

从零开始学电子技术丛书

从零开始学 单片机技术

刘建清 鲁金 王春生 主编
刘建清 鲁金 王春生 编著



随书附光盘一张



国防工业出版社

National Defense Industry Press



责任编辑：杨星豪
文字编辑：吴晓涛
责任校对：钱辉玲
封面设计：王晓军

xhyang@ndip.cn

xjwang@ndip.cn

从**零**开始学电子技术丛书

从零开始学电路仿真Multisim与电路设计Protel技术

从零开始学电气控制与PLC技术

从零开始学电子测量技术

从零开始学CPLD和Verilog HDL编程技术

从零开始学单片机C语言

从零开始学单片机技术

从零开始学电路基础

从零开始学元器件识别与检测

从零开始学电动机控制与维修

从零开始学模拟电子技术

从零开始学数字电子技术

◎ 上架建议：电子技术 ◎

<http://www.ndip.cn>

ISBN 7-118-04599-3



9 787118 045994 >



ISBN 7-118-04599-3/TP · 1058

定价：39.00 元（含光盘）

从零开始学电子技术丛书

从零开始学单片机技术

刘建清 主编

刘建清 鲁金 王春生 编著

国防工业出版社

·北京·



内 容 简 介

本书以实战操作为基础,用最简单的方法,将初学者领进单片机世界的大门。本书首先介绍了单片机的基础知识,然后简要阐述了单片机的指令系统、简单程序设计、存储器和 I/O 接口扩展、中断、定时器以及串行通信技术,最后对单片机常用接口技术(LED 显示接口、键盘接口、LCD 液晶接口、SPI 总线接口和看门狗电路、I²C 总线接口、红外遥控接口、语音接口、A/D 和 D/A 转换接口)和单片机的应用系统设计进行了精要介绍。

本书附赠光盘一张,光盘中包含有 Keil、MedWin 中文版软件以及书中所有实验的源程序。本书中提供的所有实验均具有较高的实用性和代表性,且全部通过了实验板验证。

本书可供电子技术人员、单片机爱好者、业余电子爱好者阅读,也可作为相关专业的教学用书或教学参考书使用。

图书在版编目(CIP)数据

从零开始学单片机技术/刘建清主编. —北京:国防工业出版社,2006.8

ISBN 7-118-04599-3

I. 从... II. 刘... III. 单片微型计算机
IV. TP368.1

中国版本图书馆 CIP 数据核字(2006)第 069348 号

※

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号 邮政编码 100044)
北京奥鑫印刷厂印刷
新华书店经售

*

开本 787×1092 1/16 印张 23 字数 528 千字
2006 年 8 月第 1 版第 1 次印刷 印数 1—5000 册 定价 39.00 元(含光盘)

(本书如有印装错误,我社负责调换)

国防书店:(010)68428422
发行传真:(010)68411535

发行邮购:(010)68414474
发行业务:(010)68472764

丛书前言

我们所处的时代是一个知识爆炸的新时代。新产品、新技术层出不穷,电子技术的发展更是日新月异。可以毫不夸张地说,电子技术的应用无处不在,电子技术正在不断地改变着我们的生活,改变着我们的世界。

读者朋友:当你为妙趣横生的电子世界发生兴趣时;当你徘徊于就业的关口,想成为电子产业中的一名员工时;当你跃跃欲试,想成为一名工厂的技术革新能手时;当你面对“无所不能”的“单片机”,梦想成为一名自动化高手时;当你的头脑里冒出那么多的奇思妙想,急于把它们应用于或转化为产品时……都是那么急切地想补充自己有关电子技术方面的知识,这时,你首先想到的是找一套适合自己学习的电子技术图书阅读。《从零开始学电子技术丛书》正是为了满足广大读者特别是电子爱好者的实际需要和零起点入门的阅读要求而编著的。

和其他电子技术类图书相比,本丛书具有以下特点:

内容全面,体系完备。本丛书给出了广大电子爱好者学习电子技术的全方位解决方案,既有初学者必须掌握的电路基础、模拟电路和数字电路等基础理论,又有电子元器件检测、电子测量仪器的使用、电路仿真与设计等操作性较强的内容,还有电气控制与PLC、单片机、CPLD等综合应用方面的知识,因此,本丛书内容翔实,覆盖面广。

通俗易懂,重点突出。传统的电子技术图书和教材在介绍电路基础和模拟电子技术等内容时,大都借助高等数学这一工具进行分析,这就给电子爱好者自学电子技术设置了一道门槛,使大多数电子爱好者失去了学习的热情和兴趣。本丛书在编写时,完全考虑到了初学者的需要,不涉及高等数学方面的公式,尽可能地把复杂的理论通俗化和实用化,将烦琐的公式简易化,再辅以简明的分析及典型的实例,从而形成了本丛书通俗易懂的特点。为了满足不同层次读者的需求,本丛书对难点和扩展知识用“*”进行了标注,初学者可跳过此内容。

实例典型,实践性强。本丛书最大程度地强调了实践性,书中给出的例子大都经过了验证,可以实现,并且具有代表性;本丛书中每本书都配有光盘,光盘中收录了书中的实例、常用软件、实验程序和大量珍贵资料,以方便读者学习和使用。

内容新颖,风格活泼。本丛书所介绍的都是电子爱好者最为关心并且在业界获得普遍认同的内容,本丛书的每一分册都各有侧重,又互相补充,论述时疏密结合,重点突出。对于重点、难点和容易混淆的知识,书中还特别进行了标注和提示。

把握新知,结合实际。电子技术发展日新月异,为适应时代的发展,本丛书还对电子技术的新知识做了详细的介绍;本丛书中涉及的应用实例都是编著者开发经验的提炼和总结,相信一定会给读者带来很大的帮助。在讲述电路基础、模拟和数字电子技术时,还

专门安排了计算机辅助软件的仿真实验,实验过程非常接近实际操作的效果,使电子技术的学习变得更为直观,使学习变得更加生动有趣,这可以加深读者对电路理论知识的认识。

总之,对于需要学习电子技术的电子爱好者而言,选择《从零开始学电子技术丛书》不失为一个好的选择。本丛书一定能给你耳目一新的感觉,当你认真阅读之后将会发现,无论是你所读的书,还是读完书的你,都有所不同。

感谢本丛书的策划者——电子科普领域中的知名专家、中国电子学会高级会员刘午平先生,他与我们共同交流,共同探讨,达成了共识,确立了写作方向,并为本丛书的编排、修改和出版做了大量卓有成效的工作,他以丰富的专业知识和认真、敬业的态度为我们所敬佩;感谢山东持恒开关厂总经理陈培军先生和山东金曼克电气集团设计处总工程师高广海先生,他们对本丛书的编写提出了很多建设性的意见和建议,为本丛书的许多实验提供了强有力的支持与帮助,并参与了部分图书的编写工作;感谢网络,本丛书的许多新知识、新内容都是我们通过网络而获得的,我们在写作过程中遇到的许多疑难问题也大都通过网络得以顺利解决,对于这么多乐于助人、无私奉献的站主和作者们,无法在此一一列举,只能道一声“谢谢了!”感谢众多电子报刊、杂志的编辑和作者,他们为本丛书提供了许多有新意、有实用价值的参考文献,使得这套丛书能够别出心裁、与时俱进;感谢国防工业出版社,能与国内一流的出版社合作,我们感到万分的荣幸;感谢其他对本丛书的出版付出过辛勤工作的人士,没有他们的热心与支持,本丛书不知何时才能与读者见面!

最后,祝愿本丛书的每一位读者在学习电子技术的过程中,扬起风帆,乘风破浪!

丛书编者



前 言

单片机就是把一个计算机系统集成到一个芯片上,概括地讲,一块芯片就成了一台计算机。目前市面上流行的单片机,其价格是出奇的便宜,对于广大爱好者来说,真是上帝赐予的礼物。对于一般人来讲,单片机似乎很神秘,其实不然。一般的了解和应用也不难,从小学生到中学生再到大学生,从一般技术人员到工程师再到高级工程师,都能学能用,只是深浅程度不同而已。只要你玩起了单片机,你就会有一种成就感,我怎么这样聪明!从而对单片机的应用和开发产生浓厚的兴趣。坚持努力下去,说不定哪一天就能开发出一个智能产品。单片机,再结合适当的硬件接口电路,有什么事情做不到呢?对它的评价是8个字:软硬兼施,老少皆宜。

本书编写的宗旨是,以实战操作为基础,用最简单的方法,将初学者领进单片机世界的大门,使仅稍懂硬件原理的人通过实践能理解软件的作用,让他们知道在单片机组成的系统中,硬件与软件的区分并不绝对,硬件能做的工作,一般情况下软件也能完成,软件的功能也可用硬件替代。使习惯于传统电路设计的人对单片机产生一种妙不可言的相见恨晚之感,体会到单片机的强大作用,从而投身于单片机的应用领域中。

作为入门,本书首先介绍了单片机的基础知识,然后简要阐述了单片机的指令系统、简单程序设计、存储器和I/O接口扩展、中断、定时器和串行通信技术,最后对单片机常用接口技术(LED显示接口、键盘接口、LCD液晶接口、SPI总线接口和看门狗电路、I²C总线接口、红外遥控接口、语音接口、A/D和D/A转换接口)和单片机的应用系统设计进行了精要介绍,尤其珍贵的是,本书中的所有实验均具有较好的实用性和代表性,且全部通过了实验板验证。

本书附赠光盘一张,光盘中包含有Keil、MedWin中文版软件以及书中所有实验的源程序。

由于时间仓促,书中错漏之处在所难免,敬请广大读者批评指正。

作者
2006年6月

目 录

第一章 单片机入门.....	1
第一节 单片机基础知识.....	1
一、数制	1
二、数制的转换	2
三、二进制的算术运算	5
四、编码	6
五、存储器基础知识	8
第二节 单片机概述	11
一、什么是单片机.....	11
二、单片机名称的由来.....	12
三、单片机与单片机系统.....	12
第三节 单片机的分类、发展及应用.....	13
一、单片机的分类.....	13
二、单片机发展的历史.....	14
三、单片机技术发展的特点.....	15
四、MCS-51 单片机家族简介	16
五、单片机的应用.....	18
第四节 到单片机世界去遨游	19
一、如何学习单片机.....	19
二、单片机的开发步骤.....	24
第二章 单片机的组成	28
第一节 80C51 单片机的内部结构和外部引脚	28
一、80C51 单片机的内部结构框图	28
二、单片机的外部引脚.....	30
第二节 80C51 单片机内部存储器的配置	32
一、程序存储器.....	32
二、数据存储器.....	33
第三节 80C51 单片机的并行 I/O 接口	38
一、P0 口	39
二、P1 口	40
三、P2 口	41
四、P3 口	42

第四节	80C51 单片机的时钟电路和复位电路	43
一、	单片机的时钟电路.....	43
二、	单片机的复位电路.....	44
三、	单片机的低功耗方式.....	45
第五节	AT89C51 和 AT89C2051/1051 简介	47
一、	AT89C51 简介	47
二、	AT89C2051/1051 简介	47
第三章	单片机实验软件和硬件环境的建立	52
第一节	单片机仿真软件 Keil C51 的使用	52
一、	Keil C51 软件的启动.....	52
二、	建立一个新工程.....	53
三、	工程的设置.....	58
四、	程序的编译和链接.....	62
五、	程序调试.....	63
六、	常用窗口介绍.....	66
第二节	用单片机实验板进行仿真实验	71
一、	AT89C51 单片机实验开发板	71
二、	MON51 仿真器	75
三、	Insight SE-52 仿真器	81
第三节	编程器及其使用方法	83
一、	RF-810 编程器简介	84
二、	RF-810 编程器的安装	84
三、	RF-810 编程软件的功能	84
四、	RF-810 编程软件的使用	85
第四节	下载型实验板简介	90
一、	硬件结构.....	91
二、	使用简介.....	94
三、	下载型编程器的使用.....	95
第四章	单片机的指令系统	97
第一节	指令概述	97
一、	指令与指令系统.....	97
二、	指令的格式.....	98
三、	指令的字节数.....	98
四、	指令的寻址方式	100
第二节	MCS-51 单片机指令分类介绍	106
一、	数据传送类指令(28 条)	106
二、	算术运算类指令(24 条)	112
三、	逻辑运算及移位类指令(25 条)	117
四、	控制转移类指令(17 条)	121

五、位操作类指令(17 条)	126
第三节 使用 I/O 访问指令应注意的问题	129
一、可对口进行操作的指令	129
二、读引脚与读锁存器	130
第四节 指令上机练习	132
第五章 汇编语言简单程序设计	135
第一节 概述	135
一、程序与语言	135
二、汇编语言源程序的格式	137
三、51 汇编软件汇编失败原因	137
第二节 汇编语言的伪指令	138
一、汇编起始地址伪指令 ORG	138
二、汇编结束伪指令 END	139
三、定义字节伪指令 DB	139
四、定义字伪指令 DW	139
五、定义内存空间伪指令 DS	139
六、赋值伪指令 EQU	140
七、位地址定义伪指令 BIT	140
第三节 汇编语言典型程序结构	140
一、顺序结构程序	140
二、分支结构程序	141
三、循环结构程序	143
第四节 汇编语言子程序的设计	145
一、子程序调用过程中参数的传递	145
二、调用子程序时的现场保护问题	146
第五节 汇编语言实用程序举例	148
一、定时程序	148
二、查表程序	150
第六节 几个简单的实验	154
一、闪烁的发光管	154
二、8 路流水灯	156
三、用按键控制发光管	157
四、二进制加法运算	158
五、加 1 计数器	159
六、二进制乘法运算	160
七、逻辑运算	161
第六章 中断、定时/计数器和串行数据通信	163
第一节 中断系统及实验	163
一、中断概述	163

二、中断源	164
三、中断控制	164
四、中断的响应	167
五、中断的撤除	170
六、中断程序设计及实验	171
第二节 定时/计数器及实验	175
一、定时/计数器概述	175
二、定时/计数器的控制寄存器	177
三、定时/计数器的工作方式	178
四、定时/计数器应用举例及实验	182
第三节 串行数据通信技术及实验	193
一、串行数据通信概述	193
二、串行口的基本结构	196
三、串行通信控制寄存器	197
四、串行口工作方式	201
五、串行数据通信应用举例	204
第七章 单片机存储器和 I/O 接口的扩展	217
第一节 系统扩展概述	217
一、地址总线(AB)	217
二、数据总线(DB)	218
三、控制总线(CB)	218
第二节 存储器的扩展	219
一、程序存储器的扩展	219
二、数据存储器的扩展	226
第三节 I/O 接口的扩展	228
一、I/O 接口扩展概述	228
二、I/O 接口扩展举例	229
第八章 单片机实用接口技术	232
第一节 LED 显示器接口	232
一、8 段 LED 显示器的结构及原理	232
二、LED 显示器的显示方式	234
三、几种常见的显示接口电路	236
四、LED 显示器接口实验	244
第二节 键盘接口	249
一、键盘的工作原理	249
二、键盘与单片机的连接	251
三、键盘的工作方式	259
四、键盘接口实验	260
第三节 LCD 显示器接口	262

一、字符型液晶显示器概述	263
二、字符显示模块内部结构	264
三、字符型液晶控制器的指令	267
四、字符显示实验	270
五、汉字图形的显示原理与实验	275
第四节 SPI 总线接口和看门狗电路	279
一、SPI 总线接口	279
二、看门狗电路	280
第五节 I ² C 总线接口	288
一、I ² C 总线及其软件包	288
二、I ² C 总线串行存储器 AT24Cxx	291
三、I ² C 总线时钟芯片 PCF8563	300
第六节 红外遥控接口	311
一、红外遥控系统	311
二、红外遥控的编码与解码	312
三、红外遥控实验	313
第七节 语音接口	316
一、ISD1400 系列语音电路	316
二、ZY1420A 语音电路	322
第八节 A/D 和 D/A 转换接口	326
一、A/D 转换接口	326
二、D/A 转换接口	328
第九章 单片机应用系统设计	331
第一节 单片机应用系统设计的原则	331
一、确定任务	331
二、总体设计	331
三、硬件设计	331
四、软件设计	332
五、系统调试	332
第二节 单片机电子钟应用系统的设计	333
一、系统软件、硬件功能的划分	333
二、硬件设计	333
三、软件设计	335
四、用下载型实验板进行仿真实验	345
五、系统功能的扩充	345
第三节 单片机应用系统的可靠性设计	345
一、提高单片机系统稳定性的硬件措施	345
二、提高单片机系统稳定性的软件措施	349
附录 MCS51 指令表汇总	351

第一章 单片机入门

单片机又称单片微控制器,它不是完成某一个逻辑功能的芯片,而是把一个计算机系统集成到一个芯片上。概括地讲,一块芯片就成了一台计算机。它的体积小、质量轻、价格便宜,为学习、应用和开发提供了便利条件。同时,也为了解计算机原理与结构提供了最佳选择。随着电子技术的迅速发展,单片机已深入地渗透到我们的生活中,许多电子爱好者开始学习单片机知识,但单片机的内容比较抽象,相对电子爱好者已熟悉的模拟电路、数字电路,单片机中不但出现了一些新概念、新内容,而且具有较强的实践性。作为入门,本章首先介绍单片机的基础知识,然后简要阐述单片机的概念、发展及分类等内容,最后对单片机的开发步骤进行精要介绍。

第一节 单片机基础知识

一、数制

人们在日常生活中,习惯于用十进制,而在数字系统中,例如在计算机或单片机中,多采用二进制,有时也采用八进制或十六进制。

数制所使用的数码的个数为基,数制的每一位所具有的值称为权。

1. 十进制(D)

十进制是以 10 为基数的计数体制。即它所使用的代码为 0、1、2、3、4、5、6、7、8、9,共有 10 个。十进制的权是以 10 为底的幂,如 $10 = 1 \times 10^1 + 0 \times 10^0$ 。每个十进制数都可以用位权值表示,其中:个位的位权为 10^0 ,十位的位权为 10^1 ,百位的位权为 10^2 ,依此类推。例如:

$$402.8 = 4 \times 10^2 + 0 \times 10^1 + 2 \times 10^0 + 8 \times 10^{-1}$$

从数字电路的角度来看,采用十进制是不方便的,因为构成数字电路的基本思路是把电路的状态与数码对应起来,而十进制的 10 个数码,必须有 10 个不同的而且能严格区分的电路状态与之对应起来,这样将在技术上带来许多困难,而且不经济,因而在数字电路中一般不直接采用十进制,而采用二进制。

2. 二进制(B)

二进制是以 2 为基数的计数体制。二进制与十进制的区别在于数码的个数和进位规律不同。二进制是用两个数码 0 和 1 表示,而且是“逢二进一”,即 $1 + 1 = 10$ (读为“壹零”)。

注意 这里的“10”与十进制数的“10”是完全不同的,它不代表“十”,右边的“0”表示 0 个 2^0 ,左边的“1”表示 1 个 2^1 ,即 $10 = 1 \times 2^1 + 0 \times 2^0$,可见,二进制的权是以 2 为底的幂。

二进制的数字装置简单可靠,应用元件少;二进制只有两个数码 0 和 1,因此,它的每一位都用任何具有两个不同稳定状态的元件来表示,如晶体管的饱和和截止,继电器接点的闭合和断开,灯泡的亮和不亮等。只要规定一种状态表示 1,另一种状态表示 0,就可以表示二进制数。这样,数码的存储、分析和传输,就可以用最简单而可靠的方式进行。另外,二进制的基本运算规则简单,运算操作也比较方便。

但是,二进制也有一些缺点。比如,用二进制表示一个数时,位数多,使用起来不方便也不习惯。因此在运算时,原始数据多用人们习惯的十进制,在送入计算机时,就必须将十进制数据转换成数字系统能接受的二进制数,而运算结束后再将二进制数转换为十进制数,表示最终结果。

3. 八进制(O)和十六进制(H)

由于使用二进制数位数很多,不便于书写和记忆,因此在数字计算机的资料中常采用十六进制数或八进制数来表示二进制数。上述十进制和二进制的表示法可以推广到十六进制和八进制。例如,八进制数采用 8 个数码,而且“逢八进一”。这种数制中有 8 个不同的数字,即 0、1、2、3、4、5、6、7,它是以 8 为基数的计数体制。十六进制数采用 16 个数码,而且“逢十六进一”。这种数制中有 16 个不同的数字,即 0、1、2、3、4、5、6、7、8、9、A(对应于十进制数中的 10)、B(11)、C(12)、D(13)、E(14)、F(15)。它是以 16 为基数的计数体制。

为便于记忆和理解,表 1-1 对二进制、八进制、十进制和十六进制进行了对照和比较。

表 1-1 二进制、八进制、十进制和十六进制对照比较表

常用进制	英文表示符号	数码符号	进位规律	进位基数
二进制	B	0、1	逢二进一	2
八进制	O	0、1、2、3、4、5、6、7	逢八进一	8
十进制	D	0、1、2、3、4、5、6、7、8、9	逢十进一	10
十六进制	H	0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F	逢十六进一	16

二、数制的转换

1. 其他进制转换为十进制

其他进制转换为十进制方法是,将其他进制按权位展开,然后各项相加,就得到相应的十进制数。

例 1 将二进制数(11011.101)₂转换成十进制数。

解 $(11011.101)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 = (27.625)_{10}$

例 2 将八进制数(136.524)₈转换成十进制数。

解 $(136.524)_8 = 1 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 5 \times 8^{-1} + 2 \times 8^{-2} + 4 \times 8^{-3} = 64 + 24 + 6 + 0.625 + 0.03125 + 0.0078125 = (94.6640625)_{10}$

例 3 将十六进制 $(FF)_{16}$ 转换为十进制。

解 $(FF)_{16} = 15 \times 16^1 + 15 \times 16^0 = 240 + 15 = (255)_{10}$

例 4 将十六进制数 $(13DF.B8)_{16} [= (13DF.B8)_H]$ 转换成十进制数。

解 $(13DF.B8)_{16} = 1 \times 16^3 + 3 \times 16^2 + 13 \times 16^1 + 15 \times 16^0 + 11 \times 16^{-1} + 8 \times 16^{-2} = 4096 + 768 + 208 + 15 + 0.6875 + 0.03125 = (5087.71875)_{10}$

2. 将十进制转换成其他进制

十进制转换成其他进制时,需分两部分进行,即整数部分和小数部分。

(1) 整数部分(基数除法)。把要转换的十进制数除以新的进制的基数,把余数作为新进制的最低位;把上一次得的商在除以新的进制基数,把余数作为新进制的次低位;继续上一步,直到最后的商为零,这时的余数就是新进制的最高位。

(2) 小数部分(基数乘法)。把要转换数的小数部分乘以新进制的基数,把得到的整数部分作为新进制小数部分的最高位;把上一步得的小数部分再乘以新进制的基数,把整数部分作为新进制小数部分的次高位;继续上一步,直到小数部分变成零为止。或者达到预定的要求也可以。

例 5 把十进制数 $(25)_{10}$ 转换成二进制数。

解 由于二进制基数为 2,所以逐次除以 2 取其余数(0 或 1),即

2	25	
2	12	……余 1
2	6	……余 0
2	3	……余 0
2	1	……余 1
0		……余 1

所以, $(25)_{10} = (11001)_2$, 即 $25D = 11001B$ 。

例 6 把十进制数 $173D$ 转换成二进制数。

解 由于二进制基数为 2,所以逐次除以 2 取其余数(0 或 1),即

2	173	
2	86	……余 1
2	43	……余 0
2	21	……余 1
2	10	……余 1
2	5	……余 0
2	2	……余 1
2	1	……余 0
0		……余 1

所以, $(173)_{10} = (10101101)_2$ 。

例 7 将十进制数 $(25)_{10}$ 转换成八进制数。

解 由于基数为 8, 逐次除以 8 取余数, 即

8		25	
8		3余 1
		0余 3

所以, $(25)_{10} = (31)_8$ 。

例 8 将十进制 $(64536)_{10}$ 转换为十六进制数。

解 由于基数为 16, 逐次除以 16 取余数, 即

16		64536	
16		4033余 8
16		252余 1
16		15余 C
16		0余 F

所以, $(64536)_{10} = (FC18)_{16}$ 。

例 9 将十进制小数 $(0.375)_{10}$ 转换成二进制数。

解 由于二进制基数为 2, 所以逐次乘以 2 取整数部分(0 或 1), 即

	0.375
×	2
<hr/>	
	[0].750
	2
<hr/>	
	[1].500
	2
<hr/>	
	[1].000

所以, $(0.375)_{10} = (0.011)_2$ 。

3. 二进制与八进制、十六进制的相互转换

八进制数和十六进制数的基数分别为 $8=2^3$ 、 $16=2^4$, 所以 3 位二进制数恰好相当一位八进制数, 4 位二进制数相当一位十六进制数, 它们之间的相互转换是很方便的。

二进制数转换成八进制数的方法是, 从小数点开始, 分别向左、向右, 将二进制数按每三位一组分组(不足三位的补 0), 然后写出每一组等值的八进制数。

例 10 求 $(01101111010.1011)_2$ 的等值八进制数。

解 二进制 001 101 111 010. 101 100

八进制 1 5 7 2. 5 4

所以 $(01101111010.1011)_2 = (1572.54)_8$

二进制数转换成十六进制数的方法和二进制数与八进制数的转换相似,从小数点开始分别向左、向右将二进制数按每 4 位一组分组(不足四位补 0),然后写出每一组等值的十六进制数。

例 11 将 $(1101101011.101)_2$ 转换为十六进制数。

解 二进制 0011 0110 1011 . 1010
十六进制 3 6 B . A

所以 $(1101101011.101)_2=(36B.A)_{16}$

八进制数、十六进制数转换为二进制数的方法可以采用与前面相反的步骤,即只要按原来顺序将每一位八进制数(或十六进制数)用相应的 3 位(或 4 位)二进制数代替即可。

例 12 将 $(678.A5)_{16}$ 转换为二进制数。

解 十六进制 6 7 8 . A 5
二进制 0110 0111 1000.1010 0101

所以, $(678.A5)_{16}=(011001111000.10100101)_2$

三、二进制的算术运算

二进制数的加、减、乘、除四则运算,在数字系统中是经常遇到的,它们的运算规则与十进制数很相似。加法运算是最基本的一种运算,利用它的运算规则可以实现其他 3 种运算。例如,减法运算可以借助改变减数的符号再与被减数相加,乘法运算可视为被乘数的连加,而除法则可视为被除数重复地减去除数。

1. 二进制加法

加法运算的规则是“逢 2 进 1”,如表 1-2 所列。

表 1-2 加法运算规则

					1
被加数	0	0	1	1	1
加 数	<u>+0</u>	<u>+1</u>	<u>+0</u>	<u>+1</u>	<u>+1</u>
和	0	1	1	10	11

2. 二进制减法

二进制减法运算的规则是“向高位借 1 当 2”,如表 1-3 所列。

表 1-3 无符号数的减法运算规则

	无借位	无借位	无借位	有借位
被减数	0	1	1	1 0
减 数	<u>-0</u>	<u>-0</u>	<u>-1</u>	<u>-1</u>
差	0	1	0	1

3. 二进制乘法与除法

二进制乘法、除法与十进制乘法、除法相同,下面列出了 4 条乘法规则:

$0 \times 0 = 0$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

例 13 求 $(1101)_2$ 与 $(0101)_2$ 的乘积。

解

$$\begin{array}{r}
 1101 \\
 \times 0101 \\
 \hline
 1101 \\
 0000 \\
 1101 \\
 0000 \\
 \hline
 1000001
 \end{array}$$

四、编码

数字系统中的信息可分为两类，一类是数值，另一类是文字符号(包括控制符)。为了表示文字符号信息，往往也采用一定位数的二进制码表示，这个特定的二进制码称为代码。建立代码与十进制数、字母、符号的一一对应关系的方法称为编码。

1. 码的基本概念

(1) 数位与比特。

码的位叫做数位，对于十进制码叫做十进制数位，对于二进制码叫做二进制数位，二进制数位一般用比特(bit)表示，简写为 b。例如，某一个二进制码是 100101，该码共有 6 数位，所以称为 6bit(比特)。

(2) 字。用二进制数表示某一个数值或字符时，该二进制数称为字(Word)。在数字系统电路中，所有的信息，包括数据、字母、符号、代表机器操作的指令或数据以及指令在存储器中的存放地址等，都是以二进制代码表示的，作为一个整体来处理或运算的一组二进制数码，称为一个字。字是二进制数的基本单位，是数据总线宽度。

(3) 字长。在单片机中，一个字的二进制位数称之为字长。单片机的字长有 1 位、4 位、8 位和 16 位等，一般来说，字长越长，单片机的性能越好。例如，8051 单片机是 8 位机，是指它的字长是 8 位，其内容的运算器都是 8 位的，每次参加运算的二进制位只有 8 位。

(4) 字节。字节是由一组二进制位形成的计算机的一种存储单位，它可以表示一个字符，通常一个 8 位二进制数称为一个字节(Byte)，用 B 表示。

字节 B 是一个比较小的单位，常用的还有 KB 和 MB，它们的关系是：

$$1\text{KB}=1024\text{B}$$

$$1\text{MB}=1024\text{KB}=1024 \times 1024\text{B}$$

(5) 字内位的名称。字内各个位的名称是有规定的，具体规定如下：

最高一位叫做 MSB，次高位叫做 2SB，第三位叫做 3SB，以此类推，最后一位叫做 LSB。

2. 二-十进制编码(BCD 码)

二-十进制编码也称 BCD 码,它是一个用 4 比特二进制码来表示十进制数的二进制码,即用 4 数位的二进制码来表示十进制数中的 0~9。表 1-4 列出了几种常用 BCD 码编码方式。

表 1-4 几种常用的 BCD 码

十进制数	8421 码	5421 码	2421 码	余 3 码
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0010	0101
3	0011	0011	0011	0110
4	0100	0100	0100	0111
5	0101	1000	1011	1000
6	0110	1001	1100	1001
7	0111	1010	1101	1010
8	1000	1011	1110	1011
9	1001	1100	1111	1100

1)8421 BCD 码

8421 BCD 码是最基本和最常用的 BCD 码,从表(1-4)可看出,十进制数中的 0~9 十个数每个数都用一个 4 比特的二进制码表示,而十进制数中的两位及两位以上的数,则采用每一位都用一个 4 比特的二进制码来表示。例如:123 用 8421 BCD 码表示就是 0001 0010 0011,其中 0001 是表示百位数的 1,0010 表示的是十位数的 2,0011 表示的是个位数 3。

8421 BCD 码的权自左至右为 8、4、2、1。具体地讲,8 是最高位(第 4 位)的权,4 是次高位(第 3 位)的权,2 是第 2 位的权,1 是最低位的权。

根据每一位的权,可以方便地计算出十进制数,例如某一个二进制数码是 0111,则该数码就是十进制数中的 $0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 7$ 。

2)其他 BCD 码

在 5421 码中,5、4、2、1 是这种编码的权,即最高一位的权是 5,次高位的权是 4,第 3 位的权是 2,最低一位的权是 1。

在 2421 码中,2、4、2、1 是这种编码的权,即最高一位的权是 2,次高位的权是 4,第 3 位的权是 2,最低一位的权是 1。

用 4 位二进制码中的十组代码为 0011~1100 来表示十进制中 0~9 十个数,同一个十进制数所对应的余三码等于所对应的 8421 码加上 3(0011)。余三码这种编码是无权的。

3. ASCII 码

在数字系统中,除数字需要编码成为二进制码外,各种字母和符号也必须用某种特定规则的二进制编码来表示。目前国际上普遍采用的是 ASCII 码(American Standard Code for Information Interchange),是美国标准信息交换码的简称,ASCII 码用 7 位二进制数码来表示,故可表示 $2^7 = 128$ 种不同的字符,这其中包括了 26 个大小写的英文字母、10 个十进制数字符号 0~9、7 个标点符号、9 个运算符号以及 50 个其他符号等。

4. 原码、反码和补码

1) 原码

在生活中,表示数的时候一般都是把正数前面加一个“+”,负数前面加一个“-”,但是在数字设备中,机器是不认识这些的。一般规定,用二进制的最低位“0”表示“+”,用二进制的最高位“1”表示“-”,其余数位表示数的大小。

例如, $[+105]_{\text{原}} = 01101001$, $[-105]_{\text{原}} = 11101001$, $[+0]_{\text{原}} = 00000000$, $[-0]_{\text{原}} = 10000000$ 。

有符号的8位(1B)二进制数表示的范围为 $-127 \sim +127$ 。

2) 反码

正数的反码与其原码相同;而负数的反码等于其绝对值各位求反。

例如, $[+105]_{\text{反}} = 01101001$, $[-105]_{\text{反}} = 10010110$, $[+0]_{\text{反}} = 00000000$, $[-0]_{\text{反}} = 11111111$ 。

3) 补码

在介绍补码概念之前,有必要先介绍补数的概念。以天天见到的时钟为例,设当前的准确时间是下午2点,可时钟却停在10点位置上,将时钟修正到下午2点位置有两种方法:一是将时钟顺时针方向拨4h,二是将时钟逆时针方向拨8h,其结果是一样的,上述两种拨钟方法可用下式表示:

$$10+4=14(\text{点}), 10-8=2(\text{点})$$

14点就是2点,这是因为时钟按12h为一个周期计数,12称为模,即时钟系统的模是12,14和2是以12为模的同余数,在12为模的系统中,4和8互为补数,此外,3和9,2和10等都是互为补数,它们的共同特点是两数相加等于12。因此,要求一个数的补数,用模减去这个数就可以了。

有了上述补数的概念之后,就容易理解补码的概念了,在数字系统电路中的补数就是补码。

一般来说,正数的补码与其原码相同;负数的补码是把其原码除符号位外的各位先求其反码,然后在最低位加1。

例如, $[+105]_{\text{补}} = 01101001$, $[-105]_{\text{补}} = 10010111$, $[0]_{\text{补}} = 00000000$, $[-1]_{\text{补}} = 11111111$ 。

在计算机内部,一般采用二进制补码形式表示一个二进制数,二进制数的最高位为符号位,若为“1”表示负数,若为“0”表示正数。

5. 奇偶校验码

在数据的存取、运算和传送过程中,难免会发生错误,把“1”错当成“0”或把“0”错当成“1”。奇偶校验码是一种能检验这种错误的代码。它分为两部分;信息位和奇偶校验位。有奇数个“1”称为奇校验,有偶数个“1”则称为偶校验。

五、存储器基础知识

1. 存储器的分类和指标

1) 半导体存储器的分类

半导体存储器是一种通用型LSI(大规模集成电路),主要用来存放大量的二值信息,

它是数字系统不可缺少的组成部分。半导体存储器的种类很多,一般按下述方法分类。

从制造工艺上分,有双极型和 MOS 型两类。双极型存储器以双极型触发器为基本存储单元,它具有工作速度快、功耗大的特点,主要用于对速度要求较高的场合;MOS 型存储器以 MOS 触发器或电荷存储结构为存储单元,它具有集成度高、功耗小、工艺简单等特点,主要用于大容量存储系统中。

从存取信息方式上分,有只读存储器(ROM, Read Only Memory)和随机存取存储器(RAM, Random Access Memory)两大类。只读存储器在正常工作时只能读出信息,而不能写入信息。ROM 的信息是在制造时或用专门的写入装置写入的,并可以长期保留,即断电后器件中信息不会消失,因此也称为非易失性存储器。RAM 正常工作时可以随时写入(存入)或读出(取出)信息,但断电后器件中的信息也随之消失,因此也称为易失性存储器。

存储器按信息的传输方式分有下列两种:一是并行存储器;二是串行存储器。

2) 半导体存储器的指标

存储器的存储容量和存取时间是反映系统性能的两个重要指标。

存储容量指存储器所能存放信息的多少,存储容量越大,说明存储的信息越多,系统的功能越强。通常,表示存储器存储容量的方式有下列两种:

一种是用存储单元数(所能记忆的字数)乘上字长表示,例如,存储器有 4096 个存储单元,字长为 8 位,则该存储器的存储容量为 4K 字节(4KB)。

另一种是用存储器所能记忆的全部二进制信息量直接表示。例如,上述的 4KB 的存储容量可表示为 $4K \times 8 = 32Kb$ (每字节为 8 位)。

存取时间一般用读(或写)周期来描述。读(或写)周期越短,即存取时间越短,存储器的工作速度就越快。

2. 只读存储器

1) 只读存储器的分类

ROM 由专用的装置写入数据,数据一旦写入,不能随意改写,切断电源后,数据不会消失,具有非易失性。ROM 器件的种类很多,按存储内容存入方式的不同,可以分成固定 ROM 和可编程 ROM。可编程 ROM 又可以细分为一次可编程存储器、光可擦除可编程存储器、电可擦除可编程存储器和快闪存储器等。

(1) 掩模 ROM。掩模 ROM 中存放的信息是由生产厂家采用掩模工艺专门为用户制作的,这种 ROM 出厂时其内部存储的信息就已经“固化”在里边了,所以也称固定 ROM。它在使用时只能读出,不能写入,因此通常只用来存放固定数据、固定程序和函数表等。

(2) 可编程 ROM(PROM)。PROM 在出厂时,存储的内容为全 0(或全 1),用户根据需要,可将某些单元改写为 1(或 0)。这种 ROM 采用熔丝或 PN 结击穿的方法编程,由于熔丝烧断或 PN 结击穿后不能再恢复,因此 PROM 只能改写一次。

(3) 可擦除的可编程 ROM(EPROM)。这类 ROM 利用特殊结构的浮栅 MOS 管进行编程,ROM 中存储的数据可以用紫外线照射进行多次擦除和改写。照射一般需要 15min~20min。为了便于照射擦除,芯片的封装外壳装有透明的石英盖板。EPROM 的擦除为一次全部擦除,数据写入需要通用或专用的编程器,编程器通常与微型计算

机联用。

(4)电可擦除的可编程 ROM(E²PROM/EEPROM)。EEPROM 的功能与 EPROM 类似,写进去的内容可以擦掉重写,而且不需要紫外线照射,只需用电学方法就可以擦除,所以它的使用要比 EPROM 方便一些,而且寿命也要长。EEPROM 芯片虽然能用电的方法擦除其内容,但它仍然是一种 ROM,具有 ROM 的典型特征,断电后芯片中的内容不会丢失。不管是 EPROM 还是 EEPROM,其可擦除的次数都是有限的。

EEPROM 对硬件电路没有特殊要求,操作十分简便。由于 EEPROM 片内设有编程所需的高压脉冲产生电路,因而无需外加编程电源和编程脉冲即可完成写入工作。

(5)快闪存储器(Flash ROM)。EEPROM 虽然具有既可读又可写的特点,但写入的速度较慢,使用起来不太方便,而 Flash ROM 是在 EPROM 和 EEPROM 的基础上发展起来的一种只读存储器,这种存储器读写速度很快,存取时间可达 70ns,存储容量可达 64MB,芯片的可改写次数为 1 万次~100 万次。目前,Flash ROM 在单片机上应用十分普遍。

3. 随机存储器

随机存取存储器(RAM)可随时从其中任一指定地址读出(取出)或写入(存入)数据。按照存储机理的不同,RAM 又可分为静态 RAM(SRAM)和动态 RAM(DRAM)。RAM 使用灵活方便,但是 RAM 具有易失性,一旦失电,所存储的数据立即丢失。

1)随机存储器的结构

RAM 主要由地址译码器、读/写控制和输入/输出(I/O)接口电路、片选控制和存储矩阵等部分组成,如图 1-1 所示。

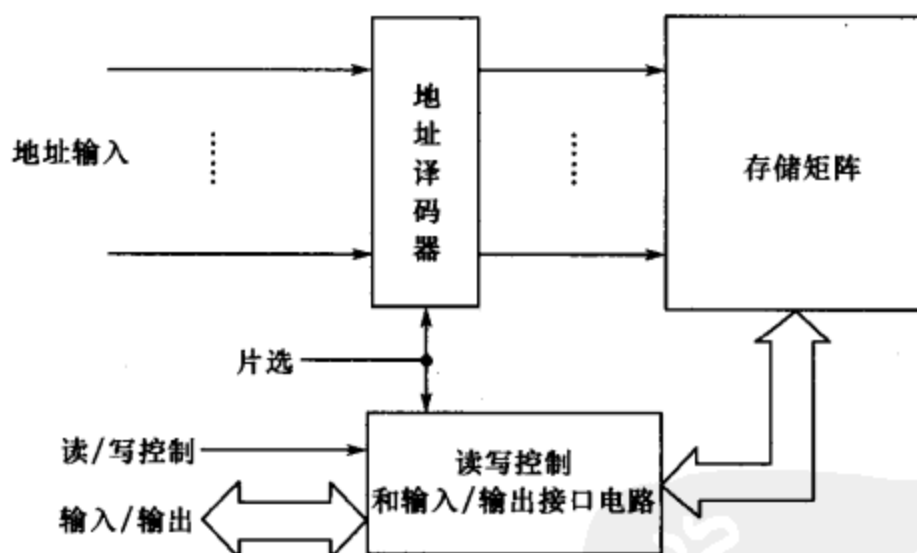


图 1-1 随机存储器的结构示意图

(1)地址译码器。存储器的每个存储单元都有地址,每次访问存储器只能与存储器中的某一个存储单元发生数据交换,微处理器发出访问存储器中某一存储单元的指令后,通过地址译码器找到该指定的存储单元。具体地讲,地址译码器将从存储器外部输入的二进制地址码进行译码,在译码器相应的某一条输出线上给出信号,控制被选中的存储单元与存储器输入/输出端接通,以便读出该存储单元的数据或将数据写入该单元。

(2)读/写控制。访问 RAM 时,对于被选中地址的存储单元,是进行写入操作还是读

出操作,要由读/写控制线进行控制。有的 RAM 的读/写控制线为高电平读、低电平写。也有的 RAM 的读/写控制线是分开的,一条为读,另一条为写。

(3)输入/输出接口电路。输入/输出接口电路是存储器数据进出的通道, RAM 通过输入/输出电路与微处理器交换信息,在进行读操作时这一接口是存储器的输出端,在写入时这一接口是输入端,一线进出二用,由读/写控制线控制。存储器中的输入/输出端数目决定于存储单元的位数。

(4)片选控制。由于集成度的限制,通常需要把多片 RAM 组装在一起,才能构成微处理器的存储器。微处理器在访问存储器时,一次只与一片 RAM 交换信息,而与其他片 RAM 不发生联系,片选信号就是用来实现这种控制的。当某一片的片选信号为有效低电平时,则该片被选中。

(5)存储矩阵(存储器)。RAM 中的存储单元通常被排列成矩阵形式,称为存储矩阵。地址译码器的输出信号,控制着存储矩阵与输入/输出端的连接状态,被选中的存储单元就接通,未被选中的存储单元就处于断开的状态。

2)静态和动态随机存储器

(1)静态随机存储器(SRAM)。采用静态存储单元构成的 RAM 称之为 SRAM。SRAM 电路结构复杂,所用管子数目多,功耗大,工作速度低,优点是,只要不断电,只要不对存储单元进行改写,静态存储单元所存储的信息就不会丢失。单片机一般使用 SRAM。

(2)动态随机存储器(DRAM)。采用动态存储单元构成的 RAM 称之为 DRAM。动态存储单元与静态存储单元不同,即使不断电,但一段时间后如果不对原信息进行刷新,存储单元所存储的原信息就会丢失,所以 DRAM 存在一个刷新的问题。所谓刷新,是指每经过一段时间(MOS 管电路约为 20ms),要对所保存的信息全面进行一次重写,这种周期性地重复重写,就是刷新。

DRAM 功耗小,工作速度快,存储单元所用的元器件数目较少,更能适应制成大规模集成电路。但是,DRAM 需要刷新电路,使这种存储器的外围电路比较复杂。另外,在 DRAM 进行刷新期间,不能对存储进行读出和写入操作,使存储器的有效利用时间受到限制。

第二节 单片机概述

一、什么是单片机

可以说,20 世纪跨越了 3 个“电”的时代,即电气时代、电子时代和现已进入的电脑时代。不过,这种电脑,通常是指个人计算机,简称 PC 机。它由主机、键盘、显示器等组成,如图 1-2 所示。还有一类计算机,大多数人却不怎么熟悉。这种计算机就是把智能赋予各种机械的单片机,也称微控制器,如图 1-3 所示。顾名思义,这种计算机的最小系统只用了一片集成电路,即可进行简单运算和控制。因为它体积小,通常都藏在被控机械的“肚子”里。它在整个装置中,起着犹如人类头脑的作用,它一旦出了毛病,整个装置就瘫痪了。现在,这种单片机的使用领域已十分广泛,如智能仪表、实时工控、通信设备、导航

系统、家用电器等。各种产品一旦用上了单片机,就能起到使产品升级换代的功效,常在产品名称前冠以形容词——“智能型”,如智能型洗衣机等。现在有些工厂的技术人员或其他业余电子开发者搞出来的某些产品,不是电路太复杂,就是功能太简单,且极易被仿制。究其原因,可能就卡在产品未使用单片机或其他可编程逻辑器件上。



图 1-2 PC 机的外形

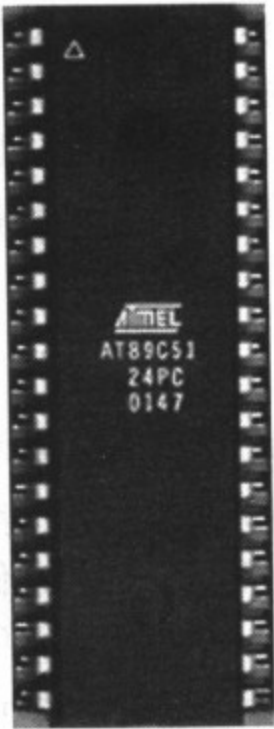


图 1-3 单片机的外形

二、单片机名称的由来

单片机因将其主要组成部分集成在一个芯片上而得名,具体说就是把中央处理器(CPU)、随机存储器(RAM)、只读存储器(ROM)、中断系统、定时器/计数器以及输入/输出(I/O)口电路等主要微型机部件,集成在一块芯片上。虽然单片机只是一个芯片,但从组成和功能上看,它已具有了计算机系统的属性,为此称它为单片微型计算机(SCMC, Single Chip Microcomputer),简称单片机。

单片机主要应用于控制领域,用以实现各种测试和控制功能,为了强调其控制属性,也可以把单片机称为微控制器(MCU, Micro Controller Unit)。在国际上,“微控制器”的叫法似乎更通用一些,而在我国则比较习惯于“单片机”这一名称,因此本书使用“单片机”一词。

由于单片机在应用时通常是处于被控系统的核心地位并融入其中,即以嵌入的方式进行使用,为了强调其“嵌入”的特点,也常常将单片机称为嵌入式微控制器(EMCU, Embedded Micro Controller Unit)。在单片机的电路和结构中有许多嵌入式应用的特点。

三、单片机与单片机系统

单片机通常是指芯片本身,它是由芯片制造商生产的,在它里面集成的是一些作为基本组成部分的运算器电路、控制器电路、存储器、中断系统、定时器/计数器以及输入/输出口电路等。但一个单片机芯片并不能把计算机的全部电路都集成到其中,例如,组成谐振电路和复位电路的石英晶体、电阻、电容等,这些元件在单片机系统中只能以分立元件的形式出现。此外,在实际的控制应用中,常常需要扩展外围电路和外围芯片。从中可以看

到单片机和单片系统的差别,即单片机只是一个芯片,而单片机系统则是在单片机芯片的基础上扩展其他电路或芯片构成的具有一定应用功能的计算机系统。

通常所说的单片机系统都是为实现某一控制应用需要由用户设计的,是一个围绕单片机芯片而组建的计算机应用系统。在单片机系统中,单片机处于核心地位,是构成单片机系统的硬件和软件基础。

在单片机硬件的学习上,既要学习单片机,也要学习单片机系统,即单片机芯片内部的组成和原理,以及单片机系统的组成方法。

第三节 单片机的分类、发展及应用

一、单片机的分类

1. 按单片机数据总线的位数分类

按单片机数据总线的位数,分为4位、8位、16位、32位单片机。

(1)4位单片机。4位单片机适合用于各种规模较小的家电类消费产品。一般的单片机厂家均有自己的4位单片机产品,如NEC公司的75006×系列、EPSON公司的SMC62系列等。

4位单片机的典型应用领域有PC机用的输入装置(鼠标、游戏杆)、电池充电器(Ni-Cd电池、锂电池)、运动器材、带液晶显示的音/视频产品控制器、一般家用电器的控制及遥控器、玩具控制、记时器、时钟、表、计算器、多功能电话、LCD游戏机等。

(2)8位单片机。8位单片机是目前品种最为丰富、应用最为广泛的单片机,具有体积小、功耗低、功能强、性能价格比高、易于推广应用等显著优点。目前主要分为MCS-51系列及其兼容机型和非MCS-51系列单片机。

MCS-51兼容产品因开发工具及软、硬件资源齐全而占主导地位,ATMEL、PHILIPS、WINBOND是MCS-51单片机生产的老牌厂家,CYGNAL及SST公司也推出新的产品,其中SST新推出的 μ PSD系列片内有大容量Flash(128KB/256KB)、8KB/32KB的SRAM、集成A/D、看门狗、上电复位电路、两路UART、支持在系统编程ISP及在应用中编程IAP等诸多先进特性,迅速被广大51单片机用户所接受。

非51系列单片机在中国应用较广的有摩托罗拉(MOTOROLA)的68HC05/08系列、微芯(Microchip)的PIC单片机以及ATMEL公司的AVR单片机。

8位单片机在自动化装置、智能仪器仪表、过程控制、通信、家用电器等许多领域得到广泛应用。

(3)16位单片机。16位单片机操作速度及数据吞吐能力在性能上比8位机有较大提高。目前以Intel的MCS-96/196系列、TI的MSP430系列及MOTOROLA的68HC11系列为主。

16位单片机主要应用于工业控制、智能仪器仪表、便携式设备等场合。其中TI的MSP430系列以其超低功耗的特性广泛应用于低功耗场合。

(4)32位单片机。32位单片机是单片机的发展趋势,随着技术发展及开发成本和产品价格的下降,将会与8位机并驾齐驱。生产32位单片机的厂家与8位机的厂家一样

多。在众多的 32 位单片机中,以 ARM 单片机最为著名,在 2001 年,32 位 ARM 单片机市场占有率超过了 75%,广泛应用在信息电器如掌上电脑、个人数字助理(PDA)、可视电话、移动电话、TV 机顶盒、数码相机等嵌入式设备。

ARM 是微处理器行业的一家知名企业,是知识产权(IP)供应商,本身不生产芯片,靠转让设计许可由合作伙伴来生产各具特色的芯片。ARM 公司设计了大量高性能、廉价、耗能低的 RISC 处理器、相关技术及软件。目前,有超过 30 家半导体公司与 ARM 签订了硬件技术使用许可协议,其中包括英特尔(Intel)、IBM、SAMSUNG、OKI、LG、NEC、索尼(SONY)、飞利浦(PHILIPS)等大公司。至于软件系统的合伙人,则包括微软、SYMBIAN 和 MRI 等一系列知名公司。

2. 根据控制应用的需要分类

根据控制应用的需要,可以将单片机分成为通用型和专用型两种类型。

通用型单片机是一种基本芯片,它的内部资源比较丰富,性能全面且适用性强,能覆盖多种应用需求。用户可以根据需要设计成各种不同应用的控制系统,即通用单片机有一个再设计的过程,通过用户的进一步设计,才能组建成一个以通用单片机芯片为核心,再配以其他外围电路的应用控制系统。

然而,在单片机的控制应用中,有许多时候是专门针对某个特定产品的,例如电度表和 IC 卡读写器上的单片机等。这种应用的最大特点是针对性强,而且数量巨大,为此,厂家常与芯片制造商合作,设计和生产专用的单片机芯片。由于专用单片机芯片是针对一种产品或一种控制应用而专门设计的,设计时已经对系统结构的最简化、软硬件资源利用的最优化、可靠性和成本的最佳化等方面都做了通盘的考虑和论证,所以专用单片机具有十分明显的综合优势。

3. 按单片机的兼容性分类

按单片机的兼容性分,单片机可分为不兼容型和兼容型两种。

不兼容型单片机的指令系统是为了使之具有较强的控制能力和控制效率而根据控制的需要重新设计的,如 Intel 公司的 MCS-48、MCS-51、MCS-96 系列单片机之间就不兼容。兼容型单片机的指令系统与原有单片机系统基本相同,兼容型单片机在原来的通用微处理器(CPU)的基础上扩展存储器、I/O 接口等部件而成。PHILIPS 公司的 80C51 及其派生产品、ATMEL 公司的 AT89 系列快闪存储器型单片机等均与 Intel 公司 MCS-51 系列单片机完全兼容,因其更为灵活、强大的硬件功能而后来居上,尤其是 ATMEL 公司的 AT89 系列单片机取代 MCS-51 系列单片机中的 EPROM 型产品的趋势更是显而易见。

二、单片机发展的历史

1971 年 11 月,美国 Intel 公司首先设计成 4 位微处理器 Intel 4004,并且配有随机存取存储器(RAM)、只读存储器(ROM)和移位寄存器等芯片,构成第一台 MCS-4 微型计算机。1972 年 4 月 Intel 公司又研制成了功能较强的 8 位微处理器 Intel 8008,这些微处理器虽说还不是单片机,但从此拉开了研制单片机的序幕。

1976 年,Intel 公司推出了 MCS-48 系列单片机,它以体积小、控制功能全、价格低等特点,赢得了广泛的应用和好评,为单片机的发展奠定了坚实的基础,成为单片机发展史

上的一个重要阶段。其后,在 MCS-48 成功的刺激下,许多半导体芯片生产厂商竞相研制和发展自己的单片机系列。到 20 世纪 80 年代末,世界各地已相继研制出大约 50 个系列 300 多个品种的单片机产品,其中有 MOTOROLA 公司的 6801、6802, Zilog 公司的 Z-8 系列, Rockwell 公司的 6501、6502 等。此外,日本的 NEC 公司、日立(HITACHI)公司等也不甘落后,相继推出了各自的单片机品种。

尽管目前单片机的品种很多,但是在我国使用最多的是 Intel 公司的 MCS-51 单片机系列。MCS-51 是在 MCS-48 的基础上于 20 世纪 80 年代初发展起来的,虽然它仍然是 8 位的单片机,但其功能较 MCS-48 有很大的增强。此外,它还具有品种全、兼容性强及软、硬件资料丰富等特点,因此应用愈加广泛,成为比 MCS-48 更重要的单片机品种。直到现在, MCS-51 仍不失为单片机的主流系列。

继 8 位单片机之后,又出现了 16 位单片机,1983 年 Intel 公司推出的 MCS-96 系列单片机就是其中的典型代表。与 MCS-51 相比, MCS-96 不但字长增加一倍,而且在其他性能方面也有很大的提高,特别是芯片内还增加了一个 4 路或 8 路的 10 位模/数(A/D)转换器,使其具有 A/D 转换功能。

随着集成电路的发展及信息时代的到来,开始出现了以 ARM 为代表的 32 位单片机,目前, ARM 单片机在移动通信、手持计算、多媒体数字消费等嵌入式设备中得到了广泛的应用。

目前,单片机园地里,单片机品种异彩纷呈,争奇斗艳。有 8 位、16 位和 32 位机,但 8 位单片机仍以它的价格低廉、品种齐全、应用软件丰富、支持环境充分、开发方便等特点而占着主导地位。而 Intel 公司凭着他们雄厚的技术,性能优秀的机型和良好的基础,目前仍是单片机的主流产品。

三、单片机技术发展的特点

单片机是现代计算机、电子技术结合的产物,无论是单片机本身,还是单片机应用系统,都在随着时代的发展不断发生变化。目前及未来相当长一段时间内,单片机技术将表现出以下几个特点。

1. 8 位、32 位单片机共同发展

这是当前单片机技术发展的另一动向。长期以来,单片机技术的发展是以 8 位机为主的。随着移动通信、网络技术、多媒体技术等高科技产品进入家庭,32 位单片机应用将得到长足、迅猛的发展。

2. 单片机速度越来越快

为提高单片机抗干扰能力,降低噪声,降低时钟频率而不牺牲运算速度是单片机技术发展的追求。一些 8051 单片机兼容厂商改善了单片机的内部时序,在不提高时钟频率的条件下,使运算速度提高了很多, MOTOROLA 公司的单片机则使用了锁相环技术或内部倍频技术,使内部总线速度大大高于时钟产生器的频率。

3. 低电压与低功耗

几乎所有的单片机都有省电运行方式。允许使用的电源电压范围也越来越宽。一般单片机都能在 3V~6V 范围内工作,对电池供电的单片机不再需要对电源采取稳压措施。低电压供电的单片机电源下限已由 2.7V 降至 2.2V、1.8V。0.9V 供电的单片

机已经问世。

4. 低噪声与高可靠性技术

为提高单片机系统的抗电磁干扰能力,使产品能适应恶劣的工作环境,满足电磁兼容性方面更高标准的要求,各单片机商家在单片机内部电路中采取了一些新的技术措施。如 ST 公司的 μ PSD 系列单片机片内增加了看门狗定时器,NS 的 COP8 单片机内部增加了抗 EMI 电路,增强了“看门狗”的性能。

5. ISP 及 IAP

在片编程技术(ISP, In System Programming)及在应用中编程(IAP, In Application Programming)是通过单片机上引出的编程线、串行数据、时钟线等对单片机编程,编程线与 I/O 线共用,不增加单片机的额外引脚。ISP 为开发调试提供了方便,并使单片机系统远程调试、升级成为现实。

6. 制作工艺 CMOS 化

出于对低功耗的普遍要求,目前各大厂商推出的各类单片机产品都采用了 CMOS 工艺。80C51 系列单片机采用两种半导体工艺生产。一种是 HMOS 工艺,即高密度短沟道 MOS 工艺;另外一种是 CHMOS 工艺,即互补金属氧化物的 HMOS 工艺。CHMOS 是 CMOS 和 HMOS 的结合,除保持了 HMOS 的高速度和高密度的特点之外,还具有 CMOS 低功耗的特点。例如 8051 的功耗为 630mW,而 80C51 的功耗只有 120mW。在便携式、手提式或野外作业仪器设备上低功耗是非常有意义的。因此,在这些产品中必须使用 CHMOS 的单片机芯片。

四、MCS-51 单片机家族简介

虽然目前单片机的品种很多,但其中最具代表性的当属 Intel 公司的 MCS-51(以下简称 51)单片机系列。51 单片机以其典型的结构和完善的总线专用寄存器的集中管理,众多的逻辑位操作功能及面向控制的丰富的指令系统,堪称为一代“名机”,为以后的其他单片机的发展奠定了基础。正因为其优越的性能和完善的结构,导致后来的许多厂商多沿用或参考了其体系结构,有许多大的电气厂商丰富和发展了 51 单片机,像 PHILIPS、Dallas、ATMEL 等著名的半导体公司都推出了兼容 51 的单片机产品,就连我国台湾的 WINBOND 公司也发展了兼容 51 单片机品种。

近年来 51 单片机获得了飞速的发展,51 的发源公司 Intel 由于忙于开发 PC 机及高端微处理器而无精力继续发展自己的单片机,而由其他厂商将其发展,最典型的是 PHILIPS 和 ATMEL 公司。

PHILIPS 公司主要是改善其性能,在原来的基础上发展了高速 I/O 接口、A/D 转换器、脉宽调制(PWM)、WDT 等增强功能,并在低电压、低功耗、扩展串行总线(I²C)和控制网络总线(CAN)等功能加以完善。

ATMEL 公司推出的 AT89Cxx 系列兼容 51 的单片机,完美地将 Flash ROM 与 80C51 内核结合起来,仍采用 51 的总体结构和指令系统,Flash ROM 的可反擦写程序存储器能有效地降低开发费用,并使单片机能多次重复使用。

西门子(SIEMENS)公司推出的 C500 系列单片机在保持与 80C51 兼容的前提下,增强了各项性能,尤其是增强了电磁兼容性能,增加了 CAN 总线接口,特别适用于工业控

制、汽车电子、通信和家电领域。

我国台湾的 WINBOND 公司也开发了一系列兼容 C51 的单片机,其产品通常具备丰富的功能特性,又因其质优价廉,因而在市场也占有一定的份额。

1. MCS - 51 单片机主要技术指标

8051/80C51 系列又分成 51 和 52 两个子系列,并以芯片型号的最末位数字作为标志。其中 51 子系列是基本型,而 52 子系列则属增强型。8051/80C51 单片机系列芯片的技术指标如表 1-5 所列。

表 1-5 80C51 单片机系列芯片的技术指标

系列	典型芯片	片内 ROM 形式	片内 RAM	并行 I/O 接口	定时器/计数器	中断源	串行 I/O 接口
51 子系列	8031/80C31	无	128B	4×8	2×16	5	1
	8051/80C51	4KB 掩模 ROM	128B	4×8	2×16	5	1
	8751/87C51	4KB EPROM	128B	4×8	2×16	5	1
	89C51	4KB Flash ROM	128B	4×8	2×16	5	1
	89LV51	4KB Flash ROM	128B	4×8	2×16	5	1
52 子系列	8032/80C32	无	256B	4×8	3×16	6	1
	8052/80C52	8KB 掩模 ROM	256B	4×8	3×16	6	1
	8752/87C52	8KB EPROM	256B	4×8	3×16	6	1
	89C52	8KB Flash ROM	256B	4×8	3×16	6	1
	89SLV52	8KB Flash ROM	256B	4×8	3×16	6	1
	89S8252	8KB Flash ROM	256B	4×8	3×16	6	1
1051	89C1051	1KB Flash ROM	64B	15	1×16	3	1
2051	89C2051	2KB Flash ROM	128B	15	2×16	5	1

2. MCS - 51 单片机主要型号的特点

1)8031/80C31

8031/80C31 片内不带程序存储器 ROM,使用时用户需外接程序存储器和一片逻辑电路 373,外接的程序存储器多为 EPROM 的 2764 系列。用户若想对写入到 EPROM 中的程序进行修改,必须先用一种特殊的紫外线灯将其照射擦除,之后才可写入。写入到外接程序存储器的程序代码没有什么保密性可言。

2)8051/80C51

8051/80C51 片内有 4KB ROM,无需外接外存储器和 373,更能体现“单片”的简练。但是用户编的程序无法烧写到其 ROM 中,只有将程序交芯片厂代用户烧写,并且是一次性的,今后用户和芯片厂都不能改写其内容。

3)8751/87C51

8751/87C51 与 8051 基本一样,但 8751 片内有 4KB 的 EPROM,用户可以将自己编写的程序写入单片机的 EPROM 中进行现场实验与应用,EPROM 的改写同样需要用紫外线灯照射一定时间擦除后再烧写。

由于上述类型的单片机应用得早,影响很大,已成为事实上的工业标准。后来很多芯

片厂商以各种方式与 Intel 公司合作,也推出了同类型的单片机,如同一种单片机的多个版本一样,虽都在不断地改变制造工艺,但内核却一样,也就是说这类单片机指令系统完全兼容,绝大多数管脚也兼容;在使用上基本可以直接互换。人们统称这些与 8051 内核相同的单片机为“51 系列单片机”。对于学习者来说,学了其中一种,便会所有的 51 系列。

4)89 系列

在众多的 51 系列单片机中,要算 ATMEL 公司的 AT89xx 更实用,因他不但和 8051 指令、管脚完全兼容,而且其片内的程序存储器是 Flash 工艺的,这种工艺的存储器用户可以用电的方式瞬间擦除、改写,一般专为 ATMEL AT89xx 做的编程器均带有这些功能。显而易见,这种单片机对开发设备的要求很低,开发时间也大大缩短。写入单片机内的程序还可以进行加密,这又很好地保护了用户的劳动成果。再者,AT89C51 目前的售价比 8031 还低,市场供应也很充足。

89 系列单片机有多种型号,主要有 AT89C51、AT89LV51、AT89C52、AT89LV52、AT89C1051、AT89C2051 和 AT89S8252 等。其中 AT89LV51 和 AT89LV52 分别是 AT89C51 和 AT89C52 的低电压产品,最低电压可以低至 2.7V。而 AT89C1051 和 AT89C2051 则是低档型低电压产品。它们只有 20 条引脚,最低电压也为 2.7V。AT89S8252 属高档型,除了 8KB Flash 存储器外,AT89S8252 还含有一个 2KB 的 EEPROM,从而提高了存储容量。

ATMEL 的 89 系列有多种封装,如 AT89C51 有 DIP、PLCC 和 PQFP/TQFP 等封装;2051/1051 有 DIP 和 SOIC 封装等。

五、单片机的应用

目前单片机渗透到我们生活的各个领域,几乎很难找到哪个领域没有单片机的踪迹。导弹的导航装置,飞机上各种仪表的控制,计算机的网络通信与数据传输,工业自动化过程的实时控制和数据处理,广泛使用的各种智能 IC 卡,民用豪华轿车的安全保障系统,录像机、摄像机、全自动洗衣机的控制,以及程控玩具、电子宠物等,这些都离不开单片机。更不用说自动控制领域的机器人、智能仪表、医疗器械了。因此,单片机的学习、开发与应用将造就一批计算机应用与智能化控制的科学家、工程师。下面简要说明单片机的主要应用领域。

1. 工业自动化方面

自动化能使工业系统处于最佳状态、提高经济效益、改善产品质量和减轻劳动强度。因此,自动化技术广泛应用于机械、电子、电力、石油、化工、纺织、食品等轻、重工业领域中,而在工业自动化技术中,无论是过程控制技术、数据采集和测控技术,还是生产线上的机器人技术,都需要有单片机的参与。

在工业自动化的领域中,机电一体化技术将发挥愈来愈重要的作用,在这种集机械、微电子和计算机技术于一体的综合技术中,单片机将发挥越来越大的作用。

2. 仪器仪表方面

现代仪器仪表(例如测试仪表和医疗仪器等)的自动化和智能化要求越来越高,对此,最好使用单片机来实现,而单片机的使用又将加速仪器仪表向数字化、智能化、多功能化和柔性化方向发展。

此外,单片机的使用还有助于提高仪器仪表的精度和准确度,简化结构、减小体积及质量而易于携带和使用,并具有降低成本,增强抗干扰能力,便于增加显示、报警和自诊断等功能。

3. 家用电器方面

当前,家用电器产品的一个重要发展趋势是不断提高其智能化程度,而家电智能化的进一步提高就需要有单片机的参与,所以生产厂家常标榜“电脑控制”以提高其产品的档次,例如洗衣机、电冰箱、空调机、微波炉、电视机和音像、视频设备等,这里所说的“电脑”实际上就是“单片机”。

智能化家用电器将给我们带来更大的舒适和方便,进一步改善我们的生活质量,使我们的生活变得更加丰富多彩。

4. 信息和通信产品方面

信息和通信产品的自动化和智能化程度很高,这当然离不开单片机的参与。例如,计算机的外部设备(键盘、打印机、磁盘驱动器等)和自动化办公设备(传真机、复印机、考勤机、电话机等)中,都有单片机在其中发挥着作用。

5. 军事装备方面

科技强军、国防现代化离不开计算机,在现代化的飞机、军舰、坦克、火炮、导弹火箭和雷达等各种军用装备上,都有单片机深入其中。

第四节 到单片机世界去遨游

独具魅力的单片机能使使用者体会到电脑的真谛,使用者可以亲自动手用单片机设计智能玩具,可以设计不同的应用程序实现不同的功能。既有硬件制作又有软件设计,既动脑、又动手。初级水平的可开发智能玩具。中级水平的可开发一些智能控制器,如电脑鼠、智能车、各种遥控模型。高级水平的可开发工业控制单元、网络通信等,并用汇编语言或C语言设计应用程序。围绕单片机形成的电子产业的未来,将会为电子爱好者提供广阔的天地。投身到单片机世界来,将使使用者一生受益。

一、如何学习单片机

学习单片机是否很困难呢?应当说,对于已经具有电子电路,尤其是数字电路基本知识的读者来说,不会有太大困难,如果读者对PC机有一定基础,学习单片机就更容易。学习单片机技术,还要有好的学习方法,方法找对时,即使具有中学文化水平的电子爱好者也很容易入门,一旦入了门,就可利用单片机做各种电路实验。下面简要介绍学习单片机的几点经验,可供读者在学习时参考。

1. 找准突破口

目前,单片机的种类很多,学习单片机最好从51系列单片机开始,第一是书多、资料多,而且掌握51技术的人多,碰到问题能请教的老师也就多了。51系列的实验芯片(如AT89C51)价格低廉而且很容易买到,AT89C51芯片而且可以反复擦写1000次以上,对于初学者来说真是太合适了,就算以后考虑工业运用,也可以先学透51后再学其他类型的单片机,毕竟技术是相通的。

2. 多读书

单片机是一个知识密集的课程,不看书是绝对不行的。需要说明的是,单片机技术是以数字电路为基础的,所以,学习单片机前,应对数制、数字电路有比较全面的了解,学习数字电路时,应重点掌握数字集成电路的电路功能使用,而对具体的内部工作原理只需作简单了解即可。如果读者已经学会了数字电路的用法,就走到单片机的门口了。

3. 购买实验工具

单片机和PC机一样,是实践性很强的一门技术,有人说“计算机是玩出来的”,单片机也一样,只有多“玩”,也就是多练习、多实际操作,才能真正掌握它。因此,本书会提供各种练习和实验,并介绍一些适用于初学者且性价比较高的单片机和开发系统。只有认真完成这些实践环节,才能为进一步深造打好基础。下面简要介绍单片机实验中的几种常用工具。

1) 编程器

编程器通过与电脑连接可以对存储器、单片机等器件进行读写,并通过电脑编辑软件可以对单片机中的程序(未加密)进行编辑修改,然后重新写入芯片内,又被称为烧录器。

编程器根据其支持烧录器件的多少和性能,以及品牌、价格、档次有很大的差异,从最便宜的几百元到性能比较高档的几千多元都有。编程器可以分为专用编程器和通用编程器,专用编程器是针对某一类器件开发的烧录器,针对性强。通用型编程器针对常用器件,适用面广。不同档次的编程器都有自己针对的应用群体。选购编程器首先必须适合自己使用,简而言之就是编程器必须支持用户所需要烧录的芯片和一些附加功能如加密。笔者用的编程器是一种性能较好的 RF-810 通用编程器,RF-810 可对 100 余厂家的 1000 多种常用器件进行编程、测试,采用 40 脚锁紧插座,与计算机并口(打印机)联机工作。价格便宜,经济实用,软件功能齐备,对芯片的编程不需要人工干预,软件用户界面易学,使用相当方便。RF-810 编程器及其配件外观如图 1-4 所示。



图 1-4 RF-810 编程器及其配件外观

目前,市场上为了满足一些低收入初学者学习单片机的需求,推出了多款低价位普及型编程器,这类编程器的价格一般都在 200 元以下,和通用编程器最大的区别在于它们支持编程的单片机芯片少,一般只支持 51 系列单片机中的几种。而且为了降低成本,这类编程器一般都没有外壳,做成裸板结构:

除以上介绍的通用编程器和专用编程器外,目前,市场上还出现了一种 ISPROM 下载型编程器,外观如图 1-5 所示。

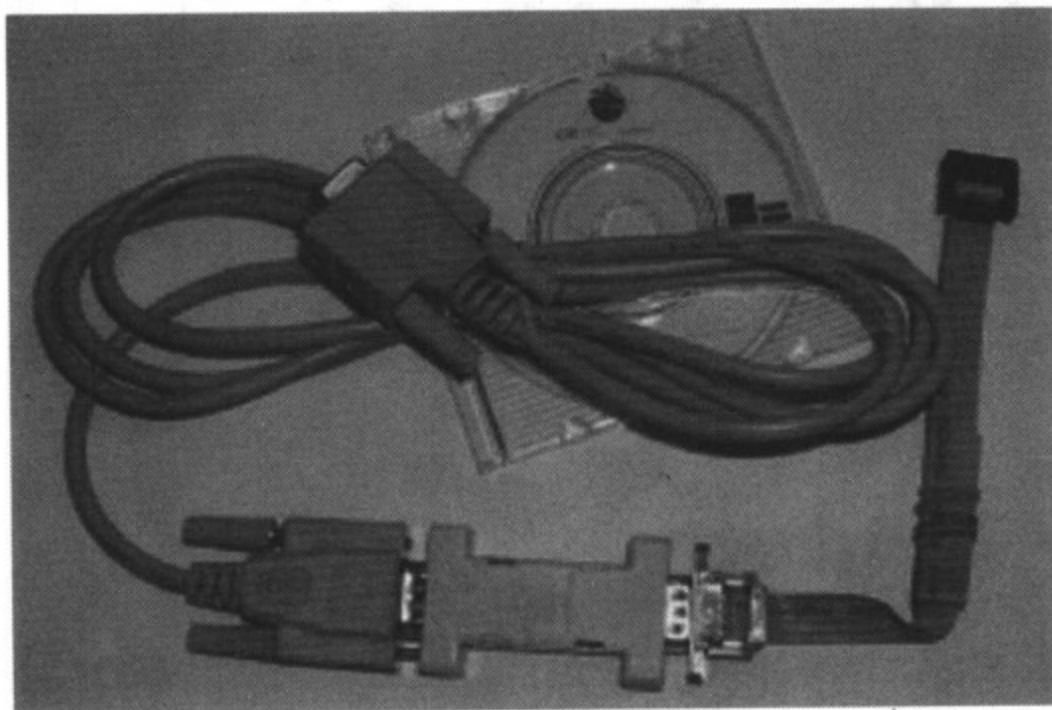


图 1-5 下载型编程器

这种编程器结构简单,价格便宜,使用方便,性能稳定可靠,可以通过实验板上的 ISP 插座(有些实验板上有 ISP 插座),对单片机进行编程。

ISP(In-System Programming)是“在系统可编程”的意思,指电路板上的空白器件可以编程写入最终用户代码,而不需要从电路板上取下器件,已经编程的器件也可以用 ISP 方式擦除或再编程。ISP 技术是未来发展方向。

2)实验板

单片机是一门实践性很强的课程,需要反复练习和实践,因此,学习时需要一块 51 试验板,通过做一系列的实验,从而比较容易地领会了单片机那些枯燥、难懂的专业术语,在这里,笔者推荐两种性价比较好的实验板,一种是需要配合通用编程器的实验板,如图 1-6 所示,另一种是配合下载型编程器的实验板,如图 1-7 所示。

3)仿真器

单片机仿真器的作用是在产品开发阶段用来替代单片机进行软、硬件调试的非常有用的开发工具。使用单片机仿真器可以对单片机程序进行单步、全速、断点等手段的调试,检查程序运行中单片机 RAM、寄存器内容的变化,观察程序的运行情况,与此同时可以对硬件电路进行实时的调试。使用单片机仿真器可以迅速发现、纠正程序中的错误,从而大大缩短单片机开发的周期。

如果不使用单片机仿真器,而是利用单片机烧录器反复烧写单片机进行开发,对于程序设计中的错误就只能通过分析、猜测,然后修改程序、重新烧录、重新实验来完成,这样就大大增加了调试的难度,延长了开发时间,特别对于单片机开发经验并不丰富的开发者来说尤为困难。

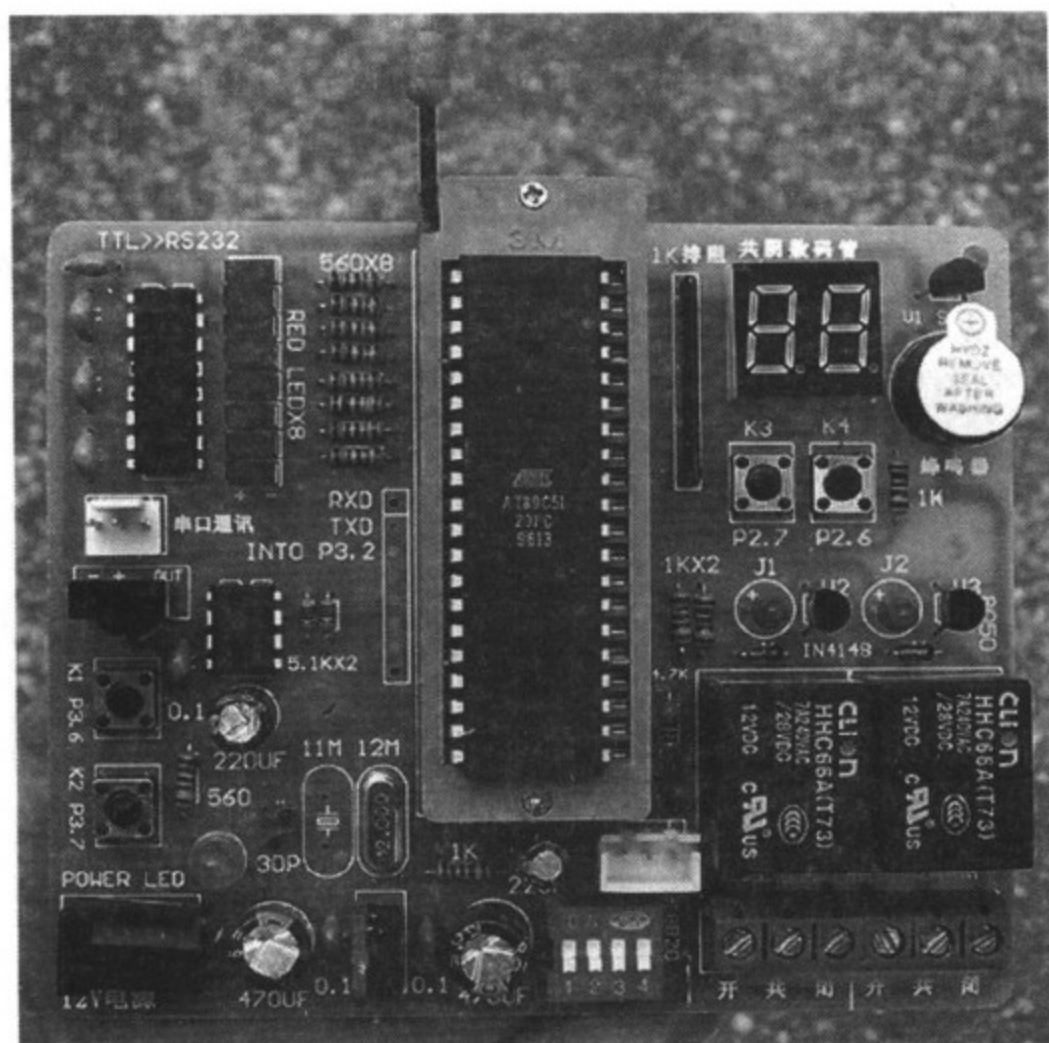


图 1-6 配合通用编程器的实验板

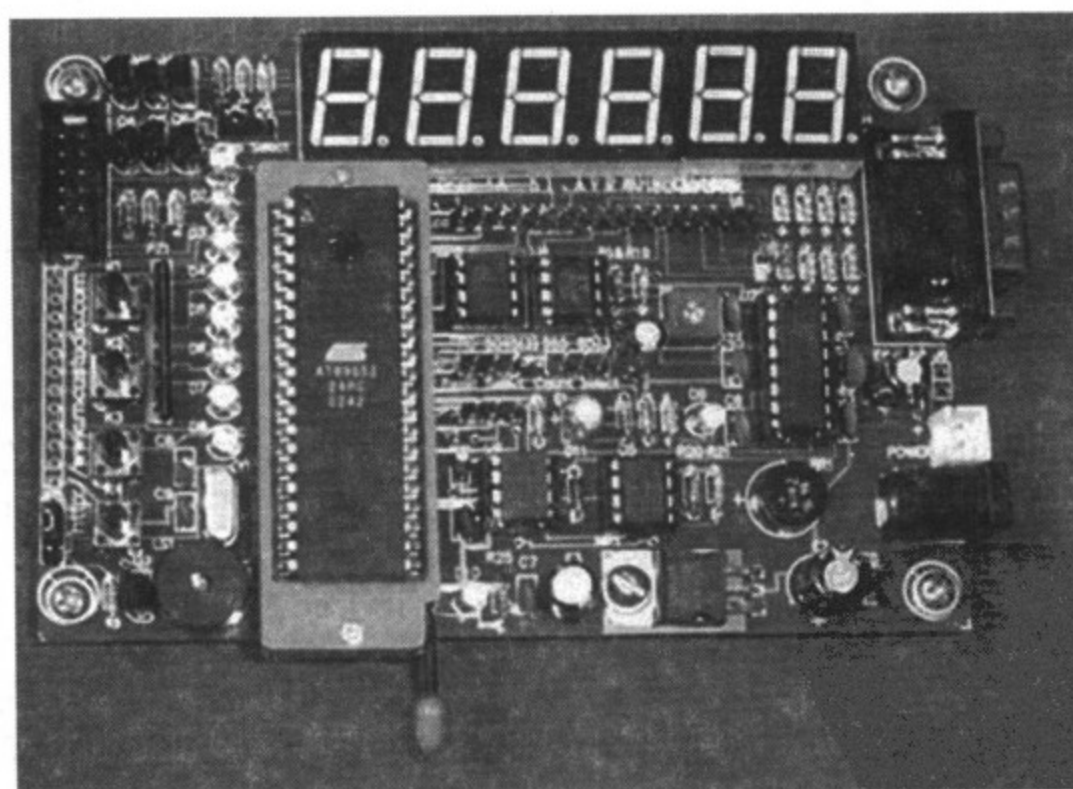


图 1-7 配合下载型编程器的实验板

目前市场上传统的仿真器价格都在几百元以上,对经济不是非常宽裕的人来说是不小的开支,好在仿真器是用来提高调试程序效率的,并不是非需要不可;不过,如果想模拟实验的真实性,最好还是拥有一台仿真器。图 1-8 是 Insight 仿真器外观图。

现在,市场上出现了一种新型的廉价简易在线仿真器,如 MON 51 仿真器,其外观如图 1-9 所示。在线仿真器 MON 51 是完全依托 KEIL 51 软件强大的功能来实现仿真的,

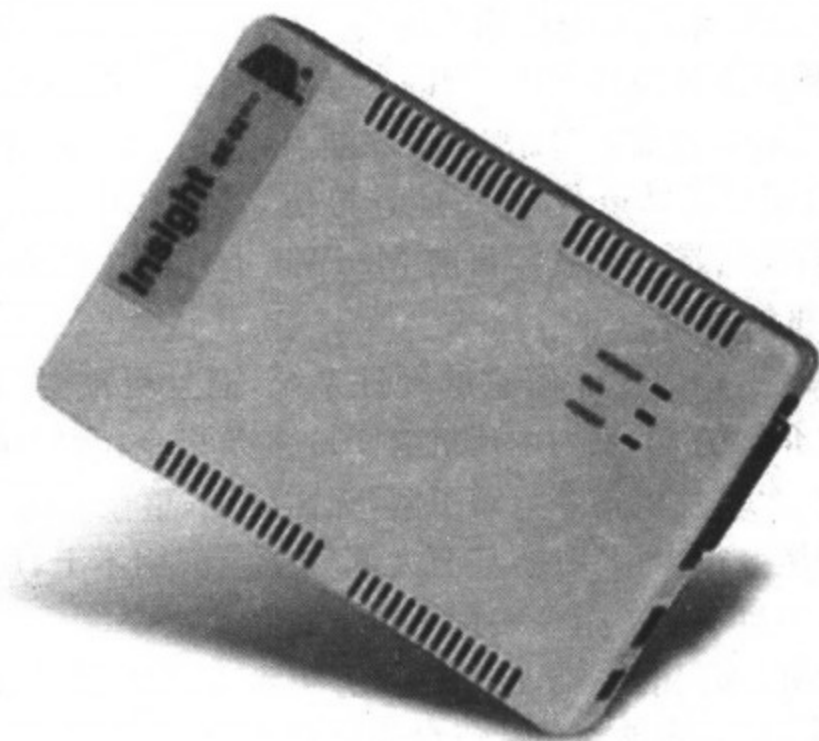


图 1-8 Insight 仿真器外观图

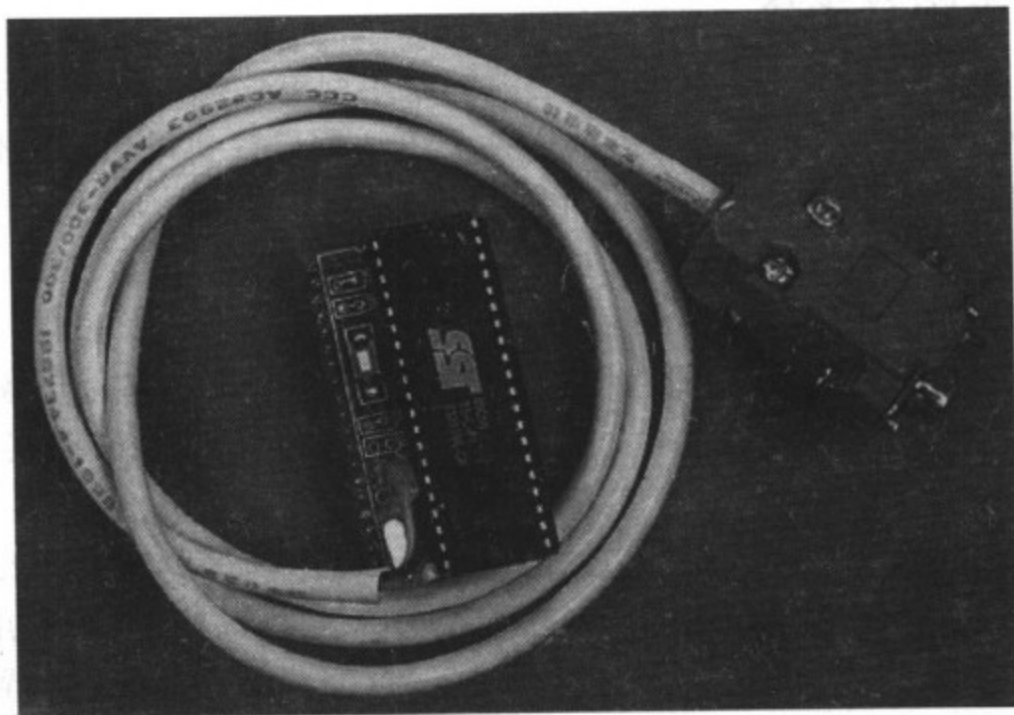


图 1-9 MON 51 简易在线仿真器外观

所以必须配合 KEIL 51 软件才能工作。

4. 软件的准备

现在,普遍采用基于 Windows 的单片机集成开发系统(IDE)进行单片机应用程序的开发,它是指将编辑、编译/汇编、连接、调试等开发单片机所要用的程序集成到一个系统软件中,如德国的 Keil C51 软件、南京万利电子有限公司(Manley)的 MedWin 等。

Keil C51 软件是众多单片机应用开发的优秀软件之一,它集编辑、编译、仿真于一体,支持汇编、PLM 语言和 C 语言的程序设计,界面友好,易学易用。Keil C51 软件带一个集成开发环境 μ Vision2, μ Vision2 是一种文件管理编译环境,集成了文本编辑处理、编译链接、项目管理、窗口、工具引用和软件仿真调试等多种功能,是相当强大的 C51 开发工具。在 μ Vision2 的仿真功能中,有两种仿真模式:软件模拟方式和目标板调试方式。在软件模拟方式下,不需要任何 51 单片机硬件即可完成用户程序仿真调试,极大地提高了用户程序开发效率。在目标板调试方式下,用户可以将程序装到自己的 51 单片机系统板

上,利用 51 的串口与 PC 机进行通信来实现用户程序的实时在线仿真。

MedWin 是万利公司开发的具有 Microsoft Visual Studio 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、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,即可进行软件仿真,又可配合万利仿真器进行实时在线仿真。目前 MedWin 有免费中文版,很适合初学者使用。

5. 掌握好单片机的指令

要进入单片机领域,不但要了解它的硬件系统,而且更重要的是掌握它的软件系统(指令),如本书重点介绍的 51 系列单片机有 100 多条指令,利用单片机的指令就可以编程了,初学单片机时,对单片机指令只需了解,无需死记,重在使用,事实上,把单片机用在某种用途时,它们的指令很少用完过,甚至大部分指令都用不上,所以初学者只需先找几条指令来实际操作,一旦掌握了指令编程后再学习其他指令。

总之,学习单片机,入门既不难,深造也是办得到的,只要你有恒心、有决心,按照本书的内容一步步走下去,就一定能在单片机世界里自由遨游。

二、单片机的开发步骤

MCS-51 系列单片机是由用户决定其功能的通用型单片机。由于单片机内部电路复杂,外部观察点少,因此,传统的万用表加烙铁已不适用单片机的电子制作和产品开发。一般来说,单片机除了根据功能要求进行硬件电路设计和制作外,还要进行软件的设计、调试和软、硬件联调。具体来说,开发步骤有以下几点。

1. 硬件系统设计

单片机是一个功能强大的智能微控制器,它把传统的单元功能电路高密度集成在一片集成电路内,因此,用单片机开发产品简化了硬件电路,很容易实现产品的小型化、智能化。

一个单片机应用系统的硬件设计包括两大部分内容:一是单片机系统的扩展部分设计,它包括存储器扩展和接口扩展;二是各功能模块的设计,如信号测量功能模块、信号控制功能模块、人机对话功能模块、通信功能模块等,根据系统功能要求配置相应的 A/D、D/A、键盘、显示器、打印机等外围设备。

在进行硬件的设计时,所涉及到的具体电路可借鉴他人在这方面进行的工作。因为经过别人调试和考验过的电路往往具有一定的合理性。如果在此基础上,结合自己的设计目的进行一些修改,则是一种简便、快捷的做法。为使硬件设计尽可能合理,在设计时还应注意以下几个方面:

(1)尽可能选择标准化、模块化的典型电路,提高设计的成功率和结构的灵活性。

(2)在条件允许的情况下,尽可能选用功能强、集成度高的电路或芯片。因为采用这种器件可能代替某一部分电路,不仅元件数量、接插件和相互连线减少,使系统可靠性增加,而且成本往往比用多个元件实现的电路要低。

(3)注意选择通用性强、市场货源充足的元器件,尤其对需大批量生产的场合,更应注意这方面的问题。其优点是,一旦某种元器件无法获得,也能用其他元器件直接替换或对电路稍作改动后用其他器件代替。

(4)应尽量选用通用接口方式,如采用 STD 总线结构、PC 总线结构等。因为对于这

些总线结构的接口方式应用比较广泛,不少厂家已开发出了适合于这些总线结构的接口板,如输入板、输出板、A/D板等。在必要的情况下,选用现成的模块板作为系统的一部分,尽管成本有些偏高,但会大大缩短研制周期,提高工作效率。当然,在有些特殊情况和小系统的场合,用户必须自行设计接口,定义连线方式。此时要注意接口协议,一旦接口方式确定下来,各个模块的设计者应遵守该接口方式。

(5)系统的扩展及各功能模块的设计在满足应用系统功能要求的基础上,应适当留有余地,以备将来修改、扩展之需。实际上,电路设计一次成功而不作任何修改的情况是很少的,如果在设计之初未留有任何余地,后期很可能因为一点小小的改动或扩展而被迫进行全面返工。举例来说,在进行ROM扩展时,尽量选用2764以上的芯片,这样不仅将来升级方便,成本也会降低;在进行RAM扩展时,为使系统升级或增加内存方便,系统的RAM空间应留足位置,哪怕多设计一个RAM插座,不插芯片也好;在进行I/O接口扩展时,也应给出一定的余量,这样对临时增加一些测量通道或被控对象就极为方便了。在电路板设计时,可适当安排一些机动布线区,在此区域中安排若干集成芯片插座和金属化孔,但不布线,这样在样机研制过程中,若发现硬件电路有不足之处,需增加元器件时,可在机动布线区临时连线来完成,从而避免整个系统返工。在进行模拟信号处理电路设计时,尤其要注意这一点。因为在设计这类电路时,经常会增加一些电容、电阻等元器件。当然,一旦试验完成,制作正式电路板时,可以去掉机动布线区。

(6)设计时应尽可能地做些调研,采用最新的技术。因为电子技术发展迅速,器件更新换代很快,市场上不断推出性能更优、功能更强的芯片,只有时刻注意这方面的发展动态,采用新技术、新工艺,才能使产品具有最先进的性能,不落后于时代发展的潮流。

(7)在电路设计时,要充分考虑应用系统各部分的驱动能力。一些经验欠缺者往往忽视电路的驱动能力及时序问题,认为原理上通就行了,其实不然。因为不同的电路有不同的驱动能力,对后级系统的输入阻抗要求也不一样。如果阻抗匹配不当,系统驱动能力不够,将导致系统工作不可靠甚至无法工作。

注意 这种不可靠很难通过一般的测试手段来确定而排除这种故障,往往需要对系统做较大的调整。因此,在电路设计时,要注意增加系统的驱动能力或减少系统的功耗。

(8)工艺设计,包括机箱、面板、配线、接插件等,这也是一个初次进行系统设计人员容易疏忽但又十分重要的问题。在设计时要充分考虑到安装、调试、维修的方便。

2. 编写应用程序

想让单片机按你的意思(想法)完成一项任务,必须先编写供其使用的程序,编写单片机的程序应使用该单片机可以识别的“语言”,否则将是对“机”弹琴。目前较流行的有汇编和C语言;汇编语言可以精确的控制单片机工作的每一步,而C语言则注重结果,不必关心单片机具体的每一步。习惯上宜先学汇编语言后学C语言,这样可以对单片机有一个更深的了解,再说,就是用C语言编程,在需要精确控制时还需要嵌入汇编语句。当然,也有一开始就用C语言的,后来再学汇编;若学过计算机的Turbo C,而再学单片机的C语言也许会更快一些。

单片机程序是用文本编辑器编写的纯文本文件,像我们平常在Windows记事本中用汉语写计划一样,按照单片机语言的语法,将编程者的想法把单片机要做的事“一件一件”

地依次写下来,写完后,将文件进行保存,文件的扩展名应与所使用的语言要求的名字一致:汉语文章一般保存为.txt扩展名,而汇编语言的文件扩展名一般为.asm;有的开发系统则有自己的规定,要求编写的汇编程序扩展名为.a51。

对于8051系列单片机来说,Keil C开发系统具有编辑、编译、模拟单片机C语言程序的功能,也能编辑、编译、模拟汇编语言程序;对于初学者,开始编写的程序难免出现语法错误或其他不规范的语句,由于Keil C编译时对错误语句有明确的提示,因此,十分方便程序的修改和维护。

设计人员在进行程序设计时,应从以下几个方面加以考虑:

(1)根据软件功能要求,将系统软件分成若干个相对独立的部分。根据它们之间的联系和时间上的关系,设计出合理的软件总体结构,使其清晰、简洁、流程合理。

(2)培养结构化程序设计风格,各功能程序实行模块化、子程序化。既便于调试、连接,又便于移植、修改。

(3)为提高软件设计的总体效率,以简明、直观的方法对任务进行描述,在编写应用软件之前,应绘制出程序流程图。这不仅是程序设计的一个重要组成部分,而且是决定成败的关键部分。从某种意义上讲,多花一份时间来设计程序流程图,就可以节约几倍源程序编辑调试时间。

(4)要合理分配系统资源,包括ROM、RAM、定时/计数器、中断源等。其中最关键的是片内RAM分配。当RAM资源规划好后,应列出一张RAM资源详细分配表,以备编程查用方便。

(5)注意在程序的有关位置处写上功能注释,提高程序的可读性。

3. 编译/汇编源程序

无论使用汇编语言(扩展名为.asm或.a51),还是C语言(扩展名为.c)编写的程序,都还必须经过与该语言对应的软件将我们能看懂的汇编或C“翻译”(编译)成所用单片机可以识别的机器码。

要将编写的源程序转变成CPU可以执行的机器码,可采用手工汇编和机器汇编的方法。目前手工汇编的方法已被淘汰。机器汇编是指通过汇编软件将源程序变为机器码,用于MCS-51单片机的汇编软件早期有A51,目前流行的Keil软件或MedWin软件,通过汇编软件(汇编器)对汇编语言源程序进行汇编,连接目标模块和库模块产生目标代码,生成.hex(十六进制)或.bin(二进制)目标文件,以使用编程器烧写到单片机中。

4. 应用程序的仿真调试

编译/汇编通过只是说明源程序没有语法错误,至于源程序中存在的其他错误,往往还需要通过反复的仿真调试才能发现。所谓仿真即是对目标样机进行排错、调试和检查,一般分为硬件仿真和软件仿真两种。

硬件仿真是通过仿真器(仿真机)与目标样机联机进行实时在线仿真,如图1-10所示。一块单片机应用电路板包括单片机部分及为达到使用目的而设计的应用电路。硬件仿真就是利用仿真器来代替应用电路板(称目标样机)的单片机部分,由仿真器向目标样机的应用电路部分提供各种信号、数据进行测试、调试的方法。这种仿真可以通过单步执行、连续执行等多种方式来运行程序,并能观察到单片机内部的变化,便于修改程序中的错误。图中,将仿真插头插到电路板上的单片机插座上,此时可将仿真器看做是一个独立

的单片机,通过运行 PC 机上的仿真软件(如 KeilC 51 软件),使目标样机处于一个真实的工作环境之中,可模拟开发单片机的各种功能。显然,这种仿真因为需要仿真器、电路板等硬件装置,因而投资较大。

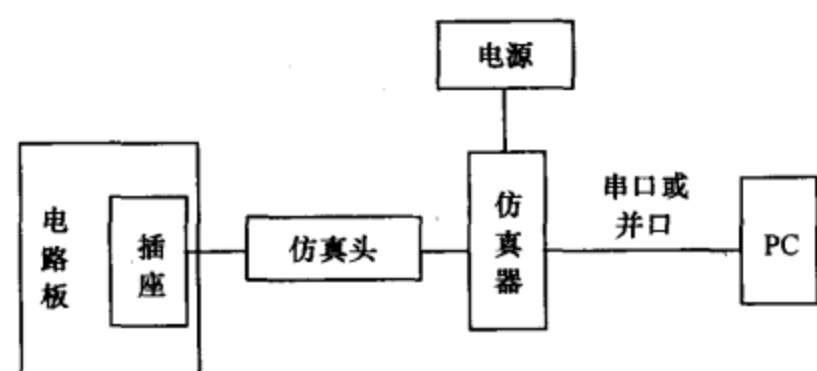


图 1-10 硬件仿真连接图

软件仿真是指在 PC 机上运行仿真软件来实现对单片机的硬件模拟、指令模拟和运行状态模拟,故这种仿真方法又称为软件(程序)模拟调试。它不需要硬件,简单易行,可采用 Keil、MedWin 或 8051DEBUG 等软件进行模拟调试。软件仿真的缺点是不适用于实时性很强的单片机应用系统的调试,在实时性要求不高的场合,软件仿真已被广泛应用。

5. 单片机应用程序的烧写

仿真调试通过后,要将扩展名为 .hex 或 .bin 的代码文件送到单片机系统中,单片机在电路中才能按设计者的“计划”去工作。

使用编程器可将 .hex 或 .bin 代码文件烧写到单片机(或外接的可编程 ROM)中。编程器一般通过并口或串口与 PC 连接,具有相应的服务程序;在连接好 PC 与编程器后运行其服务程序,在服务程序中先选择所要编程的单片机型号,再调入前面所得到的 .hex 目标文件,编程器就将这个目标文件烧写到单片机中。烧写后再将单片机插入到电路板上相应的插座上,如符合设计要求,则完成工作。

6. 系统脱机运行检查

设计者不可能一次就将其“计划”用单片机的语言完美正确地写好源程序,而需要反复修改源程序,反复编译、烧写到单片机中,反复将单片机装到电路中去实验,针对硬件或软件出现的问题,进行修改,逐步进行完善。



第二章 单片机的组成

本章主要以 Intel 公司生产的 8 位 80C51 单片机为例,介绍单片机的内部结构、外部引脚、存储器配置、并行 I/O 端口、外围时钟和复位电路等内容,最后对目前应用十分广泛的 AT89C51、AT89C1051 和 AT89C2051 的组成进行简要介绍。

第一节 80C51 单片机的内部结构和外部引脚

一、80C51 单片机的内部结构框图

80C51 单片机内部包含中央处理器(CPU)、程序存储器(ROM)、数据存储器(RAM)、定时/计数器、并行接口、串行接口和中断系统等几大单元及数据总线、地址总线和控制总线等三大总线,其内部结构框图如图 2-1 所示。

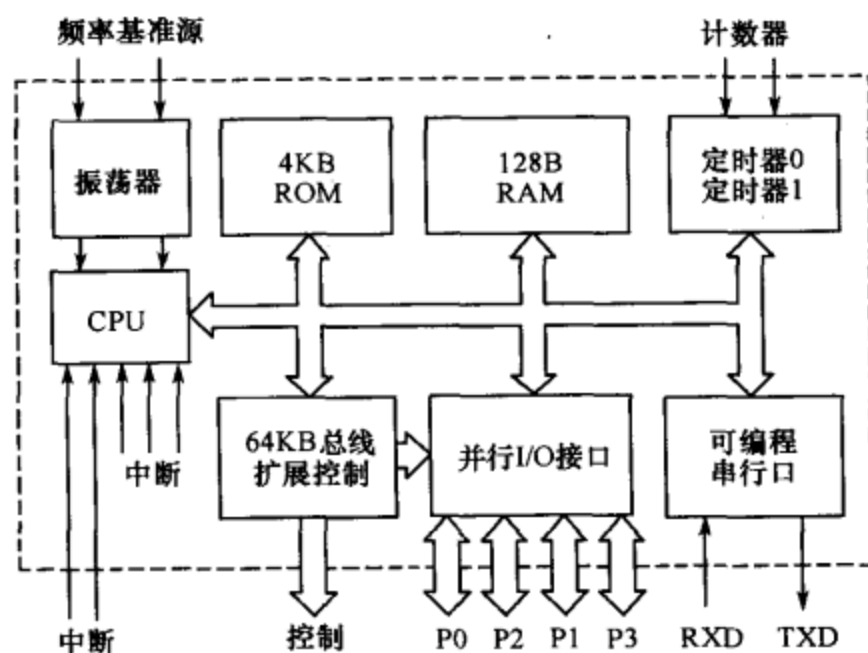


图 2-1 80C51 单片机内部结构框图

从框图中可以看出,80C51 单片机虽然只是一个芯片,但“麻雀虽小,五脏俱全”,作为计算机应该具有的基本部件在单片机内部几乎都包括,因此,80C51 单片机实际上已经是一个简单的微型计算机系统。下面对主要部件分别加以说明。

1. 中央处理器

中央处理器(CPU)是整个单片机的核心部件,是 8 位数据宽度的处理器,能处理 8 位二进制数据或代码,CPU 负责控制、指挥和调度整个单元系统协调的工作,完成运算和控制输入输出功能等操作。中央处理器主要由运算器和控制器两部分组成。

(1)运算器。运算器是单片机的运算部件,用于实现算术和逻辑运算。运算器主要由算术/逻辑运算部件 ALU、暂存器 TMP、累加器 ACC、寄存器 B、程序状态标志寄存器

PSW 及布尔处理器(位处理器)等组成。累加器 ACC 是一个 8 位寄存器,它是 CPU 中工作最频繁的寄存器。在进行算术、逻辑运算时,累加器 ACC 往往在运算前暂存一个操作数(如被加数),而运算后又保存其结果(如代数和)。寄存器 B 主要用于乘法和除法操作。标志寄存器 PSW 也是一个 8 位寄存器,用来存放运算结果的一些特征,如有无进位、借位等。

(2)控制器。控制器是单片机的指挥控制部件,保证单片机各部分能自动而协调地工作。控制器主要包括定时控制逻辑电路、指令寄存器、译码器、地址指针 DPTR 及程序计数器 PC、堆栈指针 SP 等。

这里程序计数器 PC 是由 16 位寄存器构成的计数器。要单片机执行一个程序,就必须把该程序按顺序预先装入存储器 ROM 的某个区域。单片机动作时应按顺序一条条取出指令来加以执行。因此,必须有一个电路能找出指令所在的单元地址,该电路就是程序计数器 PC。当单片机开始执行程序时,给 PC 装入第一条指令所在地址,它每取出一条指令(如为多字节指令,则每取出一个指令字节),PC 的内容就自动加 1,以指向下一条指令的地址,使指令能顺序执行。只有当程序遇到转移指令、子程序调用指令,或遇到中断时,PC 才转到所需要的地方去。

80C51 CPU 按 PC 指定的地址,从 ROM 相应单元中取出指令字节放在指令寄存器中寄存,然后,指令寄存器中的指令代码被译码器译成各种形式的控制信号,这些信号与单片机时钟振荡器产生的时钟脉冲在定时与控制电路中相结合,形成按一定时间节拍变化的电平和时钟,即所谓控制信息,在 CPU 内部协调寄存器之间的数据传输、运算等操作。

重点提示 在单片机中,基本上有 3 类信息在流动,第 1 类是数据,即各种原始数据、中间结果、程序等。这样要由外部设备通过“口”进入单片机,再存放在存储器中,在运算处理过程中,数据从存储器读入运算器进行运算,运算的中间结果要存入存储器中,或最后由运算器经“口”输出。第 2 类信息是用户要单片机执行的各种命令(程序),它们也以数据的形式由存储器送入控制器,由控制器解读(译码)为各种控制信号,以便执行加、减、乘、除等功能,所以,这一类信息就称为控制命令,即由控制器去控制运算器一步步地进行运算和处理,又控制存储器的读(取出数据)和写(存入数据)等。第 3 类信息是地址信息,其作用是告诉运算器和控制器在何处去取命令、取数据,将结果存放到什么地方,通过哪个口输入和输出信息等。

2. 存储器

存储器又分为 ROM 和 RAM 两种,前者存放调试好的固定程序和常数,后者存放一些随时有可能变动的数据。

80C51 共有 4096 个 8 位(4KB)掩模 ROM,有 128 个 8 位用户数据存储单元和 128 个专用寄存器单元,专用寄存器只能用于存放控制指令数据,用户只能访问,而不能用于存放用户数据,所以,用户能使用的 RAM 只有 128 个。

3. 定时/计数器

单片机除了进行运算外,还要完成控制功能,所以离不开计数和定时。因此,在单片机中就设置有定时器兼计数器。80C51 有两个 16 位的可编程定时/计数器,以实现定时或计数。

4. 并行输入/输出(I/O)口

80C51 共有 4 组 8 位 I/O 接口(P0、P2、P1 和 P3),用于和外部数据进行并行传输。

5. 全双工串行口

80C51 内置一个全双工串行通信口,用于与其他设备间的串行数据传送,该串行口既可以用做异步通信收发器,也可以当同步移位器使用。

6. 中断系统

中断系统相当于“传达室”,当单片机控制对象的参数到达某个需要加以干预的状态时,就可经此“传达室”通报给 CPU,使 CPU 根据外部事态的轻重缓急来采取适当的应付措施。

80C51 具备较完善的中断功能,有两个外中断、两个定时/计数器中断和一个串行中断,可满足不同的控制要求,并具有两级的优先级别选择。

7. 时钟电路

单片机里面还有一个时钟电路,使单片机在进行运算和控制时,都能有节奏地进行。80C51 内置最高频率达 12MHz 的时钟电路,用于产生整个单片机运行的脉冲时序,但 80C51 单片机需外置振荡电容。

现在,我们已经知道了单片机的组成,实际上,单片机内部有一条将它们连接起来的“纽带”,即所谓的“内部总线”。而 CPU、ROM、RAM、I/O 接口、中断系统等就分布在此“总线”的两旁,并和它连通。从而,一切指令、数据都可经内部总线传送。

二、单片机的外部引脚

80C51 单片机采用 40 脚封装的双列直接 DIP 结构,其引脚配置如图 2-2 所示。

40 个引脚中,正电源和地线 2 个引脚,外置石英振荡器的时钟线 2 个引脚,4 组 8 位共 32 个 I/O 接口(中断口线与 P3 口线复用),控制引脚 4 个。现在我们对这些引脚的功能加以说明。

1. 电源和接地引脚(2 个)

V_{SS} (PIN20):接地脚。

V_{CC} (PIN40):正电源脚,正常工作或对片内 EPROM 烧写程序时,接+5V 电源。

2. 外接晶体引脚(2 个)

XTAL1(PIN19):时钟 XTAL1 脚,片内振荡电路的输入端。

XTAL2(PIN18):时钟 XTAL2 脚,片内振荡电路的输出端。

8051 单片机的时钟有两种方式,一种是片内时钟振荡方式,但需在 18 脚和 19 脚外接石英晶体(2MHz~12MHz)和振荡电容,振荡电容的值一般取 30pF。另外一种方式是外部时钟方式,即将 XTAL1 接地,外部时钟信号从 XTAL2 脚输入。对于 80C51 单

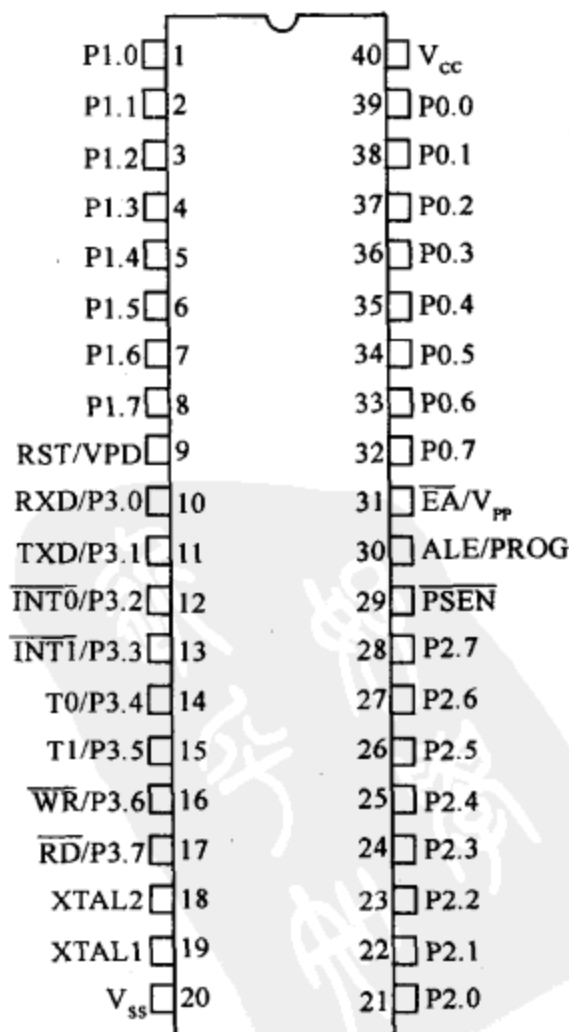


图 2-2 80C51 外部引脚配置图

片机,情况有些不同。外引脉冲信号需从 XTAL1 引脚注入,而 XTAL2 引脚悬空。

3. 输入/输出引脚(32 个)

P0.0~P0.7(PIN39~PIN32):P0 口是一个 8 位漏极开路的双向 I/O 接口,它是一个多功能口。在访问外部存储器时,用做分时多路转换地址(低 8 位)和数据总线,P0 口常用做此方式。在没有外部存储器时,P0 口可作为并行 I/O 接口使用,但需外接上拉电阻。它的带负载能力为 8 个 LSTTL 门电路。

P1.0~P1.7(PIN1~PIN8):P1 口是一个带有内部上拉电阻的 8 位准双向 I/O 接口。它通常用做通用 I/O 接口,能带动 4 个 LSTTL 门。

P2.0~P2.7(PIN21~PIN28):P2 口是一个带有内部上拉电阻的 8 位准双向 I/O 接口,它是一个多功能口。在访问外部存储器时,它送出地址的高 8 位;在没有外部存储器时,可作为通用 I/O 接口使用。可带动 4 个 LSTTL 门。

P3.0~P3.7(PIN10~PIN17):P3 口是一个带有内部上拉电阻的 8 位准双向 I/O 接口。它是一个多功能口。P3 口的第一功能是作为通用 I/O 接口,第二功能如表 2-1 所示。

表 2-1 P3 口的第二功能

引脚	第二功能	引脚	第二功能
P3.0	串行数据输入(RXD)	P3.4	定时器/计数器 0 外部输入(T0)
P3.1	串行数据输出(TXD)	P3.5	定时器/计数器 1 外部输入(T1)
P3.2	外部中断 0 输入(INT0)	P3.6	外部 RAM 写选通信号(\overline{WR})
P3.3	外部中断 1 输入(INT1)	P3.7	外部 RAM 读选通信号(\overline{RD})

4. 控制引脚(4 个)

RST/VPD(PIN9):复位信号引脚。当振荡器运行时,在此引脚上出现两个机器周期以上的高电平将使单片机复位。一般在此引脚与 V_{SS} 之间连接一个下拉电阻,与 V_{CC} 引脚之间连接一个电容。此外,RST/VPD 还是一复用脚, V_{CC} 掉电期间,此脚可接上备用电源,当电源电压下降到下限值时,备用电源经此端向内部 RAM 提供电压,以保证内部 RAM 中的信息不丢失。

ALE/PROG(PIN30):地址锁存允许信号。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。即使不访问外部存储器,ALE 端仍以不变的频率周期性地输出脉冲信号,此频率为石英晶振荡频率的 1/6。因此,它可用做对外输出的时钟,或用于定时的目的。另外,ALE/PROG 还是一个复用脚,在编程其间,PROG 将用于输入编程脉冲。

\overline{PSEN} (PIN29):外部程序存储器的读选通信号。在读外部 ROM 时, \overline{PSEN} 有效(低电平),以实现对外部程序存储器的读操作。

\overline{EA}/V_{PP} (PIN31):程序存储器的内外部选通线。当 \overline{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 \overline{EA} 接高电平时,对 ROM 的读操作(执行程序)从内部开始。对于 80C51 单片机,内置有 4KB 的程序存储器,当 \overline{EA} 为高电平并且程序地址小于 4KB 时,读取内部程序存储器指令数据,而超过 4KB 地址则读取外部指令数据。如 \overline{EA} 为低电平,则不管地址大小,一律读取外部程序存储器指令。显然,对内部无程序存储器的 8031, \overline{EA} 端必须接地。另外, \overline{EA}/V_{PP} 还是一个复用脚,在编程时, V_{PP} 脚需加

上 21V 的编程电压。

重点提示 由于工艺及标准化等方面的原因,芯片的引脚数目是有限的,例如,80C51 把引脚数目限定为 40 条,但单片机为实现其功能所需要的信号数目都远远超过此数,因此就出现了需要与可能的矛盾。为了解决这个矛盾,给一些信号赋予双重功能即“兼职”。如果把前述信号定义为第一功能,则根据需要再定义的信号就是第二功能。像 30 脚(ALE/PROG),31 脚(\overline{EA} / V_{PP})和 9 脚(RST/VPD)都具有第二功能。

一个信号引脚,是第一功能又是第二功能,会不会在使用时引起混乱造成错误呢?不会的。对于 9 脚、30 脚和 31 各引脚,由于第一功能信号与第二功能信号是单片机在不同工作方式下的信号,因此不会发生使用上的矛盾。对于 P3 口线,在实际使用时,总是先按需要优先选用它的第二功能,剩下不用的才作为口线使用。

第二节 80C51 单片机内部存储器的配置

存储器是单片机的又一个重要组成部分,图 2-3 给出了 80C51 存储器的配置图,从图中可以看出,80C51 的存储器主要有 4 个物理存储空间,即片内程序存储器、片外程序存储器、片内数据存储器 and 片外数据存储器。

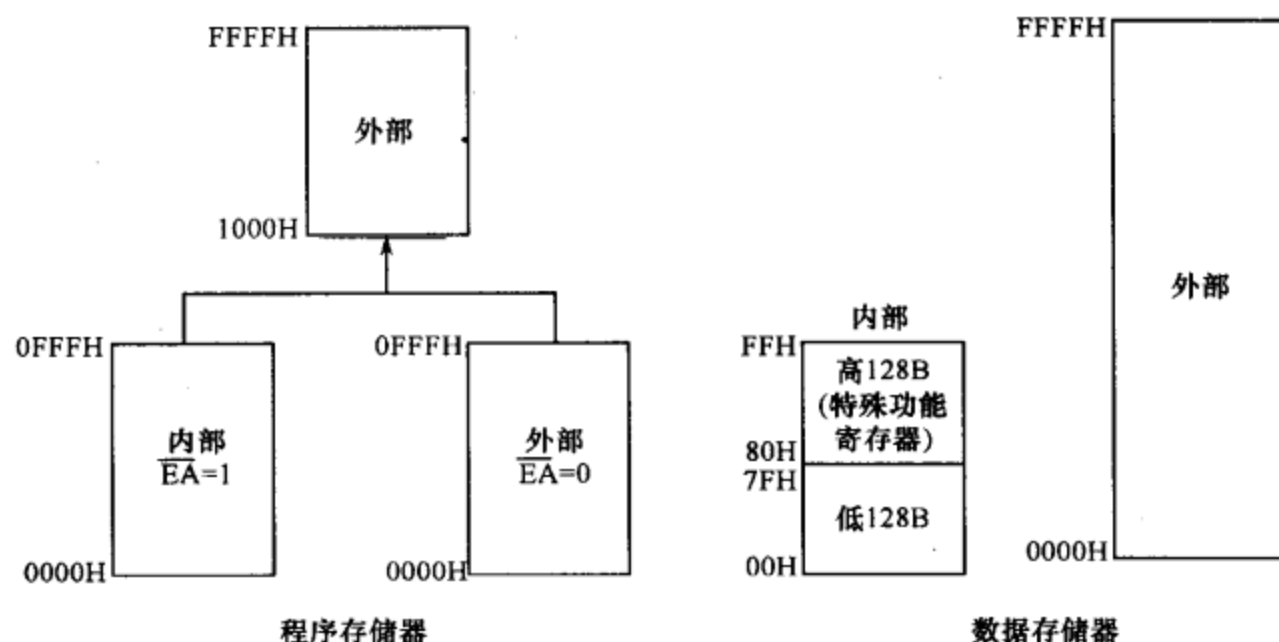


图 2-3 80C51 存储器配置图

重点提示 从逻辑上分,80C51 有 3 个存储器地址空间,即片内统一编址的 64KB 的程序存储器地址、256B 的内部数据存储器地址空间(其中高 128B 为特殊功能寄存器,仅低 128B 用于内部数据存储器)和 64KB 的外部数据存储器地址空间。为了区分不同的存储空间,在访问这 3 个不同的逻辑空间时,采用了不同形式的指令,具体指令形式将在后续章节中进行介绍。

一、程序存储器

单片机能够聪明地执行某种任务,除了它们强大的硬件外,还需要它们运行的软件,其实,单片机并不聪明,它们只是完全按照人们预先编写的程序而执行之。

程序是控制计算机动作的一系列命令,单片机只认识由“0”和“1”代码构成的机器指

令。如用助记符编写的命令 MOV A, #20H(这个指令的意思是将立即数 20H 送到 A 中),换成机器认识的代码为 74H、20H,写成二进制就是 01110100B 和 00100000B。在单片机处理问题之前,必须事先将编好的程序、表格、常数汇编成机器代码后存入单片机的程序存储器中。

程序存储器可以放在片内或片外,也可片内、片外同时设置。由于 MCS-51 单片机 PC 程序计数器为 16 位,使得程序存储器可用 16 位二进制地址,因此,内外存储器的地址为 0000H~FFFFH。对于内部无 ROM 的 8031 单片机,它的程序存储器必须外接,空间地址为 64KB,此时单片机的 \overline{EA} 端必须接地。强制 CPU 从外部程序存储器读取程序。对于内部有 ROM 的 80C51 等单片机,正常运行时, \overline{EA} 则需接高电平,使 CPU 先从内部的程序存储器中读取程序,当 PC 值超过内部 ROM 的容量时,才会转向外部的程序存储器读取程序。

8051 片内有 4KB 的程序存储单元,其地址为 0000H~0FFFH,单片机启动复位后,程序计数器的内容为 0000H,所以系统将从 0000H 单元开始执行程序。但在程序存储器中有些特殊的单元,这在使用中应加以注意:

一组是 0000H~0002H 单元,系统复位后,PC 值为 0000H,单片机从 0000H 单元开始执行程序,如果程序不是从 0000H 单元开始,则应在这 3 个单元中存放一条无条件转移指令,让 CPU 直接去执行用户指定的程序。

另一组特殊单元是 0003H~002AH,这 40 个单元各有用途,它们被均匀地分为五段,它们的定义如下:

0003H~000AH	外部中断 0 中断地址区
000BH~0012H	定时器/计数器 0 中断地址区
0013H~001AH	外部中断 1 中断地址区
001BH~0022H	定时器/计数器 1 中断地址区
0023H~002AH	串行中断地址区

可见,以上的 40 个单元是专门用于存放中断处理程序的地址单元,中断响应后,按中断的类型,自动转到各自的中断区去执行程序。因此以上地址单元不能用于存放程序的其他内容,只能存放中断服务程序。但是通常情况下,每段只有 8 个地址单元是不能存下完整的中断服务程序的,因而一般也在中断响应的地址区,安放一条无条件转移指令,指向程序存储器的其他真正存放中断服务程序的空间去执行,这样中断响应后,CPU 读到这条转移指令,便转向其他地方去继续执行中断服务程序。

二、数据存储器

单片机的数据存储器由读写存储器 RAM 组成,用于存储实时输入的数据。80C51 的存储器分为内部数据存储器 and 外部数据存储器,其最大容量可扩展到 64KB,实际使用时,应首先充分利用内部存储器。

1. 内部数据存储器

内部存储器的存储容量为 256 个单元。其中每个存储单元对应一个地址,256 个单元共有 256 个地址,用两位 16 进制数表示,即存储器的地址(00H~FFH)。存储器中每个存储单元可存放一个 8 位二进制信息,通常用两位 16 进制数来表示,这就是存储器的

内容。存储器的存储单元地址和存储单元的内容是两个不同的概念,不能混淆。

从使用角度讲,搞清内部数据存储器的结构和地址分配是十分重要的。因为将来在学习指令系统和程序设计时会经常用到它们。80C51 内部 256B 的数据存储器被分为两部分,其中,内部数据 RAM 的地址为 00H~7FH(即 0~127),而用做特殊功能寄存器的地址为 80H~FFH。

1) 数据存储器(低 128 单元)

数据存储器的低 128 单元是供用户使用的数据存储单元,一般称之为内部 RAM 存储器,其配置情况如图 2-4 所示。

按用途可将低 128 单元分为 3 个区域。

(1)通用寄存器区。在 00H~1FH 共 32 个单元中被均匀地分为四块,每块包含 8 个 8 位寄存器,均以 R0~R7 来命名,称这些寄存器为通用寄存器。这四块中的寄存器都称为 R0~R7,那么在程序中怎么区分和使用它们呢? 聪明的工程师们又安排了一个寄存器——程序状态字寄存器(PSW)来管理它们,CPU 只要定义这个寄存的 PSW 的第 3 位和第 4 位(RS0 和 RS1),即可选中这四组通用寄存器。对应的编码关系如表 2-2 所列。

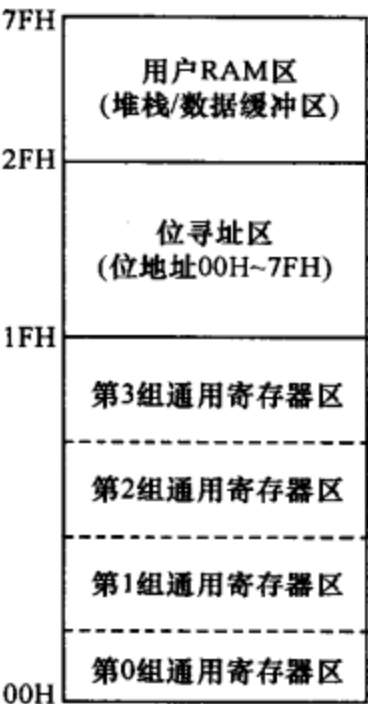


图 2-4 内部数据存储器低 128 单元配置图

表 2-2 PSW 与通用寄存器的对应关系

RS1	RS0	寄存器组	R0~R7 地址	RS1	RS0	寄存器组	R0~R7 地址
0	0	组 0	00~07H	1	0	组 2	10~17H
0	1	组 1	08~0FH	1	1	组 3	18~1FH

重点提示 从表中可以看出,在任一时刻,CPU 只能使用其中的一组寄存器,并且把正在使用的那组寄存器称之为当前寄存器区。到底选择哪一个工作组为当前工作区,取决于程序状态字寄存器 PSW 中的 RS1 和 RS0 位的状态。RS1 和 RS0 的状态可通过指令来改变。用户可以通过设置 RS1 和 RS0 位的状态来选择和切换当前工作寄存器区,这给用户保护寄存器中的内容提供了极大的方便。

(2)位寻址区。内部 RAM 的 20H~2FH 单元为位寻址区,既可作为一般单元用字节寻址,也可对它们的位进行寻址。位寻址区共有 16B,128 位,位地址为 00H~7FH。位地址分配如表 2-3 所列,CPU 能直接寻址这些位,执行例如置“1”、清“0”、求“反”、转移、传送和逻辑等操作。我们常称 MCS-51 具有布尔处理功能,布尔处理的存储空间指的就是这些位寻址区。

表 2-3 内部 RAM 位寻址区的位地址

单元地址	位 地 址							
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	77H	76H	75H	74H	73H	72H	71H	70H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H

(续)

单元地址	位 地 址							
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	47H	46H	45H	44H	43H	42H	41H	40H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	37H	36H	35H	34H	33H	32H	31H	30H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	27H	26H	25H	24H	23H	22H	21H	20H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	17H	16H	15H	14H	13H	12H	11H	10H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	07H	06H	05H	04H	03H	02H	01H	00H

注意 位地址 7CH 与字节地址 7CH 的区别

位地址 7CH 表示 7CH 这一二进制位的地址,字节地址 7CH 表示地址为 7CH 的单元地址。位地址 7CH 在内存中 2FH 单元的第 4 位。

(3)用户 RAM 区。在内部 RAM 低 128 单元中,通用寄存器占去 32 个单元,位寻址区占去 16 个单元,剩下的 60 个单元就是供用户使用的一般 RAM 区,地址单元为 30H~7FH。对这部分区域的使用不作任何规定和限制,但应当说明的是,堆栈一般开辟在此区。

2)特殊功能寄存器 SFR(高 128 单元)

内部 RAM 的高 128 单元是给特殊寄存器使用的,因此称之为专用寄存器区,其单元地址为 80H~FFH。因为这些寄存器的功能已作专门规定,所以称其为专用寄存器或特殊功能寄存器(SFR)。特殊功能寄存器的总数为 22 个,其中可寻址的 21 个,不可寻址的 1 个。特殊功能寄存器的配置情况如图 2-5 所示。

下面把几个主要特殊功能寄存器作一简要介绍。

(1)程序计数器 PC:PC 是一个 16 位的计数器。其内容为将要执行的指令地址,寻址范围达 64KB。PC 有自动加 1 功能,从而实现程序的顺序执行。PC 没有地址,是不可寻址的(但在物理上是存在的),因此用户无法对它进行读/写;但可以通过转移、调用、返回

(高128单元)		
FFH	B	特殊功能寄存器 SFR
F0H	ACC	
E0H	PSW	
D0H	IP	
B8H	P3	
B0H	IE	
A8H	P2	
A0H	SBUF	
99H	SCON	
98H	P1	
90H	TH1	
8DH	TH0	
8CH	TL1	
8BH	TL0	
8AH	TMOD	
89H	TCON	
88H	PCON	
87H	DPH	
83H	DPL	
82H	SP	
81H	P0	
80H		

图 2-5 特殊功能寄存器配置图

等指令改变其内容,以实现程序的转移。

(2)累加器 A(或 ACC):累加器 A 为 8 位寄存器,是最常用的专用寄存器,功能较多。它既可用于存放操作数,也可用来存放中间结果。80C51 单片机中大部分单操作数指令的操作数就取自累加器,许多双操作数指令中的一个操作数也取自累加器。加、减、乘、除运算指令的运算结果都存放在累加器 A 或 AB 寄存器对中。

(3)B 寄存器:B 寄存器是一个 8 位寄存器,主要用于乘除运算。乘法运算时,B 是乘数,乘法操作后,乘积的高 8 位存于 B 中。除法运算时,B 存放除数;除法操作后,余数存于 B 中。此外,B 寄存器也可作为一般数据寄存器使用。

(4)程序状态字 PSW: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。其中,有些位的状态是程序执行的结果,是由硬件自动置位的;而有些位的状态则采用软件的方法来设定。PSW 的位状态可以用专门指令进行测试,也可以用指令读出。一些条件转移指令会根据 PSW 有关位的状态进行程序转移。PSW 的各位含义如表 2-4 所列。其中 PSW.1 为保留位,未用。

表 2-4 程序状态字 PSW 的含义

位序	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
位含义	CY	AC	F0	RS1	RS0	OV	—	P

①CY(PSW.7):进位标志位。80C51 中的运算器是一种 8 位的运算器,我们知道,8 位运算器只能表示 0~255,如果做加法,两数相加可能会超过 255,这样最高位就会丢失,造成运算的错误,有了 CY,最高位就进到这里来。这样就没事了。

②AC(PSW.6):辅助进位位。当进行加法或减法操作而产生由低 4 位向高 4 位的进位或借位时,由硬件将 AC 置 1;否则,就被清除。AC 还用于十进制调整,同 DAA 指令结合起来使用。

③F0(PSW.5):用户标志位。它是用户定义的一个状态标记,可以用软件来使它置位或清除,也可用软件测试 F0 以控制程序的流向。

④RS1、RS0(PSW.4、PSW.3):当前寄存器区选择位。用软件来置位或清除,以选择和确定当前工作寄存器区。

⑤OV(PSW.2):溢出标志位。在带符号数运算中,OV=1,表示加减运算结果超出了累加器 A 所能表示的符号数的有效范围(-128~+127),即产生了溢出,因此运算结果是错误的;否则,OV=0,运算结果正确,无溢出。在乘法运行中,OV=1,表示乘积超过 255,即乘积分别放在 B(高 8 位)与 A(低 8 位)中;否则,OV=0,表示乘积只放在 A 中,B=0。在除法运行中,OV=1,表示除数为 0,除法不能进行;否则,OV=0,除数不为 0,除法可正常进行。

⑥P(PSW.0):奇偶位。每个指令周期都由硬件来置位或清除,以表示累加器 A 中 1 的个数的奇偶性。若 P=1,则累加器 A 中 1 的个数为奇数;若 P=0,则累加器 A 中 1 的个数为偶数。

(5)栈指针 SP:栈指针 SP 是一个 8 位专用寄存器。它指示出堆栈顶部在内部数据存储单元中的位置。系统复位后,SP 初始化为 07H,使得堆栈向上由 08H 单元开始。考虑到 08H~1FH 单元属于工作寄存器区,若程序设计中要用到这些区,最好把 SP 的值置为 1FH 或更大一些,一般将堆栈开辟在 30H~7FH 区域中。SP 的值越小,堆栈深度就越

深,但最大为 128B。

(6)数据指针 DPTR:数据指针 DPTR 是唯一一个 16 位专用寄存器,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,也可作为两个独立的 8 位寄存器(DPH 和 DPL)来使用。DPTR 通常用来存放 16 位地址。既可访问外部 RAM,也可访问 ROM。

80C51 单片机的特殊功能寄存器还有一些,这些寄存器将在以后内容中介绍。

特殊功能寄存器不连续地分散在内部 RAM 的高 128 单元之中,尽管其中还有许多空闲地址,但用户不能使用。在 22 个特殊功能寄存器中,有 21 个是可寻址的,这些可寻址的寄存器的名称、符号及地址如表 2-5 所列。

表 2-5 80C51 特殊功能寄存器一览表

序号	寄存器符号	寄存器地址	寄存器名称	说明
1	ACC	0E0H	累加器	可位寻址
2	B	0F0H	B 寄存器	可位寻址
3	PSW	0D0H	程序状态字	可位寻址
4	SP	81H	堆栈指示器	
5	DPL	82H	数据指针低 8 位	
6	DPH	83H	数据指针高 8 位	
7	IE	0A8H	中断允许控制寄存器	可位寻址
8	IP	0B8H	中断优先控制寄存器	可位寻址
9	P0	80H	I/O 接口 0	可位寻址
10	P1	90H	I/O 接口 1	可位寻址
11	P2	0A0H	I/O 接口 2	可位寻址
12	P3	0B0H	I/O 接口 3	可位寻址
13	PCON	87H	电源控制及波特率选择寄存器	
14	SCON	98H	串行口控制寄存器	可位寻址
15	SBUF	99H	串行数据缓冲寄存器	
16	TCON	88H	定时器控制寄存器	可位寻址
17	TMOD	89H	定时器方式选择寄存器	
18	TL0	8AH	定时器 0 低 8 位	
19	TL1	8BH	定时器 1 低 8 位	
20	TH0	8CH	定时器 0 高 8 位	
21	TH1	8DH	定时器 1 高 8 位	

从表中可以看出,80C51 单片机中,有 11 个寄存器不仅可以字节寻址,也可以进行位寻址。凡是能进行位寻址的 SFR,其特征是字节地址都能被 8 整除(字节地址的末位是 0 或 8)。80C51 单片机的位地址表如表 2-6 所列。

表 2-6 80C51 单片机的位地址表

寄存器 符号	MSB 位地址/位名称 LSB							
B	0F7H	0F6H	0F5H	0F4H	0F3H	0F2H	0F1H	0F0H
A	0E7H	0E6H	0E5H	0E4H	0E3H	0E2H	0E1H	0E0H
PSW	0D7H	0D6H	0D5H	0D4H	0D3H	0D2H	0D1H	0D0H
	CY	AC	F0	RS1	RS0	OV	—	P
IP	0BFH	0BEH	0BDH	0BCH	0BBH	0BAH	0B9H	0B8H
	—	—	—	PS	PT1	PX1	PT0	PX0
IE	0AFH	0AEH	0ADH	0ACH	0ABH	0AAH	0A9H	0A8H
	EA	—	—	ES	ET1	EX1	ET0	EX0
SCON	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
TCON	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
P0	87H	86H	85H	84H	83H	82H	81H	80H
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	97H	96H	95H	94H	93H	92H	91H	90H
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	0A7H	0A6H	0A5H	0A4H	0A3H	0A2H	0A1H	0A0H
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	0B7H	0B6H	0B5H	0B4H	0B3H	0B2H	0B1H	0B0H
	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

从表中可以看出,IP 中有 3 位、IE 中有 2 位、PSW 中有 1 位对用户无实际意义,所以直接寻址位为 82 位;再加上数据存储器中的 128 位,80C51 共计有 210 位可寻址位。

2. 外部数据存储器

当用户需处理的数据量较大而 80C51 的内部 RAM 不够用时,单片机需要在芯片外部连接数据存储器(RAM)和 I/O 接口。80C51 单片机可访问的外部 RAM 的地址空间为 0~64KB,最多可由 16 位地址线寻址;外部 RAM 与外部 I/O 接口统一编址,即 CPU 对 RAM 和 I/O 接口不加区分。

注意 尽管 80C51 单片机的存储器有内部 ROM、外部 ROM、内部 RAM 和外部 RAM,存储器空间也是重叠的,但在实际应用中不会发生混乱。对内部 ROM 和外部 ROM,单片机是通过 EA 引脚来控制的,内外统一,不会出错;对内外部 RAM 而言,通过指令 MOV 和 MOVX 加以区分。因此,用户在使用时可尽管放心,只要使用正确的指令,就能指挥单片机正常地工作。

第三节 80C51 单片机的并行 I/O 接口

80C51 有 4 组 8 位 I/O 接口:P0、P1、P2 和 P3 口,P1、P2 和 P3 为准双向口,P0 口则

为双向三态 I/O 接口,这 4 个口除了按字节寻址之外,还可以按位寻址。4 个口在结构和功能上各具特点,具有一定的差异,下面我们分别介绍这几个口线。

一、P0 口

P0 口的字节地址为 80H,位地址为 80H~87H。口的各位口线具有完全相同但又相互独立的逻辑电路,如图 2-6 所示。

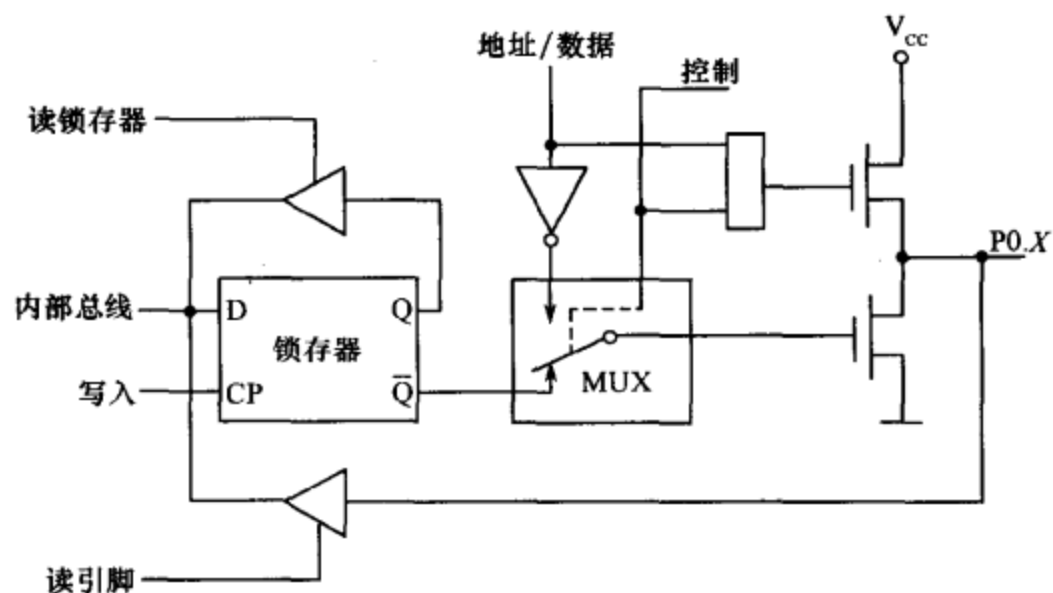


图 2-6 P0 口逻辑电路

1. P0 口的组成

P0 口逻辑电路主要由以下几部分组成：

(1)两个三态输入缓冲器。两个三态输入缓冲器分别用于锁存器数据和引脚数据的输入缓冲。

(2)一个多路转接开关 MUX。多路转接开关 MUX 的一个输入来自锁存器,另一个输入为“地址/数据”。输入转接由“控制”信号控制。之所以设置多路转接开关,是因为 P0 口既可以作为通用的 I/O 接口进行数据的输入、输出,又可以作为单片机系统的地址/数据线使用。即在控制信号的作用下,由 MUX 实现锁存器输出和地址/数据线之间的接通转接。

(3)一个数据输出锁存器。数据输出锁存器用于进行数据位的锁存。我们知道,在不同时刻,不同的部件需要不同的信号。比如某一时刻 P0.0 要求输出高电平并要求保持若干时间,在这段时间里,CPU 不能停在那里,它还需要与其他部件联络,因此这根数据线上的电平未必能保持原来的值不变,这样输出将会发生变化。为解决这一问题,在每一个输出端加一个锁存器。若要某个 I/O 接口输出数据,只要将待输出的数据写入相应的 I/O 接口(实际是写入相应的锁存器)即可;然后 CPU 就可以去做其他事情,不必再理会输出的状态了。锁存器会把数据“锁”住,直到 CPU 下一次改写数据为止。

(4)数据输出的驱动电路。数据输出的驱动电路由两只场效应管(FET)组成,上面的场效应管构成上拉电路。

2. P0 口的使用

在实际应用中,P0 口绝大多数情况下都是作为单片机系统的地址/数据线使用。当传送地址或数据时,控制信号为高电平“1”,转换开关把反相器输出端与下拉 FET 接通,

同时与门开锁。输出的地址或数据信号既通过与门去驱动上拉 FET, 又通过反相器去驱动下拉 FET。这时的输出驱动电路由于上下两个 FET 处于反相, 形成推拉式电路结构, 大大地提高了负载能力。而当输入数据时, 数据信号则直接从引脚通过输入缓冲器进入内部总线。

P0 口也可作为通用 I/O 接口使用, 这时, CPU 发来的控制信号为低电平“0”, 封锁了与门, 并将输出驱动电路的上拉场效应管截止, 而多路转接开关(MUX)接通锁存器 \bar{Q} 端的输出通路。下面简要介绍输出和输入操作的过程。

1) 输出操作(写操作)

当 P0 口作为输出口(写)使用时, 由锁存器和驱动电路构成数据输出通路。由于通路中已有输出锁存器, 因此数据输出可以与外设直接连接, 无需再加数据锁存电路。进行数据输出时, 来自 CPU 的写脉冲加在 D 触发器的 CP 端, 数据写入锁存器, 并向端口引脚输出。但要注意, 由于输出电路是漏极开路电路, 必须外接上拉电阻才能有高电平输出。

2) 输入操作(读操作)

当 P0 口作为输入口使用时, 应区分读引脚和读锁存器(端口)两种情况。为此, 在电路中有两个用于读入的三态缓冲器。

(1) 读引脚。所谓读引脚就是读芯片引脚上的数据, 也就是直接读取外部数据, 这时使用锁存器下方的缓冲器, 由“读引脚”信号把缓冲器打开, 引脚上的数据经缓冲器通过内部总线读进来。

说明 在 P0 口作为输入口读引脚使用时, 应先向锁存器写“1”(一般用传送指令), 使输出级的两个 FET 截止(系统复位时 $P0=FFH$)。

(2) 读锁存器。在端口已处于输出状态的情况下读锁存器。读锁存器是通过上方的缓冲器读锁存器 Q 端的状态。在端口已处于输出状态的情况下, 不能正常读取引脚的信号, 只能读取锁存器的状态; 这样安排的目的是适应对端口进行“读—修改—写”操作指令的需要。例如“ANL P0, A”就属于这类指令, 执行时先读入 P0 口锁存器中的数据, 然后与 A 的内容进行逻辑“与”, 再把结果送到 P0 口输出。从这种意义上说, 该指令又可看做是输出指令。除 MOV 类指令外的其他口操作指令都属于这种情况。

注意 P0 口既可作为单片机系统的地址/数据线使用, 也可作为通用 I/O 口使用。当作为通用 I/O 接口输出时, 由于输出电路是漏极开路电路, 必须外接上拉电阻才能有高电平输出。作输入使用时, 应区分读引脚和读锁存器。读引脚时, 必须先向电路中的锁存器写入“1”, 使 FET 截止, 引脚处于悬浮状态, 可作为高阻抗输入。因为, 若 FET 导通, 引脚上的电位始终被钳在“0”电平上, 输入数据不可能被正确地读入。

关于读引脚和读锁存器在后面介绍指令系统时还要详细说明。

二、P1 口

P1 口的字节地址为 90H, 位地址为 90H~97H。P1 口逻辑电路如图 2-7 所示。

P1 口只能作为通用 I/O 接口使用, 所以在电路结构上与 P0 口有一些不同。首先, 因为它只传送数据, 所以不再需要多路转接开关 MUX; 其次, 由于只用来传送数据, 因此输出电路中有上拉电阻, 上拉电阻与场效应管共同组成输出驱动电路。因为这样电路的输出不是三态的, 所以 P1 口是准双向口。

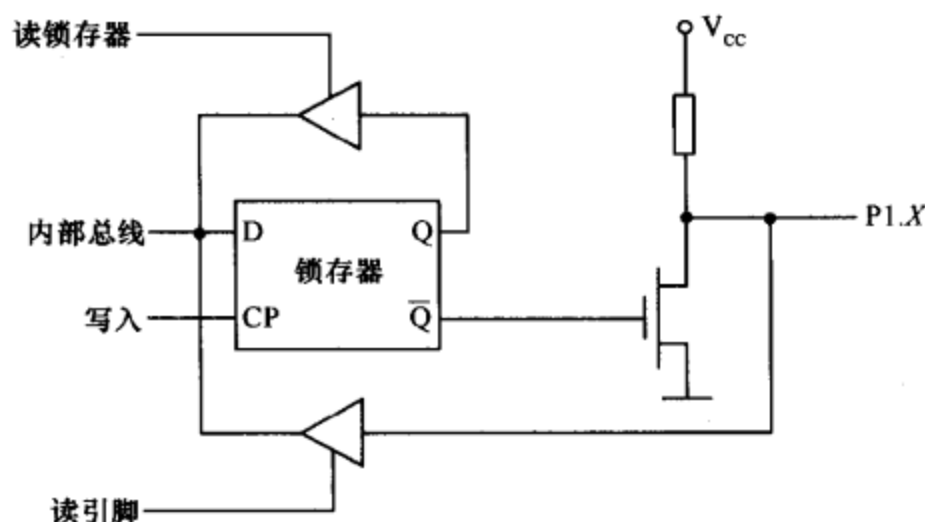


图 2-7 P1 口逻辑电路

注意 P1 口只能作为通用 I/O 接口使用,当 P1 口作为输出口使用时,已能对外提供推拉电流负载,外电路无需再接上拉电阻。当 P1 口作为输入口使用时,应区分读引脚和读锁存器,读引脚时,应先向其锁存器写入“1”,使输出驱动电路的 FET 截止。

三、P2 口

P2 口的字节地址为 0A0H,位地址为 0A0H~0A7H。P2 口的电路逻辑如图 2-8 所示。

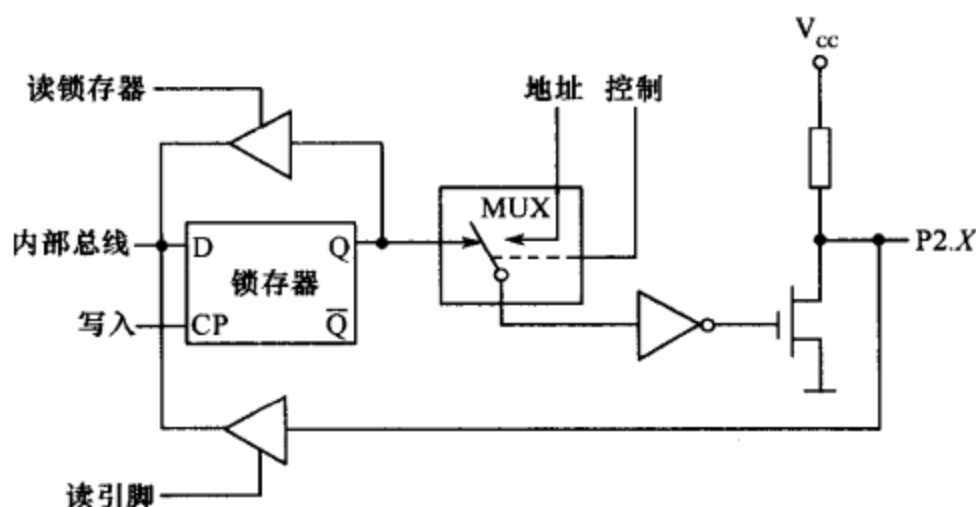


图 2-8 P2 口逻辑电路

因为在实际应用中 P2 口用于为系统提供高位地址,因此同 P0 口一样,在口电路中有一个多路转接开关 MUX。但 MUX 的一个输入端不再是“地址/数据”,而是单一的“地址”,这是因为 P2 口只作为地址线使用而不作为数据线使用。当 P2 口作为高位地址线使用时,多路转接开关应倒向“地址”端。正因为只作为地址线使用,口的输出用不着是三态的,所以 P2 口也是一个准双向口。

注意 P2 口既可作为地址总线(高位地址),也可以作为通用 I/O 接口使用,当作为通用 I/O 接口使用时,多路转接开关 MUX 倒向锁存器 Q 端。其使用方法与 P1 口类似,作输出口使用时无需接上拉电阻,作输入口使用时,应区分读引脚和读锁存器,读引脚时应先向锁存器写“1”。

四、P3 口

P3 口的字节地址为 0B0H, 位地址为 0B0H~0B7H。P3 口逻辑电路如图 2-9 所示。

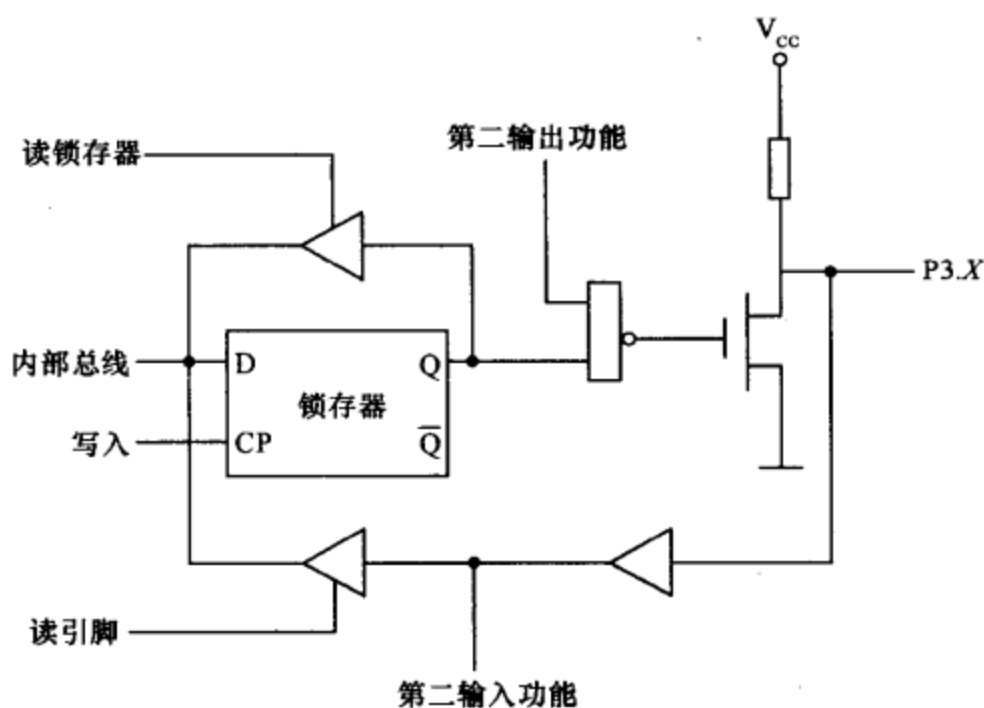


图 2-9 P3 口逻辑电路

虽然 P3 口不但可作为通用 I/O 接口使用,而且还具有第二功能,因此,P3 口在内电路中增加了第二功能控制逻辑。由于第二功能信号有输入和输出两类,因此我们分两种情况说明。

对于输出的第二功能信号引脚,当作为通用 I/O 接口使用时,电路中的“第二输出功能”信号线应保持高电平,与非门开通,以维持从锁存器到输出端数据输出通路的畅通。当输出第二功能信号时,该锁存器应预先置“1”,使与非门对第二功能信号的输出是畅通的,从而实现第二功能信号的输出。

对于第二功能为输入信号的引脚,在口线的输入通路上增加了一个缓冲器,输入的信号就从这个缓冲器的输出端取得。而作为通用 I/O 接口线使用的数据输入,仍取自三态缓冲器的输出端。

注意 P3 口既可作为第二功能使用,又可作为通用 I/O 接口使用,当某些口作为第二功能使用时,不能再把它当做通用 I/O 接口使用,但其他未使用的口线可作为通用 I/O 接口线。

P3 口作为通用 I/O 接口使用时,其使用方法与 P1 口类似。作为输出口使用时无需接上拉电阻;作输入口使用时,应区分读引脚和读锁存器,读引脚时应先向锁存器写“1”。

归纳总结 (1)P0、P1、P2、P3 都是并行 I/O 接口,都可用于数据的输入/输出传送,但 P0 口和 P2 口除了可进行数据的输入/输出外,通常是用来构建系统的数据总线和地址总线,所以,在口电路逻辑中有一个多路转接开关 MUX,以便进行两种用途的转换。而 P1 和 P3 口没有构建数据和地址总线的功能,因此在电路中没有多路转接开关 MUX。

(2)在 4 个口中,只有 P0 一个口是真正的双向口,而其余的 3 个口都是准双向口。原因是在应用系统中,P0 口作为系统的数据总线使用时,为了保证正确的数据传送,需要解决芯片内外的隔离问题,即只有在数据传送时芯片内外才接通;不进行数据传送时,芯片

内外应处于隔离状态。为此就要求 P0 口的输出缓冲器是一个三态门。

在 P0 中输出三态门是由两个场效应管(FET)组成的,所以说它是一个真正的双向口。而其他 3 个口中,上拉电阻代替了 P0 口中的场效应管,输出缓冲器不是三态的,因此不是真正的双向口,而只称其为准双向口。

(3)P3 口的口线具有第二功能,为系统提供一些控制信号。因此在 P3 口电路中增加了第二功能控制逻辑。这是 P3 口与其他各口不同之处。

第四节 80C51 单片机的时钟电路和复位电路

一、单片机的时钟电路

时钟电路用于产生单片机工作所需要的时钟信号,单片机本身就是一个复杂的同步时序电路,为了保证同步工作方式的实现,电路应在唯一的时钟信号控制下严格地按时序进行工作。

1. 时钟信号的产生

在 80C51 芯片内部有一个高增益反相放大器,其输入端为芯片引脚 XTAL1,输出端为引脚 XTAL2,在芯片的外部通过这两个引脚跨接晶体振荡器和微调电容,形成反馈电路,就构成了一个稳定的自激振荡器,如图 2-10 所示。

电路中对电容 C1 和 C2 的要求不是很严格,如使用高质的晶振,则不管频率多少, C1、C2 一般都选择 30 pF。晶体的振荡频率范围通常是 1.2 MHz~12 MHz,晶体振荡频率高,则系统的时钟频率也高,单片机运行速度也就快。振荡电路产生的振荡脉冲并不直接使用,而是经分频后再为系统所用,如图 2-11 所示。

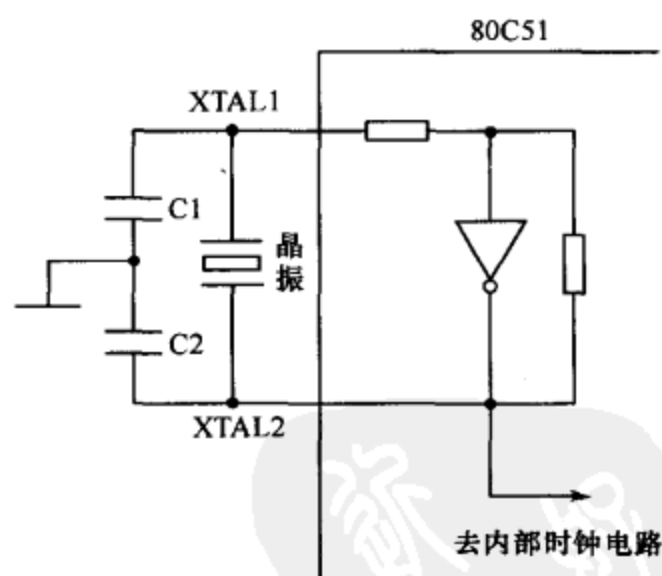


图 2-10 80C51 单片机的振荡电路

振荡脉冲经过 2 分频后作为系统的时钟信号,在 2 分频的基础上再 3 分频产生 ALE 信号,即 ALE 为晶振固定频率的 1/6,在 2 分频的基础上再 6 分频得到机器周期信号,即机器周期为晶振固定频率的 1/12。

2. 时序定时单位

时序是用定时单位来描述的,80C51 单片机的时序单位有 4 个,它们分别是节拍、状态、机器周期和指令周期,下面分别加以说明。

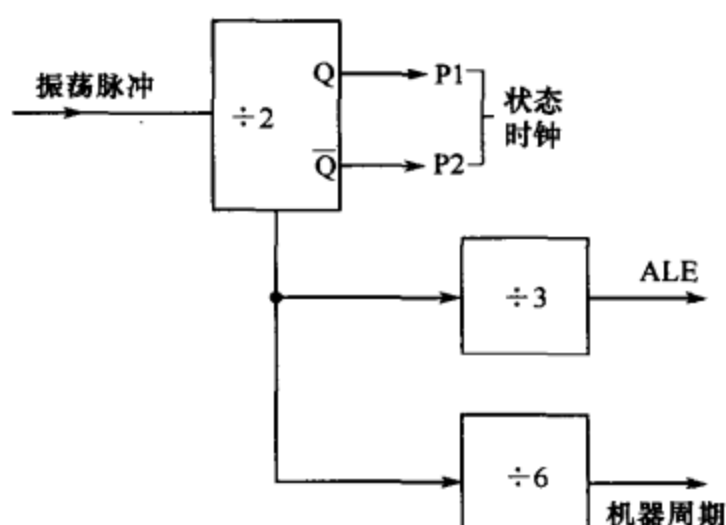


图 2-11 时钟的分频

(1)节拍与状态。把振荡脉冲的周期(也称振荡周期)定义为节拍(为方便描述,用P表示),振荡脉冲经过2分频后即得到整个单片机工作系统的时钟信号,把时钟信号的周期定义为状态(用S表示),这样一个状态就有两个节拍,前半周期相应的节拍定义为节拍1(P1),后半周期对应的节拍定义为节拍2(P2)。

(2)机器周期。在单片机中,为了便于管理,常把一条指令的执行过程划分为若干个阶段,每一阶段完成一项工作。例如,取指令、存储器读、存储器写等,这每一项工作称为一个基本操作。完成一个基本操作所需要的时间称为机器周期。一般情况下,一个机器周期由若干个状态组成。80C51单片机的一个机器周期由6个状态组成。分别表示为S1~S6,而一个状态包含两个节拍,那么一个机器周期就有12个节拍,记为:S1P1,S1P2,...,S6P1,S6P2,一个机器周期共包含12个振荡脉冲,即机器周期就是振荡脉冲的12分频,显然,如果使用6MHz的时钟频率,一个机器周期就是2μs,而如使用12MHz的时钟频率,一个机器周期就是1μs。

(3)指令周期。指令周期是执行一条指令所需要的时间,一般由若干个机器周期组成。指令周期以机器周期的数目来表示,指令不同,所需要的机器周期数也不同。80C51单片机的指令周期根据指令的不同,可包含有1个、2个、3个或4个机器周期。

对于一些简单的单字节指令,在取指令周期中,指令取出到指令寄存器后,立即译码执行,不再需要其他的机器周期。对于一些比较复杂的指令,例如转移指令、乘除指令,则需要两个或者两个以上的机器周期。

二、单片机的复位电路

复位是单片机的初始化操作,其主要功能是把PC初始化为0000H,使单片机从0000H单元开始执行程序。除了进入系统的正常初始化之外,当由于程序运行出错或操作错误使系统处于死锁状态时,也需按复位键以重新启动。

1. 复位对专用寄存器的影响

除PC之外,复位操作还对其他一些专用寄存器有影响,如表2-7所列。

表 2-7 专用寄存器的复位状态

寄存器	复位状态	寄存器	复位状态
PC	0000H	TCON	00H
ACC	00H	TL0	00H
PSW	00H	TH0	00H
SP	07H	TL1	00H
DPTR	0000H	TH1	00H
P0~P3	FFH	SCON	00H
IP	xxx00000B	SBUF	不定
IE	0xx00000B	PCON	0xxx0000B
TMOD	00H		

2. 复位电路

80C51单片机的RST引脚是复位信号的输入端,复位信号是高电平有效,其有效时

间应持续 24 个振荡脉冲周期(即两个机器周期)以上,若使用频率为 6MHz 的晶振,则复位信号持续时间应超过 $4\mu\text{s}$ 才能完成复位操作。

复位操作有上电自动复位和按键手动复位两种方式,如图 2-12 所示。

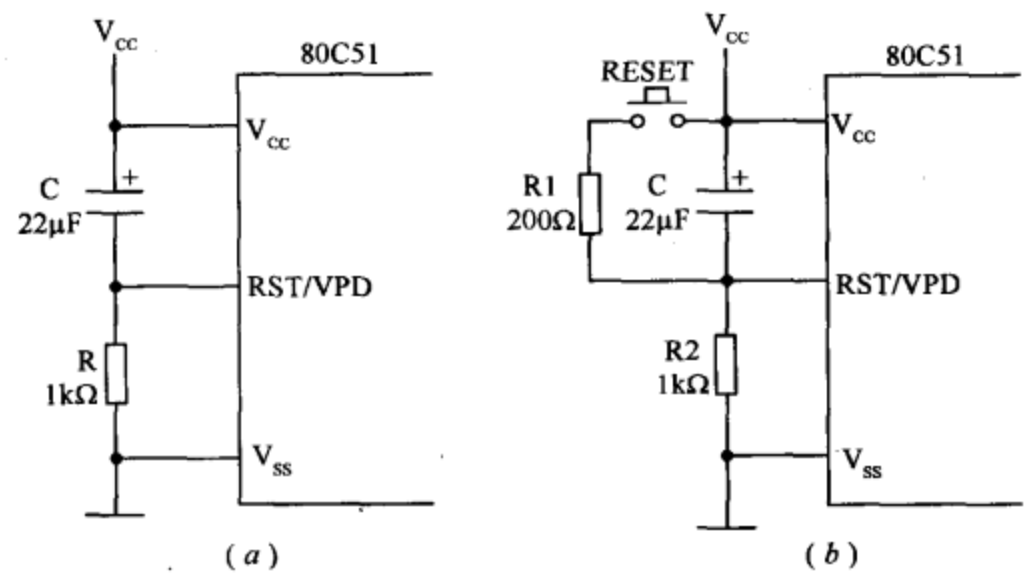


图 2-12 复位电路
(a)上电复位; (b)手动复位。

(1)上电复位。上电自动复位是通过外部复位电路的电容充电来实现的,如图 2-13(a)所示。对于 CMOS 型单片机,由于在 RST 端内部有一个下拉电阻,可将外部电阻去掉。

上电复位的过程是在加电时,复位电路通过电容加给 RST 端一个短暂的高电平信号,此高电平信号随着 V_{cc} 对电容的充电过程而逐渐回落,即 RST 端的高电平持续时间取决于电容的充电时间。为了保证系统能够可靠地复位,RST 端的高电平信号必须维持足够长的时间。

(2)手动复位。手动复位需要人为在复位输入端 RST 上加入高电平。一般采用的办法是在 RST 端和正电源 V_{cc} 之间接一个按钮,如图 2-13(b)所示。当按下按钮时,则 V_{cc} 的 +5 V 电平就会直接加到 RST 端。即使按下按钮的动作较快,也会使按钮保持接通达数十毫秒,所以,保证能满足复位的时间要求。

三、单片机的低功耗方式

80C51 有两种低功耗方式,即待机(或称空闲)方式和掉电保护(或称停机)方式,目的是为了尽可能地降低功耗。

待机方式和掉电方式都是由特殊功能寄存器 PCON(电源控制寄存器)中的有关位来控制。其各位作用如下:

PCON	D7	D6	D5	D4	D3	D2	D1	D0
(87H)	SMOD	—	—	—	GF1	GF0	PD	IDL

- (1)SMOD:波特率倍增位。在串行通信时使用。
- (2)GF1 和 GF0:通用标志位。由软件置、复位。
- (3)PD:掉电方式位。若 $PD=1$,进入掉电工作方式。
- (4)IDL:待机方式位。若 $IDL=1$,进入待机工作方式。

如果 PD 和 IDL 同时为 1, 则进入掉电工作方式。复位时 PCON 中所有定义位均为 0。要想使单片机进入待机或掉电工作方式, 只要执行一条使 IDL 或 PD 位为“1”的指令即可。下面介绍两种低功耗方式操作过程。

1. 待机方式

若写一个字节到 PCON, 使 $IDL=1$, 单片机即进入待机方式。例如, 执行过“ORL PCON, #1”指令后, 单片机即进入待机方式, 此指令即为待机方式的启动指令。

在待机方式下, 振荡器仍然工作, 并向中断逻辑、串行口和定时/计数器电路提供时钟, 但向 CPU 提供时钟的电路被阻断, 因此 CPU 不能工作, 与 CPU 有关的如 SP、PC、PWS、ACC 以及全部通用寄存器也都被“冻结”在原状态。

终止待机方式的方法有以下两种:

一是通过硬件复位。由于在待机方式下时钟振荡器一直在运行, RST 引脚上的有效信号只需保持两个时钟周期就能使 IDL 置 0, 单片机即退出待机状态, 从它停止运行的地方恢复程序的执行, 即从空闲方式的启动指令之后继续执行。

二是通过中断方法。若在待机期间, 任何一个允许的中断被触发, IDL 都会被硬件置 0, 从而结束待机方式。其实在中断服务程序中只需安排一条 RETI 指令, 就可以使单片机恢复正常工作后返回断点继续执行程序。

2. 掉电保护方式

当 CPU 执行一条置 PCON 寄存器的 PD 为 1 的指令后, 系统进入掉电工作方式。在这种工作方式下, 内部振荡器停止工作。由于没有振荡时钟, 因此, 所有的功能部件都停止工作。但内部 RAM 区和特殊功能寄存器区的内容被保留, 而端口的输出状态值都被存在对应的 SFR 中。

退出掉电方式的唯一方法是硬件复位。复位后所有特殊功能寄存器的内容初始化, 但不改变内部 RAM 中的数据。

在掉电工作方式下, V_{CC} 可以下降到 2V; 但在进入掉电方式以前, V_{CC} 不能降低。而在准备退出掉电方式之前, V_{CC} 必须恢复正常的工作电压, 并保持一段时间 (10ms), 使振荡器重新启动并稳定后方可退出掉电方式。

方法技巧 单片机在运行过程中, 如发生掉电故障, 将会使系统数据丢失, 为此, MCS-51 单片机设置有掉电保护措施, 进行掉电保护处理。其具体做法如下:

(1) 数据转存。所谓数据转存是指当电源出现故障时, 应立即将系统的有用数据转存到内部 RAM 中。数据转存是通过中断服务程序完成的, 即通常所说的“掉电中断”。

因为单片机电源端 (V_{CC}) 都接有滤波电容, 掉电后电容储存的电能量能维持有效电压为几个毫秒之久, 足以完成一次掉电中断操作。为此应在单片机系统中设置一个电压检测电路, 一旦检测到电源电压下降, 立即通过 INT0 或 INT1 产生外部中断请求, 中断响应后执行中断服务程序, 把有用数据送内部 RAM 中保护起来。

(2) 接通备用电源。为了保存转存后的有用数据, 掉电后应给内部 RAM 供电。为此, 系统应预先装有备用电源, 并在掉电后立即接通备用电源。备用电源由单片机的 RST/VPD 引脚接入。为了在掉电能及时接通备用电源, 系统中还需具有备用电源与 V_{CC} 电源的自动切换电路, 如图 2-13 所示。

切换电路由两个二极管组成, 当电源电压 V_{CC} 高于 RST/VPD 引脚的备用电源电压

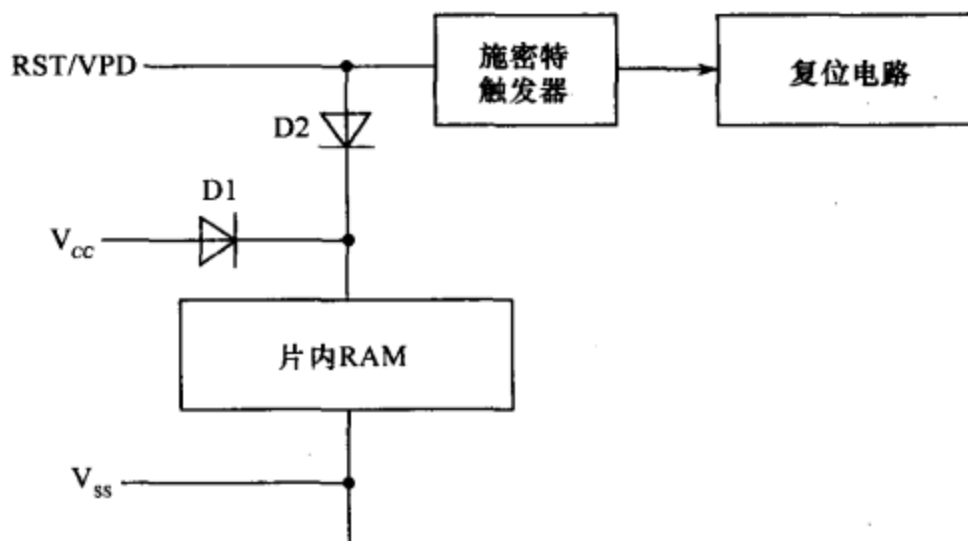


图 2-13 备用电源与 V_{CC} 电源的自动切换电路

时, D1 导通, D2 截止, 内部 RAM 由 V_{CC} 电源供电。而当 V_{CC} 电源降至备用电源电压以下时, 则 D1 截止, D2 导通, 内部 RAM 由备用电源供电。这时单片机就进入掉电保护方式。

由于备用电源容量有限, 为减少消耗, 掉电后时钟电路和 CPU 皆停止工作, 只有内部 RAM 单元和专用寄存器继续工作, 以保持其内容。

第五节 AT89C51 和 AT89C2051/1051 简介

AT89 系列是 ATMEL 公司的 8 位 Flash 单片机, 这个系列的单片机的最大特点是片内含有 Flash 存储器, 因此, 有着广泛的用途, 特别是在便携式、省电和特殊信息保存的仪器中显得更为有用。AT89 系列的产品较多, 下面简要介绍其中的几种。

一、AT89C51 简介

AT89C51 是一种低功耗、高性能的 8 位单片机, 片内带有一个 4 KB 的 Flash 可编程、可擦除只读存储器, 它采用了 CMOS 工艺和 ATMEL 公司的高密度非易失性存储器技术, 而且其输出引脚和指令系统都与 MCS-51 兼容。片内的 Flash 存储器允许在系统内改编程或常规的非易失性可编程存储器来编程。因此 AT89C51 是一种功能强、灵活性高, 且价格合理的单片机, 可方便地应用在各种控制领域。

另外, AT89C51 是用静态逻辑来设计的, 其工作频率可下降到零, 并提供两种可用软件来选择的省电方式——空闲方式和掉电方式。在空闲方式中, CPU 停止工作, 而 RAM、定时/计数器、串行口和中断系统都继续工作。在掉电方式中, 片内振荡器停止工作, 由于时钟被“冻结”, 使一切功能都暂停, 故只保存片内 RAM 中的内容, 直到下一次硬件复位为止。

AT89C51 的管脚排列与功能与 80C51 完全一致, 这里不再重复。

二、AT89C2051/1051 简介

ATMEL 公司的 AT89C2051、AST89C1051 是在 AT89C51 的基础上将一些功能精简掉后形成的精简版。AT89C2051 取掉了 P0 口和 P2 口, 内部的程序 FLASH 存储器容量也小到 2KB, 封装形式也由 51 的 40 脚改为 20 脚, 相应的价格也低一些, 特别适

合在一些智能玩具、手持仪器等程序不大的电路环境下应用;AT89C1051 在 2051 的基础上,再次精简掉了串口功能等,程序存储器再次减小到 1KB,当然价格也更低。对 2051 和 1051 来说,虽然减掉了一些资源,但他们片内都集成了一个精密比较器,别小看这小小的比较器,他为我们测量一些模拟信号提供了极大的方便,在外加几个电阻和电容的情况下,就可以测量电压、温度等日常需要的量。这对很多日用电器的设计是很宝贵的资源。

下面重点以 AT89C2051 为例进行说明。

1. AT89C2051 单片机的主要特性

AT89C2051 单片机的主要特性如下:

- (1)与 MCS-51 系列产品兼容;
- (2)片内含有 2KB 的 Flash 程序存储器;
- (3)内含 128B 的 RAM;
- (4)具有 15 线可编程的 I/O 接口;
- (5)含有两个 16 位的定时器(T0 和 T1);
- (6)具有 5 个中断向量、两级中断优先权的中断结构;
- (7)具有可编程 UART 串行通信口;
- (8)两级程序锁定位;
- (9)低功耗节电方式包括空闲模式和掉电模式;
- (10)完全静态操作模式为 0~24MHz;
- (11)电源电压为 2.7V~6V;
- (12)可直接驱动 LED;
- (13)片内含有一个模拟比较器。

2. AT89C2051 的引脚功能

图 2-14 为 AT89C2051 单片机引脚排列图。

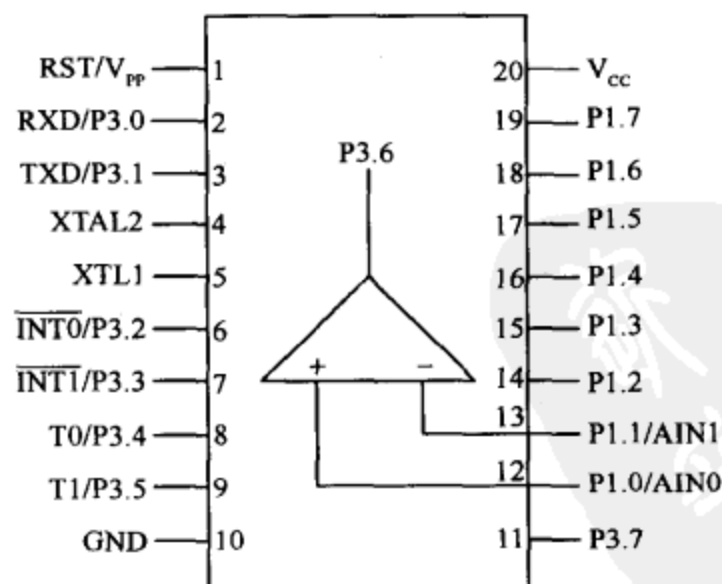


图 2-14 AT89C2051 引脚排列图

(1)P1 口。P1 口是 8 位双向 I/O 接口。P1.2~P1.7 内含上拉电阻,P1.0 和 P1.1 需要外接上拉电阻。P1.0 和 P1.1 又是片内模拟比较器的同相输入(AIN0)和反相输入

(AIN1)端。

P1 口用做输出时,输出缓冲器可驱动 20mA 的灌电流负载,可直接驱动 LED 显示器。P1 口用做输入时,应先对端口写 1。外部的输入信号将 P1.2~P1.7 拉为低电平时,通过片内上拉电阻向外输出电流。

(2)P3 口。P3 口是 7 位双向 I/O 接口,内部具有上拉电阻。

P3 口提供给用户可使用的 I/O 接口是 P3.0~P3.5 和 P3.7。P3.6 在片内与模拟比较器的输出端相连,不可当做通用 I/O 接口那样访问。

P3 口的输出缓冲器可提供 20mA 的灌电流负载。P3 口用做输入时,应先对端口写 1,当外部输入信号将其拉为低电平时,通过片内上拉电阻向外输出电流。

P3 口还具有第二功能,如表 2-8 所列。

表 2-8 P3 口的第二功能

引脚	第二功能	引脚	第二功能
P3.0	RXD(串行输入口)	P3.3	INT1(外部中断 1)
P3.1	TXD(串行输出口)	P3.4	T0(定时器 0 外部输入)
P3.2	INT0(外部中断 0)	P3.5	T1(定时器 1 外部输入)

(3)RST。RST 为复位输入端。RST 保持两个机器周期的高电平(振荡器正常工作),器件便可复位,所有 I/O 接口引脚都输出高电平。

(4)时钟信号脚。XTAL1 和 XTAL2 分别是构成片内振荡器反相放大器的输入和输出。采用片内振荡器时,须外接晶体振荡器或陶瓷振荡器。AT89C2051 单片机也可采用外部振荡器,使用片外振荡器时,引脚 XTAL2 悬空不用,引脚 XTAL1 作为片外振荡器信号的输入端。

(5)电源和接地脚。 V_{CC} 为电源电压供电端,GND 为电源接地脚。

3. AT89C2051 的存储器

AT89C2051 单片机有两个存储器空间:片内 2KB 的 Flash 程序存储器空间(000H~7FFH)和片内数据存储器空间,片内数据存储器包含 128B 的特殊功能寄存器区(80H~FFH)和 128B 的片内 RAM 区(00H~7FH)。

AT89C2051 单片机与 MCS-51 结构完全兼容,可使用 MCS-51 指令系统编程。但是由于存储空间的局限性,某些指令的应用受到一定限制。

(1)无条件转移指令有 LCALL、LJMP、ACALL、AJMP、SJMP、JMP@A+DPTR。这些指令用于 AT89C2051 单片机时,目的转移地址必须在 000H~7FFH 区间。

(2)条件转移指令有 CJNE、DJNZ、JB、JBC、JNB、JC、JNC、JZ、JNZ。这些指令用于 AT89C2051 单片机时,目的转移地址必须在 000H~7FFH 区间。

(3)片内 RAM 为 128B,栈区的设置必须限于 00H~7FH 区间。

(4)AT89C2051 单片机没有片外数据存储器,在 AT89C2051 单片机的应用程序中不应包含 MOVX 类指令。

AT89C2051 单片机的 5 个中断向量与 MCS-51 系列单片机完全一样。

AT89C2051 单片机共有 19 个特殊功能寄存器,如表 2-9 所列。

表 2-9 AT89C2051 特殊功能寄存器(SFR)

序号	地址	符号	复位值	说明
1	81H	SP	07H	堆栈指针
2	82H	DPL	00H	数据指针 DPTR 低 8 位
3	83H	DPH	00H	数据指针 DPTR 高 8 位
4	87H	PCON	0xxx 0000B	电源控制器
5	88H	TCON	00H	定时器控制器
6	89H	TMOD	00H	定时器模式寄存器
7	8AH	TL0	00H	定时器 0 低 8 位
8	8BH	TL1	00H	定时器 1 低 8 位
9	8CH	TL1	00H	定时器 0 高 8 位
10	8DH	TH1	00H	定时器 1 高 8 位
11	90H	P1	FFH	P1 口锁存器
12	98H	SCON	00H	串行口控制器
13	99H	SBUF	不定	串行口数据缓冲器
14	0A8H	IE	0x00 0000B	中断允许寄存器
15	0B0H	P3	FFH	P3 口锁存器
16	0B8H	IP	xx00 0000B	中断优先寄存器
17	0D0H	PSW	00H	程序状态字
18	0E0H	ACC	00H	累加器
19	0F0H	B	00H	B 寄存器

地址空间 80H~FFH 并没有完全被 SFR 占用,未被占用的地址单元,用户不可访问。

4. 低功耗运行模式

AT89C2051 单片机具有两种低功耗节电运行方式:空闲模式和掉电模式。

(1)空闲模式。在空闲模式下,CPU 进入休眠状态,片内 RAM 和 SFR 中的内容保持不变。

利用软件(使 IDL=1)可进入空闲模式。结束空闲模式有两种方法:任何一个可允许中断和硬件复位。

如果没有外部上拉电阻,则将 P1.0 和 P1.1 设置为 0;如果有外部上拉电阻,则将 P1.0 和 P1.1 设置为 1。

用硬件复位结束空闲模式时,在复位逻辑发挥控制作用前的两个机器周期内,器件从断点处(IDL=1 指令后面一条指令)开始执行指令。在该状态下,片内硬件电路可以禁止访问片内 RAM,但不禁止对端口的访问。为了避免用硬件复位退出空闲模式时对端口不应有的写入,在进入空闲模式的指令(使 IDL=1 的指令)后面不应是写入端口引脚的指令。

(2)掉电模式。在掉电模式下,振荡器停止工作,片内 RAM 和特殊功能寄存器的内容保持不变。

进入掉电模式的指令(使 PD=1 的指令)是执行程序的最后一条指令。结束掉电模式只能是硬件复位。复位将重新定义 SFR 的内容,但不改变片内 RAM 中的内容。当 V_{CC} 恢复到正常运行值,且当振荡器重新启动并达到稳定后,复位方可有效。

在没有外部上拉电阻时,将 P1.0 和 P1.1 设置为 0;在有外部上拉电阻时,将 P1.0 和 P1.1 设置为 1。



第三章 单片机实验软件和 硬件环境的建立

学习单片机离不开实验,边学边练,这样才能尽快掌握,而单片机实验则需要一定的软件和硬件环境来支持。目前开发单片机的应用软件比较杂乱,使初学者无所适从,这里我们推荐 Keil C51 和万利(Manley)公司的 MedWin 软件,这两个软件均为集成系统。所谓集成,是指将原程序编写、汇编/编译/连接、调试等开发单片机所要用的程序集合到一个软件中。对于硬件,则需要实验板、编程器、仿真器等。目前,市场上这类硬件设备品种很多,价格也相差很大。本章主要选择几种价格便宜,同时又十分适合初学者学习的硬件设备进行介绍。

第一节 单片机仿真软件Keil C51 的使用

Keil C51 是德国开发的一个 51 单片机开发软件平台,开始只是一个支持 C 语言和汇编语言的编译器软件。后来随着开发人员的不断努力以及版本的不断升级,使它已经成为一个重要的单片机开发平台。Keil C51 的界面和常用的微软 VC++ 的界面相似,界面友好,易学易用,在调试程序,软件仿真方面也有很强大的功能。因此,很多开发 51 应用的工程师或普通的单片机爱好者,都对它十分喜欢。

Keil C51 软件带一个集成开发环境 μ Vision2, μ Vision2 是一种文件管理编译环境,集成了文本编辑处理、编译链接、项目管理、窗口、工具引用和软件仿真调试等多种功能,是相当强大的 C51 开发工具。在本书中,不对 Keil C51 和 μ Vision2 两个术语做严格的区分,一般来说,都指的是 μ Vision2 集成开发环境。

在 μ Vision2 的仿真功能中,有两种仿真模式:软件模拟方式和硬件仿真。在软件模拟方式下,不需要任何 51 单片机硬件即可完成用户程序仿真调试,极大地提高了用户程序开发效率。在硬件仿真方式下,用户可以将程序装到自己的 51 单片机系统板上,利用 51 的串口与 PC 进行通信来实现用户程序的实时在线仿真。

学习程序设计语言和某种程序软件,最好的方法是直接操作实践。下面通过简单的实例,引导大家学习 Keil C51 软件的基本使用方法和基本的调试技巧。

一、Keil C51 软件的启动

要使用 Keil C51 软件,必须先安装。Keil C51 是一个商业的软件,它同时也提供了学习者使用的 Eval 版本。该版本同正式版本一样,但有一定的限制,最终生成的代码不能超过 2KB,但用于学习已经足够了。读者可到有关搜索网站输入关键词 Keil C51,找到一个合适的网站,可立即下载该软件的最新版本。

安装好后,在桌面上双击 Keil μ Vision2 图标,即可启动该软件。启动屏幕如图 3-1 所示。



图 3-1 μ Vision2 启动屏幕

几秒钟后,出现编辑界面,如图 3-2 所示。

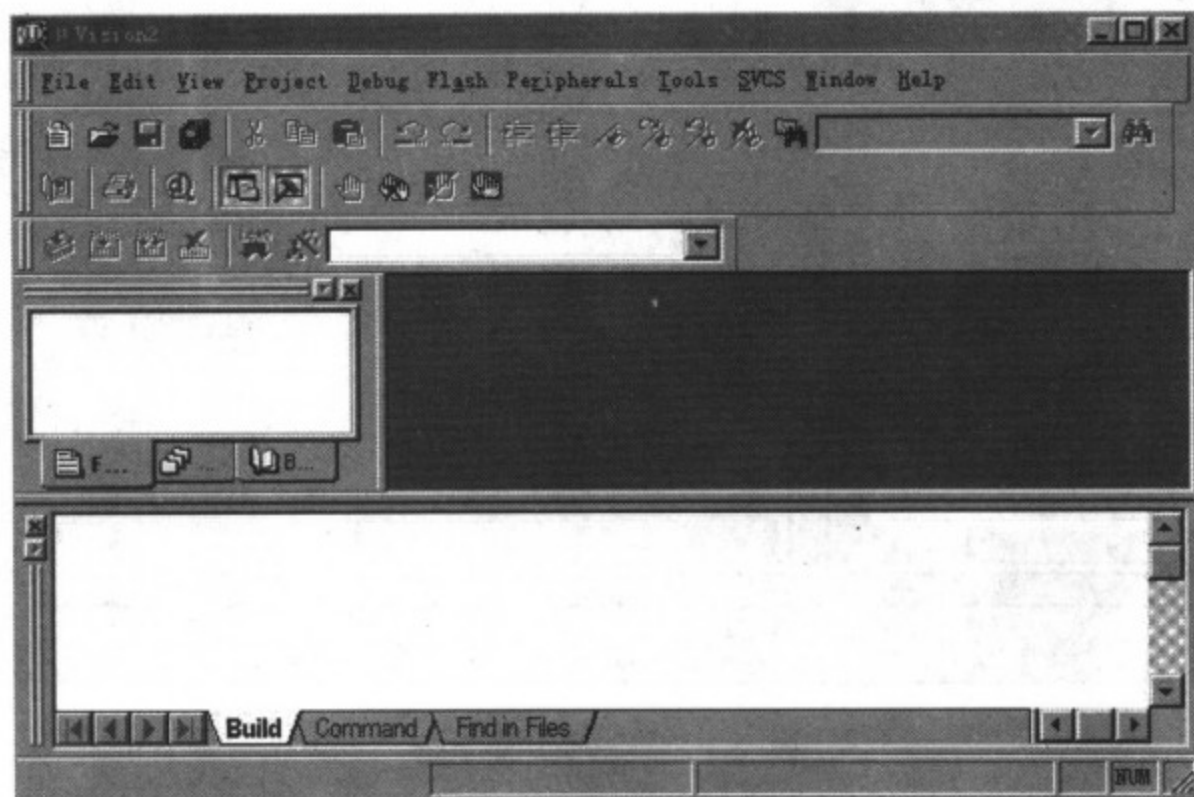


图 3-2 编辑界面

二、建立一个新工程

(1) 点击 Project 菜单,选择下拉式菜单中的 New Project,弹出文件对话框,选择要保存的路径,在“文件名”中输入第一个汇编程序项目名称,这里用 one,如图 3-3 所示。

保存后的文件扩展名为 .uv2,这是 Keil μ Vision2 项目文件扩展名,以后可以直接点击此文件以打开先前做的项目。

(2) 点击“保存”后,这时会弹出一个对话框,要求选择单片机的型号,如图 3-4 所示。



图 3-3 文件对话框窗口

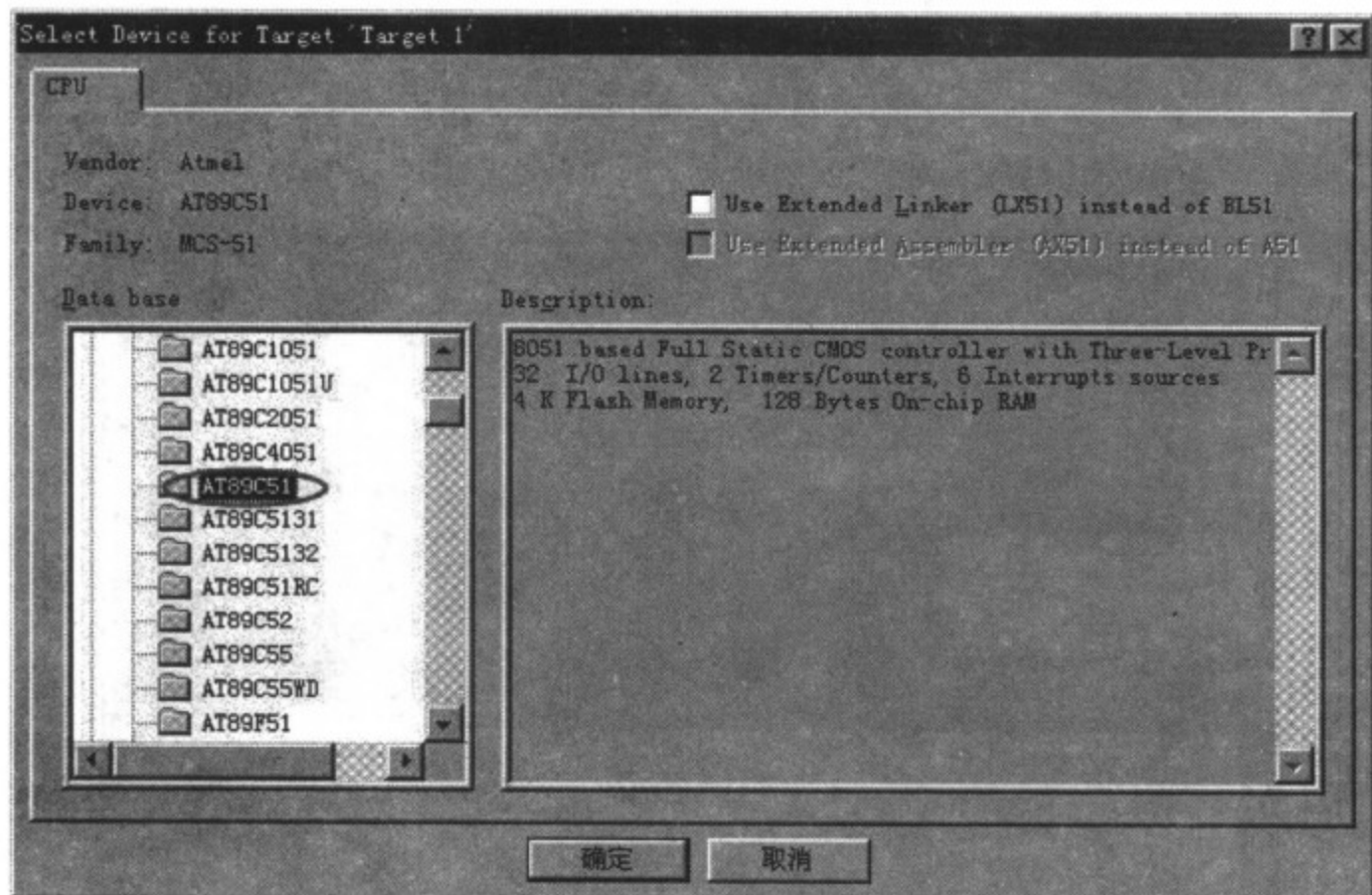


图 3-4 选择单片机窗口

读者可以根据所使用的单片机来选择,Keil 几乎支持所有的 51 核的单片机,这里还是以大家用的比较多的 Atmel 的 89C51 来说明,选择 AT89C51 之后,右边栏是对这个单片机的基本的说明,然后点击确定。

(3)完成以上步骤后,出现如图 3-5 所示的窗口。

(4)下面开始编写我们的第一个程序。在图 3-5 中,点击 File 菜单,再在下拉菜单中点击 New 选项,新建文件后的窗口如图 3-6 所示。

(5)此时光标在编辑窗口里闪烁,这时可以键入用户的应用程序了,但笔者建议首先保存该空白的文件,点击菜单上的 File,在下拉菜单中选中 Save As 选项,屏幕显示如图 3-7 所示,在“文件名”栏右侧的编辑框中,键入欲使用的文件名,同时,必须键入正确的扩展名。

注意 如果用 C 语言编写程序,则扩展名为 .c;如果用汇编语言编写程序,则扩展名

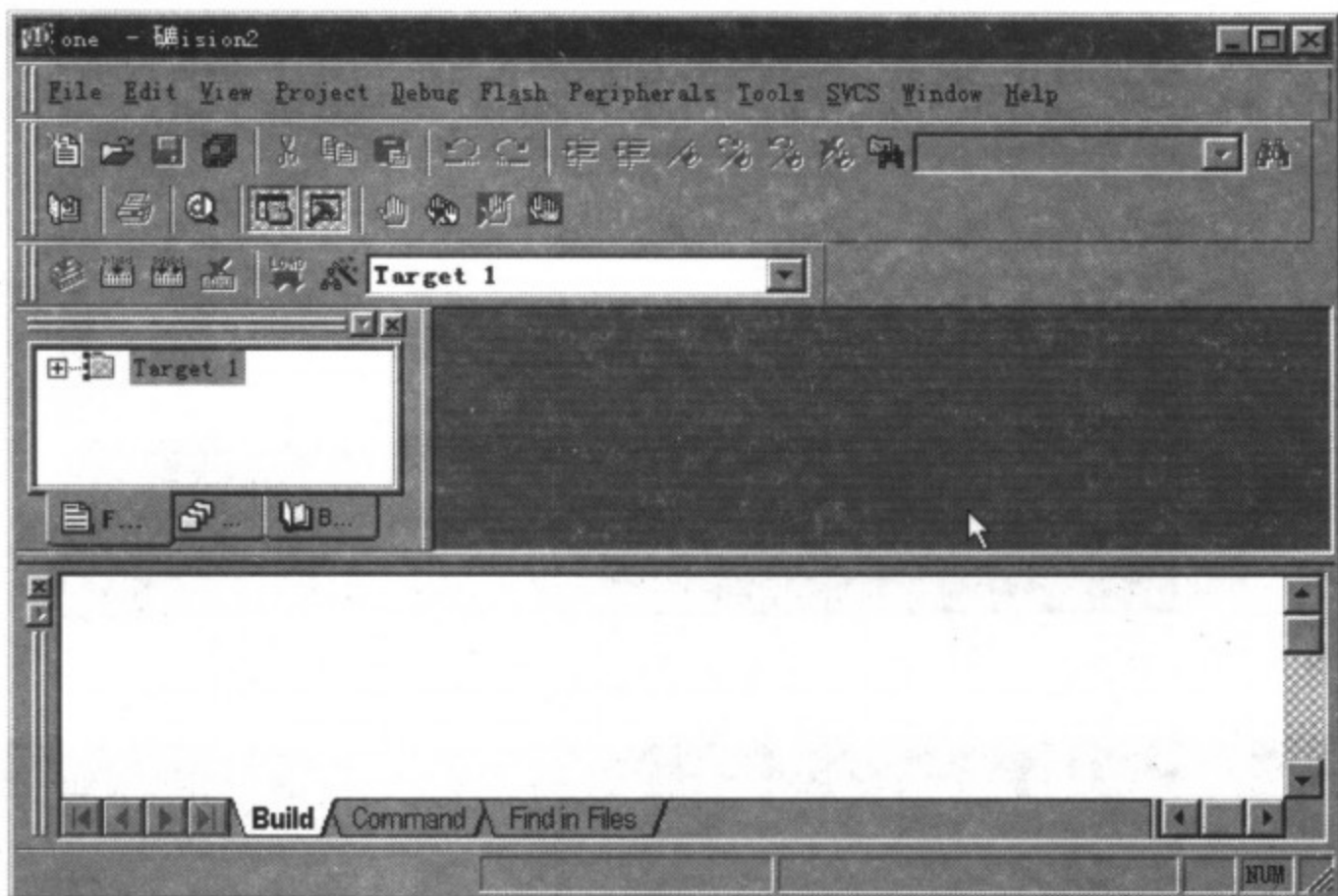


图 3-5 完成窗口

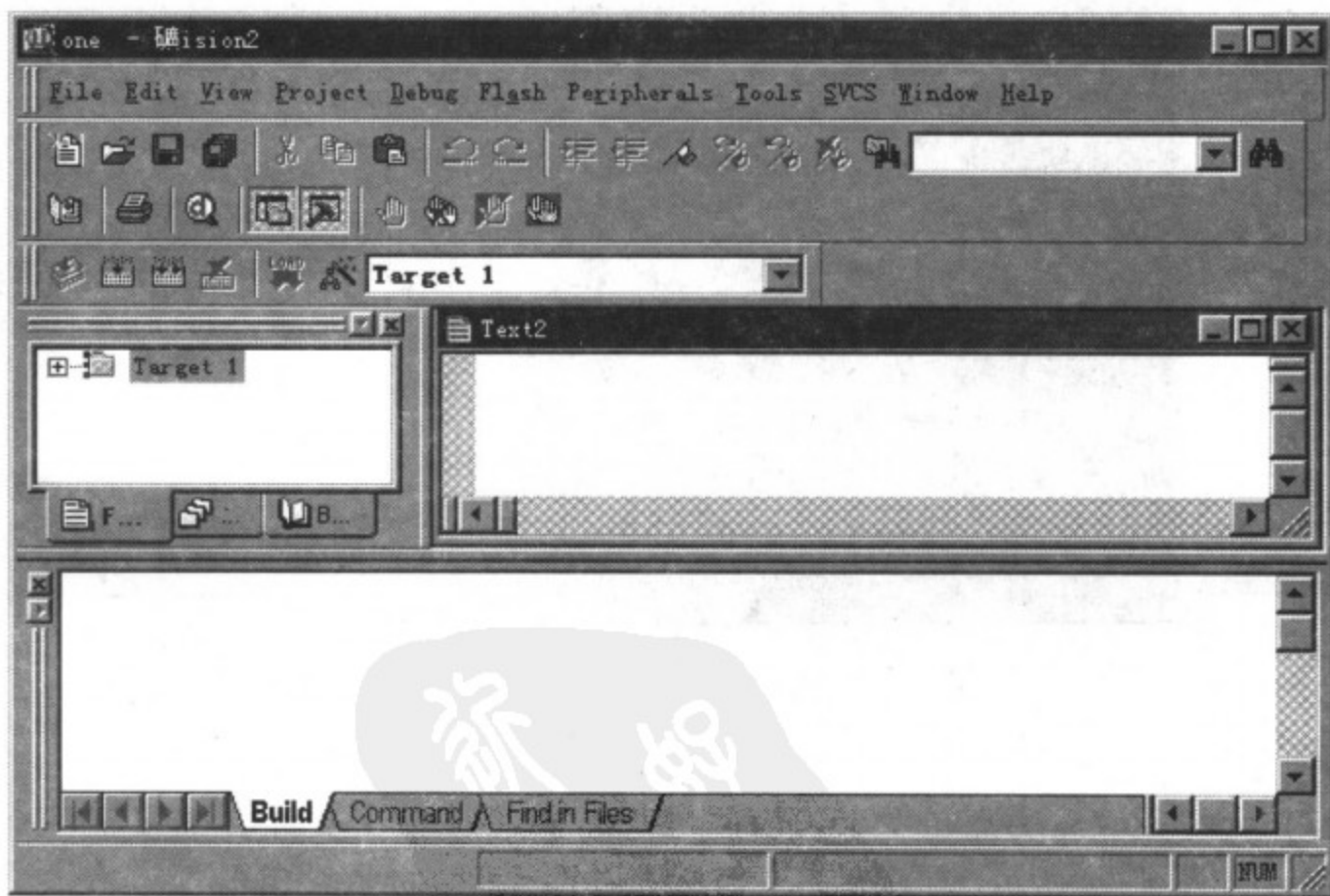


图 3-6 新建文件后的窗口

必须为 .asm。这里选用文件名为 one.asm, 然后, 点击“保存”按钮。

(6) 回到编辑界面后, 点击 Target 1 前面的“+”号, 然后在 Source Group 1 上单击鼠标右键, 弹出如图 3-8 所示的菜单。

(7) 然后点击 Add File to Group ‘Source Group 1’, 出现如图 3-9 所示的增加源文件



图 3-7 保存文件

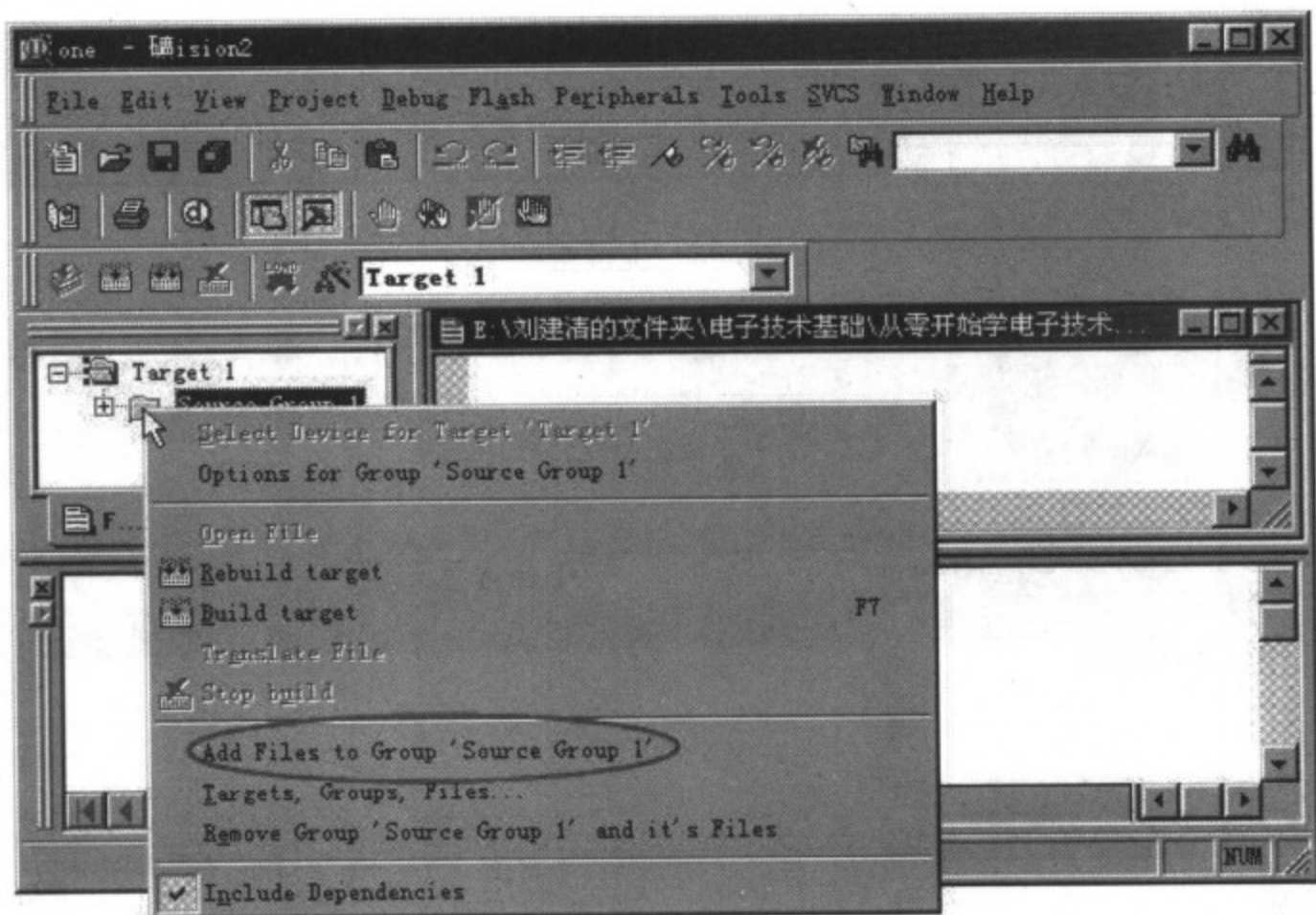


图 3-8 弹出的菜单

对话框。

选中 one.asm, 然后点击 Add, 此时, 在 Source Group 1 文件夹中多了一个子项 one.asm, 如图 3-10 所示。子项的多少与所增加的源程序的多少相同。

重点提示 将文件 one.asm 加入 Source Group 1 后, 增加源文件对话框并不消失, 等待继续加入其他文件, 但初学时常会误认为操作没有成功而再次点击 Add 按钮, 这时会出现如图 3-11 所示的提示窗口, 提示所选文件已在列表中, 此时应点击“确定”, 返回前一对话框, 然后点击 Close, 即可返回主界面。返回后, 点击 Source Group 1 前的加号, 会发现 one.asm 文件已在其中。

(8) 现在, 在编辑窗口中输入如下的汇编语言源程序(在附光盘的 example\ch_3\one

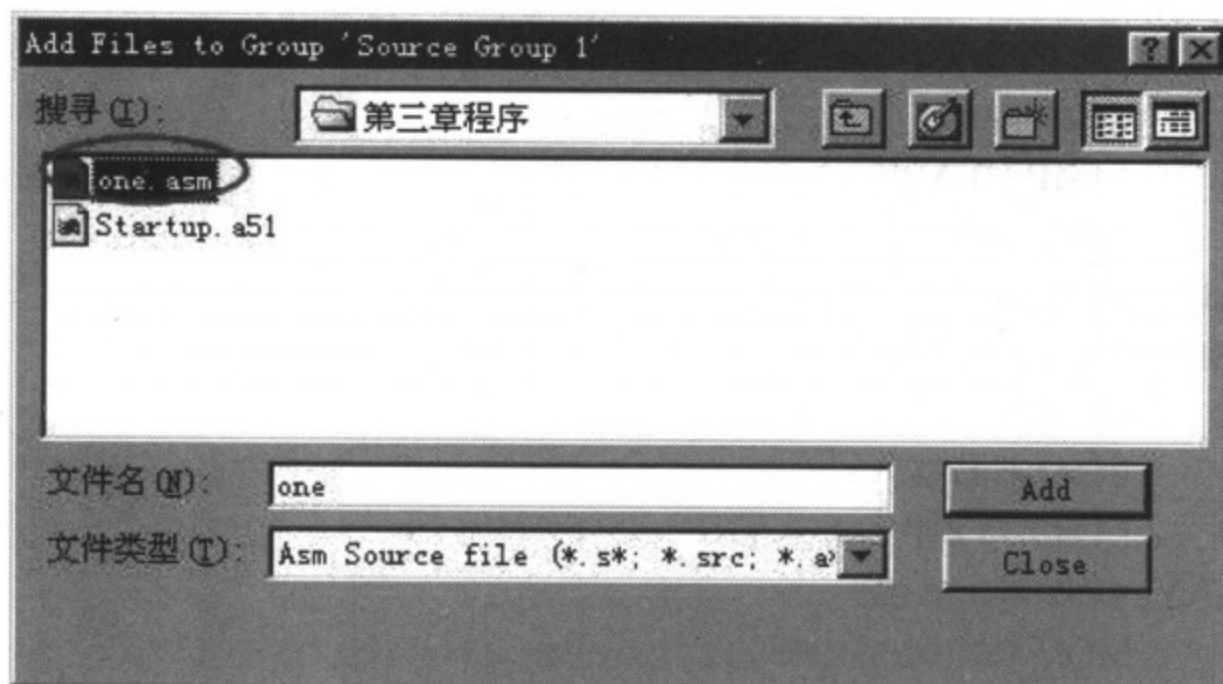


图 3-9 增加源文件对话框

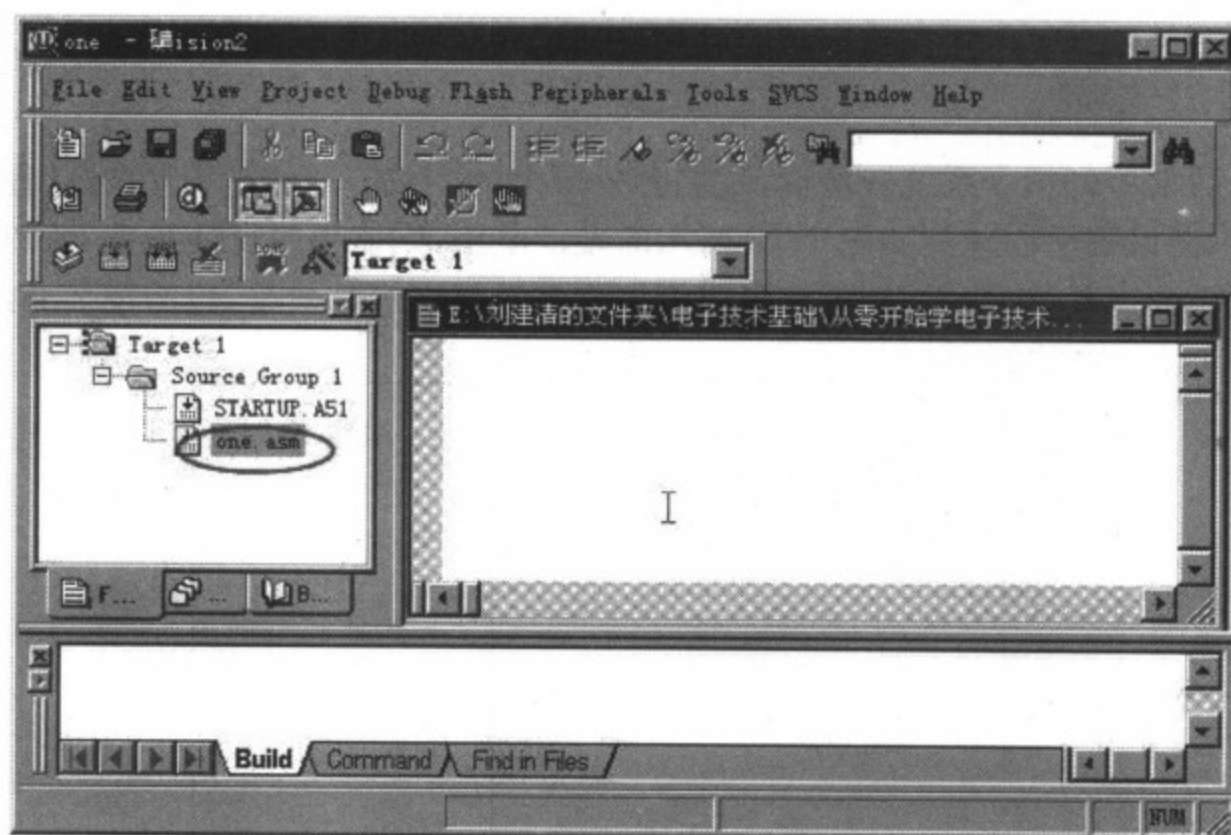


图 3-10 增加文件后的屏幕

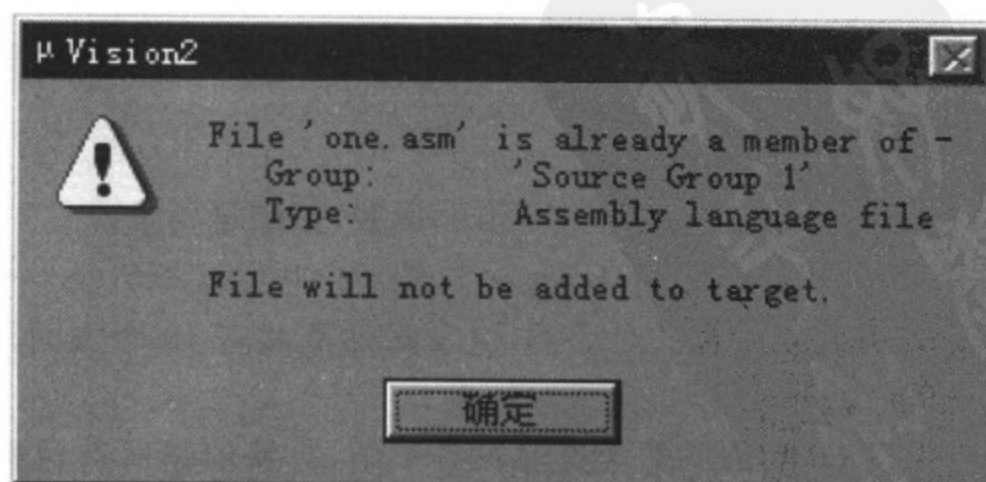


图 3-11 提示窗口

文件夹中):

```
ORG 0000H
LJMP START
ORG 30H
START: MOV A, #0FEH
LOOP:  MOV P1, A
      RR A
      LCALL DELAY
      LJMP LOOP
DELAY: MOV R7, #255
D1:    MOV R6, #255
D2:    DJNZ R6, D2
      DJNZ R7, D1
      RET
      END
```

在以后学习到指令系统一章后,我们会分析出,以上是一个“8 路流水灯”程序。也就是说,如果把该程序固化到如图 3-12 所示的硬件电路,则该程序可使接在 P1 口的发光二极管依次循环点亮,当然,用软件也可以模拟出 8 路流水灯的效果,以后还要介绍。

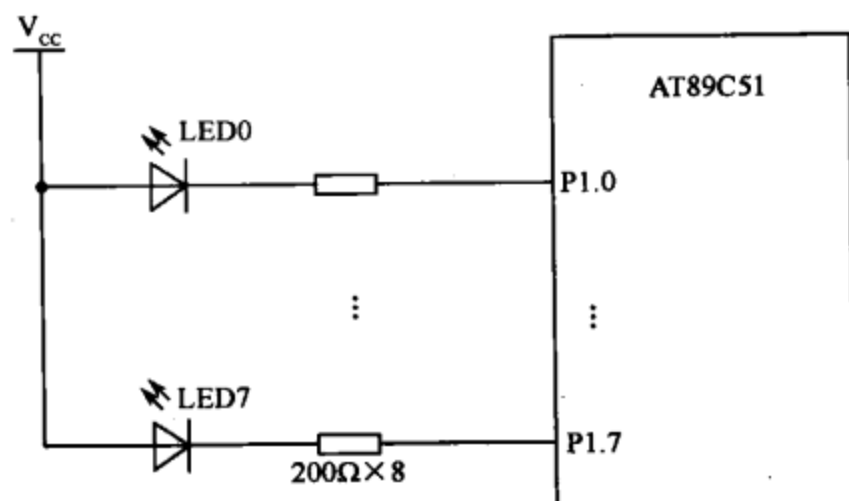


图 3-12 硬件电路

程序输入完毕后,屏幕显示如图 3-13 所示。

三、工程的设置

工程建立好以后,还要对工程进行进一步的设置,以满足要求。

(1)首先点击左边 Project 窗口的 Target 1,然后使用菜单“Project→Option for Target‘Target1’”,即出现对工程设置的对话框,这个对话框共有 10 个页面,如图 3-14 所示,大部分设置项取默认值就行了。

(2)在图 3-14 中,Xtal 后面的数值是晶振频率值,默认值是所选目标 CPU 的最高可用频率值,该值与最终产生的目标代码无关,仅用于软件模拟调试时显示程序执行时间。正确设置该数值可使显示时间与实际所用时间一致,一般将其设置成与硬件所用的晶振

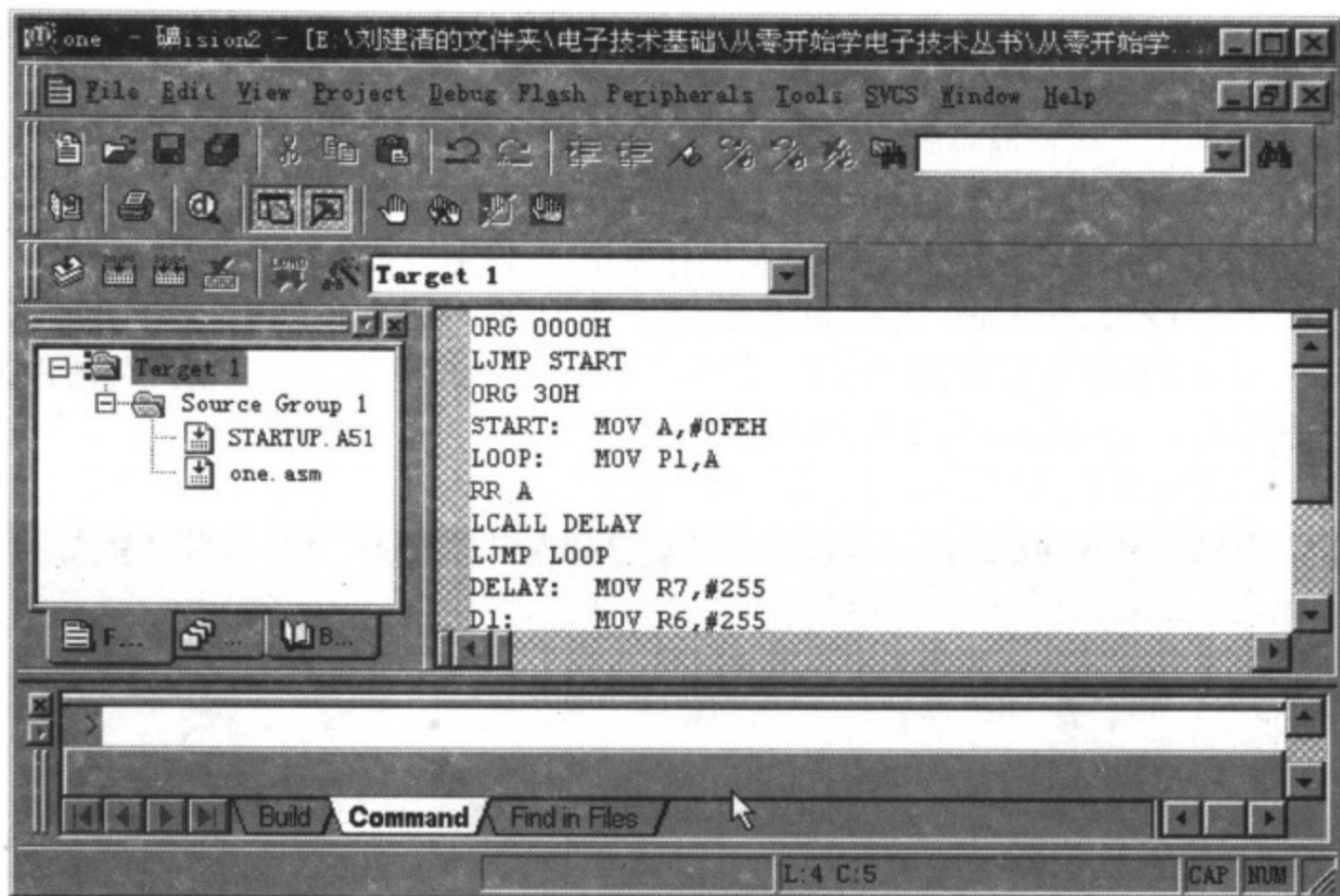


图 3-13 输入程序后的屏幕

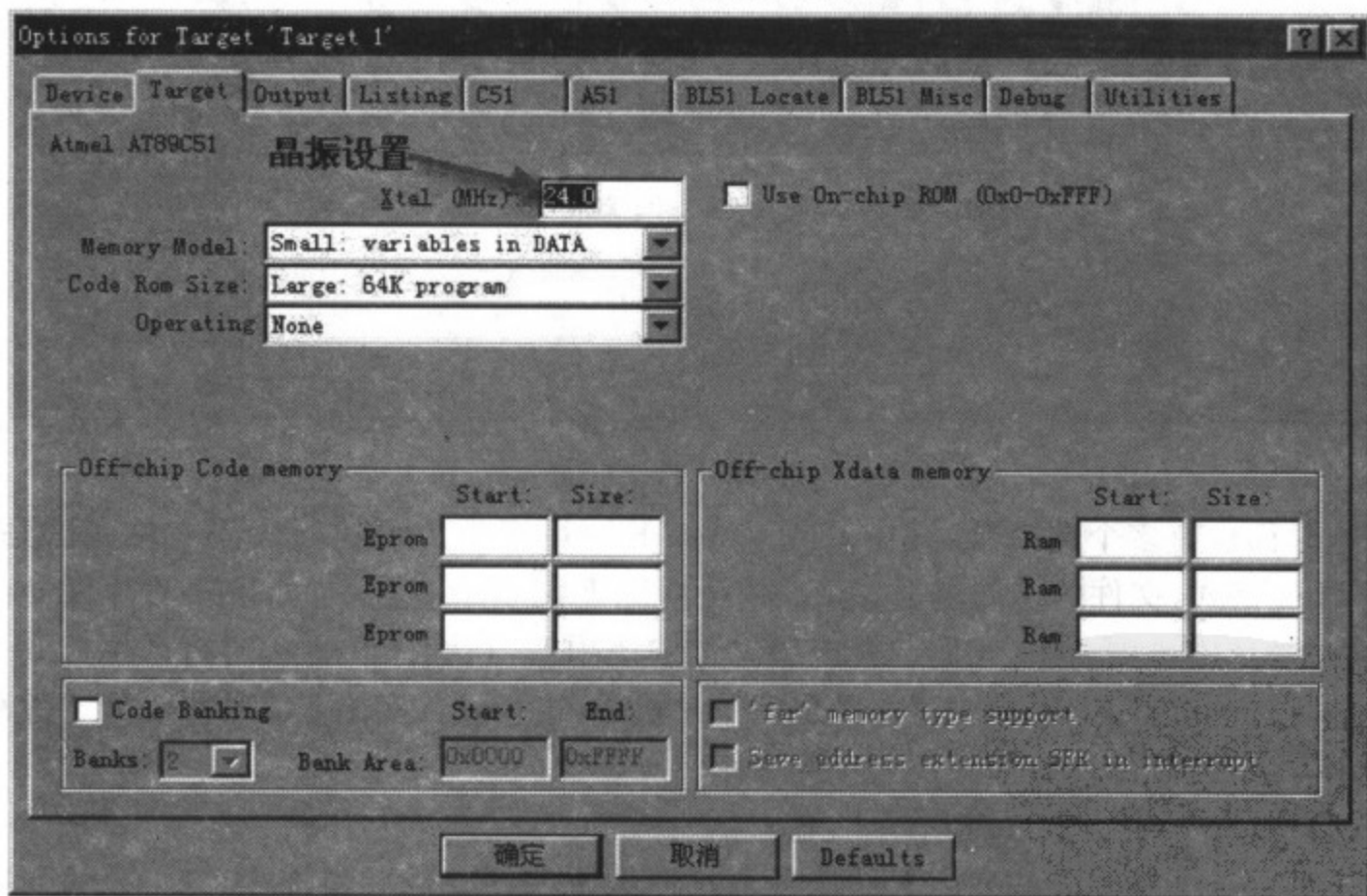


图 3-14 工程设置对话框

频率相同,如果没必要了解程序执行的时间,也可以不设。

Memory Model 用于设置 RAM 使用情况,有 3 个选择项,其中 Small 表明所有变量都在单片机的内部 RAM 中;Compact 表明可以使用一页(256B)外部扩展 RAM;Large 表明可以使用全部外部的扩展 RAM。

Code Rom Size 用于设置 ROM 空间的使用,同样也有 3 个选择项:Small 表明只用低

于 2Kb 的程序空间;Compact 表明单个函数的代码量不能超过 2Kb,整个程序可以使用 64Kb 程序空间;Larget 表明可用全部 64Kb 空间。

这些选择项必须根据所用硬件来决定,对于本例,按默认值设置。

Operating 项是操作系统选择,Keil 提供了两种操作系统:Rtx tiny 和 Rtx full,关于操作系统本书不作介绍,通常不使用任何操作系统,即使用该项的默认值 None。

Off-chip Code memory 用以确定系统扩展 ROM 的地址范围。Off-chip Xdata memory 组用于确定系统扩展 RAM 的地址范围。这些选择项必须根据所用硬件来决定,一般均不需要重新选择,按默认值设置即可。

(3)OutPut 页如图 3-15 所示。

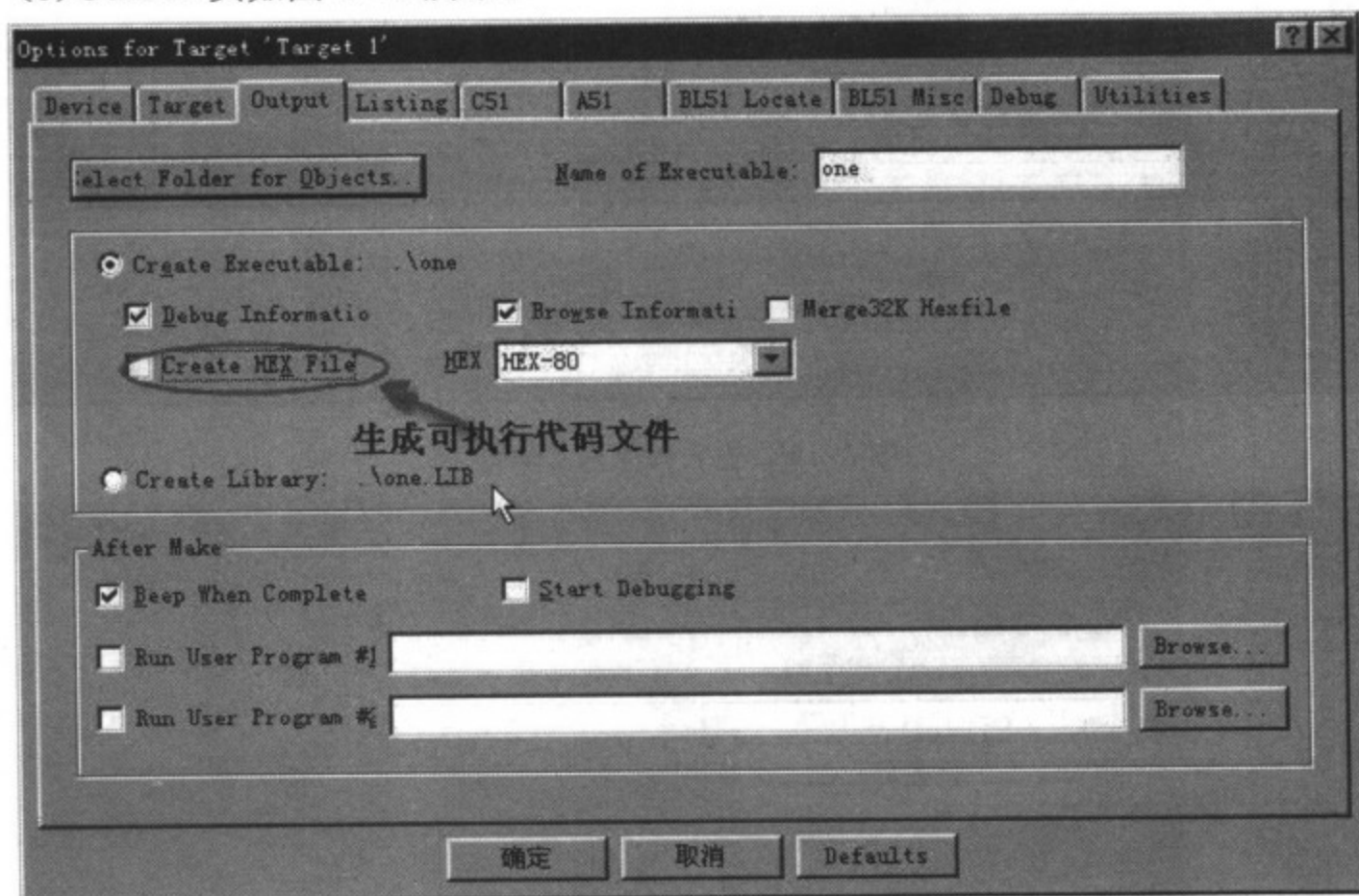


图 3-15 OutPut 页

这里面也有多个选择项,其中 Creat Hex File 用于生成可执行代码文件,其格式为 Intel Hex 格式,文件的扩展名为 .hex,默认情况下该项未被选中,如果要写片做硬件实验,就必须选中该项,这里我们选择该项。选中该项后,在编译和链接时将产生*.hex 代码文件,该文件可用编程器去读取并烧到单片机中,再用硬件实验板看结果。至于编程器的使用方法,在硬件设备部分还要具体介绍。

(4)Listing 页如图 3-16 所示。

该页用于调整生成的列表文件选项。在汇编或编译完成后将产生(*.lst)的列表文件,在连接完成后也将产生(*.m51)的列表文件,该页用于对列表文件的内容和形式进行细致的调节,一般采用默认设置。

(5)Debug 页如图 3-17 所示。

该页用于设置调试器,Keil 提供了两种工作模式,即 Use Simulator(软件模拟仿真)和 Use(硬件仿真),Use Simulator 是将 Keil 设置成软件模拟仿真模式。在此模式下不需要实际的目标硬件就可以模拟 MCS-51 单片机的很多功能,在准备硬件之前就可以测试

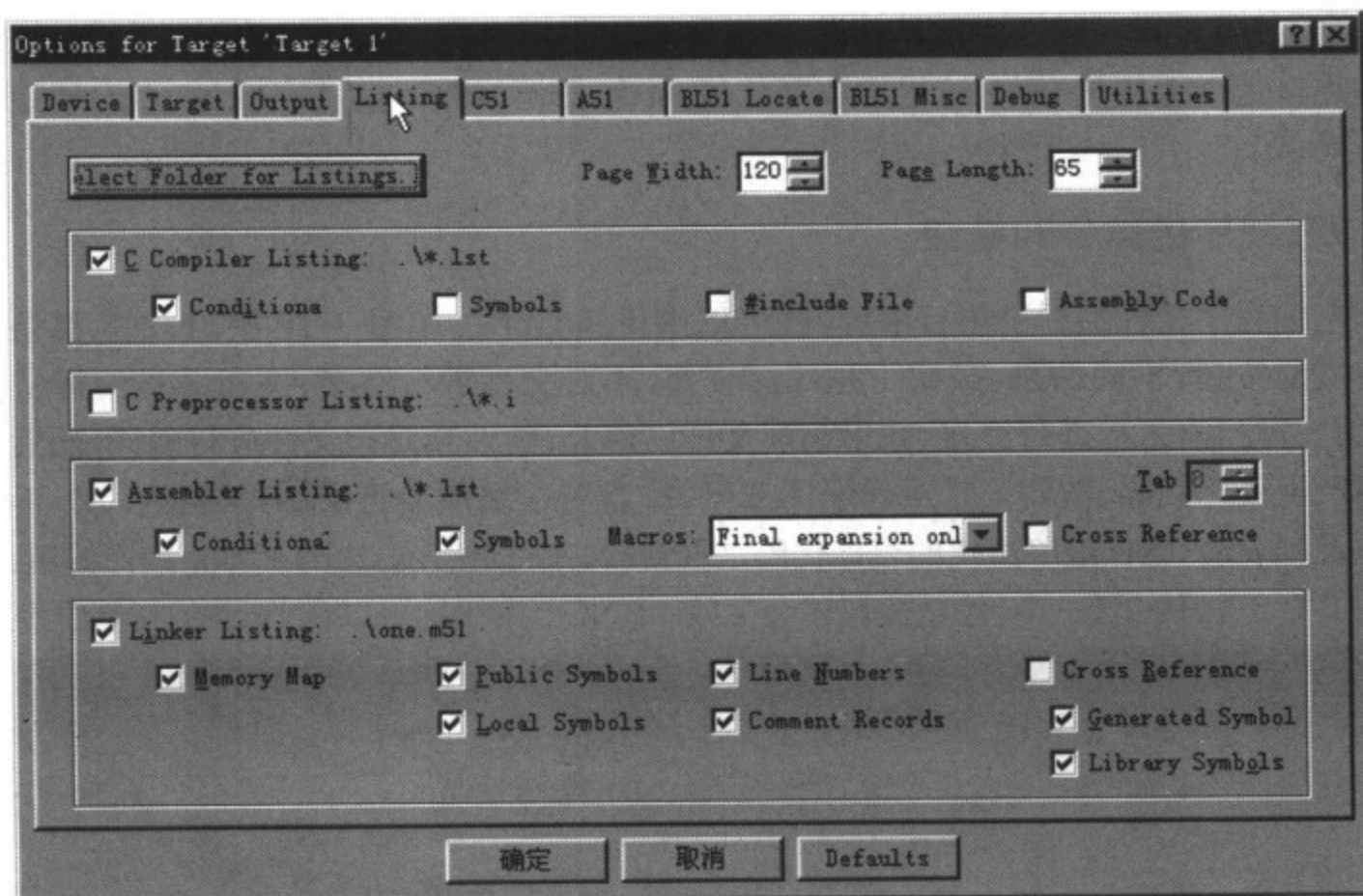


图 3-16 Listing 页

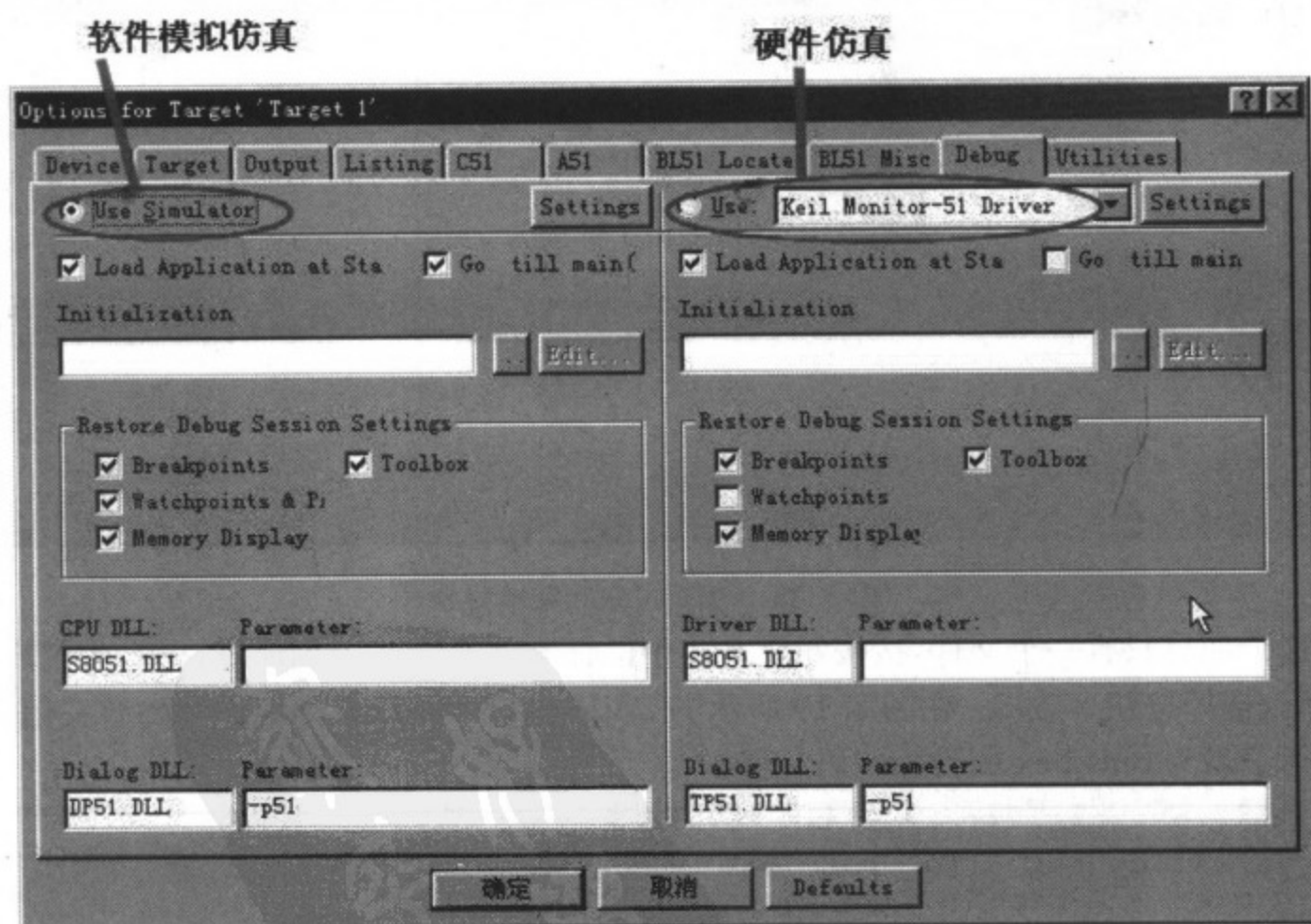


图 3-17 Debug 页

用户的应用程序,这是很有用的;Use 选项是高级 GDI 驱动,运用此功能高级用户可以把 Keil C51 嵌入到自己的系统中,从而实现在目标硬件上调试程序。如果没有相应的硬件调试器,应选择 Use Simulator,这里,我们选择 Use Simulator 项,即将 Keil 配置为软件模拟仿真。

(6)工程设置对话框中的其他各页面与 C51 编译选项、A51 的汇编选项、BL51 连接器的连接选项等用法有关,这里均取默认值,不作任何修改。设置完成后,点击确定按钮进行确认。

四、程序的编译和链接

汇编程序文件加到了项目中后,就可以编译运行了。在图 3-18 中,1、2、3 都是和编译有关的按钮,不同的是:按钮 1 是编译按钮,不对文件进行链接;按钮 2 是编译链接按钮,用于对当前工程进行链接,如果当前文件已修改,软件会对该文件进行编译,然后再链接以产生目标代码;按钮 3 是重新编译按钮,每点击一次均会再次编译链接一次,不管程序是否有改动,确保最终产生的目标代码是最新的。在按钮 3 右边的是停止编译按钮,只有点击了前 3 个中的任一个,停止按钮才会生效。这几个按钮均可以在 Project 菜单中找到。

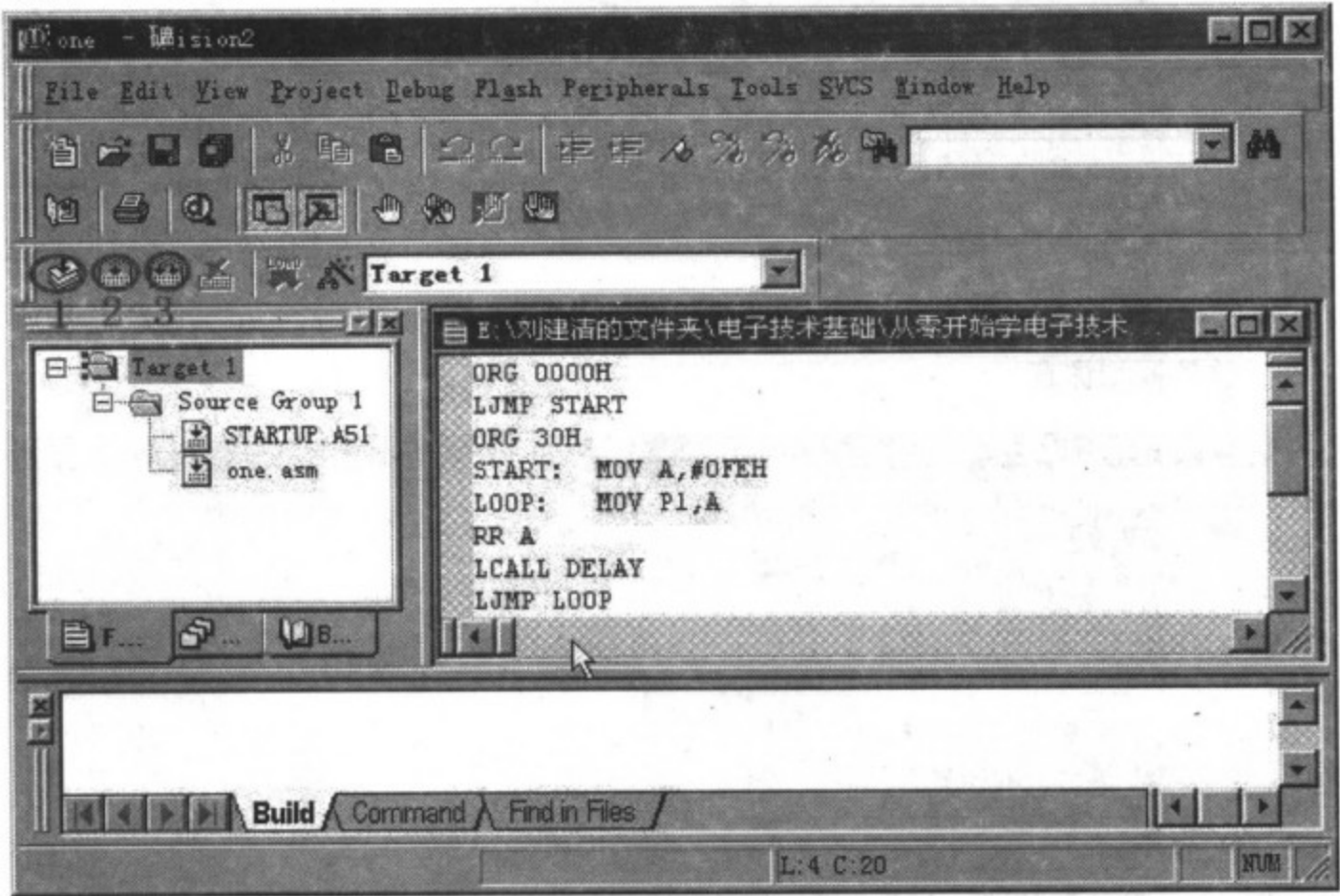


图 3-18 编辑按钮和菜单命令

这个项目只有一个文件,按按钮 1、2、3 中的任意一个都可以编译。这里,为了产生目标代码,选择按钮 2 或 3。在图 3-19 所示的 Build 窗口中可以看到编译后的有关信息,提示获得了名为 one.hex 的目标代码文件。

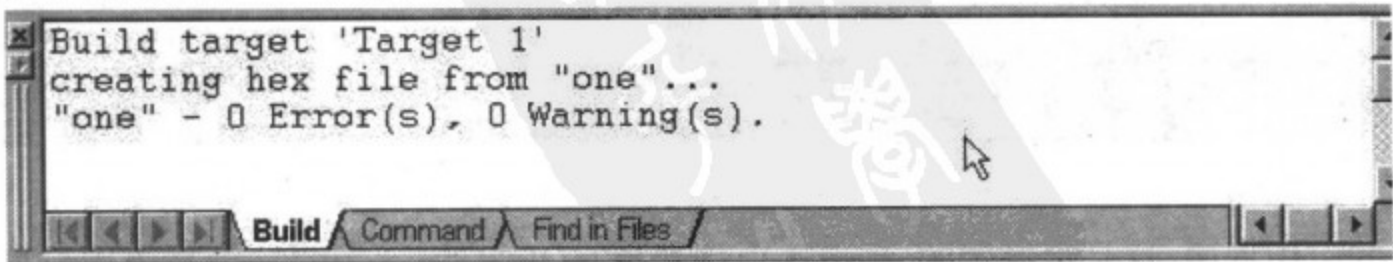


图 3-19 编译后的有关信息

如果源程序有语法错误,会有错误报告出现,用户应根据提示信息,更正程序中出现的错误,重新编译,直至正确为止。

五、程序调试

前面学习了如何建立工程、编译,并获得目标代码,但是做到这一步仅仅代表你的源程序没有语法错误,至于源程序中存在着的其他错误,必须通过调试才能发现并解决,事实上,除了极简单的程序以外,绝大部分的程序都要通过反复调试才能得到正确的结果,因此,调试是软件开发中重要的一个环节,下面将介绍调试的方法和步骤。

对工程成功地进行汇编、连接以后,使用菜单 Debug→Start/Slop Debug Session(或按 Ctrl+F5 键)即可进入调试状态,如图 3-20 所示。

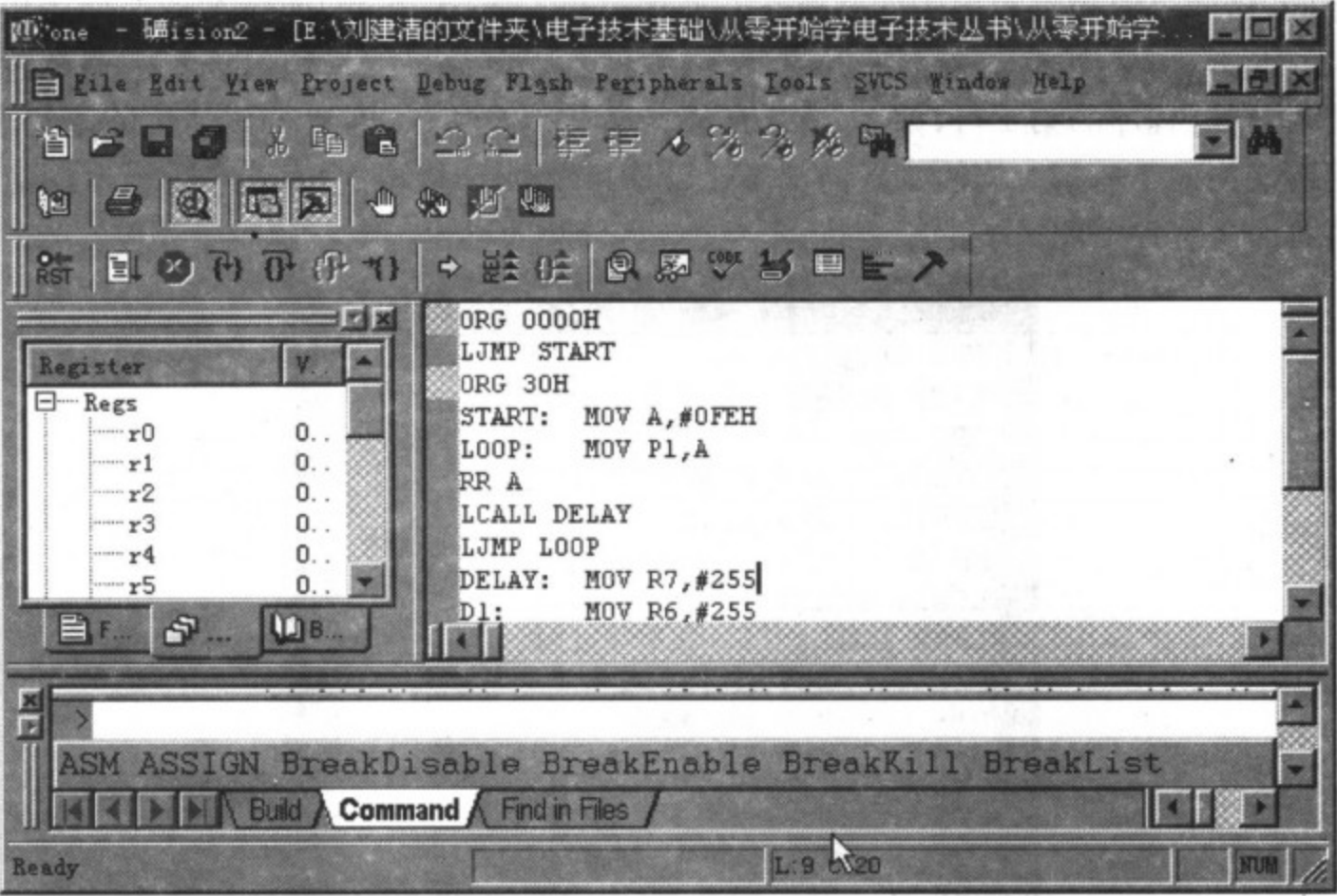


图 3-20 程序调试状态

重点提示 Keil 内建了一个仿真 CPU 用来模拟执行程序,该仿真 CPU 功能强大,可以在没有硬件和仿真机的情况下进行程序的调试。不过,模拟毕竟只是模拟,与真实的硬件执行程序肯定还是有区别的,其中最明显的就是时序。软件模拟是不可能和真实的硬件具有相同的时序的,具体的表现就是程序执行的速度和每人使用的计算机有关,计算机性能越好,运行速度越快。


进入调试状态后,界面与编辑状态相比有明显的变化,Debug 菜单项中原来不能用的命令现在已经可以使用了,工具栏会多出一个用于运行和调试的工具条,如图 3-21 所示。




图 3-21 运行和调试工具条

Debug 菜单上的大部分命令可以在此找到对应的快捷按钮,从左到右依次是复位、全速运行、暂停、单步、过程单步、执行完当前子程序、运行到当前行、下一状态、打开跟踪、观

察跟踪、反汇编窗口、观察窗口、代码作用范围分析、I#串行窗口、内存窗口、性能分析、工具按钮等命令。

全速执行是指一行程序执行完以后紧接着执行下一行程序，中间不停止，这样程序执行的速度很快，并可以看到该段程序执行的总体效果，即最终结果正确还是错误，但如果程序有错，则难以确认错误出现在哪些程序行。使用菜单 Debug→Go 或  按钮或使用快捷键 F5 可以单步执行程序。

单步执行是每次执行一行程序，执行完该行程序以后即停止，等待命令执行下一行程序，此时可以观察该行程序执行完以后得到的结果，是否与写该行程序时所想要得到的结果相同，借此可以找到程序中问题所在。使用菜单 Debug→Step 或  按钮或使用快捷键 F11 可以单步执行程序。按下 F11 键，可以看到源程序窗口的左边出现了一个黄色调试箭头，指向源程序的第 1 行，如图 3-22 所示。每按一次 F11，即执行该箭头所指程序行，然后箭头指向下一行。当箭头指向 LCALL DELAY 行时，再次按下 F11，会发现箭头指向了延时子程序 DELAY 的第 2 行；不断按 F11 键，即可逐步执行延时子程序。

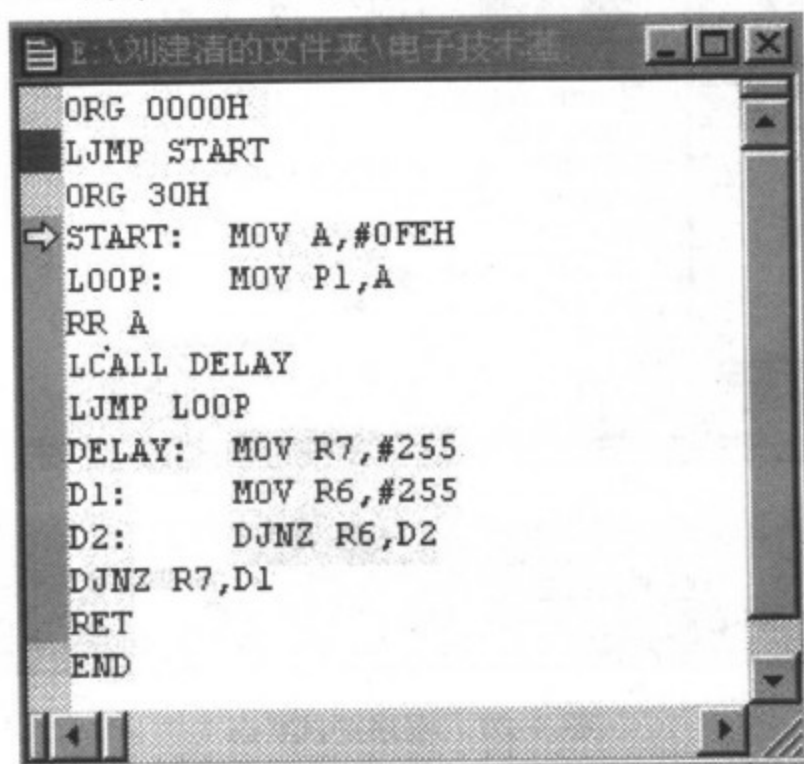





图 3-22 单步执行

过程单步是指将汇编语言中的子程序或 C 语言中的函数作为一个语句来全速执行。使用菜单 Debug→Step Over 或  按钮或功能键 F10 可以用过程单步形式执行命令。

重点提示 通过单步执行程序，可以找出一些问题的所在，但是仅依靠单步执行来查错有时是困难的，或虽能查出错误但效率很低，为此必须辅之以其他的方法，如在本例中的延时程序是通过将 D2:DJNZ R6,D2 这一行程序执行 6 万多次来达到延时的目的。如果用按 F11 6 万多次的方法来执行完该程序行，显然不合适。为此，可以采取以下一些方法：

(1) 在进入该子程序后，使用菜单 Debug→Step Out of Current Function(单步执行到该函数外)或  按钮或 Ctrl+F11 键，即全速执行完调试光标所在的子程序，并指向主程序中的下一行程序，在这里，执行完后指向 LJMP LOOP 行。

(2) 用鼠标在子程序的最后一行点一下，把光标定位于该行，然后用菜单 Debug→



Run to Cursor Line(执行到光标所在行)或  按钮或 Ctrl+F10 键,即可全速执行完黄色箭头与光标之间的程序行。

(3)在开始调试时,按过程单步快捷键 F10,程序将单步执行,执行到子程序 LCALL DELAY 行时,按下 F10 键,调试光标不进入子程序的内部,而是全速执行完该子程序,然后直接指向下一行 LJMP LOOP。

灵活应用以上几种方法,可以大大提高查错的效率。

程序调试时,一些程序行必须满足一定的条件才能被执行到(如程序中某变量达到一定的值、按键被按下、串口接收到数据、有中断产生等),这些条件往往是异步发生或难以预先设定的,这类问题使用单步执行的方法是很难调试的,这时就要使用到程序调试中的另一种非常重要的方法——断点设置。断点设置的方法有多种,常用的是在某一程序行设置断点,设置好断点后,可以全速运行程序,一旦执行到该程序行即停下,可在此时观察有关变量值,以确定问题所在。在程序行设置/移除断点的方法是将光标定位于需要设置断点的程序行,使用菜单 Debug→Insert/Remove Breakpoint 设置或移除断点(也可以用鼠标在该行双击实现同样的功能);Debug→Enable/Disable Breakpoint 是开启或暂停光标所在行的断点功能;Debug→Disable All Breakpoint 暂停所有断点;Debug→Kill All Breakpoint 清除所有的断点设置。这些功能也可以用工具条上的快捷按钮进行设置。

下面简要说明调试程序时如何进行查错。假设将程序行 D2:DJNZ R6,D2 改为 D2:DJNZ R6,D1,然后重新编译,由于这样的改动并没有语法错误,所以,编译时不会报错。

进入调试状态后,按 F10 过程单步的形式执行程序,当执行到 LCALL DELAY 行时,程序不能继续往下执行,同时发现调试工具条上的 Halt 按钮变成了红色,说明程序在此不断地执行着,而我们预期这一行程序将执行完后停止,这个结果与预期不同,可以看出调用的子程序出了差错。为查明出错原因,按  按钮,使程序停止执行,然后按  按钮,使程序复位,再次按下 F10 单步执行,但在执行到 LCALL DELAY 行时,改按 F11 键跟踪到子程序内部。单步执行程序,可以发现在执行到 D2:DJNZ R6,D1 行时,程序不断地从这一行转移到上一行。同时观察左侧的寄存器的值。会发现 R6 的值始终在 FFH 和 FEH 之间变化,不会减小,而我们的预期是 R6 的值不断减小,减到 0 后往下执行,因此这个结果与预期不符。通过这样的观察,不难发现问题是因为标号写错而产生的,发现问题即可以修改。再进行编译链接、调试,发现程序能够正确地执行了,这说明修改是正确的。

讲到这里,我们再回过头来看一下图 3-12 所示的硬件电路,在这里我们的 LED 要 AT89C51 的 P1 引脚为低电平才会点亮,所以我们要向 P1 口的各引脚写数据 0,对应连接的 LED 才会被点亮,P1 口的 8 个引脚刚好对应 P1 口特殊寄存器的 8 个二进位,如向 P1 口定数据 0FEH,转成二进制就是 11111110,最低位为 0,这里 P1.0 引脚输出低电平,LED0 被点亮。如此类推,不难算出想要做的效果了。如果用编程器把该程序烧写到实验板上的单片机上,效果就出来,显示的速度可以根据需要调整延时时间。如果没有实验板,用 Keil 软件也可以仿真程序运行的结果。

按外部设备菜单 Peripherals→I/O Ports→Port1 就打开 Port1 I/O 观察窗口了,如图 3-23 所示。

图 3-23 中,凡框内打“√”者为高电平,未打“√”者为低电平。按 F5 全速运行,会发

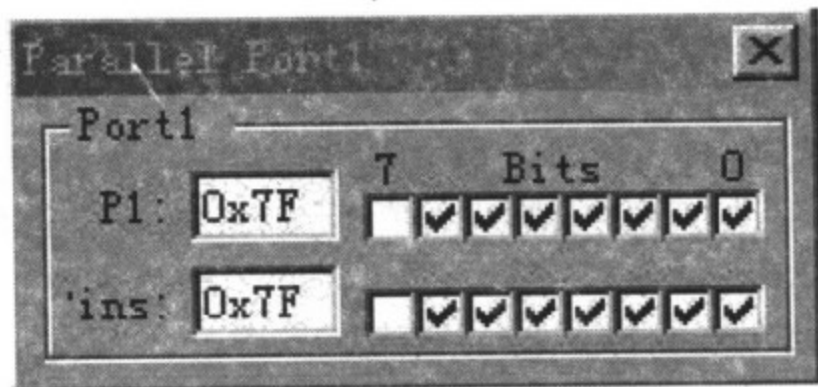



图 3-23 Port1 调试窗口

现窗口中的“√”在 Port1 调试窗口中不停地闪动,不能看到具体的效果,这时可以采用过程单步执行的方法进行调试,不停地按动 F10 键,可以看到 Port1 调试窗口中未打“√”的小框(表示低电平)不停地右移。也就是说,Port1 调试窗口模拟了 P1 口的电平状态。

六、常用窗口介绍

为帮助用户分析和调试程序,Keil 提供了以下几个重要的调试窗口。

1. 变量观察窗口

在调试状态下点击菜单 View→Watch & Call Stack Window 选项或点击  按钮,即可打开或关闭该窗口(若原先没有打开该窗口,执行该命令就可以打开该窗口;若原先已经打开了该窗口,执行该命令将关闭该窗口)。打开后的窗口如图 3-24 所示。

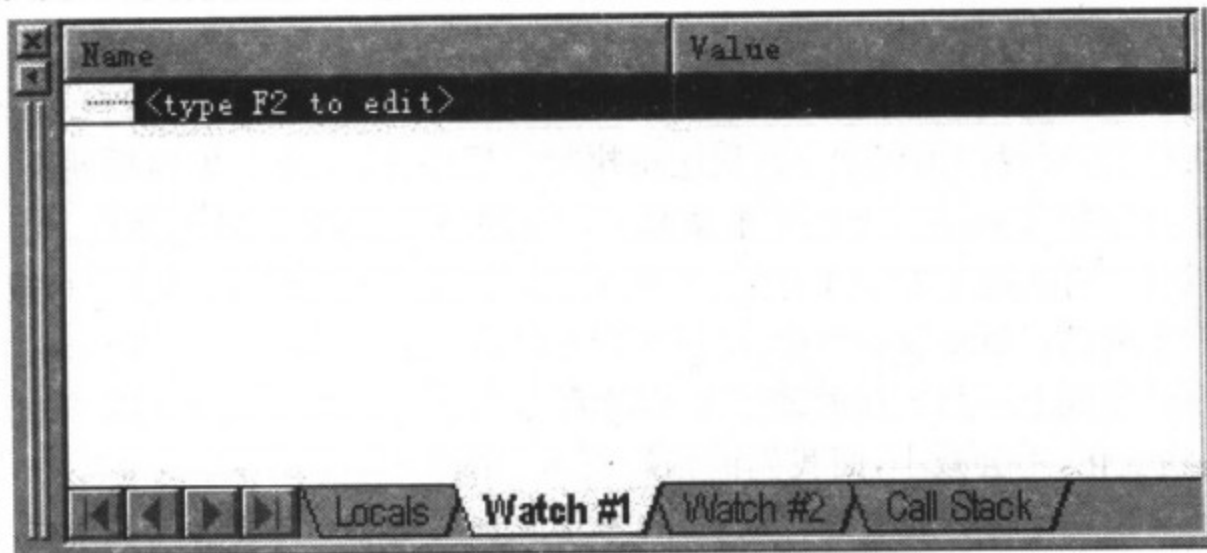



图 3-24 变量观察窗口

此窗门又包括 4 个小窗口(分 4 页显示),分别是 Locals、Watch #1、Watch #2 和 Call Stack。可以在 Locals 窗口中输入相应局部变量的值;也可以在 Watch #1、Watch #2 观察窗口中输入被调试的变量名,系统会自动在 Value 栏内显示该变量的值;而 Call Stack 观察窗口主要给出了一些调用子程序时的基本信息。

如果想观察寄存器 A 中的值,可以在 Watch #1 窗口中选中 type F2 to edit,然后按 F2 键,输入“A”即可。修改后的窗口如图 3-25 所示。

此时,点击全速运行按钮 ,会发现 Watch #1 窗口中寄存器 A 的值会不断地变化。

重点提示 一般情况下,仅在单步执行时才对变量值的变化感兴趣。全速运行时,变量的值是不变的,只有在程序停下来之后,才会将这些值的最新变化反映出来。但是,在一些特殊场合,也可能需要在全速运行时观察变量的变化。此时可以点击 View→>Pe-

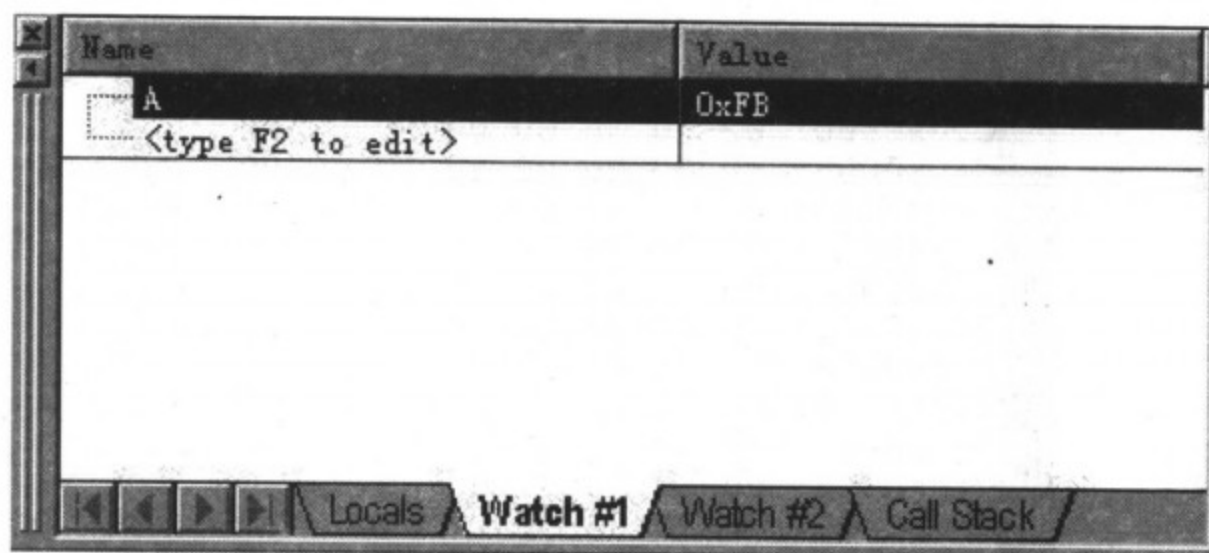



图 3-25 修改后的 Watch #1 窗口

periodic Window Update(周期更新窗口), 确认该项处于被选中状态, 即可在全速运行时动态地观察有关值的变化。但是, 选中该项, 将会使程序模拟执行的速度变慢。

2. 存储器观察窗口

在调试状态下点击菜单 View→Memory Window 或  按钮, 即可打开或关闭该窗口。此窗口也同样包括 4 个小窗口, 分别是 Memory #1~Memory #4, 如图 3-26 所示。

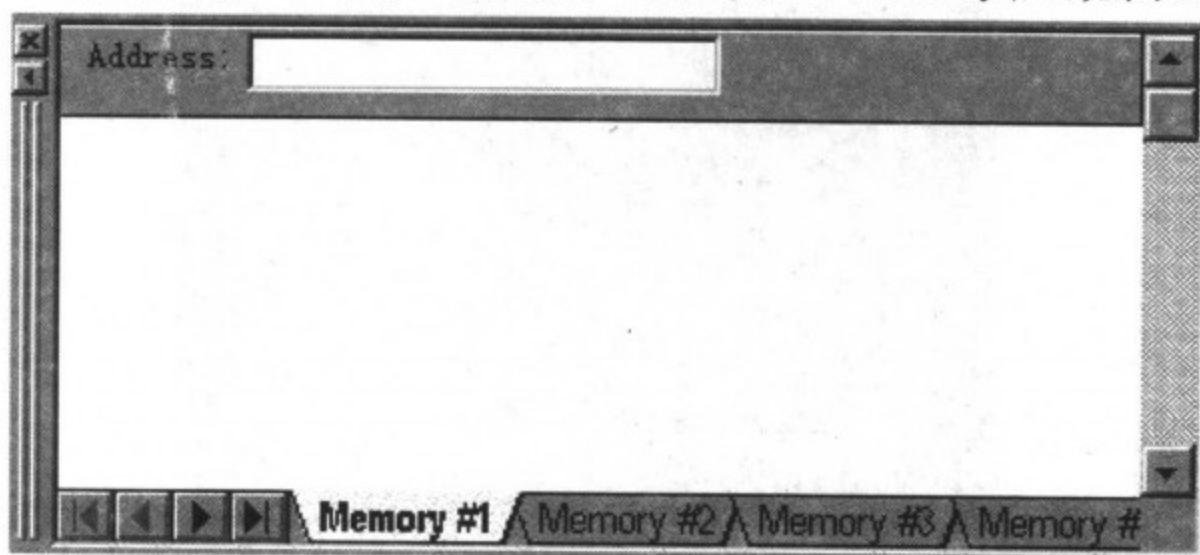


图 3-26 存储器观察窗口

通过这些窗口可以观察不同存储区不同单元, DATA 是可直接寻址的片内数据存储区, XDATA 是外部数据存储区, IDATA 是间接寻址的片内数据存储区, CODE 是程序存储区。可以在存储区观察窗口的 Address 栏内输入相应的字母(D、X、I、C)来观察不同的存储单元, 如输入 C:0x00, 则系统会给出从 00H 单元开始的程序存储器(ROM)及其相应的值, 即查看程序的二进制代码, 如图 3-27 所示。

该窗口的显示值可以各种形式显示, 如十进制、十六进制、字符型等, 改变显示方式的方法是点鼠标右键, 弹出如图 3-28 所示的快捷菜单。

该菜单用分隔条分成 3 部分, 其中第 1 部分与第 2 部分的 3 个选项为同一级别, 选中第 1 部分的任一选项, 内容将以整数形式显示, 而选中第 2 部分的 Ascii 项则将以字符型式显示, 选中 Float 项将以相邻 4B 组成的浮点数形式显示, 选中 Double 项则将相邻 8B 组成双精度形式显示。第 1 部分又有多个选择项, 其中 Decimal 项是一个开关, 如果选中该项, 则窗口中的值将以十进制的形式显示, 否则按默认的十六进制方式显示。Unsigned 和 Signed 后分别有 3 个选项: Char、Int、Long 分别代表以单字节方式显示、将相邻

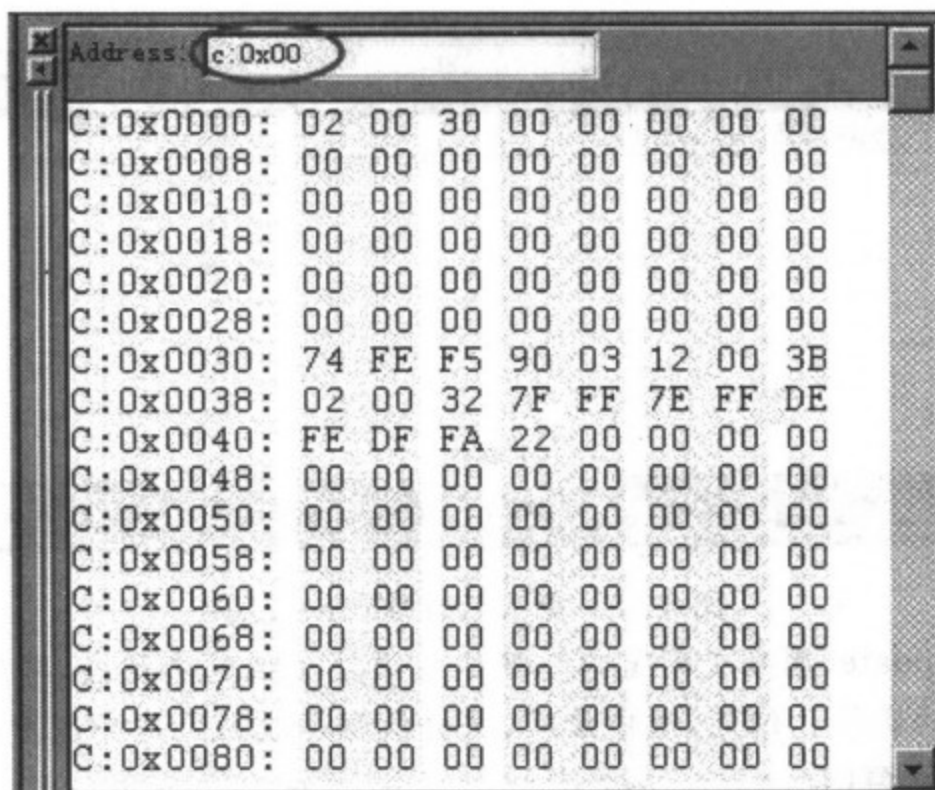


图 3-27 00H 单元开始的程序存储器及其相应的值

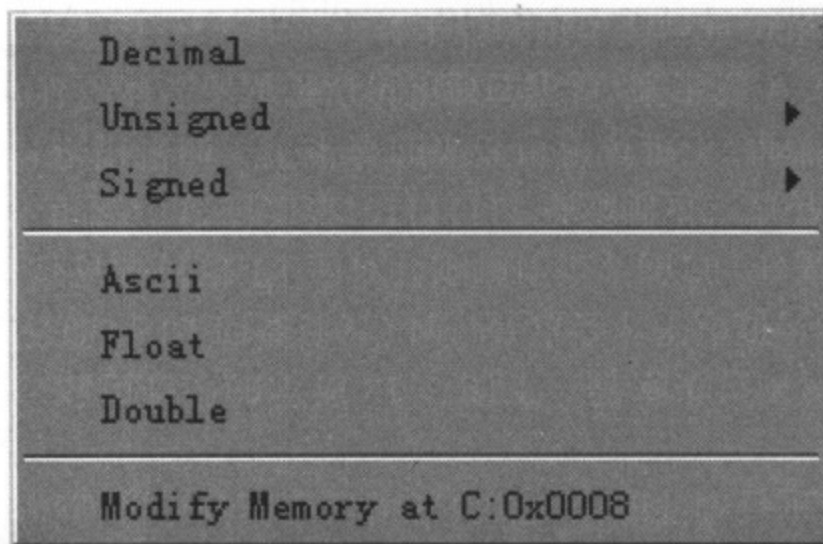



图 3-28 快捷菜单

双字节组成整型数方式显示、将相邻 4B 组成长整型方式显示。而 Unsigned 和 Signed 则分别代表无符号形式和有符号形式,究竟从哪一个单元开始的相邻单元则与设置有关。以整型为例,如果输入的是 I:0,那么 00H 和 01H 单元的内容将会组成一个整型数;而如果输入的是 I:1,01H 和 02H 单元的内容会组成一个整型数,依此类推。默认以无符号单字节方式显示,一般采用默认方式即可。第 3 部分的 Modify Memory at X:xx 用于更改鼠标处的内存单元值,选中该项即出现对话框,可以在对话框内输入要修改的内容。

3. 寄存器观察窗口

在调试状态下点击菜单 View→Project Window 选项或工具栏上的  按钮即可打开或关闭如图 3-29 所示的寄存器观察窗口。

寄存器窗口包括两组:通用寄存器组 Regs 和系统特殊寄存器组 Sys。通用寄存器组包括 r0~r7 共 8 个寄存器,而系统寄存器组包括寄存器 a、b、sp、pc、dptr、psw 和 sec(能够观察每条指令执行时间)等共 9 个寄存器。这些寄存器是程序中经常使用的和控制程序运行中至关重要的。通过观察这些寄存器的变化将更加有利于用户分析程序。

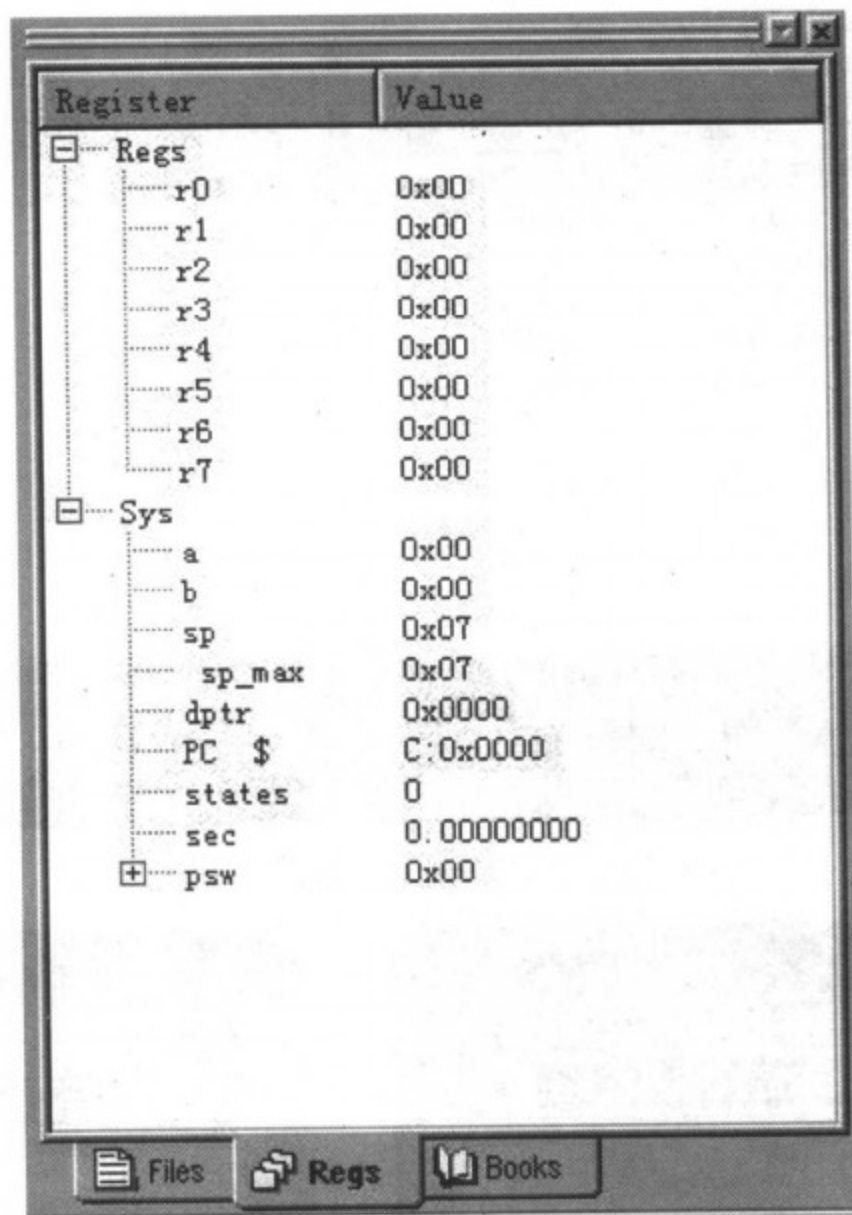


图 3-29 寄存器观察窗口

每当程序中执行到对某寄存器的操作时,该寄存器会以反色(蓝底白字)显示,单击鼠标,然后按下 F2 键,即可修改该值。


4. 串口调试观察窗口

在调试状态下点击菜单 View→Serial Window #1 或 SerialWindow #2 选项即可打开或关闭该窗口,该窗口提供了一个调试串口的界面,串口发送或接收的数据都可以在该窗口输出或输入。

5. 外围设备观察窗口

外围设备观察窗口通常包括中断系统观察窗口、I/O 接口观察窗口、串口属性观察窗口和定时/计数器观察窗口 4 个基本组成部分。这些外围设备观察窗口可以在调试状态下点击主窗口中的 Peripherals 菜单栏选项,在弹出的下拉菜单中选择相应的选项即可打开或关闭该外设的观察窗口。I/O 观察窗口在前面已作了介绍,中断系统观察窗口如图 3-30 所示,串口属性观察窗口如图 3-31 所示,定时/计数器观察窗口如图 3-32 所示。这几个窗口的使用方法将在第六章介绍中断、定时/计数器和串口通信时进行介绍。

6. 代码作用范围分析窗口

在程序中,有些代码可能永远不会被执行到(这是无效的代码),也有一些代码必须在满足一定条件后才能被执行到,借助于代码范围分析工具,可以快速地了解代码的执行情况。进入调试后,全速运行,然后按停止按钮,停下来后,使用调试工具条上的  按钮,可

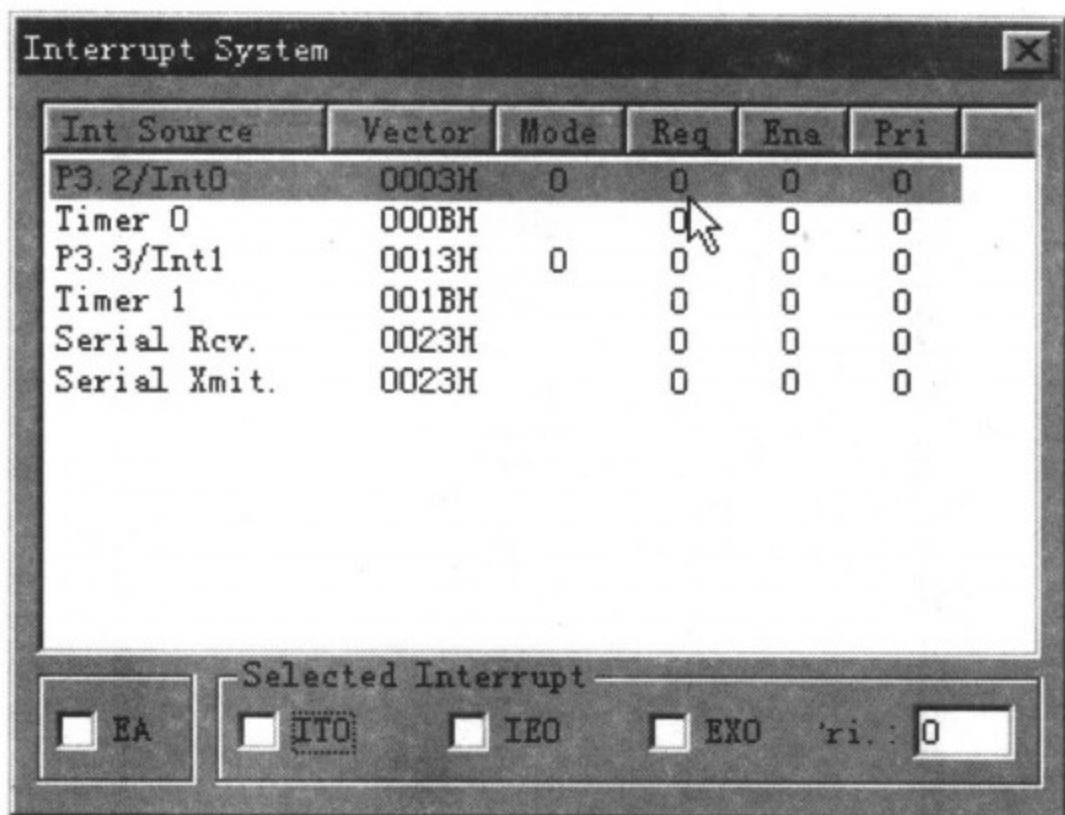


图 3-30 中断系统观察窗口

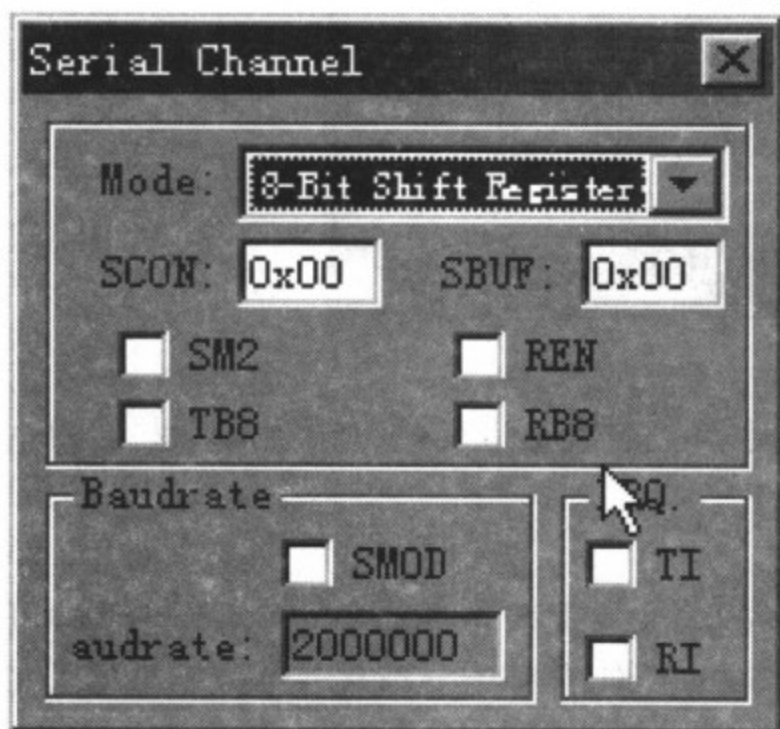


图 3-31 串口属性观察窗口



图 3-32 定时计数器观察窗口

打开代码作用范围分析的对话框,里面有各个模块代码执行情况的更详细的分析。同时,在源程序的左列有 3 种颜色:灰、淡灰和绿。其中,淡灰所指的行并不是可执行代码,如变量或函数定义、注释行等;灰色行是可执行但从未执行过的代码;而绿色则是已执行过的程序行。如果发现全速运行后有一些未被执行到的代码,那么就要仔细分析,这些代码究竟是无效的代码还是因为条件没有满足而没有被执行到。

7. 输出窗口

点击菜单 View→Out Windows 或按工具栏上的 按钮,可打开输出窗口,如图3-33所示。

进入调试程序后,输出窗口自动进入到 Command 页,该页用于输入调试命令和输出调试信息。

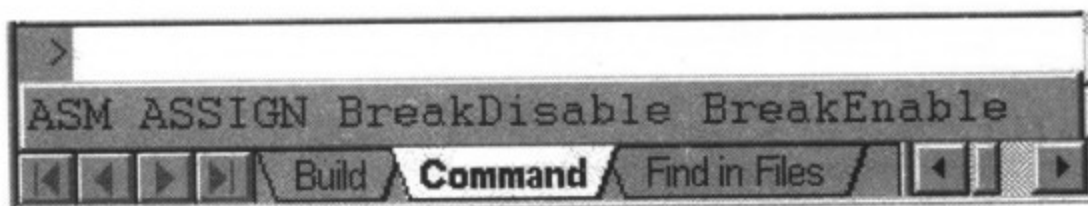


图 3-33 输出窗口

除以上几个窗口外,其他还有反汇编观察窗口等,这里不再一一介绍。

第二节 用单片机实验板进行仿真实验

仿真包括软件仿真和硬件仿真,在第一节,利用 Keil C51 做了一个 8 路流水灯程序的软件仿真,软件仿真不需要硬件,简单易行,但不够直接,要想真实地展现实验的效果,需要用单片机实验板和仿真器进行硬件仿真。

一、AT89C51 单片机实验开发板

AT89C51 单片机实验开发板是电子制作实验室(<http://www.xie-gang.com>)设计开发的,其外观如图 3-34 所示,电路原理图如图 3-35 所示。

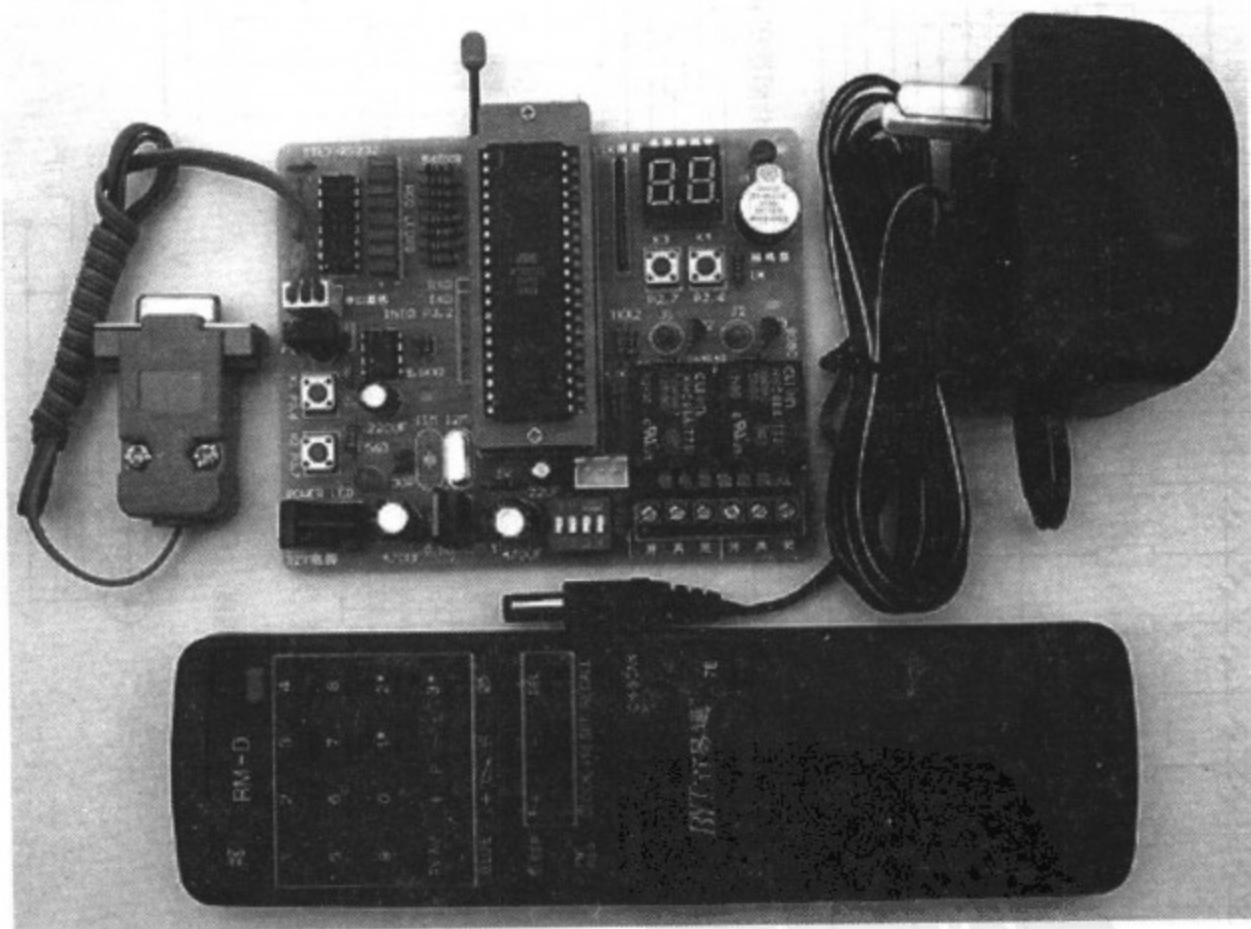
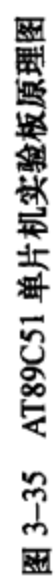


图 3-34 AT89C51 单片机实验板外观

1. 简介

AT89C51 单片机实验开发板上安装了 AT89C51、发光管、数码管、蜂鸣器、继电器等器件。板上有独立的稳压电路,只要向单片机中烧好程序,接上配套电源就可以运行。板上的发光二极管是为进行单片机 I/O 接口控制实验而设,用他可以进行流水灯实验、二进制显示以及其他实验的指示。板上的两只数码管用于 LED 显示器学习;同时也是学习其他程序时用于人机对话的窗口,例如定时器显示、外来数据显示、定时记数显示等。4



只按钮开关,用于外界对程序的控制,拨码开关可以模拟外来数据,即将拨码开关的数据让单片机读入。蜂鸣器用于单片机发声实验,同时也是其他实验声音提示的设备,另外,板上还设有红外遥控接收器,可用于各种红外线遥控和红外线数据传输。

2. 硬件电路

1) 电源电路

AT89C51 实验开发板需要配备输出直流电压为 10V~15V 的稳压电源,并且插头极性为内正、外负的电,稳压电源输出的直流电压通过专门的电源插座把直流电压引入实验开发板,加到实验开发板的电源电路,如图 3-36 所示。

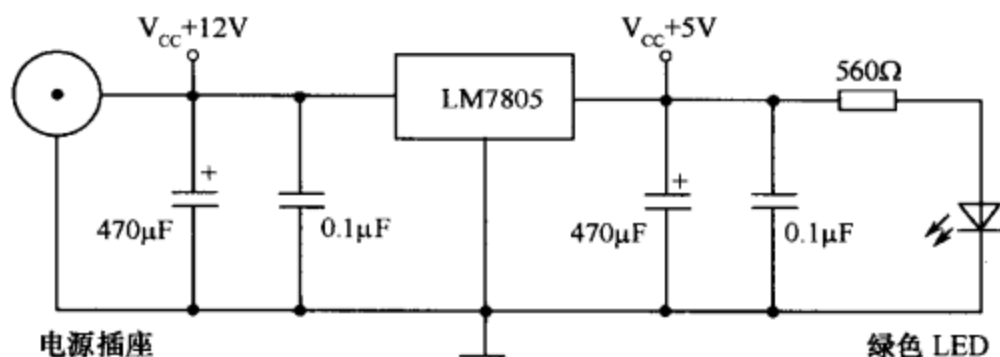


图 3-36 实验开发板内的电源电路

电源部分分为两路:一路直接提供 12V 的直流电源,主要是提供给继电器使用的;另一路通过三端稳压芯片 7805 稳压成 5V 直流电源提供给单片机系统使用,电阻和绿色的 LED 组成 5V 电源的工作指示电路,只要电源部分正常,绿色的 LED 就会点亮,我们可以根据这个 LED 来判断整个电源部分是否工作正常。

2) 时钟电路和复位电路

单片机的时钟电路有一个 12MHz 的晶振和两个 30pF 的小电容组成,它们决定了单片机的工作时间精度为 1μs(微秒)。复位电路由 22μF 的电容器和 1kΩ 的电阻组成,以使单片机可靠复位。

重点提示 判断单片机芯片及时钟系统是否正常工作有一个简单的办法,就是用万用表测量单片机晶振引脚(18 脚、19 脚)的对地电压,以正常工作的单片机用数字万用表测量为例:18 脚对地约 2.24V,19 脚对地约 2.09V。若怀疑复位电路有故障,可以采用模拟复位的方法来判断,单片机正常工作时 9 脚对地电压为零,可以用导线短时间和 +5V 连接一下,模拟一下上电复位,如果单片机能正常工作了,说明这个复位电路有问题。

3) AT89C51 的 31 脚(\overline{EA})

实验开发板上采用的是 AT89C51 芯片,它内部自带 4KB 的 Flash 程序存储器,一般情况下,这 4KB 的存储空间足够使用,所以将 AT89C51 芯片的 31 脚固定接高电平,只用芯片内部的 4KB 程序存储器。

4) 发光二极管接口

图 3-37 是实验开发板的发光二极管电路原理图。从图中可以看到,AT89C51 的 P1 端口接了 8 个发光二极管,这些发光二极管的负端接到端口引脚,而其正端则通过 8 个 560Ω 的电阻接到正电源端,这样,这些发光二极管发光的条件就是相应的端口为低电平。也就是说,如果 P1 端口某引脚输出为 0,则相应的灯亮;如果输出为 1,则相应的灯灭。

例如:

```
MOV P1, #0FH
```

该程序的意思是,将 0FH 送到 P1 端口,0FH(十六进制)转换成二进制就是 00001111B。因此,执行该程序后,接在 P1.0~P1.3 端口的发光二极管熄灭,而 P1.4~P1.7 端口的发光二极管点亮。

5) 数码管接口

P0 口和 P2 口的部分(P2.6、P2.7)引脚构成了 2 位 LED 数码管驱动电路,数码管有共阳和共阴两种类型,这里采用了共阴型。共阴型数码管对应的 a、b、c、d、e、f、g 是二极管的正极,所有二极管的负极连在一起,构成公共端,即片选端。

+5V 通过 1k Ω 的排阻直接给数码管的 8 个段位供电,P2.6(27 脚)和 P2.7(28 脚)端口分别控制数码管的个位和十位的供电,当相应的端口变成低电平时,相应的位可以吸入电流。单片机的 P0 口(32 脚~39 脚)输出的数据相当于将数码管不要显示的数字段对地短路,这样数码管就会显示需要的数字。

6) 音响接口

AT89C51 单片机的 P2.5(26 脚)口控制一个 S8550 的三极管,三极管控制电磁蜂鸣器的电源通断。

我们知道,声音的频谱范围约在几十赫到几千赫,若能利用程序来控制单片机某个口线的高电平或低电平,则在该口线上就能产生一定频率的矩形波,接上喇叭就能发出一定频率的声音,若再利用延时程序控制“高”、“低”电平的持续时间,就能改变输出频率,从而改变音调。

7) 按键输入电路

P3 口的 P3.6(16 脚)、P3.7(17 脚)和 P2 口的 P2.6(27 脚)、P2.7(28 脚)接了 4 个按钮开关 K1~K4,由于是实验板,没有过多考虑抗干扰等问题,所以没有给这 4 个引脚接上拉电阻,直接利用了单片机芯片内部的弱上拉。

正常情况下,单片机的 P3.6、P3.7 和 P2.6、P2.7 端口都被程序初始化时置“1”,当有按键按下时,对应的单片机引脚被按钮开关下拉为“0”,这种方法比较直观,而且比较简单,在按键数量不多的场合下使用很广泛。因为机械开关在开关时有抖动,所以需要在程序中加一个软件去抖动程序。

8) 四位拨码开关

四位拨码开关接在 P2 口的 P2.0~P2.3,当开关拨到 ON 一侧时,对应的那路就会接通,反之断开。它在单片机中一般用于设置初始参数,而且是不经常改变的场合。这里因为单片机引脚资源不够,所以只使用了拨码开关的第 1 位~第 3 位,第 4 位闲置。3 个开关可以逻辑组合出 8 种状态。

9) 继电器电路

两个继电器由 P2 口的 P2.3、P2.4 控制,有了继电器,就可以控制一些负载,使实验开发板的实用功能大大增强。

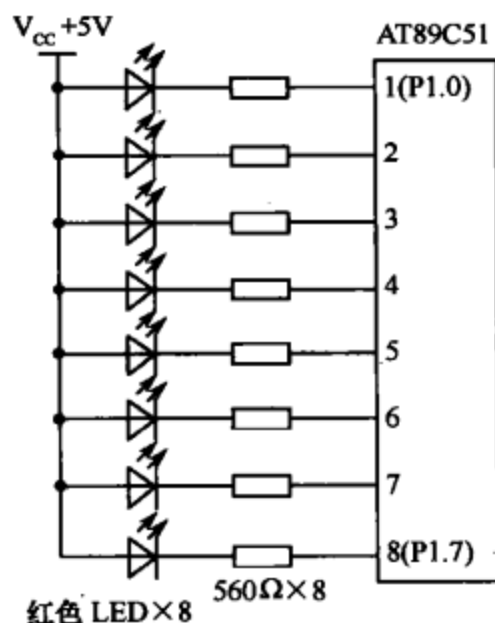


图 3-37 发光二极管接口

这里继电器由相应的 S8050 三极管来驱动,开机时,单片机初始化后的 P2.3、P2.4 为高电平,+5V 电源通过电阻使三极管导通,所以开机后继电器始终处于吸合状态,如果在程序中给单片机一条 CLR P2.3 或者 CLR P2.4 的指令,相应三极管的基极就会被拉低到 0V 左右,使相应的三极管截止,继电器就会断电释放,每个继电器都有一个常开转常闭的接点,便于在其他电路中使用,继电器线圈两端反相并联的二极管则起到吸收反向电动势的功能,保护相应的驱动三极管,这种继电器驱动方式的硬件结构比较简单。

10) 串行接口

通信功能是目前单片机应用中经常要用到的,本实验开发板设计了 MAX232 接口芯片,这样,该板就可以和 PC 机进行通信了。51 单片机的 P3 口的引脚 P3.0(10 脚)和 P3.1(11 脚)的第二功能是串行口 RXD 与 TXD,其内部的串行接口电路具有异步通信功能,但是单片机输出的信号是 TTL 电平,也必须以 TTL 电平作为输入,而 PC 机是标准的 RS232 接口,以 12V 为高电平,0V 为低电平。为获得电平匹配,实验板上扩充了一片 MAX232 芯片,该芯片是一个电平变换芯片,利用该芯片,可以将单片机的 TTL 电子变换为 RS232 所要求的电平,该芯片内部有电荷泵,只要单一 5V 电源供电即可自行产生高电压,使用非常方便,MAX232 的外围电路也很简单,只要 4 只电容即可,电容的值为 $0.1\mu\text{F}\sim 2\mu\text{F}$,这里使用 4 只 $0.1\mu\text{F}$ 的电容,实践证明,它们工作得很好。

要让实验板与 PC 机进行通信,必须配通信电缆,通信电缆可以使用普通的 3 芯线来制作,3 芯线的一端安装 3 芯插座,以便插到实验电路板上,另一端接一个 DB9 插座(母),以便插入 PC 机后的串行口(COM1 或 COM2)中。

11) 红外遥控接口

AT89C51 的 P3.2(12 脚)外接一个红外接收器。红外线接收器是一种集红外线接收和放大于一体,体积小巧,适合于各种红外线遥控和红外线数据传输。

二、MON51 仿真器

1. MON51 仿真器简介

MON51 是一款利用 Keil C51 的 IDE 集成开发环境作为仿真环境的廉价仿真器,它体积小巧,价格低廉,特殊适合使用 Keil 软件的用户使用。

这种仿真器的仿真 CPU 使用 SST 公司的 SST89C58 或 SST89C54(其他相容的芯片也可)。SST89C58 片内带有 32KB 的 Flash ROM,仿真程序时可以使用地址是 0000H~6FFFH,总共 28KB,对于一般的应用开发已足够使用了。7000H~7FFFH 这 4KB 被仿真器监控程序占用,监控程序用于和上位机软件 Keil 进行联络,7000H~7FFFH 用户不能使用,如果强行使用会造成仿真器监控程序覆盖修改,甚至导致仿真器不能正常工作,此时需要用可以用 SST89C58 RB1 位的编程器重新写入监控程序。不过因为在开发过程中作了特殊的烧录设置,用户的外部程序就不能强行使用和存取 7000H~7FFFH,这样就能够保证仿真器芯片内部的监控程序不被修改。

F-MON51 仿真器的外形如图 3-38 所示。

MON51 仿真器的主要性能如下:

- (1) 直接支持 Keil C51 的集成开发仿真环境。
- (2) 可单步、断点、全速运行调试程序。

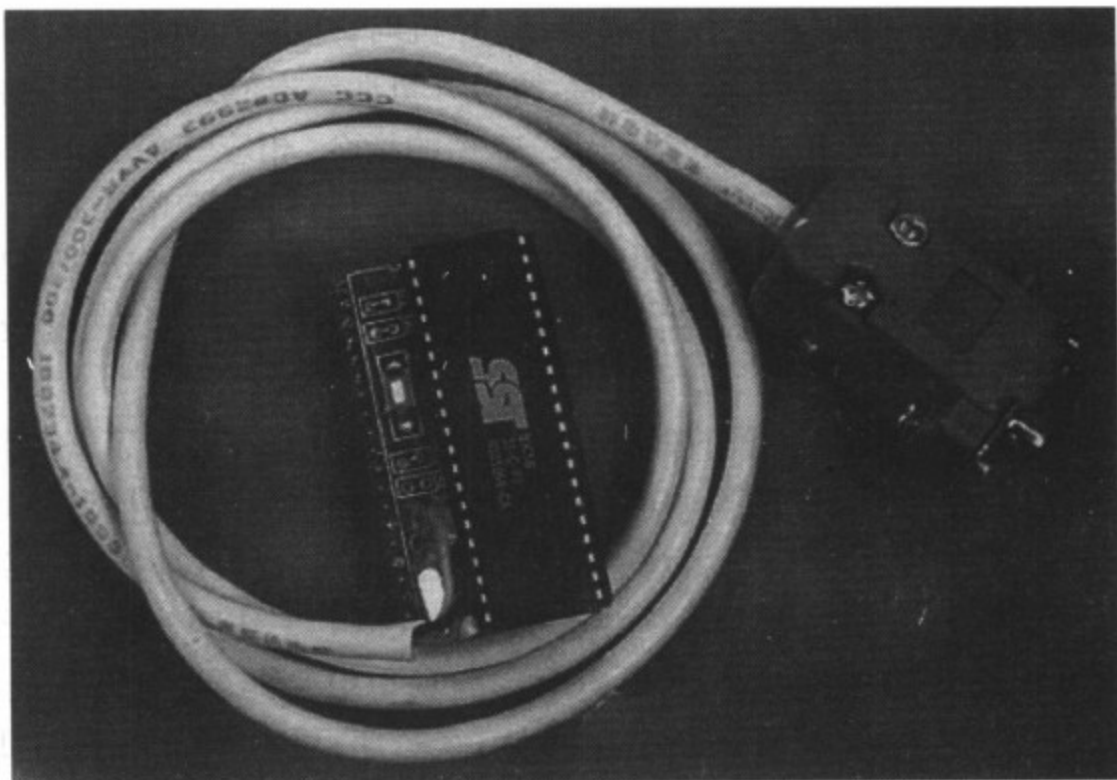


图 3-38 F-MON51 仿真器的外形图

- (3)支持汇编、C 语言混合调试。
- (4)片内 28KB 程序空间可以随时进行在线程序更新。
- (5)可以仿真标准的 89C51、89C52、89C58 等 51 内核的单片机。
- (6)带有复位按钮,可避免系统复位不良而引起的故障。
- (7)仿真频率为 6MHz~33MHz 晶振可选,系统配置频率为 11.0592MHz。
- (8)仿真器占用了单片机的串行口和定时器 2 的资源以及部分程序空间。

2. MON51 仿真器的原理

MON51 仿真器主要是利用了 SST89C58 的 IAP(In Application Programming)功能的英文缩写,是在应用编程的意思,通俗地讲就是:它可以通过串口将用户的程序下载到单片机中,可以通过串口对单片机进行编程。之所以具有这种功能,实际上是因为它有两块程序 Flash 区,其中一块 Flash 中运行的程序可以更改另外的一块程序 Flash 区中的程序,正是利用这一特性才用它做成了仿真器。我们把仿真器的监控程序事先烧入 SST89C58,监控程序通过 SST89C58 的串口和 PC 通信,当使用 Keil C51 的 IDE 环境仿真时,用户的程序通过串口被监控程序写入 Flash 程序区中,当用户设置断点等操作仿真程序时,Flash 程序中的用户程序也在相应地更改,从而实现了仿真功能。

3. MON51 仿真器的制作

该仿真器需要用串口连接 SST89C58 芯片和 PC 上位机进行通信数据传输,因此,需要制作一个带 RS232/TTL 转换的应用板,图 3-39 是可以用于制作 SST89C58/54 仿真器或 51 系列实验应用的小型应用板电路。

1)有编程器的做法

做好以上所说的电路后,就可以把仿真 CPU 的 .hex 文件烧到 SST89C58 里面,再把它插到上面的电路中就可以了。因为 SST89C58 有两个程序存储区,在这里要注意的是在烧写时就把仿真监控程序烧到 SST89C58 的第 2 个存储区,也就是的 RB1。烧写时要求用支持 SST89C58 的编程器,如用 TOP2000 编程器,烧写设置画面如图 3-40 所示。在

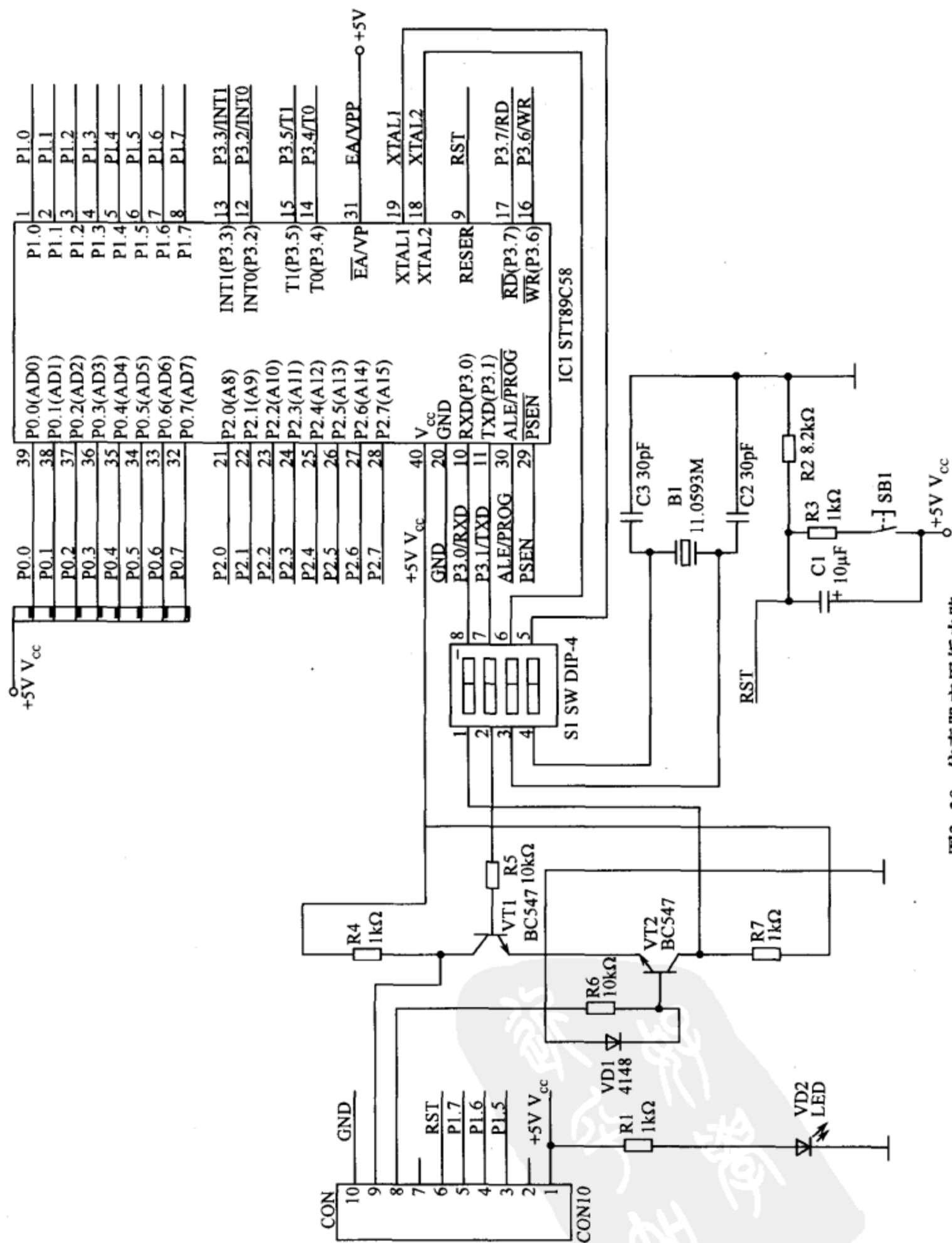


图3-39 仿真器应用板电路

这里要注意的是,烧写时不要对加密位 SB1~SB3 进行加密,并设存储位为 RB1,否则无法用 IAP 功能。各编程器的设置不尽相同,具体参看编程器的说明书。

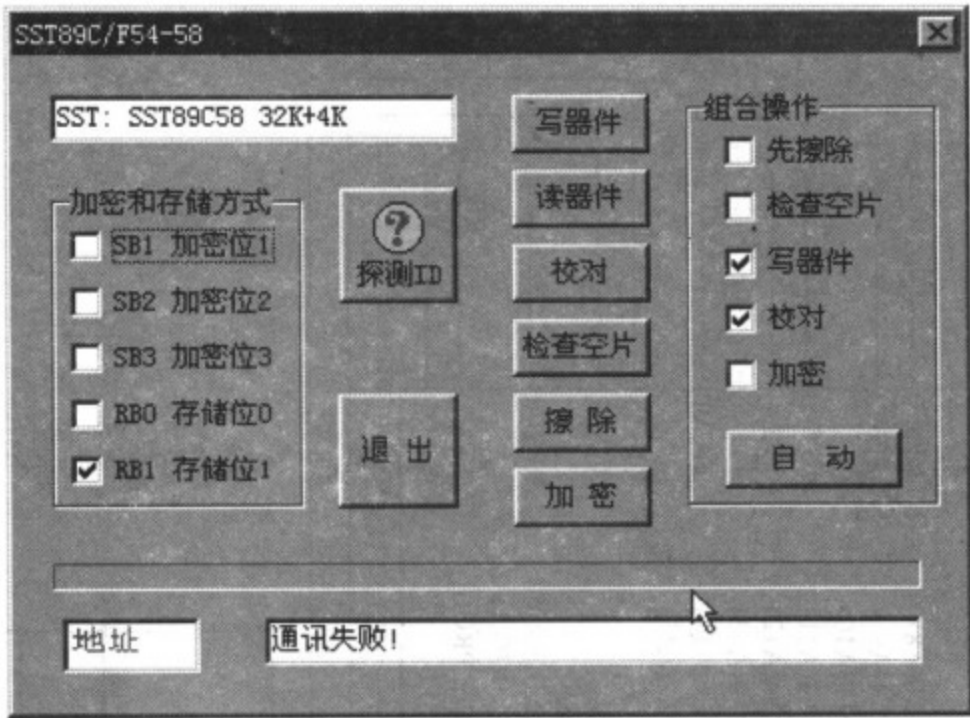


图 3-40 采用 TOP2000 编程器的烧写设置画面

重点提示 SST89C58 里的监控程序丢失是最主要的故障,表现为仿真机无法连接,或不能执行断点等操作,主要原因是意外删除了监控程序。用户碰到这个问题,可以用编程器将监控文件烧到 SST89C58 芯片里,如果重新烧录之后还是不行,这时可能是串口部分损坏。

2) 没有编程器的做法

如果手头上没有可以烧写 SST89C58 的编程器,应选用内置 BSL (BOOT-Strap Loader) 固件程序的 SST89C58 芯片。BSL 的意思就是“可引导装载”,形象来说,就像电脑用 DOS 启动盘。选好芯片后,将 SST89C58 芯片插到上面制作的应用板中,安装并运行 SSTEasyIAP11F.exe 软件(可从相关网站下载),将准备好的仿真监控程序 Soft-ICE58.hex(可从相关网站下载)加载到 SST89C58 中,此时, SST89C58 即具有仿真功能。

4. 用 MON51 仿真器进行硬件仿真

在线仿真器 MON51 是完全依托 Keil C51 软件强大的功能来实现仿真的,所以必须配合 Keil C51 软件才能工作,学习使用 F-MON51 在线仿真器的过程也就是对 Keil C51 软件的学习过程,关于 Keil C51 软件的软件仿真过程在前面已作了较为详细的介绍,下面简要介绍用 Keil C51 软件配合 F-MON51 仿真器和 AT89C51 实验板进行硬件仿真的过程。

(1) 启动 Keil C51 软件,建立工程,输入前面介绍的 8 路流水灯程序。

(2) 工程建立好以后,要对工程进行设置,以满足硬件仿真的要求。

点击菜单 Project→Option for target 'target1', 出现对工程设置的对话框,在 Target 页中,Xtal 后面的数值是晶振频率值,输入仿真器的工作频率(11.0592MHz),如图 3-41 所示。

在 Debug 页中,Keil 提供了两种工作模式,即 Use Simulator(软件模拟仿真)和 Use

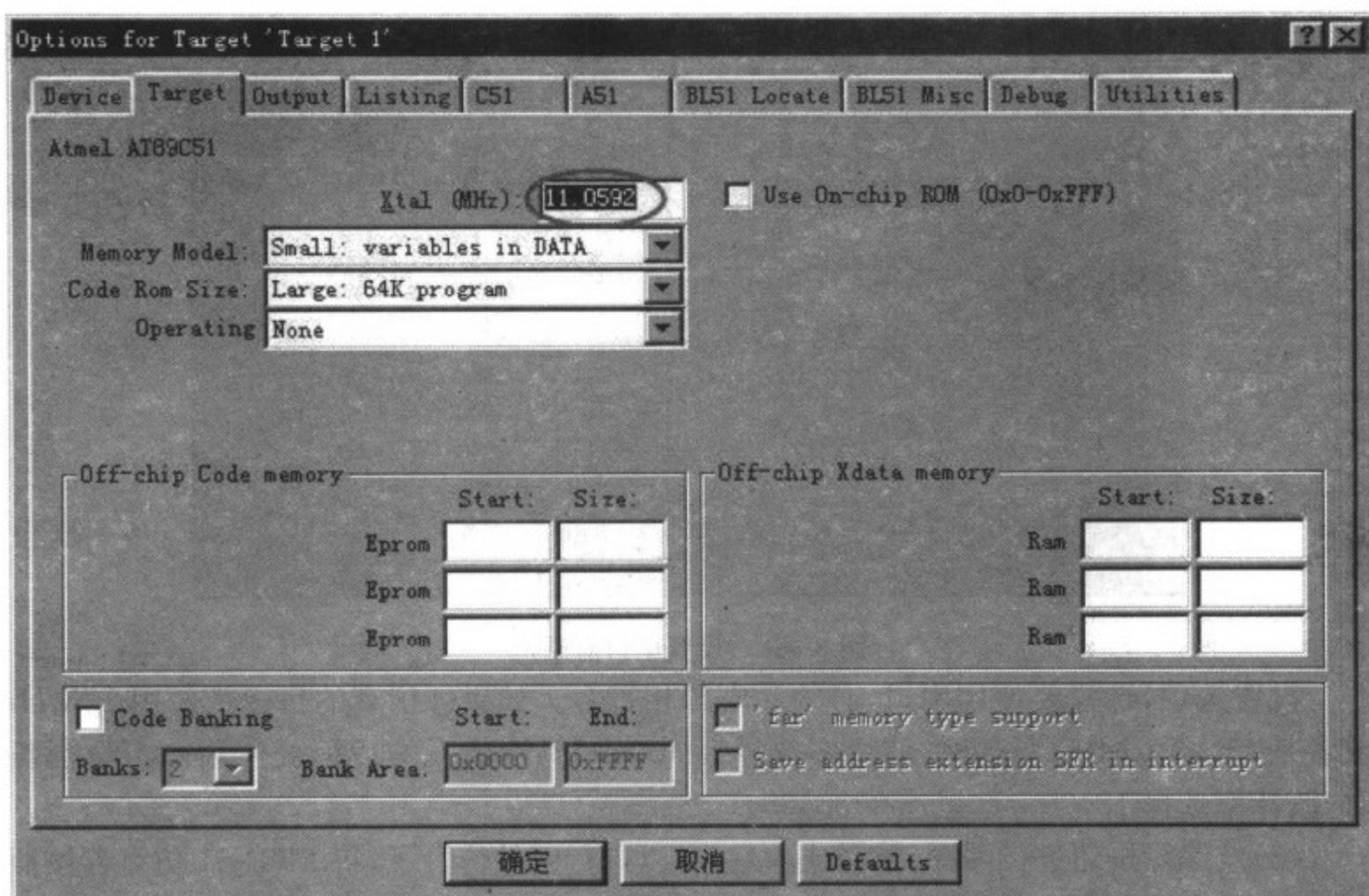


图 3-41 仿真频率的设置

Keil Monitor-51 Driver(硬件仿真),在这里,选择 Use,即将 Keil 配置为硬件仿真,如图 3-42 所示。

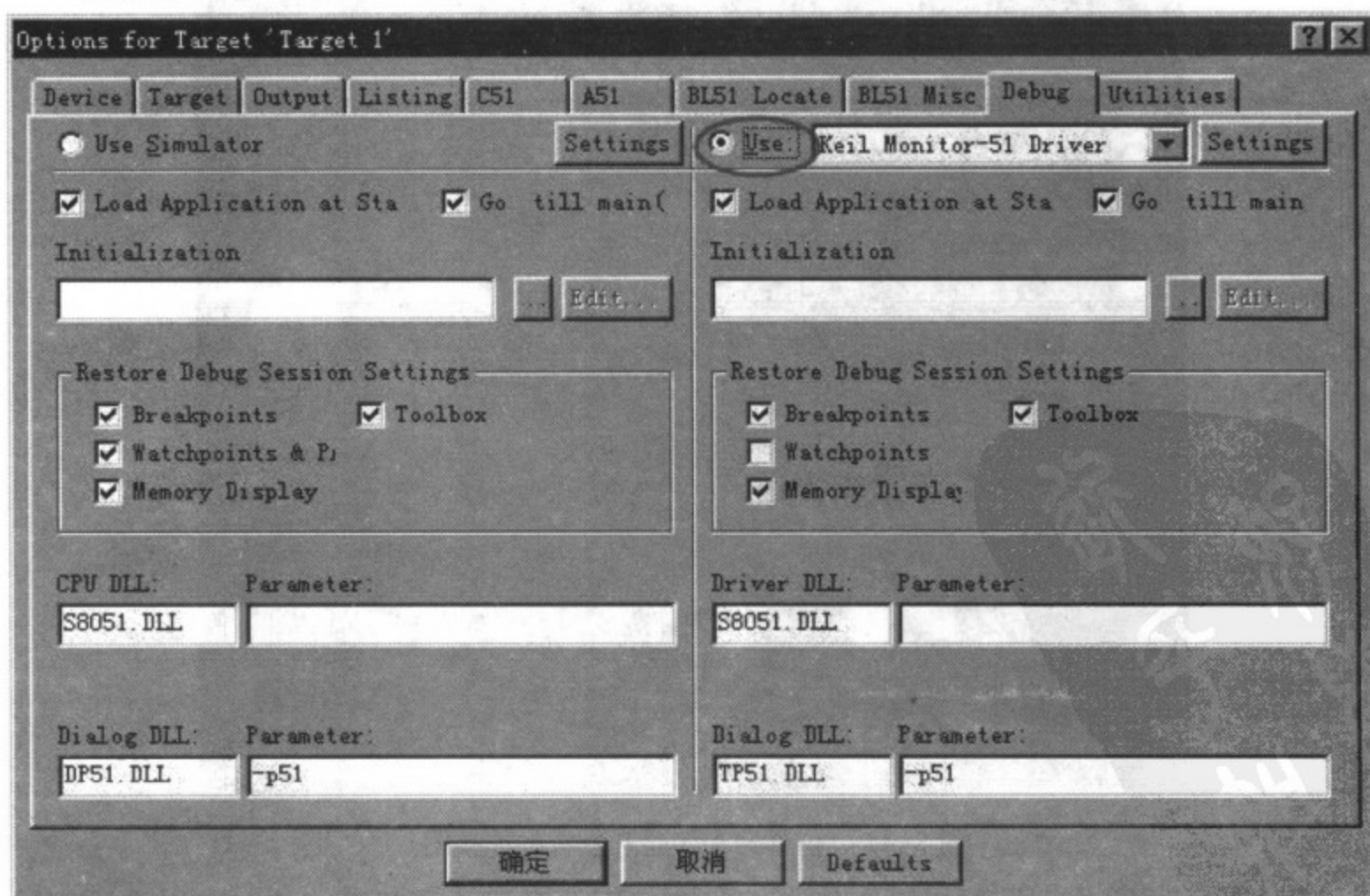


图 3-42 调试方式的设置

点击 Debug 页中的 Settings, 出现如图 3-43 所示的设置串口对话框。

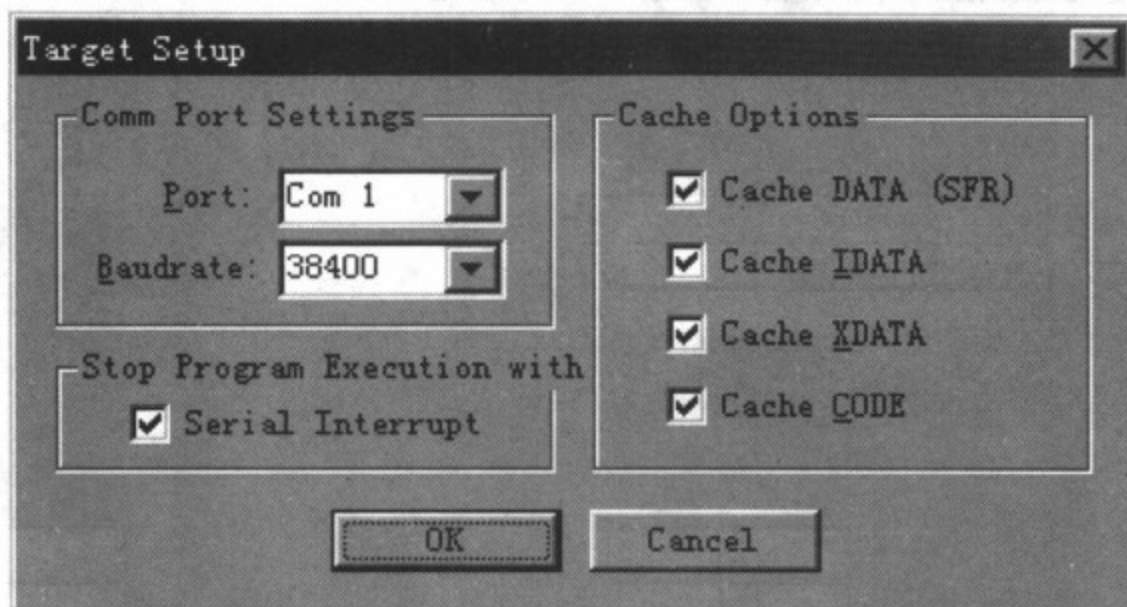


图 3-43 设置串口对话框

设置选项, 选择要使用串口, 必须和实际相符合。如果被仿真的目标板使用 12MHz 或者是 11.0592MHz 晶振时, 波特率选择 38400; 如果被仿真的目标板使用 6MHz 晶振时, 波特率选择 18400。这里选择 38400。

(3) 按编译按钮进行编译, 编译成功后, 可以进行硬件仿真了, 将 MON51 仿真器按照图 3-44 放入 AT89C51 试验开发板的 40 脚活动插座中, 这时仿真器的电源由实验开发板提供, 开始仿真时务必按一下仿真器上的复位按钮。

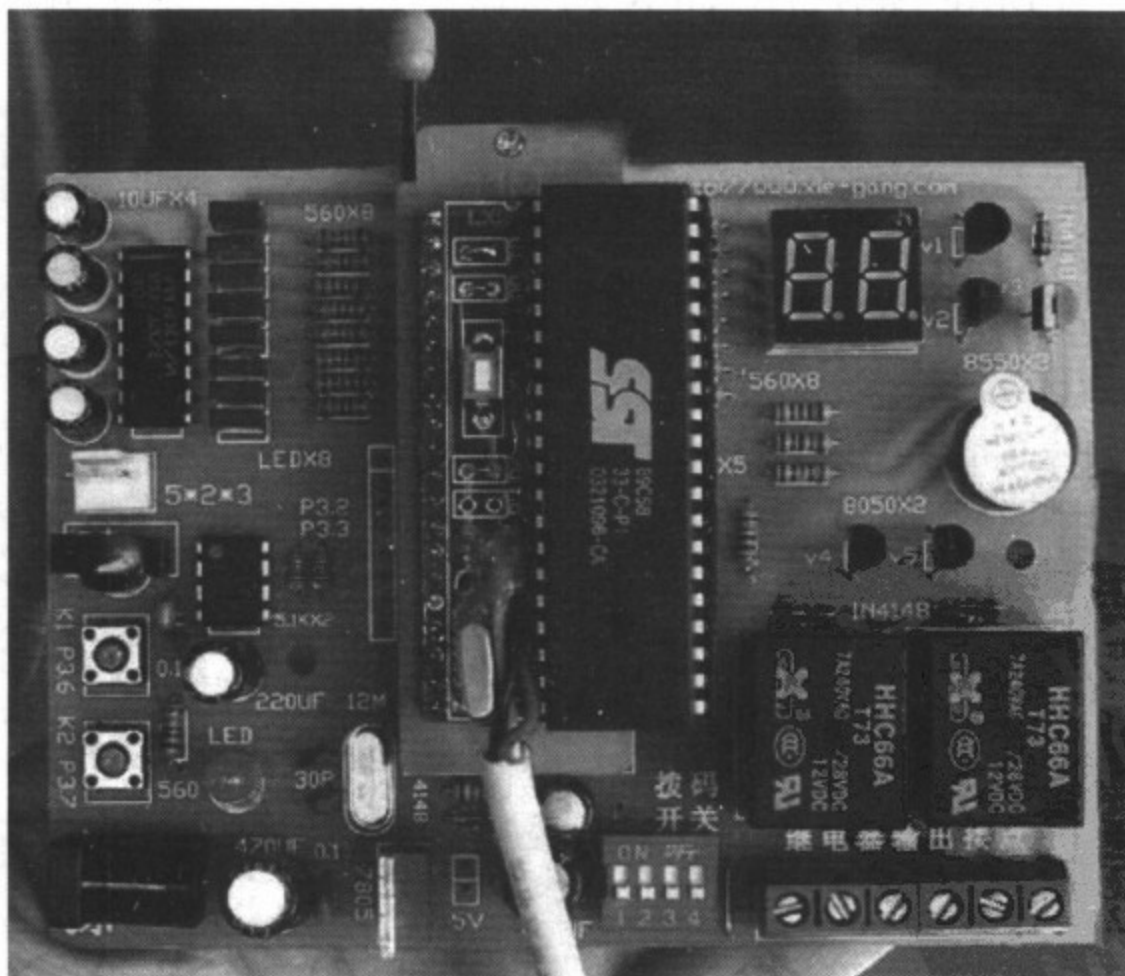


图 3-44 仿真器和实验开发板的连接图

(4) 现在按 Ctrl+F5 可以进入仿真, 这时再按 F5 全速运行状态。这时可以看到实验板开发板 P1 的 8 个红色 LED 轮流点亮, 表示运行成功。同时可以查看相关的变量和参

数,非常方便。

(5)当调试时出现如图 3-45 所示的界面时,说明 Keil C51 软件和仿真器之间通信失败,请先退出仿真然后按 F7,进行通信连接,再按 Ctrl+F5 可以进入仿真,这时再按 F5 全速运行状态。



图 3-45 通信失败对话框

注意 ①仿真器和实验开发板连接时,请注意方向,否则会烧毁 MON51 仿真器。②请在断电后再拔插通信线。③仿真程序所占内存请不要大于 28KB。

三、Insight SE-52 仿真器

Insight 系列单片机仿真开发系统是万利电子在仿真器领域开发的产品,可实时仿真 51 全系列单片机,并且对目标系统没有任何限制:不占用单片机任何资源,完全实时仿真 P3. 6、P3. 7、P0、P2 的口特性,完全实时仿真中断特性和定时器特性,完全实时仿真休眠方式和掉电方式;具有 8KB Frame/32bit 实时跟踪存储器 (TRACE),可清晰地看到仿真器实际运行的全过程。

使用万利 Insight SE-52 仿真器时,需配合万利公司的 MedWin 集成开发系统。下面结合 MedWin 的安装和使用介绍 Insight SE-52 仿真器的仿真方法和技巧。

(1)购买 Insight SE-52 仿真器时,会随机赠送 MedWin 集成开发系统光盘(也可从万利网站下载),将光盘插入光驱,按照提示要求即可安装 MedWin。安装好后,在桌面上生成 MedWin 图标。

(2)拆下 AT89C51 实验开发板上的 CPU(AT89C51),将仿真器的仿真头插入到 AT89C51 的位置,注意不要插反,并连接好仿真头与实验开发板上的地线。最后,将 SE-52 连接到 PC 的并口上,连接示意图如图 3-46 所示。

(3)点击桌面上的 MedWin 图标,启动 MedWin 开发系统,启动后,出现如图 3-47 所示的对话框。

因为要做实时在线仿真,所以,出现的对话框中选择“仿真器”。

(4)使用 MedWin 集成开发环境,配合 Insight SE-52 仿真器进行在线仿真时,必须对仿真器产品进行注册,点击菜单“帮助→注册仿真器”,在出现的对话框中输入正确的注册码后,即可注册成功。

(5)点击菜单“文件→新建”,在出现的对话框中输入 one. asm,即可建立名为

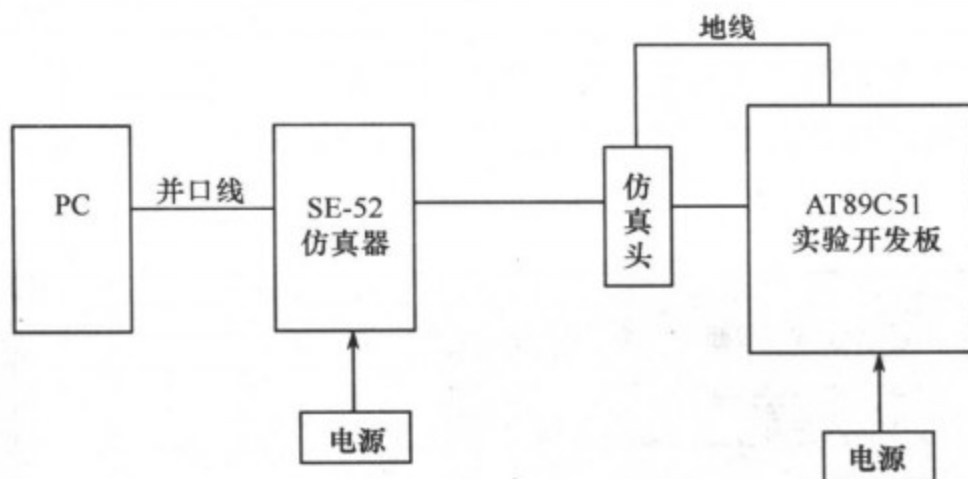


图 3-46 仿真器的连接示意图

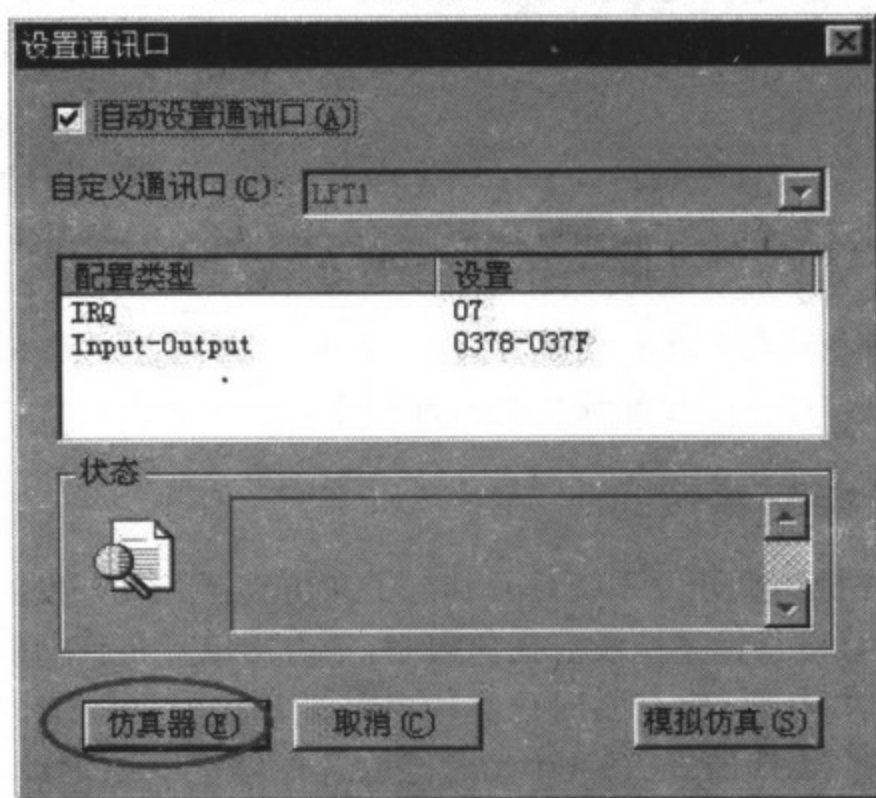


图 3-47 端口选择对话框

one.asm 的文件,如图 3-48 所示。点击“打开”,在出现的编辑窗口中输入前面介绍的“8路流水灯”源程序。

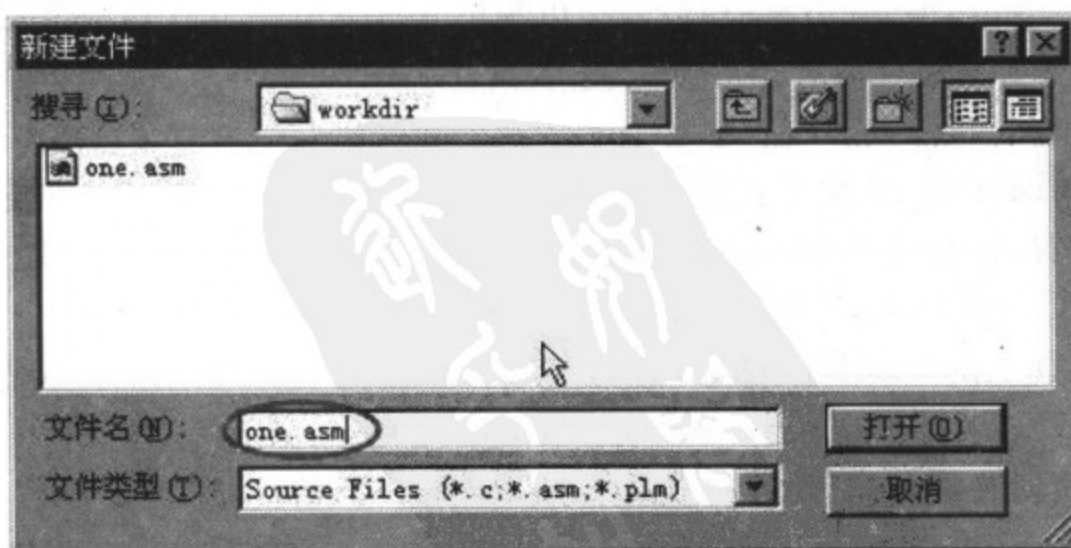


图 3-48 新建文件

(6)点击菜单“项目管理→编译/汇编”,汇编产生的结果出现在下面的消息窗口中,如图 3-49 所示。如果程序存在错误,程序将自动关联到源程序的错误处。

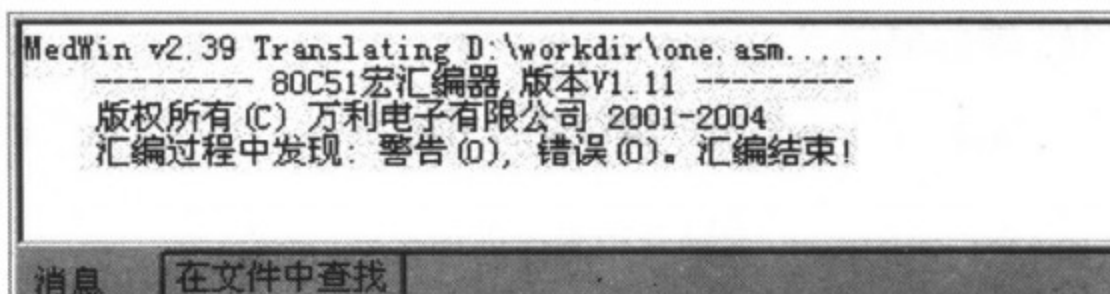


图 3-49 汇编结果

(7) 点击菜单“项目管理→产生代码并装入”，将产生代码装入仿真器，此时 MedWin 集成开发环境进入调试状态。MedWin 集成开发环境的文件窗口 one.asm 的左侧出现了一列小圆点，表示程序的有效行，即此行存在相应的代码，并且在 LJMP START 行的左侧出现黄色的箭头，表示当前的程序计数器，如图 3-50 所示。

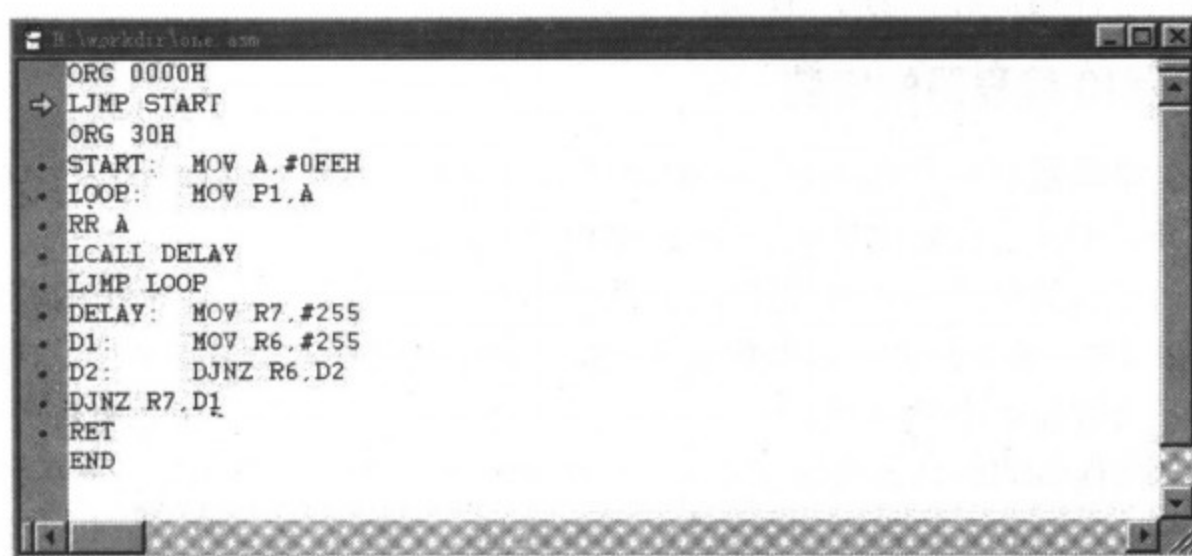


图 3-50 调试状态

(8) 点击“调试→全速运行”，程序即开始运行，这时可以看到实验开发板 P1 的 8 个红色 LED 轮流点亮，表示运行成功。

(9) 使用 MedWin 集成开发环境对程序调试无误后，选择“项目管理→输出 Intel Hex 文件”或“输出 Binary 文件”，产生输出 Intel Hex(16 进制)格式文件或输出 Binary(二进制)格式文件，用于对芯片的编程。

第三节 编程器及其使用方法

仿真调试通过后，使用编程器可将 .hex 目标文件烧写到单片机(或外接的可编程 ROM)中。编程器一般通过并口或串口与 PC 机连接，具有相应的服务程序；在连接好 PC 机与编程器后运行其服务程序，在服务程序中先选择所要编程的单片机型号，再调入 .hex 目标文件，编程器就将目标文件烧写到单片机中。烧写后，再将单片机插入到实验板上相应的插座上，则单片机可将程序的功能通过实验板表现出来。

目前，常用的编程器主要有两种，一种是通用编程器，另一种是下载型编程器。本节主要以 RF-810 为例，介绍通用编程器的使用方法，下载型编程器将在第四节和下载型实验板一同介绍。

一、RF-810 编程器简介

RF-810 编程器是一款性能较好的编程器,工作相当稳定,性价比也很好,其性能特点如下:

- (1)可对 100 余厂家的 1000 多种常用器件进行编程、测试。
- (2)采用 40 脚锁紧插座,与计算机并口(打印机)联机工作。
- (3)可自行调整烧录电压的参数,具有芯片损坏、插反检测功能,可以有效地保护芯片。
- (4)支持 EPROM、EEPROM、Flash ROM、串行 EEPROM、PLD、MPU 编程功能。
- (5)支持 TTL74/54、CMOS40/45、SRAM 测试功能。

RF-810 编程器配备全中文 Windows 驱动软件。软件功能齐备,对芯片的编程不需要人工干预,软件用户界面易学,使用相当方便。

二、RF-810 编程器的安装

RF-810 编程器套件包括 RF-810 编程器主机、并口电缆、并口匹配器、AC/DC 电源适配器等。RF-810 的安装包括硬件安装和软件安装。










硬件安装时,首先关闭计算机的电源,先把并口匹配器安装在计算机的打印口上,然后将联机电缆的一端连接在并口匹配器上,另一端连接在编程器主机的接口上。接通交流电源,找开编程器的电源开关,编程器主机上的电源灯亮(红灯),表示电源开始供电。

RF-810 编程器的软件安装也很简单,和普通软件的安装方法相同,下载最新版的软件后,运行 Setup. exe 安装程序即可,这里不再详述。软件安装完毕后,自动在桌面上形成 RF810 编程器的图标。

三、RF-810 编程软件的功能

软件和硬件安装完毕后,点击 RF-810 编程器的图标,进入图 3-51 所示的主菜单。

主菜单包括五大部分:功能项菜单栏、工具栏、工作区、信息栏和状态栏。系统把常用的功能做成了快捷方式,放在工具栏中,点击这些图标,就可以直接进行该操作。这些快捷方式有:

-  选择与芯片相对应的厂家、类型、型号、容量等。
-  对缓存区内容进行修改、浏览操作。
-  按擦除、查空、编程、校验等操作顺序自动完成对器件的全部操作过程。
-  检查器件是否处于空白状态。
-  对电可擦除的器件的内容进行删除。
-  把缓存内的数据写入到芯片内并进行校验。
-  把器件内容读入到缓存区。
-  校对器件内容和缓存区内容是否一致,并列出有差异的第 1 个单元的地址。
-  逐单元比较器件内容和缓存区内容有无差异,并将有差异的单元列表显示。

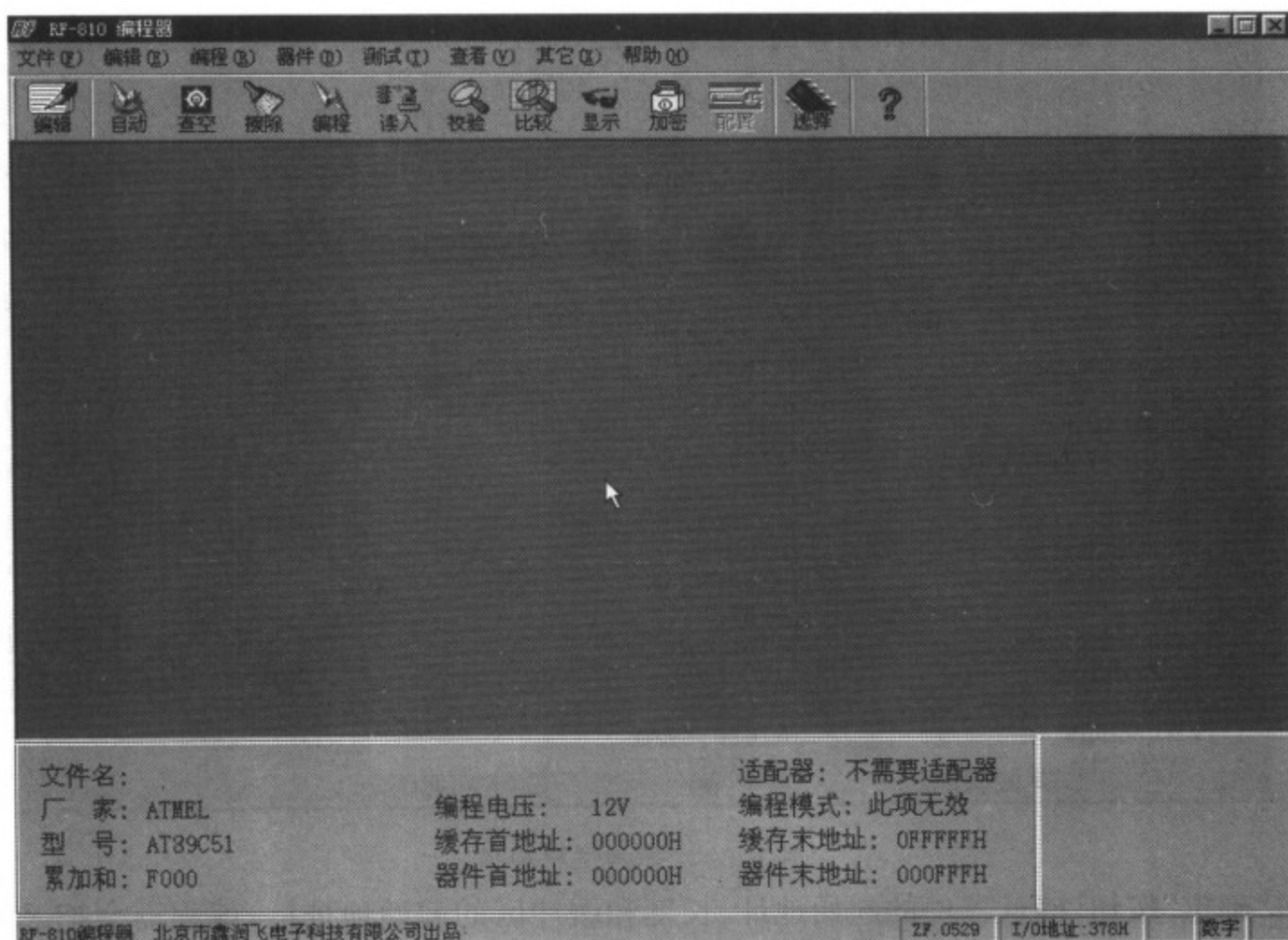


图 3-51 RF-810 编程软件的主菜单

 将器件内容显示在屏幕上供浏览。


四、RF-810 编程软件的使用

下面以读、写一片 AT89C51 单片机芯片为例,介绍 RF-810 编程软件的使用方法。


把待读写的 AT89C51 插到编程器上的 ZIP 插座上,在 ZIP 插座旁有几块芯片重叠的示意图,这至关重要,它告诉使用者芯片插入的正确方向。在半导体芯片上,其陶瓷封装的一边有一个缺口,表示芯片管脚的排列方向,插入时,要注意芯片上缺口的位置和编程器上 ZIP 插座旁芯片方向示意图对应。

1. 备份操作

为了确保操作万无一失,不管进行什么操作,一般都要把芯片的内容备份下来。

(1)在进行读、写操作前,都必须首先选择器件。点击工具栏上的  按钮,弹出“器件选择”窗口,在 EPROM、SPROM、EEPROM、PLD、MPU/MCU 等几种类型中选择所需的芯片类型(MPU/MCU),然后进行厂家选择,在厂家选择区中找到芯片对应的厂家名称并点击(ATMEL)。如果芯片类型选择不当,因不同类型的芯片管脚工作电压不同,有可能烧毁芯片;选择芯片的厂家后,再进行器件选择,在器件型号选择区中找到所需的型号并点击 AT89C51,如图 3-52 所示。

在全部选择完毕后,点击“确定退出”按钮完成器件选择操作。此时选定的型号作为当前型号在主菜单信息栏中列出,与其相应的其他信息也一并列出。

(2)点击  按钮,弹出“读入操作”窗口,如果刚才选择的芯片的类型和容量都正确,

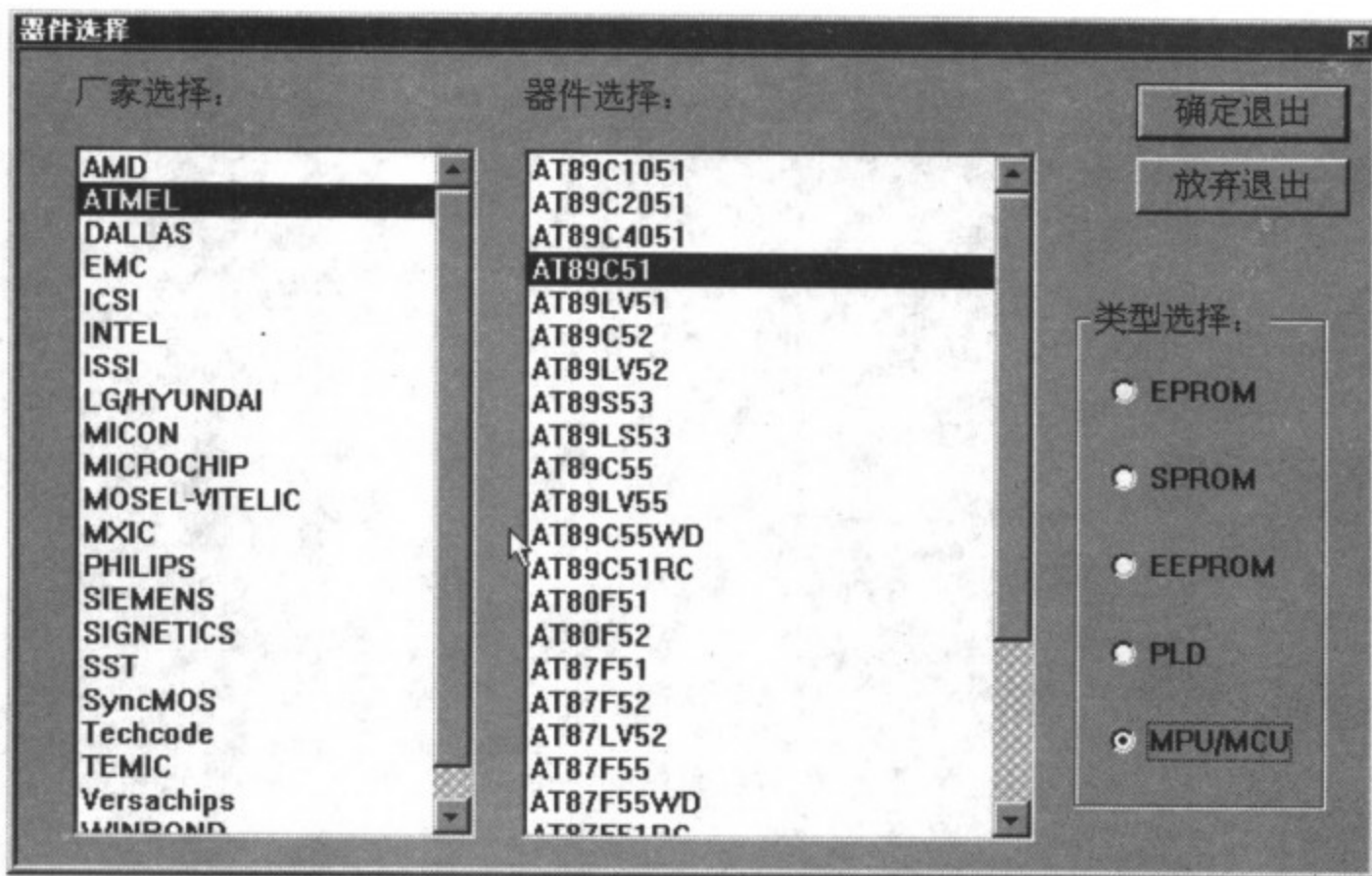


图 3-52 器件选择窗口

点击“确定”按钮，确认器件的起始地址、长度和缓冲区的起始地址后，编程器会把芯片内的数据读入 PC 机的内存中，如图 3-53 所示。

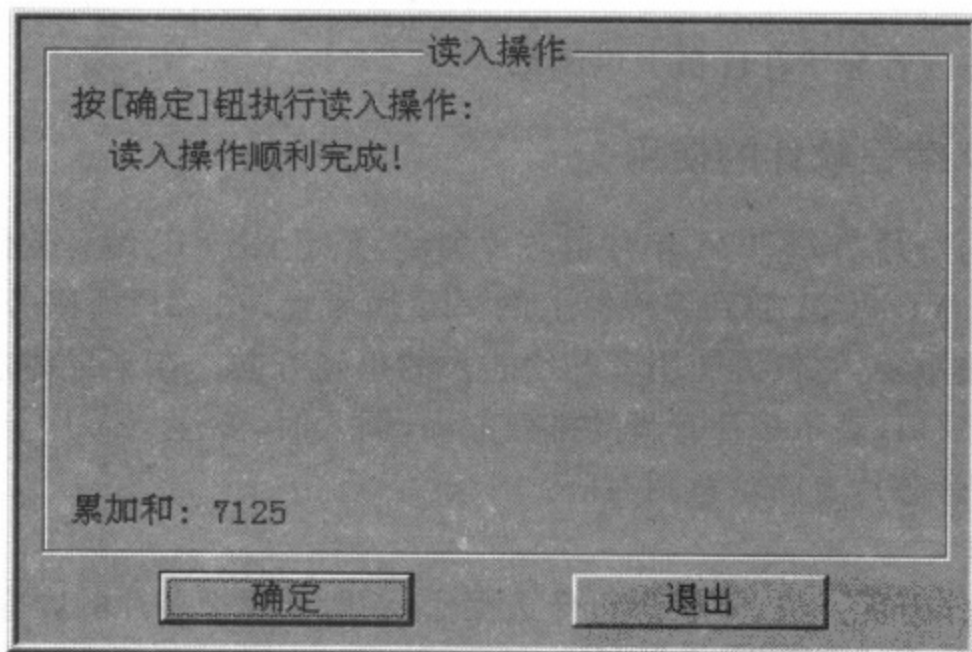



图 3-53 读入窗口

(3) 要对缓存区内的数据进行编辑，可点击工具栏上的  按钮，出现编辑画面，如图 3-54 所示。

将光标移向所要修改的位置，点击鼠标左键出现闪烁光标，在闪烁位置可以用键盘输入十六进制字符串。

注意 如果没有特别的需要，建议不要对内容作任何改动。

(4) 为了防止以后的操作失误，把缓存区内的数据保存成文件。选择“文件”菜单中的“保存文件”选项，点击“保存文件”操作项，弹出所要存盘的缓存区的首末地址及存盘格式



图 3-54 编辑画面

窗口,如图 3-55 所示。默认的文件是二进制格式,扩展名为. bin,这里选择十六进制(Intel Hex),扩展名为. hex。

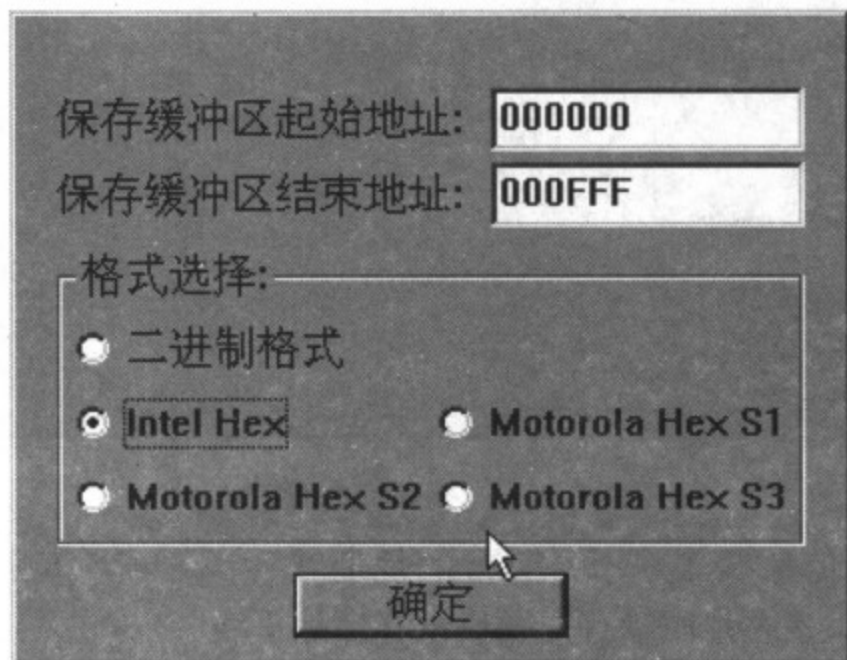




图 3-55 “保存文件”操作项

“确定”后,出现如图 3-56 所示的“保存文件”对话框。输入要保存的文件名后,点击“保存”按钮。

2. 写入操作

(1)在写入目标代码之前,要先把片内的数据进行擦除,使其处于空白状态。点击  按钮,弹出“擦除操作”窗口,如图 3-57 所示。如果器件已正确放置,点击“确定”按钮开始擦除操作。

(2)擦除之后要进行查空,以确认当前芯片中的数据信息是否全部为空,这需要点击  按钮。弹出“查空操作”窗口后,检查器件放置是否正确,点击“确定”按钮开始查空操作,系统开始对芯片进行空片检查。在检查过程中,滚动条不断显示检查单元的总数、百分比和字节总数,按 Esc 键可终止空检查操作。检查完毕后,如果器件为空白,则提示“查空操作顺利完成”,如图 3-58 所示。此时,缓冲区内的数据信息应显示为 FF,也就是没有

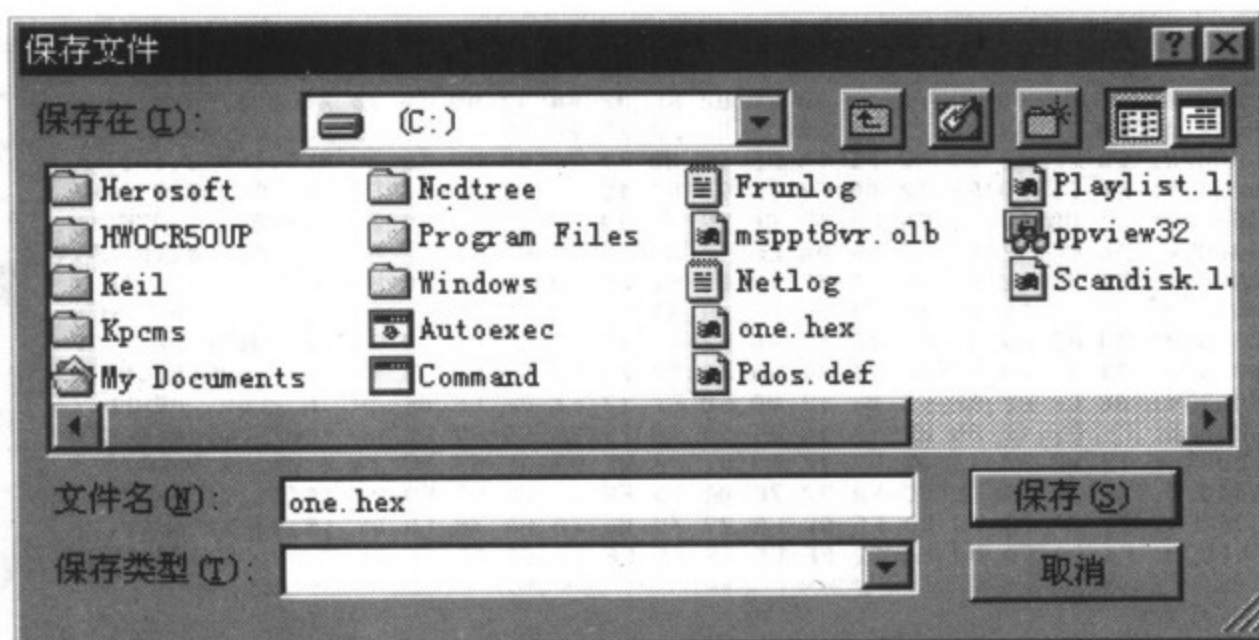


图 3-56 “保存文件”对话框

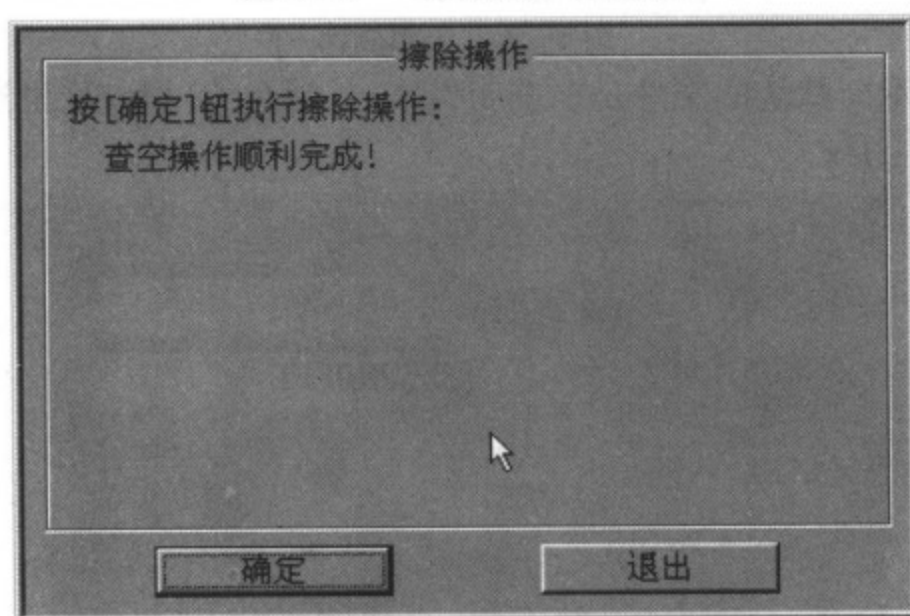


图 3-57 “擦除操作”窗口

任何数据。如果数据信息不全为 FF, 会提示“地址××××××查空错误”, 表示芯片中的部分代码清除不掉, 该芯片质量有问题或没有擦除干净。

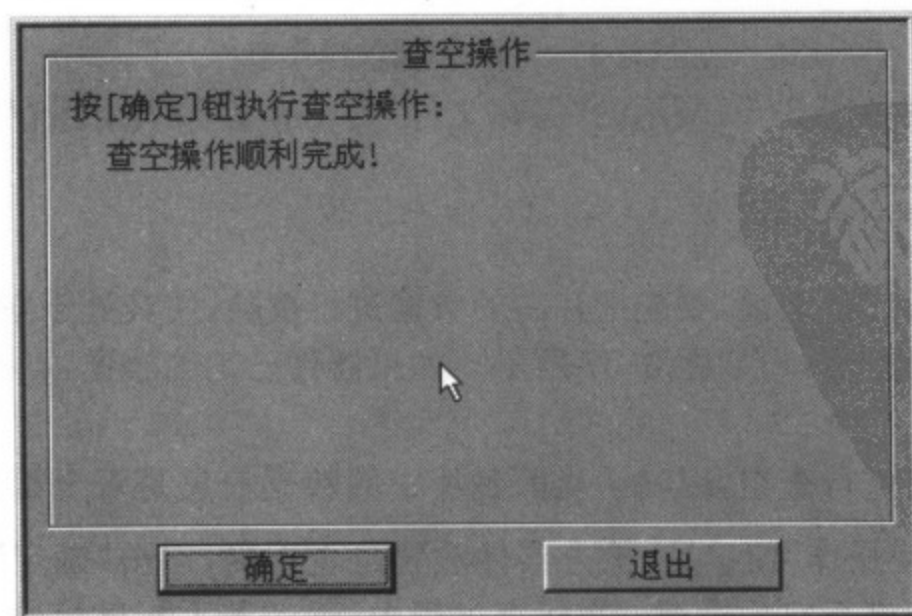


图 3-58 “查空操作”窗口

(3) 确认芯片已完成清空后, 点击菜单栏中的“文件”项, 在下拉菜单中选择“读入文件”操作项, 弹出“文件格式”选择菜单, 如图 3-59 所示, 共有 4 种格式可以选择, 分别是二进制格式(B)、Intel Hex 格式(I)、Motorola Hex 格式(M)等。

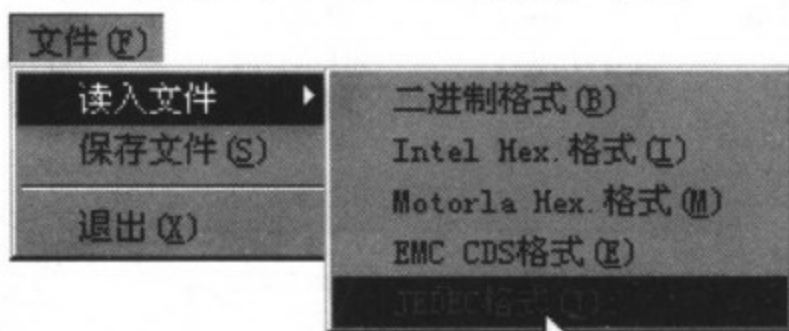


图 3-59 “文件格式”选择菜单

(4) 点击所需的格式(Intel Hex), 确认装入数据缓冲区的首址(默认地址为 000000, 一般不需修改)和读入方式(默认方式为“全部读入”)后, 如图 3-60 所示。

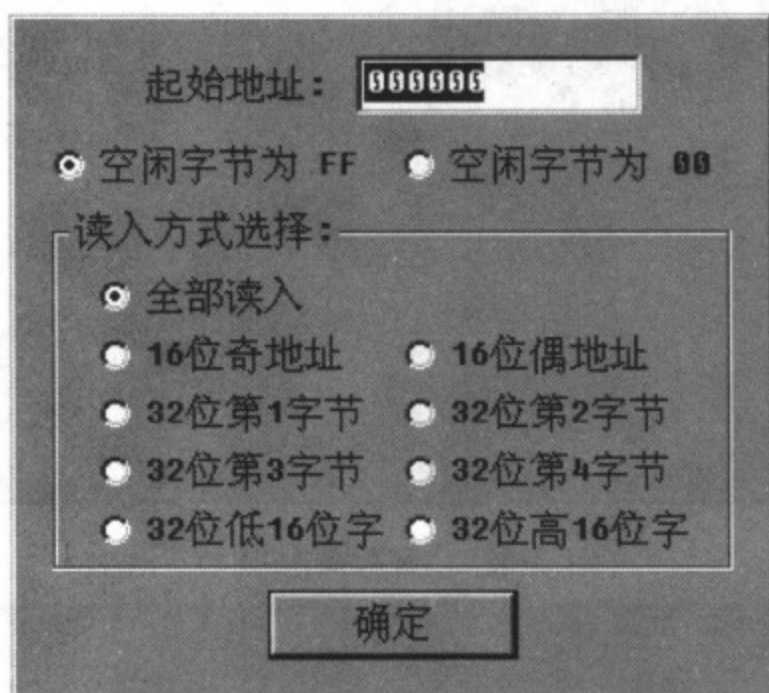




图 3-60 读入方式

(5) 点击“确定”按钮, 出现如图 3-61 所示的选择“读入文件”对话框, 点击“打开”按钮, 从硬盘上把以前保存的文件 one. hex 读入到内存中。文件调入缓存后, 计算机根据该文件调入缓存区的起始地址和文件长度, 计算出文件在缓存区中的安放位置并提示。

(6) 点击  按钮, 弹出“编程操作”窗口, 点击“确定”按钮开始编程操作, 系统确认缓存的起始地址、长度和器件的起始地址后, 把当前内存中的数据写入芯片中, 如图 3-62 所示。屏幕提示编程进度, 编程操作结束后, 自动进行校验操作。如某一操作失败, 将提示错误信息。

(7) 写入完毕后, 为了保证数据的完整性和检验写入操作是否正确, 可再点击  按钮, 弹出“校验操作”窗口, 点击“确定”按钮开始校验操作。RF-810 编程器的最大的特点是: 可自行调整烧录电压的参数及作 V_{CC} 、 $V_{CC} \pm 5\%$ 甚至最严格的 $V_{CC} \pm 10\%$ 的验证。在校验过程中, 编程器读出芯片内容和缓存区内的数据进行对比, 滚动条会不断显示完成校验的数据的百分比。如数据校验通过, 完全无误后, 屏幕会出现“校验操作顺利完成”的字样, 如图 3-63 所示。

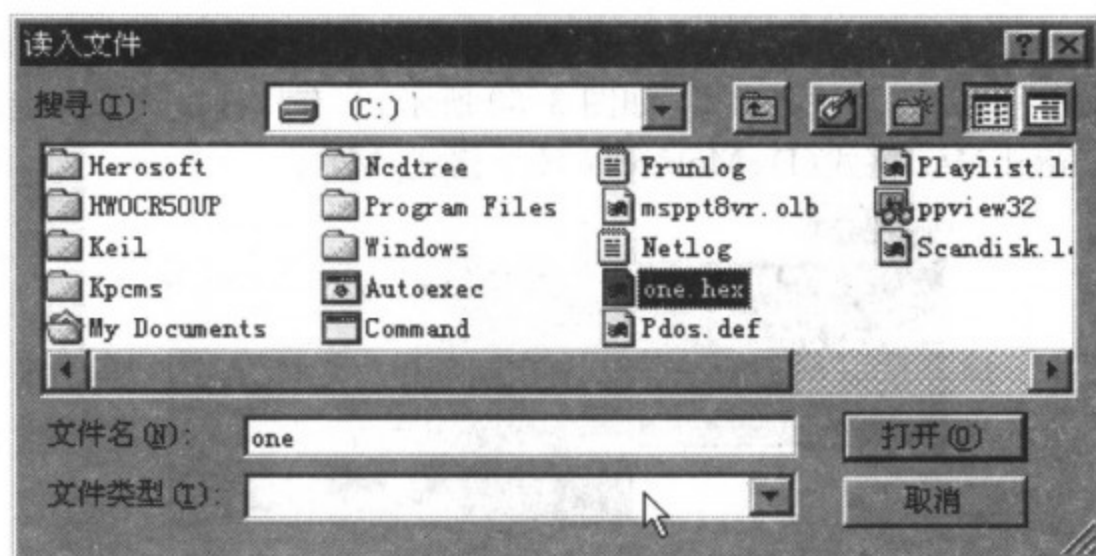


图 3-61 “读入文件”对话框

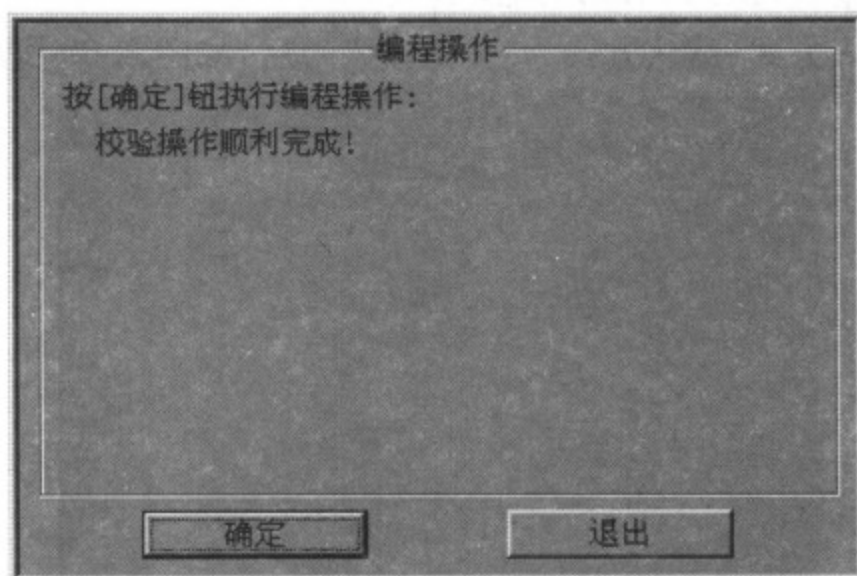


图 3-62 “编程操作”窗口

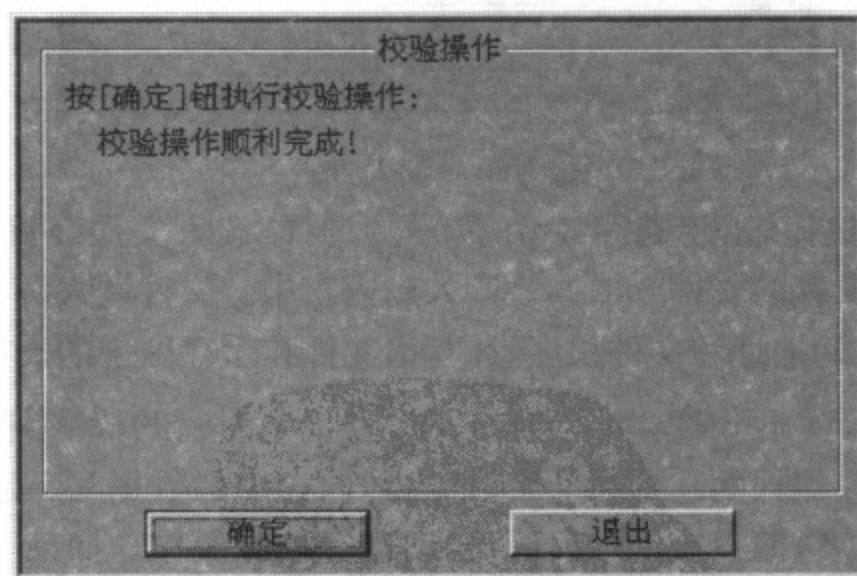


图 3-63 “校验操作”窗口

第四节 下载型实验板简介

近年来,随着 Flash 型单片机的广泛应用,采用软件模拟加写片验证成为一种经济实用的实验方法,尤其是随着单片机技术的发展,很多单片机都具有了 ISP 功能,只要一根下载线即可以编程。开始学习时,不再需要仿真机、编程器,使得单片机的入门门槛大为

降低。而美国 SST 公司推出的 SST 系列单片机更是集成了仿真功能,配合 Keil 软件,可使用用户的实验板直接具有仿真功能,将单片机的易用性推向一个新的高度。

图 3-64 是平凡单片机工作室(<http://www.mcustudio.com>)开发的一种实用下载型实验板。其电路原理图在本书所附光盘中。

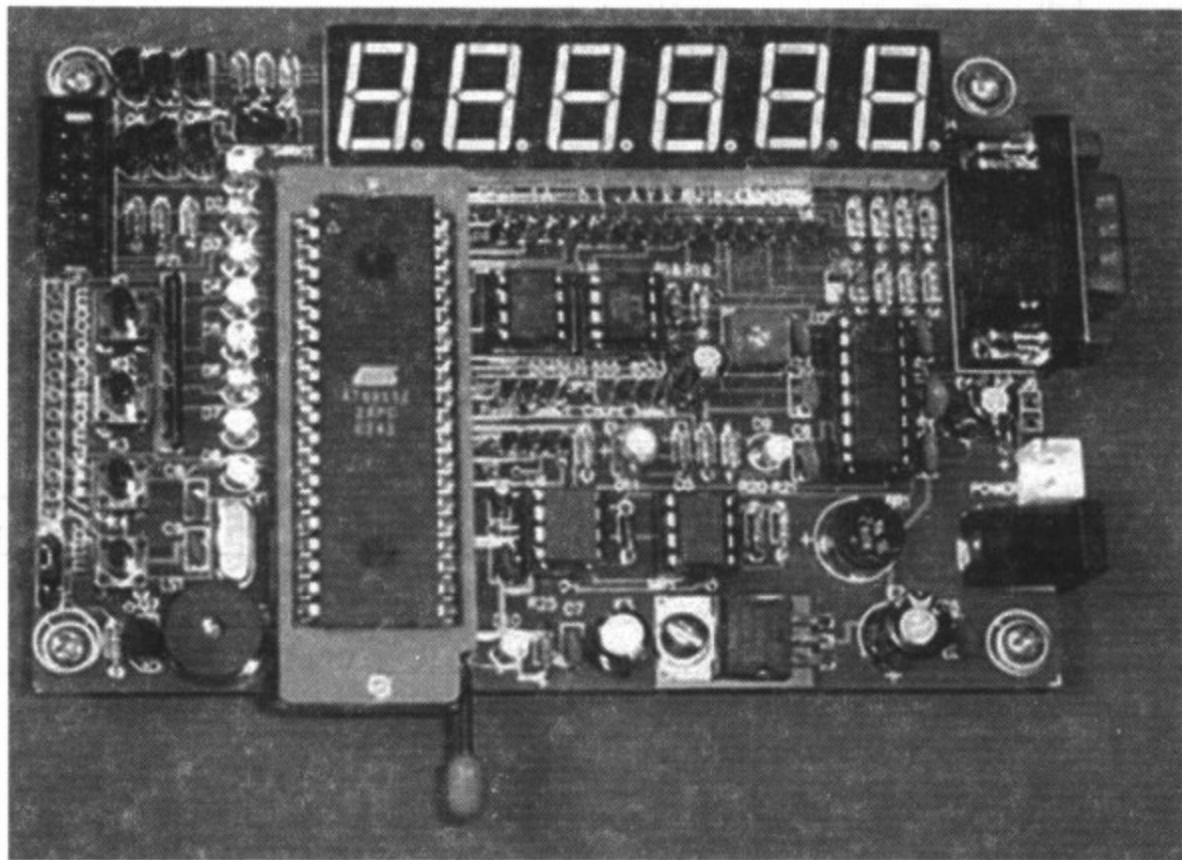


图 3-64 下载型实验板

实验电路板上安装了 6 位数码管、8 个发光二极管、4 个按钮开关、1 个简单的音响电路、1 个用于计数实验的振荡器、1 个 AT24CXXX 类芯片插座、1 个 X5045 芯片插座、1 个 RS232 串行接口、1 个 PCF8563 实时钟芯片插座、1 个字符型 LCD 插座,以及 1 个带有标准 ISP 插座,可用下载线对 AT89S5X 或 AT90S8515 单片机编程。本板还具有其他一些特点,如:既可使用 89 系列芯片学习 51 系列单片机,也可使用 AT90S8515 芯片学习 AVR 单片机;板上还设有扩展接口,可方便地扩展其他串行接口芯片。

使用这块实验板可以进行流水灯、人机界面程序设计、音响、中断、计数器等基本编程练习,还可以学习 I²C 接口芯片使用、SPI 接口芯片使用、字符型液晶接口技术、与 PC 机进行串行通信等目前较为流行的技术。

一、硬件结构

1. 发光二极管

单片机的 P1 端口接了 8 个发光二极管,这些发光二极管的负极接到 P1 端口各引脚,而正极则通过一个排电阻接到正电源端。发光二极管亮的条件是 P1 口相应的引脚为低电平,即:如果 P1 口某引脚输出为 0,则相应的灯亮;如果输出为 1,则相应的灯灭。

2. 数码管

单片机的 P0 口和 P2 口的部分引脚构成了 6 位 LED 数码管驱动电路。这里 LED 数码管采用了共阳型,使用 6 只 PNP 型三极管作为片选端的驱动。基极通过限流电阻分别接 P2.2~P2.7,集电极分别向 6 只数码管供电。图 3-65 是显示部分的电路原理图。

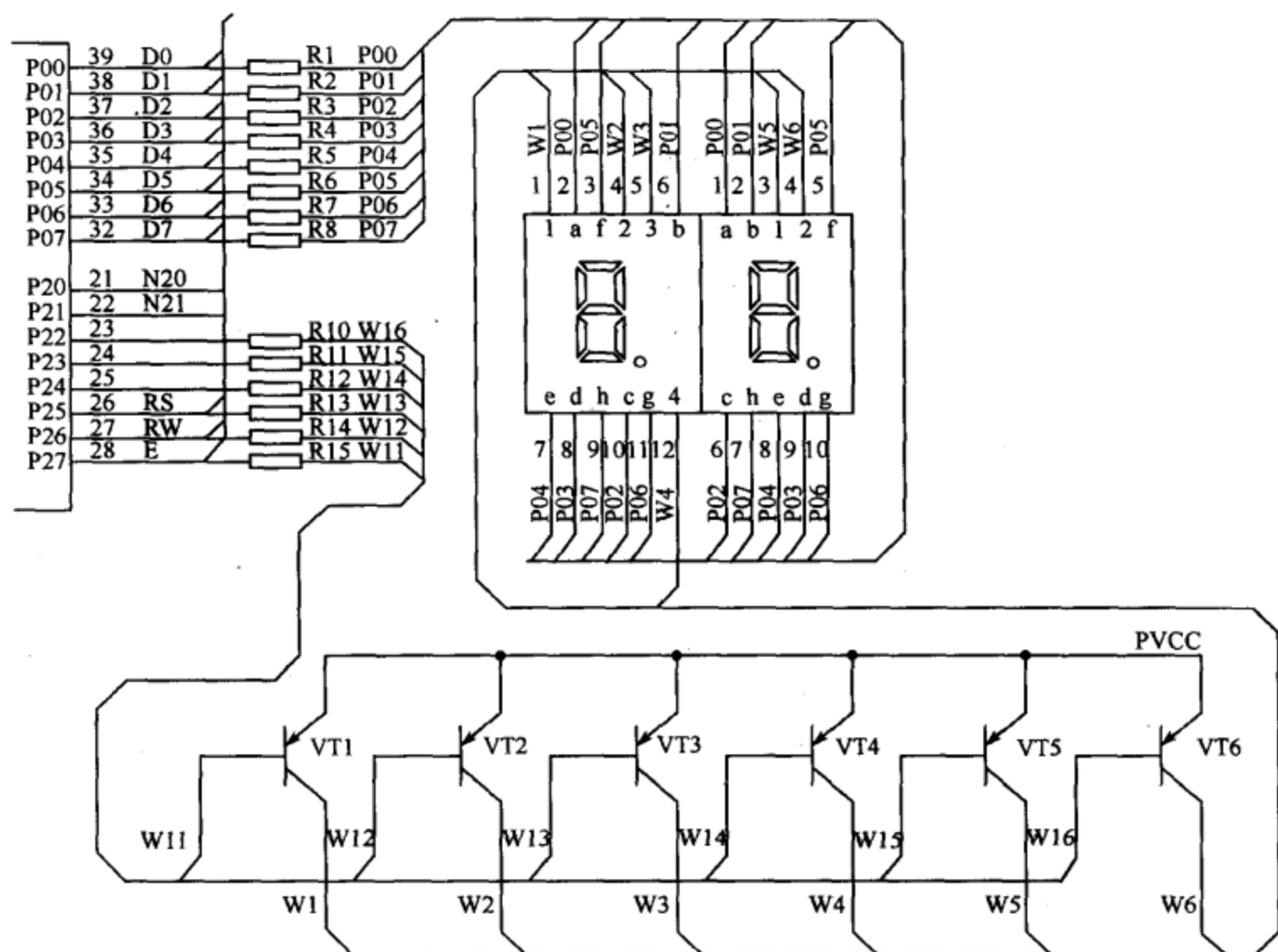


图 3-65 显示部分电路原理图

3. 串行接口

串行通信功能是目前单片机应用中经常要用到的功能,80C51 系列单片机 P3.0 和 P3.1 引脚的第二功能是串行口 RXD 与 TXD,其内部的串行接口电路具有全双工异步通信功能,但是单片机输出的信号是 TTL 电平,为获得电平匹配,实验板上扩充了一片 HIN232 芯片,利用该芯片进行电平转换,该芯片内部有电荷泵,只要单一的 5V 电源供电即可自行产生 RS232 所需的高电压,使用方便。

4. 按键输入

P3 口的 P3.2~P3.5 接 K1~K4 按钮开关,用做键盘。

5. 计数源

实验板有两路脉冲信号产生:其中一路由 555 集成电路及相关阻容元件构成典型的多谐振荡电路,输出方波,在输出端接有发光二极管,用于指示振荡器的输出;另一路由 PCF8563 集成电路提供。PCF8563 是实时钟芯片,通过编程可输出 1Hz、32Hz、1024Hz、32768Hz 的脉冲信号。在 PCF8563 的输出端接有发光二极管,用于指示振荡器的输出。通过 JP2 插针座可分别选择这两个脉冲信号中的一个作为单片机的计数信号。

6. 音响接口

P3.2 引脚经过三极管驱动一个无源蜂鸣器,构成一个简单的音响电路,如图 3-66 所示。由于 P3.2 同时作为按键输入使用,为了避免按键操作对发声电路的影响,使用 JP5 插针,只在需要时才用短路子将两个引脚连起来,这时 P3.2 作为输出口来使用。

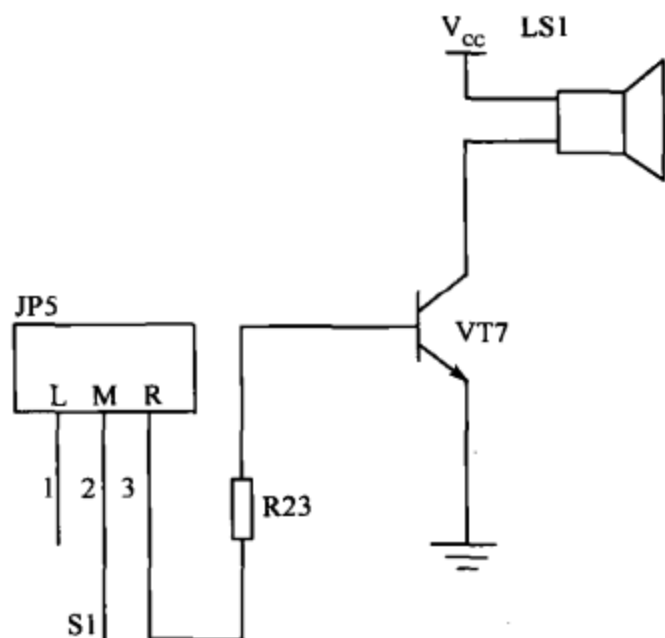


图 3-66 音响电路

7. AT24CXXX 芯片接口

在单片机应用系统中,经常会有一些数据需要长期保存,传统的方法是用 RAM 加后备电池的方法,但这种方法成本较高,电路也较复杂,近年来,多采用 EEPROM 来长期保存数据。与 RAM 相比,EEPROM 不能够无限次地擦除和写入数据,这是其缺点,但是,在断电之后,不需要特殊的供电方式,这是其优点。AT24CXXX 是 EEPROM 中的一种,仅 8 个引脚,采用 I²C 总线接口。为学习该芯片的使用,在实验板上设计有 I²C 接口电路,可进行芯片的读写实验。实验板中 P3.6 引脚接串行时钟线(SCL),P3.7 接串行数据线(SDA)。

8. 实时钟 PCF8563 芯片接口

PCF8563 是目前常用的低功耗的 CMOS 实时时钟/日历芯片。它提供一个可编程时钟输出、一个中断输出和掉电检测器,所有的地址和数据通过 I²C 总线接口串行传递。实验板上设计的 PCF8563 与 AT24CXXX 芯片共同挂接在 I²C 总线上,通过插针选择是否向单片机送出中断信号,通过插针选择是否将振荡信号送到单片机的计数端。

9. X5045 接口

X5045 是一片多功能的芯片,它具有以下的一些功能:上电复位、电压跌落检测、看门狗定时器、512B 的 EEPROM。该芯片采用三线制 SPI 接口方式与单片机相连,这也是目前一个应用比较广泛的芯片,通过学习这块芯片与单片机接口的方法,还可以了解和掌握三线制 SPI 总线接口的工作原理及一般编程方法。

硬件电路上,P2.1 接 X5045 的 CS 端,P3.7 接 X5045 的 SI 和 SO,P3.6 接 X5045 的 SCK,P2.0 接 X5045 的 WP 端。

10. 字符型液晶接口

液晶显示器由于体积小、质量轻、功耗低等优点,日渐成为各种便携式电子产品的理想显示器。从液晶显示器显示内容来分,可分为段式、字符式和点阵式 3 种。其中字符式液晶显示器以其价廉、显示内容丰富、美观、无需定制、使用方便等特点成为 LED 显示器的理想替代品。字符型液晶显示器专门用于显示数字、字母、图形符号并可显示少量自定义符号。这类显示器均把 LCD 控制器、点阵驱动器、字符存储器等做在一块板上,再与液

晶屏一起组成一个显示模块,因此,这类显示器安装与使用都较简单。

字符型液晶一般均采用 HD44780 及兼容芯片作为控制器,因此,其接口方式基本是标准的。实验板上带有 LCD 接口,可直接与字符型液晶相连。实验板上数据线被连到 P0 口,P2.5 接 RS 端,P2.6 接 RW 端,P2.7 接 E 端。

11. 扩展接口

标号为 J1 的接口为 12 芯插座,是扩展接口。该接口将 P1 口、T1 计数器端、INT1 外中断接口端及电源端引出,供扩展板使用。

二、使用简介

1. 电源提供

实验板需外接电源。可以通过插座 J5 向实验板供电,要求输出的直流电压在 9V~15V,电流不小于 300 mA。由于板上装有整流全桥,因此不必考虑电源的输出极性,直接将插头插入其中即可。

2. 复位选择

实验板提供了两种复位电路,即 RC 复位与外接芯片复位。JP1 用于复位选择,在该插针座下标有 Reset Select 字样,很容易辨认。在该插针座上方左侧标有 RC 字样,右侧标有 X5045(3)字样。如果用短路子插于左侧,则选择 RC 复位电路,避免刚开始对 X5045 芯片不熟悉而影响学习;如果用短路子插于右侧,则选择 X5045 复位,可用于测试 X5045 芯片的“看门狗”特性。不论短路子插于哪一侧,X5045 芯片内部的 EEPROM 存储器总是可用的。

注意 在使用 ISP 在线可编程功能时,必须将短路子拔除,既不选择 RC 复位,也不选择 X5045 复位,由下载线控制复位端。

3. 计数源选择

实验板提供了两个计数源,可供单片机做计数实验。第 1 个计数源由 555 电路提供,第 2 个计数源由 PCF8563 提供,通过 JP2 选择。JP2 的下方标有 Count Select 字样,上方左侧标有 555,右侧标有 8563,分别代表选择这两个计数源。

注意 当 P3.3 作为输入端使用时,应将短路子取下,不接于任何一方。

当使用 PCF8563 作为时钟源时,需对 PCF8563 芯片编程。通过编程,可提供 1Hz、32Hz、1024Hz 和 32768Hz 等多个标准信号源。

4. 音响电路工作选择

JP5 用于选择 P3.2 究竟工作于输出方式还是输入方式。当需要将 P3.2 作为驱动音响电路工作的输出端时,应将短路子插于 JP5 的下方(插座旁印有“↓”标志);反之应将短路子插于 JP5 的上方。

5. 字符型 LCD 实验

进行 LCD 实验时,需断开数码管的供电电路。JP4 用于选择显示器。在 JP4 的下方标有 Disp Select 字样,上方分别标有 LED 和 LCD 字样。将短路子插于 LED 一方,选择 LED 作为显示器;插于 LCD 一方,则选择 LCD 作为显示器。

本板提供了供 LCD 使用的 16 针标准接线插座,标号为 J3。在 J3 下方注有 LCD 字样;在 J3 的上方用数字写出了 1~16 的字样,标出了接线的序号。安装时应注意与液晶

显示模块上的引脚序号对应。下方的蓝色电位器用于调整 LCD 的对比度。

6. ISP 功能的使用

拔去复位插座上的短路子,使复位端悬空。标号为 J1 的插座为 ISP 下载插座,将下载电缆与实验板正确连接。本插座采用标准接法,具体接法如表 3-1 所列。

表 3-1 ISO 插座接线

标 号	名 称	功 能
1	SCK	串行时钟
3	MISO	主器件输入—从器件输出
4	VCC	电源
5	RST	复位端
• 9	MOSI	主器件输出—从器件输入
2,10	GND	地
6,7,8	NC	未接

7. 仿真功能的使用

使用实验板可以用 MON51 仿真器进行仿真实验。关于仿真器的使用在前面已作了详细介绍。

三、下载型编程器的使用

ISP 指电路板上的空白器件可以编程写入最终用户代码,而不需要从电路板上取下器件,已经编程的器件也可以用 ISP 方式擦除或再编程。ISP 技术是未来发展方向。

ISPro 编程器全称 ISPro 下载型编程器,ISPro 编程器是具备下载线外型和使用方式的通用编程器,ISPro 编程器套件如图 3-67 所示。

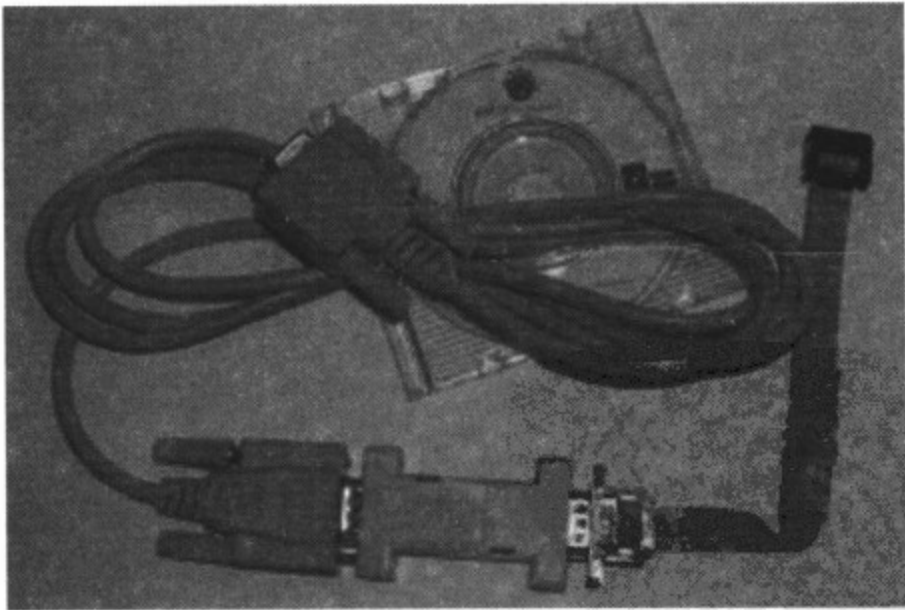


图 3-67 ISPro 套件

购买 ISPro 编程器时,会随机赠送安装光盘,安装时,将插光盘插入光驱,运行 setup.exe 即可。如果提醒系统文件过旧则需要重新启动计算机。重启后继续安装。安装后,在桌面上建立一个“ISPro.exe 下载型编程器”图标,双击该图标,即可启动编程软件。启动后的画面如图 3-68 所示。

该软件和 RF-810 软件的使用方法基本相同,这里不再具体介绍。

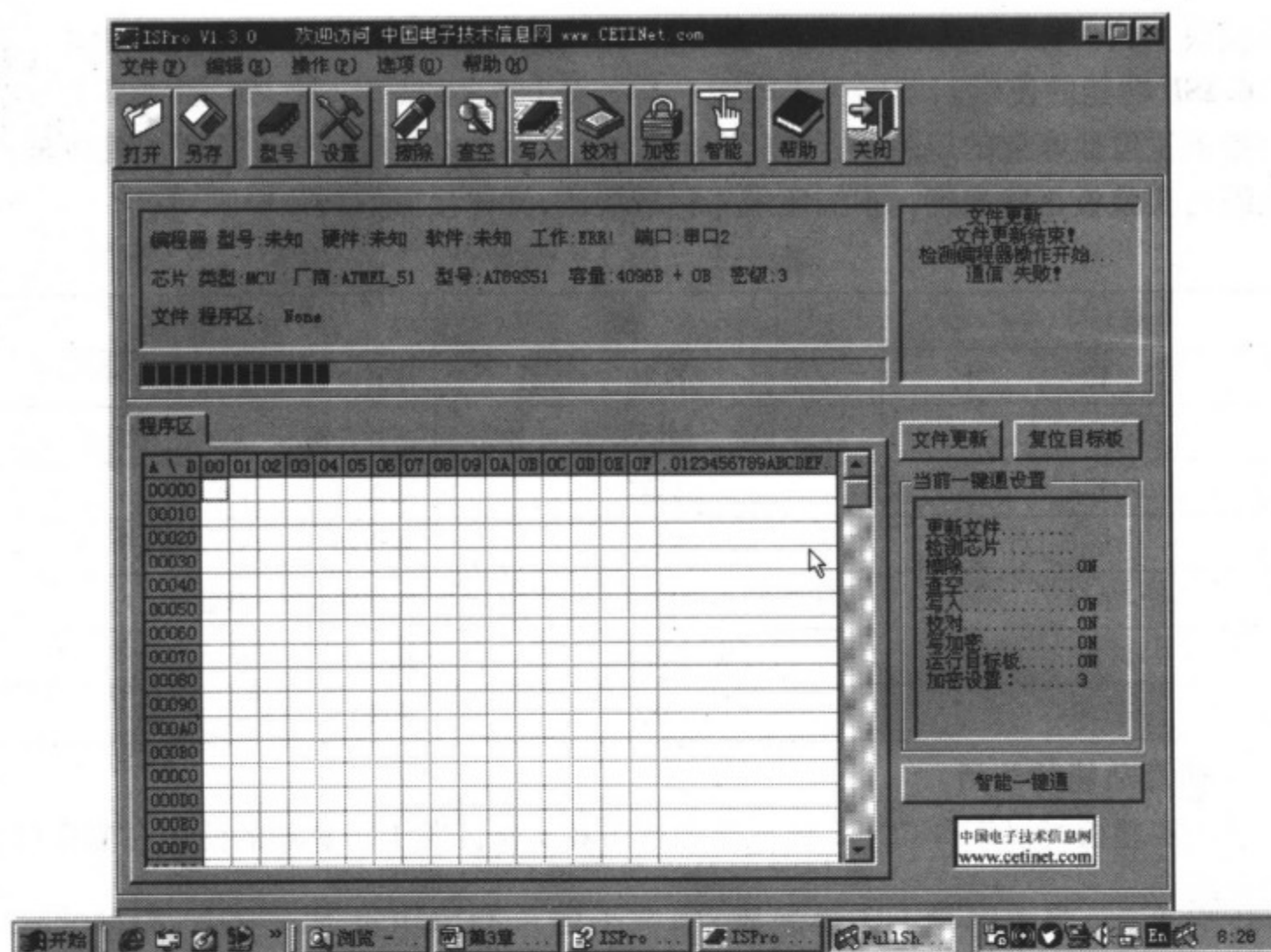


图 3-68 启动后的画面

第四章 单片机的指令系统

一台单片机系统如果只有硬件还是不能工作,必须有相应的软件才能发挥其运算、控制功能。而软件中最基础的东西,就是计算机的指令系统。这一章中,将先对指令系统作一般介绍,然后再详细介绍 MCS-51 常用指令的使用,最后,对单片机 I/O 访问指令的使用和指令的记忆方法作简要阐述。

第一节 指令概述

一、指令与指令系统

1. 指令

在第二章已经讲述了单片机的几个主要组成部分,它们构成了单片机的硬件。所谓硬件(Hardware),就是看得到、摸得着的实体。但是,光有这样的硬件,还只是有了实现计算和控制功能的可能性。单片机要真正地能进行计算和控制,还必须有软件(Software)的配合。软件主要指的是各种程序。只有将各种正确的程序烧录到单片机,它才能有效地工作。单片机所以能自动地进行运算和控制,正是由于人把实现计算和控制的步骤一步步地用命令的形式,即一条条指令(Instruction)预先存入到单片机的片内或片外程序存储器中,单片机在 CPU 的控制下,将指令一条条地取出来,并加以翻译和执行。就以两个数相加这一简单的运算来说,当需要运算的数已存入存储器后,还需要进行以下步骤:

第 1 步,把第 1 个数从它的存储单元(Location)中取出来,送至运算器;

第 2 步,把第 2 个数从它所在的存储单元中取出来,送至运算器;

第 3 步,相加;

第 4 步,把相加完的结果,送至存储器中指定的单元。

所有这些取数、送数、相加、存数等都是一种操作(Operation),把要求计算机执行的各种操作用命令的形式写下来,这就是指令。因为单片机是一种可编程器件,只“认得”二进制码(0,1)。要单片机运作,单片机系统中的所有指令,都必须以二进制编码的形式来表示。例如,要做 10 加 20 的加法,在 MCS-51 中,可用这样的指令:

01110100 ;把 10 放在累加器 A 中

00001010

00100100 ;A 加 20,结果仍在 A 中

00010100

如果用十六进制数表示指令码。此时,以上两条指令可写为

740AH

2414H

符号 H 有时也可省略。

这种由一连串的 0 和 1 组成的代码称为机器码(或指令码),机器码没有明显的特征,不好记忆,不易理解,易出错。所以,常采用指令助记符,即一条指令用一组有一定意义的英文缩写字符来表示,这样就更便于理解和记忆了。如上面两条指令可写为

(740A) MOV A, #0AH ;把 10 放在累加器 A 中

(2414) ADD A, #14H ;A 加 20,结果仍在 A 中

尽管采用助记符后,书写的字符增多了,但由于可读性增强了,使用时反而感到方便。但是,用助记符编写的程序虽然便于人理解,可单片机却只认识二进制机器代码,因此,为了让单片机能“读懂”汇编语言程序,必须再转换成由二进制机器码构成的程序,这种转换过程,就称为“汇编”。汇编可借助于人工查表法来实现,也可由一种称为汇编程序的软件(如 Keil)进行翻译,经过翻译后就可以变成机器可执行的目标程序了。

2. 指令系统

一条指令,对应着一种基本操作;单片机所能执行的全部指令,就是该单片机的指令系统(Instruction Set)。不同种类的单片机,其指令系统亦不同,例如,MCS-51 单片机指令系统共有 111 条指令,而 PIC 单片机的指令系统则只有 30 多条指令。

二、指令的格式

指令通常分为操作码(Opcode)和操作数(Operand)两大部分。操作码表示计算机执行什么操作,即指令的功能;操作数表示参加操作的数或操作数所在的地址(即操作数所存放的地方编号)。例如,以下指令:

ADD A, #14H

这条指令表示把累加器 A 中的内容(设为 #0AH)和立即数 14H,通过算术逻辑单元相加,并将结果保留在 A 中。这里,ADD 称为操作码,而 A、#14H 等均称为操作数。

在汇编语言程序中,操作码通常由英文单词缩写而成,这样有助于记忆,所以又称助记符。如 MOV 就是英文单词 MOVE 的缩写,含有“搬移”的意思;而 ADD 即为英文单词,其意为“相加”。因此,对于略懂英语的用户,掌握单片机指令的含意是较为方便的。

操作数可以直接是一个数,如以上的 #14H 称为立即数,即 14H 就是真正的操作数。操作数也可以是一个数据所在的地址,即并不直接在指令中表明所操作的数据,而是指出数据所存放的位置。在执行指令时,再从指定的地址中取出操作数。另外,操作数还可以是寄存器、间接地址等,在下面介绍寻址方式时还要具体介绍。

三、指令的字节数

操作码和操作数都是二进制代码,8 位二进制数为一字节(1B),指令由指令字节组成。对于不同的指令,指令的字节数是不同的。在 MCS-51 系统中,可以是单字节、双字节或三字节指令。

1. 单字节指令

单字节或一字节(1B)指令中既包含操作码的信息,也包含操作数的信息。有两种情况。一种是指令的含义和对象都很明确,不必再用另一个字节来表示操作数。例如,数据

指针 DPTR 内容加 1 指令 INC DPTR, 由于操作的内容和对象都很明确, 故不必再加操作数字节, 其指令码为

1010	0011
------	------

另一种情况是, 一个字节中的几位来表示操作数或操作数所在的位置。例如, 从工作寄存器向累加器 A 传送数据的指令为 MOV A, R_n, 其中 R_n 可以是 8 个工作寄存器中的一个, 故在指令码中分出 3 位来表示这 8 个工作寄存器, 用其余各位表示操作码的作用, 指令码为

1110	1rrr
------	------

其中最低 3 位码就用来表示从哪一个寄存器取数, 故一字节也就够了。

MCS-51 系统中共有 49 条单字节指令。

2. 双字节指令

双字节或两字节(2B)指令一般是用一个字节表示操作码, 另一个字节表示操作数或者操作数的地址。这时, 操作数或操作数地址就是一个 8 位二进制数, 因此, 必须专门用一个字节来表示。

例如, 立即数与累加器 A 的内容与操作指令: ANL A, #data, 其中用 #data 表示 8 位二进制数, 也称立即数, 这条指令就是双字节指令, 其指令码为

0101	0100
立即数	

双字节指令的第 2 个字节, 也可以是操作数所在的地址。MCS-51 系统中有 45 条双字节指令。

3. 三字节指令

三字节(3B)指令则是一个字节的操作码, 两个字节的操作数。操作数可以是数据, 也可以是地址。例如, 逻辑操作指令 ANL direct, #data, 直接地址单元内容与立即数进行“与”操作, 指令码为

0101	0011
直接地址	
立即数	

MCS-51 系统中有 17 条三字节指令。

方法技巧 如何估计指令的字节数。

若指令中既不包含直接地址, 又不包含立即数的, 为单字节指令; 若指令中包含直接地址(direct)或立即数(#data)的, 为双字节指令; 若指令中既包含直接地址又包含立即数, 为三字节指令。如:

MOV A, @R0 为一字节指令, 指令码为

1110	0110
------	------

MOV A, #data 为双字节指令, 指令码为

0111	0100
立即数	

MOV direct, #data 为三字节指令,指令码为

0111 0101
直接地址
立即数

四、指令的寻址方式

寻址就是寻找操作数的地址。在用高级语言编程时,编程者不必关心参与运算数据(操作数)的存放问题,也不必关心这些运算是在哪里(哪个寄存器)完成的。例如,对于以下的语句:

```
x=10;  
y=20;  
z=x+y;
```

编程者只需关心语句的使用是否正确,结果是否正确。至于变量 x 和 y 的值存放在何处,则根本不必关心。但在汇编语言编程时,数据的存放、传送、运算都要通过指令来完成,编程者必须自始至终都要十分清楚操作数的位置以及如何将它们传送至适当的寄存器去运算。因此,如何从各个存放操作数的区域去寻找和提取操作数就变得十分重要。所谓寻址方式就是如何通过确定操作数所在的位置(地址),把操作数提取出来的方法。例如指令:

ADD A,70H

表示把累加器 A 中的内容(设为 20H)和存储器中地址为 70H 单元中的内容相加,并将结果保留在 A 中。注意,70H 是存储器中某个单元的地址,在该单元中,存放着操作数(比如说是 3AH)。“ADD A,70H”不是将 70H 和 A 中的内容相加,而是从存储器 70H 单元中将 3AH 取出和 A 中的内容相加。可见,要找到实际操作数,有时就要转个弯,甚至转几个弯,这个过程就是寻址,寻址是汇编语言程序设计中最基本的内容之一,必须十分熟悉,牢固掌握。

寻址有如找人,如被找的人有手机、小灵通、固定电话等多种联系方式,则就容易找到他。单片机也是如此,寻址方式越多,找操作数越方便,单片机的功能就越强。MCS-51 共有 7 种寻址方式,现介绍如下。

1. 寄存器寻址

寄存器寻址就是以通用寄存器的内容作为操作数,在指令的助记符中直接以寄存器名字来表示操作数位置。

在 MCS-51 的 CPU 中,并没有专门的硬件通用寄存器,而是把内部数据 RAM 中的一部分(00~1FH)作为工作寄存器来使用;每次可以使用其中的一组,并以 R0~R7 来命名。仿佛它们就是专门的通用寄存器似的。

说明 有时需要对 PSW 中的 RS1、RS0 位的状态设置,来进行当前寄存器组的选择。

在 MCS-51 指令中,若操作数是以 R0~R7 来表示操作数时,就属于寄存器寻址方式,例如指令:

MOV A,R0

其功能是把寄存器 R0 的内容传送到累加器 A 中,由于操作数在 R0 中,因此,在指令中指定了 R0,就能从中取得操作数,所以是寄存器寻址方式。

说明 部分专用寄存器,如累加器 A、AB 寄存器对以及数据指针 DPTR 表示操作数地址的指令也属于寄存器寻址的范围。

2. 直接寻址

在指令中直接给出操作数地址,就属于直接寻址方式。此时,指令的操作数部分是操作数的地址。例如指令:

MOV A,3AH

就属于直接寻址。其中 3AH 就是表示直接地址,即内部 RAM 的 3A 单元。若 3AH 单元中存放的内容是 58H,则该指令的功能就是把内部 RAM 的 3AH 单元内容 58H 传送到累加器 A 中。

直接寻址的操作数在指令中以存储单元形式出现,因为直接寻址方式只能使用 8 位二进制数表示的地址,因此这种寻址方式的寻址范围只限于内部 RAM,具体说就是:

(1)低 128 单元。在指令中直接以单元地址形式给出。

(2)专用寄存器。专用寄存器除以单元地址形式给出外,还可以以寄存器符号形式给出。例如以下两条指令:

MOV A,TH0

MOV A,8CH

都是把特殊功能寄存器 TH0(定时器 0 的高 8 位寄存器)内容送累加器 A,这里,8CH 是 TH0 寄存器的 RAM 地址。因此,这两种表示的作用是等价的,译成机器码也是相同的。一般编程者都愿意写成第 1 种形式,以提高程序的可读性。

3. 立即寻址

若指令的操作数是一个 8 位二进制数或 16 位二进制数,就称为立即寻址。指令中出现的操作数就称为立即数。

由于 8 位立即数和直接地址都是 8 位二进制数(两位 16 进制数),因此,在书写形式上必须有所区别。在 MCS-51 系统中,采用“#”来表示后面的是立即数而不是直接地址。如 #3AH 是表示立即数 3AH,而直接写 3AH 则表示 RAM 地址 3AH 单元。要注意能区分这两种不同的表示方法及其含义。例如以下两条指令:

MOV A,#3AH

MOV A,3AH

前一条指令为立即寻址,表示把立即数 3AH 送到累加器 A 中。执行后,A=3AH;后一条指令为直接寻址,表示把 3AH 单元中的内容送到累加器 A 中,执行后,A 为 3AH 单元的内容。

在 MCS-51 系统中,8 位立即数表示为 #data,这种带有 8 位立即数的指令,一般都是双字节的指令,即一个字节为操作码,一个字节为 8 位立即数。

在 MCS-51 系统中,只有一条 16 位立即数的指令,即:

MOV DPTR,#data16

其功能是将立即数高 8 位送 DPH,立即数的低 8 位送 DPL。由于是 16 位立即数,因此是一条三字节指令,即 1B 的指令码,2B 的立即数。

4. 寄存器间接寻址

寄存器间接寻址方式是指寄存器中存放的是操作数的地址,即操作数是通过寄存器间接得到的,因此称为寄存器间接寻址。寄存器间接寻址也以寄存器符号的形式给出,为了区别寄存器寻址和寄存器间接寻址,在寄存器间接寻址中,寄存器的前面加@以示区别。

寄存器间接寻址的寻址范围如下:

(1)内部 RAM 的低 128 单元。此时,应使用 R0 或 R1 做间址寄存器。例如,MOV A,@R0 就是寄存器间接寻址,其功能把 R0 寄存器的内容(设为 20H)作为地址,把该地址单元的内容(设为 25H)送到累加器 A 中,功能示意图如图 4-1 所示。

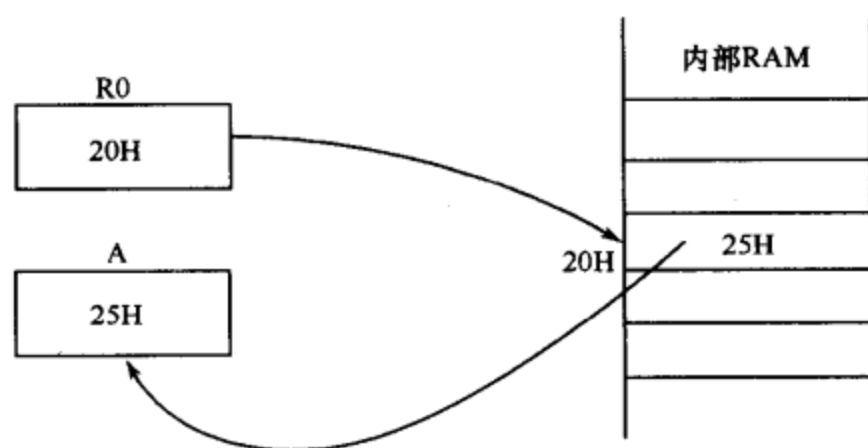


图 4-1 寄存器间接寻址功能示意图

(2)外部 64KB 的 RAM,此时,应使用 DPTR 作为间址寄存器。例如,MOVX A,@DPTR 就是寄存器间接寻址,即把 DPTR 指定的外部 RAM 单元的内容送到累加器 A。

(3)外部 RAM 的低 256 单元。该单元是一个特殊的寻址区,除可以使用 DPTR 作间址寄存器寻址外,还可以使用 R0 或 R1 作间址寄存器寻址。例如,MOVX A,@R0,即把 R0 指定的外部 RAM 单元的内容送到累加器 A。

(4)后面将要介绍的堆栈操作指令(PUSH、POP)也应算做寄存器间接寻址,即以堆栈指针 SP 作为间址寄存器的寻址方式。

可能有人会问,在指令中直接给出实际操作数,不是简单、明了吗?为什么还要用这种复杂的寻址方式呢?下面从一个问题谈起:某程序要求从片内 RAM 的 30H 单元开始,取 20 个数,分别送入 A 累加器。也就是从 30H,31H,32H,33H,...,44H 单元中取出数据,依次送入 A 中。就目前掌握的方法而言,要从 30H 单元取数,就用指令 MOV A,30H;下一个数在 31H 单元中,只能用 MOV A,31H。因此,取 20 个数,就要用 20 条指令才能完成。这个例子中只有 20 个数。如果要送 200 个数,就要写上 200 条指令。用这种方法未免太笨了,所以应当避免用这样的方法。如果采用寄存器间接寻址,就可以方便地解决这一类问题。

以下是解决上面问题的程序:

```
MOV R7, #10
MOV R0, #30H
LOOP: MOV A,@R0
      INC R0
      DJNZ R7, LOOP
```

这个例子中:

第1条指令是将立即数20送到R7中,执行完该指令后,R7中的值应当是20。

第2条指令是将立即数30H送入R0中,执行完该指令后,R0单元中的值是30H。

第3条指令是应用寄存器间接寻址的方式写的一条指令。其用途是取出R0单元中的值,把这个值作为地址,取这个地址单元的内容送入A中。第1次执行这条指令时,工作寄存器R0中的值是30H,因此执行这条指令的结果就相当于执行MOV A,30H。

第4条指令加1指令,即把R0中的值加1。这条指令执行完后,R0中的值变成31H。

第5条指令是减1条件转移指令,将R7中的值减1,然后判断该值是否等于0。如果R7的值为0,则程序顺序执行;如果不等于0,则转到标号LOOP处继续执行。由于R7中的值是20,减1后是19而不等于0。因此,执行完这条指令后,将转去执行标号为LOOP的第3条指令,就相当于执行MOV A,31H;此时R0中的值已是31H了,对R7中的值再次减1,并判断是否等于0。若不等于0,又转去执行第3条指令……如此不断循环,直到R7中的值经过逐次相减后等于0为止。也就是说,第3条~第5条指令一共被执行20次,实现了上述要求:将从30H单元开始的20个数据送入A中。这样,仅用了5条指令,就替代了20行的程序。这里,R0是用来存放“有数据的内存单元的地址”的,称之为“间址寄存器”。

说明 在寄存器间址寻址方式中,只能用R0和R1作为间址寄存器。

5. 变址寻址

变址寻址是以某个寄存器的内容为基本地址,然后在这个基本地址基础上加上地址偏移量才是真正的操作数地址,并将这个地址单元的内容作为指令的操作数。

在MCS-51系统中,没有专门的变址寄存器,而是采用数据指针DPTR或者程序计数器PC的内容为基本地址,而地址偏移量则是累加器A的内容,并以DPTR+A或者PC+A的值作为实际的操作数地址。

变址寻址方式是一种专门用于片内和片外程序存储器的寻址方式。用变址寻址对外部程序存储器的内容进行访问时,访问的范围可为64kB,当然,这种访问只能是从ROM中读数据,而不可能对ROM写入。例如以下指令:

MOVC A,@A+DPTR

指令执行前,设A=54H,DPTR=3F21H,故操作数地址为3F21H+54H=3F75H,指令执行后,将程序存储器3F75H单元的内容(设为7FH)传送到累加器A,故指令执行后,A=7FH,其余均不变。其功能示意图如图4-2所示。

重点提示 尽管用DPTR作为基址寄存器,但变址寻址的区域却是程序存储器ROM,而不是数据存储器RAM;另外,虽然变址寻址指令的助记符和指令操作都比较复杂,但却只是一字节指令。在有的指令系统中,变址寻址指令的字节数很多,这也可以说是MCS-51指令系统的一个优点。

6. 位寻址

MCS-51有位处理功能,可以对数据位进行操作,因此就有相应的位寻址方式。采用位寻址方式的指令,其操作数将是8位二进制数中的某一位。在指令中则给出位地址,即给出是哪个内部RAM单元的哪一位。位地址在指令中用bit表示。

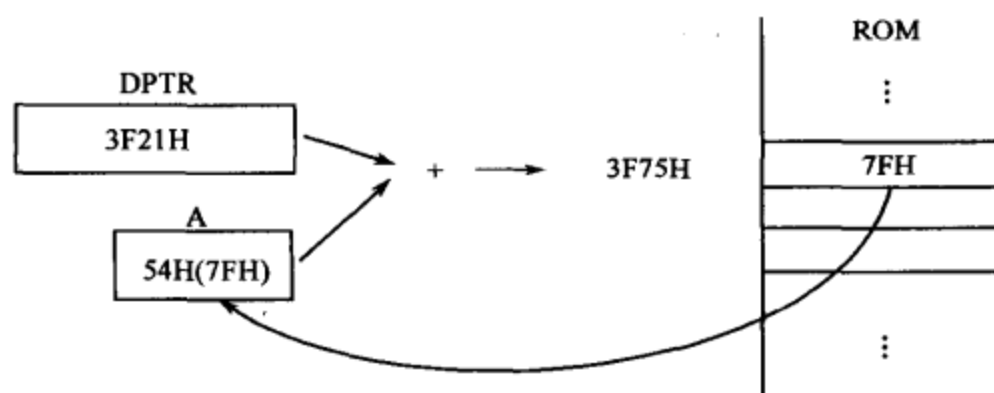


图 4-2 变址寻址功能示意图

MCS-51 系统的内部数据 RAM 有两个可以按位寻址的区域：

(1)内部 RAM 中的位寻址区。内部 RAM 的 20H~2FH 共 16 个单元中的每一位都可单独作为位操作数，共 128 位。位寻址区中的位有两种表示方法：一是直接用位地址 00H~FFH 来表示。对于 20H~2FH 共 16 个单元的 128 位的位地址即为 00H~7FH，例如，20H 单元的 0~7 位的位地址为 00H~07H。二是采用第几单元第几位的表示方法。如 25H. 5 表示 25H 单元的 D5 位。这种表示法可以避免查表或计算，比较方便。

(2)特殊功能寄存器的可寻址位。可供位寻址的特殊功能寄存器共有 11 个，其特征是这些特殊功能寄存器的单元地址能被 8 整除。表 4-1 中列出了这些特殊功能寄存器及其位地址。表中的位地址是不连续的，有些没有列出的位地址(如 C0H~C7H)是没有意义的；若在指令中加以使用，会得出不确定的结果。

表 4-1 可以位寻址的特殊功能寄存器位地址表

寄存器	单元地址	位地址表示	位地址
P0	80H	P0. 0~P0. 7	80H~87H
TCON	88H	TCON. 0~TCON. 7	88H~8FH
P1	90H	P1. 0~P1. 7	90H~97H
SCON	98H	SCON. 0~SCON. 7	98H~9FH
P2	A0H	P2. 0~P2. 7	A0H~A7H
IE	A8H	IE. 0~IE. 7	A8H~AFH
P3	B0H	P3. 0~P3. 7	B0H~B7H
IP	B8H	IP. 0~IP. 7	B8H~BFH
PSW	D0H	PSW. 0~PSW. 7	D0H~D7H
ACC	E0H	ACC. 0~ACC. 7	E0H~E7H
B	F0H	B0. 0~B0. 7	F0H~F7H

特殊功能寄存器的寻址位在指令中有以下 4 种表示方法：一是直接用位地址。例如，PSW 寄存器位 5 地址为 0D5H。二是单元地址加位的表示法。例如，0D0H 单元(PSW 寄存器)位 5 表示为 0D0H. 5。三是位名称表示法。例如，PSW 的位 5 是 F0 标志位，则可使用 F0 表示该位。四是专用寄存器加位的表示方法。例如，PSW 寄存器的位 5 表示为 PSW. 5。

另外，位地址还可以用伪指令进行定义，将在下一章进行介绍。

7. 相对寻址

前面讲述的 6 种寻址方式主要是解决操作数的给出问题，而此处的相对寻址方式则是为解决程序转移而专门设置的。在高级语言中有“GO TO”或者“GO SUB”之类的语句

完成转移功能。在单片机的指令系统中,也设有转移指令。转移指令分为直接转移指令和相对转移指令。在相对转移指令中,采用相对寻址方式,此时指令的操作数部分给出的是地址的相对偏移量,在 MCS-51 中常以 rel 表示。把程序计数器 PC 的当前值加上偏移量就构成了程序转移的目的地址。但这里的 PC 当前值是指执行完该转移指令后的 PC 值,即转移指令的 PC 值加上它的字节数。因此转移的目的地址可用如下公式表示:

目的地址=转移指令地址+转移指令字节数+rel

偏移量 rel 是一个带符号的 8 位二进制补码数。所能表示的数的范围是 -128~+127,因此相对转移是以转移指令所在地址为基点,向前(地址增加方向)最大可转移(127+转移指令字节数)个单元地址,向后(地址减少方向)最大可转移(128-转移指令字节数)个单元地址。

目前,在单片机的程序设计中,一般采用机器汇编,通常用标号来表示目标位置,基本上不需要人工计算相对寻址的值,因此,读者不必对相对寻址有过深的研究。

重点提示 在两个操作数的指令中,把左边的操作数称为目的操作数,把右边的操作数称为源操作数,我们所讲的各种寻址方式都是针对源操作数的,但实际上,源操作数和目的操作数都有寻址的问题,例如指令:

MOV 45H,R1

其源操作数是寄存器寻址方式,而目的操作数则是直接寻址方式,因此指令的功能是把按寄存器寻址取出的 R1 内容再以直接寻址方式存放于内部 RAM 的 45H 单元中。

由于 MCS-51 中目的操作数的寻址方式较少,只有寄存器寻址、直接寻址、寄存器间接寻址和位寻址 4 种方式。因此知道了源操作数的寻址方式也就不难了解目的操作数的寻址问题了。

归纳总结

- (1)片内和片外 ROM 只能使用变址寻址方式。
- (2)内部 RAM 由于使用频繁,寻址方式较多,如表 4-2 所列。

表 4-2 内部 RAM 的寻址方式

寻址方式	高 128 单元 (特殊功能寄存器)	低 128 单元		
		用户 RAM 区	位寻址区	通用寄存器区
	(0FFH~80H)	(7FH~30H)	(2FH~20H)	(1FH~00H)
寄存器寻址方式	—	—	—	全部
位寻址方式	部分(11 个)	—	全部	—
间接寻址方式	—	全部		
直接寻址方式	全部			

- (3)外部 RAM 只能使用寄存器间接寻址方式,如表 4-3 所列。

表 4-3 外部 RAM 寻址

寄存器间接寻址	高地址单元(65280 个) 0FFFFH~0101H	低地址单元(256 个) 0100H~0000H
以 R0 或 R1 作为间址寄存器	—	全部
以 DPTR 作为间址寄存器	全部	

(4)立即数寻址方式只涉及 8 位数或 16 位数,由于数据已在指令中给出,所以没有在表 4-3 中列出。

第二节 MCS-51 单片机指令分类介绍

MCS-51 指令系统可以分为五大类,即数据传送类指令(28 条)、算术运算类指令(24 条)、逻辑运算及移位类指令(25 条)、控制转移类指令(17 条)和位操作(布尔操作)类指令(17 条),总共 111 条。

初学者在阅读各指令表时,因注释简单常感困难。阅读指令表时必须懂得各种符号代表的具体内容才能达到使用目的。指令表中符号的定义十分严格。初学者必须牢记。MCS-51 指令中采用了以下的符号:

Rn——表示工作寄存器,Rn 可以为 R0~R7。

#data——表示 8 位立即数,即包含在指令中 8 位常数。

#data16——表示 16 位立即数,即包含在指令中 16 位常数。

direct——表示 8 位直接地址,具体来说,就是代表内部 RAM 的 128 个单元(00H~7FH)以及所有的特殊功能寄存器。对于特殊功能寄存器可直接用其名称来代替其直接地址。

@——间址寄存器的前缀标志。

@Ri——表示寄存器间接寻址,Ri 可以为 R0 或 R1。

@DPTR——以 DPTR 内容(16 位)为地址的间接寻址,用来对外部 64Kb 个 RAM 寻址。

addr16——16 位目的地址,只限于在 LCALL 和 LJMP 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。

bit——内部 RAM(包括专用寄存器)中的直接寻址位。

A——累加器。

ACC——直接寻址方式的累加器。

B——寄存器 B。

C——进位标志位,它是布尔处理机的累加器,也称之为累加位。

/——加在位地址的前面,表示对该位状态取反。

(X)——某寄存器或某单元的内容。

((X))——由 X 间接寻址的单元中的内容。

←——箭头左边的内容被箭头右边的内容所取代。

一、数据传送类指令(28 条)

数据传送操作属复制性质,而不是剪切性质。一般传送类指令的助记符为“MOV”,通用格式为:MOV <目的操作数>,<源操作数>

传送指令中,右边操作数为源操作数,表达的是数据的来源;左边操作数为目的操作数,表达的则是数据的去向。源操作数可以是累加器 A、通用寄存器 Rn、直接地址 direct、间址寄存器和立即数。而目的操作数可以是累加器 A、通用寄存器 Rn、直接地址 di-

rect 和间址寄存器。两者只差一个立即数。

1. 内部 RAM 之间的数据传送指令

内部数据 RAM 区是数据传送最活跃的区域,可用的指令数也最多。按源操作数的寻址方式可以分为 4 组。

1) 立即寻址

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

这组指令表明,8 位立即数可以直接传送到内部数据 RAM 的各个位置,包括内部 RAM 的 80H~FFH 单元。

有些指令的功能是相同的,但指令字节数不同。例如:

```
MOV A, #data      (2B)
MOV ACC, #data    (3B)
```

前一条指令中 A 是累加器,后一条指令中 ACC 表示累加器的直接地址 E0H,但功能是不同的。初学者应注意 A 和 ACC 的不同。

2) 直接寻址

```
MOV A, direct      ; A ← (direct)
MOV Rn, direct      ; Rn ← (direct)
MOV @Ri, direct     ; (Ri) ← (direct)
MOV direct2, direct1 ; direct2 ← (direct1)
```

这组指令的功能是将直接地址所规定的内部 RAM 单元内容传送到累加器 A、寄存器 Rn 和内部 RAM 单元。

注意 内部 RAM 单元之间也可以直接传送,当然各个特殊功能寄存器之间也可以直接传送数据。

直接寻址数据传送指令比较丰富,使得内部 RAM 之间的数据传送十分方便,并且不需通过累加器 A 或者工作寄存器来间接传送,从而提高了数据传送的效率。

3) 间接寻址

```
MOV A, @Ri          ; A ← ((Ri))
MOV direct, @Ri      ; direct ← ((Ri))
```

这一组只有两种指令:通过间接寻址传送操作数到 A 和传送到直接地址。但通过寄存器间接寻址取得的源操作数不能直接传送到工作寄存器 Rn,这一点应引起注意。

4) 寄存器寻址

```
MOV A, Rn            ; A ← (Rn)
MOV direct, Rn        ; direct ← (Rn)
MOV Rn, A             ; Rn ← (A)
MOV direct, A         ; direct ← (A)
MOV @Ri, A            ; (Ri) ← (A)
```

例 1 使内存 30H 单元和 40H 单元交换内容,可采用以下的几条指令:

MOV A,30H ;A←(30H),暂存
MOV 30H,40H ;30H←(40H)
MOV 40H,A ;40H←(A),暂存 A 内容

注意 不能只用一条 MOV 30H,40H 指令来完成 30H 单元和 40H 单元内容的交换。

例 2 当 R0=30H,而 30H 单元的内容为 50H 时,执行以下指令的结果如“;”号后所示,即

MOV A,R0 ;A=30H
MOV A,@R0 ;A=(30H)=50H
MOV A,30H ;A=(30H)=50H
MOV A,#30H ;A=30H

2. 涉及外部存储器的数据传送指令

外部存储器包括数据存储器 and 程序存储器,涉及外部存储器的数据传送指令比较少,下面分别进行介绍。

1) 16 位地址传送指令

16 位地址传送指令也称为 16 位立即数传送指令,只有以下一条:

MOV DPTR,#data16

其功能是将立即数的高 8 位送 DPH,立即数的低 8 位送 DPL。例如以下指令:

MOV DPTR,#1234H

则执行完了之后,DPH 中的值为 12H,DPL 中的值为 34H。反之,如果分别向 DPH、DPL 送数,则结果也一样。如有下面两条指令:

MOV DPH,#35H

MOV DPL,#12H

则就相当于执行

MOV DPTR,#3512H

重点提示 16 位地址传送指令主要用于将外部存储器某单元地址送到数据指针寄存器 DPTR。但这个存储单元可以是外部 RAM,也可以是外部 ROM。如果地址传送到 DPTR 后是用到 MOVC 指令中,则所传送的一定是 ROM 地址;若用到 MOVX 指令中,则所传送的一定是 RAM 地址。

2) 访问外部 RAM 的指令

外部 RAM 中的数据可以和累加器 A 互相传送,要用 MOVX 指令,X 代表外部的意思,以区别于内部 RAM 的数据传送指令 MOV,访问外部 RAM 的指令有以下 4 条:

MOVX @Ri,A ;(Ri)←(A)
MOVX A,@Ri ;A←((Ri))
MOVX A,@DPTR ;A←((DPTR))
MOVX @DPTR,A ;(DPTR)←(A)

使用 Ri 的间接寻址,只能传送外部 RAM 的 256 个单元(0000~00FFH)的数据,若要对整个 64Kb 的 RAM 单元寻址,要用 DPTR 来间接寻址。

3) 外部 ROM 的指令

外部 ROM 中的字节传送要用 MOVC 指令,其中的 C 就是 code(代码)的缩写,即表示读取外部 ROM 中的代码。这种指令只有两条,也称为查表指令,如下所示:

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

MOVC A,@A+PC ; $PC \leftarrow (PC) + 1, A \leftarrow ((A) + (PC))$

这两条指令都是一字节指令。前一条指令是采用 DPTR 作为基址寄存器,因此可以很方便地实现整个 64Kb 个 ROM 单元到累加器 A 的数据传送,使用比较方便。后一条指令是 A 的内容作为无符号数和程序计数器 PC 内容(下一条指令的起始地址)相加后得到一个 16 位地址,由该地址指出的 ROM 单元的内容送到累加器 A。该指令使用起来比较麻烦,初学者应尽量使用前一条指令。关于查表指令的具体使用方法,将在第五章进行介绍。

3. 堆栈操作指令

1) 什么是堆栈

日常生活中,我们都注意到过这样的现象:家里洗的碗,一只一只摆起来,最晚放上去的放在最上面,而最早放上去的则放在最下面,在取的时候正好相反,先从最上面取,这种现象我们用一句话来概括,即“先进后出,后进先出”。其实,这种现象还有很多,如往弹仓压入子弹和从弹仓中弹出子弹等,都是“先进后出,后进先出”。这实际是一种存取物品的规则,称之为“堆栈”。

在单片机中,也可以在 RAM 中构造这样一个区域,用来存放数据,这个区域存放数据的规则就是“先进后出,后进先出”。为什么需要这样来存放数据呢?存储器本身不是可以按地址来存放数据吗?对,知道了地址的确就可以知道里面的内容,但如果需要存放的是一批数据,每一个数据都需要知道地址那不是麻烦吗?如果让数据一个接一个地放置,那么只要知道第一个数据所在地址单元就可以了,如果第 1 个数据在 27H,那么第 2 个、第 3 个就在 28H、29H 了。所以利用堆栈这种方法来存放数据可以简化操作。

2) 堆栈的功用

堆栈主要是为子程序调用和中断操作而设立的。其具体功能有两个:保护断点和保护现场。因为在计算机中,无论是执行子程序调用操作还是执行中断操作,最终都要返回主程序。因此,在计算机转去执行子程序或中断服务之前,必须考虑其返回问题。为此,应预先把主程序的断点保护起来,为程序的正确返回做准备。计算机在转去执行子程序或中断服务程序以后,很可能要使用单片机中的一些寄存单元,这样就会破坏这些寄存单元中的原有内容。为了既能在子程序或中断服务程序中使用这些寄存单元,又能保证在返回主程序之后恢复这些寄存单元的原有内容。所以,在转中断服务程序之前,要把单片机中各有关寄存单元的内容保存起来,这就是所谓现场保护。那么,把断点和现场内容保存在哪儿呢?保存在堆栈中。可见,堆栈主要是为中断服务操作和子程序调用而设立的。为了使计算机能进行多极中断嵌套及多重子程序嵌套,所以,要求堆栈具有足够的容量(或者说足够的堆栈深度)。此外,堆栈也可用于数据的临时存放,在程序设计中时常用到。

3) 堆栈指示器

如前所述,堆栈共有两种操作:进栈和出栈。但不论是数据进栈还是数据出栈,都是对堆栈的栈顶单元进行的,即对栈顶单元的写和读操作。为了指示栈顶地址,所以要设置

堆栈指示器 SP。

重点提示 堆栈操作实际上是通过堆栈指示器 SP 进行的读写操作,因为 SP 的内容就是堆栈栈底的存储单元地址,所以堆栈操作是以 SP 为间址寄存器的间接寻址方式。

MCS-51 单片机由于堆栈设在内部 RAM 中,因此 SP 是一个 8 位寄存器,实际上 SP 就是专用寄存器的一员。系统复位后,SP 的内容为 07H,但由于堆栈最好在内部 RAM 的 30H~7FH 单元中开辟,所以在程序设计时应注意把 SP 值初始化为 30H 以后,以免占用宝贵的寄存器区和位寻址区。SP 的内容一经确定,堆栈的位置也就跟着确定下来,由于 SP 可初始化为不同值,因此堆栈位置是浮动的。

4) 堆栈指令

堆栈操作有进栈和出栈两种。数据写入堆栈称为插入运算(PUSH),也叫进栈;数据从堆栈中读出称之为删除运算(POP),也叫出栈。这里所说的“进”与“出”就是数据的进栈和出栈。即先进栈的数据由于存放在栈的底部,因此后出栈;而后进栈的数据存放在栈的顶部,因此先出栈。

进栈时,堆栈指针要先加 1,然后再将数据推入堆栈,原有的数据不变。进栈操作指令为

PUSH direct ; $SP \leftarrow (SP) + 1, (SP) \leftarrow (\text{direct})$

进栈操作只能以直接寻址方式来取得操作数,而不能用累加器 A 或工作寄存器 Rn 作为操作数。例如,要把累加器 A 的内容推入堆栈,应用指令

FUSH ACC

这里 ACC 表示累加器 A 的直接地址 E0H。

出栈操作是把堆栈中的内容弹出到指定的内部 RAM 单元,然后将堆栈指针 SP 减 1,出栈指令和操作为

POP direct ; $\text{direct} \leftarrow ((SP)), SP \leftarrow (SP) - 1$

例 3 说明以下指令的执行过程:

MOV SP, #5FH

MOV A, #10

MOV B, #20

PUSH ACC

PUSH B

POP B

POP ACC

执行 PUSH ACC 时,先将 SP 中的值加 1,即变为 $(SP) = 60H$,然后将 A 中的值(10)送到 60H 单元中,因此执行完本条指令后,内存 60H 单元的值就是 10。

执行 PUSH B 时,先将 SP 的值加 1,变为 61H,然后将 B 中的值送入到 61H 单元中,即执行完本条指令后,61H 单元中的值变为 20。

执行 POP B 时,先将 SP 中的值(现在是 61H)作为地址,取 61H 单元中的数值(现在是 20),送到 B 中,所以执行完本条指令后 B 中的值是 20,然后将 SP 减 1,因此,本条指令执行完后,SP 的值变为 60H,即 $(SP) = 60H$ 。

执行 POP ACC 时,先将 SP 中的值(60H)作为地址,从该地址中取数(10),并送到

ACC 中,所以,执行完本条指令后,ACC 中的值是 10。

方法技巧 ACC 和 B 中的值本来是 10 和 20,执行这段指令后,ACC 的值和 B 的值都没有发生变化。这有什么意义吗?当然,单纯这段程序的确没有意义,但在实际工作中,则在 PUSH B 后往往要执行其他指令,而且这些指令会把 A 中的值、B 中的值改掉,所以在程序的结束,如果要把 A 和 B 中的值恢复原值,那么这些指令就有意义了。

还有一个问题,如果不用堆栈,比如说在 PUSH ACC 指令处用 MOV 60H, A, 在 PUSH B 处用指令 MOV 61H, B, 然后用 MOV A, 60H, MOV B, 61H 来替代两条 POP 指令,不是也一样吗?是的,从结果上看是一样的,但是从过程看是不一样的, PUSH 和 POP 指令都是单字节、单周期指令,而 MOV 指令则是双字节、双周期指令。更何况,堆栈的作用不止于此,所以一般的计算机上都设有堆栈,而在编写子程序、需要保存数据时,通常也不采用后面的方法,而是用堆栈的方法来实现。

例 4 设 SP 的值是 5FH,交换片内 RAM 中 40H 单元与 43H 单元的内容。

程序如下:

PUSH 40H ;(SP)+1=60H, (60H)=(40H)

PUSH 43H ;(SP)+1=61H, (61H)=(43H)

POP 40H ;(40H)=(61H), (SP)-1=60H

POP 43H ;(43H)=(60H), (SP)-1=5FH

该程序的功能示意图如图 4-3 所示。

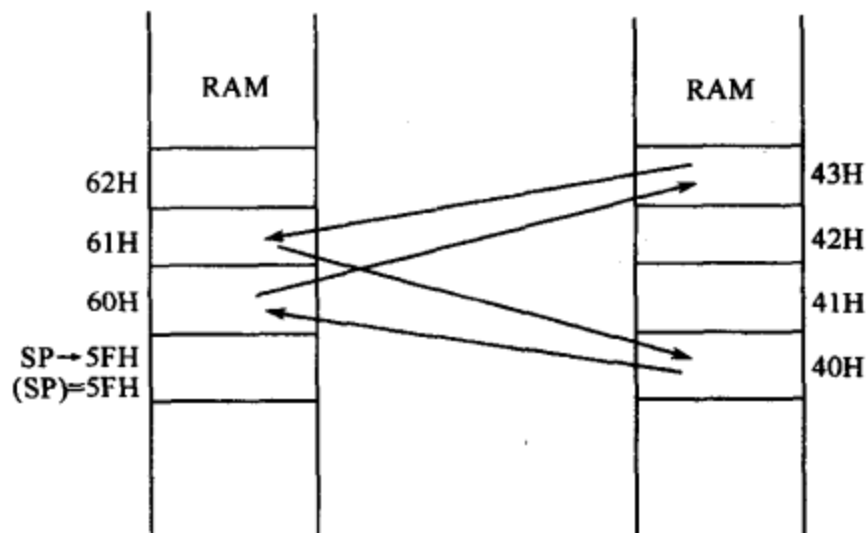


图 4-3 程序功能示意图

从上面的程序可以看到,通过堆栈的“先进后出,后进先出”的规律,实现了两个不同地址单元内容的交换。

方法技巧 从这个例子可以看出,压入堆栈前,堆栈指针 SP 指向 5FH 地址 (SP → 5FH), 即 (SP) = 5FH; 压入堆栈时,首先是堆栈指针 SP 所指向的地址 5FH 加 1, 即 SP 指向 5FH 的下一单元 (SP → 60H)。然后,将 PUSH 指令所指定的地址单元 40H 中的内容送到 60H 中去。而 POP 指令则正好相反,首先是根据堆栈指针 SP 所指向地址的内容送入 POP 后面的参数指定的单元中去,然后再把 SP 的值减 1。

在 PUSH 和 POP 指令中都只有一个操作数,其中 PUSH 指令后面的参数实际是一个“源操作数”,就是待传递的数。那么,还有一个参数在什么地方呢?有“源”就要有“目的”,但这两条指令中并没有写出来。实际上“目的操作数”是隐含在指令中的,它就是堆

栈指针 SP 所指的单元地址。

5) 堆栈的使用

堆栈的使用有两种方式。一种是自动方式,即在调用子程序或中断时,返回地址(断点)自动进栈。程序返回时,断点再自动弹回 PC。这种堆栈操作无需用户干预,因此称为自动方式。另一种是指令方式,即使用专用的堆栈操作指令,进行进出栈操作。进栈用指令 PUSH,出栈用指令 POP。例如现场保护就是指令方式的进栈操作,而现场恢复则是指令方式的出栈操作。

4. 数据交换指令

数据交换用于内部 RAM 和累加器 A 之间进行,有整字节和半字节两种交换。

(1) 整字节交换指令。字节单元与累加器 A 进行 8 位数据交换,有以下 3 条指令:

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

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

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

以上 3 条指令表明,累加器 A 的内容可以和内部 RAM 中任何一个单元内容进行交换。

(2) 半字节交换指令。字节单元与累加器 A 进行低 4 位的半字节交换,只有一条指令:

XCHD A, @Ri ; (A)_{0~3} \leftrightarrow ((Ri))_{0~3}

指令执行后,只将 A 的低 4 位和 Ri 间接寻址单元的低 4 位交换,而各自的高 4 位内容都维持不变。

重点提示 以上的传送和交换指令,一般都不影响各种标志。只是在传送或交换后影响累加器 A 的值时,奇偶标志要按 A 的值来重新设定。

二、算术运算类指令(24 条)

算术操作指令可分为加法(4 条)、带进位加(4 条)、带借位减(4 条)、加 1(5 条)、减 1(4 条)、乘除(2 条)和十进制调整(1 条)指令,共 7 组,前 3 组指令的源操作数分别采用了寄存器寻址、直接寻址、寄存器间接寻址以及立即寻址,而目的操作数都是累加器 A。因此很容易记忆。

说明 算术运算指令都是针对 8 位二进制数的,如果要进行多字节二进制数运算,需要编写程序,通过执行程序实现。

1. 加法指令

加法共 4 条指令:

ADD A, Rn ; A \leftarrow (A) + (Rn)

ADD A, direct ; A \leftarrow (A) + (direct)

ADD A, @Ri ; A \leftarrow (A) + ((Ri))

ADD A, #data ; A \leftarrow (A) + data

这 4 条指令使得累加器 A 可以和内部 RAM 的任何一单元内容进行相加,也可以和一个 8 位立即数相加,相加结果存放在 A 中。无论是哪一条加法指令,参加运算的都是两个 8 位二进制数。对指令使用者来说,这些 8 位二进制数可以当做无符号数(0~255),也可以当做带符号数,即补码数(-128~127)。

例如,对于一个二进制数 1101 0011,用户可以认为它是无符号数,如果用十六进制表

示为 D3, 转换成十进制数为 $D3 = 13 \times 16^1 + 3 \times 16^0 = 211$; 也可以认为它是带符号数, 因最高位为“1”, 说明是负数, 将 1101 0011 转换为补码后为 1010 1101 (补码是把其原码除符号位外的各位先求其反码, 然后在最低位加 1), 如用十六进制表示则为 -2D, 转换成十进制为 $-2D = -(2 \times 16^1 + 13 \times 16^0) = -45$ 。

在求和时, 总是把操作数直接相加, 而无需任何变换。例如, 若 $(A) = D3H$, $(R1) = E8H$, 执行指令 ADD A, R1 时, 有

```
1101 0011
+1110 1000
-----
1←1011 1011
```

即相加后 $(A) = BBH$ 。若认为是无符号数相加, 则 A 的值代表十进制数 187; 若认为是带符号数相加, 则 A 的值为十进制负数 -69。

使用加法指令时要注意对程序状态字 PSW 的影响, 其中包括: 如果位 3 有进位, 将辅助进位标志 AC 置“1”, 反之, AC 清“0”; 如果位 7 有进位, 则进位标志 CY 置“1”, 反之, CY 清“0”; 如果位 6 有进位而位 7 没有进位, 或者位 7 有进位而位 6 没有进位, 则溢出标志 OV 置“1”, 反之, OV 清“0”。

溢出标志的状态, 只有在符号数加法运算时才有意义。当两个符号数相加时, $OV = 1$, 表示加法运算超出了累加器 A 所能表示的符号数有效范围 ($-128 \sim +127$), 即产生了溢出, 因此运算结果是错误的, 否则运算是正确的, 即无溢出产生。例如: $(A) = 0C2H$, $(R1) = 0A9H$, 执行 ADD A, R1 指令后有

```
1100 0010
+1010 1001
-----
1←0110 1011
```

运算结果为: $(A) = 6BH$, $AC = 0$, $CY = 1$, $OV = 1$ 。若 0C2H 和 0A9H 是两个无符号数, 则结果是正确的; 反之, 若 0C2H 和 0A9H 是两个带符号数, 则由于有溢出, 而表明相加结果是错误的, 因为两个负数相加不可能得到正数的和。

2. 带进位加法指令

带进位的加法指令也有 4 条:

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

这 4 条指令的操作, 除了指令中所规定的两个操作数相加外, 还要加上进位标志 CY 的值。

注意 这里所指的 CY 是指令开始执行时的进位标志值, 而不是相加过程中产生的进位标志值。

带进位加法指令对 CY、AC、OV 的影响与前面讲过的不带进位的加法指令相同。即如果位 7 有进位, 则进位标志 CY 置“1”, 反之, CY 清“0”; 如果位 3 有进位, 则辅助进位标志 AC 置“1”, 反之, AC 清“0”; 如果位 6 有进位而位 7 没有进位, 或者位 7 有进位而位 6 没有进位, 则溢出标志 OV 置“1”, 反之, OV 清“0”。

带进位加法指令主要用于多字节二进制数(多于8位)的加法运算中。

3. 带借位减法指令

带借位减法指令有4条:

SUBB A, Rn ; $A \leftarrow (A) - (Rn) - CY$
SUBB A, direct ; $A \leftarrow (A) - (\text{direct}) - CY$
SUBB A, @Ri ; $A \leftarrow (A) - ((Ri)) - CY$
SUBB A, #data ; $A \leftarrow (A) - \text{data} - CY$

减法指令只有一组带借位减法指令,而没有不带借位的减法指令。若要进行不带借位的减法操作,则在减法之前要先用指令使CY清“0”,即使得 $CY=0$,然后再相减。所用的指令为

CLR C ; $CY \leftarrow 0$

它属于布尔操作类指令,这里先拿来用一下。

减法指令也要影响CY、AC和OV标志。即如果位7有借位,则进位标志CY置“1”,反之,CY清“0”;如果位3有借位,则辅助进位标志AC置“1”,反之,AC清“0”;如果位6有借位而位7没有借位,或者位7有借位而位6没有借位,则溢出标志OV置“1”,反之,OV清“0”。

在带借位的减法运算中,只有两个符号数相减时OV标志才有意义, $OV=1$,表示减法运算超出了累加器A所能表示的符号数有效范围($-128 \sim +127$),即产生了溢出,因此运算结果是错误的,否则运算是正确的,即无溢出产生。

例如: $(A)=0C9H$, $(R0)=54H$, $CY=1$ 。执行SUBB A, R0指令为

```
  1100 1001
- 0101 0100
  -----
         1
  -----
  0111 0100
```

运算结果为: $(A)=74H$, $CY=0$, $OV=1$ 。若0C9H和54H是两个无符号数,则结果74H是正确的;反之,若为两个带符号数,则由于 $OV=1$ 有溢出,表明结果是错误的,因为负数减正数其差不可能是正数。

4. 加1指令

加1指令有5条:

INC A ; $A \leftarrow (A) + 1$
INC Rn ; $Rn \leftarrow (Rn) + 1$
INC direct ; $\text{direct} \leftarrow (\text{direct}) + 1$
INC @Ri ; $(Ri) \leftarrow ((Ri)) + 1$
INC DPTR ; $DPTR \leftarrow (DPTR) + 1$

加1指令以使所指定的单元内容加1,不影响PSW的标志位。

下面对指令INC DPTR作一简要说明。数据指针DPTR用来存放16位外部RAM的地址。这条指令能对16位数据指针实行加1操作,结果仍放在数据指针寄存器中。这是唯一的一条16位算术运算指令,这条指令也不影响任何标志。

加1指令有着广泛的用途,例如可以用来修改操作数的地址,以便使用间接寻址的指

令,以下是一个简单的例子。

例 5 有两个无符号 16 位数,分别存放在从 30H 和 40H 开始的数据区中,低 8 位先存,高 8 位在后。写出两个 16 位数的加法程序,和存于 R3(高 8 位)和 R4(低 8 位)。设和不超过 16 位。

由于不存在 16 位数的加法指令,所以只能先加低 8 位,后加高 8 位,而在加高 8 位时要连低 8 位相加的进位一起相加。取操作数时,用寄存器间接寻址方式,并用加 1 指令来修改寄存器的内容,也即修改了操作数的地址。程序如下:

```
MOV R0, #30H      ;第 1 个加数首地址送 R0
MOV R1, #40H      ;第 2 个加数首地址送 R1
MOV A, @R0        ;取第 1 个加数低 8 位
ADD A, @R1        ;低 8 位相加
MOV R4, A         ;存和之低 8 位
INC R0            ;修改地址
INC R1            ;修改地址
MOV A, @R0        ;取第 1 个加数高 8 位
ADDC A, @R1       ;高 8 位相加
MOV R3, A         ;存和之高 8 位
```

5. 减 1 指令

减 1 指令有 4 条:

```
DEC A      ; $A \leftarrow (A) - 1$ 
DEC Rn     ; $Rn \leftarrow (Rn) - 1$ 
DEC direct ; $direct \leftarrow (direct) - 1$ 
DEC @Ri    ; $(Ri) \leftarrow ((Ri)) - 1$ 
```

与加 1 指令一样,减 1 指令也不影响 PSW 的标志位。

说明 对于 DPTR 没有减 1 操作的指令。如果要使 DPTR 完成减 1 操作,则需要几条指令来完成。例如,可以这样来完成:

```
PUSH ACC      ;保护 ACC
CLR C         ;CY=0,准备作减法
MOV A, DPL    ;取 DPTR 低 8 位
SUBB A, #1    ;减 1,  $(A) - 1 - CY$ 
MOV DPL, A    ;保存结果
MOV A, DPH    ;DPTR 高 8 位
SUBB A, #0    ;减可能产生的借位,  $(A) - 0 - CY$ 
MOV DPH, A    ;保存
POP ACC       ;恢复 ACC
```

6. 乘法和除法指令

1) 乘法指令

```
MUL AB      ; $A \times B = BA$ 
```

乘法指令为一字节指令,执行时需 4 个机器周期,相当于执行 4 条加法指令的时间,

是指令系统中执行时间最长的指令。相乘按无符号数进行,两个 8 位无符号数相乘结果为 16 位无符号数,它的高 8 位存于 B 寄存器,低 8 位存于 A 寄存器。

乘法指令执行后,会影响 PSW 的 CY 和 OV 标志位,即执行乘法指令后,进位标志 CY=0。而溢出标志 OV 可以为 1,也可以为零。若相乘后有效积为 8 位,即 B=0,则 OV=0;若相乘后 B≠0,则 OV=1。

2) 除法指令

DIV AB ; A ÷ B = A...B

除法指令也是单字节四周期指令,按两个无符号数进行相除。被除数置于累加器 A,除数则置于寄存器 B。相除之后,商存于累加器 A,余数在寄存器 B 中。除法指令也影响 CY、OV 标志位。相除之后,CY 也一定为零,溢出标志只是在除数 B=0 时才被置 1,因为除数为零时的除法没有意义,故 OV=1,其他情况下 OV 都清零。

7. 十进制调整指令

十进制调整指令用于对 BCD 码十进制数加法运算的结果进行修正,指令格式为:

DA A

前面讲过的 ADD 和 ADDC 指令都是二进制数加法指令,对二进制数的加法运算都能得到正确的结果。但对于十进制数(BCD 码)的加法运算,指令系统中并没有专门的指令,因此只能借助于二进制加法指令,即以二进制加法指令来进行 BCD 码的加法运算。然而二进制的加法运算原则不能完全适用于十进制数的加法运算,有时会产生错误结果,例如:

	1+6=7		7+6=13		9+7=16
(1)	$\begin{array}{r} 0001 \\ + 0110 \\ \hline 0111 \end{array}$	(2)	$\begin{array}{r} 0111 \\ + 0110 \\ \hline 1101 \end{array}$	(3)	$\begin{array}{r} 1001 \\ + 0111 \\ \hline 10000 \end{array}$

其中:式(1)的运算结果是正确的,因为 7 的 BCD 码就是 0111;式(2)的运算结果是不正确的,因为十进制数的 BCD 码中没有 1101 这个编码;式(3)的运算结果也是错误的,因为(9+7)的正确结果应是 16,而运算所得到的结果却是 10(其实 10 正是 16 的十六进制表示法,但我们要的是 BCD 表示法,所以是不对的)。

这种情况表明,二进制数加法指令不能完全适用于 BCD 码十进制数的加法运算,因此,在使用 ADD 和 ADDC 指令对十进制数进行加法运算之后,要对结果作有条件的修正,这就是所谓的十进制调整问题。

出错的原因在于 BCD 码是 4 位二进制编码,4 位二进制数共有 16 个编码,但 BCD 码只用了其中的 10 个,剩下 6 个没用。通常把这 6 个没用的编码(1010、1011、1100、1101、1110 和 1111)称之为无效码。

在 BCD 码的加法运算中,凡结果进入或者跳过无效编码区时,其结果就是错误的。因此,一位 BCD 码加法运算出错情况有以下两种:一是相加结果大于 9,说明已进入无效编码区;二是相加结果有进位,说明已跳过无效编码区。但不管是哪一种出错情况,都是相加结果比正确值小 6,这是因为出错是由 6 个无效编码所造成的。为此,对 BCD 码加法运算只要结果出现上述两种情况之一时,就必须进行调整才能得到正确的结果。调整的方法是把结果加 6,以便把因 6 个无效码所造成的“损失”补回来。

二—十进制的调整过程,是用指令 DA A 根据加法运算后 A 中的值和 PSW 中的 AC、CY 标志位状态,自动选择 4 个修正值(00H、06H、60H 和 66H)中的一个与原运算结果相加,以获得正确的结果。具体调整方法是:

若累加器低 4 位大于 9 或者辅助进位标志 AC=1,则 $A \leftarrow (A) + 06H$;

若累加器 A 的高 4 位大于 9 或者 CY=1(包括由于低 4 位调整后导致上述结果),则高 4 位也作加 6 调整,即 $A \leftarrow (A) + 60H$;

若累加器 A 的高 4 位为 9、低 4 位大于 9,则进行高 4 位和低 4 位分别加 6 调整,即 $A \leftarrow (A) + 66H$ 。

以式(2)为例,如果给结果加上 06H,即 0110,则 $1101 + 0110 = 10011$ 。结果是 13 就对了。

以上所介绍的是十进制调整的原理和方法,具体操作是通过逻辑电路实现的,必须注意,十进制调整指令不能用于十进制减法的调整。

例 6 (A)=56H,(R1)=67H,执行指令:

ADD A,R1

DA A

计算过程如下:

```
0110 0111
+0101 0110
-----
1011 1101
+0110 0110
-----
1←0010 0011
```

由于 67H(0110 0111)加 56H(0101 0110)得 1011 1101,其高 4 位(1011)和低 4 位(1101)均大于 9,所以,执行 DA A 指令需要进行加 66H(0110 0110)校正。因此,执行 ADD A,R1 和 DA A 两条指令后,(A)=23H(0010 0011),CY=1,即可得到 BCD 码 123。

三、逻辑运算及移位类指令(25 条)

逻辑运算包括与、或、异或 3 类,每类都有 6 条指令,移位指令 4 条,此外还有 3 条对累加器 A 清零、求反和半字节交换指令,共计 25 条。

1. 逻辑与运算指令

逻辑与运算用符号 \wedge 表示,6 条逻辑与指令如下:

ANL A,Rn	; $A \leftarrow (A) \wedge (Rn)$
ANL A,direct	; $A \leftarrow (A) \wedge (\text{direct})$
ANL A,@Ri	; $A \leftarrow (A) \wedge ((Ri))$
ANL A,#data	; $A \leftarrow (A) \wedge \text{data}$
ANL direct,A	; $\text{direct} \leftarrow (\text{direct}) \wedge (A)$
ANL direct,#data	; $\text{direct} \leftarrow (\text{direct}) \wedge \text{data}$

逻辑与运算以及其他逻辑运算都是按位进行的。例如,设(A)=07H,(R1)=0FEH,则执行指令:

```

ANL A,R1
    0000 0111
  ^ 1111 1110
  -----
    0000 0110

```

其结果是, $(A)=06H$ 。

逻辑运算除了可用累加器 A 为目的操作数外,还有两条以直接地址单元为目的操作数的指令,这样就很便于对各个特殊功能寄存器的内容按需要进行变换。

2. 逻辑或运算指令

逻辑或运算用符号 \vee 表示,6 条逻辑或指令如下:

```

ORL A,Rn      ;  $A \leftarrow (A) \vee (Rn)$ 
ORL A,direct   ;  $A \leftarrow (A) \vee (\text{direct})$ 
ORL A,@Ri      ;  $A \leftarrow (A) \vee ((Ri))$ 
ORL A,#data    ;  $A \leftarrow (A) \vee \text{data}$ 
ORL direct,A   ;  $\text{direct} \leftarrow (\text{direct}) \vee (A)$ 
ORL direct,#data ;  $\text{direct} \leftarrow (\text{direct}) \vee \text{data}$ 

```

逻辑或指令的执行过程比较简单,例如,设 $(30H)=04H$, $(A)=33H$,执行指令

```

ORL 30H, A
    0000 0100
  V 0011 0011
  -----
    0011 0111

```

其结果是: $(30H)=37H$ 。

逻辑与和逻辑或结合在一起,可方便地对 RAM 单元的内容,特别是对特殊功能寄存器的内容进行变换。

例 7 将累加器 A 的低 4 位送到 P1 口的低 4 位,而 P1 口的高 4 位保持不变。

这种操作不便简单地用 MOV 指令来实现,而可以借助与、或逻辑运算。程序如下:

```

MOV R0,A      ; A 内容暂存 R0
ANL A,#0FH    ; 屏蔽 A 的高 4 位
ANL P1,#0F0H  ; 屏蔽 P1 口的低 4 位
ORL P1,A      ; 完成所需操作
MOV A,R0      ; 恢复 A 的内容

```

在实用中,常会遇到希望使一个单元的某几位内容不变,其余几位为 0。这种操作常用与运算完成:不需要变的各位和“1”相与,需要变为“0”的各位和“0”相与。

3. 逻辑异或运算指令

逻辑异或运算用符号 \oplus 表示,6 条逻辑异或指令如下:

```

XRL A,Rn      ;  $A \leftarrow (A) \oplus (Rn)$ 
XRL A,direct   ;  $A \leftarrow (A) \oplus (\text{direct})$ 
XRL A,@Ri      ;  $A \leftarrow (A) \oplus ((Ri))$ 
XRL A,#data    ;  $A \leftarrow (A) \oplus \text{data}$ 
XRL direct,A   ;  $\text{direct} \leftarrow (\text{direct}) \oplus (A)$ 

```

XRL direct, #data ;direct←(direct)⊕data

逻辑异或运算是当两个操作数不一致时结果为1,两个操作数一致时结果为0,当然这种运算也是按位进行的。例如,设(A)=91H,(R3)=73H,执行指令

```
XRL A,R3
      1001 0001
⊕ 0111 0011
-----
      1110 0010
```

其结果是:(A)=0E2H。

重点提示 利用异或指令可对任何一个内部 RAM 单元取反。异或运算有个特性,即

$$X \oplus 1 = \bar{X}$$

因此,可采用指令 XRL direct, #0FFH,使某一单元的每一位都对 0FF 作异或运算,就可对这个单元的内容求反。

4. 累加器清零、取反和半字节交换指令

清零、取反和半字节交换指令只有对累加器才有效,指令如下:

CLR A ;A←0

这条指令是将累加器 A 的内容清零。

CPL A ;A← \bar{A}

这条指令是将累加器 A 的每一位取反,原来的 1 变为 0,原来的 0 变为 1。

SWAP A ;(A)_{4~7}⇌(A)_{0~3}

这条指令是将累加器 A 的高半字节和低半字节互换。实际上,相当于执行循环左移指令 4 次。

MCS-51 中没有对累加器 A 求补指令。若要进行求补操作,可按“求反加 1”来进行。

例 8 在 30H 和 31H 单元有两个 BCD 数,现要将它们合并放到 30H 单元,以节省内存空间。编写相应的程序段。

使用 SWAP 指令可使这个问题变得很简单,程序如下:

```
MOV R1, #30H ;一个 BCD 数的地址 30H 送 R1
MOV A, @R1 ;A←(30H)
SWAP A ;一个 BCD 码移到高 4 位,低 4 位为 0
INC R1 ;指向另一个 BCD 数的地址 31H
ORL A, @R1 ;31H 的 BCD 码合并到低 4 位
MOV 30H, A ;送回 30H
```

5. 移位指令

移位指令有 4 条:

1) 循环左移指令

RL A ;A_{n+1}←A_n, A₀←A₇

这条指令是将累加器 A 的内容向左环移 1 位,位 7 移至位 0,如图 4-4 所示。

2) 带进位循环左移指令

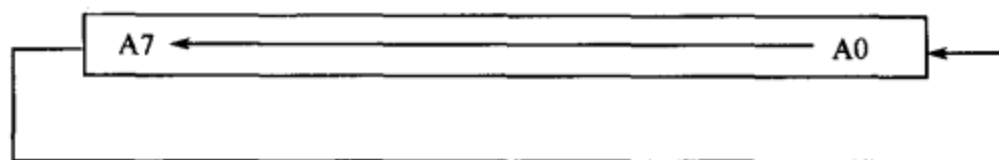


图 4-4 循环左移

RLC A ; $A_{n+1} \leftarrow A_n, CY \leftarrow A_7, A_0 \leftarrow CY$

这条指令是将累加器 A 的内容和进位标志 CY 一起向左环移 1 位, 位 7 移至 CY, CY 移至位 0, 如图 4-5 所示。



图 4-5 带进位循环左移

方法技巧 RLC 通常用做对 A 的内容作乘 2 运算。运算前, 先用指令 CLR C 清 CY, 使 $CY=0$, 然后执行指令 RLC A, 将 CY 连同 A 的内容一同左移循环, 循环后 A 的值是循环前 A 的值的 2 倍。例如: 设 $(A)=1010\ 0000=80H$, 执行以下指令:

CLR C

RLC A

结果为 $(A)=0100\ 0000=40H, CY=1$, 而 140H 正是 80H 的 2 倍。

3) 循环右移指令

RR A ; $A_n \leftarrow A_{n+1}, A_7 \leftarrow A_0$

这条指令是将累加器 A 的内容向右环移 1 位, 位 0 移至位 7, 如图 4-6 所示。

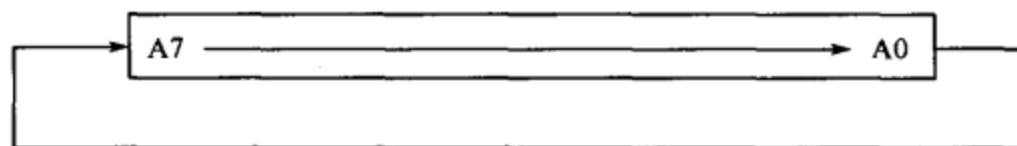


图 4-6 循环右移

4) 带进位循环右移指令

RRC A ; $A_n \leftarrow A_{n+1}, A_7 \leftarrow CY, CY \leftarrow A_0$

这条指令是将累加器 A 的内容和进位标志 CY 一起向右环移 1 位, 位 0 移入 CY, CY 移至位 7, 如图 4-7 所示。

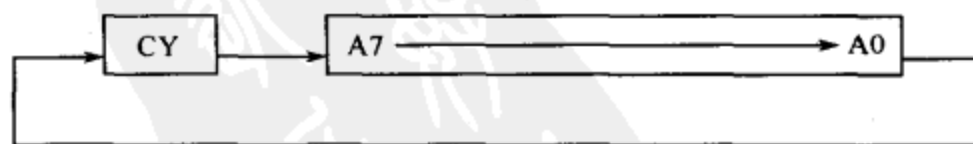


图 4-7 带进位循环右移

例 9 将 DPTR 的内容循环左移 4 位。

由于需要移位 4 次, 安排了一个循环程序。每作一次循环, 对 DPTR 循环左移 1 次。程序如下:

```
PUSH ACC      ;保存 ACC (1)
PUSH R0       ;保存 R0 (2)
```


	MOV R0, #4	;移位次数 (3)
STAR:	MOV A,DPH	;取高字节 (4)
	MOV C,ACC.7	;取 DPTR 的最高位 (5)
	MOV A,DPL	;取低字节 (6)
	RLC A	;循环左移,原 DPTR 最高位移入最低位 (7)
	MOV DPL,A	;存低字节 (8)
	MOV A,DPH	;取高字节 (9)
	RLC A	;循环左移 (10)
	MOV DPH,A	;存高字节 (11)
	DJNZ R0,STAR	;循环,作下一次移位 (12)
	POP R0	;恢复 R0 (13)
	POP ACC	;恢复 ACC (14)

为便于分析,在程序的语句前加入了(1)~(14)标记,以上程序的执行过程是:执行到第(5)条指令时,将 DPH 的最高位 D15 送到 CY,执行到第(6)条指令时,DPL 中的内容如图 4-8(a)所示,执行到第 8 条指令时,DPL 中的内容如图 4-8(b)所示,执行到第(9)条指令时,DPH 中的内容如图 4-8(c)所示,执行到第(11)条指令时,DPH 中的内容如图 4-8(d)所示。完成了 DPTR 中的内容循环左移 1 位的功能。

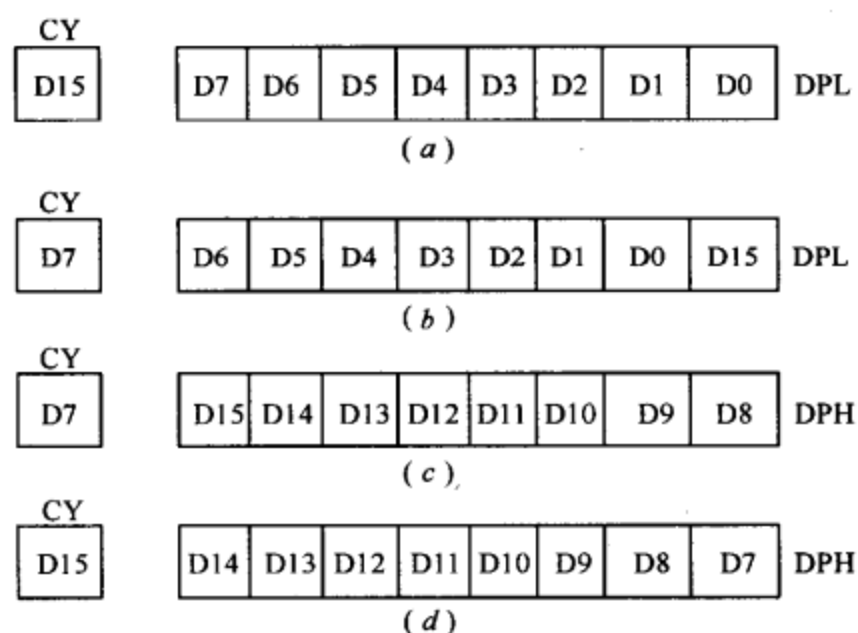


图 4-8 程序执行过程图

指令中,DJNZ 指令是一条转移指令,它先对 R0 减 1,减 1 后,如不等于 0,就转移到 STAR 所在的指令。由于 R0 原来赋值为 4,所以要作 4 次循环转移,结果使 DPTR 循环移位 4 次。

四、控制转移类指令(17 条)

程序的执行是由 PC 值自动加 1 实现的,要改变程序的执行顺序,实现分支转向,应通过强迫改变 PC 值的方法实现,这就是程序转移类指令的基本功能,MCS-51 有比较丰富的控制转移指令,包括无条件转移指令、条件转移指令以及子程序调用及返回指令。

1. 无条件转移指令

无条件转移指令是指程序执行到该指令时,程序立即无条件地转移到所指定的目的地址去执行后面的程序。主要包括以下几种:

1)长转移指令

LJMP addr16 ;PC←addr16

长转移指令 addr16 的操作数是一个 16 位地址。执行这条指令后,PC 的值就等于指令中规定的 16 位地址,即 addr16。16 位地址可以寻址 64K,所以用这条指令可转移到 64KB 程序存储器的任何位置,故称为“长转移”,长转移指令是三字节指令。

2)绝对转移指令

AJMP addr11

指令执行后,首先是 PC 的内容加 2,PC←PC+2,这里 PC 就是指令所在位置的地址。然后由 PC 加 2 后的 PC 值的高 5 位和指令中的 11 位地址 addr11 构成转移地址。

这条绝对转移指令是双字节,而且也能提供 2KB 范围的转移空间,若 2KB 的转移范围已足够,就不必用三字节的长转移指令 LJMP,这样,可以减少指令字节数。

3)短转移指令

SJMP rel

短转移指令是无条件相对转移指令,而不像前两条指令是绝对转移指令。由于 rel 是有符号数,因此也可以向前或向后转移,转移的范围为 256 个单元。

为方便起见,在汇编程序中都有计算偏移量的功能。用户编写汇编源程序时,只需在相对转移指令中直接写上要转向的地址标号就可以了,程序汇编时由汇编程序自动计算和填入偏移量。但手工汇编时,偏移量的值需程序设计人员自己计算。

此外,在汇编语言程序中,为等待中断或程序结束,常有使程序“原地踏步”的需要,对此可使用 SJMP 指令完成:

HERE: SJMP HERE

或 HERE: SJMP \$

在汇编语言中,以“\$”代表 PC 的当前值。

4)变址方式的转移指令

JMP @A+DPTR ;PC←(A)+(DPTR)

这条变址方式的转移指令是一条单字节无条件转移指令,转移的地址由累加器 A 的内容和数据指针 DPTR 内容之和来决定,一般是以 DPTR 的内容为基址,而由 A 的值决定具体的转移地址,两者都是无符号数。

这条指令的特点是转移地址可以在程序运行中加以改变。例如,当 DPTR 为确定的值时,根据 A 的不同的值就可以实现多分支的转移,起到一条指令完成多条指令的功能。

2. 条件转移指令

条件转移指令是指当某种条件满足时,转移才进行;而条件不满足时,程序就继续执行。条件转移指令的条件可以是上一条指令或者更早一点的指令的执行结果。例如:运算结果为零或不为零,为正或者为负,等等。这些结果往往体现在标志位上,即用标志的变化来反映运算结果的某些特征,然后再根据标志来决定是否产生条件转移。另一种情况是条件转移指令本身也包含某种运算,然后根据这种运算的结果来判别是否转移。在

MCS-51 系统中,这两类条件转移指令都存在,但是它的大部分条件转移指令都不是用标志来判断是否存在转移的。

1)累加器判零转移指令

JZ rel

这条指令的功能是:若 A 中的值为 0,则转移;若 A 中的值不为 0,则顺序执行(本指令的下一条指令)。

JNZ rel

这条指令的功能是:若 A 中的值不为 0,则转移;若 A 中的值为 0,则顺序执行(本指令的下一条指令)。

在 MCS-51 的标志位中,没有零标志。这条指令不是用标志来作为条件的,只要前面的指令能使累加器内容为零(或非零),就可以使用这条指令。

这两条都是双字节的相对转移指令,rel 为相对转移偏移量。但在书写源程序时,经常用标号来代替,只是在翻译成机器码时,才换算成 8 位相对地址。

例 10 将外部数据存储器的一个数据块传送到内部数据 RAM,两者的首地址分别为 data1 和 data2,遇到传送的数据为 0 时停止传送。

外部 RAM 向内部 RAM 的数据传送一定要以累加器 A 作为过渡,不能直接在这两种 RAM 之间传送数据。利用判零条件转移正好可以判别是否要继续传送或者终止。

MOV R0, #data1	;外部数据块首址
MOV R1, #data2	;内部数据块首址
LOOP: MOVX A, @R0	;外部 RAM 数据入 A
HERE: JZ HERE	;为零原地踏步
MOV @R1, A	;不为零传送内部 RAM 单元
INC R0	;修改地址指针
INC R1	;修改另一个地址指针
SJMP LOOP	;继续循环

2)比较转移指令

比较转移指令是先对两个规定的操作数进行比较,然后根据比较的结果来决定是否转移到目的地址。比较条件转移指令共有 4 条,差别只在于操作数的寻址方式不同。

CJNE A, #data, rel ;累加器内容与立即数不等就转移

CJNE A, direct, rel ;累加器内容与内部 RAM(包括特殊功能寄存器)内容不等就转移

CJNE Rn, #data, rel ;工作寄存器内容与立即数不等就转移

CJNE @Ri, #data, rel ;内部 RAM 前 128 单元内容与立即数不等就转移

这 4 条指令的功能可以从程序转移和数值比较两个方面来说明:

(1)程序转移:

若左操作数=右操作数,则程序顺序执行,进位标志位 CY 清 0;

若左操作数>右操作数,则程序转移,进位标志位 CY 清 0;

若左操作数<右操作数,则程序转移,进位标志位 CY 置 1。

在 MCS-51 指令系统中,没有单独的比较指令,但可以利用比较条件转移指令来弥补

这一不足。比较操作实际就是作左操作数(目的操作数)减去右操作数(源操作数)的减法运算,只是不保存减法所得到的差,而将结果反映在标志位 CY 和是否转移上。这样就很容易理解并记忆当左操作数大于右操作数时, $CY=0$, 反之 $CY=1$, 这和作减法的结果一致。

(2)数值比较:

若程序顺序执行,则左操作数=右操作数;

若程序转移且 $CY=0$, 则左操作数>右操作数;

若程序转移且 $CY=1$, 则左操作数<右操作数。

3)减 1 转移指令

减 1 转移指令共两条:

DJNZ Rn, rel

这是工作寄存器减 1 转移指令,2B,其功能为:寄存器内容减 1,若没有减到 0,则程序转移;若所得结果为 0,则程序顺序执行。

DJNZ direct, rel

这是直接地址单元内容减 1 转移指令,3B,其功能为:直接地址单元内容减 1,若没有减到 0,则程序转移;若所得结果为 0,则程序顺序执行。

这组指令对于构成循环程序十分有用,可以指定任何一个工作寄存器或者内部 RAM 单元为计数器。对计数器赋以初值以后,就可以利用上述指令;若对计数器减 1 后不为零,就进行循环操作,从而构成循环程序。以下是个简单的例子。

例 11 将内部 RAM 中从 30H 单元开始的 10 个无符号数相加,相加结果送 40H 单元。设相加结果不超过 8 位二进制数。

```
MOV R0, #0AH      ;计数器置初值
MOV R1, #30H      ;数据块首地址送 R1
CLR A              ;A=0
LOOP: ADD A, @R1
INC R1              ;修改地址指针
DJNZ R0, LOOP      ;R0 减 1 不为零循环
MOV 40H, A         ;存和
SJMP $             ;结束
```

以上介绍了 MCS-51 中的各种条件转移指令。这些条件转移指令都是相对转移指令,因此,转移的范围是很有限的。若要在大范围实现条件转移,可将条件转移指令和长转移指令结合起来后加以实现。

3. 子程序调用及返回指令

子程序结构是一种重要的程序结构。在一个程序中经常遇到反复多次执行某程序段的情况,如果重复书写这个程序段,会使程序变得冗长而杂乱。对此,可采用子程序结构,即把重复的程序段编写为一个子程序,通过主程序调用而使用它。这样不但减少了编程工作量,而且也缩短了程序的长度。

主程序调用子程序及子程序的返回过程如图 4-9 所示。当主程序执行到 A 处,执行调用子程序 SUB 时,CPU 将 PC 当前值(下一条指令的第 1 个字节地址)保存到堆栈区,

将子程序 SUB 的起始单元地址送给 PC,从而转去执行子程序 SUB。这是主程序对子程序的调用过程。当子程序 SUB 被执行到位于结束处的返回指令时,CPU 将保存在堆栈区中的原 PC 当前值返回给 PC,于是 CPU 又返回到主程序(A+1)处继续执行,这就是子程序的返回过程。若主程序执行到 B 处时又需要调用子程序 SUB,则再次重复执行上述过程。这样,子程序 SUB 便可被主程序多次调用。

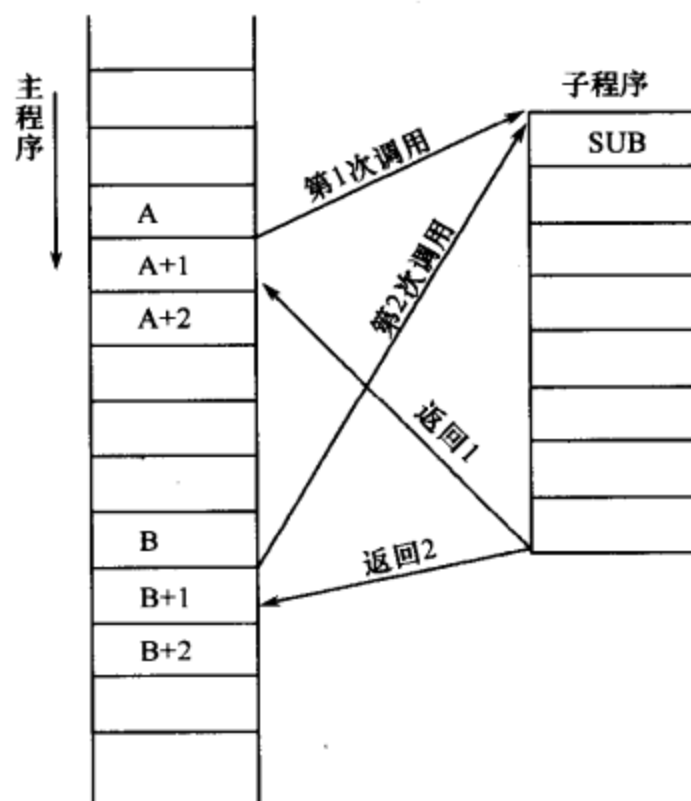


图 4-9 子程序调用与返回示意图

在程序设计过程中,往往还会出现在子程序中要调用其他子程序的情况,这被称之为子程序的嵌套。

子程序调用和转移指令不同,子程序执行完毕后,要返回到原有程序中中断的位置,继续往下执行。因此,子程序调用指令必须能将程序中断位置的地址保存起来;而转移指令则不需要保护断点,也不涉及转移后是否再次转移回来。

1)绝对调用指令

ACALL addr11

这是一条双字节指令,调用的范围为围绕调用指令的 2KB 内。若把 64KB 内存空间以 2KB 为一页,共可分为 32 个页面,绝对调用指令应该和所调用的子程序在同一个页面之内。因此,要注意 ACALL 指令和所调用的子程序的入口不能相距太远,否则就不能实行正确的调用。例如,当 ACALL 指令所在地址为 2300H,可调用的范围是 2000H~27FFH。

为了完成子程序的调用,在调用时还须进行断点保护,断点保护是通过自动方式的堆栈操作实现的,即把下一条指令的 PC 值自动保存起来,待子程序返回时再送回 PC。

方法技巧 8051 有这种指令的目的是减少程序的空间,如果程序做了 100 次的子程序调用,选用 ACALL 会比 LCALL(长调用指令,3B)少 100B 的空间。另外需要注意的是,每做一次调用子程序,会耗掉堆栈 2B 的空间,如果程序开头设置堆栈有 32B 时,连续调用 16 次子程序,就有可能使堆栈爆掉。

2)长调用指令

LCALL addr16

LCALL 调用就没有相同 2KB 位置的限制,所以它可以调控 64KB 的程序空间。LCALL 指令需要 3B 的存放空间。如果汇编语言产生的二进位文件空间小于 2KB,那程序的调用可以全部采用 ACALL;若超过 2KB 时,可能有部分调用要改成 LCALL 才行。

3)返回指令

返回指令也有两条,但并不和两条调用指令对应。一条是一般的子程序返回指令,另一条是从中断服务子程序返回指令。

RET ;子程序返回

RETI ;中断返回

指令的功能都是从堆栈中取出断点,送给程序计数器 PC,使程序从断点处继续执行。RET 应写在子程序的末尾,而 RETI 应用在中断服务子程序的最后。执行 RETI 指令后,将清除中断响应时所置位的优先级状态,使得已申请的较低级中断申请可以响应。

4. 空操作指令

空操作指令是一条控制指令,即控制 CPU 不作任何操作,而只消耗这条指令执行所需要的一个机器周期的时间。

NOP ;PC←(PC)+1

执行这条指令后,只使程序计数器加 1,因为它是单字节指令,在时间上消耗 1 个机器周期,不作其他操作。因此,这条指令可用于等待、延迟等情况。

五、位操作类指令(17 条)

布尔变量即开关变量,它是以位(bit)作为单位来进行运算和操作的。MCS-51 系统加强了对布尔变量的处理能力。在硬件方面它有一个布尔处理器,实际上是一个一位微处理器。它有自己的位累加器(借用进位标志 CY,它是位传送的中心)、自己的存储器(即位寻址区中的各位),也有完成位操作的运算器等。从指令方面,与此相应,也有一个专门处理布尔变量的子集,可以完成以布尔变量为对象的传送、运算、转移控制等操作。这些指令也可称为位操作指令。这一组指令的操作对象是内部 RAM 中的位寻址区的 128 个可寻址位,以及特殊功能寄存器中可以进行位寻址的各位。

1. 位传送指令

可位寻址的各位和位累加器(即进位标志 CY)之间可以互相传送内容,即可将位地址所规定的布尔变量值传送到 CY,也可以将 CY 的值传送到位地址所规定的位。共有两条位传送指令:

MOV C,bit ;CY←(bit)

MOV bit,C ;bit←CY

在指令中 CY 直接用 C 表示,以便于书写。两个可寻址位之间没有直接的传送指令。若要完成这种传送,可以通过 CY 作为中间媒介来进行。

例 12 将 30H 位的内容传送到 20H 位。

传送可通过 CY 来进行,但要注意保持原有 CY 的值不被破坏。

MOV 10H,C ;暂存 CY 内容

MOV C,30H ;CY←(30H)
MOV 20H,C ;(20H)←CY
MOV C,10H ;恢复 CY 的值

重点提示 这里指令中所用的都是位地址,而不是存储单元的地址,故(30H)是表示位地址 30H 这一位的内容,而不是 30H 单元内容。

2. 位置位清零指令

对于进位标志 CY 以及位地址所规定的各位都可以进行置位或清零操作,共有 4 条指令:

CLR C ;CY←0
CLR bit ;bit←0
SETB C ;CY←1
SETB bit ;bit←1

3. 位运算指令

位运算都是逻辑运算,有与、或、非 3 种,共 6 条指令:

ANL C,bit ;CY←CY ∧ (bit)
ANL C,bit ;CY←CY ∧ ($\overline{\text{bit}}$)
ORL C,bit ;CY←CY ∨ (bit)
ORL C,/bit ;CY←CY ∨ ($\overline{\text{bit}}$)
CPL C ;CY←($\overline{\text{CY}}$)
CPL bit ;bit←($\overline{\text{bit}}$)

位操作指令中没有位异或指令,位异或操作可用若干条位操作指令来实现。

例 13 设 E、B、D 都代表位地址,试编写程序完成 E、B 内容的异或操作,结果送 D。

根据 $D = E \oplus B = \overline{E}B + E\overline{B}$,编写的程序如下:

MOV C,B ;取一个操作数
ANL C,/E ;CY← $\overline{E}B$
MOV D,C ;暂存
MOV C,E ;取另一个操作数
ANL C,/B ;CY← $E\overline{B}$
ORL C,D ;C 为异或操作结果
MOV D,C ;存入 D

此外,通过位逻辑运算还可以对各种组合逻辑电路进行模拟,即用软件方法来实现组合逻辑电路的功能。

4. 位控制转移指令

位控制转移指令都是条件转移指令,即以进位标志 CY 或者位地址 bit 的内容作为是否转移的条件。

1) 以 CY 内容为条件的转移指令

JC rel ;CY=1 时就转移,否则顺序执行

JNC rel ;CY=0 时就转移,否则顺序执行

这两条指令常和比较条件转移指令 CJNE 一起使用,先由 CJNE 指令判别两个操作数是否相等;若相等,就继续执行;若不相等,再根据 CY 中的值来决定两个操作数哪一个大,或者来决定如何进一步分支。

2)以位地址内容为条件的转移指令

JB bit,rel ;(bit)=1 就转移,否则顺序执行

JNB bit,rel ;(bit)=0 就转移,否则顺序执行

若(bit)=0,PC←PC+3+rel

JBC bit,rel ;(bit)=1 就转移,并使 bit 位清 0,否则顺序执行

3 条指令都是以位地址的内容作为转移的条件,从条件转移的角度看,JB 和 JBC 指令的作用是一样的,所不同的是,JBC 指令还将使被测试位清 0,使一条 JBC 指令相当于两条指令的功能,即相当于:

JB bit,NEXT

:

NEXT;CLR bit

利用字组指令,可以根据位地址所规定的内容来决定是否产生转移。在 MCS-51 系统中,有许多特殊功能寄存器,它们大多数起着某种控制作用,而这种控制作用一般不是以字节作为控制单元,而是根据位的内容来进行控制的。

例 14 判别累加器 A 中数的正负

判别累加器 A 中数的正负可采用以下两种方法。

方法一 采用字节指令,程序如下:

ANL A,#80H

JNZ NEG

若为负数则转向 NEG 单元。

方法二 采用位控制转移指令,程序如下:

JB ACC.7, NEG

这里利用了位控制转移指令,只需一条指令就可完成判断和转移。

注意 这里的位地址应写为 ACC.7 而不能写为 A.7。

归纳总结 影响标志的指令

当以 MCS-51 指令执行程序时,凡是属于算术运算、逻辑运算和某些位寻址的指令,该指令被执行完后,会将其重要的结果存入标志寄存器中,标志寄存器即我们通称的程序状态字符(PSW),会影响的标志值分别为 CY(借位标志)、OV(溢位标志)和 AC(辅助借位标志)。其中以 CY 借位标志的使用概率最多。MCS-51 影响标志值的指令如表 4-1 所列。

表 4-1 MCS-51 影响标志位的指令一览表

指 令	影 响 标 志		
	CY	OV	AC
ADD	√	√	√

(续)

指 令	影 响 标 志		
	CY	OV	AC
ADDC	✓	✓	✓
SUBB	✓	✓	✓
MUL	0	✓	
DIV	0	✓	
DA	✓		
RRC	✓		
RLC	✓		
SETB C	1		
CLR C	0		
ANL C, bit	✓		
ANL C, /bit	✓		
ORL C, bit	✓		
ORL C, /bit	✓		
MOV C, bit	✓		
CJNE	✓		

注:打✓代表会受影响,CY在MUL和DIV运算指令执行会被清除为0

除了表 4-1 所列举的指令以外,其他的指令不论其写法如何都不会影响到 PSW 的 CY、OV 和 AC 值,例如,我们执行了 INC A 或 DEC A 的指令,不论以上指令被连续执行多少次,都不会影响 CY 和 OV 的值,但是执行 MUL 和 DIV 指令时,C 却固定被设成 0。类似这种差异性在各种单片机 CPU 的指令中时常看到,若不事先知道这些特点,在写程序时就会暗藏危机,往往会误认为一个看起来绝对没有问题的程序,却一直无法顺利地执行程序。

第三节 使用 I/O 访问指令应注意的问题

MCS-51 有 4 个 8 位的双向 I/O 接口,供单片机输入/输出数据使用,对这些口既可以按口寻址,进行字节数据操作,也可以按口线寻址,进行位操作。由于口的操作存在一些特殊问题,而在具体应用中又会常常涉及到它,因此,在这里专门进行说明。

一、可对口进行操作的指令

因为 MCS-51 中把 4 个 I/O 接口归结为专用寄存器之列,因此可以说凡是能对专用寄存器寻址的指令都能用于口的操作,但为了使用方便,在讲述时把口的操作指令分为按口操作和按口线操作两类。在下面的指令中,以 m 代表口的序号,以 n 代表口线的序号。

1. 按字节寻址的口操作指令

凡是能对专用寄存器按字节寻址的指令都能用于口的操作,但其中的典型代表是口输入/输出,因为 MCS-51 单片机中没有专门的输入/输出指令,数据输入/输出操作都是

使用 MOV 传送指令来完成的。输出数据时,就是用 MOV 指令把输出数据写入各口电路的锁存器,而输入数据时,则是用 MOV 指令把各口的引脚状态读入。

(1)按字节寻址的口输出(写)指令有

MOV Pm,A

MOV Pm,#data

MOV Pm,direct

(2)按字节寻址的口输入(读)指令有

MOV A,Pm

MOV direct,Pm

2. 按位寻址的口操作指令

专用寄存器也是可以位寻址的,对于口来说,也可以位寻址。按位寻址的口操作指令有以下几种:

(1)按位寻址的口输入/输出指令有

MOV Pm.n,C

MOV C,Pm.n

(2)按位寻址的口置位、清 0 指令有

SETB Pm.n

CLR Pm.n

(3)按位寻址的口逻辑运算指令有

ANL C,Pm.n

ORL C,Pm.n

(4)按位寻址的口状态判跳指令有

JB Pm.n,rel

JBC Pm.n,rel

二、读引脚与读锁存器

当把口作为输入口而进行读操作时,应注意区分读引脚和读端口两种情况。在第二章介绍口电路结构图 2-7~图 2-10 中,锁存器下方的缓冲器用于读引脚,而锁存器上方的缓冲器则用于读端口(或称读锁存器)。

1. 读引脚

所谓读引脚就是读芯片引脚上的数据,是真正地从外部引脚读进输入的值;这时使用锁存器下方的缓冲器,由“读引脚”信号把缓冲器打开,引脚上的数据经缓冲器通过内部总线读进来,使用 MOV 指令进行数据输入就属于读引脚的操作。

MCS-51 的 4 个 I/O 接口在用于数据输入时均呈准双向口特性,可用图 4-10 所示的结构示意简图进行说明。

图 4-10 中只绘出其中一个端口的一个引脚(设为 P1.0,以下均以此为例说明),单片机内的场效应管 FET 因工作于开关状态,相当于一个电子开关,因此,可以用一个受控的电子开关代替。

这种口在引脚信号输入操作中存在一个特殊问题,即如果电子开关 FET 闭合,导致

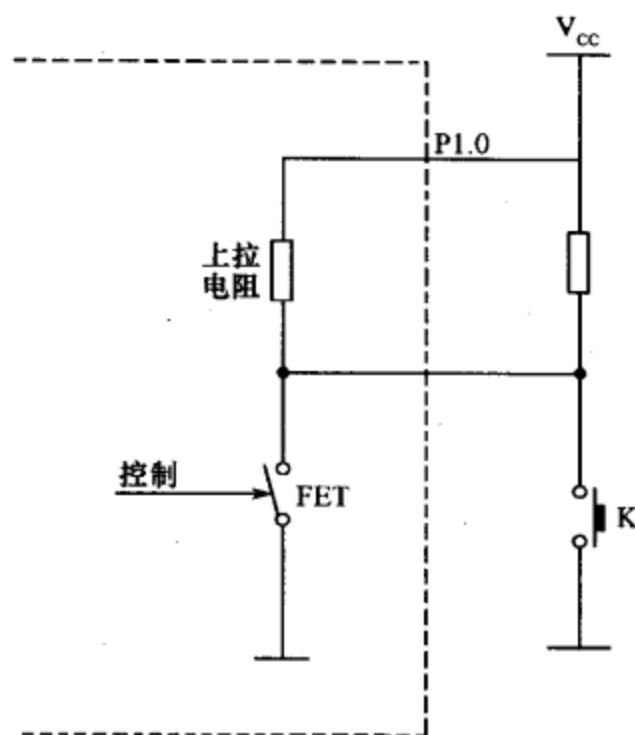


图 4-10 输入/输出口结构示意简图

P1.0 为低电平时，则此时外部的开关 K 无论是断开还是闭合，均被引脚 P1.0 拉低为低电平，不能反映出开关的状态，这实际上就是封锁了 P1.0，使 P1.0 的外部信号不能输入。更为严重的是，当外界输入信号为高电平信号（比如开关 K 断开）时，在拉低过程中产生的大电流还有可能把场效应管 FET 烧坏。为此，在进行引脚数据输入操作之前，必须先向电路中的锁存器写入 1，使 FET 断开（截止），然后再进行读操作。具体应用时，可采用以下程序：

 ORL P1, #01H ; 将 P1.0 置 1，先让单片机内部电子开关 FET 断开，准备输入信号

 MOV A, P1.0 ; 开始输入信号

2. 读锁存器

读锁存器是指该引脚（比如 P1.0）处于输出状态时，有时需要改变这一位的状态，则并不需要真正地读引脚状态，而只是读入锁存器的状态，然后作某种变换后再输出。例如取反指令 CPL P1.0（如果引脚 P1.0 目前的状态是 1，执行该指令后输出变为 0；若引脚目前的状态是 0，执行该指令后输出变为 1），其最终结果虽然把并行口作为输出来使用，但在执行它的过程中却要先“读”入原先的状态，然后经过“取反”电路后再输出。这类指令包括以口做目的操作数的逻辑运算指令（例如 ANL、ORL 和 XRL 等）和口的位操作指令（例如 JBC、CPL 等），通常把这类指令称之为“读—改—写”指令。

对于这类“读—改—写”指令，不直接“读引脚”而“读锁存器”，是为了避免可能出现的错误。例如，本来口线的状态应为“1”（锁存器与引脚状态均为“1”），而如果口线的输出负载恰是一个晶体管的基极，如图 4-11 所示。则导通了的晶体管会把引脚的高电平拉低，这样直接读引脚就会把本来的“1”误读为“0”。为了保证这一类指令的正确执行，80C51 单片机引入了“读锁存器”这种操作。执行这一类指令时，读的是控制锁存器，而不是引脚本身，这样就保证了总能获得正确的结果。

对于“读引脚”和“读锁存器”，使用时，单片机能根据不同的指令，分别发出“读引脚”和“读锁存器”信号，以完成不同的操作。

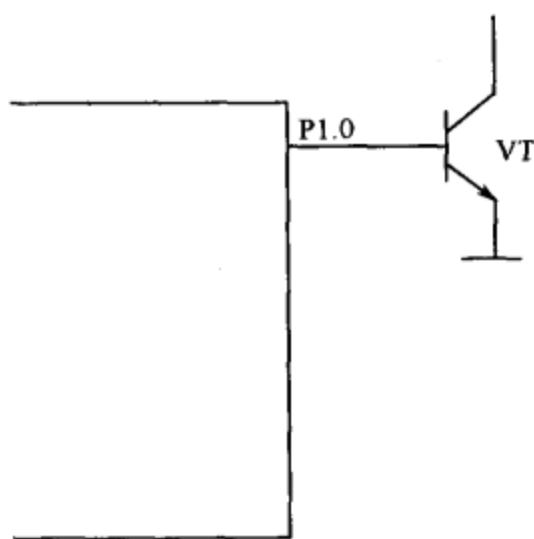


图 4-11 读锁存器示意图

第四节 指令上机练习

在第三章,我们介绍了 Keil C51 软件的使用方法,用 Keil C51 的软件仿真功能学习单片机指令,不但方便直观,而且还可以强化记忆,提高编程能力,下面以一个实例进行说明。

例 15 将 DPTR 的内容循环左移 1 位。

程序如下:

```

    ORG 0000H
    LJMP START
    ORG 30H
START: MOV DPTR, #8001H    ;送 8001H 到 DPTR
      MOV A, DPH           ;取高字节
      MOV C, ACC. 7       ;取 DPTR 的最高位
      MOV A, DPL           ;取低字节
      RLC A               ;循环左移,原 DPTR 最高位移入最低位
      MOV DPL, A          ;存低字节
      MOV A, DPH           ;取高字节
      RLC A               ;循环左移
      MOV DPH, A          ;存高字节
      END

```

打开 μ Vision,单击 File 菜单,再在下拉菜单中单击 New 选项,输入以上源程序;单击菜单上的 File 中的 Save As,输入文件名为 two.asm(在附光盘的 example\ch_4\two 文件夹中),然后,单击“保存”按钮。按 F7 功能键汇编、链接,获得目标文件。注意观察窗口的输出信息,如果源程序有语法错误,会有错误报告出现,应根据提示信息,更正程序中出现的错误,重新编译,直至正确为止。按下 Ctrl+F5 进入调试状态,如图 4-12 所示。

在调试状态下,按 F11 键,使程序单步运行,同时观察窗口左侧寄存器 DPTR 和 A 的

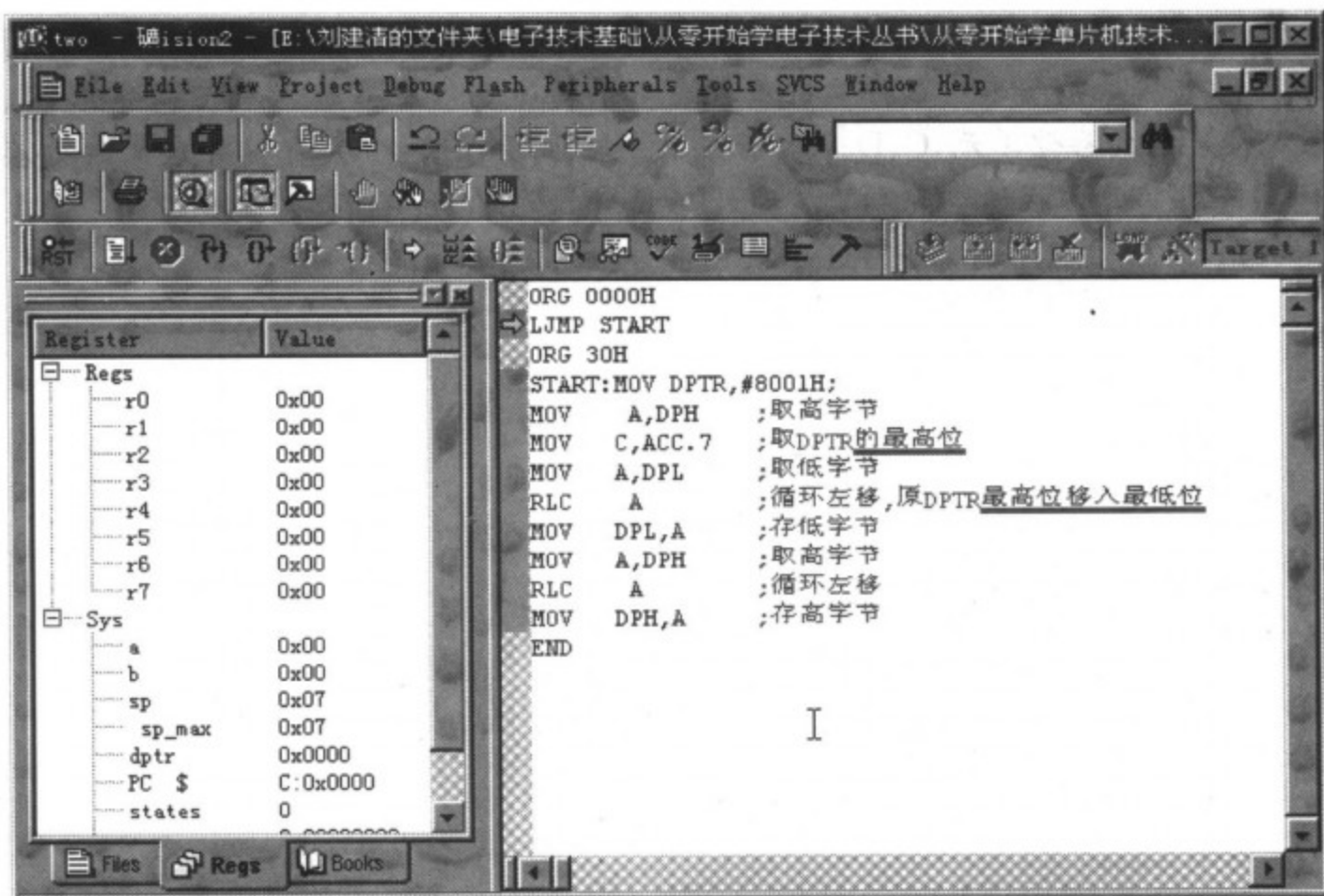


图 4-12 调试状态窗口

状态,当黄色箭头指向 MOV A,DPH 一行时,此时,左边的窗口显示出 DPTR 的值为 0x8001(即十六进制数 8001H),如图 4-13 所示。

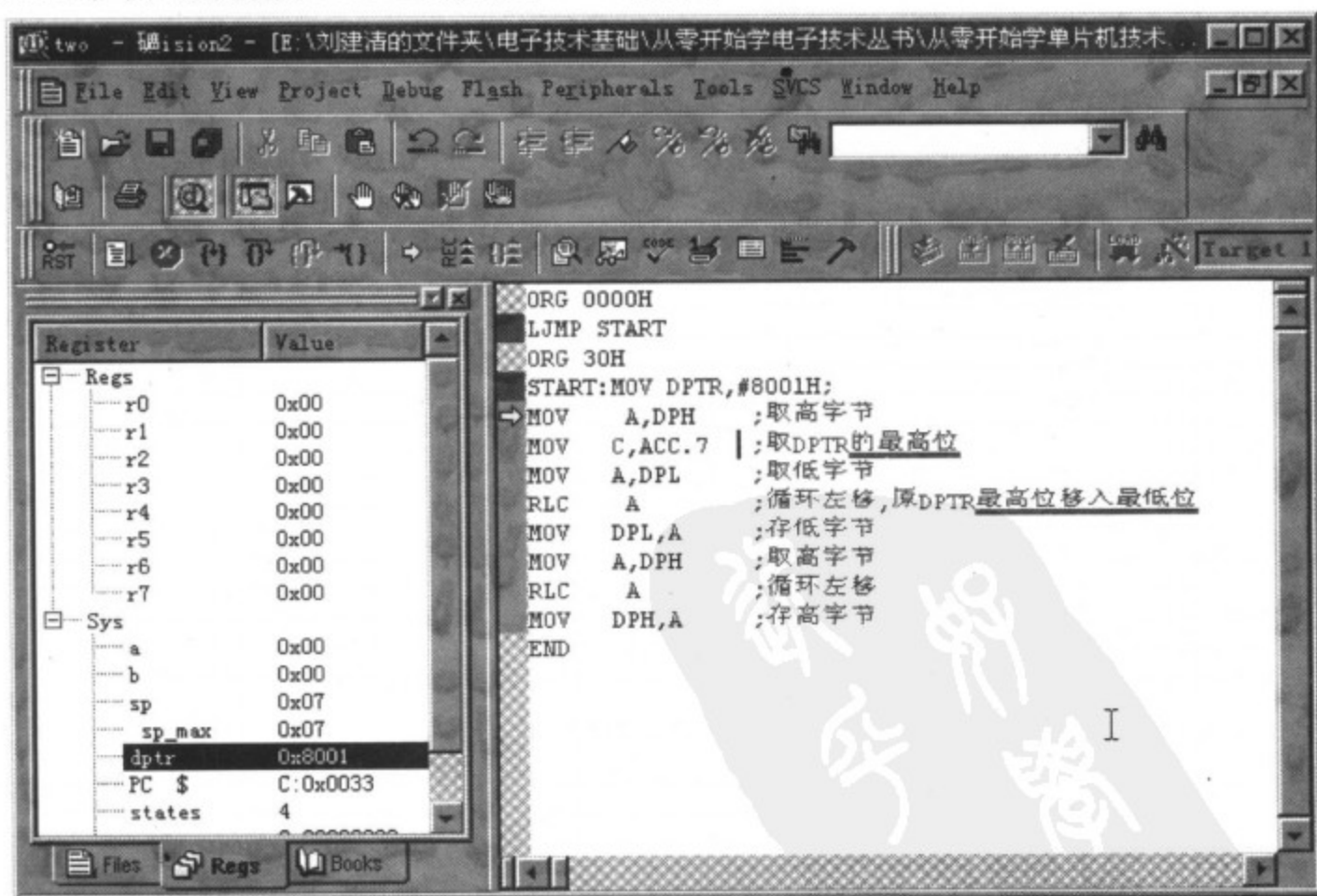


图 4-13 程序单步运行结果

每按一次 F11 键,程序执行一行,程序执行后,随时观察左边的窗口相关寄存器的结果是否和程序的内容保持一致。

方法技巧 调试有些程序时,有时需要观察内存单元的内容,此时,可打开内存观察窗口(打开方法见第三章),在存储区观察窗口的 Address 栏内输入相应的字母(C、D、I、X)来观察不同的存储单元,C、D、I、X 分别代表 ROM 存储空间、直接寻址的片内存储空间、间接寻址的片内存储空间和扩展的外部 RAM 空间,数字代表想要查看的地址。如输入 D:30H,表示要显示内部 RAM 中从 30H 开始的内存单元地址。



第五章 汇编语言简单程序设计

在掌握单片机系统结构基础上,利用 MCS-51 单片机的指令系统,根据应用系统的要求,就可以编写单片机应用程序。本章首先介绍汇编语言的格式,对汇编语言的伪指令进行简要分析,然后分类介绍汇编语言常见的典型程序结构,并对子程序、定时程序和查表程序进行分析和讨论。最后,通过几个简单的实验,来加深读者对程序设计的兴趣,以达到学以致用、用以促学、学用相长的效果。

第一节 概 述

一、程序与语言

程序是指人们按照自己的思维逻辑,使计算机按照一定的规律进行各种操作,以实现某种特定的控制功能而编制的有关指令的集合。编制程序的过程就叫做程序设计。

程序设计语言是实现人机相互交换信息(对话)的基本工具,迄今为止,程序设计语言的种类很多,归纳起来有 3 类:第 1 类是完全面向机器的机器语言;第 2 类是非常接近机器语言的符号化语言即汇编语言;第 3 类为面向过程的高级语言。

1. 机器语言

机器语言是一系列二进制或十六进制编码表示的能够被计算机直接识别和执行的语言。用机器语言表示的程序,称为机器语言程序或目标程序。

下面是一段用 MCS-51 机器语言表示的程序(括号内为十六进制表示法)。

```
1110 0101 0011 0000(E5 30)
0010 0101 0011 0001(25 31)
1111 0101 0011 0010(F5 32)
```

这段程序完成的任务是将内部 RAM 30H 和 31H 单元中的内容相加,结果存入 32H 单元中。可见,用机器语言编程难学、难记。此外,机器语言还随机型不同而异。一般来说,不同型号的计算机的机器语言是互不通用的。不过,不论用何种语言编写程序,最终都必须翻译成机器语言才能执行。

2. 汇编语言

由于机器指令用一系列二进制或十六进制编码表示,不易记忆,也不易查错,因此,在编写程序时,都采用助记符和其他一些符号来编写指令程序,称之为汇编语言程序或汇编语言源程序。

例如,前面提到的“将内部 RAM 30H 和 31H 单元中的内容相加,结果存入 32H 单元”的机器语言代码可以用汇编语言表示如下:

```
MOV A,30H      ;A←(30H)
ADD A,31H      ;A←(A)+(31H)
```

MOV 32H,A ;32H←(A)

显然,汇编语言比机器语言易学易记。但是,计算机不能直接识别和执行汇编语言程序,而要通过“翻译”把源程序译成机器语言程序——目标程序才能执行,这一“翻译”工作称为汇编。目前这种汇编工作既可由计算机借助汇编程序自动完成,也可由人工完成。

有时,需要根据已有的机器语言程序,将其转化为相应的汇编语言程序,这个过程称为反汇编。反汇编对于借鉴他人的编程经验、改进和提高现有系统的性能是大有好处的。一般单片机开发系统都提供了反汇编功能。

说明 汇编语言也是面向机器的,也是一种低级语言。每一类计算机分别有自己的汇编语言。如,适用于PIC的单片机的汇编语言,适用于51系列单片机的汇编语言,适用于AVR单片机的汇编语言等,它们的指令系统是不同的。为充分发挥其灵活性,编程时不仅要掌握指令系统,还要了解计算机的内部结构。

汇编语言在单片机应用系统程序设计中得到了广泛的应用,这是因为汇编语言具有占用内存单元少,执行速度快等优点,但是,汇编语言程序不易维护,可移植性较差。

3. 高级语言

高级语言是一种不依赖于具体计算机的语言,它面向问题或过程,其形式类似于自然语言和数学公式。高级语言的出现,使人们不必深入了解主机的内部结构和工作原理,只要设计出算法就能很容易地将它用高级语言表示,从而可以集中精力考虑解决问题的方法,提高编程效率。

但是,计算机并不能直接执行高级语言程序。用高级语言写的程序在执行时必须先“翻译”成机器语言,一般通过解释程序或编译程序实现。编译方式与解释方式的不同之处在于编译程序还把高级语言源程序整个地翻译成用机器语言表示的目标程序。C语言等高级语言采用编译方式,而像BASIC高级语言则采用解释方式。

目前一些单片机开发系统在提供汇编程序的同时,也同时提供C编译程序,允许用户用C语句编写程序。采用C语言,不必对单片机的硬件接口的结构有深入的了解,编译器可自动完成变量存储单元的分配,大大加快软件的开发速度;采用C语言可以很容易地进行单片机的程序移植工作,有利用产品中单片机的重新选型,而且C语言还可以嵌入汇编来解决高时效性的代码编写问题。用C语言编程的主要缺点是:编程要经过编译程序编译,其目标程序较长,占用内存单元多、运行速度相对较慢,一般不太适宜于控制领域。

归纳总结 机器语言、汇编语言和高级语言对比情况如表5-1所列。

表 5-1 机器语言、汇编语言和高级语言对比情况

名称	特点	缺点	优点	适用场合
机器语言	用机器码书写指令	不易被人们识别和读写。 难写、难读、难交流	计算机可以直接识别和执行	无
汇编语言	用符号书写指令(用助记符表示操作码,特殊符号表示操作数)	机器不能直接识别;程序员必须了解机器的结构和指令系统,不易推广和普及; 不能移植,不具备通用性	易为人们识别、记忆和读写	实时控制系统

(续)

名称	特点	缺点	优点	适用场合
高级语言	用以英语为基础的语句编程	机器不能直接识别,执行时间长	易于推广和交流;不依赖于机器,具有通用性	科学运算和数据处理

二、汇编语言源程序的格式

汇编语言源程序由一条一条汇编语句组成,每条汇编语句独占一行,典型的汇编语句格式如下:

[标号]:操作码 [操作数];[注释]

即一条汇编语言由标号、操作码、操作数和注释 4 部分组成,其中,方括号括起来的是可选部分,视需要而定。

1. 标号

标号是指指令的符号地址,标号和操作码之间用“:”作分隔符,也可再加若干空格。有了标号,程序的其他语句才能访问该语句。标号并不是每一行都必须有,而是在需要时才使用,对于不需要转移的语句,一般不需要加标号。

标号必须符合以下规定:由 8 个或 8 个以内的字母、数字构成,第 1 个必须是字母,同一程序内不能有相同的标号,不能用助记符、伪指令、寄存器名和特殊符号等作标号。AB1、NEXT、LOOP1 等是一些正确的标号;而 2A、S+M、EQU、ADD(后两种为指令保留字)等不能用做标号。

2. 操作码

操作码说明语句的功能,它是汇编语句中必不可少的部分。操作码和操作数之间要用空格进行隔开。

3. 操作数

操作数说明操作的对象。操作数可以是具体的数、标号(符号地址)、寄存器、直接地址等。对于用十六进制表示的操作数,若字母 A、B、C、D、E、F 在最高位,应在前面补 0,如 0A1H(不能写做 A1H),0FFEEH(不能写做 FFEEH)等。

根据指令的不同,操作数可以有 1 个、2 个、3 个或者没有,如果有 2 个或 3 个,则各个操作数之间用逗号隔开。

4. 注释

注释用于说明语句的功能,增加程序的可读性。使用时必须以“;”开头,表明以下为注释内容,若一行不够,可以另起一行,新行也必须以“;”开头。

三、51 汇编软件汇编失败原因

单片机的汇编语言编写时要注意语法,语法错误会造成汇编失败,常见的汇编错误有如下几种:

(1)标号重复,常见于复制、粘贴程序时忘记修改标号,造成出现多个相同的标号,标号是不允许重复的。

(2)标点符号以全角方式输入,MCS-51 程序要求标点符号为半角方式,否则汇编失败。

(3)注释太长,有时为了以后读懂程序,写了很长的注释,超过 80 个字符(或 40 个汉字)也会造成汇编失败,解决办法可以将太长的注释分成多个注释。

(4)数值 A~F 在最前面时遗漏 0,根据要求应该在 A~F 前加 0,例如,#FFH 应写成#0FFH。

(5)字母 O 和数字 0 搞混,有时候这两个字看上去完全相同,在计算机键盘中的位置又十分靠近,书写时要十分注意。

(6)字母 I 和数字 1 混淆,这也是十分常见错误。

(7)标号后边遗漏“:”。

(8)将 A 与 ACC 混淆,如将“PUSH ACC”误写为“PUSH A”等。

(9)标号使用了特殊字符,比如:T1、T2,这些字符有特定的含义,不允许用于标号。

(10)AJMP 跳转超过 2Kb 地址,AJMP 属于短跳转命令,有 2Kb 地址范围的限制。

(11)创造发明不存在的汇编语言指令,这种指令汇编程序不支持,芯片也不认可。

(12)寄存器重复调用,比如主程序中设定了 R4=5,在延时子程序又用到 R4,使 R4 的值发生紊乱,将会造成程序无法正常执行。

第二节 汇编语言的伪指令

汇编语言程序的机器汇编是由计算机自动完成的。为此,在源程序中应向汇编程序发出指示信息,告诉它应该如何完成汇编工作,这一任务是通过使用伪指令来实现的,伪指令具有控制汇编程序的输入/输出、定义数据和符号、按条件汇编、分配存储空间等功能。

伪指令是程序员发给汇编程序的命令,也称为汇编命令或汇编程序控制指令。只有在汇编前的源程序中才有伪指令。汇编得到目标程序后,伪指令已无存在的必要,所以伪指令没有相应的机器代码,在目标程序中见不到与伪指令相对应的机器码。

在各种汇编语言中,伪指令的符号和含义可能有所不同。当使用某一种汇编程序进行自动汇编时,应先参考其用户手册。下面介绍 MCS-51 常用的伪指令。

一、汇编起始地址伪指令 ORG

ORG 伪指令的功能是规定下面的源程序或数据的起始地址,一般格式为
[标号]:ORG 地址

其中标号是可选项,可根据情况选用,例如:

```
ORG 2000H
```

```
START:MOV A,#00H
```

规定了标号 START 所在的地址为 2000H,第一条指令就从 2000H 开始存放。

一般在一个汇编语言源程序或数据块的开始,都用一条 ORG 伪指令规定程序或数据块存放的起始位置。但是,在一个源程序或数据块中,可以多次使用 ORG 指令,以规定不同的程序段或数据块的起始位置。规定的地址应该是从小到大,而且不允许有重叠。

注意 一个源程序若不用 ORG 指令,则从 0000H 开始存放目标码。

二、汇编结束伪指令 END

END 伪指令的功能是用来告诉汇编程序汇编到此结束。其格式为
END

在 END 以后所写的指令,汇编程序都不予理会。一个源程序只能有一个 END 指令,并放到所有指令的最后。否则,就有一部分指令不能被汇编。

三、定义字节伪指令 DB

DB 伪指令的功能是从程序存储器的某地址单元开始,存入一组规定好的 8 位二进制常数。其一般格式为

[标号]:DB 8 位二进制常数表

这个伪指令在汇编以后,将影响程序存储器的内容。例如:

ORG 2000H

TAB:DB 3FH,06H,5BH,0E6H

以上伪指令经汇编以后,将对 2000H 开始的若干内存单元赋值:

(2000H)=33FH,(2001H)=06H,(2002H)=5BH,(2003H)=0E6H

其中 36H 为字符 6 的 ASCII 码。

四、定义字伪指令 DW

DW 伪指令的功能是从指定地址开始,定义若干个 16 位数据。其格式为

[标号]:DW 16 位数据表

每个 16 位数要占两个 ROM 单元,在 MCS-51 系统中,16 位二进制数的高 8 位先存入低地址单元,低 8 位存入高地址单元。例如:

ORG 2000H

TAB:DW 100H,1A2H

汇编以后,将对 2000H 开始的若干内存单元赋值:

(2000H)=01H,(3001H)=00H,(3002H)=01H,(3003)=A2H

五、定义内存空间伪指令 DS

DS 伪指令的功能是从指定地址开始,保留若干字节内存空间备用。其格式为

[标号]:DS 16 位数据表

在汇编以后,将根据表达式的值来决定从某地址开始留出多少个字节空间。表达式也是一个指定的数值,例如:

ORG 2000H

ADDR: DS 08H

START:MOV A,40H

汇编以后,从 2000H 单元开始,保留 8B 的内存单元留做它用,那么第 3 条指令的标号 START 的地址应为 2008H 单元。

说明 以上的 DB、DW、DS 伪指令都只对程序存储器起作用,即不能用它们来对数据存储器的内容进行赋值或其他的初始化工作。

六、赋值伪指令 EQU

EQU 伪指令的功能是将一个常数或特定的符号赋予规定的字符串,其格式为
字符名称 EQU 赋值项

这里使用的“字符名称”不是标号,不用“:”来做分隔符,若加上“:”反而被汇编程序认为错误。字符名称是指不包括分隔符的字符串。用 EQU 指令赋值以后的字符名称可以用做数据地址、代码地址或者直接当做一个立即数使用。使用 EQU 指令时,必须先赋值后使用。例如:

```
ORG 8000H
ABC EQU 30H
MOV A, AAC
```

这里将 ABC 等值为汇编符号 30H,在指令中,ABC 就可以取代 30H 来使用。

七、位地址定义伪指令 BIT

BIT 伪指令的功能是将位地址赋予所规定的字符名称。其格式为
字符符号 BIT 位地址

例如:

```
IN1 BIT P1.0
IN2 BIT P1.1
```

这样就把两个位地址分别赋给两个变量 IN1 和 IN2,在编程中它们就可当做位地址来使用。

第三节 汇编语言典型程序结构

在程序设计中,将会遇到一些复杂的程序,但不论程序如何复杂,都可以看成是一个个基本程序结构的组合。下面简要介绍汇编语言中几种典型的程序结构。

一、顺序结构程序

顺序程序结构是最简单的一种结构,在程序中即无分支、循环,也不调用子程序。在流程图中表示为任务框一个一个地串行连接。计算机执行程序时表现为,从头至尾严格按照次序一条语句一条语句地顺序执行,并且每一条语句均被执行一遍。顺序结构程序往往用来解决一些简单的算术及逻辑运算问题,主要用数据传送指令和数据运算类指令实现。

例 1 将内部 RAM 30H、31H、32H 单元中的无符号数相加,和存入 R0(高位)及 R1(低位)。

3 个无符号数相加,和可能超过 8 位二进制数(255),因此要按双字节加法来处理。即将相加结果中的进位,存入高 8 位寄存器中,并与以后相加产生的进位相加。流程图如图 5-1 所示。

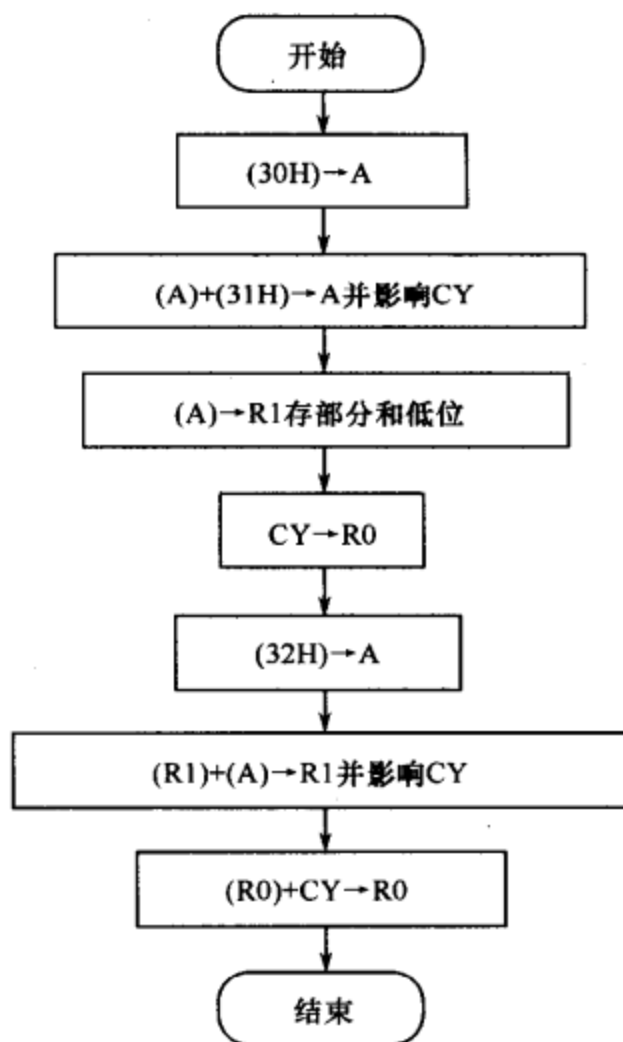


图 5-1 例 1 流程图

根据流程图,编写源程序如下:

MOV A,30H	;取 30H 单元值
ADD A,31H	;A←(30H)+(31H),并影响 CY
MOV R1,A	;暂存于 R1 中
CLR A	;A 清零
ADDC A,#00H	;A←0+0+CY
MOV R0,A	;CY 送 R0
MOV A,32H	;取 32H 单元值
ADD A,R1	;A←(30H)+(31H)+(33H),并影响 CY
MOV R1,A	;和数存入 R1
CLR A	;A 清零
ADDC A,R0	;上次产生的高位加本次进位
MOV R0,A	;高位和数存入 R0

二、分支结构程序

分支程序是利用条件转移指令,使程序执行某一指令后,根据条件(即上面运算的情况)是否满足,来改变程序执行的次序。在设计分支程序时,关键是如何判断分支的条件。在 MCS-51 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、减 1 转移指令 DJNZ 和位条件转移指令 JC(JNC)、JB

(JNB)、JBC 等指令。通过这些指令,就可以完成各种各样的条件判断,如正负判断、溢出判断、大小判断等。

注意 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。

例 2 设变量 X 存放于 $M1$ 单元,函数值 Y 存放在 $M2$ 单元。试按照下式对 Y 赋值。

$$Y = \begin{cases} 1(X > 0) \\ 0(X = 0) \\ -1(X < 0) \end{cases}$$

X 是有符号数,因此可以根据它的符号位来决定其正负。判别符号位是 0 还是 1,则可利用 JB 或 JNB 指令。而判别 X 是否等于 0,则可以直接使用累加器判零指令。把这两种指令结合使用就可以完成本题的要求。根据要求可画出流程图如图 5-2 所示。

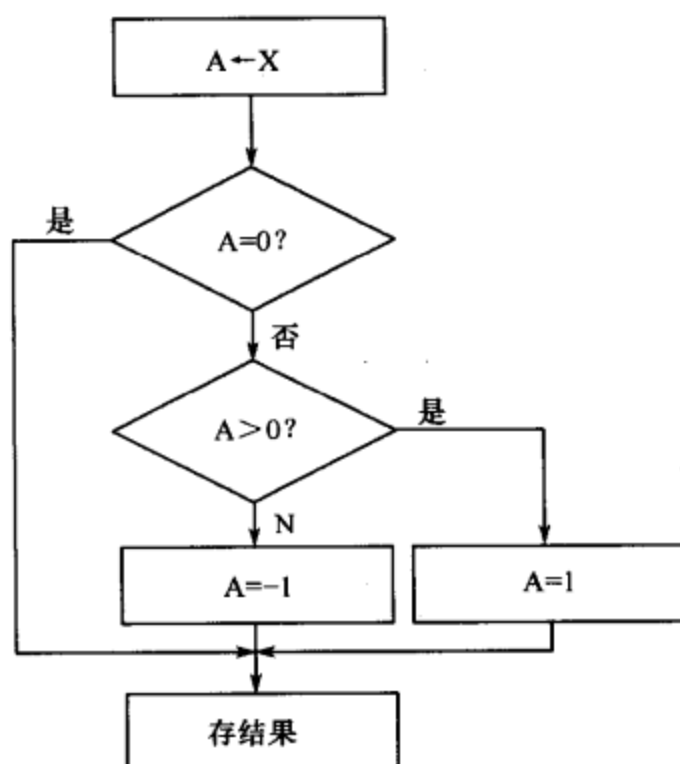


图 5-2 例 2 流程图

先进行比较、判断,然后按比较结果赋值。这实际是个三支而归一的流程图,因此,至少要用两个转移指令。程序如下:

M1 EQU 30H	
M2 EQU 31H	
ORG 0000H	;源程序开始
SJMP MAIN	;跳过中断入口区
ORG 0100H	;主程序开始
MAIN: MOV A, M1	;取出 X
JZ COMP	;若 X=0 则转移到 COMP
JNB ACC. 7, POSI	;X>0 则转移到 POSI
MOV A, #0FFH	;X<0 则 Y=-1
SJMP COMP	

```

POSI:  MOV A, #1           ;X>0 则 Y=1
COMP:  MOV M2, A          ;存函数值
HERE:  SJMP $
      END

```

三、循环结构程序

在实际工作中,有时要求对某一问题进行多次重复处理,而仅仅只是初始条件不同,这种计算过程称之为具有循环特征的,而循环程序设计是解决这类问题的一种行之有效的方法。循环程序是采用重复执行某一段程序来实现要求完成计算的编程方法。

1. 循环程序的组成

循环程序一般包括以下 5 个部分:

- (1)初始化部分。为循环做准备工作,包括预置变量、计数器和数据指针初值等。
- (2)循环工作部分。它是循环程序的主体,用来完成循环的基本操作。
- (3)修改部分。为循环参数作必要的修改,如修改操作数地址、计数器,为下一次执行循环体做好准备。
- (4)控制部分。根据循环条件来判断、控制循环的继续和终止。
- (5)结束部分。主要是对循环的结果进行必要的处理,如将结果送入某一寄存器或内存区域。

2. 循环程序的基本结构形式

根据循环组成部分的排序,循环程序通常有“先判断后处理”和“先处理后判断”两种结构形式,如图 5-3(a),(b)所示。

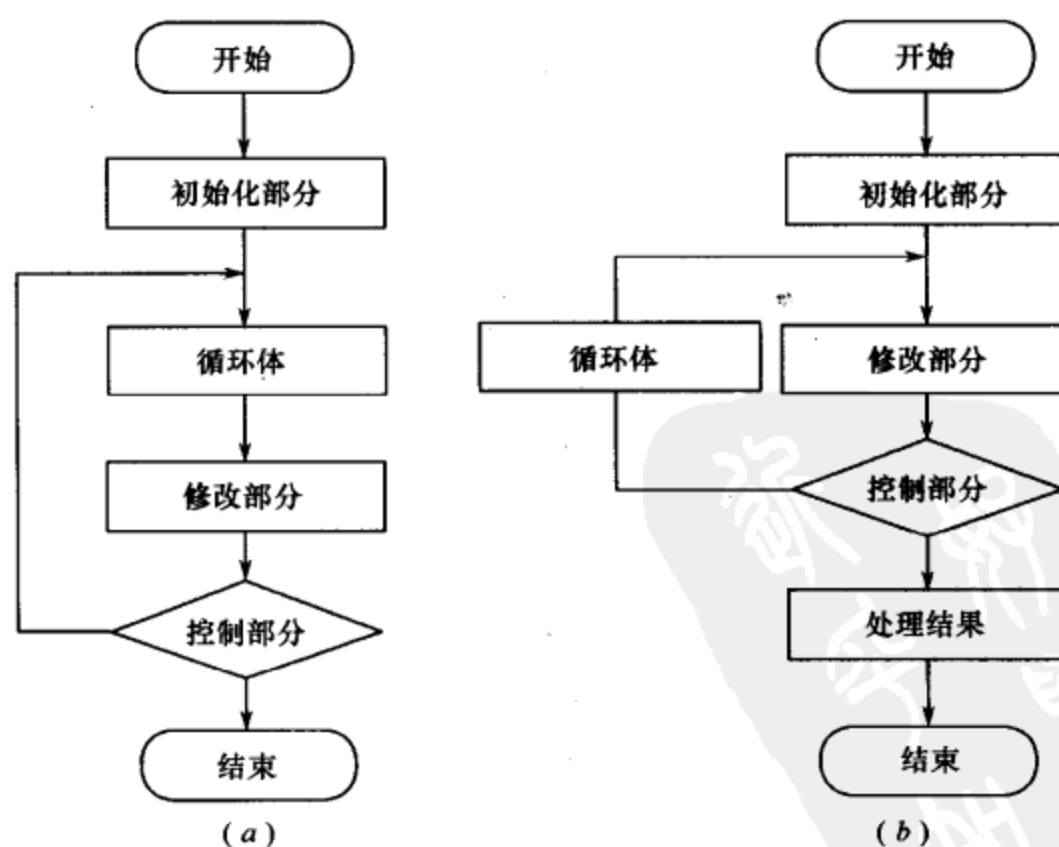


图 5-3 循环程序的结构
(a)先判断后处理; (b)先处理后判断。

3. 循环控制的方法

常用的循环控制方法有计数器控制和条件标志控制两种。用计数器控制循环时,循环次数是已知的,可在循环初始部分将次数置入计数器中,每循环一次计数器减1,当计数器的内容减到零时,循环结束,常用 DJNZ 指令实现;相反,有些循环程序中事先无法知道循环次数,而只知道循环有关的条件,这时只能根据给定的条件标志来判断循环是否继续,一般用条件判别指令实现。

例 3 将片内 20H~70H 中的内容传送到 0ABH~0FBH 中去。

源程序如下:

```
START: MOV R0, #20H      ;R0 作为地址指针,首先指向 20H 单元
        MOV R1, #0ABH    ;R1 作为地址指针,首先指向 0ABH 单元
        MOV R7, #51H     ;R7 作为循环变量,控制循环次数
LOOP:   MOV A, @R0        ;将 R0 所指地址中的数送至累加器 A
        MOV @R1, A       ;再将 A 中的数送至 R1 所指地址单元
        INC R0            ;R0 指向下一地址
        INC R1            ;R1 指向下一地址
        DJNZ R7, LOOP     ;循环变量 R7 减 1,减 1 后不为 0 则至 LOOP 处循环
```

例 4 将片外 2040H~205FH 中的内容送片内 20H~3FH 单元。

源程序如下:

```
START: MOV R0, #20H
        MOV DPTR, #2040H
        MOV R6, #20H
LOOP:   MOVX A, @DPTR
        MOV @R0, A
        INC R0
        INC DPTR
        DJNZ R6, LOOP
```

例 5 把片外 RAM 起始地址为 0000H 的 100 个连续单元中的内容送到片外以 2000H 开始的单元中。

程序如下:

```
START: MOV R0, #00H
        MOV DPTR, #2000H
        MOV R7, #100D
LOOP:   MOVX A, @R0
        MOVX @DPTR, A
        INC R0
        INC DPTR
        DJNZ R7, LOOP
```

例 6 设字符串存放在内部 RAM 31H 开始的单元中,以“\$”作为结束标志,现要求计算该字符串的长度,并将其存放在 30H 单元。

源程序如下：

```
CLR A
MOV R0, #31H           ;取数
LOOP: CJNE @R0, #24H, NEXT ;与“$”(ASCII 值为 16 进制 24)比较
      SJMP COMP         ;找到“$”结束
NEXT: INC A             ;不为“$”，则计数器加 1
      INC R0             ;修改地址指针
      SJMP LOOP
COMP: MOV 30H, A        ;存结果
```

第四节 汇编语言子程序的设计

在程序设计中,常常会遇到某些功能完全相同的程序段在同一程序的多处或不同程序中出现,为了节省存储空间,减少编制程序的重复劳动,可以将这些多次重复的程序段独立出来,将它编制成一种具有公用性的、独立的程序段——子程序,并通过适当的方法把它与其他程序段链接起来,这种程序设计的方法称为子程序设计。

每个子程序应该有一个入口,并以标号作为标识,以便主程序调用,主程序通过调用指令 ACALL 或 LCALL 调用子程序,子程序以 RET 指令作为结束,以便正确地返回到主程序中的断点地址,继续执行主程序。

重点提示 所谓断点地址是指子程序调用指令的下一条指令的地址,取决于调用指令的字节数,若采用 ACALL(2B)调用指令,则断点地址为 PC+2(PC 为调用指令所在地址);若采用 LCALL(3B)调用指令,则断点地址为 PC+3。

一、子程序调用过程中参数的传递

为了使子程序具有通用性,子程序处理过程中用到的数据都由主程序提供,子程序的某些执行结果也应送回主程序。这就存在着主程序和子程序之间的参数传递问题。参数传递通常采用以下几种方法。

1. 寄存器或累加器传送

数据通过工作寄存器 R0~R7 或累加器 A 来传送。在调用子程序之前数据先送入寄存器或者累加器,子程序执行以后,结果仍由寄存器或累加器送回。这是一种最常使用的方法,其优点是程序简单、速度快,其缺点是传递的参数不能太多。

2. 指针寄存器传送

数据一般存放在数据存储器中,可用指针来指示数据的位置,这样可大大节省传递数据的工作量,并可实现变长度运算。若数据在内部 RAM 中,可用 R0、R1 作为指针;参数在外部 RAM 或程序存储器中,可用 DPTR 作为指针,参数传递时只通过 R0、R1、DPTR 传送数据所存放的地址,调用结束后,传送回来的也只是存放数据的指针寄存器所指的数据地址。

3. 堆栈传送

可以用堆栈向子程序传递参数和从堆栈获取结果。调用子程序前,先把要传送的参

数用 PUSH 压入堆栈,进入子程序后,可用堆栈指针间接访问堆栈中的参数。同时可把结果送回堆栈中。返回主程序后,可用 POP 指令得到这些结果。

注意 在调用子程序时,断点地址也会压入堆栈,占用两个单元,在子程序中弹出参数时,不要把断点地址也弹出。此外,在返回主程序时,要把堆栈指针指向断点地址,以便能正确返回。

二、调用子程序时的现场保护问题

在子程序执行时,可能要使用累加器和某些工作寄存器,而在调用子程序前,这些寄存器中可能存放有主程序的中间结果,它们在子程序返回后仍需使用,这样就需要在进入了程序时,将要使用的累加器和某些工作寄存器的内容转移到安全区域保存起来,即现场保护。

当子程序执行完、即将返回主程序之前,再将这些内容取出,送回到累加器和原来的工作寄存器中,即恢复现场。

保护现场和恢复现场通常使用堆栈操作,即在进入子程序时,将需要保护的数据压入堆栈加以保护,在返回主程序之前将压入的数据弹出到原来的工作单元中,恢复其原来的状态。由于堆栈操作是“后进先出”,因此,先压入堆栈的参数应该后弹出,才能保证恢复原来的状态。至于具体的子程序是否要进行现场保护以及对哪些对象保护应视具体情况而定。

例 7 在内部 RAM 的 30H 单元中存有两个十六进制数(0~F),试将其转换为 ASCII 码,并存放于 31H 和 32H 两个单元中。

这是一个十分重要的程序,对理解堆栈的使用方法十分有利。

```
      ;以下是主程序
      MOV SP, #5FH      ;设置堆栈指针(1)
MAIN: PUSH 30H          ;十六进制数进栈(2)
      ACALL SUBB        ;调用转换子程序(3)
      POP 31H           ;第 1 个转换结果送 asc 单元(4)
      MOV A, 30H        ;再取原十六进制数(5)
      SWAP A            ;高低半字节交换(6)
      PUSH ACC          ;交换后的十六进制数进栈(7)
      ACALL SUBB        ;调用转换子程序(8)
      POP 32H           ;第 2 个转换结果送 asc+1 单元(9)
      ;以下是子程序
SUBB: DEC SP            ;跨过断点保护内容(10)
      DEC SP            ;跨过断点保护内容(11)
      POP ACC           ;弹出转换数据(12)
      ANL A, #0FH       ;屏蔽高位(13)
      ADD A, #7          ;修改变址寄存器内容(14)
      MOVC A, @A+PC     ;查表(15)
      PUSH ACC          ;查表结果进栈(16)
```

```

INC SP          ;修改堆栈指针回到断点保护内容(17)
INC SP          ;修改堆栈指针回到断点保护内容(18)
RET             ;返回主程序(19)
TAB:  DB "0,1,2,3,4,5,6,7";以字符串的形式列出 ASCII 码表(20)
      DB "8,9,A,B,C,D,E,F"

```

下面对以上程序进行简要说明:

执行第(1)条语句时,堆栈指针 SP 指向 5FH,即 $(SP)=5FH$ 。

执行第(2)条指令时,堆栈指针 SP 的内容(5FH)先加 1,SP 指向 60H,然后,再将 30H 中的内容压入堆栈的 60H 中,即 $60H \leftarrow (30H)$ 。

执行第(3)条指令时,调用子程序,从第(10)条指令执行;另外,在执行子程序调用指令 ACALL 时,要将 2B 的断点地址压入到堆栈的 61H、62H 中。此时,堆栈指针 SP 指向 62H。

执行第(10)条、第(11)条指令,堆栈指针 SP 的内容(62H)减 2,此时,SP 指向 60H,目的是跨过断点保护内容。

执行第(12)条指令时,先将堆栈指针 SP 所指的 60H 单元中的内容(待转换的十六进制数)弹出到累加器 A 中,即 $A \leftarrow (60H)$,再将堆栈指针 SP 的内容(60H)减 1,此时,SP 指向 5FH。

执行第(13)条指令时,屏蔽高 4 位,即先转换低 4 位的十六进制数。

第(14)条指令用于修改变址寄存器的内容;

第(15)条指令是一个变址寻址指令,执行该条指令后,后面还有

PUSH ACC(2B)

INC SP(2B)

INC SP(2B)

RET(1B)

共 $2+2+2+1=7(B)$ 的指令,所以,在第(14)指令中要进行“加 7”处理,只有这样,才能查到所需要的数。

执行第(16)条指令的目的是将查表得到的结果压入堆栈,执行时,先将堆栈指针 SP 内容(5FH)加 1,即 SP 指向 60H。再将累加器 A 中的内容压入堆栈的 60H 中。

执行第(17)条、第(18)条指令,堆栈指针 SP 的内容(60H)加 2,此时,SP 指向 62H,目的是回到断点保护内容。

执行第(19)条 RET 返回指令时,将堆栈 61H、62H 中的 2B 断点地址弹出到 PC,返回到主程序,开始执行调用子程序的下一条指令,同时,堆栈指针 SP 的内容(62H)减 2,SP 指向 60H。本指令的操作可表示为

$PC_{15} \sim PC_8 \leftarrow (SP), SP \leftarrow (SP) - 1$

$PC_7 \sim PC_0 \leftarrow (SP), SP \leftarrow (SP) - 1$

执行第(4)条指令时,先将 60H 中的内容弹出,送到 31H 中,再将 SP 中的内容(60H)减 1,此时,堆栈指针又指向 5FH。

执行第(5)条~第(9)条指令的情况与以上基本相同,不再分析。

随便说明一下,第(20)条指令的 ASCII 码表,是以字符串形式列出十六进制数,但在

汇编时,则是以 ASCII 码形式写入存储单元的,因此读出来的是被转换数据的 ASCII 码。

重点提示 本程序的一个特点是堆栈的使用,这对于读者加深堆栈的概念十分有利。在本程序中,两种使用堆栈的方法都涉及到了,一种是通过堆栈传递数据,被转换的数据在主程序中进栈而在子程序中出栈,最后再把转换结果返回主程序;另一种使用方法是系统自动的,即调用子程序要用堆栈来保存断点。由于是被转换的数据在主程序中先进栈,而断点地址是在调用子程序时才进栈,为此在子程序中要取出转换数据,就得修改堆栈指针 SP,以指向该数据。

第五节 汇编语言实用程序举例

下面对汇编语言中应用十分广泛的定时程序和查表程序作一简要介绍。

一、定时程序

在单片机的控制应用中,常有定时的需要,如定时中断、定时检测和定时扫描等。定时功能除可以使用定时/计数器实现之外,还可以使用定时程序完成。定时程序是典型的循环程序,它是通过执行一个具有固定延迟时间的循环体来实现延时的。

1. 单循环定时程序

下面是一个最简单的单循环定时程序。

```
DELAY: MOV R3, # TIME      ;1 个机器周期 (1)
LOOP:  NOP                  ;1 个机器周期 (2)
        NOP                  ;1 个机器周期 (3)
        DJNZ R3, LOOP        ;2 个机器周期 (4)
        RET                  ;2 个机器周期 (5)
```

第(1)条指令 MOV R3, # TIME 是将循环次数 TIME 送到 R3 中,这条指令占用 1 个机器周期的时间;

第(2)条 NOP 是空指令,不进行任何操作,但它在程序存储器中占了 1B 的位置,执行时也需要占用 1 个机器周期的时间;

第(3)条也是空指令,占 1 个机器周期;

第(4)条 DJNZ 指令的机器周期为 2;

第(5)条指令 RET 的机器周期为 2;

从以上可以看出,第(1)条指令和第(5)条指令不参与循环,可忽略不计,定时程序每循环一次共占用 4 个机器周期(第(2)条~第(4)条指令)。由于 1 个机器周期是由 12 个振荡周期组成,若单片机的晶振频率为 12MHz,它时振荡周期是 $\frac{1}{12}\mu\text{s}$, 1 个机器周期是 $12 \times (1/12) = 1\mu\text{s}$ 。因此一次循环的延迟时间为 $4 \times 1 = 4\mu\text{s}$ 。定时程序的总延迟时间为 $4 \times \text{TIME}(\mu\text{s})$, TIME 是装入寄存器 R3 的时间常数, R3 是 8 位寄存器,因此这个程序的最长定时时间为

$$256 \times 4 = 1024(\mu\text{s}) \approx 1\text{ms}$$

2. 多循环定时程序

单循环定时程序的时间延迟比较小,为了加长定时时间,通常采用多重循环的方法。

下面就是一个采用双重循环的定时程序。

```
DELAY: MOV R5, # TIME1      ;1 个机器周期 (1)
LOOP2: MOV R4, # TIME2      ;1 个机器周期 (2)
LOOP1: NOP                  ;1 个机器周期 (3)
      NOP                  ;1 个机器周期 (4)
      DJNZ R4, LOOP1        ;2 个机器周期 (5)
      DJNZ R5, LOOP2        ;2 个机器周期 (6)
      RET                  ;2 个机器周期 (7)
```

以下分析中,设单片机的晶振频率为 12 MHz,则 1 个机器周期是 $1\mu\text{s}$ 。

第(1)条指令是传递数据。TIME1 是被传递的数,执行这条指令后,将数据 TIME1 送到 R5 中去。

第(2)条指令也是传递数据,执行后,R4 中的值是 TIME2。

第(3)条、第(4)条是空指令,各占 1 个机器周期。

第(5)条指令用于控制第(3)条~第(5)条指令的循环次数,执行过程是:将 R4 的值 (TIME2)减 1,然后看 TIME2 是否等于 0。如果等于 0,往下执行;如果不等于 0,则转移到标号为 LOOP1 的位置去执行。该条指令的最终执行结果是:第(3)条~第(5)条指令 (共 4 个机器周期)被执行 TIME2 次。

执行第(6)条指令时,由于 R5 中的值 TIME1 不为 0,所以减 1 后转去 LOOP2 标号处,即执行第(2)条指令“MOV R4, # TIME2”。这样,R4 中又被送入了 TIME2 这个数,然后再去执行第(3)条~第(5)条指令,最终的结果是第(3)条~第(5)条指令将被执行 $\text{TIME1} \times \text{TIME2}$ 次,第(2)条、第(6)条指令 (共 3 个机器周期)被执行 TIME1 次。这样,总定时时间约为

$$(4 \times \text{TIME1} \times \text{TIME2} + 3 \times \text{TIME1}) \times 1\mu\text{s}$$

最大定时时间约为

$$(4 \times 256 \times 256 + 3 \times 256) \times 1\mu\text{s} = 262912\mu\text{s} \approx 263\text{ms}$$

从以上分析中可以看出,无论是改变循环变量 (TIME1、TIME2) 的大小,还是改变循环程序中指令的数目 (如增加或减少 NOP 指令的个数),都可方便地对定时时间进行调整。

注意 在定时程序中,循环程序段的指令操作并无实际意义,只是起到调节机器周期的作用,通常把这些指令称之为哑指令,上述程序中的 NOP 指令就是一个典型的哑指令,此外,还可以加入其他指令作为哑指令。使用哑指令要注意:一是不要破坏有用存储单元的内容;二是不要破坏有用寄存器的内容;三是不要破坏有用标志位的状态。

例 8 设单片机晶振频率 6MHz,编写一段程序,大约延时 1s。

由于 DJNZ 指令最多的循环次数是 256 次,而 $1\text{s} \div 256 \div 256$ 约等于 $15\mu\text{s}$,晶振频率 6MHz,则 1 个机器周期为 $12 \times (1/6) = 2\mu\text{s}$,把内循环 (以下程序的第(3)~第(9)条指令)定为 8 个机器周期 ($16\mu\text{s}$),则 $16\mu\text{s} \times 250 \times 250 = 1\text{s}$ 。根据以上分析,所设计的定时程序如下:

```
DELAY: MOV R5, # 0FAH      ;1 个机器周期 (1)
LOOP2: MOV R4, # 0FAH      ;1 个机器周期 (2)
LOOP1: NOP                ;1 个机器周期 (3)
```

NOP	;1 个机器周期 (4)
NOP	;1 个机器周期 (5)
NOP	;1 个机器周期 (6)
NOP	;1 个机器周期 (7)
NOP	;1 个机器周期 (8)
DJNZ R4, LOOP1	;2 个机器周期 (9)
DJNZ R5, LOOP2	;2 个机器周期 (10)
RET	;2 个机器周期 (11)

实际定时时间约为

$$(8 \times 250 \times 250 + 3 \times 250) \times 2\mu\text{s} = 1001500\mu\text{s} \approx 1\text{s}$$

3. 延时程序的调用技巧

如果系统中有多个定时需要,可以先设计一个基本的延时程序,使其延迟时间为各定时时间的最大公约数,然后就以此基本程序作为子程序,通过调用的方法实现所需要的不同定时。例如,要求的定时时间分别为 2s、4s、8s 和 16s,并设计一个 1s 延时子程序 DELAY,则不同定时的调用情况表示如下:

```

MOV R0, #02H      ;2s 延时
LOOP1: LCALL DELAY
      DJNZ R0, LOOP1

```

```

MOV R0, #04H      ;4s 延时
LOOP2: LCALL DELAY
      DJNZ R0, LOOP2

```

```

MOV R0, #08H      ;8s 延时
LOOP3: LCALL DELAY
      DJNZ R0, LOOP3

```

```

MOV R0, #10H      ;16s 延时
LOOP4: LCALL DELAY
      DJNZ R0, LOOP4

```

二、查表程序

预先把数据以表格形式存放在程序存储器(ROM)中,然后使用程序读出,这种能读出表格数据的程序就称之为查表程序。查表操作对单片机的控制应用十分重要,因此 MCS-51 准备了专用的查表指令:

```

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

```

这两条 MOVC 指令的功能是完全相同的,它们的共同优点是能在不改变 PC 和 DPTR 的状态下,只根据 A 的内容就可以取出表格中的数据,这对于提高查表功能,缩短

程序长度极为重要。

1. 用 DPTR 作为地址的查表方法

用 DPTR 作为地址的查表方法是：

- (1)先将所查表格的首地址用 MOV 指令存入 DPTR 数据指针寄存器。
- (2)将访问项的偏移值(即在表中的位置是第几项),用 MOV 指令装入累加器 A 中。
- (3)用 MOVC A,@A+DPTR 指令读数,查表结果送回累加器中。

例 9 有一个数(取值范围为 0~9)在 R0 中,要求用查表的方法确定它的平方值。
程序如下：

```
MOV DPTR, #TAB      (1)
MOV A,R0              (2)
MOVC A,@A+DPTR      (3)
```

TAB: DB 0,1,4,9,16,25,36,49,64,81

DB 是一条伪指令,它的用途是将其后面的数,也就是 0、1、4、9、16、25、36、49、64 和 81 放在 ROM 中。这里的“放”不是在程序执行时,而是在程序被编译并写入芯片时就完成了。图 5-4 是查表指令存储器的示意图。

图 5-4 中的数在 ROM 中是顺序存放的,而 0 所在单元的地址就是 TAB,TAB 在这里只是一个符号,到了最终变成代码时(汇编时),TAB 就是一个确定的值,如 1FEH 或 2003H 等。

以下分析程序的执行情况。首先执行第(1)条指令,即将 TAB 送入 DPTR 中,然后执行第(2)条指令,取出欲查表的值,假设这个值是 2;接着执行第(3)条指令,将 DPTR 中的值(现在是 TAB)与 A 中的值相加,得到结果 TAB+2;然后以这个值为地址,到 ROM 中相应单元中去取数。查看图 5-4 中这个单元中的值是 4,正是 2^2 ,这样就获得了正确的结果。其他数据也可以类推。

方法技巧 这里,标号 TAB 的真实含义就是地址数值。在这里它代表了 0、1、4、9、16、25、36、49、64 和 81 这几个数据在 ROM 中存放的起点位置。单片机正是通过这个地址才找到这段程序的。
可以通过以下的例子再来看一看标号的含义：

```
MOV DPTR, #200H
MOV A,R0
MOVC A,@A+DPTR
:
ORG 200H
DB 0,1,4,9,16,25,36,49,64,81
```

如果 R0 中的值为 2,则最终地址 200H+2 为 202H,到 202H 单元中找到的是 4。那为什么前面的程序不这样写,而要用标号 TAB 呢?这是因为,如果这样,在写程序时,就必须确定这张表格在 ROM 中的具体的位置,如果写完程序后,又想在这段程序前插入一段程序,那么这张表格的位置就又要变了,即要改动 ORG 200H 这条指令了。由

地址	内容
...	...
TAB+9	81
TAB+8	64
TAB+7	49
TAB+6	36
TAB+5	25
TAB+4	16
TAB+3	9
TAB+2	4
TAB+1	1
TAB	0
...	...

图 5-4 查表指令存储器示意图

于经常需要修改程序,所以就用标号来替代,只要一编译程序,位置就自动发生变化,十分方便。

例 10 某智能化设备的键盘扫描程序中,可根据按下的键值(0,1,2,3,...,9),转换成对应的 2B 命令入口地址,设键值在 30H 单元中,转换后的入口地址存放在 31H(高 8 位)及 32H(低 8 位)。对应关系如表 5-2 所列。

表 5-2 键值与命令入口地址对应表

键值	0	1	2	3	4	5	6	7	8	9
入口地址	0100H	0110H	0120H	0130H	0140H	0150H	0160H	0170H	0180H	0190H

源程序如下:

```
MOV DPTR, # TAB      ;指向表首(1)
MOV A, 30H            ;取得键值(2)
RL A                  ;乘 2 作表偏移量(3)
MOV 30H, A            ;存偏移量(4)
MOVC A, @A+DPTR       ;查表得高 8 位地址(5)
MOV 31H, A            ;存高 8 位地址(6)
INC DPTR              ;指向入口低 8 位(7)
MOV A, 30H            ;取偏移量(8)
MOVC A, @A+DPTR       ;查表得低 8 位地址(9)
MOV 32H, A            ;存低 8 位地址(10)
RET

TAB: DW 0100H          ;键值 0 入口地址,高 8 位在前(低地址),低 8 位在
                        ;后(高地址)
      DW 0110H          ;键值 1 入口地址
      DW 0120H          ;键值 2 入口地址
      DW 0130H          ;键值 3 入口地址
      DW 0140H          ;键值 4 入口地址
      DW 0150H          ;键值 5 入口地址
      DW 0160H          ;键值 6 入口地址
      DW 0170H          ;键值 7 入口地址
      DW 0180H          ;键值 8 入口地址
      DW 0190H          ;键值 9 入口地址
```

以下分析程序的执行情况。首先执行第(1)条指令,即将 TAB 送入 DPTR 中,然后执行第(2)条指令,取出欲查表的值,假设这个值是 2(0000 0010B);接着执行第(3)条指令,左移 1 位,此时 A 中的值为 4(0000 0100B),第(4)条指令用于将 A 的值(现在为 4)暂存于 30H 中,执行第(5)条指令时,将 DPTR 中的值(现在是 TAB)与 A 中的值(现在是 4)相加,得到结果 TAB+4;然后以这个值为地址,到 ROM 中相应单元中去取数。查看图 5-5 所示的查表示意图可知:这个单元中的值是

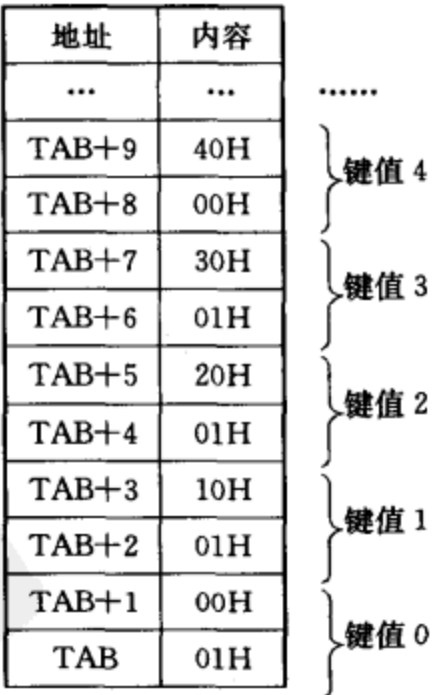


图 5-5 查表示意图

01H,第(6)条指令用于将取出的高8位地址(01H)存于31H中,第(7)条指令是将DPTR的值加1,此时,DPTR的值为TAB+1,第(8)条指令的作用是将暂存在30H中的值(此是为4)送到累加器A中,执行第(9)条指令时,将DPTR中的值(现在是TAB+1)与A中的值(现在是4)相加,得到结果TAB+5;然后以这个值为地址,到ROM中相应单元中去取数。这个单元中的值是20H,执行第(10)条指令后,将取出的低8位地址(20H)存于32H中。

0120H正是键值2的入口地址,这样就获得了正确的结果。其他数据也可以类推。

方法技巧 若用keil软件进行模拟仿真该程序,可在本程序最前面加入指令MOV 30H,#data,(#data可选0~9中的任一数字),给30H单元赋初值,并在本程序的最后加入END结束指令。

2. 用PC作为基地址的查表方法

当数据指针DPTR另有它用、所查表格为位于源程序邻近的较小表格(偏移量不超过256)时,采用以PC内容为基地址的查表指令是很有用的。使用方法是:

(1)首先将所查表中访问项的偏移值装入累加器A中。

(2)使用ADD A,#DATA指令,进行地址调整。DATA是查表指令MOVC A,@A+PC执行以后的地址(即该指令地址加1的值)到所查表的首地址之间的距离,即算出这两个地址之间其他指令所占的字节数,把这个结果作为DATA值。

(3)执行查表指令MOVC A,@A+PC进行查表,查表结果送到累加器A。

例11 以查表的方法将累加器A中的十六进制数转换为ASCII码,并送回累加器中。源程序如下:

```
ORG 2000H
ADD A,#01H
MOVC A,@A+PC
RET
DB 30H      ;以下是十六进制数的ASCII码表
DB 31H
DB 32H
:
DB 39H
DB 41H
DB 42H
DB 43H
DB 44H
DB 45H
DB 46H
```

程序中,ORG 2000H是伪指令,用于规定目标程序的起始地址为2000H,即指令ADD A,#01H(2B)的起始地址为2000H、2001H,由于MOVC A,@A+PC和RET都是1B指令,因此,MOVC A,@A+PC的地址号为2002H,RET的地址号为2003H,DB 30H的地址号为2004H,其他依次类推。

假设 A 中的十六进制数为 02H, 执行 ADD A, #01H 后, A 变为 03H, PC 内容(下一条指令的起始地址)为 2003H, 所以 $(A) + (PC) = 2006H$, 从 2006 单元中取数送 A, 则 $(A) = 32H$, 也就是说, 十六进制数 03H 的 ASCII 码值为 32H。查表之前作 A 加 1 是因为 MOVC 指令与数据表之间有一个地址单元的指令 RET。

注意 MOVC A, @A+DPTR 和 MOVC A, @A+PC 这两条指令在具体使用上有一些差异。前一条指令(MOVC A, @A+DPTR)的基址寄存器 DPTR 能提供 16 位基址, 而且还能在使用前给 DPTR 赋值, 因此查表范围可达整个程序存储器的 64KB 空间。数据表格的大小和位置可以在 64KB 程序存储器中任意安排, 一个数据表格可以被多个程序块公用, 使用起来比较方便, 而后一条指令(MOVC A, @A+PC)是以 PC 作为基址寄存器, 虽然也能提供 16 位基址, 但 PC 不能赋值, 所以其基址值是固定的。由于 A 的内容为 8 位无符号数, 因此只能在当前指令下面的 256 个地址单元范围内进行查表, 即数据表格只能放在该指令后面 256 个地址单元之内, 而且表格只能被本程序段所使用。

第六节 几个简单的实验

单片机是一门实践性非常强的学科, 只有通过做一系列的实验, 才能比较容易地领会单片机那些枯燥、难懂的术语。下面以第三章介绍的 AT89C51 实验开发板为例, 介绍几个简单、直观的实验, 来熟悉和理解单片机的指令及编程方法。

一、闪烁的发光管

1. 功能

把 P1 端口同时置高和置低, 使 P1 端口外接的 8 只 LED 灯不断闪烁。

2. 源程序

源程序如下:

```
ORG 0000H      ;程序开始
LJMP MAIN
ORG 030H
;以下是主程序
MAIN: MOV P1, #00H      ;把 P1 全部置低电平, P1 口外接的 8 只灯全亮(1)
      ACALL DELAY        ;延时(2)
      MOV P1, #0FFH      ;把 P1 全部置高电平, P1 口外接的 8 只灯全灭(3)
      ACALL DELAY        ;延时
      AJMP MAIN          ;重新开始(4)
;以下是 0.5s 延时子程序
DELAY: MOV R5, #0FAH     ;1 个机器周期
LOOP2: MOV R4, #0FAH     ;1 个机器周期
LOOP1: NOP               ;1 个机器周期
      NOP
      NOP
```

```

NOP
NOP
NOP
DJNZ R4, LOOP1      ;2 个机器周期
DJNZ R5, LOOP2      ;2 个机器周期
RET                  ;2 个机器周期
END

```

该程序比较简单,为便于分析,在重点程序语句后的注释中加入了(1)、(2)、(3)等标志。

第(1)条指令是把 P1 全部置低电平,由于 P1 口外接的 8 只灯的负极接 P1 口,正极接电源,因此,执行该指令后,P1 口外接的 8 只灯全亮。

第(2)条指令是调用延时子程序。因 AT89C51 实验开发板的晶振为 12MHz,所以,延迟时间约

$$(8 \times 250 \times 250 + 3 \times 250) \times 1\mu s = 501500\mu s \approx 0.5s$$

第(3)条 P1 全部置高电平,P1 口外接的 8 只灯全灭。

第(4)条指令是再回到标号 MAIN 处,使 8 只灯反复闪烁。


注意 如果将 AT89C51 实验开发 LED 的正端接 P1 口,而负极接地时,8 只 LED 会产生怎样的变化呢?除了明灭的顺序颠倒之外,亮度上也会变暗许多,这是因为由 AT89C2051 的 P1 输出是 1 时所送出来的电流较小(约 5mA),LED 当然就亮不起来,但 P1 输出 0 时它可以吸入多达 20mA 的电流,而 LED 要点到很亮也只要 10mA 的电流就够了。所以,正确的连接方法应该像实验开发板电路图一样接成负逻辑的状态,即低电平时 LED 点亮,高电平时 LED 熄灭,刚好跟我们的习惯相反。

3. 实验步骤

(1)打开 Keil 软件,输入上面的程序,保存为 exam1.asm。对程序进行编译、链接和调试,产生 exam1.hex 目标文件。

Keil 软件的使用方法见第三章有关内容。下面再简要介绍一下如何用 Keil 软件计算出延迟时间。即确定是否延时为 0.5s。

打开寄存器窗口 Regs 页中的 Sys 目录树,找到 sec 项,如图 5-6 所示。

sec 项记录了从程序开始执行到当前程序结束的时间(s)。点击工具栏上的  按钮以复位程序,sec 的值回零,按下 F10 键(过程单步),程序窗口中的黄色箭头指向“ACALL DELAY”行。此时,记录下 sec 值为 0.00000400,然后再按 F10,黄色箭头指向“MOV P1, #0FFH”行,此时延时程序执行完,再次查看 sec 的值为 0.50075900,两者相减大约是 0.5s,所以延时时间大致是正确的。

说明 调试之前,一定要将晶振频率设置为 12MHz(菜单 Project→Options for Target‘Target1’→Target 中)。

(2)先可用软件或硬件仿真器对源程序进行仿真,具体使用方法可参见第三章有关内容。然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电实验,会发现 P1 口外接的 8 只灯不断闪烁。事实上,凡以 P 开头的这 32 个引脚都是可以点亮灯的,也就是说:这 32 个引脚都可以作为输出使用,如果不用来点亮 LED,可以用来控制继电器其他的执行机构。

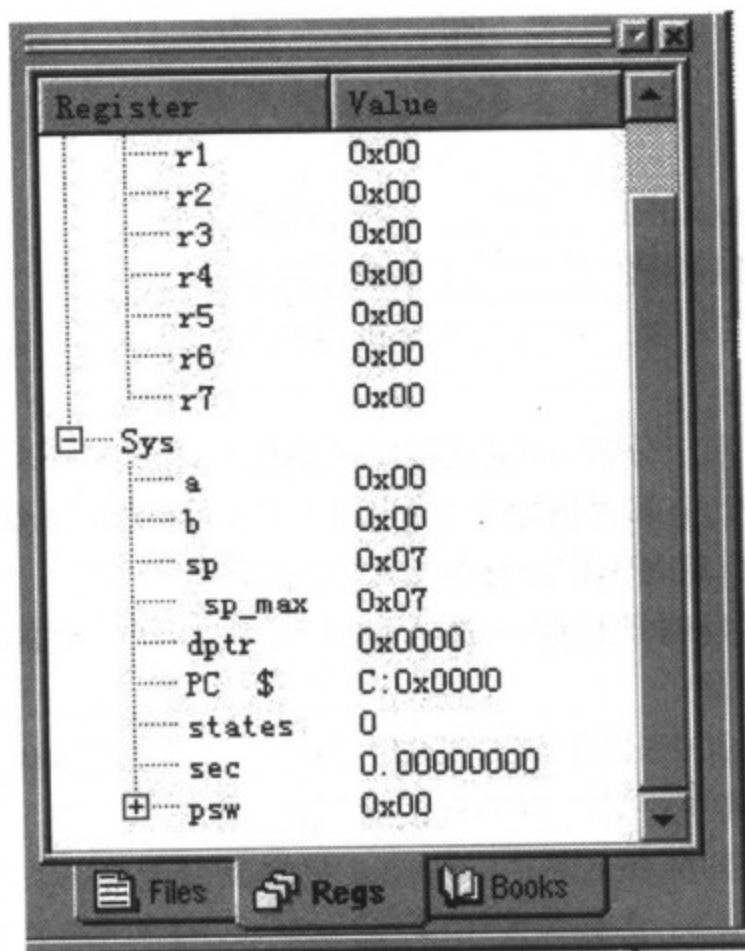


图 5-6 寄存器窗口

该实验程序在本书所附光盘的 example\ch_5\exam1 文件夹中。

二、8 路流水灯

1. 功能

把 P1 端口的依次置低,使 P1 端口外接的 8 只 LED 灯循环点亮。

2. 源程序

源程序如下:

```

    ORG 0000H
    LJMP MAIN
    ORG 30H
;以下是主程序
MAIN:MOV A, #0FEH
LOOP:MOV P1,A
    RL A
    LCALL DELAY
    LJMP LOOP
;以下是 0.5s 延时子程序
DELAY:MOV R5, #250
LOOP2:MOV R4, #250
LOOP1:NOP
    NOP
    NOP

```



```

NOP
NOP
NOP
DJNZ R4, LOOP1
DJNZ R5, LOOP2
RET
END

```

这段程序中的 RL A 是一条左移指令,将累加器 A 的值将循环左移 1 位,开始时, A 的值为 0FEH(111 1110B),执行一次左移指令时, A 中的值变为 1111 1101B,执行第 2 次时,变为 1111 1011B,也就是各位数字不断地向左移动,而最右 1 位由最左 1 位移入。送 1 个数,延迟 1 段时间,送完 8 个数后,从头开始循环。这 8 个数依次是:

1111 1110(0FEH)→1111 1101(0FDH)→11111011(0FBH)→1111 0111(0F7H)→
1110 1111(0EFH)→1101 1111(0DFH)→1011 1111(0BFH)→0111 1111(7FH)

3. 实验步骤

(1)打开 Keil 软件,输入上面的程序,保存为 exam2. asm。对程序进行编译、链接和调试,产生 exam2. hex 目标文件。

(2)先可用软件或硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电实验,会发现 P1 口外接的 8 只灯依次循环点亮并熄灭。

该实验程序在本书所附光盘的 example\ch_5\exam2 文件夹中。

三、用按键控制发光管

1. 功能

按下按键 K1, P1. 6 外接的灯亮,按下 K2, P1. 7 外接的灯亮。

2. 源程序

```

ORG 0000H
LJMP MAIN
ORG 30H
MAIN: MOV P3, #0FFH
LOOP: MOV A, P3
      MOV P1, A
      AJMP LOOP
      END

```

3. 实验步骤

(1)打开 Keil 软件,输入上面的程序,保存为 exam3. asm。对程序进行编译、链接和调试,产生 Exam3. hex 目标文件。

(2)用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电实验会发现,所有灯全部不亮,然后按下

一个 K1 按钮, P1.6 外接的灯亮, 再按下 K2, P1.7 外接的灯亮, 松开按钮灯就灭。有关实验电路如图 5-7 所示。

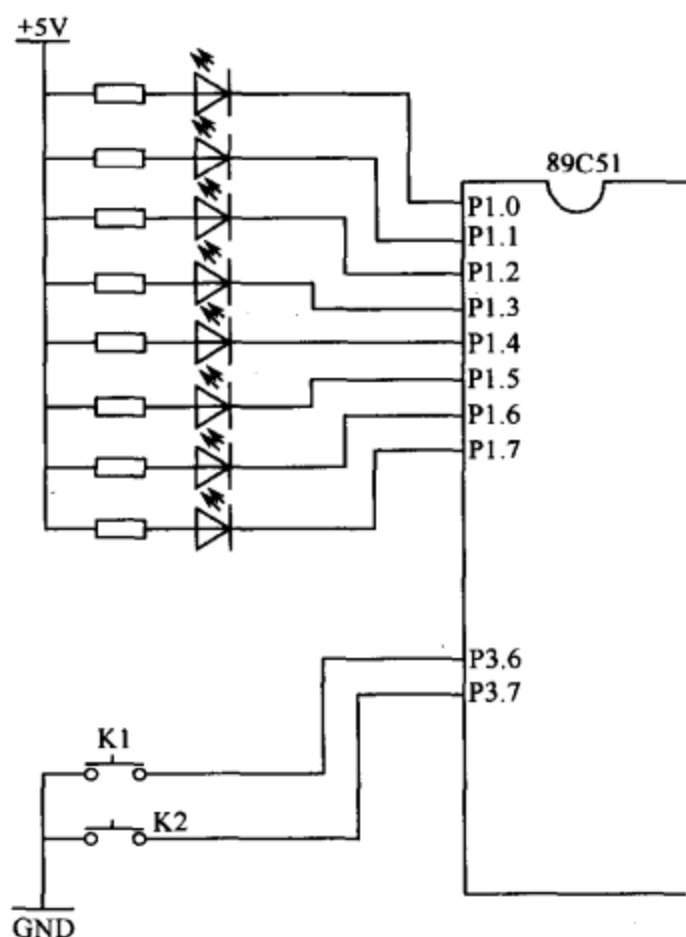


图 5-7 实验三电路

这个程序说明 P1 口(包括 P0、P2、P3 口)除了可以作为输出外,还可以作为输入口使用。不过,在作为输入读引脚时,应先写“1”。

该实验程序在本书所附光盘的 example\ch_5\exam3 文件夹中。

四、二进制加法运算

1. 功能

用实验开发板的 P1 口灯的亮和灭模拟二进制加法。

2. 源程序

源程序如下:

```

    ORG 0000H      ;程序开始
    LJMP MAIN
    ORG 030H
MAIN: ACALL DELAY
    MOV A, #052H   ;寄存器 A 送 52H
    MOV R0, #0FCH  ;寄存器 R0 送 0FCH
    ADD A, R0      ;相加
    NOP
    MOV P1, A      ;结果送 P1 口
    END
    
```

3. 实验步骤

(1)打开 Keil 软件,输入上面的程序,保存为 exam4. asm。对程序进行编译、链接和调试,产生 exam4. hex 目标文件。

(2)先可用软件或硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电实验,会发现,P1. 7~P1. 0 口外接的 8 只灯分别为亮、灭、亮、亮、灭、灭、灭、亮。这是因为:

源程序的两个数分别为 1111 1100B(FCH)和 0101 0010B(52H),相加后结果为

1←0100 1110(1←4EH),因相加后的数已经超过 256,所以进位 CY=1。将相加后的数送到 P1 口后,则 P1. 7~P1. 0 口外接的 8 只灯为亮、灭、亮、亮、灭、灭、灭、亮(对应相加后的数 0100 1110)。

该实验程序在本书所附光盘的 example\ch_5\exam4 文件夹中。

五、加 1 计数器

1. 功能

用 P1 口的 8 只灯循环显示十六进制 00H~FFH 对应的二进制数。

2. 源程序

源程序如下:

```
ORG 0000H      ;程序开始
LJMP MAIN
ORG 030H
;以下是主程序
MAIN:MOV A,#00H ;先送 0
PLAY:MOV P1,A   ;输出
      MOV R0,#04H;4×0.5s 延时
LOOP:LCALL DELAY
      DJNZ R0,LOOP
      INC A      ;加 1
      AJMP PLAY  ;再输出
;以下是 0.5s 延时子程序
DELAY:MOV R5,#250
LOOP2:MOV R4,#250
LOOP1:NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      DJNZ R4,LOOP1
```

```

DJNZ R5,LOOP2
RET
END

```

3. 实验步骤

(1)打开 Keil 软件,输入上面的程序,保存为 exam5. asm。对程序进行编译、链接和调试,产生 exam5. hex 目标文件。

(2)先可用软件或硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电实验,会发现,运行结果先输出 00000000 (P1 口的 8 只灯全亮),然后输出 00000001,然后 0000010,直到 11111111(P1 口的 8 只灯全灭)。分别对应十六进制的 00H~FFH。该实验的目标代码文件为 exam5. hex
该实验程序在本书所附光盘的 example\ch_5\exam5 文件夹中。

六、二进制乘法运算

1. 功能

用实验开发板的 P1 口灯的亮和灭模拟二进制数的乘法运算。

2. 源程序

源程序如下:

```

ORG 0000H           ;程序开始
LJMP MAIN
ORG 030H
;以下是主程序
MAIN:ACALL DELAY    ;调延时子程序
PLAY:MOV A,#0FFH    ;寄存器 A 送 0FFH
MOV B,#03H          ;寄存器 B 送 03H
MUL AB              ;相乘
MOV P1,B            ;高 8 位输出结果是 00000010
ACALL DELAY
MOV P1,A            ;低 8 位输出结果是 11111101
ACALL DELAY         ;此句不可少,否则会看不到低 8 位输出的现象
AJMP PLAY           ;再输出
;以下是 0.5s 延时子程序
DELAY:MOV R5,#250
LOOP2:MOV R4,#250
LOOP1:NOP
NOP
NOP
NOP
NOP

```



```

NOP
DJNZ R4,LOOP1
DJNZ R5,LOOP2
RET
END

```

3. 实验步骤

(1)打开 Keil 软件,输入上面的程序,保存为 exam6. asm。对程序进行编译、链接和调试,产生 exam6. hex 目标文件。

(2)先可用软件或硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电实验,运行结果先输出寄存器 B 中的高 8 位 0000 0010(0 对应 P1 口灯亮,1 对应 P1 口的灯灭),然后再输出寄存器 A 中的低 8 位 1111 1101。这是因为:

源程序是计算 0FFH 和 03H 的乘积,即十进制 $255 \times 3 = 765$,转移为二进制为 0000 0010 1111 1101

所以,高 8 位为 0000 0010,低 8 位为 1111 1101。

由于该数已经超过 256,所以 PSW 寄存器的溢出标志位 OV=1。

该实验程序在本书所附光盘的 example\ch_5\exam6 文件夹中。

七、逻辑运算

1. 功能

用实验开发板的 P1 口灯的亮和灭模拟逻辑运算。注意实验中 P1. 7~P1. 0 口输出 1 时灯灭,输出 0 时灯亮。

2. 源程序

源程序如下:

```

ORG 0000H      ;程序开始
LJMP MAIN      ;调主程序
ORG 030H
;以下是主程序
MAIN:MOV A, #03CH ;将 03CH(0011 1100)送 A
MOV R0, #0AAH   ;将 0AA(1010 1010)送 R0
CPL A           ;A 的内容取反
MOV P1, A       ;输出到 P1 结果为 11000011,逻辑非模拟
ACALL DELAY     ;调用延时便于观察
MOV P1, #0FFH   ;关断显示
MOV A, #0C3H    ;A 的值重新载入
ANL A, R0       ;A 与 R0 相与,逻辑与模拟
MOV P1, A       ;输出到 P1 结果为 1000 0010
ACALL DELAY     ;调用延时便于观察

```

```

MOV P1, #0FFH    ;关断显示
MOV A, #0C3H     ;A 的值重新载入
ORL A, R0        ;A 与 R0 相或, 结果为 11101011, 逻辑或模拟
MOV P1, A        ;输出到 P1
ACALL DELAY      ;调用延时便于观察
MOV P1, #0FFH    ;关断显示
MOV A, #0C3H     ;A 的值重新载入
XRL A, R0        ;A 与 R0 相异或, 结果为 01101001, 模拟异或逻辑
MOV P1, A        ;输出到 P1
ACALL DELAY      ;调用延时便于观察
MOV P1, #0FFH    ;关断显示
AJMP MAIN        ;重新开始
;以下是 0.5s 延时子程序
DELAY: MOV R5, #250
LOOP2: MOV R4, #250
LOOP1: NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      DJNZ R4, LOOP1
      DJNZ R5, LOOP2
      RET
      END

```

3. 实验步骤

(1) 打开 Keil 软件, 输入上面的程序, 保存为 exam7.asm。对程序进行编译、链接和调试, 产生 exam7.hex 目标文件。

(2) 先可用软件或硬件仿真器对源程序进行仿真, 然后用 RF-810 编程器对 AT89C51 芯片编程。

(3) 将 AT89C51 芯片插入实验开发板, 通电观察实验结果。

该实验程序在本书所附光盘的 example\ch_5\exam7 文件夹中。

第六章 中断、定时/计数器和串行数据通信

中断就是打断正在执行的工作,转去做另外一件事,利用中断功能,不但可提高 CPU 的效率,实现实时控制,而且还可以对一些难以预料的情况进行及时处理;单片机的定时/计数器主要用于定时或延时控制,以及对外部事件进行检测、计数等;串行数据通信是指单片机与其他设备之间以及单片机之间数据的串行传送。这 3 部分内容既相互独立,又互相联系,因此,本章将其安排在一起进行介绍。

第一节 中断系统及实验

一、中断概述

1. 什么是中断

什么是中断,我们从一个生活中的例子引入。你正在家中看书——突然电话铃响了——你在书上做个记号——去接电话,和来电话的人交谈——门铃响了——你让打电话的对方稍等——你去开门,并在门旁与来人交谈——谈话结束,关好门——回到电话机旁继续通话——通话完毕,放下电话——从做记号的地方继续看书。

这是一个很典型的中断现象,就是正常的工作过程被外部的事件打断了。从看书到接电话,是一次中断过程,而从打电话到与门外来人交谈,则是在中断过程中发生的又一次中断,即所谓中断嵌套。为什么会发生上述的中断现象呢?就是因为在一个特定的时刻,面对着 3 项任务:看书、打电话和接待来人。但一个人又不可能同时完成 3 项任务,因此只好采用中断方法穿插着去做。

此种现象同样也出现在单片机中。在单片机中,要求 CPU 能及时地响应被控对象提出的分析、计算和控制等请求,使被控对象保持在最佳工作状态,以达到预定的控制效果。由于这些控制参量的请求都是随机发出的,而且要求单片机必须作出快速响应并及时处理,对此,只有靠中断技术才能实现。

在中断系统中,通常将 CPU 正常情况下运行的程序称为主程序,把引起中断的设备或事件称为中断源。由中断源向 CPU 所发出的请求中断的信号称为中断请求信号, CPU 接受中断申请终止现行程序而转去为服务对象服务称中断响应,为服务对象服务的程序称为中断服务程序(也称中断处理程序)。现行程序中断的地方称为断点,为中断服务对象服务完毕后返回原来的程序称为中断返回。整个过程称为中断。

2. 中断的优点

采用中断技术具有以下优点:

(1)实行分时操作,提高了 CPU 的效率。只有当服务对象向 CPU 发出中断申请时, CPU 才转去为它服务,这样,利用中断功能,就可以同时为多个服务对象服务,从而大大

提高了 CPU 的效率。

(2)实现实时处理。利用中断技术,各服务对象可根据需要,随时向 CPU 发出中断申请,CPU 可以及时发现和处理中断请求并为之服务,以满足实时控制要求。

(3)进行故障处理。单片机在运行过程中突然发生的硬件故障、运算错误及程序故障等,可以通过中断系统由故障源向 CPU 请求中断,由 CPU 转到相应的故障处理程序进行处理。

二、中断源

向 CPU 发出中断请求的来源称之为中断源。MCS-51 有 3 类共 5 个中断源,即 2 个外中断 INT0 和 INT1(由 P3.2 和 P3.2 引入)、2 个定时中断(定时器 T0、定时器 T1)和 1 个串行中断,其中,定时中断和串行中断属于内中断。每个中断源对应一个中断标志位,当某个中断源有申请时,相应的中断标志位置 1,外中断和定时中断源的中断标志位在 TCON 中,串行中断源的中断标志位在 SCON 中。

1. 外中断

外中断是由外部信号引起的,共有 2 个中断源,即外部中断“0”和外部中断“1”。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,即电平方式和脉冲方式。可通过设置有关控制位进行定义。

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

而脉冲方式的中断请求则是脉冲的后沿负跳有效。这种方式下,CPU 在两个相邻机器周期对中断请求引入端进行的采样中,如前一次为高电平,后一次为低电平,即为有效中断请求。因此在这种中断请求信号方式下,中断请求信号的高电平状态和低电平状态都要至少维持一个机器周期,以确保电平变化能被单片机采样到。

2. 定时中断

定时中断是为满足定时或计数的需要而设置的。在单片机内部,有两个定时/计数器,通过对其中的计数结构进行计数,来实现定时或计数功能。当计数结构发生计数溢出时,即表明定时时间到或计数值已满,这时就以计数溢出信号作为中断请求,去置位一个溢出标志位,作为单片机接受中断请求的标志。由于这种中断请求是在单片机芯片内部发生的,因此无需在芯片上设置引入端。

3. 串行中断

串行中断是为串行数据传送的需要而设置的。每当串行接收或发送完一组串行数据时,就产生一个中断请求。因为串行中断请求也是在单片机芯片内部自动发生的,所以同样不需在芯片上设置引入端。

三、中断控制

在 80C51 单片机中,有 4 个寄存器是供用户对中断进行控制的,这 4 个寄存器分别是定时器控制寄存器 TCON、串行口控制寄存器 SCON、中断允许控制寄存器 IE 以及中断优先控制寄存器 IP。这 4 个控制寄存器都属于专用寄存器,可完成中断请求标志寄存、

中断允许管理和中断优先级的设定。由它们所构成的中断系统如图 6-1 所示。

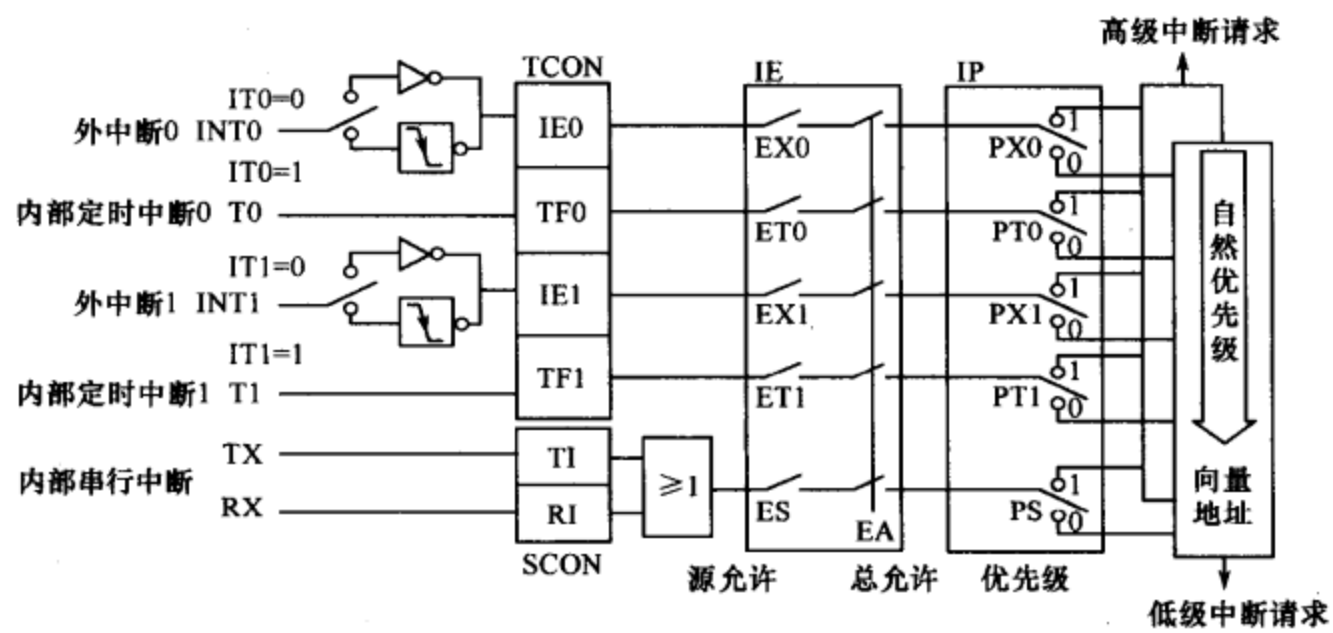


图 6-1 中断系统的结构

1. 定时器控制寄存器 TCON

定时器控制寄存器 TCON 用于保存外部中断请求以及定时器的计数溢出。寄存器地址 88H, 位地址 8FH~88H。寄存器的内容及位地址表示如下：

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

TCON 寄存器既有定时/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，对控制寄存器中的每一位要从含义、功能和置位/清除的方式（硬件方式还是软件方式）等方面来进行理解。

1) IE0(IE1)——外部中断请求标志位

当 CPU 采样到 $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$) 端出现的有效中断请求信号时，此位由硬件置 1，在中断响应完成后转向中断服务子程序时，再由硬件自动清 0。

2) IT0(IT1)——外部中断触发方式控制位

IT0(IT1)=1, 脉冲触发方式，下降沿触发有效。

IT0(IT1)=0, 电平触发方式，低电平有效。

此位由软件置位或清除。

3) TF0(TF1)——内部定时/计数器 0(定时/计数器 1)溢出标志位

当片内定时/计数器 0(定时/计数器 1)产生计数溢出时，TF0(TF1)由硬件置 1。当转向中断服务时，再由硬件自动清 0。

此外还有 2 位(TR0 和 TR1)，在后面介绍定时/计数器时再作介绍。

2. 串行口控制寄存器 SCON

串行口控制寄存器 SCON 地址 98H, 位地址 9FH~98H, 具体格式如下：

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

与中断有关的控制位共两位，即 TI 和 RI。

TI 是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，表示串行

口发送器正在向 CPU 申请中断;CPU 响应发送器的中断请求,转向执行中断服务程序时,不会自动清零 TI,必须由用户在中断服务程序中用 CLR TI 等指令清零。

RI 是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;在转向中断服务程序后,用软件清 0。

TI 和 RI 由逻辑或得到,也就是说,无论是发送标志还是接收标志,都产生串行中断请求。

3. 中断允许控制寄存器 IE

计算机中断系统中有两种不同类型的中断:一类称为非屏蔽中断,另一类称为可屏蔽中断。所谓非屏蔽中断,是指用户不能用软件方法加以禁止,一旦有中断申请,CPU 必须予以响应。对于可屏蔽中断,用户则可以通过软件方法来控制是否允许某中断源的中断。

从前面的图 6-1 中可以看出,80C51 单片机的 5 个中断源都是可屏蔽中断,CPU 对中断源的中断开放(允许)或中断屏蔽(禁止)是通过中断允许寄存器 IE 设置的,IE 既可按字节地址寻址,其字节地址为 A8H,又可按位寻址,位地址 AFH~A8H,具体格式如下:

位地址	0AFH	0AEH	0ADH	0ACH	0ABH	0AAH	0A9H	0A8H
位名称	EA	—	—	ES	ET1	EX1	ET0	EX0

1)EA——中断允许总控制位

EA=0,中断总禁止,关闭所有中断,由软件设置。

EA=1,中断总允许,总允许后,各中断的禁止或允许由各中断源的中断允许控制位进行设置。

2)EX0(EX1)——外部中断允许控制位

EX0(EX1)=0,禁止外中断 0(外中断 1)。

EX0(EX1)=1,允许外中断 0(外中断 1)。

3)ET0(ET1)——定时中断允许控制位

ET0(ET1)=0,禁止定时中断 0(定时中断 1)。

ET0(ET1)=1,允许定时中断 0(定时中断 1)。

4)ES——串行中断允许控制位

ES=0,禁止串行中断。

ES=1,允许串行中断。

可见,80C51 单片机通过中断允许控制寄存器对中断的允许实行两级控制。以 EA 位作为总控制位,以各中断源的中断允许位作为分控制位。当总控制位为禁止时,不管分控制位状态如何,整个中断系统为禁止状态;当总控制位为允许时,才能由各中断源的分控制位设置各自的中断允许与禁止。单片机复位后,(IE)=00H,因此,整个系统处于禁止状态。

说明 单片机在中断响应后不会自动关闭中断。因此,在转向中断服务程序后,应使用有关指令禁止中断,即用软件方式关闭中断。

4. 中断优先级控制寄存器 IP

设想一下,我们正在看书,电话铃响了,同时又有人按了门铃,你该先做哪样呢? 如果

你正在等一个很重要的电话，一般不会去理会门铃的；而反之，你正在等一个重要的客人，则可能就不会去理会电话了。如果不是这两者（即不等电话，也不是等人上门），你可能会按你通常的习惯去处理。总之这里存在一个优先级的问題，单片机中也是如此，也有优先级的问題。80C51 的中断优先级控制比较简单，只有高低两个优先级。当多个中断源同时申请中断时，CPU 首先响应优先级最高的中断请求，在优先级最高的中断处理完了之后，再响应级别较低的中断。

80C51 单片机各中断源的优先级由优先级控制寄存器 IP 进行设定（软件设置）。

IP 寄存器地址 B8H，位地址 BFH~B8H，具体格式如下：

位地址	0BFH	0BEH	0BDH	0BCH	0BBH	0BAH	0B9H	0B8H
位名称	—	—	—	PS	PT1	PX1	PT0	PX0

PX0(PX1)是外中断 0(外中断 1)优先级设定位；

PT0(PT1)是定时中断 0(定时中断 1)优先级设定位；

PS 是串行中断优先级设定位。

各位为 0 时，为低优先级；各位为 1 时，为高优先级。

80C51 中断优先级的控制原则是：

- (1)低优先级中断请求不能打断高优先级的中断服务；反之，则可以，从而实现中断嵌套。
- (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 CPU 查询次序确定哪个中断请求被响应。从高到低依次为

外部中断 0→定时中断 0→外部中断 1→定时中断 1→串行中断

例如，某单片机应用系统需使用 3 个中断：外部中断 0，定时中断 0 和串行口中断，若要求它们的优先级次序为定时中断 0 优先级最高、外部中断 0 次之、串行口中断最低，则只要把中断优先级控制寄存器的 PT0 位置 1，就可以实现该系统对优先级次序的要求。

方法技巧 上面所讲的 4 个寄存器都是为用户需要而设置的，因此在采用中断方式时，要在程序初始化时进行设置，外中断初始化主要有中断总允许、外中断允许、中断方式和中断优先级设定。而定时中断则没有中断方式控制。

例如，假定要开放外中断 1，采用脉冲触发方式，则需要做如下工作：

设置中断允许位——SETB EA(中断总允许)；SETB EX1(外中断 1 允许)。

设置中断请求信号方式——SETB IT1(脉冲触发方式)。

设置优先级——SETB PX1(外中断 1 优先级最高)。

四、中断的响应

中断响应就是单片机 CPU 对中断源提出的中断请求的接受。中断请求被响应后，再经过一系列的操作，而后转向中断服务程序，完成中断所要求的处理任务。下面简要说明 80C51 的中断响应过程。

1. 外中断采样和内中断置位

1)外中断采样

要想知道外中断是否有请求发生，需要对外中断进行采样。

当通过软件将寄存器 TCON 的 IT0(或 IT1)位设置为 0 时, $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$) 为电平触发方式, CPU 在每个机器周期的 S5P2(第 5 状态第 2 拍节)期间对 $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$) 采样, 一旦在 P3.2(或 P3.3)上检测到低电平时, 则认为有外中断申请, 随即由硬件使 TCON 的 IE0(或 IE1)位置 1, 向 CPU 申请中断。在中断响应完成后转向中断服务子程序时, 再由硬件自动对 IE0(或 IE1)位清 0。

当寄存器 TCON 的 IT0(或 IT1)位为 1 时, $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$) 为脉冲触发方式, 则 CPU 在每个机器的 S5P2 期间对 $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$) 采样, 当检测到前一周期为高电平、后一周期为低电平时, 由硬件使 TCON 的 IE0(IE1)位置 1, 向 CPU 申请中断, 在中断响应完成后转向中断服务子程序时, 再由硬件自动对 IE0(IE1)位清 0。在边沿触发方式中, 为保证 CPU 在两个机器周期内检测到由高到低的负跳变, 高电平与低电平的持续时间不得少于一个机器周期的时间。

2) 内中断置位

80C51 把所有中断标志都集中到 TCON 和 SCON 寄存器中。其中外中断是使用采样的方法把中断请求锁定在 TCON 寄存器的 IE0(IE1)标志位上, 而定时中断和串行中断的中断请求由于都发生在芯片的内部, 定时中断可以直接去置位 TCON 的 TF0(TF1), 串行中断可以直接去置位 SCON 的 RI 和 TI。内中断不存在采样问题。

2. 中断查询

所谓查询, 就是由 CPU 测试 TCON 和 SCON 中各标志位的状态, 以确定有没有中断请求发生以及是哪一个是中断请求。80C51 单片机是在每一个机器周期的最后一个状态(S6), 按优先级顺序对中断请求标志位进行查询, 即先查询高级中断后再查询低级中断, 同级中断按“外部中断 0→定时中断 0→外部中断 1→定时中断 1→串行中断”的顺序查询。如果查询到有标志位为“1”, 则表明有中断请求发生, 接着就从相邻的下一个机器周期的 S6 状态开始进行中断响应。

由于中断请求是随机发生的, CPU 无法预先得知, 因此在程序执行过程中, 中断查询要在指令执行的每个机器周期中不停地重复进行。换句话说, 就相当于你在看书的时候, 每一秒钟都会抬起头来听一听, 看一看, 是不是有人按门铃, 是否有电话, 烧的开水是否开了……看来, 单片机比人蠢多了。

3. 中断响应

中断响应就是对中断源提出的中断请求的接受, 是在中断查询之后进行的, 当查询到有效的中断请求时, 紧接着就进行中断响应。中断响应时, 根据寄存器 TCON、SCON 中的中断标记, 由硬件自动生成一条长调用指令 LCALL XXXX, 这里的“XXXX”就是程序存储器中断区中相应中断的入口地址。对于 80C51 的 5 个独立中断源, 这些入口地址已由系统设定。这样在产生了相应的中断以后, 就可转到相应的位置去执行, 就像听到电话铃、门铃就会分别到电话机、门边去一样。80C51 中 5 个独立中断源所对应的向量地址如下:

外中断 0	0003H
定时器 0	000BH
外中断 1	0013H
定时器 1	001BH

串口

0023H

它们的自然优先级由高到低排列。

例如,对于外部中断 0 的响应,产生的长调用指令为

LCALL 0003H

生成 LCALL 指令后,紧接着就由 CPU 执行,首先将当前程序计数器 PC 的内容(准备执行的指令的地址)压入堆栈以保护断点,再将中断入口地址装入 PC,使程序转向相应的中断区入口地址。从中断源所对应的向量地址中可以看出,一个中断向量入口地址到下一个中断向量入口地址之间(如 0003H~000BH)只有 8 个单元。也就是说,中断服务程序的长度如果超过了 8B,就会占用下一个中断的入口地址,导致出错。但一般情况下,很少有一段中断服务程序只占用少于 8B 的情况,为此可以在中断入口处写一条“LJMP XXXX”或“AJMP XXXX”指令,这样可以把实际处理中断的程序放到 ROM 的任何一个位置。

例如,若采用外中断 0,在程序的开始处可以这样写:

```
ORG 0000H
LJMP MAIN
ORG 0003H
LJMP INT_0
;以下是主程序
MAIN:
:
;以下是外中断 0 服务程序
INT_0:
:
    RETI
END
```

中断服务程序完成后,一定要执行一条 RETI 指令,执行这条指令后,CPU 将会把堆栈中保存着的地址取出,送回 PC,那么程序就会从主程序的中断处继续往下执行了。

说明 CPU 所做的保护工作是很有限的,只保护了一个地址(主程序中断处的地址),而其他的所有东西都不保护,所以如果你在主程序中用到了如 A、DPTR、PSW 等,在中断程序中又要用它们,还要保证回到主程序后这里面的数据还是没执行中断以前的数据,就得自己保护起来。

依据 80C51 手册上的写法,CPU 会在机器周期的 S5P2 阶段读入中断标志,并在下一个机器周期中检查,如果中断条件成立时,系统会自行产生一个 LCALL 到相对应的中断服务例程中,可是如果有下面 3 种情况时,系统是不会对中断要求信号有反应的:

(1)有相等或更高级的中断正在执行中,这与处理突发事情的状况相同,既然已经在处理突发情况,当然就不再接受其他中断条件,除非接下来的中断情形的优先权比较高。

由此得到一个观念:所有的中断程序都应该尽量简捷,一处理完中断事项后立即回主程序,才不会占用过多时间,进而影响系统的性能。

(2)目前的机器周期不是该指令的最后一个周期,由于 80C51 在指令执行时,分别有 1 个、2 个和 4 个机器周期之分,也就是说,必须完全执行完此指令后,系统对中断信号才会有所反应。比方说,当系统正在执行 MUL AB 指令(需花 4 个机器周期)时,中断信号必须出现在第 4 个机器周期上才算有效。这也就意味着,中断信号必须持续足够长的时间,以便 80C51 的 CPU 有时间去反应。

(3)若正在执行的指令为 RETI 或者是关于中断设置 IE、IP 的指令时,对正好出现的中断信号不反应,因为上述的情况刚好是某个中断服务程序的结束,或是允许/禁止某个中断的指令,当然是等到这些指令执行完毕后,才会对中断信号有所反应,这些指令最多占用两个机器周期的时间,所以这时的中断信号必须保持有两个机器周期以上的时间,才能被 80C51 接受。

五、中断的撤除

中断响应后,TCON 或 SCON 中的中断请求标志应及时清除。否则就意味着中断请求仍然存在,弄不好就会造成中断的重复查询和响应,因此就存在一个中断请求的撤除问题。

1. 定时中断请求的撤除

定时中断响应后,硬件自动把标志位 TF0(或 TF1)清 0,因此定时中断的中断请求是自动撤除的,不需要用户干预。

2. 串行中断软件撤除

对于串行中断,CPU 响应中断后,没有用硬件清除它们的中断标志 RI、TI,必须在中断服务程序中用软件清除,以撤除其中断请求。

3. 外中断请求的撤除

外部中断的撤除包括中断标志位 IE0(或 IE1)的清 0 和外中断请求信号的撤除。其中,IE0(或 IE1)清“0”是在中断响应后由硬件电路自动完成的。剩下的只是外中断引脚请求信号的撤除了。下面对脉冲和电平两种触发方式分别进行讨论。

(1)脉冲方式外部中断请求的撤除。对于脉冲方式的中断请求,由于脉冲信号过后就消失了,也可以说中断请求信号是自动撤除的。

(2)电平方式外部中断请求的撤除。对于电平方式的外部中断,中断标志的撤除是自动的,但中断请求信号的低电平可能继续存在,在以后机器周期采样时,又会把已清 0 的 IE0 或 IE1 标志位重新置 1。为此,要彻底解决电平方式外中断的撤除,除了标志位清 0 之外,必要时还需在中断响应后把中断请求信号引脚从低电平强制改变为高电平,为此,可在系统中增加如图 6-2 所示的电路(以外中断 0 为例)。

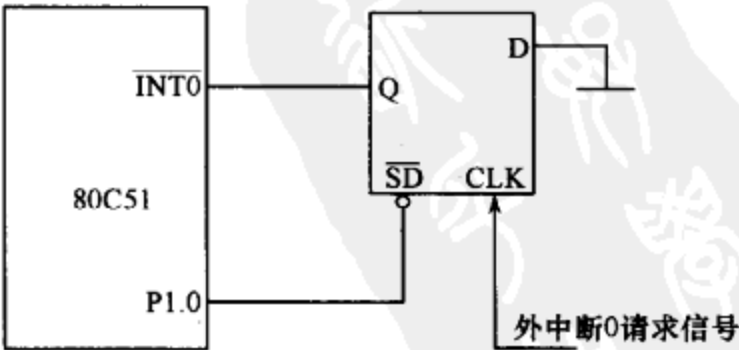


图 6-2 外中断请求标志撤除电路

从图 6-2 中可以看出,外部中断 0 请求信号加在 D 触发器(可选用 74LS74)的时钟输入端。当外部设备有中断请求信号(为低电平)出现时,Q 端输出为低电平,INT0 有效,向 CPU 发出中断请求信号。

CPU 响应中断后,在中断服务程序中由软件安排 1 个低电平中断应答信号,从 P1.0 送至 D 触发器的 \overline{SD} (置位端,低电平有效),使 D 触发器的 Q 端输出为高电平,从而撤除了低电平的外中断 0 请求信号。 \overline{SD} 端所需的低电平可通过在中断服务程序中增加以下指令得到:

ANL P1, #0FEH ;使 P1.0 输出为低电平,D 触发器置位

在中断服务程序中还要加上撤除外中断 0 标志指令,即

CLR IE0 ;清外中断标志,以便下次可再次中断

可见,电平方式外部中断请求信号的撤除是通过软、硬件相结合的方法实现的。

六、中断程序设计及实验

为实现中断而设计的有关程序称为中断程序。中断程序由中断控制程序和中断服务程序两部分组成。中断控制程序用于实现对中断的控制,中断服务程序用于完成中断源所要求的各种操作。从程序所处位置看,中断控制程序在主程序中,作为主程序的一部分并和主程序一起运行,中断服务程序则存放在主程序之外的其他存储区,只是在主程序运行过程中发生中断时,CPU 才暂停主程序执行,转而去执行中断服务程序。中断服务完毕之后,还得再转回来继续执行主程序。

1. 中断控制程序

中断控制程序也称中断初始化程序。要使 CPU 在执行主程序的过程中能够响应中断,就必须先对中断系统进行初始化。80C51 单片机中断系统初始化包括设置堆栈、选择中断触发方式(对外部中断而言)、开中断、设置中断优先级等。

系统复位或加电后,堆栈指针总是初始化为 07H,使得堆栈区实际上是从 08H 单元开始,但由于 08H~1FH 单元属于工作寄存器区,考虑到程序设计中经常要用到这些区,故常在中断控制程序中设置一条指令,使 SP 的值改为 1FH 或更大些,即

MOV SP, #5FH

另外,系统复位后,寄存器 TCON、SCON、IE、IP 等均复位为 00H,也需要根据中断控制的要求,在中断控制程序中对这些寄存器进行设置。此外,后面将要介绍的定时/计数器的启动、初始值的预置、工作方式的设定等往往也是在中断控制程序中完成的。

2. 中断服务程序

中断服务程序也称为中断处理服务,在设计时主要考虑以下因素:

(1)一个中断服务程序的入口地址到下一个中断向量入口地址之间(如 0003H~000BH)只有 8 个单元。而一般服务程序长度都会超过 8B,这样就必须在程序的开始安排一条跳转指令,把中断服务程序放到 ROM 的合适位置。

(2)由于 CPU 执行完中断服务程序后仍要返回主程序,因此,在执行中断服务程序之前,要将主程序中断处的地址保存起来,称为“保护断点”。又由于 CPU 执行中断处理程序时可能要使用主程序中使用过的累加器及其他寄存器甚至一些标志位,因此,在为中断源服务之前,也要将有关的寄存器内容保存起来,称为“保护现场”。而 CPU 为中断源服务完后,还必须要恢复原寄存器的内容及原程序中断处的地址,即要“恢复断点”和“恢复现场”。

在这里,保护现场和恢复现场是通过在中断服务程序中采用堆栈操作指令 PUSH 及 POP 实现的,而保护断点、恢复断点是由 CPU 响应中断和中断返回时自动完成的。

在使用 PUSH 和 POP 指令“保护现场”和“恢复现场”时,一方面要注意堆栈操作的“后进先出”原则,另一方面,要注意 PUSH 和 POP 指令必须成对使用。否则,可能会使保存在堆栈中的数据丢失,或使中断不能正确返回。此外,只有那些在中断服务程序中要使用的寄存器内容才需要加以保护。

从 CPU 中止当前程序,转向另一程序这点看,中断过程很像子程序,区别在于,中断发生时间一般是随机的,而子程序调用是按程序进行的。

(3)高优先级中断源的中断禁止。单片机具有两级中断优先级,可实现两级中断嵌套。高优先级的中断请求可以中断低优先级的中断处理。但是,对于某些不允许被中断的服务程序来说,也可以在 CPU 响应中断后用 CLR 指令(或其他指令)对 IE 寄存器某些位清 0,来禁止相应高优先级中断源的中断。

3. 中断服务程序设计举例及实验

中断的应用十分广泛,在后续内容中还要详细介绍,这里以第三章介绍的下载型实验板为例,介绍两个外中断的实例。

实验 1 在下载型实验板上,P3.2($\overline{\text{INT0}}$)上接有一只开关 K1,用 K1 按钮来模拟外部中断 0 的输入信号,并用 P1 口外接的 LED 作为中断响应。要求将外中断 0 触发方式设置为下降沿触发。按一次按键 K1 引发一次外中断 0,将 P1 口各位送 0,P1 口外接的灯全亮。有关电路如图 6-3 所示。

根据上述要求,设计的源程序如下:

```
ORG 0000H
AJMP MAIN          ;转主程序
ORG 0003H          ;外部中断 0 地址入口
AJMP INT_0         ;转外中断 0 服务程序
;以下是主程序
ORG 0200H
MAIN: MOV SP, #5FH  ;初始化堆栈
      MOV P1, #0FFH ;灯全灭
      SETB P3.2     ;P3.2 置高电平
      SETB IT0      ;外中断 0 置脉冲下降沿触发
      SETB EA       ;开总中断
      SETB EX0      ;开外部中断 0
      HERE: SJMP HERE ;跳转到本行
;以下是中断服务程序
INT_0: MOV P1, #00H ;P1 口外接的灯全亮
      RETI          ;中断返回
      END
```

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 int0.asm。对程序进行编译、链接和调

试,产生 int0. hex 目标文件。

(2)用硬件仿真器对源程序进行仿真,观察实验现象。

该实验程序在本书所附光盘的 example\ch_6\int0 文件夹中。

实验 2 在下载型实验板上,单片机的 P1.0~P1.7 接 8 只 LED 指示灯。在主程序中设计程序让 8 个指示灯轮流亮,在外部中断 0 服务程序中让 P1 口外接的 LED 同时闪烁 8 次,在外部中断 1 服务程序中让左右 4 个 LED 交替闪烁,要求外中断 0 和外中断 1 不互相影响。有关电路如图 6-4 所示。

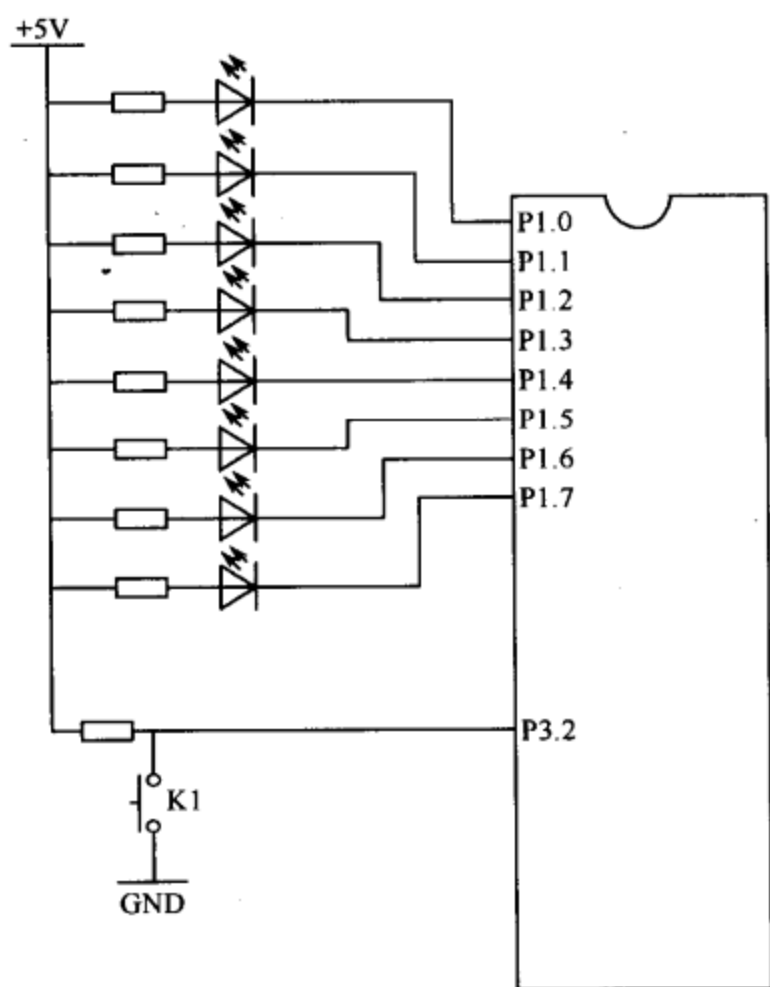


图 6-3 外中断 0 实验电路

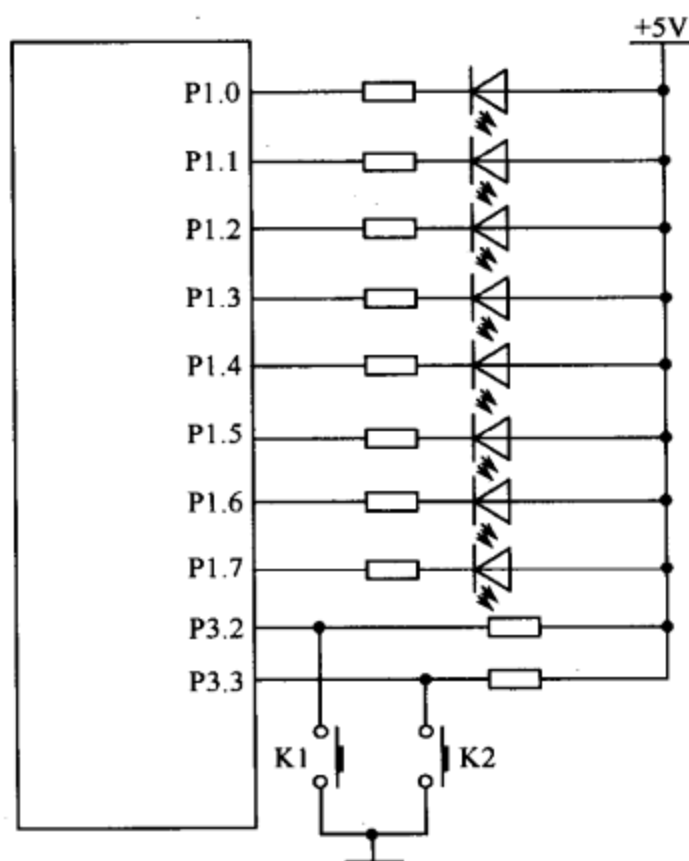


图 6-4 外中断 0 和外中断 1 实验电路

根据上述要求,设计的源程序如下:

```
ORG 0000H
AJMP MAIN                ;转主程序
ORG 0003H
AJMP INT_0               ;外中断 0 入口地址
                           ;转外中断 0 服务程序
ORG 0013H
AJMP INT_1               ;外中断 1 入口地址
                           ;转外中断 1 服务程序
;以下是主程序
ORG 0100H
MAIN: MOV SP, #5FH        ;初始化堆栈
      SETB IT0            ;外中断 0 为脉冲触发方式
      SETB IT1            ;外中断 1 为脉冲触发方式
      SETB EA             ;开总中断
```

SETB EX0	;开外中断 0
SETB EX1	;开外中断 1
MOV A, #0FEH	;将 1111 1110B 送 A
LOOP: MOV P1, A	;P1.0 外接的灯亮
RL A	;累加器 A 中的值循环左移 1 位
LCALL DELAY	;调延时子程序
AJMP LOOP	;再显示
;以下是外中断 0 服务程序	
INT_0: CLR EX1	;关闭外中断 1, 不影响中断 0
PUSH ACC	;保护现场
MOV R2, #04H	;显示 4 次
LOOP0: MOV A, #00H;	
MOV P1, A	;P1 口外接的 8 只灯同时亮
ACALL DELAY	;调延时子程序
MOV A, #0FFH;	
MOV P1, A	;P1 口外接的 8 只灯全灭
ACALL DELAY	;调延时子程序
DJNZ R2, LOOP0	;显示 4 次
POP ACC	;恢复现场
SETB EX1	;开外中断 1
RETI	;中断返回
;以下是外中断 1 服务程序	
INT_1: CLR EX0	;关闭外中断 0, 不影响中断 1
PUSH ACC	;保护现场
MOV R3, #04H	;显示 4 次
LOOP1: MOV A, #0F0H	;送 1111 0000B 到 A
MOV P1, A	;P1.0~P1.3 外接的灯亮, P1.3~P1.7 外接的灯灭
ACALL DELAY	;调延时子程序
MOV A, #0FH	;送 0000 1111B 到 A
MOV P1, A	;P1.0~P1.3 外接的灯灭, P1.3~P1.7 外接的灯亮
ACALL DELAY	;调延时子程序
DJNZ R3, LOOP1	;显示 4 次
POPACC	;恢复现场
SETB EX0	;开外中断 0
RETI	;中断返回
;以下是 0.5s 延时子程序	
DELAY: MOV R5, #0FAH	;1 个机器周期

```

D2: MOV  R4, #0FAH      ;1 个机器周期
D1: NOP                  ;1 个机器周期
    NOP
    NOP
    NOP
    NOP
    NOP
    DJNZ R4, D1          ;2 个机器周期
    DJNZ R5, D2          ;2 个机器周期
    RET                  ;2 个机器周期
    END

```

实验步骤如下：

(1) 打开 Keil 软件, 输入上面的程序, 保存为 int.asm。对程序进行编译、链接和调试, 产生 int.hex 目标文件。

(2) 用硬件仿真器对源程序进行仿真, 观察实验现象。

该实验程序在本书附光盘的 example\ch_6\int 文件夹中。

第二节 定时/计数器及实验

一、定时/计数器概述

1. 什么是计数

计数是指对外部事件进行计数。外部事件的发生以输入脉冲表示, 因此计数功能的实质就是对外来脉冲进行计数。80C51 单片机有 T0(P3.4) 和 T1(P3.5) 两个信号引脚, 分别是这两个计数器的计数输入端。外部输入的脉冲在负跳变时有效, 进行计数器加 1 (加法计数)。

计数方式下, 单片机在每个机器周期的 S5P2 拍节对外部计数脉冲进行采样。如果前一个机器周期采样为高电平, 后一个机器周期采样为低电平, 即为一个有效的计数脉冲。在下一机器周期的 S3P1 进行计数。可见采样计数脉冲是在两个机器周期进行的。鉴于此, 计数脉冲的频率不能高于振荡脉冲频率的 1/24。

2. 什么是定时

定时是通过计数器的计数来实现的, 不过此时的计数脉冲来自单片机的内部, 即每个机器周期产生一个计数脉冲。也就是每个机器周期计数器加 1。定时和计数的脉冲来源如图 6-5 所示。

由于一个机器周期等于 12 个振荡脉冲周期, 因此计数频率为振荡频率的 1/12。如果单片机采用 12MHz 晶体, 则计数频率为 1MHz。即每微秒计数器加 1。这样不但可以根据计数值计算出定时时间, 也可以反过来按定时时间的要求计算出计数器的预置值。

重点提示 在第五章中曾介绍过定时程序(或延迟程序), 它是通过循环程序来进行

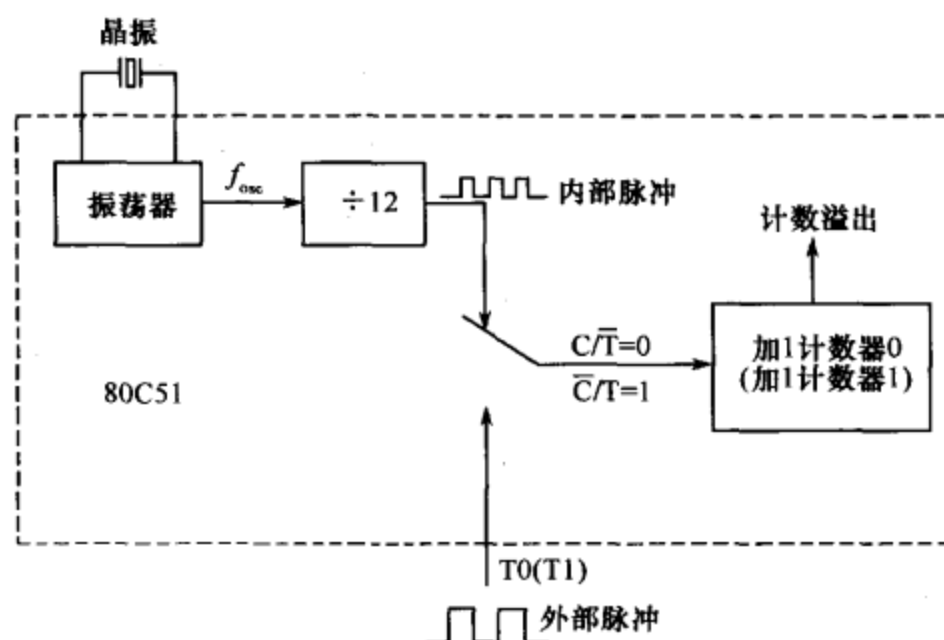


图 6-5 定时和计数脉冲的来源

延时的,特点是时间精确,且不需外加硬件电路,但定时要占用 CPU,增加 CPU 开销,因此,定时的时间不宜太长。本章介绍的利用 80C51 内部定时/计数器实现的定时控制,不但可实现任意定时和计数,而且 CPU 不必通过等待来实现延时,因此,可以提高 CPU 的效率。

3. 定时/计数器的组成

图 6-6 是 80C51 单片机内部定时/计数器结构图。

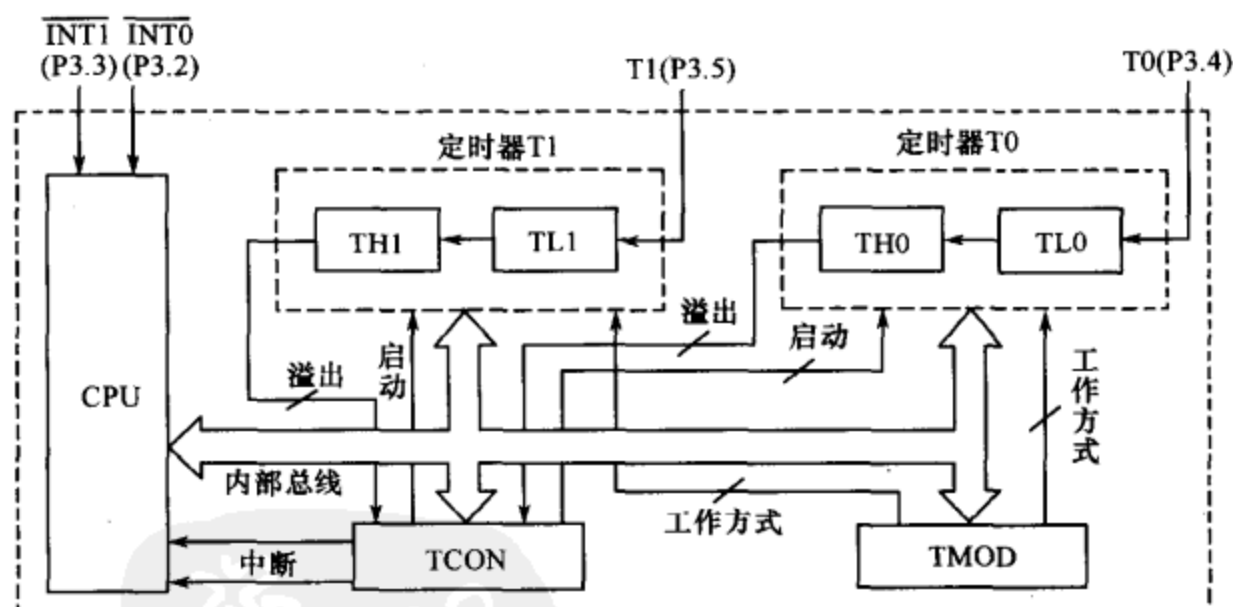


图 6-6 定时/计数器的结构

从图 6-6 中可以看出,定时/计数器主要由几个特殊功能寄存器 TH0、TL0、TH1、TL1 以及 TMOD、TCON 组成。TH0(高 8 位)、TL0(低 8 位)构成 16 位计数器,用来存放定时器 T0 的计数初值,TH1(高 8 位)、TL1(低 8 位)构成 16 位计数器,用来存放定时器 T1 的计数初值,这两个 16 位计数器都是 16 位的加 1 计数器。TMOD 用来控制两个定时/计数器的工作方式,TCON 用做中断溢出标志并控制定时器的启停。

两个定时/计数器都可由软件设置为定时或计数的工作方式,其中 T1 还可作为串行口的波特率发生器。不论 T0 或 T1 是工作于定时方式还是计数方式,它们在对内部时钟

或外部事件进行计数时,都不占用 CPU 时间,直到定时/计数器产生溢出。如果满足条件,CPU 才会停下当前的操作,去处理“时间到”或者“计数溢出”这样的事件。因此,定时/计数器是与 CPU“并行”工作的,不会影响 CPU 的其他工作。

二、定时/计数器的控制寄存器

与两个定时/计数器 T0 和 T1 有关的控制寄存器有 TMOD、TCON 和 IE,它们主要用来设置各个定时/计数器的工作方式、选择定时或计数功能、控制启动运行以及作为运行状态的标志等。

1. 工作方式控制寄存器 TMOD

TMOD 寄存器是一个特殊功能寄存器,字节地址为 89H,不能位寻址。各位定义如下:

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

TMOD 的低半字节用来定义定时/计数器 0,高半字节定义定时/计数器 1。复位时 TMOD 为 00H。

1)M1、M0——工作方式选择位

M1、M0 用来选择工作方式,对应关系如表 6-1 所列。

表 6-1 定时/计数器的方式选择

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

2)C/ \overline{T} ——定时/计数功能选择位

C/ \overline{T} =0 为定时方式。在定时方式中,以振荡输出时钟脉冲的 12 分频信号作为计数信号,如果单片机采用 12MHz 晶体,则计数频率为 1MHz,则计数脉冲周期为 1 μ s,即每微秒计数器加 1。

C/ \overline{T} =1 为计数方式。在计数方式中,单片机在每个机器用期的 S5P2 拍节对外部计数脉冲进行采样。如果前一个机器周期采样为高电平,后一个机器周期采样为低电平,即为一个有效的计数脉冲。在下一机器周期的 S3P1 进行计数。

3)GATE——门控位

GATE=1,定时/计数器的运行受外部引脚输入电平的控制,即 $\overline{INT0}$ 控制 T0 运行, $\overline{INT1}$ 控制 T1 运行。

GATE=0,定时/计数器的运行不受外部输入引脚的控制。

2. 定时器控制寄存器 TCON

TCON 寄存器既参与中断控制又参与定时控制。在前面介绍中断时已作了简要介绍,下面再对与定时控制有关的功能加以说明。

1) TF0 和 TF1——计数溢出标志位

当计数器计数溢出(计满)时,该位置 1;使用查询方式时,此位作为状态位供查询,但应注意查询有效后应用软件方法及时将该位清 0;使用中断方式时,此位作为中断标志位,在转向中断服务程序时由硬件自动清 0。

2) TR0 和 TR1——定时器运行控制位

TR0(TR1)=0,停止定时/计数器工作。

TR0(TR1)=1,启动定时/计数器工作。

该位根据需靠软件来置 1 或清 0,以控制定时器的启动或停止。

3. 中断允许控制寄存器 IE

IE 寄存器在前面介绍中断时已作了简要介绍,下面对与定时/计数器有关的位再重复介绍如下:

ET0 和 ET1——定时/计数中断允许控制位

ET0(ET1)=0,禁止定时/计数中断。

ET0(ET1)=1,允许定时/计数中断。

三、定时/计数器的工作方式

80C51 单片机的定时/计数器共有 4 种工作方式,由寄存器 TMOD 的 M1M0 位进行控制,现以定时/计数器 0 为例进行介绍,定时/计数器 1 与定时/计数器 0 完全相同。

1. 工作方式 0

1) 逻辑电路结构

工作方式 0 是 13 位计数结构的工作方式,其计数器由 TH0 全部 8 位和 TL0 的低 5 位构成。TL 的高 3 位未用,图 6-7 为工作方式 0 的逻辑电路结构图。

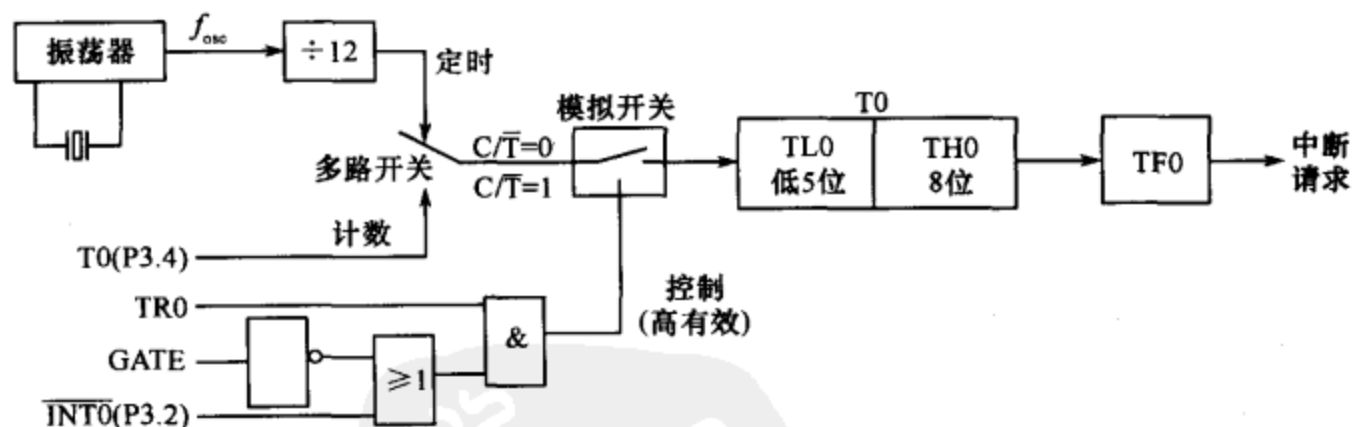


图 6-7 工作方式 0 逻辑电路结构图

当 $C/\overline{T}=0$ 时,多路开关接通振荡脉冲的 12 分频输出,13 位计数器以此进行计数,这就是定时方式。

当 $C/\overline{T}=1$ 时,多路开关接通计数引脚 P3.4(T0),外部计数脉冲由引脚 P3.4 输入。当计数脉冲发生负跳变时,计数器加 1,这就是计数方式。

不管是定时方式还是计数方式,当 TL0 的低 5 位计数溢出时,向 TH0 进位,而全部 13 位计数溢出时,则向计数溢出标志位 TF0 进位。在满足中断条件时,向 CPU 申请中断,若需继续进行定时或计数,则应用指令对 TL0、TH0 重新置数,否则,下一次计数将会从 0 开始,造成计数或定时时间不准确。

说明 T0 能否启动,取决于 TR0、GATE 和引脚 $\overline{\text{INT0}}$ 的状态。

当 GATE=0 时,GATE 信号封锁了或门,使引脚 $\overline{\text{INT0}}$ 信号无效。而或门输出端的高电平状态却打开了与门。这时如果 TR0=1,则与门输出为 1,模拟开关接通,定时/计数器 0 工作。如果 TR0=0,则断开模拟开关,定时/计数器 0 不能工作。

当 GATE=1,同时 TR0=1 时,模拟开关是否接通由 $\overline{\text{INT0}}$ 控制。当 $\overline{\text{INT0}}=1$ 时,与门输出高电平,模拟开关接通,定时/计数器 0 工作;当 $\overline{\text{INT0}}=0$ 时,与门输出低电平,模拟开关断开,定时/计数器 0 停止工作。这种情况可用于测量外信号的脉冲宽度。

2) 计数初值的计算

工作方式 0 是 13 位计数结构,其最大计数为 $2^{13}=8192$,也就是说,每次计数到 8192 都会产生溢出,去置位 TF0。但在实际应用中,经常会有少于 8192 个计数值的要求,例如,要求计数到 1000 就产生溢出,那么怎么办呢? 其实,仔细想一想,这个问题很好解决,在计数时,不从 0 开始,而是从一个固定值开始,这个固定的值的大小,取决于被计数的大小。如要计数 1000,预先在计数器里放进 7192,再来 1000 个脉冲,就到了 8192,这个 7192 计数初值,也称做预置值。

定时也有同样的问题,并且也可采用同样的方法来解决。假设单片机的晶振是 12 MHz,那么每个计时脉冲是 $1\mu\text{s}$,计满 8192 个脉冲需要 8.192ms,如果只需定时 1ms,可以作这样的处理:1ms 即 $1000\mu\text{s}$,也就是计数 1000 时满。因此,计数之前预先在计数器里面放进 $8192-1000=7192$,开始计数后,计满 1000 个脉冲到 8192 即产生溢出。如果计数初值为 X,则可按以下公式进行计算定时时间:

$$\text{定时时间} = (2^{13} - X) \times \text{机器周期}$$

因为机器周期 = $12 \times$ 时钟周期 (机器周期的频率是时钟频率的 12 分频)

$$\text{而时钟周期} = \frac{1}{\text{晶振频率}}$$

所以
$$\text{定时时间} = (2^{13} - X) \times \frac{12}{\text{晶振频率}}$$

例如,如果需要定时 3ms ($3000\mu\text{s}$),晶振为 12MHz,设计数初值为 X,则根据上述公式可得

$$3000 = (2^{13} - X) \times \frac{12}{12}$$

由此得

$$X = 5192$$

说明 单片机中的定时器通常要求不断重复定时,一次定时时间到之后,紧接着进行第 2 次的定时操作。一旦产生溢出,计数器中的值就回到 0,下一次计数从 0 开始,定时时间将不正确。为使下一次的定时时间不变,需要在定时溢出后马上把计数初值送到计数器。

2. 工作方式 1

1) 逻辑电路结构

工作方式 1 是 16 位计数结构的工作方式,计数器由 TH0 全部 8 位和 TL0 全部 8 位构成。其逻辑电路和工作情况与方式 0 基本相同,如图 6-8 所示 (以定时/计数器 0 为例)。所不同的只是组成计数器的位数,它比工作方式 0 有更宽的计数范围,因此,在实际

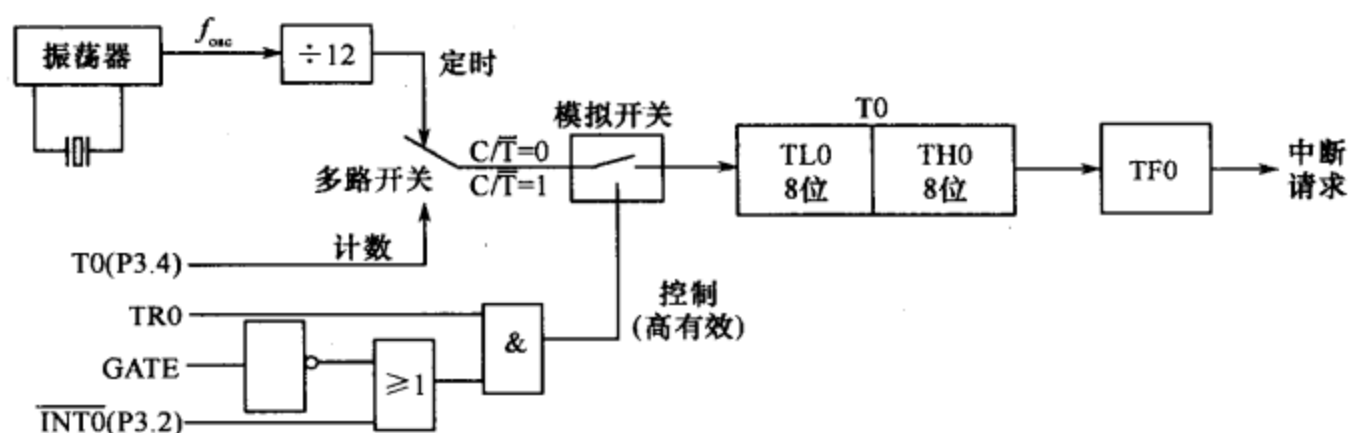


图 6-8 工作方式 1 逻辑电路结构图

应用中,工作方式 1 可以代替工作方式 0。

2) 计数初值的计算

由于工作方式 1 是 16 位计数结构,因此,其最大计数为 $2^{16} = 65536$,也就是说,每次计数到 65536 都会产生溢出,去置位 TF0。如果计数初值为 X,则可按以下公式计算定时时间:

$$\text{定时时间} = (2^{16} - X) \times \text{机器周期}$$

3. 工作方式 2

1) 逻辑电路结构

工作方式 0 和工作方式 1 若用于循环重复定时或计数时,每次计满溢出后,计数器回到 0,要进行新一轮的计数,就得重新装入计数初值。因此,循环定时或循环计数应用时就存在反复设置计数初值的问题,这项工作是由软件来完成的,需要花费一定时间。这样就会造成每次计数或定时产生误差。如果用于一般的定时,则是无关紧要的。但是有些工作,对时间的要求非常严格,不允许定时时间不断变化,用工作方式 0 和工作方式 1 就不行了,所以就引入了工作方式 2。图 6-9 是定时/计数器 0 在工作方式 2 的逻辑结构。

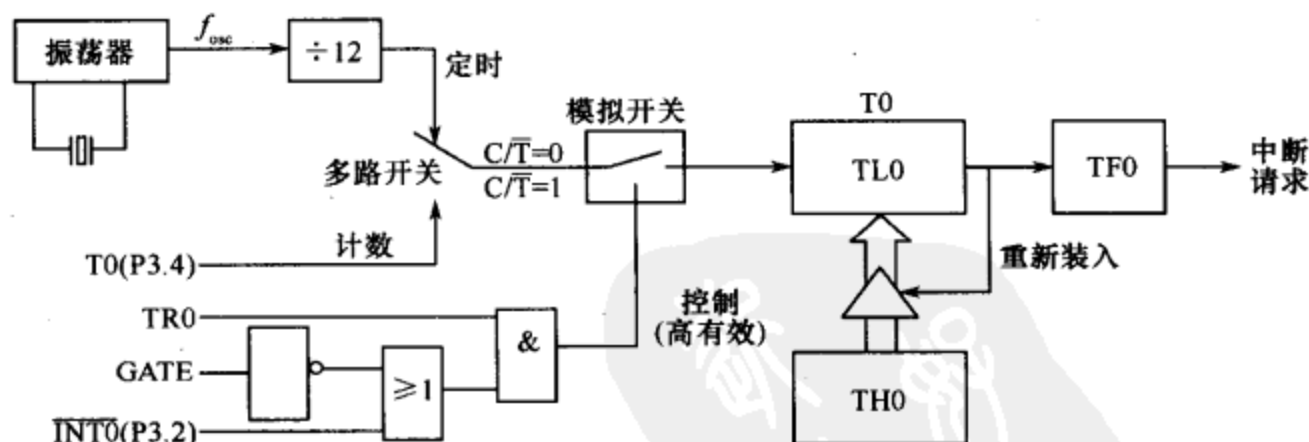


图 6-9 工作方式 2 逻辑电路结构图

在工作方式 2 下,把 16 位计数器分为两部分,即以 TL0 作为计数器,以 TH0 作为预置寄存器,初始化时把计数初值分别装入 TL0 和 TH0 中。当计数溢出后,不是像前两种工作方式那样通过软件方法,而是由预置寄存器 TH0 以硬件方法自动给计数器 TL0 重新加载。变软件加载为硬件加载。这不但省去了用户程序中的重装指令,而且也有利于提高定时精度。

2) 计数初值的计算

由于工作方式 2 是 8 位计数结构, 因此, 其最大计数值为 $2^8 = 256$, 计数值十分有限。如果计数初值为 X , 则可按以下公式计算定时时间:

$$\text{定时时间} = (2^8 - X) \times \text{机器周期}$$

4. 工作方式 3

1) 逻辑电路结构

工作方式 3 的作用比较特殊, 只适用于定时器 T0。如果企图将定时器 T1 置为工作方式 3, 则它将停止计数, 其效果与置 $TR1 = 0$ 相同, 即关闭定时器 T1。

当 T0 在工作方式 3 时, 它被拆成两个独立的 8 位计数器 TL0 和 TH0, 其逻辑结构如图 6-10 所示。

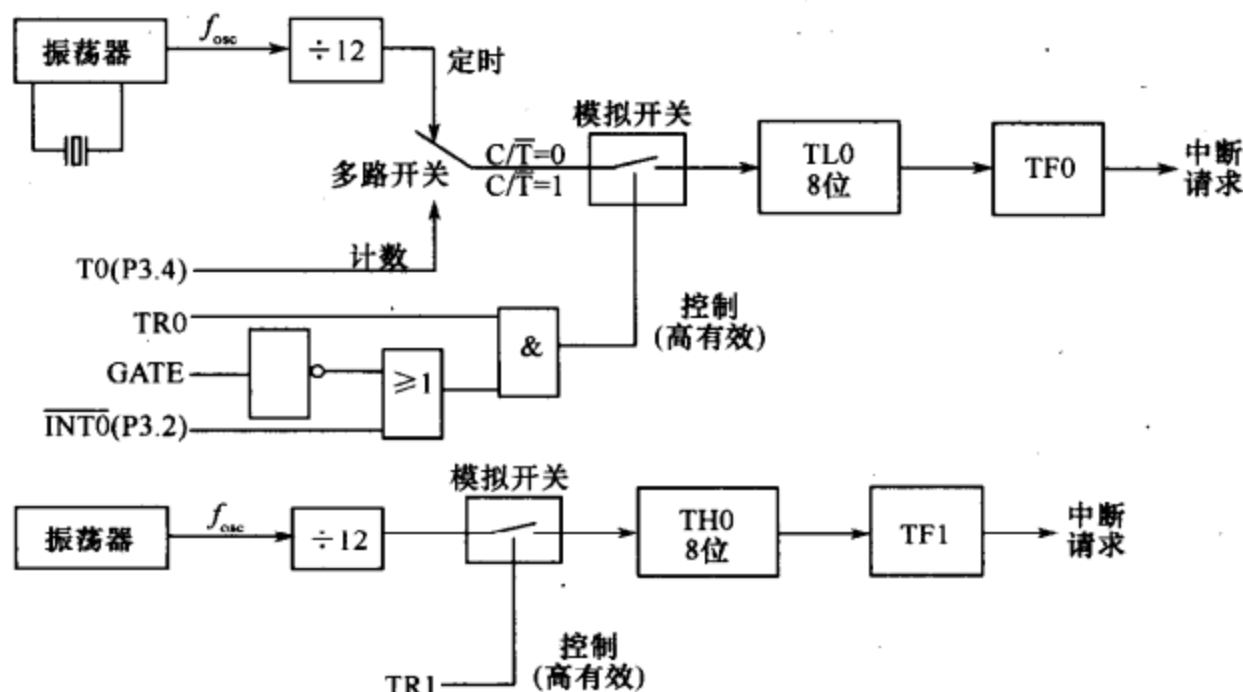


图 6-10 工作方式 3 逻辑电路结构

图 6-10 中, 上方的 8 位计数器 TL0 使用原定时器 T0 的控制位 C/\overline{T} 、GATE、TR0 和 $\overline{INT0}$, TL0 既可以计数使用, 又可以定时使用, 其功能和操作与前面介绍的工作方式 0 或工作方式 1 完全相同。

下方的 TH0 只能作为简单的定时器使用。而且由于定时/计数器 0 的控制位已被 TL0 独占, 因此只好借用定时/计数器 1 的控制位 TR1 和 TF1。即以计数溢出去置位 TF1, 而定时的启动和停止则受 TR1 的状态控制。

由于 TL0 既能作为定时器使用也能作为计数器使用, 而 TH0 只能作为定时器使用却不能作为计数器使用, 因此在工作方式 3 下, 定时/计数器 0 可以构成两个定时器或一个定时器一个计数器。

注意 如果定时/计数器 0 已在工作方式 3 时, 则定时/计数器 1 只能在工作方式 0、工作方式 1 或工作方式 2 下, 因为它的运行控制位 TR1 及计数溢出标志位 TF1 已被定时/计数器 0 借用。在这种情况下, 定时/计数器 1 通常是作为串行口的波特率发生器使用, 以确定串行通信的速率, 因为已没有计数溢出标志位 TF1 可供使用, 因此只能把计数溢出直接送给串行口。当作为波特率发生器使用时, 只需设置好工作方式, 便可自动运行。如要停止工作, 只需送入一个把它设置为工作方式 3 的方式控制字就可以了。因为

定时/计数器 1 不能在工作方式 3 下使用,如果硬把它设置为工作方式 3,就停止工作。

2) 计数初值的计算

由于工作方式 3 是 8 位计数结构,因此,其最大计数值为 $2^8=256$,如果计数初值为 X ,则可按以下公式计算定时/计数器 0 的定时时间:

$$\text{定时时间}=(2^8-X)\times\text{机器周期}$$

四、定时/计数器应用举例及实验

1. 定时/计数器的初始化

在利用定时/计数器进行定时和计数之前,首先要通过软件对它进行初始化,初始化主要包括以下几个方面。

(1)对工作方式寄存器 TMOD 赋值,确定工作方式。

(2)计算计数初值,并将其写入寄存器 TH0、TL0 和 TH1、TL1。

(3)根据需要,对中断允许控制寄存器 IE 赋初值,开放定时器中断。

(4)使用定时器控制寄存器 TCON 的 TR0、TR1 位启动定时/计数器,TR0 或 TR1 置位之后,计数器即可按规定的工作方式进行定时或开始计数。

2. 应用举例及实验

实验 3 设单片机晶振频率为 12MHz,使用定时器 0 的工作方式 1,以查询方式进行定时,由 P1.0 输出周期为 2ms 的等宽方波。采用 Keil 进行软件模拟仿真。

(1)对 TMOD 寄存器赋值。使用定时器 0 的工作方式 1,应使 $M1M0=01$;为实现定时功能,应使 $C/\bar{T}=0$;为实现定时/计数器 0 的运行控制,则 $GATE=0$ 。定时/计数器 1 不用,有关位设定为 0。因此 TMOD 寄存器器初始化为 01H。

(2)计算计数初值。要使 P1.0 输出 2ms 的等宽方波,只需使 P1.0 每隔 1ms 取反一次即可。为此,定时时间应为 1ms。设计数初值为 X ,由于使用 12MHz 晶振和工作方式 1,根据

$$\text{定时时间}=(2^{16}-X)\times\frac{12}{\text{晶振频率}}$$

可得

$$1000=(65536-X)\times 1$$

所以 $X=64536D$ (D 表示十进制)

将 64536D 转换为十六进制后为 0FC18H。其中,高 8 位为 0FCH,放入 TH0,低 8 位为 18H,放入 TL0。

(3)对 IE 赋初值。对于本例,因采用查询方法,不需要中断,因此,将 IE 置 00H。

(4)启动定时器 0。将定时器控制寄存器 TCON 中的 TR0 设置为 1,可启动定时器 0,TR0 设置为 0,定时器 0 停止定时。

根据以上分析,设计的源程序如下:

```
ORG 0000H
```

```
AJMP MAIN
```

```
ORG 200H
```

```
MAIN: MOV TMOD, #01H ;将 T0 为工作方式 1
```

```

MOV TH0, #0FCH ;设置计数初值
MOV TL0, #18H
MOV IE, #00H    ;禁止中断
SETB TR0        ;启动定时器 0
LOOP: JBC TF0, NEXT ;计数溢出 TF0 等于 1 则转到 NEXT,并对 TF0 清 0
      AJMP LOOP
NEXT: MOV TH0, #0FCH ;重新设置计数初值
      MOV TL0, #18H
      CLR TF0        ;此行也可不加,因此在 JBC 指令中 TF0 已清 0
      CPL P1.0       ;输出取反
      AJMP LOOP      ;重复循环
END

```

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 tim1.asm。对程序进行编译、链接,产生 tim1.hex 目标文件。

(2)点击菜单 Project→Option for Target ‘Target’,在出现的窗口中设置为软件模拟仿真(Use Simulator)。

(3)按 Ctr+F5 进入调试状态

(4)点击菜单 Peripherals→Timer→Timer0,打开定时器 0 窗口,在 Mode 选项中设定“1:16Bit Timer/Counter”(工作方式 1)和“Timer”(定时方式),设置好计数初值,并将 TR0 前的“√”选中。设置好的窗口如图 6-11 所示。

(5)点击菜单 Peripherals→I/O—Ports→Port1,打开输入/输出窗口,如图 6-12 所示。



图 6-11 定时器 0 观察窗口

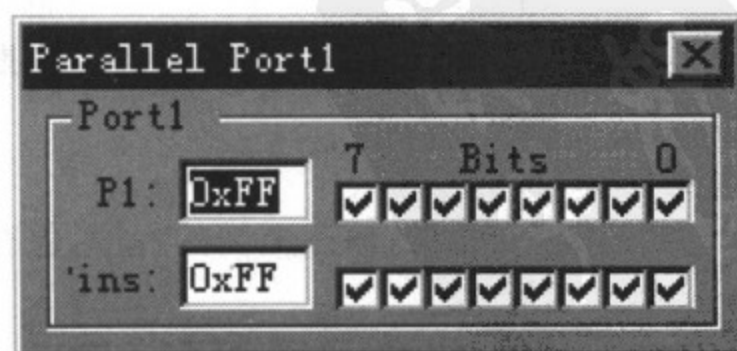


图 6-12 输入/输出观察窗口

(6)按 F5,进行全速运行,会发现定时器 0 窗口中 TH0、TL0 中的值会不断地发生变化(说明在计数),输入/输出窗口中的 P1.0 框中的“√”在不停地闪动(说明计数溢出后

P1.0 口取反)。

(7)再按 F11,进行单步运行,再观察实验现象。

该实验程序在本书所附光盘的 example\ch_6\tim1 文件夹中。

方法技巧 TF0 是定时/计数器 0 的溢出标记位,当产生溢出后,该 TF0 由 0 变 1,所以查询该位就可以知道定时时间是否已到。该位为 1 后,不会自动清 0,必须用软件将标记位清 0;否则,在下一次查询时,即便时间未到,这一位仍是 1,会出现错误的执行结果。

这个程序不但可以定时,而且在“LOOP:…”与“AJMP LOOP”指令之间插入一些指令来做其他事情,不过,要保证执行这些指令的时间一定短于定时时间。

实验 4 设单片机晶振频率为 12MHz,使用定时器 0 的工作方式 1,以中断方式进行定时,由 P1.0 输出周期为 2ms 的等宽方波。采用 Keil 进行软件模拟仿真。

该例与实验 3 条件相同,只是以中断方式完成外,根据实验 3 计算出的有关数据,编写的源程序如下:

```
ORG 0000H
AJMP MAIN
ORG 000BH
AJMP TIME
ORG 200H
;以下是主程序
MAIN: MOV TMOD, #01H      ;将 T0 为工作方式 1
      MOV TH0, #0FCH      ;设置计数初值
      MOV TL0, #18H
      SETB EA              ;开中断
      SETB ET0             ;开定时器 0 中断
      SETB TR0             ;启动定时器 0
      HERE: SJMP $          ;等待中断,此处可完成其他工作
;以下是中断服务程序
TIME: MOV TH0, #0FCH      ;重置计数初值
      MOV TL0, #18H
      CPL P1.0             ;输出取反
      RETI                 ;中断返回
      END
```

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 tim2.asm。对程序进行编译、链接,产生 tim2.hex 目标文件。

(2)点击菜单 Project→Option for Target ‘Target’,在出现的窗口中设置为软件模拟仿真(Use Simulator)。

(3)按 Ctr+F5 进入调试状态

(4)点击菜单 Peripherals→Timer→Timer0,打开定时器 0 窗口,设置同实验 3。

(5)点击菜单 Peripherals→I/O—Ports→Port1,打开输入/输出口 1。

(6)点击菜单 Peripherals→Interrupt,打开中断窗口,选中 EA、ET0 框中的“√”,如图 6-13 所示。

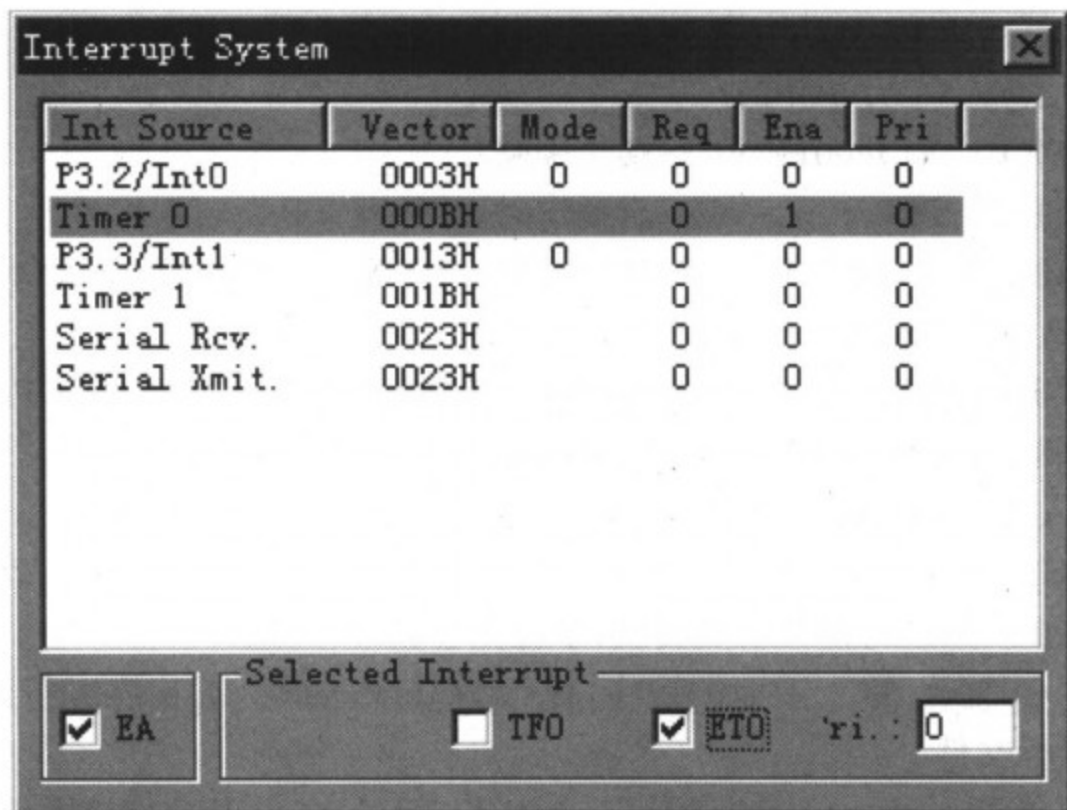


图 6-13 中断窗口

(7)按 F5,进行全速运行,会发现定时器 0 窗口中 TH0、TL0 中的值会不断地发生变化(说明在计数),输入/输出窗口中的 P1.0 框中的“√”在不停地闪动(说明计数溢出后 P1.0 口取反)。将中断窗口 EA、ET0 框中的“√”去掉(禁止中断),P1.0 不再取反(中断禁止)。

(8)按 F11 进行单步运行,再观察实验现象。

该实验程序在本书所附光盘的 example\ch_6\tim2 文件夹中。

实验 5 用定时/计数器 T0 方式 2 计数,外部计数信号由 T0(P3.4)引脚输入,每出现一次负跳变,计数器加 1,要求采用查询方式,每计满 10 次,使 P1.0 端取反。先采用 Keil 进行软件模拟仿真,再采用下载型实验板进行硬件仿真实验。

(1)对 TMOD 寄存器赋值。使用定时器 0 的方式 2,应使 M1M0=10;为实现计数功能,应使 C/T=1;为实现定时/计数器 0 的运行控制,则 GATE=0。定时/计数器 1 不用,有关位设定为 0。因此 TMOD 寄存器初始化为 06H。

(2)计算计数初值。计数器的初值 X 为

$$X=2^8-10=246D=0F6H$$

故 TH0 和 TL0 的值均为 0F6H。

(3)对 IE 赋初值。对于本例,因采用查询方法,不需要中断,因此,将 IE 置 00H。

(4)启动定时器 T0。将定时器控制寄存器 TCON 中的 TR0 设置为 1,可启动定时器 0,TR0 设置为 0,定时器 0 停止定时。

根据以上分析,设计的源程序如下:

```
ORG 0000H
```

```

    AJMP MAIN
    ORG 200H
MAIN: MOV TMOD, #06H ;将 T0 设为工作方式 2 计数方式
      MOV TH0, #0F6H ;设置计数初值
      MOV TL0, #0F6H
      MOV IE, #00H   ;禁止中断
      SETB TR0       ;启动定时/计数器 T1
LOOP: JBC TF0, NEXT  ;计数溢出, TF0 等于 1, 则转到 NEXT, 并对 TF0 清 0
      AJMP LOOP
NEXT: CLR TF0        ;此行也可不加, 因为在 JBC 指令中 TF0 已清 0
      CPL P1.0       ;输出取反
      AJMP LOOP      ;重复循环
      END

```

先采用 Keil 软件进行模拟仿真实验, 步骤如下:

(1) 打开 Keil 软件, 输入上面的程序, 保存为 tim3.asm。对程序进行编译、链接, 产生 tim3.hex 目标文件。

(2) 点击菜单 Project→Option for Target ‘Target’, 在出现的窗口中设置为“软件模拟仿真”(Use Simulator)。

(3) 按 Ctr+F5 进入调试状态

(4) 点击菜单 Peripherals→I/O-Ports, 打开 Port1 和 Port3, 如图 6-14 所示。

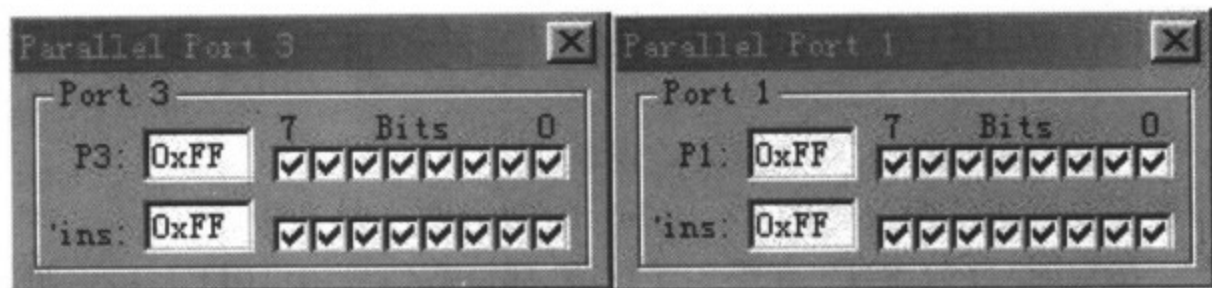


图 6-14 打开的 Port1 和 Port3 窗口

(5) 按 F5, 进行全速运行, 用鼠标连续点击 Port3 窗口中的 P3.4 框(即 T0 输入端)10 次(模拟输入 10 次脉冲信号), 会发现 Port1 窗口中的 P1.0 中的“√”消失(即 P1.0 取反), 再次用鼠标连续点击 Port3 窗口中的 P3.4 框 10 次, Port1 窗口中的 P1.0 中的“√”出现(即 P1.0 再次取反)。

下面再采用下载型实验板进行硬件仿真实验, 实验步骤如下:

(1) 选择计数源。下载型实验板提供了两个计数源, 可供单片机做计数实验。第 1 个计数源由 555 电路提供, 第 2 个计数源由 PCF8563 提供, 通过 JP2 选择。JP2 的下方标有 Count Select 字样, 上方左侧标有“555”, 右侧标有“8563”, 分别代表选择这两个计数源。这里将短路夹将 JP2 的左侧两个脚短路, 即选中 555 作为计数源。此时, 555 输出的脉冲输入到单片机的 P3.4(T0)。

(2) 将下载型实验板、MOON-51 仿真器和 PC 机连接好, 打开 Keil 软件, 输入上面的程序, 保存为 tim3.asm。对程序进行编译、链接, 产生 tim3.hex 目标文件。

(3)点击菜单 Project→Option for Target ‘Target1’,出现对工程设置的对话框,在 Target 页中,输入仿真器的工作频率(11.0592MHz)。在 Debug 页中,选择 Use Keil Monitor-51 Driver(硬件仿真)。

(4)按编译按钮进行编译,编译成功后,可以进行硬件仿真了,将 Mon51 仿真器放入下载型实验板的 40 脚活动插座中,这时仿真器的电源由实验开发板提供,开始仿真时按一下仿真器上的复位按钮。

(5)按 Ctrl+F5 进入调试状态,这时再按 F5 全速运行状态。这时,可以看到下载型实验板 555 的输出脚外接的灯闪烁 10 次后,P1.0 外接的灯开始点亮,再闪烁 10 次后,P1.0 外接的灯开始熄灭,反复循环。

该实验程序在本书所附光盘的 example\ch_6\tim3 文件夹中。

实验 6 设单片机晶振频率为 12MHz,使用定时器 0 的工作方式 1,以中断方式进行定时,由 P1.0 输出周期为 2s 的等宽方波。采用下载型实验板进行硬件仿真实验。

(1)对 TMOD 寄存器赋值。使用定时器 0 的工作方式 1,应使 M1M0=01;为实现定时功能,应使 C/T=0;为实现定时/计数器 0 的运行控制,则 GATE=0。定时/计数器 1 不用,有关位设定为 0。因此 TMOD 寄存器器初始化为 01H。

(2)计算计数初值。周期为 2s 的方波要求定时值为 1s,在时钟为 12MHz 的情况下,即使采用定时器 0 工作方式 1(16 位计数器),这个值也超过了工作方式 1 可能提供的最大定时值(65.536ms)。此时可采取以下方法:让定时器 T0 在工作方式 1,定时时间为 50ms。另设一个软件计数器,初始值为 20(注意 20 不是十六进制数)。每隔 50ms 定时时间到,产生溢出中断,在中断服务程序中使软件计数器减 1,这样,当软件计数器减到 0 时,就获得 1s 定时。

选用定时器 T0 方式 1,时钟频率 12MHz,50ms 定时所需的计数初值 X 可根据下式计算:

$$50000 = (65536 - X) \times 1$$

所以 $X = 15536D = 3CB0H$,则 TH0 初值为 3CH,TL0 初值为 B0H。

(3)对 IE 赋初值。对于本例,因采用定时器 T0 中断方法,因此,将 IE 的 EA、ET0 置 1。

(4)启动定时器 T0。将定时器控制寄存器 TCON 中的 TR0 设置为 1,可启动定时器 0,TR0 设置为 0,定时器 0 停止定时。

根据以上分析,设计的源程序如下:

```
ORG 0000H
AJMP MAIN           ;转主程序
ORG 000BH           ;定时器 0 的中断向量地址
AJMP TIMO           ;转定时器 T0 中断服务程序
;以下是主程序
ORG 200H
MAIN: MOV TMOD, #01H ;定时器 T0 方式 1
      MOV TL0, #0B0H  ;T0 低 8 位初值
      MOV TH0, #3CH   ;T0 高 8 位初值
```

```

        SETB EA                ;开总中断
        SETB ET0               ;开定时器 0 中断
        SETB TR0               ;启动 T0
        MOV R1, #20            ;软件计数器初值
HERE:   AJMP HERE              ;等待中断,真正工作时,这里可加入任意程序
;以下是中断服务程序
TIMO:   DJNZ R1, NEXT          ;R1 不等 0,则转移
        CPL P1.0              ;输出方波
        MOV R1, #20            ;重装软件计数器初值
NEXT:   MOV TL0, #0B0H         ;重装定时器初值
        MOV TH0, #3CH
        RETI                   ;中断返回
END

```

方法技巧 在以上定时程序中,中断服务程序都要进行重装计数器初值等操作。这样,在定时器溢出发出中断请求到重装完定时器初值,并在此基础上重新开始计数定时时,总有一定的时间间隔,造成定时时间多增加了若干微秒。为了减少这种定时误差,就要对重装计数初值作适当的调整。在一般情况下,使定时时间缩短 7 个~10 个机器周期,对于本例,可将定时时间由 50ms 降为 49.99ms,即计数初值 X 由 15536D 提高到 15546D。如果采用定时器工作方式 2(自动重装初值),则可避免重装初值引起的误差,使定时比较精确,但方式 2 的计数长度为 256,定时时间太小。

下面用下载实验板对上述源程序进行简单的验证,若要精确地测出 P1.0 输出周期为 1s 的方波,需要用示波器进行测量。

(1)将下载型实验板、MOON-51 仿真器和 PC 机连接好,打开 Keil 软件,输入上面的程序,保存为 tim4.asm。对程序进行编译、链接,产生 tim4.hex 目标文件。

(2)点击菜单 Project→Option for target 'target1',出现对工程设置的对话框,在 Target 页中,输入仿真器的工作频率(11.0592MHz)。在 Debug 页中,选择 Use Keil Monitor-51 Driver(硬件仿真)。

(3)按编译按钮进行编译,编译成功后,可以进行硬件仿真了,将 Mon51 仿真器放入下载型实验板的 40 脚活动插座中,按一下仿真器上的复位按钮。

(4)按 Ctrl+F5 进入调试状态,再按 F5 全速运行。这时,可以看到下载型实验板 P1.0 外接的 LED 灯每 1s 闪烁一次。

该实验程序在本书所附光盘的 example\ch_6\tim4 文件夹中。

实验 7 下面是乐曲《梁祝》的片段,编写程序,在下载型实验板上进行硬件仿真实验。

3—5̣·6̣|1·2̣ 6̣ 1̣ 5̣|5̣ 1̣ 6̣ 5̣ 3̣ 5̣|2———|
2·3̣ 7̣ 6̣|5̣·6̣ 1̣ 2̣|3̣ 1̣ 6̣ 5̣ 6̣ 1̣|5̣———|

乐曲演奏的原理是这样的:组成乐曲的每个音符的频率值(音调)及其持续的时间(音

长)是乐曲能连续演奏所需的两个基本数据,因此只要控制输出到扬声器激励信号频率的高低和持续的时间,就可以使扬声器发出连续的乐曲声。

1)音调的控制

首先来看一下怎样控制音调的高低变化。乐曲是由不同音符编制而成的。音符中有7个音名:C、D、E、F、G、A、B。它们分别唱做哆、咪、咪、法、嗦、啦、唏。声音是由空气振动产生的,每个音名,都有一个固定的振动频率,频率的高低决定了音调的高低。音乐的12平均率规定:每两个八度音(如简谱中的中音1与高音1)之间的频率相差一倍。在两个八度音之间又可分为12个半音,每两个半音的频率比为 $\sqrt[12]{2}$ 。另外,音名A(简谱中的低音6)的频率为440Hz,音名B(简谱中的音7)到C(简谱中的音1)之间、E(简谱中的音3)到F(简谱中的音4)之间为半音,其余为全音。由此可以计算出简谱中从低音1至高音1之间每个音名对应的频率,如表6-2所列。

表 6-2 简谱中的音名与频率的关系

音名	频率/Hz	音名	频率/Hz	音名	频率/Hz
低音 1	262	中音 1	523	高音 1	1047
低音 2	294	中音 2	587	高音 2	1175
低音 3	330	中音 3	659	高音 3	1319
低音 4	349	中音 4	699	高音 4	1397
低音 5	392	中音 5	784	高音 5	1569
低音 6	440	中音 6	880	高音 6	1760
低音 7	494	中音 7	988	高音 7	1976

在下载型实验板上,P3.2 引脚经过三极管驱动一个无源蜂鸣器,构成一个简单的音响电路,因此,只要有了某个音的频率数,就能产生出这个音来。现以 A(低音 6)这个音名为例来进行分析。A 音的频率数为 440Hz,则其周期为

$$T=\frac{1}{f}=\frac{1}{440}\approx 0.00228\text{s}=2.28\text{ms}$$

如果用定时器 1 方式 1 进行定时,要 P3.2 输出周期为 2.28ms 的等宽方波。则定时值为 1.14ms,设计数初值为 X,根据定时值 $= (2^{16}-X)\times \frac{12}{\text{晶振频率}}$ 可得

$$1140=(65536-X)\times 1$$

所以,X=64396D=FB8CH

只要将计数初值装入 TH1、TL1,就能使下载型实验板 P3.2 的高电平或低电平的持续时间为 1.14ms,从而发出 440Hz 的音调。表 6-3 是采用定时器 1 的方式 1 时,《梁祝》片段中各音名、频率和计数初值对照表。

表 6-3 《梁祝》片段中音名、频率和计数初值对照表

音名	低音 3	低音 5	低音 6	中音 1	中音 2	低音 6	中音 1	低音 5	中音 5	高音 1
频率/Hz	330	392	440	523	587	440	523	392	784	1047
计数初值	FA15	FB05	FB8C	FC44	FCAC	FB8C	FC44	FB05	FD82	FE22
音名	中音 6	中音 5	中音 3	中音 5	中音 2	中音 2	中音 3	低音 7	低音 6	低音 5
频率/Hz	880	784	659	784	587	587	659	494	440	392
计数初值	FDC8	FD82	FD09	FD82	FCAC	FCAC	FD09	FC0C	FB8C	FB05
音名	低音 6	中音 1	中音 2	低音 3	中音 1	低音 6	低音 5	低音 6	中音 1	低音 5
频率/Hz	440	523	587	330	523	440	392	440	523	392
计数初值	FB8C	FC44	FCAC	FA15	FC44	FB8C	FB05	FB8C	FC44	FB05

2)音长的控制

乐曲中的音符不单有音调的高低,还要有音的长短,如有的音要唱 1/4 拍,有的音要唱二拍等。在节拍符号中,如用×代表某个音的唱名,×下面无短线为 4 分音符,有一条短横线代表 8 分音符,有两条横线代表 16 分音符,×右边有一条短横线代表 2 分音符,有“.”的音符为符点音符。节拍控制可以通过调用延时子程序(设延时时间为 130ms)的次数来进行控制,以每拍 520ms 的节拍时间为例,那么,1 拍需要循环调用延时子程序 4 次(4×130ms)。同量,半拍需要调用延时子程序 2 次(2×130ms)。具体调用情况如表 6-4 所列。

表 6-4 节拍与调用延时子程序的关系

节拍符号	$\underline{\underline{\times}}$	$\underline{\times}$	$\underline{\times} \cdot$	\times	$\times \cdot$	$\times -$	$\times ---$
名称	16 分音符	8 分音符	8 分符点音符	4 分音符	4 分符点音符	2 分音符	全音符
拍数	1/4 拍	半拍	3/4 拍	1 拍	1 又 1/2 拍	2 拍	4 拍
调用延时程序次数	1H	2H	3H	4H	6H	8H	10H(16D)

乐曲中,每一音符对应着确定的频率,将每一音符的计数初值和其相应的节拍常数(调用延时程序的次数)作为一组,按顺序将乐曲中的所有常数排列成一个表,然后由查表程序依次取出,产生音符并控制节奏,就可以实现演奏效果。

此外,结束符和休止符可以分别用代码 0FFH 和 00H 来表示,若查表结果为 0FFH,则表示曲子終了;若查表结果为 00H,则产生相应的停顿效果。为了产生手弹的节奏感,在某些音符(例如两个相同长音符)间可以插入一个时间单位的相近音符。

根据以上分析,编写的《梁祝》片段源程序如下:

```
ORG 0000H
LJMP MAIN                ;调主程序
ORG 001BH                ;定时器 T1 中断入口
AJMP TIM1                ;定时器 T1 中断
;以下是主程序
ORG 200H
MAIN: MOV TMOD, #10H      ;定时器 T1 方式 1
      SETB EA             ;中断总允许
      SETB ET1            ;允许定时器 T1 中断
      MOV DPTR, #TAB      ;表格首地址
LOOP: CLR A               ;A 清 0
      MOVC A, @A+DPTR      ;查音名高 8 位计数初值
      MOV R1, A            ;将高 8 位计数初值存在 R1 中
      INC DPTR             ;DPTR 加 1,以便查找音名低 8 位计数初值
      CLR A               ;A 清 0
      MOVC A, @A+DPTR      ;查音名低 8 位计数初值
      MOV R0, A            ;将低 8 位计数初值存在 R0 中
```

ORL A,R1	;将低 8 位与高 8 位计数初值相或后送 A
JZ XZF	;如果 A 中的值为 0,则认为是休止符,转 XZF
MOV A,R0	;将低 8 位计数初值送 A
ANL A,R1	;将低 8 位与高 8 位计数初值相与后送 A
CJNE A,#0FFH,NEXT	;若 A 的值与 0FFH 不等,转 NEXT
SJMP MAIN	;若 A 与 0FFH 相等,表示乐曲结束,再从开头开始演奏
NEXT: MOV TH1,R1	;将音名计数初值高 8 位装入 TH1
MOV TL1,R0	;将音名计数初值低 8 位装入 TL1
SETB TR1	;启动定时器 T1
SJMP NEXT1	;转 NEXT1,准备查找延迟常数(调用延时子程序的次数)
XZF: CLR TR1	;关闭定时器,停止发音,使音符休止
NEXT1: CLR A	;A 清 0
INC DPTR	;DPTR 加 1,指向延迟常数
MOVC A,@A+DPTR	;查延迟常数
MOV R2,A	;将延迟常数存在 R2 中
SETB TR1	;因休止时关闭了定时器 T1,因此,需重新启动
LOOP1: LCALL D130	;调用 130ms 延时子程序
DJNZ R2,LOOP1	;控制调用延时子程序的次数
INC DPTR	;DPTR 加 1,指向下一音名的计数初值
AJMP LOOP	;处理下一个音名
;以下是 130ms 延时子程序	
D130: MOV R5,#160	
D2: MOV R4,#200	
D1: NOP	
NOP	
DJNZ R4,D1	
DJNZ R5,D2	
RET	;延时子程序返回
;以下是定时器 T1 中断服务程序	
TIM1: MOV TH1,R1	;重装定时初值
MOV TL1,R0	
CPL P3.2	;P3.2 是音频输出端
RETI	;中断返回
;以下是《梁祝》片段每个音名计数初值和调用延迟次数表格	
TAB: DB 0FAH,15H,08H	;低音 3 的计数初值和调用延时子程序次数
DB 0FBH,05H,06H	;低音 5 的计数初值和调用延时子程序次数

DB 0FBH,8CH,02H	;低音 6 的计数初值和调用延时子程序次数
DB 0FCH,44H,04H	;中音 1 的计数初值和调用延时子程序次数
DB 0FCH,0ACH,02H	;中音 2 的计数初值和调用延时子程序次数
DB 0FBH,8CH,02H	;低音 6 的计数初值和调用延时子程序次数
DB 0FCH,44H,02H	;中音 1 的计数初值和调用延时子程序次数
DB 0FBH,05H,04H	;低音 5 的计数初值和调用延时子程序次数
DB 0FDH,82H,04H	;中音 5 的计数初值和调用延时子程序次数
DB 0FEH,22H,04H	;高音 1 的计数初值和调用延时子程序次数
DB 0FDH,0C8H,02H	;中音 6 的计数初值和调用延时子程序次数
DB 0FDH,82H,02H	;中音 5 的计数初值和调用延时子程序次数
DB 0FDH,09H,02H	;中音 3 的计数初值和调用延时子程序次数
DB 0FDH,82H,02H	;中音 5 的计数初值和调用延时子程序次数
DB 0FCH,0ACH,10H	;中音 2 的计数初值和调用延时子程序次数
DB 0FBH,8CH,01H	;在两个相同音符间加入 1 个时间单位低音 6,以产生节奏感
DB 0FCH,0ACH,06H	;中音 2 的计数初值和调用延时子程序次数
DB 0FDH,09H,02H	;中音 3 的计数初值和调用延时子程序次数
DB 0FCH,0CH,04H	;低音 7 的计数初值和调用延时子程序次数
DB 0FBH,08CH,04H	;低音 6 的计数初值和调用延时子程序次数
DB 0FBH,05H,04H	;低音 5 的计数初值和调用延时子程序次数
DB 0FBH,8CH,02H	;低音 6 的计数初值和调用延时子程序次数
DB 0FCH,44H,04H	;中音 1 的计数初值和调用延时子程序次数
DB 0FCH,0ACH,04H	;中音 2 的计数初值和调用延时子程序次数
DB 0FAH,15H,04H	;低音 3 的计数初值和调用延时子程序次数
DB 0FCH,44H,04H	;中音 1 的计数初值和调用延时子程序次数
DB 0FBH,8CH,02H	;低音 6 的计数初值和调用延时子程序次数
DB 0FBH,05H,02H	;低音 5 的计数初值和调用延时子程序次数
DB 0FBH,8CH,02H	;低音 6 的计数初值和调用延时子程序次数
DB 0FCH,44H,02H	;中音 1 的计数初值和调用延时子程序次数
DB 0FBH,05H,10H	;低音 5 的计数初值和调用延时子程序次数
DB 0FFH,0FFH	;乐曲结束
END	

下面用下载型实验板对上述源程序进行验证,实验步骤如下:

(1)下载型实验板上的 JP5 用于选择 P3.2 究竟工作于输出方式还是输入方式。因本例需要将 P3.2 作为输出端,驱动音响电路,因此,应将短路子插于 JP5 的下方(插座旁印有“↓”标志)。

(2)将下载型实验板、Mon51 仿真器和 PC 机连接好,打开 Keil 软件,输入上面的程序,保存为 tim5.asm。对程序进行编译、链接,产生 tim5.hex 目标文件。

(3)点击菜单 Project→Option for Target‘Target1’,出现对工程设置的对话框,在

Target 页中,输入仿真器的工作频率(11.0592MHz)。在 Debug 页中,选择 Use Keil Monitor-51 Driver(硬件仿真)。

(4)按编译按钮进行编译,编译成功后,可以进行硬件仿真了,将 Mon51 仿真器放入下载型实验板的 40 脚活动插座中,按一下仿真器上的复位按钮。

(5)按 Ctrl+F5 进入调试状态,再按 F5 全速运行。这时,可以听到悦耳动听的《梁祝》音乐片段不断地反复演奏。

该实验程序在本书所附光盘的 example\ch_6\tim5 文件夹中。

第三节 串行数据通信技术及实验

一、串行数据通信概述

1. 并行通信与串行通信

计算机与其外围设备之间进行数据交换称为通信。通信的基本方式可分为并行通信和串行通信两种。

并行通信是将组成数据的各位同时传送。并通过并行门(如 P1 口等)来实现,如图 6-15 所示为 80C51 系列单片机与外部设备之间 8 位数据并行通信的连接方式。在并行通信中,数据传送线的根数与传送的数据位数相等,传送数据速度快,但所占用的传输线位数多。因此,并行通信适合于短距离通信。

串行通信是指数据一位一位地按顺序传送。串行通信通过串行口来实现。在全双工的串行通信中,仅需一根发送线和一根接收线,图 6-16 所示为 80C51 单片机与外部设备之间串行通信的连接方式,串行通信可大大节省传送线路的成本,但数据传送速度慢。因此,串行通信适合于远距离通信。

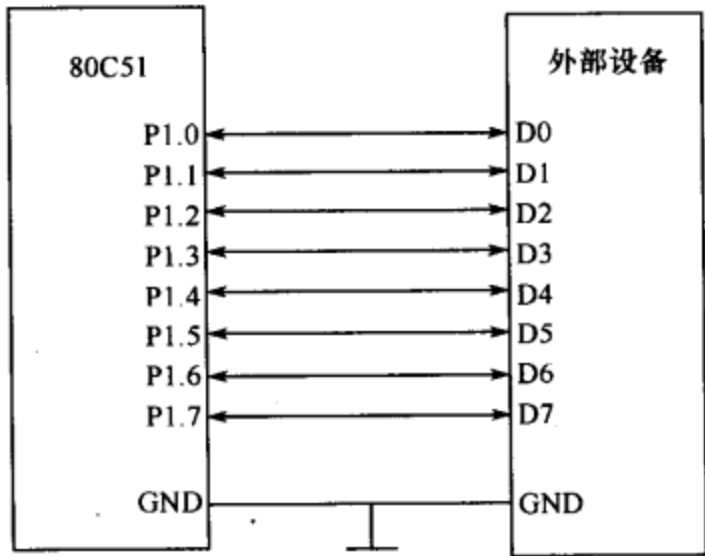


图 6-15 并行通信的连接方式

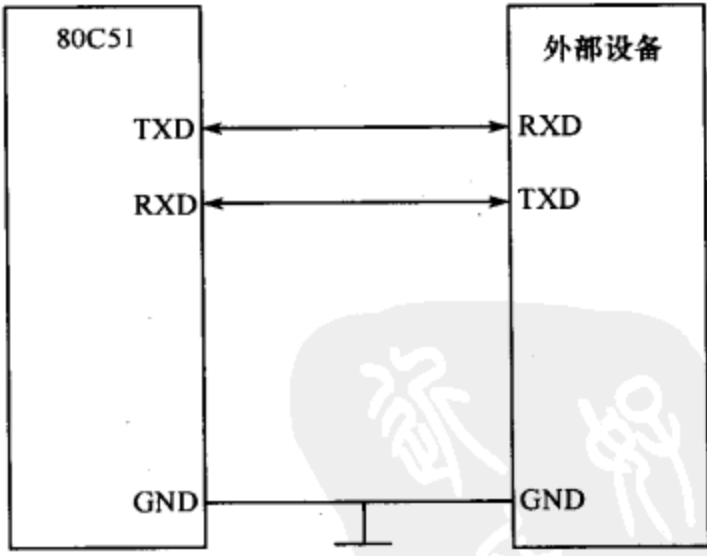


图 6-16 串行通信的连接方式

2. 串行通信的两种基本形式

串行通信根据数据传送时的编码格式不同,分为同步通信和异步通信两种方式。

(1)同步通信。在同步通信中,数据是连续传送的,即数据以数据块为单位传送。在数据开始传送前用同步字符来指示(常约定 1 个或 2 个字符),并由时钟来实现发送端和接收端同步,即检测到规定的同步字符后,下面就连续按顺序传送或接收数据,直到数据

传送结束为止。在同步传送时,数据与数据之间没有间隙。其典型格式如图 6-17 所示。

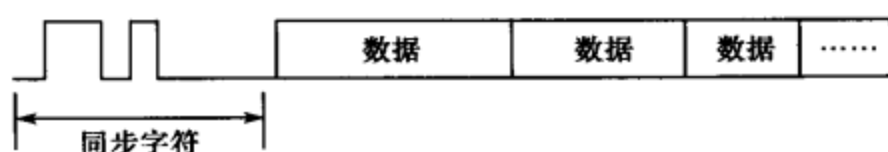


图 6-17 同步传送数据格式

在同步传送中,接收端和发送端必须有同步时钟进行严格同步。在发送时要插入同步字符,接收端检测到同步字符后,便可接收串行数据位。发送端在数据流发送过程中,若出现数据没准备好的情况,便用同步字符来填充,直到下一字符准备好。

(2)异步通信。在异步通信中,数据是不连续传送的。它以字符为单位进行传送,各个字符可以是连续传送,也可以是间断传送。每个被传送字节数据由 4 部分组成:起始位、数据位、校验位和停止位,这 4 部分在通信中称为一帧。首先是一个起始位(0),它占用 1 位,用低电平表示;数据位 8 位(规定低位在前,高位在后);奇偶检验位只占一位(可省略);最后是停止位(1),停止位表示一个被传送字传送的结束,它一定是高电平。接收端不断检测传输线的状态,若连续为 1 后,下一位测到一个 0,就知道发送出一个新字符,应准备接收。由此可见,字符的起始位还被用做同步接收端的时钟,以保证以后的接收能正确进行。图 6-18 所示为异步传送数据格式。

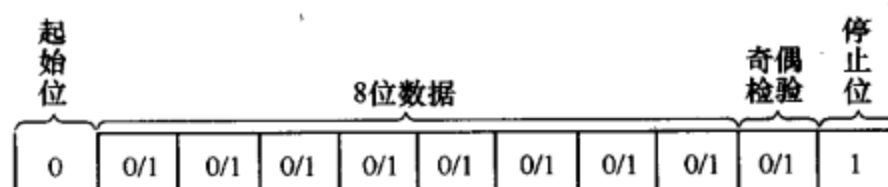


图 6-18 异步传送数据格式

3. 奇偶校验

为了确保传送的数据准确无误,在串行通信中,常在传送过程中进行相应的检测,避免不正确数据被误用。奇偶校验是常用的检测方法。

奇偶校验的工作原理如下:P 是特殊功能寄存器 PSW 的最低位,它的值根据累加器 A 中的运算结果而变化。如果 A 中“1”的个数为偶数,则 $P=0$;如果为奇数,则 $P=1$ 。如果在进行串行通信时,把 A 的值(数据)和 P 的值(代表所传送数据的奇偶性)同时传送,那么接收到数据后,也对数据进行一次奇偶校验。如果校验的结果相符(校验后 $P=0$,而传送过来的数据位也等于 0;或者校验后 $P=1$,而接收到的检验位也等于 1),就认为接收到的数据是正确的。反之,如果对数据校验的结果是 $P=0$,而接收到的校验位等于 1,或者相反,那么就认为接收到的数据是错误的。

4. 串行通信的传输速率

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

例如,在同步通信中传送数据速度为 450 字符/s,每个字符又包含 10 位,则波特率为:450 字符/s \times 10 位/字符 = 4500 位/秒(b/s),一般串行通信的波特率为 50b/s ~ 9600b/s。

5. 单工、半双工与全双工通信

在通信线路上按数据传输方向划分有单工、半双工和双工通信方式。

(1)单工通信。单工通信指传送的信息始终是同一方向,而不能进行反向传送,设备无发送权。

(2)半双工通信。半双工通信是指信息流可在两个方向上传输,但同一时刻只能有一个站发送,两个方向上的数据传送不能同时进行。

(3)全双工通信。全双工通信是指同时可以作双向通信,两个既可同时发送、接收,又可同时接收、发送。

单工通信、半双工通信和全双工通信示意图如图 6-19 所示。

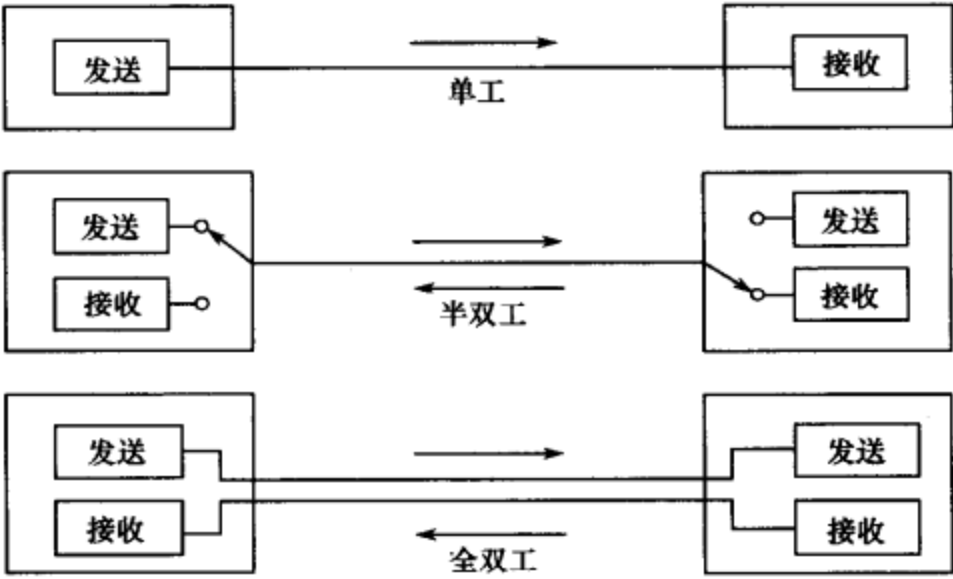


图 6-19 单工通信、半双工通信和全双工通信示意图

6. 串行通信总线标准接口

标准接口是指明确定义若干信号线,使接口电路标准化、通用化的接口,借助串行通信标准接口,不同类型的数据通信设备可以很容易实现它们之间的串行通信连接。采用标准接口后,能很方便地把各种计算机、外部设备、单片机等有机地连接起来,进行串行通信。标准异步串行通信接口有 RS-232C、RS-422、RS-423 和 RS-485 等。其中 RS-232C 是由美国电子工业协会(EIA)正式公布的、在异步串行通信中应用最广的标准总线,它包括了按位串行传输的电气和机械方面的规定,适合于短距离或带调制解调器的通信场合。

RS-232C 中的 RS 是英文“推荐标准”的缩写,232 为标识号,C 表示修改次数。RS-232C 总线标准规定了 21 个信号和 25 个引脚,包括一个主通道和一个辅助通道,在多数情况下主要使用主通道。对于一般双工通信,仅需几条信号线就可实现,包括一条发送线、一条接收线和一条地线。

RS-232C 标准规定的数据传输速率为 50b/s、75b/s、100b/s、150b/s、300b/s、600b/s、1200b/s、2400b/s、4800b/s、9600b/s、19200b/s。驱动器允许有 2500pF 的电容负载,通信距离将受此电容限制。信号传输速率为 20Kb/s 时,最大传输距离为 15m。传输距离短的另一原因是 RS-232 属单端信号传送,存在共地噪声和不能抑制共模干扰等问题,因此一般用于短距离通信。

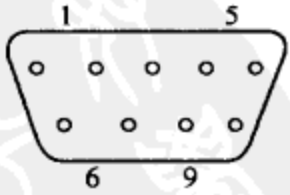


图 6-20 计算机 9 芯串口引脚排列图

图 6-20 是计算机 9 芯串口引脚排列图,表 6-5 是计算机 9 芯引脚信号功能。

表 6-5 计算机 9 芯串口引脚功能

脚号	信号名称	方向	信号功能
1	DCD	PC 机←对方	PC 机收到远程信号(载波检测)
2	RXD	PC 机←对方	PC 机接收数据
3	TXD	PC 机→对方	PC 机发送数据
4	DTR	PC 机→对方	PC 机准备就绪
5	GND	—	信号地
6	DSR	PC 机←对方	对方准备就绪
7	RTS	PC 机→对方	PC 机请求接收数据
8	CTS	PC 机←对方	双方已切换到接收状态(清除发送)
9	RI	PC 机←对方	通知 PC 机,线路正常(振铃指示)

由于 RS-232C 是早期(1969 年)为促进公用电话网络进行数据通信而制定的标准,其逻辑电平对地是对称的,与 TTL、MOS 逻辑电平完全不同。逻辑 0 电平规定为 +5V~+15V,逻辑 1 电平为 -5V~-15V,因此,RS-232C 与 TTL 电平连接必须经过电平转换,目前,比较常用的方法是直接选用 232 芯片,图 6-21 为 80C51 单片机串行口电平转换电路。

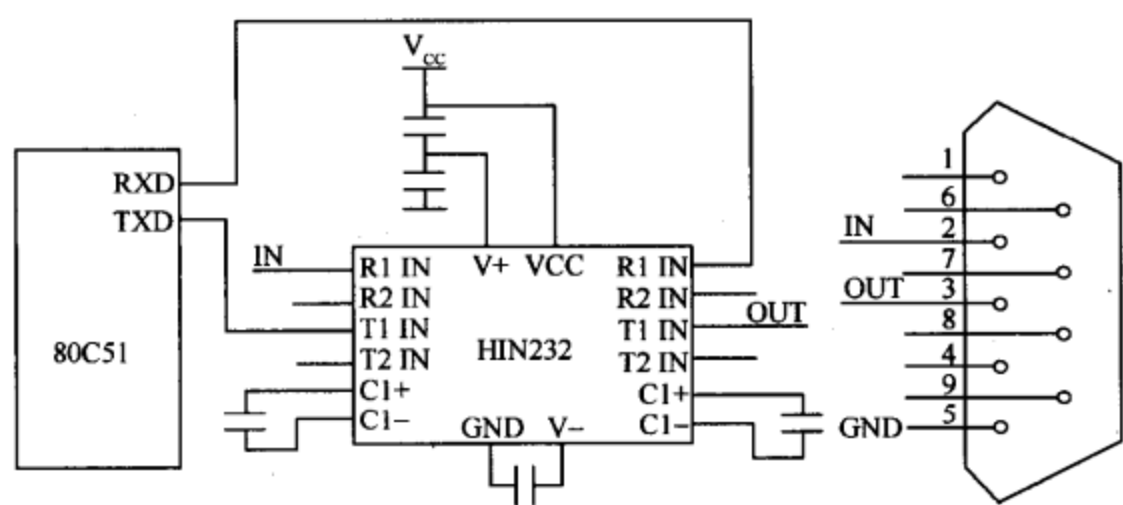


图 6-21 80C51 单片机的串行口电平转换电路

RS-232C 由于发送器和接收器之间具有公共信号地,不能使用双端信号,因此,共模噪声会耦合到信号系统中,这是迫使 RS-232C 使用较高传输电压的主要原因。即使如此,该标准的信号传输速率也只能达到 20Kb/s,而且最大距离仅 15m。只有在这种条件下才能可靠地进行数据传输。

二、串行口的基本结构

串行口电路也称为通用异步收发器(UART),从原理上说,一个 UART 应包括发送器电路、接收器电路和控制电路。80C51 单片机的 UART 已集成在其中,构成一个全双工串行口,这个口既可以用于网络通信,也可以实现串行异步通信,还可以作为同步移位

寄存器使用。

80C51 的串行口通过引脚 RXD(P3.0, 串行口数据接收端)和引脚 TXD(P3.1, 串行口数据发送端)与外部设备之间进行串行通信。图 6-22 所示为 80C51 单片机内部串行口结构示意图。

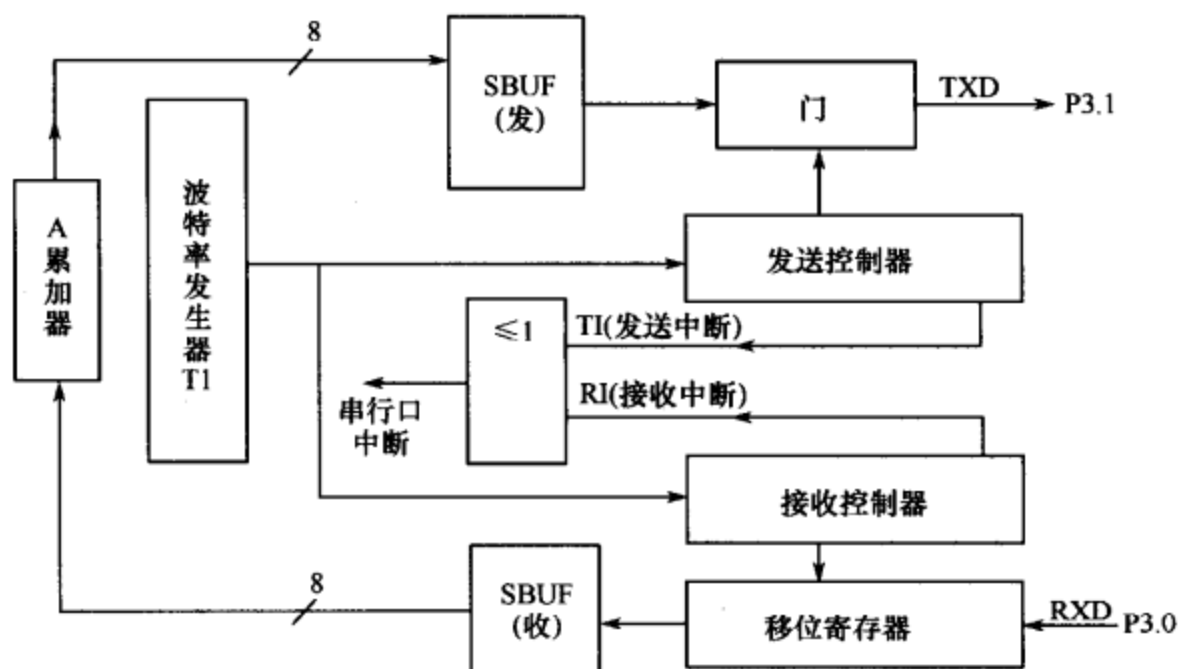


图 6-22 单片机内部串行口结构示意图

图 6-22 中共有两个串行口缓冲寄存器(SBUF)，一个是发送寄存器，一个是接收寄存器，以便 80C51 能以全双工方式进行通信。串行发送时，从片内总线向发送 SBUF 写入数据；串行接收时，从接收 SBUF 向片内总线读出数据。它们都是可寻址的寄存器，但因为发送与接收不能同时进行，所以给这两个寄存器赋以同一地址 99H。

在接收方式下，串行数据通过引脚 RXD(P3.0)进入，由于在接收寄存器之前还有移位寄存器，从而构成了串行接收的双缓冲结构，以避免在数据接收过程中出现帧重叠错误，即在下一帧数据来时，前一帧数据还没有读走。

在发送方式下，串行数据通过引脚 TXD(P3.1)进出。与接收数据情况不同，发送数据时，由于 CPU 是主动的，不会发生帧重叠错误，因此发送电路就不需双重缓冲结构，这样可以提高数据发送速度。

三、串行通信控制寄存器

串行口的通信由 3 个特殊功能寄存器对数据的接收和发送进行控制。它们分别是串行口控制寄存器 SCON、电源控制寄存器 PCON 和中断控制允许寄存器 IE。

1. 串行口控制寄存器 SCON

串行口控制寄存器 SCON 地址 98H，位地址 9FH~98H，具体格式如下：

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

在前面学习中断时，已对 SCON 的 RI 和 TI 位进行了简要分析，为便于读者对 SCON 有一个全面的了解，下面对 SCON 各位的控制功能一同进行分析。

1) SM0、SM1——串行口工作方式选择位

SM0、SM1 对应的 4 种通信方式如表 6-6 所列(表中, f_{osc} 为晶振频率)。

表 6-6 串行口工作方式

SM0	SM1	工作方式	功能	波特率
0	0	方式 0	8 位同步移位方式	$f_{osc}/12$
0	1	方式 1	10 位 UART	可变
1	0	方式 2	11 位 UART	$f_{osc}/32$ 或 $f_{osc}/64$
1	1	方式 3	11 位 UART	可变

2) SM2——多机通信控制位

该位为多机通信控制位,主要用于工作方式 2 和工作方式 3。

当 SM2 为 0 时,则接收到的第 9 位数据(RB8)无论是 0 还是 1,都将接收到的数据装入 SBUF 中,在接收完当前帧后,产生中断申请。

当 SM2 为 1 时,则只有当接收到的第 9 位数据(RB8)为 1,才将接收到的数据装入 SBUF 中,在接收完当前帧后,产生中断申请。若接收到的第 9 位数据(RB8)为 0,则接收到的前 8 位数据丢弃,且不产生中断申请。

在方式 0 时,SM2 必须为 0。

3) REN——允许接收位

由软件置位或清 0,只有当 REN=1 时,才允许接收,它相当于串行接收的开关;若 REN=0 时,则禁止接收。

在串行通信过程中,如果满足 REN=1 且 RI=1,则启动一次接收过程,一帧数据就装入接收缓冲器 SBUF 中。

4) TB8——发送数据位 8

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

在工作方式 1 和工作方式 1 中,该位未用。

5) RB8——接收数据位 8

RB8 是接收数据的第 9 位,在工作方式 2 和工作方式 3 中,接收数据的第 9 位数据放在 RB8 中,它可能是约定的奇偶校验位,也可能是地址/数据标志等。

在工作方式 1 中,RB8 存放的是接收的停止位。

在工作方式 0 中,该位未用。

6) TI——发送中断标志

当方式 0 时,发送完第 8 位数据后,该位由硬件置 1,在其他方式下,于发送停止位之前,由硬件置 1,因此 TI=1,表示帧发送结束,其状态既可供软件查询使用,也可请求中断。TI 位必须由软件清 0。

7) RI——接收中断标志

当方式 0 时,接收完第 8 位数据后,该位由硬件置 1,在其他方式下,当接收到停止位时,该位由硬件置 1,因此 RI=1,表示帧接收结束。其状态既可供软件查询使用,也可以请求中断。RI 位也必须由软件清 0。

2. 电源控制寄存器 PCON

PCON 主要是为 CHMOS 型单片机 80C51 的电源控制而设置的专用寄存器。单元地址为 87H,不能位寻址。其格式如下:

位号	D7	D6	D5	D4	D3	D2	D1	D0
位符号	SMOD	—	—	—	GF1	GF0	PD	ID

电源控制寄存器 PCON 中,与串行口工作有关的仅有它的最高位 SMOD,SMOD 称为串行口的波特率倍增位。当 SMOD=1 时,波特率加倍;系统复位时,SMOD=0。

3. 中断允许寄存器 IE

中断允许寄存器 IE 在前面已作了简要介绍,其中,与串行通信有关的位有 ES 位,ES 为串行中断允许位。ES=0,禁止串行中断,ES=1,允许串行中断。

归纳总结 为便于读者对中断、定时/计数器和串行通信有关控制寄存器 TCON、IE、IP、TMOD、SCON 和 PCON 有一个全面的认识,下面列表对照进行说明,如表 6-7 所列。

表 6-7 与中断、定时/计数器和串行通信有关的寄存器功能一览表

寄存器	位符号	功能	备 注
定时器 控制寄存器 TCON(88H) (可位寻址)	TF0(TF1)	计数溢出标志位	溢出时由硬件自动置 1; 使用查询方式查询有效后用软件清 0; 使用中断方式转向中断服务程序时由硬件自动清 0
	TR0(TR1)	定时器运行控制位	由软件进行设置。 TR0(TR1)=1,启动定时/计数器工作; TR0(TR1)=0,停止定时/计数器工作
	IE0(IE1)	外中断请求标志位	有外中断时,由硬件自动置 1; 中断响应完成后,由硬件自动清 0
	IT0(IT1)	外中断请求触发方式控制位	由软件进行设置。 IT0(IT1)=1,脉冲触发方式; IT0(IT1)=0,电平触发方式
中断允许控制寄存器 IE(0A8H) 可位寻址	EA	中断总允许控制位	由软件进行设置。 EA=0,中断总禁止; EA=1,中断总允许
	EX0(EX1)	外中断允许控制位	由软件进行设置。 EX0(EX1)=0,禁止外中断; EX0(EX1)=1,允许外中断
	ET0(ET1)	定时中断允许控制位	由软件进行设置。 ET0(ET1)=0,禁止定时中断; ET0(ET1)=1,允许定时中断
	ES	串行中断允许控制位	由软件进行设置。 ES=0,禁止串行中断; ES=1,允许串行中断

(续)

寄存器	位符号	功能	备 注
中断优先控制寄存器 IP(0B8H) 可位寻址	PX0(PX1)	外中断优先级 设定位	由软件进行设置。 PX0(PX1)=1, 优先级高; PX0(PX1)=0, 优先级低
	PT0(PT1)	定时中断优先 级设定位	由软件进行设置。 PT0(PT1)=1, 优先级高; PT0(PT1)=0, 优先级低
	PS	串行中断优先 级设定位	由软件进行设置。 PS=1, 优先级高; PS=0, 优先级低
定时器工作方式 控制寄存器 TMOD(89H) 不能位寻址	M1 M0	工作方式选择 位	由软件在字节中设置。 M1 M0=0 0, 工作方式 0; M1 M0=0 1, 工作方式 1; M1 M0=1 0, 工作方式 2; M1 M0=1 1, 工作方式 3
	GATE	门控位	由软件在字节中设置。 GATE=0, 由 TR0(TR1)启动定时器; GATE=1, 由外中断请求信号启动定时器
	C/T	定时或计数方 式选择位	由软件在字节中设置。 C/T=0, 定时工作方式; C/T=1, 计数工作方式
串行口控制寄存器 SCON(98H) 可位寻址	TI(RI)	串行口发送(接 收)中断请求标 志位	帧发送(接收)完由硬件自动置 1; 转向中断服务程序后用软件清 0
	SM0 SM1	串行口工作方 式选择位	由软件进行设置。 SM0 SM1=0 0, 工作方式 0; SM0 SM1=0 1, 工作方式 1; SM0 SM1=1 0, 工作方式 2; SM0 SM1=1 1, 工作方式 3
	SM2	多机通信控制 位	由软件进行设置, 主要用于工作方式 2 和工作方式 3。 SM2=1, 只有接收到的第 9 位数据为 1, 才将接收到的 前 8 位送入 SUBF, 并置位 RI 产生中断; SM2=0, 不论第 9 位数据为 1 还是 0, 都将接收到的 前 8 位送入 SUBF, 并置位 RI 产生中断
	REN	允许接收位	由软件进行设置。 REN=0, 禁止接收; REN=1, 允许接收

(续)

寄存器	位符号	功能	备 注
串行口控制寄存器 SCON(98H) 可位寻址	TB8	发送数据位 8	由软件进行设置。 在工作方式 2、方式 3 时,存放要发送的第 9 位数据
	RB8	接收数据位 8	由软件进行设置。在工作方式 2、方式 3 时,存放接收到的第 9 位数据
电源控制及波特率 选择寄存器 PCON(87H) 不可位寻址	SMOD	串行口波特率 倍增位	由软件在字节中设置。 SMOD=1 时,串行口波特率加倍, SMOD=0 时,串行口波特率不加倍

四、串行口工作方式

80C51 单片机串行口有 4 种工作方式,分别为工作方式 0、工作方式 1、工作方式 2 和工作方式 3,由串行口控制寄存器 SCON 中最高两位 SM0、SM1 的状态,通过软件设置来决定选择何种工作方式。其中有 8 位、10 位和 11 位为一帧的数据传送格式。

1. 方式 0

方式 0 以 8 位数据为一帧进行传输,不设起始位和停止位,先发送或接收最低位。其一帧数据格式如下:

...	D0	D1	D2	D3	D4	D5	D6	D7	...
-----	----	----	----	----	----	----	----	----	-----

1) 发送与接收

使用方式 0 实现数据的移位输入/输出时,实际上是把串行口变成为并行口使用。串行口作为并行输出口使用时,要有“串入并出”的移位寄存器(例如 CD4094、74LS164 等)配合,图 6-23 是采用 CD4094 的连接示意图。

数据预先写入串行口数据缓冲寄存器,然后从串行口 RXD 端在移位时钟脉冲 TXD 的控制下逐位移入 CD4094,当 8 位数据全部移出后,SCON 寄存器的发送中断标志 TI 被自动置 1。其后主程序就可以中断或查询的方法,通过设置 STB 状态的控制,把 CD4094 的内容并行输出。

如果把能实现“并入串出”功能的移位寄存器(例如 CD4014、74LS165 等)与串行口配合使用,就可以把串行口变为并行输入口使用。图 6-24 所示是采用 CD4014 的连接示意图。

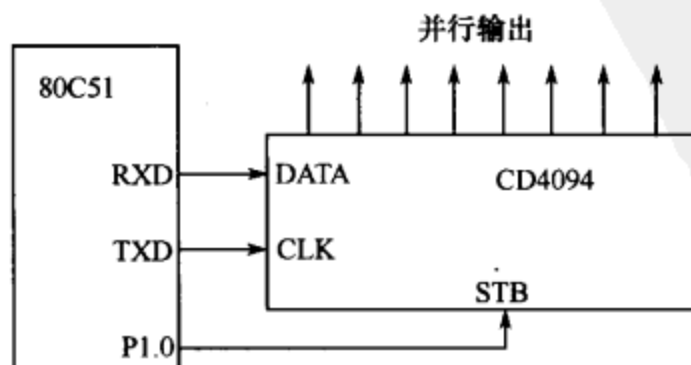


图 6-23 串行口与 CD4094 配合

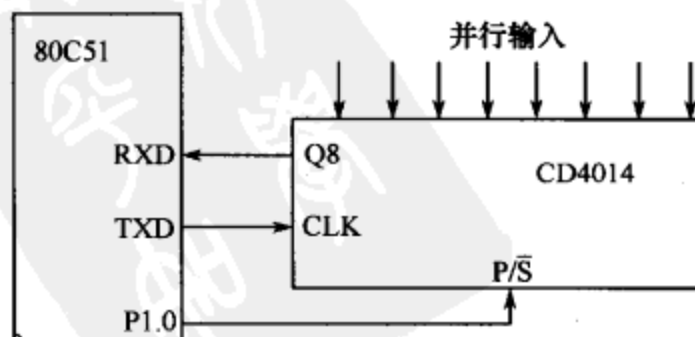


图 6-24 串行口与 CD4014 配合

CD4014 移出的串行数据同样经 RXD 端串行输入,还是由 TXD 端提供移位时钟脉冲。8 位数据串行接收需要有允许接收的控制,具体由 SCON 寄存器的 REN 位实现。REN=0,禁止接收;REN=1,允许接收。当软件置位 REN 时,即开始从 RXD 端输入数据(低位在前),当接收到 8 位数据时,置位接收中断标志 RI。

从以上可以看出,在方式 0 下,串行口为 8 位同步移位寄存器输入/输出方式,这种方式不适合用于两个 80C51 单片机芯片之间的直接数据通信,但可以通过外接移位寄存器来实现单片机的接口扩展。

2)波特率的设定

方式 0 时,移位操作(串入或串出)的波特率是固定的,如图 6-25 所示。波特率为单片机晶振频率的 1/12,如晶振频率以 f_{osc} 表示,则波特率为 $f_{osc}/12$ 。按此波特率也就是一个机器周期进行一次移位,如 $f_{osc}=12\text{MHz}$,则波特率为 1 Mb/s,即 $1\mu\text{s}$ 移位一次。

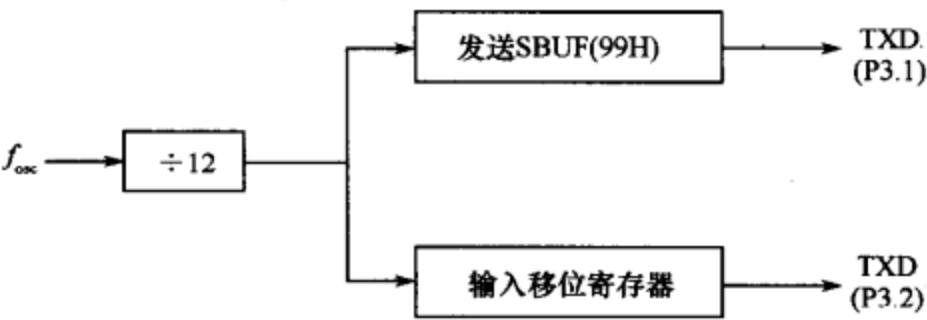


图 6-25 方式 0 波特率产生电路

2. 工作方式 1

工作方式 1 以 10 位数据为一帧进行传输,设有 1 个起始位(0)、8 个数据位,1 个停止位(1),其一帧数据格式如下:

起始	D0	D1	D2	D3	D4	D5	D6	D7	停止
----	----	----	----	----	----	----	----	----	----

1)发送与接收

工作方式 1 为 10 位异步通信接口,TXD 和 RXD 分别用于发送与接收数据。收发一帧数据为 10 位,数据位是先低位,后高位。

发送时,数据从 TXD(P3. 0)端输出,当 TI=0,执行数据写入发送缓冲器 SBUF 指令时,就启动了串行口数据的发送操作。指令如下:

```
MOV SBUF,A
```

启动发送后,串行口自动在起始位清 0,而后是 8 位数据和一位停止位 1,一帧数据为 10 位。数据依次从 TXD 端发出,一帧数据发送完毕,使 TXD 输出线维持在 1 状态下,并将 SCON 寄存器的 TI 置 1,以便查询数据是否发送完毕或作为发送中断申请信号。TI 必须由软件清 0。

接收时,数据从 RXD(P3. 0)端输入,SCON 的 REN 位应处于允许接收状态(REN=1)。在此前提下,串行口采样 RXD 端,当采样到从 1 向 0 的状态跳变时,就认定是接收到起始位。随后在移位脉冲的控制下,把接收到的数据位移入接收寄存器中。直到停止位到来之后把停止位送入 RB8 中,并置位中断标志位 RI,通知 CPU 从 SBUF 取走接收到的一个字符。

2) 波特率的设定

方式 0 的波特率是固定的,但方式 1 的波特率则是可变的,如图 6-26 所示。

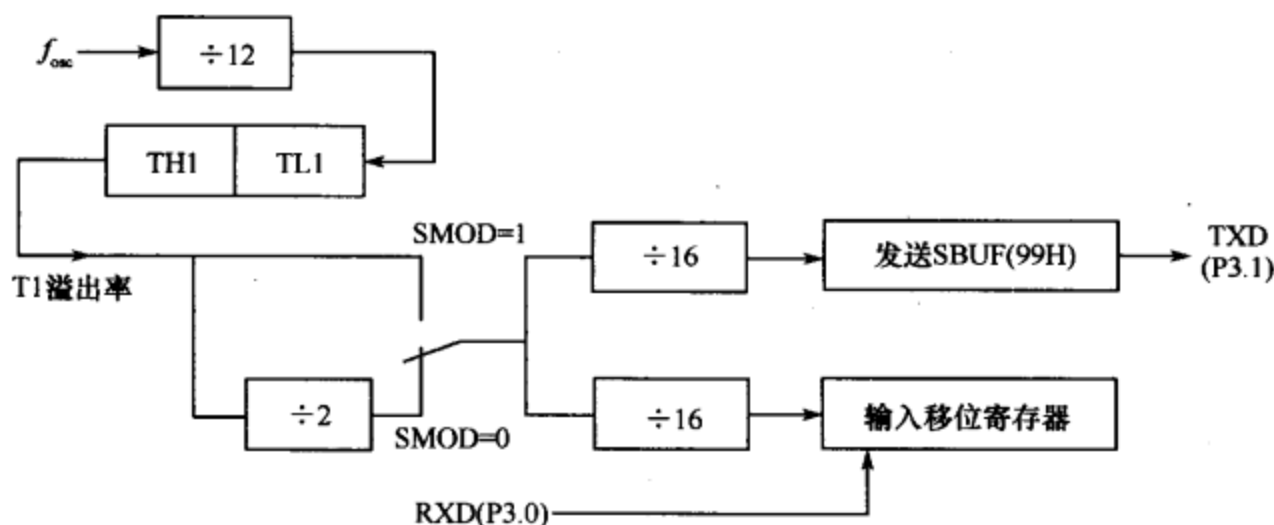


图 6-26 方式 1 的波特率产生电路

以定时器 T1 作为波特率发生器使用,其值由定时器 1 的计数溢出率来决定,其公式为

$$\text{波特率} = \frac{2^{\text{smod}}}{32} \times \text{定时器 T1 溢出率}$$

式中,T1 溢出率为一次定时时间的倒数,即

$$\text{定时器 T1 溢出率} = \frac{1}{(2^M - X) \times \frac{12}{f_{\text{osc}}}} = \frac{f_{\text{osc}}}{(2^M - X) \times 12}$$

式中,X 为计数初值;M 由定时器 T1 的工作方式所决定,即 M=8、13 或 16。当定时器 1 作波特率发生器使用时,一般选用工作方式 2,之所以选择工作方式 2,是因为方式 2 具有自动加载功能,可避免通过程序反复装入初值所引起的定时误差,使波特率更加稳定。因此,对于定时器 T1 的工作方式 2,定时器 T1 的溢出率又可简化为

$$\text{定时器 T1 溢出率} = \frac{f_{\text{osc}}}{(256 - X) \times 12}$$

此时,波特率为

$$\text{波特率} = \frac{2^{\text{smod}}}{32} \times \frac{f_{\text{osc}}}{(256 - X) \times 12} = \frac{2^{\text{smod}} \times f_{\text{osc}}}{384 \times (256 - X)}$$

因此,计数初值 X 为

$$X = 256 - \frac{2^{\text{smod}} \times f_{\text{osc}}}{384 \times \text{波特率}}$$

例如,设两机通信的波特率为 2400b/s,若 $f_{\text{osc}} = 12\text{MHz}$,串行口工作在方式 1,用定时器 T1 作为波特率发生器,选定时器工作在方式 2(要禁止 T1 中断,以免产生不必要的中断带来频率误差)。

若 SMOD=1,则计数初值 X 为

$$X = 256 - \frac{2^{\text{smod}} \times f_{\text{osc}}}{384 \times \text{波特率}} = 256 - \frac{2 \times 12 \times 10^6}{384 \times 2400} \approx 230\text{D} = 0\text{E6H}$$

若 SMOD=0,则计数初值 X 为

$$X = 256 - \frac{2^{\text{SMOD}} \times f_{\text{osc}}}{384 \times \text{波特率}} = 256 - \frac{1 \times 12 \times 10^6}{384 \times 2400} \approx 243\text{D} = 0\text{F3H}$$

3. 工作方式 2

工作方式 2 是 11 位为一帧的串行通信方式,即 1 个起始位、9 个数据位和 1 个停止位。其帧格式为

起始	D0	D1	D2	D3	D4	D5	D6	D7	D8	停止
----	----	----	----	----	----	----	----	----	----	----

1) 发送和接收

在工作方式 2 下,字符还是 8 个数据位,只不过增加了一个第 9 个数据位(D8),而且其功能由用户确定,是一个可编程位。

在发送数据时,应预先在 SCON 的 TB8 位中把第 9 个数据位的内容准备好。这可使用如下指令完成:

SETB TB8 ; TB8 位置 1

CLR TB8 ; TB8 位清 0

发送数据(D0~D7)由 MOV 指令向 SBUF 写入,而 D8 位的内容分别由硬件电路从 TB8 中直接送到发送移位寄存器的第 9 位,并以此来启动串行发送。一个字符帧发送完毕后,将 TI 位置 1,其他过程与工作方式 1 相同。

工作方式 2 的接收过程也与方式 1 基本类似,所不同的只在第 9 数据位上,串行口把接收到的前 8 个数据位送入 SBUF,而把第 9 数据位送入 RB8。

2) 波特率的设定

工作方式 2 的波特率是固定的,且有两种,如图 6-27 所示。

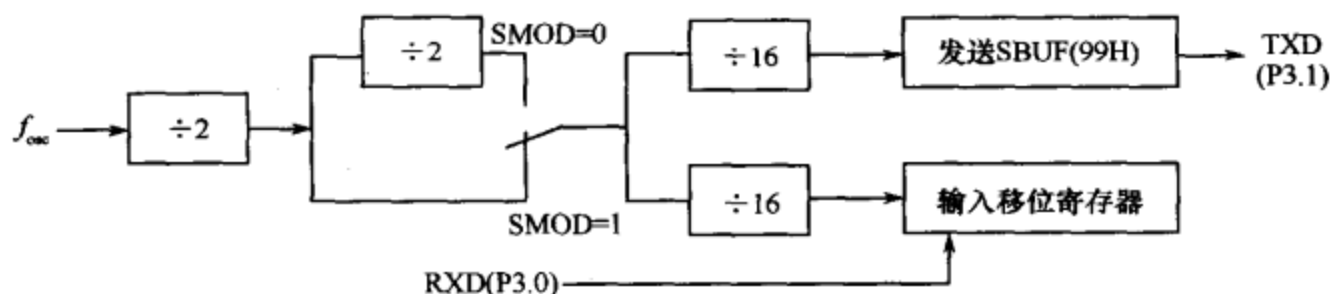


图 6-27 工作方式 2 的波特率产生电路

工作方式 2 的波特率与 PCON 寄存器中 SMOD 位的值有关。当 SMOD=0 时,波特率为 f_{osc} 的 1/64;当 SMOD=1 时,波特率等于 f_{osc} 的 1/32。

4. 工作方式 3

工作方式 3 是 11 位为一帧的串行通信方式,其通信过程与工作方式 2 完全相同,所不同的仅是在于波特率:工作方式 2 的波特率只有固定的两种,而工作方式 3 的波特率则可由用户根据需要设定,其设定方法与工作方式 1 相同,即通过设置定时器 1 的初值来设置波特率。

重点提示 根据以上介绍可知,串行通信工作方式 0 和工作方式 1 的波特率是固定的,不需要计算计数初值,而串行通信的工作方式 2 和工作方式 3,则需要计算定时器的计数初值(一般采用定时器的工作方式 2),以便设置波特率。

五、串行数据通信应用举例

例 1 图 6-28 所示电路中,CD4094 的各个输出端均接一个发光二极管(共阴接法),

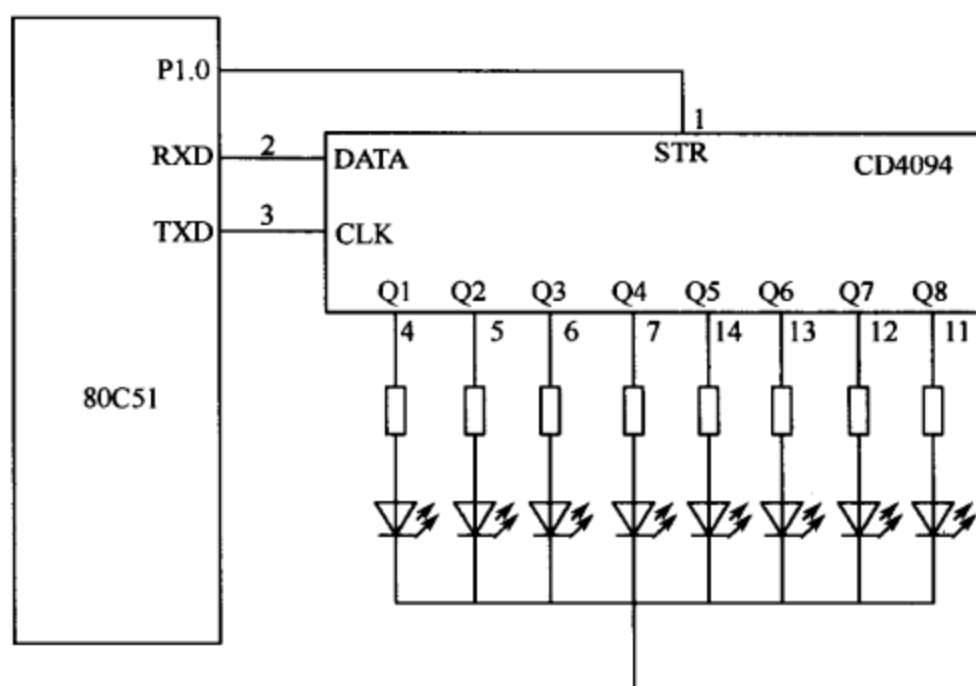


图 6-28 例 1 电路图

要求采用方式 0 使发光二极管呈流水灯显示。

CD4094 是 8 位移位/锁存总线寄存器芯片。CLK 是时钟端, DATA 是数据端。当有时钟上升沿到来时, DATA 引脚上的状态进入 CD4094 内部的移位寄存器, 同时移位寄存器向前移一位。STR 引脚是锁存端。如果这一位是 0, 则并行输出端保持不变, 但是串行数据依然可以进入移位寄存器。数据输入时首先变化的是 Q1, 即最先到达的数据位会被移到 Q8, 而最后到达的数据位则由 Q1 输出。

串行口输出时, 最先送出的是最低位的数据。由图中可以看出, Q1 接的 LED 在最左边, 而 Q8 接的 LED 在最右边, 所以应当先送一个二进制数 10000000 (这样, 1 会被送到最左边的 Q1 点亮最左边的 LED); 然后延迟一段时间, 将数进行右移, 即变为 01000000, 再次送入 CD4094, 第 2 只 LED 点亮。如此不断右移, 数据就依次按 1000 0000, 0100 0000, 0010 0000, 0001 0000... 变化, 也就是灯按从左到右流水显示。

说明 在数据送出之前, 首先将 STR 清 0, 以保持输出端不发生变化, 在数据送完之后再将 STR 置位, 以送出数据进行显示。否则, 当数据在 CD4094 内部的移位寄存器中移动时, 同时也会反映到输出引脚上, 造成输出引脚的电平产生不希望的变化。从现象上来说, 这会造成本不应当显示的 LED 会产生一些微弱的显示。

采用中断方式的源程序如下:

```

ORG 0000H
AJMP MAIN
ORG 0023H
AJMP PS1
;以下是主程序
ORG 200H
MAIN: MOV SCON, #00H
      MOV A, #80H
      SETB EA

```

; 串行口中断入口地址
; 转中断服务程序
; 设定串行口于工作方式 0
; 输入数据 1000 0000
; 总中断允许

SETB ES	;允许串行口中断
LOOP: CLR P1.0	;关闭并行输出
MOV SBUF,A	;开始串行输出
HERE: SJMP HERE	;等待中断
;以下是中断服务程序	
PS1: CLR TI	;清 TI 标志位,为下次做准备
SETB P1.0	;开并行输出
ACALL DELAY	;调延时子程序
RR A	;右移至下一位 LED
CLR P1.0	;关闭并行输出
MOV SBUF,A	;重新进行串行输出
RETI	
;以下是延时子程序	
DELAY: MOV R5,#0FAH	
D2: MOV R4,#0FAH	
D1: NOP	
NOP	
DJNZ R4,D1	
DJNZ R5,D2	
RET	
END	
采用查询方式的源程序如下:	
ORG 0000H	
AJMP MAIN	
;以下是主程序	
ORG 200H	
MAIN: MOV SCON,#00H	;置串行口于工作方式 0
MOV A,#80H	;输入数据 1000 0000
LOOP: CLR P1.0	;关闭并行输出
MOV SBUF,A	;开始串行输出
LOOP1: JBC TI,NEXT	;若 TI 为 1,则转 NEXT,并清 TI
AJMP LOOP1	;否则再查询
NEXT: SETB P1.0	;开并行输出
CALL DELAY	;调延时子程序
RR A	;右移至下一位 LED
AJMP LOOP	;循环
;以下是延时子程序	
DELAY: MOV R5,#0FAH	
D2: MOV R4,#0FAH	

```

D1:    NOP
      NOP
      DJNZ R4,D1
      DJNZ R5,D2
      RET
      END

```

方法技巧 数据传送可采用中断方式,也可采用查询方式,无论哪种方式,都要借助于 TI 或 RI 标志。在串行口发送时,当 TI 置 1(发完一帧数据后)后向 CPU 申请中断,在中断服务程序中要用软件清零 TI,以便发送下一帧数据。在采用查询方式时,CPU 不断查询 TI 的状态,只要 TI 为 0 就继续查询,TI 为 1 就结束查询,TI 为 1 后也要及时用软件清零 TI,以便发送下一帧数据。

在串行接收时,则由 RI 引起中断或对 RI 查询来确定何时接收下一帧数据。无论采用什么方式,在开始通信之前,都要先对控制寄存器 SCON 进行初始化,进行工作方式的设定。在方式 0 中,SCON 寄存器中的控制字为 00H。

例 2 设甲、乙两台单片机,连接情况如图 6-29 所示。试按以下要求设计程序:

甲机发送,要求将片内 RAM 中的 30H~3FH 单元中的数据由串行口发送,串行口设定为工作方式 2,TB8 控制位作为奇偶校验位。

乙机接收,要求甲机发送的数据送乙机片内 RAM 的 50H~5FH 单元中,采用工作方式 2,并核对奇偶校验位。

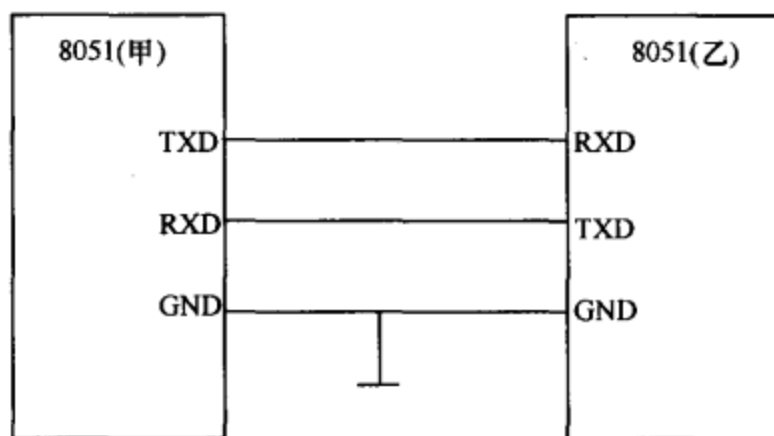


图 6-29 甲乙两机的连接

1) 甲机发送程序

根据串行口工作方式 2 的特点,在数据写入 SBUF 之前,应先将数据的奇偶标志 P 写入 SCON 中的 TB8,因此,发送数据的第 9 位便可用做奇偶校验。

采用查询方式的发送源程序如下:

```

      ORG 0000H
      AJMP MAIN          ;转主程序
      ORG 200H
MAIN: MOV  SCON, #80H     ;设串行口为工作方式 2
      MOV  PCON, #80H    ;置 SMOD 为 1,使波特率倍增
      MOV  R0, #30H      ;给地址指针 R0 赋初值

```

MOV R7, #10H	;设置计数器初值为 16
LOOP: MOV A, @R0	;将 R0 指向的存储单元内容送 A, 并建立奇偶标志
MOV C, P	;取奇偶校验标志
MOV TB8, C	;奇偶标志送 TB8
MOV SBUF, A	;发送数据
HERE: JBC TI, CONT	;TI=1 转 CONT, 并清 TI
AJMP HERE	;等待中断标志 TI 变 1
CONT: INC R0	;地址指针 R0 加 1, 指向下一存储单元
DJNZ R7, LOOP	;数据尚未发送完, 继续
SJMP \$	
END	

2) 乙机接收程序

采用查询方式的乙机接收源程序如下:

ORG 0000H	
AJMP MAIN	;转主程序
ORG 200H	
MAIN: MOV SCON, #90H	;设置串行口为方式 2, 置 REN=1
MOV R0, #50H	;给地址指针 R0 赋初值
MOV R7, #10H	;设置计数器初值为 16
HERE: JBC RI, REC	;有接收中断申请(RI=1)时转 REC, 并清 RI
SJMP HERE	
REC: MOV A, SBUF	;将接收缓冲器的数据送 A
JNB P, REC1	;若 P 为 0 转 REC1, 看接收到的 RB8 为 0 还是为 1
JNB RB8, ERR	;接收的数据错误, 转 ERR
SJMP RIGHT	;接收的数据正确, 转 RIGHT
REC1: JB RB8, ERR	;若 RB8 为 1, 说明 P 和 RB8 不等, 接收错误, 转 ERR
RIGHT: MOV @R0, A	;接收的数据正确, 将 A 的内容送 R0 指向的存储单元
INC R0	;R0 加 1, 指向下一单元地址
DJNZ R7, HERE	;R7 不等 0, 转 HERE
CLR PSW. 5	;清标志位
RET	
ERR: SETB PSW. 5	;接收的数据错误, 置位 PSW 的第 5 位(用户标志位)
RET	
END	

方法技巧 在工作方式 2、工作方式 3 的发送过程中,将数据和附加在 TB8 中的奇偶位一块发向对方。因此,作为接收的一方进行奇偶核对,若接收数据的 RB8 位和奇偶校验位 P 一致,说明接收成功,若 P 与 RB8 不一致,说明接收失败,并在 PSW 寄存器的 D5 位(自定义该位功能)建立接收失败标志。

例 3 设有甲、乙两台单片机,以工作方式 2 全双工串行通信,第 9 位作为奇偶校验位。编出能实现如下功能的程序。

甲机:将片内 RAM 中的 30H~3FH 数据发送到乙机,乙机对接收的数据进行奇偶校验,若校验正确则乙机向甲机发出“数据发送正确”的信息(现取 00H 作为回答信号),甲机接收到乙机的此信息再发送下一个字节。若奇偶校验错,则乙机发出“数据发送不正确”的信息(现取 FFH 作为回答信号)给甲机,要求甲机再次发送原数据,直至数据发送正确。

乙机:将甲机发送的数据存入乙机片内 RAM 的 50H~5FH 单元中,进行奇偶校验,并发出相应的回答信息(即 00H 或 FFH)给甲机。

甲机源程序如下:

```

ORG 0000H
AJMP MAIN                ;调主程序
ORG 0023H                ;串行口中断服务程序入口
AJMP INTSE               ;调中断程序
ORG 200H

;以下是主程序
MAIN: MOV PCON, #80H      ;波特率加倍
      MOV SCON, #90H      ;置工作方式 2, REN=1
      MOV R0, #30H        ;给地址指针 R0 赋初值
      MOV R7, #00H        ;发送字节数初值
      SETB EA             ;CPU 开中断
      SETB ES             ;允许串行口中断
      MOV A, @R0          ;取第 1 个发送数据
      MOV C, P            ;取奇偶校验标志
      MOV TB8, C          ;奇偶校验标志位送 TB8
      MOV SBUF, A         ;启动串行口,发送数据
      SJMP $              ;等待中断

;以下是中断服务程序
INTSE: JB RI, REC         ;若 RI=1, 转 REC, 接收乙机发送的应答信息
                        ;因 RI=0, 表明甲机发送中断请求标志 TI=1, 再清 TI
      CLR TI
                        ;甲机发送一数据完毕, 跳至中断返回指令处
      SJMP ENDT          ;清接收中断标志 RI
REC:  CLR RI              ;取乙机的应答数据
      MOV A, SBUF

```

SUBB A, #01H	;将接收的数据和 01 相减
JC RIGHT	;若有借位,说明乙机应答为 00H,数据传送正确,转 RIGHT
MOV A, @R0	;无借位,说明乙机应答为 FFH,数据不正确,甲应重发
MOV C, P	
MOV TR8, C	
MOV SBUF, A	;启动串行口,重发一次数据
SJMP ENDT	;跳至中断返回程序
RIGHT: INC R0	;修改地址指针
INC R7	;修改发送字节数计数值
MOV A, @R0	;下一个发送数据送 A
MOV C, P	
MOV TR8, C	
MOV SBUF, A	;启动串行口,发送新的数据
CJNE R7, #16, ENDT	;判别 16 数据是否发送完,如没有,中断返回后继续发送
CLR ES	;全部发送完毕,禁止串行口中断
ENDT: RETI	;中断返回
END	
乙机源程序如下:	
ORG 0000H	
AJMP MAIN	;主程序入口
ORG 0023H	;串行中断入口地址
AJMP INTSE	;转至串行口中断服务程序
;以下是主程序	
ORG200H	
MAIN: MOV PCON, #80H	;波特率加倍
MOV SCON, #90H	;工作方式 2, REN=1
MOV R0, #50H	;给地址指针 R0 赋初值
MOV R7, #00H	;接收数据字节数初值
SETB EA	;CPU 开中断
SETB ES	;串行口开中断
SJMP \$;等待中断
;以下是串行口中断服务程序	
INTSE: JB RI, REC	;检测是否是本机的接收口中断,若 RI=1,转 REC
CLR TI	;因 RI=0,说明中断是 TI=1 引起发送中断,应清零

SJMP ENDT	;跳至中断返回程序,继续接收
REC: CLR RI	;清本机的接收中断标志
MOV A,SBUF	;取接收的数据
MOV C,P	;将奇偶标志位送 C
JC REC1	;若 C=1,说明接收数据 1 的个数为奇数,转 REC1,再检测 RB8 位
ORL C,RB8	;若接收数据 1 的个数为偶(P=0),使 P 与 RB8 相或
JC ERR	;若 P 与 RB8 相或后为 1,说明数据不正确,转 ERR
SJMP RIGHT	;若 P 与 RB8 相或后为 0,说明数据正确,转 RIGHT
REC1: ANL C,RB8	;将 RB8 与 C 中的奇偶校验位 P 相与后送 C
JC RIGHT	;若相与后 C 为 1,说明 RB8=P=1,校验正确,转 RIGHT
ERR: MOV A,#FFH	;本机发出应答信息(FFH)给甲机,表明数据传送不正确
MOV SBUF,A	
SJMP ENDT	;跳至中断返回程序
RIGHT: MOV @R0,A	;将接收的正确数据送数据缓冲区
MOV A,#00H	;送正确应答信息 00H 给 A
MOV SBUF,A	;将 00H 送 SBUF,表明数据传送正确,甲机应发送下一个数据
INC R7	;修改指针
INC R0	
CJNE R7,#16,ENDT	;判别 16 个数据是否接收完,若没有,继续接收
CLR ES	;接收完毕,关串行口中断
ENDT: RETI	;中断返回
END	

例 4 如图 6-30 所示的多机通信电路中,主机可以和 3 个从机进行串行通信,试设计主机和 3 个从机(设为 1 号、2 号和 3 号)的通信程序,要求采用串行工作方式 3,将主机的 20H~25H 中 6 个单元的数据分别送给 1、2、3 号从机的 30H 和 31H 中,设通信波特率为 9600b/s,单片机晶振为 11.0592MHz。

单片机的多机串行通信过程和课堂上教师提问学生的过程差不多。教师提问学生前,先点某个学生的名字;然后,所有的学生都把教师点的这个名字和他位的名字比较,其中必有一个学生发现这个名字是他的名字;然后,他就从座位上站起来,准备回答老师的问题,而其余的学生发现这个名字与自己无关,则他们都不用站起来;然后,教师就开始提问。教师提问时,所有的学生都听见了,但只有站起来的那个学生对提问的问题作出响

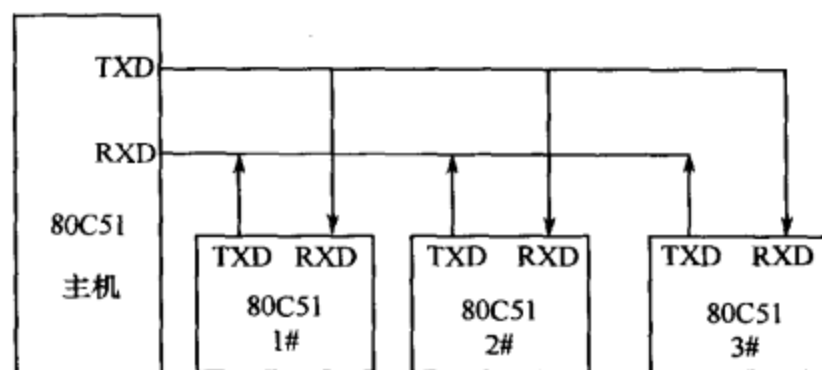


图 6-30 多机串行通信电路

应,当教师提问一会后,他可能想换一个学生提问,这时,他再点一个学生的名字,则这次被点的那个学生站起来,其余的学生坐下。

在单片机多机通信中同样如此,主机相当于教师,从机相当于学生。通信前,主机发出一个第 9 位(TB8)为 1 的地址,相当于教师点一个学生的名字。由于从机接收到的该地址第 9 位(RB8)为 1,所以,不论从机的 SM2 位为 1 还是 0,将接收到的地址装入 SBUF 中,在接收完当前帧后,产生中断申请,相当于所有的学生都听见了教师的点名。其中必然有一台从机会发现接收到的地址和它本身保存在存储器中的从机号相同,相当于其中有一个学生判断出教师要对他提问;则该从机将其多机通信控制位 SM2 置 0,相当于这个学生从座位上站起来;这样才能接收主机发送的第 9 位(TB8)为 0 的数据,相当于只有站起来的学生才能回答教师的提问;而其余从机肯定会发现他们接收到的地址与他们的从机号不相符,则这些从机都将其多机通信控制位 SM2 置 1,不能接收主机发送的第 9 位(TB8)为 0 的数据,相当于其余学生全部坐下,不能回答教师的提问。

主机在发送一个从机的地址后,紧接着把发往该从机的数据依次发出,每个数据的第 9 位(TB8)都为 0,这相当于教师提问;每个从机都检测到了这些数据,但只有 SM2 为 0 的从机才将这些数据装入接收缓冲器并申请中断,让 CPU 处理这些数据;而其余的从机因为 SM2 为 1,所以将收到的这些数据丢失,相当于只有站起来的学生才能回答这个问题。

下面计算一下串行方式 3 的计数初值,设 PCON 的 SMOD 为 0,则计数初值 X 为

$$X = 256 - \frac{2^{\text{smod}} \times f_{\text{osc}}}{384 \times \text{波特率}} = 256 - \frac{1 \times 11.0592 \times 10^6}{384 \times 9600} = 253\text{D} = 0\text{FDH}$$

根据以上分析和计算,编制的主机源程序和 3 个从机源程序如下:

主机源程序:

```
ORG 0000H
AJMP MAIN          ;转主程序
ORG 0023H          ;串行口中断入口地址
AJMP SUB1          ;转串行中断服务程序
```

;以下是主程序

```
MAIN: MOV SCON, #0F8H ;工作方式 3,置 SM2=1,REN=1,TB8=1,
                        ;RB8=0
      MOV TMOD, #20H   ;设置定时器 T1 为工作方式 2
      MOV TL1, #0FDH   ;设置定时器 T1 的初始值
```



```

MOV TH1, #0FDH
SETB EA ;开总中断
SETB ES ;开串行中断
SETB TR1 ;启动定时器 T1
MOV SBUF, #1 ;发送第 1 台从机号地址
MOV R0, #20 ;给待发送的地址指针赋初值
MOV R1, #2 ;指出下一次将要发送的从机地址为 1
SJMP $ ;等待串行中断
;以下是串行中断服务程序
SUB1: MOV A, R0
ANL A, #01 ;屏蔽 A 的高 7 位
JNZ LAB1 ;若 A 的内容不为 0,说明 A 是奇数,转数据处理 LAB1
JB TB8, LAB1 ;TB8 为 1 转 LAB1 发送数据, TB8 为 0 顺序执行
SETB TB8 ;发送地址前要将 TB8 置 1
MOV SBUF, R1 ;发送第 2 台从机号地址
INC R1 ;指向下一从机号地址
CLR TI ;清 TI
RETI
LAB1: CLR TB8 ;发送数据前要清 TB8
MOV SBUF, @R0 ;发送数据
INC R0 ;指向下一待发数据的地址
CJNZ R0, #26H, LAB2 ;判断数据是否发送完,不等,转 LAB2
CLR ES ;清串行中断
LAB2: CLR TI
RETI
END

```

方法技巧 在中断服务程序中有两类,一类是地址,一类是数据,每发送一个地址,就要发送两个数据,因此,在中断服务程序中必须判断出当前发送的是地址还是数据。若发送的是地址,就从工作寄存器 R1 中提取;若发送的是数据,就从数据指针 R0 指向的内部存储器中提取。

判断发送的是地址还是数据的方法如下:

如果 R0 为偶数(例如 20H、22H、24H),且此时 TB8 为 0,则应发送地址(发送地址前要将 TB8 置 1),否则,如果 R0 为奇数(例如 21H、23H、25H),且 TB8 为 1,则应发送数据(发送数据前要将 TB8 置 0)。判断奇偶数的方法是将 R0 中的数与 0000 0001 相与,屏蔽掉高 7 位。若相与后 R0 中的数为 0,说明 R0 是偶数,若相与后 R0 中的数是 1,说明 R0 是奇数。例如,22H(0010 0010)与 0000 0001 相与后为 0,说明 22H 为偶数;再如,23H(0010 0011)与 0000 0001 相与后为 0000 0001,因此,23H 是奇数,其他依次类推。

从机 1 的源程序如下:

```
ORG 0000H
AJMP MAIN          ;转主程序
ORG 0023H          ;串行口中断入口地址
AJMP SUB1          ;转串行中断服务程序
NAME EQU #1
;以下是主程序
MAIN: MOV SCON, #0F8H ;工作方式 3,置 SM2=1,REN=1,TB8=1,
                      ;RB8=0
      MOV TMOD, #20H ;设置定时器 T1 为工作方式 2
      MOV TL1, #0FDH ;设置定时器 T1 为初始值
      MOV TH1, #0FDH
      SETB EA        ;开总中断
      SETB ES        ;开串行中断
      SETB TR1       ;启动定时器 T1
      MOV R0, #30    ;给接收机存储器地址指针 R0 赋初值
      SJMP $         ;等待串行中断
;以下是串行中断服务程序
SUB1: JNB RB8, LAB1  ;RB8 为 0,转 LAB1 处理数据,RB8 为 1 顺序
                      ;执行
      MOV A, SBUF     ;RB8 为 1,说明接收到的是地址,送入接收缓
                      ;冲器
      CJNE A, NAME, LAB2 ;判断接收到的地址与本机地址号是否一致,
                        ;不一致转 LAB2
      CLR SM2         ;地址一致,SM2 清 0,以便接收主机发送的第
                        ;9 位(TB8)为 0 的数据
LAB2: CLR RI         ;清接收中断标志
      RETI
LAB1: MOV @R0, SBUF  ;将接收缓冲器中的数据送 R0 指向的存储器
                      ;单元
      INC R0         ;指向下一存储单元
      CLR RI        ;清接收中断标志
      RETI
      END
```

从机 2 和从机 3 的源程序与从机 1 的源程序基本相同,仅第 5 条指令不同:对于从机 1 为 NAME EQU #1,对于从机 2 为 NAME EQU #2,对于从机 3 为 NAME EQU #3。

重点提示 从机程序运行过程如下:系统加电后,主机和 3 台从机各自运行自己的初始化程序;主机发送 1 号从机地址,3 台从机都收到了这个地址号,都向 CPU 申请中断。

从机 1 处理中断过程:判断接收到的第 9 位数据(RB8)是否为 1,为 1,说明接收到的是地址,然后,取出接收缓冲器中接收到的地址号,判断这个地址号是否与本机存储器中的地址号是否相同,相同,则对 SM2 清 0,以便接收主机发送的第 9 位(TB8)为 0 的数据。处理完后,清除接收中断申请标志,中断返回。

从机 2 和从机 3 处理中断过程:判断接收到的第 9 位数据(RB8)是否为 1,若为 1,说明接收到的是地址;然后,取出接收缓冲器中接收到的地址号,判断这个地址号是否与本机存储器中的地址号相同,不同,则转到 LAB2,清除接收中断申请标志,中断返回。

主机发送 2 号从机地址和 3 号从机地址时的过程与以上基本相同,不再分析。

实验 8 用 AT89C51 实验开发板向 PC 机的串口单向发送数据 AF,通信波特率为 4800Kb/s,只要按一次 K1(P3.6 引脚变成低电平),就发送一个 16 进制的 AF 字符。

为了能够在计算机端看到单片机发出的数据,必须借助串口调试软件(可上网下载),串口调试软件无需安装,可以直接在当前位置运行这个软件。软件界面如图 6-31 所示。

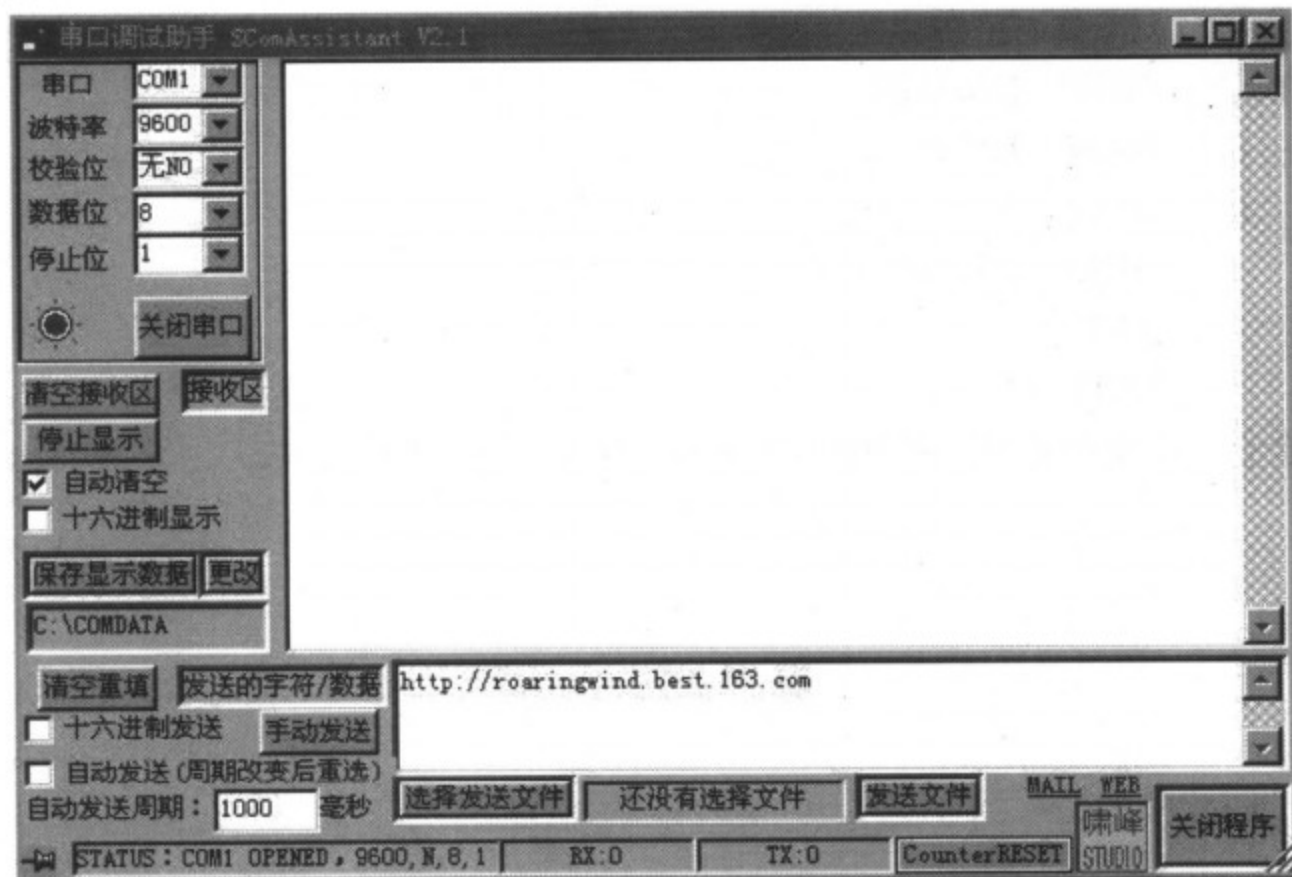


图 6-31 串口调试软件界面

实验前,先要设置一下串口通信的参数,将波特率调整为 4800,勾选十六进制显示。串口选择为 COM1,当然,要将 AT89C51 实验开发板的串口也要和计算机的 COM1 连接,将烧写有以下程序的单片机插入实验开发板的插座中,并接通实验开发板的电源,这时只要按 K1 一次,在串口调试助手软件的接收区界面中就会增加一个“AF”字符,表示单片机向计算机发送“AF”字符成功。

先计算一下计数初值 X(SMOD=0,晶振为 11.0592MHz,波特率为 4800b/s);

$$X = 256 - \frac{2^{\text{smod}} \times f_{\text{osc}}}{384 \times \text{波特率}} = 256 - \frac{1 \times 11.0592 \times 10^6}{384 \times 4800} = 250\text{D} = 0\text{FAH}$$

串口实验的源程序如下:

```

ORG 0000H
MOV SCON, #50H      ;设置成串口方式 1,允许接收
MOV TMOD, #20H      ;设置定时器 T1 为工作方式 2
MOV TH1, #0FAH      ;预置初值
MOV TL1, #0FAH      ;预置初值
SETB TR1            ;启动定时器 T1
WRIT: JB P3.6, $      ;判断 K1 是否按下,如果没有按下就等待
      ACALL DELAY     ;延时 10ms 触点抖动
      JB P3.6, WRIT    ;去除干扰信号
      JNB P3.6, $      ;等待按键松开
      MOV A, #0AFH     ;将 16 进制的字符 AF 发送到串口去
      MOV SBUF, A      ;将 AF 通过串口发送出去
      AJMP WRIT
;以下是 10ms 延时子程序
DELAY: MOV R5, #50
      D2: MOV R4, #100
      D1: DJNZ R4, D1
          DJNZ R5, D2
      RET
      END

```

该实验程序在本书所附光盘的 example\ch_6\com1 文件夹中。



第七章 单片机存储器和 I/O 接口的扩展

80C51 单片机的芯片内集成了计算机的基本功能部件。从一定意义上讲,一块单片机芯片就相当于一台单板机或一个基本的微型计算机系统。这对于在智能仪器、仪表、小型检测及控制系统中就可直接应用单片机而不必再扩展外围芯片,使用极为方便。但对于一些较大的应用系统,单片机片内所具有的功能显得不足,这时就必须在外围扩展一些芯片,适应特定应用的需要。本章主要介绍 80C51 单片机的存储器和 I/O 接口扩展技术。

第一节 系统扩展概述

用单片机组成应用系统时,首先要考虑单片机具有的各种功能能否满足应用系统的要求。如能满足,则称这样的系统为最小应用系统。对于片内程序存储器 8031、8032 单片机,其最小系统还应包括外接的 EPROM 程序存储器芯片,如图 7-1 所示。

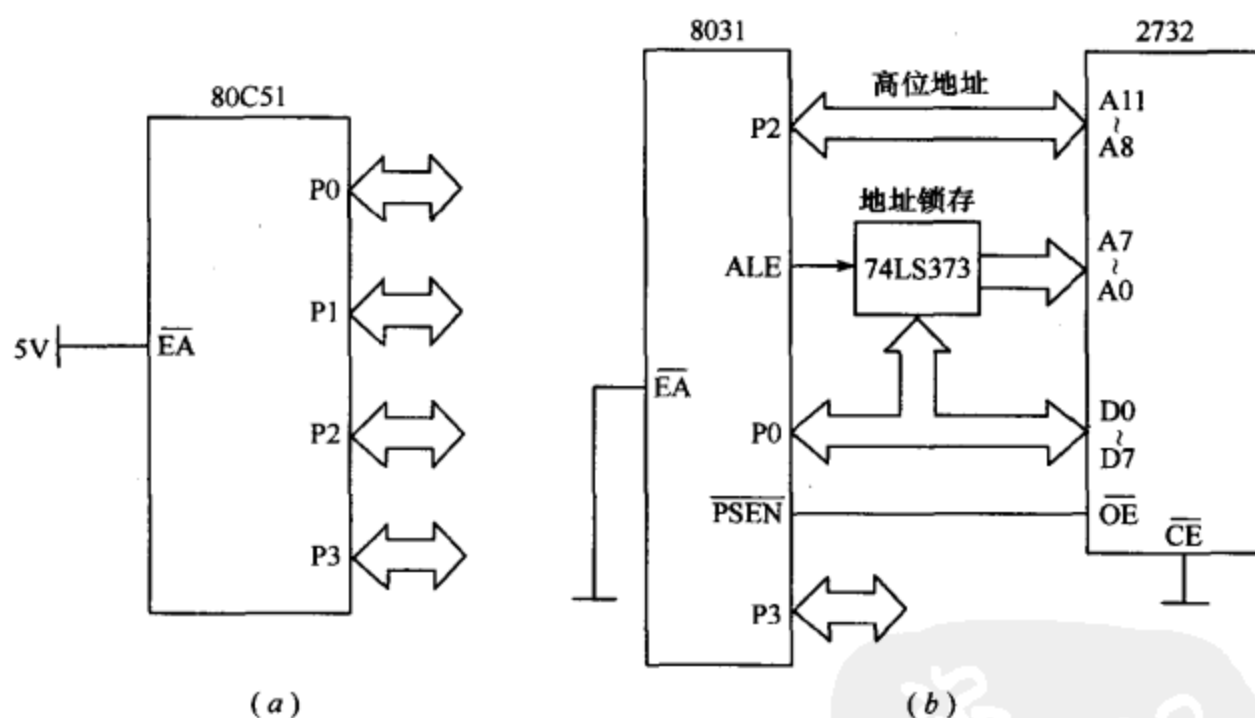


图 7-1 80C51 单片机最小化系统

当单片机最小系统不能满足系统功能的要求时,就需要扩展程序存储器、数据存储器、I/O 接口及其他所需的外围芯片。为了使单片机能方便地与各种扩展芯片连接,应将单片机的外部连线变成为一般的微型机三总线结构形式,即地址总线(AB)、数据总线(DB)和控制总线(CB),它们统称为系统总线,如图 7-2 所示。

一、地址总线(AB)

地址总线上传送的是地址信号,用于存储单元和 I/O 端口的选择。地址总线是单向

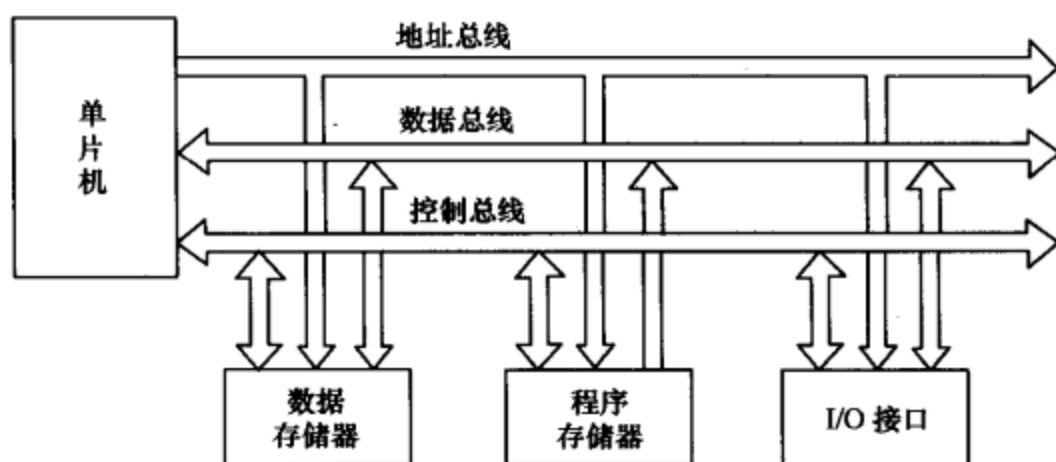


图 7-2 系统总线结构

的,地址信号只能由单片机向外送出。地址总线的数目决定着可直接访问的存储单元的数目,例如, n 位地址,可以产生 2^n 个连续地址编码,因此可访问 2^n 个存储单元,即寻址范围为 2^n 地址单元,如 80C51 单片机地址总线最多有 16 根(用 A15 ~ A0 表示),存储器最多可扩展为 2^{16} (即 64KB) 地址单元。

80C51 单片机由 P2 口提供高 8 位地址线,此口具有输出锁存的功能,能保留地址信息;由 P0 口提供低 8 位地址线和 8 位数据线。由于 P0 口是地址、数据分时使用的通道口,所以为保存地址信息,需外加地址锁存器,锁存低 8 位的地址信息,一般都用 ALE 正脉冲信号的下降沿控制锁存时刻。

二、数据总线(DB)

数据总线用于在单片机与存储器之间或单片机与 I/O 端口之间传送数据。单片机系统数据总线的位数与单片机处理数据的字长一致,例如 80C51 单片机是 8 位字长,所以数据总线的位数是 8 位(用 D7 ~ D0 表示)。数据总线是双向的,例如,80C51 单片机由 P0 口提供,此口是双向、输入三态控制的通道口,可以进行两个方向的数据传送。

三、控制总线(CB)

控制总线是一组控制信号线,包括单片机发出的,以及从其他部件传送给单片机的,对于一条具体的控制信号线来说,其传送方向是单向的,但是由不同方向的控制信号线组合的控制总线则表示为双向。80C51 单片机的控制信号线如图 7-3 所示。

图中:

ALE 控制信号线——作地址锁存的选通信号,把 P0 口输出的低 8 位地址送入锁存器锁存起来,以实现低位地址和数据的分时传送。

$\overline{\text{SPEN}}$ 控制信号线——作外部程序存储器的读选通信号。在读外部 ROM 时 $\overline{\text{PSEN}}$ 为低电平(有效),以实现外部 ROM 的读操作。

$\overline{\text{EA}}$ 控制信号线——作为内外程序存储器的选择信号。

$\overline{\text{RD}}$ (读)和 $\overline{\text{WR}}$ (写)控制信号线——作为扩展数据存储器 and I/O 端口的读写选通信号,低电平有效。

在具有片外扩展存储器的系统中,由 P2 口送出存储器高 8 位地址;P0 口分时工作,先送出存储器低 8 位地址,然后用来传送存储单元的数据。P3 是双功能口,除作为 I/O

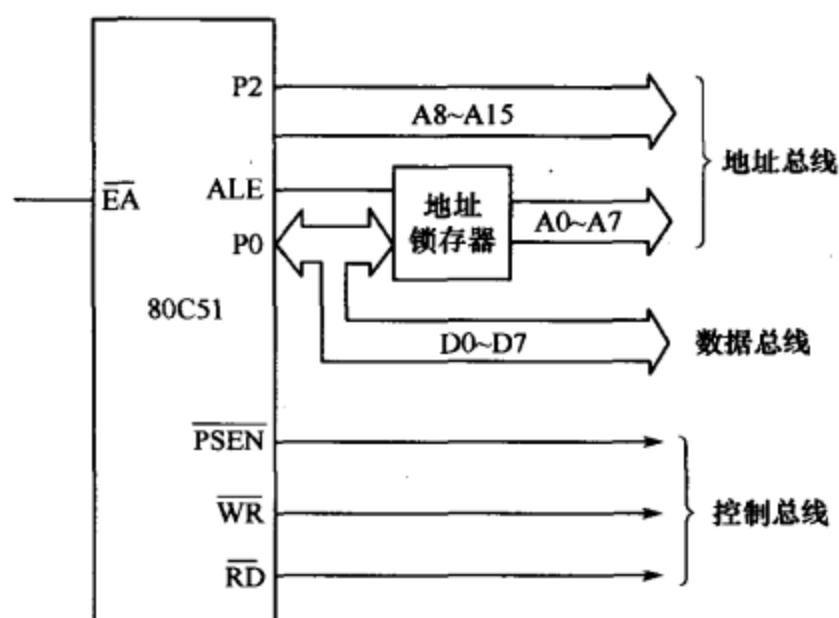


图 7-3 80C51 单片机的控制信号线

接口使用外,每位还独立定义了第二功能,如:定时/计数器中使用的 T0、T1 的外部脉冲的输入端就分别占用了 P3 口的 P3.4 和 P3.5 引脚;外部中断 0、外部中断 1 分别占用了 P3.2 和 P3.3 引脚等。由此可知,虽然 80C51 系列单片机内部有 4 个 8 位 I/O 端口,但可供外部的 I/O 设备直接使用的只有 P1 口。

第二节 存储器的扩展

一、程序存储器的扩展

程序存储器顾名思义是用于存储程序代码的,也存放程序常数。对于无 ROM 型的单片机(如 8031、8032 等),或当单片机内部程序存储器容量不够时,就需要在外部扩展程序存储器。

1. 程序存储器的分类

程序存储器分类在第一章只读存储器部分已作介绍,这里不再重复。

2. 常用程序存储器介绍

1) 常用 EPROM 存储器

常用的 EPROM 电路芯片有 Intel 公司的 2716、2732、2764、27128、27256、27512 等。其容量分别是 2KB、4KB、8KB、16KB、32KB、64KB,即容量值是芯片尾数(后两位或后 3 位数)除以 8。因为 $2^{10} = 1024B = 1KB$,也就是说:1KB 的存储器有 10 根地址线寻址;2KB 的存储器就有 11 根地址线……依次类推,可知不同容量存储器芯片的地址线位数,以便于使用。它们均为 8 位的存储器,所以有 8 条数据线。常用 EPROM 芯片的引脚图如图 7-4 所示。

图中涉及的引脚符号含义如下:

A0~A15——地址输入线;

D0~D7——数据 I/O 线,读存储器操作时为输出线,编程时为数据输入线;

\overline{CE} ——片选输入线,低电平时,芯片被选中;

\overline{OE} ——输出允许信号,低电平时,输出缓冲器打开,被寻址的单元的内容才能被

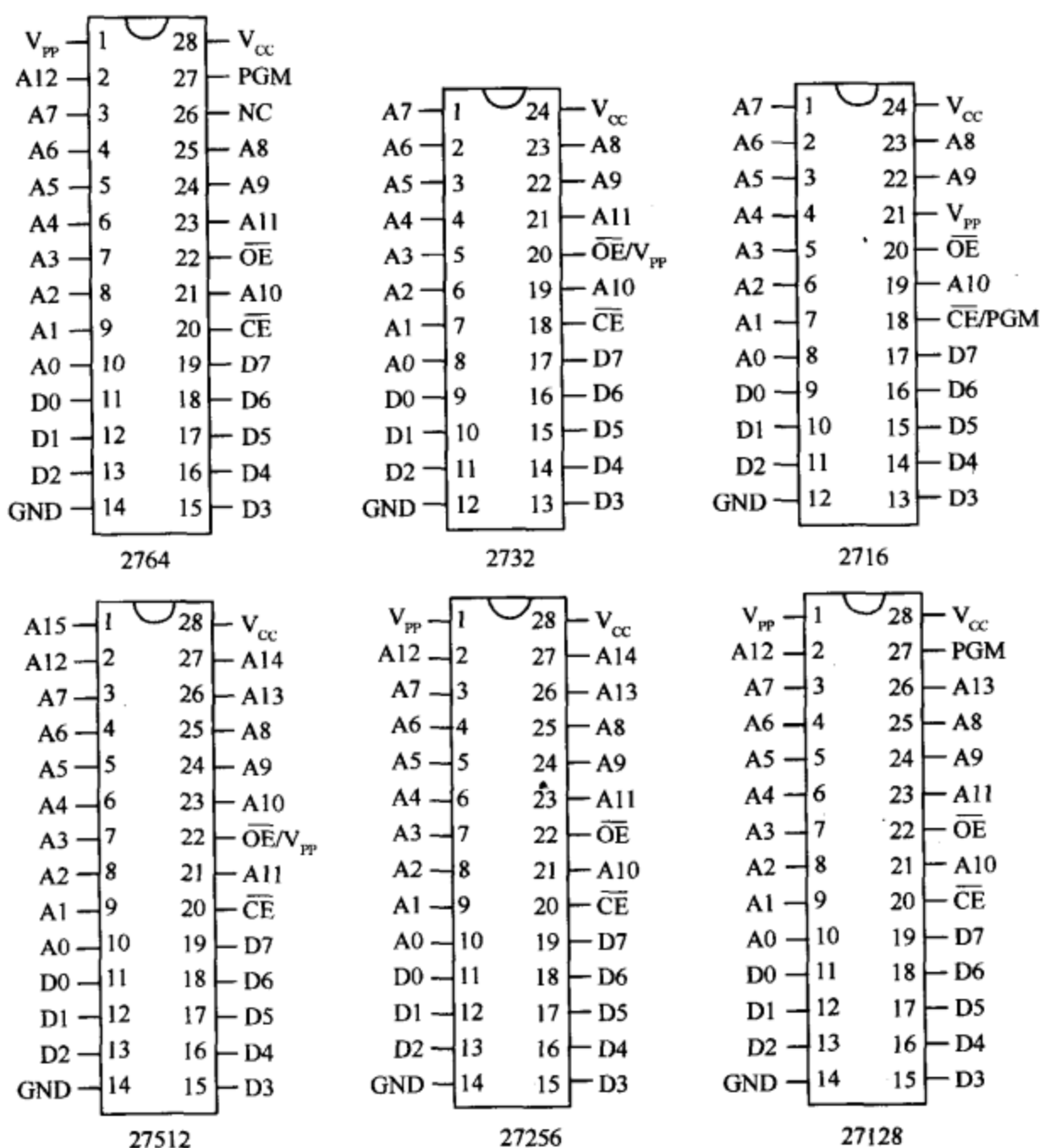


图 7-4 典型 EPROM 芯片管脚功能

读出;

PGM——编程脉冲输入线;

V_{PP}——编程电源,当芯片编程时,该端加上+25V 编程电压,当芯片使用时,该端加上+5V 电压;

V_{CC}——主电源输入线,一般为+5V;

GND——接地端。

从外观上可以看见,在 EPROM 芯片的中央有一个透明的小窗口,紫外线光即是通过这个小窗口将芯片上保存的信息擦除掉的,因为在日光和荧光中都含有紫外线,因此,通常用一块不透明的标签将已保存了信息的 EPROM 芯片的紫外线窗口封住。当然,写入 EPROM 芯片时,首先必须先用紫外线擦除器将 EPROM 中的信息清除掉,使它变为空的芯片后才能进行写操作,应该说明的是这里“空芯片”的“空”并非通常意义上的“空白”,而是此时芯片内部变为全“1”信息,因此,芯片的写入原理实际上是将指定位置上的“1”改为“0”。到这里,有的朋友一定想问:既然日光和荧光均含有紫外线,为什么不让 EPROM 芯片在这些光线下曝露一段时间来擦除呢?要知道,完全擦除一块 EPROM 中

的内容,在日光下至少要一周,在室内荧光下至少要 3 年了! 而且随着芯片容量的增大,时间也得相应拉长。

下面以 2716 为例,简要介绍 EPROM 存储器的工作方式。

2716 的工作方式由 \overline{OE} 、 \overline{CE} 及 V_{PP} 各信号的状态组合决定,各种工作方式的基本情况如表 7-1 所列。

表 7-1 2716 的工作方式

方式 \ 引脚	\overline{CE}	\overline{OE}	V_{PP}	$D_7 \sim D_0$
读出	低	低	+5V	程序读出
未选中	高	任意	+5V	高阻
编程	低	高	+25V	程序写入
程序检验	低	低	+25V	程序读出

读方式:当 \overline{CE} 及 \overline{OE} 均为低电平, $V_{PP}=+5\text{ V}$ 时,2716 芯片被选中,并处于读出工作方式。这时被寻址单元的内容经数据线 $D_7 \sim D_0$ 读出。

未选中方式:当 \overline{CE} 为高电平时,芯片不被选中,其数据线输出为高阻抗状态。这时 2716 处于低功耗维持状态,其功耗从读出方式的 252 mW 下降到 132 mW。

编程方式:当 V_{PP} 为 +25V 高电压, \overline{OE} 端加 TTL 高电平时,2716 处于编程工作方式,进行信息的重新写入。这时编程地址和写入数据分别由 $A_{10} \sim A_0$ 及 $D_7 \sim D_0$ 引入。

程序检验方式:程序检验是检查写入的信息是否正确,因此程序检验通常总是紧跟编程之后。这时 $V_{PP}=+25\text{ V}$, \overline{CE} 及 \overline{OE} 为低电平。

2)常用 EEPROM 存储器

电擦除可编程只读存储器 EEPROM 主要优点是能在应用系统中进行在线擦除和改写,且不再需要专用的编程电源,可以直接使用单片机系统的 5V 电源,并能在断电情况下保存数据而不需要保持电源,这就使得它既有 RAM 的读/写特性,又具有非易失性存储器 ROM 在掉电后仍能保持所存储数据的优点。因此,EEPROM 芯片在单片机存储器扩展中,可以用做程序存储器,也可以用做数据存储器。

通常,EEPROM 芯片分为串行 EEPROM 和并行 EEPROM 两种,串行 EEPROM 在读写时数据的输入/输出是通过 2 线、3 线、4 线或 SPI 总线等接口方式进行的,而并行 EEPROM 的数据输入/输出则是通过并行总线进行的。系统扩展用的主要是并行存储器。

近年来 Intel 公司生产的可用做单片机并程序存储器的 EEPROM 器件典型产品有 2816A、2817A、2864A 等。如图 7-5 为两种典型 EEPROM 存储器芯片引脚分配情况。

图中涉及的引脚符号的含义为:

$A_0 \sim A_{12}$ ——地址输入线;

$D_0 \sim D_7$ ——数据 I/O 线,读存储器操作时为输出线,写存储器操作时为数据输入线;

\overline{CE} ——片选输入线,低电平时,芯片被选中;

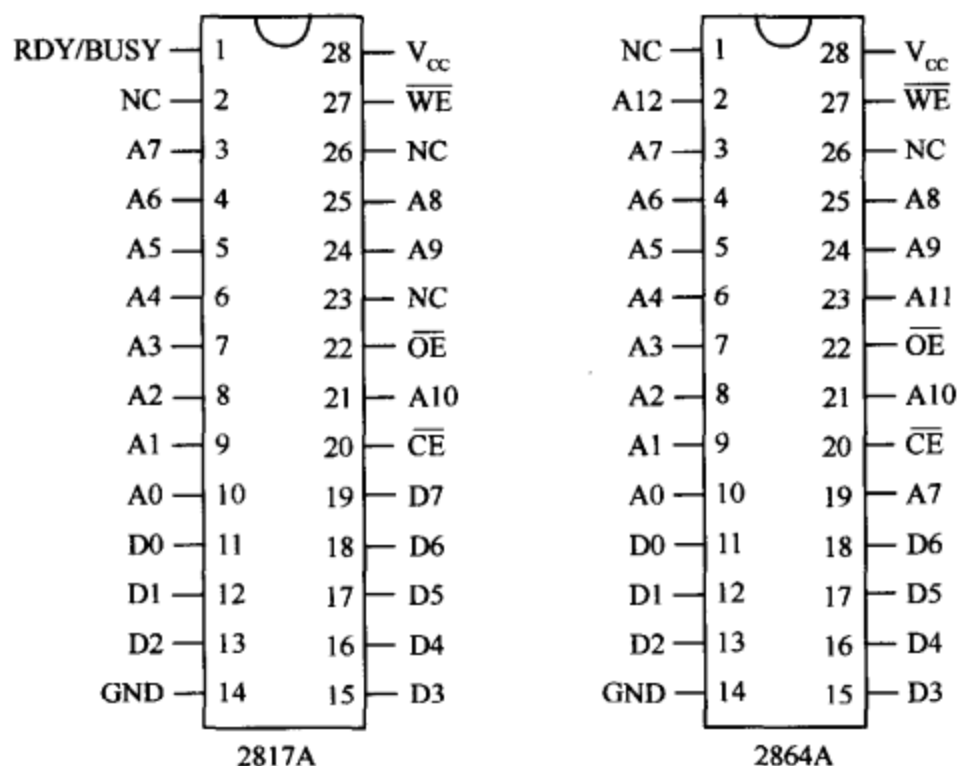


图 7-5 并行 EEPROM 的引脚功能

\overline{OE} ——输出允许信号,低电平时,输出缓冲器打开,被寻址的单元的内容才能被读出;

\overline{WE} ——写选通信号输入信号,低电平有效;

RDY/\overline{BUSY} ——2817A 的状态输出线,低电平时表示正在进行写操作,写入完毕呈高阻态;

V_{CC} ——主电源输入线,一般为+5V;

GND——接地端。

下面以 2717A 为例,简要介绍 EEPROM 存储器的工作方式。

当向 2817A 发出字节写入命令后,2817A 便锁存地址、数据及控制信号,从而启动一次写操作。在此期间,2817A 的 RDY/\overline{BUSY} 脚呈低电平,一旦一次字节写入操作完毕,2817A 便将 RDY/\overline{BUSY} 线置高,由此来通知单片机。2817A 的工作方式如表 7-2 所列。

表 7-2 2817A 的工作方式

方式 \ 引脚	\overline{CE}	\overline{OE}	\overline{WE}	RDY/\overline{BUSY}	D7~D0
读	低	低	高	高阻	输出
维持	高	任意	任意	高阻	高阻
写	低	高	低	低	输入
字节擦除		字节写入之前自动擦除			

3. 程序存储器扩展举例

例 1 如图 7-6 所示,使用一片 EPROM 27128 和 741S373 构成的 8031 程序存储器扩展电路。试分析其总线连接情况和地址空间的分配。

电路中,74LS373 为 8 位三态输出锁存器,其引脚排列如图 7-7 所示,它内含 8 个 D 触发器,输入为 D0~D7,输出为 Q0~Q7。 \overline{OE} 为输出使能端,G 为脉冲输入端。

当输出允许 \overline{OE} 端加以低电平或负脉冲时,三态门处于导通状态,允许数据信息反映

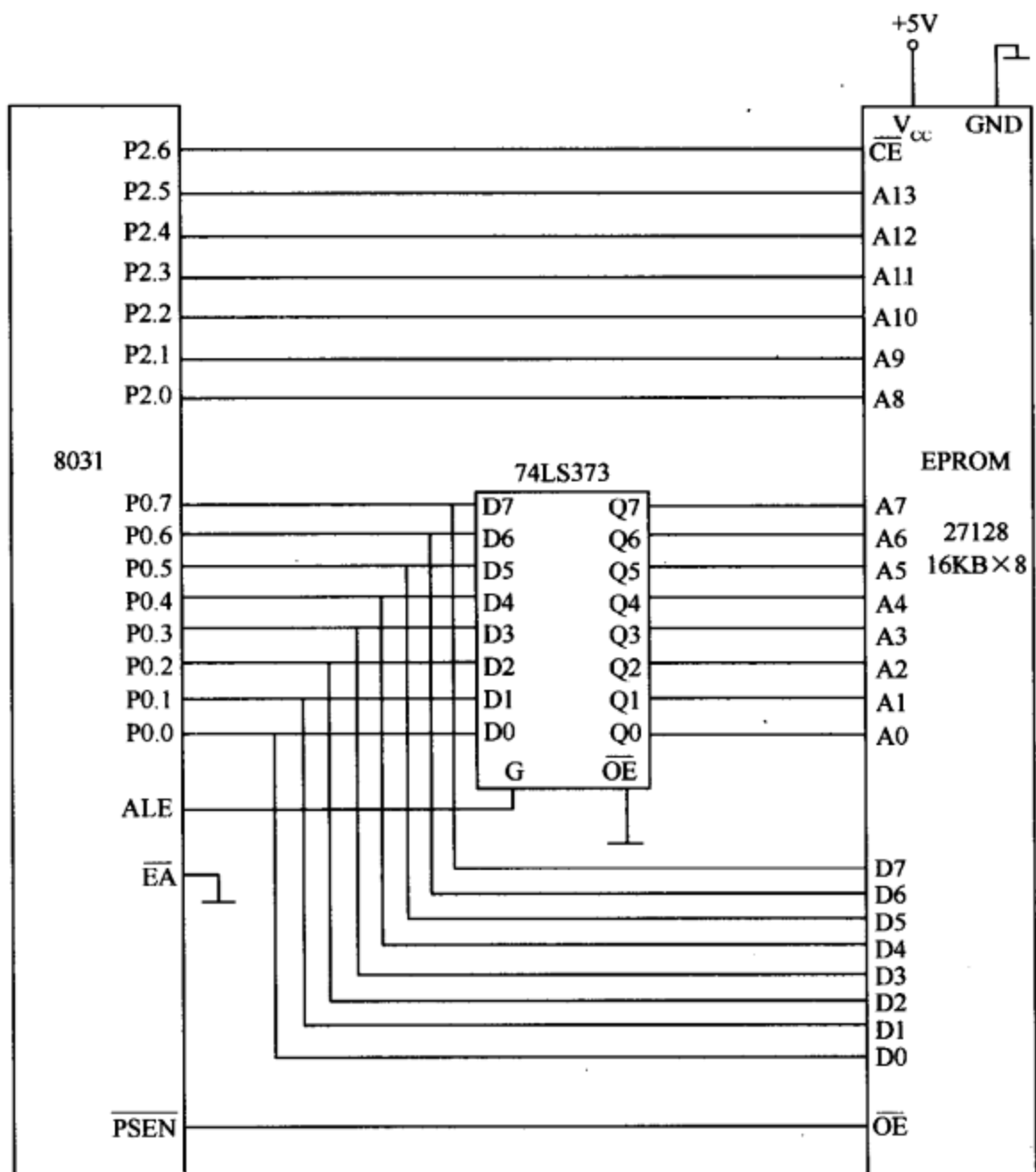


图 7-6 扩展程序存储器 27128

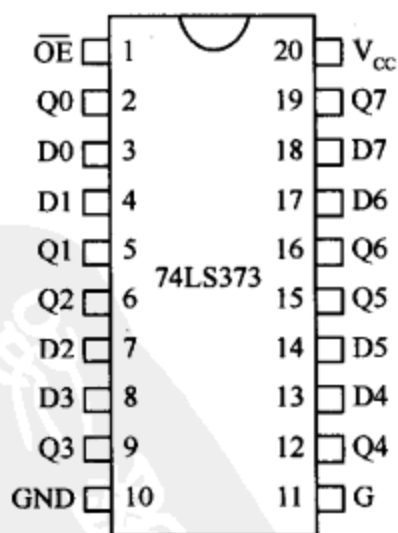


图 7-7 8 位三态输出锁存器 74LS373 引脚排列图

到输出端 $Q_0 \sim Q_7$ 上；当 \overline{OE} 端为高电平时，输出三态门断开。如不需要三态，只要将 \overline{OE} 端接地，即成为两态输出。

G 为锁存脉冲输入端（触发端）。当 G 端为高电平时，加在并行输入端 $D_0 \sim D_3$ 数据就立即送入内部寄存器中。当 G 为低电平时，内部寄存器保持内容不变，输出 $Q_0 \sim Q_3$

的状态与输入端 D0~D3 端数据无关。

8 位 3 态输出锁存器 74LS373 状态表如表 7-3 所列。

表 7-3 8 位 3 态输出锁存器 74LS373 状态表

输 入			输出	说明
\overline{OE}	G	D0~D7	Q0~Q7	
0	1	D0~D7	D0~D7	送数
0	0	×		保持
1	×	×		高阻

(1)地址线的连接。27128 芯片是 16KB×8 位 EPROM 存储器。它有 14 根地址线 A13~A0,这 14 根地址线分别与 8031 的 P0 口和 P2.0~P2.5 连接,当 8031 通过 P0 口、P2 口给 27128 地址总线上发送地址信息时,可分别选中 27128 片内 16KB 存储空间中任何一个单元。

(2)数据线的连接。27128 芯片的数据线有 8 条,分别是 D0~D7,它们直接与 8031 芯片的 P0 口 8 位口线(即 P0.0~P0.7)相连。

(3)控制线的连接。27128 的输出允许端 \overline{OE} 引脚与 8031 的 \overline{PSEN} 相连,即 \overline{OE} 端由 8031 的 \overline{PSEN} 引脚控制实现存储器读操作。

27128 的 \overline{CE} 引脚为片选信号输入端与 8031 的 P2.6 引脚连接,该片选信号决定了 27128 芯片的 16KB 存储器在整个 8031 扩展程序存储器 64KB 空间中的位置。

方法技巧 扩展 2764 芯片时,需要去掉 P2.5 引脚的连接;扩展 2732 芯片时,需要去掉 P2.5、P2.4 引脚的连接;扩展 2716 芯片时,需要去掉 P2.5、P2.4、P2.3 引脚的连接;扩展 27256 芯片时,则需要增加 P2.6、P2.7。它们分别与扩展程序存储器的 A14、A15 相连。片选端 \overline{CE} 直接接地。也就是说,存储器芯片有多少位地址线就从 P0 口和 P2 口连接多少位,连接时遵循的原则是:先从 P0 的 8 位(地址线的低 8 位)连接,接着是连接 P2 的低位,地址线的连接是由低位向高位顺序连接的。

若只扩展一片程序存储器时,程序存储器 EPROM 的片选端 \overline{CE} 可接地,也可接闲置的高位地址线(即 P2 口高位引脚线)。近年来,由于大容量芯片的出现,多片较小容量的 EPROM 完全可用较大容量的 EPROM 代替,使电路大为简化,增加了系统的可靠性。

(4)地址空间的分配。80C51 单片机共有 16 条地址线,最大可以扩展 64KB,地址范围为 0000H~0FFFFH。而每一块存储器芯片的容量不一定都能达到 64KB×8 位,例如,27128 的只有 14 条地址线(A13~A0),其寻址范围为

$$\times\times00\ 0000\ 0000\ 0000\sim\times\times11\ 1111\ 1111\ 1111$$

决定存储器芯片地址范围的因素有两个:一是片选端 \overline{CE} 的连接方法;一是存储器芯片的地址线与单片机地址线的连接。在确定地址范围时,必须保证片选端 \overline{CE} 为低电平。由于存储器 27128 的片选端 \overline{CE} 与 P2.6 连接。因此,程序存储器的寻址范围为

$$\times000\ 0000\ 0000\ 0000\sim\times011\ 1111\ 1111\ 1111$$

其中,×表示 P2.7 地址线悬空,不与存储器相连,即与存储器无关,取 1 或 0 都可以。如果取 0,存储器地址空间为 0000H~3FFFFH;如果取 1,存储器地址空间为 8000H~0BFFFFH。

可见由于有地址线悬空,结果对一个芯片的一个确定的存储单元来说,可能有两个地址。为了防止这种情况出现,要为一个芯片指定一个唯一的空间,以便使用方便。

方法技巧 对于内部无 ROM 的 8031 单片机,它的程序存储器必须外接,最大地址空间为 64KB,此时单片机的 $\overline{\text{EA}}$ 端必须接地。强制 CPU 从外部程序存储器读取程序。对于内部有 ROM 的 80C51 单片机,正常运行时, $\overline{\text{EA}}$ 一般接高电平,使 CPU 先从内部的程序存储器中读取程序,当 PC 值超过内部 ROM 的容量(0000H~0FFFH)时,才会转向外部的程序存储器,从 1000H 处读取程序。

例 2 如图 7-8 所示,使用一片 EEPROM 2817A 和 74LS373 构成的 8031 程序存储器扩展电路。试分析其总线连接情况。

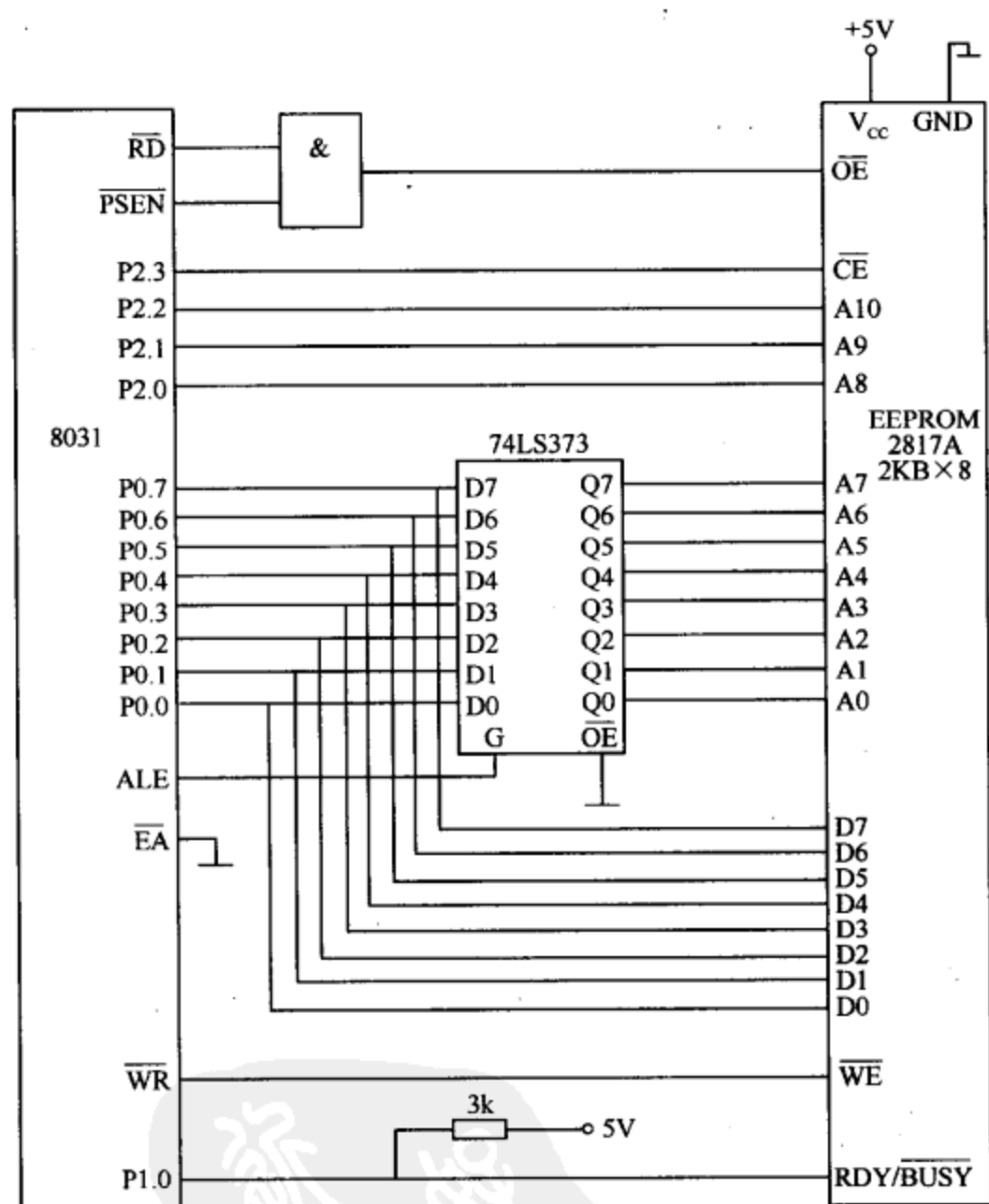


图 7-8 扩展程序存储器 2817A

(1)地址线的连接。2817A 为 2KB 的 EEPROM 芯片,所以有 11 条地址线 A0~A10。这 11 根地址线分别与 8031 的 P0 口(即 P0.0~P0.7)和 P2 口的 P2.0~P2.2 连接。

(2)数据线的连接。2817A 的数据线有 8 条,分别是 D0~D7,它们直接与 P0 口的 8 位口线相连。

(3)控制线的连接。2817A 的输出允许端 $\overline{\text{OE}}$ 脚与 8031 的 $\overline{\text{PSEN}}$ 和 $\overline{\text{RD}}$ 相与后连接,使

得无论 \overline{RD} 有效还是 \overline{PSEN} 有效都能使 2817A 的 \overline{OE} 端有效,这种连接方法是把 EEPROM 既作为程序存储器,也作为数据存储器。

2817A 的片选线 \overline{CE} 脚与 8031 的 P2.3 相连,只有当 P2.3 为低电平时才能选中 2817A 芯片。

2817A 的写允许控制端 \overline{WE} 与 8031 的写控制信号 \overline{WR} 相连。

2817A 的 RDY/ \overline{BUSY} 与 8031 的 P1.0 相连,这样,可采用查询方式对 2817A 的写操作进行管理。在擦、写操作期间,RDY/ \overline{BUSY} 脚为低电平,当字节擦写完毕时,RDY/ \overline{BUSY} 为高电平,故可通过 P1.0 来查询 RDY/ \overline{BUSY} 状态,以便判断字节是否擦写完。

当系统中还有其他程序存储器或数据存储器时,要统一考虑地址空间的分配。该例中,如果无关的位取 0,则 2817A 芯片的地址范围是 0000H~07FFH。

注意 2817A 是 5V 电擦除可编程只读存储器,因此扩展电路中不需要专门配置写入电源,在正常运行时,能在线随时写入或读出。但是,80C51 单片机指令系统中没有向程序存储器写数据的指令,因此,对 EEPROM 写入操作应使用 MOVX 类指令,这时是将 EEPROM 视做外部扩展的数据存储器来进行操作,所以,2817A 的写允许控制端 \overline{WE} 与 8031 的写控制信号 \overline{WR} 相连。又因 8031 的 \overline{PSEN} 和 \overline{RD} 相与后与 2817A 的输出允许端 \overline{OE} 脚相连,如果 \overline{PSEN} 和 \overline{RD} 两个信号中的一个有效(低电平),则与门的输出就为低电平,在 \overline{OE} 端就可得到一个有效的读选通信号,从而使两个选通信号中任何一个都可以控制该存储芯片的读操作。这样,该芯片就既可以作为数据存储器使用,又可以作为程序存储器使用。

二、数据存储器的扩展

80C51 单片机芯片内部有 256B 的 RAM,在大多数控制场合,内部 RAM 能满足对数据存储器的要求,但在一些需要大容量数据缓冲器场合(如数据采集系统等),仅靠片内的数据存储器往往不够,在这种情况下,需要对数据存储器进行扩展。

扩展片外数据存储器的地址线也是由 P0 口和 P2 口提供的,因此最大寻址范围为 64KB(0000H~0FFFFH)。可见,外部数存储器与程序存储器的地址是重叠编址的(地址空间相同),所以两者的地址总线 and 数据总线可完全并联使用,但数据存储器的读和写分别由 \overline{WR} 和 \overline{RD} 控制,而程序存储器的读操作由 \overline{RSEN} 进行控制,故不会发总线冲突。

1. 常用数据存储器介绍

目前,用于单片机扩展用的数据存储器主要静态随机存储器(SRAM),常见型号有 6116(2KB×8)、6264(8KB×8)、62256(32KB×8)等,其引脚功能如图 7-9 所示。

图中,各引脚符号的功能如下:

A0~Ai——地址输入线;

D0~D7——双向三态数据线;

\overline{CE} ——片选信号输入线,低电平有效;

\overline{OE} ——读选通信号输入线,低电平有效;

\overline{WE} ——写允许信号输入线,低电平有效;

V_{CC}——电源电压,为+5V;

GND——接地端。

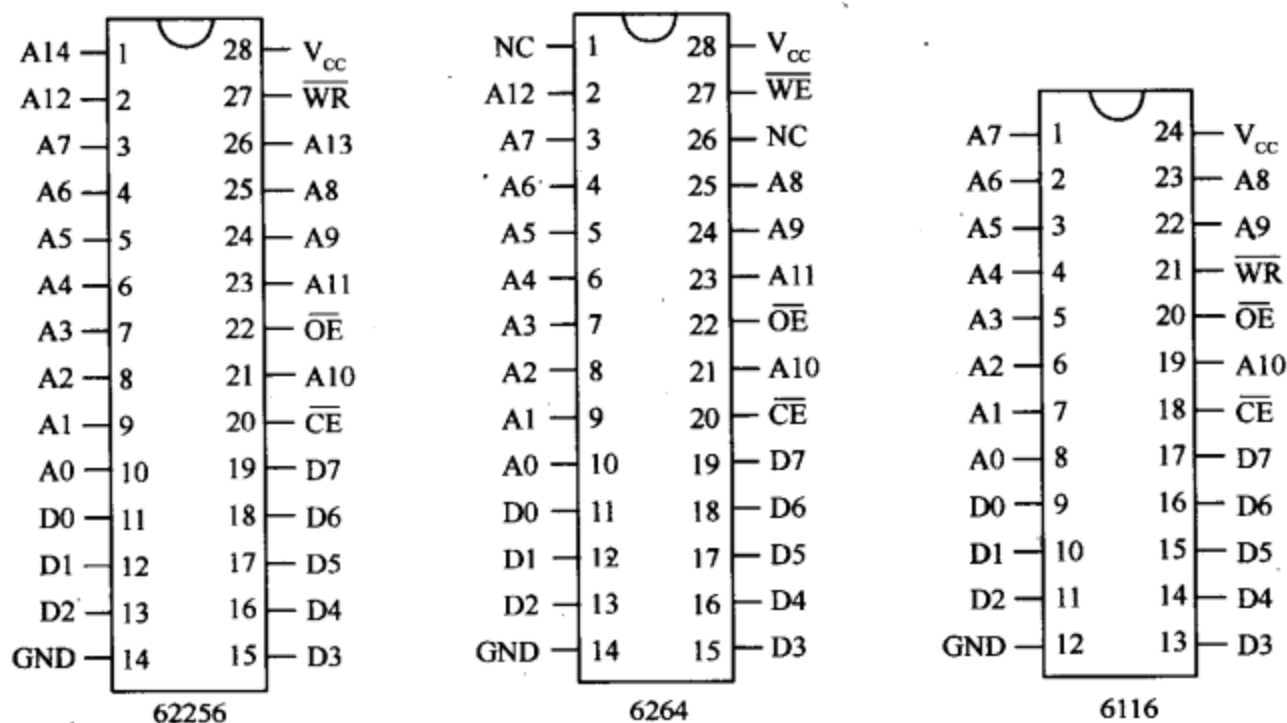


图 7-9 常用静态随机存储器引脚功能

2. 数据存储器扩展举例

例 3 扩展 6116 数据存储器的电路原理图如图 7-10 所示,试分析其总线连接情况。

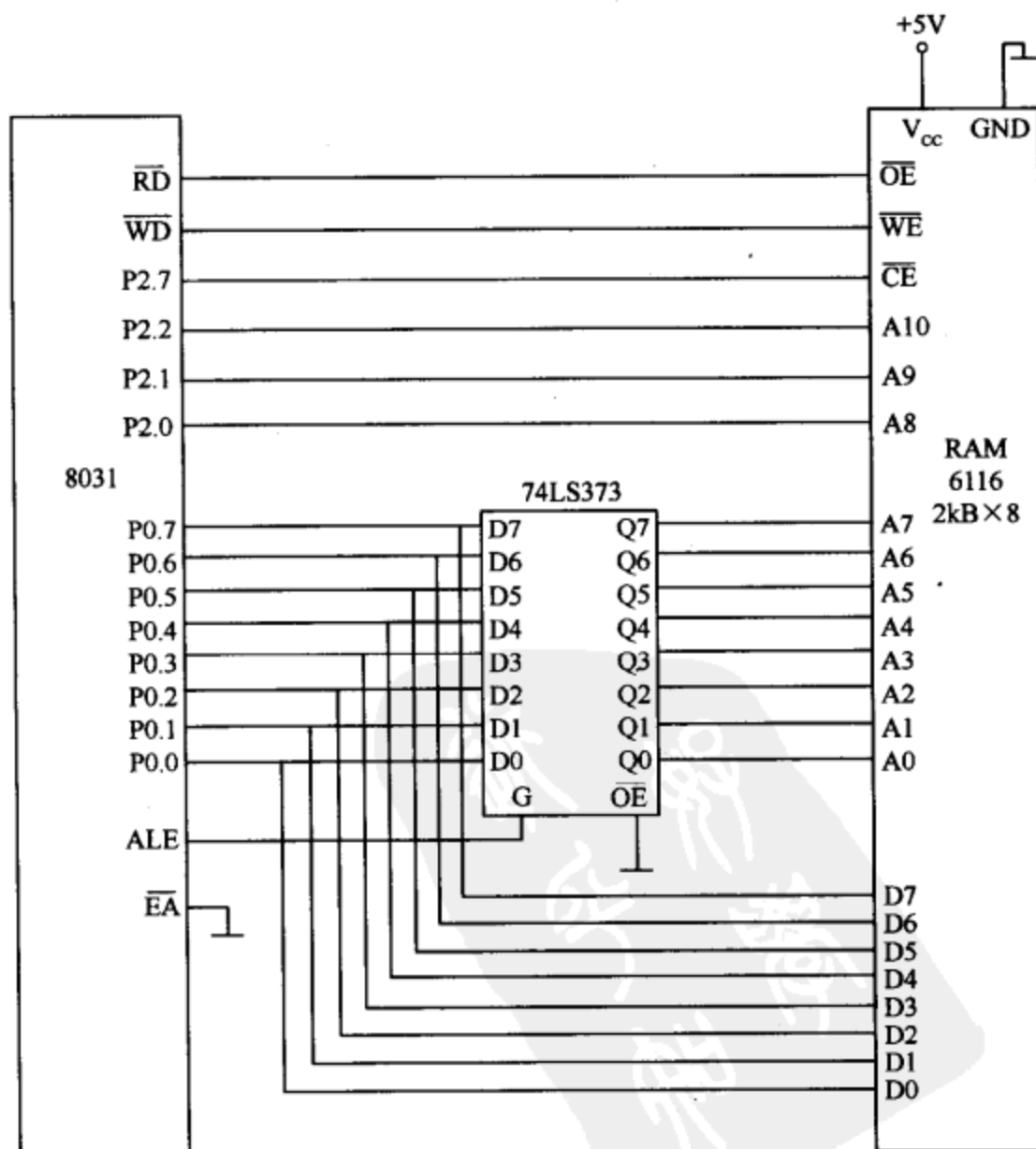


图 7-10 扩展数据存储器 6116

(1)地址线的连接。6116 芯片是 2KB 的 RAM,它有 11 位地址,分别是 A0~A10,分别与 8031 芯片的 P0 口和 P2 口的 P2.0、P2.1、P2.2 相连。

(2)数据线的连接。6116 芯片的 8 位数据线 D0~D7 直接与 P 口的 8 位口线相连。

(3)控制线的连接。片选端 \overline{CE} 连接 8031 的 P2.7;读允许线 \overline{OE} 连接 8031 的读控制信号 \overline{RD} 脚;写允许线 \overline{WE} 连接 8031 的写控制信号 \overline{WR} 。

其地址空间的确定方法与 ROM 相同。

第三节 I/O 接口的扩展

一、I/O 接口扩展概述

单片机虽然有 4 个 8 门双向 I/O 接口,但在实际应用中,这些口并不能全部用于输入/输出。通常,P0 口作为低 8 位地址线 and 数据线,P2 口作为高 8 位地址线,而 P3 口线一般用于第二功能。这样,真正能作为 I/O 使用的就只有 P1 口了。因此,设计应用系统时,常需要对 I/O 接口进行扩展。

1. I/O 接口的特点

在单片机系统中主要有两类数据传送操作:一类是单片机和存储器之间的数据读写操作;另一类则是单片机和其他设备之间的数据输入/输出操作。

由于存储器与单片机具有相同的电路形式和信号形式,能相互兼容,直接使用,因此存储器与单片机之间采用同步定时工作方式,它们之间只要在时序关系上能相互满足就可以正常工作。因此,存储器与单片机之间的连接十分简单,除地址线、数据线之外,就是读写、选通等信号,实现起来非常方便。

但是,单片机与外部设备之间的数据传送却十分复杂,主要表现在以下两个方面:一是速度差异大。如:开关、继电器、机械传感器等属于慢速设备,每秒钟传送不了一个数据;而高速采样设备,每秒钟要传送成千上万个数据位,使得单片机不可能以一个固定的时序外部设备进行协调工作。二是设备种类繁多。单片机的外部设备既可能是机械式的,又可能是电子式的,由于不同设备之间性能各异、对数据的要求互不相同,因此无法按统一格式进行数据传送。

由此可见,单片机的 I/O 操作比较复杂,单靠单片机本身的口电路是无法实现的,为此,必须扩展接口电路,对单片机与设备之间的数据传送进行协调和控制。

2. 扩展 I/O 接口需要考虑的问题

在单片机应用系统中,扩展 I/O 接口电路主要考虑以下几个问题:

(1)速度协调。由于速度上的差异,使得单片机的 I/O 数据传送只能以异步方式进行,即只能在确认设备已为数据传送做好准备的前提下才能进行 I/O 操作。而要知道设备是否准备好,就需要通过接口电路产生或传送设备的状态信息,以此实现单片机与设备之间的速度协调。

(2)输出数据锁存。在单片机应用系统中,数据输出都是通过系统的公用数据通道(数据总线)进行的,但是由于单片机的工作速度快,数据在数据总线上保留的时间十分短暂,无法满足慢速输出设备的需要。为此,在扩展 I/O 接口电路中应具有数据锁存器,以

保存输出数据,直至能为输出设备所接收。

(3)输入数据三态缓冲。数据输入时,输入设备向单片机传递的数据也要通过数据总线,但数据总线是系统的公用数据通道,上面可能“挂”着多个数据源,工作比较繁忙。为了维护数据总线上数据传送的秩序,因此只允许当前时刻正在进行数据传送的数据源使用数据总线,其余数据源都必须与数据总线处于隔离状态。为此要求接口电路能为数据输入提供三态缓冲功能。

3. 扩展 I/O 接口的方法

扩展 I/O 接口的方法较多,主要有两种:一种是采用可编程通用接口芯片,如 8155、8255、8253、8279 等,这些接口芯片均可与 80C51 单片机总线直接连接;第二种方法是采用 TIL 电路芯片,常用的有各种锁存器如 74LS273/373/374/377 和三态缓冲器如 74LS244/245 等。

二、I/O 接口扩展举例

下面以主要介绍采用 TIL 电路芯片扩展 I/O 接口的方法。

例 4 如图 7-11 所示,采用 74 系列 TTL 电路芯片,将并行数据输入或输出。74LS244 用做扩展输入;74LS273 用做扩展输出。试分析其总线连接情况和芯片的地址。

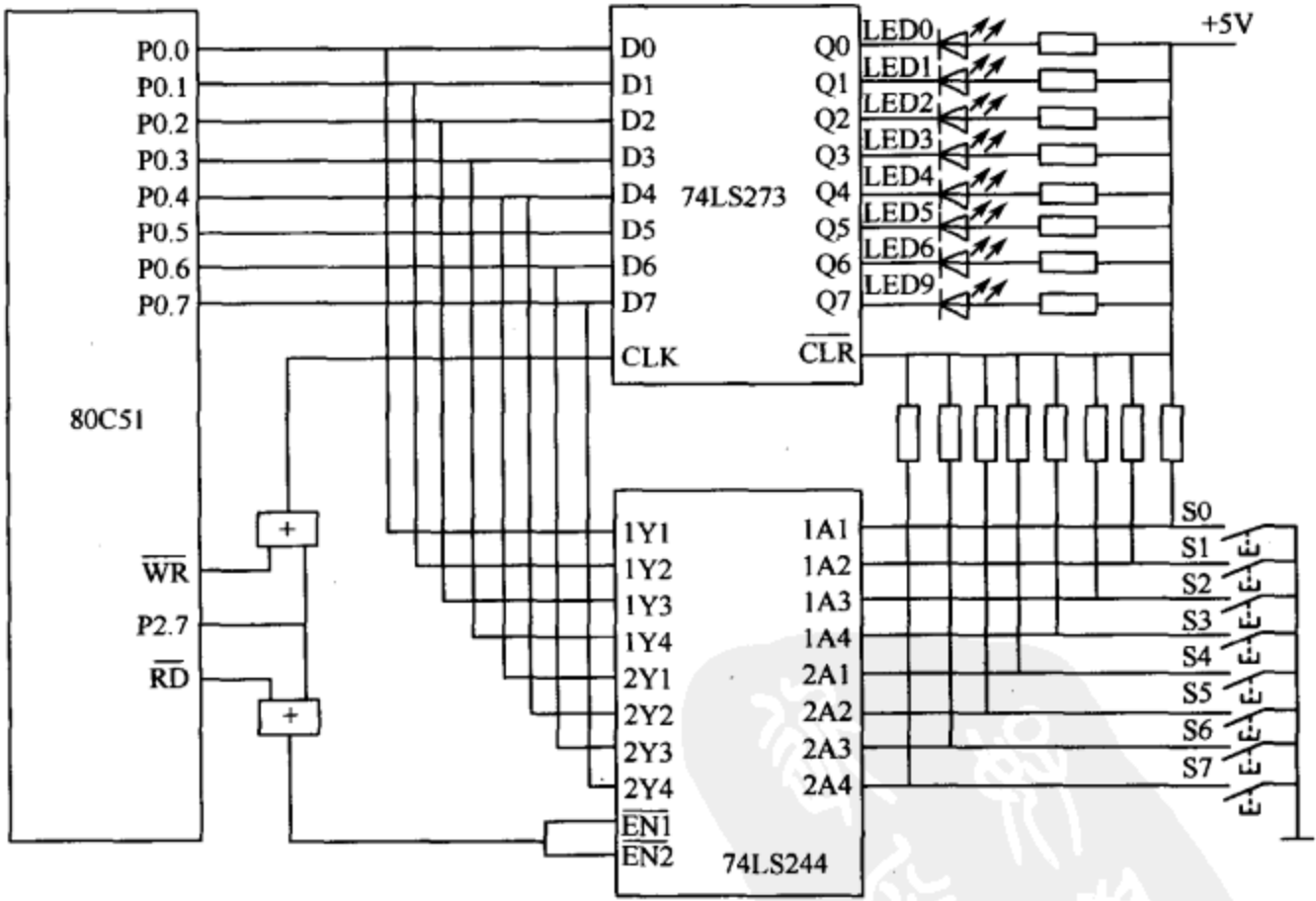


图 7-11 扩展 I/O 接口

电路中,74LS244 是双 4 位三态缓冲器,管脚封装如图 7-12 所示。

74LS244 内部共有 8 只三态缓冲器、分成两组,每组 4 个,共用一个“输出允许”控制信号 \overline{EN} 。当 $\overline{EN}=0$ 时,对应的 4 个缓冲器的 $Y=A$;当 $\overline{EN}=1$ 时,对应的 4 个缓冲器输出为高阻($Y=Z$)。

电路中,74LS273 是两态输出 8D 锁存器,其管脚排列如图 7-13 所示。

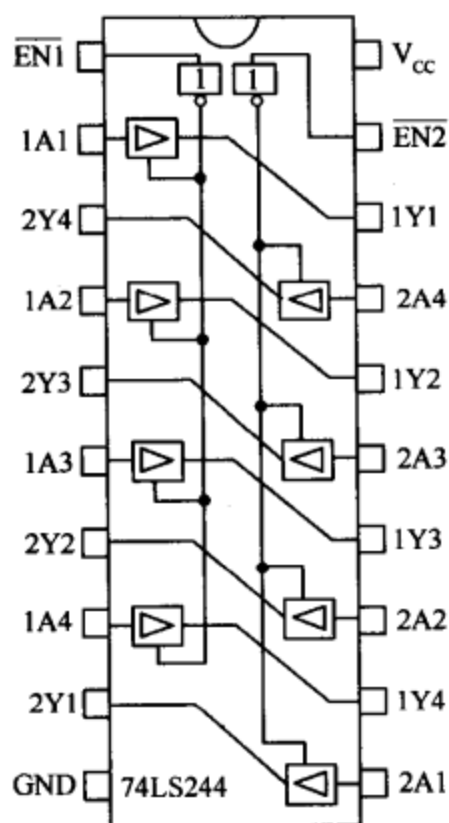


图 7-12 74LS244 管脚排列

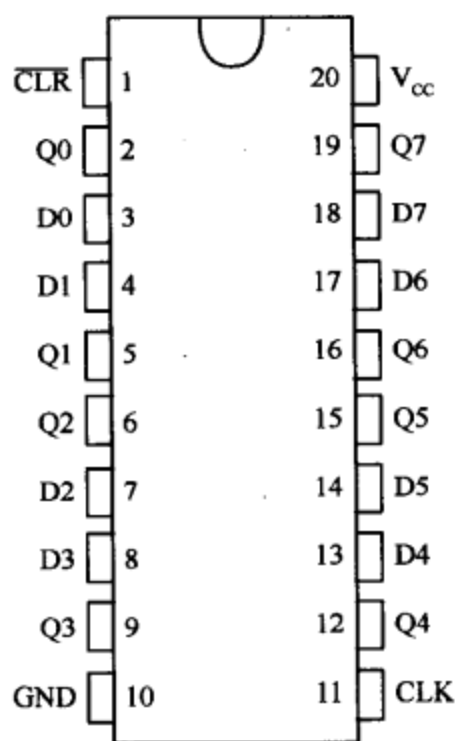


图 7-13 74LS272 管脚排列

74LS273 的引脚“1”为清零端($\overline{\text{CLR}}$),低电平有效,因此正常工作时须将该端接高电平。74LS273 状态表如表 7-4 所列。

表 7-4 锁存器 74LS273 状态表

输 入			输出	说明
$\overline{\text{CLR}}$	CLK	D0~D7	Q0~Q7	
0	×	×	0	清零
1	↑	D0~D7	D0~D7	送数
1	0	×	保持	

从状态表中可以看出,74LS273 具有以下功能:

$\overline{\text{CLR}}=0$ 时,无论寄存器中原来的内容是什么,只要 $\overline{\text{CLR}}=0$,就立即清零。

当 $\overline{\text{CLR}}=1$ 时,在 CLK 的上升沿,加在并行输入端 D0~D3 数据就立即送入寄存器中。

当 $\overline{\text{CLR}}=1$,在 CLK 端的脉冲上跳沿过后,寄存器保持内容不变,输出 Q0~Q3 的状态与输入端 D0~D3 端数据无关。

(1)连线说明。P0 口为双向数据线,既能从 74LS244 输入数据,能将数据传送给 74LS273 输出。输出控制信号由 P2.7 和 $\overline{\text{WR}}$ 合成。当 P2.7 和 $\overline{\text{WR}}$ 同时为低电平 0 时,或门输出为 0,将 P0 口数据锁存到 74LS273,其输出控制着发光二极管 LED,当某线输出为低电平 0 时,该线上的 LED 发光。

输入控制信号由 P2.7 和 $\overline{\text{RD}}$ 合成。当 P2.7 和 $\overline{\text{RD}}$ 同时为低电平 0 时,或门输出为 0,选通 74LS244,将外部信号输入到总线。根据电路原理:无键按下时,输入为全 1;若按下某键,则所在线输入为 0。

(2)芯片地址的确定。80C51 单片机把外扩 I/O 接口和片外 RAM 统一编址,每个扩展的接口相当于一个扩展的外部 RAM 单元,访问外部接口就同访问外部 FRAM 一样,

用的都是 MOVX 类指令。用 \overline{RD} 和 \overline{WR} 作为输入/输出控制信号。

因为输入和输出都是在 P2.7 为低电平 0 时有效,所以 74LS244 和 74LS273 的地址相同,为 7FFFH(这个地址不是唯一的,也就是说,该扩展图中的两个芯片还可以为其他地址)。

74LS244 和 74LS273 分别是由 \overline{RD} (输入)和 \overline{WR} (输出)信号控制,这两个信号不会同时有效。输入时, \overline{RD} 有效,此时执行指令为 MOVX A,@DPTR 或 MOVX A,@Ri;输出时, \overline{WR} 有效,此时执行指令为 MOVX @DPTR,A 或 MOVX @Ri,A。因此,它们的地址虽然相同,但不会在操作时发生冲突。

(3)程序控制。图 7-11 中所示电路中可实现的功能是:按下任意键,其对应的 LED 发光。程序如下:

LOOP:MOV DPTR,#7FFFH	;数据指针指向扩展的 I/O 接口地址
MOVX A,@DPTR	;向 74LS244 读入数据,检测按键
MOVX @DPTR,A	;向 74LS273 输出数据,驱动 LED
SJMP LOOP	;循环

从该程序可以看到,对于接口的输入/输出就像从外部 RAM 读/写数据一样方便。



第八章 单片机实用接口技术

在单片机开发工作中,根据用户的不同要求,单片机应用系统常常需要配接 LED 显示器、键盘、LCD 显示器、A/D(模/数)转换器和 D/A(数/模)转换器等外围设备,接口技术就是解决单片机与外设之间相互联系的问题。本章将通过一些实例和实验,对单片机开发中的实用接口技术进行精要分析和介绍。

第一节 LED 显示器接口

一、8 段 LED 显示器的结构及原理

1. LED 显示器的结构

LED 是发光二极管的简称,其 PN 结是用某些特殊的半导体材料(如磷砷化镓)做成的,当外加正向电压时,可以将电能转换成光能,从而发出清晰悦目的光线。如果将多个 LED 管排列好并封装在一起,就成为 LED 显示器。LED 发光管和常用的 8 段 LED 显示器(也称 LED 数码管)的结构如图 8-1 所示。

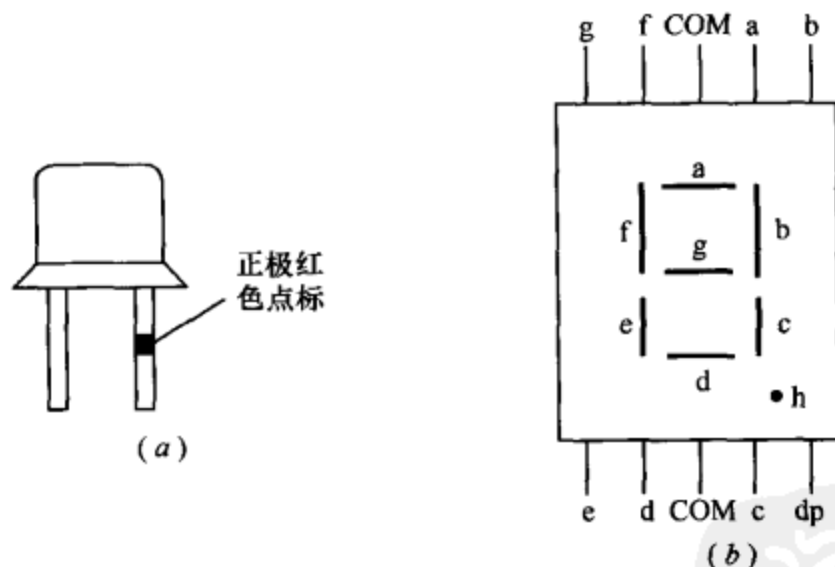


图 8-1 LED 发光二极管和 LED 显示器
(a)发光二极管; (b)数码管。

图 8-1 中,LED 显示器内部是 8 只发光二极管,分别记为 a、b、c、d、e、f、g、dp,其中除 dp 制成圆形用以表示小数点外,其余 7 只全部制成条形,并排列成如图 8-1(b)所示的“8”字形状。每只发光二极管都有一根电极引到外部引脚上,而另外一根电极全部连接在一起,引到外引脚,称为公共极(COM)。

图 8-2 是 LED 显示器的共阳极和共阴极电路原理图。

图 8-2(a)是把各个发光二极管的阳极都连在一起,从 COM 端引出,阴极分别从其他 8 根引脚引出,称为共阳结构。使用时,公共阳极接+5V,这样,阴极端输入低电平的发光

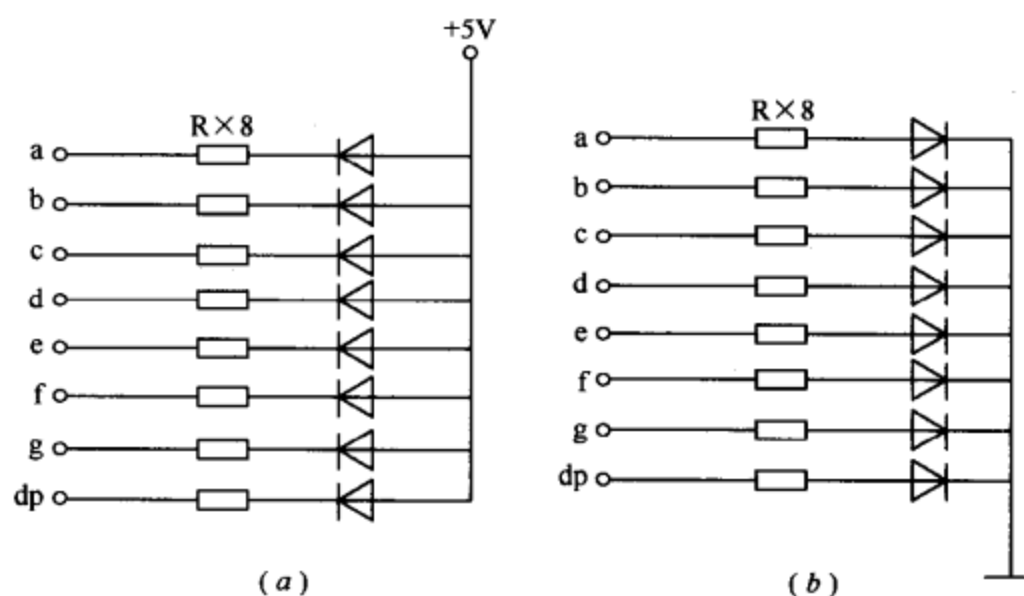


图 8-2 共阳和共阴 LED 显示器
(a)共阳结构；(b)共阴结构。

二极管就导通点亮，而输入高电平的段则不能点亮。

图 8-2(b)是把各个发光二极管的阴极都接在一起，从 COM 端引出，阳极分别从其他 8 根引脚引出，称为共阴结构。使用时，公共阴极接地，这样，阳极端输入高电平的发光二极管就导通点亮，而输入低电平的段则不能点亮。

在购买和使用 LED 显示器时，必须说明是共阴还是共阳结构。

2. LED 显示器的原理

根据 LED 显示器结构可知，如果希望显示“8”字，那么除了“dp”管不要点亮以外，其余管全部点亮，其余均不必点亮。同理，如果要显示“1”，那么，只需 b、c 两个发光二极管点亮。对于共阳结构，就是要把公共端 COM 接到电源正极，而 b、c 两个负极分别经过一个限流电阻后接低电平；对于共阴结构，就是要把公共端 COM 接低电平（电源负极），而 b、c 两个正极分别经一个限流电阻后接到高电平。按照同样的方法分析其他显示数和字型码，如表 8-1 所列。

表 8-1 8 段 LED 数码管段位与显示字型码的关系

显示	共 阳									共 阴								
	dp	g	f	e	d	c	b	a	16 进制数	d p	g	f	e	d	c	b	a	16 进制数
0	1	1	0	0	0	0	0	0	0C0H	0	0	1	1	1	1	1	1	3FH
1	1	1	1	1	1	0	0	1	0F9H	0	0	0	0	0	1	1	0	06H
2	1	0	1	0	0	1	0	0	0A4H	0	1	0	1	1	0	1	1	5BH
3	1	0	1	1	0	0	0	0	0B0H	0	1	0	0	1	1	1	1	4FH
4	1	0	0	1	1	0	0	1	99H	0	1	1	0	0	1	1	0	66H
5	1	0	0	1	0	0	1	0	92H	0	1	1	0	1	1	0	1	6DH
6	1	0	0	0	0	0	1	0	82H	0	1	1	1	1	1	0	1	7DH
7	1	1	1	1	1	0	0	0	0F8H	0	0	0	0	0	1	1	1	07H
8	1	0	0	0	0	0	0	0	80H	0	1	1	1	1	1	1	1	7FH

(续)

显示	共 阳									共 阴								
	dp	g	f	e	d	c	b	a	16 进制数	d p	g	f	e	d	c	b	a	16 进制数
9	1	0	0	1	0	0	0	0	90H	0	1	1	0	1	1	1	1	6FH
A	1	0	0	0	1	0	0	0	88H	0	1	1	1	0	1	1	1	77H
B	1	0	0	0	0	0	1	1	83H	0	1	1	1	1	1	0	0	7CH
C	1	1	0	0	0	1	1	0	0C6H	0	0	1	1	1	0	0	1	39H
D	1	0	1	0	0	0	0	1	0A1H	0	1	0	1	1	1	1	0	5EH
E	1	0	0	0	0	1	1	0	86H	0	1	1	1	1	0	0	1	79H
F	1	0	0	0	1	1	1	0	8EH	0	1	1	1	0	0	0	1	71H
H	1	0	0	0	1	0	0	1	89H	0	1	1	1	0	1	1	0	76H
L	1	1	0	0	0	1	1	1	0C7H	0	0	1	1	1	0	0	0	38H
P	1	0	0	0	1	1	0	0	8CH	0	1	1	1	0	0	1	1	73H
U	1	1	0	0	0	0	0	1	0C1H	0	0	1	1	1	1	1	0	3EH
Y	1	0	0	1	0	0	0	1	91H	0	1	1	0	1	1	1	0	6EH
灭	1	1	1	1	1	1	1	1	0FFH	0	0	0	0	0	0	0	0	00H

重点提示 这种规定和定义并非是一成不变的,在实际应用中,为了减少走线交叉便于电路板布线,设计者可自行定义 8 段 LED 显示器的引脚符号,并根据其工作原理编制相应的“段码表”以及设计与之相匹配的连接电路。

二、LED 显示器的显示方式

点阵式显示器件不仅可以显示数字,还可显示各种符号、文字甚至图形,但点阵式显示电路较复杂,仅使用硬件电路难以实现,必须在单片机(如 8051)的控制下配以相应的软件才能实现。下面主要以 8 段 LED 显示器为例来介绍数码显示的基本方法和相关电路。

1. 静态显示方法

所谓静态显示,就是当显示某一个数字时,代表相应笔划的发光二极管恒定发光,例如:8 段数码管的 a、b、c、d、e、f 笔段亮时显示数字“0”;b、c 亮时显示“1”;a、b、d、e、g 亮时显示“2”等。

图 8-3 是 LED 显示器静态显示电路。每位数码管的公共端 COM 接在一起接正电压(共阳显示器)或接地(共阴显示器)。段选线分别通过限流电阻与段驱动电路连接。限流电阻的阻值根据驱动电压和 LED 显示器的额定电流确定。

静态显示的优点是显示稳定,在驱动电流一定的情况下显示的亮度高,缺点是使用元器件较多(每一位都需要一个驱动器,每一段都需要一个限流电阻),连接线多。

2. 动态显示方法

上面介绍的静态显示方法的最大缺点是使用元件多、引线多、电路复杂,而动态显示

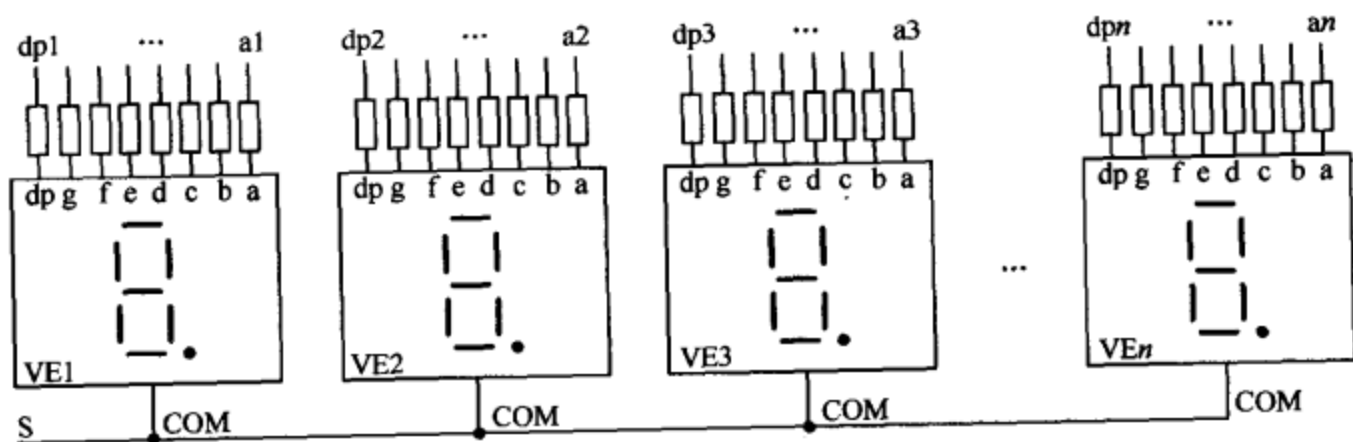


图 8-3 静态显示电路

使用的元件少、引线少、电路简单。仅从引线角度考，静态显示从显示器到控制电路的基本引线数为“段数 \times 位数”，而动态显示从显示器到控制电路的基本引线数为“段数+位数”。以 6 位显示为例，动态显示时的基本引线数为 $7+6=13$ （无小数点）或 $8+6=14$ （有小数点），而静态显示的基本引线数为 $7\times 6=42$ （无小数点）或 $8\times 6=48$ （有小数点）。在实际应用中，显示器与主电路往往不会安装在同一块电路板上，有时甚至有一段距离，因此静态显示的引线数大多会给实际安装、加工工艺带来困难。

动态显示所显示的若干位数是逐位轮流显示的，周而复始不断循环，只要轮流的速度足够快（每秒轮流 50 次以上），由于人眼“视觉暂留”的物性，感觉不到显示器的闪动，所看到的是连续显示一组数字。图 8-4(a) 是 3 位 LED 显示器采用动态显示方法的接线图，它把 3 个 LED 显示器的各个段对应并联，而各位的公共极（本图采用共阴 LED 显示器，公共极是发光二极管的阴极）受到独立控制，控制信号分别为 S1、S2、S3，其时序关系如图 8-4(b) 所示。

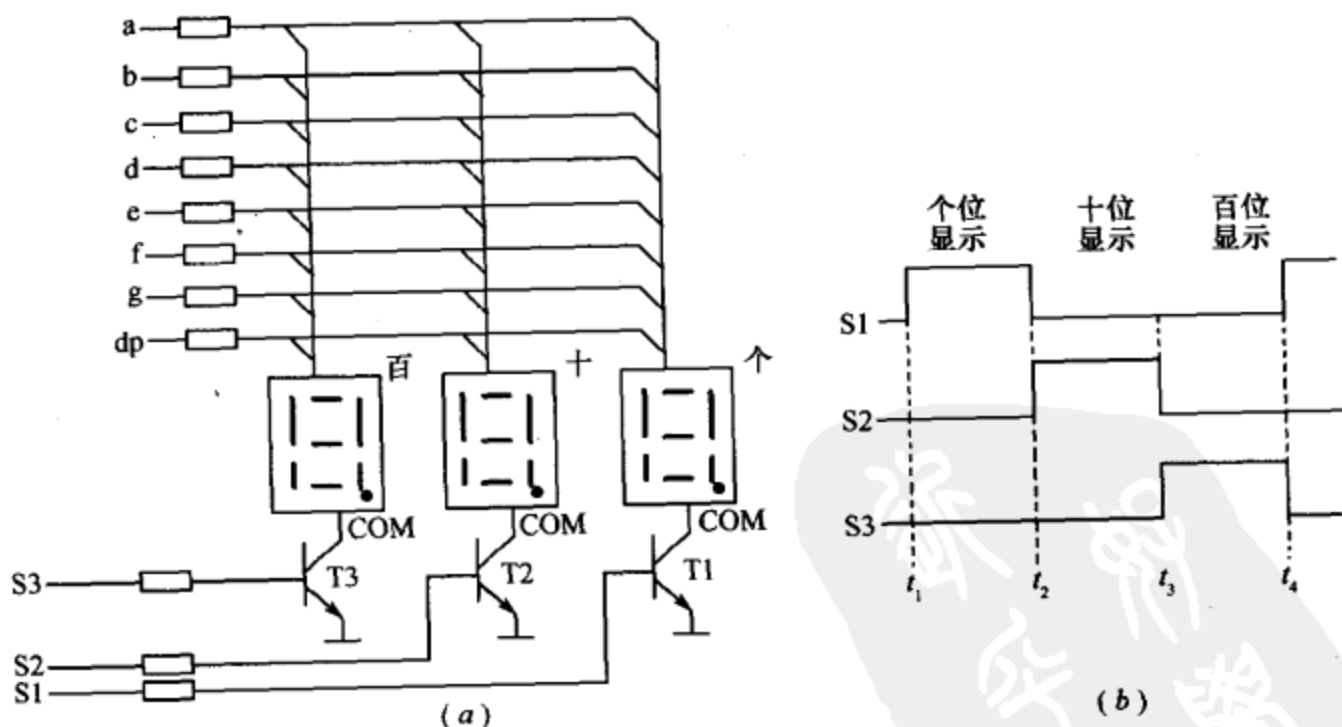


图 8-4 动态显示电路

(a) 接线图；(b) 时序图。

在 $t_1\sim t_2$ 期间，单片机输出个位段码加到各个 LED 显示器数码管的阳极 dp~a；在这期间，控制信号 S1 为高电平，三极管 T1 饱和导通，个位 LED 显示器的阴极接地，使个位 LED 显示器相应的字段点亮，显示个位数；而控制信号 S2、S3 为低电平，故三极管 T2、

T3 截止,使十位、百位 LED 显示器的阴极不能接向低电平,因此十位、百位的灯不亮。在 $t_2 \sim t_3$ 期间,单片机输出“十位”的段码加到各个 LED 显示器数码管的阳极 $dp \sim a$;相应地使控制信号 S2 为高电平,而 S1、S3 为低电平,故只有三极管 T2 饱和导通,而 T1、T3 截止,即只有“十位”LED 显示器点亮。同理在 $t_3 \sim t_4$ 期间,只有“百位”LED 显示器点亮。此规律周而复始不断循环,就使得个位、十位、百位 3 个 LED 显示器依次轮流循环显示,这就是动态扫描显示的原理,控制信号 S1、S2、S3 称为“位扫描”信号。从以上分析可以看到,动态扫描显示方式可以十分有效地减少元件、减少引线,尤其是在显示位数较多时,其优越性就更为突出,因此在单片机显示电路中,大多采用这种方式。

三、几种常见的显示接口电路

显示接口电路的形式很多,下面介绍几种常见的显示接口电路。

1. 多只共阳 LED 显示器静态锁存

当系统需要多只 LED 显示器而并口又不够用时,需要扩展并口,下面以多只共阳 LED 显示器静态锁存接口电路为例进行介绍,有关电路如图 8-5 所示。

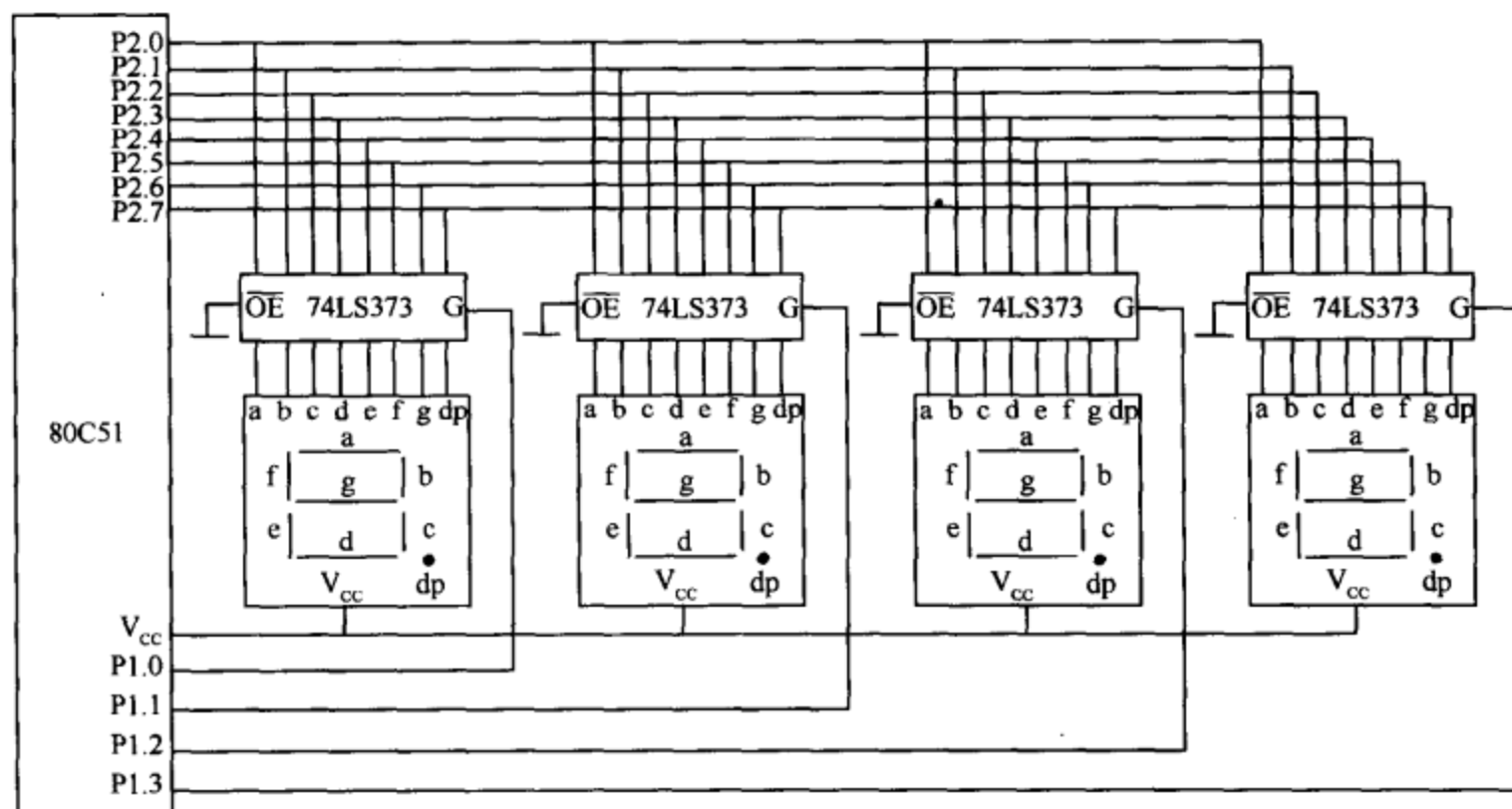


图 8-5 多只共阳 LED 显示器静态锁存

4 只 74LS373 的输入端并接在一起,输出端接 4 只 LED 显示器的段选线(a~dp),输出允许 OE 端接地,表示任何时刻都将 74LS373 的输入数据输出到 LED 显示器的段选线上,4 只 74LS373 的锁存脉冲输入端 G 受控于 P1.0、P1.1、P1.2、P1.3。4 只共阳 LED 显示器的公共端接 V_{cc} 。

下面是显示“2005”4 个数据的源程序:

```
MOV P1, #0      ;向 74LS373 的锁存控制端送 0,使 74LS373 输出与输入断开
MOV P2, #0A4H   ;字型 2 的显示码送 P2 口
SETB P1.0       ;将第 1 只 74LS373 输出与输入连接,使字型 2 段选码出现在段选线上
```


CLR P1.0 ;将第 1 只 74LS373 输出与输入断开
 MOV P2, #0C0H ;字型 0 的显示码送 P2 口
 SETB P1.1 ;将第 2 只 74LS373 输出与输入连接,使字型 0 段选码出现在段选线上
 CLR P1.1 ;将第 2 只 74LS373 输出与输入断开
 MOV P2, #0C0H ;字型 0 的显示码送 P2 口
 SETB P1.2 ;将第 3 只 74LS373 输出与输入连接,使字型 0 段选码出现在段选线上
 CLR P1.2 ;将第 3 只 74LS373 输出与输入断开
 MOV P2, #92H ;字型 5 的显示码送 P2 口
 SETB P1.3 ;将第 4 只 74LS373 输出与输入连接,使字型 5 段选码出现在段选线上
 CLR P1.3 ;将第 4 只 74LS373 输出与输入断开

2. 多只共阴 LED 静态译码锁存

采用 74LS373 线选法扩展多位静态显示占用单片机的并口引脚仍然较多,比如 4 条端口线作位选线时,只能扩展 4 只 LED 显示器,如果位选线采用译码方式,那么,同样用 4 条端口线就可以扩展 16 只显示器,下面以采用 74LS139 译码器的多只共阴 LED 静态译码锁存电路为例进行介绍。有关电路如图 8-6 所示。

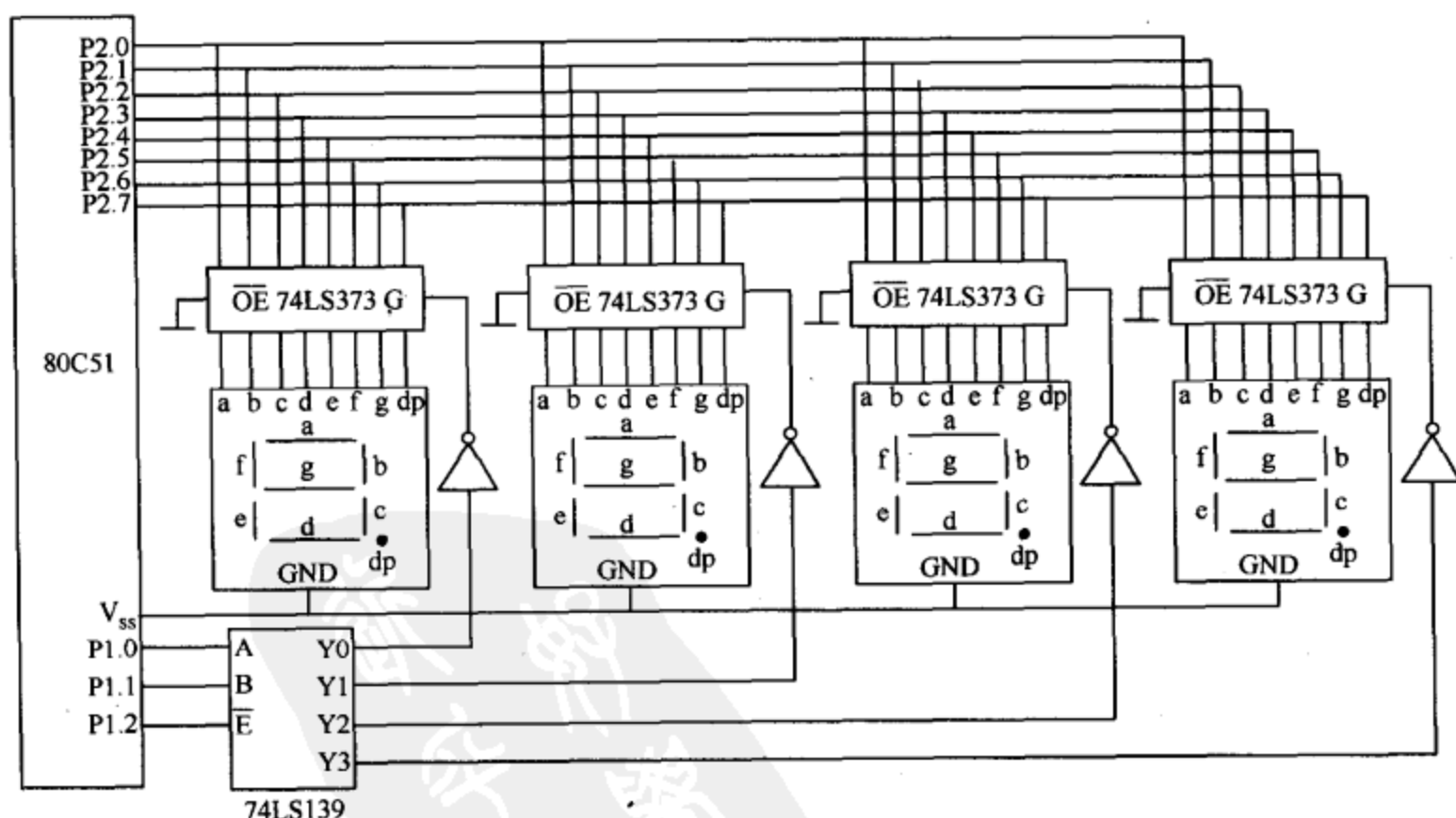


图 8-6 多只共阴 LED 静态译码锁存电路

74LS139 为双 2-4 译码器,其引脚排列如图 8-7 所示。

A、B 为选择端,即译码输入,控制译码输出的有效性;Y0、Y1、Y2、Y3 为译码输出信号,低电平有效; \bar{E} 为使能端,低电平有效。74LS139 对输入信号译码后得到 4 个输出状

态,其真值表如表 8-2 所列。

4 只 74LS373 的输入端并接在一起,输出端接 4 只 LED 显示器的段选线(a~dp),输出允许 \overline{OE} 端接地,表示任何时刻都将 74LS373 的输入数据输出到 LED 的段选线上,4 只 74LS373 的锁存脉冲输入端 G 受控于译码器 74LS139 的 Y0、Y1、Y2、Y3。由于 74LS139 为低电平有效,所以,74LS139 的输出必须经反相才能控制 4 只 74LS373。P1.0、P1.1 连接到 74LS139 的输入端 B、A,P1.2 用于控制 74LS139 的使能端 \overline{E} 。4 只共阴 LED 显示器的公共端接 GND。

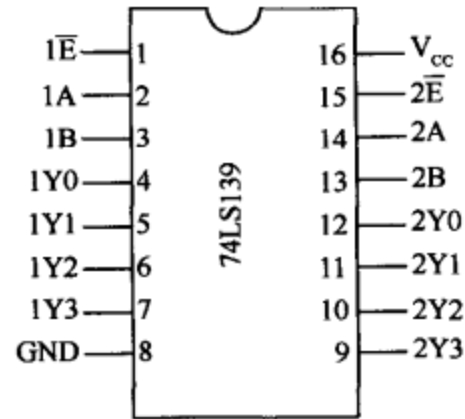


图 8-7 74LS139 引脚排列

表 8-2 74LS139 真值表

输入端			输出端			
使能	选择		Y0	Y1	Y2	Y3
\overline{E}	B	A				
1	×	×	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

下面是显示“2005”4 个数据的源程序：

```
SETB P1.2      ;向 4 只 74LS373 的 G 端送低电平,使输出与输入断开
MOV P2, #5BH   ;字型 2 的显示码送 P2 口
MOV P0, #0      ;第 1 只 74LS373 输出与输入接通,使字型 2 显示码出现在第 1
                ;只 LED 上
SETB P1.2      ;将第 1 只 74LS373 的输出与输入断开
MOV P2, #3FH   ;字型 0 的显示码送 P2 口
MOV P0, #1      ;第 2 只 74LS373 输出与输入接通,使字型 0 显示码出现在第 2
                ;只 LED 上
SETB P1.2      ;将第 2 只 74LS373 的输出与输入断开
MOV P2, #3FH   ;字型 0 的显示码送 P2 口
MOV P0, #2      ;第 3 只 74LS373 输出与输入接通,使字型 0 显示码出现在第 3
                ;只 LED 上
SETB P1.2      ;将第 3 只 74LS373 的输出与输入断开
MOV P2, #6DH   ;字型 5 的显示码送 P2 口
MOV P0, #3      ;第 4 只 74LS373 输出与输入接通,使字型 0 显示码出现在第 4
                ;只 LED 上
SETB P1.2      ;将第 4 只 74LS373 的输出与输入断开
```

3. 串行口多位共阴 LED 静态显示

80C51 单片机应用系统中,如果并行 I/O 接口不够,而串行口又没有其他用处时,则可用来扩展并行 I/O 接口,从而节省了单片机的硬件资源。80C51 单片机内部的串行口在方式 0 工作状态下,使用移位寄存器芯片可以扩展一个或多个 8 位并行 I/O 接口,下面以 4 位共阴 LED 串行口静态显示电路为例进行说明,有关电路如图 8-8 所示。

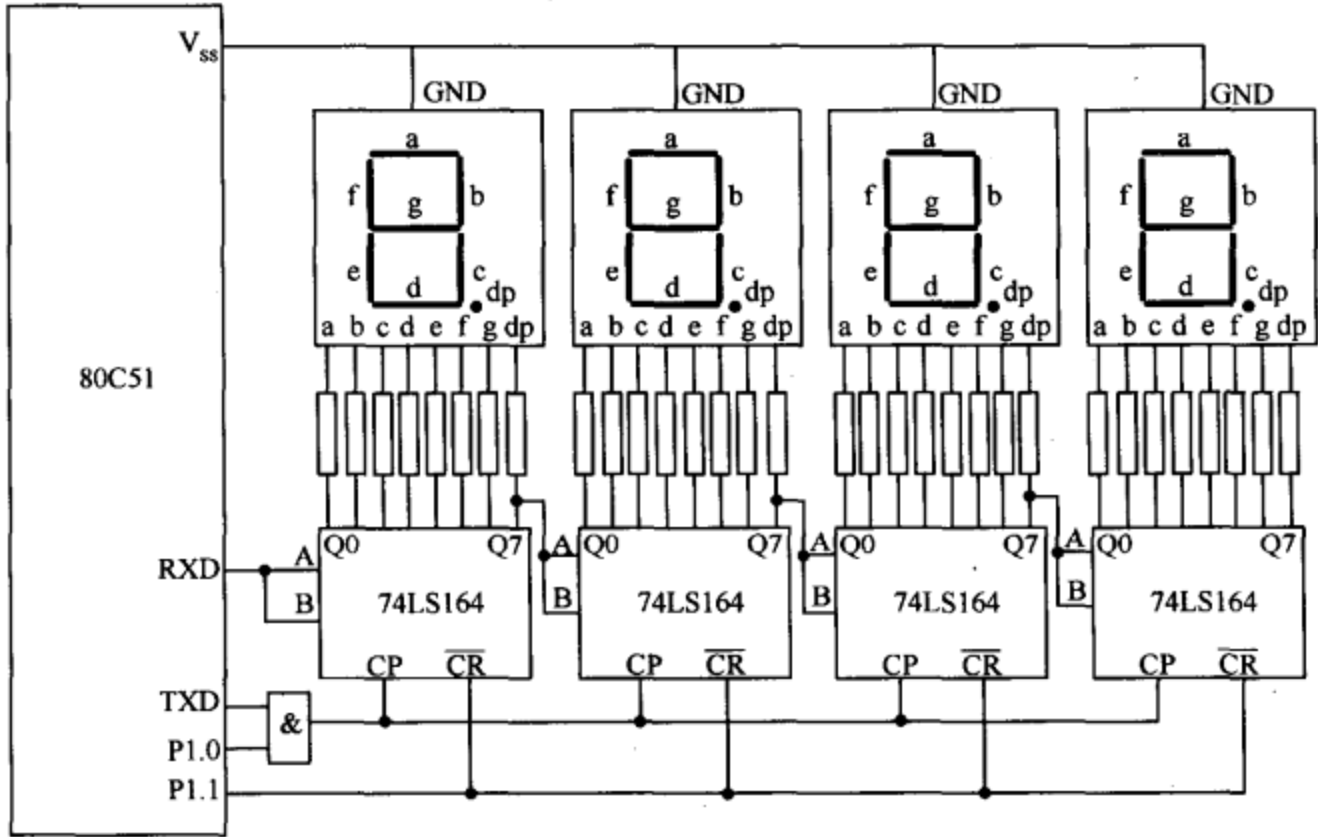


图 8-8 串行口多位共阴 LED 静态显示

74LS164 是 8 位串行输入、并行输出移位寄存器,并带有清除端。其引脚如图 8-9 所示。

图中,A、B 为串行数据输入端,实际使用时把 A、B 连在一起;CP 为移位时钟输入端,当 CP 信号为上跳沿时,数据右移一位; $\overline{\text{CR}}$ 为清零输入端,低电平有效,当该端加低电平时,所有输出端 $\text{Q0} \sim \text{Q7}$ 均为低电平; $\text{Q0} \sim \text{Q7}$ 为并行数据输出端,同时 Q7 端也是串行数据输出端,串行数据(低位在前)从 A、B 端最先进入的从 Q0 端输出,最后进入的从 Q7 输出。

8 位单向移位寄存器 74LS164 状态表如表 8-3 所示。

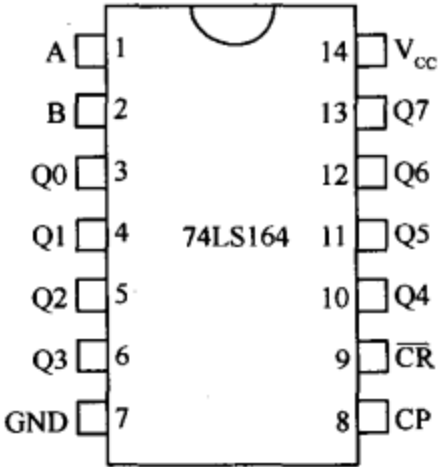


图 8-9 74LS164 管脚排列图

表 8-3 8 位单向移位寄存器 74LS164 状态表

输入			输出								说明
$\overline{\text{CR}}$	A · B	CP	Q0^{n+1}	Q1^{n+1}	Q2^{n+1}	Q3^{n+1}	Q4^{n+1}	Q5^{n+1}	Q6^{n+1}	Q7^{n+1}	
0	×	×	0	0	0	0	0	0	0	0	清零
1	×	0	Q0^n	Q1^n	Q2^n	Q3^n	Q4^n	Q5^n	Q6^n	Q7^n	保持
1	1	↑	1	Q0^n	Q1^n	Q2^n	Q3^n	Q4^n	Q5^n	Q6^n	输入一个 1
1	0	↑	0	Q0^n	Q1^n	Q2^n	Q3^n	Q4^n	Q5^n	Q6^n	输入一个 0

采用串行口扩展显示器节省了 I/O 接口,但传送速度较慢;扩展的芯片越多,速度越慢。

下面是实现“2005”和“8051”交替显示的源程序(交替时间为 0.5s):

```
MAIN: MOV SCON, #10H      ;设置为工作方式 0,REN=1,允许接收
      CLR P1.1             ;关闭 4 个 LED 显示器
      SETB P1.0            ;允许显示输入
      SETB P1.1            ;开放显示器
      MOV A, #6DH          ;输入字型 5 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST0:   JNB TI, ST0          ;TI=0 等待发送, TI=1 发送
      CLR TI               ;清 TI
      MOV A, #3FH          ;输入字型 0 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST1:   JNB TI, ST1          ;TI=0 等待发送, TI=1 发送
      CLR TI               ;清 TI
      MOV A, #3FH          ;输入字型 0 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST2:   JNB TI, ST2          ;TI=0 等待发送, TI=1 发送
      CLR TI               ;清 TI
      MOV A, #5BH          ;输入字型 2 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST3:   JNB TI, ST3          ;TI=0 等待发送, TI=1 发送
      CLR TI               ;清 TI
      CLR P1.0             ;保持 74LS164 中的数据
      LCALL DELAY          ;调延时子程序
      CLR P1.1             ;关闭 4 个 LED 显示器
      SETB P1.0            ;允许显示输入
      SETB P1.1            ;开放显示器
      MOV A, #06H          ;输入字型 1 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST4:   JNB TI, ST4          ;TI=0 等待发送, TI=1 发送
      CLR TI               ;清 TI
      MOV A, #6DH          ;输入字型 5 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST5:   JNB TI, ST5          ;TI=0 等待发送, TI=1 发送
      CLR TI               ;清 TI
      MOV A, #3FH          ;输入字型 0 的显示代码
      MOV SBUF, A          ;将数据送发送缓冲器
ST6:   JNB TI, ST6          ;TI=0 等待发送, TI=1 发送
```

	CLR TI	;清 TI
	MOV A, #7FH	;输入字型 8 的显示代码
	MOV SBUF, A	;将数据送发送缓冲器
ST7:	JNB TI, ST7	;TI=0 等待发送, TI=1 发送
	CLR TI	;清 TI
	CLR P1.0	;保持 74LS164 中的数据
	LCALL DELAY	;调延时子程序
	LJMP MAIN	
;以下是 0.5s 延时程序		
DELAY:	MOV R5, #250	;1 个机器周期
D2:	MOV R4, #250	;1 个机器周期
D1:	NOP	;1 个机器周期
	NOP	;1 个机器周期
	NOP	;1 个机器周期
	NOP	;1 个机器周期
	NOP	;1 个机器周期
	NOP	;1 个机器周期
	DJNZ R4, D1	;2 个机器周期
	DJNZ R5, D2	;2 个机器周期
	RET	;2 个机器周期

4. 多位共阳 LED 线选动态显示

前面已经讲过,动态显示是指一个系统的若干个 LED 显示器,任何时候只显示其中的一只,其余显示器都不显示。下面介绍多位共阳 LED 线选动态显示电路,如图 8-10 所示。

图 8-10 中,4 只共阳 LED 显示器的段选位和 P2 口相连,4 只 LED 显示器的 V_{CC} 端受 P1.0、P1.1、P1.2、P1.3 控制,这 4 条控制线中,在某一时刻只允许一条线为高电平,其余 3 条线为低电平,这就使得某一时刻只有一个 LED 显示器工作。

显示 2005 的源程序如下:

	MOV P1, #0	;使 4 只显示器都不亮
	MOV P2, #0A4H	;字型 2 的显示码送 P2
DISPLAY:	SETB P1.0	;第 1 只显示器显示 2
	ACALL DELAY	;调延时子程序
	CLR P1.0	;第 1 只显示器熄灭
	MOV P2, #0C0H	;字型 0 的显示码送 P2
	SETB P1.1	;第 2 只显示器显示 0
	ACALL DELAY	;调延时子程序
	CLR P1.1	;第 2 只显示器熄灭
	MOV P2, #0C0H	;字型 0 的显示码送 P2
	SETB P1.2	;第 3 只显示器显示 0

ACALL DELAY	;调延时子程序
CLR P1.2	;第3只显示器熄灭
MOV P2, #92H	;字型5的显示码送P2
SETB P1.3	;第4只显示器显示5
ACALL DELAY	;调延时子程序
CLR P1.3	;第4只显示器熄灭
AJMP DISPLAY	

;以下是10ms延时子程序

DELAY: MOV R5, #50	;1个机器周期	
D2: MOV R4, #100	;1个机器周期	
D1: DJNZ R4, D1	;2个机器周期	
	DJNZ R5, D2	;2个机器周期
	RET	;2个机器周期

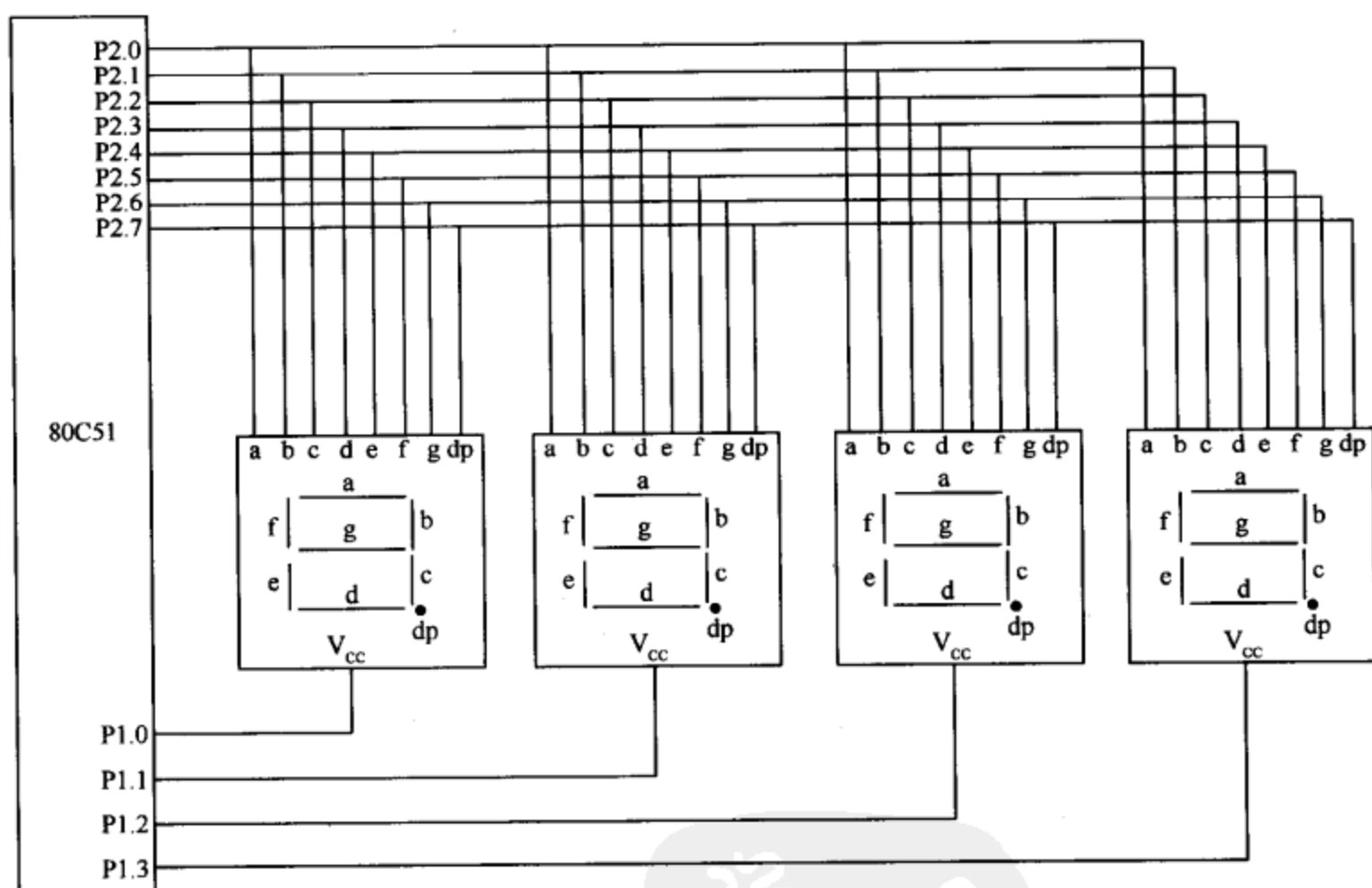


图 8-10 多位共阳 LED 线选动态显示电路

实际上,这是一段死循环程序,此后,这4只显示器被不断地点亮,由于人眼的视觉暂留效应,因而看不到闪烁现象。

5. 多位共阴 LED 译码动态显示

多位共阴 LED 译码动态显示电路如图 8-11 所示。

4只共阴 LED 显示器的 GND 端受译码器 74LS139 的控制,74LS139 的4个输出端 Y0~Y3 任何时刻只有一条为低电平,这样,就使在任何时刻4只 LED 显示器只有一个显示,P2 口发送字型码送到每个 LED 显示器的段选位上。

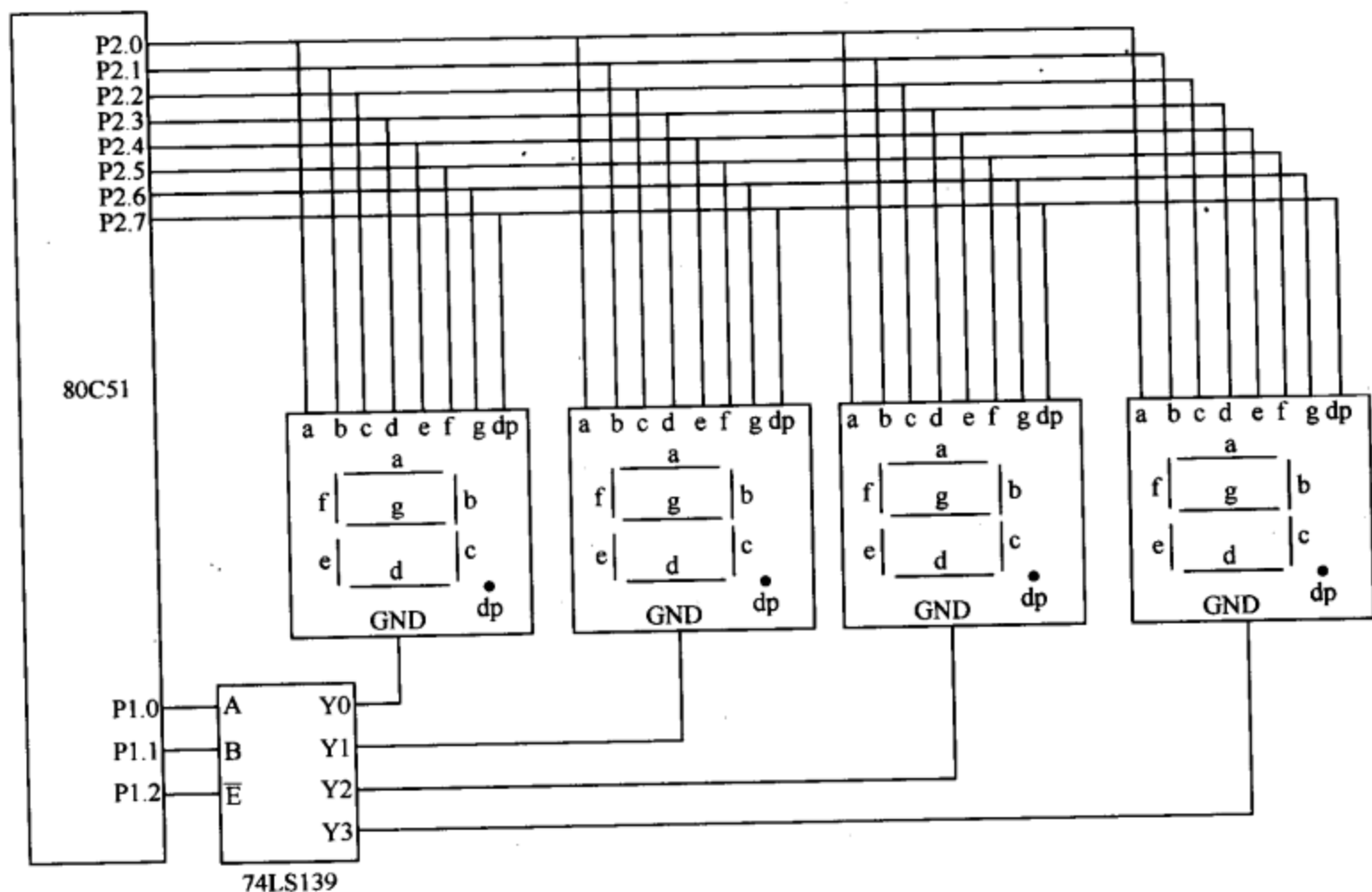


图 8-11 多位共阴 LED 译码动态显示电路

设 20H~23H 为键盘缓冲区,要求将这 4 个单元中键名分别送 4 只显示器显示,由于缓冲区中有效的数据不是显示码,所以,在程序中必须将缓冲区中的键名转换成键名所对应的显示码。源程序如下:

SETB P1.2	;将 4 只显示器都关闭
MOV DPTR, #TAB	;将显示码的基址装入 DPTR
DISPLAY: MOV A, 20H	;从键值缓冲区的第 1B 中取键码
MOVC A, @A+DPTR	;从程序存储器中取该键码所对应的显示码
MOV P2, A	;将显示码送 P2 口
MOV P1, #0	;将第 1 只显示器点亮
ACALL DELAY	;调延时子程序
SETB P1.2	;关闭 4 只显示器
MOV A, 21H	;从键值缓冲区的第 2B 中取键码
MOVC A, @A+DPTR	;从程序存储器中取该键码所对应的显示码
MOV P2, A	;将显示码送 P2 口
MOV P1, #1	;将第 2 只显示器点亮
ACALL DELAY	;调延时子程序
SETB P1.2	;关闭 4 只显示器
MOV A, 22H	;从键值缓冲区的第 3B 中取键码
MOVC A, @A+DPTR	;从程序存储器中取该键码所对应的显示码
MOV P2, A	;将显示码送 P2 口
MOV P1, #2	;将第 3 只显示器点亮

ACALL DELAY	;调延时子程序	
SETB P1.2	;关闭4只显示器	
MOV A,23H	;从键值缓冲区的第4B中取键码	
MOVC A,@A+DPTR	;从程序存储器中取该键码所对应的显示码	
MOV P2,A	;将显示码送P2口	
MOV P1,#3	;将第4只显示器点亮	
ACALL DELAY	;调延时子程序	
SETB P1.2	;关闭4只显示器	
AJMP DISPLAY		
;以下是10ms延时子程序		
DELAY: MOV R5,#50	;1个机器周期	
D2: MOV R4,#100	;1个机器周期	
D1: DJNZ R4,D1	;2个机器周期	
	DJNZ R5,D2	;2个机器周期
	RET	;2个机器周期
TAB: DB 3FH,06H,5BH,4FH	;0~3的显示代码	
	DB 66H,6DH,7DH,07H	;4~7的显示代码
	DB 7FH,6FH,77H,7CH	;8~B的显示代码
	DB 39H,5EH,79H,71H	;C~F的显示代码

四、LED 显示器接口实验

实验1 AT89C51 实验开发板中有两个 LED 数码管,试利用其中的一只 LED 数码管做实验,循环显示 0~9,时间间隔为 1s。有关电路如图 8-12 所示。

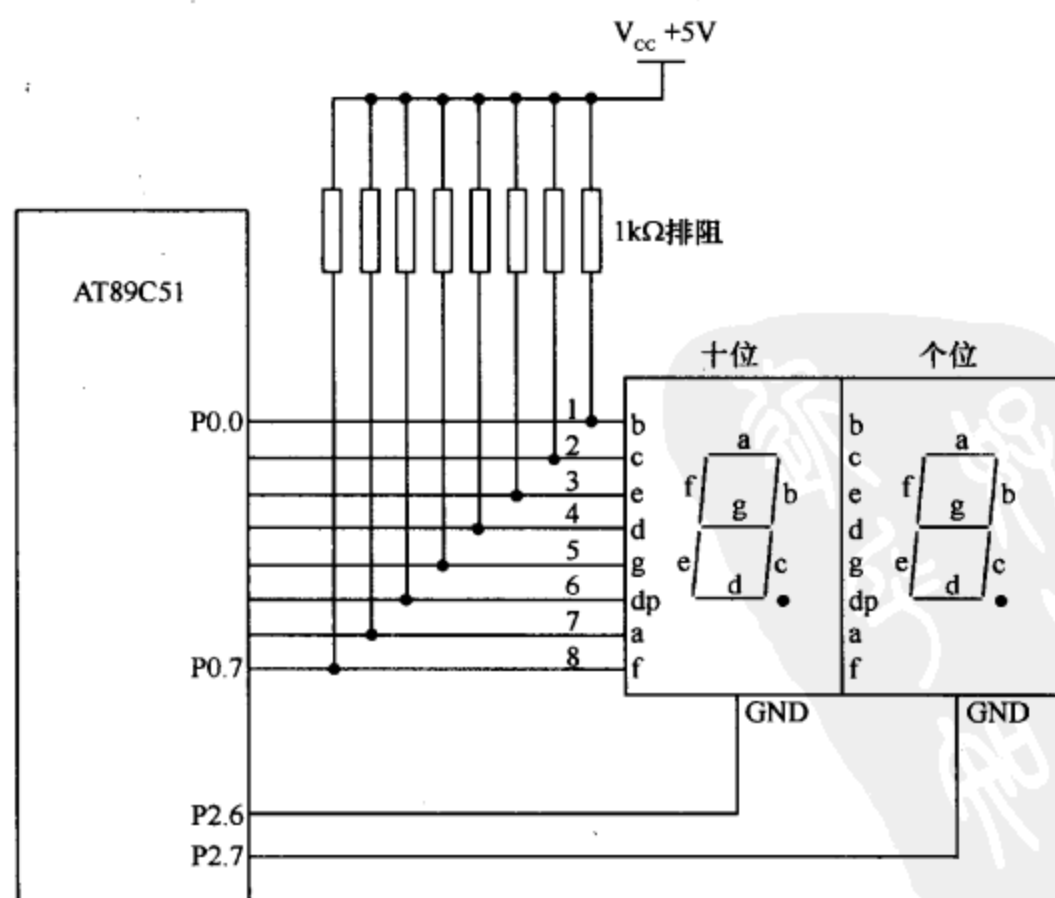


图 8-12 AT89C51 实验开发板数码管电路

图 8-12 中, P0 口和 P2 口的部分(P2. 6、P2. 7)引脚构成了两位 LED 数码管驱动电路, 数码管采用了共阴型。即对应的 a、b、c、d、e、f、dp 是二极管的正极, 所有二极管的负极连在一起, 构成公共端。+5V 通过 1k Ω 的排阻直接给数码管的 8 个段位供电, P2. 6 和 P2. 7 分别控制数码管的十位和个位的供电, 当 P2. 6 或 P2. 7 变成低电平时, 相应的位可以吸入电流。这里, 以个位 LED 数码管循环显示 0~9 为例进行实验, 源程序如下:

```

ORG 0000H
AJMP MAIN
;以下是主程序
ORG 0100H
MAIN:  MOV R2, #00H           ;段码表地址指针清 0
        MOV DPTR, #TAB        ;指向段码表起始地址
DSUP:   MOV A, R2              ;将段码表地址指针的值送 A
        MOVC A, @A+DPTR       ;将显示字型段码表送 A
        MOV P0, A              ;从 P0 口输出段码表
        SETB P2. 6            ;P2. 6 置 1, 十位 LED 数码管熄灭
        CLR P2. 7             ;P2. 7 清 0, 个位 LED 数码管点亮
        MOV R0, #02H          ;两次调用 0. 5s 延时子程序, 以实现 1s 延时
LOOP:   ACALL DELAY            ;调 0. 5s 延时程序
        DJNZ R0, LOOP         ;调用两次 0. 5s 延时程序后, 延时时间为 1s
        INC R2                ;段码表指针指向下一数字
        CJNE R2, #0AH, DSUP   ;0~9 数字是否显示结束, 若未结束继续显示
        AJMP MAIN             ;返回主程序循环显示
;以下是 0. 5s 延时程序
DELAY:  MOV R5, #250           ;1 个机器周期
D2:     MOV R4, #250           ;1 个机器周期
D1:     NOP                   ;1 个机器周期
        NOP                   ;1 个机器周期
        NOP                   ;1 个机器周期
        NOP                   ;1 个机器周期
        NOP                   ;1 个机器周期
        NOP                   ;1 个机器周期
        DJNZ R4, D1            ;2 个机器周期
        DJNZ R5, D2            ;2 个机器周期
        RET                   ;2 个机器周期
TAB:    DB 0CFH, 03H, 5DH, 5BH, 93H ;5~9 的共阴极段码表
        DB 0DAH, 0DEH, 43H, 0DFH, 0DBH ;0~4 的共阴极段码表

```

END

重点提示 AT89C51 实验开发板上的段码表与前面介绍的段码表不一致,这是因为该 LED 段与 P0 口的接法不一致造成的。LED 显示器的各段与 P0 口的对应关系如下所示:

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
f	a	dp	g	d	e	c	b

从对应关系中可以看出,当显示 P0 口输出的数为 0CFH(1100 1111)时,LED 数码管中对应的 fadecb 为高电平,因此,LED 数码管显示为 0,同理,可分析出数字 1,2,...,9 的段码表。

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 disp_1.asm。对程序进行编译、链接和调试,产生 disp_1.hex 目标文件。

(2)先可用硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电进行实验。

该实验程序在本书所附光盘的 example\ch_8\disp_1 文件夹中。

实验 2 用 AT89C51 实验开发板中的两个 LED 数码管,循环显示 00~99,时间间隔为 1s。

因为要显示两位不同的数字,所以必须用动态扫描的方法来实现,先个位显示 10ms,再十位显示 10ms,每秒扫描 50 次以上,因为人眼的视觉残留效应,就可以看到两位不同的数字稳定显示。如果取个位和十位循环显示次数为 50 次,则循环显示 00~99 的时间间隔为 $(10+10) \times 50 = 1000(\text{ms}) = 1\text{s}$ 。

源程序如下:

```
A_BIT EQU 20H           ;数码管个位数存放内存位置
B_BIT EQU 21H           ;数码管十位数存放内存位置
ORG 0000H
ACALL MAIN
;以下是主程序
ORG 0100H
MAIN:  MOV SP, #5FH      ;设置堆栈
       MOV R2, #0        ;段码表地址指针 R2 清 0
LOOP:  ACALL DISPLAY     ;调用显示子程序
       INC R2            ;段码表地址指针 R2 加 1
       MOV A, R2
       CJNE A, #100, NEXT ;判断计数器是否满 100?
       MOV R2, #0        ;满 100 就清零重新开始
NEXT:  LJMP LOOP         ;不满就循环执行
;以下是显示子程序
```


DISPLAY: PUSH ACC	
MOV A, R2	;将 R2 中的十六进制数转换成 10 进制
MOV B, #0AH	;将 16 进制的 0AH 转换成 10 进制
DIV AB	
MOV B_BIT, A	;转换为十进制后的十位在 A
MOV A_BIT, B	;转换为十进制后的个位在 B
MOV DPTR, #TAB	;指定查表起始地址
MOV R1, #50	;循环显示 50 次,
LOOP1: MOV A, A_BIT	;取个位数
MOVC A, @A+DPTR	;查个位数的 7 段代码
MOV P0, A	;送出个位的 7 段代码
CLR P2.7	;开个位显示
ACALL DELAY	;显示 10ms
SETB P2.7	;关闭个位显示,防止“鬼影”
MOV A, B_BIT	;取十位数
MOVC A, @A+DPTR	;查十位数的 7 段代码
MOV P0, A	;送出十位的 7 段代码
CLR P2.6	;开十位显示
ACALL DELAY	;显示 10ms
SETB P2.6	;关闭十位显示,防止鬼影
DJNZ R1, LOOP1	;循环执行 50 次,时间间隔为(10+10)×50=1(s)
POP	ACC
RET	
;以下是 10ms 延时子程序	
DELAY: MOV R5, #50	;1 个机器周期
D2: MOV R4, #100	;1 个机器周期
D1: DJNZ R4, D1	;2 个机器周期
DJNZ R5, D2	;2 个机器周期
RET	;2 个机器周期
TAB: DB 0CFH, 03H, 5DH, 5BH, 93H	;0~4 数字共阴代码
DB 0DAH, 0DEH, 43H, 0DFH, 0DBH	;5~9 数字共阴代码
END	

重点提示 从这个实验例子可以看出,动态扫描显示必须由 CPU 不断地调用显示子程序,才能保证持续不断的显示。

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 disp_2. asm。对程序进行编译、链接和调试,产生 disp_2. hex 目标文件。

(2)先可用硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片

编程。

(3)将 AT89C51 芯片插入实验开发板,通电进行实验。

该实验程序在本书所附光盘的 example\ch_8\disp_2 文件夹中。

实验 3 用 AT89C51 实验开发板中的两个数码管交替显示“80”和“51”,显示时间间隔为 1s。

源程序如下:

```
ORG 0000H
AJMP MAIN
ORG 0100H
;以下是主程序
MAIN: MOV P0, #0FFH
      MOV P2, #0FFH      ;初始化,关闭 2 个 LED 显示器
;以下显示“80”
      MOV R1, #50        ;设置循环显示次数为 50 次
LOOP0: MOV A, #0DFH      ;输入字型 8 的显示代码
      MOV P0, A          ;从 P0 口输出 8 的显示代码
      CLR P2.6           ;十位显示字型 8
      ACALL DELAY        ;调 10ms 延时子程序
      SETB P2.6;        ;关十位显示
      MOV A, #0CFH       ;输入字型 0 的显示代码
      MOV P0, A          ;从 P0 口输出 0 的显示代码
      CLR P2.7           ;个位显示字型 0
      ACALL DELAY        ;调 10ms 延时子程序
      SETB P2.7         ;关个位显示
      DJNZ R1, LOOP0     ;判断是否循环 50 次
;以下显示“51”
      MOV R1, #50        ;继续设置循环显示次数为 50 次
LOOP1: MOV A, #0DAH      ;输入字型 5 的显示代码
      MOV P0, A          ;从 P0 口输出 5 的显示段码
      CLR P2.6           ;十位显示字型 5
      ACALL DELAY        ;调 10ms 延时子程序
      SETB P2.6         ;关十位显示
      MOV A, #03H        ;输入字型 1 的显示代码
      MOV P0, A          ;从 P0 口输出 1 的显示段码
      CLR P2.7           ;个位显示字型 1
      ACALL DELAY        ;调 10ms 延时子程序
      SETB P2.7         ;关个位显示
      DJNZ R1, LOOP1     ;判断是否循环显示次数为 50 次
      AJMP MAIN          ;反复循环
```

;以下是 10ms 延时子程序

```
DELAY: MOV R5, #50      ;1 个机器周期
D2:    MOV R4, #100     ;1 个机器周期
D1:    DJNZ R4, D1       ;2 个机器周期
        DJNZ R5, D2      ;2 个机器周期
        RET              ;2 个机器周期
        END
```

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 disp_3.asm。对程序进行编译、链接和调试,产生 disp_3.hex 目标文件。

(2)先可用硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电进行实验。

该实验程序在本书所附光盘的 example\ch_8\disp_3 文件夹中。

第二节 键盘接口

键盘是单片机十分重要的输入设备,是实现人机对话的纽带。键盘是由一组规则排列的按键组成,一个按键实际上就是一个开关元件,即键盘是一组规则排列的开关。根据按键结构、原理的不同,主要分为两类:一类是触点式开关按键,如机械开关式等;另一类是无触点开关按键,如磁感应按键等。前者造价低,后者寿命长。目前,单片机应用系统中最常见的是触点式开关按键。

一、键盘的工作原理

1. 键盘的特性

键盘是由一组按键开关组成的。通常,按键所用开关为机械弹性开关,这种开关一般为常开型。平时(按键不按下时),按键的触点是断开状态,按键被按下时,它们才闭合。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一定的弹跳时间,即在这段时间里连续产生多个脉冲,在断开时也不会一下子断开,存在同样的问题,按键抖动信号波形如图 8-13 所示。

从波形图可以看出,按键开关在闭合及断开的瞬间,均伴随有一连串的抖动。抖动时间的长短由按键的机械特性决定,一般为 5ms~10ms,而按键的稳定闭合期的长短则是由操作人员的按键动作决定的,一般为十分之几秒的时间。

2. 按键的确认

按键的确认就是判别按键是否闭合,反映在电压上就是和按键相连的引脚呈现出高电平或低电平。如果高电平表示断开,那么低电平则表示闭合,所以通过检测电平的高低状态,便可确认按键是否按下。

3. 按键抖动的消除

因为机械开关存在抖动问题,为了确保 CPU 对一次按键动作只确认一次按键,必须

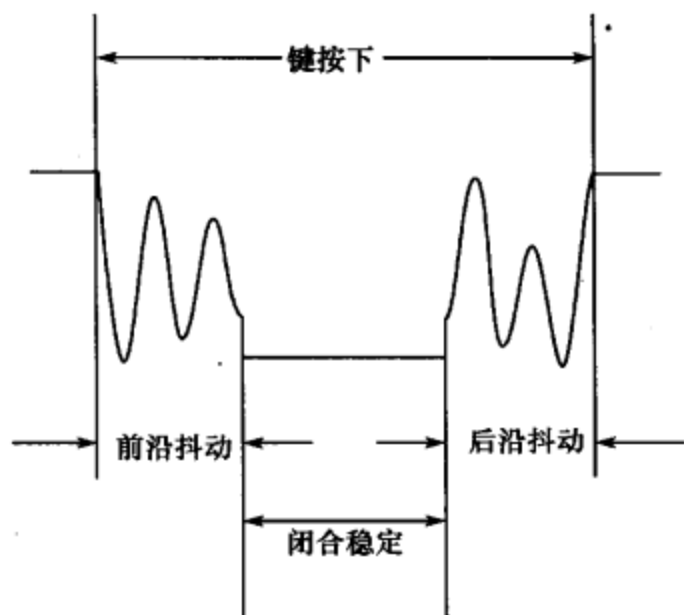


图 8-13 按键抖动信号波形

消除抖动的影响。消除按键的抖动,通常有硬件、软件两种消除方法。在键数较少时,可用硬件消除抖动,而当键数较多时,采用软件消除抖动。

(1) 硬件消除抖动

常用的硬件消除抖动有双稳态消除抖动和滤波消除抖动电路两种。

图 8-14 是双稳态消除抖动电路。图中用两个与非门构成一个基本 RS 触发器。当按键未按下时,因 $a=0, b=1$, 输出 Q 为 1; 当按键按下时, 因按键的机械性能, 使按键因弹性抖动而产生瞬时不闭合(抖动跳开 b), 当开关 K 没有稳定到达 b 时, 因与非门 2 输出为 0 反馈到与非门 1 的输入端, 封锁了与非门 1, 双稳态电路的状态不会改变, 输出保持为 1, 不会产生抖动的波形。当开关 K 稳定到达 b 时, 因 $a=1, b=0$, 从而使 $Q=0$, 状态产生翻转。当松开开关 K , 在开关未稳定达到 a 端时, 因 $Q=0$, 所以封锁了与非门 2, 从而消除了后沿的抖动, 使 $Q=0$ 不变。只有当开关稳定地达到 a 端后, 输出才重新返回到原状态。也就是说, 即使开关输出的电压波形是抖动的, 但经双稳态电路之后, 其输出为正规的矩形方波, 不会出现“毛刺”的现象。

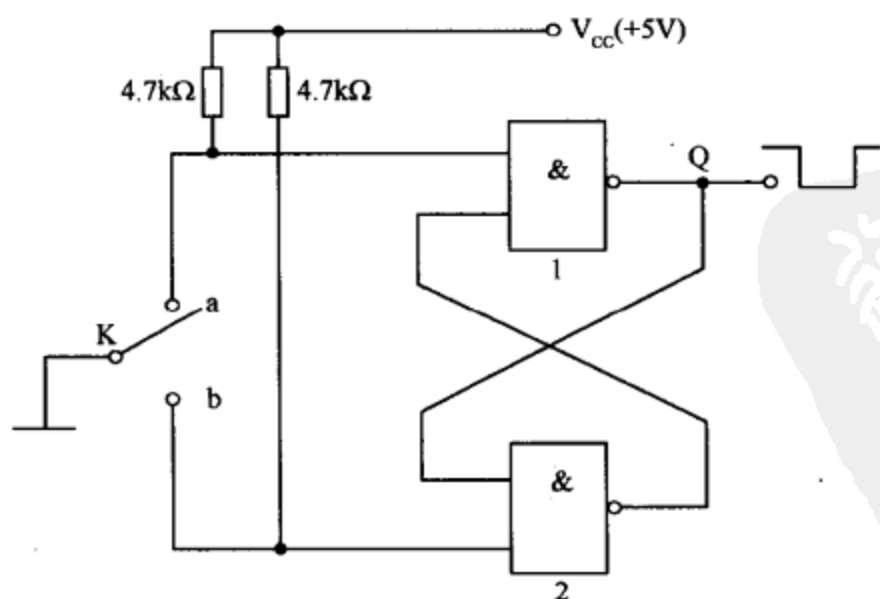


图 8-14 双稳态消除抖动电路

如果利用 RC 积分电路, 来吸收机械按键开关产生的干扰脉冲, 则这种电路称为滤波消除抖动电路, 如图 8-15 所示。当 K 未按下时, 电容两端电压为 0, 与非门输出为 1。当

K 按下时,由于 C 两端电压不能突变,即使在接触过程中出现抖动,只要适当选取 R1、R2 和 C 的值,使 C 两端的充电电压波动不超过门的开启电压(TTL 为 0.8V 左右),门的输出将不会改变,同样,K 在断开过程中,即使出现抖动,由于 C 两端电压不能突变,而要经过 R2 放电,只要 C 两端的放电电压波动不超过门的关闭电压,门的输出也不会改变。所以,从上面分析可看出,只要正确选取 R1、R2 和 C 的时间常数,就能保证 C 由稳态电压充电到开启电压或放电到关闭电压的延迟时间大于或等于 10ms,达到消除按键抖动的目的。

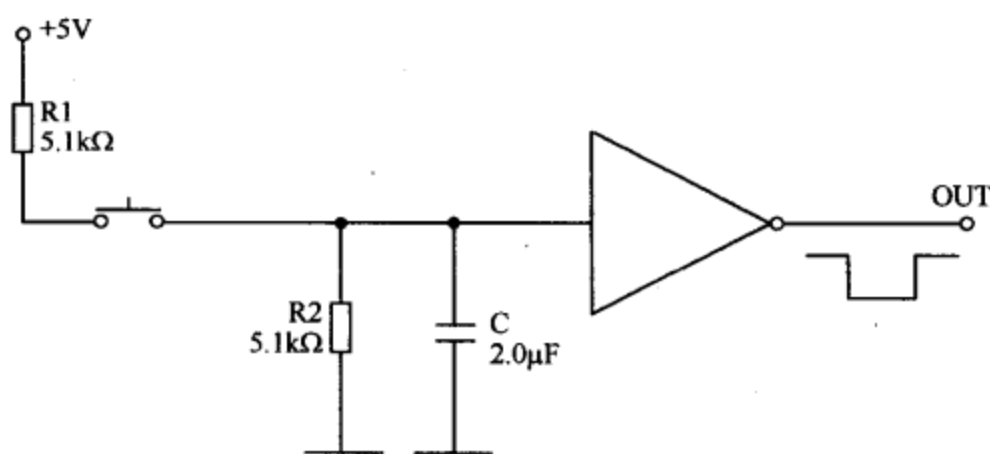


图 8-15 滤波消除抖动电路

(2) 软件消除抖动

单片机应用系统中,常用软件方法来消除抖动,即检测出键闭合后执行一个延时程序,产生 5ms~10ms 的延时,以避开按键按下去的抖动时间,待信号稳定之后再进行键查询,如果仍保持闭合状态电平,则确认为真正有键按下。一般情况下,不对按键释放的后沿进行处理。

二、键盘与单片机的连接

单片机中的键盘可分为独立式、编码式、串行口扩展式和矩阵式(行列式)等几类,下面分别进行介绍。

1. 独立式按键

独立式按键就是各按键相互独立、每个按键各接一根输入线,一根输入线上的按键是否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可以很容易判断哪个按键被按下了。独立式按键电路配置灵活,软件结构简单。但每个按键需占用一根输入口线,在按键数量较多时,输入口浪费大,电路结构显得很繁杂,故此种键盘适用于按键较少或操作速度较高的场合。下面介绍几种独立式按键的接口。

图 8-16 是直接与单片机的 I/O 接口线相接的独立式按键接口工作电路,通过读 I/O 接口,判定各 I/O 口线的电平状态,即可识别出按下的按键。

上述独立式按键电路中,各按键开关均采用了上拉电阻,这是为了保证在按键断开时,各 I/O 接口有确定的高电平,当然,如果输入口线内部已有上拉电阻,则外电路的上拉电阻可省去。下面采用查询式方式进行编程,方法是:先逐位查询每条 I/O 接口线的输入状态,如某一条 I/O 接口线输入为低电平,则可确认该 I/O 接口线所对应的按键已

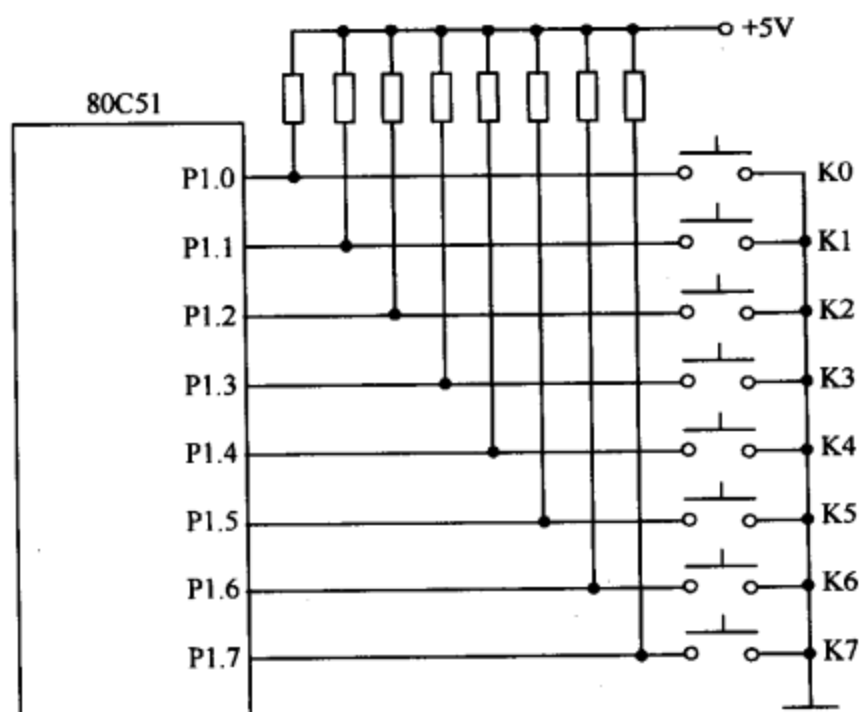


图 8-16 独立式按键

按下, 然后再转向该键的功能处理程序。查询程序如下(按键子程序 KEY0~KEY7 未给出):

KEY:	MOV P1, #0FFH	;把 P1 口置 1
	MOV A, P1	;读 P1 口各按键状态, 按键按下状态为 0, 未按为 1
	MOV R0, A	;保存键盘状态值
	ACALL DELAY	;调 10ms 延时子程序
	MOV A, P1	;再读 P1 口各按键状态
	CJNE A, R0, DODO	;两次结果不同, 说明是抖动引起
	CJNE A, #0FEH, K1	;K0 键未按下, 转 K1
	AJMP KEY0	;K0 键按下, 转 K0 键处理程序
K1:	CJNE A, #0FDH, K2	;K1 键未按下, 转 K2
	AJMP KEY1	;K1 键按下, 转 K1 键处理程序
K2:	CJNE A, #0FBH, K3	;K2 键未按下, 转 K3
	AJMP KEY2	;K2 键按下, 转 K2 键处理程序
K3:	CJNE A, #0F7H, K4	;K3 键未按下, 转 K4
	AJMP KEY3	;K3 键按下, 转 K3 键处理程序
K4:	CJNE A, #0EFH, K5	;K4 键未按下, 转 K5
	AJMP KEY4	;K4 键按下, 转 K4 键处理程序
K5:	CJNE A, #0DEH, K6	;K5 键未按下, 转 K6
	AJMP KEY5	;K5 键按下, 转 K5 键处理程序
K6:	CJNE A, #0BFH, K7	;K6 键未按下, 转 K7
	AJMP KEY6	;K6 键按下, 转 K6 键处理程序
K7:	CJNE A, #7FH, DODO	;K7 键未按下, 转抖动处理 DODO
	AJMP KEY7	;K7 键按下, 转 K7 键处理程序
	DODO: RET	;重键或无键按下, 返回

2. 编码式按键

采用编码式按键接口可提高单片机 I/O 接口线的利用率,当按键较多,I/O 接口线不够用时,可采用这种接口方式。图 8-17 是采用 74LS148 编码式按键接口电路。

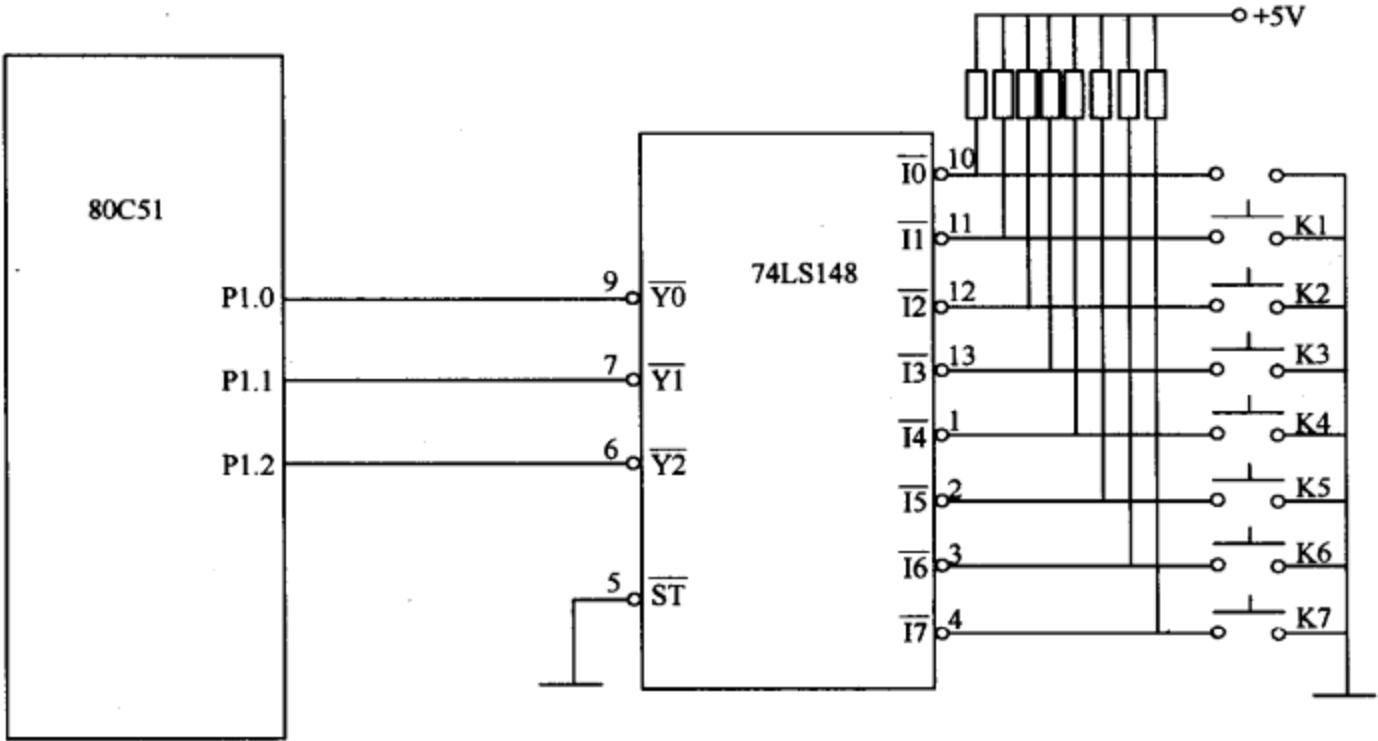


图 8-17 采用 74LS148 编码式按键接口电路

74LS148 是将 8 条数据线编码为 3 条数据线的 8—3 线优先编码器。它把占有 8 条线的按键状态变为用 3 位二进制方式进行传送。74LS148 的引脚功能如图 8-18 所示。

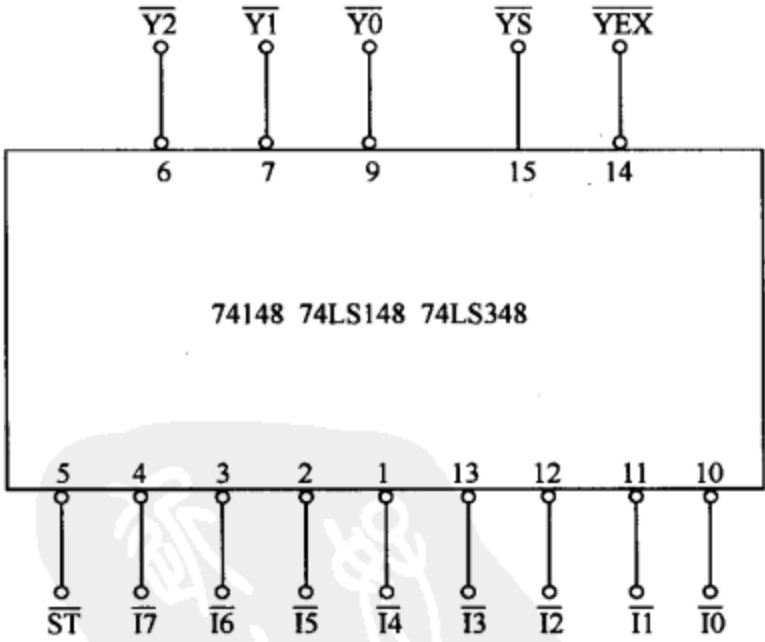


图 8-18 74LS148 的引脚功能图

74LS148 的 $\overline{I0} \sim \overline{I7}$ 为输入脚,低电平有效 $\overline{Y2}$ 、 $\overline{Y1}$ 、 $\overline{Y0}$ 为输出脚,共可输出 8 组二进制码,也为低电平有效。输入 $\overline{I0} \sim \overline{I7}$ 中,优先权排列顺序为: $\overline{I7}$ (最高), ..., $\overline{I0}$ (最低)。例如,当器件工作时,若 $\overline{I7}=0$ ($\overline{I7}$ 权限最高),则不论 $\overline{I6} \sim \overline{I0}$ 的状态如何,输出 $\overline{Y2Y1Y0}=000$; 当 $\overline{I7}=1$ 时,则观察 $\overline{I6}$ 是否为有效信号(即是否为“0”),若 $\overline{I6}=0$,那么输出 $\overline{Y2Y1Y0}=001$,而不论 $\overline{I5} \sim \overline{I0}$ 的状态如何……74LS148 编码器的真值表如表 8-4 所列。

表 8-4 74LS148 编码器真值表

输 入									输 出				
\overline{ST}	$\overline{I7}$	$\overline{I6}$	$\overline{I5}$	$\overline{I4}$	$\overline{I3}$	$\overline{I2}$	$\overline{I1}$	$\overline{I0}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$	\overline{YEX}	YS
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	×	×	×	×	×	×	×	0	0	0	0	1
0	1	0	×	×	×	×	×	×	0	0	1	0	1
0	1	1	0	×	×	×	×	×	0	1	0	0	1
0	1	1	1	0	×	×	×	×	0	1	1	0	1
0	1	1	1	1	0	×	×	×	1	0	0	0	1
0	1	1	1	1	1	0	×	×	1	0	1	0	1
0	1	1	1	1	1	1	0	×	1	1	0	0	1
0	1	1	1	1	1	1	1	0	1	1	1	0	1

74LS148 除了 8 根输入线和 3 根输出线外,还有 3 根信号线。其中 \overline{ST} 是输入允许控制端,称为输入使能端,当 $\overline{ST}=0$ 时编码器工作, $\overline{ST}=1$ 时编码器不工作,所有输出均为 1。 YS 及 \overline{YEX} 为输出信号,用以指示输入信号的情况,当器件不工作($\overline{ST}=1$)时, $YS=1$, $\overline{YEX}=1$;当器件工作($\overline{ST}=0$)时,若 $\overline{I0} \sim \overline{I7}$ 中有无信号输入,则 $YS=1$,否则 $YS=0$ 。

编码式按键的输入程序如下(按键子程序 KEY0~KEY7 未给出):

```

KEY: MOV P1, #0FFH      ;把 P1 口置 1
      MOV A, P1          ;读按键状态
      ANL A, #07H        ;将不代表键状态的各位屏蔽
      MOV R0, A          ;保存键盘状态值
      ACALL DELAY        ;调 10ms 延时子程序
      MOV A, P1          ;再读 P1 口各按键状态
      ANL A, #07H        ;将不代表键状态的各位屏蔽
      CJNE A, R0, DODO    ;两次结果不同,说明是抖动引起
      CJNE A, #06H, K2    ;K1 键未按下,转 K2
      AJMP KEY1           ;K1 键按下,转 K1 键处理程序
K2:   CJNE A, #05H, K3    ;K2 键未按下,转 K3
      AJMP KEY2           ;K2 键按下,转 K2 键处理程序
K3:   CJNE A, #04H, K4    ;K3 键未按下,转 K4
      AJMP KEY3           ;K3 键按下,转 K3 键处理程序
K4:   CJNE A, #03H, K5    ;K4 键未按下,转 K5
      AJMP KEY4           ;K4 键按下,转 K4 键处理程序
K5:   CJNE A, #02H, K6    ;K5 键未按下,转 K6
      AJMP KEY5           ;K5 键按下,转 K5 键处理程序
K6:   CJNE A, #01H, K7    ;K6 键未按下,转 K7
      AJMP KEY6           ;K6 键按下,转 K6 键处理程序

```

K7: CJNE A, #00H, DODO
AJMP KEY7
DODO:RET

;K7 键未按下,转抖动处理 DODO
;K7 键按下,转 K7 键处理程序
;重键或无键按下,返回

3. 串行口扩展按键

80C51 单片机应用系统中,如果并行 I/O 接口不够,而串行口又没有其他用处时,则可用来扩展并行 I/O 接口,从而节省了单片机的硬件资源。80C51 单片机内部的串行口在方式 0 工作状态下,使用移位寄存器芯片可以扩展一个或多个 8 位并行 I/O 接口,下面以 74LS165 为例,通过串行口来扩展键盘电路,有关电路如图 8-19 所示。

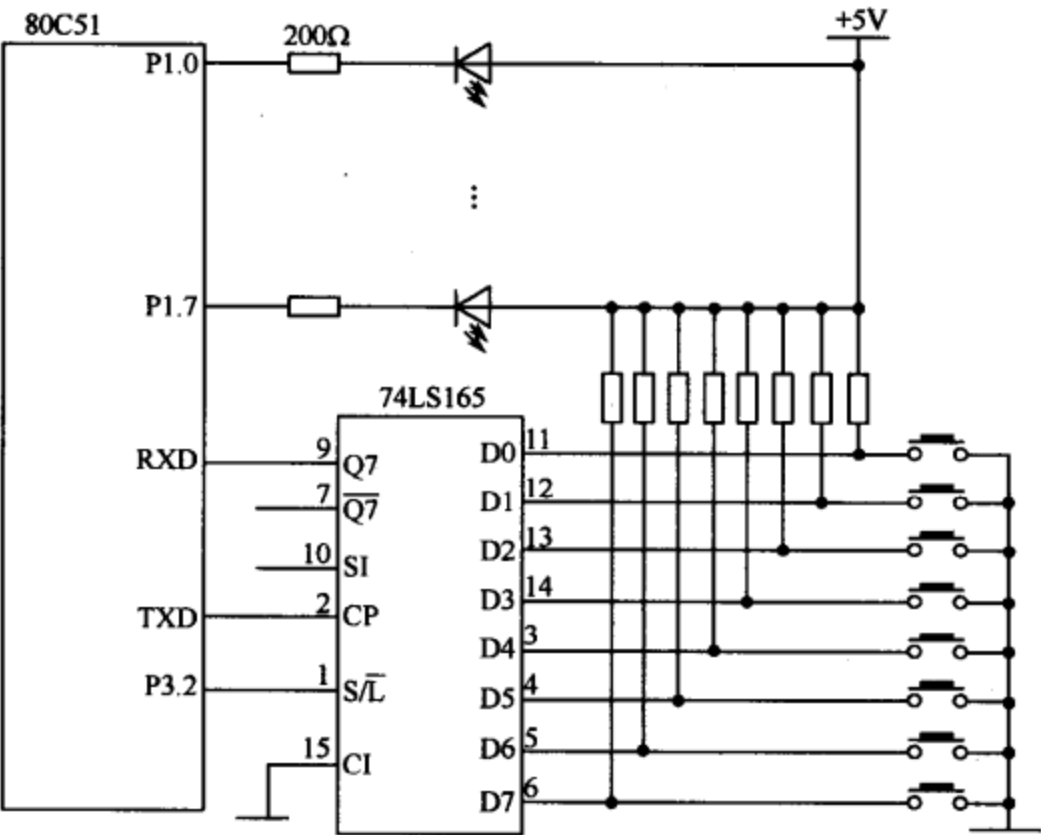


图 8-19 串行口扩展键盘电路

8 位单向移位寄存器 74LS165 为并行(串行)输入/串行输出寄存器,它能在一个信号的控制下并行置入一个字节的数据,然后在时钟脉冲的作用下逐位移出,也能使数据从另外一个引脚串行输入。74LS165 引脚功能如图8-20所示。

图 20 中,D0~D7 为一个字节的并和数据输入端。S/ \overline{L} (Shift/Load):控制信号输入端,该引脚为高电平时具有移位功能,为低电平时,将 D0~D7 端的数据输入到内部保存。CP 为时钟信号(即移位脉冲)输入端,当 S/ \overline{L} 端为高电平时,CP 端的每一次正跳变,都会使已收入内部的数据(D0~D7)从 Q7 端移出一位,移位的顺序是 D7 最先从 Q7 端移出,D0 最后从 Q7 端移出。CI(Clock inhibit)为时钟脉冲禁止端,当该引脚为高电平时,时钟信号(移位脉冲)不能进入,因而也就不可能移位,正常工作时必须接低电平。SI(Serial Input)为串行数据输入端,该器件也能从该端接收串行数据

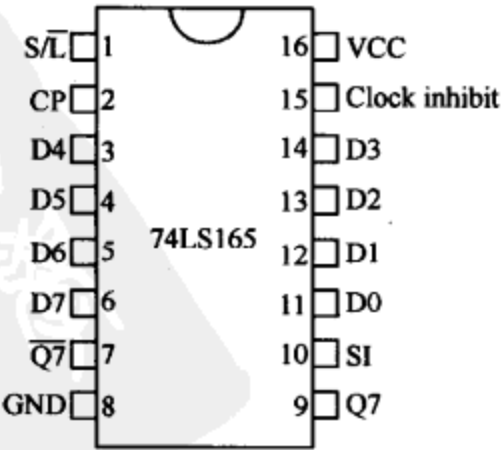


图 8-20 并行输入/串行输出移位寄存器 74LS165 管脚功能

并从 Q7 端移出。

8 位单向移位寄存器 74LS165 状态表如表 8-5 所列。

表 8-5 8 位单向移位寄存器 74LS165 状态表

S/L	CI	CP	SI	D0~D7	Q0 ⁿ⁺¹	Q1 ⁿ⁺¹	Q7 ⁿ⁺¹	说明
0	×	×	×	D0~D7	D0	D1	D7	并行输入
1	0	0	×	×	Q0 ⁿ	Q1 ⁿ	Q7 ⁿ	保持
1	0	↑	1	×	1	Q0 ⁿ	Q6 ⁿ	输入一个 1
1	0	↑	0	×	0	Q0 ⁿ	Q6 ⁿ	输入一个 0
1	1	×	×	×	Q0 ⁿ	Q1 ⁿ	Q7 ⁿ	保持

图 8-19 所示的电路中, TXD(P3. 1)为移位脉冲输出端, 与 74LS165 的移位脉冲输入端 CP 相连; RXD(P3. 0)作为串行输入端, 与 74LS165 的串行输出端 Q7 相连; P3. 2 用来控制 74LS165 的移位与置入。

80C51 的 P1 口连接了 8 个 LED 发光二极管, 以下程序可实现用按键控制发光二极管的亮与灭。

```

MAIN: MOV SCON, #10H      ;设置串行口为工作方式 0, REN=1, 允许接收
      CLR P3. 2            ;允许键盘输入
      SETB P3. 2           ;串行口开始接收数据
      CLR RI               ;清除串行口接收中断标志
STO:  JNB RI, STO          ;等待接收完毕
      MOV A, SBUF          ;将接收缓冲器中的数据送 A
      MOV P1, A            ;将 A 中的数据送 P1 口
      AJMP MAIN            ;程序循环执行
  
```

4. 矩阵式(行列式)按键

为了减少键盘与单片机接口时所占用 I/O 线的数目, 在按键数较多时, 通常将键盘排列成行列矩阵形式, 如图 8-21 所示。

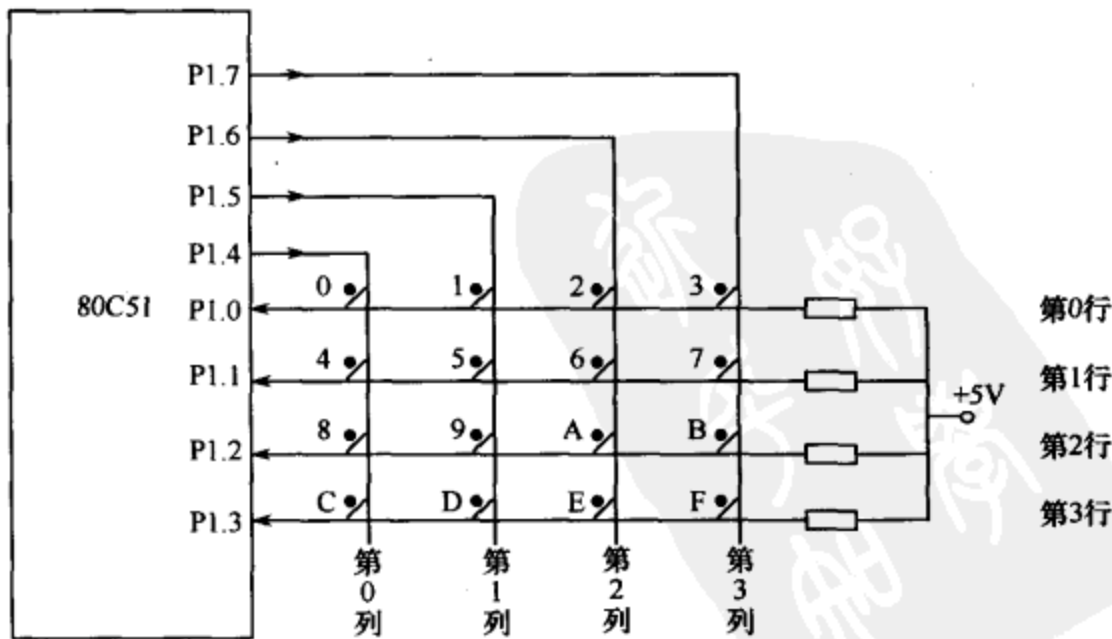


图 8-21 矩阵式键盘电路

键盘的行线一端经电阻接+5V电源,另一端接单片机的输入口;各列线的一端接单片机的输出口,另一端悬空。每一行线(水平线)与列线(垂直线)的交叉处不相通,而是通过一个按键来连通。利用这种矩阵结构只需 n 条行线和 m 条列线,即可组成具有 $n \times m$ 个按键的键盘。由此可见,在需要的键位数比较多时,一般需要采用矩阵式键盘。

(1)判断有无键被按下。键被按下时,与此键相连的行线与列线将导通,而行线电平在无键按下时处于高电平,显然,如果让所有列线处于高电平,那么键按下与否不会引起行线电平的状态变化,所以只有让所有列线处于低电平,当有键按下时按键所在行电平将被拉成低电平,根据此行电平的变化,便能判定此行一定有键被按下。

矩阵式结构的键盘比独立连接式要复杂一些,判断有无键被按下也要复杂一些。图 8-21 中,行线通过电阻接正电源,并将行线所接的单片机的 I/O 接口作为输入端;而列线所接的 I/O 接口则作为输出端。为了判断有无键被按下,可先经输出口向所有列线输出低电平,然后再输入各行线状态。若行线状态皆为高电平,则表明无键按下;若行线状态中有低电平,则表明有键被按下。

(2)判断按键是否真的被按下。当判断出有键被按下之后,用软件延时的方法延时 10ms,再判断键盘的状态,如果仍为有键被按下,则认为确实有键按下,否则,当做键抖动处理。

(3)判断哪一个按键被按下。下面以图 8-21 中 7 号键被按下为例,来说明此键是如何被识别出来的。

让所有列线处于低电平,只能得出某行有键被按下。例如,若所有列线均为低电平,当按键 4、5、6、7 任一按键按下后,均会造成第 1 行为低电平;为了进一步地判定到底是哪一列的键被按下,可在某一时刻只让一条列线处于低电平,而其余所有列线处于高电平。

当第 0 列为低电平,其余各列为高电平时,因为是按键 7 被按下,所以,按键 7 所在的第 1 行仍处于高电平状态;

当第 1 列为低电平,其余各列为高电平时,按键 7 所在的第 1 行仍处于高电平状态。

当第 2 列为低电平,其余各列为高电平时,同样会发现按键 7 所在的第 1 行仍处于高电平状态。

当第 3 列为低电平,其余各列为高电平时,因为按键 7 被按下,所以第 1 行的电平将由高电平转换为低电平,据此,可以确信第 1 行第 3 列交叉点处的按键即 7 号键被按下。

(4)计算键码。键号是按从左到右、从上到下的顺序编排,按这种编排规律,各行的首键号依次是 00H、04H、08、0CH;如列线按 0~3 的顺序进行编号,则键码的计算公式为

$$\text{键码} = \text{行首键号} + \text{列号}$$

(5)等待键释放。键释放之后,可以根据键码转相应的键处理子程序,进行数据的输入或命令的处理。

根据以上分析,编写的 4 行 \times 4 列按键程序如下:

;以下是判断有无按键按下

KEY: MOV R5, #00H	;用于存放键码,初值为 0(定义为无键按下)
MOV P1, #0FH	;列扫描线输出 0,行扫描线置 1
MOV A, P1	;读键状态
ANL A, #0FH	;屏蔽高 4 位(列线),检测行线状态

CJNE A, #0FH, PD1	;判断有键按下否,有则转到 PD1 处理程序
AJMP KEY	;重新扫描
;以下是判断按键是否被真的按下处理程序	
PD1: ACALL DELAY	;延时 10ms,以消除按键抖动和排除干扰
MOV A, P1	;再读键状态
ANL A, #0FH	;列扫描线输出 0,行扫描线置 1
CJNE A, #0FH, PD2	;确认按键是否按下,有则转到 PD2 处理程序
AJMP KEY	;无键按下,重新扫描
;以下是哪一个按键被按下处理程序	
PD2: MOV R2, #0EFH	;扫描初值送 R2
MOV R4, #00H	;扫描次数初值送 R4
MOV A, R2	;被扫描的列线送 0
KPD: MOV P1, A	;将 A 中的值(初值为 1110 1111B)输出到 P1
MOV A, P1	;读键状态
JB ACC. 0, HONE	;ACC. 0=1,第 0 行键无键被按下,转 HONE 处理程序
MOV A, #00H	;装 0 行行首键号
AJMP LKP	;跳转到计算键值程序
HONE: JB ACC. 1, HTWO	;ACC. 1=1,第 1 行键无键被按下,转 HTWO 处理程序
MOV A, #04H	;装 1 行行首键号
AJMP LKP	;跳转到计算键值程序
HTWO: JB ACC. 2, HTHR	;ACC. 2=1,第 2 行键无键被按下,转 HTHR 处理程序
MOV A, #08H	;装 2 行行首键号
AJMP LKP	;跳转到计算键值程序
HTHR: JB ACC. 3, NEXT	;ACC. 3=1,第 3 行键无键被按下,转下一列 NEXT
MOV A, #0CH	;装 3 行行首键号
AJMP LKP	;跳转到计算键值程序
NEXT: INC R4	;扫描列号加 1(扫描下一列)
MOV A, R2	;将扫描初值 0EFH 送 R2
RL A	;循环左移一位, A 的值为 0DFH
MOV R2, A	;将 A 的值存于 R2 中,以便进行下一列的扫描
CJNE R4, #04H, KPD	;若列未扫描 4 次,则继续扫描
AJMP KEY	;扫描完毕,开始新一轮扫描
;以下为计算键码程序	
LKP: ADD A, R4	;行首键号加列号是被按下的键号
MOV R5, A	;被按下的键号存放在 R5
AJMP KEY	;开始新一轮扫描,若为键盘子程序,该行为 RET

;以下是 10ms(采用 12MHz 晶振)延时子程序

DELAY: MOV R7, #50	;1 个机器周期
D2: MOV R6, #100	;1 个机器周期
D1: DJNZ R6, D1	;2 个机器周期
DJNZ R7, D2	;2 个机器周期
RET	;2 个机器周期

三、键盘的工作方式

键盘的工作方式有 3 种,即程序控制扫描、定时扫描和中断扫描方式。

1. 程序控制扫描方式

程序控制扫描方式是指单片机在空闲时,才调用键盘扫描子程序,并反复地扫描键盘,直到用户从键盘上输入命令或数据,而在执行键入命令或处理键入数据过程中,CPU 将不再响应键入要求,直到 CPU 重新扫描键盘为止。主要包含以下内容:

(1)判别有无键按下。

(2)键扫描取得闭合键的行、列值。

(3)用算法得到键值。

(4)判断闭合键是否释放,如果没有释放则等待;如果闭合键释放,将闭合键的键号保存,并转去执行该闭合键的功能。

(6)返回。

前面介绍的几种按键程序均采用了这种扫描方式。

2. 定时扫描方式

定时扫描方式就是每隔一定时间对键盘扫描一次,它利用单片机内部的定时器产生一定时间(例如 10ms)的定时,当定时时间到就产生定时器溢出中断,CPU 响应中断后对键盘进行扫描,并在有键按下时识别出该键执行响应的键功能程序。

3. 中断扫描方式

键盘工作在程序控制扫描方式时,当无键按下时,CPU 要不间断地扫描键盘,直到有键按下为止。如果 CPU 要处理的事情很多,这种工作方式将不能适应。定时扫描方式只要定时时间到,CPU 就去扫描键盘,工作效率有了进一步的提高。由此可见,这两种方式常使 CPU 处于空扫状态,而中断扫描方式下,CPU 可以一直处理自己的工作,直到有键闭合时发出中断申请,CPU 响应中断,执行相应的中断服务程序,才对键盘进行处理,从而提高了 CPU 的工作效率。图 8-22 为中断方式键盘接口电路。

该键盘由单片机的 P1 口构成 4×4 矩阵键盘。键盘的列线和 P1 口的高 4 位 P1.4~P1.7 相接,键盘的行线和 P1 口的低 4 位 P1.0~P1.3 相接。P1.4~P1.7 经过与门和单片机的外中断 $\overline{\text{INT0}}$ 相接。当有键按下时, $\overline{\text{INT0}}$ 变为低电平,向 CPU 发出中断申请,若 CPU 开放外部中断 0,则 CPU 响应外部中断请求,执行相应的中断服务程序。为了在中断服务中不因按键的识别错误再引起中断,因此,在中断服务程序中,首先应关闭中断,再进行键扫描、消除抖动、识别按键错误等工作,具体处理方法同程序控制扫描方式。

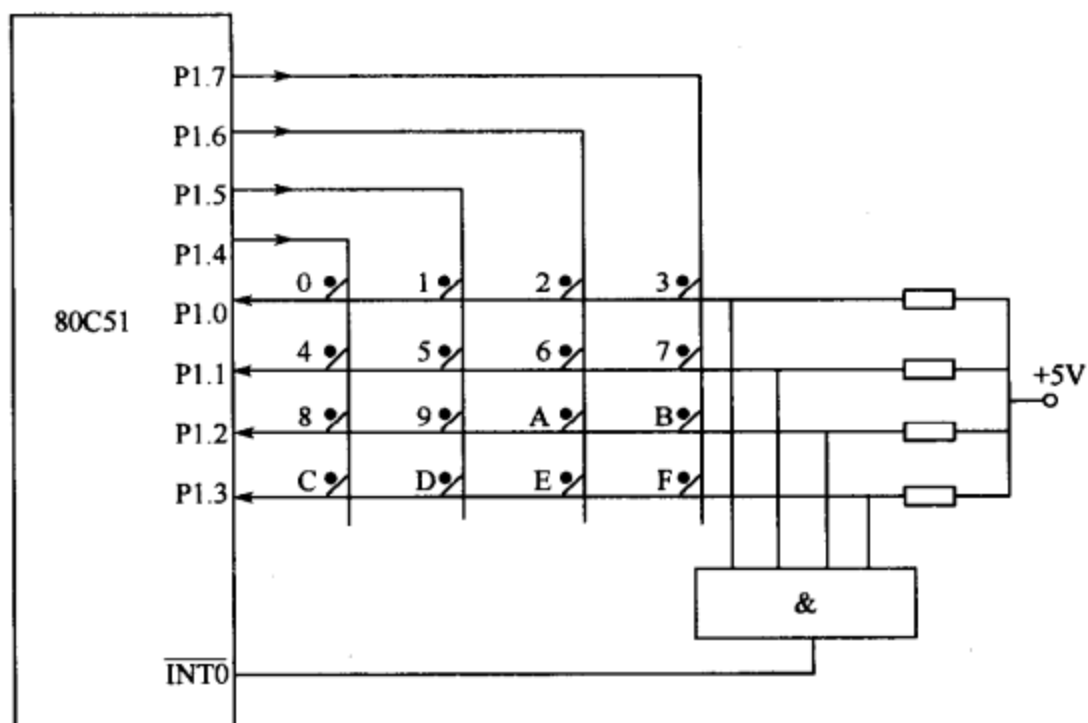


图 8-22 中断扫描方式

四、键盘接口实验

实验 4 用 AT89C51 实验开发板做以下实验：每按一次 K1 键(P3. 6),P1 口外接的灯向左被轮流点亮 1 位,每按 K2 键(P3. 7),P1 口外接的灯向右被轮流点亮 1 位。有关电路如图 8-23 所示。

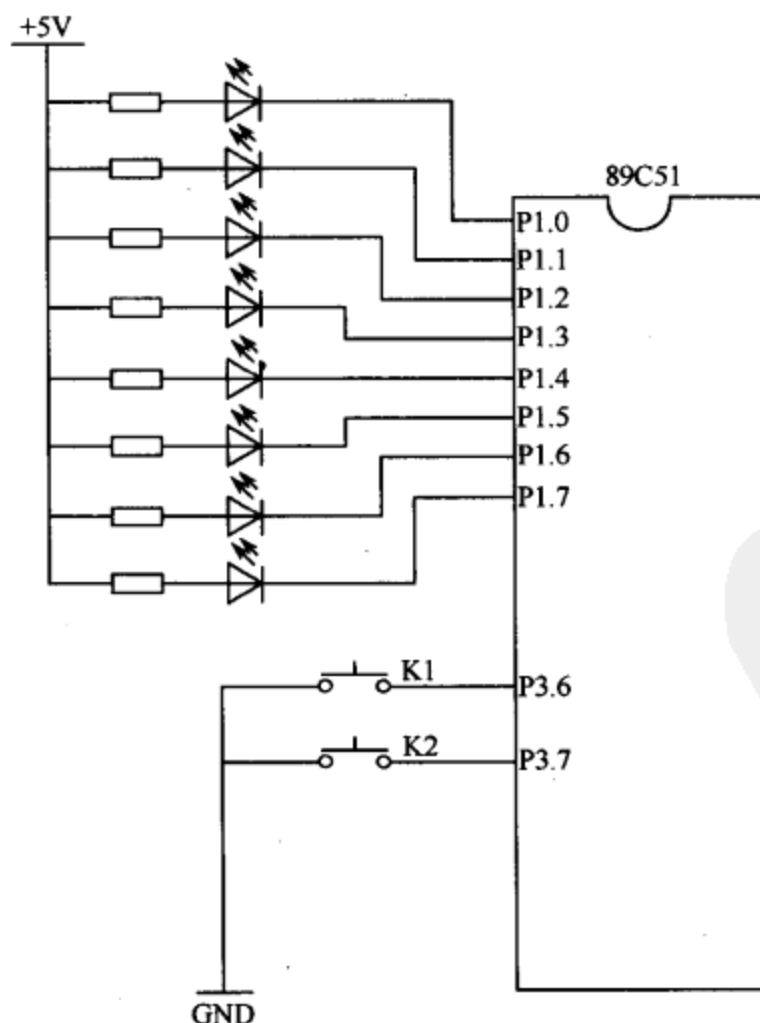


图 8-23 用按键控制流水灯

实验的源程序如下:

```
        LEFT BIT 00H           ;设置流水灯动标志
        RIGHT BIT 01H          ;设置停止标志
        ORG 0000H
        AJMP MAIN
        ORG 0100H

;以下是主程序
MAIN:    MOV SP, #5FH
        MOV P1, #0FFH          ;P1 口外接的 8 只灯全灭
        MOV P3, #0FFH          ;P3 口置 1
        MOV R1, #0FEH          ;将流水灯的流动码送 R1
LOOP:    ACALL KEY              ;调用判断按键子程序
        JNB F0, LOOP            ;F0=0 说明无键按下,转 LOOP,继续调用
        ACALL PROC              ;F0=1 说明有键按下,转 PROC 键盘处理子程序
        ACALL LAMP_L            ;调流水灯子程序
        AJMP LOOP

;以下是判断按键子程序
KEY:     CLR F0                 ;清 F0,表示无键按下
        MOV A, P3               ;取 P3 的值
        ANL A, #11000000B       ;屏蔽 A 的低 6 位
        CJNE A, #11000000B, KEY1;有键按下,转 KEY1 键处理程序
        AJMP KEY_RET            ;无键按下,转 KEY_RET 处理程序
KEY1:    ACALL D10MS             ;调 10ms 延时子程序,转抖动处理程序
        MOV A, P3               ;读取 P3 的值
        ANL A, #11000000B       ;屏蔽 A 的低 6 位
        CJNE A, #11000000B, KEY2;确实有键按下,转 KEY2 键处理程序
        AJMP KEY_RET            ;键抖动引起,转 KEY_RET 处理程序
KEY2:    MOV R0, A               ;确实有键按下,将键值存入 R0 中
        SETB F0                 ;设置有键按下的标志
KEY3:    MOV A, P3               ;继续读取 P3 的值
        ANL A, #11000000B       ;屏蔽 A 的低 6 位
        CJNE A, #11000000B, KEY3;键未释放,转 KEY3 键处理程序,继续等待
KEY_RET: RET                     ;按键释放,程序返回

;以下是键盘处理子程序
PROC:    MOV A, R0               ;从 R0 寄存器中获取键值
        JB ACC. 6, PROC1         ;若 ACC. 6 = 1,说明按键 K1 未被按下,转
                                   PROC1 处理程序
        SETB LEFT                ;K1 被按下,设置 LEFT=1,流水灯左移
        CLR RIGHT                ;设置 RIGHT=0
```


PROC1: JB ACC. 7, PROC_RET	; 若 ACC. 7=1, 说明按键 K2 未被按下, 转 PROC_RET 处理程序
SETB RIGHT	; K2 被按下, 设置 RIGHT=1, 流水灯右移
CLR LEFT	; 设置 LEFT=0
PROC_RET: RET	; 程序返回
; 以下是流水灯子程序	
LAMP_L: JNB LEFT, LAMP_R	; 如果 LEFT=0, 转 LAMP_R 处理程序
MOV A, R1	; 从 R1 中取出流水灯的流动码送 A
RL A	; 左移位
MOV R1, A	; 将 A 中的数存于 R1 中
MOV P1, A	; 输出到 P1 口
AJMP LAMP_RET	; 调 LAMP_RET, 返回到主程序
LAMP_R: JNB RIGHT, LAMP_RET	; 如果 RIGHT=0, 转 LAMP_RET 处理程序, 程序返回
MOV A, R1	; 从 R1 中取出流水灯的流动码送 A
RR A	; 右移位
MOV R1, A	; 将 A 中的数存于 R1 中
MOV P1, A	; 输出到 P1 口
LAMP_RET: RET	; 程序返回
; 以下是 10ms 延时子程序	
D10MS: MOV R5, #50	
D2: MOV R4, #100	
D1: DJNZ R4, D1	
DJNZ R5, D2	
RET	
END	

实验步骤如下:

(1) 打开 Keil 软件, 输入上面的程序, 保存为 key_1.asm。对程序进行编译、链接和调试, 产生 key_1.hex 目标文件。

(2) 先可用硬件仿真器对源程序进行仿真, 然后用 RF-810 编程器对 AT89C51 芯片编程。

(3) 将 AT89C51 芯片插入实验开发板, 通电进行实验。

该实验程序在本书所附光盘的 example\ch_8\key_1 文件夹中。

第三节 LCD 显示器接口

液晶显示器由于体积小、质量轻、功耗低等优点, 应用十分广泛, 如电子表、传真器、复印机以及电脑中的液晶显示器, 都使用了 LCD 显示器。

从液晶显示器的显示内容来分, 可分为段式、字符式和点阵式 3 种。其中字符式液晶

显示器以价廉、显示内容丰富、美观、无需定制、使用方便等特点,成为 LED 显示器的理想替代品。一般初学者由字符型 LCD 显示器入手比较简单,学完之后,再进一步控制图案型 LCD 模块就十分方便和顺手。

一、字符型液晶显示器概述

字符型 LCD 专门用于显示数字、字母、图形符号及少量自定义符号。这类显示器均把液晶显示控制器、驱动器、字符存储器等做在一块板上,再与液晶屏(LCD)一起组成一个显示模块,称为 LCM,因此,这类显示器安装与使用都十分简单和方便。

字符型 LCD 是由若干个 5×7 或 5×11 等点阵字符位组成。每一个点阵字符位都可以显示一个字符。点阵字符位之间有一空点距的间隔起到了字符间距和行距的作用。目前市面上常用的有 16 字×1 行、16 字×2 行、20 字×2 行和 40 字×2 行等的字符模块组。这些 LCM 虽然显示字数各不相同,但输入/输出接口都相同。

图 8-24 是 16 字×2 行液晶显示模块的外形,其接口引脚有 16 只,管脚功能如表8-6 所列。

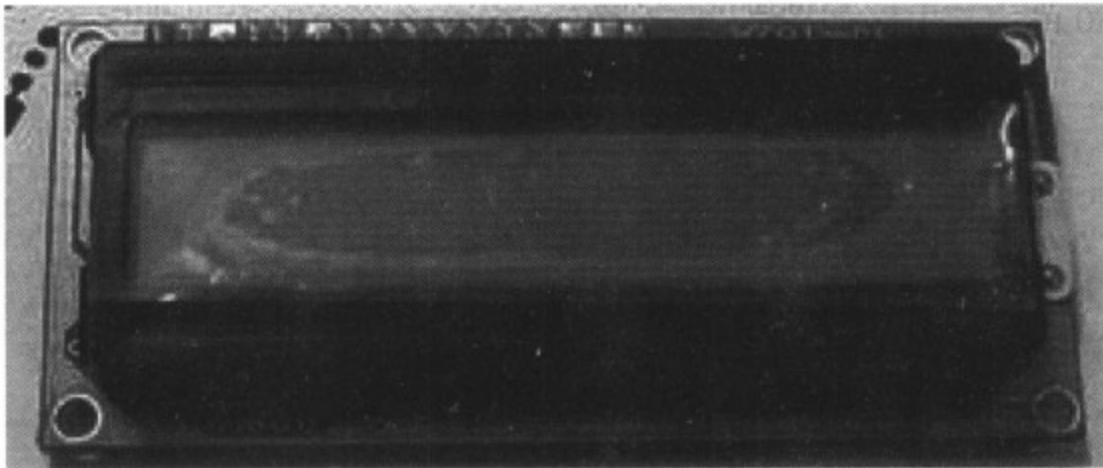


图 8-24 16 字×2 行液晶显示模块外形

表 8-6 字符型液晶显示接口功能

编号	符号	功能	编号	符号	功能
1	V _{SS}	电源地	9	D2	数据 2
2	V _{DD}	电源正极	10	D3	数据 3
3	VLCD	液晶显示对比度调整	11	D4	数据 4
4	RS	数据/命令选择	12	D5	数据 5
5	R/W	读/写选择	13	D6	数据 6
6	E	使能信号	14	D7	数据 7
7	D0	数据 0	15	BLA	背光源正极
8	D1	数据 1	16	BLK	背光源负极

表 8-6 中:V_{SS}为电源地;V_{DD}接 5V 正电源;VLCD 为液晶显示器对比度调整端,接正电源时对比度最弱,接地时对比度最高,对比度过高时会产生“鬼影”,使用时可以通过一个 10kΩ 的电位器调整对比度;RS 为寄存器选择,高电平时选择数据寄存器,低电平时选择指令寄存器;R/W 为读/写信号线,高电平时进行读操作,低电平时进行写操作;E 端为使能端,当 E 端由高电平跳变成低电平时,液晶模块执行命令;D0~D7 为 8 位双向数据

线;BLA、BLK 用于带背光的模块,不带背光的模块这两个管脚悬空不接。

二、字符显示模块内部结构

目前大多数字符显示模块的控制器都采用型号为 HD44780 的集成电路作控制器。HD44780 是集控制器、驱动器于一体,专用于字符显示控制驱动集成电路。HD44780 是字符型液晶显示控制器的代表电路。

1. HD44780 集成电路的特点

(1)HD44780 不仅作为控制器,而且还具有驱动 40×16 点阵液晶像素的能力,并且其驱动能力可通过外接驱动器扩展为 360 列驱动。

(2)HD44780 的显示缓冲区及用户自定义的字符发生器 CGRAM 全部内藏在芯片内。

(3)HD44780 接口数据传输可为 8 位数据和 4 位数据传输两种方式。

(4)HD44780 具有简单而功能较强的指令集,可实现字符移动、闪烁等显示功能。

2. HD44780 的工作原理

HD44780 的内部电路如图 8-25 所示。

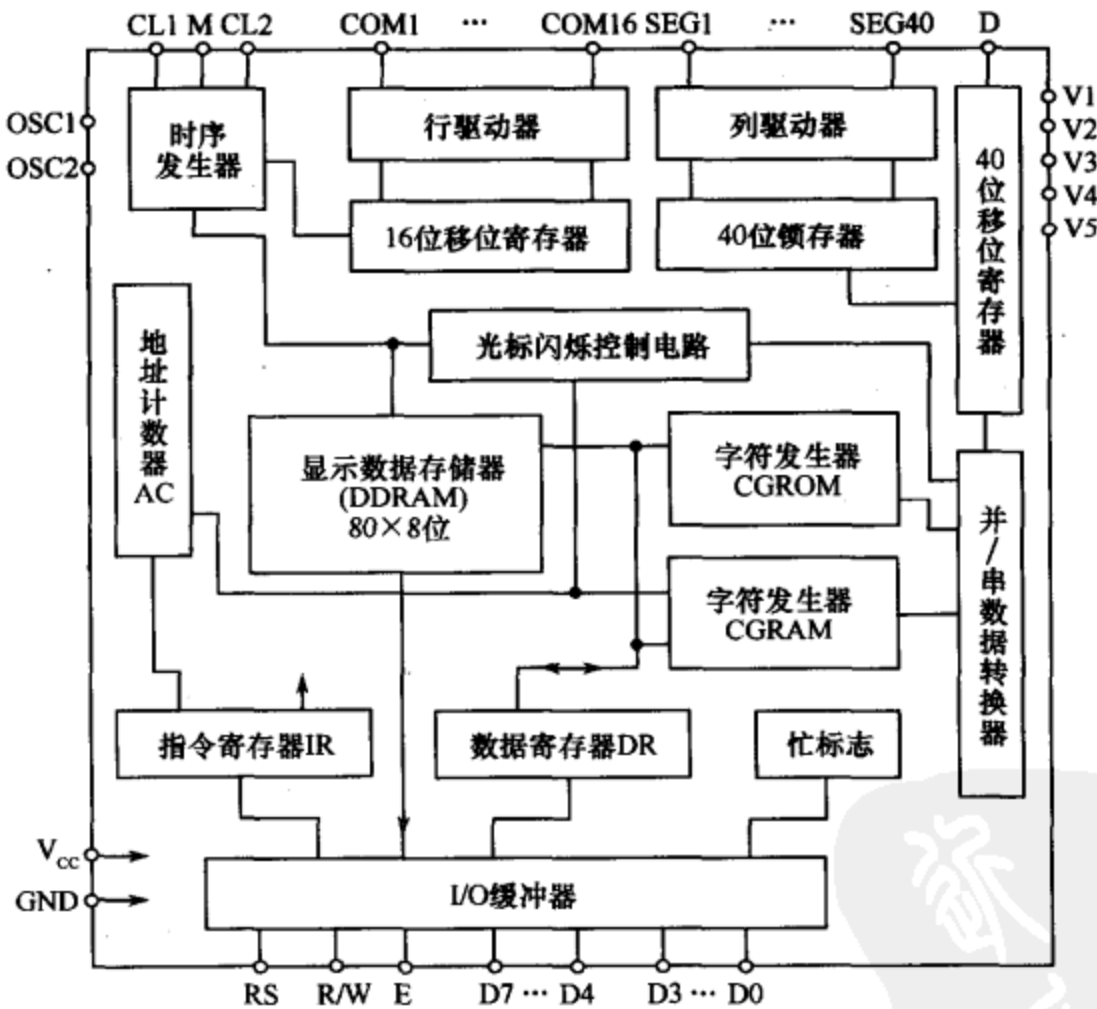


图 8-25 HD44780 的内部电路

(1)显示数据存储器 DDRAM。DDRAM 用来存放要 LCD 显示的数据,只要将标准的 ASCII 码送入 DDRAM,内部控制电路会自动将数据传送到显示器上,例如要 LCD 显示字符 A,则只须将 ASCII 码 41H 存入 DDRAM 即可。DDRAM 有 80B 空间,共可显示 80 个字(每个字为 1B),其存储器地址和实际显示位置的排列顺序的对应关系如图 8-26 所示。

	0	1	2	3	...	12	13	14	15	-->显示位置	
第1行-->	00	01	02	03	...	0C	0D	0E	0F	-->DDRAM地址	
(a)											
	0	1	2	3	...	15	16	17	18	19	-->显示位置
第1行-->	00	01	02	03	...	0F	10	11	12	13	-->DDRAM地址
第2行-->	40	41	42	43	...	4F	50	51	52	53	-->DDRAM地址
(b)											
	0	1	2	3	...	15	16	17	18	19	-->显示位置
第1行-->	00	01	02	03	...	0F	10	11	12	13	-->DDRAM地址
第2行-->	40	41	42	43	...	4F	50	51	52	53	-->DDRAM地址
第3行-->	14	15	16	17	...	23	23	25	26	27	-->DDRAM地址
第4行-->	54	55	56	57	...	63	64	65	66	67	-->DDRAM地址
(c)											

图 8-26 存储器地址与实际显示位置的排列顺序对应关系

(a)16 字×1 行; (b)20 字×2 行; (c)20 字×4 行。

图 8-26(a)为 16 字×1 行的 LCM,它的地址为 00H~0FH;图(b)为 20 字×2 行的 LCM,第 1 行的地址为 00H~13H,第 2 行的地址为 40H~53H;图(c)为 20 字×4 行的 LCM,第 1 行的地址为 00H~13H,第 2 行的地址为 40H~53H,第 3 行的地址为 14H~27H,第 4 行的地址为 54H~67H。

(2)字符发生器 CGROM。HD44780 内藏的字符发生存储器(CGROM),存储了 160 个不同的点阵字符图形,如表 8-7 所列,这些字符有阿拉伯数字、英文字母的大小写、常用的符号和日文假名等,每一个字符都有一个固定的代码。例如,字符码 41H 为 A 字符,要在 LCD 中显示 A,就是将 A 的代码 41H 写入 DDRAM 中,同时电路到 CGROM 中将 A 的字型点阵数据找出来,显示在 LCD 上,即能看到字母 A。

(3)字符发生器 CGRAM。CGRAM 是供使用者储存自行设计的特殊造型的造型码 RAM,CGRAM 共有 512 位(64B)。一个 5×7 点矩阵字型占用 8×8 位,所以 CGRAM 最多可存 8 个造型。

(4)指令寄存器 IR。IR 寄存器负责储存单片机要写给 LCM 的指令码。当单片机要发送一个命令到 IR 寄存器时,必须要控制 LCM 的 RS、R/W 及 E 这 3 个引脚,当 RS 及 R/W 引脚信号为 0,E 引脚信号由 1 变为 0 时,就会把在 D0~D7 引脚上的数据送入 IR 寄存器。

(5)数据寄存器 DR。DR 寄存器负责储存单片机要写到 CGRAM 或 DDRAM 的数据,或储存单片机要从 CGRAM 或 DDRAM 读出的数据,因此 DR 寄存器可视为一个数据缓冲区,它也是由 LCM 的 RS、R/W 及 E 3 个引脚来控制。当 RS 及 R/W 引脚信号为 1,E 脚信号为 1 时,LCM 会将 DR 寄存器内的数据由 D0~D7 输出,以供单片机读取;当 RS 脚信号为 1,R/W 接脚信号为 0,E 脚信号由 1 变为 0 时,就会把在 D0~D7 引脚上的数据存入 DR 寄存器。

(6)忙碌标志信号 BF。BF 的功能是通知单片机,LCM 内部是否正忙着处理数据。当 BF=1 时,表示 LCM 内部正在处理数据,不能接受单片机送来的指令或数据。LCM

表 8-7 内藏字符发生器存储的字符

高4位 低4位	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	1	P	`	P				-	タ	ミ	α	ρ
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	△	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	×	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	ω
xxxx0100	(5)		\$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	ε	Ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ア	キ	ヌ	ラ	g	π
xxxx1000	(1)		(8	H	X	h	x			ィ	ク	ネ	リ	♪	Σ
xxxx1001	(2))	9	I	Y	i	y			ッ	ケ	ノ	ル	・	Y
xxxx1010	(3)		*	:	J	Z	j	z			エ	コ	ハ	レ	j	キ
xxxx1011	(4)		+	;	K	[k	<			オ	サ	ヒ	ロ	×	万
xxxx1100	(5)		,	<	L	¥	l				ヤ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M]	m	}			ユ	ズ	ハ	ン	も	÷
xxxx1110	(7)		.	>	N	^	n	→			ヨ	セ	ホ	◇	ん	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	□	ö	

设置 BF 的原因因为单片机处理一个指令的时间很短,只需几微秒左右,而 LCM 得花上 $40\mu\text{s}\sim 1.64\text{ms}$ 的时间,所以单片要机要写数据或指令到 LCM 之前,必须先查看 BF 是否为 0。

(7)地址计数器 AC。AC 的工作是负责计数写到 CGRAM、DDRAM 数据的地址,或从 DDRAM、CGRAM 读出数据的地址。使用地址设定指令写到 IR 寄存器后,则地址数据会经过指令解码器,再存入 AC。当单片机从 DDRAM 或 CGRAM 存取资料时,AC 依照单片机对 LCM 的操作而自动的修改它的地址计数值。

方法技巧 HD44780 的特点是:读写操作由使能信号 E 完成,不操作时为低电平,操作时产生一个正脉冲。在读信号且 E 为高电平时,控制器将所需的数据送入数据总线上,供单片机读取;在写操作时,E 信号的下降沿处将数据总线上的数据写入控制器接口

部的寄存器内。HD44780 对读/写操作的识别是判断 R/W 信号端上的电平状态,R/W=1 为读操作选择,R/W=0 为写操作选择。RS 信号是 HD44780 识别数据总线上的数据是属于指令代码还是属于显示数据。RS=0,选通指令寄存器通道,数据总线传输的是指令代码或标志位;RS=1,选通数据寄存器通道,数据总线传输的是显示数据或自定义字符的字模数据。接口部信号端的逻辑功能组合表如表 8-8 所列。

表 8-8 HD44780 接口部信号逻辑功能组合表

RS	R/W	E	D7~D0	功能
0	0	下降沿	输入态	写指令代码
0	1	高电平	输出态	读忙碌标志和 AC 码
1	0	下降沿	输入态	写数据
1	1	高电平	输出态	读数据

三、字符型液晶控制器的指令

用单片机来控制 LCM 模块,方式十分简单,LCM 模块其内部可以看成两组寄存器,一个为指令寄存器 IR,一个为数据寄存器 DR,由 RS 引脚来控制。所有对指令寄存器或数据寄存器的存取均需检查 LCM 内部的忙碌标志 BF,此标志用来告知 LCM 内部正在工作,并不允许接收任何的控制命令。而此位的检查可以令 RS=0,用读取 D7 来加以判断,当 D7 为 0 时,才可以写入指令或数据寄存器。LCM 控制指令共有 11 组,以下分别介绍。

1. 清屏

清屏指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

指令代码为 01H,将 DDRAM 数据全部填入“空白”的 ASCII 代码 20H,执行此指令将清除显示器的内容,同时光标移到左上角。

2. 光标归位

光标归位指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	×

指令代码为 02H,地址计数器 AC 被清 0,DDRAM 数据不变,光标移到左上角。×表示可以为 0 或 1。

3. 输入方式设置

输入方式设置指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

该指令用来设置光标、字符移动的方式。具体情况如表 8-9 所列。

表 8-9 光标和字符移动方式的设置

状态位		指令代码	功 能
I/D	S		
0	0	04H	光标左移 1 格, AC 值减 1, 字符全部不动
0	1	05H	光标不动, AC 值减 1, 字符全部右移 1 格
1	0	06H	光标右移 1 格, AC 值加 1, 字符全部不动
1	1	07H	光标不动, AC 值加 1, 字符全部左移 1 格

4. 显示开关控制

显示开关控制指令格式如下：

控制信号		控制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

指令代码为 08H~0FH。该指令控制字符、光标及闪烁的开与关,有 3 个状态位 D、C、B,这 3 个状态位分别控制着字符、光标和闪烁的显示状态。

D 是字符显示状态位。D=1 时,为开显示;D=0 时,为关显示。注意关显示仅是字符不出现,而 DD RAM 内容不变。这与清屏指令不同。

C 是光标显示状态位。C=1 时,为光标显示;C=0 时,为光标消失。光标为底线形式(5×1 点阵),光标的位置由地址指针计数器 AC 确定,并随其变动而移动。当 AC 值超出了字符的显示范围,光标将随之消失。

B 是光标闪烁显示状态位。B=1 时,光标闪烁;B=0 时,光标不闪烁。

5. 光标、字符位移

光标、字符位移指令的格式如下：

控制信号		控制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	S/C	R/L	×	×

执行该指令将产生字符或光标向左或向右滚动一个字符位。如果定时间隔地执行该指令,将产生字符或光标的平滑滚动。具体情况如表 8-10 所列。

表 8-10 光标、字符位移的设置

状态位		指令代码	功能
S/C	R/L		
0	0	10H	光标左滚动
0	1	14H	光标右滚动
1	0	18H	字符左滚动
1	1	1CH	字符右滚动

6. 功能设置

功能设置指令格式如下：

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	N	F	0	0

该指令用于设置控制器的工作方式,有 3 个参数 DL、N 和 F,它们的作用是:

DL 用于设置控制器与计算机的接口形式。接口形式体现在数据总线长度上。DL=1 设置数据总线为 8 位长度,即 D7~D0 有效;DL=0 设置数据总线为 4 位长度,即 D7~D4 有效。在该方式下 8 位指令代码和数据将按先高 4 位后低 4 位的顺序分两次传输。

N 用于设置显示的字符行数。N=0 为一行字符行;N=1 为两行字符行。

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

7. CGRAM 地址设置

CGRAM 地址设置指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	A5	A4	A3	A2	A1	A0

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

8. DDRAM 地址设置

DDRAM 地址设置指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	A6	A5	A4	A3	A2	A1	A0

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

9. 读 BF 及 AC 值

读 BF 及 AC 指令的格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

LCD 的忙碌标志 BF 用以指示 LCD 目前的工作情况。当 BF=1 时,表示正在进行内部数据的处理,不接受单片机送来的指令或数据;当 BF=0 时,则表示已准备接收命令或数据。当程序读取此数据的内容时,D7 为忙碌标志,而另外 D6~D0 的值表示 CGRAM 或 DDRAM 中的地址,至于是指向哪一地址则根据最后写入的地址设定指令而定。

10. 写数据到 CGRAM 或 DDRAM

写数据到 CGRAM 或 DDRAM 的指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0								

先设定 CGRAM 或 DDRAM 地址,再将数据写入 D7~D0 中,以使 LCD 显示出字形。也可将使用者自创的图形存入 CGRAM。

11. 从 CGRAM 或 DDRAM 读取数据

从 CGRAM 或 DDRAM 读取数据的指令格式如下:

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	1								

先设定 CGRAM 或 DDRAM 地址,再读取其中的数据。

四、字符显示实验

实验 5 在下载型实验板上带有 LCD 接口,可直接与字符型液晶相连。实验板上数据线被连到 P0 口,P2.5 接 RS 端,P2.6 接 RW 端,P2.7 接 E 端,如图 8-27 所示。采用下载型实验板和 1602 液晶显示模块做如下实验:在液晶模块的第 1 行第 1 个字符的位置显示字母 A。

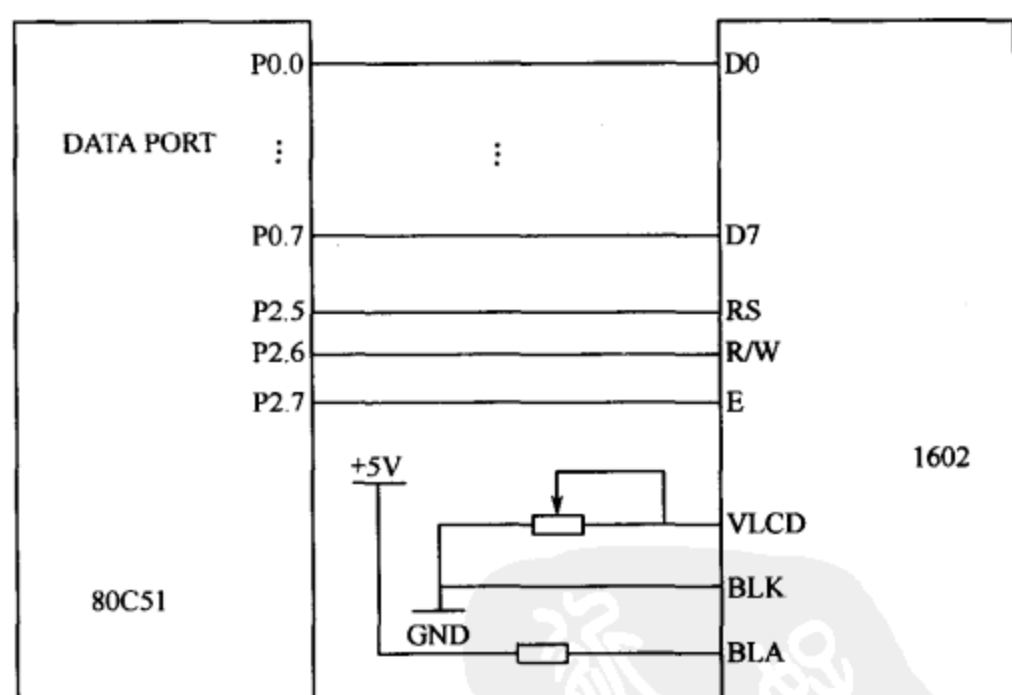


图 8-27 字符型液晶显示器与单片机的连接

实验的源程序如下:

```

RS EQU P2.5
RW EQU P2.6
E EQU P2.7
D0_D7 EQU P0
ORG 0000H

```

```

    AJMP MAIN
    ORG 0100H
;以下是主程序,进行初始化
MAIN: MOV SP, #60H
      MOV A, #00111000B ;功能设置指令,8位接口,显示2行,5×7字符
      LCALL WriteIR      ;调写指令寄存器子程序
      MOV A, #00001111B ;显示开关控制指令,显示器开,光标开,光标闪烁
      LCALL WriteIR
      MOV A, #00000110B ;输入方式设置指令,字符不动,光标自动右移一格
      LCALL WriteIR
      MOV A, #10000000B ;DDRAM地址设置指令,写入显示地址为第1
      .                  ;行第1位
      ACALL WriteIR
      MOV A, #00000001B ;清屏指令,将DDRAM数据全部填入“空白”
      ACALL WriteIR
      MOV A, #01000001B ;将字母A的代码送A
      ACALL WriteDDR     ;将字母A的代码写入DDRAM
      SJMP $
;检查忙碌子程序
CheckBusy: PUSH ACC
LOOP:  CLR RS            ;选择指令寄存器
      SETB RW           ;选择读模式
      MOV D0_D7, #0FFH  ;P0口写1,准备写入
      SETB E            ;使能LCD
      MOV A, D0_D7       ;将LCD的D0~D7由P0口送A,以便查第7
      .                  ;位BF是否为0
      CLR E             ;禁止LCD
      JB ACC.7, LOOP     ;判断由LCD读入第7位BF是否为1,若为1
      .                  ;表示LCD忙
      ACALL DELAY        ;调延时子程序
      POP ACC
      RET
;写入IR寄存器子程序
WriteIR: PUSH ACC
      ACALL CheckBusy    ;调检查忙碌子程序
      CLR E              ;禁止LCD
      CLR RS            ;选择指令寄存器
      CLR RW           ;选择写模式
      SETB E            ;使能LCD

```



```

MOV D0_D7,A           ;将控制指令写入 LCD
SETB E                 ;使能 LCD
CLR E                  ;禁止 LCD
POP ACC
RET
;写入 DR 寄存器子程序
WriteDDR:PUSH ACC
ACALL CheckBusy        ;调检查忙子程序
CLR E                  ;禁止 LCD
SETB RS                ;选择数据寄存器
CLR RW                 ;选择写模式
SETB E                 ;使能 LCD
MOV D0_D7,A           ;将数据写入 LCD
SETB E                 ;使能 LCD
CLR E                  ;禁止 LCD
POP ACC
RET
;以下是 2.5ms 延时子程序
DELAY:MOV R5,#5
D2:   MOV R4,#248
D1:   DJNZ R4,D1
      DJNZ R5,D2
      RET
      END

```

实验步骤如下：

(1)用下载型实验板进行 LCD 实验时,需断开数码管的供电电路。JP4 用于选择显示器。在 JP4 的下方标有 Disp Select 字样,上方分别标有 LED 和 LCD 字样。将短路子插于 LED 一方,选择 LED 作为显示器;若插于 LCD 一方,则选择 LCD 作为显示器。这里应选择 LCD 作为显示器。

(2)下载型实验板上有一个 16 针标准接线插座,标号为 J3。在 J3 下方注有 LCD 字样;在 J3 的上方用数字写出了 1~16 的字样,标出了接线的序号。实验板与 LCD 显示模块连接时,应注意引脚序号对应。实验板下方的蓝色电位器 P1 用于调整 LCD 的对比度。

(3)将 MON51 硬件仿真器和 PC 机连接好。

(4)取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

(5)给实验板和仿真器通电。

(6)打开 Keil 软件,输入上面的程序,保存为 lcd1.asm。对程序进行编译、链接,调试运行。图 8-28 是 LCD 仿真效果图。

该实验程序在本书所附光盘的 example\ch_8\lcd1 文件夹中。

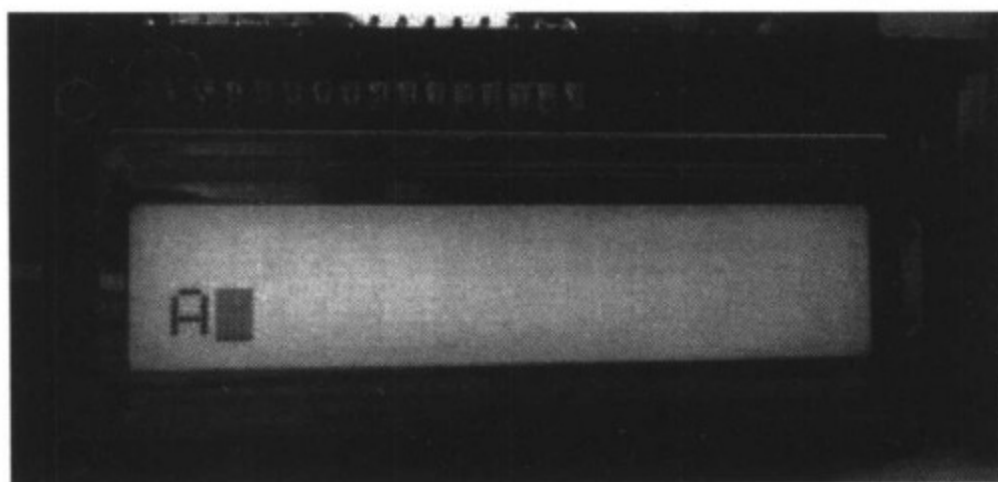


图 8-28 LCD 显示字母 A 仿真效果图

实验 6 在下载型实验板做以下实验, LCD 的第 1 行显示 80C51, 第二行显示 Dian Pian Ji(单片机的拼音)

实验的源程序如下:

```
RS EQU P2.5
RW EQU P2.6
E EQU P2.7
D0_D7 EQU P0
ORG 0000H
AJMP MAIN
ORG 0100H
```

;以下是主程序,进行初始化

MAIN: MOV SP, #60H

MOV A, #00111000B

;功能设置指令,8 位接口,显示 2 行,5×7 字符

LCALL WriteIR

;调写指令寄存器子程序

MOV A, #00001110B

;显示开关控制指令,显示器开,光标开,光标不闪烁

LCALL WriteIR

MOV A, #00000110B

;输入方式设置指令,字符不动,光标自动右移一格

LCALL WriteIR

MOV A, #00000001B

;清屏指令,将 DDRAM 数据全部填入“空白”

ACALL WriteIR

MOV A, #10000000B

;DDRAM 地址设置指令,写入显示地址为第 1 行第 1 位

ACALL WriteIR

MOV DPTR, #TAB1

;指向 TAB1 表首

ACALL STRING

;调字符串处理子程序

MOV A, #11000000B

;DDRAM 地址设置指令,写入显示地址为第 2 行第 1 位

ACALL WriteIR;

MOV DPTR, #TAB2	;指向 TAB1 表首
ACALL STRING	;调字符串处理子程序
SJMP \$	
;检查忙碌子程序	
CheckBusy: PUSH ACC	
LOOP: CLR RS	;选择指令寄存器
SETB RW	;选择读模式
MOV D0_D7, #0FFH	;P0 口写 1, 准备写入
SETB E	;使能 LCD
MOV A, D0_D7	;将 LCD 的 D0~D7 由 P0 口送 A, 以便查第 7 位 BF 是否为 0
CLR E	;禁止 LCD
JB ACC. 7, LOOP	;判断由 LCD 读入第 7 位 BF 是否为 1, 若为 1 表示 LCD 忙
ACALL DELAY	;调延时子程序
POP ACC	
RET	
;写入 IR 寄存器子程序	
WriteIR: PUSH ACC	
ACALL CheckBusy	;调检查忙碌子程序
CLR E	;禁止 LCD
CLR RS	;选择指令寄存器
CLR RW	;选择写模式
SETB E	;使能 LCD
MOV D0_D7, A	;将控制指令写入 LCD
SETB E	;使能 LCD
CLR E	;禁止 LCD
POP ACC	
RET	
;写入 DR 寄存器子程序	
WriteDDR: PUSH ACC	
ACALL CheckBusy	;调检查忙碌子程序
CLR E	;禁止 LCD
SETB RS	;选择数据寄存器
CLR RW	;选择写模式
SETB E	;使能 LCD
MOV D0_D7, A	;将数据写入 LCD
SETB E	;使能 LCD
CLR E	;禁止 LCD

```

        POP ACC
        RET
;以下是字符串处理子程序
STRING: PUSH ACC
LOOP1:  MOV A, #00H
        MOVC A, @A+DPTR
        JZ PROC
        ACALL WriteDDR
        INC DPTR
        AJMP LOOP1
PROC:   POP ACC
        RET
;以下是 2.5ms 延时子程序
DELAY:  MOV R5, #5
D2:     MOV R4, #248
D1:     DJNZ R4, D1
        DJNZ R5, D2
        RET
TAB1: DB 38H,30H,43H,35H,31H,00H;80C51 的代码,00H 表示结束
TAB2: DB 44H,69H,61H,6EH,0FEH,50H, 69H,61H,6EH,0FEH,4AH,69H,00H
                                           ;Dian Pian Ji 的代码,0FEH 为空格,
                                           00H 表示结束

        END

```

实验步骤同实验 5。

该实验程序在本书所附光盘的 example\ch_8\lcd2 文件夹中。

五、汉字图形的显示原理与实验

HD44780 内置两种字符发生器。一种为 CGROM,即已固化好的字模库,参见表 8-7 所列。单片机只要写入某个字符的字符代码,LCD 就可以将该字符显示出来。另一种为 CGRAM,即可随时定义的字符字模库。HD44780 提供 64B 的 CGRAM,地址为 00H~3FH。它可以生成 8 个 5×8 点阵(作为光标行也可以占用)的自定义字符或 4 个 5×11 点阵(作为光标行也可以占用)的自定义字符,由于 HD44780 仅使用一行 5 位数据为字符点阵,所以作为 CGRAM 字模库,仅使用存储单元字节的低 5 位,而高 3 位虽然存在,但不作为字模数据使用。HD44780 提供给 CGRAM 的字符字模代码为 00H~07H 或 08H~0FH。作为 5×8 点阵字符的字模库,CGRAM 每 8B 为一个字符的字模数据,字模数据存储顺序是从上至下排列。每个字符代码都对应着 CGRAM 的 8 个单元,作为 5×11 点阵字符的字模库,CGRAM 每 16B 为一个字符的字模数据,其中前 11B 为字模数据存储单元,后 5B 与字模无关。字符代码与 CGRAM 地址的对应关系如表 8-11 所列。

表 8-11 字符代码与 CGRAM 地址的对应关系

5×8 点阵字符		5×11 点阵字符	
字符代码	CGRAM 地址	字符代码	CGRAM 地址
00H(08H)	00H~07H	00H(08H)	00H~0FH
01H(09H)	08H~0FH	01H(09H)	10H~1FH
02H(0AH)	10H~17H	02H(0AH)	20H~2FH
03H(0BH)	18H~1FH	03H(0BH)	30H~3FH
04H(0CH)	20H~27H		
05H(0DH)	28H~2FH		
06H(0EH)	30H~37H		
07H(0FH)	38H~3FH		

图 8-29 给出了 CGRAM 地址及“月”字模数据与显示效果的对应关系。









CGRAM 地址	CGRAM 单元数据	显示屏的显示效果
00H	0FH	
01H	09H	
02H	0FH	
03H	09H	
04H	0FH	
05H	09H	
06H	13H	
07H	00H	

图 8-29 CGRAM 地址及“月”字模数据与显示效果的对应关系

实验 7 在下载型实验板上做以下实验：LCD 的第 2 行显示“2005 年 5 月 1 日”。
实验的源程序如下：

```
RS EQU P2.5
RW EQU P2.6
E EQU P2.7
D0_D7 EQU P0
ORG 0000H
AJMP MAIN
ORG 0100H
;以下是主程序,进行初始化
MAIN: MOV SP, #60H
      MOV A, #00000001B ;清屏指令,将 DDRAM 数据全部填入“空白”
      ACALL WriteIR
      MOV A, #00111000B ;功能设置指令,8 位接口,显示 2 行,5×7 字符
      ACALL WriteIR ;调写指令寄存器子程序
      MOV A, #00001110B ;显示开关控制指令,显示器开,光标开,光标不闪烁
```


ACALL WriteIR	
MOV A, #00000110B	;输入方式设置指令,字符不动,光标自动右移一格
ACALL WriteIR	
ACALL WriteCGRAM	;调自定义字体子程序
ACALL STRING	;调字符串处理子程序
SJMP \$	
;检查忙碌子程序	
CheckBusy: PUSH ACC	
LOOP: CLR RS	;选择指令寄存器
SETB RW	;选择读模式
MOV D0_D7, #0FFH	;P0 口写 1,准备写入
SETB E	;使能 LCD
MOV A, D0_D7	;将 LCD 的 D0~D7 由 P0 口送 A,以便查第 7 位 BF 是否为 0
CLR E	;禁止 LCD
JB ACC. 7, LOOP	;判断由 LCD 读入第 7 位 BF 是否为 1,若为 1 表示 LCD 忙
ACALL DELAY	;调延时子程序
POP ACC	
RET	
;写入 IR 寄存器子程序	
WriteIR: PUSH ACC	
ACALL CheckBusy	;调检查忙子程序
CLR E	;禁止 LCD
CLR RS	;选择指令寄存器
CLR RW	;选择写模式
SETB E	;使能 LCD
MOV D0_D7, A	;将控制指令写入 LCD
SETB E	;使能 LCD
CLR E	;禁止 LCD,使 E 信号变为下降沿
POP ACC	
RET	
;写入 DR 寄存器子程序	
WriteDDR: PUSH ACC	
ACALL CheckBusy	;调检查忙碌子程序
CLR E	;禁止 LCD
SETB RS	;选择数据寄存器
CLR RW	;选择写模式

```

SETB E                ;使能 LCD
MOV D0_D7,A           ;将数据写入 LCD
SETB E                ;使能 LCD
CLR E                 ;禁止 LCD,使 E 信号变为下降沿
POP ACC
RET

```

;以下是自定义字体程序

```

WriteCGRAM:PUSH ACC
MOV A,#01000000B      ;设置 CGRAM 的地址
ACALL WriteIR
MOV R0,#24
MOV DPTR,#CGRAMTAB;指向 CGRAMTAB 表首
LOOP1: CLR A
MOVC A,@A+DPTR
ACALL WriteDDR        ;将自定义字符存入 CGRAM 中
INC DPTR
DJNZ R0,LOOP1
POP ACC
RET

```

;以下是字符串处理子程序

```

STRING:PUSH ACC
MOV A,#11000000B      ;设置 DDRAM 的地址,从第 2 行第 1 列开始显示
ACALL WriteIR
MOV R1,#09H
MOV DPTR,#TAB         ;指向 TAB 表首
LOOP2: CLR A
MOVC A,@A+DPTR
ACALL WriteDDR
INC DPTR
DJNZ R1,LOOP2
POP ACC
RET

```

;以下是 2.5ms 延时子程序

```

DELAY:MOV R5,#5
D2:    MOV R4,#248
D1:    DJNZ R4,D1
        DJNZ R5,D2
RET

```

CGRAMTAB:

```

DB 08H,0FH,12H,0FH,0AH,1FH,02H,02H    ;年的字模
DB 0FH,09H,0FH,09H,0FH,09H,13H,00H    ;月的字模
DB 1FH,11H,11H,1FH,11H,11H,1FH,00H    ;日的字模

```

TAB:

```

DB 32H,30H,30H,35H,00H,35H,01H,31H,02H;2005 年 5 月 1 日
END

```

实验步骤同实验 5。该实验程序在本书所附光盘的 example\ch_8\lcd3 文件夹中。

第四节 SPI 总线接口和看门狗电路

一、SPI 总线接口

1. SPI 总线简介

SPI 是 MOTOROLA 公司推出的串行扩展接口,它可以使单片机与各种外围设备以串行方式进行通信以交换信息。SPI 总线接口一般使用 4 条线:串行时钟线(SCK)、主机输入/从机输出数据线 MISO、主机输出/从机输入数据线 MOSI 和低电平有效的从机选择线 \overline{CS} ,由于 SPI 系统总线一共只需 4 位数据和控制线,而扩展并行总线则需要 8 根数据线、8 位~16 位地址线、2 位~3 位控制线,因此,采用 SPI 总线接口可以简化电路设计,节省很多常规电路中的接口器件和 I/O 接口线,提高设计的可靠性。由此可见,在 MCS-51 系列等不具有 SPI 接口的单片机组成的智能仪器和工业测控系统中,当传输速度要求不是太高时,使用 SPI 总线可以增加应用系统接口器件的种类,提高应用系统的性能。

2. SPI 总线的组成

利用 SPI 总线可在软件的控制下构成各种系统。如一个单片机(主设备)和一个或几个 I/O 设备(从设备)等。可使用单片机作为主机来控制数据,并向一个或几个从外围器件传送该数据。从器件只有在主机发命令时才能接收或发送数据。其数据的传输格式是高位(MSB)在前,低位(LSB)在后。SPI 总线接口系统的典型结构如图 8-30 所示。

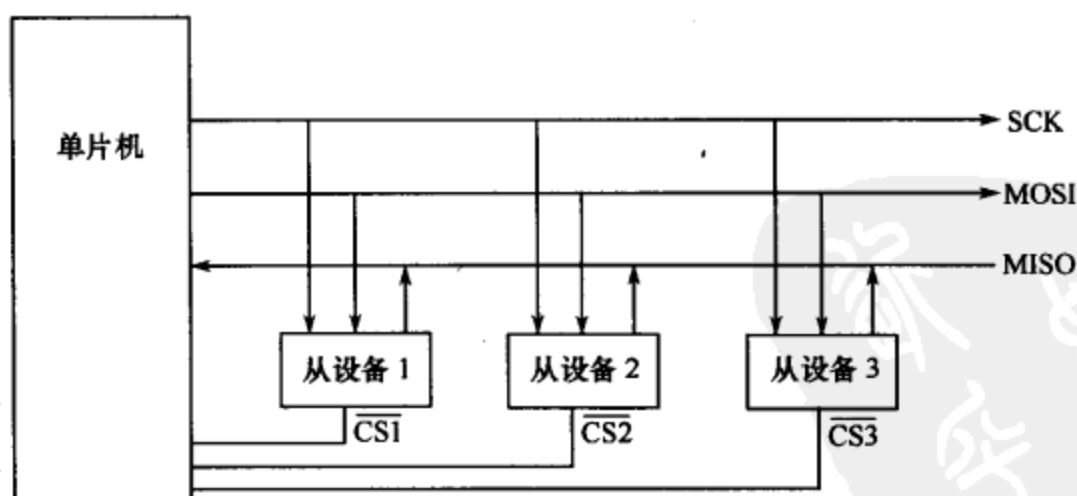


图 8-30 SPI 总线接口系统的典型结构

当单片机(主机)通过 SPI 与几种不同的串行 I/O 芯片(从机)相连时,必须使用每片的允许控制端 \overline{CS} ,这可通过单片机的 I/O 端口输出线来实现。但应特别注意这些串行 I/O 芯片的输入/输出特性:首先是输入芯片的串行数据输出是否有三态控制端。平时未选

中芯片时,输出端应处于高阻态。若没有三态控制端,则应外加三态门。否则,单片机的MISO端只能连接一个输入芯片。其次是输出芯片的串行数据输入是否有允许控制端。因此只有在此芯片允许时,SCK脉冲才把串行数据移入该芯片;在禁止时,SCK对芯片无影响。若没有允许控制端,则应在外围用门电路对SCK进行控制,然后再加到芯片的时钟输入端。当然,也可以只在SPI总线上连接一个芯片,而不再连接其他输入或输出芯片。

二、看门狗电路

看门狗(WDT, Watch Dog Timer)电路,其硬件主要由一个定时器组成,在打开看门狗时,定时器开始工作,定时时间一到,触发单片机复位,这样,在软件设计时,在合适的地方对看门狗定时器清0,只要软件运行正常,单片机就不会出现复位。当软件死机或出错时,则不对看门狗定时器清0,看门狗定时时间到后对单片机进行复位,使系统重新开始工作,从而保证系统的正常运行。其中,看门狗定时时间可以由用户设定,软件上对看门狗的操作很简单,只有3种操作:打开看门狗、关闭看门狗和看门狗定时器清0。

看门狗电路的型号较多,应用较广泛的是XICOR公司X25045和X5045,X25045属早期产品,X5045是后期改进型产品,与X25045完全兼容。下面以X5045为例进行介绍。

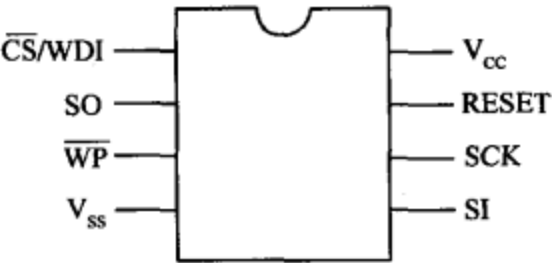


图 8-31 X5045 引脚图

1. X5045 的结构

X5045 是将上电复位、可编程看门狗、电压监控、串行 EEPROM 4 种功能于一体的多功能芯片,具有体积小、占用 I/O 少等优点,可以简化单片机系统的设计,并完善其性能。

X5045 芯片共有 8 个引脚,如图 8-31 所示,引脚功能如表 8-12 所列。

表 8-12 X25045 引脚功能

引脚	名称	功能说明
1	$\overline{\text{CS}}/\text{WDI}$	片选择输入:当 $\overline{\text{CS}}$ 为高电平时,芯片未被选中,SO呈高阻态;当 $\overline{\text{CS}}$ 为低电平时,器件被选中。 看门狗输入:在看门狗定时器超时并产生复位之前,一个加在WDI引脚的由高到低的电平变化将复位看门狗定时器
2	SO	串行输出:SO是一个串行数据输出引脚,在读数据时,数据在SCK脉冲的下降沿,由这个引脚送出
3	$\overline{\text{WP}}$	写保护输入:当 $\overline{\text{WP}}$ 引脚为低电平时,向X5045中写的操作被禁止,但是其他功能可以正常执行
4	V _{SS}	地
5	SI	串行输入:SI是串行数据输入端,指令码、地址、数据都通过这个引脚进行输入。在SCK的上升沿进行数据输入,并且高位(MSB)在前
6	SCK	串行时钟输入:串行时钟的上升沿通过SI脚进行数据的输入,下降沿通过SO引脚进行数据的输出
7	RESET	复位输出:该脚是一个开漏输出脚,在使用时必须接上拉电阻。只要V _{CC} 跌落到最小允许值,这个引脚就会输出高电平,一直到V _{CC} 上升超过最小允许值之后200ms。同时它也受看门狗定时器控制。只要看门狗处于激活状态,并且WDI引脚上电平(高电平或者低电平)超过了定时的时间,就会产生复位信号。 $\overline{\text{CS}}$ 引脚上的一个下降沿会复位看门狗定时器
8	V _{CC}	电源电压

2. X5045 的状态寄存器

X5045 内部有一个状态寄存器,可以用来提供 X5045 的状态信息以及设置块保护和看门狗的超时功能。

状态寄存器的格式如下:

位	7	6	5	4	3	2	1	0
符号	—	—	WD1	WD0	BL1	BL0	WEL	WIP

其中,WD1、WD0 是看门狗定时时间设置位。表 8-13 是 WD1、WD0 组合的含义。

表 8-13 看门狗定时器的时间设定

状态寄存器位		看门狗定时值/ms
WD1	WD0	
0	0	1400
0	1	600
1	0	200
1	1	禁止看门狗工作

BL1、BL0 是内部 EEPROM 存储单元数据写保护区设置位,其设置情况如表 8-14 所列。

表 8-14 EEPROM 存储单元数据写保护设置

状态寄存器位		写保护的单元地址
BL1	BL0	
0	0	没有保护
0	1	180H~1FFH
1	0	100H~1FFH
1	1	000H~1FFH

WEL 是只读标志。1 表明写使能开关打开,在进行任何写操作前都必须打开写使能开关,在上电初始化写操作完成时,写使能开关自动关闭。

WIP 也是只读标志。WIP=1 时,表示芯片正忙于写操作;WIP=0 时,表示没有进行写操作。

上电复位时,状态寄存器各位都被清零。

3. X5045 的功能

X5045 的功能包括以下 4 种:

(1)上电复位。在对 X5045 通电时,RESET 引脚输出有效的高电平复位信号,并保持至少 200ms,使 CPU 有效复位。

(2)电源电压监控。当检测到电源电压低于内部门槛电压 V_{TRIP} 时,RESET 输出高电平复位信号,直至电源电压高于 V_{TRIP} 并保持至少 200ms,复位信号才被撤消,使得单片机可以继续工作。

(3)看门狗定时器。看门狗定时器通过监测 WDI 的输入来判断单片机是否正常工作,在设定的定时时间内,单片机必须在 WDI 引脚上产生一个由高到低的电平变化,否则,X5045 将产生一个高电平的复位信号。

状态寄存器的 WD1、WD0 是看门狗定时设置位,通过状态寄存器写指令 WRSR 修改这两个标志位,就能在 3 种定时间隔中进行选择或关闭定时器。对看门狗的复位由 \overline{CS} 输入电平的下降沿完成。

重点提示 看门狗定时器对单片机提供了独立的保护系统,当系统出现故障时,只要计时达到其编程的超时极限,或者当电源电压降到最低转换点以下时,RESET 引脚就会立即输出高电平复位信号。该芯片在系统上电或掉电时,RESET 引脚也会立即输出高电平复位信号,从而避免了因系统故障、电源开断、瞬时电压不稳等的影响。

(4)串行 EEPROM

X5045 内 4KB 的 EEPROM 可以 10 万次擦写,可以保存一些重要数据(如参数设定),即使掉电也不会丢失。单片机对 X5045 的读写是分别通过写端口 SI、读端口 SO 和时钟端口 CLK 按照一定的时序配合来实现的。X5045 内的 EEPROM 除可以由 \overline{WP} 引脚置低保护以外,还可以被软件保护,通过指令可以设置状态寄存器,来保护这 4 KB 存储器中的某一部分或者全部。

在实际使用时,X5045 的 SO 和 SI 脚不会被同时用到,可以将 SO 和 SI 接在一起,因此,也称这种接口为三线制 SPI 接口。

4. X5045 的指令寄存器

X5045 内部有一个 8 位指令寄存器,单片机通过对指令寄存器写命令,实现对 X5045 的操作,X5045 的指令有 6 条,如表 8-15 所列,指令、地址和数据均以高位在前的方式串行传送。

表 8-15 X5045 的指令

指令名称	指令格式	内容
WREN	0000 0110	打开写使能开关
WRDI	0000 0100	关闭写使能开关
RDSR	0000 0101	读状态寄存器
WRSR	0000 0001	写状态寄存器
READ	0000 A ₈ 011	读 EEPROM 存储单元
WRITE	0000 A ₈ 010	写 EEPROM 存储单元

下面对表中的 6 条指令再作一简要说明：

(1)WREN 和 WRDI 是写使能开关的开/关指令。它们都是单字节指令。

(2)RDSR 和 WRSR 是状态寄存器的读/写指令。在从 SI 输入指令后,RDSR 的执行结果,即状态寄存器内容须从 SO 读出;而 WRSR 需要紧接着输入修改数据。

(3)READ 和 WRITE 是 EEPROM 存储单元的读/写指令。输入指令后(指令码第 3 位代表存储单元地址的最高位 A8),接着输入低 8 位地址,最后就可以连续读出或写入数据。另外,由于 EEPROM 的写入时间长,所以在连续两条写指令之间应读取 WIP 状态,只有内部写周期结束时才可输入下一条写指令。

5. X5045 程序设计

图 8-32 是 X5045 与单片机的接口电路,图中的 K 键起手动复位的作用。

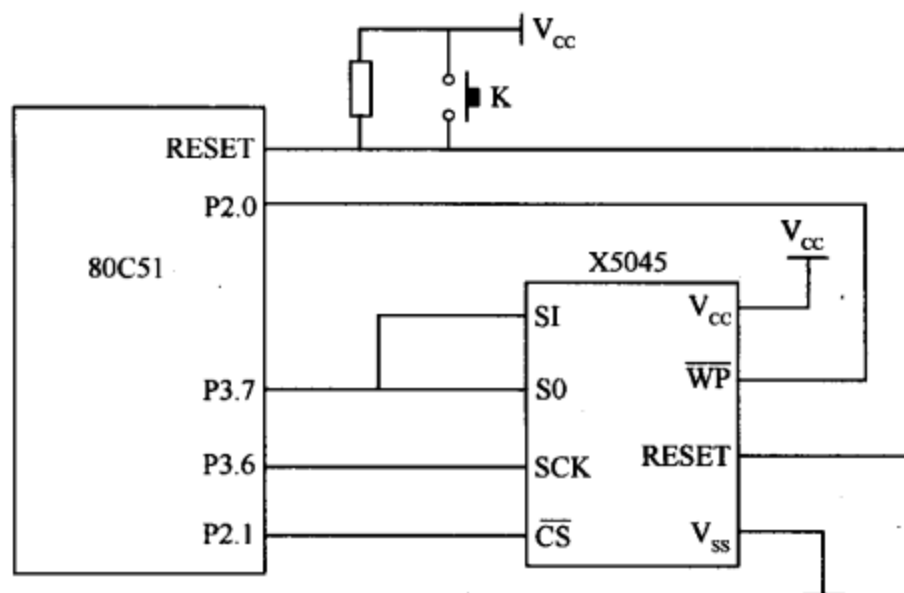


图 8-32 X5045 与单片机的接口电路

X5045 采用 SPI 串行接口,只要了解 SPI 原理和 X5045 的指令定义,就很容易对 X5045 进行操作。下面介绍几种常用的子程序。

在编程之前,先用伪指令 BIT 对有关变量进行定义:

SO BIT P3.7

WP BIT P2.0

SI BIT P3.7

SCK BIT P3.6

CS BIT P2.1

(1)看门狗复位子程序(\overline{CS} 引脚上的一个下降沿会复位看门狗定时器)如下:

```
WD_RST:SETB CS
```

```
        NOP
```

```
        CLR CS
```

```
        NOP
```

```
        SETB CS
```

```
        RET
```

(2)看门狗定时时间设置成 600ms,程序如下:

```
WD_SET:MOV R0, #00011000B    ;设看门狗定时时间为 600ms,块保护地址为
                                100H~1FFH
        LCALL WRITE_SR        ;调“写状态寄存器”子程序
        RET
```

(3)读状态寄存器子程序,将读出的 X5045 的状态存在 A 累加器中,程序如下:

```
READ_SR:LCALL STAX            ;调“启动 X5045”指令
        MOV A, #00000101B    ;将读状态寄存器指令送 A 累加器
        LCALL WRITE          ;调“写指令子程序”
        LCALL READ           ;调“读数据子程序”
        LCALL ENDX           ;调“结束 X5045”指令
        RET
```

方法技巧 要读状态寄存器,首先将 \overline{CS} 接低电平以选择该器件,然后送一个 8 位的读状态寄存器 RDSR 指令,然后状态寄存器的内容就通过 SO 线进行输出,当然,必须要有相应的时钟加到 SCK 线上。

(4)写状态寄存器子程序如下:

```
WRITE_SR:LCALL STAX           ;调“启动 X5045”指令
        LCALL READ_SR        ;调“读状态寄存器”子程序
        JB ACC.0, WRITE_SR    ;ACC.0(WIP)等于 1,转 WRITE_SR
WRITE_SR1:LCALL STAX          ;调“启动 X5045”指令
        MOV A, #00000110B    ;打开写使能开关
        LCALL WRITE          ;将指令写入
        LCALL STAX           ;调“启动 X5045”指令
        MOV A, #00000001B    ;送写状态寄存器指令
        LCALL WRITE          ;将指令写入
        MOV A, #00011000B    ;设看门狗定时时间为 600ms,块保护
                                地址为 100H~1FFH
        LCALL WRITE          ;将数据写入
        LCALL ENDX           ;调“结束 X5045”指令
        RET
```

方法技巧 要将数据写入状态寄存器,首先必须用 WREN 命令将 WEL 置 1,打开写使能开关,接着写入写状态寄存器 WRSR 指令,跟着写入 8 位数据,这个 8 位数据就是相应的寄存器中的内容。写入结束后必须将 \overline{CS} 拉至高电平,如果 \overline{CS} 没有在 WREN 和 WRSR 期间变高,则 WRSR 指令将被忽略。

(5)读 X5045 存储器 1B 数据子程序,待读出的地址在 R3 中,读出后数据存 R4,程序如下:

```
READ_B:LCALL READ_SR         ;调“读状态寄存器”子程序
        JB ACC.0, READ_B     ;ACC.0(WIP)等于 1,转 READ_B 等待
        LCALL STAX           ;调“启动 X5045”指令
        MOV A, #03H          ;送“读 EEPROM”指令,地址为 00H~FFH
```

LCALL WRITE	;将指令写入
MOV A,R3	;将 R3 中的地址送 A
LCALL WRITE	;将地址写入 X5045
LCALL READ	;读出 X5045 中的内容
MOV R4,A	;将读出的内容送 R4
LCALL ENDX	;调“结束 X5045”指令
RET	

重点提示 要读存储器的内容,首先将 \overline{CS} 拉低以选中该器件,然后将 8 位的读指令送到器件中去,跟着送 8 位的地址,读指令的 3 位(A8)用于选择存储器的上半区或下半区,在读指令和地址发送完毕后,所选中的地址单元的数据通过 SO 线送出,在读完这一字节后,如果继续提供时钟脉冲,则这一地址单元的下一个单元的数据将会被顺序读出,地址将会自动地增加。当到达最高地址之后,地址将会回绕到 00H 单元,读周期在 \overline{CS} 变为高电平后终止。

(6)写 1B 数据子程序,写入地址在 R3 中,数据在 R4 中,程序如下:

WRITE_B:LCALL READ_SR	;调“读状态寄存器”子程序
JB ACC.0,WRITE_B	;ACC.0(WIP)等于 1,转 WRITE_B 等待
LCALL STAX	;调“启动 X5045”子程序
MOV A,#06H	;打开写使能开关
LCALL WRITE	;将指令写入
LCALL STAX	;调“启动 X5045”子程序
MOV A,#02H	;送写 EEPROM 指令,地址为 00H~FFH
LCALL WRITE	;将指令写入
MOV A,R3	;将 R3 中的地址送 A
LCALL WRITE	;将地址写入 X5045
MOV A,R4	;送待写数据
LCALL WRITE	;将数据写入
LCALL ENDX	;调“结束 X5045”子程序
RET	

方法技巧 要写存储器内容,WEL 位必须通过 WREN 指令置为 1,随后写入 WRITE 指令,并跟随 8 位的地址和数据,WRITE 指令的位 3 用于选择存储器的上半区和下半区,如果 \overline{CS} 没有在 WREN 和 WRITE 指令之间变为高电平,则 WRITE 指令被忽略。

(7)供调用的子程序如下:

;以下是启动 X5045 子程序

STAX: SETB CS	;启动 X5045 指令
NOP	;先拉高 CS,再拉低 SCK,最后拉低 CS
CLR SCK	
NOP	
CLR CS	

```

NOP
RET
;以下是结束 X5045 子程序
ENDX: CLR SCK           ;结束 X5045 指令
      SETB CS           ;先拉低 SCK 后拉高 CS
      NOP
      NOP
      RET
;以下是写 X5045 子程序
WRITE:MOV R0, #08H      ;写 8 位数据
WRITE1:RLC A            ;将 A 中的指令数据循环左移
      MOV SI,C          ;将 C 中的第 1 位(最高位)送 X5045 的 SI
      CLR SCK
      SETB SCK          ;将 SCK 设置为上升沿,以便通过 SI 脚进行数据的
                        ;输入
      DJNZ R0,WRITE1    ;R0 不等 0 则转 WRITE1 处理程序,以便将 A 中
                        ;的 8 位数据写完
      RET
;以下是读 X5045 子程序
READ: MOV R0, #08H      ;读 8 位数据
READ1:SETB SCK
      CLR SCK           ;将 SCK 设置为下降沿,以便通过 SO 脚进行数据的
                        ;输出
      MOV C,SO          ;将 X5045 中的数据通过 SO 脚输出到 C
      RLC A             ;将 A 中的数据循环左移
      DJNZ R0,READ1     ;若 R0 不等于 0 转 READ1 处理程序,以便将
                        ;X5045 中 8 位数据读完
      RET

```

归纳总结 综合起来,读写 X5045 有以下几条规则:

(1)SCK 由 1 变 0(下降沿)时,从 SO 引脚读取 X5045 的 1 位数据,SCK 由 0 变为 1(上升沿)时,向 X5045 的 SI 引脚发送的 1 位数据。X5045 正是基于这一原理实现基本读写操作的。

(2)在进行任何以字节为单位的读写操作前,应先选中芯片,即复位 \overline{CS} ;置位 \overline{CS} 则表示操作结束。为了防止误操作,每一次复位或置位 \overline{CS} 时应复位 SCK。

(3)写操作前应先读取状态寄存器,判断 WIP 为 0 时,在写使能允许命令后就可以写状态寄存器或向 EEPROM 写数据。

6. X5045 实验

实验 8 用下载型实验板做以下实验:对 X5045 进行写入和读出,并对看门狗超时周期进行设置(设置为 600ms)。

实验板上 P2.1 接 X5045 的 $\overline{\text{CS}}$ 端, P3.7 接 X5045 的 SI 和 SO, P3.6 接 X5045 的 SCK, P2.0 接 X5045 的 $\overline{\text{WP}}$ 端。

实验源程序如下:

```
SO BIT P3.7
WP BIT P2.0
SI BIT P3.7
SCK BIT P3.6
CS BIT P2.1
ORG 0000H
AJMP START
ORG 0040H
START: MOV SP, #60H
      SETB WP           ;写操作不保护
TEST:  MOV R4, #89H     ;定义 R4 为写入的数据(89H)
      MOV R3, #00H     ;定义 R3 为写入数据的地址(00H)
      ACALL WRITE_B
      ACALL READ_B
      SJMP TEST
;下面加入前面介绍的几个子程序
:
END
```

实验步骤如下:

(1) 打开 Keil 软件, 输入上面的程序, 保存为 x5045.asm。对程序进行编译、链接和调试, 产生 x5045.hex 目标文件。

(2) 下载型实验板提供了两种复位电路, 即 RC 复位与外接芯片复位。JPI 用于复位选择, 在该插针座下标有 Reset Select 字样, 很容易辨认。在该插针座上方左侧标有 RC 字样, 右侧标有 X5045(3) 字样。如果用短路子插于左侧, 则选择 RC 复位电路; 如果用短路子插于右侧, 则选择 X5045 复位, 可用于测试 X5045 芯片的看门狗特性。不论短路子插于哪一侧, X5045 芯片内部的 EEPROM 存储器总是可用的。这里, 用短路子插于右侧, 选择 X5045 复位。

(3) 将 MON51 硬件仿真器和 PC 机连接好。

(4) 取下下载型实验板的 CPU(SST89E554RC), 将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

(5) 点击菜单 Project→Option for Target 'Target', 在出现的窗口中设置为软件模拟仿真(Use Simulator)。

(6) 按 Ctr+F5 进入调试状态。

(7) 按 F11 或 F10, 进行单步运行和单步过程运行, 运行后, R3、R4、A 寄存器的状态如图 8-33 所示。

该实验程序在本书所附光盘的 example\ch_8\x5045 文件夹中。

Register	Value
<input type="checkbox"/> Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x89
r5	0x03
r6	0x03
r7	0x01
<input type="checkbox"/> Sys	
a	0x89
b	0x00
sp	0x60
dptr	0x0000
PC \$	C:0x004D
<input checked="" type="checkbox"/> psw	0x00

图 8-33 寄存器的状态

第五节 I²C 总线接口

80C51 单片机本身不具有 I²C 接口,但是,通过用软件进行模拟,可以挂接具有 I²C 接口的芯片。

一、I²C 总线及其软件包

串行扩展总线在单片机系统中的应用是目前单片机技术发展的一种趋势。在目前比较流行的几种串行扩展总线中,I²C 总线以其严格的规范和众多带 I²C 接口的外围器件而获得广泛的应用。I²C 总线是 PHILIPS 公司推出的芯片间串行传输总线。它由两根线组成,一根是串行时钟线(SCL),一根是串行数据线(SDA)。主控器利用串行时钟线发出时钟信号,利用串行数据线发送或接收数据。I²C 总线由主控器电路引出,凡具有 I²C 接口的电路(受控器)都可以挂接在 I²C 总线上,主控器通过 I²C 总线对受控器进行控制。

随着 I²C 总线研究的深入,I²C 总线已经广泛应用于视/音频领域、IC 卡行业和一些家电产品中,在智能仪器、仪表和工业测控领域也越来越多地得到应用。

1. I²C 总线的特点

I²C 总线的广泛应用是同它卓越的性能和简便的操作方法分不开的。I²C 总线的特点主要表现在以下几个方面:

(1)硬件结构上具有相同的硬件接口界面。I²C 总线系统中,任何一个 I²C 总线接口的外围器件,不论其功能差别有多大,都是通过串行数据线(SDA)和串行时钟线(SCL)连

接到 I²C 总线上。这一特点给用户在设计应用系统中带来了极大的便利性。用户不必理解每个 I²C 总线接口器件的功能如何,只要将器件的 SDA 和 SCL 引脚连到 I²C 总线上,然后对该器件模块进行独立的电路设计,从而简化了系统设计的复杂性,提高了系统抗干扰的能力。

(2)总线接口器件地址具有很大的独立性。每个 I²C 接口芯片具有唯一的器件地址,由于不能发出串行时钟信号而只能作为从器件使用。各器件之间互不干扰,相互之间不能进行通信,各个器件可以单独供电。单片机与 I²C 器件之间的通信是通过独一无二的器件地址来实现的。

(3)软件操作的一致性。由于任何器件通过 I²C 总线与单片机进行数据传送的方式是基本一样的,这就决定了 I²C 总线软件编写的一致性。

(4)PHILIPS 公司在推出 I²C 总线的同时,也为 I²C 总线制定了严格的规范,如接口的电气特性、信号时序、信号传输的定义等。规范的严密性,结构的独立性和硬件、软件接口界面的一致性,极大地方便了 I²C 总线设计的模块化和规范化,伴随而来的是用户在使用 I²C 总线时的“傻瓜”化。

2. I²C 总线数据的传输规则

(1)在 I²C 总线上的数据线 SDA 和时钟线 SCL 都是双向传输线,它们的接口各自通过一个上拉电阻接到电源正端。当总线空闲时,SDA 和 SCL 必须保持高电平,为了使总线上所有电路的输出能完成一个线“与”的功能,各接口电路的输出端必须是开路漏极或开路集电极。

(2)进行数据传送时,在时钟信号高电平期间,数据线上的数据必须保持稳定;只有时钟线上的信号为低电平期间,数据线上的高电平或低电平才允许变化,如图 8-34 所示。

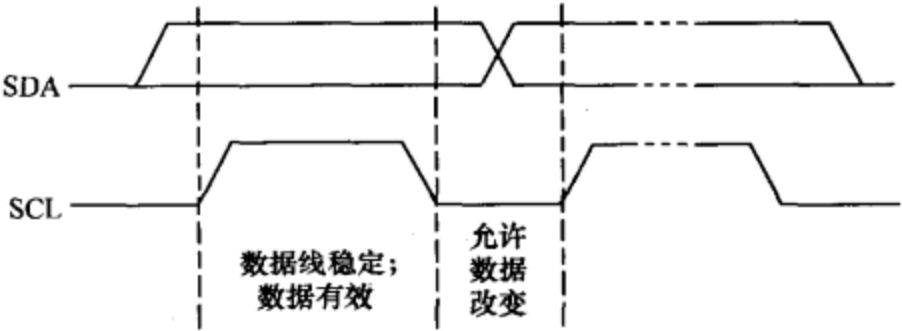


图 8-34 数据的有效性

(3)在 I²C 总线的工作过程中,当时钟线保持高电平期间,数据线由高电平向低电平变化定义为起始信号(S),而数据线由低电平向高电平的变化定义为一个终止信号(P),如图 8-35 所示,起始信号和终止信号均由主控制器产生。

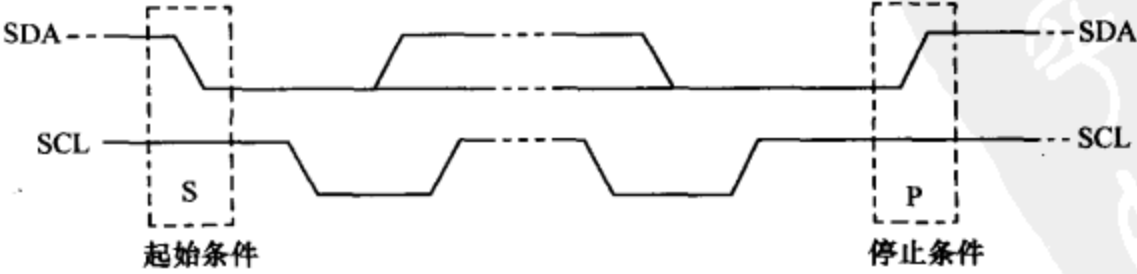


图 8-35 起始和停止条件

(4) I²C 总线传送的每 1B 均为 8 位,但每启动一次总线,传输的字节数没有限制,由主控器发送时钟脉冲及起始信号、寻址字节和停止信号,受控器件必须在收到每个数据字节后作出响应,在传送 1B 后的第 9 个时钟脉冲位,受控器输出低电平作为应答信号。此时,要求发送器在第 9 个时钟脉冲位上释放 SDA 线,以便受控器送出应答信号,将 SDA 线拉成低电平,表示对接收数据的认可,应答信号用 ACK 或 A 表示,非应答信号用 $\overline{\text{ACK}}$ 或 $\overline{\text{A}}$ 表示,当确认后,主控器可通过产生一个停止信号来终止总线数据传输。I²C 总线数据传输示意图如图 8-36 所示。

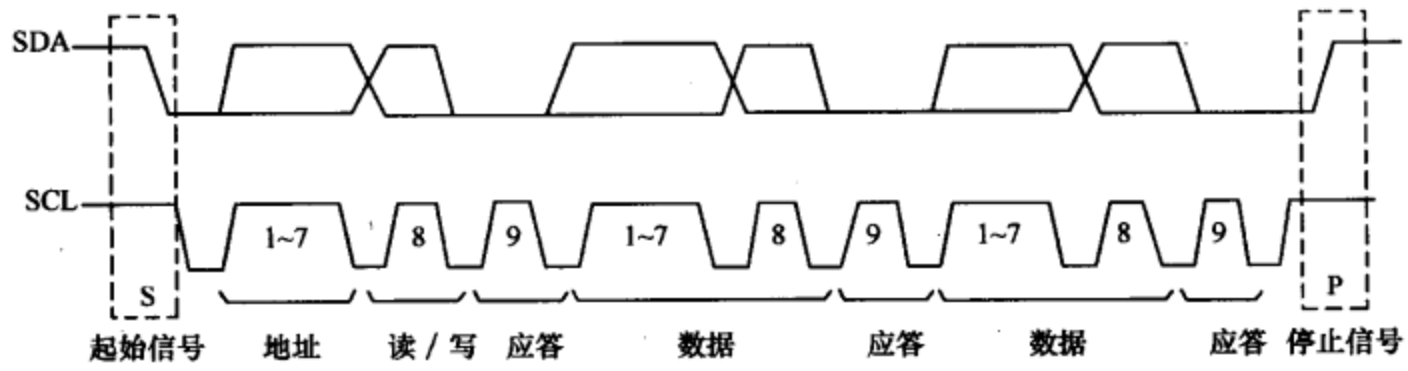


图 8-36 I²C 总线数据传输示意图

说明 当主控器接收数据时,在最后一个数据字节,必须发送一个非应答位,使受控器释放 SDA 线,以便主控器产生一个停止信号来终止总线数据传输。

3. I²C 总线数据的读写格式

总线上传送数据的格式是指:为被传送的各项有用数据安排的先后顺序,这种格式是根据串行通信的特点、传送数据的有效性、准确性和可靠性而制定的。另外,总线上数据的传送还是双向的,也就是说主控器在指令操纵下,既能向受控器发送数据(写入),也能接收受控器中某寄存器内存放的数据(读取),所以传送数据的格式有“写格式”与“读格式”之分。

(1) 写格式。I²C 总线数据的写格式如图 8-37 所示。

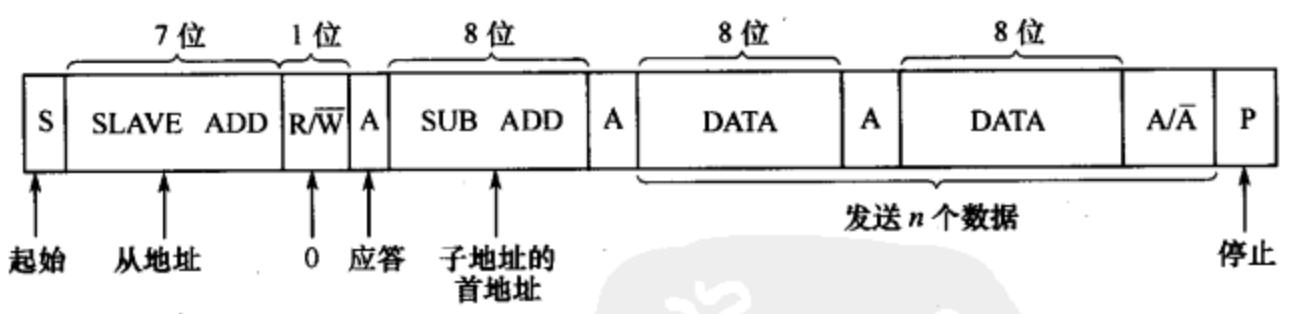


图 8-37 I²C 总线数据的写格式

写格式是指主控器向受控器发送数据,工作过程是:先由主控器发出启动信号(S),随后传送一个带读/写(R/W)标记的从地址(SLAVE ADD)字节,从地址只有 7 位长,第 8 位是“读/写”位(R/W),用来确定数据传送的方向。对于写格式,R/W 应为 0,表示主控器将发送数据给受控器,接着传送第 2B,即从地址的子地址(SUB ADD),若受控器有多字节的控制项目,该子地址是指首(第 1 个)地址,因为子地址在受控器中都是按顺序编制的,这就便于某受控器的数据一次传送完毕;接着才是若干字节的控制数据的传送,每传送 1B 的地址或数据后的第 9 位是受控器的应答信号,数据传送的顺序要靠主控器中程序的支持才能实现,数据发送完毕后,由主控器发出停止信号(P)。

(2)读格式。I²C 总线数据的读格式如图 8-38 所示。

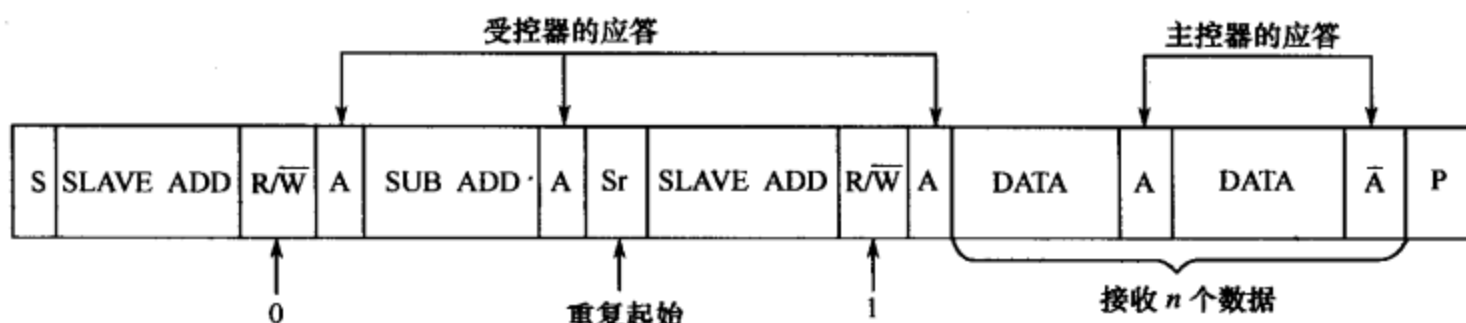


图 8-38 I²C 总线数据的读格式

与写格式不同,读格式首先要找到读取数据的受控器的地址,包括从地址和子地址,所以格式中在启动读之前,用写格式发送受控器,再启动读格式,不过前 3 个应答信号因为是指向受控器,所以应由受控器发出;然后,所有数据字节的应答信号因为是指向主控器,因此由主控器发出。不过最后的 $\bar{A}=1$ 。

重点提示 在设置众多受控器中,为了将控制数据可靠地传送给指定的受控 IC,必须使每一块 IC 编制一个地址码,通常称为从地址,显然从地址不能在不同的 IC 间重复使用。主控器发送寻址字节时,总线上所有受控器都将寻址字节中的 7 位地址与自己的从地址相比较,如果两者相同,则该器件就是被寻址的受控器(从器件),受控器内部的 n 个数据地址(子地址)的首地址由子地址数据字节指出,I²C 总线接口内部具有子地址指针自动加 1 功能,所以主控器不必一一发送 n 个数据字节的子地址。

二、I²C 总线串行存储器 AT24Cxx

一个单片机系统中,存储器起着非常重要的作用。常用的 89C51 或 52 系列单片机有 128B 或 256B 的内部数据存储区(RAM),在一个规模较小、数据量不大的应用系统中,可不必外部扩充数据存储区,单片机内部数据存储区的容量已经够用。当一个单片机应用系统要处理的数据不仅很多而且又非常重要时,通常的做法是通过单片机总线外部扩充数据存储芯片,早期的数据存储芯片主要有 61xx、62xx 系列,如 62256 便有 32KB 容量。这些芯片虽然与单片机接口方便,但也有自身的缺陷:系统掉电后保存在数据存储区内部的数据会丢失,对于某些对数据要求非常严格的系统而言,这个问题往往是致命的。近年来,随着半导体技术的不断发展,陆续出现了一些新的数据存储芯片,比较典型的有:基于 I²C 总线接口的 24 系列,基于 SPI 总线的 25 系列,以及并行总线接口的 28 系列、29 系列和内部带锂电池的 NVRAM 等新型数据存储芯片。这些芯片的共同特点是:芯片掉电后数据不会丢失,数据往往可以保存几年甚至几十年。一般情况下,如果系统要求数据处理实时性高,并且容量较大,可以采用并行总线协议接口的芯片,如 28、29 系列,或者是新型的 NVRAM 和铁电 FRAM 存储芯片。如果系统要求数据处理的实时性较低,也可采用串行总线协议的存储芯片,典型的有 24xx 和 25xx 系列的芯片。这些芯片一般采用 I²C、SPI 总线协议,与单片机接口通常仅占用 2 个~4 个 I/O 接口,可以最大限度地节省单片机的资源,并且数据可以反复擦写,这些芯片自身的功耗也较低(μA 级),并且价格便宜,因此近年来在数据处理中已得到广泛的应用。下面以比较典型的 I²C 总线结构的 AT24Cxx 为例,介绍具有 I²C 串行总线结构的数据存储芯片的基本应用。

1. I²C 总线串行存储器的结构

AT24Cxx 是美国 ATMEL 公司的串行 EEPROM 芯片,容量分别为 128×8 位和 256×8 位,该芯片具有页写功能,数据保存周期达 100 年。存储器容量与型号有关,如:AT24C02 为 256B(又称 1 页);AT24C04 为 512B(2 页);AT24C08 为 1024B(4 页)以及 AT24C16 为 2048B(8 页)。AT24Cxx 的管脚排列如图 8-39 所示。其中 A0、A1、A2 为器件地址选择线,SDA 为串行数据线,SCL 为串行时钟线,WP(EN)为写保护端(当该端为高电平时,不可对存储器写操作),V_{CC}为 1.8V~5.5V 正电压,V_{SS}为地。

2. I²C 总线串行存储器的从地址

AT24Cxx 从地址设置如图 8-40 所示。

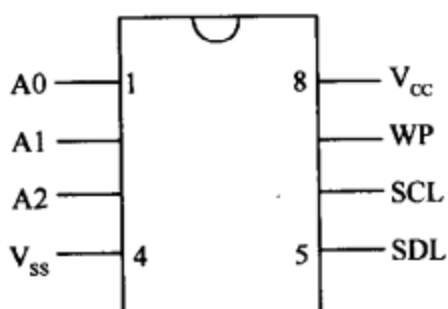


图 8-39 AT24Cxx 的管脚排列

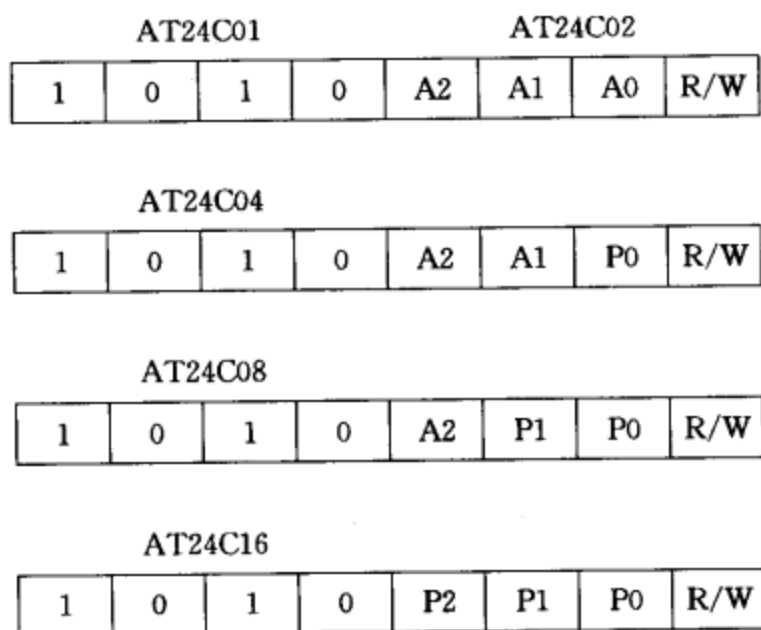


图 8-40 AT24Cxx 从地址设置

从图 8-40 中可以看出,AT24Cxx 的从地址由 7 位地址和 1 位方向位组成,其中,高 4 位 1010 为器件地址,由 I²C 委员会分配,最低 1 位 R/W 为方向位。当 R/W=0 时,对存储器进行写操作;当 R/W=1 时,对存储器进行读操作。其他 3 位为硬地址位,可选择接地、接 V_{CC}或悬空。

对于容量只有 128B 或 256B 的 AT24C01/02 而言,A2、A1、A0 为硬地址,可选择接地或 V_{CC}。当选择接地时,则该存储器的写从地址为 101000000(十六进制为 0A0H),读从地址为 10100001(十六进制为 0A1H)。

对于容量具有 512B 的 AT24C04 而言,硬地址是 A2、A1,其中 A0 悬空,划规页地址给 P0 使用:读/写第 0 页的 256B 子地址时,其从地址应赋予 P0=0;读/写第 1 页的 256B 子地址时,其从地址应赋予 P0=1。因为 8 位子地址只能寻址 256B,可见,当 A0 悬空时,可对 512B 进行寻址。若 A0 接地,其子地址只能在第 0 页(256B)中寻址,这说明,尽管 AT24C04 的容量为 512B,但第 1 页的存储容量被放弃。

对于 AT24C08,A1、A0 应选择悬空,对于 AT24C16,A0、A1、A2 应选择悬空,只有这样,才能充分利用其内部地址单元。

另一方面,若 A2、A1、A0 未悬空,可以任选接地或接 V_{CC},这样,A2、A1、A0 就有 8 种不同的选择,说明一对总线系统最多可以同时连接 8 块 AT24C02 而不发生地址冲突。不言而喻,一对总线最多也可以同时连接 4 块 AT24C04 或者 2 块 AT24C08,因为可以选

择的硬地址位分别有 A2、A1 或 A2。不过这种使用多块存储器的方法在单片机设计中很少采用。

重点提示 通常 EEPROM 写入时,总需要一定的写入时间(5ms~15ms),因此,在写入程序中无法连续写入多个数据字节,为解决连续写入多个数据字节,常设置页写功能,即在 EEPROM 器件中设有一定容量(页写)的数据寄存器,用户一次写入 EEPROM 的数据字节不大于页写字节数时,可按通常 RAM 的写入速度,装载至 EEPROM 中的数据寄存器中,随后启动自动写入定时控制逻辑,经过 5ms~10ms 时间,自动将数据寄存器中的数据同步写入 EEPROM 的指定单元中。这样一来,只要一次写入的字节数不多于页写容量,总线对 EEPROM 的操作可视为对静态 RAM 的操作,只要求下次数据操作在 5ms~10ms 时间之后进行。AT24C01/02/04/08/16 的页写字节数分别为 4B、8B、16B、16B、16B。

3. I²C 总线软件设计

PHILIPS 公司提供了标准的 I²C 总线状态处理软件包,并要求系统主从器件都具有 I²C 总线接口,这对于 PHILIPS 公司的单片机,如 87LPC76x 系列而言,通过这个软件包去处理 I²C 器件是比较容易的。但是目前其他公司的绝大多数单片机并不具有 I²C 总线接口,如 89C51、78E51 等,这时可以采用普通 I/O 接口模拟 I²C 总线的工作方式来实现 I²C 总线上主控制器对从器件的读、写操作,软件编写只要符合 I²C 总线数据传输的时序要求即可,启动、停止、发送应答和发送非应答信号时序要求如图 8-41 所示。

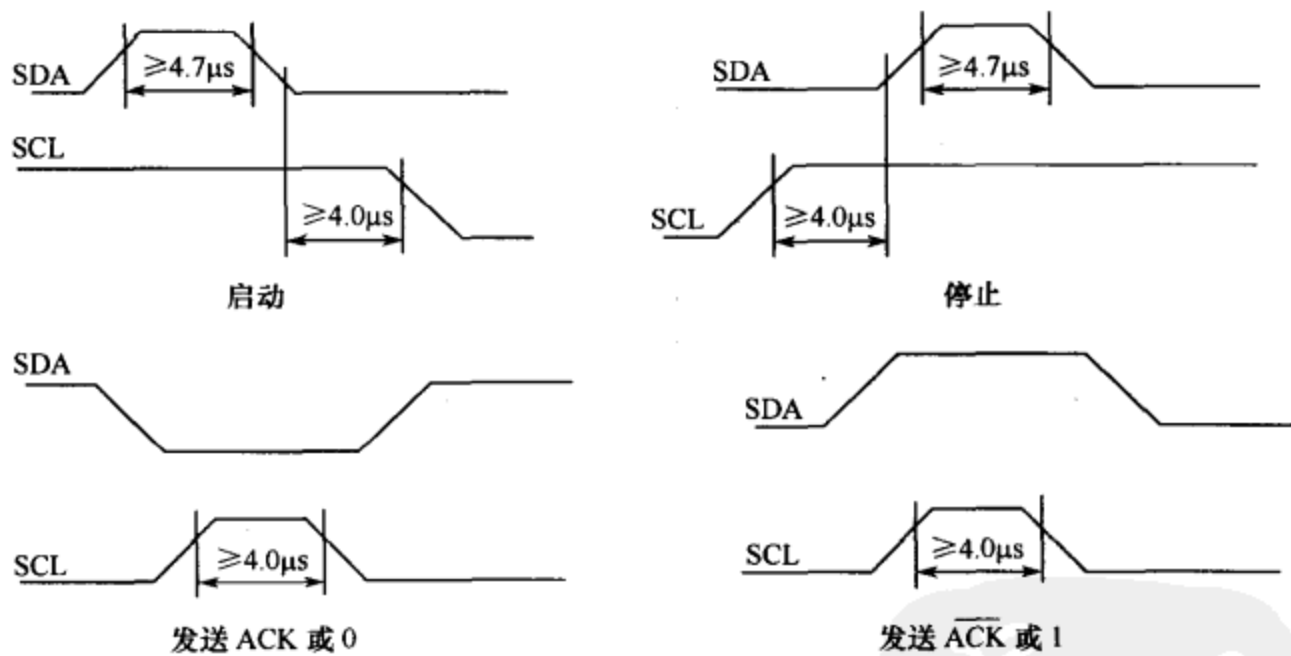


图 8-41 串行 EEPROM 时序规范

下面以 AT24C01 为例,介绍 I²C 总线器件软件的实现方法。以下程序中,要求单片机频率不高于 12MHz(1 个机器周期为 1μs),若频率高于 12MHz,则要相应地增加 NOP 指令数。

以下程序要使用单片机的 R0、R1、R2、R3、F0 及 C 等资源。程序中有许多符号标记,使用者必须了解,并在源程序开头进行定义。这些符号如下:

SDA——虚拟 I²C 总线的数据线;

SCL——虚拟 I²C 总线的时钟线;

SLA——器件从地址；

SUBA——器件子地址；

ACK——位变量,应答信号,这里规定 $ACK=0$ 表示有应答；

NUMBYT——传送字节数存放单元；

MTD——发送数据的缓冲区；

MRD——接收数据的缓冲区。

使用以下子程序时,使用者应根据实际情况,定义这些符号所指的单元值。

1) 启动开始信号子程序

启动条件是:在 SCL 是高电平时,SDA 线从高电平向低电平切换?

START: SETB SDA

SETB SCL ;起始条件建立时间大于 $4.7\mu s$

NOP

NOP

NOP

NOP

CLR SDA

NOP ;起始条件锁定时间大于 $4.7\mu s$

NOP

NOP

NOP

CLR SCL ;钳住总线,准备发送数据

NOP

RET

2) 停止 I²C 信号子程序

停止条件是:在 SCL 是高电平时,SDA 线从低电平向高电平切换?

STOP: CLR SDA

SETB SCL

NOP ;结束总线时间大于 $4\mu s$

NOP

NOP

NOP

SETB SDA

NOP ;保证终止信号和起始信号的空闲时间大于 $4.7\mu s$

NOP

NOP

NOP

CLR SDA

CLR SCL

RET

3) 发送应答位子程序

受控器件必须在收到每个数据字节后作出响应,要求发送器在第 9 个时钟脉冲位上释放 SDA 线,以便受控器送出应答信号,将 SDA 线拉成低电平,表示对接收数据的认可,应答信号用 ACK 或 A 表示,这里规定用 ACK=0 表示有应答?

```

MACK: CLR SDA      ;将 SDA 置 0
      SETB SCL      ;保持数据时间,即 SCL 为高,时间大于 4.7μs
      NOP
      NOP
      NOP
      NOP
      CLR SCL
      SETB SDA
      RET

```

4) 发送非应答位子程序

```

MNACK: SETB SDA     ;将 SDA 置 1
      SETB SCL      ;保持数据时间,即 SCL 为高,时间大于 4.7μs
      NOP
      NOP
      NOP
      NOP
      CLR SCL
      CLR SDA
      RET

```

5) 应答位检查子程序

```

CACK:  SETB SDA      ;应答位检查
      SETB SCL
      CLR ACK
      MOV C, SDA
      JNC CEND        ;若 SDA 为 0,有应答(此时 ACK=0),转 CEND 结束
      SETB ACK        ;若 SDA 为 1,无应答,定义 ACK=1
CEND:  CLR SCL
      RET

```

6) 发送一个数据字节的子程序

```

WRBYT: MOV R0, #08H   ;向 SDA 线上发送 1 个数据字节
WLP:   RLC A           ;循环左移
      JC WR1          ;C=1 转 WR1 处理程序,发送 1
      AJMP WR0        ;C=0 转 WR0 处理程序,发送 0
WLP1:  DJNZ R0, WLP    ;R0 不等于 0,转 WLP 继续发送
      RET

```

```

WR1:  SETB SDA
      SETB SCL
      NOP
      NOP
      NOP
      NOP
      CLR SCL
      CLR SDA
      AJMP WLP1

```

```

WRO:  CLR SDA
      SETB SCL
      NOP
      NOP
      NOP
      NOP
      CLR SCL
      AJMP WLP1

```

7)从 SDA 线上读取一个数据字节的子程序

```
RDBYT:MOV R0, #08H
```

```

RLP:  SETB SDA
      SETB SCL
      MOV C,SDA
      MOV A,R2
      RLC A
      MOV R2,A
      CLR SCL
      DJNZ R0,RLP
      RET

```

8)向器件指定子地址写数据

```
IWRNBYT:MOV R3,NUMBYT
```

```
      LCALL START      ;启动总线
```

```
      MOV A,SLA        ;发送从地址
```

```
      LCALL WRBYT
```

```
      LCALL CACK       ;检查应答位
```

```
      JB ACK,IWRNBYT   ;ACK=1 表示无应答,转 IWRNBYT 重新写
```

```
      MOV A,SUBA       ;ACK=0 表示有应答,发送子地址
```

```
      LCALL WRBYT
```

```
      LCALL CACK

```


	MOV R1, #MTD	;将 MTD 缓冲区中的数据送 R1
IWRDA:	MOV A, @R1	
	LCALL WRBYT	;开始写入数据
	LCALL CACK	
	JB ACK, IWRNBYT	; ACK = 1 表示无应答, 转 IWRNBYT 重新写
	INC R1	;ACK=0 表示有应答, R1 指针加 1, 指向下一单元
	DJNZ R3, IWRDA	;R3 不等于 0, 转 IWRDA
	LCALL STOP	;R3 等于 0, 结束
	RET	
9) 向器件指定子地址读取数据		
IRDNBYT:	MOV R3, NUMBYT	
	LCALL START	
	MOV A, SLA	
	LCALL WRBYT	;发送器件从地址
	LCALL CACK	
	JB ACK, IRDNBYT	;ACK=1 表示无应答, 转 IRDNBYT 重新读
	MOV A, SUBA	;ACK=0 表示有应答, 发送子地址
	LCALL WRBYT	
	LCALL CACK	
	LCALL START	;重新启动总线
	MOV A, SLA	
	INC A	;准备进行读操作
	LCALL WRBYT	
	LCALL CACK	
	JB ACK, IRDNBYT	; ACK = 1 表示无应答, 转 IRDNBYT 重新读
	MOV R1, #MRD	;ACK=0 表示有应答, 将缓冲区 MRD 中的数据送 R1
IRDNI:	LCALL RDBYT	;读操作开始
	MOV @R1, A	
	DJNZ R3, SACK	;R3 不等于 0, 转 SACK, 发送应答信号
	LCALL MNACK	;R3 等于 0, 发送发非应答信号
	LCALL STOP	;并结束总线
	RET	
SACK:	LCALL MACK	
	INC R1	
	SJMP IRDNI	

4. I²C 总线实验

实验 9 在下载型实验板上做以下实验:将数据 3FH、4FH 写入到 AT24C01 的 05H、06H 子地址中,断电片刻之后重新上电,再读出 05H、06H 子地址中的内容,观察是否为 3FH、4FH,以验证 AT24C01 的数据断电保存功能。

下载型实验板采用的串行存储器是 AT24C01,AT24C01 的 5 脚(SDA)与单片机的 P3.7 相连,AT24C01 的 6 脚(SCL)与单片机的 P3.6 相连,AT24C01 的 1 脚、2 脚、3 脚(A0、A1、A2)接地,因此,AT24C01 的写从地址为 0A0H,读从地址为 0A1H。

写 AT24C01 源程序如下:

```
SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0          ;应答标志位变量
MTD EQU 30H         ;发送数据缓冲区首址
MRD EQU 40H         ;接收数据缓冲区首址
AT24Cxx EQU 0A0H    ;定义器件写从地址为 0A0H
SLA EQU 50H         ;器件从地址变量
SUBA EQU 51H        ;器件子地址变量
NUMBYT EQU 52H      ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN:  MOV R4, #0F0H    ;延时,等待其他芯片复位
       DJNZ R4, $
       MOV MTD, #3FH
       MOV MTD+1, #4FH ;赋初值以便观察
WR24Cxx: MOV SLA, #0A0H ;指定器件地址
         MOV SUBA, #05H  ;指定子地址为 05H
         MOV NUMBYT, #2  ;写 2B 数据
         LCALL IWRNBYT
         AJMP WR24Cxx
;以下加入前面介绍的子程序 1)~9)
       :
       END
```

写程序实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 I2C_W.asm。对程序进行编译、链接和调试,产生 I2C_W.hex 目标文件。

(2)将 MON51 硬件仿真器和 PC 机连接好。

(4)取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

(5)点击菜单 Project→Option for Target‘Target’,在出现的窗口中设置为软件模拟

仿真(Use Simulator)。

(6)按 Ctr+F5 进入调试状态。

(7)按 F11 或 F10,进行单步运行和单步过程运行,运行后,打开存储器观察窗口,输入 D:30H,发现 30H、31H 的内容为 3F、4F,如图 8-42 所示。

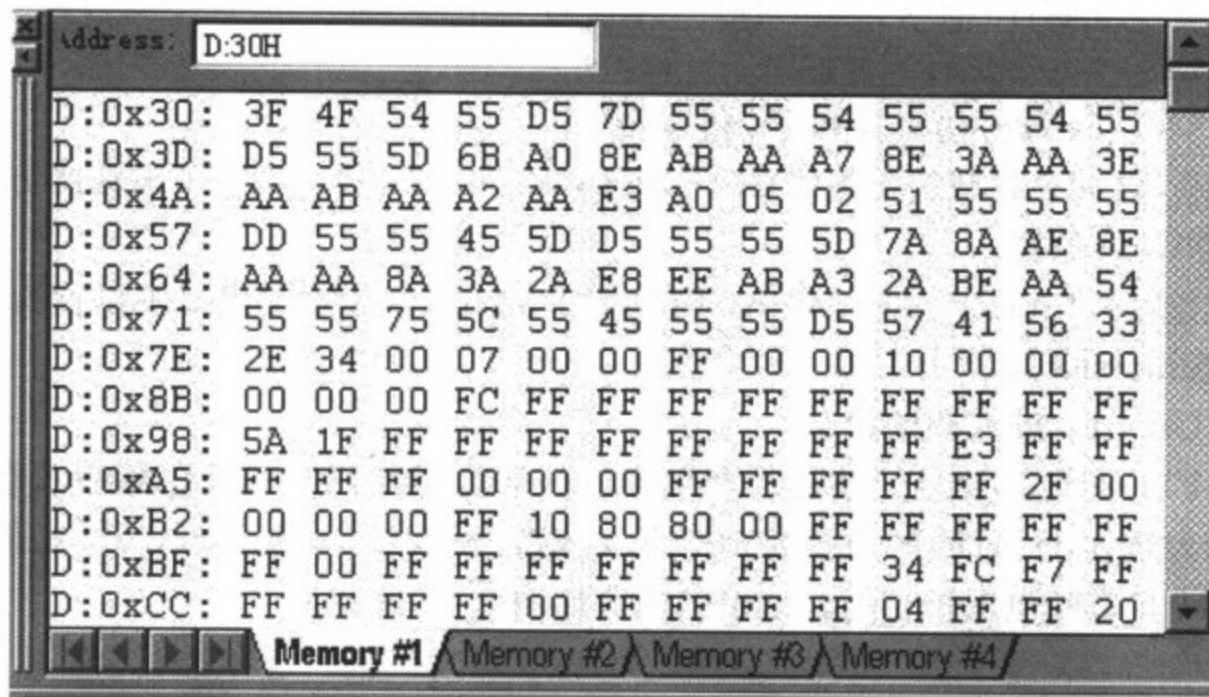


图 8-42 写入的内容

该实验程序在本书所附光盘的 example\ch_8\I2C_W 文件夹中。

读 AT24C01 源程序如下:

```
SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0          ;应答标志位变量
MTD EQU 30H         ;发送数据缓冲区首址
MRD EQU 40H         ;接收数据缓冲区首址
AT24 Cxx EQU 0A0H   ;定义器件写从地址为 0A0H
SLA EQU 50H         ;器件从地址变量
SUBA EQU 51H        ;器件子地址变量
NUMBYT EQU 52H      ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN: MOV R4, #0F0H  ;延时,等待其他芯片复位
      DJNZ R4, $
RD24Cxx:MOV SLA, #AT24Cxx
        MOV SUBA, #00H
        MOV NUMBYT, #2
        LCALL IRDNBYT
        AJMP WR24Cxx
```

;以下加入前面介绍的子程序 1)~9)

⋮

END

读程序实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 I2C_R.asm。对程序进行编译、链接和调试,产生 I2C_R.hex 目标文件。

(2)将 MON51 硬件仿真器和 PC 机连接好。

(4)断电,取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

(5)点击菜单 Project→Option for Target‘Target’,在出现的窗口中设置为软件模拟仿真(Use Simulator)。

(6)按 Ctr+F5 进入调试状态。

(7)按 F11 或 F10,进行单步运行和单步过程运行,运行后,打开存储器观察窗口,输入 D:30H,发现 40H、41H 的内容为 3F、4F,这就是 AT24C01 的 05H、06H 中的内容,说明 AT24C01 具有断电保护功能。读出的内容如图 8-43 所示。

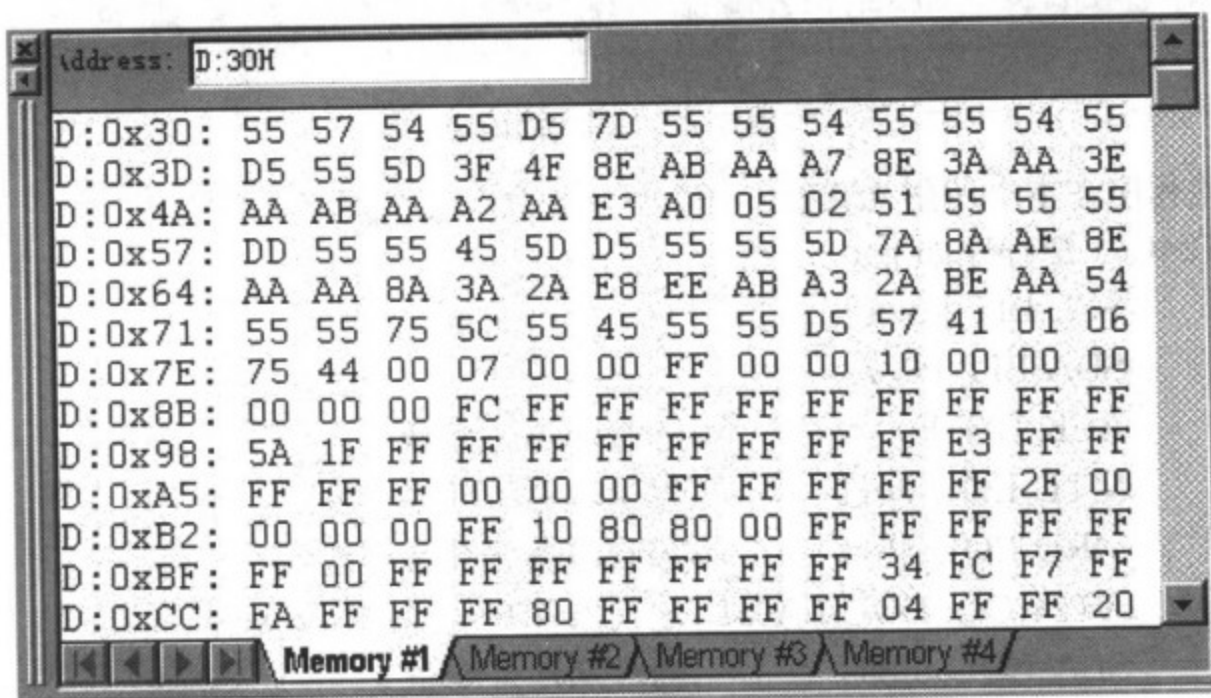


图 8-43 读出的内容

该实验程序在本书所附光盘的 example\ch_8\I2C_R 文件夹中。

三、I²C 总线时钟芯片 PCF8563

1. PCF8563 简介

PCF8563 是 PHILIPS 公司推出的一款带 I²C 总线具有极低功耗的多功能时钟/日历芯片,其写从地址为 0A2H,读从地址为 0A3H。它具有 4 种报警功能、定时功能和中断输出功能;内含时钟电路、振荡电路、低电压检测电路,不但使外围电路简洁,而且增加了芯片的可靠性。

PCF8563 的管脚排列如图 8-44 所示。

图 8-44 中,SCL 为 I²C 时钟输入端,数据随时钟信号同步输入器件或从器件输出;

SDA 为 I²C 数据 I/O 脚,用于串行数据的输入/输出;
 $\overline{\text{INT}}$ 是中断信号输出端,可通过设置报警寄存器按指
定时间在该脚产生报警信号,低电平有效;SDA、SCL、
 $\overline{\text{INT}}$ 均为漏极开路,必须上拉电阻;OSCI、OSCO 分别
为反相放大器的输入、输出端;可外接 32.768kHz 的
石英晶振,配置成片内振荡器。

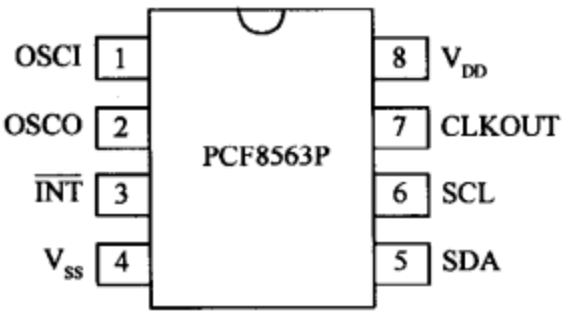


图 8-44 PCF8563 的管脚排列

2. PCF8563 的寄存器

PCF8563 有 16 个寄存器,16 个寄存器设计成可
寻址的 8 位并行寄存器,但不是所有位都有用,前两个寄存器(内存地址 00H、01H)为控
制/状态寄存器;内存地址 02H~08H 用于时钟计数器;内存地址 09H~0CH 用于报警寄
存器(定义报警条件);内存地址 0DH 用于控制 CLKOUT 管脚的输出频率;内存地址
0EH 和 0FH 分别用于定时器控制寄存器和定时器寄存器。各寄存器的结构和功能描述
如表 8-16~表 8-38 所列。

表 8-16 寄存器结构

地址	寄存器名称	D7	D6	D5	D4	D3	D2	D1	D0
00H	控制/状态寄存器 1	TEST1	0	STOP	0	TTESTC	0	0	0
01H	控制/状态寄存器 2	0	0	0	TI/TP	AF	TF	AIE	TIE
02H	秒寄存器	VL	00~59BCD 码格式数						
03H	分寄存器	—	00~59BCD 码格式数						
04H	时寄存器	—	00~23BCD 码格式数						
05H	日寄存器	—	00~31BCD 码格式数						
06H	星期寄存器	—	00~06BCD 码格式数						
07H	月/世纪寄存器	C	00~12BCD 码格式数						
08H	年寄存器		00~99BCD 码格式数						
09H	分钟报警寄存器	AE	00~59BCD 码格式数						
0AH	时钟报警寄存器	AE	00~23BCD 码格式数						
0BH	日报警寄存器	AE	01~31BCD 码格式数						
0CH	星期报警寄存器	AE	00~06BCD 码格式数						
0DH	CLKOUT 频率寄存器	FE	—	—	—	—	—	FD1	FD0
0EH	定时控制寄存器	TE	—	—	—	—	—	TD1	TD0
0FH	倒计时数值寄存器	定时器倒计数值							

表 8-17 控制/状态寄存器 1 位描述(00H)

位	符号	功 能
7	TEST1	TEST1=0 为普通模式;TEST1=1 为 EXT_CLK 测试模式
5	STOP	STOP=0 芯片时钟运行;STOP=1 所有芯片分频器异步置 0,芯片时钟停止运行(CLKOUT 在 32.768kHz 时可用)
3	TESTC	TESTC=0 电源复位功能失效; TESTC=1 电源复位功能有效
6,4,2,1,0	0	默认值为逻辑 0

表 8-18 控制/状态寄存器 2 位描述(01H)

位	符号	功 能
7,6,5	0	默认值置逻辑 0
4	TI/TP	TI/TP=0;当 TF 有效时,INT 有效(取决于 TIE 的状态)。 TI/TP=1;INT 脉冲有效,参见表 8-19(取决于 TIE 的状态)。 注意:若 AF 和 AIE 都有效时则 INT 一直有效
3	AF	当报警发生时,AF 被置 1;在定时器倒数计数结束时,TF 被置 1。它们在被软件重写前一直保持原有值。若定时器和报警中断都请求时,中断源由 AF 和 TF 决定。若要使清除一个标志位而防止另一标志位被重写,应运用逻辑指令 AND、标志位 AF 和 TF 值描述,参见表 8-20
2	TF	
1	AIE	标志位 AIE 和 TIE 决定一个中断的请求有效或无效,当 AF 或 TF 中一个为 1 时,中断是 AIE 和 TIE 都置 1 时的逻辑或。 AIE=0,报警中断无效;AIE=1,报警中断有效。 TIE=0,定时器中断无效;TIE=1,定时器中断有效
0	TIE	

表 8-19 INT 操作(BIT TI/TP=1)

源时钟频率/Hz	INT 周 期	
	$n^{\text{①}}=1$	$n>1$
4096	1/8192	1/4096
64	1/128	1/64
1	1/64	1/64
1/60	1/64	1/64

①n 为倒数计数定时器的数值,n 为 0 时定时器停止工作

表 8-20 AF 和 TF 值描述

R/W	BIT:AF		BIT:TF	
	值	功 能	值	功 能
R(读)	0	报警标志无效	0	定时器标志无效
	1	报警标志有效	1	定时器标志有效
W(写)	0	报警标志被清除	0	定时器标志被清除
	1	报警标志保持不变	1	定时器标志保持不变

表 8-21 秒 VL 寄存器位描述(02H)

位	符号	描 述
7	VL	VL=0:保证准确的时钟/日历数据
6~0	秒	代表 BCD 格式的当前秒数值,值为 00~59,例如 1011001 代表 59s

表 8-22 分寄存器位描述(03H)

位	符号	描 述
7	—	无效
6~0	分	代表 BCD 格式的当前分钟数值,值为 00~59

表 8-23 小时寄存器位描述(04H)

位	符号	描 述
7~6	—	无效
5~0	小时	代表 BCD 格式的当前小时数值,值为 00~23

表 8-24 日寄存器位描述(05H)

位	符号	描 述
7~6	—	无效
5~0	日	代表 BCD 格式的当前日数值,值为 01~31,当年计数器的值是闰年时,PCF8563 自动给 2 月增加一个值,使其为 29 天

表 8-25 星期寄存器位描述(06H)

位	符号	描 述
7~3	—	无效
2~0	星期	代表当前的星期数值 0~6,参见表 8-26。当年计数器的值是闰年时,PCF8563 自动给 2 月增加一个值,使其为 29 天

表 8-26 星期分配表

日(Day)	第 2 位	第 1 位	第 0 位	日(Day)	第 2 位	第 1 位	第 0 位
星期日	0	0	0	星期四	1	0	0
星期一	0	0	1	星期五	1	0	1
星期二	0	1	0	星期六	1	1	0
星期三	0	1	1	星期日	0	0	0

表 8-27 月/世纪寄存器位描述(07H)

位	符号	功 能
7	C	世纪位:C=0,指定世纪数为 20××;C=1,指定世纪数为 19××(××为年寄存器中的值,参见表 8-29)。当年寄存器中的值由 99 变为 00 时,世纪位会改变
6~5	—	无效
4~0	月	代表 BCD 格式的当前月份,值为 01~12,参见表 8-28

表 8-28 月分配表

月份	第 4 位	第 3 位	第 2 位	第 1 位	第 0 位	月份	第 4 位	第 3 位	第 2 位	第 1 位	第 0 位
1 月	0	0	0	0	1	7 月	0	0	1	1	1
2 月	0	0	0	1	0	8 月	0	1	0	0	0
3 月	0	0	0	1	1	9 月	0	1	0	0	1
4 月	0	0	1	0	0	10 月	1	0	0	0	0
5 月	0	0	1	0	1	11 月	1	0	0	0	1
6 月	0	0	1	1	0	12 月	1	0	0	1	0

表 8-29 年寄存器位描述(08H)

位	符号	功 能
7~0	年	代表 BCD 格式的当前年数值,值为 00~99

表 8-30 分报警寄存器位描述(09H)

位	符号	功 能
7	AE	AE=0,分报警有效;AE=1,分报警无效
6~0	分报警	代表 BCD 格式的分报警数值,值为 00~59

表 8-31 小时报警寄存器位描述(0AH)

位	符号	功 能
7	AE	AE=0,小时报警有效;AE=1,小时报警无效
6~0	小时报警	代表 BCD 格式的小时报警数值,值为 00~23

表 8-32 日报警寄存器位描述(0BH)

位	符号	功 能
7	AE	AE=0,日报警有效;AE=1,日报警无效
6~0	日报警	代表 BCD 格式的日报警数值,值为 01~31

表 8-33 星期报警寄存器位描述(0CH)

位	符号	功 能
7	AE	AE=0,星期报警有效;AE=1,星期报警无效
6~0	星期报警	代表 BCD 格式的星期报警数值,值为 0~6

表 8-34 CLKOUT 频率寄存器位描述(0DH)

位	符号	功 能
7	FE	FE=0,CLKOUT 输出被禁止,并设置成高阻抗; FE=1,CLKOUT 输出有效
6~2	—	无效
1 0	FD1 FD0	用于控制 CLKOUT 的频率输出引脚,参见表 8-35

表 8-35 CLKOUT 频率选择表

FD1	FD0	CLKOUT 输出脚
0	0	32.768kHz
0	1	1024kHz
1	0	32Hz
1	1	1Hz

表 8-36 定时器控制寄存器位描述(0EH)

位	符号	功 能
7	TE	TE=0,定时器无效;TE=1,定时器有效
6~2	—	无效
1 0	TD1 TD0	定时器时钟频率选择位,决定倒数定时器的时钟频率,参见表 8-37。不用时,TD1 和 TD0 设为 11(1/60Hz),以降低电源损耗

表 8-37 定时器时钟频率选择

TD1	TD0	定时器时钟频率/Hz
0	0	4096
0	1	64
1	0	1
1	1	1/60

表 8-38 定时器倒计数值寄存器位描述(0FH)

位	符号	功 能
7~0	定时器倒计数值	倒计数值为 n 倒计数周期= n /时钟频率

3. PCF8563 功能说明

(1)报警功能模式。一个或多个报警寄存器 AE(报警使能位)清 0 时,相应的报警条件有效。这样,一个报警将在每分钟至每星期范围内产生一次。设置报警标志位 AF(控制状态寄存器 2 的位 3),用于产生中断,AF 只可以用软件清除。

(2)定时器。8 位的倒计数器(地址 0FH)由定时器控制寄存器(地址 0EH)控制,定时器控制寄存器用于设定定时器的频率(4096Hz、64Hz、1Hz 或 $\frac{1}{60}$ Hz)以及设定定时器有效或无效,定时器从软件设置的 8 位二进制数倒数,每次倒数结束,定时器设置标志位 TF。定时器标志位 TF 只可以用软件清除,TF 用于产生一个中断INT,每个倒数周期产生一个脉冲,作为中断信号。TI/TP 控制中断产生的条件,当读定时器时,返回当前倒计数的数值。

(3)CLKOUT 输出。管脚 CLKOUT 可以输出可编程的方波,CLKOUT 频率寄存器(地址 0DH)决定方波的频率,CLKOUT 可以输出 32.768kHz(默认值)、1024Hz、32Hz、1Hz 的方波,CLKOUT 为漏极开路输出,管脚上电时输出有效,无效时输出为高阻抗。

(4)复位。PCF8563 包含一个片内复位电路,当振荡器停止工作时,复位电路开始工

作,在复位状态下,I²C 总线初始化,寄存器 TF、VL、TD1、TD0、TESTC、AE 位被置逻辑 1,其他的寄存器和地址指针被清 0。

(5)掉电检测器和时钟监控。PCF8563 内嵌掉电检测器,当 V_{DD} 低于 V_{LOW} 时,位 VL (秒寄存器的位 7)被置 1,用于指明可能产生不准确的时钟日历信息,VL 标志位只可以用软件清除。

4. PCF8563 的操作

1)读时钟

下面的程序将“秒”至“月”共 6 位时间信息读出,并放入 MRD 为首地址的接收缓冲区中,注意:时间读出后须屏蔽无效位,方能读出正确的信息。

```

SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0           ;应答标志位变量
MTD EQU 30H          ;发送数据缓冲区首址
MRD EQU 40H          ;接收数据缓冲区首址
PCF8563 EQU 0A2H      ;定义器件写从地址为 0A2H
SLA EQU 50H          ;器件从地址变量
SUBA EQU 51H         ;器件子地址变量
NUMBYT EQU 52H       ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN:  ACALL RCV8563
      NOP
      NOP
      SJMP MAIN
RCV8563: MOV SLA, #0A2H      ;取器件地址
      MOV SUBA, #02H        ;取读时间的首字节地址从秒开始读
      MOV NUMBYT, #07H      ;读 6 个时间信息
      LCALL IRDNBYT        ;读取时间并放入接收缓冲区中
      MOV A, MRD            ;取“秒”字节
      ANL A, #7FH          ;屏蔽无效位
      MOV MRD, A
      MOV A, MRD+1          ;取“分”字节
      ANL A, #7FH          ;屏蔽无效位
      MOV MRD+1, A
      MOV A, MRD+2          ;取“小时”字节
      ANL A, #3FH          ;屏蔽无效位
      MOV MRD+2, A
      MOV A, MRD+3          ;取“日”字节

```



```

ANL A, #3FH          ;屏蔽无效位
MOV MRD+3, A
MOV A, MRD+4          ;取“星期”字节
ANL A, #07H          ;屏蔽无效位
MOV MRD+4, A
MOV A, MRD+5          ;取“月”字节
ANL A, #1FH          ;屏蔽无效位
MOV MRD+5, A
RET

```

2)写时钟

下面的程序将 2005 年 5 月 1 日星期日下午 3:(15 点)59:30:的时间写入 PCF8563

```

SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0          ;应答标志位变量
MTD EQU 30H         ;发送数据缓冲区首址
MRD EQU 40H         ;接收数据缓冲区首址
PCF8563 EQU 0A2H    ;定义器件写从地址为 0A2H
SLA EQU 50H         ;器件从地址变量
SUBA EQU 51H        ;器件子地址变量
NUMBYT EQU 52H      ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN: ACALL SEND8563
      NOP
      NOP
      SJMP MAIN
SEND8563: ACALL LOAD8563 ;将时间装入发送缓冲区(MTD)中
          MOV SLA, #0A2H ;取器件地址
          MOV SUBA, #00H ;取写入寄存器的首字节地址从 00H 开始写
          MOV NUMBYT, #09H ;写 7 个时间信息和 2 个控制命令
          LCALL IWRNBYT
          RET
LOAD8563: MOV MTD, #00H ;寄存器 1,启动时钟
          MOV MTD+1, #1FH ;寄存器 2,设置报警及定时器中断,定时器中
                           ;断为脉冲形式
          MOV MTD+2, #30H ;将“秒”字节写入发送缓冲区中
          MOV MTD+3, #59H ;将“分字节”写入发送缓冲区中
          MOV MTD+4, #15H ;将“小时字节”写入发送缓冲区中

```

```

MOV MTD+5, #01H    ;将“日字节”写入发送缓冲区中
MOV MTD+6, #00H    ;将“星期字节”写入发送缓冲区中
MOV MTD+7, #05H    ;将“月字节”写入发送缓冲区中
MOV MTD+8, #05H    ;将“年字节”写入发送缓冲区中
RET

```

3) 报警功能的设置

PCF8563 共有 4 种报警方式,分别为小时报警(每小时的同一分时刻报警)、日报警(每天的同一小时时刻报警)、月报警(每月的同一日时刻报警)和星期报警(每星期的同一日时刻报警)。发生报警时,AF 位变为 1,设置报警有效的方法是将相应报警寄存器的最高位 AE 置 0。若同时置 AIE=1,则在 AF 置 1 的同时将在 $\overline{\text{INT}}$ 引脚产生一个中断,低电平有效,清除中断信号的方法是用软件清 AF。由此看出,AIE 相当于单片机中的中断允许控制位,而 AF 相当于中断申请标志位。

例 1 PCF8563 在每小时的 30 分时产生报警,并在 $\overline{\text{INT}}$ 端产生一个中断给单片机。

源程序如下:

```

SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0          ;应答标志位变量
MTD EQU 30H         ;发送数据缓冲区首址
MRD EQU V40H        ;接收数据缓冲区首址
PCF8563 EQU 0A2H    ;定义器件写从地址为 0A2H
SLA EQU 50H         ;器件从地址变量
SUBA EQU 51H        ;器件子地址变量
NUMBYT EQU 52H      ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
;以下是取原控制信息子程序
MAIN: MOV SLA, #0A2H ;取器件地址
      MOV SUBA, #01H ;取中断控制字节地址
      MOV NUMBYT, #01H
      LCALL IRDNBYT ;读中断控制字节信息
      NOP
;以下为中断配置子程序
      MOV A, MRD
      ORL A, #02H    ;置 AIE=1
      MOV MTD, A
      MOV SUBA, #01H ;取中断控制字节地址
      MOV NUMBYT, #01H

```

```
LCALL IWRNBYT    ;送中断控制字节命令
NOP
```

;以下是报警配置子程序

```
MOV MTD, #30H    ;30分报警时刻送发送缓冲区(最高位 AE 为 0,报警有效)
MOV SUBA, #09H   ;取分报警控制字节地址
MOV NUMBYT, #01H
LCALL IWRNBYT    ;送报警信息
SJMP $
```

以上配置完成后,即可在 $\overline{\text{INT}}$ 脚产生中断信号,在软件清除 AF 位之前,该中断信号一直有效,清除中断信号的程序如下:

```
SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0        ;应答标志位变量
MTD EQU 30H       ;发送数据缓冲区首址
MRD EQU 40H       ;接收数据缓冲区首址
PCF8563 EQU 0A2H  ;定义器件写从地址为 0A2H
SLA EQU 50H       ;器件从地址变量
SUBA EQU 51H      ;器件子地址变量
NUMBYT EQU 52H    ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
```

;以下是取原控制信息子程序

```
MAIN: MOV SLA, #0A2H ;取器件地址
      MOV SUBA, #01H ;取中断控制字节地址
      MOV NUMBYT, #01H
      LCALL IRDNBYT  ;读中断控制字节信息
      NOP
```

;以下是中断配置子程序

```
MOV A, MRD
ANL A, #17H      ;设置成 AF=0,但保持其他位不变
MOV MTD, A
MOV SUBA, #01H   ;取中断控制字节地址
MOV NUMBYT, #01H
LCALL IWRNBYT    ;送中断清除命令
SJMP $
```

4) 定时器功能的设置

PCF8563 的定时器为倒计时定时器。当 TE=1 时有效,倒计数值为 0FH 中的的二

进制数;当倒计数值计为 0 时,TF 位置 1。若同时置 TIE=1,则在 TF 置 1 的同时,将在 $\overline{\text{INT}}$ 引脚产生一个中断,低电平有效。与报警中断不同的是,定时器中断信号有两种方式:由 TI/TP 位控制,设置 TI/TP=0,中断信号和报警中断信号相同,均为低电平方式,置 TF=0,可清除中断信号;设置 TI/TP=1,则中断信号为脉冲方式,其脉冲低电平宽度约为 15ms,此时可不考虑 TF 位的影响。由此看出,TIE 相当于单片机中的定时中断允许控制位,而 TF 相当于定时中断申请标志位。需要说明的是,定时器功能可以和报警功能同时有效。

例 2 让 PCF8563 每秒产生一次报警,并在 $\overline{\text{INT}}$ 端产生一个脉冲给单片机,在中断服务程序中可以读取时钟以供显示。

源程序如下:

```

SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0           ;应答标志位变量
MTD EQU 30H          ;发送数据缓冲区首址
MRD EQU 40H          ;接收数据缓冲区首址
PCF8563 EQU 0A2H     ;定义器件写从地址为 0A2H
SLA EQU 50H          ;器件从地址变量
SUBA EQU 51H         ;器件子地址变量
NUMBYT EQU 52H       ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
;以下是取原控制信息子程序
MAIN:MOV SLA, #0A2H   ;取器件地址
      MOV SUBA, #01H   ;取中断控制字节地址
      MOV NUMBYT, #01H
      LCALL IRDNBYT    ;读中断控制字节信息
;以下是中断配置子程序
      MOV A, MRD
      ORL A, #01H
      MOV MTD, A
      MOV SUBA, #01H   ;取中断控制字节地址
      MOV NUMBYT, #1
      LCALL IWRNBYT    ;送中断控制字节命令
;以下是定时配置子程序
      MOV MTD, #81H    ;启动定时器命令及时钟频率(64Hz)送发送缓冲区
      MOV MTD+1, #64   ;倒计数值为 64
      MOV SUBA, #0EH   ;取定时器控制字节首地址

```

```

MOV NUMBYT, #02H    ;写两个字节
LCALL IWRNBYT        ;写 PCF8563
SJMP $

```

以上配置完成后,即可在 $\overline{\text{INT}}$ 脚产生周期为 1s 的脉冲,清除脉冲中断的方法有 3 种,即将 TIE、TE 或 0FH 寄存器三者中任一的内容清 0 即可。

例 3 在 PCF8563 的 CLKOUT 脚输出一 32.768kHz 的方波。

源程序如下:

```

SCL BIT P3.6
SDA BIT P3.7
ACK BIT F0                ;应答标志位变量
MTD EQU 30H               ;发送数据缓冲区首址
MRD EQU 40H               ;接收数据缓冲区首址
PCF8563 EQU 0A2H          ;定义器件写从地址为 0A2H
SLA EQU 50H               ;器件从地址变量
SUBA EQU 51H              ;器件子地址变量
NUMBYT EQU 52H            ;读/写的字节数变量
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN: MOV MTD, #80H        ;时钟输出使能命令及 32.768kHz 频率选择
                                送发送缓冲区
MOV SLA, #0A2H
MOV SUBA, #0DH             ;取时钟输出控制字节地址
MOV NUMBYT, #01H          ;写一个字节
LCALL IWRNBYT              ;开始时钟输出
SIMP $

```

第六节 红外遥控接口

红外线遥控是目前使用最广泛的一种通信和遥控手段。由于红外线遥控装置具有体积小、功耗低、功能强、成本低等特点,因而,继彩电、录像机之后,在空调机以及玩具等其他小型电器装置上也纷纷采用红外线遥控。工业设备中,在高压、辐射、有毒气体、粉尘等环境下,采用红外线遥控不仅安全可靠,而且能有效地隔离电气干扰。

一、红外遥控系统

通用红外遥控系统由发射和接收两大部分组成,应用编/解码专用集成电路芯片来进行控制操作,如图 8-45 所示。

发射部分包括键盘矩阵、编码调制、LED 红外发送器;接收部分包括光/电转换放大器、解调、解码电路。

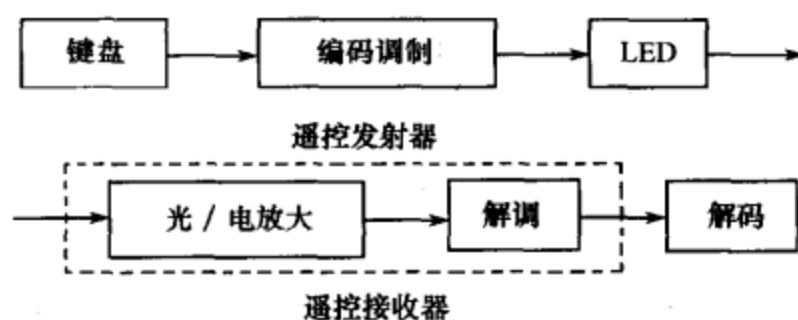


图 8-45 红外遥控系统框图

二、红外遥控的编码与解码

1. 遥控编码

遥控发射器专用芯片很多,根据编码格式可以分成脉冲宽度调制和脉冲相位调制两大类,这里我们以运用比较广泛、解码比较容易的脉冲宽度调制来加以说明,现以 LC7461 组成发射电路为例说明编码原理。当发射器按键按下后,LC7461 即有遥控编码发出,所按的键不同,遥控编码也不同。LC7461 输出的红外遥控编码是由一个引导码、26 位用户码(13 位用户码和 13 位用户反码)和 16 位键数据码(8 位数据码和 8 位数据反码)组成,如图 8-46 所示。

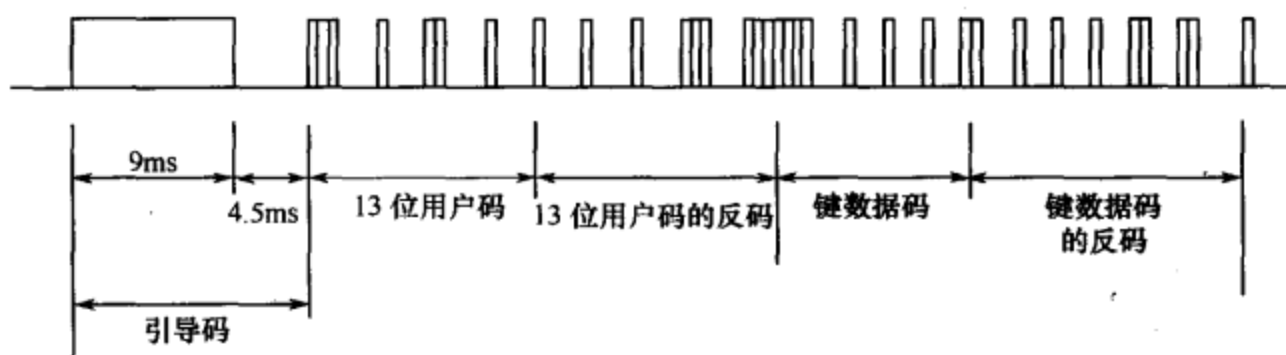


图 8-46 LC7461 输出的红外码

LC7461 输出的红外编码经过一个三极管反相驱动后向外发射出去,因此,遥控器发射的红外编码与图 8-46 的红外码反相,即高电平变为低电平,低电平变为高电平。

遥控器发射的红外遥控码具有以下特征:

当遥控器上任意一个按键按下超过 36ms 时,LC7461 芯片的振荡器使芯片激活,将发射一个特定的同步码头,遥控器发射的引导码是一个 9ms 的低电平和一个 4.5ms 的高电平,这个同步码头使程序知道从这个同步码头以后可以开始接收数据。

引导码之后是用户码,用户码能区别不同的红外遥控设备,防止不同机种遥控码互相干扰。用户码采用脉宽调制,以脉宽为 0.56ms、间隔 0.565ms、周期为 1.125ms 的组合表示二进制的 0;以脉宽为 1.685ms、间隔 0.565ms、周期为 2.25ms 的组合表示二进制的 1,如图 8-47 所示。

最后 16 位为 8 位的操作码和 8 位的操作反码,用于核对数据是否接收准确。

上述 0 和 1 组成的 42 位二进制码经 38kHz 的载频进行二次调制,以提高发射效率,达到降低电源功耗的目的。然后再通过红外发射二极管产生红外线向空间发射。

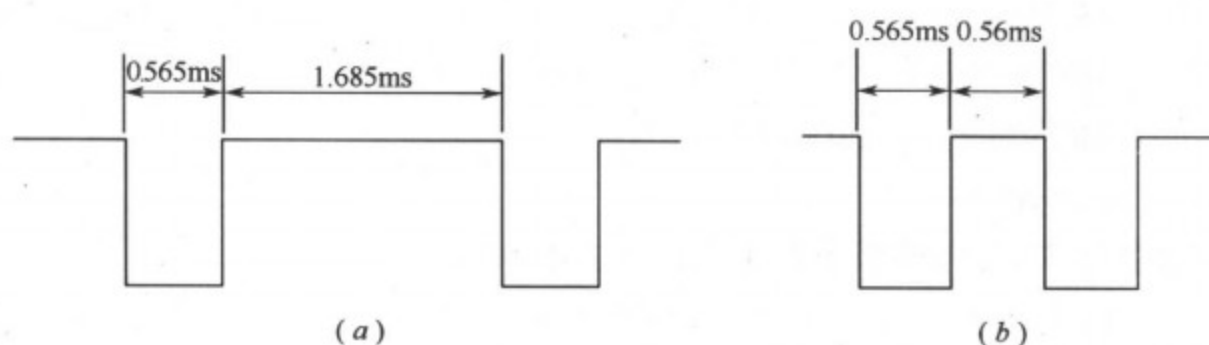


图 8-47 编码 0 和编码 1
(a)编码 1; (b)编码 0。

2. 遥控解码

解码的关键是如何识别 0 和 1,从位的定义可以发现,0、1 均以 0.565ms 的低电平开始,不同的是高电平的宽度不同,0 为 0.56ms,1 为 1.685ms,所以,必须根据高电平的宽度区别 0 和 1。如果从 0.565ms 低电平过后开始延时,0.56ms 以后,若读到的电平为低,说明该位为 0;反之则为 1。为了可靠起见,延时必须比 0.56ms 长些,但又不能超过 1.12ms,否则如果该位为 0,读到的已是下一位的高电平,因此取 $(1.12\text{ms} + 0.56\text{ms})/2 = 0.84\text{ms}$ 最为可靠,一般取 0.8ms~0.9ms 即可。

另外,根据红外编码的格式,程序应该等待 9ms 的起始码和 4.5ms 的结束码完成后才能读码。

三、红外遥控实验

实验 10 用 AT89C51 实验开发板做以下实验:按红外遥控器上不同的按键,实验板 P1 口的 8 个 LED 用不同的灯显示出来,在解码成功的同时,能发出“滴滴滴”的提示音。

AT89C51 实验开发板带有 32 按键的遥控器,如图 8-48 所示。

AT89C51 实验开发板设有红外接收头,接收头是塑封一体化红外线接收器,它是一种集红外线接收、放大、整形于一体的集成电路,不需要任何外接元件,就能完成从红外线接收到输出与 TTL 电平信号兼容的所有工作,没有红外遥控信号时为高电平,收到红外信号时为低电平。红外接收头的输出接到单片机 AT89C51 的 P3.2 脚(外中断 0)。

实验的源程序如下:

```

ORG 0000H
AJMP MAIN           ;转入主程序
ORG 0003H           ;外部中断 P3.2 脚 INT0 入口地址
AJMP INT            ;转入外部中断服务子程序(解码程序)
;以下为主程序

```



图 8-48 遥控器

```

MAIN:  SETB EA           ;打开 CPU 总中断请求
        SETB IT0        ;设定 INT0 的触发方式为脉冲负边沿触发
        SETB EX0        ;打开 INT0 中断请求
        MOV P2, #11100111B
        AJMP $

;以下为进入 P3.2 脚外部中断子程序,也就是解码程序
INT:    CLR EA           ;暂时关闭 CPU 的所有中断请求
        MOV R6, #10
SB:     ACALL DELAY1      ;调用 0.882ms 延时子程序
        JB P3.2, EXIT    ;延时 0.882ms 后,若 P3.2 为 1,转 EXIT 退出解
                        ;码程序
        DJNZ R6, SB      ;若 P3.2 为 0,重复 10 次,获得 9ms 的低电平引
                        ;导码
        JNB P3.2, $      ;P3.2 为 0,则等待
        ACALL DELAY2    ;若 P3.2 为 1,延时 4.74ms,以获得 4.5ms 的高电
                        ;平引导码
        MOV R7, #26     ;获得 26 位用户码
LOOP:   JNB P3.2, $      ;若 P3.2 为 0,则等待
        LCALL DELAY1    ;若 P3.2 为 1,延时 0.882ms
        MOV C, P3.2     ;将 P3.2 引脚此时的电平状态 0 或 1 存入 C 中
        JNC LOOP0       ;延时 0.882ms 后若 C 为 0,说明 P3.2 接收的是
                        ;0,转 LOOP0 处理程序
        LCALL DELAY3    ;若 C 为 1,说明 P3.2 接收的是 1,再延时 1ms 等
                        ;待低电平结束

LOOP0:  DJNZ R7, LOOP
        MOV R1, #1AH    ;设定 1AH 为起始 RAM 区
        MOV R2, #2      ;接收从 1AH 到 1BH 的两个 RAM 区,用于存放
                        ;操作码和操作反码

PP:     MOV R3, #8       ;每组数据为 8 位
OPR:    JNB P3.2, $      ;若 P3.2 为 0,则等待
        LCALL DELAY1    ;若 P3.2 为 1,延时 0.882ms
        MOV C, P3.2     ;将 P3.2 引脚此时的电平状态 0 或 1 存入 C 中
        JNC LOOP1       ;延时 0.882ms 后若 C 为 0,说明 P3.2 接收的是
                        ;0,转 LOOP1 处理程序
        LCALL DELAY3    ;若 C 为 1,说明 P3.2 接收的是 1,再延时 1ms 等
                        ;待低电平结束

LOOP1:  MOV A, @R1       ;将 R1 中的地址给 A
        RRC A           ;将 C 中的值 0 或 1 移入 A 中的最低位
        MOV @R1, A      ;将 A 中的数暂时存放在 R1 数值存放的寄存器中

```

DJNZ R3, OPR	;接收满 8 位换一个 RAM
INC R1	;对 R1 中的值加 1,换下一个 RAM
DJNZ R2, PP	;接收完 8 位数据码和 8 位数据反码,存放在 1AH/1BH 中
MOV A, 1AH	
CPL A	;对 1AH 取反后和 1BH 比较
CJNE A, 1BH, EXIT	;如果不等表示接收数据发生错误,放弃
MOV P1, 1AH	;将按键的键值通过 P1 口的 8 个 LED 显示出来
CLR P2. 5	;蜂鸣器发出“滴滴滴”的声音,表示解码成功
LCALL DELAY2	
LCALL DELAY2	
LCALL DELAY2	
SETB P2. 5	;蜂鸣器停止
EXIT: SETB EA	;允许中断
RETI	;退出解码子程序
DELAY1: MOV R4, # 20	;延时子程序 1,精确延时 0. 882ms
D1: MOV R5, # 20	
DJNZ R5, \$	
DJNZ R4, D1	
RET	
DELAY2: MOV R4, # 10	;延时子程序 2,精确延时 0. 4740ms
D2: MOV R5, # 235	
DJNZ R5, \$	
DJNZ R4, D2	
RET	
DELAY3: MOV R4, # 2	;延时程序 3,精确延时 1ms
D3: MOV R5, # 248	
DJNZ R5, \$	
DJNZ R4, D3	
RET	
END	

实验步骤如下:

(1)打开 Keil 软件,输入上面的程序,保存为 Rmcl. asm。对程序进行编译、链接和调试,产生 Rmcl. hex 目标文件。

(2)先可用硬件仿真器对源程序进行仿真,然后用 RF-810 编程器对 AT89C51 芯片编程。

(3)将 AT89C51 芯片插入实验开发板,通电进行实验。

该实验程序在本书所附光盘的 example\ch_8\Rmcl 文件夹中。

实验 11 用 AT89C51 实验开发板制作一个红外线遥控器声光测试器,制作的测试

器可以方便地判断遥控器是否能发射红外信号,各个按键工作是否可靠。

AT89C51 实验开发板上有一个一体化红外接收器,它将接收到的红外信号输入到单片机的 P3.2,当接收到遥控信号时,P3.2 变低,据此,设计的程序如下:

```
ORG 0000H
START:MOV P0,#0FFH      ;开机初始化
      MOV P1,#0FFH
      MOV P2,#11100111B
      MOV P3,#0FFH
      JB P3.2,$          ;等待遥控信号出现
      MOV P1,#0
      MOV P2,#0
      JNB P3.2,$         ;如果是低电平就原地等待,如果出现高电平就退出
      AJMP START
      END
```

实验步骤同上。该实验程序在本书所附光盘的 example\ch_8\Rmc2 文件夹中。

第七节 语音接口

一、ISD1400 系列语音电路

ISD1400 系列目前应用十分广泛的单片不挥发录入语音集成电路,片内由时钟振荡器、128KB 的 EEPROM(电可编程可擦除只读存储器)、微音放大器、自动增益控制电路、抗干扰滤波器、差动功率放大器等高品质语音录放系统所需的全部基本功能电路。一个最小的录放系统仅由一个驻极体话筒、一个喇叭、两个按钮、一个电源和少量的电阻、电容组成。

1. ISD1400 系列电路的特点

和其他同类语音电路相比,ISD1400 系列具有以下特点:

所需外围元件少,电路简单,操作方便。

采用直接模拟量存储技术 DAST,再现优质原声。

采用 EEPROM 存储器,因此无需电池即能保存信息 10 年以上,可反复录放达 10 万次。

语音固化无需专用编程或开发装置。

具有自动省电模式,此时仅需 $0.5\mu\text{A}$ 的保持电流。

2. 管脚功能

目前,ISD1400 系列有下列型号:ISD1408、ISD1410、ISD1412、ISD1416、ISD1420。录放时间分别为 8s、10s、12s、16s、20s。

ISD1400 系列主要采用 28 脚 DIP 和 SOG(小型双列封装)塑料包装。另一种是标准 28 脚双烈直插式 COB 软包装,其性能指标与 DIP、SOG 包装相同,并可与 DIP 互换代用,其价格是 DIP、SOG 包装的一半,目前国内普遍使用 COB 包装,其型号规格与 ISD1400 系列对应,分别为 HY408、HY410、HY412、HY416、HY420。图 8-49 为是 ISD1420 管脚

排列图。

ISD1420 各管脚功能说明如下。

1 脚~6 脚、9 脚、10 脚: A0~A7, 当 A6、A7 至少有一位为 0 时, 输入认为是地址输入, 输入的地址被当做当前录音或放音的起始地址。

7 脚、8 脚、11 脚、22 脚: 空脚。

12 脚: V_{SSD} , 数字电路地。

13 脚: V_{SSA} , 模拟电路地。

14 脚、15 脚: SP+、SP-; 喇叭“+”、“-”极输出端。

16 脚: V_{CCA} , 模拟电路电源。

17 脚: MIC, 驻极体话筒输入。

18 脚: MIC REF, 驻极体话筒参考输入。

19 脚: AGC, 自动增益控制。

20 脚: ANA IN, 模拟量输入。

21 脚: ANA OUT, 模拟量输出。

23 脚: $\overline{\text{PLAYL}}$, 放音电平触发端, 低电平有效, 当该脚加低电平时放音。

24 脚: $\overline{\text{PLAYE}}$, 放音边沿触发端, 下降沿有效, 当该脚加一个下跳变信号, 电路进入放音状态。

25 脚: RECLED, 录音指示信号输出端, 当电路处于录音状态时, 该端内部被拉向低电平。通常该端接一个 LED 指示灯, 用做录音状态的指示。

26 脚: XCLK, 外接时钟输入端, 一般接地。

27 脚: $\overline{\text{REC}}$, 录音触发端, 该脚加低电平时进入录音状态, 回复高电平时录音停止。

3. 基本录放电路

图 8-50 是应用 ISD1420(HY420) 作为基本录放音的电路, 所有的地址线均设置为“0”, 所以录放音的起始地址是 0。当按下录音按钮 REC 时开始录音, 所得到的录音数据从存储器的 0 地址开始存放, 直到存储器满或者录音键松开; 当按下播放按钮 PLAYL 时, 则开始放音, 直到 PLAYL 松开或者存储器全部用完。LED 指示灯作录音指示使用, 当处于录音状态时, 25 脚内部被拉向低电位, 因此 LED 有电流流过而发光。语音信号由驻极体话筒 MIC 拾取, 从 MIC 和 MICREF 两端输入芯片内部的话筒放大器放大, 该放大器的输出信号从 ANA OUT 端引出, 外部使用耦合电容耦合到另一个放大器的输入端 ANA IN, 作进一步放大, 经功放后的音频信号从 SP+ 和 SP- 两端输出并推动扬声器发音。扬声器的接法也可以一端接地, 另一端任意接 SP+ 或 SP-, 扬声器的功率只宜在 0.25W 以内, 否则需外加功率放大器。

4. 分段方法

ISD1420(HY420) 根据输入端 A7~A0 的有效取值可以被划分为 160 个区, 共可录放 160 段语音信息, 每小段占时为 $20\text{s}/160=0.125\text{s}$ 。A7~A0 的设置只确定录放的起始地址, 对于录音, 其结束地址仅决定于实际录音状态的结束(即 REC 端回到高电平时), 在

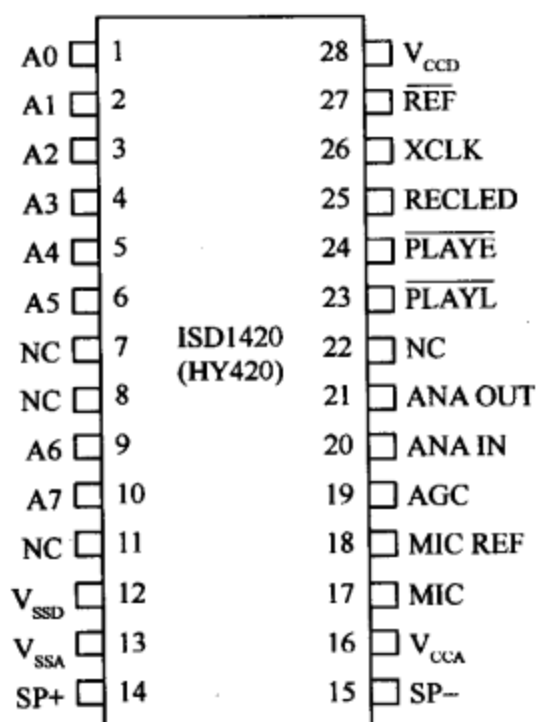


图 8-49 ISD1420 管脚排列图

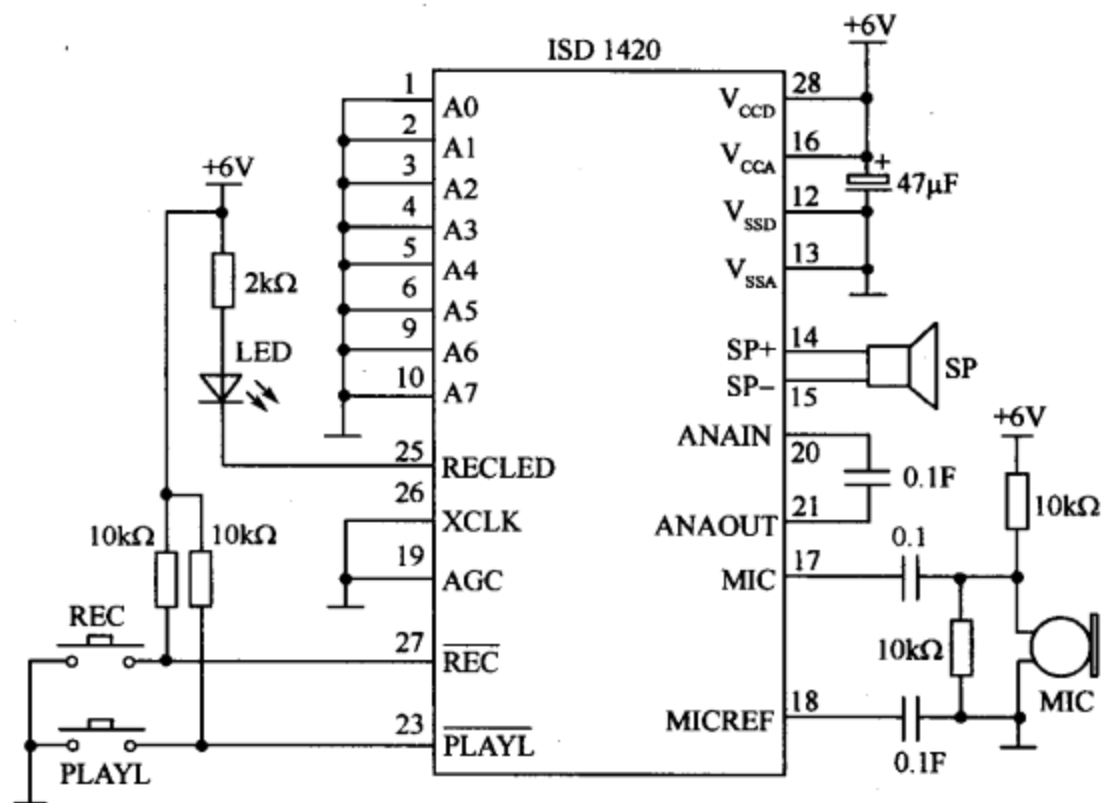


图 8-50 ISD1420 基本录放电路

录音状态结束时,存储区里能留下一个 EOM 标志,因此使用分段录音时,能产生许多个 EOM 标志;在放音时则从由 A0~A7 所确定的起始地址开始放音,当遇到 EOM 标志时自动停止(无外界因素干涉)。

图 8-51 是一个分段录放应用的电路,它把存储区 00~9FH 共分为 4 个区段,也就是把总共 20s 的时间分为 4 段使用,分别由 4 个按钮 KEY1~KEY4 来控制。由图可知,当 KEY1 被按下时,通过二极管把 A3~A7 全部接地,这时地址为 00(A0~A2 已固定接地),若转换开关 K 接向 REC 一边,则 REC 端也通过二极管被接向低电平,于是录音便从 00 地址开始;若 K 是打向 $\overline{\text{PLAYL}}$ 一边的,则 $\overline{\text{PLAYL}}$ 端通过二极管被拉向低电平,从 00 地址开始放音。

若 KEY2 被按下,则除 A4 仍保持高电平外,其余的 A3、A5~A7 都通过二极管接地,因此地址为 00010000,即 10H,同时 $\overline{\text{PLAYL}}$ 端或 $\overline{\text{REC}}$ 端也通过开关 K 及二极管被接地,因此放音或录音从地址 10H 开始。

同理,若 KEY3 被按下,只有 A5 保持高电平,地址设置为 00100000,即 20H,录放的起始地址是 20H;若 KEY4 被按下,只有 A6、A4 保持高电平,录放的起始地址是 01010000 即 50H。

由此可知,存储区分为 0~10H、10H~20H、20H~50H、50H~9FH 共 4 个区段,对应的录放时间是,第 1 段的起始时间为 0s;第 2 段的起始时间为 $0.125 \times 10H = 0.125s \times 16 = 2s$;第 3 段的起始时间为 $0.125 \times 20H = 0.125 \times 32 = 4s$;第 4 段的起始时间为 $0.125 \times 50H = 0.125 \times 80 = 10s$,录音时要求按住按键的时间(即录音时间)绝不能越界。

图 8-52 是 ISD1420 等分 4 段的应用电路,通过对开关 K1、K2 的设置,可以把存储区等分为 4 个区段,相应的时间段等分为每段 5s。K1 的“刀”与 A3、A5 相连,K2 的“刀”与 A4、A6 相连,通过将 K1、K2 接向高电平或低电平,可以设置地址线为不同的状态,各段的起始地址与时间分配如表 8-39 所列。

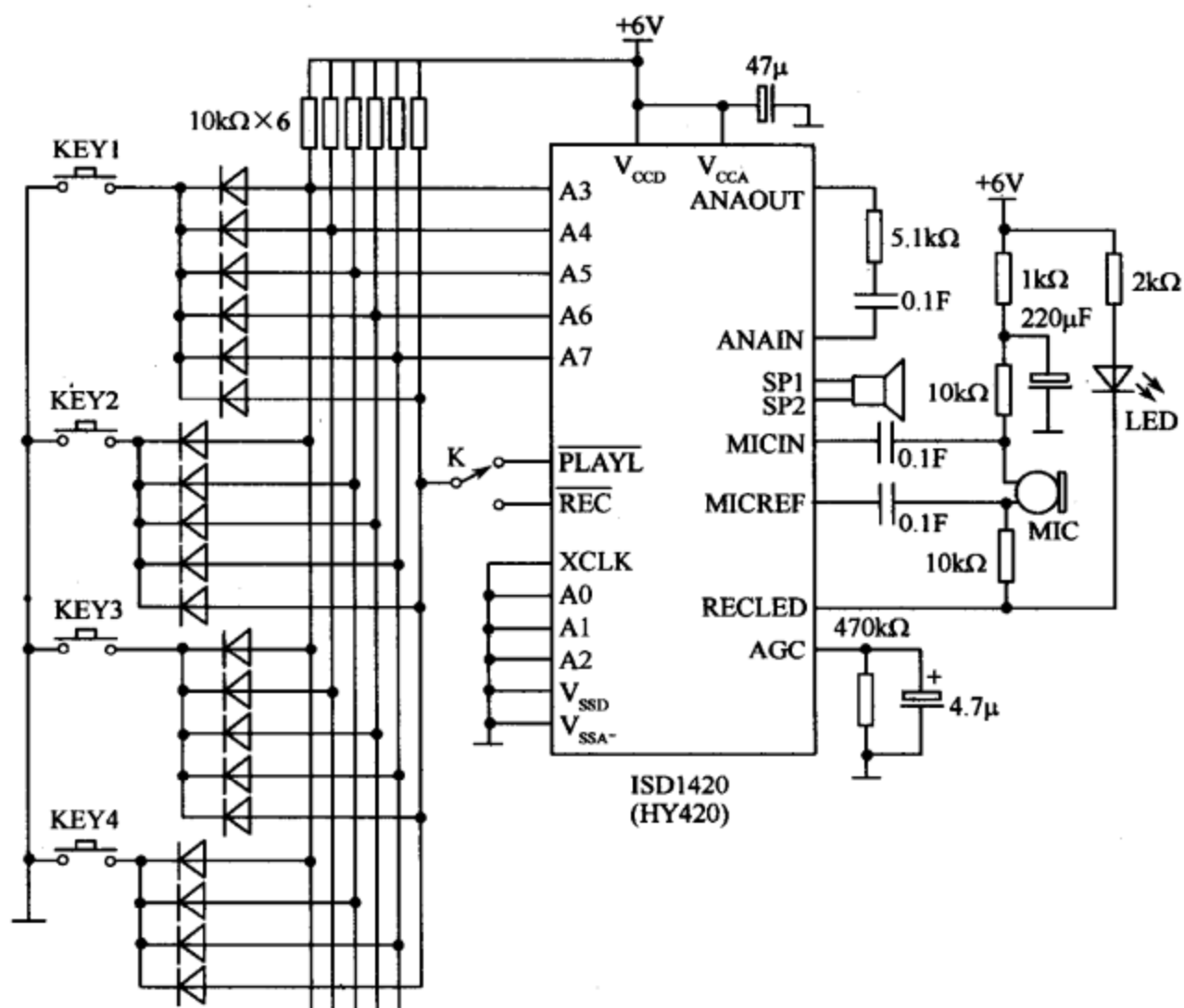


图 8-51 分段录音电路

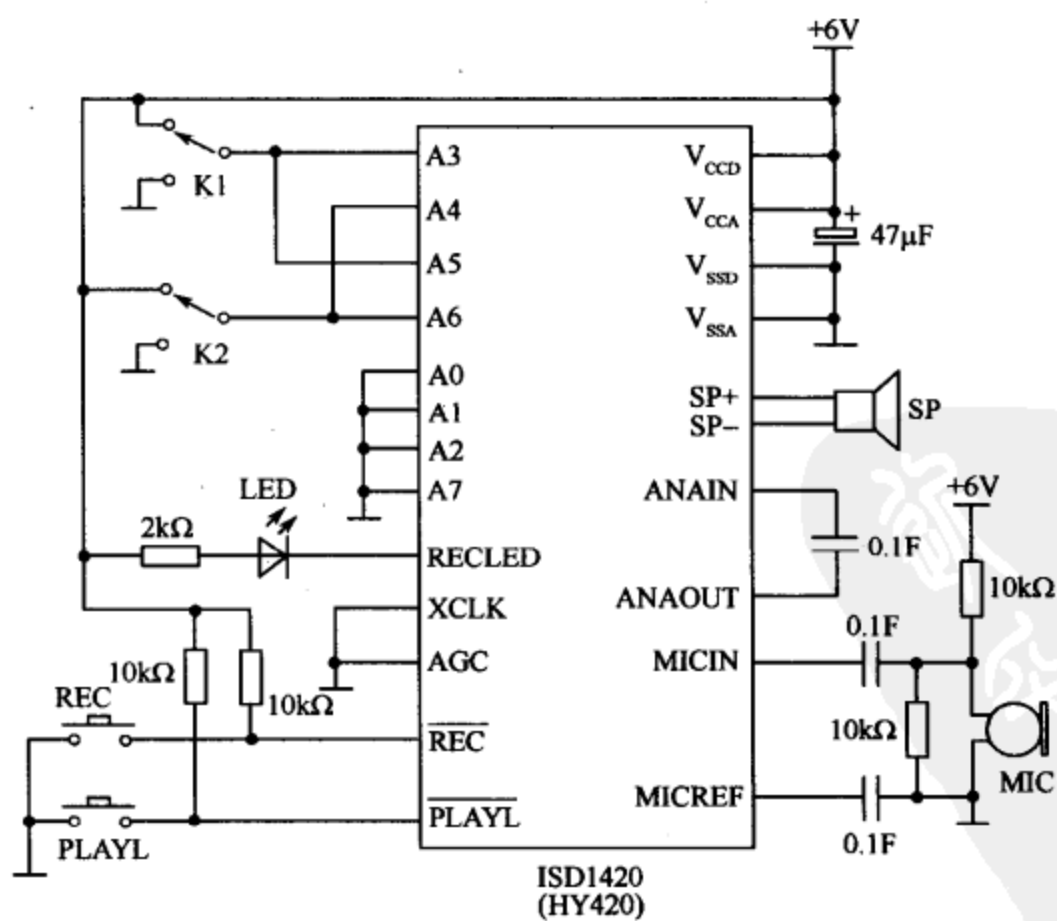


图 8-52 ISD1420 等分 4 段的应用

表 8-39 ISD1420 的 4 段等分地址设置表

A7	A6	A5	A4	A3	A2	A1	A0	16 进制数	K1	K2	段数	时间
0	0	0	0	0	0	0	0	00H	0	0	1	5s
0	0	1	0	1	0	0	0	28H	1	0	2	5s
0	1	0	1	0	0	0	0	50H	0	1	3	5s
0	1	1	1	1	0	0	0	78H	1	1	4	5s

5. ISD1420 的应用

会“说话”的测量仪器是智能化仪器仪表设计的一个方向,目前已在许多场合得到应用,它能用语音自动报告测量结果。下面介绍使用 ISD1420(HY420)设计成能报告测量结果的数字毫伏表,方法如下:

(1)编制语音库。预计需要发的音有“拾、百、千、万、毫伏,0~9”共 15 个音,现必须把 ISD1420 的存储器分为 15 个以上的区段,并把 15 个音进行分段独立录音,这样就把这 15 个音分别存入了存储器相应的区段。例如,我们按照表 8-40 所列的分段方法,将存储器分为 00—08—10H—18H—……—70H 共 15 个区段,并进行分段录音后构成了语音库。

表 8-40 发音、语音代码和地址的关系

发音	语音代码	语音直接地址	语音间接地址(电子表格地址)
零	00	00	300
一	01	08	301
二	02	10	302
三	03	18	303
四	04	20	304
五	05	28	305
六	06	30	306
七	07	38	307
八	08	40	308
九	09	48	309
十	0A	50	30A
百	0B	58	30B
千	0C	60	30C
万	0D	68	30D
毫伏	0E	70	30E

按照上述分段法,除最后一段外,其余每段占时为 $0.125s \times 8 = 1s$ 。在进行录音时,按下录音键 REC 的时间必须小于 1s,这在具体操作上有一定困难,必须借助一个自动定时控制电路,或多次返工才能完成(根据经验,每秒可发 3 个汉字语音)。

(2)规定语音代码。用代码来代表所发的语音,一般用代码 01~09 来代表数字语音 0~9,其余几个语音“拾、百……”分别用代码“0A、0B、0C……”等代表,如表 8-40 所列。

(3)制作电子表格。表格的格式见上表,制作时把各段的首地址写入到存储器中,即把语音“零”的首地址 00 写入到表格地址(即存储器地址)的 300H 存储单元中;把语音

“一”的首地址 08 写入到表格地址的 301H 存储单元中;把语音“二”的首地址 10H 写入到表格地址的 302H 存储单元中……

(4)硬件结构。本例的硬件电路如图 8-53 所示。使用单片机 AT89C51 的 P1.0~P1.3 读取 A/D 转换器 ICL7135 所输出的 BCD 码,P1.4~P1.7 以及 P3.7 读取 ICL7135 输出的位选通信号,P3.0~P3.4 去控制语音电路的地址,以便从不同的地址段发音。

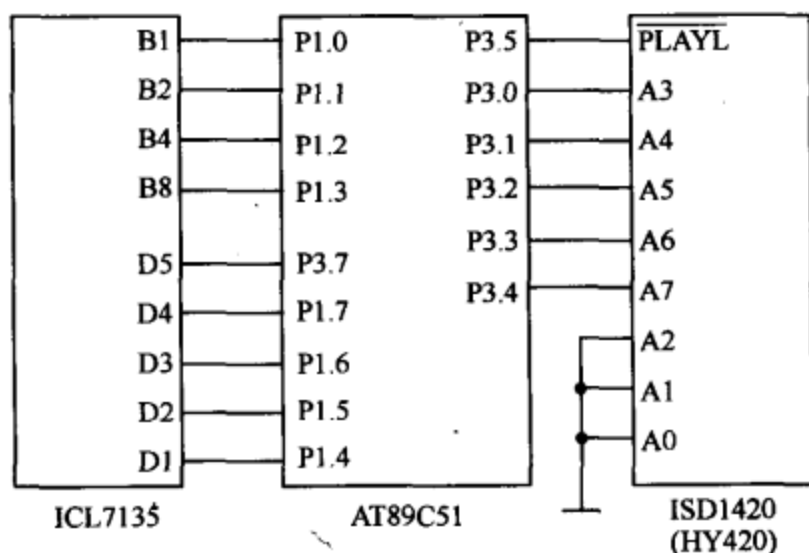


图 8-53 会“说话”的电压表硬件结构

(5)软件编制。首先,AT89C51 必须先读得 A/D 转换的结果,在判别到其输出端 D5 为高电平时读取 B1、B2、B4、B8 端的 BCD 码为“万”位,判别到 D4 为高电平时读取的 BCD 码为千位……因此,待测量数据稳定时,不难读得由 5 位数组成的测量值;第 2 步,根据所得到的测量值,用语音代码组成“语音代码串”,例如当所测到的电压为 365mV 时,发音应该是“三百六拾五毫伏”,所组成的语音代码串为 03、0B、06、0A、05、0E;第 3 步,根据语音代码计算出表格地址(也称做“语音间接地址”);最后按照所得到的表格地址(也就是存储单元的地址)从存储器(注意这不是语音电路 ISD1420 内部的语音存储器)里取出语音地址(又称语音直接地址)并从 P3.0~P3.4 送出,这样就为语音电路给定了发音的起始地址(图中 A0~A2 已固定接为低电平),同时,从 P3.5 送出低电平使放音控制端 PLAYL 为低电平,语音电路就以给定的 A0~A7 的值为起始地址而发音。本例中已得到语音代码为 03、0B、06、0A、05、0E,先根据语音代码 03 计算得表格地址为 303H,从单片机存储器的 303H 单元中取出的语音地址为 18H,则从 P3.0~P3.4 送出 18H,使 ISD1420 的 A7~A0 被设置为 00011000,且 PLAYL 端为低电平,这样 ISD1420 就从地址 18H 处开始,把原先录入的语音“三”发出;随后又根据语音代码 0B,计算得表格地址为 30BH,从单片机 30B 单元取出语音地址为 58H,则从 P3.0~P3.4 送出 58H,使 ISD1420 从地址 58H(即 A7~A0 为 01011000)处发音,且发出的是事先录入的“百”的语音……

电子表格是与单片机的程序一同放在程序存储器中的。

采用 ISD1420(HY420)等语音集成电路设计的具有自动报数功能的测量仪表,其主要缺点在于语音分段录制比较困难,导致报告数据时字与字之间的停顿、语调等难以控制(因为是一个字音一个字音拼接起来的),使得发音听起来不那么顺耳,甚至有些别扭。当需要高质量的语音报数时,则必须采用通用的语音处理器,如 UM93510 等,其语音的录制是在另外的设备上进行的,并经过了多次反复试听与剪辑,然后写入由不挥发存储器构成的语音库。

二、ZY1420A 语音电路

ZY1420A 是广州致远电子有限公司出品的优质微型语音录放模块。ZY1420A 内部使用 ISD1420 作为主控芯片,且具备 ISD1420 的全部优良性能,如大容量的 EEPROM 存储器,消噪的话筒放大器,自动增益调节 AGC 电路,专用语音滤波电路,高稳定性的时钟振荡电路和语音处理电路。除此以外,ZY1420A 还对 ISD1420 的标准外围电路作了优化并全部集成于模块内部。

ZY1420A 使用有专利技术的模拟处理存储方式,使录放音质极佳,没有常见的背景噪声,且电路断电后语音内容仍不会丢失。电路内部由振荡器、语音存储单元、前置放大器、自动增益控制电路、抗干扰滤波器、输出放大器组成。一个最小的录放系统仅由一个麦克风、一个喇叭、两个按钮、一个电源就可以组成。ZY1420A 的引脚排列图如图 8-54 所示。

录音控制	1	REC	A7	28	地址位 7
触发放音控制	2	PLAYE	A6	27	地址位 6
电平放音控制	3	PLAYL	A5	26	地址位 5
地	4	V _{ss}	A4	25	地址位 4
喇叭 +	5	SP+	A3	24	地址位 3
喇叭 -	6	SP-	A2	23	地址位 2
正电源	7	V _{cc}	A1	22	地址位 1
麦克 -	8	MIC-	A0	21	地址位 0
麦克 +	9	MIC+	NC	20	无连接
外部音频输入	10	ANA IN	NC	19	无连接
录音指示输出	11	RECLED	NC	18	无连接
无连接	12	NC	NC	17	无连接
无连接	13	NC	NC	16	无连接
无连接	14	NC	NC	15	无连接

图 8-54 ZY1420A 引脚排列图

1. 普通操作方法

ZY1420A 具备 ISD1420 的多种工作模式,对于通常的使用,用户一般是采用一段录音和放音的方法,这样 ZY1420A 能为用户提供最长 20s 的录音和放音时间。图 8-55 给出的是采用最简单的按键普通操作的使用方法,用户也可以使用单片机或逻辑电路来加以控制。

当开始录音时,RECLED 脚变为低电平,可以下拉电流驱动一个 LED 显示。ZY1420A 内部已经设计了一个 LED 位置,用户也可以在外部设计一个 LED 显示。图 8-55 所示电路中,接通电源后,电路自动进入节电准备状态。

(1)录音。按住录音按键(REC保持低电平),电路进入录音状态,当REC变高或录音存储器录满时,电路退出录音状态进入准备状态。注意REC的优先级大于PLAYE和PLAYL。

说明 为了操作方便,可以使用 ZY1420A 专用编程器,在编程软件的控制下进行录音。

(2)放音。放音有两种方式:触发放音和电平放音。

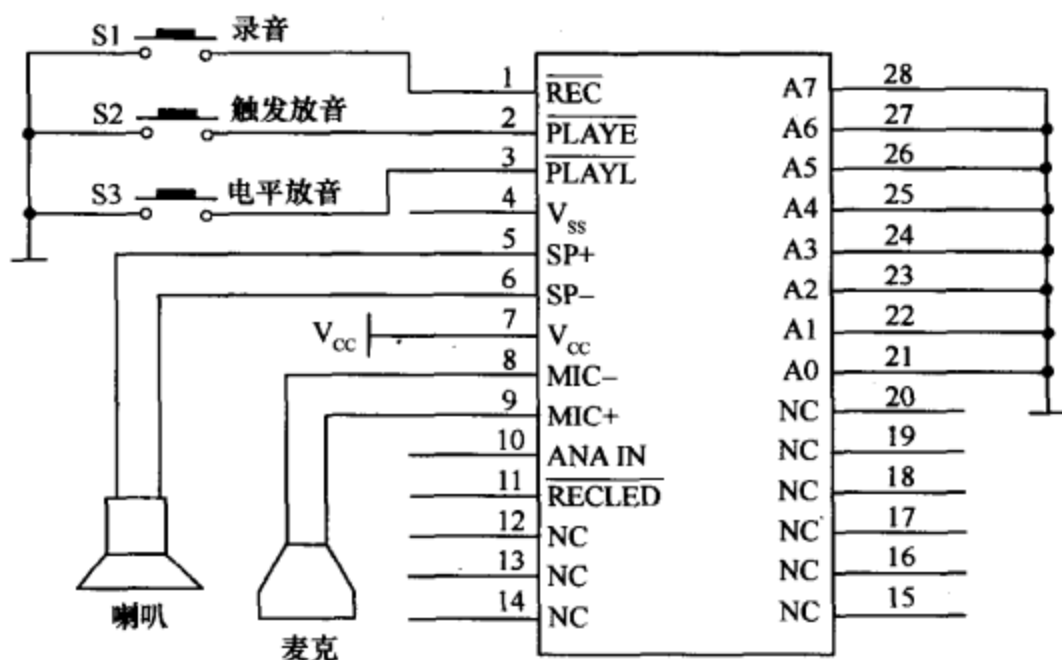


图 8-55 ZY1420A 的普通操作

触发放音:轻按 $\overline{\text{PLAYE}}$ 按键,这样给 $\overline{\text{PLAYE}}$ 脚一个低电平脉冲,电路进入放音状态,直到放音结束。

电平放音:按下 $\overline{\text{PLAYL}}$ 按键($\overline{\text{PLAYL}}$ 脚保持为低电平),电路进入放音状态,直到 $\overline{\text{PLAYL}}$ 变高或放音结束,电路重新进入准备状态。

2. 复杂操作方法

根据 A6、A7 的电平不同,电路可以进入两种不同的工作模式:地址模式和操作模式。

(1)地址模式。如果 A7、A6 至少有一位为低电平,则电路认为 A0~A7 全部为地址位, A0~A7 的数值将作为本次录音或放音操作的起始地址。在地址模式中, A0~A7 由低位向高位排列,每位地址代表 125ms 的寻址,160 个地址覆盖 20s 的语音范围($160 \times 0.125\text{s} = 20\text{s}$)。录音及放音功能均从设定的起始地址开始,录音结束由停止键操作决定,芯片内部自动在该段的结束位置插入结束标志(EOM),而放音时芯片遇到 EOM 标志即自动停止放音。

(2)操作模式。当 A7、A6 全部为 1 时,器件进入操作模式。ZY1420A 内部具备有多种操作模式,并能以最少的元件实现较多的功能。操作模式的选择使用地址管脚来实现,但实际的地址在 ZY1420A 的有效地址外部。当地址的最高两位 A7、A6 为高电平时,其余的地址位将成为状态标志位而不再是地址位。因此,操作模式和寻址模式不能兼容,也就是说不能同时使用。

可以使用单片机来控制操作模式,也可以直接使用连线来实现需要的功能。

A0:信息检索。信息检索允许用户对内容跳转浏览,而不必关心每个信息的实际物理位置。每个控制信号的低电平脉冲将内部地址指针转移到下一个信息位置。这种模式只能在放音中使用,通常与 A4 操作同时应用。

A1:删除 EOM 结尾标志。A1 操作模式允许多次记录的信息组合成一个信息,结束标志只出现在最后录制信息的结尾。当配置成这种模式后,多次录制的信息在放音时会形成连续的信息。

A3:循环播放。A3 操作模式能够实现自动连续的信息播放,播放的信息处于地址空间的开始。如果一个信息充满了 ZY1420A,则用循环模式可以从头到尾连续的播放。 $\overline{\text{PLAYE}}$ 脉冲可以启动播放, $\overline{\text{PLAYL}}$ 脉冲可以结束播放。

A4:连续寻址。在通常的操作中,当放音操作遇到结尾标志(EOM)时,地址指针将复原到 0。A4 操作模式将禁止地址指针的复位,允许信息能连续录制和播放。当电路处于静止状态,不是处于录音或放音状态,即可的设置该脚为低电平将复位地址指针。

A2、A5:没有使用。

3. 编程举例

下面以播放“请 19 号顾客到 6 号窗口”为例,来讲解 ZY1420A 的语音模块的编程方法。有关电路如图 8-56 所示。

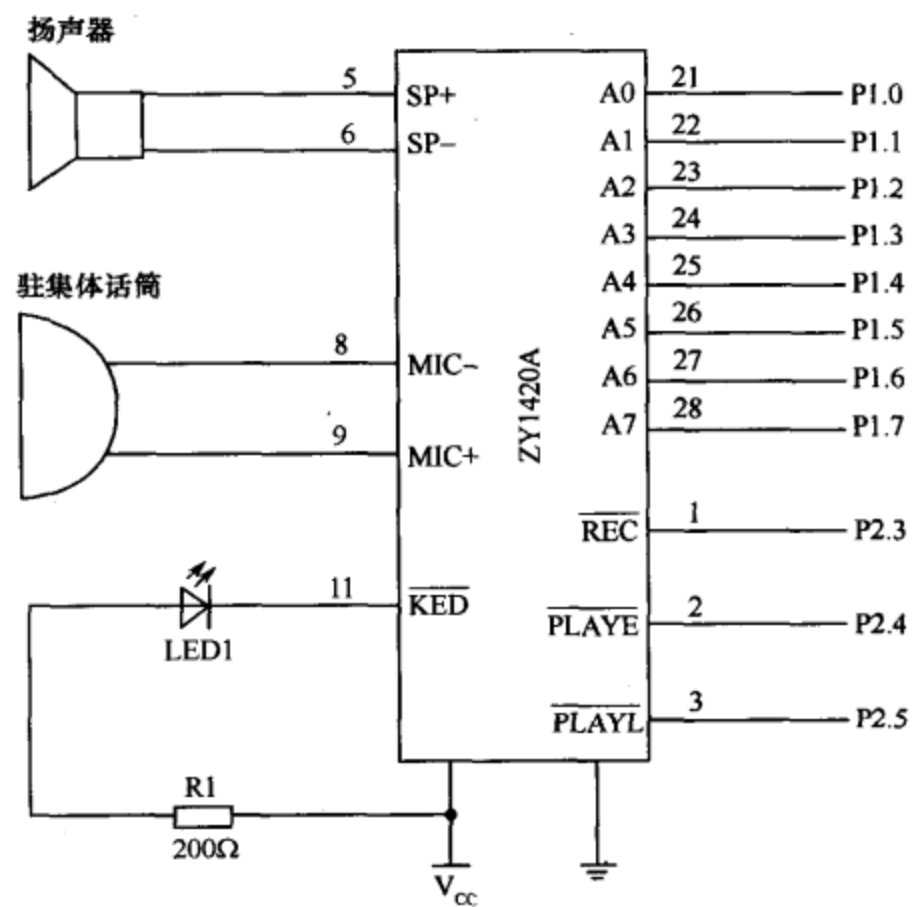


图 8-56 硬件电路

由图可知,只需控制单片机 P1 和 P2. 3~P2. 5 口即可以实现对 ZY1420A 语音模块的录音、电平控制放音和脉冲触发放音。由于在一般的产品中用到语音模块主要是应用它的自动语音提示功能,因此在此种方式下,电平控制的放音模式是必需的。

在电平控制放音模式下,开始地址和播放时间是必不可少的两个参数,只要给出了这两个参数就可以确定播放的内容。不过需要注意的是,播放的时候总是从一段的开头开始播放的,当需要分段播放控制时,录音必须从某一段的段头开始存放。为此可以首先把要播放的内容写入到 ZY1420A 中,并建立存放信息一览表,如表 8-41 所列。

表 8-41 存放信息一览表

发音	语音代码	语音直接地址	时间/s	发音	语音代码	语音直接地址	时间/s
零	00	00	0.5	七	07	42	0.5
一	01	06	0.5	八	08	48	0.5
二	02	12	0.5	九	09	54	0.5
三	03	18	0.5	十	0A	60	0.5
四	04	24	0.5	请	0B	66	0.5
五	05	30	0.5	号顾客到	0C	72	1. 25
六	06	36	0.5	号窗口	0D	100	

根据以上分析,编写的源程序如下:

```
A0 BIT P1.0
A1 BIT P1.1
A2 BIT P1.2
A3 BIT P1.3
A4 BIT P1.4
A5 BIT P1.5
A6 BIT P1.6
A7 BIT P1.7
REC BIT P2.3
PLAYE BIT P2.4
PLAYL BIT P2.5
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN: MOV SP, #60H      ;给堆栈指针赋初值
LOOP: MOV R6, #66       ;送“请”的首地址
      MOV R7, #40       ;送播放时间
      ACALL PLAY        ;开始播放“请”
      MOV R6, #6        ;送“一”的首地址
      MOV R7, #30       ;送播放时间
      ACALL PLAY        ;开始播放“一”
      MOV R6, #60       ;送“十”的首地址
      MOV R7, #30       ;送播放时间
      ACALL PLAY        ;开始播放“十”
      MOV R6, #54       ;送“九”的首地址
      MOV R7, #30       ;送播放时间
      ACALL PLAY        ;开始播放“九”
      MOV R6, #72       ;送“号顾客到”的首地址
      MOV R7, #100      ;送播放时间
      ACALL PLAY        ;开始播放“号顾客到”
      MOV R6, #36       ;送“六”的首地址
      MOV R7, #40       ;送播放时间
      ACALL PLAY        ;开始播放“六”
      MOV R6, #100      ;送“号窗口”的首地址
      MOV R7, #80       ;送播放时间
      ACALL PLAY        ;开始播放“号窗口”
      MOV R7, #200
      ACALL DELAY
```

```

    AJMP LOOP
;播放子程序,R6 存放播放信息的首地址,R7 存放播放时间
PLAY:  MOV P1,R6          ;送首地址
        CLR PLAYL         ;开始播放
        ACALL DELAY       ;延时相应的播放时间
        SETB PLAYL        ;播放完,则停止
        MOV R7,#01        ;延时 50ms
        ACALL DELAY
        RET
;延时子程序 50ms
DELAY:  MOV R5,#25
D1:     MOV R6,#249
D2:     DJNZ R6,D2
        DJNZ R5,D1
        DJNZ R7,DELAY
        RET
        END

```

第八节 A/D 和 D/A 转换接口

单片机的外部设备不一定是数字式的,也经常会和模拟式的设备连接。例如,单片机来控制温度、压力时,温度和压力都是连续变化的,都是模拟量,在单片机与外部环境通信的时候,就需要有一种转换器来把模拟信号变为数字信号,以便能够输送给单片机进行处理。这种将模拟量转换为数字量的接口电路称为模/数(A/D)转换电路。

单片机送出的控制信号,也必须经过变换器变成模拟信号,才能为控制电路所接受。这种将数字量变换为模拟量的接口电路称为数/模(D/A)转换电路。

A/D 转换器和 D/A 转换器已成为计算机系统中不可缺少的接口电路。

一、A/D 转换接口

A/D 转换器的种类很多,按其工作原理不同分为直接 A/D 转换器和间接 A/D 转换器两类。直接 A/D 转换器可将模拟信号直接转换为数字信号,这类 A/D 转换器具有较快的转换速度,其典型电路有逐次比较型 A/D 转换器。而间接 A/D 转换器则是先将模拟信号转换成某一中间电量(时间或频率),然后再将中间电量转换为数字量输出。此类 A/D 转换器的速度较慢,典型电路是双积分型 A/D 转换器、电压频率转换型 V/F 转换器。下面主要以常用的 A/D 转换器 TLC0831 为例进行介绍。

TLC0831 是美国德州仪器公司出品的 8 位串行 A/D 转换器,单通道 8 位分辨率,输入/输出电平与 TTL/CMOS 兼容;工作频率为 250 kHz 时,转换时间为 32 μ s。图 8-57 是该器件的引脚图。

图中 $\overline{\text{CS}}$ 为片选端;IN+为正输入端,IN-是负输入端。TLC0831 可以接入差分信

号,如果输入单端信号,IN-应该接地。REF 是参考电压输入端,使用中应接参考电压或直接与 V_{cc}接通。DO 是数据输出端,CLK 是时钟信号端。这两个引脚用于与单片机通信。图 8-58 是 ADC0831 与单片机的接线图。

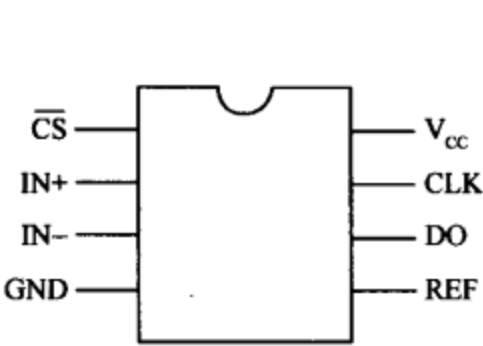


图 8-57 TLC0831 引脚功能图

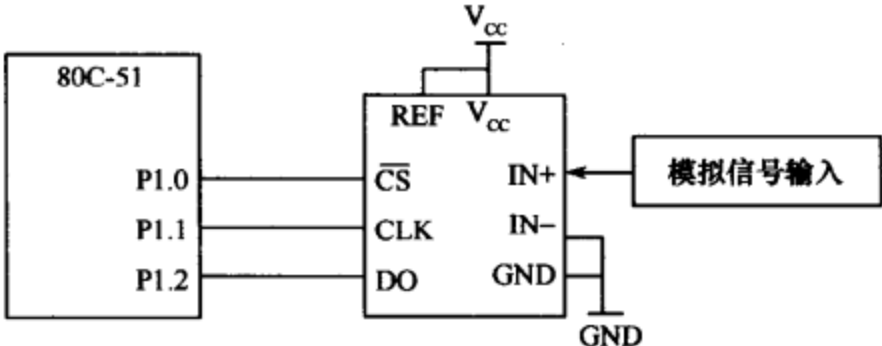


图 8-58 TLC0831 与单片机的接线图

置CS为低电平开始一次转换,在整个转换过程中CS必须为低。连续输入 10 个脉冲完成一次转换,数据从第 2 个脉冲的下降沿开始输出。转换结束后应将CS置高电平,当CS重新拉低时将开始新的一次转换。

以下是获得 TLC0831 的 IN+端的输入值的源程序。

```

CS BIT P1.0
CLK BIT P1.1
DO BIT P1.2
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN:…           ;加入其他程序
    ACALL ADC      ;调用 A/D 转换子程序
    :              ;加入其他程序
;以下是 A/D 转换子程序
ADC: CLR CS        ;拉低片选端
    NOP
    NOP
    SETB CLK       ;拉高 CLK 端
    NOP
    NOP
    CLR CLK        ;拉低 CLK 端,形成下降沿
    NOP
    NOP
    SETB CLK       ;拉高 CLK 端
    NOP
    NOP
    CLR CLK        ;拉低 CLK 端,形成第 2 个脉冲的下降沿
    
```

```

NOP
NOP
MOV R7, #8      ;准备发送后 8 个时钟脉冲
ADC1: MOV C, DO  ;接收数据
MOV ACC.0, C
RL A             ;左移 1 次
SETB CLK
NOP
NOP
CLR CLK         ;形成 1 次时钟脉冲
NOP
NOP
DTNZ R7, ADC1   ;循环 8 次
SETB CS         ;拉高片选端
CLR CLK         ;拉低 CLK 端
SETB DO         ;拉高数据端,回到初始状态
RET
END

```

二、D/A 转换接口

目前,D/A 转换器从接口上可分为两大类:并行接口 D/A 转换器和串行接口 D/A 转换器。并行接口 D/A 转换器的引脚多,体积大,占用单片机的口线多;而串行 D/A 转换器的体积小,占用单片机的口线少。为减少线路板的面积和占用单片机的口线,可采用串行 D/A 转换器。下面以常见的串行 D/A 转换器 TLC5615 为例进行介绍。

TLC5615 为美国德州仪器公司推出的产品,是具有串行接口的 10 位 D/A 转换器,其输出为电压型,最大输出电压是基准电压值的两倍。带有上电复位功能,即把 D/A 转换寄存器复位至全零。TLC5615 性能价格比高,目前在国内市场很方便购买。图 8-59 是 TLC5615 的引脚图。

图 8-59 中,DIN 为串行数据输入端;SCLK 为串行时钟输入端; \overline{CS} 为片选信号,低电平有效;DOUT 为串行数据输出端,用于级联;AGND 是模拟地;REFIN 为基准电压输入;OUT 为 D/A 转换模拟电压输出端; V_{CC} 为电源端。

图 8-60 是单片机与 TLC5615 的接线图。

根据 TLC5615 的时序关系可知,当片选 \overline{CS} 为低电平时,输入数据 DIN 由时钟 SCLK 同步输入或输出,而且最高有效位在前,低有效位在后。输入时,SCLK 的上升沿把串行输入数据 DIN 移入内部的移位寄存器,SCLK 的下降沿输出串行数据 DOUT,片选 \overline{CS} 的上升沿把数据传送至 D/A 转换寄存器。

当片选 \overline{CS} 为高电平时,串行输入数据 DIN 不能由时钟同步送入移位寄存器;输出数据 DOUT 保持最近的数值不变而不进入高阻状态。由此要想串行输入数据和输出数据

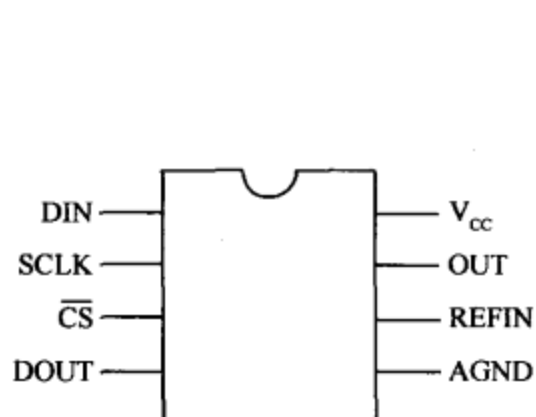


图 8-59 TLC5615 引脚功能图

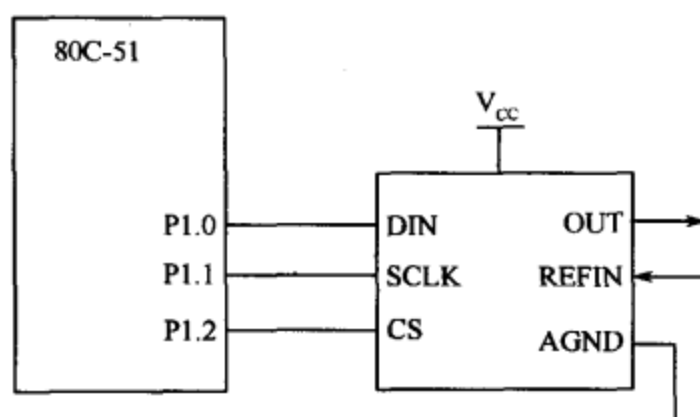


图 8-60 单片机与 TLC5615 的接线图

必须满足两个条件:第一,时钟 SCLK 的有效跳变;第二,片选 \overline{CS} 为低电平。下面的源程序是将内存 30H 和 31H 单元的数值转换为模拟量,其中高 2 位在 30H 单元,低 8 位在 31H 单元。

```

DIN BIT P1.0
SCLK BIT P1.1
CS BIT P1.2
ORG 0000H
AJMP MAIN
ORG 0100H
MAIN: MOV R1,30H
      MOV R2,31H
      :           ;加入其他程序
      ACALL DAC   ;调用 D/A 转换子程序
      :           ;加入其他程序
;以下是 D/A 转换子程序
DAC:  SETB CS     ;拉高片选端
      NOP
      NOP
      CLR DIN
      CLR SCLK
      CLR CS     ;拉低时钟、数据和片选端
      NOP
      NOP
      MOV A,R1   ;取得待输出数据高 2 位
      MOV R3,#02H ;准备循环两次
DAC1: RLC A
      MOV DIN,C  ;送出数据
      NOP
      NOP

```

```

CLR SCLK
NOP
NOP
SETB SCLK      ;形成上升沿时钟脉冲
DJNZ R3,DAC1
MOV R3,#08H
MOV A,R2        ;取得待输出数据低 8 位
DAC2: RLC A
MOV DIN,C       ;送出数据
NOP
NOP
CLR SCLK
NOP
NOP
SETB SCLK      ;形成上升沿时钟脉冲
DJNZ R3,DAC2
SETB CS
CLR SCLK
CLR DIN         ;拉高片选端,拉低时钟端与数据端,回到初始状态
RET
END

```



第九章 单片机应用系统设计

通过前面各章的学习,我们已经掌握了单片机的基本工作原理、简单程序设计方法、存储器和 I/O 接口的扩展方法以及常用接口技术等。它们是设计单片机应用系统的软件和硬件基础。有了这些基础以后,就可以进行单片机应用系统的设计与开发了。本章首先介绍单片机应用系统的设计的基本原则;然后,以“单片机电子钟”为例,对单片机硬件和软件的设计进行详细分析,使读者能够了解和领会单片机应用系统设计开发的思路、技巧和方法;最后简要介绍单片机应用系统的可靠性设计技术。

第一节 单片机应用系统设计的原则

单片机应用系统是完成某项任务而研制开发的用户系统,虽然每个系统都有很强的针对性,结构和功能各异,但它们的开发过程和方法大致相同。下面简要介绍单片机应用系统开发的一般方法和步骤。

一、确定任务

单片机应用系统的开发过程是以确定系统的功能技术指标开始的。首先要细致分析、研究实际问题,明确各项任务与要求,综合考虑系统的各种性能,拟定出合理可行的技术性能指标。

二、总体设计

在对应用系统进行总体设计时,应根据应用系统提出的各项技术性能指标,拟定出一套合理的方案。首先,根据任务的繁杂程度和技术指标要求选择单片机芯片。其次,选择系统中要用到的其他外围元器件,如显示器、执行机构等。

三、硬件设计

硬件设计是指应用系统的电路设计,包括单片机芯片、控制电路、存储器、I/O 接口等。硬件设计时,应考虑留有充分余量,电路设计力求正确无误,因为在系统调试中不易修改硬件结构。在单片机应用设计系统中,硬件电路设计时应注意的以下几个问题。

1. 程序存储器

在外扩程序存储器时,一般选用容量较大的 EPROM 芯片,如 2764(8KB)、27128(16KB)或 27256(32KB)等。尽量避免用小容量芯片组合扩充大容量的存储器。程序存储器容量大些,则可用编程空间充裕。

2. 数据存储器

根据系统功能的要求,如果需要扩展外部 RAM,那么 RAM 芯片可选用 6116(2KB)、

6264(8KB)等。扩展外部 RAM 的原则和扩展外部 ROM 相同,尽量减少芯片数量,使电路结构简单。

3. I/O 接口芯片

I/O 接口芯片的扩展也需要根据应用系统功能的要求来确定。常用的 I/O 接口芯片一般选用 8155 芯片。这类芯片可利用功能多、具有口线多、硬件逻辑简单等特点。

4. 总线驱动能力

MCS-51 系列单片机的外部扩展功能很强,但 4 个 8 位并行口的带负载能力是有限的。P0 口能驱动 8 个 TIL 电路,P1~P3 口只能驱动 3 个 TIL 电路。在实际应用中,这些端口的负载不应超过总负载能力的 70%,以保证留有一定的余量,以增强系统的抗干扰能力。在外接负载较多的情况下,应采用总线驱动电路,以提高端口的驱动能力和系统的抗干扰能力。总线驱动有相应的驱动器,如:双向 8 路三态缓冲器 74LS245 可作为数据总线使用;地址和控制总线可采用单向 8 路三态缓冲器 74LS244 作为单向总线驱动器。

四、软件设计

通过对单片机的学习,我们已经了解到单片机应用系统的软件设计是研制过程中关键的一项工作。没有软件,就无法实现单片机的控制;不同软件可以实现功能不同的控制。所以,要编写软件一定要把要实现的控制对象及其功能全面掌握,要做到心中有数。

单片机应用系统的软件设计千差万别,不存在统一模式。开发一个软件的基本方法是尽可能采用模块化结构。根据系统软件的总体构思,按照先粗后细的方法,把整个系统软件分成多个功能独立模块。应明确规定各模块的功能、各模块间的接口信息,尽可能使各模块的联系减少到最低限度。这样,各个模块可以分别独立设计、编制和调试,最后再将各个程序模块连接成一个完整的程序进行总调试。

较为复杂软件的设计,是建立在各个基本模块的基础上。如果对基本模块熟悉了,编写一个较为复杂的软件相对就容易。

五、系统调试

系统调试包括硬件调试和软件调试。硬件调试的任务是排除系统的硬件电路故障,包括设计错误和工艺故障。软件调试是利用开发工具进行在线仿真调试,除发现和解决程序错误外,也可以发现硬件故障。

程序调试一般是一个模块一个模块的进行、一个子程序一个子程序的调试,最后连接起来统一调试。利用开发工具的单步和断点运行方式,通过检查应用系统的 CPU 现场、RAM 和 SFR 的内容以及 I/O 接口的状态,来检查程序的执行结果和系统 I/O 设备的状态变化是否正常,从中发现程序的逻辑错误、转移地址错误以及随机的录入错误等。也可以发现硬件设计与工艺错误和软件算法错误。在调试过程中,要不断调整、修改系统的硬件和软件,直到其符合预期结果为止。

联机调试运行正常后,将软件固化到 EEPROM 中,脱机运行,并到生产现场投入实用,检验其可靠性,直到完全满足要求,系统才算研制成功。

第二节 单片机电子钟应用系统的设计

下面以一个单片机电子钟为例,介绍单片机应用系统设计的方法和技巧。

在一个单片机系统中,要使单片机能够实现电子钟的功能,一般需要进行定时,定时时间通常有两种实现方法:一是通过单片机内部的定时/计数器,采用软件实现一般用在对时间精度要求不高的场合;二是采用时钟芯片,比较典型的时钟芯片有 PHILIPS 公司的 PCF8563、PCF8583, MOTOROLA 公司的 MC146818 及 Dallas 公司的 DS12887、DS1302 等。

下面以单片机电子钟为例,介绍应用系统设计的方法技巧。电子钟可以实现以下功能:显示时、分、秒并能定时报警;可以通过键盘修改显示时间和报警时刻。

一、系统软件、硬件功能的划分

电子钟的功能要求比较简单,因此主要考虑最大限度地节约系统的硬件成本,所有能用软件实现的功能都用软件完成,如,按键的去抖动采用软件延时的方法,显示部分采用动态显示等。这样,硬件部分的设计可以采用单片机最小系统,所谓最小系统是仅具有程序存储器和时钟及复位电路的单片机系统。

二、硬件设计

电子钟的硬件原理图如图 9-1 所示。

1. 单片机及程序存储器

AT89C51 内部有 4KB 的 Flash 存储器,本电子钟的程序不会超过 4KB,因此无需扩展外部的程序存储器。晶振采用 12MHz,两个 20pF 的电容是微调电容。

上电复位电路中,电阻和电容的选择是要保证电容充电时 RST 引脚上的高电平的维持时间大于 2 个机器周期,对应 12MHz 晶振,机器周期为 $1\mu\text{s}$,也即充电时 RST 引脚上高电平维持时间要求为 $2\mu\text{s}$ 以上。该复位电路复位引脚上的复位信号维持时间由 RC 的值确定,取电阻为 $1\text{k}\Omega$,电容为 $20\mu\text{F}$,这样完全可满足复位信号的要求。

2. 键盘

键盘用于校正时间,选用普通的按键,有功能键、确认键、减 1 键、增 1 键,共 4 个键。调整时,先按“功能键”选择调节的对象(哪一位小数点亮表示调整的是该位的值),再用“增 1 键”和“减 1 键”进行具体的调整,最后按“确认键”结束校时操作。

4 个按键通过 P3 口的 P3.2~P3.5 接入。当有键按下时,输入端为低电平;无键按下时,为高电平。

3. 显示器

显示器主要用于显示时间,选用 LED 显示器,共 6 位,分别显示时、分、秒,采用动态显示方式,通过软件进行动态扫描刷新。

本系统 LED 显示器的段控接 P0 口,位控接 P2 口的 P2.2~P2.7。P0 口做段控口输出时,由于其内部是漏极开路的,因此要外接 $1\text{k}\Omega$ 左右的上拉电阻(图 9-1 中未画出),而 P2 口由于其内部已有上拉电阻,无需外接。

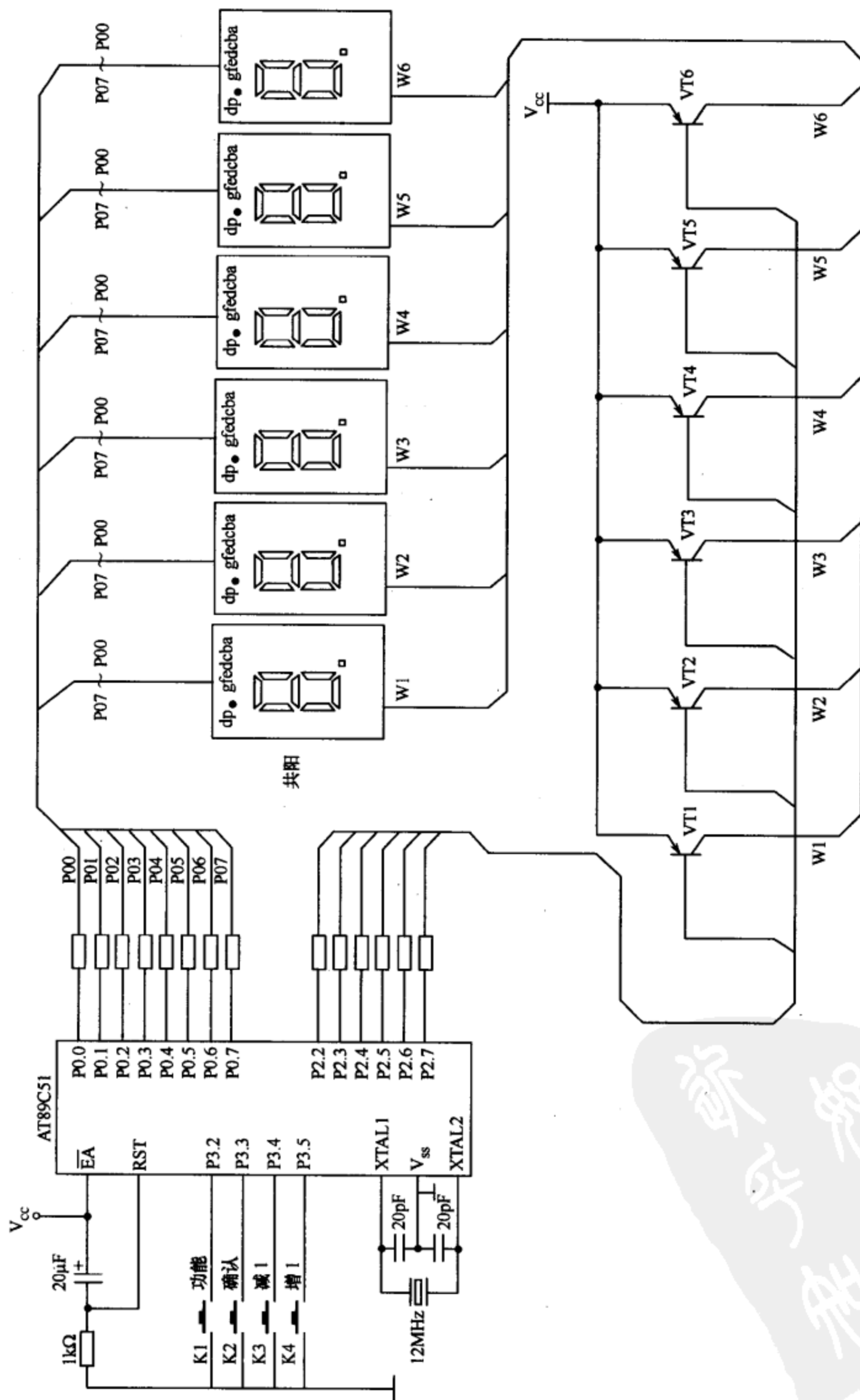


图 9-1 单片机电子钟硬件电路

LED 显示器是共阳的,要点亮某段,必须在位控端加高电平,段控端加低电平,一般有 5mA~30mA 的电流即可点亮。因此,可计算段控端的限流电阻(R1~R8)为 $(5-1.5)V/(5\sim30)mA$,式中的 1.5V,是发光二极管的每段压降。表 9-1、表 9-2 是根据图 9-1 所示电路原理图列出的字形码。

表 9-1 根据数码管连接方法写出的字形码(小数点不亮)

显示	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	字形码
	dp	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	0C0H
1	1	1	1	1	1	0	0	1	0F9H
2	1	0	1	0	0	1	0	0	0A4H
3	1	0	1	1	0	0	0	0	0B0H
4	1	0	0	1	1	0	0	1	99H
5	1	0	0	1	0	0	1	0	92H
6	1	0	0	0	0	0	1	0	82H
7	1	1	1	1	1	0	0	0	0F8H
8	1	0	0	0	0	0	0	0	80H
9	1	0	0	1	0	0	0	0	90H
灭	1	1	1	1	1	1	1	1	0FFH

表 9-2 根据数码管连接方法写出的字形码(小数点亮)

显示	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	字形码
	dp	g	f	e	d	c	b	a	
0	0	1	0	0	0	0	0	0	40H
1	0	1	1	1	1	0	0	1	79H
2	0	0	1	0	0	1	0	0	24H
3	0	0	1	1	0	0	0	0	30H
4	0	0	0	1	1	0	0	1	19H
5	0	0	0	1	0	0	1	0	12H
6	0	0	0	0	0	0	1	0	2H
7	0	1	1	1	1	0	0	0	78H
8	0	0	0	0	0	0	0	0	00H
9	0	0	0	1	0	0	0	0	10H
灭	0	1	1	1	1	1	1	1	7FH

三、软件设计

1. 软件设计的基本思路

在以前的章节中,我们依照从易到难的顺序,向读者分别介绍了单片机的内部结构、功能以及软件、硬件的实现方法,并给出了大量的子程序。面对大量的子程序,有些初学者往往会有这样一个疑问:单独的子程序都好理解,但一遇到具体的项目就糊涂了。之所

以有这样的疑问,一个最重要的原因就是:在读者的头脑里,对一个单片机项目的软件设计还没有形成一个完整的、总体的概念,有的只是一个个的孤立的个体,如何将这些独立的子程序组织在一起并设计出一个完整的应用系统,这是摆在初学者面前最大的问题。下面结合“单片机电子钟”这个具体的例子,介绍如何设计一个具有完整功能的项目的方法和技巧。

我们知道,任何一个单片机的软件系统都是由主程序和子程序组成的,主程序除了和初始化的工作外,最大的功能就是如何组织、调用子程序。图 9-2 则说明了单片机软件系统的主程序和子程序的关系,这也是读者必须掌握的。

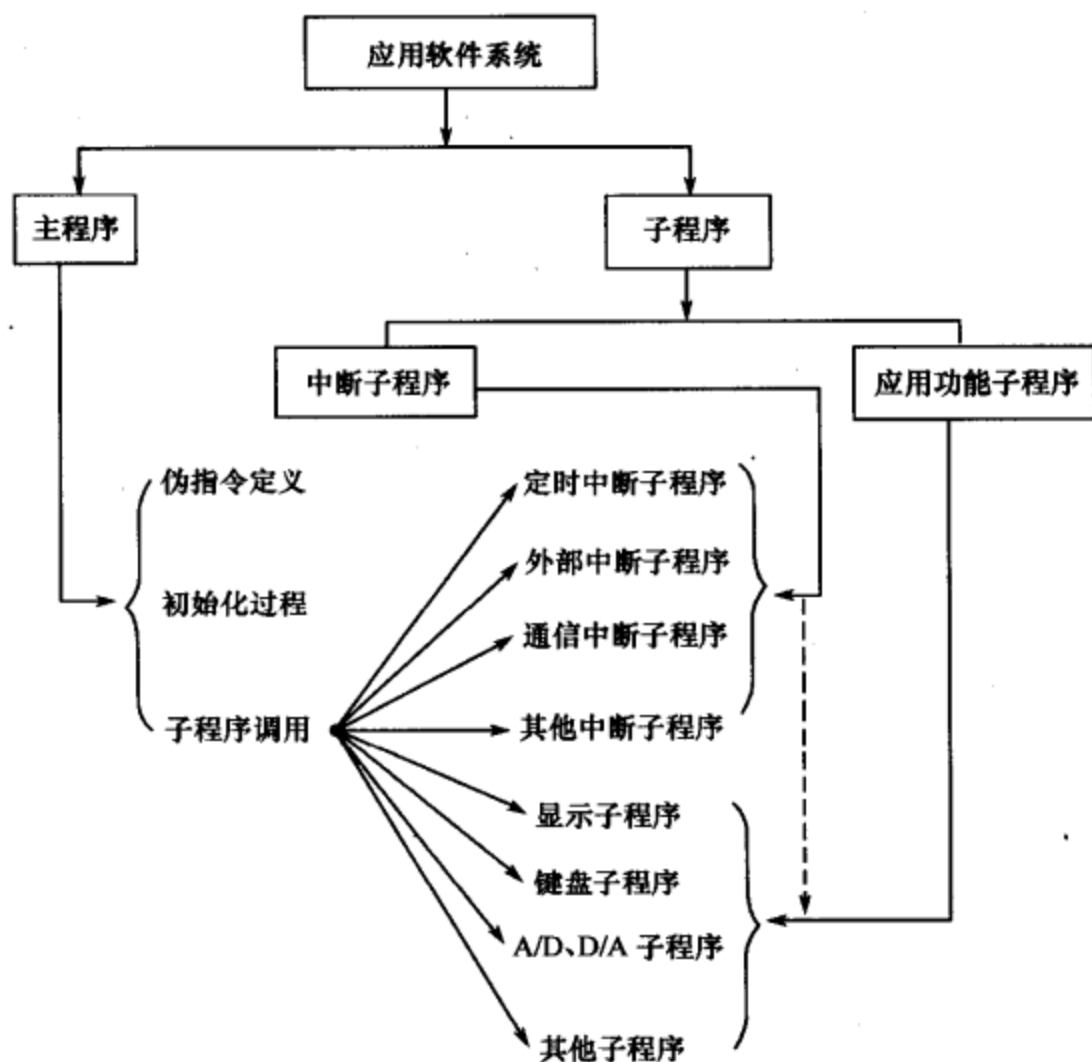


图 9-2 主程序和子程序的关系

从图 9-2 中可以看出,主程序主要包含伪指令定义、初始化过程和子程序调用 3 部分内容。伪指令是一种控制汇编进程的特殊的控制符号,通常在主程序开头定义,用好伪指令可以使程序具有很好的可读性,并易于操作。初始化过程主要是对一些控制寄存器(如中断控制)、数据区和外部芯片(如 I/O 扩展芯片 8155)进行初始参数设置和定义。子程序调用是主程序的基本任务,一个主程序可以调用多个子程序,对于 89C51 单片机,由于系统资源有限,主程序通常是一个无限循环的过程,即是一个反复调用子程序的过程(图中的粗线箭头实际就是一个子程序调用过程)。子程序主要分为中断子程序和功能子程序,它们之间可以相互嵌套和调用,即中断子程序可以调用功能子程序。在应用软件的设计中,读者一定要注意,应尽可能将各个功能模块写成子程序的形式,并通过主程序调用,只有这样,软件结构才会清晰明了,也易于维护。

2. 软件设计的方法

1) 主程序

主程序首先是初始化部分,主要是计时单元清零、中断初始化、堆栈指针初始化、启动定时器工作,然后是调用显示子程序,接着是判断有无按键。无按键则回到调用显示子程序处;有按键,则执行按键处理子程序,执行完后回到调用显示子程序处,重复循环。主程序流程如图 9-3 所示。

2) 中断服务程序

本电子钟的计时是用单片机内部的定时计数器 T0,定时 50ms,即 0.05s,20 次中断即为 1s,60s 为 1min,60min 为 1h,24h 为 1 天,如此循环,从而实现计时功能。其流程图如图 9-4 所示。

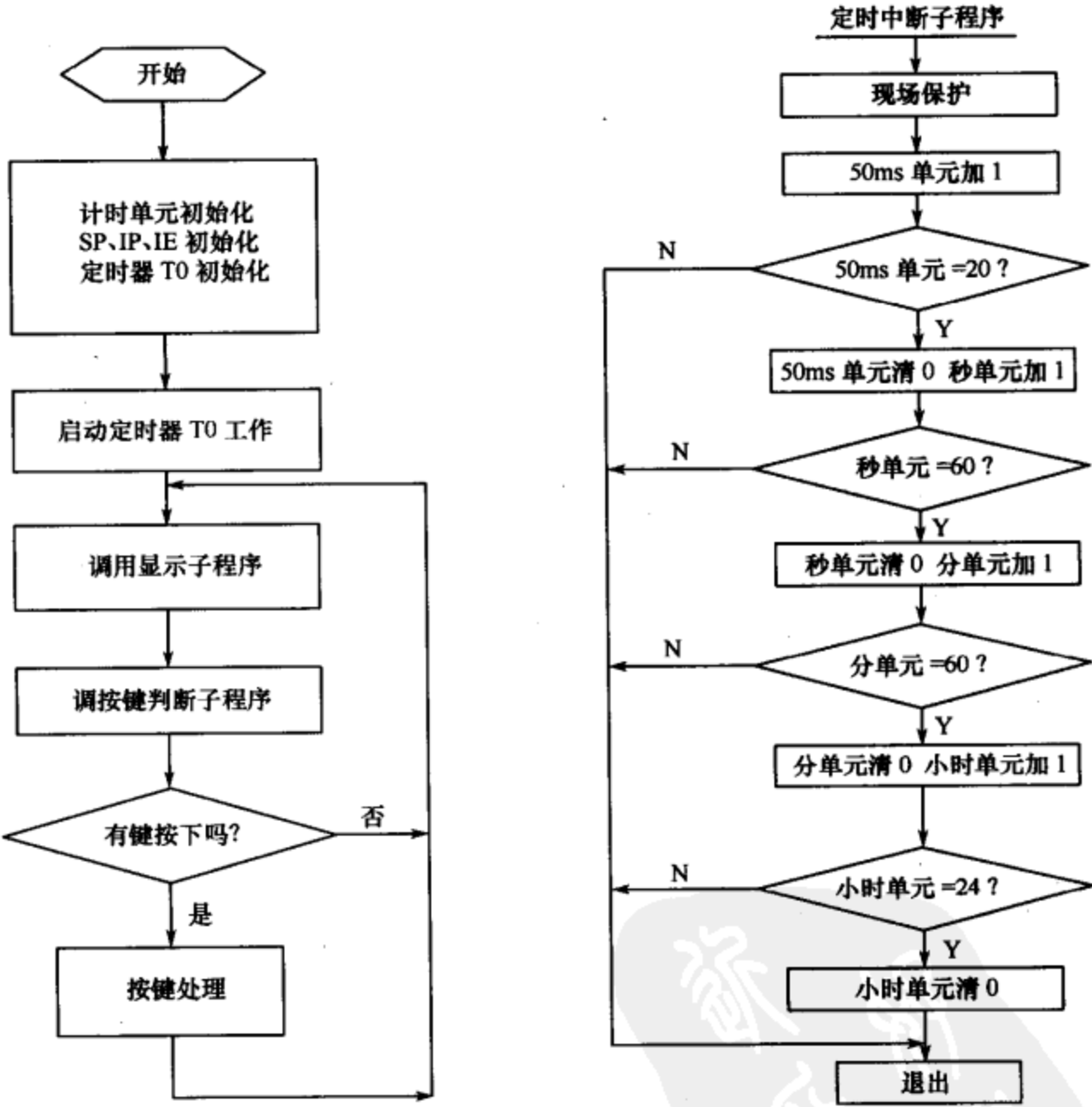


图 9-3 主程序流程图

图 9-4 中断服务子程序

如果采用定时器 T0 的方式 1,设计数初值为 X,根据

$$\text{定时时间} = (2^{16} - X) \times \frac{12}{\text{晶振频率}}$$

可知

$$50000 = (65536 - X) \times 1$$

所以, $X=15536D=3CB0H$ 。

定时器 T0 的计时初值为 3CB0H, 考虑到中断响应时间, 可在 3CB0H~3CB7H 调整计时初值。

编写中断服务程序关键要以下几点: 一是现场保护, 在本系统中是累加器 A 和程序状态字 PSW 值的保护。二是计时处理时采用的是十进制, 因此时、分、秒单元加 1 后要进行十进制调整, 即要执行 DA A 指令, 还要注意的是计时到 24 就回零, 分和秒是计到 60 就回零。三是定时器 T0 计时初值的重置, 考虑到中断响应的时间, 重置时送 3CB4H, 以减小计时误差。四是中断返回前的现场恢复。

3) 显示子程序

显示采用的是动态显示, LED 显示器是共阳的, 位控信号输出时是低电平有效。在校“时”时, 采用的是点亮小数点做位调节标志, 哪位小数点亮表示调整的是该位的值。

显示子程序的第 1 部分是拆字, 显示缓冲区是 2FH~2AH; 第 2 部分是查字形代码, 输出段控和位控信号, 由于采用的是动态显示, 所以每输出一位的段控和位控信号要延时一定的时间, 使 LED 显示器显示的字符是稳定的。

4) 按键判断与按键处理程序

按键判断子程序在编写时应注意按键的去抖动, 该系统采用的是延时去抖动法, 延时是通过调用显示子程序来实现的, 每个键按下后要等待释放后再返回。

按键处理子程序中的按键是用于校时的, 所以进入按键处理子程序后就关闭定时中断, 对于功能键注意设置显示标志(将显示缓冲区某单元的 D4 位设为 1, 这样查显示代码时就查小数点亮的代码表)。

编程时还需要考虑出现程序跑飞的现象, 这时可在程序存储器空的单元中存放若干条 NOP 指令及在适当的地方存放 LIMP 0000H, 或者如本系统采用 LJMP NEXT 指令。这样程序不管跑飞到什么地方都能回到主程序中继续执行, 不至于出现死机现象。

根据以上分析, 设计的源程序如下:

```
ORG 0000H
AJMP MAIN
ORG 000BH
AJMP TIME
;以下是主程序
ORG 0100H
MAIN: MOV 20H, #00H           ;计时的 0.05s 到, 时、分和秒单元清 0
      MOV 21H, #00H
      MOV 22H, #00H
      MOV 23H, #00H
      MOV IP, #02H           ;中断优先级寄存器 IP 设置为定时中断 0 优先
                                ;开中断
      SETB EA
                                ;开定时/计数中断 T0
      SETB ET0
                                ;工作方式设置为定时器 T0 的方式 1
      MOV TMOD, #01H
```

MOV TL0, #0B0H	
MOV TH0, #3CH	
SETB TR0	;启动定时器工作
MOV SP, #40H	;堆栈指针设为 40H
NEXT: LCALL DISP	;调用显示子程序
LCALL KEY	;调按键判断子程序
JZ NEXT	;无键按下转 NEXT
LCALL ANKEY	;有键按下转按键处理子程序
SJMP NEXT	;按键处理完转 NEXT
NOP	
NOP	
NOP	
;以下是定时中断处理程序	
TIME: PUSH ACC	;现场保护
PUSH PSW	
MOV TL0, #0B4H	;重置初值
MOV TH0, #3CH	
INC 20H	;计时处理, 20H 中存放 50ms
MOV A, 20H	
CJNE A, #20, RETI1	;A 的值不等于 20, 则转 RETI1 处理程序
MOV 20H, #00H	;1s 到, 20H 清 0
MOV A, 21H	;将 21H(存放秒)的内容送 A
ADD A, #01H	;将 21H 中的内容加 1, 即定时 20 次后秒单元加 1
DA A	;十进制调整
MOV 21H, A	;将调整后的 A 送 21H
CJNE A, #60H, RETI1	;21H 中的内容不等于 60, 则转 RETI1 处理程序
MOV 21H, #00H	;1min 到, 21H(存放秒)清 0
MOV A, 22H	;将 22H(存放分)中的内容送 A
ADD A, #01H	;将 22H 中的内容加 1, 即 60s 后分单元加 1
DA A	;十进制调整
MOV 22H, A	;将 A 的内容送 22H
CJNE A, #60H, RETI1	;22H 中的内容不等于 60 则转 RETI1 处理程序
MOV 22H, #00H	;1h 时间到, 22H(存放分)清 0
MOV A, 23H	;将 23H(存放时)中的内容送 A
ADD A, #01H	;23H 中的内容加 1
DA A	;十进制调整
MOV 23H, A	;将 A 的内容送 23H

CJNE A, #24H, RETI1	;23H 中的内容不等于 24, 转 RET1 处理单元
MOV 23H, #00H	;时间到达 24h, 23H(存放时)清 0
RETI1: POP PSW	;恢复现场
POP ACC	
RETI	;中断返回
NOP	
NOP	
;以下是显示子程序	
DISP: ANL 2FH, #10H	;设置 2FH 的 D4 位为秒显示标志, 以便处理秒后的小数点
MOV A, 21H	;将 21H(存放秒)送 A
ANL A, #0FH	;取秒单元的低 4 位
ORL A, 2FH	;查带小数点的字形码, 处理秒后的小数点
MOV 2FH, A	;将秒单元的低 4 位送 2FH
MOV A, 21H	;将 21H(存放秒)送 A
ANL A, #0F0H	;取秒单元的高 4 位
SWAP A	;交换 A 的高 4 位和低 4 位
MOV 2EH, A	;将秒单元的高 4 位送 2EH
ANL 2DH, #10H	;设置 2DH 的 D4 位为分显示标志, 以便处理分后的小数点
MOV A, 22H	;将 22H(存放分)送 A
ANL A, #0FH	;取分单元的低 4 位
ORL A, 2DH	;查带小数点的字形码, 处理分后的小数点
MOV 2DH, A	;将分的低 4 位送 2DH
MOV A, 22H	;将 22H 送 A
ANL A, #0F0H	;取分单元的高 4 位
SWAP A	;交换 A 的高 4 位和低 4 位
MOV 2CH, A	;将分单元的高 4 位送 2CH
ANL 2BH, #10H	;设置 2BH 的 D4 位为时显示标志, 以便处理时后的小数点
MOV A, 23H	;将 23H(存放时)送 A
ANL A, #0FH	;取时单元的低 4 位
ORL A, 2BH	;查带小数点的字形码, 处理时后的小数点
MOV 2BH, A	;将时单元的低 4 位送 2BH
MOV A, 23H	;将 23H(存放时)的内容送 A
ANL A, #0F0H	;取时单元的高 4 位
SWAP A	;交换 A 的高 4 位和低 4 位
MOV 2AH, A	;将时单元的高 4 位送 2AH
MOV R0, #2FH	;显示偏移量

MOV R3, #06H	;将 6 个显示器循环显示
MOV DPTR, #TABLE	;指向表首址
MOV A, #11111011B	;将 1111 1011B 送 A, 准备动态扫描
LOOP1: MOV B, A	;将 A 存于 B
MOV P2, A	;将 A 送 P2 位选端
MOV A, @R0	;将 R0 指针地址送 A
MOVC A, @A+DPTR	;查表
MOV P0, A	;送显示
MOV R2, #80H	;延时
DJNZ R2, \$	
DEC R0	;R0 减 1, 以便进行时、分和秒的切换
MOV A, B	;将暂存在 B 中的内容送 A
RL A	;左移 1 位
DJNZ R3, LOOP1	;6 个显示器循环显示
RET	
TABLE: DB 0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H	
DB 00H, 00H, 00H, 00H, 00H, 00H	;若 2FH、2DH、2BH 的第 4 位为 0, 可以从以上 16 个不带小数点的字形码中查找
DB 40H, 79H, 24H, 30H, 19H, 12H, 2H, 78H, 00H, 10H	
DB 00H, 00H, 00H, 00H, 00H, 00H	;若 2FH、2DH、2BH 的第 4 位为 1, 可以从以上 16 个带小数点的字形码中查找
NOP	
NOP	
;按键判断子程序	
KEY: MOV P3, #0FFH	;向 P3 口写 1
MOV A, P3	;读 P3 口的状态, 按键按下时, 相应的位变为低电平
CPL A	;将 A 取反, 按键按下时, 相应的位变为高电平
ANL A, #00111100B	;将 A 与 0011 1100B 相与, 以获取(P3. 2~P3. 5)是否有键被按下
JZ RETX	;若 A 的值为 0, 说明无键按下, 转 RETX 处理程序
LCALL DISP;	;若 A 的值为 1, 说明有键按下, 调 DISP 显示子程序, 延时
LCALL DISP	
MOV A, P3	;再读 P3 口的状态
CPL A	

ANL A, #00111100B	
JZ RETX	;若 A 的值为 0,说明是键抖动
MOV R6, A	;若 A 的值为 1,说明有键按下,将键值存入 R6
LOOP2: LCALL DISP	;调显示子程序
MOV A, P3	;读 P3 口的状态
CPL A	
ANL A, #00111100B	
JNZ LOOP2	;若 A 的值不等于 0,说明键未释放,转 LOOP2 处理程序,等待键释放
MOV A, R6	;若 A 的值为 0,说明键已释放,将 R6 中的键值送 A
RETX: RET	
NOP	
NOP	
;按键处理子程序	
ANKEY: CLR EA	;关中断
MOV 25H, #00H	;将 25H 标志位清 0
LX: MOV A, R6	;将 R6 中的键值送 A
JB ACC. 2, L1	;若 ACC 的第 2 位是 1,说明是 P3. 2 外接的功能键按下,转 L1 处理程序
JB ACC. 3, L2	;若 ACC 的第 3 位是 1,说明是 P3. 3 外接的确认键按下,转 L2 处理程序
JB ACC. 4, L3	;若 ACC 的第 4 位是 1,说明是 P3. 4 外接的减 1 键按下,转 L3 处理程序
JNB ACC. 5, L12	;若 ACC 的第 5 位是 0,说明 P3. 5 外接的增 1 键未按,转 L12 处理程序
JB 2BH. 4, L6	;增 1 键按下,若 2BH 的第 4 位为 1,转 L6 处理程序(时单元加 1)
JB 2DH. 4, L8	;增 1 键按下,若 2DH 的第 4 位为 1,转 L8 处理程序(分单元加 1)
JB 2FH. 4, L9	;增 1 键按下,若 2FH 的第 4 位为 1,转 L9 处理程序(秒单元加 1)
L12: LCALL DISP	
LCALL DISP	
LCALL KEY	;判断有无键按下
JZ L12	
LJMP LX	
L2: MOV 25H, #00H	;确认键处理子程序
CLR 2BH. 4	

	CLR 2DH. 4	
	CLR 2FH. 4	
	SETB EA	
	RET	
L3:	JB 2BH. 4, L611	;减 1 键按下,若 2BH 的第 4 位为 1,转 L611 处理程序(时单元减 1)
	JB 2DH. 4, L811	;减 1 键按下,若 2DH 的第 4 位为 1,转 L811 处理程序(分单元减 1)
	JB 2FH. 4, L911	;减 1 键按下,若 2FH 的第 4 位为 1,转 L911 处理程序(秒单元减 1)
	AJMP L12	
L1:	MOV A, 25H	;功能键处理程序,将 25H(标志位)中的内容 送 A
	JZ LB1	;若 A 的值为 0,转 LB1 处理程序
	JB ACC. 0, LB2	;若 25H 的第 0 位为 1 转 LB2 处理程序,调整 分单元的值
	JB ACC. 1, LB3	;若 25H 的第 1 位为 1 转 LB3 处理程序,调整 秒单元的值
	JNB ACC. 2, L12	;若 25H 的第 2 位为 0,转 L12 处理程序返回,
LB1:	MOV 25H, #01H	;若 25H 的第 2 位为 1, (25H)=01H, 调整时 单元的值
	SETB 2BH. 4	;设置 2BH. 4 为 1
	CLR 2DH. 4	
	CLR 2FH. 4	
	AJMP L12	
LB3:	MOV 25H, #04H	;25H 单元是标志位, (25H)=04H, 调整秒单 元的值
	SETB 2FH. 4	;设置 2FH. 4 为 1
	CLR 2DH. 4	
	CLR 2BH. 4	
	AJMP L12	
LB2:	MOV 25H, #02H	;25H 单元是标志位, (25H)=02H, 调整分单 元的值
	SETB 2DH. 4	;设置 2DH. 4 为 1
	CLR 2BH. 4	
	CLR 2FH. 4	
	AJMP L12	
L6:	MOV A, 23H	;将 23H(存放时)中的内容送 A
	ADD A, #01H	;时单元加 1

	DA A	;十进制调整
	MOV 23H,A	;调整后送 23H
	CJNE A, #24H,L112	;23H 中的内容不等于 24H 则转 L112 处理程序
	MOV 23H, #00H	;23H 单元的内容等于 24H,23H 单元清 0
L112:	AJMP L12	
L8:	MOV A,22H	;将 22H(存放分)单元的内容送 A
	ADD A, #01H	;分单元加 1
	DA A	;十进制调整
	MOV 22H,A	;调整后送 22H
	CJNE A, #60H,L12	;22H 单元的内容不等于 60H,转 L112 处理程序
	MOV 22H, #00H	;22H 单元的内容等于 60H,22H 单元清 0
	AJMP L12	
L9:	MOV A,21H	;将 21H(存放秒)单元的内容送 A
	ADD A, #01H	;秒单元加 1
	DA A	;十进制调整
	MOV 21H,A	;调整后的内容送 21H
	CJNE A, #60H,L112	;21H 单元的内容不等于 60H,转 L112 处理程序
	MOV 21H, #00H	;21H 单元的内容等于 60H,21H 单元清 0
	AJMP L12	
L611:	MOV A,23H	;时单元减 1
	ADD A, #99H	
	DA A	
	MOV 23H,A	
	CJNE A, #99H,L112	
	MOV 23H, #23H	
	AJMP L12	
L811:	MOV A,22H	;分单元减 1
	ADD A, #99H	
	DA A	
	MOV 22H,A	
	CJNE A, #99H,L112	
	MOV 22H, #59H	
	AJMP L12	
L911:	MOV A,21H	;秒单元减 1
	ADD A, #99H	
	DA A	

```
MOV 21H,A
CJNE A, #99H,L112
MOV 21H, #59H
AJMP L12
NOP
NOP
END
```

四、用下载型实验板进行仿真实验

以上介绍的程序可以在下载型实验板上进行仿真调试,方法如下:

(1)将 MON51 硬件仿真器和 PC 机连接好。

(2)取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

(3)给实验板和仿真器通电。

(4)打开 Keil 软件,输入上面的程序,保存为 clock.asm。对程序进行编译、链接,调试运行。

该实验程序在本书所附光盘的 example\ch_9\clock 文件夹中。

五、系统功能的扩充

前面所介绍的是最基本的时钟,如要增加打铃功能,可以加一个输出信号,并由 P1 口的某一位输出,还需要将打铃的时间表存入程序存储器。这时在中断服务程序中,每当分单元加 1 时就要与时间表进行比较,判断是不是要进行打铃输出,从而设置打铃输出的标志位,在主程序中增加一段对打铃输出标志的判断程序,有打铃输出标志,则 P1.0 输出低电平使铃响,否则输出高电平,铃不响,并且还要控制打铃输出的时间,打铃输出也可用发光二极管模拟指示。

第三节 单片机应用系统的可靠性设计

一个成熟可靠的单片机系统,除了单片机及应用软件的设计外,还涉及到各种类型的外围接口电路的设计,如模拟电路、数字电路等。仔细划分又有驱动电路、A/D 转换、D/A 转换、功率接口控制、电源甚至印制电路板的布线等。由于各个组成部分的电路结构不尽相同,并且系统所处环境的差异,以至于同样结构和电路的单片机系统在不同的环境下的可靠性和稳定性大不相同,这就涉及到系统可靠性的问题。而一个单片机系统的可靠性又和系统的抗干扰能力有关。下面从单片机设计入手,简要分析单片机系统的抗干扰措施。

一、提高单片机系统稳定性的硬件措施

一个稳定的单片机应用系统是由硬件和软件组成,因此系统稳定性也要从软、硬件这两个方面来分析。由于单片机的软件设计在很大程度上是由程序员的经验和水平决定的,是一个长期的时间积累过程,在短时间内不可能有本质的提高,因此从设计者的角度

来看,通过较为完善的硬件设计来弥补软件设计的不足是一个比较明智的做法。为提高系统的抗干扰能力,在硬件设计时一般要注意以下 3 个方面。

1. 单片机及其相关元器件的选择

一个单片机系统的核心是处理器,如果处理器的稳定性和可靠性比较差,那么整个系统的可靠性也不可能得到保证。目前,市面上流行的单片机型号不下几十种,而 8 位单片机则是我国单片机市场的主流产品。从国内流行的产品来看,MCS-51 系列及其兼容机型仍占主流,国内常用的有:ATMEL 公司的 89C5x、89S5xx 系列,WINBOND 公司的 W77E5x、W78E5x 系列,PHILIPS 公司的 P87LPC7x、P89C5x、P87C5x 系列,SST 公司的 SST89C5x 系列,CYGNAL 公司的 C8051F 系列。非 51 系列的 8 位单片机在中国应用较广的有 MOTOROLA 的 M68HC05/08 系列、微芯公司的 PIC 单片机以及 ATMEL 公司的 AVR 单片机。为提高单片机系统的抗电磁干扰能力,使产品能适应恶劣的工作环境,满足电磁兼容性方面更高标准的要求,各个单片机厂家在设计单片机内部电路时均采取了一些新的技术措施:一些新兴的单片机还在单片机内部增加了看门狗定时器,如 AT89C51 的换代产品 AT89S51。有的单片机还内置电源检测和复位电路,如微芯公司的某些型号的 PIC 单片机,这些措施都大大增强了单片机自身的抗干扰能力。因此从选型上来看,如果不太过分计较成本,可以考虑选择一些新型的单片机,如:ATMEL 公司新近推出的 AT89S51,不但内含看门狗定时器,而且可以关闭 ALE 信号以提高系统的可靠性;PHILIPS 公司的 89C51RD2 单片机除了可以关闭 ALE 信号外,还改善了单片机的内部时序,6 个时钟周期为 1 个机器周期,在不提高时钟频率的条件下,提高了处理器运算速度,自身的抗干扰能力也得到了加强。如果设计的系统要在环境条件非常恶劣的条件下工作,可以选择 MOTOROLA 公司的一些产品,其芯片的可靠性和稳定性已在国内工控界得到普遍认可。PIC 单片机则在家电产品中得到了广泛应用,它采用的一些技术如 OTP(一次性可编程)、RISC 结构等是保证其产品稳定性的基础。

除了 MCU 的选择外,其他元器件的选择也很重要,如半导体二极管、三极管以及集成电路各项电气参数应能满足系统性能的基本要求。在环境比较恶劣的场合还应考虑温度对系统的影响,应尽量选择温漂系数小、稳定性好的器件。如果能使用集成电路,则尽量不采用分立器件。在集成电路的选择上也有一个基本的原则,一般情况下,CMOS 数字集成电路的抗干扰能力要强于 TTL 集成电路。这是因为 CMOS 数字集成电路的噪声容限较 TTL 的高,因而比较而言其抗干扰能力也强。对于常用的 TTL 门电路,其抗干扰能力也有区别,54 系列集成电路的工作温度和电源电压都比 74 系列的高,一般应用于环境较为恶劣的场合,抗干扰的能力也高于 74 系列。在使用 CMOS 芯片时,由于 CMOS 芯片的输入电阻极大,对于干扰信号比较敏感,因此电路不用的输入引脚不可开路,可以根据实际情况将输入端接电源或直接接地,否则,很容易增加 CMOS 芯片的功耗。严重的会导致芯片被静电击穿。另外,为了兼顾 TTL 芯片的高速度和 CMOS 芯片的低功耗,可选用 74HC 系列的集成电路,如果要兼顾二者,可采用 74HCT 系列的芯片。在系统总体设计时,应包括电源监控及看门狗电路,如 IMP809、IMP706、IMP813、X5043、X5045 等,也可大幅度提高整个电路的抗干扰性能。

2. 电路板 PCB 布线的可靠性设计

印制电路板设计的好坏直接影响到单片机系统的稳定性和可靠性。随着单片机技术

的不断发展,PCB 布线的密度也越来越高。PCB 设计的好坏对抗干扰能力影响极大。因此,在进行印制电路板 PCB 设计时,要使系统获得最佳性能,元器件的布置及导线的布设相当重要,必须遵守 PCB 设计的一般原则,并应符合抗干扰设计的要求,以下规则是设计电路板必须遵守的。

1) 总体布局

电路板的总体布局要合理分区,强、弱信号,数字、模拟信号一定要分区布置。在电路设计中尽可能把干扰源(如电机、继电器)与敏感元件(如单片机)远离,易受干扰的元器件不能相互挨得太近,输入和输出元件应尽量远离。尤其要注意高压电路部分的元器件与低压部分要分隔开放置,如晶闸管控制交流终端的器件一定要远离单片机,如有可能,最好将高压电路部分的元器件独立布板,这样可减少许多不必要的麻烦。在设计单片机系统的电路板时,应尽可能以单片机为中心,按照电路的流程安排各个功能电路单元的位置,使布局便于信号流通,并使信号尽可能保持一致的方向。如果系统有高频器件,应尽可能缩短高频元器件之间的连线,设法减少它们的分布参数和相互间的电磁干扰。

2) 布线

PCB 导线的布设应尽可能得短,印制导线的拐弯如能成圆角则不采用直角形式,以减小高频信号对外的发射与耦合。布置双面板时,正反两面的导线应采用垂直布线,避免相互平行,以减小寄生耦合。走线的宽度应能满足电气性能要求,导线宽度在大电流情况下还要考虑其温度。通常情况下(经验值)1mm 宽度走线的最大承载电流为 1A~2A(根据板材而定),信号线可选择为 0.25mm~0.3mm。在高密度、高精度的印制线路中,导线宽度和间距一般可取 0.3mm。为了提高系统的抗干扰能力,应采取线路板全局性环型屏蔽,并尽可能让地线和电源线宽一些。走线导线的间距必须能满足电气安全要求,最小间距至少要能满足所承受的电压,这个电压一般包括工作电压、附加波动电压以及其他原因引起的峰值电压,如条件允许,间距应尽量宽些。由于电路板的一个过孔会带来大约 10pF 的电容效应,这对于高频电路,将引入极多的干扰,所以在布线的时候,应尽可能地减少过孔的数量。

3) 接地

在地线的布置上,数字地和模拟地要分开布置,最后都要接到电源地上,尤其是在设计 A/D、D/A 转换电路时一定要遵守该规则,否则可能会大幅度降低 A/D 采样的精度。其次,地线应尽量加宽、加粗,若线径很细,接地电位则随电流的变化而变化,会造成系统的抗噪声性能变坏。因此应将接地线尽量加粗,如有可能,接地线的宽度应大于 3mm。最后,接地线最好构成闭环路,这是因为印制电路板上的集成电路元件在流过大电流时,因受接地线粗细的限制,会在地线上产生较大的电位差,引起抗噪声能力下降,若将接地线构成环路,则会缩小电位差值,提高电子设备的抗噪声能力,在设计数字电路时更应如此。图 9-5 为正确的地线布置示意图。

3. 采取的抗干扰措施

1) 滤波和退耦

充分考虑电源对单片机的影响,电源做得好,整个电路的抗干扰就解决了一大半。单片机对电源噪声比较敏感,应给电源加滤波电路或稳压器,以减小电源噪声对单片机的干扰;可在单片机电路板的 V_{CC} 入口处并联一个数百微法和 0.1 μ F 的电容器以减少电源的高

频和低频干扰。其次,最好在每个集成电路处并接一个 $0.01\mu\text{F}\sim 0.1\mu\text{F}$ 高频退耦电容,以减小集成电路对电源的影响,正确的退耦电容的布置如图 9-6 所示,退耦电容的接地端应直接到地,不能多点接地,否则会增大退耦电容的等效串联电阻,影响滤波效果。

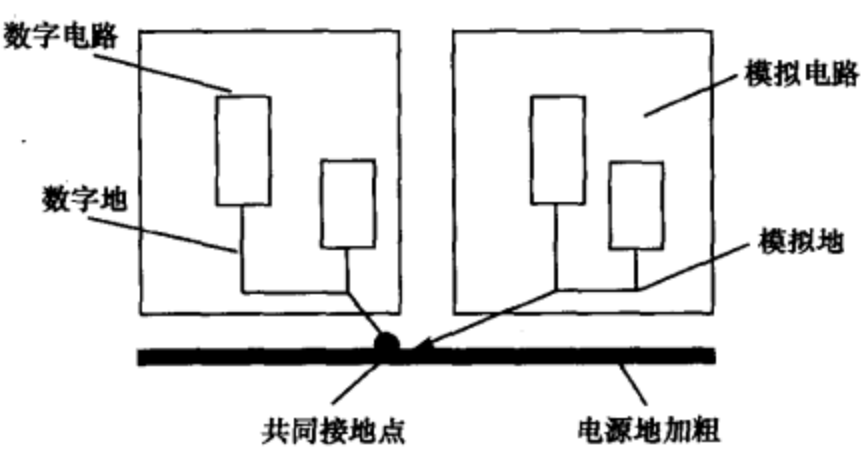


图 9-5 正确的地线布置示意图

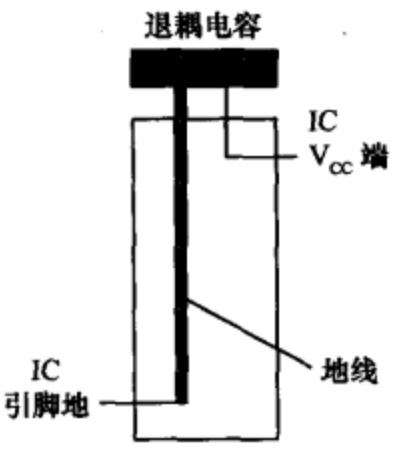


图 9-6 退耦电容的接法

2)抑制干扰源

常用的方法有:在继电器线圈处增加续流二极管,消除断开线圈时产生的反电动势干扰,如图 9-7 所示的二极管 D11。在继电器接点两端并接火花抑制电路,减小电火花影响,如图 9-7 中的 $0.1\mu\text{F}$ 的电容。

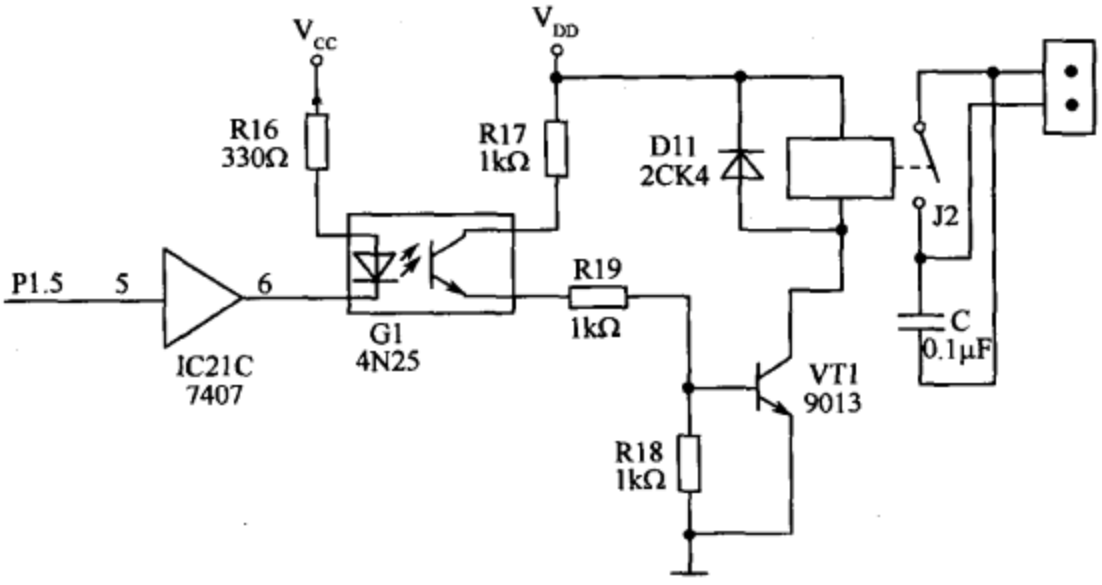


图 9-7 续流二极管和火花抑制电路

3)一些经验

对于单片机闲置的 I/O 接口,不要悬空,要接地或接电源。其他 IC 的闲置端在不改变系统逻辑的情况下也应接地或接电源。单片机晶体的引脚要尽量短,外壳要焊接到电路板的接地端,这样可大大减少一些莫名其妙的问题。在速度和性能满足设计要求的前提下,尽量降低单片机的晶振频率和选用低速数字电路。在控制功率器件时,如通过继电器、晶闸管控制电机或其他容性和感性负载时,应对输出通道进行光耦隔离,如有条件,尽量采用固态继电器。图 9-7 中,光耦 4N25 的作用就是为了隔离前后通道信号间的相互干扰。另外,现在流行的一些 A/D 转换芯片,如 TLC2543、TLC5510 均有 AGND、DGND 两个接地端,分别对应模拟地和数字地。AGND、DGND 在芯片的内部一般不连接,只有通过外部引线相接。在设计电路板时,AGND、DGND 线应通过独立的电源线单独走线,可采用屏蔽良好的双绞线,最后统一接到电源地。AGND、DGND 应当分别用 $0.1\mu\text{F}$ 的

电容去耦,电容应尽量靠近 AGND 和 DGND 引脚。同时,模拟输入信号端、数字信号输出端应严格与 AGND 走线隔离,不得交叉,以防止 AGND 上的杂波信号对输入、输出端形成干扰。若有条件,AGND 的走线最好放在模拟信号输入端以形成屏蔽。

二、提高单片机系统稳定性的软件措施

一个成熟的单片机应用系统中,硬件设计是系统抗干扰能力的基础,而软件抗干扰则是对硬件的补充。这些措施往往是一些有经验的程序员长期积累的结晶。软件抗干扰措施一般包括有开机自检、软件陷阱、指令冗余、软件滤波、软件看门狗等。

1. 开机自检

开机自检程序通常包括对 RAM、ROM、I/O 接口状态等的检测。在程序编制中,可将 RAM 或 ROM 区中的重要内容分区存放,在程序运行的初始或中间过程中经常对这些数据进行比较检查,如发现数据出错,则重写这些数据。

2. 软件陷阱

软件陷阱就是在程序存储器的未使用区域中,加上若干条空操作和无条件跳转指令,无条件跳转指令指向复位入口地址。如果程序跳到这些未用区域,就通过强行执行无条件跳转指令,转到复位入口地址。例如,在 0200H 以后的程序区均未使用,可在该区域用 NOP 和 LJMP 0000H 指令填充,程序如下:

```
ORG 0000H
AJMP MAIN
;主程序区
MAIN:
:
ORG 0200H
NOP
NOP
LJMP MAIN
```

3. 指令冗余

指令冗余与软件陷阱有点相似,只是软件陷阱用在程序存储器的未使用区域中,而指令冗余通常在程序区中。一般的做法是在十几条正常的指令后填充 2 个~3 个 NOP 空指令,尤其是在 LCALL、ACALL、RET、RETI、LJMP 等一些跳转和子程序调用指令的前面,如果能加上几条 NOP 空指令则对于程序的正确流向会起到一定的保护作用。

4. 软件滤波

在数据采集系统中,可以通过软件滤波来达到提高系统的数据采集精度的作用。软件滤波包括算术平均法、中值滤波法和 RC 低通滤波法等。算术平均法就是对一点数据连续采样多次;然后取其平均,可以减少系统随机干扰对数据采集造成的影响。中值滤波法则是对数据采集奇数次,取其中间值,可以减少错误概率。RC 低通滤波法则是用软件模拟低通滤波器,对周期性干扰有比较好的效果。

5. 软件看门狗

软件看门狗与常用的硬件看门狗一样,都是为了更好地监测程序的运行。软件看门

狗一般要占用单片机系统中的定时器,如在 51 系统中可使用 T0~T1,并且将这个定时器设置为最高级中断。在主程序中,要根据定时器的溢出周期对定时器进行初始化,一旦程序受到干扰跑飞,则在中断子程序(如下例中的 ERRINT)里设置一条出错跳转指令,将程序转移到 ERR 出错子程序中,在 ERR 出错子程序中完成整个程序的初始化过程,使程序从头执行。例如,下面的例子是用 T0 作为看门狗定时器。

```

    ORG 0000H
    AJMP MAIN
    ORG 000BH
    AJMP ERRINT
;以下是主程序
MAIN:  MOV TH0, #**           ;T0 参数的初始化
       MOV TL0, #**
       AJMP MAIN
;以下是出错中断子程序
ERRINT: CLR EA
        LJMP ERR              ;程序转移到出错子程序 ERR
        :
        RETI
;以下是出错子程序
ERR:    :
        RET

```

;出错子程序中完成整个程序的初始化过程

当然,从系统可靠性的角度上讲,不管是采用软件看门狗还是硬件看门狗,也仅仅是事后有作用,即常说的亡羊补牢,不能从根本上防止程序出错。要真正解决单片机系统的抗干扰难题,则要从软件、硬件两个方面着手,从设计这个源头开始,处处防患于未然,采取多种措施,从根本上解决这个问题。



附录 MCS51 指令表汇总

十六进制代码	助 记 符	功 能	对标志影响				字节数	周期数
			P	OV	AC	CY		
算术运算指令								
28~2F	ADD A, Rn	$(A) + (Rn) \rightarrow A$	✓	✓	✓	✓	1	1
25	ADD A, direct	$(A) + (\text{direct}) \rightarrow A$	✓	✓	✓	✓	2	1
26, 27	ADD A, @Ri	$(A) + (Ri) \rightarrow A$	✓	✓	✓	✓	1	1
24	ADD A, # data	$(A) + \text{data} \rightarrow A$	✓	✓	✓	✓	2	1
38~3F	ADDC A, Rn	$(A) + (Rn) + CY \rightarrow A$	✓	✓	✓	✓	1	1
35	ADDC A, direct	$(A) + (\text{direct}) + CY \rightarrow A$	✓	✓	✓	✓	2	1
36, 37	ADDC A, @Ri	$(A) + (Ri) + CY \rightarrow A$	✓	✓	✓	✓	1	1
34	ADD A, # data	$(A) + \text{data} + CY \rightarrow A$	✓	✓	✓	✓	2	1
98~9F	SUBB A, Rn	$(A) - (Rn) - CY \rightarrow A$	✓	✓	✓	✓	1	1
95	SUBB A, direct	$(A) - (\text{direct}) - CY \rightarrow A$	✓	✓	✓	✓	2	1
96, 97	SUBB A, @Ri	$(A) - ((Ri)) - CY \rightarrow A$	✓	✓	✓	✓	1	1
94	SUBB A, # data	$(A) - \text{data} - CY \rightarrow A$	✓	✓	✓	✓	2	1
04	INC A	$(A) + 1 \rightarrow A$	✓	×	×	×	1	1
08~0F	INC Rn	$(Rn) + 1 \rightarrow Rn$	×	×	×	×	1	1
05	INC direct	$(\text{direct}) + 1 \rightarrow \text{direct}$	×	×	×	×	2	1
06, 07	INC @Ri	$((Ri)) + 1 \rightarrow (Ri)$	×	×	×	×	1	1
A3	INC DPTR	$(DPTR) + 1 \rightarrow DPTR$					1	2
14	DEC A	$(A) - 1 \rightarrow A$	✓	×	×	×	1	1
18~1F	DEC Rn	$(Rn) - 1 \rightarrow Rn$	×	×	×	×	1	1
15	DEC direct	$(\text{direct}) - 1 \rightarrow \text{direct}$	×	×	×	×	2	1
16, 17	DEC @Ri	$((Ri)) - 1 \rightarrow (Ri)$	×	×	×	×	1	1
A4	MUL AB	$(A) * (B) \rightarrow AB$	✓	✓	×	✓	1	4
84	DIV AB	$(A) / (B) \rightarrow AB$	✓	✓	×	✓	1	4
D4	DA A	对 A 进行十进制调整	✓	×	×	✓	1	1
逻辑运算指令								
58~5F	ANL A, Rn	$(A) \wedge (Rn) \rightarrow A$	✓	×	×	×	1	2
55	ANL A, direct	$(A) \wedge (\text{direct}) \rightarrow A$	✓	×	×	×	2	1
56, 57	ANL A, @Ri	$(A) \wedge ((Ri)) \rightarrow A$	✓	×	×	×	1	1

(续)

十六进制代码	助 记 符	功 能	对标志影响				字节数	周期数
			P	OV	AC	CY		
逻辑运算指令								
54	ANL A, # data	$(A) \wedge \text{data} \rightarrow A$	✓	×	×	×	2	1
52	ANL direct, A	$(\text{direct}) \wedge (A) \rightarrow \text{direct}$	×	×	×	×	2	1
53	ANL direct, # data	$(\text{direct}) \wedge \text{data} \rightarrow \text{direct}$	×	×	×	×	3	2
48~4F	ORL A, Rn	$(A) \vee (Rn) \rightarrow A$	✓	×	×	×	3	2
45	ORL A, direct	$(A) \vee (\text{direct}) \rightarrow A$	✓	×	×	×	1	1
46,47	ORL A, @Ri	$(A) \vee ((Ri)) \rightarrow A$	✓	×	×	×	2	1
44	ORL A, # data	$(A) \vee \text{data} \rightarrow A$	✓	×	×	×	1	1
42	ORL direct, A	$(\text{direct}) \vee (A) \rightarrow \text{direct}$	×	×	×	×	2	1
43	ORL direct, # data	$(\text{direct}) \vee \text{data} \rightarrow \text{direct}$	×	×	×	×	2	1
68~6F	XRL A, Rn	$(A) \oplus (Rn) \rightarrow A$	✓	×	×	×	3	2
65	XRL A, direct	$(A) \oplus (\text{direct}) \rightarrow A$	✓	×	×	×	1	1
66,67	XRL A, @Ri	$(A) \oplus ((Ri)) \rightarrow A$	✓	×	×	×	2	1
64	XRL A, # data	$(A) \oplus \text{data} \rightarrow A$	✓	×	×	×	1	1
62	XRL direct, A	$(\text{direct}) \oplus (A) \rightarrow \text{direct}$	×	×	×	×	2	1
63	XRL direct, # data	$(\text{direct}) \oplus \text{data} \rightarrow \text{direct}$	×	×	×	×	2	1
E4	CLR A	$0 \rightarrow A$	✓	×	×	×	3	2
F4	CLR A	$(\bar{A}) \rightarrow A$	×	×	×	×	1	1
23	RL A	A 循环左移 1 位	×	×	×	×	1	1
33	RLC A	A 带进位循环左移 1 位	✓	×	×	✓	1	1
03	RR A	A 循环右移 1 位	×	×	×	×	1	1
13	RRC A	A 带进位循环右移 1 位	✓	×	×	✓	1	1
C4	SWAP A	A 半字节交换	×	×	×	×	1	1
数据传送指令								
E8~EF	MOV A, Rn	$(Rn) \rightarrow A$	✓	×	×	×	1	1
E5	MOV A, direct	$(\text{direct}) \rightarrow A$	✓	×	×	×	2	1
E6,E7	MOV A, @Ri	$((Ri)) \rightarrow A$	✓	×	×	×	1	1
74	MOV A, # data	$\text{data} \rightarrow A$	✓	×	×	×	2	1
F8~FF	MOV Rn, A	$(A) \rightarrow Rn$	×	×	×	×	1	1
A8~AF	MOV Rn, direct	$(\text{direct}) \rightarrow Rn$	×	×	×	×	2	2
78~7F	MOV Rn, # data	$\text{data} \rightarrow Rn$	×	×	×	×	2	1
F5	MOV direct, A	$(A) \rightarrow \text{direct}$	×	×	×	×	2	1
88~8F	MOV direct, Rn	$(Rn) \rightarrow \text{direct}$	×	×	×	×	2	1
85	MOV direct 1, direct 2	$(\text{direct } 2) \rightarrow \text{direct } 1$	×	×	×	×	2	2
86,87	MOV direct, @Ri	$((Ri)) \rightarrow \text{direct}$	×	×	×	×	3	2

(续)

十六进制代码	助 记 符	功 能	对标志影响				字节数	周期数
			P	OV	AC	CY		
数据传送指令								
75	MOV direct, # data	data→direct	×	×	×	×	2	2
F6, F7	MOV @Ri, A	(A)→((Ri))	×	×	×	×	3	2
A6, A7	MOV @Ri, direct	(direct)→(Ri)	×	×	×	×	1	1
76, 77	MOV @Ri, # data	data→(Ri)	×	×	×	×	2	2
90	MOV DPTR, # data 16	data 16→DPTR	×	×	×	×	2	1
93	MOVC A, @A+DPTR	((A)+(DPTR))→A	✓	×	×	×	3	2
83	MOVC A, @A+PC	((A)+(PC))→A	✓	×	×	×	1	2
E2, E3	MOVX A, @Ri	((Ri)+(P2))→A	✓	×	×	×	1	2
E0	MOVX A, @DPTR	((DPTR))→A	✓	×	×	×	1	2
F2, F3	MOVX @Ri, A	(A)→(Ri)+(P2)	×	×	×	×	1	2
F0	MOVX @DPTR, A	(A)→(DPTR)	×	×	×	×	1	2
C0	PUSH direct	(SP)+1→SP (direct)→(SP)	×	×	×	×	2	2
D0	POP direct	((SP))→direct (SP)-1→SP	×	×	×	×	2	2
C8~CF	XCH A, Rn	(A)↔(Rn)	✓	×	×	×	1	1
C5	XCH A, direct	(A)↔(direct)	✓	×	×	×	2	1
C6, C7	XCH A, @Ri	(A)↔((Ri))	✓	×	×	×	1	1
D6, D7	XCHD A, @Ri	(A)0~3↔((Ri))0~3	✓	×	×	×	1	1
位操作指令								
C3	CLR C	0→cy	×	×	×	✓	1	1
C2	CLR bit	0→bit	×	×	×		2	1
D3	SETB C	1→cy	×	×	×	✓	1	1
D2	SETB bit	1→bit	×	×	×		2	1
B3	CPL C	$\overline{\text{cy}} \rightarrow \text{cy}$	×	×	×	✓	1	1
B2	CPL bit	$\overline{\text{bit}} \rightarrow \text{bit}$	×	×	×		2	1
82	ANL C, bit	(cy) ∧ (bit)→cy	×	×	×	✓	2	2
B0	ANL C, /bit	(cy) ∧ $\overline{(\text{bit})} \rightarrow \text{cy}$	×	×	×	✓	2	2
72	ORL C, bit	(cy) ∨ (bit)→cy	×	×	×	✓	2	2
A0	ORL C, /bit	(cy) ∨ $\overline{(\text{bit})} \rightarrow \text{cy}$	×	×	×	✓	2	2
A2	MOV C, bit	(bit)→cy	×	×	×	✓	2	1
92	MOV bit, C	cy→bit	×	×	×	×	2	2
控制转移指令								
1	ACALL acklrl	(PC)+2→PC, (SP)+1→SP (PC)L→(CP) (SP)+1→SP, (PC)H→(SP) addrll→PC10~0	×	×	×	×	2	2

(续)

十六进制代码	助 记 符	功 能	对标志影响				字节数	周期数
			P	OV	AC	CY		
控制转移指令								
12	LCALL addr 16	(PC)+2→PC, (SP)+1→SP (PC)L→(CP), (SP)+1→SP (PC)H→(SP), ack16→PC	×	×	×	×	3	2
22	RET	((SP))→PCH, (SP)-1→SP ((SP))→PCL, (SP)-1→SP	×	×	×	×	1	2
32	RET1	((SP))→PCH, (SP)-1→SP ((SP))→PCL, (SP)-1→SP 从中断返回	×	×	×	×	1	2
1	AJMP addr11	addr11→PC10~0	×	×	×	×	2	2
02	LJMP addr16	addr16→PC	×	×	×	×	3	2
80	SJMP rel	(PC)+(rel)→PC	×	×	×	×	2	2
73	JMP @A+DPTR	(A)+(DPTR)→PC	×	×	×	×	1	2
60	JZ rel	(PC)+2→PC, 若(A)=0, 则 (PC)+(rel)→PC	×	×	×	×	2	2
70	JNZ rel	(PC)+2→PC, 若(A)≠0, 则 (PC)+(rel)→PC	×	×	×	×	2	2
40	JC rel	(PC)+2→PC, 若 cy=1, 则 (PC)+(rel)→PC	×	×	×	×	2	2
50	JNC rel	(PC)+2→PC, 若 cy=0, 则 (PC)+(rel)→PC	×	×	×	×	2	2
20	JB bit, rel	(PC)+3→PC, 若 bit=1, 则 (PC)+(rel)→PC	×	×	×	×	3	2
30	JNB bit, rel	(PC)+3→PC, 若(bit)=0, 则 (PC)+(rel)→PC	×	×	×	×	3	2
10	JBC bit, rel	(PC)+3→PC, 若(bit)=1, 则 0+bit, (PC)+(rel)→PC	×	×	×	×	3	2
B5	CJNE A, direct, rel	(PC)+3→PC, 若(A)不等于 (direct), 则(PC)+(rel)→PC; 若(A)<(direct), 则1→cy	×	×	×	✓	3	2
B4	CJNE A, #data, rel	(PC)+3→PC, 若(A)不等于 data, 则(PC)+(rel)→PC; 若 (A)小于 data, 则1→cy	×	×	×	✓	3	2
B8~BF	CJNE Rn, #data, rel	(PC)+3→PC, 若(Rn)不等于 data, 则(PC)+rel→PC; 若 (Rn)小于 data, 则1→cy	×	×	×	✓	3	2

(续)

十六进制代码	助 记 符	功 能	对标志影响				字 节 数	周 期 数
			P	OV	AC	CY		
控制转移指令								
B6, B7	CJNE @Ri, # data, rel	(PC)+3→PC, 若(Ri)不等于 data, 则 (PC) + rel→PC; 若 (Rn)小于 data, 则 1→cy	×	×	×	✓	3	2
D8~DF	DJNZ Rn, rel	(PC)+3→PC, (Rn)-1→Rn, 若(Rn)不等于 0, 则 (PC) + rel→PC	×	×	×	×	2	2
D5	DJNZ direct, rel	(PC)+3→PC, (direct)-1→direct, 若 (direct) 不等于 0。则 (PC) + rel→PC	×	×	×	×	3	2
00	NOP	空操作	×	×	×	×	1	1



参考文献

- [1] 李广弟,朱月秀,王秀山. 单片机基础. 北京:北京航空航天大学出版社,2001.
- [2] 周坚. 单片机轻松入门. 北京:北京航空航天大学出版社,2004.
- [3] 孟凤果,曹振军. 单片机应用自学通. 北京:中国电力出版社,2005.
- [4] 肖洪兵,胡辉,郭速学. 跟我学用单片机. 北京:北京航空航天大学出版社,2002.
- [5] 周立功,等. 增强型 80C51 单片机速成与实践. 北京:北京航空航天大学出版社,2003.
- [6] 朱宇光,等. 单片机应用新技术教程. 北京:电子工业出版社,2000.
- [7] 林伸茂,等. 8051 单片机彻底研究基础篇. 北京:人民邮电出版社,2004.
- [8] 林伸茂,等. 8051 单片机彻底研究经验篇. 北京:人民邮电出版社,2004.
- [9] 林伸茂,等. 8051 单片机彻底研究实习篇. 北京:人民邮电出版社,2004.
- [10] 冯建华,赵亮. 单片机应用系统设计与产品开发. 北京:人民邮电出版社,2004.
- [11] 楼然苗,李光飞. 51 系列单片机设计实例. 北京:北京航空航天大学出版社,2003.

