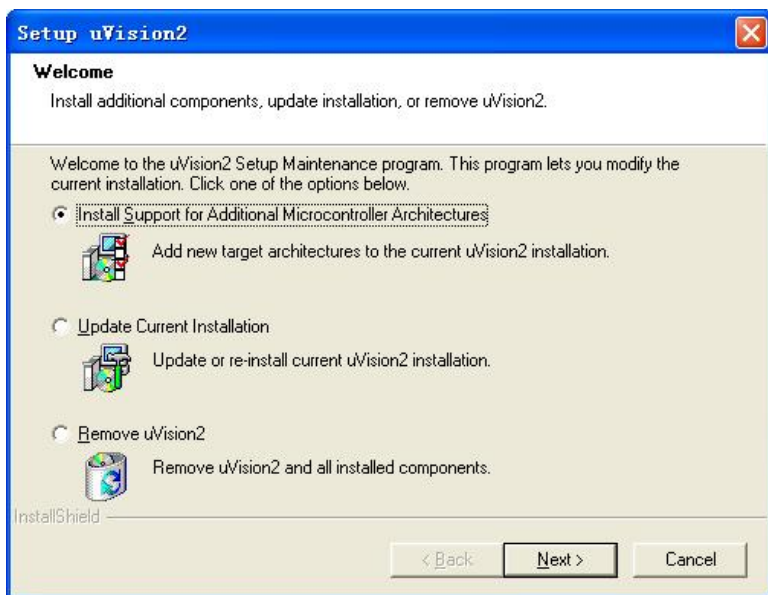


图 1-3-2 安装向导界面

(3) 单击 Next 按钮，此时会出现图 1-3-3 所示的安装询问对话框，提示用户是安装完全版还是评估版。如果购买的是正版 Keil C51 软件则选择 Full Version，否则选择 Eval Version 选项。



图 1-3-3 安装询问对话框

(4) 选择完毕后，紧接着会弹出几个确认对话框，点击 Next 按钮，这时会出现如图 1-3-4 所示的安装路径设置对话框，默认路径是 C:\KEIL，可以点击 Browse 按钮选择合适自己安装的目录。

(5) 点击 Next 按钮，如果安装的为评估版的软件，会出现如图 1-3-5 所示的安装进度指示界面，若安装的是完全版的软件，则下面会弹出用户信息对话框，要求用户输入软件序列号、姓名、公司及 E-mail 等信息，信息输入完后点击 Next 按钮，在接下来的几个确认对话框中点击 Next 确认按钮，即可出现图 1-3-5 所示的安装进度指示界面。

(6) 安装完毕点击 Finish 按钮，此时就可以在桌面上看到 Keil  $\mu$ Vision2 软件的快捷图标，如图 1-3-6 所示，双击此图标可以进入 Keil C51 集成开发环境。

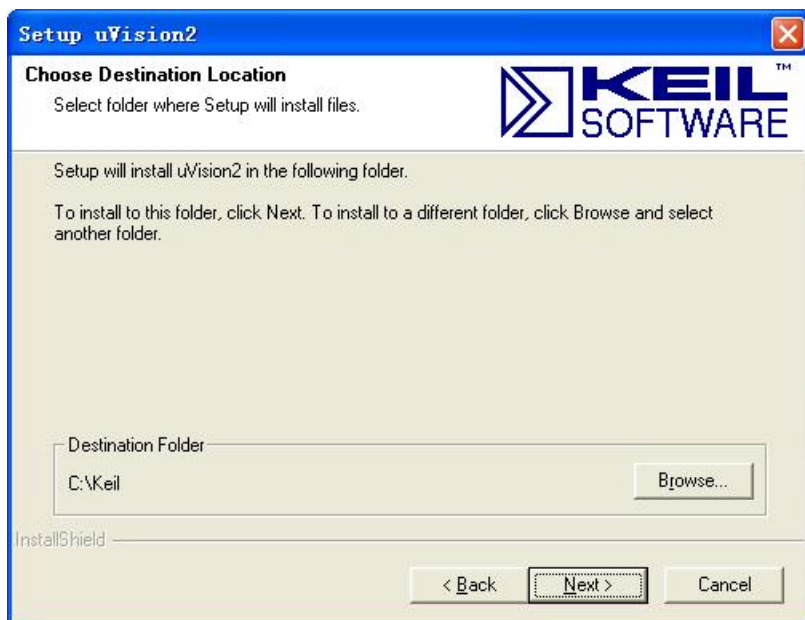


图 1-3-4 安装路径设置对话框

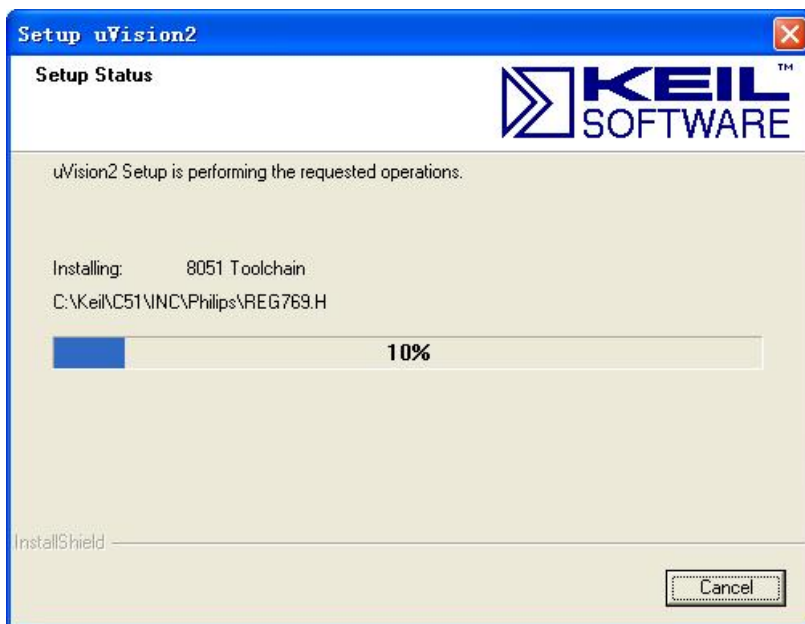


图 1-3-5 安装进度指示界面



图 1-3-6 快捷图标

## 1.4 $\mu$ Vision2 集成开发环境

$\mu$ Vision2 支持所有的 Keil 80C51 的工具软件, 包括 C51 编译器、宏汇编器、链接器/定位器、软硬件调试器和目标文件到 HEX 格式文件转换器等,  $\mu$ Vision2 可以自动完成编译、汇编、链接程序等操作。

$\mu$ Vision2 具有强大的软件环境、友好的操作界面和简单快捷的操作方法。

双击桌面上的 Keil  $\mu$ Vision2 快捷图标, 可以进入如图 1-4-1 所示的集成开发调试环境, 各种调试工具、命令菜单都集成在此开发环境中。菜单栏提供了各种操作菜单, 如编辑器操作、工程维护、程序调试、窗体选择以及操作帮助等。工具栏按钮和快捷键可以快速执行  $\mu$ Vision2 命令。常用的菜单栏及相对应的工具栏按钮与快捷键介绍如表 1-4-1~表 1-4-6 所列。

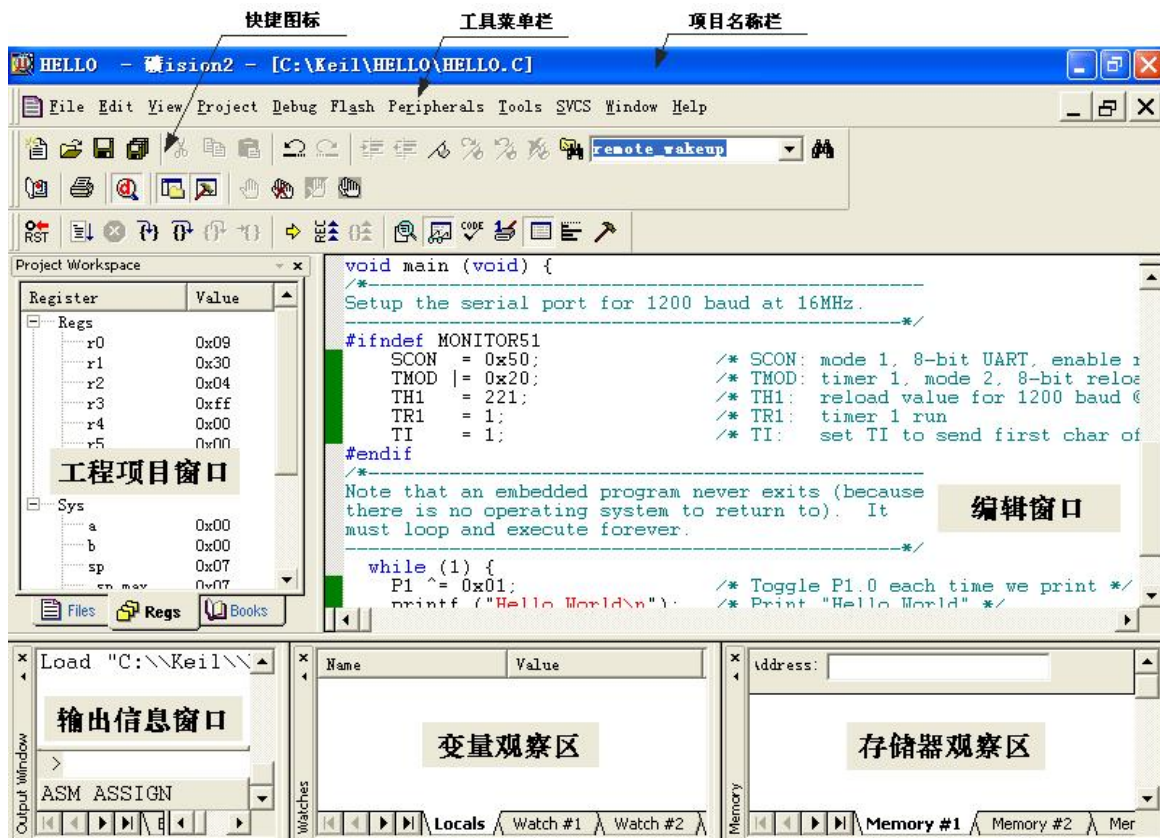



图 1-4-1  $\mu$ Vision2 集成环境界面

表 1-4-1 文件菜单和文件命令 (File)

File 菜单	工具栏	快捷键	描述
New		Ctrl+N	创建一个新的源文件或文本文件
Open		Ctrl+O	打开已有的文件
Close			关闭当前的文件
Save		Ctrl+S	保存当前文件



Save All			保存所有打开的源文件和文本文件
Device Database			维护μVision2 器件数据库
Print Setup...			设置打印机
Print		Ctrl + P	打印当前文档
Print Preview			打印预览
1~9			打开最近使用的源文件或文本文件
Exit			退出μVision2 并提示保存

表 1-4-2 编辑菜单和编辑器命令 (Edit)

Edit 菜单	工具栏	快捷键	描 述
Undo		Ctrl + Z	撤消上一次的操作
Redo		Ctrl + Shift + Z	恢复上一次撤消的命令
Cut		Ctrl + X	将选中的文字剪切到剪贴板
Copy		Ctrl + C	将选中的文字复制到剪贴板
Paste		Ctrl + V	粘贴剪贴板的文字
Indent Selected Text			将选中的文字向右缩进一个制表符位
Unindent Selected Text			将选中的文字向左缩进一个制表符位
Toggle Bookmark		Ctrl + F2	在当前行放置书签
Goto Next Bookmark		F2	将光标移到下一个书签
Goto Previous Bookmark		Shift + F2	将光标移到上一个书签
Clear All Bookmarks			清除当前文件中的所有书签
Find		Ctrl + F F3 Shift + F3	在当前文件中查找文字 继续向前查找文字 继续向后查找文字
Find in Files...			在几个文件中查找文字
Goto Matching Brace		Ctrl + ]	查找匹配的花括号、圆括号、方括号
Replace		Ctrl + H	替换特定的文字

表 1-4-3 视图菜单 (View)

View 菜单	工具栏	快捷键	描 述
Status Bar			显示或隐藏状态栏
File Toolbar			显示或隐藏文件工具栏
Build Toolbar			显示或隐藏编译工具栏
Debug Toolbar			显示或隐藏调试工具栏
Project Window			显示或隐藏工程窗口
Output Window			显示或隐藏输出窗口
Source Browser			打开源 (文件) 浏览器窗口
Disassembly Window			显示或隐藏反汇编窗口
Watch&Call Stack Window			显示或隐藏观察和堆栈窗口

Memory Window			显示或隐藏存储器窗口
Code Coverage Window			显示或隐藏代码覆盖窗口
Performance Analyzer Window			显示或隐藏性能分析窗口
Symbol Window			显示或隐藏符号变量窗口
Serial Window # 1			显示或隐藏串行窗口 1
Serial Window # 2			显示或隐藏串行窗口 2
Toolbox			显示或隐藏工具箱
Periodic Window Update			在运行程序时，周期刷新调试窗口
Workbook Mode			显示或隐藏工作簿窗口的标签
Options...			设置颜色、字体、快捷键和编辑器选项

表 1-4-4 工程菜单和工程命令 (Project)

Project 菜单	工具栏	快捷键	描 述
New Project...			创建一个新的工程
Import μVision1 Project...			输入一个μVision1 工程文件
Open Project...			打开一个已有的工程
Close Project...			关闭当前的工程
Target Environment			定义工具系列、包含文件、库文件的路径
Targets, Groups, Files			维护工程的对象、文件组 and 文件
Select Device for Target			从器件数据库选择一个 CPU
Remove Item			从工程中删除一个组或文件
Options for Target...		Alt + F7	设置对象、组或文件的工具选项
Build Target		F7	编译当前的文件
Rebuild all Target files			重新编译所有的文件
Translate...		Ctrl + F7	转换当前的文件
Stop Build			停止当前的编译进程

表 1-4-5 调试菜单和调试命令 (Debug)

Debug 菜单	工具栏	快捷键	描 述
Start/Stop Debugging		Ctrl + F5	启动或停止μVision2 调试模式
Go		F5	运行程序，直到遇到下一个有效的断点
Step		F11	跟踪运行程序
Step Over		F10	单步运行程序
Step out of current function		Ctrl + F11	单步出当前函数
Run to cursor line		Ctrl + F10	执行程序到光标所在行
Stop Running		ESC	停止程序运行
Breakpoints...			打开断点对话框
Insert/Remove Breakpoint			在当前行设置/清除断点
Enable/Disable Breakpoint			使能/禁止当前行的断点

Disable All Breakpoints			禁止程序中的所有断点
Kill All Breakpoints			清除程序中的所有断点
Show Next Statement			显示下一条执行的语句/指令
Enable/Disable Trace Recording			使能跟踪记录，可以显示程序运行轨迹
View Trace Records			显示以前执行的指令
Memory Map...			打开存储器空间配置对话框
Performance Analyzer...			打开性能分析器的设置对话框
Inline Assembly...			对某一行重新汇编，可以修改汇编代码
Function Editor...			编辑调试函数和调试配置文件

表 1-4-6 外围器件菜单 (Peripherals)

Peripheral 菜单	工具栏	快捷键	描 述
Reset CPU			复位 CPU
Interrupt, I/O-Ports, Serial, Timer, SPI			打开片上外围器件的对话框。对话框的列表和内容由在器件数据库中选择的 CPU 决定。

1.5 仿真调试与脱机运行间的切换方法

SST 公司独创的 IAP 技术将单片机内部的程序存储器进行分块，巧妙的将系统程序与用户应用程序分别放置在不同的存储块中，以实现单片机的仿真调试或脱机运行。如果单片机内部的系统程序为 SoftICE，那么可以与 Keil C51 软件联机进行仿真调试；如果系统程序为启动加载程序，可以代替编程器，下载用户目标代码实现脱机运行。改变系统程序便可以进行仿真调试与脱机运行间的切换。随机光盘提供有 SSTEasyIAP11F.exe 软件，SoftIce554.hex 文件和 Convert\_to\_BSLx554.txt 文件以实现切换。

1.5.1 脱机运行

SST 公司提供的 SSTEasyIAP11F 软件，为 SST 单片机的用户提供了通过 IAP 技术把用户应用程序下载到单片机的程序存储器或者从单片机的程序存储器读出用户应用程序的方法。当单片机内部的系统程序为启动加载程序时，用户可以通过 SSTEasyIAP11F 软件，将得到的目标代码 (\*.HEX) 下载到单片机内部的 FLASH 中，系统复位后，单片机便会全速执行用户程序。目标代码下载的具体步骤如下：

- (1) 运行软件 SSTEasyIAP11F，出现如图 1-5-1 所示操作界面。



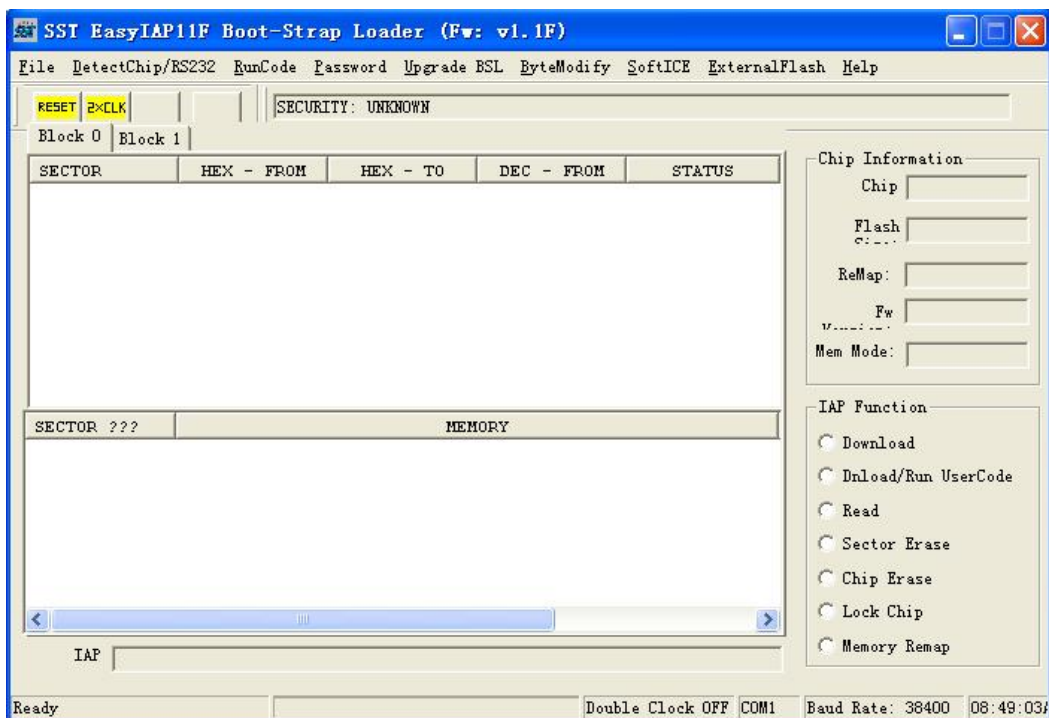


图 1-5-1 SSTEasyIAP11F 软件操作界面

- (2) 点击“Detect Chip/RS232”菜单，出现如图 1-5-2 所示下拉菜单。



图 1-5-2 Detect Chip/RS232 下拉菜单

- (3) 点击“Detect Target MCU for Firmware1.1F and RS232 Config.”选项出现如图 1-5-3 所示的芯片选择和存储器模式窗口。芯片类型选择“SST89E554RC”，存储器模式选择“Internal Memory”，选择完后点击“OK”按钮。然后可以看到如图 1-5-4 所示的 RS232 配置与目标检测窗口。
- (4) 直接点击“Detect MCU”按钮，便会弹出如图 1-5-5 所示的提示信息，告诉用户在点击“确定”按钮后按系统的复位键来复位 MCU 以检测波特率和芯片。



图 1-5-3 芯片类型与存储器模式选择窗口



图 1-5-4 RS232 配置与目标检测窗口



图 1-5-5 提示信息窗

- (5) 检测成功后，可以看到检测后的信息，如图 1-5-6 所示。在 IAP Function 功能框中选择 Download，以下载目标代码。随后会弹出密码校验对话框，直接点击“OK”即可。

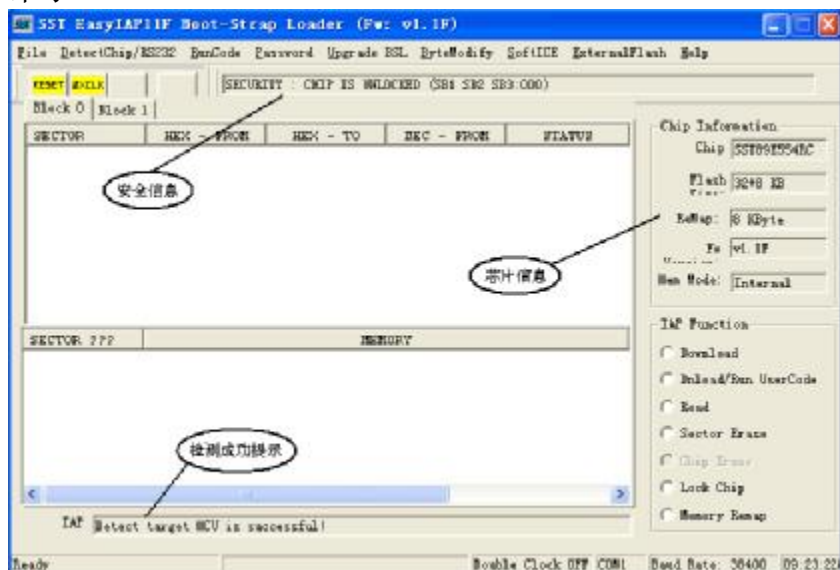


图 1-5-6 检测后显示信息



- (6) 如图 1-5-7 所示, 将弹出下载对话框, 点击按钮 “...” 来选择要下载的文件, 然后点击 “OK” 会弹出如图 1-5-8 所示擦除提示信息窗, 在写入新数据前会擦除 Flash 的原有内容,。点击 “是” 来完成下载。下载完成, 就可以脱机运行程序了。

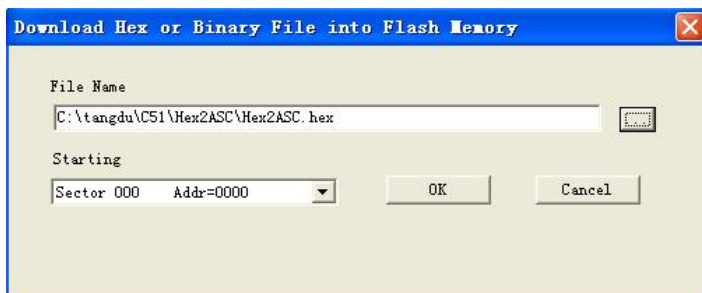


图 1-5-7 下载对话框



图 1-5-8 下载提示窗

## 1.5.2 与 Keil C51 开发环境联机调试的方法

在 SST 单片机内部固化了 SoftICE 后, 便可以实现单片机与 Keil C51 集成开发环境的联机调试。要求 SoftICE554.hex 文件与 SSTEasyIAP11F 软件在同一目录下。具体步骤如下:

- (1) 同下载目标代码到单片机的步骤 1~5, 使用 SSTEasyIAP11F 软件必须先检测 MCU。
- (2) 点击菜单栏的 “SoftICE”, 弹出下拉菜单 “Download SoftICE”, 如图 1-5-9 所示, 然后点击 Download SoftICE 选项以下载 SoftICE。

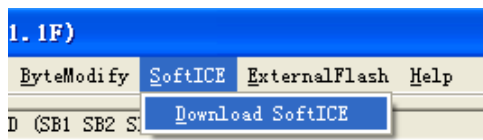


图 1-5-9 下载 SoftICE 菜单选项

- (3) 同样会弹出密码校验对话框, 直接按 “OK”, 会弹出如图 1-5-10 所示的提示信息框, 信息提示这将会删除 IAP 引导程序。



图 1-5-10 下载提示信息窗

- (4) 选择“是”，开始下载 SoftICE，下载完成会出现如图 1-5-11 所示的完成信息提示。

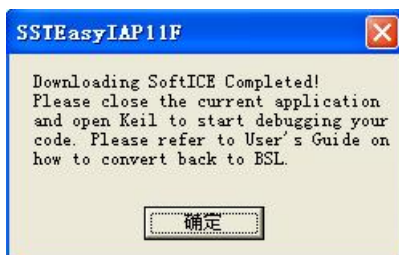


图 1-5-11 下载完成提示

- (5) 完成 SoftICE 下载，便可以开始与 Keil C51 联机调试了。

### 1.5.3 从 SoftICE 返回 IAP 引导程序的方法

当用户需要将目标代码\*.Hex 文件下载到单片机脱机运行而系统程序还是 SoftICE 时，就需要通过 Convert\_to\_BSLx554.txt 文件将系统程序从 SoftICE 切换回 IAP 引导加载程序。

具体操作步骤如下：

- (1) 启动 Keil C51，进入联机调试状态。
- (2) 得到 Convert\_to\_BSLx554.txt 文件的路径，在输出窗口的 Command 页的命令行内输入 “Include C:\Keil\Convert\_to\_BSLx554.txt” 命令后回车。如图 1-5-12 所示。

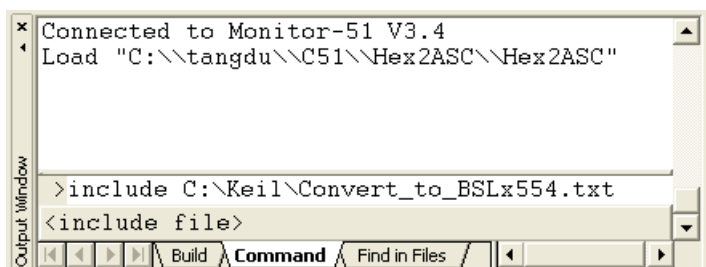


图 1-5-12 从 SoftICE 返回 IAP 引导程序命令输入窗

- (3) 耐心等待（这需要较长的时间），当出现如图 1-5-13 所示信息时表示已成功地从 SoftICE 返回 IAP 引导程序了。

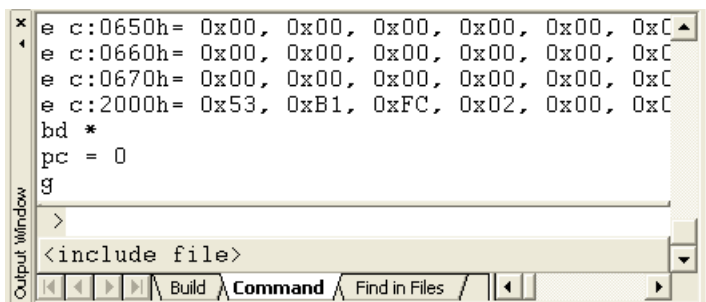


图 1-5-13 从 SoftICE 成功返回 IAP 引导程序

## 第2章 单片机原理实验

通过本章的实验，旨在使学生掌握 Keil C51 的操作方法，学习 80C51 的指令系统及汇编语言的程序设计方法。

### 2.1 系统认识实验

#### 2.1.1 实验目的

1. 学习 Keil C51 集成开发环境的操作；
2. 熟悉 TD-51 系统板的结构及使用。

#### 2.1.2 实验内容

编写实验程序，将 00H~0FH 共 16 个数写入单片机内部 RAM 的 30H~3FH 空间。  
通过本实验，学生需要掌握 Keil C51 软件的基本操作，便于后面的学习。

#### 2.1.3 实验步骤

##### 1. 创建 Keil C51 应用程序

在 Keil C51 集成开发环境下使用工程的方法来管理文件，所有的源文件、头文件甚至说明性文档都可以放在工程项目文件里统一管理。

下面创建一个新的工程文件 Asm1.Uv2，以此详细介绍如何创建一个 Keil C51 应用程序。

(1) 运行 Keil C51 软件，进入 Keil C51 集成开发环境。

(2) 选择工具栏的 Project 选项，如图 2-1-1 所示，弹出下拉菜单，选择 NewProject 命令，建立一个新的  $\mu$ Vision2 工程。这时会弹出如图 2-1-2 所示的工程文件保存对话框，选择工程目录并输入文件名 Asm1 后，单击保存。

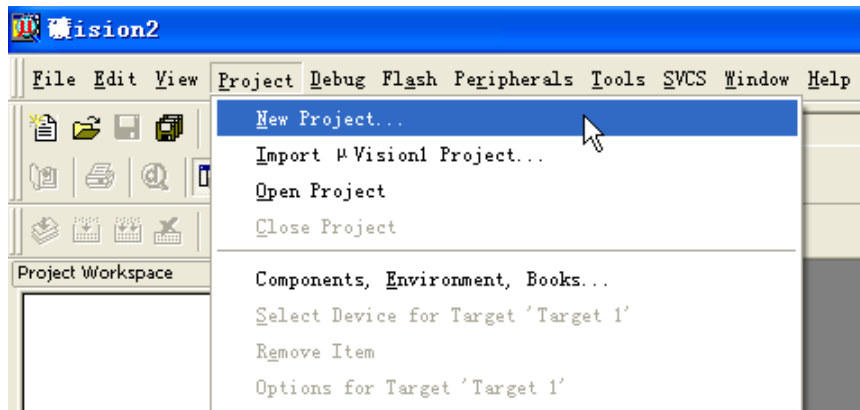


图 2-1-1 工程下拉菜单

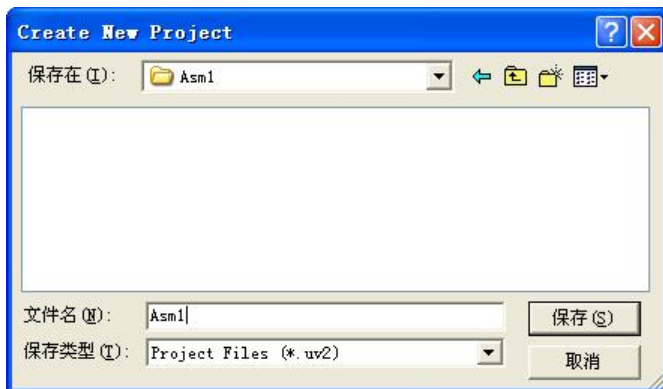


图 2-1-2 工程保存对话框

(3) 工程建立完毕后， $\mu$ Vision2 会马上弹出如图 2-1-3 所示的器件选择窗口。器件选择的目的是告诉  $\mu$ Vision2 使用的 80C51 芯片的型号是哪一个公司的哪一个型号，不同型号的 51 芯片内部资源是不同的。此时选择 SST 公司的 SST89E554RC。另外，可以选择 Project 下拉菜单中的“Select Device for Target ‘Target 1’”命令来弹出图 2-1-3 所示的对话框。

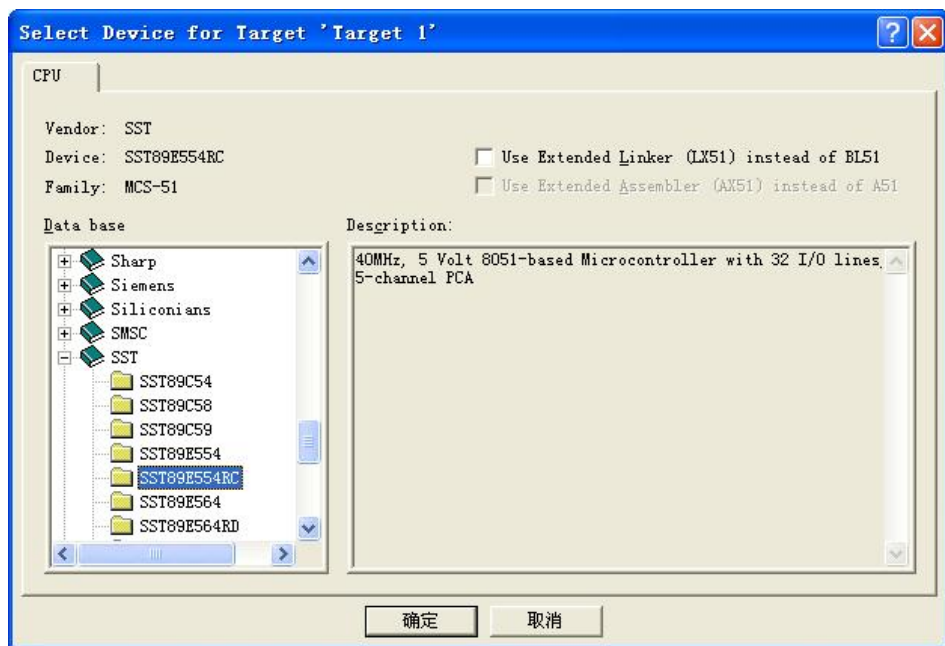



图 2-1-3 器件选择对话框

(4) 到此建立好一个空白工程，现在需要人工为工程添加程序文件，如果还没有程序文件则必须建立它。选择工具栏的 File 选项，在弹出的下拉菜单中选择 New 目录，如图 2-1-4 所示，或点击 。此时会在文件窗口出现如图 2-1-5 所示的新文件窗口 Text1，若多次执行 New 命令，则会出现 Text2、Text3 等多个新文件窗口。

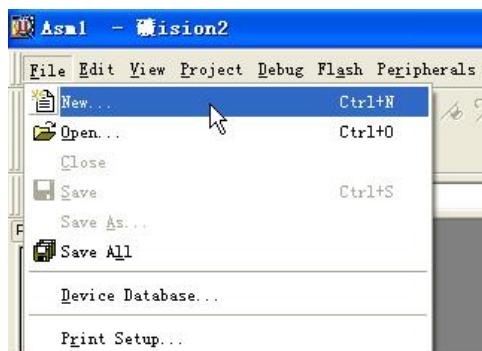


图 2-1-4 新建源文件下拉菜单

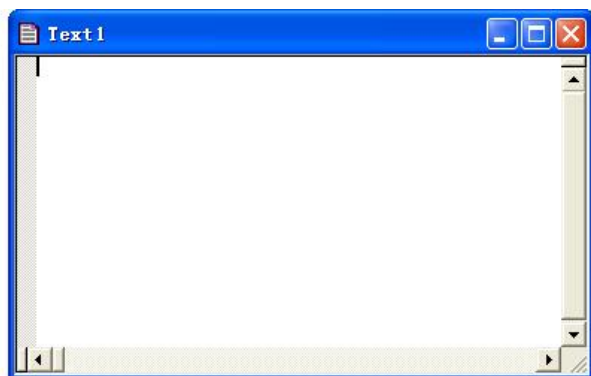


图 2-1-5 源程序编辑窗口

(5) 输入程序，完毕后点击“保存”命令保存源程序，如图 2-1-6 所示，将 Text1 保存成 Asm1.asm。Keil C51 支持汇编和 C 语言， $\mu$ Vision2 会根据文件后缀判断文件的类型，进行自动处理，因此保存时需要输入文件名及扩展名.ASM 或.C。保存后，文件中字体的颜色会发生一定变化，关键字会变为蓝色。

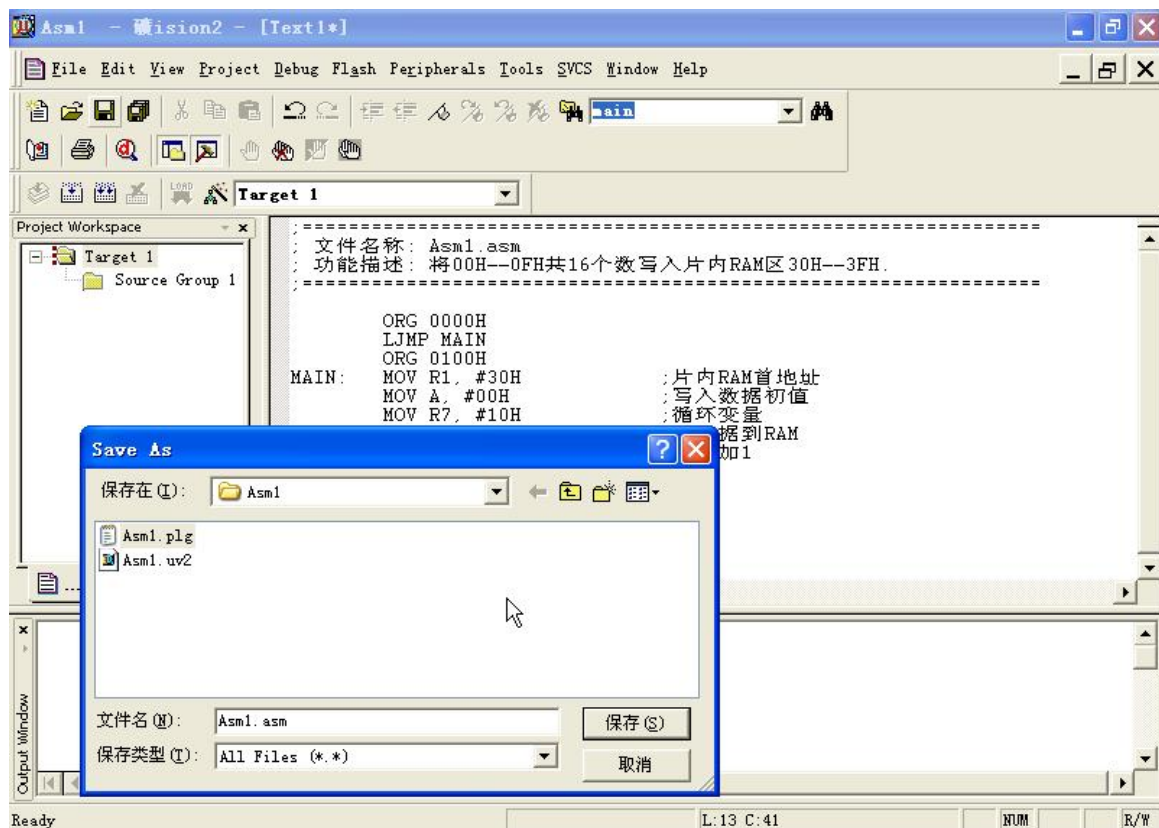


图 2-1-6 源文件保存对话框

(6) 程序文件建立后，并没有与 Asm1.Uv2 工程建立任何关系。此时，需要将 Asm1.asm 源程序添加到 Asm1.Uv2 工程中，构成一个完整的工程项目。在 Project Window 窗口内，选中

Source Group1 点击鼠标右键,会弹出如图 2-1-7 所示的快捷菜单,选择 Add Files to Group‘Source Group1’ 命令,此时弹出如图 2-1-8 所示的添加源程序文件对话框,选择文件 Asm1.asm, 点击 Add 命令按钮即可将源程序文件添加到工程中。

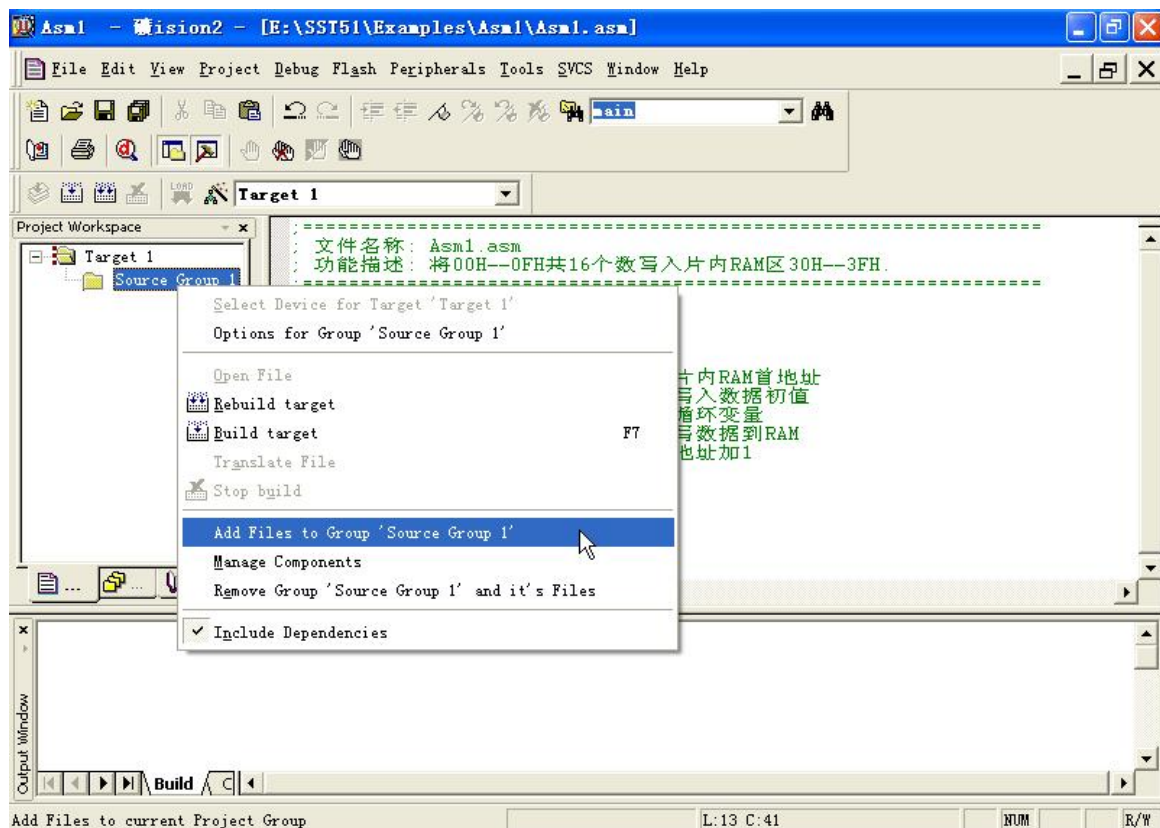


图 2-1-7 添加源程序文件快捷菜单

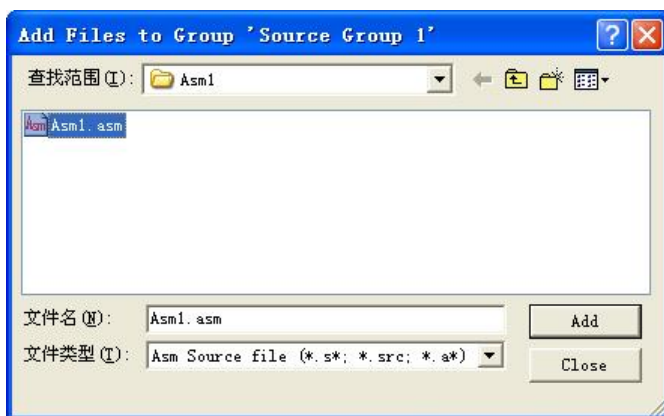



图 2-1-8 添加源程序文件对话框

## 2. 编译、链接程序文件

(1) 设置编译、链接环境,点击  命令,会出现如图 2-1-9 所示的调试环境设置窗口,在



这里可以设置目标系统的时钟。单击 Output 标签, 在打开的选项卡中选中 Create Hex File 选项, 在编译时系统将自动生成目标代码\*.Hex。点击 Debug 标签会出现如图 2-1-10 所示的调试模式选择窗口。



从图 2-1-10 可以看出,  $\mu$ Vision2 有两种调试模式: Use Simulator (软件仿真) 和 Use (硬件仿真)。这里选择硬件仿真, 点击 Settings 可以设置串口。



图 2-1-9 Keil C51 调试环境设置窗口



图 2-1-10 调试设置窗口

(2) 点击  或  命令编译、链接程序，此时会在 Output Window 信息输出窗口输出相关信息，如图 2-1-11 所示。

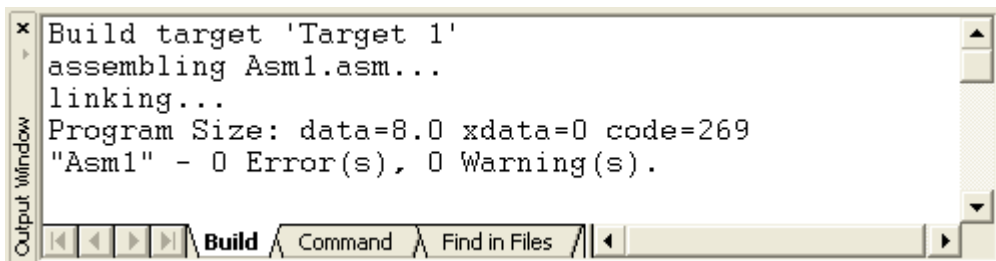


图 2-1-11 编译、链接输出窗口

### 3. 调试仿真程序

(1) 打开系统板的电源，给系统复位后点击  调试命令（注：每次进入调试状态前确保系统复位正常），将程序下载到单片机的 FLASH 中，此时出现如图 2-1-12 所示调试界面。

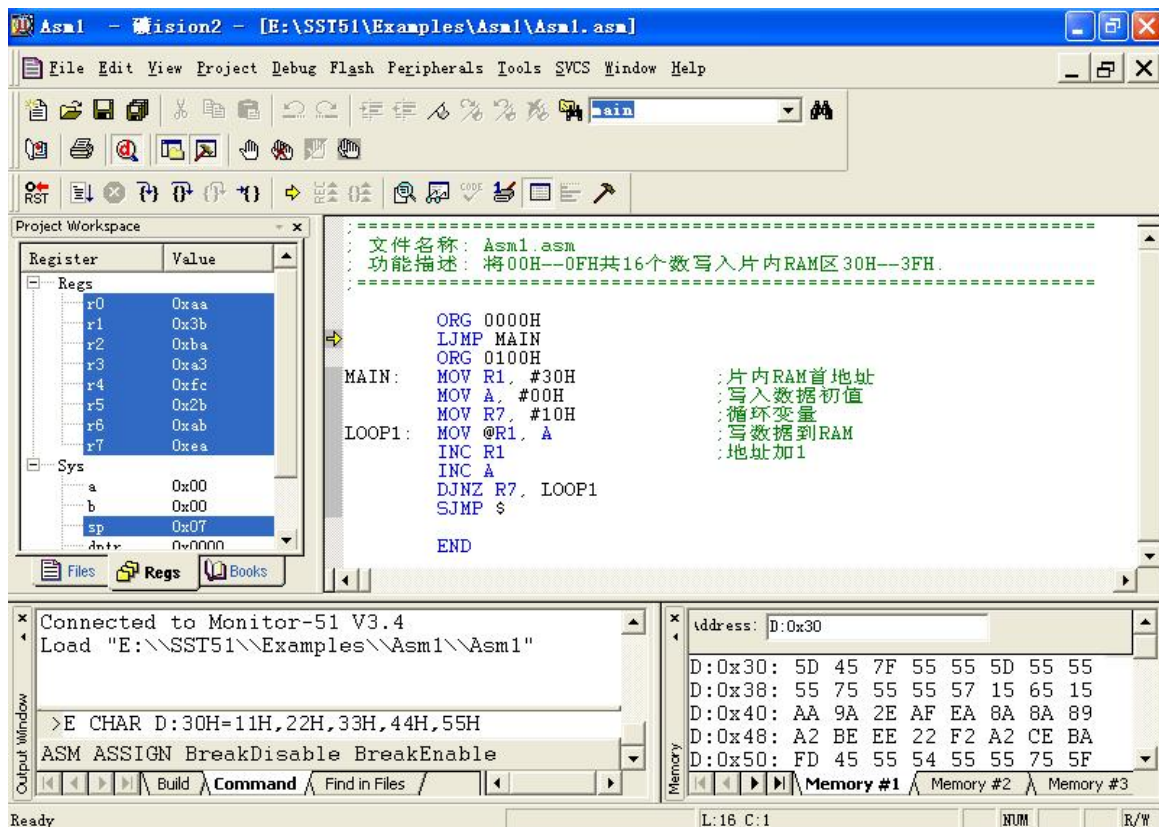





图 2-1-12 调试界面

(2) 点击  命令，可以打开存储器观察窗口，在存储器观察窗口的‘Address:’栏中输入 D:30H（或 0x30）则显示片内 RAM30H 后的内容，如图 2-1-12 所示。如果输入‘C:’表示显示代码存储器的内容，‘I:’表示显示内部间接寻址 RAM 的内容，‘X:’表示显示外部数据存储器的



中的内容。

(3) 将光标移到 SJMP \$ 语句行, 点击  命令, 在此行设置断点。

(4) 接下来点击  命令, 运行实验程序, 当程序遇到断点后, 程序停止运行, 观察存储器中的内容, 如图 2-1-13 所示, 验证程序功能。

(5) 如图 2-1-12 所示, 在命令行中输入 ‘E CHAR D:30H=11H,22H,33H,44H,55H’ 后回车, 便可以改变存储器中多个单元的内容, 如图 2-1-14 所示。

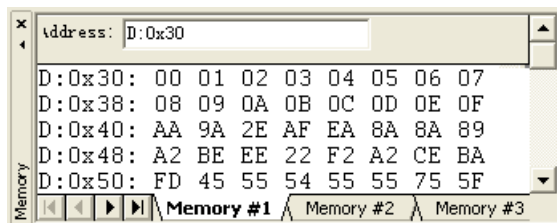


图 2-1-13 运行程序后存储器窗口

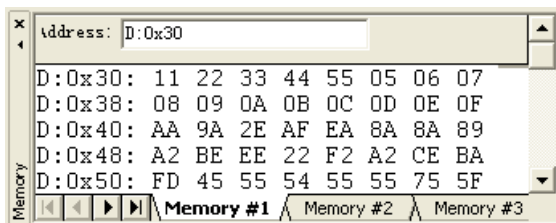


图 2-1-14 修改存储器内容

(6) 修改存储器的内容的方法还有一个, 就是在要修改的单元上点击鼠标右键, 弹出快捷菜单, 如图 2-1-15 所示, 选择 ‘Modify Memory at D:0x35’ 命令来修改 0x35 单元的内容, 这样每次只能修改一个单元的内容。

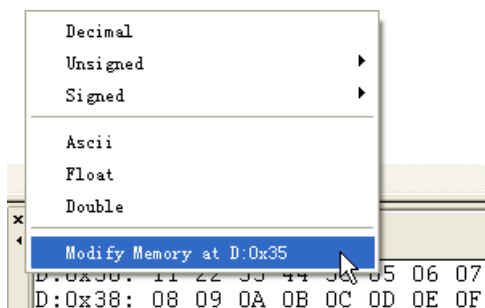



图 2-1-15 存储器修改单元

(7) 点击  命令, 可以复位 CPU, 重新调试运行程序, 点击  命令, 单步跟踪程序。

(8) 实验结束, 按系统的复位按键可以复位系统, 点击  命令, 退出调试。

在此以 Asml.Uv2 工程为例简要介绍了 Keil C51 的使用, Keil C51 功能强大, 关于 Keil C51 的使用需要通过日后的使用慢慢掌握。

随机光盘中提供了 SoftICE\_Tutorial.exe 文件, 该文件以动画的形式展示了基于 SST 公司的 SoftICE 在 Keil C51 环境下的调试过程。

## 2.2 数码转换实验

### 2.2.1 实验目的

1. 掌握不同进制数及编码相互转换的程序设计方法，加深对数码转换的理解；
2. 熟悉 Keil C51 集成开发环境的操作及程序调试的方法。

### 2.2.2 实验内容

1. 将 BCD 码整数 0~255 存入片内 RAM 的 20H、21H、22H 中，然后转换为二进制整数 00H~FFH，保存到寄存器 R4 中。

2. 将 16 位二进制整数存入 R3R4 寄存器中，转换为十进制整数，以组合 BCD 形式存储在 RAM 的 20H、21H、22H 单元中。


### 2.2.3 实验步骤

#### 1. BCD 整数转换为二进制整数

实验参考例程：(Asm2-1.asm)

```
ORG 0000H
LJMP MAIN
ORG 0100H
MAIN:  MOV R0, #20H      ;BCD 存放高位地址
        MOV R7, #03H    ;BCD 码 0--255, 最多 3 位
        CLR A
        MOV R4, A
LP1:   MOV A, R4
        MOV B, #0AH
        MUL AB           ;乘 10
        ADD A, @R0       ;加下一位的值
        INC R0           ;指向下一单元
        MOV R4, A        ;结果存入 R4
        DJNZ R7, LP1     ;转换未结束则继续
        SJMP MAIN        ;设置断点, 观察实验结果 R4 中的内容
END
```

实验步骤：


(1) 输入程序，检查无误后，编译、链接程序，首先给系统复位，然后点击  命令进入调试状态；

(2) 修改 20H、21H、22H 单元的内容，如：00H，05H，08H；

(3) 在 SJMP MAIN 语句行设置断点，然后运行程序；

(4) 程序遇到断点后停止程序运行，此时查看寄存器 R4 的内容，应为 3AH；

(5) 重新修改 20H、21H、22H 单元的内容，再次运行程序，验证程序的正确性；

(6) 实验结束，按复位键将系统复位，点击  退出调试状态。

## 2. 二进制整数转换为十进制整数

实验参考例程：(Asm2-2.asm)

```

        ORG 0000H
        LJMP MAIN
        ORG 0100H
MAIN:    MOV R0, #22H    ;转换结果低位地址
        MOV A, R0
        PUSH ACC        ;ACC 表示累加器 A 的直接地址
        MOV R7, #03H
        CLR A
LP1:     MOV @R0, A      ;结果存储地址清零
        DEC R0
        DJNZ R7, LP1
        POP ACC
        MOV R0, A
        MOV R7, #16
LP2:     PUSH ACC
        CLR C
        MOV A, R4        ;R4 中为二进制数的低位
        RLC A
        MOV R4, A
        MOV A, R3        ;R3 中为二进制数的高位
        RLC A
        MOV R3, A
        MOV B, #03H
LP3:     MOV A, @R0
        ADDC A, @R0      ;执行乘 2 操作
        DA A            ;十进制调整
        MOV @R0, A      ;结果保存
        DEC R0
        DJNZ B, LP3
        POP ACC
        MOV R0, A
        DJNZ R7, LP2
        LJMP MAIN      ;设置断点观察结果, 可进行下一次转换
        END

```

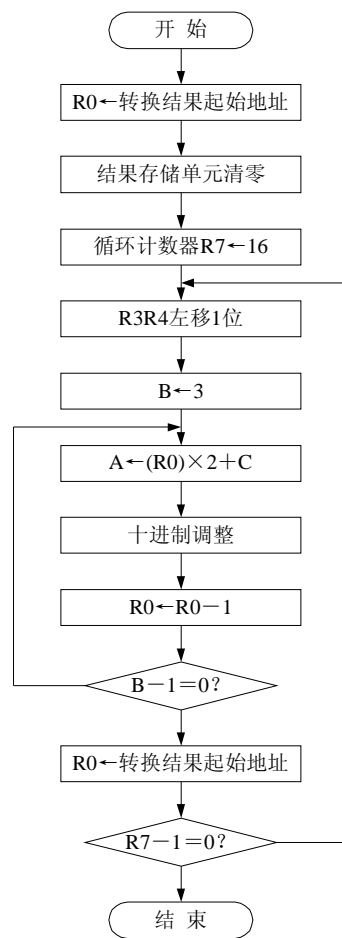


图 2-2-1 程序流程图

实验步骤：

- (1) 编写实验程序，程序流程图如图 2-2-1 所示，编译、链接无误后，进入调试状态；
- (2) 修改 R3R4 寄存器，例如 A2H、FCH；
- (3) 在 LJMP MAIN 语句行设置断点，然后运行程序；
- (4) 程序停止后，查看存储器 20H 的内容，应为：04H、17H、24H；
- (5) 反复修改 R3R4 寄存器的内容，运行实验程序，验证程序的正确性。

## 2.3 运算程序设计实验

### 2.3.1 实验目的

了解运算类指令以及运算类程序的设计方法。

### 2.3.2 实验内容

- 1. 多字节十进制加法程序，被加数存放于 20H 起始的 RAM 空间，加数存放于 2AH 起始的 RAM 空间，将两数相加，结果存放于 20H 起始的 RAM 空间；
- 2. 双字节无符号数乘法程序，被乘数在 R2R3 中，乘数在 R4R5 中，将相乘的结果保存在 20H~23H 中；
- 3. 双字节除法程序，被除数在 R7R6 中，除数在 R5R4 中，将商存入 R7R6 中，余数存入 R3R2 中。

### 2.3.3 实验步骤

#### 1. 多字节加法程序

实验参考例程：(Asm3-1.asm)

```
ORG 0000H
LJMP MAIN
ORG 0100H      ;执行程序前先修改 R7 的值
MAIN:  MOV R0, #20H      ;被加数起始地址
        MOV R1, #2AH     ;加数起始地址
        CLR C
LP1:   MOV A, @R0
        ADDC A, @R1      ;带进位加法运算
        DA A             ;十进制调整
        MOV @R0, A       ;保存运算结果
        INC R0            ;指向下一单元
        INC R1
        DJNZ R7, LP1
        CLR A
        MOV ACC.0, C
        MOV @R0, A       ;最高位有进位时此地址中为 1
        NOP
        SJMP MAIN        ;设置断点查看运算结果
                        ;若需继续运算，改变加数，被加数及 R7
END
```

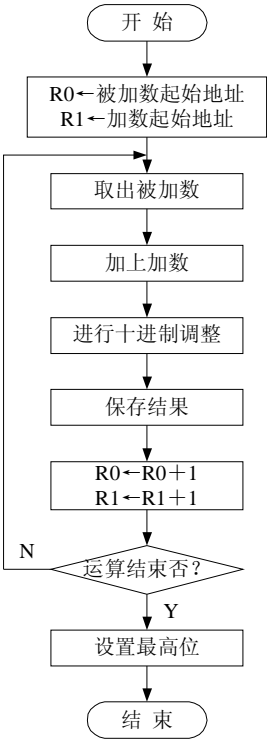


图 2-3-1 程序流程图

实验步骤：

- (1) 编写实验程序，程序流程图如图 2-3-1 所示，编译、链接无误后，进入调试状态；
- (2) 为被加数及加数赋值，即 4574 与 6728，低位在低字节，修改字节数 R7 为 2；

- (3) 在语句行 SJMP MAIN 设置断点，然后运行实验程序；
- (4) 当程序停止运行时，查看 20H 单元起始的内容，应为 02、13、02；
- (5) 修改被加数、加数及字节数 R7 的值，重新运算，验证程序的功能。

## 2. 双字节无符号数乘法

利用单字节乘法指令来扩展成多字节乘法运算，扩展时以字节为单位进行乘法运算。假定被乘数为 R2R3，乘数为 R4R5，乘积写入 R0 指向的内部 RAM 空间，运算法则见图 2-3-2。

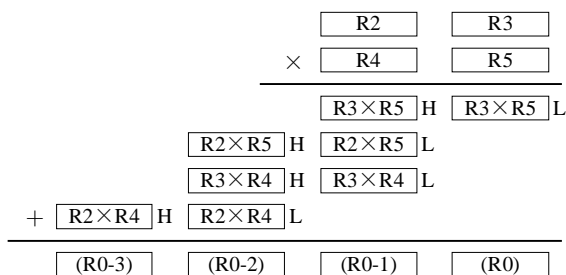


图 2-3-2 乘法运算法则

实验参考程序：(Asm3-2.asm)

```

ORG 0000H
LJMP MAIN
ORG 0100H
MAIN:  MOV R0, #23H      ;运算结果低位地址
        MOV A, R0
        PUSH ACC
        CLR A
        MOV R7, #4
LP1:    MOV @R0, A        ;运算结果存储单元清零
        DEC R0
        DJNZ R7, LP1
        POP ACC
        MOV R0, A
        MOV A, R3
        MOV B, R5
        MUL AB            ;计算 R3*R5
        MOV @R0, A        ;计算的低 8 位保存
        MOV A, B
        DEC R0
        MOV @R0, A        ;保存运算的高 8 位
        MOV A, R2
        MOV B, R5
        MUL AB            ;计算 R2*R5
        ADD A, @R0        ;此次运算低 8 位+R3*R5 运算的高 8 位
        MOV @R0, A
        DEC R0
        MOV A, B
        ADDC A, #00H      ;R2*R5 的高 8 位加进位位
        MOV @R0, A
        INC R0
        MOV A, R3

```

```

MOV B, R4
MUL AB          ;计算 R3*R4
ADD A, @R0
MOV @R0, A
MOV A, B
DEC R0
ADDC A, @R0
MOV @R0, A
DEC R0
CLR A
ADDC A, #00H
MOV A, @R0
MOV A, R2
MOV B, R4
MUL AB          ;计算 R2*R4
INC R0
ADD A, @R0
MOV @R0, A
MOV A, B
DEC R0
ADDC A, @R0
MOV @R0, A
NOP
LJMP MAIN       ;设置断点查看运算结果
                  ;如需进行下一次运算, 可修改 R2R3, R4R5 的值

END

```

#### 实验步骤:

- (1) 编写实验程序, 经编译、链接无误后, 联机调试;
- (2) 改变被乘数 R2R3 及乘数 R4R5 的值, 如 0x03、0x50 和 0x04、0x60;
- (3) 在语句行 LJMP MAIN 设置断点, 然后运行程序;
- (4) 程序停止后, 查看存储区 20H、21H、22H、23H 的内容, 应为 00、0E、7E、00;
- (5) 重新改变被乘数 R2R3 及乘数 R4R5 的值, 运行程序, 验证程序的正确性。

### 3. 双字节除法

51 指令系统中仅有单字节除法指令, 无法扩展为双字节除法。可以采用“移位相减”的算法来实现双字节的除法。例如要实现:

$$R7R6 \div R5R4 \rightarrow R7R6 \text{ (商)} \cdots \cdots R3R2 \text{ (余数)}$$

程序流程图如图 2-3-3 所示。

实验程序清单: (Asm3-3.asm)

```

ORG 0000H
LJMP MAIN
ORG 0100H
MAIN:  MOV A, R4          ;执行程序前为 R7R6(被除数), R5R4(除数)赋值
      JNZ DDIV0          ;除数不为 0, 转 DDIV0
      MOV A, R5
      JZ ERROR           ;除数为 0, 转 ERROR
DDIV0: MOV R2, #00H       ;余数寄存器清零
      MOV R3, #00H

```

```

MOV R1, #16 ;循环次数为 16
DDIV1: CLR C      ;R3R2R7R6 左移 1 位
MOV A, R6
RLC A
MOV R6, A
MOV A, R7
RLC A
MOV R7, A
MOV A, R2
RLC A
MOV R2, A
MOV A, R3
RLC A
MOV R3, A
MOV A, R2 ;部分余数减除数
SUBB A, R4 ;低 8 位相减
JC DDIV2 ;不够减, 转 DDIV2
MOV R0, A ;暂存相减结果
MOV A, R3
SUBB A, R5 ;高 8 位相减
JC DDIV2 ;不够减, 转 DDIV2
INC R6 ;够减, 则商为 1
MOV R3, A ;相减结果送 R3R2 中
MOV A, R0
MOV R2, A

```

```

DDIV2: DJNZ R1, DDIV1 ;16 位未除完则继续

```

```

CLR F0 ;除数合法标志

```

```

LJMP MAIN ;设置断点观察结果, 可继续下一轮运算

```

```

ERROR: SETB F0 ;除数非法标志

```

```

LJMP MAIN ;设置断点观察结果, 可继续下一轮运算

```

```

END

```

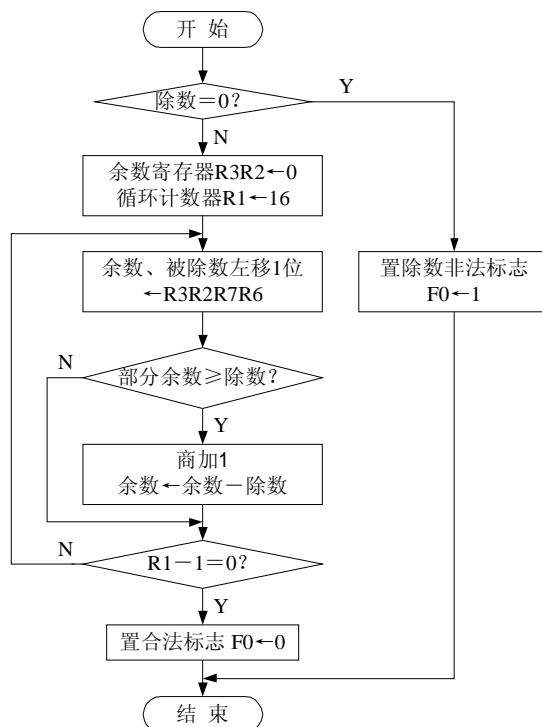


图 2-3-3 程序流程图

### 实验步骤:

- (1) 绘制流程图, 编写实验程序, 编译、链接无误后联机调试;
- (2) 修改被除数 R7R6 和除数 R5R4 的值, 如 0x46、0xEE 和 0x23、0x67;
- (3) 在语句行 LJMP MAIN 处设置断点, 运行实验程序;
- (4) 程序停止后, 查看寄存器 R7R6 (商) 与 R3R2 (余数), 应为 0x00、0x02 与 0x00、0x20;
- (5) 重新修改被除数及除数的值, 验证程序的功能。

## 2.4 查表程序设计实验

### 2.4.1 实验目的

学习查表程序的设计方法，熟悉 51 的指令系统。

### 2.4.2 实验内容

1. 通过查表的方法将 16 进制数转换为 ASCII 码；
2. 通过查表的方法实现  $y=x^2$ ，其中  $x$  为 0~9 的十进制数，以 BCD 码表示，结果仍以 BCD 码形式输出。

### 2.4.3 实验步骤

#### 1. 采用查表的方法将 16 进制数转换为 ASCII 码

根据 ASCII 码表可知，0~9 的 ASCII 码为 30H~39H，A~F 的 ASCII 码为 41H~46H，算法为（假定待转换的数存放在 R7 中）：

当  $R7 \leq 9$  时，相应的 ASCII 码为： $R7 + 30H$ ；

当  $R7 > 9$  时，相应的 ASCII 码为： $R7 + 30H + 07H$ 。

实验程序清单：(Asm4-1.asm)

```

                ORG 0000H
                LJMP MAIN
                ORG 0100H
MAIN:  MOV DPTR, #ASCTAB      ;表格首地址送 DPTR
        MOV A, R7            ;R7 中为待转换的数
        ANL A, #0FH          ;取低 4 位
        MOVC A, @A+DPTR      ;查表
        MOV R5, A            ;低 4 位转换结果送 R1
        MOV A, R7
        ANL A, #0F0H         ;取待转换数的高 4 位
        SWAP A               ;高 4 位与低 4 位交换
        MOVC A, @A+DPTR      ;查表
        MOV R6, A            ;高 4 位转换结果送 R2
        SJMP MAIN           ;设置断点观察结果

;ASCII 码表
ASCTAB: DB 30H, 31H, 32H, 33H, 34H
        DB 35H, 36H, 37H, 38H, 39H
        DB 41H, 42H, 43H, 44H, 45H, 46H
        END

```

实验步骤：

- (1) 编写实验程序，编译、链接无误后联机调试；
- (2) 将待转换的数存放在 R7 中，如令 R7 中的值为 0x86；
- (3) 在语句行 SJMP MAIN 设置断点，运行程序；



(4) 程序停止后查看寄存器 R6、R5 中的值，R6 中为高 4 位转换结果 0x38，R5 中为低 4 位转换结果 0x36；

(5) 反复修改 R7 的值，运行程序，验证程序功能。

## 2. 通过查表实现 $y=x^2$

x 为 0~9 的十进制数，存放与 R7 中，以 BCD 码的形式保存，结果 y 以 BCD 码的形式存放于寄存器 R6 中。

实验程序清单：(Asm4-2.asm)

```
ORG 0000H
LJMP MAIN
ORG 0100H
MAIN:  MOV DPTR, #SQR           ;取表格首地址
        MOV A, R7               ;要计算的值
        MOVC A, @A+DPTR         ;查表
        MOV R6, A               ;结果保存
        SJMP MAIN

;平方表
SQR:    DB 00H, 01H, 04H, 09H, 16H
        DB 25H, 36H, 49H, 64H, 81H
        END
```

实验步骤：

- (1) 编写实验程序，经编译、链接无误后，进入调试状态；
- (2) 改变 R7 的值，如 0x07；
- (3) 在语句行 SJMP MAIN 处设置断点，运行程序；
- (4) 程序停止后，查看寄存器 R6 中的值，应为 0x49；
- (5) 反复修改 R7 中的值，运行程序，验证程序功能。

## 2.5 数据排序实验

### 2.5.1 实验目的

熟悉 51 的指令系统，掌握数据排序程序的设计方法。

### 2.5.2 实验内容

在单片机片内 RAM 的 30H~39H 写入 10 个数，编写实验程序，将这 10 个数按照由小到大的顺序排列，仍写入 RAM 的 30H~39H 单元中。

### 2.5.3 实验步骤


根据实验内容要求，画出程序流程图，可参考图 2-5-1，编写实验程序。

实验程序清单：(Asm5.asm)

```
ORG 0000H
LJMP MAIN
ORG 0100H
MAIN:  MOV R0, #30H      ;数据起始地址
        MOV R7, #0AH     ;排序数据个数: 10 个数
LP1:   MOV A, R7
        MOV R6, A
        MOV A, R0
        MOV R1, A
        INC R1
LP2:   MOV A, @R0        ;取出一个数据
        CLR C
        SUBB A, @R1      ;与第二个数进行比较
        JC LP3           ;R0 中的数小于 R1 中的数则跳转
        MOV A, @R0       ;R0 中的数大于 R1 中的数, 交换数据
        XCH A, @R1
        MOV @R0, A
LP3:   INC R1
        DJNZ R6, LP2
        INC R0
        DJNZ R7, LP1

        SJMP $
        END
```

实验步骤:

- (1) 编写实验程序，编译、链接无误后联机调试；
- (2) 为 30H~39H 赋初值，如：在命令行中键入 E CHAR D:30H=9, 11H, 5, 31H, 20H, 16H, 1, 1AH, 3FH, 8 后回车，可将这 10 个数写入 30H~39H 中；
- (3) 将光标移到语句行 SJMP \$ 处，点击  命令，将程序运行到该行；
- (4) 查看存储器窗口中 30H~39H 中的内容，验证程序功能；

(5) 重新为 30H~39H 单元赋值，反复运行实验程序，验证程序的正确性。

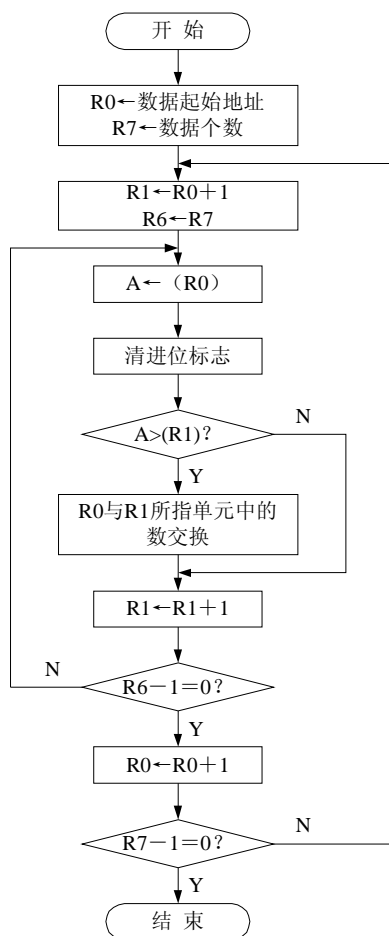


图 2-5-1 实验程序流程图

## 2.6 位操作实验

### 2.6.1 实验目的

掌握位指令的使用，学习位程序的设计方法。

### 2.6.2 实验内容

编写实验程序，计算  $Y = A \oplus B$ ，也可表示为  $Y = \overline{A}B + A\overline{B}$ 。

MCS-51 单片机内部有一个一位微处理器，借用进位标志 Cy 作为位累加器。位操作指令的操作对象是内部 RAM 的位寻址区，即字节地址为 20H~2FH 单元中连续的 128 位（位地址为 00H~7FH），以及特殊功能寄存器中的可位寻址的位。

### 2.6.3 实验步骤

程序要实现 A 与 B 的异或运算，将 A、B 分别存放在位地址 00H、01H 中，结果 Y 存放在位地址 04H 中。

实验程序清单：(Asm6.asm)

异或真值表

```
QA EQU 00H
QB EQU 01H
QY EQU 04H

ORG 0000H
LJMP MAIN
ORG 0100H
MAIN: MOV C, QA
      ANL C, /QB      ;C=QA(QB 非)
      MOV QY, C
      MOV C, QA
      CPL C
      ANL C, QB      ;C=(QA 非)QB
      ORL C, QY      ;C=QA(QB 非)+(QA 非)QB
      MOV QY, C
      SJMP MAIN      ;设置断点，观察 20H 或 C 中的值
      END
```

A	B	Y	20H
0	0	0	00
0	1	1	12
1	0	1	11
1	1	0	03

实验步骤：

- (1) 编写实验程序，经编译、链接无误后，联机调试；
- (2) 修改 20H 单元的值，例如 01H；
- (3) 在语句行 SJMP MAIN 设置断点，运行实验程序；
- (4) 程序停止运行后查看 20H 中的值，应为 11H；
- (5) 修改 20H 中的值，重新运行程序，验证程序的正确性。

## 第3章 单片机集成功能模块实验

SST89E554RC 集成有例如中断、定时/计数器、看门狗、PCA、串口和 SPI 等功能模块，通过本章的实验，学习、了解这些功能模块及其程序设计方法。

### 3.1 数字量输入输出实验

#### 3.1.1 实验目的

了解 P1 口作为输入输出方式使用时，CPU 对 P1 口的操作方式。

#### 3.1.2 实验内容

P1 口是 8 位准双向口，每一位均可独立定义为输入输出。编写实验程序，将 P1 口的低 4 位定义为输出，高 4 位定义为输入，数字量从 P1 口的高 4 位输入，从 P1 口的低 4 位输出控制发光二极管的亮灭。

#### 3.1.3 实验步骤

实验参考程序及实验步骤如下。

实验参考程序：(DigitIO.C)

```
#include "SST89x5x4.H"
void main(void)
{
    unsigned char data i;        //data 为存储器类型说明
    while(1)
    {
        P1 = P1 | 0xF0;          //声明高 4 位为输入
        i = P1;
        P1 = (i>>4)&0x0F;
    }
}
```

实验步骤：

1. 按图 3-1-1 所示，连接实验电路图，图中“圆圈”表示需要通过排线连接；
2. 编写实验程序，编译链接无误后进入调试状态；
3. 运行实验程序，观察实验现象，验证程序正确性；
4. 按复位按键，结束程序运行，退出调试状态；
5. 自行设计实验，验证单片机其它 IO 口的使用。

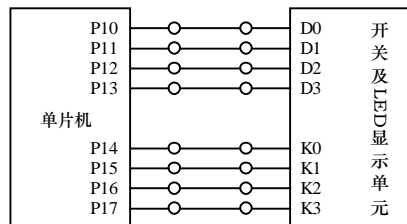


图 3-1-1 实验接线图

开关及 LED 显示单元原理图如图 3-1-2 所示。

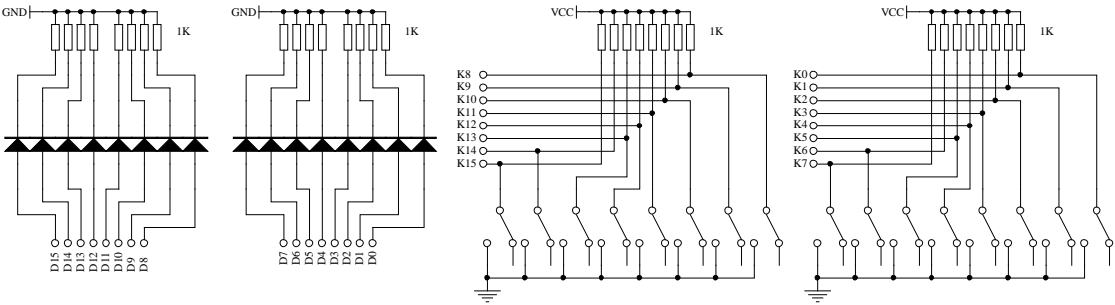


图 3-1-2 开关及 LED 显示单元原理图

## 3.2 中断系统实验

### 3.2.1 实验目的

了解 MCS-51 单片机的中断原理，掌握中断程序的设计方法。

### 3.2.2 实验内容

1. 单片机集成的定时器可以产生定时中断，利用定时器 0 和定时器 1，编写实验程序在 P1.0 及 P1.1 引脚上输出方波信号，通过示波器观察实验现象并测量波形周期。

2. 手动扩展外部中断 INT0、INT1，当 INT0 产生中断时，使 LED8 亮 8 灭闪烁 4 次；当 INT1 产生中断时，使 LED 由右向左流水显示，一次亮两个，循环 4 次。

因为 51 单片机加入了中断系统，从而提高了 CPU 对外部事件的处理能力和响应速度。增强型单片机 SST89E54RC 共有 8 个中断源，即外部中断 0 (INT0)、定时器 0 (T0)、外部中断 1 (INT1)、定时器 1 (T1)、串行中断 (TI 和 RI)、定时器 2 (T2)、PCA 中断和 Brown-out 中断。

中断使能寄存器 (IE)

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
A8H	EA	EC	ET2	ES	ET1	EX1	ET0	EX0	00H

中断使能 A (IEA)

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
E8H	—	—	—	—	EBO	—	—	—	00H

### 3.2.3 实验步骤

#### 1. 定时器中断

实验参考程序：(Int1.C)

```
#include "SST89x5x4.h"
sbit Wave1 = P1^0;           //声明位变量
sbit Wave2 = P1^1;
void int_timer0() interrupt 1
{
    Wave1 = ~Wave1;          //位变量取反
    TH0 = 0xF8;
    TL0 = 0x00;
}
void int_timer1() interrupt 3
{
    Wave2 = ~Wave2;
    TH1 = 0xF8;
    TL1 = 0x00;
}
```

```
void main()
{
    TH0 = 0xF8;           //初始化定时器 0
    TL0 = 0x00;
    TH1 = 0xF8;           //初始化定时器 1
    TL1 = 0x00;
    TMOD = 0x11;
    TCON = 0x50;
    IE = 0x8A;            //开中断
    while(1);
}
```

### 实验步骤:

- (1) 编写实验程序, 经编译、链接无误后, 启动调试功能;
- (2) 运行实验程序, 使用示波器观察 P1.0 及 1P.1 引脚上的波形;
- (3) 使用示波器测量波形周期, 改变计数值, 重新运行程序, 反复验证程序功能;
- (4) 按复位键退出调试状态。

## 2. 外部中断

### 实验参考程序: (INT2.C)

```
#include "SST89x5x4.h"
#include "Intrins.h"
void delay(void)
{
    unsigned int x;
    for(x=0; x<0xFFFF; x++);
}
void int0_isr() interrupt 0    //INT0 中断
{
    unsigned char j;
    for(j=0; j<4; j++)
    {
        P1 = 0xFF;           //使 LED 闪烁
        delay();
        P1 = 0x00;
        delay();
    }
}
void int2_isr() interrupt 2    //INT1 中断
{
    unsigned char i=0x03, j;
    for(j=0; j<16; j++)
    {
        P1 = i;              //使 LED 流水显示
        i = _crol_(i, 2);
        delay();
    }
    P1 = 0x00;
}
void main()
{
    P1 = 0x00;
```



```

IT0 = 1;
EX0 = 1;           //中断 0
IT1 = 1;
EX1 = 1;           //中断 1
EA = 1;
while(1);
}

```

实验步骤:

- (1) 按图 3-2-1 连接实验电路;
- (2) 编写实验程序, 编译、链接无误后启动调试;
- (3) 运行实验程序, 先按 KK1-, 观察实验现象, 然后按 KK2-, 观察实验现象;
- (4) 验证程序功能, 实验结束按复位按键退出调试。

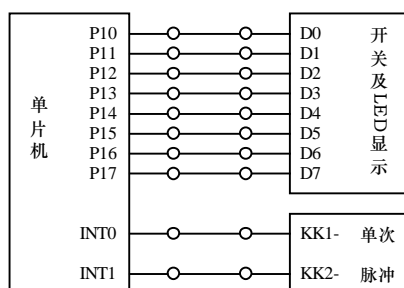


图 3-2-1 外中断实验接线图

单次脉冲单元原理图如图 3-2-2 所示。

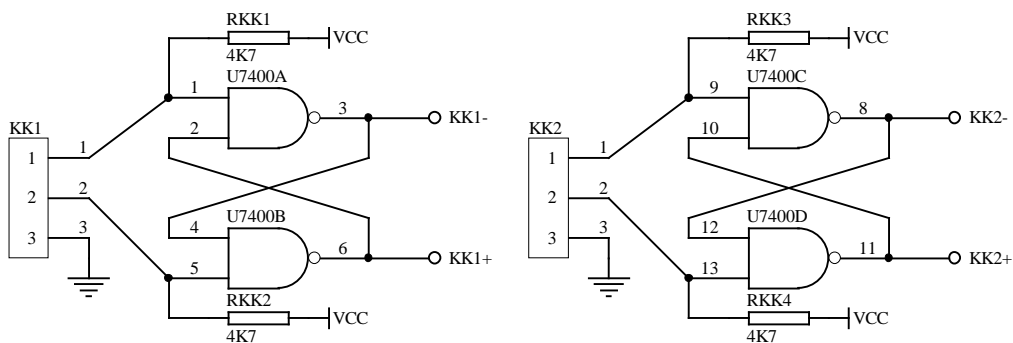


图 3-2-2 单次脉冲单元原理图

### 3.3 定时/计数器实验

#### 3.3.1 实验目的

1. 了解 MCS-51 单片机定/计数器的工作原理与工作方式；
2. 掌握定时/计数器 T0 和 T1 在定时器和计数器两种方式下的编程；
3. 学习定时/计数器 T2 的可编程时钟输出功能。

#### 3.3.2 实验内容

1. 使用定时器 0 与定时器 1 进行定时，在 P1.0 和 P1.1 引脚上输出方波信号，通过示波器观察波形输出，测量并记录方波周期。
2. 将定时/计数器 1 设定为计数器方式，每次计数到 10 在 P1.0 引脚上取反一次，观察发光二极管的状态变化。
3. 定时器 2 可以作为时钟发生器使用，并在 P1.0 引脚上输出占空比为 50% 的方波。编程定时器 2，使用示波器测量输出时钟，测量时钟周期。

#### 3.3.3 实验原理

通常，8051 单片机内部有 2 个 16 位定时/计数器，即定时器 0 (T0) 和定时器 1 (T1)。增强型单片机 SST89E554RC 内部还有一个 16 位定时器 T2，与其相关的特殊功能寄存器有 TL2、TH2、RCAP2L、RCAP2H、T2CON 等。

定时器/计数器 2 特殊功能寄存器

符号	描述	直接地址	位地址，符号或可选端口功能								复位值
			MSB				LSB				
T2CON <sup>1</sup>	定时器/计数器 2 控制	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2#	CP/RL2#	00H
T2MOD	定时器 2 模式控制	C9H	—	—	—	—	—	—	T2OE	DCEN	xxxxxx00b
TH2	定时器 2 MSB	CDH	TH2[7:0]								00H
TL2	定时器 2 LSB	CCH	TL2[7:0]								00H
RCAP2H	定时器 2 捕捉 MSB	CBH	RCAP2H[7:0]								00H
RCAP2L	定时器 2 捕捉 LSB	CAH	RCAP2L[7:0]								00H

注：1 表示该特殊功能寄存器可位寻址。

定时器/计数器 2 控制寄存器 (T2CON) 各位的含义简述如下：

TF2：定时器溢出标志，当定时器溢出时置位，必须由软件清除。当 RCLK=1 或 TCLK=1 时此位将不会被置位。

EXF2：定时器 2 外部标志，当 EXEN2=1 并且 T2EX 引脚上出现负跳变引起捕捉或重载发生时此位置 1。如果定时器 2 中断使能，EXF2=1 会引起中断，此位必须软件清除。DCEN=1 时，EXF2 不会引起中断。

RCLK：接收时钟标志，RCLK=1，串行口使用 T2 的溢出脉冲作为方式 1 和 3 下的接收时

钟；RCLK=0，串行口使用 T1 的溢出脉冲作为接收时钟。

TCLK：发送时钟标志，与 RCLK 的作用相同。

EXEN2：定时器 2 外部使能标志。EXEN2=1 且 T2 未被用于串口时钟时，若 T2EX 引脚上出现负跳变则出现捕捉或重载。EXEN2=0 时，T2 忽略 T2EX 引脚上的变化。

TR2：启动/停止定时器 2，为 1 时启动定时器 2。

C/T2#：定时器/计数器选择。C/T2#=1 为计数功能；C/T2#=0 为定时功能。

CP/RL2#：捕捉/重载标志。CP/RL2#=1，当 EXEN2=1 且 T2EX 引脚上出现负跳变时捕捉发生。CP/RL2#=0，T2 溢出时重载发生，或当 EXEN2=1 且 T2EX 引脚上出现负跳变时重载发生。如果 RCLK=1 或 TCLK=1，此位会被忽略，T2 溢出时自动重载。

定时器/计数器 2 模式寄存器 (T2MOD) 各位的含义简述如下：

T2OE：定时器 2 输出使能位。

DCEN：递减计数使能位。

### 3.3.4 实验步骤

#### 1. 定时器实验

按照实验要求编写实验程序，参考例程如下：(Timer.C)

```
#include "SST89x5x4.h"
sbit Wave1 = P1^0;
sbit Wave2 = P1^1;
void main()
{
    TMOD = 0x11;           //定时器方式寄存器
    TH0 = 0x0F8;           //定时器 0 计数初值
    TL0 = 0x00;
    TH1 = 0x0F8;           //定时器 1 计数初值
    TL1 = 0x00;
    TR0 = 1;               //启动定时器 0
    TR1 = 1;               //启动定时器 1
    while(1)
    {
        if(TF0 == 1)       //定时器 0 溢出标志
        {
            TH0 = 0x0F8;
            TL0 = 0x00;
            Wave1 = ~Wave1;
            TF0 = 0;
        }
        else if(TF1 == 1)  //定时器 1 溢出标志
        {
            TH1 = 0x0F8;
            TL1 = 0x00;
            Wave2 = ~Wave2;
            TF1 = 0;
        }
    }
}
```

实验步骤：

- (1) 编写实验程序，编译、链接后联机调试；
- (2) 运行实验程序，使用示波器观察 P1.0 与 P1.1 引脚上的波形并记录周期；
- (3) 改变计数初值，观察实验现象，验证程序功能。

## 2. 计数器实验

实验参考例程：(Count.C)

```
#include "SST89x5x4.h"
sbit P10Value = P1^0;
void main()
{
    TMOD = 0x60;           //设定定时器 1 计数方式
    TH1 = 0xF6;            //计数初值
    TL1 = 0xF6;
    TR1 = 1;               //启动定时器 1
    for(;;)
    {
        while(TF1 == 0);   //判定定时器 1 溢出标志
        P10Value = ~P10Value;
        TF1 = 0;
    }
}
```

实验步骤：

- (1) 按图 3-3-1 连接实验线路图；
- (2) 编写程序，联机调试；
- (3) 运行实验程序，按单次脉冲 KK1，观察发光管 D0 的状态，每 10 次变化一次；
- (4) 实验结束，按复位按键退出调试。

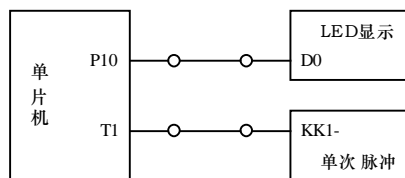


图 3-3-1 计数器实验接线图

## 3. 可编程时钟输出

引脚 P1.0 与 T2 复用，除作为普通 I/O 引脚外，还有两个功能，即为定时/计数器 2 输入外部时钟，输出占空比为 50% 的周期时钟。

如果将 T2 配置为时钟发生器，那么必须将 C/T2# 设置为 0，将 T2OE 设置为 1，并设置 TR2 为 1 以启动定时器。输出时钟的频率取决于晶振频率以及捕捉寄存器的重载值，公式如下：

输出频率 = 晶振频率 ÷ [n × (65536 - RCAP2H, RCAP2L)]

其中 n=2 (6 时钟模式) 或 n=4 (12 时钟模式)

晶振频率为 11.0592MHz，工作于 12 时钟模式下，输出频率的范围为：42Hz~2.76MHz。

实验参考例程：(ClkOut.C)

```
#include "SST89x5x4.h"
void main(void)
{
    RCAP2H = 0xFF;
    RCAP2L = 0x00;
    T2MOD = 0x02;           //定时器 2 输出使能
}
```

```
T2CON = 0x04;      //启动定时器 2
while(1);
}
```

实验步骤:

- (1) 编写实验程序, 编译、链接无误后联机调试;
- (2) 运行实验程序, 使用示波器观察 P1.0 引脚上的输出波形, 并测量波形周期;
- (3) 假定需要输出 1MHz 的方波信号, 试修改程序, 并使用示波器测量, 验证程序的正确性;
- (4) 实验结束, 按复位按键退出调试。

### 3.4 看门狗实验

#### 3.4.1 实验目的

了解看门狗的工作原理，学习看门狗的编程方法。

#### 3.4.2 实验内容

学习 SST89E554RC 的看门狗功能模块，编写实验程序，程序正常运行时 8 个 LED 闪烁，通过按键使看门狗产生超时，引起系统复位。

#### 3.4.3 实验原理

SST89E554RC 提供了一个可编程看门狗定时器 (WDT)，可以防止软件跑飞并自动恢复，提高系统的可靠性。

用户程序中如果使用了看门狗，那么必须在用户自己定义的时间内刷新 WDT，亦称“喂狗”。若在规定的时间内没有刷新 WDT，则产生内部硬件复位。WDT 以系统时钟 (XTAL1) 作为自己的时基，WDT 寄存器每隔 344064 个时钟加 1，时基寄存器 (WDTD) 的高 8 位被用作 WDT 的重载寄存器。WDT 的结构框图如图 3-4-1 所示。WDT 超时周期计算如下：

周期 = (255 - WDTD) × 344064 ÷ f<sub>CLK</sub> (XTAL1)

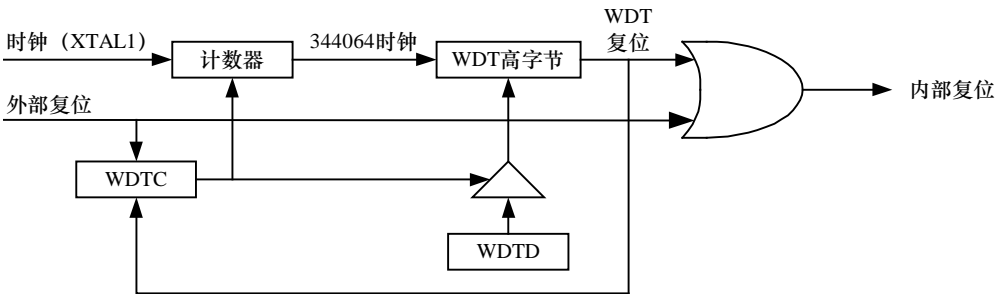


图 3-4-1 WDT 结构框图

看门狗定时器控制寄存器 (WDTC)

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
C0H	—	—	—	WDOUT	WDRE	WDTS	WDT	SWDT	xxx00x00b

各位说明如下：

- WDOUT: 看门狗输出允许  
0: 看门狗复位不在复位引脚上输出  
1: 如果看门狗复位允许位 WDRE=1，看门狗复位将在复位脚上输出复位信号 32 个时钟
- WDRE: 看门狗定时器复位允许  
0: 禁止看门狗定时器复位  
1: 允许看门狗定时器复位

WDTS:	看门狗定时器复位标志
	0: 外部硬件复位或上电会清除此位, 向此位写 1 会清除此位, 若由于看门狗引起的复位将不影响此位。
	1: 看门狗溢出, 此位置 1
WDT	看门狗定时器刷新
	0: 刷新完成, 硬件复位此位。
	1: 软件设置此位以强迫看门狗刷新, 俗称“喂狗”。
SWDT	启动看门狗定时器
	0: 停止 WDT
	1: 启动 WDT

### 看门狗定时器数据/重载寄存器 (WDTD)

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
85H	看门狗定时器数据/重载								00000000b

### 3.4.4 实验步骤

#### 实验程序清单: (WDT.C)

```
#include "SST89x5x4.h"
sfr WDTC = 0xC0;           //定义特殊功能寄存器
sbit WDT = 0xC1;           //定义位寻址变量
sbit WDTS = 0xC2;
bit bdata WDTFlag = 1;     //定义位类型变量
void delay(void)           //延时函数
{
    unsigned int i;
    for(i=0; i<0xFFFF; i++);
}
void int0_isr() interrupt 0 //外部中断 0 服务函数
{
    P1 = 0x00;
    WDTFlag = 0;
}
void main(void)
{
    WDTD = 0x9F;           //WDT 重载值
    WDTC = 0x1F;
    IT0 = 1;               //外部中断 0 初始化
    EX0 = 1;
    EA = 1;
    while(WDTFlag==1)
    {
        P1 = 0xFF;
        delay();
        P1 = 0x00;
        delay();
        WDT = 1;           //喂狗
    }
    while(1);
}
```

## 实验步骤:

- (1) 按图 3-4-2 连接实验电路图;
- (2) 编写实验程序, 编译、链接无误后启动调试;
- (3) 允许实验程序, LED 闪烁;
- (4) 按单次脉冲 KK1—, 对 WDT 停止刷新;
- (5) 经过大概 3 秒钟, 可观察软件界面, 产生复位, 程序停止运行 (注意界面变化);
- (6) 改变 WDT 的超时周期, 反复实验几次, 验证看门狗功能。

每次重新运行程序前, 都应该先停止调试, 然后重新启动调试, 这样方可保证系统正常工作。

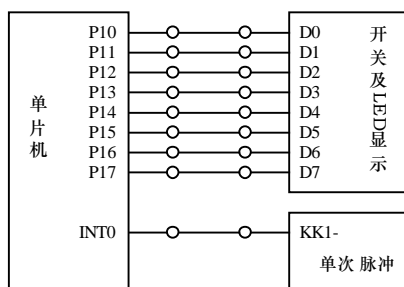


图 3-4-2 看门狗实验接线图



## 3.5 低功耗实验

### 3.5.1 实验目的

1. 了解单片机的空闲模式，及其程序设计；
2. 了解单片机的掉电模式，及其程序设计。

### 3.5.2 实验原理

SST89E554RC 为降低器件功耗，提供了两种模式：空闲模式和掉电模式。如表 3-5-1 所列。

表 3-5-1 低功耗模式

模 式	启动方法	MCU 状态	退出方法
空闲模式 Idle	软件设置 PCON 寄存器中的 IDL 位 MOV PCON, #01H	CLK 运行 中断、串口和定时/计数器有效 程序计数器 PC 停止 ALE 和 PSEN # 引脚为高电平 所有寄存器保持不变	使能中断或外部硬件复位。 中断产生会清除 IDL 位，退出空闲模式。用户应考虑在请求空闲模式指令后增加两到三条 NOP 指令，以消除可能出现的问题。
掉电模式 Power-Down	软件设置 PCON 寄存器中的 PD 位 MOV PCON, #02H	CLK 停止 SRAM 和 SFR 的值保持不变 ALE 和 PSEN # 引脚为低电平 仅外部电平触发的中断有效	使能外部电平触发中断或外部硬件复位。 中断产生会清除 PD 位以退出掉电模式。用户应考虑在请求空闲模式指令后增加两到三条 NOP 指令，以消除可能出现的问题。

### 3.5.3 实验内容及步骤

#### 1. 空闲模式实验

编写实验程序，控制 P1 口使 LED 循环流水显示，每次点亮一个 LED，通过 P2.7 请求进入空闲模式，通过外部中断 0 退出空闲模式。

实验参考程序：(Idle.C)

```
#include "SST89x5x4.h"
#include "Intrins.h"
sbit P27 = P2^7;
#define NOP _nop_()
void delay(void)
{
    unsigned int i;
    for(i=0; i<0xffff; i++);
}
void int0_isr() interrupt 0 { }
void main(void)
{
    unsigned char i=0x01;
    IT0 = 1;
```

```

EX0 = 1;
EA = 1;
while(1)
{
    P1 = i;
    i = _crol_(i, 1);    //将 i 左移 1 位
    delay();
    if(P27==0)
    {
        PCON = 1;      //使 IDL 位为 1，进入空闲模式
        NOP;
        NOP;
        NOP;
    }
}
}

```

实验步骤：

- (1) 按图 3-5-1 连接实验电路图；
- (2) 编写实验程序，编译、链接无误后启动调试；
- (3) 运行实验程序，按 KK2—请求空闲模式（按键不要松的太快）；
- (4) 观察实验现象，流水灯停止流动，使用示波器测量 ALE 信号，应为高电平；
- (5) 按 KK1—退出空闲模式，ALE 引脚出现波形，LED 继续流动；
- (6) 反复实验，验证程序正确性，熟悉单片机空闲模式的特征。

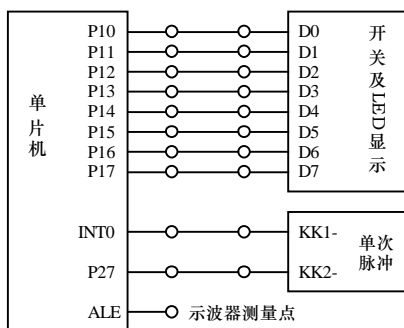


图 3-5-1 低功耗实验接线图

## 2. 掉电模式实验

编写实验程序，控制 P1 口使 LED 循环流水显示，每次点亮一个 LED，通过 P2.7 请求进入掉电模式，通过外部中断 0 退出掉电模式。

实验参考程序：(Pdown.C)

```

#include "SST89x5x4.h"
#include "Intrins.h"
sbit P27 = P2^7;
#define NOP _nop_()

void delay(void)
{

```

```
    unsigned int i;
    for(i=0; i<0xFFFF; i++);
}

void int0_isr() interrupt 0 {}

void main(void)
{
    unsigned char i=0x01;
    IT0 = 0;           //外部中断 0 电平触发
    EX0 = 1;
    EA = 1;
    P1 = 0x00;
    while(1)
    {
        P1 = i;
        i = _crol_(i, 1);
        delay();
        if(P27==0)
        {
            PCON = 0x02;    //使 PD 为 1，进入掉电模式
            NOP;
            NOP;
            NOP;
        }
    }
}
```

#### 实验步骤:

- (1) 按图 3-5-1 连接实验电路;
- (2) 编写实验程序, 编译、链接无误后启动调试;
- (3) 运行实验程序, 按 KK2—请求掉电模式, 观察 LED 的状态, 应停止流动, 使用示波器测量 ALE 引脚, 应为低电平;
- (4) 按 KK1—退出掉电模式, 观察 LED 的状态, 使用示波器测量 ALE 引脚, 应有波形输出;
- (5) 多次实验, 验证程序的正确性, 了解掉电模式的特征;
- (6) 按系统复位按键, 退出调试。

### 3.6 PCA 实验

#### 3.6.1 实验目的

- 1. 了解可编程计数器阵列（PCA）的结构和工作方式；
- 2. 掌握 PCA 在高速输出模式和脉冲宽度调制模式下的程序设计方法。

#### 3.6.2 实验原理

SST89E554RC 提供了一个特殊的 16 位定时器，该定时器具有五个 16 位捕捉/比较模块。每个模块都可被编程工作于以下四种模式：上升和/或下降沿捕捉、软件定时器、高速输出（HSO）和脉冲宽度调制（PWM）。第五个模块除上述四种模式外还可被编程为看门狗定时器。

每个模块都有一个外部引脚，与 P1 口复用：模块 0 连接至 P1.3（CEX0），模块 1 连接至 P1.4（CEX1），模块 2 连接至 P1.5（CEX2），模块 3 连接至 P1.6（CEX3），模块 4 连接至 P1.7（CEX4）。图 3-6-1 说明了 PCA 的结构框图。

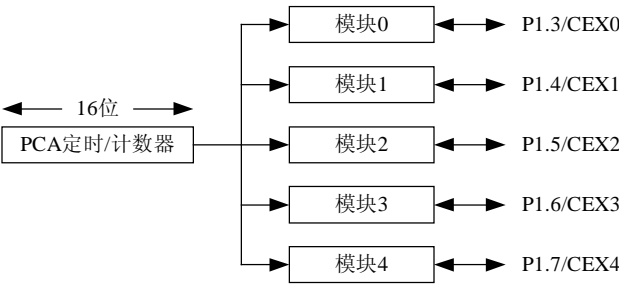


图 3-6-1 PCA 结构框图

PCA 定时器是一个自由运行的 16 位定时器，由寄存器 CH 和 CL（计数值的高、低字节）组成。PCA 定时器的 5 个模块共用一个时基，由 CMOD 寄存器的位 CPS1 和 CPS0 决定，详见表 3-6-1 所列。

表 3-6-1 PCA 定时/计数器信号源

CPS1	CPS0	12 时钟模式	6 时钟模式
0	0	$f_{osc}/12$	$f_{osc}/6$
0	1	$f_{osc}/4$	$f_{osc}/2$
1	0	定时器 0 溢出	定时器 0 溢出
1	1	ECI (P1.2) 引脚的外部时钟 最大= $f_{osc}/8$	ECI (P1.2) 引脚的外部时钟 最大= $f_{osc}/4$

与 PCA 相关的寄存器主要有 PCA 定时/计数器模式寄存器 CMOD、PCA 定时/计数器控制寄存器 CCON、PCA 捕捉/比较模块模式寄存器 CCAPMn，详见表 3-6-2 所列。

表 3-6-2 PCA 定时/计数器特殊功能寄存器

符号	描述	直接地址	位地址, 符号或可选端口功能								复位值
			MSB				LSB				
CH	PCA 定时/计数	F9H	CH[7:0]								00H
CL	器	E9H	CL[7:0]								00H
CCON <sup>1</sup>	PCA 控制寄存器	D8H	CF	CR		CCF4	CCF3	CCF2	CCF1	CCF0	00x00000b
CMOD	PCA 模式寄存器	D9H	CIDL	WDTE				CPS1	CPS0	ECF	00xxx000b
CCAP0H	PCA 模块 0 比较	FAH	CCAP0H[7:0]								00H
CCAP0L	/捕捉寄存器	EAH	CCAP0L[7:0]								00H
CCAP1H	PCA 模块 1 比较	FBH	CCAP1H[7:0]								00H
CCAP1L	/捕捉寄存器	EBH	CCAP1L[7:0]								00H
CCAP2H	PCA 模块 2 比较	FCH	CCAP2H[7:0]								00H
CCAP2L	/捕捉寄存器	ECH	CCAP2L[7:0]								00H
CCAP3H	PCA 模块 3 比较	FDH	CCAP3H[7:0]								00H
CCAP3L	/捕捉寄存器	EDH	CCAP3L[7:0]								00H
CCAP4H	PCA 模块 4 比较	FEH	CCAP4H[7:0]								00H
CCAP4L	/捕捉寄存器	EEH	CCAP4L[7:0]								00H
CCAPM0	PCA 比较/捕捉模块模式寄存器	DAH		ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x0000000b
CCAPM1		DBH		ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000b
CCAPM2		DCH		ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x0000000b
CCAPM3		DDH		ECOM3	CAPP3	CAPN3	MAT3	TOG3	PWM3	ECCF3	x0000000b
CCAPM4		DEH		ECOM4	CAPP4	CAPN4	MAT4	TOG4	PWM4	ECCF4	x0000000b

注：1 表示该特殊功能寄存器可位寻址。

#### PCA 定时/计数器模式寄存器 (CMOD)

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
D9H	CIDL	WDTE				CPS1	CPS0	ECF	00xxx000b

位说明如下：

- CIDL: 计数器空闲控制  
 0: 编程 PCA 计数器在空闲模式继续工作  
 1: 编程 PCA 计数器在空闲模式关闭
- WDTE: 看门狗定时器使能  
 0: 禁止 PCA 模块 4 的看门狗定时器功能  
 1: 允许 PCA 模块 4 的看门狗定时器功能
- CPS1、CPS0 见表 3-6-1
- ECF: 使能 PCA 计数器溢出中断  
 0: 禁止 CCON 中的 CF 位  
 1: 允许 CCON 中的 CF 位以产生中断

#### PCA 定时/计数器控制寄存器 (CCON)

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
D8H	CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0	00x00000b

位说明如下：

CF	PCA 计数器溢出标志 当计数器翻转时硬件设置此位。若 CMOD 中的 ECF 被设置，那么 CF 将标志一个中断。CF 可以被硬件或软件设置，但仅能由软件清除。
CR	PCA 计数器运行控制位 由软件设置此位以打开 PCA 计数器，关闭 PCA 计数器必须软件清除此位。
CCF4	PCA 模块 4 中断标志。当一个匹配或捕捉出现由硬件置此位，必须由软件清除。
CCF3	PCA 模块 3 中断标志。当一个匹配或捕捉出现由硬件置此位，必须由软件清除。
CCF2	PCA 模块 2 中断标志。当一个匹配或捕捉出现由硬件置此位，必须由软件清除。
CCF1	PCA 模块 1 中断标志。当一个匹配或捕捉出现由硬件置此位，必须由软件清除。
CCF0	PCA 模块 0 中断标志。当一个匹配或捕捉出现由硬件置此位，必须由软件清除。

每一个 PCA 模块都有一个模式特殊功能寄存器 (CCAPMn, n=0, 1, 2, 3, 4)，具体如表 3-6-2 所示。ECCF 位置 1 将使能 CCON 中的 CCF 标志，当 CCFn=1 时便产生中断。PWM 位为 1 可以使能 PWM 脉冲宽度调制模式。置位 MAT 位，当 PCA 计数器与模块的捕捉/比较寄存器发生匹配时 CCON 中的 CCFn 位置 1。如果 TOG 位为 1，CEXn 引脚会与相应的模块关联起来，当 PCA 计数器与模块的捕捉/比较寄存器发生匹配时进行翻转。CAPN 和 CAPP 位决定捕捉输入信号的有效沿，CAPN 为 1 捕捉下降沿，CAPP 为 1 捕捉上升沿，都为 1 则上升或下降沿都会进行捕捉。ECOM 位置 1 使能比较功能。

表 3-6-3 PCA 模块的模式

ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	模块功能
0	1	0	0	0	0	0/1	在引脚 CEX[4:0]上出现上升沿触发 16 位捕捉
0	0	1	0	0	0	0/1	在引脚 CEX[4:0]上出现下降沿触发 16 位捕捉
0	1	1	0	0	0	0/1	在引脚 CEX[4:0]上出现上升沿或下降沿都会触发 16 位捕捉
1	0	0	1	0	0	0/1	比较：软件定时器
1	0	0	1	1	0	0/1	比较：高速输出
1	0	0	0	0	1	X 注 1	比较：8 位 PWM
1	0	0	1	0/1 注 3	0	X 注 2	比较：PCA WDT (仅 CCAPM4)

注：1. 产生 PWM 不需要 PCA 中断；  
2. 为看门狗允许中断将消除看门狗功能；  
3. 为 0 则禁止翻转功能；为 1 允许在 CEX[4:0]引脚上产生翻转。

3.6.3 实验内容及步骤

1. 高速输出模式实验

在高速输出模式，PCA 计数器 (CH 和 CL) 与捕捉寄存器 (CCAPnH 和 CCAPnL) 之间数据匹配时在 CEX 输出引脚的状态发生翻转。

编写实验程序，使 PCA 的模块 0 工作与高速输出模式下，产生 1KHz 的方波，通过示波器

观察 CEX0 (P1.3) 的输出, 并测量周期。

实验参考程序: (PCA1.C)

```
#include "SST89x5x4.h"
sfr CH = 0xF9;
sfr CL = 0xE9;
sfr CCON = 0xD8;
sfr CMOD = 0xD9;
sfr CCAP0H = 0xFA;
sfr CCAP0L = 0xEA;
sfr CCAPM0 = 0xDA;
sbit CCF0 = 0xD8;
sbit CR = 0xDE;
sbit CF = 0xCF;
void PCA_ISR() interrupt 6 using 1
{
    CCF0 = 0;           //清除中断标志
    CH = 0x00;
    CL = 0x00;
}
void main()
{
    CMOD = 0x02;        //设置 PCA 定时器
                        //输入时钟=fosc/4
    CH = 0x00;          //计数初值
    CL = 0x00;
    CCAP0L = 0x66;      //设置事件触发值=fosc/8000
    CCAP0H = 0x05;
    CCAPM0 = 0x4D;      //设置 PCA 模块 0 为 HSO 模式
    IE = 0xC0;
    CR = 1;             //启动 PCA 定时器
    while(1);
}
```

实验步骤:

- (1) 编写实验程序, 经编译、链接无误后启动调试;
- (2) 运行实验程序, 使用示波器测量 CEX0 (P1.3) 引脚, 观察输出, 测量波形周期;
- (3) 修改程序, 使输出 2KHz 的波形, 并验证。

## 2. PWM 脉冲实验

编写实验程序, 配置 PCA 的模块 0 工作于 PWM 模式下, 输出占空比为 75% 的方波。

PWM 模式被用于产生一个连续的占空比可调的方波信号。脉冲宽度调制通过比较 PCA 定时器的低字节 (CL) 和比较寄存器的低字节 (CCAPnL) 产生 8 位 PWM 脉冲。当  $CL < CCAPnL$  时, 输出低电平。当  $CL > CCAPnL$  时, 输出高电平。输出频率取决于 PCA 定时器的信号源。占空比由写入比较寄存器的高字节 (CCAPnH) 控制, 计算如下:

$$CCAPnH = 256 (1 - \text{Duty cycle})$$

其中 CCAPnH 为 8 位整型数, Duty cycle 为百分数。

实验参考程序: (PCA2.C)

```
#include "SST89x5x4.h"
sfr CH = 0xF9;
sfr CL = 0xE9;
sfr CCON = 0xD8;
sfr CMOD = 0xD9;
sfr CCAP0H = 0xFA;
sfr CCAP0L = 0xEA;
sfr CCAPM0 = 0xDA;
sbit CCF0 = 0xD8;
sbit CR = 0xDE;
sbit CF = 0xCF;

void main(void)
{
    CMOD = 0x00;           //设置 PCA 定时器
    CL = 0x00;
    CH = 0x00;
    CCAP0L = 0x40;         //设置初始值与 CCAP0H 相同
    CCAP0H = 0x40;         //75%占空比
    CCAPM0 = 0x42;         //设置 PCA 模块 0 为 PWM 模式

    CR = 1;                //启动 PCA 定时器
    while(1);
}
```

#### 实验步骤:

- (1) 编写实验程序,经编译、链接无误后启动调试;
- (2) 运行实验程序,实验示波器测量 CEX0 (P1.3) 引脚,观察输出波形的占空比;
- (3) 修改实验程序,输出占空比为 30%的方波并验证。

PCA 其它工作模式请自行设计实验验证,加深对 PCA 的了解。



## 3.7 串口通讯实验

### 3.7.1 实验目的

1. 学习 MCS-51 单片机串口的工作原理及程序设计；
2. 了解使用 SSTEasyIAP11F.EXE 软件实现程序脱机运行的方法；
3. 熟悉启动加载代码与 SoftICE 相互切换的方法。

### 3.7.2 实验原理

MCS-51 单片机内部的全双工串行接口部分，包含有串行接收器和串行发送器。有两个物理上独立的接收缓冲器和发送缓冲器。接收缓冲器只能读出接收的数据，但不能写入。发送缓冲器只能写入发送的数据，但不能读出。因此可以同时收、发数据，实现全双工通讯。两个缓冲器是特殊功能寄存器 SBUF，它们公用地址为 99H，SBUF 是不可位寻址的。此外，还有两个寄存器 SCON 和 PCON 分别用于控制串行口的工作方式以及波特率，定时器 T1 可以用作波特率发生器。

SST89E554RC 提供了增强型全双工串行接口，具有帧错误检测和自动地址识别的功能。

由于 SST89E554RC 的串口用作调试目的，所以 Keil C51 软件提供了串口模拟窗口，可以借助此窗口调试串口通讯程序。也可以将程序编译生成目标代码（.HEX），脱机运行。

### 3.7.3 实验内容

编写实验程序，每隔一定的时间单片机向串口发送一次数据“Xi'an Tangdu Corp.”。

### 3.7.4 实验步骤

实验参考程序：(Serial.C)

```
#include "REG51.h"
#include "stdio.h"

/*****
 * 函数原型: void Init_Serial(void)
 * 函数描述: 初始化串口, 晶振为 11.0592MHz, 波特率为 19200bps
 *****/
void Init_Serial(void)
{
    SCON = 0x50;           // 串口工作方式 1
    TMOD = (TMOD & 0x0F) | 0x20; // 选择定时器 1 方式 2
    PCON = 0x80;           // 波特率倍增
    TH1 = 0xFD;            // 计数初值, 19200bps
    TR1 = 1;               // 启动定时器 1
    ES = 0;
}


void delay(void)
{
    unsigned int i;
```

```

        for(i=0; i<35000; i++);
    }
    void main(void)                //===== 主程序 =====//
    {
        Init_Serial();
        SBUF=0x00;
        while(1)
        {
            printf("Xi'an Tangdu Corp.\n");
            delay();
        }
    }
}

```

### 实验步骤:

- (1) 串口通讯实验电路如图 3-7-2 所示;
- (2) 编写实验程序, 经编译、链接无误后启动调试;
- (3) 进入调试界面, 点击  命令, 打开串口 1 监视窗口;
- (4) 运行实验程序, 观察此时有如图 3-7-2 所示输出;
- (5) 阅读 1.7 节的内容, 首先将系统程序由 SoftICE 切换到启动加载程序;
- (6) 将编译生成的 Hex 文件通过 SSTEasyIAP11F.EXE 软件下载到单片机内部 Flash 中;
- (7) 复位单片机, 打开超级终端或串口调试软件, 将端口号及波特率等设置好, 观察 PC 显示, 如图 3-7-3 和图 3-7-4 所示;
- (8) 实验结束, 重新将 SoftICE 下载到单片机系统区替换启动加载程序。

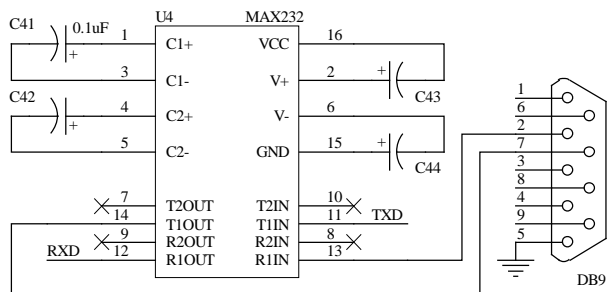


图 3-7-1 RS-232 电路

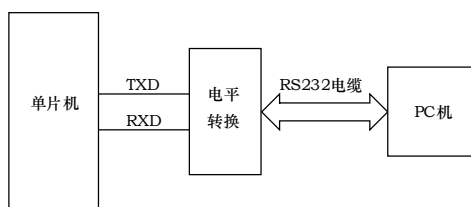


图 3-7-2 串口通讯实验电路

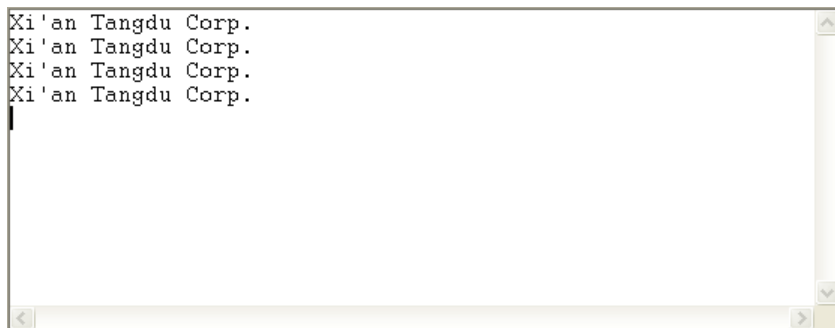


图 3-7-2 串口监视输出窗口



图 3-7-3 超级终端监视界面



图 3-7-4 串口监视界面

## 3.8 SPI 总线实验

### 3.8.1 实验目的

1. 了解 SPI 总线协议，熟悉 SST89E554RC 集成 SPI 接口的使用；
2. 了解 SPI 芯片 X5045 的功能，学习程序设计方法。

### 3.8.2 实验设备

PC 机一台，TD-NMC+实验装置一套。

### 3.8.3 实验内容

编写实验程序，对 X5045 进行读/写操作。

### 3.8.4 实验原理

#### 1. SST89E554RC 的 SPI 接口

SST89E554RC 集成了 SPI 接口，可以外接具有 SPI 接口的外围器件。具有如下特征：

- 主控或从控操作
- 最高 10MHz 位频率
- 数据传输时首先传输最高位或最低位可选
- 四种可编程位速率
- 具有结束标志 (SPIF)
- 写冲突标志 (WCOL)
- 可以从 idle 模式唤醒 (仅当工作于从模式时)

引脚定义如下：

P1.4/SS#：SPI 的从端口选择输入引脚；

P1.5/MOSI：主设备输出引脚，从设备输入引脚；

P1.6/MISO：主设备输入引脚，从设备输出引脚；

P1.7/SCK：主设备时钟输出引脚，从设备时钟输入引脚。

SPI 接口的寄存器定义如下：

SPI 控制寄存器 (SPCR)：

位置	7	6	5	4	3	2	1	0	复位值
D5H	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	04H

SPIE：若将 SPIE 和 ES 都置为 1，则 SPI 中断被使能；

SPE：SPI 使能位；当 SPE=0，禁止 SPI；SPE=1，使能 SPI，同时将信号 SS#、MOSI、MISO 和 SCK 连接到引脚 P1.4、P1.5、P1.6 和 P1.7 上；

DORD：数据传输顺序控制；

0：首先传输最高位 (MSB)

1：首先传输最低位 (LSB)

MSTR：主/从模式选择位

0: 选择主模式                      1: 选择从模式

CPOL: 时钟极性设置

0: 空闲时 SCK 为低电平（高有效）      1: 空闲时 SCK 为高电平（低有效）

CPHA: 时钟相位控制

0: 时钟的前沿产生移位触发              1: 时钟的后沿产生移位触发

SPR1、SPR0: SPI 时钟频率选择, 决定了 SCK 的频率, 与  $f_{osc}$  的关系;

SPR1	SPR0	$SCK = f_{osc} / \text{分频因子}$
0	0	4
0	1	16
1	0	64
1	1	128

SPI 状态寄存器 (SPSR):

位置	7	6	5	4	3	2	1	0	复位值
AAH	SPIF	WCOL	-	-	-	-	-	-	00xxxxxxb

SPIF: SPI 中断标志, 一旦数据传输完成, 此位被置为 1; 若 SPIE=1 且 ES=1, 则产生 SPI 中断; 读寄存器 SPSR, 然后访问 SPDR 可以清除此位;

WCOL: 写冲突标志, 若在数据传输期间 SPI 数据寄存器被写, 则此位置 1; 读寄存器 SPSR, 然后访问 SPDR 可以清除此位。

SPI 数据寄存器 (SPDR):

位置	7	6	5	4	3	2	1	0	复位值
86H	SPD7	SPD6	SPD5	SPD4	SPD3	SPD2	SPD1	SPD0	00H

## 2. SPI 器件 X5045 简介

X5045 是一款集电源管理、看门狗和串行 EEPROM 为一体的具有 SPI 接口的高性能器件。

X5045 的外部引脚如图 3-8-1 所示, 引脚定义如下:

CS#: 芯片选择输入;

SO: 串行数据输出;

SI: 串行数据输入;

SCK: 串行时钟;

WP#: 写包含输入, 为“0”禁止写入;

RST#/RST: 复位输出。

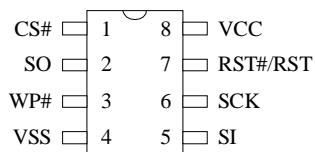


图 3-8-1 X5045 引脚图

X5045 包含了一个指令寄存器来控制器件的操作, 通过 SI 引脚将指令代码输入。所有的指令、地址及数据都由 SCK 进行时钟控制, 指令、地址及数据传输时首先传输 MSB 位。

当输入数据时, SI 引脚上的数据在 CS#信号变低后的 SCK 的第一个上升沿锁存; 而从 SO 引脚输出的数据在 SCK 的下降沿输出, 在整个操作过程中 CS#信号必须保持为低。

指令表:

指令名称	指令格式	指令描述
WREN	0000 0110 (06H)	设置写使能锁存 (使能写操作)
WRDI	0000 0100 (04H)	复位写使能锁存 (禁止写操作)

RDSR	0000 0101 (05H)	读状态寄存器
WRSR	0000 0001 (01H)	写状态寄存器 (看门狗和块锁)
READ	0000 A <sub>8</sub> 011	从指定的地址读存储器中的内容
WRITE	0000 A <sub>8</sub> 010	向指定的地址空间写数据 (1~16 个字节)

**写使能寄存器:** X5045 包含一个写使能寄存器, 在写操作之前必须对这个寄存器置 1, WREN 指令置位该寄存器, WRDI 指令复位该寄存器。该寄存器在上电或在完成写周期后自动复位。

**状态寄存器:** 状态寄存器包括了 4 个非易失状态位和 2 个易失状态位, 状态寄存器格式如下:

7	6	5	4	3	2	1	0	默认值
0	0	WD1	WD0	BL1	BL0	WEL	WIP	30H

WIP (Write In Progress): 该位属于易失位。用于指示是否在忙于写操作, 为 1 表示正在进行写操作, 为 0 表示未进行写操作。此位只读。

WEL (Write Enable Latch): 该位为易失、只读位。用于指示写使能寄存器的状态, 为 1 表示寄存器被置位, 为 0 表示寄存器被复位。

BL1、BL0 (Block Lock): 为非易失位, 由 WRSR 指令编程写入。指示被保护块的范围。

状态寄存器位		保护空间
BL1	BL0	X5045
0	0	None
0	1	\$180~\$1FF
1	0	\$100~\$1FF
1	1	\$000~\$1FF

WD1、WD0 (Watch Dog Timer): 非易失位, 由 WRSR 指令编程写入。选择看门狗定时输出间隔。

状态寄存器位		看门狗定时输出
WD1	WD0	
0	0	1.4 秒
0	1	600 毫秒
1	0	200 毫秒
1	1	禁止 (出厂定义)

SPI 总线单元原理图如图 3-8-2 所示, SPI 总线实验连线图如图 3-8-3 所示。

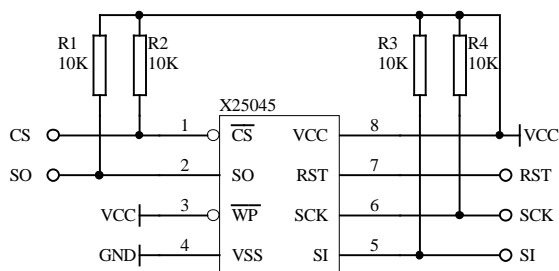


图 3-8-2 SPI 总线单元原理图

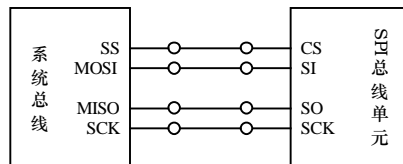


图 3-8-3 SPI 总线实验接线图

### 3.8.5 实验步骤

1. 按图 3-8-3 连接实验接线图；
2. 编写实验程序，编译、链接无误后启动调试；
3. 运行程序，先向 X5045 写入初始值，然后再从其中将写入的数据读出，观察读出数据与写入数据是否一致，验证程序功能；
4. 修改几组数据，反复验证程序功能。

#### 实验程序清单 (X5045.C)

```
#include "sst89x5x4.h"

sbit CS = P1^4;

void Delay(unsigned int j)
{
    unsigned int i;
    unsigned char k;
    for(i=0; i<j; i++)
        for(k=0; k<10; k++);
}

// 处理主从控制器间的字节传输
unsigned char SST_MasterIO(unsigned char HW_SPI_out)
{
    unsigned char temp;
    SPDR = HW_SPI_out;
    do
    {
        temp = SPSR & 0x80;
    } while (temp != 0x80);    // 判断 SPIF 标志位
    SPSR = SPSR & 0x7F;        // 清除 SPIF
    return SPDR;
}

// 读取状态寄存器
unsigned char Read_Status_Register()
{
    unsigned char byte = 0;
    CS = 0;                    // 使能设备
    SST_MasterIO(0x05);        // 发送 RDSR 命令
    byte = SST_MasterIO(0x00); // 接收字节
    CS = 1;                    // 禁止设备
    return byte;
}

// 写状态寄存器(看门狗设置)
void WRSR(unsigned char byte)
{
    CS = 0;                    // 使能设备
    SST_MasterIO(0x01);        // 发送 WRSR 命令
    SST_MasterIO(byte);        // 写入的状态字
    CS = 1;                    // 禁止设备
}

// 使能写操作
```

```

void WREN()
{
    CS = 0;                // 使能设备
    SST_MasterIO(0x06);    // 发送 WREN 命令
    CS = 1;                // 禁止设备
}
// 禁止写操作
void WRDI()
{
    CS = 0;                // 使能设备
    SST_MasterIO(0x04);    // 发送 WRDI 命令
    CS = 1;                // 禁止设备
}
// 读操作
unsigned char Read(unsigned char Addr)
{
    unsigned char byte = 0;
    CS = 0;                // 使能设备
    SST_MasterIO(0x03);    // 发送 READ 命令
    SST_MasterIO(Addr);    // 发送读取数据的地址地址
    byte = SST_MasterIO(0x00); // 接收字节
    CS = 1;                // 禁止设备
    return byte;
}
// 读多个数据(1--16)
void Read_Cont(unsigned char Addr, unsigned char no_bytes, unsigned char DataArray[])
{
    unsigned char i = 0;
    CS = 0;
    SST_MasterIO(0x03);    // 读命令
    SST_MasterIO(Addr);    // 地址
    for (i = 0; i < no_bytes; i++) // 读取 no_bytes 个数据
    {
        DataArray[i] = SST_MasterIO(0x00); // 接收数据并存储
    }
    CS = 1;
}
// 写一个字节
unsigned char Byte_Write(unsigned char Addr, unsigned char byte)
{
    WREN();
    Delay(1);
    CS = 0;                // 使能设备
    SST_MasterIO(0x02);    // 发送 Write 命令
    SST_MasterIO(Addr);    // 发送读取数据的地址地址
    SST_MasterIO(byte);    // 写一个字节
    CS = 1;                // 禁止设备
}

void main(void)
{
    unsigned char R_Byte[16], i;

```



```
SPCR = 0x51;                // 设置 SPI 的控制寄存器
//SPCR: bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
//      SPIE SPE  DORD MSTR CPOL CPHA SPR1 SPR0
for(i=0; i<16; i++)
{
    Byte_Write(i, i);        // 写 0--16 到地址 00 开始的连续 16 个单元中
    Delay(50);
}
Read_Cont(0x00, 16, R_Byte); // 读取地址 00 开始的连续 16 个单元中的数
while(1);
}
```

## 第 4 章 单片机系统扩展实验

MCS-51 单片机虽然在一块芯片上集成了一些基本功能模块，如定时器、计数器、中断、程序存储器、数据存储器等，但在实际应用中，往往要根据需要对单片机系统进行功能扩展，将详细讨论如何对单片机系统进行功能扩展。

### 4.1 静态存储器扩展实验

#### 4.1.1 实验目的

- 1. 掌握单片机系统中存储器扩展的方法；
- 2. 掌握单片机内部 RAM 和外部 RAM 之间数据传送的特点。

#### 4.1.2 实验内容

编写实验程序，在单片机内部一段连续 RAM 空间 30H~3FH 中写入初值 00H~0FH，然后将这 16 个数传送到 RAM 的 0000H~000FH 中，最后再将外部 RAM 的 0000H~000FH 空间的内容传送到片内 RAM 的 40H~4FH 单元中。

#### 4.1.3 实验原理

存储器是用来存储信息的部件，是计算机的重要组成部分，静态 RAM 是由 MOS 管组成的触发器电路，每个触发器可以存放 1 位信息。只要不掉电，所储存的信息就不会丢失。因此，静态 RAM 工作稳定，不要外加刷新电路，使用方便。但一般 SRAM 的每一个触发器是由 6 个晶体管组成，SRAM 芯片的集成度不会太高，目前较常用的有 6116 (2K×8 位)，6264 (8K×8 位) 和 62256 (32K×8 位)。本实验以 62256 为例讲述单片机扩展静态存储器的方法。

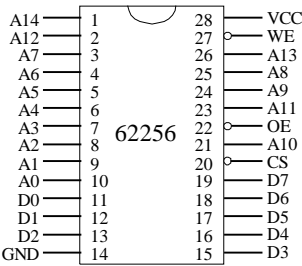


图 4-1-1 62256 引脚图

SST89E554RC 内部有 1K 字节 RAM，其中 768 字节(00H~2FFH)扩展 RAM 要通过 MOVX 指令进行间接寻址。内部 768 字节扩展 RAM 与外部数据存储器在空间上重叠，这要通过 AUXR 寄存器的 EXTRAM 位进行切换，AUXR 寄存器说明如下：

位置	D7	D6	D5	D4	D3	D2	D1	D0	复位值
8EH							EXTRAM	AO	xxxxxx00b

EXTRAM: 内部/外部 RAM 访问

0: 使用指令 MOVX @Ri/@DPTR 访问内部扩展 RAM，访问范围 00H~2FFH，300H 以上的空间为外部数据存储器；

1: 0000H~FFFFH 为外部数据存储器。

AO: 禁止/使能 ALE

0: ALE 输出固定的频率;

1: ALE 仅在 MOVX 或 MOVC 指令期间有效。

#### 4.1.4 实验步骤

1. 按图 4-1-2 连接使用电路;
2. 按实验内容编写实验程序, 经编译、链接无误后启动调试;
3. 打开存储器观察窗口, 在存储器 #1 的 Address 中输入 D:0x30, 在存储器 #2 的 Address 中输入 X:0x0000 来监视存储器空间;
4. 可单步运行程序, 观察存储器内容的变化, 或在 while(1)语句行设置断点再运行程序, 验证实验功能。

实验参考程序: (SRAM.C)

```
#include <Absacc.h>
sfr AUXR = 0x8E;           //特殊功能寄存器定义
void main()
{
    unsigned char i;
    AUXR = 0x02;           //访问外部 RAM
    for(i=0; i<=15; i++)
        DBYTE[0x30 + i] = i;    //将 00H~0FH 写入内部 RAM30H~3FH
    for(i=0; i<=15; i++)
        XBYTE[0x0000 + i] = DBYTE[0x30 + i]; //写入外部 RAM0000H~000FH 中
    for(i=0; i<=15; i++)
        DBYTE[0x40 + i] = XBYTE[0x6000 + i]; //写入内部 RAM40H~4FH 中
    while(1);
}
```

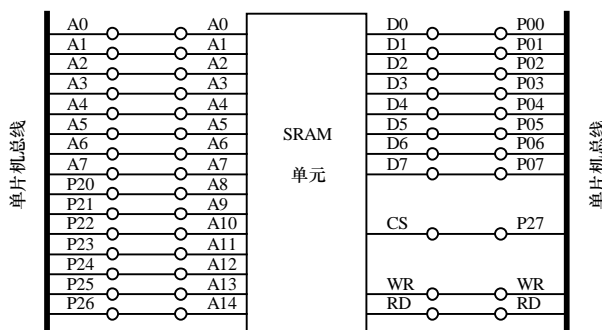


图 4-1-2 扩展存储器实验线路图

注: 连接实验线路时, 若使用 PITE 接口实验箱, 应将 BHE#和 BLE#信号接 GND; 若使用 PIT+实验箱, 需将 BE3~BE0 接 GND。

## 4.2 FLASH 存储器扩展实验

### 4.2.1 实验目的

1. 学习 FLASH 存储器的工作原理与读/写方式；
2. 了解 AT29C010 的编程特性。

### 4.2.2 实验内容

编写实验程序对 FLASH ROM 进行操作，要求对 FLASH 的读/写、数据保护功能、芯片擦除等特性进行验证。带保护写入 0~127 共 128 个数，不带保护写入 0x55 共 128 个。

### 4.2.3 实验原理

#### 1. FLASH ROM 简介

在系统可编程可擦除只读存储器 FLASH 通常称为“闪存”，该类型的存储器具有掉电时数据不丢失、扇区编程、芯片擦除、单一供电和高密度信息存储等特性，主要用于保存系统引导程序和系统参数等需要长期保存的重要信息，现广泛应用于各种产品中。AT29C010 为 5V 在系统可编程可擦除只读 FLASH，存储容量为 128K×8，封装为 PLCC32，其管脚如图 4-2-1 所示。引脚说明如下：

A0~A16：地址信号；

CE#：芯片使能信号；

OE#：输出使能信号；

WE#：写使能信号；

I/O0~I/O7：数据输入/输出信号；

NC：空脚，不连接。

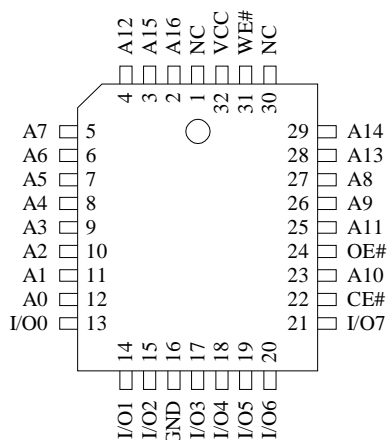


图 4-2-1 AT29C010 引脚图

#### 2. FLASH 的编程

AT29C010A 的数据编程以扇区为单位来进行操作。该器件共有 1024 个扇区，每个扇区为 128 字节。当进行数据编程时，首先将连续的 128 字节在内部进行锁存，然后存储器进入编程周期，将锁存器中的 128 字节数据依次写入存储器的扇区中，对于扇区的编程时间一般需要 10ms，接下来才能对下一个扇区进行编程。在对一个扇区进行编程前，存储器会自动擦除该扇区内的全部数据，然后才进行编程。

**软件数据保护：**AT29C010A 提供软件数据保护功能，在编程之前写入三个连续的程序命令，即按顺序将规定的数字写入指定的地址单元，便可以启动软件数据保护功能。在软件数据保护功能启动以后，每次编程之前都需要加上这三条命令，否则数据将无法写入 FLASH。断电不会影响该功能，即重新上电软件数据保护仍然有效。这样可以防止意外操作而破坏 FLASH 中的数据。如果需要去除软件数据保护功能，可以用同样的方法写入连续的六个命令。启动软件数据

保护功能的命令序列如图 4-2-2 的 (a) 所示, 取消软件数据保护功能的命令序列如图 4-2-2 的 (b) 所示。

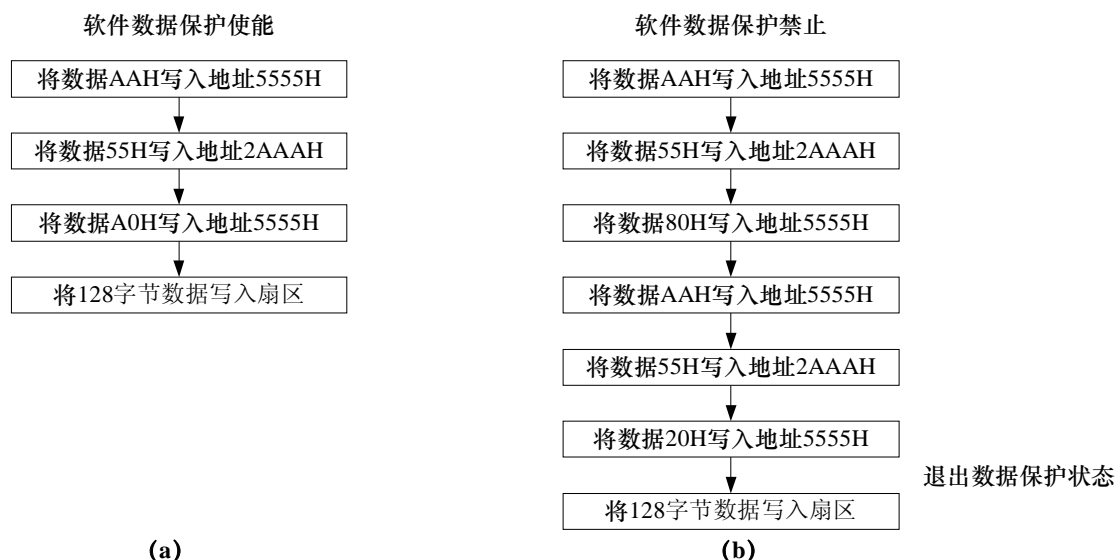


图 4-2-2 软件数据保护命令序列

**芯片擦除:** AT29C010A 可以对整个芯片进行擦除, 通过写入六个连续的命令实现, 具体命令序列如图 4-2-3 所示。

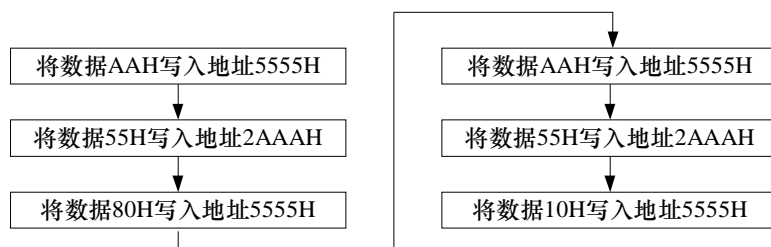


图 4-2-3 芯片擦除命令序列

## 4.2.4 实验步骤

实验程序清单: (Flash.C)

```

#define START_ADDR      ((unsigned char volatile xdata *) 0x0000)
#define ADDR_UNLOCK1    ((unsigned char volatile xdata *) 0x5555)
#define ADDR_UNLOCK2    ((unsigned char volatile xdata *) 0x2AAA)
sfr AUXR = 0x8E;
void delay()
{
    unsigned int i;
    for(i=0; i<1000; i++);
}
void unprotect()          // 取消软件数据保护
{
    *ADDR_UNLOCK1 = 0xAA;
  
```

```

        *ADDR_UNLOCK2 = 0x55;
        *ADDR_UNLOCK1 = 0x80;
        *ADDR_UNLOCK1 = 0xAA;
        *ADDR_UNLOCK2 = 0x55;
        *ADDR_UNLOCK1 = 0x20;
    }
void erase()                // 芯片擦除
{
    *ADDR_UNLOCK1 = 0xAA;
    *ADDR_UNLOCK2 = 0x55;
    *ADDR_UNLOCK1 = 0x80;
    *ADDR_UNLOCK1 = 0xAA;
    *ADDR_UNLOCK2 = 0x55;
    *ADDR_UNLOCK1 = 0x10;
}
void unp_write()           // 无保护写
{
    unsigned int i;
    unsigned char *Des;
    Des = START_ADDR;
    for(i=0; i<128; i++)
    {
        *Des = 0x55;        //全部写 0x55
        Des++;
    }
}
void p_write()             // 带保护写
{
    unsigned int i;
    unsigned char *Des;
    Des = START_ADDR;
    *ADDR_UNLOCK1 = 0xAA;
    *ADDR_UNLOCK2 = 0x55;
    *ADDR_UNLOCK1 = 0xA0;
    for(i=0; i<128; i++)    //写 0~128 到 Flash
    {
        *Des = i;
        Des++;
    }
}
void main()
{
    AUXR = 0x2;

    ①p_write();             // 带保护写
    delay();

    ②unp_write();           // 不带保护写
    delay();

    unprotect();           // 去除保护
    ③unp_write();
    delay();

    ④erase();               // 擦除
    while(1);
}

```

## 实验步骤:

1. 按图 4-2-4 连接实验电路;
2. 编写实验程序, 可参考上述程序, 编译、链接后启动调试;
3. 打开存储器观察窗口, 在存储器 # 1 的 Address 栏内输入 X:0x0000, 查看存储器的内容;
4. 执行完标号为①的函数, 带保护写 Flash, 写入内容 0~7F, 观察存储器窗口;
5. 执行完标号为②的函数, 不带保护写 Flash, 观察存储器窗口, 正确情况下数据不会改变, 仍为 0~7F;
6. 执行完标号为③的函数, 首先会去除写保护, 然后不带保护写 Flash, 观察存储器窗口, 此时应显示 128 个 55;
7. 执行完标号为④的函数, 可以将整个 Flash 擦除, 观察存储器窗口, 内容变为全 FF, 表示 Flash 已被擦除;
8. 通过一步一步实验, 了解 Flash 的特性, 实验结束, 按复位按键退出调试。

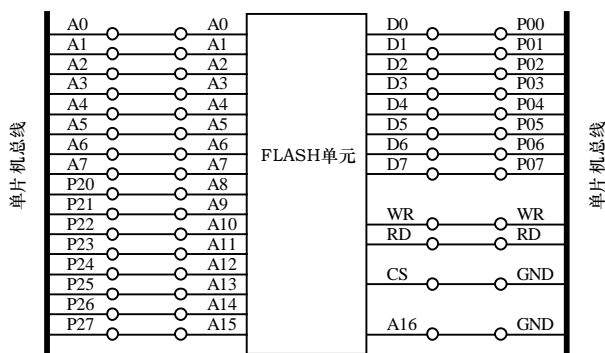


图 4-2-4 FLASH ROM 参考接线图

4.3 A/D 转换实验

4.3.1 实验目的

- 1. 学习理解模/数信号转换的基本原理；
- 2. 掌握模/数转换芯片 ADC0809 的使用方法。

4.3.2 实验内容

编写实验程序，将 ADC 单元中提供的 0V~5V 信号源作为 ADC0809 的模拟输入量，进行 A/D 转换，转换结果通过变量进行显示。

4.3.3 实验原理

ADC0809 包括一个 8 位的逐次逼近型的 ADC 部分，并提供一个 8 通道的模拟多路开关和联合寻址逻辑。用它可直接输入 8 个单端的模拟信号，分时进行 A/D 转换，在多点巡回检测、过程控制等应用领域中使用非常广泛。ADC0809 的主要技术指标为：

- 分辨率：8 位
- 单电源：+5V
- 总的不可调误差：±1LSB
- 转换时间：取决于时钟频率
- 模拟输入范围：单极性 0~5V
- 时钟频率范围：10KHz~1280KHz

ADC0809 的外部管脚如图 4-3-1 所示，地址信号与选中通道的关系如表 4-3-1 所示。

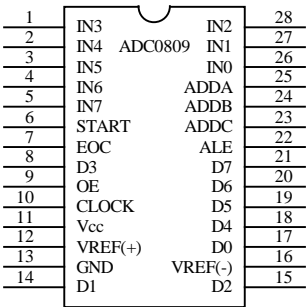


图 4-3-1 ADC0809 外部引脚图

表 4-3-1 地址信号与选中通道的关系

地 址			选中通道
A	B	C	
0	0	0	IN0
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

实验程序清单：(AD0809.C)

```
#include "SST89x5x4.h"
#include "Absacc.h"
#define STARTAD XBYTE[0x7F00]
#define ADRESULT XBYTE[0x7F08]
```



```

sbit ADBUSY = P3^3;
void Delay()
{
    unsigned char i;
    for(i=0; i<100; i++);
}
unsigned char AD0809(void)
{
    unsigned char result;
    STARTAD = 0;           //启动 AD
    while(ADBUSY == 1);    //等待转换结束
    Delay();
    result = ADRESULT;
    return result;         //返回转换结果
}
void main(void)
{
    unsigned char ADV;      //变量
    while(1)
    {
        ADV = AD0809();
        Delay();            //设置断点
    }
}

```

A/D 转换单元原理图如图 4-3-2 所示。

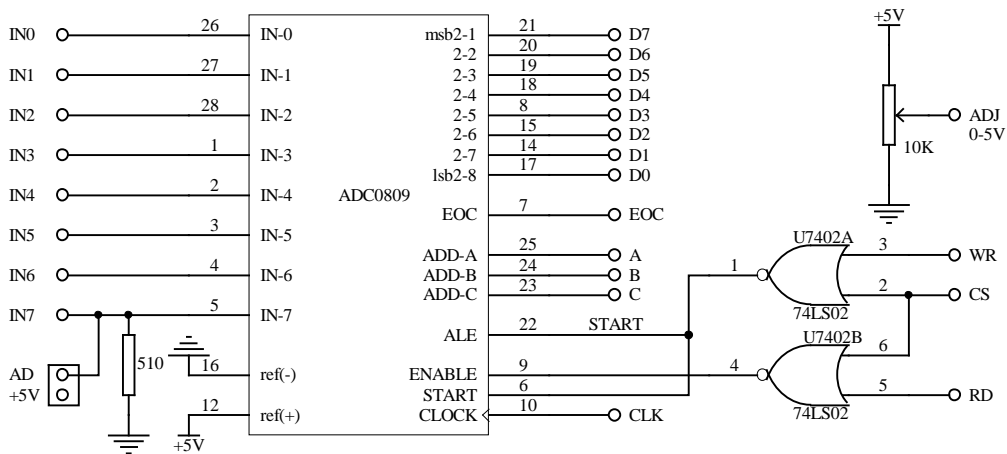


图 4-3-2 A/D 转换单元原理图

### 4.3.4 实验步骤

1. 按图 4-3-3 连接实验线路，AD 的时钟线需要与实验平台中的系统总线单元的 CLK 相连；
2. 编写实验程序，经编译、链接无误后装入系统，启动调试；
3. 将变量 ADV 添加到变量监视窗口中；
4. 在 Delay()语句行设置断点，使用万用表测量 ADJ 端的电压值，计算对应的采样值，然后运行程序；

5. 程序运行到断点处停止运行，查看变量窗口中 ADV 的值，与计算的理论值进行比较，看是否一致（可能稍有误差，相差不大）；
6. 调节电位器，改变输入电压，比较 ADV 与计算值，反复验证程序功能；制表并记录结果。

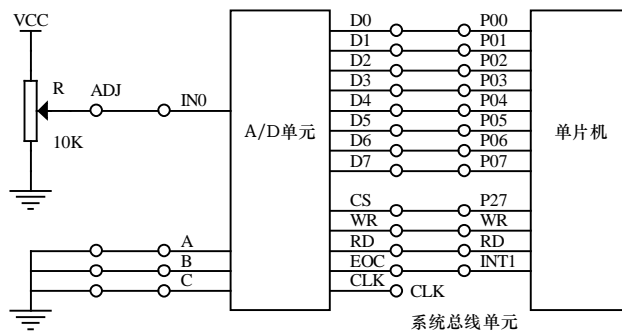


图 4-3-3 AD 转换实验接线图

## 4.4 D/A 转换实验

### 4.4.1 实验目的

1. 学习数/模转换的基本原理；
2. 掌握 DAC0832 的使用方法。

### 4.4.2 实验内容

设计实验电路图实验线路并编写程序，实现 D/A 转换，要求产生锯齿波、脉冲波，并用示波器观察电压波形。

### 4.4.3 实验原理

D/A 转换器是一种将数字量转换成模拟量的器件，其特点是：接收、保持和转换的数字信息，不存在随温度、时间漂移的问题，其电路抗干扰性较好。大多数的 D/A 转换器接口设计主要围绕 D/A 集成芯片的使用及配置响应的外围电路。DAC0832 是 8 位芯片，采用 CMOS 工艺和 R-2RT 形电阻解码网络，转换结果为一对差动电流  $I_{out1}$  和  $I_{out2}$  输出，其主要性能参数如表 4-4-1 示，引脚如图 4-4-1 所示。

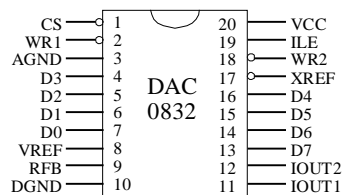


图 4-4-1 DAC0832 引脚图

表 4-4-1 DAC0832 性能参数

性能参数	参数值
分辨率	8 位
单电源	+5V ~ +15V
参考电压	+10V ~ -10V
转换时间	1 $\mu$ s
满刻度误差	$\pm 1$ LSB
数据输入电平	与 TTL 电平兼容

### 4.4.4 实验步骤

1. 实验接线图如图 4-4-2 所示，按图接线；
2. 编写实验程序，经编译、链接无误后装入系统，启动调试；
3. 运行程序，用示波器测量 DA 的输出，观察实验现象；
4. 自行编写实验程序，产生三角波形，使用示波器观察输出，验证程序功能。

### 实验程序清单：(DA0832.C)

```
#include <Absacc.h>

#define DA XBYTE[0x7FFF]
```

```
void main(void)
{
    unsigned int i;
    while(1)
    {
        for(i=0; i<255; i++)
            DA = i;          //写 DA
    }
}
```

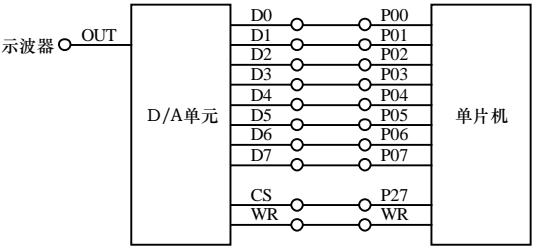


图 4-4-2 D/A 实验接线图

D/A 转换单元原理图如图 4-4-3 所示。

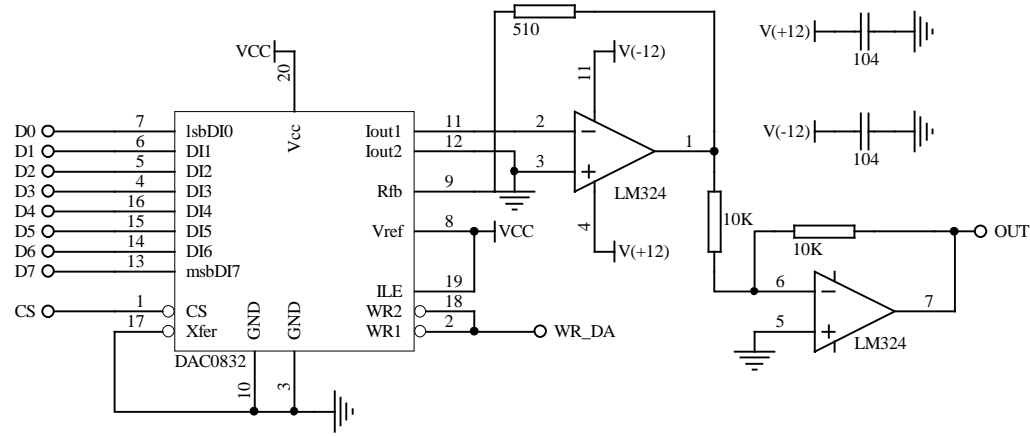


图 4-4-3 A/D 转换单元原理图

## 4.5 键盘扫描及显示设计实验

### 4.5.1 实验目的

1. 了解 8255 的工作方式及应用；
2. 了解键盘扫描及数码显示的基本原理，熟悉 8255 的编程。

### 4.5.2 实验内容

将 8255 单元与键盘及数码管显示单元连接，编写实验程序，扫描键盘输入，并将扫描结果送数码管显示。键盘采用  $4 \times 4$  键盘，每个数码管显示值可为 0~F 共 16 个数。实验具体内容如下：将键盘进行编号，记作 0~F，当按下其中一个按键时，将该按键对应的编号在一个数码管上显示出来，当再按下一个按键时，便将这个按键的编号在下一个数码管上显示出来，数码管上可以显示最近 4 次按下的按键编号。

### 4.5.3 实验原理

并行接口是以数据的字节为单位与 I/O 设备或被控制对象之间传递信息。CPU 和接口之间的数据传送总是并行的，即可以同时传递 8 位、16 位或 32 位等。8255 可编程外围接口芯片是 Intel 公司生产的通用并行 I/O 接口芯片，它具有 A、B、C 三个并行接口，用 +5V 单电源供电，能在以下三种方式下工作：方式 0--基本输入/输出方式、方式 1--选通输入/输出方式、方式 2--双向选通工作方式。8255 的内部结构及引脚如图 4-5-1 所示，8255 工作方式控制字和 C 口按位置位/复位控制字格式如图 4-5-2 所示。

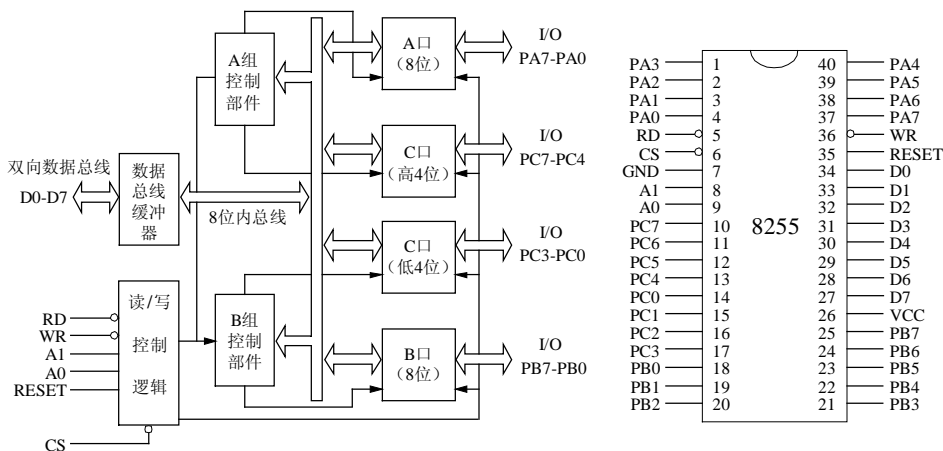


图 4-5-1 8255 内部结构及外部引脚图

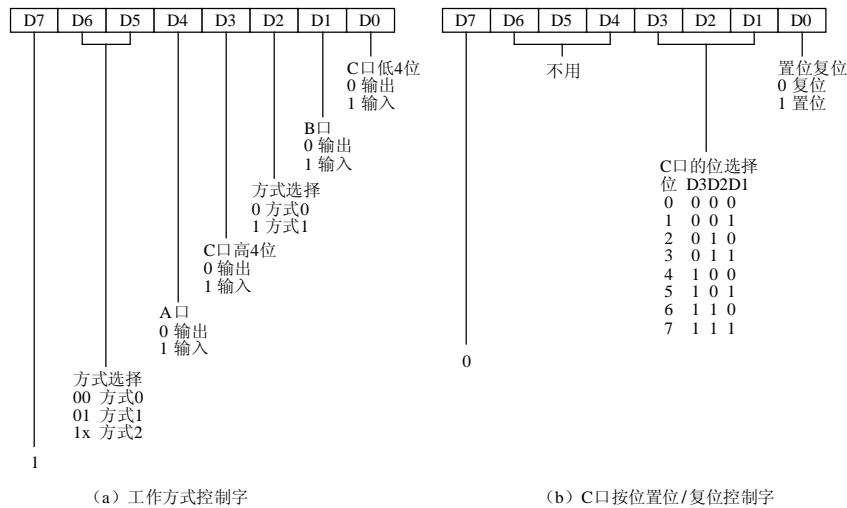


图 4-5-2 8255 控制字格式

键盘扫描及数码管显示单元原理图如图 4-5-3 所示。

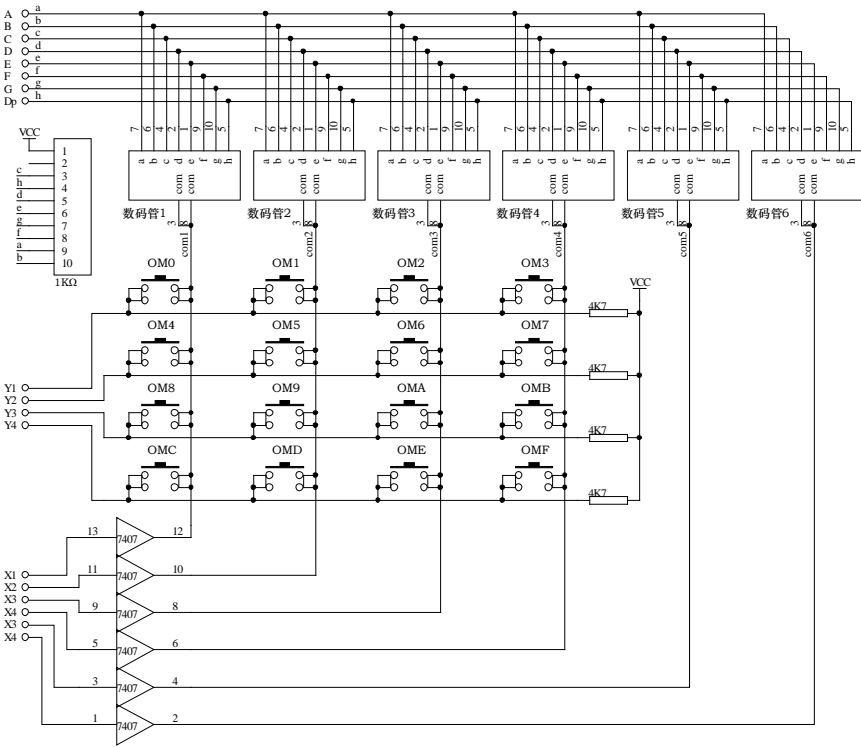


图 4-5-3 键盘扫描及数码管显示单元原理图

8255 单元原理图如图 4-5-4 所示。

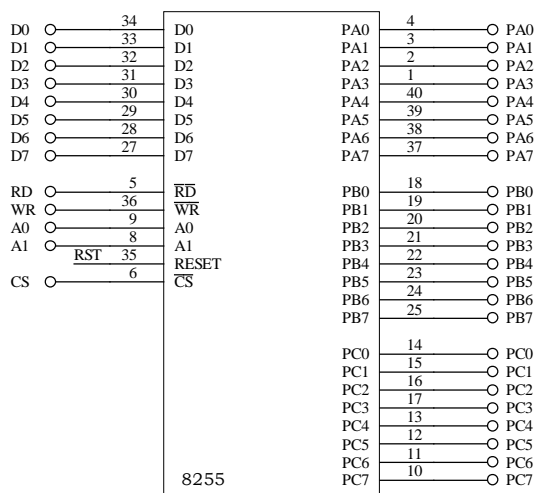


图 4-5-4 8255 单元原理图

#### 4.5.4 实验步骤

1. 按图 4-5-5 连接线路图；
2. 编写实验程序，检查无误后编译、连接后启动调试；
3. 运行实验程序，按下按键，观察数码管的显示，验证程序功能。

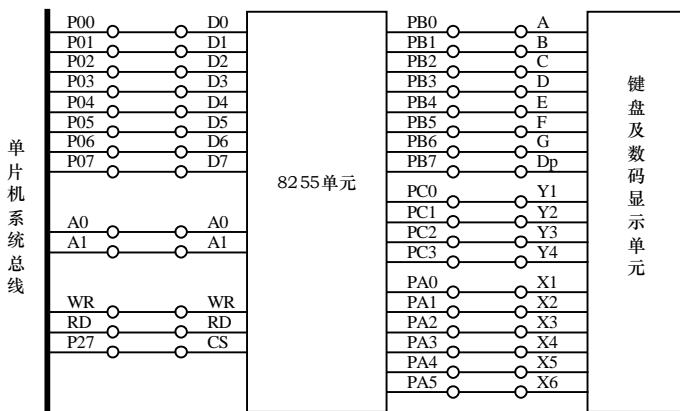


图 4-5-5 键盘扫描及数码显示实验接线图

#### 实验程序清单：(KeyScan.C)

```
#include "Absacc.h"

#define C8255_A    XBYTE[0x7F00]    //8255 端口地址定义
#define C8255_B    XBYTE[0x7F01]
#define C8255_C    XBYTE[0x7F02]
#define C8255_CON  XBYTE[0x7F03]

//数码管显示编码
unsigned char a[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
```

```

        0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
unsigned char b[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};    //显示缓冲

unsigned char key_down;
unsigned char key_value;
unsigned char key_count;

void delay(unsigned int time)
{
    unsigned int i;
    for(i=0; i<time; i++);
}
void keyscan()                //按键扫描函数
{
    unsigned char cc;
    C8255_A = 0x00;           //X1~X4 置 0
    cc = C8255_C;              //得到 Y1~Y4 的值
    key_down = (~cc) & 0x0f;
}
void display()                //显示函数
{
    unsigned char i, j = 0xDF;
    for(i=0; i<6; i++)
    {
        C8255_A = 0xFF;
        C8255_B = a[b[i]];    //查表输出显示
        C8255_A = j;
        delay(0x100);
        j = (j>>1)|(j<<7);
    }
}
void clear()                  //清屏
{
    C8255_B = 0x00;
}
void writebuffer()
{
    b[key_count] = key_value;
    key_count--;
    if(key_count == -1)
        key_count = 5;
    display();
    clear();
    keyscan();
    while(key_down)           //键盘消抖
    {
        display();
        clear();
        keyscan();
    }
}
void getkey()                 //得到按键值
{
    unsigned char value;
    unsigned char i, j = 0xFE;

    for(i=0; i<4; i++)

```



```
{
    C8255_A = j;
    value = C8255_C;
    if(!(value & 0x01))    //行 1
    {
        key_value = i + 0;
        writebuffer();
        return;
    }
    if(!(value & 0x02))    //行 2
    {
        key_value = i + 4;
        writebuffer();
        return;
    }
    if(!(value & 0x04))    //行 3
    {
        key_value = i + 8;
        writebuffer();
        return;
    }
    if(!(value & 0x08))    //行 4
    {
        key_value = i + 12;
        writebuffer();
        return;
    }
    j <= 1;
}
}
void main()
{
    C8255_CON = 0x81;        //8255 初始化
    key_count = 5;
    while(1)
    {
        display();           //显示
        clear();             //清屏
        keyscan();           //按键扫描
        if(key_down)         //判是否有键按下
        {
            display();
            delay(0x80);
            clear();
            keyscan();
            if(key_down)
            {
                getkey();     //得到按键值
            }
        }
    }
}
```

4.6 电子发声设计实验

4.6.1 实验目的

学习使用定时/计数器使扬声器发声的编程方法。

4.6.2 实验内容

根据实验提供的音乐频率表和时间表，编写程序控制单片机，使其输出连接到扬声器上能发出相应的乐曲。

4.6.3 实验说明及步骤

一个音符对应一个频率，将对应一个音符频率的方波通到扬声器上，就可以发出这个音符的声音。将一段乐曲的音符对应频率的方波依次送到扬声器，就可以演奏出这段乐曲。利用定时器控制单片机的 IO 引脚输出方波，将相应一种频率的计数初值写入计数器，就可产生对应频率的方波。计数初值的计算如下：

计数初值 = 输入时钟 ÷ 输出频率

例如输入时钟采用 1MHz，要得到 800Hz 的频率，计数初值即为 1000000÷800。音符与频率对照关系如表 4-6-1 所示。对于每一个音符的演奏时间，可以通过软件延时来处理。首先确定单位延时时间程序（根据 CPU 的频率不同而有所变化）。然后确定每个音符演奏需要几个单位时间，就几次调用延时子程序即可。

表 2-11-1 音符与频率对照表 (单位: Hz)

音符 音调	1	2	3	4	5	6	7
A	221	248	278	294	330	371	416
B	248	278	312	330	371	416	467
C	131	147	165	175	196	221	248
D	147	165	185	196	221	248	278
E	165	185	208	221	248	278	312
F	175	196	221	234	262	294	330
G	196	221	248	262	294	330	371

音符 音调	1	2	3	4	5	6	7
A	441	495	556	589	661	742	833
B	495	556	624	661	742	833	935
C	262	294	330	350	393	441	495
D	294	330	371	393	441	495	556
E	330	371	416	441	495	556	624
F	350	393	441	467	525	589	661
G	393	441	495	525	589	661	742

音符 音调	1	2	3	4	5	6	7
A	882	990	1112	1178	1322	1484	1665
B	990	1112	1248	1322	1484	1665	1869
C	525	589	661	700	786	882	990
D	589	661	742	786	882	990	1112
E	661	742	833	882	990	1112	1248
F	700	786	882	935	1049	1178	1322
G	786	882	990	1049	1178	1322	1484

下面提供了乐曲《友谊地久天长》实验参考程序。程序中频率表是将曲谱中的音符对应的频率值依次记录下来（B 调、四分之二拍），时间表是将各个音符发音的相对时间记录下来（由曲谱中节拍得出）。

频率表和时间表是一一对应的，频率表的最后一项为 0，作为重复的标志。根据频率表中的频率算出对应的计数初值，然后依次写入 T0 的计数器。将时间表中相对时间值带入延时程序来得到音符演奏时间。实验参考程序流程如图 4-6-1 所示。

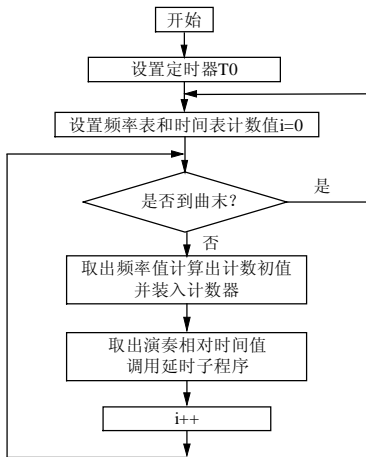


图 4-6-1 实验参考流程图

实验步骤如下：

1. 电子发声单元原理图如图 4-6-2 所示，按图 4-6-3 所示连接实验线路；
2. 编写实验程序，经编译、连接无误后启动调试；
3. 运行程序，听扬声器发出的音乐是否正确。

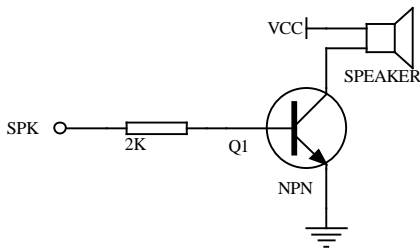


图 4-6-2 电子发声单元原理图

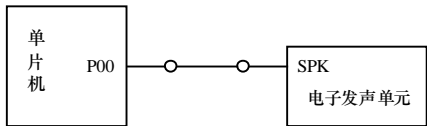


图 4-6-3 电子发声实验接线图

实验参考例程：(Sound.C)

```
#include "SST89x5x4.h"
sbit P00 = P0^0;           //扬声器控制引脚
#define Clk 0x070000
unsigned char data val_H;   //计数器高字节
unsigned char data val_L;   //计数器低字节
//频率表
```

```

unsigned int code freq_list[] = {371,495,495,495,624,556,495,556,624,
                                495,495,624,742,833,833,833,742,624,
                                624,495,556,495,556,624,495,416,416,
                                371,495,833,742,624,624,495,556,495,
                                556,833,742,624,624,742,833,990,742,
                                624,624,495,556,495,556,624,495,416,
                                416,371,495,0};

//时间表
unsigned char code time_list[] = {4, 6, 2, 4, 4, 6, 2, 4, 4, 6,
                                   2, 4, 4,12, 1, 3, 6, 2, 4, 4,
                                   6, 2, 4, 4, 6, 2, 4, 4,12, 4,
                                   6, 2, 4, 4, 6, 2, 4, 4, 6, 2,
                                   4, 4,12, 4, 6, 2, 4, 4, 6, 2,
                                   4, 4, 6, 2, 4, 4,12};

void t0_isr() interrupt 1           //定时器0 中断处理程序
{
    P00 = ~P00;                    //产生方波
    TH0 = val_H;                   //重新装入计数值
    TL0 = val_L;
}
void Delay(unsigned char cnt)       //单位延时
{
    unsigned char i;
    unsigned int j;
    for(i=0; i<cnt; i++)
    {
        for(j=0; j<0x3600; j++);
    }
}
void main(void)
{
    unsigned int val;
    unsigned char i;
    TMOD = 0x01;                   //初始化
    IE = 0x82;
    TR0 = 1;
    while(1)
    {
        i = 0;
        while(freq_list[i])        //频率为 0 重新开始
        {
            val = Clk/(freq_list[i]);
            val = 0xFFFF - val;     //计算计数值
            val_H = (val>>8)&0xff;
            val_L = val&0xff;
            TH0 = val_H;
            TL0 = val_L;
            Delay(time_list[i]);
            i++;
        }
    }
}

```

## 4.7 点阵 LED 显示设计实验

### 4.7.1 实验目的

1. 了解 LED 点阵的基本结构；
2. 学习 LED 点阵的扫描显示方法。

### 4.7.2 实验内容

编写程序，控制点阵的扫描显示，使  $16 \times 16$  LED 点阵循环显示汉字“西安唐都科教仪器公司”。

### 4.7.3 实验原理

实验系统中的  $16 \times 16$  LED 点阵由四块  $8 \times 8$  LED 点阵组成，如图 4-7-1 所示， $8 \times 8$  点阵内部结构图及外部引脚图如图 4-7-2 与图 4-7-3 所示。由图 4-7-2 可知，当行为“0”，列为“1”，则对应行、列上的 LED 点亮。汉字显示如图 4-7-4 所示。

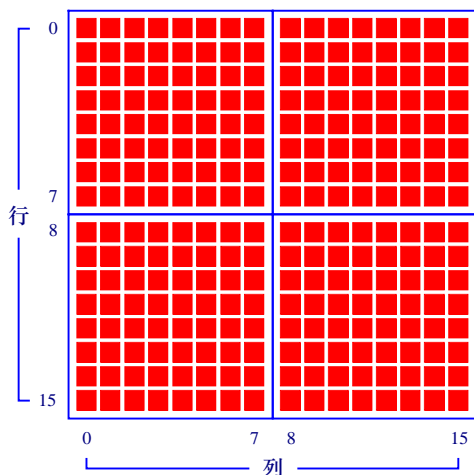


图 4-7-1  $16 \times 16$  点阵示意图

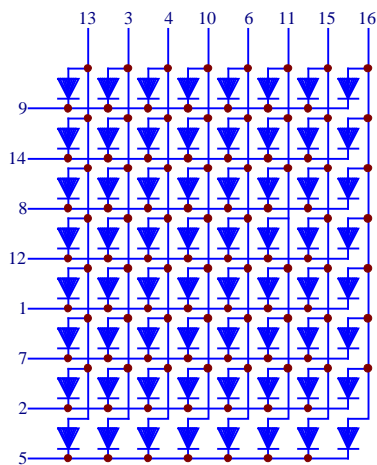


图 4-7-2 点阵内部结构图

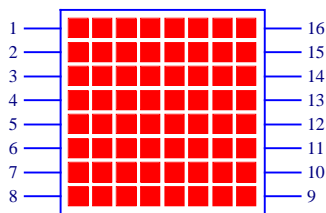


图 4-7-3 点阵外部引脚图

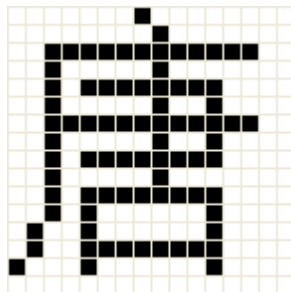


图 4-7-4 显示示例

点阵单元的原理图如图 4-7-5 所示，实验连线图如图 4-7-6 所示。

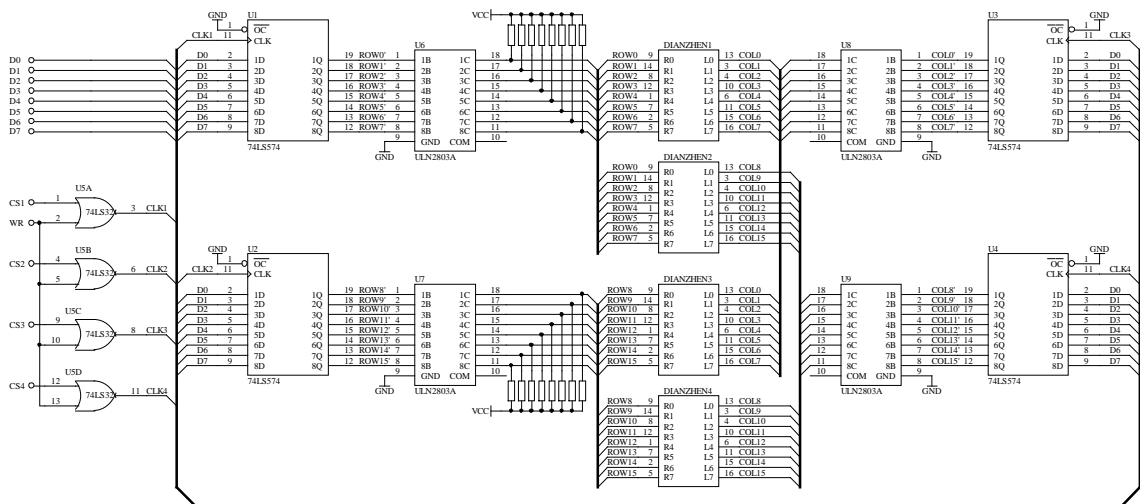


图 4-7-5 点阵实验单元原理图

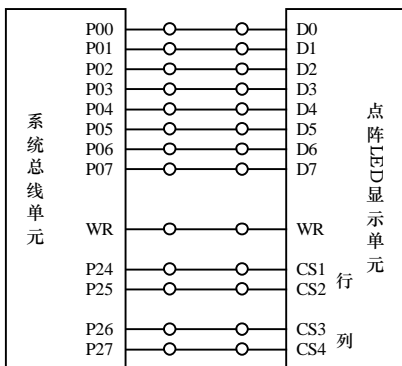


图 4-7-6 点阵显示实验接线图

### 4.7.3 实验步骤

1. 按图 4-7-6 连接实验电路图；
2. 编写实验程序，检查无误后，编译、链接并启动调试；
3. 运行实验程序，观察点阵的显示，验证程序功能；
4. 自己可以设计实验，使点阵显示不同的符号。

### 实验程序清单：(LED16.C)

```
#include "sst89x5x4.h"
#include "Absacc.h"
// #include "hzdot.h"
#include "hzdot1.h"

#define Row1 XBYTE[0xef00] // 端口定义
#define Row2 XBYTE[0xdf00]
#define Col1 XBYTE[0xbf00]
```

```

#define Col2 XBYTE[0x7f00]

void Delay(void)                // 延时子程序
{
    unsigned char i;
    for(i=0; i<60; i++);
}

void clear(void)                // 清屏
{
    Row1 = 0x00;
    Row2 = 0x00;
    Col1 = 0xff;
    Col2 = 0xff;
}

void main(void)                // 主程序
{
    unsigned char Scan=0x01, i, j;
    unsigned int count=0;
    clear();                    // 首先进行清屏
    while(1)
    {
        for(i=0; i<50; i++)
        {
            for(j=0; j<8; j++)
            {
                Row1 = 0x00;
                Col1 = ~hzdot[count];    // 0--7 列
                Col2 = ~hzdot[count+1];  // 8--15 列
                count+=2;
                Row1 = Scan;              // 0--7 行
                Scan = (Scan<<1)|(Scan>>7); // 行扫
                Delay();
            }
            Row1 = 0x00;
            for(j=0; j<8; j++)
            {
                Row2 = 0x00;
                Col1 = ~hzdot[count];
                Col2 = ~hzdot[count+1];
                count+=2;
                Row2 = Scan;              // 8--15 行
                Scan = (Scan<<1)|(Scan>>7);
                Delay();
            }
            Row2 = 0x00;
            count-=32;
        }
        count+=2;
        if(count == (320-32)) count=0;
    }
}

```

## 4.8 接触式 IC 卡读写实验

### 4.8.1 实验目的

1. 了解接触式 IC 卡的知识；
2. 掌握 IC 卡的通讯的协议（IIC 总线协议）及其的编程。

### 4.8.2 实验设备

PC 机一台，TD-NMC+实验装置一套。

### 4.8.3 实验内容

了解 IC 卡的通讯协议，编写实验程序，对 IC 卡进行读/写操作。

### 4.8.4 实验原理

接触式 IC 卡的触点定义遵循 ISO7816 规定，IC 卡 8 个触点分布如图所示。基于 Atmel 公司的 AT24C01 生产的 IC 卡就是一种简单易用的存储卡，其管脚排列如图 4-8-1 所示。

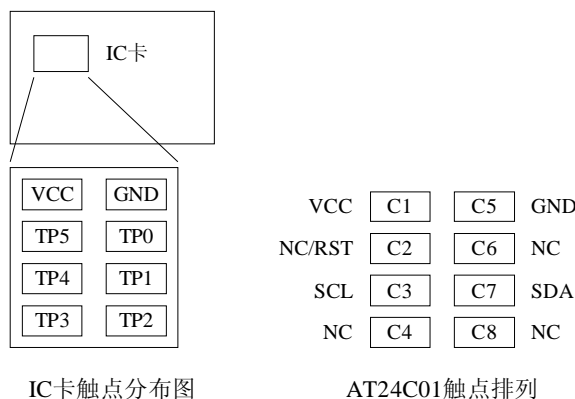


图 4-8-1 IC 卡触点分布图及 AT24C01 触点排列图

接触式 IC 卡与 CPU 采用 IIC 总线通讯形式，IIC 总线只用两条线，不需要片选线，支持带电插拔，SCL 是时钟线，SDA 是数据线，总线上的各节点都采用漏极开路结构与总线相连。在标准 IIC 模式下数据传输率可达 100Kb/s，高速模式下可达 400Kb/s。

IIC 总线上传送的每一个字节均为 8 位，并且高位在前。首先由起始信号启动 IIC 总线，其后为寻址字节，寻址字节由高 7 位地址和最低 1 位方向位组成，方向位表明主控器的操作方式，为读操作或写操作，其后的数据传输字节数是没有限制的。每传送一个字节后都必须跟随一个应答位或非应答位，在全部数据传送结束后主控制器发送终止信号。IIC 总线上每传输一个数据位，必须产生一个时钟脉冲。

SDA 线上的数据必须在时钟线 SCL 的高电平周期保持稳定，数据线的电平状态只有在 SCL 线的时钟信号是低电平时才能改变。在标准模式下，高低电平宽度必须不小于 4.7μs。



起始条件（重复起始条件）：当 SCL 线为高电平时，SDA 线从高电平向低电平切换。

停止条件：当 SCL 线为高电平时，SDA 线由低电平向高电平切换。具体如图 4-8-2 所示。

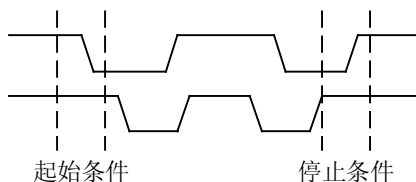


图 4-8-2 起始和停止条件

IC 卡实验单元原理图如图 4-8-3 所示，实验参考接线图如图 4-8-4 所示。

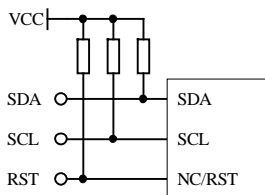


图 4-8-3 IC 卡单元原理图

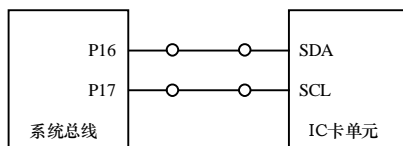


图 4-8-4 IC 卡实验接线图

### 4.8.5 实验步骤

1. 实验参考接线图如图 4-8-4 所示，按图接线；
2. 编写实验程序，编译、链接无误后启动调试；
3. 执行程序，首先向 IC 卡中先写入初始值；
4. 然后再继续运行程序，从 IC 卡中将写入的数据读出，并保存到 RAM 中；
5. 查看 RAM 中的数据，看与原始数据是否一致，验证程序功能；
6. 重复几组数据，进行程序功能测试。

### 实验程序清单

#### 主文件 (Icmain.C)

```
#include "Reg51.h"
#include "IIC_C51.c"
// 写入 IC 卡中的数据，用户可以修改
unsigned char code WData[]={0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80};

void Delay(unsigned int count)
{
    unsigned int i;
    for(i=0; i<count; i++);
}

void main()
{
    bit temp;
    unsigned char *ptr;
    ptr = (unsigned char *)0x30;    // 指向内部 RAM 30H 处
    Init_I2c();                    // 初始化 IIC 总线
```

```

    ISendStr(0xA0,0,WData,8);          // 将 WData 中的数据写入卡内
    Delay(300);
    temp = IRcvStr(0xA0,0,ptr,8);      // 从卡内读出数据并存入指定 RAM 中
    if(temp == 1)                      // 接收数据出错
    {
        *ptr = 0x55;
        ptr++;
        *ptr = 0xaa;
    }
    while(1);
}

```

## IIC 总线驱动文件 (IIC\_C51.C)

```

#include <Reg51.h>          /*头文件的包含*/
#include <intrins.h>
#define uchar unsigned char /*宏定义*/
#define uint unsigned int
#define _Nop() _nop()      /*定义空指令*/
/*端口位定义*/
sbit SDA=P1^6;             /*模拟 I2C 数据传送位*/
sbit SCL=P1^7;             /*模拟 I2C 时钟控制位*/
/*状态标志*/
bit ack;                   /*应答标志位*/

void Start_I2c()           //启动总线函数
{
    SDA=1;                 /*发送起始条件的数据信号*/
    _Nop();
    SCL=1;
    _Nop();                /*起始条件建立时间大于 4.7us,延时*/
    _Nop();                _Nop();    _Nop();    _Nop();
    SDA=0;                 /*发送起始信号*/
    _Nop();                /*起始条件锁定时间大于 4us*/
    _Nop();                _Nop();    _Nop();    _Nop();
    SCL=0;                 /*钳住 I2C 总线,准备发送或接收数据 */
    _Nop();                _Nop();
}

void Stop_I2c()            //结束总线函数
{
    SDA=0;                 /*发送结束条件的数据信号*/
    _Nop();                /*发送结束条件的时钟信号*/
    SCL=1;                 /*结束条件建立时间大于 4us*/
    _Nop();                _Nop();    _Nop();    _Nop();
    _Nop();
    SDA=1;                 /*发送 I2C 总线结束信号*/
    _Nop();                _Nop();    _Nop();    _Nop();
    _Nop();                _Nop();
}

void Init_I2c()            //初始化 IIC 总线
{
    SCL = 0;
    Stop_I2c();
}

```

```

}
void SendByte(uchar c)    // 字节数据传送函数
{
    uchar BitCnt;
    for(BitCnt=0;BitCnt<8;BitCnt++)    /*要传送的数据长度为 8 位*/
    {
        if((c<<BitCnt)&0x80)SDA=1;    /*判断发送位*/
        else SDA=0;
        _Nop();
        SCL=1;    /*置时钟线为高, 通知被控器开始接收数据位*/
        _Nop();
        _Nop();    /*保证时钟高电平周期大于 4μs*/
        _Nop();    _Nop();    _Nop();
        SCL=0;
    }

    _Nop();
    SDA=1;    /*8 位发送完后释放数据线, 准备接收应答位*/
    _Nop();
    SCL=1;
    _Nop();    _Nop();    _Nop();    _Nop();
    if(SDA==1)ack=0;
    else ack=1;    /*判断是否接收到应答信号*/
    SCL=0;
    _Nop();    _Nop();
}

uchar RcvByte()    // 字节数据接收函数
{
    uchar retc=0, BitCnt;
    SDA=1;    /*置数据线为输入方式*/
    for(BitCnt=0;BitCnt<8;BitCnt++)
    {
        _Nop();
        SCL=0;    /*置时钟线为低, 准备接收数据位*/
        _Nop();    /*时钟低电平周期大于 4.7μs*/
        _Nop();    _Nop();    _Nop();
        _Nop();
        SCL=1;    /*置时钟线为高使数据线上数据有效*/
        _Nop();    _Nop();
        retc=retc<<1;
        if(SDA==1) retc = retc+1; /*读数据位, 接收的数据位放入 retc 中 */
        _Nop();    _Nop();
    }
    SCL=0;
    _Nop();    _Nop();
    return(retc);
}

void Ack_I2c(bit a)    // 应答子函数
{
    if(a==0) SDA=0;    /*在此发出应答或非应答信号 */
    else SDA=1;
    _Nop();    _Nop();    _Nop();
    SCL=1;
    _Nop();    /*时钟低电平周期大于 4μs*/
    _Nop();    _Nop();    _Nop();
}

```

```

    _Nop();
    SCL=0;                /*清时钟线，钳住 I2C 总线以便继续接收*/
    _Nop();
    _Nop();
}
// 向有子地址器件发送多字节数据函数
bit ISendStr(uchar sla,uchar suba,uchar *s,uchar no)
{
    uchar i;
    Start_I2c();          /*启动总线*/
    SendByte(sla);        /*发送器件地址*/
    if(ack==0) return(1);
    SendByte(suba);       /*发送器件子地址*/
    if(ack==0) return(1);
    for(i=0;i<no;i++)
    {
        SendByte(*s);     /*发送数据*/
        if(ack==0) return(1);
        s++;
    }
    Stop_I2c();           /*结束总线*/
    return(0);
}
// 从有子地址器件读取多字节数据函数
bit IRcvStr(uchar sla,uchar suba,uchar *s,uchar no)
{
    uchar i;

    Start_I2c();          /*启动总线*/
    SendByte(sla);        /*发送器件地址*/
    if(ack==0) return(1);
    SendByte(suba);       /*发送器件子地址*/
    if(ack==0) return(1);

    Start_I2c();
    SendByte(sla+1);
    if(ack==0) return(1);
    for(i=0;i<no-1;i++)
    {
        *s=RcvByte();     /*发送数据*/
        Ack_I2c(0);       /*发送就答位*/
        s++;
    }
    *s=RcvByte();
    Ack_I2c(1);           /*发送非应位*/
    Stop_I2c();           /*结束总线*/
    return(0);
}

```

## 4.9 单总线数字温度传感器实验

### 4.9.1 实验目的

了解单总线控制的方法，学习数字温度测量的原理及其程序设计方法。

### 4.9.2 实验设备

PC 机一台，TD-NMC+实验装置一套，温度计一只。

### 4.9.3 实验内容

编写实验程序，测量环境温度，并将温度通过数码管显示出来。

### 4.9.4 实验原理

DS18B20 数字温度计提供了 9~12 位（可配置）温度数据，以指示器件的温度值。单片机与 DS18B20 之间的数据传递都通过单总线（1-Wire）接口进行。每一个 DS18B20 都有唯一的序列号以保证一根总线上同时存在多个器件。

DS18B20 有 3 个引脚，即 GND (1)、DQ (2) 和 VCC (3)，DQ 引脚为数据输入/输出引脚。DS18B20 由四个部分构成：64 位 ROM、温度传感器、非易失温度报警触发器 TH 和 TL、配置寄存器。

通过 1-Wire 端口访问 DS18B20 的协议如下：

- 初始化
- ROM 功能命令
- 存储器功能命令
- 数据处理

单总线上的所有处理都从初始化开始，初始化操作由一个复位脉冲（总线控制器发出）和一个或多个应答信号（从设备发出）组成。根据应答信号，总线控制器可以知道从设备存在并已准备好工作。当总线控制器检测到有从设备存在时，就发送一个 ROM 功能命令，共有 5 种 ROM 功能命令。各命令描述如下：

#### 读 ROM (33H)

此命令允许总线控制器读取 DS18B20 的 8 位厂商代码，唯一的 48 位序列号以及 8 位的 CRC 码。此命令仅用在总线上只有一个从设备时，若有多个设备，则会出现数据冲突。

#### 匹配 ROM (55H)

匹配 ROM 命令后紧跟 64 位 ROM 序列，这允许总线控制器寻址多分支总线上一个指定的 DS18B20，只有匹配了 64 位 ROM 序列的 DS18B20 才会响应接下来的存储器功能命令。未能匹配的从器件将等待复位脉冲。

#### 跳过 ROM (CCH)

这个命令对于单分支总线可以节省时间，允许总线控制器不提供 64 位 ROM 序列而直接访问存储器功能。

查询 ROM (F0H)

当系统初始建立时，通过此命令识别总线上的各器件。

报警查询 (ECH)

查询测量温度超越报警设置的器件。

DS18B20 的高速暂存器由便笺式 RAM 和非易失性电擦写 EERAM 组成,后者用于存储 TH、TL 及配置寄存器的值。数据先写入便笺式 RAM，经校验后再传给 EERAM。便笺式 RAM 占 9 个字节，包括温度信息（第 0、1 字节）、TH 和 TL 值（2、3 字节）、配置寄存器数据（4 字节）、CRC（第 8 字节）等，第 5、6、7 字节不用。如图 4-9-1 所示。

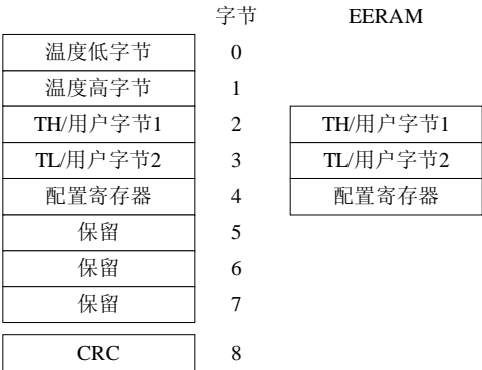


图 4-9-1 暂存器结构

其中配置寄存器的格式如下：

0	R1	R0	1	1	1	1	1
MSB			LSB				

R1、R0 决定了温度测量的分辨率，下表说明了这两位的组合关系：

R1	R0	温度分辨率	最大转换时间
0	0	9 位	93.75ms ( $t_{CONV}/8$ )
0	1	10 位	187.5ms ( $t_{CONV}/4$ )
1	0	11 位	375ms ( $t_{CONV}/2$ )
1	1	12 位	750ms ( $t_{CONV}$ )

温度按如下格式读取：（温度寄存器是只读的）

$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	LSB
MSb	单位：℃						LSb	
S	S	S	S	S	$2^6$	$2^5$	$2^4$	MSB

如：读出数据为 0550H，则温度值为 +85℃。

DS18B20 共有 6 个存储控制命令，如下：

写暂存器 (4EH)

从 TH 寄存器开始写入 3 个字节，在复位脉冲前必须写 3 个字节。

读暂存器 (BEH)

读出暂存器中的内容，共 9 个字节，从第 0 个字节开始。

### 复制暂存器 (48H)

此命令将暂存器中的内容复制到 EERAM 中。

### 温度转换 (44H)

这个命令开始温度转换。

### 读 EERAM (B8H)

此命令将温度设定值及配置寄存器中的值复制到暂存器中。

### 读电源供电方式 (B4H)

了解 DS18B20 的供电方式，0 为寄生供电，1 为外部电源供电。

单总线数字温度测量单元原理图如图 4-9-2 所示，实验接线图如图 4-9-3 所示。

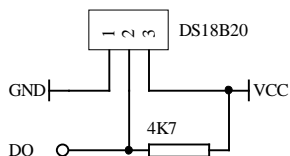


图 4-9-2 温度测量原理图

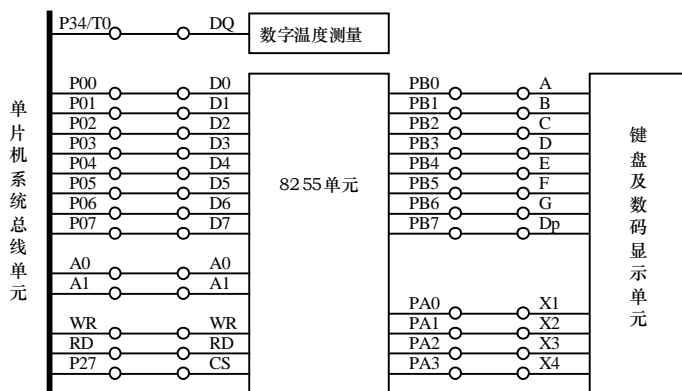


图 4-9-3 数字温度测量实验接线图

## 4.9.5 实验步骤

1. 按图 4-9-3 连接实验线路；
2. 编写实验程序，编译、链接无误后启动调试；
3. 运行实验程序，观察数码管的显示，比较与温度计的数据是否一致；
4. 用手触摸改变器件的表面温度，观察数码管的显示是否变化。

## 实验程序清单 (DS18B20.C)

```
#include <reg51.h>
#include <intrins.h>
#include <Absacc.h>
typedef unsigned char uchar;
typedef unsigned int uint;

#define C8255_A    XBYTE[0x7f00]
#define C8255_B    XBYTE[0x7f01]
#define C8255_CON  XBYTE[0x7f03]

sbit dq = P3^4;
bit flag;
uint Temperature;
uchar temp_buff[9]; //存储读取的字节，read scratchpad 为 9 字节
```

```

uchar id_buff[8]; //read rom ID为8字节
uchar *p;

unsigned char code Led[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char data Dispbuff[] = {0x00,0x00,0x00};

//延时处理
void TempDelay (uint us)
{
    while(us--);
}
//显示程序
void Display()
{
    uchar i, j=0xfe;
    C8255_B = 0x00;
    Dispbuff[0] = Temperature/100;           // 百位
    Dispbuff[1] = (Temperature%100)/10;       // 十位
    Dispbuff[2] = (Temperature%100)%10;       // 个位

    for(i=0; i<3; i++)
    {
        C8255_A = j;
        C8255_B = Led[Dispbuff[i]];
        TempDelay(60);
        j = (j<<1)|(j>>7);
    }
    C8255_B = 0x00;
}
//18B20 初始化
void Init18b20 (void)
{
    dq=1;
    _nop_();
    dq=0;
    TempDelay(86);           //delay 530 uS
    _nop_();
    dq=1;
    TempDelay(14);           //delay 100 uS
    _nop_();
    _nop_();
    _nop_();

    if(dq==0)
        flag = 1;           //detect 1820 success!
    else
        flag = 0;           //detect 1820 fail!
    TempDelay(20);           //20
    _nop_();
    _nop_();
    dq = 1;
}
//向 18B20 写入一个字节
void WriteByte (uchar wr)   //单字节写入
{
    uchar i;
    for (i=0;i<8;i++)

```



```

    {
        dq = 0;
        _nop_();
        dq=wr&0x01;
        TempDelay(5); //delay 45 uS //5
        _nop_();
        _nop_();
        dq=1;
        wr >>= 1;
    }
}

//读 18B20 的一个字节
uchar ReadByte (void) //读取单字节
{
    uchar i,u=0;
    for(i=0;i<8;i++)
    {
        dq = 0;
        u >>= 1;
        dq = 1;
        if(dq==1)
            u |= 0x80;
        TempDelay (4);
        _nop_();
    }
    return(u);
}

//读 18B20 多字节读
void read_bytes (uchar j)
{
    uchar i;
    for(i=0;i<j;i++)
    {
        *p = ReadByte();
        p++;
    }
}

//读取温度
void GemTemp (void)
{
    unsigned char templ;
    read_bytes (9); // 读取 scratchpad 中的值
    templ = (temp_buff[0]>>4)&0x0f; // 舍去小数点
    if((temp_buff[0]&0x08)==0x08) // 四舍五入
        templ += 1;
    templ = ((temp_buff[1]<<4)&0x70)|templ;
    Temperature = templ;
    TempDelay(1);
}

//内部配置
void Config18b20 (void) //重新配置报警限定值和分辨率
{
    Init18b20();
    WriteByte(0xcc); //skip rom
    WriteByte(0x4e); //写 scratchpad, 后跟 3 个字节数据
}

```

```

    WriteByte(0x1E);          //上限: 30 (TH)
    WriteByte(0x0A);          //下限: 10 (TL)
    WriteByte(0x7F);          //设置分辨率: 12 bit
    Init18b20();
    WriteByte(0xcc);          //skip rom
    WriteByte(0x48);          //保存设定值, 写 EERAM
    Init18b20();
    WriteByte(0xcc);          //skip rom
    WriteByte(0xb8);          //回调设定值, 读 EERAM
}
//读 18B20ID
void ReadID (void)           //读取器件 id
{
    Init18b20();
    WriteByte(0x33);          //read rom
    read_bytes(8);
}
//18B20 测温处理
void TemperatuerResult(void)
{
    Init18b20 ();
    WriteByte(0xcc);          //skip rom
    WriteByte(0x44);          //温度转换指令
    TempDelay(300);
    Init18b20 ();
    WriteByte(0xcc);          //skip rom
    WriteByte(0xbe);          //读取温度指令, 即读 scratchpad
    p = temp_buff;
    GemTemp();
}

void main(void)
{
    p = id_buff;
    ReadID();
    Config18b20();
    C8255_CON = 0x81;         // 初始化 8255
    Display();
    while(1)
    {
        TemperatuerResult();  // 测温
        Display();            // 显示
    }
}

```

## 4.10 字符型 LCD 显示设计实验

### 4.10.1 实验目的

了解字符型液晶的控制方法及程序设计方法。

### 4.10.2 实验设备

PC 机一台，TD-NMC+实验装置一套。

### 4.10.3 实验内容

编写实验程序，在液晶第一行上显示字符串“TD-NMC+ Xi'an Tang Du Crop.”。在液晶第二行上显示字符串“[www.tangdu.com](http://www.tangdu.com) 029-88375025”。并循环动态显示。

### 4.10.4 实验原理

字符型液晶显示模块是一类专门用于显示字母、数字及符号等的点阵型液晶显示模块。常用控制驱动器为 HD44780 及其兼容控制器。

#### 1. 液晶模块的接口信号及工作时序

液晶接口信号如下表所列：

引脚	符号	说明
1	VSS	地信号 (0V)
2	VDD	电源信号 (5V)
3	VO	LCD 驱动电压输入 (调节显示对比度)
4	RS	数据/指令寄存器选择。RS=0 为指令寄存器，RS=1 为数据寄存器。
5	R/W	读/写信号选择。R/W=1 为读信号，R/W=0 为写信号。
6	E	使能信号。读状态时高电平有效，写状态时下降沿有效。
7~14	DB0~DB7	数据总线
15	A	LED 背光阳极
16	K	LED 背光阴极

控制信号线 E、R/W 和 RS 的组合功能如下表所列：

RS	R/W	E	功能
0	0	下降沿	写指令代码
0	1	高电平	读 BF 和 AC 值
1	0	下降沿	写数据
1	1	高电平	读数据

总线操作时序图如图 4-10-1 和图 4-10-2 所示，时序参数如下表所示。

项目	符号	最小值	典型值	最大值	单位
使能周期时间	Tcyce	400	-	-	ns
使能脉冲宽度 (高电平)	Pweh	150	-	-	ns

使能上升/下降时间	Ter/Tef	-	-	25	ns
地址设置时间	Tas	30	-	-	ns
地址保持时间	Tah	10	-	-	ns
数据设置时间	Tdsw	40	-	-	ns
数据保持时间	Th	10	-	-	ns

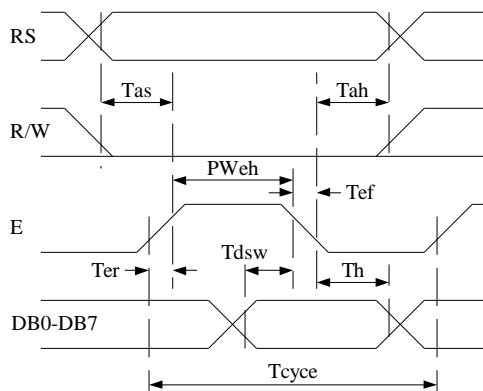


图 4-10-1 写操作时序图

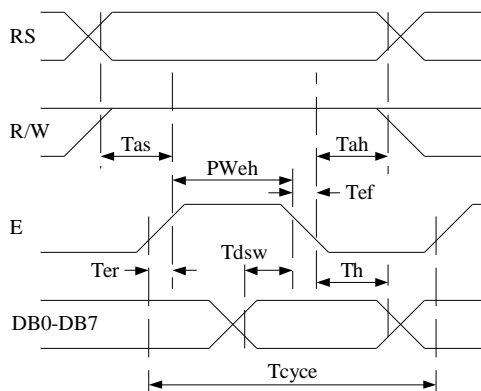


图 4-10-2 读操作时序图

## 2. 字符型液晶显示模块的软件特性

字符型液晶显示模块的软件特性就是 HD44780 的软件特性，HD44780 有 8 条指令，指令格式非常简单。指令一览表如下表所示。

指令名称	控制信号		控制代码							
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
清屏	0	0	0	0	0	0	0	0	0	1
归 HOME 位	0	0	0	0	0	0	0	0	1	*
输入方式设置	0	0	0	0	0	0	0	1	I/D	S
显示状态设置	0	0	0	0	0	0	1	D	C	B
光标画面滚动	0	0	0	0	0	1	S/C	R/L	*	*
工作方式设置	0	0	0	0	1	DL	N	F	*	*
CGRAM 地址设置	0	0	0	1	A5	A4	A3	A2	A1	A0
DDRAM 地址设置	0	0	1	A6	A5	A4	A3	A2	A1	A0
读 BF 和 AC 值	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0
写数据	1	0	数 据							
读数据	1	1	数 据							

注：“\*”表示任意值，在实际应用中一般认为是“0”。

指令详细解释如下：

### ● 清屏（Clear Display）

格式     

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

     代码：01H

该指令完成下列功能：

将空码（20H）写入 DDRAM 的全部 80 个单元内，将地址指针计数器 AC 清零，光标或闪

烁归 HOME 位，设置输入方式参数 I/D=1，即地址指针 AC 为自动加一输入方式。

该指令多用于上电时或更新全屏显示内容时。在使用该指令前要确认 DDRAM 的当前内容是否有用。

### ● 归 HOME 位

格式 

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

 代码：02H

该指令将地址指针计数器 AC 清零。执行该指令的效果有：将光标或闪烁位返回到显示屏的左上第一字符位上，即 DDRAM 地址 00H 单元位置，这是因为光标和闪烁位都是以地址指针计数器 AC 当前值定位的。如果画面已滚动，则撤消滚动效果，将画面拉回到 HOME 位。

### ● 输入方式设置 (Enter Mode Set)

格式 

0	0	0	0	0	1	I/D	S
---	---	---	---	---	---	-----	---

 代码：04H~07H

该指令的功能在于设置了显示字符的输入方式，即在计算机读/写 DDRAM 或 CGRAM 后，地址指针计数器 AC 的修改方式，反映在显示效果上，当写入一个字符后画面或光标的移动。该指令的两个参数位 I/D 和 S 确定了字符的输入方式。

I/D 表示当计算机读/写 DDRAM 或 CGRAM 的数据后，地址指针计数器 AC 的修改方式，由于光标位置也是由 AC 值确定，所以也是光标移动的方式。

I/D=0：AC 为减一计数器，光标左移一个字符位。

I/D=1：AC 为加一计数器，光标右移一个字符位。

S 表示在写入字符时，是否允许显示画面的滚动

S=0：禁止滚动

S=1：允许滚动

S=1 且 I/D=0：显示画面向右滚动一个字符

S=1 且 I/D=1：显示画面向左滚动一个字符

综合而论，该指令可以实现四种字符的输入方式，见表所示。

输入方式	指令代码	参数状态	示例
画面不动光标左移	04H	I/D=S=0	$\overline{\_t}$ _left
画面右滚动	05H	I/D=0, S=1	$\overline{\_t}$ _Shift
画面不动光标右移	06H	I/D=1, S=0	$\overline{R\_}$ Right_
画面左滚动	07H	I/D=S=1	$\overline{S\_}$ Shift_

注：“\_”为光标符号。

注意画面滚动方式在计算机读 DDRAM 数据时，或在读/写 CGRAM 时无效，也就是说该指令主要应用在计算机写入 DDRAM 数据的操作时，所以称该指令为输入方式设置指令。需要说明的是在计算机读 DDRAM 数据或在读/写 CGRAM 数据时，建议将 S 置零。

● 显示状态设置 (Display on/off Control)

格式	0	0	0	0	1	D	C	B	代码: 08H~0FH
----	---	---	---	---	---	---	---	---	-------------

该指令控制着画面, 光标及闪烁的开与关。该指令有三个状态位 D、C、B, 这三个状态位分别控制着画面, 光标和闪烁的显示状态。

D: 画面显示状态位。当 D=1 时为开显示, 当 D=0 时为关显示。主要关显示仅是画面不出现, 而 DDRAM 内容不变。这与清屏指令截然不同。

C: 光标显示状态位。当 C=1 时为光标显示, 当 C=0 时为光标消失。光标为底线形式 (5×1 点阵), 出现在第八行或第十一行上。光标的位置由地址指针计数器 AC 确定, 并随其变动而移动。当 AC 值超出了画面的显示范围, 光标将随之消失。

B: 闪烁显示状态位。当 B=1 时为闪烁启用, 当 B=0 时为闪烁禁止。闪烁是指一个字符位交替进行正常显示态和全亮显示态。闪烁频率在控制器工作频率为 250KHz 时为 2.4Hz。闪烁位置同光标一样受地址指针计数器 AC 的控制。

闪烁出现在有字符或光标显示的字符位时, 正常显示态为当前字符或光标的显示; 全亮显示态为该字符位所有点全显示。若出现在无字符或光标显示的字符位时, 正常显示态为无显示。全亮显示态为该字符位所有点全显示。这种闪烁方式可以设计成块光标, 如同计算机 CRT 上块状光标闪烁提示符的效果。

该指令代码表, 见下表所示。

指令代码	状态位			功能
	D	C	B	
08H~0BH	0	*	*	关显示
0CH	1	0	0	画面显示
0DH	1	0	1	画面, 闪烁显示
0EH	1	1	0	画面, 光标显示
0FH	1	1	1	画面, 光标, 闪烁显示

● 光标或画面滚动 (Cursor Or Display Shift)

格式	0	0	0	1	S/C	R/L	0	0
----	---	---	---	---	-----	-----	---	---

执行该指令将产生画面或光标向左或向右滚动一个字符位。如果定时间隔地执行该指令将产生画面或光标地平滑滚动。画面的滚动是在一行内连续循环进行的, 也就是说一行的第一单元与最后一个单元连接起来, 形成了闭环式的滚动。

该指令有两个参数位:

S/C: 滚动对象选择。S/C=1: 画面滚动; S/C=0: 光标滚动。

R/L: 滚动方向选择。R/L=1: 向右滚动; R/L=0: 向左滚动。

该指令代码表如下表所示:

指令代码	状态位		功能
	S/C	R/L	
10H	0	0	光标左滚动
14H	0	1	光标右滚动
18H	1	0	画面左滚动

ICH	1	1	画面右滚动
-----	---	---	-------

该指令与输入方式设置指令都可以产生光标或画面的滚动，区别在于该指令专用于滚动功能，执行一次，显示呈现一次滚动效果；而输入方式设置指令仅是完成了一种字符输入方式的设置，仅在计算机对 DDRAM 等进行操作时才能产生滚动的效果。

### ● 工作方式设置 (Function Set)

格式	0	0	1	DL	N	F	0	0
----	---	---	---	----	---	---	---	---

该指令设置了控制器的工作方式，包括有控制器与计算机的接口形式和控制器显示驱动的占空比系数等。该指令有三个参数 DL，N 和 F。它们的作用是：

DL：设置控制器与计算机的接口形式。接口形式体现在数据总线长度上。

DL=1：设置数据总线为 8 位长度，即 DB7~DB0 有效

DL=0：设置数据总线为 4 位长度，即 DB7~DB4 有效。该方式下 8 位指令代码和数据将按先高 4 位后低 4 位的顺序分两次传输。

N：设置显示的字符行数。N=0 为一行字符行；N=1 为两行字符行。

F：设置显示字符的字体。F=0 为 5×7 点阵字符体；F=1 为 5×10 点阵字符体。

N 和 F 的组合设置了控制器的确定占空比系数。见下表：

N	F	字符行数	字符体形式	占空比系数	备注
0	0	1	5×7	1/8	
0	1	1	5×10	1/11	
1	0	2	5×7	1/16	仅 5×7 字体

该指令可以说是字符型液晶显示控制器的初始化设置指令，也是唯一的软件复位指令。HD44780U 虽然具有复位电路，但为了可靠的工作，HD44780U 要求计算机在操作 HD44780U 时首先对其进行软件复位。也就是说在控制字符型液晶显示模块工作时首先要进行的软件复位。

### ● CGRAM 地址设置 (Set CGRAM Address)

格式	0	1	A5	A4	A3	A2	A1	A0
----	---	---	----	----	----	----	----	----

该指令将 6 位的 CGRAM 地址写入地址指针计数器 AC 内，随后计算机对数据的操作是对 CGRAM 的读/写操作。

### ● DDRAM 地址设置 (Set DDRAM Address)

格式	1	A6	A5	A4	A3	A2	A1	A0
----	---	----	----	----	----	----	----	----

该指令将 7 位 DDRAM 地址写入地址指针计数器 AC 内，随后计算机对数据的操作是对 DDRAM 的读/写操作。

### ● 读“忙”标志和地址指针值 (Read Busy Flag and Address)

格式	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0
----	----	-----	-----	-----	-----	-----	-----	-----

计算机对指令寄存器通道读操作 (RS=0, R/W=1) 时，将读出此格式的“忙”标志 BF 值和 7 位地址指针计数器 AC 的当前值。计算机随时都可以对 HD44780U 读“忙”操作。

BF 值反映 HD44780U 的接口状态。计算机在对 HD44780U 每次操作时首先都要读 BF 值判

断 HD44780U 的当前接口状态, 仅有在  $BF=0$  时计算机才可以向 HD44780U 写指令代码或显示数据和从 HD44780U 读出显示数据。

计算机读出的地址指针计数器 AC 当前值可能是 DDRAM 地址也可能是 CGRAM 的地址, 这取决于最近一次计算机向 AC 写入的是哪类地址。

### ● 写数据 (Write Data to CG or DDRAM)

计算机向数据寄存器通道写入数据, HD44780U 根据当前地址指针计数器 AC 值的属性及数值将该数据送入相应的存储器内的 AC 所指的单元里。如果 AC 值为 DDRAM 地址指针, 则认为写入的数据为字符代码并送入 DDRAM 内 AC 所指的单元里; 如果 AC 值为 CGRAM 的地址指针, 则认为写入的数据是自定义字符的字模数据并送入 CGRAM 内 AC 所指的单元里。所以计算机在写数据操作之前要先设置地址指针或人为的确认地址指针的属性及数值。

### ● 读数据 (Read Data from CG or DDRAM)

在 HD44780U 内部运行时序的操作下, 地址指针计数器 AC 的每一次修改, 包括新的 AC 值的写入, 光标滚动位移所引起的 AC 值的修改或由计算机读写数据操作后所产生的 AC 值的修改, HD44780U 都会把当前 AC 所指单元的内容送到接口部数据输出寄存器内, 供计算机读取。如果 AC 值为 DDRAM 地址指针, 则认为接口部数据输出寄存器的数据为 DDRAM 内 AC 所指单元的字符代码; 如果 AC 值为 CGRAM 的地址指针, 则认为数据输出寄存器的数据是 CGRAM 内 AC 所指单元的自定义字符的字模数据。

计算机的读数据是从数据寄存器通道中数据输出寄存器读取当前所存放的数据。所以计算机在首次读数据操作之前需要重新设置一次地址指针 AC 值, 或用光标滚动指令将地址指针计数器 AC 值修改到所需的地址上, 然后进行的读数据操作将能获得所需的数据。在读取数据后地址指针计数器 AC 将根据最近设置的输入方式自动修改。

以 8 位总线模式为例, 软件复位流程如图 4-10-3 所示。8 位数据总线操作流程如图 4-10-4 所示。

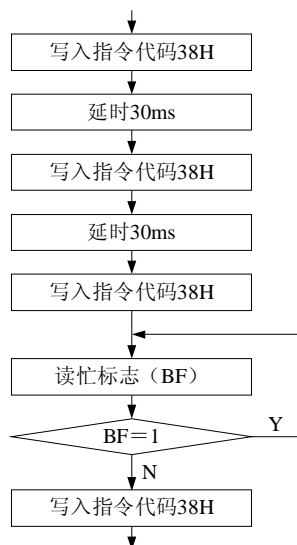


图 4-10-3 软件复位流程图

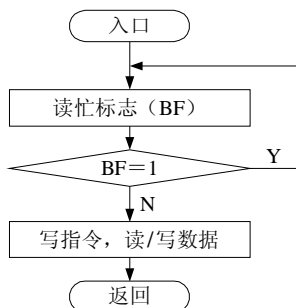


图 4-10-4 8 位数据总线操作流程



LCD 液晶显示单元原理图如图 4-10-5 所示，实验接线图如图 4-10-6 所示。

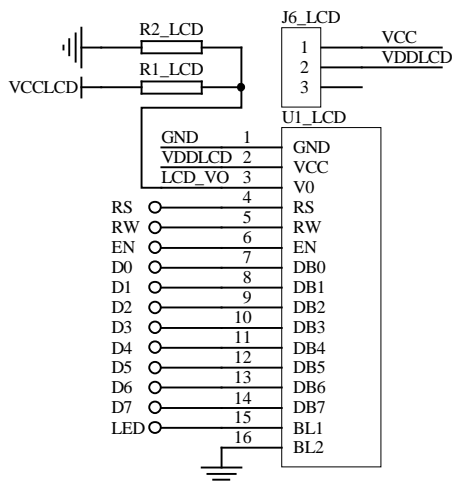


图 4-10-5 液晶单元原理图

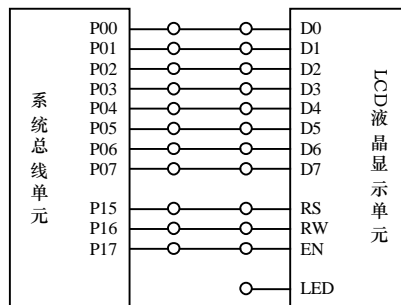


图 4-10-6 LCD 液晶显示实验接线图

### 4.10.5 实验步骤

1. 按图 4-10-6 连接实验线路图；
2. 编写实验程序，编译、链接无误后启动调试；
3. 运行实验程序，观察液晶的显示结果。

### 实验程序清单 (LCD1602)

```
#include<reg51.h>
#include<intrins.h>
//引脚定义
sbit RSPIN = P1^5;
sbit RWPIN = P1^6;
sbit EPIN = P1^7;

unsigned char XPOS,YPOS;
unsigned char DisTab1[] = "TD-NMC+ Xi'an Tang Du Crop.      ";
unsigned char DisTab2[] = "www.tangdu.com 029-88375025  ";

void delay(unsigned int t)
{
    unsigned int i,j;
    for(i=0;i<t;i++)
        for(j=0;j<10;j++);
}

void lcdwaitidle(void)          //忙状态判别
{
    P0=0xff;    RSPIN=0;    RWPIN=1;    EPIN=1;
    while((P0&0x80)==0x80);    //读取忙标志 BF, 判为 1 否, 为 1 等待
    EPIN=0;
}

void lcdwcn(unsigned char c)    //写指令 c
{
    RSPIN=0;    RWPIN=0;    P0=c;    EPIN=1;
    _nop_();    EPIN=0;
}
```

```

void lcdwc(unsigned char c)           //查询忙标志, 然后写指令 c
{
    lcdwaitidle();
    lcdwcn(c);
}
void lcdwd(unsigned char d)           //查询忙标志, 然后写数据 d
{
    lcdwaitidle();
    RSPIN=1;    RWPIN=0;    P0=d;    EPIN=1;
    _nop_();    EPIN=0;
}
void lcdpos(void)
{
    XPOS&=0x3f;    YPOS&=0x03;
    if(YPOS==0x00)
        lcdwc(XPOS|0x80);           //DDRAM 地址设置(第 1 行)
    else if(YPOS==0x01)
        lcdwc((XPOS+0x40)|0x80);    //DDRAM 地址设置(第 2 行)
}
void lcdinit(void)                   //LCD 初始化
{
    delay(150); lcdwcn(0x38);        //总线 8 位, 两行显示, 5*7 点阵字符体
    delay(50); lcdwcn(0x38);
    delay(50); lcdwcn(0x38);
    lcdwc(0x38);
    lcdwc(0x08);                     //关闭显示, 光标消失, 闪烁禁止
    lcdwc(0x01);                     //清屏
    lcdwc(0x06);                     //AC 加 1 计数, 禁止滚动
    lcdwc(0x0e);                     //开显示
}
void Display(void)                   // 显示子程序
{
    for(XPOS=0; XPOS<16; XPOS++)
    {
        YPOS=0; lcdpos(); lcdwd(DisTab1[XPOS]);
        YPOS=1; lcdpos(); lcdwd(DisTab2[XPOS]);
        delay(2000);
    }
    for(XPOS=16; XPOS<30; XPOS++)
    {
        lcdwc(0x18);                // 滚屏
        YPOS=0; lcdpos(); lcdwd(DisTab1[XPOS]);
        YPOS=1; lcdpos(); lcdwd(DisTab2[XPOS]);
        delay(2000);
    }
}
void main(void)
{
    EPIN=0;
    lcdinit();
    while(1)
    {
        lcdwc(0x01);
        lcdwc(0x02);
        Display();
        delay(5000);
    }
}

```

## 4.11 图形 LCD 显示设计实验（选配）

### 4.11.1 实验目的

了解图形 LCD 的控制方法。

### 4.11.2 实验内容

本实验使用的是  $128 \times 64$  图形点阵液晶，编写实验程序，通过单片机控制液晶，显示“唐都科教仪器公司欢迎你！”，并使该字串滚屏一周。

### 4.11.3 实验原理

#### 1. 液晶模块的接口信号及工作时序

该图形液晶内置有控制器，这使得液晶显示模块的硬件电路简单化，它与 CPU 连接的信号线如下：

CS1、CS2：片选信号，低电平有效；

E：使能信号；

RS：数据和指令选择信号，RS=1 为 RAM 数据，RS=0 为指令数据；

R/W：读/写信号，R/W=1 为读操作，R/W=0 为写操作；

D7~D0：数据总线；

LT：背景灯控制信号，LT=1 时打开背景灯，LT=0 时关闭背景灯。

该液晶的时序参数说明如表 4-11-1 所列，读写时序图如图 4-11-1 和 4-11-2 所示。

表 4-11-1 时序参数说明

特性曲线	助记符	最小值	典型	最大值	单位
E 周期	tcyc	1000	-	-	ns
E 高电平宽度	twhE	450	-	-	ns
E 低电平宽度	twlE	450	-	-	ns
E 上升时间	tr	-	-	25	ns
E 下降时间	tf	-	-	25	ns
地址建立时间	tas	140	-	-	ns
地址保持时间	tah	10	-	-	ns
数据建立时间	tdsw	200	-	-	ns
数据延迟时间	tddr	-	-	320	ns
数据保持时间（写）	tdhw	10	-	-	ns
数据保持时间（读）	tdhr	20	-	-	ns

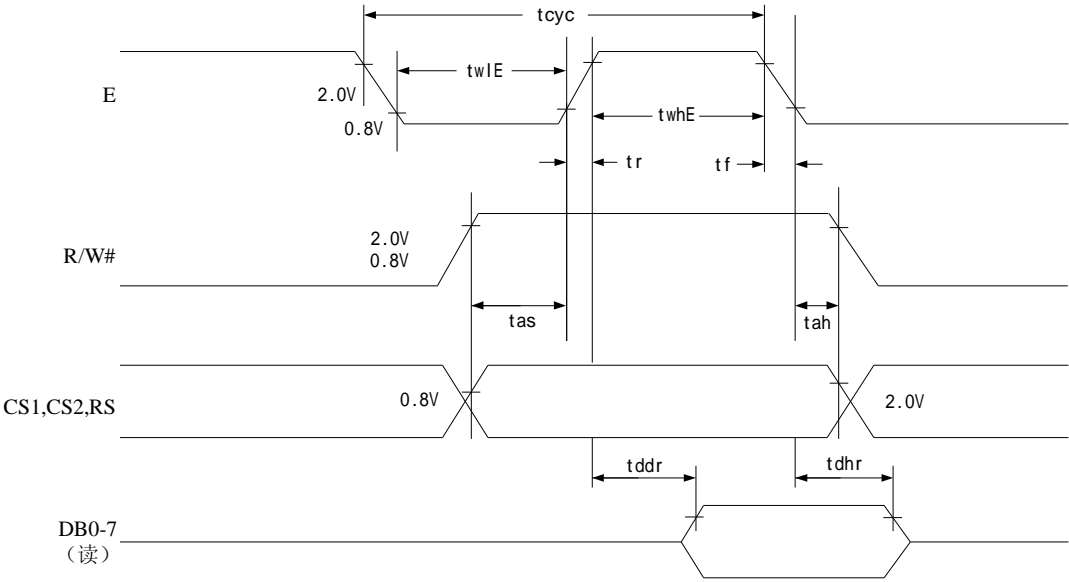


图 4-11-1 读操作时序图

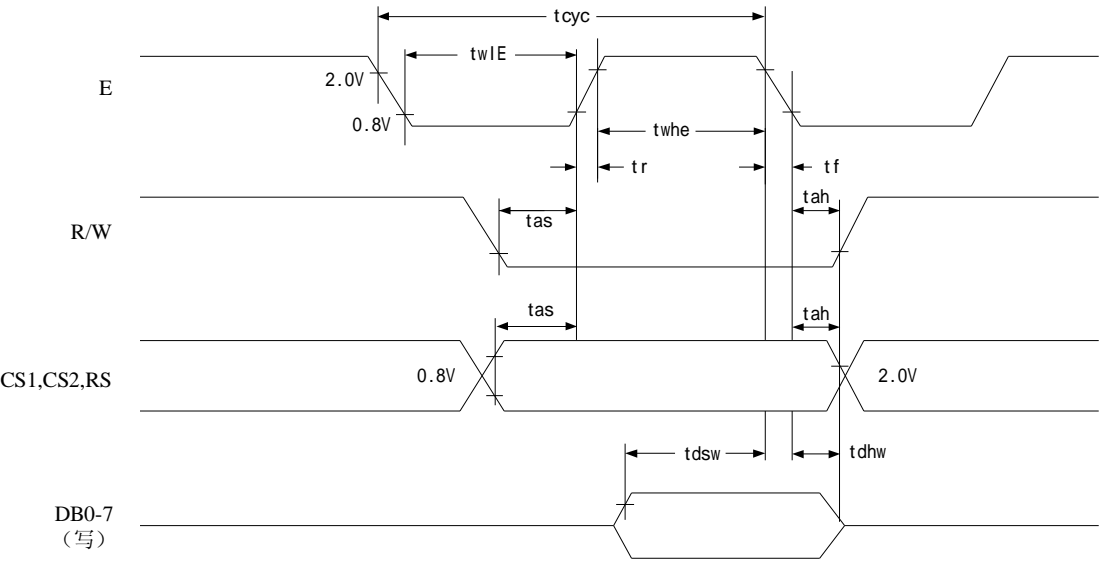


图 4-11-2 写操作时序图

2. 显示控制指令

显示控制指令控制着液晶控制器的内部状态，具体如表 4-11-2 所列。

表 4-11-2 显示控制命令列表

指令	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
显示 开/关	0	0	0	0	1	1	1	1	1	0/1
设置地址 (Y 地址)	0	0	0	1	Y 地址 (0~63)					
设置页 (X 地址)	0	0	1	0	1	1	1	页 (0~7)		
显示起始行 (Z 地址)	0	0	1	1	显示起始行 (0~63)					
状态读	0	1	忙	0	开/关	复位	0	0	0	0
写显示数据	1	0	写数据							
读显示数据	1	1	读数据							

显示开/关:

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	0	0	1	1	1	1	1	D

该指令设置显示开/关触发器的状态, 当 D=1 为显示数据, 当 D=0 为关闭显示设置。

设置地址 (Y 地址):

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

该指令用以设置 Y 地址计数器的内容, AC5~AC0=0~63 代表某一页面上的某一单元地址, 随后的一次读或写数据将在这个单元上进行。Y 地址计数器具有自动加一功能, 在每次读或写数据后它将自动加一, 所以在连续读写数据时, Y 地址计数器不必每次设置一次。

设置页 (X 地址):

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	1	0	1	1	1	AC2	AC1	AC0

该指令设置页面地址寄存器的内容。显示存储器共分 8 页, 指令代码中 AC2~AC0 用于确定当前所要选择的页面地址, 取值范围为 0~7, 代表第 1~8 页。该指令指出以后的读写操作将在哪一个页面上进行。

显示起始行 (Z 地址):

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	1	1	L5	L4	L3	L2	L1	L0

该指令设置了显示起始行寄存器的内容。此液晶共有 64 行显示的管理能力, 指令中的 L5~L0 为显示起始行的地址, 取值为 0~63, 规定了显示屏上最顶一行所对应的显示存储器的行地址。若等时间、等间距地修改显示起始行寄存器的内容, 则显示屏将呈现显示内容向上或向下

滚动的显示效果。

### 状态读：

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	1	忙	0	开/关	复位	0	0	0	0

状态字是 CPU 了解液晶当前状态的唯一信息渠道。共有 3 位有效位，说明如下。

忙：表示当前液晶接口控制电路运行状态。当忙位为 1 表示正在处理指令或数据，此时接口电路被封锁，不能接受除读状态字以外的任何操作。当忙位为 0 时，表明接口控制电路已准备好等待 CPU 的访问。

开/关：表示当前的显示状态。为 1 表示关显示状态，为 0 表示开显示状态。

复位：为 1 表示系统正处于复位状态，此时除状态读可被执行外，其它指令不可执行，此位为 0 表示处于正常工作状态。

在指令设置和数据读写时要注意状态字中的忙标志。只有在忙标志为 0 时，对液晶的操作才能有效。所以在每次对液晶操作前，都要读出状态字判断忙标志位，若不为 0 则需要等待，直到忙标志为 0 为止。

### 写显示数据：

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	1	0	D7	D6	D5	D4	D3	D2	D1	D0

该操作将 8 位数据写入先前确定的显示存储单元中。操作完成后列地址计数器自动加一。

### 读显示数据：

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	1	1	D7	D6	D5	D4	D3	D2	D1	D0

该操作将读出显示数据 RAM 中的数据，然后列地址计数器自动加一。

#### 4.11.4 实验步骤

1. 按照图 4-11-3 连接实验接线图；
2. 得到需显示汉字或图形的显示数据，这里需要得到“唐都科教仪器公司欢迎你！”的字模；
3. 编写实验程序，编译、链接无误后进入调试界面；
4. 运行实验程序，观察 LCD 显示，验证程序功能。

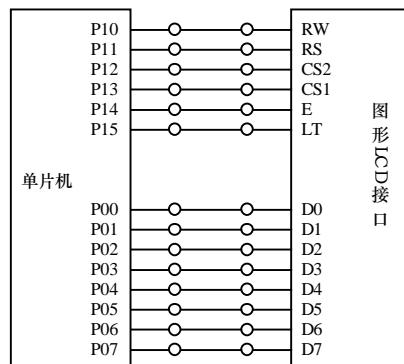


图 4-11-3 液晶实验线路图

## 实验程序清单：(CLCD.C)

```

#include "SST89x5x4.h"
#include "lcd.h"

sbit RW = P1^0;           //读写控制
sbit RS = P1^1;           //数据, 命令切换
sbit CS2 = P1^2;          //控制器 2 片选
sbit CS1 = P1^3;          //控制器 1 片选
sbit E = P1^4;            //使能信号
sbit LT = P1^5;           //背景灯控制

void delay(char time);
void Query(char chip);     //状态查询
void Clear();              //清屏
void writehz(char xadd,char yadd,char chip,char* hz);

char Wdata;
char xadd;                 //X 地址
char yadd;                 //Y 地址

void SendMCmd(unsigned char cmd) //向控制器 1 发送命令
{
    Query(1); delay(1);         //查询忙状态
    RS = RW = CS1 = 0;         //控制写入命令
    P0 = cmd;
    E = 0; E = 1; E = 0; CS1 = 1;
}

void SendSCmd(unsigned char cmd) //向控制器 2 发送命令
{
    Query(2); delay(1);
    RS = RW = CS2 = 0;
    P0 = cmd;
    E = 0; E = 1; E = 0; CS2 = 1;
}

void SendMData(unsigned char dat) //向控制器 1 写数据
{
    Query(1); delay(1);
    RS = 1; RW = 0; CS1 = 0;
    P0 = dat;
    E = 0; E=1; E=0; CS1 = 1;
}

void SendSData(unsigned char dat) //向控制器 2 写数据
{
    Query(2); delay(1);
    RS = 1; RW = 0; CS2 = 0;
    P0 = dat;
    E=0; E=1; E=0; CS2 = 1;
}

void Query(char chip) //状态查询,chip 指出控制器号
{
    //chip=1 为控制器 1
    unsigned char xdata statu, count = 100;
    RS = 0; RW = 1;

```

```

    if(chip == 1) CS1 = 0;
    else CS2 = 0;
    do
    {
        E = 1; statu = P0; E = 0;
        if((statu&0x80)!=0x80) break;
    }while(count--);
    if(chip == 1) CS1 = 1;
    else CS2 = 1;
}
void Clear() //清除显示屏
{
    unsigned char i, j;
    for(i=0; i<8; i++)
    {
        xadd = 0xb8+i; yadd = 0x40;
        SendMCmd(xadd); SendSCmd(xadd); //设置 X,Y 地址
        SendMCmd(yadd); SendSCmd(yadd);
        for(j=0; j<64; j++)
        {
            SendMData(0x00); SendSData(0x00); //为所以 RAM 写 00
        }
    }
}
void writehz(char xadd,char yadd,char chip,char* hz)
{ //写汉字子程序 16*16
    int x,y;
    if(chip==0x01) //为控制器 1
    {
        SendMCmd(xadd); SendMCmd(yadd);
        for(x=0;x<2;x++) //写入汉字字模
        {
            for(y=0;y<16;y++)
            {
                Wdata = hz[y+(x*16)];
                SendMData(Wdata);
            }
            xadd++;
            SendMCmd(xadd); SendMCmd(yadd);
        }
    }
    else //为控制器 2
    {
        SendSCmd(xadd); SendSCmd(yadd);
        for(x=0;x<2;x++) //写入汉字字模
        {
            for(y=0;y<16;y++)
            {
                Wdata = hz[y+(x*16)];
                SendSData(Wdata);
            }
            xadd++;
            SendSCmd(xadd); SendSCmd(yadd);
        }
    }
}
void main(void)

```



```
{
    unsigned char x; unsigned int y;
    LT = 0; //关闭背景灯
    SendMCmd(0x3f); SendSCmd(0x3f); //打开显示
    SendMCmd(0xc0); SendSCmd(0xc0); //设置起始行
    Clear();
    writehz(0xba,0x40,1,tang); //显示"唐"
    writehz(0xba,0x50,1,du); //都
    writehz(0xba,0x60,1,ke); //科
    writehz(0xba,0x70,1,jiao); //教
    writehz(0xba,0x40,2,yi); //仪
    writehz(0xba,0x50,2,qi); //器
    writehz(0xba,0x60,2,gong); //公
    writehz(0xba,0x70,2,si); //司
    writehz(0xbc,0x60,1,huan); //欢
    writehz(0xbc,0x70,1,ying); //迎
    writehz(0xbc,0x40,2,nin); //您
    writehz(0xbc,0x50,2,gantan); //!

    for(x=0xc1;x<0xFF;x++) //滚屏一周
    {
        SendMCmd(x); SendSCmd(x);
        for(y=0; y<0x1000; y++) delay(0xff);
    }
    while(1);
}

void delay(char time)
{
    char i;
    char j;
    for(i=0;i<=time;i++)
    {
        for(j=0;j<=0x10;j++);
    }
}
```

## 第 5 章 单片机控制应用实验

单片机在控制方面也有广泛的应用，这里以三个理解介绍单片机在控制系统中的应用，步进电机实验视选择的接口平台而言，若接口平台未配步进电机，则需要另购。

### 5.1 步进电机实验

#### 5.1.1 实验目的

了解单片机控制步进电机的方法。

#### 5.1.2 实验原理

使用开环控制方式能对步进电机的转到方向、速度和角度进行调节。所谓步进，就是指每给步进电机一个递进脉冲，步进电机各绕组的通电顺序就改变一次，即电机转动一次。根据步进电机控制绕组的多少可以将电机分为三相、四相和五相。实验中所使用的步进电机为四相八拍电机，电压为 DC5V，其励磁线圈及其励磁顺序如图 5-1-1 及表 5-1-1 所示。

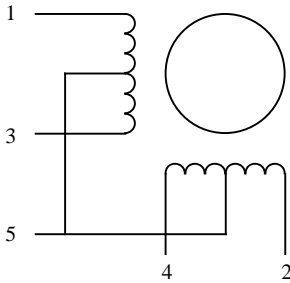


图 5-1-1 励磁线圈

表 5-1-1 励磁顺序

	1	2	3	4	5	6	7	8
5	+	+	+	+	+	+	+	+
4	-	-						-
3		-	-	-				
2				-	-	-		
1						-	-	-

#### 5.1.3 实验内容

编写实验程序，通过单片机的 P0 口控制步进电机运转。参考接线图如图 5-1-2 所示。

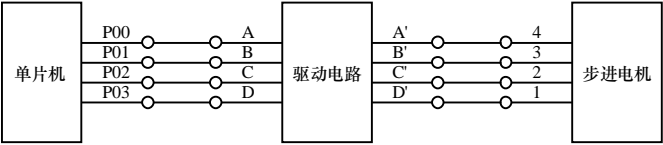


图 5-1-2 步进电机实验参考线路图

实验中 P0 端口各线的电平在各步中的情况如表 5-1-2 所示。

表 5-1-2 P0 端口引脚的电平在各步中的情况

步序	P0.3	P0.2	P0.1	P0.0	P0 口输出值
1	1	1	1	0	0EH
2	1	1	0	0	0CH
3	1	1	0	1	0DH
4	1	0	0	1	09H
5	1	0	1	1	0BH
6	0	0	1	1	03H
7	0	1	1	1	07H
8	0	1	1	0	06H

实验参考程序：(Bujin.C)

```
#include <reg51.h>
unsigned char L_value[8] = {0x0E, 0x0C, 0x0D, 0x09, 0x0B, 0x03, 0x07, 0x06};

void delay()
{
    unsigned int i;
    for(i=0; i<30000; i++);
}

void main()
{
    unsigned char m;
    P0 = 0x0;
    while(1)
    {
        for(m=0; m<7; m++)
        {
            P0 = L_value[m];
            delay();
        }
    }
}
```

#### 5.1.4 实验步骤

1. 按图 5-1-2 连接实验线路，编写实验程序，编译无误后联机调试。
2. 运行程序，观察步进电机的运转情况。

**注意：**步进电机在不使用时请断开连接，以免误操作使电机过分发热。

## 5.2 直流电机 PWM 调速实验

### 5.2.1 实验目的

了解单片机控制直流电机的方法，并掌握脉宽调制直流调速的方法。

### 5.2.2 实验原理

直流电机单元由 DC12V、1.1W 的直流电机，小磁钢，霍尔元件及输出电路构成。PWM 的示意图如图 5-2-1 所示。通过调节 T1 的脉冲宽度，可以改变 T1 的占空比，从而改变输出，达到改变直流电机转速的目的。

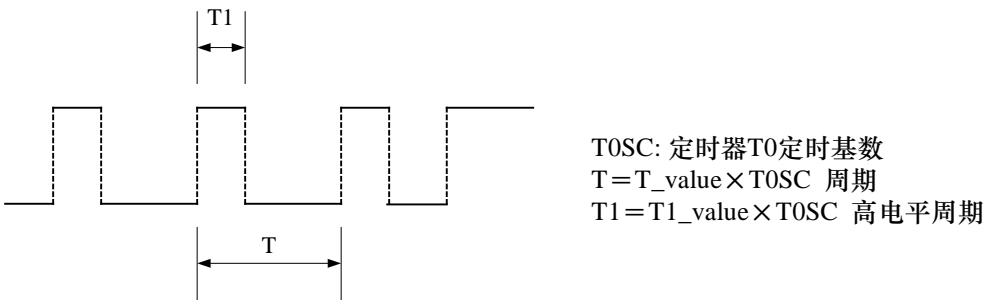


图 5-2-1 PWM 脉冲示意图

### 5.2.3 实验内容

实验接线图如图 5-2-2 所示，通过单片机的 P1.7 口来模拟 PWM 输出，经过驱动电路来驱动直流电机，实现脉宽调速。在 TD-NMC+实验平台中将 P1.7 直接与驱动电路的 A 端连接，驱动单元的输出 A'连接直流电机单元的 2 端。

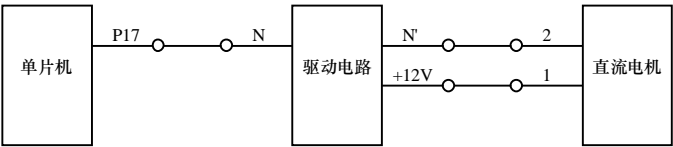


图 5-2-2 直流电机实验接线

实验参考程序: (Zhiliu.C)

```
#include <reg51.h>

#define T_value      (unsigned char)0x80      // T 周期值
#define T1_value     (unsigned char)0x20      // T 周期中高电平周期 T1 值
#define TH0_value    (unsigned char)0xFE      // 定时器 T0 计数值 (高)
#define TL0_value    (unsigned char)0x00      // 定时器 T0 计数值 (低)

sbit DRV = P1^7;
```

```
unsigned char T_Count;           // 延时次数

void init_tim0()                 // 定时器 0 初始化, 定时基数
{
    TMOD = 0x01;
    TH0 = TH0_value;
    TL0 = TL0_value;
    TR0 = 1;
    ET0 = 1;
    EA = 1;
}

void int_tim0() interrupt 1
{
    TH0 = TH0_value;
    TL0 = TL0_value;
    T_Count--;
}

void main()
{
    unsigned char Tx;
    DRV = 0;
    init_tim0();
    T_Count = T1_value;
    Tx = T1_value;
    while(1)
    {
        if(T_Count == 0)
        {
            DRV = ~DRV;
            Tx = T_value - Tx;
            T_Count = Tx;
        }
    }
}
```

#### 5.2.4 实验步骤

1. 按照图 5-2-2 接线, 编写实验程序, 编译无误后联机调试。
2. 运行程序, 观察电机运转情况。
3. 复位并停止调试, 改变 T1\_value 的值, 重新编译、链接后运行程序观察实验现象。
4. 也可以改变定时器时间来改变时间脉宽, 观察实验现象。
5. 实验结束, 复位系统板, 退出调试状态。

## 5.3 温度闭环控制实验

### 5.3.1 实验目的

1. 了解温度调节闭环控制方法；
2. 掌握 PID 控制规律及算法。

### 5.3.2 实验设备

TD 系列接口实验平台一套、TD-51 系统板一块。

### 5.3.3 实验内容

温度闭环控制原理如图 5-3-1 所示。人为数字给定一个温度值，与温度测量电路得到的温度值（反馈量）进行比较，其差值经过 PID 运算，将得到控制量并产生 PWM 脉冲，通过驱动电路控制温度单元是否加热，从而构成温度闭环控制系统。

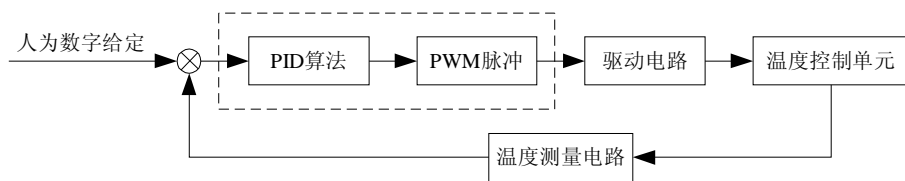
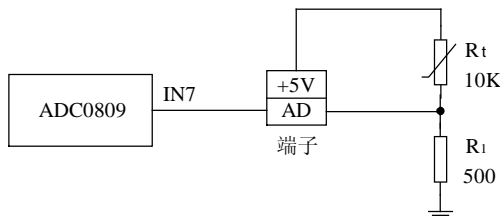


图 5-3-1 温度控制实验原理图

温度控制单元中由 7805 与一个  $24\Omega$  的电阻构成回路，回路电流较大使得 7805 芯片发热。用热敏电阻测量 7805 芯片的温度可以进行温度闭环控制实验。由于 7805 裸露在外，散热迅速。实验控制的温度范围为  $50\sim 70^{\circ}\text{C}$ 。

### 5.3.4 实验原理

实验电路中采用的是 NTC MF58-103 型热敏电阻，实验电路连接如下：



温度值与对应 AD 值的计算方法如下：

25℃: $R_t = 10\text{K}$	$V_{AD} = 5 \times 500 / (10000 + 500) = 0.238(\text{V})$	对应 AD 值: 0CH
30℃: $R_t = 5.6\text{K}$	$V_{AD} = 5 \times 500 / (5600 + 500) = 0.410(\text{V})$	对应 AD 值: 15H
40℃: $R_t = 3.8\text{K}$	$V_{AD} = 5 \times 500 / (3800 + 500) = 0.581(\text{V})$	对应 AD 值: 1EH
50℃: $R_t = 2.7\text{K}$	$V_{AD} = 5 \times 500 / (2700 + 500) = 0.781(\text{V})$	对应 AD 值: 28H

60℃:  $R_t=2.1K$      $V_{AD}=5 \times 500 / (2100 + 500)=0.962(V)$     对应 AD 值: 32H  
100℃:  $R_t=900$      $V_{AD}=5 \times 500 / (900 + 500)=1.786 (V)$     对应 AD 值: 5AH  
.....

测出的 AD 值是程序中数据表的相对偏移, 利用这个值就可以找到相应的温度值。例如测出的 AD 值为 5AH=90, 在数据表中第 90 个数为 64H, 即温度值: 100℃。

5.3.5 实验步骤

- 1. 实验接线图如图 5-3-2 所示, 按图连接实验线路图, 注意 AD 的 CLK 引脚与实验平台中的系统总线单元的 CLK 相连;
- 2. 编写实验程序, 实验参数取值范围见表 5-3-1, 编译、链接后启动调试;
- 3. 运行实验程序程序, 观察数码管的变化;
- 4. 根据实验现象, 改变 PID 参数 IBAND、KPP、KII、KDD, 重复实验, 观察实验结果, 找出合适的参数并记录。

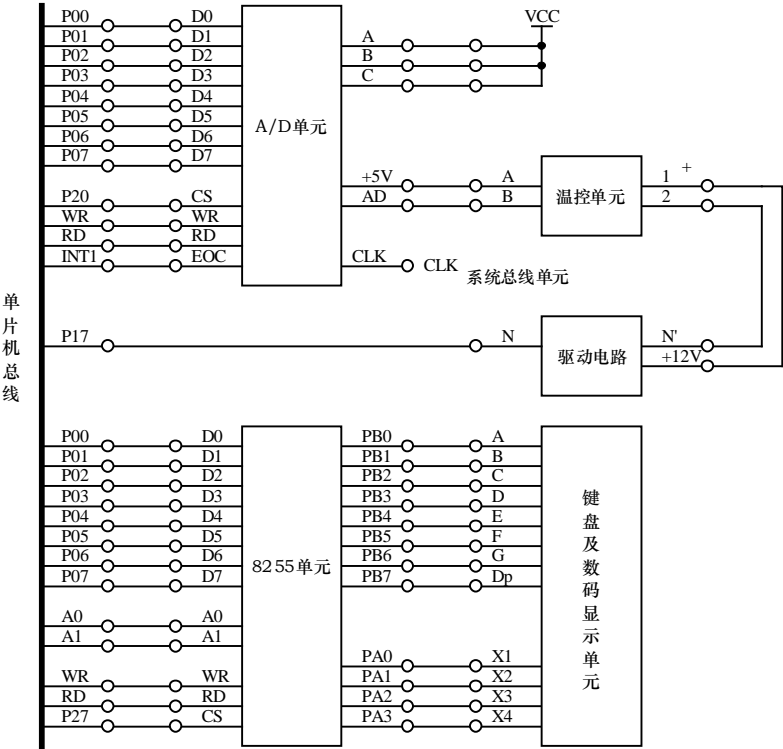


图 5-3-2 温度控制实验线路图

表 5-3-1 实验参数取值范围

符号	单位	取值范围	名称及作用
TS	10mS	00H-FFH	采样周期:决定数据采集处理快慢程度
SPEC	℃	14H-46H	给定:要求达到的温度值
IBAND		0000H-007FH	积分分离值:PID 算法中积分分离值

KPP		0000H-1FFFH	比例系数:PID 算法中比例项系数值
KII		0000H-1FFFH	积分系数:PID 算法中积分项系数值
KDD		0000H-1FFFH	微分系数:PID 算法中微分项系数值
YK	℃	0014H-0046H	反馈:通过反馈算出的温度反馈值
CK		00H-FFH	控制量:PID 算法产生用于控制的量
TKMARK		00H-01H	采样标志位
ADMARK		00H-01H	A/D 转换结束标志位
ADVALUE		00H-FFH	A/D 转换结果寄存单元
TC		00H-FFH	采样周期变量
FPWM		00H-01H	PWM 脉冲中间标志位

### 实验程序清单: (Wendu.C)

```
#include <reg51.h>
#include <absacc.h>
#include <math.h>

void pid(void);
void init(void);
void Display(void);
void clear();

int mmul(int x, int y);           /*16 位乘法, 溢出赋极值*/
int madd(int x, int y);           /*16 位加法, 溢出赋极值*/
int change32_16(int z, int t);    /*将 32 位数转化成 16 位*/
char change16_8(int wd);          /*将 16 位数转化成 8 位*/

#define C8255_A    XBYTE[0x7F00]
#define C8255_B    XBYTE[0x7F01]
#define C8255_C    XBYTE[0x7F02]
#define C8255_CON  XBYTE[0x7F03]
#define AD0809     XBYTE[0xFEFF]

sbit P17 = P1^7;

char TS      = 0x64;
int  X       = 0x80;
char SPEC    = 0x28;    // 数字给定(40)
char IBAND   = 0x60;    // 积分分离值
int  KP      = 12;      // 比例系数
char KI      = 20;      // 积分系数
char KD      = 32;      // 微分系数
int  CK;             // 控制量: PID 算法产生用于控制的量
int  TC;             // 采样周期变量
char FPWM;           // PWM 脉冲中间标志位
int  CK_1;           // 控制量变量: 记录上次控制量值
int  AAAA;           // PWM 脉冲高电平时间计算
int  VAA;            // AAAA 变量
```



```

int BBB;          // PWM 脉冲低电平时间计算
int VBB;          // BBB 变量
int TKMARK;       // 采样标志值
int ADMARK;       // AD 转换结束标志位
int ADVALUE;      // AD 采样值保存
int YK;           // 反馈: 测量温度值
int EK;
int EK_1;
int AEK;
int BEK;
unsigned char DIS;
// BCD 码显示表
unsigned char Led[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
unsigned char b[] = {0x00, 0x00, 0x00, 0x00};    // 位选

// 温度表
unsigned char code a[0x100]={0x14, 0x14, 0x14, 0x14, 0x14, 0x14, 0x14, 0x14, 0x14,
                                0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C,
                                0x1D, 0x1E, 0x1E, 0x1F, 0x20, 0x21, 0x23, 0x24, 0x25,
                                0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E,
                                0x2F, 0x31, 0x32, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
                                0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40,
                                0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A,
                                0x4B, 0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x4F, 0x50, 0x51,
                                0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A,
                                0x5B, 0x5C, 0x5D, 0x5E, 0x5F, 0x60, 0x61, 0x62, 0x63,
                                0x64, 0x64, 0x65, 0x65, 0x66, 0x66, 0x67, 0x68, 0x69,
                                0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6E, 0x6F, 0x6F, 0x70,
                                0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
                                0x7A, 0x7B, 0x7C, 0x7D, 0x7E, 0x7F, 0x80, 0x81, 0x82,
                                0x83, 0x84, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8A,
                                0x8B, 0x8C, 0x8E, 0x8F, 0x90, 0x91, 0x92, 0x93, 0x94,
                                0x95, 0x96, 0x97, 0x98, 0x99, 0x9A, 0x9B, 0x9B, 0x9C,
                                0x9C, 0x9D, 0x9D, 0x9E, 0x9E, 0x9F, 0x9F, 0xA0, 0xA1,
                                0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7, 0xA8, 0xA9, 0xAA,
                                0xAB, 0xAC, 0xAD, 0xAE, 0xAF, 0xB0, 0xB0, 0xB1, 0xB2,
                                0xB3, 0xB4, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8, 0xB9, 0xBA,
                                0xBB, 0xBD, 0xBE, 0xBE, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5,
                                0xC6, 0xC8, 0xCA, 0xCC, 0xCE, 0xCF, 0xD0, 0xD1, 0xD2,
                                0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, 0xDA, 0xDB, 0xDC,
                                0xDD, 0xDE, 0xE3, 0xE6, 0xE9, 0xEC, 0xF0, 0xF2, 0xF6,
                                0xFA, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                                0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                                0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                                0xFF, 0xFF, 0xFF, 0xFF };

void delay(unsigned int time)
{
    unsigned int i;
    for(i=0; i<time; i++);
}
void main()
{
    init();
    while (1)
    {
        while (1)
        {

```

```

        if (TKMARK == 0x01)                //采样周期到否
            break;
    }
    TKMARK = 0x00;
    while (1)
    {
        if (ADMARK == 0x01)                //AD 转换是否结束
            break;
    }
    ADMARK = 0x00;
    YK = a[ADVALUE];                        //查温度表
    DIS = (char)YK;
    pid();                                  //调 PID 运算函数

    if( CK <= 0x80)                          //根据 CK 产生 PWM
        AAAA = 0x00;
    else
        AAAA = CK - 0x80;

    BBB = 0x7f - AAAA;
}
}
void init(void)                            //初始化函数
{
    YK      = 0x00;                        //变量初始化
    EK      = 0x00;    EK_1    = 0x00;
    AEK     = 0x00;    BEK     = 0x00;
    CK      = 0x00;    CK_1    = 0x00;
    BBB     = 0x00;    VBB     = 0x00;
    ADVALUE = 0x00;    TKMARK  = 0x00;
    ADMARK  = 0x00;    TC      = 0x00;
    FPWM    = 0x01;    AAAA    = 0x7F;
    VAA     = 0x7F;

    C8255_CON = 0x81;
    Display();
    clear();

    TMOD = 0x11;                          //T1, T0 16 位定时器
    IP = 0x02;                            //定时器 0 中断为高优先级
    IT1 = 1;                              //外中断 1 为下降沿有效
    EX1 = 1;                              //允许 INT1 中断
    TH0 = 0xD8;    TL0 = 0xEF;
    TH1 = 0xD8;    TL1 = 0xEF;
    ET0 = ET1 = 1;                        //使能定时器中断
    TR0 = TR1 = 1;                        //启动定时器
    EA = 1;
    AD0809 = 1;                          //启动 AD
}
void myint3(void) interrupt 3              //定时器 1, LED 显示
{
    TH1 = 0xD8;
    TL1 = 0xEF;
    ET1 = 1;
    Display();

```

```

    clear();
}
void myint1(void) interrupt 2           //外中断 1, 读 AD 转换结果
{
    ADVALUE = AD0809;
    ADMARK = 0x01;
}
void myint2(void) interrupt 1           //定时器 0, 启动 AD 转换
{
    TH0 = 0xD8;
    TL0 = 0xEF;
    ET0 = 1;
    AD0809 = 1;
    if (TC < TS)
        TC++;
    else
    {
        TKMARK = 0x01;
        TC = 0x00;
    }
    if (FPWM == 0x01)                   //产生 PWM
    {
        if (VAA != 0x00)
        {
            VAA = VAA - 1;
            P17 = 0;                     //P10 输出为低, 加热
        }
        else
        {
            FPWM = 0x02;
            VBB = BBB / 2;
        }
    }
    if (FPWM == 0x02)
    {
        if (VBB != 0x00)
        {
            VBB = VBB - 1;
            P17 = 1;                     //P10 输出为高, 停止加热
        }
        else
        {
            FPWM = 0x01;
            VAA = AAAA / 2;
        }
    }
    return;
}
void pid(void)                           //PID 函数
{
    int K,P,I,D;

    K=P=I=D=0;
    EK=SPEC-YK;                          //得到偏差
    BEK=EK-EK_1-AEK;                     //△2EK
    AEK=EK-EK_1;                          //△EK 偏差变化量

```

```

    if (abs(EK)>abs(IBAND)) I=0;           //判积分分离
    else I=(EK*TS)/KI;                     //计算积分项

    P=AEK;
    D=((KD/TS)*BEK)/10000;                 //计算微分项

    //△UK=KP*△EK + KI*Ek + KD*(△EK-△EK_1) UK=△UK+UK_1
    K=madd(I,P);
    K=madd(D,K);
    K=mmul(K,KP);
    K=K/10;
    CK=K+CK_1;

    //将UK值转化成8位数据,取K的低8位值并取符号
    CK=change16_8(CK);
    CK_1=CK;
    EK_1=EK;
    CK=CK+X;
}

int mmul( int x,int y)
{
    int t,z;
    long s;
    s=x*y;
    z=(int)(s&0xFFFF);
    t=(int)((s>>16)&0xFFFF);
    s=change32_16(z,t);
    return(s);
}

int change32_16(int z,int t)              //t=高字, z=低字
{
    int s;

    if(t==0)
    {
        if((z&0x8000)==0) s=z;
        else s=0x7fff;
    }
    else if ((t&0xffff)==0xffff)
    {
        if((z&0x8000)==0) s=0x8000;
        else s=z;
    }
    else if ((t&0x8000)==0) s=0x7fff;
    else s=0x8000;
    return(s);
}

int madd(int x,int y)
{
    int t;
    t=x+y;
    if(x>=0 && y>=0)                      //同号相乘,符号位变反说明溢出
    { if((t&0x8000)!=0) t=0x7fff; }
    else if (x<=0 && y<=0)

```

```
{ if((t&0x8000)==0) t=0x8000; }
return(t);
}

char changel6_8(int wd)                //t=高字节, z=低字节
{
    char z,t,s;

    z=wd&0xFF;
    t=(wd>>8)&0xFF;

    if(t==0x00)
    {
        if((z&0x80)==0) s=z;
        else s=0x7f;
    }
    else if ((t&0xff)==0xff)
    {
        if((z&0x80)==0) s=0x80;
        else s=z;
    }
    else if((t&0x80)==0) s=0x7f;
    else s=0x80;

    return(s);
}

void Display()                          //数码管显示函数
{
    unsigned char i, j = 0xF7;
    b[3] = SPEC/10;
    b[2] = SPEC%10;
    b[1] = DIS/10;
    b[0] = DIS%10;
    for(i=0; i<4; i++)
    {
        C8255_A = j;
        C8255_B = Led[b[i]];
        delay(0x55);
        j >>= 1;
    }
}

void clear()
{
    C8255_B = 0x00;
}
```