

51 单片机开发入门与典型实例

王守中 编著

人民邮电出版社
北京

图书在版编目 (CIP) 数据

51 单片机开发入门与典型实例 / 王守中编著 . —北京 : 人民邮电出版社 , 2007.8

ISBN 978-7-115-16189-5

.5... .王... .单片微型计算机 - 系统开发 . TP368.1

中国版本图书馆 CIP 数据核字 (2007) 第 062696 号

内 容 提 要

本书以 51 单片机为主线 , 按照初学者学习的一般步骤 , 详细介绍了单片机开发的入门知识和经典实例。全书分 5 篇 , 共 25 章 , 首先介绍单片机开发环境的构建方法、单片机应用程序开发流程、单片机指令系统和单片机 C 语言基础等单片机入门知识 , 然后详细讲解实际开发中常用的单片机汇编语言开发实例和单片机 C 语言开发实例 , 最后讲解时钟设计、液晶显示和制作单片机实验板等单片机应用开发综合实例。

本书语言通俗、实例丰富、代码分析详尽 , 有较强的实用性和参考价值 , 适合大专院校计算机、电子、电气、控制及相关专业学生学习参考 , 也可供单片机开发人员和系统设计人员参考使用。

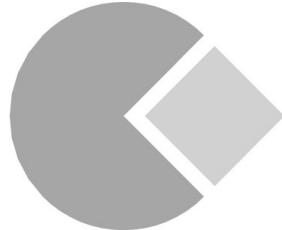
51 单片机开发入门与典型实例

-
- ◆ 编 著 王守中
 - 责任编辑 黄 焰
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
 - 邮编 100061 电子函件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 北京密云春雷印刷厂印刷
 - 新华书店总店北京发行所经销
 - ◆ 开本 : 787×1092 1/16
 - 印张 : 29
 - 字数 : 693 千字 2007 年 8 月第 1 版
 - 印数 : 1~4000 册 2007 年 8 月北京第 1 次印刷

ISBN 978-7-115-16189-5/TP

定价 : 49.00 元 (附光盘)

读者服务热线 : (010)67132692 印装质量热线 : (010)67129223



前　　言

行业背景

当前，单片机被广泛地应用于人们生活的各个领域，社会需要大量掌握单片机技能的人才，而单片机性能不断提高、价格不断降低、技术也已成熟，用户可以很容易自行搭建开发环境，任何有志者都可以通过自身努力掌握单片机开发技术。

关于本书

很多读者刚开始接触单片机的时候会感觉很困难，本书结合作者自己从零开始自学单片机的经历和体会编写而成。编写此书的初衷是希望给初学者一些指引和启发，使初学者掌握单片机学习的要领，少走弯路，快速入门，掌握单片机开发的典型实例，并能自己动手制作一块开发板，供以后实验、开发使用。

本书内容组织方式

本书按照单片机初学者的学习过程，由浅入深逐步讲解 51 单片机基础、典型实例、单片机 C 语言以及综合实例，全书分 5 篇，共 25 章。

第 1 篇为 51 单片机快速入门篇，主要讲解以下内容：

- 单片机的历史、分类以及入门时单片机的选择方法；
- 单片机开发环境的构建方法，使读者能构建自己的开发平台；
- 单片机程序的开发流程；
- 单片机的基本工作原理及其寄存器的使用；
- 单片机的指令系统以及单片机编程的经验与技巧。

第 2 篇为 51 单片机典型实例篇，本篇包含以下实例：

- 跑马灯实例；
- LED 显示实例；
- 键盘控制实例；
- 中断实例；
- 定时器/计数器实例；

- 音乐发声实例；
- 串行通信实例；
- LCD 模块实例；
- AD/DA 实例；
- 步进电机控制实例。

第 3 篇为 51 单片机 C 语言基础篇，主要讲解以下内容：

- 单片机 C 语言基础；
- 单片机 C 语言程序开发流程；
- Keil C51 的使用。

第 4 篇为 51 单片机 C 语言实例篇，本篇包含以下实例：

- 节日彩灯设计实例；
- 开关输入设计实例；
- 报警声设计实例；
- 交通灯控制实例；
- 通信测试实例。

第 5 篇为 51 单片机综合实例篇，本篇包含以下实例：

- 时钟设计综合实例；
- 液晶显示应用实例；
- 自己动手制作单片机实验板。

本书特色

- 用形象的比喻和图解的方式讲解单片机的结构和工作原理；
- 在学习编程中学指令，易于理解，学以致用；
- 书中每个实例程序都是从不同侧面、多个角度加以说明，程序的解释非常详尽；
- 充分利用模拟仿真工具，使读者加深对程序的理解；
- 详细地介绍了单片机开发环境的搭建，并教会读者如何自己动手制作单片机实验板。

联系作者

本书由王守中组织编写，贵国庆为本书第 23 章和第 24 章的编写提供了实例资料，同时参与资料整理和代码编写的有艾元、王佐臣、唱新、才力、艾智军、刘凤杰、刘克俭、翟瑞霞、李淑云、刘微、闫文龙、王建宏、李涛、王福海、刘春娥，在此一并表示感谢。

由于水平有限、时间仓促，书中有错误和不足之处在所难免，恳请广大读者批评指正。读者可以通过发送 E-mail 到 huangyan@ptpress.com.cn 与本书责任编辑联系。

编者
2007 年 6 月



目 录

51 单片机快速入门篇

第 1 章 单片机的历史和分类	3
1.1 单片机名称的由来	3
1.1.1 单片机名称的由来	3
1.1.2 单片机的特点	3
1.1.3 单片机的应用	4
1.2 单片机的分类	5
1.2.1 按用途分类	5
1.2.2 按位数分类	5
1.2.3 按系列分类	5
1.3 入门级单片机的选择	6
1.3.1 入门学习选择什么样的单片机	6
1.3.2 80C51 与 AT89C51 的区别	6
1.3.3 AT89S51 与 AT89C51 的区别	6
第 2 章 单片机开发环境的建立	8
2.1 学习单片机的必备工具	8
2.1.1 计算机	8
2.1.2 单片机集成开发系统软件	8
2.1.3 51 编程器	9
2.1.4 实验板	9
2.2 集成开发软件的下载和安装	9
2.2.1 从网上免费下载集成开发软件	9
2.2.2 MedWin 集成开发软件的安装	10
2.3 编程器的安装与使用	11

2.3.1 编程器的连接	11
2.3.2 安装编程器软件	12
2.3.3 编程器的使用	12
2.4 实验板的使用	13
 第 3 章 单片机程序开发流程	15
3.1 编写一个简单的单片机程序	15
3.1.1 目的	15
3.1.2 工作原理	15
3.1.3 用汇编语言编写程序	16
3.2 用 MedWin 开发单片机程序	16
3.2.1 编写源程序	16
3.2.2 创建项目	17
3.2.3 编译/汇编	17
3.2.4 输出 Intel HEX 文件	17
3.3 把目标文件写入单片机	18
3.3.1 选择单片机型号	18
3.3.2 进行擦除	18
3.3.3 写入文件	19
3.4 在实验板上实验	19
 第 4 章 单片机寄存器	20
4.1 80C51 单片机引脚简介	20
4.2 单片机工作的基本条件	21
4.2.1 接电源	22
4.2.2 接石英晶体振荡器和复位	22
4.2.3 单片机内装入程序	22
4.3 单片机的存储器	22
4.3.1 单元与位	23
4.3.2 字、字节和数制	23
4.3.3 程序存储器	24
4.3.4 数据存储器	24
4.4 单片机工作的基本原理	26
4.4.1 引脚与寄存器的关系	27
4.4.2 单片机中 0 和 1 的作用	27
4.4.3 工作基本原理	28
 第 5 章 单片机指令	29
5.1 学习单片机指令与编程的经验与技巧	29

5.2 单片机编程语言概述	30
5.2.1 编程语言概述	30
5.2.2 单片机使用的编程语言	31
5.2.3 80C51 汇编语言的语句结构	32
5.3 80C51 单片机指令系统	33
5.3.1 数据传送类指令	33
5.3.2 控制转移类指令	33
5.3.3 逻辑运算及移位类指令	34
5.3.4 算数运算类指令	34
5.3.5 位操作类指令	34
5.4 常用的伪指令	35
5.4.1 伪指令与 80C51 指令的不同点	35
5.4.2 常用的伪指令	35

51 单片机典型实例篇

第 6 章 跑马灯	41
6.1 点亮一只灯	41
6.1.1 硬件设计	41
6.1.2 程序设计	42
6.1.3 代码详解	42
6.1.4 实例测试	43
6.1.5 经验总结	44
6.2 模拟仿真	44
6.2.1 进入模拟仿真状态	44
6.2.2 展现观察窗口	44
6.2.3 选择调试方式	45
6.2.4 观察寄存器值的变化	45
6.3 点亮 6 只灯	46
6.3.1 程序设计	47
6.3.2 代码详解	47
6.3.3 模拟仿真	48
6.3.4 实例测试	48
6.3.5 经验总结	48
6.4 亮灯循环左移	48
6.4.1 程序设计	49
6.4.2 代码详解	49
6.4.3 模拟仿真	51

6.4.4 实例测试	51
6.4.5 经验总结	51
6.5 亮灯循环右移	51
6.5.1 程序设计	52
6.5.2 代码详解	53
6.5.3 模拟仿真	54
6.5.4 实例测试	54
6.5.5 经验总结	54
6.6 延时时间的计算	54
6.6.1 机器周期和指令周期	55
6.6.2 单重循环短暂延时	55
6.6.3 多重循环较长时间延时	55
6.6.4 对延时程序的改进	56
6.7 亮灯左移与右移循环	57
6.7.1 程序设计	57
6.7.2 代码详解	58
6.7.3 模拟仿真	58
6.7.4 实例测试	58
6.7.5 经验总结	59
6.8 双灯左移右移加闪烁	59
6.8.1 程序设计	59
6.8.2 代码详解	60
6.8.3 模拟仿真	61
6.8.4 实例测试	61
6.8.5 经验总结	61
6.9 用取表方式实现灯移动	63
6.9.1 程序设计	63
6.9.2 代码详解	64
6.9.3 模拟仿真	66
6.9.4 实例测试	67
6.9.5 经验总结	67
第 7 章 LED 显示	68
7.1 数码管工作原理及显示码	68
7.1.1 LED 数码管结构	68
7.1.2 工作原理	68
7.1.3 数码管显示码	69
7.2 让数码管静态显示 6	69
7.2.1 硬件设计	70

7.2.2 程序设计	70
7.2.3 代码详解	71
7.2.4 模拟仿真	72
7.2.5 实例测试	72
7.2.6 经验总结	72
7.3 循环显示 0 ~ 9	72
7.3.1 程序设计	72
7.3.2 代码详解	74
7.3.3 模拟仿真	74
7.3.4 实例测试	74
7.3.5 经验总结	75
7.4 两位数码管显示 00 ~ 99	75
7.4.1 硬件设计	75
7.4.2 程序设计	75
7.4.3 代码详解	77
7.4.4 模拟仿真	79
7.4.5 实例测试	79
7.4.6 经验总结	79
第 8 章 键盘控制	80
8.1 用 8 位 DIP 开关控制 LED	80
8.1.1 硬件设计	80
8.1.2 程序设计	81
8.1.3 代码详解	81
8.1.4 模拟仿真	82
8.1.5 实例测试	82
8.1.6 经验总结	82
8.2 用 4 位 DIP 开关控制数码管显示	82
8.2.1 硬件设计	82
8.2.2 程序设计	83
8.2.3 代码详解	84
8.2.4 模拟仿真	85
8.2.5 实例测试	86
8.2.6 经验总结	86
8.3 按键开关控制指示灯	86
8.3.1 硬件设计	86
8.3.2 程序设计	86
8.3.3 代码详解	88
8.3.4 模拟仿真	88

8.3.5 实例测试	89
8.3.6 经验总结	89
8.4 键盘控制概述	89
8.4.1 按键的特性	89
8.4.2 键盘输入中要解决的问题	90
8.4.3 独立按键式键盘	90
8.4.4 矩阵式按键键盘	90
8.5 用独立式键盘控制灯移动	91
8.5.1 硬件设计	91
8.5.2 程序设计	91
8.5.3 代码详解	93
8.5.4 模拟仿真	94
8.5.5 实例测试	95
8.5.6 经验总结	95
8.6 用矩阵式键盘控制显示器	95
8.6.1 硬件设计	96
8.6.2 程序设计	96
8.6.3 代码详解	97
8.6.4 模拟仿真	101
8.6.5 实例测试	102
8.6.6 经验总结	102
第9章 中断	103
9.1 中断控制功能的作用	103
9.1.1 什么是中断	103
9.1.2 实现中断的好处	103
9.1.3 中断处理过程	104
9.1.4 中断源及入口地址	104
9.2 中断的控制及设置	105
9.2.1 中断允许控制寄存器 IE	105
9.2.2 中断优先级控制寄存器 IP	106
9.2.3 定时器控制寄存器 TCON	107
9.2.4 串行口控制寄存器 SCON	108
9.3 用外部中断控制灯闪烁	108
9.3.1 硬件设计	108
9.3.2 程序设计	109
9.3.3 代码详解	110
9.3.4 模拟仿真	112
9.3.5 实例测试	112

9.3.6 经验总结	113
9.4 用多级外部中断控制灯移动	113
9.4.1 硬件设计	113
9.4.2 程序设计	114
9.4.3 代码详解	116
9.4.4 模拟仿真	117
9.4.5 实例测试	117
9.4.6 经验总结	117
第 10 章 定时器/计数器	118
10.1 定时器/计数器的用途及工作原理	118
10.1.1 定时器/计数器的用途	118
10.1.2 定时器/计数器的结构	118
10.1.3 定时器/计数器的工作原理	118
10.2 定时器/计数器的控制寄存器	120
10.2.1 工作模式控制寄存器 TMOD	120
10.2.2 定时器控制寄存器 TCON	121
10.2.3 4 种工作模式的特点	121
10.3 定时器/计数器的初始化设置	122
10.3.1 模式 0 的初始化步骤	122
10.3.2 模式 1 的初始化步骤	123
10.3.3 模式 2 的初始化步骤	123
10.3.4 模式 3 的初始化步骤	123
10.4 使用定时器延时	124
10.4.1 硬件设计	124
10.4.2 程序设计	124
10.4.3 代码详解	125
10.4.4 模拟仿真	126
10.4.5 实例测试	127
10.4.6 经验总结	127
10.5 定时器加软件计数延时	127
10.5.1 程序设计	127
10.5.2 代码详解	129
10.5.3 模拟仿真	129
10.5.4 实例测试	130
10.5.5 经验总结	130
10.6 定时与计数演示灯	130
10.6.1 硬件设计	130
10.6.2 程序设计	131

10.6.3 代码详解	132
10.6.4 实例测试	132
10.6.5 经验总结	133
第 11 章 音乐发声	134
11.1 发声实验	134
11.1.1 硬件设计	134
11.1.2 程序设计	134
11.1.3 代码详解	135
11.1.4 模拟仿真	135
11.1.5 实例测试	136
11.1.6 经验总结	136
11.2 变频报警	136
11.2.1 程序设计	136
11.2.2 代码详解	137
11.2.3 模拟仿真	138
11.2.4 实例测试	138
11.2.5 经验总结	138
11.3 歌曲演奏	139
11.3.1 编程演奏器原理	139
11.3.2 程序设计	141
11.3.3 代码详解	143
11.3.4 模拟仿真	144
11.3.5 实例测试	145
11.3.6 经验总结	145
11.4 电子琴	145
14.4.1 硬件设计	145
14.4.2 程序设计	146
14.4.3 代码详解	148
14.4.4 模拟仿真	149
14.4.5 实例测试	149
14.4.6 经验总结	149
第 12 章 串行通信	150
12.1 单片机串行通信功能	150
12.1.1 单片机串行通信的作用	150
12.1.2 串行通信中双方基本约定	150
12.1.3 串行口的结构和通信过程	151
12.2 串行口的控制	152

12.2.1 电源和数据传输率控制寄存器 PCON	152
12.2.2 串行口控制寄存器 SCON	152
12.2.3 串行口的 4 种工作方式	153
12.3 扩展 8 个输出端口	153
12.3.1 硬件设计	153
12.3.2 程序设计	154
12.3.3 代码详解	156
12.3.4 模拟仿真	156
12.3.5 实例测试	157
12.3.6 经验总结	157
12.4 扩展 8 个输入端口	157
12.4.1 硬件设计	157
12.4.2 程序设计	158
12.4.3 代码详解	159
12.4.4 模拟仿真	160
12.4.5 实例测试	160
12.4.6 经验总结	160
12.5 向计算机发送一封信	160
12.5.1 硬件设计	161
12.5.2 程序设计	161
12.5.3 代码详解	162
12.5.4 模拟仿真	163
12.5.5 实例测试	163
12.5.6 经验总结	163
第 13 章 LCD 模块及其应用	164
13.1 LCD 模块	164
13.1.1 LCD 的分类	164
13.1.2 LCD 模块的引脚	164
13.1.3 寄存器选择及显示器地址	165
13.1.4 LCM 控制指令	166
13.2 一个简单的液晶显示程序	167
13.2.1 硬件设计	167
13.2.2 程序设计	167
13.2.3 代码详解	170
13.2.4 实例测试	171
13.2.5 经验总结	171
13.3 使 LCD 显示两行字符	172
13.3.1 程序设计	172

13.3.2 代码详解	173
13.3.3 实例测试	174
13.3.4 经验总结	174
13.4 LCD 显示字符串	175
13.4.1 程序设计	175
13.4.2 代码详解	177
13.4.3 实例测试	178
13.4.4 经验总结	178
13.5 LCD 循环显示	178
13.5.1 程序设计	178
13.5.2 代码详解	181
13.5.3 实例测试	182
13.5.4 经验总结	182
13.6 自编图形显示	183
13.6.1 程序设计	183
13.6.2 代码详解	186
13.6.3 实例测试	187
13.6.4 经验总结	187
第 14 章 AD 与 DA 及其应用	188
14.1 信号转换概述	188
14.1.1 模拟信号	188
14.1.2 数字信号	188
14.1.3 信号转换	188
14.2 简单 DA 转换程序	189
14.2.1 硬件设计	189
14.2.2 程序设计	190
14.2.3 代码详解	190
14.2.4 实例测试	190
14.2.5 经验总结	191
14.3 指拨开关控制输出电压	191
14.3.1 硬件设计	191
14.3.2 程序设计	191
14.3.3 代码详解	192
14.3.4 模拟仿真	192
14.3.5 实例测试	192
14.3.6 经验总结	193
14.4 DAC 输出锯齿波	193
14.4.1 程序设计	193

14.4.2 代码详解	194
14.4.3 模拟仿真	194
14.4.4 实例测试	194
14.4.5 经验总结	194
14.5 单线数字温度传感器	195
14.5.1 引脚及其与单片机的连接方式	195
14.5.2 DS18B20 的主要特性	195
14.5.3 内部结构	196
14.5.4 高速暂存存储器	196
14.5.5 DS18B20 通信协议	197
14.5.6 使用注意事项	198
14.6 数字温度计	198
14.6.1 硬件设计	198
14.6.2 程序设计	199
14.6.3 代码详解	203
14.6.4 实例测试	204
14.6.5 经验总结	204
第 15 章 步进电机的控制	205
15.1 步进电机的工作原理	205
15.1.1 步进电机的种类	205
15.1.2 步进电机工作原理	205
15.1.3 小型步进电机驱动电路	207
15.2 步进电机正转	207
15.2.1 硬件设计	207
15.2.2 程序设计	207
15.2.3 代码详解	209
15.2.4 模拟仿真	209
15.2.5 实例测试	210
15.2.6 经验总结	210
15.3 步进电机反转	210
15.3.1 程序设计	211
15.3.2 代码详解	212
15.3.3 模拟仿真	213
15.3.4 实例测试	213
15.3.5 经验总结	213
15.4 步进电机转速控制	213
15.4.1 程序设计	213
15.4.2 代码详解	215

15.4.3 模拟仿真	216
15.4.4 实例测试	216
15.4.5 经验总结	216
15.5 开关控制步进电机正反转	217
15.5.1 硬件设计	217
15.5.2 程序设计	217
15.5.3 代码详解	220
15.5.4 模拟仿真	222
15.5.5 实例测试	222
15.5.6 经验总结	222

51 单片机 C 语言基础篇

第 16 章 单片机 C 语言入门	225
16.1 C 语言与 C51	225
16.1.1 C 语言与 C51	225
16.1.2 C 语言编程的优点	225
16.1.3 C 语言和汇编语言混合编程	225
16.2 学习 C51 的准备工作	226
16.2.1 计算机	226
16.2.2 51 单片机 C 语言编译器	226
16.2.3 51 编程器和实验板	226
16.3 单片机 C 语言程序开发流程	226
16.4 单片机 C 语言入门实例	227
16.4.1 程序工作原理	227
16.4.2 源程序	228
16.4.3 程序说明	228
16.5 单片机 C 语言编程特点	228
16.5.1 程序工作原理	228
16.5.2 用两种语言编写	229
16.5.3 C 语言程序编写特点	230
16.6 单片机 C 程序的基本结构	231
16.6.1 主函数	231
16.6.2 函数	231
16.6.3 头文件	231
16.7 C51 数据类型、常量与变量	232
16.7.1 C51 的数据类型	232
16.7.2 常量	233

16.7.3 变量	234
16.7.4 数组	235
16.8 C51 常用的运算符	235
16.8.1 赋值运算符	235
16.8.2 增量和减量运算符	236
16.8.3 关系运算符	236
16.8.4 逻辑运算符	236
16.8.5 位运算符	237
16.8.6 运算符的运算优先次序	239
16.9 C51 流程控制语句	239
16.9.1 流程结构及其流程图	239
16.9.2 流程控制语句按功能分类	240
16.9.3 循环语句	241
16.9.4 选择语句	243
16.10 C51 函数	244
16.10.1 C51 函数定义的一般形式	245
16.10.2 C51 库函数	246
16.10.3 C51 中断函数	246
第 17 章 Keil C51 的使用	248
17.1 Keil C51 的安装	248
17.2 用 Keil C51 开发单片机	250
17.2.1 编写源程序	250
17.2.2 建立工程项目文件	251
17.2.3 产生可执行的 HEX 文件	255
17.3 Keil 中的软件仿真	256
17.3.1 操作的一般步骤	257
17.3.2 仿真举例说明	258
17.3.3 几个常用命令使用区别	262

51 单片机 C 语言实例篇

第 18 章 节日彩灯设计	265
18.1 彩灯闪烁	265
18.1.1 硬件设计	265
18.1.2 程序设计	266
18.1.3 代码详解	267
18.2 延时模块	267

18.2.1 延时原理	267
18.2.2 软件延时	267
18.2.3 利用定时器延时	268
18.3 彩灯由右向左侧逐渐点亮	269
18.3.1 程序设计	269
18.3.2 代码详解	270
18.4 单组彩灯循环左右移动	270
18.4.1 程序设计	270
18.4.2 代码详解	272
18.4.3 经验总结	273
18.5 采用制表方法实现彩灯变化	274
18.5.1 程序设计	274
18.5.2 代码详解	275
第 19 章 开关输入设计	277
19.1 单开关输入状态指示灯	277
19.1.1 硬件设计	277
19.1.2 程序设计	278
19.1.3 代码详解	278
19.1.4 经验总结	279
19.2 多路开关输入状态指示灯	279
19.2.1 硬件设计	279
19.2.2 程序设计	280
19.2.3 代码详解	281
19.3 多路开关控制灯	281
19.3.1 硬件设计	281
19.3.2 程序设计	282
19.3.3 代码详解	283
19.4 按钮开关次数显示灯	283
19.4.1 硬件设计	283
19.4.2 程序设计	283
19.4.3 代码详解	285
19.5 一键多功能控制	285
19.5.1 程序设计	285
19.5.2 代码详解	286
第 20 章 报警声设计	288
20.1 发出 1kHz 声音	288
20.1.1 硬件设计	288

20.1.2 程序设计	288
20.1.3 代码详解	289
20.2 发出嘀、嘀声	289
20.2.1 程序设计	290
20.2.2 代码详解	290
20.2.3 经验总结	291
20.3 救护车声	291
20.3.1 程序设计	291
20.3.2 代码详解	292
20.4 闹钟铃声	292
20.4.1 程序设计	292
20.4.2 代码详解	293
20.4.3 经验总结	293
20.5 发出 20 次的报警声	294
20.5.1 程序设计	294
20.5.2 代码详解	294
20.5.3 经验总结	295
20.6 警报的同时 LED 闪烁	295
20.6.1 硬件设计	295
20.6.2 程序设计	295
20.6.3 代码详解	296
第 21 章 交通灯信号控制	298
21.1 采用定时器延时	298
21.1.1 硬件设计	298
21.1.2 程序设计	299
21.1.3 代码详解	300
21.1.4 经验总结	300
21.2 灯交互闪烁	301
21.2.1 硬件设计	301
21.2.2 程序设计	301
21.2.3 代码详解	303
21.2.4 经验总结	303
21.3 交通信号灯控制	303
21.3.1 硬件设计	304
21.3.2 程序设计	304
21.3.3 代码详解	306
21.3.4 经验总结	307
21.4 改进的交通信号灯控制	307

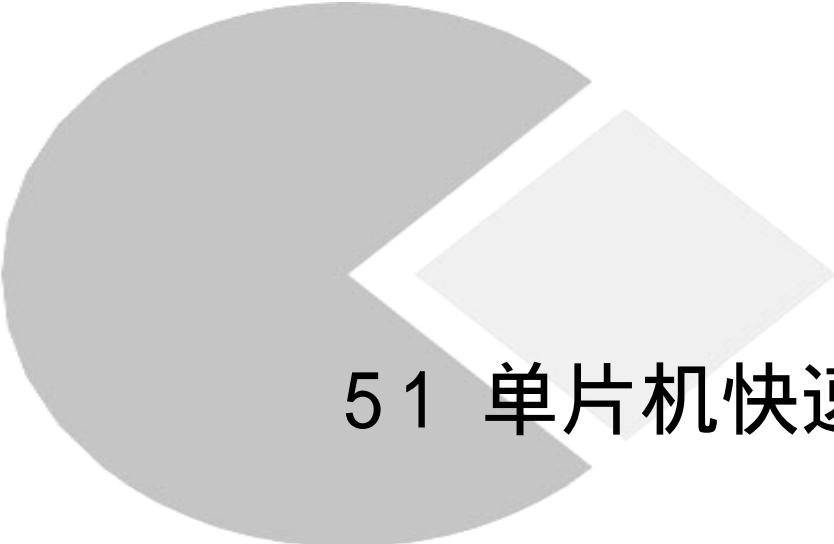
21.4.1 硬件设计	307
21.4.2 程序设计	307
21.4.3 代码详解	310
21.4.4 经验总结	310
第 22 章 通信测试	311
22.1 发送一个字符	311
22.1.1 硬件设计	311
22.1.2 程序设计	311
22.1.3 代码详解	313
22.1.4 经验总结	313
22.2 发送一个字符串	313
22.2.1 程序设计	313
22.2.2 代码详解	315
22.2.3 经验总结	316
22.3 接收指令	316
22.3.1 程序设计	316
22.3.2 代码详解	317
22.3.3 经验总结	317
22.4 发送接收测试程序	317
22.4.1 程序设计	318
22.4.2 代码详解	319
22.4.3 经验总结	320

51 单片机综合实例篇

第 23 章 时钟设计综合实例	323
23.1 简单时钟设计	323
23.1.1 学习单片机时钟设计目的	323
23.1.2 时钟结构与原理	324
23.1.3 走时功能的设计	325
23.1.4 显示部分的设计	326
23.1.5 调整时间部分的设计	328
23.1.6 喇叭和指示灯等子程序	330
23.1.7 时钟主程序	331
23.1.8 简单时钟程序清单	333
23.2 带定时功能的闹铃时钟设计	339
23.2.1 闹钟结构与原理	340

23.2.2 闹铃功能主要子程序	340
23.2.3 闹钟主程序	344
23.2.4 闹钟程序清单	346
23.3 带定时和倒计时功能的时钟设计	356
23.3.1 结构与原理	356
23.3.2 倒计时功能主要子程序	357
23.3.3 带倒计时闹钟程序清单	360
23.4 简单时钟的 C 语言程序设计	373
23.4.1 时钟结构和使用方法	373
23.4.2 走时功能的设计	373
23.4.3 显示功能的设计	374
23.4.4 调整时间功能的设计	375
23.4.5 按键扫描等其他函数	376
23.4.6 时钟主函数	377
23.4.7 简单时钟 C 语言程序清单	378
第 24 章 液晶显示应用实例	383
24.1 液晶显示秒表	383
24.1.1 硬件设计	383
24.1.2 程序设计	383
24.1.3 代码详解	393
24.1.4 操作说明	394
24.2 液晶显示温度控制器	395
24.2.1 硬件设计	395
24.2.2 程序设计	395
24.2.3 代码详解	414
24.2.4 操作说明	416
第 25 章 动手制作单片机实验板	417
25.1 制作实验板准备工作	417
25.1.1 制作实验板的目的	417
25.1.2 制作前的准备工作	418
25.1.3 焊接技巧	419
25.2 单片机外围常用元器件及其检测方法	420
25.2.1 发光二极管和 LED 数码管	420
25.2.2 三极管	421
25.2.3 电阻和电容	423
25.3 实验板制作过程	425
25.3.1 实验板功能简介	425

25.3.2 简单稳压电源的制作	427
25.3.3 单片机最小系统的制作	428
25.3.4 LED 数码管显示模块的制作	429
25.3.5 其他实验电路的制作	430
25.3.6 单片机端口插针座连接线	433
附录 A 80C51 单片机指令速查表	435
附录 B 数的制式转换表	441
附录 C 光盘使用说明	442



5.1 单片机快速入门篇

这里是一个梦想开始的地方。只需要一台普通计算机，使用网上下载的免费单片机集成开发软件，购买一块编程器，再自己动手制作一块实验板，便可以拥有自己的实验工作室，拥有自己学习和产品开发的平台，也就等于拥有了自己的起飞基地。

本篇先概括地介绍单片机的历史、分类，接着详细介绍单片机的结构、工作原理及开发语言，单片机程序开发流程，入门时对单片机、编程器、实验板的选择，以及如何构建开发环境、开发平台等。



第1章 单片机的历史和分类

单片机是一台微型计算机。大家对计算机都非常熟悉，只要安装相应的程序，它就能自动地完成相应的工作。单片机是一台微型计算机，虽然功能没有普通计算机那么强大，可是它的体积很小，在很多场合下普通计算机不能完成的工作，单片机却能出色地完成。

1.1 单片机名称的由来

1.1.1 单片机名称的由来

单片机在外观上与常见的集成电路块一样，体积很小，多为黑色长条状，条状左右两侧各有一排金属引脚，可与外电路连接，如图 1.1 所示。



图 1.1 AT89C51 单片机外观

单片机体积虽小，但“五脏俱全”，其内部结构与普通计算机结构类似，也是由中央处理器（CPU）、存储器和输入/输出（I/O）3 大基本部分构成。实际就是把一台普通计算机经过简化，浓缩在一小片芯片内，形成了芯片级的计算机（single chip microcomputer），即单芯片微型计算机，简称单片机。

单片机也称为微控制器或嵌入式微控制器。计算机是靠输入程序来工作的，同样，单片机工作也需要事先输入程序。

1.1.2 单片机的特点

1. 高性能、低价格

一片单片机从功能上讲相当于一台微型计算机，可是价格却很低，一片单片机的价

格一般在几元到几十元之间。而且，随着科学技术的发展和市场竞争的需要，世界上生产单片机的各大公司都在不断地采用新技术来提高单片机的性能，同时又进一步降低其价格。

2. 体积小、可靠性高

在单片机的片内，除了一般必须具有的 ROM、RAM、定时器/计数器、中断系统外，还尽可能地把众多的各种外围功能器件集成在片内，减少了外部各芯片之间的连接，大大提高了单片机的可靠性和抗干扰能力。

3. 低电压、低功耗

一般单片机的工作电压为 5V，有的单片机可以在 1.8~3.6V 的电压下工作，而且，功耗降至 μA 级。例如，MSP430 超低功耗类型的单片机，两个纽扣电池就可以保障其运行长达近 10 年。单片机的这种低电压、低功耗的特性，对于设计和开发携带式智能产品和家用消费类产品显得非常重要。

1.1.3 单片机的应用

只需在电路中添加少许元器件，通过编写程序就可以实现多种功能的单片机自动控制。单片机接上键盘可以进行信号输入；单片机接上显示器可以实现数据显示；单片机接上喇叭可以实现声音输出；单片机可以用来通信，也可以用来计数和定时，还可以控制彩灯的闪烁、电机的运转以及机器人的活动等，如图 1.2 所示。

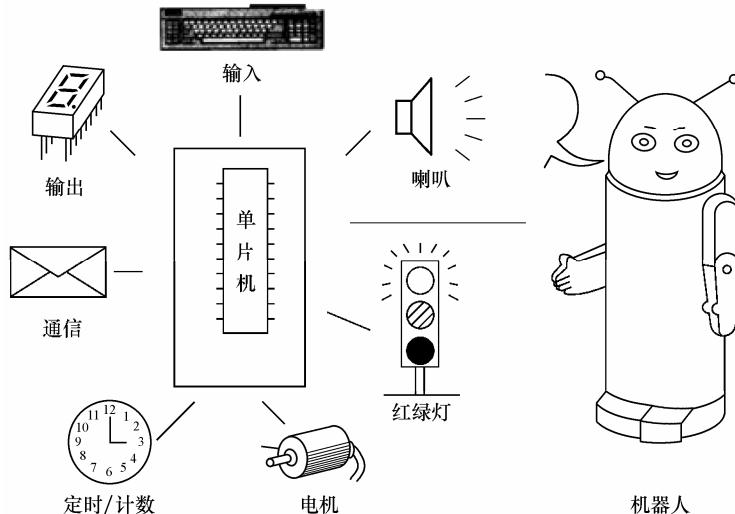


图 1.2 单片机应用示意图

由于单片机体积小巧、功能强大、应用灵活、价格便宜，所以应用十分广泛。已经在工业控制、国防装备、智能仪器等领域得到了广泛应用。现在，人们日常生活中所使用的各种家用电器，例如，洗衣机、电冰箱、空调、微波炉、电饭煲、音响、电风扇及高档电

子玩具等，也普遍采用了单片机来代替传统的控制电路，既降低了成本，又提高了自动化程度。

1.2 单片机的分类

单片机可以按用途、位数和系列进行分类。

1.2.1 按用途分类

单片机按用途可分为两大类：专用型单片机和通用型单片机。

专用型单片机用途比较专一，出厂时程序已经一次性固化好、不能再修改，电子表里的单片机就是其中的一种，其特点是生产成本低，适合大批量生产。

通用型单片机的用途很广泛，使用不同的接口电路、编写不同的应用程序就可实现不同的功能。例如，小到家用电器、仪器仪表，大到工控设备和生产流水线都可用通用型单片机来实现自动化控制。新产品开发、教学实验所使用的单片机也多是通用型单片机。

1.2.2 按位数分类

单片机按位数分可分为有低档的4位机、8位机，高档的8位机、16位机、32位机。

其中，低档的4位机、8位机属于早期生产的初级单片机，寻址范围不大于4KB，且无串行口。

当前广泛应用的是高档的8位机，这类单片机已能满足控制领域中多数场合的需求。

Intel公司生产的80C196/296系列、台湾凌阳公司生产的SPCE061A系列等是典型的16位机，随着性能的提高、价格的降低，也开始得到广泛应用。

32位机，如Motorola公司生产的MC68HC376等，具有极强的数据处理、逻辑运算和信息存储能力。单片机的位数越高其性能也越强。

1.2.3 按系列分类

单片机按系列分可分为80C51系列、PIC系列和AVR系列等，目前最常用的单片机有如下几种。

- Intel公司生产的80C51系列、MCS96系列单片机。
- Atmel公司生产的AT89系列（80C51内核）AVR系列等单片机。
- Microchip公司生产的PIC系列单片机。
- Motorola公司生产的68HCXX系列单片机。
- Zilog公司生产的Z86系列单片机。
- Philips公司生产的87、80系列（80C51内核）单片机。
- Siemens公司生产的SAB80系列（80C51内核）单片机。
- NEC公司生产的78系列单片机。

1.3 入门级单片机的选择

1.3.1 入门学习选择什么样的单片机

入门学习时一般选择 80C51 系列单片机，主要原因如下。

1. 目前介绍 80C51 系列单片机的书籍比较多，这就为初学者学习和查找资料提供了方便。同时，80C51 系列单片机的开发工具比较多，在网上可以免费下载，很容易建立学习、开发环境。

2. 80C51 系列单片机在我国普及的时间比较早，开发和应用的实例比较多，在学习编写程序时有丰富的实例可以参考和借鉴。

3. 80C51 的核心技术是单片机发展的基础，学会 80C51 系列单片机之后，再学其他单片机会触类旁通，因为单片机的开发方法是类似的。

1.3.2 80C51 与 AT89C51 的区别

80C51 系列单片机最早是由 Intel 公司开发和生产的，Intel 公司在 1980 年推出 MCS-51 单片机，也称 80C51 单片机。由于 80C51 单片机应用早，影响很大，已成为事实上的工业标准。

后来很多著名厂商如 Atmel、Philips 等公司申请了版权，生产了各种与 80C51 兼容的单片机系列。虽然制造工艺在不断地改进，但内核却没有变化，我们称这些与 80C51 内核相同的单片机为 80C51 系列单片机或 51 系列单片机。

这类单片机的指令系统完全兼容，而且大多数管脚也兼容。所以，在 51 系列单片机教材方面目前仍然沿用 Intel 公司 80C51 (MCS-51) 单片机名为书名；开发软件和工具也是一样，统称为 80C51 开发系统、开发环境等。常用的 ASM51、Keil C51、MedWin 等均是 80C51 系列单片机的开发工具软件。

但是，80C51 (MCS-51) 单片机是早期产品，用户无法将自己编写的程序烧写到单片机内的存储器中，只能将程序交由芯片厂商代为烧写，并且是一次性的。8751 单片机的内部存储器有了改进，用户可以将自己编写的程序写入单片机的内部存储器中，但需要用紫外线灯照射一定时间后再烧写，烧写次数也是有一定限制的。

AT89C51 单片机是 Atmel 公司 1989 年生产的产品，Atmel 率先把 80C51 内核与 Flash 技术相结合，推出轰动业界的 AT89 系列单片机。

AT89C51 单片机与 80C51 单片机的基本结构是一样的，编程所使用的指令以及单片机的管脚都与 80C51 单片机相同，即完全兼容。由于采用了 Flash 工艺制作的内部存储器（也称闪速存储器），用户可以用电的方式进行反复快速擦除、改写，这给初学者学习单片机提供了极大的方便。本书在实验中选用 AT89C51 等具有 Flash 内部存储器的单片机。

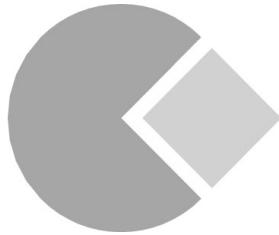
1.3.3 AT89S51 与 AT89C51 的区别

AT89S51 和 AT89C51 单片机都是 Atmel 公司生产的 80C51 系列单片机。AT89S51 单片

机对 AT89C51 单片机进行了很多改进，新增了很多功能，性能有了较大提升。在使用上与 AT89C51、80C51 单片机完全兼容，在 AT89C51、80C51 单片机上能运行的程序，在 AT89S51 单片机上都能运行。

AT89S51 相对于 AT89C51 增加的新功能主要有：ISP 在线编程功能、最高工作频率提升为 33MHz、具有双工 UART 串行通道、内部集成看门狗计时器等。

但要注意，向 AT89C51 单片机写入程序与向 AT89S51 单片机写入程序的方法有所不同，所以，购买的编程器，必须具有写入 AT89S51 单片机的功能，以适应产品的更新。Atmel 公司现在已经停止生产 AT89C51 型号的单片机，被其 AT89S51 型号的单片机所代替。



第2章 单片机开发环境的建立

边学习边动手实践对于单片机初学者来说非常重要，可以使初学者快速入门。所以，首先要建立一个开发环境。

所需要的器材有：一台普通计算机、51编程器和实验板，相关的软件会在购买编程器时一并获得，有的还可以上网免费下载。

2.1 学习单片机的必备工具

学习单片机首先要创建一个开发环境。实践证明多实验能够帮助初学者逐步理解单片机的原理并掌握开发技巧。所以，学习前应准备好实验所需的器材和单片机开发的相关软件。

2.1.1 计算机

单片机开发对计算机的要求不高，只要能正常运行 Windows 操作系统的计算机即可。

2.1.2 单片机集成开发系统软件

单片机集成开发系统软件，是指用来在计算机上编写、汇编和仿真、调试单片机程序软件。

目前用来开发单片机的应用软件比较多，如 Keil 公司的 Keil C51（评估展示版）以及 Medwin 等，都是比较好的 51 单片机集成开发系统软件。单片机的程序开发流程如图 2.1 所示。

单片机的程序开发流程是：编写程序 编译 连接（软件仿真） 烧写 测试。

（1）用编辑软件编写程序，注意，使用汇编语言编写的程序文件名后缀（即扩展名）是 .ASM，编写的程序称为源程序。例如，汇编语言源程序 Test.ASM，其中，Test 是文件名（可任意），.ASM

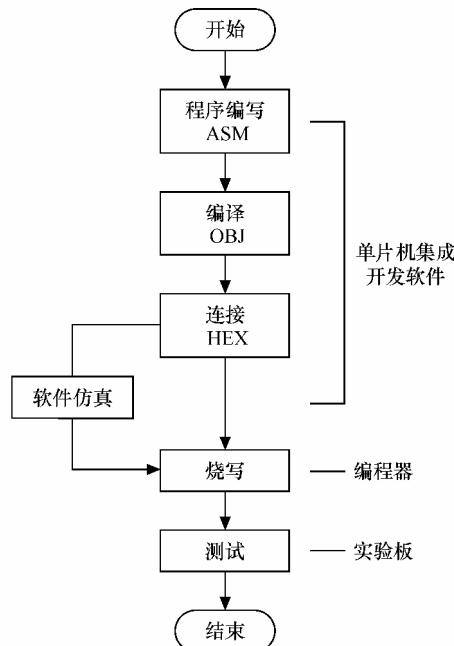


图 2.1 单片机的程序开发流程

是扩展名(必须用)。

- (2) 将源程序用编译软件进行编译，生成扩展名为.OBJ的文件(如Test.OBJ)。
- (3) 用连接软件进行连接，生成扩展名为.HEX的文件(如Test.HEX)。
- (4) 通过编程器将扩展名为.HEX的可执行文件烧写到单片机内。在写入单片机之前还可以进行软件仿真，即在软件上模拟单片机程序运行情况，以便进行调试和修改。

上述4步工作既可以通过4个分立工具软件来完成，也可以用一个单片机集成开发系统软件来完成。所谓集成，就是指将源程序编写、编译、连接、调试等开发单片机所要用到的工具集成到一个软件中。

例如，Keil C51、MedWin就是两个常用的单片机集成开发系统软件。其中，MedWin单片机集成开发系统软件界面类似于Windows，用MedWin可以编写源程序，同时还可以进行编译、连接和软件仿真，操作简便，很适合初学者使用；Keil C51功能强大，使用广泛，并支持80C51的C语言编程，是单片机初学者的理想选择。

2.1.3 51编程器

51编程器是用来将编好的程序烧写到51单片机内的一个设备。

用集成开发系统软件(如Keil C51或MedWin)编写并生成单片机目标代码后，需要用编程器将目标代码，即扩展名为HEX的可执行文件烧写到单片机中。编程器是一个设备，上面有单片机插座及与计算机的连线等，一般需要购买。

编程器按功能可分单一型和万能型。单一型编程器只能对单一系列的某些型号的单片机芯片进行写入操作；万能型编程器能对多种系列的多种型号的单片机芯片进行写入操作。前者结构简单、价格便宜，很适合初学者使用；后者功能强大，但价格较贵。

编程器按与计算机的连接方式可分为串口编程器和并口编程器两种。串口编程器通过连线接在计算机的串行端口，即通信端口上；并口编程器通过连线接在计算机的并行端口，即打印机端口上。

购买时一般选择串口编程器，串口编程器还可以很方便地进行通信程序实验。

2.1.4 实验板

实验板实际上是一个小的单片机实验系统。

写入程序的单片机需要装到实验板上运行后才能验证编写的程序正确与否。实验板上带有单片机插座、发光二极管、数码管、蜂鸣器等器件。实验板可以自制，也可以购买。初学者最好先买一块装好的实验板，待学习深入后，再进行性能扩展或自己组装。

一般销售编程器的厂家也都销售实验板，有的是把实验板部分与编程器部分合装在一起，有的是分立的。

2.2 集成开发软件的下载和安装

2.2.1 从网上免费下载集成开发软件

Keil C51集成开发软件可以到Keil公司的网站www.keil.com去下载，当进入www.keil.

com 后，在首页的左边“ Software Downloads ”(软件下载) 的栏目下，单击 Evaluation Software (评估版软件)，便出现 “ Downloads the keil c51 Evaluation Tools ”(Keil C51 评估版工具) 的提示，点击该提示对话框后，就可以免费下载 Keil C51 评估版软件。

MedWin 集成开发软件可到 www.manley.com.cn 网站下载，也可以到其他单片机网站下载。

2.2.2 MedWin 集成开发软件的安装

Keil C51 的安装过程可参考本书 51 单片机 C 语言基础篇的 17.1 节。下面介绍 MedWin 集成开发系统软件的安装过程。

1. 安装

将下载的 MedWin 软件解压后，在其所属目录中双击 Setup.exe 安装文件进行安装。安装中当出现安装提示对话框时，只要不断单击“ Next ”按钮，安装程序会自动完成全部安装，其默认目录为 C:\Manley\MedWin。安装完后，桌面上会生成一个 MedWin 图标。

如果桌面上没有此图标，可按下面步骤将图标发送到桌面：点击“开始”——“程序”——“ ManLey ” → “ MedWin ”，右键单击 MeWin 文件便出现“发送到”——“桌面快捷方式”，再单击“桌面快捷方式”即可将图标发送到桌面，使启动操作更加方便。

2. 设置工作目录

工作目录是以后存放源程序及经过编译/连接等操作所生成的各种文件的地方。最好不要将工作目录设置在 C 盘，以免操作系统重新安装时编写的程序遭到破坏，一般将工作目录设置为 D 盘或 E 盘等。

工作目录可在首次安装时设置，双击 MedWin 图标，启动 MedWin 集成开发系统，启动后弹出的第一个对话框如图 2.2 所示。

单击“模拟仿真”按钮，弹出设置工作目录的“工作向导”对话框，如图 2.3 所示（只有首次安装时会出现）。



图 2.2 “端口”对话框



图 2.3 “工作向导”对话框

在“工作向导”对话框中单击“新建或打开一个文件”按钮，就可以在指定的地方建立新的工作目录，目录名可任意设定。

3. 进入 MedWin 的工作窗口

当工作目录设置完成后便进入了 MedWin 的工作窗口，如图 2.4 所示。



图 2.4 Medwin 窗口

以后启动 MedWin 单片机开发集成系统软件时，只要单击“模拟仿真”按钮，便会直接进入 MedWin 的工作窗口。

MedWin 是一个具有 Windows 窗口风格的集成开发环境。支持带语法分析的彩色文本显示、源程序断点设置记忆、实时程序计数器、PC 显示、仿真器断电自动重载、自适应连接仿真器等功能，并且支持全空间程序代码和数据空间的模拟仿真，包含对中断、定时器的模拟仿真和单片机外部设备状态分析设置、程序性能分析等实用功能。

2.3 编程器的安装与使用

在单片机集成开发系统上生成的 HEX 文件，需要使用编程器才能写入到单片机内，因此，需要安装编程器。编程器的安装非常简单，主要有两个步骤：一是将编程器连接到计算机上；二是在计算机上安装编程器程序软件。

下面以一款单一型串口方式联接的 AT51S 编程器为例介绍编程器安装过程，如图 2.5 所示。

2.3.1 编程器的连接

编程器采用串口与计算机连接，端口自动识别，无需设置，安装十分简便。先将配套的串口电缆一端与编程器的 9 芯座连接，另一端接到计算机的 COM1 或 COM2 上，电缆两端插头相同，不需区分。

再将配套的直流电源调到 12V 档，将插头插到编程器电源座上，编程器指示灯大约每隔一秒闪烁一次，表示编程器工作正常，等待服务软件的指令。若接上电源后编程器指示灯常亮或常灭，则说明编程器工作异常，需要将电源断开数秒后重新连接。

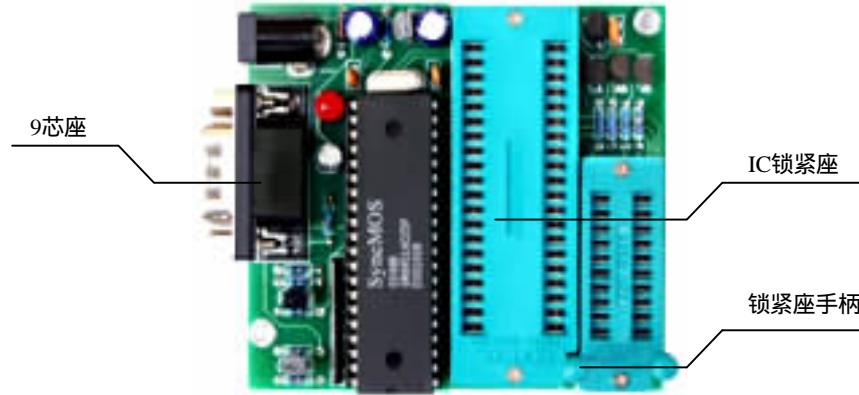


图 2.5 AT51S 编程器

将需要写入程序的单片机插入编程器 IC 锁紧座内，单片机 1 脚朝向锁紧座手柄的方向。请注意，不是所有编程器都是将单片机 1 脚朝向锁紧座手柄的方向，要依据说明书中的说明进行操作。

2.3.2 安装编程器软件

当完成编程器与计算机的连接，并且编程器处在正常的工作状态时，就可以安装编程器软件，编程器程序软件可以工作在 Windows95/98/XP/2000 操作系统上。

打开存有编程器软件的目录，双击安装文件 Setup.exe，当出现对话框时，只要不断地单击“下一步”按钮就可完成自动安装，并自动完成端口的识别。图 2.6 所示为编程器软件的窗口。

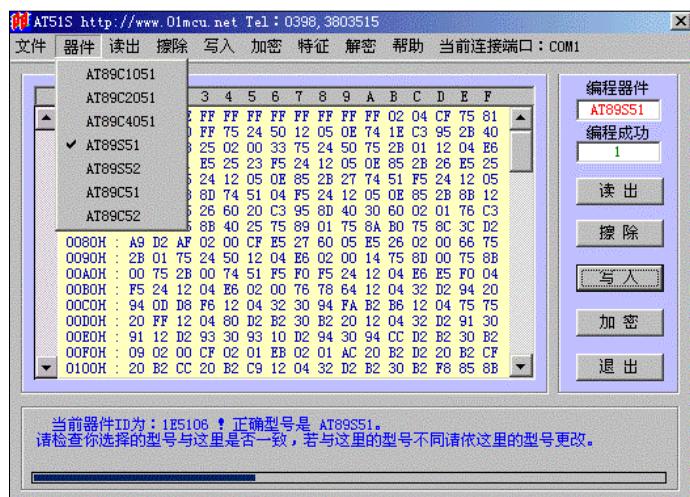


图 2.6 编程器软件窗口

2.3.3 编程器的使用

接通编程器电源，然后双击编程器软件启动图标，在出现编程器工作窗口的同时完成编

器软件与编程器的连接。如果连接失败，可根据故障原因提示进行检查，排除后再重试。当连接成功后可按下面步骤把编好的程序写入单片机。

(1) 在编程器上插入单片机

将单片机插入编程器插座并锁紧，注意使单片机 1 脚朝向锁紧座手柄的方向。

(2) 选择单片机型号

单击编程器软件工作窗口上方“器件”按钮，在出现的下拉菜单中选择相应的单片机型号。

(3) 读出与擦除

对于已经写过数据的单片机，应先使用软件的擦除功能对单片机进行擦除，擦除后从单片机读出的数据是全 FF，也可以在擦除前先将数据读出并将数据保存到磁盘中，然后再进行擦除。

(4) 打开 HEX 文件

单击软件窗口左上方“文件”菜单选项，选择“打开 HEX 文件”选项，在存放 HEX 文件的目录里找到需要写入的文件，单击该文件名后软件工作窗口中就会出现十六进制的 HEX 文件。

(5) 写入文件

单击“写入”按钮，被选中的文件就会被写入到单片机内。

2.4 实验板的使用

有的实验板与编程器部分合装在一起，如图 2.7 所示；有的是分立的，如图 2.8 所示。

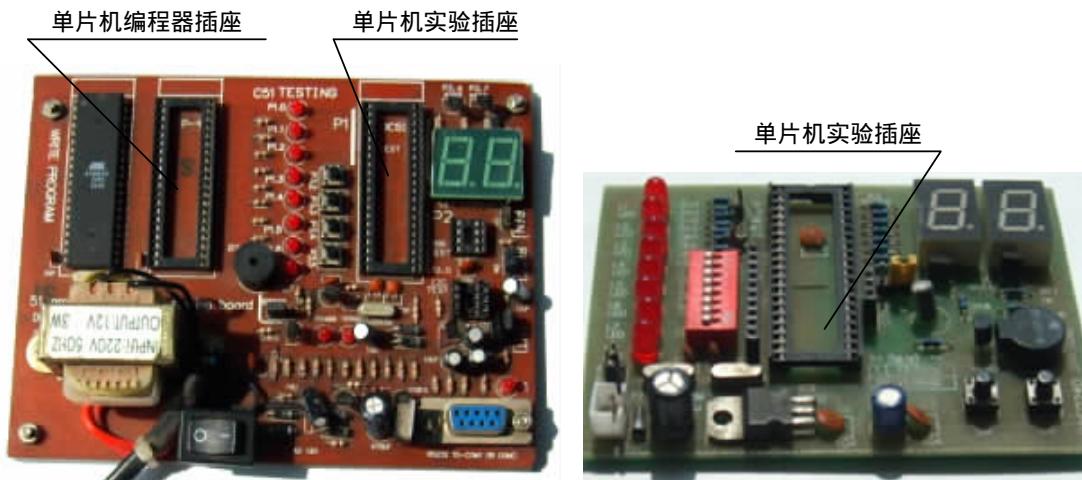


图 2.7 编程器与实验板

图 2.8 实验板

实验板的使用需注意的两个问题。

(1) 要弄清楚实验板上的器件与单片机端口的连接情况。

例如，要编写一个跑马灯流动效果的程序，编写程序前首先要弄清楚实验板上的发光二

极管是接在哪个端口上，假如是接在端口 P1 上，程序中的输出端口就是 P1。尤其是当参考示例程序时，一定要检查程序中的输入/输出端口是否与实验板上的输入/输出相一致，如果不一致要进行修改，否则无法执行。

(2) 要将单片机正确地插入到实验板插座内，同时，要确认实验板是否处在正常的工作状态。实验板是检验写入到单片机内程序是否正确的一个小实验系统，有时实验系统本身也会出现问题。所以，应该保留一片检验实验板的单片机芯片。



第3章 单片机程序开发流程

本章将通过编写一个简单的单片机程序，使读者初步了解单片机开发的全过程以及集成开发软件的使用，单片机开发的主要步骤如下。

- 编写程序。
- 编译/汇编和生成 HEX 可执行文件。
- 写入单片机。
- 在实验板上实验。

其中，在将 HEX 可执行文件写入单片机之前，还可以进行软件仿真，检查程序是否存在错误。上述步骤中的编写程序、编译/汇编、生成 HEX 可执行文件及软件仿真都可以在一个集成开发软件中完成。

3.1 编写一个简单的单片机程序

3.1.1 目的

使用汇编语言编写一个简单的单片机程序，该程序的功能是控制单片机 P1 端口输出，使接在 P1 端口（P1.0~P1.7）第 0、2、3、5、6、7 位置的发光二极管被点亮。程序很简单，目的是演示单片机开发的全过程。

3.1.2 工作原理

单片机 P1 端口输出低电平“0”会点亮 LED，输出高电平“1”会使 LED 熄灭。P1 端口输出示意图如图 3.1 所示。

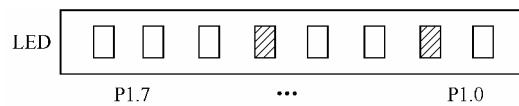


图 3.1 P1 端口输出示意图

本例程序中要使 P1.0、P1.2、P1.3、P1.5、P1.6、P1.7 位的二极管点亮，必须使其位为 0，而 P1.1、P1.4 为 1，整理后得出 P1 的数据为 00010010B。因此，只要将“00010010B”数据

送到 P1 并输出，就能达到设计目的。

3.1.3 用汇编语言编写程序

用汇编语言编写简单的单片机源程序 TEST.ASM，代码如下。

```
MOV P1, #00010010B
JMP $
END
```

这是一个非常简单的程序，只有 3 行指令。

第一行指令的作用是将数据“00010010”输送到 P1 端口输出，将接在 P1 端口 8 个发光二极管中的 6 个点亮。

第二行指令的作用是使程序运行在当前的输出状态下，即保持输出状态。

第三行指令程序是结束的标志，每个程序的最后都要用“END”结束。

上述程序在单片机集成开发软件上进行编写、编译/汇编和输出 Intel HEX 文件，再通过编程器将程序写入单片机，最后经过实验板上的实验，如果程序编写正确，将会在实验时看到发光二极管中的第 1、3、4、6、7、8 个灯被点亮。

3.2 用 MedWin 开发单片机程序

使用 MedWin 开发单片机程序，具体操作步骤如下。

3.2.1 编写源程序

双击 Medwin 启动图标，在启动后弹出的第一个对话框中选择“模拟仿真”进入 Medwin 工作窗口。接着选择窗口上方“文件”“新建”，填写“文件名”：TEST.ASM，单击“打开”按钮，在出现的编写程序窗口中编写程序，如图 3.2 所示。将写好的文件保存在一个目录中，例如 F:\C51，然后单击“保存”按钮。

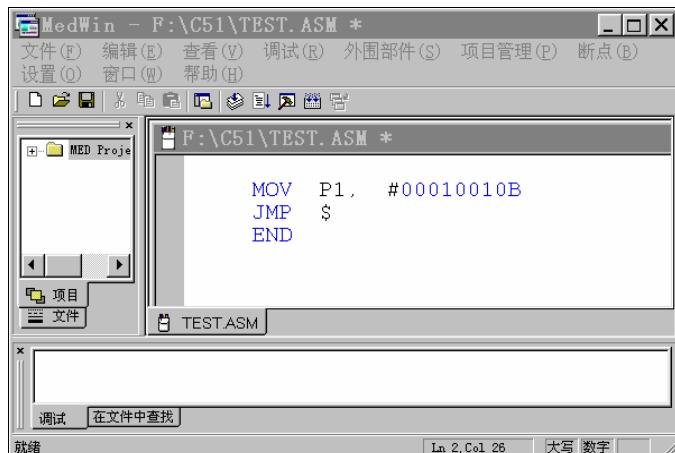


图 3.2 编程程序窗口

3.2.2 创建项目

选择窗口上方“项目管理”“新建项目文件”，在“创建项目”对话框中填写项目名为“TEST”，如图3.3所示。

接着单击“确定”按钮，在出现的“Add files to project”对话框中选择TEST.ASM文件，单击“打开”按钮，此时TEST.ASM文件便被添加到所创建的项目中。

3.2.3 编译/汇编

单击窗口上方“项目管理”，在出现的下拉菜单中选择“编译/汇编”选项，如图3.4所示。

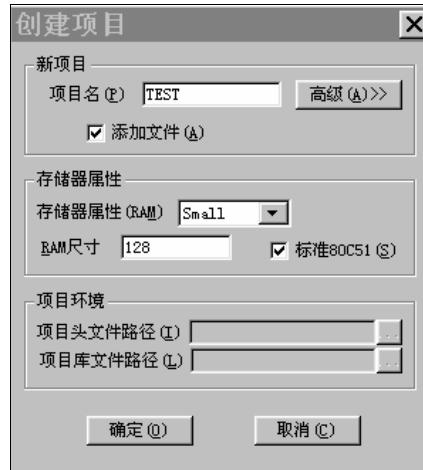


图3.3 “创建项目”对话框

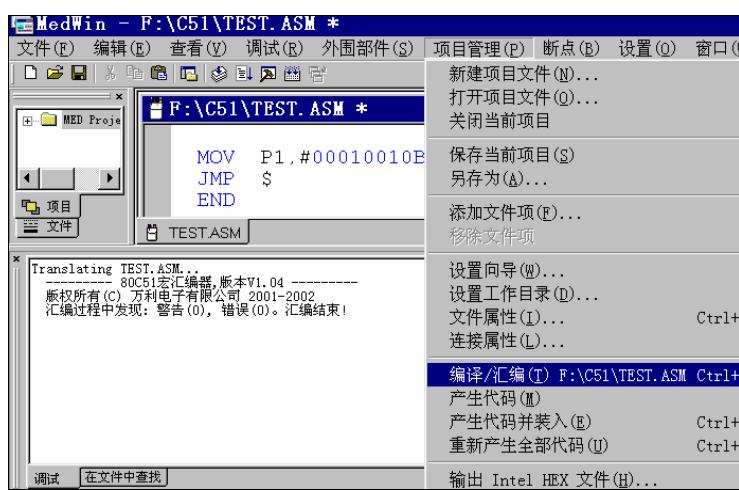


图3.4 “编译/汇编”窗口

单击“编译/汇编”按钮开始汇编，如果汇编成功，下方的状态窗口中会显示“汇编过程中发现：警告(0)，错误(0)。汇编结束！”，否则，将显示警告并指出错误所在的行。

3.2.4 输出Intel HEX文件

单击窗口上方“项目管理”，选择“输出Intel HEX文件”选项，在出现的“Export Intel HEX file”对话框中单击“保存”按钮，如图3.5所示。此时，程序完成了连接/定位，并输出TEST.hex文件。

如果使用Keil C51集成开发软件，请参见本书第17章。两个集成开发软件虽然具体操作有所不同，但主要步骤基本相同，均为编写源程序 建立工程项目 产生可执行的HEX文件。由于源程序相同，无论使用哪种集成开发软件，最后产生的可执行的HEX文件都是相同的。



图 3.5 输出 Intel HEX 文件

3.3 把目标文件写入单片机

通过集成开发软件生成的 TEST.hex 可执行文件，也称目标文件。目标文件需要使用编程器才能写入单片机，具体步骤如下。

3.3.1 选择单片机型号

先将 AT89C51 单片机插入编程器插座内，并锁紧（注意方向）。然后，启动编程器软件，单击编程器窗口上方的“器件”按钮，在出现的下拉菜单中选择 AT89C51 单片机型号，如图 3.6 所示。



图 3.6 选择单片机型号

3.3.2 进行擦除

对已经写过数据的单片机，在写入新文件前要对原数据进行擦除操作。如果需要保存原

数据，可在擦除前单击“读出”按钮将原数据读出，保存后再对单片机进行擦除操作，擦除后从单片机读出的数据是全 FFH。

3.3.3 写入文件

单击软件窗口左上方“文件”，选择“打开 HEX 文件”，在存放 HEX 文件的目录里找到需要写入单片机的文件（如 TEXT.hex 文件），单击该文件名后，软件工作窗口就会出现十六进制的 HEX 文件。

单击窗口上方“写入”按钮，被选中的文件就会被写入到单片机内，这时窗口的下方将显示“编程成功”的字样，如图 3.7 所示。



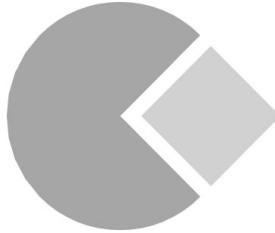
图 3.7 写入单片机

尽管编程器型号有所不同，但对单片机的写入操作基本相同，主要步骤为：选择单片机型号 擦除 选择文件 写入。

3.4 在实验板上实验

首先要检查实验板上的 LED 是否接在 P1 端口，假如 LED 不是接在 P1 端口而接在 P0 端口，则需要将程序中的 P1 改写为 P0，然后重新编译/汇编并生成 HEX 可执行文件后才能实验。

当程序所使用的端口与实验板上 LED 实际连接的端口一致时，就可以进行实验。将需要写入程序的单片机插入实验板的插座上。要注意单片机上的三角标志（1 脚标志），并对照实验板说明书，不能插错，否则会烧坏单片机。插好后并将其锁紧，再接通电源，就会看到发光二极管中的第 1、3、4、6、7、8 个 LED 被点亮。



第4章 单片机寄存器

单片机内有两个存储器，一个用于存放程序，称为程序存储器；另一个用于程序运行时暂时存放数据，称为数据存储器，也称寄存器。在寄存器中有一部分寄存器具有特殊功能，称为特殊功能寄存器。单片机的主要功能是通过向特殊功能寄存器写入0或1二进制数来实现的。

4.1 80C51 单片机引脚简介

80C51单片机共有40个引脚，其中有2个是电源引脚，2个外接晶体振荡器引脚，4个控制引脚，还有4个端口（P0~P3），即32个输入/输出引脚，如图4.1所示。

1. 电源引脚（2个）

- V_{cc}（40脚）：电源端，接+5V电源。
- V_{ss}（20脚）：接地端。

2. 外接晶振引脚（2个）

- XTAL1（19脚）和XTAL2（18脚）：接石英晶体振荡器。

3. 控制引脚（4个）

- RST（9脚）：复位信号引脚。当石英晶体振荡器运行时，在此引脚上出现两个机器周期的高电平，使单片机复位。
- ALE/PROG（30脚）：地址锁存允许信号端。当访问外部存储器时，ALE（允许地址锁存）的输出用于锁存地址的低8位。当不访问外部存储器时，ALE端仍以不变的频率周期性地输出脉冲信号，此频率为石英晶振振荡频率的1/6。因此，它可用作对外输出的时钟或用于定时。

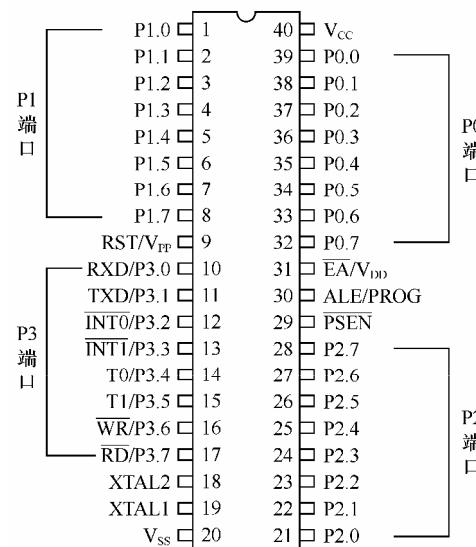


图4.1 80C51单片机引脚

在向单片机写入程序时，此引脚用于输入编程脉冲（PROG）。

- PSEN（29脚）：外部程序存储器的读选通信号端。在读外部ROM时，PSEN有效（低电平），以实现对外部程序存储器的读操作。
- EA/VPP（31脚）：访问程序存储器选择控制信号。当EA信号接低电平时，对ROM的读操作（执行程序）限定在外部程序存储器；当EA接高电平时，对ROM的读操作（执行程序）从内部开始。

4. 输入/输出引脚（32个）

80C51共有4个8位输入/输出（I/O）端口，分别为P0、P1、P2、P3。每个端口有8个引脚，共计32个引脚，每个引脚都可单独作输入或输出使用。

P0~P3的内部结构差别不大，但使用功能有所不同。

- P0端口：P0.0~P0.7（32~39脚）在外部扩充存储器时，可用作数据总线或地址总线；不扩充时，可用作一般I/O使用，但内部无上拉电阻，当作为输入或输出时应在外部接上拉电阻。
 - P1端口：P1.0~P1.7（1~8脚）只做I/O使用，其内部有上拉电阻。
 - P2端口：P2.0~P2.7（21~28脚）用作一般I/O使用，其内部有上拉电阻。在扩充外部存储器时，也可当作地址总线使用。
 - P3端口：P3.0~P3.7（10~17脚）。

除了用作I/O使用外（内部有上拉电阻），还有一些特殊功能，也称第二功能，如表4.1所示。

表4.1 P3端口的第二功能表

引脚	第二功能	功能说明
P3.0	RXD	串行数据接收端
P3.1	TXD	串行数据发送端
P3.2	INT0	外部中断0申请信号线
P3.3	INT1	外部中断1申请信号线
P3.4	T0	定时器/计数器0计数输入端
P3.5	T1	定时器/计数器1计数输入端
P3.6	WR	外部数据存储器写选通端
P3.7	RD	外部数据存储器读选通端

4.2 单片机工作的基本条件

单片机工作需要3个基本条件：接电源、接石英晶体振荡器和复位电路、单片机内装入程序，如图4.2所示。

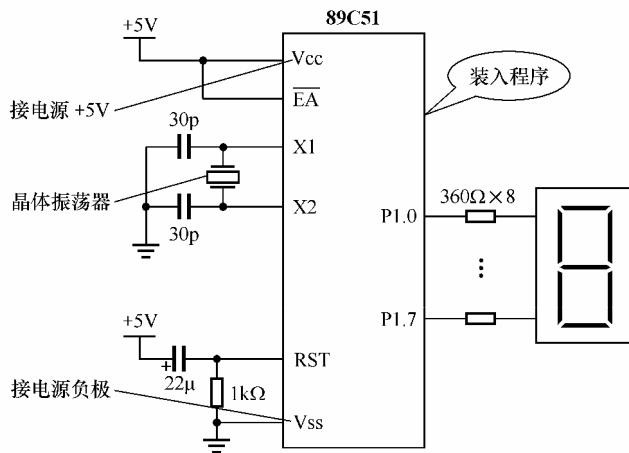


图 4.2 单片机工作基本条件

4.2.1 接电源

将单片机第 40 脚 Vcc 接电源+5V，第 20 脚 Vss 接地（电源负极），为单片机工作提供电源。

由于 80C51 片内带有程序存储器，当使用片内程序存储器时要将 EA（31 脚）接高电平，即接到电源+5V。

4.2.2 接石英晶体振荡器和复位

将单片机第 19 脚（XTAL1）与 18 脚（XTAL2）分别接外部晶体的两个引脚，由石英晶体组成振荡器，保证单片机内部各部分有序工作。

复位信号引脚 RST（9 脚）与 Vss 之间连接一个下拉电阻，与 Vcc 引脚之间连接一个电容，目的是保证可靠的复位。

单片机运行程序的速度与振荡器的频率有关。单片机在读、写操作时都需要消耗一定的时间。机器周期是指单片机完成一个基本操作所用的时间，当外接石英晶体为 12MHz 时，1 个机器周期为 1ms；当外接石英晶体为 6MHz 时，1 个机器周期为 2ms。

4.2.3 单片机内装入程序

单片机只有在装入程序后才能工作。例如，图 4.2 所示的单片机的 P1 端口接有一位数码管，数码管能显示数字，其显示方式有静态显示和动态显示，显示方式的设定无需改变外电路，由装入单片机内的程序所决定。如果单片机内没有装入程序，尽管单片机的外部电路连接都正确，数码管也不会显示数字。

因此，学习单片机开发的关键是编写单片机程序。

4.3 单片机的存储器

单片机的体积虽小，但“五脏俱全”，结构非常复杂，它是普通计算机的缩小版。不过

初学者可以把它看成是一个“黑匣子”，只需知道它的内部有两个存储器，即程序存储器和数据存储器，如图 4.3 所示。

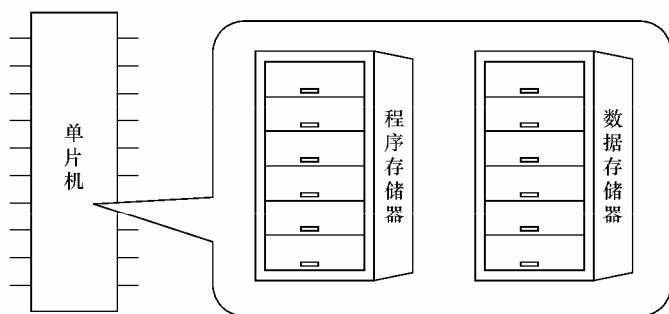


图 4.3 单片机内有两个存储器示意图

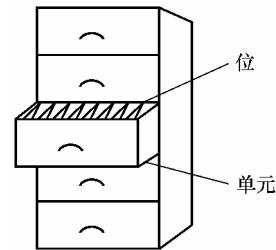
存储器的作用可以类比为我们生活中的大书柜或是我们出门时在寄存处临时寄存东西时看到的竖柜等。单片机内两个存储器一个是存放单片机程序的地方，称为程序存储器；另一个是程序运行中暂时存放数据的地方，称为数据存储器。

4.3.1 单元与位

存储器分若干“层”，“层”在存储器中称为单元。每层还有“隔”（8个或16个），“隔”在存储器中称为位（bit），如图 4.4 所示。

位是计算机所能表示的最基本、最小的数据单元。因为计算机采用二进制数，所以位就是1个二进制位，它有两种状态：0和1。

4.3.2 字、字节和数制



1. 计算机中的数制

图 4.4 单元与位示意图

人们最常用的是十进制数，而在计算机中数是采用二进制表示的。但是，二进制数书写起来太长，且不便阅读和记忆，所以在计算机中一般采用十六进制数来表示。十进制数、二进制数和十六进制数之间可以相互转换，它们之间的关系如表 4.2 所示。

表 4.2 十进制数、二进制数及十六进制数对照表

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

为了区别上述 3 种数制，可在数的后面加一个字母，B 表示二进制数制，D 或不带字母表示十进制数制，H 表示十六进制数制。

例如， $15=1111B=0FH$ 。注意，十六进制数只有一位是字母时，书写时要在字母前加 0，如 $0FH$ ，其中 H 表示十六进制数制，F 表示 15，书写时 F 前要加 0。

在 C 语言编程中，十六进制数还有一种常用的表示方法：即在数的前边标有“0x”表示是十六进制数，例如， $15=0xf$ 。

2. 字 (word) 和字长

字是计算机内部进行数据处理的基本单位。通常它与计算机内部的寄存器、算数逻辑单元、数据总线宽度相一致。计算机的每一个字所包含的二进制位数称为字长。

3. 字节 (byte)

把相邻的 8 位二进制数称为字节。由于计算机只能识别和处理二进制数，因此在计算机内部把所有的数据，如字母、数字、特殊符号等都用二进制代码表示。

二进制代码常用到的是 ASCII 码，它的长度为 8 位，其中低 7 位表示字母本身的编码，例如，大写字母 W 用二进制数表示为 $1010111B$ ，第 8 位用作奇偶效验位或规定为 0。80C51 数据存储器每个单元是 8 位，正好能放入 8 位二进制数，即 1 个字节。字节的多少代表着存储器的容量大小。

4.3.3 程序存储器

程序存储器主要用于存储程序，其最大特点是电源关掉后，所存储的程序不会消失，像计算机中的硬盘一样。80C51 程序存储器在片内有 4KB，使用片内存储器时要将单片机 EA（第 31 引脚）接高电平，即接到电源+5V。如果片内容量不够时，可在片外安装存储芯片扩展 60KB，如图 4.5 所示，使程序存储器（片内加片外）达到 64KB。

程序存储器是十六位的，用 4 位十六进制数来表示地址。其中片内 4KB 的地址范围是 $0000H \sim 0FFFH$ ，片外 60KB 的地址范围是 $1000H \sim FFFFH$ 。

程序存储器中有特殊用途的地址，将在中断程序章节中再加以说明。



图 4.5 程序存储器配置

4.3.4 数据存储器

数据存储器是程序运行中暂时存放数据的地方，也称为寄存器。其特点是存储内容会随着电源的关闭而消失，像计算机中的内存一样。

数据存储器是 8 位存储器，一个单元是一个字节，片内部有 256 字节（可以想象为一个 256 层的存储柜，每层 8 个隔），地址范围用十六进制数可表示为 $00H \sim FFH$ 。

图 4.6 为片内数据存储器的配置示意图。可分两部分，其中，低 128 字节（ $00H \sim 7FH$ ）为一般用途寄存器区；高 128 字节（ $80H \sim FFH$ ）为特殊功能寄存器区，如图 4.6 (a) 所示。

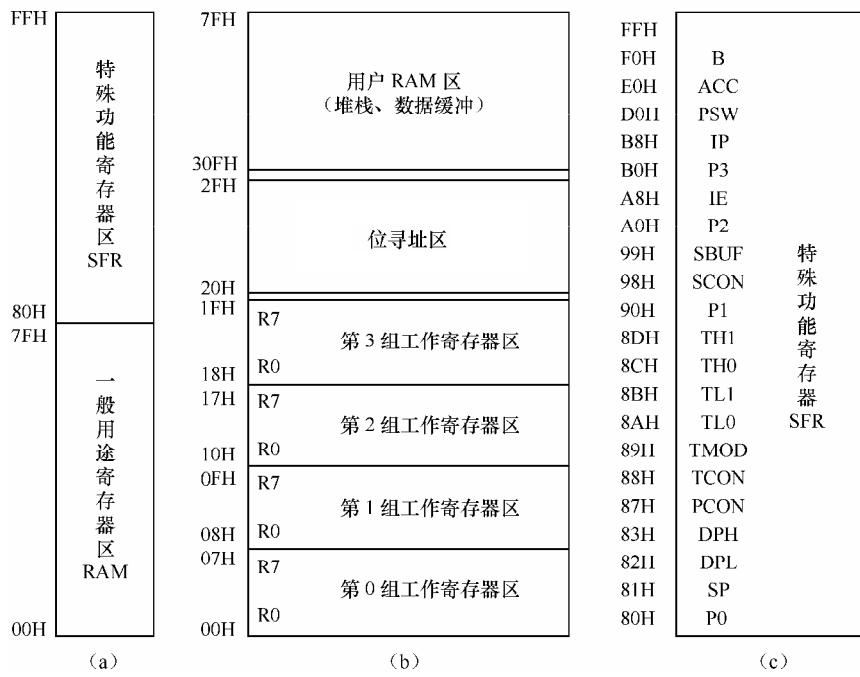


图 4.6 片内数据存储器的配置示意图

1. 一般用途寄存器区

一般用途寄存器区 RAM 的容量为 128 字节，根据用途可划分为工作寄存器区、位寻址区和用户 RAM 区，如图 4.6 (b) 所示。

- 工作寄存器区

在低 128 字节中，00H ~ 1FH 共 32 个单元（字节）是工作寄存器区，又分为 4 组，每组由 8 个单元组成，分别用 R0 ~ R7 作为这 8 个单元的寄存器名。

在单片机复位后，选中的是第 0 组工作寄存器。每组寄存器均可选作 CPU 当前工作寄存器，可以通过 PSW 状态字中 RS1、RS0 的设置来改变 CPU 当前使用的工作寄存器。

- 位寻址区

低 128 字节中的 20H ~ 2FH 共 16 个单元是位存储区，可用位寻址方式访问其各位。

- 用户 RAM 区

低 128 字节中的 30H ~ 7FH 共 80 个单元是用户 RAM 区，用作堆栈或数据缓冲。

2. 特殊功能寄存器区

特殊功能寄存器，简称 SFR。它在单片机中扮演着非常重要的角色，使用输入/输出、中断、串行口、计时/计数等功能，都必须先设置 SFR 中的各相关寄存器。

特殊功能寄存器的地址范围为 80H ~ FFH，如图 4.6 (c) 所示，其中包括如下所列的寄存器。

- 累加器 ACC (A)
- B 寄存器。

- 程序状态字组 PSW。
- 数据指针寄存器 DPTR。
- 堆栈指针寄存器 SP。
- P0、P1、P2、P3 端口寄存器。
- 中断允许控制寄存器 IE。
- 中断优先权 IP 寄存器。
- 计时/计数模式寄存器 TMOD。
- 计时/计数器控制/状态寄存器 TCON。
- 串行通信控制寄存器 SCON。
- 串行数据寄存器 SBUF。
- 电源控制及数据传输率选择寄存器 PCON。

这些寄存器的功能将在以后各相关章节中作详细介绍。其中，常用的有 P0、P1、P2、P3 端口寄存器及累加器 A 等。

4.4 单片机工作的基本原理

单片机的功能主要是通过向特殊寄存器输送 0 或 1 二进制数来实现的，所以，了解 0 和 1 的作用对理解单片机工作的基本原理非常重要，下面就以一个程序实例来说明输入/输出端口的工作原理。

程序 TEST.ASM 是一个实例程序，图 4.7 是它的电路图，在图中单片机 P1 端口的 8 个引脚分别与 8 只发光二极管相连。

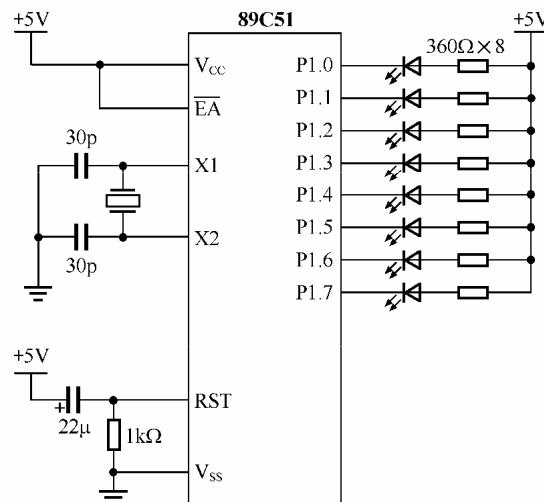


图 4.7 P1 端口输出电路

如果将该程序写入单片机后，在程序运行时会看到 8 个发光二极管中接在 P1.0、P1.2、P1.3、P1.5~P1.7 的发光二极管被点亮；而 P1.1 和 P1.4 所接的发光二极管熄灭。

程序：TEST.ASM

```

MOV P1, #00010010B
JMP $
END

```

4.4.1 引脚与寄存器的关系

程序中的 P1 对外是输入/输出端口，共有 8 个引脚，分别与 8 只发光二极管相连；而 P1 对内是特殊功能寄存器里的一个单元，地址在 90H，单元内共有 8 个位，每位接出一根引线，就是在外面看到的引脚 P1.0 ~ P1.7，P1 寄存器与引脚的对应关系示意图如图 4.8 所示。

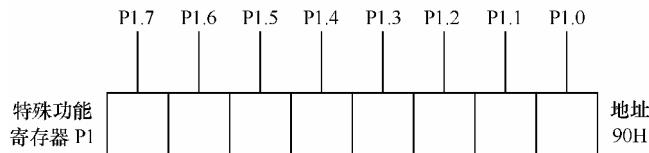


图 4.8 P1 寄存器与引脚的对应关系示意图

单片机的 P0、P2、P3 与上述的逻辑结构相同。

4.4.2 单片机中 0 和 1 的作用

1. 在端口的外部 0 与 1 是表示电平的高低

图 4.9 所示是图 4.7 P1 端口输出的等效电路，D 是发光二极管，当正向连接有电流流过时，二极管就会发光。R 是限流电阻，防止二极管因电流过大而被烧坏。

从电路中可以看出，当开关 K 向下与 b 点相接时，二极管的负极接地，即 P1 端口输出呈现低电平，用 0 来表示，此时二极管因有电流通过而发光；当开关 K 向上与 a 点相接时，二极管的负极接电源 +5V，即 P1 端口输出呈现高电平，用 1 来表示（此时的 1 不是 1V，而是代表 +5V），此时二极管被截止，没有电流通过，也就不能发光。

所以，当 P1 引脚为低电平 0 时，灯会点亮；当 P1 引脚为高电平 1 时，灯会灭。0 和 1 在端口的外部是表示电平的高低。

2. 在单片机内部对寄存器而言，0 和 1 是表示二进制数

要想使输出端引脚呈现低电平或高电平，只需要向寄存器 P1 输入相应的二进制数 0 或 1 即可。例如，通过使用单片机的 MOV 指令（MOV 指令的功能是输送数据），把 8 位二进制数 000010010 送到特殊功能寄存器 P1 单元，程序为：MOV P1, #00010010B，如图 4.10 所示。

0 所对应的引脚是低电平，发光二极管导通发光；1 所对应的引脚是高电平，发光二极管截止，灯灭。

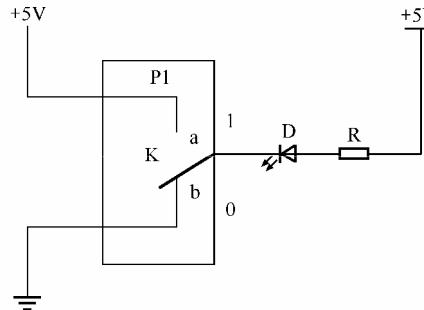


图 4.9 P1 端口等效电路

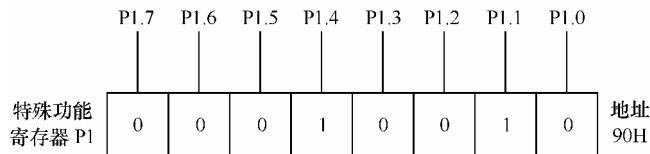


图 4.10 装入 8 位二进制数

3. 对连接的发光二极管来说，0 和 1 表示灯亮与灭

这是指发光二极管按图 4.7 接法而言，不是绝对的。因大部分单片机电路都是按此接法，所以记住此规律对分析单片机电路是很有益的。

从上述可归纳出：0 和 1 在寄存器内表示二进制的数；0 和 1 在输出端口处表示电平的高低；0 和 1 又代表灯亮与灭（导通与截止）。

4.4.3 工作基本原理

单片机通过执行程序中的指令，向特殊功能寄存器各位内放入 0 或 1，就能在输出端口获得相应的输出信号，如图 4.11 所示。

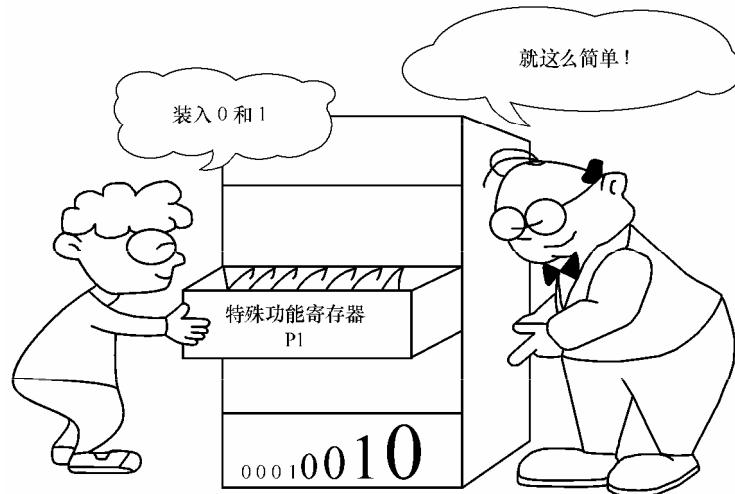


图 4.11 工作原理示意图

也都是说，如果把输入/输出端口的引脚看成是一个个智能开关的话，这些开关就会按照人们的意愿（通过装入 0 或 1 来告诉单片机）自动开与关，从而自动控制所连接的各电器设备。



第5章 单片机指令

汇编语言是由单片机系统指令和伪指令所构成。80C51 单片机的指令系统共有指令 111 条，但是常用的只有三十几条。初学时首先熟悉出现频率比较高的指令，然后再逐步扩充，并在使用中加深对指令的理解，在编程中学指令。

5.1 学习单片机指令与编程的经验与技巧

单片机是通过执行程序工作的，而编写程序使用的汇编语言又是由系统指令和伪指令所构成。因此要逐渐掌握各类指令的功能，不断学习编程技巧，学习单片机指令与编程的经验与技巧如下。

1. 首先掌握出现频率比较高的指令

80C51 单片机的指令系统常用的只有三十几条，在三十几条中，可先掌握几条出现频率比较高的指令。

如指令 MOV 是传送的意思，属于数据传送类指令，出现频率最高，并且每个程序里都会多次出现。

伪指令 END 是结束的意思，每个程序最后都必须使用 END，表示程序结束。

只要先掌握了几条出现频率比较高的指令，就能看懂简单的程序，然后再逐步扩充。

2. 在使用中加深对指令的理解

指令的解释往往会有比较抽象，一时不懂不要紧，可以在动手反复实验的过程中去慢慢体会它，加深理解。所以学习指令不必硬背，要在用中学。

3. 对具有显著特征的语句要重点去记

如“MOV P1,A”，它是单片机 P1 端口的输出语句。在复杂的程序中先找到输出语句，便于理出程序的头绪，因为编写程序的最终意图，一般都体现在输出端口上。

4. 注意程序的模块化设计与分析

在程序设计时，一般是将一个复杂工程分解若干个模块，模块是具有单一功能，并具有

相对独立性的部分。分解后的各模块比较小，既容易编写，又容易调试。然后再把各模块有机地联系在一起，便组成一个大程序。

所以，我们在分析一个大程序时，首先要弄清它是由几个模块所组成，每个模块主要功能是什么，模块之间是怎样联系在一起的。这样先有个粗线条，有个总体轮廓，然后再逐步深入分析，即使模块中有的语句一时没能弄清楚也不要紧，可以通过模拟仿真，在实验板上实验，不断加深理解。

每个子程序就是一个模块，子程序段的特点是：开始行有程序标号，以便主程序调用；末尾行有子程序返回指令 RET 或 RETI。

5. 掌握程序的 3 种基本结构特点

任何复杂的程序都可以看成是一个个基本程序结构的组合。基本程序结构可归纳为 3 种：顺序结构、选择结构、循环结构。在这里只是简单地介绍一下，希望读者在学习中逐步加深体会。

(1) 顺序结构程序。

顺序结构程序是指一种无分支的直线程序，即从第一条指令开始依次执行每一条指令，直到最后一条结束。顺序结构程序是一种最简单、最基本的程序，是构成复杂程序的基础。

(2) 选择结构程序。

选择结构程序利用条件转移指令，根据条件是否满足来改变程序执行的次序。选择结构使程序智能化，是设计程序和分析程序的重要部分。

选择结构的关键是如何判断分支条件，指令系统中可以直接用于判断分支条件的指令有：累加器判零条件转移指令 JZ(JNZ)，比较条件转移指令 CJNE 和位条件转移指令 JC(JNC)，JB (JNB)，JBC 等。

通过这些指令，可以完成各种各样的条件判断。但是，每执行一条判断指令，只能形成两路分支，若要形成多路分支，就要进行多次判断。

(3) 循环结构程序。

循环程序也是常用的一种程序结构形式。当在程序设计中需要某一段程序重复执行多次时，可采用循环结构，如软件延时程序就是典型的循环结构程序。

掌握程序的 3 种基本结构特点，会为以后分析程序、编写程序打下坚实的基础。

6. 利用模拟仿真工具加深对程序的理解

模拟仿真不但能够解决程序实际运行中的一些问题，而且能够清楚地看到每条语句在单片机内运行的情况，是初学者理解指令、学习编程、了解单片机工作原理非常有用的工具。

本书汇编语言的编程实例都介绍了模拟仿真的过程。

5.2 单片机编程语言概述

5.2.1 编程语言概述

所谓编程，就是用计算机能识别的语言告诉计算机去做什么、怎样去做。计算机看起来

神通广大、无所不能，但实际上它做的每一项工作，都需要人事先用计算机能识别的语言详细地告诉它第一步做什么、第二步做什么，这种语言称为计算机程序设计语言。

计算机程序设计语言按结构及其功能可以分为3种：机器语言、汇编语言和高级语言，下面分别对其进行介绍。

1. 机器语言

机器语言是由二进制码“0”和“1”组成的能够被计算机直接识别和执行的语言。用机器语言编写的程序，称为机器语言程序或目标程序。

例如，我们让程序完成的任务是将内部RAM中的第20H单元中内容和第21H单元中内容相加，用机器语言编写的程序为：

```
11100101 00100000
00100101 00100001
```

很显然，用机器语言编写的程序不易阅读，难记、难学。

2. 汇编语言

汇编语言是以人们易于理解和记忆的英文名称或缩写形式（助记符）来表示二进制指令。例如用MOV在程序里代表传送数据。ADD在程序里代表相加。上述机器语言程序可用汇编语言改写为：

```
MOV A, 20H
ADD A, 21H
```

第一句用了MOV指令，意思是将第20H单元中内容传送到累加器A中；第二句用了ADD指令，意思是将第21H单元中内容与累加器A中的内容相加。汇编语言比机器语言更容易理解、记忆和交流。

用汇编语言编写的程序称为汇编语言程序，或称源程序。但是，计算机不能直接识别和执行汇编语言程序，必须经过一个翻译过程，即把汇编语言程序译成机器语言程序计算机才能执行，这一翻译工作又称为汇编，汇编一般是借助专用软件由计算机自动完成。汇编后的机器语言程序也称目标程序。

汇编语言和机器语言随机型不同而异，一般互不通用的。

3. 高级语言

高级语言接近于人类自然语言，用高级语言编写的程序与人们通常解题的步骤比较相近，而且不依赖于计算机的结构和指令系统，是面向过程而独立于机器的通用语言。

用高级语言编写的源程序，也需要翻译生成目标程序机器才能执行。高级语言的特点是易学、通用性好、便于移植。

5.2.2 单片机使用的编程语言

单片机本身就是微型计算机，所以，编写单片机程序也需要用计算机程序设计语言。其中汇编语言是编写单片机程序的常用语言，用汇编语言编写单片机程序的特点是占用资源少、

运行速度快。

同时，初学者使用汇编语言还有利于加深对单片机硬件的了解，为了打好基础，应该先学习和掌握用汇编语言编写单片机程序。

单片机的汇编语言是由单片机的指令和汇编伪指令所组成，不同系列单片机的指令是不同的，所以，单片机汇编语言不能通用。但是掌握了一种机器的汇编语言后，学习其他机器的汇编语言就会很简单了。

单片机编程也可以使用 C 语言。C 语言是高级语言，它具有通用性好、便于移植的特点。所以，在掌握了汇编语言编程后，需要进一步学会使用 C 语言编程。

5.2.3 80C51 汇编语言的语句结构

80C51 单片机汇编语言是由 80C51 系统指令和伪指令所组成。其语句结构的一般形式如下：

```
标号: 操作码(指令) 操作数 ;注释
START: MOV A, 20H ;A (20H)
```

它是由 4 部分组成，即标号、操作码、操作数和注释。其中，标号和操作码之间必须用冒号“：“作分隔符隔开，操作码与操作数之间要用空格作分隔符；操作数内部要用逗号“，”将源操作数和目的操作数隔开；注释段与操作数之间要用分号“；”隔开。

1. 标号

标号用来给一个语句定义一个名字，一般第一个字符用英文字母，第二个以后的字符用英文或数字 0 ~ 9 组成均可，组成标号的字符最多为 6 个。但是，不能使用指令助记符、伪指令或寄存器名作标号，以免引起混淆。

2. 操作码

操作码用来指定计算机完成某种操作，是每一个语句不可缺少的核心部分，操作码可以是指令助记符或伪指令。

3. 操作数

操作数是操作码操作的对象，它是参加操作的数（也称立即数）或是操作数据所在的地址（也称直接地址）。当语句中出现一个以上的操作数时，要用逗号分隔开。

在操作数中，注意立即数与直接地址在书写上的区别，例如：

```
MOV A, 8H
MOV A, #8H
```

两条语句中“8H”与“#8H”概念是不同的。“8H”是指第 8 个单元，即将第 8 个单元中存储的数据放入累加器 A 中；“#8H”是数，即把立即数 8 放入累加器 A 中。

4. 注释

注释是程序设计者对该行程序语句功能所作的说明，以便阅读和交流。当语句中需注释

时必须以“ ;”开头，如果注释的内容超过一行，则换行后前面还要加上分号。

这样在汇编时，分号后面的注释部分就不会被翻译成机器码，也就是说，注释内容无论长短都丝毫不会影响程序运行的结果。

5.3 80C51 单片机指令系统

在用汇编语言编写程序时，数据的存放、传送运算都是通过指令来完成的，所以，必须逐步了解和熟悉指令。这一节仅对 80C51 系列单片机的指令系统做简单介绍，每一条指令的具体应用，将在以后各章节结合编程应用中来详细讲解。

80C51 系列单片机的指令共有 111 条，有 42 种助记符，按功能可分为算术运算类指令、逻辑运算及移位类指令、数据传送类指令、控制转移类指令和位数据传送类指令。

为了方便查找，本书将各类指令列表说明，详见书后附录 A《80C51 指令速查表》。

5.3.1 数据传送类指令

数据传送类指令是最常用的一类指令，共有 29 条。其作用是把源操作数传送到指令所指定的目标地址处。数据传送指令用到的助记符有 MOV、MOVC、MOVX、XCH、XCHD、SWAP、PUSH、POP 等 8 种。附录 A 中表 5.2 为数据传送类指令表。

数据传送类指令的基本功能是传送数据。例如，单片机的 P1 端口接有 8 个发光二极管，如果让中间 4 只亮，需要把一组二进制数 11000011（0 代表亮）送到 P1 寄存器（P1 对内是寄存器，对外是输入/输出端口），使用下面的数据传送指令：

```
MOV A, #11000011B
MOV P1, A
```

第一句是将 11000011 送到寄存器 A 中暂时存放；第二句是将 A 中的数据，即 11000011 再送到寄存器 P1 中输出，用了两次 MOV 指令完成了数据的传送任务。MOV 指令是编写程序时使用频率最高的指令。

5.3.2 控制转移类指令

控制转移类指令共有 17 条，可进一步细分为无条件转移指令、有条件转移指令、子程序调用指令等，主要功能是改变程序的执行顺序。控制转移指令用到的助记符共 10 种：ACALL、AJMP、LCALL、LJMP、SJMP、JMP、JZ、JNZ、CJNE、DJNZ。附录 A 中表 A.3 控制转移类指令表。

这类指令的基本功能是将程序运行方向转移，即改变程序的执行顺序，如程序：

```
01: STRTY: .....
02: LOOP: .....
.....
10:           ACALL  DELAY
11:           DJNZ  R0, LOOP
```

```

12:      JMP     START
        .....
30:  DELAY:  .....

```

程序运行时如果不遇到转移语句，程序会按照编写时语句排列顺序依次向下运行，即执行第一条语句后，接着执行第二条语句，依次向下执行。

程序中的 10、11、12 这 3 条语句都是控制转移类指令，它们的执行不再按照书写顺序，而是按照转移指令所指的程序标号（STRTY、LOOP、DELAY 是程序行的标号）去执行标号处的语句。

第 10 行语句中的 ACALL 是调用子程序指令，执行该语句后，程序将转移到第 30 语句调用延时子程序执行，直到运行完子程序后才能返回到第 11 行语句。

第 11 行语句是有条件转移语句（DJNZ 是有条件转移指令），在 R0 不为 0 时，会转移到第 02 行语句处去执行。

第 12 行语句中的 JMP 是无条件转移指令，执行该语句后程序将转移到标号 STRTY 处，即到第 01 行语句处去执行。

在控制转移类指令中，ACALL、DJNZ 和 JMP 是经常用到的指令。

5.3.3 逻辑运算及移位类指令

逻辑运算及移位类指令共有 24 条，主要用于对 8 位数进行逻辑与、或、异或、取反、清 0 以及循环移位。逻辑运算及移位类指令用到的助记符共 9 种：CLR、CPL、ANL、ORL、XRL、RL、RLC、RR、RRC。附录 A 中表 5.4 为逻辑运算及移位类指令表。

5.3.4 算数运算类指令

算数运算类指令共 24 条，主要是对 8 位无符号二进制数进行加、减、乘、除四则运算以及加 1、减 1、压缩 BCD 码加减运算等操作。算术运算指令用到的助记符共 8 种：ADD、ADDC、SUBB、INC、DEC、DA、MUL、DIV。附录 A 中表 A.5 为算数运算类指令表。

5.3.5 位操作类指令

位操作类指令共有 17 条，是专门对位地址空间进行操作的指令。位操作指令包括位传送、逻辑运算、控制转移等。所用的助记符有 11 种：MOV、CLR、CPL、SETB、ANL、ORL、JC、JNC、JB、JNB、JBC。附录 A 中表 A.6 为位操作类指令表。

位操作类指令的特点是以位为单位进行运算和操作。例如，如果需要点亮 P1 端口所接的发光二极管中第一只 LED，可使用下面指令之一：

```

指令 1：
MOV  A, #11111110B
MOV  P1, A
指令 2：
CLR  P1.0

```

两种操作都使第一只发光二极管点亮。不同的是前面语句是对 P1 寄存器单元（单元包括 8 个位）进行操作；后面语句中 CLR 指令的功能是对位清 0，即对 P1 寄存器单元中右边第 1 位（P1.0）进行置 0 操作。

5.4 常用的伪指令

编写单片机源程序用到的指令既包括 80C51 指令系统中的指令，也包括一些伪指令。

5.4.1 伪指令与 80C51 指令的不同点

1. 伪指令是在汇编过程中起作用的指令

我们知道，计算机只认识机器语言，因此在应用系统中必须把汇编语言源程序通过汇编程序翻译成机器语言程序（目标程序），这个翻译过程称为汇编。

汇编程序在汇编过程中，必须要提供一些专门的指令，用来对汇编过程进行某种控制，或者对符号、标号赋值等，也就是说伪指令是在汇编过程中起作用的指令，所以又叫汇编控制指令。

2. 伪指令不产生可执行的目标代码

同样是指令，80C51 指令通过汇编后将产生机器可执行的目标代码；而伪指令在汇编过程中只是协助程序的编译工作，并不产生可执行的目标代码，不占用存储器地址，所以称伪指令。

不同版本的汇编语言中伪指令的符号和含义有所不同，但基本的用法是相似的。

5.4.2 常用的伪指令

常用的伪指令有：起始伪指令 ORG、结束伪指令 END、定义字节伪指令 DB、定义数据字伪指令 DW、定义内存空间伪指令 DS、等值伪指令 EQU、位地址符号伪指令 BIT 和表示目前的地址伪指令\$等。

1. 起始伪指令 ORG

ORG 伪指令的功能是规定源程序或数据的起始地址，一般在汇编语言源程序或数据块的开始都用一条 ORG 伪指令，规定程序存放的起始位置，如：

```
ORG    30H
START: MOV    A, #10101010B
```

以上代码表示 START 标号地址等于 30H，即汇编源程序从地址 30H 开始存放。在一个源程序中，可以多次使用 ORG 伪指令，以规定不同程序段的起始位置，但所规定的地址应从小到大，不允许有重叠。

一个源程序若不用 ORG 指令开始，则会自动从地址 0000H 开始存放目标码。

2. 结束伪指令 END

结束伪指令 END 的功能是告诉汇编程序汇编到此结束，即使 END 后边还有汇编语言源程序也不参加汇编，不生成机器码。

一个源程序都必须有一个结束伪指令 END，而且只能是一个，放在程序行的最后面。

3. 定义字节伪指令 DB

定义字节伪指令 DB 的功能是规定从程序存储器的某地址单元开始，存入一组 8 位二进制常数，如：

```
ORG    1000H
TABLE: DB      0C0H,    0F9H,  0A4H,  0B0H
```

以上代码表示 TABLE 的地址等于 1000H，将 0C0H 存入 1000H 地址单元，0F9H 存入 1001H 地址单元，0A4H 存入 1002H 地址单元，0B0H 存入 1003H 地址单元。

4. 定义数据字伪指令 DW

DW 伪指令的功能是按字的形式（双字节），把数据存放在存储单元中，使用方法与 DB 类似。区别是 DB 指令是存入 8 位数据表；而 DW 指令能存入 16 位数据表，如：

```
ORG    1000H
TABLE: DW      0001H,  0020H
```

以上代码表示将 0001H 存入 1000H 和 1001H 地址单元，将 0020H 存入 1002H 和 1003H 地址单元，每一项需要占用两个存储单元。

5. 等值伪指令 EQU

EQU 伪指令的功能是将一个常数或特定的符号赋值给规定的字符串，如：

```
B1    EQU    #0FEH
MOV    A,    B1
```

这里将 B1 等值为常数#0FEH，在程序中 B1 就可以代替#0FEH 使用。

使用 EQU 伪指令时要注意必须先赋值后使用，而不能先使用后赋值。

6. 定义存储区伪指令 DS

定义存储区伪指令 DS 的功能是从指定地址开始，保留一定数量的储存单元，如：

```
ORG    30H
DS     8H
```

以上代码表示从 30H 地址单元开始留出 8 个字节的存储单元作为他用。

7. 位地址符伪指令 BIT

BIT 伪指令的功能是将位地址赋予所规定的字符名称，如：

```
A1    BIT    P1.0
```

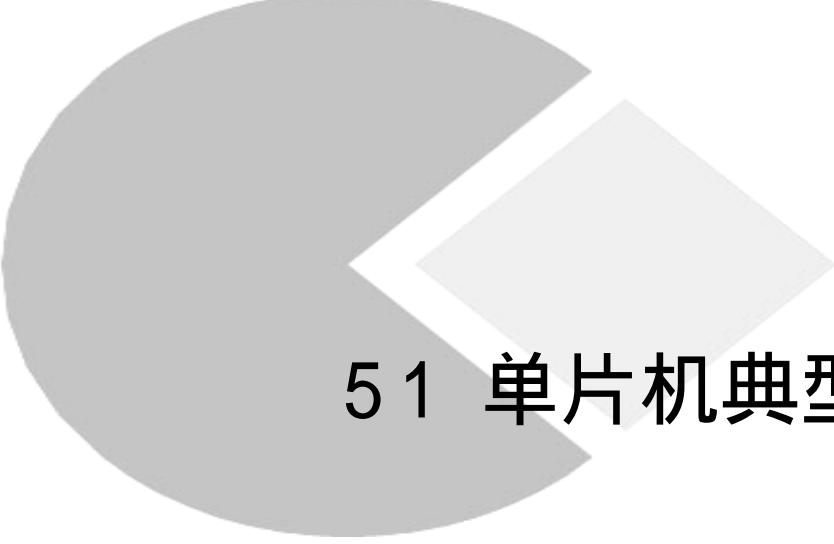
以上代码表示 A1 为 P1.0 位的地址。

8. 表示目前的地址伪指令\$

伪指令\$表示目前的地址，程序计数值不变，如：

```
MOV    P1,    #00010010B  
JMP    $
```

该伪指令\$的作用是使程序运行在当前的输出状态下。



5.1 单片机典型实例篇

本篇在经典实例中边学指令边学编程，逐渐了解和掌握单片机的 5 大基本功能：输入功能、输出功能、中断功能、定时器功能和通信功能。

第 6 章 跑马灯



单片机的功能最终要体现在输出端，输出端通过输出适当的控制信号去控制特定设备。本章通过实例程序并用 LED 演示来讲述单片机的基本输出功能。本章中各节所用的硬件电路均相同，故硬件设计只在 6.1 节中介绍，其余各节不再赘述。

6.1 点亮一只灯

功能说明：控制单片机 P1 端口输出，使 P1.0 位所接的 LED 点亮，其他 7 只灯熄灭。

6.1.1 硬件设计

本例中单片机电路如图 6.1 所示。

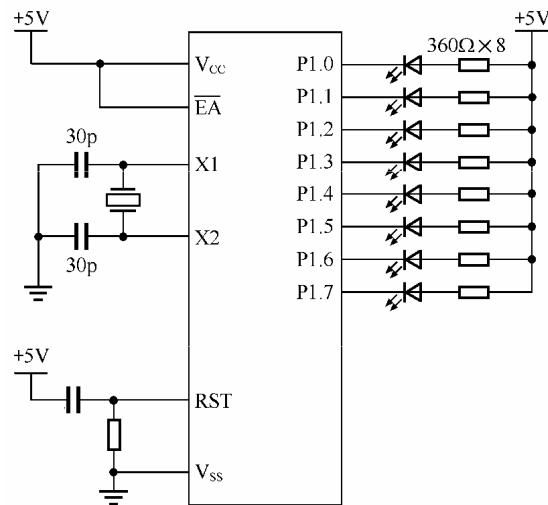


图 6.1 输出口应用实验电路

将 51 单片机第 40 脚 Vcc 接电源+5V，第 20 脚 Vss 接地，为单片机工作提供能源；再将第 19 脚 XTAL1 与 18 脚 XTAL2 分别接外部晶体两个引脚，由石英晶体组成振荡器，保证单片机内部各部分有序地工作。

将 8 个发光二极管(简称 LED)分别接在单片机 P1 端口的 P1.0 ~ P1.7 引脚上,注意 LED 有长短两个引脚,分别表示正负极,其中较短的负极与单片机引脚相连,较长的为正极,通过限流电阻 R 与 Vcc 相连。

如果是购买的实验板,实验电路已经连好,只要注意实验板上单片机插座所使用的端口是否和程序指定的端口一致,如果不一致,要将程序中指定的端口改为与实验板上所使用的端口,单片机的 P0、P1、P2、P3 端口都可以用来控制 LED。

6.1.2 程序设计

1. 流程图

程序设计流程如图 6.2 所示。

2. 程序

汇编语言编写的点亮一只灯源程序 JS01.ASM, 代码如下:

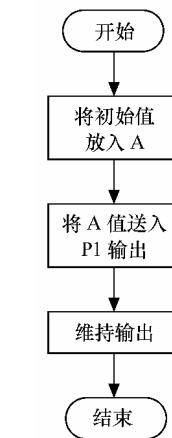


图 6.2 程序流程图

01:	MOV	A, #11111110B	; 存入欲显示灯的位置数据
02:	MOV	P1, A	; 点亮第一只灯
03:	JMP	\$; 保持当前的输出状态
04:	END		; 程序结束

6.1.3 代码详解

上述程序可以在 MedWin 或 Keil 集成开发系统上进行编写。请注意,每行前边的序号,如 01、02、03、04 不要写,后面的解释部分可写也可省略。此外,还要注意,如果是在 Word 上编写程序,当复制到 MedWin 上的时候,解释前边的分号“;”需要重写,因为在 Word 里键入的分号“;”可能是中文的,汇编时会出错。

1. 寄存器使用分配情况

程序中 A、P 为特殊寄存器。

A 累加器(Accumulator),又称为 A 寄存器。它是 CPU 工作中使用频率最高的寄存器,大多数指令的操作都需要通过累加器 A 来实现。该寄存器位于数据寄存器的第 E0H 层,用临时寄存一个操作数或运行中间结果。在本程序中 A 寄存器寄存的是 P1 端口欲显示的 LED 灯的位置,即数据 11111110B。

端口 P1 对外是输入/输出端口,对内也是寄存器,位于数据寄存器的第 90H 层,所寄存的二进制数直接控制着灯的亮与灭。如寄存的数是 11111110B,则右数第 1 个灯亮,如果是 11111101B,则右数第 2 个灯亮,0 表示亮。

2. 程序分析解释

01: 在程序第一行语句里,通过数据传送指令 MOV(功能是传送数据)将二进制数 #

11111110B 传送到寄存器 A 中寄存，0 位表示灯亮。可以看出，要被点亮的是第 1 只灯。数 11111110 前面的“#”符表示后面跟的是一个数字，而不是地址。数后面的“B”表示该数是二进制数。

02：在程序第二行语句里，通过 MOV 指令，将寄存器 A 中的寄存值 11111110 传送到 P1 输出。P1 对外是 I/O 端口，对内也是寄存器，寄存的值 11111110 与 P1 端口的 8 条引线所接的灯——对应，0 表示亮，1 表示灭。所以，P1 端口的第一个灯被点亮，其他 7 个灯灭。

03：在程序第三行语句里，通过控制转移类指令 JMP 使程序保持当前的输出状态。“\$”符号表示目前的地址，即程序计数值不变，程序不向下运行，也称动态停机。

04：第四行，程序结束。

3. 边用边学指令

本节用到的指令有 MOV、JMP 和 END。

- MOV：是使用频率最高的指令，每个程序中都会多次用到它。它的功能是传送数据，传送与英文 move 是一致的。与 MOV 指令相似的还有 MOVX(MOV+X)和 MOVC(MOV+C)指令，MOVX 指令用于在单片机外部存储器与累加器之间的数据传送，MOVC 指令在需要将查表数据传送到累加器时使用。

- JMP：是控制转移类指令。控制转移类指令分为有条件转移类指令和无条件转移类指令。JMP 是无条件转移类指令，指令的后边是标出的转移地址。

如“JMP \$”，\$表示目前的地址，程序会在目前的地址处运行。再如“JMP START”，程序就会跳转到 START 地址处运行。JMP 指令的功能是指令指向哪里，程序就转移到哪里，不需要任何附加条件。

与 JMP 指令相似的还有 AJMP (A+JMP)、LJMP (L+JMP) 和 SJMP (S+JMP) 指令，它们的基本功能都是无条件转移，但使用上略有不同。

AJMP 指令使用于 2KB 范围内的绝对转移；LJMP 指令使用于 64KB 范围内的长转移；SJMP 为相对转移指令。在上述 4 个转移类指令中，只要记住 JMP 指令的基本功能，其他 3 个指令容易记忆。

- END：是汇编语言源程序结束的标志，是伪指令。伪指令只在汇编时起作用，汇编后不产生目标代码，不影响程序的执行。一个源程序只能有一个 END 伪指令，放在程序的最后一行。

6.1.4 实例测试

将编好的源程序经过编译写入单片机，然后就可以在实验板上测试，看编写是否正确。在写入单片机之前还可以进行模拟仿真，模拟仿真方法，将在下节介绍。

在实验板上测试时，首先要注意实验板上单片机插座与 LED 相连的端口是否和程序指定的端口一致，如果不一致，要将源程序指定的端口改为实验板上与 LED 相连的端口；

还要注意 89C51 芯片的缺口要与实验板插座上的缺口方向相一致，单片机 1 脚朝向锁紧座手柄的方向，不要插反，否则会烧毁芯片。

当检查无误后，接通电源就会看到第一个 LED 亮起来。

6.1.5 经验总结

本节程序 JS01.ASM 是简单的顺序结构程序，程序执行是从第一行开始，依次向下运行。顺序结构程序是最简单的，但也是最基础的，其他程序结构组成中都含有顺序结构的形式。

所以，编写程序既要学习 80C51 系统指令和伪指令，即汇编语言，也要注意逐步掌握一些编程的技巧，其中包括掌握几种典型程序的结构形式。

最常见的形式有顺序结构程序、分支结构程序、循环结构程序、子程序和查表程序。这几种基本结构程序的设计方法是程序设计的基础。

6.2 模拟仿真

模拟仿真纯软件仿真，是依靠计算机模拟单片机的实际工作环境，来验证单片机程序的编写是否正确。它不但能够解决程序实际运行中的一些问题，而且能够清楚地看到每条语句在单片机内运行的情况，有利于初学者理解指令、学习编程、了解单片机的工作原理。

Keil C51 和 MedWin 集成开发系统软件都具有模拟仿真功能，使用方法也基本相同。Keil C51 软件模拟仿真的使用方法请参照第 17 章第 3 节，下面介绍在 MedWin 集成开发系统软件上进行模拟仿真的方法，以上节程序 JS01.ASM 为例，具体步骤如下。

6.2.1 进入模拟仿真状态

- (1) 双击 MedWin 图标，选择“模拟仿真”，进入 MedWin 工作窗口。
- (2) 如果 JS01.ASM 源程序刚编写完，则需要从菜单上选择“项目管理”“新建项目文件”“编译/汇编”。
- (3) 在项目管理下拉菜单中选择“产生代码并装入”选项，程序便进入模拟仿真状态，在窗口左上角出现一个黄色的小箭头，如图 6.3 所示。

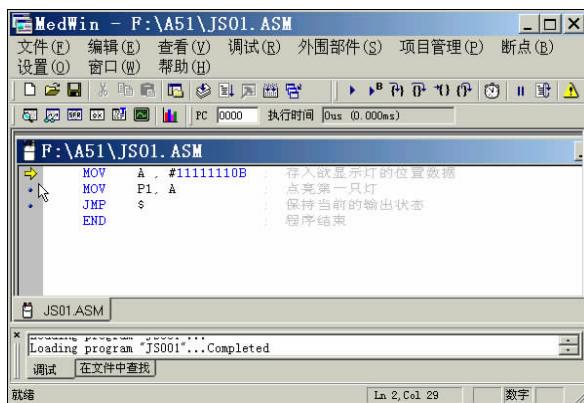


图 6.3 进入调试状态

6.2.2 展现观察窗口

在模拟仿真里，软件提供了寄存器及外围部件的观察窗口。

(1) 单击窗口上方“查看”菜单项，在下拉菜单中选择“寄存器”、“特殊功能寄存器”选项。

(2) 单击窗口上方“外围部件”菜单项，将出现单片机外围部件名，如“中断”、“端口”、“定时器/计数器”和“串行口”等，本程序只需要选择“端口”即可，出现的观察窗口如图6.4所示。

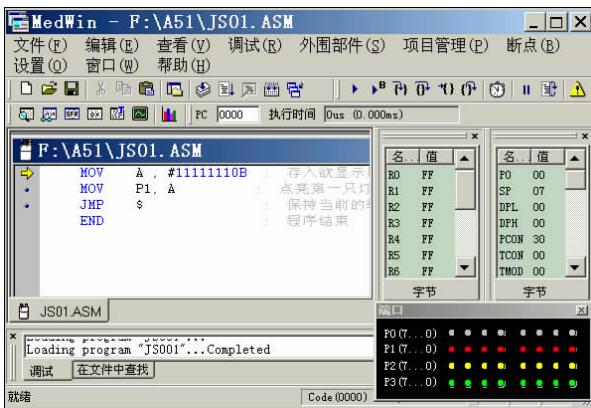


图 6.4 寄存器及端口观察窗口

6.2.3 选择调试方式

单击上方“调试”菜单项，在下拉菜单中显示“开始调试”、“终止调试”及各种调试方式，如“指令跟踪”、“指令单步”等，同时，在上方工具栏内还有相应的调试工具图标，如图6.5所示。选择“指令跟踪”进行调试，其快捷键为F7。

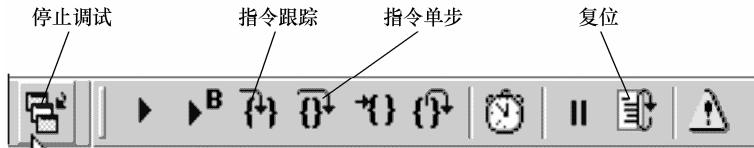


图 6.5 部分调试工具图标

6.2.4 观察寄存器值的变化

注意观察寄存器内的A值和特殊功能寄存器内P1值的变化，P1对外是输出端口。寄存器的值为十六进制。在程序没运行前，寄存器A值为00(00H=00000000B)，寄存器P1值为FF(FFH=11111111B)。表6.1是P1寄存器值与P1端口亮灯情况的对照表。

表 6.1 P1 寄存器值与 P1 端口亮灯情况对照表

二进制	十六进制	亮灯情况	二进制	十六进制	亮灯情况
11111111	FF	全灭	11101111	EF	5亮
11111110	FE	1亮	11011111	DF	6亮
11111101	FD	2亮	10111111	BF	7亮
11111011	FB	3亮	01111111	7F	8亮
11110111	F7	4亮	00000000	00	全亮

按 F7 (跟踪的快捷键) 键 , 当程序运行到第一行语句后 , 会看到寄存器 A 的值由 00 变为 FE , 如图 6.6 所示 , 这说明通过执行 MOV 数据传送指令 , 已将数字 11111110 送到 A 寄存器中。特殊功能寄存器 P1 的值仍为 FF , FF=11111111 , 1 表示灯熄灭 , 此时 8 个灯全灭。

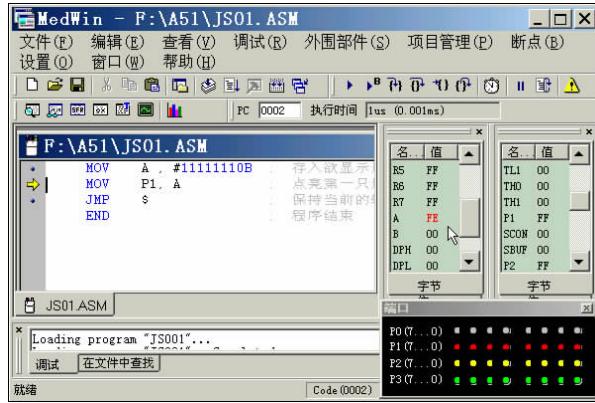


图 6.6 寄存器 A 值变为 FE

再按 F7 键 , 使程序运行到第二行语句 , 此时会看到 P1 的值由 FF 变为 FE , FE=11111110 , 0 表示灯亮 , 如图 6.7 所示。



图 6.7 P1 值变为 FE

因为 P1 本身就是输出端口 , 所以从端口的观察窗口中可以看到 P1 最右边的圆点变成灰色 , 说明灯亮。

再按 F7 键 , 使程序运行到第 3 行语句 , 此时 A 值和 P1 值都不再发生变化 , 程序停在此处 , 此时也称为动态停机 , 以保持当前输出状态不变。

注意的是 : 当模拟仿真无误后 , 在 “ 项目管理 ” 菜单中选择 “ 输出 Intel HEX 文件 ” , 并使用编程器将程序写入单片机 , 为在实验板上实验做准备。

6.3 点亮 6 只灯

功能说明 : 单片机 P1 端口接 8 只 LED , 点亮第 1 、 3 、 4 、 6 、 7 、 8 只灯。

6.3.1 程序设计

1. 流程图

程序设计流程如图 6.8 所示。

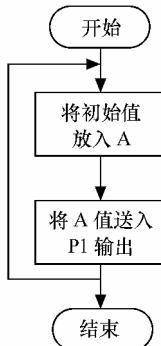


图 6.8 程序流程

2. 程序

汇编语言编写的点亮 6 只灯源程序 JS02.ASM，代码如下。

01:	START:	MOV	A , #00010010B	; 存入欲显示灯的位置数据
02:		MOV	P1, A	; 点亮灯
03:		JMP	START	; 重新设定显示值
04:		END		; 程序结束

6.3.2 代码详解

上述程序是在 MedWin 或 Keil C51 集成开发系统软件上编写的。

1. 标号说明

START：为程序标号，是起始程序的进入点。

标号是用户自己给语句设定的一个符号。标号要直接写在指令语句前，用冒号与指令语句分开。如程序语句“ START: MOV A, #00010010B ”中的“ START ”就是为程序第一行设定的一个符号。

标号由字母或数字构成，第一个字符必须是字母，系统中保留的字符或字符串不能作标号，以免引起混淆。

2. 程序分析解释

本节程序与第一节程序基本相同，区别有两点：一是二进制数不同，8 位中有 6 位是 0，所以会使对应的 6 个灯亮；二是采用了循环结构形式维持 P1 端口的输出状态。

01：在程序第一语句里，通过 MOV 数据传送指令将二进制数 00010010B 传送到寄存器

A 中寄存，0 表示灯亮，可以看出，要被点亮的是第 1、3、4、6、7、8 只灯。

02：在程序第二行语句里，通过 MOV 指令将寄存器 A 中的寄存值 00010010B 传送到 P1 输出。P1 对外是输出端口，对内是寄存器，寄存的 00010010B 值与 P1 端口的 8 条引线所接的 8 只灯一一对应，0 表示灯亮，1 表示灯灭。

03：在程序第三行语句里，通过控制转移类指令 JMP，使程序转移到标号 START 处运行，即指向程序第一行，重新开始运行，不断地循环，维持 P1 端口的输出。

04：程序结束。

6.3.3 模拟仿真

模拟仿真步骤同上节。注意观察程序运行时的循环过程，程序运行到第三行语句后，既没停止，又没向下运行，而是转到第一行，重新开始运行，这样不断地循环下去，维持 P1 口输出状态，保持灯亮。

6.3.4 实例测试

当检查实验板上安装无误后，接通电源就会看到第 1、3、4、6、7、8 只灯亮起。对照程序中送入 P1 寄存器的数据 00010010B，观察 P1 寄存器置 0 的位所接的灯是否亮，置 1 的位所接的灯是否灭。

P1 在单片机内部是一个特殊寄存器，有 8 位，在单片机的外部引出 8 根线，成为输入/输出端口。观察时既要看到外部变化，也要思考其内部基本工作原理。

6.3.5 经验总结

在程序设计时，往往会遇到同样一个程序段要重复运行多次的情况，这时应该采用循环结构形式。这种结构可以大大简化程序，是一种常用的程序结构形式。

循环程序可分为单重循环和多重循环。

- 单重循环

单重循环是指循环体中不再包含循环的程序，本节讲过的程序 JS02.ASM 就是简单的单重循环程序，当程序执行到第 3 行语句后，在转移指令 JMP 的作用下使程序运行又回到第一行重新开始，不断循环。

- 多重循环

多重循环是指在循环体中还包含循环（也称循环嵌套）的程序。在多重循环中，不允许循环体互相交叉，也不允许从循环程序的外部跳入循环程序的内部。正确的多重循环如图 6.9 所示。

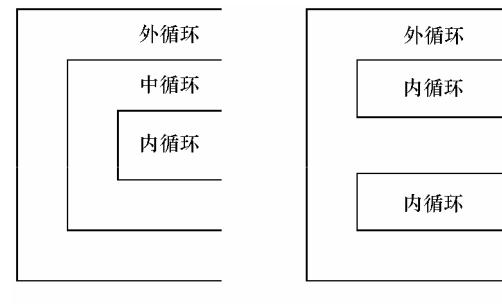


图 6.9 多重循环示意图

6.4 亮灯循环左移

功能说明：单片机 P1 端口接 8 只 LED，每次点亮一只，向左移动点亮，重复循环。

该程序缺少延时环节，适合模拟仿真观察。

6.4.1 程序设计

1. 流程图

程序设计流程如图 6.10 所示。

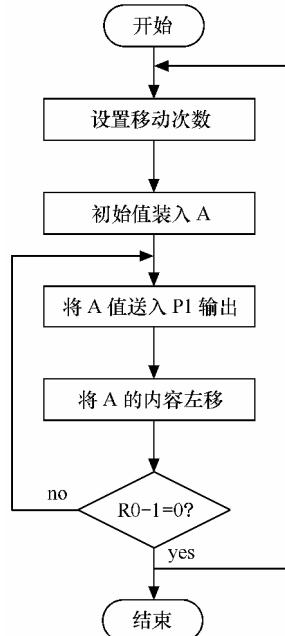


图 6.10 程序流程

2. 程序

汇编语言编写的灯循环左移源程序 JS03.ASM，代码如下：

01:	START:	MOV	R0, #8	; 设左移 8 次
02:		MOV	A, #11111110B	; 存入开始点亮灯位置
03:	LOOP:	MOV	P1, A	; 传送到 P1 并输出
04:		RL	A	; 左移一位
05:		DJNZ	R0, LOOP	; 判断移动次数
06:		JMP	START	; 重新设定显示值
07:		END		; 程序结束

6.4.2 代码详解

1. 标号说明

START：起始程序的进入点。

LOOP：左移循环执行的进入点。

2. 寄存器使用分配情况

程序中 A、P 和 R0 为寄存器。其中 A、P 为特殊功能寄存器。

R0 为普通寄存器，在程序里作计数器用。

3. 程序分析解释

01：设左移 8 次（8 只灯）。通过 MOV 数据传送指令，将数字 8 传送到 R0 寄存器寄存，此时 R0 寄存器当作计数器用。

02：存入亮灯的位置。通过二进制数#11111110B 可以看出，灯将从右边第一位亮起（0 代表灯亮）。此数通过 MOV 数据传送指令传送到累加器 A 中寄存。

03：送到 P1 并输出。将累加器 A 中的数 11111110 传送到 P1，故灯从右边第一位亮起。

04：通过左环移位指令 RL，使累加器 A 中的数由 11111110 变为 11111101，0 向左移一位，可以看出下一次亮起的将是右边第二只灯。

05：判断移动次数。DJNZ 是有条件转移指令，计数器 R0 的数值（最初为 8）减 1 后不为 0 时转移，转移到 LOOP 处循环，此时计数器在倒计时。当计数器为 0 时转移条件不成立，即打破了循环，程序向下运行。此时亮的灯已经由最右边移到了最左边，完成了一次左移。

06：重新设定显示值。通过转移指令 JMP，使程序转移到标号 START 处，即程序从头开始运行。

07：程序结束。

4. 边用边学指令

本节程序用到了两个新指令 DJNZ 和 RL。

- DJNZ：控制转移类指令。

DJNZ 是有条件转移指令，只有满足一定的条件才能转移。其条件是与计数器配合使用的，每循环一次计数器减 1，只有计数器减 1 不为 0 时才转移。

当计数器的值减到 0 时，DJNZ 指令就失去转移功能，程序向下运行。所以，DJNZ 也称为减 1 非 0 转移指令。

而 JMP 是无条件转移指令，转移时不需要条件，JMP 指令后边的标号标在什么地方，就转移到什么地方。

如执行语句“JMP START”后，程序将无条件的转移到标号 START 处运行。而执行语句“DJNZ R0，LOOP”后，能否转移到标号 LOOP 处，是由计数器 R0 控制的，只有当 R0 的值不为 0 时才能转移，当为 0 时就不转移。

- RL：逻辑运算及位移类指令中的累加器左环移位指令。80C51 的位移指令只能对累加器 A 进行位移。

如语句“RL A”使累加器 A 内各位向左环移一位，例如累加器 A 原来各位为 11111110，运行“RLA”语句后，累加器 A 各位变为 11111101，即第 1 位 0 移到第 2 位，第 2 位移到第 3 位，依此类推向左环移位，最后的第 8 位移到了第 1 位。

6.4.3 模拟仿真

本节在仿真时需注意观察以下两点。

- 特殊功能寄存器 A 和 P1 的值的变化。A 和 P1 的值是相同的，只是 A 的值先变，P1 的值随后变。其值变化为 FE FD FB F7 EF DF BF 7F，参照表 6.1 就可以看出亮灯左移情况。
- 注意观察 05 语句“DJNZ R0 LOOP”，当 R0 不为 0 时，程序跳转到标号 LOOP 处进行内循环。当 R0 为 0 时，程序向下执行“JMP START”语句，此时程序进入外循环。

6.4.4 实例测试

将单片机插入实验板接通电源后，看不到亮灯在移动，而是 8 只发光二极管全亮。这是人眼视觉暂留原理的结果，当一个物体在眼前迅速离开的时候，该物体的影像并不是立即消失，而会有短暂的停留。如果此时该物体再迅速回到眼前，人眼就感觉不到它曾离开过。

由于程序运行非常快，LED 熄灭后，LED 的影像还没消失，LED 又被点亮。所以，在变化频率很高的情况下，人眼是无法辨认清楚的。只有在 LED 的亮与灭之间加上适当的延时才能解决这个问题。

6.4.5 经验总结

本节程序 JS03.ASM 在上节单重循环程序 JS02.ASM 的基础上增加了 3 条语句“MOV R0, #8”、“RL A”和“DJNZ R0, LOOP”构成内循环，使整个程序成为双重循环结构的程序。

由于内循环的循环次数是已知的，所以，循环的控制采用了计数器控制方法，即在循环初始部分将循环次数 8 输入计数器 R0 中，作为 DJNZ 指令的转移条件，实现内循环控制，如下面代码所示。

程序：JS03.ASM

```

START: MOV      R0, #8
        MOV      A, #11111110B
LOOP:  MOV      P1, A
        RL      A
        DJNZ   R0, LOOP
        JMP     START
        END
    
```

6.5 亮灯循环右移

功能说明：单片机 P1 端口接 8 只 LED，每次点亮一只，向右移动点亮，重复循环。

6.5.1 程序设计

1. 流程图

程序设计流程如图 6.11 所示。

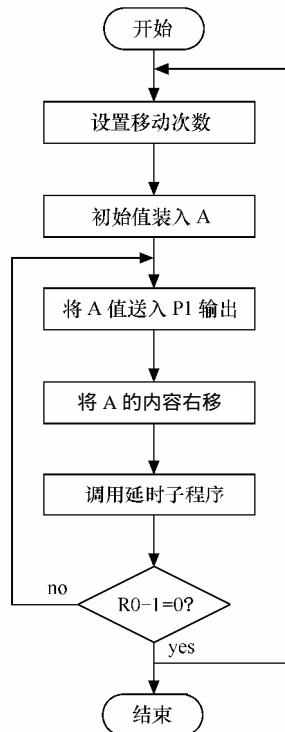


图 6.11 程序流程

2. 程序

汇编语言编写的灯循环右移源程序 JS04.ASM，代码如下：

01:	START:	MOV	R0, #8	; 设右移 8 次
02:		MOV	A, #01111111B	; 存入开始点亮灯位置
03:	LOOP:	MOV	P1, A	; 传送到 P1 并输出
04:		ACALL	DELAY	; 调延时子程序
05:		RR	A	; 右移一位
06:		DJNZ	R0, LOOP	; 判断移动次数
07:		JMP	START	; 重新设定显示值
08:	DELAY:	MOV	R5, #50	
09:	DLY1:	MOV	R6, #100	
10:	DLY2:	MOV	R7, #100	
11:		DJNZ	R7,\$	
12:		DJNZ	R6, DLY2	

```

13:          DJNZ    R5, DLY1
14:          RET             ; 子程序返回
15:          END             ; 程序结束

```

6.5.2 代码详解

1. 标号说明

START：起始程序的进入点。
LOOP：右移循环执行的进入点。
DELAY：延时子程序的进入点。
DEL1：延时重复递减判断的进入点 1。
DEL2：延时重复递减判断的进入点 2。

2. 寄存器使用分配情况

程序中 A、P、R0、R5、R6 和 R7 为寄存器，其中 A、P 为特殊功能寄存器。

R0 为循环计数器。

R5、R6 和 R7 为子程序延时循环计数器。

3. 程序分析解释

本程序是由主程序和子程序两部分组成。

01~07：主程序部分，主程序部分与上节程序 JS03.ASM 基本相同，但是灯向右移动点亮，重复循环，同时，增加 04 行“ACALL DELAY”语句。指令 ACALL 是调用延时子程序指令，当执行该语句时，程序会转移到标号 DELAY 处运行延时子程序。

延时子程序最后一行是子程序返回指令 RET，该指令会使程序返回到主程序调用处的下一行运行，即执行第 05 行。增加延时子程序的目的是使人能看清楚 LED 的移动。

08~14：延时 1s 的子程序，15 行表示程序结束。

4. 边用边学指令

本程序用了 3 个新指令 RR、ACALL 和 RET。

- RR 和 RL 都是位移类指令，位移类指令只能对累加器 A 进行位移。其中“RL A”使累加器 A 内各位向左环移一位，例如累加器 A 原来各位为 11111110，运行“RL A”语句后，累加器 A 各位变为 11111101。语句“RR A”使累加器 A 内各位向右环移一位，例如累加器 A 原来各位为 0111111，运行“RR A”语句后，累加器 A 各位变为 10111111。

- ACALL 和 RET 是控制转移类指令中的子程序调用指令和返回指令。子程序调用指令 ACALL 放在主程序里，子程序返回指令 RET 放在子程序中的最后一行。两条指令在程序中是成对出现的。

与 ACALL 相似的指令还有 LCALL，它们都是子程序调用指令。区别是 ACALL 指令可以在 2KB 范围内进行绝对调用，LCALL 指令可以在 64KB 范围内进行调用，扩展了调用范围，一般在单片机外接扩展寄存器时调用程序时使用。

6.5.3 模拟仿真

模拟仿真在前节的基础上，需要注意以下3个环节。

- 在模拟仿真时，为了缩短运行延时子程序所消耗的时间，要将子程序中的第08、09和10行里的存入数缩小，例如可以将50、100、100分别改为5、10、10。更改是在MedWin软件编辑窗口中进行的，更改后在进行调试时，程序要先进行重新“编译/汇编”和“产生代码并装入”，之后才能跟踪调试。

- 注意观察调用子程序04行和子程序返回14行的过程。
- 注意观察延时子程序多重循环结构的程序运行路线。

6.5.4 实例测试

将写入程序的单片机插入实验板，接通电源后会看到LED在不断地向右移动，当移到最右端后，又回到最左端重新开始向右移动，不断循环。其移动的速度由延时的时间决定，延时的时间越长，移动的速度越慢；延时的时间越短，移动的速度越快。

6.5.5 经验总结

在编程过程中，常会遇到有相同问题多次出现的情况。例如，在LED右移循环的程序里，点亮的灯每移动一次，就需要延时1s，如果每次延时都要编写一段程序，会使编程十分繁琐，也会占用大量程序存储空间。

通常把这些在程序中多次出现的程序片段编成相对独立的子程序，如前面讲的延时程序就是子程序，它可以被多次调用。这样既避免重复，又节省了存储空间。

主程序可在不同的位置通过调用指令ACALL或LCALL多次调用子程序，又通过子程序中最后一条返回指令RET，返回到主程序的断点地址继续执行主程序，如图6.12所示。

如果子程序在执行过程中又调用别的子程序，这种在子程序中再次调用子程序的结构，称为子程序嵌套。图6.13所示为一个两层嵌套的子程序调用结构示意图。

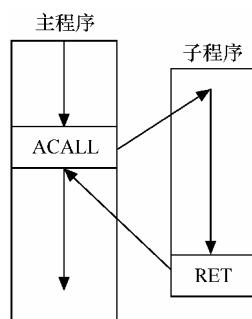


图6.12 子程序的调用与返回

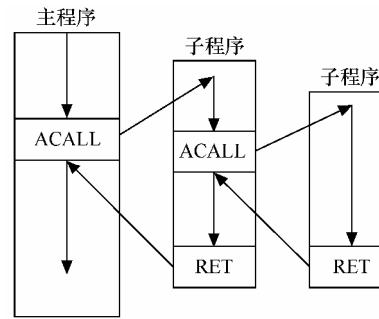


图6.13 子程序的嵌套调用与返回

6.6 延时时间的计算

在单片机的实时控制系统中，常常需要用到延时操作，所以，延时子程序往往是编写单

片机程序中不可缺少的一部分。延时方法有硬件延时和软件延时，硬件延时将在后面有关章节中介绍，本节将介绍软件延时方法。

所谓软件延时，就是让计算机重复执行一些无具体任务的程序，利用执行程序的时间来达到延时的目的。

6.6.1 机器周期和指令周期

单片机读、写操作都需要消耗一定的时间，机器周期是指单片机完成一个基本操作所用的时间，如读操作、写操作等。当石英晶体为 12MHz 时，1 个机器周期为 $1\mu s$ 。

指令周期是指单片机执行一条指令所需要的时间，一个指令周期通常含有 1~4 个机器周期，其中常用的 DJNZ 指令周期为两个机器周期，即执行 DJNZ 指令需要 $2\mu s$ ；MOV 指令周期为一个机器周期，即执行 MOV 指令需要 $1\mu s$ 。

6.6.2 单重循环短暂延时

短暂的时间延时可采取简单的单重循环结构来实现，例如，下面程序为延时 $540\mu s$ 的短暂延时子程序，程序中采取了单重循环。

$540\mu s$ 延时子程序：

源程序	机器周期数	执行时间	
1 DEX:			
2 MOV R7, #180	1	$1\mu s$	
3 DE1: NOP	1	$1\mu s$	
4 DJNZ R7, DE1	2	$2\mu s$	—— 单重循环
5 RET	2	$2\mu s$	

此子程序中，由第 3、4 行代码构成单重循环结构，其中，DJNZ 指令为控制转移指令，该指令每执行一次，寄存器 R 值减 1，只要 R 值减 1 后不为 0，就会转移到第 3 行标号 DE1 处去执行。

每循环一次需要的时间为 $3\mu s$ ，由于 R 值为 180，所以要循环 180 次，循环花费的时间为 $540\mu s$ 。该延时子程序总的延时时间还要包括执行 MOV 指令时间 $1\mu s$ 和执行 RET 返回指令时间 $2\mu s$ ，但由于这个时间比循环时间显得很短，所以，延时时间的长短主要是由循环次数来控制。

6.6.3 多重循环较长时间延时

如果需要较长时间的延时，则需采用多重循环结构。例如下面程序为 1s 延时子程序，程序中采取了多重循环。

1s 延时子程序：

源程序	机器周期数	执行时间	
1 DELAY_1s:			

2	MOV	R5, #50	1	1μs	外
3	DLY1:	MOV	R6, #100	1	2μs 中
4	DLY2:	MOV	R7, #100	1	1μs 内
5		DJNZ	R7, \$	2	2μs
6		DJNZ	R6, DLY2	2	2μs
7		DJNZ	R5, DLY1	2	2μs 多重循环
8		RET		2	2μs

此子程序采用了 3 重循环结构，先运行第 5 行代码操作，每次减 1，减到 0 为止；再运行第 6 行代码对 R6 进行减操作，每次减 1，减 1 后不为 0，则转移到标号 DLY2 处运行，此时将 R7 赋值为 100，并再对 R7 进行减法内循环。

当 R6 减为 0 时，程序运行到第 7 行，开始外循环，R5 减 1 不为 0 时转移到标号 DLY1 处运行。为了计算上的简便，可以忽略赋值语句的时间，只计算“DJNZ R7, \$”语句的执行时间，该指令执行一次需 2μs，执行的次数为 R5、R6 和 R7 值的乘积，即：

$$\begin{aligned} \text{总延时} &= 2\mu s \times R7 \times R6 \times R5 \\ &= 2\mu s \times 100 \times 100 \times 50 = 1\ 000\ 000\mu s \\ &= 1s \end{aligned}$$

6.6.4 对延时程序的改进

从上节程序可以看出，只要改变寄存器 R5、R6 和 R7 的值，就可以获得不同延时时间。在上例中假设 R6 和 R7 值不变，只改变 R5 值（R 取值范围为 1 ~ 255），可获得不同的延时。由于忽略了赋值语句的执行时间，实际延时时间要比计算的时间略大一些。

为了使一个延时子程序能产生不同的延时，可以将 6.6.3 中的程序改进为以下的形式。其中 R7、R6 值不变，通过改进 R5 值，即改变外循环次数来改变延时时间，延时子程序的延时时间为 $0.02s \times R5$ 。

使用时，在调用 DELAY 延时子程序之前，要根据对延时时间的要求，先对 R5 赋值，假如延时为 0.5s，需将 R5 值赋值为 25；延时为 1s，需将 R5 值为 50，R5 最大值为 255，这是因为 R0 ~ R7 都是 8 位寄存器，最大存放数据为二进制数 11111111，即 255，在使用时注意不要超出其有效范围。改进后的延时程序使用形式如下。

```
MOV     R5, #50
ACALL  DELAY_20ms×R5
```

延时时间为 $0.02s \times R5$ 的延时程序如下：

```
1  DELAY_20ms×R5:
2
3  DLY1: MOV   R6, #100
4  DLY2: MOV   R7, #100
5      DJNZ  R7, $
6      DJNZ  R6, DLY2
7      DJNZ  R5, DLY1
```

8 RET

与 6.6.3 节不同的是，R5 的赋值语句“MOV R5, #50”放在调用延时程序之前，可以根据程序需要灵活改变延时时间。

6.7 亮灯左移与右移循环

功能说明：单片机 P1 端口接 8 只 LED，每次点亮一只，先把右边的第 1 只点亮，0.5s 后点亮右数的第 2 只灯，第 1 只熄灭，再过 0.5s 点亮右数的第 3 只灯，第 2 只熄灭，亮灯按此顺序由右向左移动。当亮灯移到最左侧后，开始与上述反方向移动，即亮灯由左向右移动，重复循环。

6.7.1 程序设计

本程序是前边介绍的灯循环左移和循环右移的组合。

1. 流程图

程序设计流程如图 6.14 所示。

2. 程序

汇编语言编写的灯左移与右移循环源程序 JS05.ASM 代码如下：

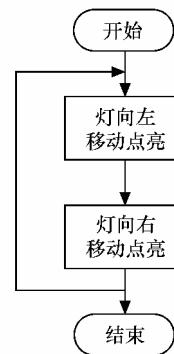


图 6.14 程序流程

01:	START:	MOV	R0, #8	; 设左移 8 次
02:		MOV	A, #0FEH	; 存入开始亮灯位置
03:	LOOP:	MOV	P1, A	; 传送到 P1 并输出
04:		ACALL	DELAY	; 调用延时子程序
05:		RL	A	; 左移一位
06:		DJNZ	R0, LOOP	; 判断移动次数
07:		MOV	R1, # 8	; 设右移 8 次
08:	LOOP1:	RR	A	; 右移一位
09:		MOV	P1, A	; 传送到 P1 并输出
10:		ACALL	DELAY	; 调延时子程序
11:		DJNZ	R1, LOOP1	; 判断移动次数
12:		JMP	START	; 重新设定显示值
13:	DELAY:	MOV	R5, #25	; 延时 0.5s 子程序
14:	DLY1:	MOV	R6, #100	
15:	DLY2:	MOV	R7, #100	
16:		DJNZ	R7, \$	
17:		DJNZ	R6, DLY2	
18:		DJNZ	R5, DLY1	
19:		RET		; 子程序返回
20:		END		; 程序结束

6.7.2 代码详解

1. 标号说明

START：起始程序的进入点。
 LOOP：循环左移操作的进入点。
 LOOP1：循环右移操作的进入点。
 DELAY：延时子程序的进入点。
 DEL1：延时重复递减判断的进入点 1。
 DEL2：延时重复递减判断的进入点 2。

2. 寄存器使用分配情况

程序中 A、P、R0、R1、R5、R6 和 R7 为寄存器，其中 A、P 为特殊功能寄存器。
 R0 为左移循环计数器。
 R1 为右移循环计数器。
 R5、R6 和 R7 为子程序延时循环计数器。

3. 程序分析解释

本程序是前边介绍的灯左移循环和灯右移循环两个程序的结合。

01~06：完成灯的左移操作，其中为了编写方便，累加器 A 的数用十六进制表示。
 07~11：完成灯的右移操作，右移部分与左移部分的程序结构相似。
 13~19：延时 0.5s 的子程序，在本例中寄存器 R5 的值为 25，子程序的延时为 0.5s。

6.7.3 模拟仿真

本节模拟仿真时，主要要注意观察左右移语句的运行情况，左移时 P1 值的变化为 FE FD FB F7 EF DF BF 7F；而右移时其值变化为 7F BF DF EF F7 FB FD FE，参照 6.2 节表 6.1 就可以看出亮灯左移、右移的情况。

在模拟仿真时，为了缩短运行时间，可以不运行延时子程序，办法是在调用延时子程序语句（第 04、10 行）前加分号，此时会看到该行语句颜色变淡，程序运行时就会跳过该行语句，延时子程序自然就不起作用了。

但要注意改后一定要重新进行“编译/汇编”和“产生代码并装入”（这一过程可通过重新单击“开始调试”按钮自动来完成）。当恢复程序原来状态（去掉分号）时也要重新进行“编译/汇编”和“产生代码并装入”。

6.7.4 实例测试

将写入程序的单片机插入实验板接通电源后，会看到亮灯先向左移动，移到最左端后，再向右移动，不断循环。如果将程序中的移动指令 RL 改为 RR，并将原 RR 改为 RL，程序中其他部分不变，重新编译写入单片机再测试，会看到亮灯先向右移动，移到最右端后，再向左移动，不断循环。

6.7.5 经验总结

灯左移和右移程序结构是相同的，其程序结构都是在开始时设置移动位数，并输入亮灯移动初始值，后边利用 DJNZ 指令判断移动次数是否完成。

6.8 双灯左移右移加闪烁

功能说明：单片机 P1 端口接 8 只 LED，每次点亮两只，先从右边向左边移动点亮，再从左边向右边移动点亮，然后闪烁两次，重复循环。

6.8.1 程序设计

本程序是在上节程序基础上，由单灯左右移动变为双灯左右移动，并增加了灯闪烁的功能。

1. 流程图

程序设计流程如图 6.15 所示。

2. 程序

汇编语言编写的双灯左移右移加闪烁源程序 JS06.ASM 代码如下：

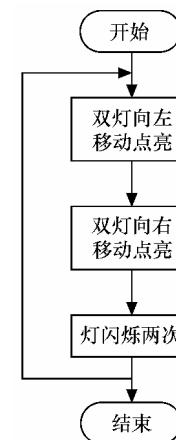


图 6.15 程序流程

01:	START:	MOV R0, #7	; 设左移 7 次
02:		MOV A, #0FCH	; 存入开始亮灯位置
03:	LOOP:	MOV P1, A	; 传送到 P1 并输出
04:		ACALL DELAY	; 调延时子程序
05:		RL A	; 左移一位
06:		DJNZ R0, LOOP	; 判断移动次数
07:		MOV R1, #7	; 设右移 7 次
08:	LOOP1:	RR A	; 右移一位
09:		MOV P1, A	; 传送到 P1 并输出
10:		ACALL DELAY	; 调延时子程序
11:		DJNZ R1, LOOP1	; 判断移动次数
12:		MOV R2, #3	; 设置闪烁次数
13:		MOV A, #00H	; 设初始值
14:	LOOP2:	MOV P1, A	; P1 端口灯亮
15:		ACALL DELAY	; 调用延时子程序
16:		CPL A	; A 取反值
17:		DJNZ R2, LOOP2	; 判断闪烁次数
18:		JMP START	; 重新设定显示值
19:	DELAY:	MOV R5, #25	
20:	DLY1:	MOV R6, #100	

```

21: DLY2:    MOV      R7, #100
22:          DJNZ    R7, $
23:          DJNZ    R6, DLY2
24:          DJNZ    R5, DLY1
25:          RET           ; 子程序返回
26:          END           ; 程序结束

```

6.8.2 代码详解

1. 标号说明

START：起始程序的进入点。
 LOOP：左移循环操作的进入点。
 LOOP1：右移循环操作的进入点。
 LOOP2：闪烁循环操作的进入点。
 DELAY：延时子程序的进入点。
 DEL1：延时重复递减判断的进入点 1。
 DEL2：延时重复递减判断的进入点 2。

2. 寄存器使用分配情况

程序中 A、P、R0、R1、R2、R5、R6 和 R7 为寄存器，其中 A、P 为特殊功能寄存器。
 R0 为左移循环计数器。
 R1 为右移循环计数器。
 R2 为闪烁循环计数器。
 R5、R6 和 R7 为子程序延时循环计数器。

3. 程序分析解释

本程序是在上节程序的基础上，将代码中的第 2 行亮灯初始值由 FEH 改为 FCH，使每次点亮两只灯，并将亮灯的移动位数由 8 改为 7，同时程序中增加了灯闪烁功能的语句，其他部分与上节说明完全相同。下面仅就闪烁功能的语句加以说明。

12 ~ 17：是实现闪烁功能的语句。其中第 12 行语句功能为设置闪烁次数。该语句向计数器中存入数字 3，这时灯将亮两次，中间灭一次，共计循环 3 次。

13 行：设置循环初始值，在 A 中存入 0000000B，0 代表灯亮，可以看出开始时灯将全部点亮；第 14 行语句将 A 中数据传送到寄存器 P1 同时使 P1 端口的灯全部点亮。

15：调用延时 0.5s 子程序。

16：A 中各位取反，A 中原来存入的数是 0000000B，执行“CPL A”语句后变为 1111111B，1 代表灯灭，所以第 2 次循环时 P1 端口的灯全部熄灭。

17：判断闪烁次数。当 R2 减 1 后不为 0，即 3 次循环没完成时，程序将跳转到标号 LOOP2 处继续执行，当 R2 减 1 为 0，即 3 次循环完成时，程序继续向下运行。

4. 边用边学指令

本程序用到新指令 CPL。

CPL 是位操作类指令，位操作类指令又称布尔运算指令，它是以位为单位进行运算和操作的。其中 CPL 指令的功能是将位取反相，如 A 中各位原来值是 11111111B，执行“CPL A”语句指令将对 A 中的各位取反相，变为 00000000B。

CPL 指令还可以对 I/O 端口中任何一个位单独作处理，如“CPL P1.2”，P1.2 是指 P1 端口第 3 位，如果第 3 位原来为 1 则将变为 0，如果第 3 位原来为 0 则将为变 1。

位操作类指令 CPL 与前边介绍的位移类指令 RL 和 RR 的区别在于 CPL 指令是对位取反，而 RL 和 RR 是对 A 的各位作环形移动。

如 A 中存储的数据为 11111110B，执行“RL A”指令后，将变为 11111101B，原来的第 1 位移动到第 2 位，原来的第 2 位移动到第 3 位，作依次移动，使原来第 8 位移动到第 1 位，作位环形移动。

6.8.3 模拟仿真

本节模拟仿真主要注意观察闪烁语句的运行情况，可以看到 P1 值为 00H FFH 00H 的变化，其中 00H 代表灯全亮，FFH 代表灯全灭。

在模拟仿真时，可以将延时程序注释掉，不运行延时子程序。

6.8.4 实例测试

将写入程序的单片机插入实验板接通电源后，会看到两只亮灯先向左移动，然后向右移动，再闪烁两次，不断循环。

将程序 JS06.ASM 改写为 6.8.5 中的 JS0601.ASM 形式后，在主程序中改变调用各子程序的顺序再做测试。

例如，将第 2 行“ACALL R_M”调整到第一行位置，测试时会看到亮灯先向右移动，然后向左移动，再闪烁两次，不断循环。再如，将第 3 行“ACALL RAY”调整到第 2 行位置，测试时会看到亮灯先向右移动，然后闪烁两次，再向左移，不断循环。

6.8.5 经验总结

本节程序 JS06.ASM 中的主程序部分完成亮灯左移、右移和闪烁功能，只有延时功能部分写为子程序。可以将左移、右移闪烁功能部分也写为子程序，主程序只负责按一定顺序调用各子程序及无限循环，这样会使程序更加灵活，编写和分析程序也更为轻松。

现将 JS06.ASM 程序改写为 JS0601.ASM，代码如下：

01	START:	ACALL	L_M
02		ACALL	R_M
03		ACALL	RAY
04		JMP	START
05			

```

06 L_M : MOV R0, #7
07          MOV A, #0FCH
08 LOOP:   MOV P1, A
09          ACALL DELAY
10         RL A
11         DJNZ R0, LOOP
12         RET
13
14 R_M:    MOV R1, #7
15 LOOP1:  RR A
16          MOV P1, A
17          ACALL DELAY
18         DJNZ R1, LOOP1
19         RET
20
21 RAY:    MOV R2, #3
22          MOV A, #00H
23 LOOP2:  MOV P1, A
24          ACALL DELAY
25         CPL A
26         DJNZ R2, LOOP2
27         RET
28
29 DELAY:  MOV R5, #25
30 DLY1:   MOV R6, #100
31 DLY2:   MOV R7, #100
32         DJNZ R7, $
33         DJNZ R6, DLY2
34         DJNZ R5, DLY1
35         RET
36 END

```

在分析一个较复杂的程序时，首先要弄清它是由几个模块所组成，每个模块主要功能是什么，模块之间是怎样联系在一起的。

以改写后的本节程序为例，它是由左移子程序、右移子程序、闪烁子程序和延时子程序4部分组成。每个之程序模块完成其特定的指定功能，4个子程序模块是通过主程序组合起来完成了程序的总体设计要求。

再分析各子程序模块结构可以看出，虽然每个模块功能不同，但程序结构基本相同，都是由最基本的循环结构所组成。在循环结构里先设置循环次数，再使用判断语句检查是否达到循环次数，没达到时继续进行内循环；达到循环次数后，程序将进入外循环。

这样将复杂程序划分若干个模块去分析，就会使问题由复杂变为简单。

不同功能的子程序模块可以看成是不同形状的积木，用一块块积木可以拼成不同形状的物体，即组成不同功能的程序。

6.9 用取表方式实现灯移动

功能说明：单片机端口接 8 只 LED，编程时利用取表的方法，使端口 P1 的 LED 先一次点亮 1 只跳跃左移，点亮顺序为 P1.0、P1.2、P1.4、P1.6；接着一次点亮 2 只，从左向右移动 2 次；一次点亮 3 只，左右移 3 次；然后 4 只灯 P1.0、P1.2、P1.4、P1.6 与 P1.1、P1.3、P1.5、P1.7 交互点亮 4 次；最后 8 只灯闪烁 6 次，不断循环。

6.9.1 程序设计

1. 流程图

程序设计流程如图 6.16 所示。

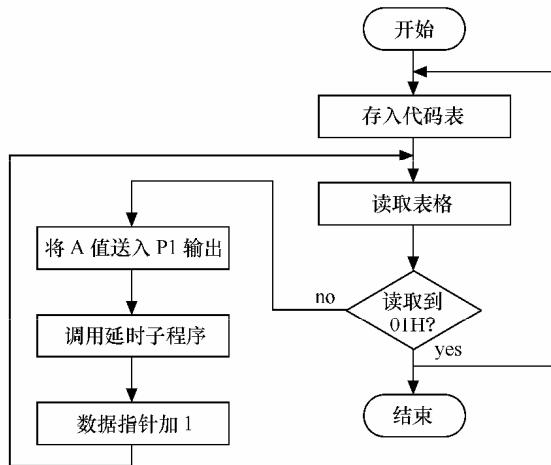


图 6.16 程序流程

2. 程序

汇编语言编写的用取表方式实现灯移动源程序 JS07.ASM 代码如下：

01:	START:	MOV	DPTR, #TABLE	; 存表
02:	LOOP:	CLR	A	; A 清 0
03:		MOVC	A, @A+DPTR	; 取表代码
04:		CJNE	A, #01H, LOOP1	; 不是 01H, 循环
05:		JMP	START	; 重新设定
06:	LOOP1:	MOV	P1, A	; 将 A 值送入 P1 输出
07:		ACALL	DELAY	; 调用延时子程序
08:		INC	DPTR	; 数据指针加 1
09:		JMP	LOOP	; 跳转到 LOOP 处
10:	DELAY:	MOV	R5, #25	; 延时 0.5s

```

11: DLY1: MOV R6, #100
12: DLY2: MOV R7, #100
13: DJNZ R7, $
14: DJNZ R6, DLY2
15: DJNZ R5, DLY1
16: RET ; 延时子程序返回
17: ; 单灯向左跳移 1 次控制码
18: TABLE: DB 0FEH, 0FBH, 0EFH, 0BFH
19: ; 2 灯向右移动 2 次控码
20: DB 3FH, 9FH, 0CFH, 0E7H
21: DB 0F3H, 0F9H, 0FCH
22: DB 3FH, 9FH, 0CFH, 0E7H
23: DB 0F3H, 0F9H, 0FCH
24: ; 3 灯左右移动 3 次控制码
25: DB 0F8H, 1FH, 0F8H, 1FH
26: ; 4 灯交互移动 4 次控制码
27: DB 0AAH, 55H, 0AAH, 55H
28: ; 8 灯闪烁 6 次控制码
29: DB 00H, 0FFH, 00H, 0FFH
30: DB 00H, 0FFH, 00H, 0FFH
31: DB 00H, 0FFH, 00H, 0FFH
32: DB 01H ; 结束码
33: END ; 程序结束

```

6.9.2 代码详解

1. 标号说明

START：起始程序的进入点。
 LOOP：循环操作的进入点。
 LOOP1：输出循环的进入点。
 DELAY：延时子程序的进入点。
 DEL1：延时重复递减判断的进入点 1。
 DEL2：延时重复递减判断的进入点 2。
 TABLE：代码表的进入点。

2. 寄存器使用分配情况

程序中 A、P、R5、R6、R7、DPTR 为寄存器，其中 A、P、R5、R6、R7 寄存器的作用与前节相同。

DPTR 是新用到的寄存器，也是惟一的 16 位专用寄存器，它是由两个 8 位寄存器 DPH 和 DPL 拼装而成，其中 DPH 为 DPTR 的高 8 位，DPL 为 DPTR 的低 8 位。它既可作为一个 16 位寄存器来使用，也可以作为两个独立的 8 位寄存器（DPH 和 DPL）来使用。

我们把代码表存入 DPTR，确切地说是将代码表的首地址 TABLE 送到 DPTR 中作地址，A 作为变址寄存器，基址寄存器和变址寄存器的内容相加（@A+DPTR）形成 16 位地址，该地址既是操作数的地址，也是存入地址指针即叫数据指针。知道了地址指针，按照地址指针所指，自然就能取到代码表中的控制码了。

3. 程序分析解释

本程序利用取表方式实现亮灯的移动。

利用取表的方式将控制码建成一个表，利用“MOV DPTR, #TABLE”语句存表，再利用“MOVC A, @A+DPTR”语句从表中读取出控制码，这种方法可方便地处理一些复杂的控制动作，下面重点介绍这一部分。

第 01~09 行代码的作用是存表、读取表并输出，是程序的主要部分。

01：存入 TABLE 表，用来控制灯的一组组十六进制数控制码建成一个表，取表名为 TABLE。将 TABLE 表存放的地址存入特殊功能寄存器 DPTR 中，以便查找。

所以存表是指存入表的首地址，即表的地址指针或数据指针。地址指针指向第 1 行第 1 位控制码的位置。总之，地址指针指向哪个位置就到哪个位置读取控制码。

02：将寄存器 A 中的内容清除为 0。

03：查表读取控制码。“MOVC A, @A+DPTR”是一条很重要的查表语句，它将 A+DPTR 指定的表地址的数据送入寄存器 A 中。当程序第一次运行时，地址指针指向的位置是表中首位放置的控制码 OFEH，将 OFEH 取出并存入寄存器 A 中。

04：判断读取出的控制码是否为结束码 01H，如果不是则跳转 LOOP1 处。CJNE 指令的功能是比较两个操作数，若不相等就跳转，只有相等时才继续向下运行。01H 是结束标志码，放在表中的最后位置。

如果寄存器 A 中的数不是 01H，即不相等，说明表还没有查完，程序跳转到 LOOP1 处继续查表读取出控制码并输出。

若寄存器 A 中的数是 01H，即两个操作数相等了，说明表已经查完一遍，程序执行下一条语句。

05：程序跳转到标号 START 处，即跳转到程序开始处，重新设定显示值。

06：将取来的控制码送入 P1 并输出。

07：调用 0.5s 延时子程序。

08：数据指针加 1，指向下一个操作码。INC 指令的功能是使寄存器 A 中的数据加 1，即地址指针向下移动一个位置，指向第二个控制码，为下一次取码做准备。

09：跳转到 LOOP 处，重新开始，不断循环。

10~16：0.5s 延时子程序。

第 18~32 行代码为控制码编制表，表名为 TABLE。

18：单灯跳跃左移控制码。

20~23：两只灯从左向右移动控制码。

25：3 只灯左右移控制码。

27：4 只灯交互点亮控制码。

亮灯的移动是由控制码控制的，控制码采用十六进制数，如果参照表 6.1 将十六进制码

换成二进制码就可以清楚地看出灯是怎样被控制的。

29 ~ 31 : 闪烁 6 次控制码。控制码 00H 表示 8 个灯都亮 (0 表示亮) ; 控制码 0FFH 表示 8 个灯都灭 (1 表示灭) 。一亮一灭形成一次闪烁 , 共 6 对控制码 , 闪烁 6 次。

32 : 结束控制码。结束码可以是任意的数 , 放的位置要在表的最后 , 标志控制码已经结束。此时 , 由于第 04 句语句的作用 , 当寄存器 A 中的数是 01H , 即两个操作数相等 , 程序将执行下一条语句 , 即跳转到标号 LOOP 处 , 程序重新开始。

如果没有结束码 , 当取表码结束后 , 程序运行下一行 END 指令 , 使整个程序结束。

此外 , 制表时要注意 , 每一行控制码的前边都要加伪指令 DB , DB 是定义字节的指令。

33 : 程序结束。

4. 边用边学指令

本节使用新的指令有 : MOVC、INC、CLR、CJNE 和 DB。

- MOVC : MOVC 和前边经常用到的 MOV 指令都属于数据传送类指令 , 基本功能是传送数据 , 但使用上有所区别。MOVC 的功能是将表中查到的一位组数据送到寄存器 A 。常用形式如 “ MOVC A, @A+DPTR ” , 意思是将 A+DPTR 指定的内存地址的数据传送到寄存器 A , 所以也称查表指令 MOVC 。

- INC : 是算术运算类指令。算术运算类指令包括有加、减、乘和除法指令 , 其中 INC 指令的功能是使累加器加 1 。

- CLR : 是属于位移类指令。CLR 的功能是将累加器 A 内清 0 。

- CJNE : 是属于控制转移类指令。CJNE 与前边介绍的 DJNZ 指令都是有条件转移指令 , DJNZ 的条件是寄存器减 1 不为 0 转移 , 一般是与寄存器配合使用 , 用来控制已知循环次数的循环结构程序中。而 CJNE 的条件是对两数进行比较 , 不等转移 , 多用在分支结构的程序中。

- DB : 是伪指令。功能是从指定的地址单元开始 , 定义若干个字节作为内存单元的内容。

5. 本章用过的指令归类

- 数据传送类指令 : MOV、MOVC 。
- 算术运算类指令 : INC 。
- 逻辑运算及位移类指令 : RL、RR、CLR 。
- 控制转移类指令 : JMP、DJNZ、ACALL、RET、CJNE 。
- 位操作类指令 : CPL 。
- 伪指令 : END、DB 。

6.9.3 模拟仿真

本节模拟仿真时主要注意观察取控制码的过程。

运行程序第 1 行语句时 , 把表 TABLE 的首地址存入到 DPTR 寄存器。

注意 , 在特殊功能寄存器窗口中看不到 DPTR , 而是看到两个独立的 8 位寄存器 DPH 和 DPL , DPTR 寄存器是由 DPH 和 DPL 组成。此时 DPL=1C , 此地址是表的开头第 1 个控制码

的地址，也称地址指针或数据指针，即该指针指向表中的第1个控制码0FEH。

运行程序第2行语句时将寄存器A将清0。

运行程序第3行语句时，由于在第3行语句中是将A和DPTR两个寄存器内容相加，此时A为0，相加后仍为原DPTR的地址，此地址指针指向的是第1个控制码0FEH，所以，送入寄存器A的控制码是0FEH。

接下来再运行第6行语句时将控制码0FEH送入P1并输出。

运行到第8行语句“INC DPTR”时，地址指针加1即 $1CH+01H=1DH$ ，所以DPL=1D，此地址指针指向的是第2个控制码0FBH，依次不断循环。

6.9.4 实例测试

将写入程序的单片机插入实验板接通电源后，会看到单灯向左跳移1次，双灯向右移动2次，3只灯左右移动3次，4只灯交互点亮，8只灯闪烁6次，不断循环。

如果将程序改动一下，把表TABLE中的结束码取消，再在实验板上运行程序，会看到亮灯在进行闪烁后，程序马上停止，不再循环。

这是因为当表查完后，程序要执行下一个语句“END”，END是程序结束指令，所以程序就此结束。

如果想查完一遍表后程序不立即结束，需要在表中最后一行设置一个结束标志码，当程序取到该结束标志码时，通过判断语句，如“CJNE A, #01H, LOOP1”的作用，使程序不运行结束语句END，而跳转执行其他程序段。

6.9.5 经验总结

利用查表方法编写程序，可以很方便地完成一些复杂的控制功能。

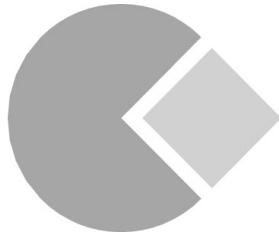
首先通过DB伪指令把控制码制成一个表TABLE，然后再通过查表的指令查表取码。用于查表的指令有两条：

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

其中使用DPTR作为基址来查表比较简单，可通过两步操作来完成。

(1) 利用“MOV DPTR, #TABLE”语句，将所查表的首地址存入DPTR数据指针寄存器。

(2) 再利用“MOVC A,@A+DPTR”查表指令，就可以到数据指针所指的表格内读取出数据。



第 7 章 LED 显示

LED 数码管是单片机控制系统中最常见的显示器件之一，一般用来显示处理结果或输入/输出信号的状态。

7.1 数码管工作原理及显示码

7.1.1 LED 数码管结构

LED 数码管显示器由 8 个发光二极管中的 7 个长条形发光二极管（称 7 笔段）按 a、b、c、d、e、f、g 顺序组成“8”字形，另一个点形的发光二极管 dp 放在右下方，用来显示小数点用，如图 7.1 (a) 所示。

数码管按内部连接方式又分为共阳极数码管和共阴极数码管两种。在内部 8 个发光二极管的阳极连在一起接电源正极，就称为共阳极数码管，如图 7.1 (b) 所示；8 个发光二极管的阴极连在一起接地，则称为共阴极数码管，如图 7.1 (c) 所示。

共阳极数码管引脚如图 7.2 所示，外部有 10 个引脚，其中 3、8 引脚连通，作为公共端，接电源正极。

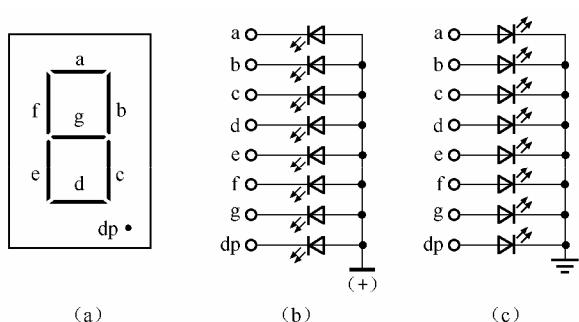


图 7.1 LED 数码管电路图

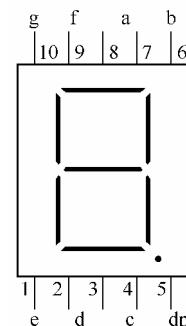


图 7.2 共阳极数码管引脚

7.1.2 工作原理

从 LED 数码管结构可以看出，不同笔段的组合就可以构成不同的字符，例如当笔段 b、

c 被点亮时，就可以显示数字 1；又如当笔段 a、b、c 被点亮时，就可以显示数字 7；当笔段 a、b、c、d、e、f、g 都被点亮时，就可以显示数字 8。只要控制 7 个发光二极管按一定的要求亮与灭，就能显示出十六进制 0~F。

7.1.3 数码管显示码

数码管显示码是表述二进制数与数码管所显示字符的对应关系的。例如，从图 7.1(b) 共阳极数码管可以看出，由于 8 个发光二极管的阳极已连在一起接到电源正极，所以，只要其负端 a、b、c、d、e、f、g 接地，发光二极管就会亮。

如果将负端接电源正极，由于两端都接到电源正极，没有电位差，所以就没有电流流过，发光二极管不会亮。0 和 1 即代表电平的高低，可以组成 8 位二进制数，与 8 个发光二极管的负端 a、b、c、d、e、f、g 相对应，如表 7.1 所示。

表 7.1 数码管显示码

字符	D _b	g	f	e	d	c	b	a	共阳笔段码	共阴笔段码
0	1	1	0	0	0	0	0	0	C0H	3FH
1	1	1	1	1	1	0	0	1	F9H	06H
2	1	0	1	0	0	1	0	0	A4H	5BH
3	1	0	1	1	0	0	0	0	B0H	4FH
4	1	0	0	1	1	0	0	1	99H	66H
5	1	0	0	1	0	0	1	0	92H	6DH
6	1	1	0	0	0	0	1	0	82H	7DH
7	1	1	1	1	1	0	0	0	F8H	07H
8	1	0	0	0	0	0	0	0	80H	7FH
9	1	0	0	1	0	0	0	0	90H	6FH
A	1	0	0	0	1	0	0	0	88H	77H
B	1	0	0	0	0	0	1	1	83H	7CH
C	1	1	0	0	0	1	1	0	C6H	39H
D	1	0	1	0	0	0	0	1	A1H	5EH
E	1	0	0	0	0	1	1	0	86H	79H
F	1	0	0	0	1	1	1	0	8EH	71H

例如，如果想让数码管显示 0，只要将 dp 和 g 两个发光二极管置高电平，而其他 6 个发光二极管置低电平即可。

在表中正好与 0 相对应的是 11000000，用 16 进制码表示为 C0H。再如想让数码管显示 8，查表可知，与 8 相对应的是 10000000，用 16 进制码表示为 80H，在程序中输入数据 10000000，与输入 80H 是同样效果，前者是 8 的二进制码，后者是 8 的 16 进制码。

7.2 让数码管静态显示 6

功能说明：让数码管静态显示数字 6。

7.2.1 硬件设计

电路设计如图 7.3 所示。

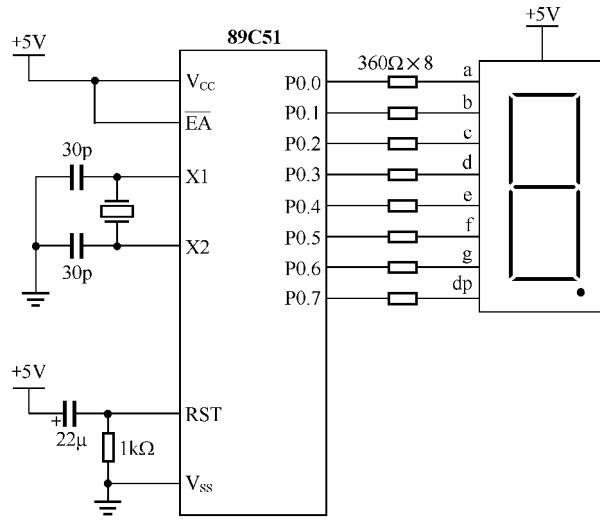


图 7.3 一位数码管显示电路

单片机 P0 端口接有一只共阳极数码管显示器，其中 3、8 引脚是阳极公共端，接到+5V，其他 8 个引脚依照 a、b、c、d、e、f、g、dp 顺序，依次与 P0 端口的 8 个引脚 P0.1、P0.2、P0.3、P0.4、P0.5、P0.6、P0.7 相连，R 是限流电阻。

7.2.2 程序设计

本程序利用取表的方法，控制数码管显示器的输出。

1. 流程图

程序设计流程如图 7.4 所示。

2. 程序

汇编语言编写的让数码管静态显示 6 源程序 SMG1.ASM 代码如下：

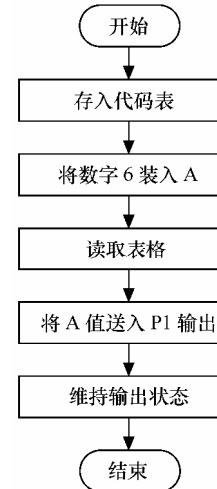


图 7.4 程序流程

```

01: START:    MOV      DPTR, #TABLE           ; 存入表的起始地址
02:           MOV      A, #6                  ; 将欲显示的数字 6 存入 A
03:           MOVC    A, @A+DPTR            ; 按地址取代码并存入 A
04:           MOV      P0, A                ; 将代码送 P0 转变为数字显示
05:           JMP      $                  ; 程序运行在当前状态
06: TABLE:     DB      0C0H, 0F9H, 0A4H, 0B0H ; 代码表
07:           DB      99H, 92H, 82H, 0F8H

```

```

08:           DB      80H,  90H,   88H,   83H
09:           DB      0C6H,  0A1H,   86H,   8EH
10:          END               ; 程序结束

```

7.3.3 代码详解

1. 标号说明

START：起始程序的进入点。

TABLE：表的进入点。

2. 寄存器使用分配情况

DPTR：特殊功能寄存器，主要作用是存放地址指针，即指向表的开始地址。

A：特殊功能寄存器，又称累加器，是每个程序中不可缺少的寄存器。

P0：对外是输入/输出端口，对内是寄存器。

3. 程序分析解释

01：存入表的起始地址。

02：将欲显示的数字 6 存入 A。

03：按地址取代码并存入 A。在“MOVC A, @A+DPTR”读取表语句中，A+DPTR 是地址，其中 DPTR 存入的是基准地址，是不变的，始终指向表头即表的第一位码，而累加器 A 中的值是可变的。

所以，取表中哪一个代码主要取决于 A 的值，如 A 的值是 6，A+DPTR 就意味着地址指针在原来指向表头的基础上，再移动 6 位，即指向第 7 位，从表中可以看出第 7 位代码是 82H。

再通过 MOVC 指令，把 82H 存入累加器 A，要注意，无论累加器 A 内原来是什么值，都将被 82H 所覆盖，即此时 A 中的值是 82H。

04：将代码送到 P0 转变为数字显示。由于代码 82H 所对应的字符是 6，所以，将 82H 送到 P0 后将转变为数字 6 输出显示。

05：使程序保持在当前“6”的输出状态。

4. 复习学过指令

本程序使用的指令有：MOV、MOVC、JMP、DB，都是以前学过的。

MOV 和 MOVC 都是数据传送类指令。MOVC 的功能是将程序区段复制一组数据到累加器 A，在传送表中的代码数据时要使用 MOVC，而不是 MOV。

DB 是伪指令。功能是从指定的地址单元开始，定义若干个字节作为内存单元的内容，也就是说在每行编码前边都要用 DB 伪指令，否则程序在编译时将不识别该行编码。

JMP 是无条件控制转移指令。\$代表当前处，“JMP \$”语句使程序运行在当前状态下。

7.2.4 模拟仿真

模拟仿真方法请参阅第 6.2 节。本节模拟仿真主要注意观察寄存器 DPTR、A 和 P0 值的变化，其中 DPTR 是由两个 8 位寄存器 DPH 和 DPL 所组成的 16 位寄存器。在模拟仿真窗口中 DPTR 是由 DPH 和 DPL 两个独立的 8 位寄存器来体现的。

按跟踪的快捷键 F7，当程序运行第一行语句后，会看到 DPL 的值由 00 变为 0A，这是指向表头的地址，即指向表中第一位代码 0C0H 的位置。

再按 F7 键使程序运行第二行语句，这时累加器 A 的值由 FE 变为 6。

按 F7 键，程序运行第 3 行语句后，累加器 A 的值由 6 又变为 82，即程序表中的第 7 位代码“82H”。

当程序运行到第 4 行语句，P0 值由 00 变为 82，代码“82H”所对应字符是数字 6，所以数码管显示 6。

再按 F7 键，程序在原地不动，这是程序语句“LJMP \$”中“\$”的作用。

7.2.5 实例测试

首先要注意实验板上单片机插座与数码管相连的端口是否和程序指定的端口相一致，如果不一致，要将原程序指定的端口改为实验板上与数码管相连的端口。

如果电路上数码管的公共阳极是由三极管控制的，请参照本章第 7.4 节电路，在编程时在程序第 04 和 05 行之间加“CLR P2.0”语句，作用是为公共阳极提供+5 伏电压，使数码管开通。

程序经过模拟仿真通过后，用编程器将程序写入单片机内。

开通电源，就会看到数码管显示数字 6。可以改变程序第二行语句中的数字，重新“编译/汇编”和“输出 HEX 文件”后，再通过编程器写入单片机内重新测试。

7.2.6 经验总结

单片机控制数码管输出时，利用取表的方法编程非常方便。

将数码管显示器的显示码编成一个表，利用“MOV DPTR, #TABLE”语句存表，再利用“MOVC A, @A+DPTR”语句将显示码从表中读取出来，并通过“MOV P0, A”语句转换成字符输出。这 3 条语句是利用取表编程中最基本语句，在取表编程中都要用到。

7.3 循环显示 0~9

功能说明：用一位数码管循环显示数字 0~9，数字变换的间隔时间为 1s。本节硬件设计与 7.2 节相同。

7.3.1 程序设计

本程序是在上节的基础上，增加了判断循环和延时两部分。延时是为了使显示的字符能看清楚，如果在字符变换中间不加延时，字符显示就会模糊不清；判断循环部分保证字符有序移动。

1. 流程图

程序设计流程如图 7.5 所示。

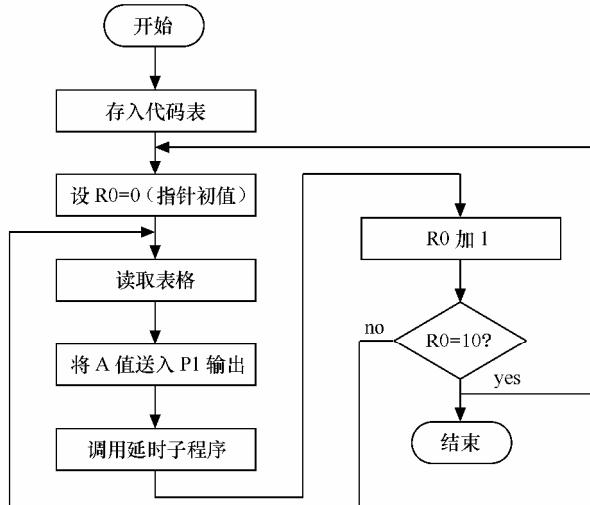


图 7.5 程序流程

2. 程序

汇编语言编写的循环显示 0~9 源程序 SMG2.ASM 代码如下：

```

01: START: MOV      DPTR, #TABLE           ; 存表
02:          MOV      R0, #0                 ; 设定初始值
03: LOOP:   MOV      A, R0
04:          MOVC    A, @A+DPTR           ; 取表代码
05:          MOV      P0, A               ; 送 P0 输出
06:          ACALL   DLY1S              ; 调用延时程序
07:          INC     R0                 ; R0 值加 1
08:          CJNE    R0, #10, LOOP         ; 不是 10，循环
09:          JMP     START              ; 重新开始
10: DLY1S:  MOV      R5, #50              ; 1s 延时子程序
11: D1:     MOV      R6, #100
12: D2:     MOV      R7, #100
13:          DJNZ    R7, $
14:          DJNZ    R6, D2
15:          DJNZ    R5, D1
16:          RET
17: TABLE:  DB      0C0H, 0F9H, 0A4H, 0B0H ; 代码表
18:          DB      99H, 92H, 82H, 0F8H
19:          DB      80H, 90H, 88H, 83H
20:          DB      0C6H, 0A1H, 86H, 8EH
21:          END                ; 程序结束
  
```

7.3.2 代码详解

1. 标号说明

START：起始程序的进入点。
LOOP：循环执行的进入点。
DELAY：延时子程序的进入点。
TABLE：表的进入点。

2. 寄存器使用分配情况

DPTR、A 和 P0 与上节作用相同。R0、R5、R6、R7 为计数器，其中 R0 是显示字符位移的计数器，其他是延时计数器。

3. 程序分析解释

01~06：存表和读取表显示。其中 06 语句是调用延时子程序，使显示的字符停留 1s 后再变化。

07~09：判断循环次数。其中 07 语句“INC R0”的作用是使每移动一位（更换一个字符）后 R0 值加 1。08 语句能判断是否移动了 10 次，如果 R0 计数器的值不是 10，意味着还没完成 0 到 9 的显示，程序运行需要跳转到循环的进入点标号 LOOP 处继续循环。

10~16：延时 1s 子程序。

17~20：代码表，可对照表 7.1 来理解。

21：程序结束。

4. 复习学过指令

本程序使用的指令有：MOV、MOVC、JMP、DJNZ、CJNE、ACALL、RET、INC、DB、END，这些都是以前学过的，注意它们的使用特点。

JMP、DJNZ、CJNE、ACALL、RET 都属于控制转移类指令，ACALL 与 RET 在程序中是成对出现，ACALL 的作用是调用，调用子程序；RET 的作用是返回，返回子程序。

JMP 是无条件转移指令，不需要任何条件，进行跳转。

DJNZ 与 CJNE 是有条件转移指令，跳转需要一定的条件，如 DJNZ 是在寄存器减 1 后不为 0 时跳转；而 CJNE 是比较转移指令，即寄存器 A 与立即数相比较，不等时跳转。

7.3.3 模拟仿真

本节模拟仿真注意观察程序运行 07~09 行时 R0 值的变化以及程序运行的转移过程，从而加深对指令 INC、CJNE、DJNZ 和 JMP 的理解。

7.3.4 实例测试

参照上节，检查无误后开通电源，就会看到数码管显示器字符由 0~9 不断循环显示。

可以将程序 08 行语句中的数字“#10”改为“#16”，再重新“编译/汇编”和“输出 HEX 文件”后，通过编程器写入单片机内重新测试，这时，会看到数码管显示器由原来 0~

9 循环显示，改为 0~F 循环显示。

7.3.5 经验总结

本程序是在上节程序的基础上，增加了判断循环和延时两部分。其中，判断循环部分使程序更加智能，使程序能自动判断是否完成由 0~9 的显示，并不断循环。

本程序表中的代码是 C0H~8EH，所对应的数码管能显示字型为 0~F，其对应关系如表 7.1 所示。

7.4 两位数码管显示 00~99

功能说明：使用两位数码管显示器，利用扫描方式循环显示两位数 00~99。

7.4.1 硬件设计

电路设计如图 7.6 所示。

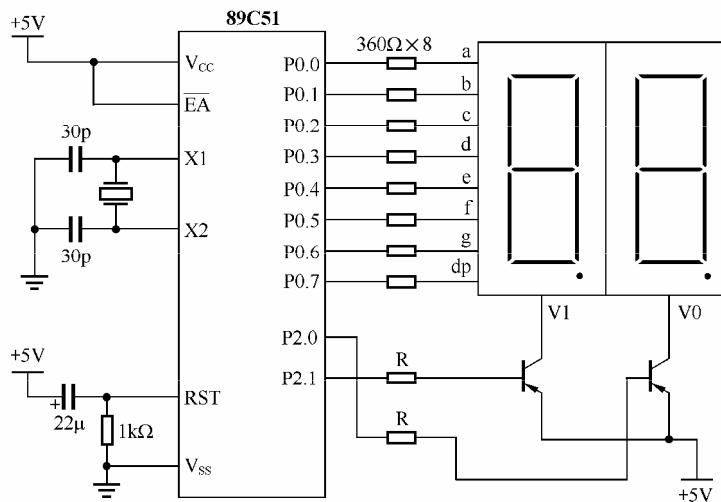


图 7.6 数码管动态显示电路

单片机 P0 端口并行连接两只共阳极数码管显示器。数码管的 8 个引脚依照 a、b、c、d、e、f、g、dp 顺序依次与 P0 端口的 8 个引脚 P0.1、P0.2、P0.3、P0.4、P0.5、P0.6、P0.7 相连，R 是限流电阻。

V0 和 V1 是两只共阳极数码管的控制三极管，三极管的基极分别接在单片机 P2 端口的 P2.0 和 P2.1 引脚上。也就是说，P2.0 输出为 0 时三极管 V0 导通，与其相连的共阳极数码管显示器开始工作；P2.0 输出为 1 时三极管 V0 截止，与其相连的数码管显示器停止工作。三极管 V1 的作用与 V0 三极管相同。

7.4.2 程序设计

本程序使用两位数码管显示器，利用扫描方式显示循环 00~99。程序设计在上节的基础

上，增加了扫描显示部分和十进制转换部分。

1. 流程图

主程序设计流程如图 7.7 所示。

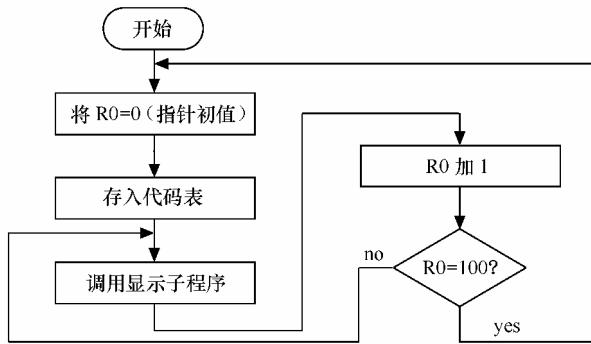


图 7.7 主程序流程

2. 程序

汇编语言编写的两位数码管显示 00 ~ 99 源程序 SMG3.ASM 代码如下：

01:	START:	MOV	R0, #0	; 初始化计数器
02:		MOV	DPTR, #TABLE	; 存入查表启始地址
03:	LOOP:	ACALL	DISPLAY	; 调用显示子程序
04:		INC	R0	; 计数器加 1
05:		CJNE	R0, #100, LOOP	; 没到 100
06:		JMP	START	; 跳转到开始处
07:	DISPLAY:	MOV	A, R0	; 扫描子程序
08:		MOV	B, #10	; 16 进制换成 10 进制
09:		DIV	AB	; A ÷ B 的商存入 A，余数存入 B
10:		MOV	R1, A	; R1 存放十位数
11:		MOV	R2, B	; R2 存放个位数
12:		MOV	R3, #50	; 设导通频率为 50
13:	LOOP1:	MOV	A, R2	; 个位数显示
14:		ACALL	CHANG	; 调用显示子程序
15:		CLR	P2.0	; 开个位显示
16:		ACALL	DLY10mS	; 调用延时 10ms 程序
17:		SETB	P2.0	; 关闭个位显示
18:		MOV	A, R1	; 取十位数
19:		ACALL	CHANG	; 调用取表显示子程序
20:		CLR	P2.1	; 开十位显示
21:		ACALL	DLY10mS	; 调用延时 10ms 程序
22:		SETB	P2.1	; 关闭十位显示
23:		DJNZ	R3, LOOP1	; 100 次显示未完继续扫描
24:		RET		; 扫描子程序返回

```

25: CHANG: MOVC    A, @A+DPTR      ; 取表子程序
26:        MOV     P0, A
27:        RET      ; 取表子程序返回
28: DLY10ms: MOV     R6, #20        ; 10 ms 延时子程序
29: D1:      MOV     R7, #248
30:      DJNZ    R7, $
31:      DJNZ    R6, D1
32:      RET      ; 延时子程序返回
33: TABLE:   DB      0C0H, 0F9H, 0A4H, 0B0H ; 代码表
34:           DB      99H,   92H,   82H,   0F8H
35:           DB      80H,   90H,   88H,   83H
36:           DB      0C6H, 0A1H, 086H, 08EH
37:        END      ; 程序结束

```

7.4.3 代码详解

1. 标号说明

START : 起始程序的进入点。
 LOOP : 循环执行的进入点。
 DISPLAY : 显示子程序进入点。
 LOOP1 : 个位和十位显示的进入点。
 CHANG : 取表显示子程序的进入点。
 DLY10ms : 延时 10ms 子程序的进入点。
 TABLE : 代码表的进入点。

2. 寄存器使用分配情况

DPTR、A 和 P0 与上节作用相同。
 R0 : 进位计数器。初始为 0，计数到 100 时再从 0 开始，从而保证显示器进行 00 ~ 99 的循环显示。
 R1 : 十位数的存放处。
 R2 : 个位数的存放处。
 R3 : 扫描频率计数器。
 R6、R7 : 延时子程序中的计数器。
 P2.0、P2.1 : P2 端口的两个引脚，用来对电路中控制公共阳极数码管的三极管进行导通和关闭控制。
 B : 一种特殊寄存器，主要用来做乘法指令及除法指令的运算，不做乘、除法时，也可以作一般用途的寄存器使用。

3. 程序分析解释

01 ~ 06 : 主程序。

主程序主要完成 4 项工作：初始化计数器（01 语句），即计数器由 0 开始计数；存入表的启始地址（02 语句）；调用显示子程序（03 语句），使数码管显示器进行 0~99 的循环显示；判断选择（04~06 语句），如果显示未到 100 次将转移到 LOOP 处继续循环，执行到 100 次后则重新开始。

07~24：显示子程序。在显示子程序里完成了转换十进制、个位显示、十位显示、调延时 10ms 程序、调用表显示子程序、扫描等工作。其中转换十进制语句中，语句“DIV AB”是指将 $A \div B$ 的余数存入寄存器 B，商存入寄存器 A。

下边举例说明，假如计数器 R0 存的数是 9，即不足 10，寄存器 B 存的数是 10，执行“DIV AB”语句时，因为商不够整数 1，所以将 0 存入累加器 A，余数 9 存入寄存器 B，再通过 10、11 语句，将 0 存入 R1，将余数 9 存入 R2。

假如 R0 存的数是 19，寄存器 B 存的数是 10，执行“DIV AB”商数是 1，将 1 存入累加器 A，余数是 9，将 9 存入寄存器 B，再通过 10、11 语句，将 1 存入 R1，将余数 9 存入 R2，这样就完成了十进制转换。

个位和十位显示：个位显示语句是将 R2 中存放的个位数传送到累加器 A，再调用表显示子程序（14 语句），这时还不能显示，还必须使数码管公共阳极的控制三极管 V0 导通，

V0 是由 P2 端口中的 P2.0 控制，通过“CLR P2.0”语句，使 P2.0 为 0，开启个位显示，此时 P2.1 为 1，十位数码管处于关闭状态。经过 10ms 延时，再通过“SETB P2.0”语句，使 P2.0 为 1，关闭个位显示，开始进行十位显示，十位显示的语句结构与个位显示相同。

扫描：扫描方式是指让两位数码管轮流工作，即个位显示时，十位关闭；十位显示时，个位关闭。这样做的目的一是省电，二是节省输出端口。在静态显示时一个端口只能接一个数码管显示器，而利用扫描方式一个端口可以接几个数码管显示器，由数码管公共阳极的控制三极管决定哪只数码管显示。

两位数码管是轮流工作的，可是人眼却感到同时在亮，这是由于人眼的视觉暂留，只要足够快，就能够形成同时亮的感觉。“MOV R3, #50”语句确定了在每位显示的时间内轮换导通 50 次，每次显示都延时 10ms，轮换导通 50 次后新进一位，否则继续轮换导通。

25~27：取表子程序。

28~32：延时 10ms 子程序。

33~36：显示控制码表。

37：程序结束。

4. 边用边学指令

本节用到新的指令有：DIV、CLR 和 SETB。

- DIV：算术运算类指令中的除法指令。“DIV AB”是指 $A \div B$ 的余数存入寄存器 B，商存入累加器 A。
- CLR 和 SETB：位操作类指令。其中 CLR 将指定位清除为 0；SETB 将指定位设定为 1。

5. 本章用过的指令归类

- 数据传送类指令：MOV、MOVC。

- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、CLR。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE。
- 位操作类指令：CPL、CLR、SETB。
- 伪指令：END、DB。

7.4.4 模拟仿真

本例在模拟仿真时首先使程序运行跳过调用延时语句，办法是在调用延时语句前加上分号，另外再将语句 12 中的“#50”改为“#2”。上述做法的目的是为了节省程序模拟运行的时间。重新将程序“编译/汇编”和“产生代码并装入”调试。

在模拟仿真中注意观察两个方面：一是程序运行路线，该程序在主程序中调用显示子程序，在显示子程序中又调用延时子程序和取表显示子程序等，注意观察子程序的调用和返回；二是注意观察寄存器以及特殊功能寄存器 R0、R1、R2、A、B 和 P0 及 P2 的变化。

例如，当模拟仿真中从寄存器窗口看到 R0 的值为 3 时，R1 的值为 0，R2 的值为 3。P0 一会显示 C0，一会显示 B0。C0 是代码表中的第一位代码 0C0H，相对应的字符是 0，由于两个数码管显示器都接在端口 P0 上，两个数码管显示器都会得到数据 0，但由于此时个位数码管是在关闭状态，所以只有十位的数码管能显示 0。

当 P0 显示 B0 时，B0 是代码表中的第 4 位代码 0B0H，相对应的字符是 3，两个数码管显示器也都会得到数据 3，但此时十位数码管是在关闭状态，所以只有个位的数码管才显示 3。由于人眼的视觉暂留作用，感觉不到是在轮换显示，所以十位显示为 0，同时个位显示为 3。

7.4.5 实例测试

将单片机插入实验板插座上并锁紧，检查无误后开通电源，就会看到数码管显示器字符由 00~99 不断循环显示。

数码管显示字符的亮度及清晰度与每位点亮停留时间和每位显示的时间内轮换导通次数有关。可改变程序中的延时时间进行测试，观察延时时间与亮度关系。

7.4.6 经验总结

LED 数码管显示器的显示方法分静态和动态两种。本章第 7.2 节和第 7.3 节介绍的例子属于静态显示。静态显示是指显示某一字符时，相应的发光二极管恒定导通或截止，这种方法，每一显示位都需要由一个 8 位输出口控制，占用硬件较多，一般仅用于显示位数较少的场合。

动态显示是指一位一位地轮流点亮各位显示器。本节的例子是采用了动态显示方法，即扫描方法，每一位每隔一段时间点亮一次，点亮时间和间隔时间的比例与亮度有关。由于动态显示节省硬件，所以多位显示时常常采用动态显示方法。

本程序在结构上还采用了子程序嵌套的方法，在显示子程序 DISPLAY 里嵌套取表子程序 CHANG 和延时子程序 DLY10ms。



第8章 键盘控制

单片机具有人机对话功能，开关、键盘就是实现人机对话的主要输入设备，通过它能随时发出各种控制指令和数据到单片机。本章介绍使用指拨开关 DIP、独立式按键开关及键盘作为输入信号控制输出端的方法。

8.1 用 8 位 DIP 开关控制 LED

功能说明：用 8 位指拨开关 DIP 作单片机的输入，控制输出端口连接的 8 只 LED。如果 DIP1 开关为 ON 则 LED1 被点亮。

8.1.1 硬件设计

本例单片机开关输入电路如图 8.1 所示。

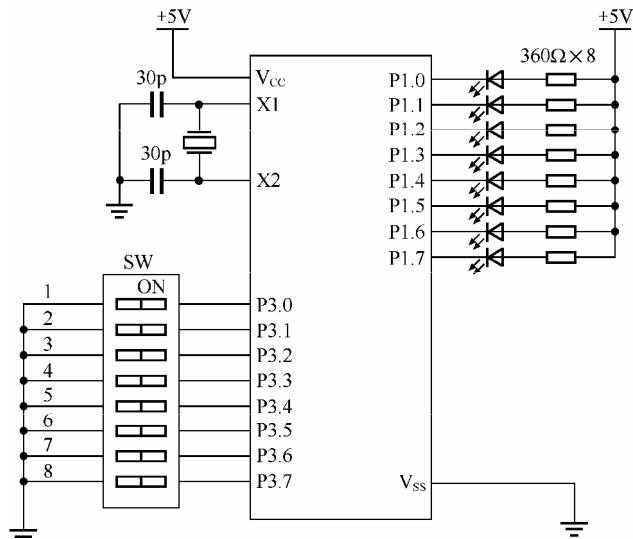


图 8.1 DIP 开关输入基本电路

单片机 P3 端口接有一组 8 个 DIP 拨动开关，开关的另一端接地。当开关向右拨动(ON)

时，P3 端口与地相接，此时端口为低电平；当开关向左拨动（OFF）时，P3 端口与地断开，此时端口为高电平。该组开关与 P1 端口所接的 8 只 LED 发光二极管一一对应。

8.1.2 程序设计

这是一个简单的程序，开关信号从 P3 端口输入，控制 P1 端口的输出。

1. 流程图

程序设计流程如图 8.2 所示。

2. 程序

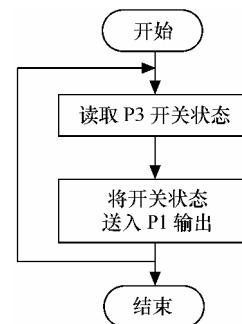


图 8.2 程序流程

汇编语言编写的用 8 位 DIP 开关控制 LED 源程序 SR01.ASM 代码如下：

01:	LOOP:	MOV	A, P3	; 从 P3 读入 DIP 开关值
02:		MOV	P1, A	; 从 P1 输出
03:		JMP	LOOP	; 无穷循环
04:		END		; 程序结束

8.1.3 代码详解

1. 标号说明

LOOP：循环执行的进入点。

2. 寄存器使用分配情况

本程序使用的寄存器有 A、P1 和 P3。寄存器 A 和 P1 寄存器的作用与前面各章相同。寄存器 P3 对外是输入/输出端口，对内是寄存器。

3. 程序分析解释

01：从 P3 读入 DIP 开关值。DIP 开关值取决于开关所在的位置，例如，将 DIP1 开关拨向右端（ON）位置，此时 P3.0（P3 端口的第 1 个引脚）与地相接，引脚 P3.0 为低电，即开关 DIP1 值为 0。

其他开关位置都在左侧，单片机的引脚与地断开，故引脚为高电平。用二进制码表示为 11111110，此值通过执行“MOV A,P3”语句将其输入到寄存器 A 中。

02：通过执行“MOV P1,A”语句，将 A 值 11111110 送到 P1 并输出，使 P1 端口的第一只灯点亮。

03：程序通过执行“JMP LOOP”语句，跳转到程序标号 LOOP 处执行，即重新读入 DIP 开关值，这样不断循环。如果读入的开关值没变，仍然是 11111110，那么还是第一只灯亮。

04：程序结束。

8.1.4 模拟仿真

本程序在进行模拟仿真时，需要在原程序前增加一行模拟开关语句，如“MOV P3, #11111110B”，增加语句后程序如下。

```

01:           MOV    P3, #11111110B
02: LOOP:      MOV    A, P3          ; 从 P3 读入 DIP 开关值
03:           MOV    P1, A          ; 从 P1 输出
04:           JMP    LOOP         ; 无穷循环
05:           END                ; 程序结束

```

二进制数 11111110 表示 8 个开关位置，0 表示开关拨向右端（ON）位置，即第 1 个开关已经接通；1 表示开关拨向左端（断开）位置。改变任意一个二进制数，就相当于拨动对应的开关，P1 值也会发生相应变化，表示 P3 的输入信号控制着 P1 的输出。

8.1.5 实例测试

将写入程序的单片机插入实验板插座上，检查无误后接通电源。拨动接在 P3 端口的 DIP1 开关，会看到接在 P1 端口的相应 LED 发光二极管被点亮或熄灭。例如，若 DIP1 开关为 ON，则输出端 LED1 亮；若 DIP2 开关为 ON，则输出端 LED2 亮。

8.1.6 经验总结

实现用 DIP1 开关作输入来控制输出，关键是读入 DIP1 开关值，因为开关接在 P3 端口，所以 P3 端口值就是开关的值。

开关有两种状态，接通时为低电平，断开时为高电平。因为开关与 P3 端口直接连接，所以程序是通过检测 P3 各位电平来反映相应的开关状态的。

当检测到电平为 0 时，表示该位所接的开关处于接通状态；当检测到电平为 1 时，表示该位所接的开关处于断开状态。然后再将所检测到的 P3 的电平 0 或 1 读出，送入输出端 P1 输出，所以输出端 P1 的电平与开关状态是一致的。

8.2 用 4 位 DIP 开关控制数码管显示

功能说明：用 DIP 开关中的低 4 位为输入，控制输出端数码管显示器的输出。

8.2.1 硬件设计

电路设计如图 8.3 所示。

P3 为输入端口接有一组 8 个 DIP 拨动开关，开关的另一端接地。P0 为输出端口接有一位 LED 数码管显示器。

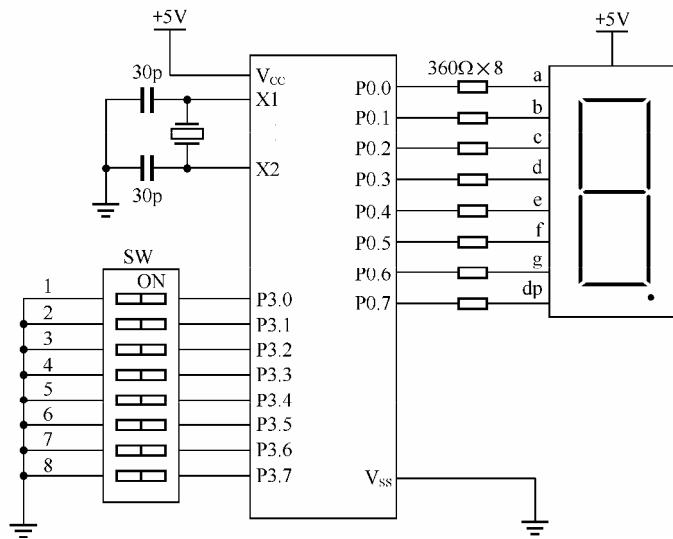


图 8.3 DIP 开关输入基本电路

8.2.2 程序设计

本程序是用低 4 位开关作二进制数从 P3 端口输入，去控制 P0 端口数码管显示。

1. 流程图

程序设计流程如图 8.4 所示。

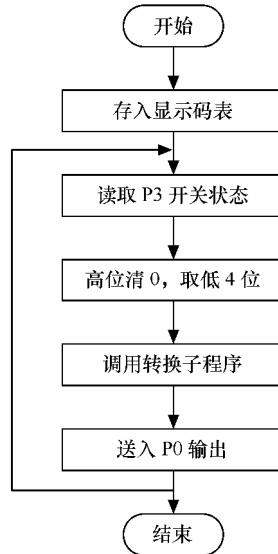


图 8.4 程序流程

2. 程序

汇编语言编写的用 4 位 DIP 开关控制数码管显示源程序 SR02.ASM 代码如下：

```

01:      MOV      DPTR, #TABLE           ; 存表
02:      MOV      P0, #0FFH             ; LED 全灭
03:  LOOP:   MOV      A, P3              ; 从 P3 口读入 DIP 开关值
04:      ANL      A, #0FH               ; 高 4 位清 0，取低 4 位
05:      ACALL   CHANG              ; 转成七段显示码
06:      MOV      P0, A               ; 从 P0 输出
07:      JMP      LOOP               ; 跳转到 LOOP 处循环
08:  CHANG:  MOVC    A, @A+DPTR        ; 转换显示码子程序
09:      RET                  ; 子程序返回
10:  TABLE:  DB      0C0H, 0F9H, 0A4H, 0B0H ; 显示码表
11:          DB      99H, 92H, 82H, 0F8H
12:          DB      80H, 90H, 88H, 83H
13:          DB      0C6H, 0A1H, 86H, 8EH
14:      END                  ; 程序结束

```

8.2.3 代码详解

1. 标号说明

LOOP：循环执行的进入点。
 CHANG：转换显示码子程序进入点。
 TABLE：显示码表的进入点。

2. 寄存器使用分配情况

本程序使用的寄存器有 A、P0、P3 和 DPTR，其中寄存器 A、P0 和 P3 的作用与上节相同。DPTR 是特殊功能寄存器，用来存放地址指针，即指向表的开始地址。

3. 程序分析解释

- 01：存表。将显示码表 TABLE 地址存入数据指针寄存器 DPTR，以便查找。
 - 02：LED 全灭。因为存入 P0 的值为 FFH，FFH=11111111B，所以 P0 中相对应的 8 位都是 1，LED 全灭。
 - 03：从 P3 口读入 DIP 开关值。8 个开关与 P3 端口 8 条引线一一相连，开关拨通时为低电平，开关断开时为高电平。由于开关与端口一一相连，所以开关值就是 P3 值，并将此值存入寄存器 A。
 - 04：高 4 位清 0，屏蔽高 4 位，取低 4 位工作。此语句是理解整个程序的关键。
- 下面举例说明，假如开关都在关的位置，这时我们把第一位开关开通，拨向右端 ON 位置，通过 03 行语句可知，A 中值为从 P3 口读入的 DIP 开关值 11111110，在 04 行语句“ANL A, #0FH”中，将两个二进制数 11111110 与 00001111 进行与运算，运算结果为 00001110，其中低 4 位可以转换一位十六进制数，对照附录 B 中表 B.1，二进制数 1110 为十六进制数 E，所以数码管显示为 E。

在 8 个开关中，只有前 4 个开关起作用，组成 4 位二进制数，控制一位数码管的显示输出。

- 05：调用转换显示码子程序，转成七段显示码。
- 06：从 P0 输出，即数码管显示一位十六进制数。
- 07：跳转到程序标号 LOOP 处执行，即重新读入 DIP 开关值，不断循环。
- 08 ~ 13：转换显示码子程序及显示码表。
- 14：程序结束。

4. 边用边学指令

本节程序新用到的指令为 ANL。

- ANL 是逻辑运算及位移类指令中的逻辑与指令。在单片机指令系统中，逻辑运算指令占有极其重要的作用，这些指令包括逻辑非（即取反运算）与、或、异或和循环移位操作等。

其中逻辑与指令 ANL 的意义是对目的操作数和源操作数按位进行与操作，将结果回送到目的操作数中，源操作数保持不变。

在这里首先需要理解与运算的特点，与运算的运算符为“ ”，其运算规则是：0 和 0、1 相与的结果均为 0，1 和 1 相与的结果为 1。

例如，当 A 为 11111110B 时，它与立即数 00001111B 相与，即执行指令

```
ANL A, #00001111B
```

相与的过程为：

$$\begin{array}{r} 11111110 \\ 00001111 \\ \hline 00001110 \end{array}$$

相与的结果值为 00001110B。

在二进制数与十六进制数转换中，4 位二进制数可代替 1 位十六进制数，换算关系可参照附录 B 中表 B.1。所以，将 8 位二进制数中的高 4 位清 0，保留低 4 位即可对应转换一位十六进制数。

8.2.4 模拟仿真

本程序进行模拟仿真时，与上节一样，需要在原程序 02 与 03 之间增加一行模拟开关的语句，如“MOV P3, #11111110B”，并注意观察寄存器 P0、P3、A 的变化以及 04 行语句“ANL A, #0FH”执行后各寄存器值的变化情况。

当运行模拟开关语句“MOV P3, #11111110B”后，从寄存器窗口看到 P3 值为 FE。

运行 03 行语句“MOV A, P3”后，A 值为 FE。

运行 04 行语句“ANL A, #0FH”后，A 中存的数 FE 与立即数 0F 进行相与逻辑运算。两个数转换成二进制数分别为 11111110 与 00001111，进行相与逻辑运算后，A 的值由 FE 变为 0E，即将高 4 位清 0，保留低 4 位。

运行 05 行语句，调用转换显示码子程序，进行取码，即数码管显示码的转换，这时会

看到 A 的值变为 86。

执行 06 行语句“MOV P0, A”后，输出端口 P0 的值也为 86。86H 是编写的数码管显示码表中的倒数第二个数，它对应十六进制数为 E。此时如果单片机在实验板上进行测试，数码管显示的数为 E。

8.2.5 实例测试

将写入程序的单片机插入实验板插座上，并检查实验板上 P3 端口所接的 DIP 拨动开关，使其低 4 位开关处于断开位置（高 4 位开关已经被程序屏蔽，不起作用），检查无误后接通电源，此时数码管显示 F。

因为 4 位开关处于断开位置时，编制的 4 位二进制数为 1111B，1111B 换算十六进制数为 F，故数码管显示器显示 F。

再接通第 1 个开关，此时编制的 4 位二进制数为 1110B，换算十六进制数为 E，所以数码管显示 E。可参照书后附录 B 编制二进制控制码，控制数码管显示器的输出。

8.2.6 经验总结

本程序实现在输入端用 4 位开关编制 4 位二进制控制码，控制数码管显示器的输出。此程序的 3 个关键步骤如下。

- (1) 读出开关值。
- (2) 通过逻辑与运算得出需要的二进制码。
- (3) 再将此码经过显示码转换子程序的转换，完成控制数码管的输出任务。

8.3 按键开关控制指示灯

功能说明：用两个按键开关 K1 和 K2 作输入，K1 为电源指示灯开关，K2 为工作指示灯开关。分别控制 P1.0 接的电源指示灯和 P1.7 接的工作指示灯的接通和关闭。接通电源时，电源指示灯处于点亮状态。

当按 K2 时，工作指示灯点亮，电源指示灯熄灭；按 K1 时，电源指示灯点亮，工作指示灯熄灭。

8.3.1 硬件设计

电路设计如图 8.5 所示。

输入部分由两个按键开关 K1 和 K2 组成，分别与 P2 端口中的 P2.4 和 P2.5 连接。K1 为电源指示灯开关，K2 为工作指示灯开关。输出部分为 P1.0 所接的电源指示灯和 P1.7 所接的工作指示灯。

8.3.2 程序设计

本程序为独立式按键的简单应用程序。

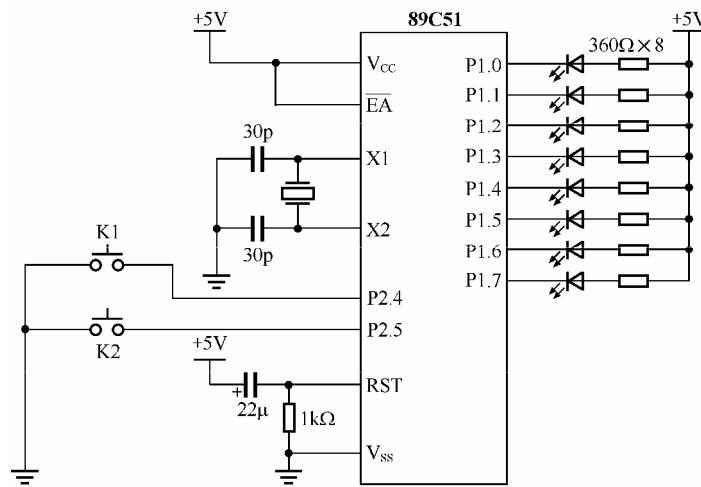


图 8.5 按键开关输入基本电路图

1. 流程图

程序设计流程如图 8.6 所示。

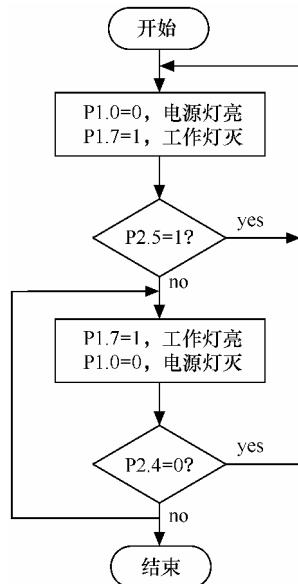


图 8.6 程序流程图

2. 程序

汇编语言编写的按键开关控制指示灯源程序 SR03.ASM 代码如下：

01: START: MOV	P1, #11111110B	; P1.0 所接 LED 亮
02: JB	P2.5, \$; 判断 P2.5 (K2 键是否为 1)
03: ON: MOV	P1, #01111111B	; P1.7 所接 LED 亮

04:	JNB	P2.4, START	; 判断 P2.4 (K1 键) 是否为 0
05:	JMP	ON	; 未按 K1 键，跳至 ON
06:	END		; 程序结束

8.3.3 代码详解

1. 标号说明

START：程序开始执行的进入点。

ON：保持工作指示灯亮的循环进入点。

2. 寄存器使用分配情况

本程序使用的寄存器有 P1 和 P2，其中 P2.4 和 P2.5 是 P2 寄存器中的第 4 位和第 5 位，即 P2 端口的第 4 条引脚和第 5 条引脚。

3. 程序分析解释

01：使 P1 端口的第 1 个灯亮，即接 P1.0 引脚的电源指示灯亮。

02：判断接在 P2.5 引脚上的电源指示灯开关 K2 是否被按下。K2 没被按下时 P2.5 位为 1，JB 指令的作用是当 P2.5 位为 1 时转移到\$处，\$表示当前状态，即保持电源指示灯亮的状态。当 K2 被按下时，P2.5 位为 0，程序执行下一行语句，使工作指示灯亮。

03：使 P1 端口引脚 P1.7 所接的工作指示灯亮。

04：判断接在 P2.4 引脚上的工作指示灯开关 K1 是否被按下。当 K1 被按下时 P2.4 位为 0，JNB 指令的作用是当 P2.4 位为 0 时转移到 START 处，即程序重新开始运行，电源指示灯亮。当 K1 没被按下时，P2.4 位为 1，程序执行下一行语句，使工作灯继续保持点亮。

05：使程序运行跳至 ON 处，继续保持工作指示灯亮。

06：程序结束。

4. 边用边学指令

本节程序新用到的指令有 JB、JNB。

JB 和 JNB 是位操作类指令，即以位 (bit) 为单位进行运算和操作。

- JB：该指令的作用是，若位为 1，则转移。例如在语句“JB P2.5, \$”里，P2.5 是 P2 寄存器中第 5 位，若 P2.5=1 时则转移，P2.5=0 时程序执行下一行指令。
- JNB：JNB 指令的作用与 JB 相反，当位为 0 时，则转移；若位为 1 时，程序执行下一行。

8.3.4 模拟仿真

本程序进行模拟仿真时，需要在原程序中增加模拟开关的语句，如在 01 与 02 行代码之间增加一行模拟工作指示灯按键开关 K2 被按下的语句“CLR P2.5”，此时，程序运行会通过 02 行语句，运行 03 行语句。

在 03 与 04 行之间增加一行模拟电源按键开关 K1 被按下的语句“ CLR P2.4 ”，此时，程序运行会转移到程序标号 START 处，从程序 01 行开始运行。

8.3.5 实例测试

将写入程序的单片机插入实验板插座上，检查无误后接通电源，此时 P1 端口的第一只灯即电源指示灯被点亮。

当按 K2 工作指示灯按键开关时，工作指示灯被点亮，电源指示灯熄灭。

当按 K1 电源指示灯按键开关时，电源指示灯被点亮，工作指示灯熄灭。

8.3.6 经验总结

本程序是独立式按键输入的简单应用程序。

独立式按键输入是指直接用输入/输出 (I/O) 线构成的单个按键电路。每个独立式按键单独占有一根 I/O 线，每根 I/O 线的工作状态不会影响其他 I/O 线的工作状态。

独立式按键接口电路配置灵活，软件结构简单，但每个按键必须占一根 I/O 线，在按键数较多时，I/O 口浪费太大，故只在按键数量不多时才采用这种按键电路。

在此电路中，按键输入为低电平有效，即查询到位为 0 时，说明有键按下。

8.4 键盘控制概述

键盘是由一组按键开关所组成。按键开关所组成的键盘可分为两种形式：独立式按键键盘和矩阵式按键键盘。

8.4.1 按键的特性

键盘是由一组按键开关所组成。通常，按键所用开关为机械弹性开关，当按下键帽时，按键内的复位弹簧被压缩，动片触点与静片触点相连，键盘的两个引脚被接通；松手后，复位弹簧将动片弹开，使动片与静片触点脱离接触，键盘的两个引脚被断开。

在理想状态下，按键引脚电平的变化如图 8.7 (a) 所示。但实际上，由于机械触点的弹性作用，一个按键开关从开始接上至接触稳定要经过数 ms 的抖动时间，抖动时间的长短与按键的机械特性有关，一般为 5 ~ 10ms，在这段时间里会连续产生多个脉冲；在断开时也不会一下子断开，按键抖动电压波形如图 8.7 (b) 所示。

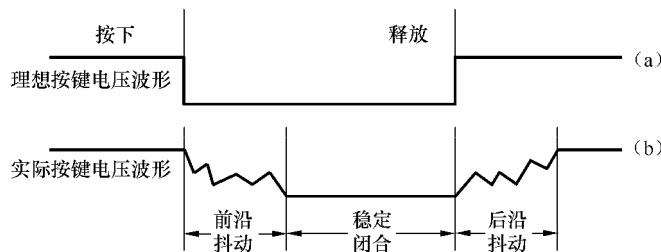


图 8.7 按键抖动电压波形

8.4.2 键盘输入中要解决的问题

键盘输入中主要解决两个问题：一是判断是否有键按下；二是消除按键抖动的影响，以免产生错误信号。

1. 按键的确认

按键的确认就是判断按键是否闭合，反映在电压上就是和按键相连的引脚呈现出高电平还是低电平。如果是高电平，则表示没有闭合；如果是低电平，则表示闭合。所以通过检测电平的高低状态，就能确认是否有键按下。

2. 按键抖动的消除

为了确保一次按键动作只确认一次按键，必须消除机械开关的抖动影响。消除按键的抖动，通常有硬件、软件两种消除方法。

一般在按键较多时，常采用软件的方法消除抖动。即在第一次检测到有按键被按下时，执行一段延时 12~15ms（因为机械按键由按下到稳定闭合的时间为 5~10ms）的子程序后，再确认该键电平是否仍保持为闭合状态电平，如果保持为闭合状态电平就可以确认真正有键按下，从而消除抖动的影响。

8.4.3 独立按键式键盘

独立按键就是各按键相互独立，每个按键各接一根输入线，如图 8.8 所示。

独立按键式键盘，一根输入线上的按键是否被按下，不会影响其他输入线上的工作状态。因此，通过检测输入线的电平状态就可以很容易判断哪个按键被按下了，如：

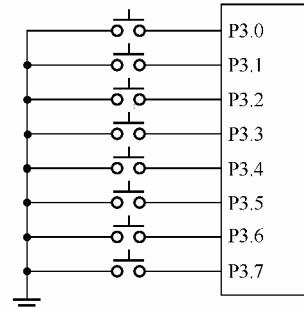


图 8.8 独立按键式键盘

JNB	P3.0 A0	; 如果 P3.0 键被按下，就跳转至 A0
JNB	P3.1 A1	; 如果 P3.1 键被按下，就跳转至 A1
JNB	P3.2 A2	; 如果 P3.2 键被按下，就跳转至 A2
JNB	P3.3 A3	; 如果 P3.3 键被按下，就跳转至 A3
JNB	P3.4 A4	; 如果 P3.4 键被按下，就跳转至 A4
JNB	P3.5 A5	; 如果 P3.5 键被按下，就跳转至 A5
JNB	P3.6 A6	; 如果 P3.6 键被按下，就跳转至 A6
JNB	P3.7 A7	; 如果 P3.7 键被按下，就跳转至 A7

独立按键电路配置灵活，软件结构简单。但每个按键需要占用一根输入口线，在按键数量较多时，输入口浪费较大，故此种键盘适用于按键较少的场合。

8.4.4 矩阵式按键键盘

矩阵式按键键盘适用于按键数量较多的场合，它由行线和列线组成，也称行列式键盘，按键位于行、列的交叉点上，其结构如图 8.9 所示，其中 0~3 为行线，4~7 为列线。一个

4×4 的行、列结构可以构成一个含有 16 个按键的键盘，与独立式按键键盘相比，要节省很多的 I/O 口。

其工作原理：按键设置在行、列线交点上，行、列线分别连接到按键开关的两端。行线通过上拉电阻接到 $+5V$ 上。当无键按下时，行线处于高电平状态，而当有键按下时，行线和列线将导通，因此，行线电平状态将由与此行线相连接的列线电平决定。假如此时列线电平设置为 0，由于行、列接通，故行线电平也为 0。

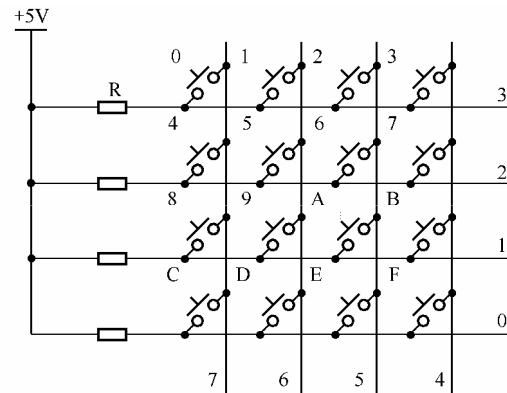


图 8.9 矩阵式按键键盘结构

8.5 用独立式键盘控制灯移动

功能说明：由 4 个按键开关组成独立式键盘，控制灯左移、右移和闪烁。

8.5.1 硬件设计

用独立式键盘控制灯移动的电路如图 8.10 所示。

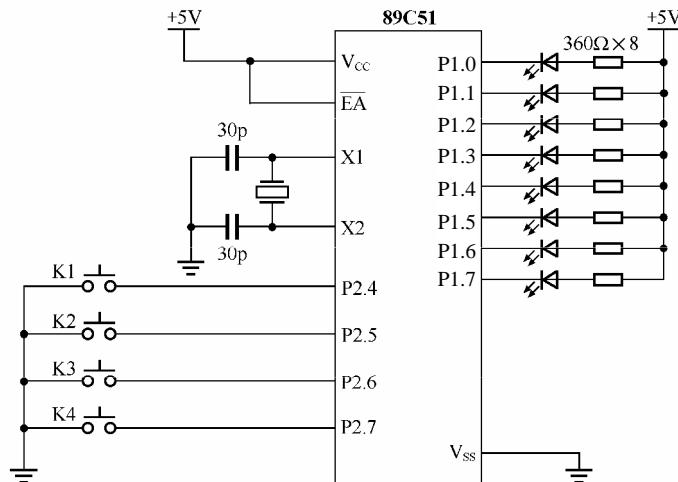


图 8.10 独立式键盘控制电路

4 个按键开关 K1、K2、K3 和 K4 分别与单片机 P2.4、P2.5、P2.6 和 P2.7 相连，组成独立式键盘接口输入电路。P1 端口接有 8 只发光二极管，用作输出演示。

8.5.2 程序设计

本程序是采用查询方式的独立式键盘处理程序。

1. 流程图

程序设计流程如图 8.11 所示。

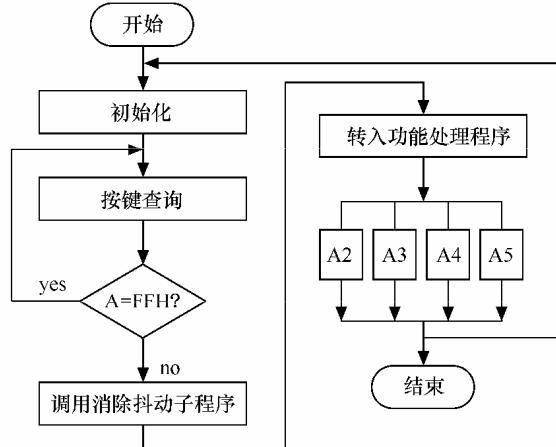


图 8.11 主程序流程

2. 程序

汇编语言编写的用独立式键盘控制灯移动源程序 SR04.ASM 代码如下：

01:	START:	MOV	P1, #0FFH	; 设置输出口初值
02:		MOV	A, #0FFH	; 设置输入方式
03:		MOV	P2, A	; 将 FF 送入 P2
04:	LOOP:	MOV	A, P2	; 读入键盘状态
05:		CJNE	A, #0FFH, LP0	; 是否有键按下
06:		JMP	LOOP	; 无键按下等待
07:	LP0:	ACALL	DELAY1	; 调用延时去抖动
08:		MOV	A, P2	; 重新读入键盘状态
09:		CJNE	A, #0FFH, LP1	; 非误读则转跳
10:		JMP	LOOP	; 按键循环查询
11:	LP1:	JNB	P2.4, A1	; K1 键按下时跳转到 A1
12:		JNB	P2.5, A2	; K2 键按下时跳转到 A2
13:		JNB	P2.6, A3	; K3 键按下时跳转到 A3
14:		JNB	P2.7, A4	; K4 键按下时跳转到 A4
15:		JMP	START	; 无键按下返回
16:	A1:	MOV	R0, #8	; 设置左移位数
17:		MOV	A, #0FEH	; 设置左移初值
18:	LOOP2:	MOV	P1, A	; 输出至 P1
19:		ACALL	DELAY	; 调用延时 1s 子程序
20:		RL	A	; 左移一位
21:		DJNZ	R0, LOOP2	; 判断移动位数
22:		JMP	START	; 返回主程序开始处

23:	A2:	MOV	R0, #8	; 设置右移位数
24:		MOV	A, #0FCH	; 设置右移初值
25:	LOOP3:	RR	A	; 右移一位
26:		MOV	P1, A	; 输出至 P1
27:		ACALL	DELAY	; 调用延时 1s 子程序
28:		DJNZ	R0, LOOP3	; 判断移动位数
29:		JMP	START	; 返回主程序开始处
30:	A3:	MOV	R0, #6	; 设置闪烁次数
31:		MOV	A, #0F0H	; 设置初值
32:	LOOP4:	MOV	P1, A	; 输出至 P1
33:		ACALL	DELAY	; 调用延时 1s 子程序
34:		CPL	A	; 反相
35:		MOV	P1, A	; 输出
36:		DJNZ	R0, LOOP4	; 判断移动位数
37:		JMP	START	; 返回主程序开始处
38:	A4:	MOV	R0, #10	; 设置闪烁次数
39:		MOV	A, #00H	; 设置初值
40:	LOOP5:	MOV	P1, A	; 输出
41:		ACALL	DELAY	; 调用延时子程序
42:		CPL	A	; 反相
43:		MOV	P1, A	; 输出
44:		DJNZ	R0, LOOP5	; 判断闪烁次数
45:		JMP	START	; 返回主程序开始处
46:	DELAY1:	MOV	R3, #60	; 消抖延时子程序
47:	D2:	MOV	R4, #248	
48:		DJNZ	R4, \$	
49:		DJNZ	R3, D2	
50:		RET		; 消抖延时子程序返回
51:	DELAY:	MOV	R5, #50	; 延时 1s 子程序
52:	DLY1:	MOV	R6, #100	
53:	DLY2:	MOV	R7, #100	
54:		DJNZ	R7, \$	
55:		DJNZ	R6, DLY2	
56:		DJNZ	R5, DLY1	
57:		RET		; 延时子程序返回
58:		END		; 程序结束

8.5.3 代码详解

A1 ~ A4 分别是 K1 ~ K4 按键的功能子程序，按键去抖动采用软件延时的方法，按键的接口选用 P2 端口中的 P2.4 ~ P2.7。

程序可分为 4 大部分：按键查询部分、键按下后跳转功能处理程序部分、功能键处理程序部分和延时子程序部分。

1. 按键查询部分

弄清按键查询部分是理解程序的关键，按键查询的工作过程如下。

(1) 判断是否有键按下。

第 02 行语句将数 FF 送入累加器 A 中。

第 03 行语句又将此数 FF 送入 P2，此时 P2 寄存器各位全是 1，即 P2 端口全是高电平。

第 04 行语句读入键盘状态，如果没有键被按下，此时 P2 寄存器各位仍然全是 1，送入累加器 A 中的也全是 1；如果有键被按下，从按键接口电路可看出，按下的键使与其相连的端口与地连接，其电平为 0。如 K1 键被按下，P2.4 为 0，此时读入的键盘值为 11101111。送入累加器 A 中的值也为 11101111。

第 05 行语句是判断语句，CJNE 指令用于比较两个操作数，若不相等就跳转。此时，因为 A 中的值为 0EFH，不等于操作数 OFFH，说明有键被按下。程序跳转到消抖动处理语句(标号 LP0) 处。如果没有键被按下，通过 06 语句转移到 LOOP 处，继续重新开始查询。

(2) 消除抖动影响。

本例中按键去抖动采用软件延时的方法。当判断出有键被按下后，通过 07 语句调用消抖延时子程序 DELAY1，消除抖动影响。

(3) 再重新判断是否有键按下

为了防止误读，08 ~ 10 语句作用是再重新判断是否有键按下，当确认无误后，程序将转移到按键功能处理部分。

2. 键按下后跳转功能处理程序部分

这部分通过 JNB 指令进行判断，当判断位是 0 时，则转移到功能处理程序；无键被按时，则返回到程序开始处重新查询。

3. 功能键处理程序部分

第 16 ~ 22 行语句为 1 号功能键 K1 处理程序，该程序使灯左移。

第 23 ~ 29 行语句为 2 号功能键 K2 处理程序，该程序使灯右移。

第 30 ~ 37 行语句为 3 号功能键 K3 处理程序，该程序使右边 4 只灯与左边 4 只交替闪烁。

第 38 ~ 45 行语句为 4 号功能键 K4 处理程序，该程序使灯闪烁。

4. 延时子程序部分

这部分有两个延时子程序，其中 DELAY1 是消除抖动延时子程序，DELAY 是灯移动间隔延时子程序。

8.5.4 模拟仿真

1. 模拟仿真前注意事项

(1) 在第 03 条语句与 04 条语句之间，增加一条模拟按键开关按下的语句，例如，模拟 1 号按键开关被按下的语句：

```
MOV P2,#11101111B
```

使程序能向下运行到 2 号功能键处理程序，以便观察。

(2) 在调用延时子程序前加上分号，即在第 07、19、27、33 和 41 语句前加“；”，使程序运行越过调用延时语句，目的是缩短模拟仿真时间。

2. 模拟仿真中注意观察的事项

(1) 当程序执行模拟开关语句“MOV P2,#11101111B”后，P2 的值为 EF，意味着接在 P2.4 口的 K1 开关键被按下。再经过重读键盘状态，并确认无误后，程序运行进入标号 LP1 处，执行 1 号键按下转移语句；接着程序运行转移到 A1 处，A1 是 1 号功能键处理程序，这时可以从 P1 值中观察到灯左移的情况。

(2) 可以改动模拟开关语句中的数值，如将其数值分别改为 11011111、10111111 或 01111111，模拟开关 K2、K3 或 K4 被按下的状态，观察程序运行路线，理解程序的功能原理。

8.5.5 实例测试

单片机插入实验板上后，按不同按键开关，可以看到 P1 端口的灯作不同方向的移动或闪烁，说明是在执行不同功能键的处理程序，例如。

按 K1 键，亮灯从右向左移动。

按 K2 键，亮灯从左向右移动。

按 K3 键，右边 4 只灯与左边 4 只灯交替闪亮。

按 K4 键，使灯闪烁 5 次。

8.5.6 经验总结

在使用开关不多的情况下，一般采用独立式键盘。设计时主要解决以下 3 个问题。

- 不断循环查询是否有键被按下。从电路中可知，当有键被按下时单片机端口接地，其位电平为 0。所以，可通过位操作类指令 JB 或 JNB 准确判断其位状态，也可以使用 CJNE 指令比较两个操作数，当检测到端口不是 FF 时证明整个端口中有某个键被按下。

两类指令相比较，位指令能准确判断出是哪个开关被按下；比较指令是先判断整个端口是否有键被按下，再通过位指令判断具体是哪个开关被按下。

- 消除按键时产生的抖动，防止误操作。当有键被按下时，通过延时程序停留一小段时间，就可以消除按键时产生的抖动。

- 当确定有按键操作后，使程序跳转到该键的功能子程序处运行。本节程序是通过位操作类指令 JNB 来确定被按键的具体开关位置，同时转移到该键的功能子程序处。功能子程序是按键所要达到的具体目的，即要完成的具体工作。

8.6 用矩阵式键盘控制显示器

功能说明：使用 4×4 矩阵式键盘，共有 16 个按键，通过扫描方法控制显示器输出 0~F

十六进制数。

8.6.1 硬件设计

用 4×4 矩阵式键盘控制显示器的电路如图 8.12 所示。

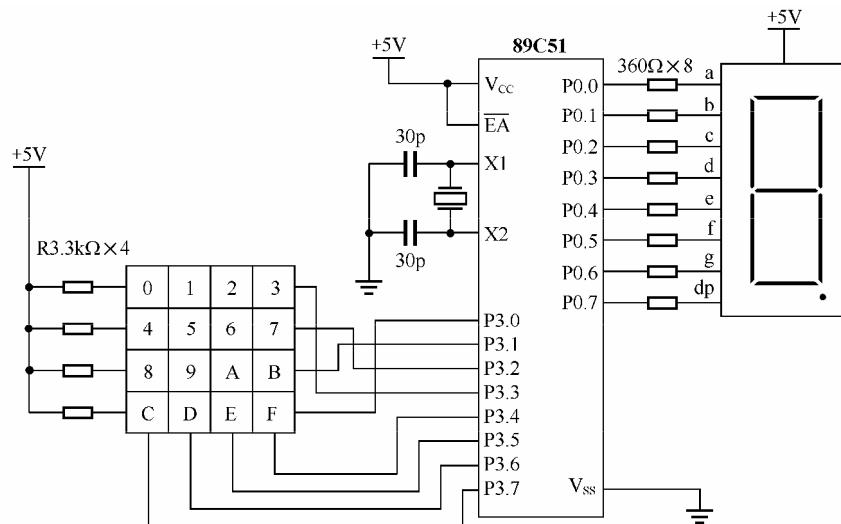


图 8.12 矩阵式键盘控制显示器电路

在单片机 P3 端口接有 4×4 矩阵式键盘，矩阵式键盘由 4 条行线和 4 条列线所组成，16 个按键设置在行、列线交点上，其中 4 条行线的一端分别与单片机 P3 端口的 P3.0、P3.1、P3.2 和 P3.3 相接，另一端通过上拉电阻接到+5V 上，平时使行线处于高电平状态；而 4 条列线的一端分别与 P3 端口的 P3.4、P3.5、P3.6 和 P3.7 相接。

当扫描开始时，首先将行设置低电平，在判断有键被按下后，读入列状态。如果列状态出现并非全部为 1 状态，这时 0 状态的列与行相交的键就是被按下的键。

单片机的 P0 端口为输出端，接有数码管显示器作演示用。

8.6.2 程序设计

1. 流程图

程序设计流程如图 8.13 所示。

2. 程序

汇编语言编写的用矩阵式键盘控制显示器源程序 SR05.ASM 代码如下：

```

01:          ORG      00H           ; 设置初值
02: START:    MOV      R4, #00H
03: L1:       MOV      R3, #0F7H      ; 扫描初值 (P3.3=0)
04:       MOV      R1, #00H      ; 取码指针
05: L2:       MOV      A, R3      ; 开始行扫描
06:       MOV      P3, A      ; 将扫描值输出至 P3

```

```

07:      MOV      A, P3          ; 读入 P3 值，判断是否有键被按下
08:      MOV      R4, A          ; 存入 R4，以判断是否放开
09:      SETB     C              ; C=1
10:      MOV      R5, #04H        ; 扫描 P3.4 ~ P3.7
11: L3:     RLC      A              ; 将按键值左移一位
12:      JNC      KEY            ; 有键按下 C=0，跳至 KEY
13:      INC      R1              ; C=1，没键按下，指针值加 1
14:      DJNZ    R5, L3          ; 4 列扫描是否完毕
15:      MOV      A, R3            ; 扫描值载入
16:      SETB     C              ; C=1
17:      RRC      A              ; 扫描下一行
18:      MOV      R3, A            ; 存回扫描寄存器
19:      JC      L2              ; C=1，程序跳转到 L2 处
20:      JMP      L1              ; C=0 则 4 行已扫描完毕
21: KEY:    ACALL   DELAY          ; 调用延时子程序
22: D1:     MOV      A, P3          ; 读入 P3 值
23:      XRL      A, R4          ; 与上次读入值作比较
24:      JZ      D1              ; A=0，表示按键未放
25:      MOV      A, R1            ; 按键已放开，指针载入 A
26:      ACALL   DISP             ; 调用显示子程序
27:      JMP      L1
28: DISP:   MOV      DPTR, #TABLE    ; 数据指针指到 TABLE
29:      MOVC    A, @A+ DPTR        ; 至 TABLE 取码
30:      MOV      P0, A            ; 输出
31:      RET
32: DELAY:  MOV      R7, #60          ; 消除抖动延时子程序
33:      MOV      R6, #248
34: DLY1:   DJNZ    R6, $              ; 循环
35:      DJNZ    R7, DLY1
36:      RET
37: TABLE:  DB      0C0H, 0F9H, 0A4H, 0B0H ; 编码表
38:      DB      99H, 92H, 82H, 0F8H
39:      DB      80H, 90H, 88H, 83H
40:      DB      0C6H, 0A1H, 86H, 8EH
41:      END
42:      ; 程序结束

```

8.6.3 代码详解

1. 新用到的寄存器

C 是标志寄存器 PSW 中的一个进位标志位 CY (C)。标志寄存器 PSW 也称程序状态字 PSW，是一个 8 位寄存器，用于存放程序运行的状态信息，其格式及各位的功能如图 8.14 所示。

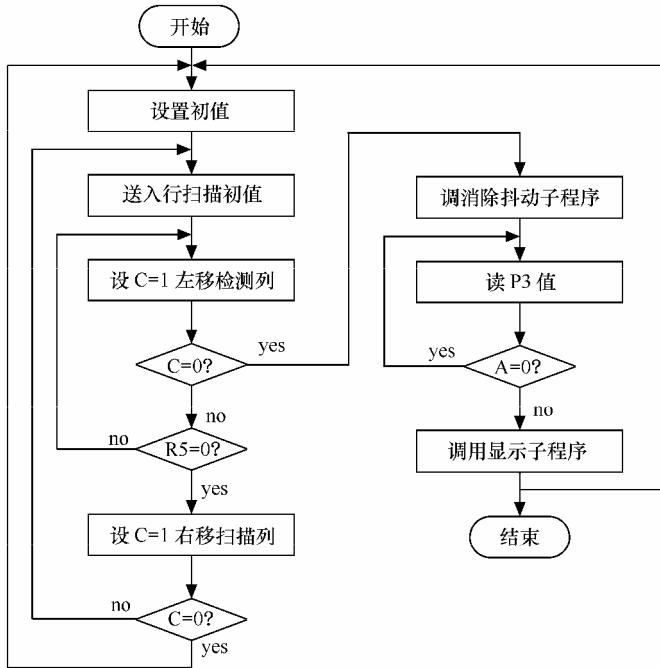


图 8.13 程序流程

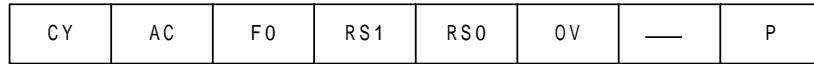


图 8.14 标志寄存器 PSW 示意图

- CY : PSW 中最常用的标志位 , 即进位标志位。其功能有两个 : 一是存放算术运算的进位标志 ; 二是在位操作中作累加位使用。
- F0 : 使用者标志 , 可自由使用。

2. 程序分析解释

此程序是使用 4×4 矩阵式键盘 , 利用扫描方法来控制显示器输出的程序。在理解该程序时 , 要重点理解扫描程序中的 5 个主要步骤。

(1) 向 P3 输入行扫描初值 (P3.3=0)

01 ~ 08 行语句主要完成向 P3 输入行扫描初值 , 开始行扫描。首先将行 P3.3 设置为低电平 , 在判断有键被按下后 , 同时读入列状态。

第 01 行语句设置程序从 00H 处开始。

第 02 行语句设置计数器 R4 初始值为 0 , 即 R4 从 0 开始计数。R4 是用来存入扫描值的寄存器。

第 03 行语句将扫描初值 #0F7H 送入 R3 寄存器。

第 04 行语句设置存放取码指针的寄存器 R1 初始值为 0。

第 05 行语句将 R3 中的值 #0F7H 送入累加器 A 中。

第 06 行语句再将累加器 A 中的值 #0F7H 送入 P3 , 即 P3.3 位为 0。从电路图可以看出 P3.3 ~ P3.0 为行线 , 也就是说开始扫描时先将与 P3.3 引脚相连的这条行线设置为 0。

第 07 语句读入 P3 值 ,如果有键被按下时 ,在读入的 P3 值里 P3.4 ~ P3.7 位中必然会有 0。如果没有键被按下 , 读入累加器 A 中的 P3 值仍为扫描初值 F7H。

第 08 行语句将读入的 P3 值存入 R4 , 为以后对按下的键判断是否放开作依据。

(2) 左移检测列。

09 ~ 14 行语句主要完成对列的检测。如果列状态出现并非全部为 1 状态 , 这时 0 状态的列与行交点的键就是所按下的键。

第 09 行语句使 C=1。C 是标志寄存器 PSW 中的一个进位标志位 CY (C)。

第 10 行语句将数字 4 (指 4 列) 存入寄存器 R5 中 , 此时 R5 是列扫描的计数器。

第 11 行语句将按键值左移一位 , 开始进行列扫描。指令 RLC 将累加器 A 连同进位标志位 C 左旋移动。累加器 A 有 8 位 , 加 1 进位标志位 C , 一共是 9 位 , 移位前 9 位为 1111101111 , 然后进行左旋移动。

A 值由 F7 (11110111) 变为 EF (11101111) , 由于连同进位标志位 C 一起左旋移动 , 实际是将累加器 A 中的最高位的 1 移至进位标志位 C 中 , C 中的 1 移至累加器 A 的最低位 , 所以进位标志位 C 移位后其值仍为 1 , 移位后 9 位为 : 111101111 , 此时开始进行列扫描。

第 12 行语句有键被按下时 C=0 , 跳转至 KEY。JNC 是位操作类指令。若 C=0 , 则转移到 KEY 处。因为 , A 中的高 4 位表示列 , 在没有键按下时 , 4 列都为 1 , 经 4 次列扫描 , 即执行 4 次左移 “ RLC A ” 后 , 进位标志位 C 仍为 1。当有键被按下时 , 4 列中必然有一列为 0 , 在列扫描中会使 0 移至 C 标志位。所以 , 当检测到 C=0 时 , 表明有键被按下。

第 13 行语句使 C=1 , 表明没键被按下 , 取码指针值加 1。语句 “ INC R1 ” 中的 R1 为取码指针计数器 ; INC 指令是加 1 指令 , 使计数器 R1 加 1。

第 14 行语句判断 4 列是否扫描完毕。

(3) 右移扫描行。

15 ~ 20 行语句完成右移扫描行。

第 15 行语句将扫描初值 F7 输入累加器 A。

第 16 行语句使 C=1。

第 17 行语句向右移一位 , 扫描下一行。语句 “ RRC A ” 中的 RRC 指令的作用是带进位循环右移。每执行一次该指令 , 向右移动一位。当 0 移至进位标志位 C , 即 C=0 时 , 说明 4 行扫描完毕。

第 18 行语句将扫描值存入寄存器 R3。

第 19 行语句判断进位标志位 C , JC 的指令作用是当 C 为 1 时进行行移。即 C=1 , 说明行扫描没有完成 , 跳转到 L2 处继续进行行扫描。每完成一次行扫描都要进行 4 次列扫描。

第 20 行语句 : C=0 时 , 4 行扫描已完毕 , 开始重新扫描。

(4) 消除按键抖动

第 21 行语句通过调用延时子程序 , 消除按键时产生的抖动。

(5) 判断按键释放后取码

22 ~ 26 行判断按键释放后 , 调用显示子程序。

第 22 行语句读入 P3 值 , 即将 P3 值存入累加器 A。

第 23 行语句将累加器 A 中的值与寄存器 R4 的值相比较 , 如果累加器 A 中的值与寄存器 R4 中的值相等 , 进行异或运算 , A 的值为 0。

第 24 行语句中 JZ 指令的作用是判断累加器 A 的值。如果 A 为 0 则跳转。当 A 为 0 时，表示按键未放开，跳转到 D1 处继续比较，等待按键放开；当 A 不等于 0 时，说明按键已放开，程序向下运行。

第 25 行语句将取码指针 R1 的值存入 A。

第 26 行语句调用显示子程序，取码显示。

27：重复扫描。

28~31：显示子程序。

32~36：消除抖动延时子程序。

37~41：编码表。

42：程序结束。

3. 边用边学指令

本节程序新用到的指令有：RLC、RRC、JNC、JC 和 XRL。

- RLC 和 RRC：RLC 和 RRC 与前边用过的 RL 和 RR 都是位移类指令，基本功能是进行移位。但每个指令的具体功能各不相同，现归纳说明如下。

80C51 的位移指令只能对累加器 A 进行移位，一条指令执行一次只能移动 1 位。移位有循环左移（RL）、循环右移（RR）、带进位循环左移（RLC）和带进位循环右移（RRC）4 种。

（1）循环左移 RL。

循环左移如图 8.15 所示。

指令的意义是将累加器 A 中的 8 位二进制数向左移动 1 位，最高位移至最低位。

例如 $A=abcdefg$ （a、b、c、d、e、f、g、h 均为二进制 1 或 0），循环左移指令执行后， $A=bcdefgh$ 。

（2）循环右移 RR。

循环右移如图 8.16 所示。

RL A;



图 8.15 循环左移 RL 示意图

RR A;

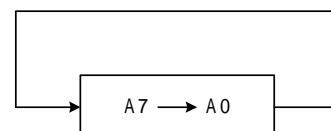


图 8.16 循环右移 RR 示意图

指令的意义是将累加器 A 中的 8 位二进制数向右移动 1 位，最低位移至最高位。

例如 $A=abcdefg$ （a、b、c、d、e、f、g、h 均为二进制 1 或 0），循环右移指令执行后， $A=habcdgfg$ 。

（3）带进位循环左移 RLC。

带进位循环左移如图 8.17 所示。

指令的意义是将累加器 A 中的 8 位二进制数和进位位 CY 一同向左移动 1 位，累加器 A 中的最高位移至 CY，CY 移至累加器 A 的最低位。

例如 $A=abcdefg$ 、 $CY=i$ （a、b、c、d、e、f、g、h 均为二进制 1 或 0）。

带进位循环左移指令执行后， $A=bcdefghi$ 、 $CY=a$ 。

(4) 带进位循环右移 RRC。

带进位循环右移如图 8.18 所示。

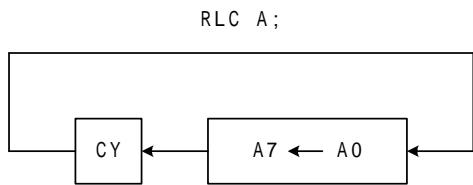


图 8.17 带进位循环左移 RLC 示意图

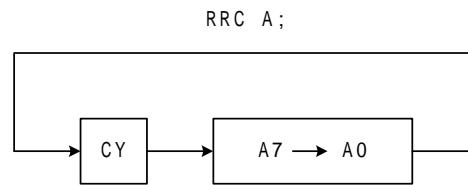


图 8.18 带进位循环右移 RRC 示意图

指令的意义是将累加器 A 中的 8 位二进制数和进位位 CY 一同向右移动 1 位，累加器 A 中的最低位移至 CY，CY 移至累加器 A 的最高位。

例如 $A=abcdefg$ 、 $CY=i$ (a, b, c, d, e, f, g, h 均为二进制 1 或 0>)。

带进位循环右移指令执行后， $A=abcdeg$ 、 $CY=h$

- JNC：位操作类指令。功能是判断 C 标志位，若 $C=0$ ，则跳转。
- JC：位操作类指令。功能是判断 C 标志位，若 $C=1$ ，则跳转。
- XRL：逻辑运算类指令中的累加器异或寄存器运算。

例如： $A=10011111$ ， $R4=10111111$ ，执行指令：

```
XRL A, R4
```

运算过程为：

$$\begin{array}{r} 10011111 \\) 10111111 \\ \hline 00100000 \end{array}$$

结果为 $A=00100000=0H$ 。

4. 至本章用过的指令归类

- 数据传送类指令：MOV、MOVC、PUSH、POP。
- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、RLC、RRC、CLR、ANL、ORL、XRL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI、JZ。
- 位操作类指令：CPL、CLR、SETB、JB、JNB、JBC、JNC、JC。
- 伪指令：END、DB、ORG。

8.6.4 模拟仿真

1. 模拟仿真前注意事项

(1) 在第 05 条语句与 06 条语句之间，增加一条模拟按键按下的语句，例如模拟按键“3”按下的语句：

```
MOV A,#11100111B
```

(2) 在第 22 条语句与 23 条语句之间，增加一条模拟按键放开的语句：

```
MOV A,#11110111B
```

(3) 将第 21 条语句“KEY: ACALL DELAY”中的“KEY:”后边加上分号，使程序运行跳过调用延时语句，目的是缩短模拟仿真时间。

2. 模拟仿真中注意观察

(1) 注意观察 P0 输出值。当程序运行到第 30 行语句时，P0 输出值为 B0。对应编码表可知，此时在显示器上应显示 3。

(2) 可以改动模拟按键的数值，观察 P0 输出值，再对照编码表，可以看出按键的数与显示的数是一一对应的。同时，注意观察程序的运行路线，理解程序的功能原理。

(3) 去掉模拟按键语句，程序运行在 03 ~ 27 行语句之间不断地反复扫描。由于没有按键按下，P0 没有输出。

8.6.5 实例测试

若单片机实验板上没有 4×4 矩阵式键盘，可参考第 26 章 26.3.5 节在多空板上制作一个 4×4 矩阵式键盘。

该程序中的键盘上字符排序与编码表 TABLE 中代码的排序是相同的，例如，按下第一排左数第一个键，数码管显示器将显示 0；按下第一排左数第二个键，数码管显示 1。按键盘中不同键号，显示器将会显示出相应的十六进制数。

8.6.6 经验总结

在使用按键数量较多的情况下，可以将按键开关按矩阵式排列组成矩阵式键盘，这将比独立式按键节省很多的 I/O 口。例如，使用 16 个按键开关，如果采用独立式键盘则需要 16 个 I/O 口；如果采用矩阵式键盘，只需要 8 个 I/O 口，节省了硬件资源。

使用矩阵式键盘时，通过不断对键盘进行扫描的办法来确定是否有键被按下。

第 9 章 中断



单片机中断功能可以提高 CPU 的效率；可以实现实时处理，以满足实时控制要求；可以及时处理故障，提高单片机的可靠性。所以，单片机中断系统功能的强弱是衡量其性能好坏的重要标志之一。

9.1 中断控制功能的作用

9.1.1 什么是中断

所谓中断，就是打断正在进行的工作，转而去做另外一件事情。

比如说，会计正在记账，桌面上摆着摊开的账本和正在使用的计算器，这时候有人敲门叫他去办另一件事情，他暂时放下手头工作，并保存好账本和计数器以免被弄乱或丢失（在中断中称“保护现场”）；等处理完事情后返回办公桌前再拿出帐本和计数器（在中断中称“恢复现场”），继续记账。这一过程就是中断以及中断处理的过程。

单片机的中断过程与上述过程类似，如图 9.1 所示。

9.1.2 实现中断的好处

1. 提高了 CPU 的效率

CPU 是计算机系统的指挥中心，它与外围设备（如按键、显示器等）联系沟通的方法有轮询和中断两种。

轮询的方法是指无论外围设备 I/O 是否需要服务，CPU 每隔一段时间都要依次询问一遍，此种方法 CPU 需要花费一些时间来做询问服务工作。

而中断方法是指当外围设备需要服务时，外设会主动告诉 CPU，CPU 得知有外围设备需要服务时，才去执行中断处理子程序，所以省去了依次查询的时间，提高了 CPU 的利用率。

2. 可实现实时处理

在实时控制中，利用中断技术，外围设备在任何时刻都可以发出中断请求，CPU 接到请

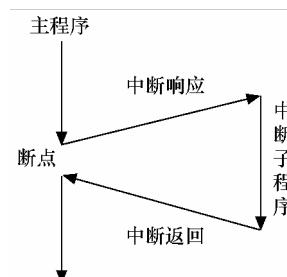


图 9.1 中断过程示意图

求后就能及时处理，以满足实时控制要求。

3. 可及时处理故障

计算机系统在运行过程中难免会出现一些事先无法预料的故障，如电源掉电、存储器出错、外围设备工作不正常等，这时可以通过中断系统由故障源向 CPU 发送中断请求，由 CPU 及时转到相应的故障处理程序进行处理，提高了计算机的可靠性。

9.1.3 中断处理过程

中断处理过程如图 9.2 所示。

1. 中断当前程序并保护断点

CPU 响应中断后，把被中断程序的断点，即 PC 值，压入堆栈保存，以备中断处理完毕后能返回被中断的程序。

2. 转入中断服务入口

给出中断入口地址，转入被响应的中断处理程序。

3. 保护现场

为了使被中断的程序及其所使用的寄存器内容不被破坏，以免在中断返回后影响被中断程序的执行，要将被中断程序的有关信息及其所使用的寄存器内容保护起来，压入堆栈保存，这就是保护现场。

4. 执行中断服务程序

中断服务程序是中断处理的具体内容。

5. 恢复现场

把保护现场所保存的中断程序的有关信息及其所使用的寄存器内容恢复原样，以便返回被中断的程序后能够正常执行。

6. 中断返回

中断返回是把程序运行从中断处理程序转回到被中断的程序中去。中断返回是通过中断返回指令 RETI 完成的。

9.1.4 中断源及入口地址

1. 中断源

80C51 是一个多中断源的单片机，共有 3 类 5 个中断源，分别是两个外部中断、两个定时中断和一个串行中断，如表 9.1 所列。

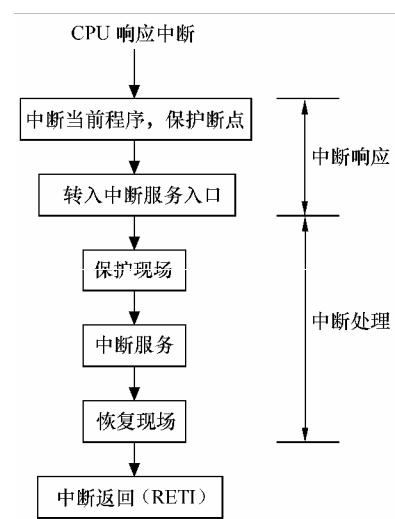


图 9.2 中断处理流程图

表 9.1

中断源

中 斷 名 称	中 断 源	中 斷 名 称	中 断 源
外部中断 0	外部事件由 P3.2 提供	定时器 1 溢出中断	由片内定时器/计数器 1 提供
外部中断 1	外部事件由 P3.3 提供	串行口中断	由片内串行口提供
定时器 0 溢出中断	由片内定时器/计数器 0 提供		

在 3 类中断源中，外部中断类是指由外部原因引起的中断，共有两个中断源，即外部中断 0 (INT0) 和外部中断 1 (INT1)。它们的中断请求信号分别由引脚 INT0 (P3.2) 和 INT1 (P3.3) 引入。

外部中断请求有两种信号方式：电平方式和脉冲方式。

电平方式是低电平有效。只要单片机在中断请求引入端 (INT0 和 INT1) 上采样到有效的低电平时，就激活外部中断。

脉冲方式是脉冲的下降沿有效。如在中断请求引入端采样到前一次为高，后一次为低，即为有效中断请求。上述两种信号方式可通过有关控制位进行设置。

定时中断类和串行中断类将在有关章节中介绍。

2. 中断源入口地址

每一个中断源都有一个固定的中断处理程序入口，它们的地址如表 9.2 所示。

表 9.2

中断源的入口地址

中 断 源	入 口 地 址	中 断 源	入 口 地 址
外部中断 0	0003H	定时器/计数器 1	001BH
定时器/计数器 0	000BH	串行口中断	0023H
外部中断 1	0013H		

从表 9.2 可见，各中断源入口地址间只相隔 8 个单元。一些简单的中断处理程序可以直接安排在这些单元之内；复杂的中断处理程序，则可在各中断入口地址处设置一条无条件转移指令，跳转到其他地址安排中断处理程序。

9.2 中断的控制及设置

中断系统结构如图 9.3 所示，从图可见，中断控制的实质是对 4 个特殊功能寄存器 TCON、SCON、IE 和 IP 进行管理和控制。只要按照人们的要求对这些寄存器的相应位进行设置（存入 0 或 1，一般设置 1 为开通），就能完成对中断的有效管理和控制。

9.2.1 中断允许控制寄存器 IE

1. IE 的格式及位的含义

寄存器 IE 的地址为 A8H，作用是控制各中断源的开放与关闭。它实行两级控制，IE 中有

个总的控制位 EA , 当 EA=0 时 , 则屏蔽所有的中断请求 ; 而当 EA=1 时 , CPU 开放总中断。

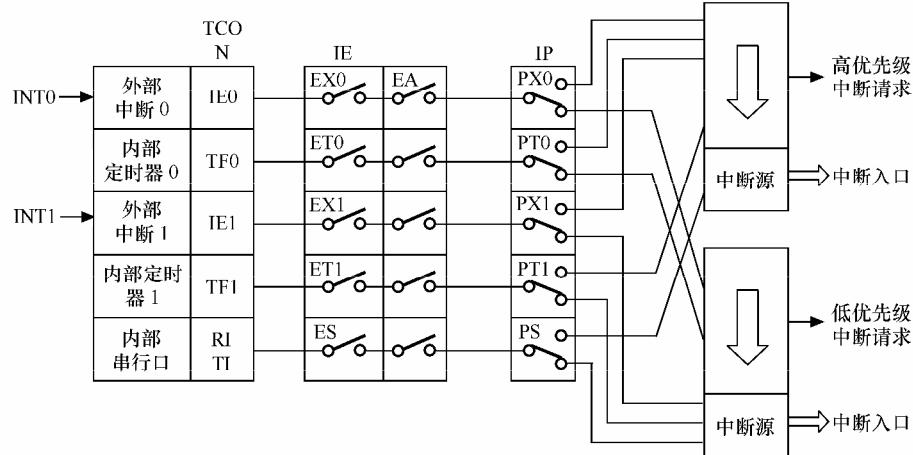


图 9.3 中断系统结构示意图

各个中断源的请求是否被开放 , 还要看 IE 中各个中断源自己的中断允许控制位的状态。IE 的格式及各位的含义如图 9.4 所示。

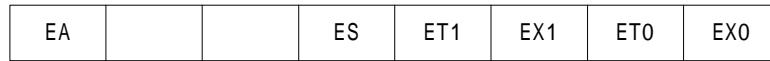


图 9.4 中断允许控制寄存器 IE 示意图

- EA : 中断允许总控制位。EA=0 时 , 中断总禁止 , 关闭所有中断 ; EA=1 , 中断总允许。中断总允许后各中断的禁止或允许则由中断源的中断允许控制位进行设置。它们之间的关系类似于电表控制盘上的总开关及各户的分开关一样。
- EX0 : 外部中断 0 (INT0) 允许控制位。EX0=0 时 , 禁止外部中断 0 中断 ; EX0=1 时 , 允许外部中断 0 中断。
- EX1 : 外部中断 1 (INT1) 允许控制位 , 其功能与 EX0 类同。
- ET0 : 定时器 / 计数器 T0 的溢出中断允许控制位。ET0=0 时 , 禁止定时器 / 计数器 T0 中断 ; ET0=1 时 , 允许定时器 / 计数器 T0 中断。
- ET1 : 定时器 / 计数器 T1 的溢出中断允许控制位 , 其功能与 ET0 类同。
- ES : 串行中断允许控制位。ES=0 时 , 禁止串行中断 ; ES=1 时 , 允许串行中断。

2. IE 的设置举例

假如程序需要使用外部中断 0 (INT0) , 只要将寄存器 IE 的外部中断允许控制位和中断允许总控制位设置为 1 即可 , 可通过下面的语句来实现 :

```
MOV IE, #10000001B ; INT0 中断开通
```

9.2.2 中断优先级控制寄存器 IP

1. IP 的格式及位的含义

寄存器 IP 的地址为 B8H , 作用是控制中断的优先级 , 其格式及各位的含义如图 9.5 所示。

			PS	PT1	PX1	PT0	PX0
--	--	--	----	-----	-----	-----	-----

图 9.5 中断优先级控制寄存器 IP 示意图

- PX0：外部中断 0 优先级设置位。
- PX1：外部中断 1 优先级设置位。
- PT0：定时器/计数器中断 0 优先级设置位。
- PT1：定时器/计数器中断 1 优先级设置位。
- PS：串行中断优先级设置位。

中断优先级只分高与低两个级别，各位为 0 时，为低优先级；各位为 1 时，为高优先级。

2. IP 的设置举例

假如程序需要使外部中断 0 为高优先级，只要将 IP 的第一位设置为 1 即可，可通过下面的语句来实现：

```
MOV IP, #00000001B ; INT0 中断优先
```

3. 中断优先级控制原则

- (1) 低优先级中断请求不能打断高优先级的中断处理；反之，则可以。
- (2) 如果一个中断请求已被响应，则同级的其他中断响应被禁止。
- (3) 如果同级的多个中断请求同时出现，则按次序从高到低依次为：外部中断 0、定时/计数中断 0、外部中断 1、定时/计数中断 1、串行中断。

9.2.3 定时器控制寄存器 TCON

1. TCON 的格式及位的含义

寄存器 TCON 的地址为 88H。TCON 既有定时器/计数器的控制功能，又有中断控制功能。其中，与中断有关的控制位共 6 位，其格式及位的含义如图 9.6 所示。

TF1		TF0		IE1	IT1	IE0	IT0
-----	--	-----	--	-----	-----	-----	-----

图 9.6 定时器控制寄存器 TCON 示意图

- IT0：选择外部中断 INT0 的中断触发方式，IT0=0 时采用电平触发方式；IT0=1 时采用脉冲触发方式。
- IT1：选择外部中断 INT1 的中断触发方式，其功能与 IT0 类同。
- IE0：外部中断 INT0 的中断请求标志位。当检测到 INT0 引脚有中断请求信号时，此位由硬件置 1；在中断响应完成后转向中断处理子程序时，再由硬件自动清 0。
- IE1：外部中断 INT1 的中断请求标志位，其功能与 IE0 类似。
- TF0：片内定时器/计数器 0 溢出中断请求标志位。当定时器/计数器 0 溢出发生时，此位由硬件置 1；在中断响应完成后转向中断处理子程序时，再由硬件自动清 0。
- TF1：片内定时器/计数器 1 溢出中断请求标志位，其功能与 TF0 类同。

2 . TCON 的设置举例

从上面寄存器各位的含意可以看出，在对寄存器 TCON 的设置上只要注意设置触发方式即可。

例如，程序需要使用外部中断 INT0，并采用电平触发方式。此时只要将 TCON 的第一位设置为 0 即可，通过下面语句来实现：

```
MOV TCON, #0000000B ; 设置 INT0 为电平触发
```

9.2.4 串行口控制寄存器 SCON

寄存器 SCON 的地址为 98H，其中与中断有关的控制位共两位，其格式及位的含义如图 9.7 所示。



图 9.7 串行口控制寄存器 SCON

- TI：串行口发送中断请求标志位。当发送完一帧串行数据后，由硬件置 1；在转向中断处理程序后，用软件清 0。

- RI：串行口接收中断请求标志位。当接收完一帧串行数据后，由硬件置 1；在转向中断处理程序后，用软件清 0。

9.3 用外部中断控制灯闪烁

功能说明：单片机 P1 端口的 8 只 LED 作左移右移，不断循环。当按外部中断 INT0 开关 K1 时，循环停止，转而使 8 只 LED 闪烁 4 次，然后再恢复灯的左右移循环。

9.3.1 硬件设计

外部中断 INT0 电路如图 9.8 所示。

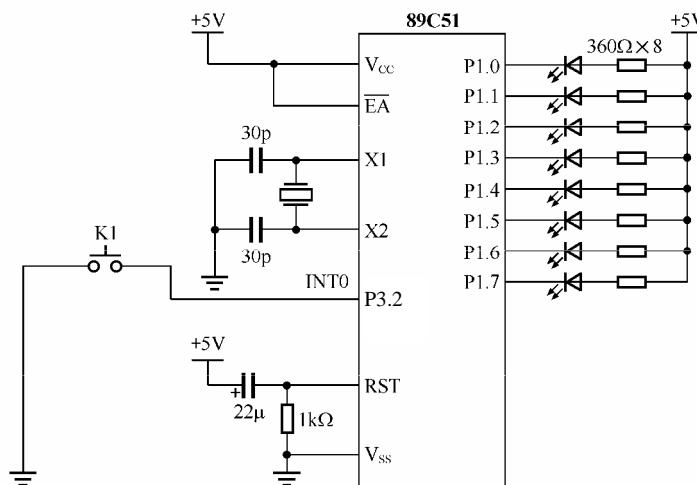


图 9.8 外部中断 INT0 电路

单片机的 P3 端口除当一般 I/O 使用外，还有第二功能，其 P3.2 (INT0) 脚是外部中断 0 的输入脚，其 P3.3 (INT1) 脚是外部中断 1 的输入脚。

本电路图中，在 P3 端口的 P3.2 引脚上接有外部中断 INT0 控制开关 K1，作为外部中断的输入信号。输出部分由 P1 端口接的 8 只 LED 作输出显示。

9.3.2 程序设计

本程序在前边第 6 章第 6.7 节灯左右移程序的基础上，增加了外部中断 INT0。

1. 流程图

程序设计流程如图 9.9 所示。

2. 程序

汇编语言编写的用外部中断控制灯闪烁源程序 ZD01.ASM 代码如下：

```

01 :      ORG      00H           ;起始地址
02 :      JMP      START        ;跳到主程序 START
03 :      ORG      03H           ;INT0 中断起始地址
04 :      JMP      EXT0        ;跳到中断子程序
05 : START : MOV     IE, #10000001B ;INT0 中断开通
06 :      MOV     IP, #00000001B ;INT0 中断优先
07 :      MOV     TCON, #00000000B ;INT0 为电平触发
08 :      MOV     SP, #70H         ;设定堆栈指针
09 : LOOP :   MOV    R0, #8          ;设置左移位数
10 :      MOV    A, #0FEH        ;设置左移初值
11 : LOOP1 :  MOV    P1, A          ;输出至 P1
12 :      ACALL  DELAY        ;调用延时子程序
13 :      RL     A             ;左移一位
14 :      DJNZ   R0, LOOP1      ;判断移动位数
15 :      MOV    R0, #8          ;设置右移位数
16 : LOOP2 :  RR    A             ;右移一位
17 :      MOV    P1, A          ;输出至 P1
18 :      LCALL  DELAY        ;调用延时子程序
19 :      DJNZ   R0, LOOP2      ;判断移动位数
20 :      JMP    LOOP          ;重设显示值
21 : EXT0 :  PUSH   ACC          ;将 A 值压入堆栈
22 :      PUSH   PSW          ;将 PSW 值压入堆栈
23 :      SETB  RS0          ;设定工作寄存器组 1
24 :      CLR    RS1          ;清除工作寄存器组 1
25 :      MOV    R0, #4          ;设置闪烁次数

```

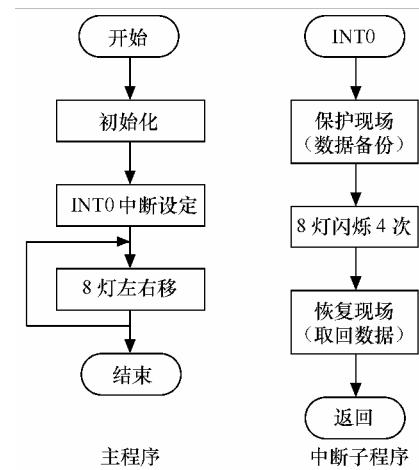


图 9.9 程序流程图

```

26 : LOOP3 : MOV      A, #00H          ;A 存有 P1 口欲显值
27 :      MOV      P1, A          ;P1 端口灯亮
28 :      LCALL    DELAY        ;调用延时 1s 子程序
29 :      CPL      A           ;将 A 的值反相
30 :      DJNZ    R0, LOOP3     ;判断闪烁次数
31 :      POP      PSW         ;从堆栈取回 PSW 值
32 :      POP      ACC         ;从堆栈取回值 A 值
33 :      RETI                 ;返回主程序
34 : DELAY : MOV      R5, #50        ;延时 1s 子程序
35 : DLY1 : MOV      R6, #100
36 : DLY2 : MOV      R7, #100
37 :      DJNZ    R7, $          ;延时子程序
38 :      DJNZ    R6, DLY2
39 :      DJNZ    R5, DLY1
40 :      RET
41 :      END                 ;程序结束

```

9.3.3 代码详解

1. 标号说明

START：起始程序的进入点。
 LOOP：左右移循环执行的进入点。
 LOOP1：左移循环的进入点。
 LOOP2：右移循环的进入点。
 LOOP3：闪烁程序循环的进入点。
 EXT0：中断子程序的进入点。
 DELAY：延时子程序的进入点

2. 寄存器使用分配情况

寄存器 P1 和 A (ACC) 的作用与前相同。R0 作计数器，在左移、右移和闪烁程序中都由 R0 负责计数，R5、R6 和 R7 是延时子程序中的计数器。

PSW、SP、IE、IP、TCON 是新用到的特殊功能寄存器，其中 PSW 是一个 8 位的专用寄存器，称为程序状态字寄存器，用于存储程序运行中的各种状态信息。其中有两位名为 RS0 和 RS1，改变其值就能选择 CPU 当前工作的寄存器组。

寄存器组是 CPU 工作时临时存储数据的地方，共有 0 ~ 3 组，每组 8 个单元 R0 ~ R7，RS0、RS1 与寄存器组的对应关系见表 9.3。

表 9.3 R S0、R S1 与寄存器组的对应关系

RS1	RS0	寄存器组	片内 RAM 地址
0	0	第 0 组	00H ~ 07H
0	1	第 1 组	08H ~ 0FH

续表

RS1	RS0	寄存器组	片内 RAM 地址
1	0	第 2 组	10H ~ 17H
1	1	第 3 组	18H ~ 1FH

单片机开始工作时，RS0=0、RS1=0，CPU 选用第 0 组的 8 个单元为当前工作寄存器。当主程序要调用中断子程序时，通过“CLR RS1”语句使 RS1=0；通过“SETB RS0”语句使 RS0=1，即将 RS1 置为 0、RS0 置为 1，则中断子程序就可以使用第 1 组 8 个单元为当前工作寄存器 R0 ~ R7。

这样，使主程序使用的第 0 组 R0 ~ R7 的内容就能保持不变，即主程序与子程序各自使用各自的寄存器，以免内容混淆。

SP 也是一个 8 位的专用寄存器，称为堆栈指针寄存器，用于暂存堆栈顶部的地址。堆栈是按先进后出、后进先出的原则存取数据的一个专用存储区。数据的进栈出栈由指针 SP 统一管理。

TCON、SCON、IE 和 IP 是 4 个特殊功能寄存器，它的作用是对中断进行管理和控制。

3. 程序分析解释

01 ~ 04：规定起始地址。通过“ORG 00H”语句，规定了标号 START 所在的地址为 00H，即第一条指令从 00H 开始存放；通过“ORG 03H”语句规定了下面标号 EXT0 的中断子程序起始地址为 03H，03H 是外部中断 0 的入口地址。

05 ~ 08：中断设置。首先是对中断允许控制寄存器 IE 的设置。允许中断是由两层控制，第一层为全面控制（EA），它是总开关，第二层（EX0）是对个别中断的控制，它是分开关。

通过“MOV IE, #10000001B”语句，使中断允许控制寄存器 IE 中的允许总控制位 EA 值为 1，即中断总允许。使 EX0=1，即外中断 0 允许使用。

接着对优先级控制寄存器 IP 进行设置。中断有优先级，各中断源的优先级由优先级控制寄存器 IP 进行管理，分低优先级和高优先级：PX0=0 时，为低优先级；PX0=1 时，为高优先级。“MOV IP, #00000001B”语句使 PX0=1，即使外部中断 0 为高优先级。

中断触发有两种方式：脉冲方式和电平触发方式。两种方式的选择是通过定时寄存器 TCON 来设定的。通过“MOV TCON, #00000000B”语句将 IT0 位设置为 0，则使外中断 0 选择了电平触发方式。

语句“MOV SP, #70H”设置堆栈指针地址，当程序中执行保护现场指令 PUSH 或恢复现场指令 POP 时，SP 会自动加 1 或减 1，然后将数据压入或弹出堆栈，目的是使中断结束后能正确地回到程序调用点继续执行。

09 ~ 20：使 LED 左移和右移。

21 ~ 33：中断子程序。

中断子程序由 3 部分组成：保护现场部分（21 ~ 24 行语句）、闪烁功能部分（25 ~ 30 行语句）和恢复现场部分（31 ~ 32 行语句）。

其中闪烁功能部分是使用中断的目的。而闪烁功能部分前边的保护现场和后边的恢复现场部分是为了使中断结束后能正确地回到原程序调用点继续执行。

21 ~ 22 行语句是将累加器 A 和程序状态字寄存器 PSW 中的值压栈保存；23 ~ 24 行语句

是设置 RS1=0、RS0=1，目的是使中断子程序使用第 1 组工作寄存器。这样，第 0 组 R0 ~ R7 的内容就能保持不变

34 ~ 40：延时 1s 子程序。

41：程序结束。

4. 边用边学指令

本节程序新用到的指令有：PUSH、POP、RETI 和 ORG。

- PUSH 和 POP 是两条专用堆栈操作指令，属于数据传送类。堆栈操作的特点是按先进后出、后进先出原则存取数据。PUSH 和 POP 是成对出现的，PUSH 是进栈指令，POP 是出栈指令，一般用于在中断子程序中保护现场和恢复现场。

- RETI 是控制转移类指令，RETI 的功能是使中断服务程序结束返回。RETI 除具有子程序返回指令 RET 所具有的全部功能之外，还有清除中断响应时被置位的优先级状态、开放较低级的中断和恢复中断逻辑等功能。

- ORG 是伪指令，ORG 的功能是规定下面目标程序存放的起始地址。

9.3.4 模拟仿真

1. 模拟仿真前注意事项

(1) 将第 12、18、28 行调用延时语句前加上分号，如“ACALL DELAY”改为“;ACALL DELAY”。目的是使程序跳过调用延时的语句，节省模拟运行的时间。

(2) 将第 20 行语句“JMP LOOP”改为“JMP EXT0”，使程序能运行标号为 EXT0 的中断子程序子程序。

(3) 重新将程序“编译/汇编”和“产生代码并装入”，然后再调试。

2. 模拟仿真中注意事项

(1) 注意观察中断设定中特殊功能寄存器 IE、IP、TCON 和 SP 值的变化。其中 IE、IP、TCON 在程序中给出的是二进制值，但在模拟仿真特殊功能寄存器窗口中看到的是十六进制数。

(2) 在运行 20 行语句前，ACC 的值是 FE、PSW 的值是 01H，即 PSW 中控制工作寄存器组的两个位 RS0=0、RS1=0，所以此时 CPU 选中的是第 0 组工作寄存器。

当运行 21 ~ 24 行语句之后，PSW 的值改变为 09H，即 PSW 中控制工作寄存器组的两个位 RS0=1、RS1=0，所以此时 CPU 使用的工作寄存器由第 0 组改为第 1 组。使原来第 0 组工作寄存器的内容能保持不变，起到保护现场的作用。

(3) 当程序运行到 31 ~ 32 行之后，PSW 的值为 01、ACC 的值为 FE，这是通过 POP 指令取回了 PSW 和 A 在进入中断子程序前的值（即原来值），起到恢复现场的作用。

9.3.5 实例测试

将写入程序的单片机插入实验板插座上，检查无误后接通电源，此时将看到亮灯从右向左移动，移到最左端后再从左向右移动，不断循环。

当按开关 K1 产生中断信号时，灯移动循环停止，转而执行中断服务程序，8 只 LED 闪

烁 4 次。中断服务程序结束之后，亮灯再恢复左右移动循环。

在实例测试时要仔细观察，程序从什么地方发生中断，将从什么地方开始恢复。

亮灯从右端向左移动到第 3 个时，下一个将是第 4 个位置灯亮，可在此时按 K1 按钮发生中断，灯开始闪烁，当灯闪烁停止后，亮灯会接着从第 4 个位置开始向左移动。中断后程序能准确地从原中断点开始恢复，主要是因为在中断服务程序中采用了保护现场和恢复现场的措施。

9.3.6 经验总结

编写中断程序要注意以下几步。

- (1) 确定中断入口地址。
- (2) 中断设置，包括开通中断、确定优先级和触发方式等，即对 4 个特殊功能寄存器 TCON、SCON、IE 和 IP 进行设置。
- (3) 在中断处理子程序里，要有保护现场和恢复现场部分，这是中断处理子程序与前面讲的子程序的不同点。

9.4 用多级外部中断控制灯移动

功能说明：开始时 P1 端口 8 只 LED 灯在闪烁。当按外部中断 INT0 开关 K1 时，使一个灯左右移 3 次。当按外部中断 INT1 开关 K2 时，使二个灯左右移 4 次。之后，8 个 LED 灯恢复闪烁。

9.4.1 硬件设计

电路设计如图 9.10 所示。

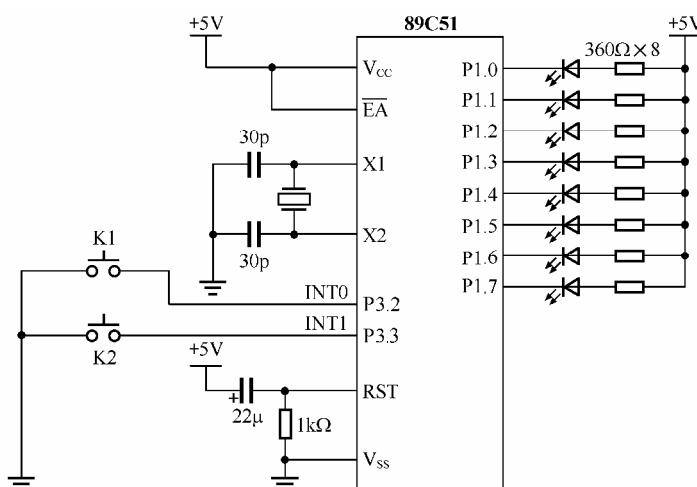


图 9.10 多级外部中断电路

本电路是在上节电路图 9.8 的基础上增加了开关 K2，K2 接在单片机 P3 端口的 P3.3 引

脚上，作为外部中断 INT1 的信号输入，其他部分与上节电路相同。

9.4.2 程序设计

本程序在上节 ZD01.ASM 程序的基础上，增加了外部中断 INT1 功能，并设置 INT1 为高优先级。

1. 流程图

程序设计流程如图 9.11 所示。

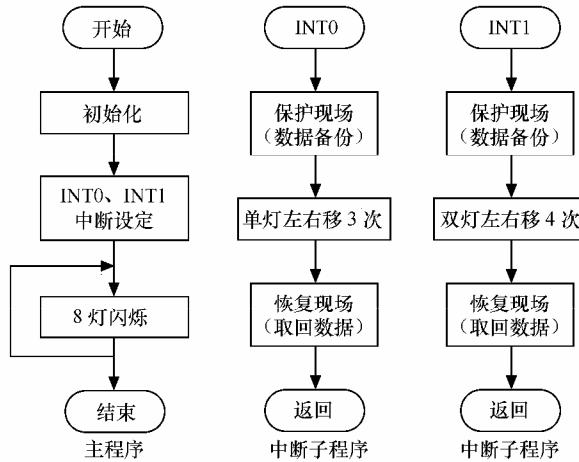


图 9.11 程序流程

2. 程序

汇编语言编写的用多级外部中断控制灯移动源程序 ZD02.ASM 代码如下：

01 :	ORG	00H	; 主程序起始地址
02 :	JMP	START	; 跳到主程序 START
03 :	ORG	03H	; INT0 中断子程序起始地址
04 :	JMP	EXT0	; 跳至中断子程序 EXT0
05 :	ORG	13H	; INT1 中断子程序起始地址
06 :	JMP	EXT1	; 跳至中断子程序 EXT1
07 : START :	MOV	IE, #10000101B	; 中断开通
08 :	MOV	IP, #00000100B	; INT1 优先中断
09 :	MOV	TCON, #00000000B	; INT0、INT1 为电平触发
10 :	MOV	SP, #70H	; 设定堆栈在 (70 H)
11 :	MOV	A, #00H	; 设初始值
12 : LOOP :	MOV	P1,A	; 使 P1 闪烁
13 :	LCALL	DELAY	; 调用延时子程序
14 :	CPL	A	; 将 A 的值取反
15 :	JMP	LOOP	; 重复循环
16 : EXT0 :	PUSH	ACC	; 将 A 值压栈

17 :	PUSH	PSW	;将 PSW 值压栈
18 :	SETB	RS0	
19 :	CLR	RS1	;设置寄存器组 1
20 :	MOV	R3, #03	;左右移 3 次
21 : LOOP1 :	MOV	R0, #08	;设置左移位数
22 :	MOV	A, #0FEH	;设置左移初值
23 : LOOP2 :	MOV	P1, A	;输出至 P1
24 :	ACALL	DELAY	;调用延时子程序
25 :	RL	A	;左移 1 位
26 :	DJNZ	R0, LOOP2	;判断移动位数
27 :	MOV	R0, #07	;设置右移位数
28 : LOOP3 :	RR	A	;右移 1 位
29 :	MOV	P1, A	;输出至 P1
30 :	LCALL	DELAY	;调用延时子程序
31 :	DJNZ	R0, LOOP3	;右移 7 位 ?
32 :	DJNZ	R3, LOOP1	;左右移 3 次 ?
33 :	POP	PSW	;从堆栈取回 PSW 值
34 :	POP	ACC	;从堆栈取回值 A 值
35 :	RETI		;返回主程序
36 : EXT1 :	PUSH	ACC	;将 A 值压入堆栈
37 :	PUSH	PSW	;将 PSW 值压入堆栈
38 :	SETB	RS1	;设工作组 2 , RS1=1
39 :	CLR	RS0	;RS0=0
40 :	MOV	R3, #04	;左右移 4 次
41 : LOOP4 :	MOV	R0, #06	;设置左移位数
42 :	MOV	A, #0FCH	;设置左移初值
43 : LOOP5 :	MOV	P1, A	;输出至 P1
44 :	ACALL	DELAY	;调用延时子程序
45 :	RL	A	;左移一位
46 :	DJNZ	R0, LOOP5	;判断移动位数
47 :	MOV	R0, #06	;设置右移位数
48 : LOOP6 :	RR	A	;右移一位
49 :	MOV	P1, A	;输出至 P1
50 :	LCALL	DELAY	;调用延时子程序
51 :	DJNZ	R0, LOOP6	;判断
52 :	DJNZ	R3, LOOP4	;左右移 4 次 ?
53 :	POP	PSW	;从堆栈取回 PSW 值
54 :	POP	ACC	;从堆栈取回值 A 值
55 :	RETI		;返回主程序
56 : DELAY :	MOV	R5, #20	;延时 0.2s 子程序
57 : DLY1 :	MOV	R6, #20	
58 : DLY2 :	MOV	R7, #248	
59 :	DJNZ	R7, \$	

```

60 :      DJNZ      R6, DLY2
61 :      DJNZ      R5, DLY
62 :      RET
63 :      END          ;程序结束

```

9.4.3 代码详解

1. 标号说明

START：起始程序的进入点。
 LOOP：闪烁程序循环的进入点。
 EXT0：INT0 中断子程序的进入点。
 LOOP1：一个灯左右移循环的进入点。
 LOOP2：一个灯左移循环的进入点。
 LOOP3：一个灯右移循环的进入点。
 EXT1：INT1 中断子程序的进入点。
 LOOP4：两个灯左右移循环的进入点。
 LOOP5：两个灯左移循环的进入点。
 LOOP6：两个灯右移循环的进入点。
 DELAY：延时子程序的进入点

2. 程序分析解释

01~06：地址设置。通过 ORG 指令，设置 03H 为外部中断 0 (EXT0) 的入口地址；设置了 13H 为外部中断 1 (EXT1) 的入口地址。

07~10：中断设置。设置外部中断 EXT0 和 EXT1 能够开通，即允许；设置 EXT1 为高优先级；设置 EXT0、EXT1 为电平触发方式；设置堆栈指针地址为 70H。

11~15：使 LED 闪烁循环。

16~35：EXT0 中断子程序，由 3 部分组成：保护现场部分、一个灯左右移循环部分和恢复现场部分。

36~55：EXT1 中断子程序，由 3 部分组成：保护现场部分、两个灯左右移循环部分和恢复现场部分。

EXT1 中断子程序与 EXT0 中断子程序基本结构是一致的，但有两点不同要注意。

- 在保护现场部分，EXT0 中断子程序使用第 1 组工作寄存器，RS1=0、RS0=1，在 EXT1 中断子程序里使用第 2 组工作寄存器，RS1=1、RS0=0。

- 在 EXT0 中断子程序中是一个灯左右移循环，而在 EXT1 中断子程序中是两个灯左右移循环。

是“一个”还是“两个”主要由初始值的设置决定，在 EXT0 中断子程序中灯左右移初始值是 FEH (11111110B)，而在 EXT1 中断子程序中灯左右移初始值是 FCH (11111100B)，0 表示灯亮，两个 0 表示两个灯同时亮，故在 EXT1 中断子程序中是两个灯左右移循环。

56~62：延时 1s 子程序。

63：程序结束。

3. 至本章用过的指令归类

- 数据传送类指令：MOV、MOVC、PUSH、POP。
- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、CLR、ANL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI。
- 位操作类指令：CPL、CLR、SETB、JB、JNB。
- 伪指令：END、DB、ORG。

9.4.4 模拟仿真

模拟仿真前注意事项。

- 将第 13、24、30、44、50 行调用延时语句前加上分号，使程序运行时跳过调用延时语句，节省模拟运行的时间。
- 将第 15 行语句“JMP LOOP”改为“JMP EXT0”，使程序能运行标号为 EXT0 的中断子程序（相当于按中断开关 K1）。
- 在 35 行语句 RETI 前加分号，使 EXT0 中断子程序返回指令 RETI 失效，使程序能向下运行，以便调试和观察 EXT1 中断子程序运行情况。

在模拟仿真中注意观察事项请参考上节。

9.4.5 实例测试

接通电源后会看到 8 只灯在闪烁，这时要注意观察。

- 当按开关 K1 产生 EXT0 中断信号时，8 只 LED 中一个灯左右移 3 次，之后，恢复闪烁。
- 当按开关 K2 产生 EXT1 中断信号时，8 只 LED 中两个灯左右移 3 次，之后，恢复闪烁。
- 当按开关 K1 使一只灯左右移 1 次时再按开关 K2，立即开始两只灯左右移 3 次，然后再继续完成 EXT0 中没有完成的一只灯移动次数，即一只灯再接着移动 2 次数。这就是 INT1 高优先级中断的结果，它可以使 INT0 低优先级中断再次被中断。
- 如果先按 K2 开关，在两只灯左右移 3 次没进行完时按开关 K1，两只灯左右移不受任何影响，直到做完规定次数。这说明在进行 INT1 高优先级中断时，不会受 INT0 低优先级中断的影响。

9.4.6 经验总结

本实例中两个外部中断 INT0 和 INT1 同时存在，均采用电平触发方式，并设置 INT1 为高优先级、INT0 为低优先级。

利用中断技术，外围设备在任何时刻都可以发出中断请求，CPU 接到请求后可以中断正在运行的程序，转去执行中断服务子程序内容，以满足实时控制的要求。

在执行低优先级中断服务程序时，可以被高优先级中断再次中断，但是低优先级中断请求不能中断高优先级中断服务程序。如果一个中断请求已被响应，则同级的其他中断响应会被禁止。



第 10 章 定时器/计数器

80C51 系列单片机的 51 子系列内部有两个定时器/计数器，它既可以作为定时器使用，也可以作为计数器使用。定时器/计数器可以用于对某种事件的计数结果进行控制，或按一定时间间隔进行控制。

10.1 定时器/计数器的用途及工作原理

10.1.1 定时器/计数器的用途

在单片机应用技术中，往往需要定时检查某个参数，或按一定时间间隔来进行某种控制；有时还需要根据某种事件的计数结果进行控制，这就需要单片机具有定时和计数功能。单片机内的定时器/计数器正是为此而设计的。

定时功能虽然可以用延时程序来实现，但这样做是以降低 CPU 的工作效率为代价的，定时器则不影响 CPU 的效率。由于单片机内集成了硬件定时器/计数器部件，这样就简化了应用系统的设计。

10.1.2 定时器/计数器的结构

80C51 系列单片机的 51 子系列内部有两个 16 位定时器/计数器，简称定时器 0 和定时器 1，分别用 T0 和 T1 表示，52 子系列单片机还增加了另一个 16 位定时器/计数器 T2。

定时器的基本结构如图 10.1 所示。

从图中可以看出，它是由两个 16 位定时器 T0、T1 和两个寄存器 TCON、TMOD 组成。其中 T0、T1 又可分成两个独立的 8 位计数器即 TH0、TL0 和 TH1、TL1，用于存储定时器、计数器的初值；TMOD 为模式控制寄存器，主要用来设置定时器/计数器的操作模式；TCON 为控制寄存器，主要用来控制定时器/计数器的启动与停止。

10.1.3 定时器/计数器的工作原理

定时器和计数器的原理是一样的，都是进行计数操作，每次加 1，加满溢出后，再从 0 开始计数，定时器和计数器不同之处是输入的计数信号来源不同。下面以定时器 T0 为例，

说明定时器/计数器的工作原理。

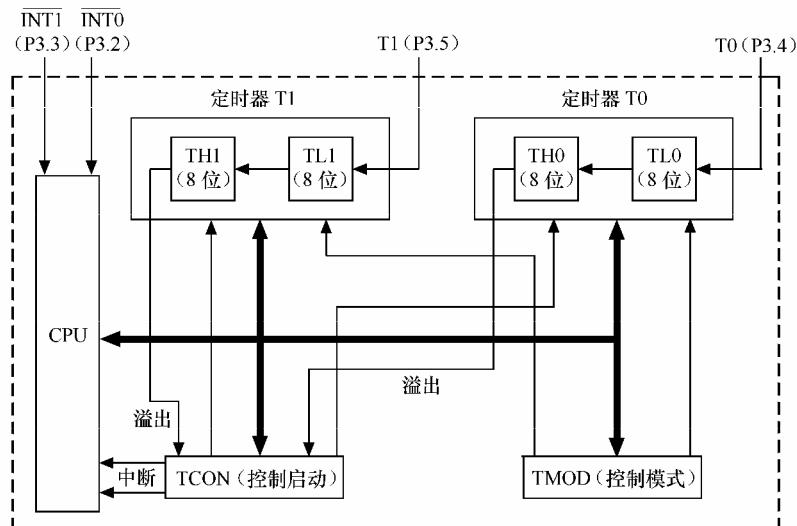


图 10.1 定时器/计数器结构框图

图 10.2 为定时器/计数器 T0 在模式 0 下的结构示意图。在这种模式下，16 位寄存器只用了 13 位，即由 TL0 的低 5 位和 TH0 的高 8 位组成的加法计数器。

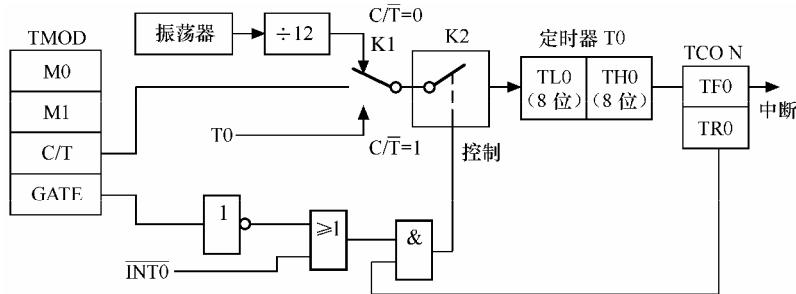


图 10.2 T0 (T1) 在模式 0 下的结构示意图

K1 为定时或计数的选择开关，由寄存器 TMOD 控制。K2 为定时或计数的启动开关，由寄存器 TCON 控制。TCON 中的 TF0 位是溢出标志位，完成定时或计数后，由此位输出信号。T0 与 T1 结构和工作过程完全相同。

1. 定时功能

在图 10.2 中，当 $C/\bar{T}=0$ 时，定时器 T0 经多路开关 K1 与振荡器的 12 分频器接通，这时计数输入信号是内部的时钟脉冲，即对机器周期进行计数，每过一个机器周期 $1\mu s$ ，计数器的值加 1。

13 位计数器的计数最大值为 8192，所以用时间为 $8192\mu s$ 。当计数到 8192 时，计数器计满溢出，即输出信号，这就是定时功能。

例如想定时 $1000\mu s$ ，首先必须使计数器初始值为 7192，从 7192 开始计数，到计满溢出发出信号时，即用了 $1000\mu s$ 。

2. 计数功能

当 $C/T=1$ 时，多路开关 K1 与引脚 T0 (P3.4) 接通，这时计数器 T0 的输入信号来自外部引脚 T0 的脉冲信号，当输入信号产生由 1 到 0 的跳变时，计数器的值就会增 1，即对脉冲信号进行计数，这就是计数功能。

10.2 定时器/计数器的控制寄存器

定时器/计数器的功能和工作模式的选择是由工作模式控制寄存器 TMOD 和定时器控制寄存器 TCON 来控制的。TMOD 和 TCON 都是 8 位特殊功能寄存器，通过对寄存器的设置来完成定时或计数功能以及工作模式的选择。

10.2.1 工作模式控制寄存器 TMOD

1. TMOD 的格式及位的含义

TMOD 是一个专用寄存器，地址为 89H，用于控制定时器 T0 和 T1 的工作方式及操作模式，其中 TMOD 的高 4 位用于对 T1 的控制，低 4 位用于对 T0 的控制，各位定义及格式如图 10.3 所示。

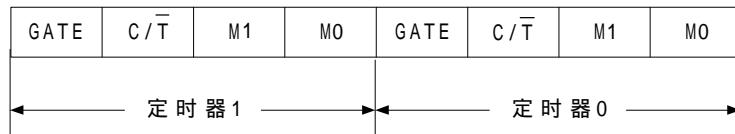


图 10.3 工作模式控制寄存器 TMOD 示意图

- GATE：门控位，用来控制定时器启动方式。

当 GATE=0 时，定时器由软件控制位 TR0 或 TR1 来控制启动。

当 GATE=1 时，定时器由外中断请求信号 (INT0 或 INT1) 来控制启动。

- C/T ：定时或计数方式选择位。

当 $C/T=0$ 时，为定时工作方式。

当 $C/T=1$ 时，为计数工作方式。

- M1、M0：工作模式选择位。两个工作模式选择位可以形成 4 种编码，对应 4 种工作模式，如表 10.1 所示。

表 10.1 工作模式表

M1	M0	工作模式	M1	M0	工作模式
0	0	模式 0	1	0	模式 2
0	1	模式 1	1	1	模式 3

此外，中断允许控制寄存器 IE 中的 EX 位和中断优先级控制寄存器 IP 中的 PX 位，分别

用来禁止/开放定时器的中断和进行优先级设定。

2 . TMOD 设置举例

假定，程序需要使用定时器 T0，并设置 T0 工作在模式 1。工作模式控制寄存器 TMOD 的低 4 位是用来对 T0 进行控制的，而在低 4 位中的 M1 和 M0 的组合是用来确定工作模式的，只有当 M1=0、M0=1 时，才是工作模式 1，可以通过下面语句实现：

```
MOV TMOD, #00000001B ; 设置 T0 工作在模式 1
```

10.2.2 定时器控制寄存器 TCON

1 . TCON 的格式及位的含义

TCON 既参与中断控制，又参与定时器/计数器控制，其中与中断控制有关的部分可参见第 9 章第 2 节，此处只介绍与定时器/计数器有关的控制位，如图 10.4 所示。

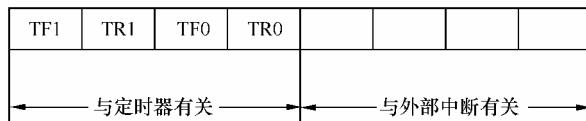


图 10.4 定时器控制寄存器 TCON 示意图

- TR0：定时器/计数器 T0 的运行控制位。TR0=0 时，停止定时器/计数器 T0；TR0=1 时，启动定时器/计数器 T0。
- TR1：定时器/计数器 T1 的运行控制位，其功能与 TR0 类同。
- TF0：片内定时器/计数器 T0 溢出标志位。当定时器/计数器 0 发生溢出时，此位由硬件置 1，并可申请中断。在中断响应完成后转向中断处理子程序时，再由硬件自动清 0。TF0 也可作为程序查询的标志位，在查询方式下由应用软件来清 0。
- TF1：片内定时器/计数器 T1 溢出标志位，其功能与 TF0 类同。

2 . TCON 的设置举例

例如，程序需要使用定时器 T0，首先要启动定时器 T0 开始工作，这就需要将 TCON 中的定时器运行控制位 TR0 设置为 1，可以用两种语句之一来实现：

```
MOV TCON, #0001000B ; 启动 T0 开始工作
SETB TR0             ; 启动 T0 开始工作
```

前一条语句将 8 位二进制数送入寄存器 TCON，此时 TR0 对应位是 1，故能启动 T0。

后一条语句中的 SETB 是位操作指令，作用是将 TR0 位设置为 1，故也能启动 T0。

10.2.3 4 种工作模式的特点

每个定时器/计数器既可选择定时功能，也可选择计数功能，而且定时器 T0 或 T1 还可以根据使用位的多少组成 4 种工作模式，其特点如下。

1. 最大定时时间不同(采用12MHz的晶体)

模式0(13位)时: $T_{MAX}=8192\mu s=8.192ms$ 。

模式1(16位)时: $T_{MAX}=65536\mu s=65.536ms$ 。

模式2和3(两个8位)时: $T_{MAX}=256\mu s=0.256ms$ 。

2. 加载方式不同

模式0和模式1的最大特点是计数溢出后,计数器全为0,因此,循环定时或计数时就要反复设置计数初值;模式2可以自动加载计数初值;模式3适合用于串行口数据传输率发生器。

10.3 定时器/计数器的初始化设置

使用定时器定时或计数时,需要先对定时器进行设置,即进行初始化,初始化步骤如下。

(1) 确定工作方式,并写入TMOD寄存器中。

采用不同工作方式,其定时或计数长度不一样,在定时或计数时不能超过此最大值。

(2) 将计算出的定时或计数初始值装入TL0、TH0或TL1、TH1中。

以定时器T0在方式1下定时0.95ms为例,先将0.95换成十六进制数FC4AH,然后将此数的高位FC装入TH0;再将此数的低位4A装入TL0。

还有一种简便计算方法,即将设计时初始值直接代入表中所提供的公式里,如使用定时器T0在模式0下定时1000μs,设计初始值为1000,将1000分两次代入公式即可。

(3) 启动定时器工作。

可使用SETB TR0或SETB TR1启动。此外,如果采用中断方式,还需要对IE位进行设置。

表10.2、表10.3、表10.4、表10.5和表10.6分别为4种工作方式的初始化步骤。

10.3.1 模式0的初始化步骤

模式0的初始化步骤如表10.2所示。

表10.2 模式0的初始化步骤

第1步	设置定时或计数	设置定时	T0	MOV TMOD, #00000000B
			T1	MOV TMOD, #00000000B
第2步	装入计数初值	设置计数	T0	MOV TMOD, #00000100B
			T1	MOV TMOD, #01000000B
第3步	启动定时器		MOV TLX, #(8192-初始值)MOD 32	
			MOV THX, #(8192-初始值)/32	
			SETB TRX	

注:X=0表示T0,X=1表示T1。

10.3.2 模式 1 的初始化步骤

模式 1 的初始化步骤如表 10.3 所示。

表 10.3 模式 1 的初始化步骤

			T0	MOV TMOD, #00000001B
第 1 步	设置定时或计数	T1	MOV TMOD, #00010000B	
		设置定时	T0	MOV TMOD, #00000101B
			T1	MOV TMOD, #01010000B
第 2 步	装入计数初值	装入低位	MOV TLX, #LOW(65536-初始值)	
		装入高位	MOV THX, #HIGH(65536-初始值)	
第 3 步	启动定时器	SETB TRX		

注 : X=0 表示 T0 , X=1 表示 T1。

10.3.3 模式 2 的初始化步骤

模式 2 的初始化步骤如表 10.4 所示。

表 10.4 模式 2 的初始化步骤

			T0	MOV TMOD, #00000010B
第 1 步	设置定时或计数	T1	MOV TMOD, #00100000B	
		设置定时	T0	MOV TMOD, #00000110B
			T1	MOV TMOD, #01100000B
第 2 步	装入计数初值	装入低位	MOV TLX, # (256-初始值)	
		装入高位	MOV THX, # (256-初始值)	
第 3 步	启动定时器	SETB TRX		

注 : X=0 表示 T0 , X=1 表示 T1。

10.3.4 模式 3 的初始化步骤

模式 3 的初始化步骤如表 10.5 和 10.6 所示。

表 10.5 模式 3 的初始化步骤 (TL0)

			T0	MOV TMOD, #00000011B
第 1 步	设置定时或计数	T1		
		设置定时	T0	MOV TMOD, #00000111B
			T1	
第 2 步	装入计数初值	装入低位	MOV TL0, # (256-初始值)	
第 3 步	启动定时器	SETB TR0 (启动定时器 T0)		

表 10.6

模式 3 的初始化步骤 (TH0)

第 1 步	设置定时或计数	设置定时	T0	MOV TMOD, #00000011B
		设置计数		
第 2 步	装入计数初值			
		装入高位	MOV TH0, # (256-初始值)	
第 3 步	启动定时器	SETB TR1 (启动定时器 T0)		

10.4 使用定时器延时

功能说明：开始时接在单片机 P1 端口中的 P1.7 亮，用定时器延时 60ms 后 P1.6 亮，依次向右移动，当最右端 P1.0 亮后又回到最左端重新开始向右移动，不断循环。

10.4.1 硬件设计

电路设计如图 10.5 所示。

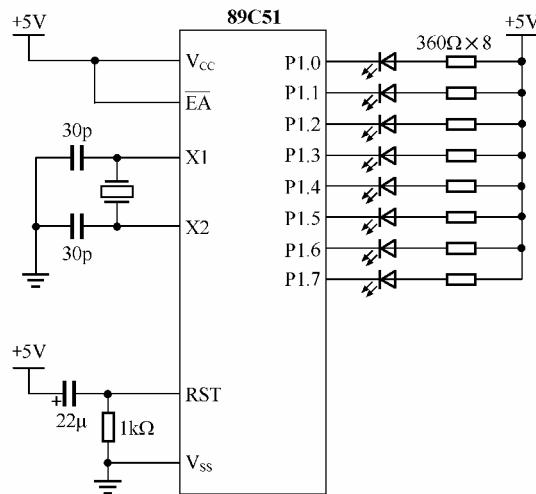


图 10.5 定时器应用实验电路

10.4.2 程序设计

本程序的功能与第 6 章 6.5 节程序 JS04.ASM 相同，使灯向右移动。不同的是前者采用软件计数方法延时，本节程序采用了定时器延时。

1. 流程图

程序设计流程如图 10.6 所示。

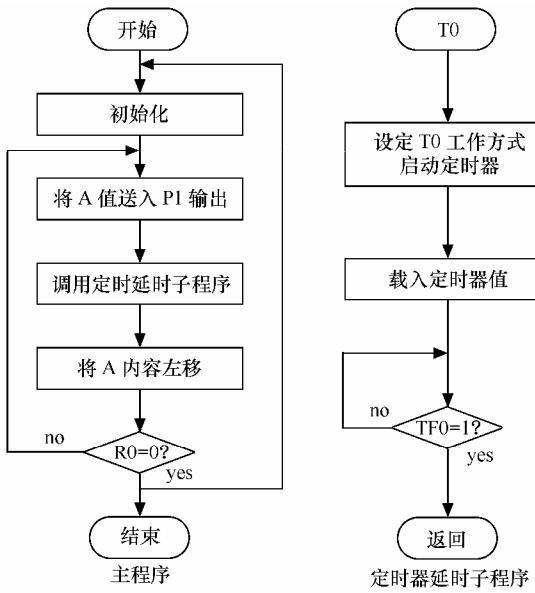


图 10.6 程序流程

2. 程序

汇编语言编写的使用定时器延时源程序 DS01.ASM 代码如下：

01: START: MOV	R0, #8	;设右移 8 次
02: MOV	A, #01111111B	;存入开始点亮灯位置
03: LOOP: MOV	P1, A	;传送到 P1 并输出
04: ACALL	DELAY	;调用延时子程序
05: RR	A	;右移 1 位
06: DJNZ	R0, LOOP	;判断移动次数
07: JMP	START	;重新设定显示值
08: DELAY: MOV	TMOD, #00000001B	;设定 T0 工作在 MODE1
09: SETB	TR0	;启动 T0 开始计时
10: MOV	TLO, #LOW(65536-60000)	;装入低位
11: MOV	TH0, #HIGH(65536-60000)	;装入高位
12: JNB	TF0, \$;T0 没有溢出，等待
13: CLR	TF0	;产生溢出，清除标志位
14: RET		;子程序返回
15: END		;程序结束

10.4.3 代码详解

1. 标号说明

START：起始程序的进入点。

LOOP：循环执行的进入点。

DELAY：定时器延时子程序的进入点。

2. 寄存器使用分配情况

程序中 A、P、R0 和 TMOD 为寄存器，其中 A、P 为特殊功能寄存器。

R0：移动位数的计数器。

TMOD：定时器工作模式控制寄存器。

TR0 和 TF0：定时器 0 控制寄存器 TCON 的两个控制位。

TL0 和 TH0：定时器 0 的计数器，TL0 为低 8 位，TH0 为高 8 位。

3. 程序分析解释

01 ~ 07：主程序，完成灯的右移、循环。

08 ~ 14：定时延时子程序。

08：设定 T0 工作在模式 1。

语句中 TMOD 是工作模式控制寄存器，它是 8 位寄存器。

其中低 4 位用来控制定时器 0。将二进制数 00000001 送入寄存器 TMOD 后，低 4 位中的第 1 位和第 2 位是工作模式选择位，其位值组成 01，表示 T0 工作在模式 1 下；第 3 位是定时器或计数器方式选择位，其值为 0，表示选择了定时器方式；第 4 位是定时器或计数器的运行标志位，其值为 0，表示只要 TR0 设置为 1，就能启动定时器 T0 工作。

09：设置 TR0 为 1，启动 T0 开始计时。

10 ~ 11：装入初始值。程序中要求用定时器延时 60ms，即计数 60000。将定时器的最大值 65536 减去初始值 60000 作为计数的初始值，装入计数器。

12：判断 TF0 位是 0 还是 1，如果是 0 说明定时的时间没到，继续等待；如果是 1，说明定时时间已到，程序向下运行。

13：将溢出标志位 TF0 清 0，为下次使用定时器做准备。

14：定时延时子程序返回。

15：程序结束。

10.4.4 模拟仿真

重点观察定时器延时子程序的运行并理解定时的工作原理。

1. 模拟仿真前注意事项

(1) 将第 10、11 语句中的 60000 改为 16，目的是节省模拟运行时间，便于观察。

(2) TR0 和 TF0 是 TCON 的两个控制位，所以要通过观察 TCON 值的变化来了解 TR0 和 TF0 的控制作用。

2. 模拟仿真中注意事项

(1) 在运行 08 语句后，TMOD 的值由 00 变为 01，01H=00000001B，设置定时器 T0 工

作在模式 1 的状态下。

(2) 运行 09 语句后，TCON 的值由 00 变为 10，10H=0001000B，使 TR0 位的值为 1，启动定时器 T0 开始计时。

(3) 运行第 10、11 语句，开始装入初始值，TL0 由 00 变 F0 再变为 F2。

(4) 运行第 12 语句时，会看到 TL0 的值每次都增加 2，这是因为执行装入初始值的语句需要用两个机器周期才能完成。当计完 16 后，此值变为 00。此时再看 TCON 的值已变为 30 (30H=00110000B)，表明 TCON 中的溢出标志位（第 6 位）TF0 为 1，说明已经计满溢出，完成了定时任务。

(5) 运行第 13 语句，将 TF0 位清 0，为重新开始定时做准备，此时 TCON 的值又变为 10。

(6) 运行第 14 语句，RET 是子程序返回指令，返回到主程序中。

10.4.5 实例测试

将写入程序的单片机插入实验板插座上，检查无误后接通电源，会看到亮灯从左向右移动，当移动到最右端后，马上跳回到最左端，再从左向右移动，不断循环。

亮灯移动的速度是由定时器延时时间决定，定时器延时时间长，亮灯移动的速度慢；延时时间短，亮灯移动的速度快。

可以通过改变定时器延时时间来观察移动速度，改变定时器延时时间的方法是在下边两行程序中改变括号内减数的大小，减数的有效范围为 1 ~ 65536，每一个数为 1 μ s，例如，将定时器延时时间改为 10ms，需要将括号内减数改写为 10000，即：

```
MOV TL0, #LOW(65536-10000)      ;装入低位
MOV TH0, #HIGH(65536-10000)      ;装入高位
```

改变定时器延时时间进行实验的目的是为了增强对定时器使用的理解。

10.4.6 经验总结

延时子程序是编写程序中不可缺少的一部分，实现延时可以采用软件延时方法，也可以采用定时器延时，本节程序的延时是采用了定时器。

10.5 定时器加软件计数延时

功能说明：开始时 P1.0 亮，延时 10s 后，左移使 P1.1 亮，如此左移到第 6 个灯（P1.5）亮后，再从头开始向左移动，1 分钟循环一次。

10.5.1 程序设计

本程序在上节定时器延时的基础上，增加了软件计数器 R1，使之通过重复定时次数，增加延时时间。

1. 流程图

程序设计流程如图 10.7 所示。

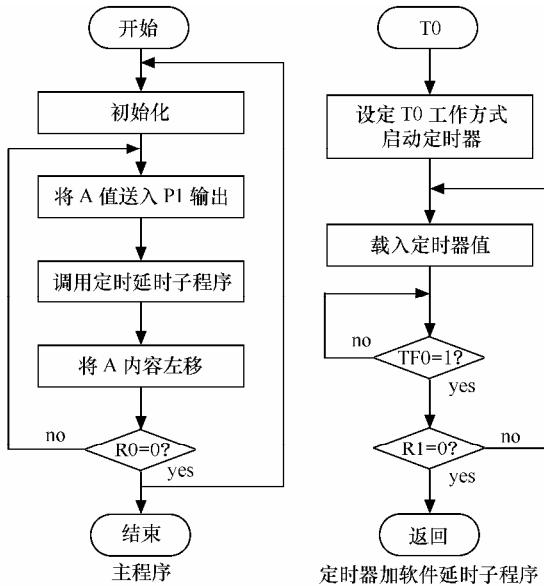


图 10.7 程序流程

2. 程序

汇编语言编写的定时器加软件计数延时源程序 DS02.ASM 代码如下：

01: START: MOV	R0, #6	; 设置左移 6 次
02: MOV	A, #11111110B	; 存入开始点亮灯位置
03: LOOP: MOV	P1, A	; 传送到 P1 并输出
04: ACALL	DELAY	; 调用延时子程序
05: RL	A	; 左移 1 位
06: DJNZ	R0, LOOP	; 判断移动次数
07: JMP	START	; 重新设定显示值
08: DELAY: MOV	R1, #200	; 设置软件计数初值
09: MOV	TMOD, # 00000001B	; 设定 T0 工作在 MODE1
10: SETB	TR0	; 启动 T0 开始计时
11: AGAIN: MOV	TLO, # LOW(65536-50000)	; 装入低位
12: MOV	TH0, # HIGH(65536-50000)	; 装入高位
13: LOOP1: JBC	TF0, LOOP2	; TF0 是 1 跳转至 LOOP2 并清 0
14: JMP	LOOP1	
15: LOOP2: DJNZ	R1, AGAIN	; R1 不是 0，则跳转至 AGAIN
16: CLR	TR0	; 是，则停止 T0 计时
17: RET		; 延时子程序返回
18: END		; 程序结束

10.5.2 代码详解

1. 标号说明

START：起始程序的进入点。
 LOOP：左移位循环执行的进入点。
 DELAY：定时器加软件计数延时子程序的进入点。
 AGAIN：循环装入计数初始值的进入点。
 LOOP1：判断 TF0 是否溢出的进入点。
 LOOP2：判断 R0 是否为 0 的进入点。

2. 寄存器使用分配情况

本节程序与上节相比增加了寄存器 R1，R1 为软件计数器，其他寄存器使用情况与上节相同。

3. 程序分析解释

本程序在上节定时器延时的基础上增加了软件计数器 R1，R1 内存入数值 200，意味着每次调用延时子程序时，定时器在子程序里将重复定时 200 次。

下面仅就新增加的部分加以说明。

08：设置软件计数初值，R1 内存入数值 200。

13：该语句是一条判断语句。判断定时器溢出标志位 TF0 的值是 0 还是 1，如果 TF0 是 0，则说明定时的时间没完成，程序运行下一行，下一行的作用是返回上一行语句，继续判断并等待定时的完成；如果 TF0 是 1，程序跳转到标号 LOOP2 处运行。

15：该语句也是一条判断语句，判断 R1 值是否减到 0，如果不是 0，则跳转到标号 AGAIN 处，重新开始定时；如果 R1 的值是 0，说明定时已经重复了 200 次，程序向下运行，执行第 16 行语句，使定时运行位 TR0 清 0，即停止定时。

4. 边用边学指令

本节程序新用到的指令是 JBC。

JBC 与前边讲过的 JB 和 JNB 都是位操作中的条件转移指令。

JNB：如果直接寻址的位值为 0，则执行跳转。

JB：如果直接寻址的位值为 1，则执行跳转。

JBC：如果直接寻址的位值为 1，则执行跳转，然后清 0 直接寻址位。

10.5.3 模拟仿真

模拟仿真前，要将第 08 行语句中的 R1 值减小，同时将第 11 和 12 行语句中的初始值（50000）减小。目的是节省模拟运行时间，便于观察。

模拟仿真中，注意观察两条判断语句（13 和 15）的运行，只有当 R1 的值为 0 时，子程序才返回主程序，否则将不断地重复定时。

10.5.4 实例测试

将写入程序的单片机插入实验板插座上，检查无误后接通电源，会看到亮灯每隔 10s 向左移动一位，1 分钟循环一次。

可以改变延时子程序的延时时间，来观察亮灯移动速度。延时时间取决于定时器的定时时间和软件计数器 R1 的数值，在定时时间不变的情况下，改变计数器 R1 的数值可以非常方便地调节延时时间，R1 的数字每改变 1，时间就会改变 50ms。

10.5.5 经验总结

定时器的延时时间是有限的，采用工作模式 1 最长延时时间为 65536ms。但是可以通过与软件计数相结合，增加延时时间。

常时间的延时设计，一般是采用工作模式 1，使定时器定时的时间为 40ms 或 50ms，再将计数器 R 赋值 250 或 200，使之产生 1s 标准时间，然后仍然通过软件的办法对 1s 标准时间再进行增加，后边讲到的时钟设计是采用此种方法。

10.6 定时与计数演示灯

功能说明：采用两个定时器，T0 设置为定时方式，T1 设置为计数方式。T0 定时时间为 50ms，计满产生的输出信号由 P1.7 口发光二极管显示。

同时还将该信号输入到计数器 T1（P3.5 脚），作为 T1 的计数输入脉冲，当输入信号产生由 1 到 0 的跳变时触发计数器工作，使计数器的值增 1。

计数器 T1 的初始值设置为 100，计满时所用的时间为 $50\text{ms} \times 2 \times 100$ ，即 10s，并由 P1.0 端口发光二极管来显示，所以 P1.0 端口的灯每 10s 点亮或熄灭一次。

10.6.1 硬件设计

电路设计如图 10.8 所示。

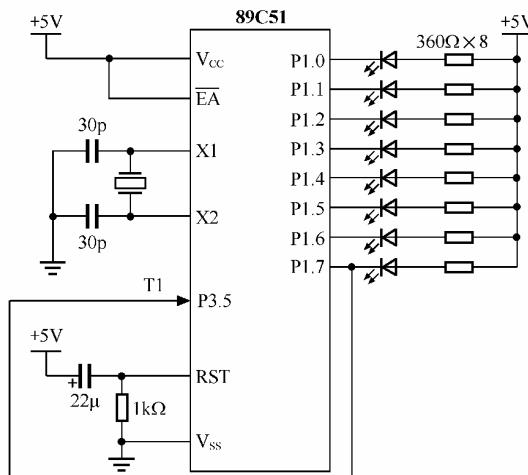


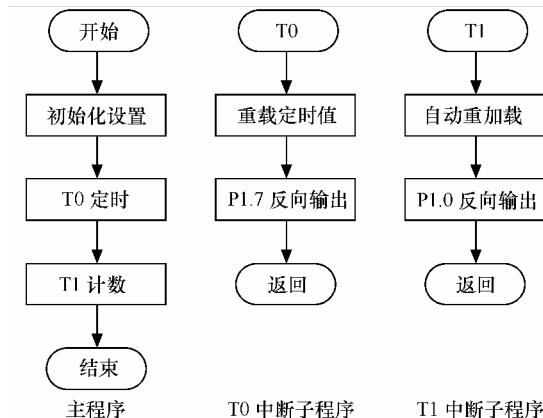
图 10.8 定时器与计数器串联

P1.7 端口是定时器 T0 的输出端，由该端口引出一根线接到计数器 T1 的输入端 P3.5。T0 定时输出两次，T1 计数一次。在电路中，P1.7 端口接的发光二极管显示定时器输出状态，P1.0 端口接的发光二极管显示计数器输出状态。

10.6.2 程序设计

在本程序中两个定时器 T0 和 T1 串联，设定 T0 为定时器，T1 为计数器。

1. 流程图



程序设计流程如图 10.9 所示。

图 10.9 程序流程

2. 程序

汇编语言编写的定时与计数演示灯源程序 DS03.ASM 代码如下：

01:	ORG	0000H	;起始地址
02:	JMP	MAIN	;跳转到主程序 MAIN 处
03:	ORG	000BH	;定时器 T0 溢出入口地址
04:	JMP	EXT0	;跳到定时器中断 EXT0 处
05:	ORG	001BH	;计数器 T1 溢出入口地址
06:	JMP	EXT1	;跳到计数器中断 EXT1 处
07:	MOV	SP, #60H	;设置堆栈
08: MAIN:	MOV	TMOD, #01100001B	;设置 T0 工作在模式 1，T1 工作在模式 2
09:	MOV	TL0, #LOW(65536-50000)	;装入定时初值
10:	MOV	TH0, #HIGH(65536-50000)	
11:	MOV	TL1, #(256-100)	;装入计数初值
12:	MOV	TH1, #(256-100)	
13:	MOV	IE, #10001010B	;开总中断及 T0、T1 中断
14:	SETB	TR0	;启动 T0 开始定时
15:	SETB	TR1	;启动 T1 开始计数
16:	JMP	\$;等待溢出
17: EXT0:	MOV	TL0, #LOW(65536-50000)	;重加载
18:	MOV	TH0, #HIGH(65536-50000)	
19:	CPL	P1.7	;P1.7 反相输出，并输入 P3.5
20:	RETI		;T0 中断子程序返回
21: EXT1:	CPL	P1.0	;P1.0 反相输出
22:	RETI		;T1 中断子程序返回
23:	END		;程序结束

10.6.3 代码详解

1. 标号说明

MAIN：主程序的进入点。

EXT0：定时器 T0 中断子程序的进入点。

EXT1：定时器 T1（作计数器用）中断子程序的进入点。

2. 寄存器使用分配情况

程序 DS03.ASM 与上节相比增加了寄存器 SP。SP 是一个堆栈指针寄存器，在调用子程序或执行中断子程序时，SP 先加一，程序计数器的值会被压入堆栈中。而在执行子程序返回指令 RET 或 RETI 时，存放在堆栈中的 PC 值也会回存至程序寄存器中，以保证正确地回到原程序调用点继续执行。

3. 程序分析解释

在本节程序中两个定时器 T0 和 T1 串联。

第 13 行语句设定 T0 为定时器，T1 作为计数器，并开放中断。T0 定时溢出产生中断信号后，该信号一方面通过引线输入到计数器的输入端 P3.5，作为计数器的控制信号；另一方面由程序标号 EXT0 处转入到中断子程序，启动中断处理子程序工作。

在中断处理子程序里完成两项任务：一是重新向 TL0 和 TH0 装入计数初始值（定时器工作在模式 1 方式下，计数溢出后，需要重新装入初始值）；二是使 P1.7 作反相输出。然后执行中断返回指令 RETI，返回到主程序。

定时器 T1 作计数器使用，当计满产生中断信号后，由程序标号 EXT1 转入到 T1 的中断子程序，使 T1 的中断处理子程序工作。由于 T1 采用模式 2 的工作方式，在主程序计满溢出后会自动重新装入计数初始值，所以，在子程序里不用像模式 1 那样再设置初值，只完成 P1.0 的反相输出。

4. 本章用过的指令归类

- 数据传送类指令：MOV、MOVC、PUSH、POP。
- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、CLR、ANL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI。
- 位操作类指令：CPL、CLR、SETB、JB、JNB、JBC。
- 伪指令：END、DB、ORG。

定时器部分的模拟仿真可参照上节。

10.6.4 实例测试

实例测试时，注意观察 P1.7 和 P1.0 口灯的亮、灭变化。其中 P1.7 口发光二极管显示的是定时器 T0 的中断输出信号，定时器的定时时间为 50ms，即每隔 50ms 产生一次中断信号，

使 P1.7 反相输出一次。所以 P1.7 口灯亮的时间反映 T0 定时器的定时时间。

P1.0 口发光二极管显示的是计数器 T1 的计数中断输出信号，T0 定时中断每输出两次，T1 计数器计数 1 次，当计数到 100 时产生中断信号，使 P1.0 反相输出一次。所以 P1.0 灯亮的时间反映的是 T1 计数器计满 100 所使用的时间，该时间为 10s。

可以改变 T1 计数器的计数值，观察 P1.0 灯亮灭情况。

10.6.5 经验总结

单片机内 T0 和 T1 既可以做定时器，也可以做计数器。定时和计数的工作原理是一样的，都是计数操作，每次加 1，加满溢出输出后，再从 0 开始计数。定时与计数的不同是输入信号来源不同，定时器的输入信号是内部的时钟脉冲，计数器的输入信号是来自外部引脚的脉冲信号，如在本节程序中将 P1.7 的反相输出信号做 T1 计数器的输入信号，当输入信号产生由 1 到 0 的跳变时，计数器的值就会加 1。



第 11 章 音乐发声

使用单片机可以进行发声，还可以进行歌曲演奏。

11.1 发声实验

功能说明：用软件延时方法，使单片机产生 1kHz 的方波，通过 P3.4 脚控制三极管导通或截止，使接在三极管上的蜂鸣器发出 1kHz 的响声。

11.1.1 硬件设计

电路设计如图 11.1 所示。

单片机的 P3.4 引脚通过限流电阻 R 与三极管基极相接，三极管的集电极接有蜂鸣器。当单片机的 P3.4 引脚电平为 0 时，三极管导通，蜂鸣器有电流流过；当 P3.4 引脚电平为 1 时，三极管截止，蜂鸣器没有电流流过。这样，在蜂鸣器两端就会出现波动的电流，波动的电流就会使蜂鸣器发声。

11.1.2 程序设计

1. 流程图

程序设计流程如图 11.2 所示。

2. 程序

汇编语言编写的发声实验源程序 FS01.ASM 代码如下：

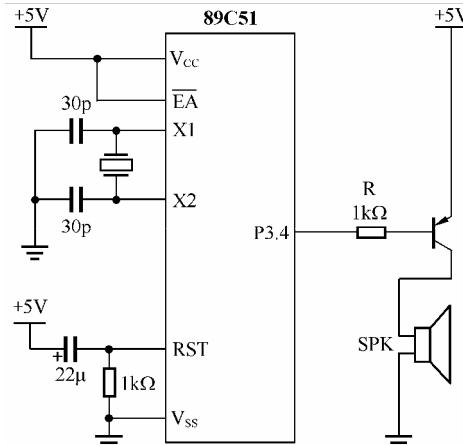


图 11.1 发声实验电路

```
01: START: CLR      P3.4          ;使 P3.4 为 0
02:      ACALL    DELAY        ;延时 500μs
03:      SETB     P3.4          ;使 P3.4 为 1
04:      ACALL    DELAY        ;延时 500μs
```

```

05:      JMP      START      ;返回循环
06:  DELAY: MOV      R6,#250    ;延时 500μs 子程序
07:      DJNZ     R6,$       ;子程序返回
08:      RET      ;程序结束
09:      END

```

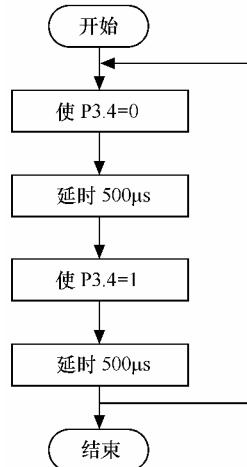


图 11.2 程序流程

11.1.3 代码详解

1. 标号说明

START：主程序的进入点。

DELAY：延时 500μs 子程序的进入点。

2. 寄存器使用分配情况

P3.4：对内是寄存器 P3 的一个位，对外是 P3 端口的第 4 个引脚。

R6：在延时程序里作计数器用。

3. 程序分析解释

01~05：输出 1kHz 方波。P3.4 引脚电平为 0 时延时 500μs，为 1 时延时 500μs，即 1 个方波需要时间为 1ms。在 1s 时间里可输出 1000 个方波。输出的方波控制着三极管的导通与截止，从而使蜂鸣器两端电流每秒钟变化 1000 次，即发出 1kHz 的声音。

06~09：延时 500μs 子程序。其中“DJNZ R6, \$”每执行 1 次需要 2 个机器周期，即 2μs，一共运行 250 次，需要的时间为 500μs。

11.1.4 模拟仿真

此程序模拟仿真比较简单，可参照以前说明进行。

11.1.5 实例测试

先检查实验板上蜂鸣器是否接在 P3.4 端口上，假如是接在 P2.4 端口，程序中 P3.4 应改为 P2.4，当检查无误后，接通电源会听到 1kHz 的响声。

改变一下延时时间再做测试，注意体会延时时间与声音变化的关系。

11.1.6 经验总结

发声原理很简单，只要使单片机 I/O 引脚不断地产生方波，三极管就会不断地导通和截止，使蜂鸣器发出声音。

控制方波的频率就能控制蜂鸣器的发声频率。在本节程序里，单片机 I/O 引脚产生方波的频率是由软件延时的时间决定的。

11.2 变频报警

功能说明：用软件延时方法实现变频振荡报警，即用 P3.4 端口输出 1kHz 和 2kHz 的变频信号以示报警，每隔 1s 交替变换 1 次。本节硬件设计与上节相同。

11.2.1 程序设计

本程序利用软件延时方法，使 P3.4 端口输出 1kHz 和 2kHz 的变频信号，每隔 1s 交替变换 1 次。

1. 流程图

程序设计流程如图 11.3 所示。

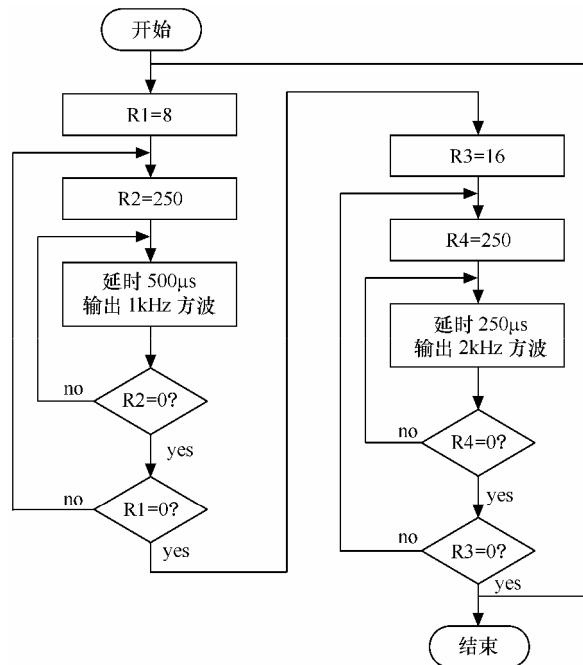


图 11.3 程序流程

2. 程序

汇编语言编写的变频报警源程序 FS02.ASM 代码如下：

01:	MAIN:	MOV	R1, #8	; 1kHz 持续时间
02:	LOOP1:	MOV	R2, #250	
03:	LOOP2:	CPL	P3.4	; 输出 1kHz 方波
04:		ACALL	DELAY2	; 调用延时 500μs 子程序
05:		DJNZ	R2, LOOP2	; 持续 1s
06:		DJNZ	R1, LOOP1	
07:		MOV	R3, #16	; 2kHz 持续时间
08:	LOOP3:	MOV	R4, #250	
09:	LOOP4:	CPL	P3.4	; 输出 2kHz 方波
10:		ACALL	DELAY1	; 调用延时 250μs 子程序
11:		DJNZ	R4, LOOP4	; 持续 1s
12:		DJNZ	R3, LOOP3	
13:		JMP	MAIN	
14:	DELAY1:	MOV	R5, #125	; 延时 250μs 子程序
15:		DJNZ	R5, \$	
16:		RET		; 子程序返回
17:	DELAY2:	MOV	R6, #250	; 延时 500μs 子程序
18:		DJNZ	R6, \$	
19:		RET		; 子程序返回
20:		END		; 程序结束

11.2.2 代码详解

1. 标号说明

START：程序开始的进入点。

LOOP1 和 LOOP2：输出 1kHz 方波，持续 1s 时间是采用双重循环来实现的。LOOP2 是内循环的进入点，LOOP1 是外循环的进入点。

LOOP3 和 LOOP4：输出 2kHz 方波，持续 1s 时间也是采用双重循环来实现的。LOOP4 是内循环的进入点，LOOP3 是外循环的进入点。

DELAY1：延时 250μs 子程序的进入点。

DELAY2：延时 500μs 子程序的进入点。

2. 寄存器使用分配情况

在程序中，P3 与 R1 ~ R6 都是寄存器。P3 是特殊寄存器，对外是输出端口；R1 ~ R6 是普通寄存器，用作计数器。

P3.4 对内是寄存器 P3 的一个位，对外是 P3 端口的第 5 个引脚。

R1 和 R2：输出 1kHz 方波时采用双重循环来实现持续 1s 所使用的两个计数器。

R3 和 R4：输出 2kHz 方波时，采用双重循环来实现持续 1s 所使用的两个计数器。

R5：延时 250μs 子程序中所用的计数器。

R6：延时 500μs 子程序中所用的计数器。

3. 程序分析解释

本程序使 P3.4 端口输出 1kHz 和 2kHz 的变频信号，每隔 1s 交替变换 1 次。

01 ~ 06：完成 1kHz 的输出，并持续 1s。

07 ~ 12：完成 2kHz 的输出，并持续 1s。

14 ~ 16：输出 1kHz 时的延时子程序；17 ~ 19 行语句为输出 2kHz 时的延时子程序。

第 01 ~ 06 行语句中，03 行语句使 P3.4 端口作反相输出，每次反相输出后，都要调用延时子程序延时 500μs，该延时的时间决定着 P3.4 端口输出的频率为 1kHz。

02 和 05 行语句组成内循环，01 和 06 行语句组成外循环。双重循环使输出 1kHz 方波时间持续为 1s，如图 11.4 所示。

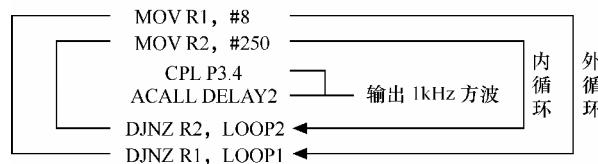


图 11.4 双重循环路线示意图

第 07 ~ 12 行语句的结构与第 01 ~ 06 行语句完全相同，只是调用的延时子程序的延时时间缩短了，输出 2kHz 方波，并持续 1s。

第 14 ~ 16 行语句与第 17 ~ 19 行语句组成两个延时子程序，延时的时间不一样，但是程序结构一样，都属于单重循环结构的延时程序。

11.2.3 模拟仿真

该程序语句行数不多，但循环较多。因此，在模拟仿真时，主要是观察程序的运行路线，弄清每次循环的意图，以便更好地理解程序。

为了节省时间，在模拟仿真之前，可以先将寄存器 R2、R4、R5 和 R6 的数值减小。

11.2.4 实例测试

将写入变频报警程序的单片机插入实验板插座内，并检查实验板上蜂鸣器是否接在 P3.4 端口上，当检查无误后接通电源，会听到变频报警声。

实例测试时注意听声音的频率变化及频率变化的时间。该程序开始时先发出 1kHz 的声音，持续 1s，接着发出 2kHz 的声音，也持续 1s。不断循环，便产生变频报警声。

可以改变程序中延时子程序 DELAY1 或 DELAY2 的延时时间，观察声音的变化；再改变一下程序中的 R1 或 R3 的数值，观察发声长度的变化，从而加深对变频报警声产生原理的理解。

11.2.5 经验总结

本节程序利用两个频率声音交替变化，产生报警声，其中延时子程序中的延时时间决定

输出声音的频率，双重循环延时的时间决定声音的长短。

11.3 歌曲演奏

功能说明：利用单片机作演奏一首生日快乐歌，本节硬件设计与 11.1 相同。

11.3.1 编程演奏器原理

1. 演奏器原理

(1) 通过控制单片机定时器的定时时间产生不同频率的音频脉冲，经放大后驱动蜂鸣器发出不同音节的声音。

(2) 用软件延时来控制发音时间的长短，控制节拍，表 11.1 是各调 1/4 节拍的时间表。

表 11.1 各调 1/4 节拍的时间

曲 调 值	延 时 时 间	曲 调 值	延 时 时 间
调 4/4	125ms	调 2/4	250ms
调 3/4	187ms		

(3) 把乐谱中的音符和相应的节拍变换为定时常数和延时常数，作为数据表格存放在存储器中。由程序查表得到定时常数和延时常数，分别用来控制定时器产生的脉冲频率和发出该音频脉冲的持续时间。

(4) 表 11.2 为单片机晶振频率为 12MHz 时，乐曲中的音符、频率及定时常数之间的对应关系表。

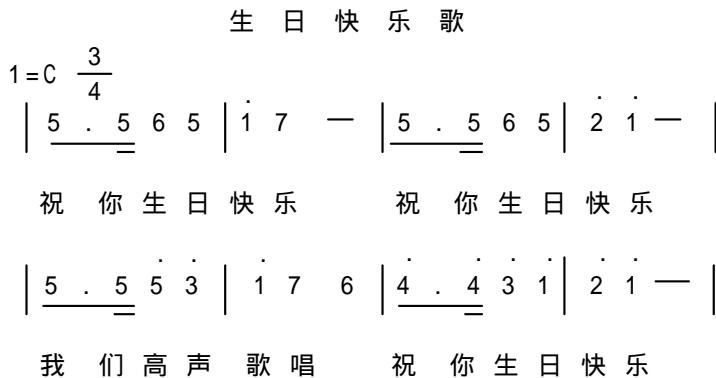
表 11.2 音符、频率及定时常数对应关系

C 调音符	频率 (Hz)	半周期 (ms)	定 时 值
1.	262	1.90	F894H
2.	294	1.70	F95CH
3.	330	1.51	FA1AH
4.	349	1.43	FA6AH
5.	392	1.28	FB00H
6.	440	1.14	FB8CH
7.	494	1.01	FC0EH
1	523	0.95	FC4AH
2	587	0.85	FCAEH
3	659	0.76	FD08H
4	698	0.72	FD30H

续表

C 调音符	频率 (Hz)	半周期 (ms)	定时值
5	784	0.64	FD80H
6	880	0.57	FDC6H
7	988	0.51	FE02H
♩	1046	0.47	FE2AH
♪	1175	0.42	FE5CH
♫	1318	0.38	FE84H
♩	1397	0.36	FE98H
♪	1568	0.32	FEC0H
♫	1760	0.28	FEE8H
♩	1967	0.25	FF06H

2. 歌谱



3. 建立步骤

(1) 先把乐谱的音符找出，然后根据表 11.2 给出的定时值按乐谱的音符顺序建立编码表 TABLE。

定时值为十六进制 4 位数，拆开分为两组，如 5 对应的定时值为 FD80H，拆分为 FDH 和 80H 两组。前组装入定时器的高位 TH0，后组装入定时器的低位 TL0。程序中将进行两次查表来完成一个音符对应的定时初值的装入。

(2) 在程序中使用定时器 T0 方式 1 来产生歌谱中各音符对应频率的音频脉冲，由 P3.4 输出，再经三极管将信号放大后驱动蜂鸣器发出不同音节的声音。

(3) 程序中节拍的控制是通过调用延时子程序 DELAY 的次数来实现，1 拍为 748 ms，即需要调用 4 次 DELAY；3/4 拍需要调用 3 次 DELAY；2/4 拍需要调用 2 次 DELAY。

(4) 节拍的控制码在表 TABLE 中位于音符码的后面。如第 1 行“DB 0FDH,80H,03H,...”中，0FDH 和 80H 是音符 5 的音符码，其后边的 03H 是节拍码，即 3/4 拍的时间。

(5) 当一个音符的发音时间到时，再查下一个音符的定时常数和延时常数。依此进行下去，就可演奏出悦耳动听的乐曲。

11.3.2 程序设计

1. 流程图

程序设计流程如图 11.5 所示。

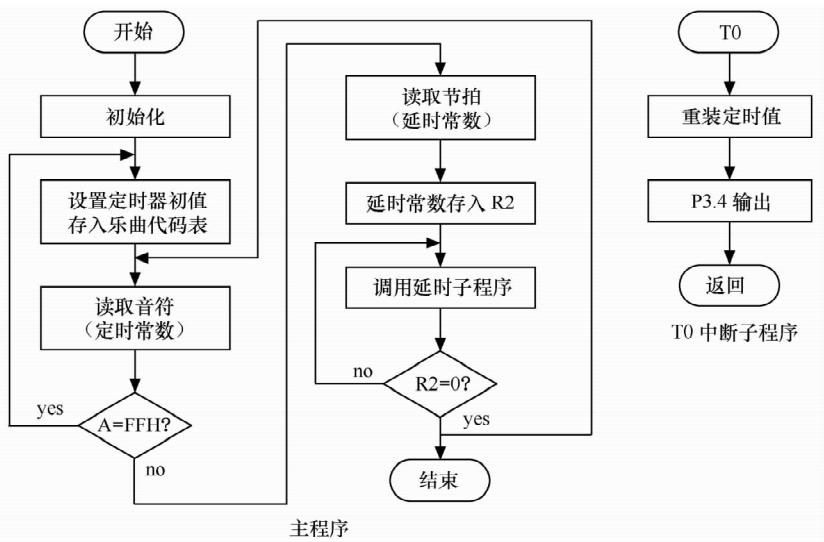


图 11.5 程序流程

2. 程序

汇编语言编写的歌曲演奏源程序 FS03.ASM 代码如下：

01 :	ORG	00H	; 主程序起始地址
02 :	JMP	START	; 跳转至主程序
03 :	ORG	0BH	; 定时器 T0 中断入口
04 :	JMP	EXT0	; 跳转至 T0 中断子程序
05 :	START:	MOV TMOD, #00000001B	; 设置 T0 方式 1
06 :	MOV IE, #10000010B		; 允许 T0 中断
07 :	MOV DPTR, #TABLE		; 存表首地址
08 :	LOOP:	CLR A	; 清 0
09 :	MOVC A, @A+DPTR		; 查表
10 :	MOV R1, A		; 定时器高 8 位存入 R1
11 :	INC DPTR		; 指针加 1
12 :	CLR A		; 清 0
13 :	MOVC A, @A+DPTR		; 查表
14 :	MOV R0, A		; 定时器低 8 位存入 R0
15 :	ORL A, R1		; 进行或运算

```

16:      JZ      NEXT0      ;全 0 为休止符
17:      MOV     A, R0
18:      ANL     A, R1      ;进行与运算
19:      CJNE    A, #0FFH, NEXT ;全 1 表示乐曲结束
20:      JMP     START      ;从头开始循环演奏
21: NEXT:   MOV     TH0, R1      ;装入高位定时值
22:      MOV     TL0, R0      ;装入低位定时值
23:      SETB    TR0       ;启动定时器 T0
24:      JMP     NEXT1      ;跳转到 NEXT1 处
25: NEXT0: CLR    TR0       ;关闭定时器，停止发音
26: NEXT1: CLR    A        ;清 0
27:      INC     DPTR      ;指针加 1
28:      MOVC   A, @A+DPTR  ;查延时常数
29:      MOV     R2, A       ;延时常数存入 R2
30: LOOP1: ACALL  DELAY     ;调用延时子程序
31:      DJNZ   R2, LOOP1   ;控制延时次数
32:      INC     DPTR      ;指针加 1
33:      JMP     LOOP       ;跳转到 LOOP 处
34: EXT0:   MOV     TH0, R1      ;重装定时值
35:      MOV     TL0, R0
36:      CPL    P3.4      ;反相输出
37:      RETI
38: DELAY: MOV    R7, #02      ;延时 187ms 子程序
39: D2:    MOV     R6, #187
40: D3:    MOV     R5, #248
41:      DJNZ   R5,$
42:      DJNZ   R6, D3
43:      DJNZ   R7, D2
44:      RET      ;延时子程序返回
45: TABLE: DB  0FDH,80H,03H, 0FDH,80H,01H  ;编码表
46:      DB  0FDH,0C6H,04H, 0FDH,80H,04H
47:      DB  0FEH,2AH,04H, 0FEH,02H,04H
48:      DB  00H,00H,04H
49:      DB  0FDH,80H,03H, 0FDH,80H,01H
50:      DB  0FDH,0C6H,04H, 0FDH,80H,04H
51:      DB  0FEH,5CH,04H, 0FEH,2AH,04H
52:      DB  00H,00H,04H
53:      DB  0FDH,80H,03H, 0FDH,80H,01H ;
54:      DB  0FEH,0C0H,04H, 0FEH,84H,04H ;
55:      DB  0FEH,2AH,04H, 0FEH,02H,04H
56:      DB  0FDH,0C6H,04H
57:      DB  0FEH,98H,03H, 0FEH,98H,01H
58:      DB  0FEH,84H,04H, 0FEH,2AH,04H

```

```

59 :      DB  0FEH,5CH,04H, 0FEH,2AH,04H
60 :      DB  00H,00H,04H
61 :      DB  OFFH,OFFH          ;结束码
62 :      END                  ;程序结束

```

11.3.3 代码详解

1. 标号说明

START：程序开始的进入点。
 LOOP：处理下一个音符的进入点。
 NEXT：装入定时初值的进入点。
 NEXT0：关闭定时器、停止发音的进入点。
 NEXT1：查找延时常数的进入点。
 LOOP1：处理节拍时间的进入点。
 EXT0：中断子程序的进入点。
 DELAY：延时 180ms 子程序的进入点。

2. 寄存器使用分配情况

R0、R1、R2、R5、R6、R7：普通寄存器。其中 R0 存放低位定时器初值；R1 存放高位定时器初值；R2 存放延时常数值；R5、R6 和 R7 在延时子程序中作计数器用。

A 和 DPTR：特殊功能寄存器，其中 A 又称累加器，DPTR 为数据指针。在程序中把表 TABLE 的首地址存入 DPTR 作基础地址，A 作为变址寄存器。将基址寄存器和变址寄存器的内容相加（@A+DPTR）形成操作数的地址。

TMOD：定时器工作模式控制寄存器。
 TL0 和 TH0：定时器 0 的计数器，TL0 为低 8 位，TH0 为高 8 位。
 TR0：定时器 0 控制寄存器 TCON 的一个控制位。
 IE：中断允许控制寄存器。
 P3.4：对内是 P3 寄存器的一个位，对外是输入/输出端口的一个引脚，作音频信号输出端口。

3. 程序分析解释

01 ~ 04：设定入口地址。其中定时器中断是从标号 EXT0 处进入中断子程序。

05 ~ 07：设置定时器 T0 为方式 1，并允许 T0 中断。其中 07 行语句将表 TABLE 的首地址存入数据指针寄存器 DPTR 中。

08 ~ 10：第一次查表，取出表中的第一个码，即乐谱中的第一个音符 5 的定时常数（定时初值）FD80H 中的高位部分 FDH，并存入寄存器 R1 中。

11 ~ 14：第二次查表，取出表中的第二个码，即乐谱中的第一个音符 5 的定时常数 FD80H 中的低位部分即 80H，并存入寄存器 R0 中。

15 ~ 20：判断、转移语句。对二次查取到的码，检查是否有休止符码，该判断是通过第 15 行语句“ORL A,R1”对 A 和 R1 寄存器的内容进行或运算实现的。

如果累加器 A 中的值是 00H , R1 寄存器的值也是 00H , 进行或运算后再存入累加器 A 中 , 此时累加器 A 中的值是全 0 , 说明取到的是休止符码。通过第 16 行语句 “ JZ NEXT0 ” , 跳转至标号 NEXT0 处 , 关闭定时器 , 停止发音 , 完成乐谱中休止符的作用。

取码时是否取到了结束码是通过第 18 行语句指令 ANL 对 A 和 R1 进行与运算来判断的。如果累加器 A 中的值是 FFH , R1 寄存器的值也是 FFH , 进行与运算后再存入累加器 A 中 , 此时累加器 A 中的值是全 1 , 说明取到的是结束码。

第 19 行语句 “ CJNE A, #FFH, NEXT ” , 如果累加器 A 中的值与#FFH 不相等 , 则跳转移到 NEXT 处 , 开始向定时器装入定时常数 ; 如果 A 与#FFH 相等 , 说明取到的码是结束码 , 程序向下执行 , 即从头开始循环演奏。

21 ~ 23 : 开始向定时器装入定时常数并启动定时器工作。

24 ~ 25 : 处理拍节。

26 ~ 29 : 查取延时码并存入 R2。

30 ~ 31 : 决定拍节时间。如果取来的码是 02H , 程序调用两次 DRLAY 延时 , 即为 2/4 拍节时间 ; 如果取来的码是 04H , 程序调用 4 次 DRLAY 延时 , 即为 1 拍节时间。

32 ~ 33 : 每次查表取码后 , 数据指针都要加 1 , 以便指向下一个待查的码。第 33 行语句将程序运行跳转到标号 LOOP 处 , 处理下一个音符。

34 ~ 37 : 中断子程序。当定时器计时满后将产生中断 , 由标号 EXT0 处进入中断子程序。在中断子程序中重装定时值 , 并通过第 36 行语句在 P3.4 端口输出音频信号。

38 ~ 44 : 延时 187ms 子程序。

45 ~ 61 : 编码表 TABLE 。在表中每个音节由 3 个码组成 , 前两个为音符码 , 后一个为节拍码。如表中第 1 行 “ 0FDH,80H,03H ” , 其中 0FDH 和 80H 即为 FD80H , 是音符 5 的发音编码 ; 03H 是节拍码 , 3/4 节拍。

表中的最后一行 “ OFFH,OFFH ” 是结束码 , 表示乐曲演奏结束。

4 . 边用边学指令

本程序用到新的指令有 ORL 和 JZ 。

- ORL : 逻辑运算及位移类指令中的按位或操作指令。该指令将累加器 A 中的内容与源操作数所指出的内容按位进行逻辑或运算 , 结果存 A 中。

- JZ : 控制转移类指令 , 功能是当累加器 A 为 0 时跳转。

11.3.4 模拟仿真

1 . 模拟仿真前注意事项

在 23 与 24 行语句之间 , 加一条 “ SETB TF0 ” 语句 , 该语句使定时器 T0 的溢出标志位 TF0 为 1 , 模拟产生中断 , 使程序能进入中断子程序中运行。

在 39 ~ 43 行语句的前边加分号 , 使延时子程序的延时时间缩短为 2ms , 缩短模拟仿真时间。

2 . 模拟仿真中注意事项

观察程序主要运行路线 : 第一次查表 第二次查表 中断子程序及返回 第三次查表

进入延时子程序及返回 处理下一个音符 (跳转到 LOOP 处)。

观察累加器 A 中值的变化。第一次查表执行第 09 行语句 “MOVC A,@A+DPTR” 后，A 值为 FD。第二次查表，执行第 13 行语句 “MOVC A,@A+DPTR” 后，A 值为 80。两次查表取回音符 5 的对应码 FD80H。第三次查表，执行第 28 行语句 “MOVC A,@A+DPTR” 后，A 值为 03，取回的是音符 5 的对应拍节码。

11.3.5 实例测试

将写入歌曲演奏程序的单片机插入实验板插座内，并检查实验板上蜂鸣器接口是否与程序中声音输出端口一致，当检查无误后接通电源，就会听到生日快乐歌。

11.3.6 经验总结

在用单片机作可编程乐曲演奏器的程序里，一般用定时器 T0 方式 1 来控制音符的频率，调用延时子程序 DELAY 来控制节拍。

程序采用查表的方法，将乐谱转换成控制码并制成表 TABLE，然后进行 3 次查表。第一、第二次查表完成对音符的控制，第三次查表读取节拍码，完成对节拍的控制。

在完成第一、第二次查取音符的控制码后，程序中还设有判断语句，如果判断出取来的是休止符码，就将定时器关闭一段时间；如果判断出取来的是结束码，程序就从头开始循环。

11.4 电子琴

功能说明：使用 4×4 矩阵式键盘设计出 16 个音符，随意弹奏。

14.4.1 硬件设计

电子琴电路设计如图 11.6 所示。

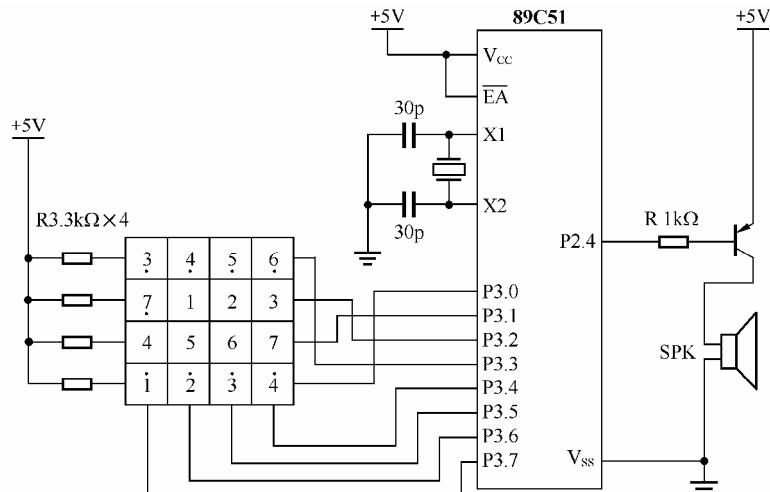


图 11.6 电子琴电路

单片机 P3 为输入端口，接有 4×4 矩阵式键盘，键盘上标出 16 个音符。键盘的 4 条行线的一端与单片机 P3 端口的 P3.0、P3.1、P3.2 和 P3.3 口相接，另一端通过上拉电阻接到 +5V 上；4 条列线的一端与 P3 端口的 P3.4、P3.5、P3.6 和 P3.7 口相接。单片机的 P2.4 端口为输出端，通过限流电阻 R 与三极管基极相接，三极管的集电极接有蜂鸣器。

11.4.2 程序设计

电子琴程序设计的思路是：设计 16 个音符，与 4×4 矩阵式键盘 16 个按键一一对应。音符通过定时器 T0 产生，然后通过键盘扫描法，将按下的键转换成相对应的音符。

1. 流程图

程序设计流程如图 11.7 所示。

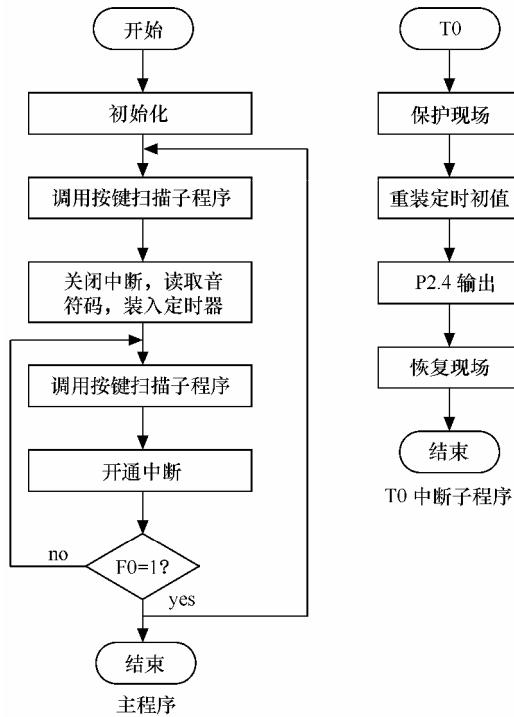


图 11.7 程序流程

2. 程序

汇编语言编写的电子琴源程序 JP04.ASM 代码如下：

01 :	ORG	00H	; 主程序起始地址
02 :	JMP	START	; 跳转至主程序
03 :	ORG	0BH	; T0 中断起始地址
04 :	JMP	TIM0	; 跳转至 T0 中断子程序
05 : START:	MOV	TMOD, #00000001B	; 设 T0 在 MODE1
06 :	MOV	IE, #10000010B	; 开通中断

07 :	SETB	TR0	;启动定时器 T0
08 : L1:	ACALL	KEY	;调用 KEY , 判断是否有键按下
09 :	CLR	EA	;中断屏蔽
10 :	JB	F0, L1	;没按下则 F0=1 , 否则 F0=0
11 :	MOV	A, 22H	;将取码指针暂存地址载入 A
12 :	RL	A	;左移 1 位
13 :	MOV	DPTR, #TABLE	;存表
14 :	MOVC	A, @A+DPTR	;到至 TABLE 取码 , 取 T 的值
15 :	MOV	TH0, A	;取到的高位字节存入 TH0
16 :	MOV	21H, A	;取到的高位字节存入 21H
17 :	MOV	A, 22H	;载入取码指针值
18 :	RL	A	;向左移 1 位
19 :	INC	A	;加 1
20 :	MOVC	A, @A+DPTR	;到表 TABLE 取低位字节计数值
21 :	MOV	TL0, A	;取到的低位字节存入 TL0
22 :	MOV	20H, A	;取低位字节存入 20H
23 : L2:	ACALL	KEY	;调用 KEY , 判断有无键按下
24 :	SETB	EA	;开通中断
25 :	JB	F0, L1	;是否有键按下
26 :	JMP	L2	;有则跳转至 L2
27 : KEY:	SETB	F0	;设 F0=1
28 :	MOV	R3, #0F7H	;扫描初值 (P3.3=0)
29 :	MOV	R1, #00H	;取码指针初值
30 : L3:	MOV	A, R3	;载入扫描指针
31:	MOV	P3, A	;输出至 P3 , 开始扫描
32:	MOV	A, P3	;读入 P3
33:	SETB	C	;令 C=1
34:	MOV	R5, #04H	;检测 P3.7 ~ P3.4
35: L4:	RLC	A	;左移 1 位
36:	JNC	KEYIN	;检测行 C=0 , 表示被按下
37:	INC	R1	;无键按下则取码指针加 1
38:	DJNZ	R5, L4	;4 列检测完毕?
39:	MOV	A, R3	;载入扫描指针
40:	SETB	C	;令 C=1
41:	RRC	A	;扫描下一行
42:	MOV	R3, A	;存回 R3 扫描指针寄存器
43:	JC	L3	;C=0 , 表示行扫描完毕
44:	RET		;子程序返回
45: KEYIN:	MOV	22H, R1	;取码指针存入地址 22H
46:	CLR	F0	;令 F0=0
47:	RET		;子程序返回
48: TIM0:	PUSH	ACC	;将 A 的值暂存于堆栈
49:	PUSH	PSW	;将 PSW 的值暂存于堆栈

```

50:      MOV      TL0, 20H      ;重设计数值
51:      MOV      TH0, 21H
52:      CPL      P2.4          ;将 P2.4 位反相
53:      POP      PSW          ;从堆栈取回 PSW 的值
54:      POP      ACC          ;从堆栈取回 A 的值
55:      RETI               ;返回到主程序
56: TABLE: DW    64021,64103,64260,64400    ;编码表
57:           DW    64524,64580,64684,64777
58:           DW    64820,64898,64968,65030
59:           DW    65058,65110,65157,65178
60:      END               ;程序结束

```

11.4.3 代码详解

1. 程序分析解释

01~04：设置程序开始地址和中断 T0 起始地址。

05~07：设置 T0 工作在模式 1 下；开通中断并启动 T0 工作。

08：调用扫描子程序 KEY 开始扫描，判断是否有键按下。

09~22：存表（13 行语句），取码（14 和 20 行语句），向定时器装初始值（第 15 和 21 行语句）。与此同时，还将取到的高位字节和低位字节分别存入地址 21H（第 16 行语句）和 20H 处（第 22 行语句），以便在中断子程序中对定时器重装初始值用。地址 22H 用于暂存取码指针。整个过程由于已将中断关闭，所以，定时器不能产生中断信号，输出端 P2.4 没有输出。

第 10 行语句是判断 F0 是 0 还是 1。F0 是状态字寄存器 PSW 的一个自由使用的标志位。程序规定 F0=1 表示没有键按下，F0=0 表示有键按下。

23~26：再调用扫描程序，确认是否有键按下。当有键按下时，由于此时已经开通了中断，定时器会产生中断信号，使程序进入到中断服务子程序运行，于是输出端 P2.4 便有输出。

27~44：扫描子程序。

45~47：有键按下后，将取码指针存入地址 22H，并令 F0=0，用来标志有键按下。

48~55：中断服务子程序。其中第 48~49 行为进栈保护现场，第 53~54 行为出栈恢复现场。50~51 行重装定时器初始值。第 52 行语句是输出语句。第 55 行语句是中断服务子程序返回主程序的指令。

56~59：编码表。表中的 16 个数是定时器发出 16 个音的定时器初始值。

60：程序结束。

2. 至本章用过的指令归类

- 数据传送类指令：MOV、MOVC、PUSH、POP。
- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、RLC、RRC、CLR、ANL、ORL、XRL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI、JZ。

- 位操作类指令：CPL、CLR、SETB、JB、JNB、JBC、JNC、JC。
- 伪指令：END、DB、ORG。

11.4.4 模拟仿真

1. 模拟仿真前注意事项

在第 30 行语句与 31 行语句之间，增加一条模拟按键按下的语句。

```
MOV A,#11100111B
```

在第 24 行语句与 25 行语句之间，增加一条模拟中断溢出的语句，使程序能进入到中断服务程序里运行。

```
SETB TF0
```

2. 模拟仿真中注意事项

注意观察程序运行路线。当第一次扫描检测到有键按下时，由于第 09 行语句将中断屏蔽，所以程序运行不能进入到中断服务子程序，输出端 P2.4 没有输出。

在第二次扫描再次确认有键按下时，由于第 24 行语句将中断开启，所以，定时器 T0 将产生中断信号，通过程序标号 TIM0 进入中断服务子程序，使 P2.4 端口产生输出。

(2) 去掉模拟按键和模拟中断溢出语句，观察程序运行情况，由于没有按键按下，所以 P2.4 没有输出。

11.4.5 实例测试

将写入程序的单片机插入实验板插座内，并检查实验板上蜂鸣器接口是否与程序中声音输出端口一致，当检查无误后接通电源，即可按图 11.6 所示进行音乐弹奏。

11.4.6 经验总结

电子琴程序从功能设计上可分两大部分。

- 利用单片机内定时器 T0 产生 16 个音符，16 个音符代码在表中的排列顺序与 16 个按键上所标出的音符是对应的。
- 通过对键盘不断进行扫描，当确定有按键操作后，将所按下的键转换成相对应的音符，并通过输出端口使蜂鸣器发出相对应的声音，声音的长短由按键的时间决定。



第 12 章 串行通信

一台计算机与其他计算机之间往往需要交换信息，这些信息交换称为通信。单片机也具有通信功能，本章将介绍单片机串行通信功能中应用的几个简单实例。

12.1 单片机串行通信功能

80C51 单片机的内部除了含有并行 I/O 接口外，还带有 1 个串行 I/O 接口，通过 P3 的两个引脚 P3.0 和 P3.1 与外部连接。其中，RXD (P3.0) 为串行接收端，TXD (P3.1) 为串行发送端，专门用于串行通信。

12.1.1 单片机串行通信的作用

单片机串行通信是指单片机与其他设备之间以及单片机与单片机之间的数据逐位地按顺序传送。串行传送的优点是传输线少、通信距离长，特别适用于控制系统以及远程通信。

此外，利用串行口特点可对单片机的输入/输出端口进行扩充，以获得更多的输入和输出端口。

12.1.2 串行通信中双方基本约定

在串行通信中，数据通常以字符或字节为单位组成字符帧进行传送。字符帧通过传输线由发送端一帧一帧地发送到接收端，接收端一帧一帧地接收。通信双方必须遵守以下两项基本约定。

1. 字符帧格式

字符帧格式，即字符的编码形式，通信双方必须具有相同的字符帧格式，否则不能进行通信，其规定的格式如图 12.1 所示。

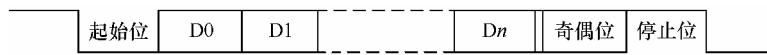


图 12.1 串行通信的字符帧格式

它用一个起始位表示字符的开始，用停止位表示字符的结束。数据位在起始位之后、停止位之前，这样构成一帧数据。

奇偶校验位位于数据位之后、停止位之前，用于表示串行通信中采用奇校验还是偶校验，由用户根据需要决定。

2. 数据传输率

数据传输率用于表示数据传输的速度，收发双方必须设置相同的数据传输率。

12.1.3 串行口的结构和通信过程

1. 串行口的结构

80C51 单片机的串行口主要由两个数据缓冲寄存器 SBUF、一个输入位移寄存器以及两个控制寄存器 SCON 和 PCON 组成，其结构如图 12.2 所示。

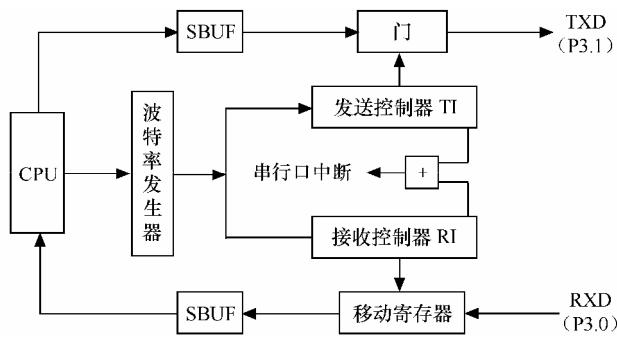


图 12.2 串行口结构框图

其中，缓冲寄存器 SBUF 是主体，它是专用寄存器：一个作发送缓冲器，一个作接收缓冲器。两个缓冲器共用一个地址 99H，由读写信号区分，CPU 写 SBUF 就是发送缓冲器的内容；读 SBUF 就是接收缓冲器的内容。

控制寄存器 SCON 和 PCON 用来设定串行口的工作方式并对接收和发送进行控制。

2. 串行通信过程

(1) 接收数据的过程。

在进行通信时，当 CPU 允许接收时，即 SCON 的 REN 位设置 1 时，外界数据通过引脚 RXD (P3.0) 串行输入，数据的最低位首先进入移位寄存器，一帧接收完毕后再并行送入接收数据缓冲寄存器 SBUF 中，同时将接收控制位即中断标志位 RI 置位，向 CPU 发出中断请求。

CPU 响应中断后读取输入的数据，同时用软件将 RI 位清除，准备开始下一帧的输入过程，直至所有数据接收完。

(2) 发送数据的过程。

CPU 要发送数据时，将数据并行写入发送数据缓冲寄存器 SBUF 中，同时启动数据由 TXD (P3.1) 引脚串行发送，当一帧数据发送完，即发送缓冲器空时，由硬件自动将发送中断标志位 TI 置位，向 CPU 发出中断请求。CPU 响应中断后用软件将 TI 位清除，同时又将下一帧数据写入 SBUF 中，重复上述过程直到所有数据发送完毕。

12.2 串行口的控制

串行口通信是由控制寄存器 PCON 和 PCON 控制的。

12.2.1 电源和数据传输率控制寄存器 PCON

PCON 是电源和数据传输率控制寄存器 , 其中第 7 位 SMOD 为串行口数据传输率系数控制位 , 当 SMOD=1 时 , 数据传输率加倍。

12.2.2 串行口控制寄存器 SCON

SCON 主要功能是设定串行口的工作方式、接收和发送控制以及设置状态标志 , 地址为 98H , 其格式及各位的功能如图 12.3 所示。



图 12.3 串行口控制寄存器 SCON 示意图

- RI : 接收中断标志。

在方式 0 中 , 第 8 位接收完毕 , RI 由硬件置位。在其他方式中 , 在接收停止位的一半时 , 自动设定为 1 。 RI 也需由软件清 0 。

- TI : 发送中断标志。

在方式 0 中 , 发送完第 8 位数据时 , 由硬件自动置位 (设定为 1) ; 在其他方式中 , 在发送停止位开始时由硬件自动置位。 TI=1 时 , 申请中断 , CPU 响应中断后 , 发送下一帧数据。在任何方式中 , TI 都必须由软件清 0 。

- RB8 : 在方式 2 、方式 3 中 , RB8 是接收的第 9 位数据 ; 在多机通信中为地址、数据标志位 ; 方式 0 中 RB8 未用 ; 在方式 1 中 , 若 SM2=0 , 则接收的停止位自动存入方式 RB8 中。

- TB8 : 在方式 2 、方式 3 中 , 此位会被当成第 9 位数据发送出去。根据需要用软件置位或清 0 。

- REN : 允许串行接收位。

REN=1 时 , 允许接收 ; REN=0 时 , 禁止接收。该位由软件置位或清 0 。

- SM2 : 允许方式 2 和方式 3 进行多机通信控制位。

若 SM2=1 接收到第 9 位数据(RB8)为 0 , 则接收中断不被激活。在方式 1 中 , 若 SM2=1 , 只有接收到有效的停止位 RI 才被激活。在方式 0 中 , SM2 必须是 0 。

- SM0 、 SM1 : 串行口工作方式选择位。两个选择位对应 4 种方式 , 如表 12.1 所示。

表 12.1 串行口工作方式

SM0	SM1	方式	功 能 说 明
0	0	0	移位寄存器方式 (用于扩展 I/O 口)
0	1	1	8 位 UART , 数据传输率可变 (T1 溢出率 /n)

续表

SM0	SM1	方式	功 能 说 明
1	0	2	9 位 UART , 数据传输率为 $f_{osc}/64$ 或 $f_{osc}/32$
1	1	3	9 位 UART , 数据传输率可变 (T1 溢出率/n)

注 : f_{osc} 为时钟频率。

12.2.3 串行口的 4 种工作方式

1. 方式 0

方式 0 又叫同步移位寄存器输出方式。在这种方式下 , 数据从 RXD (P3.0) 端串行输出或输入 , 同步信号从 TXD (P3.1) 端输出 , 其数据传输率是固定的 , 为 $f_{osc}/12$ 。该方式以 8 位数据为一帧 , 没有起始位和停止位 , 先发送或接收最低位。

2. 方式 1

方式 1 为 8 位异步通信接口 , 适合于点对点的异步通信。该方式规定发送或接收一帧信息为 10 位 , 即 1 个启始位、 8 个数据位和 1 个停止位 , 数据传输率可以改变。

3. 方式 2

方式 2 为 9 位异步通信接口 , 接收或发送一帧信息由 11 位数据组成 , 1 个启始位、 8 个数据位、 1 个可编程位和 1 个停止位。数据传输率固定 , 只有两种选择 , 即数据传输率为 $f_{osc}/64$ 或 $f_{osc}/32$ 。该方式适用于多机通信。

4. 方式 3

方式 3 与方式 2 完全类似 , 惟一的区别是方式 3 的数据传输率是可变的。每帧信息与方式 2 一样为 11 位组成 , 也适合于多机通信。

12.3 扩展 8 个输出端口

功能说明 : 将单片机串行口设定在方式 0 工作状态下 , 并将 RXD (P3.0) 和 TXD (P3.1) 接 74LS164 芯片 , 扩展成 8 个输出端口。使 8 个 LED 中每 4 个为一组 , 亮灯从中间开始向左移动一次 , 再从中间开始向右移动一次 , 接着从最右端向中间移动一次 , 再从最左端向中间移动一次 , 然后闪烁两次 , 不断循环。

12.3.1 硬件设计

1. 电路图

电路设计如图 12.4 所示。

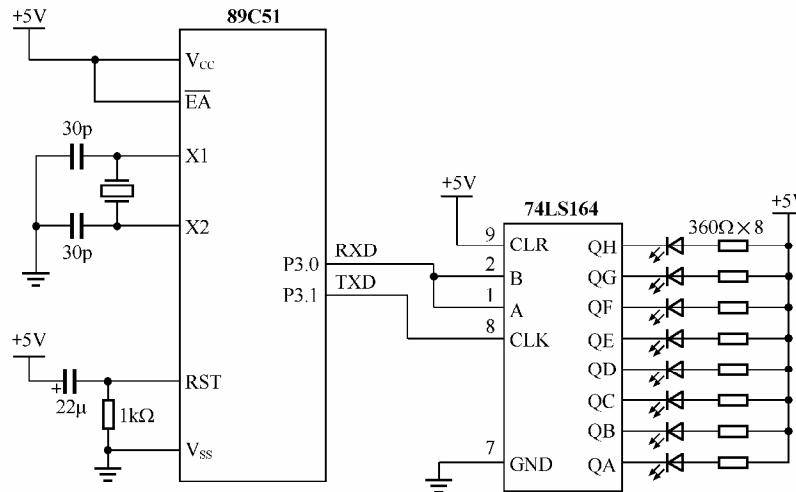


图 12.4 串行输出扩展电路

单片机内部的串行口在方式 0 工作状态下使用 74LS164 移位寄存器芯片扩展出 8 个输出口 QA ~ QH，接有 8 个 LED 输出显示。

芯片的串行输入端 A 和 B 与单片机的串行口输入端 RXD 相连。单片机的串行输出信号由 RXD 输入到芯片内。

注意 在方式 0 下，TXD (P3.0) 引脚既是输入端也是输出端。

单片机的 TXD (P3.1) 引脚与芯片的时钟脉冲输入端 CLK 相连，向芯片提供脉冲信号。芯片的第 9 脚接+5V，第 8 脚接地。

2. 74LS164 芯片

74LS164 芯片是串行输入并行输出的移位寄存器，其引脚如图 12.5 所示。

- Q0 ~ Q7：并行输出端。
- A、B：串行输入端。
- CLR：清除端，0 电平时使 74LS164 输出清 0。
- CLK：时钟脉冲输入端，在脉冲的上升沿实现位移。

12.3.2 程序设计

本程序利用单片机串行口功能将数据发送到 74LS164 芯片内，再由芯片来完成 8 个输出口扩展。

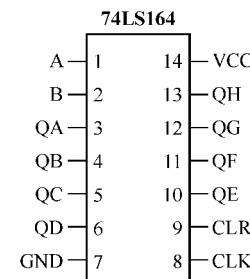


图 12.5 74LS164 引脚图

1. 流程图

主程序设计流程如图 12.6 所示。

2. 程序

汇编语言编写的扩展 8 个输出端口源程序 CX01.ASM 代码如下：

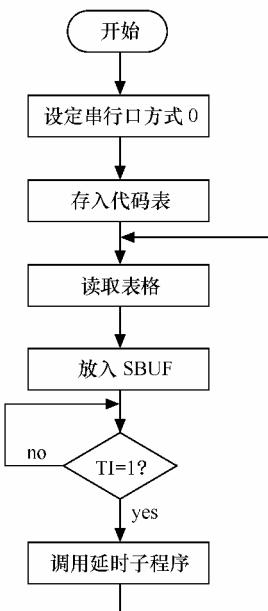


图 12.6 主程序流程

01:	MOV	SCON, #0000000B	;设串行口方式 0
02:	START:	MOV DPTR, #TABLE	;存入表起始地址
03:	LOOP:	CLR A	;清除 ACC
04:		MOVC A, @A+DPTR	;按地址取代码并存入 A
05:		CJNE A, #03H, A1	;是否 03H? 不是则跳转到 A1
06:		JMP START	;是, 则跳转到程序开始处
07:	A1:	MOV SBUF, A	;将 A 值存入 SBUF
08:	LOOP1:	JBC TI, LOOP2	;检测 TI=1? 是则跳转到 LOOP2
09:		JMP LOOP1	;不是, 则再检测
10:	LOOP2:	ACALL DELAY	;调延时子程序
11:		INC DPTR	;数据指针加 1
12:		JMP LOOP	;跳转到 LOOP 处
13:	DELAY:	MOV R5, #20	;延时 0.2s 子程序
14:	DLY1:	MOV R6, #20	
15:	DLY2:	MOV R7, #248	
16:		DJNZ R7, \$	
17:		DJNZ R6, DLY2	
18:		DJNZ R5, DLY1	
19:		RET	;延时子程序返回
20:	TABLE:	DB 0EFH,0DFH, 0BFH, 7FH	;中间开始向左移动控制码
21:		DB 0F7H,0FBH, 0FDH,0FEH	;中间开始向右移动控制码
22:		DB 0FEH,0FDH, 0FBH,0F7H	;最右端向中间移动控制码
23:		DB 7FH, 0BFH, 0DFH,0EFH	;最左端向中间移动控制码
24:		DB 00H, 0FFH, 00H, 0FFH	;闪烁 2 次
25:		DB 03H	;结束码

26:	END	;程序结束
-----	-----	-------

12.3.3 代码详解

1. 标号说明

START：从头开始取表循环的进入点。

LOOP：依次取表代码的进入点。

LOOP1：判断等待进入点。

LOOP2：调用延时子程序。

DELAY：延时子程序的进入点。

TABLE：表的进入点。

2. 寄存器使用分配情况

程序中 R5、R6、R7、A、DPTR、SBUF 和 SCON 为寄存器。其中 R5、R6、R7 为普通寄存器，在延时子程序中作为计数器用；A 是特殊寄存器，又称累加器；DPTR 为数据指针。

SBUF 和 SCON 是新用到的寄存器。

SBUF 是串行数据缓冲寄存器，用于存放欲发送或接收的数据，是串行口主要组成部分。

SCON 是串行口控制寄存器，用来设定串行口的工作方式、设置状态标志以及对接收发送的控制。

3. 程序分析解释

本程序利用单片机通信功能对串行输出端口进行扩充。实现的办法是先将数据串行发送到 74LS164 芯片内，再由芯片来扩展成 8 个输出口，重点需要理解串行口发送语句。

01：该语句设定串行口工作在方式 0。SCONS 是串行口控制寄存器，最后两位 SM0 和 SM1 是串行口工作方式选择位，两个位都为 0 时，为工作方式 0。方式 0 适合对 I/O 口的扩展。

07：串行口数据发送语句。即将 A 中的数据写入发送缓冲器 SBUF 中，同时启动数据由 RXD (P3.0) 引脚串行发送到芯片的输入端。在方式 0 状态时，发送和接收都用 RXD 端口，当一帧数据发送完即发送缓冲器空时，由硬件自动将发送中断标志位 TI 置位。

08：该语句是判断语句。TI 是控制寄存器 SCON 中的中断标志位，判断 TI 位是 0 还是 1。TI=0 说明没发送，继续检测、等待；TI=1 说明一帧数据已发送，调用延时子程序延时一段时间再发送下一帧，同时，JBC 指令还将 TI 位清 0，为发送下一帧做准备。

20 ~ 25：控制码编制表，表名为 TABLE，一共有 8 个码，对应 8 个 LED。码的编排位置确定了亮灯的移动方向。

12.3.4 模拟仿真

1. 模拟仿真前注意事项

(1) 07 与 08 语句行之间，加一句“SETB TI”，该语句使中断标志位 TI=1，模拟产生

中断，使程序能向下运行。

(2) 在第 10 语句的程序标号后边加分号 (LOOP2: ; ACALL DELAY)，使调用延时语句不起作用，以便缩短模拟仿真时间。

2. 模拟仿真中注意事项

(1) 串行口控制寄存器 SCON 值的变化：该值为 00 时，说明串行口设定为方式 0 工作状态，中断标志位 TI=0，等待输出；该值为 02 时中断标志位 TI=1，此时向 CPU 提出中断请求，由 SBUF 发送数据。

(2) 串行口缓冲寄存器 SBUF 值的变化：缓冲寄存器 SBUF 的值是按表中的编码次序发生变化，开始运行 04 行语句时将表中第一位存放的码 FE 放入累加器 A 中，运行 07 行语句时再将 A 中的数据写入缓冲寄存器 SBUF 中并发送出去。SBUF 中的值就是表中的码，每个码为一帧，写入 SBUF 中一一发出。

12.3.5 实例测试

一般购买的实验板不带扩展电路部分，需要自己按电路图组装，并与单片机相连。确认连线无误后接通电源，会看到亮灯先从中间开始向左移动一次，再从中间开始向右移动一次，接着从最右端向中间移动；再从最左端向中间移动，然后闪烁 2 次，照此不断循环。

12.3.6 经验总结

利用单片机通信功能可以做串行输出端口扩展。扩展时需要使用移位寄存器芯片，如 74LS164 就是常用的一种输出端口扩展的芯片，用此芯片可扩展出更多输出端口。

采用串行口扩展可以节省单片机的 I/O 口，但传送速度较慢，扩展的芯片越多，速度越慢。

12.4 扩展 8 个输入端口

功能说明：利用一个并入串出的移位寄存器 IC74166 芯片与单片机串行口相连，扩展成 8 个输入端口。IC74166 芯片连接 8 位拨动开关，作为单片机的数据输入端，控制单片机输出端口 P1 所接的 8 个 LED。

12.4.1 硬件设计

1. 电路图

电路设计如图 12.7 所示。

单片机的 TXD (P3.1) 作为移位脉冲输出端与 74LS166 的移位脉冲输入端 CLK 相连，RXD (P3.0) 作为串行输入端与 74LS166 的串行输出端 QH 相连，P3.2 用来控制 74LS166 的位移与置入 (SH/LD 脚)，时钟禁止端 INH 接地，芯片的 VCC 端接 +5V。

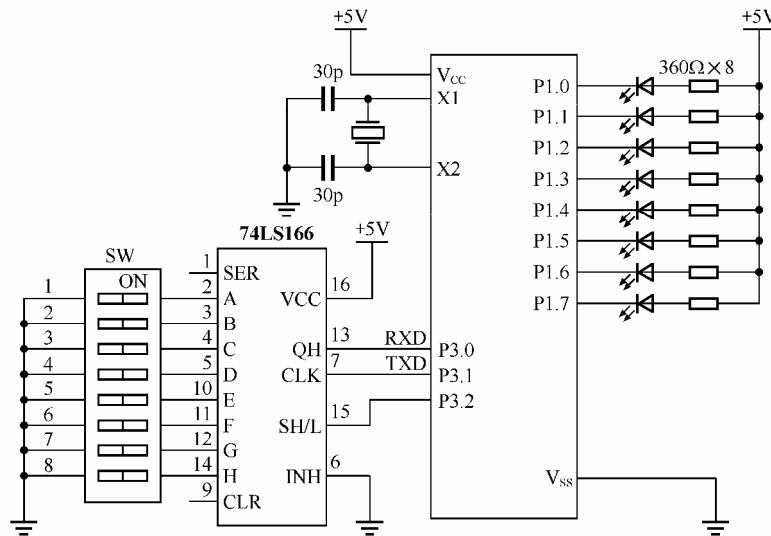


图 12.7 串行输入口扩展电路

2. 74LS166 芯片

74LS166 是串行输入并行输出 8 位移位寄存器，其引脚如图 12.8 所示。

- A ~ H：并行输入端。
- QH：串行输出端。
- CLK：时钟脉冲输入端在脉冲的上升沿实现位移。
- INH：时钟禁止端。
- SH/LD：位移与置位控制。
- SER：扩展多个 74LS166 的首尾连接端。
- VCC：接+5V。
- GND：接地端。

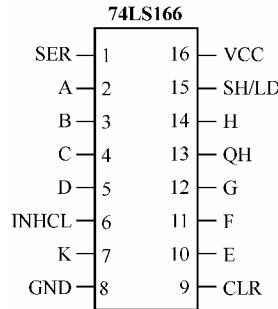


图 12.8 74LS166 引脚图

12.4.2 程序设计

1. 流程图

程序设计流程如图 12.9 所示。

2. 程序

汇编语言编写的扩展 8 个输入端口源程序 CX02.ASM 代码如下：

```

01: START: MOV      SCON, #00010000B ; 设定 MODE0,REN=1
02:     CLR      P3.2          ; P3.2 = 0, 载入 74166
03:     ACALL   DELAY         ; 调用延时
04:     SETB    P3.2          ; P3.2=1, 74166 移位串出
05:     CLR      RI            ; RI=0, 清除接收中断标志

```

```

06 :      JNB      RI, $          ;动态停机，等待接收完毕
07 :      MOV      A, SBUF        ;将 SBUF 数据装入 A
08 :      MOV      P1, A          ;将 A 中数据送入 P1 口输出
09 :      JMP      START         ;程序循环执行
10 :  DELAY: MOV      R7, #02      ;延时子程序
11 :      DJNZ     R7,$
12 :      RET
13 :      END                ;程序结束

```

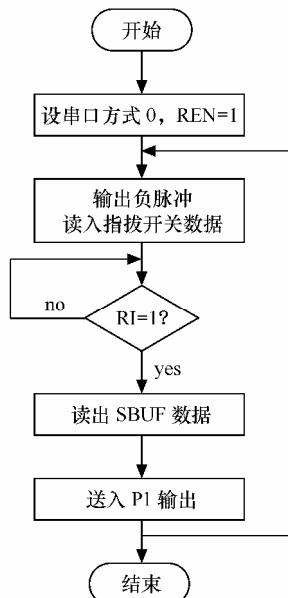


图 12.9 程序流程

12.4.3 代码详解

1. 标号说明

START：起始程序的进入点。

DELAY：延时子程序的进入点。

2. 寄存器使用分配情况

程序中 R7、A、P1、P3、SBUF 和 SCON 为寄存器。

3. 程序分析解释

本程序利用串行口通信功能，对串行输入端口进行扩充，需要重点理解串行口接收相关语句。

01：该语句设定串行口工作在方式 0 下，同时设置 SCON 中的允许串行接收位 REN 为 1。

02：允许指拨开关信号输入 P3.2 控制着芯片 74LS166 的位移与置入口 SH/LD 脚 P3.2=0 时载入 74LS166 数据。

- 03：调用延时子程序延时一段时间，防止误操作。
- 04：使 P3.2=1，74LS166 移位串出，即串行口开始接收芯片输入的数据。
- 05：清除串行口接收中断标志。
- 06：动态停机，等待接收完毕。
- 07：读出缓冲寄存器 SBUF 中的数据，“MOV A, SBUF”是串行口接收的重要语句。
- 08：将累加器 A 中的数据送入 P1 口输出。
- 09：程序循环执行。

12.4.4 模拟仿真

1. 模拟仿真前注意事项

由于该程序信号源是外部的指拨开关输入，所以在模拟仿真前需要增加模拟指拨开关输入的语句才能进入模拟仿真，即在 05 行和 06 行之间增加 3 条语句。

- 增加语句“MOV A,#11111110B”，该语句模拟指拨开关中第一位开通。
- 增加语句“MOV SBUF,A”，该语句是将 A 中的数据写入缓冲寄存器 SBUF，是串行口发送语句。
- 增加语句“SETB RI”，该语句使中断标志位 RI=1，表明已经发完数据。

2. 模拟仿真中注意事项

该程序语句不多，仿真时主要观察串行口缓冲寄存器 SBUF 和控制寄存器 SCON 值的变化。

12.4.5 实例测试

一般购买的实验板没有扩展电路部分，所以需要自己先按电路图组装扩展电路部分，再与单片机相连。确认连线无误后接通电源，并拨动指拨开关，会看到相应的发光二极管点亮与熄灭。

本节 8 个指拨开关是通过一个串行口输入信号，同样达到对输出端口的控制效果，但节省了输入端口资源。

12.4.6 经验总结

利用单片机通信功能可以对串行输入端口进行扩展。扩展时需要使用移位寄存器芯片，如 74LS166 或 74LS165 等，此类芯片是并入串出，可扩展更多输入端口。

串行口做串行输入端口扩展时，一般需要进行如下工作。

- (1) 设定工作方式 0，并将允许串行接收位置 1，即 REN=1，允许接收。
- (2) 将数据从 SBUF 中读出，其语句为“MOV A, SBUF”。
- (3) 数据发送后，将中断标志位清 0。

12.5 向计算机发送一封信

功能说明：单片机与计算机通信，将单片机内一段文字“祝贺您：迈入单片机世界！OK！”

发送到计算机中并在计算机屏幕上显示。

12.5.1 硬件设计

电路设计如图 12.10 所示。

计算机一般具有两个串行口 COM1 和 COM2，其电器特性符合国际 EIA-RS232C 标准。单片机的实验板上也必须装有与其标准相符合的插座，即 RS232 插座以及相应的电器转换电路，如图 12.10 所示。

实验板上有 RS232 接口跳转线，在进行串口通信实验时按说明书进行设置。

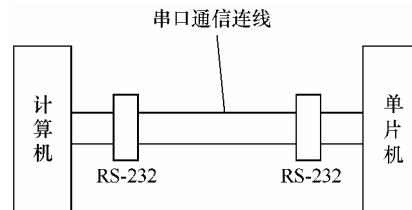


图 12.10 单片机与计算机通信示意图

12.5.2 程序设计

1. 流程图

程序设计流程如图 12.11 所示。

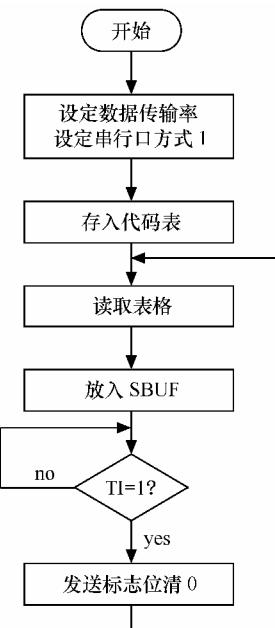


图 12.11 程序流程

2. 程序

汇编语言编写的向计算机发送一封信源程序 CX03.ASM 代码如下：

01 :	MOV	SCON, #40H	; 设串口方式 1、REN=0
02 :	MOV	PCON, #80H	; SMOD=1，数据传输率提高 1 倍
03 :	MOV	TMOD, #20H	; 设置定时器 1 工作于方式 2
04 :	MOV	TH1, #0FAH	; 定时器 1 装入计数初值

05 :	SETB	TR1	;启动定时器 1
06 :	MOV	DPTR, # TABLE	;存入表起始地址
07 : LOOP:	CLR	A	;清 0
08 :	MOVC	A, @A+DPTR	;按地址取代码并存入 A
09 :	CJNE	A, #01H, LOOP1	;比较不等转移
10 :	RET		;子程序返回
11 : LOOP1:	MOV	SBUF, A	;发送取码数据
12 :	JNB	TI, \$;等待发送完
13 :	CLR	TI	;将发送中断标志位清 0
14 :	INC	DPTR	;数据指针加 1
15 :	AJMP	LOOP	;跳转到 LOOP 处
16 : TABLE:	DB	0DH, 0AH, "祝贺你：迈入单片机世界！OK!", 0DH, 0AH, 01H	
17 :	END		;程序结束

12.5.3 代码详解

1. 标号说明

LOOP : 取表码的进入点。

LOOP1 : 发送数据的进入点。

TABLE : 代码表的进入点。

2. 寄存器使用分配情况

程序中 SCON、PCON、SBUF、TMOD、TH1、DPTR 和 A 为寄存器，TR1 和 TI 是相关寄存器中的控制、标志位。

SCON、PCON 和 SBUF 为串行口通信使用的寄存器。其中 SBUF 用于存放欲发送或接收的数据，SCON 用于设定串行口的工作方式的，PCON 用于控制数据传输率。

TH1 为定时器 1，TMOD 用来控制定时器工作方式的。

DPTR 为数据指针，它与累加器 A 配合使用可完成编码表的代码存取。

TR1 是定时器控制寄存器 TCON 中的定时/计数 T1 的运行控制位。

TI 是串行口控制寄存器 SCON 中的发送中断标志位。

3. 程序分析解释

本程序的功能是将单片机中的数据发送到计算机中。由于属于点对点的异步通信，所以将串行口的工作状态设置为方式 1。串行口在方式 1 下工作时数据传输率是由定时器 1 决定的，定时器 1 选择了方式 2 的工作方式，可自动加载初始值。

被发送的数据放在 TABLE 表中，通过查表的方式取送。

01 ~ 05 : 对串行口和定时器进行设置，设置串行口的工作方式为方式 1；定时器 1 的工作方式为方式 2，并启动定时器工作。

06 ~ 10 : 存表、读取表。

11 ~ 15 : 向计算机发送数据。

16 : TABLE 为代码表 , 待发送的数据存储在表中

4 . 至本章用过的指令归类

- 数据传送类指令 : MOV、MOVC、PUSH、POP。
- 算术运算类指令 : INC、DIV。
- 逻辑运算及位移类指令 : RL、RR、RLC、RRC、CLR、ANL、ORL、XRL。
- 控制转移类指令 : JMP、DJNZ、ACALL、RET、CJNE、RETI、JZ。
- 位操作类指令 : CPL、CLR、SETB、JB、JNB、JBC、JNC、JC。
- 伪指令 : END、DB、ORG。

12.5.4 模拟仿真

1 . 模拟仿真前

在第 1 行语句和 12 行语句之间增加一条 “ SETB TI ” 语句 , 模拟硬件将 TI 置位 , 使程序能向下运行。

2 . 模拟仿真中注意观察事项

仿真时注意观察各寄存器值的变化 , 其中串行口缓冲寄存器 SBUF 的数据就是向计算机发送的数据。

12.5.5 实例测试

用串行口通信连线将单片机实验板与计算机连接起来 , 并按说明书设置 RS232 跳转线。

1 . 运行串行口调试工具软件如 SS COM 等进行测试 , 也可以使用 Windows 操作系统中附件里的 “ 超级终端 ” 进行测试 , 选择开始 / 程序 / 附件 / 通信中的 “ 超级终端 ” 并设置串口号为 COM2 , 数据传输率为 9600 。

2 . 将写入串行口通信程序的单片机安装在实验板上 , 接通电源后 , 单片机开始向计算机发送数据并在超级终端的窗口中显示 , 如图 12.12 所示。

12.5.6 经验总结

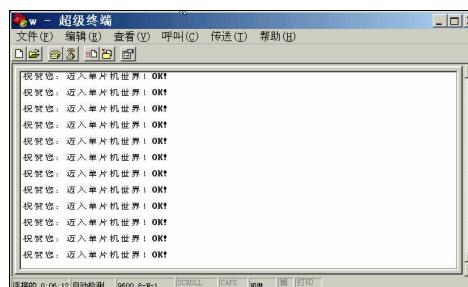


图 12.12 串行口通信调试窗口

串行口有 4 种工作方式 , 即方式 0~3 , 可根据设计任务要求进行选择。例如 , 利用串行口做串行输入端口扩充或输出端口扩充时 , 需要将串行口设置为方式 0 的工作状态 ; 如若做异步通信 , 需要将串行口设置为方式 1 的工作状态。



第 13 章 LCD 模块及其应用

LCD 为液晶显示器，应用非常广泛。

13.1 LCD 模块

13.1.1 LCD 的分类

LCD 可分为两种类型，一种是字符模式 LCD，另一种是图形模式 LCD。其中，字符模式 LCD 是点阵型液晶显示器，专门用来显示字母、数字、符号等。

由于 LCD 的控制需专用的驱动电路，一般不会单独使用，而是将 LCD 面板、驱动与控制电路组合成 LCD 模块（Liquid Crystal display Moulde，简称为 LCM）一起使用。

目前，常用的有 16 字×1 行、16 字×2 行、20 字×2 行、40 字×2 行等字符模块。这些 LCM 虽然显示字数不同，但都有相同的输入/输出界面。

13.1.2 LCD 模块的引脚

下面介绍常用的 20 字×2 行（简称 20×2）字符模块，外形如图 13.1 所示，引脚如图 13.2 所示。



图 13.1 LCM 模块

20×2 LCD 每行可以显示 20 个字，可显示的行数为 2 行，有 16 只引脚，其中数据线 DB0 ~ DB7 与控制信号线 RS、R/W、E 用来与单片机连接，另外 3 只引脚为电源信号线 VSS、VDD、V0，其各脚功能如表 13.1 所示。

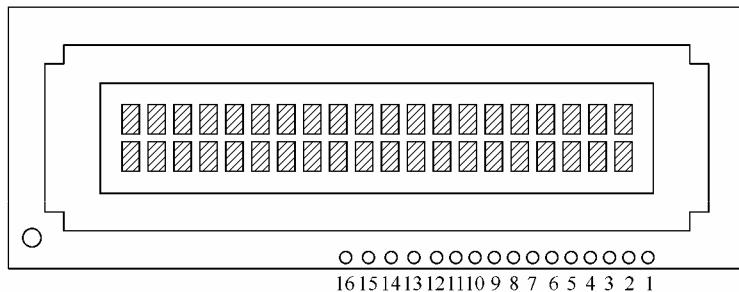


图 13.2 LCD 引脚图

表 13.1

LCD 引脚功能

引脚	符 号	功 能 说 明
1	VSS	接地
2	VDD	+5V
3	V0	显示屏明亮度调整脚，一般将此脚接地
4	RS	寄存器选择 0：指令寄存器（WRITE） Busy flag，位置计数器（READ） 1：数据寄存器（WRITE，READ）
5	R/W	READ/WRITE 选择 1：READ 0：WRITE
6	E	读写使能，下降沿使能
7	DB0	低 4 位三态、双向数据总线
8	DB1	
9	DB2	
10	DB3	
11	DB4	高 4 位三态、双向数据总线 DB7 也是一个 Busy flag
12	DB5	
13	DB6	
14	DB7	
15	BLA	背光源正极
16	BLK	背光源负极

13.1.3 寄存器选择及显示器地址

1. 寄存器选择

LCD 内部有两个寄存器，一个是指令寄存器 IR，另一个是数据寄存器 DR。IR 用来存放由微控制器所送来的指令代码，如光标归位、清除显示等；DR 用来存放欲显示的数据。

显示的次序是先把欲存放数据地址写入 IR，再把欲显示的数据写入 DR，DR 就会自动把数据送至相应的 DD RAM 或 CG RAM 地址，DD RAM 是显示数据的存储器，用来存放 LCD

的显示数据；CG RAM 是字符产生器，用来存放自己设计的 5×7 点图形的显示数据。

LCD 指令寄存器和数据寄存器的选择如表 13.2 所示，通常 R/W 与 RS 信号线一起使用。

表 13.2 LCD 寄存器的选择

E	R/W	RS	功 能 说 明
1	0	0	写入命令寄存器
1	0	1	写入数据寄存器
1	1	0	读取忙碌标志及 RAM 地址
1	1	1	读取 RAM 数据
0	X	X	不动作

当 RS=0 时，选择指令寄存器；RS=1 时，选择数据寄存器。

当 R/W=0 时，数据写入 LCD 控制器；当 R/W=1，到 LCD 控制器读取数据。

E：高电位使能信号线。

2. 显示器地址

20×2(20字2行)显示器地址如表13.3所示。

表 13.3 20×2 LCD 显示器地址

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3

13.1.4 LCM 控制指令

LCM 提供了 11 项指令，如表 13.4 所示，这些指令会在编程使用中逐渐加深理解。

表 13.4 LCD 指令表

续表

序号	指令功能	控制线		数据线																				
		RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0													
5	移位方式	0	0	0	0	0	1	S/C	R/L	×	×													
		S/C=0、R/L=0：光标左移；S/C=0、R/L=1：光标右移 S/C=1、R/L=0：字符和光标左移；S/C=1、R/L=1：字符和光标右移																						
6	功能设定	0	0	0	0	1	DL	N	F	×	×													
		DL=1：数据长度为 8 位，DL=0：数据长度为 4 位 N=1：双列字，N=0：单列字；F=1:5×10 字形，F=0:5×7 字形																						
7	CG RAM 地址设定	0	0	0	1	CG RAM 地址																		
		将所要操作的 CG RAM 地址放入地址计数器																						
8	DD RAM 地址设定	0	0	1	DD RAM 地址																			
		将所要操作的 DD RAM 地址放入地址计数器																						
9	忙碌标志位 BF	0	1	BF	地址计数器内容																			
		读取地址计数器，并查询 LCM 是否忙碌 BF=1 表示 LCM 忙碌，BF=0 表示 LCM 可接受指令或数据																						
10	写入数据	1	0	写入数据																				
		将数据写入 CG RAM 或 DD RAM																						
11	读取数据	1	1	读取数据																				
		读取 CG RAM 或 DD RAM 的数据																						

13.2 一个简单的液晶显示程序

功能说明：本节将进行简单的 LCD 显示实验。单片机接有 LCD 液晶显示器，开机后在液晶显示屏第一行显示“OK”。

13.2.1 硬件设计

LCD 模块与单片机连接电路非常简单，如图 13.3 所示。

单片机 P1.0 ~ P1.7 分别与 LCD 模块的 DB0 ~ DB7 数据线连接，P3.5 ~ P3.7 接至 LCD 模块控制信号引脚 RS、R/W 和 E，LCD 模块的 VDD 引脚接电源+5V，VSS 和 V0 引脚接地。

13.2.2 程序设计

本程序是由主程序、初始化子程序、写指令子程序、判断 LCM 是否忙碌子程序和写数据子程序组成。

1. 流程图

程序设计流程如图 13.4 所示。

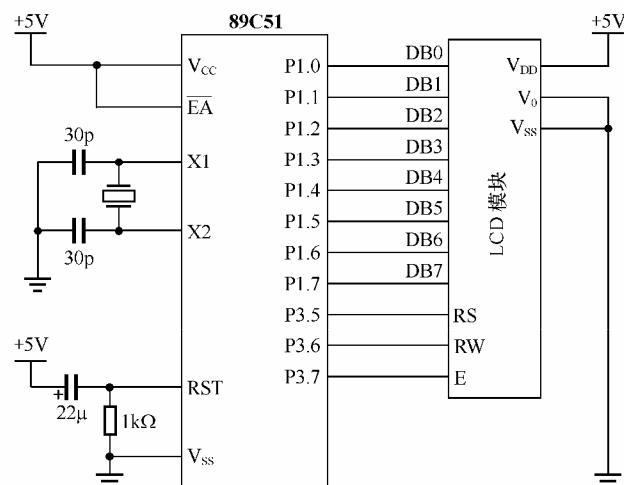


图 13.3 LCD 与单片机连接电路图

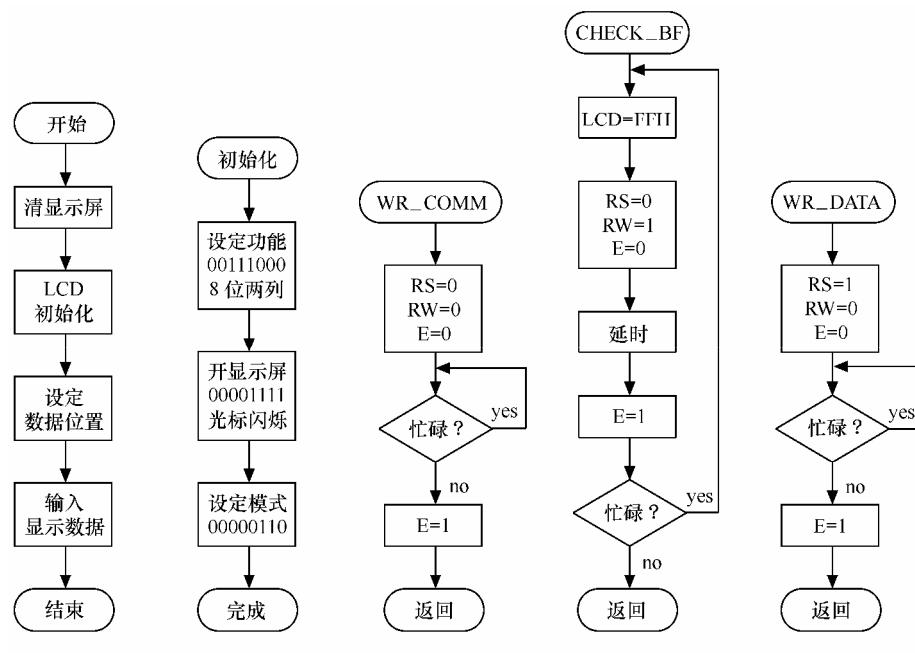


图 13.4 流程图

2. 程序

汇编语言编写的一个简单的液晶显示程序源程序 LCD1301.ASM 代码如下：

01	RS	bit	P3.5
02	RW	bit	P3.6
03	E	bit	P3.7
04	LCD	EQU	P1

```

05 ;主程序
06 MAIN:
07 MOV      LCD, #00000001B          ;清屏并复位光标
08 ACALL   WR_COMM                ;调用写指令子程序
09 ACALL   INIT_LCD              ;调用初始化子程序
10 MOV      LCD, #82H                ;写入显示起始地址
11 ACALL   WR_COMM                ;调用写指令子程序
12 MOV      LCD, #'O'               ;显示"O"
13 ACALL   WR_DATA                ;调用写数据子程序
14 MOV      LCD, #'K'               ;显示"K"
15 ACALL   WR_DATA                ;调用写数据子程序
16 JMP $                 ;维持当前输出状态

17 ;LCM 初始化子程序
18 INIT_LCD:
19 MOV      LCD, #00111000B          ;设置 8 位、2 行、5×7 点阵
20 ACALL   WR_COMM                ;调用写指令子程序
21 MOV      LCD, #00001111B          ;显示器开，光标允许闪烁
22 ACALL   WR_COMM                ;调用写指令子程序
23 MOV      LCD, #00000110B          ;文字不动，光标自动右移
24 ACALL   WR_COMM                ;调用写指令子程序
25 RET                  ;子程序返回

26 ;写指令子程序
27 WR_COMM:
28 CLR      RS                   ;RS=0，选择指令寄存器
29 CLR      RW                   ;RW=0，选择写模式
30 CLR      E                    ;E=0，禁止读/写 LCM
31 ACALL   CHECK_BF              ;调用判断 LCM 忙碌子程序
32 SETB    E                   ;E=1，允许读/写 LCM
33 RET                  ;子程序返回

34 ;判断忙碌子程序
35 CHECK_BF:
36 MOV      LCD, #0FFH              ;此时不接收外来指令
37 CLR      RS                   ;RS=0，选择指令寄存器
38 SETB    RW                   ;RW=1，选择读模式
39 CLR      E                    ;E=0，禁止读/写 LCM
40 NOP                  ;延时 1μs
41 SETB    E                   ;E=1，允许读/写 LCM
42 JB      LCD.7, CHECK_BF        ;忙碌循环等待
43 RET                  ;子程序返回

44 ;写数据子程序
45 WR_DATA:
46 SETB    RS                   ;RS=1，选择数据寄存器
47 CLR      RW                   ;RW=0，选择写模式

```

48	CLR	E	; E=0 , 禁止读/写 LCM
49	ACALL	CHECK_BF	; 调用判断忙碌子程序
50	SETB	E	; E=1 , 允许读/写 LCM
51	RET		; 子程序返回
52			
53	END		; 程序结束

13.2.3 代码详解

1. 主要标号说明

MAIN : 主程序。

INIT_LCD : LCM 初始化设定。

WR_COMM : 写指令子程序。

CHECK_BF : 判断是否忙碌子程序。

WR_DATA : 写数据子程序。

2. 程序分析解释

01 : 定义 LCM 的 RS 引脚由单片机的 P3.5 引脚控制。

02 : 定义 LCM 的 RW 引脚由单片机的 P3.6 引脚控制。

03 : 定义 LCM 的 E 引脚由单片机的 P3.7 引脚控制。

04 : 定义 LCM 的数据口 DB0 ~ DB7 引脚由单片机的 P1 引脚控制。

06 : 主程序。

07 : 清除显示屏，并使复位光标。

08 : 调用 WR_COMM 写指令子程序。

09 : 调用 INIT_LCD 初始化子程序。

10 : 写入显示起始地址为 82H , 即第 1 行第 3 个位置。

11 : 调用 WR_COMM 写指令子程序。

12 : 输入欲显示的数据 “OK”的第一个字母 “O”。

13 : 调用 WR_DATA 写数据子程序。

14 : 输入欲显示的数据 “OK”的第二个字母 “K”。

15 : 调用 WR_DATA 写数据子程序。

16 : 动态停机 , 维持当前输出状态。

18 : LCM 初始化设定子程序。

19 : 设定 8 位数据传送两行显示、5×7 点阵字型。

20 : 调用 WR_COMM 写指令子程序。

21 : 00001111B 指令使显示屏开启 , 显示光标 , 并允许光标闪烁。

22 : 调用 WR_COMM 写指令子程序。

23 : 00000110B 指令使光标自动右移 , 文字不动。

24 : 调用 WR_COMM 写指令子程序。

25 : LCM 初始化，设定子程序返回。
27 : 写指令子程序。
28 : 设置 RS=0 , RS 为低电平时选择指令寄存器。
29 : 设置 RW=0 , RW 为低电平时选择写模式。
30 : 设置 E=0 , E 为低电平时禁止读/写 LCM。
31 : 调用 CHECK_BF , 判断 LCM 是否忙碌子程序。
32 : 设置 E=1 , E 为高电平时允许读/写 LCM。
33 : 写指令子程序返回。
35 : CHECK_BF 为判断是否忙碌子程序。
36 : 此时不接收外来指令。
37 : 设置 RS=0 , RS 为低电平时选择指令寄存器。
38 : 设置 RW=1 , RW 为高电平时选择写读模式。
39 : 设置 E=0 , E 为低电平时禁止读/写 LCM。
40 : 延时 1 μ s。
41 : 设置 E=1 , E 为高电平时允许读/写 LCM。
42 : LCD.7 为 LCM 的第 7 位 DB7 , 该位忙碌标志信号 BF 为高电平时，说明 LCM 正忙，跳转回到 CHECK_BF 处继续查询；如果 BF 为低电平时，程序向下执行。
43 : 判断是否忙碌子程序返回。
45 : WR_DATA 为写数据子程序。
46 : 设置 RS=1 , RS 为高电平时选择数据寄存器。
47 : 设置 RW=0 , RW 为低电平时选择写模式。
48 : 设置 E=0 , E 为低电平时禁止读/写 LCM。
49 : 调用 CHECK_BF 判断忙碌子程序。
50 : 设置 E=1 , E 为高电平时允许读/写 LCM。
51 : 写数据子程序返回。
53 : 程序结束。

13.2.4 实例测试

测试该程序时，要首先检查实验板上安装的液晶显示屏端口是否与程序中的输出端口一致，如果不一致需要按照实验板实际电路更改程序输出端口，程序其他部分不变。

如果实验板上不带液晶显示屏，需要自行按图 13.3 与单片机连接。LCD 面板和驱动电路是组合在一起的，所以安装时无需考虑驱动器问题，线路连接也很简单。

当实验板连接线路检查好后接通电源，会看到在液晶显示屏第 1 行第 3、4 位置上显示“OK”，且右侧光标闪烁。

13.2.5 经验总结

LCM 在使用时是先把欲存放数据的地址写入 IR，再把欲显示的数据写入 DR，在写入前还需要先检查忙碌标志 BF，只有当 BF=0 时才可写入。因此，在程序里输入数据过程中会反复出现调用写指令子程序、判断 LCM 是否忙碌子程序及写数据子程序。

13.3 使 LCD 显示两行字符

功能说明：单片机接有 LCD 液晶显示器，开机后在液晶显示屏第 1 行第 2 个位置上显示“A”，在第 2 行第 7 个位置上显示“B”。本节电路设计上与上节相同。

13.3.1 程序设计

本节程序由主程序、初始化子程序、写指令子程序、判断 LCM 是否忙碌子程序和写数据子程序所构成，在设定第 1 行显示数据位置显示“A”后，又设定了第 2 行显示数据的位置并输出显示数据“B”。

1. 流程图

主程序设计流程如图 13.5 所示。

2. 程序

汇编语言编写的使 LCD 显示两行字符源程序 LCD1302.ASM 代码如下：

```

01      RS      bit      P3.5
02      RW      bit      P3.6
03      E       bit      P3.7
04      LCD     EQU      P1
05;主程序
06 MAIN:
07      MOV      LCD, #00000001B      ;清屏并复位光标
08      ACALL   WR_COMM            ;调用写指令子程序
09      ACALL   INIT_LCD          ;调用初始化子程序
10
11      MOV      LCD, #81H        ;写入第 1 行第 2 个位置
12      ACALL   WR_COMM            ;调用写指令子程序
13      MOV      LCD, #'A'        ;显示"A"
14      ACALL   WR_DATA           ;调用写数据子程序
15
16      MOV      LCD, #0c6H        ;写入第 2 行第 7 个位置
17      ACALL   WR_COMM            ;调用写指令子程序
18      MOV      LCD, #'B'        ;显示"B"
19      ACALL   WR_DATA           ;调用写数据子程序
20      JMP      $                  ;维持当前输出状态
21 LCM 初始化子程序

```

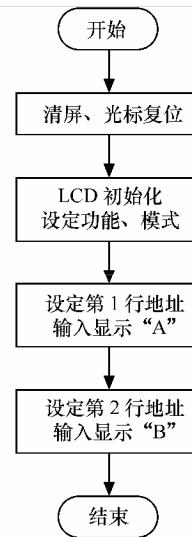


图 13.5 主程序流程

```

22 INIT_LCD:
23   MOV      LCD, #00111000B      ;设置8位、2行、5×7点阵
24   ACALL   WR_COMM             ;调用写指令子程序
25   MOV      LCD, #00000111B      ;显示器开，光标允许闪烁
26   ACALL   WR_COMM             ;调用写指令子程序
27   MOV      LCD, #00000110B      ;文字不动，光标自动右移
28   ACALL   WR_COMM             ;调用写指令子程序
29   RET                 ;子程序返回
30 ;写指令子程序
31 WR_COMM:
32   CLR      RS                ;RS=0，选择指令寄存器
33   CLR      RW                ;RW=0，选择写模式
34   CLR      E                 ;E=0，禁止读/写LCM
35   ACALL   CHECK_BF           ;调用判断LCM忙碌子程序
36   SETB    E                 ;E=1，允许读/写LCM
37   RET                 ;子程序返回
38 ;判断是否忙碌子程序
39 CHECK_BF:
40   MOV      LCD, #0FFH          ;此时不接受外来指令
41   CLR      RS                ;RS=0，选择指令寄存器
42   SETB    RW                ;RW=1，选择读模式
43   CLR      E                 ;E=0，禁止读/写LCM
44   NOP                 ;延时1μs
45   SETB    E                 ;E=1，允许读/写LCM
46   JB      LCD.7, CHECK_BF    ;忙碌循环等待
47   RET                 ;子程序返回
48 ;写数据子程序
49 WR_DATA:
50   SETB    RS                ;RS=1，选择数据寄存器
51   CLR      RW                ;RW=0，选择写模式
52   CLR      E                 ;E=0，禁止读/写LCM
53   ACALL   CHECK_BF           ;调用判断忙碌子程序
54   SETB    E                 ;E=1，允许读/写LCM
55   RET                 ;子程序返回
56
57 END                  ;程序结束

```

13.3.2 代码詳解

1. 主要标号说明

MAIN：主程序。

INIT_LCD：LCM 初始化设定。

WR_COMM：写指令子程序。
 CHECK_BF：判断是否忙碌子程序。
 WR_DATA：写数据子程序。

2. 程序分析解释

01 ~ 04：定义引脚。
 06：主程序。
 07：清除显示屏，并使复位光标。
 08：调用 WR_COMM 写指令子程序。
 09：调用 INIT_LCD 初始化子程序。
 11：写入显示起始地址为 81H，即第 1 行第 2 个位置。
 12：调用 WR_COMM 写指令子程序。
 13：输入欲显示的数据“ A ”。
 14：调用 WR_DATA 写数据子程序，显示“ A ”。
 16：写入显示起始地址 C6H，即第 2 行第 7 个位置。
 17：调用 WR_COMM 写指令子程序。
 18：输入欲显示的数据“ B ”。
 19：调用 WR_DATA 写数据子程序，显示“ B ”。
 20：动态停机，维持当前输出状态。
 22 ~ 29：LCM 初始化。设定 8 位数据传送、两行显示 5×7 点阵字型，并开启显示屏，显示光标，允许光标闪烁，自动右移等。
 31 ~ 37：写指令子程序 WR_COMM。
 39 ~ 47：判断是否忙碌子程序 CHECK_BF。
 49 ~ 55：写数据子程序 WR_DATA。
 57：程序结束。

13.3.3 实例测试

将写入程序的单片机插入实验板插座内，检查连接正常后接通电源，会看到在液晶显示屏第 1 行第 2 个位置上显示“ A ”，在第 2 行第 7 个位置上显示“ B ”。

可以在程序中将“ A ”和“ B ”的位置相互颠倒一下，再重新测试时会看到在液晶显示屏第 1 行第 2 个位置上显示的是“ B ”，在第 2 行第 7 个位置上显示的是“ A ”。

再将程序中显示位置代码#81H 改为#86H，原#0c6H 改为 #0c1H，再重新测试时，会看到液晶显示屏上显示字符的位置发生变化，第 1 行是在第 7 个位置显示，第 2 行是在第 2 个位置上显示。

13.3.4 经验总结

使 LCD 在第一行显示字符、在第二行显示字符、在两行同时显示，其程序结构基本是相同的，其区别只是在主程序中指出的该字符具体显示起始地址不同。

13.4 LCD 显示字符串

功能说明：LCD 显示字符串。开机后，从液晶显示屏第 1 行第 2 个位置开始显示字符串“Hello!”。本节硬件设计与 13.2 节相同。

13.4.1 程序设计

本程序采用查表的方法，在主程序中将表 LINE 中字符串的内容送入特殊寄存器 DPTR，并增加了查表显示子程序。

1. 流程图

主程序设计流程如图 13.6 (a) 所示，查表显示子程序流程如图 13.6 (b) 所示。

2. 程序

汇编语言编写的显示字符串源程序 LCD1303.ASM
代码如下：

```

01      RS      bit    P3.5
02      RW      bit    P3.6
03      E       bit    P3.7
04      LCD     EQU    P1
05 ;主程序
06 MAIN:
07      ACALL   INIT_LCD           ;调用初始化子程序
08
09      MOV      LCD, # 81H        ;写入显示起始地址
10      ACALL   WR_COMM          ;调用写指令子程序
11      MOV      DPTR, # LINE     ;字符串地址送入 DPTR
12      MOV      R0, # 6          ;字符串的字符数
13      ACALL   DISP_LCD         ;调用查表子程序
14      JMP      $              ;动态停机
15 ;LCM 初始化子程序
16 INIT_LCD:
17      MOV      LCD, #00000001B  ;清屏并复位光标
18      ACALL   WR_COMM          ;调用写指令子程序
19      MOV      LCD, #00111000B  ;设置 8 位、2 行、5×7 点阵
20      ACALL   WR_COMM          ;调用写指令子程序
21      MOV      LCD, #00001111B  ;显示器开，光标允许闪烁
22      ACALL   WR_COMM          ;调用写指令子程序

```

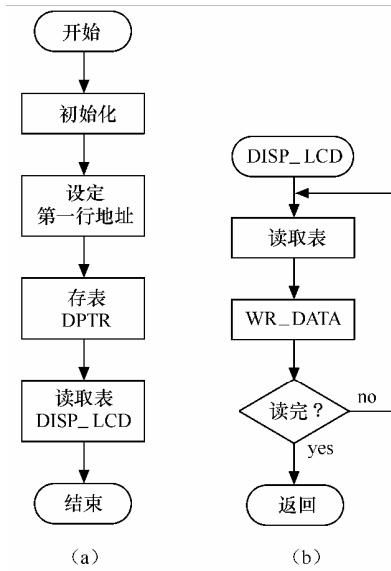


图 13.6 流程图

```

23    MOV     LCD, #00000110B      ;文字不动，光标自动右移
24    ACALL   WR_COMM            ;调用写指令子程序
25    RET                 ;子程序返回
26 ;写指令子程序
27 WR_COMM:
28    CLR     RS                ;RS=0，选择指令寄存器
29    CLR     RW                ;RW=0，选择写模式
30    CLR     E                 ;E=0，禁止读/写 LCM
31    ACALL   CHECK_BF          ;调用判断 LCM 忙碌子程序
32    SETB   E                 ;E=1，允许读/写 LCM
33    RET                 ;子程序返回
34 ;判断是否忙碌子程序
35 CHECK_BF:
36    MOV     LCD, #0FFH          ;此时不接收外来指令
37    CLR     RS                ;RS=0，选择指令寄存器
38    SETB   RW                ;RW=1，选择读模式
39    CLR     E                 ;E=0，禁止读/写 LCM
40    NOP                 ;延时 1μs
41    SETB   E                 ;E=1，允许读/写 LCM
42    JB     LCD.7, CHECK_BF    ;忙碌循环等待
43    RET                 ;子程序返回
44 ;写数据子程序
45 WR_DATA:
46    SETB   RS                ;RS=1，选择数据寄存器
47    CLR     RW                ;RW=0，选择写模式
48    CLR     E                 ;E=0，禁止读/写 LCM
49    ACALL   CHECK_BF          ;调用判断忙碌子程序
50    SETB   E                 ;E=1，允许读/写 LCM
51    RET                 ;子程序返回
52 ;查表显示子程序
53 DISP_LCD:
54    MOV     R1, #0              ;查表地址初始值
55 LOOP: MOV A, R1              ;将表地址初值赋予 A
56    MOVC   A, @A+DPTR         ;查表将字符串内容送入 A
57    MOV     LCD, A              ;将字符串内容送入 LCD
58    INC     R1                ;地址值加 1
59    ACALL   WR_DATA           ;调用写入数据子程序
60    DJNZ   R0, LOOP            ;判断查表是否 6 次？
61    RET                 ;子程序返回
62
63 LINE: DB     'Hello!', 00H    ;字符串
64
65 END                  ;程序结束

```

13.4.2 代码详解

1. 主要标号说明

MAIN : 主程序。

INIT_LCD : LCM 初始化设定。

WR_COMM : 写指令子程序。

CHECK_BF : 判断是否忙碌子程序。

WR_DATA : 写数据子程序。

DISP_LCD : 查表显示子程序

2. 程序详细说明

01 ~ 04 : 定义引脚。

06 : 主程序。

07 : 调用 INIT_LCD 初始化子程序。

09 : 写入字符串从第 1 行第 2 个位开始显示的起始地址。

10 : 调用 WR_COMM 写指令子程序。

11 : 将字符串的起始地址送入 DPTR 寄存器。

12 : 将字符串的字符数送入 R0。

13 : 调用查表显示子程序 , 将表中数据取出并输出。

14 : 动态停机 , 维持当前输出状态。

16 ~ 25 : LCM 初始化子程序。清显示屏 , 并使光标复位 , 设定 8 位数据传送、两行显示
5×7 点阵字型 , 并开启显示屏显示光标、允许光标闪烁、自动右移等。

27 ~ 33 : 写指令子程序 WR_COMM。

35 ~ 43 : 判断是否忙碌子程序 CHECK_BF。

45 ~ 51 : 写数据子程序 WR_DATA。

53 : DISP_LCD 为查表显示子程序。

54 : 将查表地址初始值 0 送入 R1。

55 : 将 R1 值赋予 A。

56 : 将查表取出的字符串送入 A。

57 : 将字符串内容送入 LCD。

58 : 地址值加 1 , 为取字符串的第二个字符做准备。

59 : 调用写入数据子程序 , 使送入 LCD 的内容能够输出。

60 : 判断查表是否为 6 次。该表中字符串的字符为 6 个 , 所以将 R0 先赋值为 6 , 每查一次表 R0 值减 1 , R0 的值减到 0 时 , 说明已经把表中字符串的内容读取完一遍 , 否则将跳转到 NEXT 处继续查表。

61 : 查表显示子程序返回。

63 : 字符串表。

65 : 程序结束。

13.4.3 实例测试

将写入程序的单片机插入实验板插座内，检查连接正常后接通电源，会看到在液晶显示屏第1行第2个位置上开始显示“Hello!”。

可以改动字符串中的字符数量重新测试，但必须同时改变查表次数 R0 数值，使之与字符数相一致。使用 20×2 的液晶显示屏，一行最多可显示 20 个字符。

13.4.4 经验总结

本节是 LCD 显示字符串程序，显示字符串如果采用前两节编程方法，每显示 1 个字符程序中就需要增加 4 条语句，这样，一个长的字符串就会使程序增加很多语句，编程时非常繁琐。所以，显示字符串一般采用查表的方式编程，既方便，又简化程序。

13.5 LCD 循环显示

功能说明：LCD 循环显示字符串。首先在第一行显示“Hello！”，2s 后在第二行显示“Welcome to LCD！”，再过 2s 后第一行改为“Nice to meet you”，再过 2s 后将第二行改为“Good luck！”本节硬件设计与 13.2 节相同。

13.5.1 程序设计

本程序是在上节程序的基础上，在主程序中进行 4 次存取字符串操作，每次显示字符串时停留 2s，更换屏幕时清屏，如此无限循环。因此在程序中增加了延时子程序和清屏子程序。

1. 流程图

主程序设计流程如图 13.7 所示。

2. 程序

汇编语言编写的 LCD 循环显示源程序 LCD1304.ASM 代码如下：

```

01     RS      bit  P3.5
02     RW      bit  P3.6
03     E       bit  P3.7
04     LCD    EQU   P1
05 ;主程序
06 MAIN:

```

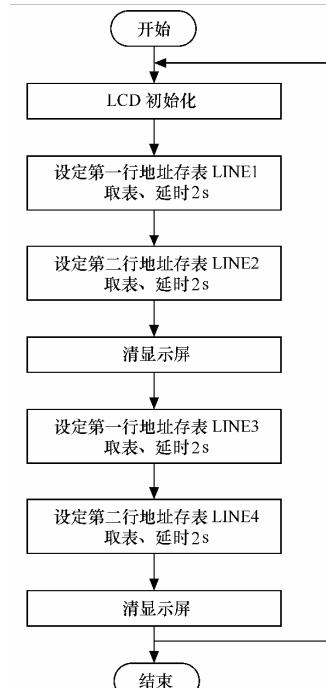


图 13.7 主程序流程

```

07    ACALL  INIT_LCD          ;调用初始化子程序
08
09    LOOP1:                  ;循环
10    MOV     LCD, #81H         ;写入显示起始地址
11    ACALL  WR_COMM          ;调用写指令子程序
12    MOV     DPTR, #LINE1     ;字符串地址送入 DPTR
13    MOV     R0, #6            ;字符串的字符数
14    ACALL  DISP_LCD         ;调用查表子程序
15    ACALL  DELAY            ;调用延时子程序
16
17    MOV     LCD, #0C0H        ;写入显示起始地址
18    ACALL  WR_COMM          ;调用写指令子程序
19    MOV     DPTR, #LINE2     ;字符串地址送入 DPTR
20    MOV     R0, #14           ;字符串的字符数
21    ACALL  DISP_LCD         ;调用查表子程序
22    ACALL  DELAY            ;调用延时子程序
23    ACALL  CLS              ;调用清屏子程序
24
25    MOV     LCD, #80H         ;写入显示起始地址
26    ACALL  WR_COMM          ;调用写指令子程序
27    MOV     DPTR, #LINE3     ;字符串地址送入 DPTR
28    MOV     R0, #16           ;字符串的字符数
29    ACALL  DISP_LCD         ;调用查表子程序
30    ACALL  DELAY            ;调用延时子程序
31
32    MOV     LCD, #0C0H        ;写入显示起始地址
33    ACALL  WR_COMM          ;调用写指令子程序
34    MOV     DPTR, #LINE4     ;字符串地址送入 DPTR
35    MOV     R0, #10           ;字符串的字符数
36    ACALL  DISP_LCD         ;调用查表子程序
37    ACALL  DELAY            ;调用延时子程序
38    ACALL  CLS              ;调用清屏子程序
39    JMP     LOOP1            ;无限循环
40 ;LCM 初始化子程序
41 INIT_LCD:
42    MOV     LCD, #00000001B   ;清屏并复位光标
43    ACALL  WR_COMM          ;调用写指令子程序
44    MOV     LCD, #00111000B   ;设置 8 位、2 行、5×7 点阵
45    ACALL  WR_COMM          ;调用写指令子程序
46    MOV     LCD, #00001111B   ;显示器开，光标允许闪烁
47    ACALL  WR_COMM          ;调用写指令子程序
48    MOV     LCD, #00000110B   ;文字不动，光标自动右移
49    ACALL  WR_COMM          ;调用写指令子程序

```

```

50      RET          ;子程序返回
51 ;写指令子程序
52 WR_COMM:
53     CLR    RS          ;RS=0 , 选择指令寄存器
54     CLR    RW          ;RW=0 , 选择写模式
55     CLR    E           ;E=0 , 禁止读/写 LCM
56     ACALL  CHECK_BF   ;调用判断 LCM 忙碌子程序
57     SETB    E          ;E=1 , 允许读/写 LCM
58     RET          ;子程序返回
59 ;判断是否忙碌子程序
60 CHECK_BF:
61     MOV    LCD, #0FFH   ;此时不接收外来指令
62     CLR    RS          ;RS=0 , 选择指令寄存器
63     SETB    RW          ;RW=1 , 选择读模式
64     CLR    E           ;E=0 , 禁止读/写 LCM
65     NOP          ;延时 1μs
66     SETB    E          ;E=1 , 允许读/写 LCM
67     JB     LCD.7, CHECK_BF  ;忙碌循环等待
68     RET          ;子程序返回
69 ;写数据子程序
70 WR_DATA:
71     SETB    RS          ;RS=1 , 选择数据寄存器
72     CLR    RW          ;RW=0 , 选择写模式
73     CLR    E           ;E=0 , 禁止读/写 LCM
74     ACALL  CHECK_BF   ;调用判断忙碌子程序
75     SETB    E          ;E=1 , 允许读/写 LCM
76     RET          ;子程序返回
77 ;查表显示子程序
78 DISP_LCD:
79     MOV    R1, #0         ;查表地址初始值
80 LOOP:  MOV    A, R1        ;将表地址初值赋予 A
81     MOVC   A,@A+DPTR    ;查表将字符串内容送入 A
82     MOV    LCD,A        ;将字符串内容送入 LCD
83     INC    R1          ;地址值加 1
84     ACALL  WR_DATA     ;调用写入数据子程序
85     DJNZ   R0, LOOP     ;判断查表是否 6 次 ?
86     RET          ;子程序返回
87 ;清屏子程序
88 CLS:
89     MOV    LCD, #00000001B
90     ACALL  WR_COMM
91     RET          ;子程序返回
92 ;延时 2s 子程序

```

```

93  DELAY:
94      MOV      R5, #100
95  D1:  MOV      R6, #100
96  D2:  MOV      R7,#100
97      DJNZ    R7, $
98      DJNZ    R6, D2
99      DJNZ    R5, D1
100     RET          ;子程序返回
101 ;字符串表
102 LINE1:   DB  'Hello!'
103 LINE2:   DB  'Welcome to LCD!'
104 LINE3:   DB  'Nice to meet you!'
105 LINE4:   DB  'Good luck!'
106
107 END        ;程序结束

```

13.5.2 代码详解

1. 主要标号说明

MAIN : 主程序。
INIT_LCD : LCM 初始化设定。
WR_COMM : 写指令子程序。
CHECK_BF : 判断是否忙碌子程序。
WR_DATA : 写数据子程序。
DISP_LCD : 查表显示子程序。
DELAY : 延时子程序。
CLS : 清显示屏子程序。

2. 程序详细说明

01 ~ 04 : 定义引脚。
06 : 主程序。
07 : 调用 INIT_LCD 初始化子程序。
09 : LOOP1 为语句标号。
10 : 写入字符串第 1 行第 2 个位开始显示的起始地址。
11 : 调用 WR_COMM 写指令子程序
12 : 将第 1 行 LINE1 字符串的地址送入 DPTR 寄存器。
13 : 将字符串的字符数送入 R0。
14 : 调用查表显示子程序 , 将表中数据读取并输出。
15 : 调用延时子程序 , 延时 2s。
17 : 写入字符串第 2 行第 1 个位开始显示的起始地址。

- 18 : 调用 WR_COMM 写指令子程序。
 19 : 将第 2 行 LINE2 字符串的地址送入 DPTR 寄存器。
 20 : 将字符串的字符数送入 R0。
 21 : 调用查表显示子程序 , 将表 LINE2 中的字符串读取并输出。
 22 : 调用延时子程序 , 延时 2s。
 23 : 调用清屏子程序。
 25 : 写入字符串第 1 行第 1 个位开始显示的起始地址。
 26 : 调用 WR_COMM 写指令子程序。
 27 : 将第 1 行 LINE3 字符串的地址送入 DPTR 寄存器。
 28 : 将字符串的字符数送入 R0。
 29 : 调用查表显示子程序 , 将表 LINE3 中的字符串读取并输出。
 30 : 调用延时子程序 , 延时 2s。
 32 : 写入字符串第 2 行第 1 个位开始显示的起始地址。
 33 : 调用 WR_COMM 写指令子程序。
 34 : 将第 2 行 LINE4 字符串的地址送入 DPTR 寄存器。
 35 : 将字符串的字符数送 10 入 R0 中。
 36 : 调用查表显示子程序 , 将表 LINE4 中的字符串读取并输出。
 37 : 调用延时子程序 , 延时 2s。
 38 : 调用清屏子程序。
 39 : 跳转到程序标号 LOOP1 处重新开始运行 , 无限循环。
 41 ~ 50 : LCM 初始化子程序。清屏并使光标复位 , 设定 8 位数据传送 , 2 行显示 , 5×7 点阵字型 , 并开启显示屏 , 显示光标 , 允许光标闪烁 , 自动右移。
 52 ~ 58 : 写指令子程序 WR_COMM。
 60 ~ 68 : 判断是否忙碌子程序 CHECK_BF。
 70 ~ 76 : 写数据子程序 WR_DATA。
 78 ~ 86 : 查表显示子程序。
 88 ~ 91 : 清显示屏子程序。
 93 ~ 100 : 延时子程序。
 102 ~ 105 : 字符串表。
 107 : 程序结束。

13.5.3 实例测试

将写入程序的单片机插入实验板插座内 , 检查连接正常后接通电源 , 会看到在液晶显示屏上滚动循环显示字符串 :

```
"Hello! " , "Welcome to LCD!" ;
"Nice to meet you! " , "Good luck".
```

13.5.4 经验总结

本程序是 LCD 循环显示字符串程序 , 在主程序中进行 4 次存取字符串操作 , 每次显示

字符串时停留 2s，更换屏幕时清屏，如此无限循环。上述采取每次显示字符串时停留 2s 及清除显示屏的目的是使字符串显示清晰、稳定。

液晶显示屏上的字符串实际上是一个一个字符的显示，由于速度快，所以会感觉到是一串一串地显示。如果在查表显示子程序中的第 84 行之后加有调用延时语句，就会出现字符一个一个跳出的效果，如写入：

```
MOV      R5, #25
ACALL   D1
```

13.6 自编图形显示

功能说明：在 LCD 显示屏上要显示温度标记符号“C”，其中“C”字符可以在 LCD 内的字符生成器 CGROM 里找到，CGROM 储存了 192 个 5×7 的点矩阵字型，包括字符 C。但是 CGROM 里却没有 C 的左上角的小圆点，需要自己设计。LCD 内有一个字形、字符产生器，简称 CGRAM，共有 512 位。

一个 5×7 点矩阵字型占用 8×8 位，所以 CGRAM 最多可提供 8 个自己设计的造型图。本节硬件设计与 13.2 相同。

13.6.1 程序设计

1. 5×7 点矩阵图

5×7 点矩阵图如图 13.8 所示，点矩阵图中 1 代表点亮该点元素，0 代表熄灭该点元素。*为无效位，可任意取 0 或 1，一般取 0。图 13.9 是本程序要建的字型，通过向表中输入相应数据，使点阵图中的 1 围成一个小圆形，就是我们在 LCD 显示屏上看到的“C”左上角的小圆圈。

*	*	*	0	0	0	0
*	*	*	0	0	0	0
*	*	*	0	0	0	0
*	*	*	0	0	0	0
*	*	*	0	0	0	0
*	*	*	0	0	0	0
*	*	*	0	0	0	0
*	*	*	*	*	*	*

图 13.8 5×7 点阵

*	*	*	0	1	1	0	0	00001100B=0CH
*	*	*	1	0	0	1	0	00010010B=12H
*	*	*	1	0	0	1	0	00010010B=12H
*	*	*	0	1	1	0	0	00001100B=0CH
*	*	*	0	0	0	0	0	00000000B=00H
*	*	*	0	0	0	0	0	00000000B=00H
*	*	*	0	0	0	0	0	00000000B=00H
*	*	*	*	*	*	*	*	00H 光标位置

图 13.9 自建图型

2. 程序

汇编语言编写的自编图形显示源程序 LCD1305.ASM 代码如下：

```
01      RS      bit      P3.5
02      RW      bit      P3.6
03      E       bit      P3.7
04      LCD     EQU      P1
05 ;主程序
```

```

06 MAIN:
07 ACALL INIT_LCD ;调用初始化子程序
08 ACALL TEMP_BJ ;显示温度标记"C"
09
10 ACALL STORE_DATA ;将自定义字符存入 CGRAM
11 MOV A, #0CBH ;写入显示起始地址
12 ACALL WR_COMM ;调用写指令子程序
13 MOV A, #00H ;CGRAM 内第 0 号图形
14 ACALL WR_DATA ;调用写数据指令子程序
15 JMP $ ;动态停机
16
17 ;显示温度标记子程序
18 TEMP_BJ:
19 MOV A, #0CBH ;设定第二行起始地址
20 ACALL WR_COMM ;调用写指令子程序
21 MOV DPTR, #BJ ;存代码表
22 MOV R1, #0 ;使指针指到表中第一个码
23 MOV R0, #2 ;取码次数
24 TP1:
25 MOV A, R1 ;A 为 0
26 MOVC A, @A+DPTR ;取码
27 ACALL WR_DATA ;调用写数据子程序
28 INC R1 ;R1 值加 1
29 DJNZ R0, TP1 ;判断是否将代码读取完？
30 RET ;子程序返回
31 BJ: ;代码表
32 DB 00H, "C"
33
34 ;将自定义字符写入 LCD1602 的 CGRAM 中
35 STORE_DATA:
36 MOV A, #40H ;指定 CG RAM 起始地址
37 ACALL WR_COMM ;将指令写入 LCD
38 MOV R2, #08H ;图形数据长度为 8 个字节
39 MOV DPTR, #TAB ;存代码表
40 MOV R3, #00H ;使指针指到表中第一个码
41 S_D: MOV A, R3 ;A 为 0
42 MOVC A, @A+DPTR ;读取表代码
43 ACALL WR_DATA ;调用写入数据指令
44 INC R3 ;R3 值加 1
45 DJNZ R2, S_D ;判断是否将代码读取完？
46 RET ;子程序返回
47 TAB: ;代码表
48 DB 0CH, 12H, 12H, 0CH

```

```

49     DB      00H,00H,00H,00H
50
51 ;初始化子程序
52 INIT_LCD:                                ;LCD 接口初始化
53     MOV      A, #01H                         ;LCD 清屏
54     ACALL   WR_COMM                         ;调用写指令子程序
55     ACALL   DELAY1                          ;延时 5ms
56     MOV      A, #38H                         ;设置 8 位、两行、5×7 点阵
57     ACALL   WR_COMM                         ;调用写指令子程序
58     ACALL   DELAY1                          ;延时 5ms
59     MOV      A, #0CH                          ;开显示屏
60     ACALL   WR_COMM                         ;调用写指令子程序
61     ACALL   DELAY1                          ;延时 5ms
62     RET                               ;子程序返回
63 ;写指令子程序
64 WR_COMM:
65     MOV      P1, A                           ;写入指令
66     CLR      RS                            ;RS=0，选择指令寄存器
67     CLR      RW                            ;RW=0，选择写模式
68     SETB    E                             ;E=1，允许读/写 LCM
69     ACALL   DELAY1                          ;延时 5ms
70     CLR      E                            ;E=0，禁止读/写 LCM
71     RET                               ;子程序返回
72
73 ;写数据子程序
74 WR_DATA:
75     MOV      P1, A                           ;写入数据
76     SETB    RS                            ;RS=1，选择数据寄存器
77     CLR      RW                            ;RW=0，选择写模式
78     SETB    E                             ;E=1，允许读/写 LCM
79     ACALL   DE                            ;延时 0.5ms
80     CLR      E                            ;E=0，禁止读/写 LCM
81     ACALL   DE                            ;延时 0.5ms
82     RET                               ;子程序返回
83 ;延时 0.5ms 子程序
84 DE:   MOV      R7, #250
85     DJNZ    R7, $
86     RET                               ;子程序返回
87 ;延时 5ms 子程序
88 DELAY1:
89     MOV      R6, #25
90 DL2:  MOV      R7, #100
91     DJNZ    R7, $

```

```

92    DJNZ      R6, DL2
93    RET          ;子程序返回
94
95    END          ;程序结束

```

13.6.2 代码详解

1. 主要标号说明

MAIN : 主程序。
INIT_LCD : LCM 初始化设定。
TEMP_BJ : 显示温度标记子程序。
STORE_DATA : 自定义字符子程序。
WR_COMM : 写指令子程序。
WR_DATA : 写数据子程序。

2. 程序分析解释

01 ~ 04 : 定义引脚。
06 : 主程序。
07 : 调用 INIT_LCD 初始化子程序。
08 : 调用显示温度标记 “C” 的子程序 TEMP_BJ。
10 : 调显示自定义字符子程序 STORE_DATA。
11 : 将自定义字符写入 LCD 第 2 行第 12 个位。
12 : 调用 WR_COMM 写指令子程序。
13 : LCD 里的 CGRAM 内可存 8 个自制图形 , 该图的行定义为第 0 号图形。
14 : 调用写数据指令将自制 0 号图形写入 LCD。
15 : 动态停机 , 维持当前输出状态。
18 ~ 32 : 显示温度标记中 “C” 字符程序。其显示位置设定在第 2 行第 12 位 , 由于代码表中第一个码为 00H , 是空位 , 第二个码才是 “C” , 所以 “C” 的显示位置实际是第 2 行第 13 位。
35 ~ 49 : 显示自制图形程序。与显示字符程序的区别主要有两点 , 一是要指定 CG RAM 起始地址 , 二是自制一个图形需要占 8 个字节 , 因此需要设定读取表 8 次。
52 ~ 62 : 对 LCD 进行初始化设置 , 包括清屏、设置工作模式。
64 ~ 71 : 写指令子程序。
74 ~ 82 : 写数据子程序。
84 ~ 86 : 延时 0.5ms 子程序。
88 ~ 93 : 延时 5ms 子程序。
95 : 程序结束。

3. 至本章用过的指令归类

- 数据传送类指令 : MOV、MOVC、PUSH、POP。

- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、RLC、RRC、CLR、ANL、ORL、XRL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI、JZ。
- 位操作类指令：CPL、CLR、SETB、JB、JNB、JBC、JNC、JC。
- 伪指令：END、DB、ORG。

13.6.3 实例测试

将写入程序的单片机插入实验板插座内，检查连接正常后接通电源，会看到在液晶显示屏第 2 行第 12 个位置上开始显示温度标记“ ”。

可以自制简单的汉字，如，将上述程序中 TAB 代码表的代码改为“ DB 1FH,04H,04H,1FH,04H,04H,1FH,00H ”后，将显示出自制的“ 王 ”字。

13.6.4 经验总结

LCD 内储存了常用的 192 个 5×7 的点矩阵字型。同时，LCD 内有字形、字符产生器 CGRAM，最多可提供存有 8 个自己设计的造型图。自己设计图形时，主要是编制代码表。

代码表的编制方法：

- (1) 先在纸上画出 5×7 点阵图，如图 13.8 所示。
- (2) 在勾出的有效范围 5×7 内，将要显示的位写为 1。
- (3) 查看书后附录中“数制式转换表”，将二进制数转换成十六进制数，该数即是代码。
- (4) 将代码列表。



第 14 章 AD 与 DA 及其应用

从宏观角度看，信号只有两种：模拟信号和数字信号。本章将讲述 AD/DA 转换基本原理及其应用。

14.1 信号转换概述

14.1.1 模拟信号

模拟 (Analog) 信号是人类可以直接感受到的信号，如温度、湿度、光线、声音等都属于模拟信号。模拟信号是一种连续性的信号，其波形如图 14.1 所示。对模拟信号直接处理与传输，容易失真，不容易保存。

14.1.2 数字信号

数字 (Digital) 信号是一种非 0 即 1 的非连续性的信号，通常有高、低两种电平，如图 14.2 所示。

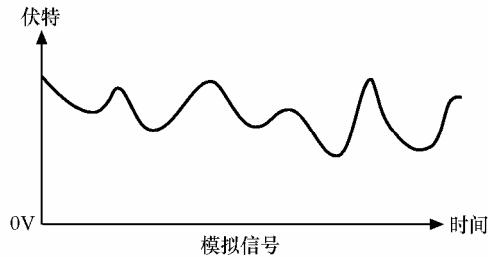


图 14.1 模拟信号示意图

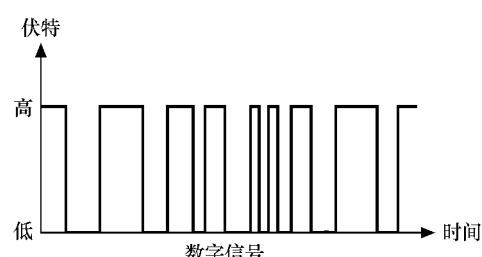


图 14.2 数字信号示意图

数字信号比较容易保存与处理，而且效率高，在传输上不易失真，可以使用计算机来对数据进行处理和控制。

14.1.3 信号转换

日常生活中，大部分数据是模拟信号，我们可以把测得的模拟信号经模拟/数字转换器

(analog-digital converter, 简称 ADC) 转换成数字信号, 这样可以进行较高效率的处理、保存和传输。当处理完成后, 再经数字/模拟转换器 (digital-analog converter, 简称 DAC) 转换成模拟信号, 以驱动控制设备, 如图 14.3 所示。

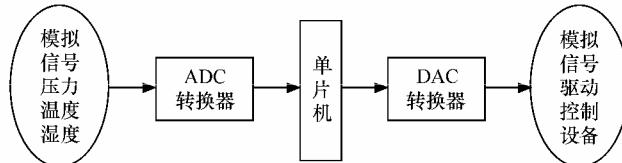


图 14.3 ADC、DAC 系统图

本书将以简单的 R-2R 梯形 DAC 转换器及单线数字温度传感器 ADC 来介绍 AD/DA 转换及其应用。

14.2 简单 DA 转换程序

功能说明: 将数据 151 送入 P2 输出, 经 DAC 转换, 输出 3V 直流电压, 并在 P1 端口由 8 位 LED 显示二进制数。

14.2.1 硬件设计

电路设计如图 14.4 所示。

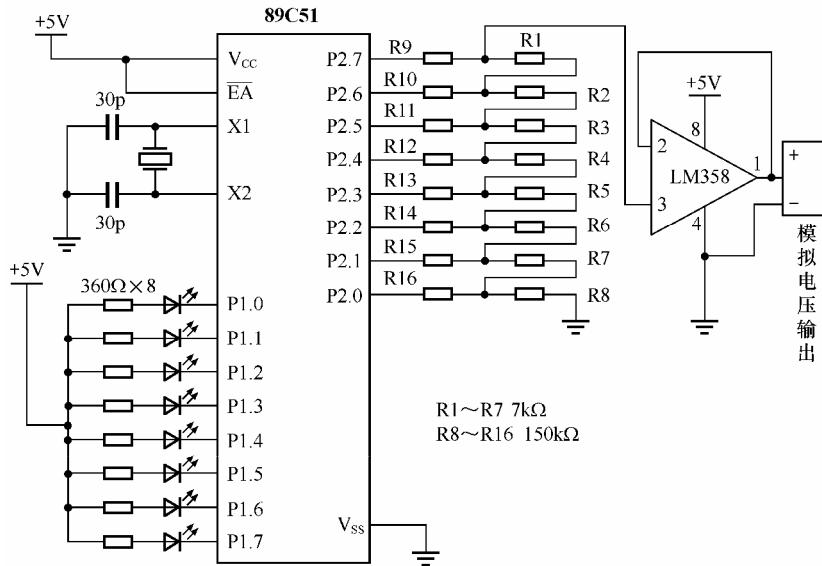


图 14.4 DAC 转换电路

单片机 P2 端口连有 R-2R 电阻网络, 即 R-2R 梯形 DAC 转换器。数字信号由 P2 输出, 经 R-2R 梯形 DAC 转换, 最后经运算放大器转成模拟电压输出。运算放大器采用 LM358 型

号 IC，它是低功耗双运算放大器，引脚如图 14.5 所示。

14.2.2 程序设计

该电路由 R-2R 电阻组成梯形 DAC 转换器，该转换器有 8 条输入线，故有 00H ~ FFH 一共 256 种不同输入组合以得到不同的输出电压。

当输入数据为 00H 时，其输出电压为 0V；当输入数据为 FFH 时，其输出电压为 5V。本程序输入数据为 151，其输出电压约为 3V，其运算关系式如下。

$$V_0 = \frac{N}{255} \times 5V = \frac{151}{255} = 2.96V$$

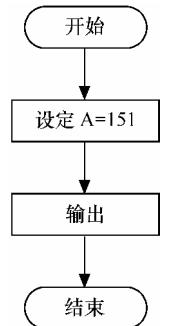


图 14.6 程序流程

注：

V_0 为模拟输出电压。
 N 为计数所需电压的数字值。

1. 流程图

程序设计流程如图 14.6 所示。

2. 程序

汇编语言编写的简单 DA 转换程序源程序 DA14-1.ASM 代码如下：

01	MOV	A, #151	; 设定 A 值为 151
02	MOV	P2, A	; 将数字送入 P2 输出
03	MOV	P1, A	; 将数字送入 P1 输出
04	JMP	\$; 程序在此等待
05	END		; 程序结束

14.2.3 代码详解

这是一个非常简单的程序，主要是通过 R-2R 梯形 DAC 转换器将输入的数字 151 转换为 3V 模拟输出电压。

01：将数字 151 输入累加器 A 中，151 是输出 3V 电压所需的数字。

02：将数字送入 P2 输出。P2 的输出端是 R-2R 梯形 DAC 转换器的输入端，通过 DAC 转换成模拟输出电压 3V。

03：由 P1 输出数字 151 的二进制数 10010111B。

04：程序在此动态停机，维持输出状态。

05：程序结束。

14.2.4 实例测试

测试该程序时，要首先检查一下实验板上装的 R-2R 梯形 DAC 转换器是否与程序中输出端口一致，如果不一致，需要按照实验板实际电路更改程序输出端口。

当实验板连接线路检查好后接通电源，使用万用表可在电路模拟电压输出端测出 3V 直

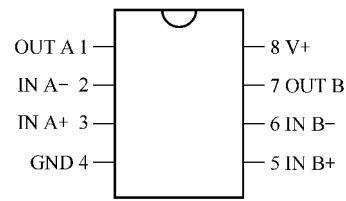


图 14.5 LM358 引脚图

流电压。同时，可以观察 P1 端口由 8 个 LED 显示的二进制数，其值显示为 10010111B，二进制数 10010111B 转换为十进制数为 151。

14.2.5 经验总结

该程序很简单，将数字转换为电压是由 R-2R 梯形 DAC 转换器完成的。其数字调整的有效范围为 0~255，相应电压变化范围为 0~5V，每改变一个数字，电压会变化 0.02V。

14.3 指拨开关控制输出电压

功能说明：使用 DIP-8 指拨开关控制模拟输出电压。开关状态由 P3 输入，经 P2 输出到 DAC 转换器，再转换成模拟电压输出。

14.3.1 硬件设计

电路设计如图 14.7 所示。

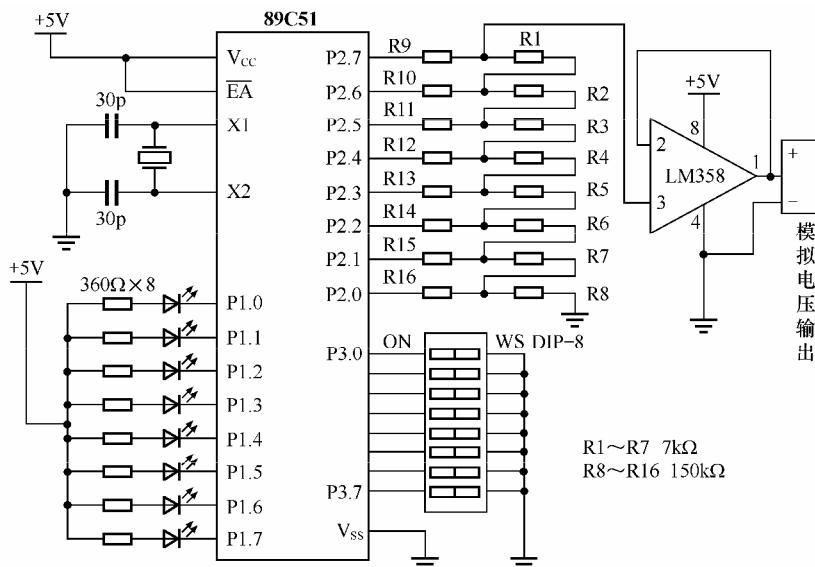


图 14.7 指拨开关控制输出电压电路

P3 端口增加一组 DIP-8 指拨开关，作为二进制数的输入端。

14.3.2 程序设计

该程序将上节程序中输入固定数字数据，改为使用一组 DIP-8 指拨开关输入二进制数据，数据由 P3 读入。

1. 流程图

程序设计流程图如 14.8 所示。

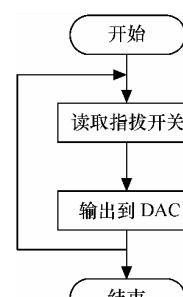


图 14.8 流程图

2. 程序

汇编语言编写的指拨开关控制输出电压源程序 DA14-2.ASM 代码如下：

```

01    LOOP:   MOV     A, P3          ; 读取 P3 开关状态
02        MOV     P2, A           ; 送入 P2 输出至 DAC 转换
03        MOV     P1, A           ; 送入 P1 输出
04        JMP     LOOP          ; 跳转到 LOOP 处，循环
05        END                 ; 程序结束

```

14.3.3 代码详解

该程序是在上节程序基础上，将输入固定数据改为读入开关 P3 状态，即由开关形成二进制数，然后送入 P2 输出至 DAC 转换成模拟电压，同时又送入 P1 输出，由 LED 显示二进制数。

01：读取开关 P3 状态。由于使用 8 位开关输入，所以可以组成 00H ~ FFH 不同的输入数字数据。当 8 位开关都断开时，读入的数据为 1111111B，模拟电压输出为 5V；当 8 位开关都接通时，读入的数据为 0000000B，模拟电压输出为 0V。

02：将读入 P3 的数字数据送入 P2 输出至 DAC 转换器，经 DAC 转换器输出模拟电压，调节开关状态，就能控制输出模拟电压。

03：将数字信号同时送入 P1 输出，P1 接有 8 个 LED，可以显示二进制数，亮的位为 0，不亮的位为 1。

04：跳到 LOOP 处，不断循环。

05：程序结束。

14.3.4 模拟仿真

该程序进行模拟仿真时，需要在程序中使用模拟开关语句，如，模拟第一位开关接通需要输入数据 11111110B，此时需要在程序中增加模拟第一位开关接通的语句“CLR P3.0”，使程序改为：

```

LOOP: CLR      P3.0
      MOV     A, P3          ; 读取 P3 开关状态
      MOV     P2, A           ; 送入 P2 输出至 DAC 转换
      MOV     P1, A           ; 送入 P1 输出
      JMP     LOOP          ; 跳到 LOOP 处，循环
      END                 ; 程序结束

```

再如，要模拟第 3 位开关接通，需要输入数据 11111011B，此时需要在程序中增加模拟第 3 位开关接通的语句“CLR P3.2”。

14.3.5 实例测试

实例测试时，每改变一个指拨开关的接通状态，就改变了输入的二进制数。此时，使用万用表可以在模拟电压输出端测出输出直流电压值，观察分析其输出的模拟电压是否符合指

拔开关输入数据。

14.3.6 经验总结

本节程序对上节程序的实用性进行改进，由输入固定数字数据改为使用一组 DIP-8 指拨开关输入数据，无需改变程序就可以很方便地改变输出直流电压值。

指拨开关输入的是二进制数，没有十进制数直观。而且 14.2 节介绍的运算关系公式也是使用十进制数，为了方便将二进制数转换成十进制数，下面给出二进制/十进制转换表，如表 14.1 所示。

表 14.1

二进制/十进制转换表

制 式	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位	第 0 位
二进制位	1	1	1	1	1	1	1	1
2 的幂	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
十进制	128	64	32	16	8	4	2	1

14.4 DAC 输出锯齿波

功能说明：当输入数字每次都较前次增加 1 时，DAC 输出电压也上升 1 个单位，因此可以得到一个 255 阶的锯齿波。本节硬件设计与 14.2 节相同。

14.4.1 程序设计

输入的数字是从 0 开始，每次增加 1，增到 255 后降为 0，再从 0 开始增加，不断循环。输出的模拟电压也随着数字的增加由 0V 经 255 阶增到 5V，然后降到 0V，再从 0V 开始增加，如此循环。所以，输出的电压是一个 255 阶的锯齿波。

1. 流程图

程序设计流程如图 14.9 所示。

2. 程序

汇编语言编写的 DAC 输出锯齿波源程序 DA14-3.ASM 代码如下：

```

01  MAIN:                                ;主程序
02      MOV     A, #0H                      ;设定 A 初始值
03  LOOP:       MOV     P2, A              ;由 P2 输出
04      MOV     P1, A              ;送入 P1 输出
05      ACALL   DELAY                  ;调用延时程序
06      INC     A                   ;A 值加 1
07      JMP     LOOP                  ;跳转到 LOOP 处

```

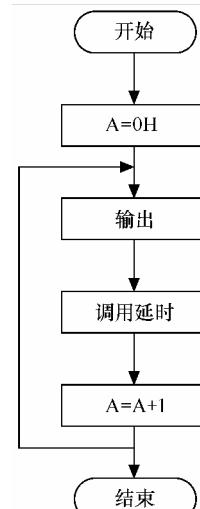


图 14.9 流程图

```

08 ;延时子程序
09  DELAY:
10      MOV     R5, #5
11  D1:    MOV     R6, #100
12  D2:    MOV     R7, #100
13      DJNZ    R7, $
14      DJNZ    R6, D2
15      DJNZ    R5, D1
16      RET
17      END          ;程序结束

```

14.4.2 代码详解

- 01：DAC 转换产生锯齿波的主程序。
- 02：将累加器 AD 的初始值设定为 0，使输入的数字从 0 开始。
- 03：数字信号从 P2 输出，经 DAC 转换成模拟电压。
- 04：数字信号同时送入 P1 输出，P1 接有 8 个 LED，可以显示二进制数，亮的位为 0，不亮的位为 1。
- 05：调延时子程序，延时 0.1s。
- 06：寄存器 A 值加 1。特殊寄存器 A 为 8 位寄存器，当寄存的数字超过 255 后，将会重新由 0 开始计数。
- 07：跳转到 LOOP 处，不断循环。
- 09 ~ 16：延时子程序。
- 17：程序结束。

14.4.3 模拟仿真

模拟仿真时，特别注意端口 P2 的变化，P2 的输出直接进入 DAC 转换成模拟电压，所以端口 P2 的变化反映数字的变化。

端口的模拟观察窗口中 P2 的 8 个小点表示 8 位，点呈现黄色时，代表二进制数的 1，点为白色时，表示为 0。

当程序执行第 2 句时，会看到 8 个点都为白色，此时表示数字输出为 0，模拟电压也为 0V，然后数字在不断增加，当看到 8 个点都变黄色时，说明数字增到 255，模拟电压增到 5V，之后 8 个点又都变白色，表明输出数字又变为 0，模拟电压也为 0V，如此循环。

14.4.4 实例测试

实例测试时，使用万用表测试模拟电压输出端，要注意表笔正负极性，不要接反。此时会观察到测出的输出电压由 0 开始逐渐增加到 5V，然后突然回到 0V，完成一个锯齿波形。然后电压再重新由 0 开始逐渐增加，如此不断循环，不断输出锯齿波。

14.4.5 经验总结

数字信号转换成模拟信号的程序比较简单，只要将输出的数字信号送到 DAC 转换器就能

转换成模拟信号。DAC转换器实际是由电阻网络所构成，常见的是前边介绍的R-2R电阻网络。

14.5 单线数字温度传感器

美国DALLAS公司生产的单线数字温度传感器DS18B20，可以把温度信号直接转换成串行数字信号供微机处理，是模/数转换器件，而且读DS1820信息或写DS1820信息仅需要单线接口，使用非常方便。

新型的单线数字温度传感器DS18B20体积更小、精度更高、使用更灵活，其引脚排列如图14.10所示。

14.5.1 引脚及其与单片机的连接方式

1. 引脚

- GND接地。
- DQ为数字信号输入/输出端。
- VDD为外接电源输入端（在寄生电源接线方式时接地）。



图14.10 DS18B20
引脚排列图

2. 与单片机的连接方式

单线数字温度传感器DS18B20与单片机连接电路非常简单，引脚1(GND)接地，引脚3(VCC)接电源+5V，引脚2(DQ)接单片机输入/输出一个端口，电源+5V和信号线(DQ)之间接有一个 $4.7k\Omega$ 的电阻。

由于每片DS18B20含有惟一的串行数据口，所以在一条总线上可以挂接多个DS18B20芯片。

外部供电方式单点测温电路如图14.11所示。

外部供电方式多点测温电路如图14.12所示。

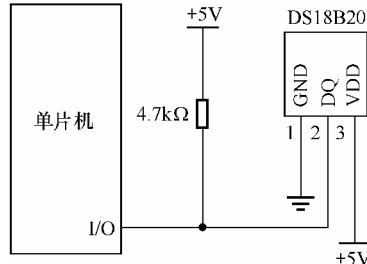


图14.11 外部供电方式单点测温电路

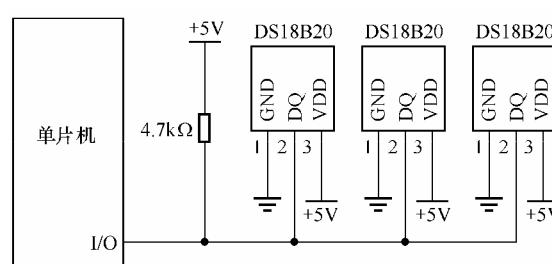


图14.12 外部供电方式多点测温电路

14.5.2 DS18B20的主要特性

- 使用电压范围宽，电压范围为 $3.0V \sim 5.5V$ ，在寄生电源方式下可由数据线供电。
- 单线接口方式，DS18B20在与微处理器连接时仅需要一条线即可实现微处理器与DS18B20的双向通信。
- DS18B20支持多点组网功能，实现组网多点测温。

- DS18B20 在使用中不需任何外围元件，全部传感元件及转换电路都集成在形如一只三极管的集成电路内。
- 测温范围为 $-55 \sim +125$ ，在 $-10 \sim +85$ 时精度为 ± 0.5 。
- 可编程的分辨率为 9 ~ 12 位，对应的可辨温度分别为 0.5° 、 0.125° 和 0.0625° ，可实现高精度测温。
- 转换速度快，在 9 位分辨率时最多在 93.75ms 内把温度转换为数字，12 位分辨率时最多在 750ms 内把温度转换为数字。
- 测量结果直接输出数字温度信号，以“一线总线”串行传送给 CPU，同时可传送 CRC 校验码，具有极强的抗干扰纠错能力。
- 用户可自设定温度报警上下限，其值是非易失性的。
- 报警搜索命令可识别哪片 DS18B20 超温度限。

14.5.3 内部结构

DS18B20 内部结构主要由 4 部分组成：64 位光刻 ROM，温度传感器，非挥发的温度报警触发器 TH、TL 和配置寄存器，如图 14.13 所示。

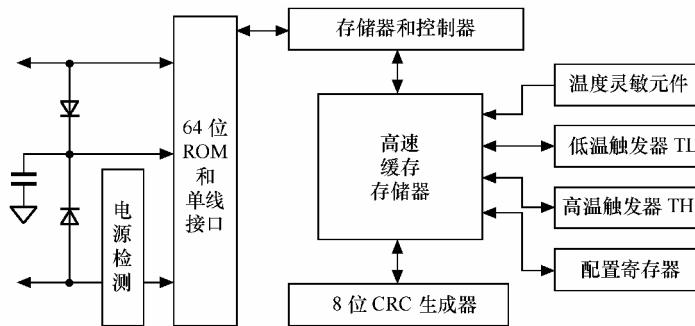


图 14.13 DS18B20 内部结构

光刻 ROM 中的 64 位序列号是出厂前被光刻好的，是 DS18B20 的地址序列码，使每一个 DS18B20 都各不相同，这样就可以实现一根总线上挂接多个 DS18B20 的目的。

DS18B20 中的温度传感器完成对温度的测量。

DS18B20 温度传感器内部的存储器，包括一个高速暂存 RAM 和一个非易失性的可电擦除的 E²PRAM，后者存放高温度和低温度触发器 TH、TL 和结构寄存器。

配置存储器，主要用来设置 DS18B20S 的工作模式和分辨率。

14.5.4 高速暂存存储器

高速暂存存储器由 9 个字节组成，其分配如表 14.2 所示。

当温度转换命令发出后，经转换所得的温度值存放在高速暂存存储器的第 0 和第 1 个字节内。第 0 个字节存放的是温度的低 8 位信息，第 1 个字节存放的是温度的高 8 位信息。单片机可通过单线接口读到该数据，读取时低位在前，高位在后。

第 2、3 个字节是 TH、TL 的易失性拷贝，第 4 个字节是结构寄存器的易失性拷贝，这 3 个字节的内容在每一次上电复位时被刷新。

第 5、6 和 7 个字节用于内部计算，第 8 个字节是冗余检验字节。

表 14.2 DS18B20 暂存寄存器分布表

寄存器内容	字节地址	寄存器内容	字节地址
温度值低位 (LS)	0	保留	5
温度值高位 (MS)	1	保留	6
高温限值 (TH)	2	保留	7
低温限值 (TL)	3	CRC 校验值	8
配置寄存器	4		

14.5.5 DS18B20 通信协议

在对 DS18B20 进行读写编程时，必须严格保证读写时序，否则将无法读取测温结果。根据 DS18B20 的通信协议，主机控制 DS18B20 完成温度转换必须经过 3 个步骤：每一次读写之前都要对 DS18B20 进行复位，复位成功后发送一条 ROM 指令，最后发送 RAM 指令，这样才能对 DS18B20 进行预定的操作。

复位要求主 CPU 将数据线下拉 500μs，然后释放，DS18B20 收到信号后等待 16~60μs，然后发出 60~240μs 的存在低脉冲，主 CPU 收到此信号表示复位成功。

DS18B20 的 ROM 指令如表 14.3 所示，RAM 指令如表 14.4 所示。

表 14.3 ROM 指令表

指 令	约定代码	功 能
温度变化	44H	启动 DS18B20 进行温度转换，12 位转换时最长为 750ms (9 位为 93.75ms)，结果存入内部 9 字节 RAM 中
读暂存器	0BEH	读内部 RAM 中 9 字节的内容
写暂存器	4EH	发出向内部 RAM 的 3、4 字节写上、下限温度数据命令，紧跟该命令之后是传送两字节的数据
复制暂存器	48H	将 RAM 中第 3、4 字节的内容复制到 E ² PROM 中
重调 E ² PROM	0B8H	将 E ² PROM 中内容恢复到 RAM 中的第 3、4 字节
读供电方式	0B4H	读 DS18B20 的供电模式。寄生供电时 DS18B20 发送 0，外接电源供电 DS18B20 发送 1

表 14.4 RAM 指令表

指 令	约定代码	功 能
读 ROM	33H	读取 DS18B20 温度传感器 ROM 中的编码 (即 64 位地址)
符合 ROM	55H	发出命令之后，接着发出 64 位 ROM 编码，访问单总线上与该编码对应的 DS18B20，使之作出响应，为下一步对该 DS18B20 读写作准备
搜索 ROM	0F0H	用于确定挂接在同一总线上 DS18B20 的个数和识别 64 位 ROM 地址，为操作各器件作好准备
跳过 ROM	0CCR	跳过 ROM 工作
报警搜索命令	0ECH	执行后只有温度超过设定值上限或下限的芯片才能作出响应

14.5.6 使用注意事项

DS1820 虽然具有测温系统简单、测温精度高、连接方便、占用口线少等优点，但在实际应用中也应注意以下几个问题。

- 因为硬件开销较小，需要复杂的软件进行补偿，由于 DS1820 与微处理器间采用串行数据传送，因此，在对 DS1820 进行读写编程时必须严格保证读写时序，否则将无法读取测温结果。
- 当单总线上所挂 DS1820 超过 8 个时，就需要解决微处理器的总线驱动问题，这一点在进行多点测温系统设计时要加以注意。
- 连接 DS1820 电缆的长度超过 50m 时，最好采用屏蔽 4 芯双绞线，其中一对为接地线与信号线，另一组接 VCC 和地线，屏蔽层在源端单点接地，正常通信距离可达 150m。
- 在 DS1820 测温程序设计中，向 DS1820 发出温度转换时总要等待 DS1820 的返回信号，一旦某个 DS1820 接触不好或断线当程序读该 DS1820 时，将没有返回信号，程序进入死循环。这一点在进行 DS1820 硬件连接和软件设计时也要加以注意。

14.6 数字温度计

功能说明：使用单线数字温度传感器 DS18B20 把温度信号直接转换成数字信号输入单片机，经单片机处理后，将实时温度显示在两个 7 段 LED 数码管上。

14.6.1 硬件设计

电路设计如图 14.14 所示。

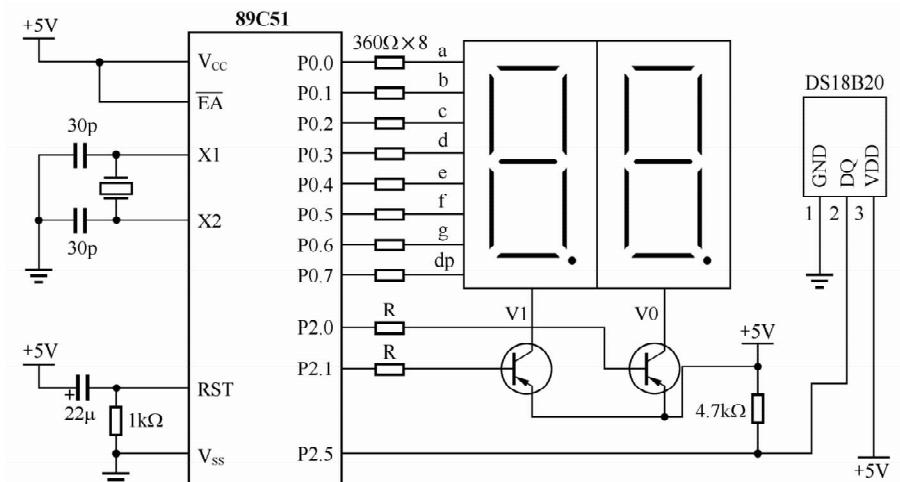


图 14.14 数字温度计电路

单线数字温度传感器 DS18B20 的引脚 2 接单片机 P2.5 端口，用于串行数据通信。引脚 3 接电源+5V，引脚 2 与电源+5V 之间接有一个 4.7kΩ 的上拉电阻，传感器引脚 1 接地。

单片机的P0端口连接两个7段LED数码管，显示温度值。

14.6.2 程序设计

程序开始首先对温度传感器DS18B20进行复位，检测是否正常工作；接着读取温度数据，主机发出CCH指令与在线的DS18B20联系，接着向DS18B20发出温度A/D转换44H指令，再发出读取温度寄存器的温度值BE指令，并反复调用复位、写入及读取数据子程序，之后再经数据转换，由数码管显示出来，不断循环。

1. 流程图

主程序设计流程如图14.15所示。

2. 程序

汇编语言编写的数字温度计源程序AD14-4.ASM代码如下：

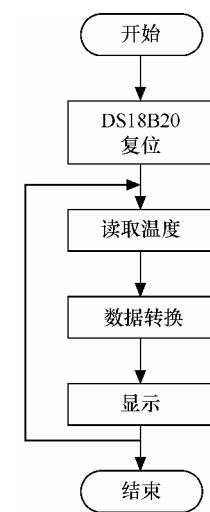


图14.15 主程序流程

```

01      A_BIT      EQU 20H          ;存放个位数变量
02      B_BIT      EQU 21H          ;存放十位数变量
03      FLAG       EQU 38H          ;DS18B20是否存在标志
04      DQ         EQU P2.5        ;DQ引脚由P2.5控制
05  MAIN:                                ;主程序
06      ACALL     RE_TEMP        ;调用读取温度子程序
07      ACALL     TURN          ;数据转化子程序
08      ACALL     DISPLAY        ;显示子程序
09      JMP       MAIN          ;循环
10  ;初始化及读取温度值子程序
11  RE_TEMP:
12      SETB      DQ
13      ACALL     RESET_1820    ;调用复位子程序
14      JB       FLAG, ST      ;判断DS1820是否存在
15      RET
16  ST:                                ;DS18B20存在
17      MOV       A, #0CCH        ;跳过ROM匹配
18      ACALL     WRITE_1820    ;调用写入数据子程序
19      MOV       A, #44H         ;发出温度转换命令
20      ACALL     WRITE_1820    ;调用写入数据子程序
21      ACALL     RESET_1820    ;准备读温度前先复位
22      MOV       A, #0CCH        ;跳过ROM匹配
23      ACALL     WRITE_1820    ;调用写入数据子程序
24      MOV       A, #0BEH        ;发出读温度命令
25      ACALL     WRITE_1820    ;调用写入数据子程序
26      ACALL     READ_1820     ;调用读取数据子程序
27      RET

```

```

28 ;复位子程序
29 RESET_1820:
30     SETB      DQ
31     NOP
32     CLR      DQ
33 ;
34 ;主机发出复位低脉冲
35     MOV      R1,#3
36 DLY:   MOV      R0, #107
37     DJNZ    R0, $
38     DJNZ    R1,DLY
39 ;
40 ;然后拉高数据线
41     SETB      DQ
42     NOP
43     NOP
44     NOP
45 ;
46 ;等待 DS18B20 回应
47     MOV      R0, #25H
48 T2:   JNB      DQ ,T3
49     DJNZ    R0, T2
50     JMP      T4
51 ;
52 ;标志位 FLAG=1 , 表示 DS1820 存在
53 T3:   SETB      FLAG
54     JMP      T5
55 ;
56;标志位 FLAG=0 , 表示 DS1820 不存在
57 T4:   CLR      FLAG
58     JMP      T7
59 ;
60 ;时序要求延时一段时间
61 T5:   MOV      R0, #117
62 T6:   DJNZ    R0, T6
63 ;
64 T7:   SETB      DQ
65     RET
66 ; 写入子程序
67 WRITE_1820:
68     MOV      R2, #8          ;一共 8 位数据
69     CLR      C              ;C=0
70 WR1:

```

```

71      CLR      DQ          ;总线低位，开始写入
72      MOV      R3, #7
73      DJNZ    R3,$        ;保持 16μs 以上
74      RRC      A          ;把字节 DATA 分成 8 个位，环移给 C
75      MOV      DQ, C       ;写入一个位
76      MOV      R3, #23
77      DJNZ    R3, $        ;等待
78      SETB    DQ          ;重新释放总线
79      NOP
80      DJNZ    R2, WR1       ;写入下一个位
81      SETB    DQ
82      RET
83 ;读取子程序
84 READ_1820:
85      MOV      R4, #2        ;读出两个字节的数据
86      MOV      R1, #29H      ;低位存入 29H，高位存入 28H
87 RE0:
88      MOV      R2, #8        ;数据一共有 8 位
89 RE1:
90      CLR      C
91      SETB    DQ
92      NOP
93      NOP
94      CLR      DQ          ;读前总线保持为低
95      NOP
96      NOP
97      NOP
98      SETB    DQ          ;开始读总线释放
99      MOV      R3, #9
100 RE2:
101     DJNZ    R3, RE2       ;延时 18μs
102     MOV      C, DQ        ;从总线读到一个位
103     MOV      R3, #23
104 RE3:
105     DJNZ    R3, RE3       ;等待 50μs
106     RRC      A          ;把读得的位值环移给 A
107     DJNZ    R2, RE1       ;读下一个位
108     MOV      @R1, A
109     DEC      R1
110     DJNZ    R4, RE0
111     RET
112 ; 数据转化子程序
113 TURN:

```

```

114      MOV      A, 29H
115      MOV      C, 40H          ;将 28 中的最低位移入 C
116      RRC      A
117      MOV      C, 41H
118      RRC      A
119      MOV      C, 42H
120      RRC      A
121      MOV      C, 43H
122      RRC      A
123      MOV      29H, A
124      RET
125 ; 温度显示子程序
126 DISPLAY:
127      MOV      A, 29H
128      MOV      B, #10
129      DIV      AB
130      MOV      B_BIT, A        ;十位在 A
131      MOV      A_BIT, B        ;个位在 B
132      MOV      DPTR, #TABLE   ;指定查表启始地址
133      MOV      R0, #4
134 DP1:
135      MOV      R1, #250        ;显示 1000 次
136 LOOP:
137      MOV      A, A_BIT        ;取个位数
138      MOVC     A, @A+DPTR    ;查个位数的 7 段代码
139      MOV      P0, A          ;送出个位的 7 段代码
140      CLR      P2.3          ;开个位显示
141      ACALL    DELAY
142      SETB    P2.3
143      MOV      A, B_BIT        ;取十位数
144      MOVC     A, @A+DPTR    ;查十位数的 7 段代码
145      MOV      P0, A          ;送出十位的 7 段代码
146      CLR      P2.2          ;开十位显示
147      ACALL    DELAY        ;显示 1ms
148      SETB    P2.2
149      DJNZ    R1, LOOP        ;250 次未完循环
150      DJNZ    R0, DP1         ;4 个 250 次未完循环
151      RET
152 ; 延时子程序
153 DELAY:                 ;
154      MOV      R7, #80
155      DJNZ    R7, $
156      RET

```

```

157 ; 代码表
158 TABLE:
159     DB  0C0H,0F9H,0A4H,0B0H,99H
160     DB  92H,82H,0F8H,80H,90H
161
162     END                      ;程序结束

```

14.6.3 代码详解

1. 程序分析解释

01：定义20H单元存放个位数单元，名为A_BIT。

02：定义21H单元存放十位数单元，名为B_BIT。

03：定义38H单元存放DS18B20是否存在的标志，名为FLAG，FLAG=1，表示DS18B20存在；FLAG=0，表示DS18B20不存在。

04：定义DS18B20的DQ引脚（数据输入/输出引脚）由单片机P2.5引脚控制。

05：主程序开始。

06：调用传感器设置及读取温度子程序，其中包括对传感器DS18B20的复位、写入和读取温度数据。

07：调用数据转化子程序，获得实际测量的温度。

08：调用显示子程序，将测量的温度数据在两位数码管上显示出来。

09：程序循环。

11~27：读取温度数据子程序。

传感器DS18B20的连接很简单，但读取温度数据的过程较为复杂，有严格的时序要求。先进行复位，搜索DS18B20是否存在，如果存在，主机将发出CCH指令与在线的DS18B20联系；接着向DS18B20发出温度A/D转换44H指令；再发出读取温度寄存器的温度值BE指令。

在此过程中还需要调用复位、写入及读取数据子程序，用较为复杂的软件换取简单的硬件接口。

29~65：DS18B20的复位子程序。根据DS18B20的通信协议，每一次读写数据之前都要对DS18B20进行复位，复位要求主机先发出复位低脉冲（大于480μs），然后释放，DS18B20收到信号后等待16~60μs，然后发出60~240μs的存在低脉冲，主机收到此信号表示复位成功。

67~82：DS18B20的写入子程序，将数据写入温度寄存器中，有时序要求。

84~111：DS18B20的读取子程序，读取温度寄存器的温度值，也有时序要求。

113~124：数据转化子程序。传感器DS18B20所测得的温度数据低位存入29H(TEMPER_L)，高位存入28H(TEMPER_H)，由于本程序中不要求显示小数，可以舍去29H的低4位，将28H中的低4位移入29H的高4位，获得一个新字节，这个字节就是实际测量的温度。

126~151：温度显示子程序，将十六进制数转换成十进制数，在两位数码管上显示。

153~156：延时子程序，延时的时间为160μs。

158~160：数码管共阳极0~9代码表。

162：程序结束。

2. 至本章用过的指令归类

- 数据传送类指令：MOV、MOVC、PUSH、POP。
- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、RLC、RRC、CLR、ANL、ORL、XRL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI、JZ。
- 位操作类指令：CPL、CLR、SETB、JB、JNB、JBC、JNC、JC。
- 伪指令：END、DB、ORG。

14.6.4 实例测试

将写入程序的单片机插入实验板插座内，检查温度传感器 DS18B20 连接正常后接通电源，此时，在两位 7 段 LED 数码管上将会准确地显示出环境温度，无需做任何调整。

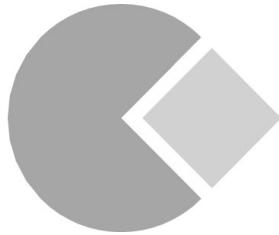
为了观察温度传感器 DS18B20 对稳定变化的灵敏度，可以用手捏住 DS18B20 管，会看到数码管上显示的温度很快上升至人体温度值，再将手离开 DS18B20 管，温度又会很快降到环境温度值。

温度传感器 DS18B20 的测温范围为 $-55 \sim +125$ ，在 $-10 \sim +85$ 时精度为 ± 0.5 。

14.6.5 经验总结

温度传感器 DS18B20 外形象一个小三极管，硬件连接非常简单，应用非常方便。它不仅能测量温度，而且也是一个 ADC 转换器，它能将测得的温度信号直接转换成数字信号输入到单片机。

硬件开销较小，相对需要复杂的软件进行补偿，DS18B20 软件编程比较复杂，但是可以把复位、读和写 3 个基本操作步骤的子程序看成是 3 个固定的基本模块，编写使用温度传感器 DS18B20 的程序可以参考本例。



第 15 章 步进电机的控制

步进电机是一种以脉冲信号控制转速的电机，很适合使用单片机来进行控制。在数控机床、医疗器械、仪器仪表、机器人以及其他自动设备中得到了广泛应用。我们使用的计算机外围的一些设备，如软驱、打印机、扫描仪等其运动部件的控制都采用了步进电机。

15.1 步进电机的工作原理

15.1.1 步进电机的种类

小型步进电机外形如图 15.1 所示。



图 15.1 步进电机

步进电机按绕在定子的线圈配置分类可分为 2 相、4 相、5 相等，如图 15.2 所示。

步进电机按外部引线可分为三线式、五线式、六线式等，但其控制方法均相同，均以脉冲信号进行驱动。

15.1.2 步进电机工作原理

步进电机，顾名思义，就是一步步走的电动机，所谓“步”指的是转动角度，一般每步为 1.8° ，若转一圈 360° ，需要 200 步才能完成。有的每步为 7.5° ，还有的每步为 18° ，转一圈只需 20 步。

步进电机每走一步，就要加一个脉冲信号，也称激磁信号。无脉冲信号输入时，转子保

持一定的位置，维持静止状态。

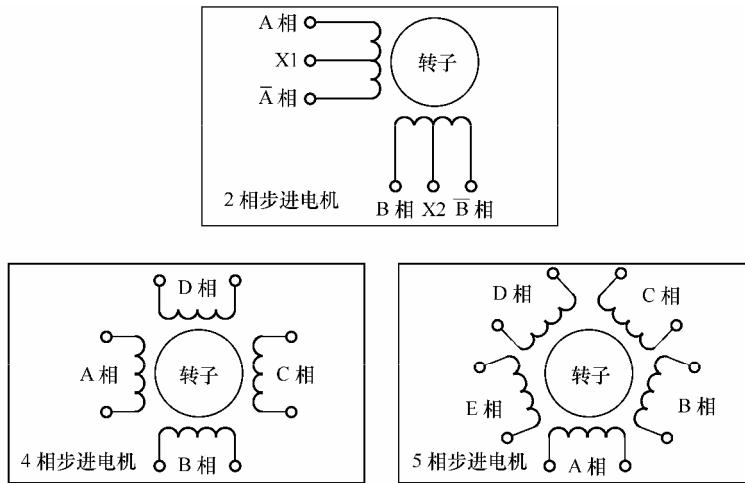


图 15.2 步进电机的种类

若加入适当的脉冲信号时，转子则会以一定的步数转动。如果加入连续的脉冲信号，步进电机就连续转动，转动的角度与脉冲频率成正比，正、反转可由脉冲的顺序来控制。

步进电机的激磁方式有 1 相激磁、2 相激磁和 1-2 相激磁。

- 1 相激磁法：在每一瞬间只有一个线圈导通，其他线圈在休息。其特点是激磁方法简单、消耗电力小、精确度良好。但是转矩小、振动较大，每送一次激磁信号可走 1.8° 。
- 2 相激磁法：在每一瞬间会有两个线圈同时导通，特点是转矩大、振动较小，每送一次激磁信号可走 1.8° 。
- 1-2 相激磁法：1 相与 2 相轮流交替导通，精确度提高，且运转平滑。但每送一激磁信号只走 0.9° ，又称为半步驱动。

1 相激磁、2 相激磁和 1-2 相激磁方式如表 15.1 所示。

表 15.1

3 种激磁方式

1 相激磁					2 相激磁					1-2 相激磁				
步	A	B	\bar{A}	\bar{B}	步	A	B	\bar{A}	\bar{B}	步	A	B	\bar{A}	\bar{B}
1	0	1	1	1	1	0	0	1	1	1	0	1	1	1
2	1	0	1	1	2	1	0	0	1	2	0	0	1	1
3	1	1	0	1	3	1	1	0	0	3	1	0	1	1
4	1	1	1	0	4	0	1	1	0	4	1	0	0	1
5	0	1	1	1	5	0	0	1	1	5	1	1	0	1
6	1	0	1	1	6	1	0	0	1	6	1	1	0	0
7	1	1	0	1	7	1	1	0	0	7	1	1	1	0
8	1	1	1	0	8	0	1	1	0	8	0	1	1	0

改变线圈激磁的顺序可以改变步进电机的转动方向。每送一次激磁信号后要经过一小段时间延时，让步进电机有足够的时间建立激场及转动。

15.1.3 小型步进电机驱动电路

单片机的输出电流太小，不能直接连接步进电机，需要加驱动电路。对于电流小于 0.5A 的步进电机，可以采用 ULN2003 类的驱动 IC。

图 15.3 所示为 2001/2002/2003/2004 系列驱动器引脚图，图左边 1~7 引脚为输入端，接单片机输出端，引脚 8 接地；右侧 10~16 引脚为输出端，接步进电机，引脚 9 接电源+5V，该驱动器可提供最高 0.5A 的电流。

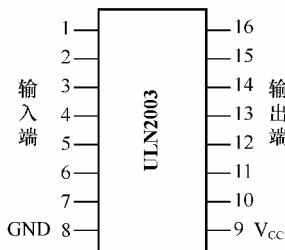


图 15.3 ULN2003 引脚图

15.2 步进电机正转

功能说明：由单片机的 P0.0~P0.3 端口来控制小型步进电机，步进电机每步为 18°。直接采用 ULN2003 驱动电路，以 1 相激磁法使步进电机正转一圈之后停下来。

15.2.1 硬件设计

电路设计如图 15.4 所示。

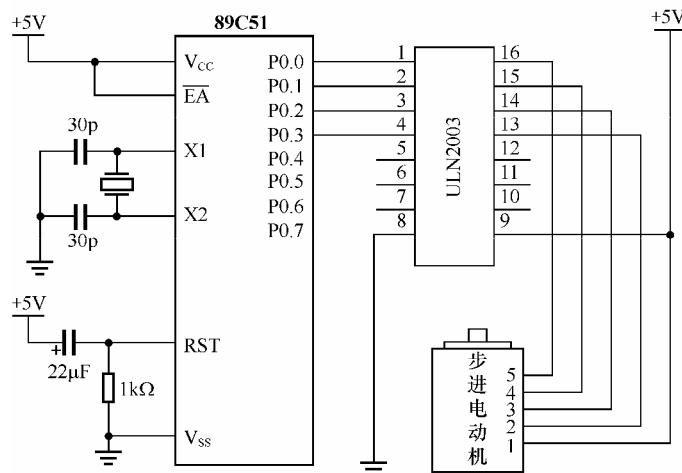


图 15.4 电路图

单片机的 P0.0~P0.3 端口直接与 ULN2003 驱动器的 1~4 输入引脚连接，ULN2003 的输出端连接小型 2 相 5 线步进电机。步进电机的线圈中心抽头 X1 与 X2 连接一起，接电源正极。

15.2.2 程序设计

该电路使用 2 相步进电机，采用 1 相激磁法，正转的激磁信号时序为 0FEH 0FDH 0FBH 0F7H，如表 15.2 所示。循环 5 次为 20 步，该步进电机每步为 18°，20 步为一圈。

表 15.2

步进电机正转时序

	步进	P0.3	P0.2	P0.1	P0.0
OFEH	1	1	1	1	0
OFDH	2	1	1	0	1
OFBH	3	1	0	1	1
OF7H	4	0	1	1	1

1. 流程图

程序流程如图 15.5 所示。

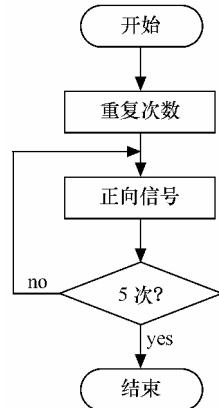


图 15.5 程序流程

2. 程序

汇编语言编写的步进电机正转源程序 DJ15-1.ASM 代码如下：

01	Z_M:	MOV	R0, #5	; 重复次数
02				
03	LOOP:	MOV	P0, #OFEH	; 第一步驱动信号
04		ACALL	DELAY	; 延时
05		MOV	P0, #OFDH	; 第二步驱动信号
06		ACALL	DELAY	; 延时
07		MOV	P0, #OFBH	; 第三步驱动信号
08		ACALL	DELAY	; 延时
09		MOV	P0, # OF7H	; 第四步驱动信号
10		ACALL	DELAY	; 延时
11				
12		DJNZ	R0, LOOP	; 未到 5 次，跳至 LOOP 循环
13				
14	DELAY:	MOV	R5, #50	; 延时子程序
15	DLY1:	MOV	R6, #100	

```

16 DLY2:    MOV      R7, #100
17        DJNZ    R7, $
18        DJNZ    R6, DLY2
19        DJNZ    R5, DLY1
20        RET
21        END          ;程序结束

```

15.2.3 代码详解

1. 标号说明

Z_M：正转主程序。

DELAY：延时子程序。

2. 寄存器使用分配情况

P0：特殊寄存器，对外是输入/输出端口。

R0：普通寄存器，在程序中作正转循环计数器。

R5 ~ R7：在延时程序中作计数器。

3. 程序分析解释

01：设定循环次数为 5 次。每循环一次为 4 步，经 5 次循环共运转 20 步，每步为 18°，20 步为 360°，正好运转一圈。

03：输出正转第一步驱动信号 0FEH (11111110B)。

04：调用延时程序，延时 1s。

05：输出正转第二步驱动信号 0FDH (11111101B)。

06：调用延时程序，延时 1s。

07：输出正转第三步驱动信号 0FBH (11111011B)。

08：调用延时程序，延时 1s。

09：输出正转第四步驱动信号 0F7H (11110111B)。

10：调用延时程序，延时 1s。

12：判断循环是否完成，当 R0 值不为 0 时，说明循环次数没完，程序会跳转到 LOOP 处继续循环。循环 5 次后，R0 值减为 0，程序向下运行 END 指令，程序结束。

14 ~ 20：延时子程序，延时为 1s。

21：程序结束。

15.2.4 模拟仿真

单击 WedWin 窗口上方菜单中的“调试”命令，在出现的下拉菜单中单击“开始调试”，使程序进入模拟仿真调试状态。再到窗口上方菜单中选择“查看”命令，在出现的下拉菜单中单击“寄存器”下的“特殊功能寄存器”选项，使之出现相应的观察窗口。再回到上方菜单中单击“外围部件”并选择“端口”，便出现 P0 ~ P3 观察窗口，如图 15.6 所示。

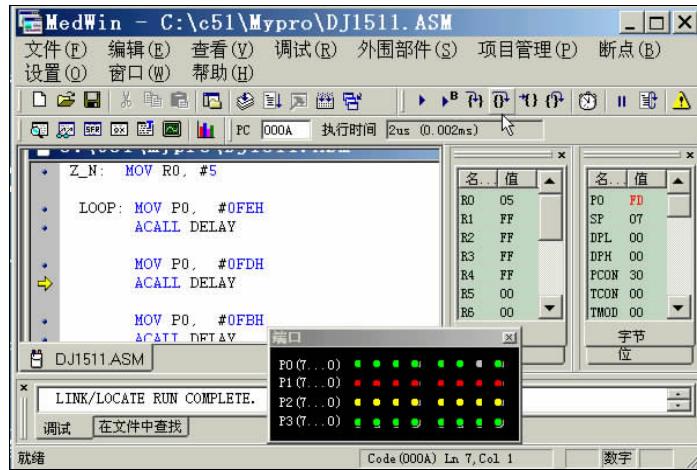


图 15.6 模拟仿真调试窗口

这时可以单击上方菜单中的“指令单步”图标，每击一次，程序运行一行。主要观察 R0、P0 值的变化以及端口 P0 出现的输出位的移动方向。输出位向左移动代表步进电机正转，单点移动说明是 1 相激磁方式。

15.2.5 实例测试

为了便于观察步进电机的转向和转速，可在电机的转动轴上套一个指针形的纸片如图 15.7 所示，测试时注意转向及转一圈的步数。

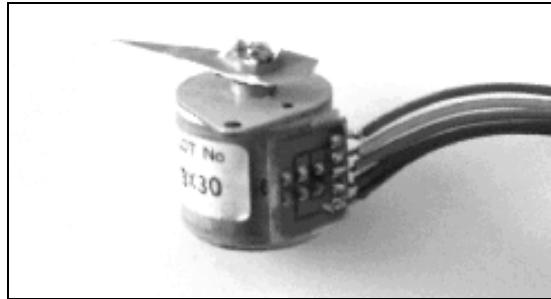


图 15.7 测试电机

15.2.6 经验总结

步进电机是依靠脉冲信号来驱动的，脉冲信号的时序决定电机的转向。二相步进电机的正转脉冲信号是由 4 步组成，在 1 相激磁方式下，其信号时序为 0FEH 0FDH 0FBH 0F7H。

15.3 步进电机反转

功能说明：由单片机的 P0.0 ~ P0.3 端口来控制小型步进电机，步进电机每步为 18°。直接

采用 ULN2003 驱动电路，以 1 相激磁法使步进电机反转两圈之后停下来。本节硬件设计与 15.2 相同。

15.3.1 程序设计

该电路使用 2 相步进电机，采用 1 相激磁法，反转的激磁信号时序为 0F7H 0FBH 0FDH 0FEH，如表 15.3 所示。循环 10 次为 40 步，每步为 18° ，40 步为两圈。

表 15.3

步进电机反转时序

	步进	P0.3	P0.2	P0.1	P0.0
OF7H	1	0	1	1	1
OFBH	2	1	0	1	1
OFDH	3	1	1	0	1
OFEH	4	1	1	1	0

1. 流程图

程序设计流程如图 15.8 所示。

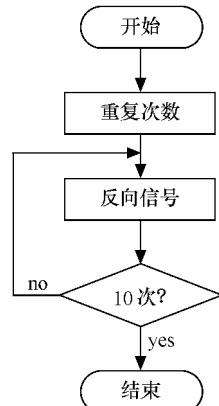


图 15.8 程序流程

2. 程序

汇编语言编写的步进电机反转源程序 DJ15-2.ASM，代码如下。

```

01 F_M:    MOV      R1, #10          ;重复次数
02
03 LOOP1:  MOV      P0, #0F7H        ;第一步驱动信号
04      ACALL   DELAY            ;延时
05      MOV      P0, #0FBH        ;第二步驱动信号
06      ACALL   DELAY            ;延时
07      MOV      P0, #0FDH        ;第三步驱动信号
08      ACALL   DELAY            ;延时
  
```

```

09      MOV      P0, #0FEH      ;第四步驱动信号
10      ACALL    DELAY      ;延时
11
12      DJNZ    R1, LOOP1      ;未到 10 次，跳转至 LOOP 循环
13
14  DELAY:  MOV      R5, #50      ;延时子程序
15  DLY1:   MOV      R6, #100
16  DLY2:   MOV      R7, #100
17      DJNZ    R7, $
18      DJNZ    R6, DLY2
19      DJNZ    R5, DLY1
20      RET
21      END      ;程序结束

```

15.3.2 代码详解

1. 标号说明

F_M：反转主程序。

DELAY：延时子程序。

2. 寄存器使用分配情况

P0：特殊寄存器，对外是输入/输出端口。

R1：普通寄存器，在程序中作反转循环计数器。

R5 ~ R7：在延时程序中作计数器。

3. 程序分析解释

01：设定循环次数为 10。每循环一次为 4 步，经 10 次循环共计转 40 步，每步为 18°，40 步为 720°，正好运转两圈。

03：输出反转第一步驱动信号 0F7H (11110111B)。

04：调延时程序，延时 1s。

05：输出反转第二步驱动信号 0FBH (11111011B)。

06：调延时程序，延时 1s。

07：输出反转第三步驱动信号 0FDH (11111101B)。

08：调延时程序，延时 1s。

09：输出反转第四步驱动信号 0FEH (11111110B)。

10：调延时程序，延时 1s。

12：判断循环是否完成，当 R1 值不为 0 时，说明循环次数没完，程序会跳转到 LOOP1 处继续循环。循环 10 次后，R1 值减为 0，程序向下运行 END 指令，程序结束。

14 ~ 20：延时子程序，延时时间为 1s。

21：程序结束。

15.3.3 模拟仿真

模拟仿真的方法与上节基本相同，注意端口 P0 出现的输出位的移动方向。输出位向右移动代表步进电机反转，单点移动说明是 1 相激磁方式。

15.3.4 实例测试

将写入电机反转程序的单片机插入实验板插座内，再检查一下电动机连接是否正常。为了便于观察步进电机的转向和转速，可在电机的转动轴上套一个指针形的纸片，并记住转动开始的位置，然后接通电源，观察步进电机的转向和运行步数，该程序使步进电机反向运行 40 步，反转两圈后停止。

15.3.5 经验总结

2 相步进电机的反转激磁信号也是由 4 步组成，在 1 相激磁方式下，其信号时序为 0F7H 0FBH 0FDH 0FEH。

15.4 步进电机转速控制

功能说明：由单片机的 P0.0 ~ P0.3 端口控制小型步进电机，步进电机每步为 18° 。直接采用 ULN2003 驱动电路，以 2 相激磁法使步进电机正转，20s 转 1 圈，每步中间延时 1s，之后，反转时加速度，20s 转 10 圈，停留 5s，不断循环。

本节硬件设计与 15.2 节相同。

15.4.1 程序设计

该程序由主程序及正转子程序和反转子程序所构成，子程序结构与第 6 章《跑马灯》中的灯左右移程序相似。步进电机的电路使用 2 相步进电机，采用 2 相激磁法，正转的激磁信号时序为 0FCH 0F9H 0F3H 0F6H，循环 20 次，转 1 圈。

每步间隔时间为 1s，即转 1 圈的时间为 20s；反转的激磁信号时序为 0F3H 0F9H 0FCH 0F6H，如表 15.4 所示。程序循环 200 次，每步间隔时间为 0.1s，转 10 圈用了 20s。

表 15.4

2 相激磁方式反转时序

	步进	P0.3	P0.2	P0.1	P0.0
OF7H	1	0	0	1	1
OFBH	2	1	0	0	1
OFDH	3	1	1	0	0
OFEH	4	0	1	1	0

1. 流程图

程序设计流程如图 15.9 所示。

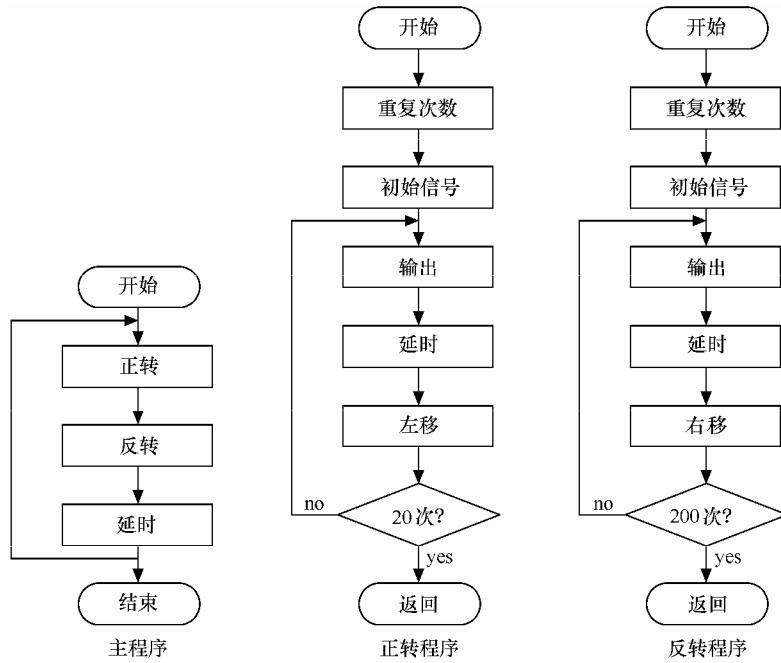


图 15.9 程序流程

2. 程序

汇编语言编写的步进电机转速控制源程序 DJ15-3.ASM 代码如下：

01	MAIN :	;	主程序
02	ACALL	Z_M	; 调用正转子程序
03	ACALL	F_M	; 调用反转子程序
04	MOV	R5, #250	; 将 R5 赋值为 250
05	ACALL	DELAY	; 调用延时子程序
06	JMP	MAIN	; 跳转至 MAIN 形成循环
07			
08	Z_M :	;	正转子程序
09	MOV	R0, #20	; 重复 20 次
10	MOV	A, #11001100B	; 左移初始值
11	LOOP : :	MOV P0, A	; 输出
12	MOV	R5, #50	; 将延时计数器 R5 赋值为 50
13	ACALL	DELAY	; 调用延时子程序延时 1s
14	RL	A	; 左移
15	DJNZ	R0, LOOP	; 判断是否循环 20 次
16	RET		; 子程序返回
17			
18	F_M :	;	反转子程序
19	MOV	R1, #200	; 重复 200 次

```

20      MOV      A, #00110011B ;右移初始值
21  LOOP1: MOV      P0, A      ;输出
22      MOV      R5, #5       ;将延时计数器 R5 赋值为 5
23      ACALL   DELAY      ;调用延时子程序延时 0.1s
24      RR      A          ;右移
25      DJNZ    R1,  LOOP1  ;判断是否循环 200 次
26      RET      R1        ;子程序返回
27
28  DELAY:
29  DLY1:  MOV      R6, #100
30  DLY2:  MOV      R7, #100
31      DJNZ    R7,  $      ;延时子程序
32      DJNZ    R6,  DLY2
33      DJNZ    R5,  DLY1
34      RET      R1        ;程序结束
35      END

```

15.4.2 代码详解

1. 标号说明

MAIN : 主程序。

Z_M : 反转子程序。

F_M : 正转子程序。

DELAY : 延时子程序。

2. 寄存器使用分配情况

P0 : 特殊寄存器 , 对外是输入/输出端口。

R0 : 普通寄存器 , 在程序中作正转循环计数器。

R1 : 普通寄存器 , 在程序中作反转循环计数器。

R5 ~ R7 : 在延时程序中作计数器。

3. 程序分析解释

01 : 主程序开始。

02 : 调用正转子程序 , 使步进电机运转 20 步 , 正转 1 圈。

03 : 调用反转子程序 , 使步进电机运转 200 步 , 正转 10 圈。

04 : R5 是延时子程序中的循环计数器 , 延时子程序的延时时间为 $20\text{ms} \times R5$, 所以只要对 R5 赋不同的值就会获得不同延时时间。

05 : 调用延时子程序 , 延时 5s。

06 : 使程序转移到标号 MAIN 处运行 , 不断循环。

08 : 正转子程序。

09 : 重复 20 次 , 即运行 20 步 , 转一圈。

10：左移初始值。2相激磁方式的左移初始值为11001100B，而不是11111100B，其目的是为了简化程序，在左移（或右移）时不必判断是否超出范围。

12：寄存器R5在延时子程序中作计数器使用，R5赋值为50，使延时时间为1s。

13：调用延时子程序，延时时间为1s。

14：激磁信号向左移动，使电机正向转动一步。

15：判断是否循环20次。每循环一次R0值减1，当R0值减为0时，说明程序已经重复执行了20遍，即电机走了20步。DJNZ指令使程序向下运行。

16：子程序返回。

18~26：步进电机反转子程序，程序工作原理与步进电机正转子程序结构基本相同，不同之处有如下。

- 激磁信号初始值不同，正转激磁信号初始值为11001100B，反转激磁信号初始值为00110011B。

- 信号的移动方向不同，正转时向左移，反转时向右移。

- 转动的步数不同，正转为20步1圈，反转为200步10圈。

- 每步间隔时间不同，正转间隔时间为1s，反转间隔时间为0.1s。

28~34：延时子程序。延时的时间为 $20\text{ms} \times R5$ ，调用延时子程序之前，必须先将R5赋值。本程序中先后将R5赋值3次，分别为250、50和5，其延时时间分别为5s、1s和0.1s。

35：程序结束。

15.4.3 模拟仿真

模拟仿真时，主要观察R0、R1和P0值的变化以及端口P0出现的输出位的移动方向。输出位向左移动代表步进电机正转，输出位向右移动代表步进电机反转。单点移动说明是1相激磁方式，双点移动说明是2相激磁方式。

由于该程序采用2相激磁方式，所以在模拟仿真时端口P0出现的是双位位移。先双位向左移动（正转），后双位向右移动（反转）。向右移动的速度比向左移动快。

15.4.4 实例测试

将写入电机程序的单片机插入实验板插座内，同时检查电动机连接是否正常，并在电动机的转动轴上做好标记，记住转动开始的位置，以便观察步进电机的转向和转速。

当接通电源后，会看到电动机先是正转运行20步，转1圈。每步间隔时间为1s，即正转1圈的时间为20s。之后电动机开始反转200步，转10圈。每步间隔时间为0.1s，即反转10圈用了20s。步进电机的反向转速为正向转速的10倍。

15.4.5 经验总结

- 2相激磁方式与1相激磁方式的转角是一样的，实际使用时，采用2相激磁方式要好于1相激磁方式。

- 激磁信号的时序决定转向。

- 激磁信号的频率与转数成正比，频率越高，转速越高；频率越低，转速越低。在本节程序中正转每步间隔时间为1s，反转每步间隔时间为0.1s，反转脉冲信号的频率是正转的10

倍，其电机的反向转速比正向转速也提高 10 倍。

15.5 开关控制步进电机正反转

功能说明：单片机的 P3.2 ~ P3.4 引脚分别接有按钮开关 K1、K2 和 K3，用来控制步进电机的转向。

开始供电时，步进电机停止。

按 K1 时，电动机正转；按 K2 时，电动机反转。

按 K3 时，电动机停止转动。

正转采用 1 相激磁方式，反转采用 1-2 相激磁方式。

15.5.1 硬件设计

电路设计如图 15.10 所示。K1、K2 和 K3 按钮开关分别接在单片机的 P3.2 ~ P3.4 引脚上，作为控制信号的输入端，输出端直接采用 ULN2003 驱动电路控制步进电机的转向。

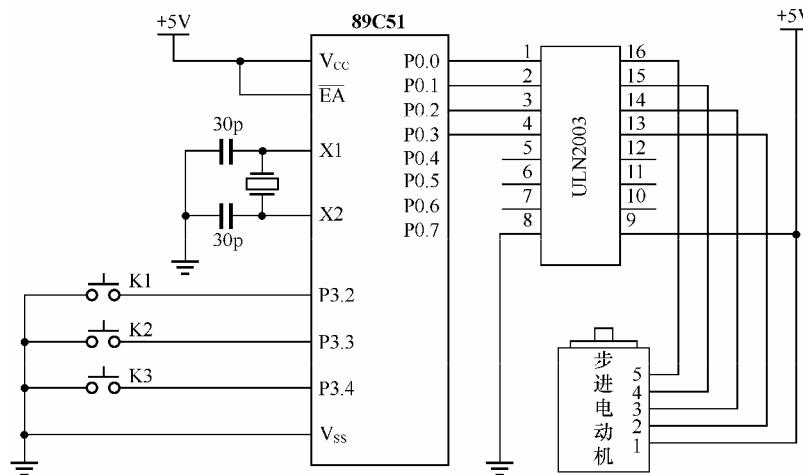


图 15.10 开关控制电动机电路

15.5.2 程序设计

编程采用制表的方法，步进电机正转采用 2 相激磁方式，时序如表 15.5 所示。

表 15.5 2 相激磁方式正转时序

	步进	P0.3	P0.2	P0.1	P0.0
OFCH	1	1	1	0	0
OF9H	2	1	0	0	1
OF3H	3	0	0	1	1
OF6H	4	0	1	1	0

步进电机反转采用 1-2 相激磁方式，时序如表 15.6 所示，

表 15.6 1-2 相激磁反转时序

	步进	P0.3	P0.2	P0.1	P0.0
OF7H	1	0	1	1	1
OF3H	2	0	0	1	1
OFBH	3	1	0	1	1
OF9H	4	1	0	0	1
OFDH	5	1	1	0	1
OFCH	6	1	1	0	0
OFEH	7	1	1	1	0
OF6H	8	0	1	1	0

1. 流程图

程序设计流程如图 15.11 所示。

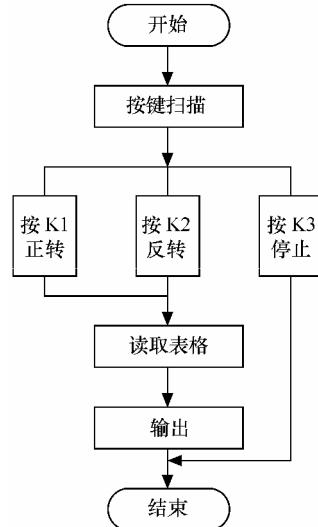


图 15.11 流程图

2. 程序

汇编语言编写的开关控制步进电机正反转源程序 DJ15-4.ASM 代码如下：

01	K1	EQU P3.2	; 设定 P3.2 以 K1 表示
02	K2	EQU P3.3	; 设定 P3.3 以 K2 表示
03	K3	EQU P3.4	; 设定 P3.4 以 K3 表示
04			
05	STOP:	MOV P0, #0FFH	; 步进电机停止

```

06  LOOP:   JNB      K1, Z_M2          ;是否按 K1, 是则正转
07      JNB      K2, F_M2          ;是否按 K2, 是则反转
08      JNB      K3, STOP1        ;是否按 K3, 是则停止
09      JMP      LOOP           ;跳转至 LOOP 处
10
11  STOP1:  ACALL    DELAY         ;按 K3 的消除抖动
12      JNB      K3, $           ;K3 放开否 ?
13      ACALL    DELAY         ;放开消除抖动
14      JMP      STOP          ;电机停止
15
16  Z_M2:   ACALL    DELAY         ;按 K1 的消除抖动
17      JNB      K1, $           ;K1 放开否 ?
18      ACALL    DELAY         ;放开消除抖动
19      JMP      Z_M            ;转至 Z_M 处
20
21  F_M2:   ACALL    DELAY         ;按 K2 的消除抖动
22      JNB      K2, $           ;K2 放开否 ?
23      ACALL    DELAY         ;放开消除抖动
24      JMP      F_M            ;转至 F_M 处, 循环
25 ; 正转子程序
26  Z_M:    MOV      R0, #00H       ;正转到 TABLE 取码指针初值
27  Z_M1:   MOV      A, R0          ;到 TABLE 取码
28      MOV      DPTR, #TABLE     ;存表
29      MOVC    A, @A+DPTR      ;取表代码
30      JZ      Z_M            ;是否取到结束码?
31      MOV      P0, A           ;输出至 P0, 正转
32      JNB      K3, STOP1        ;是否按 K3, 是则停止运转
33      JNB      K2, F_M2        ;是否按 K2, 是则反转
34      ACALL    DELAY         ;步进电机转速
35      INC      R0            ;取下一个码
36      JMP      Z_M1          ;转至 Z_M 处, 循环
37      RET
38 ; 反转子程序
39  F_M:    MOV      R0, #05        ;反转到 TABLE 取码指针初值
40  F_M1:   MOV      A, R0          ;到 TABLE 取码
41      MOV      DPTR, #TABLE     ;存表
42      MOVC    A, @A+DPTR      ;取表代码
43      JZ      F_M            ;是否取到结束码?
44      MOV      P0, A           ;输出至 P0, 反转
45      JNB      K3, STOP1        ;是否按 K3, 是则停止运转
46      JNB      K1, Z_M2        ;是否按 K1, 是则正转
47      ACALL    DELAY         ;步进电机转速
48      INC      R0            ;取下一个码

```

```

49      JMP      F_M1          ;转至 F_M1 处，循环
50      RET
51
52  DELAY: MOV      R6, #40      ;延时时间 20ms
53  D1:    MOV      R7, #248
54      DJNZ     R7, $
55      DJNZ     R6, D1
56      RET
57 ; 控制码表
58  TABLE:
59      DB      0FCH, 0F9H, 0F3H, 0F6H ;正转
60      DB      00H                  ;正转结束码
61      DB      0F7H, 0F3H, 0FBH, 0F9H ;反转
62      DB      0FDH, 0FCH, 0FEH, 0F6H
63      DB      00H                  ;反转结束码
64
65      END          ;程序结束

```

15.5.3 代码详解

1. 主要标号说明

LOOP：按键扫描。

STOP1：K3 键消除抖动。

Z_M2：K1 键消除抖动。

F_M2：K2 键消除抖动。

Z_M：反转子程序。

F_M：正转子程序。

DELAY：延时子程序。

TABLE：控制码表。

2. 寄存器使用分配情况

P0、P3：特殊寄存器，对外是输入/输出端口。

A：特殊寄存器，也称累加器。

R0：普通寄存器，在程序中存入 TABLE 表取码指针初值。

R6、R7：在延时程序中作计数器。

DPTR：为特殊寄存器，用于存表。

3. 程序分析解释

01~03：将 P3.2、P3.3、P3.4 引脚分别用 K1、K2、K3 表示。

05：使步进电机停止转动。

06~09：对按键扫描。当按键开关未按下时，相应引脚为高电平；当开关按下时，相

应引脚为低电平。所以通过循环检测按键开关所连接的引脚电平的高低，就可以判断出开关状态。

- 11：按 K3 时，调延时子程序，延时一小段时间来消除按键时的抖动。
- 12：按键未放开时，程序将在此等待，只有按键放开后，程序才会向下运行。
- 13：按键放开时，调延时子程序，延时一小段时间来消除按键放开时的抖动。
- 14：当确定 K3 按键被按下又被放开后，才认定是一次有效按键，程序将跳转到 STOP 处，执行停机语句，使电机停止转动。
- 16 ~ 19：与上述 11 ~ 14 行语句原理相同，是对 K1 键按下和放开时消除抖动的处理，并跳转到 Z_M 处，执行正转子程序。
- 21 ~ 24：对按 K2 键时消除抖动的处理，并跳转到 Z_M 处，执行反转子程序。
- 26：正转子程序开始，将 R0 赋值为 0，使取表指针指向表 TABLE 的第一个码位置。
- 27：将 R0 值送入累加器 A。
- 28：将编制的电机控制码表 TABLE 存入特殊寄存器 DPTR。
- 29：从特殊寄存器 DPTR 中取出第一个控制码。
- 30：检查是否取到结束码，结束码为 00H。如果取到结束码 00H，JZ 指令会使程序跳转到标号 Z_M 处，从第一个码开始读取；如果取到的不是结束码 00H，程序向下运行。
- 31：将取到的数据由 P0 端口输出。
- 32：检查是否按 K3，是则停止运转。
- 33：检查是否按 K2，是则反转。
- 34：调用延时子程序。此处调用延时子程序与前边调用的作用不同，前边调用延时子程序是为了消除按键的抖动。
此处调用延时子程序是电机运转两步之间的间隔时间，决定电机的转速。延时时间长，送入电机的脉冲信号频率低，电机转速慢；延时时间短，送入电机的脉冲信号频率高，电机转速快。所以在此处改变延时时间就可以改变电机转速。
- 35：使 R0 值加 1，取下一个码。
- 36：跳转到 Z_M 处，开始新的循环。
- 37：正转子程序返回。
- 39 ~ 50：反转子程序。其中，第 39 行语句将 R0 赋值为 5，是为了取码时从反转控制码中第一个码开始读取。
- 52 ~ 56：延时子程序，延时的时间为 20ms。
- 58 ~ 63：步进电动机激磁信号编码表。其中，第 59 行为电机正转 2 相激磁码；第 60 行为正转结束码；第 61、62 行为反转 1-2 激磁码；第 63 行为反转结束码。
- 65：程序结束。

4. 至本章用过的指令归类

- 数据传送类指令：MOV、MOVC、PUSH、POP。
- 算术运算类指令：INC、DIV。
- 逻辑运算及位移类指令：RL、RR、RLC、RRC、CLR、ANL、ORL、XRL。
- 控制转移类指令：JMP、DJNZ、ACALL、RET、CJNE、RETI、JZ。

- 位操作类指令：CPL、CLR、SETB、JB、JNB、JBC、JNC、JC。
- 伪指令：END、DB、ORG、EQU。

15.5.4 模拟仿真

在模拟仿真时，程序会在 06~09 行语句之间循环进行按键扫描。这时可以使用模拟开关状态语句，如在 05 与 06 行之间使用“clr k1”语句模拟开关被按下状态，如图 15.12 所示。此时程序才能到 Z_M2 处运行。在第 16 与 17 语句之间增加模拟开关放开语句，如增加“setb k1”语句，这时程序才能进入“Z_M”正转子程序中运行。在模拟 K2 及 K3 开关按下或放开时都可以采用此办法，以便观察程序在模拟仿真中运行情况。



图 15.12 模拟仿真

15.5.5 实例测试

测试该程序时，要在电动机的转动轴上做好标记，记住转动开始的位置，以便观察步进电动机的转向和转速。之后，将写入电动机程序的单片机插入实验板插座内，接通实验板电源。

开始供电时，步进电机停止状态。按 K1 时电动机正转，按 K2 时，电动机反转，反转的速度要比正转慢。任何时候按 K3 按钮，电动机都会停止转动。

15.5.6 经验总结

本程序是通过 K1、K2 和 K3 3 个按钮开关控制步进电机转动和改变转向。电动机使用 1-2 相激磁，编程时采用制表的方法。

本节程序正转与反转的脉冲信号频率是相同的，但由于使用激磁方式不一样，反转使用了 1-2 相激磁法，故反转速度为正转的一半。



5.1 单片机 C 语言基础篇

C 语言是一种使用非常方便的高级语言。在单片机的开发应用中，除了使用汇编语言外，也逐渐引入了 C 语言。本篇将初步介绍 C 语言在单片机开发中的运用，并对 C 语言程序的开发软件 Keil C51 的使用进行详细说明。



第 16 章 单片机 C 语言入门

C51 是 80C51 单片机的 C 语言，除了遵循一般 C 语言的规则外，还有其自身的特点。

16.1 C 语言与 C51

16.1.1 C 语言与 C51

C 语言是一种使用非常方便的高级语言。所以，在单片机的开发应用中，除了使用汇编语言外，也逐渐引入了 C 语言。早在 1985 年便出现了 80C51 单片机的 C 语言，简称 C51。

单片机 C 语除了遵循一般 C 语言的规则外，还有其自身的特点，例如，增加了位变量数据类型（如 bit、sbit）中断服务函数（如 interrupt n），对 80C51 单片机特殊功能寄存器的定义是 C51 所特有的，是对标准 C 语言的扩展。

16.1.2 C 语言编程的优点

使用 C 语言与使用汇编语言相比，具有如下优点。

- 不需要掌握 8051 系列单片机的指令集，只要了解单片机内部特殊功能寄存器的用途即可。
- 编程人员不必考虑寄存器的分配和寻址方式等细节，这些都由编译器自动进行管理。
- 利于结构化编程，易于维护。
- 可使用与人的思维更相近的关键字和操作函数。
- 由于 C 语言可实现模块化编程技术，并与硬件无关，这样可以将大量的例程直接调用，节省开发时间。
- 程序的开发和调试的时间大大缩短，提高了效率。

16.1.3 C 语言和汇编语言混合编程

C 语言编程有明显的优势，但在有些场合下不适合使用 C 语言编程，例如，在对时序要求比较严格的场合用 C 语言编程比较困难，这时可以使用汇编语言。C 语言可以和汇编语言混合编程。

在实际编程中常常以 C 语言为主，汇编语言为辅，充分发挥各自的优势。

16.2 学习 C51 的准备工作

学习单片机 C 语言，首先要选择相应的单片机 C 语言编译器，即单片机的 C 语言开发集成系统软件，另外需要一台普通计算机、51 编程器及实验板。

16.2.1 计算机

C 语言对计算机的要求不高，能正常运行 Windows 操作系统的计算机即可。

16.2.2 51 单片机 C 语言编译器

8051 单片机的 C 语言编译器是一个应用软件，其作用是把用 C 语言编写的程序编译成 8051 单片机能够识别和执行的目标代码，即生成后缀为.HEX 的文件。将此文件通过 51 编程器写入单片机后，单片机就可以按照 C 语言编写的程序工作了。

目前单片机 C 语言编译器的功能不仅如此，而是一个单片机开发的集成系统，一般都具有工程建立和管理、编译、连接、目标代码生成、软件仿真及硬件仿真等完整的开发功能。

单片机 C 语言编译器有很多种，目前应用最广泛的是德国 Keil 公司出品的 Keil C51 编译器。它几乎支持所有 51 系列单片机的 C 语言和汇编语言编程，并具有软件仿真和硬件仿真功能，是一个优秀的单片机集成开发软件，其演示版可以到 <http://www.keil.com> 网站免费下载。

16.2.3 51 编程器和实验板

单片机 C 语言源程序经 Keil C51 编译、连接后产生可执行文件(扩展名为.HEX 的文件)，该文件与使用汇编语言编译后产生的.HEX 文件一样，需要使用 51 编程器才能写入到单片机内。使用 C 语言进行单片机开发时所用的编程器和实验板与使用汇编语言进行单片机开发时一样，使用方法也相同。

16.3 单片机 C 语言程序开发流程

单片机 C 语言程序开发流程与使用汇编语言基本相同，其流程如图 16.1 所示。

Keil C51 单片机开发集成系统软件可以完成 C 语言程序开发。

首先用 Keil C51 自带编辑器编写源程序，源程序文件名的后缀为.C；然后用 C51 编译器进行编译，生成扩展名为.OBJ 的文件；再用 BL51 连接器/定位器进行连接、定位，生成扩展名为.HEX 的文件，然后就可以通过 51 编程器写入单片机内。

在写入单片机之前还可以进行软件仿真或硬件仿真。

C 语言程序开发流程与汇编语言程序开发流程基本相同，区别有以下两点。

- 单片机是使用 C 语言开发，必须选择具有 C 语言编译能力的开发集成系统软件，如

Keil C51 单片机开发集成系统，既可以完成 C 语言的程序开发，也可以进行汇编语言的程序开发。

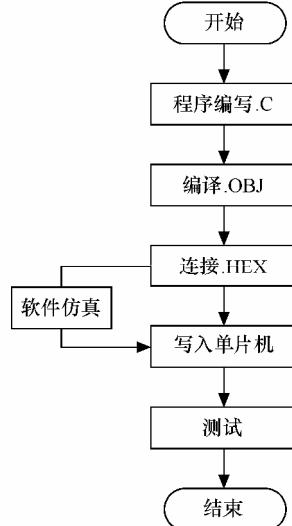


图 16.1 单片机 C 语言程序开发流程

- 编写 C 语言源程序时，文件名的后缀为.C。Keil C51 通过检测源程序后缀来确定使用不同的编译器。如果检测到源程序后缀是.ASM，就自动使用 A51 汇编器；如果检测到源程序后缀是.C，就自动使用 C51 编译器。

16.4 单片机 C 语言入门实例

下面介绍一个简单的单片机 C 语言编程实例，使读者初步了解单片机 C 编程的特点。

单片机 P1 端口接 8 个发光二极管，电路如图 16.2 所示。程序的功能是使中间 4 个（P1.2、P1.3、P1.4、P1.5 位）发光二极管点亮。

16.4.1 程序工作原理

只要将数据 11000011B 送到 P1 输出，就能使 P1 端口中间 4 个发光二极管点亮。数据 0xc3（11000011B）送入 P1，使 P1 中的 P1.2～P1.5 位设置为 0；P1.0、P1.1、P1.6、P1.7 位设置为 1。低电平使发光二极管导通发光，高电平使发光二极管截止。

将数据 11000011B（0xc3）送到 P1 使用汇编语言语句为：

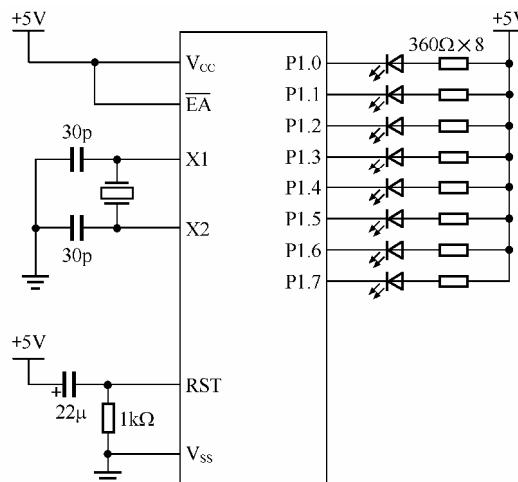


图 16.2 单片机 P1 端口输出电路

```
MOV P1, #11000011B
```

使用 C 语言语句为：

```
P1=0xc3
```

其中，11000011B 为二进制数，0xc3 为二进制数 11000011B 对应的十六进制数。

16.4.2 源程序

C 语言编写的源程序 Test 01.C 代码如下：

```
01  #include<reg51.h>          /* 头文件 */
02  main( )                     /* 主函数 */
03  {
04      P1=0xc3 ;              /* 从 P1 输出 */
05 }
```

16.4.3 程序说明

该程序是由主函数和头文件两部分组成。其中，主函数是 C 语言程序开始执行的地方，不可缺少；头文件用来定义 I/O 地址、参数、符号等。

01：头文件。

02：主函数，主函数名为 main。

03、05：主函数的内容用大括号{}括起来。

04：将数据传送到 P1 并输出。语句中“=”为赋值符号，与汇编语言的指令 MOV 作用相同；“；”为语句结束符号；“/*”和“*/”之间括起来的内容为注释信息。

上述源程序 Test 01.C 经 C 语言 Keil C51 编译、连接生成目标代码 Test01.HEX 后，就可以通过编程器写入单片机，然后，再把单片机插入实验板，就会看到 P1 端口中间 4 个发光二极管被点亮。

16.5 单片机 C 语言编程特点

本节对单片机 C 语言与汇编语言编程进行对比分析，使读者进一步理解和掌握 C 语言的特点。

在单片机电路图 16.2 中，使中间 4 个发光二极管与两边 P1.0、P1.1、P1.6、P1.7 位的发光二极管交替亮起，交替间隔时间为 1s。

16.5.1 程序工作原理

程序的工作原理如图 16.3 所示。

开始时，将数据 0xc3（11000011B）送入 P1，使 P1.2=P1.3=P1.4=P1.5=0，低电平 0 所对应的发光二极管因导通而发光；而 P1.0=P1.1=P1.6=P1.7=1，高电平 1 使发光二极管截止而熄灭。灯点亮后，必须要保持一段时间才看得到，所以要有延时程序。

P1 端口	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
数据 0xc3	1	1	0	0	0	0	1	1
数据 0x3c	0	0	1	1	1	1	0	0

图 16.3 将数据送入 P1 示意图

经过延时一段时间后再将数据 0x3c (00111100B) 送入 P1 , 使 P1.2=P1.3=P1.4=P1.5=1 ; P1.0=P1.1=P1.6=P1.7=0。此时 , 中间 4 个发光二极管因截止而熄灭 ; 两边的发光二极管因导通而发光 , 这样不间断地循环。

将数据送入 P1 输出 , 汇编语言使用 MOV 指令 , 语句为 :

```
MOV P1, #11000011B
MOV P1, # 00111100B
```

C 语言使用 “ = ” 赋值符号 , 语句为 :

```
P1=0xc3;
P1=0x3c;
```

汇编语言使用 ACALL 指令调用延时子程序 , 语句为 :

```
ACALL DELAY
```

C 语言实现办法是将延时函数写入主函数中 , 语句为 :

```
delay( );
```

下面是使用汇编语言与 C 语言分别进行编程。注意两种语言的程序结构及书写特点。

16.5.2 用两种语言编写

1. 使用汇编语言编写

汇编语言编写的源程序 Test02a.ASM 代码如下 :

```
START:    MOV      P1, #11000011B      ; 将数据送到 P1 并输出
          ACALL    DELAY        ; 调用延时子程序
          MOV      P1, # 00111100B     ; 将 A 数据送到 P1 并输出
          ACALL    DELAY        ; 调用延时子程序
          JMP      START       ; 跳转移到标号 START 处运行

DELAY:    MOV      R5, #25        ; 延时子程序
DLY1:    MOV      R6, #100
DLY2:    MOV      R7, #100
          DJNZ    R7, $           ; 
          DJNZ    R6, DLY2
```

```

DJNZ      R5, DLY1
RET          ;子程序返回
END          ;程序结束

```

2. 使用 C 语言编写

C 语言编写的源程序 Test02b.C 代码如下：

```

#include<reg51.h>           /* 头文件 */
void delay( );               /* 声明 delay( )延时函数 */

/* ----- 函数 ----- */
void delay( )                /* 延时子函数 */
{
    unsigned int i;           /* 声明无符号整型变量 i */
    for (i=0;i<50000;i++)    /* 循环延时语句 */
}

/* ----- 主函数 ----- */
void main( )                 /* 声明主函数 main( ) */
{
    while(1)                  /*无限循环 */
    {
        P1=0xc3;              /* 点亮中间 , 0xc3=11000011 */
        delay( );              /* 调用延时函数 delay( ) */
        P1=0x3c;              /* 点亮两边 , 0x3c=00111100 */
        delay( );              /* 调用延时函数 delay( ) */
    }
}

```

16.5.3 C 语言程序编写特点

C 语言程序与汇编语言程序从编写特点上比较，主要有以下 6 点不同。

- C 语言程序中的主函数是汇编程序中的主程序；C 语言程序中的函数是汇编程序中的子程序。程序运行都是从主函数或主程序开始，并终止于主函数或主程序中的最后一条语句。

但是在编写方面，汇编程序中的主程序必须编写在整个程序的最前面，因为汇编程序运行时是从整个程序中的第一行开始；而 C 语言程序中的主函数可以放在程序的前面，也可放在后面或其他位置，无论主函数在什么位置，程序运行时都会先自动找到主函数，并从主函数中的第一条语句开始执行。

- 编写 C 语言程序一般使用小写英文字母，C 语言的关键字均为小写英文字母，也可以使用大写英文字母，但大写字母一般都有特殊意义。
- C 语言严格区分子母大小写，也就是说 abc、Abc、ABC 是 3 个不同的名称，而汇编语言不区分子母大小写，编程时大小字母可以混用。

- C 语言不使用行号，一行可以写多条语句，但每一条语句最后必须有一个“ ;”作为结尾，而汇编语言一行就是一条语句。
- C 语言每一个独立完整的程序单元都由一对大括号括起来，大括号必须成对使用。
- C 语言的程序注释信息需要使用 “/*” 和 “*/” 括起来，如/* 头文件 */，或是用双斜杠，如 “//头文件”；而汇编程序语句的注释信息使用一个分号，如 “ ; 延时程序 ”。

16.6 单片机 C 程序的基本结构

单片机 C 语言程序一般由主函数、函数和头文件 3 部分组成。

16.6.1 主函数

主函数，即主程序，是单片机 C 语言程序执行的开始，不可缺少。主函数以 main 为其函数名称，例如：

```
main ( )
{
    C 语言语句 ;
}
```

单片机 C 语言主函数是一个特殊的函数，每个程序必须有而只有一个主函数。单片机 C 语言程序运行时都是由主函数 main () 开始的，主函数可以调用其他子函数，调用完毕后回到主函数，在主函数中结束整个程序的运行。

主函数内容用大括号 { } 括起来，括号内为单片机 C 语言程序语句，每行程序语句结束加 “ ; ”。

16.6.2 函数

函数，即子程序，是指除了主函数之外的各个函数。函数可以命名为各种名称，但不可与 C 语言保留字相同。

函数与主函数的格式是一样的，函数内容用大括号 { } 括起来，括号内为单片机 C 语言程序语句、每行程序语句结束加 “ ; ”。

16.6.3 头文件

头文件用来定义 I/O 地址、参数、符号。使用时通过 include 指令加载，将头文件包含在所编写的单片机 C 程序中，这样在编写单片机 C 程序时，就可以不需要考虑单片机内部的存储器分配等问题了，例如：

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

例中 reg51.h 为 51 单片机的头文件，使用时需要用 include 指令，并将头文件用括号 “ < > ” 括起来。

简单的单片机 C 语言程序（如 Test01.C），只包含主程序和程序库所载入的头文件。

16.7 C51 数据类型、常量与变量

单片机的基本功能是进行数据处理，数据在进行处理时需要先存放到单片机的存储器中。所以编写程序时对使用的常量与变量都要先声明数据类型，以便把不同的数据类型定位在 51 单片机的不同存储区中。

16.7.1 C51 的数据类型

数据类型是用来表示数据存储方式及所代表的数值范围的。C51 的数据类型与一般 C 语言的数据类型大多相同，但也有其扩展的数据类型。

1. 基本数据类型

基本数据类型有 char、int、long、float 等。

- char：字符型，代表存放 8 位数据。
- int：整型，代表存放 16 位数据。
- long：长整型，代表存放 32 位数据。
- float：实型（单精度），代表存放 32 位数据。

C51 语言常用的基本数据类型主要是 char（单字节字符型）和 int（双字节整型）两种，这两种数据类型对数据表示范围不同，处理速度也不相同。

char 类型可以表示 0 ~ 255 或 -128 ~ 127 之内的整数；int 类型则可以表示 0 ~ 65535 或 -32768 ~ 32767 之内的整数。

51 单片机的 CPU 是 8 位字长的，所以处理 char 类型的数据速度最快，而处理 16 位的 int 类型数据则要慢得多。

2. C51 扩展的数据类型

C51 相对于标准 C 语言扩展的数据类型主要有 bit、sbit、sfr、sfr16 等。

- bit：位，其值为 0 或 1。
- sbit：从字节中声明的位变量，其值为 0 或 1。
- sfr：特殊功能寄存器，sfr 字节地址为 0 ~ 255。
- sfr16：特殊功能寄存器，只是 sfr 字节地址为 0 ~ 65535。

3. 基本类型修饰符

基本类型修饰符放在基本数据类型的前边，其作用是改变基本数据类型的使用范围。

- signed：表示带符号。
- unsigned：表示不带符号。
- short：短。未加入 short 的变量为 short。
- long：长。其数据长度为原数据类型的 2 倍。

例如：“signed int i;”，声明 i 为带符号整型变量。其变量 i 的数值范围为 -32768 ~

+32768；若将 int 前加上修饰符 unsigned，写为“unsigned int i;”时，则是声明 i 为无符号整数变量，其变量 i 的数值范围为 0 ~ 65535。

基本类型 int 前不带修饰符时，如“int i;”，与“signed int i;”的数值范围相同。

C51 语言数据类型如表 16.1 所示。

表 16.1 C51 语言数据类型表

数据类型	位数	所占内存字节	数值范围
bit	1		0 或 1
sbit	1		0 或 1
sfr	8	1	0 ~ 255
sfr16	16	2	0 ~ 65535
char	8	1	0 ~ 255
signed char	8	1	-128 ~ +127
unsigned char	8	1	0 ~ 255
int	16	2	-32768 ~ +32768
signed int	16	2	-32768 ~ +32768
unsigned int	16	2	0 ~ 65535
short int	16	2	-32768 ~ +32768
signed short int	16	2	0 ~ 65535
unsigned short int	16	2	-32768 ~ +32768
long int	32	4	-2147483648 ~ +2147483647
signed long int	32	4	-2147483648 ~ +2147483647
unsigned long int	32	4	0 ~ 4294967296
float	32	4	+1.18E-38 ~ +3.39E+38

16.7.2 常量

常量是在程序执行中其值不能改变的量，常量的数据类型有整型、字符型和字符串型等。

整型常量就是整型常数，通常采用十进制表示，如 0、1、23 等。在表示十六进制数时，要在开头写上“0x”作为前缀，后面是要表示的十六进制数字 0 ~ 9，超过的使用字母 a ~ f 表示，也就是对应十进制的 10 ~ 15。例如，14 在十进制中表示为 14；在十六进制中表示为 0xe。

字符型常量的表示方法，是将字符使用单引号括起来，例如，‘a’、‘b’、‘x’ 等。

字符串型常量是由双引号“”内的字符组成，如“ABCD”、“\$1234”等都是字符串常量。当双引号内的字符个数为 0 时，称为空字符串。书写时要特别注意单引号与双引号的使用，如‘a’与“a”是不一样的，‘a’是字符常量，“a”是字符串常量。

对于不可显示的控制字符，可以在该字符前面加一个反斜杠“\”组成专用转义字符。利用转义字符可以完成一些特殊功能和输出时的格式控制。例如，转义字符‘\0’可以作为一

个字符串结束标志位，一旦遇到字符‘\0’就表示字符串结束。

16.7.3 变量

变量是一种在程序执行过程中其值能不断变化的量。C语言中的每个变量都必须有一个标志符作为它的变量名。在使用一个变量之前，必须先对该变量进行定义，指出该变量的数据类型。

同时，程序中所使用的变量，根据所使用的效果范围不同，有局部变量和全局变量之分。

1. 变量定义的格式

变量定义的格式如下：

变量数据类型 变量名称

如，`unsigned int i;`

声明*i*为无符号整数变量。其中*i*为变量名称，变量名称可以使用大写英文字母(A~Z)或小写英文字母(a~z)，大写与小写的变量，代表不同变量。变量名称*i*前边的“`unsigned int`”是定义变量*i*的数据类型。

2. 局部变量

局部变量是指变量只有在它所声明的函数内才有效，例如：

```
void scan_k1( )
{
    unsigned int i; /* 局部变量 i 声明 */
    .
    .
}


```

程序中变量*i*为局部变量，其只有在`scan_k1()`函数内才有效。

3. 全局变量

全局变量的有效范围是在整个文件内(*.c)，文件内可能包含很多的函数，全局变量在各函数内的任何处均有效，例如：

```
void scan_k1( ); /* scan_k1( ) 函数声明 */
unsigned char set; /* 全局变量 set 声明 */
void main( )
{
    while(1)
    {
        if(P3_2==0)scan_k1();
        switch(set)
```

```

{
...
}
}

```

在程序中，变量“set”放在整个程序的前边进行声明，成为全局变量。它不仅在主函数main()内有效，而且在函数scan_k1()内也有效。

16.7.4 数组

数组是一种结构化的数据类型。它是将许多相同数据类型的变量集合起来，以一个名称来代表，称为数组名。数组中元素是按顺序存放在一个连续的存储空间中，即最低的地址存放第一个元素，最高的地址存放最后的一个元素，数组中元素的顺序用下标表示，下标放在[]方括号内。数组必须先定义，然后才能使用。

一维数组定义的一般形式：

数据类型 数组名 [元素个数] ;

例如：int a [8]

是定义一个数组，数组名为 a，数组包括 8 个整型的元素。

C 语言中数组的下标是从 0 开始的，因此对于数组 int a [8]来说，其中 8 个元素是 a[0] ~ a[7]，不存在元素 a[8]，这一点在引入数组元素时加以注意。

在编程中，数组的一个很有用的用途就是查表。要求单片机处理一些复杂的控制动作时，编程时一般采用查表的方法，先将控制码建成一个表存入程序存储器中，再根据需要在表中将控制码按顺序读取出，其中所说的表就是数组。

16.8 C51 常用的运算符

运算符是完成某种特定运算的符号，利用这些运算符可以组成各种表达式和语句。C51 语言编程中常用的基本运算符有。

- 赋值运算符：=
- 增量和减量运算符：++、--
- 关系运算符：<、!=、==、<=
- 逻辑运算符：||、!
- 位运算符：>>、<<、~、|

16.8.1 赋值运算符

符号“=”在 C 语言中是赋值运算符。赋值运算符的作用是将一个数据值赋给一个变量，如，“P1=0x00”是将数据 0x00 传送到 P1。

赋值运算符“=”的作用与汇编语言“MOV”作用一样。例如，C 语言中的“P1=0x00；”语句与汇编语言中的“MOV P1,#00H”语句作用是相同的，都是将数据“0”送入单片机

P1 端口输出。

16.8.2 增量和减量运算符

C51 语言中除了基本的加、减、乘、除运算符之外，还提供一种特殊运算符。

`++`：增量运算符。

`--`：减量运算符。

增量和减量运算符的作用是对运算对象作加 1 或减 1 运算，例如：

`a=5, a++, 结果 a=6。`

`++`或`--`可以放在操作数（必须是变量）的右侧，如`i++`，也可以放在操作数的左侧，如`++i`。但是由于运算符`++`所处位置不同，使变量`i`加 1 的运算过程也不同。`++i`（或`-i`）是先执行`i+1`（或`i-1`）操作，再使用`i`的值；而`i++`（或`i--`）则是先使用`i`的值，再执行`i+1`（或`i-1`）操作，例如：

```
i=7;
j = ++i           /* j 值为 8, i 值也是为 8 */
j = i++           /* j 值为 7, i 值为 8 */
```

因为执行`i++`后，是先将操作数`i=7`记录下来，故`j`值为 7 之后再将`i`值加 1，所以`i`值为 8。

16.8.3 关系运算符

C51 语言中有 6 种关系运算符，如表 16.2 所示。

表 16.2

关系运算符

运算符符号	例 子	说 明
<code>></code>	<code>a > b</code>	<code>a</code> 大于 <code>b</code>
<code>>=</code>	<code>a >= b</code>	<code>a</code> 大于或等于 <code>b</code>
<code><</code>	<code>a < b</code>	<code>a</code> 小于 <code>b</code>
<code><=</code>	<code>a <= b</code>	<code>a</code> 小于或等于 <code>b</code>
<code>==</code>	<code>a == b</code>	<code>a</code> 等于 <code>b</code>
<code>!=</code>	<code>a != b</code>	<code>a</code> 不等于 <code>b</code>

关系运算符实际上是一种比较运算，即将两个值进行比较，判断其比较的结果是否符合给定的条件。例如，`a < b` 是一个关系表达式，小于号“`<`”是一个关系运算符，如果`a`的值为 1，`b`的值为 2，则满足给定的`a < b` 条件，因此关系表达的值为“真”，即条件满足；如果`a`的值为 3，不满足给定的`a < b` 条件，则称关系表达的值为“假”。

16.8.4 逻辑运算符

C51 语言中有 3 种逻辑运算符，如表 16.3 所示。

表 16.3

3 种逻辑运算符

运算符符号	例 子	说 明
&&	a&&b	逻辑与 (AND)
	a b	逻辑或 (OR)
!	! a	逻辑非 (NOT)

逻辑运算符是用来求某个条件式的逻辑值，将非 0 看作逻辑 1(真)，0 看作逻辑 0(假)。逻辑运算的真值表如 16.4 所示。

表 16.4

逻辑运算的真值表

a	b	!a	!b	a&&b	a b
真 (1)	真 (1)	假 (0)	假 (0)	真 (1)	真 (1)
真 (1)	假 (0)	假 (0)	真 (1)	假 (0)	真 (1)
假 (0)	真 (1)	真 (1)	假 (0)	假 (0)	真 (1)
假 (0)	假 (0)	真 (1)	真 (1)	假 (0)	假 (0)

1. 逻辑与运算

进行 $a \&\& b$ 逻辑与运算时，只有 a、b 值都为真（非 0 值），运算结果为真（值为 1），否则，运算结果为假（值为 0）。例如：若 a 的值是 2，b 的值是 0，因为 a 的值不为 0，所以为真；而 b 的值为 0，所以为假，真和假做逻辑与运算，结果为假，即该逻辑运算结果的值为 0。

2. 逻辑或运算

进行 $a|b$ 逻辑或运算时，只要 a、b 中有一个为真，逻辑运算结果便为真（值为 1），只有 a、b 都为假，运算结果才为假（值为 0）。

例如：在逻辑表达式中 a 值为 1，b 值为 2，c 值为 3。当进行 $(a < c) | (b > c)$ 逻辑或运算时，因为 $(a < c)$ 为真，所以运算结果为真。若是 $(a > c) | (b > c)$ 运算时，因为 $(a > c)$ 和 $(b > c)$ 都为假，则运算结果为假。

3. 逻辑非运算

进行 $!a$ 逻辑非运算时，若 a 值为真，则经逻辑非运算的结果就为假；若 a 值为假，逻辑非运算的结果就为真，即“假 (0)”变为“真 (1)”，“真 (1)”变为“假 (0)”。

16.8.5 位运算符

在 C51 语言中，位操作运算使用比较多，位运算符有 6 种，如表 16.5 所示。

表 16.5

位运算符表

运算符符号	例 子	说 明
&	A&b	将 a 与 b 各位作 AND 运算

续表

运算符符号	例 子	说 明
	a b	将 a 与 b 各位作 OR 运算
^	a^b	将 a 与 b 各位作 XOR 运算
~	~a	将 a 作反相
>>	a>>b	将 a 的值右移 b 个位
<<	a<<b	将 a 的值左移 b 个位

位运算符会将各变量或常量的每一个位 (bit) 做逻辑运算，并将结果写入某变量。

1. 按位与运算符&

进行按位与运算时，参加运算的两个操作数，如果两个相应的位都为 1，则该位的结果值为 1，否则为 0。

按位与运算的主要作用：将不需要的位清 0。

2. 按位或运算符|

进行按位或运算时，参加运算的两个操作数，如果在两个相应的位中只要有一个为 1，则该位的结果值为 1，否则为 0。

按位或运算的主要作用：将指定的位置 1。

3. 异或运算符^

异或运算又称 XOR 运算，它的运算规则是：参加运算的两个操作数，如果两个相应的位若相同，则该位的结果值为 0，否则为 1。

按位异或运算的主要作用如下。

(1) 与 1 异或，使特定位翻转

任何数与 1 异或都会变成相反数，利用这一点，可以把一个操作数的指定位翻转而不影响其他位的值。

(2) 与 0 异或，使指定位保留原值

因为任何数与 0 异或都保持不变，利用这一点可以把一个操作数的指定位保留原值。

4. 位取反运算符 ~

进行按位取反运算时，将操作数的各位分别翻转为相反值。

即： $\sim 1=0$ ； $\sim 0=1$ 。

5. 位左移运算符<<

左移运算符的作用是将一个数的各位顺序左移若干位。在左移运算中，高位移出舍弃不用，低位补 0。

例如：若 $a=11110101B$ ，表达式 $a<<3$ ，则表示将 a 值左移 3 位，空出的右边 3 位填入 0，

其结果为 10101000B。

6. 位右移运算符`>>`

右移运算符的作用是将一个数的各位顺序右移若干位。在右移运算中，低位移出舍弃不用，高位补 0。

例如：若 $a=11110101B$ ，表达式 $a>>3$ ，则表示将 a 值右移 3 位，空出的左边 3 位填入 0，其结果为 00011110B。

16.8.6 运算符的运算优先次序

运算符的运算优先次序，是由运算符的优先级决定，如表 16.6 所示。

- 在 C51 语言中以括号的运算优先级最高。
- 相同优先次序的运算符，由左而右运算。
- 括号内的运算，按照各运算符的优先次序来运算。

表 16.6

基本运算符优先级表

基本运算符优先级
<code>! , ~ , ++ , -- , - (取负) , * , \ , %</code>
<code>+ , - (减号) , << , >> , < , <= , > , >= , == , !=</code>
<code>& , ^ , , && , </code>

16.9 C51 流程控制语句

16.9.1 流程结构及其流程图

C51 语言有 3 种基本流程结构：顺序结构、选择结构和循环结构。

1. 顺序结构

顺序结构是最基本、最简单的结构，这种结构的程序流程是按语句的顺序依次执行的。如图 16.4 所示，程序先执行 A 操作，然后执行 B 操作，按着语句先后排列顺序进行。

2. 选择结构

选择结构可以使程序具有决策能力，它根据给定的条件进行判断，由判断的结果决定执行两支或多支程序段中的一支。如图 16.5 所示，程序根据给的条件 P 选择执行 A 或 B，当条件满足，即条件语句为“真”时，则执行 A 操作；当条件不满足，条件语句为“假”时，则执行 B 操作。

3. 循环结构

循环结构一般是在给定的条件为真时，反复执行某个程序段。循环结构又分为两种：当

型和直到型。

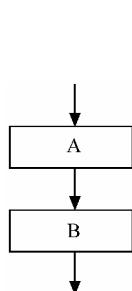


图 16.4 顺序结构示意图

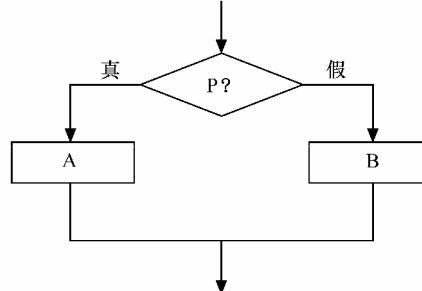


图 16.5 选择结构示意图

(1) 当型循环结构。

如图 16.6 所示，当给定的条件 P 为真时，重复执行 A 操作，当条件为假时，退出循环。

(2) 直到型循环结构。

如图 16.7 所示，先执行 A 操作，再判断给定的条件 P，若 P 为真时，重复执行 A 操作，直到条件为假时，退出循环。

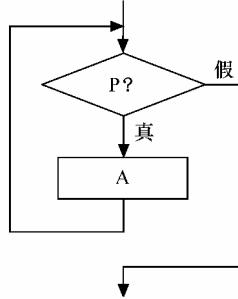


图 16.6 当型循环结构

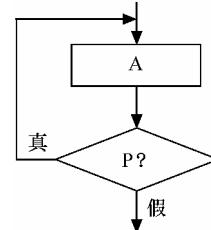
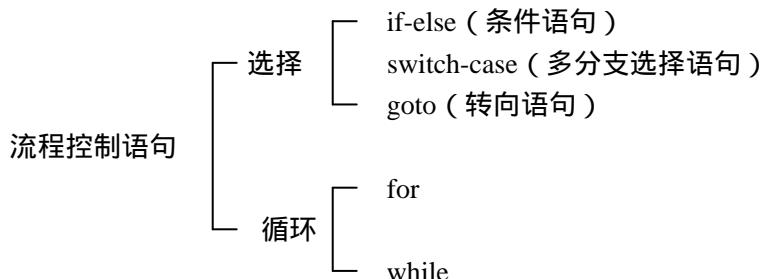


图 16.7 直到型循环结构

当型循环结构与直到型循环结构的区别是：当型是先判断，后执行 A 操作；直到型是先执行 A 操作，后判断。

16.9.2 流程控制语句按功能分类

流程控制语句是用来实现不同流程结构的语句。C51 语言的流程控制语句不多，按其功能分类如下：



此外还有：

辅助控制语句 [break
continue
return]

16.9.3 循环语句

C51 语言的流程控制循环语句主要有 for 和 while。

1. for 循环语句

for : 流程控制语句中的循环语句。在 C 语言中 , for 语句是使用最灵活的循环控制语句。
for 语句的一般形式为 :

```
for(表达式 1 ; 表达式 2 ; 表达式 3 )
{
    语句 A ;
}
语句 B ;
```

【说明】

表达式 1 : 设定循环变量初始值。

表达式 2 : 判断是否为真 , 是真则继续执行大括号内语句 A ; 如果为假 , 则跳出循环执行语句 B 。

表达式 3 : 循环控制变量值的运算 (递增或递减)

【应用】

(1) 无限循环

```
for( ; ; ) ; /* 执行无限多次 */
```

或

```
for ( ; ; )
{
    语句 ; /* 执行无限多次 */
}
```

for 语句中 , 可以没有表达式 1 、表达式 2 或表达式 3 ; 若 3 个表达式都没有 , 则相当于一个无限循环 , 与汇编语言 “ JMP \$ ” 相同。

(2) 执行 n 次语句

```
int i;
for( i=0 ; i<100 ; i++ )
{
    语句 ; /* 执行 100 次 */
}
```

执行 for 循环 100 次后 , 表达式 2 为假 , 此时离开 for 循环 , 往下执行程序。

(3) 时间延时

```
int i;
for(i=0;i<1000;i++) ; /* 执行 1000 次 */
```

此循环是一个空循环语句，不做其他事情，只是让 CPU 等待一段时间，起延时作用。当执行 for 循环 1000 次后，表达式 2 为假，此时离开 for 循环，往下执行程序。

2. while 循环语句

while 与 for 一样，也是循环结构语句，一般形式为：

```
while( 表达式 )
{
    语句 A ;
}
语句 B ;
```

【说明】

- (1) 如果表达式为真，则执行语句 A。若表达式为假，则循环结束，往下执行语句 B。
- (2) 如{}内，只有一个语句，可以省略括号。
- (3) 多层循环与 for() 规则相同。

【应用】

(1) 无限循环

```
while (1); /* 执行无限多次 */
```

或

```
while (1)
{
    语句 ; /* 执行无限多次 */
}
```

(2) 执行 n 次语句

```
int i;
i =100
while( i>0 )
{
    语句 ; /* 执行 100 次 */
    --i ;
}
```

(3) 时间延时

```
int i;
```

```
i = 0
while( i<1000 )
{
    ++
    /* 执行 1000 次 */
}
```

16.9.4 选择语句

C51语言的流程控制选择语句主要有if-else、switch-case和goto。

1. If-else 条件选择语句

if-else：流程控制语句中的条件选择语句。在if语句中如果条件成立，则执行语句A，否则执行语句B。if-else语句的一般形式为：

(1) if{ }else{ }

```
if(表达式)
{
    语句 A
}
else
{
    语句 B
}
语句 C
```

(2) 省略else

```
if(表达式)
{
    语句 A
}
语句 C
```

【说明】

(1) if{ }else{ }：首先判断表达式是否成立，若表达式成立（为“真”）时，则执行语句A，执行完语句A后，执行语句C；若表达式不成立（为“假”），则执行语句B，执行完语句B后，执行语句C。

(2) if{ }：首先判断表达式是否成立，若表达式成立（为“真”）时，则执行语句A；若表达式不成立（为“假”）时，则跳过语句A，而直接执行下面语句C。

2. switch-case 多分支选择语句

switch-case：流程控制语句中的多分支选择语句，它的一般形式为：

```
switch(变量)
```

```

{
    case 条件 1:语句 1;
        break;
    case 条件 2:语句 2;
        break;

    case 条件 n:语句 n;
        break;
    default:语句 n+1;
        break;
}
语句 A

```

【说明】

(1) switch 语句的执行过程：先将“变量”与“条件”相比较，若相等，则执行该 case 条件后面的语句，然后执行 break 语句，退出 switch 语句，继续执行后面的语句；若“变量”不与任何“条件”相等，就执行 default 后面的语句，然后退出 switch 语句，向下执行语句 A。

(2) switch 的变量，只能是整数或字符。

(3) break：是中断语句。在 switch-case 语句中，是中断 case 语句判断的功能。该语句也可以跳离任何循环，如 for、while、if-else 循环。

3. goto 转向语句

goto：是无条件转向语句，它的一般形式是：

```
goto 语句标号；
```

执行 goto 语句后，程序将无条件地转到标号后面的语句处执行。goto 语句可以与 if 语句一起构成循环结构，例如：

```
F0: if(P3_2==0)goto F0;
```

该语句是由 goto 语句和 if 语句一起构成循环结构，只要键未放开时，程序一直在此循环。其中，“F0”为语句标号，语句标号是一个标识符。C 语言中任何一个语句都可以有一个语句标号，其一般形式是：

语句标号：语句
例：loop: a=P1;

16.10 C51 函数

C51 语言程序是由一个主函数和若干个函数构成。其中主函数即是主程序，它是程序的

起点，由它调用其他函数，其他函数也可以互相调用。除了主函数之外的其他函数称为一般函数，简称函数。

在 C51 语言中，子程序是用函数来实现的，每一个函数用来实现一个特定的功能。

16.10.1 C51 函数定义的一般形式

从用户使用的角度看，函数有两种：标志库函数和用户自定义函数。标志库函数是由 C51 编译器提供的，不需要用户进行定义，可以直接调用。用户自定义函数是用户根据自己的需要编写的能实现特定功能的函数，它必须先进行定义后才能调用。函数定义的一般形式为：

```
返回值类型 函数名(形式参数列表)
{
    函数体
}
```

函数要有函数名，函数名的前边是标注返回值形式，函数后边括号内是参数，例如：

```
void DELAY (char k )
{
    C 语言语句 ;
}
```

在例中，DELAY 为函数名，由使用者来命名；函数名的前边为返回值类型，标识符 void 说明该函数没有返回值；函数名后边括号内 (char k) 为函数的参数。

函数根据返回值形式及有无参数，写法共有种：

- 无返回值，无参数输入。
- 无返回值，有参数输入。
- 有返回值，无参数输入。
- 有返回值，有参数输入。

前两种函数形式使用较多。

1. 无返回值，无参数输入

例如：

```
void DELAY ( ) /* 延时函数 */
{
    unsigned int i;
    for(i=0;i<30000;i++) /* 延时约 0.3s */
}
```

无参数函数一般不带返回值，因此函数返回值类型标志符可以省略，写为“DELAY()”的形式，去掉 DELAY() 前边的“void”。

2. 无返回值，有参数输入

例如：

```

void delay250(unsigned int k)      /* 延时函数 */
{
    unsigned int j,i;
    for(i=0;i<k;i++)
    {
        for(j=0;j<25;j++);
    }
}

```

函数括号内的“k”是参数，假如在主函数中调用延时函数写为：delay250(2)，即参数为2，其延时时间为 $250 \times 2 = 500\mu\text{s}$ 。如果调用延时函数写为：delay250(2000)，即参数k为2000，其延时时间为 $250 \times 2000 = 500000\mu\text{s} = 0.5\text{s}$ 。

16.10.2 C51 库函数

C51 编译器提供了丰富的库函数，用户可以根据需要随时调用。使用时只需在源程序的开头用编译预处理命令 #include 将相关的头文件包含进来即可。如 AT89X51.H 就是库函数，它定义了 C51 单片机各 I/O 地址、参数、符号等。使用时通过加载头文件，在编写 C 程序时，就可以不需要考虑单片机内部的存储器分配等问题，例如：

```

#include <AT89X51.H>
main( )
{
    if(P3==0xfe) P1=0x55;      /* P3、P1 已在头文件 reg51.h 中定义 */
}

```

16.10.3 C51 中断函数

中断服务程序在 C51 中是以中断函数的形式出现的，C51 编译器支持在源程序中直接开发中断服务程序，中断函数的编写形式为：

```

void 函数名( void ) interrupt n using m
{ 函数体语句 }

```

其中，interrupt 表示函数是一个中断服务函数，interrupt 后的整数 n 表示该中断服务函数是对应哪一个中断源。每个中断源都有系统指定的中断编号，如表 16.7 所示。using 指定该中断服务程序要使用的工作寄存器组号，m 为 0~3。

表 16.7 中断源的中断编号

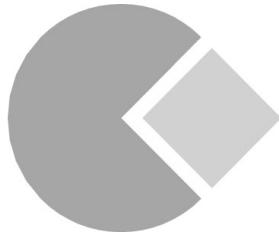
中断源	中断编号
外部中断 0	0
定时器 T0	1
外部中断 1	2

续表

中 断 源	中 断 编 号
定时器 T1	3
串行口中断	4

用 C 语言编写中断服务程序时，只要使用 interrupt 和 using 声明是中断服务函数，就不必像使用汇编语言哪样，考虑中断入口、现场保护和现场恢复处理等，这些问题都由系统自动解决，显然比使用汇编语言编写中断程序时简单。

中断服务函数是系统调用的，程序中的任何函数都不能调用中断服务函数。



第 17 章 Keil C51 的使用

Keil C51 集成开发软件功能强大、使用广泛，支持 80C51 的 C 语言编程。

17.1 Keil C51 的安装

安装 Keil μ Vision2 集成开发环境的步骤如下。

- (1) 将 Keil C51 解压到某个目录下，如果不需要解压则跳过这一步。
- (2) 在第 1 步解压目录下找到 setup.exe 安装文件，双击该文件即可开始安装，如图 17.1、图 17.2 所示。

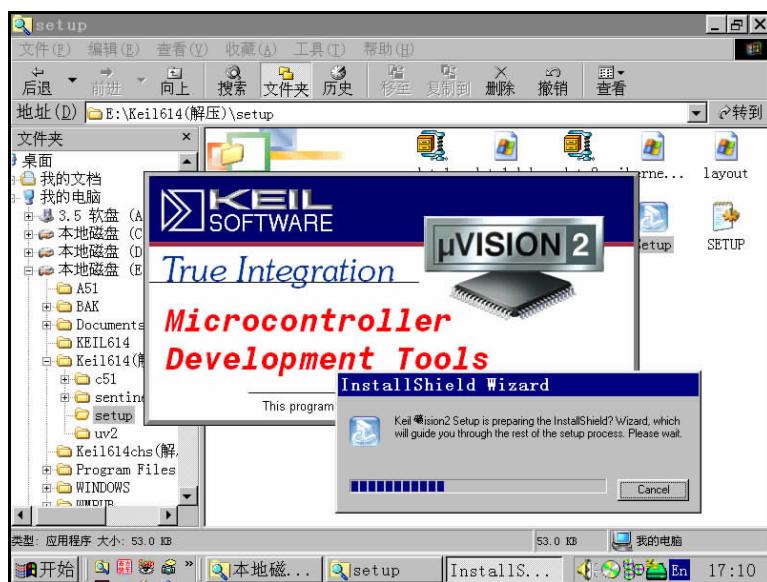


图 17.1 安装初始化

- (3) 单击“Next”命令按钮，这时会出现如图 17.3 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，如果是免费使用可选择 Eval Version 评估版（不需要注册码）。

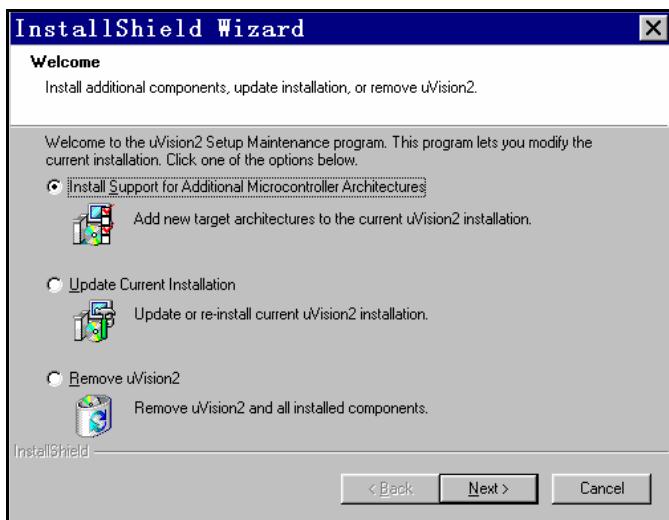


图 17.2 安装向导对话框

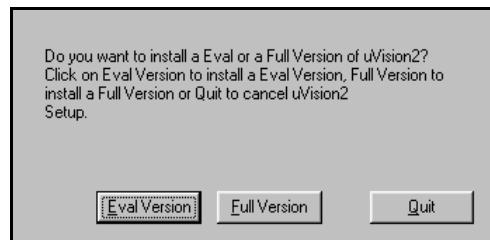


图 17.3 安装询问对话框

(4) 此后在弹出的几个确认对话框中单击“Next”按钮，这时会出现如图 17.4 所示的安装路径设置对话框，默认路径是 C:\KEIL，也可以选择安装在自己设置的目录里。

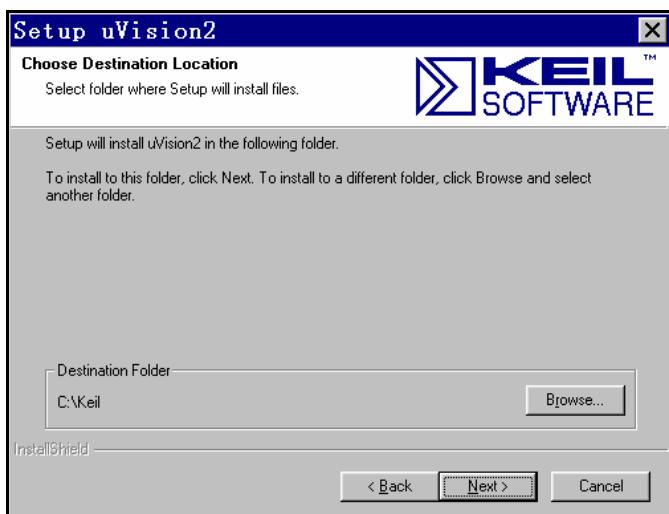


图 17.4 安装路径设置对话框

(5) 安装完毕后单击“Finish”按钮加以确认，此时可以在桌面上看到 Keil μ Vision2 的

快捷图标。

17.2 用 Keil C51 开发单片机

双击桌面的 Keil 快捷图标进入如图 17.5 所示的 Keil μ Vision2 集成开发环境。

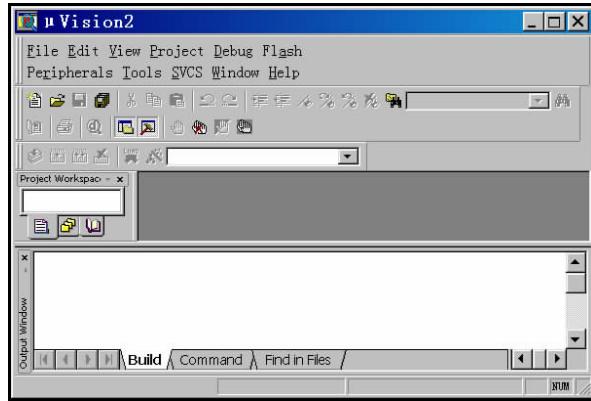


图 17.5 Keilμ Vision2 集成开发工作窗口

该平台包含一个编辑器、一个项目管理器和相应的工具，工具包括编译器、连接/定位器、目标代码到 HEX 的转换器等。在该平台上进行的主要操作有：编写源程序，建立工程项目文件，汇编及编译，连接及产生可执行的 hex 文件。

17.2.1 编写源程序

在 Keil C51 中编写源程序的具体操作步骤如下所示。

1. 选择“New”

在集成开发工作窗口中选择 File → New，如图 17.6 所示。

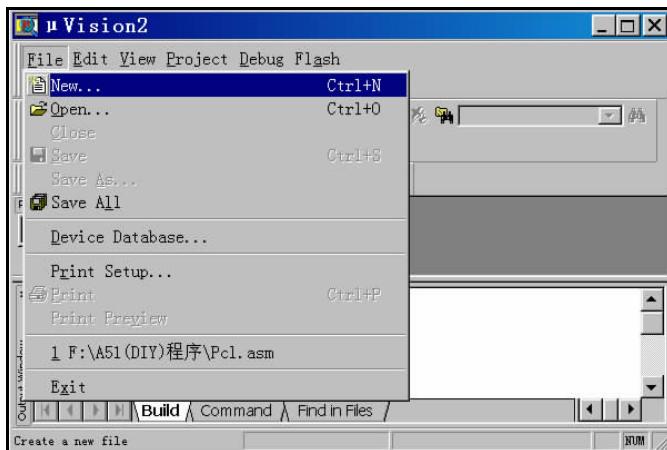


图 17.6 选择“New”选项

2. 打开文本编辑窗口

单击“New”选项出现如图 17.7 所示的 Text1 文本编辑窗口。此时，可在 Text1 中编写 C 语言源程序，如图 17.8 所示。

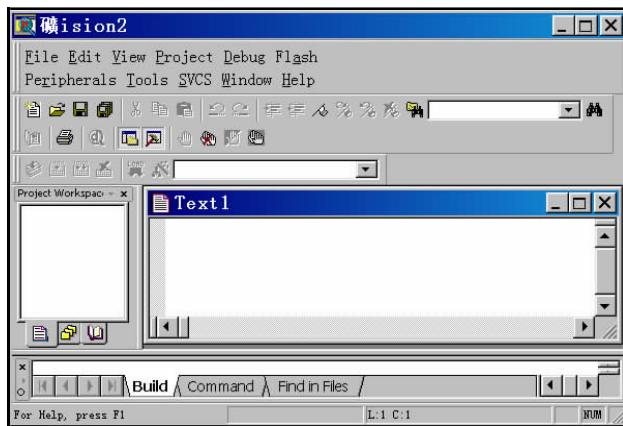


图 17.7 Text1 文本编辑窗口

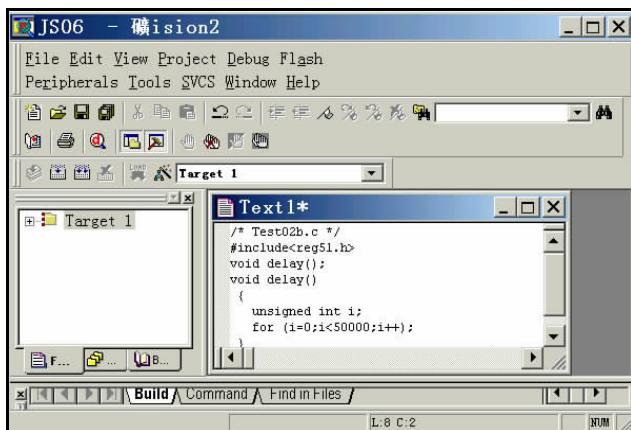


图 17.8 在 Text1 中编写源程序

3. 保存文件

选择 File Save As，如图 17.9 所示，填写文件名并保存在设定的目录中。假设目录设置为 F:\程序\C51，填写的文件名为“Test02b.C”。文件名可任意，但后缀必须是.C，然后单击“保存”按钮。

17.2.2 建立工程项目文件

建立工程项目的目的是为了便于程序管理，因为在编译、连接/定位等过程中，将会生成几个文件，用“工程项目”可以把它们管理起来。

在 Keil C51 集成开发环境下进行项目管理，主要有 4 步：填写新工程项目名称、选择单片机型号、设置输出选项和向工程项目里添加源程序。



图 17.9 选择目录及填写文件名

1. 填写新工程项目名称

在“Project (工程)”菜单中选择“New Project (新工程)”，如图 17.10 所示。

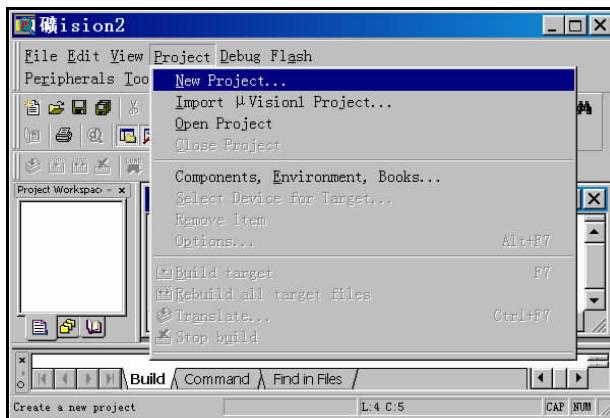


图 17.10 选择新工程项目

单击“New Project (新工程)”，便出现如图 17.11 所示的保存工程项目对话框，在文件名里要填写新工程名称，如“Test02b”，工程项目名一般是与文件名相同，便于记忆。

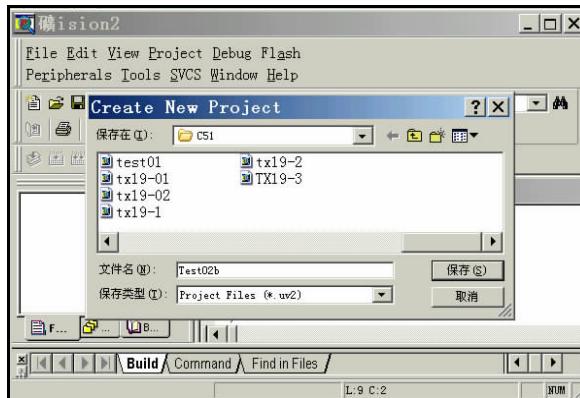


图 17.11 保存工程项目对话框

2. 选择单片机型号

单击“保存”按钮后，便弹出选择单片机型号对话框。在对话框的列表中选择“Atmel”（公司名），单击 Atmel 前边的“+”号后，会展现出所有该公司生产的单片机型号，从中选定“AT89C51”型号，如图 17.12 所示。

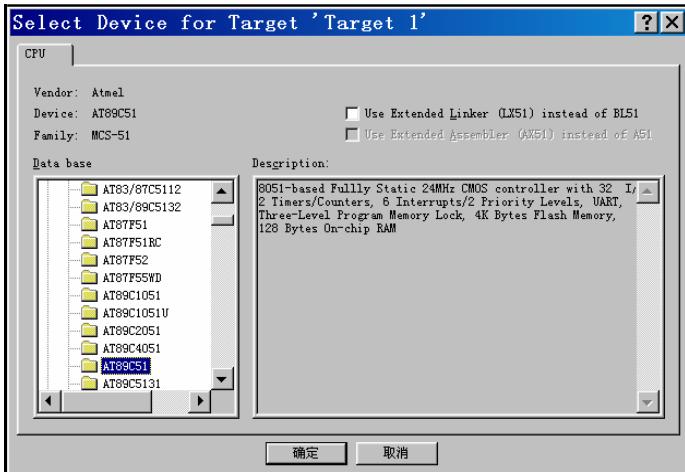


图 17.12 选择单片机型号对话框

3. 设置输出选项

单片机型号确定后，出现如图 17.13 所示对话框，要注意选择按“否 (N)”按钮，当对话框消失后，在窗口左边“Target 1”上单击鼠标右键，并在弹出的下拉菜单中选择工作环境的命令“Options for Target ‘Target 1’”，如图 17.14 所示。

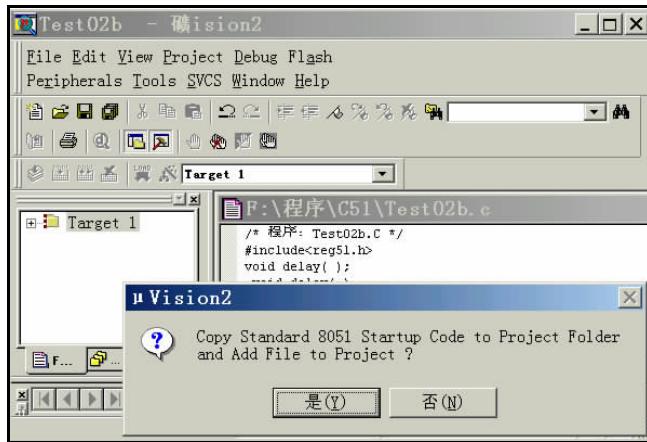


图 17.13 对话框

在工作环境的对话框中选择 Output 选项卡以设置输出选项，并选中 Debug information 及 Create HEX file 两复选框，表示在输出的 HEX 可执行文件过程中，产生调试的相关文件，方便调试用，如图 17.15 所示。

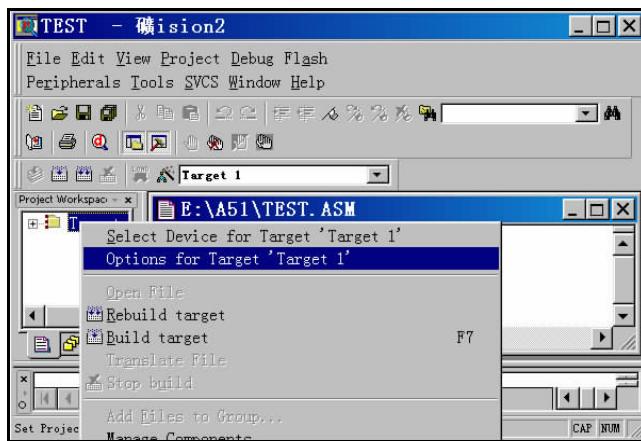


图 17.14 选择命令设置工作环境选项

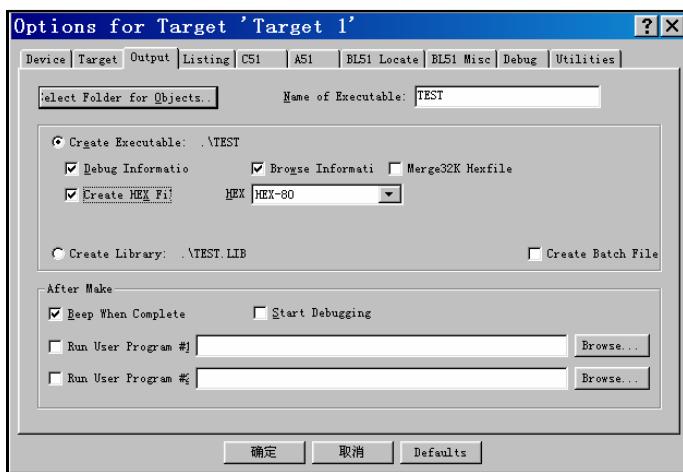


图 17.15 设置输出选项

4. 向工程项目里添加源程序

(1) 单击“Target 1”前面的+号，展开里面的内容 Source Group1，如图 17.16 所示。

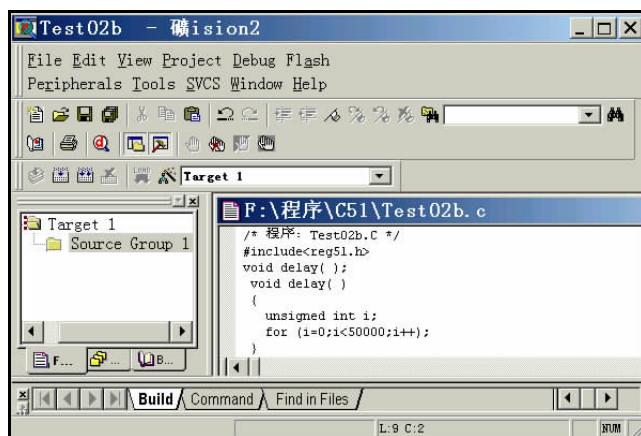


图 17.16 Target1 展开图

(2) 右键单击“Source Group1”，在弹出的菜单中选择“Add Files to Group ‘Source Group1’”添加程序选项，如图 17.17 所示。

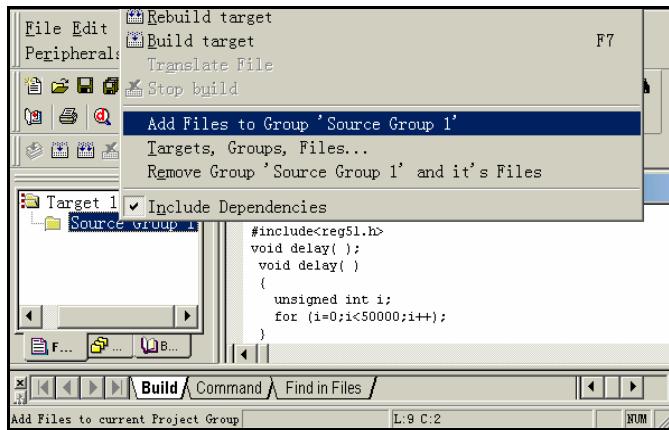


图 17.17 选择添加程序命令

(3) 单击添加程序命令后，弹出添加源程序文件窗口，把文件类型设置为“*.c”C 语言文件格式，并填上文件名“Test02b.c”，如图 17.18 所示。

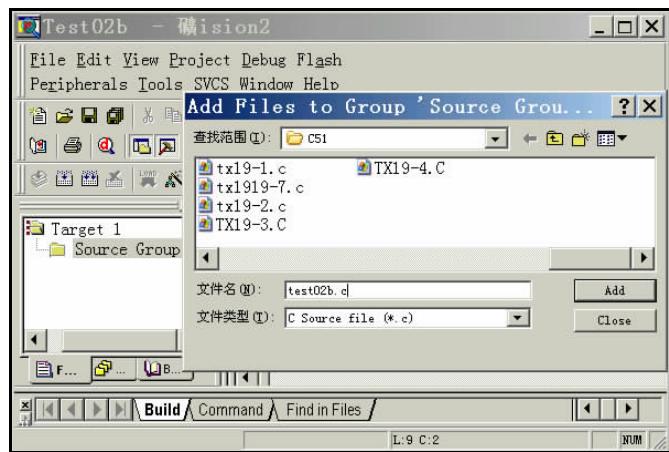


图 17.18 添加源程序文件窗口

(4) 在上图中，单击“ADD”按钮将文件加入项目。再单击“Close”按钮，关闭窗口，完成向工程项目里添加源程序的任务，如图 17.19 所示。

如果单击“ADD”后（因为还没单击“Close”按钮，对话框并不消失），误认为操作没有成功而再击一次，这时会出现提示你所选文件已在列表中的对话，如图 17.20 所示，此时应单击“确定”，返回前对话框，然后点击“Close”按钮即可。

17.2.3 产生可执行的 HEX 文件

按下 F7 键则会进行汇编及编译，链接和产生可执行的 HEX 文件，并在窗口的下方状态栏里显示出：“Test02b.c” - 0 Error(s), 0 Warning(s)，表示成功，如图 17.21 所示。

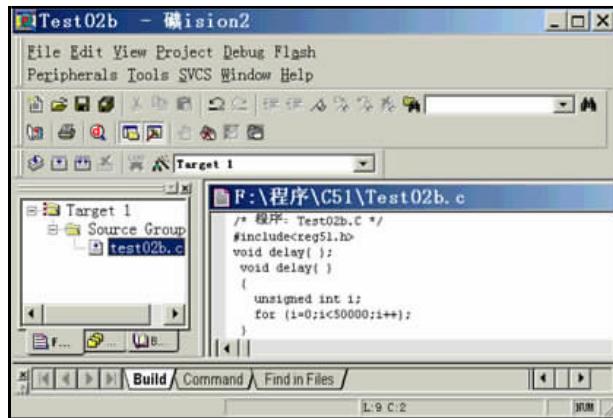


图 17.19 源程序文件加入项目

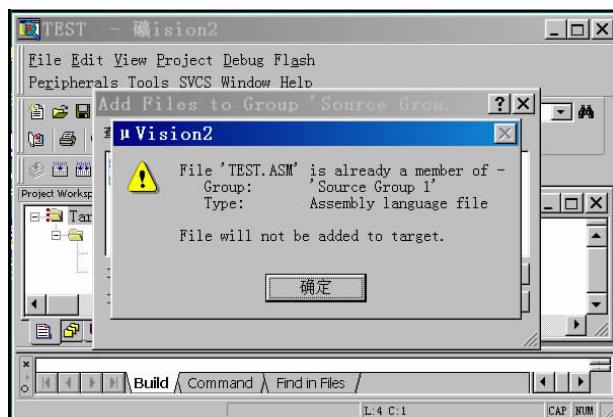


图 17.20 文件已在列表中提示

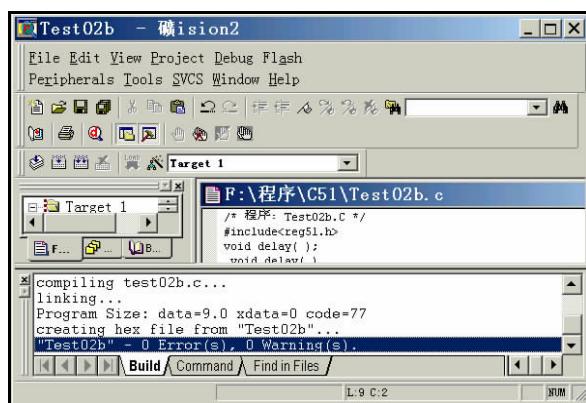


图 17.21 对源程序进行编译、连接

17.3 Keil 中的软件仿真

Keil C51 集成开发软件内设有调试仿真器，在集成开发工作窗口的上方菜单栏中“Debug”

为调试菜单命令，“Peripherals”为外围器件菜单。同时，在工具栏内有调试仿真器图标，如图 17.22 所示。

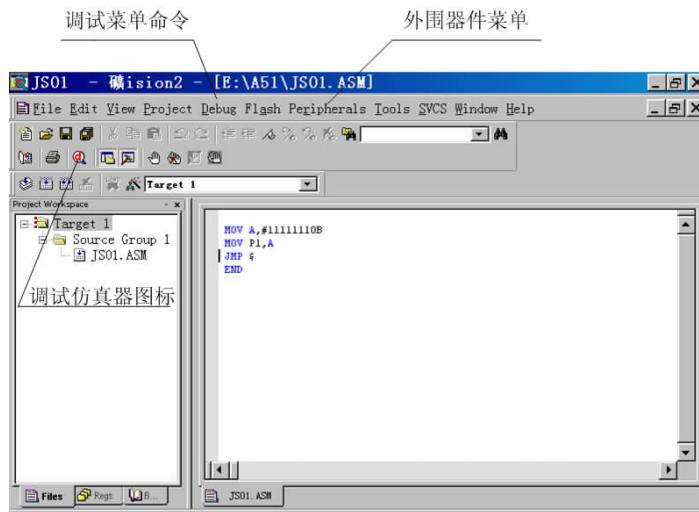


图 17.22 Keil C51 内设有调试仿真器

17.3.1 操作的一般步骤

1. 进入调试仿真状态

双击上图里调试仿真器图标，便进入调试仿真工作窗口，如图 17.23 所示，左侧会出现 80C51 内部主要的寄存器，如 r0~r7、a 等，右侧为程序调试窗口。

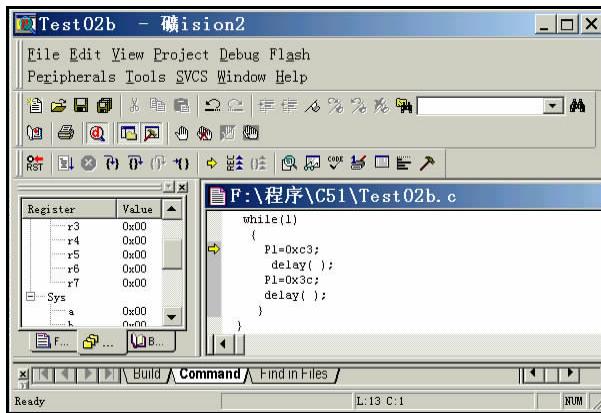


图 17.23 调试仿真窗口器

2. 打开外围控制的特殊寄存器窗口

选择“Peripherals”菜单中的各种命令，可以打开 80C51 外围控制的特殊寄存器显示窗口，方便调试时观察内部寄存器值的变化，如图 17.24 所示。

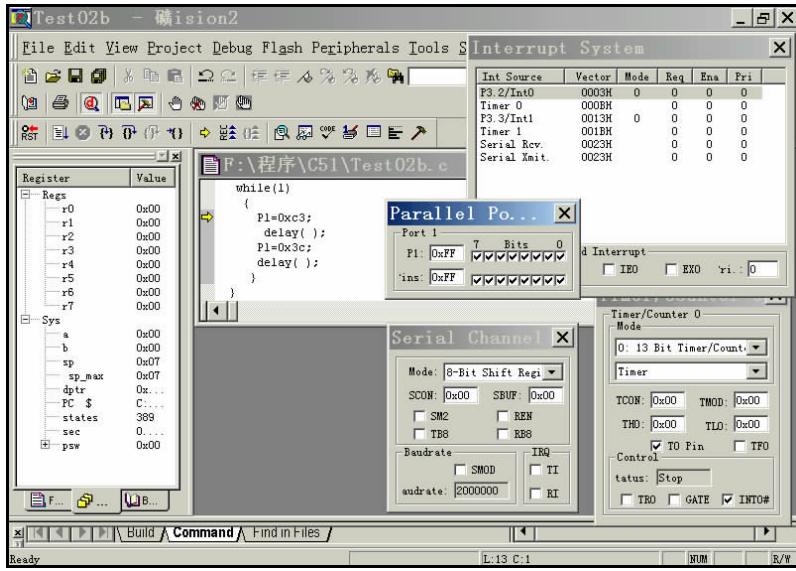


图 17.24 调出特殊寄存器显示窗口

其中，Interrupt 为中断观察窗口，I/O-Ports 为 I/O 口观察窗口，Serial 为串口观察窗口、Timer 为定时器观察窗口。

3. 常用的调试功能图标

在做调试仿真时常用的功能图标有 7 种，如图 17.25 所示。

(1) Reset CPU: 软件重置动作，复位 CPU，程序计数器归 0 (PC=0)，使程序可以重新开始调试。

(2) Run/ go (F5): 运行程序，除非遇到断点或是下达 stop 命令，才可使系统停止。

(3) Halt/ Stop (Esc): 停止执行程序。

(4) Step into (F11): 单步进入，单步执行程序，遇到子程序会进入子程序中来执行程序。

(5) Step over (F10): 单步越过，单步执行程序，遇到子程序时会跳过子程序。

(6) Step out (Ctrl+F11): 单步跳出，跳出目前子程序，来继续执行程序。

(7) Run to cursor (Ctrl+F10): 执行程序一直到光标所在处。

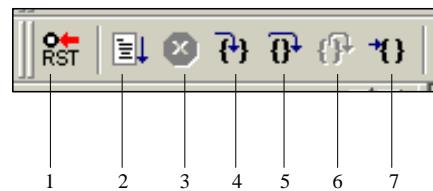


图 17.25 常用的功能图标

17.3.2 仿真举例说明

1. 单步执行程序

(1) 打开需要调试仿真的文件，如图 17.26 所示。

(2) 双击调试仿真器图标，进入仿真状态。

(3) 选择“Peripherals”菜单，该程序中外围器件使用的是 P1 寄存器，所以应选择“I/O-Ports”命令，单击“Port 1”后，便出现寄存器 P1 观察窗口，如图 17.27 所示。

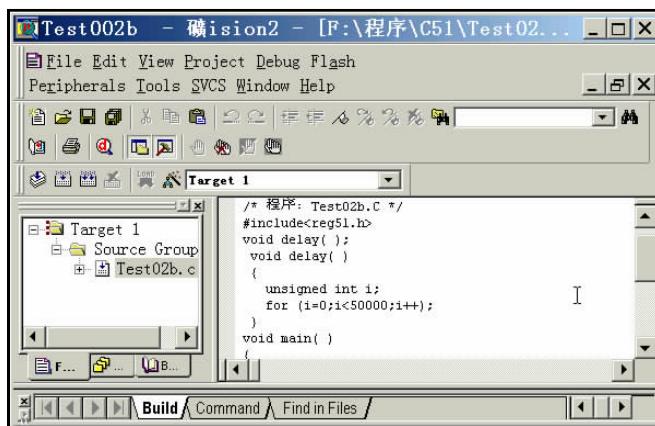


图 17.26 打开需要调试仿真的文件

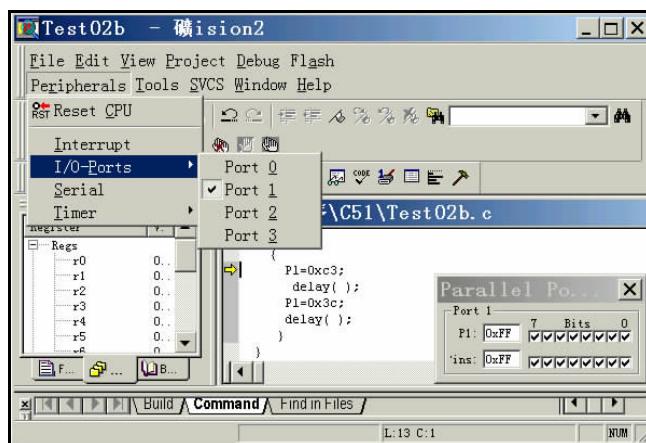


图 17.27 P1 寄存器观察窗口

(4) 单步执行程序。开始进入仿真时，黄色箭头指向主函数“P1=0xc3”一行语句，即指向将要运行的语句，如图 17.28 所示。

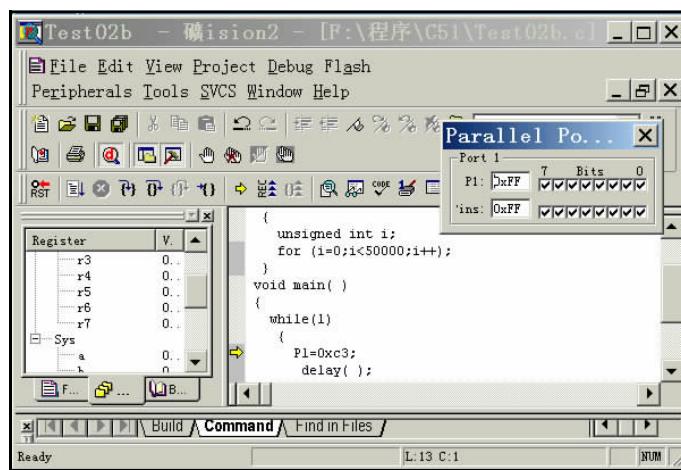


图 17.28 开始进入仿真

为了观察仔细，选择单步执行程序方法，即单击一次“Step into”，程序向下运行一行。单击“Step into”命令按钮（或按 F11 键）后，程序开始执行原来黄色箭头指向的语句（黄色箭头随之向下移动），此时会看到 P1 寄存器的值发生变化，由原来 0xff 变为 0xc3（即二进制数 11000011B）。

从 P1 观察窗口可以看到 8 位（0~7）引脚中，中间 4 位无勾号，表明此引脚有输出，与该引脚连接的发光二极管会亮起，如图 17.29 所示。

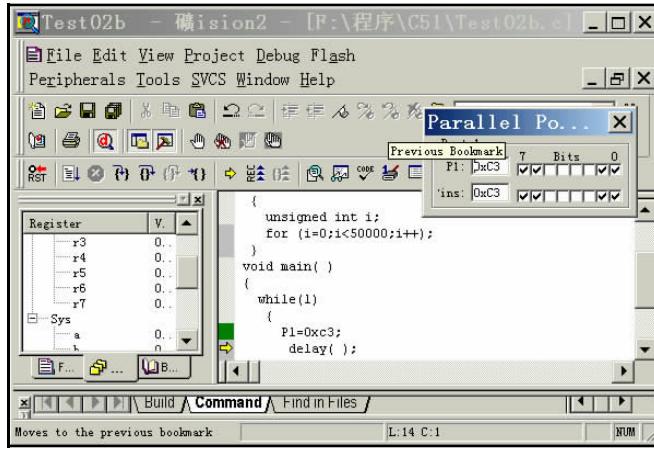


图 17.29 观察 P1 窗口变化

再单击“Stop into”执行下一语句，此时程序运行进入了 delay() 延时函数内，黄色箭头指向“for(i=0;i<50000;i++);”语句，如图 17.30 所示。

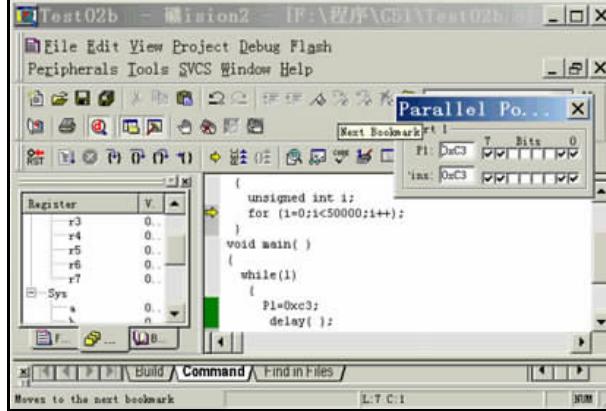


图 17.30 程序进入延时函数内运行

（5）跳出子程序。请注意，此时再单击“Stop into”，程序开始执行延时语句，如果继续这样单步执行，需要执行 50000 次（语句中 i=50000）才能跳出延时语句，显眼时间太长。

解决办法有两种。

- 将原延时数值减小，如将 50000 改为 5 并重新编译，这样单击 5 次即可跳出循环；
- 单击“Step out (Ctrl+F11)”跳出目前子程序，使程序回到主函数处，继续向下运行，如图 17.31 所示。

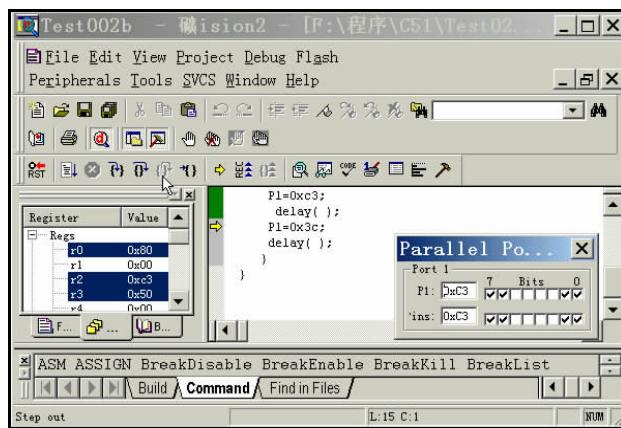


图 17.31 点击 Step out 跳出子程序

(6) 继续单击“Stop into”，执行“P1=0x3c;”语句后，P1 寄存器的值由 0xc3(11000011B) 变为 0x3c(00111100B)，从 P1 观察窗口可以看到中间 4 个有勾号，两边各有两个无勾号，如图 17.32 所示。表明中间 4 个灯熄灭，两边灯亮起。程序不断循环，P1 端口所接的灯不断交替点亮。

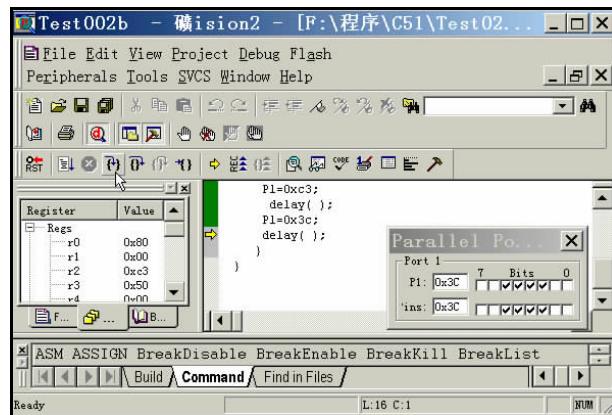


图 17.32 P1 观察窗口两边去掉打挑

2. 停止和重置

在程序调试仿真中，有时需要停止执行程序，还有时需要程序重新开始。

- (1) 单击“Halt/Stop”(或按 Esc 键)，程序运行会停止。
- (2) 单击“Reset CPU”，使程序计数器归零，黄箭头重新指向程序的第一行。这时，仿真可以从头开始。

3. 设置断点和取消断点

(1) 将光标指到所要设置断点的程序的前方空白处，双击鼠标左键，则最左方处会出现红色标记，表示断点设于此。再次双击鼠标左键一次，就可以取消此次断点设置。

(2) 一旦设置了断点，例如，将断点设在主程序第二次调用延时函数处，按 F5 后，程

序便会自动停在该断点处，如图 17.33 所示。设置断点主要是方便追踪程序执行的结果。

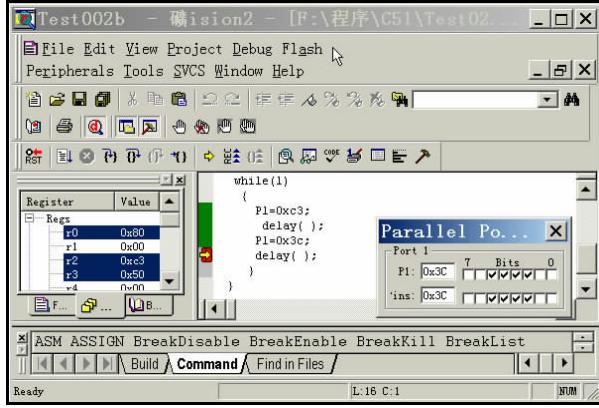


图 17.33 设断点

(3)“Run to cursor”(Ctrl+F10)命令与设置断点有相同的作用，将光标指到所要设置的程序前方空白处，执行该命令后，程序会一直运行到光标所在处才会停止。

17.3.3 几个常用命令使用区别

常用的 Run、Step over、Step into 和 Step out 命令，在使用上主要有以下区别。

(1)“Run”是自动执行程序，除非遇到断点才停止。例如在该程序仿真中，如果点击“Run”图标，程序会一直运行到第 3 行才停止。假如事先在第 2 行设置一个断点，则执行 Run 命令后运行到第 2 行停止。

(2)“Step over”命令和“Step into”命令是单步执行程序，区别是“Step over”命令只能在主程序中运行，不会进入子程序来执行程序；而“Step into”命令能进入子程序。

(3)当执行“Step into”命令进入子程序后，再执行一次“Step out”命令，则会立即跳出子程序。也就是说，执行“Step over”命令不能进入子程序中运行；而执行“Step out”命令是从子程序运行中跳出来。“Step over”与“Step out”命令只能在仿真运行有子程序的程序中体现出来。



5.1 单片机 C 语言实例篇

本篇将通过节日彩灯设计、开关输入设计、报警声设计、交通灯控制和通信测试等实例，使读者学会单片机 C 语言程序设计的方法。



第 18 章 节日彩灯设计

在迎接和庆祝重大节日时，许多单位都要利用彩灯进行装饰布置，那五颜六色的彩灯和不断变换着的图形会使人感到十分奇妙，增加节日喜庆的气氛。本章将介绍使用单片机制作彩灯控制电路，各串彩灯的亮灭顺序、间隔时间、闪烁次数都由单片机 C 语言程序来控制。

18.1 彩灯闪烁

功能说明：使单片机 P1 端口所接的彩灯不断闪烁。

18.1.1 硬件设计

单片机彩灯控制电路如图 18.1 所示。

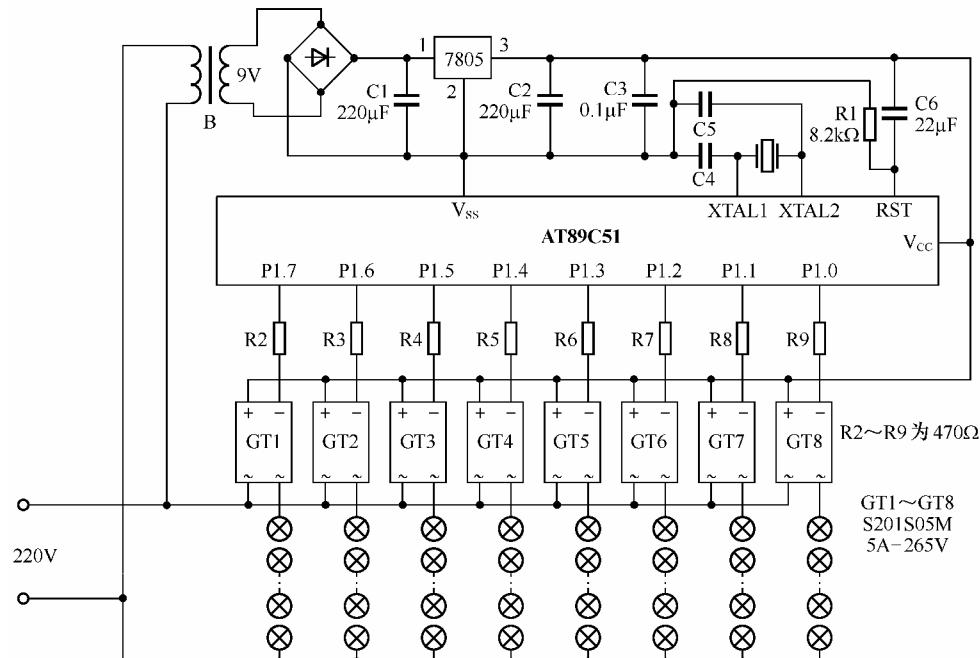


图 18.1 单片机控制彩灯电路

控制彩灯电路的核心元件是 AT89C51 单片机，用单片机控制彩灯的特点：电路简单、输出控制路数多，最多可控制 32 路（组）。彩灯是按预定程序自动运行，只要改变程序设计，就可以改变花样，无需改变硬件，设计灵活。

该电路的电源部分是由电源变压器 B、桥式整流器、7805 三端稳压器及 C1 ~ C3 组成，为整个电路提供稳定的+5V 直流工作电源。单片机的 XTAL1 (19 脚) 和 XTAL2 (18 脚) 引脚接有晶体和电容 C4、C5，组成了单片机的时钟电路。C6 和 R1 组成了简易的上电自动复位电路，从而保证单片机有序地工作。

单片机的输出采用了光电耦合式的固态继电器 GT 进行无触点隔离输出控制，可以有效防止在电源接通或断开时对系统产生的不良影响。

P1 输出端口均通过一只限流电阻连接到光电耦合式固态继电器 GT1 ~ GT8 的输入端，继电器的输出端则分别与一串（组）彩灯相连。

这样，单片机就可以通过控制 GT1 ~ GT8 的导通与截止来控制各组彩灯的加电与断电，从而实现整个彩灯的控制效果。

18.1.2 程序设计

1. 工作原理

程序的工作原理如图 18.2 所示。

P1 端口	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
数据 0x00	0	0	0	0	0	0	0	0
数据 0xff	1	1	1	1	1	1	1	1

图 18.2 将数据送入 P1 示意图

先将数据 0x00 (00000000B) 传送到 P1，使 P1 输出低电平 0，低电平使端口所接的继电器导通，点亮彩灯；再将数据 0xff (11111111B) 传送到 P1，使 P1 输出高电平 1，高电平使端口所接的继电器截止，彩灯因断电而熄灭。

亮与灭后都通过延时语句停留一段时间，以便能看清亮和灭，产生闪烁效果，再利用“ for (; ;) ”语句无限循环，保持着闪烁。

2. 程序

C 语言编写的彩灯闪烁源程序 cd18-1.c 代码如下：

```

01 #include<reg51.h>           /* 头文件 */
02 main( )                      /* 主函数 */
03 {
04     unsigned int i;            /* 声明无符号整型变量 i */
05     for(;;)                   /* 无限循环 */
06     {

```

```

07     P1=0x00;           /* 彩灯全亮 */
08     for(i=0;i<50000;i++); /* 延时 */
09     P1=0xff;           /* 彩灯全灭 */
10     for(i=0;i<50000;i++); /* 延时 */
11   }
12 }
```

18.1.3 代码详解

- 01：头文件，用来定义单片机 I/O 地址、参数、符号。
 02：主函数。
 03：主函数内容开始。
 04：声明无符号整型变量 i，延时语句中所使用的变量 i 在使用前必须进行声明。由于 i 变量数值范围为 0 ~ 50000，所以声明为无符号整型变量。
 05：循环语句，使彩灯一直保持闪烁。
 06：循环体开始。
 07：将数据 0x00(00000000B)送入 P1 并输出，使彩灯全亮。
 08：延时语句。延时约为 0.5s，以便能看到灯被点亮。
 09：将数据 0xff(11111111B)送入 P1 并输出，使彩灯全熄灭。
 10：延时语句，延时约 0.5s，以便能看到灯被熄灭。
 11：循环体结束。
 12：主函数结束。

18.2 延时模块

C51 语言编程中对延时时间的计算比较复杂，一般为试验值。

18.2.1 延时原理

延时原理：让单片机重复执行一些无具体任务的程序，利用 CPU 执行程序的时间来达到延时的目的。单片机的延时时间与使用的 CPU 时钟有关，当时钟频率为 12MHz 时，一个机器周期为 $1\mu s$ 。

延时可分软件延时和利用定时器延时。

18.2.2 软件延时

软件延时常以 delay 子程序来完成。它是利用程序循环来延时时间，每一次循环约 $10\mu s$ ，与所使用的循环语句有关。

【例】利用单层循环延时 500ms。

```
while(j<50000){ ++j; };
```

或

```
for(j=0;j<50000;j++);
```

语句中，j 变量里的条件式是试验值，单层循环没有多重循环延时准确，而且 j 的取值在 20000 以下为好。

【例】利用两层 for 循环，总延时时间为 1ms×count。

```
void delay×1ms(unsigned int count)
{
    unsigned int i, j;
    for(i=0;i<count;i++)
        for(j=0;j<120;j++);
}
```

语句中，j 变量里的条件式 $j < 120$ 是试验得出来的，为延时 1ms。

【例】利用两层 while 循环，总延时时间为 1ms×count。

```
void delay×1ms(unsigned int count)
{
    unsigned int j;
    while (count-- != 0)
    {
        for(j=0;j<72;j++);
    }
}
```

语句中，j 变量里的条件式 $j < 72$ 是试验得出来的，为延时 1ms。可以看出，利用 while 循环和上例中使用 for 循环来比较，其执行速度是有差异的。

【例】利用 3 层 for 循环，总延时时间为 10ms×count。

```
void delay×10ms(unsigned int count)
{
    unsigned int i, j, k;

    for(i=0;i<count;i++)
        for(j=0;j<10;j++)
            for(k=0;k<120;k++);
}
```

18.2.3 利用定时器延时

在对延时时间要求非常准确的场合下，例如时钟设计、音乐等，一般是利用定时器作延时，下面是利用定时器延时的程序。

【例】利用定时器延时，延时的时间为 50ms×i。

```
void delay50ms(unsigned int i)
{
```

```

TR0=1;
while(i!=0)
{
    TH0=-(50000/256);
    TL0=-(50000%256);
    while(TF0!=1);
    TF0=0;
    i--;
}
TR0=0;
}

```

18.3 彩灯由右向左侧逐渐点亮

功能说明：彩灯由右侧向左侧逐渐点亮，然后熄灭，再从右侧向左侧逐渐点亮，以此不断循环。本节硬件设计与18.1节相同。

18.3.1 程序设计

1. 工作原理

程序的工作原理如图18.3所示。

初始数据 a=0xfe	1	1	1	1	1	1	0
左移一次 <<a	1	1	1	1	1	0	0
再左移一次 <<a	1	1	1	1	1	0	0

图18.3 数据左移示意图

C语言中，<<会使数据左移，右边低位补0。每执行一次<<操作，数据0会向左移动一位，右边低位补0。0所对应的继电器导通，彩灯被点亮。

2. 程序

C语言编写的彩灯由右向左侧逐渐点亮源程序cd18-2.c，代码如下：

```

01 #include<reg51.h>           /* 头文件 */
02 main( )                      /* 主函数 */
03 {
04     unsigned char i,temp,a;   /* 声明无符号字符型变量 i、temp、a */
05     unsigned int s;           /* 声明无符号整型变量 s */
06     while(1)                  /* 无限循环 */
07     {
08         temp=0xfe;            /* 左移初始值 */
09         P1=temp;              /* 输出 */

```

```

10     for(s=0;s<30000;s++) ;      /* 延时 */
11     for(i=1;i<8;i++)          /* 设置移动位数 */
12     {
13         a=temp<<i;           /* 左移 i 位 */
14         P1=a;                 /* 输出 */
15         for(s=0;s<30000;s++) ; /* 延时 */
16     }
17 }
18 }
```

18.3.2 代码详解

- 01：头文件，用来定义单片机 I/O 地址、参数、符号。
- 02：主函数。
- 03：主函数内容开始。
- 04：声明变量 i、temp、a 为无符号字符型变量。
- 05：声明变量 s 为无符号整型变量。
- 06：无限循环。
- 07：while(1)循环体开始。
- 08：将左移初始值 0xfe (11111110B) 赋予变量 temp。
- 09：将数据 0xfe 送入 P1 并输出，使右边第一组灯亮。
- 10：延时约 0.3s，以便看清楚。
- 11：设置移动位数，即 for 循环次数。
- 12：for 循环开始。
- 13：左移 i 位，并将 i 值赋予变量 a。
- 14：将 a 值送入 P1 并输出。
- 15：延时一段时间。
- 16：for 循环体结束。
- 17：while(1)循环体结束。
- 18：主函数结束。

18.4 单组彩灯循环左右移动

功能说明：每次点亮一组彩灯，点亮的彩灯先向左移动；移动到左侧后，再从左侧向右侧移动，不断循环。本节硬件设计与 18.1 相同。

18.4.1 程序设计

1. 工作原理

图 18.4 所示为单组彩灯左移原理示意图，单组彩灯右移只是方向与之相反，其原理与左

移基本相同。

初始数据 a=0xfe	1	1	1	1	1	1	1	0
左移一次 <<a	1	1	1	1	1	0	1	
再左移一次 <<a	1	1	1	1	1	0	1	1

图 18.4 单组彩灯左移原理示意图

在 C51 语言中，<<会使数据左移，>>会使数据右移。但由于被移动的位会自动补 0，所以需要通过 |（位逻辑或）运算才能使移动时只有一组灯被点亮，即每执行一次<<操作，只能出现一个 0 向左（或向右）移动一位。0 所对应位接的继电器导通，彩灯被点亮；1 所对应的位截止，彩灯因断电而熄灭。

2. 程序

C 语言编写的单组彩灯循环左右移动源程序 cd18-3.c 代码如下：

```

01 #include<reg51.h>           /* 头文件 */
02 L_M();                         /* 声明左移函数 */
03 R_M();                         /* 声明右移函数 */
04 /* ----- 主函数 ----- */
05 main() {                       /* 主函数 */
06 {
07     while(1) {                  /* 无限循环 */
08     {
09         L_M();                 /* 调用左移函数 */
10         R_M();                 /* 调用右移函数 */
11     }
12 }
13 /* ----- 函数 ----- */
14 L_M() {                        /* 左移函数 */
15 {
16     unsigned char i,temp,a,b;  /* 声明无符号字符型变量 i、temp、a、b */
17     unsigned int s;            /* 声明无符号整型变量 s */
18
19     temp=0xfe;                /* 左移初始值 */
20     P1=temp;                 /* 输出 */
21     for(s=0;s<30000;s++);    /* 延时 */
22     for(i=1;i<8;i++) {        /* 设置移动位数 */
23     {
24         a=temp<<i;           /* 灯左移 i 位 */
25         b=temp>>(8-i);       /* 数据右移(8-i)位 */
26         P1=a|b;               /* 按位逻辑或运算，并输出 */
27         for(s=0;s<30000;s++); /* 延时 */
28     }
29 }
30 }
```

```

28      }
29  }
30 /* -----函数 ----- */
31
32 R_M( )           /* 右移函数 */
33 {
34     unsigned char i,temp,a,b; /* 声明无符号字符型变量 i、temp、a、b */
35     unsigned int s;          /* 声明无符号整型变量 s */
36     temp=0x7f;              /* 右移初始值 */
37     P1=temp;                /* 输出 */
38     for(s=0;s<30000;s++);   /* 延时 */
39     for(i=1;i<8;i++)        /* 设置循环(移动)位数 */
40     {
41         a=temp>>i;         /* 灯右移 i 位 */
42         b=temp<<(8-i);     /* 数据左移(8-i)位 */
43         P1=a|b;              /* 按位逻辑或运算，并输出 */
44         for(s=0;s<30000;s++); /* 延时 */
45     }
46 }
```

18.4.2 代码详解

- 01：头文件。
- 02：声明左移函数。
- 03：声明右移函数。
- 05：主函数。
- 06：主函数内容开始。
- 07：无限循环。
- 08：while(1)循环体开始。
- 09：调用左移函数。
- 10：调用右移函数。
- 11：while(1)循环体结束。
- 12：主函数结束。
- 14：左移函数。
- 15：左移函数开始。
- 16：声明变量 i、temp、a、b 为无符号字符型变量。
- 17：声明变量 s 为无符号整型变量。
- 19：将左移初始值 0xfe (11111110B) 赋予变量 temp。
- 20：将数据 0xfe 送入 P1 并输出，使右侧第一组灯被点亮。
- 21：延时约 0.3s，以便看清楚。
- 22：设置循环移动位数，即 for 循环次数。
- 23：for 循环开始。

- 24 : 左移 i 位，并将 i 值赋予变量 a。
 25 : 将数据右移 (8-i) 位，并赋予变量 b。
 26 : 将 a、b 值进行按位逻辑或运算，并送入 P1 输出。
 27 : 延时一段时间。
 28 : for 循环体结束。
 29 : 右移函数体结束。
 32 : 右移函数。
 33 : 右移函数开始。
 34 : 声明变量 i、temp、a、b 为无符号字符型变量。
 35 : 声明变量 s 为无符号整型变量。
 36 : 将右移初始值 0x7f (01111111B) 赋予变量 temp。
 37 : 将数据 0x7f 送入 P1 并输出，使左侧一组灯被点亮。
 38 : 延时约 0.3s，以便看清楚。
 39 : 设置循环移动位数，即 for 循环次数。
 40 : for 循环开始。
 41 : 右移 i 位，并将 i 值赋予变量 a。
 42 : 将数据左移 (8-i) 位，并赋予变量 b。
 43 : 将 a、b 值进行按位逻辑或运算，并送入 P1 输出。
 44 : 延时一段时间。
 45 : for 循环体结束。
 46 : 右移函数体结束。

18.4.3 经验总结

本节程序将彩灯左移和右移写成两个函数，由主函数调用实现循环左右移动的效果。左移函数 L_M() 和右移函数 R_M() 的区别是移动方向不同，但函数的基本结构相同。

函数中的运算符 <<、>> 的基本功能和汇编语言中的 RL、RR 指令功能类似，都是完成数据的左、右移动，但使用上有区别。以本节程序中左移函数为例，函数中是通过将逻辑运算符 <<、>> 和 | 综合应用来实现单组彩灯向左移动，实现步骤归纳如下。

(1) 将 a 左移 n 位，其右面低位补 0。

```
a<<n;
```

例如：

a 初始值为 11111110B，若 n 为 1，即 a 左移 1 位，a<<1 后低位补 0，结果 a 为 11111100。

(2) 将 b 右移 (8-n) 位，其左侧高位补 0。

```
b=a>>(8-n);
```

例如：

a 初始值为 11111110B，若 n 为 1，b 右移 (8-n) 位，即右移 7 位，右移后其左侧高位补 0，结果 b 为 00000001B。

(3) 将 a、b 进行或运算。

```
P=a|b;
```

则： $P=a \mid b = 11111100B \mid 00000001B = 11111101B$ ，通过对 a、b 进行或运算后，结果 $P=11111101B$ ，使彩灯从右数第二个灯开始被点亮。

假如将 a 左移 2 位 ($n=2$)，a、b 数据进行或运算后，结果 $P=11111011B$ ，使右数第 3 个灯亮。

彩灯单组右移原理与左移相同。

18.5 采用制表方法实现彩灯变化

功能说明：开始时彩灯由中间向两侧方向逐渐点亮两次；熄灭后再由两侧向中间逐渐点亮两次；之后，单组彩灯左移两次；右移两次；再闪烁两次，不断循环。本节硬件设计与 18.1 节相同。

18.5.1 程序设计

1. 工作原理

本节程序采用制表的编程方法控制彩灯变化。

- 将控制灯的数据集合在一起，构成一个数组 TABLE[]。数组中各元素（控制码）是按控制彩灯变化的顺序排列，数组中最后的一个元素是结束码。
- 取码按照数组下标值先由 0 开始 ($i=0$)，通过 if-else 选择语句，判断是否取到了结束码，如果取到了结束码说明一个循环结束，取码操作再从 0 开始。通过 while(1) 循环语句不断循环。
- 数组使用前需要先定义。

2. 程序

C 语言编写的采用制表方法实现彩灯变化源程序 cd18-4.c 代码如下：

```

01 #include <reg51.h>                                /* 头文件 */
02 unsigned char code TABLE[ ]={
03     0xff,0xe7,0xc3,0x81,0x00,                      /* 从中间向两侧点亮控制码 */
04     0xff,0xe7,0xc3,0x81,0x00,
05     0xff,0x7e,0x3c,0x18,0x00,                      /* 从两侧向中间点亮控制码 */
06     0xff,0x7e,0x3c,0x18,0x00,
07     0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f,/* 左移控制码 */
08     0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f,
09     0x7f,0xbff,0xdf,0xef,0xf7,0xfb,0xfd,0xfe,/* 右移控制码 */
10     0x7f,0xbff,0xdf,0xef,0xf7,0xfb,0xfd,0xfe,
11     0x00,0xff,0x00,0xff,                            /* 闪烁控制码 */

```

```

12     0x01                         /* 结束码 */
13 } ;                           /* 声明无符号字符数组 TABLE */
14 unsigned char i;             /* 声明无符号字符型变量 i */
15
16 DELAY( )                      /* 延时函数 */
17 {
18     unsigned int s;            /* 声明 s 为无符号整型变量 */
19     for(s=0;s<30000;s++);    /* for 循环延时语句 */
20 }
21
22 main( )                        /* 主函数 */
23 {
24     while(1)                   /* 无限循环 */
25     {
26         if ( TABLE[i]!=0x01 )      /* 取码不等于 0x01 时，执行 if{ }内的语句 */
27         {
28             P1=TABLE[i];          /* 输出 */
29             i++;                 /* 递加（加 1） */
30             DELAY( );           /* 调用延时函数 DELAY( ) */
31         }
32
33     else                         /* 取码等于 0x01 时，执行“ i=0 ”语句 */
34     {
35         i=0;                     /* 重新开始循环 */
36     }
37 }                               /* while(1)循环体结束 */
38 }                               /* 主函数结束 */

```

18.5.2 代码详解

01：头文件。

02 ~ 12：定义无符号字符数组 TABLE[]，数组中各元素就是彩灯的控制码，按控制彩灯变化的顺序排列，数组中的最后的一个元素是结束码，用来标志一个循环的结束。

14：声明无符号字符型变量 i。

16 ~ 20：延时函数。

23：主函数。

24：主函数开始。

24：无限循环开始。

26 ~ 36：为 if- else 选择语句，主要判断取码一个循环是否结束，如果没取到结束码 0x01，程序继续向下取码；如果取到结束码 0x01，说明一个循环结束。新开始循环取码再从 0 开始，即重新读取存放的第一个控制码。

在该段程序语句中，第 26 行 if 括号内的表达式 (TABLAE[i]!=0x01)，是指取码不等于

- 0x01 时表达式为“真”，执行 if{ }内语句。
- 28 行为 P1 输出语句。
- 29 行为下标值 i，每执行一次 i 值递加 1。
- 30 行语句是调用延时函数 DELAY()，使彩灯每次变化都停一段时间，以便看清楚。
- 33：如果 (else) 取码等于 0x01，则执行第 35 行语句。
- 35：使下标 i=0，程序重新开始循环。
- 37：while(1) 循环体结束。
- 38：主函数结束。



第 19 章 开关输入设计

在编程中经常会遇到开关信号的处理问题，本章通过几个开关输入程序实例来介绍常用的编程方法。

19.1 单开关输入状态指示灯

功能说明：开关 K 接在单片机 P3.0 引脚上，由 P1.0 和 P1.7 所连接的 LED 作开关状态指示灯。当开关 K 断开时 P1.0 指示灯亮；开关 K 接通时 P1.0 指示灯熄灭，P1.7 指示灯亮。

19.1.1 硬件设计

电路设计如图 19.1 所示。

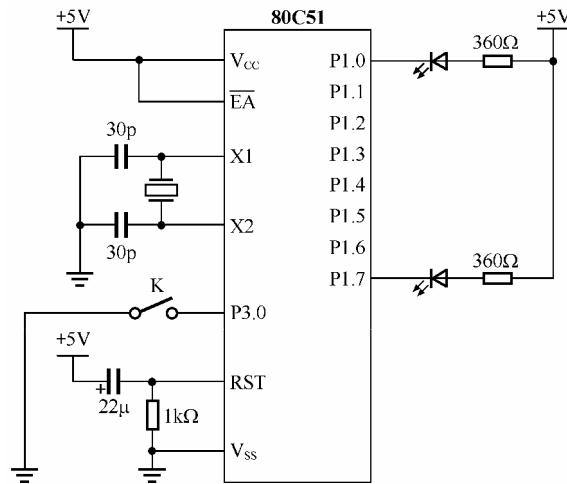


图 19.1 开关输入基本电路

开关 K 的一端与单片机 P3.0 连接，开关的另一端接地。单片机的 P1.0 与 P1.7 端口接有 LED 用来显示开关状态。当 P1.0 灯亮时，表示开关 K 处于断开状态；当 P1.0 灯熄灭而 P1.7 灯亮时，表示开关 K 处于接通状态。

19.1.2 程序设计

1. 工作原理

- 读出 P3.0 端口电平，如果是高电平，说明开关 K 处于断开状态；如果读出的是低电平，说明开关 K 处于接通状态。
- 通过条件选择语句 if-else 进行判断处理，如果读出 P3.0 端口是低电平，向 P1 送入数据 0x7 (0111111B) 并输出，使 P1.7 指示灯亮，P1.0 指示灯熄灭。否则，即读不出低电平时，向 P1 送入数据 0xfe (11111110B) 并输出，使 P1.7 指示灯熄灭，P1.0 指示灯被点亮。
- 通过 while(1)语句不断地循环检测开关 K 的状态。

2. 程序

C 语言编写的源程序 kg19-1.c 代码如下：

```

01 #include <AT89X51.H>          /* 头文件 */
02 main( )                      /* 主函数 */
03 {
04     unsigned char K;           /* 声明无符号字符型变量 K */
05     while(1)                  /* 无限循环 */
06     {
07         K=P3_0;                /* 读入 P3_0 状态 */
08         if ( K==0 )            /* 如果开关 K 接通 */
09         {
10             P1=0x7f;            /* 输出 0x7f */
11         }
12     else                      /* 否则 */
13     {
14         P1=0xfe;              /* 输出 0xfe */
15     }
16 }
17 }
```

19.1.3 代码详解

- 01：头文件。
- 02：主函数 (02 行 ~ 17 行)。
- 03：主函数开始。
- 04：声明无符号字符型变量 K，用于存放 P3.0 状态值。
- 05：无限循环 (05 行 ~ 16 行)。
- 06：循环体开始。
- 07：读入 P3_0 状态，并赋值给 K。
- 08 ~ 16：通过 if-else 选择语句进行两种选择。其中第 08 行语句判断开关 K 是否处

在接通状态，如果 K 值等于 0 说明开关 K 接通，于是程序执行第 10 行语句，使 P1 输出 0x7f (01111111B)，即 P1.7 指示灯亮，用来显示开关 K 接通状态；否则，程序执行第 14 行语句，使 P1 输出 0xfe (11111110B)，点亮 P1.0 指示灯，用来显示开关 K 处于断开状态。

19.1.4 经验总结

本节程序实现使用两个指示灯反映一个开关的两种状态，是一个简单的开关输入程序。开关只有两种状态，接通或断开，当将开关接在单片机输入端口时，开关接通时输入端口为低电平；开关断开时输入端口为高电平。所以通过检测单片机输入端口电平的高低，便可知开关状态。

程序中是通过先读出输入端口状态值并赋值于开关“K=P3_0;”，再判断开关值是否等于 0，即“if (K==0)”，从而获得开关的输入信号。P3.0 为单片机的输入端，P1.0 和 P1.7 为输出端。

19.2 多路开关输入状态指示灯

功能说明：4 位指拨开关 SW1、SW2、SW3、SW4 分别与 P3.0、P3.1、P3.2、P3.4 相连，作为单片机的输入信号，并由 P1 端口的 8 只 LED 显示开关状态。SW1 接通时 P1 端口 1 个灯亮；SW2 接通时 P1 端口 2 个灯亮；SW3 接通时 P1 端口 3 个灯亮；SW4 接通时 P1 端口 4 个灯亮。因此只要通过观察亮灯情况便知道指拨开关的输入处状态。

19.2.1 硬件设计

硬件设计如图 19.2 所示。

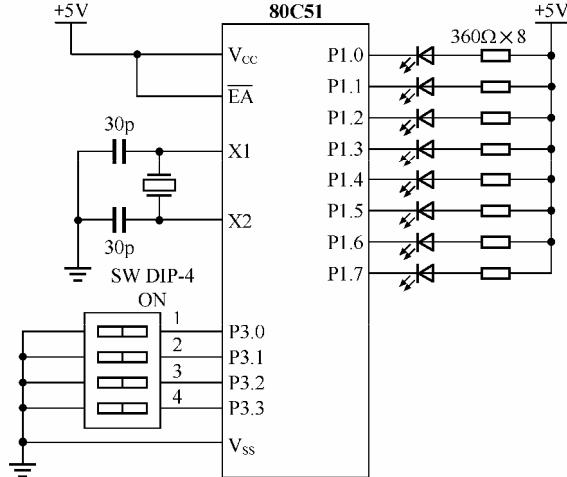


图 19.2 DIP 开关输入基本电路

4个指拨开关SW1、SW2、SW3和SW4的一端分别与单片机P3.0、P3.1、P3.2和P3.3端口相接，指拨开关的另一端接地。若将指拨开关拨到“ON”的一边，则接点接通，拨到另一边则为不通。单片机的P1端口接有8只LED，用来显示开关状态。

19.2.2 程序设计

1. 工作原理

(1) 首先将输入状态编码。

只有 SW1 为 ON 时，P3=11111110B=0xfe。

只有 SW2 为 ON 时，P3=11111001B=0xf9。

只有 SW3 为 ON 时，P3=11000111B=0xc7。

只有 SW4 为 ON 时，P3=00001111B=0x0f。

1

输入 P3	输出 P1
0xfe	0xfe (1个灯亮)
0xfd	0xf9 (2个灯亮)
0xfb	0xc7 (3个灯亮)
0xf7	0x0f (4个灯亮)

其中，输出 P1 码可任意编制，它只是表示依据输入 P3 的变化而产生的相应变化。

(3) 运用 C51 语言中的 switch-case 语句 , 该语句的最大功能是依据变量的变化来产生相应动作 . 它具有多重选择的功能 .

2 程序

C 语言编写的多路开关输入状态指示灯源程序 kg19-2.c 代码如下：

```
01 #include <AT89X51.H>          /* 头文件 */
02 main( )                      /* 主函数 */
03 {
04     unsigned char sw;          /* 声明无符号字符型变量 sw */
05     while(1)                  /* 无限循环 */
06     {
07         sw=P3|0xf0;           /* 读入 P3 的状态 */
08         switch(sw)            /* 判断 SW 的状态 */
09         {
10             case 0xfe: P1=0xfe; /* 只有 SW1 为 ON 时，输出 0xfe */
11                 break;        /* 离开 switch-case 循环 */
12             case 0xfd: P1=0xf9; /* 只有 SW2 为 ON 时，输出 0xf9 */
13                 break;        /* 离开 switch-case 循环 */
```

```

14     case 0xfb: P1=0xc7;      /* 只有 SW3 为 ON 时，输出 0xc7 */
15             break;           /* 离开 switch-case 循环 */
16     case 0xf7: P1=0x0f;      /* 只有 SW4 为 ON 时，输出 0x0f */
17             break;           /* 离开 switch-case 循环 */
18     default:   P1=0xff;      /* SW 的状态都不是上述 4 种情形，
19                           则输出 0xff */
20             break;           /* 离开 switch-case 循环 */
21     }
22   }
23 }
```

19.2.3 代码详解

01：头文件。

02：主函数（02 行～23 行）。

03：主函数开始。

04：声明无符号字符型变量 SW。

05：无限循环（05 行～22 行）。

06：循环体开始。

07：读入 P3 状态。由于 SW 开关是接在 P3 的 P3.0～P3.3 上，即 P3 的低 4 位，而 P3.4～P3.7 高 4 位并不包括在 SW 的状态之内。所以读入 P3 状态时，先将读入的数据通过按位或处理，屏蔽高 4 位，只取低 4 位有效值。

08～20：通过 switch-case 语句进行多分支选择，如第 10 行语句只有 SW1 为 ON 时，其输入状态码为 0xfe，输出才能为 0xfe（11111110B），此时会有 1 个灯亮起。接着程序执行第 11 行 break 语句，退出 switch 选择。下面的第 12 行～17 行语句与上同。

第 18 行～20 行语句的意思是当 SW 的状态都不是上述 4 种情形，则输出 0xff（11111111B），即灯全部熄灭，并离开 switch-case 循环。

21：switch 语句结束。

22：while(1) 循环体结束。

23：主函数结束。

19.3 多路开关控制灯

功能说明：4 位指拨开关 SW1、SW2、SW3 和 SW4 作为单片机的输入信号，控制输出端口 P1.7 所接的 LED 亮与灭。只有当 SW1、SW2、SW3、SW4 同时拨到 ON 的一边，即 4 个指拨开关都接通时 P1.7 灯才能亮起，否则灯处于熄灭状态。

19.3.1 硬件设计

硬件设计如图 19.3 所示。

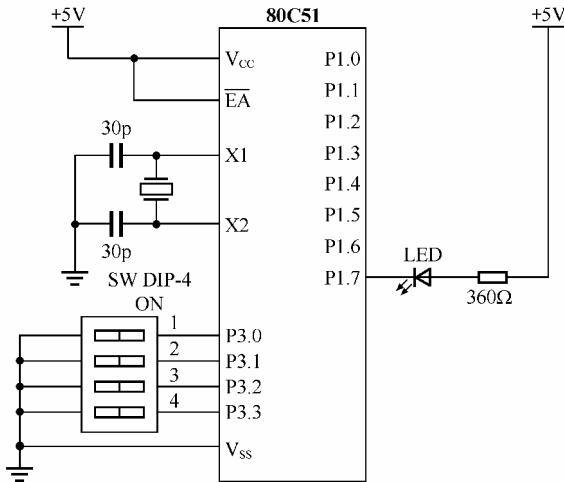


图 19.3 4 位指拨开关控制 LED

19.3.2 程序设计

1. 工作原理

(1) SW 开关拨到 ON 时，相对应的位为 0，所以通过判断位是否为 0，就可以判断开关状态。

(2) SW1、SW2、SW3 和 SW4 同时拨到 ON 时，相对应的位都为 0，此时经逻辑或运算：
 $SW1 \parallel SW2 \parallel SW3 \parallel SW4 = 0$ ，即结果为 0。

(3) 只有当逻辑或运算结果为 0 时，if 表达式成立，则 P1 输出 0x7f (01111111B)，使 P1.7 所接的灯被点亮；否则，P1 输出 0xff (11111111B)，使 P1.7 灯熄灭。

2. 程序

C 语言编写的多路开关控制灯源程序 kg19-3.C 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 main( )                      /* 主函数 */
03 {
04     unsigned char sw1,sw2,sw3,sw4; /* 声明无符号字符型变量 sw1 ~ sw4 */
05     while(1)                     /* 无限循环 */
06     {
07         sw1=P3_0;
08         sw2=P3_1;
09         sw3=P3_2;
10         sw4=P3_3;
11         if ((sw1||sw2||sw3||sw4)==0)    /* sw1、sw2、sw3、sw4 都拨到 NO */
12             {
13                 P1=0x7f;            /* 输出 0x7f，P1.7 灯亮 */

```

```

14      }
15  else                         /* 否则 , 执行第 17 行语句 */
16  {
17      P1=0xff;                  /* 输出 0xff , P1.7 灯灭 */
18  }
19 }
20 }
```

19.3.3 代码详解

01 : 头文件。

02 : 主函数 (02 行 ~ 20 行)。

03 : 主函数开始。

04 : 声明无符号字符型变量 SW1、SW2、SW3、SW4。

05 : 无限循环 (05 行 ~ 19 行)。

06 : 循环体开始。

07 ~ 10 : P3.0 ~ P3.3 位的电平反映 SW1 ~ SW4 开关状态。位电平为 0 时 , 说明 SW 开关是在接通位置 , 位电平不是 0 时 , 说明 SW 开关是在断开位置。

11 ~ 18 : if-else 语句。如果 sw1、sw2、sw3、sw4 都拨到 ON 位置 , if 表达式成立 , 程序将执行第 13 行语句 , P1 输出 0x7f (01111111B) , 点亮 P1.7 所接的灯。否则 , 执行第 17 行语句 , P1 输出 0xff (11111111B) , 使 P1.7 所接的灯熄灭。

19 : while(1) 循环体结束。

20 : 主函数结束。

19.4 按钮开关次数显示灯

功能说明 : 计算按钮开关被按的次数。按钮开关 K1 作为单片机的输入信号 , 开关 K1 被按的次数由 P1 端口所接的 8 只 LED 来显示。8 只 LED 可看作 8 位二进制数 , 亮的为 0 , 灭的为 1。在观察 LED 计数时可参照书后附录 B , 将二进制数转换十进制数。

19.4.1 硬件设计

电路设计如图 19.4 所示。

按钮开关 K1 与 P3 端口中的 P3.2 引脚连接 , 作为信号的输入部分 , 信号的输出由 P1 端口所接的 8 只 LED 显示。

19.4.2 程序设计

1. 工作原理

(1) 按钮开关 K1 的信号在未按下时输出为高电平 , 按下时为低电平。所以 , 按下 K1 后再放开 K1 就会产生一个负向脉冲输入到 P3.2 引脚。

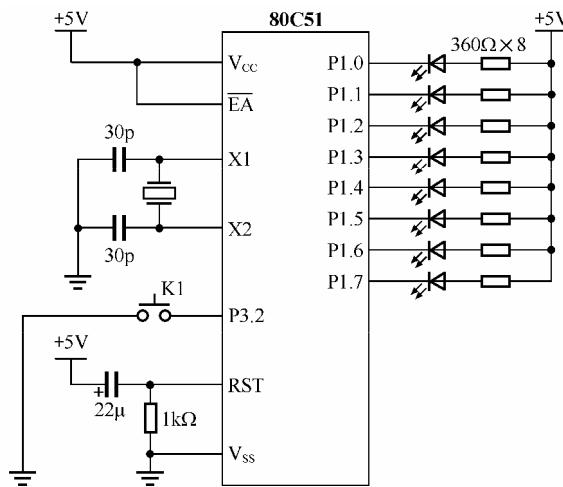


图 19.4 按钮开关输入电路图

(2) 若要检测按钮开关 K1 被按下后又被放开，必须先检测到 P3.2 为 0，然后再检测到 P3.2 为 1，即表示 K1 按下又放开（为按动一次），并将按钮被按动的次数存入变量 set。

(3) 判断按钮按动一次所使用的语句如下：

```
while(P3_2); /* 直到 K1 被按下才执行下一个语句 */
while(!P3_2); /* 直到 K1 被放开才执行下一个语句 */
```

运用上述两条语句，可以完成对按钮开关 K1 输入信号的检测。

(4) 使用消除按键抖动的语句来消除按键抖动对信号读取的影响。

2. 程序

C 语言编写的按钮开关次数显示灯源程序 kg19-4.c 代码如下：

```
01 #include <AT89X51.H> /* 头文件 */
02 main( ) /* 主函数 */
03 {
04     unsigned char set; /* 声明无符号字符型变量 set */
05     unsigned int i; /* 声明无符号整型变量 i */
06     set=0; /* 初始计算值为 0 */
07     P1=0; /* 初始值为 0 */
08     while(1) /* 无限循环 */
09     {
10         P1=set; /* 将计算值送入到 P1 输出 */
11         while(P3_2); /* 直到按键被按下才执行下一条语句 */
12         while(!P3_2); /* 直到按键被放开才执行下一条语句 */
13         for(i=0;i<20000;i++); /* 延时一段时间，消除抖动 */
14         ++set; /* 计数值增加 1 */
15     }
16 }
```

19.4.3 代码详解

01：头文件。
 02：主函数（02~14行）。
 03：主函数开始。
 04：声明无符号字符型变量 set。
 05：声明无符号整型变量 i。
 06：设置变量 set 的初始计算值为 0。
 07：设置 P1 的初始值为 0，即灯全亮。
 08：无限循环。
 09：循环体开始。
 10：P1 输出计数值，由 8 只 LED 组成二进制数。
 11、12：判断按钮是否被按下而后又放开，如果按钮开关 K1 没被按下，程序运行将停留（维持）在第 11 行语句处。如果按钮按下而没被放开，则程序将停留在第 12 行语句处。只有按下后又放开，程序才能向下运行。
 13：延时一段时间，消除按键抖动对信号读取的影响。
 14：每按动一次按钮开关 K1，变量 set 计数值增加 1。
 15：while(1)循环体结束。
 16：主函数结束。

19.5 一键多功能控制

功能说明：使用一个按钮开关 K1 作为输入信号，控制 P1 输出端口 8 只 LED，实现 4 种变化效果。本节硬件设计与上节相同

19.5.1 程序设计

1. 工作原理

(1) 通过对按键不断扫描来检测按键被按动的次数，并将检测到的按键次数存入变量 set 中。

(2) 由于只需要控制输出端 4 种动作，所以按键的次数有效值范围设定为 0~4，当按到 5 次时又重新归 0。

(3) 运用 switch-case 多分支选择语句，根据变量 set 中存有的按键次数值，执行不同功能。

2. 程序

C 语言编写的一键多功能控制源程序 kg19-5.c 代码如下：

```
01 #include <AT89X51.H>           /* 头文件 */
```

```

02 void scan_k1( );           /* 声明 void scan_k1()函数 */
03 unsigned char set;         /* 声明无符号字符型变量 set */
04 void main( )              /* 主函数 */
05 {
06     while(1)               /* 无限循环 */
07     {
08         if(P3_2==0)scan_k1(); /* 如果按下 k1，则调用按键扫描函数 */
09         switch(set)
10         {
11             case 1: P1=0x66;    /* 按键 1 次时，输出 0x66 */
12                 break;        /* 离开 switch-case 循环 */
13             case 2: P1=0xf0;    /* 按键 2 次时，输出 0xf0 */
14                 break;        /* 离开 switch-case 循环 */
15             case 3: P1=0x0f;    /* 按键 3 次时，输出 0x0f */
16                 break;        /* 离开 switch-case 循环 */
17             case 4: P1=0x00;    /* 按键 4 次时，输出 0x00 */
18                 break;        /* 离开 switch-case 循环 */
19             default: P1=0xff; /* 不是上述 4 种情形，则输出 0xff */
20                 break;        /* 离开 switch-case 循环 */
21         }
22     }
23 }
24
25 void scan_k1( )           /* 按键次数扫描函数 */
26 {
27     unsigned int i;          /* 声明无符号整型变量 i */
28     if(P3_2==0)set++;        /* 如果 K1 按下，计数值增加 1 */
29     if(set>=5)set=0;        /* 按 5 次，计数值归 0 */
30     F0:if(P3_2==0)goto F0;  /* 键未放开时，一直在此循环 */
31     for(i=0;i<20000;i++);  /* 延时一段时间，消除抖动 */
32 }
```

19.5.2 代码详解

本程序由主函数和按键扫描函数所组成。

04 ~ 23：主函数部分。主函数部分主要是运用 switch-case 多分支选择语句，根据按键次数执行不同功能。

这部分与第 19.2 节程序 kg19.2.c 基本相同，只是增加了第 08 行语句“ if(P3_2==0) scan_k1(); ”，意思是如果按下 K1 则调用按键扫描函数 scan_k1()，以便确定按键的次数。主函数根据按键的次数，即 set 值来执行不同的功能语句。

25 ~ 32：按键扫描函数。

其中第 28 行为 if-else 语句，但是省略了 else，而且由于 if 后面的{ }内只有一个语句“ set++ ”，所以可以省略大括号，写为“ if(P3_2==0)set++; ”，意思是如果 P3.2=0，即按一下

K1 键，计数值增加 1。

第 30 行语句中 F0 为语句标号，goto 为无条件转向语句，执行 goto 操作后程序运行将无条件地跳转到标号 F0 语句处执行，也就是说，按键未放开时程序一直在此循环。只有当按键放开时，if 的条件表达式不能成立，程序才能向下执行第 31 行延时语句，延时一段时间消除抖动。

32：按键扫描函数 scan_k1()结束。



第 20 章 报警声设计

报警声在自动化设备的设计中会经常用到，本章通过一些实例小程序，介绍单片机发声原理及报警声设计方法。

20.1 发出 1kHz 声音

功能说明：通过单片机的 I/O 引脚以软件延时的方式产生方波，使蜂鸣器发出 1kHz 的声音。

20.1.1 硬件设计

电路设计如图 20.1 所示。

单片机的 P3.4 引脚通过限流电阻 R 与三极管基极相接，三极管的集电极接有蜂鸣器。P2.7 引脚上接有按钮开关 K，开关 K 的另一端接地。

20.1.2 程序设计

1. 工作原理

若使蜂鸣器发出 1kHz 的声音，则需要使 P3.4 引脚产生 1kHz 的方波，来控制晶体管不断地 ON/OFF，从而使蜂鸣器发出声音。1kHz 的方波信号周期为 1ms，高电平占用 0.5ms，低电平占用 0.5ms，通过调用软件延时程序来实现。

在设计时，将开关 K 作为发声控制按钮，当按下 K 后才能发出声音。

2. 程序

C 语言编写的发声实验源程序 fs20-1.c 代码如下：

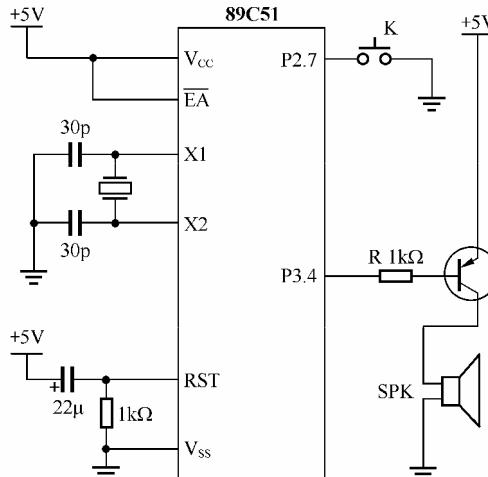


图 20.1 发声实验电路

```
01 #include<AT89X51.h> /* 头文件 */
```

```

02 void delay250( unsigned int k );      /* 声明延时函数 */
03 main( )                           /* 主函数 */
04 {
05     for (;;)                      /* 无限循环 */
06     {
07         while(P2_7==0)           /* 按钮 K 按下，执行 while 循环 */
08         {
09             P3_4=0;              /* P3.4=0，晶体管导通 */
10             delay250(2);        /* 调用延时函数，延时 500 μs */
11             P3_4=1;              /* P3.4=1，晶体管截止 */
12             delay250(2);        /* 调用延时函数，延时 500 μs */
13         }
14     }
15 }
16
17 /*延时为：250 μs*k */
18 void delay250(unsigned int k)          /* 延时函数 */
19 {
20     unsigned int j,i;
21     for(i=0;i<k;i++)
22     {
23         for(j=0;j<25;j++);
24     }
25 }
```

20.1.3 代码详解

本程序由主函数和延时函数两部分所组成。

03 ~ 15：主函数部分。

其中第 05 行语句为无限循环语句，在循环体里又包含“while(P2_7==0)”循环语句，只有当按钮开关 K 被按下，使 P2.7=0 时，while 循环才能运行。

循环体里第 09 行语句使 P3.4 引脚为低电平，此时晶体管导通，有电流流经蜂鸣器，并通过第 10 行语句调用延时函数，使 P3.4 引脚低电平保持 0.5ms。

之后运行第 11 行语句，使 P3.4 引脚为高电平，此时晶体管截止，没有电流流经蜂鸣器，并通过第 12 行语句保持高电平 0.5ms。就这样使通过蜂鸣器的电流一会有，一会无，使得蜂鸣器发出声音。

如果开关 K 没有被按下，此时 P2.7=1，故 while 循环不能进行，蜂鸣器不能发出声音。

18 ~ 25：延时函数。延时的时间为 0.5ms。

20.2 发出嘀、嘀声

功能说明：使单片机产生嘀、嘀声音。本节硬件设计与上节相同。

20.2.1 程序设计

1. 工作原理

使 P3.4 引脚产生 1kHz 的方波，先响 0.2s，然后断 0.2s。开关 K 作为发声控制按钮，当按下 K 后才能发出声音。

2. 程序

C 语言编写的发出嘀、嘀声源程序 fs20-2.c 代码如下：

```

01 #include<AT89X51.h>           /* 头文件 */
02 void delay250( unsigned int k ); /* 声明延时函数 */
03 main( )                      /* 主函数 */
04 {
05     for ( ; ; )                /* 无限循环 */
06     {
07         while(P2_7==0)          /* 按钮 K 按下，执行 while 循环 */
08         {
09             unsigned int j;
10             for(j=0;j<400;j++)      /* 循环 400 次，嘀响 0.2s */
11             {
12                 P3_4=0;            /* P3.4=0，晶体管导通 */
13                 delay250(2);       /* 调用延时函数，延时 500 μs */
14                 P3_4=1;            /* P3.4=1，晶体管截止 */
15                 delay250(2);       /* 调用延时函数，延时 500 μs */
16             }
17             delay250(800);        /* 停 0.2s */
18         }
19     }
20 }
21 /* 延时为：250 μs*k */
22 void delay250(unsigned int k)    /* 延时函数 */
23 {
24     unsigned int j,i;
25     for(i=0;i<k;i++)
26     {
27         for(j=0;j<25;j++);
28     }
29 }
30 }
```

20.2.2 代码详解

本程序是在上节 fs20-1.c 程序的基础上，增加了第 10 行语句 “`for(j=0;j<400;j++)`” 和第

17 行语句“ delay250(800); ”。其中，第 10 行语句使用 for 将 1kHz 发声循环执行 400 次，使发声的时间为 0.2s，然后又经第 17 行语句使发声停止 0.2s。

在按下按钮开关 K 使 P2_7=0 时，上述过程经 while 语句不断循环，使蜂鸣器发出嘀、嘀声音效果。

程序中延时函数为有参数输入函数。延时函数的延时时间取决于参数 K 的值，当 K 值为 2 时延时 500μs；当 K 值为 800 时延时 0.2s。

其他语句作用与上节程序详解相同。

20.2.3 经验总结

本节程序中，第 13 行和第 15 行语句的延时时间决定发声频率，发声频率的高低决定音调的高低，频率高，则音调高；频率低，则音调低。

而发声时间的长短是指发声的持续时间的长短，如在本节程序第 10 行语句中，J 数值大时，则发出深长的声音，J 数值小时则发出急促短暂的声音。

20.3 救护车声

功能说明：由单片机发出救护车声音，本节硬件设计与 20.1 节相同。

20.3.1 程序设计

1. 工作原理

通过软件延时，使 P3.4 引脚输出 1kHz 和 2kHz 的变频信号，每隔 1s 交替变化 1 次。开关 K 作为发声控制按钮，当按下 K 后才能发出声音。

2. 程序

C 语言编写的救护车声源程序 fs20-3.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 void delay250(unsigned int k);   /* 声明延时函数 */
03 void main( )                   /* 主函数 */
04 {
05     unsigned int i,j;          /* 声明无符号整型变量 i、j */
06     {
07         for(;;)                /* 无限循环 */
08         {
09             while( P2_7==0 )      /* 按钮 K 按下，执行 while 循环 */
10             {
11                 for(i=0;i<2000;i++)    /* 循环 2000 次，1kHz 响 1s */
12                 {
13                     P3_4= ~ P3_4;        /* 反相 */

```

```

14         delay250(2);           /* 延时 500 μs */
15     }
16     for(j=0;j<4000;j++)      /* 循环 4000 次，2kHz 响 1s*/
17     {
18         P3_4=~P3_4;           /* 反相 */
19         delay250(1);        /* 延时 250 μs */
20     }
21 }
22 }
23 }
24 }
25
26 /*延时时间为：250 μs×k */
27 void delay250(unsigned int k)    /* 延时函数 */
28 {
29     unsigned int j,i;
30     for(i=0;i<k;i++)
31     {
32         for(j=0;j<25;j++);
33     }
34 }
```

20.3.2 代码详解

本程序中第 13 行、14 行语句使 P3.4 引脚输出 1kHz 方波，并通过第 11 行语句使 1kHz 声音持续 1s。

第 18、19 行语句使 P3.4 引脚输出 2kHz 方波，再通过第 16 行语句使 2kHz 声音持续 1s。当按下开关 K 时，通过第 09 行 while 语句使 1kHz 和 2kHz 的变频信号每隔 1s 交替变化 1 次，不断循环。

第 07 行语句为 for 无限循环语句，其循环体为第 08 行 ~ 22 行。

27 ~ 34 延时函数。

20.4 闹钟铃声

功能说明：使单片机实现发出闹钟铃声。本节硬件设计与 20.1 节相同。

20.4.1 程序设计

1. 工作原理

使 P3.4 引脚输出 1kHz 和 500Hz 的变频信号。1kHz 发声长为 0.075s，500Hz 发声长为

0.1s，交替变化。开关 K 作为发声控制按钮，当按下 K 后才能发出声音。

2. 程序

C51 语言编写的闹钟铃声源程序 fs20-4.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 void main( )                  /* 主函数 */
03 {
04     unsigned char i,j;        /* 声明无符号字符型变量 i、 j */
05     {
06         for(;;)                /* 无限循环 */
07     {
08         while( P2_7==0 )       /* 按钮 K 按下，执行 while 循环 */
09     {
10         for(i=0;i<150;i++)    /* 循环 150 次，1kHz 音长为 0.075s */
11         {
12             P3_4=~P3_4;        /* 反相输出 */
13             for(j=0;j<50;j++); /* 延时 500 μs */
14         }
15         for(i=0;i<100;i++)    /* 循环 100 次，500Hz 音长为 0.1s */
16         {
17             P3_4=~P3_4;        /* 反相输出 */
18             for(j=0;j<=100;j++); /* 延时 1000 μs */
19         }
20     }
21 }
22 }
23 }
```

20.4.2 代码详解

本程序中第 12 行、 13 行语句使 P3.4 引脚输出 1kHz 方波，并通过第 10 行语句使 1kHz 声音持续 0.075s。

第 17 行、 18 行语句使 P3.4 引脚输出 500Hz 方波，再通过第 15 行语句使 500Hz 声音持续 0.1s。

当按下开关 K 时，通过第 08 行 while 语句使 1kHz 和 500Hz 的变频信号交替变化，不断循环。第 06 行语句为 for 无限循环语句，其循环体为第 07 ~ 21 行语句。

20.4.3 经验总结

本节程序与上节程序结构基本相同，都是通过两个不同频率交替发声，产生相应的发声效果，程序 fs20-3.c 产生 1kHz 和 2kHz 的变频信号，发声长度都为 1s。

程序 fs20-4.c 产生 1kHz 和 500Hz 的变频信号，1kHz 发声长为 0.075s，500Hz 发声长为 0.1s。

不同频率、不同发声长度的交替变化，就可以产生不同的报警声音效果。

20.5 发出 20 次的报警声

功能说明：单片机实现发出 20 次报警声。本节硬件设计与 20.1 节相同。

20.5.1 程序设计

1. 工作原理

利用 3 层嵌套 for 循环，内循环产生方波为音调，中循环决定音长，外循环为发出声音的次数。

2. 程序

C 语言编写的发出 20 次的报警声源程序 fs20-5.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 void delay1ms(unsigned int k);   /* 延时函数声明 */
03 void main( )                   /* 主函数 */
04 {
05     unsigned int i,j,k;         /* 声明无符号整数变量 i、j、k */
06     for(k=0;k<20;k++)          /* 发声次数循环 */
07     {
08         for(i=0;i<160;i++)      /* 音长的循环 */
09         {
10             P3_4=~P3_4;          /* 反相输出 */
11             for(j=0;j<100;j++);  /* 半波，延时 1ms */
12         }
13         delay1ms(200);          /* 停留一段时间 */
14     }
15 }
16 void delay1ms(unsigned int k)    /* 延时函数 */
17 {
18     unsigned int j,i;
19     for(i=0;i<k;i++)
20         for(j=0;j<100;j++);
21 }
```

20.5.2 代码详解

本程序利用 3 层嵌套 for 循环。

06：发声的次数循环，循环 20 次使程序发出 20 次报警响声。

08：声的音长循环，循环 160 次，使音长为 80 个方波，160ms。

10 ~ 11：产生声的音调。P3.4 反相输出一次为方波的半个周期，时间为 1ms，循环两次为一个方波，即一个方波的周期时间为 2ms，所以，产生的方波频率为 500Hz。频率的高低

决定音调的高低。

13：每响一声后，停留 200ms。

16~21：延时函数，延时的时间为 $1\text{ms} \times k$ 。

20.5.3 经验总结

音调、音长、发出声音的次数是 3 个不同概念，有不同的作用效果。本程序中都是利用 for 循环语句来实现的，所以，要注意 for 语句在不同的地方所起的不同作用。

- 音调是由发声频率决定的。本节程序中第 11 行 for 循环语句决定音调。
- 音长是指连续发声的时间。本节程序中第 08 行 for 循环语句决定音长。
- 发声的次数是指响几次报警声而言。本节程序中第 06 行 for 循环语句决定发声的次数。

20.6 警报的同时 LED 闪烁

功能说明：在发出警报声的同时，LED 也闪烁。

20.6.1 硬件设计

电路设计如图 20.2 所示。

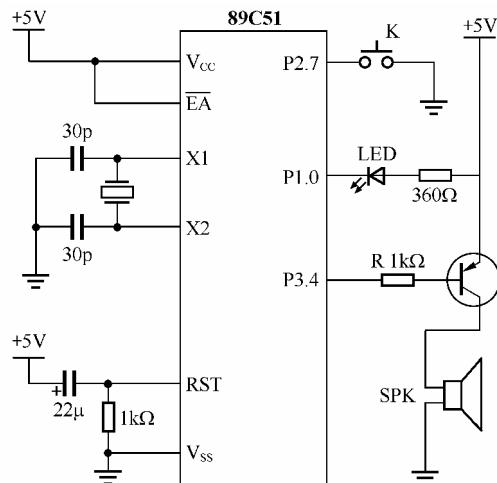


图 20.2 警报同时 LED 闪烁电路

在图 20.1 的基础上，将单片机 P1.0 引脚接有发光二极管 LED，LED 正极通过限流电阻 R 接电源的正极。当 P1.0 输出低电平时 LED 导通发光，其他部分与图 20.1 说明相同。

20.6.2 程序设计

1. 工作原理

本程序是对上节程序 fs20-5.c 功能的扩充，并将上节程序中的音调、音长、发声次数的

固定数值分别利用 tone、sound、count 3 个参数来代替，使之在不同的应用中只需通过改变参数就能达到各种需求，无需再重新编写程序。

为了加强警报效果，在警报的同时增加了 LED 的闪烁。

2. 程序

C 语言编写的警报的同时 LED 闪烁源程序 fs20-6.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 void delay1ms(unsigned int k);    /* 延时函数声明 */
03 void main( )                   /* 主函数 */
04 {
05     unsigned int i,j,k;          /* 声明无符号整数变量 i、j、k */
06     unsigned int count,sound,tone; /* 声明参数变量 */
07     count=20;                   /* 对 count 参数（发声次数）赋值 */
08     sound=160;                  /* 对 sound 参数（音长）赋值 */
09     tone=100;                   /* 对 tone 参数（音调）赋值 */
10     for(k=0;k<count;k++)        /* 发声次数循环 */
11     {
12         P1_0=~P1_0;             /* 反相输出，使 LED 闪烁 */
13         for(i=0;i<sound;i++)   /* 音长循环 */
14         {
15             P3_4=~P3_4;           /* 反相输出，产生方波 */
16             for(j=0;j<tone;j++); /* 音调循环 */
17         }
18         delay1ms(200);          /* 停留一段时间 */
19     }
20 }
21
22 /* 延时时间为：1ms × k */
23 void delay1ms( unsigned int k) /* 延时函数 */
24 {
25     unsigned int j,i;
26     for(i=0;i<k;i++)
27         for(j=0;j<120;j++);
28 }
```

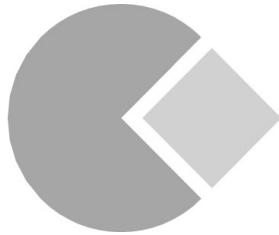
20.6.3 代码详解

01：头文件。

02：延时函数声明。该函数的函数名为 delay1ms，函数名前面的 void 是返回值类型标识符，该标识符说明是无返回值的函数。函数名后边括号内的 k 为参数。“ unsigned int k ” 声明参数 k 是无符号整数变量。

03：主函数。

- 04 : 主函数开始。
- 05 , 06 : 声明无符号整数变量 i、j、k 和参数 count、sound、tone。
- 07 : 对发声次数参数 count 赋值。
- 08 : 对音长参数 sound 赋值。
- 09 : 对音调参数 tone 赋值。
- 10 : 发声的次数循环。由于传递的参数 count 的值为 20 , 所以将循环 20 次 , 即程序将发出 20 次警报响声。
- 12 : 发声的次数每循环一次 , P1.0 将反相输出一次 , 使警报的同时 LED 闪烁。
- 13 : 声的音长循环。传递的参数 sound 的值为 160 次 , 音长为 160ms。
- 15、16 : 产生声的音调。反相输出一次为方波的半个周期 , 时间为 1ms。循环两次为一个方波 , 其周期的时间为 2ms , 所产生的方波频率 (1/周期) 为 500Hz。频率的高低决定音调的高低。
- 18 : 停留一段时间。
- 23 ~ 28 : 延时函数。



第 21 章 交通灯信号控制

本章在交通灯信号控制设计中，综合利用了单片机的定时器功能和多种中断功能。

21.1 采用定时器延时

功能说明：利用单片机内的定时器功能进行延时，使接在 P1.5 引脚上的 LED 点亮后连续亮 10s 再熄灭，熄灭 3s 后再被点亮，不断循环。

21.1.1 硬件设计

电路设计如图 21.1 所示。

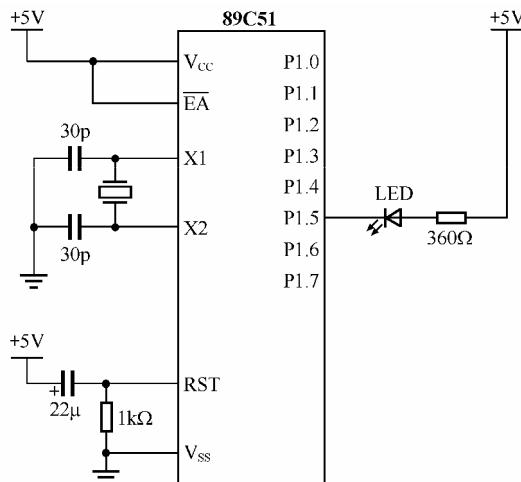


图 21.1 实验电路

单片机实验电路的左侧部分为单片机最小系统电路：5V 直流电源、单片机时钟电路和自动复位电路。右侧 P1 端口中 P1.5 引脚接有 LED 用作程序功能显示。

21.1.2 程序设计

1. 工作原理

(1) 首先设置定时器工作模式。

```
TMOD=0x01;           /* 设定 T0 为模式 1 */
```

(2) 设置定时器初始值。当定时器定时时间为 50ms 时，其初始值如下。

```
TH0=-(50000/256);      /* 定时初始值为 50000 */
TL0=-(50000%256);
```

(3) 启动定时器。

```
TR0=1;
```

(4) 检测定时器溢出标志 TF0。

```
while(TF0!=1);          /* TF0 为 1 则表示定时时间到了 */
```

每次定时时间为 50 ms，定时 20 次延时的时间为 1s。

2. 程序

C51 语言编写的使用定时器延时源程序 JT 21-1.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 void delay50ms(unsigned int i);   /* 声明延时函数 delay50ms() */
03 /* ----- 主函数 ----- */
04 main( )                      /* 主函数 */
05 {
06     TMOD=0x01;                /* 设定 T0 为模式 1 */
07     for( ; ; )                /* 无限循环 */
08     {
09         P1=0xdf;              /* 输出 */
10         delay50ms(200);       /* 调用延时函数，延时 10s */
11         P1=0xff;              /* 输出 */
12         delay50ms(60);        /* 调用延时函数，延时 3s */
13     }
14 }
15 /* ----- 延时函数 ----- */
16 void delay50ms(unsigned int i)  /* 延时时间为 50ms×i */
17 {
18     TR0=1;                   /* 启动定时器 */
19     while(i!=0)              /* 执行 i 次循环 */
20     {
21         TH0=-(50000/256);    /* 定时器初始值 50000 */
```

```

22     TL0=-(50000%256);
23     while(TF0!=1);           /* 等待计数终了 */
24     TF0=0;                  /* 清除 TF0 */
25     i--;                   /* 循环数-1 */
26   }
27   TR0=0;                  /* 关闭定时器 */
28 }

```

21.1.3 代码详解

01：头文件。

02：声明延时函数 delay50ms()。该函数是有参数输入的函数，参数为 i，并在函数的括号内声明为无符号整型变量。

04：主函数。

05：主函数开始。

06：TMOD 是定时器/计数器模式控制寄存器，赋值 0x01，设置定时器 T0 工作模式 1。

07：无限循环。

08：for 语句循环开始。

09：P1 输出 0xdf (11011111B)，P1.5 灯被点亮。

10：调用延时函数 delay50ms (200)，延时时间为 10s。

11：P1 输出 0xff (11111111B)，灯被熄灭。

12：调用延时函数 delay50ms (60)，延时时间为 3s。

13：for 语句循环结束。

14：主函数结束。

16：延时函数，延时时间为 50ms×i

17：延时函数开始。

18：启动 T0 定时器。

19：执行 i 次循环，当 i=20 时，即循环 20 次。

20：while 循环开始。

21：设置定时器 T0 初始值，向 TH0 寄存器装入初始值 50000。

22：设置定时器 T0 初始值，向 TL0 寄存器装入初始值 50000。

23：等待计数终了。TF0 为定时器溢出标志，当定时器溢出，即定时时间到了，此时 TF0=1。所以 TF0 不为 1 时说明定时器定时的时间还没到，程序在此等待，只有 TF0=1 时程序才向下运行。

24：当定时器溢出，即 TF0 = 1 后，把 TF0 位清 0，为下次定时做准备。

25：每定时一次，即循环 1 次，i 值减 1。

26：while 循环结束。

27：关闭定时器。

28：延时函数结束。

21.1.4 经验总结

本节程序采用定时器作延时，定时器延时最大优点是延时的时间准确度高，所以，对延时时

间要求严格的地方一般使用定时器延时。同时，定时器延时不占用CPU资源，能提高单片机效率。

定时器延时的时间是有一定限制的，当单片机CPU时钟为12MHz时，最多（采用模式1）延时的时间为 $65536\mu s$ 。但是可以通过与软件相结合的办法延长时间，例如在本节程序中将定时器连续定时60次，使延时时间达到3s，再将延时函数的参数i改为200，则延时的时间增加到10s。

21.2 灯交互闪烁

功能说明：利用定时器中断功能，使P1端口所接8只LED中的左侧4只与右侧4只交替闪烁，间隔时间为0.5s。

21.2.1 硬件设计

硬件设计如图21.2所示。

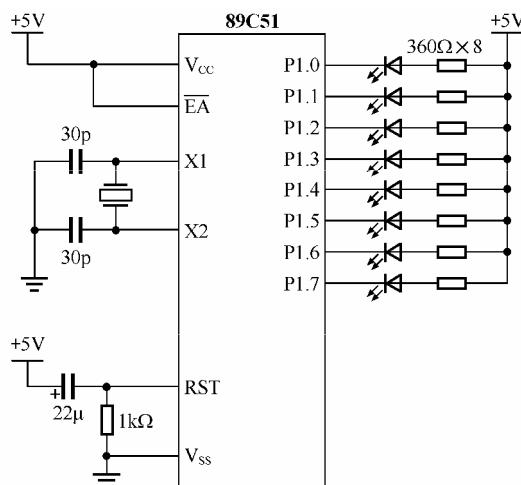


图21.2 LED交互闪烁实验电路

单片机P1端口接有8只LED，其中P1.0、P1.1、P1.2和P1.3所接的LED为右侧4只；P1.4、P1.5、P1.6和P1.7所接的LED为左侧4只。

21.2.2 程序设计

1. 工作原理

利用定时器的中断功能，当定时的时间到后便产生中断，使程序进入中断服务程序内执行，编写步骤如下。

（1）设定时器T0为可中断：

```
EA=1
ET0=1
```

(2) 设定 T0 工作模式 1 :

```
TMOD=0x01
```

(3) 设定 T0 定时计数初始值 (计数 50000 次, 即 50ms) :

```
TH0=-(50000/256)
TL0=-(50000%256)
```

(4) 启动定时器 :

```
TR0=1
```

(5) 编写中断服务函数 (中断服务程序) :

```
T0_srv( ) interrupt 1 using 1 /* 中断服务函数 */
```

2. 程序

C51 语言编写的灯交互闪烁源程序 JT 21-2.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 char i=10;                     /* 声明变量 i 并赋予初始值 10 */
03
04 /* ----- 主函数 (主程序) ----- */
05 main( )                       /* 主函数 */
06 {
07     TMOD=0x01;                /* 设定 T0 为模式 1 */
08     TH0=-(50000/256);         /* 定时器初始值 50000 */
09     TL0=-(50000%256);
10     P1=0xf0;                 /* P1 端口输出 0xf0 (11110000B) */
11     EA=1;                     /* 总允许中断 */
12     ET0=1;                   /* 允许定时器 T0 中断 */
13     TR0=1;                   /* 启动定时器 */
14     While(1);                /* 无限循环 */
15 }
16 /* ----- 中断函数 (中断服务程序) ----- */
17 T0_srv( ) interrupt 1 using 1 /* 中断服务程序 */
18 {
19     TH0=-(50000/256);         /* 设定计数 50000 次 */
20     TL0=-(50000%256);
21     -- i;                    /* i 减 1 */
22     if ( i<=0 )              /* 判断 i 是否等于 0 */
23     {
24         P1=~P1;               /* 反相输出 */
25         i=10;                 /* i 赋值 10 */
26     }
}
```

21.2.3 代码详解

本程序由主函数和中断函数构成。其中主函数完成对定时器和定时器中断的设置及使程序无限循环；中断函数主要完成重新装入定时器初始值，并使定时器连续定时 10 次，即延时 0.5s 后 P1 作反相输出。

01：头文件。

02：声明 i 为字符变量，并赋予初始值 10。

05 ~ 15：主函数。其中第 07 行语句设置定时器 T0 为模式 1；第 08 行和第 09 行语句装入定时器的初始值 50000，即延时 50000 μ s；第 10 行语句为 P1 输出；第 11 行和第 12 行语句设置中断，EA=1 设置总允许中断，ET0=1 设置允许定时器 T0 中断；第 13 行语句启动定时器开始工作；第 14 行语句为无限循环。

17：为中断函数，“interrupt 1”表示使用定时器 T0, using 1 表示使用 1 号工作寄存器组。

18：中断函数开始。

19、20：重装定时器的初始值。

21：变量 i 值减去 1，例如，开始 i 值为 10，执行一次该语句后 i 值为 9。

22：if 语句，成立的条件是 i 等于 0，也就是说，只有 i 等于 0 时才能执行序号 24、25 语句。

23：if 内容开始。

24：P1 端口反相输出。比如，原来输出为 0xf0，使右侧 4 个灯亮，经反相输出为 0x0f，使左侧 4 个灯亮。中间间隔时间为变量 i 由 10 减为 0，表示定时器连续定时 10 次，即延时 0.5s 后 P1 反相输出。

25：i 重新赋值 10。

26：if 内容结束。

27：中断函数体结束。

21.2.4 经验总结

定时器在使用前需要进行设置，设置的要求与前边使用汇编语言编程基本相同，例如，设定工作模式、装入定时计数初始值、启动定时器等。

定时器的定时时间到后会发出两种信号，一是溢出标志 TF=1；二是产生中断。上节是通过检测溢出标志 TF 是否为 1，来获得定时器输出信号，本节是利用定时器的中断功能。

由于在 C51 语言中中断函数是一个特殊函数，只要使用 interrupt 和 using 声明是中断服务函数，就不必像使用汇编语言那样，考虑中断入口、现场保护和现场恢复等处理，这些问题都由系统自动解决，显然比使用汇编语言编写中断程序简单。

中断服务函数是系统调用的，程序中的任何函数都不能调用中断服务函数。

21.3 交通信号灯控制

功能说明：利用单片机 P1 端口控制 6 个交通信号灯，并使用定时器 T0 作定时，实现交通信号灯控制。

21.3.1 硬件设计

单片机的 I/O 端口直接控制交通信号灯，其电路原理如图 21.3 所示。

4 个路口应该安装 12 个交通信号灯，图中只画出 6 个来说明控制原理和方法，其路口信号灯示意图如图 21.4 所示。

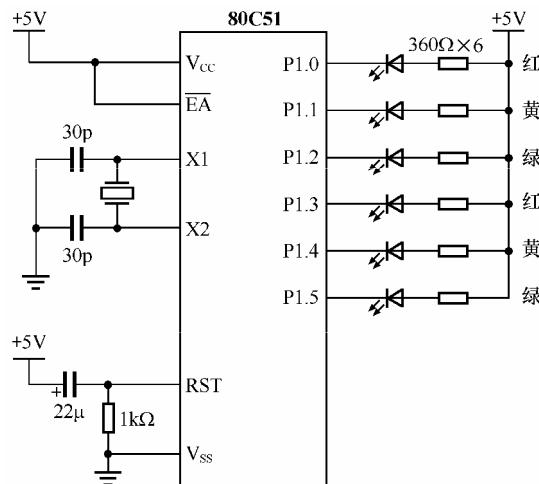


图 21.3 交通信号灯控制电路

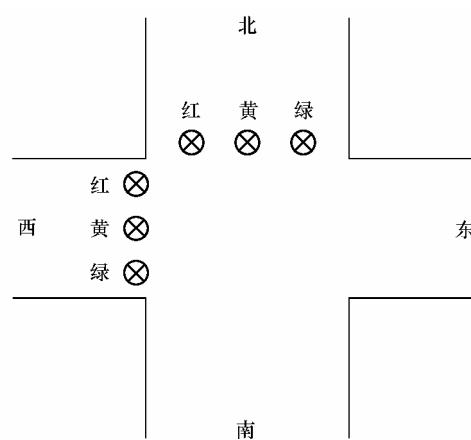


图 21.4 4 个路口信号灯示意图

在马路的十字路口东西南北各设置红、黄、绿 3 种信号灯。由 P1.0 ~ P1.2 控制南北方向，其中 P1.0 为红灯，P1.1 为黄灯，P1.2 为绿灯。P1.3 ~ P1.5 控制东西方向，其中 P1.3 为红灯，P1.4 为黄灯，P1.5 为绿灯。

21.3.2 程序设计

1. 工作原理

该程序使用定时器 T0 作定时，由 I/O 端口直接控制交通信号灯。

控制的过程为：当南北方向通车时绿灯亮，而东西方向红灯亮；当通车的时间到后，南北方向的绿灯熄灭，而黄灯亮，黄灯亮后由南北方向切换到东西方向。

此时南北方向黄灯熄灭，而红灯亮，东西方向红灯熄灭，而绿灯亮，东西方向开始通车。通车的时间到后，东西方向的绿灯熄灭，而黄灯亮，黄灯亮后由东西方向切换到南北方向，东西方向黄灯熄灭，红灯亮；而南北方向红灯熄灭，绿灯亮，如此不断循环。

2. 程序

C51 语言编写的交通信号灯控制源程序 JT 21-3.c 代码如下：

```

01 #include <AT89X51.H>          /* 头文件 */
02 char count;                   /* 声明 count 为字符变量 */
03 bit flag;                    /* 声明 flag 为位变量 */

```

```

04 char sum =0;                                /* 声明 sum 为字符变量并赋初始值 0 */
05 xhd( );                                     /* 信号灯函数 xhd( ) */
06 /* ----- 主函数 (主程序) -----*/
07 main( )                                      /* 主函数 */
08 {
09     TMOD=0x01;                                /* 设定 T0 为模式 1 */
10    TH0=-(50000/256);                          /* 设定时器初始值 50000 */
11    TL0=-(50000%256);
12    EA=1;                                       /* 总允许中断 */
13    ET0=1;                                       /* 允许定时器 T0 中断 */
14    TR0=1;                                       /* 启动定时器工作 */
15
16    P1=0xff;                                    /* P1 初始值，关闭所有信号灯 */
17    xhd( );                                     /* 调用信号灯函数 */
18 }
19 /* ----- 中断函数 (中断服务程序) ----- */
20 T0_srv( ) interrupt 1 using 1 /* 中断服务程序 */
21 {
22    TH0=-(50000/256);                          /* 重装定时器初始值 */
23    TL0=-(50000%256);
24    count++;                                    /* count 加 1 */
25    if(count==20)                               /* 判断 count 是否等于 20 */
26    {
27        count=0;                                /* count 清 0，以便重新计数 */
28        flag=1;                                 /* 标志位为 1 */
29    }
30 }
31 /* ----- 信号灯函数 (子程序) ----- */
32 xhd( )                                      /* 信号灯函数 */
33 {
34     while(1)                                  /* 无限循环 */
35     {
36 /* ----- */
37     P1_3=0; P1_2=0;                            /* 东西红灯亮，南北绿灯亮 */
38     while( sum<15)                           /* 循环 15 次，延时 15s */
39     {
40         while( ! flag);                      /* 等待 1s */
41         flag=0;                             /* 标志位清 0 */
42         sum++;                            /* sum 加 1 */
43     }
44     sum=0;                                     /* sum 清 0，以便重新计数 */
45 /* ----- */
46     P1_2=1; P1_1=0;                            /* 南北绿灯熄灭，黄灯亮 */

```

```

47     while( sum<3)          /* 循环 3 次，延时 3s */
48     {
49         while(! flag);
50         flag=0;
51         sum++;
52     }
53     sum=0;
54 /*-----*/
55     P1_1=1; P1_0=0;          /* 南北黄灯熄灭，红灯亮 */
56     P1_3=1; P1_5=0;          /* 东西红灯熄灭，绿灯亮 */
57     while( sum<10)          /* 延时 10s */
58     {
59         while(! flag);
60         flag=0;
61         sum++;
62     }
63     sum=0;
64 /*-----*/
65     P1_5=1; P1_4=0;          /* 东西绿灯熄灭，黄灯亮 */
66     while( sum<3)          /* 延时 3s */
67     {
68         while(! flag);
69         flag=0;
70         sum++;
71     }
72     sum=0;
73 /*----- */
74     P1_4=1; P1_0=1;          /* 东西黄灯熄灭，南北红灯熄灭 */
75 }
76 }

```

21.3.3 代码详解

本程序由主函数、中断函数和信号灯函数所构成。

01 ~ 05：头文件和变量、函数声明。

07 ~ 18：为主函数，其实现内容如下。

- 设置定时器 T0 的工作模式、装入计数初始值、允许中断，定时中断的时间为 50ms。
- 初始时，关闭所有信号灯。
- 调用信号灯函数。

20 ~ 30：为中断函数，其实现内容如下。

- 重装定时器初始值。
- 产生 1s 标准时间，并用 flag 做标志位。

32 ~ 76：为信号灯函数。使信号灯按照规定的次序、时间来点亮和熄灭，不断循环。

21.3.4 经验总结

交通信号灯的控制有严格时序要求，程序必须保证信号灯按照规定的次序、时间来点亮和熄灭，不断循环。

信号灯控制中使用的标准时间是由定时器产生的。定时器定时的时间设置为 50ms，再经软件计数方法使定时器重复定时 20 次，计为 1s，并用 flag 做标志位，即 flag 位由 0 变 1，为 1s 时间。在信号灯函数中，是通过检测 flag 位的变化次数，来获得延时的时间。本节程序是单片机基本输出功能和定时、中断功能的综合应用。

21.4 改进的交通信号灯控制

功能说明：在上例基础上，增加一个手动控制开关，接在 INT0 输入端，当有救护车、消防车等特殊车辆通过时，可以使两个方向均亮红灯，救护车和消防车通过后，恢复原来状态。增加了对出现特殊情况的处理能力。

21.4.1 硬件设计

电路设计如图 21.5 所示。

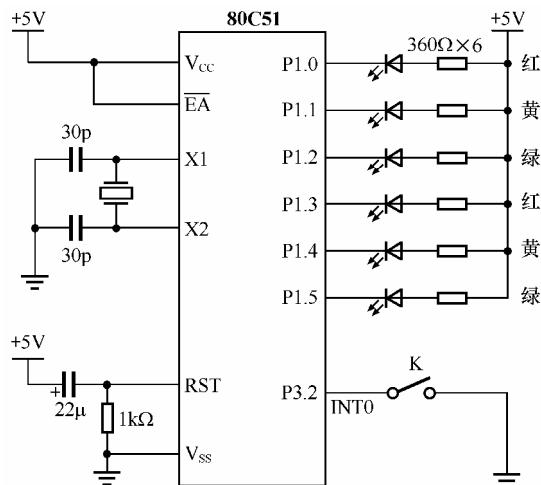


图 21.5 交通信号灯控制电路

本节电路是在上节电路图基础上，增加一个手动控制开关，接在 INT0 (P3.2) 输入端，其他部分与上节同。

21.4.2 程序设计

1. 工作原理

本节程序在上节程序基础上增加了外部中断 INT0 服务程序，当外部中断发生时，使两个方向均亮红灯，外部中断停止后，恢复原来状态。

外部中断INT0的设置。

(1) 开放外部中断INT0。

EX0=1

(2) 设置INT0为高优先中断。

PX0=1

2. 程序

C51语言编写的改进的交通信号灯控制源程序 JT 21-4.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 char count;                   /* 声明 count 为字符变量 */
03 bit flag;                    /* 声明 flag 为位变量 */
04 char sum =0;                 /* 声明 sum 为字符变量并赋初始值 0 */
05 xhd( );                     /* 信号灯函数 xhd( ) */

06 /* ----- 主函数 ----- */
07 main( )                      /* 主函数 */
08 {
09     TMOD=0x01;                /* 设定 T0 为模式 1 */
10     TH0=-(50000/256);         /* 设定时器初始值 50000 */
11     TL0=-(50000%256);
12     EA=1;                     /* 总允许中断 */
13     ET0=1;                   /* 允许定时器 T0 中断 */
14     TR0=1;                   /* 启动定时器工作 */

15
16     EX0=1;                   /* 允许外部中断 0 中断 */
17     PX0=1;                   /* 外部中断 0 为高优先级 */

18
19     P1=0xff;                 /* P1 初始值，关闭所有信号灯 */
20     xhd( );                  /* 调用信号灯函数 */

21 }
22 /* ----- 中断函数 ----- */
23 T0_srv( ) interrupt 1 using 1 /* 中断服务程序 */
24 {
25     TH0=-(50000/256);        /* 重装定时器初始值 */
26     TL0=-(50000%256);
27     count++;                 /* count 加 1 */
28     if(count==20)            /* 判断 count 是否等于 20 */
29     {
30         count=0;              /* count 清 0，以便重新计数 */
31         flag=1;               /* 标志位为 1 */
32     }
33 }
```

```
34 /* ----- 信号灯函数 ----- */
35 xhd( )                      /* 信号灯函数 */
36 {
37     while(1)                  /* 无限循环 */
38     {
39 /* -----*/
40     P1_3=0; P1_2=0;          /* 东西红灯亮，南北绿灯亮 */
41     while( sum<15)           /* 循环 15 次，延时 15s */
42     {
43         while( ! flag);      /* 等待 1s */
44         flag=0;              /* 标志位清 0 */
45         sum++;               /* sum 加 1 */
46     }
47     sum=0;                   /* sum 清 0，以便重新计数 */
48 /* -----*/
49 P1_2=1; P1_1=0;              /* 南北绿灯熄灭，黄灯亮 */
50 while( sum<3)                /* 循环 3 次，延时 3s */
51 {
52     while( ! flag);
53     flag=0;
54     sum++;
55 }
56 sum=0;
57 /* -----*/
58 P1_1=1; P1_0=0;              /* 南北黄灯熄灭，红灯亮 */
59 P1_3=1; P1_5=0;              /* 东西红灯熄灭，绿灯亮 */
60 while( sum<10)                /* 延时 10s */
61 {
62     while( ! flag);
63     flag=0;
64     sum++;
65 }
66 sum=0;
67 /* -----*/
68 P1_5=1; P1_4=0;              /* 东西绿灯熄灭，黄灯亮 */
69 while( sum<3)                /* 延时 3s */
70 {
71     while( ! flag);
72     flag=0;
73     sum++;
74 }
75 sum=0;
76 /* ----- */
```

```

77     P1_4=1; P1_0=1;           /* 东西黄灯熄灭，南北红灯熄灭 */
78   }
79 }
80 /* ----- 外部中断函数 ----- */
81 int0_srv( ) interrupt 0 using 2
82 {
83   char a;
84   unsigned int i;
85   for (i=0;i<10000;i++);
86   if ( INT0==0)
87   {
88     a=P1;                      /* 保留 P1 状态 */
89     P1=0xff;
90     P1=0xf6;                  /* 使两边亮起红灯 */
91     while (INT0==0);          /* 等待开关断开 */
92     P1=a;                      /* 恢复 P1 状态 */
93   }
94 }
```

21.4.3 代码详解

本程序在上节程序基础上，增加了外部中断函数，其他部分与上节同。

01~05：头文件和变量、函数声明。

07~21：主函数。主函数在上节基础上增加了对外部中断 INT0 的设置，即设置允许外部 INT0 中断，并设置 INT0 为高优先级。

23~33：定时中断函数。

35~79：为信号灯函数。使信号灯按照规定的次序、时间来点亮和熄灭，不断循环。

81~94：外部 INT0 中断函数。该函数主要解决两个问题。

- 先保留 P1 端口状态，待外部中断停止时，恢复原来状态。
- 在外部中断发生后，使两边亮起红灯。

21.4.4 经验总结

本节程序是在上节程序的基础上，增加了外部中断函数。在硬件方面只增加一个开关，但在交通信号灯控制功能却得到了进一步加强，提高了系统对出现特殊情况的处理能力。

程序中的外部中断函数和定时器中断函数都是特殊函数，在 C 语言编程中不需要考虑中断入口等设置问题，比用汇编语言编程简单。

在本程序中设置了外部中断 INT0 为高优先级中断，高优先级中断可以中断低优先级中断，所以当手动控制开关接通时，所产生的外部中断信号终止了其他正在运行的程序，转而去执行外部中断服务程序，停止普通车辆通行，给特殊车辆让路。外部中断服务程序结束后，信号灯又恢复原来正常运行状态。



第 22 章 通信测试

本章先介绍简单的发送、接收程序，再介绍由发送和接收两部分组合在一起的发送与接收测试程序，从而讲解单片机串行口通信原理及编程方法。

22.1 发送一个字符

功能说明：由单片机向计算机不断地发送一个字符“ A ”，在计算机超级终端窗口上显示出来。计算机中也可以安装串口调试工具如 SSCOM 等（到网站上免费下载），既能接收也能发送，使用很方便。

22.1.1 硬件设计

单片机与计算机通信连接的示意图如图 22.1 所示。

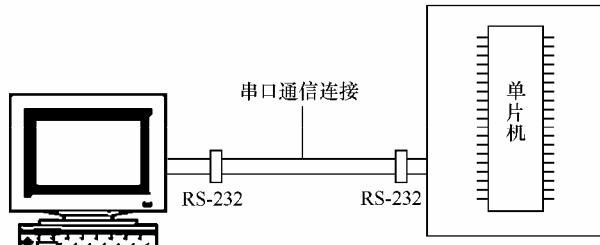


图 22.1 单片机与计算机通信连接示意图

单片机与计算机通信时，要用随机带的串口连线将单片机实验板与计算机连接。计算机中使用 COM1 还是使用 COM2，可在串行口调试工具软件上设置；“数据传输率”一项的设置要与程序中设定的数据传输率相一致。

22.1.2 程序设计

1. 工作原理

（1）设定串行口工作方式。

SCON 是串行口控制寄存器，主要功能是设定串行口的工作方式、接收和发送控制以及设置状态标志，其格式如下：

98H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
-----	-----	-----	-----	-----	-----	-----	----	----

- 设定串行口工作方式 1。

```
SM0=0;  
SM1=1;  
SM2=0
```

- 设定由软件来驱动串行口的发射与接收。

```
REN=1;
```

由以上得知：

```
SCON=01010000B=0x50
```

(2) 数据传输率的设定。

数据传输率是指发送或接收的数据速率，串行口工作于方式 1 时由定时器 T1 的溢出率控制。当设置数据传输率为 9600 时：

```
TMOD=0x20;  
TH1 = 0xfd;
```

(3) 激活定时器 T1。

```
TR1=1;
```

(4) 发送一字符“ A ”。

```
SBUF='A'
```

2. 程序

C51 语言编写的发送一个字符源程序 TX22-1.c 代码如下：

```
01 #include <AT89X51.H>           /* 头文件 */  
02 main( )                      /* 主函数 */  
03 {  
04     unsigned int i;            /* 声明无符号整数变量 i */  
05     SCON=0x50;                /* 设定串行口工作方式 1 */  
06     TMOD=0x20;                /* 定时器 1 工作于 8 位自动重载模式 */  
07     TH1 = 0xfd;                /* 设定数据传输率为 9600 */  
08     TR1=1;                    /* 启动定时器 1 */  
09     loop:  
10         for (i=0;i<50000;i++);    /* 延时 */  
11         SBUF='A';                /* 传送字符 */  
12         goto loop;              /* 跳转到 loop 处 */
```

13 }

22.1.3 代码详解

- 01：头文件。
- 02：主函数。
- 03：主函数开始。
- 04：声明 i 为无符号整数变量。
- 05：设定串行口工作方式 1。串行口有 4 种工作方式：方式 0~ 方式 3。通过对串行口控制寄存器 SCON 的设置来进行选择，不同的工作方式适应不同的要求。方式 0 用于扩展 I/O 口；方式 1 适合于点对点的异步通信；方式 2、方式 3 适用多机通信。
- 06：设定定时器 1 工作于方式 2，作数据传输率发生器，该工作方式为 8 位自动重载模式。
- 07：设定数据传输率为 9600，收发双方必须设置相同的数据传输率。
- 08：启动定时器 1 开始工作。
- 09：loop 为程序标号，也是本程序的循环进入点。
- 10：延时大约 0.5s。
- 11：发送字符。SBUF 为串行口数据缓冲寄存器，要发送的数据首先写入 SBUF 中，然后才能发送。
- 12：转移到 loop 处不断循环，屏幕不断显示字符“ A ”。
- 13：程序结束。

22.1.4 经验总结

这是由单片机向计算机不断发送一个字符的简单程序。设计该程序时，先对串行口工作模式及数据传输率进行设置，再将要发送的字符送入缓冲寄存器 SBUF 中，该字符就会被发送出去。

22.2 发送一个字符串

功能说明：每按一次 K1 键，单片机向计算机发送一个字符串“Hello! ”，并在计算机的接收软件窗口上显示出来。本节硬件设计与上节基本相同，但在单片机的 P3.2 端口要接一个按钮开关 K1，作为字符串发送按钮。本节硬件设计与 22.1 节相同。

22.2.1 程序设计

1. 工作原理

- (1) 设定串行口工作方式 1。

SCON=0x50

(2) 设定数据传输率为 9600。

```
TMOD=0x20;
TH1 = 0xfd;
```

(3) 激活定时器 T1。

```
TR1=1;
```

(4) 发送字符串“Hello!”。

```
SBUF = str[i];
```

2. 程序

C51 语言编写的发送一个字符串源程序 TX22-2.c 代码如下：

```

01 #include<AT89X51.H>           /* 头文件 */
02 char code str[] = "Hello! \0";    /* 声明字符数组 str[] */
03 void send_str();                 /* 发送字符串函数 */
04 sbit K1 = P3^2;                  /* 定义按键 */
05 unsigned char temp;              /* 声明无符号字符变量 temp */
06 main()                          /* 主函数 */
07 {
08     SCON = 0x50;                 /* 设定串行口工作方式 1 */
09     TMOD = 0x20;                 /* 定时器 1，自动重载，产生数据传输率 */
10     TH1 = 0xfd;                  /* 数据传输率为 9600 */
11     TR1 = 1;                     /* 启动定时器 1 */
12
13     while(1)                    /* 无限循环 */
14 {
15     unsigned int j;             /* 声明无符号整数变量 j */
16     if (K1==0)                  /* 如果按 K1 键，则执行{ } */
17     {
18         for(j=0;j<20000;j++);   /* 延时一段时间，消除抖动 */
19         while(!K1);            /* 等待按键放开 */
20         send_str();            /* 调用发送字符串函数 */
21     }
22 }
23 }
24
25 void send_str()                /* 发送字符串函数 */
26 {
27     unsigned char i = 0;          /* 声明无符号字符变量 i */
28     while(str[i] != '\0')        /* 遇到 '\0'，则停止发送 */
29     {
30         SBUF = str[i];           /* 发送字符 */

```

```

31     while(!TI);           /* 等待数据传送 */
32     TI = 0;               /* 清除数据传送标志 */
33     i++;                 /* 下一个字符 */
34   }
35 }
36 }
```

22.2.2 代码详解

- 01 : 头文件。
- 02 : 声明字符数组 str[] , 字符串中的 ‘ \0 ’ 为字符串结束标志符。
- 03 : 声明发送字符串函数。
- 04 : 定义按键 K1 由 P3.2 控制。
- 05 : 声明 temp 无符号字符变量。
- 06 : 主函数。
- 07 : 主函数开始。
- 08 : 设定串行口工作方式 1。
- 09 : 设置定时器 1 , 工作于 8 位自动重载模式 , 用于产生数据传输率。
- 10 : 设定数据传输率为 9600。
- 11 : 启动定时器 1 开始工作。
- 13 : 无限循环语句。
- 14 : while 循环开始。
- 15 : 声明无符号整数变量 j。
- 16 : 判断 K1 键是否按下 , 如果按下 , 程序执行下边大括号{ }内的语句。
- 17 : if 语句开始。
- 18 : 延时一段时间 , 消除按键的抖动 。
- 19 : 等待按键 K1 释放。
- 20 : 调用发送字符串函数。
- 21 : if 语句结束。
- 22 : while 循环结束。
- 23 : 主函数结束。
- 25 : 发送字符串函数。
- 26 : 发送字符串函数开始。
- 27 : 声明无符号字符变量 i。
- 28 : 遇到结束标志符号 ‘ \0 ’ , 说明字符串已经发送完成 , 则停止发送。
- 29 : while 循环语句 , 开始执行发送字符串。
- 30 : 发送字符 “ Hello! ”。
- 31 : 等待数据传送。
- 32 : TI 为发送中断标志位 , 当第 8 位结束时 , 硬件会将其设置为 1 ; 之后 , 必须由软件清除为 0。
- 34 : i 加 1 , 发送下一个字符。

35：while 循环发送字符结束。

36：发送字符串函数结束。

22.2.3 经验总结

本节是简单的发送字符串程序。设计该程序时，先设置串行口工作模式及数据传输率，再将要发送的字符串送入缓冲寄存器 SBUF 中，该字符串就会被发送出去。

上述过程基本与上节相同，区别是 C 语言中没有字符串变量，需要用字符数组来处理字符串，例如，发送语句应写为 `SBUF = str[i]`。

为了测定字符串的实际长度和使用系统提供的各种字符串函数，C 语言规定一个字符串结束标志位 ‘\0’。在一个字符串数组中，一旦遇到字符 ‘\0’ 就表示字符串结束，如，执行语句 `while(str[i] != '\0')` 后，‘\0’ 后的字符忽略不计。

22.3 接收指令

功能说明：由计算机向单片机发送“00”指令，单片机接收到指令后，P1 端口 8 个 LED 全部亮起。之后，计算机发送 1~9 数字，单片机 P1 端口 8 个 LED 将模拟显示二进制数，如表 22.1 所示。本节硬件设计与 22.1 节相同。

表 22.1 P1 口显示二进制数

计算机发送	1	2	3	4	5	6	7	8	9
P1 显示	00000001	00000010	00000011	00000100	00000101	00000110	00000111	00001000	00001001

22.3.1 程序设计

1. 工作原理

(1) 设定串行口工作方式 1。

```
SCON=0x50
```

(2) 设定数据传输率为 9600。

```
TMOD=0x20;  
TH1 = 0xfd;
```

(3) 激活定时器 T1。

```
TR1=1;
```

(4) 接收数据。

```
temp = SBUF;
```

2. 程序

C51 语言编写的接收指令源程序 TX22-3.c 代码如下：

```

01 #include <AT89X51.H>           /* 头文件 */
02 unsigned char temp;           /* 声明无符号字符变量 i */
03 main()                      /* 主函数 */
04 {
05     SCON = 0x50;             /* 设定串行口工作方式 1 */
06     TMOD = 0x20;             /* 定时器 1，自动重载，产生数据传输率 */
07     TH1 = 0xfd;              /* 数据传输率为 9600 */
08     TR1 = 1;                 /* 启动定时器 1 */
09
10    while(1)                  /* 无限循环 */
11    {
12        while(!RI);          /* 等待数据接收 */
13        RI = 0;               /* 清除数据传送标志 */
14        temp = SBUF;           /* 暂存接收到的数据 */
15        P1 = temp&0x0f;         /* 数据传送到 P0 口 */
16    }
17 }
```

22.3.2 代码详解

01、02：头文件和变量声明。

03：主函数。

05~08：对串行通信口初始设置。

10~16：接收数据。其中，第 14 行语句将串行口缓存寄存器 SBUF 中所接收到的数据存入 temp 中，第 15 行语句将接收到的数据送到 P1 输出。

17：程序结束。

22.3.3 经验总结

本节是单片机接收计算机发送信息的简单程序。设计该程序时，先设置串行口工作模式及数据传输率，再将接收到的信息从串行口接收数据缓冲寄存器 SBUF 中读出，即为接收。单片机按照接收到的指令去控制 P1 端口输出。

22.4 发送接收测试程序

功能说明：将单片机实验板与计算机连接，参考 22.1 节和 22.2 节说明。首先，由单片机向计算机发送信息（按 K1 键），如果发送工作正常，计算机调试软件窗口将显示：“PC 已接收到信息，OK！等待主机发送 00 指令！”，如图 22.2 所示。接着，由计算机向单片机发送

“00”指令，如果单片机接收正常，P1端口接的8只LED全部亮起，同时还将该指令返回到计算机屏幕上。



图 22.2 串口通信调试窗口

22.4.1 程序设计

1. 工作原理

本节程序是前边介绍的发送与接收程序的综合应用。程序由主函数、发送函数 send_str() 和回传函数 send_char() 所组成。

其中在主函数中对串口进行初始化设置，并不断对按键 K1 进行扫描，一旦检测到有键按下时，程序将调用发送函数进行发送，同时程序也不断检测是否收到主机发来的数据，如果收到数据，一方面送入 P1 端口输出；同时还将该数据返回到主机显示。

2. 程序

C51 语言编写的发送接收测试程序源程序 TX22-4.c 代码如下：

```

01 #include<AT89X51.H>           /* 头文件 */
02 char code str[] = "PC 已接收到信息, OK ! 等待主机发送 00 指令 ! \0";
03 void send_str();                /* 发送函数 */
04 void send_char();               /* 回传函数 */
05 sbit K1 = P3^2;                 /* 定义按键 */
06 unsigned char temp;             /* 声明无符号字符变量 temp */
07 main()                         /* 主函数 */
08 {
09     SCON = 0x50;                  /* 设定串行口工作方式 1 */
10     TMOD = 0x20;                  /* 定时器 1，自动重载，产生数据传输率 */
11     TH1 = 0xFD;                  /* 数据传输率为 9600 */

```

```

12   TR1 = 1;                      /* 启动定时器 1 */
13
14   while(1)                      /* 无限循环 */
15   {
16       unsigned int j;            /* 声明无符号整数变量 j */
17       if (K1==0)                /* 如果按 K1 键，则执行{ } */
18   {
19       for(j=0;j<20000;j++);    /* 延时消除抖动 */
20       while(!K1);             /* 等待按键放开 */
21       send_str();              /* 调用发送字符串函数 */
22   }
23   if(RI)                         /* 是否有数据到来 */
24   {
25       RI = 0;                  /* 清除数据传送标志 */
26       temp = SBUF;             /* 将接收到的数据暂存 temp 中 */
27       P1 = temp&0x0f;           /* 数据传送到 P1 口输出 */
28       send_char();              /* 调用回传函数 */
29   }
30 }
31 }
32
33 void send_char()                /* 回传函数 */
34 {
35     SBUF = temp;                /* 回传发送接收到的数据 */
36     while(!TI);                /* 等待数据传送 */
37     TI = 0;                    /* 清除数据传送标志 */
38 }
39
40 void send_str()                /* 发送字符串函数 */
41 {
42     unsigned char i = 0;         /* 声明无符号字符变量 i，初始值 i=0 */
43     while(str[i] != '\0')        /* 遇到 '\0' 则停止发送 */
44     {
45         SBUF = str[i];           /* 发送字符串 */
46         while(!TI);             /* 等待数据传送 */
47         TI = 0;                  /* 清除数据传送标志 */
48         i++;                     /* i 加 1，发送下一个字符 */
49     }
50 }
```

22.4.2 代码详解

01~06：头文件和函数及变量声明。

07~32：主函数。

09 ~ 12 : 串口通信初始化设置，包括选择工作方式、产生数据传输率和启动定时器。

14 ~ 30 : while 无限循环部分。

- 扫描 K1 按键，一旦检测到有按键时，程序将调用发送函数发送数据。

- 检测是否收到主机发来的数据，如果收到数据，送入 P1 端口输出，同时调用回传函数将该数据返回到主机显示。

31 : 主函数结束。

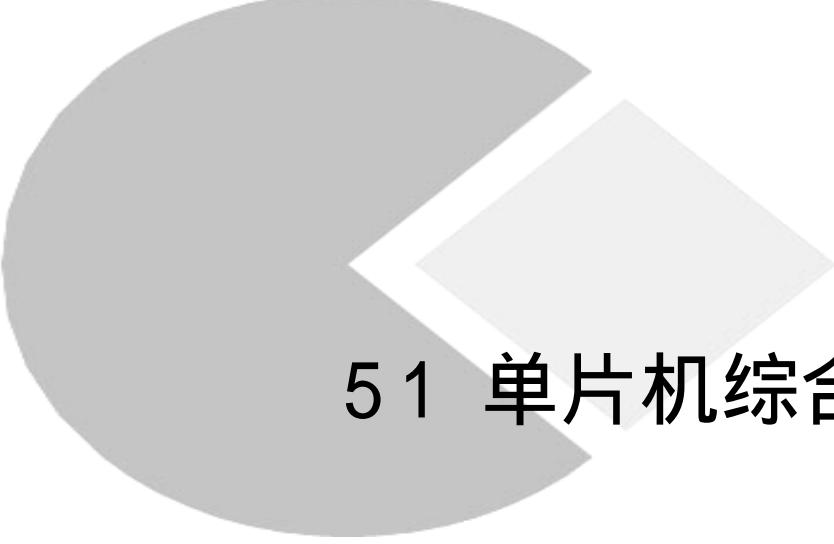
33 ~ 38 : 回传函数，功能也是发送，它是将接收到的数据再返回给主机。

40 ~ 50 : 发送字符串函数。

22.4.3 经验总结

本节程序是单片机发送与接收的简单测试程序。在单片机与计算机连接通信时，可使用该程序检测单片机发送和接收信息的工作是否正常。

发送程序与接收程序的语句大部分相同，两个程序的主要区别是：发送时需要先将数据写入数据缓冲寄存器 SBUF 中，例如，语句“SBUF = temp”；而接收时是从数据缓冲寄存器 SBUF 中将数据读出，例如，语句“temp = SBUF”。发送与接收语句是通信程序中两条重要语句，掌握这两条语句的特征，对编写与分析串行口通信程序非常有用。



5.1 单片机综合实例篇

本篇的时钟设计综合实例和液晶显示综合实例是单片机基本功能的实际综合应用，也是对前边学的知识总结和升华。



第 23 章 时钟设计综合实例

设计步骤：简单时钟 带闹铃时钟 带闹铃倒计时时钟。在设计中会感受到单片机产品的硬件结构比较简单，而产品的性能主要靠软件来实现。

23.1 简单时钟设计

简单单片机时钟设计是后边将要讲解的闹铃时钟和带闹铃倒计时时钟设计的基础。

23.1.1 学习单片机时钟设计目的

1. 实用性

利用 80C51 单片机结合数码管显示器设计一个电子小闹钟，由于用 LED 数码管显示数据，在夜晚或黑暗的场合里也可以使用，具有一定的实用性。

2. 对单片机基础知识的一次总复习

电子小闹钟的设计利用单片机的输入/输出功能、定时/计数功能和中断功能，因此，是对前边讲的单片机基本功能知识的一次复习、总结和提高。

3. 单片机时钟系统是学习单片机的实验板

电子小闹钟的结构简单，但却具备了进行单片机实验的基本构成，其中，所用到的按键开关、数码管、发光二极管和蜂鸣器是单片机系统中最为常用的输入输出设备。所以，单片机时钟系统也是学习单片机的实验板。

4. 掌握程序的模块化设计思想

由于设计单片机时钟时尽量减少硬件，所以丰富的功能只能由软件来完成，因此使程序语句比较多，看起来会比较复杂，但仔细研究就会发现大部分模块都是以前看到的学过的，这里只是根据需要把它们有机地组合在一起。所以，在学习设计过程中会给我们两点启发。

(1) 注意程序的模块化设计与分析。在程序设计时，一般将比较复杂的大程序分解若干个功能模块，然后再把各模块通过主程序有机地联系在一起。

分析程序是设计程序的逆过程，首先要弄清程序是由几个模块所组成，每个模块主要功能是什么，模块之间是怎样联系在一起的。先有一个粗线条、总轮廓，然后再逐步深入分析。

每个子程序就是一个模块，子程序段的特征是：开始行有程序标号，以便主程序调用；末尾行有子程序返回指令 RET 或 RETI。

(2) 每一个典型模块既可在这个程序使用，也可以在其他程序中使用。所以，掌握一些典型模块的功能、原理，建立一个模块库，一旦设计新的程序需要使用时就不必重新编写，可以减少重复劳动。

23.1.2 时钟结构与原理

1. 单片机时钟结构

单片机时钟的结构分硬件和软件两部分。其中，硬件部分比较简单，主要由单片机、LED 数码管显示器和按键开关组成，电路设计如图 23.1 所示。

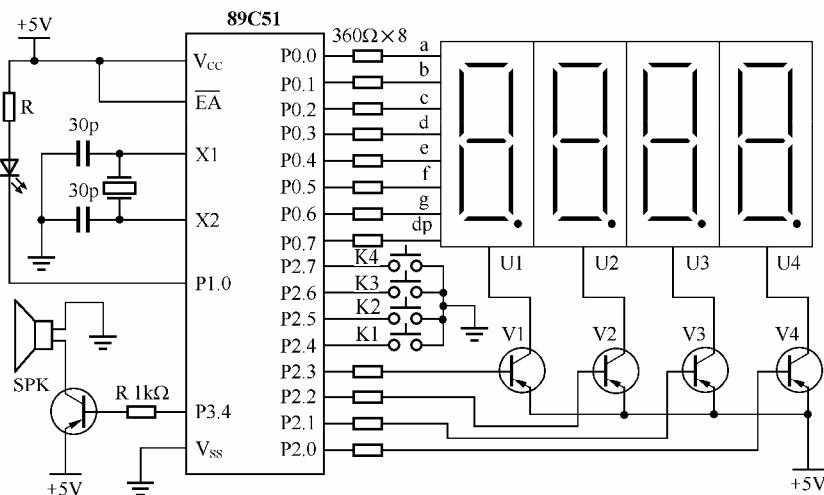


图 23.1 单片机时钟电路

单片机 P0 端口接有 4 位共阳极 LED 数码管显示器。

数码管的 8 个引脚依照 a、b、c、d、e、f、g、dp 顺序依次与 P0 端口的 8 个引脚 P0.1、P0.2、P0.3、P0.4、P0.5、P0.6、P0.7 相连，R 是限流电阻。

4 位 LED 数码管的共阳极引脚分别与 V1 ~ V4 三极管的集电极相连，三极管的基极通过限流电阻分别接在单片机 P2 端口的 P2.0 ~ P2.3 引脚上。

4 位数码管显示器分别由 4 只三极管控制，例如，P2.0 输出为低电平时，V4 三极管导通，与其相连的共阳极数码管显示器开始工作；P2.0 输出为高电平时，V4 三极管截止，与其相连的数码管显示器停止工作。

2. 使用方法

接通电源后，蜂鸣器连续两次发出响声，同时工作指示灯 LED 闪动，表示程序开始执行，数码管显示“0000”。

接着需要设置现在时间：K1 为设置现在时间功能键，按一下 K1 键，发光二极管 LED 停止闪动，表明进入了设置现在时间状态，此时，按 K2 键为小时调整键，按一次，数值增 1；K3 键为分钟调整键，按一次，数值增 1。设置完成后，要按一下 K4 键，LED 恢复闪动，表明设置完成，时钟进入了正常走时状态。

3. 基本原理

时钟一般是由走时、显示和调整时间 3 项基本功能组成，这些功能在单片机时钟里主要由软件设计体现出来。其中，走时部分利用单片机里的定时/计数器产生的中断。

例如，设置定时器 T0 工作在模式 0 状态下，设置每隔 5ms 中断一次，中断 200 次正好是 1s。中断服务程序里记载着中断的次数，中断 200 次为 1 秒，60 秒为 1 分，60 分为 1 小时，24 小时为 1 天。

时钟的显示是使用 4 位 LED 数码管，其软件设计原理是：由中断产生的秒、分、小时数据，经转换子程序转换成适应 LED 数码管显示的数据，并通过单片机的输出功能输入到数码管显示器，再通过显示器扫描程序，显示出时钟的走时时间。

调整时钟时间是利用了单片机的输入功能，把按键开关作为单片机的输入信号，通过检测被按下的开关，从而执行赋予该开关调整时间功能。

因此，在设计程序时把单片机时钟功能分解为走时、显示和调整时间 3 个主要部分，每一部分的功能通过编写相应的子程序来完成，然后再通过主程序调用子程序，使这 3 部分有机地连在一起，完成单片机的时钟设计。

23.1.3 走时功能的设计

单片机时钟实现走时功能使用了两个子程序：定时器子程序 INIT_TIMER 和中断服务子程序 TO_INT。其中，定时器子程序 INIT_TIMER 的作用是每隔 5ms 产生一次中断信号；中断服务子程序 TO_INT 的主要作用是记载中断的次数。

1. 定时器子程序 INIT_TIMER

定时器子程序 INIT_TIMER 的作用是每隔 5ms 产生一次中断信号，它是时钟标准时间的来源和保证。下面是定时器子程序 INIT_TIMER：

```
INIT_TIMER:           ; 定时器 T0 初始化
    MOV TMOD, #00000000B   ; 设置定时器 T0 工作模式为 0
    MOV TL0, #(8192-5000)MOD 32 ; 加载低字节计数初值
    MOV TH0, #(8192-5000)/32 ; 加载高字节计数初值
    MOV IE, #10000010B       ; 启用定时器 T0 中断
    SETB TR0                 ; 启动定时器 T0 开始计时
    RET                      ; 子程序返回
```

定时器 T0 设置为工作模式 0 状态。定时器每隔 5ms 中断一次，在晶振频率为 12MHz 时，此

5ms 的初值为 5000，但实际上程序还要作其他运算，使得时间偏长，因此此值需经实验作些调整。

2. 中断服务程序 TO_INT

中断服务程序 TO_INT 的作用是重置定时器 T0 初始值，记录中断次数，并输出秒、分和小时。其中，DEDA 存放 5ms 计数值，SEC 存放秒钟变量，MIN 存放分钟变量，HOUR 存放小时变量。中断服务程序 TO_INT 如下：

1	TO_INT:	; 中断服务程序
2	PUSH ACC	; 将 A 值压栈，现场保护
3	MOV TL0, #(8192-4150)MOD 32	; 低位重置 T0 初值
4	MOV TH0, #(8192-4150)/32	; 高位重置 T0 初值
5	INC DEDA	; 加 1
6	MOV A, DEDA	;
7	CJNE A, #200, TT1	; 是否 1s 到了
8	MOV DEDA, #0	; 计数值清 0
9	CPL WLED	; LED 灯亮灭变换
10	INC SEC	; 秒计数加 1
11	MOV A, SEC	;
12	CJNE A, #60, TT1	; 是否 1min 到了
13	INC MIN	; 分计数加 1
14	MOV SEC, #0	; 秒计数清 0
15	MOV A, MIN	;
16	CJNE A, #60, TT1	; 是否 1h 到了
17	INC HOUR	; 小时计数加 1
18	MOV MIN, #0	; 分计数清 0
19	MOV A, HOUR	;
20	CJNE A, #24, TT1	; 是否 24h 到了
21	MOV SEC, #0	; 秒钟变量清 0
22	MOV MIN, #0	; 分钟变量清 0
23	MOV HOUR, #0	; 小时变量清 0
24	TT1:	
25	POP ACC	; 出栈，现场恢复
26	RETI	; T0 中断服务程序返回

程序中秒、分和小时的输出语句在结构上相同，由递增指令 INC 记录数，并通过比较条件转移指令 CJNE 来实现对输出的判断。例如，比较累加器 A 中存入的秒数小于 60 时，秒数值继续递增。当增到等于 60 时，程序向下运行，使分计数加 1，同时秒计数清 0，即从头开始计秒数。

23.1.4 显示部分的设计

单片机时钟实现显示功能的子程序有 转换时、分子程序 CONV 和扫描显示子程序 DISP。先通过转换时、分子程序 CONV，把由中断服务程序中产生的分、小时数据转换成适应 LED 数码管显示的数据，即进行十进制计时处理，并存入数码管显示内存中（事先设置的存放显示器数据单元）。再经扫描显示子程序 DISP，便能在 LED 数码管上显示出时间。

1. 转换时分子程序 CONV

转换时分子程序 CONV 的作用是将中断服务程序中产生的分、小时数据，转换成适应 LED 数码管显示的数据。通过执行 DIV 指令进行十进制计时处理，并将处理后的数据分别存入数码管显示内存 BUF、BUF+1、BUF+2、BUF+3 内。其中 BUF、BUF+1 存放小时；BUF+2、BUF+3 存放分钟，与数码管显示器 U1、U2 和 U3、U4 相对应。下面是转换时分子程序 CONV。

```

1 CONV:
2     MOV      A, HOUR           ;开始转换小时数据
3     MOV      B, #10            ;16 进制换成 10 进制
4     DIV      AB              ; A ÷ B , 商存 A , 余数存 B
5     MOV      DPTR, #TABLE    ;查表转换
6     MOVC    A, @A+DPTR
7     MOV      BUF, A
8     MOV      A, B
9     MOVC    A, @A+DPTR
10    MOV     BUF+1, A
11    MOV      A, MIN           ;开始转换分钟数据
12    MOV      B, #10            ;16 进制换成 10 进制
13    DIV      AB              ; A ÷ B , 商存 A , 余数存 B
14    MOV      DPTR, #TABLE    ;查表转换
15    MOVC    A, @A+DPTR
16    MOV      BUF+2, A
17    MOV      A, B
18    MOVC    A, @A+DPTR
19    MOV      BUF+3, A
20    RET
21
22 TABLE:                      ;字型数据编码表
23     DB  0C0H,0F9H,0A4H,0B0H
24     DB  99H,92H,82H,0F8H
25     DB  80H,90H,88H,83H
26     DB  0C6H,0A1H,86H,8EH

```

转换时分子程序 CONV 由 3 部分组成，第 1 ~ 10 行转换小时数据；第 11 ~ 19 行转换分钟数据，这两部分语句结构基本相同；第 22 ~ 26 行是字型数据编码表。

2. 扫描显示子程序 DISP

显示器扫描子程序 DISP 的作用是动态显示送入 4 位 LED 数码管的时间数据。所谓动态显示是指一位一位轮流点亮 LED 数码管，每一位停留 4ms 左右，利用人的视觉暂留消除闪烁现象。下面是显示器扫描子程序。

```

1 DISP:
2     MOV      R0, #BUF          ;指向显示器显示缓冲区起始地址

```

```

3      MOV      R2, #4          ;循环执行 4 次
4      MOV      ACC, #11110111b   ;加载扫描信号初值 11110111B
5      S1:
6      PUSH     ACC
7      MOV      A, @R0          ;取出显示器数据
8      MOV      P0, A           ;由 P0 送出一位显示器数据
9      POP      ACC
10     MOV     P2, ACC          ;由 P2 送出扫描输出信号
11     MOV      R5, #2          ;延迟
12     ACALL   DELAY
13; 改变扫描码 EX:XXXX1011
14     RR      A               ;A 向右移动一位
15     INC     R0               ;显示器显示缓冲区地址加 1
16     DJNZ   R2, S1           ;循环判断是否继续执行
17     RET

```

扫描显示子程序 DISP 由部分组成。

(1) 加载扫描初始值部分。

第 1 行 ~ 4 行语句将显示器缓冲区起始地址 BUF，即存入十位上小时数的地址，送入寄存器 R0；将扫描信号初值 11110111B 送入累加器 ACC；扫描循环执次数，送入寄存器 R2。

(2) 输出 1 位字符。

第 5 行 ~ 12 行语句是输出 1 位字符的过程。

第 7 行语句从地址 BUF 里取出显示器数据。

第 8 行语句将取出的显示器数据送入 P0 输出。

 注意 此时 4 位数码管显示器都会收到此数据，但是此时输出的显示器数据是十位上的小时数，应该在左边第一位数码管显示，即在 U1 数码管显示，其他数码管应该关闭。

数码管的导通还是关闭是由扫描信号初值 11110111B 控制的，该扫描信号低 4 位数值控制着 4 位数码管是否导通，0 对应的数码管为导通，1 对应的数码管为不显示，此控制信号由第 10 行语句使 P2 端口送出，所以此时只有 U1 数码管导通，显示十位上的小时数。第 11 行、12 行语句使显示的时间延时 4ms，以便使显示的字符稳定。

(3) 进行下一位扫描。

第 14 ~ 16 行语句是进行下一位扫描的转变过程。

第 14 行语句将显示器显示缓冲区地址加 1，即从 BUF+1 单元地址处取数据，此数据是个位小时数，左数第 2 位数码管导通，所以第 13 行语句改变了扫描码，将 0 向右移一位，保证了左数第 2 位 U2 数码管导通显示字符，其他数码管都不显示。

就这样循环执行 4 次完成显示器扫描一遍的任务，实际上数码管是在轮流显示，每一时刻只是一个数码管亮。

23.1.5 调整时间部分的设计

SET_TIME 是调整时钟时间子程序。当单片机时钟每次重新启用时，都需要重新设置目前时钟的时间，其程序如下。

```

1 SET_TIME: ;调整目前时间子程序
2     CLR     TR0 ;定时器暂停
3     MOV     SEC, #0 ;秒钟变量清 0
4 L0: ;判断是否按 K2 键
5     ACALL   SCAN ;调用扫描显示器子程序
6 ;判断是否按 K2 键
7     JB      K2, L1 ;未按下 K2 键，则继续扫描
8     JNB    K2, $ ;按下，则等待放开
9 ;小时调整
10    INC    HOUR ;小时计数加 1
11    MOV    A, HOUR
12    CJNE   A, #24, L11 ;是否 24h 到了
13    MOV    HOUR, #0 ;小时变量清 0
14 L11: ;
15    ACALL   CONV ;转换显示数据
16    ACALL   SCAN ;扫描显示器
17    JMP     L0 ;使程序跳转到 L0 处执行
18 ;判断是否按 K3 键
19 L1: ;判断是否按 K3 键
20    JB      K3, L2 ;未按下 K3 键，则继续扫描
21    JNB    K3, $ ;按下，则等待放开
22 ; 分钟调整
23    INC    MIN ;分钟计数加 1
24    MOV    A, MIN
25    CJNE   A, #60, L21 ;是否 60 min 到了
26    MOV    MIN, #0 ;分钟变量清 0
27 L21: ;
28    ACALL   CONV ;转换显示数据
29    ACALL   SCAN ;扫描显示器
30    JMP     L0 ;使程序跳转到 L0 处执行
31 ;判断是否按 K4 键
32 L2: ;判断是否按 K4 键
33    JB      K4, L0 ;未按下 K4 键，则继续扫描
34    JNB    K4, $ ;按下，则等待放开来
35    ACALL   BZ ;蜂鸣器鸣响一声
36 ;设置完成
37    ACALL   LED_BL ;LED 闪动
38    SETB    TR0 ;启动定时器
39    RET ;子程序返回

```

当按下设置时间功能键 K1 后，程序便进入调整时间子程序 SET_TIME 中运行，该子程序可分 4 部分。

(1) 完成设置时间前的准备工作。第 1~5 行语句完成设置时间前的准备工作，包括暂

停定时器工作；将秒钟清 0 和调用扫描显示器子程序。

(2) 小时调整。第 7~17 行语句为小时调整部分。

(3) 分钟调整。第 19~30 行语句为分钟调整部分。

(4) 设置完成处理。第 32~39 行语句为设置完成处理部分。

第二、三、四部分语句结构基本相同，首先通过第 7、8、20、21、33、34 行语句判断是否有按键，若无按键则继续扫描下一键，若有按键则执行相应按键的处理程序。

其中，K2 键是小时调整键，K3 键是分钟调整键，K4 键为设置完成键。当设置时间完成后，按 K4 键通过第 38 行语句启动定时器开始工作，第 39 行语句使子程序返回。

23.1.6 喇叭和指示灯等子程序

除了上面主要子程序外，还有喇叭和指示灯等其他子程序。

1. 计时单元清 0 子程序

```

1 INIT:                                ; 计时单元清 0
2     MOV      DEDA, #0                 ; 5ms 计数值清 0
3     MOV      SEC, #0                 ; 秒钟变量清 0
4     MOV      MIN, #0                 ; 分钟变量清 0
5     MOV      HOUR, #0                ; 小时变量清 0
6     RET                               ; 子程序返回

```

2. 蜂鸣器鸣响一声子程序

```

1 BZ:
2     MOV      R6, #250    ;
3 B1:    ACALL    DEX                  ; 调短暂延时子程序
4     CPL      SPK                  ; 位反向输出
5     DJNZ    R6, B1                ; 控制响声长度
6     MOV      R5, #50    ;
7     ACALL    DELAY               ; 调延时子程序
8     RET                               ; 子程序返回

```

第 3 行语句是调 DEX 短暂延时子程序，该短暂延时的时间决定蜂鸣器响声的频率。延时短，则频率高；延时长，则频率低。第 5 行语句决定蜂鸣器响声长度。第 7 行语句调用 DELAY 延时子程序，该延时时间决定连续两次响声之间的间隔时间大小。

3. 工作指示灯闪烁子程序

```

1 LED_BL:
2     MOV      R4, #6                 ; 闪烁次数
3 LE1:    CPL      WLED               ; 位反向输出
4     MOV      R5, #25
5     ACALL    DELAY               ; 延时 50ms
6     DJNZ    R4, LE1                ; 子程序返回
7     RET

```

第2行语句是赋予闪烁次数，即每调用一次该子程序，工作指示灯闪烁6次。第3行语句位反向输出，即原来截止，现在导通；原来导通，现在截止。第4行语句通过对R5赋值来调节DELAY的延时时间，其延时时间为50ms。第5行语句调用DELAY延时子程序使工作灯亮（或灭）后停一段时间。

4. 延时子程序

调用该延时子程序之前，要对R5赋值。

```

1  DELAY
2      MOV      R6,#10
3  D1:  MOV      R7,#100
4      DJNZ    R7,$
5      DJNZ    R6,D1
6      DJNZ    R5,  DELAY      ;总延时 2ms×R5
7      RET      ;子程序返回

```

5. 蜂鸣器短暂延时子程序

```

1  DEX:
2      MOV      R7, #180
3  DE1: NOP
4      DJNZ    R7, DE1
5      RET      ; 子程序返回

```

23.1.7 时钟主程序

主程序通过调用子程序，将各部分功能有机地组合到一起，完成单片机时钟的总体设计要求，主程序前要对程序进行初始化。

1. 程序初始化

主程序前要对程序进行初始化，主要完成定义计时单元、定义引脚、定义中断服务程序人口地址等。

(1) 定义存放计时单元地址。

```

1  BUF    EQU  30H          ;30~33H连续4字节存放显示器数据
2  HOUR   EQU  34H          ;存放小时变量
3  MIN    EQU  35H          ;存放分钟变量
4  SEC    EQU  36H          ;存放秒钟变量
5  DEDA   EQU  37H          ;存放5ms计数值

```

(2) 按键输入引脚定义。

```

1  K1    EQU  P2.4          ;按键K1引脚定义
2  K2    EQU  P2.5          ;按键K2引脚定义
3  K3    EQU  P2.6          ;按键K3引脚定义
4  K4    EQU  P2.7          ;按键K4引脚定义

```

(3) 蜂鸣器和指示灯引脚定义。

1	SPK EQU P3.4	;蜂鸣器控制信号
2	WLED EQU P1.0	;工作指示灯引脚定义

(4) 程序开始执行地址。

1	ORG 0H	;程序代码由地址 0 开始执行
2	JMP MAIN	
3	ORG 0BH	;设置定时器 0 中断地址
4	JMP TO_INT	

2. 主程序

主程序通过调用子程序，将各部分功能有机地组合到一起，并不断循环扫描 K1 ~ K4 功能键。在此主程序里只有 K1 键赋予了设置目前时间的功能，而其他 K2 ~ K4 键为空键，预留以后使用。如果按下 K1 键，则执行相应按键控制程序，按键控制程序执行完毕后，仍然回到主程序循环扫描，其主程序如下：

1	MAIN:	;主程序
2	ACALL BZ	;蜂鸣器两次鸣响
3	ACALL BZ	
4	ACALL LED_BL	;LED 闪动
5		
6	MOV A, #0C0H	
7	MOV P0, A	;加载显示器初值数据
8		
9	ACALL INIT	;计时单元清 0
10	ACALL INIT_TIMER	;设置定时器
11		
12	LOOP:	;无穷循环
13	ACALL CONV	
14	ACALL SCANS	;扫描显示器
15	;扫描 K1 键	
16	JB K1, M1	;未按下 K1 键，则继续扫描
17	ACALL LED_BL	;LED 闪动
18	ACALL SET_TIME	;设置目前时间
19	JMP LOOP	;跳转到 LOOP 处执行
20	;扫描 K2 键	
21	M1: JB K2, M2	;未按下 K2 键，则继续扫描
22	JMP LOOP	;继续循环执行
23	;扫描 K3 键	
24	M2: JB K3, M3	;未按下 K3 键，则继续扫描
25	JMP LOOP	;继续循环执行
26	;扫描 K4 键	

```

27 M3: JB      K4,M4          ;未按下 K4 键，则继续扫描
28     JMP     LOOP           ;跳转到 LOOP 处执行
29 M4:          ;
30     JMP     LOOP           ;跳转到 LOOP 处执行

```

第1~4行语句使蜂鸣器两次鸣响，并使LED闪动，表示程序开始执行。

第6行、7行语句加载显示器初值。

第9行、10行语句使计时单元清0，并开始启动定时器，计时开始。

第12~30行语句为无穷循环语句，首先调用转换子程序和显示子程序，接着是扫描K1~K4功能键，有按键时则执行按键处理子程序，执行完后回到主程序重复循环。

23.1.8 简单时钟程序清单

程序运行从主程序开始，通过主程序调用各子程序来完成时钟功能。

1. 程序标号

- MAIN：主程序。
- INIT_TIMER：定时器设置子程序。
- TO_SRV：中断服务子程序。
- CONV：转换时分子程序。
- DISP：扫描显示子程序。
- SET_TIME：设置目前时间子程序。
- INIT：计时单元清0子程序。
- BZ：蜂鸣器鸣响一声子程序。
- LED_BL：工作指示灯闪烁子程序。
- DELAY：延时子程序。
- DEX：蜂鸣器短暂延时子程序。

2. 程序清单

在单片机时钟程序中，最前面是程序初始化部分，接着是主程序部分，主程序后面是各子程序。在汇编语言程序中，主程序必须放在各子程序的前面，而各子程序之间的位置可以掉换。下面是简单时钟程序清单。

```

;-----;
;文件名称：SZ01.ASM
;程序功能：时钟
;-----;

0 ;----- 程序初始化 -----
1 ; 定义存放计时单元地址
2     BUF    EQU    30H          ;30~33H 连续 4 个字节存放显示器数据
3     HOUR   EQU    34H          ;存放小时变量
4     MIN    EQU    35H          ;存放分钟变量

```

```

5      SEC    EQU    36H          ;存放秒钟变量
6      DEDA   EQU    37H          ;存放 5ms 计数值
7
8 ;按键输入引脚定义
9      K1     EQU    P2.4        ;按键 K1 引脚定义
10     K2     EQU    P2.5        ;按键 K2 引脚定义
11     K3     EQU    P2.6        ;按键 K3 引脚定义
12     K4     EQU    P2.7        ;按键 K4 引脚定义
13
14 ;蜂鸣器和指示灯引脚定义
15     SPK    EQU    P3.4        ;蜂鸣器控制信号
16     WLED   EQ     P1.0        ;工作指示灯引脚定义
17
18 ;程序开始执行地址
19     ORG    0H                ;程序代码由地址 0 开始执行
20     JMP    MAIN
21     ORG    0BH              ;定时器 T0 中断地址设置
22     JMP    TO_SRV
23
24 ;----- 主 程 序 -----
25
26 MAIN:
27     ACALL   BZ              ;蜂鸣器连续两次鸣响一声
28     ACALL   BZ
29     ACALL   LED_BL         ;LED 闪动，表示程序开始执行
30     ACALL   INIT            ;初始变化量
31     ACALL   INIT_TIMER     ;设置定时器
32
33 ;加载显示器初始值数据
34     MOV     A, #0C0H
35     MOV     P0, A
36
37 ;无穷循环
38 LOOP:                         ;无穷循环
39     ACALL   CONV
40     ACALL   DISP            ;扫描显示
41
42     JB     K1, M1          ;未按下 K1 键，则继续扫描
43     ACALL   LED_BL         ;LED 闪动
44     ACALL   SET_TIME       ;设置目前时间
45     JMP    LOOP            ;跳转到 LOOP 处执行
46
47 M1:   JB     K2, M2          ;未按下 K2 键，则继续扫描

```

```

48    JMP      LOOP          ;跳转到 LOOP 处执行
49
50 M2:  JB       K3, M3      ;未按下 K3 键，则继续扫描
51    JMP      LOOP          ;跳转到 LOOP 处执行
52
53 M3:  JB       K4, M4      ;未按下 K4 键，则继续扫描
54    JMP      LOOP          ;跳转到 LOOP 处执行
55
56 M4:
57    JMP      LOOP          ;跳转到 LOOP 处执行
58
59;----- 实现走时功能的子程序 -----
60
61;使用定时器 T0 模式 0 计时
62 INIT_TIMER:                 ;初始化定时器
63    MOV      TMOD, #0000000B   ;设置定时器 T0 工作模式为 0
64    MOV      IE, #10000010B    ;启用定时器 T0 中断产生
65    MOV      TL0, #(8192-5000)MOD 32 ;加载初始值
66    MOV      TH0, #(8192-5000)/32
67    SETB    TR0            ;启动定时器 T0 开始计时
68    RET
69;-----
70;中断服务程序
71 TO_SRV:
72    PUSH    ACC           ;A 值压入堆栈
73    MOV      TL0, #(8192-5000)MOD 32 ;重加载初始值
74    MOV      TH0, #(8192-5000)/32
75    INC      DEDA          ;加 1
76;秒输出
77    MOV      A, DEDA
78    CJNE    A, #200, TT1     ;是否 1s 到了
79    MOV      DEDA, #0         ;计数值清 0
80    CPL      WLED          ;LED 灯亮灭变换
81    INC      SEC           ;秒计数加 1
82    MOV      A, SEC
83    CJNE    A, #60, TT1      ;是否 1min 到了
84;分输出
85    INC      MIN           ;分计数加 1
86    MOV      SEC, #0         ;秒计数清 0
87    MOV      A, MIN
88    CJNE    A, #60, TT1      ;是否 1h 到了
89;时输出
90    INC      HOUR          ;小时计数加 1

```

```

91    MOV      MIN, #0          ;分计数清 0
92    MOV      A, HOUR
93    CJNE    A,#24, TT1        ;是否 24h 到了
94    MOV      SEC, #0          ;秒钟变量清 0
95    MOV      MIN, #0          ;分钟变量清 0
96    MOV      HOUR, #0         ;小时变量清 0
97    TT1:
98    POP     ACC              ;将 A 值由堆栈取出
99    RETI
100
101 ;----- 实现显示功能的子程序 -----
102
103
104 CONV:
105 ;转换小时数据
106    MOV      A, HOUR
107    MOV      B, #10
108    DIV     AB
109    MOV      DPTR, #TABLE    ;查表转换
110    MOVC   A, @A+DPTR
111    MOV      BUF, A
112    MOV      A, B
113    MOVC   A, @A+DPTR
114    MOV      BUF+1, A
115 ;转换分钟数据
116    MOV      A, MIN
117    MOV      B, #10
118    DIV     AB
119    MOV      DPTR, #TABLE    ;查表转换
120    MOVC   A, @A+DPTR
121    MOV      BUF+2, A
122    MOV      A, B
123    MOVC   A, @A+DPTR
124    MOV      BUF+3, A
125    RET
126
127 ;字型数据编码表
128 TABLE:
129    DB  0C0H,0F9H,0A4H,0B0H
130    DB  99H,92H,82H,0F8H
131    DB  80H,90H,88H,83H
132    DB  0C6H,0A1H,86H,8EH
133

```

```

134 ; - - - - -
135 ;扫描显示
136 DISP:
137 MOV R0, #BUF          ;指向显示器显示缓冲区起始地址
138 MOV R2, #4             ;循环执行 4 次
139 MOV ACC, #11110111b    ;加载扫描信号初值 11110111B
140 S1:
141 PUSH ACC
142 MOV A, @R0            ;取出显示器数据
143 MOV P0, A              ;由 P0 送出一位显示数据
144 POP ACC
145 MOV P2, ACC            ;由 P2 送出扫描输出信号
146 MOV R5, #2              ;延迟
147 ACALL DELAY
148 ;改变扫描码 EX:XXXX1011
149 RR A                  ;A 向右移动一位
150 INC R0                ;显示器显示缓冲区地址加 1
151 DJNZ R2, S1            ;判断是否继续执行循环
152 RET
153
154 ; - - - - - 实现调整时间子程序 - - - - -
155
156 ;设置目前时间
157 SET_TIME:
158 CLR TR0               ;定时器暂停
159 MOV SEC, #0             ;秒钟变量清 0
160 L0:
161 ACALL DISP             ;扫描显示
162
163 JB K2, L1              ;未按下 K2 键，则继续扫描
164 JNB K2, $                ;有键按下，则等待放开来
165 ;K2 小时调整
166 INC HOUR               ;小时计数加 1
167 MOV A, HOUR
168 CJNE A, #24, L11        ;是否 24h 到了
169 MOV HOUR, #0             ;小时变量清 0
170 L11:
171 ACALL CONV             ;转换显示数据
172 ACALL DISP             ;扫描显示
173 JMP L0                 ;跳转到 L0 处执行
174
175 L1: JB K3, L2           ;未按下 K3 键，则继续扫描
176 JNB K3, $                ;有键按下，则等待放开

```

```

177 ;K3 分钟调整
178 INC MIN ;分钟计数加 1
179 MOV A, MIN
180 CJNE A, #60, L21 ;是否 60min 到了
181 MOV MIN, #0 ;分钟变量清 0
182 L21:
183 ACALL CONV ;转换显示数据
184 ACALL DISP ;扫描显示
185 JMP L0 ;跳转到 L0 处执行
186
187 L2: JB K4, L0 ;未按下 K4 键，则继续扫描
188 JNB K4, $ ;按下，则等待放开
189 ACALL BZ ;蜂鸣器鸣响一声
190 ;K4 设置完成
191 ACALL LED_BL ;LED 闪动
192 SETB TR0 ;启动定时器
193 RET
194
195 ;----- 喇叭和指示灯等其他子程序 -----
196
197 ;计时单元清 0
198 INIT:
199 MOV DEDA, #0 ;5ms 计数值清 0
200 MOV SEC, #0 ;秒钟变量清 0
201 MOV MIN, #0 ;分钟变量清 0
202 MOV HOUR, #0 ;小时变量清 0
203 RET
204 ;
205 ;蜂鸣器
206 BZ: ;蜂鸣器鸣响一声
207 MOV R6, #250
208 B1: ACALL DEX
209 CPL SPK
210 DJNZ R6, B1
211 MOV R5, #50
212 ACALL DELAY
213 RET
214
215 ;-----
216 ;工作指示灯
217 LED_BL: ;工作指示灯闪烁
218 MOV R4, #6
219 LE1: CPL WLED ;位反向

```

```

220     MOV      R5,#25
221     ACALL   DELAY          ;延迟 50ms
222     DJNZ    R4, LE1
223     RET
224 ; - - - - -
225 ;延时子程序
226     DELAY:           ;总延迟时间 2ms×R5
227     MOV      R6,#10
228     D1:    MOV      R7,#100
229     DJNZ    R7,$
230     DJNZ    R6, D1
231     DJNZ    R5, DELAY
232     RET
233 ; - - - - -
234 ; 蜂鸣器短暂延时子程序
235     DEX:    MOV      R7,#180      ;短暂延时
236     DE1:    NOP
237     DJNZ    R7, DE1
238     RET
239 ; - - - - -
240     END

```

3. 程序说明

1~23：程序初始化部分。定义变量、引脚和中断服务程序入口地址。

26~57：主程序部分。调用子程序和扫描 K1~K4 功能键，并无穷循环。

59~100：实现走时功能部分。启动定时器，并通过中断服务程序开始计时，输出秒、分和小时。

102~153：实现显示功能部分。将中断服务程序输出的秒、分和小时数据转换成适合数码管显示的字符并输出。

154~194：实现调整时间功能的子程序，主要是设置目前时间。

195~239：喇叭和指示灯等其他子程序。其中：第 198~203 行为计时单元清 0 子程序；第 206~213 行为蜂鸣器鸣响一声子程序；第 217~213 行为工作指示灯闪烁子程序；第 226~232 行为延时子程序；第 235~238 行为蜂鸣器短暂延时子程序。

240：程序结束。

23.2 带定时功能的闹铃时钟设计

在上节介绍的简单时钟基础上，硬件不变，通过软件扩展时钟功能，使之成为带定时闹铃的时钟。

23.2.1 闹钟结构与原理

1. 带定时闹铃时钟结构

带定时闹铃时钟的硬件结构与上节时钟完全相同，所增加的定时闹铃功能是通过增加软件部分来实现的。

2. 使用方法

接通电源后，蜂鸣器连续两次发出响声，同时工作指示灯 LED 闪动，表示程序开始工作，数码管显示“0000”。

(1) 设置现在时间

K1 为设置现在时间功能键，按一下 K1 键便进入设置现在时间状态，然后，按 K2 键为小时调整；按 K3 键为分钟调整。设置完成后要按一下 K4 键，使之恢复正常时钟走时状态。

(2) 设置闹铃时间

走时的时间设置完并按 K4 键，进入正常的时钟走时状态后再按一下 K2 键，此时 K2 键为设置闹铃功能键，进入设置闹铃时间状态后，此时再按一下 K2 键即为小时调整；按 K3 键为分钟调整。

(3) 闹铃 ON/OFF 的设置

闹铃时间设置完成后，再按 K4 键，此时的 K4 键为设置闹铃 ON/OFF 键，若设为 ON，则启动闹铃时会听到连续 3 次发出响声；若设为 OFF，则关闭闹铃时会听到发出 1 次响声。

3. 基本原理

定时闹铃的基本原理是：首先设置闹铃时间，然后，将走时时间与设置的闹铃时间不断进行比较，当走时时间与闹铃时间相等时，说明定时时间已到，响起铃声。

23.2.2 闹铃功能主要子程序

闹铃功能由设置闹铃时间子程序、加载闹铃时间子程序、检查闹铃时间子程序和执行闹铃时间处理子程序构成。

1. 设置闹铃时间子程序

SET_ATIME 是设置闹铃时间子程序，其功能是设置闹铃的启动时间。该子程序结构与上节讲的调整时间子程序 SET_TIME 基本相同。

注意 在程序中小时和分钟数据存储的变量名称是不同的，前者将小时数据存储到变量 HOUR 处；分钟数据存储到变量 MIN 处。而设置的闹铃时间是将小时数据存储到变量 HH 处；将分钟数据存储到变量 MM 处。其中，HH 与 MM 已经在程序的初始化部分里进行了定义。

设置闹铃时间子程序如下：

```

1 SET_ATIME:
2     ACALL CONVA          ;加载闹铃时间数据
3 N0:
4     ACALL DISP           ;扫描显示
5 ; K2 调整小时
6     JB    K2, N1          ;未按下 K2 键，则继续扫描
7     JNB   K2, $           ;按下，则等待放开
8
9     INC   HH              ;小时数加 1
10    MOV   A, HH
11    CJNE  A, #24, N11    ;是否 24h 到了
12    MOV   HH, #0          ;小时清 0
13 N11:
14    ACALL CONVA          ;加载闹铃时间数据
15    ACALL DISP           ;扫描显示
17    JMP   N0              ;跳转到 N0 处执行程序
18 ; K3 调整分钟
19 N1:   JB    K3, N2          ;未按下 K3 键，则继续扫描
20     JNB   K3, $           ;按下，则等待放开
21
22     INC   MM              ;分钟数加 1
23     MOV   A, MM
24     CJNE  A, #60, N21    ;是否 60min 到了
25     MOV   MM, #0          ;分钟数清为 0
26 N21:
27     ACALL CONVA          ;加载闹铃时间数据
28     ACALL DISP           ;扫描显示
29     JMP   N0              ;跳转到 N0 处执行
30 ;K4 设置完成
31 N2:   JB    K4, N0          ;未按下 K4 键，则继续扫描
32     JNB   K4, $           ;等 K4 键放开
33 ;设置完成
34     ACALL LED_BL         ;LED 闪动
35     ACALL CONV           ;加载现在时间数据
36     RET

```

程序说明：

当第一次按 K2 键时(按过 K4 键后再按 K2 键也为第一次)，此时的 K2 键为设置闹铃功能键，使程序进入设置定时闹铃状态。

第 1~4 行语句，加载闹铃时间数据和扫描显示。

第 6~17 行语句，调整小时。首先判断是否按下小时调整 K2 键，如果按下后再放开表示进行小时调整，否则继续对 K2 键进行扫描。

第 19~29 行语句，调整分钟时间。首先判断是否按下分钟调整 K3 键，如果按下后再放

开，表示进行分钟调整，否则继续对 K3 键进行扫描。

第 31~36 行语句，判断是否设置完成。首先判断是否按下设置完成 K4 键，如果按下后再放开，表示设置定时闹铃已经完成，子程序返回。

2. 加载闹铃时间数据子程序

CONVA 为加载闹铃时间数据子程序，其功能是将设置的闹铃时间转换成适应 LED 数码管显示的时间，并加载到数码管显示内存，其程序结构与上节 CONV 转换时分子程序基本相同，程序如下：

```

1 CONVA:
2 ;转换小时数据
3     MOV      A,  HH      ;
4     MOV      B,  #10          ;16 进制换成 10 进制
5     DIV      AB          ; A ÷ B , 商存 A , 余数存 B
6     MOV      DPTR, #TABLE    ;查表转换
7     MOVC    A, @A+DPTR
8
9     MOV      BUF,  A
10    MOV     A,  B
11    MOVC    A, @A+DPTR
12    MOV      BUF+1, A
13 ;转换分钟数据
14    MOV      A,  MM
15    MOV      B,  #10          ;16 进制换成 10 进制
16    DIV      AB          ; A ÷ B , 商存 A , 余数存 B
17
18    MOV      DPTR, #TABLE    ;查表转换
19    MOVC    A, @A+DPTR
20
21    MOV      BUF+2, A
22    MOV     A,  B
23    MOVC    A, @A+DPTR
24    MOV      BUF+3, A
25    RET          ;子程序返回

```

第 1~12 行语句，转换小时数据。先进行 10 进制转换，再查表取字符码，最后存入数码管显示内存 BUF 和 BUF+1。

第 13~25 行语句，转换分钟数据。先进行 10 进制转换，再查表读取字符码，最后存入数码管显示内存 BUF+2 和 BUF+3。

3. 检查闹铃时间子程序

检查闹铃时间有两个子程序：检查是否启动闹铃子程序和检查闹铃时间是否到了子

程序。

(1) 检查是否启动闹铃子程序。

TIME_PRO 为检查是否启动闹铃子程序。在程序初始化部分定义了闹铃设置标志 ALB，当闹铃标志 ALB=0 时，闹铃为 OFF，即闹铃停止；当闹铃标志 ALB=1 时，闹铃为 ON，即闹铃启动。

程序第 2 行通过 JB 指令判断闹铃标志 ALB 是 0 还是 1，如果是 1，则表示已经设置了闹铃为启动状态。接着程序运行跳转到标号 TI3 处，进入检查闹铃时间是否到了的子程序，如果 ALB=0，则子程序返回。检查是否启动闹铃子程序如下：

```
1 TIME_PRO:
2     JB      ALB, TI3           ; 判断闹铃标志 ALB
3     RET               ; 子程序返回
```

(2) 检查闹铃时间是否到了子程序。

TI3 为检查闹铃时间是否到了子程序。其工作原理是：先比较走时小时 HOUR 是否与闹铃定时小时 HH 数据相一致，再比较走时分钟 MIN 是否与闹铃定时分钟 MM 数据相一致，如果时、分都相一致，说明闹铃定时时间已经到了，执行闹铃时间处理子程序，否则继续进行检查比较，其检查闹铃时间程序如下：

```
1 TI3:
2 ; 检查闹铃小时时间
3     MOV      A, HOUR
4     MOV      B, HH
5     CJNE    A, B, BK           ; HOUR 与 HH 比较
6 ; 检查闹铃分钟时间
7     MOV      A, MIN
8     MOV      B, MM
9     CJNE    A, B, BK           ; MIN 与 MM 比较
10 ; 执行检查时间处理
11     ACALL   TIME_OUT
12     BK:
13     RET               ; 子程序返回
```

4. 执行闹铃时间处理子程序

TIME_OUT 为执行闹铃时间处理子程序，功能是响起闹铃，按 K4 键闹铃停止，程序如下：

```
1 TIME_OUT:
2 X1: ACALL    LED_BL          ; LED 闪动
3     ACALL    BZ              ; 蜂鸣器发声
4     JB      K4, X1           ; 判断是否按下 K4 键
5     JNB    K4, $             ; 若按下，则等待按键放开
6     ACALL    LED_BL          ; LED 闪动
7     CLR     ALB             ; 清除闹铃位
8     RET               ; 子程序返回
```

第2行语句是调用工作灯闪烁子程序；第3行语句是调用蜂鸣器发声；第4行语句判断是否按下K4键，如果没有按下K4键，程序返回第2行重复执行，使闹铃声不停。如果按下并再放开K4键，程序将向下执行6、7、8行语句，第7行语句使闹铃标志ALB=0，闹铃停止，第8行语句程序返回。

23.2.3 闹钟主程序

在上节时钟主程序中增加了定时闹铃部分，并在程序开头的初始化部分增加了对定时闹铃计时单元的定义。

1. 程序初始化

在程序初始化部分里，增加了定义定时闹铃计时单元的语句，程序如下：

```

1 ; 定义存放计时单元地址
2 ; .....
3 ALB EQU 20H.0 ;闹铃设置标志
4 ; .....
5 BUF EQU 30H ;30~33H 连续4个字节存放显示器数据
6 HOUR EQU 34H ;存放小时变量
7 MIN EQU 35H ;存放分钟变量
8 SEC EQU 36H ;存放秒钟变量
9 DEDA EQU 37H ;存放5ms 计数值
10 ; .....
11 HH EQU 38H ;闹铃设置小时变量
12 MM EQU 39H ;闹铃设置分钟变量
13 ; .....
14 ;按键输入引脚定义
15 K1 EQU P2.4 ;按键K1引脚定义
16 K2 EQU P2.5 ;按键K2引脚定义
17 K3 EQU P2.6 ;按键K3引脚定义
18 K4 EQU P2.7 ;按键K4引脚定义
19
20 ;蜂鸣器和指示灯引脚定义
21 SPK EQU P3.4 ;蜂鸣器控制信号引脚
22 WLED EQU P3.7 ;工作指示灯引脚定义
23
24 ;程序开始执行地址
25 ORG 0H ;程序代码由地址0开始执行
26 JMP MAIN
27 ORG 0BH ;定时器T0中断地址设置
28 JMP TO_INT

```

新增加了3行语句：第3行语句设置闹铃标志；第11行语句设置闹铃小时变量，即将闹铃小时数据存储在变量名为HH中；第12行语句设置闹铃分钟变量，即将闹铃分钟数据存储

在变量 MM 中。

2. 主程序

定时闹铃时钟主程序在简单时钟主程序基础上增加了定时闹铃部分，程序如下：

```

1 MAIN:                                ;主程序
2     ;.....
3     CLR    ALB                      ;清除闹铃标志
4     ;.....
5     ACALL  BZ                      ;蜂鸣器连续两次鸣响
6     ACALL  BZ
7     ACALL  LED_BL                  ;LED 闪动，表示程序开始执行
8     ACALL  INIT                     ;初始变化量
9     ACALL  INIT_TIMER              ;设置定时器
10
11;加载显示器初值数据
12    MOV    A, #0C0H                 ;关闭显示器
13    MOV    P0, A
14
15
16;无穷循环
17 LOOP:                               ;无穷循环
18     ACALL  CONV
19     ;.....
20     ACALL  TIME_PRO               ;检查闹铃时间
21     ;.....
22     ACALL  DISP                   ;扫描显示
23
24     JB     K1, M1                 ;未按下 K1 键，则继续扫描
25     ACALL  LED_BL                ;LED 闪动
26     ACALL  SET_TIME              ;设置目前时间
27     JMP    LOOP                  ;跳转到 LOOP 处执行
28
29 ;设置闹铃时间
30 M1:JB     K2,M2                  ;未按下 K2 键则继续扫描
31     ;.....
32     ACALL  LED_BL
33     ACALL  SET_ATIME             ;设置闹铃时间
34     ;.....
35     JMP    LOOP                  ;跳转到 LOOP 处执行
36
37 M2:J B   K3, M3                ;未按下 K3 键，则继续扫描
38     JMP    LOOP                  ;跳转到 LOOP 处执行

```

```

39
40 M3: JB K4, M4           ;未按下 K4 键，则继续扫描
41     .....
42     CPL ALB
43     JNB ALB, M31
44
45     ACALL BZ           ;闹铃启动，连续 3 次发出响声
46     ACALL BZ
47     ACALL BZ
48     .....
49     JMP LOOP            ;跳转到 LOOP 处执行
50     .....
51 M31:
52     ACALL BZ           ;闹铃停止，发出一声响
53     JMP LOOP            ;跳转到 LOOP 处执行
54     .....
55 M4:
56     JMP LOOP            ;继续循环执行

```

新增加的定时闹铃语句有：第 3 行清除闹铃标志，即闹铃标志位 ALB=0；第 20 行调用检查闹铃时间子程序 TIME_PRO，将走时时间不断与定时时间进行比较，看是否一致。

当按下 K2 设置闹铃时间功能键时，便执行第 32、33 行的按键控制程序语句，调用工作指示灯和设置闹铃时间子程序，以便设置闹铃时间。

第 42 ~ 47 行和第 51 ~ 54 行语句对闹铃状态进行设置。

通过按 K4 键启动闹铃时会听到连续 3 次发出响声；停止闹铃时会听到发出一声响声。其他行语句作用与上节相同。

23.2.4 闹钟程序清单

在上节简单时钟程序基础上增加了定时闹铃程序部分，其中，新增加的个别语句插入到程序前边的定义部分里和主程序中，而定时闹铃子程序则放在整个程序的后边，程序代码如下。

1. 程序标号

(1) 简单时钟程序标号。

- MAIN：主程序。
- INIT_TIMER：定时器设置子程序。
- TO_SRV：中断服务子程序。
- CONV：转换时分子程序。
- DISP：扫描显示子程序。
- SET_TIME：设置目前时间子程序。
- INIT：计时单元清 0 子程序。

- BZ：蜂鸣器鸣响一声子程序。
- LED_BL：工作指示灯闪烁子程序。
- DELAY：延时子程序。
- DEX：蜂鸣器短暂延时子程序。

(2) 增加的定时闹铃子程序。

- SET_ATIME：设置闹铃时间子程序。
- CONVA：加载闹铃时间数据子程序。
- TIME_PRO：检查是否启动闹铃子程序。
- TIME_OUT：执行闹铃时间处理子程序。

2. 程序清单

带定时闹铃单片机时钟程序代码如下：

```

0;-----
1 ;文件名称：SZ02.ASM
2 ;程序功能：带定时闹铃时钟
3 ;-----
4 ;*****简单时钟（含插入闹铃）程序部分 *****
5
6;----- 程序初始化 -----
7 ; 定义存放计时单元地址
8         ;
9     ALB    EQU  20H.0      ;闹铃设置标志
10    ;
11    BUF    EQU  30H          ;30~33H 连续4个字节存放显示器数据
12    HOUR   EQU  34H          ;存放小时变量
13    MIN    EQU  35H          ;存放分钟变量
14    SEC    EQU  36H          ;存放秒钟变量
15    DEDA   EQU  37H          ;存放5ms 计数值
16    ;
17    HH     EQU  38H          ;闹铃设置小时变量
18    MM     EQU  39H          ;闹铃设置分钟变量
19    ;
20 ;按键输入引脚定义
21    K1    EQU  P2.4          ;按键K1引脚定义
22    K2    EQU  P2.5          ;按键K2引脚定义
23    K3    EQU  P2.6          ;按键K3引脚定义
24    K4    EQU  P2.7          ;按键K4引脚定义
25
26 ;蜂鸣器和指示灯引脚定义
27    SPK   EQU  P3.4          ;蜂鸣器控制信号引脚
28    WLED  EQU  P1.0          ;工作指示灯引脚定义
29

```

```

30 ;程序开始执行地址
31 ORG 0H           ;程序代码由地址 0 开始
32 JMP MAIN
33 ORG 0BH          ;定时器 T0 中断地址设置
34 JMP TO_SRV
35
36 ;----- 主 程 序 -----
37
38 ;主程序
39 MAIN:
40 ;.....
41 CLR ALB          ;清除闹铃标志
42 ;....
43 ACALL BZ          ;蜂鸣器连续两次鸣响一声
44 ACALL BZ
45 ACALL LED_BL      ;LED 闪动，表示程序开始执行
46 ACALL INIT         ;初始变化量
47 ACALL INIT_TIMER   ;设置定时器
48
49 ;加载显示器初值数据
50 MOV A, #0C0H
51 MOV P0, A
52
53 ;无穷循环
54 LOOP:             ;无穷循环
55 ACALL CONV
56 ;.....
57 ACALL TIME_PRO    ;检查闹铃时间
58 ;.....
59 ACALL DISP          ;扫描显示
60
61 JB K1, M1          ;未按下 K1 键，则继续扫描
62 ACALL LED_BL        ;LED 闪动
63 ACALL SET_TIME      ;设置目前时间
64 JMP LOOP            ;跳转到 LOOP 处执行
65
66 ;设置闹铃时间
67 M1:JB K2, M2        ;未按下 K2 键，则继续扫描
68 ;.....
69 ACALL LED_BL
70 ACALL SET_ATIME     ;设置闹铃时间
71 ;.....
72 JMP LOOP            ;跳转到 LOOP 处执行

```

```

73
74 M2: JB      K3, M3          ;未按下 K3 键，则继续扫描
75     JMP     LOOP            ;跳转到 LOOP 处执行
76
77 M3: JB      K4, M4          ;未按下 K4 键，则继续扫描
78     ; .....
79     CPL    ALB
80     JNB    ALB, M31
81
82     ;闹铃启动连续 3 次发出响声
83     ACALL  BZ
84     ACALL  BZ
85     ACALL  BZ
86     ; .....
87     JMP     LOOP            ;跳转到 LOOP 处执行
88     ; .....
89 M31:
90     ACALL  BZ            ;闹铃停止，发出一声响
91     JMP     LOOP            ;跳转到 LOOP 处执行
92     ; .....
93 M4:
94     JMP     LOOP            ;跳转到 LOOP 处执行
95
96; - - - - - 实现走时功能的子程序 - - - - -
97
98 ;使用定时器 T0 模式 0 计时
99 INIT_TIMER:                 ;初始化定时器，使用定时器 T0 模式 1 计时
100    MOV    TMOD,#00000000B   ;设置定时器 T0 工作模式 0
101    MOV    IE, #10000010B    ;启用定时器 T0 中断产生
102    MOV    TL0, #(8192-4900)MOD 32 ;加载初始值
103    MOV    TH0, #(8192-4900)/32
104    SETB   TR0            ;启动定时器 T0 开始计时
105    RET
106 ; - - - - -
107 ;中断服务程序
108 TO_SRV:                   ;定时器 T0 计时中断程序每隔 5ms 中断一次
109    PUSH   ACC            ;将 A 值压入堆栈
110    MOV    TL0, #(8192-4900)MOD 32 ;重加载初始值
111    MOV    TH0, #(8192-4900)/32
112    INC    DEDA           ;加 1
113 ;秒输出
114    MOV    A, DEDA
115    CJNE   A, #200, TT1      ;是否 1s 到了

```

```

116    MOV    DEDA, #0          ;计数值清 0
117    CPL    WLED             ;LED 灯亮灭变换
118    INC    SEC              ;秒计数加 1
119    MOV    A, SEC
120    CJNE   A, #60, TT1      ;是否 1min 到了
121    ;分输出
122    INC    MIN              ;分计数加 1
123    MOV    SEC, #0          ;秒计数清 0
124    MOV    A, MIN
125    CJNE   A, #60, TT1      ;是否 1h 到了
126    ;时输出
127    INC    HOUR             ;小时计数加 1
128    MOV    MIN, #0          ;分计数清 0
129    MOV    A, HOUR
130    CJNE   A, #24, TT1      ;是否 24h 到了
131    MOV    SEC, #0          ;秒钟变量清 0
132    MOV    MIN, #0          ;分钟变量清 0
133    MOV    HOUR, #0          ;小时变量清 0
134    TT1:
135    POP    ACC              ;将 A 值由堆栈取出
136    RETI
137
138    ;----- 实现显示功能的子程序 -----
139
140
141    CONV:
142    ;转换小时数据
143    MOV    A, HOUR
144    MOV    B, #10
145    DIV    AB
146    MOV    DPTR, #TABLE      ;查表转换
147    MOVC   A, @A+DPTR
148    MOV    BUF, A
149    MOV    A, B
150    MOVC   A, @A+DPTR
151    MOV    BUF+1, A
152    ;转换分钟数据
153    MOV    A, MIN
154    MOV    B, #10
155    DIV    AB
156    MOV    DPTR, #TABLE      ;查表转换
157    MOVC   A, @A+DPTR
158    MOV    BUF+2, A

```

```

159    MOV    A, B
160    MOVC   A, @A+DPTR
161    MOV    BUF+3, A
162    RET
163
164 ;字型数据编码表
165 TABLE:
166    DB    0C0H,0F9H,0A4H,0B0H
167    DB    99H,92H,82H,0F8H
168    DB    80H,90H,88H,83H
169    DB    0C6H,0A1H,86H,8EH
170
171 ;-----扫描显示
172 ;扫描显示
173 DISP:
174    MOV    R0, #BUF          ;指向显示器显示缓冲区起始地址
175    MOV    R2, #4            ;执行 4 次循环
176    MOV    ACC, #11110111b   ;加载扫描信号初值 11110111B
177 S1:
178    PUSH   ACC
179    MOV    A, @R0           ;读取出显示器数据
180    MOV    P0, A            ;由 P0 送出一位显示器数据
181    POP    ACC
182    MOV    P2, ACC          ;由 P2 送出扫描输出信号
183    MOV    R5, #2            ;延迟一段时间
184    ACALL  DELAY
185 ;改变扫描码 EX:XXXX1011
186    RR    A                ;累加器 A 向右移动一位
187    INC   R0                ;显示器显示缓冲区地址加 1
188    DJNZ  R2, S1           ;判断是否继续执行循环
189    RET
190
191 ;----- 实现调整时间子程序 -----
192
193 ;设置目前时间
194 SET_TIME:
195    CLR    TR0              ;定时器工作暂停
196    MOV    SEC, #0           ;秒钟变量清除为 0
197 L0:
198    ACALL  DISP             ;扫描显示
199
200    JB    K2, L1             ;未按下 K2 键，则继续扫描
201    JNB   K2, $              ;按下，则等待放开

```

```

202 ;K2 小时调整
203 INC HOUR ;小时计数加 1
204 MOV A, HOUR
205 CJNE A, #24, L11 ;是否 24h 到了
206 MOV HOUR, #0 ;小时变量清 0
207 L11:
208 ACALL CONV ;转换显示数据
209 ACALL DISP ;扫描显示
210 JMP L0 ;继续程序执行
211
212 L1: JB K3, L2 ;未按下 K3 键，则继续扫描
213 JNB K3, $ ;按下，则等待放开
214 ;K3 分钟调整
215 INC MIN ;分钟计数加 1
216 MOV A, MIN
217 CJNE A, #60, L21 ;是否 60min 到了
218 MOV MIN, #0 ;分钟变量清 0
219 L21:
220 ACALL CONV ;转换显示数据
221 ACALL DISP ;扫描显示
222 JMP L0 ;跳转到 L0 处执行
223
224 L2: JB K4, L0 ;未按下 K4 键，则继续扫描
225 JNB K4, $ ;按下，则等待放开
226 ACALL BZ ;蜂鸣器鸣响一声
227 ;K4 设置完成
228 ACALL LED_BL ;LED 闪动
229 SETB TR0 ;启动定时器
230 RET
231
232 ; - - - - - 喇叭和指示灯等其他子程序 - - - - -
233
234 ;计时单元清 0
235 INIT:
236 MOV DEDA, #0 ;5ms 计数值清 0
237 MOV SEC, #0 ;秒钟变量清 0
238 MOV MIN, #0 ;分钟变量清 0
239 MOV HOUR, #0 ;小时变量清 0
240 RET
241 ; - - - - -
242 ;蜂鸣器
243 BZ: ;蜂鸣器鸣响一声
244 MOV R6, #250

```

```

245 B1: ACALL DEX
246 CPL SPK
247 DJNZ R6,B1
248 MOV R5, #50
249 ACALL DELAY
250 RET
251
252 ; - - - - -
253 ;工作指示
254 LED_BL: ;工作指示灯闪烁
255 MOV R4,#6
256 LE1:CPL WLED ;位反向
257 MOV R5,#25
258 ACALL DELAY
259 DJNZ R4, LE1 ;延迟 50ms
260 RET
261 ; - - - - -
262 ;延时子程序
263 DELAY: ;总延迟时间 2ms×R5
264 MOV R6,#10
265 D1:MOV R7,#100
266 DJNZ R7,$
267 DJNZ R6, D1
268 DJNZ R5, DELAY
269 RET
270 ; - - - - -
271 ; 蜂鸣器短暂延时子程序
272 DEX: MOV R7,#180 ;短暂延时
273 DE1: NOP
274 DJNZ R7, DE1
275 RET
276
277;***** 增加定时闹铃程序部分 *****
278
279; - - - - - - - - - - 设置闹铃时间 - - - - - - - - -
280 ;设置闹铃时间
281 SET_ATIME:
282 ACALL CONVA ;加载闹铃时间数据
283 N0:
284 ACALL DISP
285 ; - - - - - - - - -
286 JB K2, N1 ;未按下 K2 键，则继续扫描
287 JNB K2, $ ;按下，则等待放开

```

```

288 ;K2 调整小时
289     INC      HH          ;小时数加 1
290     MOV      A, HH
291     CJNE    A, #24,N11   ;是否 24h 到了
292     MOV      HH, #0       ;小时变量清 0
293 N11:
294     ACALL   CONVA        ;加载闹铃时间数据
295     ACALL   DISP         ;扫描显示
296     JMP     N0          ;跳转到 N0 处执行
297 ; - - - - -
298 N1: JB      K3, N2      ;未按下 K3 键，则继续扫描
299     JNB     K3, $        ;按下，则等待放开
300 ;K3 调整分钟
301     INC      MM          ;调整分钟数
302     MOV      A, MM
303     CJNE    A, #60,N21   ;是否 60min 到了
304     MOV      MM, #0       ;分钟变量清 0
305 N21:
306     ACALL   CONVA        ;加载闹铃时间数据
307     ACALL   DISP         ;扫描显示
308     JMP     N0          ;跳转到 N0 处执行
309 ; - - - - -
310 N2: JB      K4, N0      ;未按下 K4 键，则继续扫描
311     JNB     K4, $        ;等 K4 键放开
312 ;K4 设置完成
313     ACALL   LED_BL        ;LED 闪动
314     ACALL   CONV         ;加载现在时间数据
315     RET
316
317; - - - - - 加载闹铃时间数据 - - - - -
318
319
320 CONVA:
321 ;转换小时数据
322     MOV      A, HH
323     MOV      B, #10
324     DIV      AB
325 ; - - - - -
326     MOV      DPTR, #TABLE  ;查表转换
327     MOVC   A, @A+DPTR
328     MOV      BUF, A
329 ; - - - - -
330     MOV      A, B

```

```

331 MOVC A, @A+DPTR
332 MOV BUF+1, A
333 ;转换分钟数据
334 MOV A, MM
335 MOV B, #10
336 DIV AB
337 ; - - - - -
338 MOV DPTR, #TABLE           ;查表转换
339 MOVC A, @A+DPTR
340 MOV BUF+2, A
341 ; - - - - -
342 MOV A, B
343 MOVC A, @A+DPTR
344 MOV BUF+3, A
345 RET
346
347 ; - - - - - 检查闹铃时间处理 - - - - -
348
349 ;检查是否启动闹铃
350 TIME_PRO:
351 JB ALB, TI3
352 RET
353 ; - - - - -
354
355 ;检查闹铃时间是否到了
356 TI3:
357 MOV A, HOUR
358 MOV B, HH
359 CJNE A, B, BK             ;检查闹铃小时时间
360 ; - - - - -
361 MOV A, MIN
362 MOV B, MM
363 CJNE A, B, BK             ;检查闹铃分钟时间
364 ; - - - - -
365 ACALL TIME_OUT            ;检查闹铃时间，到了进行处理
366 BK:
367 RET
368
369 ; - - - - - 执行闹铃时间处理 - - - - -
370
371 TIME_OUT:                 ;所设置的闹铃数据到了
372 X1: ACALL LED_BL          ;LED 闪动
373 ACALL BZ                  ;蜂鸣器发声

```

```

374 ;按下 K4 键停止闹铃
375 JB K4, X1 ;等待是否按下 K4 键
376 JNB K4,$ ;若有按下键，则等待按键放开
377 ACALL LED_BL ;LED 闪动
378 CLR ALB ;清除闹铃位
379 RET
380; -----
381 END

```

3. 程序说明

下面仅就所增加的定时闹铃程序部分加以说明。

在程序初始化部分里增加了第 9 行和 17~18 行语句，设置了闹铃设置标志、定义了闹铃小时变量 HH 和闹铃分钟变量 MM。

K2 为设置闹铃时间功能键，当按下 K2 键时便进入设置闹铃时间状态。

在主程序里第 69~70 行语句是 K2 键功能控制语句，其中第 70 行语句是调用设置闹铃时间子程序 SET_ATIME。

该子程序在设置闹铃时间的同时又调用加载闹铃时间数据子程序，将设置闹铃时间数据转换为适应数码管显示的数据，并加载到显示内存中。再通过扫描程序，使之能够边设置边看到所设置的时间。

设置闹铃时间完成后，按 K4 键使闹铃处于启动状态，主程序中第 79 行语句使闹铃标志位 ALB 反相，第 80 行语句判断 ALB 位是 0 还是 1。如果 ALB=1，则是闹铃处于启动状态，接着执行 79~80 行语句，使蜂鸣器响 3 声，作为闹铃设置在启动状态的提示；如果 ALB=0，电喇叭响 1 声，表明闹铃处于停止状态。

当闹铃时间和启动状态都设置完成后，通过在主程序中第 57 行语句，调用 TIME_PRO 检查闹铃时间子程序，对比走时的时间是否与设置的闹铃时间相一致，如果小时和分钟都一致，说明定时时间已到，这时由第 365 行语句调用 TIME_OUT，执行闹铃时间处理子程序，使闹铃发出响声，直到再按 K4 键闹铃停止，完成由定时到闹铃的全过程。

程序中其他语句与上节解释相同。

23.3 带定时和倒计时功能的时钟设计

在上节闹钟的基础上，硬件不变，通过软件扩展了时钟功能，使之成为带定时闹铃和倒计时的时钟。

23.3.1 结构与原理

1. 带定时和倒计时时钟结构

带定时闹铃和倒计时时钟的结构，在硬件方面与简单时钟结构完全相同，其增加的倒计

时功能是通过增加软件部分来实现的。

2. 使用方法

接通电源后，蜂鸣器连续两次发出响声，同时工作指示灯 LED 闪动，表示程序开始工作，数码管显示“0000”。

(1) 设置现在时间。

K1 键为设置现在时间功能键，按一下 K1 键进入设置现在时间状态，然后按 K2 键为小时调整；按 K3 键为分钟调整。设置完成后要按一下 K4 键，恢复正常时钟走时状态。

(2) 设置闹铃时间。

K2 键为设置闹铃功能键，当走时的时间设置完按过 K4 键后，再按一下 K2 键便进入设置闹铃时间状态，此时再按一下 K2 键，即为小时调整；按 K3 键为分钟调整。

闹铃时间设置完成按过 K4 键后，此时的 K4 键变为设置闹铃 ON/OFF 键，设为启动闹铃时会听到连续 3 次发出声响；设为关闭闹铃时会发出 1 次声响。

(3) 设置倒计时时间。

K3 键为设置倒计时功能键，当按一下 K3 键进入倒计时状态后，K3 和 K2 键便执行第二功能，此时 K2 键为倒计时分钟数增加键，按 K2 倒计时增加；K3 键为倒计时分钟数减少键，按 K3 倒计时减少。倒计时的时间最长为 59min。

倒计时时间设置好后，按一下 K4 键会听到 4 次声响，表示设置完毕，进入倒计时工作状态。倒计时结束后会发出警报声，此时需要按一下 K4 键，使警报声停止，时钟恢复正常走时状态。

在倒计时时，时钟在正常地走时，只是在倒计时期间里 U3、U4 数码管反映的是倒计时的时间，而 U1、U2 数码管仍然反映正常的走时时间。当倒计时结束后 U3、U4 数码管也会立即恢复正常走时时间。

3. 基本原理

倒计时基本原理是：首先设置倒计时时间，然后不断地检查时间秒、分是否都减为 0，如果秒钟数为 0，分钟数为 0，说明倒计时终止，随即调用闹铃处理程序发出声响。

23.3.2 倒计时功能主要子程序

该部分主要由设置倒计时时间子程序、加载倒计时时间子程序和检查倒计时时间子程序构成。

1. 设置倒计时时间子程序

DOWN_ATIME 为设置倒计时时间子程序，功能是设置倒计时时间，其程序如下：

01	DOWN_ATIME:	; 设置倒计时子程序
02	CLR	TR0 ; 定时器工作暂停
03	MOV	SS_D, #1 ; 秒钟变量为 1
04	S0:	

```

05      ACALL    DISP          ;扫描显示
06
07 ;K2 为分钟数增加键
08      JB K2,S12           ;判断是否按下键
09      JNB K2,$            ;等待放开
10
11      INC     MM_D         ;分钟数增加 1
12      MOV     A, MM_D
13      CJNE   A, #60,S11    ;是否 60min 到了
14      MOV     MM_D,#0       ;分钟变量清 0
15 S11:
16      ACALL   DOWN_CONV   ;显示倒数的时间
17      ACALL   DISP         ;扫描
18      JMP     S0           ;循环执行
19 ; -----
20 ;K3 为分钟数减少键
21 S12:    JB     K3, S2      ;判断是否按下键
22      JNB   K3, $          ;等待放开
23 ;分钟数减少
24      DEC     MM_D         ;分钟数减少 1
25      MOV     A, MM_D
26      CJNE   A, #0, S21    ;判断分钟数是否为 0
27      MOV     MM_D, #60      ;分钟数赋值为 60
28 S21:
29      ACALL   DOWN_CONV   ;显示倒数的时间
30      ACALL   DISP         ;扫描显示
31      JMP     S0           ;循环执行
32 ; -----
33 ;K4 为 OK 键

34 S2:    JB     K4, S0      ;未按下 K4，则继续扫描
35      JNB   K4, $          ;等待放开
36      ACALL   BZ           ;连续 4 次鸣响
37      ACALL   BZ
38      ACALL   BZ
39      ACALL   BZ
40      ACALL   LED_BL        ;LED 闪烁
41      SETB   TR0          ;启动定时器
42 ; -----
43 S3:
44      ACALL   DOWN_TIME    ;检查倒计时
45      ACALL   DOWN_CONV    ;显示倒数的时间
46      ACALL   DISP         ;扫描显示

```

```

47      JMP     S3          ;循环
48      RET

```

第 01 ~ 05 行语句，暂停定时使秒钟变量为 1，并调用扫描显示子程序，为设置倒计时做准备。

第 08 ~ 18 行语句，通过按 K2 键使倒计时分钟数增加。

第 21 ~ 31 行语句，通过按 K3 键使倒计时分钟数减少。

第 34 ~ 41 行语句，通过按 K4 键完成倒计时设置，进入倒计时工作状态。

第 43 ~ 47 行语句，通过循环不断检查倒计时时间。

第 48 语句，子程序返回。

2. 加载倒计时时间子程序

DOWN_CONV 为加载倒计时时间子程序，功能是将倒计时的分钟数存入显示内存，以便通过扫描在数码管上显示出来，程序如下：

```

01  DOWN_CONV:
02  MOV      A, MM_D           ;倒计时分钟数存入 A
03  MOV      B, #10            ;设置被除数
04  DIV      AB               ;除法运算，结果 A 存商数，B 存余数
05  MOV      DPTR, #TABLE     ;查表转换
06  MOVC    A, @A+DPTR       ;读取显示码
07  MOV      BUF+2, A         ;商数存入 BUF+2
08  MOV      A, B              ;余数暂存入 A
09  MOVC    A, @A+DPTR       ;读取显示码
10  MOV      BUF+3, A         ;余数存入 BUF+3
11  RET

```

DOWN_CONV 加载倒计时子程序的结构，与前边的转换时分子程序和加载闹铃时间数据子程序基本一致，只是存入倒计时分钟数据的变量名为 MM_D，该变量名已经在程序的开始程序初始化部分进行了定义。时钟走时的时间数据、定时闹铃时间的数据和倒计时时间的数据必须存在不同的变量名中。

3. 检查倒计时时间子程序

DOWN_TIME 为检查倒计时时间子程序，功能是检查倒计时的时间，当秒数为 0 而且分钟数也为 0 时，便执行闹铃处理程序，开始闹铃，表明倒计时终止，其程序如下：

```

01  DOWN_TIME:                 ;检查倒计时
02  MOV      A, SEC            ;加载现在秒数
03  MOV      B, SEC0           ;载入旧秒数
04  CJNE    A, B, D01          ;现在秒数与旧秒数比较
05  RET
06  D01:

```

```

07    MOV     SEC0, SEC          ;记录旧的秒数
08    DEC     SS_D              ;秒数减 1
09    MOV     A, SS_D
10    CJNE    A, #0, D11        ;秒数与 0 比较
11    MOV     A, MM_D
12    CJNE    A, #0, D12        ;分钟数与 0 比较
13    ;秒钟数为 0 且分钟数为 0 则倒数终止
14    ACALL   TIME_OUT         ;倒数终了执行闹铃处理程序
15    JMP     LOOP
16    RET
17    ; - - - - -
18 D12:
19    MOV     SS_D, #59          ;秒钟数为 59
20    DEC     MM_D              ;分钟数减 1
21
22 D11:
23    ACALL   DOWN_CONV         ;加载倒计时时间
24    RET                      ;子程序返回

```

第 01~05 行语句，用现在秒数与旧秒数比较，不相等时，说明时间已经过了 1s，则更新倒数时间并检查是否倒数终止。

第 06~10 行语句，记录旧的秒数，更新倒数时间，使秒数减 1。接下来通过秒数与 0 比较，看秒钟数是否为 0。不为 0，则执行第 22~24 行语句，显示倒计时时间，子程序返回；如果秒钟数为 0 时，程序向下执行第 11~12 行语句。

第 11~16 行语句，其中第 11~12 行语句将分钟数与 0 比较，看分钟数是否为 0。如果秒钟数为 0 而且分钟数也为 0，则倒数终止，并通过第 14 行语句执行闹铃处理程序；如果分钟数不为 0，则执行 18~20 行语句。

第 18~20 行语句，当秒钟数为 0，分钟数不为 0 时，说明倒计时的时间未到，分钟数应该继续递减，即秒数每次出现为 0，分钟数应减 1，并且存储秒数变量 SS_D 应该重新赋值。

第 22~24 行语句，调用加载倒计时时间子程序，检查倒计时时间子程序返回。

23.3.3 带倒计时闹钟程序清单

1. 程序标号

(1) 简单时钟程序标号。

- MAIN：主程序。
- INIT_TIMER：定时器设置子程序。
- TO_SRV：中断服务子程序。
- CONV：转换时分子程序。
- DISP：扫描显示子程序。

- SET_TIME：设置目前时间子程序。
- INIT：计时单元清0子程序。
- BZ：蜂鸣器鸣响一声子程序。
- LED_BL：工作指示灯闪烁子程序。
- DELAY：延时子程序。
- DEX：蜂鸣器短暂延时子程序。

(2) 增加的定时闹铃子程序。

- SET_ATIME：设置闹铃时间子程序。
- CONVA：加载闹铃时间数据子程序。
- TIME_PRO：检查是否启动闹铃子程序。
- TIME_OUT：执行闹铃时间处理子程序。
- DOWN_ATIME：设置倒计时时间子程序。

(3) 增加的倒计时子程序。

- DOWN_CONV：加载倒计时时间子程序。
- DOWN_TIME：检查倒计时时间子程序。

2. 程序清单

```

01;-----
02 ;文件名称：SZ03.ASM
03 ;程序功能：带定时闹铃及倒计时时钟
04 ;-----
05;***** 闹钟(含插入倒计时)程序部分 *****
06
07 ;----- 程序初始化 -----
08 ; 定义存放计时单元地址
09     ALB    EQU    20H.0          ;闹铃设置标志
10     BUF    EQU    30H          ;30~33H连续4个字节存放显示器数据
11     HOUR   EQU    34H          ;存放小时变量
12     MIN    EQU    35H          ;存放分钟变量
13     SEC    EQU    36H          ;存放秒钟变量
14     DEDA   EQU    37H          ;存放5ms计数值
15     HH     EQU    38H          ;闹铃设置小时变量
16     MM     EQU    39H          ;闹铃设置分钟变量
17     ;
18     SEC0   EQU    3AH          ;存放旧的秒数
19     MM_D   EQU    3BH          ;倒数时间分钟数
20     SS_D   EQU    3CH          ;倒数时间秒钟数
21     ;
22 ;按键输入引脚定义
23     K1     EQU    P2.4          ;按键K1引脚定义
24     K2     EQU    P2.5          ;按键K2引脚定义

```

```

25     K3      EQU      P2.6          ;按键 K3 引脚定义
26     K4      EQU      P2.7          ;按键 K4 引脚定义
27
28 ;蜂鸣器和指示灯引脚定义
29     SPK     EQU      P3.4          ;蜂鸣器控制信号引脚定义
30     WLED    EQU      P1.0          ;工作指示灯引脚定义
31
32 ;程序开始执行地址
33     ORG      0H                  ;程序代码由地址 0 开始
34     JMP      MAIN
35     ORG      0BH                  ;定时器 T0 中断地址设置
36     JMP      TO_SRV
37
38 ;----- 主 程 序 -----+
39
40 MAIN:                           ;主程序
41     CLR      ALB                ;清除闹铃标志
42     ACALL    BZ                 ;蜂鸣器连续两次鸣响
43     ACALL    BZ
44     ACALL    LED_BL             ;LED 闪动，表示程序开始执行
45     ACALL    INIT               ;初始变化量
46     ACALL    INIT_TIMER         ;设置定时器
47
48 ;加载显示器初值数据
49     MOV      A, #0C0H
50     MOV      P0, A
51
52 ;无穷循环
53 LOOP:
54     ACALL    CONV
55     ACALL    TIME_PRO           ;检查闹铃时间
56     ACALL    DISP               ;扫描显示
57
58 ;设置走时时间
59     JB      K1, M1              ;未按下 K1 键，则继续扫描
60     ACALL    LED_BL             ;LED 闪动
61     ACALL    SET_TIME           ;设置目前时间
62     JMP      LOOP               ;跳转到 LOOP 处运行
63
64 ;设置闹铃时间
65 M1:    JB      K2, M2              ;未按下 K2 键，则继续扫描
66     ACALL    LED_BL             ;LED 闪动
67     ACALL    SET_ATIME          ;设置闹铃时间

```

```

68      JMP      LOOP          ;跳转到 LOOP 处运行
69
70 ;设置倒计时间
71 M2:   JB       K3, M3        ;未按下 K3 键，则继续扫描
72 ; .....
73      ACALL    LED_BL
74      ACALL    DOWN_ATIME   ;设置倒计时间
75 ; .....
76      JMP      LOOP          ;跳转到 LOOP 处运行
77
78 M3:   JB       K4, M4        ;未按下 K4 键，则继续扫描
79     CPL     ALB
80     JNB     ALB, M31
81
82 ;闹铃启动连续 3 次发出响声
83     ACALL    BZ
84     ACALL    BZ
85     ACALL    BZ
86     JMP      LOOP          ;跳转到 LOOP 处运行
87
88 M31:
89     ACALL    BZ          ;闹铃停止，发出一声响
90     JMP      LOOP          ;跳转到 LOOP 处运行
91
92 M4:
93     JMP      LOOP          ;跳转到 LOOP 处运行
94
95 ;----- 实现走时功能的子程序 -----
96
97 ;使用定时器 T0 模式 0 计时
98 INIT_TIMER:                   ;初始化定时器，使用定时器 T0 模式 1 计时
99     MOV      TMOD, #00000000B ;设置定时器 T0 工作模式 0
100    MOV      IE, #10000010B  ;启用定时器 T0 中断产生
101    MOV      TL0, #(8192-4900)MOD 32 ;加载初始化数据
102    MOV      TH0, #(8192-4900)/32
103    SETB    TR0          ;启动定时器 0 开始计时
104    RET
105 ;-----
106 ;中断服务程序
107 TO_SRV:                     ;定时器 T0 计时中断程序每隔 5ms 中断一次
108    PUSH    ACC          ;将 A 值压入堆栈
109    MOV      TL0, #(8192-4900)MOD 32 ;重加载初始化数据
110    MOV      TH0, #(8192-4900)/32

```

```

111     INC      DEDA          ;加 1
112     ;秒输出
113     MOV      A, DEDA
114     CJNE    A, #200, TT1   ;是否 1s 到了
115     MOV      DEDA, #0      ;计数值清 0
116     CPL      WLED         ;LED 灯亮灭变换
117     INC      SEC           ;秒计数加 1
118     MOV      A, SEC
119     CJNE    A, #60, TT1   ;是否 1min 到了
120     ;分输出
121     INC      MIN           ;分计数加 1
122     MOV      SEC, #0       ;秒计数清 0
123     MOV      A, MIN
124     CJNE    A, #60, TT1   ;是否 1h 到了
125     ;小时输出
126     INC      HOUR          ;小时计数加 1
127     MOV      MIN, #0       ;分计数清 0
128     MOV      A, HOUR
129     CJNE    A, #24, TT1   ;是否 24h 到了
130     MOV      SEC, #0       ;秒钟变量清 0
131     MOV      MIN, #0       ;分钟变量清 0
132     MOV      HOUR, #0      ;小时变量清 0
133     TT1:
134     POP     ACC           ;将 A 值由堆栈取出
135     RETI
136
137 ;----- 实现显示功能的子程序 -----
138
139
140 CONV:
141 ;转换小时数据
142     MOV      A, HOUR
143     MOV      B, #10
144     DIV      AB
145     MOV      DPTR, #TABLE  ;查表转换
146     MOVC   A, @A+DPTR
147     MOV      BUF, A
148     MOV      A, B
149     MOVC   A, @A+DPTR
150     MOV      BUF+1, A
151     ;转换分钟数据
152     MOV      A, MIN
153     MOV      B, #10

```

```

154      DIV      AB
155      MOV      DPTR, #TABLE      ;查表转换
156      MOVC     A, @A+DPTR
157      MOV      BUF+2, A
158      MOV      A, B
159      MOVC     A, @A+DPTR
160      MOV      BUF+3, A
161      RET
162
163 ;字型数据编码表
164 TABLE:
165      DB      0C0H,0F9H,0A4H,0B0H
166      DB      99H,92H,82H,0F8H
167      DB      80H,90H,88H,83H
168      DB      0C6H,0A1H,86H,8EH
169
170 ; -----
171 ;扫描显示
172 DISP:
173      MOV      R0, #BUF      ;指向显示器显示缓冲区起始地址
174      MOV      R2, #4        ;循环执行 4 次
175      MOV      ACC, #11110111b ;加载扫描信号初值 11110111B
176 S1:
177      PUSH     ACC
178      MOV      A, @R0      ;取出显示器数据
179      MOV      P0, A        ;由 P0 送出一位显示器数据
180      POP      ACC
181      MOV      P2, ACC      ;由 P2 送出扫描输出信号
182      MOV      R5, #2        ;延迟一段时间
183      ACALL    DELAY
184 ;改变扫描码 EX:XXXX1011
185      RR      A          ;累加器 A 向右移动一位
186      INC     R0          ;显示器显示缓冲区地址加 1
187      DJNZ    R2, S1        ;判断是否继续执行循环
188      RET
189
190 ; ----- 实现调整时间子程序 -----
191
192 ;设置目前时间
193 SET_TIME:
194      CLR     TR0        ;定时器工作暂停
195      MOV     SEC, #0        ;秒钟变量清 0
196 L0:

```

```

197     ACALL    DISP          ;扫描显示
198
199     JB       K2, L1        ;未按下 K2 键，则继续扫描
200     JNB      K2, $         ;按下，则等待放开
201     ;K2 小时调整
202     INC      HOUR         ;小时计数加 1
203     MOV      A, HOUR       ;将小时值赋给显示寄存器
204     CJNE    A, #24, L11    ;是否 24h 到了
205     MOV      HOUR, #0       ;小时变量清 0
206 L11:
207     ACALL    CONV         ;转换显示数据
208     ACALL    DISP         ;扫描显示
209     JMP      L0           ;跳转到 L0 处执行
210
211 L1:   JB       K3, L2        ;未按下 K3 键，则继续扫描
212     JNB      K3, $         ;按下，则等待放开
213     ;K3 分钟调整
214     INC      MIN          ;分钟计数加 1
215     MOV      A, MIN        ;将分钟值赋给显示寄存器
216     CJNE    A, #60, L21    ;是否 60min 到了
217     MOV      MIN, #0       ;分钟变量清 0
218 L21:
219     ACALL    CONV         ;转换显示数据
220     ACALL    DISP         ;扫描显示
221     JMP      L0           ;跳转到 L0 处运行
222
223 L2:   JB       K4, L0        ;未按下 K4 键，则继续扫描
224     JNB      K4, $         ;按下，则等待放开
225     ACALL    BZ           ;蜂鸣器鸣响一声
226     ;K4 设置完成
227     ACALL    LED_BL       ;LED 闪动
228     SETB     TR0          ;启动定时器
229     RET
230
231 ;----- 喇叭和指示灯等其他子程序 -----
232
233 ;计时单元清 0
234 INIT:
235     MOV      DEDA, #0       ;5ms 计数值清 0
236     MOV      SEC, #0        ;秒钟变量清 0
237     MOV      MIN, #0        ;分钟变量清 0
238     MOV      HOUR, #0       ;小时变量清 0
239     RET

```

```

240 ;
241 ;蜂鸣器
242 BZ:                                ;蜂鸣器鸣响一声
243     MOV      R6, #250
244 B1:    ACALL   DEX
245     CPL      SPK
246     DJNZ    R6, B1
247     MOV      R5, #50 ;#10
248     ACALL   DELAY
249     RET
250
251 ; - - - - -
252 ;工作指示
253 LED_BL:                            ;工作指示灯闪烁
254     MOV      R4, #6
255 LE1:    CPL      WLED            ;位反向
256     MOV      R5,#25 ;#5
257     ACALL   DELAY
258     DJNZ    R4, LE1           ;延迟 50ms
259     RET
260 ; - - - - -
261 ;延时子程序
262 DELAY:                            ;总延迟时间 R5×2ms
263     MOV      R6,#10 ;#50
264 D1:    MOV      R7,#100
265     DJNZ    R7,$
266     DJNZ    R6, D1
267     DJNZ    R5, DELAY
268     RET
269 ; - - - - -
270 ; 蜂鸣器短暂延时子程序
271 DEX:    MOV      R7, #180        ;短暂延时
272 DE1:    NOP
273     DJNZ    R7, DE1
274     RET
275
276 ;***** 增加定时闹铃程序部分 *****
277
278 ; - - - - - 设置闹铃时间 - - - - -
279 ;设置闹铃时间
280 SET_ATIME:
281     ACALL   CONVA            ;加载闹铃时间数据
282 NO:

```

```

283      ACALL    DISP
284 ; - - - - -
285      JB       K2, N1          ;未按下 K2 键，则继续扫描
286      JNB      K2, $           ;按下，则等待放开
287 ;K2 调整小时
288      INC      HH              ;小时数加 1
289      MOV      A, HH
290      CJNE    A, #24, N11     ;是否 24h 到了
291      MOV      HH, #0          ;小时数清 0
292 N11:
293      ACALL    CONVA         ;加载闹铃时间数据
294      ACALL    DISP          ;扫描显示
295      JMP      N0              ;跳转到 N0 处运行
296 ; - - - - -
297 N1:   JB       K3, N2          ;未按下 K3 键，则继续扫描
298      JNB      K3, $           ;按下，则等待放开
299 ;K3 调整分钟
300      INC      MM              ;调整分钟数
301      MOV      A, MM
302      CJNE    A, #60, N21     ;是否 60min 到了
303      MOV      MM, #0          ;分钟数清 0
304 N21:
305      ACALL    CONVA         ;加载闹铃时间数据
306      ACALL    DISP          ;扫描显示
307      JMP      N0              ;跳转到 N0 处运行
308 ; - - - - -
309 N2:   JB       K4, N0          ;未按下 K4 键，则继续扫描
310      JNB      K4, $           ;等 K4 键放开
311 ;K4 设置完成
312      ACALL    LED_BL        ;LED 闪动
313      ACALL    CONV          ;加载现在时间数据
314      RET
315
316 ; - - - - - 加载闹铃时间数据 - - - - -
317
318
319 CONVA:
320 ;转换小时数据
321      MOV      A, HH
322      MOV      B, #10
323      DIV      AB
324 ; - - - - -
325      MOV      DPTR, #TABLE   ;查表转换

```

```
326     MOVC      A, @A+DPTR
327     MOV       BUF, A
328 ; -----
329     MOV       A, B
330     MOVC      A, @A+DPTR
331     MOV       BUF+1, A
332 ;转换分钟数据
333     MOV       A, MM
334     MOV       B, #10
335     DIV       AB
336 ; -----
337     MOV       DPTR, #TABLE    ;查表转换
338     MOVC      A, @A+DPTR
339     MOV       BUF+2, A
340 ; -----
341     MOV       A, B
342     MOVC      A, @A+DPTR
343     MOV       BUF+3, A
344     RET
345
346 ; ----- 检查闹铃时间处理 -----
347
348 ;检查是否启动闹铃
349 TIME_PRO:
350     JB       ALB, TI3
351     RET
352 ; -----
353
354 ;检查闹铃时间是否到了
355 TI3:
356     MOV       A, HOUR
357     MOV       B, HH
358     CJNE      A, B, BK        ;检查闹铃小时时间
359 ; -----
360     MOV       A, MIN
361     MOV       B, MM
362     CJNE      A, B, BK        ;检查闹铃分钟时间
363 ; -----
364     ACALL    TIME_OUT        ;调用执行检查闹铃时间处理子程序
365 BK:
366     RET
367
368 ; ----- 执行闹铃时间处理 -----
```

```

369
370 TIME_OUT:                                ;所设置的闹铃数据到了
371 X1: ACALL LED_BL                      ;LED 闪动
372          ACALL BZ                         ;蜂鸣器发声
373 ;按下 K4 键停止闹铃
374     JB      K4, X1                     ;等待是否按下 K4 键
375     JNB     K4,$                      ;若按下键，则等待按键放开
376     ACALL LED_BL                      ;LED 闪动
377     CLR     ALB                        ;清除闹铃位
378     RET
379; -----
380
381 ;***** 增加倒计时程序部分 *****
382
383 ;----- 设置倒计时时间 -----
384 ;设置倒计时时间
385 DOWN_ATIME:
386     CLR     TR0                        ;定时器工作暂停
387     MOV     SS_D, #1                  ;秒数变量为 1
388 S0:
389     ACALL DISP                       ;扫描显示
390 ;-----
391 ;按 K2 分钟数增加
392     JB      K2, S12                  ;未按下 K2，则继续扫描
393     JNB     K2,$                   ;按下，则等待放开
394
395     INC     MM_D                     ;分钟数加 1
396     MOV     A, MM_D
397     CJNE   A, #60, S11              ;是否 60min 到了
398     MOV     MM_D, #0                ;分钟变量清 0
399 S11:
400     ACALL DOWN_CONV                 ;显示倒数的时间
401     ACALL DISP                     ;扫描显示
402     JMP     S0                      ;跳转到 S0 处运行
403 ;-----
404 ;按 K3 分钟数减少
405 S12: JB      K3, S2                  ;未按下 K3，则继续扫描
406     JNB     K3,$                   ;按下，则等待放开
407
408     DEC     MM_D                     ;分钟数减少 1
409     MOV     A, MM_D
410     CJNE   A, #0,S21              ;判断是否为 0
411     MOV     MM_D, #60              ;分钟数赋值为 60

```

```

412 S21:
413     ACALL    DOWN_CONV      ;加载倒数时间数据
414     ACALL    DISP          ;扫描显示
415     JMP     S0            ;循环执行
416 ; -----
417 S2:   JB      K4, S0        ;未按下 K4 键，则继续扫描
418     JNB     K4, $          ;等 K4 键放开
419 ;K4 设置完成
420     ACALL    BZ            ;响 4 声
421     ACALL    BZ
422     ACALL    BZ
423     ACALL    BZ
424
425     ACALL    LED_BL        ;LED 闪动
426     SETB     TR0           ;启动定时器
427
428 S3:
429     ACALL    DOWN_TIME     ;检查倒计时
430     ACALL    DOWN_CONV      ;加载倒数时间数据
431     ACALL    DISP          ;扫描显示
432     JMP     S3            ;返回
433     RET
434
435 ; ----- 加载倒计时时间 ----- -
436 ;显示倒数的分钟数
437 DOWN_CONV:
438     MOV     A, MM_D
439     MOV     B, #10          ;设置被除数
440     DIV     AB            ;除法运算，结果 A 存商数，B 存余数
441     MOV     DPTR, #TABLE    ;查表转换
442     MOVC    A, @A+DPTR
443     MOV     BUF+2, A        ;商数存入 BUF+2
444     MOV     A, B            ;余数暂存入 A
445     MOVC    A, @A+DPTR
446     MOV     BUF+3, A        ;余数存入 BUF+3
447     RET
448
449 ; ----- 检查倒计时 ----- -
450 ;检查倒计时
451 DOWN_TIME:                   ;检查是否倒数终止
452     MOV     A, SEC          ;加载现在秒数
453     MOV     B, SEC0         ;载入旧秒数
454     CJNE    A, B, D01       ;是否过了 1s

```

```

455      RET
456 ; -----
457 D01:
458      MOV      SEC0, SEC      ;记录旧的秒数
459      DEC      SS_D       ;秒数减 1
460      MOV      A, SS_D
461      CJNE    A, #0, D11    ;秒钟数是否为 0
462      MOV      A, MM_D
463      CJNE    A, #0, D12    ;分钟数是否为 0
464 ;秒钟数为 0 且分钟数为 0，则倒数终止
465      ACALL   TIME_OUT    ;倒数终了处理程序
466      JMP     LOOP
467      RET
468 ; -----
469 D12:
470      MOV      SS_D, #59    ;秒钟数赋值为 59
471      DEC      MM_D       ;分钟数减 1
472
473 D11:
474      ACALL   DOWN_CONV    ;显示倒计时时间
475      RET
476 ; -----
477      END

```

3. 程序说明

下面仅就增加的倒计时程序部分加以说明。

在程序初始化部分里，第 18~20 行语句对倒计时程序中引用的 3 个新变量进行了定义。其中，SEC0 存放旧的秒数；MM_D 存放倒数时间分钟数；SS_D 存放倒数时间秒钟数。

在主程序里，将 K3 设计为倒计时功能键，当按下 K3 键时便进入倒计时设置状态，主程序里第 74 行语句是 K3 键功能控制语句，该语句调用了 DOWN_ATIME 设置倒数时间子程序。

进入设置倒计时状态后，K3 键和 K2 键便执行第二功能，这时按 K2 键使倒计时分钟数增加；按 K3 键使倒计时分钟数减少。

设置倒计时时间完成后，按 K4 键，听到鸣响 4 声，表明进入倒计时工作状态。此时，程序循环运行第 428~432 行语句。

在该循环中，一方面调用 DOWN_CONV 加载倒计时时间子程序和 DISP 显示器扫描程序，使之显示倒计时时间；另一方面调用 DOWN_TIME 检查倒计时子程序，检查看是否倒数终止。

当检查到秒钟数为 0 而且分钟数也为 0 时，表明倒计时终了，执行闹铃处理程序，开始闹铃。这时按 K4 键闹铃停止，同时程序进入主程序循环，恢复时钟正常走时功能。

23.4 简单时钟的 C 语言程序设计

使用 C 语言编写单片机时钟程序，比使用汇编语言简便。

23.4.1 时钟结构和使用方法

1. 时钟结构

时钟硬件结构与前边 23.1 节介绍的基本相同，主要由单片机、LED 数码管和按键开关所组成，稍有区别的地方是使用了数码管 U2 的小数点为秒点，每秒闪烁一次。

2. 使用方法

接通电源后，数码管闪烁显示“0000”，数码管 U2 的小数点每秒闪烁一次，表示时钟开始工作。

接着需要设置现在时间：K1 键为设置功能键，按一下 K1 键，U2 小数点停止闪动，表明进入了调时状态，此时，按 K2 键为小时调整，每按 0.4s 递加 1；按 K3 键为分钟调整，每按 0.4s 递加 1。设置完成后再按一下 K1 键，U2 小数点恢复闪动，表明设置完成，时钟进入了正常走时状态。

3. 软件设计原理

程序设计使用 C51 语言与使用汇编语言的思路是一致的，首先，把时钟功能主要分解为走时、显示和调时 3 个主要部分，并将每一部分的功能编写成相应的函数，即子程序，然后，再通过主函数调取各函数，把各部分有机地连在一起，完成单片机的时钟设计。

23.4.2 走时功能的设计

走时功能部分包含 3 个子程序，即定时器设置函数 void init_timer()、中断服务函数 void T0_srv(void) interrupt 1 和转换时、分、秒走时单元函数 void conv()。

1. 定时器函数 void init_timer()

定时器函数 void init_timer() 的作用是每隔 5ms 产生一次中断信号，它是时钟标准时间的来源和保证。

在该函数中，选择使用了单片机内的定时器 T0，并设置在模式 1 工作状态下每隔 5ms 中断一次，在晶振频率为 12MHZ 时，此 5ms 的初值为 5000（实际上程序还要作其他运算，使得时间偏长，因此，此值需经实验作些调整），其程序代码如下：

```
void init_timer()
{
    TMOD=0x01; /* 设置定时器 T0 工作模式 1 */
```

```

TH0=-(4800/256);           /* 加载高字节计数初值 */
TL0=-(4800%256);          /* 加载低字节计数初值 */
IE=0x82;                   /* 启用定时器 T0 中断产生 */
TR0=1;                     /* 启动定时器 T0 开始计时 */
}

```

2. 中断服务函数 void T0_srv(void) interrupt 1

中断服务函数 void T0_srv(void) interrupt 1 的作用如下。

- (1) 重置定时器 T0 计时初始值。
- (2) 记录中断次数，每中断一次，计数单元 deda 的值递增 1。

中断服务函数如下：

```

void T0_srv(void) interrupt 1
{
    TH0=-(4800/256);           /* 重置定时器计时初始值 */
    TL0=-(4800%256);
    deda++;                   /* 计数单元 deda 值递增 */
}

```

3. 转换时、分、秒走时单元函数 void conv()

转换时、分、秒走时单元函数 void conv() 的主要作用为：根据计数单元 deda 的值，中断 200 次转换为 1 秒，60 秒转换为 1 分，60 分转换为 1 小时，24 小时转换为 1 天，其函数 void conv() 如下：

```

1 void conv()
2 {
3     if(deda<=100)d_05s=0;      /* 秒位标志，每秒的后 0.5s 置 0 */
4     else d_05s=1;              /* 秒位标志，每秒的前 0.5s 置 1 */
5     if(deda>=200){sec++;deda=0;} /* 中断 200 次后，秒加 1，deda 清 0 */
6     if(sec==60){min++;sec=0;}   /* 秒满 60 次后，分加 1，秒清 0 */
7     if(min==60){hour++;min=0;}  /* 分满 60 次后，时加 1，分清 0 */
8     if(hour==24){hour=0;}       /* 小时满 24 后，小时清 0 */
9 }

```

23.4.3 显示功能的设计

时钟的时间通过 4 位 LED 数码管来显示，采用动态扫描方法，U1 数码管显示小时的十位数，U2 显示小时的个位数；U3 显示分钟的十位数，U4 显示分钟的个位数。

显示函数的功能为：将小时、分钟时间数据转换成适合 LED 数码管显示的数据，即进行十进制计时处理；对数码管进行扫描，使之动态显示。走时时间显示函数 void disp() 程序代码如下：

```

1 void disp()
2 {

```

```

3   P0=DATA_7SEG[hour/10];P2=0xf7;delay(1);
4   P0=DATA_7SEG[hour%10];P2=0xfb;delay(1);
5   if(d_05s==1){if(P2_2==0)P0_7=0;else P0_7=1;}
6   delay(1);
7   P0=DATA_7SEG[min/10];P2=0xfd;delay(1);
8   P0=DATA_7SEG[min%10];P2=0xfe;delay(1);
9 }

```

第3行语句对小时数据进行十进位取整处理 (hour/10) 后，显示的数据为十位上的小时数，此数据虽然同时输入4位数码管，但是哪位数码管导通显示是由选通信号P2所控制的，该扫描信号低4位数值对应控制着4位数码管是否导通，0对应的数码管为导通，1对应的数码管为截止。

此时0正好对应U1数码管，即显示十位数的数码管U1导通显示，并调延时子函数delay(1)延迟一段时间，以便能看清楚。

第4行语句显示的是小时数据的个位数，此时，选通信号为P2=0xfb=11111011B，可以看出扫描码中的0向右移一位，使所对应的U2数码管导通，显示小时数据的个位数。

序号7~8语句与序号3~4语句工作原理相同，分别显示分钟的十位数和个位数。

第5~6行语句作用是使秒点闪烁。当秒位标志d_05s==1时（占0.5s时间），如果此时U2数码管导通，则U2数码管的小数点亮；否则小数点灭。这样使秒点每秒钟闪烁1次。

第7、8行语句与第3行、4行语句工作原理相同，分别显示分钟的十位数和个位数。

23.4.4 调整时间功能的设计

void set_time()为调整走时时间函数，其作用是调整时钟时间。当单片机时钟每次重新启用时，都需要重新设置目前时钟的时间。首先按K1功能键使U2小数点停止闪动，表明进入了调时状态，此时，接在P2_5端口的K2键为小时调整键；接在P2_6端口的K3键为分钟调整键，调整走时时间函数如下。

```

1 void set_time()
2 {uchar m;
3  if(P2_5==0)delay(1);           /* 按下K2键，消除抖动 */
4  if(P2_5==0)hour++;            /* 小时数递增 */
5  if(hour==24)hour=0;           /* 小时数到24，从0开始 */
6  for(m=0;m<30;m++)           /* 循环30次 */
7  {
8    disp();                     /* 调用disp()显示函数 */
9    if(P2_2==0)P0_7=0;          /* 点亮U2小数点（秒点） */
10   else P0_7=1;
11   delay(1);                 /* 调用delay(1)延时函数 */
12 }
13 if(P2_6==0)delay(1);           /* 按下K3键时，消除抖动 */
14 if(P2_6==0)min++;             /* 分钟数递增 */

```

```

15  if(min==60)min=0;          /* 分钟数到 60 , 从 0 开始 */
16  for(m=0;m<30;m++)        /* 循环 30 次 */
17  {
18      disp();                /* 调用 disp() 显示子函数 */
19      if(P2_2==0)P0_7=0;      /* 点亮 U2 小数点 ( 秒点 ) */
20      else P0_7=1;
21      delay(1);             /* 调用 delay(1) 延时函数 */
22  }
23 }
```

第 3 ~ 5 行语句为小时调整。

第 13 ~ 15 行语句为分钟调整。

第 6 ~ 12 行与第 16 ~ 21 行语句结构功能相同，如下所示。

- 调 disp() 显示子函数，显示调整时间。
- 点亮秒点。
- 按 K2 键或按 K3 时，每 0.4s 递加 1。

23.4.5 按键扫描等其他函数

除了上面讲到的主要功能函数外，还有按键扫描及延时等其他函数，具体包括调用程序函数、按键扫描函数和延时函数。

1. 调用程序函数 void time()

该函数的作用是调用走时单元转换函数 conv() 和调用走时时间显示函数 disp()，其调用程序函数如下：

```

void time()
{
    conv();    /* 走时单元转换函数 */
    disp();    /* 走时时间显示函数 */
}
```

2. 扫描按键函数 void scan_key()

扫描按键函数的作用是记载连接在 P2_4 端口的 K1 键的按键次数。因为 K1 键定义为功能设置键，不同按键次数将在主函数中代表不同功能，其扫描按键函数如下：

```

1 void scan_key()
2 {
3     delay(1);                  /* 调用延时函数 */
4     if(P2_4==0)set++;          /* 按一下 K1 键 , set 加 1 */
5     if(set>=2)set=0;          /* 按两次 K1 键 , set 为 0 */
6     if(set==1)flag=0x55;       /* set =1 , flag 等于 55H */
7     F0: if(P2_4==0)goto F0;   /* 按键未释放 , 在此等候 */
8 }
```

第3行语句是调用延时函数。

第4行语句中的P2_4端口接有K1按键开关，当K1按键按下时，P2_4为0，此时，功能设定标志set的值会增加1。

第5行语句是定义set值的有效范围：当K1键不按时，set为0；K1键按一次时，set为1；再按时，set又为0，也就是说set值的有效范围是0~1。

第6行语句的作用是重新设定时间后，使掉电标志flag内写入标志数55H。

第7行语句的作用是在按键未释放时，在此循环等候。

3. 延时函数 void delay(uint k)

这是一个双层循环的延时函数，内循环延时1ms，外循环次数决定变量k的值，总延时时间为：1ms×k。例如，k=1时，延时时间为1ms；k=400时，延时时间为400ms。

```

1 void delay(uint k)
2 {
3     uint i,j; /* 定义局部变量 i、j */
4     for(i=0;i<k;i++){ /* 外层循环 */
5         for(j=0;j<121;j++) /* 内层循环 */
6             {};
7 }
```

23.4.6 时钟主函数

时钟主函数通过调用各函数（子程序）将各部分功能有机地组合到一起，完成单片机时钟的总体设计要求，时钟主函数如下：

```

1 void main()
2 {
3     init_timer(); /* 定时器T0初始化 */
4     while(1) /* 无限循环 */
5     {
6         if(P2_4==0)scan_key(); /* 若有按键，调用按键扫描函数 */
7         switch(set) /* 根据set键值散转 */
8         {
9             case 0:time(); break; /* 走时时间程序 */
10            case 1:set_time();break; /* 走时时间调整 */
11            default:break; /* 其他，退出 */
12        }
13        if(flag!=0x55) /* 判断掉电标志 */
14        {
15            P0=0xc0; P2=0xc0; delay(100);/* 点亮4个数码管 */
16            P2=0xff; delay(400); /* 熄灭4个数码管 */
17        }
18    }
19 }
```

第3行语句调用定时器设置函数 init_timer()，使定时器初始化。

第4行语句进入无限循环。

第6行语句检查是否按K1键，如果按K1键，则调用按键扫描子函数。

第7~11行语句根据功能设定标志 set 值，调用相应的函数。例如，set=0，调用走时函数 time()，使时钟进入走时工作状态；set=1，调用走时时间调整函数 set_time()，使时钟进入走时时间调整状态。set 值只能从0到1，然后又回到0。

第13行语句判断掉电标志 flag 是否等于55H。若不等，说明是刚开机还没进行校对时间，或是运行过程中掉过电，此时程序将运行第15~16行语句；若相等，说明工作正常，程序将又回到初始化之后进行循环运行。

第15~16行语句使4个数码管显示0并闪烁，提示时间不准。

23.4.7 简单时钟C语言程序清单

在C语言程序中，无论主函数放在什么位置，程序运行都是从主函数开始，通过主函数调用其他各函数来完成时钟功能。所以分析整个时钟程序时，应该首先分析主函数的工作过程，使之对整个程序的结构有个初步了解，然后再详细研究相应各函数的工作原理。

1. 程序标号

- main()：主函数。
- init_timer()：定时器设置函数。
- T0_srv(void) interrupt 1：中断服务函数。
- conv()：转换时、分、秒及走时单元函数。
- disp()：扫描显示函数。
- set_time()：设置走时时间函数。
- time()：走时时间程序函数。
- scan_key()：扫描按键函数。
- delay(uint k)：延时函数。

2. 程序清单

最前边是头文件及变量和函数的全局声明部分；接着是走时函数、显示函数、调整时间函数和按键扫描及延时等函数；最后是主函数，其程序清单如下：

```
/*
 * 文件名称：SZ04.c
 * 程序功能：单片机时钟
 */
1 #include <AT89X51.H>          /* 包含器件配置文件 */
2 #define uchar unsigned char
3 #define uint unsigned int
4 char DATA_7SEG[10]={0xC0,0xF9,0xA4,0xB0,0x99,
5     0x92,0x82,0xF8,0x80,0x90,};    /* 0~9 的数码管段码 */
```

```

6     uchar hour=0,min=0,sec=0;           /* 时、分、秒单元清 0 */
7     uchar deda=0;                      /* 5ms 计数单元清 0 */
8     bit d_05s=0;                      /* 0.5s 标志 */
9     uchar set=0;                      /* 模式设定标志 */
10    uchar m=0;
11    uchar flag=0;                     /* RAM 掉电标志 */
12    void delay(uint k);              /* 延时函数 */
13    void conv();                     /* 走时单元转换 */
14    void disp();                     /* 走时时间显示函数 */
15
16 /* ----- 走时函数部分 ----- */
17
18 /* 定时器 T0 初始化 */
19 void init_timer()
20 {
21     TMOD=0x01;                      /* 设置定时器 T0 工作模式为 1 */
22     TH0=-(4800/256);                /* 加载高字节计数初值 */
23     TL0=-(4800%256);                /* 加载低字节计数初值 */
24     IE=0x82;                        /* 启用定时器 T0 中断产生 */
25     TR0=1;                          /* 启动定时器 T0 开始计时 */
26 }
27
28 /*-----*/
29 /*5ms 定时中断服务函数*/
30 void T0_srv(void) interrupt 1
31 {
32     TH0=-(4800/256);                /* 重置定时器计时初始值 */
33     TL0=-(4800%256);
34     deda++;                         /* 计数单元 deda 值递增 */
35 }
36 /*-----*/
37
38 /*时、分、秒单元及走时单元转换*/
39 void conv()
40 {
41     if(deda<=100)d_05s=0;           /* 秒位标志，每秒的后 0.5s 置 0 */
42     else d_05s=1;                  /* 秒位标志，每秒的前 0.5s 置 1 */
43     if(deda>=200){sec++;deda=0;}   /* 中断 200 次秒加 1，deda 清 0 */
44     if(sec==60){min++;sec=0;}       /* 秒满 60 次后，分加 1，秒清 0 */
45     if(min==60){hour++;min=0;}      /* 分满 60 次后，时加 1，分清 0 */
46     if(hour==24){hour=0;}           /* 小时满 24 后，小时清 0 */
47 }
48

```

```

49 /* - - - - - 显示函数部分 - - - - - */
50
51 /*走时时间显示函数*/
52 void disp()
53 {
54     P0=DATA_7SEG[hour/10];P2=0xf7;delay(1);
55     P0=DATA_7SEG[hour%10];P2=0xfb;delay(1);
56     if(d_05s==1){if(P2_2==0)P0_7=0;else P0_7=1;}
57     delay(1);
58     P0=DATA_7SEG[min/10];P2=0xfd;delay(1);
59     P0=DATA_7SEG[min%10];P2=0xfe;delay(1);
60 }
61
62 /* - - - - - 调整时间函数部分 - - - - - */
63
64 /* 调整走时时间 */
65 void set_time()
66 {uchar m;
67     if(P2_5==0)delay(1);                      /* 按下 K2 键，消除抖动 */
68     if(P2_5==0)hour++;                         /* 小时数递增 */
69     if(hour==24)hour=0;                        /* 小时数到 24，从 0 开始 */
70     for(m=0;m<30;m++)                         /* 循环 30 次 */
71     {
72         disp();                                /* 调用 disp() 显示函数 */
73         if(P2_2==0)P0_7=0;                      /* 点亮 U2 小数点（秒点） */
74         else P0_7=1;
75         delay(1);                            /* 调用 delay(1) 延时函数 */
76     }
77     if(P2_6==0)delay(1);                      /* 按下 K3 键，消除抖动 */
78     if(P2_6==0)min++;                         /* 分钟数递增 */
79     if(min==60)min=0;                        /* 分钟数到 60，从 0 开始 */
80     for(m=0;m<30;m++)                         /* 循环 30 次 */
81     {
82         disp();                                /* 调取 disp() 显示函数 */
83         if(P2_2==0)P0_7=0;                      /* 点亮 U2 小数点（秒点） */
84         else P0_7=1;
85         delay(1);                            /* 调取 delay(1) 延时函数 */
86     }
87 }
88
89 /* - - - - - 按键扫描及延时等函数部分 - - - - - */
90
91 /*走时时间程序函数*/

```

```

92 void time()
93 {
94     conv();                                /*走时单元转换*/
95     disp();                                /*走时时间显示函数*/
96 }
97
98 /*-----*/
99
100 /* 扫描按键函数 */
101 void scan_key()
102 {
103     delay(1);                            /* 调用延时函数 */
104     if(P2_4==0)set++;
105     if(set>=2)set=0;
106     if(set==1)flag=0x55;
107     F0:if(P2_4==0)goto F0;
108 }
109 /*-----*/
110
111 /*延时函数*/
112 void delay(uint k)                      /* 总延时时间：1ms×k */
113 {
114     uint i,j;                            /* 定义局部变量 i、j */
115     for(i=0;i<k;i++){
116         for(j=0;j<121;j++)
117             {;}
118     }
119
120 /*----- 主函数 -----*/
121
122 /*主函数*/
123 void main()
124 {
125     init_timer();                         /* 定时器 T0 初始化 */
126     while(1)                            /* 无限循环 */
127     {
128         if(P2_4==0)scan_key();           /* 有按键，调用按键扫描函数 */
129         switch(set)                  /* 根据 set 键值散转 */
130         {
131             case 0:time(); break;       /* 走时时间程序 */
132             case 1:set_time();break;   /* 走时时间调整 */
133             default:break;           /* 其他，退出 */
134         }

```

```

135     if(flag!=0x55)          /* 判断掉电标志 */
136     {
137         P0=0xc0; P2=0xc0; delay(100); /* 点亮 4 个数码管 */
138         P2=0xff; delay(400);        /* 熄灭 4 个数码管 */
139     }
140 }
141 }
```

3. 程序说明

第 1 ~ 14 行语句为头文件及变量和函数的全局声明部分。

第 18 ~ 47 行语句为实现走时功能的函数部分，启动定时 T0 并选择模式 1，在晶振频率为 12MHz 时产生 5ms 中断的初值应为 5000，但实际上由于程序还要作其他运算，使得时间偏长，需经实验进行些调整；中断服务函数主要是将产生的中断次数存入 deda 记时单元；走时单元转换函数是将定时器产生的秒、分、时。

第 51 ~ 60 行语句为显示函数部分。将小时、分钟时间数据进行十进制计时处理，以便适应 LED 数码管显示时间；对数码管进行扫描，使之动态显示。

第 65 ~ 87 行语句为调整时间函数部分，其作用是调整时钟时间。当单片机时钟每次重新启用时，都需要重新设置目前时钟的时间。

第 92 ~ 118 行语句为按键扫描及延时等函数部分，包括走时时间程序函数、扫描按键函数和延时函数。

第 123 ~ 144 行语句为主函数。在 C 语言程序中，主函数所在位置灵活，可以放在函数的前边、后边或中间，程序运行时会自动找到主函数，并从主函数开始运行。

当时钟启动后，在主函数里首先调用定时器设置函数，使定时器开始工作。接着进入无限循环，在循环中主要检查按 K1 功能设置键的次数，并根据次数选相应功能。例如，按 1 次 K1 键，进入时钟调整状态，此时可通过按 K2 键调整小时和按 K3 键调整分钟，设置好后再按 1 次 K1 键便进入正常走时工作状态。



第 24 章 液晶显示应用实例

在智能化电子产品中，一般会使用液晶显示器显示信息，本章介绍的液晶显示秒表和液晶显示温度控制器是液晶显示的具体应用实例。

24.1 液晶显示秒表

24.1.1 硬件设计

液晶显示秒表的硬件结构非常简单，在单片机的 P1 端口上接有 LCD 模块用来显示计时时间，P3.3、P3.4 引脚分别接按钮开关 K1、K2 作为秒表的计时控制按钮，其电路如图 24.1 所示。

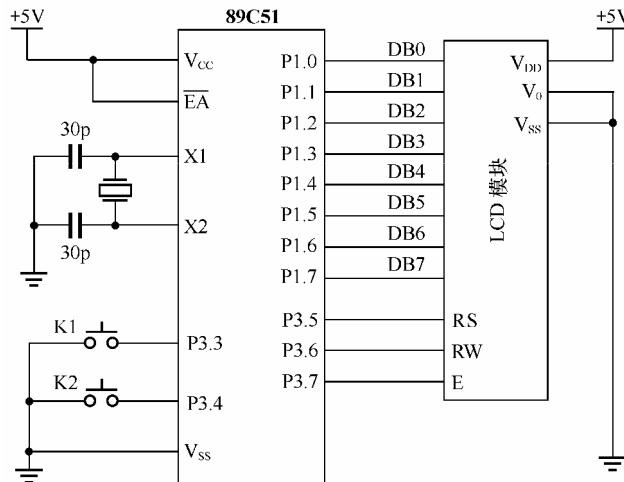


图 24.1 液晶秒表电路

24.1.2 程序设计

由定时器 T0 产生标准计时时间，并由按钮开关输入信号控制计时、暂停、累加计时和

清 0，计时的时间由 LCD 显示。

1. 流程图

程序设计主程序流程如图 24.2 所示。

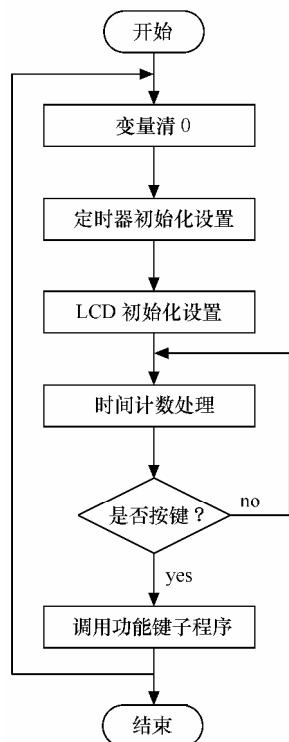


图 24.2 主程序流程

2. 程序

汇编语言编写的液晶显示秒表源程序 LCD2401.ASM 代码如下：

```

01 ; 定义计时单元地址
02 HOUR EQU 30H ; 存放小时变量
03 MIN EQU 31H ; 存放分钟变量
04 SEC EQU 32H ; 存放秒钟变量
05 DEDA EQU 33H ; 存放 10ms 计数值
06 ; -----
07 ; 按键端口状况值
08 K1_N EQU 34H ; 存放按键当前端口状况值
09 K1_P EQU 35H ; 存放按键上次端口状况值
10 K1_C EQU 37H ; 存放按键计数单元
11 X EQU 36H ; LCD 地址变量
12 ;
  
```

```

13 ; 按键引脚定义
14 K1 EQU P3.3 ;按键 1 引脚定义
15 K2 EQU P3.4 ;按键 2 引脚定义
16 ; 喇叭引脚
17 SPK EQU P2.4 ;
18 ;
19 ;LCD 引脚定义
20 RS EQU P3.5 ;LCD RS 引脚定义
21 RW EQU P3.6 ;LCD RW 引脚定义
22 E EQU P3.7 ;LCD RS 引脚定义
23 ;
24 ORG 0000H ;程序由地址 0 开始执行
25 JMP MAIN
26 ORG 0BH ;定时器 T0 中断地址设置
27 JMP T0_INT
28 ; - - - - - 主程序 - - - - -
29 MAIN:
30 MOV SP, #60H ;堆栈指针指向 60H
31 CLR E ;E=0, 禁止读/写 LCM
32 ACALL SET_LCD ;调用 LCD 控制子程序
33 ACALL INIT ;初始化变量
34 MOV K1_P, #01H ;按键上次端口设置为 1
35 ACALL INIT_TIMER ;初始化定时器
36 ACALL MEU ;调用工作菜单子程序
37 LOOP: ACALL CONV ;时间计数处理
38 ACALL LOOP1 ;调用清 0 键子程序
39 ACALL KEY ;判断是否有键按下
40 JZ LOOP ;无按键, 则跳转至 LOOP 处
41 MOV K1_P, K1_N ;交换数据
42 ACALL KEY0 ;调用按键功能子程序
43 JMP LOOP ;跳转到 LOOP 处
44
45 ;
46 ; 初始变量清 0 子程序
47 INIT: ;初始变量清 0
48 CLR A ;A 清 0
49 MOV K1_C, A ;K1_C 初始为 0
50 MOV DEDA, A ;DEDA 初始为 0
51 MOV SEC, A ;SEC 初始为 0
52 MOV MIN, A ;MIN 初始为 0
53 MOV HOUR, A ;HOUR 初始为 0
54 MOV K1_N, A ;K1_N 初始为 0
55 MOV K1_P, A ;K1_P 初始为 0

```

```

56     CLR    TR0          ;启动中断
57     RET
58 ; -----
59 ;定时器初始化设置子程序
60 INIT_TIMER:                 ;定时器初始化
61     MOV    TMOD, #00000001B ;定时器 T0 模式 1
62     MOV    IE, #10000010B   ;开通中断
63     MOV    TL0, #LOW(65536-10000) ;定时初值装入低位
64     MOV    TH0, #HIGH(65536-10000) ;定时初值装入高位
65     RET
66 ; -----
67 ;中断服务程序
68 TO_INT:                     ;定时器 T0 中断程序
69     PUSH   ACC           ;压入栈保护
70     MOV    TL0, #LOW(65536-10000) ;重加载
71     MOV    TH0, #HIGH(65536-10000)
72     INC    DEDA
73     MOV    A, DEDA        ;10ms 计数值加 1
74     CJNE   A, #100,TT
75     MOV    DEDA, #0
76     INC    SEC            ;秒加 1
77     MOV    A, SEC
78     CJNE   A, #60,TT
79     INC    MIN            ;分加 1
80     MOV    SEC, #0
81     MOV    A, MIN
82     CJNE   A, #60,TT
83     INC    HOUR           ;时加 1
84     MOV    MIN, #0
85     MOV    A, HOUR
86     CJNE   A, #24,TT
87     MOV    DEDA, #0
88     MOV    SEC, #0         ;秒、分、时单元清 0
89     MOV    MIN, #0
90     MOV    HOUR, #0
91 TT:    POP    ACC           ;出栈
92     RETI               ;中断程序返回
93
94 ; -----
95 ;判断键是否按下子程序
96 LOOP1:
97     JB    K2, LOOP2       ;判断清 0 键是否按下
98     ACALL  SPK_BZ

```

```

99      JMP     MAIN          ;跳转至主程序处
100     LOOP2: RET
101     ;-----
102     ; 判断 K1 键是否按下
103     KEY:
104     CLR     A              ;A 清 0
105     MOV     K1_N, A        ;A 值送入 K1_N
106     MOV     C, K1           ;K1 值送入 C
107     RLC     A              ;同进位标志左移一位
108     ORL     K1_N, A        ;两个位作逻辑 OR 运算
109     MOV     A, K1_N         ;K1_N 值送入 A
110     XRL     A, K1_P         ;有键按下，A 中内容不为 0
111     RET
112
113; -----
114 ;功能键子程序
115 ;第一次按 K1 键功能子程序
116 KEY0:
117     MOV     A, K1_P         ;K1_P 值送入 A
118     JB    ACC.0, KEY3       ;A 的 0 位是 1, 转到 KEY3
119     INC     K1_C            ;K1_C 加 1
120     MOV     A, K1_C         ;K1_C 值送入 A
121     CJNE   A, #01H, KEY1       ;K1 键是否第一次按?
122     MOV     DPTR, #MENU1      ;是，则存入 MENU1 信息
123     MOV     A, #1             ;设置第一行显示
124     ACALL  LCD_PRINT        ;调用显示字符子程序
125     SETB   TR0              ;启动中断
126     RET
127 ; -----
128 ;第二次按 K1 键功能子程序
129 KEY1:
130     MOV     A, K1_C         ;K1_C 值送入 A
131     CJNE   A, #02H, KEY2       ;K1 键是否第二次按?
132     MOV     DPTR, #MENU2      ;是，存入 MENU2 信息
133     MOV     A, #1             ;设置第一行显示
134     ACALL  LCD_PRINT        ;调用显示字符子程序
135     CLR     TR0              ;停止中断
136     RET
137 ; -----
138 ;第三次按 K1 键功能子程序
139 KEY2:
140     MOV     A, K1_C         ;K1_C 值送入 A
141     CJNE   A, #03H, KEY3       ;K1 键是否第三次按?

```

```

142     MOV      DPTR, #MENU3      ;是，存入 MENU3 信息
143     MOV      A, #1           ;设置第一行显示
144     ACALL   LCD_PRINT       ;调用显示字符子程序
145     SETB    TR0            ;启动中断
146     RET
147 ; -----
148 ;第四次按 K1 键功能子程序
149 KEY3:
150     MOV      A, K1_C         ;K1_C 值送入 A
151     CJNE   A, #04H, KEY4    ;K1 键是否第四次按?
152     MOV      DPTR, #MENU4    ;是，存入 MENU4 信息
153     MOV      A, #1           ;设置第一行显示
154     ACALL   LCD_PRINT       ;调用显示字符子程序
155     CLR    TR0            ;启动中断
156 KEY4:
157     RET                  ;子程序返回
158
159; ----- LCD 显示 -----
160 ; LCD 控制子程序
161 SET_LCD:    ;
162     CLR    E
163     ACALL  INIT_LCD        ;初始化 LCD
164     MOV    R5, #10
165     ACALL  DELAY
166     MOV    DPTR, #LMESS1    ;指针指到显示消息 1
167     MOV    A, #1           ;显示在第一行
168     ACALL  LCD_PRINT       ;调用显示字符子程序
169     MOV    DPTR, #LMESS2    ;指针指到显示消息 2
170     MOV    A, #2           ;显示在第二行
171     ACALL  LCD_PRINT       ;调用显示字符子程序
172     RET
173 ; -----
174 LMESS1:  DB  "          ", 0      ;LCD 第一行显示消息
175 LMESS2:  DB  "TIME      ", 0      ;LCD 第二行显示消息
176 ; -----
177
178 ;LCD 初始化子程序
179 INIT_LCD:
180     MOV    A, #38H          ;设置 8 位、2 行、5×7 点阵
181     ACALL  WR_COMM        ;调用写指令子程序
182     ACALL  DELAY1         ;调用延时子程序
183     MOV    A, #0CH          ;开显示，光标不闪烁
184     ACALL  WR_COMM        ;调用写指令子程序

```

```

185      ACALL    DELAY1          ;调用延时子程序
186      MOV      A, #01H         ;清除 LCD 显示屏
187      ACALL    WR_COMM        ;调用写指令子程序 ;
188      ACALL    DELAY1          ;调用延时子程序
189      RET
190
191; -----
192 ;写指令子程序
193 WR_COMM:
194      MOV      P1,A
195      CLR      RS             ;RS=0 ,选择指令寄存器
196      CLR      RW             ;RW=0 ,选择写模式
197      SETB    E              ;E=1 ,允许读/写 LCM
198      ACALL    DELAY1          ;调用延时子程序
199      CLR      E              ;E=0 ,禁止读/写 LCM
200      RET
201; -----
202
203 ;写数据子程序
204 WR_DATA:
205      MOV      P1, A
206      SETB    RS             ;RS=1 ,选择数据寄存器
207      CLR      RW             ;RW=0 ,选择写模式
208      SETB    E              ;E=1 ,允许读/写 LCM
209      ACALL    DE             ;调用延时子程序
210      CLR      E              ;E=0 ,禁止读/写 LCM
211      ACALL    DE             ;调用延时子程序
212      RET
213
214; -----
215 ;清除该行 LCD 的字符
216 CLR_LINE:
217      MOV      R0, #24
218 CL1:   MOV      A, #' '
219      ACALL    WR_DATA
220      DJNZ    R0, CL1
221      RET
222; -----
223 ;LCD 存入工作菜单
224 MEU:
225      MOV      DPTR, #MENU0     ;存入工作菜单
226      MOV      A, #1           ;第一行
227      ACALL    LCD_PRINT

```

```

228          RET
229 ; -----
229 ; 工作菜单
230 MENU0:   DB  " SECOND-CLOCK 0 ",0
231 MENU1:   DB  " BEGIN COUNT 1 ",0
232 MENU2:   DB  " PAUST COUNT 2 ",0
233 MENU3:   DB  " BEGIN COUNT 3 ",0
234 MENU4:   DB  " PAUST COUNT 4 ",0
235 ; -----
236 ; -----
237 ; 菜单显示子程序
238 ;一行、二行显示字符
239 LCD_PRINT:
240     CJNE    A, #1, LINE2      ;判断是否为第一行
241 LINE1:
242     ACALL   CLR_LINE        ;清除该行字符数据
243     MOV     A, #80H           ;设置 LCD 的第一行地址
244     ACALL   WR_COMM         ;写入命令
245     JMP     FILL
246 LINE2:
247     ACALL   CLR_LINE        ;清除该行字符数据
248     MOV     A, #0C0H          ;设置 LCD 的第二行地址
249     ACALL   WR_COMM         ;写入命令
250 FILL:   CLR    A           ;填入字符
251     MOVC    A, @A+DPTR       ;由消息区取出字符
252     CJNE    A, #0,LC1         ;判断是否为结束码
253     RET
254 ; -----
255 ; 写入数据
256 LC1:    ACALL   WR_DATA
257     INC    DPTR             ;指针加 1
258     JMP    FILL             ;继续填入字符
259     RET
260
261 ; -----
261 ; 转换数据子程
262 CONV:      ;转换为 ASCII 码并显示
263     MOV    A, HOUR           ;加载小时数据
264     MOV    X, #5              ;设置位置
265     ACALL  SKOW_LINE2        ;显示数据
266     INC    X
267     MOV    A, '#:'
268     MOV    B, X

```

```

269      ACALL    LCDP2
270      MOV      A, MIN          ;加载分钟数据
271      INC      X              ;设置位置
272      ACALL    SKOW_LINE2     ;显示数据
273      INC      X
274      MOV      A, #':'
275      MOV      B, X
276      ACALL    LCDP2
277      MOV      A, SEC          ;加载秒数数据
278      INC      X              ;设置位置
279      ACALL    SKOW_LINE2     ;显示数据
280      INC      X
281      MOV      A, #':'
282      MOV      B, X
283      ACALL    LCDP2
284      MOV      A, DEDA         ;加载秒数数据
285      INC      X              ;设置位置
286      ACALL    SKOW_LINE2     ;显示数据
287      RET      ;
288
289 ; -----
290 ;在 LCD 的第二行显示数字
291 SKOW_LINE2:
292      MOV      B, #10           ;设置被除数
293      DIV      AB             ;结果 A 存商数 , B 存余数
294      ADD      A, #30H          ;A 为十位数 , 转换为字符
295      PUSH     B              ;B 压入堆栈暂存
296      MOV      B,X            ;设置 LCD 显示的位置
297      ACALL    LCDP2          ;由 LCD 显示出来
298      POP      B              ;出栈
299      MOV      A, B            ;B 为个位数
300      ADD      A, #30H          ;转换为字符
301      INC      X              ;LCD 显示位置加 1
302      MOV      B, X            ;设置 LCD 显示的位置
303      ACALL    LCDP2          ;由 LCD 显示出来
304      RET      ;
305 ; -----
306 ;在 LCD 的第二行显示字符
307 LCDP2:
308      PUSH     ACC
309      MOV      A, B          ;设置显示地址
310      ADD      A, #0C0H        ;设置 LCD 的第二行地址
311      ACALL    WR_COMM        ;写入命令

```

```
312      POP      ACC          ;由堆栈取出 A
313      ACALL    WR_DATA     ;写入数据
314      RET
315
316 ; -----
317 ;喇叭\鸣响子程序
318 SPK_BZ:
319      MOV      R6, #100
320 B1:   ACALL    DEX
321      CPL      SPK
322      DJNZ    R6, B1
323      MOV      R5, #10
324      ACALL    DELAY
325      RET
326 DEX:  MOV      R7, #180
327 DE1:  NOP
328      DJNZ    R7, DE1
329      RET
330
331 ; -----
332 ;延时 10ms
333     DELAY:
334      MOV      R6, #50
335 D1:   MOV      R7, #100
336      DJNZ    R7, $
337      DJNZ    R6, D1
338      DJNZ    R5, DELAY
339      RET
340 ; -----
341 ;延时 5ms 子程序
342     DELAY1:
343      MOV      R6, #25
344 DEY:  MOV      R7, #100
345      DJNZ    R7, $
346      DJNZ    R6, DEY
347      RET
348 ; -----
349 ;延时 500μs
350     DE:
351      DJNZ    R7, $
352      RET
353 ; -----
356      END          ;程序结束
```

24.1.3 代码详解

1. 主要标号说明

- MAIN：主程序。
- KEY：判断是否按键子程序。
- KEY0：按键功能子程序。
- INIT：初始变量清0子程序。
- INIT_TIMER：定时器初始化子程序。
- T0_INT：定时器0中断子程序。
- CONV：转换为ASCII码子程序。
- SKOW_LINE2：LCD第二行显示数字子程序。
- LCDP2：LCD第二行显示字符子程序。
- WR_COMM：写指令子程序。
- WR_DATA：写数据子程序。
- SET_LCD：LCD控制子程序。
- INIT_LCD：LCD初始化子程序。
- LCD_PRINT：菜单显示子程序。
- CLR_LINE：清除行字符子程序。
- MEU：存入工作菜单子程序。
- DELAY：延时10ms子程序。
- DELAY1：延时5ms子程序。
- DE：延时500μs子程序。
- SPK_BZ：喇叭鸣响子程序。

2. 程序分析解释

03~05：定义计时单元地址，分别存放小时变量、分钟变量、秒钟变量和10ms计数值。

08~10：定义34H、35H、37H地址，分别存放按键开关K1当前端口状况值K1_N、上次端口状况值K1_P和按键计数次数K1_C。

- 11：存放LCD地址变量。
- 14：K1控制P3.3引脚。
- 15：K2控制P3.4引脚。
- 17：喇叭引脚
- 20：定义LCM的RS引脚由单片机的P3.5引脚控制。
- 21：定义LCM的RW引脚由单片机的P3.6引脚控制。
- 22：定义LCM的E引脚由单片机的P3.7引脚控制。
- 24、25：主程序MAIN由地址0开始。
- 26、27：0BH为定时器T0中断地址。
- 29~36：进行初始化设置，包括定时器、LCD、按键上次端口及调用工作菜单等，进入

工作状态。

37 ~ 43 : 调用数据转换子程序和判断是否有键按下及清 0 键子程序。在判断是否有键按下的子程序里，无键按下时，经逻辑运算语句处理后，A 中内容为 0，故 JZ 指令使程序跳转到标号 LOOP 处循环查询。当有键按下后，A 中内容不为 0，程序回到主程序后向下运行，调用按键功能子程序，如此循环。

47 ~ 57 : 将已经定义的变量，初始时清除为 0。

60 ~ 65 : 设置定时器 T0 在方式 1 下工作，并开通中断。

68 ~ 92 : 中断服务程序，产生秒、分、时的计时数据。

98 ~ 100 : 判断 K2 键是否按下，若按下程序将回到主程序开始处运行，使计时从 0 开始。按键的同时还调用喇叭鸣响子程序，发出一声响声。

103 ~ 111 : 判断 K1 功能键是否按下，有键按下，A 中内容不为 0。

116 ~ 157 : 为 K1 功能键子程序。K1 键每按一次，计数单元 K1_C 内的值便加 1，根据对 K1_C 值的比较来确定按下的次数，并根据按下的次数执行相应的功能。

161 ~ 191 : 对 LCD 进行初始化设置，包括清显示屏、开显示屏、显示光标，以及在第一行和第二行显示消息等。

193 ~ 200 : 写指令子程序。

204 ~ 212 : 写数据子程序。

216 ~ 220 : 清除行字符子程序。

224 ~ 228 : LCD 存入工作菜单子程序。

230 ~ 234 : 工作菜单表。

238 ~ 259 : 在显示屏上显示工作菜单。其中第一行上显示列表上的工作菜单；在第二行上显示“TIME”。

262 ~ 287 : 转换数据子程序，将时、分、秒计时数转换为 ASCII 码，以便在液晶显示屏上显示。

291 ~ 304 : 在 LCD 第二行显示数字子程序。功能是在第二行上显示计时的时、分、秒数。

307 ~ 314 : 在 LCD 第二行显示字符子程序。功能是在第二行显示的时、分、秒中间显示隔离号“：“。

318 ~ 329 : 喇叭鸣响子程序。

335 ~ 339 : 延时子程序，延时的时间为 10ms。

342 ~ 347 : 延时子程序，延时的时间为 5ms。

350 ~ 352 : 延时子程序，延时的时间为 500μs。

356 : 程序结束。

24.1.4 操作说明

液晶秒表上的 K1 为功能键，K2 为清 0 键。秒表接通电源后，LCD 第一行显示菜单为“SECOND-CLOCK 0”。

第一次按下 K1 按钮时，显示菜单为“BEGIN COUNT 1”，计时开始。

第二次按下时显示菜单为“PAUST COUNT 2”，暂停计时。

第三次按下时，显示菜单为“BEGIN COUNT 3”，累计计时。

第四次按下时，显示菜单为“PAUST COUNT 4”，暂停计时。
在任何状态下，按一下 K2 并伴随声响，均可清 0。

24.2 液晶显示温度控制器

本装置的功能是对温度进行实时监测与控制。由温度传感器 DS18B20 对温度进行采样和转换成数字信号送入单片机，并与设定的报警温度上、下限值进行比较，信息通过 LCD 显示出来。

如果实时温度超过设定的上、下限值，一方面由 LCD 显示信息，并发出警报声；另一方面自动控制继电器（Relay）接通或断开，从而控制加热源的开与断，达到对温度进行实时控制的目的。

24.2.1 硬件设计

液晶显示温度控制器硬件结构也很简单，在单片机的 P1 端口接有 LCD 模块，用来显示温度。

P2.5 引脚与温度传感器 DS18B20D 的信号线引脚 DQ 连接，作为数据总线传递控制指令，接收温度信息。

P2.1 ~ P2.4 引脚分别接有 K1 ~ K4 4 个按钮开关，作为温度报警值的输入和调整，其电路如图 24.3 所示。

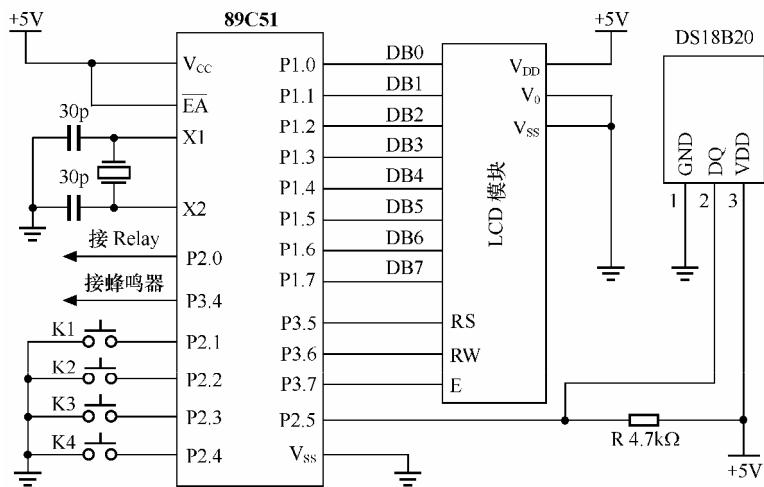


图 24.3 液晶显示温度控制器电路

单片机的 P2.0 引脚接继电器（Relay）由继电器控制加热设备。

P3.4 引脚接蜂鸣器，这两部分电路没有画出。

24.2.2 程序设计

1. 工作原理

程序开始时首先对 LCD 进行初始化设置，并写入报警温度上、下限值。然后对温度传感器 DS18B20 进行复位，检测是否存在，如果传感器没有正常工作，LCD 显示屏上会显示

出“ERROR”的信息，如果工作正常显示屏上将显示出“OK”。

接着读取温度数据，再经转换，由LCD显示屏显示出来。同时，不断地将实时温度与设定的报警温度上、下限值进行比较，如果超过报警上限，鸣响并关断加热源；如果超过报警下限，鸣响并接通加热源。

由于报警温度的上、下限值的设定、查看和调整都是由按钮开关控制的，所以程序还要对按键进行扫描，如此不断循环，其流程如图24.4所示。

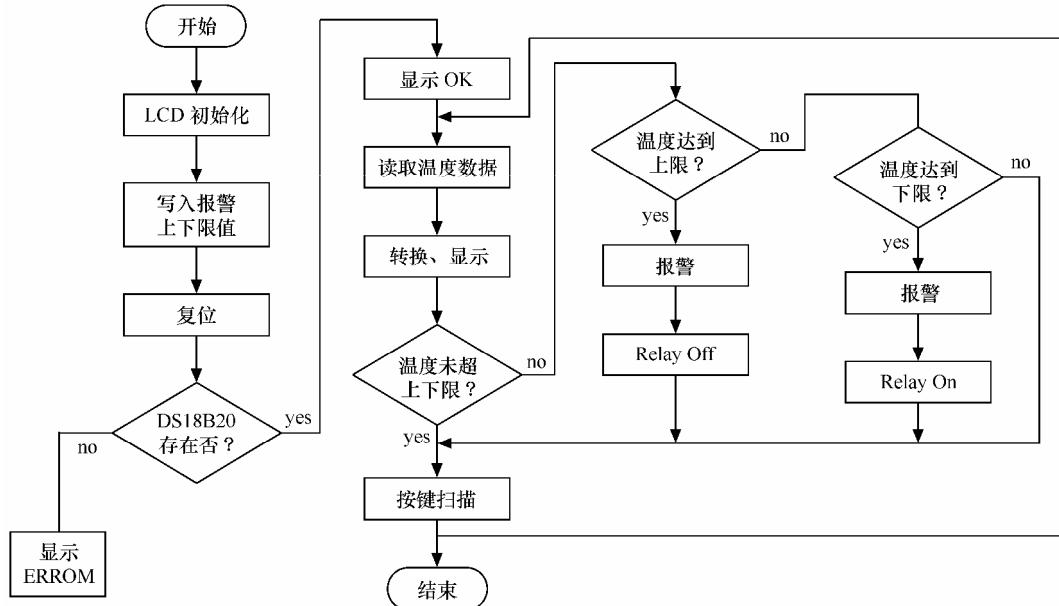


图24.4 温度控制流程

2. 程序

汇编语言编写的液晶显示温度控制器源程序LCD2402.ASM代码如下：

```

01 ; -----
02 ; 温度值存放单元
03 TEMP_ZH EQU 24H ; 实时温度值存放单元
04 TEMPL EQU 25H ; 低温度值存放单元
05 TEMP_H EQU 26H ; 高温度值存放单元
06 TEMP_TH EQU 27H ; 高温报警值存放单元
07 TEMP_TL EQU 28H ; 低温报警值存放单元
08 TEMPHC EQU 29H ; 存十位数的BCD码
09 TEMPCLC EQU 2AH ; 存个位数的BCD码
10 ;
11 ; 按键输入引脚定义
12 K1 EQU P2.1
13 K2 EQU P2.2
14 K3 EQU P2.3
  
```

```

15      K4          EQU      P2.4
16 ;
17      SPK         EQU      P3.4          ;蜂鸣器引脚
18      RELAY        EQU      P2.0          ;继电器引脚
19      X            EQU      2FH          ;LCD 地址变量
20 ;
21 ;LCD 控制引脚
22      RS           EQU      P3.5
23      RW           EQU      P3.6
24      E            EQU      P3.7
25 ;
26      FLAG         EQU      20H.0        ;DS18B20 是否存在标记
27      KEY_UD       EQU      20H.1        ;设定 KEY 的 UP 与 DOWN 标记
28      DQ           EQU      P2.5
29
30 ;===== 主程序 =====
31
32 MAIN:
33      ACALL      SET_LCD          ;LCD 初始化设置子程序
34      ACALL      WR_THL          ;将报警上下限写入暂存寄存器子程序
35 TOOP: ACALL      RESET_1820      ;调用 18B20 复位子程序
36      JNB        FLAG, TOOP1      ;DS1820 不存在，则跳转至 TOOP1 处
37      ACALL      MEU_OK          ;调用显示“OK”信息子程序
38      ACALL      RE_THL          ;把 E2ROM 里温度报警值拷贝回暂存器
39      ACALL      TEMP_BJ          ;显示温度标记“ ”
40      JMP        TOOP2
41 TOOP1: ACALL      MEU_ERROR      ;显示“ERROR”信息
42      ACALL      TEMP_BJ          ;显示温度标记
43      JMP        $
44 ;
45 TOOP2:
46      ACALL      RE_TEMP          ;调用读取温度数据子程序
47      ACALL      SET_DATA          ;调用处理显示温度数据子程序
48      ACALL      TEMP_COMP          ;实际温度值与标记温度值比较子程序
49      ACALL      P_KEY           ;调用按键扫描子程序
50      SJMP      TOOP2          ;循环
51
52 ;----- 读取温度数据子程序 -----
53 RE_TEMP:
54      ACALL      RESET_1820      ;18B20 复位子程序
55      JNB        FLAG, TOOP1      ;DS1820 不存在
56      MOV        A, #0CCH          ;跳过 ROM 匹配
57      ACALL      WRITE_1820      ;写入子程序

```

```

58     MOV      A, #44H          ;发出温度转换命令
59     ACALL   WRITE_1820       ;调用写入子程序
60     ACALL   RESET_1820       ;调用复位子程序
61     MOV      A, #0CCH          ;跳过 ROM 匹配
62     ACALL   WRITE_1820       ;写入子程序
63     MOV      A, #0BEH          ;发出读温度命令
64     ACALL   WRITE_1820       ;写入子程序
65     ACALL   READ_1820        ;调用读取子程序
66     RET
67
68 ;----- 温度数据处理显示子程序 -----
69 SET_DATA:
70     ACALL   CONV_TEMP        ;处理温度 BCD 码子程序
71     ACALL   DISP_BCD         ;显示区 BCD 码温度值刷新子程序
72     ACALL   CONV             ;LCD 显示子程序
73     RET
74
75 ;----- 按键键扫描子程序 -----
76 P_KEY: ;按键 K1 处理
77     JB      K1, PK1          ;K1 键未按，则跳转到 PK1 处
78     ACALL   SPK_BZ           ;K1 键按下，发出一声响声
79     JNB    K1,$              ;等按键放开
80     MOV    DPTR, #M_ALAX1    ;存 M_ALAX1 表
81     MOV    A, #1
82     ACALL   LCD_PRINT       ;显示字符
83     ACALL   LOOK_ALARM       ;显示信息区子程序
84     JB      K3, $            ;等待 K3 按下
85     ACALL   SPK_BZ           ;发出一声响声
86     JMP    PK2
87 PK1:
88     JB      K2, PK3          ;K2 键未按，则跳转到 PK3 处
89     ACALL   SPK_BZ           ;K2 键按下，发出一声响声
90     JNB    K2,$              ;等按键放开
91     MOV    DPTR, #TA1         ;存#TA1 表
92     MOV    A, #1
93     ACALL   LCD_PRINT       ;显示字符
94     ACALL   SET_ALARM        ;设定报警值 TH、TL
95     ACALL   WR_THL           ;将设定的 TH, TL 值写入 DS18B20 内
96     ACALL   WRITE_E2          ;调用报警值拷贝 E2ROM 子程序
97 PK2:
98     ACALL   MEU_OK            ;显示“OK”信息子程序
99     ACALL   TEMP_BJ           ;显示温度标记子程序
100 PK3:

```

```

101      RET
102 ; -----
103 TA1:                                ;菜单表
104     DB  "RESET ALERT CODE"
105 ; -----
106
107 ; ----- 设定报警值 TH、TL 子程序 ----- 
108 SET_ALARM:
109     ACALL    LOOK_ALARM      ;调用显示信息区子程序
110 A0:   JB       K1, A2        ;按下 K1(查看键), 程序向下运行
111     ACALL    SPK_BZ         ;蜂鸣器响一声
112     JNB      K1, $          ;等待放开
113     CPL      20H.1          ;UP/DOWN 标记反向
114 A2:   JB       20H.1, A3      ;20H.1=1, UP, 跳转到 A3。
115     JMP      A8            ;20H.1=0, DOWN, 跳转到 A8
116 ;
117 ; TH 值调整(增加)
118 A3:   JB       K2, A5        ;按下 K2 键, 程序向下运行
119     ACALL    SPK_BZ         ;蜂鸣器响一声
120     INC      TEMP_TH        ;TH 值调整(增加)
121     MOV      A, TEMP_TH      ;TH 值送入 A
122     CJNE    A, #120, A4      ;TH 值增到 120, 程序向下运行
123     MOV      TEMP_TH, #0      ;TH 值清 0
124 A4:   ACALL    LOOK_ALARM      ;调用显示信息区子程序
125     MOV      R5, #10
126     ACALL    DELAY         ;调用延时程序
127     JMP      A3            ;循环
128 ;
129 ;TL 值调整(增加)
130 A5:   JB       K3, A7        ;按下 K3 键程序向下运行
131     ACALL    SPK_BZ         ;蜂鸣器响一声
132     INC      TEMP_TL        ;TL 值增加 1
133     MOV      A, TEMP_TL      ;TL 值送入 A
134     CJNE    A, #99, A6      ;比较, 若 A=99, 程序向下运行
135     MOV      TEMP_TL, #00H    ;TL 值清 0
136 A6:   ACALL    LOOK_ALARM      ;调用显示信息区子程序
137     MOV      R5, #10
138     ACALL    DELAY         ;调用延时程序
139     JMP      A5            ;程序跳转到 A5, 循环
140 ;
141 ;确定调整 OK
142 A7:   JB       K4, A0        ;按下 K4 键, 程序向下运行
143     ACALL    SPK_BZ         ;蜂鸣器响一声

```

```

144     JNB      K4, $          ;等放开
145     RET                  ;程序返回
146 ; -----
147 ;TH 值调整 (减少)
148 A8:   JB      K2, A10      ;按下 K 键，程序向下运行
149     ACALL    SPK_BZ        ;蜂鸣器响一声
150     DEC      TEMP_TH       ;TH 值减 1
151     MOV      A, TEMP_TH     ;TH 值送入 A
152     CJNE    A, #0FFH, A9      ;比较，若 A=0FF，程序向下运行
153     JMP      A12            ;跳转到 A12
154 A9:   ACALL    LOOK_ALARM    ;调用显示信息区子程序
155     MOV      R5, #10
156     ACALL    DELAY         ;调用延时子程序
157     JMP      A0              ;跳转到 A12
158 ; -----
159 ;TL 值调整 (减少)
160 A10:  JB      K3, A13      ;按 K3 键，程序向下运行
161     ACALL    SPK_BZ        ;蜂鸣器响一声
162     DEC      TEMP_TL       ;TL 值减 1
163     MOV      A, TEMP_TL      ;TH 值送入 A
164     CJNE    A, #0FFH, A11      ;比较，若 A=0FF，程序向下运行
165     JMP      A12            ;转移到 A12
166 A11:  ACALL    LOOK_ALARM    ;调用显示信息区子程序
167     MOV      R5, #10
168     ACALL    DELAY         ;调用延时子程序
169     JMP      A0              ;转移到 A0
170 A12:  CPL      20H.1        ;UP/DOWN 标记反向
171     JMP      A3              ;跳转到 A3，TH 值调整 (增加)
172 A13:  JMP      A7              ;跳转到 A7，确定调整 OK
173     RET                  ;子程序返回
174
175 ; ----- 显示信息区子程序 ----- 
176 LOOK_ALARM:
177     MOV      DPTR, #M_ALAX2    ;存表
178     MOV      A, #2            ;显示在第二行
179     ACALL    LCD_PRINT       ;调用显示字符子程序
180     MOV      A, #0C6H          ;显示起始地址为第二行第 7 位
181     ACALL    TEMP_BJ1         ;调用显示温度标记子程序
182     MOV      A, TEMP_TH        ;加载 TH 数据
183     MOV      X, #3            ;设置位置
184     ACALL    SHOW_LINE2H       ;显示数据
185 ; -----
186     MOV      A, #0CEH          ;设定显示起始地址为第二行第 15 位

```

```

187      ACALL    TEMP_BJ1          ;调用显示温度标记子程序
188      MOV      A,TEMP_TL        ;加载 TL 数据
189      MOV      X, #12           ;设置位置
190      ACALL    SHOW_LINE2L      ;显示数据
191      RET
192 ; -----
193 M_ALAX1:
194     DB      " LOOK ALERT CODE",0
195 M_ALAX2:
196     DB      "TH:      TL:      ",0
197 ; -----
198 ; ----- 显示温度标记子程序 -----
199 TEMP_BJ1:
200     ACALL    WR_COMM          ;调用写指令子程序
201     MOV      DPTR, #BJ2        ;存代码表
202     MOV      R1, #0            ;使指针指到表中第一个码
203     MOV      R0, #2            ;取码次数
204 B0:    MOV      A, R1           ;A 为 0
205     MOVC    A, @A+DPTR        ;读取码
206     ACALL    WR_DATA          ;调用写数据子程序
207     INC      R1              ;R1 值加 1
208     DJNZ    R0, B0            ;判断是否将代码读取完？
209     RET
210 ; -----
211 BJ2:                           ;代码表
212     DB      00H,"C"
213 ; -----
214
215 ; ----- 在 LCD 的第二行显示高温数字 -----
216 SHOW_LINE2H:
217     MOV      B, #100           ;设置被除数，B 为百位数
218     DIV      AB              ;除法运算，结果 A 存商数，B 存余数
219     ADD      A,#30H           ;低半字节加 30 得到 ASCII 码（转换为字符）
220     PUSH    B                ;B 放入堆栈暂存起来
221     MOV      B,X             ;设置 LCD 显示的位置
222     ACALL    LCDP2           ;由 LCD 显示出来
223 ; -----
224     POP      B                ;B 由堆栈取出来
225     MOV      A, #0AH           ;A 赋值 10
226     XCH    A,B              ;A、B 数据互换，B 为十位数
227     DIV      AB              ;除法运算，结果 A 存商数，B 存余数
228     ADD      A, #30H           ;转换为字符
229     INC      X                ;LCD 显示位位置加 1

```

```

230      PUSH    B          ;B 放入堆栈暂存起来
231      MOV     B, X       ;设置 LCD 显示的位置
232      ACALL   LCDP2     ;由 LCD 显示出来
233 ; - - - - -
234      POP     B          ;B 由堆栈取出来
235      INC     X          ;LCD 显示位位置加 1
236      MOV     A, B       ;B 为个位数
237      MOV     B, X       ;设置 LCD 显示的位置
238      ADD     A, #30H    ;转换为字符
239      ACALL   LCDP2     ;由 LCD 显示出来
240      RET
241
242 ; - - - - 在 LCD 的第二行显示低温数字 - - - -
243 SHOW_LINE2L:
244      MOV     B, #100     ;设置被除数
245      DIV     AB         ;除法运算，结果 A 存商数，B 存余数
246      MOV     A, #0AH     ;A 赋值 10
247      XCH     A, B       ;A、B 数据互换，B 为十位数
248      DIV     AB         ;除法运算，结果 A 存商数，B 存余数
249      ADD     A, #30H    ;转换为字符
250      PUSH    B          ;B 压入堆栈暂存起来
251      MOV     B, X       ;设置 LCD 显示的位置
252      ACALL   LCDP2     ;由 LCD 显示出来
253 ; - - - - -
254      POP     B          ;B 由堆栈取出来
255      INC     X          ;LCD 显示位位置加 1
256      MOV     A, B       ;B 为个位数
257      MOV     B, X       ;设置 LCD 显示的位置
258      ADD     A, #30H    ;转换为字符
259      ACALL   LCDP2     ;由 LCD 显示出来
260      RET
261
262 ; - - - - - 温度值比较子程序 - - - - -
263
264 ; 实际温度值与标记温度值比较
265 TEMP_COMP:
266      MOV     A, TEMP_TH   ;高温报警值送入 A
267      SUBB   A, TEMP_ZH   ;减数>被减数，则
268      JC     TCL1        ;借位标志位 C=1，转到 TCL1
269      MOV     A, TEMP_ZH   ;实时温度送入 A
270      SUBB   A, TEMP_TL   ;减数>被减数，则
271      JC     TCL2        ;借位标志位 C=1，转到 TCL2
272      MOV     DPTR, #BJ5    ;存表 BJ5

```

```

273      ACALL    TEMP_BJ3          ;调用显示高、低温度及加热标记子程序
274      CLR      RELAY           ;继电器吸合
275      RET
276 ; -----
277 TCL1:   ;实时温度>高温报警值的处理程序
278      MOV      DPTR ,#BJ3        ;存入#BJ3 表
279      ACALL    TEMP_BJ3          ;调用显示高、低温度及加热标记子程序
280      SETB    RELAY           ;继电器关闭
281      ACALL    SPK_BZ           ;调用鸣响子程序
282      RET
283 ; -----
284 TCL2:   ;实时温度<高温报警值的处理程序
285      MOV      DPTR ,#BJ4        ;存入#BJ4 表
286      ACALL    TEMP_BJ3          ;调用显示高、低温度及加标记子程序
287      ACALL    SPK_BZ           ;调用鸣响子程序
288      RET
289
290 ; ---- 显示高、低温度及加热标记程序 -----
291 TEMP_BJ3:
292      MOV      A, #0CEH          ;设定显示位置
293      ACALL    WR_COMM          ;调用写指令子程序
294      MOV      R1, #0            ;使指针指到表中第一个码
295      MOV      R0, #2            ;取码次数
296 BJJ:   MOV      A, R1           ;A 为 0
297      MOVC    A, @A+DPTR        ;读取码
298      ACALL    WR_DATA          ;调用写数据子程序
299      INC     R1             ;R1 值加 1
300      DJNZ   R0,BJJ           ;判断是否将代码读取完？
301      RET
302 ; -----
303 BJ3:
304      DB     ">H"
305 BJ4:
306      DB     "<L"
307 BJ5:
308      DB     " !"
309 ; -----
310
311 ; ----- 报警上下线写入暂存器子程序 -----
312 WR_THL:
313      JB      FLAG, WR_T ;FLAG=1 , 表示 DS18B20 存在 , 转 WR_T 处
314      RET
315 ; -----

```

```

316 WR_T:
317     ACALL    RESET_1820      ;调用 18B20 复位子程序
318     MOV      A, #0CCH       ;跳过 ROM 匹配
319     ACALL    WRITE_1820      ;调用写入子程序
320     MOV      A, #4EH        ;写暂存寄存器
321     ACALL    WRITE_1820      ;调用写入子程序
322     MOV      A, TEMP_TH      ;TH(报警上限)
323     ACALL    WRITE_1820      ;调用写入子程序
324     MOV      A, TEMP_TL      ;TL(报警下限)
325     ACALL    WRITE_1820      ;调用写入子程序
326     MOV      A, #7FH        ;12 位精确度
327     ACALL    WRITE_1820      ;调用写入子程序
328     RET
329
330;----- 报警值拷贝到 EEPROM 子程序 -----
331 WRITE_E2:
332     ACALL    RESET_1820      ;调用 18B20 复位子程序
333     MOV      A, #0CCH       ;跳过 ROM 匹配
334     LCALL    WRITE_1820      ;调用写入子程序
335     MOV      A, #48H        ;[48H]为拷贝到 EEPROM 的指令代码
336     LCALL    WRITE_1820      ;调用写入子程序
337     RET
338
339;----- 报警值拷贝回暂存器子程序 -----
340 RE_THL:
341     ACALL    RESET_1820      ;调用 18B20 复位子程序
342     MOV      A, #0CCH       ;跳过 ROM 匹配
343     LCALL    WRITE_1820      ;调用写入子程序
344     MOV      A, #0B8H        ;把 EEPROM 里的温度报警值拷贝回暂存器
345     ACALL    WRITE_1820      ;调用写入子程序
346     RET
347
348;----- 处理温度 BCD 码子程序 -----
349 CONV_TEMP:
350     MOV      A, TEMPH       ;判断温度是否零下
351     ANL      A, #80H
352     JZ      TC1           ;温度为零，则转到 TC1 处
353     CLR      C             ;C=0
354     MOV      A, TEMPL       ;二进制数求补（双字节）
355     CPL      A             ;取反加 1
356     ADD      A, #01H
357     MOV      TEMPL, A
358     MOV      A, TEMPH

```

```

359      CPL      A
360      ADDC    A, #00H
361      MOV     TEMP.H, A
362      JMP     TC2
363 ; -----
364 TC1:   MOV     TEMP.HC, #0AH
365 TC2:   MOV     A, TEMP.HC
366      SWAP    A           ;高、低位交换
367      MOV     TEMP.HC, A
368      MOV     A, TEMP.L
369      ANL     A, #0FH        ;乘 0.0625
370      MOV     DPTR, #DOTTAB
371      MOVC   A, @A+DPTR
372      MOV     TEMP.LC, A      ;TEMP.LC, LOW 为小数部分, BCD
373 ; -----
374      MOV     A, TEMP.L       ;整数部分
375      ANL     A, #0F0H        ;取出高 4 位
376      SWAP    A           ;高、低位交换
377      MOV     TEMP.L, A
378      MOV     A, TEMP.H
379      ANL     A, #0FH        ;取出低 4 位
380      SWAP    A           ;高、低位交换
381      ORL     A, TEMP.L      ;重新组合
382      MOV     TEMP.ZH, A
383      LCALL  HEX2BCD1
384      MOV     TEMP.L, A
385      ANL     A, #0F0H        ;取出高 4 位
386      SWAP    A           ;高、低位交换
387      ORL     A, TEMP.HC      ;TEMP.HC, LOW 为十位数, BCD
388      MOV     TEMP.HC, A
389      MOV     A, TEMP.L
390      ANL     A, #0FH        ;取出低 4 位
391      SWAP    A           ;高、低位交换
392      ORL     A, TEMP.LC      ;TEMP.LC, HI 为个位数, BCD
393      MOV     TEMP.LC, A
394      MOV     A, R4
395      JZ      TC3
396      ANL     A, #0FH        ;取出低 4 位
397      SWAP    A           ;高、低位交换
398      MOV     R4, A
399      MOV     A, TEMP.HC      ;TEMP.HC HI 为百位数, BCD
400      ANL     A, #0FH        ;取出低 4 位
401      ORL     A, R4

```

```

402      MOV      TEMPHC, A
403  TC3:   RET
404; -----
405  HEX2BCD1:
406      MOV      B, #064H          ;十六进制转 BCD
407      DIV      AB             ;B= A % 100
408      MOV      R4, A           ;R4 为百位数
409      MOV      A, #0AH
410      XCH      A, B
411      DIV      AB             ;B = A % B
412      SWAP    A              ;高、低位交换
413      ORL      A, B
414      RET
415; -----
416; 小数部分码表
417  DOTTAB:
418      DB      00H,00H,01H,01H,02H,03H,03H,04H
419      DB      05H,05H,06H,06H,07H,08H,08H,09H
420; -----
421
422; ----- 显示区 BCD 码温度值刷新子程序 -----
423  DISP_BCD:
424      MOV      A, TEMPLC        ;个位数 BCD 码送入 A
425      ANL      A, #0FH          ;取低位码
426      MOV      70H, A           ;小数位
427      MOV      A, TEMPLC
428      SWAP    A              ;高、低位交换
429      ANL      A, #0FH          ;取结果数的高位
430      MOV      71H, A           ;个位
431      MOV      A, TEMPHC
432      ANL      A, #0FH          ;取低位码
433      MOV      72H, A           ;十位
434      MOV      A, TEMPHC
435      SWAP    A              ;高、低位交换
436      ANL      A, #0FH          ;取结果数的高位
437      MOV      73H, A           ;百位
438      MOV      A, TEMPHC
439      ANL      A, #0FOH         ;取低位码
440      CJNE    A, #010H,D10
441      JMP     D12
442; -----
443  D10:   MOV      A, TEMPHC
444      ANL      A, #0FH          ;取低位码

```

```

445      JNZ      DI2          ;十位数是 0
446      MOV      A, TEMPHC
447      SWAP    A           ;高、低位交换
448      ANL      A, #0FH      ;取结果数的高位
449      MOV      73H, #0AH     ;符号位不显示
450      MOV      72H, A       ;十位数显示符号
451  DI2:   RET
452
453 ; ----- DS18B20 复位初始化子程序 -----
454  RESET_1820:                 ;复位(有具体的时序要求)
455      SETB    DQ
456      NOP
457      CLR     DQ
458 ;
459 ;主机发出延时 537μs 的复位低脉冲
460      MOV      R1, #3
461  DLY:   MOV      R0, #107
462      DJNZ    R0, $
463      DJNZ    R1, DLY
464 ;
465 ;然后拉高数据线
466      SETB    DQ
467      NOP
468      NOP
469      NOP
470 ;
471 ;等待 DS18B20 回应
472      MOV      R0, #25H
473  T2:   JNB      DQ, T3
474      DJNZ    R0, T2
475      JMP     T4
476 ;
477 ;置标志位 FLAG=1, 表示 DS1820 存在
478  T3:   SETB    FLAG
479      JMP     T5
480 ;
481 ;清标志位 FLAG=0, 表示 DS1820 不存在
482  T4:   CLR     FLAG
483      JMP     T7
484 ;
485 ;时序要求延时一段时间
486  T5:   MOV      R0, #117
487  T6:   DJNZ    R0, T6

```

```

488 ;
489 T7:    SETB      DQ
490        RET
491
492; - - - - - 写入 DS18B20 子程序 - - - - -
493 ;写入 DS18B20
494 WRITE_1820:
495     MOV      R2, #8          ;一共 8 位数据
496     CLR      C             ;C=0
497     WR1:
498     CLR      DQ            ;总线低位，开始写入
499     MOV      R3, #6
500     DJNZ    R3, $           ;保持 16μs 以上
501     RRC      A             ;把字节 DATA 分成 8 位，环移给 C
502     MOV      DQ, C           ;写入一个位
503     MOV      R3, #23
504     DJNZ    R3, $           ;等待
505     SETB    DQ             ;重新释放总线
506     NOP
507     DJNZ    R2, WR1         ;写入下一个位
508     SETB    DQ             ;释放总线
509     RET
510
511; - - - - - 读出 DS18B20 子程序 - - - - -
512 ;将温度值从 DS18B20 中读出
513 READ_1820:
514     MOV      R4, #4
515     MOV      R1, #TEMPL       ;存入 25H、26H、27H、28H
516     RE0:
517     MOV      R2, #8          ;数据一共有 8 位
518     RE1:
519     CLR      C
520     SETB    DQ
521     NOP
522     NOP
523     CLR      DQ            ;读前总线保持为低位
524     NOP
525     NOP
526     NOP
527     SETB    DQ             ;总线释放
528;
529     MOV      R3, #9
530     DJNZ    R3, $           ;延时 18μs

```

```

531      MOV      C, DQ          ;从总线读到一个位
532 ; -----
533      MOV      R3, #23 ; 
534      DJNZ    R3, $          ;等待 50μs
535      RRC     A              ;把读得的位环移给 A
536      DJNZ    R2, RE1         ;读下一个位
537      MOV      @R1, A
538      INC     R1              ;R1 内数据递增
539      DJNZ    R4, RE0         ; 
540      RET
541
542 ===== LCD 1602 显示程序 =====
543 ;初始化设置
544 SET_LCD:
545      CLR     E
546      ACALL   INIT_LCD        ;初始化 LCD
547      ACALL   STORE_DATA       ;将自定义字符存入 LCD 的 CGRAM
548      RET
549
550; ----- LCD 初始化子程序 -----
551 INIT_LCD:
552      MOV     A, #38H          ;设置 8 位、2 行、5×7 点阵
553      ACALL   WR_COMM         ;调用写指令子程序
554      ACALL   DELAY1          ;调用延时子程序
555      MOV     A, #0CH           ;开显示，光标不闪烁
556      ACALL   WR_COMM         ;调用写指令子程序 ;
557      ACALL   DELAY1          ;调用延时子程序
558      MOV     A, #01H           ;清除 LCD 显示屏
559      ACALL   WR_COMM         ;调用写指令子程序 ;
560      ACALL   DELAY1          ;调用延时子程序
561      RET
562
563; ----- 显示温度标记子程序 -----
564 TEMP_BJ:
565      MOV     A, #0CBH          ;设定第二行起始地址
566      ACALL   WR_COMM         ;调用写指令子程序
567      MOV     DPTR, #BJ          ;存代码表
568      MOV     R1, #0             ;使指针指到表中第一个码
569      MOV     R0, #2             ;取码次数
570 TP1:
571      MOV     A, R1              ;A 为 0
572      MOVC   A, @A+DPTR        ;取码
573      ACALL   WR_DATA         ;调用写数据子程序

```

```

574      INC      R1          ;R1 值加 1
575      DJNZ    R0,  TP1      ;判断是否将代码读取完 ?
576      RET
577 ; -----
578  BJ:                      ;代码表
579      DB  00H, "C"
580; -----
581
582; ----- 自定义字符子程序 ----- 
583 ;将自定义字符写入 LCD1602 的 CGRAM 中
584  STORE_DATA:
585      MOV      A, #40H      ;指定 CG RAM 起始地址
586      ACALL   WR_COMM     ;将指令写入 LCD
587      MOV      R2, #08H     ;图形数据长度 8 个字节
588      MOV      DPTR, #TAB  ;存代码表
589      MOV      R3, #00H     ;使指针指到表中第一个码
590  S_D:  MOV      A, R3      ;A 为 0
591      MOVC    A, @A+DPTR   ;读取表代码
592      ACALL   WR_DATA    ;调用写入数据指令
593      INC      R3          ;R3 值加 1
594      DJNZ    R2,  S_D      ;判断是否将代码读取完
595      RET
596 ; -----
597  TAB:                      ;代码表
598      DB  0CH,12H,12H,0CH
599      DB  00H,00H,00H,00H
600 ; -----
601
602; ----- 显示“OK”信息子程序 ----- 
603  MEU_OK:
604      MOV      DPTR, #M_OK1  ;指针指到显示消息
605      MOV      A, #1        ;显示在第一行
606      ACALL   LCD_PRINT   ;LCD 显示
607      MOV      DPTR, #M_OK2  ;指针指到显示消息
608      MOV      A, #2        ;显示在第一行
609      ACALL   LCD_PRINT   ;LCD 显示
610      RET
611 ; -----
612  M_OK1:
613      DB  "  DS18B20 OK  ",0
614  M_OK2:
615      DB  "  TEMP:      ",0
616 ; -----

```

```

617
618 ;----- 显示“ERROR”信息子程序 -----
619 MEU_ERROR:
620     MOV      DPTR, #M_ERROR1      ;指针指到显示消息 1
621     MOV      A, #1                ;显示在第一行
622     ACALL   LCD_PRINT          ;调用菜单显示子程序
623     MOV      DPTR, #M_ERROR2      ;指针指到显示消息 1
624     MOV      A, #2                ;显示在第一行
625     ACALL   LCD_PRINT          ;调用菜单显示子程序
626     RET
627 ;
628 M_ERROR1:                      ;代码表
629     DB      " DS18B20 ERROR  ", 0
630 M_ERROR2:
631     DB      " TEMP: ----      ", 0
632 ;
633
634 ;----- 菜单显示子程序 -----
635 ;在 LCD 的第一行或第二行显示字符
636 LCD_PRINT:
637     CJNE    A, #1, LINE2        ;判断是否为第一行
638 LINE1:
639     ACALL   CLR_LINE          ;清除该行字符数据
640     MOV     A, #80H             ;设置 LCD 的第一行地址
641     ACALL   WR_COMM            ;写入命令
642     JMP    FILL
643 LINE2:
644     ACALL   CLR_LINE          ;清除该行字符数据
645     MOV     A, #0C0H             ;设置 LCD 的第二行地址
646     ACALL   WR_COMM            ;写入命令
647 FILL:
648     CLR    A                  ;填入字符
649     MOVC   A,@A+DPTR          ;由消息区取出字符
650     CJNE   A, #0, LC1          ;判断是否为结束码
651     RET
652 LC1:
653     ACALL   WR_DATA            ;写入数据
654     INC    DPTR              ;指针加 1
655     JMP    FILL               ;继续填入字符
656     RET
657
658 ;----- LCD 显示子程序 -----
659 CONV:

```

```

660      MOV      A, 73H          ;加载百位数据
661      MOV      X, #6           ;设置位置
662      CJNE    A, #1, CO1
663      JMP     CO2
664 CO1:
665      MOV      A, #"
666      MOV      B, X
667      ACALL   LCDP2
668      JMP     CO3
669 CO2:
670      ACALL   SHOW_LINE2       ;显示数据
671 CO3:   INC     X             ;位加 1
672      MOV      A, 72H          ;十位
673      ACALL   SHOW_LINE2       ;显示数据
674      INC     X             ;位加 1
675      MOV      A, 71H          ;个位
676      ACALL   SHOW_LINE2       ;显示数据
677      INC     X             ;位加 1
678      MOV      A, #'.'
679      MOV      B, X
680      ACALL   LCDP2          ;显示字符
681      MOV      A, 70H          ;加载小数点位
682      INC     X             ;设置位置
683      ACALL   SHOW_LINE2       ;显示数据
684      RET
685
686;----- 显示第二行 -----
687 ;在 LCD 的第二行显示数字
688 SHOW_LINE2:
689      ADD     A, #30H
690      MOV     B, X
691      ACALL   LCDP2
692      RET
693 LCDP2:                      ;在 LCD 的第二行显示字符
694      PUSH    ACC            ;压入堆栈
695      MOV     A, B            ;设置显示地址
696      ADD     A, #0C0H         ;设置 LCD 的第二行地址
697      ACALL   WR_COMM        ;写入命令
698      POP     ACC            ;由堆栈取出 A
699      ACALL   WR_DATA        ;写入数据
700      RET
701
702 ;----- 写指令子程序 -----

```

```

703 WR_COMM:
704     MOV      P1, A          ;写入指令
705     CLR      RS           ;RS=0 , 选择指令寄存器
706     CLR      RW           ;RW=0 , 选择写模式
707     SETB    E            ;E=1 , 允许读/写 LCM
708     ACALL   DELAY1       ;延时 5ms
709     CLR      E            ;E=0 , 禁止读/写 LCM
710     RET
711
712 ;----- 写数据子程序 -----
713 WR_DATA:
714     MOV      P1, A          ;写入数据
715     SETB    RS           ;RS=1 , 选择数据寄存器
716     CLR      RW           ;RW=0 , 选择写模式
717     SETB    E            ;E=1 , 允许读/写 LCM
718     ACALL   DE           ;延时 0.5ms
719     CLR      E            ;E=0 , 禁止读/写 LCM
720     ACALL   DE           ;延时 0.5ms
721     RET
722
723 ;----- 清除 LCD 的字符 -----
724 CLR_LINE:
725     MOV      R0, #24        ;设置计数值
726 CL1:   MOV      A, #' '
727     ACALL   WR_DATA       ;输出字符至 LCD
728     DJNZ    R0, CL1        ;判断
729     RET
730
731 CLR_LINE1:                   ;清除 LCD 的第一行字符
732     MOV      A, #80H        ;设置 LCD 的第一行地址
733     ACALL   WR_COMM       ;调用写指令子程序
734     MOV      R0, #24        ;设置计数值
735 C1:   MOV      A, #' '
736     ACALL   WR_DATA       ;输出字符至 LCD
737     DJNZ    R0, C1         ;计数结束
738     RET
739
740 ;----- 鸣响子程序 -----
741 SPK_BZ:
742     MOV      R6, #100
743 BZ2:   ACALL   DEX1
744     CPL      SPK
745     DJNZ    R6, BZ2

```

```

746      MOV      R5, #10
747      ACALL   DELAY
748      RET
749 DEX1:   MOV      R7, #180
750 DE2:    NOP
751      DJNZ    R7, DE2
752      RET
753
754 ;----- 延时子程序 -----
755 DELAY:           ;延时时间为 R5×10ms
756 DL1:   MOV      R7, #100
757      DJNZ    R7, $
758      DJNZ    R6, DL1
759      DJNZ    R5, DELAY
760      RET
761 ;
762 DELAY1:          ;延时时间为 5ms
763      MOV      R6, #25
764 DL2:   MOV      R7, #100
765      DJNZ    R7, $
766      DJNZ    R6, DL2
767      RET
768 ;
769 DE:              ;延时时间为 0.5ms
770      MOV      R7, #250
771      DJNZ    R7, $
772      RET
773 ;
774      END

```

24.2.3 代码详解

1. 主要标号说明

- MAIN：主程序。
- SET_LCD：LCD 初始化设置子程序。
- WR_THL：将报警上下限写入暂存寄存器子程序。
- SET_DATA：处理显示温度数据子程序。
- TEMP_COMP：实际温度值与标记温度值比较子程序。
- P_KEY：按键扫描子程序。
- CONV_TEMP：处理温度 BCD 码子程序。
- DISP_BCD：显示区 BCD 码温度值刷新子程序。
- CONV：LCD 显示子程序。

- RESET_1820 : DS18B20 复位子程序。
- WRITE_1820 : DS18B20 写入子程序。
- READ_1820 : DS18B20 读取子程序。
- INIT_LCD : LCD 初始化子程序。
- WR_COMM : LCD 写指令子程序。
- WR_DATA : LCD 写数据子程序。

2. 程序分析解释

01 ~ 29 : 定义温度值存放单元、按键输入引脚、LCD 控制引脚、传感器 DS18B20 的信号线引脚、继电器和蜂鸣器引脚等。

33 ~ 43 : 对传感器 DS18B20 和显示设备 LCD 进行初始化设置。

45 ~ 50 : 调用读取温度数据子程序、处理显示温度数据子程序、实时温度与设置的报警上下限温度比较子程序和按键扫描子程序，这 4 个子程序是整个程序工作的主要内容，所以不断循环地调用。上述每个子程序中还包含下一级的子程序。

53 ~ 66 : 读取温度数据子程序。先进行复位，搜索 DS18B20 是否存在，如果存在，主机将发出[CCH]指令与在线的 DS18B20 联系，接着向 DS18B20 发出温度 A/D 转换[44H]指令，再发出读取温度寄存器的温度值[BE]指令。在上述过程中还需要调用复位、写入及读取数据子程序。

69 ~ 73 : 温度数据处理和显示子程序，包括转 BCD 码、显示区温度值刷新和在 LCD 显示子程序。

76 ~ 260 : 主要是对按键扫描，如果确定有键按下，使程序转到相应功能子程序中。其中第 76 ~ 105 行语句完成查看温度报警值和退出查看状态的功能，第 108 ~ 260 行语句主要完成报警值的设定和调整。

265 ~ 308 : 实际温度值与报警上下限值进行比较，其比较结果用来控制继电器（由继电器控制加热源）开或断；同时还在 LCD 显示屏上用“>H”或“<L”符号显示出来，并随之发出报警声。

312 ~ 346 : 包括将报警值写入暂存器、拷贝到 EEPROM 中和拷贝回暂存器 3 个子程序。其中将暂存器中的报警值拷贝到 EEPROM 中，是为了保证报警值数据不丢失；再从 EEPROM 中拷贝回到暂存器是为了读取该数据，以便与实时温度进行比较。

349 ~ 451 : 将温度数据转换成 BCD 码，以便在 LCD 显示。

454 ~ 540 : 包括对传感器 DS18B20 进行复位、写、读 3 个子程序，每一步都有严格的时序要求，是使用 DS18B20 的基本操作模块。其中复位检测 DS18B20 是否存在；写是指将数据写入温度寄存器中；读是指读取温度寄存器的温度值。

544 ~ 561 : 对 LCD 初始化。

564 ~ 657 : 主要在 LCD 显示屏上显示菜单及温度标志“”，其中“C”左上角的小圆圈是自制图形。

660 ~ 685 : 主要在 LCD 第二行指定位置上显示温度值。

704 ~ 722 : 包含写入指令和写数据两个子程序。写指令是指写入指令寄存器中，写数据是指写入到数据寄存器中，先写指令后才能写数据，这是使用 LCD 的基本操作。这一点与使用传感器 DS18B20 有点相同之处，使用 DS18B20 时，也是先将数据写入，之后再读取。

725 ~ 740 : 程序通过向 LCD 写入空格的办法 , 来清除 LCD 的字符。
 743 ~ 753 : 鸣响子程序 , 在程序中调用一次 APK_BZ , 可听到一次报警声。
 756 ~ 773 : 3 个延时子程序。
 775 : 程序结束。

24.2.4 操作说明

1 . 检测传感器 DS18B20 工作状态

液晶显示温度控制器接通电源后 , 在工作正常情况下 , 液晶显示屏上第一行显示信息为 “ DS18B20 OK ” ; 第二行显示为 “ TEMP : × × . × ” (实际温度值) 。若传感器 DS18B20 工作不正常 , 显示屏上第一行显示信息为 “ DS18B20 ERROR ” ; 第二行显示为 “ TEMP:--- ” 。这时要检查 DS18B20 是否连接好 , 如果连接没问题 , 则需要更换一个新的 DS18B20 芯片。

2 . 查看温度报警值

按 K1 键 , 进入查看温度报警值状态 , 此时显示屏第一行显示为 “ LOOK ALERT CODE ” ; 第二行显示 “ TH:0 × × TL: × × ” 。其中 TH 为高位报警值 , TL 为低位报警值。按 K3 键退出查看温度报警状态。

3 . 设定温度报警值

按 K2 键 , 进入设定温度报警值状态 , 此时显示屏第一行显示为 “ RESET ALERT CODE ” ; 第二行显示 “ TH:0 × × TL: × × ” 。

此时的 K1 键为设定值加、减方式选择键 , 默认为减少。 K2 键为 TH 值设定键 ; K3 键为 TL 值设定键 ; K4 键为确定键 , 按此键退出设定状态。

4 . 报警状态显示标志

(1) 当实际温度大于 TH 的设定值时 , 在显示屏第二行上显示符号为 “ >H ” 。此时关闭继电器 , 蜂鸣器响起 , 表示超温。

(2) 当实际温度小于 TL 的设定值时 , 在显示屏第二行上显示符号为 “ <L ” 。此时蜂鸣器也会响起 , 表示加热部分出现故障。

(3) 当实际温度小于 TH 的设定值时 , 继电器吸合 , 开始加热 , 加热的标记为 “ ! ” 。

5 . 简单测试

报警上、下限值设定后 , 可以进行简单测试。

(1) 用手捏住 DS18B20 管 , 会看到显示屏的温度不断上升 , 当上升的温度超过设定的上位报警值时 , 蜂鸣器会响起 ; 然后再将手放开 , 会看到显示屏的温度开始不断下降 , 当降到低于上位报警值时 , 蜂鸣器停止响声。

(2) 用冰放在 DS18B20 管处 , 会看到显示屏上的温度迅速下降 , 当下降的温度低于设定的下位报警值时 , 蜂鸣器也会响起 ; 然后再将冰拿走 , 会看到显示屏的温度开始不断上升 , 当温度超过设定的下位报警值时 , 蜂鸣器停止响声。

DS18B20 的测温范围为 : -55 ~ +125 , 在 -10 ~ +85 时精度为 ± 0.5 。



第 25 章 动手制作单片机实验板

制作单片机实验板并不难，只要按照本章的讲解，一步一步地去制作，你就能制作出自己的实验板。

25.1 制作实验板准备工作

25.1.1 制作实验板的目的

虽然已经购买了实验板，但最好还是要亲手制作一块实验板，这样能满足自己不断深入学习和编程实验的需要，熟悉单片机外围常用元器件的选择、测量和使用方法，并能增强动手能力和实践创新能力。

1. 满足学习实验的需要

购买的简易实验板，端口连接的元器件都是相对固定的，所以实验项目也有限，随着学习的不断深入会感到实验板的功能有限，而拓展又很不方便。自己设计制作实验板可以在端口使用上更为灵活，并且，将实验板留有充足的扩展实验余地，可根据需要不断去增加实验项目。

制作实验板可以一次性完成，也可以事先大体规划好后，根据学习进度逐步制作完成，需要实验哪部分，就制作那部分，这样能把软件设计和硬件设计结合得更紧密。

2. 熟悉单片机外围常用元器件

单片机实验板所使用的元器件，也是单片机实际应用中常使用到的元器件。通过制作实验板，可以熟悉这些元器件的选择、简单测量和使用方法。

3. 增强动手能力

通过制作实验板，既可以了解单片机电子产品的制造过程，熟悉电子产品的制作工艺。也增强了实践动手能力，掌握制作单片机电子产品的操作技能，为今后进行单片机产品的设计开发打下一定的基础。

25.1.2 制作前的准备工作

1. 多孔实验板和面包板

多孔实验板如图 25.1 所示，板上布满小孔，用来插元器件，小孔的一面周围有铜箔，便于焊接，其尺寸、规格可根据用途选定。自己制作单片机实验板可选用尺寸大些的，留有充足的扩展余地。

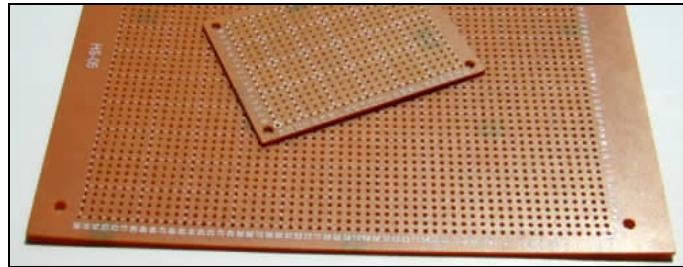
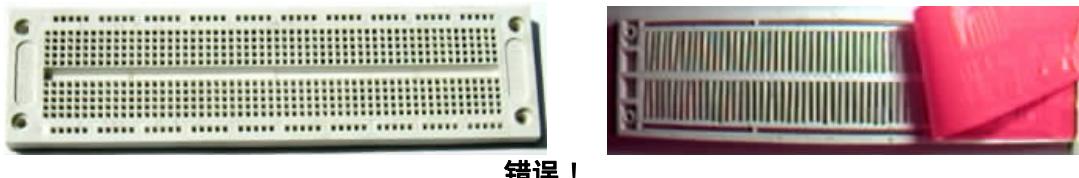


图 25.1 多孔实验板

面包板是插接式的实验板，也称万用接线板，如图 25.2 所示，其中，图 25.2 (a) 为面包板的正面，图 25.2 (b) 为面包板的背面。



错误！

(a) 正面

(b) 背面

图 25.2 面包板

面包板上面布满小孔，小孔插元器件时有一定紧度，而且，从面包板的背面可以看出，每 5 个小孔连接在一起为一行，起到连线作用。所以，使用面包板插元器件时不需要焊接，可以通过导线插接。面包板对临时性实验搭接电路非常方便，不但免去焊接和拆焊的麻烦，也避免元器件受到损坏。

下面主要介绍的是使用多孔板制作单片机实验板的方法。

2. 插件及开关

实验板上用到的插件及开关如图 25.3 所示，其中，标号 1 为实验插座，使用该插座插拔单片机非常方便。标号 2 为插针座，座上有双排插针，用于安装在实验插座两边，一排插针与实验插座引脚相接，另一排与实验模块相连。

标号 3 也是插针座，与前者不同的是安装在多孔板后，上面露出的是插针孔。标号 4 是 8 针配套的插座，插针安装在多孔板上，8 根连接线与插孔座相连。标号 5 也是插针座，下面是圆针，上面露出的是圆孔，可用作数码管的插座。标号 6 是指拨开关，标号 7 是微型按钮

开关。

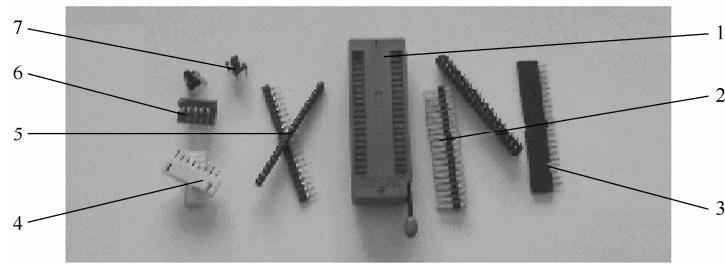


图 25.3 插件及开关

3. 工具

制作实验板所用工具如图 25.4 所示，万用表用于测量元件，剥线钳能剥较细的导线，因为在制作实验板时所需要的连线都是较细的导线。其他工具有尖嘴钳子，用于夹持零件、导线及零件脚弯折，在焊接时候夹持元件可以防止元件因过热而损坏。斜口钳子主要用途是剪断导线、零件脚等。镊子主要用于夹持小的元器件，辅助焊接，弯曲电阻、电容和导线。



图 25.4 常用工具

25.1.3 焊接技巧

焊接技术是电子爱好者必须掌握的一项基本技术，需要多多练习，熟能生巧。

1. 选择电烙铁及焊锡

焊接单片机实验板时，电烙铁功率不易过大，一般为 25~30W，电烙铁头应为圆锥形，如图 25.5 所示，圆锥形适合焊接热敏感元件。

焊锡选用焊接电子元件用的低熔点焊锡丝，焊锡丝要细一些。电烙铁在使用前要上锡，具体方法是：将电烙铁烧热，待刚刚能熔化焊锡时，涂上助焊剂，再用焊锡均匀地涂在烙铁头上，使烙铁头均匀地吃上一层锡。所使用的助焊剂可用 25% 的松香溶解在 75% 的酒精中制成，也可以购买专用的焊油。

图 25.6 所示为吸锡器，主要用途是检修时将零件上的焊锡吸走，以便更换元件。



图 25.5 圆锥形电烙铁



图 25.6 吸锡器

2. 电路板和元器件引脚的处理

电路板要清除铜箔面氧化层，办法是用细砂纸打磨干净后，再涂上助焊剂。元件的引脚和连接导线的线头，在插入多孔板之前，都必须清洁后镀上锡。

3. 焊点的控制

焊接时，电烙铁头部蘸锡量要适当，不可太少，也不可太多。烙铁头在焊点处稍停留一下后，再轻轻往上一提离开焊点。好的焊点表面应当光亮、圆滑、无锡刺、锡量适中。同时，还要注意焊接时间不宜过长，否则容易烫坏元件，必要时可用镊子夹住管脚帮助散热。

焊接完成后，要用酒精把线路板上残余的助焊剂清洗干净，以防炭化后的助焊剂影响电路正常工作。

25.2 单片机外围常用元器件及其检测方法

25.2.1 发光二极管和 LED 数码管

1. 发光二极管

发光二极管 (LED) 是一种能发光的二极管，与普通的二极管一样由一个 PN 结组成，P 为正极，N 为负极，当正向连接时，即 P 接正极、N 接负极时二极管导通；反之，二极管截止，这就是二极管的单向导电特性。当导通有足够的正向电流通过发光二极管 PN 结时，发光二极管便会发光。

发光二极管的种类很多，有发红光、黄光、绿光、蓝光、紫光和变色发光的二极管，其图形符号如图 25.7 所示，外形如图 25.8 所示。

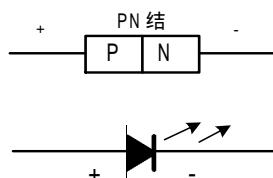


图 25.7 发光二极管符号



图 25.8 发光二极管外形

2. LED 数码管

LED 数码管结构与工作原理见 7.1 节。

图 25.9(a) 为 4 个连在一起的共阳极数码管，图 25.9(b) 为引脚图，外部有 12 个引脚，其中，C1、C2、C3、C4 分别为 4 个数码管的公共端，接电源正极。



图 25.9 LED 数码管外观与引脚图

3. 简单检测

使用万用表检测发光二极管和 LED 数码管的方法相同的，具体方法是将万用表的挡位开关拨在“Rx10k”挡，用万用表黑表笔（表内电池正极）接发光二极管的正极，红表笔（表内电池负极）接发光二极管的负极，这时发光二极管为正向接入，表针应偏转过半。

将两表笔对调后与发光二极管相连，这时发光二极管为反向接入，表针应不动，说明发光二极管是好的。

如果无论正向接入还是反向接入表针都不动，则说明发光二极管内部断路，已经损坏。如果无论正向接入还是反向接入表针都偏转很大，则说明发光二极管内部被击穿，已损坏。

一般发光二极管的引脚较长的为正极，较短的为负极，数码管的极性可参照引脚图来测量。

25.2.2 三极管

晶体三极管通常简称为晶体管或三极管，三极管有 3 个电极，分别为发射极 e、集电极 c 和基极 b，它是一种具有两个 PN 结的半导体器件。

1. 三极管作用

三极管在电路中主要起到对电信号放大或开关作用，在单片机实验板里，三极管主要起开关作用。

2. 三极管的表示符号

三极管的表示符号如图 25.10 所示。三极管有 PNP 和 NPN 两种结构，例如 9015 三极管为 PNP 结构，9014 为 NPN 结构。其中，PNP 结构三极管表示符号中发射极箭头向里；NPN 结构的三极管表示符号中发射极箭头向外。

使用三极管时要注意，两种结构的三极管在电路中供电极性不同，PNP 三极管工作时发射极 e 接正极，集电极 c 和基极 b 接负极，电流由发射极 e 流向集电极 c 和基极 b；

NPN 三极管工作时，发射极 e 接负极，集电极 c 和基极 b 接正极，电流由集电极 c 和基极 b 流向发射极 e，也就是说，在三极管的符号中发射极 e 的箭头方向代表着电流通过该管的流向。

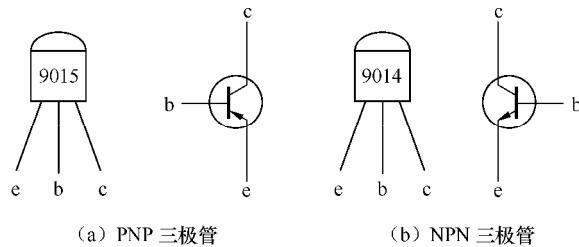


图 25.10 三极管表示符号

3. 简单检测

三极管可以用万用表进行管脚识别和检测，图 25.11 是三极管结构示意图。

从电路可知，对于 PNP 型三极管，c、e 极分别为两个内部 PN 结的正极，b 极为它们共同的负极；NPN 三极管情况恰好相反。显然，根据前面讲的二极管检测知识，就能很快判断出是 PNP 型还是 NPN 型三极管，并找出基极 b，检测时将万用表置于“R $\times 1k$ ”挡。

(1) 先确定基极。

先任意假定一个极为基极（实际上，多数三极管中间引脚为基极），用红表笔接假定的基极，再用黑表笔分别接触另两个脚。

如果万用表的读数都很小（PNP 结正向连接），则与红表笔接触的那一脚便是基极。同时可知这个三极管为 PNP 型管。若用黑表笔接假定的基极，再用红表笔分别接触另两脚。如果万用表的读数都很小（NPN 结正向连接），则与黑表笔接触的那一脚便是基极。同时可知这个三极管为 NPN 型管。

(2) 确定发射极和集电极。

现以 PNP 型三极管为例，确定基极后，首先假定一脚为集电极，并将红表笔接在此脚上，黑表笔则接到假定的发射极上，用手指把假定的集电极和基极提起来，这时表针应向右摆动；然后调换一下表笔再测量一次，两次测量中，表针摆动幅度较大的一次，黑表笔所接为发射极，红表笔所接为集电极。

若判别 NPN 型三极管，方法同上，表针摆动幅度较大的一次，红表笔所接为发射极，黑表笔所接为集电极。

(3) 性能的鉴别。

一般为表针摆动幅度越大，说明被测三极管的放大倍数大；如果测出发射极与集电极的正反电阻无限大，说明三极管内部断路，不能使用；如果测出发射极与集电极的正反电阻非常小，说明三极管内部被击穿而损坏。

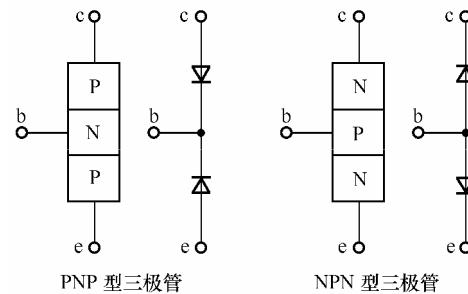


图 25.11 三极管结构示意图

25.2.3 电阻和电容

1. 电阻

电阻是电子电路常用元件，对交流、直流都有阻碍作用，常用于控制电路的电流、电压大小。

(1) 电阻符号及单位。电阻在电路中的文字代号为 R，电路图形符号及使用单位如图 25.12 所示。

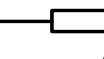
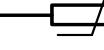
符号	R	 固定电阻符号  可调电位器符号
单位	欧姆	Ω
	千姆	$k\Omega$
	兆姆	$M\Omega$
	吉姆	$G\Omega$
	太姆	$T\Omega$
功率	瓦特	W

图 25.12 电阻符号及单位

基本单位是欧姆，简称欧 (Ω)，常用的单位还有千欧 ($k\Omega$) 和兆欧 ($M\Omega$)，它们之间的换算关系是： $1M\Omega=1000k\Omega$ ， $1k\Omega=1000\Omega$ 。

(2) 电阻色环标示法。

电阻的阻值标示方法有两种：一种是直接标示法，即在电阻上印有阻值数；另一种是色标法，如图 25.13 所示。

在电阻上印有 4 或 5 道色环表示阻值，阻值的单位为欧姆 (Ω)。紧靠电阻端的为第一色环，其余依次为第二、三、四色环。

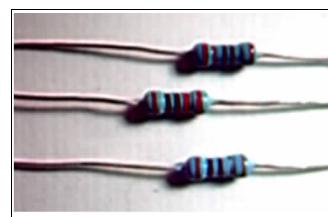


图 25.13 色标法电阻外观

对于 4 环电阻，第 1、2 环表示有效数字，第 3 环表示被乘数，第 4 环表示允许偏差。对于 5 环电阻，第 1、2、3 环表示有效数字，第 4 环表示被乘数，第 5 环表示允许偏差。精密色环电阻常用 5 道色环标示法，其他电阻一般多采用 4 环标示法。

电阻上的色环是用黑、棕、红、橙、黄、绿、蓝、紫、灰、白、金和银 12 种颜色，它们的意义如表 25.1 所示。

表 25.1

电阻色标的含意

颜色	第一色环	第二色环	第三色环	第四色环
	第一位数	第二位数	倍数	允许误差
黑	0	0	$\times 10^0$	
棕	1	1	$\times 10^1$	$\pm 1\%$

续表

颜色	第一色环	第二色环	第三色环	第四色环
	第一位数	第二位数	倍数	允许误差
红	2	2	$\times 10^2$	$\pm 2\%$
橙	3	3	$\times 10^3$	
黄	4	4	$\times 10^4$	
绿	5	5	$\times 10^5$	$\pm 0.5\%$
蓝	6	6	$\times 10^6$	$\pm 0.25\%$
紫	7	7	$\times 10^7$	$\pm 0.1\%$
灰	8	8	$\times 10^8$	
白	9	9	$\times 10^9$	
金				$\pm 5\%$
银				$\pm 10\%$

例如，某电阻的 4 道色环依次为黄、紫、橙、银，则此电阻值为 $47k\Omega$ ，误差为 $\pm 10\%$ 。再如，某电阻 5 道色环依次为棕、黑、黑、棕、金，则此电阻值为 $1k\Omega$ ，误差为 $\pm 5\%$ 。

(3) 电阻的额定功率。

额定功率也是电阻的主要参数，常用电阻的功率有 $1/8W$ 、 $1/4W$ 、 $1/2W$ 、 $1W$ 、 $2W$ 及 $5W$ 等，但大部分业余电子制作中对电阻功率没有要求，一般可选用 $1/8W$ 或 $1/4W$ 的电阻。

(4) 检测。

检测电阻时首先根据电阻的阻值大小，将万用表旋转到适当的“ Ω ”挡位，接着对电阻挡进行校 0。即将万用表两表笔互相短接，转动调零旋钮使表针指向电阻刻度的 0 位。

然后，将万用表两表笔（不分正、负）分别与电阻器的两端引线相接，表针应指在相应的阻值刻度上。如果表针不动或指示值与电阻的标示值相差太大，则说明电阻已损坏。注意在测量几十千欧以上阻值的电阻时，不可用手同时接触电阻两端引线，以免接入人体电阻会带来测量误差。

2. 电容

电容是一种存储电荷的容器，称电容器。电容器在电路中起耦合、滤波、旁路、调谐、振荡等作用。电容器的种类有固定电容器、半可调电容器、可变电容器，图 25.14 是电子制作常用的固定电容器。

(1) 符号及单位。

电容器的文字符号为 C，电路符号如图 25.15 所示，其中，25.15(a) 为电容的一般符号；25.15(b) 为有极性的电解电容器符号。

电容器的基本单位是法拉，简称法 (F)。法拉作单位在实际运用中往往显得太大，常用

微法 (μF) 纳法 (nF) 和皮法 (pF) 作为单位。

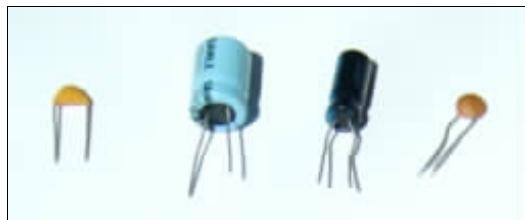


图 25.14 常用固定电容器

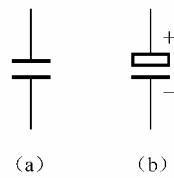


图 25.15 电容符号

(2) 检测。

电容器的好坏可以用万用表的电阻挡去检测。首先根据电容器的容量大小选择适当的挡位，一般来说， $100\mu\text{F}$ 以上的电容器用“ $\text{R} \times 100$ ”挡； $1 \sim 100\mu\text{F}$ 的电容器用“ $\text{R} \times 1\text{k}$ ”挡； $1\mu\text{F}$ 以下的电容器用“ $\text{R} \times 10\text{k}$ ”挡。

检测时，用万用表笔（不分正、负）分别去与电容器的两引线相接，在刚接触的一瞬间，表针应向右偏转，然后慢慢向左回归。接着对调表笔后再测，表针应重复以上过程。电容器的容量越大，表针向右偏转越大，向左回归越慢。

按上述方法测试，如果电容器断路，则表针不偏转；如果电容器短路，则表针偏转后不会再回归。

对于容量非常小的电容器，由于充电电流极小，几乎看不出表针摆动，只能检测是否短路。

25.3 实验板制作过程

25.3.1 实验板功能简介

制作的单片机实验板如图 25.16 所示。

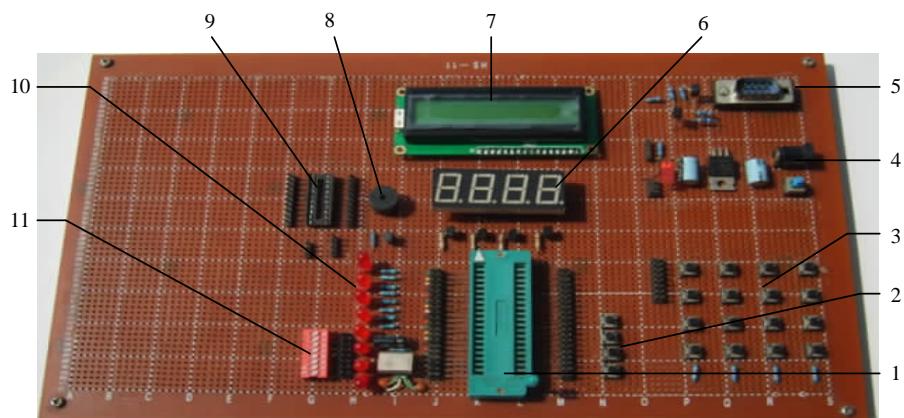


图 25.16 制作的单片机实验板

1. 实验板功能

实验板由 11 个部分（模块）组成。

- 单片机最小系统部分。所谓最小系统，是指单片机开发人员进行各类产品开发必须具有的单片机系统。

- 按键输入部分，可做简单按键输入实验。
- 键盘输入部分，可做矩阵式键盘扫描输入实验。
- 稳压电源部分，提供 5V 稳压电源。
- RS-232 串行接口部分，可进行单片机与计算机通信实验。
- LCD 显示部分，可做 LCD 显示实验。
- LED 数码管显示器部分，可进行定时计数等数据显示实验。
- 喇叭发声部分，可做单片机发声实验。
- 74 系列数字电路插座，可做串行输出输入端口扩充实验。
- 指拨开关输入部分，可做基本输入功能实验。
- LED 发光二极管部分，可做基本输出功能实验。

这些部分的相互组合，不但可以完成单片机 5 大基本功能的基本实验，还可以完成带定时闹铃和倒计时的单片机时钟设计等实验。同时，由于实验板留有充分的扩展空间，所以可根据需要增加制作实验项目。

2. 多孔板的尺寸选择

多孔板的尺寸选择上不仅要考虑到能安装现在实验所用到的元器件，而且一定还要考虑有充足的扩充余地，为进一步深入学习单片机做准备。例如进行模拟/数字转换实验、步进马达控制实验等，所以多孔板的尺寸选择可大一些。

3. 制作前要在多孔板上做好规划

制作实验板前要在多孔板上做好规划，如图 25.17 所示。

		LCD 显示器		RS-232 串行接口
74 系 列插座	喇 叭	LED 显示器		稳压电源
指 拨 开 关	发 光 二 极 管	单片机 最小系统	按 钮 开 关	键盘

图 25.17 多孔板上规划示意图

在多孔板的中间下方安装单片机最小系统部分，右边安装键盘。键盘的上方安装稳压电源和 RS-232 串行接口。单片机最小系统的上方安装 LED 显示器和 LCD 显示器；左侧安装

LED 发光二极管、蜂鸣器、指拨开关输入部分及 74 系列数字 IC 插座等。

25.3.2 简单稳压电源的制作

稳压电源的作用是为单片机提供稳定的 5V 直流工作电源。

1. 电路图

稳压电源的电路如图 25.18 所示。

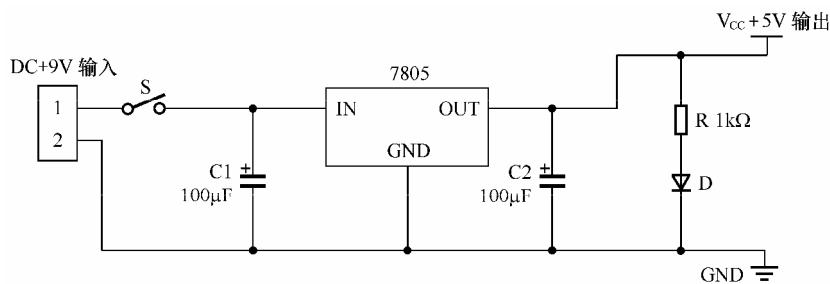


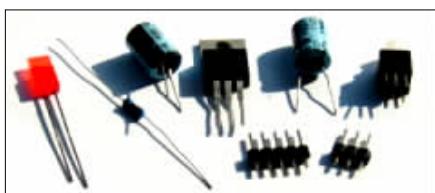
图 25.18 稳压电源电路

本章使用市面销售的 9V 电源整流器将市电交流 220V 转换为 9V 直流电源，再经过 7805 稳压和 C1、C2 电容滤波后，输出稳定的 5V 直流电压作为单片机实验板电源。电路图中 D 为发光二极管，用作电源指示灯，R 为限流电阻，S 为按键开关。

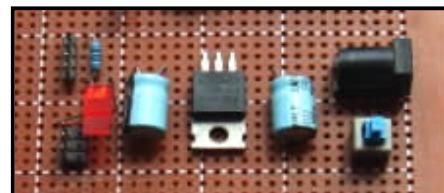
2. 元器件组装与测试

(1) 组装。

组装所需要的元器件如图 25.19 (a) 所示。



(a) 元器件



(b) 组装的稳压电源

图 25.19 稳压电源的组装

一共使用 7 个元器件，其中，稳压 7805 是个关键的器件。此外，开关选用的是微型复位按钮开关，按下时电源接通，再按一次电源断开，也可以选用其他微型开关。把这些元器件按照稳压电源原理图的要求，在多孔板所规划放置稳压电源的位置处进行组装，组装后如图 25.19 (b) 所示。

组装时主要注意元器件的极性，电容可看上边标出的引脚正负极标示；发光二极管可以通过测量来进一步确定，元件的正极引脚一定要与电源正极相连。稳压 7805 中间引脚为负极，与地相连。其他两个引脚按现在的摆放位置，右边为输入端 (IN)，左边引脚为输出端 (OUT)。

(2) 测试。

- 焊接安装完成后，要根据原理图再仔细检查一遍，看是否连接正确。
- 接通电源后，观察工作灯亮度是否正常。如果不亮，说明连接有错或假焊、露焊；如果亮度发暗，说明输出有短路的地方，应马上断开输入电源检查；如果亮度正常，说明稳压电源工作正常。
- 使用万用表直流档测量稳压电源输出端，应该有 5V 直流电压。

25.3.3 单片机最小系统的制作

单片机最小系统是指使用单片机工作时应具备的最基本条件，是单片机开发人员进行各类产品开发时必须具有的单片机系统。

1. 电路图

单片机最小系统电路如图 25.20 所示。

单片机第 40 脚 (V_{CC}) 接电源+5V，第 20 脚 (V_{SS}) 接地，为单片机工作提供电源。同时，由于 80C51 片内部带有程序存储器，当使用片内程序存储器时要将第 31 脚 (EA) 接高电平，即接到电源+5V。

复位信号第 9 引脚 (RST) 与接地 (V_{SS}) 之间连接一个下拉电阻，再与+5V 之间连接一个电容，形成单片机复位电路。

第 19 脚 (XTAL1) 与 18 脚 (XTAL2) 分别接到外部晶体两个引脚，并通过电容接地，组成石英晶体振荡器，保证单片机内部各部分有序地工作。

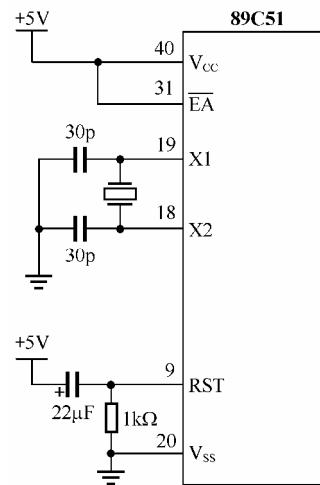


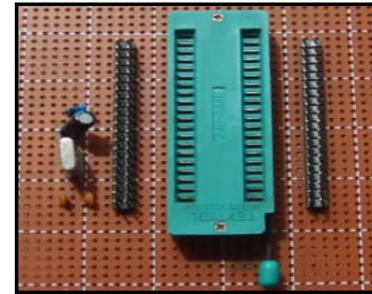
图 25.20 单片机最小系统电路

2. 元器件组装

组装的元器件如图 25.21 (a) 所示，共用 8 个元件，其中，石英晶体振荡频率为 12MHz，单片机插座采用 40 脚的实验插座，目的是使单片机插拔方便，还有两只 30pF 小电容和两条插针座，一只 1kΩ 电阻和一只 22μF 电容。将这些元件按照原理图的要求在多孔板所规划的位置上进行焊接，焊接后的单片机最小系统如图 25.21 (b) 所示。



(a) 组装的元件



(b) 最小系统

图 25.21 最小系统的组装

实验插座引脚和插针座之间，可以预先用适当长度的单根或拧成股的细铜线在多孔板的上面插入对应孔内，目的是方便焊接，其他部分的连接按照原理图进行。

3. 测试

- (1) 焊接安装完成后，要根据原理图再仔细检查一遍，看是否连接正确。
- (2) 使用万用表电阻挡，测量各元件之间连线是否有断路或短路的地方。断路是指应该导通地方测量时没有导通，原因是漏焊或虚焊；短路是指不应该连接的地方而导通，原因是焊点过大，与相邻接点有了接触或是连线接错。
- (3) 接通电源后，使用万用表直流电压挡测量单片机工作电压，将黑表笔连接第 20 脚，红表笔连接第 40 脚，测得的电压应为+5V，再将红表笔移到第 31 脚，测得的电压也应为+5V。
- (4) 如果已经将发光二极管测试模块安装完，可将相对单片机 P1 端口的插针座插针用短路块连接，运行前边讲的 LED 演示程序，应该显示工作正常。

25.3.4 LED 数码管显示模块的制作

1. 电路图

4 位 LED 数码管显示模块电路如图 25.22 所示。

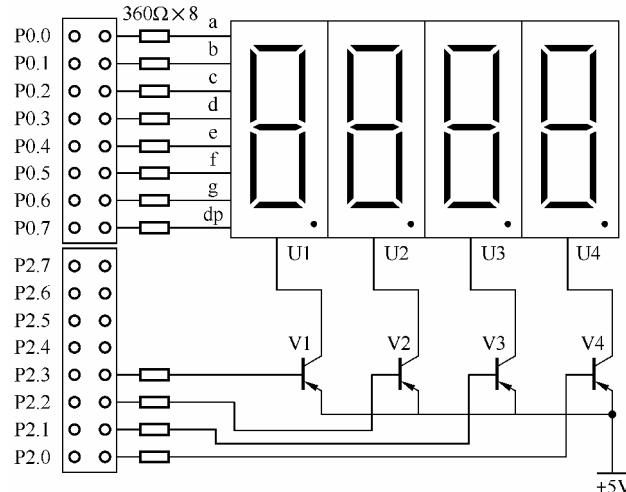


图 25.22 LED 数码管显示模块电路

4 位 LED 数码管引脚 a、b、c、d、e、f、g、dp 通过限流电阻与插针座的一排插针相连，插针座的另一排插针与单片机 P0 端口相连。4 只位选通信号放大管 V1~V4 的基极通过限流电阻接到另一插针座的 0~3 位插针上，相对应的是单片机 P2 端口的 P2.0~P2.3 位。放大管的集电极分别与数码管共阳极相连。

当使用短路块将插针座的两排插针相连时，其显示数据就能从 P0 端口分时输出到 LED 数码管显示器上。拔掉短路块后，数码管显示模块可由其他任意端口通过连线提供信号。

2. 元器件组装

组装的元器件如图 25.23 (a) 所示，有 4 位共阳极数码管显示器、8 只 470Ω 电阻作为数码管限流电阻、4 只 9012 三极管和 4 只 $1k\Omega$ 串阻作为三极管基极电阻。另外，还有两条 6 针圆孔插针座，作为数码管显示器引脚插座，使用插座的好处是避免直接焊接由于过热而损坏数码管。

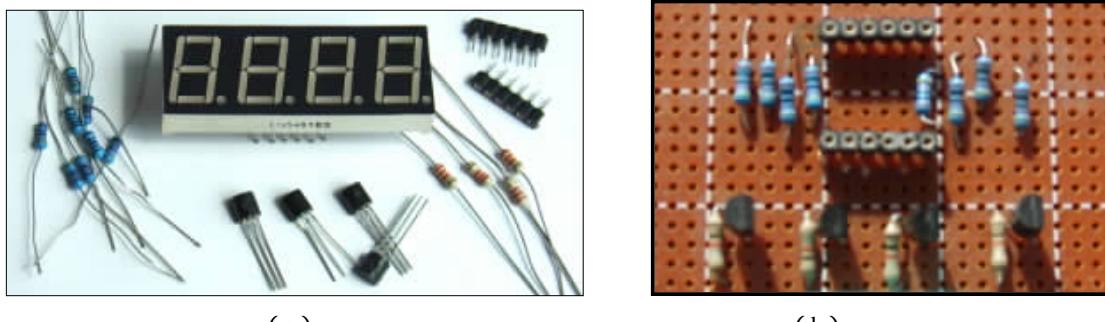


图 25.23 数码管显示模块

将这些元件按照原理图要求，在多孔板所规划的位置上进行焊接，元件具体摆放位置如图 25.23 (b) 所示。8 只 470Ω 限流电阻安排在数码管插座两边，插座的下方放置 4 只晶体管和 4 只 $1k\Omega$ 电阻。

3. 测试

- (1) 焊接安装完成后，要根据原理图再仔细检查一遍，看是否连接正确。
- (2) 使用万用表电阻挡，测量各元件之间连线是否有断路或短路的地方。
- (3) 接通电源后，使用一段导线将三极管的基极电阻与地相接，该管此时处于导通状态，提供了数码管阳极电流。再使用一段导线，导线一端接地，使用导线的另一端依次接触 P0 端口的 P0.1 ~ P0.7 处，数码管的相应笔段应该亮起。
如果发现有的笔段没亮，说明漏接；如果出现两个笔段一起亮，说明有混线的地方；如果没有按笔段标志图顺序亮，说明连线的顺序有错。
为了防止所使用的测试连接导线误将电源短路，或误将限流电阻跨接，从而造成元件通过的电流过大而烧坏，可将导线接一个 $1k\Omega$ 的电阻（起限流作用）再使用，这样既不影响短接测试观察效果，又非常安全。
- (4) 运行数码管显示程序，应该正常工作。

25.3.5 其他实验电路的制作

1. LED 发光二极管实验模块的制作

- (1) 电路。发光二极管实验电路原理如图 25.24 (a) 所示，8 只发光二极管的正极通过限流电阻连接电源+5V，二极管的负极分别与单片机 P1 端口的插针座相连，当使用短路块将插针座短路连接后，单片机就可以通过 P1 端口输出信号来控制二极管的亮与灭。

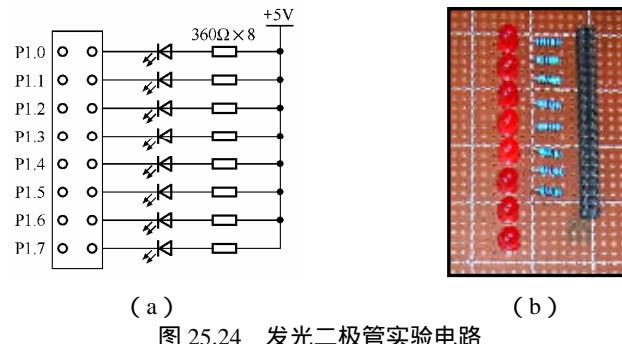


图 25.24 发光二极管实验电路

(2) 元器件组装与测试。二极管焊接时要注意极性，组装完后如图 25.24 (b) 所示。通电后可使用导线测量办法，即将导线的一端接地，用另一端依次点接发光二极管的负极，会看到依次发光。如果有的不亮，说明该二极管没焊接好或极性接反，如果两个一起亮，说明有混线的地方。

2. 发声实验模块的制作

(1) 电路及元件和组装图。

发声测试模块的电路原理如图 25.25 (a) 所示，信号由 P3.4 引脚通过电阻 R 进入三极管进行放大，推动蜂鸣器 SPK 发声。

该部分只用 3 个元件，三极管型号为 9015，电阻为 $1k\Omega$ ，蜂鸣器为微型压电式。

组装时注意蜂鸣器的极性，较长的引脚为正极，同时蜂鸣器上面有 正极标志，组装后如图 25.25 (b) 所示。

(2) 测量。

当检查组装无误后，可使用测试导线（接有 $1k\Omega$ 电阻的一段导线）一端接地，而使用另一端触接三极管基极，会听到响声。

3. 指拨开关输入模块的制作

指拨开关模块电路原理如图 25.26 (a) 所示。组装时开关的一端接地，另一端接 P3 端口的插座上，组装后如图 25.26 (b) 所示。平时开关处于断路状态，所以不会影响 P3 端口的使用。

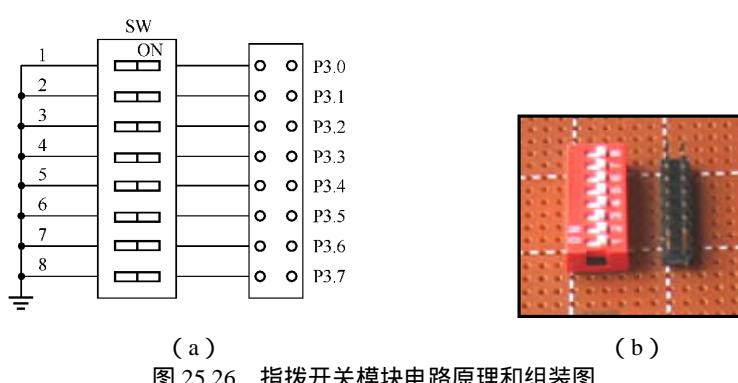


图 25.26 指拨开关模块电路原理和组装图

4. 按键开关输入模块的制作

按键开关模块电路原理如图 25.27 (a) 所示。组装时将 4 个按键开关的一端接地，另一端分别接在 P2 端口的 P2.4 ~ P2.7 插针座上，组装后如图 25.27 (b) 所示。

5. 74 系列数字电路插座模块的制作

当进行串行输出、输入端口扩充实验时，需要使用 74S164 和 74S166 芯片，该插座为此而设计，如图 25.28 所示。

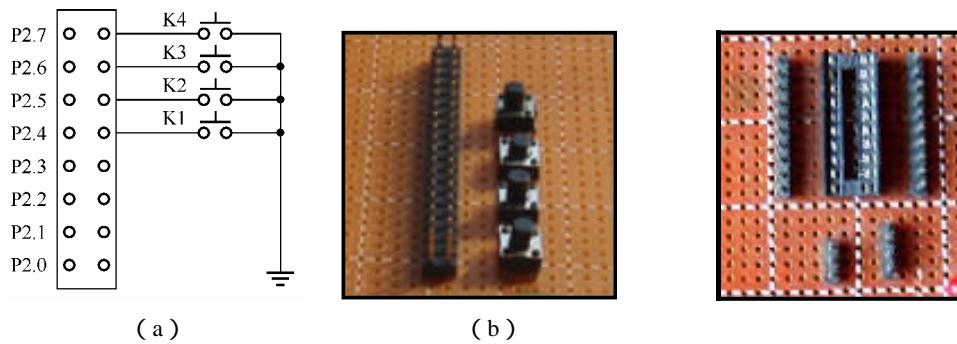


图 25.27 按键开关电路和组装图

图 25.28 插座模块

可采用 20 脚芯片插座，插座两边安装单排插针座以便将芯片引脚引出，方便与单片机端口相连。插座下方的 3 针单排插针座接电源负极，4 针的单排插针座接电源+5V，这样使用芯片时连接电源更加方便。

安装后主要注意检查是否有虚焊以及两个插针之间是否存在短路的地方。

6. 矩阵式键盘的制作

使用 16 个按钮开关组成 4×4 矩阵式键盘，键盘电路如图 25.29 所示，0 ~ 7 端接插针座上，组装后的矩阵式键盘如图 25.30 所示。

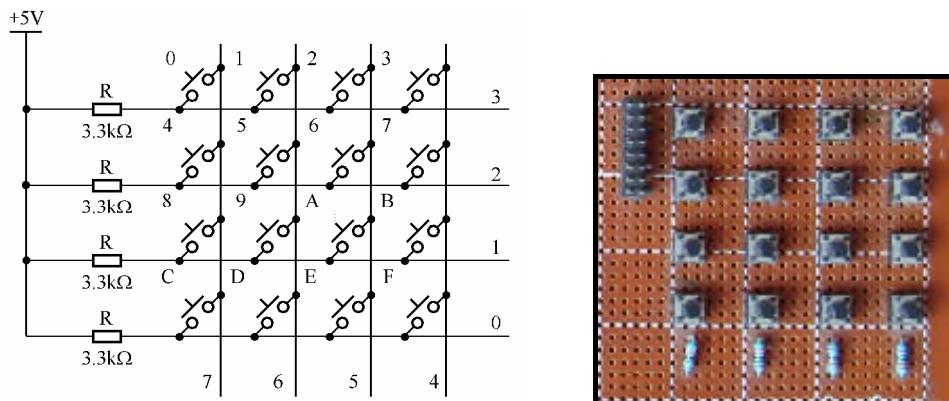


图 25.29 键盘电路原理图

图 25.30 矩阵式键盘

7. RS-232 串行接口模块的制作

RS-232 串行接口电路如图 25.31 所示，电路里 9015 为 PNP 型三极管，9014 为 NPN 型三极管，5 只电阻都为 $4.7\text{k}\Omega$ 。这些元件组装后如图 25.32 所示，测试时需要运行通信程序。

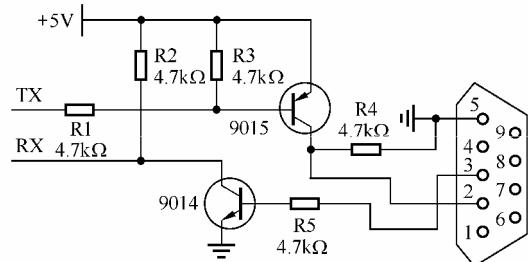


图 25.31 RS-232 接口电路

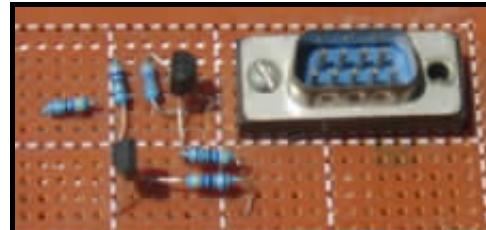


图 25.32 串行接口组装图

25.3.6 单片机端口插针座连接线

在单片机实验板上，单片机的端口与插针座两排插针中的一排相连，另一排插针与实验模块相接，所以，单片机端口与实验模块之间通过插针连接线连接，这样做的好处是使单片机的端口使用灵活。常用的连接方式有使用短路块、单针连接线、针排线等，如图 25.33 所示。

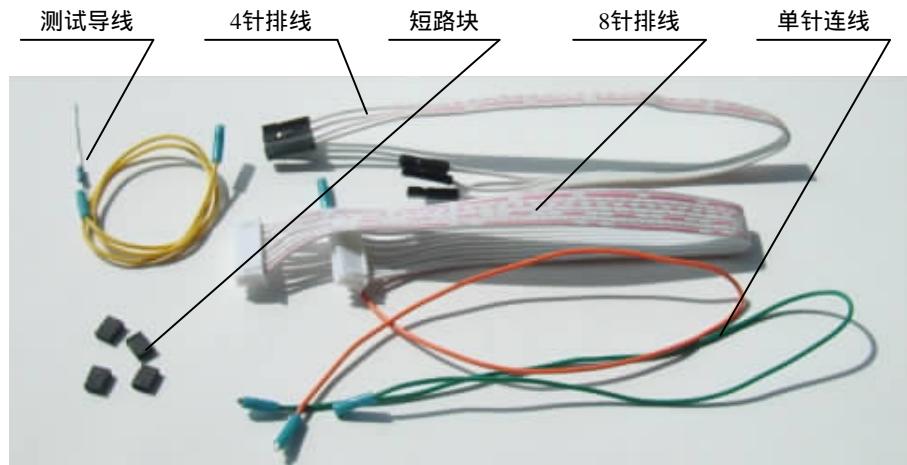


图 25.33 单片机端口插针座连接线

1. 短路块

在一条插针座上如果想把上面的两排插针相对连接，需要使用短路块，短路块插入后，会使两针短路，即连接在一起。

2. 单针连接线和针排线

如果单片机的端口连接插针与要连接的模块连接插针不在一个插座上，就需要较长的连

线，有时需要使用单针连线，如图 25.34 所示，有时需要使用针排线，如图 25.35 所示。

单针的连线制作是将一段导线的两端接上插芯，然后再用塑料管套上即可。针排线的制作可以用专用压线钳将一排导线（4 根或 8 根）压接在单排插针座上，也可以用简易的办法，将几根导线依次插入针座上，为了牢固可点少许粘合剂。

3. 测试导线

所谓测试导线，是将一段导线的一端插入 $1k\Omega$ 电阻即可，如图 25.36 所示，目的是在进行短路连接测试时防止由于误接而烧坏元件。



图 25.34 单针连线



图 25.35 制作针排线



图 25.36 测试导线



附录 A 80C51 单片机指令速查表

80C51 系列单片机的指令共有 111 条，并有 42 种助记符。依其功能可分为：算术运算类指令、逻辑运算及移位类指令、数据传送类指令、控制转移类指令和位数据传送类指令。为了学习时查找方便，现将各类指令列表说明，指令表中符号意义如表 A.1 所示。

表 A.1

指令符号意义说明

符 号	说 明
Rn	当前寄存器组的 8 个通用寄存器 R0 ~ R7 , n 为 0 ~ 7
Ri	用作间接寻址的寄存器，只能是 R0、R1 两个寄存器，i 为 0 或 1
direct	内部 RAM 的 8 位地址。既可以是内部 RAM 的低 128 个单元地址，也可以是专用寄存器的单元地址或符号。在指令中 direct 表示直接寻址方式
#data	8 位立即数
#data16	16 位立即数
addr16	16 位目的地址，只限于在 ACALL 和 AJMP 指令中使用
addr11	11 位目的地址，只限于在 ACALL 和 AJMP 指令中使用
rel	相对转移指令中的偏移量，为 8 位带符号补码数
DPTR	16 位数据指针
bit	内部 RAM (包括专用寄存器) 中的直接寻址位
A	累加器 (直接寻址方式的累加器表示为 ACC)
B	B 寄存器
C	进位标志位，是布尔处理机的累加器，也称之为累加位
@	间接寻址寄存器的前缀标志
/	加在位地址的前面，表示对该位状态取反
(×)	某寄存器或某单元的内容
((×))	由 × 间接寻址的单元的内容
	箭头左边的内容被箭头右边的内容所代替

表 A.2 数据传送类指令表

指令	功 能	功 能 说 明	机 械 周 期	备 注
MOV A, #data	A data	立即数送累加器	1	立即数传送 (5条)
MOV direct, #data	Direct data	立即数直接寻址单元	2	
MOV Rn, #data	Rn data	立即数送寄存器	1	
MOV @Ri, #data	(Ri) data	立即数送内部 RAM 单元	1	
MOV DPTR, #data16	DPTR data16	16位立即数送数据指针	2	
MOV direct2, direct1	Direct2 (direct1)	直接寻址单元送直接寻址单元	2	
MOV direct, Rn	Direct (Rn)	寄存器送直接寻址单元	2	
MOV Rn, direct	Rn (direct)	直接寻址单元送寄存器	2	
MOV direct, @Ri	Direct ((Ri))	内部 RAM 单元送直接寻址单元	2	
MOV @Ri, direct	(Ri) (direct)	直接寻址单元送内部 RAM 单元	2	
MOV , A, Rn	A (Rn)	寄存器送累加器	1	累加器的数据传送 (6条)
MOV Rn, A	Rn (a)	累加器送寄存器	1	
MOV A, direct	A (direct)	直接寻址单元送累加器	1	
MOV direct, A	direct (A)	累加器送直接寻址单元	1	
MOV A, @Ri	A ((Ri))	内部 RAM 单元送累加器	1	
MOV @Ri, A	(Ri) (A)	累加器送内部 RAM 单元	1	
MOV A, @Ri	A (Ri)	外部 RAM 单元送累加器 (8位地址)	2	外部 RAM 的 数据传送 (4条)
MOVX @Ri, A	(Ri) (A)	累加器送外部 RAM 单元 (8位地址)	2	
MOVX, A, @DPTR	A ((PDTR))	外部 RAM 单元送累加器 (16位地址)	2	
MOVX @DPTR, A	(DPTR) (A)	累加器送外部 RAM 单元 (16位地址)	2	
MOVC A, @A+DPTR	A ((A)+(DPTR))	查表数据送累加器 (数据指针为基础)	2	查表指令 (2条)
MOVC A, @A+PC	A ((A)+(PC))	查表数据送累加器 (程序计数器为基础)	2	
XCH A, Rn	(A)↔(Rn)	累加器与寄存器交换	1	交换类指令 (5条)
XCH A, @Ri	(A)↔((Ri))	累加器与内部 RAM 单元交换	1	

续表

指 令	功 能	功能说明	机械周期	备 注
XCH A, direct	(A) \leftrightarrow (direct)	累加器与直接寻址单元交换	1	
XCHD A, @Ri	(A) _{3~0} \leftrightarrow ((Ri)) _{3~0}	累加器与内部 RAM 低 4 位交换	1	
SWAP A	(A) _{3~0} \leftrightarrow (A) _{7~4}	累加器高4位与低4位交换	1	
POP direct	Direct ((SP)) SP (SP)-1	栈顶弹至直接寻址单元	2	堆栈操作指令 (2条)
PUSH direct	SP (SP)+1 (SP) (direct)	直接寻址单元压入栈顶	2	

表 A.3 控制转移类指令表

指 令	功 能	功能说明	机械周期	备 注
AJMP addr11	PC _{10~0} addr11	2KB 范围内绝对转移	2	无条件转移类 (4条)
LJMP addr16	PC addr16	64KB 范围内长转移	2	
SJMP rel	PC (PC)+rel	相对转移	2	
JMP @A+DPTR	PC (A)+(DPTR)	变址转移	2	
LCALL addr16	SP (SP)+1, (SP) (PC) _{7~0} SP (SP)+1, (SP) (PC) _{15~8} PC _{15~0} addr16	64KB 范围内长调用	2	调用及返回类 (4条)
ACALL addr11	SP (SP)+1, (SP) (PC) _{7~0} SP (SP)+1, (SP) (PC) _{15~8} PC _{10~0} addr11	2KB 范围内绝对调用	2	
RET	PC _{15~8} ((SP)), SP (SP)-1 PC _{7~0} ((SP)), SP (SP)-1	子程序返回	2	
RET1	PC _{15~8} ((SP)), SP (SP)-1 PC _{7~0} ((SP)), SP (SP)-1	中断返回	2	
JZ rel	若 (A) = 0, 则 PC (PC)+rel	累加器为 0 转移	2	判 A 转移 (2条)
JNZ rel	若 (A) ≠ 0, 则 PC (PC)+rel	累加器非 0 转移	2	
CJNE A, #data, rel	若左操作数 右操作数, 则 PC (PC)+rel 若左操作数=右操作数, 则程序顺序执行; 进位标志位 CY 的状态取决于左操作数—右操作数的状态, 但不保存相减的结果	累加器与立即数不等转移	2	比较转移指令, 影响 CY 的状态 (4条)
CJNE A, direct, rel		累加器与直接寻址单元不等转移	2	
CJNE Rn, #data, rel		寄存器与立即数不等转移	2	
CJNE @Ri, #data, rel		内部 RAM 单元与立即数不等转移	2	

续表

指令	功 能	功 能 说 明	机 械 周 期	备 注
DJNZ Rn, rel	Rn (Rn)-1; 若(Rn) 0, 则 PC (PC)+rel	寄存器减1不为0转移	2	(2条)
DJNZ, direct, rel	Direct (direct)-1; 若(direct) 0, 则 PC (PC)+rel	直接寻址单元减1不为0转移	2	
NOP	PC (PC)+1	空操作	1	(1条)

表 A.4 逻辑运算及移位类指令表

指令	功 能	功 能 说 明	机 械 周 期	备 注
ANL A, Rn	A (A) (Rn)	累加器“与”寄存器	1	按位“与”操作(6条)
ANL A, @Ri	A (A) ((Ri))	累加器“与”内部RAM单元	1	
ANL A, #data	A (A) data	累加器“与”立即数	1	
ANL A, direct	A (A) (direct)	累加器“与”直接寻址单元	1	
ANL direct, A	Direct (direct) (A)	直接寻址单元“与”累加器	1	
ANL direct, #data	Direct (direct) data	直接寻址单元“与”立即数	2	
ORL A, Rn	A (A) (Rn)	累加器“或”寄存器	1	按位“或”操作(6条)
ORL A, @Ri	A (A) ((Ri))	累加器“或”内部RAM单元	1	
ORL A, #data	A (A) data	累加器“或”立即数	1	
ORL A, direct	A (A) (direct)	累加器“或”直接寻址单元	1	
ORL direct, A	Direct (direct) (A)	直接寻址单元“或”累加器	1	
ORL direct, #data	Direct (direct) data	直接寻址单元“或”立即数	2	
XRL A, Rn	A (A)⊕(Rn)	累加器“异或”寄存器	1	按位“异或”操作(6条)
XRL A, @Ri	A (A)⊕((Ri))	累加器“异或”内部RAM单元	1	
XRL A, #data	A (A)⊕data	累加器“异或”立即数	1	
XRL A, direct	A (A)⊕(direct)	累加器“异或”直接寻址单元	1	
XRL direct, A	Direct (direct)⊕(A)	直接寻址单元“异或”累加器	1	
XRL direct, #data	Direct (direct)⊕data	直接寻址单元“异或”立即数	2	
RL A	An+1 An, A0 A7	累加器左环移位	1	移位操作(4条)
RLC A	An+1 An, CY A7, A0 CY	累加器连进位标志左环移位	1	
RR A	An An+1, A7 A0	累加器右环移位	1	
RRC A	An An+1, A7 CY, CY A0	累加器连进位标志右环移位	1	
CPL A	A (Ā)	累加器取反	1	(1条)
CPL A	A 0	累加器清0	1	(1条)

表 A.5 算术运算类指令表

指 令	功 能	功 能 说 明	机 械 周 期	备 注
ADD A, Rn	A (A)+(Rn)	累加器加寄存器	1	不带进位加法，影响 PSW (4条)
ADD A, @Ri	A (A)+(Ri)	累加器加内部 RAM 单元	1	
ADD A, direct	A (A)+(direct)	累加器加直接寻址单元	1	
ADD A, #data	A (A)+data	累加器加立即数	1	
ADDC, A, Rn	A (A)+(Rn)+(CY)	累加器加寄存器和进位标志	1	带进位加法，影响 PSW (4条)
ADDC A, @Ri	A (A)+(Ri)+(CY)	累加器加内部 RAM 单元和进位标志	1	
ADDC A, #data	A (A)+data+(CY)	累加器加立即数和进位标志	1	
ADDC A, direct	A (A)+(direct)+(CY)	累加器加直接寻址单元和进位标志	1	
DA A		十进制调整	1	加1指令，不影响 PSW (5条)
INC A	A (A)+1	累加器加1	1	
INC Rn	Rn (Rn)+1	寄存器加1	1	
INC direct	Direct (direct)+1	直接寻址单元加1	1	
INC @Ri	(Ri) ((Ri))+1	内部 RAM 单元加1	1	
INC DPTR	DPTR (DPTR)+1	数据指针加1	2	带链位减法，影响 PSW (4条)
SUBB A, Rn	A (A)-(Rn)-(CY)	累加器减寄存器和进位标志	1	
SUBB A, @Ri	A (A)-(Ri)-(CY)	累加器减内部 RAM 单元和进位标志	1	
SUBB A, #data	A (A)-data-(CY)	累加器减立即数和进位标志	1	
SUBB A, direct	A (A)-(direct)-(CY)	累加器减直接寻址单元和进位标志	1	减1指令，不影响 PSW (4条)
DEC A	A (A)-1	累加器减1	1	
DEC Rn	Rn (Rn)-1	寄存器减1	1	
DEC @Ri	(Ri) ((Ri))-1	内部 RAM 单元减1	1	
DEC direct	direct (direct)-1	直接寻址单元减1	1	乘法指令，影响 PSW (4条)
MUL AB	B(积高8位) A(积低8位) (A)×(B)	累加器乘寄存器 B	4	
DIV AB	B(余数) A(商) (A)/(B)	累加器除以寄存器 B	4	

表 A.6

位操作类指令表

指 令	功 能	功 能 说 明	机械周期	备 注
MOV C, bit	CY (bit)	直接寻址位送 C		位的置 / 复位 (4 条)
MOV bit, C	bit (CY)	C 送直接寻址位		
CLR C	CY 0	C 清 0		
CLR bit	bit 0	直接寻址位清 0		
SETB C	CY 1	设定进位标志为 1		
SETB bit	bti 1	设定 bit 为 1		
CPL C	CY (CY)	C 取反		
CPL bit	Bit (bit)	将 bit 反相		
ANL C, bit	CY (CY) (bit)	C 逻辑“与”直接寻址位		
ANL C, /bit	CY (CY) (bit)	C 逻辑“与”直接寻址位的反相		
ORL C, bit	CY (CY) (bit)	C 逻辑“或”直接寻址位		
ORL C, /bit	CY (CY) (bit)	C 逻辑“或”直接寻址位的反相		
JC rel	若 C=1 则 PC (PC)+rel	C 为 1 转移		
JNC rel	若 C=0 则 PC (PC)+rel	C 为 0 转移		
JB bit, rel	若 bit=1 则 PC (PC)+rel	直接寻址位为 1 转移		
JNB bit, rel	若 bit=0 则 PC (PC)+rel	直接寻址位为 0 转移		
JBC bit, rel	若 bit=1 则 bit 0, PC (PC)+rel	直接寻址位为 1 转移并清除该位		



附录 B 数的制式转换表

表 B .1

十进制数、二进制数及十六进制数对照表

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

使用说明：

例一：将二进制数 00111100B 转换成十六进制数，查表可知，0011 对应的十六进制数为 3；1100 对应的十六进制数为 C，因此 00111100B=3CH。

例二：将十六进制数 4DH 转换成二进制数，查表可知，十六进制数 4 所对应的二进制数为 0100B；十六进制数 D 所对应的二进制数为 1101B，因此 4DH=01001101B。

表 B .2

二进制转换十进制表

	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位	第 0 位
二进制位	1	1	1	1	1	1	1	1
2 的幂	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
十进制	128	64	32	16	8	4	2	1

使用说明：

例如，将 10010111B 转换成十进制数，对照上表可知第 7 位、第 4 位、第 2 位、第 1 位和第 0 位为 1，因此 $128+16+4+2+1=151$ 。



附录 C 光盘使用说明

本光盘中包括了书中所有示例的源代码，具体说明如下。

示例源程序及在 Keil 环境下编译生成的所有文件，均按章节顺序存放。例如：Chapter06 文件夹中包含了第 6 章的示例源程序，Chapter06 下的 6-1 文件夹内为第 6 章第 1 节的示例源程序及在 Keil 环境下编译生成的所有文件。其中，文件夹内的 Word 文档为书中本节示例程序代码。

Chapter06 ~ Chapter15 文件夹中包含了第 6 章 ~ 第 15 章汇编语言实例源代码以及编译生成的所有文件。

Chapter18 ~ Chapter22 文件夹中包含了第 18 章 ~ 第 22 章单片机 C 语言实例源代码以及编译生成的所有文件。

Chapter23 ~ Chapter24 文件夹中包含了第 23 章 ~ 第 24 章汇编语言、单片机 C 语言实例源代码以及编译生成的所有文件。

读者只需在 Keil 环境下打开该工程项目文件，进行模拟仿真便可以看到例子中的运行结果，文件夹中的.HEX 文件可以直接写入单片机进行实验。

另外，为了使读者使用本书方便，光盘中还附有本书所有原理图、电路图。

光盘中资料，仅供学习本书使用。