

普通高等教育“十三五”规划教材
电子设计系列规划教材

单片机原理及应用

(第3版)

张迎新 王盛军 邢春香 等编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书以 80C51 系列中的 89 系列单片机为例介绍单片机的硬件结构、工作原理、指令系统、汇编语言及 C 语言程序设计、接口技术、中断系统及单片机应用等内容。主要包括：概况、单片机结构及原理、指令系统、汇编语言程序设计、C51 语言程序及开发环境、定时/计数器、80C51 的串行接口、中断系统、单片机的系统扩展、接口技术、单片机应用系统的设计与开发等。本书在各章中对关键性内容都结合实例予以说明，并附大量思考题和习题，配套电子课件、程序代码、参考答案等。

本书可作为高等学校本专科单片机课程的教材，也可作为相关领域科技人员学习开发单片机的参考书。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

单片机原理及应用 / 张迎新等编著. —3 版. —北京：电子工业出版社，2017.8

电子设计系列规划教材

ISBN 978-7-121-32305-8

单... 张... 单片微型计算机—教材 TP368.1

中国版本图书馆 CIP 数据核字（2017）第 182462 号

策划编辑：王羽佳

责任编辑：裴 杰

印 刷：

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：20 字数：512 千字

版 次：2004 年 10 月第 1 版

2017 年 8 月第 3 版

印 次：2017 年 8 月第 1 次印刷

定 价：49.90 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254535，wylj@phei.com.cn。

前 言

《单片机原理及应用》第2版自2009.2发行以来,已多次印刷,说明此书有一定的读者群。在此期间单片机技术又有了很大的发展,因而电子出版社决定对原书内容进行修订更新。在重新编写时,对原书内容作了多处改进。在本书的实例中多数都采用汇编与C语言双语编程,使读者更易于从中比较两种语言的特长,可以有选择地掌握一种,并认识另一种。

本书还增加了Proteus硬件仿真环境的内容,Keil C51和Proteus仿真软件是在单片机应用系统设计中使用的最广泛的软件。Proteus是一种电子设计自动化软件,它不仅能完成各种电路的设计与仿真,还能仿真单片机及其外围电路系统。这些软件使单片机的学习更加简单易懂,是学校进行单片机教学的首选软件。

此外,对部分章节内容进行了调整与修改,例如修改了C51语言章节与Keil集成开发环境编排顺序,使学生更容易掌握汇编与C51语言,使得单片机的学习更容易入门。对于各章节中比较过时的内容做了删减,例如删除了关于8255接口的内容(8255已经停产多年)。对于全书均做了适当改进,力求重点更突出,语言更精炼,表述更清晰。

目前物联网与移动互联技术正迅速地改变着世界,随着嵌入式技术的发展,32位MCU(单片机)成为主流,64位MCU也开始进入这个市场,但是8位MCU的市场占有率并没有降低,至今仍然是开发嵌入式系统的主要器件,物联网给MCU应用创造了更多机会,人工智能的感知与控制终端领域的多样性,需要功能特点不同的MCU,所以不论是8位还是32位MCU都有用武之地。一些厂商一直致力于8位MCU的创新与研发,使8位MCU仍然保持旺盛的生命力。

因为以8051为内核的80C51系列单片机在世界上生产量比较大,派生产品比较多,可以满足大多数用户的需要;而且80C51的软件工具也比较成熟,因而80C51系列单片机仍然是单片机教学的首选机型,特别是80C51系列中的典型型号在基本结构、工作原理和引脚上与MCS-51系列单片机的8051是完全兼容的。基于这种情况,本书在介绍单片机时,是以80C51系列为例进行讲述的。而在介绍具体型号时选用了美国ATMEL公司的AT89系列产品。AT89系列单片机的成功使得几个著名的半导体厂家也相继生产了类似的产品,例如,PHILIPS的P89系列、美国SST公司的SST89系列等,后来人们就简称这一类产品为89系列单片机,它实际上还是属于80C51系列。89系列单片机虽然并不是功能最强、最先进的单片机,但它是源于经典的MCS-51系列,从教学角度看,由于8位单片机具有稳定、便宜、易用等特点,考虑到教学的连续性及89系列单片机和所用开发装置的普及性,因而本书的单片机芯片实例将采用ATMEL公司的AT89S51/52单片机,在做一般共性介绍时还是用80C51符号代表。

本教材在章节的安排顺序和内容上都有不同程度的改进。第1章概述,增加了存储器的介绍,第2章以AT89S51/52单片机为例介绍单片机的结构及原理,第3章介绍指令系统,第4章介绍汇编语言程序设计,第5章介绍C51语言程序及开发环境,增加了Keil集成开发环境与Proteus仿真软件的内容,第6章介绍定时/计数器,第7章介绍串行接口,第8章介绍中断系统,第9章介绍系统扩展方法,第10章介绍接口技术,第11章介绍单片机应用系统设计。为了能给读者较多的应用实例和方法,同时又不致于使篇幅过长,在举例时对于关键和核心的内容尽量讲透,其他内容点到为止。

本书提供配套电子课件、程序代码和参考答案，请登录华信教育资源网（<http://www.hxedu.com.cn>）注册下载。

本书是作者多年教学和科研的积累，同时为了使本书的内容更加丰富和完整，书中也引用了部分国内外的参考文献、书籍，主要来源见参考文献。在此，对有关作者表示衷心感谢。

本书由张迎新担任主编，王盛军编写了第 5 章和各章节中的 C 语言程序，邢春香编写了 10.2、10.4、11.5 节，姚静波编写了 2.3 和 2.4 节，陈胜编写了 11.4 节，迟明华编写了 1.2 节，其余由张迎新编写。参加本教材编写及审稿的还有雷道振、樊桂花、刘绍南、雷文。

在本书的编写中，清华大学的陆延丰老师、浙江大学城市学院的万光毅老师等都提出了很好的建议，并提供了部分素材，在此表示衷心感谢。

由于作者水平有限，书中的错误与不妥之处在所难免，恳请广大读者批评指正。

编著者

目 录

第 1 章 概述	1	2.5.3 P1 口	42
1.1 单片机发展概况	1	2.5.4 P2 口	42
1.1.1 单片机的发展历史	1	2.5.5 P3 口	43
1.1.2 单片机的组成及特点	3	2.6 时序及时钟电路	44
1.1.3 单片机与嵌入式系统	4	2.6.1 CPU 时序及有关概念	44
1.2 80C51 系列单片机简介	5	2.6.2 振荡器和时钟电路	45
1.2.1 80C51 系列单片机的发展	5	2.6.3 80C51 的指令时序	47
1.2.2 AT89 系列单片机的特点 及分类	6	2.7 复位和复位电路	47
1.3 其他常用单片机系列简介	7	2.7.1 内部复位信号的产生	47
1.3.1 低端产品概述	7	2.7.2 复位状态	48
1.3.2 高端产品概述	8	2.7.3 外部复位电路设计	48
1.4 单片机基础知识	9	2.8 80C51 系列单片机的低功耗方式	49
1.4.1 数制与编码	9	2.8.1 电源控制寄存器 PCON	49
1.4.2 计算机的基本组成电路	12	2.8.2 待机方式	50
1.4.3 存储器简介	15	2.8.3 掉电方式	50
思考与练习	19	思考与练习	51
第 2 章 单片机结构及原理	20	第 3 章 指令系统	52
2.1 单片机结构	20	3.1 指令系统简介	52
2.1.1 单片机组成及结构	20	3.2 寻址方式	53
2.1.2 引脚定义及功能	23	3.2.1 符号约定	53
2.2 80C51 的存储器	25	3.2.2 寻址方式说明	54
2.2.1 存储器结构和地址空间	26	3.3 指令系统分类介绍	59
2.2.2 程序存储器	26	3.3.1 数据传送类指令	59
2.2.3 数据存储器	27	3.3.2 算术运算类指令	64
2.3 特殊功能寄存器 SFR	30	3.3.3 逻辑操作类指令	67
2.3.1 80C51 系列的 SFR	30	3.3.4 控制转移类指令	70
2.3.2 SFR 地址分布及寻址	31	3.3.5 位操作类指令	73
2.3.3 SFR 的功能与作用	32	思考与练习	75
2.4 单片机的工作原理	35	第 4 章 汇编语言程序设计	79
2.4.1 指令与程序概述	36	4.1 概述	79
2.4.2 CPU 的工作原理	36	4.1.1 程序设计语言	79
2.4.3 单片机执行程序过程	37	4.1.2 汇编语言规范	80
2.5 输入/输出端口结构	39	4.1.3 汇编语言程序设计步骤	83
2.5.1 4 个 I/O 端口的异同点	40	4.2 汇编语言程序设计举例	83
2.5.2 P0 口	40	4.2.1 顺序程序设计	84
		4.2.2 循环程序设计	84

4.2.3 分支程序设计	86	6.3.1 方式 0	134
4.2.4 查表程序设计	89	6.3.2 方式 1	134
4.2.5 子程序设计	91	6.3.3 方式 2	135
思考与练习	95	6.3.4 方式 3	135
第 5 章 C51 语言程序及开发环境	97	6.4 定时器 T0、T1 应用举例	137
5.1 C51 语言基础知识	97	6.4.1 定时应用举例	137
5.1.1 C51 语言简介	97	6.4.2 计数应用举例	138
5.1.2 C51 语言的运算符及 表达式	97	6.4.3 门控位应用举例	139
5.1.3 C51 语言的程序结构	99	6.5 定时/计数器 T2	141
5.1.4 C51 语言的流程控制语句	99	6.5.1 T2 的寄存器	142
5.2 C51 语言对通用 C 语言的扩展	102	6.5.2 定时器 T2 的工作方式	143
5.2.1 数据类型	102	6.5.3 应用例题	147
5.2.2 数据的存储类型	104	6.6 定时器 T3——WDT 监视定时器	149
5.2.3 指针	106	6.6.1 WDT 的功能及应用特点	149
5.2.4 函数	107	6.6.2 辅助寄存器 AUXR	149
5.2.5 C51 语言对单片机硬件的 访问	109	思考与练习	150
5.3 C51 语言编程举例	110	第 7 章 80C51 的串行接口	151
5.4 Keil C51 软件开发环境	114	7.1 串行通信概述	151
5.4.1 Keil 软件简介	114	7.1.1 同步通信和异步通信方式	151
5.4.2 项目的建立与设置	116	7.1.2 串行通信的数据传送速率	152
5.4.3 运行调试	118	7.1.3 串行通信的方式	152
5.5 Proteus 硬件仿真环境	121	7.1.4 通信协议	153
5.5.1 Proteus 软件简介	121	7.2 80C51 串行口简介	153
5.5.2 Proteus ISIS 窗口功能	121	7.2.1 串行口结构与工作原理	153
5.5.3 Proteus ISIS 的基本操作	123	7.2.2 串行口控制寄存器 SCON	154
思考题与练习	129	7.2.3 80C51 的帧格式	155
第 6 章 定时/计数器	130	7.2.4 波特率的设置	156
6.1 定时/计数器 T0、T1 概述	130	7.3 串行通信工作方式	157
6.1.1 定时器/计数器 T0、T1 的 结构	130	7.3.1 方式 0	157
6.1.2 定时/计数器的原理	131	7.3.2 方式 1	158
6.2 定时/计数器的控制方法	131	7.3.3 方式 2 和方式 3	158
6.2.1 定时/计数器寄存器	131	7.3.4 多机通信	159
6.2.2 定时/计数器的初始化	133	7.4 串行口应用举例	160
6.2.3 定时/计数器初值的 确定方法	133	7.4.1 用串行口扩展 I/O 口	160
6.3 定时/计数器 T0、T1 的 工作方式	134	7.4.2 用串行口进行异步通信	163
		思考与练习	169
		第 8 章 中断系统	170
		8.1 概述	170
		8.1.1 中断的概念	170
		8.1.2 引进中断技术的优点	170

8.1.3	中断源	171	9.5.3	I ² C 总线的通用模拟 软件包	208
8.1.4	中断系统的功能	171	9.5.4	I ² C 总线应用举例	213
8.2	AT89S51 单片机的中断系统	173	9.6	扩展数/模转换器	217
8.2.1	中断系统的结构	173	9.6.1	数/模转换器简介	218
8.2.2	中断源及中断入口	173	9.6.2	数/模转换电路原理	218
8.2.3	与中断控制有关的寄存器	175	9.6.3	D/A 转换器的 主要技术指标	219
8.3	中断处理过程	178	9.6.4	并行 D/A 转换器	220
8.3.1	中断响应	178	9.7	扩展模/数转换器	223
8.3.2	中断处理	179	9.7.1	模/数转换器简介	223
8.3.3	中断返回	180	9.7.2	模/数转换器的 主要技术指标	224
8.3.4	中断请求的撤除	180	9.7.3	逐次逼近式 A/D 转换器	224
8.3.5	中断响应时间	181	9.7.4	双积分 A/D 转换器	228
8.3.6	扩充外中断源	181	9.7.5	串行 A/D 转换器	232
8.4	中断程序的设计与应用	182	思考与练习		234
8.4.1	中断程序的一般设计方法	183	第 10 章	接口技术	236
8.4.2	中断程序应用举例	185	10.1	键盘接口	236
思考与练习		192	10.1.1	键盘工作原理	236
第 9 章	单片机的系统扩展	193	10.1.2	独立式按键	237
9.1	并行扩展概述	193	10.1.3	行列式键盘	239
9.1.1	系统扩展常用接口芯片	193	10.2	显示器接口	246
9.1.2	外部并行扩展总线	195	10.2.1	LED 显示器的 结构与原理	246
9.1.3	并行扩展的寻址方法	196	10.2.2	LED 静态显示方式	247
9.2	存储器的并行扩展	197	10.2.3	LED 动态显示方式	248
9.2.1	数据存储器扩展概述	197	10.2.4	液晶显示器概述	251
9.2.2	访问片外数据存储器的 操作时序	197	10.2.5	字符型液晶显示模块 LCM 的组成及原理	252
9.2.3	数据存储器扩展举例	198	10.2.6	字符型液晶显示模块 LCM 的引脚及说明	253
9.3	并行 I/O 接口的扩展	199	10.2.7	LCM 的指令	254
9.3.1	扩展并行 I/O 口简述	199	10.2.8	LCM 的复位及初始化	256
9.3.2	简单并行 I/O 口的扩展	200	10.2.9	LCM 的接口及应用举例	257
9.4	串行扩展概述	201	10.3	功率驱动器件及接口电路	263
9.4.1	常用串行总线与 串行接口简介	201	10.3.1	输出接口的隔离技术	264
9.4.2	单片机串行扩展的 模拟技术	205	10.3.2	直流负载驱动电路	264
9.4.3	串行扩展的主要特点	205	10.3.3	晶闸管负载驱动电路	265
9.5	I ² C 总线	206	10.3.4	继电器接口电路	266
9.5.1	I ² C 总线的组成及 基本工作原理	206			
9.5.2	I ² C 总线的传输时序	206			

10.3.5 固态继电器接口电路	267	11.4 PC 机与单片机间的	
10.4 打印机接口	268	串行通信设计	286
10.4.1 微型打印机简介	268	11.4.1 RS-232C 标准接口	286
10.4.2 字符代码及打印命令	269	11.4.2 单片机与 PC 机之间的	
10.4.3 打印机与单片机接口		电平转换芯片	287
举例	270	11.4.3 PC 机与单片机串行通信	
思考与练习	272	应用实例	288
第 11 章 单片机应用系统的设计		11.5 步进电机控制设计	293
与开发	273	11.5.1 步进电机的工作原理	294
11.1 应用系统研制过程	273	11.5.2 步进电机的控制方法	294
11.1.1 总体方案设计	273	11.5.3 步进电机控制应用举例	295
11.1.2 硬件设计	274	思考与练习	299
11.1.3 软件设计	276	附录 A 80C51 指令表	300
11.2 开发工具和开发方法	278	附录 B 各数制对照表	305
11.2.1 开发工具	278	附录 C ASCII	
11.2.2 开发方法	280	(美国标准信息交换码) 表	306
11.3 恒温箱温度控制监测系统	281	附录 D 二进制逻辑单元图形符号	
11.3.1 题目分析	282	对照表	307
11.3.2 硬件设计	282	附录 E 常用芯片引脚图	308
11.3.3 软件设计	283	参考文献	310

第 1 章 概 述

为适应嵌入式应用的需要，单片机应运而生，标志着微型计算机进入了通用计算机与嵌入式计算机两大分支并行发展的时代，推动了计算机产业革命的高速发展。从 1976 年开始至今 40 多年的时间里，单片机已发展成为一个品种齐全、功能丰富的庞大家族。

目前单片机已成为工控领域、尖端武器、日常生活中最广泛使用的控制器，因而对广大理工科高等院校的学生和科技人员来说，单片机原理及应用已成为一门重要的基础知识课程。

1.1 单片机发展概况

单片机一词最初是源于“Single Chip Microcomputer”，简称 SCM。在单片机诞生时，因为它的组成与原理都基于计算机，所以 SCM 是一个准确的、流行的称谓。随着 SCM 在技术上、体系结构上的不断进步，使其控制功能不断扩展，它的主要作用已不是计算，而是控制。国际上也逐渐采用“MCU”（Micro Controller Unit），即微控制器来代替 SCM，形成了单片机界公认的、最终统一的名词。为了与国际接轨，以后应将中文“单片机”一词和“MCU”作为唯一的对应翻译。在国内因为单片机一词已约定成俗，所以可继续沿用。

1.1.1 单片机的发展历史

如果将 8 位单片机的推出作为起点（1976 年），那么，单片机的发展历史大致可分为 4 个阶段。

1. 单片机的探索阶段（1974 年—1976 年）

主要是探索如何把计算机的主要部件集成在单芯片上。Intel 公司推出的 MCS-48 就是在工控领域探索的代表，参与这一探索的还有 Motorola, Zilog 等公司，也都取得了满意的效果。这是单片微型计算机的诞生年代，单片机一词即由此而来。

2. 单片机完善阶段（1976 年—1978 年）

Intel 公司在 MCS-48 基础上推出了完善的、典型的 MCS-51 单片机系列。它在以下几个方面奠定了典型的通用总线型单片机体系结构。

- ① 设置了经典、完善的 8 位单片机的并行总线结构。
- ② 外围功能单元由 CPU 集中管理的模式。
- ③ 体现控制特性的位地址空间、位操作方式。
- ④ 指令系统趋于丰富和完善，并且增加了许多突出控制功能的指令。

由于 MCS-51 系列单片机在结构上的逐渐完善，奠定了它在这一阶段的领先地位。它的产品曾经在世界单片机市场占有 50% 以上的份额，因而多年来国内一直以 MCS-51 系列单片

机作为教学的主要机型。在这一阶段，Motorola 公司的 M68 系列和 Zilog 公司的 Z8 系列也占据了一定的市场份额。

3. 向微控制器发展的阶段（1978 年—1983 年）

为满足测控系统要求的各种外围电路与接口电路，突出其智能化控制能力，Philips 等一些著名半导体厂商在 8051 基本结构的基础上，加强了外围电路的功能，突出了单片机的控制功能，将一些用于测控对象的模/数转换器、数/模转换器、程序运行监视器、脉宽调制器等纳入芯片中，体现了单片机的微控制器特征。

为了进一步缩小单片机体积，出现了为满足串行外围扩展要求的串行总线及接口，如 I²C、SPI、MICROWIRE 等串行总线及接口。同时带有这些接口的各种外围芯片也应运而生，例如存储器、A/D、时钟等，出现了有较高性能的 16 位单片机。

单片机的首创公司 Intel 将其 MCS-51 系列中的 8051 内核使用权以专利互换或出售形式转让给世界许多著名 IC 制造厂商，如 Philips、Atmel、NEC、SST、华邦等。这些公司的产品都在保持与 8051 单片机兼容的基础上增强了 8051 的许多特性，在工艺上都采用了 CHMOS 和闪存技术。为了与 Intel 早期的 MCS-51 系列产品区别，后来统称为 80C51 系列，也有人简称为 51 系列。这样 80C51 系列得到众多制造厂商的支持，继而发展成上百个品种的大家族。从此作为单片机领军代表的 Intel 公司退出了 8 位单片机市场，但它的历史功绩是不会被抹杀的。在本书中提到的 80C51 已经不是 MCS-51 系列中的 80C51 型号单片机，而是 80C51 系列的一个统称。专家认为虽然世界上的 MCU 品种繁多，功能各异，开发装置也互不兼容，但是客观发展表明：尽管 80C51 系列单片机现在并不是最完善和最先进的单片机，但从综合因素（如教学的连续性和更换教学设备的资金等问题）考虑，它仍然适合作为单片机教学的首选机型。

4. 单片机的全面发展阶段（1983 年—今）

由于很多大半导体和电气厂商都开始加入单片机的研制和生产，单片机世界出现了百花齐放，欣欣向荣的景象。随着单片机在各个领域全面深入地发展和应用，出现了高速、大寻址范围、强运算能力的 16 位、32 位通用型单片机以及小型廉价的专用型单片机，还有功能全面的片上单片机系统。其中 8 位单片机是目前品种最多，应用最广泛的单片机，众多半导体厂商在竞争中发展，在发展中互相取长补短，使单片机的发展与完善速度始终处于其他各类产品的前列。

目前单片机正朝着高性能和多品种方向发展，嵌入式应用对产品的主要要求是更高的集成度、更低的功耗和更丰富的外设。所以今后单片机的发展趋势将是进一步向着低功耗、小体积、大容量、高性能、高可靠性、串行扩展技术、低价格和混合信号集成化（即数字—模拟相混合的集成技术）等几个方面发展。此外，单片机开始由复杂指令系统计算机（CISC，Complex Instruction Set Computer）向精简指令系统计算机（RISC，Reduced Instruction Set Computer）发展，CISC 功能较全，但指令条数较多，RISC 指令条数大为精简，且多数情况均为单周期指令，因而它的指令执行速度可大幅度提高。

近年来，随着信息技术的飞速发展，对嵌入式系统提出了更高的要求，随后产生了许多新型设备，如手持电脑、可上网的无线移动手机、机顶盒、可上网的电视机、智能家用电器等等。相应地对嵌入式软件也提出了更高的要求，促使软件也随着硬件同步发展。

1.1.2 单片机的组成及特点

单片机是微型机的一个主要分支，它在结构上的最大特点是把 CPU、存储器、定时器和多种输入/输出接口电路集成在一块超大规模集成电路芯片上。就其组成和基本工作原理而言，一块单片机芯片就是一台计算机。

1. 单片机的组成

图 1-1 为单片机的结构框图。由图可见，单片机的核心部分是中央处理器 CPU，它是单片机的大脑，由它统一指挥和协调各部分的工作。时钟电路用于给单片机提供工作时所需要的时钟信号。程序存储器和数据存储器分别用于存放单片机工作的用户软件和临时数据。中断系统用于处理系统工作时出现的突发事件。定时/计数器用于对时间定时或对外部事件计数。它通过内部总线把计算机的各主要部件连接为一体，其内部总线包括地址总线、数据总线和控制总线。其中，地址总线的作用是为数据交换时提供地址，CPU 通过它们将地址输出到存储器或 I/O 接口；数据总线用于 CPU 与存储器或 I/O 接口之间，或 I/O 接口与外设之间交换数据；控制总线包括 CPU 发出的控制信号线和外部送入 CPU 的应答信号线等。输入/输出接口（I/O 接口）是计算机与输入/输出设备之间的接口。输入/输出设备（I/O 设备）是计算机与设备交换信息的装置，如显示器、键盘和打印机等。

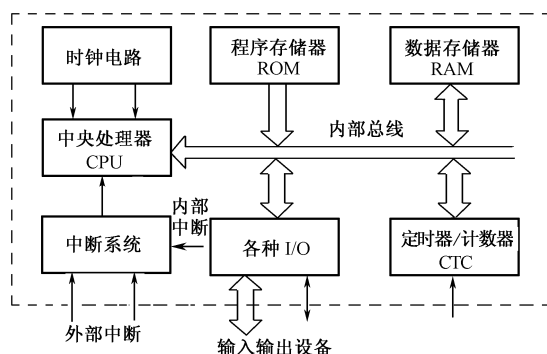


图 1-1 单片机结构框图

2. 单片机的特点

正是由于单片机的这种结构形式及它所采取的半导体工艺，使其具有很多显著的优点和特点，因而能在各个领域得到广泛的应用。

单片机主要特点如下：

（1）控制功能强。其指令丰富，因而控制灵活、方便，容易满足一般控制的要求。对于所有的被控对象，均可实现一旦启动将自动循环操作，不需要人工干预。

（2）集成度高，可靠性强。由于单片机把各功能部件集成在一块芯片上，减少了芯片内部之间的连线，大大提高了单片机的可靠性与抗干扰能力。对于强磁场环境易于采取屏蔽措施，适合于在恶劣环境下工作。

（3）有优异的性能价格比。

- (4) 低功耗、体积小、低电压，便于生产便携式产品。
- (5) 单片机的系统扩展、系统配置较典型、规范，容易构成各种规模的应用系统。

1.1.3 单片机与嵌入式系统

正是由于单片机具有上述显著的优点，它已成为科技领域的有力工具，人类生活的得力助手。它的应用遍及各个领域，应用形式主要为嵌入式。

1. 嵌入式系统的定义与分类

所谓嵌入式系统 (Embedded System)，实际上是“嵌入式计算机系统”的简称，它是相对于通用计算机而言的。

简言之，嵌入式系统就是一个嵌入到对象 (目标) 系统中的一个专用计算机系统，这个计算机就成为系统的一部分。它是面向产品、面向实际应用的系统，主要用于对目标系统各种信号的处理和控制，应用范围遍及各个领域，通常要求它具有很高的可靠性和稳定性。

2. 嵌入式系统的分类

嵌入式计算机是嵌入式系统的核心，它是一种软、硬件高度专业化的特定计算机，它的核心部件是嵌入式处理器，根据目前发展现状，嵌入式处理器可以分成下面几类：

(1) 嵌入式微处理器 (Embedded Microprocessor Unit, EMPU)

微处理器实际是计算机或单片机的 CPU，即他们的中央处理器。

目前采用的嵌入式微处理器主要是 32 位的，常用的型号有 ARM、MIPS、AM186/88、68000 等，其中广为流行的微处理器是 32 位的 ARM，ARM 是 Advanced RISC (Reduced Instruction Set Computer) Machines 的缩写，是设计 ARM 处理器技术的公司 (英国) 简称，同时可以认为它是一种技术的名称，它几乎变成 32 位微处理器的代名词，目前全世界较大的半导体厂家都在利用 ARM 技术。

(2) 微控制器 (Micro Controller Unit, MCU)

微控制器即单片机。为适应不同的应用需求，一般一个系列的单片机具有多种衍生产品，每种衍生产品的处理器内核都是一样的，不同的是存储器和外设的配置及封装。这样可以使单片机最大限度地和应用需求相匹配，从而减少功耗和成本，提高可靠性。

(3) 嵌入式 DSP 处理器 (Embedded Digital Signal Processor, EDSP)

为满足数字滤波、FFT、谱分析等运算量大的智能系统的要求，DSP 算法已经大量进入嵌入式领域，为适合执行 DSP 算法，DSP 处理器对系统结构和指令进行了特殊设计，使其编译效率较高，指令执行速度也较快，能满足高速算法的要求。实际上现在已经出现了很多 DSP 单片机，它是把单片机中的 CPU 改为 DSP 内核，其他基本不变。

DSP 处理器的典型产品是 TI 公司的 TMS320 系列、Motorola 公司的 DSP56800 系列等。

(4) 嵌入式片上系统 (System on Chip, SoC)

随着电子技术、半导体技术的迅速发展，已经实现了把嵌入式系统的大部分功能集成到一块芯片上去，这就是片上系统 SoC，在这上面除了具有计算机的主要部件之外，还增加了 A/D、D/A 及通信单元等用户需要的各种功能模块。这使应用系统电路板变得更简洁，体积更小，功耗更低，可靠性更高。

在上述 4 种嵌入式系统中，单片机应用最广泛，因为它有专门为嵌入式应用设计的体系结构和指令系统，此外，它还具有体积小、价格低、易于掌握和普及的特点。

3. 单片机与嵌入式系统应用

单片机以单片器件的形式进入到了电子技术领域，主要用于电子系统的智能化。嵌入式系统起源于微型计算机时代，然而微型计算机的体积、价位、可靠性都无法满足广大对象系统的嵌入式应用要求，因此，嵌入式系统的单芯片化应运而生，从而进入了嵌入式系统独立发展的单片机时代。从此，以单片机为主的嵌入式系统迅速地将传统的电子系统发展到智能化的现代电子系统时代。嵌入式系统目前在应用数量上远远超过了一般的通用计算机。

现在，从需要高、精、尖技术的火箭、飞船到日常生活中常见的手机、汽车电子、智能玩具、日用家电、医疗器械等，都已经嵌入了单片机。单片机已经成为人类社会进入全面智能化时代不可或缺的工具。

即将来临的物联网时代是继计算机、互联网和移动通信之后的又一次信息产业的革命性发展。物联网的英文名称是“The Internet of things”。由此，可以把物联网理解为“物物相连的互联网”。物联网是把任何物体通过信息传感设备，按照约定的协议，通过各种接入技术与互联网连接起来，进行信息交换和通信，以实现对物体的识别、跟踪、监控与管理的网络。无论何时何地，世界上任何物体都可以通过物联网实现连接。这将使我们的工作和生活更加方便和快捷，可以推动各行业的快速发展，因而物联网被称为继计算机、互联网之后，世界信息产业的第三次浪潮。在嵌入式系统具有互联网接入功能时，将互联网变革到物联网。

物联网需要感测、控制、计算和通信相结合。物联网给 MCU 应用创造了更多机会，物联网时代是高低端交叉融合的时代，物联网的前端需要依赖软件与硬件结合的嵌入式系统技术，所以物联网的发展将进一步扩大单片机的应用范围。

1.2 80C51 系列单片机简介

单片机作为嵌入式系统的一员，应用面很广，发展很快。自单片机诞生至今的 40 多年中，加入单片机生产和研制的厂家在世界已经有上百家，它已发展为几百个系列的上千个机种，使用户有较大的选择余地。随着集成电路的发展，单片机从 4 位发展到 8 位、16 位、32 位，根据近年来的使用情况看，8 位单片机仍然是低端应用的主要机型，专家预测，在未来的相当长时间中，仍将保持这个局面。所以，目前教学的首选机型还是 8 位单片机，而 8 位单片机中最有代表性和最经典的机型是 80C51 系列单片机。

1.2.1 80C51 系列单片机的发展

80C51 系列单片机是在 Intel 公司的 MCS—51 系列单片机基础上发展起来的。MCS—51 和 80C51 系列单片机现在常简称为 51 系列单片机。90 年代中期随着 Intel 公司对 8051 内核彻底的技术开放，使得众多的半导体厂商（例如 Philips、Atmel、SST、Winbond 等）参与了 MCS-51 单片机的技术开发。不同厂家在发展 80C51 系列时都保证了产品的兼容性，主要是指指令兼容、总线兼容和引脚兼容。与此同时，这些公司又融入了自身的优势，扩展了针对满足不同测控对象要求的外围电路，从而众多厂家的参与使 80C51 的发展长盛不衰，形成了一

个既具有经典性，又有旺盛生命力的单片机系列。

纵观 80C51 系列单片机的发展史，可以看出它曾经历过 3 次技术飞越。

1) 从 MCS-51 到 MCU 的第一次飞越

在 Intel 公司实行技术开放后，著名半导体厂商 Philips 利用它在电子应用方面的优势，在 8051 基本结构的基础上，着重发展 80C51 的控制功能及外围电路的功能，突出了单片机的微控制器特征，这使得单片机出现了第一次飞越。

2) 引入快擦写存储器的第二次飞越

1998 年以后 80C51 系列单片机又出现了一个新的分支，称为 AT89 系列单片机。这种单片机是由美国 Atmel 公司率先推出的，它的最突出优点是把 Flash 存储器（详见 2.2）应用于单片机中。这使得单片机系统的开发周期大大缩短，因此它很快从单片机市场脱颖而出。这使得单片机的发展出现了第二次飞越。

3) 向 SoC（System on Chip）转化的第三次飞越

美国 Silabs 公司推出的 C8051F 系列单片机把 80C51 系列单片机从 MCU（微控制器）推向 SoC 时代，它的主要特点是改进了 8051 内核，使得其指令运行速度比一般的 80C51 系列单片机提高了大约 10 倍，在片上增加了多种功能模块如模数和数模转换模块等。这是 80C51 单片机的第三次飞越。

1.2.2 AT89 系列单片机的特点及分类

AT89 系列单片机的成功使得几个著名的半导体厂家也相继生产了类似的产品，例如，Philips 的 P89 系列、美国 STC 公司的 STC89 系列、华邦公司的 W78 系列等。后来人们就简称这一类产品为 89 系列单片机，它实际上还是属于 80C51 系列。这些产品主要功能类似，但又各具特色。在这些型号中 AT89S51、P89C51、STC89C51、W78E51 都是与 MCS-51 系列的 80C51 兼容的型号。这些芯片互相之间也是兼容的，所以如果不写前缀，仅写 89C51 可能是其中任何一个厂家的产品。

89 系列单片机的主要特点如下：

- 内部含 Flash 存储器；
- 89 系列单片机的内部结构和 80C51 相近；
- 工作原理和指令系统完全相同。

89 系列单片机可分成标准型号、低档型号和高档型号 3 类。标准型单片机的主要结构与性能详见第 2 章。低档 AT89 单片机是在标准型结构的基础上，适当减少某些功能部件，例如减少 I/O 引脚数，减少存储器和 RAM 容量等，这样可使其体积更小，价格更便宜。在 89 系列单片机中，高档（即增强）型产品是在标准型的基础上增加了一些功能形成的，所增加的功能部件主要有串行外围接口 SPI、CAN 和 A/D 功能模块等。

89 系列单片机是 80C51 系列单片机的典型代表，89 系列单片机目前在世界上应用很广泛，可以满足大多数用户的需要。由于 80C51 系列中的典型型号在基本结构、工作原理和引脚上与 MCS-51 系列单片机的 8051 是完全兼容的，所以 89 系列单片机虽然并不是功能最强，最先进的单片机，但它是源于经典的 MCS-51 系列，考虑到教学的连续性及 89 系列单片机和所用开发装置的普及性，因而 89 系列单片机成为单片机教学的首选机型。

本书在介绍具体单片机结构时选用 AT89S51/52 单片机（因为 AT89C51/52 在 2003 年已

经停产，AT89S51/52 是其替代产品，不过 Philips 等公司的 89C51/52 仍然有产品)，但在作一般共性介绍时还是用符号 80C51 代表，注意此时它指的是 80C51 系列芯片，而不是 Intel 以前生产的 80C51 型号芯片。掌握了这种单片机对于其他型号单片机的学习可以起到举一反三、触类旁通的作用。

1.3 其他常用单片机系列简介

在准备用单片机进行应用开发时，首先应了解单片机市场的常用单片机系列概况。目前加入单片机生产和研制的厂家在世界已经有上百家，他们的产品都各具有一定特色。

1.3.1 低端产品概述

低端产品主要是指 8 位及少数 16 位单片机，它们可以满足各领域一般的智能化与控制要求。在大多数应用场合采用 8 位单片机就可以圆满解决问题，所以 8 位单片机还是目前产品最多、用量最大的单片机，由于篇幅关系，在此仅简介知名度较高，销量较大的几种产品。

1. Freescale 单片机

Freescale（音译为飞思卡尔）是从原 Motorola 半导体部分分离出来的，是世界上最大的单片机厂商，在 8 位单片机方面主要有 68HC08、68HC05 等 30 多个系列的 200 多个品种。

其 8 位单片机的主要特点如下：

- 品种全，选择余地大，它能生产 8 位、16 位和 32 位各种档次单片机，除了有通用单片机之外，还有具有电动机控制的、具有 CAN 接口的、具有 USB 接口的、具有彩色液晶监视器控制和无线通讯功能等的单片机。
- 抗干扰能力强，适于恶劣的工作环境：在同样的指令速度下所用的时钟频率较低。

2. Philips 单片机简介

Philips 公司是较早生产 51 系列单片机的厂商，先后推出了基于 8051 内核的普通型 8 位单片机、增强型单片机、LPC700 系列、LPC900 系列等多种类型。

Philips 单片机主要特点如下：

- EMI 电磁兼容性能好：可以在上电初始化时“静态关闭 ALE”，还可以在运行中“动态关闭 ALE”，以改善电磁兼容性能。
- 有 6/12Clock 时钟频率切换功能：可以在运行中“动态切换 6/12Clock”。
- 速度快：在同一时钟频率下，其速度为标准 80C51 器件的 6 倍。

3. PIC 系列单片机简介

PIC 系列单片机是美国 Microchip 公司推出的高性能 8 位系列单片机。

PIC 系列单片机主要特点如下：

- 开发易、周期短：PIC 采用精简指令集，指令执行速度比一般单片机要快 4~5 倍。
- 低功耗：PIC 的 CMOS 设计结合了诸多的节电特性，使其功耗较低。
- 低价实用：PIC 配备有多种形式的芯片，特别是其 OTP 型芯片的价格很低。

4. TI 公司的 MSP430 系列

MSP430 系列单片机是 TI 公司（美国德州仪器公司）生产的，它的最主要特点是超低功耗，MSP430 是属于 16 位单片机。

MSP430 系列单片机的主要特点如下：

- 低电压、超低功耗：MSP430 系列单片机一般在 1.8~3.6V 电压、1MHz 的时钟条件下运行，耗电电流（在 0.1~400 μ A 之间）因不同的工作模式而不同。
- 丰富的片内外设：单片机的片上外设除了具有定时器、看门狗等常见功能模块之外还具有液晶驱动器、10/12/14 位 ADC 等。

5. 深圳宏晶科技有限公司的 STC15 系列

深圳宏晶科技有限公司生产的 STC15 系列单片机是我国生产的 8 位单片机，其内核也是 8051，目前在国内市场占有率较高的市场份额。

STC15 系列单片机的主要特点如下：

- 其在抗干扰、运行速度等方面都有创新，且还进行了特别加密设计，集成了更多的功能模块，例如 A/D、PWM 以及更多的定时器。还集成了时钟振荡器，内部上电复位电路，可省去外部晶振电路和复位电路。
- 具有 2KB 的大容量 SRAM，64KB 的闪存，还有 1~2KB 的 EEPROM。

除上述厂家之外较著名的还有 NEC、东芝、富士通等公司，由于篇幅关系在此不可能全面介绍上述厂家产品，也不能一一列举所有厂家的产品，

1.3.2 高端产品概述

嵌入式系统的高端产品是在低端产品的基础上发展起来的，是为了满足复杂图像处理、手机、网络、机器人及通信等方面的需求而产生的，由于 32 位高端产品的高性能，使嵌入式系统的应用面更广、更深，把嵌入式系统提高到一个新的水平，随后各大单片机厂商都推出了自己的 32 位单片机，对嵌入式系统市场产生了巨大的冲击力，使嵌入式系统从普遍的低端应用进入到高、低端并行发展阶段。

目前广为流行的 32 位单片机的内核主要是英国 ARM 公司开发的微处理器，简称 ARM，它几乎变成 32 位微处理器的代名词，目前全世界较大的半导体厂家都在利用 ARM 技术。基于 ARM 技术的处理器约占据了 32 位微处理器 80% 以上的市场，被授权厂商有 Intel、NEC、Motorola、IBM、Philips 等 100 多家著名芯片厂商。ARM 公司开发了很多系列的 ARM 处理器核，应用较多的主要有 ARM7、ARM9、ARM10、ARM11 等系列，还有 Intel 的 XScale 系列和 MPCore 系列等。

32 位单片机虽然在基本概念与工作原理上与 8 位单片机有相同之处，但其功能与性能上要强大得多。其主要特点如下：

- ARM 微处理器采用 RISC（精简指令集）体系结构，支持 16/32 位双指令集。
- 采用多级流水线预取指令，这样可使几个操作（取指、译码和执行）同时进行，所以执行速度比 8 位机高多倍。
- 其体积小，功耗低、功能强大，具有很高的性价比。
- 寻址方式灵活，执行效率高，还可以很好地兼容 8 位/16 位机。

- 具有 30 个以上的 32 位通用寄存器，且均可作为累加器。
- 寻址范围可达 64MB，片内 RAM 可达 8KB。

上述特点使 32 位机适于进行高速海量的数据处理。可以满足网络、通信及多媒体的高端应用。

由于人工智能的感知与控制终端领域的多样性，需要功能特点不同的 MCU，所以在嵌入式系统市场中，不论是 8 位、32 位或者 64 位 MCU，都有各自的用武之地，总之单片机世界不是一枝独秀，而是百花齐放，他们既有共性，也有个性。不同的单片机适合不同的应用场合，当然，同一应用场合，也可以选择不同的单片机实现。在准备用单片机进行应用开发时，首先应了解单片机市场的常用单片机系列概况。对于用户来说最重要的是学会针对不同需要，选择最适合的单片机。选择的依据是在充分考虑应用对象的特点、要求及应用环境等情况下，如何使产品达到性能、功能以及成本的最佳平衡。

1.4 单片机基础知识

单片机的基本原理基于计算机，计算机是计算数学与微电子学相结合的产物。微电子学的基本元件及其集成电路是计算机的硬件基础，而计算数学的计算方法与数据结构则形成计算机的软件基础。

本节简要地介绍计算机中最主要的数学知识及最基本的单元电路。本节的内容是必要的入门知识，是以后各章的基础。

1.4.1 数制与编码

1. 数制

数制是人们利用符号来计数的科学方法。数制有很多种，按一定进位方式计数的数制，简称进位制。在计算机的设计与使用上常用到的进位制是十进制、二进制和十六进制。

(1) 数制的基与权

数制所使用的数码的个数称为基，基数是某种进位制中产生进位的数值，它等于每个数位中所允许的最大数码值加 1，即各数位允许的数码个数。数制每一位所具有的值称为权。在进位制中每个数位都有自己的权值，显然各位的权是不同的。

① 十进制 十进制是在一般的数学计算中最常用的数制，十进制的基为“十”，即它所使用的数码为 0~9 共 10 个数字。十进制各位的权是以 10 为底的幂，每个数所处的位置不同，它的值是不同的。每一位数是其右边相邻那个位数的 10 倍，例如，数 368 按权的展开式为：

$$368D = 3 \times 10^2 + 6 \times 10^1 + 8 \times 10^0$$

上式中的后缀 D 表示十进制数 (Decimal)，通常对十进制可不加后缀。

由上式可见，在十进制中，每个（位）数字的值都是以该个（位）数字乘以基数的幂次来表示，通常将基数的幂次称为权。

② 二进制 十进制所用数字较多，如果用电路实现其计算则电路会很复杂，而二进制数只有两个数码，即 0 和 1，在电子计算机中容易实现。例如用高电平表示 1，低电平表示 0。采用二进制，就可以方便地利用电路进行计数工作。所以，计算机中常用的进位制是二

进制。

二进制的基为“二”，即其使用的数码为 0、1，共两个数字。二进制各位的权是以 2 为底的幂，例如二进制数 1101 按权的展开式为：

$$1101\text{B} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13$$

上式中的后缀 B (Binary) 表示二进制数。

③ 十六进制 由于二进制位数太长，不易记忆，不易书写，所以人们又提出了十六进制的书写形式。十六进制的基为“十六”，即其数码共有 16 个：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。其中 A~F 相当于十进制数的 10~15。十六进制的权是以 16 为底的幂，有时也称其各位的权为 0 权、1 权、2 权等。例如数 A3EH 按权的展开式为：

$$\text{A3EH} = 10 \times 16^2 + 3 \times 16^1 + 14 \times 16^0 = 2560 + 48 + 14 = 2622$$

上式中的后缀 H 表示为十六进制数 (Hexadecimal)。十六进制数如果是字母打头，则在使用汇编指令时前面需加一个 0。

由于十六进制数易于书写和记忆，而且与二进制之间的转换十分方便，因而人们在书写计算机的语言时多用十六进制。

(2) 数制的转换

① 二、十六进制数转换成十进制数 根据定义，只需将二、十六进制数按权展开后相加。

例：
$$1101\text{B} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$$

$$\text{B5H} = 11 \times 16^1 + 5 \times 16^0 = 181$$

② 十进制数转换成二、十六进制数 一个十进制整数转换成二进制数时，通常采用“除二取余”法，即用 2 连续除十进制数，直至商为 0，逆序排列余数即可得到。例如，将 11 转换成二进制数：

	余数	低位
2 11 1	↑ 高位
2 5 1	
2 2 0	
2 1 1	
0		

结果：11=1011B

同理，将十进制数“除十六取余”即可得到十六进制数。

2. 计算机中数的表示及运算

计算机中的数均是以二进制表示的，通常称为机器数，其数值为真值。真值可以分别用有符号数和无符号数表示，下面分别介绍其表示方法及运算。

(1) 有符号数的表示方法

数学上有符号数的正负号分别用“+”和“-”来表示。在计算机中由于采用二进制，只有“1”和“0”两个数字，一般规定最高位是符号位，最高位 D7 为“0”表示正数，为“1”表示负数。因为符号位占据了最高位 D7，所以 8 位二进制数实际可表达的数据位为 D0~D6。

计算机中的有符号数有三种表示法，即原码、反码和补码。因在 8 位单片机中多数情况以 8 位二进制数为单位表示数字，下面所举例子均为 8 位二进制数。用 2 个数值相同，但符号相反的二进制数 X1、X2 举例说明。

① 原码

正数的符号位用 0 表示，负数的符号位用 1 表示。这种表示法称为原码。

例如：X1=+1010111 [X1]_原=01010111

X2=-1010111 [X2]_原=11010111

左边数称为真值，即为某数的实际有效值。右边为用原码表示的数，两者的最高位分别用“0”、“1”代替了“+”、“-”。

② 反码

反码是在原码的基础上求得的。如果是正数，则其反码和原码相同；如果是负数，则其反码除符号为 1 外，其他各数位凡是 1 转换为 0，0 转换为 1。

例如：X1=+1010111 [X1]_反=01010111

X2=-1010111 [X2]_反=10101000

③ 补码

补码是在反码的基础上求得的。如果是正数，则其补码和反码相同，亦即与原码相同；如果是负数，则其补码为反码加 1。

例如：X1=+1010111 [X1]_补=01010111

X2=-1010111 [X2]_补=10101001

(2) 有符号数的运算

显然，原码简单、直观容易理解。但在计算机中采用原码进行加、减运算时，所需要的电路将比较复杂；如果采用补码，就可以把减法变成加法运算，省去了减法器，大大简化了硬件电路。

因为在 8 位机中，最高位 D7 的进位已超出计算机字长的范围，所以是自然丢失的。由此可见，在不考虑最高位产生进位的情况下，作减法运算与补码相加的结果完全相同。

当数用补码表示时，无论是加法还是减法运算都可采用加法运算，而且是连同符号位一起进行的，不必关心符号位，并能得到正确结果，因此在计算机中普遍用补码来表示带符号的数。

(3) 无符号数的表示方法

无符号数因为不需要专门的符号位，所以 8 位二进制数的 D7~D0 均为数值位，它的表示范围为 0~+255。

在计算机中的同一个二进制数，当采用不同表达方式时，它所表达的实际数值是不同的，例如在此特别典型的数值即 128。当用原码、反码和补码表达时，其对应的十进制数分别是 -0、-127 和 -128。要确切地知道计算机中的二进制数所对应的十进制数究竟是多少，首先要确定这个数是有符号数（一定要用补码表示）还是无符号数。注意：在单片机中的有符号数通常是用补码表示的。

3. 二进制编码

由于计算机中采用二进制数，所以，要在计算机中表示数字、字母、字符、汉字等都要用特定的二进制码表示。在计算机中把二进制代码按一定规律编排，使每组代码具有一个特定含义，即为计算机中的编码。

(1) 二—十进制编码

因为人们最熟悉和习惯使用的数码还是十进制数，而在计算机中常采用一种二进制编码表示的十进制数，即二—十进制数。

① 二一十进制的表示

二一十进制数称为二进制编码的十进制数 (Binary Coded Decimal)，简称 BCD 码。在 BCD 码中是用 4 位二进制数给 0~9 这 10 个数字编码的。例如，十进制数 78 用压缩 BCD 码表示为 01111000。

使用这种数制既考虑了计算机的特点，又顾及到人们使用十进制数的习惯，在计算机中输入和输出数据时常采用这种数制。

② BCD 码与十进制数的相互转换

按照 BCD 的 10 位编码与十进制的关系，可以很容易地实现 BCD 码与十进制数之间的转换。

例： 0101 1000 0110 BCD = 586

BCD 码与二进制数之间的转换不是直接的，要先经过十进制转换，然后再转换为二进制数，反之过程类似。

上述四种数制的对照表见附录 B，由此可见各数制的异同。

4. 字母与字符的编码

计算机除了要处理数字量之外，还要处理字母、字符等。因此计算机中的字母、字符等也必须采用特定的二进制码表示。

字母与字符用二进制码表示的方法很多。目前在计算机中最普遍采用的是美国标准信息交换码，简称为 ASCII 码 (American Standard Code for Information Interchange)。编码表见附录 C。它是 7 位 (b0~b6) 二进制编码，故可以表示 128 个字符，其中包括数字 (0~9) 以及英文字母等可打印的字符。

1.4.2 计算机的基本组成电路

无论多么复杂的计算机，都是由若干基本电路单元组成的。本节针对计算机中最常见的基本电路做一简单介绍，这些电路是组成计算机的硬件基础。

1. 常用逻辑电路

逻辑电路是计算机实现运算、控制功能所必需的电路，是计算机的基本单元电路。

逻辑电路中，其输入和输出只有两种状态，即高电平和低电平。通常以逻辑 1 和 0 表示电平高低。下面简单介绍常用逻辑电路的逻辑符号和逻辑功能，具体电路可参看有关书籍。

常用逻辑电路主要包括与门、或门、非门、异或门、与非门、或非门等，它们的真值表如表 1-1 所示。

表 1-1 常用逻辑电路真值表

输 入		输 出					
A	B	与门	或门	非门	异或门	与非门	或非门
0	0	0	0	1	0	1	1
0	1	0	1	1	1	1	0
1	0	0	1	0	1	1	0
1	1	1	1	0	0	0	0

注：非门没有 B 端。

2. 触发器和寄存器

触发器是计算机记忆装置的基本单元，各类触发器是由不同的门电路组成的。它具有把以前的输入“记忆”下来的功能，一个触发器能存储 1 位二进制代码。计算机中常用的几种触发器包括 R-S 触发器、D 触发器、J-K 触发器等。

寄存器是由触发器组成的。一个触发器就是一个 1 位寄存器。多个触发器就可以组成一个多位寄存器。由于寄存器在计算机中的作用不同，从而被命名为不同的名称。常见的寄存器有：缓冲寄存器、移位寄存器、计数器等，它们是组成计算机中存储器和定时器的基础。下面简要介绍这些寄存器的电路结构及工作原理。

(1) 缓冲寄存器 (Buffer)

它是用来暂存某个数据，以便在适当的时间节拍和给定的计算步骤将数据输入或输出到其他记忆元件中去。由于单片机与外设的传送速度不同，为达到速度匹配，常常需要采用数据缓冲器实现缓冲。

图 1-2 是一个并行输入与并行输出 4 位寄存器的电路原理图，它由 4 个 D 触发器组成。

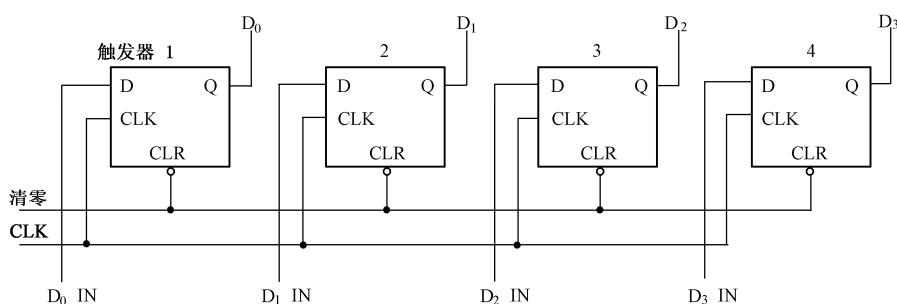


图 1-2 4 位缓冲寄存器电路原理图

启动时，先在清零端加清零脉冲，把各触发器置 0，即 Q 端为 0。然后把数据加到触发器的 D 输入端，在 CLK 时钟信号作用下，输入端的信息就保存在各触发器中 ($D_0 \sim D_3$)。

(2) 移位寄存器 (Shifting Register)

移位寄存器除了可以存放二进制数码外，还具有移位功能。移位寄存器能将所存储的数据逐位向左或向右移动，以达到计算机运行过程中所需的功能。图 1-3 所示即为一串行输入移位寄存器电路。

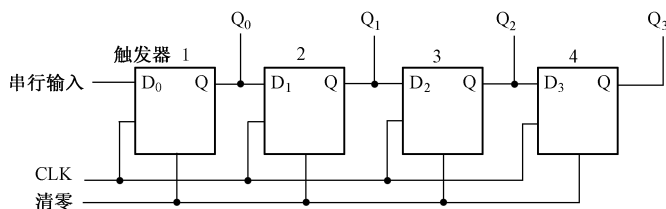


图 1-3 4 位串行输入移位寄存器

启动时，先在清零端加清零脉冲，使触发器输出置 0。然后第一个数据 D_0 加到触发器 1 的串行输入端，在第一个 CLK 脉冲的上升沿 $Q_0 = D_0$ ， $Q_1 = Q_2 = Q_3 = 0$ ；其后第二个数据 D_1 加到串行输入端，在第二个 CLK 脉冲到达时， $Q_0 = D_1$ ， $Q_1 = D_0$ ， $Q_2 = Q_3 = 0$ 。以此类推，当第四个 CLK 来到之后，各输出端分别是 $Q_0 = D_3$ ， $Q_1 = D_2$ ， $Q_2 = D_1$ ， $Q_3 = D_0$ 。输出数据可用

串行的形式取出，也可用并行形式取出。

(3) 计数器 (Counter)

计数器也是由若干个触发器组成的寄存器。计数器是一种累加时钟脉冲的逻辑部件，其特点是能够把存储在其中的数字进行加 1 操作（也有可以进行减 1 操作的计数器）。它不仅可用于时钟脉冲计数，还可以产生节拍脉冲；可用于定时、分频及数字运算等。它可以按触发方式，计数容量，数的增、减等方法分类。触发方式又分为同步和异步。同步计数器输入脉冲时触发器同时翻转，而异步计数器不是同时翻转。

由计数器组成的顺序脉冲发生器又称为节拍脉冲发生器，能产生一组在时间上有先后顺序的脉冲。这组脉冲可以使控制器形成所需要的各种控制信号。

计数器的种类很多，有行波计数器，同步计数器等，在此仅以行波计数器为例加以介绍。

图 1-4 是由 J-K 触发器组成的行波计数器的工作原理图。这种计数器的特点是：各位的 J、K 输入端都是悬浮的。这相当于 J、K 输入端都是置 1 的状态，亦即各位都处于准备翻转的状态。只要时钟脉冲边沿一到，最右边的触发器就会翻转，第一个时钟脉冲促使其最低有效位加 1，使其由 0 变 1。第二个时钟脉冲促使最低有效位由 1 变 0，同时推动第二位，使其由 0 变 1；同理，第二位由 1 变 0 时又去推动第三位，使其由 0 变 1。这样犹如水波前进一样逐位进位下去。

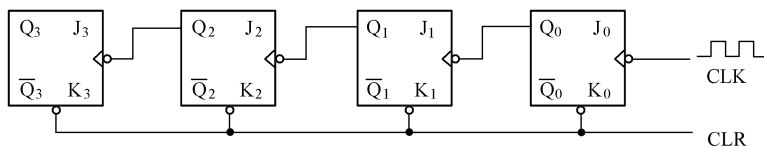


图 1-4 行波计数器工作原理图

图 1-4 中的计数器是 4 位的，因此可以计 0~15 的数。如果要计更多的数，需要增加位数，如 8 位计数器可计 0~255 的数，16 位则可以计 0~65535 的数。

(4) 三态门 (三态缓冲器)

为减少信息传输线的数目，大多数计算机中的信息传输线均采用总线形式，即凡要传输的同类信息都走同一组传输线，且信息是分时传送的。在计算机中一般有三组总线，即数据总线、地址总线、控制总线。为防止信息相互干扰，要求凡挂到总线上的寄存器或存储器的输出端不仅能呈现 0、1 两个信息状态，而且还应能呈现第三种状态——高阻抗状态（又称悬浮状态），即此时它们的输出好像被开关断开，对总线状态不起作用，此时总线可由其他器件占用。三态门可实现上述功能，它除具有输入、输出端之外，还有控制端，如图 1-5 所示。

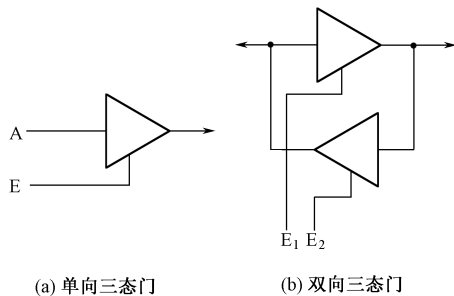


图 1-5 三态门

对于图 1-5 (a) 单向三态门的情况，当控制端 $E=1$ 时，输出=输入，此时总线由该器件驱动，总线上的数据由输入数据决定，当 $E=0$ 时，输出端呈高阻抗状态，该器件对总线不起作用。当寄存器输出端接至三态门，再由三态门输出端与总线连接起来，就构成三态输出的缓冲寄存器。图 1-6 所示即为一个 4 位的三态输出缓冲寄存器，对于单向三态门，数据只能从寄存器输出至数据总线。如果来实现双向传送，则要用双向三态门，如图 1-5 (b) 所示。上述电路是构成计算机的基础电路。

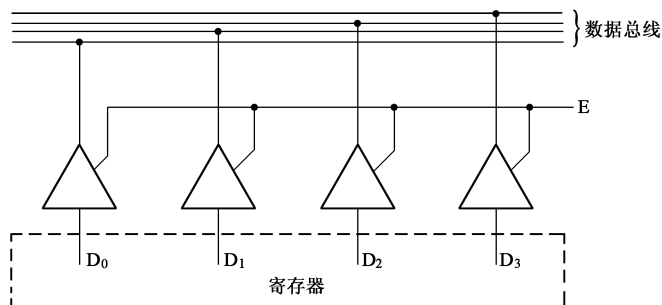


图 1-6 三态输出缓冲寄存器

1.4.3 存储器简介

存储器是计算机的主要组成部分，其用途是存放程序和数据，使计算机具有记忆功能。这些程序和数据在存储器中是以二进制代码表示的，根据计算机命令，按照指定地址，可以把代码取出来或是存入新代码。在计算机的指令中几乎有一半都涉及存储器，因而了解存储器便于对后面章节的学习与理解。

1. 存储器的分类

与计算机有关的存储器种类很多，因而存储器的分类方法也较多。例如，从其组成材料和单元电路类型来划分，可分为磁芯存储器（早期产品）、半导体存储器（从制造工艺方面又可分为 MOS 型存储器、双极型存储器等）、电荷耦合存储器等；从其与微处理器的关系来划分，又可分为内存和外存。通常把直接同微处理器进行信息交换的存储器称为“内存”，其特点是存取速度快，但容量有限；而把通过内存间接与 CPU 进行信息交换的存储器称为“外存”，如磁盘、光盘、磁带等，其特点是容量大，速度较慢，且独立于计算机之外，便于携带与存放。可根据需要，随时把外存的内容调入内存，或把内存的内容写入外存。因为在单片机中主要是采用半导体存储器，因而在此仅对半导体存储器进行介绍。

2. 半导体存储器的分类

通常人们习惯于按存储信息的功能分类，下面将按照半导体存储器的不同功能特点进行分类。

1) 只读存储器 ROM (Read Only Memory)

只读存储器在使用时只能读出而不能写入，断电后 ROM 中的信息不会丢失。因此一般用来存放一些固定程序，如监控程序、字库及数据表等。按存储信息的方法 ROM 又可分为以下 3 种。

(1) 掩膜 ROM。

掩膜 ROM 也称固定 ROM，它是由厂家编好程序写入 ROM（称固化）供用户使用，用户不能更改它。其价格最便宜。

(2) 可编程序的一次只读存储器 OTP (Only Time Programmable)

它的内容可一次性写入，一旦写入，只能读出，而不能再进行更改。这类存储器以前称为可编程序的只读存储器 PROM (Programmable Read Only Memory)。

(3) 可电改写只读存储器 EEPROM (Electrically Erasable Programmable Read Only Memory)。

EEPROM 可用电的方法写入和清除其内容，其编程电压和清除电压均与 CPU 的 5V 工作电压相同，不需另加电压，它既有 RAM 读写操作简便、又有数据不会因掉电而丢失的优点。除此，E²PROM 保存的数据至少可达 10 年以上，每块芯片可擦写 1 千次以上。

2) 随机存储器 RAM (Random Access Memory)

这种存储器又叫读写存储器，它不仅能读取存放在存储单元中的数据，还能随时写入新的数据，写入后原来的数据就丢失了，断电后 RAM 中的信息全部丢失，因此 RAM 常用于存放经常要改变的程序或中间计算结果等。

RAM 按照存储信息的方式，又可分为静态和动态两种。

(1) 静态 SRAM (Static RAM)

其特点为只要有电源加于存储器，数据就能长期保留。

(2) 动态 DRAM (Dynamic RAM)

写入的信息只能保持若干毫秒时间，因此每隔一定时间必须重新写入一次，又称刷新，以保持原来的信息不变。故动态 RAM 控制电路较复杂，但动态 RAM 价格比静态便宜。

3) 可改写的非易失存储器

由于多数 E²PROM 的最大缺点是改写信息的速度还比较慢。随着新的半导体存储技术的发展，各种新的可现场改写信息的非易失存储器被推上市场，且发展速度很快，这主要有快擦写存储器 (Flash Memory)、新型非易失静态存储器 NVSRAM (Non Volatile SRAM)、磁阻式存储器 MRAM (Magnetic-Resistive RAM) 和铁电存储器 FRAM (Ferroelectric RAM)。这些存储器的共同特点是从原理上看，它们属于 ROM 型存储器，但是从功能上看，它们又可以随时改写信息，因而作用又相当于 RAM。随着存储器技术的发展，过去传统意义上的易失性存储器、非易失性存储器的概念已经发生变化，所以 ROM、RAM 的定义和划分已不是很严格。但由于这类存储器写的速度还是慢于一般的 RAM，所以在单片机中主要还是用作程序存储器；只是当需要重新编程，或者某些数据修改后需要保存时，采用这种存储器十分方便。

目前应用最广泛的是 Flash 存储器，它是在 EEPROM 的制造基础上产生的一种非易失存储器，它的读写速度比一般的 EEPROM 快得多。Flash 存储器现在均简称为闪存，现在的单片机产品多数程序存储器都已经配置为闪存。

3. 存储器中常用名词术语

(1) 存储单元——存储器是由大量缓冲寄存器组成的，其中每一个寄存器就称为一个存储单元。如图 1-2 中的缓冲寄存器就可以作为一个 4 位的存储单元，它可存放一个有独立意义的二进制代码。

(2) 位 (bit) ——计算机中最小的数据单元，是一个二进制位。

(3) 字长——代码的位数称为字长，习惯上也称为位长。基本字长一般是指参加一次运算的操作数的位数。基本字长可反映寄存器、运算部件和数据总线的位数。在计算机中每个存储单元存放二进制数的位数一般情况下和它的算术运算单元的位数是相同的。例如 8 位单片机的算术运算单元是 8 位，则其字长就是 8 位。

(4) 字节 (Byte) ——在计算机中把一个 8 位的二进制代码称为一个字节 (Byte)，常简称为 B。这是计算机中最通用的基本单元。一个字节的最低位称为第 0 位 (位 0)，最高位称为第 7 位 (位 7)。2 个字节称为一个字 (Word)，4 个字节称为双字 (Double Word)，以上均为代码位数常用单位。

(5) 容量单位——容量指一片存储器能容纳的最大存储单元数，由于现在存储器的位长有 4、8、16 位的，所以，在标注存储器容量时，经常同时标出存储单元的数目和位数；因此，存储芯片容量=单元数×数据线位数，通常把乘积结果再换算为字节单位。

例如：Intel 6264 芯片容量 = $8K \times 8\text{bit/片} = 64Kb/\text{片} = 8KB/\text{片}$ 。常用单位有 KB (1KB=1024B)，MB (1MB=1024KB)，GB (1GB 千兆=1024MB)，都为容量常用单位。

4. 存储单元和存储单元地址

本小节所介绍的半导体存储器（以下简称“存储器”）仅限于在单片机片内使用时的情况，暂不涉及作为一块独立的存储器芯片的情况。

存储器是由大量缓冲寄存器组成的，其中的每一个寄存器称为一个存储单元。例如图 1-2 中的缓冲寄存器就可以作为一个 4 位的存储单元，它可存放一个有独立意义的二进制代码。一个代码由若干位 (bit) 组成。

在计算机的存储器中往往有成千上万个存储单元。为了使存入和取出不发生混淆，必须给每个存储单元一个惟一的固定编号，这个编号就称为存储单元的地址。因为存储单元数量很大，为了减少存储器向外引出的地址线，故存储器内部都带有译码器。根据二进制编码、译码的原理，除地线公用之外， n 根导线可译成 2^n 个地址号。即地址线数 n 与存储单元数 N 的关系可以表达为 $N=2^n$ 。例如，当地址线为 3 根时，可译成 $2^3=8$ 个地址号；当地址线为 8 根时，可以译成 $2^8=256$ 个地址号。以此类推，在 80C51 单片机中，地址线为 16 根，则可译成 $2^{16}=65536$ 个地址号，也称为 16 根地址线的最大寻址范围。需要注意的是存储单元数是可以小于最大寻址范围的。

由此可见，存储单元地址与该存储单元的内容含义是不同的。存储单元如同一个旅馆中的每个房间，存储单元地址则相当于每个房间的房号，存储单元内容（二进制代码）就是这个房间中的房客。表 1-2 所列为程序存储器和数据存储器中部分存储单元的地址和内容，均用十六进制数表示。

表 1-2 存储器地址和内容

程序存储器		数据存储器	
地址	内容	地址	内容
0000	02	0206	3A
0001	00	0207	44
0002	30	0208	C0
...

5. 存储器的寻址原理

对于存储器工作原理的理解很大程度上决定于对存储器寻址原理的理解,由于篇幅所限,本书仅以随机存储器为例,简介 CPU 在读出存储单元信息时的寻址原理。

存储器一般由地址译码器、存储体、输入/输出缓冲器和读写控制电路等组成。一个随机存储器的基本结构框图如图 1-7 所示。

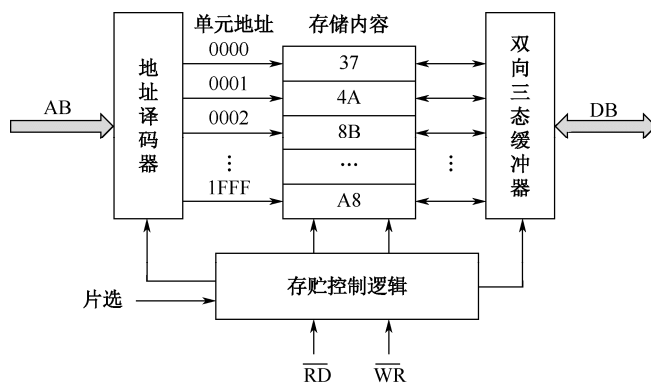


图 1-7 随机存储器的基本结构框图

图中 AB 为地址线, DB 为数据线, \overline{RD} 和 \overline{WR} 分别为读、写线, 片选线用于选择该存储体, 其主要部分说明如下。

1) 存储体

存储体也称为存储阵列, 它是存储器中所有存储单元的集合, 通常它是由一个或者几个存储器芯片组成。这些存储单元按一定方式排列, 每个单元都有一个唯一的地址, 在一个存储单元中可以存储一位或者多位二进制数, 在 8 位单片机中通常都是 8 位二进制数。假设图 1-7 中的地址线为 13 根, 8 位单片机每个存储单元是 8 位, 其最多可寻址 8192 个单元, 且该存储体的容量为 8 KB 个单元, 则其最大寻址单元地址如图示为 1FFFH。

2) 地址译码器

地址译码器用于对输入地址译码, 以选择指定的存储单元。译码器的译码方式与存储器的结构有关, 主要有单译码方式和复合译码方式, 篇幅关系在此省略。

3) 控制逻辑电路

控制逻辑电路接收来自 CPU 的读写信号, 并根据地址译码的结果, 控制存储器选中单元的数据读/写。片选信号将启动这个存储体进入工作状态。

4) 双向三态缓冲器

存储器与数据线相连的双向三态缓冲器用于暂存输出/输入的数据。

读/写操作过程如下:

① 存储器的读操作。

CPU 首先通过地址线, 经译码选中某一单元, 然后发读命令, 当读信号有效时, 则选中单元的数据读出, 经双向三态缓冲器送到数据总线。

② 存储器的写操作。

CPU 首先通过地址线, 经译码选中某一单元, 同时, 要写入的数据经数据总线送至双向三态缓冲器。当 CPU 发写信号时, 通过内部控制电路, 将外部数据线上的数据写入到所选中的单元。

思考与练习

1. 将下列各二进制数转换为十进制数。
(1) 11011110B (2) 01011010B (3) 10101011B (4) 1011111B
2. 将第 1 题中各二进制数转换为十六进制数。
3. 将下列各数转换为十六进制数。
(1) 224D (2) 143D (3) 01010011BCD (4) 00111001BCD
4. 什么叫原码、反码及补码？
5. 已知原码如下，请写出其补码和反码（其最高位为符号位）。
(1) $[X]_{\text{原}} = 01011001$ (3) $[X]_{\text{原}} = 11011011$
(2) $[X]_{\text{原}} = 00111110$ (4) $[X]_{\text{原}} = 11111100$
6. 当计算机把下列数看成无符号数时，它们相应的十进制数为多少？若把它们看成是补码，最高位为符号位，那么它们相应的十进制数是多少？
(1) 10001110 (2) 10110000 (3) 00010001 (4) 01110101
7. 触发器、寄存器及存储器之间有什么关系？
8. 三态门有何作用？其符号如何画？
9. 除地线公用外，6 根地址线和 11 根地址线各可选多少个地址？
10. 存储器分几类？各有何特点和用处？
11. 假定一个存储器，有 4096 个存储单元，其首地址为 0，则末地址为多少？

第2章 单片机结构及原理

本章内容是单片机的硬件基础知识，是本书的重点也是难点。本章将以 80C51 系列的 AT89S51/S52 为典型例子，详细介绍单片机的结构、工作原理、存储器结构、时序及复位电路等内容。通过对这些内容的掌握，使读者学习和了解其他单片机时可以起到举一反三、触类旁通的作用。

2.1 单片机结构

本节将以 89 系列单片机中 ATMEL 公司的 AT89S51/S52 单片机为例，介绍单片机的组成、结构及引脚功能，为读者了解它的工作原理打下基础。89 系列单片机是 80C51 系列单片机的一个子系列。文中内容凡属于与 80C51 系列单片机相同处，均用 80C51 为代表予以介绍。

AT89S51/S52 单片机与 Intel 公司 MCS-51 系列的 80C51 型号单片机在芯片结构与功能上基本相同，外部引脚完全相同。主要不同点是 89 系列产品中程序存储器全部采用快擦写存储器，简称闪存。此外，ATMEL 公司的 AT89S51/S52 单片机与在 2003 年停产的 AT89C51/C52 单片机主要不同点是增加了 ISP 串行接口（可实现串行下载功能）和看门狗定时器等。本书中提到的 89C51/C52 可以是 ATMEL 公司的产品也可以是 Philips 公司或其他公司的产品。

2.1.1 单片机组成及结构

本节介绍单片机的组成及结构，为读者了解其工作原理奠定基础。

1. 标准型单片机的组成

AT89S51/S52 属于标准型单片机，其基本组成如图 2-1 所示。从图中可以看出，在这块芯片上集成了一台微型计算机的各个主要部分，包括 CPU、存储器、I/O 口、串行 I/O 口、定时/计数器等，各部分通过内部总线相连。

如图 2-1 所示的 AT89S51/S52 单片机基本组成简要介绍如下。

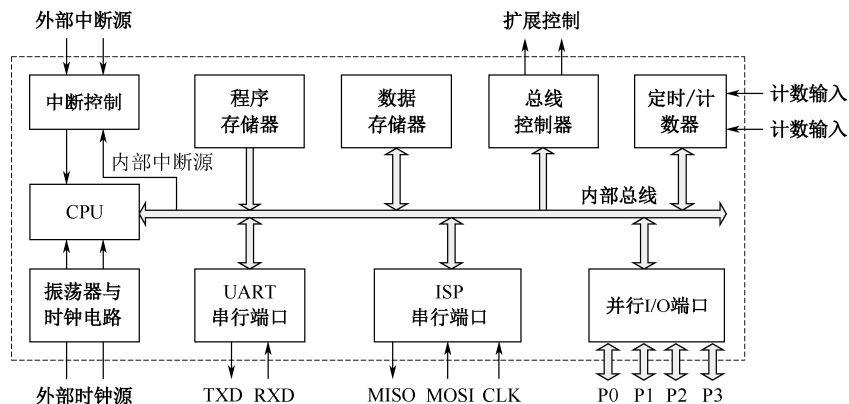


图 2-1 AT89S51/S52 的基本组成功能框图

中央处理器（CPU）

中央处理器是单片机最核心的部分，是单片机的大脑，主要完成运算和控制功能。这一点与通用微处理器基本相同，只是中央处理器的控制功能更强。80C51 系列的 CPU 是一个字长为 8 位的中央处理单元，它对数据的处理是以字节为单位进行的。CPU 中的主要部件例如 PC、ALU、指令寄存器等的功能将在 2.3 节叙述。

- 数据存储器（内部 RAM）

数据存储器用于存放变化的数据。在 80C51 单片机中，通常把控制与管理寄存器（简称为“专用寄存器”）在逻辑上划分在内部 RAM 中，因为其地址与 RAM 是连续的。AT89S51 中数据存储器的地址空间为 256 个 RAM 单元，但其中能作为数据存储器供用户使用的仅有前面 128 个，后 128 个被专用寄存器占用；AT89S52 中可供用户使用的数据存储器比 AT89S51 多 128 个，共 256 个。

- 程序存储器（内部 ROM）

程序存储器用于存放程序和固定不变的常数等。通常采用只读存储器，且其有多种类型，在 89 系列单片机中全部采用闪存。AT89S51/C51 内部配置了 4KB 闪存，AT89S52/C52 配置了 8KB 闪存。

- 定时/计数器

定时/计数器用于实现定时和计数功能。AT89S51 共有 2 个 16 位定时/计数器，AT89S52 共有 3 个 16 位定时/计数器（详见第 6 章）。

- 并行 I/O 口

AT89S51/S52 共有 4 个 8 位的 I/O 口（P0、P1、P2、P3），这些 I/O 口主要是用于实现与外部设备中数据的并行输入/输出，但有些 I/O 口的引脚还可作为其他功能电路的引脚，详见 2.5 节。

- 串行口

AT89S51/S52 有 1 个 UART 全双工异步串行口，用以实现单片机和其他具有相应接口的设备之间的异步串行数据传送（详见第 7 章）。AT89S51/S52 还有一个 ISP 串行接口，用于实现串行在线下载程序。

- 时钟电路

振荡器与定时与控制电路共同组成了时钟电路，其作用是产生单片机工作所需要的时钟脉冲序列。89 系列单片机内部的时钟电路需要外接晶振和微调电容才能工作（详见 2.6 节）。几年前已经出现了可以不需外接晶振和微调电容的单片机型号。

- 中断系统

中断系统的主要作用是对外部或内部的中断请求进行管理与处理，有关中断的作用及使用方法详见第 8 章。AT89S51/S52 的中断系统可以满足一般控制应用的需要：AT89S51 共有 5 个中断源，其中有 2 个外部中断源 INT0 和 INT1、3 个内部中断源（2 个定时/计数中断和 1 个串行口中断）；此外，AT89S52 还增加了一个定时器 2 的中断源。

- 总线控制器

当该单片机需要外扩外围接口芯片时，用于控制外接芯片的寻址与数据传输。

2. 单片机的内部结构

图 2-2 所示为 AT89S51/S52 的内部结构框图。从图中可以看出，单片机内部除了有 CPU、

RAM、ROM、定时器和串行口等主要功能部件之外，还有驱动器、锁存器、地址寄存器等辅助电路部分，在结构图中 CPU 的主要组成部分例如指令寄存器、指令译码器、ALU 等全部按基本运行关系分别画于图中，由图 2-2 还可以看出各功能模块在单片机中的相互关系。

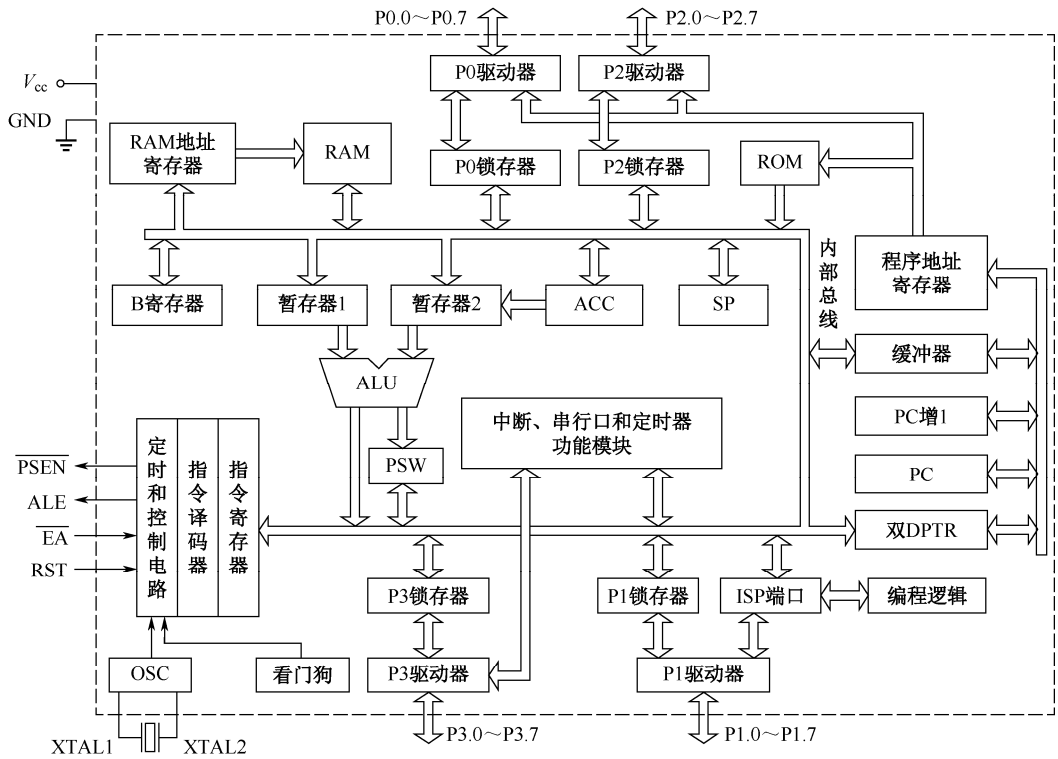


图 2-2 AT89S51/S52 内部结构框图

图 2-2 中的 4 个并行 I/O 端口都配置了 1 个驱动器和 1 个锁存器。UART 串行口的输出/输入线是利用了 P3 端口 2 个引脚的第二功能实现，ISP 串行口的输出/输入线是利用了 P1 端口 3 个引脚的第二功能实现（详见 2.5 节）。

图 2-2 中 RAM 地址寄存器用于存放 RAM 地址值，程序地址寄存器用于存放 ROM 地址值，还用于存放 P0 和 P2 地址值，这 2 个端口在单片机访问片外存储器或者其他外设时将被用作地址/数据总线的控制器如图 2-1 所示。

图 2-2 中的定时和控制电路包括了时序部件及控制部件，作用详见 2.6 节。

图 2-2 中 PSW、ACC 等部件的作用将在 2.3 和 2.4 节陆续介绍。

AT89S51 与 AT89S52 的主要差别是程序存储器和 RAM 容量不同，且 AT89S52 还增加了一个定时器 2，其余完全相同。

AT89C51/C52 与 AT89S51/S52 在结构上的主要不同点是，后者有看门狗、双 DPTR 和 ISP 端口。

由上述可知，虽然 AT89S51/S52 仅是一块芯片，但它包括了构成计算机的基本部件，因此，可以说它是一台简单的计算机。又由于其主要作用是控制，所以又称为“微控制器”。

本章重点介绍 AT89S51/S52 的 CPU 部分、并行 I/O 口、存储器、时钟电路等，硬件结构的其他部分将在以后章节中陆续介绍。

2.1.2 引脚定义及功能

AT89S51/S52 单片机实际有效的引脚为 40 个，有 3 种封装形式，引脚图如图 2-3 所示。(a)是 DIP(Dual Inline Package)封装形式，这是普通 40 脚双列直插形式；(b)是 LCC(Leaded Chip Carrier)封装形式，这种形式是具有 44 个“J”形脚的方型芯片，使用时需要插入与其相配的方型插座中；(c)是 PQFP (Plastic Quad Flat Package)封装形式，这种形式也是具有 44 个“J”形脚的方型芯片，但它的体积更小、更薄，是一种表面贴焊的封装形式。因为不同封装形式的引脚排列不一致，所以在使用时一定要注意。

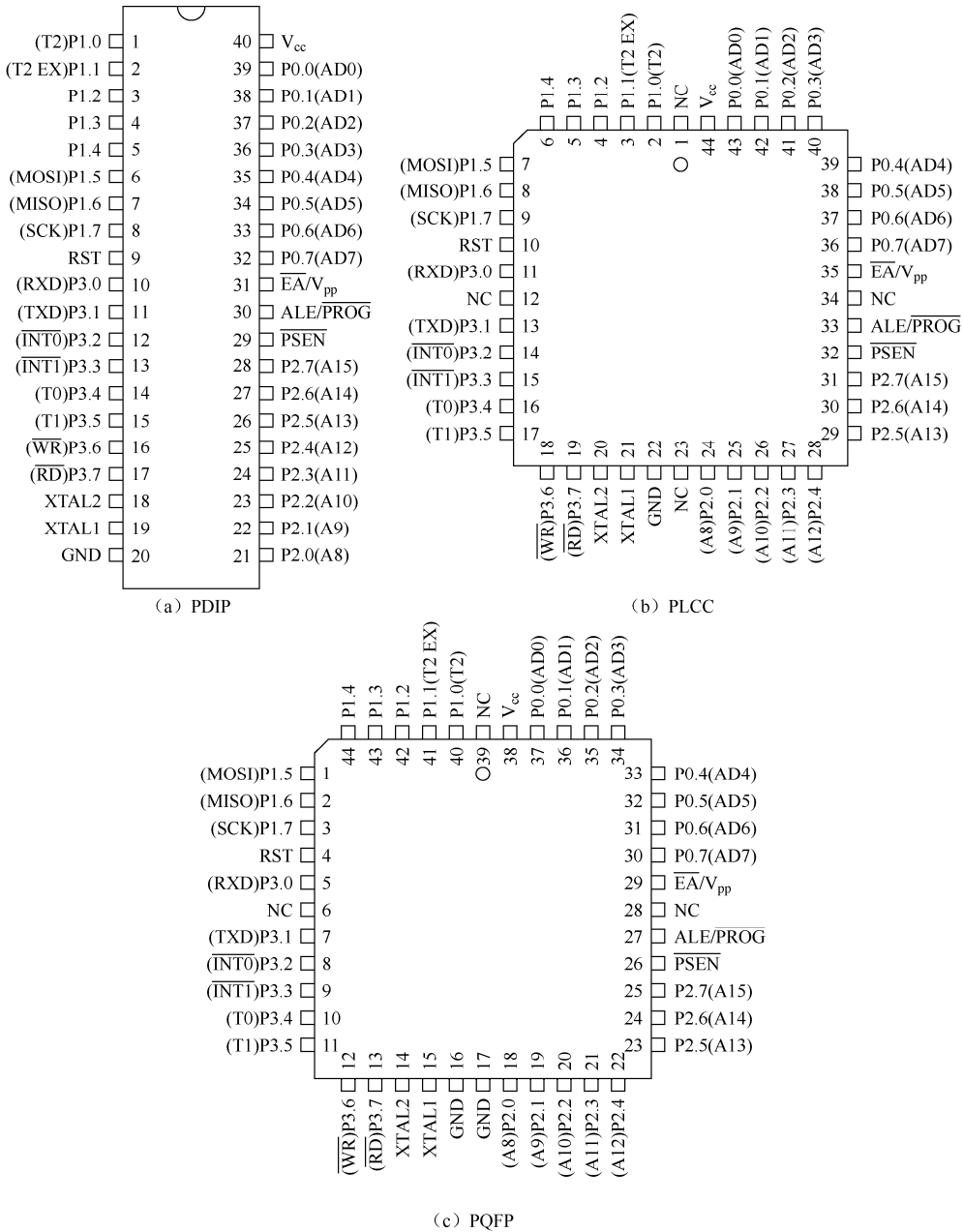


图 2-3 AT89S51/S52 引脚图

为了尽可能缩小体积，减少引脚数，AT89S51/S52 单片机的不少引脚具有第二功能，也称为复用功能。下面说明这些引脚的名称和功能。

1. 主电源引脚 GND 和 VCC

GND：接地。

VCC：主电源+4.5~+5.5V。

2. 时钟电路引脚 XTAL1 和 XTAL2

XTAL1：接外部晶体的一端。它是片内振荡器反相放大器的输入端。在采用外时钟时，外部时钟振荡信号直接送入此引脚作为驱动端。

XTAL2：接外部晶体的另一端。它是片内振荡器反相放大器的输出端，振荡电路的频率是晶体振荡频率。若需采用外部时钟电路时，此引脚应悬空不用。

3. 控制信号引脚 RST、 $\overline{\text{ALE/PROG}}$ 、 $\overline{\text{PSEN}}$ 、 $\overline{\text{EA/VPP}}$

- RST：复位输入端。在该脚输入 2 个机器周期以上的高电平将使单片机复位。

- $\overline{\text{ALE/PROG}}$ ：地址锁存允许输出/编程脉冲输入端。这个引脚具有 2 种功能。

在访问片外存储器时，ALE 作为锁存扩展地址低位字节的输出控制信号（称允许锁存地址）。平时不访问片外存储器时，该端也以六分之一的时钟振荡频率固定输出正脉冲，供定时或其他需要使用。ALE 端的负载驱动能力为 8 个 LSTTL（低功耗高速 TTL）。

在对片内存储器编程（固化）时，此引脚用于输入编程脉冲，此时为低电平有效。

- $\overline{\text{PSEN}}$ ：片外程序存储器选通控制信号端。在访问片外程序存储器时，此端输出负脉冲作为程序存储器读选通信号。CPU 在向片外程序存储器取指令期间， $\overline{\text{PSEN}}$ 信号在 12 个时钟周期中两次生效。不过，在访问片外数据存储器时，这两次有效的 $\overline{\text{PSEN}}$ 信号不出现。 $\overline{\text{PSEN}}$ 端同样可驱动 8 个 LSTTL 负载。

- $\overline{\text{EA/VPP}}$ ：为内、外程序存储器选择/编程电源输入端。这个引脚具有 2 种功能。

当 $\overline{\text{EA}}$ 端接高电平时，CPU 从片内程序存储器地址 0000H 单元开始执行程序。当地址超出 4KB（对于 89S52 为 8KB）时，将自动执行片外程序存储器的程序；当 $\overline{\text{EA}}$ 端接低电平时，CPU 仅访问片外程序存储器，即 CPU 直接从片外程序存储器地址 0000H 单元开始执行程序。

在对闪存编程时，此引脚用于施加编程电压 VPP。80C51 系列不同型号单片机的编程电压不同，有 12V 和 5V 两种。

4. 输入/输出引脚（P0、P1、P2 和 P3 端口引脚）

P0~P3 是 AT89S51/S52 单片机与外界联系的 4 个 8 位双向并行 I/O 端口，其工作原理与使用详见 2.5 节。

- P0.0~P0.7：P0 口的 8 位 I/O 端口。在访问片外存储器时，它分时提供低 8 位地址和 8 位数据，故这些 I/O 线有地址/数据总线之称，简写做 AD0~AD7。在不做总线时，也可以作为普通 I/O 口使用。

在对程序存储器编程时，从 P0 输入指令字节；在验证程序时，则输出指令字节（验证时，要外接上拉电阻）。

- P1.0~P1.7: P1 口的 8 位准双向 I/O 端口。AT89S51/S52 单片机的 P1 口除了可以作为一般的 I/O 口外，其中 5 位还有第二功能，见表 2-1。

表 2-1 P1 口各位的第二功能

P1 口的各位	第二功能的名称及作用
P1.0	T2 (定时/计数器 2 的外部计数输入/时钟输出)
P1.1	T2EX (定时/计数器 2 的捕获触发和双向控制)
P1.5	MOSI (输入线，用于在系统编程)
P1.6	MISO (输出线，用于在系统编程)
P1.7	SCK (串行时钟线，用于在系统编程)

由表 2-1 可见 P1.0、P1.1 用于定时器 2 (AT89S51 除外)，P1.5、P1.6、P1.7 用于 ISP (IN System Programmable)，称为在系统中编程。它的作用是把在 PC 上编好的程序通过所定义的这 3 根 ISP 接口线进行在线下载，即直接传输并且烧录到 AT89S51/S52 单片机的闪存中。

烧录时，RST 引脚要接到 Vcc 端；编程前，首先要擦除该芯片，接入 SCK 引脚的时钟频率不能大于单片机频率的 1/16。

这种方法比使用一般的编程器更廉价、方便。一般厂商都配有在线下载接口板和相应软件，读者只需学会如何使用即可。

- P2.0~P2.7: P2 口的 8 位准双向 I/O 端口。在访问片外存储器时，它输出高 8 位地址，即 A8~A15。在不做总线时，也可以作为普通 I/O 口使用。
在对闪存编程和验证程序时，它输入高 8 位地址（这个作用一般用户不必关心）。
- P3.0~P3.7: P3 口的 8 位准双向 I/O 端口。这 8 个引脚都具有专门的第二功能，见表 2-2。

表 2-2 P3 口各位的第二功能

P3 口的各位	第二功能的名称及作用
P3.0	RXD (串行口输入)
P3.1	TXD (串行口输出)
P3.2	$\overline{\text{INT0}}$ (外部中断 0 输入)
P3.3	$\overline{\text{INT1}}$ (外部中断 1 输入)
P3.4	T0 (定时/计数器 0 的外部输入)
P3.5	T1 (定时/计数器 1 的外部输入)
P3.6	$\overline{\text{WR}}$ (片外数据存储器写选通控制输出)
P3.7	$\overline{\text{RD}}$ (片外数据存储器读选通控制输出)

以上各引脚的功能与作用只有在后面章节的学习中才能逐渐加深理解并学会应用。

2.2 80C51 的存储器

存储器是单片机的重要组成部分，不同计算机存储器的用途是相同的，但结构与存储容量却不完全相同。本节以 80C51 系列单片机的 AT89S51/S52 型号为例介绍其存储器的结构、地址空间及程序与数据存储器各自配置特点。

2.2.1 存储器结构和地址空间

80C51 系列单片机的存储器结构与一般通用计算机不同。一般通用计算机通常只有一个逻辑空间，即它的程序存储器和数据存储器是统一编址的。访问存储器时，同一地址对应唯一的存储空间，可以是 ROM 也可以是 RAM，并用同类访问指令，这种存储器结构称为冯·诺伊曼结构。而 80C51 系列单片机的程序存储器和数据存储器在物理结构上是分开的，这种结构称为哈佛结构，80C51 系列单片机的存储器在物理结构上可以分为 4 个存储空间：片内程序存储器、片外程序存储器、片内数据存储器、片外数据存储器。

图 2-4 为 AT89S51 存储器空间分布图，AT89S52 的存储器空间分布与其略有不同，待后详述。由图 2-4 可以清楚地看出这 4 个存储空间的地址范围。

在此要特别说明的是，随着单片机片上存储器容量的不断加大，在很多情况下只需要采用片上的 2 个存储器空间（见图 2-4 的虚线框内）即可，片外的存储器空间不是必须使用的。

这种结构在物理上是把程序存储器和数据存储器分开的，但在逻辑上，即从用户使用的角度上，80C51 系列有 3 个存储空间：片内外统一编址的 64KB 的程序存储器地址空间（用 16 位地址）；片内数据存储器地址空间，寻址范围为 00~FFH；64KB 片外数据存储器地址空间。

由图 2-4 可以看出，片内程序存储器的地址空间（0000~0FFFH）与片外程序存储器的低地址空间是相同的，片内数据存储器的地址空间（00~FFH）与片外数据存储器的低地址空间是相同的。

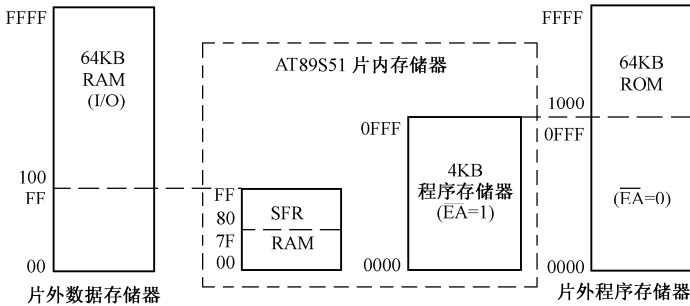


图 2-4 AT89S51 存储空间分布图

通过采用不同形式的指令（见第 3 章），产生不同存储空间的选通信号，可以访问 3 个不同的逻辑空间。

下面分别叙述程序存储器和数据存储器的配置特点。

2.2.2 程序存储器

程序存储器用于存放编好的程序和数据表格。

1. 程序存储器的结构和地址分配

AT89S51 片内有 4KB(AT89S52 为 8KB)闪存，通过片外 16 位地址线最多可扩展到 64KB，两者是统一编址的。如果 \overline{EA} 端保持高电平，AT89S51 的程序计数器 PC 在 0000H~0FFFH 范围内（即前 4KB 地址）是执行片内 ROM 的程序。当寻址范围在 1000H~FFFFH 时，则从

片外存储器取指令。当 \overline{EA} 端保持低电平时，AT89S51 的所有取指令操作均在片外程序存储器中进行，这时片外存储器可以从 0000H 开始编程。AT89S52 的片内程序地址在 0000H~1FFFH 范围内（即前 8KB 地址）。

2. 程序存储器的入口地址

在程序存储器中，以下 7 个单元具有特殊用途。

0000H: 51 系列单片机上电复位后，PC=0000H，程序将自动从 0000H 开始执行指令。

0003H: 外部中断 0 入口。

000BH: 定时器 0 溢出中断入口。

0013H: 外部中断 1 入口。

001BH: 定时器 1 溢出中断入口。

0023H: 串行口中断入口。

002BH: 定时器 2 溢出中断入口（只有 AT89S52 有）。

在上述地址中 0000H 是单片机复位后的起始地址，通常在设计程序时应该在 0000H~0002H 存放一条无条件跳转指令（见第 3 章）跳转到用户设计的主程序入口地址。在 0003H~0002BH 之间的 6 个单元是外部中断 0 等 6 个中断源的中断程序入口地址。通常在这些入口地址处存放一条绝对跳转指令，使程序跳转到用户安排的中断程序起始地址（详细说明见第 8 章）。

2.2.3 数据存储器

单片机中的数据存储器主要用于存放经常要改变的中间运算结果、数据暂存或标志位等，通常都是由随机存储器 RAM（Random Access Memory）组成的。数据存储器可分为片内和片外 2 个部分；如果片内够用，则不必扩充片外的数据存储器。

1. 片内数据存储器的结构及操作

数据存储器的配置如图 2-5 所示。片内数据存储器为 8 位地址，寻址范围为 00~FFH，AT89S51 片内供用户使用的 RAM 为片内低 128 字节，地址范围为 00~7FH，对其访问可采用直接寻址和间接寻址的方式（见 3.2 节）。其中 80H~FFH 为特殊功能寄存器 SFR（Special Function Register）所占用的空间。图 2-5 中*表示仅 AT89S52 才有的寄存器，这些寄存器只能采用直接寻址方式访问。

AT89S52 片内供用户使用的 RAM 为 256 字节，地址范围为 00~FFH。显然，80H~FFH 这个存储器空间还有与专用寄存器区地址相同的 128 字节数据存储器，它们的地址是重叠的，通过采用不同的寻址方式区分它们。对于 AT89S52 内 80H~FFH RAM 区的访问只能采用间接寻址方式访问，访问实例见 3.3.1 节。

特殊功能寄存器虽然在地址空间上被划分在数据存储器中，但它们并不是作为数据存储器使用的，它们的作用非常重要，将在 2.3 节专门介绍。

2. 低 128 字节 RAM

在低 128 字节 RAM 区中，根据存储器的用途，又可以分为 3 个部分，见图 2-5。其中

00H~1FH 地址空间为通用工作寄存器区，20H~2FH 地址空间为位寻址区，30H~7FH 地址空间为用户 RAM 区。下面分别予以介绍。

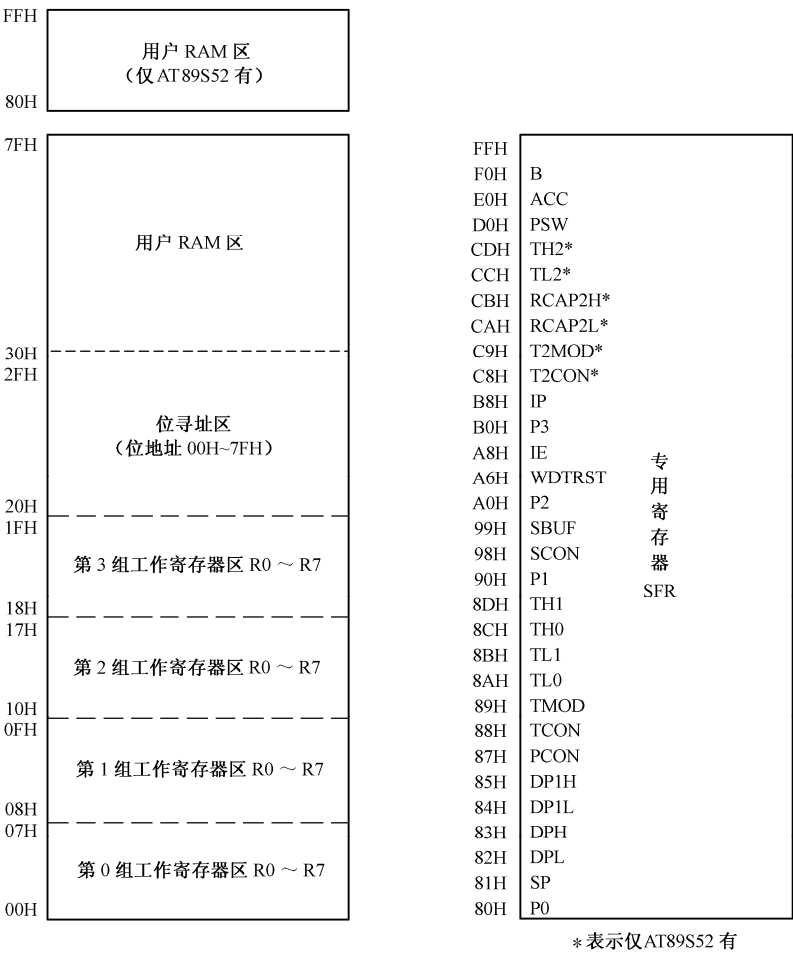


图 2-5 片内数据存储器的配置

(1) 通用工作寄存器区

80C51 系列单片机的工作寄存器共分为 4 组，每组由 8 个工作寄存器（R0~R7）组成，共占 32 个单元，表 2-3 为工作寄存器的地址表。每组寄存器均可选做 CPU 当前的工作寄存器组，通过对程序状态字 PSW（见 2.3 节）中 RS1、RS0 的设置来决定 CPU 当前使用哪一组。若程序中并不需要 4 组，那么其余的可作为一般的数据存储器。在 CPU 复位后，选中第 0 组工作寄存器。

表 2-3 工作寄存器的地址表

组	RS1 RS0	R0	R1	R2	R3	R4	R5	R6	R7
0	0 0	00H	01H	02H	03H	04H	05H	06H	07H
1	0 1	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
2	1 0	10H	11H	12H	13H	14H	15H	16H	17H
3	1 1	18H	19H	1AH	1BH	1CH	1DH	1EH	1FH

(2) 位寻址区

工作寄存器区后的 16 字节（即 20H~2FH），称为位寻址区，可用位寻址方式（见 3.2 节）访问其每个位，这 128 个位的位地址（位地址指的是某个二进制位的地址）为 00H~7FH，字节地址与位地址的关系见表 2-4。它们可用做软件标志位或用于 1 位（布尔）的处理。这种位寻址能力体现了单片机主要用于控制的重要特点。

表 2-4 RAM 位寻址区位地址表

字节地址	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

在此要特别说明的一点是，通用工作寄存器区和位寻址区在不作为寄存器或位寻址时都可作为一般的用户数据区。

(3) 用户 RAM 区

30H~7FH 为用户 RAM 区，可以通过直接或间接寻址方式（见第 3 章）访问这个 RAM 区，对于不使用的通用寄存器或位寻址区，它们也都可以作为一般的 RAM 使用。例如，如果在程序中只用到第 0 组通用寄存器，那么 08H~1FH 的区域就可以作为一般的 RAM 使用。

在 AT89S52/C52 单片机中还增加了 128 字节的用户 RAM 区，其地址范围为 80~FFH，与 SFR 的地址相同。通过采用不同的寻址方式访问它们加以区别，访问 SFR 必须采用直接寻址方式，访问 AT89S52/C52 增加的 128 字节用户 RAM 区，需要采用间接寻址方式。

3. 片外数据存储器的结构及操作

片外数据存储器最多可扩充到 64KB，由图 2-4 可见片内 RAM 和片外 RAM 的低地址部分（00~0FFH）的地址码是相同的，但它们却是两个地址空间。区分这两部分地址空间的方法是采用不同的寻址指令，访问片内 RAM 用“MOV”指令，访问片外 RAM 用“MOVX”指令。

对片外数据存储器采用间接寻址方式时（详见 3.2 节），R0、R1 和 DPTR 都可以作为间址寄存器。前两个是 8 位地址指针，寻址范围仅为 256 字节，而 DPTR 是 16 位地址指针，寻址范围可达 64KB。这个地址空间除了可安排数据存储器外，其他需要和单片机接口的外设地址也安排在这个地址空间（详见 9.2 节）。

2.3 特殊功能寄存器 SFR

特殊功能寄存器（Special Function Register）主要用于管理片内和片外的功能部件（指定定时器、中断系统及外部扩展的存储器、外围芯片等）。用户通过对 SFR 进行编程操作，即可方便地管理与单片机有关的所有功能部件，并且可方便地完成各种操作和运算。用户通过对这些 SFR 的逐步了解，即可逐渐理解单片机的工作原理，并学会使用它。

2.3.1 80C51 系列的 SFR

80C51 系列的 SFR 在数量与功能上大同小异，在此以 AT89 系列为例说明。AT89S51 有 23 个（AT89S52 有 29 个）特殊功能寄存器 SFR，它们离散地分布在片内数据存储器的高 128 字节地址 80H~FFH 中，但它们是不能作为数据存储器使用的，所以对这些特殊功能寄存器是不能随意写入数字的，特别是功能部件中的控制寄存器，不同的数字使它们具有不同的工作方式。

特殊功能寄存器并未占满 80H~FFH 整个地址空间，对空闲地址的操作是无意义的。若访问到空闲地址，则读出的是随机数。

这些特殊功能寄存器是：

ACC	累加器 A
B	B 寄存器
PSW	程序状态字
SP	堆栈指针
DPTR0	数据指针 0（由 DP0H 和 DP0L 组成）
DPTR1	数据指针 1（由 DP1H 和 DP1L 组成）
P0~P3	端口 0~3
IP	中断优先级
IE	中断允许
TMOD	定时器/计数器方式
TCON	定时器/计数器控制
TH0	定时器/计数器 0（高字节）
TL0	定时器/计数器 0（低字节）
TH1	定时器/计数器 1（高字节）
TL1	定时器/计数器 1（低字节）
TH2*	定时器/计数器 2（高字节）
TL2*	定时器/计数器 2（低字节）
T2CON*	定时器/计数器 2 控制
T2MOD*	定时器/计数器 2 方式
RCAP2H*	定时器/计数器 2 捕获寄存器高字节
RCAP2L*	定时器/计数器 2 捕获寄存器低字节
SCON	串行控制
SBUF	串行数据缓冲器

PCON 电源控制
WDTRST 看门狗复位寄存器
AUXR 辅助寄存器
AUXR1 辅助寄存器 1

*表示仅 AT89S52 有。

2.3.2 SFR 地址分布及寻址

AT89S51/S52 的 SFR 地址分布见表 2-5。访问这些专用寄存器，仅允许使用直接寻址的方式（详见第 3 章）。对于 AT89S52 单片机，其片内 RAM 的 80H~FFH 地址上有 2 个物理空间，一个是 SFR 的物理空间，另一个是扩展的高 128 字节的数据存储器物理空间，它们所用的地址单元相同，并通过不同的寻址方式区分这 2 个空间。

表 2-5 AT89S51/S52 特殊功能寄存器地址表

SFR		位地址/位定义							
名称	字节地址	7	6	5	4	3	2	1	0
ACC	E0H	E7	E6	E5	E4	E3	E2	E1	E0
		ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
B	F0H	F7	F6	F5	F4	F3	F2	F1	F0
		B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
PSW	D0H	D7	D6	D5	D4	D3	D2	D1	D0
		CY	AC	F0	RS1	RS0	OV	—	P
IP	B8H	BF	BE	BD	BC	BB	BA	B9	B8
		/	/	/	PS	PT1	PX1	PT0	PX0
P3	B0H	B7	B6	B5	B4	B3	B2	B1	B0
		P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
IE	A8H	AF	AE	AD	AC	AB	AA	A9	A8
		EA	/	/	ES	ET1	EX1	ET0	EX0
P2	A0H	A7	A6	A5	A4	A3	A2	A1	A0
		P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
SBUF	(99H)								
SCON	98H	9F	9E	9D	9C	9B	9A	99	98
		SM0	SM1	SM2	REN	TB8	RB8	TI	RI
P1	90H	97	96	95	94	93	92	91	90
		P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
WDTRST+	A6H								
TH2*	(CDH)								
TL2*	(CCH)								
RCAP2H*	CBH								
RCAP2L*	CAH								
T2CON*	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
T2MOD*	C9H							DCEN	T2OE

续表

SFR		位地址/位定义							
名称	字节地址	7	6	5	4	3	2	1	0
AUXR+	(8EH)				WDIDLE	DISET0			DISALE
AUXR1+	(A2H)								DPS
TH1	(8DH)								
TH0	(8CH)								
TL1	(8BH)								
TL0	(8AH)								
TMOD	(89H)	GATE	C/ \bar{T}	M1	M0	GATE	C/ \bar{T}	M1	M0
TCON	88H	8F	8E	8D	8C	8B	8A	89	88
		TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
PCON	(87H)	SMOD	/	/	/	GF1	GF0	PD	IDL
DPIH+	(85H)								
DPIL+	(84H)								
DP0H	(83H)								
DP0L	(82H)								
SP	(81H)								
P0	80H	87	86	85	84	83	82	81	80
		P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

注：+ 表示 89C51/C52 单片机没有，*表示仅 AT89S52 有。

这 23/29 个专用寄存器都可以用字节寻址，其中有 12/14 个专用寄存器还具有位寻址能力，它们的字节地址正好能被 8 整除。

在此要特别提醒读者注意的一个问题是由于仅 AT89S52 单片机才有的定时器 2 和其内部相关的寄存器都只能用直接地址寻址，因为汇编指令不能识别它们的符号，例如不能识别 TH2 等。

2.3.3 SFR 的功能与作用

可以说 SFR 是单片机的核心，单片机的工作是由其统一控制和管理的，理解和学会应用 SFR 就基本掌握了单片机的应用。本节将介绍部分通用特殊功能寄存器的功能与作用，其余与单片机功能部件有关的特殊功能寄存器将在后面有关章节中陆续介绍。

1. 程序状态字寄存器 PSW

PSW 是用于反映程序运行状态的 8 位寄存器，当 CPU 进行各种逻辑操作或算术运算时，为反映操作或运算结果的状态，把相应的标志位置 1 或清 0。这些标志位的状态，可由专门的指令来测试，也可通过指令来读出。它为计算机确定程序的下一步运行方向提供依据。

其格式如下：

PSW 位地址	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
字节地址 D0H	CY	AC	F0	RS1	RS0	OV	—	P

下面说明各标志位的作用。

P：奇偶标志。该位表示累加器 A 内容的奇偶性。在 80C51 的指令系统中，凡是改变累加器 A 中内容的指令均影响奇偶标志位 P。

当 P=1，表示有奇数个“1”；

当 P=0，表示有偶数个“1”。

DIH：由用户置位或复位，在汇编语言中没有给该位定义位名称。

OV：溢出标志。该位表示在进行算术运算时，是否发生了溢出。

在有符号数进行加、减运算时：

当 OV=1，表示运算结果发生了溢出；

当 OV=0，表示运算结果没有溢出。

当把 1 字节看做有符号数时，如果用最高位表示正、负号，则只有 7 位有效位，能表示 -128~+127 之间的数；如果运算结果超出了这个数值范围，就会发生溢出，此时，OV=1，否则 OV=0。例如，两个正数（116、97）相加超过 +127 范围，使其符号由正变负，由于溢出得负数，结果是错误的，这时 OV=1；两个负数（-66、-105）相加，和小于 -128，由于溢出得正数，OV=1。

例：

$$\begin{array}{rcl} & 01110100 & (+116) \\ +) & 01100001 & (+97) \\ \hline & 011010101 & (\text{结果为负数}) \end{array} \qquad \begin{array}{rcl} & 10111110 & (-66) \\ +) & 10010111 & (-105) \\ \hline & 101010101 & (\text{结果为正数}) \end{array}$$

在执行乘法指令后，OV=0 表示乘积没有超过 255，乘积就在 A 中；OV=1 表示乘积超过 255，此时积的高 8 位在 B 中，低 8 位在 A 中。

在执行除法指令后，OV=0 表示除数不为 0，OV=1 表示除数为 0。

RS0、RS1：工作寄存器组选择位。这两位用以选择当前所用的工作寄存器组。用户用软件改变 RS0 和 RS1 的组合，可以选择当前选用的工作寄存器组，其组合关系见表 2-6。

表 2-6 RS0、RS1 对工作寄存器组的选择

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

单片机在复位后，RS0=RS1=0，CPU 自然选中第 0 组为当前工作寄存器组。根据需要，用户可利用传送指令或位操作指令来改变其状态，这样的设置便于在程序中快速保护现场。

F0：用户标志位，由用户置位或复位。

AC：半进位标志。该位表示当进行加法或减法运算时，低半字节向高半字节是否有进位或借位。

当 AC=1，表示低半字节向高半字节有进位或借位；

当 AC=0，表示低半字节向高半字节没有进位或借位。

CY：进位标志，该位（在指令中用 C 表示）表示当进行加法或减法运算时，操作结果最高位（位 7）是否有进位或有借位。

当 CY=1，表示操作结果最高位（位 7）有进位或借位；
当 CY=0，表示操作结果最高位（位 7）没有进位或借位。
在进行位操作时，CY 又作为位操作累加器 C。

2. 累加器 ACC

ACC 是 8 位寄存器，通过暂寄存器与 ALU 相连。它是 CPU 中工作最繁忙的寄存器，因为在进行算术、逻辑类操作时，运算器的一个输入多为 ACC 的输出，而运算器的输出即运算结果也大多要送到 ACC 中运算。在指令系统中累加器的助记符为 A，以下将简称 ACC 为 A。

3. 双数据指针寄存器 DPTR0/1

为方便对 16 位地址的片内、片外存储器和外部扩展的 I/O 器件的访问，在 AT89S51/S52 中有 2 个 16 位的数据指针寄存器（80C51 系列多数型号为 1 个）DPTR0 和 DPTR1。它们主要用以存放外接的数据存储器和 I/O 接口电路的 16 位地址，作为间址寄存器使用。它们也可拆成高字节 DPH 和低字节 DPL 两个独立的 8 位寄存器，占据地址分别为 82H~85H（见表 2-6）。

在 80C51 的指令系统中数据指针只有 DPTR 一种表示方法，通过辅助寄存器 1（AUXR1）的 DPS 位选择 DPTR0 或 DPTR1。

当 DPS=0 时，选择 DPTR0 指针；当 DPS=1 时，选择 DPTR1 指针。用户在访问各自的数据指针寄存器之前，应该将 DPS 位初始化为适当的值，其默认的数据指针寄存器是 DPTR0。

辅助寄存器 1（AUXR1）的地址=A2H，复位值为 xxxxxx0B，各位格式如下：

AUXR1 位地址	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
字节地址 A2H	-	-	-	-	-	-	-	DPS

辅助寄存器 1（AUXR1）的 DPS 选择位作用如下：

当 DPS=0，选择 DPTR 寄存器的高、低字节为 DP0 H、DP0 L；

当 DPS=1，选择 DPTR 寄存器的高、低字节为 DP1 H、DP1 L（应用举例见第 3 章）。

4. B 寄存器

B 寄存器可以作为一般的寄存器使用，在乘、除法运算中用来暂存其中的一个数据。乘法指令的两个操作数分别取自 A 和 B，结果的高字节存于 B 中，低字节存于 A 中。除法指令中被除数取自 A，除数取自 B，结果商存于 A 中，余数存放在 B 中。

在其他指令中，B 寄存器可作为 RAM 中的一个单元来使用。B 寄存器的地址为 B0H。

5. 堆栈指针 SP（Stack Pointer）

堆栈指针 SP 是一个 8 位的特殊功能寄存器，在这个寄存器中始终存放着堆栈栈顶的地址。每存入或取出 1 字节数据，SP 就自动加 1 或减 1。SP 始终指向新的栈顶。

（1）堆栈的概念

堆栈是在单片机内部数据存储器中专门开辟的一个特殊的存储区，主要功能是暂时存放数据和地址，通常用来保护断点和现场（详见 8.3 节）。它的特点是按照“先进后出”的原则存取数据。这里的“进与出”是指进栈与出栈操作，也称为压入和弹出。如图 2-6 所示（图

中均为十六进制数), 第一个进栈的数据所在的存储单元称为栈底, 然后逐次进栈, 最后进栈的数据所在的存储单元称为栈顶, 随着存放数据的增减, 栈顶是变化的, 从栈中取数, 总是先取栈顶的数据, 即最后进栈的数据最先取出。在图 2-6 (a) 中, 堆栈的栈底为 60H, 堆栈指针 SP 的内容为 6BH, 即它的栈顶为 6BH, 栈顶中的内容为 98H。在图 2-6 (b) 中, 向堆栈中压入 1 个数 D0H 后, SP 的内容为 6CH。在图 2-6 (c) 中, 从堆栈中连续取 2 个数, 即将连续取出 D0H 和 98H 后, SP 的内容为 6AH。此时, 栈顶的数为 40H, 而最先进栈的数据最后取出, 即图 2-6 中 60H 中的 57H 最后取出。

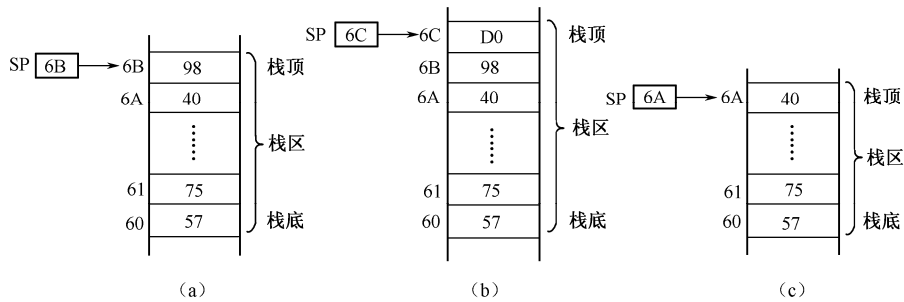


图 2-6 堆栈和堆栈指针示意图

(2) 堆栈的操作

堆栈的操作有两种方式。一种是指令方式, 即使用堆栈操作指令进行“进/出”栈操作。用户可根据其需要使用堆栈操作指令对现场进行保护和恢复。另一种是自动方式, 即在调用子程序或产生中断 (见 8.3 节) 时, 返回地址 (断点) 自动进栈。程序返回时, 断点地址再自动弹回 PC。这种堆栈操作不需用户干预, 是通过硬件自动实现的。

(3) 堆栈的设置

在 80C51 单片机中通常指定内部数据存储器 08~7FH (AT89C52/S52 可到 FFH) 中的一部分作为堆栈。

在使用堆栈前, 一般要先给它赋值, 规定堆栈的起始位置, 即栈底。系统复位后, SP 初始化为 07H, 使得堆栈事实上由 08H 开始。因为 08H~1FH 单元为工作寄存器区 1~3, 20H~2FH 为位寻址区, 在程序设计中很可能要用到这些区, 所以用户在编程时最好把 SP 初值设为 2FH 或更大值, 当然同时还要顾及允许的堆栈深度。在使用堆栈时要注意, 由于堆栈的占用, 会减少内部 RAM 的可利用单元, 如果设置不当, 可能引起内部 RAM 单元冲突, 特别是在使用中断功能时, 更要注意正确地设置 SP 值。

6. 端口 P0~P3

特殊功能寄存器 P0~P3 分别是 I/O 端口 P0~P3 的锁存器。在 AT89S51/S52 单片机中, 是把 I/O 端口当做一般的专用寄存器来使用的, 没有专门设端口操作指令, 使用方便。当 I/O 端口某一位用于输入信号时, 对应的锁存器必须先置“1”, 原因详见 2.5 节。

单片机进入复位状态后, 除 SP 为 07H, P0~P3 为 FFH 外, 其余均为 0。

2.4 单片机的工作原理

学习单片机并不需要十分详细地了解其内部结构中的具体线路, 但为了便于对后面章节

的学习和理解，需要比较清楚地理解单片机的工作原理。单片机是通过执行程序工作的，执行不同的程序就能完成不同的任务，因此单片机执行程序的过程实际上也体现了单片机的工作原理。

2.4.1 指令与程序概述

指令是规定计算机执行某种操作的命令，CPU 就是根据指令来指挥和控制计算机各部分协调地动作，完成规定的操作。指令是由二进制代码表示的，通常指令包括操作码和操作数两部分，操作码规定操作的类型，操作数给出参加操作的数据或存放数据的地址（只有少数指令是没有操作数的）。计算机全部指令的集合称为指令系统，指令系统的性能与计算机硬件密切相关，不同的计算机指令系统不完全相同。程序是根据任务要求有序地编排指令的集合，程序的编制称为程序设计。为了运行和管理计算机所编制的各种程序的总和称为系统软件，一般单片机中没有系统软件，而只能装载用户自己编制的应用软件。

2.4.2 CPU 的工作原理

在执行程序中起关键作用的是 CPU，所以首先介绍 CPU 的工作原理。

CPU 主要由运算器和控制器这两大部分组成。控制器根据指令码产生控制信号，使运算器、存储器、输入/输出端口之间能自动协调地工作；运算器进行算术、逻辑运算以及位操作处理等。

1. 控制器

控制器是用来统一指挥和控制计算机工作的部件，它的功能是接收来自存储器中的逐条指令，进行指令译码，并通过定时和控制电路，在规定的时刻发出各种操作所需的全部内部控制信息及 CPU 外部所需的控制信号，使各部分协调工作，完成指令所规定的各种操作。它由指令部件、时序部件和操作控制部件等三部分组成。

(1) 指令部件

指令部件是一种能对指令进行分析、处理和产生控制信号的逻辑部件，也是控制器的核心。通常，由程序计数器 PC、指令寄存器、指令译码器等组成。

下面介绍与 CPU 工作有关的几个部件，注意这几个寄存器用户都不能直接访问，介绍的目的是便于对单片机工作原理的了解和对指令的学习。

- 程序计数器 PC (Program Counter) 程序计数器 PC 是用于存放和指示下一条要执行指令的地址寄存器。它是一个 16 位专用寄存器，由 2 个 8 位寄存器 PCH（存放地址的高 8 位）和 PCL（存放地址的低 8 位）组成。它有自动加 1 的功能，当一条指令（确切地说是指令字节）按照 PC 所指的地址从存储器中取出之后，PC 就会自动加 1。由于在单片机中取指令的操作是以字节为单位（在 80C51 系列单片机中的指令长度一般为 1~3 字节），因而 PC 在自动加至该指令字节个数后，才指向下一条将要执行的指令地址。所以 PC 是维持单片机有秩序地执行程序的关键性寄存器。计算机执行程序的过程是顺序地把存储器内的指令依次取到 CPU 里进行识别，然后去执行规定的操作，要取的指令地址码就是由程序计数器提供的，PC 可保证程序顺序执行。

如果要求不按顺序执行指令，例如，想要跳过一段程序，再执行指令，则可通过执行一条跳转指令或调用指令，将要执行的指令地址送入程序计数器，取代原有指令地址，这样才可实现程序的跳转或调用。

- 指令寄存器 指令寄存器是 8 位寄存器，用于暂时存放指令，等待译码。
- 指令译码器 指令译码器用于对送入指令译码器中的指令进行译码。所谓译码就是把指令转变成执行此指令所需要的电信号，当指令送入译码器后，由译码器对该指令进行译码，根据译码器输出的信号，CPU 控制电路定时地产生执行该指令所需的各种控制信号，使计算机正确执行程序所要求的各种操作。

(2) 时序部件

时序部件由时钟电路和脉冲分配器组成，用于产生操作控制部件所需的时序信号，产生时序信号的部件称为时序发生器或时序系统，它由一个振荡器和一组计数分频器组成。振荡器是一个脉冲源，输出频率稳定的脉冲，也称为时钟脉冲，为 CPU 提供时钟基准。时钟脉冲经过进一步的计数分频，产生所需要的节拍信号或机器周期信号。

(3) 操作控制部件

操作控制部件可以与时序电路相配合，把译码器输出的信号转变为执行该指令所需的各种微命令，以完成规定的操作，该部件也可以处理外部输入的信号（如复位、中断源等）。

2. 运算器

运算器是对数据进行算术运算和逻辑操作的执行部件，包括算术/逻辑部件 ALU、累加器 ACC (Accumulator)、暂存寄存器、程序状态字寄存器 PSW (Program Status Word)、BCD 码运算调整电路等。为了提高数据处理和位操作功能，片内增加了一个通用寄存器区和一些专用寄存器，而且还增加了位处理逻辑电路的功能。

(1) 算术/逻辑部件 ALU

ALU (Arithmetic Logic Unit) 是对数据进行算术运算和逻辑操作的执行部件，由加法器和其他逻辑电路（移位电路和判断电路等）组成。在控制信号的作用下，它能完成算术加、减、乘、除，逻辑“与”、“或”、“异或”等运算以及循环移位操作、位操作等功能。ALU 的运算结果将通过数据总线送到累加器 A，同时影响程序状态标志寄存器的有关标志位。

(2) 暂存器

用以暂存进入算术/逻辑部件 ALU 之前的数据。暂存器 1 和 2 均可暂存来自寄存器、立即数、直接寻址单元及内部 RAM 的数，暂存器 2 还可以暂存累加器 A 的数。暂存器不能通过编程访问，设置暂存器的目的是暂时存放某些中间过程所产生的信息，以避免破坏通用寄存器的内容。

运算器的其他部分已经在其他章节详细介绍。

CPU 正是通过对这几部分的控制与管理，使单片机完成指定的任务。

2.4.3 单片机执行程序过程

单片机的工作过程实质就是执行程序的过程，即逐条执行指令的过程。计算机每执行一条指令都可分为三个阶段进行，即取指令，译码分析指令和执行指令。

取指令阶段的任务是：根据程序计数器 PC 中的值，从程序存储器读出当前要执行的指

译码分析指令阶段的任务是：将指令寄存器中的指令操作码取出后进行译码，分析指令要求实现的操作性质，如是执行传送还是加减等操作。

大多数 8 位单片机取指、译码和执行指令这 3 步是串行顺序进行的，32 位单片机的这 3 步也是不能缺少的，但 32 位单片机是采用预取指的流水线方法操作，32 位单片机采用精简指令集，均为单周期指令，其允许指令重叠并行操作。例如在第一条指令取出后，开始译码的同时，就取第二条指令；在第一条指令开始执行，第二条指令开始译码的同时，就取第三条指令，如此循环，从而 CPU 在同一时间可以对不同指令进行不同的操作。这样就实现了不同指令的并行处理，显然这种方法大大加快了指令的执行速度。

为便于了解程序的执行过程，在这里给出单片机执行一条指令过程的示意图，如图 2-7 所示。该图是在图 2-2 的基础上予以简化和提炼，突出了指令运行过程，图中大部分功能前面已说明，在此不赘述。另外，图 2-2 中的输入输出、定时器等部分因与待介绍指令执行无关，在此没有画出。



• 38 •

下面通过一条指令的执行过程，简要说明单片机的工作过程。

现在假设准备执行的指令是“MOV A, 32H”，这条指令的作用是把片内 RAM32H 中的内容 FFH 送入累加器 A 中，这条指令的机器码（计算机能识别的数字）是“E5H, 32H”。这条指令存放在程序存储器的 0031H, 0032H 单元。存放形式示意如图 2-7。复位后单片机在时序电路作用下自动进入执行程序过程。执行过程实际上就是单片机取指令（取出存储器中事先存放的指令阶段）和执行指令（分析执行指令阶段）的循环过程。

为便于说明，现在假设程序已经执行到 0031H，即 PC 变成 0031H。在 0031H 中已存放 E5H，0032H 中已存放 32H。当单片机执行到 0031H 时，首先是进入取指令阶段。其执行过程如下：

- (1) 程序计数器的内容（这时是 0031H）送到地址寄存器。
- (2) 地址寄存器中内容（0031H）通过地址译码电路，使地址为 0031H 的单元被选中。
- (3) CPU 使读控制线有效。
- (4) 在读命令控制下被选中存储器单元的内容（此时应为 E5H）送到内部数据总线上，因为是取指令阶段，该内容通过数据总线被送到指令寄存器。
- (5) 程序计数器的内容自动加 1（变为 0032H）。

至此，取指令阶段完成，进入译码分析和执行指令阶段。

由于本次进入指令寄存器中的内容是 E5H（操作码），经译码器译码后单片机就会知道该指令是要把一个数送到 A 累加器中，而该数是在片内 RAM 的 32H 存储单元中。所以执行该指令还必须把数据（FFH）从存储器中取出送到 A。即还要到 RAM 中取第二个字节，其过程与取指令阶段很相似，只是此时 PC 已为 0032H，指令译码器结合时序部件，产生 E5H 操作码的微操作系列，使数据 FFH 从 RAM 的 32H 单元取出。因为指令是要求把取得的数送到 A 累加器中，所以取出的数据经内部数据总线进入 A 累加器，而不进入指令寄存器。至此，一条指令执行完毕。PC 寄存器在 CPU 每次向存储器取指令或取数时都自动加 1，此时 PC=0033H，单片机又进入下一个取指令阶段。这一过程一直重复下去，直到收到暂停指令或循环等待指令才暂停。CPU 就是这样一条一条执行指令，完成指令所规定的功能，这就是单片机的基本工作原理。

不同指令的类型、功能是不同的，因而其执行的具体步骤和涉及的硬件部分也不完全相同，但它们执行指令的 3 个阶段是相同的，篇幅关系不能一一列举。读者通过对指令系统的学习，将能逐渐了解每条指令的作用及执行过程。

后续章节将详细介绍 AT89S51/S52 各部分的工作原理及应用，读者通过对这些内容的学习，将能更透彻地理解单片机的工作原理。

2.5 输入/输出端口结构

AT89S51/S52 单片机共有 4 个并行输入/输出端口，简称 I/O 端口，这 4 个端口的名称为 P0~P3。I/O 端口是单片机与外界联系的重要通道，由于在数据的传输过程中，CPU 需要对接口电路中输入/输出数据的寄存器进行读写操作，所以在单片机中对这些寄存器像对存储单元一样进行编址。通常把接口电路中这些已编址并能进行读、写操作的寄存器称为端口（PORT），或简称口。

通过对 I/O 端口结构的学习，可以深入理解 I/O 端口的工作原理，以便正确地设计单片

机的外围逻辑电路，学会正确合理地使用端口。

2.5.1 4 个 I/O 端口的主要异同点

AT89S51/S52 单片机这 4 个 I/O 端口在结构和特性上是基本相同的，但也有一定差别，因此它们的负载能力和接口要求有相同之处，但又各具特点，在使用上也有一定差别。掌握它们的主要异同点便于学习和记忆。

1. 主要相同点

- 都是 8 位双向口，在无片外扩展存储器的系统中，这 4 个端口的每一位都可以作为双向通用 I/O 端口使用。
- 每个端口都包括锁存器（即专用寄存器 P0~P3）、输出驱动器端口引脚和输入缓冲器。
- 在作为一般的输入时，都必须先向锁存器写入“1”，使驱动管 FET 截止。
- 系统复位时，4 个端口锁存器全为“1”。如果程序执行后没有改变过 I/O 口的状态，则作为输入时不必再写“1”。
- 4 个端口均可以按字访问，也可以按位访问。

2. 主要不同点

- P0 口是一个真正的双向口，它的每一位都具有输出锁存、输入缓冲和悬浮状态（即高阻态），这 3 种工作状态。
- P1~P3 口被称为准双向口。它的每一位都具有输出锁存，输入缓冲 2 种工作状态。
- P0 口的每一位可驱动 8 个 LSTTL 负载。P0 口在低电平状态下每一位的最大灌入电流为 3.2mA，1 个标准 LSTTL 门电路的最大灌入电流为 0.4mA。所以 P0 口可以驱动 8 个 LSTTL 门电路。
- P1~P3 口每位可驱动 4 个 LSTTL 负载，每一位的最大灌入电流为 1.6mA。
- P0 口既可做 I/O 端口使用，也可做地址/数据总线使用。作为通用口输出时，输出级是开漏电路，在驱动 NMOS 或其他拉电流负载时，只有外接上拉电阻，才有高电平输出；作地址/数据总线时，无须外接电阻，此时不能再做 I/O 口使用。
- P1 口除作为一般的 I/O 口外，某些位还增加了第二功能，见表 2-1。
- P2 口除作为一般的 I/O 口外，在具有片外并行扩展存储器的系统中，P2 口通常作为高 8 位地址线，P0 口分时作为低 8 位地址线和双向数据总线。
- P3 口除作为一般的 I/O 口外，其各位均增加了第二功能，见表 2-2。

下面分别介绍各端口结构，通过对其内部结构的了解，可以进一步理解这 4 个端口的主要异同点。

2.5.2 P0 口

P0 口是一个标准的双向 8 位并行口，既可以用作通用 I/O 口，也可以用作地址/数据线。由特殊功能寄存器 P0 管理 P0 口各位的工作状态，其地址为 80H，各位地址为 80H~87H。

在访问片外存储器时，它分时提供低 8 位地址和 8 位数据，故这些 I/O 线有“地址/数据

总线”之称，简写为 AD0~AD7。在不作总线时，它也可以作为普通 I/O 口使用。

1. P0 口的位电路结构

P0 口各位的结构完全相同，但又相互独立。图 2-8 所示为 P0 口某位 P0.n ($n=0\sim7$) 的结构图，它由 1 个输出锁存器、2 个三态输入缓冲器和 2 个输出驱动场效应管 FET (Field-Effect-Transistor) 以及控制电路等组成。其输出级在结构上的主要特点是无内部上拉电阻。

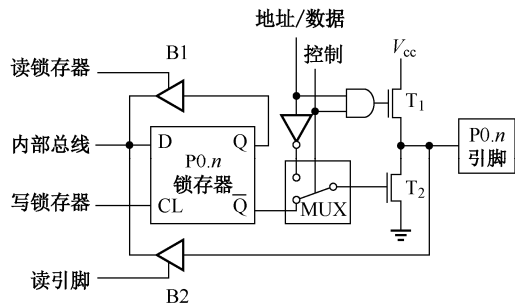


图 2-8 P0 口某位结构

输出锁存器用于锁存输出数据，2 个三态输入缓冲器 B1 和 B2 分别用于对锁存器和引脚输入数据进行缓冲。在 P0 口的电路中有一个多路转换开关 MUX，它的一个输入为锁存器，另一个输入为地址（低 8 位）/数据线的反相输出。在控制信号的作用下，多路开关 MUX 可以分别接通锁存器输出和地址/数据线输出。2 个输出驱动场效应管 T1 和 T2 用于驱动输出的数据。

2. P0 口的工作原理

下面按照其不同的功能分别介绍。

(1) P0 口用作一般 I/O 口

当 P0 作为一般 I/O 口使用时，通过执行传送指令（详见第 3 章），则 CPU 内部发控制电平“0”，封锁“与”门，将输出上拉场效应管 T1 截止；同时控制多路开关 MUX，将锁存器 Q 与输出驱动场效应管 T2 的栅极接通。此时，其输出与输入的工作原理分别如下：

当 P0 用作输出口时，显然内部总线与 P0 端口同相位，写脉冲加在 D 触发器 CL 上，内部总线就会向端口引脚输出数据。由于输出驱动级是漏极开路电路（称“开漏电路”），若驱动 NMOS 或其他拉电流负载，则需要外接上拉电阻（阻值一般为 5~10k Ω ）。

当 P0 用作输入口时，分为读端口（即读锁存器）和读引脚两种输入方式，因此，端口中设有 2 个三态输入缓冲器用于读操作。通过 CPU 发出不同指令，实现不同的输入方式。

读引脚时，需要先向对应的锁存器写入“1”，以使 FET 截止。通过执行传送指令，则读脉冲把三态缓冲器 B2 打开，端口引脚上的数据经过缓冲器读入到内部总线。同样，P1~P3 口在进行读操作时，也需要先向对应的锁存器写入“1”。

读端口时，通过执行对端口的“读一改一写”指令（详见第 3 章），则 CPU 首先通过图 2-8 的缓冲器 B1 读锁存器 Q 端的数据，读入后进行运算修改，再写入锁存器，该数据通过缓冲器 B1 进入内部总线。这种方法可以避免因引脚外部电路的原因而使引脚状态变化引起误读。

AT89S51/S52 的 4 个端口 P0~P3 都采用了具有两套输入缓冲器的电路结构。通过执行不同指令区分输入方式是读端口还是读引脚，该操作过程是 CPU 自动进行的，用户不必考虑。

(2) P0 口用作地址/数据总线

在扩展系统中（详见第 9 章），P0 口作为地址/数据总线使用，可分为以下两种情况。

一种是由 P0 引脚输出地址/数据信息。CPU 内部发控制电平“1”，打开“与”门，同时用多路开关 MUX 把低 8 位地址/数据线的某一位通过反相器与驱动场效应管 T2 栅极接通。

从图 2-8 可以看到，上、下 2 个 FET 处于反相，从而构成了推拉式的输出电路。推拉式电路驱动能力较大，因而 P0 的输出级驱动能力比 P1~P3 口大。当地址/数据状态为“1”时，T1 导通，T2 截止，T1 相当于一个较大的电阻，从而使加在 T1 上的 V_{CC} 与外接电路形成回路，输出为高电平“1”；当地址/数据状态为“0”时，T2 导通，T1 截止，T2 相当于一个较大的电阻，从而使 T2 经过地与外接电路形成回路，输出为低电平“0”。

另一种情况是由 P0 输入数据。此时，通过指令 CPU 将自动对该口写入“1”，以使 FET 截止，此时输入信号从引脚通过输入缓冲器 B2 进入内部总线。

2.5.3 P1 口

P1 口是一个准双向的 8 位并行口，主要作为通用 I/O 口使用。由特殊功能寄存器 P1 管理 P1 口各位的工作状态，其地址为 90H，各位地址为 90H~97H。

1. P1 口的位电路结构

P1 口各位的电路结构如图 2-9 所示。其主要部分与 P0 口相同，但输出驱动部分与 P0 口不同，其内有与电源相连的上拉负载电阻。

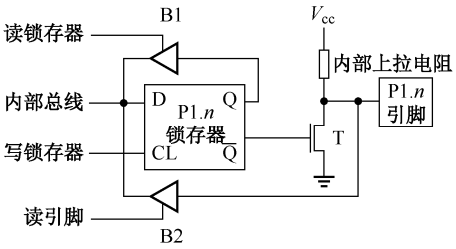


图 2-9 P1 口某位结构

2. P1 口的工作原理

下面按照其不同的功能分别介绍。

(1) P1 口用作输出

P1 口用作输出时，因其内部已经有上拉电阻，故可不外接上拉电阻。当 CPU 输出 1 时， $Q=1$ ， $\bar{Q}=0$ ，使 FET 截止，此时，P1 口引脚输出为 1。当 CPU 输出 0 时， $Q=0$ ， $\bar{Q}=1$ ，使 FET 导通，此时，P1 口引脚输出为 0。

(2) P1 口用作输入

P1 口用作输入时，与 P0 口相同也分为读锁存器和读引脚两种输入方式。读锁存器时，锁存器 Q 端的状态通过缓冲器 B1 进入内部总线；读引脚时，需要先向对应的锁存器写入“1”，使 FET 截止，然后引脚的状态通过缓冲器 B2 进入内部总线，由于片内负载电阻较大，约为 20~40k Ω ，所以不会对输入的数据产生影响。

AT89S51/S52 单片机的 P1 口除可以用作一般的 I/O 口外，其中 5 位还具有第二功能，如表 2-1 所列。

2.5.4 P2 口

P2 口是一个准双向的 8 位并行口，既可以作为通用 I/O 口使用，也可以作为高 8 位地址线使用。由特殊功能寄存器 P2 管理 P2 口各位的工作状态，其地址为 A0H，各位地址为 A0H~A7H。

在访问片外存储器或者外围器件时，它输出高 8 位地址，即 A8~A15；在不作总线时，

也可以作为普通 I/O 口使用。

1. P2 口的位电路结构

P2 口各位的结构如图 2-10 所示。从该图中可看到，P2 口的位结构比 P1 口多了一个转换控制部分，其余部分相同。

当 P2 口用作通用 I/O 口时，多路开关 MUX 倒向锁存器输出 Q 端，构成输出驱动电路，此时用法与 P1 口相同；当 P2 口用作高 8 位地址线时，多路开关 MUX 与内部高 8 位地址线的某一位相接。

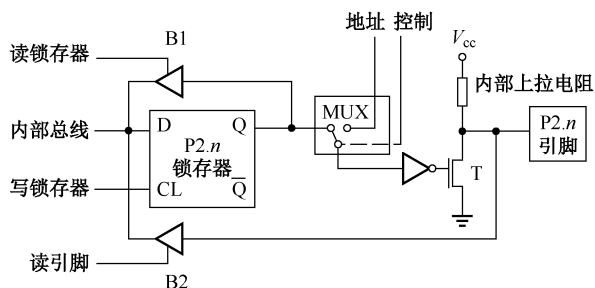


图 2-10 P2 口某位结构

2. P2 口的工作原理

下面按照其不同的功能分别介绍。

(1) P2 口用作高 8 位地址线

在系统扩展片外存储器时，高 8 位地址由 P2 口输出（低 8 位地址由 P0 口输出）。此时，MUX 在 CPU 的控制下与内部地址线相接。当地址线为“0”时，FET 导通，P2 口的引脚输出“0”；当地址线为“1”时，FET 截止，P2 口的引脚输出“1”。由于访问片外存储器的操作往往接连不断，P2 口要不断送出高 8 位地址，故此时 P2 口无法再用作通用 I/O 口。

在只需扩展较小容量的片外数据存储器的系统中，使用“MOVX @Ri”类指令访问片外 RAM，寻址范围为 256 字节（也可称为页寻址），则只需低 8 位地址线就可实现。P2 口不受该指令的影响，仍可用作通用 I/O 口。

如果寻址范围大于 256 字节，且小于 64KB，可以用软件方法只利用 P1~P3 中的某几根口线送高位地址，而保留 P2 中的部分或全部口线以用作通用 I/O 口。

若扩展的数据存储器或外部器件容量超过 256 字节，则要使用“MOVX @DPTR”类指令，寻址范围扩展到 64KB，此时，高 8 位地址总线由 P2 口输出。在读/写周期内，P2 口锁存器仍保持原来端口的数据，在访问片外 RAM 周期结束后，多路开关自动切换到锁存器 Q 端。由于 CPU 对 RAM 的访问不是经常的，在这种情况下，P2 口在一定的限度内仍可用作通用 I/O 口。

(2) P2 口用作通用 I/O 口

在内部控制信号作用下，MUX 在 CPU 的控制下与输出锁存器 Q 端相接。

作输出时，CPU 输出 1，Q=1，FET 截止，P2 口的引脚输出 1；CPU 输出 0，Q=0，FET 导通，P2 口的引脚输出 0。

作输入时，MUX 仍然保持与输出锁存器 Q 端相接。其输入情况也分为读锁存器和读引脚两种输入方式，同 P0 口和 P1 口。

2.5.5 P3 口

P3 口是一个多功能的准双向 8 位并行口，它的每一位既可以作为通用 I/O 口使用，又都具有第二输出功能。由特殊功能寄存器 P3 管理 P3 口各位的工作状态，其地址为 B0H，各位

地址为 B0H~B7H。

1. P3 口的位电路结构

P3 口各位的结构如图 2-11 所示。P3 口也是多功能端口，与 P1 口结构相比，多了一个“与非”门和缓冲器 B3。“与非”门的作用相当于一个开关。

2. P3 口的工作原理

下面按照其不同的功能分别介绍。

(1) P3 口用作通用 I/O 口

当执行对 P3 口的输出指令时，第二输出功能端自动置为高电平，则锁存器输出可通过“与非”门送至 T，其状态决定了引脚端输出电平。

输入时，也分为读锁存器和读引脚两种输入方式，同其他口。当 CPU 发出读命令时，使左边缓冲器 B2 上的“读引脚”有效，右边的缓冲器 B3 是长开的，于是引脚信号读入 CPU。

(2) P3 口用作第二功能引脚

P3 口的 8 个引脚均具有专门的第二功能，如表 2-2 所列。当执行与第二功能有关的输出操作时，锁存器输出 Q 为 1，则端口用于第二功能，第二输出功能端为输出时，信号通过“与非”门和 FET 送至端口引脚，此时，当第二输出功能端为 1 时，FET 截止，P3 口引脚为 1；当第二输出功能端为 0 时，FET 导通，P3 口引脚为 0，从而实现第二功能信号输出。

当执行与第二功能有关的输入操作时，该位的锁存器和第二输出功能端均置 1，FET 保持截止，端口引脚的第二功能信号通过右边的缓冲器 B3 送到第二输入功能端。

以上各引脚的功能与作用，只有在后面章节的学习中才能逐渐加深理解并学会如何应用。

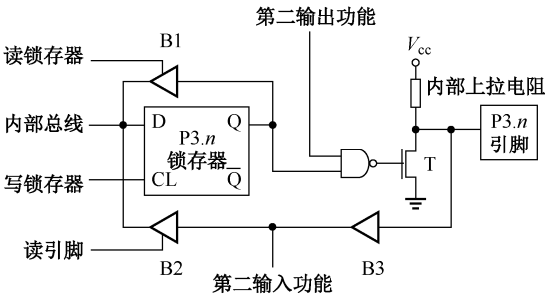


图 2-11 P3 口某位结构

2.6 时序及时钟电路

时序就是 CPU 在执行指令时各控制信号之间的时间顺序关系。为了保证各部件间协调一致地同步工作，单片机内部的电路应在惟一的时钟信号控制下严格地按时序进行工作。下面介绍有关电路及 CPU 时序的概念。

2.6.1 CPU 时序及有关概念

中央处理器 CPU 执行指令的一系列动作都是在统一的时钟脉冲控制下逐拍进行的，这个脉冲是由单片机控制器中的时序电路发出的。由于指令的字节数不同，则取这些指令所需的时间就不同，即使是字节数相同的指令，因为执行操作存在较大差别，故不同指令的执行时间也不一定相同，即所需要的节拍数不同。为了便于对 CPU 时序进行分析，人们按指令的执行过程规定了几种周期，也称为“时序定时单位”，对于 80C51 系列单片机定义了如下 4 种时序单位，这几种时序单位之间的时序关系示意图如图 2-12 所示，图 2-12 选用的是 2 个机器周期的指令。下面分别予以说明。

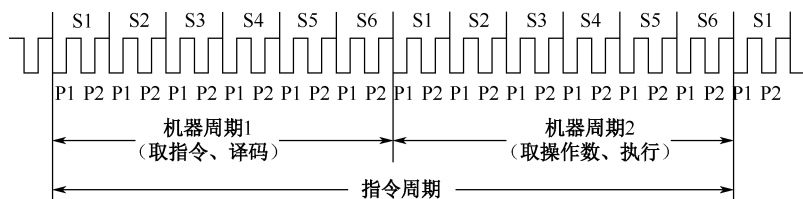


图 2-12 基本定时时序关系示意图

(1) 振荡周期

振荡周期也称为“时钟周期”，定义为时钟脉冲频率的倒数。在 80C51 系列单片机中，1 个振荡周期定义为 1 个节拍，用 P 表示。它是计算机中最基本、最小的时间单位。在一个振荡周期内，CPU 仅完成一个最基本的动作。对于某种单片机，若采用 1MHz 的时钟频率，则时钟周期为 1 μ s；若采用 4MHz 的时钟频率，则时钟周期为 250ns。由于时钟脉冲是计算机的基本工作脉冲，因此它控制着计算机的工作节奏(使计算机的每一步都统一到它的步调上来)。显然，对同一种机型的计算机，时钟频率越高，计算机的工作速度就越快；但是，由于不同的单片机其硬件电路和器件不完全相同，所以其所要求的时钟频率范围也不一定相同，是不能随意提高的，通常频率升高电路工作加快，功耗也会增加。AT89S 系列的时钟频率范围是 0~33MHz。80C51 系列单片机其他型号的时钟频率范围不完全相同，使用时须注意。

(2) 状态周期

80C51 系列单片机中 2 个节拍定义为 1 个状态周期，即图 2-12 中所示的 P1、P2，用 S 表示。由图 2-12 可以看出二者之间的相互关系。

(3) 机器周期

在计算机中，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，每一阶段完成一个基本操作，如取指令、存储器读、存储器写等。完成一个基本操作所需要的时间称为“机器周期”。80C51 的 1 个机器周期包含 6 个状态周期 S，因此，1 个机器周期可依次表示为 S1P1、S1P2、S2P1、S2P2、…、S6P1 和 S6P2 共 12 个振荡周期，如图 2-12 中所示。不同计算机 1 个机器周期所包含的振荡周期不一定相同。

(4) 指令周期

指令周期是指执行一条指令所需要的时间，一般由若干个机器周期组成。指令不同，所需要的机器周期数也不同。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，仅用 1 个机器周期即可；对于一些比较复杂的指令（如转移指令、乘除指令），则需要 2 个或 2 个以上的机器周期。

图 2-12 中还标明了 CPU 取指令和执行指令的时序。通常，包含 1 个机器周期的指令称为“单周期指令”，包含 2 个机器周期的指令称为“双周期指令”。不同计算机的时序关系一般是不完全相同的，比如每个指令周期包含的机器周期数、每个机器周期包含的时钟周期数都可能不同。目前的发展趋势是尽可能都精简为单周期指令，且 1 个振荡周期即 1 个机器周期，这样在相同的运行速度下可大大降低时钟频率，因而在选择单片机时，不仅要看其适用的工作频率范围，还要看它是否为精简的单周期指令集，每条指令的执行时间是否都很短。在采用精简指令的单片机中已经没有“机器周期”、“状态周期”这样的时序单位了。

2.6.2 振荡器和时钟电路

要给 CPU 提供上述时序需要有相关的硬件电路，即振荡器和时钟电路。下面介绍其工作

原理和外部电路的不同接法。

1. 振荡器和时钟电路的工作原理

图 2-13 所示为振荡器和时钟电路的工作原理。80C51 系列单片机内部有一个高增益反相放大器，用于构成振荡器，但要形成时钟，外部还需附加电路。XTAL1 引脚为反相放大器和时钟发生电路的输入端，XTAL2 引脚为反相放大器的输出端。

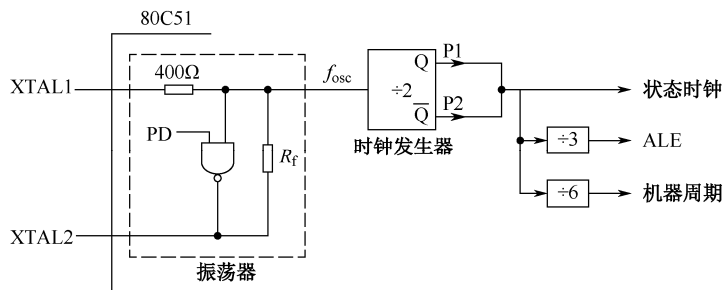


图 2-13 振荡器和时钟电路工作原理

片内时钟发生器实质是个 2 分频的触发器，其输入来自振荡器的 f_{osc} ，输出为 2 相时钟信号，即节拍信号 P1、P2，其频率为 $f_{osc}/2$ 。2 个节拍为 1 个状态时钟 S。状态时钟再 3 分频后为 ALE 信号，其频率为 $f_{osc}/6$ ；状态时钟 6 分频后为机器周期信号，其频率为 $f_{osc}/12$ 。

特殊功能寄存器 PCON(详见 2.8 节)的 PD 位可以控制振荡器的工作。当 PD=1 时，振荡器停止工作，单片机进入低功耗工作状态；复位后，PD=0，振荡器正常工作。

2. 时钟电路接法

不同计算机的时钟电路接法是不完全相同的，80C51 的时钟电路接法有以下两种方式。

(1) 内部时钟方式

通过在引脚 XTAL1 和 XTAL2 两端跨接晶体或陶瓷谐振器，再利用芯片内部的振荡电路，就构成了稳定的自激振荡器，其发出的脉冲直接送入内部时钟电路，如图 2-14 所示。外接晶振时， C_1 和 C_2 值通常选择为 20~30pF；外接陶瓷谐振器时， C_1 和 C_2 为 30~50pF。 C_1 、 C_2 对频率有微调作用，影响振荡的稳定性和起振速度。所采用的晶体或陶瓷谐振器的频率选择 0~24/33MHz（不同型号之间有所差别）。为了减小寄生电容，更好地保证振荡器稳定、可靠地工作，谐振器和电容应尽可能与单片机芯片靠近安装。

(2) 外部时钟方式

此方式是利用外部振荡脉冲接入 XTAL1。对于 AT89S51/S52 单片机，因内部时钟发生器的信号取自反相放大器的输入端，故采用外部时钟源时，接线方式为外时钟信号接至 XTAL1，XTAL2 悬空，如图 2-15 所示。

外部时钟信号经单片机振荡器中的 400Ω 电阻直接进入 2 分频的触发器而成为内部时钟信号，参考图 2-13。要求这个信号高、低电平的持续时间均大于 20ns，一般为频率低于 24MHz 或 33MHz 的方波。当多块芯片同时工作时，这种方式有利于同步。

现在已有某些型号的单片机将振荡器集成到单片机内部，不接外部晶振即可工作，这样进一步简化了单片机的使用。只是目前采用这种方法时钟精度不如采用外部晶体谐振器的方法高，选择时须注意使用场合。

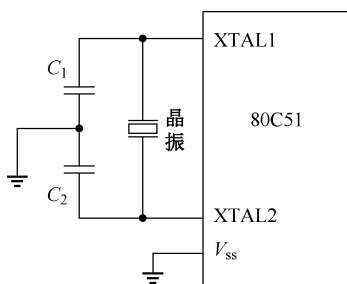


图 2-14 内部时钟方式

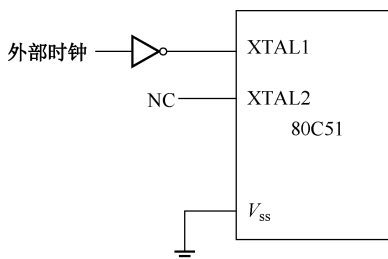


图 2-15 80C51 外时钟源接法

2.6.3 80C51 的指令时序

在 80C51 指令系统中，根据各种操作的繁简程度，其指令可由单字节、双字节和 3 字节组成。从单片机执行指令的速度看，单字节和双字节指令都可能是单周期和双周期，而 3 字节指令都是双周期，只有乘/除法指令占 4 周期。所以不同指令的取指令与操作指令的时序与执行时间是不完全相同的，此外不同指令所用到的控制信号也不完全相同，例如 ALE、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 这几个控制信号只在外扩存储器或者其他外围器件时用到。通过图 9-2 访问片外 RAM 的操作时序可以对时序与控制信号的关系有一定了解，并可以更好地理解 ALE、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 、P0 及 P2 等信号和数据线的作用。

作为一般用户不必了解每条指令的取指令与操作指令的时序，但要知道当振荡周期确定后，每条指令的指令周期所需时间。例如，如果采用 24MHz 晶振，则执行 1 条单周期、双周期和 4 周期指令的时间（指令周期）分别为 0.5 μs 、1 μs 和 2 μs 。在编制软件延时程序或者定时中断程序时需要有这方面的知识。

单片机在工作时的内部时钟信号无法从外部观察，故当采用 XTAL1 和 XTAL2 之间接晶振的方法时，可通过示波器观察 XTAL2 端引脚信号，判断晶振是否已经起振。ALE 引脚可用作内部工作状态指示信号。通过 ALE 端引脚信号，简单判断 CPU 是否已经工作。在此要特别提醒读者注意 ALE 引脚在访问外部存储器时，在一个指令周期中将丢失一个脉冲。

2.7 复位和复位电路

复位是单片机的初始化操作，单片机在启动运行时，都需要先复位，它的作用是使 CPU 和系统中其他部件都处于一个确定的初始状态，并从这个状态开始工作。例如复位后，PC 初始化为 0，于是单片机自动从 0 单元开始执行程序。因而复位是一个很重要的操作方式。80C51 系列单片机本身，一般是不能自动进行复位的，必须配合相应的外部电路才能实现。

2.7.1 内部复位信号的产生

单片机的整个复位电路包括芯片内、外两部分，外部电路产生的复位信号通过复位引脚 RST 进入片内一个施密特触发器（抑制噪声作用）再与片内复位电路相连，80C51 内部复位电路原理图见图 2-16。复位电路每个机器周期对

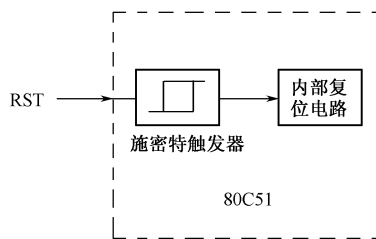


图 2-16 复位电路原理图

施密特触发器的输出采样一次。当 RST 引脚端保持两个机器周期（24 个时钟周期）以上的高电平时，80C51 进入复位状态。

2.7.2 复位状态

复位后片内各专用寄存器的状态如表 2-7 所示，表中 X 为随机数。

表 2-7 复位后的内部专用寄存器状态

寄 存 器	内 容	寄 存 器	内 容
PC	0000H	TMOD	00H
ACC	00H	TCON	00H
B	00H	TH0	00H
PSW	00H	TL0	00H
SP	07H	TH1	00H
DPTR0	0000H	TL1	00H
DPTR1	0000H	TH2 *	00H
P0~P3	FFH	TL2 *	00H
IP	XX000000B	T2MOD*	XXXXXX00B
IE	0X000000B	T2CON*	00H
SCON	00H	RCAP2H*	00H
SBUF	XXXXXXXXB	RCAP2L*	00H
PCON	0XXX0000B	WDTRST	XXXXXXXXXB
AUXR	XXX00XX0B	AUXR1	XXXXXXXX0B

* 只有 AT89S52 有。

复位时，ALE 和 $\overline{\text{PSEN}}$ 成输入状态，即 $\text{ALE}=\overline{\text{PSEN}}=1$ ，片内 RAM 不受复位影响。复位后，P0~P3 口输出高电平且使这些双向口皆处于输入状态，并且将 07H 写入堆栈指针 SP，同时将 PC 和其余 SFR 清为初始状态，此时单片机从起始地址 0000H 开始重新执行程序，所以单片机运行出错或进入死循环时，可使其复位后重新运行。

2.7.3 外部复位电路设计

80C51 系列单片机的外部复位电路有上电自动复位和按键手动复位两种。

上电复位利用电容器充电来实现，上电瞬间，RC 电路充电，RST 引脚端出现正脉冲，只要 RST 引脚端保持 10ms 以上高电平，就能使单片机有效地复位。

按键手动复位又分为按键电平复位和按键脉冲复位。按键电平复位，相当于按复位键后复位端通过电阻与 VCC 电源接通；按键脉冲复位是利用 RC 微分电路产生正脉冲。几种基本复位电路见图 2-17（a）、（b）、（c），参数选取应保证复位高电平持续时间大于两个机器周期（图中参数适宜 12MHz 晶振）。

在实际的应用系统中，有些外围芯片也需要复位，如果这些复位端的复位电平要求与单片机的复位要求一致，则可以与之相连。

复位电路关系到一个系统能否可靠地工作，由电阻、电容元件和门电路组成的复位电路虽然在多数情况下均能良好地工作，但对于电源瞬时跌落的情况，这种电路可能无法保证复

位脉冲的宽度。另外，阻、容复位电路的复位触发门限较难在设计时确定，因为它与电阻、电容的精度，供电电源的精度以及门电路的触发电平有关，且受温度的影响较大。对于要求不高的场合，选用阻、容元件和门电路作为复位电路是一种廉价而简单的选择方案，并且这种电路多数情况下均能正常工作。但对于应用现场干扰大、电压波动大的工作环境，常常要求系统在任何异常情况下都能自动复位恢复工作，这样的系统选用专用复位监控芯片作为系统的复位产生器是最理想的。复位监控芯片在上电、掉电情况下，均能提供正确的复位脉冲，其宽度和触发门限值均是由生产厂家设计并经出厂测试保证的，近年来已陆续出现了多种专用复位监控器。当应用系统中有多个需要复位的器件时，这种芯片能保证可靠地同步复位。随着技术的发展，这类芯片已发展为多功能芯片，即它除了能提供复位脉冲外，还有其他可选的功能。其中比较典型的芯片如 X5043/45。

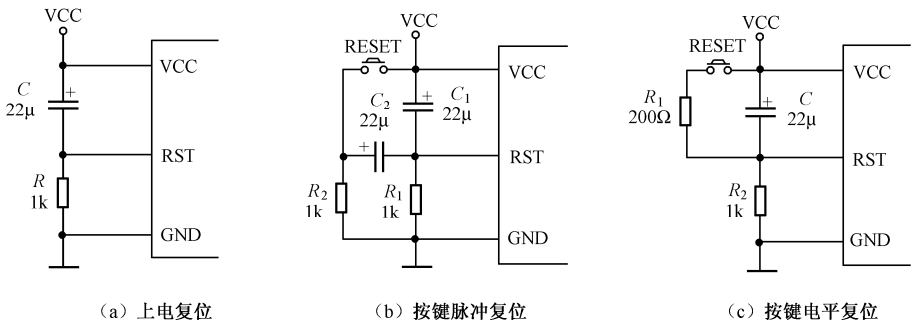


图 2-17 复位电路

由上述可知，给一块内部含有程序存储器的单片机配上时钟电路和复位电路就可构成单片机的最小应用系统。

目前有的系列型号单片机内部已配有复位电路，这样外部就不再需要接复位电路了。另外本身带有看门狗复位电路的单片机在热启动时，外部复位电路不起作用。

2.8 80C51 系列单片机的低功耗方式

为了降低单片机的功耗，减少外界干扰，单片机通常都有可程序控制的低功耗工作方式。低功耗方式也称为省电方式，80C51 系列单片机除具有一般的程序执行方式外，还具有两种低功耗方式：待机或称空闲（IDLE）方式和掉电或称停机（Power-down）方式，备用电源直接由 VCC 端输入。第一种方式可使功耗减小，电流一般为正常工作时的 15%，而后一种方式可使功耗减到最小，电流一般为 15mA 以下，最小可降到 50μA。因此，这种单片机适合于低功耗应用场合。

2.8.1 电源控制寄存器 PCON

在 80C51 系列单片机中有一个专用电源控制寄存器 PCON，通过对其中有关位的设置，可以选择待机（或称空闲）方式和掉电（或称停机）方式。其各位格式如下。

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

各位的作用如下所述。

(1) SMOD: 波特率倍增位。在串行口工作方式 1、2 或 3 下, SMOD=1 使波特率加倍 (详见第 7 章)。

(2) GF1 和 GF0: 通用标志位。用户用软件置、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。下面介绍两种低功耗方式的操作过程。

2.8.2 待机方式

1. 待机方式的工作特点

在待机方式下, 振荡器继续运行, 时钟信号继续提供给中断逻辑、串行口和定时器, 但提供给 CPU 的内部时钟信号被切断, CPU 停止工作。这时堆栈指针 SP、程序计数器 PC、程序状态字 PSW、累加器 ACC 以及所有的工作寄存器内容都被保留起来。

通常, CPU 耗电量占芯片耗电的 80%~90%, 所以 CPU 停止工作就会大大降低功耗。在待机方式下, AT89S51/S52 消耗的电流可由正常的 20mA 降为 50 μ A, 甚至更低。

2. 单片机进入待机方式的方法

如果向 PCON 中写 1 字节, 使 IDL=1, 单片机即进入待机方式。例如, 执行“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。

3. 单片机终止待机方式的方法

终止待机方式的方法有以下两种。

(1) 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 复 0, 单片机即退出待机状态, 从它停止运行的地址恢复程序的执行, 即从空闲方式的启动指令之后继续执行。注意, 为了防止对端口的操作出现错误, 置空闲方式指令的下一条指令不应该是写端口或写外部 RAM 的指令。

(2) 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, 单片机就进入中断服务程序, 这时通用标志 GF0 或 GF1 可用来指示中断是在正常操作还是在待机期间发生的。例如使单片机进入待机方式的那条指令也可同时将通用标志置位, 中断服务程序可以先检查此标志位, 以确定服务的性质。中断结束后, 程序将从空闲方式的启动指令之后继续执行。

2.8.3 掉电方式

1. 掉电方式的工作特点

在掉电方式下, VCC 可降至 2V, 使片内 RAM 处于 50 微安左右的“饿电流”供电状态,

以最小的耗电保存信息。在进入掉电方式之前，VCC 不能降低，而在退出掉电方式之前，VCC 必须恢复正常的电压值。VCC 恢复正常之前，不可进行复位。当单片机进入掉电方式时，必须使外围器件、设备都处于禁止状态。为此，在请求进入掉电方式之前，应将一些必要的数写入到 I/O 口的锁存器中，以禁止外围器件或设备产生误动作。例如，当系统扩展有外部数据存储时，在进入掉电方式之前，应当在 P2 口中置入适当数据，使之不产生任何外部存储器的片选信号。

在这种方式下，片内振荡器被封锁，一切功能都停止，只有片内 RAM 单元的内容被保留，端口的输出状态值都保存在对应的 SFR 中， $\overline{\text{ALE}}$ 和 $\overline{\text{PSEN}}$ 都为低电平。

2. 单片机进入掉电方式的方法

PCON 寄存器的 PD 位控制单片机进入掉电方式。当 CPU 执行一条置 PCON.1 位 (PD) 为 1 的指令后，单片机就进入掉电方式。例如，执行过“ORL PCON, #2”指令后，单片机即进入掉电方式。

3. 单片机退出掉电方式的方法

退出掉电方式的唯一方法是硬件复位，硬件复位 10ms 即能使单片机退出掉电方式。复位后将所有的特殊功能寄存器的内容重新初始化，但内部 RAM 区的数据不变。

思考与练习

1. AT89S51/S52 单片机内部包含哪些主要逻辑功能部件？各有什么主要功能？
2. 什么是指令？什么是程序？简述程序在单片机中的执行过程。
3. 如何认识 AT89S51/S52 存储器空间在物理结构上可划分为 4 个空间，而在逻辑上又可划分 3 个空间？
4. 开机复位后，CPU 使用的是哪组工作寄存器？它们的地址是什么？CPU 如何确定和改变当前工作寄存器组？
5. 什么是堆栈？堆栈有何作用？在程序设计时，有时为什么要对堆栈指针 SP 重新赋值？如果 CPU 在操作中要使用两组工作寄存器，SP 的初值应为多大？
6. AT89S51/S52 的时钟周期、机器周期、指令周期是如何分配的？当振荡频率为 8MHz 时，一个单片机机器周期为多少微秒？
7. 在 AT89S51/S52 扩展系统中，片外程序存储器和片外数据存储器共处同一地址空间为什么不会发生总线冲突？
8. 程序状态寄存器 PSW 的作用是什么？常用状态标志有哪几位？作用是什么？
9. 位地址 7CH 与字节地址 7CH 有何区别？位地址 7CH 具体在内存中什么位置？
10. AT89S51/S52 4 个 I/O 端口的作用是什么？AT89S51/S52 的片外三总线是如何分配的？
11. AT89S51/S52 的 4 个 I/O 端口在结构上有何异同？使用时有何注意事项？
12. 复位的作用是什么？有几种复位方法？复位后单片机的状态如何？
13. AT89S51/S52 有几种低功耗方式？如何实现？

第3章 指令系统

指令系统是熟悉和应用单片机必要的软件基础，不同种类的单片机指令系统一般是不同的，但 89 系列单片机属于 80C51 系列，因而它的指令系统与 80C51 系列相同，也就是与 MCS-51 系列相同。本章将详细介绍 80C51 系列单片机指令系统的寻址方式、各类指令的格式及功能。学习和使用单片机的一个重要环节就是理解和熟练掌握它的指令系统。但要真正掌握指令系统，一方面必须与单片机的硬件结构结合起来，另一方面要结合实际问题的程序做练习，才能达到应有的效果。

3.1 指令系统简介

指令是规定单片机进行某种操作的命令。一条指令只能完成有限的功能，为使单片机完成一定的或复杂的功能就需要一系列指令。单片机能够执行的各种指令的集合就称为指令系统。单片机的主要功能也是由指令系统来体现的。一般来说，若这台单片机的指令越丰富，寻址方式（见 3.2 节）越多，且每条指令的执行速度都较快，则它的总体功能越强。

80C51 的指令系统共有 111 条指令，见附录 A。如按运算速度分类，单周期指令为 64 条，双周期指令为 45 条，四周期指令为 2 条；如按字节数分类，其中单字节指令 49 条，双字节指令 45 条，三字节指令 17 条。由此可见，80C51 指令系统在占用存储空间和运行时间方面，效率都比较高。另外，80C51 单片机有丰富的位操作指令，这样可方便地用于各种逻辑控制，这是所有单片机指令系统的共同特点。

由于单片机只能识别二进制数，所以指令均由二进制代码组成。通常称这样的指令为机器指令，简称机器码。一般的单片机都有几十甚至几百条指令。为了阅读和书写方便，常把它写成十六进制形式。显然，即使用十六进制去书写和记忆也是极不方便、不容易的。因而，为了人们记忆和使用方便，制造厂家对指令系统的每一条指令都给出了助记符。助记符是根据机器指令不同的功能和操作对象来描述指令的符号，由于助记符是用英文缩写来描述指令特征的，因此它不但便于记忆，也便于理解和分类。这种用助记符形式来表示的机器指令称为汇编语言指令。单片机的指令一般用汇编语言指令来表示。

指令一般由两部分组成，即操作码和操作数。对于单字节指令有两种情况：一种是操作码、操作数均包含在这一字节之内；另一种情况是只有操作码无操作数。对于双字节指令，均为一字节是操作码，一字节是操作数。对于三字节指令，一般是一字节为操作码，二字节为操作数。

80C51 汇编语言指令格式如下：

操作码 [操作数] ； [注释]

操作码：是由助记符表示的字符串，它规定了指令的操作功能。

操作数：是指参加操作的数据或数据的地址。

注释：是为该条指令做的说明，以便于阅读。

在 80C51 指令系统中，操作码是指令的核心，不可缺少。操作码与操作数之间必须用空格分隔，操作数与操作数之间必须用逗号“,”分开。带方括号的项称可选项。

操作数可以是 1 个、2 个或 3 个，也可以没有。不同功能的指令，操作数的作用不同。例如，传送类指令多数有两个操作数，写在左面的称为目的操作数（表示操作结果存放单元地址），写在右面的称为源操作数（指出操作数的来源）。

操作数的表达方式较多，可以是常数、寄存器名、标号名、直接地址单元、表达式等，还可以使用一个特殊符号“\$”，用来表示程序计数器的当前值，通常用在转移指令中（见后面的例题）。操作数如果是常数，必须用整数，不能用分数和小数，负数用补码表示。

例如一条传送指令的书写格式为：

MOV A, 5BH ; (5BH)→A

它表示将 5BH 存储单元的内容送到累加器 A 中。

3.2 寻址方式

寻址方式是指在指令代码中用以表示操作数地址的各种规定，它指出了参与操作的数或数所在的地址。寻址方式与计算机的存储器空间结构是密切联系的，寻址方式越多则计算机的功能越强，灵活性亦越大，能更有效地处理各种数据。为了很好地理解、掌握指令系统，需先了解它的寻址方式。

3.2.1 符号约定

在说明 80C51 系列的指令及对其功能进行注释时，需要用到一些描述寄存器、地址及数据等符号，这些符号约定如下。

1. Rn

当前选中的工作寄存器组 R0~R7 (n=0~7)。它在片内数据存储器中的地址由 PSW 中 RS1、RS0 确定，可以是 00H~07H（第 0 组）、08H~0FH（第 1 组）、10H~17H（第 2 组）、18H~1FH（第 3 组）。

2. Ri

当前选中的工作寄存器组中可作为地址指针的两个工作寄存器 R0、R1 (i=0 或 1)。它在片内数据存储器中的地址由 RS0、RS1 确定，分别为 00H、01H；08H、09H；10H、11H；18H、19H。

3. #data

8 位立即数，即包含在指令中的 8 位常数。

4. #data16

16 位立即数，即包含在指令中的 16 位常数。

5. direct

8 位片内 RAM 单元的直接地址或寄存器（包括 SFR）。

6. addr11

11 位目的地址。用于 ACALL 和 AJMP 指令中，目的地址必须放在与下一条指令第一个字节同一个 2KB 程序存储器地址空间之内。

7. addr16

16 位目的地址。用于 LCALL 和 LJMP 指令中，目的地址范围在 64KB 程序存储器地址空间。

8. rel

补码形式的 8 位地址偏移量，表示相对跳转的偏移字节，用于相对转移指令中。偏移量以下一条指令第一字节地址为基值，偏移范围为-128~+127。

9. bit

片内 RAM 或特殊功能寄存器的直接寻址位地址。

10. @

间接寻址方式中，表示间址寄存器的前缀符号。

11. /

位操作指令中，表示对该位取反操作的前缀符号。

以下符号仅出现在指令注释和功能说明中。不同教科书表达方式略有不同。

12. X

片内 RAM 的直接地址（包含位地址）或寄存器。

13. (X)

表示 X 中的内容。

14. ((X))

在间接寻址方式中，表示由间址寄存器 X 指出的地址单元中的内容。

15. →

指令操作流程，将箭头左边的内容送入箭头右边的单元内。

3.2.2 寻址方式说明

80C51 系列指令共有 7 种寻址方式，下面将以指令为例逐一介绍。

1. 寄存器寻址

在这种寻址方式中操作数存放在所选定的寄存器中。这些寄存器包括工作寄存器 R0～R7、累加器 A、通用寄存器 B、地址寄存器 DPTR 等。

工作寄存器组的选择由状态标志寄存器 PSW 中的 RS1、RS0 确定。

例 1 执行指令 MOV A, R6。

指令代码的二进制形式为 11101110，十六进制为 EEH，注意其二进制数低 3 位为 110 正好为 6，表示操作数在 R6 中。这种情况就是操作码、操作数均包含在这一字节之内。

假设这条指令存放在 60H 单元，且 PSW 中 RS1、RS0 的值分别为 0、1，则可知现在的 R6 是属于第 1 组工作寄存器的，那么它的地址为 0EH。现已知 0EH 中存放着数值 58H，则执行该指令后，58H 就被送到 A 累加器中。该指令执行过程如图 3-1 所示（图中数字均为十六进制，并且省略后缀 H，以下同）。

2. 直接寻址

在这种寻址方式中，指令直接给出参加运算的操作数地址。在 80C51 单片机中，直接地址只能用来表示特殊功能寄存器、内部数据存储器以及位地址空间。其中特殊功能寄存器和位地址空间只能用直接寻址方式来访问。

例 2 执行指令 MOV A, 50H。

指令代码为 E5H、50H，是双字节指令。设把该指令放在程序存储区的 50H、51H 单元，而 RAM 区的 50H 单元中存放的数值为 3AH。当该指令执行后，数值 3AH 就被送到 A 累加器中。该指令执行过程如图 3-2 所示。

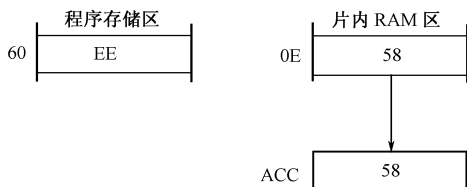


图 3-1 MOV A, R6 执行示意图

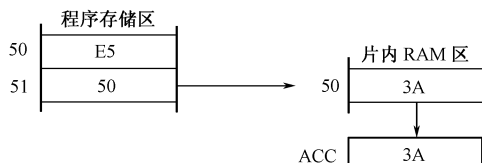


图 3-2 MOV A, 50H 执行示意图

3. 立即寻址

在这种寻址方式中，指令直接给出参加运算的操作数。指令多是双字节的，一般第一字节是操作码，第二字节是 8 位的操作数。在 80C51 的指令系统中，仅有“MOV DPTR, #data16”指令的操作数是 16 位的立即数。在立即寻址方式中的操作数直接参与操作，所以又称立即数，立即数就是存放在程序存储器中的常数，在立即数前面必须加“#”号表示。

例 3 执行指令 MOV A, #50H。

指令代码为 74H、50H，是双字节指令。

这条指令的功能是把立即数 50H 送入累加器 A 中。设把指令存放在存储区的 0040H、0041H 两单元（存放指令的起始地址是任意假设的），该指令执行过程如图 3-3 所示。

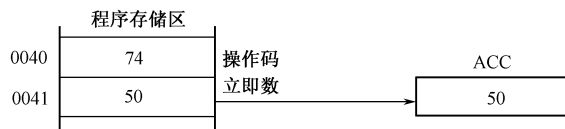


图 3-3 MOV A, #50H 执行示意图

例 4 执行指令 MOV DPTR, #563FH。

指令代码为 90H、56H、3FH，是三字节指令。其功能是向地址指针 DPTR 传送 16 位的立即数，即把立即数的高 8 位送入 DPH，低 8 位送入 DPL。假设把该指令放在存储区的 120H、121H、122H 单元。当执行该指令后，立即数 563FH 被送到 DPTR 中。该指令执行过程如图 3-4 所示。

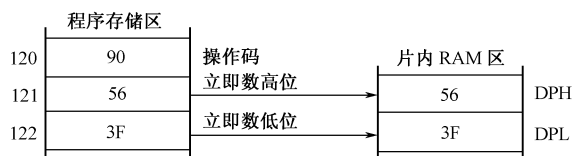


图 3-4 MOV DPTR, #563FH 执行示意图

4. 寄存器间接寻址

在这种寻址方式中，寄存器中存放的数值不是操作数，而是操作数的地址。这种寻址方式用于访问片内数据存储器（RAM）或片外数据存储器（RAM）。当访问片内 RAM 低 128 字节（AT89S52 为 256 字节）或片外 RAM 低 256 字节时，可采用当前工作寄存器组中的 R0 或 R1 作为间接地址寄存器，即由 R0 或 R1 间接给出操作数所在的地址，这样 R0 或 R1 为存放操作数单元的地址指针。在间接地址寄存器前面一定要加@前缀。

例 5 执行指令 MOV A, @R0。

指令代码的十六进制形式为 E6H，为单字节指令。

这条指令的源操作数存放在片内数据存储器中。假设此指令存放在 30H 单元，工作寄存器为第 0 组，R0 中存放 50H，50H 为片内 RAM 的一个单元，现 50H 中存放数值为 5AH，则执行该指令后，5AH 就送入 ACC 中。该指令执行过程如图 3-5 所示。

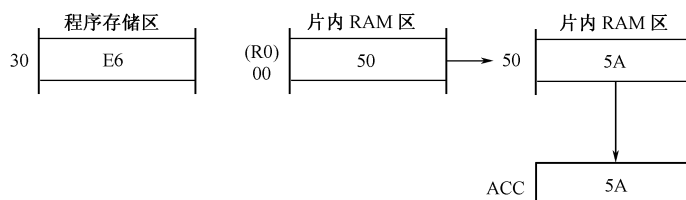


图 3-5 MOV A, @R0 执行示意图

例 6 执行指令 MOVX A, @R1。

指令代码的十六进制形式为 E3H，为单字节指令。这条指令的源操作数存放在片外数据存储器中。假设此指令存放在 170H 单元，工作寄存器组为第 3 组，在 R1 中存放的数值为 C3H，片外数据存储器 C3H 单元中值为 1FH，则执行该指令后，1FH 就送入 ACC 中。该指令执行过程如图 3-6 所示。

访问片外数据存储器时，还可用数据指针 DPTR 作为间址寄存器。DPTR 是 16 位寄存

器，故它可对整个 64KB 片外数据存储器空间寻址。

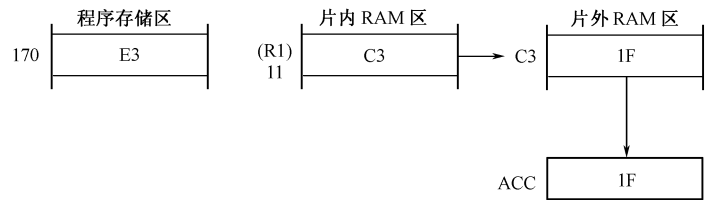


图 3-6 MOVX A, @R1 执行示意图

例如指令 MOVX A, @DPTR，其功能是把 DPTR 指定的片外 RAM 中的内容送到 A 中。在执行 PUSH（压栈）和 POP（出栈）指令时，采用堆栈指针 SP 做寄存器间接寻址。

5. 变址寻址（基址寄存器 + 变址寄存器间接寻址）

这种寻址方式以 DPTR 或 PC 为基址寄存器，累加器 A 为变址寄存器。变址寻址时，把两者的内容相加，所得到的结果作为操作数的地址。这种方式常用于查表操作。

例 7 执行指令 MOVX A, @A+DPTR。

指令代码为 93H，是单字节指令。

现假设此指令存放在 1070H 单元，ACC 中原存放值为 DCH，DPTR 中的值为 1000H，则 A+DPTR 形成的地址为 10DCH。10DCH 单元中的内容为 1FH，则执行该指令后，ACC 中原 DCH 被 1FH 代替。该指令执行过程如图 3-7 所示，ALU 在此实现加法操作。

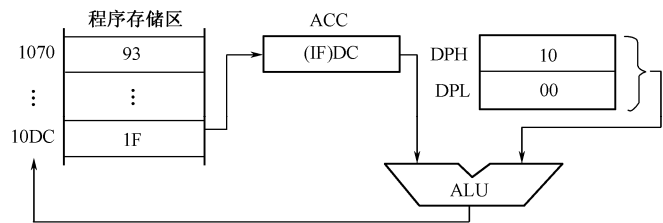


图 3-7 MOVX A, @A+DPTR 执行示意图

6. 相对寻址

相对寻址只用于相对转移指令中，这种寻址方式是将程序计数器 PC 中的当前内容与指令第二字节所给出的数相加，其结果作为跳转指令的转移地址，转移地址也称为转移目的地址。PC 中的当前内容称为基地址（它实际上是本指令之后的字节地址），指令第二字节给出的数据称为偏移量。偏移量为带符号的数，其表示的范围为+127~-128。目的地址是相对于 PC 的基地址而言的，所以这种寻址方式主要用于跳转指令。转移目的地址可按下式计算：

$$\text{目的地址} = \text{源地址} + \text{转移指令字节数} + \text{rel}$$

源地址指转移指令所在的单元地址，不同的转移指令字节数一般是不同的，通常为 2 或 3 字节。rel 为偏移量。在编程时这个偏移量的值可以直接用数字量，也可以用标号表示。通常不必手工计算，只需用地址标号代替即可（详见第 4 章），汇编程序可自动计算出目的地址。

例 8 执行指令 JNZ 30H。

指令代码为 70H、30H，是双字节指令。

此指令表示若 A=0，则程序顺序执行，即不跳转，PC = PC+2；若 A 不等于 0，则以 PC 中的当前内容为基地址，加上偏移量 30H 后所得到的结果为该转移指令的目的地址。

假设此指令存放在 1000H、1001H 单元，且目前 A 不等于 0，则取指令后，PC 当前内容为 1002H，对 A 进行判断后，把 PC 当前内容与偏移量 30H 相加，得到转移目的地址 1032H。所以执行完此指令后，PC 中的值为 1032H，程序将从 1032H 开始执行。该指令执行过程如图 3-8 所示。

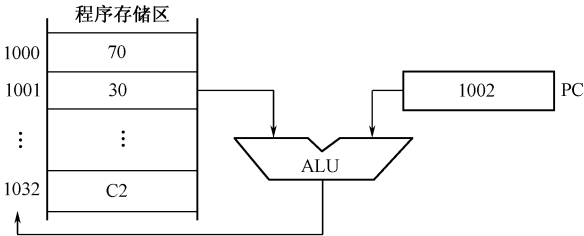


图 3-8 JNZ 30H 执行示意图

7. 位寻址

位寻址是指对片内 RAM 的位寻址区和某些可位寻址的特殊功能寄存器中的任一个二进制位进行位操作时的寻址方式。在进行位操作时，用进位位 C 作为操作累加器。操作数直接给出该位的地址，然后根据操作码的性质对其进行位操作。位地址与字节直接寻址中的字节地址形式完全一样，主要由操作码来区分，使用时必须予以注意。

例 9 执行指令 SETB 6AH。

指令代码为 D2H、6AH，是双字节指令。

6AH 这一位是片内 RAM 中 2DH 单元的第 2 位。假设 2DH 中原内容为 00H，那么执行此指令后，它就把 6AH 这一位置“1”。所以 2DH 中内容就变为 04H。图 3-9 为该指令执行示意图。

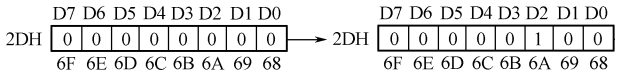


图 3-9 SETB 6AH 执行示意图

以上介绍了 80C51 指令系统中的 7 种寻址方式，表 3-1 概括了每种寻址方式所涉及的存储器空间。

表 3-1 操作数寻址方式和有关空间

寻 址 方 式	寻 址 空 间
立即寻址	程序存储器 ROM
直接寻址	片内 RAM 低 128 字节和特殊功能寄存器 SFR
寄存器寻址	工作寄存器 R0~R7, A, B, CY, DPTR
寄存器间接寻址	片内 RAM 低 128/256 字节[@R0, @R1, SP(仅 PUSH, POP)], 片外 RAM(@R0, @R1, @DPTR)
变址寻址	程序存储器(@A+PC, @A+DPTR)
相对寻址	程序存储器 256 字节范围 (PC+偏移量)
位寻址	片内 RAM 的 20H~2FH 字节地址和部分特殊功能寄存器 SFR

3.3 指令系统分类介绍

80C51 的指令系统按功能可分为五大类，即数据传送类、算术运算类、逻辑运算类、控制转移类、位操作类。本节将介绍各类指令的助记符、功能及寻址方式，相应的指令代码见附录 A。

3.3.1 数据传送类指令

数据传送类指令是最常用、最基本的一类指令。这类指令的操作一般是把源操作数传送到目的操作数，指令执行后，源操作数不变，目的操作数修改为源操作数。但交换型传送指令不丢失目的操作数，它只是把源操作数和目的操作数交换了存放单元。传送类指令一般不影响标志位，只有堆栈操作可以直接修改程序状态字 PSW。另外，对于传送目的操作数为 ACC 的指令将影响奇偶标志 P（后面就不再对此加以说明了）。

数据传送类指令用到的助记符有 MOV、MOVB、MOVC、XCH、XCHD、SWAP、PUSH、POP 共 8 种。源操作数可以采用寄存器、寄存器间接、直接、立即、变址 5 种寻址方式，目的操作数可以采用前 3 种寻址方式。数据传送指令共有 29 条，为便于记忆和掌握，下面根据这些指令的特点分为以下 5 类进行介绍。

1. 内部 RAM 数据传送指令

单片机内部的数据传送指令最多，包括寄存器、累加器、RAM 单元及专用寄存器之间数据的相互传送。下面分类介绍。

(1) 以累加器为目的操作数的指令（4 条）

汇编指令格式及注释如下：

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

这组指令的功能是将源操作数所指定的内容送入累加器 A。源操作数有寄存器、直接、寄存器间接和立即 4 种寻址方式。

上述指令在上节均有例题和图示，这里不再重复。

(2) 以寄存器 Rn 为目的操作数的指令（3 条）

汇编指令格式及注释如下：

```
MOV  Rn, A      ; (A)→Rn
MOV  Rn, direct ; (direct)→Rn
MOV  Rn, #data  ; data→Rn
```

这组指令的功能是把源操作数所指定的内容送到当前工作寄存器组 R0~R7 中的某个寄存器中。源操作数有寄存器、直接、立即 3 种寻址方式。注意，没有“MOV Rn, Rn”指令。

例 1 已知 A=5BH, R1=10H, R2=20H, R3=30H, (D0H)=4FH, 执行指令：

```
MOV  R1, A      ; (A)→R1
MOV  R2, 0D0H   ; (D0H)→R2
MOV  R3, #0B3H  ; B3H→R3
```

执行后，R1=5BH, R2=4FH, R3=B3H。

因为部分十六进制数是以字母开头的，而程序中的标号也是以字母开头的，为了与数字量区分，汇编语言规定凡是以字母开头的数字量，前面必须加一个数字“0”。直接地址也是用数字量表示的，所以字母前面也要加“0”。用“#”区别直接地址与立即数，所以在上述指令中 D0H 表示直接地址，B3H 表示一个值为 B3H 的立即数。

这一类指令操作过程类似于第一类。

(3) 以直接地址为目的的操作数的指令（5 条）

汇编指令格式及注释如下：

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

这组指令的功能是把源操作数所指定的内容送入由直接地址 **direct** 所指出的片内存储单元中。源操作数有寄存器、直接、寄存器间接、立即等寻址方式。

例 2 已知：R1 中内容为 57H，片内存储单元 57H 中的内容为 E7H，现执行如下指令：

```
MOV 43H, @R1 ; ((R1))→(43H)
```

该指令执行过程如图 3-10 所示。执行结果 43H 单元中为 E7H。

注意，“MOV direct, direct”指令在译成机器码时，源地址在前，目的地址在后，如“MOV 0A0H, 90H”机器码为 85, 90, A0。

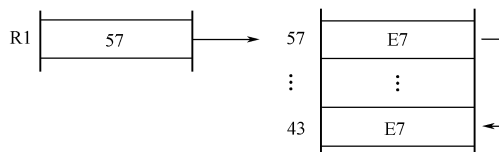


图 3-10 MOV 43H, @R1 执行示意图

另外，在汇编指令中，寄存器可写成地址形式，也可写成代号形式。例如：

MOV P1, A, 也可写成 MOV 090H, A, 因为 90H 为 P1 的地址。

(4) 以间接地址为目的的操作数的指令（3 条）

汇编指令格式及注释如下：

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

这组指令的功能是把源操作数所指定的内容送入以 R0 或 R1 为地址指针的片内存储单元中。

源操作数有寄存器、直接和立即 3 种寻址方式。

此组指令与上一组功能类似，不再举例。

(5) 16 位数据传送指令（1 条）

汇编指令格式及注释如下：

```
MOV DPTR, # data16 ; dataH→DPH, dataL→DPL
```

在 80C51 系列单片机中这是唯一的 16 位立即数传送指令。其功能是把 16 位常数送入 DPTR，在上节已举例说明。但 AT89S51/S52 单片机具有双 DPTR，通常默认的 DPTR 为 DPTR0，如果要采用 DPTR1，在使用前要编写指令：MOV A2H, #1（设置辅助寄存器 DPS 位为 1，选取 DPTR1）。执行此指令后当再出现与 DPTR 有关的指令时，则其实际的操作对象为 DPTR1。

注意，在译成机器码时，高字节在前，低字节在后，如“MOV DPTR, #1234H”，其

机器码是 90、12、34。

2. 外部数据传送指令（4 条）

在 80C51 指令系统中，CPU 对片外 RAM 的访问只能用寄存器间接寻址的方式。

汇编指令格式及注释如下：

```
MOVX  A, @Ri          ; ((Ri))→A
MOVX  @Ri, A           ; (A)→(Ri)
MOVX  A, @DPTR         ; ((DPTR))→A
MOVX  @DPTR, A         ; (A)→(DPTR)
```

外部数据传送指令主要是实现累加器 A 与片外数据存储器之间的数据传送，但要注意的是，对于 AT89C52/S52 单片机片内的高 128 字节 RAM 不能用这类指令传送，见例 3。

片外数据存储器的地址总线低 8 位和高 8 位分别与 P0 口和 P2 口相接，数据信息通过 P0 口与低 8 位地址选通信号分时传送。

前两条指令用 R0 或 R1 作为低 8 位地址指针，由 P0 口送出，寻址范围是 256 字节。如果 P2 不参加寻址，此时 P2 口仍可用做通用 I/O 口。这两条指令完成以 R0 或 R1 为低 8 位地址指针的片外数据存储器与累加器 A 之间的数据传送。如果 P2 参加寻址，则不要再把 P2 作为通用 I/O 使用。

后两条指令以 DPTR 为片外数据存储器 16 位地址指针，寻址范围达 64KB。其功能是在 DPTR 所指定的片外数据存储器与累加器 A 之间传送数据。

由于在 80C51 指令系统中，没有专门对外设的输入/输出指令，且片外扩展的 I/O 接口地址与片外 RAM 是统一编址的，因此，在片外数据存储器的地址空间上插入 I/O 接口，则上面的 4 条指令就可以作为输入/输出指令。80C51 单片机只能用这种指令方式与外部设备打交道。

例 3 把 AT89C52/S52 单片机片内 90H 单元中的数取出，传送到口地址为 B000H 的设备中去。

解：根据题意可编程序如下：

```
MOV  R0, #90H          ; 90H→R0
MOV  A, @R0             ; ((R0))→A
MOV  DPTR, #0B000H      ; B000H→DPTR
MOVX @DPTR, A           ; (A)→(DPTR)
```

3. 查表指令（2 条）

汇编指令格式及注释如下：

```
MOVC  A, @A+PC          ; PC+1→PC, ((A+PC))→A
MOVC  A, @A+DPTR        ; ((A+DPTR))→A
```

在 80C51 指令系统中，这 2 条指令主要用于查表，其数据表格通常放在程序存储器中。这两条指令执行后，不改变 PC 或 DPTR 的内容。

第一条指令为单字节指令，当 CPU 读取第一条指令后，PC 的内容自动加 1。其操作是将新的 PC 的内容与累加器 A 内 8 位无符号数相加形成地址，取出该地址单元中的内容送累加器 A。这种指令查表很方便，但只能查找指令所在地址以后 256 字节范围内的代码或常数。

例 4 已知在程序存储器中，数据表格为：

```
0210H: 02H
0211H: 0BH
0212H: 06H
```

0213H; 09H

执行程序

```
0200H: MOV  A,  #0EH      ; 0EH→A
0202H: MOVC A,  @A+PC      ; (0EH+0203H)→A
0203H: MOV  R0, A          ; (A)→R0
```

结果: A=0BH, R0=0BH, PC=0204H。

第二条指令是以 DPTR 为基址寄存器进行查表。使用前, 先给 DPTR 赋予某指定查表地址, 其范围可达整个程序存储器 64KB 空间。但此前, 若 DPTR 已赋值作为它用, 装入新查表地址值之前必须保存原值, 可用栈操作指令“PUSH”保存。

例 5 已知在程序存储器中, 数据表格为:

.....

```
3011H: 0DH
3012H: 07H
3013H: 35H
```

执行程序

```
204H: MOV  A,  #13H        ; 13H→A
206H: MOV  DPTR, #3000H     ; 3000H→DPTR
209H: MOVC A, @A+DPTR      ; (13H+3000H)→A
```

结果: A=35H, PC=20AH。

4. 堆栈操作指令 (2 条)

汇编指令格式及注释如下:

```
PUSH direct    ; (SP)+1→SP, (direct)→(SP)
POP  direct     ; ((SP))→(direct), (SP)-1→SP
```

第一条指令称入栈 (或称压栈或进栈) 指令, 其功能是将栈指针 SP 的内容加 1, 然后将直接寻址单元中的数传送 (或称压入) 到 SP 所指示的单元中。若数据已推入堆栈, 则 SP 指向最后推入数据所在的存储单元 (即指向栈顶)。

第二条指令称出栈 (也称弹出) 指令, 其功能是将栈指针 SP 所指示单元的内容送入直接寻址单元中, 然后将 SP 的内容减 1, 此时 SP 指向新的栈顶。

使用堆栈时, 一般需重新设定 SP 的初始值。由于压入堆栈的第一个数, 必须存放在 SP+1 存储单元, 故实际栈底是在 SP+1 所指示的单元。

另外, 要注意留出足够的存储单元作为栈区, 因为栈顶是随数据的压入和弹出而变化的, 如栈区设置不当, 则可能发生数据重叠, 这样会引起程序混乱, 以至无法运行。

一般情况, 执行此指令不影响标志, 但若目标操作数为 PSW, 则有可能使一些标志改变。这也是通过指令强行修改标志的一种方法。

下面举例说明执行压入和弹出指令的过程。

例 6 已知堆栈指针为 30H, 片内 RAM 50H 单元中存放的数值为 3FH, 把此数值压入堆栈, 然后再弹出到 67H 单元中。

根据题意编写指令如下:

```
MOV    SP, #30H      ; 30H→SP, 设堆栈指针
PUSH   50H            ; (SP)+1→SP, (50H)→(31H), 3FH 送入 31H 中
POP    67H            ; (31H)→(67H), (SP)-1→SP
```

程序执行过程如图 3-11 所示。

由图可见程序执行结果，数值 3FH 装入 67H 单元内，SP 终值为 30H。

5. 交换指令（5 条）

汇编指令格式及注释如下：

```
XCH  A, Rn      ; A  $\longleftrightarrow$  Rn
XCH  A, direct  ; (A) $\longleftrightarrow$ (direct)
XCH  A, @Ri     ; (A) $\longleftrightarrow$ ((Ri))
XCHD A, @Ri     ; (A.3~A.0) $\longleftrightarrow$ ((Ri.3~Ri.0))
SWAP A          ; (A.3~A.0) $\longleftrightarrow$ (A.7~A.4)
```

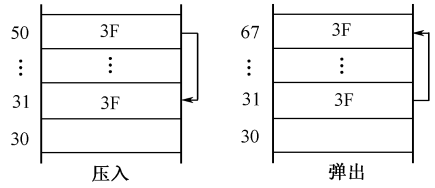


图 3-11 例 6 程序执行示意图

这组指令的前 3 条为全字节交换指令。其功能是将累加器 A 与源操作数所指出的数据相互交换。其操作执行过程如图 3-12 所示。

这组指令的后两条为半字节交换指令。其中 XCHD A, @Ri 是将累加器 A 中低 4 位与 Ri 中的内容所指示的片内 RAM 单元中的低 4 位数据相互交换，各自的高 4 位不变。

例 7 已知 A 中内容为 FAH，R0 中内容为 5FH，5FH 单元中内容为 75H，要求交换 A 与 5FH 中的低半字节。

执行指令 XCHD A, @R0 后，则可实现此交换，此时 ACC 中内容变为 F5H，5FH 中内容变为 7AH。该指令执行过程如图 3-13 所示，括号中的数为交换前的值。



图 3-12 一字节交换指令执行示意图

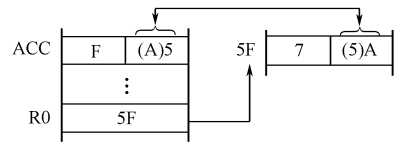


图 3-13 XCHD A, @R0 执行示意图

SWAP A 指令是将累加器 A 的高低两半字节交换。例如：A=F0H，执行指令 SWAP A 后，A=0FH。

例 8 已知 A 中内容为 CEH，R3 中内容为 34H。说明执行下述程序后 A 与 R3 中的内容。

```
XCH  A, R3
SWAP A
XCH  A, R3
```

解：根据上述程序对指令解释如下：

```
XCH  A, R3      ; A 与 R3 中的内容交换，交换后 A 中为 34H，R3 中为 CEH
SWAP A          ; 对 A 的高低半字节交换，交换后 A 中内容为 43H
XCH  A, R3      ; A 与 R3 中的内容交换，交换后 A 中为 CEH，R3 中为 43H
```

由结果可见 R3 变为 43H，A 的内容最后没变，它在这段程序中起到中间寄存器的作用。从上述传送指令中可以看出，累加器 A 是一个特别重要的寄存器，无论 A 作为目的寄存器还是作为源寄存器，CPU 对它都具有专用指令。若用 A 的地址 E0H 直接寻址，也可以实现上述功能，但机器码要多一字节，执行周期也会加长。工作寄存器 Rn 也有类似特点。

上述指令中，当前工作寄存器组由 PSW 中 RS1、RS0 选定，Rn 对应于该组寄存器 R0~R7 中的某一个。直接地址 direct 指出的存储单元为片内 RAM 的 00H~7FH 单元及 80H~FFH 中的 23/29 个特殊功能寄存器 SFR。

在用 @Ri 的间接寻址方式时，用当前 R0 或 R1 作为地址指针，用指令 MOV 和 MOVX 可访问片内 RAM 的 00H~7FH 共 128 个单元（对于 AT89C/S52 单片机还包括 80H~FFH 的

128 个单元) 和片外 RAM 的低 256 (00H~FFH) 个单元。

3.3.2 算术运算类指令

算术运算类指令主要是对 8 位无符号数据进行算术操作, 其中包括加法、减法、加 1、减 1 以及乘法和除法运算指令; 借助溢出标志, 可对有符号数进行补码运算; 借助进位标志, 可进行多精度加、减运算; 也可以对 BCD 码进行运算。

算术运算指令都影响程序状态标志寄存器 PSW 的有关位。对这一类指令要特别注意正确地判断结果对标志位的影响。

算术运算类指令共有 24 条, 下面分类加以介绍。

1. 加法指令 (4 条)

汇编指令格式及注释如下:

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

这组指令的功能是把源操作数所指出的内容加到累加器 A, 其结果存在 A 中。加法指令执行示意图见图 3-14。

加法运算指令的执行结果影响 PSW 的进位位 CY、溢出位 OV、半进位位 AC 和奇偶校验位 P。在加法运算中, 如果位 7 有进位, 则进位位 CY 置 1, 否则清 0; 如果位 3 有进位, 则半进位位 AC 置 1, 否则清 0。若看做两个带符号数相加, 还要判断溢出位 OV, 若 OV 为 1, 表示和数溢出。

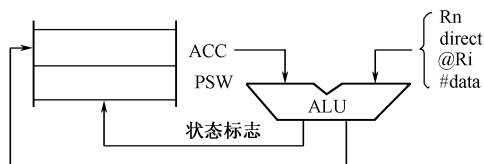


图 3-14 加法指令执行示意图

例 9 已知 A=8CH, 执行指令 ADD A, #85H, 则操作如下。

$$\begin{array}{r} 10001100 \\ +) 10000101 \\ \hline 1\ 00010001 \end{array}$$

结果: A=11H, CY=1, OV=1, AC=1, P=0。

此例中, 若把 8CH、85H 看做无符号数相加, 则结果为 111H, 当看做无符号数时, 不考虑 OV 位; 若把上述两值看做有符号数, 则有两个负数相加得到正数的错误结论, 此时 OV=1 表示有溢出, 指出了这一错误。

2. 带进位加法指令 (4 条)

汇编指令格式及注释如下:

```
ADDC  A, Rn      ; (A)+(Rn)+CY→A
ADDC  A, direct   ; (A)+(direct)+CY→A
```

ADDC A, @Ri ; (A)+((Ri))+CY→A

ADDC A, # data ; (A)+data+CY→A

这组指令的功能是把源操作数所指出的内容和累加器内容及进位标志 CY 相加，结果存放在 A 中。运算结果对 PSW 各位的影响同上述加法指令。

带进位加法指令多用于多字节数的加法运算，在低位字节相加时要考虑低字节有可能向高字节进位。因此，在做多字节加法运算时，必须使用带进位的加法指令。

例 10 已知 A=8CH, R1=85H, CY=1, 执行指令 ADDC A, R1, 则操作如下。

$$\begin{array}{r} 10001100 \\ 10000101 \\ +) \quad \quad \quad 1 \\ \hline 1 \ 00010010 \end{array}$$

结果: A=12H, CY=1, OV=1, AC=1, P=0。

3. 带借位减法指令（4 条）

汇编指令格式及注释如下：

SUBB A, Rn ; (A)−(Rn)−CY→A

SUBB A, direct ; (A)−(direct)−CY→A

SUBB A, @Ri ; (A)−((Ri))−CY→A

SUBB A, # data ; (A)−data−CY→A

这组指令的功能是将累加器 A 中的数减去源操作数所指出的数和进位位 CY，其差值存放在累加器 A 中（注意 80C51 指令系统中没有不带借位的减法指令）。

减法运算指令执行结果影响 PSW 的进位位 CY、溢出位 OV、半进位位 AC 和奇偶校验位 P。

在多字节减法运算中，被减数的低字节有时会向高字节产生借位（即 CY 置 1），所以在多字节运算中必须用带借位减法指令。在进行单字节减法或多字节的低 8 位字节减法运算时，应先将程序状态标志寄存器 PSW 的进位位 CY 清 0。

本组指令执行过程与加法类似，这里不再图示。需要强调的一点是，减法运算在单片机中实际上是变成补码相加，下面举例说明。

例 11 已知 A=ACH, R4=67H, CY=1, 执行指令 SUBB A, R4, 则操作如下。

10101100 (ACH)	10101100
01100111 (67H)	10011001 (67H的补码)
−) 1 (CY)	+) 11111111 (−1的补码)
01000100	10 01000100
常规减法	减法变补码相加

结果: A=44H, CY=0, AC=1, OV=1。

由上述两式可见，两种算法的最终结果是一样的。在此例中，若看做两个无符号数相减，则结果 44H 是正确的；若看做有符号数相减，则得出负数减正数结果是正数的错误结论，OV=1，指出了这一错误。

4. 乘法指令（1 条）

汇编指令格式及注释如下：

MUL AB ; (A)×(B)→BA, B15~8, A7~0

这条指令的功能是把累加器 A 和寄存器 B 中两个无符号 8 位数相乘，所得 16 位乘积低字节存放在 A 中，高字节存放在 B 中。若乘积大于 FFH，则 OV 置 1，否则清 0；CY 总是为 0。另外，此指令也影响奇偶标志位。

乘法运算指令执行结果影响 PSW 的溢出位 OV、奇偶校验位 P。

例 12 已知 A=ABH，B=47H，执行指令 MUL AB，则操作如下。

$$\begin{array}{r} 10101011 \text{ (ABH)} \\ \times) 01000111 \text{ (47H)} \\ \hline 10111101101101 \text{ (2F6DH)} \end{array}$$

结果：B=2FH，A=6DH，OV=1，P=1。

5. 除法指令（1 条）

汇编指令格式及注释如下：

DIV AB ; (A)÷(B)的商→A，余数→B

这条指令的功能是进行 A 除以 B 的运算，A 和 B 的内容均为 8 位无符号整数。指令执行后，整数商存于 A 中，余数存于 B 中。

本指令执行结果影响 PSW 的溢出位 OV 和奇偶校验位 P。指令执行后，标志 CY 和 OV 均清 0，当除数为 0 时，A 和 B 中的内容为不确定值，此时 OV 标志置 1，说明除法溢出，A 中的内容影响奇偶校验位 P。

例 13 已知 A=ADH，B=0CH，求 A 除以 B 的结果。

解：执行指令 DIV AB，得计算结果如下：

A=14，B=5，CY=0，OV=0，P=0。

6. 加 1/减 1 指令（9 条）

汇编指令格式及注释如下：

INC	A	; (A)+1→A
INC	Rn	; (Rn)+1→Rn
INC	direct	; (direct)+1→(direct)
INC	@Ri	; ((Ri))+1→(Ri)
INC	DPTR	; (DPTR)+1→DPTR
DEC	A	; (A)-1→A
DEC	Rn	; (Rn)-1→Rn
DEC	direct	; (direct)-1→(direct)
DEC	@Ri	; ((Ri))-1→(Ri)

这组指令的功能是将操作数所指定单元的内容加 1/减 1，加 1、减 1 指令仅当源操作数为 A 时，对 PSW 的奇偶校验位 P 有影响，其余指令操作均不影响 PSW。

第 3 条和第 8 条指令中的直接地址，如果是 I/O 端口，则自动进行“读-改-写”操作。其功能是修改出入口的内容。指令执行时，首先读入端口的内容，然后进行加/减 1 操作，再输出到端口，应注意读入内容来自端口锁存器而不是端口引脚。

第 5 条指令是唯一的一条 16 位加 1 指令。这条指令在加 1 过程中，若低 8 位有进位，可自动向高 8 位进位。

例 14 已知 DPTR=20FFH, 执行指令 INC DPTR。

结果: DPTR=2100H。

7. 十进制调整指令 (1 条)

汇编指令格式及注释如下:

DA A

这条指令是在进行 BCD 码加法运算时, 跟在 ADD 和 ADDC 指令之后, 用来对压缩 BCD 码 (所谓压缩 BCD 码, 指在一字节中存放两位 BCD 码, 通常简称 BCD 码) 的加法运算结果自动进行修正, 使其仍为 BCD 码表达形式。BCD 码表见附录 B。

下面说明为什么要用 DA A 指令, 如何使用 DA A 指令。

在单片机中, 十进制数 0~9 之间的数字可以用 BCD 码 (用二进制表示的十进制数) 来表示, 然而单片机在进行运算时, 是按照二进制规则进行的, 对于 4 位二进制数是按逢 16 进位的, 不符合十进制的要求, 可能导致错误的结果。

例如: 执行加法指令 ADD A, #84, 已知累加器 A 中 BCD 数是 99。

由于在 CPU 中是按二进制加法进行的, 所以上述指令在正常情况下结果如下:

$$\begin{array}{r} 10000100 \text{ (84的BCD码)} \\ +) 10011001 \text{ (99的BCD码)} \\ \hline 100011101 \end{array}$$

显然, 所得值为非法 BCD 码, 但如果在这条指令后接着运行一条 DA A 指令, 则 CPU 将自动把上述结果高、低 4 位分别加 6 调整, 就可得到正确的 BCD 码表达数, DA A 指令将自动进行如下操作:

$$\begin{array}{r} 100011101 \\ +) 01100110 \\ \hline 110000011 \end{array}$$

所得 BCD 数为 183, 结果正确。由上例可知, 当两个 BCD 数之和出现大于 9 的情况时, 必须对结果加 6 进行“十进制调整”, 才能得到正确的 BCD 数。

在单片机中, 遇到十进制调整指令时, 中间结果的修正是由 ALU 硬件中的十进制修正电路自动进行的。用户不必考虑何时该加“6”, 使用时只需在上述加法指令后面紧跟一条 DA A 指令即可。

注意, 在 80C51 机中, DA A 指令不对减法指令的结果进行修正。

3.3.3 逻辑操作类指令

这一类指令主要是用于对两个操作数按位进行与、或、异或逻辑操作, 操作结果送到累加器 A 或直接寻址单元。移位、取反、清除等操作也包括在这一类指令中。这些指令执行时一般不影响程序状态字寄存器 PSW, 仅当目的操作数为 ACC 时对奇偶标志位有影响。

逻辑运算类指令共 24 条, 下面分类加以介绍。

1. 逻辑与指令 (6 条)

汇编指令格式及注释如下:

ANL A, Rn ; (A) • (Rn) → A
ANL A, direct ; (A) • (direct) → A

ANL A,	@Ri	; (A) • (Ri)→A
ANL A,	#data	; (A) • data→A
ANL direct,	A	; (direct) • (A)→(direct)
ANL direct,	#data	; (direct) • data→(direct)

这组指令的功能是将两个指定的操作数按位逻辑与，结果存到目的操作数中。前 4 条指令是将累加器 A 的内容和操作数的内容按位逻辑与，结果存放在 A 中。指令执行结果影响奇偶标志位 P。

后两条指令是将直接地址单元中的内容和操作数所指出的内容按位逻辑与，结果存入直接地址单元中，指令执行结果不影响奇偶标志位。若直接地址为 I/O 端口，则为“读-改-写”操作。

例 15 已知 A=8FH, (60H)=96H, 执行指令 ANL A, 60H, 则操作如下:

10001111 (8FH)
•) 10010110 (96H)

10000110 (86H)

结果: A=86H, (60H)=96H, P=1。

2. 逻辑或指令 (6 条)

汇编指令格式及注释如下:

ORL A,	Rn	; (A)+(Rn)→A
ORL A,	direct	; (A)+(direct)→A
ORL A,	@Ri	; (A)+((Ri))→A
ORL A,	# data	; (A)+ data→A
ORL direct,	A	; (direct)+(A)→(direct)
ORL direct,	# data	; (direct)+ data→(direct)

这组指令的功能是将两个指定的操作数按位逻辑或，结果存到目的操作数中。执行后对奇偶标志位 P 及 I/O 端口的影响与上述逻辑与指令相同。

例 16 已知 A=2BH, R1=3CH, (3CH)=B9H, 执行指令 ORL A, @R1。

结果: A=BBH, R1=3CH, (3CH)=B9H, P=0。

例 17 将累加器 A 中低 4 位的状态, 通过 P1 口的高 4 位输出。

解: 根据题意可编程如下:

ANL A,	# 0FH	; 屏蔽 A.7~A.4
SWAP A		; 高、低半字节交换
ANL P1,	# 0FH	; 清 P1 口高 4 位
ORL P1,	A	; 使 P1.7~P1.4 按 A 中初始值的 A.3~A.0 值置位

本例中交换高、低半字节的功能也可用 4 条 RL A 指令实现。

3. 逻辑异或指令 (6 条)

汇编指令格式及注释如下:

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

这组指令的功能是将两个指定的操作数按位异或，结果存到目的操作数中。执行后对奇偶标志位 P 及 I/O 端口的影响与上述逻辑与指令相同。

例 18 已知 A=87H, (65H)=A7H, 执行指令 XRL 65H, A, 则操作如下:

$$\begin{array}{r} 10100111 \text{ (A7H)} \\ \oplus) 10000111 \text{ (87H)} \\ \hline 00100000 \text{ (20H)} \end{array}$$

结果: A=87H, (65H)=20H, P=1。

4. 循环移位指令（4 条）

汇编指令格式及注释如下:

RL	A	; (A)循环左移 1 位
RR	A	; (A)循环右移 1 位
RLC	A	; (A)带进位循环左移 1 位
RRC	A	; (A)带进位循环右移 1 位

图 3-15 为循环移位指令执行示意图。

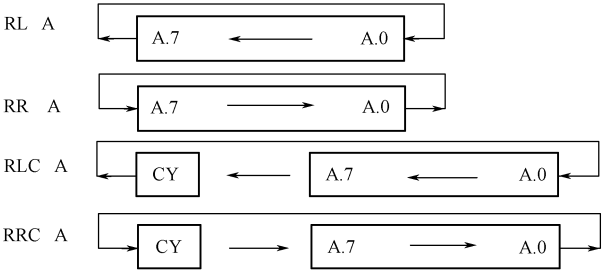


图 3-15 循环移位指令执行示意图

RL A 和 RR A 指令分别称为循环左移和循环右移指令。它们的功能分别是将累加器 A 的内容循环左移或右移一位，执行后不影响 PSW 中各位。

RLC A 和 RRC A 指令分别称为带进位循环左移和带进位循环右移指令。这两条指令的功能分别是将累加器 A 的内容与进位位 CY 一起循环左移或右移一位，执行后影响 PSW 中的进位位 CY 和奇偶状态标志位 P。

例 19 已知 A=EBH, CY=1, 执行指令 RLC A。

结果: A=D7H, CY=1, P=0。

5. 取反指令（1 条）

汇编指令格式及注释如下:

CPL A ; $\overline{A} \rightarrow A$

本指令的功能是将累加器 A 的内容按位取反。

例 20 已知 A=03H, 执行指令 CPL A。

结果: A=FCH。

6. 清 0 指令（1 条）

汇编指令格式及注释如下:

CLR A ;0→A

本指令功能是将累加器 A 的内容清 0。

3.3.4 控制转移类指令

这类指令的功能主要是控制程序从原顺序执行地址转移到其他指令地址上。

单片机在运行过程中，有时因为任务要求，需要改变程序运行方向，或者需要调用子程序，或者需要从子程序中返回，此时都需要改变程序计数器 PC 中的内容。控制转移类指令就可实现这一要求。

控制程序转移类指令包括无条件转移和条件转移，绝对转移和相对转移，长转移和短转移，还有调用和返回指令等。这类指令多数不影响程序状态标志寄存器。

80C51 指令系统中有 17 条（本节不包括位操作类的 4 条转移指令）控制程序转移类指令。下面分类加以介绍。

1. 无条件转移指令（4 条）

汇编指令格式及注释如下：

```
LJMP  addr16      ; addr16→PC
AJMP  addr11      ; (PC)+2→PC, addr11→PC.10~PC.0
SJMP  rel         ; (PC)+2+rel→PC
JMP   @A+DPTR     ; (A)+(DPTR)→PC
```

这类指令是指当程序执行完该指令时，程序就无条件地转到指令所提供的地址继续执行。下面分别予以说明。

LJMP addr16 指令称长转移指令。该指令中包含 16 位地址，所以转移的目标地址范围是程序存储器的 0000H~FFFFH。指令执行结果是将 16 位地址 addr16 送程序计数器 PC 中。

例 21 执行如下指令：

```
0023H    LJMP 2400H
```

执行后 PC 值由 0023H 变为 2400H。

AJMP addr11 指令称绝对（也称短）转移指令。该指令中包含要改变的低 11 位地址，转移的目标地址是在下一条指令地址开始的 2KB 范围内。它把 PC 的高 5 位与操作码的第 7~5 位（A10~A8）及操作数的 8 位并在一起，构成 16 位的转移地址，如图 3-16 所示。

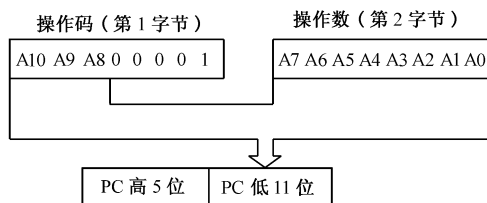


图 3-16 11 位转移地址示意图

因为地址高 5 位保持不变，为原 PC 的高 5 位，仅低 11 位发生变化，因此寻址范围必须在该指令地址加 2 后的 2KB 区域内。

例 22 执行如下指令：

```
300H     AJMP 600H
```

执行后 PC 值由 302H 变为 600H。

SJMP rel 指令是无条件相对转移指令。该指令为双字节，指令的操作数是相对地址，rel 是一个带符号的偏移字节数（2 的补码），其范围为-128~+127。负数表示向后转移，正数表示向前转移，该指令执行后目的地址值如下式计算：

目的地址值 = 本指令地址值 + 2 + rel。其执行过程类似于图 3-8 所示。

在用汇编语言编写程序时，可以用一个标号表示转移目标地址的偏移量 rel（见第 4 章），此时汇编程序可以自动计算出偏移地址，避免了人工汇编的麻烦，且不容易出错。

JMP @A+DPTR 指令称为无条件间接转移指令。该指令转移地址由数据指针 DPTR 的 16 位数和累加器 A 的 8 位无符号数相加形成，并直接送入 PC。指令执行过程对 DPTR、A 和标志位均无影响。这条指令可代替众多的判别跳转指令，具有散转功能（又称散转指令）。

在实际编程中不必写出具体的转移目的地址，通常只需要给出地址标号即可，详见例 24。

2. 条件转移指令（8 条）

汇编指令格式及注释如下：

JZ rel	; (A)=0 : (PC)+2+rel→PC (A)≠0 : (PC)+2→PC
JNZ rel	; (A)≠0 : (PC)+2+rel→PC (A)=0 : (PC)+2→PC
CJNE A, direct, rel	; (A)=(direct) : (PC)+3→PC, 0→C (A)>(direct) : (PC)+3+rel→PC, 0→C (A)<(direct) : (PC)+3+rel→PC, 1→C
CJNE A, # data, rel	; (A)=data : (PC)+3→PC, 0→C (A)>data : (PC)+3+rel→PC, 0→C (A)<data : (PC)+3+rel→PC, 1→C
CJNE Rn, # data, rel	; (Rn)=data : (PC)+3→PC, 0→C (Rn)>data : (PC)+3+rel→PC, 0→C (Rn)<data : (PC)+3+rel→PC, 1→C
CJNE @Ri, # data, rel	; ((Ri))=data : (PC)+3→PC, 0→C ((Ri))>data : (PC)+3+rel→PC, 0→C ((Ri))<data : (PC)+3+rel→PC, 1→C
DJNZ Rn, rel	; (Rn)-1→Rn, (Rn)≠0 : (PC)+2+rel→PC (Rn)=0 : (PC)+2→PC
DJNZ direct, rel	; (direct)-1→(direct), (direct)≠0 : (PC)+3+rel→PC (direct)=0 : (PC)+3→PC

这一类指令都是以相对转移的方式转向目标地址的。它们的共同特点是转移前要先测试某一条件是否满足，只有满足规定条件时，程序才能转到指定转移地址，否则程序将继续执行下一条指令。条件是由条件转移指令本身提供（或规定）的。

这组指令中前两条是累加器判别转移指令，通过判别累加器 A 中是否为 0，决定转移还是顺序执行。

第 3 至第 6 条为比较转移指令，是本指令系统中仅有的具有 3 个操作数（一个隐含在操作码中）的指令组。这些指令的功能是比较前两个无符号操作数的大小，若不相等，则转移，否则顺序执行。这 4 条指令影响 CY 位。执行结果不影响任何操作数。

最后两条指令是减 1 非零转移指令。使用此指令前要将计数值预置在工作寄存器或片内 RAM 直接地址中，然后再执行某段程序和减 1 判 0 指令。

例 23 将累加器 A 中的内容与立即数 25H 相加，如果和不等 80H，则程序跳转 7 字节后继续执行，否则顺序执行。已知程序起始地址是 40H。

解：根据题意可编程如下：

地址	机器码	源程序
40	24 25	ADD A, #25H ; (A)+25H→A
42	B4 80 07	CJNE A, #80H, 07 ; (A)≠80H; (PC)+3→PC (A)≠80H; (PC)+3+7→PC
45	...	
	...	
4C	24 9B	ADD A, #9BH

由程序可见，只有当 A 中原来的内容为 5BH 时程序才顺序执行，其他情况均跳转至 4CH 后继续执行。

例 24 将累加器 A 的内容由 0 递增，加到 50，结果放在累加器 A 中。

解：根据题意可编程如下：

地址	机器码	源程序
200	E4	CLR A ; 0→A
201	75 50 32	MOV 50H, #50 ; 50→(50H)
204	04	L1: INC A ; (A)+1→A
205	D5 50 FC	DJNZ 50H, L1 ; (50H)-1→(50H), (50H)≠0:PC+3-4→PC (50H)=0:PC+3→PC

标号 L1 在此代表偏移量，应该为-4，用补码表示为 FCH，当用标号表示后就不用换算成补码了。

3. 调用子程序及返回指令（4 条）

汇编指令格式及注释如下：

LCALL addr16	; (PC)+3→PC, (SP)+1→SP, (PCL)→(SP), (SP)+1→SP, (PCH)→(SP), addr16→PC
ACALL addr11	; (PC)+2→PC, (SP)+1→SP, (PCL)→(SP), (SP)+1→SP, (PCH)→(SP), addr11→PC.10~PC.0
RET	; ((SP))→PCH, (SP)-1→SP, ((SP))→PCL, (SP)-1→SP
RETI	; 除具有 RET 指令的功能外，还将清除内部相应的中断状态触发器， 详见第 8 章

这组指令用于实现从主程序中调用子程序和从子程序中返回到主程序的功能。调用和返回构成了子程序调用的完整过程。此类指令不影响标志位。

本组第一条指令称长调用指令（三字节）。执行时，先将 PC 加 3，指向下条指令地址（即断点地址），然后将断点地址压入堆栈，其先把 PC 的低 8 位 PCL 压入堆栈，再压入 PC 的高 8 位 PCH，然后把指令中的 16 位子程序入口地址装入 PC，程序转到子程序。子程序入口地址可以设在 64KB 的空间中。

第二条指令称绝对（也称短）调用指令（双字节），其保护断点地址过程同上，但 PC 只需加 2，其转入子程序入口的过程类似“LCALL”指令。被调用的子程序入口地址必须与调

用指令 ACALL 下一条指令的第一字节在相同的 2KB 存储区之内。其操作码的形成类似于“AJMP”指令。

第三条指令是子程序返回指令。执行时将堆栈内的断点地址弹出送入 PC，使程序返回到原断点地址。

最后一条指令是实现从中断子程序返回的指令，它只能用于中断服务程序作结束指令。RET 与 RETI 决不能互换使用。

例 25 已知调用指令 LCALL SUB1 的地址为 42H，子程序 SUB1 入口地址是 20BH，说明该段程序在调用过程中 PC 及 SP 的变化：

地址	指令	注释
40	MOV SP, # 30H	; 设置堆栈指针, 30H→SP
42	LCALL SUB1	; 调用子程序, 45H→PC, 31H→SP, 45H→(31H), 32H→SP, 00H→(32H), 20BH→PC
...	...	
20B	SUB1: MOV A, R0	
...	...	
212	RET	; (32H)→PCH, (31H)→PCL

执行结果：PC=0045H，SP=30H。

4. 空操作指令（1 条）

汇编指令格式及注释如下：

NOP

这是一条单字节指令，它控制 CPU 不进行任何操作（即空操作）而转到下一条指令。这条指令常用于产生一个机器周期的延迟。如果反复执行这一指令，则机器处于踏步等待状态。

3.3.5 位操作类指令

在 80C51 的硬件结构中，有位处理机（布尔处理机），它具有丰富的位处理功能。处理位变量的指令包括位变量传送、位逻辑运算、位条件转移等指令。

在 80C51 单片机的内部数据存储区中，20H～2FH 为位操作区域，其中每一位都有自己的位地址，可以对其每一位进行操作。位地址空间为 00H～7FH，共 16×8=128 位。另外，字节地址能被 8 整除的特殊功能寄存器的每一位也具有可寻址的位地址。

在进行位操作时，进位标志 CY 作为位累加器 C。

在汇编语言中位地址的表达方式有以下几种。

- 直接（位）地址方式：如 RAM 位寻址区的 0～7FH 中任意 1 位，SFR 寄存器中的部分寄存器可以直接位寻址，例如，PSW 寄存器中的 CY 位可以用 D7H 表示等。
- 点操作符号方式：在字节地址和位数之间用“.”隔开，如 PSW.3，(B8H).4 等。
- 位名称方式：如 RS0、P、OV 等。
- 用户定义名方式：用伪指令 bit 定义任意位，详见 4.1.2 节。

位操作类指令共 17 条，下面分类加以介绍。

1. 位数据传送指令（2 条）

汇编指令格式及注释如下：

```
MOV  C,    bit      ; (bit)→C
MOV  bit,   C        ; (C)→bit
```

这两条指令主要用于对位操作累加器 C 进行数据传送，均为双字节指令。

前一条指令的功能是将某指定位的内容送入位累加器 C 中，不影响其他标志。后一条指令是将 C 的内容传送到指定位，在对端口操作时，先读入端口 8 位的全部内容，然后把 C 的内容传送到指定位，再把 8 位内容传送到相应端口的锁存器，所以也是“读-改-写”指令。

例 26 已知片内 RAM(25H)=83H=10000011B，把 25H 的第 4 位传送到 C 中。

编写指令如下：

```
MOV  C, (25H).4      ; (25H).4→C
```

结果：C = 0。

例 27 把 P2.4 状态传送到 P1.5。按题意编写指令如下：

```
MOV  C,    P2.4      ; (P2.4)→C
MOV  P1.5,  C        ; (C)→P1.5
```

2. 位修正指令（6 条）

汇编指令格式及注释如下：

```
CLR  C          ; 0→C
CLR  bit         ; 0→bit
CPL  C          ; (/C)→C
CPL  bit         ; (/bit)→bit
SETB C          ; 1→C
SETB bit         ; 1→bit
```

这类指令的功能分别是对进位标志 C 或直接寻址位进行清除、取反、置位操作，执行结果不影响其他标志。当直接位地址为端口中某一位时，具有“读-改-写”功能。

3. 位逻辑运算指令（4 条）

汇编指令格式及注释如下：

```
ANL  C, bit      ; (C) • (bit)→C
ANL  C, /bit     ; (C) • (/bit)→C
ORL  C, bit      ; (C) + (bit)→C
ORL  C, /bit     ; (C) + (/bit)→C
```

这组指令的功能是把进位 C 的内容和直接位地址的内容逻辑与/或后的操作结果送回到 C 中。斜杠“/”表示对该位取反后再参与运算，但不改变原来的数值。

例 28 已知位 1FH = 1，CY = 0。

```
执行指令 ANL  C, 1F H      ; (C) • (1F H) →C, C 为 0
执行指令 ANL  C, /1F H     ; (C) • (/1F H) →C, C 为 0
执行指令 ORL  C, 1F H      ; (C) + (1F H) →C, C 为 1
执行指令 ORL  C, /1F H     ; (C) + (/1F H) →C, C 为 0
```

4. 判位转移指令（5 条）

汇编指令格式及注释如下：

```
JC  rel      ; (C)= 1:(PC)+2+rel→PC
              ; (C)= 0:(PC)+2→PC
```

```

JNC rel      ; (C)= 0:(PC)+2+rel→PC
              (C)= 1:(PC)+2→PC
JB  bit, rel ; (bit)= 1:(PC)+3+rel→PC
              (bit)= 0:(PC)+3→PC
JNB bit, rel ; (bit)= 0:(PC)+3+rel→PC
              (bit)= 1:(PC)+3→PC
JBC bit, rel ; (bit)= 1:(PC)+3+rel→PC, 0→bit
              (bit)= 0:(PC)+3→PC

```

这组指令的功能是分别判断进位 C 或直接寻址位是“1”还是“0”，条件符合则转移，否则继续执行程序。

前两条指令是双字节，所以 PC 要加 2，后 3 条指令是三字节，所以 PC 要加 3，其中，最后一条指令的功能是：若直接寻址位为“1”则转移，并同时将该位清 0，否则顺序执行。这类指令也具有“读-改-写”功能。

例 29 在片内 RAM 30H、40H 单元中有两个无符号数，如果 30H 中的数小则把片内 RAM 中的 30H 位置 1；若 40H 中的数小则把 40H 位置 1；若相等则把 F0 位置 1，然后返回。

解：设程序起始地址为 40H，根据题意可编程如下：

地址	机器码	源程序
40	E5 30	MOV A, 30H
42	B5 40 03	CJNE A, 40H, L1 ; 两数不等则转 L1
45	D2 D3	SETB F0 ; 两数相等，置 F0 为 1
47	22	RET ; 返回
48	40 03	L1: JC L2 ; 若 C 为“1”，则(30H)中数小，转 L2
4A	D2 40	SETB 40H ; (40H)数小则 40H 位置 1
4C	22	RET ; 返回
4D	D2 30	L2: SETB 30H ; (30H)数小，则 30H 位置 1
4F	22	RET ; 返回

思考与练习

1. 简述下列名词术语的基本概念：

指令、指令系统、程序、汇编语言指令。

2. 80C51 单片机有哪几种寻址方式？这几种寻址方式是如何寻址的？

3. 要访问特殊功能寄存器和片外数据存储器，应采用哪些寻址方式？

4. 80C51 单片机的指令系统可分为哪几类？试说明各类指令的功能。

5. 外部数据传送指令有哪几条？试比较下面每一组中两条指令的区别。

(1) MOVX A, @R0 ; MOVX A, @DPTR

(2) MOVX @R0, A ; MOVX @DPTR, A

(3) MOVX A, @R0 ; MOVX @R0, A

6. 在 AT89S51 片内 RAM 中，已知(30H)=38H，(38H)=40H，(40H)=48H，(48H)=90H，请分析下段程序中各指令的作用，并翻译成相应的机器码；说明源操作数的寻址方式，按序执行每条指令后的结果。

```

MOV A, 40H
MOV R0, A
MOV P1, #0F0H

```

```

MOV  @R0, 30H
MOV  DPTR, #1246H
MOV  40H, 38H
MOV  R0, 30H
MOV  90H, R0
MOV  48H, #30H
MOV  A, @R0
MOV  P2, P1

```

7. 试说明下列指令的作用，并将其翻译成机器码，执行最后一条指令对 PSW 有何影响？
A 的终值为多少？

(1) MOV R0, #72H

MOV A, R0

ADD A, #4BH

(2) MOV A, #02H

MOV B, A

MOV A, #0AH

ADD A, B

MUL AB

(3) MOV A, #20H

MOV B, A

ADD A, B

SUBB A, #10H

DIV AB

8. DA A 指令的作用是什么？怎样使用？

9. 试编程将片外数据存储器 60H 中的内容传送到片内 RAM 54H 单元中。

10. 试编程将寄存器 R7 的内容传送到 R1 中去。

11. 已知当前 PC 值为 210H，请用两种方法将程序存储器 20F0H 中的常数送入累加器 A 中。

12. 试说明下段程序中每条指令的作用，并分析当指令执行完后，R0 中的内容是什么？

```

MOV  R0, #0A7H

```

```

XCH  A, R0

```

```

SWAP A

```

```

XCH  A, R0

```

13. 请用两种方法实现累加器 A 与寄存器 B 的内容交换。

14. 试编程将片外 RAM 40H 单元的内容与 R1 的内容交换。

15. 已知：A=0C9H，B=8DH，CY=1。

执行指令 ADDC A, B 结果如何？

执行指令 SUBB A, B 结果如何？

16. 试编程将片外 RAM 中 30H 和 31H 单元中的内容相乘，结果存放在 32H 和 33H 单元中，高位存放在 33H 单元中。

17. 试用 3 种方法将累加器 A 中无符号数乘 2。

18. 请分析依次执行下面指令的结果：

```

MOV    30H,    # 0A4H
MOV    A,      # 0D6H
MOV    R0,     # 30H
MOV    R2,     # 47H
ANL    A,      R2
ORL    A,      @R0
SWAP   A
CPL    A
XRL    A,      # 0FFH
ORL    30H,    A

```

19. 求下列指令执行后，累加器 A 及 PSW 中进位位 CY、奇偶位 P 和溢出位 OV 的值。

(1) 当 A = 5BH 时 ; ADD A, # 8CH

(2) 当 A = 5BH 时 ; ANL A, # 7AH

(3) 当 A = 5BH 时 ; XRL A, # 7FH

(4) 当 A = 5BH, CY=1 时 ; SUBB A, # 0E8H

20. 把累加器 A 中数据的低 4 位送入外部数据存储器的 2000H 单元。

21. 试说明指令 CJNE @R1, # 7AH, 10H 的作用。若本指令地址为 250H, 其转移地址是多少?

22. 将累加器 A 的内容由 0 递增, 加到 50, 结果放在累加器 A 中。

23. 试说明压栈指令和弹栈指令的作用及执行过程。

24. 下述程序执行后, SP=? A=? B=? 解释每一条指令的作用, 并将其翻译成机器码。

```

ORG 200H
MOV    SP,     # 40H
MOV    A,      # 30H
LCALL  250H
ADD    A,      # 10H
MOV    B,      A
L1: SJMP L1
ORG    250H
MOV    DPTR,   # 20AH
PUSH   DPL
PUSH   DPH
RET

```

25. 用 80C51 单片机的 P1 端口作为输出, 经驱动电路接 8 只发光二极管, 如图 3-17 所示。当输出位是“0”时, 发光二极管点亮, 输出位是“1”时为暗。试分析下述程序执行过程及发光二极管点亮的规律。

```

LP:    MOV     P1, # 7EH
LCALL  DELAY
MOV    P1,     # 0BDH
LCALL  DELAY

```

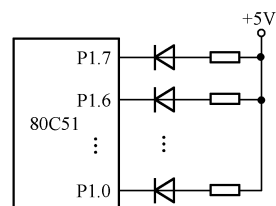


图 3-17 第 25~27 题图

```

MOV     P1,          #0DBH
LCALL   DELAY
MOV     P1,          #0E7H
LCALL   DELAY
MOV     P1,          #0DBH
LCALL   DELAY
MOV     P1,          #0BDH
LCALL   DELAY
SJMP    LP

```

子程序:

```

DELAY: MOV  R2,      #0FAH
        L1: MOV  R3,      #0FAH
        L2: DJNZ R3,      L2
           DJNZ R2,      L1
           RET

```

26. 在上题中, 若系统的晶振频率为 6MHz, 求子程序 DELAY 的延时时间。若想加长或缩短延时时间, 应怎样修改?

27. 根据图 3-17 的线路, 试编制灯亮移位程序, 即 8 个发光二极管每次亮一个, 循环左移, 一个一个地亮, 循环不止。

28. 试编一程序将外部数据存储器 2100H 单元中的高 4 位置 1, 其余位清 0。

29. 试编程将内部数据存储器 40H 单元的第 0 位和第 7 位置 1, 其余位变反。

30. 将 A.5 与 80H 位相与的结果, 通过 P1.4 输出。

31. 请用位操作指令, 求下面逻辑方程:

$$(1) P1.7 = ACC.0 \times (B.0 + P2.1) + \overline{P3.2}$$

$$(2) PSW.5 = P1.3 \times ACC.2 + B.5 \times \overline{P1.1}$$

$$(3) P2.3 = \overline{P1.5} \times B.4 + \overline{ACC.7} \times P1.0$$

第 4 章 汇编语言程序设计

本章主要介绍 80C51 单片机的汇编语言与一些常用的汇编语言程序设计方法，并列举一些具有代表性的汇编语言程序实例，作为读者设计程序的参考。通过本章的学习可以加深对指令系统的了解和掌握。

4.1 概 述

单片机与通用计算机一样，都是按照给定程序，逐条执行指令，完成某项规定的任务。因此，使用单片机，首先必须编写出单片机能执行的程序。通过程序的设计、调试和执行可以进一步熟悉指令，并在一定程度上提高单片机的应用水平。

4.1.1 程序设计语言

单片机能执行的程序，可以用很多种语言来编写，但从语言结构及其与单片机的关系来看，可分为三大类型。

1. 机器语言

用二进制代码“0”和“1”表示指令和数据的最原始的程序设计语言称为机器语言。因为单片机只能识别二进制代码，因而这种语言与单片机的关系最为直接，单片机能够立即识别这种语言，并加以执行，响应速度最快。但对于使用者来说，用机器语言编写程序非常烦琐费时，且不易看懂、不便记忆，容易出错。为了克服这些缺点，从而产生了汇编语言和高级语言。

2. 汇编语言

用助记符表示指令和数据的程序设计语言称为汇编语言。汇编语言是面向机器的，因而不同单片机所使用的汇编语言一般是不同的。助记符是一些英文字符的缩写，例如加法指令的助记符为“ADD”，显然这种语言比机器语言直观、易懂、易用，而且易于记忆，对指令中的操作码和操作数也容易区分。

用汇编语言书写程序比用机器语言方便，但单片机的 CPU 不能直接识别汇编语言，所以单片机不能立即执行，故用汇编语言编写的源程序，在交由单片机执行之前，必须将它翻译成机器语言程序。这一翻译过程称为汇编。简单的程序可以通过人工查指令系统代码对照表翻译，称为“手工汇编”或“人工代真”。这种方法易出错，且麻烦，所以通常采用“机器汇编”。机器汇编是由专门的程序来进行的，这种程序称为汇编程序，不同的指令系统汇编程序不同。汇编通常是通过 PC 实现的，汇编程序可以把用汇编语言编写的源程序翻译成由机器码表示的目标程序，也称目的码程序。与此同时还生成 1 个列表文件（便于查找问题）。汇编完成后再由 PC 把目标程序传输到单片机中。

因为汇编语言和机器语言一样是面向机器的，因而它能把单片机的工作过程刻画得非常精细而又具体。这样可以编制出结构紧凑、运行时间精确的程序。所以这种语言非常适合于实时控制的需要。但是用汇编语言编写和调试程序周期较长，程序可读性较差，因而在对实时性要求不高的情况下，最好使用高级语言。

3. 高级语言

高级语言是一种面向过程而独立于计算机硬件结构的通用计算机语言，比如 C 语言、Pascal、BASIC 等。这些语言是参照数学语言而设计的近似于日常会话的语言。使用者不必了解单片机的内部结构，因而高级语言编写的程序可读性好，模块化程度高，容易移植，比用汇编语言开发容易且省时，有利于系统的维护和升级。目前在单片机中最常用的高级语言是 C 语言。

高级语言的语句功能强，它的一条语句，往往相当于许多条指令，因而翻译后的程序要占用较多存储空间，且不易精确掌握执行时间，故在对时间要求严格的实时控制中一般是不适用的。但在一般的控制中使用高级语言却可以提高编程效率，缩短研制时间。

高级语言也不能被单片机直接识别和执行，也需要翻译为机器语言。这一翻译工作通常称为编译或解释，进行编译或解释的程序称为编译程序或解释程序。

综上所述，三种语言各自的特点是显而易见的。因为本书介绍的是单片机，要想很好地掌握和应用单片机就应该首先学会汇编语言，再学会适用于单片机的 C51 语言。

4.1.2 汇编语言规范

汇编语言是面向机器的程序设计语言，不同种类的单片机，其汇编语言一般是不同的。但是，它们之间所采用的语言规则有很多相似之处。在此，以面向 80C51 的汇编语言为例来说明汇编语言的规范。如果能掌握这一种规范，对其他机型也可以达到举一反三的目的。

1. 汇编语言源程序的格式

汇编语言源程序是由汇编语句（指令语句）构成的。汇编语句由四部分组成，每一部分称为一段。其格式如下：

标号：操作码 操作数 ；注释

在书写汇编语句时，上述各部分应该严格地用定界符加以分离。定界符包括空格符、冒号、分号、逗号等。例如：

标号：操作码 操作数 ；注释

P2: MOV A, #60H ;60H→A

在标号段之后要加冒号“:”，在操作码与操作数之间要有空格间隔。在操作数之间要用逗号“,”将源操作数与目的操作数隔开，在注释段之前要加一分号“;”。

下面分别解释这四段的含义。

- 标号段：标号是用户设定的一个符号，表示存放指令或数据的存储单元地址。

标号是以字母开始的 1~8 个字母或数字串组成的。注意，不能用指令助记符、伪指令或寄存器名来作标号名。

标号是任选的，并不是每条指令或数据存储单元都要有标号，只在需要时，才设标号。如转移指令所要访问的存储单元前面，一般要设置标号，而转移指令的转移地址也用相应的

标号表示。采用标号，便于查询、修改程序，特别是便于转移指令的书写（不必计算具体的转移地址）。

一旦使用某标号定义一地址单元，在程序的其他地方就不能随意修改这个定义，也不能重复定义。

- 操作码段：是指令或伪指令的助记符，用来表示指令的操作性质，在指令中只有它是必不可少的。
- 操作数段：给出的是参加运算（或其他操作）的数据或数据的地址。表示操作数的方法有多种，例如既可用 3 种数制（二、十、十六进制码）表示，也可用标号及表达式来表示。在汇编过程中，这个表达式的值将被计算出来。
如果有 2 个操作数，在这操作数之间要用逗号“,” 分开。
- 注释段：为便于今后阅读和交流，对本指令执行目的和所起作用的说明。在汇编时，对这部分不予理会，它不被译成任何机器码，不影响汇编结果。

2. 汇编语言伪指令

在用汇编程序对汇编语言编写的源程序进行汇编时，有一些控制汇编用的特殊指令。这些指令不属于指令系统，不产生机器代码，因此称为伪指令或汇编指令。利用伪指令告诉汇编程序如何进行汇编，同时它也为人们编程提供了方便。下面介绍几条 80C51 单片机中常用的伪指令。

（1）ORG（Origin）汇编起始指令

本指令的功能是对程序段汇编起始地址进行定位，用来规定汇编程序汇编时，目标程序在程序存储器中存放的起始地址。

它的格式如下：

操作码 操作数
ORG 表达式（exp）

表达式是 16 位的地址值。如 ORG 0060H（前面的 0 可以省略）表示这段程序从 60H 开始。在一个源程序中，可多次使用 ORG 指令，以规定不同程序段的起始位置，地址应从小到大顺序排列，不允许重叠。

（2）END 汇编结束指令

本指令的功能是通知汇编程序结束汇编。这条伪指令放在程序的末尾，表示程序已结束。汇编程序对 END 以后的指令不再汇编。

（3）EQU（EQUATE）赋值指令

本指令的功能是把操作数段中的地址或数据赋值给标号字段中的标号。赋值指令也称等值伪指令。

它的格式如下：

标号（字符名称） EQU 表达式

它的作用是使标号与表达式等值，表达式一般用数或汇编符号表示。

例 1 Q1 EQU R3 ; R3 与 Q1 等值

则 MOV A, Q1 指令与 MOV A, R3 指令结果相同。

例 2 Q1 EQU 3BH ; Q1 的值为 3BH

 L2 EQU 05CBCH

 MOV A, Q1 ; 3BH→A

LCALL L2 ;5CBCH→PC

(4) DB (Define Byte) 定义字节指令

本指令的功能是从指定单元开始定义（存储）若干字节的数值或 ASCII 码字符，常用于定义数据常数表。在每个数或字符之间要用逗号“,” 分开，在表示 ASCII 字符时需要用 ' ' 单引号。

它的格式如下：

操作码	操作数
DB	字节常数或 ASCII 码字符

例 3 ORG 0200H

DB 0A3H, 73, 'A', 'B'

DB 46H, 09

则 (200H)=0A3H(201H)=49H(202H)=41H

(203H)=42H (204H)=046H(205H)=09

在汇编语言中当用引号表示一个字母时，则汇编程序会自动把它翻译为相应的十六进制数，如例 3 所示。ASCII 码字符表见附录 C。

(5) DW (Define Word) 定义字指令

本指令的功能是从指定单元开始定义（或存储）若干个字的数据或 ASCII 码字符。它的格式如下：

操作码	操作数
DW	字常数或 ASCII 字符

例 4 ORG 5200H

DW 3456H, 13BH, 12

则 (5200H)=34H (5201H)=56H

(5202H)=01 (5203H)=3BH

(5204H)=00 (5205H)=0CH

(6) BIT 定义位地址指令

本指令的功能是把位地址赋予所规定的字符名称。

它的格式如下：

标号	操作码	操作数
字符名称	BIT	位地址

例 5

A3	BIT	P1.4
QB	BIT	P3.2

则汇编后，位地址 P1.4, P3.2 分别赋给变量 A3 和 QB。

(7) DATA 定义标号数值伪指令

本指令的功能是给标号段中的标号赋以数值，通常用于定义数据地址。

它的格式如下：

标号	操作码	操作数
字符名称	DATA	表达式

例 6 L3 DATA 4560H

汇编后 L3 的值为 4560H。

DATA 与 EQU 的区别主要有如下几点：

- DATA 定义的标识符汇编时作为标号登记在符号表中，所以可以先使用后定义，而

EQU 定义的标志符必须先定义后使用，因后者不登记在符号表中；

- EQU 指令可以给一个汇编符号赋字符名，而 DATA 指令只能给数据或一个表达式（可求值的）赋字符名。

（8）DS 定义存储空间指令

本指令的功能是从指定地址开始保留 DS 之后表达式的值所规定的若干个存储单元以备源程序使用。

它的格式如下：

	操作码	操作数
	DS	表达式

例 7 ORG 280H
 DS 20H
 DB 7BH, 3AH

汇编后，从 280H 开始保留 32 个单元，然后从 2A1H 开始按 DB 指令给存储器赋值，即 (2A1H)=7BH, (2A2H)=3AH。注意，汇编时对 280H 开始的 32 个单元不赋值。

在编写汇编语言源程序时，必须严格按照汇编语言的规范书写。在伪指令中，ORG 和 END 最重要，不可少。

4.1.3 汇编语言程序设计步骤

要想使单片机完成某一具体的工作任务，必须按序执行一条条指令。这种按工作要求编排指令序列的过程称程序设计。

使用汇编语言作为程序设计语言的编程步骤与高级语言编程步骤类似，但又略有差异。其程序设计步骤大致可分为以下几步：

- ① 明确工作任务要达到的工作目的、技术指标等；
- ② 分析任务要求，确定解决问题的计算方法和工作步骤；
- ③ 画工作流程图（其图形的符号规定均同于高级语言流程图，在此不赘述）；
- ④ 分配工作寄存器和内存工作单元，确定程序与数据区存放地址；
- ⑤ 按流程图编写源程序；
- ⑥ 上机调试、修改及最后确定源程序。

由上述步骤可以看出，在用汇编语言进行程序设计时，主要方法和思路与高级语言相同，主要不同点，也是非常重要的一点是第 4 点，而这也正是汇编语言面向机器的特点。即在设计程序时还要考虑程序与数据的存放地址，在使用内存单元和工作寄存器时注意它们相互之间不能发生冲突。

进行程序设计时，必须根据实际问题 and 所使用的计算机的特点来确定算法，然后按照尽可能使程序简短和缩短运行时间两个原则编写程序。编程技巧需经大量实践后逐渐地加以提高。

4.2 汇编语言程序设计举例

在用汇编语言进行程序设计时，它的程序结构与高级语言也是相似的，一般可分为顺序程序、循环程序、分支程序、查表程序及子程序这五大基本结构。下面将结合 80C51 单片机

的特点，介绍一些常用的程序设计方法。

4.2.1 顺序程序设计

顺序结构程序是一种最简单、最基本的程序（也称为简单程序）。它的特点是按程序编写的顺序依次执行，程序流向不变。这类程序是所有复杂程序的基础或某个组成部分。

顺序程序虽然并不难编写，但要设计出高质量的程序还是需要掌握一定的技巧。为此，需要熟悉指令系统，正确地选择指令，以达到提高程序执行效率、减少程序长度、最大限度地优化程序的目的。下面举例说明。

例 1 将 R3 中的两个 BCD 码拆开并变成 ASCII 码，存入 51H、52H 单元。

解：在此采用把 BCD 数除以 10H 的方法，除后相当于把此数右移了 4 位，刚好把两个 BCD 码分别移到 A、B 的低 4 位。由于 ASCII 码的 0~9 为 30H~39H，因此再各自与 30H 相“或”即变为 ASCII 码。

源程序如下：

```

                ORG      0000H
                LJMP     MAIN
                .....
MAIN:           ORG      30H                ; 主程序起始地址
                MOV      A,      R3
                MOV      B,      # 10H      ; 用 10H 作除数
                DIV      AB
                ORL      B,      # 30H      ; 低 4 位 BCD 码变为 ASCII 码
                MOV      52H,    B
                ORL      A,      #30H      ; 高 4 位 BCD 码变为 ASCII 码
                MOV      51H,    A
                SJMP     $                ; 循环等待
                END
```

这是一个简单的但是格式编写完整的小程序。一开始的无条件跳转指令 LJMP 是所有程序中都必须写的，当然也可以根据转移范围用 AJMP 或 SJMP 无条件跳转指令。主程序起始地址选择为 30H 是因为 03H~2BH 之间是系统保留作为中断入口地址的一些单元，主程序应该避开这个区域。最后的指令 SJMP 后面加\$表示在此指令处循环等待。80C51 系列指令系统没有专门的暂停和程序结束指令，通常采用此方法表示等待，或者程序结束。伪指令 END 仅表示汇编过程到此结束，不表示程序执行是否结束。为节约篇幅，以后有类似的情况都不再重复写了，多数情况只写出相关的程序段落。

上述例题如果采用把高、低 4 位 BCD 码分别交换出来，再各自变为 ASCII 码的方法，则占用字节数比上述方法少，而执行时间也比上述方法短，在程序较大时要考虑这些细节。

4.2.2 循环程序设计

在很多程序中会遇到需多次重复执行某段程序的情况，这时可把这段程序设计为循环结构程序（通常称其为循环体），这种结构可大大缩短程序。这是最常用的一种程序结构。

1. 循环程序概述

正确设计循环程序的关键是正确设计好以下部分。

(1) 循环初值

循环初值是指循环过程中所用工作单元的初始值。例如循环次数计数器、地址指针初值和数存放的单元初值等。在循环开始前首先要设置循环初值。

(2) 循环体

重复执行的程序段部分，包括主要的计算或操作任务。

(3) 循环控制部分

用于控制循环的执行和结束。在循环初值中已经给出了循环结束条件，即循环次数初值。循环程序每执行一次，都检查结束条件，当条件不满足时，则修改地址指针和控制变量；当条件满足时，则停止循环。

除此，在多重循环程序中，注意，不允许循环体互相交叉，也不允许从循环程序的外部跳入循环程序的内部。

2. 循环程序举例

例 2 设计一个软件延时程序，延时时间为 20ms，80C51 单片机使用的晶振为 12MHz。

解：延时程序的延时时间主要与两个因素有关，一个是所用晶振，一个是延时程序中的循环次数。一旦晶振确定之后，则主要是如何设计与计算需要给定的延时循环次数。在本题中已知晶振为 12MHz，则可知一个机器周期为 1μs，那么可预计采用单重循环是有可能实现 1ms 的延时的。现根据题意编写源程序如下：

周期数

1	MOV	R0, # 14H	; 毫秒数→R0
1 DL2:	MOV	R1, # MT	; 1ms 延时的预定值 MT→R1
1 DL1:	NOP		
1	NOP		
2	DJNZ	R1, DL1	; 1ms 延时循环
2	DJNZ	R0, DL2	; 毫秒数减 1, 不等于 0, 继续循环, 等于 0 结束

该延时程序是一个双重循环程序。外循环的初值已经设为 20，内循环中的 NOP 只是用于调整延时时间，内循环的初值 MT 尚需计算。因为各条指令的执行时间是确定的，需延时的总时间也已知，因而 MT 可如下计算：

$$(1+1+2) \times 1 \times MT = 1000(\mu s)$$

$$MT = 250 = FAH$$

因此，用 FAH 代替上述程序中的 MT，则该程序执行后，能实现 20ms 的延时。

若考虑其他指令的执行时间，则该段延时程序的精确延时时间应如下计算：

$$1\mu s \times 1 + \{(1+2) \times 1\mu s + (1+1+2) \times 1\mu s \times 250\} \times 20 = 20061\mu s$$

若需要延时更长时间，可采用更多重的循环，如 1s 延时可用 3 重循环，而 7 重循环可延时 1 年等。

采用软件实现延时功能虽然方法简单，不需要占用硬件资源，但这种方法会占用 CPU 的工作时间并且定时精度不如一般的硬件定时器高，通常用在要求不高的定时场合。

例 3 有一个用 ASCII 码值表示的字符串，其长度（即字符总个数）在 30H 单元，其首地址为 31H 单元。求出这个字符串中字符 A 的个数，并存入 40H 单元。

解：根据题意，用 R0 作为字符 A 的计数器，先设字符串中字符 A 的个数为零，此即为初值。已知字符 A 的 ASCII 码值为 41H，然后逐个取出队列中的字符与 A 的 ASCII 码值相比较，如果不等则继续比较，相等则计数值加 1 后再继续比较。当所有字符均比较完之后，则可得到字符 A 的个数。

源程序如下：

```

MOV    R0,    # 0           ; 清 R0 作为初始值
MOV    R2,    30H           ; 取字符个数初值（循环控制次数）
MOV    R1,    # 31H         ; 设字符存放区首地址
LP:    MOV    A,    @ R1     ; 取字符
       CJNE   A,    #41H, LP1 ; 与字符 A 比较
       INC    R0            ; 等于加 1 后继续
LP1:   INC    R1            ; 修改地址指针
       DJNZ   R2,    LP     ; 依次重复比较，直至 R2=0
       MOV    40H,    R0    ; A 字符个数存入 40H 单元

```

注意在指令中也可以直接把#41H用‘A’表示。

4.2.3 分支程序设计

分支结构程序的特点是程序中含有转移指令，分支结构程序可以根据程序要求无条件或有条件地改变程序执行顺序，选择程序流向。

1. 分支程序设计综述

编写分支结构程序重点在于正确使用转移指令。转移指令有 3 种，无条件转移、条件转移和散转。由这三类指令形成的分支程序有以下特点。

① 无条件转移：它的程序转移方向是设计者事先安排的，与已执行程序的结果无关，使用时只需给出正确的转移目标地址或偏移量即可。

② 条件转移：它是根据已执行程序对标志位、对累加器或对内部 RAM 某位的影响结果，来决定程序的走向，形成各种分支。在编写有条件转移语句时要特别注意以下两点。

- 在使用条件转移指令形成分支前，一定要安排可供条件转移指令进行判别的条件，以便为测试准备条件。
- 要正确选定所用的转移条件和转移目标地址。

③ 散转：它是根据某种已输入的或运算的结果，使程序转向各个处理程序中去，80C51 机具有一条专门的散转指令，可以使它较方便地实现散转功能。

2. 无条件/条件转移程序

这是分支程序中最常见的一类。在编写条件转移类程序时需要正确选定转移条件。下面举例说明。

例 4 请编写计算下述函数式的程序：

$$Y = \begin{cases} 90 & , X > 80 \\ X + 13 & , 80 \geq X \geq 20 \\ X - 2 & , X < 20 \end{cases}$$

解：显然对本函数式需要采用分支程序设计，设变量 X 存放在 R1 中，结果 Y 存入 40H

单元, R0 用做中间寄存器。

源程序如下:

```
      MOV    A,      R1          ; 把 X 装入 A
      CJNE   A,      # 20, L1
L1:   JC     L2          ; X<20 转 L2
      MOV    R0,     # 90        ; 先假设 X>80
      CJNE   A,      # 81, L3    ; 与 81 比
L3:   JNC    L4          ; X>80 转 L4
      MOV    A,      R1
      ADD    A,      # 0DH       ; 80≥X≥20, Y=X+13
      MOV    R0,     A
      SJMP   L4
L2:   MOV    A,      R1
      CLR    C
      SUBB   A,      # 02        ; X<20, Y=X-2
      MOV    R0,     A
L4:   MOV    40H,    R0
      SJMP   $
```

本程序既采用了无条件转移指令,又采用了条件转移指令,在判别 $A < 20$ 和 $A > 80$ 时采用 CJNE 和 JC 以及 CJNE、JNC 两条指令相结合的方法。

从以上条件分支程序可见,它与简单程序的区别在于:分支程序存在两个或两个以上的结果。这要根据给定的条件进行判断,以得到某一个结果。这样,就要用到比较指令、测试指令以及无条件/条件转移指令。条件分支程序设计的技巧,就在于正确地使用这些指令。

3. 散转程序设计

在 80C51 指令系统中有一条间接转移指令,也称为散转指令。它的功能是把 16 位数据指针 DPTR 的内容与累加器 A 中的 8 位无符号数相加,形成散转的目的地址,装入程序计数器 PC,使程序转入相应的分支程序中去。这条指令非常适用于设计散转程序,散转程序是根据运算结果从多个分支程序中选择其中之一。通常采用的方法是固定 DPTR 的内容,然后根据 A 中的内容进行散转。

下面举例说明。

例 5 某段程序的运算结果在 R3 中,要求根据 R3 的内容,分别转向 0~255 个操作程序,即按如下规定进行散转。

```
当 R3=0,      转入 OPR0 操作程序
当 R3=1,      转入 OPR1 操作程序
  ⋮           ⋮
当 R3=255,    转入 OPR255 操作程序
```

解: 根据本例题的要求,可以先用无条件直接转移指令“LJMP”指令按序组成一个转移指令地址表,设转移表首地址标号为 TAB1,再将其装入数据指针 DPTR 中,然后将 R3 中的内容装入累加器 A 经运算后作为变址值,最后执行“JMP @A+DPTR”指令实现散转。由于转移表是由 3 字节长转移指令“LJMP”组成的,所以累加器 A 中的变址值必须乘 3。乘 3 后的值有可能超过 256,所以应将修正值的高字节先加在数据指针高位字节 DPH 上,然后将低字节送入累加器 A 中。

程序清单如下。

```

JMP1:  MOV    DPTR,  # TAB1      ; 转移表首地址送数据指针
        MOV    A,      R3        ; 变址值送 A
        MOV    B,      # 3
        MUL    AB                ; 变址值乘 3 后变为修正值
        MOV    R1,     A          ; 修正值的低字节暂存 R1 中
        MOV    A,      B          ; 修正值的高字节送 A
        ADD    A        DPH       ; 修正值的高字节与 DPH 相加
        MOV    DPH,    A
        MOV    A,      R1        ; 修正值的低字节送 A
        JMP    @A+DPTR          ; 转向形成的散转地址入口
        ⋮

TAB1:   LJMP   OPR0              ; 直接转移地址表
        LJMP   OPR1
        ⋮
        LJMP   OPR255

```

本程序中，转移表由 3 字节长转移指令“LJMP”组成转移指令表，因而散转地址的范围可以是 64KB 内的任何地址。在此，转移表也可以用短转移指令“AJMP”组成，此时各转移指令地址依次相差两字节，所以累加器 A 中变址值必须做乘 2 修正。在程序上将做如下修改。

```

        ⋮
        MOV    A,      R3        ; 变址值送 A
        ADD    A        R3       ; R3×2 送 A
        JNC    JNP2             ; 判断是否有进位
        INC    DPH              ; 有进位则高字节加 1
JNP2:   JMP    @A+DPTR          ; 转向形成的散转地址入口
        ⋮

```

这种方法虽然编程要简单一些，但如果使用“AJMP”指令，就限制了转移的入口地址 OPR0, OPR1, …, OPR255 必须和散转表首 TAB1 位于同一个 2KB 空间范围内，使用时一定要注意。

上述方法的局限性表现在散转点不得超过 256，这是因为工作寄存器 R3 为单字节。

为了克服上述局限性，可采用双字节的工作寄存器存放散转点，采用对 DPTR 进行加法运算的方法，直接修改 DPTR，然后再用“JMP @A+DPTR”指令来执行散转。

下面举例说明。

例 6 某段程序的运算结果在 R6、R7 中，其中 R6 存放结果的高 8 位字节，R7 存放结果的低 8 位字节，要求根据 R6、R7 中的内容，分别转向 0~n 个操作程序，即按如下规定进行散转。

```

    当  R6R7 = 0,      转入 OPR0 操作程序
    当  R6R7 = 1,      转入 OPR1 操作程序
        ⋮
    当  R6R7 = n,      转入 OPRn 操作程序

```

设 $0 \leq n < 4FFFH$ ，即 $R6 < 4FH$ 。

解：本例题也采用无条件直接转移指令“LJMP”组成转移指令地址表，设转移表首地址

标号为 TAB2，再将其装入数据指针 DPTR 中。散转点在 R6、R7 中，应先乘 3 然后再计算查表偏移量。因为 R6、R7 占两字节，且 $R6 < 4FH$ ，所以散转点高 8 位乘 3 的结果没有进位，可直接加在数据指针 DPTR 高 8 位上。散转点低 8 位乘 3 后，把低 8 位（在 A 中）加在 DPL 上，高 8 位（在 B 中）再添加到 DPH 中。最后执行“`JMP @A+DPTR`”指令实现散转。

程序清单如下。

```

JUMP2: MOV    DPTR,    # TAB2        ; 跳转表首地址送数据指针
        MOV     A,      R6           ; 取散转点高 8 位
        MOV     B,      # 03H
        MUL     AB                ;  $R6 \times 3 \rightarrow BA$ 
        ADD     A,       DPH
        MOV     DPH,     A           ;  $R6 \times 3 + DPH \rightarrow DPH$ 
        MOV     A,      R7           ; 取散转点低 8 位
        MOV     B,      # 03H
        MUL     AB                ;  $R7 \times 3 \rightarrow BA$ 
        XCH     A,       B
        ADD     A,       DPH         ;  $R7 \times 3$  高位加到 DPH 上
        MOV     DPH,     A
        XCH     A,       B           ;  $R7 \times 3$  低位在 A 中
        JMP     @A+DPTR             ; 散转
        :
TAB2:   LJMP    OPR0
        LJMP    OPR1
        :
        LJMP    OPRn

```

可以实现散转的方法很多，不限于上述方法。

4.2.4 查表程序设计

查表法，就是把已知对应关系的可能范围的函数值按一定规律编成表格存放在单片机的程序存储器（一般为只读存储器）中。当用户程序中需要用到这些函数值时，直接按编排好的索引值（或程序号）寻找答案。即根据变量 X ，寻找对应的 Y 值。这种方法节省了运算步骤，使程序更简便、执行速度更快。在控制应用场合或在智能化仪器仪表中，经常使用查表法。这种方法唯一的不足是要占用较多的存储单元，但随着存储器价格的大幅度下降，使查表法的应用越来越广泛。

1. 查表程序综述

为了便于实现查表功能，在 80C51 汇编语言中设置了两条查表指令。

```

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

```

这 2 条指令的共同点是从程序存储器读取数据，并且 DPTR 和 PC 都是基址寄存器，可以用来指示表格首地址。累加器 A 的内容为查表值与表格首地址之间的无符号偏移量。它限制了表格的长度一般在 256 字节之内。

这 2 条指令的主要区别如下。

第一条查表指令采用 DPTR 作为数据表格的首地址指针，表格可设置在程序存储器的 64KB 的任何区域。其查表过程比较简单。查表前把数据表格起始地址存入 DPTR，然后把所查表的索引值送入累加器 A 中，最后使用“MOVC A, @A+DPTR”指令完成查表。

第二条查表指令采用 PC 作为数据表格的首地址指针，表格只能设置在该指令之后的 256 字节范围之内，操作过程与第一条不同，其步骤可分为 3 步。

- ① 使用传送指令把所查数据的索引值送入累加器 A。
- ② 用“ADD A, #data”指令对累加器 A 进行修正。data 值由下式确定：

PC+data=数据表首地址

其中，PC 是“MOVC A, @A+PC”的下一条指令地址。因此，data 值实际等于查表指令和数据表格之间的字节数，此值必须小于 256。

- ③ 用“MOVC A, @A+PC”指令完成查表。

由于表格范围受到限制，并且还要仔细计算 data 值，因此这种方法只限于数量不大的简单表格，它的优点是可使程序比较紧凑。

为了便于查表，要求表中的数或符号按照便于查找的次序排列，并将它存放在从指定的首地址（或称基地址）开始的存储单元。函数值在表中的序号即索引值应该和函数值有直接的对应关系。函数值的存放地址即等于首地址加上索引值。

变量 X 可以是规则变量也可以是非规则变量，但不论 X 怎样变化，X 的值与表格中的 Y 值一定是一一对应的。Y 值可以是单字节、双字节或 3 字节等，但所有的 Y 值必须具有相同的字节数。这样的表格具有规律性，便于编制查表程序。

2. 查表程序设计举例

例 7 设计一个将十六进制数转换成 ASCII 码的子程序。设十六进制数存放在 R0 中的低 4 位，要求将转换后的 ASCII 码送回到 R0 中。

解：已知 0~9 的 ASCII 码为 30H~39H，AH~FH 的 ASCII 码为 41H~46H。按题意，表中所有的值都是单字节，表格长度为 16 字节。

入口参数：R0，待转换的十六进制数

出口参数：R0，转换后的 ASCII 码

程序如下：

地址	机器码	指令	说明
		ORG 60H	
60	E8	MOV A, R0	
61	54 0F	ANL A, #0FH	; 保留低 4 位
63	24 02	ADD A, #02	; 变址调整
65	83	MOVC A, @A+PC	; 查表获取 ASCII 码值
66	F8	MOV R0, A	
67	22	RET	
68	30 31 32	DB 30H, 31H, 32H	
6B	33 34 35	DB 33H, 34H, 35H	
6E	36 37 38	DB 36H, 37H, 38H	
71	39 41 42	DB 39H, 41H, 42H	
74	43 44 45 46	DB 43H, 44H, 45H, 46H	

由本程序的机器码可以看出“MOVC A, @A+PC”指令与表格首地址（68H）相隔两个字节，即其偏移量为 2，故变址调整值为 2。

例如，当 R0 中原值为 08，则执行 `MOVC A, @A+PC` 指令后 PC 内容为 66H，加 2 后，再加 A 值后即得所寻数值 38H 的地址为 70H。

上例中表格长度不能超过 256，且 Y 值为单字节，但在实际应用中表格长度可能超过 256，Y 值可能为多字节。此时就需要用“`MOVC A, @A+DPTR`”指令，查表参数也需要用 2 字节表示。下面举例说明。

例 8 已知查表参数 X 的值为 0~2000，相对应的 Y 值均为 2 字节。设 X 的值存放在 R1、R2 中，其中 R1 中为高位字节，R2 中为低位字节。要求把从表格中查出的对应数值存放到 R3、R4 中，其中 R3 中存放高位字节，R4 中存放低位字节。

X 与 Y 的对应关系如下：

X: 0 1 2 2000
Y: 022BH 013AH CD56H8656H

解：在此程序中因 Y 值为双字节，所以需要把 X 值乘 2 作为查表偏移量。

因为表格长度超过 256，在此采用使累加器 A 清零，改变数据指针 DPTR 的方法寻找对应的入口地址。

按题意编写程序如下：

```

MOV    DPTR,    # TAB        ; 表格首地址送 DPTR
MOV    A,       R2           ; 查表参数低位字节送 A
CLR    C         ; 清进位 CY
RLC    A         ; 带进位左移一位，即 R2×2
XCH    A,       R1           ; A 与 R1 的值交换，R1 中暂存低字节偏移量
RLC    A         ; 带进位左移一位，即 R1×2
XCH    A,       R1           ; A 与 R1 的值交换，R1 中暂存高字节偏移量
ADD    A,       DPL          ; 查表参数低字节加 DPL
MOV    DPL,     A           ; 形成查表的低字节地址
MOV    A,       DPH          ; 查表参数高字节加 DPH
ADDC   A,       R1           ; 形成查表的高字节地址
MOV    DPH,     A
CLR    A         ; 清 A
MOVC   A,       @A+DPTR      ; 从表格中取高 8 位字节
MOV    R3,      A           ; 高 8 位字节送 R3
CLR    A         ; 清 A
INC    DPTR      ; 指向表格低 8 位字节的地址
MOVC   A,       @A+DPTR      ; 从表格中取低 8 位字节
MOV    R4,      A           ; 低 8 位字节送 R4
RET                                ; 返回
TAB:   DB      02,      2BH    ; X 为“0”值时，对应的 Y 值的高、低字节
       DB      01,      3AH    ; X 为“1”值时，对应的 Y 值的高、低字节
       DB      CDH,     56H    ; X 为“2”值时，对应的 Y 值的高、低字节
       :
       DB      86H,     56H    ; X 为“2000”值时，对应的 Y 值的高、低字节

```

4.2.5 子程序设计

在实际应用中，通常把多次使用的程序段，例如多字节的加、减、乘、除、代码转换、

字符处理等，按一定结构编好，存放在内存中；当需要时，程序可以去调用这些独立的程序段。将这种可以被调用的程序段称为子程序。调用子程序的程序称为主程序。使用子程序的过程，称为调用子程序。子程序执行完后返回主程序的过程，称为子程序返回。汇编语言中的子程序与高级语言相同，也具有嵌套（或称多重转子）功能，子程序嵌套次数从理论上说是无限的，但实际上，由于受堆栈深度的限制，嵌套次数是有限的。

1. 子程序设计综述

子程序是一种具有某种功能的程序段，其资源需要为所有调用程序共享。因此，子程序在功能上应具有通用性，在结构上应具有独立性。它在结构上与一般程序的主要区别是在子程序末尾有一条子程序返回指令（RET）。

编写子程序的注意事项如下。

- 子程序第一条指令的地址，通常称为子程序的入口地址或首地址。为编程方便，要给每个子程序赋一个名字，实际上是一个入口地址的标号。
- 要能正确地传递参数。首先要有入口条件，说明进入子程序时，它所要处理的数据如何得到（例如是把它放在 ACC 中还是放在某工作寄存器中等）。另外，要有出口条件，即处理的结果是如何存放的。
- 注意保护现场和恢复现场。在执行子程序时，可能要使用累加器或某些工作寄存器。而在调用子程序之前，这些寄存器中可能存放有主程序的中间结果，这些中间结果是不允许被破坏的。因而在子程序使用累加器和这些工作寄存器之前，要将其中的内容保存起来，即保护现场。当子程序执行完，即将返回主程序之前，再将这些内容取出，送回到累加器或原来的工作寄存器中，这一过程称为恢复现场。
保护和恢复现场通常用堆栈来进行。在需要保护现场的情况，编写子程序时，要在子程序的开始使用压栈指令，把需要保护的寄存器内容压入堆栈。当子程序执行完，在返回指令前使用弹出指令，把堆栈中保护的内容弹出到原来的寄存器，这样即恢复了现场。在所采用的工作寄存器超过 2 个时，可以通过选择不同工作寄存器组的方法达到保护的目的，这样可节省入栈/出栈操作指令，还可以节省堆栈空间。
- 为了使子程序具有一定的通用性，子程序中的操作对象，应尽量用地址或寄存器形式，而不用立即数形式。另外，子程序中如果含有转移指令，应尽量用相对转移指令，以便它不管放在内存的哪个区域，都能正确执行。

2. 子程序的调用与返回

主程序调用子程序是通过子程序调用指令 LCALL add16 和 ACALL add11 来实现的。前者称为长调用指令，指令的操作数给出 16 位的子程序首地址；后者为绝对（短）调用指令，它的操作数提供子程序的低 11 位入口地址，此地址与程序计数器 PC 的高 5 位并在一起，构成 16 位的转移地址（即子程序入口地址）。

子程序调用指令的功能是将 PC 中的内容（调用指令的下一条指令地址，称断点）压入堆栈（即保护断点），然后将调用地址送入 PC，使程序转入子程序的入口地址。

子程序的返回是通过返回指令 RET 实现的。这条指令的功能是将堆栈中存放的返回地址（即断点）弹出堆栈，送回到 PC 去，使程序继续从断点处执行。

主程序在调用子程序时要注意的问题：

- 在主程序中，要安排相应指令，满足子程序的入口条件，提供子程序的入口数据；
- 在主程序中，不希望被子程序更改内容的寄存器，也可在调用前在主程序中安排压栈指令保护现场，子程序返回后再安排弹出指令恢复现场；
- 在主程序中，安排相应的指令，处理子程序提供的出口数据；
- 在需要保护现场的程序中，要正确地设置堆栈指针。

3. 子程序设计举例

由于应用子程序给程序设计带来很多方便，在实际程序中，特别是在监控程序中，经常把一些常用的运算、操作等编成子程序，如数码转换、延时、拆字等。这样给用户提供了方便。

下面通过具体例子说明子程序的设计和调用。

例 9 有两个 ASCII 码值表示的字符串，2 个字符串的首地址分别为 50H 和 70H，每个字符串的第一字节都存放字符串长度。求出这 2 个字符串中字符 A 的个数，并将其和存入 4FH 单元。

解：本例采用分别求出两个字符串字符 A 的个数，然后求和的方法，求字符 A 个数的过程可采用子程序。子程序的入口条件是字符串首地址，返回参数即为个数值，放在 A 中。

下面分别列出主程序和子程序：

主程序

```

MOV    R1,    # 50H        ; 置入口条件参数
ACALL  ZF            ; 调求字符 A 个数子程序
MOV    40H,    R0        ; 第一个数据块的 A 个数暂存 40H
MOV    R1,    # 70H        ; 置入口条件参数
ACALL  ZF            ; 调求字符 A 个数子程序
MOV    A,      R0
ADD    A,      40H        ; 两个字符 A 个数相加
MOV    4FH,    A        ; 把和送入 4FH
SJMP   $

; 子程序入口参数: R1 为字符串首地址
; 子程序出口参数: R0 为字符串中 A 的个数
ZF:    MOV    R0,    #0        ; 清 R0 作为初始值
        MOV    A,      @R1        ; 取字符个数初值
        MOV    R2,    A        ; 字符个数初值送 R2
        MOV    R1,    # 31H        ; 设字符存放区首地址
LP:    MOV    A,      @R1        ; 取字符
        CJNE   A,      #41H, LP1    ; 与字符 A 比较
        INC    R0        ; 等于加 1 后继续
LP1:   INC    R1        ; 修改地址指针
        DJNZ   R2,      LP        ; 依次重复比较, 直至 R2=0
        RET                ; 返回

```

例 10 在图 4-1 所示电路中，AT89S51 的 P1 口各位分别与 8 个发光二极管相接，当 P1 口为低电平时发光二极管可被点亮。P3.1 与 P3.2 各通过开关 S1、S2 与地相接。当开关闭合时 P3.1 与 P3.2 端口为低电平。设单片机采用的晶振为 6MHz，编制一个控制发光二极管发光方式的程序。要求当 S1 闭合时，发光二极管发光方式为：二极管从第 0 位开始发光，延时 1s 后，第 0 位二极管灭，第 1 位开始发光；延时 1 秒后，第 1 位二极管灭，第 2 位开始发光；延时 1s 后……直至第 7 位开始发光。

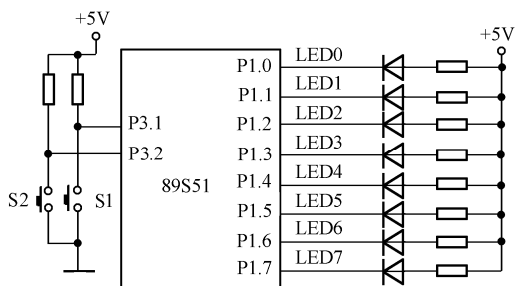


图 4-1 LED 闪烁线路

当 S2 闭合时，发光二极管发光方式为：二极管从第 0 位开始发光，延时 1s 后，第 1 位开始发光；延时 1s 后，第 2 位开始发光；延时 1s 后……直至 8 位全部发光。

解：根据本程序的要求，在二极管的第一种发光方式下，P1 口的输出值应该如下变化：

11111110→延时 1s→11111101→延时 1s→11111011→延时 1s→11110111→延时 1s→11101111→延时 1s→11011111→延时 1s→10111111→延时 1s→01111111→延时 1s→返回主程序。

在二极管的第二种发光方式下，P1 口的输出值应该如下变化：

11111110→延时 1s→11111100→延时 1s→11111000→延时 1s→11110000→延时 1s→11100000→延时 1s→11000000→延时 1s→10000000→延时 1s→00000000→延时 1s→返回主程序。

编程如下：

```

S1      EQU P3.1
S2      EQU P3.2
ORG     0000H
LJMP    MAIN
.....

MAIN:    MOV     P3, #0FFH    ; 设 P3 口为输入
MAIN2:   JB      S1, FF1      ; 检查是否按过 S1 键
        LCALL    DELAY10ms    ; 延时 10ms 去键抖动
        JB      S1, FF1      ; 如果又变为 1，说明刚才才是干扰信号
        LCALL    FF11         ; 如果仍然为 0，说明确实按过 S1 键，则调第一种发光方式子程序

FF1:     JB      S2, MAIN2     ; 检查是否按过 S2 键
        LCALL    DELAY10ms    ; 延时 10ms 去键抖动
        JB      S2, MAIN2     ; 如果又变为 1，说明刚才才是干扰信号
        LCALL    FF22         ; 如果仍然为 0，说明确实按过 S2 键，则调第二种发光方式子程序

        SJMP     MAIN2        ; 反复循环

FF11:    MOV     A, #0FEH      ; 第一种发光方式初值
L1:      MOV     P1, A          ; A 值送 P1 口
        LCALL    DL1S          ; 1s 延时
        JNB     ACC.7, MA1     ; A 值的第 7 位是否为 0
        RL      A              ; A 中数据循环左移一位
        SJMP     L1            ; 未完继续

MA1:     RET

FF22:    MOV     A, #0FEH      ; 第二种发光方式初值
L2:      MOV     P1, A          ; A 值送 P1 口
        LCALL    DL1S          ; 1s 延时
        JZ      MA2            ; A 值是否为 0
        RL      A              ; A 中数据循环左移一位
        ANL     A, P1          ; P1 口当前值与移位后值相与

```

```

                SJMP     L2
MA2:            RET
DELAY10ms:     MOV      R0, # 10
DL2:            MOV      R1, # 125      ; 1ms 延时的预定值
DL1:            NOP
                NOP
                DJNZ     R1, DL1        ; 延时循环
                DJNZ     R0, DL2
                RET
DL1S:           MOV      R3, # 100      ; 1s 延时的预定值
DL3:            LCALL    DELAY10ms     ; 延时循环
                DJNZ     R3, DL3
                RET
                END

```

把子程序结构应用到编写复杂程序中去，就可以把一个复杂的程序分割成很多独立的，关联较少的功能模块，通常称为模块化结构。这种方式不但结构清楚、易读、节省内存，而且也易于调试，是大程序中经常采用的编程方式。

思考与练习

1. 编程将片内 40H~60H 单元中内容送到以 3000H 为首的存储区中。
2. 编写计算下列算式的程序。
 - (1) $23H + 45H + ABH + 03H =$
 - (2) $CDH + 15H - 38H - 46H =$
 - (3) $1234H + 8347H =$
 - (4) $AB123H - 43ADCH =$
3. 设有两个 4 位 BCD 码，分别存放在 23H、22H 单元和 33H、32H 单元中，求它们的和，并送入 43H、42H 单元中去（以上单元均为低位在低字节，高位在高字节）。
4. 用 P1 口作为数据读入口，为了读取稳定的值，要求连续读 8 次后取平均值。
5. 编程计算片内 RAM 区 50H~59H 单元中数的算术平均值，结果存放在 5AH 中。
6. 编写计算下式的程序，设乘积和平方结果均小于 255。 a 、 b 值分别存在片外 3001H 和 3002H 单元中，结果存于片外 3000H 单元中。

$$\begin{aligned}
 (1) \quad Y &= \begin{cases} 25 & a = b \\ a \times b & a < b \\ a \div b & a > b \end{cases} \\
 (2) \quad Y &= \begin{cases} (a+b)^2 + 10 & (a+b)^2 < 10 \\ (a+b)^2 & (a+b)^2 = 10 \\ (a+b)^2 - 10 & (a+b)^2 > 10 \end{cases}
 \end{aligned}$$

7. 设有两个长度均为 15 的数组，分别存放在以 200H 和 210H 为首地址的存储区中，试编程求其对应项之和，结果存放以 220H 为首地址的存储区中。

8. 设有 100 个有符号数，连续存放在以 2000H 为首地址的存储区中，试编程统计其中正数、负数、零的个数。

9. 请将片外数据存储器地址为 1000H~1030H 的数据块，全部搬迁到片内 RAM30H~

60H 中, 并将原数据块区域全部清 0。

10. 试编写一子程序, 使间址寄存器 R1 所指向的两个片外 RAM 连续单元中的高 4 位二进制数, 合并为一字节装入累加器 A 中。已知 R0 指向低地址, 并要求该单元高 4 位放在 A 的高 4 位中。

11. 试编程把以 2040H 为首地址的连续 50 个单元中的无符号数按降序排列, 存放到以 3000H 为首地址的存储区中。

12. 试编一查表程序, 从首地址为 2000H, 长度为 100 的数据块中找出 ASCII 码 A, 将其地址送到 20A0H 和 20A1H 单元中。

13. 设在 200H~204H 单元中, 存放有 5 个压缩 BCD 码, 编程将它们转换成 ASCII 码, 存放到以 205H 单元为首地址的存储区中。

14. 有一个无符号数据块的长度在 30H 单元, 其首地址为 31H 单元。求出数据块中最大值, 并存入 40H 单元。

15. 有两个无符号数据块, 求其中的最大值。2 个数据块的首地址分别为 60H 和 70H, 每个数据块的第一字节都存放数据块长度。结果存入 5FH 单元。

16. 在以 2000H 为首地址的存储区中, 存放着 20 个用 ASCII 码表示的 0~9 之间的数, 试编程, 将它们转换成 BCD 码, 并以压缩 BCD 码 (即一个单元存放 2 位 BCD 码) 的形式存放在 3000H~3009H 单元中。

17. 试编程实现下列逻辑表达式的功能。设 P1.7~P1.0 为 8 个变量的输入端, 而其中 P1.7 又作为变量输出端。

$$\textcircled{1} \quad Y = X_0 X_1 \overline{X_2} + \overline{X_3} + X_4 X_5 X_6 + \overline{X_7}$$

$$\textcircled{2} \quad Y = \overline{X_0 X_1} + \overline{\overline{X_2 X_3 X_4} + \overline{X_5 X_6 X_7}}$$

18. 试编写一个多字节无符号数加法子程序。

19. 试编写一个多字节无符号数减法子程序。

20. 试编写延时 1s、1min、1h 的子程序。

21. 用程序实现 $c = a^2 + b^2$ 。设 a 、 b 均小于 10。 a 存在 31H 单元中, b 存在 32H 单元, 把 c 存入 33H 单元。

22. 如何实现将内存单元 40H~60H 的数逐个对应传到片外 2540H~2560H 单元中?

23. 在片内 30H 和 31H 单元各有一个小于 10 的数, 编程求这两个数的平方和, 用调用子程序方法实现, 结果存在 40H 单元中。

24. 本题内容同上题, 要求用查表方法实现。

25. 根据图 4-1 的线路, 设计灯亮移位程序, 要求 8 个发光二极管每次亮一个, 点亮时间为 40ms。顺次一个一个地循环右移点亮, 循环不止。已知时钟频率为 24MHz。

26. 根据图 4-1 的线路, 设计灯亮程序, 要求 8 只发光二极管间隔分两组, 每组 4 只, 两组交叉轮流发光, 反复循环不止, 变换时间为 100ms, 已知时钟频率为 24MHz, 请设计程序。

27. 求一个 16 位二进制数的补码。设此 16 位二进制数存放在 R1、R0 中, 求补后存入 R3、R2 中。

28. 将累加器 A 中 0~FFH 范围内的二进制数转换为 BCD 数 (0~255)。

29. 把 A 中的压缩 BCD 码转换成二进制数。

第 5 章 C51 语言程序及开发环境

由于汇编语言编程难度大,编程周期长,可移植性差,多年前就已经出现了用于单片机的高级语言。目前,绝大多数嵌入式系统工程都使用 C 语言作为编程语言。本章介绍用于 51 系列单片机的 C51 语言程序设计,考虑到多数读者已经学习过通用 C 语言,因此对于通用 C 语言的基本知识仅提纲挈领地简要介绍,重点介绍 C51 语言与通用 C 语言的不同处和应用于单片机编程时的使用注意事项,并列举相关实例。

5.1 C51 语言基础知识

汇编语言对单片机的操作直接、简洁、结构紧凑,实时性强,但对于复杂的运算或者大型程序,用汇编语言编制周期长,出错率高,相互交流差,且不易移植。此时则应该选择用高级语言编程,C 语言是一种计算机中使用较广泛的程序设计语言,C51 语言是建立在通用 C 语言基础上的,用于 51 系列单片机编程的最流行的高级语言。许多以前只能采用汇编语言来解决的问题现在都可以改用 C 语言来完成。因为两种语言各有优缺点,一个好的单片机程序员应该是在掌握汇编语言的基础上,再学会用于单片机编程的 C51 语言。

5.1.1 C51 语言简介

C51 语言是一种结构化语言,可产生紧凑代码,语言简洁,使用方便灵活。C51 语言既具有一般高级语言的特点,还可以对计算机硬件直接进行操作,可直接访问单片机的物理地址,包括寄存器、存储器以及外部接口器件,还能与汇编语言混合编程,这些正是单片机编程应用所需要的。除此,其与汇编语言相比,主要有如下优点:

- (1) 不要求十分熟悉单片机的指令系统,仅要求对单片机的基本硬件结构有一定了解;
- (2) C51 语言具有丰富的数据结构类型及多种运算符,表达和运算能力较强,使用方便;
- (3) C51 语言是以函数为程序设计基本模块的,程序容易移植;
- (4) 具有丰富的库函数,其中包括许多标准子程序,具有较强的数据处理能力;
- (5) 源代码可读性较强,容易理解和编程,源文件简短,易于交流。

大多数计算机都支持对 C 语言的应用,因而可以方便地在 PC 机上直接编写和测试部分程序。多数情况,在 PC 上调试正常的代码段可以直接移植到目标单片机上,这样可以在没有硬件情况下开始编写和调试程序,大大缩短编程和调试时间,从而提高编程效率。

5.1.2 C51 语言的运算符及表达式

运算符是完成某种运算的符号,用于进行数据处理,包括算术、关系、逻辑运算符等,表达式是由运算符和括号将运算对象连接为具有指定功能的式子,由运算符或表达式可以构成 C51 程序的各种语句,因而这是 C51 语言的基础知识。

下面简要说明如下：

1. 算术运算符

＋ 加或取正值运算符；－ 减或取负值运算符；* 乘运算符；/ 除运算符；

% 模运算符（或称求余运算符）。

算术运算符中的优先级规定为：先乘除模，后加减，括号最优先。

2. 关系运算符

< 小于运算符；> 大于运算符；<= 小于或等于运算符；

>= 大于或等于运算符；== 测试等于运算符；!= 测试不等于运算符。

前 4 种运算符优先级相同，后 2 种优先级相同，前 4 种运算符优先级高于后 2 种。关系运算符优先级低于算术运算符，高于赋值运算符。

3. 逻辑运算符

&& 逻辑与运算符；|| 逻辑或运算符；! 逻辑非运算符。

逻辑操作运算符的优先级从高至低依次为：!→&&→||。逻辑非运算符优先级高于算术运算符，逻辑或运算符优先级低于关系运算符，高于赋值运算符。

4. 位操作运算符

& 按位与运算符；| 按位或运算符；^ 按位异或运算符；~ 按位取反 运算符；

注意位逻辑运算与上述的逻辑运算是 2 个不同的概念。

<< 按位左移运算符，例 $a \ll b$ ，即 a 变量的值按位左移 b 位；

>> 按位右移运算符，例 $a \gg b$ ，即 a 变量的值按位右移 b 位。

上述两条移位指令移位后，空白位补 0，溢出位舍弃。

位操作运算符的优先级从高至低依次为：~→<<和>>→&→^→|。

5. 自增减运算符

++ 自增运算符，例 $a++$ ， $++a$ ，即 a 变量的值自动加 1；

-- 自减运算符，例 $a--$ ， $--a$ ，即 a 变量的值自动减 1；

由于自增减运算符位置不同，所以变量的运算过程也不一样， $++a$ （或 $--a$ ）是在使用 a 值之前，先使 a 值加 1（或减 1），而 $a++$ （或 $a--$ ）是在使用 a 值之后，再使 a 值加 1（或减 1）。

6. 复合赋值运算符

在赋值运算符“=”前面加上其他运算符，就构成了复合赋值运算符。C51 中共提供了 11 种复合赋值运算符，即：

$+=$ ， $-=$ ， $*=$ ， $/=$ ， $\%=$ ， $>>=$ ， $<<=$ ， $\&=$ ， $|=$ ， $\^=$ ， $\sim=$ 。

这种复合赋值运算可简化程序，提高 C51 程序编程效率。例如：

$a+=b$ ，相当于 $a=a+b$ ； $a>>=b$ ，相当于 $a=a>>b$ 等等。

当表达式中出现多种运算符时，要注意运算符的优先级及结合性。

5.1.3 C51 语言的程序结构

C51 语言的程序结构与 C 语言相同，是采用函数结构，在 C 语言的源程序中都包括一个名为 `main()` 的主函数，它是程序入口，除了主函数外，一般还包含其他函数。主函数可以调用其他函数，这些函数类似子程序，C51 程序是从执行 `main()` 函数开始，调用其他函数后返回到主函数中，最后在主函数中结束程序运行。

函数定义由类型、函数名、参数表和函数体组成。C51 函数的一般格式如下：

```
类型  函数名(参数表)
参数说明;
{
  数据定义说明部分;
  执行语句部分;
}
```

C51 的程序就是由一个个函数构成的，C51 的程序一般组成结构如下：

```
预处理命令 include<>
全程变量定义说明
函数说明
Main()/*主函数*/
{
  局部变量定义说明
  执行语句
}
function1 (形式参数表)/*功能函数 1*/
形式参数定义说明
{
  局部变量定义
  执行语句
}
.....
function n(形式参数表)/*功能函数 n*/
形式参数说明
{
  局部变量定义说明
  执行语句
}
```

C 语言的主要语句规则是：每个变量必须先定义再使用；一行可以写多条语句，但每个语句必须以“；”结尾；“{”必须成对使用；注释用 `/*...*/` 或者 `//` 表示。

5.1.4 C51 语言的流程控制语句

C51 语言是一种结构化语言，其基本单元是模块，每个模块包含若干基本结构，基本结构中可以有若干语句。C51 程序的基本结构是顺序结构、选择结构和循环结构。顺序结构是最基本、最简单的结构，不需要流程控制语句，下面仅简要介绍选择结构和循环结构中所用到的流程控制语句。

1. 选择结构

在选择结构中都有一个条件语句，按照不同的条件选择执行不同的分支。在 C51 中实现选择的语句主要是：if，switch/case。

1) if 语句

if 语句是 C51 中的基本判断语句，有 3 种形式的 if 语句：

(1) if (表达式) {语句; }

例 if(x==y) P1=0; /* 如果 x 等于 y，则执行 P1=0 */

(2) if(表达式) {语句 1; }else{语句 2; }

例 if(x==y) P1=0;

else P1=0xff; /* 如果 x 等于 y，则执行 P1=0，否则执行 P1=0xff */

(3) if(表达式 1) {语句 1; }

else if(表达式 2) {语句 2; }

else if(表达式 3) {语句 3; }

.....

else if(表达式 m-1) {语句 m-1; }

else {语句 m; }

例 if(a>3) { b=60; }

else if(a>2) { b=50; }

else if(a>1) {b=40; }

else { b=100; }

2) switch/case 语句

switch/case 是专门处理并行多分支的选择语句。它的格式如下：

switch(表达式)

{

case 常量表达式 1: {语句 1; break; }

case 常量表达式 2: {语句 2; break; }

.....

case 常量表达式 n: {语句 n; break; }

default: {语句 n+1; break; }

}

例 switch(y)

{

case 1: x=1; break;

case 10: b=2; break;

case 100: a=20; break;

default:

break;

}

2. 循环结构

循环结构用于实现程序段的重复执行，可实现循环的语句为：while，do—while，for。

1) while 语句

while 语句用于实现在循环执行之前检测循环结束条件，它的格式如下：

while(表达式) /*表达式是能否循环的条件*/

{语句; } /*循环体*/

例 while(a>10) /* a>10, 执行循环体*/

{b=20; }

2) do—while 语句

do—while 语句用于实现在循环体的结尾处检测循环结束条件，它的格式如下：

do{

语句; } /*循环体*/

while(表达式);

例 int sum=0, i=0;

do { sum +=i; /* sum 求和*/

i++; /*循环*/

}while (i<=10) ; /*判定条件*/

3) for 语句

for 语句是使用最多，也最灵活的一种循环语句，它既可用于循环次数确定的情况，也可用于循环次数不确定，但已经给出循环条件的情况。它的格式如下：

for(表达式 1; 表达式 2; 表达式 3)

{ 语句; } /*循环体*/

例 int i, sum;

sum=0 ;

for(i=0; i<=10; i++) /*i 初始值为 0, 在 i<=10 时执行循环体，每循环一次 i+1*/

{

sum=sum+i; /*循环体*/

}

3. 跳转结构

在循环语句执行过程中如果需要在满足循环条件下跳出执行程序段，则可以采用 continue、break 语句，如果要从执行程序段跳到程序的其他地方，可以使用 goto 语句。

1) continue 语句

如果在循环体中遇到 continue 语句，则跳过循环体下面的语句，然后从最后一行（通常是右括号 “}” 开始继续下一次循环。

2) break 语句

如果在循环体中遇到 break 语句，则跳出循环体，继续执行下面的语句；

如果在 switch 分支结构中遇到 break 语句，则跳出当前 switch 分支结构，继续执行下面的语句。

3) goto 语句

goto 语句是无条件跳转语句，格式为：goto 标号；

标号可以是字母、数字和下划线，第一个字符必须是字母或者下划线。

goto 语句可以在一个函数内任意跳转，这种特性破坏了程序结构，所以在正常的程序中一般不用这个语句。

5.2 C51 语言对通用 C 语言的扩展

由于 C 语言最初是为通用计算机设计的，在这种计算机中只有一个程序和数据统一寻址的内存空间。但在 51 系列单片机中，程序是保存在 ROM 中而数据存放在 RAM 中。标准 C 语言并没有提供对这部分内存地址范围的定义，为了支持 80C51 系列单片机的硬件结构，加入了一些扩展的内容。扩展的内容包括：数据类型、存储器类型、存储模式、指定函数的寄存器组、指定函数的存储模式及定义中断服务程序等。本节将简要地介绍 C51 语言对通用 C 语言的扩展，以及扩展后对单片机硬件结构的访问。

5.2.1 数据类型

在 C51 程序中用到的变量一定要先定义数据类型，定义之后 C51 编译器才能在内存中按数据类型长度给该变量分配空间，C51 支持的各种规格的数据类型列于表 5-1。除了这些数据类型以外，变量可以组合成结构、联合及数组。在用 C51 语言编写单片机程序时，需要根据单片机的存储器结构和内部资源定义相应的数据类型和变量。

表 5-1 C51 支持的数据类型

数据类型	长度	数值范围
signed char	1 字节	-128~+127
unsigned char	1 字节	0~255
signed int	2 字节	-32768~+32767
unsigned int	2 字节	0~65535
signed long	4 字节	-2147483648~+2147483647
unsigned long	4 字节	0~4294967295
float	4 字节	$\pm 1.175494\text{E}-38 \sim \pm 3.402823\text{E}+38$
bit	1 位	0 或 1
sbit	1 位	0 或 1
sfr	1 字节	0~255
sfr16	2 字节	0~65535

数据表 5-1 所列的数据类型中，**bit**、**sbit**、**sfr** 和 **sfr16** 等 4 种类型是 C51 编译器中新增的数据类型，在通用 C 中没有，**sbit**、**sfr** 和 **sfr16** 类型的数据是用于操作 80C51 的特殊功能寄存器的。分别介绍如下：

1. bit 类型

bit 数据类型可以定义一个位变量，可能在变量声明参数列表和函数返回值中 useful。所有

的 **bit** 变量放在 80C51 内部 RAM 存储区的位操作数段。因为这个区域只有 16 字节长，所以最多只能声明 128 个位变量。

bit 变量的声明中，应包含存储类型。但是因为 **bit** 变量存储在 80C51 的内部数据区，只能用 **data**、**bdata** 和 **idata**（只限于可位操作的部分）存储类型，不能用别的存储类型。

一个 **bit** 变量的声明与其他数据类型相似，例如：

```
bit flag1;           /*定义 flag1 为位变量 */
bit bdata   ag1;     /*定义 ag1 为位变量 */
```

bit 变量和 **bit** 声明有以下限制：

(1) 禁止中断的函数 (**#pragma disable**) 和用一个明确的寄存器组 (**using n**) 声明的函数，不能返回位值。

(2) 一个位不能被声明为一个指针，例如：**bit *ptr** 为非法。

(3) 不能声明 **bit** 类型的数组，例如：**bit ware[5]** 为非法。

(4) 不能用 **data** 和 **idata** 存储类型外的其他存储类型，例如：**bit xdata ag1** 为非法。

2. sbit、sfr 和 sfr16 数据类型

80C51 系列单片机用特殊功能寄存器 SFR 来控制计时器、计数器、串行口、并行口和外围设备。它们可以用位、字节和字访问。与此对应，编译器提供 **sbit**、**sfr** 和 **sfr16** 数据类型访问 SFR。下面说明这些数据类型。

1) sfr 类型

sfr 可访问一个 8 位的 SFR。例如：

```
sfr P0=0x80;         /*定义 80C51 的 P0 口，地址为 80h */
sfr TL0=0x8A;        /*定义 80C51 定时器低字节 TL0，地址为 08Ah */
```

P0 和 **TL0** 是声明的 SFR 名。在等号 (=) 后指定的地址必须是一个常数值不允许用带操作数的表达式。传统的 80C51 系列支持 SFR 地址从 0x80 到 0xFF。在 C51 语言中表示 16 进制时，在数字前面用 0x 表示，所以 0x80，即 80H。

2) sfr16 类型

sfr16 将两个 8 位的 SFR 作为一个 16 位的 SFR 来访问。访问该 16 位的 SFR 只能是低字节跟着高字节，即将低字节的地址用作 **sfr16** 声明的地址。例如：

```
sfr16 DPTR=0x82;     /* 定义数据指针 DPL 的地址为 082H，DPH 的地址为 083H */
```

在这个例子中，**DPTR** 被声明为 16 位 SFR。

sfr16 声明和 **sfr** 声明遵循相同的原则。任何符号名可用在 **sfr16** 的声明中，等号 (=) 指定的地址，必须是一个常数值。不允许使用带操作数的表达式，而且必须是 SFR 的低位和高位字节中的低位字节的地址。

3) sbit 类型

编译器用 **sbit** 数据类型访问可位寻址的 SFR 中的位。例如：

```
sbit EA=0xAF;        /* 定义 EA 位的地址为 0AFH */
sbit RS0=PSW^3;      /* 定义 RS0 是 PSW 的第 3 位 */
```

需要注意：**int** 整型数与 **long** 长整型数在 C51 中的存放格式与标准 C 语言不同，在 C51 中是高字节存放在低地址，低字节存放在高地址，而标准 C 语言则相反。在使用结构体和共用体定义变量时要明确其存放的前后顺序，其他情况可不考虑。

5.2.2 数据的存储类型

80C51 的存储区域有两个特点：

- 程序存储器和数据存储器是截然分开的；
- 特殊功能寄存器与内部数据存储器是统一编址的。

C51 编译器支持 80C51 的这种存储器结构，能够访问 80C51 的所有存储器空间。

针对 80C51 存储空间多样性，提出了修饰存储空间的修饰符，用以指明所定义的变量应分配在什么样的存储空间，如表 5-2 所示。

表 5-2 C51 存储类型与 80C51 存储空间的对应关系

存储类型	与 80C51 存储空间的对应关系
code	程序存储器空间（64 KB）；通过 <code>MOVC @A+DPTR</code> 访问。
data	直接访问的内部数据存储器；访问速度最快（128 字节）。
idata	间接访问的内部数据存储器；可以访问所有的内部存储器空间（256 字节）。
bdata	可位寻址的内部数据存储器；可用字节或位方式访问（16 字节）。
xdata	外部数据存储器（64 KB）；通过 <code>MOVX @DPTR</code> 访问。
pdata	分页的外部数据存储器（256 字节）；通过 <code>MOVX @Ri</code> 访问。

1. 程序存储区

程序（CODE）存储区是只读的，不能写入。硬件决定最多只能有 64KB 的程序存储区。

用 `code` 标识符访问片内、片外统一编址的程序存储区，寻址范围为 0~65535。在此空间存放程序编码和其他非易失性信息。在汇编语言中是用间接寻址的方式访问程序存储区数据，如 `MOVC A, @A+DPTR` 或 `MOVC A, @A+PC`。

定义举例：

```
char code text[] = "ENTER"; /*在程序代码段定义了一个字符型数组*/
```

2. 内部数据存储器

内部数据存储器是可读、可写的。80C51 系列最多可有 256 字节的内部数据存储器。内部数据区可以分成三个不同的存储类型 `data`、`idata` 和 `bdata`。

data 存储类型标识符通常是指低 128 字节的内部数据区，为片内直接寻址的 RAM 空间，寻址范围为 0~127。在此空间内存取速度最快。

idata 存储类型标识符是指全部 256 个字节的内部存储区，为片内间接寻址的 RAM 空间，寻址范围为 0~255。在汇编语言中采用的寻址指令形式为 `MOV @Ri`。由于只能间接寻址，访问速度比直接寻址慢。

bdata 存储类型标识符是指可位寻址的 16 字节内部存储区（20H~2FH），位地址范围为 0~127。此空间允许按字节和按位寻址。在本区域可以声明可位寻址的数据类型。

定义举例：

```
char data vr ;          /*定义字符变量 vr 在 data 存储区*/
float idata x, y;        /*定义浮点变量 x, y 在 idata 存储区*/
bit bdata flags ;        /*定义位变量 flags 在 bdata 存储区*/
```

3. 外部数据存储区

外部数据存储区是可读、可写的。可通过一个数据指针加载一个地址间接访问外部数据区。因此，访问外部数据存储区的速度比访问内部数据存储区慢。

外部数据存储区最多可有 64KB。这些地址不一定都用来作为数据存储区。因为单片机外围设备的地址也在该存储区（详见第 9 章）。

编译器提供两种不同的存储类型 **xdata** 和 **pdata** 用于访问外部数据存储区。

xdata 存储类型标识符是指外部数据存储区（64KB）内的任何地址，寻址范围为 0~65535。汇编语言中采用寻址指令形式为 **MOVX A, @DPTR** 和 **MOVX @DPTR, A**。

pdata 存储类型标识符仅指一页或 256 字节的外部数据存储区，寻址范围为 0~255。具体页数由 P2 口决定。汇编语言中采用寻址指令形式为 **MOVX A, @Ri** 和 **MOVX @Ri, A**。

在定义变量时，通过指明存储器类型，可以将所定义的变量存储在指定的存储区域中。

访问内部数据存储器将比访问外部数据存储器快得多。因此，应该把频繁使用的变量放置在内部数据存储器中，把很少使用的变量放在外部数据存储器中。

在变量的声明中，可以包括存储器类型和有符号 **signed** 或无符号 **unsigned** 属性。

定义举例：

```
unsigned long xdata array[10];      /*定义无符号长整型数组 array[10]在片外数据存储区*/
unsigned int pdata dimension;       /*定义无符号整型数 dimension 在一页范围内的片外数据
                                   存储区 */
unsigned char xdata vector[5][4][4]; /*定义无符号字符型三维数组变量 vector[5][4][4]
                                   在片外数据存储区 */
char bdata flags;                  /*定义字符型变量 flags 位于片内位寻址区 */
```

如果在变量的定义中，没有包括存储器类型，将自动选用默认的存储器类型。

虽然 C 语言程序看起来操作很简单，但实际上 C 编译器需要用一系列指令对其进行复杂的变量类型、数据类型的处理，特别是对于浮点变量的处理，将大大增加运算时间和程序长度。如果在编程时使用大量不必要的变量类型，最终会导致程序过于庞大，运行速度减慢，所以必须特别慎重地选择变量和数据类型。

4. 存储模式

在 C51 编译器中，可以用控制命令 **SMALL**、**COMPACT** 和 **LARGE** 定义存储模式，用以指明所定义变量的默认存储器类型。

1) 小 (SMALL) 模式

在这种模式下，所有变量都默认定义在内部数据存储器中，这和用 **data** 显式定义变量作用相同。一般情况下，应该使用小 (SMALL) 模式，它产生最快，最紧凑，效率最高的代码。如果没有指定，则系统都默认为小模式。

2) 紧凑 (COMPACT) 模式

在这种模式下，所有变量都默认存放在外部数据存储器的一页中，最多只能提供 256 字节的变量，这和用 **pdata** 显式定义变量作用相同。

3) 大 (LARGE) 模式

在这种模式下，所有的变量都默认在外部存储器 (**xdata**) 中。这和用 **xdata** 显式定义变量作用相同。

存储模式定义性格式为：

类型说明符 函数标识符（形参表） 存储模式修饰符{small, compact, large}

存储模式为本函数的参数和局部变量指定的存储空间，在指定了存储模式之后，该空间将再也不随编译模式而变。如：

```
extern int func (int i, int j) large; /* 修饰为大模式 */
```

在定义变量时，如果已经指定存储器类型，就不必加上上述存储模式命令。

5.2.3 指针

指针是 C 语言中的一个重要概念，也是主要特色之一，简言之，指针就是存放变量的地址。它用于间接访问变量，类似于汇编语言中用寄存器间接寻址方式。正确使用指针，可以有效地表示复杂的数据结构，直接处理内存地址，并可以有效方便地使用数组，可以使程序简洁、高效。

1. 指针的概念

指针的 2 个基本概念就是变量的指针和指针变量。

变量的指针——在 C 语言中，变量的指针是该变量的地址。当在程序中定义了一个变量，C51 编译器在编译时就给这个变量分配一定的字节单元进行存储。通常字符型（char）变量、整型（int）变量和浮点型（float）变量分别被分配 1、2、4 个字节的内存单元。变量名相当于内存单元的地址，变量的值即该地址中的内容。

指针变量——用于存放“变量的指针”的变量称为指针变量。如果有一个变量 b，它被指定用于存放整型变量 a 的地址，则 b 就被称为 a 的指针变量。

2. 指针变量的定义

C 语言规定所有的变量在使用前必须定义，以确定其类型。为了表示指针变量与变量地址之间的关系，C51 语言中规定了两个运算符：

& 取地址运算符；定义&后的变量为变量地址；

* 指针运算符或者指针类型说明符，说明如下：

“*”在不同的场合所代表的含义是不同的，例如：

int *sp ; 此时的*为指针变量类型说明符。

x=*sp ; 此时的*为指针变量 sp 的运算符，即把 sp 所指向的变量值赋给 x。

指针定义的一般形式为：

类型识别符 [存储器类型] *指针变量名

类型识别符是指针变量的类型，存储器类型是可选项，如果没有则是指一般（也称为通用）指针，如果有则是存储器指针。一般指针需要 3 个字节存储。第一个字节用于表示存储器类型，第二个字节是指针的高字节，第三字节是指针的低字节。

一般指针可以用来访问所有类型的变量，而不管变量存储在哪个存储空间中。因而许多库函数都使用通用指针。通过使用通用指针，一个函数可以访问数据，而不用考虑它存储在什么存储器中。一般指针的声明和标准 C 语言中一样。例如：

```
char *s; /*定义一个指向字符串变量的指针变量 s */
sp=&s ; /*通过取地址运算符&使 sp 指向变量 s 的地址 */
```

3. 存储器指针

存储器指针在定义时要包含一个存储器类型说明。用这种指针访问对象，可以只占用 1～2 个字节。指向 `idata`，`data`，`bdata` 和 `pdata` 的存储器指针使用一个字节来保存；指向 `code` 和 `xdata` 的存储器指针用两个字节来保存。例如：

```
char data *str;           /*在 data 区域中定义一个指向字符串变量的指针变量 str，
                           指针为 1 字节长 */
int xdata *numtab;        /*在 xdata 区域中定义一个指向整型变量的指针变量 numtab，
                           指针为 2 字节长*/
long code *powtab;        /*在 code 区域中定义一个指向长整型变量的指针变量 powtab，
                           指针为 2 字节长*/
```

使用存储器指针比通用指针效率高，速度快。然而，使用存储器指针不是很方便。通常是在所指向目标的存储空间明确并不会变化的情况下使用它。

4. 指针变量的引用

在对变量、指针变量定义之后，就可以间接访问指针了。下面举例说明。

```
int x;                    /*定义整型变量 x */
int a;                    /*定义整型变量 a */
int *sp;                  /*定义指针变量 sp */
a=10;                     /*给整型变量 a 赋值为 10*/
sp=&a ;                   /*通过取地址运算符&使 sp 指向变量 a 的地址 */
x=*sp ;                   /*整型变量 a 的值 10 通过间接寻址的方法赋给 x */
```

在对变量、指针变量定义之后，C 编译器会自动给它们在内存中安排相应的内存单元，例如把 `a` 安排在 120，121 内存单元中，`sp` 的地址安排在 200，201 内存单元中。在实际的编程和运算过程中变量和指针变量的地址都是不可见的，它们的对应关系完全是由 C 编译器自动确定的。程序设计者只需要通过取地址运算符`&`和指针运算符`*`把变量、指针变量联系起来。

5.2.4 函数

C 语言程序是由函数构成的，函数是 C 语言程序中的基本模块，C51 函数分为两类，一类是库函数，一类是用户定义函数。用户定义函数是用户自己设计的函数。库函数是在库文件中已经定义的函数，其函数说明在相关的头文件中，用户在编程时只要用 `include` 指令将头文件包含在用户文件中即可，如下式。

```
#include<math.h> /* math.h 为专用数学库函数，在包含了这个文件后，
                  程序就可以调用相关数学计算*/
```

库函数中通常已经由系统设计者保存了很多常用的功能模块（例如相关数学计算等），用户如果充分利用这些库函数，将能大大提高编程效率。

C51 语言中函数的定义、参数和函数值及函数调用等内容与标准 C 语言基本相同，下面仅说明几个不同点。

1. 寄存器组的切换（using 修饰符的应用）

在 51 系列单片机中有 4 个寄存器组，每组包含 8 个通用寄存器，在采用中断程序时，

经常需要保护某些寄存器组，此时采用交换寄存器组的方法很方便快捷，可避免采用很多入栈及出栈指令。C51 编译器定义了一个函数 `using` 可方便地用于寄存器组的交换。

函数使用指定寄存器组的定义性说明如下：

```
void 函数标识符(形参表) using n
```

其中 $n=0\sim3$ 为寄存器组号，对应 80C51 中的 4 个寄存器组。

函数使用了 `using n` 后，C51 编译器自动在函数的汇编码中加入如下的函数头段和尾段：

```
{    push    psw
    mov     psw, #与寄存器组号 n 有关的常量
    :
    pop     psw
}
```

应该注意的是，`using n` 不能用于有返回值的函数。因为 C51 的返回值是放在寄存器中的，而返回前寄存器组却改变了，将会导致返回值发生错误。

2. 中断服务程序 (interrupt 修饰符的应用)

为实现在 C51 源程序中直接编写中断程序 (有关中断的概念详见第 8 章)，C51 编译器允许用 C 语言创建中断服务程序。在 C51 编译器中增加了一个扩展关键字 `interrupt`，在函数声明时包括 `interrupt m`，将把所声明的函数定义为一个中断服务程序。其格式为：

```
void 函数标识符(void) interrupt m [using n]
```

其中， $m=0\sim31$ ，0 对应于外部中断 0；

1 对应于定时器 0 中断；

2 对应于外部中断 1；

3 对应于定时器 1 中断；

4 对应于串行口中断；

5 对应于定时器 2 中断；

其他为预留。

`using n` 是可选项，用于为中断函数指定所用的寄存器组， n 为组号，默认为 0。

从定义中可以看出，中断函数必须是无参数、无返回值的函数。举例如下：

```
unsigned int interruptcnt;          /*定义一个无符号整型数 interruptcnt */
unsigned char second;              /*定义一个无符号字符串 second */
void timer0(void) interrupt 1 using 2 /*定义函数名为 timer0 的定时器 0 中断 1 服务函数，
                                     使用第 2 组寄存器 */
{
    if (++interruptcnt == 4000)     /* 加 1 计数，测试是否到 4000 */
    {                               /* 计数开始*/
        second++;                  /* 秒计数器加 1 */
        interruptcnt = 0;          /* 清除中断计数器 */
    }
}
```

在 C 语言中调用中断服务程序，用户只需关心中断号和寄存器组的选择。编译器将自动产生中断向量和程序的入栈及出栈代码。对于 A 累加器及 PSW 等寄存器的保护与恢复都是由 C51 编译器自动进行的，用户可不考虑，但其他寄存器和内存是要考虑保护的。

需要注意在任何情况下都不能直接调用中断服务程序。还要注意，如果在中断函数中又

调用了其他函数，则要保证使用寄存器组的一致性，通常要用到 `using` 选项。结合第 8 章的内容，能更深刻地理解这个问题。

5.2.5 C51 语言对单片机硬件的访问

用于单片机的 C51 语言最主要的特色就是解决了与单片机的硬件接口问题。通常的做法是对于片内/片外的 I/O 口、特殊功能寄存器及存储器地址进行定义，定义后变量就与实际地址建立了联系。就可以用软件对硬件进行操作了。下面分别介绍如何用 C 语言访问单片机的特殊功能寄存器、存储器及片外接口器件。

1. 访问特殊功能寄存器

在 C51 中定义了一个头文件：`reg51.h`（对于 52 系列，如 AT89S52 是 `reg52.h`），它定义了 80C51 的所有功能寄存器和中断，在 C51 语言的源程序中采用 `#include<reg51.h>` 就可包含这个文件，在包含了这个文件后，程序就可以识别这些寄存器以及可位寻址的寄存器位的符号。就可以省略用 `sbit` 和 `sfr` 定义这些寄存器的地址。

例如：

```
#include<reg51.h>          /*包含寄存器头文件 */
P0=0;                      /*将端口 P0 全部设为低电平，就可不写 sfr P0=0x00 了*/
unsigned char in1;          /*定义一个字节变量 in1 */
unsigned char in2;          /*定义一个字节变量 in2 */
in1=P0;                     /*读取端口 0 中的数据到变量 in1 中*/
in2=TL0;                    /*读取定时器 0 TL0 中的数据到变量 in2 中*/
CY=0;                       /*将进位位 CY 清为零*/
```

2. 访问存储器

在 C51 中可以通过变量的形式访问存储器，也可以通过绝对地址访问存储器。对于变量的形式实际就是通过指针的方法，用户可不关心具体地址。但有些情况是需要知道绝对地址的，在此主要介绍如何访问绝对地址。对绝对地址的访问主要有如下 3 种形式。

1) 采用 C51 中的预定义宏

C51 编译器提供了 8 个宏定义用于对 51 单片机的存储器进行绝对寻址，其函数原型为：

```
#define CBYTE ((unsigned char volatile code *) 0)
#define DBYTE ((unsigned char volatile data *) 0)
#define PBYTE ((unsigned char volatile pdata *) 0)
#define XBYTE ((unsigned char volatile xdata *) 0)
#define CWORD ((unsigned int volatile code *) 0)
#define DWORD ((unsigned int volatile data *) 0)
#define PWORD ((unsigned int volatile pdata *) 0)
#define XWORD ((unsigned int volatile xdata *) 0)
```

其中宏名 `CBYTE` 是以字节形式访问 `code` 区，`DBYTE` 是以字节形式访问 `data` 区，`PBYTE` 是以字节形式访问 `pdata` 区，`XBYTE` 是以字节形式访问 `xdata` 区。后面 4 条则是以字的形式访问这 4 个区。访问形式为：

宏名[地址]

这些函数原型放在头文件 `absacc.h` 中。使用时采用 `#include<absacc.h>` 就可包含这些函数。

例如：

```
#include<reg51.h>          /*包含寄存器头文件 */
#include<absacc.h>          /*包含绝对地址头文件 */
#define uchar unsigned char /*定义符号 uchar 为数据类型符 unsigned char */
#define uint unsigned int   /*定义符号 uint 为数据类型符 unsigned int */
void main (void)
{
    uchar var1;
    uint  var2;
    var1= DBYTE[0x30];       /*把 data 区中 30h 地址单元中的数赋给 var1 */
    var2= XWROD[0x1000];     /*把 xdata 区中 1000h 和 1001h 地址单元中的 16 位数赋给 var2 */
    .....
}
```

2) 通过指针访问

采用指针可以不关心具体地址指针，但也可以实现对任意指定存储单元的访问。例如：

```
uchar data var1;
uchar data *dp1;    /*定义一个 data 区中指针 dp1 */
uint xdata *dp2;    /*定义一个 xdata 区中指针 dp2 */
uchar pdata *dp3;   /*定义一个 pdata 区中指针 dp3 */
dp1=&var1;          /*取 data 区中变量 var1 的指针 */
*dp1=0xa0;          /*给变量 var1 赋值 a0h */
dp2=0x2000;         /* dp2 指针赋值，指向 xdata 区的 2000h */
*dp2=0x16;          /*将数据 16h 送到片外 RAM 区的 2000h 单元 */
dp3=0x20;           /* 给 dp3 指针赋值，指向 pdata 区的 20 h */
*dp3=0x80;          /*将数据 80h 送到 pdata 区的 20h 单元*/
```

3) 采用扩展关键字_at_

采用扩展关键字_at_访问存储器绝对地址的一般格式如下：

[存储器类型] 数据类型说明符 变量名 _at_ 地址常数

例如：

```
data  uchar x1 _at_ 0x20;    /* 在 data 区中定义字符变量的地址为 20 h */
xdata uint  x2 _at_ 0x2000;  /*在 xdata 区中定义整数变量的地址为 2000 h */
```

注意这个关键字的定义必须放在主函数前。

3. 访问外围接口器件地址

当单片机内部功能部件的功能不够用时，可以采用系统扩展的方法接入外围芯片，它们都可以直接与单片机接口（参看第 9 章），扩展的外围芯片 I/O 口采取与数据存储器相同的寻址方法，与片外数据存储器统一编址，可以采取上述寻址方法的任何一种。例如假设一块名称为 PD8255 的外围芯片的地址确定为 FFADH，则在程序中可以如下编制：

```
#define PD8255 XBYTE [0xFFAD] /* 定义芯片地址在片外 RAM 的 FFADH */
PD8255=0X80;                  /* 给该芯片端口赋值 80H */
```

5.3 C51 语言编程举例

C51 语言的一般程序设计方法与汇编语言相似，一般的 C51 语言程序都是顺序、选择、

循环三种结构的复杂组合。C51 语言中有一大批控制语句，用于控制程序的流程，以实现程序的选择结构和循环结构。本节将举例说明用 C51 语言编程的方法。通过对比，读者将更容易理解 C 语言编程与汇编语言编程的方法之间的异同及各自的优缺点。随着后续章节对单片机内部功能模块的学习，就可逐步掌握单片机的 C51 语言编程要领。

例 1 将数字 1 到 100 求和，结果存在片内 40H 和 41H 单元中。

用汇编语言编程如下：

```

                ORG     0000H
                SJMP    MAIN
                ORG     0030H

MAIN:
                MOV     A,      #00H
                MOV     40H,    A
                MOV     41H,    A
                MOV     R7,     #01
LOOP:          MOV     A,      R7
                ADD     A,      41H
                MOV     41H,    A
                CLR     A
                ADDC    A,      40H
                MOV     40H,    A
                INC     R7
                CJNE    R7,     #65H,LOOP
                SJMP    $
                END

```

用 C 语言编程如下：

```

#include <REG51.H>           /*包含寄存器头文件 */
#include <absacc.h>           /*包含绝对地址头文件 */
#define uchar unsigned char  /*定义符号 uchar 为数据类型符 unsigned char */
int  addresult_at_0x40;      /*定义变量*/
void main (void)
{
    uchar  i;
    addresult = 0;           /*变量清零*/    for (i=1 ;i<=100;i++)
    {
        addresult =addresult+i;    //求和
    }
    while(1){};
}

```

例 2 在 40H 开始的内存单元里存有 5 个无符号整数型值，一个整数型值占两个字节，共占 10 个字节。编程求其平均值，结果高位存放到内存单元的 50H，低位存放到 51H。首先需做多双字节加法，本例有 5 个数，因此用三个字节存放结果，再进行除法。

用汇编语言编程如下：

```

                ORG     0000H
                LJMP    MAIN
                ORG     0030H

MAIN:

```

```

        CLR      A
        MOV      0BH,    A
        MOV      0AH,    ,A
        MOV      09H,    A          ; 三个字节和清零。
        MOV      R3,     #0H        ; 计数器清零
LOOP0:  MOV      A,       R3 ;      5 个字节加法
        ADD      A,       ACC
        ADD      A,       #30H
        MOV      R0,     A
        MOV      A,       @R0
        ADD      A,       0BH
        MOV      0BH,    A
        INC      R0
        MOV      A,       @R0
        ADDC     A,       0AH
        CLR      A
        ADDC     A,       09H
        MOV      09H,    A
        INC      R3
        CJNE     R3,     #5,LOOP0
        MOV      R7,     0BH;      R4 到 R7 为被除数
        MOV      R6,     0AH
        MOV      R5,     09H
        MOV      R4,     #0H
        CLR      A
        MOV      R3,     #05H      ; R0 到 R3 为被除数
        MOV      R2,     A
        MOV      R1,     A
        MOV      R0,     A
        LCALL    SLDIV0           ; 调四字节除法子程序，结果在 R6R7 中
        MOV      51H,     R7
        MOV      50H,     R6      ; 除法结果
        SJMP     $

```

由于四字节除法子程序采用指令多达 60 条以上，且复杂难懂在此省略，可参考文献 2。

用 C 语言编程如下：

```

#include <REG51.H>
#include <absacc.h>
#define uchar unsigned char
unsigned int samle[5] _at_ 0x40;    //定义 5 变量数组

Void main (void)
{
    uchar i;
    long avge = 0;
    for (i=0 ;i<5;i++)
    {
        avge += samle[i];          //求和
    }
}

```

```

    avge = avge/5;                                //取平均值
    while(1){ };
}

```

例 3 把第 4 章的例 10 用 C 语言编写。

C 语言程序如下。

```

#include <reg51.h>
#include <intrins.h>                                //空操作函数定义
sbit S1=P3^1;
sbit S2=P3^2;
unsigned char show;
void delay10ms(void);
void delay1s(void);
void main()
{
    unsigned char times;                            //用于第二种发光方式的变量
    while(1)
    {
        P3=0Xff;                                    //设 P3 口为输入
        while((S1==1)&&(S2==1));                    //检查是否按过 S1 或 S2 键
        delay10ms();                                //延时 10ms 去键抖动
        if(S1==0)                                    //如果 S1 仍然为 0, 说明确实按过 S1 键,
                                                    //则调第一种发光方式子程序
        {
            show=0xfe;                                //第一种发光方式初值
            while(show!=0xff)                        //如果还未显示完执行循环中的程序
            {
                P1=show;                            //值送 P1 口
                delay1s();                            //1s 延时
                show=show<<1;                        //数据左移一位末位添 0
                show=show+1;                          //末位加 1
            }
        }
        if(S2==0) // 如果 S2 仍然为 0, 说明确实按过 S2 键, 则调第二种发光方式子程序
        {
            show=0xfe;                                //第二种发光方式初值
            for(times=0;times<8;times++) //循环显示 8 次
            {
                P1=show;                            //值送 P1 口
                delay1s();                            //1s 延时
                show=show<<1;                        //数据左移一位末位添 0
            }
        }
    }
}
void delay10ms() //延时 10ms 子程序
{
    unsigned char i,j;
    for(i=0;i<10;i++)
    {

```

```

        for(j=0;j<125;j++)
        {
            _nop_();
            _nop_();
        }
    }
}

void delay1s() //延时 1s 子程序
{
    unsigned char i;
    for(i=0;i<100;i++)
    {
        delay10ms();
    }
}

```

通过对比，显然采用 C 语言编程使程序更简捷明了，且较易于理解。C 语言编程与汇编语言编程各有所长。为了充分发挥 C 语言与汇编语言各自的优势，在有些情况下希望能够实现它们的混合编程。此时通常是用 C 语言编写主程序，把有严格时间限制的与硬件有关的子程序用汇编语言编写。在混合编程时汇编语言要按照 C 语言的规定进行函数名的转换和函数调用，在函数调用时存在参数传递与返回值问题，对于初学者这些问题不易理解，易出错，且在实用中这种方法较少采用，所以本书省略此内容，读者可参看文献。

5.4 Keil C51 软件开发环境

Keil C51 是 Keil 公司开发的用于 51 系列单片机的集成开发环境，它的作用是编辑、汇编、编译、仿真与调试 51 系列单片机的应用程序等，最终形成一个可执行文件，称为 Windows 下的集成开发环境-IDE (Integrated Development Environment)。本书将以 Keil μ Vision5 IDE 为例介绍。

5.4.1 Keil 软件简介

1. Keil IDE 的主要功能

Keil 公司所提供的集成开发环境 μ Vision5 IDE 是 Windows 下的集成开发环境，可仿真 51 系列及 ARM 等多种系列单片机及嵌入式微处理器，支持软件仿真和用户系统实时调试两种功能。Keil 编译器既可以对汇编语言源程序进行汇编，也可以对 C 语言源程序编译。Keil IDE 可以在没有单片机和仿真器的条件下进行软件仿真调试，不过该模拟仿真调试与真实的硬件调试还是有区别的，所以在软件模拟调试完成后，还需要用硬件进行调试。

Keil C51 开发软件包中的软件工具简介如下：

(1) C51 编译器：将 C 语言源程序（后缀是.C）编译成地址可重新定位的目标代码，并产生一个列表文件（后缀为.LST）。

(2) A51 宏汇编器：将汇编语言源程序（后缀是.ASM）汇编成可重新定位的目标代码，

也产生一个列表文件。

(3) BL51 连接/定位器：用于连接由编译器或汇编器生成的一个或多个目标模块和从库中提取的目标模块，并将可重新定位的段(段可以是数据段和程序段)分配到固定的地址上，产生一个绝对地址目标模块或文件。

(4) LIB51 库管理器：用于管理库文件，库文件是由编译器或汇编器产生的可重新定位的目标模块的集合，包括很多标准子程序（后缀为.LIB）。

所有的代码和数据连接完毕被安置在固定的存储器单元中，所产生的绝对地址目标文件可以直接写入单片机的存储器，或者用在线仿真器对程序进行调试。

在汇编和编译时如果发现程序中用到某个库文件，则连接/定位器可自动把该库文件与该程序的目标程序(后缀为.OBJ)连接到一起形成一个具有明确绝对地址的最终的目标文件(后缀为.OBJ)，然后再将 OBJ 文件转换为可以直接输出到单片机中的 HEX 目标文件格式。

2. Keil IDE 主界面简介

Keil μ Vision5 IDE 的安装与一般软件安装方法相同，安装后启动 Keil μ Vision5 IDE 程序，就出现如图 5-1 所示的主界面。下面简要介绍 Keil IDE 的主界面。

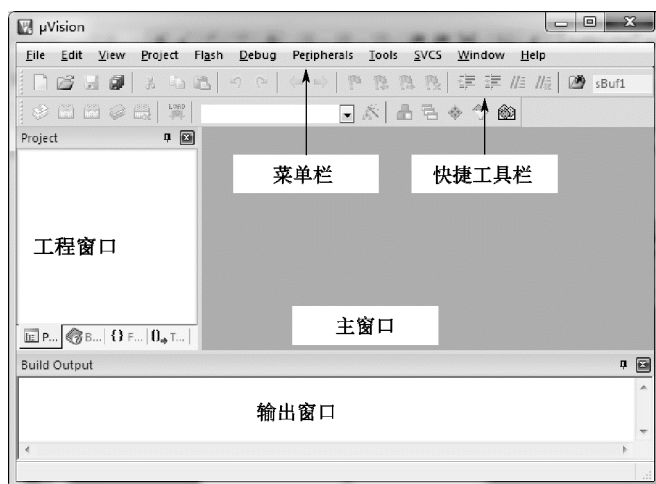


图 5-1 Keil μ Vision5 IDE 的主界面

在主界面标题栏下是菜单栏，选取菜单栏上的任意选项都会立即出现一个该选项的下拉菜单。通过鼠标或者键盘选取该菜单上的命令按钮，则可快速执行 Keil 的许多命令。菜单栏中所包括的菜单分别简要介绍如下：

1) 文件菜单 (File)

文件菜单中包括了创建新文件 (New)、打开已经存在的文件 (Open)、关闭当前文件 (Close)、保存当前文件 (Save) 等 10 多项与文件操作有关的命令。

2) 编辑菜单 (Edit)

编辑菜单中包括了取消上次操作 (Undo)、重复上次操作 (Redo)、剪贴操作 (Cut)、复制操作 (Copy) 等 10 多项与文本编辑有关的操作命令。

3) 视图菜单 (View)

视图菜单中包括了显示/隐藏状态栏 (Status Bar)、显示/隐藏工具栏 (Toolbars)、显示/隐藏项目窗口 (Project Window)、建立输出窗口 (Build Output Window) 等 10 多项与窗口显

示有关的操作命令。

4) 项目菜单 (Project)

项目(也可译工程)菜单中包括了创建新项目 (New Project)、创建新多项目工作区 (New multi-Project workspace)、打开一个已经存在的项目 (Open Project)、关闭当前的项目 (Close Project) 等 10 多项与项目有关的操作命令。

5) Flash 菜单 (Flash)

Flash 菜单用于对单片机中的 Flash (即闪存) 进行操作。包括下载到闪存 (download)、擦除闪存 (erase) 和打开闪存配置工具 (Configure flash tools) 3 项与闪存有关的操作命令。

6) 调试菜单 (Debug)

调试菜单中包括了开始/停止调试 (Start/Stop Debug Session)、CPU 复位 (Reset CPU)、连续运行 (run)、停止运行 (Stop) 等 10 多项与调试有关的操作命令。

7) 片上外设菜单 (Peripherals)

片上外设菜单中包括了打开片上外设中断对话框 (Interrupt)、打开片上外设 I/O 口对话框 (I/O-Port)、打开片上外设串行口对话框 (Serial)、打开片上外设定时/计数器对话框 (Timer) 等 4 项与片上外设有关的操作命令。片上外设菜单命令调试时才会出现, 外设对话框的内容与所选择的单片机有关。

8) 工具菜单 (Tools)

工具菜单中包括了配置 Gimpel Software 的 PC-Lint 程序 (Setup PC-Lint), 用 PC-Lint (是一种更加严格的编译器) 处理当前编辑的文件 (Lint)、用 PC-Lint 处理项目中所有的 C 源代码文件 (Lint all C Source Files)、配置联合工具 (Configure Merge Tool)、添加用户程序到工具菜单中 (Customize Tools Menu) 等 5 项操作命令。

9) 视窗菜单 (Window)

视窗菜单中包括了恢复调试窗口 view (Debug restore view)、使 view 复原为默认值 (Reset view to defaults)、分割当前的文件窗口 (Split)、关闭所有窗口 (Close all) 等几项与窗口操作有关的命令。

除上述常用菜单外还有软件版本控制系统菜单 (SVCS) 和帮助菜单 (Help)。

菜单栏下是工具栏, 工具栏中的快捷按钮允许用户快速执行一些常用的操作命令。图 5-1 中的工程窗口即项目管理窗口, 主窗口用于编辑程序文件, 输出窗口用于输出各种信息。这些窗口均可以通过视图 (View) 菜单下的命令打开与关闭。

5.4.2 项目的建立与设置

在 Keil μ Vision5 IDE 中对文件的管理是项目方式, 即将所需要的 C 语言源程序、汇编程序和头文件等都放在一个工程项目里统一管理。一般步骤如下。

1. 建立项目文件

通过执行 Project 菜单下的 New Project 命令建立项目文件, 操作如下:

选择 New Project 命令后, 出现如图 5-2 所示的 “Create New Project” 对话框。此时可以在编辑框中输入一个项目文件名, 如图中 Myproject。然后单击 “保存” 按钮, 就创建了一个文件名为 Myproject.uvproj 的新项目文件。

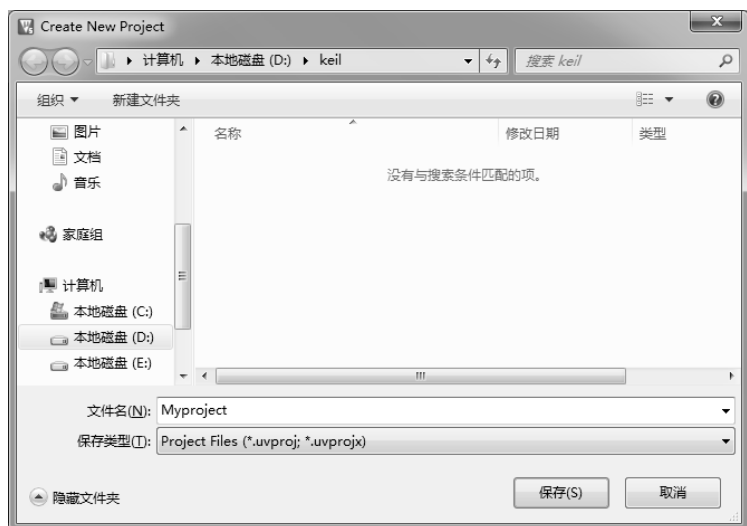


图 5-2 Create New Project 对话框

2. 选择用户芯片型号

在 Create New Project 对话框中选择建立新项目文件的位置、名称。当输完项目文件名，并单击“保存”按钮，将弹出“Select Device Target”对话框，见图 5-3。Keil IDE 将按公司分类以列表形式给出其所支持的单片机型号（不只限于 51 系列，还包括 ARM 等），用户应该根据实际所采用的单片机选择型号，也可在 Search 框中输入待选择的 AT89S51，选择后，右边的描述框将出现该型号单片机的相关信息。单击“OK”按钮，系统提示是否加入“STARTUP.A51”，单击“NO”，进入图 5-1 所示主界面。

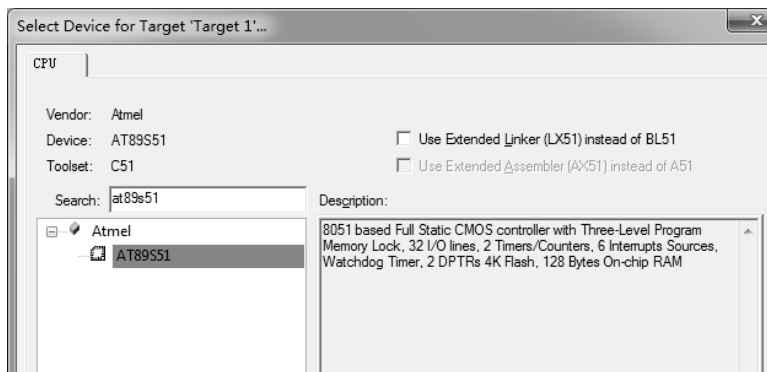


图 5-3 “Select Device Target ”对话框

3. 给项目中添加程序文件

在建立或者选择好项目文件后，要给项目文件中添加程序文件。如果还没有程序文件则应执行 File 菜单下的 New 命令建立程序文件。添加的文件可以是汇编语言程序，也可以是 C 语言程序。用户可用任意一种文本编辑工具编写源文件，在录入和编辑源文件时注意一定要用英文字符和符号。如果是新建的程序，则要将汇编语言程序用.ASM 后缀，C 语言程序用.C 后缀存盘后再添加到项目中。步骤如下：

在项目管理窗口下打开“Target1”，选择子项“Source Group1”，并右击鼠标，则出现“Add Files to Group Source Group1”对话框，如图 5-4 所示。

在图 5-4 的对话框中，就可以选择要添加的文件，单击“Add”按钮，即可把所选择文件加到项目中，一次可连续添加多个文件。

如果是已经存在的文件，则添加结束就可以编译连接。如果为新文件，则要先输入文件内容、存盘，然后再编译连接。

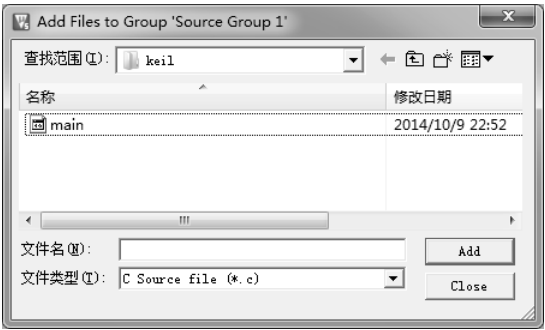


图 5-4 “Add Files to Group Source Group1”对话框

4. 编译、连接项目

用 Project 菜单下的 Built Target 命令对项目中的文件进行编译、连接以便形成目标文件。如果源程序有语法错误，则编译不能通过，并在下面的窗口给出错误类型和行号，修改后，重新编译，如果正确，则编译、连接成功，窗口给出提示信息，如图 5-5 所示。

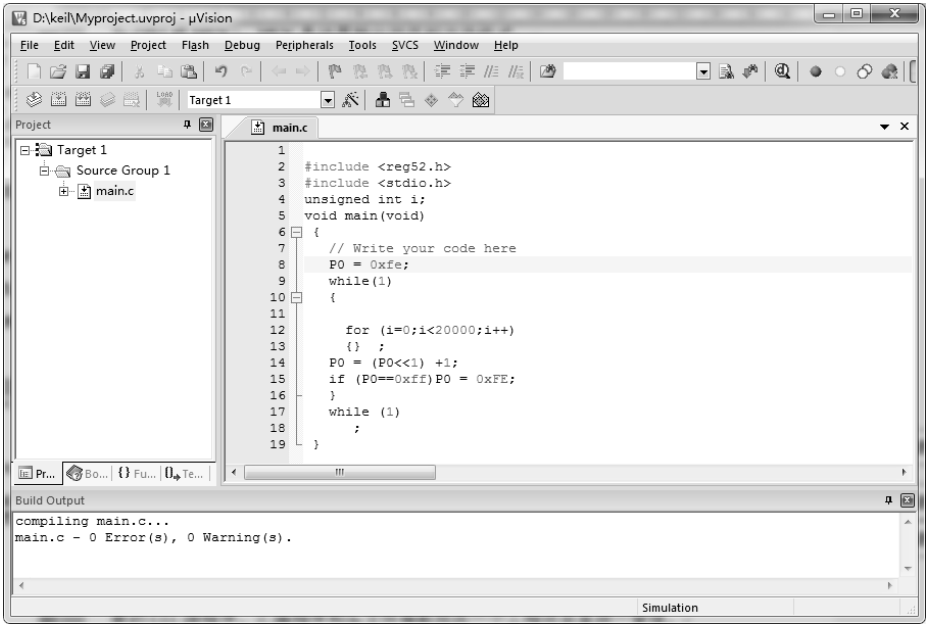


图 5-5 对项目中的文件进行编译、连接后的显示界面

5.4.3 运行调试

当编译、连接成功后就可以开始对程序运行调试。Project 菜单选项 (options for target ...) 可以对软件仿真和硬件仿真进行设置，如图 5-6 所示。硬件仿真时需要相应的驱动程序与硬件仿真器进行通信，以便把各种调试命令发送到硬件仿真器控制单片机的实时硬件仿真，同时可以接收单片机返回的实时数据。执行 Debug 菜单下的 Start/Stop Debug Session 命令启动调试过程。如图 5-7 所示。

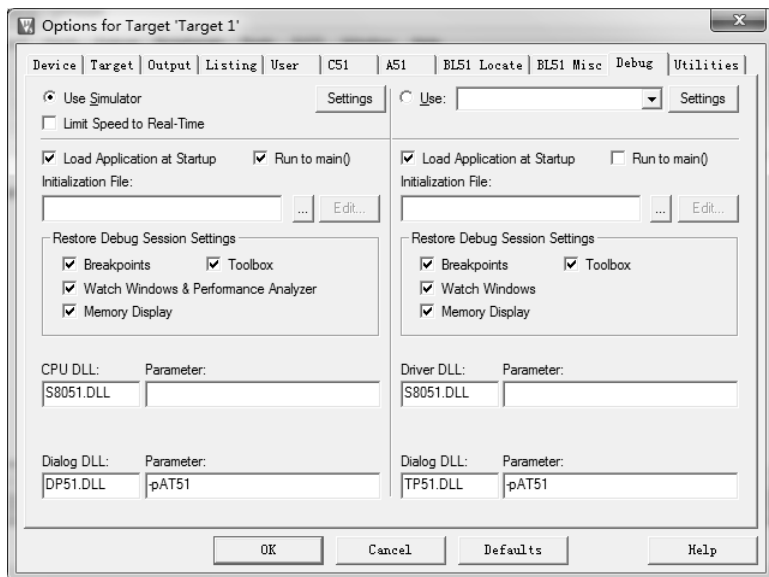


图 5-6 仿真参数设置

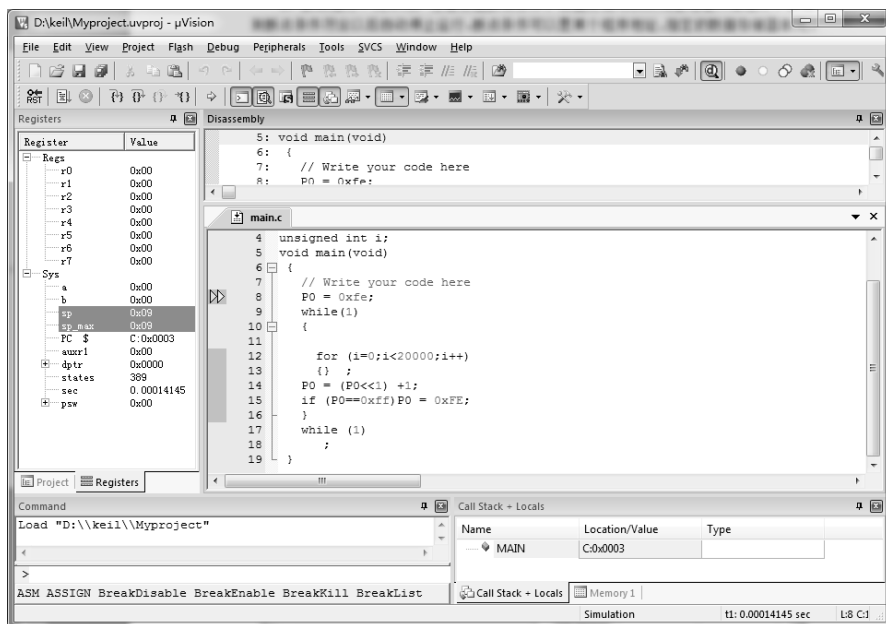


图 5-7 启动调试过程

该系统具有下列主要调试功能。

1. 运行控制功能

（该功能使用户有效地控制应用程序的运行，以便检查程序运行的结果，对存在的硬件故障和软件错误进行定位。主要功能步骤如下：

（1）Run(F5)连续运行：能使 CPU 从指定地址开始连续地全速运行应用程序。

（2）Step(F11)单步运行：能使 CPU 从任意的程序地址开始，每执行一条指令后暂停运行。此时可停下查看寄存器、变量或者存储器等的内容是否正确。

(3) Step Over(F10)过程单步运行：每次执行一条指令，与 Step 功能不同之处是遇到子程序时是连续执行的。

(4) Stop 停止控制：在各种运行方式中，允许用户根据调试的需要，来启动或者停止 CPU 执行应用程序。

(5) 断点运行：允许用户任意设置断点条件，设置好后可以从规定地址开始运行，当碰到断点条件符合以后自动停止运行。断点条件可以是某个程序地址，指定的数据存储器单元，变量达到一定值，有中断产生等。通常是在用单步方法很难调试和发现问题时，采用此方法，这是一种非常有效的调试方法。

2. 对应用系统状态的读出修改功能

可供用户读出/修改的开发系统资源包括：

- (1) 程序存储器（开发系统中的仿真 RAM 存储器或应用机中的程序存储器）；
- (2) 单片机片内资源（工作寄存器、特殊功能寄存器、I/O 口、RAM 存储器、位单元）；
- (3) 系统中扩展的数据存储器、I/O 口。

利用 Watch 窗口可以查看和修改程序变量。如图 5-8 右下方的方框。当 CPU 停止执行应用系统的程序后，光标停在变量上，软件会自动显示出变量内容，也可用鼠标右键添加到 Watch 窗口，用户可方便地读出或修改应用系统所有资源的状态，以便检查程序运行的结果，通过观察窗口内容的变化比较容易判断程序的问题。发现问题后，可重新设置断点条件以及程序的初始参数，再运行调试，直至解决程序中的所有问题。此时再用 Run 连续运行程序，确认结果正确无误，可用 Stop 停止运行，再退出调试过程。

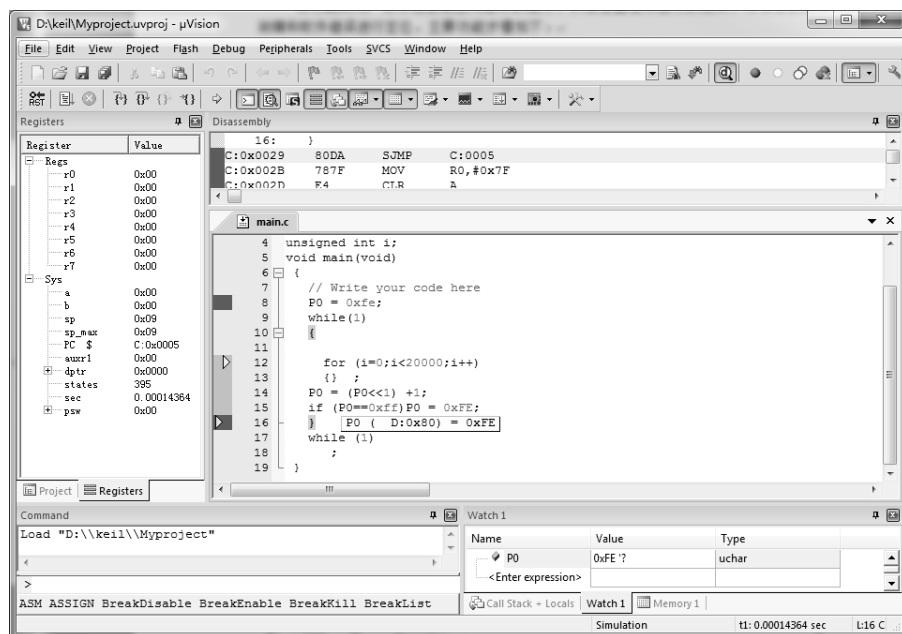


图 5-8 Watch 窗口

当系统程序调试完成后，要生成用以烧录到单片机的 HEX 格式文件，以便写到单片机中。生成 HEX 格式文件的操作如下：

鼠标选中 Target1，单击右键，选择“Option for Target ‘Target’”选项，然后激活 Output，

选择 Creat Hex File 即生成 HEX 格式文件。

5.5 Proteus 硬件仿真环境

Proteus 是英国 Lab Center Electronics 公司出版的电子设计自动化 (EDA) 软件。它不仅是模拟电路、数字电路、模/数混合电路的设计与仿真平台,还能仿真包括外围接口、模数混合电路在内的单片机应用系统,可直接验证硬件设计中的大多数问题。因此,Proteus 在国内已经得到广泛应用。

5.5.1 Proteus 软件简介

Proteus 软件是目前世界上最先进、最完整的多种型号微控制器系统的硬件设计与仿真平台。他是目前唯一能够对处理器进行实时仿真调试与测试的 EDA 工具,真正实现了没有系统原型在计算机上就可完成从原理图设计、电路分析、对系统的软件仿真和硬件仿真调试、测试与验证,直至形成 PCB 的完整的电子设计和研发过程。使用 Proteus 软件进行单片机系统仿真设计,有利于培养学生的电路设计能力及仿真软件的操作能力,在不需要硬件投入的条件下,解决了传统的电子设计流程费时、费力、而且费用高等问题,缩短了产品的开发周期,降低了开发风险,可以更快更有效地掌握单片机技术,是学校进行教学的首选软件。

Proteus 软件具有以下主要特点。

(1) 实现了单片机仿真和 SPICE (一种电路模拟软件) 电路仿真相结合。具有模拟电路及数字电路仿真、单片机及其外围电路系统的仿真。还可对 RS232 等各种接口、键盘和 LCD 系统等仿真,支持多种虚拟仪器等。

(2) 支持主流单片机系统的仿真。可以提供足够接近实时的仿真硬件环境。目前支持的处理器模型包括 8051、PIC、ARM 和 MSP430 等系列,还支持各种相关的外围芯片。

(3) 提供软件调试功能。在硬件仿真系统中具有全速、单步、设置断点等调试功能,还支持第三方的软件编译和调试环境,如 Keil C51 和 Proton 等软件。

(4) 具有强大的原理图绘制功能,具有“线路自动路径”功能,支持多层次图设计。

Proteus 软件由 ISIS 和 ARES 两个软件构成,其中 ISIS 是智能原理图输入系统,是系统设计与仿真的基本平台,ARES 是高级 PCB 布线编辑软件,本节主要介绍 ISIS 软件。

5.5.2 Proteus ISIS 窗口功能

在运行 Proteus 8 Professional 后进入 Proteus 主界面,点击图中的 ISIS 图标就进入 Proteus ISIS 的工作界面,如图 5-9 所示。其主要包括标题栏、主菜单栏、图形编辑窗口、预览窗口、对象选择器窗口、主工具栏、仿真进程控制按钮、绘图工具栏。各窗口功能简介如下:

1. 主菜单栏

Proteus ISIS 的主菜单栏包括 File (文件)、Edit (编辑)、View (视图)、Library (库)、Tools (工具)、Design (设计)、Graph (图形)、Debug (调试)、Template (模板)、System (系统) 和 Help (帮助) 等共 11 项,如图 5-9 所示。单击任一菜单后都将弹出其子菜单项。使用者可根据需要选择该级菜单中的选项。菜单展开的下一级菜单中有许多常用操作在工具栏中

有相应的快捷键，有些命令的右方还标注了该命令的快捷键，例如 Redraw Display（刷新命令）的快捷键为 R 等。各子菜单主要功能如下：

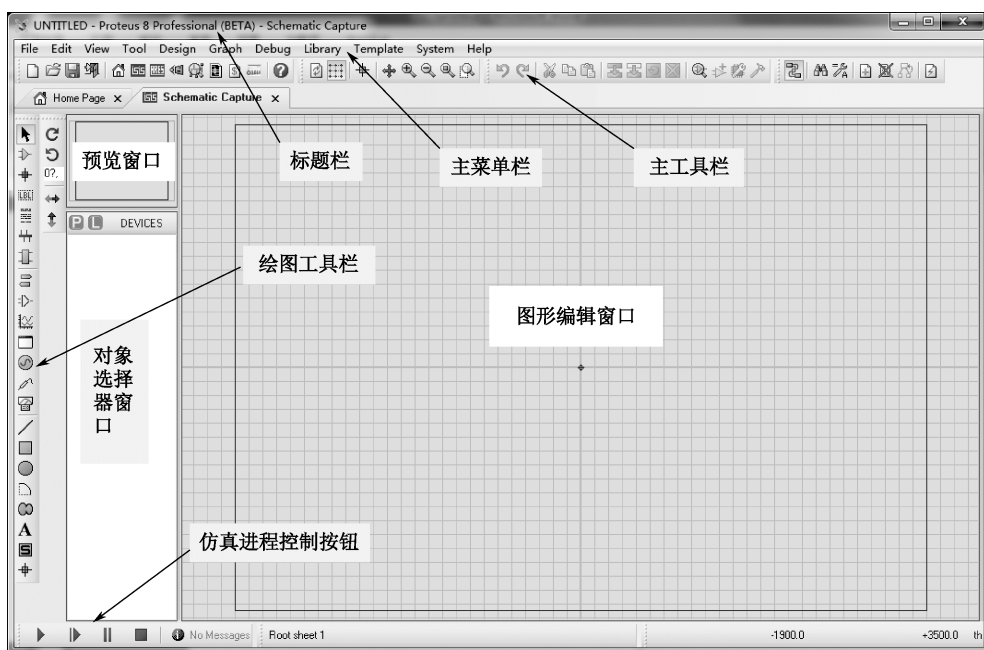


图 5-9 ISIS 的工作界面

(1) File（文件）菜单：包括常用的文件功能，如新建设计、打开设计、保存设计、导入/导出文件，也可打印、显示设计文档，以及退出 Proteus ISIS 系统等。

(2) Edit（编辑）菜单：包括撤销/恢复操作、查找与编辑元器件、剪切、复制、粘贴对象，以及设置多个对象的层叠关系等。

(3) View（显示）菜单：包括是否显示网格、设置格点间距、缩放电路图及显示与隐藏各种工具栏等。

(4) Library（库操作）菜单：包括选择元器件及符号、制作元器件及符号、设置封装工具、分解元件、编译库、自动放置库、校验封装和调用库管理等。

(5) Tools（工具）菜单：包括实时注解、自动布线、查找并标记、属性分配工具、全局注解、导入文本数据、元器件清单、电气规则检查、编译网络标号、编译模型、将网络标号导入 PCB 以及从 PCB 返回原理设计等工具栏。

(6) Design（工程设计）菜单：包括编辑设计属性，编辑原理图属性，编辑设计说明，配置电源，新建，删除原理图，在层次原理图中总图与子图以及各子图之间互相跳转和设计目录管理等功能。

(7) Graph（图形）菜单：它具有编辑仿真图形，添加仿真曲线、仿真图形，查看日志，导出数据，清除数据和一致性分析等功能。

(8) Debug（调试）菜单：包括启动调试、执行仿真、单步运行、断点设置和重新排布弹出窗口等功能。

(9) Template（模板）菜单：包括设置图形格式、文本格式、设计颜色以及连接点和图形等功能。

(10) System（系统设置）菜单：包括设置系统环境、路径、图纸尺寸、标注字体、热键

以及仿真参数和模式等功能。

(11) Help (帮助) 菜单: 包括版权信息、Proteus ISIS 学习教程和示例等内容。

2. 图形编辑窗口

编辑窗口用于放置元器件, 进行连线, 绘制原理图。设计电路及各种符号、元器件模型等, 是电路系统的仿真平台。窗口中的蓝色框内是可编辑区, 所有的硬件设计在此框内完成。

3. 对象选择器窗口

对象选择器用于选择元器件、终端、图表、信号发生器及虚拟仪器等。该窗口上方左上角有 PL。其中“P”为对象选择按钮, “L”为库管理按钮。窗口右上方显示内容为当前所处模式及其下所列对象类型, 如当前所处模式为元器件, 则显示 DEVICES。

4. 预览窗口

预览窗口可显示如下 2 个内容。

(1) 当单击对象选择器中的某个对象时, 预览窗口就会显示该对象的符号。

(2) 当鼠标在图形编辑窗口操作时, 在预览窗口中会出现一个蓝色方框与一个绿色方框。蓝色方框中是编辑区中原理图的缩略图, 绿色方框内是编辑区中当前在屏幕上的可见部分。

5. 主工具栏

Proteus ISIS 的主工具栏位于主菜单下面两行, 以图标形式给出, 包括 File 工具栏、View 工具栏、Edit 工具栏和 Design 工具栏四个部分。工具栏中每一个按钮, 都对应一个具体的菜单命令, 主要目的是为了快捷而方便地使用命令。

6. 仿真进程控制按钮

从左至右所表达的功能依次为: 运行、单步运行、暂停和停止。

7. 绘图工具栏

绘图工具栏主要包括模型选择、配件和各种图形模型。

5.5.3 Proteus ISIS 的基本操作

Proteus ISIS 运行于 Windows 环境, 本节以 4.2 节的例 10(其 C 语言程序是 5.3 节的例 3)说明如何快速使用 Proteus ISIS 原理图输入系统绘制电路原理图及仿真。

1. 创建工程文件

在图 5-9 所示的 Proteus 8 主界面中选择“File→New Project”命令或快捷图标, 即弹出“New Project Wizard”对话框, 如图 5-10 所示。

在该窗口中修改项目名称为 Myproject, 并修改路径至 D 盘下后, 就形成一个路径如图 5-10 所示的新的原理图项目文件, 单击“Next”按钮, 进入原理图设计窗口。此时。在该窗口中选择“Create a schematic from the selected template”和“Landscape A4”模板, 单击“Next”

按钮，进入 PCB 选项，本例不需要 PCB 板，选择默认设置，单击“Next”按钮，进入 Fireware（固件）窗口，如图 5-11 所示。

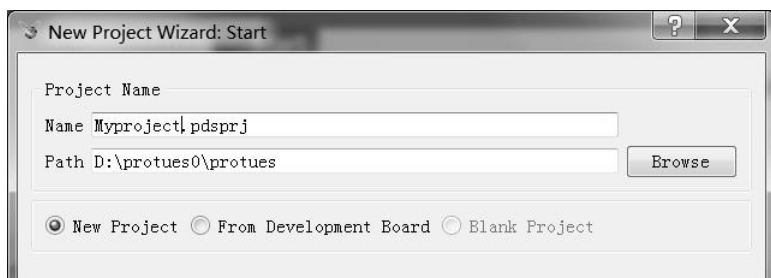


图 5-10 新项目向导对话框

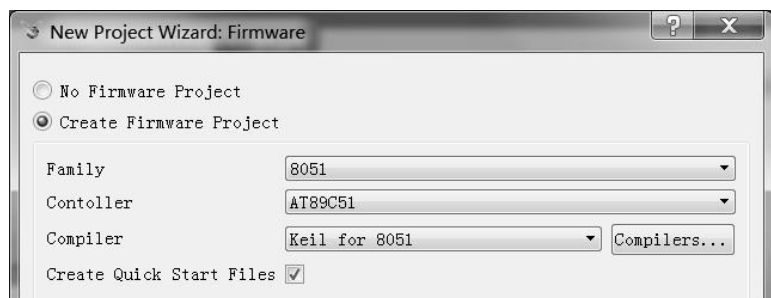


图 5-11 Firmware 对话框

在该窗口中选择 Create Firmware Project，从 Family 下拉列表框里选择 8051 系列，从 Controller 下拉列表框里选择 AT89C51 型号，从 Compiler 下拉列表框里选择 Keil for 8051（在本机中事先要装入 keil C51），单击下方 Next 按钮，出现如图 5-12 所示的 Summary 图，单击“Finish”按钮，完成工程向导，此时已自动生成项目原理图和源代码模板文件。

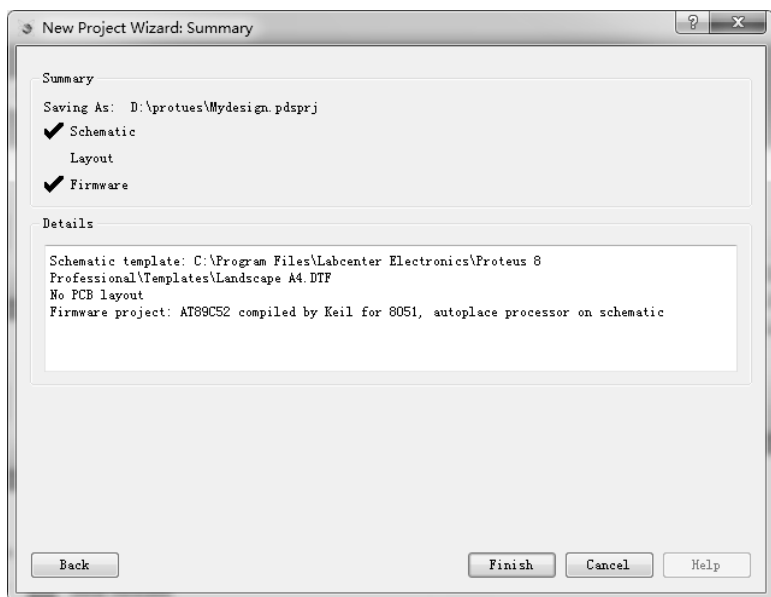


图 5-12 完成工程向导

2. 绘制电路图

1) 选择元件

在完成工程向导后,即出现图 5-13 所示原理图文件,图中已经放入了向导选择的单片机 AT89C51。图 5-13 中点状的栅格区域为编辑窗口,左上方为预览窗口,左下方为元器件列表区,即对象选择器。

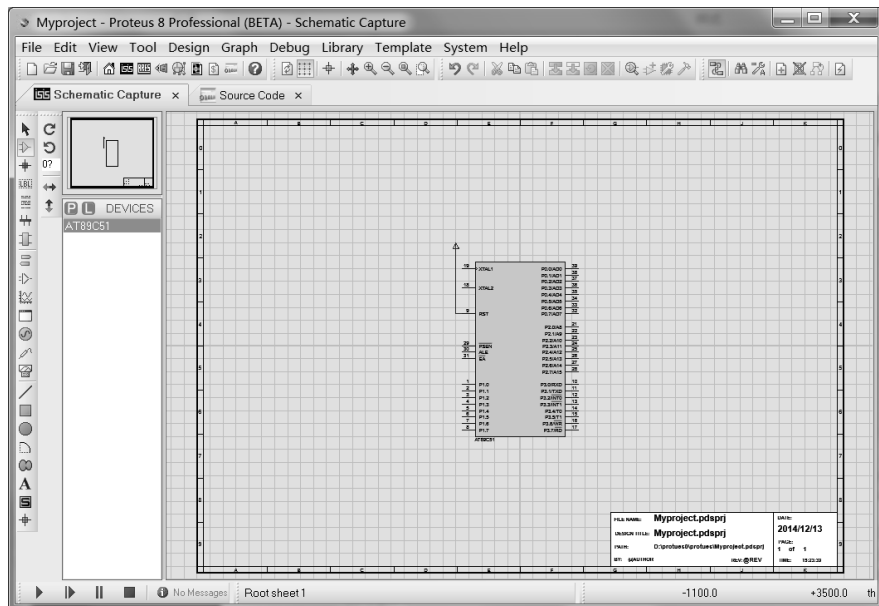


图 5-13 原理图编辑窗口

在预览窗口中,有两个框,蓝框表示当前页的边界,绿框表示当前编辑窗口显示的区域。当从对象选择器中选一个新的对象时,预览窗口可以预览选中的对象。在预览窗口上单击,Proteus ISIS 将会以单击位置为中心刷新编辑窗口。其他情况下,预览窗口显示将要放置的对象。

选择 Library-Pick Device--Symbol 命令或者点击左下方为元器件列表区的 P,弹出元件窗口,如图 5-14 所示,窗口左侧分类列出元件目录,便于查找。也可以在窗口左上方 Keywords 中写入关键词,窗口中间会列出结果,窗口右侧是器件的原理图和封装图。注意:在 Keywords 文本框中输入关键词时最好在 Category 中选择 All Categories,因为关键词搜索的依据是 Category 中的类别。如选择“RSE”(电阻),单击“OK”按钮,元件名出现在左侧的 Devices 列表中。

从元件库中选择本电路文件要用的器件放入列表中就可以画原理图了。

2) 放置与调整元件

在 Devices 列表中点击需要摆放的元件,左上方的略缩图中显示该元件的图形,在图形编辑窗口中单击鼠标,元件即出现在该位置上。例如选择电阻“RES”,在图形编辑窗口连续单击鼠标左键两次,出现两个电阻“RES”,用此方法可以加到 8 个电阻,如图 5-15 所示。用同样方法摆放其他元件。默认情况下摆放的元件方向固定,如果要改变元件方向,可以使用左下角的旋转与翻转命令。

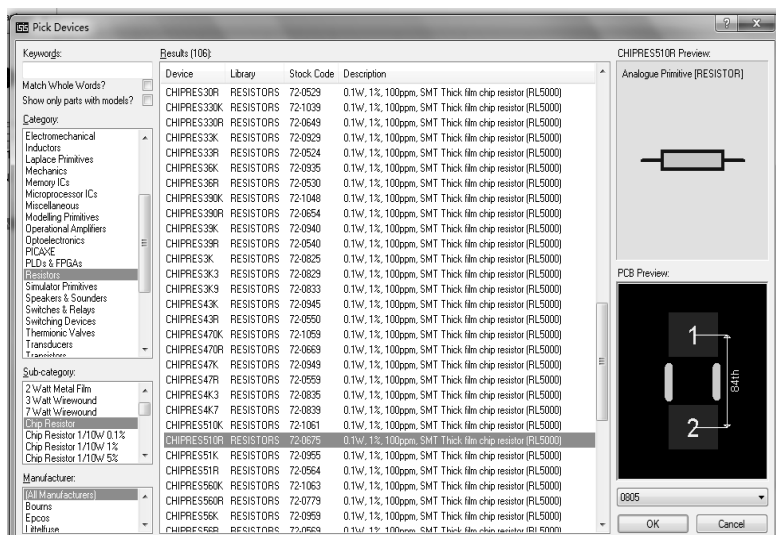


图 5-14 选择元件窗口

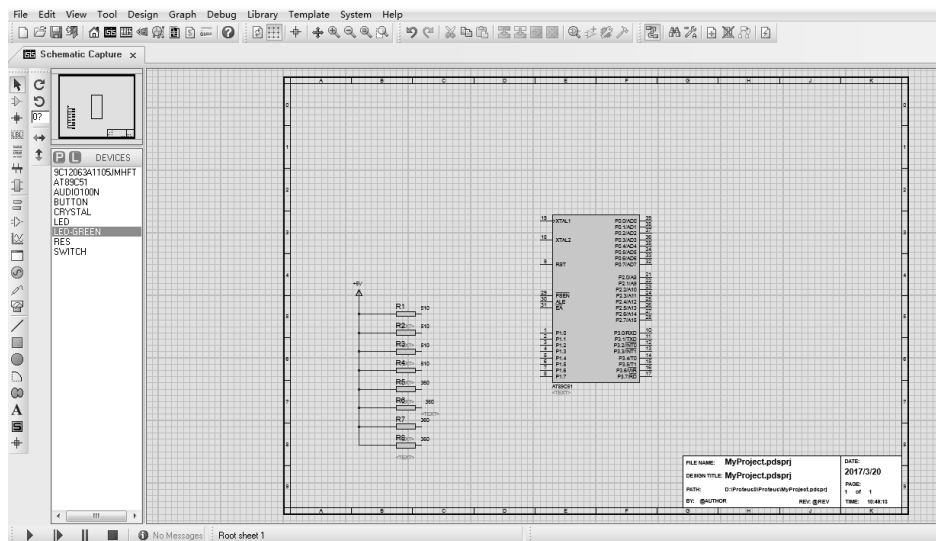


图 5-15 选择放置元件

在左侧工具栏中单击 (Terminals MODE) 图标，列表框中显示可用的终端，单击 POWER 摆放电源终端，单击 GROUND 摆放接地终端，摆放方法如同一般元件。

3) 连接导线

Proteus 支持自动布线，分别单击两个管脚（不管这两个管脚在何处），两个管脚之间会自动添加走线，还可以手动走线，连接走线后电路如图 5-16 所示。

在电源终端上双击，出现“Edit Terminal Label”对话框，在其中输入对应的电源符号如 VCC。执行“Design→Configure Power Rails”命令，出现“Power Rail Configuration”对话框，通过此对话框可对不同符号的电源输入确定的电压值。

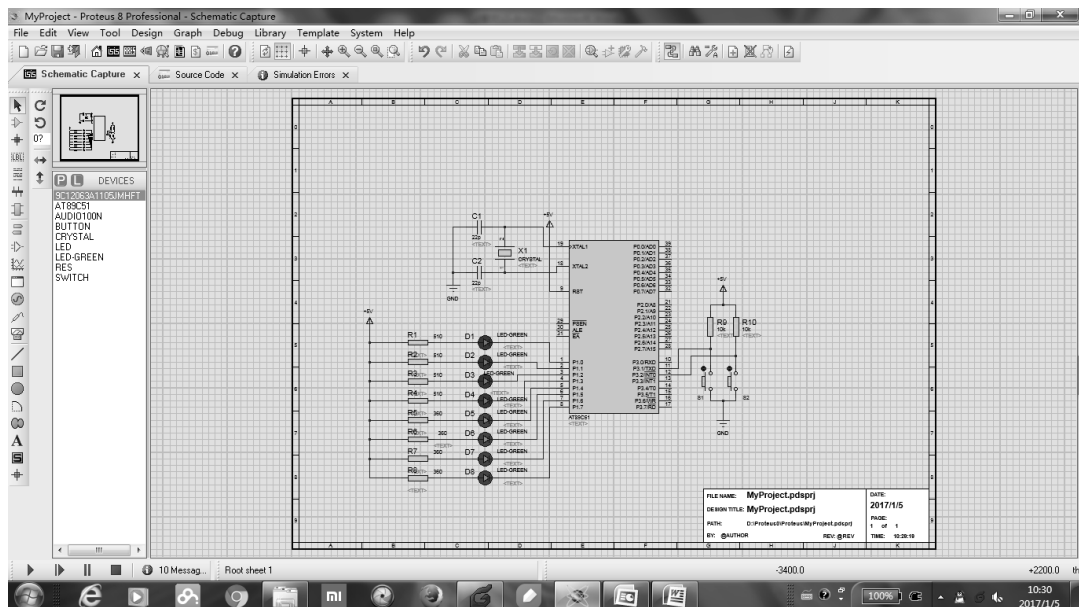


图 5-16 完成后的原理图

3. 加载源文件及调试

1) 加载或编辑源文件

单击图 5-16 中的 Source Code 选项，出现编辑窗口，如图 5-17 所示。

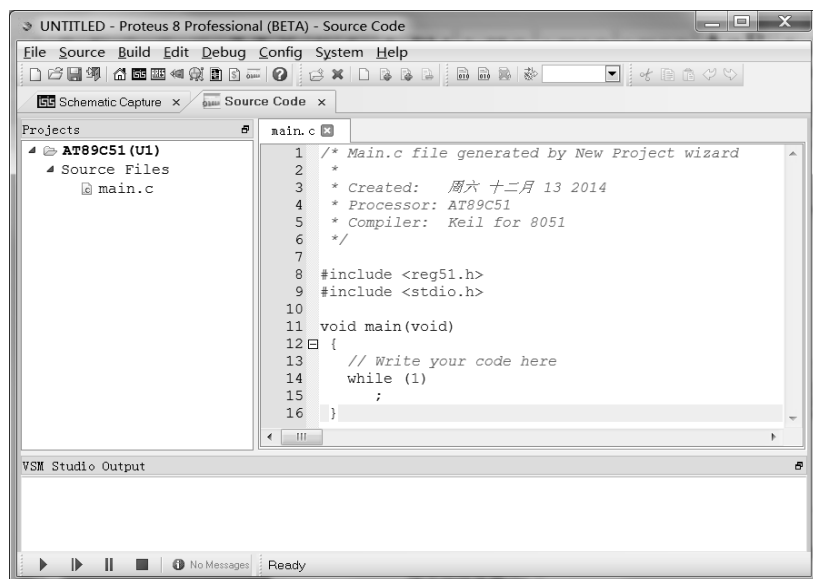


图 5-17 源文件编辑窗口

右击图 5-17 中 AT89C51 文件夹，弹出图 5-18 所示菜单。此时可选择导入在 keil C51 中已经编好的源文件，也可以直接在编辑窗口中录入源文件。

单击 Building→Build Project 按钮，对源代码进行编译，如出现错误，按提示进行修改，直至系统提示编译成功。

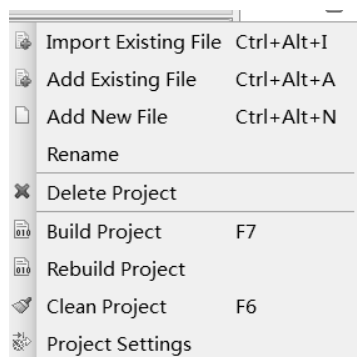



图 5-18 文件处理菜单

2) 仿真调试

虽然 Proteus 自身带有汇编编译器,但其调试功能不如 Keil C51,且不支持 C 语言的编译(在 Proteus 8 Professional 版本中支持 C 语言),所以通常是在 Keil C51 集成开发环境调试。Proteus 支持 ISIS 和 Keil 的联合调试。

联合调试步骤如下:

- 安装 Proteus 的 Keil 联机调试驱动程序 vdmagdi.exe。
- 在 ISIS 中画出正确的电路原理图。
- 在 ISIS 中打开工程文件,选择“Debug”→“Use Remote Debug Monitor”或“Enable Remote Debug Monitor”(在 Proteus 8 Professional 版本中)开启远程调试模式。
- 在 Keil 软件的主界面上,单击快捷按钮 ,或者“Flash”→“Configure Flash Tools...”,弹出“Options for Target”对话框,如图 5-19 所示。选中右侧的“Use”单选按钮,并在单片机仿真器列表中选择“Proteus VSM Simulator”,其余采用默认值即可。然后单击“Settings”按钮,并在弹出的“VDM51 Target Setup”对话框进行设置,如图 5-19 所示。

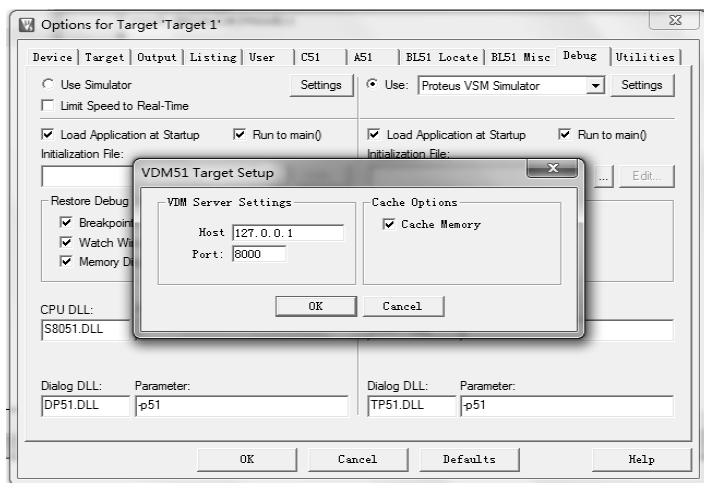


图 5-19 VDM51 配置情况

- 打开 ISIS 电路原理图,然后即可在 Keil 环境中运行和调试该程序。
- 在 Keil 中将程序编译正确后,即可运行程序,此时 Keil 中的程序即可以控制 ISIS 中

设计的电路，即实现了 ISIS 和 Keil 的联合调试。

如果程序与电路均正确，则用鼠标点击一下模拟开关按键，即可观察到屏幕上 LED 按照程序要求闪烁点亮。按停止仿真按钮，停止运行。如果仿真结果不正确，此时可以采取单步调试，设置断点等方法，并通过调试菜单观察单片机的特殊功能寄存器、存储器中的内容变化，观察窗口与仿真电路可同时实时显示更利于发现问题。

思考题与练习

1. 在单片机领域，目前最广泛使用的是哪种语言？有哪些优越性？这种语言单片机能否直接执行？

2. 有哪些数据类型是 80C51 系列单片机直接支持的？C51 语言特有的数据类型是哪些？

3. 在 C51 语言中有哪几种存储类型？分别表示哪些存储器区域？

4. 在 C51 语言中 bit 位与 sbit 位有什么区别？

5. 在 C51 语言中中断函数与一般函数有什么不同？

6. 循环结构程序有何特点？80C51 系列单片机的循环转移指令有何特点？请用汇编语言和 C 语言编写下列程序。编程时应注意些什么？

(1) 请编写延时 1 秒的延时程序段，主频为 6MHz。

(2) 请编写多字节十进制（BCD 码）减法程序段。

(3) 请编写多字节无符号十进制数（BCD 码）除法程序段。并画出程序流程图。

特别注明：本章所列举代码均采用西文半角字符。

第 6 章 定时/计数器

为实现定时控制以及对外界事件进行计数，在单片机应用系统中，常需要用到实时时钟和计数器。80C51 系列单片机内部的定时/计数器都具有这两种功能，有的型号还具有输入捕获和监视定时功能。本章将介绍定时/计数器的结构、原理、工作方式及使用方法。

6.1 定时/计数器 T0、T1 概述

所有属于 80C51 系列的单片机内部都设有两个 16 位的可编程定时/计数器，可简称为定时器 0 (T0) 和定时器 1 (T1)。不论哪一种型号，T0、T1 的结构、原理和工作方式都是相同的。可编程是指其功能如工作方式、定时时间、量程、启动方式等均可由指令来确定和改变。

6.1.1 定时器/计数器 T0、T1 的结构

定时器的原理结构框图如图 6-1 所示。虚线框内即为定时器 T0、T1 的结构图，它们通过内部总线与 CPU 相接，此外由 TCON 寄存器中还引出 2 根中断源信号线送入 CPU。

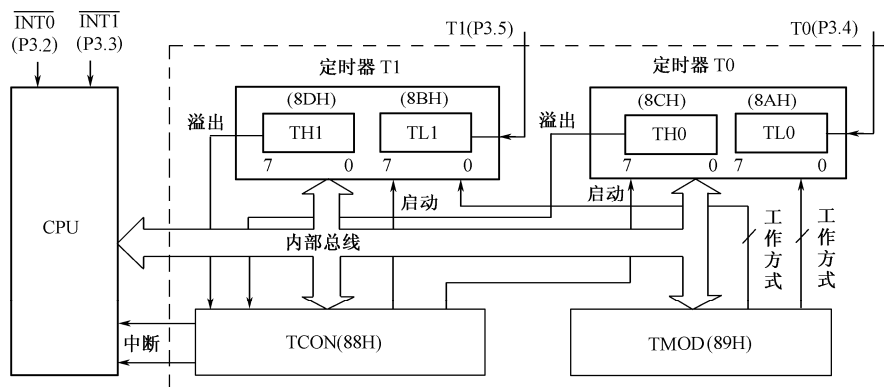


图 6-1 80C51 定时器/计数器 T0、T1 原理结构框图

从定时/计数器的结构图可以看出，与定时器有关的 8 位寄存器一共有 6 个，这些寄存器之间是通过内部总线和控制逻辑电路连接起来的。其中 16 位的定时/计数器分别由两个 8 位专用寄存器组成，即 T0 由 TH0 和 TL0 构成；T1 由 TH1 和 TL1 构成。其访问地址依次为 8AH~8DH。每个寄存器均可单独访问，这些寄存器是用于存放定时或计数初值的。除了这两个 16 位的计数器之外，在定时器中还有两个特殊功能寄存器，一个是 8 位定时器方式寄存器 TMOD，另一个是 8 位定时器控制寄存器 TCON。TMOD 主要是用于选定定时器的工作方式；TCON 主要是用于控制定时器的启动与停止。此外，TCON 还可保存 T0、T1 的溢出和中断标志。当定时器工作在计数方式时，外部事件通过引脚 T0 (P3.4) 和 T1 (P3.5) 输入。

6.1.2 定时/计数器的原理

16 位的定时/计数器实质上是一个加 1 计数器，其控制电路受软件控制、切换。通过软件可以设置为 4 种工作方式（详见 6.3 节），每种方式都可以用做定时或计数。

当选择定时/计数器作为定时器工作时，计数器的加 1 信号由振荡器的 12 分频信号产生，即每过一个机器周期，计数器增 1，直至计满溢出为止。显然，定时器的定时时间与系统的振荡频率有关。因一个机器周期等于 12 个振荡周期，所以计数频率 $f_{\text{COUNT}} = 1/12 \times f_{\text{OSC}}$ 。如果晶振为 12MHz，则计数周期为：

$$T = \frac{1}{(12 \times 10^6) \text{Hz} \times 1/12} = 1\mu\text{s}$$

这是最短的定时周期，若要延长定时时间，则需改变定时器的初值，并要适当选择定时器的长度（例如，8 位、13 位或 16 位等）。

当选择定时/计数器作为计数器工作时，通过引脚 T0 和 T1 对外部信号计数，外部脉冲的下降沿将触发计数。计数器在每个机器周期的 S5P2 期间采样引脚输入电平。若一个机器周期采样值为 1，下一个机器周期采样值为 0，则计数器加 1。此后的机器周期 S3P1 期间，新的计数值装入计数器。所以检测一个由 1 至 0 的跳变需要两个机器周期，故外部事件的最高计数频率为振荡频率的 1/24。例如，如果选用 24MHz 晶振，则最高计数频率为 1MHz。虽然对外部输入信号的占空比无特殊要求，但为了确保某给定电平在变化前至少被采样一次，则外部计数脉冲的高电平与低电平保持时间均需在一个机器周期以上。

当用软件给定时器设置某种工作方式之后，定时器就会按设定的工作方式自动运行，不再占用 CPU 的操作时间，除非定时器计满溢出，才可能中断 CPU 的当前操作。CPU 也可以随时重新设置定时器工作方式，以改变定时器的操作。由此可见，定时器是单片机中效率高而且工作灵活的部件。

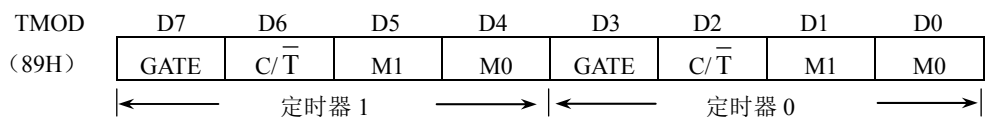
6.2 定时/计数器的控制方法

综上所述，可知定时/计数器是一种可编程部件，它不会自动开始工作，必须通过软件确定它的工作方式，并启动它开始工作。所以在定时/计数器开始工作之前，CPU 必须将一些命令（称为控制字）写入定时/计数器。将控制字写入定时/计数器的过程叫定时/计数器的初始化。在初始化程序中，要将工作方式控制字写入方式寄存器，工作状态控制字（或相关位）写入控制寄存器，赋定时/计数初值。本节将详述这些控制字的格式和各位的功能及怎样通过控制字的写入，控制定时/计数器的工作。

6.2.1 定时/计数器寄存器

1. 工作方式寄存器 TMOD

TMOD 为 T0、T1 的工作方式寄存器，其格式如下：



各位功能如下：

(1) M1 和 M0——方式选择位

由 M1 和 M0 组合可以定义 4 种工作方式，如表 6-1 所示。

表 6-1 定时器工作方式选择表

M1	M0	工 作 方 式	功 能 描 述
0	0	方式 0	13 位计数器
0	1	方式 1	16 位计数器
1	0	方式 2	自动再装入 8 位计数器
1	1	方式 3	定时器 0：分成两个 8 位计数器 定时器 1：停止计数

有关这 4 种工作方式的详细介绍见 6.3 节。

(2) C/ \overline{T} ——功能选择位

当 C/ \overline{T} =0 时，为定时器方式；

当 C/ \overline{T} =1 时，为计数器方式。

(3) GATE——门控位

当 GATE = 0 时，只要控制位 TR0 或 TR1 置 1，即可启动相应定时器开始工作；

当 GATE = 1 时，除需要将 TR0 或 TR1 置 1 外，还需要使 $\overline{INT0}$ 或 $\overline{INT1}$ 引脚为高电平时，才能启动相应的定时器开始工作（详细介绍见 6.3 节）。

TMOD 不能进行位寻址，只能用字节传送指令设置定时器的工作方式，低半字节定义定时器 0，高半字节定义定时器 1。复位时，TMOD 所有位均为 0，定时器处于停止工作状态。

为说明方式字的应用，举例如下：

设定时器 1 为计数工作方式，要求由软件启动定时器 1，按方式 1 工作。定时器 0 为定时方式，要求由软件启动定时器 0 工作，按方式 3 工作。根据 TMOD 各位的作用可知命令字为 01010011B，其指令形式为：

MOV TMOD, #53H。

2. 定时/计数器控制寄存器 TCON

TCON 的作用是控制定时器的启、停，标志定时器的溢出和中断情况。定时器控制字 TCON 格式如下。

TCON	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
(88H)	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

各位功能如下：

TCON.7 TF1——定时器 1 溢出标志。

当定时器 1 计满溢出时，由硬件使 TF1 置 1，并且申请中断。进入中断服务程序后，由硬件自动清 0，在查询方式下用软件清 0。

TCON.6 TR1——定时器 1 运行控制位。

当 TR1 = 1 时，启动定时器 1 工作；

当 TR1 = 0 时，关闭定时器 1。

TCON.5 TF0——定时器 0 溢出标志。

其功能及操作情况同 TF1。

TCON.4 TR0——定时器 0 运行控制位。

其功能及操作情况同 TR1。

TCON.3 IE1——外部中断 1 请求标志。

TCON.2 IT1——外部中断 1 触发方式选择位。

TCON.1 IE0——外部中断 0 请求标志。

TCON.0 IT0——外部中断 0 触发方式选择位。

TCON 中低 4 位与中断有关，将在中断章节中详细讨论。

当系统复位时，TCON 的所有位均清 0。

由于 TCON 是可以位寻址的，因而，如果只清溢出或启动定时器工作，可以用位操作指令。例如，执行“CLR TF0”指令后，就清定时器 0 的溢出。执行“SETB TR1”指令后，即可启动定时器 1 开始工作（当然前面还要设置方式字）。

6.2.2 定时/计数器的初始化

由于定时/计数器的功能是由软件编程确定的，所以一般在使用定时/计数器前都要对其进行初始化，使其按设定的功能工作。初始化步骤如下。

① 确定工作方式——对 TMOD 赋值。

② 预置定时或计数的初值——可直接将初值写入 TH0、TL0 或 TH1、TL1。

③ 根据需要开放定时/计数器的中断——直接对 IE 寄存器的定时器中断位赋值（详见第 8 章）。

④ 启动定时/计数器工作——若已规定用软件启动，则可把 TR0 或 TR1 置“1”；若已规定由外中断引脚电平启动，则需给外引脚加启动电平。当实现了启动要求之后，定时器即按规定的工作方式和初值开始计数或定时。

6.2.3 定时/计数器初值的确定方法

因为不同工作方式下，计数器位数不同，因而最大计数值也不同。下面介绍确定定时/计数器初值的具体方法：

现假设最大计数值为 M ，那么各方式下的 M 值如下：

方式 0: $M=2^{13}=8192$

方式 1: $M=2^{16}=65536$

方式 2: $M=2^8=256$

方式 3: 定时器 0 分成两个 8 位计数器，所以两个 M 均为 256。

因为定时/计数器是做“加 1”计数，并在计满溢出时产生中断，因此初值 X 可以这样计算：

$$X = M - \text{计数值}$$

现举例说明定时初值的计算方法，若 80C51 时钟频率为 6MHz，要求产生 1ms 的定时，试计算初值。

在时钟频率为 6MHz 时，计数器每“加 1”一次所需的时间为 2μs。如果要产生 1ms 的定时时间，则需“加 1”500 次。那么 500 即为计数值，如果要求在方式 1 情况下工作，则初值 $X = M - \text{计数值} = 65536 - 500 = 65036 = \text{FE0CH}$ 。

上式表示如果初值为 65036 时，再计 500 个脉冲，达到了 65536，则定时器产生溢出。一旦溢出，计数器中的值就变为 0，在时钟频率为 6MHz 时，正好产生 1ms 的定时时间。如果下一次计数从 0 开始，定时时间就不是 1ms 了，所以在定时溢出后，要马上把 65036 送入计数器。但是对于具有自动重装载功能的方式 2，不必由用户软件重新装入初值，详见 6.5 节。

6.3 定时/计数器 T0、T1 的工作方式

由上节可知通过对 M1、M0 位的设置，T0 可选择 4 种工作方式，T1 可选择 3 种工作方式。本节将介绍其工作方式的结构、特点及工作过程。

6.3.1 方式 0

定时器 T0、T1 都可以设置为方式 0，在方式 0 下定时器 1 与定时器 0 的结构和操作完全相同，均为 13 位的计数器。由于采用方式 0 计算初值时，比较麻烦、容易出错，一般情况下尽量避免采用此方法。它是为了与其早期产品兼容而保留下来的功能，在实际应用中完全可以用方式 1 代替它。为节省篇幅本书不予介绍，可见参考文献[2]。

6.3.2 方式 1

定时器 T0、T1 都可以设置为方式 1，在方式 1 下，定时器 T0、T1 均为 16 位的计数器。在方式 1 下定时器的结构和操作与方式 0 基本相同，唯一的差别是：在方式 1 中，定时器是以全 16 位二进制数参与操作的。图 6-2 是 T0 在方式 1 时的逻辑电路结构，当 TL0 的低 8 位溢出时向 TH0 进位，而 TH0 溢出时向中断标志 TF0 进位（称硬件置位 TF0），并申请中断。通过查询 TF0 是否置位，或是否产生定时器 0 中断，可判断定时器 0 计数是否溢出。

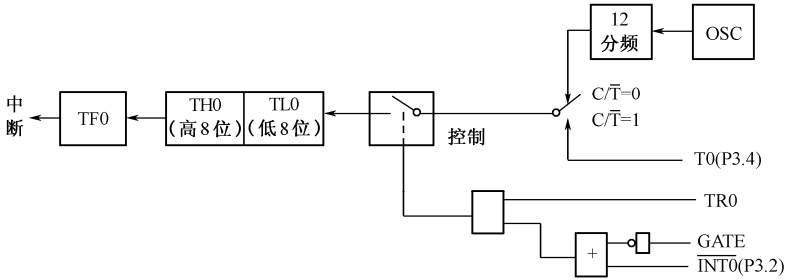


图 6-2 T0（或 T1）方式 1 结构

下面通过图 6-2 进一步说明方式 1 的工作原理。

当 $C/\bar{T} = 0$ 时，多路开关连接振荡器的 12 分频器输出，T0 对机器周期计数，这就是定时工作方式。其定时时间为：

$$t = (2^{16} - T0 \text{ 初值}) \times \text{时钟周期} \times 12$$

当 $C/\overline{T}=1$ 时，多路开关与引脚 $T0$ ($P3.4$) 相连，外部计数脉冲由引脚 $T0$ 输入，当外信号电平发生 1 到 0 跳变时，计数器加 1，这时 $T0$ 成为外部事件计数器。

当 $GATE=0$ 时，封锁“或”门，这时，“或”门输出常 1，使引脚 $\overline{INT0}$ 输入信号无效。打开“与”门，此时由 $TR0$ 控制定时器 0 的开启和关断。若 $TR0=1$ ，接通控制开关，启动定时器 0 工作，允许 $T0$ 在原计数值上做加法计数，直至溢出。溢出时，计数寄存器值为 0， $TF0$ 置位，并申请中断， $T0$ 从 0 开始计数。因此，如果希望计数器按原计数初值开始计数，在计数溢出后，应给计数器重新赋初值。如果 $TR0=0$ ，则关断控制开关，停止计数。

当 $GATE=1$ ，且 $TR0=1$ 时，或门、与门全部打开，外信号电平通过 $\overline{INT0}$ 引脚直接开启或关断定时器计数。输入 1 电平时，允许计数，否则停止计数。这种操作方法可用来测量外信号的脉冲宽度等。

6.3.3 方式 2

定时器 $T0$ 、 $T1$ 都可以设置为方式 2，在方式 2 下定时器 1 的结构和操作与定时器 0 完全相同。方式 2 设置定时器为能重置初值的 8 位定时/计数器。方式 0、方式 1 若用于循环重复定时/计数时（如产生连续脉冲信号），每次计满溢出，寄存器全部为 0，第二次计数还得重新装入计数初值。这样不仅在编程时麻烦，且影响定时时间精度。而方式 2 有自动恢复初值（初值自动再装入）功能，避免了上述缺陷，适合用做较精确的定时脉冲信号发生器。其定时时间为：

$$t = (2^8 - TH0 \text{ 初值}) \times \text{时钟周期} \times 12$$

在方式 2 中，16 位的计数器被拆成两个，见图 6-3。 $TL0$ 用做 8 位计数器， $TH0$ 用以保持初值。在程序初始化时， $TL0$ 和 $TH0$ 由软件赋予相同的初值。一旦 $TL0$ 计数溢出，则置位 $TF0$ ，并将 $TH0$ 中的初值再装入 $TL0$ ，继续计数，重复循环不止。

这种工作方式可省去用户软件中重装常数的程序，并可产生相当精确的定时时间，特别适用于作串行口波特率发生器（详见第 7 章）。

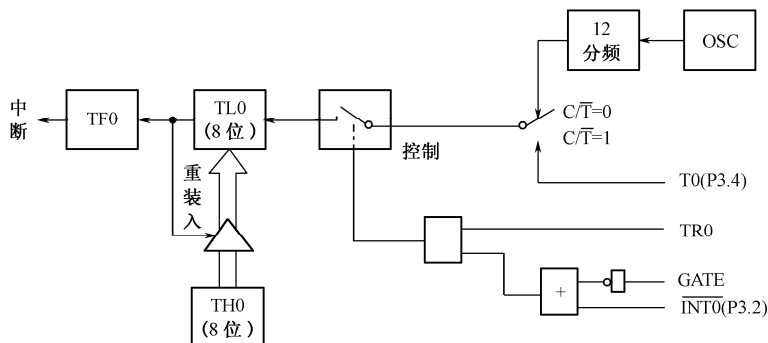


图 6-3 $T0$ （或 $T1$ ）方式 2 结构

6.3.4 方式 3

只有定时器 $T0$ 可以设置为方式 3。定时器 $T0$ 在方式 3 下被拆成两个独立的 8 位计数器 $TL0$ 和 $TH0$ ，见图 6-4。其中 $TL0$ 用原 $T0$ 的控制位、引脚和中断源，即 C/\overline{T} 、 $GATE$ 、 $TR0$ 、

TF0 和 T0 (P3.4) 引脚、 $\overline{\text{INT0}}$ (P3.2) 引脚, 均用于 TL0 的控制。除了仅用 8 位寄存器 TL0 外, 其功能和操作与方式 0、方式 1 完全相同, 既可定时也可计数。

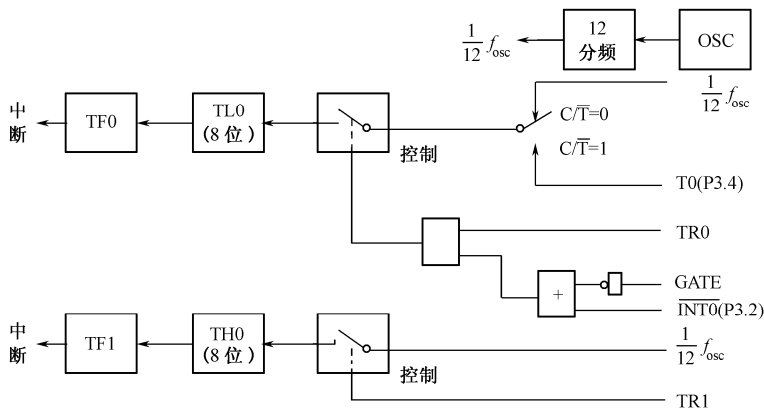


图 6-4 T0 方式 3 结构

从图 6-4 中可看出, 此时 TH0 只可用做简单的内部定时功能, 它借用原定时器 T1 的控制位 TR1 和 TF1, 同时占用 T1 的中断源, 其启动和关闭仅受 TR1 控制。当置 1 时 TH0 启动定时, 置 0 时 TH0 停止定时。方式 3 为定时器 T0 增加了一个 8 位定时器。

在定时器 T0 用做方式 3 时, T1 仍可设置为方式 0~2, 见图 6-5。由于 TR1、TF1 和 T1 中断源均被定时器 T0 占用, 此时仅有控制位 C/\overline{T} 切换其定时器或计数器工作方式, 计数溢出时, 只能将输出送入串行口。由此可见, 在这种情况下, 定时器 T1 一般用做串行口波特率发生器。当设置好工作方式时, 定时器 1 自动开始运行; 若要停止操作, 只需送入一个设置定时器 1 为方式 3 的方式控制字。通常把定时器 T1 设置为方式 2 作波特率发生器比较方便。

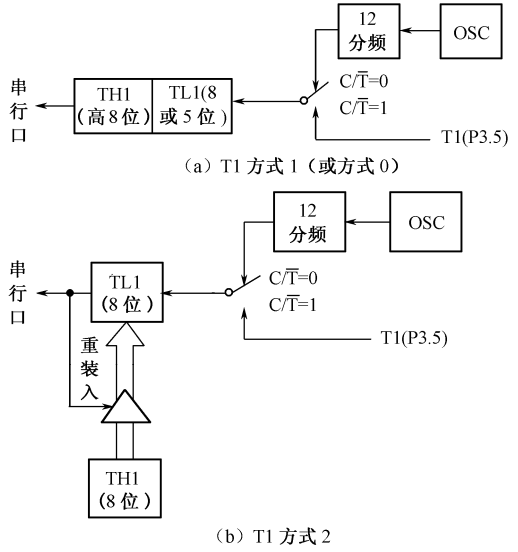


图 6-5 T0 方式 3 下的 T1 结构

6.4 定时器 T0、T1 应用举例

定时器是单片机应用系统中的重要功能部件，通过灵活应用其不同的工作方式可减轻 CPU 负担和简化外围电路。它的 4 种工作方式都可以实现定时或计数的功能，此外，它的门控位可以方便地用于测量脉冲宽度。本节将通过应用实例，介绍定时器用做定时、计数和门控位的使用方法。本节暂不使用中断方式。

6.4.1 定时应用举例

例 1 用定时器 T1 定时，完成日历时钟秒、分、时的定时。设晶振为 12MHz。

解：根据题目要求，首先要完成 1s 的定时，在这个基础上，每计满 60s，分钟加 1，而每计满 60 分钟，时钟的时加 1，计满 24 小时，时钟清零，从零时开始继续上述循环。因此要完成日历时钟的设计，首先要解决 1s 的定时。AT89S51 单片机在方式 1 下定时时间最长，最大的定时时间 T_{MAX} 为：

$$T_{\text{MAX}} = M \times 12 / f_{\text{OSC}} = 65\,536\mu\text{s} = 65.536\text{ms}$$

显然不能满足 1s 的定时时间要求，因而需要设置一个软件计数器，对分、时的计数同样通过软件计数完成。在此采用片内 50H、51H、52H、53H 单元分别进行秒、分、时以及 24 小时的计数。

可要求 T1 定时 50ms，此时 T1 的初始值 X 为：

$$(M - X) \times 1 \times 10^{-6} = 50 \times 10^{-3}$$

$$X = 65\,536 - 50\,000 = 15\,536 = 3\text{CB}0\text{H}$$

汇编语言源程序如下：

```
MOV    50H,    # 20           ; 定时 1 秒钟循环次数
MOV    51H,    # 60           ; 定时 1 分钟循环次数
MOV    52H,    # 60           ; 定时 1 小时循环次数
MOV    53H,    # 24           ; 24 小时循环次数
MOV    TMOD,   # 10H          ; 设定定时器 1 为方式 1
MOV    TH1,    # 3CH          ; 赋初值
MOV    TL1,    # 0B0H
SETB   TR1                ; 启动 T1
L2:    JBC     TF1,    L1      ; 查询计数溢出，有溢出清 TF1 为 0
      SJMP    L2
L1:    MOV     TH1,    # 3CH
      MOV     TL1,    # 0B0H
      DJNZ    50H,    L2      ; 未到 1 秒钟继续循环
      MOV     50H,    # 20
      DJNZ    51H,    L2      ; 未到 1 分钟继续循环
      MOV     51H,    # 60
      DJNZ    52H,    L2      ; 未到 1 小时继续循环
      MOV     52H,    # 60
      DJNZ    53H,    L2      ; 未到 24 小时继续循环
      MOV     53H,    # 24
      SJMP    L2              ; 反复循环
```

C51 语言程序如下:

```
#include <REG51.H>                //包含 51 单片机 SFR 库
#define Sec_data 20                //定义秒计数上限
#define Min_data 60                //定义分计数上限
#define Hou_data 60                //定义 1 小时计数上限
#define Day_data 24                //定义 1 天计数上限
unsigned char Second,Minute,Hour,Day; //定义各变量
void main (void)
{
    TMOD = 0x10;                    //设定定时器 1 为方式 1
    TH1 = 0x3c;                     //赋初值
    TL1 = 0xb0;
    TR1 = 1;                        //启动 T1
    for (Day = 0; Day < Day_data; Day++) //天计数循环
    {
        for (Hour = 0; Hour < Hou_data; Hour++) //时计数循环
        {
            for (Minute = 0; Minute < Min_data; Minute++) //分计数循环
            {
                for (Second = 0; Second < Sec_data; Second++) //秒计数循环
                {
                    while(TF1 == 0);
                    TF1 = 0;          //标志位清零
                    TH1 = 0x3c;        //重赋初值
                    TL1 = 0xb0;
                }
            }
        }
    }
}
```

在 C51 中没有对位操作判别并清零的语句, 因此只能先判别 TF1 的状态, 然后再对 TF1 进行清零, 比汇编程序复杂一点。

6.4.2 计数应用举例

例 2 设计简易频率计。频率计的闸门时间(频率计对被测信号的计数时间)为 100ms, 被测信号从 P3.5 脚输入。测量结果依次放在片内 RAM 50H、51H 单元。已知晶振频率为 6MHz。

解: 依题意, 外部信号从 T1 (P3.5) 脚输入, 因此 T1 必须工作在计数方式下。在计数方式下, 可以计数的外部信号的最高频率为晶振的 1/24, 即 250kHz。又因为闸门时间要求为 100ms, 所以定时器 T1 的最大可能计数值为 $250(\text{kHz}) \times 100(\text{ms}) = 25\,000$ 。根据上述分析, T1 应该工作在方式 1。100ms 的闸门时间可以由定时器 T0 完成, T0 工作在方式 1。

下面计算 T0 定时 100ms 的初值 X:

$$(2^{16} - X) \times 2\mu\text{s} = 100\,000\mu\text{s}$$

得出:

$$X = 65\,536 - 50\,000 = 15\,536 = 3\text{CB}0\text{H}$$

T1 用来对外部信号计数, 因此时间常数初始化为 0。

汇编语言源程序如下：

```
BEGIN:  MOV    TMOD,    # 51H          ; T0 为方式 1 定时, T1 为方式 1 外部计数
        MOV    TH0,    # 3CH          ; 定时 100ms
        MOV    TL0,    # 0B0H
        MOV    TH1,    # 0            ; 计数器初始值置零
        MOV    TL1,    # 0
        SETB   P3.5                  ; 置 T1 引脚为输入方式
        SETB   TR0                    ; 启动定时器 T0
        SETB   TR1                    ; 启动计数器 T1
DEL1:   JBC     TF0,     READ_S        ; 等待 100ms 的定时
        SJMP   DEL1
READ_S: CLR     TR1                    ; 停止 T1 对外部信号计数
        CLR     TR0                    ; 停止 T0 定时
        MOV    R0,     #50H
        MOV    @R0,    TH1
        INC    R0
        MOV    @R0,    TL1
        RET
```

C51 语言程序如下：

```
#include <REG51.H>
#include <absacc.h>
#define uchar unsigned char
unsigned int Count_at_0x50;
sbit Input = P3^5;
void main (void)
{
    TMOD = 0x51;          //T0 为方式 1 定时, T1 为方式 1 外部计数
    TH0 = 0x3C;           //定时 100ms
    TL0 = 0xB0;
    TH1 = 0x00;           //计数器初始值置零
    TL1 = 0x00;
    Input = 1;            //置 T1 引脚为输入方式
    TR0 = 1;              //启动定时器 T0
    TR1 = 1;              //启动计数器 T1
    do {} while(TF0 == 0); //等待 100ms 的定时
    TF0 = 0;              //标志位清零
    TR1 = 0;              //停止 T1 对外部信号计数
    TR0 = 0;              //停止 T0 定时
    Count = TH1*256 + TL1; //变成整数
    while(1){};
}
```

6.4.3 门控位应用举例

门控位 GATE 为 1 时，允许外部输入电平控制启动、停止定时器。利用这个特性可以测量外部输入脉冲的宽度。

例 3 利用门控位测量一个低频方波信号周期。已知低频信号频率在 100Hz 至 100kHz

之间，晶振频率为 12MHz，测量结果依次存入片内 70H、71H 单元。

解：为实现方波信号周期的测量，可以利用定时/计数器的门控位测量出方波信号的高电平时间，这个时间的两倍就是方波信号的周期。图 6-6 为门控位应用示意图。

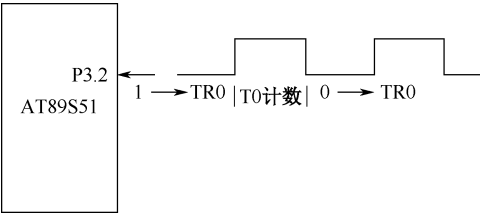


图 6-6 门控位应用示意图

被测信号从 $\overline{\text{INT0}}$ (P3.2) 管脚输入，定时器 T0 设置在定时工作方式 1 (16 位计数)，对输入的时钟信号定时计数，门控位 GATE 设为 1。测量时，应在 $\overline{\text{INT0}}$ 为低电平时，设置 TR0 为 1，这样当 $\overline{\text{INT0}}$ 变为高电平时，就自动启动定时器开始工作；当 $\overline{\text{INT0}}$ 再次变低时，定时器自动停止计数。此时读出的计数值对应被测信号高电平宽度，这个值乘以 2 就是方波信号的周期。根据信号频率范围，因为在 100Hz 时，信号的周期为 10ms，所以高电平期间定时器可能计数的最大值为 5 000。

汇编语言源程序如下：

```
MOV    TMOD,    # 09H        ; 设 T0 为方式 1, GATE=1。
MOV    TL0,     # 00H        ; 计数器清 0
MOV    TH0,     # 00H
MOV    R0,      # 71H
SETB   P3.2      ; 置 P3.2 输入方式
JB     P3.2,    $           ; 等待 P3.2 变低
SETB   TR0       ; 允许由  $\overline{\text{INT0}}$  引脚信号启、停计数器
JNB    P3.2,    $           ; 等待 P3.2 变高
JB     P3.2,    $           ; 等待 P3.2 再次变低
CLR    TR0       ; 关闭 TR0
MOV    A,       TL0
RLC    A          ; 低字节乘以 2
MOV    @R0,     A          ; 存放计数的低字节
INC    R0
MOV    A,       TH0
RLC    A          ; 高字节乘以 2
MOV    @R0,     A          ; 存放计数的高字节
RET
```

C51 语言程序如下：

```
#include <REG51.H>
sbit INPUT=P3^2;           //定义信号输入引脚
data unsigned int count_at_0x70; //定义测量结果存放位置
void main (void)
{
    TMOD = 0x09;           //设 T0 为方式 1, GATE=1
    TL0 = 0;               //计数器清 0
    TH0 = 0;
```

```

INPUT = 1;           //置 P3.2 输入方式
while(INPUT ==1);    //等待输入变低
TR0 = 1;             //允许由 INT0 引脚信号启/停计数器
while(INPUT ==0);    //等待输入变高
while(INPUT ==1);    //等待输入再次变低
TR0 = 0;             //关闭 T0
count = TL0;         //取计数值的低 8 位
count += TH0*256;    //高 8 位乘以 256 后与低 8 位相加，得到计数值
count = count * 2;   //计数值乘以 2 得到信号周期
while(1);
}

```

在这个例子中，程序定义了一个无符号的整型变量，占用两个存储字节 70H 和 71H，C51 语言中数据在内存中按高字节存放在低位地址，低字节存放在高位地址的顺序存放。因此，在 70H 单元存放的是 TH0，71H 单元存放的是 TL0。在大多数情况下不需要定位变量的绝对地址，如果程序在运行过程中与定义的变量发生地址冲突则造成整个系统崩溃。

例 3 中由于靠外部信号启动、停止计数器的工作，测量精度主要取决于时钟频率的稳定度和计数中的 ± 1 误差，因此测量精度较高，而例 2 是通过指令判断定时器 T0 溢出位的状态来停止计数器 T1 对被测信号的计数，这样会引入一定的测量误差，误差大小与相关指令的执行时间有关。

当采用软件控制计数器的启动、停止时，在某些情况下，不希望在读计数值时打断计数过程。否则，读取的计数值有可能是错的。因为不可能在同一时刻读取 THX 和 TLX 的内容。比如，先读 TL0，然后读 TH0，由于定时器在不停地运行，读 TH0 前，若恰好产生 TL0 溢出向 TH0 进位的情形，则读得的 TL0 值就完全不对了。同样，先读 TH0 再读 TL0 也可能出错（对于 T1 情况相同）。

一种解决错误读问题的方法是：先读 THX，后读 TLX，再读 THX。若两次读得的 THX 没有变化，则可确定读得的内容是正确的。若前后两次读得的 THX 有变化，则再重复上述过程，重复读得的内容就应该是正确的了。下面是按此思路编写的程序段，读得的 TH0 和 TL0 放在 R1 和 R0 内。

```

      ⋮
RP:   MOV    A,  TH0      ; 读 TH0
      MOV    R0, TL0      ; 读 TL0
      CJNE   A,  TH0, RP   ; 比较两次读得的 TH0，不等则重读
      MOV    R1, A
      ⋮

```

以上所举定时器例题均采用查询方式，使 CPU 在执行其他操作时要不断查询定时器，影响了 CPU 的工作效率，没有体现出定时器能独立运行的优越性，因而最好采用中断方式工作。在第 8 章将介绍定时器的中断工作方式。

6.5 定时/计数器 T2

在 AT89S52/C52 单片机中，增加了一个 16 位定时/计数器 T2。T2 与 T0 和 T1 有类似的功能，既可以作定时器或计数器使用，同时还增加了捕捉等新的功能。它的功能比其他 2 个定时器更强，使用也较复杂。在特殊功能寄存器组中有 6 个与 T2 有关的寄存器，它们分别是：

控制寄存器 T2CON、方式控制寄存器 T2MOD、捕捉寄存器 RCAP2L 和 RCAP2H、定时/计数器 TL2、TH2。它们在片内存储器中的地址依次从 C8H 至 CDH。

6.5.1 T2 的寄存器

在 T2 的 6 个寄存器中，TL2、TH2 与 TL0、TH0 相同，是用于存放计数值的。捕捉寄存器 RCAP2L、RCAP2H 是在捕捉工作方式下用于存放所捕获的 TL2、TH2 瞬时值的。控制寄存器 T2CON 和方式控制寄存器 T2MOD 是用于控制和管理 T2 工作的，下面予以介绍。

1. T2CON 控制寄存器

T2 是靠软件对 T2CON 寄存器进行设置而启动和运行的。T2 控制寄存器 T2CON 的格式如下。

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/ $\overline{T2}$	CP/ $\overline{RL2}$

各位定义如下：

1) TF2——T2 的溢出中断标志

T2 溢出时由硬件置为 1，须由用户用软件清 0。当 RCLK=1 或 TCLK=1 时，即使溢出也不会将 TF2 置位。

2) EXF2——T2 外部中断标志

T2 在捕捉方式和常数自动重装入方式下，如果 EXEN2=1，在 T2EX 端 (P1.1) 发生的负跳变使 EXF2 置位。如此时允许 T2 中断，则 EXF2=1 会使 CPU 响应中断。同样需要由软件清 0。T2 工作在加 1/减 1 计数方式 (DCEN=1) 时，EXF2 不会置位。

3) RCLK——串行口接收时钟选择位

RCLK=1 时，T2 工作于波特率发生器方式。此时，T2 的溢出脉冲作为串行口方式 1 和方式 3 的接收时钟。

RCLK = 0 时，T1 的溢出脉冲做接收时钟。

4) TCLK——串行口发送时钟选择位

TCLK = 1 时，T2 工作于波特率发生器方式，T2 的溢出脉冲作为串行口方式 1 和方式 3 的发送时钟。

TCLK = 0 时，T1 的溢出脉冲做发送时钟。

5) EXEN2——T2 的外部允许控制位

T2 工作于捕捉方式及常数自动重装入方式，且 EXEN2=1 时，如 T2EX 输入端上有一个负跳变信号，则会引发捕捉或常数重装入动作；而 EXEN2=0 时，T2EX 端的电平变化对 T2 没有影响。

6) TR2——T2 的计数控制位

TR2 = 1，开始计数；

TR2 = 0，禁止计数。

7) C/ $\overline{T2}$ ——定时器或计数器功能选择位

C/ $\overline{T2}$ =0，T2 为内部定时器（对振荡脉冲的 12 分频信号进行计数）；

C/ $\overline{T2}$ =1，T2 为外部事件计数器（下降沿触发）。

8) $\overline{\text{CP/RL2}}$ ——捕捉或常数重装入方式选择位

$\overline{\text{CP/RL2}}=1$ 时, T2 工作于捕捉方式, 即当 $\text{EXEN2}=1$ 时, T2EX 端的负跳变引发捕捉动作; $\overline{\text{CP/RL2}}=0$ 时, 为常数自动重装入方式, 当 $\text{EXEN2}=1$ 时, T2EX 端的负跳变引发常数重装入动作。

当 $\text{TCLK}=1$ 或 $\text{RCLK}=1$ 时, 该位无效, 定时器 2 工作于常数自动重装入方式。

2. 方式控制寄存器 T2MOD

T2 的 T2MOD 中只有 D_0 和 D_1 两位对 T2 的工作有影响。

T2MOD 的格式如下:

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	X	T2OE	DCEN

1) D_1 位 T2OE——T2 的输出允许位

T2OE=1, 允许 T2 输出。

T2OE=0, 禁止 T2 输出。

2) D_0 位 DCEN——T2 加 1/减 1 计数允许位

T2 工作在自动重装入方式时, $\text{DCEN}=1$, 允许 T2 加 1/减 1 计数。具体加 1 还是减 1 又与 T2EX(P1.1)引脚的电平有关。当 $\text{T2EX}(\text{P1.1})=1$ 时, T2 加 1 计数; 反之, 为减 1 计数。复位时, $\text{DCEN}=0$, T2 为加 1 计数。

6.5.2 定时器 T2 的工作方式

T2 共有 4 种工作方式, 分别为: 16 位捕捉方式、16 位常数自动重装入方式、波特率发生器方式和时钟输出方式。由 T2CON 寄存器中的 D_0 、 D_2 、 D_4 、 D_5 位及 T2MOD 的 D_1 位的组合可选择这几种方式。其组合的对应关系如表 6-2 所示。其中, 16 位捕捉方式和 16 位常数自动重装入方式中, 通过设置均可选择定时/计数功能。

表 6-2 定时器 T2 的工作方式

RCLK+TCLK	$\overline{\text{CP/RL2}}$	$\text{C}/\overline{\text{T2}}$	T2OE	TR2	工 作 方 式
0	0	1/0	0	1	16 位常数自动重装入
0	1	1/0	0	1	16 位自动捕捉方式
1	×	×	×	1	波特率发生器方式
×	×	0	1	1	时钟输出方式
×	×	×	×	0	停止计数

表 6-2 中 × 表示此值无影响, 1/0 表示该位按照功能要求选择为 0 或 1。

下面对 T2 增加的 4 种方式予以介绍。

1. 16 位常数自动重装入方式

16 位常数自动重装入是指在满足某规定条件时, RCAP2L 和 TCAP2H 中存放的计数初值可自动重新装入 TL2 和 TH2 中。当 $\overline{\text{CP/RL2}}=0$ 时, 选择自动重装入方式。根据 DCEN 位

所选择的计数状态，自动重装入操作又可分为以下 2 种情况。

(1) 加 1 计数方式

当 DCEN=0 时，T2 为自动加 1 计数方式，它的结构原理如图 6-7 所示。由图可见，由 T2CON 寄存器的 EXEN2 位控制选择 T2 的 2 种重装入方式。

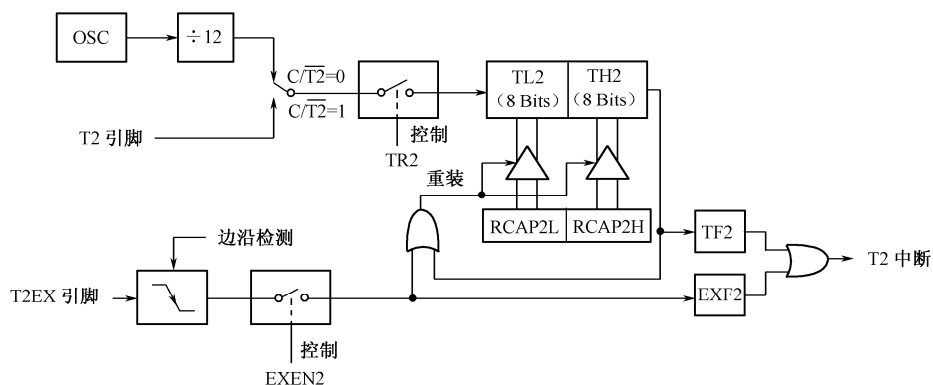


图 6-7 定时器 T2 的常数自动重装入方式结构图 (TCEN=0)

- 当 EXEN2=0 时，T2 做定时器/计数器用。当 $C/\overline{T2}=0$ 时，做定时器用，以振荡频率的 12 分频计数；当 $C/\overline{T2}=1$ 时，做计数器用，以 T2 外部输入引脚 (P1.0) 的输入脉冲做计数脉冲 (下降沿触发)。当 TR2=1 时，从初值开始增 1 计数，计数至溢出时，溢出信号控制打开三态门将 RCAP2L 和 TCAP2H 寄存器中存放的计数初值重新装入 TL2 和 TH2 中，使 T2 从该值开始重新计数，同时将溢出标志 TF2 置 1。计数器的初值在初始化时由软件编程设置。
- 当 EXEN2=1 时，T2 除可完成上述功能外，还可实现以下功能。当外部输入引脚 T2EX (P1.1) 的输入电平发生负跳变时，可以控制将捕捉寄存器 RCAP2L 和 RCAP2H 的内容重新装入 TH2 和 TL2 中，使 T2 重新以新值开始计数，同时把中断标志 EXF2 置 1，向 CPU 发出中断请求。

(2) 加 1/减 1 计数方式

当 DCEN=1 时，T2 为加 1/减 1 计数方式，它的结构原理如图 6-8 所示。由 T2EX 引脚的电平状态决定 T2 是做加计数还是做减计数，即由 T2EX 引脚状态控制选择 T2 的两种重装入方式。

- 当 T2EX 为高电平时，T2 做加 1 计数，计数至 0FFFFH 时产生溢出，将重装/捕捉寄存器 RCAP2L 和 RCAP2H 的内容重新装入 TH2 和 TL2 中，使 T2 重新以新值开始计数，同时把标志 TF2 置 1。
- 当 T2EX 为低电平时，T2 做减 1 计数，当 TH2 和 TL2 中的值减至与 RCAP2L 和 RCAP2H 的值相同时，CPU 将自动把 FFFFH 装入 TH2 和 TL2 中，同时使 TF2 标志置 1。

在这种方式下，加 1 或减 1 溢出使 EXF2 状态切换，但无论是加 1 还是减 1 计数，EXF2 标志的变化都不会引起中断，此时，可以把 EXF2 标志看做结果的第 17 位。

2. 自动捕捉方式

当 $CP/\overline{RL2}=1$ 时，T2 除可用于定时计数外，还可工作于捕捉方式。它的 16 位捕捉方式的结构原理如图 6-9 所示。T2CON 中的 EXEN2 位可控制 T2 以 2 种工作方式工作。

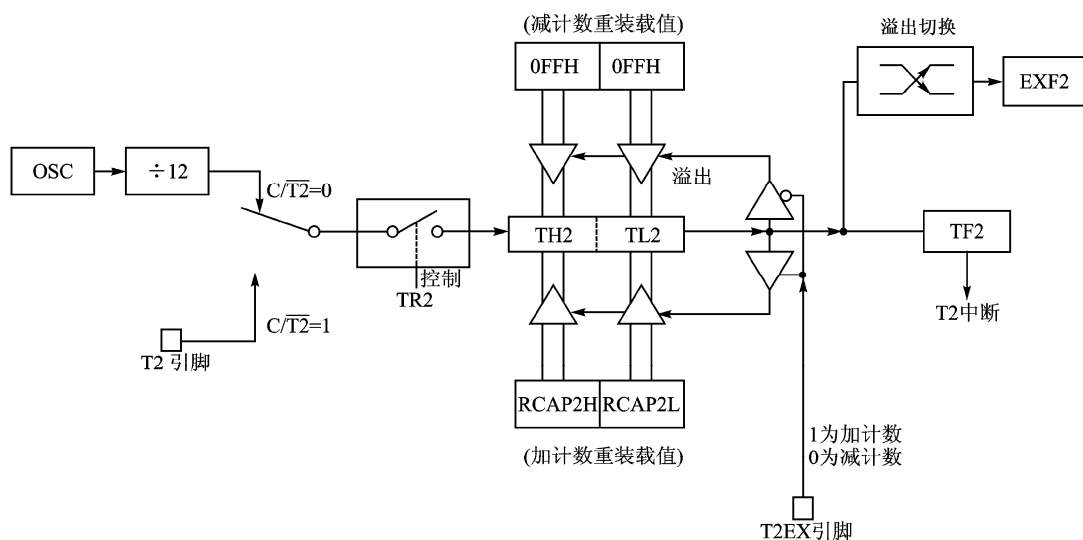


图 6-8 定时器 T2 的常数自动重装入方式 (DCEN=1)

- 如果 EXEN2=0, 则 T2 作为定时器/计数器使用, 并由 $C/\overline{T2}$ 位决定它是做定时器还是计数器。如作为定时器使用, 则其计数输入为内部振荡脉冲的 12 分频信号; 做计数器时, 是以 T2 的外部输入引脚 (P1.0) 上的输入脉冲做计数脉冲。当定时器/计数器 T2 增 1 计数至溢出后, 将 TF2 标志置 1, 并发出中断请求信号。在这种方式下, TL2 和 TH2 的内容不会送入捕捉寄存器中。
- 如果 EXEN2=1, T2 除实现上述定时器/计数器功能外, 还可实现捕捉功能, 即当外部输入端 T2EX (P1.1) 的输入电平发生负跳变时, 就会把 TH2 和 TL2 的内容锁入捕捉寄存器 RCAP2L 和 RCAP2H 中, 并将 T2CON 中的中断标志位 EXF2 置 1, 向 CPU 发出中断请求信号。

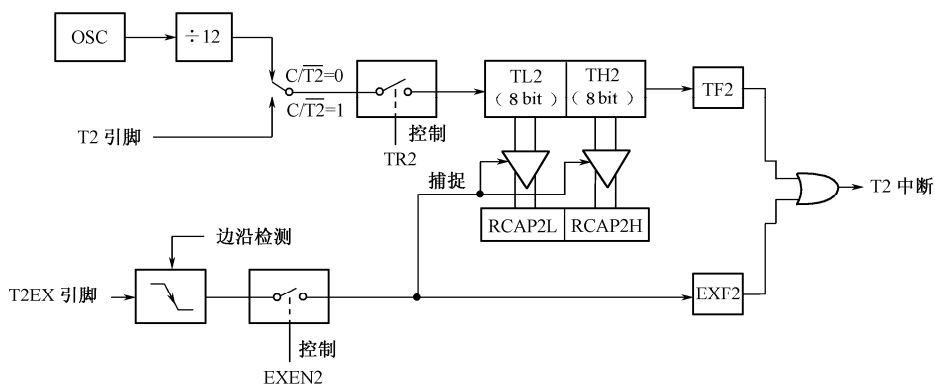


图 6-9 定时器 T2 的捕捉方式结构图

3. 波特率发生器方式

T2 还可做串行口的波特率发生器。由控制寄存器 T2CON 中的控制位 RCLK=1 或 TCLK=1 确定它的串行口波特率发生工作方式。如果 RCLK=1 或 TCLK=1, 则定时器 2 以波特率发生器方式工作。波特率发生器的结构原理图如图 6-10 所示。

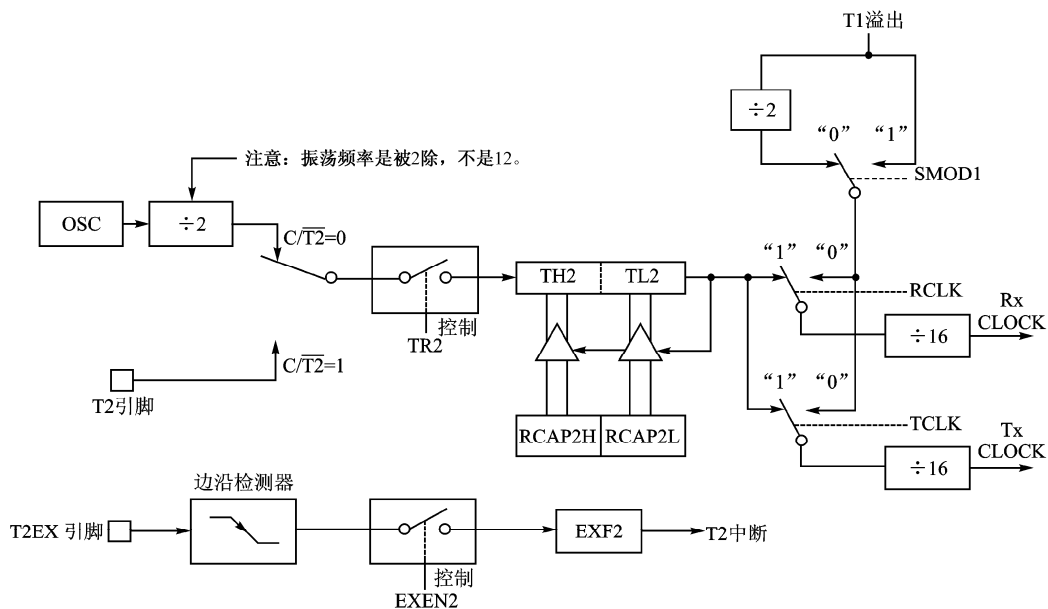


图 6-10 定时器 T2 的波特率发生器方式结构图

波特率发生器方式类似于常数自动重装入方式。在此方式下，TH2 溢出，使定时器 2 的 RCAP2L 和 TCAP2H 寄存器中存放的计数初值重新装入 TL2 和 TH2 中，使 T2 从该值开始重新计数。

在方式 1 和方式 3 时，波特率由定时器 2 的溢出速率确定，由式（6-1）计算：

$$\text{方式 1 和方式 3 的波特率} = \frac{\text{定时器 2 的溢出率}}{16} \quad (6-1)$$

定时器 2 做波特率发生器与做定时器时的操作有区别。做定时器时，是每个机器周期（1/12 振荡频率）定时器的计数值加 1。做波特率发生器时，是每个状态周期（1/2 振荡频率）定时器的计数值加 1，因而波特率的具体计算公式如下：

$$\text{方式 1 和方式 3 的波特率} = \frac{\text{振荡器频率}}{32 \times (65536 - X)} \quad (6-2)$$

式中， X 是一个 16 位的无符号整数，其高、低 8 位分别为 RCAP2H 和 RCAP2L 中的值。

定时器 2 工作在波特率发生器方式时，要特别注意以下两点。

- TH2 溢出时，不产生中断，但 T2EX 引脚可做附加的外部中断输入用，中断时不会引起自动重装。
- 当定时器 2 作为波特率发生器运行期间，不应对其 TH2、TL2 进行读写操作，如果需要访问它，则应先将 TR2 置 0，停止其运行。

4. 可编程时钟输出方式

定时器 2 通过编程可以从 P1.0 引脚输出占空比为 1:1 的时钟信号。可编程时钟输出方式的原理如图 6-11 所示。

当 $C/T2$ 置为 0，且 T2OE 位置 1 时，定时/计数器 2 工作于时钟发生器方式，TR2 位控制定时器 T2 的启动与停止。输出的时钟频率取决于振荡器频率和定时器 2 捕获寄存器

(RCAP2H、RCAP2L) 的重装载数值，计算公式如下：

$$\text{时钟输出频率} = \frac{\text{振荡器频率}}{4 \times (65536 - X)} \quad (6-3)$$

式中， X 同 (6-2) 式。当 CPU 工作频率为 16MHz 时，通过 P1.0 脚输出时钟，输出时钟频率从 61Hz 到 4MHz。

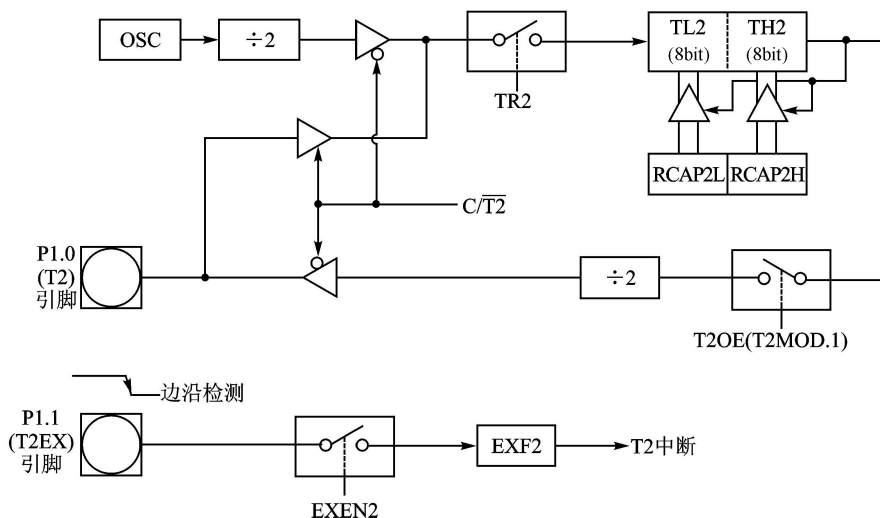


图 6-11 定时器 T2 的可编程时钟输出方式原理

在时钟输出方式，定时器 2 溢出也不会产生中断。因此，定时器 2 在用于波特率发生器的同时，还可以通过 P1.0 脚输出时钟。由于波特率发生器和时钟输出时都使用重载寄存器 RCAP2H 和 RCAP2L，因此，当定时器 2 既用作波特率发生器，又要从 P1.0 脚输出时钟时，输出的时钟的频率就不能随意编程设置，其频率值与波特率的设置有关，在使用时要注意这一点。

综上所述，定时器 T2 可以用做定时器，也可以用做计数器，而且，无论做定时器还是计数器都有捕捉方式和自动重装入方式。在 T2 作定时器时，还有波特率发生器方式。

在 AT89C52/AT89S52 中，还增加了定时器 T2 的中断请求源，入口地址是 2BH。T2 中断级别最低，其中断请求是由标志 TF2 (T2CON.7) 和 EXF2 (T2CON.6) 单独或经逻辑或以后产生的。当 CPU 响应该中断请求后，必须由软件来判别是 TF2 还是 EXF2 产生的中断，也必须由软件将该标志清 0。

6.5.3 应用例题

利用 T2 的捕捉功能测试外部输入引脚 T2EX (P1.1) 上的正脉冲宽度，捕捉到后，用 P1.7 脚报警提示，使与 P1.7 脚相接的二极管灯亮。脉冲宽度值分别存放在内存单元 60H、61H 中。

本例题的硬件连接电路见图 6-12。由于在 80C51 系列的汇编指令中没有 T2 寄存器的符号，所以有关 T2 的寄存器都要在伪指令中专门定义，否则在程序中就只能用直接地址。

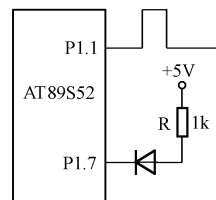


图 6-12 硬件连接示意图

程序清单如下：

```

T2CON EQU 0C8H
RCAP2L EQU 0CAH
RCAP2H EQU 0CBH
TL2 EQU 0CCH
TH2 EQU 0CDH
TR2 BIT 0CAH
EXF2 BIT 0CEH
TF2 BIT 0CFH

ORG 0000H
LJMP MAIN
ORG 002BH ; 定时器 2 中断入口地址
LJMP T2T ; 转定时器 2 中断服务程序
ORG 0030H
MAIN: MOV IE, #0A0H ; CPU 和 T2 开中断
JB P1.1, $ ; 等待信号变低
MOV T2CON, #00001001B ; 设 T2 为捕捉工作方式
JNB P1.1, $ ; 等待信号变高
MOV TH2, #0 ; 清 T2 定时器
MOV TL2, #0
SETB TR2 ; 启动 T2 开始工作
SJMP $ ; 等待
ORG 100H
T2T: CLR TR2 ; 停止 T2 工作
MOV 61H, RCAP2L ; 取捕捉寄存器的低 8 位
MOV 60H, RCAP2H ; 取捕捉寄存器的高 8 位
CLR P1.7 ; 报警灯亮
CLR EXF2 ; 清中断标志位
CLR EA
RETI
END

```

C51 语言程序如下：

```

#include <REG52.H> //包含 52 单片机 SFR 库
sbit INPUT=P1^1 ; //定义输入位
sbit ALARM=P1^7 ; //定义输出位
sfr16 RCAP2=0xca; //给定时器 2 的捕捉寄存器赋 16 位数初值
sfr16 TMR2=0xcc; //给定时器 2 赋 16 位数初值
data unsigned int count_at_0x60; //定义计数值存放位置
void main (void)
{
    IE = 0xa0; // CPU 和 T2 开中断
    ALARM = 1; //输出为 1，灭灯
    INPUT = 1; //设定为输入方式
    while(INPUT ==1); //等待信号变低
    T2CON = 0x09; //设 T2 为捕捉工作方式
    while(INPUT ==0); //等待信号变高
    TMR2 =0;
    TR2 = 1; //启动 T2 开始工作
    while(1);
}

```

```

}
void Timer2_Pro(void) interrupt 5
{
    count = RCAP2;           //取捕捉器中的值
    ALARM = 0;               //报警灯亮
    EXF2 = 0;               //清中断标志位
    EA = 0;
}

```

在这个例子中，由于采用了定时器 2 的捕捉功能，即使在读数时不停止计数过程，也不必像 T0、T1 那样采用 2 次读取高字节的方法，从而简化了读数方法，提高了测量外部脉冲高电平的精度，这时的测量误差主要是由软件启动定时器产生的。

6.6 定时器 T3——WDT 监视定时器

在 AT89S51/S52 中还增加了一个定时器 T3，称为 WDT 看门狗监视定时器。监视定时器英文全称为 Watchdog Timer，缩写为 WDT，简称为看门狗。这是一个通过软、硬件相结合的，重要常用抗干扰技术。

6.6.1 WDT 的功能及应用特点

WDT 通常是一个独立的定时器，它的主要用途是当程序运行出现“死机”即“死循环”时，能通过复位的方法使 CPU 退出“死循环”。

在 AT89S51 中 WDT 是由一个 14 位的计数器和一个看门狗复位寄存器组成的。在 AT89S52 中 WDT 是 13 位的计数器，看门狗复位寄存器是特殊功能寄存器，符号为 WDTRST，地址为 A6H，这是一个只写寄存器。

当单片机复位时，WDT 是不工作的，启动 WDT 开始工作的方法是顺序向 WDTRST 中写 1EH 和 0E1H。写完后计数器从 0 开始计数，WDT 开始工作后每个机器周期计数增 1。AT89S51 计数到 16383 (3FFFH) 时计数器溢出，AT89S52 计数到 8191 (1FFFH) 时计数器溢出，溢出后单片机复位。

通过软件编程使单片机在正常运行时不断给它发“清 0”信号，使 WDT 不会产生溢出。如果单片机出现“死机”即“死循环”，则 WDT 不能按时收到“清 0”信号；当 WDT 计时到设定时间就会产生溢出信号，使 RST 引脚出现正脉冲，单片机复位，恢复程序的正常运行。

由于在电源关断 (Power-down) 方式下，振荡器停止工作，这意味着 WDT 也停止计数，因此在进入电源关断方式之前和执行退出电源关断方式的中断服务程序期间，建议对 WDT 复位。

AT89S51/S52 单片机 WDT 的设定时间与所采用的晶振有关，一旦晶振确定，则 AT89S51/S52 WDT 的溢出周期是一个确定的值。

6.6.2 辅助寄存器 AUXR

辅助寄存器 AUXR (Auxiliary Register) 的 WDIDLE 位用来确定在进入空闲方式后 WDT

是否继续工作。辅助寄存器 AUXR 的各位格式如下。

AUXR	D7	D6	D5	D4	D3	D2	D1	D0
(8EH)	—	—	—	WDIDLE	DISETO	—	—	DISALE

各位功能如下：

(1) WDIDLE 位——WDT 方式选择位

当 WDIDLE = 0，在空闲方式期间 WDT 继续计数；

当 WDIDLE = 1，WDT 停止计数，在设置空闲方式后才恢复工作。

为了避免在空闲方式由于 WDT 的溢出使 AT89S51/S52 复位，用户应该周期性地退出空闲方式，顺序向 WDTRST 中写入 01EH 和 0E1H，重新进入空闲方式。

(2) DISETO 位——复位输出控制位

当 DISETO = 0，在 WDT 时间到后，复位引脚为高电平输出；

当 DISETO = 1，复位引脚始终为输入状态。

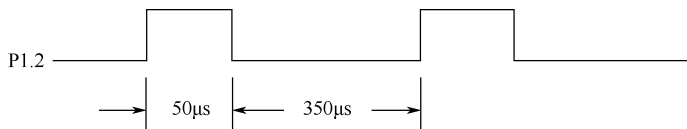
(3) DISALE 位——ALE 引脚控制位

当 DISALE = 0，ALE 引脚始终输出一个不变的六分之一的时钟振荡频率；

当 DISALE = 1，ALE 引脚仅在执行 MOVX 或 MOVC 指令时，才输出时钟振荡频率。

思考与练习

1. AT89S51/C51 单片机内部有几个定时/计数器？它们由哪些专用寄存器组成？
2. AT89S51/C51 单片机的定时/计数器有哪几种工作方式？各有什么特点？
3. 定时/计数器用做定时方式时，其定时时间与哪些因素有关？做计数时，对外界计数频率有何限制？
4. 当定时器 T0 用做方式 3 时，由于 TR1 位已被 T0 占用，如何控制定时器 T1 的开启和关闭？
5. 已知 AT89S51 单片机系统时钟频率为 24MHz，请利用定时器 T0 和 P1.2 输出矩形脉冲，其波形如下。



6. 在 AT89S51 单片机中，已知时钟频率为 12MHz，请编程使 P1.0 和 P1.1 分别输出周期为 2ms 和 500μs 的方波。
7. 设系统时钟频率为 24MHz，试用定时器 T0 做外部计数器，编程实现每计到 1000 个脉冲，使 T1 开始 2ms 定时，定时时间到后，T0 又开始计数，这样反复循环不止。
8. 利用 AT89S51 单片机定时/计数器测量某正脉冲宽度，已知此脉冲宽度小于 10ms，主机频率为 24MHz。编程测量脉冲宽度，并把结果转换为 BCD 码顺序存放在以片内 50H 单元为首地址的内存单元中（50H 单元存个位）。
9. 在 AT89S52 单片机中的定时器 2 有几种工作方式？各有什么特点？

第 7 章 80C51 的串行接口

串行通信是单片机与外界交换信息的一种基本通信方式。为了实现串行通信，绝大多数单片机都配置了 UART 串行接口，此接口也可用做同步移位寄存器方式下的串行扩展接口。本章将介绍串行通信的概念、原理及 80C51 单片机的 UART 串行接口的结构、原理及应用。

7.1 串行通信概述

计算机与外界的信息交换称为通信。基本的通信方式有两种。

并行通信：所传送数据的各位同时发送或接收；

串行通信：所传送数据的各位按顺序一位一位地发送或接收。

在并行通信中，一个并行数据占多少位二进制数，就要多少根传输线。这种方式的特点是通信速度快，但传输线多，价格较贵，适合近距离传输；而串行通信仅需一到两根传输线即可，故在长距离传输数据时，比较经济。但由于它每次只能传送一位，所以传送速度较慢。图 7-1 (a) 和 (b) 分别为计算机与外设或计算机之间的并行通信及串行通信的连接方法。

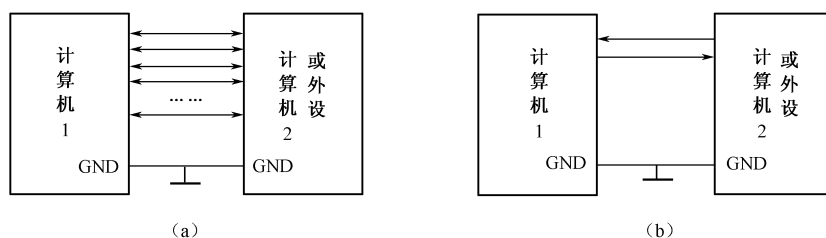


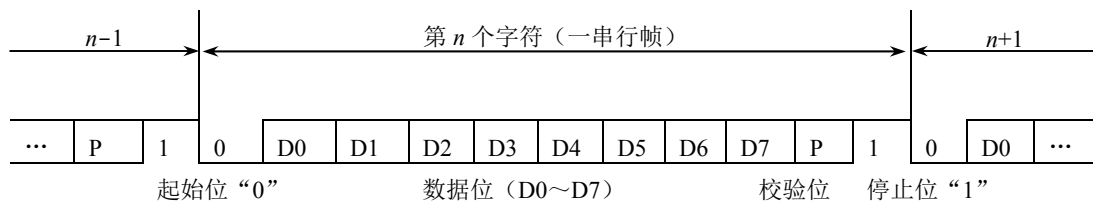
图 7-1 基本通信方式

7.1.1 同步通信和异步通信方式

串行通信分同步和异步两种方式。

1. 异步通信 ASYNC (Asynchronous Data Communication)

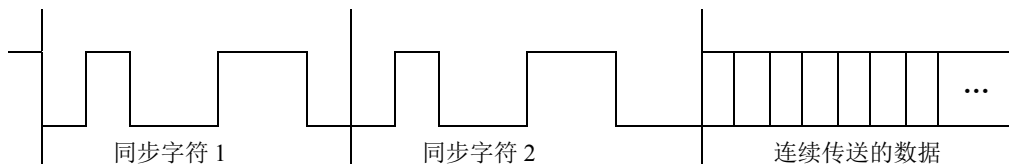
在异步通信中数据或字符是一帧 (Frame) 一帧传送的。帧定义为一个字符的完整的通信格式，通常也称为帧格式。最常见的帧格式一般是先用一个起始位“0”表示字符的开始，然后是 5~8 位数据，规定低位在前，高位在后。其后是奇偶校验位，此位通过对数据奇偶性的检查，可用于判别字符传送的正确性。其有 3 种可能的选择，即奇、偶、无校验，用户可根据需要选择（在有的格式中这位可省略）。最后是停止位，用以表示字符的结束，停止位可以是 1、1.5、2 位，不同的计算机规定有所不同。从起始位开始到停止位结束就构成完整的一帧，下面是一种 11 位的帧格式。



由于异步通信每传送一帧有固定格式，通信双方只需按约定的帧格式来发送和接收数据，所以硬件结构比同步通信方式简单；此外它还能利用校验位检测错误，所以这种通信方式应用较广泛。在早期的单片机通信中主要采用异步通信方式，现在这种方式仍然被普遍使用。

2. 同步通信 SYNC (Synchronous Data Communication)

在同步通信中，数据或字符开始处是用一同步字符来指示（常约定 1~2 个）的，以实现发送端和接收端同步，一旦检测到约定同步字符，下面就连续按顺序发送和接收数据。同步通信中，在发送数据的同时，还要发送同步脉冲（也称为同步时钟），同步脉冲与数据信号的时间间隔相同。同步数据传送格式如下。



因为同步通信数据块传送时去掉了字符开始和结束的标志，所以其速度高于异步通信，但这种方式对硬件结构要求较高。由于这种方式较易于进行串行外围扩展，所以，目前很多型号的单片机都增加了串行同步通信接口，对于不具有串行同步通信接口的单片机软件编写较复杂，详见第 9 章。

7.1.2 串行通信的数据传送速率

传送速率是指数据传送速度的指标。在串行通信中数据传送速率用波特率（Baud rate）表示，其意义是每秒钟传送多少位二进制数。假如数据传送的速率每秒为 960 个字符，每个字符由 1 个起始位、8 个数据位和 1 个停止位组成，则其传送波特率为：

$$10 \times 960 = 9600 \text{ b/s} = 9600 \text{ 波特}$$

每一位的传送时间即为波特率的倒数：

$$T_d = 1/9600 = 0.104\text{ms}$$

异步通信的传送速度一般在 50 到 100k 波特之间，常用于单片机到 CRT 终端，以及双机或多机之间的通信等。

7.1.3 串行通信的方式

在串行通信中，数据是在两机之间传送的。按照数据传送方向，串行通信可分为单工（Simplex）制式、半双工（Half Duplex）制式和全双工（Full Duplex）制式。3 种方式如图 7-2 所示。

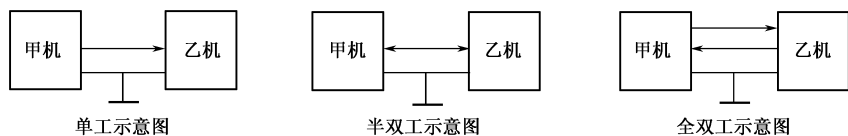


图 7-2 通信方式示意图

1. 单工制式

在单工制式下，甲机和乙机之间只允许单方向传送，例如，只能甲机发送、乙机接收，因而两机之间只需一条数据线。

2. 半双工制式

在半双工制式下，甲机和乙机之间允许双方向传送，但它们之间只有一个通信回路，接收和发送不能同时进行，只能分时发送和接收，即甲机发送、乙机接收，或者乙机发送、甲机接收，因而两机之间只需一条数据线。

3. 全双工制式

在全双工制式下，甲、乙两机之间数据的发送和接收可以同时进行，称为全双工传送。全双工形式的串行通信必须使用二根数据线。

不管哪种形式的串行通信，在两机之间均应有共地线。

7.1.4 通信协议

通信协议是指在单片机之间进行数据传输时的一些约定，包括通信方式、物理电平、波特率、双机之间握手信号的约定等。为保证单片机之间能准确、可靠地通信，相互之间必须遵循统一的通信协议，在通信之前一定要先设置好。

7.2 80C51 串行口简介

为了使单片机能实现串行通信，在 80C51 系列单片机及其他很多型号单片机芯片内部都设计了 UART (Universal Asynchronous Receiver/Transmitter) 串行接口。它是一个可编程的全双工异步串行通信接口，通过软件编程可以作为通用异步接收和发送器用，也可作为同步移位寄存器用，还能实现多机通信。其帧格式有 8 位、10 位和 11 位，并能设置各种波特率，使用灵活方便。

7.2.1 串行口结构与工作原理

80C51 串行口结构框图如图 7-3 所示。由图可见它主要由两个数据缓冲寄存器 SBUF 和一个输入移位寄存器，以及一个串行控制寄存器 SCON 等组成。波特率发生器可以采用定时器 T1 或 T2 控制发送和接收的速率。接收与发送缓冲寄存器 SBUF 采用同一个地址代码 99H，其寄存器名亦同样为 SBUF。CPU 通过不同的操作命令，区别这两个寄存器，所

以不会因为地址代码相同而产生错误。当 CPU 发写 SBUF 命令时，即向发送缓冲寄存器中装载新的信息，同时启动数据串行发送；当 CPU 发读 SBUF 命令时，读接收缓冲寄存器的内容。特殊功能寄存器 SCON 用以存放串行口的控制和状态信息。80C51 串行口正是通过对上述专用寄存器的设置、检测与读取来管理串行通信的。

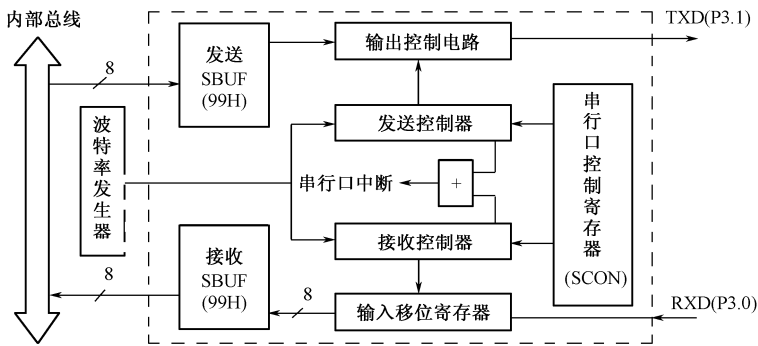


图 7-3 串行口结构框图

在进行串行通信时，外界数据是通过引脚 RXD（P3.0，串行数据接收端）输入的。当设置了允许接收命令之后，一旦发现 RXD 引脚由高变低，则认为是数据接收的起始位，单片机就开始接收数据。输入数据先逐位进入输入移位寄存器，再送入接收寄存器。在接收器中采用了双缓冲结构，以避免在接收到第二帧数据之前，CPU 未及时响应接收器的前一帧中断请求，没把前一帧数据读走，而造成两帧数据重叠的错误。在发送时，因为 CPU 是主动的，不会产生写重叠问题。一般不需要双缓冲器结构，以保持最大传送速率。在输出控制电路框中包括发送移位寄存器、或门、与非门等逻辑电路，当发送缓冲寄存器 SBUF 的内容送入发送移位寄存器后，发送控制器发出移位脉冲，使移位寄存器中的数据按低位在前的顺序逐位右移输出，移位速度由波特率发生器控制。发送完毕置 TI 位为 1。

7.2.2 串行口控制寄存器 SCON

80C51 串行通信的方式选择、接收和发送控制以及串行口的状态标志均由专用寄存器 SCON 控制和指示，其格式如下。

SCON	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
(98H)	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

各位作用如下：

(1) SCON.7 和 SCON.6 SM0 和 SM1——串行方式选择位

这 2 位用于选择串行口的 4 种工作方式，见表 7-1。由表中的功能项可以看出这几种方式的帧格式不完全相同（帧格式的详细说明见 7.2.3 节），各串行通信方式的说明见 7.3 节。

表 7-1 串行口工作方式选择

SM0	SM1	工作方式	功 能	波 特 率
0	0	方式 0	8 位同步移位寄存器	$f_{osc}/12$
0	1	方式 1	10 位 UART	可变

续表

SM0	SM1	工 作 方 式	功 能	波 特 率
1	0	方式 2	11 位 UART	$f_{osc}/64$ 和 $f_{osc}/32$
1	1	方式 3	11 位 UART	可变

(2) SCON.5 SM2——多机通信控制位

- 在方式 2 和方式 3 中，SM2 主要用于多机通信控制。当串行口以方式 2 或方式 3 接收时，如果 SM2=1，允许多机通信，且接收到第 9 位 RB8 为 0 时，则 RI 不置“1”，不接收主机发来的数据；如果 SM2=1，且 RB8 为 1 时，RI 置“1”，产生中断请求，将接收到的 8 位数据送 SBUF。当 SM2=0 时，则不论 RB8 为 0 还是为 1，都将收到的 8 位数据送入 SBUF 中，并产生中断。
- 在方式 1 中，当处于接收时，若 SM2=1，则只有收到有效的停止位时，RI 才置“1”。
- 在方式 0 中，SM2 置置“0”。

(3) SCON.4 REN——允许串行接收位

由软件置位或清除。REN=1 时，允许接收；REN=0 时，禁止接收。

(4) SCON.3 TB8——发送数据的第 9 位 (D8)

在方式 2 或方式 3 中，根据需要由软件置位或复位也称为清 0。双机通信时它可做奇偶校验位；在多机通信中可作为区别地址帧或数据帧的标志位，一般由指令设定地址帧时 TB8 为 1，数据帧时 TB8 为 0。

(5) SCON.2 RB8——接收数据的第 9 位 (D8)

在方式 2 或方式 3 中，定义与 TB8 一致（例如，可能是奇偶位，或是地址/数据标志位）。

(6) SCON.1 TI——发送中断标志位

在方式 0 中，发送完 8 位数据后，由硬件置位；在其他方式中，在发送停止位之初由硬件置位。TI=1 时，可申请中断，也可供软件查询用。在任何方式中都必须由软件来清除 TI。

(7) SCON.0 RI——接收中断标志位

在方式 0 中，接收完 8 位数据后，由硬件置位；在其他方式中，在接收停止位的中间，由硬件置位。RI=1 时，既可申请中断，也可供软件查询用。在任何方式中都必须由软件清除 RI。

SCON 中低 2 位因与中断有关，将在中断章节详细介绍。

SCON 的地址为 98H，可以位寻址。复位时，SCON 所有位均清 0。

7.2.3 80C51 的帧格式

80C51 串行口通过编程可设置 4 种工作方式，3 种帧格式。

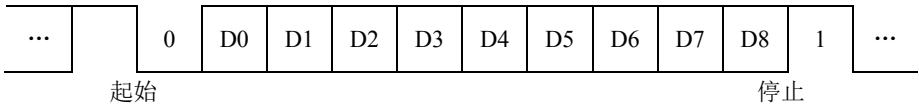
方式 0 以 8 位数据为一帧，不设起始位和停止位，先发送或接收最低位。其帧格式如下。

...	D0	D1	D2	D3	D4	D5	D6	D7	...
-----	----	----	----	----	----	----	----	----	-----

方式 1 以 10 位为一帧传输，设有一个起始位“0”，8 个数据位和一个停止位“1”。其帧格式如下。

...	0	D0	D1	D2	D3	D4	D5	D6	D7	1	...
	起始									停止	

方式 2 和 3 以 11 位为一帧传输，设有 1 个起始位 “0”，8 个数据位，1 个可编程位（第 9 数据位）D8 和 1 个停止位 “1”。其帧格式如下。



可编程位 D8 由软件置 1 或清 0。该位既可作为校验位，也可作为它用。以上 4 种工作方式的详细介绍见 7.3 节。

7.2.4 波特率的设置

在串行通信前，首先要设置收发双方对发送或接收的数据速率（即波特率）。通过软件对 80C51 串行口编程可约定 4 种工作方式。这 4 种方式波特率的计算方法不同，其中方式 0 和方式 2 的波特率是固定的，而方式 1 和方式 3 的波特率是可变的，均由定时器 T1 的溢出率控制。下面分别加以介绍。

1. 方式 0 和方式 2 的波特率

在方式 0 时，每个机器周期发送或接收一位数据，因此波特率固定为时钟频率的 1/12，且不受 SMOD 的影响。

方式 2 的波特率取决于 PCON 中 SMOD 的值，PCON 主要用于电源控制，但它的最高位 SMOD 是串行口波特率倍增位，当 SMOD=1 时，波特率加倍，复位时，SMOD=0。当 SMOD=0 时，波特率为 f_{osc} 的 1/64，若 SMOD=1 时，则波特率为 f_{osc} 的 1/32，即

$$\text{方式 2 的波特率} = \frac{2^{SMOD}}{64} \times f_{osc}$$

2. 方式 1 和方式 3 的波特率

80C51 串行口方式 1 和方式 3 的波特率由定时器 T1 的溢出率与 SMOD 值决定，即

$$\text{方式 1 和方式 3 的波特率} = \frac{2^{SMOD}}{32} \times \text{T1 溢出率}$$

其中，T1 溢出率取决于计数速率和定时器的预置值。计数速率与 TMOD 寄存器中 C/ \bar{T} 的状态有关。当 C/ \bar{T} =0 时，计数速率= $f_{osc}/12$ ；当 C/ \bar{T} =1 时，计数速率取决于外部输入时钟频率。

当定时器 T1 做波特率发生器使用时，通常选用自动重装载方式，即方式 2。在方式 2 中，TL1 做计数用，而自动重装载的值放在 TH1 内，设计数初值为 X，那么每过 “256-X” 个机器周期，定时器 1 就会产生一次溢出。为了避免因溢出而产生不必要的中断，此时应禁止 T1 中断。溢出周期为：

$$T = \frac{12}{f_{osc}} \times (256 - X)$$

溢出率为溢出周期之倒数，所以

$$\text{波特率} = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - X)}$$

则定时器 T1 方式 2 的初始值为：

$$X = 256 - \frac{f_{\text{osc}} \times (\text{SMOD} + 1)}{384 \times \text{波特率}}$$

表 7-2 列出在选择定时器 1 作为波特率发生器时，各种常用的波特率以及相应的控制位和时间常数。

表 7-2 定时器 T1 的常用波特率

波特率	f_{osc}	SMOD	定时器 1		
			C/T	模式	初值
方式 0: 1M	12MHz	×	×	×	×
方式 2: 375k	12MHz	1	×	×	×
方式 1、3:	62.5k	1	0	2	FFH
	19.2k	1	0	2	FDH
	9.6k	0	0	2	FDH
	4.8k	0	0	2	FAH
	2.4k	0	0	2	F4H
	1.2k	0	0	2	E8H
	110	0	0	2	72H
	110	0	0	1	FEE4H

例 1 已知 80C51 单片机时钟振荡频率为 11.0592MHz，选用定时器 T1 工作方式 2 做波特率发生器，波特率为 2400 波特，求时间常数。

解：设波特率控制位 SMOD=0，定时器 T1 的时间常数为：

$$X = 256 - \frac{11.0592 \times 10^6 \times (0 + 1)}{384 \times 2400} = 244 = \text{F4H}$$

所以，TH1=TL1= F4H。

由于上述公式包含除法，所以当晶振和波特率不同时，计算值有时会有一定误差。例如，如果晶振为 12MHz，波特率要求为 2400，在 SMOD=0 时，TH1=F3H，波特率的实际计算值为 2404，误差为 0.11%。但是，如果 2 个单片机之间的波特率相同，例如均为 2404，则不会影响通信，如果 2 个单片机之间的波特率误差超过 2.5%，则可能引起通信错误。

AT89S52 单片机的定时器 2 也可作为波特率发生器，其波特率设置见 6.5.2 节。

7.3 串行通信工作方式

通过软件编程可使串行通信有 4 种工作方式，下面分别予以介绍。

7.3.1 方式 0

在方式 0 下，串行口作为同步移位寄存器使用，以 8 位数据为一帧，先发送或接收最低位，每个机器周期发送或接收一位，故其波特率是固定的，为 $f_{\text{osc}}/12$ 。串行数据由 RXD (P3.0) 端输入或输出，同步移位脉冲由 TXD (P3.1) 端送出。这种方式常用于扩展 I/O 口，采用不同的指令实现输入或输出。其接收与发送情况如下。

1. 发送

当执行“MOV SBUF, A”指令时, CPU 将一字节数据写入发送缓冲寄存器 SBUF (99H), 串行口即把 8 位数据以 $f_{\text{osc}}/12$ 的波特率从 RXD 端送出 (低位在前); 发送完, 置中断标志 TI 为 1。如要继续发送必须清 TI 为 0。

2. 接收

因为 REN 是串行口允许接收控制位, 在准备接收时, 首先要用软件置 REN 为“1”, 使其允许接收。CPU 即开始从 RXD 端以 $f_{\text{osc}}/12$ 波特率输入数据 (低位在前); 当接收到 8 位数据时, 置中断标志 RI 为 1, 然后执行“MOVA, SBUF”指令, 就可读取数据。读取数据后一定要将 RI 清 0。

串行控制寄存器中 TB8 和 RB8 位在方式 0 中未用。每当发送或接收完 8 位数据时, 由硬件将发送中断 TI 或接收中断 RI 标志置为 1。不管中断方式还是查询方式, 都不会清除 TI 或 RI 标志, 必须用软件清 0。方式 0 时 SM2 位必须为 0。

7.3.2 方式 1

在方式 1 下, 串行口为 10 位通用异步接口。发送或接收一帧数据, 包括 1 位起始位 0, 8 位数据位和 1 位停止位 1。其传送波特率可调。

1. 发送

当执行“MOV SBUF, A”指令时, CPU 将一字节数据写入发送缓冲寄存器 SBUF (99H), 就启动发送器发送, 数据从引脚 TXD (P3.1) 端输出。当发送完一帧数据后, TI 标志置“1”。在中断方式下将申请中断, 通知 CPU 可以发送下一个数据。如要继续发送必须将 TI 清 0。

2. 接收

接收时, 使 REN 置 1 允许接收, 串行口采样引脚 RXD (P3.0)。当采样到 1~0 的跳变时, 确认是起始位“0”, 就开始接收一帧数据。当停止位到来之后把停止位送入 RB8 位, 则置位中断标志 RI, 在中断方式下将申请中断, 通知 CPU 从 SBUF 取走接收到的一个数据。

不管中断方式还是查询方式, 都不会清除 TI 或 RI 标志, 必须用软件清 0。

7.3.3 方式 2 和方式 3

方式 2 和方式 3 均为 11 位异步通信方式, 只是波特率的设置方法不同 (见 7.2.4 节), 其余完全相同。这 2 种方式发送或接收一帧的信息包括 1 位起始位 0, 8 位数据位, 1 位可编程位和 1 位停止位 1。其信息传送波特率与 SMOD 有关。

1. 发送

发送前, 先根据通信协议由软件设置 TB8 (如作为奇偶校验位或地址/数据标志位), 然

后将要发送的数据写入 SBUF 即能启动发送器。

发送过程是由执行任何一条以 SBUF 作为目的寄存器的指令而启动的。写“SBUF”指令，把 8 位数据装入 SBUF，同时串行口还自动把 TB8 装到发送移位寄存器的第 9 位数据位位置上，并通知发送控制器：要求进行一次发送，然后即从 TXD (P3.1) 端输出一帧数据。

2. 接收

在接收时，先置位 REN 为 1，使串行口处于允许接收状态，同时还要将 RI 清 0。在满足这个条件的前提下，再根据 SM2 的状态（因为 SM2 是方式 2 和方式 3 的多机通信控制位）和所接收到的 RB8 的状态才能决定此串行口在信息到来后是否会使 RI 置 1。如果 RI 置 1，在中断方式下将申请中断，接收数据。

当 SM2 = 0 时，不管 RB8 为 0 还是为 1，RI 都置 1，此串行口将接收发来的信息。

当 SM2=1，且 RB8 为 1 时，表示在多机通信情况下，接收的信息为地址帧，此时 RI 置 1，串行口将接收发来的地址。

当 SM2=1，且 RB8 为 0 时，表示接收的信息为数据帧，但不是发给本从机的，此时 RI 不置 1，因而 SBUF 中所接收的数据帧将丢失。

在方式 2 和方式 3 下，不管中断方式还是查询方式，都不会清除 TI 或 RI 标志，在发送和接收之后也必须用软件才能清 TI 和 RI 位。

7.3.4 多机通信

80C51 的方式 2 和方式 3 有一个专门的应用领域，即多机通信。这一功能使它可以方便地应用于集散式分布系统中，这种系统采用一台主机和多台从机，它们的通信方式之一如图 7-4 所示。

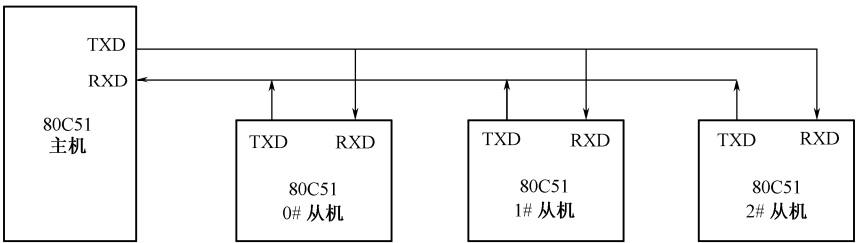


图 7-4 多机通信连接图

多机通信的实现，主要靠主、从机之间正确地设置与判断多机通信控制位 SM2 和发送或接收的第 9 数据位 (D8)。以下简述如何实现多机通信。

在编程前，首先要给各从机定义地址编号，如分别为 00H，01H，02H，…。当主机想发送一数据块给几个从机中的一个时，它首先送出一个地址字节，以辨认从机。地址字节和数据字节可用第 9 数据位来区别，主机的第 9 位应该设为 1。所以在主机发送地址帧时，地址/数据标志位 TB8 应设置为 1，以表示是地址帧。例如可这样编指令：

MOV SCON, #0D8H ; 设串行口为方式 3，TB8 置 1，准备发地址

此时，所有的从机初始化时均置 SM2=1，使它们只处于接收地址帧的状态。例如从机中可以编写这样的指令：

MOV SCON, #0F0H ; 置串行口为方式 3, SM2=1, 允许接收

当从机接收到从主机发来的信息后, 第 9 位 RB8 若为 1, 则置位中断标志 RI, 中断后判断主机送来的地址与本从机地址是否相符。若相符, 则被寻址的从机清除其 SM2 标志, 即 SM2=0, 准备接收即将从主机送来的数据帧, 未被选中的从机仍保持 SM2=1。

当主机发送数据帧时, 应该置 TB8 为 0, 此时虽然各从机都处于能接收的状态, 但由于 TB8=0, 只有 SM2=0 的那个被寻址的从机才能接收到数据, 那些未被选中的从机将不理睬进入到串行口的数据字节, 继续进行它们自己的工作, 直到一个新的地址字节到来, 这样就实现了主机控制的主从机之间的通信。综上所述, 通信只能在主从机之间进行, 从机之间的通信只有经主机才能实现。多机之间的通信过程可归纳如下。

- 主、从机均初始化为方式 2 或方式 3, 置 SM2=1, 允许多机通信。
- 主机置 TB8=1, 发送要寻址的从机地址。
- 所有从机均接收主机发送的地址, 并进行地址比较。
- 被寻址的从机确认地址后, 如果符合, 置本机 SM2=0, 向主机返回地址, 供主机核对; 如果不符合, 置本机 SM2=1。
- 核对无误后, 主机向被寻址的从机发送命令, 通知从机接收或发送数据。
- 通信只能在主、从机之间进行, 两个从机之间的通信需通过主机作中介。
- 本次通信结束后, 主、从机重置 SM2=1, 主机可再寻址其他从机。

有关多机通信的编程实例可参看文献 2。

在实际应用中, 因为单片机功能有限, 因而在较大的测控系统中, 常常把单片机应用系统作为前端机 (也称为下位机或从机) 直接用于控制对象的数据采集与控制, 而把 PC 机作为中央处理机 (也称为上位机或主机) 用于数据处理和对下位机的监控管理。它们之间的信息交换, 主要采用串行通信, 此时单片机可直接利用其串行接口, 再配上电平转换芯片, 而 PC 机可利用其配备的 RS-232 接口 (使用方法可查看有关手册)。实现单片机与 PC 机串行通信的关键是在通信协议的约定上要一致, 例如, 设定相同的波特率及帧格式等。在正式工作之前, 双方应先互发联络信号, 以确保通信收发数据的准确性。

7.4 串行口应用举例

本节将介绍串行口在做 I/O 扩展及一般异步通信和多机通信中的应用原理及实例。

7.4.1 用串行口扩展 I/O 口

串行口的方式 0 主要用于扩展并行 I/O 口。此处将给出实用线路和简单的控制程序。

例 1 用并行输入 8 位移位寄存器 74HC165 扩展 16 位并行输入口。编程实现从 16 位扩展口读入 20 字节数据, 并把它们转存到内部 RAM 的 50H~63H 中。

解: 在此采用 74HC165 与单片机相接实现 I/O 口扩展, 单片机与 74HC165 的具体接线如图 7-5 所示。本线路利用 80C51 的 3 根口线扩展为 16 根输入口线的实用电路, 由 2 块 74HC165 串接而成。74HC165 是 TTL 并入串出移位寄存器 (也可选用其他同样功能的 CMOS 器件), 74HC165 引脚图见附录 E, 图中 CK 为时钟脉冲输入端, D0~D7 为并行输入端, SIN、QH 分别为数据的输入、输出端。前级的数据输出端 QH 与后级的信号输入端 SIN 相连,

$S/\bar{L}=0$ 时允许并行置入数据, $S/\bar{L}=1$ 时允许串行移位。

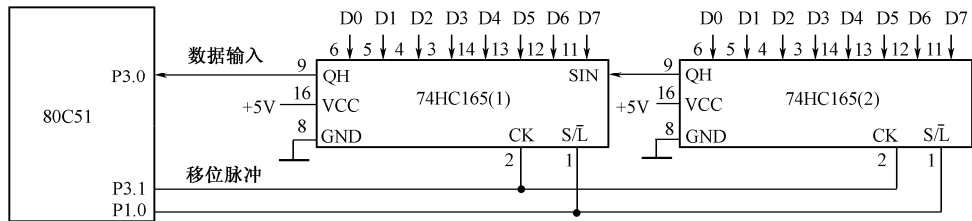


图 7-5 利用串行口扩展输入口

按题意用汇编语言编程如下:

```

MOV     R7,      # 20      ; 设置读入字节数
MOV     R0,      # 50H     ; 设片内 RAM 指针
SETB    F0        ; 设置读入字节奇偶数标志
RCV0:   CLR      P1.0      ; 允许并行置入数据
        SETB     P1.0      ; 允许串行移位
RCV1:   MOV      SCON,    # 10H ; 设串行口方式 0 并启动接收
        JNB      RI,      $   ; 等待接收一帧数据
        CLR      RI        ; 清接收中断标志
        MOV      A,       SBUF ; 取缓冲器数据
        MOV      @R0,     A
        INC      R0
        CPL      F0
        JB       F0,      RCV2 ; 判断是否接收完偶数帧, 接收完则重新并行置入
        DEC      R7
        SJMP     RCV1
RCV2:   DJNZ     R7,      RCV0 ; 判断是否已读入预定的字节数
        .....             ; 对读入数据进行处理

```

C51 语言程序如下:

```

#include <REG51.H>                //定义 51 寄存器
unsigned char Data_buf[20] _at_ 0x50; //定义输入数据数组存在 50H 开始的 RAM
unsigned char i;                  //定义变量
sbit Input_C = P1^0;              //定义输入引脚
void main (void)
{
    Input_C = 0;                  //允许并行置入数据
    Input_C = 1;                  //允许串行移位
    SCON = 0x10;                  //设串行口方式 0 并启动接收
    for (i=0;i<20;i++)
    {
        while(RI==0);            //等待接收 1 帧数据
        Data_buf[i] = SBUF;       //存入接收数组
        RI = 0;                  //清接收中断标志位
        i++;                     //
        while(RI==0);            //等待接收 1 帧数据
        Data_buf[i] = SBUF;       //存入接收数组
        RI = 0;                  //清接收中断标志位
    }
}

```

```

    }
    ...
    while(1);
}
//数据接收完毕
//进行数据的后续处理
//程序停止

```

汇编程序中 F0 用来做读入字节数的奇偶性标志。由于每次由扩展口并行置入到移位寄存器的是两字节数据，置入一次，串行口应接收二帧数据，故已接收的数据字节数为奇数时 F0=0，不再并行置入数据就直接启动接收过程；否则 F0=1，在启动接收过程前，应该先在外移位寄存器中置入新的数据。而在 C51 语言程序中，1 次循环就接受 2 字节，不再需要进行奇偶字节判别。

例 2 用两片 8 位串入并出移位寄存器 74HC164 扩展 16 位输出接口。

图 7-6 是利用 74HC164（也可选用其他同样功能的 CMOS 器件）扩展的 16 位发光二极管接口电路。编程使这 16 个发光二极管交替为间隔点亮状态，循环交替时间为 2s。

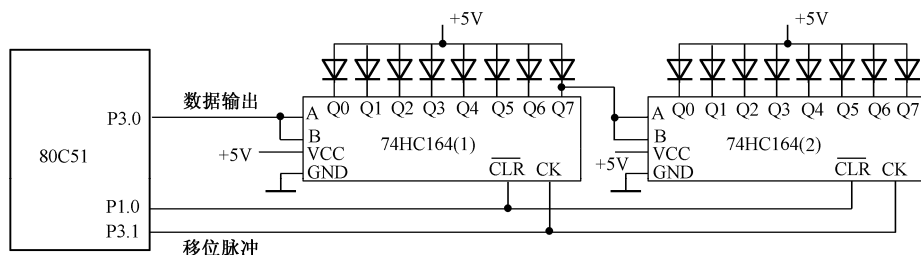


图 7-6 利用串行口扩展输出接口

解：在本电路中 74HC164 是 TTL 串行输入、并行输出移位寄存器，其外部引脚见附录 E。Q0~Q7 为并行输出端；A、B 为串行输入端， $\overline{\text{CLR}}$ 为清除端，0 电平时，使 74HC164 输出清 0；CK 为时钟脉冲输入端，在 CK 脉冲的上升沿作用下实现移位。在 CK=0， $\overline{\text{CLR}}$ =1 时，74HC164 保持原来数据状态。由于 74HC164 无并行输出控制端，在串行输入过程中，其输出端的状态会不断变化，故在某些使用场合，在 74HC164 与输出装置之间，还应加上输出可控的缓冲级（如 74HC244），以便串行输入过程结束后再输出。图 7-6 中的输出装置是 16 位发光二极管，由于 74HC164 在低电平输出时，允许通过的电流可达 8mA，故不需再加驱动电路。

按题意用汇编语言编程如下：

```

ST:      MOV     SCON, #00H    ; 设串行口方式 0
          MOV     A,      #55H  ; 二极管间隔点亮初值
LP2:     MOV     R0,      #2    ; 输出口字节数
          CLR     P1.0        ; 对 74HC164 清 0，熄灭所有发光二极管
          SETB    P1.0        ; 允许数据串行移位
LP1:     MOV     SBUF,    A     ; 启动串行口发送
          JNB     TI,      $     ; 等待一帧发送结束
          CLR     TI         ; 清串行口发送中断标志
          DJNZ    R0,      LP1   ; 判断预定字节数送完否
          LCALL   DEL2s      ; 调延时 2s 子程序（略）
          CPL     A           ; 交替点亮二极管
          SJMP    LP2        ; 循环显示

```

C51 语言程序如下：

```

#include <REG51.H>
unsigned char i;

```

```

sbit Output_C = P1^0;
void main (void)
{
    unsigned char Out_data;
    SCON = 0x0;
    Out_data = 0x55;
    while(1)                                //程序循环输出
    {
        for (i=0;i<2;i++)
        {
            Output_C = 0;                    //对 74HC164 清 0
            Output_C = 1;                    //允许数据串行移位
            SBUF = Out_data;                 //启动串行口发送
            while(TI == 0);                  //等待 1 帧发送结束
            TI = 0;
        }
        Delay2S(); //延时可通过软件或定时器实现（省略）
        Out_data = ~Out_data;               //交替点亮二极管
    }
}

```

从理论上讲，74HC164 或 74HC165 可以无限地串级上去，进一步扩展输入/输出并行口，但这种扩展方法，输入/输出的速度是不高的，移位时钟频率为 $f_{osc}/12$ ，若 $f_{osc}=12\text{MHz}$ ，则每移一位需 $1\mu\text{s}$ 。

7.4.2 用串行口进行异步通信

双机异步通信的连接线路如图 7-7 所示。下面举例分别介绍甲机发送，乙机接收的程序编制。

例 3 编程把甲机片内 RAM 60H~7FH 单元中的数据块从串行口输出。定义在工作方式 3 下发送，TB8 作奇偶校验位。采用定时器 1 方式 2 作为波特率发生器，波特率为 4800， $f_{osc}=11.0592\text{MHz}$ ，定时器初始预置值 $\text{TH1}=\text{TL1}=0\text{FAH}$ 。

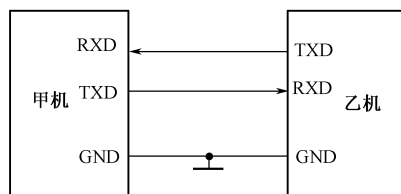


图 7-7 双机异步通信连接图

编程使乙机从甲机接收 32 字节数据块，并存入片外 1000H~101FH 单元。接收过程中要求判断奇偶校验标志 RB8。若出错置 F0 标志为 1；如果正确置 F0 标志为 0，然后返回。

解：编写汇编语言发送子程序如下：

```

MOV     TMOD, # 20H           ; 设置定时器 1 为方式 2
MOV     TL1,  # 0FAH          ; 设预置值
MOV     TH1,  # 0FAH
SETB    TR1                   ; 启动定时器 1
MOV     SCON, # 0C0H          ; 设置串行口为方式 3
MOV     PCON, # 00H           ; SMOD=0
MOV     R0,   # 60H           ; 设数据块指针
MOV     R7,   # 20H           ; 设数据长度 20H
TRS:    MOV     A, @R0         ; 取数据送 A

```

	MOV	C,	P	
	MOV	TB8,	C	; 奇偶位 P 送 TB8
	MOV	SBUF,	A	; 数据送 SBUF, 启动发送
WAIT:	JNB	TI,	\$; 判断一帧是否发送完
	CLR	TI		
	INC	R0		; 更新数据单元
	DJNZ	R7,	TR5	; 循环发送至结束
	RET			; 返回

C51 语言程序如下:

```
#include <REG52.H>
unsigned char Uart_buf[32] _at_ 0x60;
void main (void)
{
    unsigned char i;
    TMOD = 0x20;           //设置定时器 T1 为方式 2
    TL1 = 0xfa;            //设预置值
    TH1 = 0xfa;
    TR1 = 1;               //启动定时器 T1
    SCON = 0xc0;           //设置串行口为方式 3
    PCON = 0x00;           // SMOD=0
    for(i=0;i<32;i++)
    {
        ACC =Uart_buf[i];  //将要发送的数据送到累加器, 改变 P 值
        TB8 = P;           //将 P 值送至 TB8
        SBUF = Uart_buf[i]; //发送数据
        while(TI==0);      //等待发送完
        TI = 0;            //清标志位
    }
    while(1);
}
```

在进行双机通信时, 两机应用相同的工作方式和波特率, 因而接收汇编子程序编程如下:

	MOV	TMOD,	# 20H	; 设置定时器 1 为方式 2
	MOV	TL1,	# 0FAH	; 设预置值
	MOV	TH1,	# 0FAH	
	SETB	TR1		; 启动定时器 1
	MOV	SCON,	# 0C0H	; 设置串行口为方式 3
	MOV	PCON,	# 00H	; SMOD=0
	MOV	DPTR,	# 1000H	; 设置数据块指针
	MOV	R7,	# 20H	; 设数据块长度
	SETB	REN		; 允许接收
WAIT:	JNB	RI,	\$; 判断一帧是否接收完
	CLR	RI		
	MOV	A,	SBUF	; 读入一帧数据
	JNB	PSW.0,	PZ	; 奇偶位 P 为 0 则转
	JNB	RB8,	ERR	; P=1, RB8=0 则出错
	SJMP	YES		; 二者全为 1 则正确
PZ:	JB	RB8,	ERR	; P=0, RB8=1 则出错
YES:	MOVX	@ DPTR,	A	; 正确, 存放数据

	INC	DPTR	; 修改地址指针
	DJNZ	R7, WAIT	; 判断数据块接收完否
	CLR	PSW.5	; 接收正确, 且接收完清 F0 标志
	RET		; 返回
ERR:	SETB	PSW.5	; 出错置 F0 标志为 1
	RET		; 返回

C51 语言程序如下:

```
#include <REG52.H>
xdata unsigned char Uart_buf[32] _at_ 0x1000;
void main (void)
{
    unsigned char i;
    TMOD = 0x20;
    TL1 = 0xfa;
    TH1 = 0xfa;
    TR1 = 1;
    SCON = 0xc0;
    PCON = 0x00;
    REN = 1; //允许接收
    for(i=0;i<32;i++)
    {
        while(RI==0); //等待数据接收完毕
        RI = 0;
        ACC = SBUF; //数据送入累加器中
        if (P^RB8==1) //判断奇偶校验位
        {
            Uart_buf[i] = SBUF;
        }else
        {
            F0 = 1;
        }
    }
    while(1);
}
```

上例是在方式 3 下进行收发, 并用奇偶位进行校验。下面再介绍在方式 1 下进行双机通信, 用累加和进行校验的方法。

例 4 设甲机、乙机通过串口通信, 两机晶振均为 24Mhz, 波特率设为 9600。要求甲机将外部数据存储器 100H~109H 单元的数据向乙机发送。甲机依次将数据长度、数据块及数据块的累加和发送给乙机, 乙机根据接收到的数据长度, 依次将数据存入片外以 1000H 为首地址的数据存储区, 并计算接收数据的累加和; 所有数据接收完毕后, 进行累加和校验, 若校验正确, 向甲机发送正确标志 0FH, 否则发送 0F0H。

解: 因波特率已指定为 9600, 首先需要确定定时初值和 SMOD 的值。初值按下式计算:

$$X = 256 - \frac{f_{osc} \times (SMOD + 1)}{\text{波特率} \times 384} = 256 - \frac{24 \times 10^6 \times (SMOD + 1)}{9600 \times 384}$$

当取 SMOD=0, 得 $X = 249.49 \approx 249 = F9H$, 此时实际波特率 $= \frac{2^{SMOD}}{384} \times \frac{f_{osc}}{256 - X} \approx 8928$,

误差 7%。而取 SMOD =1, 得 $X=242.98 \approx F3H$, 实际波特率 $= \frac{2^{SMOD}}{384} \times \frac{f_{osc}}{256-X} \approx 9615$, 误差 0.16%, 即 SMOD 取不同值时, 实际波特率的大小是不一样的。程序中取 SMOD=1。

发送程序约定:

- 定时器 T1 按方式 2 工作, 计数初值为 F3H, SMOD =1;
- 串行口按方式 1 工作, 允许接收;
- R6 设为数据块长度寄存器, R5 设为累加和寄存器。

汇编语言程序如下:

甲机发送程序清单:

```

TRT:  MOV    TMOD,    #20H           ; 设定器 1 工作在方式 2
      MOV    TH1,     #0F3H         ; 设定器 1 初值
      MOV    TL1,     #0F3H
      SETB   TR1           ; 启动定时器 1
      MOV    SCON,    #50H         ; 串行口初始化为方式 1, 允许接收
      MOV    PCON,    #80H         ; SMOD=1
RPT:  MOV    DPTR,    #100H         ; 待发送数据的首地址
      MOV    R6,      #0AH         ; 长度寄存器初始化, 长度为 10
      MOV    R5,      #00H         ; 校验和寄存器初始化
      MOV    SBUF,    R6           ; 发送长度
      JNB    TI,$           ; 等待发送
      CLR    TI
L1:   MOVX   A,        @DPTR         ; 读取数据
      MOV    SBUF,    A           ; 发送数据
      ADD    A,       R5           ; 形成累加和送 R5
      MOV    R5,      A
      INC    DPTR           ; 修改地址指针
      JNB    TI,      $           ; 等待发送
      CLR    TI
      DJNZ   R6,      L1          ; 判断是否发送完 10 个字节
      MOV    SBUF,    R5           ; 发送校验码
      MOV    R5,      #00H
      JNB    TI,      $
      CLR    TI
      JNB    RI,      $           ; 等待乙机回答
      CLR    RI
      MOV    A,       SBUF
      CJNE   A,       #0FH,ERR     ; 数据发送正确则返回
      RET
ERR:  ... ..                    ; 数据发送出错处理

```

乙机接收程序清单:

```

RSU:  MOV    TMOD,    #20H           ; T1 初始化
      MOV    TH1,     #0F3H
      MOV    TL1,     #0F3H
      SETB   TR1
      MOV    SCON,    #50H
      MOV    PCON,    #80H

```

RPT:	MOV	DPTR,	#1000H	; 接收数据存放首地址
	JNB	RI,	\$	
	CLR	RI		
	MOV	A,	SBUF	; 接收发送长度
	MOV	R6,	A	
	MOV	R5,	#00H	; 累加和寄存器清 0
WTD:	JNB	RI,	\$	
	CLR	RI		
	MOV	A,	SBUF	; 接收数据
	MOVX	@DPTR,	A	; 存储数据
	INC	DPTR		; 修改地址指针
	ADD	A,	R5	; 形成累加和
	MOV	R5,	A	
	DJNZ	R6,	WTD	; 未接收完, 继续
	JNB	RI,	\$; 接收校验码
	CLR	RI		
	MOV	A,	SBUF	
	XRL	A,	R5	; 比较校验码
	MOV	R5,	#00H	
	JZ	L6		; 正确转 L6
	MOV	SBUF,	#0F0H	; 出错送 0F0H
	JNB	TI,	\$	
	CLR	TI		
	RET			
L6:	MOV	SBUF,	#0FH	; 正确回送 0FH
	JNB	TI,	\$; 发送完返回
	CLR	TI		
	RET			

C51 语言程序如下:

甲机发送程序清单:

```

#include<reg51.h>
#define uchar unsigned char
uchar xdata * send_pt;           //待发送的数据
void main(void)
{
    uchar addtest,temp,i;
    send_pt=0x100;               //待发送数据的首地址
    SCON=0x50;                   //定时器及串口初始化, 波特率 9600
    PCON=0X80;
    TMOD=0X20;
    TH1=0xF3;
    TL1=0XF3;
    TR1=1;
    for(;;)
    {
        .....                  //其他程序
        addtest=0;               //校验和变量初始化
        SBUF=10;                 //发送数据长度
    }
}

```

```

        while(TI==0);
    TI=0;
    for(i=0;i<10;i++)
    {
        addtest=addtest+*(send_pt+i);    //形成累加和
        SBUF=*(send_pt+i);              //发送数据
        while(TI==0);
        TI=0;
    }
    SBUF=addtest;                        //发送校验和
    while(TI==0);
    TI=0;
    while(RI==0);                        //等待接收数据
    temp=SBUF;
    RI=0;
    if(temp==0xf0)                       //接收到 0xf0, 表示出错, 转出错处理程序
    {
        Error();
    }
    .....                               //其他程序
}
}

```

乙机接收程序清单:

```

#include<reg51.h>
#define uchar unsigned char
uchar  xdata * recv_pt;                //接收数据存放区地址指针

void main(void)
{
    uchar addtest,temp,i;
    recv_pt=0x1000;                    //接收数据存放区地址
    SCON=0x50;                         //定时器及串口初始化, 波特率 9600
    PCON=0X80;
    TMOD=0X20;
    TH1=0xF3;
    TL1=0XF3;
    TR1=1;
    for(;;)
    {
        .....                          //其他程序
        addtest=0;                      //校验和变量初始化
        while(RI==0);                  //接收数据的长度
        temp=SBUF;
        RI=0;
        for(i=0;i<temp;i++)            //循环接收数据并做累加和计算
        {
            while(RI==0);
            *(recv_pt+i) = SBUF;
        }
    }
}

```

```

        RI=0;
        addtest=addtest+*(recv_pt+i);
    }
    while(RI==0);                //接收校验和字节
    temp=SBUF;
    RI=0;
    if(temp==addtest)            //比较校验，正确，发送 0x0F
    {
        SBUF=0x0f;
    }
    else
    {
        SBUF=0xf0;                //出错，发送 0xF0
    }
    while(TI==0);
    TI=0;
    //.....                    //其他程序
}
}

```

思考与练习

1. 什么是串行异步通信？它有哪些特点？有哪几种帧格式？
2. 某异步通信接口按方式 3 传送，已知其每分钟传送 3600 个字符，计算其传送波特率。
3. 80C51 单片机的串行口由哪些基本功能部件组成？简述工作过程。
4. 80C51 单片机的串行口有几种工作方式？几种帧格式？如何设置不同方式下的波特率？
5. 为什么定时器 T1 用做串行口波特率发生器时，常采用工作方式 2？若已知系统时钟频率、通信选用的波特率，如何计算其初值？
6. 已知定时器 T1 设置成方式 2，用做波特率发生器，系统时钟频率为 24MHz，求可能产生的最高和最低的波特率是多少？
7. 设计一个 AT89S51 单片机的双机通信系统，并编写程序将甲机片外 RAM 3400H~3420H 的数据块通过串行口传送到乙机的片内 RAM 40H~60H 单元中去。
8. 利用 AT89S51 串行口控制 8 位发光二极管工作，要求发光二极管每隔 1s 交替地亮灭，画出电路并编写程序。

第8章 中断系统

中断技术是计算机中的重要技术之一，它既和硬件有关，也和软件有关。正因为有了中断，才使得计算机具备了对外部事件进行处理的能力，使得工作更灵活，效率更高。本章将介绍中断的概念，并以 80C51 系列单片机的中断系统为例介绍中断的处理过程及应用。

8.1 概 述

在程序正常运行时，对 CPU 外部随机或定时（如定时器发出的信号）出现的紧急事件，常常需要 CPU 能马上响应，为了解决这一问题，在计算机中引入了“中断”技术。

8.1.1 中断的概念

计算机（包含单片机）在执行程序的过程中，由于 CPU 以外的某种原因，必须尽快中止当前程序的执行，而去执行相应的处理程序，待处理结束后，再回来继续执行被中止了的原程序。这种程序在执行过程中由于外界的原因而被中间打断的情况称为“中断”。中断是通过硬件来改变 CPU 程序运行方向的一种技术，它既和硬件有关，也和软件有关。

“中断”之后所执行的处理程序，通常称为中断服务或中断处理子程序，原来运行的程序称为主程序。主程序被断开的位置（地址）称为断点。引起中断的原因，或能发出中断申请的来源，称为中断源。中断源要求服务的请求称为中断请求（或申请）。

调用中断服务程序的过程有些类似于程序设计中的调用子程序，主要区别在于调用子程序指令在程序中是事先安排好的，而何时调用中断服务程序事先却无法确知。因为“中断”的发生是由外部因素决定的，程序中无法事先安排调用指令，因而调用中断服务程序的过程是由硬件自动完成的（详见 8.3 节）。

8.1.2 引进中断技术的优点

计算机引进中断技术之后主要有如下优点。

1. 分时操作

在单片机与外部设备交换信息时，存在着高速的 CPU 和低速的外设（如打印机等）之间的矛盾。若采用软件查询的方式，不但占用了 CPU 操作时间，而且响应速度慢。有了中断功能就解决了快速的 CPU 与慢速的外设之间的矛盾。此时，CPU 在启动外设工作后，继续执行主程序，同时外设也在工作。每当外设做完一件事，就发出中断申请，请求 CPU 中断它正在执行的程序，转去执行中断服务程序（一般情况是处理输入输出数据），中断处理完之后，CPU 恢复执行被打断的主程序，外设仍继续工作。这样 CPU 可以命令多个外设（如键盘、打印机等）同时工作，从而大大提高了 CPU 的工作效率。

2. 实现实时处理

在实时控制中，现场的各个参数、信息是随时间和现场情况不断变化的。有了中断功能，外界的这些变化量可根据要求，随时向 CPU 发出中断请求，要求 CPU 及时处理，CPU 就可以立刻响应（若中断响应条件满足）加以处理。这样的及时处理在查询方式下是做不到的，从而大大减少了 CPU 的等待时间。

3. 故障处理

单片机在运行过程中，难免会出现一些事先无法预料的故障，如存储出错，运算溢出，电源突跳等。有了中断功能，单片机就能自行处理，而不必停机处理。

8.1.3 中断源

发出中断请求的来源一般统称为中断源，中断源有多种，最常见的中断源有以下 4 种。

1. 外部设备中断源

单片机的输入、输出设备，如键盘、磁盘驱动器、打印机等，可通过接口电路向 CPU 申请中断。

2. 故障源

故障源是产生故障信息的来源，把它作为中断源使 CPU 能够以中断方式对已发生的故障进行及时处理。

单片机故障源有内部和外部之分，CPU 内部故障源，如除法中除数为零等情况；外部故障源主要有电源掉电等情况。在电源掉电时可以接入备用的电池供电，以保存存储器中的信息不丢失。当电压因掉电降到一定值时，就发出中断申请，由单片机的中断系统自动响应并进行相应处理。

3. 控制对象中断源

单片机做实时控制时，被控对象常常用做中断源。例如，电压、电流、温度等超越上限或下限时，以及继电器、开关闭合断开时都可以作为中断源申请中断。

4. 定时/计数脉冲中断源

定时/计数脉冲中断源也有内部和外部之分。内部定时中断是由单片机内部的定时/计数器溢出时自动产生的，外部计数中断是由外部脉冲通过 CPU 的中断请求输入线或定时/计数器的输入线引起的。

要求每个中断源所发出的中断请求信号符合 CPU 响应中断的条件，例如，电平的高、低，持续的时间，脉冲的幅度等。

8.1.4 中断系统的功能

为了满足上述各种情况下的中断要求，中断系统一般具有以下功能。

1. 能实现中断及返回

当某一个中断源发出中断申请时，CPU 能决定是否响应这个中断请求，当 CPU 在执行更急、更重要的工作时，可以暂不响应中断；若允许响应这个中断请求，CPU 必须在现行的指令执行完后，再把断点处的 PC 值（即下一条应执行的指令地址）推入堆栈保留下来，这称为保护断点，这一步是硬件自动执行的。同时用户在编程时，要注意把有关的寄存器内容和状态标志位推入堆栈保留下来，这称为保护现场。保护断点和现场之后即可执行中断服务程序，执行完毕，需恢复原保留寄存器的内容和标志位的状态，称恢复现场。执行返回指令 RETI，这个过程由用户编程实现。RETI 指令的功能即恢复 PC 值（称为恢复断点），使 CPU 返回断点，继续执行主程序，这个过程如图 8-1 所示。

2. 能实现优先权排队

通常，在系统中有多个中断源，有时会出现两个或多个中断源同时提出中断请求的情况。这就要求单片机既能区分各个中断源请求，又能确定首先为哪一个中断源服务。为了解决这一问题，通常给各中断源规定了优先级别，称为优先权。当两个或两个以上的中断源同时提出中断请求时，首先为优先权最高的中断源服务，服务结束后，再响应级别较低的中断源。按中断源级别高低逐次响应的过程称优先权排队。这个过程可以通过硬件电路来实现，也可以通过程序查询来实现。

3. 能实现中断嵌套

当 CPU 响应某一中断请求，正在进行中断处理时，若有优先权级别更高的中断源发出中断申请，则 CPU 能中断正在进行的 interrupt 服务程序，并保留这个程序的断点（类似于子程序嵌套），响应高级中断；在高级中断处理完以后，再继续执行被中断的中断服务程序。这个过程称中断嵌套，如图 8-2 所示。如果发出新的中断申请的中断源的优先权级别与正在处理的中断源同级或更低时，则 CPU 不响应这个中断申请，直至正在处理的中断服务程序执行完以后才去处理新的中断申请。

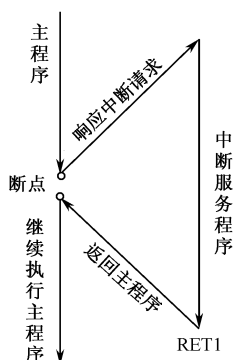


图 8-1 中断流程图

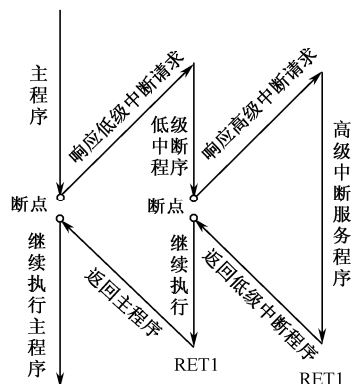


图 8-2 中断嵌套流程图

8.2 AT89S51 单片机的中断系统

由上节所述可知，中断功能是在硬件基础上再配以相应的软件而实现的。不同的单片机其硬件结构和软件指令是不完全相同的，因而中断系统结构一般是不相同的。但同一系列的单片机即使型号不同，中断系统的基本结构也是类似的，只是中断源个数不完全一样。本节将以 80C51 系列单片机中的 AT89S51 单片机中断系统为例进行介绍。

8.2.1 中断系统的结构

AT89S51 的中断系统主要由几个与中断有关的特殊功能寄存器、中断允许、顺序查询逻辑电路等组成。AT89S51 的中断系统结构框图如图 8-3 所示，AT89S51 单片机有 5 个中断源，可提供两个中断优先级，即可实现二级中断嵌套。与中断有关的特殊功能寄存器有 4 个，分别为中断源寄存器（即专用寄存器 TCON、SCON 的相关位）、中断允许控制寄存器 IE 和中断优先级控制寄存器 IP。5 个中断源的排列顺序由中断优先级控制寄存器 IP 和顺序查询逻辑电路（图 8-3 中的硬件查询）共同决定。5 个中断源对应 5 个固定的中断入口地址，亦称矢量地址。

下面分别对 AT89S51 的中断源及专用寄存器等进行介绍。

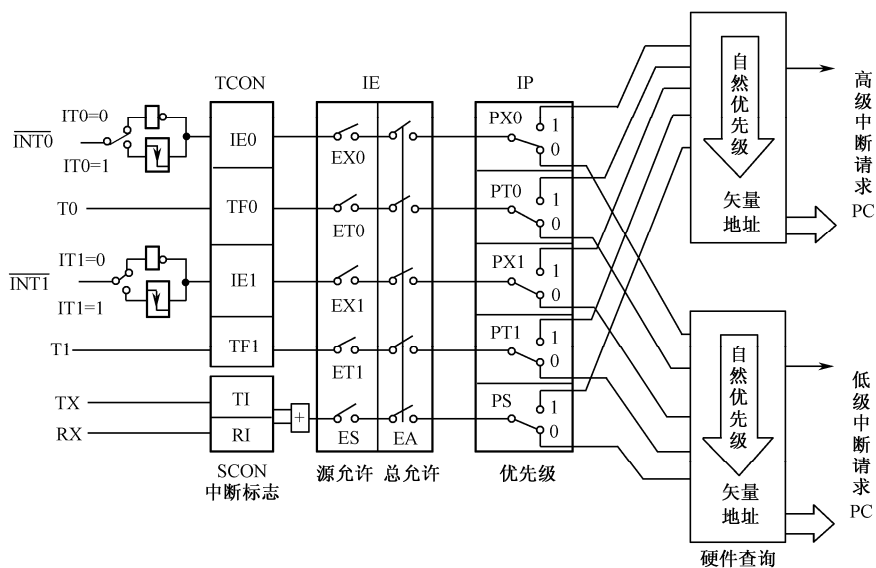


图 8-3 AT89S51 的中断系统结构

8.2.2 中断源及中断入口

1. 中断源

AT89S51 的中断源可以分为 3 类，即外部中断、定时中断和串行口中断。从图 8-3 系统结构可见，AT89S51 单片机有 5 个中断请求源，分别为：两个外部输入中断源 $\overline{\text{INT0}}$ （P3.2）和 $\overline{\text{INT1}}$ （P3.3），两个片内定时器 T0 和 T1 的溢出中断源 TF0（TCON.5）和 TF1（TCON.7），

1 个片内串行口发送和接收中断源 TI (SCON.1) 和 RI (SCON.0)。AT89S52 单片机的中断源增加了一个定时器 T2 的中断源。

(1) 外部中断类

外部中断是由外部原因 (详见 8.1 节) 引起的, 可以通过两个固定引脚, 即外部中断 0 ($\overline{\text{INT0}}$) 和外部中断 1 ($\overline{\text{INT1}}$) 输入信号。

$\overline{\text{INT0}}$ ——外部中断 0 请求信号, 由 P3.2 脚输入。通过 IT0 (TCON.0) 来决定中断请求信号是低电平有效还是下降沿有效。一旦输入信号有效, 则向 CPU 申请中断, 并且使相应的中断标志位 IE0=1。

$\overline{\text{INT1}}$ ——外部中断 1 请求信号, 由 P3.3 脚输入。通过 IT1 (TCON.2) 来决定中断请求信号是低电平有效还是下降沿有效。一旦输入信号有效, 则向 CPU 申请中断, 并且使相应的中断标志位 IE1=1。

(2) 定时中断类

定时中断是为满足定时或计数溢出处理的需要而设置的, 当定时/计数器中的计数值发生溢出时, 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使溢出标志位 TFX 置为 1。这种中断请求是在单片机芯片内部发生的, 无须在芯片上设置引入端, 但在计数方式时, 中断源可以由外部引入 (见 8.4 节)。

TF0——定时器 T0 溢出中断请求。当定时器 T0 产生溢出时, T0 中断请求标志 TF0=1, 请求中断处理。

TF1——定时器 T1 溢出中断请求。当定时器 T1 产生溢出时, T1 中断请求标志 TF1=1, 请求中断处理。

(3) 串行口中断类

串行口中断是为串行数据的传送需要而设置的。串行中断请求也是在单片机芯片内部发生的, 但当串行口作为接收端时, 必须有一完整的串行帧数据从 RXD (P3.0) 端引入芯片, 才可能引发中断。

RI 或 TI——串行中断请求。当接收或发送完一串行帧数据时, 使内部串行口中断请求标志 RI 或 TI=1, 并请求中断。

由 AT89S51 的中断系统结构框图可以看出, 当这些中断源的中断标志为 1 时, 并不是肯定能引起中断, 还需要经过 IE 寄存器的控制 (见 8.2.3 节), 才能进入中断请求。

2. 中断入口

当某中断源的中断申请被 CPU 响应之后, CPU 将把此中断源的入口地址装入 PC, 中断服务程序即从此地址开始执行。因而将此地址称中断入口, 也称中断矢量。在 AT89S51 单片机中各中断源以及与之对应的入口地址 (由单片机硬件电路决定) 分配如下:

中断源	入口地址
外部中断 0	0003H
定时器 T0 中断	000BH
外部中断 1	0013H
定时器 T1 中断	001BH
串行口中断	0023H
定时器 T2 中断	002BH (仅 AT89S52 才有)

所有的 80C51 系列单片机都有前 5 个中断源，89 系列单片机也不例外，有些型号与其完全相同，例如 Philip 公司的 P89C51；有些则增加了新的中断源，例如，AT89S52 增加了定时器 T2 中断源，入口地址为 002BH。还有一些型号中断源多达 9 个，其入口地址按 8 字节一个中断源顺序往下排，可以表达为：入口地址 = $8n+3$ ； n 为中断优先级。

8.2.3 与中断控制有关的寄存器

中断功能虽然是硬件和软件结合的产物，但对于用户来说重点是怎样通过软件实现中断控制功能。为此首先应该掌握与中断控制有关的几个寄存器，下面分别对其予以介绍。

1. 中断允许控制寄存器

专用寄存器 IE 是 80C51 单片机中的中断允许寄存器，由它控制对中断的开放或关闭。通过向 IE 写入中断控制字，可以实现对中断的 2 级控制，这里所说的 2 级是指控制字中有一个中断总允许位 EA，EA 为 0 时将屏蔽所有的中断申请，而当 EA 为 1 时，虽然 CPU 已经开放中断，但还需要设置相应中断源的控制位，才可确定允许哪个中断源中断。

IE 的格式如下。

IE	AFH		ADH	ACH	ABH	AAH	A9H	A8H
(A8H)	EA	—	ET2*	ES	ET1	EX1	ET0	EX0

其各位作用如下：

IE.7 EA——中断允许总控制位。

当 EA=1，CPU 开放中断，每个中断源是被允许还是被禁止，分别由各自的允许位确定。

当 EA=0，CPU 屏蔽所有的中断要求，称关中断。

IE.4 ES——串行口中断控制位。

当 ES=1，允许串行口中断；

当 ES=0，禁止串行口中断。

IE.3 ET1——T1 中断控制位。

当 ET1=1，允许 T1 中断；

当 ET1=0，禁止 T1 中断。

IE.2 EX1——外部中断 1 控制位。

当 EX1=1，允许外部中断 1 中断；

当 EX1=0，禁止外部中断 1 中断。

IE.1 ET0——T0 中断控制位。

当 ET0=1，允许 T0 中断；

当 ET0=0，禁止 T0 中断。

IE.0 EX0——外部中断 0 允许位。

当 EX0=1，允许外部中断 0 中断；

当 EX0=0，禁止外部中断 0 中断。

IE.5 ET2*——T2 中断允许位（仅 AT89S52/80C52 或类似型号单片机有）。

当 ET2=1，允许 T2 中断；

当 $ET2 = 0$ ，禁止 $T2$ 中断。

AT89S51 单片机复位后，IE 中各中断允许位均被清 0，即禁止所有中断。

2. 中断请求标志寄存器

当有中断源发出申请时，由硬件将相应的中断标志位置为 1。在中断请求被响应前，相应中断标志位被锁存在特殊功能寄存器 TCON 或 SCON 中。

(1) TCON 中的中断标志

TCON 为定时器 $T0$ 和 $T1$ 的控制寄存器，同时也锁存 $T0$ 和 $T1$ 的溢出中断标志及外部中断 $\overline{INT0}$ 和 $\overline{INT1}$ 的中断标志等。

TCON 中与中断有关的位如下。

TCON	8FH		8DH		8BH	8AH	89H	88H
(88H)	TF1		TF0		IE1	IT1	IE0	IT0

其每个位作用如下：

TCON.7 TF1—— $T1$ 溢出中断标志。

当 $T1$ 开始工作，并且计数值产生溢出时，由硬件使 $TF1=1$ ，在中断工作方式下向 CPU 请求中断，此标志一直保持到 CPU 响应中断后，才由硬件自动清 0。也可用软件查询该标志，并由软件清 0。如果 $T1$ 不工作，或者在工作但没有产生溢出，则 $TF1=0$ 。

TCON.5 TF0—— $T0$ 溢出中断标志。

其操作功能类似 $TF1$ 。

TCON.3 $\overline{IE1}$ —— $\overline{INT1}$ 外部中断 1 标志。

当硬件使 $\overline{IE1}=1$ ，表明外部中断 1 向 CPU 申请中断。

当 $\overline{IE1}=0$ ，表明外部中断 1 没有向 CPU 申请中断。

TCON.2 IT1——外部中断 1 触发方式控制位。

当 $IT1=0$ ，外部中断 1 设置为电平触发方式。在这种方式下，CPU 在每个机器周期的 $S5P2$ 期间对 $\overline{INT1}$ ($P3.3$) 引脚采样，若采到低电平，则认为有中断申请，随即使 $\overline{IE1}$ 标志=1；若为高电平，认为无中断申请或中断申请已撤除，随即清除 $\overline{IE1}$ 标志。在电平触发方式中，CPU 响应中断后不能自动清除 $\overline{IE1}$ 标志，也不能由软件清除 $\overline{IE1}$ 标志，所以在中断返回前必须撤销 $\overline{INT1}$ 引脚上的低电平，否则将会引起再次中断造成出错。

当 $IT1=1$ ，外部中断 1 设置为边沿触发方式。CPU 在每个机器周期的 $S5P2$ 期间采样引脚，若在连续两个机器周期采样到先高电平后低电平，则使 $\overline{IE1}$ 标志=1，此标志一直保持到 CPU 响应中断时，才由硬件自动清除。在边沿触发方式中，为保证 CPU 在两个机器周期内检测到先高后低的负跳变，输入高低电平的持续时间起码要保持 12 个时钟周期。

TCON.1 $\overline{IE0}$ —— $\overline{INT0}$ 外部中断 0 标志。

其操作功能与 $\overline{IE1}$ 相同。

TCON.0 IT0——外部中断 0 触发方式控制位。

其操作功能与 $IT1$ 相同。

AT89S52 增加的定时器 $T2$ 的中断标志在 $T2CON$ 中，详见 6.5 节。

(2) SCON 中的中断标志

SCON 是串行口控制寄存器，其低 2 位 TI 和 RI 锁存串行口的接收中断和发送中断标志。

SCON 中与中断有关的位如下。

SCON							99H	98H
(98H)							TI	RI

其每个位作用如下：

SCON.1 TI——串行发送中断标志。

当 TI=1，说明 CPU 将一字节数据写入发送缓冲器 SBUF，并且已发送完一个串行帧，此时，硬件使 TI 置位为 1。在中断工作模式下可以向 CPU 申请中断，在中断和查询工作模式下都不能自动清除 TI，必须由软件清除标志。

当 TI=0，说明没有进行串行发送，或者串行发送没有完。

SCON.0 RI——串行接收中断标志。

当 RI=1，在串行口允许接收后，每接收完一个串行帧，硬件使 RI 置位为 1。同样在中断和查询工作模式下都不会自动清除 RI，必须由软件清除标志。

当 RI=0，说明没有进行串行接收，或者串行接收没有完。

AT89S51 系统复位后，TCON 和 SCON 中各位均置为 0，应用中要注意各位的初始状态。

3. 中断优先级寄存器

80C51 单片机中断优先级的设定由专用寄存器 IP 统一管理，它具有两个中断优先级，由软件设置每个中断源为高优先级中断或低优先级中断，可实现两级中断嵌套。

高优先级中断源可中断正在执行的低优先级中断服务程序，除非在执行低优先级中断服务程序时设置了 CPU 关中断或禁止某些高优先级中断源的中断。同级或低优先级的中断源不能中断正在执行的中断服务程序。为此在 80C51 中断系统中，内部有两个（用户不能访问的）优先级状态触发器，它们分别指示出 CPU 是否在执行高优先级或低优先级中断服务程序，从而决定是否屏蔽所有的中断申请或同一级的其他中断申请。

专用寄存器 IP 为中断优先级寄存器，锁存各中断源优先级的控制位，用户可用软件设定。其格式如下。

IP			BDH	BCH	BBH	BAH	B9H	B8H
(B8H)	—	—	PT2 [*]	PS	PT1	PX1	PT0	PX0

其每个位作用如下：

IP.4 PS——串行口中断优先级控制位。

PS=1，设定串行口为高优先级中断；

PS=0，设定串行口为低优先级中断。

IP.3 PT1——T1 中断优先级控制位。

PT1=1，设定定时器 T1 为高优先级中断；

PT1=0，设定定时器 T1 为低优先级中断。

IP.2 PX1——外部中断 1 中断优先级控制位。

PX1=1，设定外部中断 1 为高优先级中断；

PX1=0，设定外部中断 1 为低优先级中断。

IP.1 PT0——T0 中断优先级控制位。

PT0=1, 设定定时器 T0 为高优先级中断;

PT0=0, 设定定时器 T0 为低优先级中断。

IP.0 PX0——外部中断 0 中断优先级控制位。

PX0=1, 设定外部中断 0 为高优先级中断;

PX0=0, 设定外部中断 0 为低优先级中断。

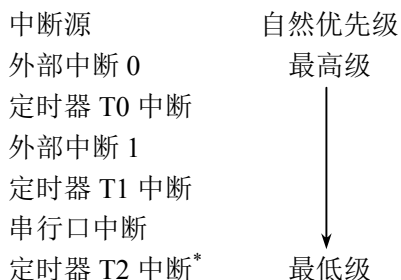
IP.5 PT2*——T2 中断优先级控制位 (仅 AT89S52/80C52 或类似型号单片机有)。

PT2=1, 设定定时器 T2 为高优先级中断;

PT2=0, 设定定时器 T2 为低优先级中断。

当系统复位后, IP 全部清 0, 将所有中断源设置为低优先级中断。

如果几个同一优先级的中断源, 同时向 CPU 申请中断, CPU 通过内部硬件查询逻辑按自然优先级顺序确定该响应哪个中断请求。其自然优先级由硬件形成, 排列如下:



这种排列顺序在实际应用中很方便, 且合理。如果重新设置了优先级, 则顺序查询逻辑电路将会相应改变排队顺序。

例如, 如果给 IP 中设置的优先级控制字为 09H, 则 PT1 和 PX0 均为高优先级中断, 但当这两个中断源同时发出中断申请时, CPU 将先响应自然优先级高的 PX0 的中断申请。

对于中断源大于 5 个的单片机型号, 其优先级顺序往下排, 例如 AT89S52 的 T2 级别低于串行口。

8.3 中断处理过程

中断处理过程可分为 3 个阶段, 即中断响应、中断处理和中断返回。所有单片机的中断处理都有这样 3 个阶段, 但不同的单片机由于中断系统的硬件结构不完全相同, 因而中断响应的方式也有所不同, 在此仅以 80C51 系列单片机为例来介绍中断处理的过程。

8.3.1 中断响应

中断响应是在满足 CPU 的中断响应条件之后, CPU 对中断源中断请求的回答, 在这一阶段, CPU 要完成执行中断服务以前的所有准备工作。这些准备工作包括保护断点和把程序转向中断服务程序的入口地址 (通常称矢量地址)。

单片机在运行时, 并不是任何时刻都会去响应中断请求, 而是在中断响应条件满足之后才会响应。

1. CPU 的中断响应条件

CPU 响应中断的条件主要有以下几点:

- ① 有中断源发出中断申请;
- ② 中断总允许位 EA=1, 即 CPU 允许所有中断源申请中断;
- ③ 申请中断的中断源的中断允许位为 1, 即此中断源可以向 CPU 申请中断。

以上是 CPU 响应中断的基本条件。若满足, CPU 一般会响应中断, 但如果有下列任何一种情况存在, 则中断响应会受到阻断。

- CPU 正在执行一个同级或高一级的中断服务程序;
- 当前的机器周期不是正在执行指令的最后一个周期, 即正在执行的指令完成前, 任何中断请求都得不到响应;
- 正在执行的指令是返回 (RETI) 指令或者对专用寄存器 IE、IP 进行读/写的指令, 此时, 在执行 RETI 或者读写 IE 或 IP 之后, 不会马上响应中断请求。至少再执行一条其他指令, 才会响应中断。

若存在上述任何一种情况, 都不会马上响应中断。此时将把该中断请求锁存在各自的中断标志位中, 然后在下一个机器周期再按顺序查询。

在每个机器周期的 S5P2 期间, CPU 对各中断源采样, 并设置相应的中断标志位。CPU 在下一个机器周期 S6 期间按优先级顺序查询各中断标志, 如查询到某个中断标志为 1, 将在再下一个机器周期 S1 期间按优先级进行中断处理。中断查询在每个机器周期中重复执行, 如果中断响应的基本条件已满足, 但如果存在上述 3 条之一而未被及时响应, 待上述封锁中断的条件被撤销之后, 由于中断标志还存在, 仍会响应。

2. 中断响应过程

如果中断响应条件满足, 且不存在中断阻断的情况, 则 CPU 将响应中断。

在 80C51 单片机的中断系统中有 2 个“优先级状态”触发器, 一个是“高优先级状态”触发器, 另一个是“低优先级状态”触发器, 这 2 个触发器是由硬件自动管理的, 用户不能对其编程。当 CPU 响应中断时, 它首先使优先级状态触发器置位, 这样可以阻断同级或低级的中断。然后, 中断系统自动把断点地址压入堆栈保护 (但不保护状态寄存器 PSW 及其他寄存器内容), 再由硬件执行一条长调用指令将对应的中断入口装入程序计数器 PC, 使程序转向该中断入口地址, 并执行中断服务程序。

8.3.2 中断处理

中断处理程序 (又称中断服务或中断子程序) 从入口地址开始执行, 直到返回指令“RETI”为止, 这个过程称为中断处理。此过程主要是处理中断源的请求, 但由于中断处理程序是由随机事件引起的实时响应, 因而使得它与一般的子程序有一定差别。

在编写中断服务程序时需注意以下几点:

① 注意保护现场和恢复现场, 因为一般主程序和中断服务程序都可能会用到累加器、PSW 寄存器及其他一些寄存器。CPU 在进入中断服务程序后, 用到上述寄存器时, 就会破坏它原来存在寄存器中的内容, 一旦中断返回, 将会造成主程序的混乱。因而在进入中断服务程序后, 一般要先保护现场, 然后再执行中断处理程序, 在返回主程序以前, 再恢复现场。对于要保护的内容一定要全面考虑, 不能遗漏。

② 在 CPU 响应中断, 使程序转向该中断入口地址后, 通常不能从此地址开始运行中断

服务程序，因为各入口地址之间，只相隔 8 字节，一般的中断服务程序是容纳不下的，因此最常用的方法是在中断入口地址单元处存放一条无条件转移指令，使程序跳转到用户安排的中断服务程序起始地址上去。这样可使中断服务程序灵活地安排在 64KB 程序存储器的任何空间。

③ 若要在执行当前中断程序时禁止更高优先级中断源中断，应先用软件关闭 CPU 中断，或屏蔽更高级中断源的中断，在中断返回后再开放中断。

④ 在保护现场和恢复现场时，为了不使现场数据受到破坏或者造成混乱，一般规定此时 CPU 不响应新的中断请求。这就要求在编写中断服务程序时，特别注意，在保护现场之前要关中断，在恢复现场之后开中断。如果在中断处理时允许有更高级的中断打断它，则在保护现场之后再开中断，恢复现场之前关中断。

8.3.3 中断返回

中断返回是指中断服务完成后，单片机返回到断点（即原来断开的位置），继续执行原来的程序。中断返回由专门的中断返回指令 RETI 实现，该指令的功能是把断点地址取出，送回到程序计数器 PC 中去。另外，它还通知中断系统已完成中断处理，将会清除优先级状态触发器，并且使部分中断源标志（除 TI、RI）清 0。

在中断服务程序中特别要注意不能用“RET”指令代替“RETI”指令。

8.3.4 中断请求的撤除

CPU 响应某中断请求后，在中断返回前，应该撤销该中断请求，否则会引起另一次中断。

对定时器 0 或 1 溢出中断，CPU 在响应中断后，利用硬件清除了有关的中断请求标志 TF0 或 TF1，即中断请求是自动撤除的，无须采取其他措施。

对于边沿触发的外部中断，CPU 在响应中断后，也是用硬件来自动清除有关的中断请求标志 IE0 或 IE1，无须采取其他措施。

对于串行口中断，CPU 响应中断后，没有用硬件清除 TI、RI，故这些中断标志不能自动撤除，而要靠软件来清除相应的标志。

以上中断的撤除都较简单，只有对于电平激活的外部中断，撤除方法较复杂。因为在电平触发方式中，CPU 响应中断时不会自动清除 IE1 或 IE0 标志，所以在响应中断后应立即撤除 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 引脚上的低电平。因为 CPU 对 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 引脚的信号不能控制，所以这个问题要通过硬件，再配合软件来解决。图 8-4 是撤除电平激活的中断可行方案之一。外部中断请求信号不直接加在 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 上，而是加在 D 触发器的 CLK 端。由于 D 端接地，当外部中断请求的正脉冲信号出现在 CLK 端时， $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 为低，发出中断请求。用 P1.0 接在触发器的 S 端作为应答线，当 CPU 响应中断后可用如下两条指令：

```
CLR    P1.0
SETB   P1.0
```

执行第一条指令使 P1.0 输出为 0，其持续时间为 2 个机器周期，足以使 D 触发器置位，

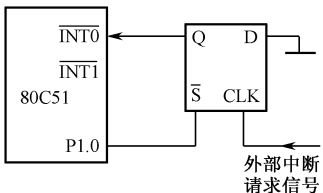


图 8-4 撤除电平激活的中断方案之一

从而撤除中断请求。第二条指令使 P1.0 变为 1，否则 D 触发器的 S 端始终有效， $\overline{\text{INT0}}$ 端始终为 1，无法再次申请中断。

除此以外，还可以采用其他方法。

8.3.5 中断响应时间

由上述可知，CPU 不是在任何情况下对中断请求都予以响应的。此外，不同的情况对中断响应的的时间也是不同的。下面以外部中断为例，说明中断响应的的时间。

在每个机器周期的 S5P2 期间， $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 端的电平被锁存到 TCON 的 IE0 和 IE1 位，CPU 在下一个机器周期才会查询这些值。这时如果满足中断响应条件，下面将由硬件自动执行一条调用指令“LCALL”，使程序转入中断矢量入口。调用本身要用 2 个机器周期，这样，从外部中断请求有效到开始执行中断服务程序的第一条指令，至少需要 3 个机器周期，这是最短的响应时间。

如果遇到中断受阻的情况，则中断响应时间会更长一些。例如，当一个同级或更高级的中断服务正在进行时，则附加的等待时间取决于正在进行的中断服务程序；如果正在执行的一条指令还没有进行到最后一个机器周期，附加的等待时间为 1~3 个机器周期（因为一条指令的最长执行时间为 4 个机器周期），如果正在执行的是 RETI 指令或者访问 IE 或 IP 的指令，则附加的等待时间在 5 个机器周期之内（为完成正在执行的指令，还需要 1 个周期，加上为完成下一条指令所需的最长时间——4 个周期，故最长为 5 个周期）。

若系统中只有一个中断源，则响应时间为 3~8 个机器周期。如果有 2 个以上中断源同时申请中断，则响应时间将更长。一般情况可不考虑响应时间，但在精确定时的场合需要考虑此问题。

8.3.6 扩充外中断源

AT89S51 单片机具有两个外部中断请求输入端 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ ，在实际应用中，若外部中断源超过两个，就需扩充外部中断源。这里介绍两种比较简单可行的方法。

1. 利用定时器扩展外中断源法

AT89S51 单片机有两个定时器，具有两个内部中断标志和外部计数引脚。将定时器设置成计数方式，计数初值设定为满量程，一旦从外部计数引脚输入一个下跳变信号，计数器加 1 产生溢出中断。把外部计数输入端 T0（P3.4）或 T1（P3.5）作为扩充中断源输入，把该定时器的溢出中断标志及服务程序作为扩充中断源的标志和服务程序。

例如，将定时器 T0 设定为方式 2（自动重装载常数）代替一个扩充外中断源，TH0 和 TL0 初值为 FFH，允许 T0 中断，CPU 开放中断，初始化程序如下：

```
MOV    TMOD,    # 06H
MOV    TL0,      # 0FFH
MOV    TH0,      # 0FFH
SETB   TR0
SETB   ET0
SETB   EA
```

当连接在 T0 (P3.4) 引脚的外部中断请求输入线发生负跳变时, TL0 计数加 1 产生溢出, 置位 TF0 标志, 向 CPU 发出中断申请, 同时 TH0 的内容 FFH 送到 TL0, 即 TL0 恢复初值。T0 引脚每输入一个负跳变信号, TF0 都会置 1 且向 CPU 请求中断, 这相当于边沿触发的外中断源输入了。

2. 中断和查询结合法

采用定时器的方法还是有一定局限性, 只能增加 2 路中断源, 如果要扩充更多的外中断源, 可以采用中断和查询结合的方法。图 8-5 所示就是一种扩充外中断源的实用方法。

图 8-5 中, 采用一个 4 与门电路扩充 4 个外中断源, 所有这些扩充的外中断源都是电平触发方式 (低电平有效)。当 4 个扩充中断源 XI1~XI4 中有一个或几个出现低电平, 与门输出为 0, 使 $\overline{\text{INT1}}$ 为低电平触发中断, 在外中断 1 服务程序中, 由软件按人为设定的顺序 (优先级) 查询外中断源哪位是高电平, 然后进入该中断处理。

在本例中, 各路输入的有效中断电平, 应该在 CPU 实际响应该中断源之前保持有效, 并且在该中断服务程序返回前取消。

$\overline{\text{INT1}}$ 的中断服务程序如下:

```
EXINT:  PUSH    PSW
        PUSH    ACC
        JNB     P1.0, AV1
        JNB     P1.1, AV2
        JNB     P1.2, AV3
        JNB     P1.3, AV4
DIB:    POP     ACC
        POP     PSW
        RETI

AV1:    ; XI1 中断服务程序
        :
        LJMP    DIB
AV2:    ; XI2 中断服务程序
        :
        LJMP    DIB
AV3:    ; XI3 中断服务程序
        :
        LJMP    DIB
AV4:    ; XI4 中断服务程序
        :
        LJMP    DIB
```

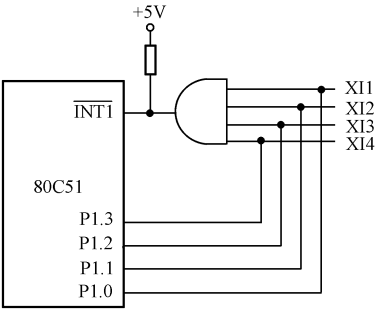


图 8-5 多外部中断源连接方法

8.4 中断程序的设计与应用

本节将介绍中断程序的一般设计方法, 并通过几个简明易懂的实例说明中断系统的应

用。通过这些实例，读者可以了解中断控制和中断服务程序的设计思想及设计时应注意的问题。

8.4.1 中断程序的一般设计方法

中断处理过程是一个和硬件、软件都有关的过程，因而它的编程方法有其特殊性。在图 8-6 的框图中把中断处理的硬件和软件过程进行了概括。图 8-6 (a) 是主程序中的中断初始化部分，图 8-6 (b) 是 CPU 响应中断后由硬件自动执行的过程，图 8-6 (c) 是中断服务程序框图。

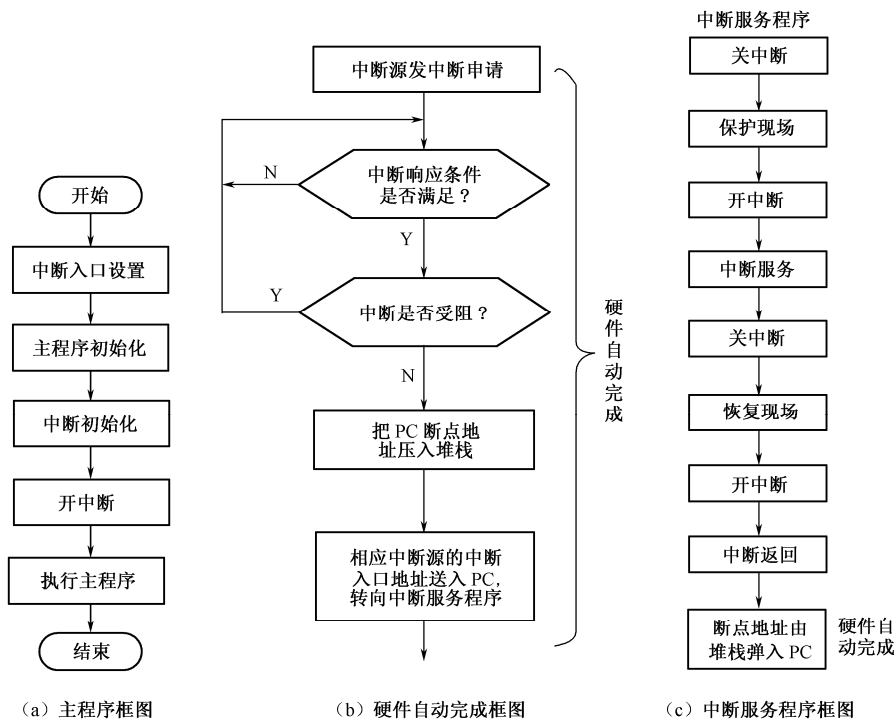


图 8-6 中断处理过程流程图

由图 8-6 可见与中断有关的程序一般包含 2 个部分，一部分是在主程序中的中断初始化部分，另一部分是中断响应后的处理程序。因为只有中断初始化，并且开放相关中断后，中断源的申请才可能得到响应，所以中断初始化一定要在中断源申请前设置。

1. 主程序中的中断初始化

在单片机复位后，与中断有关的寄存器都复位为 0，即都处于中断关闭状态。要实现中断功能，必须进行中断初始化设置。主程序中的中断初始化主要包括 2 个部分，首先是对 4 个与中断有关的特殊功能寄存器 TCON、SCON、IE 和 IP 的中断初始化，其次是对相关中断源的初始化。此外，在多数情况下还需要重新设置堆栈指针，因为系统复位后的指针为 07H，而 08H~1FH 区域为工作寄存器区，20H~2FH 为位寻址区，通常有可能用到它们，所以最好重新设置堆栈指针为 30H 以上。

对于与中断有关的特殊功能寄存器的相应位，按照要求进行状态预置后，CPU 就会按照

要求对中断源进行管理和控制。在 80C51 系列单片机中，管理和控制的项目有：

- ① CPU 开中断与关中断；
- ② 某中断源中断请求的允许和禁止（屏蔽）；
- ③ 各中断源优先级级别的设定（即中断源优先级排队）；
- ④ 外部中断请求的触发方式。

中断管理与控制程序一般不独立编写，而是包含在主程序中，根据需要通过几条指令来实现。例如 CPU 开中断，可用指令“SETB EA”或“ORL IE, #80H”来实现，关中断可用指令“CLR EA”或“ANL IE, #7FH”来实现。

对相关中断源的初始化也要在主程序中进行，例如定时器或串行口的初始化等。

图 8-6（a）为具有中断功能的主程序框图。现在假设应用程序中有 2 个中断源，一个是外部中断 0，一个是定时器 T1，则与中断初始化有关的程序一般编写格式如下：

```
ORG      0000H
LJMP     MAIN
ORG      0003H           ; 外部中断 0 入口地址
LJMP     SUB1            ; 转外部中断 0 服务程序入口地址
:
ORG      001BH           ; 定时器 T1 中断入口地址
LJMP     SUB4            ; 转定时器 T1 中断服务程序入口地址
:
MAIN:    ORG      0030H
:
:
MOV      TCON, #01       ; 外部中断 0 选择边沿触发方式
MOV      IE, #10001001B  ; CPU 开中断，外部中断 0 和定时器 T1 开中断
:
:                       ; 执行主程序
ORG      100H           ; 外部中断 0 中断服务程序入口地址
SUB1:    :
:                       ; 外部中断 0 的服务程序
:
RETI     :               ; 中断返回
:
ORG      200H           ; 定时器 T1 中断服务程序入口地址
SUB4 :    :
:                       ; 定时器 T1 的服务程序
RETI
```

当主程序初始化完成，并且开中断后，则硬件等待中断源申请中断，并自动完成响应中断和保护断点地址等工作，这个过程由图 8-6（b）说明。

在上面的示例中分别确定了外部中断 0 和定时器 T1 中断服务程序的入口地址，这样初学者容易明确中断程序的入口地址。但在使用这种方法时要特别注意中断服务程序所占的空间不能和主程序发生冲突。例如，主程序所占的空间为 30H~200H，此时把中断服务程序入口地址定为 100H，则会导致出错。在多数情况下采用的方法是不给中断程序入口地址定位，因为汇编程序能通过设置的标号自动给中断服务程序的入口地址定位。但使用这种方法时一定要注意不能把中断服务程序插在主程序中，一般是把中断服务程序安排在程序的最前面或最后面。

2. 中断服务程序

中断服务程序是一种具有特定功能的独立程序段。它为中断源的特定要求服务，以中断返回指令结束。图 8-6 (c) 为中断服务程序框图，图中的最后一个框是恢复断点地址，这个工作是硬件自动完成的。在中断响应过程中，断点的保护与恢复主要由硬件电路来实现。对用户来说，在编写中断服务程序时，主要考虑是否有需要保护的现场（指在主程序中要用到的寄存器、存储单元等在中断服务程序中也使用），如果有则应注意不要遗漏。在恢复现场时，要注意压栈与弹栈指令必须成对使用，先入栈的内容，应该后弹出。还要及时清除需用软件清除的中断标志。

保护现场之后的开中断是为了允许有更高一级中断打断此中断服务程序。如果不允许其他中断，则在中断服务程序执行过程中要一直关中断。

中断服务程序一般编写格式如下：

```
CH1:   CLR     EA      ; 关中断
        PUSH   ACC      ; 保护现场
        PUSH   PSW
        ⋮
        SETB   EA      ; 开中断（如果不希望高优先级中断进入，则不用开中断）
        ⋮              ; 中断处理程序
        CLR     EA      ; 关中断
        ⋮              ; 恢复现场
        POP     PSW
        POP     ACC
        SETB   EA
        RETI           ; 中断返回
```

对于只需要一次中断服务的程序，中断返回前可设关中断。

8.4.2 中断程序应用举例

下面通过具体实例说明中断控制和中断服务程序的设计。

例 1 利用定时器 T0 定时，在 P1.0 端输出一方波，方波周期为 20ms，已知晶振频率为 12MHz。

解：T0 的初值 $X = 65536 - 10000 = 55536 = D8F0H$ 。

汇编语言源程序如下：

```
        ORG     0000H
        LJMP    MAIN
        ⋮
T0 中断入口
        ORG     000BH
        LJMP    SUB1          ; 转 T0 中断服务程序入口
        ⋮
        ORG     30H
MAIN:    MOV     TMOD,  # 01H
        MOV     TL0,   # 0F0H      ; 置 10ms 定时初值
        MOV     TH0,   # 0D8H
```

```

MOV IE, # 82H ; CPU 开中断, T0 开中断
SETB TR0 ; 启动 T0
HERE: SJMP HERE ; 循环等待定时到
    ⋮
SUB1: MOV TL0, # 0F0H ; 重赋初值
      MOV TH0, # 0D8H
      CPL P1.0 ; 输出取反
      RETI
    ⋮

```

C51 语言程序如下:

```

#include <REG52.H>
sbit OUT = P1^0;
void main (void)
{
    TMOD = 0x01; //设置定时器工作方式
    TL0 = 0xf0;
    TH0 = 0xd8;
    IE = 0x82; //开中断
    TR0 = 1; //启动定时器 T0
    while(1); //主程序等待
}
void T0_int(void) interrupt 1 //定义 T0 中断服务程序
{
    TL0 = 0xf0; //定时器赋初值
    TH0 = 0xd8;
    OUT = !OUT; //输出反相
}

```

在本例的中断服务程序中没有关中断, 也没有保护现场, 因为只有一个中断源, 且主程序中没有需要保护的内容。

在本程序中没有用 CLR TF0 指令, 因为进入中断服务程序后, 硬件可自动将 TF0 清 0。采用中断方式后程序可以完成更多的工作, 例如本题中的 SJMP 指令, 要反复运行 10ms, 在这期间可以用来执行许多其他操作, 不必专门等待定时时间到。

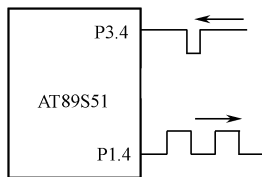


图 8-7 例 2 示意图

例 2 有一个由 AT89S51 单片机组成的计数和方波输出系统, 单片机采用 12MHz 晶振。如图 8-7 所示, 外部输入的脉冲接至 P3.4 脚, 由 T0 口进行计数。要求每计满 5 个脉冲, P1.4 口输出方波的周期在 20ms 和 40ms 两种情况进行切换。

解: 把定时器 0 设置为工作方式 1 下的外部脉冲计数方式, 定时器 T1 设置为工作方式 1 下的定时方式。

T0 的初值 $X=65536-5=65531=FFFBH$

由 P1.4 输出 20ms 方波时, 每隔 10ms 使 P1.4 的电平变化一次, 则 T1 的初值为 D8F0H; P1.4 输出 40ms 方波时, 每隔 20ms 使 P1.4 的电平变化一次, 则 T1 的初值为 B1E0。

汇编语言源程序如下:

```

ORG 0000H
LJMP MAIN
...

```

T0 中断入口

```
ORG 000BH ; 转 T0 中断服务程序入口
LJMP SUB1
```

...

T1 中断入口

```
ORG 001BH ; 转 T1 中断服务程序入口
LJMP SUB2
```

...

```
MAIN: ORG 0030H
      MOV TL0, #0FBH ; T0 赋计数初值
      MOV TH0, #0FFH
      MOV TL1, #0F0H ; T1 赋定时初值, 20ms 周期的方波
      MOV TH1, #0D8H
      MOV TMOD, #00010101B ; T1 为方式 1 定时, T0 为方式 1 计数
      MOV TCON, #01010000B ; 启动 T0,T1 工作
      MOV IE, #10001010B ; 使能 T0,T1,和总中断
      SJMP $ ; 循环等待

SUB1: ORG 100H
      PUSH ACC
      MOV TL0, #0FBH ; 重新赋初值
      MOV TH0, #0FFH
      CPL F0 ; 标志位取反
      POP ACC
      RETI

SUB2: ORG 200H
      JB F0,L1 ; 标志位为 1, 则跳转到 L1
      MOV TL1, #0F0H ; 赋 20ms 方波的初值
      MOV TH1, #0D8H
      JMP L2
L1:   MOV TL1, #0E0H ; 赋 40ms 方波的初值
      MOV TH1, #0B1H
L2:   CPL P1.4 ; P1.4 输出取反, 形成方波
      RETI
```

C 语言程序如下:

```
#include<reg51.h>
sbit WAVE=P1^4; //定义方波输出脚
bit flag=0;
main()
{
    TH0=0xFF; //T0,T1 初始化
    TL0=0xFB;
    TH1=0xD8;
    TL1=0xF0;
    TMOD=0x15;
    TCON=0x50;
    ET0=1;
    ET1=1;
```

```

    EA=1;
    while(1) { }
}
void T0_int() interrupt 1 //T0 计数器中断服务程序
{
    TH0=0xFF; //重新赋初值，并取反标志
    TL0=0xFB;
    flag=!flag;
}
void T1_int() interrupt 3 //T1 定时器中断服务程序
{
    if(flag==0) //标志为 0 则赋 20ms 方波初值
    { TH1=0xDB;
      TL1=0xF0;
    }
    else //标志为 1 则赋 40ms 方波初值
    { TH1=0xB1;
      TL1=0xE0; }
    WAVE=!WAVE; // P1.4 输出取反，形成方波
}

```

由以上程序可知，定时器仅在初始化和计满溢出产生中断时，才占用 CPU 工作时间，一旦启动之后，定时器的定时、计数过程全部是独立运行的，因而采用中断后使 CPU 有较高的工作效率。

例 3 要求在甲、乙两台 AT89S51 单片机之间进行串行通信，甲机发送，乙机接收。本例题实现的功能是把甲机片内以 50H 为数据起始地址的 16 字节数据发送到乙机。甲机首先发送数据长度，然后再开始发送数据。乙机以接收到的第 1 字节作为接收数据的长度，第 2 字节开始为数据。乙机将接收的数据存放在片内以 60H 为数据起始地址的 16 个单元内。已知两机所使用的晶振均为 11.0592MHz，传输波特率定为 9600。

解：甲机串行发送的内容包括数据块的长度和数据。在本程序中数据长度是在主程序中发送的，甲机启动运行后即开始发送数据长度，此时串行口是关中断的，当数据长度发出后，再开中断。发送数据是在中断服务程序中完成的。本例中，乙机必须先启动运行做好接收数据的准备，甲机每发送一个数据至乙机，都使乙机 RI 置“1”。因为乙机串行口是开中断的，因而它能够响应中断，转至中断服务程序处理传来的数据。在甲、乙机等待发送的时候还可以使 CPU 执行一些其他功能。

设甲、乙机的定时器 1 按方式 2 工作，串行口按方式 1 工作。

在乙机接收程序中需要设置一个数据长度和数据的识别位，在此用 F0 表示，当 F0 为 1 时表示接收的是数据，为 0 时为数据长度。

甲机有关发送的汇编语言程序如下：

```

ORG    0000H
LJMP   MAIN
ORG    0023H
LJMP   ESS
:

```

```

MAIN:
    ⋮
    MOV    TMOD,    # 20H        ; 定时器 1 设置为方式 2
    MOV    TL1,     # 0FDH       ; 定时器 1 赋初值
    MOV    TH1,     # 0FDH
    SETB   EA        ; CPU 开中断
    CLR    ES        ; 串行口关中断
    SETB   TR1       ; 启动定时器 1 工作
    CLR    TI        ; 清发送中断标志
    MOV    SCON,    # 40H       ; 串行口置工作方式 1
    MOV    08H,     #50H       ; 发送数据起始地址→第 1 组寄存器的 R0
    MOV    09H,     #16        ; 数据长度→第 1 组寄存器的 R1
    MOV    A,       #16
    MOV    SBUF,    A          ; 输出数据长度
    SETB   ES        ; 串行口开中断
    SJMP   $          ; 等待发送
    ⋮
    ORG    100H        ; 串行口中断服务程序
ESS:   PUSH   ACC      ; 把 A 压栈保护
    SETB   RS0        ; 保护第 0 组工作寄存器
    CLR    RS1        ; 选择第 1 组工作寄存器
    MOV    A,         @ R0    ; 发送数据→A
    CLR    TI        ; TI 清 0
    MOV    SBUF,     A        ; 输出数据
    INC    R0
    DJNZ   R1,       L1      ; 数据未送完转至 L1
    CLR    ES        ; 串行口关中断
L1:    POP    ACC      ; 弹栈恢复现场
    CLR    RS0        ; 恢复第 0 组工作寄存器
    RETI

```

C51 语言程序如下:

```

#include <REG52.H>                //包含 52 单片机 SFR 的定义
static unsigned char number = 0;  //定义一个静态变量
data unsigned char send_data[16] _at_ 0x50; //定义发送数据数组
void main (void)
{
    TMOD = 0x20;                  //设置定时器 1 定时方式
    TL1 = 0xfd;                   //定时器 1 赋初值
    TH1 = 0xfd;
    EA = 1;                       //开中断
    ES = 0;                       //关串行中断
    TR1 = 1;                      //启动定时器 1
    TI = 0;                       //清串行口发送标志
    SCON = 0x40;                  //串行口设置
    SBUF = 16;                    //发送数据的个数
    ES = 1;                       //串行口开中断
    while(1);
}

```

void UART_int(void) interrupt 4 using 1	//串口中断服务程序
{	
if(TI)	//判断数据是否发送完成
{	
TI = 0;	//清发送标志
if (number<16)	//数据未发送完，则继续发
{ SBUF = send_data[number] ;	//发送一个数据
number++;	//计数值加 1
}else	
{ ES = 0;	//禁止串行口中断
TR1 = 0;	//定时器 1 停止工作
}	
} else RI = 0;	//接收标志位清零
}	

乙机有关接收的汇编语言程序部分:

	ORG	0000H	
	LJMP	MAIN	
	ORG	0023H	
	LJMP	ESS	
	:		
MAIN:	:		
	MOV	TMOD, # 20H	; 定时器 1 设置为方式 2
	MOV	TL1, # 0FDH	; 定时器 1 赋初值
	MOV	TH1, # 0FDH	
	SETB	EA	; CPU 开中断
	SETB	ES	; 串行口开中断
	SETB	TR1	; 启动定时器 1 工作
	MOV	SCON, # 50H	; 置串行口方式 1 接收
	CLR	F0	; F0 为 0, 表示数据长度
	MOV	08H, #60H	; 接收数据首地址→第 1 组寄存器的 R0
	SJMP	\$; 等待接收
	:		
	ORG	100H	; 串行口中断服务程序
ESS:	SETB	RS0	; 保护第 0 组工作寄存器
	CLR	RS1	; 选择第 1 组工作寄存器
	PUSH	ACC	; A 压栈保护
	JB	F0, L1	
	MOV	A, SBUF	; 接收数据长度信息
	CLR	RI	; RI 清 0
	MOV	R1, A	; 数据长度→R1
	SETB	F0	; 置为接收数据标志
	SJMP	L2	
L1:	MOV	A, SBUF	; 接收数据信息
	MOV	@ R0, A	; 存放数据
	CLR	RI	; RI 清 0
	INC	R0	
	DJNZ	R1, L2	; 数据未接收完转至 L2

```

                CLR    ES                      ; 串行口关中断
                CLR    TR1                    ; 关定时器 1
L2:            POP     ACC                    ; 弹栈恢复现场
                CLR    RS0                    ; 恢复第 0 组工作寄存器
                RETI
            END

```

C51 语言程序如下：

```

#include <REG52.H>
data unsigned char receive_data[16] _at_ 0x60; //定义接收数据数组
static unsigned char i = 0, number = 0;       //定义两个静态变量
unsigned char flag;                            //定义一个变量
void main (void)
{
    TMOD = 0x20;                               //设置定时器 1 定时方式
    TL1 = 0xfd;                                //计数器计数初值
    TH1 = 0xfd;
    EA = 1;                                    //开中断
    ES = 1;                                    //串行口开中断
    TR1 = 1;                                   //启动定时器 1
    SCON = 0x50;                               //设置串行口允许接收
    flag=0;
    while(1);
}
void UART_int(void) interrupt 4 using 1        //串行口中断服务程序
{
    if(RI)                                     //判断是否收到数据
    {
        RI = 0;                               //清接收标志位
        if(flag)
        { receive_data[i] = SBUF;              //接收数据并存储
          i++;                                 //计数器加 1
          if (i>=number)                       //判断接收数据个数
          { ES = 0;                            //关串行中断
            TR1 = 0;                           //定时器 1 停止工作
          }
        }else
        {                                       //接收数据长度信息
          number = SBUF;
          flag = 1;                            //标志位置 1，表示接下来收数据
        }
        } else TI = 0;                         //发送标志清零
    }
}

```

显然，采用中断程序的方法省略了等待发送和接收的指令，提高了程序执行效率。在中断程序中清除 TI、RI 的作用是当下一次发送和接收完成之后，又可以自动进入中断程序。

在本例中甲机只负责发送，它不管乙机是否已接收到数据或接收正确与否，这样通信是不太可靠的，通常采用的方法是甲机先呼叫乙机，乙机应答并同意接收时，甲机再开始发送，运行时乙机应先于甲机打开。

当需在 2 台以上的单片机间传输数据时，常采用多机通信的办法。多机通信过程已在 7.5 节中叙述，多机通信的应用可参考文献 2。

思考与练习

1. 什么是中断？在单片机中中断能实现哪些功能？
2. 什么是中断优先级？中断优先级处理的原则是什么？
3. AT89S51 有几个中断源，各中断标志是如何产生的，又如何清 0 的？CPU 响应中断时，中断入口地址各是多少？
4. 中断响应时间是否确定不变的？为什么？响应中断的条件是什么？
5. 用定时器 T1 定时，要求在 P1.6 口输出一个方波，周期为 1min。晶振为 12MHz，请用中断方式实现，并分析采用中断后的优点。
6. 中断响应过程中，为什么通常要保护现场？如何保护？
7. 试用中断方法，设计秒、分脉冲发生器。
8. AT89S51 单片机的中断系统中有几个优先级？如何设定？若扩充 8 个中断源，如何确定优先级？
9. 试用中断技术设计一个秒闪电路，其功能是发光二极管 LED 每秒闪亮 400ms。主机频为 24MHz。
10. 试设计一个 AT89S51 单片机的双机通信系统，并编写程序将 A 机片内 RAM 40H~5FH 的数据块通过串行口传送到 B 机的片内 RAM 60H~7FH 中去。已知单片机晶振为 12MHz，要求采用中断方式接收，传送时进行奇偶校验；若出错，则置 F0 标志为 1，波特率为 1200。
11. 试将 4.2 节的例 10 改为用中断方法实现延时。

第9章 单片机的系统扩展

系统扩展是指当单片机内部的存储器、I/O 口、片上外设等不能满足应用系统要求时，在片外连接相应的外围芯片，对单片机进行功能扩展以满足应用要求。通常情况下，应该尽量选择内部资源可满足要求的单片机形成应用系统，这样最能体现单片机体积小、成本低的优点。由于控制对象的多样性和复杂性，有些情况下使得单片机内部的资源不能满足应用系统的要求，此时需要对单片机系统进行扩展。从传输方式上分，单片机的系统扩展方法主要有并行扩展和串行扩展两种。

9.1 并行扩展概述

并行扩展是利用单片机的三总线（地址、数据和控制）进行系统扩展，这种方法传输速度快，但占用引脚数较多，布线较复杂。80C51 系列单片机很适宜进行外部并行扩展，其扩展电路及扩展方法较典型、规范，外围扩展电路芯片大多是一些常规芯片。用户比较容易通过标准扩展电路来构成较大规模的应用系统。

9.1.1 系统扩展常用接口芯片

根据不同的用途，系统扩展所涉及的集成电路芯片很多。在选择芯片时要注意由于制造工艺不同，具有相同逻辑功能的集成芯片的速度、输入/输出电压、功耗等性能均不完全相同，为进行区别，其型号有多种表达方式，如 74LS××、74HC××、74S××、74F××等。一般情况下具有相同逻辑功能的芯片是能互换的，但在一些特殊要求的场合，需要注意速度、电压、功耗等的匹配，否则可能使工作不正常，具体使用方法要查看器件手册。对于目前 5V 供电单片机系统多采用 74HC××系列。在此主要介绍几种在系统扩展中常用的数字电路芯片，如锁存器、译码器等的使用方法。

1. 锁存器

在地址/数据线复用的单片机中，需要用锁存器锁存先出现的地址信号。锁存器有几种，不同的锁存器与单片机的连接方法不完全相同。本节介绍使用最多的 74HC373 锁存器。

74HC373 也可简称 74373，其芯片引脚图见附录 E，图 9-1 所示为其常用连接方法。图中 \overline{OE} 为使能控制端。当 \overline{OE} 为低电平时，8 路全导通；当 \overline{OE} 为 1 时，输出为高阻态。G 为锁存控制信号。

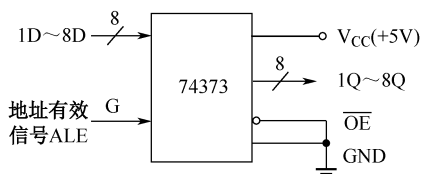


图 9-1 74373 常用连接方法

74373 有 3 种工作状态：

（1）当 \overline{OE} 为低电平、G（有的手册称为 LE）为高电平时，输出端状态和输入端状态相同，即输出跟随输入。

(2) 当 \overline{OE} 为低电平、 G 由高电平降为低电平（下降沿）时，输入端数据锁入内部寄存器中，内部寄存器的数据与输出端相同，当 G 保持为低电平时，即使输入端数据变化，也不会影响输出端状态，从而实现了锁存功能。

(3) 当 \overline{OE} 为高电平时，锁存器缓冲三态门封闭，即三态门输出为高阻态。74373 的输入端 $1D\sim 8D$ 与输出端 $1Q\sim 8Q$ 隔离，则不能输出。

当 74373 用作单片机低 8 位地址/数据线地址锁存器时，将 \overline{OE} 置成低电平，锁存允许信号 G 受控于单片机地址有效锁存信号 ALE 。这样当外部地址锁存有效信号 ALE 使 G 变为高电平时，74373 内部寄存器便处于直通状态，当 G 下降为低电平时，立即将锁存器的输入 $1D\sim 8D$ 即总线上的低 8 位地址锁入内部寄存器中。

2. 8 线同相三态数据缓冲/驱动器 74HC244

单片机在进行系统扩展时，必须解决总线的隔离和驱动问题。通常总线上连接着多个数据源设备和多个数据负载设备。但是在任何时刻，只能进行一个源和一个负载之间的数据传送，此时要求所有其他设备在电性能上与总线隔离。使外设需要的时候与总线接通，不需要的时候又能和总线隔离开，这就是总线隔离问题。此外，由于单片机每个 I/O 管脚的驱动能力亦有限。因此，为了驱动负载，往往采用缓冲/驱动器，74HC244 就具有数据缓冲隔离和驱动作用，74HC244 的最大吸收电流为 24mA，因而采用它们可加强数据总线的驱动能力。芯片引脚图见附录 E。

74HC244 使用时可分为两组，每组 4 条输入线 ($A_1\sim A_4$)，4 条输出线 ($Y_1\sim Y_4$)。 $1\overline{G}$ 和 $2\overline{G}$ 分别为每组的三态门使能端，低电平有效。一般应用是将 244 作为 8 线并行输入/输出接口器件。

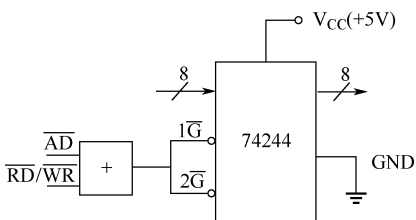


图 9-2 74244 读、写操作原理图

如果 74HC244 在系统中并不始终处于门通状态，而是在需要读或写数据时才打开缓冲门，则需采用地址编码线配合进行读或写操作，其原理如图 9-2 所示，图中 \overline{AD} 为 244 芯片在系统中的地址编码线。 \overline{AD} 信号线低电平有效； \overline{RD} 或 \overline{WR} 为系统 CPU 读或写控制信号。只有 \overline{AD} 和 \overline{RD} 或 \overline{AD} 和 \overline{WR} 同时为低电平，系统选择该芯片并且处在读或写周期时，数据才能通过 244 输入和输出。一旦有 1 个控制信号为高电平，缓冲门则为高阻态，

使输入或输出设备与系统数据总线隔离开来。

与 74244 电路功能类似的还有 8 反相缓冲/驱动器 74240 等。74240 管脚与 74244 完全兼容，只是输出信号反相。

3. 8 线总线接受/发送器 74HC245

74245 与 74244 不同之处是前者可以双向输入/输出，其工作连线如图 9-3 所示。

图 9-3 中，当 \overline{E} 端有效（低电平）时，74245 的输入/输出方向由 \overline{DIR} 端控制，使其根据需要变为高电平或低电平。在单片机系统中，可采用读信号或写信号实现控制，当 \overline{WR} 有效时，数据通过 74245 的 B 端 ($B_1\sim B_8$) 输入，由 A 端 ($A_1\sim$

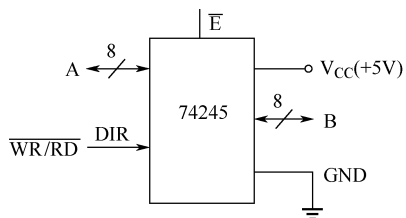


图 9-3 用读、写信号控制 74245 传输方向

A₈) 输出；当 \overline{RD} 有效时，数据由 A 端输入，B 端输出。由此可见，由于 74245 芯片具有双向缓冲和驱动作用，很适合作单片机数据总线的收发器。

4. 译码器

译码器有变量译码器、代码译码器和显示器译码器 3 类，在此仅介绍用作地址译码的变量译码器。常用译码器有 74HC138 和 74HC139 等。

74HC138 是“3-8”译码器，具有 3 个选择输入端，可组合成 8 种输入状态，输出端有 8 个，每个输出端分别对应 8 种输入状态中的 1 种，0 电平有效。即对应每种输入状态，仅允许 1 个输出端为 0 电平，其余全为 1。74138 还有 3 个使能端 E₃、 $\overline{E_2}$ 和 $\overline{E_1}$ ，必须同时输入有效电平，译码器才能工作，也就是仅当输入电平为 100 时，才选通译码器，否则译码器的输出全无效。其引脚图和逻辑功能真值表见图 9-4 (a) 和 (b)。显然采用译码器寻址可节约单片机的 I/O 口线。

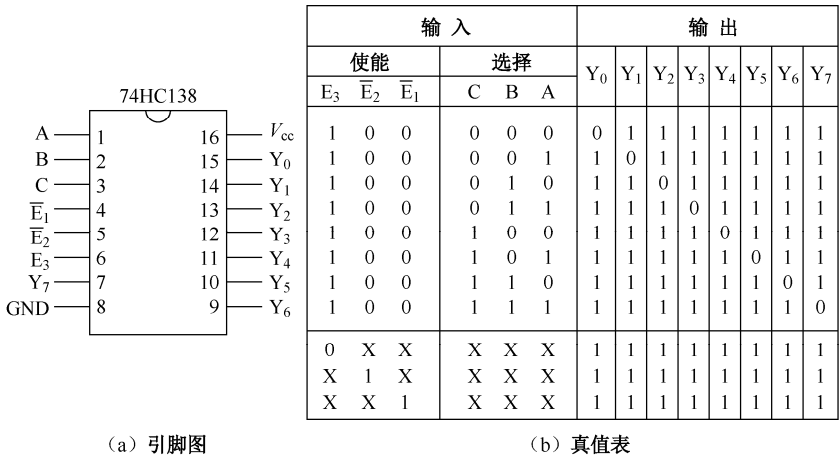


图 9-4 译码器 74138 的引脚图及真值表

在单片机进行系统扩展时，数据缓冲器、锁存器应用是很普遍的，限于篇幅这里仅介绍了以上几种。

9.1.2 外部并行扩展总线

80C51 系列单片机在进行系统并行扩展时，需要依靠地址总线、数据总线和控制总线将外部芯片与单片机连接为一体。一般的计算机外部三总线是相互独立的，但 80C51 系列单片机由于受引脚的限制，其作为低 8 位地址线的 P0 口是地址/数据复用口。为了区别地址/数据信号，需要在 P0 口外部加 1 个地址锁存器，从而形成一个与一般计算机类似的外部扩展三总线。其三总线扩展电路原理如图 9-5 所示。该图中单片机芯片为 80C51 系列中的一种，地址锁存器可以采用 74HC373。

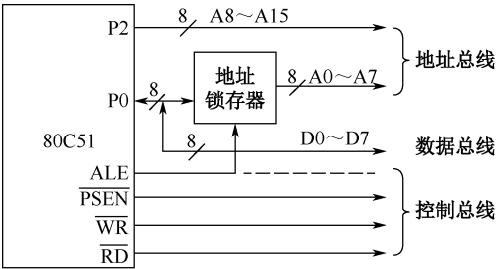


图 9-5 单片机的外部扩展三总线

由图 9-5 可知, P2 口为地址线的高 8 位, P0 口的低 8 位地址信号首先送到地址锁存器 74HC373 中。当 ALE 信号由高变低时, 此地址被锁存到 74HC373 中, 直到 ALE 信号再次变高, 低 8 位地址才会发生变化。此时, P2 口与被锁存的 P0 口的低 8 位地址共同形成了 16 位的地址总线, 寻址范围为 64KB。作为高 8 位的 P2 口在应用中可根据实际寻址范围来确定采用几根口线, 并不一定要把 8 位口全部接上。此外, 由于一些外围接口芯片的地址也在这 64KB 寻址范围之内, 选择外围芯片地址时, 须注意不要与存储器地址发生冲突, 还要保证存储器的地址是连续的。

P0 口作为地址线使用时是单向的, P0 口作为数据线使用时是双向的。P0 口的数据/地址复用功能是通过软件、硬件配合共同实现的。在第 2 章中介绍的 P0 口多路转换电路 MUX 及地址/数据控制电路就是为此而设计的。因为 P0 口是分时提供低 8 位地址和数据信息的, 所以在软件上通过采用访问片外存储器的指令, 就可以实现在送出低 8 位地址信号和锁存信号之后, 接着送出数据信息。图 9-6 所示的时序图可以帮助读者更好地理解这个问题。

单片机系统扩展所用到的控制线主要有如下几根:

- ALE 作为低 8 位地址锁存的选通信号;
- $\overline{\text{PSEN}}$ 作为扩展程序存储器的读选通信号;
- $\overline{\text{RD}}$, $\overline{\text{WR}}$ 作为扩展数据存储器 and 外接 I/O 口芯片的读、写选通信号。

由上述分析可知, 当进行并行总线的系统扩展时, 需要占用较多的 I/O 口线。

9.1.3 并行扩展的寻址方法

系统并行扩展的寻址是指当单片机扩展了存储器、I/O 接口等外围接口芯片之后, 如何寻找这些芯片的地址。外围接口芯片的寻址与存储器寻址方法类似, 一般更简单些。下面重点介绍存储器的寻址。存储器寻址是通过对地址线进行适当连接, 以使得存储器中任一单元都对应唯一的地址。存储器寻址分两步, 即存储器芯片的寻址和芯片内部存储单元的寻址。在存储器寻址问题中, 对于芯片内部存储单元的选择, 方法很简单, 就是把存储器芯片的地址线和相应的系统地址线按位相连即可; 但芯片的寻址方法有多种, 存储器寻址主要是研究芯片的寻址问题。目前, 常用的方法有两种: 线选法和译码法。

(1) 线选法寻址

当扩展存储器采用的芯片不多时, 比较简单的一种方法就是采用线选法寻址。

线选法直接以系统的几根高位地址线作为芯片的片选信号, 把选定的地址线和存储器芯片的片选端直接相连即可。线选法的特点是连接简单, 不必专门设计逻辑电路, 只是芯片占用的存储空间不紧凑, 并且地址空间利用率低, 一般用于简单的系统扩展。

(2) 译码法寻址

当扩展存储器或其他外围芯片的数量较多时, 常采用译码法寻址。译码法寻址由译码器组成译码电路对系统的高位地址进行译码, 译码电路将地址空间划分为若干块, 其输出作为存储器芯片的片选信号分别选通各芯片。这样既充分利用了存储空间, 又克服了空间分散的缺点, 还可减少 I/O 口线。这种方法也适用于其他外围电路芯片。

下面以 74HC138 为例说明译码器的用法。

如果把单片机的地址线 A13、A14、A15 分别与 74HC138 的 A、B、C 端相接, 则输出端产生 8 个片选信号 Y0~Y7, 可选择的寻址范围依次为 0000H~1FFFH, 2000H~3FFFH,

4000H~5FFFH, ..., E000H~FFFFH, 显然其寻址空间是连续的, 这样做也节约了单片机的 I/O 口线。当单片机外扩的并行芯片较多时, 适宜采用此方法。

9.2 存储器的并行扩展

当单片机片内存储器不够用或采用片内无存储器的芯片时, 需要扩展程序存储器或数据存储器, 扩展容量随应用系统的需要而定。由于单片机技术的进步, 目前单片机片内程序存储器的容量基本都能满足要求, 因而一般不必再扩展程序存储器, 本节仅介绍单片机采用并行总线扩展数据存储器的方法, 此方法也适用于具有并行 I/O 接口的其他芯片。

9.2.1 数据存储器扩展概述

在 80C51 扩展系统中, 数据存储器由随机存取存储器组成, 最大可扩展 64KB。一般采用静态 RAM, 数据读写的访问时间根据不同型号一般为几十到几百纳秒。

数据存储器地址空间同程序存储器一样, 访问时由 P2 口提供高 8 位地址, P0 口分时提供低 8 位地址和 8 位双向数据。数据存储器的读和写由 \overline{RD} (P3.7) 和 \overline{WR} (P3.6) 信号控制, 而程序存储器由读选通信号 \overline{PSEN} 控制, 两者虽然共处同一地址空间, 但由于控制信号不同, 故不会发生总线冲突。

访问片外扩展数据存储器和片外其他外围芯片可用下面 4 条寄存器间址指令:

```
MOVX  A,      @Ri
MOVX  A,      @DPTR
MOVX  @Ri,    A
MOVX  @DPTR,  A
```

在 80C51 系列单片机中, 可以用做数据存储器的芯片主要是静态数据存储器、动态数据存储器 and 可改写的只读存储器。常用芯片 62128 为 16K×8 位 RAM, 62256 为 32K×8 位 RAM, 62512 为 64K×8 位 RAM 等。可改写的只读存储器目前使用最多的是闪存类型的芯片, 如 AT29C256、AT29C512 等。

9.2.2 访问片外数据存储器的操作时序

访问片外数据存储器的操作包括读写两种操作时序, 通过对操作时序的了解, 可以更好地理解 ALE、 \overline{RD} 、 \overline{WR} 、P0 及 P2 等信号和数据线的作用, 及 P0 口是如何分时控制低 8 位地址和数据传输的。

现在以 MOVX @DPTR,A 和 MOVX A,@DPTR 指令为例, 说明访问片外数据存储器的操作时序, 见图 9-6。MOVX@Ri,A 和 MOVX A,@Ri 指令的时序与其相似, 在此省略。片外 RAM 的读写指令均为 2 个机器周期, 第 1 个机器周期为从程序存储器取指令周期, 第 2 个机器周期为向片外数据存储器读写数据。

图 9-6 (a) 为读片外 RAM 的操作时序, 在第 1 个机器周期的 S1 状态时, 开始读周期。在 S2 状态时, CPU 首先读入低 8 位指令地址 (PCL 即 A0~A7), 几乎与此同时读入程序存储器的高 8 位指令地址 (PCH 即 A8~A15), 接着是读指令, 在从片内程序存储器取指令时, 与片外相接的 P0、P2 和 ALE 不起作用, 指令是通过内部总线传输的。在第 1 个机器周期的

S4 状态之后，把片外数据存储器（也可以其他外设）地址的低 8 位（DPL）送到 P0 总线，在 ALE 由高变低时，把低 8 位地址（DPL）锁存到外部的地址锁存器中，同时把高 8 位地址（DPH）送到 P2 总线，而 P2 口的高 8 位地址信号保持不变，可以不用外部锁存器，此时选通要寻址的片外 RAM 单元。在第 2 个机器周期中，读控制信号 \overline{RD} 有效，在 S1~S2 状态，减少一个 ALE，经过适当延时后，被寻址的片外 RAM 单元中的数据送到 P0 总线。在 \overline{RD} 有效时 CPU 将数据读入累加器 A。注意，在任何情况下低 8 位地址与 8 位数据都是分时使用 P0 口的。当 \overline{RD} 变为高电平后，被寻址的片外 RAM 的总线驱动器变为悬浮状态，使 P0 总线驱动器又进入高阻状态。

图 9-6（b）为写片外 RAM 的操作时序，第 1 个机器周期是读取指令，操作过程与读片外 RAM 的操作时序类似，同样，在读指令后，CPU 把低 8 位地址（DPL）送到 P0 总线，在 ALE 由高变低时，把低 8 位地址（DPL）锁存到外部的地址锁存器中，同时把高 8 位地址（DPH）送到 P2 总线，此时选通被寻址的片外 RAM 单元，在第 2 个机器周期写控制信号 \overline{WR} 有效，在 S1~S2 状态，也减少一个 ALE，经过适当延时后，P0 口将送出累加器 A 的数据，在 \overline{WR} 有效时 CPU 将数据写入被寻址的片外 RAM 单元。

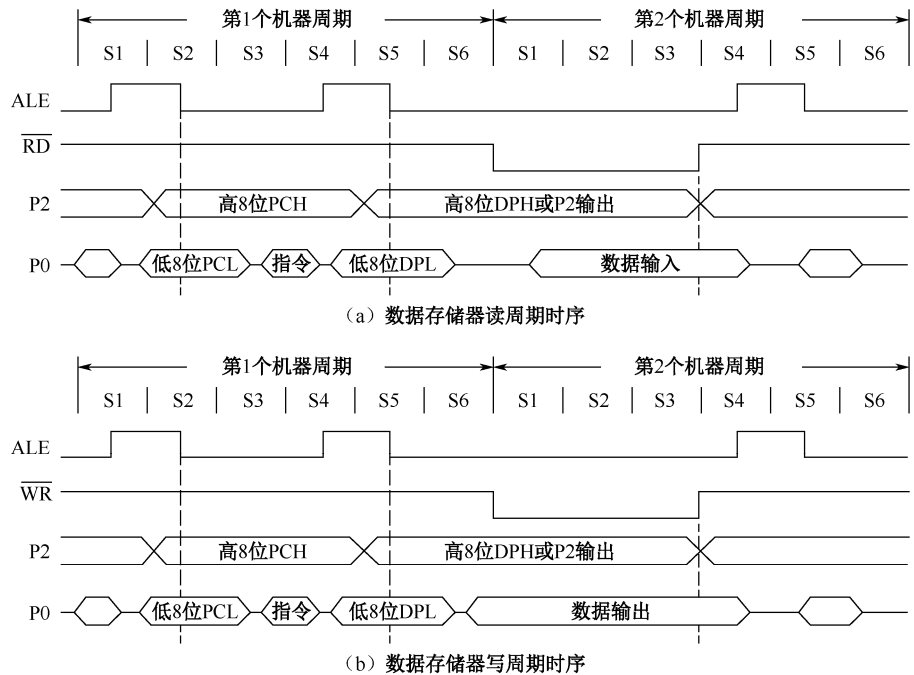


图 9-6 访问片外数据存储器的操作时序

由于在对片外存储器进行读写操作时都会减少一个 ALE，所以在这种情况不能把 ALE 信号作为一个频率不变的输出信号源使用。

9.2.3 数据存储器扩展举例

图 9-7 是扩展 32KB（采用 1 片 62256RAM）数据存储器的实例。

图 9-7 中 8D 锁存器 74HC373 的三态控制端 \overline{OE} 接地，以保持输出常通。G 端与 ALE 相连接，每当 ALE 下跳变时，74HC373 锁存低 8 位地址线 $A_0 \sim A_7$ ，并输出供外围芯片使用。

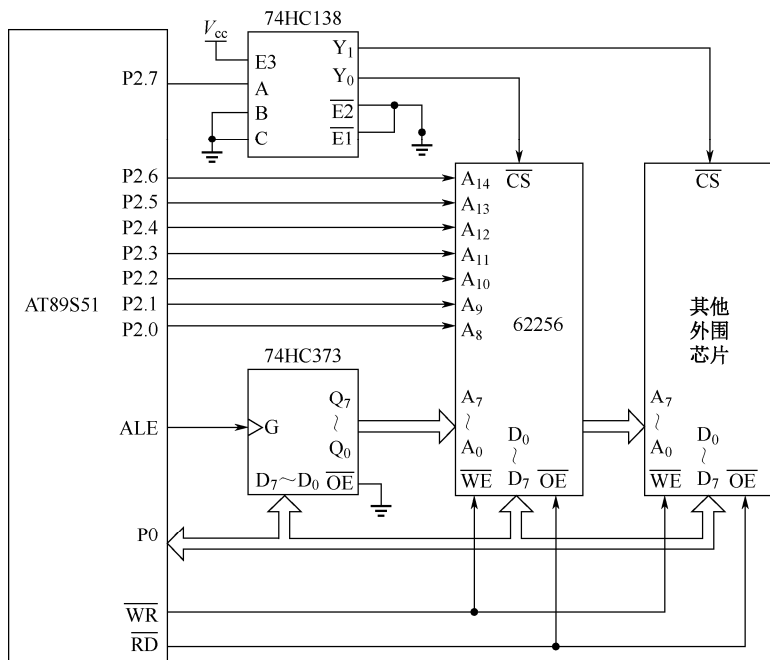


图 9-7 扩展 32KB RAM 及其他外围芯片

本例中，AT89S51 单片机采用片内程序存储器工作， \overline{EA} 应接高电平。

AT89S51 的 \overline{WR} (P3.6) 和 \overline{RD} (P3.7) 分别与 62256 的写允许 \overline{WE} 和读允许 \overline{OE} 连接，实现写/读控制。在此选用 74HC138 仅为说明当扩展多片外围芯片时地址译码线的接法及地址如何确定。62256 的片选端 \overline{CE} 与 P2.7 经 74HC138 译码后产生的 Y_0 相连，在 P2.7 为 0 时有效，所以其寻址范围为 0000~7FFFH，在 P2.7 为 1 时，可寻址范围为 8000~FFFFH，这个地址空间可用于扩充其他外围芯片。图中其他外围芯片读、写信号的名称有可能与实际芯片不同，地址线也可能不需要这么多。

在图 9-7 的线路中，若采用“MOVX @DPTR”类指令访问片外 RAM 时，AT89S51 的 P0 口和 P2 口上的全部 16 根口线同时用作传递地址信息。

9.3 并行 I/O 接口的扩展

80C51 单片机的 4 个 8 位并行 I/O 口可以满足一般的简单使用要求，但是在有些情况，例如 I/O 口数量不够用，或者带负载能力不能满足要求时，需要对单片机应用系统进行 I/O 口的扩展，当然也可选择 I/O 口多的其他型号单片机芯片。

9.3.1 扩展并行 I/O 口简述

80C51 系列单片机的 P0~P3 口具有输入数据可以缓冲，输出数据可以锁存的功能（见第 2 章），并且有一定的带负载能力，因而在有些简单应用的场合 I/O 口可直接与外设相接，例如，开关、发光二极管等。但在需要扩展 I/O 口，或者需要提高系统带负载能力的情况，则要采用锁存器、缓冲/驱动器等作为 I/O 口扩展芯片，这是单片机应用系统中经常采用的方法。扩展的 I/O 口通常应该满足以下主要功能：

1. 输出数据锁存

由于 CPU 与 I/O 设备在时序上通常不一定匹配,因而在工作时一般不同步,所以接口电路必须具备锁存与缓冲的功能。特别是 P0 口在用做数据/地址复用线时,通常是先发地址信号再发数据,CPU 在执行指令时需先把地址信息置入锁存器,以后 I/O 接口设备可按自己的时序从锁存器取得地址信息。

2. 输入数据缓冲

当单片机与多个输入接口相连时,为避免输入数据混乱,每一时刻仅允许一个接口发送数据,此时对其他接口要进行隔离操作,因此要求单片机与具有隔离功能的三态缓冲器相连。

3. 增加带负载能力

可以为外设提供足够的驱动功率,以保证外设能正常平稳地工作。

4. 地址译码及外设选择

CPU 通过译码电路可以选择 I/O 接口芯片,同时通过该芯片选择不同的外设。

在 80C51 单片机中,扩展的 I/O 口采取与数据存储器相同的寻址方法。所有扩展 I/O 口或相当于 I/O 外设以及通过扩展 I/O 口连接的外设均与片外数据存储器统一编址,所以对片外 I/O 口的输入输出指令就是访问片外 RAM 的指令。

扩展并行 I/O 口所用芯片主要有通用可编程 I/O 芯片和 TTL、CMOS 锁存器、缓冲器电路芯片等两大类。

可编程序接口芯片是指其功能可由计算机的指令来加以改变的接口芯片。为满足计算机硬件需要,生产了多种可编程并行接口芯片,例如,早期生产的可编程 I/O 芯片 8255A、计数/定时器 8253、可编程串行接口 8250 等,这些芯片多数已经在“微计算机原理及应用”等课程中作了介绍,多年前单片机在进行并行口扩展时常采用 8255A 等芯片(可参看文献 2),但现在随着单片机 I/O 口功能及数量的增强及 8255A 等芯片的停产,已经很少再用此类方法了,所以本节仅以单片机中常用的简单并行 I/O 口扩展法的具体电路为例说明问题。

9.3.2 简单并行 I/O 口的扩展

这种 I/O 口扩展方法,具有电路简单、成本低、配置灵活的优点。一般在扩展单个 8 位输出/输入口时,十分方便。

可以作为简单 I/O 扩展使用的芯片有:74HC373、74HC377、74HC244、74HC245、74HC273 等。在实际应用中可根据系统对输入、输出的要求,选择合适的扩展芯片。

图 9-8 为采用 74HC244 作扩展输入、74HC273 作扩展输出的简单 I/O 扩展电路。显然在此利用单片机的 9 个 I/O 口扩展了 16 路输入和输出口,并且均提高了带负载能力。

P0 口为双向数据线,既能从 74HC244 输入数据,又能将数据传输给 74HC273 输出。输出控制信号由 P2.0 和 \overline{WR} 合成,当二者同时为 0 电平时,“或”门输出 0,将 P0 口的数据锁存到 74HC273,其输出控制发光二极管 LED 的亮、灭。当某位输出 0 电平时,该线上的 LED 发光。

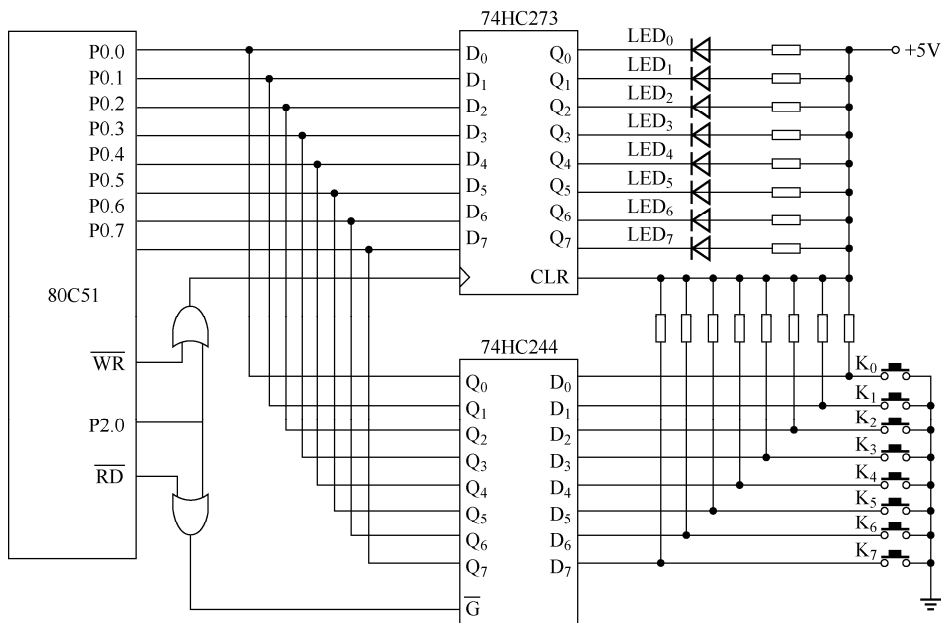


图 9-8 简单 I/O 接口扩展电路

输入控制信号由 P2.0 和 \overline{RD} 合成, 当二者同时为 0 电平时, “或”门输出 0, 选通 74HC244, 将外部信息读入到总线。当与 74HC244 相连的按键开关无键按下时, 输入全为 1, 若按下某键, 则所在线输入为 0。可见, 输入和输出都是在 P2.0 为 0 时有效, 它们占有相同的地址空间, 即口地址均为 FEFFH, 实际只要保证 P2.0=0, 与其他地址位无关, 例如, 如果地址为 00FFH 同样可以选中这 2 个芯片, 但由于它们分别用 \overline{RD} 和 \overline{WR} 信号控制, 因而尽管它们都直接与 P0 口相接, 却不可能同时被选中, 这样在总线上就不会发生冲突。

系统中若有其他扩展 RAM, 或其他输入/输出接口, 则可用线选法或译码法将地址空间区分开。

按照图 9-8 电路的接法, 要求实现如下功能: 任意按下一个键, 对应的 LED 亮, 例如, 按 K1 则 LED1 亮, 按 K2 则 LED2 亮等。则编写汇编程序如下:

```

LOOP: MOV     DPTR,    #0FEFFH           ; 数据指针指向扩展 I/O 口地址
      MOVX     A,       @DPTR             ; 从 74HC244 读入数据, 检测按钮
      MOVX     @DPTR, A                   ; 向 74HC273 输出数据, 驱动 LED
      SJMP     LOOP                     ; 循环
  
```

9.4 串行扩展概述

因为用并行总线进行系统扩展要占用较多的 I/O 口, 线路较复杂, 为了能进一步缩小单片机及其外围芯片的体积, 降低价格, 简化互连线路, 目前在单片机应用系统中开始越来越广泛地采用串行扩展总线进行系统扩展。近年来, 各单片机制造厂商先后推出专门用于串行数据传输的各类器件和接口, 这些串行总线均已获得广泛应用。

9.4.1 常用串行总线与串行接口简介

常用串行总线由早期的 UART 串行接口一种, 发展为多种。目前广泛使用的串行扩展总

线与串行扩展接口主要有 I²C 总线、SPI 串行接口、CAN、USB 串行总线和单总线等。其中 UART 串行接口已经在第 7 章详细介绍过，本节将分别简介其他较常用串行总线及接口。

1. I²C 总线

I²C（Inter Integrated Circuit）总线是 Philips 公司推出的，推出后即以其完善的性能、严格的规范（如接口的电气特性、信号传输的定义及时序等）和简便的操作方法被其他半导体厂商和用户所接受，随后出现的带 I²C 接口的单片机和带 I²C 接口的外围芯片（存储器、模数转换等）推动了它的广泛应用。

I²C 总线由 2 根线实现串行同步通信，其中一根是时钟线 SCL，一根是数据线 SDA。I²C 总线单主机系统配置见图 9-9。在 I²C 总线中每一个 I²C 接口称为一个接点。I²C 总线上数据传输的速率一般为 100Kb/s，目前的器件可以达到 400Kb/s 以上。

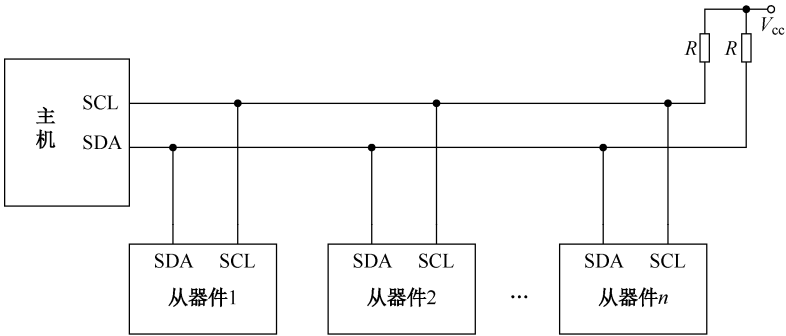


图 9-9 典型的 I²C 单主机系统配置示意图

I²C 总线的原理及应用详见 9.5 节。

2. SPI 串行扩展接口

SPI（Serial Peripheral Interface）串行接口是 MOTOROLA 公司推出的一种同步串行外围接口，允许不同厂家的单片机和带 SPI 接口的外围芯片相连。

SPI 接口可以以主机方式或从机方式工作，主机最大位传输速率为 1.05Mb/s，目前有的型号速率已经超过此值。其可与各种采用串行移位方式工作的外围器件进行通信。

采用 SPI 接口可以构成不同的系统，多数应用场合是用一个单片机作为主机，控制一个或多个从外围器件传输数据，这些外围器件接收或提供传输的数据。单主机 SPI 系统连接方法如图 9-10 所示。由图可以看出主、从机之间的关系和相互连线。从机可以是单片机，也可以是具有 SPI 接口的外围器件，主机也可以采用没有 SPI 接口的单片机，但此时需要通过软件模拟其接口。

由图 9-10 可见其主机与从机的时钟线与数据线均为同名端相接。SPI 串行扩展接口需要用到 3 根通信线，这 3 根线是 SCK 串行时钟线、MOSI 主机输出/从机输入线、MISO 主机输入/从机输出线，此外带 SPI 串行扩展接口的器件都有片选端 SS，SPI 系统主机上的输出口 1~n，用于选择从器件 1~n 的片选端 SS。

MOSI 和 MISO 这两个数据引脚用于接收和发送串行数据，传输时是最高位（MSB）在先，最低位（LSB）在后。对于主机的 SPI 口，MISO 是主机数据输入端，MOSI 是主机数据输出端。对于从机的 SPI 口，MISO 是从机数据输出端，MOSI 是从机数据输入端。主机位速

率、串行时钟极性与相位可编程。

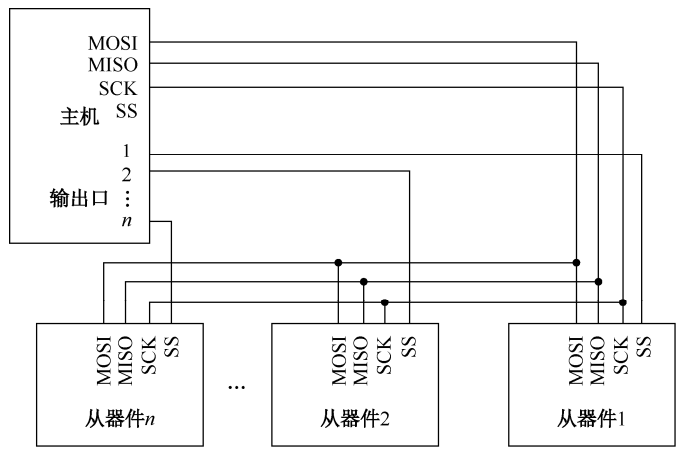


图 9-10 单主机 SPI 系统连接方法

SCK 是通过 MISO 和 MOSI 输入或输出数据的同步时钟。作为主机的芯片，SCK 是主机时钟输出端；作为从机的芯片，SCK 是从机时钟输入端。串行口每发送或者接收一位数据都有一个时钟脉冲同步控制。

3. USB 总线

USB 总线是通用串行总线（Universal Serial Bus）的简称，USB 是一种快速的、双向同步传输的、廉价并可以进行热插拔的串行接口，使用十分方便，是 PC 机的一种标准接口，可接入多达 127 个不同的设备。此外，USB 接口支持低速、全速和高速 3 种数据传输速率。目前已生产出多种带有 USB 接口的单片机和能与单片机相配接的 USB 接口芯片。

在 USB 总线中只有一个主机。USB 系统采用拓扑总线结构。该拓扑由 3 个基本部分组成：主机，集线器（Hub）和功能设备，如图 9-11 所示。集线器（Hub）是 USB 结构中的特定部分，集线器主要用于管理连接到其端口的设备，收、发主机的信息。如果想连接更多的 USB 外设，则利用 USB 集线器扩展，该集线器可提供多个 USB 端口。USB 采用 4 线电缆，其中两根是用来传输数据的串行通道，另两根为电源线，USB 设备本身不需要单独的电源，只需利用计算机或集线器中电源。

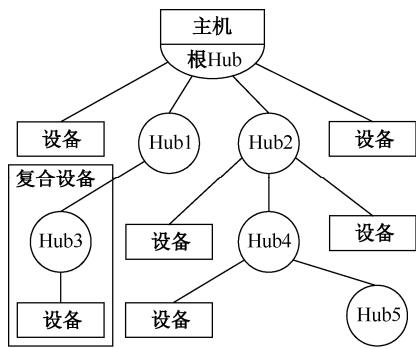


图 9-11 USB 总线系统结构

4. CAN 总线

CAN(Controller Area Network)即控制器局域网，是用于各种设备检测及控制的一种现场总线。20 多年前由德国 Bosch 公司首先提出，很快在工业控制领域得到了广泛应用，近年已经有多种带有 CAN 总线接口的单片机和能与单片机相配接的 CAN 接口芯片。

CAN 总线用于数据通信具有突出的可靠性、实时性和灵活性，抗干扰能力强。其主要特点是：结构简单，只有两根线与外部相连；通信方式灵活，为多主方式工作；通信距离最大可达 10km（速率 5kb/s 以下），最高通信速率可达 1Mb/s（此时距离最长为 40m）；通信介质可以是双绞线、同轴电缆或光导纤维。

图 9-12 所示为 CAN 总线系统结构图。一个总线节点通常包括 3 部分：控制节点任务的单片机、CAN 总线控制器及 CAN 总线驱动器。对于内部已经集成 CAN 控制器的单片机，要使总线运行，只要接 CAN 驱动器即可。

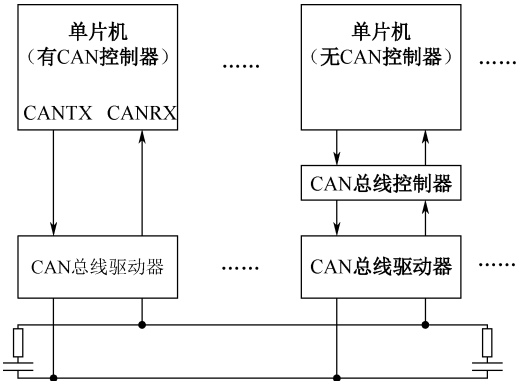


图 9-12 CAN 总线系统结构

5. 单总线

单总线是由 DALLAS 公司推出的外围串行扩展总线。单总线只有一根数据输出输入线 DQ，所有的器件都挂在这根线上。DALLAS 公司生产的最著名的单总线器件是数字温度传感器，例如 DS1820、DS1620 等。每个单总线器件都有 DQ 接口，DQ 接口是漏极开路，需加上拉电阻。DALLAS 公司为每个器件都提供了一个唯一的地址，并为器件的寻址及数据传输制定了严格的时序规范。图 9-13 为用单总线构成的温度检测系统。

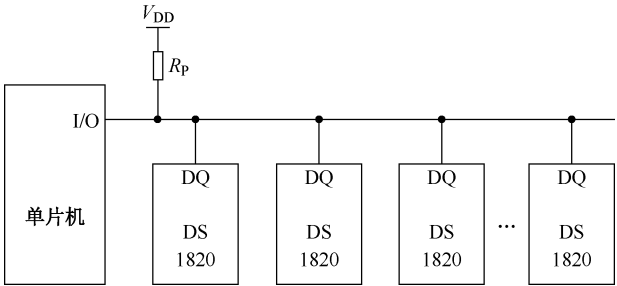


图 9-13 单总线构成的温度检测系统

单总线与目前多数标准串行数据通信方式不同，它采用单根信号线，既传输数据位，又

传输定时同步信号，而且数据传输是双向的。大多数单总线器件不需要额外的供电电源，可直接从单总线上获得足够的电源电流（即寄生供电方式）。它具有节省 I/O 口线、结构简单、成本低廉、便于总线扩展和维护等诸多优点。主要缺点是软件设计较复杂。

单总线适用于单主机系统，主机能够控制一个或多个从机设备。

9.4.2 单片机串行扩展的模拟技术

因为串行总线除了要求扩展的外围器件有相应的串行接口外，还要求计算机有相应的串行接口，目前单片机大多数都有 UART 串行接口，但多数都不同时具备上述几种串行接口。通常是具备其中的 1~3 种接口，因而为推广串行扩展技术，就需要采用模拟接口技术，即用单片机的通用 I/O 接口通过软件模拟（也可称为虚拟）串行接口的时序和运行状态，构成模拟的串行扩展接口。这样任何具有串行扩展接口的外围器件就都可以扩展到任何型号的单片机应用系统中。

成功实现串行扩展模拟技术的主要要点如下：

1. 严格模拟时序

目前大多数串行扩展总线和扩展接口都采用同步数据传输，在同步传输中，是由串行时钟控制数据传输的时序。所以在模拟串行时钟时，一定要严格按照规范的时序控制，满足数据传输的时序要求。

2. 确保硬件与软件的配合

不同的串行扩展总线和扩展接口所需要的传输线数、速率及规范一般不相同，需认真查看手册。在模拟传输时要考虑到相互间的配合。使用时在硬件上要符合接口标准对传输线数及时序的严格要求，在软件上要遵守标准要求的通信协议。对于在原来设计中没有这种接口的单片机只要在硬件和软件上能模拟它的通信要求，同样可以与带有这类串行通信标准的芯片相连使用。采用模拟方法时，只占用单片机的通用 I/O 口。

3. 设计通用模拟软件包

为简化模拟串行接口软件的设计，依据串行总线/接口规范，可以设计出各种类型接口的通用模拟软件包。这样在应用程序设计时直接调用软件包中的子程序，就可以完成相应的数据输入/输出操作，简化了串行扩展接口软件设计。

9.4.3 串行扩展的主要特点

综上所述，串行总线的共同特点是仅占用很少的资源和 I/O 线，一般只需 1~3 根信号线，结构紧凑，可大大减小系统体积，降低功耗。串行总线可十分方便地用于构成由 1 个单片机和一些外围器件组成的单片机系统，系统易修改，且可扩展性好。它的上述优点大大简化了 PC 机与单片机的连接，推动了分布式系统和网络的发展。随着串行通信协议软件包的成熟及模块化，使得串行通信的编程变得简单易行，因而近年来串行扩展技术发展迅速，使它在某些领域逐渐得到广泛应用。

这种总线结构的主要问题是沒有并行总线的吞吐能力强，且速度較慢，并且在软件编程上較复杂，通常用于对速度要求不高，传输的数据量不是很大的情况。在系统扩展时究竟采用哪种方法应根据扩展应用的主要要求决定。

在使用时，要注意不同的串行总线规范不一样，須认真参看手册中的具体规定。

限于篇幅，本章主要简介目前应用较普遍的 I²C 总线的应用方法。

9.5 I²C 总线

I²C 总线是一种用于 IC 器件之间的二线制同步串行通信总线，它通过两根线在连到总线上的器件之间传输信息，根据每个器件的硬件编址进行寻址。I²C 总线可以方便地构成多机系统和外围器件扩展系统。

9.5.1 I²C 总线的组成及基本工作原理

I²C 总线上的器件有主、从之分，当执行数据传输时，启动数据传输，并发出时钟信号的器件称为主器件，被寻址的任何器件都可看做从器件。主器件在发送启动信号和时钟后，立即发寻址字节，寻找从器件，并规定传送方向，传送完毕再发终止信号。所有主器件（单片机、微处理器等）、从器件等都连到同名端的 SDA 串行数据线、SCL 串行时钟线上。按照二线制串行通信总线的传输规定，将把数据传输到总线上的器件称为发送器，而把从总线上接收数据的器件称为接收器。在 I²C 总线传输时，主器件或从器件都可能处于发送或接收的工作方式。I²C 总线是一种多主系统，即总线上可以有多个主器件，当系统中有多个主器件时，任何一个主器件在 I²C 上工作时都可以成为主控制器，由于 I²C 总线具有仲裁功能，当主器件工作发生冲突时，只允许一个有效，所以不会影响数据传输。通常最常用的方式是单主系统。图 9-9 是一个典型的多个从器件与一个单片机连接的单主机系统配置示意图。详细应用实例见 9.5.4。SDA 和 SCL 都是双向 I/O 线。接口电路为漏极开路，要通过上拉电阻接正电源。当总线空闲时，两根线都是高电平。连接总线的外围器件都是 CMOS 器件，输出级也是开漏电路，在总线上消耗的电流很小。

在图 9-9 中，数据传输是以下述方式进行的：如果主机（主器件）要把信息送至器件 1，则主机首先寻址器件 1，然后主机（主发送）把数据送至器件 1（从接收），最后由主机终止传输；如果主机要从器件 1 接收信息，则首先由主机寻址器件 1，然后主机（主接收）接收器件 1（从发送）的数据，最后由主机终止接收。

9.5.2 I²C 总线的传输时序

I²C 总线在不同工作状态时的运行时序关系见图 9-14。由图可见，一次典型的 I²C 总线数据传输包括：一个起始条件（START）、一个地址字节（SLA6~0 即 7 位从地址：位 7~1；R/W 方向位为位 0:）、一个或多个字节的数据（本图只发了一个字节数据）和一个停止条件（STOP）。每个地址字节后面都跟随一个来自接收器的应答位 ACK（ACKNOWLEDGE 也可译为确认），每个数据字节后面都跟随一个来自接收或发送器的应答位或非应答位 NACK（NACKNOWLEDGE）。方向位被设置为逻辑 1，表示这是一个“读”操作；方向位为逻辑 0，

表示这是一个“写”操作。所有从器件都能识别一个全局呼叫地址（00+R/W），它允许一个主器件同时访问多个从器件。

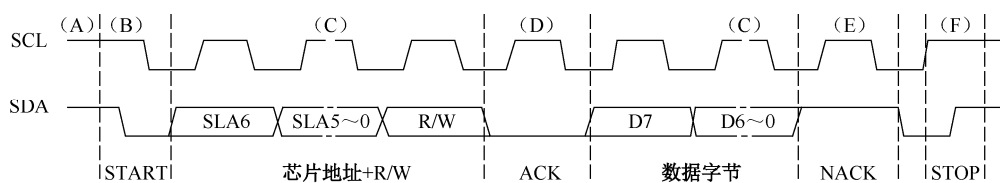


图 9-14 I²C 总线运行时序关系图

对不同总线状态的说明如下。

① A 段：总线空闲状态

数据线 SDA 和时钟线 SCL 都为高电平，为总线空闲状态，空闲时间必须大于 4.7μs。

② B 段：开始（START）数据传输

当串行时钟 SCL 处于高电平时，如果 SDA 从高电平下降到低电平，则表示启动开始状态（从 SDA 下降至 SCL 下降的时间间隔应该大于 4.0μs）。只有出现“启动开始”信号之后，其他命令才有效。

③ C 段：数据传输

在出现“开始”信号后，在时钟线 SCL 处于高电平时，这时数据线的状态就表示要传输的数据。当时钟线 SCL 为低电平期间，数据线上的数据位发生改变，每位数据需一个时钟脉冲，一次顺序发送一个字节的 8 位。注意所有的数据和地址字节都是首先发送最高位（D7）。I²C 总线发送信号的第 1 个字节用来确定芯片地址（即第一个 C 段）。该字节的高 7 位组成芯片地址，在图 9-14 中 SLA6 即地址的第 6 位（但它是该字节的第 7 位），SLA5~0 即地址的第 5~0 位。最低位 R/W 是读/写位，确定数据方向位（即读/写位）。图中第 2 个 C 段传输的是数据，对于存储器芯片通常第 1 个数据信息是存储单元地址。

④ D 段：应答（ACK）信号

应答信号是主机对从机工作状态的一种检测。主机查询到从接收器件有“0”应答信号 ACK 输出，则说明其内部定时写的周期结束，可以写入新的内容。每当从接收器件接收完一个 8 位的写入地址或数据之后，会在第 9 个时钟周期出现应答信号，如图 9-14（D）段所示。此时从器件上发一个“0”应答信号 ACK。反之，当主机接收完来自从接收器件的数据后，单片机也应通过 SDA 发 ACK 信号。

⑤ E 段：非应答（NACK）信号

NACK 称非应答信号，当串行时钟 SCL 为高电平时，如果 SDA 为高电平，则表示是非应答信号。在主器件进行读操作结束后，发送 NACK 表示这是数据传输的最后一个字节。

应答位和非应答位的时序与发送数据“0”和“1”的信号要求完全相同，只要满足在时钟 SCL 高电平大于 4.0μs 期间，SDA 线上有确定的电平状态即可。

⑥ F 段：停止（STOP）数据传输

当串行时钟 SCL 处于高电平时，如果 SDA 从低电平上升到高电平，则表示这是一个停止信号（SDA 高电平时间必须大于 4.7μs）。在出现“停止”信号后所有操作都停止。为了防止非正常传输，停止信号后 SCL 可设置为低电平。

每个数据的传输都是由启动信号开始，停止信号结束。在开始与停止信号之间传输的字节数由主机决定，从理论上说对字节数没有限制。

为了保证可靠传输 I²C 总线上的数据，对总线上的时序信号有严格规定，在用普通 I/O 口模拟 I²C 总线的数据传输时，单片机模拟的时序信号要满足 SDA 和 SCL 的时序要求。

9.5.3 I²C 总线的通用模拟软件包

为了简化 I²C 总线模拟传输的软件编程方法，根据其时序特点编制了通用软件包，这个软件包包括典型信号的通用模拟子程序和 I²C 总线信号模拟传输的通用子程序，可适用于以 80C51 系列单片机为主机的 I²C 总线单主应用系统。通用软件包中的符号单元如下：

MTD: 发送缓冲区首地址
MRD: 接收缓冲区首地址
SLAW: 芯片地址与写控制字存放单元
SLA: 寻址字节存放单元
NUMBYT: 传输字节数存放单元
ERR: 错误标志
SDA: 模拟串行数据线
SCL: 模拟串行时钟线

本节 I²C 总线的通用子程序是假设主机频率为 6MHz，使用下述子程序时，应根据从器件的频率修改延时用的 NOP 指令个数，以满足时序要求。

发启动脉冲汇编语言子程序

```
STA:   SETB  SDA                ; 准备发启动开始脉冲
        SETB  SCL
        JNB   SDA,   STA1        ; 没有变高跳转
        JNB   SCL,   STA1        ; 没有变高跳转
        CLR   SDA
        NOP                    ; 确保延时时间
        CLR   SCL
        CLR   ERR                ; 清错误标志
        SJMP  STA2
STA1:  STEB  ERR                ; 置错误标志
STA2:  RET
```

发启动脉冲 C 语言子程序

```
bit II_start(void)
{
    SDA = 1;                    //准备发启动开始脉冲
    SCL = 1;
    if ((SDA==0)|(SCL==0))      //有一个没有变高出错
        ERR =1;
    else
    {
        SDA = 0;
        _nop_();                //延时
        SCL = 0;
        ERR = 0;
    }
}
```

```

        return(ERR);                //返回标志
    }
发停止脉冲汇编语言子程序
STOP:   CLR     SDA                ; 发停止脉冲
        SETB    SCL
        NOP
        SETB    SDA
        NOP
        NOP
        CLR     SCL
        RET
发停止脉冲 C 语言子程序
void II_stop(void)
{
    SDA = 0;
    SCL = 1;
    _nop_();
    SDA = 1;
    _nop_();
    _nop_();
    SCL = 0;
}
检查应答汇编语言子程序
CACK:   SETB    SDA                ; 置应答信号位
        SETB    SCL                ; 时钟变高
        CLR     ERR                ; 清错误标志
        MOV     C,    SDA
        JNC     ACK1              ; 是否响应
        SETB    ERR                ; 为高说明没有响应
ACK1:   CLR     SCL                ; 时钟变低
        RET
检查应答 C 语言子程序
bit II_Cack(void)
{
    SDA = 1;                        //置应答信号位
    SCL = 1;                        //时钟变高
    ERR = 0;
    if(SDA==1) ERR=1;              //为高说明没有响应
    SCL = 0;
    return (ERR);                  //返回标志
}
此应答信号是由从器件发给主机的，所以加一个判断其是否变为低，效果更可靠。
发送应答汇编语言子程序（这是主机发给从器件的）
MACK:   CLR     SDA                ; 置应答位为低
        SETB    SCL                ; 时钟变高
        NOP
        NOP

```

```

        CLR    SCL                ; 时钟变低
        SETB   SDA                ; SDA 为高
        RET
发送应答 C 语言子程序
void II_Mack(void)
{
    SDA = 0;                      //置应答位为低
    SCL = 1;                      //时钟变高
    _nop_();                      //延时
    _nop_();
    SCL = 0;                      //时钟变低
    SDA = 1;                      // SDA 为高
}
非应答汇编语言子程序
NAK:    SETB    SDA                ; 置非应答信号位
        SETB    SCL                ; 时钟变高
        NOP
        NOP
        CLR     SCL                ; 时钟变低
        CLR     SDA
        RET
非应答 C 语言子程序
void II_NAK(void)
{
    SDA = 1;                      //置非应答信号位
    SCL = 1;                      //时钟变高
    _nop_();
    _nop_();
    SCL = 0;                      //时钟变低
    SDA = 0;
}
写（发送）一个字节汇编语言子程序
WR_1BYTE:MOV    R6,    #08
WR_LP:  RLC     A                ; 从最高位开始左移进入 C
        MOV     SDA,    C        ; 逐位写入 SDA
        SETB    SCL                ; 时钟变高
        NOP
        NOP
        CLR     SCL                ; 时钟变低
        DJNZ    R6,     WR_LP    ; 一个字节是否写完
        RET
写（发送）一个字节 C 语言子程序
void II_write(unsigned char I_data)
{
    unsigned char i;
    for (i=0;i<8;i++)            //8 次发送一个字节
    {

```

```

SDA = (bit)(I_data & 0x80);      //发送一位
I_data <<= 1;                     //左移一位
SCL = 1;
_nop_();
_nop_();
SCL = 0;
}
}

```

读（接收）一个字节汇编语言子程序

```

RD_1BYTE:  MOV    R6,    #08      ; 置位数
RD_LP:     SETB    SDA          ; 置 SDA 为输入状态
SETB       SCL              ; 使 SDA 数据线有效
MOV        C,        SDA      ; 读入一位
RLC        A              ; 将 C 移入 A
CLR        SCL              ; 时钟变低
DJNZ       R6,       RD_LP     ; 是否读完一个字节
RET

```

读（接收）一个字节 C 语言子程序

```

unsigned char  II_read(void)
{
    unsigned char i,I_data=0;
    for (i=0;i<8;i++)
    {
        I_data <<= 1;          //接收的数据左移一位
        SDA = 1;
        SCL = 1;
        if(SDA==1) I_data +=1; //读入一位数据
        SCL = 0;
    }
    return(I_data);
}

```

写数据块汇编语言子程序

```

WRNBYT:LCALL STA          ; 调启动脉冲子程序
        MOV    A,        SLAW      ; 芯片地址与控制字送 A
        LCALL  WR_1BYTE   ; 写入芯片地址和写命令
        LCALL  CACK       ; 检查应答位
        JB     ERR,       WRNBYT   ; 不正确返回
        MOV    A,        SLA      ; 存储单元地址送 A
        LCALL  WR_1BYTE   ; 写入一个字节数据的字节地址
        MOV    R0,       #MTD     ; 发送数据首地址送 R0
W1C:    MOV    A,        @R0
        LCALL  WR_1BYTE   ; 调写入一个字节子程序
        LCALL  CACK       ; 检查应答位
        JB     ERR,       WRNBYT   ; 不正确返回
        INC    R0
        DJNZ   NUMBYT,    W1C
        LCALL  STOP       ; 发停止脉冲子程序
        RET

```

写数据块 C 语言子程序

```
void II_write_block(uchar command,uchar address,uchar number)
{
    unsigned char i;
retry:
    II_start();                //调启动脉冲函数
    II_write(command);         //写入命令
    if (II_Cack()) goto retry; //检查应答位
    II_write(address);         //写入地址
    for(i=0;i<number;i++)     //写入数据
    {
        II_write(II_BUF[i]);
        if (II_Cack()) goto retry;
    }
    II_stop();                 //停止脉冲
}
```

读数据块汇编语言子程序

```
RDNBYT :LCALL STA                ; 调开始脉冲子程序
        MOV     A,                SLAW    ; 芯片地址与写控制字送 A
        LCALL   WR_1BYTE          ; 写入一个字节数据的芯片地址和写命令
        LCALL   CACK              ; 检查应答位
        JB      ERR,              RDNBYT  ; 不正确返回
        MOV     A,                SLA     ; 存储单元地址送 A
        LCALL   WR_1BYTE          ; 写入一个字节数据的字节地址
        LCALL   CACK              ; 检查应答位
        JB      ERR,              RDNBYT  ; 不正确返回
        MOV     A,                SLAW    ; 芯片地址与写控制字送 A
        SETB    ACC.0             ; 改为读控制
        LCALL   WR_1BYTE          ; 写入一个字节的芯片地址和读命令
        LCALL   CACK              ; 检查应答位
        JB      ERR,              RDNBYT  ; 不正确返回
        MOV     R0,              #MRD    ; 接收缓冲区首地址送 R0
RLCALL  RD_1BYTE                ; 调读 1 个字节数的子程序
        MOV     @R0,              A       ; 把读到的数保存到单片机中
        INC     R0                ; 修改地址指针
        LCALL   MACK              ; 调发应答子程序
        DJNZ    NUMBYT,          RR_D     ; 调非应答信号子程序
        LCALL   NAK               ; 调非应答信号子程序
        LCALL   STOP              ; 发停止脉冲子程序
        RET
```

读数据块 C 语言子程序

```
void II_read_block(uchar command,uchar address,uchar number)
{
    uchar i;
retry1:
    II_start();                //开始脉冲
    II_write(command);         //写入命令
    if (II_Cack()) goto retry1;
```

```

II_write(address);           //写入地址
if (II_Cack()) goto retry1;
II_write(command+1);         //写入读命令
if (II_Cack()) goto retry1;
for (i=0;i<number;i++)
{
    II_BUF[i]=II_read();      //读出数据
    II_Mack();
}
II_NAK();                     //非应答信号
II_stop();                     //停止
}

```

显然简化硬件的代价是增加了软件的复杂性，不过因为所有符合这种接口的存储器外围器件都可以采用上述模拟子程序，所以就减轻了编程的负担。

9.5.4 I²C 总线应用举例

目前，已生产出多种串行接口芯片（例如有存储器、A/D、D/A 和时钟等），其中串行 EEPROM 是在各种串行芯片中使用较多的芯片，它特别适合于存放修改后可长期保存的数据。本节介绍串行 EEPROM 芯片的寻址、操作以及和 89 系列单片机的接口技术。

串行 EEPROM 中，较为典型的有 Atmel 公司的 AT24CXX 系列，其他公司也有类似产品，这些芯片在使用方法上有一定共性，限于篇幅，本节仅以 AT24CXX 系列产品举例说明。

1. 存储器组织及引脚功能说明

AT24CXX 系列的串行电可改写及可编程只读存储器有 10 多种型号，其中典型的型号有 AT24C01A/02/04/08A/16A 等 5 种，这 5 种型号的结构原理类似，其主要差别在于存储器容量不同。所以 EEPROM 的地址硬布线也有所不同，即它们的地址引脚数量虽然有 A0~A2 共 3 条，但有效地址引脚却不同。同时，在 2 线串行总线上可以连接的片数也有区别。AT24CXX 系列的 EEPROM 典型型号的存储器参数见表 9-1。由表可见各种型号 EEPROM 存储区大小、页面大小、容量、总线可接片数及可用地址脚的情况。

表 9-1 AT24CXX 系列各种型号 EEPROM 参数

型号	容量/B	区数	页数/区	字节数/页	可连续写字字节数	可接片数	可用地址脚
AT24C01A	128	1	16	8	8	8	A0, A1, A2
AT24C02	256	1	32	8	8	8	A0, A1, A2
AT24C04	512	2	16	16	16	4	A0, A1
AT24C08A	1024	4	16	16	16	2	A2
AT24C16A	2048	8	16	16	16	1	无

该系列芯片的引脚排列如图 9-15 所示。

各引脚的名称和功能如下。

- ① Vcc: +5V 电源。
- ② GND: 地线。

- ③ SCL：串行时钟输入端。
- ④ SDA：串行数据 I/O 端，用于输入和输出串行数据。
- ⑤ A0, A1, A2：芯片地址引脚。不同型号接法不同。

例如，对于 AT24C01A 和 AT24C02，A0, A1, A2 这 3 位引脚均可以用于芯片寻址。当用 8 片 AT24C01A 组成 1KB 的存储器时，则第 1 片地址为“000”，故 A0, A1, A2 全部接地；第 2 片地址为“001”，故 A0 接高电平 5V，A2, A1 接地；……第 8 片地址为“111”，故 A0, A1, A2 全部接高电平 5V。其他型号类似。

⑥ WP：写保护端。通过此引脚可提供硬件数据保护。当把 WP 接地时，允许芯片执行一般读写操作；当把 WP 接到 VCC 时，则对芯片实施写保护。

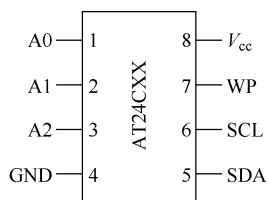


图 9-15 AT24CXX 引脚图

2. 芯片及存储单元寻址

对于 AT24C01A 等 5 种典型的 EEPROM 芯片寻址包括芯片和存储单元寻址 2 部分。

① 芯片寻址

芯片寻址（也可译为器件寻址）就是用一个 8 位的芯片地址字去选择存储器芯片。在 AT24CXX 系列的 5 种芯片中，如果逻辑电路所确定的硬件地址和芯片地址字相比较结果一致，则该芯片被选中。芯片寻址的过程和定义如下。

用于存储器 EEPROM 芯片寻址的芯片地址字如图 9-16 所示。它有 4 种形式，分别对应于 1K/2K、4K、8K 和 16K 位的 EEPROM 芯片。

从图 9-16 中看出：芯片地址及读写控制字含如下 3 个部分。

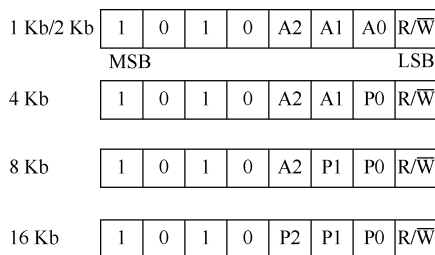


图 9-16 芯片地址及读写控制字

第 1 部分：芯片标识位，芯片地址字的最高 4 位。对于 AT24CXX 系列芯片，这 4 位的内容恒为“1010”，

第 2 部分：地址位，为第 1~3 位。这 3 位有 2 种符号：Ai (i=0~2)、Pj (j=0~2)，其中 Ai 表示外部硬件布线地址位，Pj 表示芯片页地址位。

例如，对于 AT24C10A/02 这两种 1K/2K 位的 EEPROM 芯片，硬件布线地址为“A2、A1、A0”。在应用时，“A2、A1、A0”的内容必须和 EEPROM 芯片的 A2、A1、A0 的硬件布线情况一致。工作时，芯片地址与逻辑连接情况相比较，如果一样，则芯片被选中；否则，不选择。

对于 AT24C08 这种 8 Kb 的芯片，地址位为“A2、P1、P0”。在应用时，“A2”的内容必须和芯片 A2 端的硬件布线一致，芯片才能被选中。“P1、P0”是芯片内部的区地址，所以，AT24C08 的 A1、A0 端不用连接。

对于 AT24C16A 这种内部 16 Kb 的芯片，A2、A1、A0 端不用连接。地址位“P2、P1、P0”用于内部区寻址。

第 3 部分：读/写选择位，是芯片地址字的最低位，用 R/W 表示。当 R/W=1 时，执行读操作；当 R/W=0 时，执行写操作。

② 存储单元寻址

在选中芯片地址之后，主机将接着送一个“字节地址”至总线上。“字节地址”是一个 8 位的地址信息，是用于对存储器中的存储单元寻址的。当对不同芯片内的存储单元寻址时，其实际地址是由硬布线地址中的页面地址和字地址结合而成的，其页面地址为实际的高位，字地址为低位。不同芯片的实际存储单元地址如下：

AT24C01A/02：实际地址 = 字节地址，即 0~7FH/FFH

AT24C04：实际地址 = $P0 \times 2^8 + \text{字节地址}$ ，即 $(0 \sim 1) \times 100H + (0 \sim FFH)$

AT24C08A 实际地址 = $(P1、P0) \times 2^8 + \text{字节地址}$ ，即 $(0 \sim 3) \times 100H + (0 \sim FFH)$

AT24C16A：实际地址 = $(P2、P1、P0) \times 2^8 + \text{字节地址}$ ，即 $(0 \sim 7) \times 100H + (0 \sim FFH)$

以 AT24C08A 为例说明某单元实际地址的确定方法，如果“A2、P1、P0”为“1、0、1”，字地址为 30H，则这个字节的实际地址是在 A2 接为高电平的 AT24C08 的 130H。

3. 读写操作

为了更好地理解 I²C 总线的工作原理，简介对该芯片的读、写操作过程。

① 写操作

写操作有“单字节写”、“多字节连续写”两种不同的写入方法，下面以“多字节连续写”为例介绍。

不管那种方式在起始状态中，都要首先写入 8 位的芯片地址，则 EEPROM 芯片会产生一个“0”信号 ACK 输出作为应答；接着，写入 8 位的字地址，在接收了字地址之后，EEPROM 芯片又产生一个“0”应答信号 ACK；然后写入 8 位数据。多字节写过程和单字节写过程的主要区别是在发送完第 1 个数据，并收到应答信号后，不发停止信号，而是继续发送其他数据。单片机可以连续向 EEPROM 芯片发送 1 个数据块的数据。例如对于 AT24C01A/02，可连续发送共 8 个字节。对于 AT24C04/08A/16A，则共可发送 16 个字节。当然，每发一个字节都要等待芯片的应答信号 ACK。

注意在写数据块时，可以写入的数据量超过 1 页的容量时，字地址会产生循环覆盖，先前所写的数据会被新的所覆盖。如果跨页写超过芯片规定的页地址也会产生覆盖问题。

② 读操作

对芯片读操作的启动是和写操作类似的，不同的是 R/W 位信号为“1”时执行读操作。读操作有 3 种方式，即“当前地址读”、“随机读”和“连续读”。下面以“连续读”为例说明它的工作过程。

“连续读”（也称为多字节读）即连续读存储单元的数据，可以从当前地址开始也可以从任意一个指定字节地址开始。当 EEPROM 芯片接收了芯片地址及字节地址时，在芯片产生应答信号 ACK 之后，则 EEPROM 串行输出被读数据。“连续读”和“当前地址读”、“随机读”的最大区别在于：“连续读”在读出一批数据之后才由单片机产生停止信号结束读操作；而“当前地址读”和“随机读”在读出一个数据之后就由单片机产生停止信号结束读操作。执行“连续读”时，每读出一个数据，单片机都要发应答信号，只有在单片机产生非应答信号，之后马上发停止信号，才会结束“连续读”操作。

注意，在“连续读”时，读地址范围不能超过芯片规定的地址范围，否则会产生读重叠。例如对于 AT24C01A “连续读”不能超过 128 字节。

在对 AT24CXX 系列执行读写的 2 线串行总线工作中，对于无 I²C 总线接口的单片机是

由程序产生有关时序信号。有两点特别要注意：串行时钟必须由单片机执行程序产生，而写地址或数据时由 EEPROM 产生 ACK，读数据时由单片机产生 ACK。

4. I²C 总线接口实例

本文例题是用 AT89S51 作为主机，扩展一块 AT24C08A 和 AT24C02 的接口电路，连接方法如图 9-17 所示。在图中，AT89S51 通过 P1 端口的 P1. 1（作为串行时钟线 SCL）、P1. 2（作为串行数据线 SDA）和 AT24C02、AT24C08A 的 SCL、SDA 端相连，从而形成了 2 线串行总线。为了使 SCL、SDA 能保持可靠的高电平，在 2 线串行总线上都接了上拉电阻。AT24C02 的硬件布线地址为“000”，故 A2、A1、A0 都接地；AT24C08 的硬布线地址为“100”，故 A2 接 5 V。

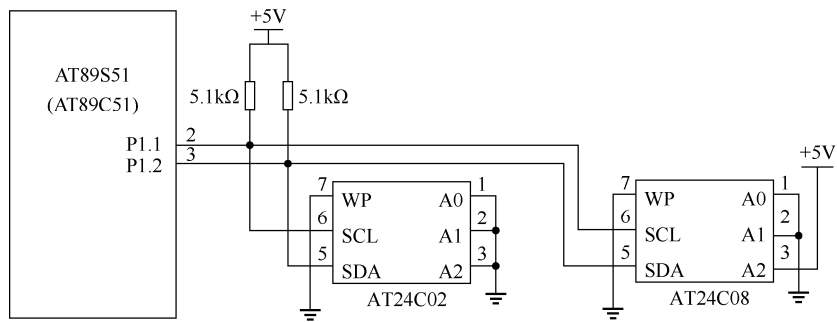


图 9-17 AT89S51 和 AT24C02、AT24C08A 的接口

为了在 AT89S51 和 AT24C02、AT24C08 之间进行正常的数据写入和读出，要求 AT89S51 的工作程序能够以恰当的时序产生串行时钟 SCL 和串行数据 SDA。

在此例题中准备实现从 AT24C02 的 40H 单元中连续读 8 个数据，然后存入单片机的以 60H 为首地址的连续 8 个单元中。再把从单片机的以 60H 为首地址的连续 16 个单元中取出的数据存入 AT24C08 的以 220H 单元为首地址的连续 16 个单元中。由于篇幅关系，省略了有关出错的处理。当所用的单片机 AT89S51 工作频率为 12MHz 时，程序满足所有 AT24CXX 系列型号的时序要求，程序中出现的 NOP 指令是用于时间延迟的。当采用更高速度的时钟时，则程序的时间延迟需要做一定的修改。

汇编语言程序清单如下：

```
SLAW    EQU    31H        ; 芯片地址与写控制字存放单元
SLA     EQU    32H        ; 寻址字节地址存放单元
NUMBYT  EQU    33H        ; 传输字节数存放单元
SDA     BIT    P1.2       ; 串行数据线
SCL     BIT    P1.1       ; 串行时钟线
ERR     BIT    30H        ; 错误标志
MTD     EQU    40H        ; 发送数据缓冲区首地址
MRD     EQU    60H        ; 接收数据缓冲区首地址
ORG     0000H
LJMP    MAIN

... ..
MAIN:   MOV     SP,      #70H    ; 设堆栈指针
        SETB    SDA          ; 初始化 AT2402/AT2408 总线
        SETB    SDL
```

```

LCALL    DELAY                ; 调延时子程序（省略）
... ..
MOV      SLAW,    #0A0H      ; 置 2402 芯片地址和写命令字
MOV      SLA,     #040H      ; 置字节地址
MOV      NUMBYT, #8          ; 置传输数据长度
LCALL    RDNBYT              ; 调读数据块子程序
... ..
MOV      SLAW,    #0ACH      ; 置 2408 芯片地址和 RAM 地址高 3 位（A2、P1、P0=1、
                                1、0）以及写命令字
MOV      SLA,     #20H       ; 送入字节地址（低 8 位）
MOV      NUMBYT, #16         ; 置数据长度
LCALL    WRNBYT              ; 调写数据块子程序
... ..
SJMP     $

```

C 语言程序清单如下：

```

#include <REG51.H>
#include <intrins.h>                //内部函数定义
#define uchar unsigned char        //将 unsigned char 定义为 uchar
#define SLAW    0xA0               //控制命令
#define SLA     0x40               //2402 地址
#define NUMBYT  0x08               //读出数据个数
unsigned char II_BUF[16]_at_ 0x60; //数据缓冲区
char  bdata Flag ;
sbit  ERR =Flag^0;
sbit  SDA  = P1^2;                  //串行数据线
sbit  SCL  = P1^1;                  //串行时钟线
void main (void)
{
    ... ..
    SP = 0x70;                      //设堆栈指针
    SDA = 1;                        //初始化 AT2402/AT2408 总线
    SCL = 1;
    delay();
    II_read_block(0xA0,0x40,8);      //读 2402 中 8 个数据到缓冲区
    II_write_block(0xAC,0x20,16);    //写入 2408 中 16 个数据
    ... ..
    while (1);
}

```

本程序利用了前面介绍的 I²C 总线的通用模拟软件包，因而程序编写并不复杂，也比较容易懂，以后类似功能的芯片均可以直接调用通用模拟软件包。

9.6 扩展数/模转换器

在单片机控制的实时闭环系统中，有些被控对象需用模拟电压量来控制。此时，就需要把单片机运算处理的结果（数字量）转换为相应的模拟量，以便操纵控制对象。这一过程即为“数/模转换”D/A（Digit to Analog）。能实现 D/A 转换的器件称为 D/A 转换器或 DAC，

为实现数/模转换功能单片机需要扩展 D/A 转换器。

9.6.1 数/模转换器简介

数/模转换器的输出是由数字输入和参考电压组合进行控制的。数/模转换器种类比较多，但其转换原理基本相同，详见 9.6.2。其主要不同点表现在以下 3 个方面。

1. D/A 转换器的输出形式

D/A 转换器的输出可以是电压也可以是电流，多数是电流。电流输出的 D/A 转换器在输出端要加一个运算放大器构成的 I-V 转换电路，将电流输出转换为电压输出。

2. D/A 转换器的输入形式

D/A 转换器的输入应该是一个数字量信号，其输入端通常是与单片机相连，由于数/模转换需要一定时间，为使数字量稳定，单片机的输出脚需要有锁存功能（例如 P1、P2 口），或者 D/A 转换器上需有锁存功能。

3. D/A 转换器的接口形式

单片机与 D/A 转换器的连接，早期多采用并行传输接口，现在除了并行接口外，还增加了串行口输出形式，目前多采用较为流行的 SPI 串行接口。在选择单片 D/A 转换器时，要根据系统结构考虑单片机与 D/A 转换器的接口形式。

D/A 转换器种类比较多，新型的 D/A 转换器开始向更高的位数和转换速度上转变。实际应用时选择什么特点的 D/A 转换器要根据具体需要和性价比确定。

目前有些单片机上也集成了 D/A 模块，通常其转换位数可以达到 10 位，如果转换速度可以达到要求，则可以考虑选择采用集成 D/A 模块的单片机芯片。

9.6.2 数/模转换电路原理

D/A 转换是将数字量信号转换成与此数值成正比的模拟量。如前所述，一个二进制数是由各位代码组合起来的，每位代码在二进制数中的位置代表一定的权。为了将数字量转换成模拟量，应将每一位代码按权大小转换成相应的模拟输出分量，然后根据叠加原理将各代码对应的模拟输出分量相加，其总和就是与数字量成正比的模拟量，由此完成 D/A 转换。

为实现上述 D/A 转换，需要使用解码网络，解码网络的主要形式有二进制权电阻解码网络和 T 型电阻解码网络。

实际应用的 D/A 转换器多数都采用 T 形电阻网络。由于它所采用的电阻阻值小，具有简单、直观、转换速度快、转换误差小等优点，因而本节仅介绍 T 形电阻网络 D/A 转换法。图 9-18 即为其结构原理图。包括一个 4 位切换开关、4 路 R-2R 电阻网络、一个运算放大器和一个比例电阻 R_F 。

T 形电阻网络整个电路是由相同的电路环节所组成的，每节有两个电阻（R、2R），一个开关，相当于二进制数的一位，开关由该位的代码所控制。由于电阻接成 T 形，故称 T 形解码网络。此电路采用了分流原理实现对输入位数字量的转换。图中无论从哪一个 R-2R 节点

向上或向下看，等效电阻都是 $2R$ 。从 $d_0 \sim d_3$ 看进去的等效输入电阻都是 $3R$ ，于是每一开关流入的电流 I 可以看做相等，即 $I = V_R/3R$ 。这样由开关 $d_0 \sim d_3$ 流入运算放大器的电流自上而下以 $\frac{1}{2}$ 系数逐渐递减，依次为 $\frac{1}{2}I$ 、 $\frac{1}{4}I$ 、 $\frac{1}{8}I$ 、 $\frac{1}{16}I$ 。设 $d_3d_2d_1d_0$ 为输入的二进制数字量，于是输出的电压值为：

$$\begin{aligned} V_0 &= -R_F \sum I_i = -(R_F \times V_R/3R) \times (d_3 \times 2^{-1} + d_2 \times 2^{-2} + d_1 \times 2^{-3} + d_0 \times 2^{-4}) \\ &= -[(R_F \times V_R/3R) \times 2^{-4}] \times (d_3 \times 2^3 + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0) \end{aligned}$$

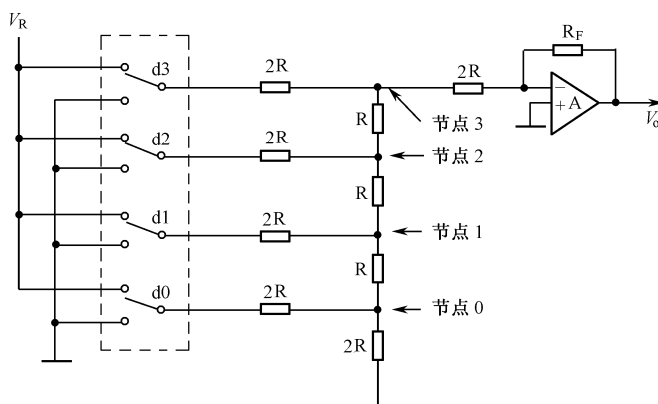


图 9-18 T 形电阻网络 D/A 转换原理图

式中 $d_0 \sim d_3$ 取值为 0 或 1，0 表示切换开关与地相连，1 表示切换开关与参考电压 V_R 接通，该位有电流输入。这就完成了由二进制数到模拟量电压信号的转换。由此公式可以看出， V_R 与 V_0 的电压符号正好相反，即要使输出电压 V_0 为正，则 V_R 必须为负。

9.6.3 D/A 转换器的主要技术指标

1. D/A 建立时间 (Setting Time)

D/A 建立时间是描述转换速率快慢的一个重要参数，是指 D/A 转换器输入数字量为满刻度值（二进制各位全为 1）时，从输入加上，到模拟量电压输出达到满刻度值或满刻度值的某一百分比（如 99%）所需的时间，也可称为输入 D/A 转换速度（Conversion Rate）。不同类型的 D/A 建立时间多数是不同的，但一般均在几十纳秒到几百微秒的范围内。

2. D/A 转换精度 (Accuracy)

精度参数用于表明 D/A 转换的精确程度，一般用误差大小表示。通常以满刻度电压（满量程电压） V_{FS} 的百分数形式给出。例如，精确度为 $\pm 0.1\%$ 指的是，最大误差为 V_{FS} 的 $\pm 0.1\%$ ，如果 V_{FS} 为 5V，则最大误差为 $\pm 5mV$ 。

3. 分辨率 (Resolution)

分辨率表示对输入的最小数字量信号的分辨能力，即当输入数字量最低位（LSB）产生一次变化时，所对应输出模拟量的变化量。它与输入数字量的位数有关，如果数字量的位数为 n ，则 D/A 转换器的分辨率为 2^{-n} 。显然，在 D/A 输出满量程电压相同的情况下，位数越

多，分辨率就越高。通常以其二进制位数表示分辨率。

需要注意的是，精度和分辨率是两个不同的概念。精度取决于构成转换器的各个部件的误差和稳定性，分辨率取决于转换器的位数。

9.6.4 并行 D/A 转换器

目前，D/A 转换器有很多现成的集成电路芯片，对应用设计人员来讲，只需要掌握典型的 DAC 集成电路性能及其与单片机之间接口的基本知识，就可以根据应用系统的要求，合理选取 DAC 集成电路芯片，并配置适当的接口电路。

DAC0832 是较典型的一种 8 位并行 D/A 转换器，其转换时间为 $1\mu\text{s}$ ，工作电压为 $+5\sim+15\text{V}$ ，基准电压为 $\pm 10\text{V}$ 。

1. DAC0832 的逻辑结构和引脚

DAC0832 的引脚图和逻辑框图分别如图 9-19 和图 9-20 所示。由图 9-20 可见各引脚的作用与逻辑关系。

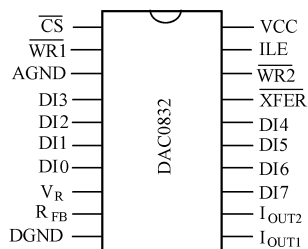


图 9-19 DAC0832 引脚图

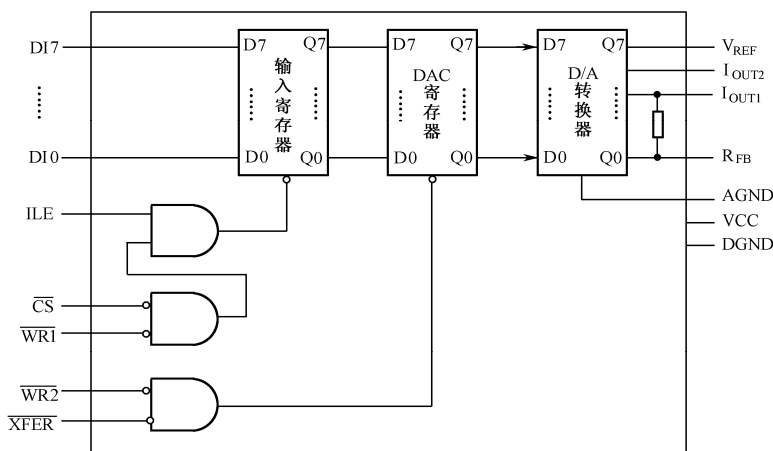


图 9-20 DAC0832 逻辑结构框图

DAC0832 芯片为 20 脚双列直插式封装，各引脚功能如下。

- DI0~DI7：数据输入端，TTL 电平，有效时间大于 90ns。
- ILE：数据锁存允许控制信号输入端，高电平有效。
- $\overline{\text{CS}}$ ：片选信号输入端，低电平有效。
- $\overline{\text{WR1}}$ ：输入寄存器的写选通输入端，负脉冲有效（脉冲宽度应大于 500ns）。当 $\overline{\text{CS}}$ 为“0”，ILE 为“1”， $\overline{\text{WR1}}$ 有效时，DI0~DI7 状态被锁存到输入寄存器。
- $\overline{\text{XFER}}$ ：数据传输控制信号输入端，低电平有效。
- $\overline{\text{WR2}}$ ：DAC 寄存器写选通输入端，负脉冲（脉冲宽度应大于 500ns）有效。当 $\overline{\text{XFER}}$ 为“0”且 $\overline{\text{WR2}}$ 有效时，输入寄存器的状态被传送到 DAC 寄存器中。
- IOUT1：电流输出 1 端，当输入全为“1”时，IOUT1 值最大，输入全为“0”时最小。
- IOUT2：电流输出 2 端，其值和 IOUT1 值之和为一常数。

- R_{FB} : 反馈电阻端, 芯片内部此端与 I_{OUT1} 之间已接有一个 $15k\Omega$ 的电阻。
- V_{CC} : 电源电压端, 范围为 $+5\sim+15V$ 。
- V_{REF} : 基准电压输入端, V_{REF} 范围为 $-10\sim+10V$ 。此端电压决定 D/A 输出电压的精度和稳定度。如果 V_{REF} 接 $+10V$, 则输出电压范围为 $0\sim-10V$; 如果 V_{REF} 接 $-5V$, 则输出电压范围为 $+5\sim 0V$ 。
- $AGND$: 模拟地, 为模拟信号和基准电源的参考地。
- $DGND$: 数字地, 为工作电源地和数字逻辑地, 此地线与 $AGND$ 地最好在电源处一点共地。

DAC0832 是电流型输出, 应用时需外接运算放大器使之成为电压型输出。

2. DAC0832 的应用

根据对 DAC0832 的输入寄存器和 DAC 寄存器的不同控制方法, 可有 3 种工作方式: 单缓冲方式、双缓冲方式和直通方式。下面介绍常用的第一种方式的接口及应用。

单缓冲方式适用于只有一路模拟量输出或几路模拟量非同步输出的情形。在这种方式下, 将两级寄存器的控制信号并接, 输入数据在控制信号作用下, 直接送入 DAC 寄存器中。也可以采用把 $\overline{WR2}$ 、 \overline{XFER} 这两个信号固定接地的方法。图 9-21 为 0832 在此方式下与 AT89S51 的连接方法。

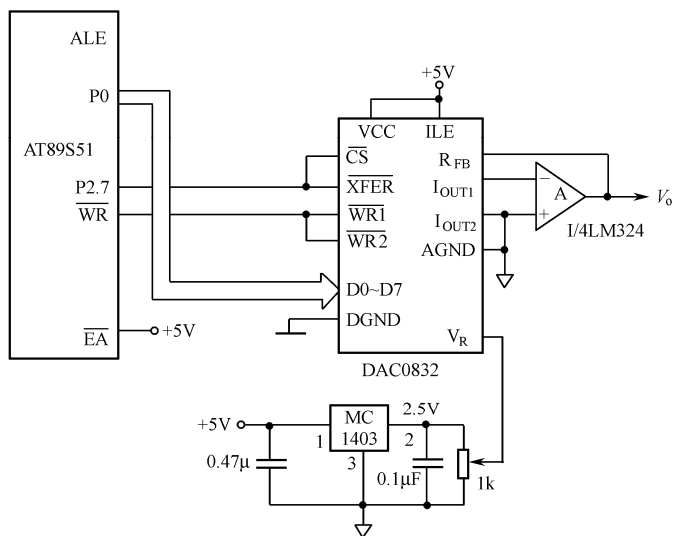


图 9-21 0832 按单缓冲方式与 AT89S51 连接图

在图 9-21 中, ILE 接 $+5V$, 片选信号 \overline{CS} 和传送信号 \overline{XFER} 都连到 $P2.7$, 这样输入寄存器和 DAC 寄存器的地址都是 $7FFFH$ 。写选通线 $\overline{WR1}$ 和 $\overline{WR2}$ 都和 AT89S51 的写信号 \overline{WR} 连接, CPU 对 0832 执行一次写操作, 则把一个数据直接写入 DAC 寄存器, 0832 的输出模拟信号随之相应变化。由于 0832 是电流型输出, 所以在电路中采用运算放大器 LM324 实现 I/V 转换。

D/A 转换器的基准电压 V_R 取自基准电源 MC1403 的输出分压。MC1403 称带隙基准电源, 其最大优点是高精度低温漂, 输入电压在 $4.5\sim15V$ 之间, 输出电压在 $2.5V$ 左右, 最大输出电流为 $10mA$ 。

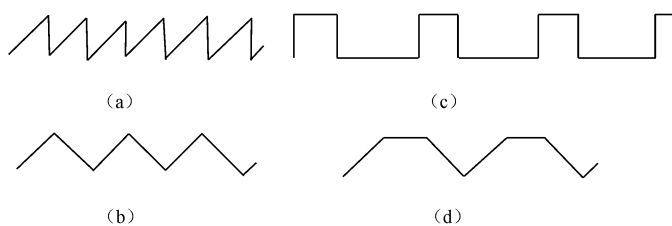


图 9-22 D/A 转换器输出的各种波形

根据图 9-21 的电路可以编出多种波形输出的 D/A 转换程序，例如要得到图 9-22 的波形，C 语言程序如下：

锯齿波：

```
#include <reg52.h>
#include <absacc.h>
#define DAC0832 XBYTE[0x7FFF]           //选中 DAC0832
unsigned char i,DAC_value;
void main(void)
{
    unsigned char ;
    DAC_value=0;
    while(1)
    {
        DAC0832 = DAC_value;             //向 DAC 送出数据
        DAC_value++;                       //输出值增 1
    };
    while (1);
}
```

电路图中运算放大器为反相输入，因此程序中值增加时，显示波形的幅度会减小。

三角波：

```
DAC_value=0;
while(1)
{
    UP: DAC0832 = DAC_value;
        DAC_value++;
        if(DAC_value !=0) goto UP;        //比较是否到达最大值
        DAC_value--;
    DOWN:
        DAC_value--;
        DAC0832 = DAC_value;
        if(DAC_value !=0) goto DOWN; //比较是否到达最小值
        else goto UP;
};
```

矩形波：

```
while(1)
{
    DAC0832 = dataH;                       //置输出矩形波上限
    Delay1();
    DAC0832 = dataL;                       //置输出矩形波下限
```

```

        Delay2();
    };
梯形波:
    while(1)
    {
        DAC_value = dataL-1;
    L1:
        DAC_value++;
        DAC0832 = DAC_value;           //输出值不断加大
        if (DAC_value != dataH) goto L1; //比较是否到达最大值
        delay();                       //上限延时
    L2:
        DAC_value--;
        DAC0832 = DAC_value;           //输出值不断减小
        if (DAC_value != dataL) goto L2; //比较是否到达最小值
    };

```

并行 D/A 转换器转换速度较高，但要占用较多的 I/O 线，随着单片机串行接口技术的发展，出现了越来越多的串行 D/A 转换芯片，篇幅关系，本节不予介绍，可参考文献 1。

9.7 扩展模/数转换器

在实际工作中经常会遇到需要对温度、压力等物理量进行测量的要求，此时首先需要通过某种传感器把物理量转换为相应的模拟电压（或者电流）量，简称模拟量，该模拟量能反映时间、数值连续变化的物理量。单片机只能识别数字量，所以要经过模拟量到数字量的转换（简称 A/D 转换），单片机才能接收该物理量信息，显然要实现此功能单片机需要扩展模/数转换器。能实现模/数转换的芯片称为 A/D（Analog to Digit）转换器或 ADC。

9.7.1 模/数转换器简介

A/D 转换电路种类很多，根据转换原理可以分为逐次逼近式、双积分式、并行式、计数器式、串并式等，目前应用较多的是前 2 种。

A/D 转换器根据不同的接口方式可分为并行接口和串行接口两种。串行接口连线简单，成本低，但速度慢，并行接口连线复杂，成本高，但速度快。

A/D 转换器根据输出数字量的位数有 8、10、12、16 位等，对于输出 BCD 码的 A/D 转换器有 3 位半、4 位半、5 位半等多种，一般位数越高价格也越高。

A/D 转换器根据转换速度可大致分为超高速（转换时间 $\leq 1\text{ns}$ ）、高速（转换时间 $\leq 1\mu\text{s}$ ）、中速（转换时间 $\leq 1\text{ms}$ ）、低速（转换时间 $\leq 1\text{s}$ ）等几种不同转换速度的芯片。通常速度越高的 A/D 转换器价格也越高。

实际应用时选择什么特点的 A/D 转换器要根据具体需要和性价比确定。

在此需要提醒读者注意的是虽然目前有些单片机上集成了 A/D 模块，但它的精度、测量范围、速度、性能和稳定性等不如专门定制的 A/D 芯片，所以只有在对模/数转换精度等要求不高的场合可以考虑选择采用集成 A/D 模块的单片机芯片。

9.7.2 模/数转换器的主要技术指标

A/D 转换器所涉及的主要技术指标包括如下几项。

1. 转换时间和转换频率

A/D 转换器完成一次模拟量变换为数字量所需时间即为 A/D 转换时间。通常，转换频率是转换时间的倒数，它反映了 A/D 转换器的实时性能。

2. 分辨率

A/D 转换器的分辨率是指转换器对输入电压微小变化响应能力的度量。习惯上以输出的二进制位数或者 BCD 码位数表示。例如 A/D 转换器 AD574A 的分辨率为 12 位，即该转换器的输出数据可以用 2^{12} 个二进制数进行量化，其分辨率为 $V_{FS}/2^{12}$ （是输入电压满量程值）。如果用百分数来表示分辨率时，其分辨率为：

$$1/2^{12} \times 100\% = (1/4096) \times 100\% \approx 0.024414\% \approx 0.0244\%$$

当转换位数相同、而输入电压的满量程值 V_{FS} 不同时，则可分辨的最小电压值不同。例如分辨率为 12 位， $V_{FS}=5V$ 时，可分辨的最小电压是 1.22mV；而 $V_{FS}=10V$ 时，可分辨的最小电压是 2.44mV。当输入电压的变化低于此值时，转换器不能分辨。例如，10~9.998V 之间所转换的数字量均为 4095。

3. 转换精度

A/D 转换器转换精度，反映了一个实际 A/D 转换器在量化值上与一个理想 A/D 转换器进行模/数转换的差值，可表示成绝对误差或相对误差，与一般测试仪表的定义相似。

对于不同 A/D 转换器生产厂家，其产品精度指标表达方式不完全相同，有的给出综合误差指标，有的给出分项误差指标。通常给出的分项误差指标有：非线性误差、零点误差、增益误差等。必须指出，A/D 转换器的精度所对应的误差指标是不包括量化误差的。

9.7.3 逐次逼近式 A/D 转换器

逐次逼近式 A/D 转换器在精度、速度和价格上都适中，是目前种类最多、应用最广的 A/D 转换器。

1. 逐次逼近式 A/D 转换原理

A/D 转换过程主要包括采样、量化与编码。采样是使模拟信号在时间上离散化；量化就是用一个基本的计量单位（量化电平）使模拟量变为一个整数的数字量；编码是把已经量化的模拟量（它是量化电平的整数倍）用二进制数码、BCD 码或其他数码表示。总之，量化与编码就是把采样后所得到的离散幅值经过舍入的方法变换为与输入量成比例的二进制数。逐次逼近型的转换原理即“逐位比较”，其过程类似于用砝码在天平上称物体重量，其方法是用一个二进制数作为计量单位与模拟量比较，把模拟量变为计量单位的整数倍，略去小于计量单位的模拟量的过程，这样所得到的整数量即为数字量。显然，计量单位越小，量化的误差也越小。图 9-23 为一个 N 位的逐次逼近式 A/D 转换器原理图。它由 N 位寄存器、D/A 转换

器、比较器和控制逻辑等部分组成， N 位寄存器代表 N 位二进制数码。

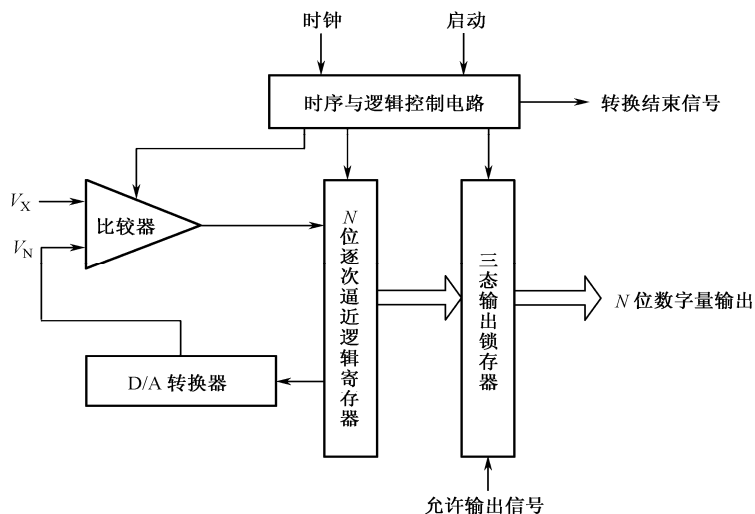


图 9-23 逐次逼近式 A/D 转换器原理图

当模拟量 V_X 送入比较器后，启动信号通过控制逻辑电路启动 A/D 开始转换，首先，置 N 位寄存器最高位 (D_{n-1}) 为“1”，其余位清“0”（相当于先放一个最重的砝码）， N 位寄存器的内容经 D/A 转换后得到整个量程一半的模拟电压 V_N ，与输入电压 V_X 比较。若 $V_X \geq V_N$ 时，则保留 $D_{n-1}=1$ ；若 $V_X < V_N$ 时，则 D_{n-1} 位清 0。然后，控制逻辑使寄存器下一位 (D_{n-2}) 置“1”，与上次的结果一起经 D/A 转换后与 V_X 比较，重复上述过程，直至判别出 D_0 位取“1”还是“0”为止。这样经过 N 次比较后， N 位寄存器的内容就是转换后的数字量数据，此时控制逻辑电路发出转换结束信号，应答后，经输出锁存器读出数据，整个转换过程就是这样一个逐次比较逼近的过程。

2. 应用举例

常用的逐次逼近法 A/D 器件有并行输出和串行输出的很多种，本节仅以最简单和经典的有 8 路输入通道的 ADC0809 为例介绍。

(1) ADC0809 的引脚与功能

ADC0809 的引脚示于图 9-24。各引脚定义与功能如下：

- IN0~IN7：8 路模拟量的输入端。
- D0~D7：A/D 转换后的数据输出端，为三态可控输出，可直接与计算机数据线相连。
- A、B、C：模拟通道地址选择端，A 为低位，C 为高位，C、B、A 的组合编码为 000~111，分别选择 IN0~IN7 通道。例如当 C、B、A=0、1、0 时选择 IN2 通道。
- $V_R(+)$ 、 $V_R(-)$ ：基准参考电压端，决定了输入模拟量的量程范围。
- CLK：为时钟信号输入端，决定 A/D 转换的速度，时钟信号输入范围为 50~800kHz。
- ALE：地址锁存允许信号，高电平有效。当此信号有效时，A、B、C 三位地址信号被锁存，译码选通对应模拟通道。
- SC：为启动转换信号，正脉冲有效。通常与系统 \overline{WR} 信号相连，控制启动 A/D 转换。
- EOC：转换结束信号，高电平有效，表示一次 A/D 转换已完成，可作为中断触发信号，也可用程序查询的方法检测转换是否结束。

- OE: 输出允许信号, 高电平有效, 可与系统读选通信号 \overline{RD} 相连。当单片机发出此信号时, ADC0809 的三态门被打开, 此时可通过数据线读到正确的转换结果。

ADC0809 的原理结构框图如图 9-25 所示。多路模拟开关允许 8 路模拟量分时输入, 8 路模拟开关的切换由地址锁存和译码电路控制, 由加到 A、B、C 引脚端的编码确定所选择的通道, 通过 ALE 锁存。8 路模拟量共用一个 A/D 转换器进行转换。A/D 转换结果通过三态输出锁存器输出, 所以在系统连接时允许直接与单片机的数据总线相连。

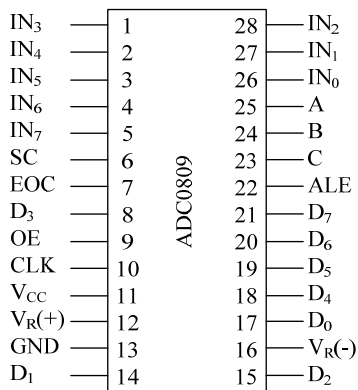


图 9-24 ADC0809 引脚图

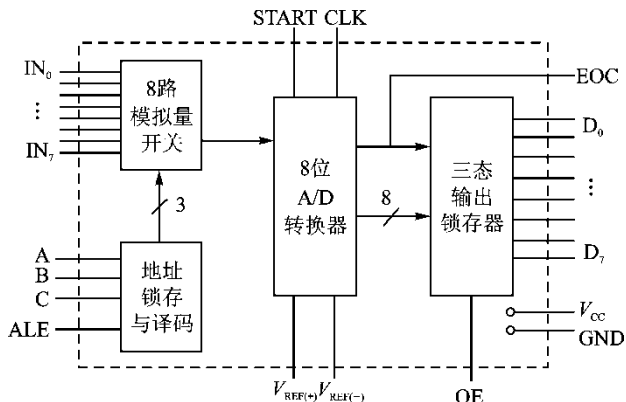


图 9-25 ADC0809 原理结构框图

(2) ADC0809 与单片机的接口及编程

图 9-26 是 ADC0809 与 AT89S51 的连接示意图, 8 路输入模拟量的变化范围是 0~5V。ADC 0809 的 EOC 用做外部中断请求源, 用中断方式读取 A/D 转换结果。AT89S51 通过地址线 P2.0 和读、写线 \overline{RD} 、 \overline{WR} 来控制转换器的模拟输入通道地址锁存、启动和转换结果的输出。模拟输入通道地址的译码输入 A、B、C 由 P0.0~P0.2 提供, 经地址锁存输出后与 A、B、C 相接。

参照图 9-26, 举例说明 ADC0809 的应用。要求采用中断方式巡回检测一遍 8 路模拟量输入, 将读数依次存放在片内数据存储单元 40H~47H 单元, 检测一遍后即可退出中断。其初始化程序和中断服务程序如下。

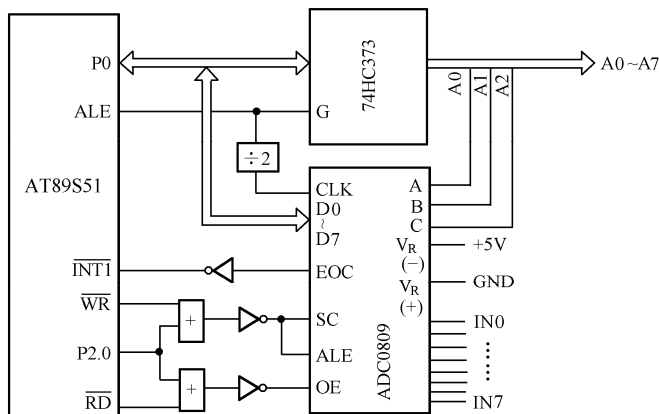


图 9-26 ADC0809 与 AT89S51 的连接示意图

初始化汇编语言程序清单：

```

MAIN:      :
            MOV    10H,    # 040H      ; 数据暂存区首址存入第 2 组寄存器的 R0
            MOV    12H,    # 08H      ; 8 路计数初值存入第 2 组寄存器的 R2
            SETB   IT1          ; 置脉冲触发方式
            MOV    IE      # 10000100B ; CPU 打开中断, 允许  $\overline{\text{INT1}}$  申请中断
            MOV    DPTR,    # 0FEF8H   ; 指向 ADC0809 首地址
            MOVBX   @DPTR, A          ; 启动 A/D 转换
            :
            SJMP    $                ; 等中断
中断服务程序: PUSH    ACC
            SETB   RS1              ; 选择第 2 组寄存器
            CLR    RS0
            MOVBX   A,    @DPTR      ; 读数
            MOVBX   @R0,    A        ; 存数
            INC     DPTR              ; 更新通道
            INC     R0                ; 更新暂存单元
            MOVBX   @DPTR, A        ; 启动 A/D 转换
            DJNZ    R2,    BACK       ; 是否检测完 8 路? 未完转中断返回
            CLR     EA                ; 结束, 关中断
BACK:      CLR     RS1              ; 恢复第 0 组寄存器
            POP     ACC
            RETI

```

为使问题简化, 在本例题中假设 DPTR 仅用于 A/D 转换的地址, 没有其他用途。

初始化 C 语言程序清单：

```

#include <REG51.H>
#include <absacc.h>
#define uchar unsigned char
uchar adbuf[8] _at_ 0x40;
uchar channel ;
void main (void)
{
    IT1 = 1;                //置脉冲触发方式
    IE = 0x84;              //CPU 打开中断, 允许 申请中断
    channel = 0;            //8 路计数初值
    XBYTE[0xFEf8] = 0;      //启动 A/D 转换
    while(1)
    {
    }
}
// 外部中断处理程序
void int_int1(void) interrupt 2 using 2 //baud
{
    uchar temp;
    temp = XBYTE[0xFEf8];    //读 A/D 转换值
    adbuf[channel] = temp;   //存数
    channel++;               //通道更新
    if (channel>=8) EA = 0;   //检测完 8 路关中断
}

```

```

else XBYTE[0xFE8+channel] = 0;    //启动 A/D 转换
}

```

9.7.4 双积分 A/D 转换器

双积分 A/D 转换器具有精度高、抗干扰性好（对周期变化的干扰信号积分为零）、价格低廉等优点，经常应用于对速度要求不高，且工频干扰比较严重的场合。

目前双积分 A/D 转换芯片种类很多，常用的 BCD 码输出的有 MC14433（3 位半，国产型号为 5G14433）、ICL7135（4 位半），二进制码输出的有 ICL7109（12 位）、ICL7104（16 位）等。本节介绍较典型的 MC14433。

MC14433 是 3 位半 A/D 转换器。其最大输入电压为 199.9mV 和 1.999V 两挡，具有 $\pm 1/1999$ 的分辨率（相当于 11 位二进制数），但转换速度较慢（约每秒 1~10 次）。

1. MC14433 的引脚及结构

MC14433 的引脚图和逻辑框图分别如图 9-27 和图 9-28 所示。

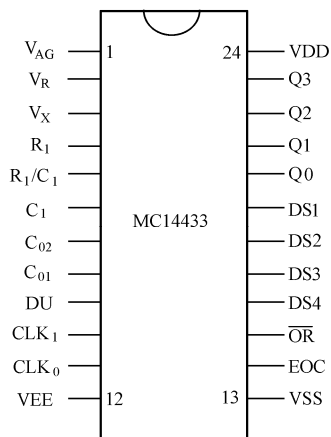


图 9-27 MC14433 的引脚图

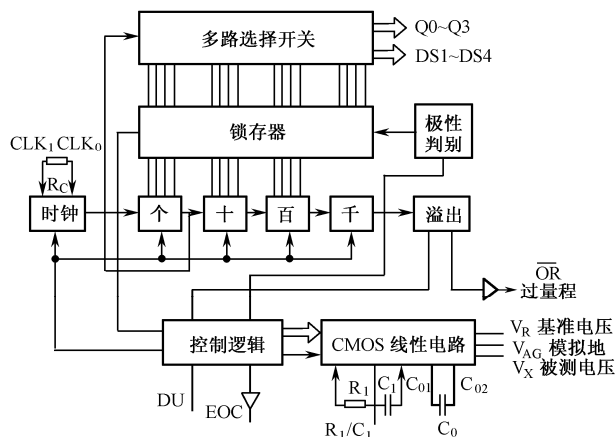


图 9-28 MC14433 的逻辑框图

MC14433 各引脚名称及功能如下。

VDD：主电源，+5V。

VEE：模拟部分的负电源，-5V。

VSS：数字地。

V_R：基准电压输入脚，V_R 的值为 200mV 或 2V。

V_X：被测电压输入脚，最大为 199.9mV 或 1.999V。

V_{AG}：V_R 和 V_X 的地（模拟地）。

R₁：积分电阻输入脚，当 V_X 量程为 2V 时，R₁ 取 470kΩ，当 V_X 量程为 200mV 时，R₁ 取 27kΩ。

C₁：积分电容输入脚，C₁ 端外接电容一般取 0.1μF。

R₁/C₁：R₁ 和 C₁ 的公共连接端。

C₀₁、C₀₂：接失调补偿电容 C₀，C₀ 的值约 0.1μF。

CLK₁、CLK₀：外接振荡器时钟频率调节电阻 R_C，R_C 的典型值为 470kΩ。

EOC：转换结束状态输出脚，EOC 是一个宽为 0.5 个时钟周期的正脉冲。

DU：更新转换控制信号输入脚，DU 若与 EOC 相连，则每次 A/D 转换结束后自动启动新的转换。

$\overline{\text{OR}}$ ：过量程状态信号输出脚，低电平有效，当 $|V_X| > V_R$ 时， $\overline{\text{OR}}$ 有效。

DS4~DS1：分别是个、十、百、千位的选通脉冲输出脚，这 4 种选通脉冲均为宽 18 个时钟周期的正脉冲，它们之间的间隔时间为两个时钟周期。

Q3~Q0：BCD 码数据输出脚，动态地输出千位、百位、十位、个位值，即

DS4 有效时，Q3~Q0 表示的是个位值（0~9）。

DS3 有效时，Q3~Q0 表示的是十位值（0~9）。

DS2 有效时，Q3~Q0 表示的是百位值（0~9）。

DS1 有效时，Q3 表示的是千位值（0 或 1）；Q2 表示转换极性（0 为负，1 为正）；Q1 无意义；Q0 为 1 时，如果 Q3 为 0 表示过量程（太大），Q3 为 1 表示欠量程（太小）。

由图 9-28 可见模拟电路部分有基准电压、模拟电压输入部分。模拟电压输入量程为 199.9mV 或 1.999V，基准电压相应为 200mV 或 2V。

数字电路部分由逻辑控制、BCD 码及输出锁存器、多路开关、时钟以及极性判别、溢出检测等电路组成。MC14433 采用了字位动态扫描 BCD 码输出方式，即千、百、十、个各位 BCD 码轮流地在 Q0~Q3 端输出，同时在 DS1~DS4 端出现同步字位选通信号。

图 9-29 为 MC14433 的输出时序，由图可见 MC14433 的 A/D 转换结果是动态分时输出的 BCD 码，单片机只能通过并行 I/O 接口或者扩展 I/O 接口与其相连。

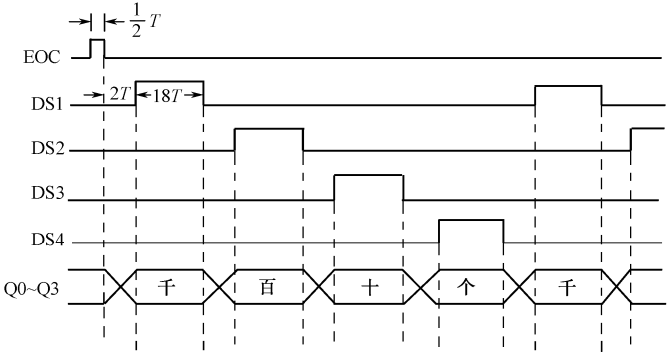


图 9-29 MC14433 的输出时序

表 9-2 为 DS1 选通时 Q0~Q3 表示的输出结果。

表 9-2 DS1 选通时 Q0~Q3 表示的输出结果

DS1	Q3	Q2	Q1	Q0	输出结果状态
1	1	×	×	0	千位数为 0
1	0	×	×	0	千位数为 1
1	×	1	×	0	输出结果为正值
1	×	0	×	0	输出结果为负值
1	0	×	×	1	输入信号过量程
1	1	×	×	1	输入信号欠量程

2. MC14433 与 AT89S51 的接口

图 9-30 为 MC14433 与 AT89S51 的连接图。其中 P1 口作为 MC14433 的 BCD 码扫描输出接口，转换结束信号经非门送入外部中断 1，显然 AT89S51 的外中断 1 应采用外部脉冲触发方式。当 MC14433 转换结束时，EOC 发脉冲向 AT89S51 申请中断。在中断服务程序中，用查询的方式读取 MC14433 的 BCD 码扫描输出值。

3. 程序举例

要求将 A/D 转换结果存放在片内 RAM 的 21H 和 22H 单元中，存放格式如下。

	D7	D6D5	D4	D3~D0		D7~D4	D3~D0
21H	符号		千位	百位	22H	十位	个位

MC14433 上电后，即对外部输入电压信号进行 A/D 转换，由于 EOC 与 DU 端相连，每次转换完毕 EOC 的正脉冲除送至 CPU，同时加到 DU 脚，14433 立即使相应的 BCD 码及相应的选通信号出现在 Q0~Q3 和 DS1~DS3 上。当 AT89S51 的 CPU 打开中断，且允许 $\overline{\text{INT1}}$ 申请中断，并置外部中断为边沿触发方式时，则每次 A/D 转换结束，都将引起中断，可在中断服务程序中处理 A/D 转换结果。程序中用 0F、0C 和 40H 分别表示符号、千位数及过、欠量程标志。

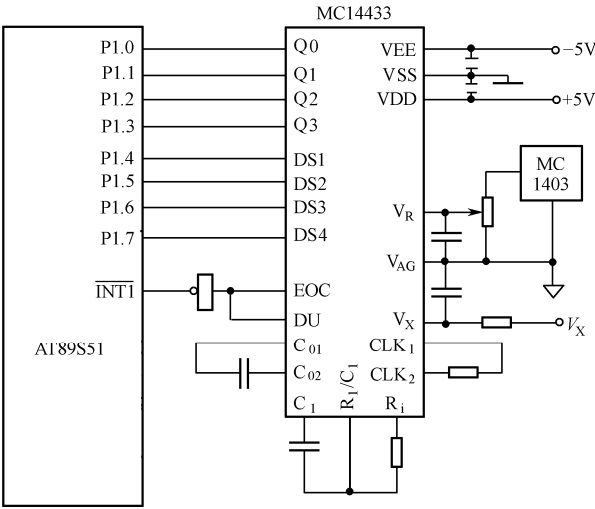


图 9-30 MC14433 与 AT89S51 的连接图

汇编语言程序如下：

初始化程序

```

:
INT1:  SETB   IT1           ; 选择  $\overline{\text{INT1}}$  边沿触发方式
      MOV    IE,    # 84H   ; CPU 打开中断，允许  $\overline{\text{INT1}}$  申请中断
      :
```

外部 $\overline{\text{INT1}}$ 中断服务程序

```

ADD1:  MOV    A,    P1
      JNB    ACC.4,  ADD1   ; 等待 DS1 选通信号
      JB     ACC.0,  ERR    ; 量程错误转 ERR
      JB     ACC.2,  PL1    ; 查结果的符号
```

	SETB	0FH		; 为负数, 符号位置 1
	SJMP	PL2		
PL1:	CLR	0FH		; 为正数, 符号位置 0
PL2:	JB	ACC.3,	PL3	; 查千位数为 0 或 1
	SETB	0CH		; 千位数字 1
	SJMP	PL4		
PL3:	CLR	0CH		; 千位数字 0
PL4:	MOV	A,	P1	
	JNB	ACC.5,	PL4	; 等待百位数选通信号 DS2
	MOV	R0,	# 21H	
	XCHD	A,	@ R0	; 百位数送入 21H 低 4 位
PL5:	MOV	A,	P1	
	JNB	ACC.6,	PL5	; 等待十位数选通信号 DS3
	SWAP	A		; 高低 4 位交换
	INC	R0		; 指向 22H 单元
	MOV	@ R0,	A	; 十位数送入 22H 高 4 位
PL6:	MOV	A,	P1	
	JNB	ACC.7,	PL6	; 等待个位数选通信号 DS4
	XCHD	A,	@R0	; 个位数送入 22H 低 4 位
	RETI			
ERR:	SETB	40H		; 置过、欠量程标志
	RETI			

C 语言程序如下:

```

#include <REG51.H>
#include <absacc.h>
#define uchar unsigned char
uchar channel ;
void main (void)
{
    IT1 = 1;                //置脉冲触发方式
    IE = 0x84;              //CPU 打开中断, 允许 INT1 申请中断
    while(1){ };
}
void int_int1(void) interrupt 2 using 2    //外部中断处理程序
{
    do {} while((P1&0xf0)==0x10);          // DS1 选通信号
        if (P1&0x0e) DBYTE[0x40] = 1;     //量程错误 置 1
        if (P1&0x0b) DBYTE[0x0F] = 1;     //为负数, 符号位置 1
    else DBYTE[0x0F] = 0;
    if (P1&0x07) DBYTE[0x0C] = 1;          //千位数为 1
    else DBYTE[0x0C] = 0;
    do {} while((P1&0xf0)==0x20);          //百位数选通信号 DS2
        DBYTE[0x21] = P1&0x0f;
    do {} while((P1&0xf0)==0x40);          //十位数选通信号 DS3
        DBYTE[0x22] = (P1&0x0f)<<4;
    do {} while((P1&0xf0)==0x80);          //个位数选通信号 DS4
        DBYTE[0x22] += P1&0x0f;
}

```

9.7.5 串行 A/D 转换器

串行 A/D 转换器的输出均为一根串行线，因而其连线简单，现在应用越来越广泛，在此选择美国 TI 公司生产的 TLC549 芯片举例说明串行 A/D 的工作原理及应用。

1. TLC549 各引脚名称及功能

TLC549 的芯片引脚如图 9-31 所示。各引脚名称及功能如下。

- REF+：正基准电压输入端。
- ANALOG IN：模拟信号输入端。
- REF-：负基准电压输入端。
- GND：地。
- $\overline{\text{CS}}$ ：芯片选择输入端。
- DATAOUT：数字量输出端，与 TTL 兼容。
- I/O CLOCK：时钟信号输入端口，串行输入/输出数据定时运行。
- VCC：电源电压（+3V~+6V）。

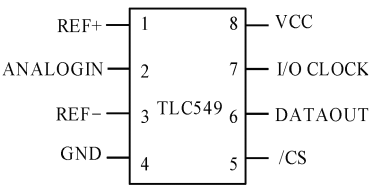


图 9-31 TLC549 的引脚图

2. 基本工作原理

TLC549 是一种 CMOS 单通道 8 位逐次逼近 A/D 转换器，它采用串行方法输出数据，具有 8 位分辨率。TLC549 的 I/O 时钟输入频率可设置到 1.1MHz，A/D 转换的采样速率为 40kHz，输入参考电压为差分式，具有 4MHz 的内部系统时钟。

TLC549 芯片通过 I/O CLOCK 和 $\overline{\text{CS}}$ 这 2 个输入控制信号和 3 态输出信号 DATAOUT 与微处理器或单片机进行串行通信。内部系统时钟和 I/O CLOCK 之间互不影响。TLC549 的片上系统时钟用于驱动“转换”电路，不需要附加外部器件即可使用。

模拟输入电压转换后的数字量值与参考电压有关，如果模拟输入电压比 $V_{\text{ref}+}$ 大，则转换成全 1 (FFH)，而比 $V_{\text{ref}-}$ 小的输入电压转换成全 0。正参考电压 $V_{\text{ref}+}$ 必须比负参考电压 $V_{\text{ref}-}$ 高 1V 以上。

TLC549 是在读出前一次数据后，马上进行电压的采样，并进行 ADC 转换，转换完后就进入保持 (HOLD) 模式，直到再次读取数据时，芯片才会进行下一次的 A/D 转换；即本次读出的数据是前一次的转换值，读操作后就会再启动一次转换。芯片本身没有 A/D 转换结束信号，需要软件延时一段时间等待转换结束。

3. 应用举例

利用图 9-32 所示电路，对串行 A/D TLC549 进行一路模拟量的测量，将读取后转换的数字量存放在内部 RAM 的 30H 单元，然后予以显示，要求这个过程重复进行。

汇编语言程序如下：

```
SCLA   BIT    P1.0           ; 时钟线
SDAA   BIT    P1.1           ; 数据线
CS549  BIT    P1.2           ; 片选线

ORG    0000H
SJMP   MAIN
```

```

        ORG      0100H
MAIN:    ACALL    TLC549      ; 启动第一次 A/D 转换
L1:      LCALL    DIR         ; 调用显示程序（省略），同时延时等待
        ACALL    TLC549      ; 读取上次 ADC 值，并再次启动 AD 转换
        MOV      30H,      A
        SJMP     L1

```

TLC549 为 ADC 转换程序，读取前一次转换值并返回，然后启动下次 ADC 转换。

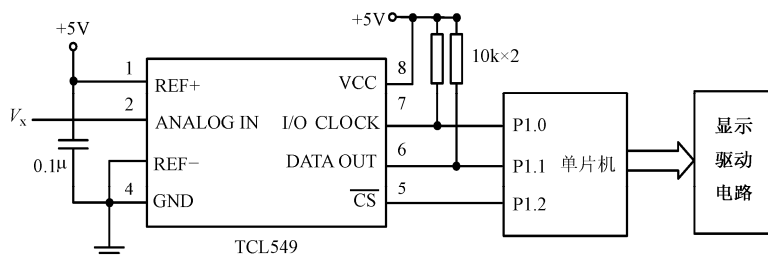


图 9-32 TLC549 例题原理图

汇编语言程序如下：

```

TLC549: CLR      SCLA          ; 启动 A/D 转换
        SETB     SDAA
        CLR      CS549         ; CS 为低，选中 TLC549
        MOV      R7,      #8
LOOP1:  SETB     SCLA
        NOP
        NOP
        MOV      C,      SDAA
        RLC      A
        CLR      SCLA          ; SCLA=0，为读出下一位数据做准备
        NOP
        DJNZ     R7,      LOOP1
        SETB     CS549         ; 禁止 TLC549，再次启动 AD 转换
        RET
        END

```

C 语言程序如下：

```

#include <REG51.H>
#include <intrins.h>
#include <absacc.h>
#define uchar unsigned char
void sendbyte(uchar senddata);
sbit  SCLA  = P1^0;           //串行时钟线
sbit  SDAA  = P1^1;           //串行数据线
sbit  CS549 = P1^2;
uchar ADC549(void)            //AD 转换程序
{
    uchar i,advalue;
    advalue = 0;
    SCLA  = 0;

```

```

    SDAA = 1;
    CS549 = 0;
    for (i=0;i<8;i++)
    {
        SCLA = 1;
        _nop_();
        _nop_();
        if (SDAA) advalue +=1;
        advalue = advalue<<1;
        SCLA = 0;
        _nop_();
    }
    CS549 = 1;
    return(advalue);
}
void dis_pro(void) // 显示程序略
{}
void main (void)
{
    while(1)
    {
        DBYTE[0x30] = ADC549(); //启动第一次 A/D 转换;
        dis_pro();
    }
}

```

随着单片机内存容量的不断增大及内部功能的不断完善，例如增加了更多的 I/O 口，扩大了 RAM、ROM 容量，增加了 EEPROM，增加了 A/D、D/A 等功能模块，使单片机“单片”应用的情况更加普遍，采用系统扩展增加单片机功能的方法将逐渐减少，这也是单片机发展的一种趋势。

思考与练习

1. 在 AT89S51 单片机的扩展系统中，程序存储器和数据存储器共用 16 位地址线和 8 位数据线，为什么两个存储空间不会发生冲突？
2. 为什么当 P2 作为扩展存储器的高 8 位地址后，不再适宜作通用 I/O 口了？
3. 以 AT89S51 作为主机，扩展 2 片 6264RAM 存储器芯片，设计硬件布线图。
4. 根据图 9-8 线路设计程序，其功能是按下 K0~K3 按键后，对应 LED4~LED7 发光，按下 K4~K7 时，对应 LED0~LED3 发光。
5. 请利用译码器 74HC138 设计一个译码电路，分别选中两片 29C256 和两片 62256，且列出各芯片所占的地址空间范围。
6. 说明 I²C、SPI 两种串行总线的传输方法，与并行总线相比各有什么优缺点？
7. 如果 AT24C08 中某单元的实际地址为 310H，则其写芯片地址的命令字怎样设置？
8. AT89S51 单片机扩展了一片 AT24C04，要求把从 AT24C04 的 50H 单元开始的 16 个字节数据存入 AT89S51 单片机片内 RAM 的 60H 开始的单元中。

9. 什么是 D/A 转换器? 简述 T 形电阻网络转换器的工作原理。
10. DAC0832 与 AT89S51 单片机连接时有哪些控制信号? 其作用是什么?
11. 在一个晶振为 12MHz 的 AT89S51 应用系统中, 接有一片 DAC0832, 它的地址为 7FFFH, 输出电压为 0~5V。请画出有关逻辑框图, 并编写一个程序, 使其运行后, DAC 能输出一个矩形波, 波形占空比为 1:4。高电平时电压为 2.5V, 低电平时为 1.25V。
12. 试说明逐次逼近 A/D 转换器的工作原理。
13. 在一个晶振为 12MHz 的 AT89S51 系统中, 接有一片 A/D 器件 ADC0809, 它的地址为 0EFF8H~0FFFFH。试画出有关逻辑框图, 并编写定时采样 0~3 通道的程序。设采样频率为 2ms 一次, 每个通道采 50 个数。把所采的数按 0、1、2、3 通道的顺序存放在以 3000H 为首址的片外数据存储区中。

第 10 章 接 口 技 术

单片机广泛地应用于工业测控、智能化仪器仪表和家电产品中，由于实际工作需要和用户的不同要求，单片机应用系统常常需要配接键盘、显示器、打印机以及功率器件等外设，接口技术就是解决单片机与外设联系的技术。在前面已经介绍了单片机片内的 I/O 端口及定时器、串行口等片上功能部件及系统的扩展芯片等。本章将从一些常用的外设接口电路入手，帮助读者了解单片机与外设的接口技术。

10.1 键 盘 接 口

在单片机应用系统中，通常都要有人-机对话功能。它包括人对应用系统的状态干预、数据的输入以及应用系统向人报告运行状态与运行结果等。

对于需要人工干预的单片机应用系统，键盘就成为人-机联系的必要手段，此时需配置适当的键盘输入设备。键盘电路的设计应使 CPU 不仅能识别是否有键按下，还要能识别是哪一个键按下，而且能把此键所代表的信息翻译成单片机所能接收的形式，例如 ASCII 码或其他预先约定的编码。

单片机常用的键盘有全编码键盘和非编码键盘两种。全编码键盘能够由硬件逻辑自动提供与被按键对应的编码。此外，还具有去抖动和多键、窜键保护电路，这种键盘使用方便，但需要专门的硬件电路，价格较贵，一般的单片机应用系统较少采用。

非编码键盘分为独立式键盘和矩阵式键盘，硬件上此类键盘只简单地提供通、断两种状态，其他工作都靠软件来完成，由于其经济实用，目前在单片机应用系统中多采用这种办法。本节将着重介绍非编码键盘接口。

10.1.1 键盘工作原理

在单片机应用系统中，除了复位键有专门的复位电路以及专一的复位功能外，其他的按键都是以开关状态来设置控制功能或输入数据的。因此，这些按键只是简单的电平输入。键信息输入是与软件功能密切相关的过程。对某些应用系统，例如智能仪表，键输入程序是整个应用程序的重要组成部分。

1. 键输入原理

键盘中每个按键都是一个常开的开关电路，当所设置的功能键或数字键按下时，则处于闭合状态，对于一组键或一个键盘，需要通过接口电路与单片机相连，以便把键的开关状态通知单片机。单片机可以采用查询或中断方式了解有无键输入并检查是哪一个键被按下，并通过转移指令转入执行该键的功能程序，执行完又返回到原始状态。

2. 键输入接口与软件应解决的问题

键盘输入接口与软件应可靠而快速地完成键信息输入与执行键功能任务。为此，应解决下列问题。

(1) 键开关状态的可靠输入

目前，无论是按键或键盘大部分都是利用机械触点的合、断作用。机械触点在闭合及断开瞬间由于弹性作用的影响，均有抖动过程，从而使电压信号也出现抖动。图 10-1 所示为按键时电压的抖动情况示意图，当按键 K 断开时 a 点为高电平，闭合时为低电平，抖动时间长短与开关的机械特性有关，一般为 5~10ms。

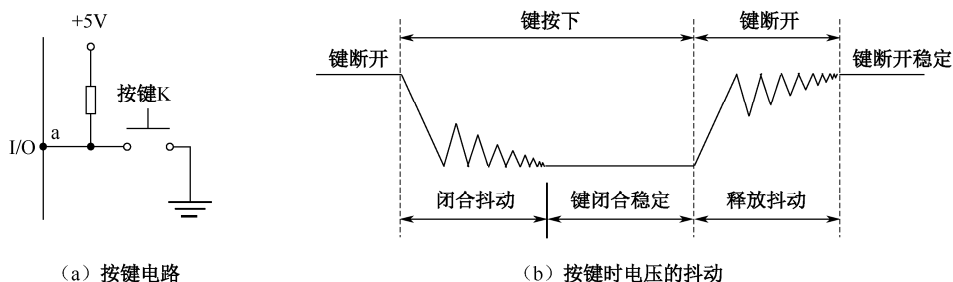


图 10-1 键闭合及断开时的电压波动

按键的稳定闭合时间，由操作人员的按键动作所确定，一般为十分之几秒至几秒时间。为了保证 CPU 对键的一次闭合仅做一次键输入处理，必须去除抖动影响。

通常去抖动影响的方法有硬、软件两种。在硬件上是采取在键输出端加 R-S 触发器或单稳态电路构成去抖动电路。在软件上采取的措施是，在检测到有键按下时，执行一个 10ms 左右的延时程序后，再确认该键电平是否仍保持闭合状态电平，若仍保持为闭合状态电平，则可确认该键处于闭合状态，否则认为是干扰信号，从而去除了抖动影响。为简化电路通常采用软件方法。

(2) 对按键进行编码以给定键值或直接给出键号

任何一组按键或键盘都要通过 I/O 口线查询按键的开关状态。根据不同的键盘结构，采用不同的编码方法。但无论有无编码，以及采用什么编码，最后都要转换成为与累加器中数值相对应的键值，以实现按键功能程序的散转转移（相应的散转指令为 `JMP @A+DPTR`），因此，一个完善的键盘控制程序应能完成下述任务：

- 监测有无键按下；
- 有键按下后，在无硬件去抖动电路时，应用软件延时方法除去抖动影响；
- 有可靠的逻辑处理办法，如 n 键锁定，即只处理一个键，其间任何按下又松开的键不产生影响，不管一次按键持续有多长时间，仅执行一次按键功能程序；
- 输出确定的键号应满足散转指令要求。

10.1.2 独立式按键

独立式按键是指直接用 I/O 口线构成的单个按键电路。每个独立式按键单独占有一根 I/O 口线，每根 I/O 口线的工作状态不会影响其他 I/O 口线的工作状态，这是一种最简单易懂的

按键结构。

1. 独立式按键结构

独立式按键电路结构如图 10-2 所示，图中每根 I/O 口上都加了上拉电阻。在实际使用中，如 I/O 口内部已有上拉电阻（如 P1 口），可省去。

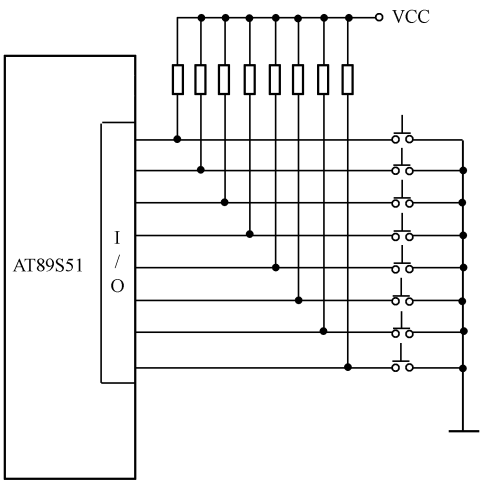


图 10-2 独立式按键电路

独立式按键电路配置灵活，硬件结构简单，但每个按键必须占用一根 I/O 口线，在按键数量较多时，I/O 口线浪费较大，故只在按键数量不多时采用这种按键电路。

在此电路中，按键输入都设置为低电平有效，上拉电阻保证了按键断开时，I/O 口线有确定的高电平。

2. 独立式按键的软件编制

下面是一简化的键盘程序。这段程序的作用是当检测到相应的键按下时就转向每个按键的功能程序。程序中省略了软件延时部分；OPR0~OPR7 分别为每个按键的功能程序的入口地址。设 I/O 口为 P1 口，P1.0~P1.7 对应 OPR0~OPR7。

程序清单：

```
START:  MOV    A,      # 0FFH      ; 置输入方式
        MOV    P1,    A
L1:     MOV    A,      P1          ; 输入键状态
        CJNE   A,      #0FFH, L3   ; 有键按下转 L3
        LCALL  DELAY             ; 延时 5ms
        SJMP   L1
L3:     LCALL  DELAY             ; 延时 5ms
        LCALL  DELAY             ; 延时 5ms
        MOV    A,      P1          ; 再读 P1 口
        CJNE   A,      #0FFH, L2   ; 确实有键按下转 L2
        SJMP   L1                ; 误读键，返回
L2:     JNB    ACC.0,    TAB0       ; 为 0 转 0 号键首地址
        JNB    ACC.1,    TAB1       ; 为 1 转 1 号键首地址
        :
```

```

                JNB      ACC.7,   TAB7      ; 为 7 转 7 号键首地址
                SJMP     L1              ; 再次读入键状态
TAB0:           LJMP     OPR0            ; 转向 0 号键功能程序
TAB1:           LJMP     OPR1
                ⋮
TAB7:           LJMP     OPR7
                ⋮
OPR0:           ⋮                      ; 0 号键功能程序
                ⋮
                ⋮
                ⋮      LJMP     START    ; 0 号键程序执行完返回
                ⋮
OPR7:           ⋮
                ⋮
                LJMP     START

```

C 语言程序清单:

```

#include <reg52.h>                //包含 SFR 寄存器的头文件
#define uint unsigned int        //定义数据类型
#define uchar unsigned char     //定义数据类型
main()
{ uint i;
  uchar value;
  while(1)
  { P1 = 0xff; //设置 P1 口为输入方式
    do{} while(P1 == 0xff);        //等待键盘输入
    for(i=0;i<1000;i++){};        //延时去抖动
    value = P1;                   //取键值
    switch (value)
    {
      case 0xfe: K0_pro();break;    //0 号键调用 K0_pro()键处理程序
      case 0xfd: K1_pro();break;    //1 号键调用 K1_pro()键处理程序
      .....
      case 0x7f: K7_pro();break;    //7 号键调用 K7_pro()键处理程序
      default : break;
    }
    do{} while(P1 != 0xff);        //等待键盘释放
    for(i=0;i<1000;i++){};        //延时（值可自定）去抖动
  }
}

```

10.1.3 行列式键盘

独立式按键电路每一个按键开关占一根 I/O 口线，当按键数较多时，要占用较多的 I/O 口线。因此，在按键数大于 8 时，通常多采用行列式（也称矩阵式）键盘电路。

1. 行列式键盘电路的结构及原理

图 10-3 为用 AT89S51 单片机 I/O 口组成的行列式键盘电路，图中行线 P2.0～P2.3，通过

4 个上拉电阻接+VCC，处于输入状态，列线 P1.0~P1.7 为输出状态。按键设置在行、列线交点上，行、列线分别连接到按键开关的两端。图 10-3 中右上角为每个按键的连接图。

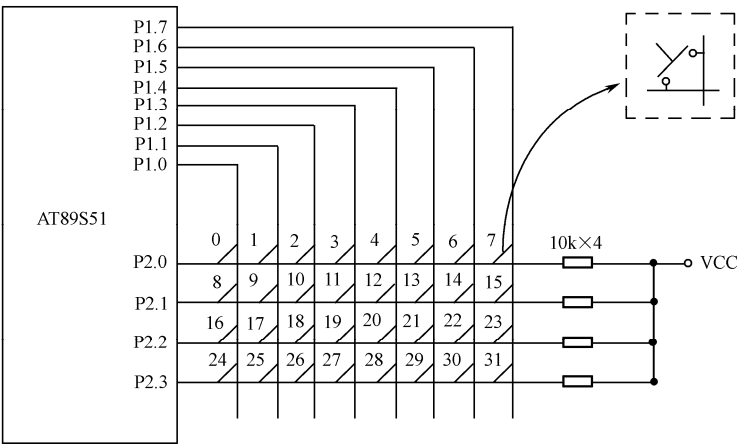


图 10-3 单片机 I/O 口组成的行列式键盘

当键盘上没有键闭合时，行、列线之间是断开的，所有行线 P2.0~P2.3 输入全部为高电平。当键盘上某个键被按下闭合时，则对应的行线和列线短路，行线输入即为列线输出。若此时初始化所有列线输出低电平，则通过读取行线输入值 P2.0~P2.3 的状态是否全为“1”，即可判断有无键按下。

但是键盘中究竟哪一个键被按下，并不能立刻判断出来，只能用列线逐列置低电平后，检查行输入状态的方法来确定。先令列线 P1.0 输出低电平“0”，P1.1~P1.7 全部输出高电平“1”，读行线 P2.0~P2.3 输入电平。如果读得某行线为“0”电平，则可确认对应于该行线与列线 P1.0 相交处的键被按下，否则 P1.0 列上无键按下。如果 P1.0 列线上无键按下，接着令 P1.1 输出低电平“0”，其余为高电平“1”，再读 P2.0~P2.3，判断其是否全为“1”，若是，表示被按键也不在此列，依次类推直至列线 P1.7。如果所有列线均判断完，仍未出现 P2.0~P2.3 读入值有“0”的情况，则表示此次并无键按下。这种逐列检查键盘状态的过程称做对键盘进行扫描。

2. 键盘的工作方式

在单片机应用系统中，扫描键盘只是 CPU 的工作任务之一。在实际应用中要想做到既能及时响应键操作，又不过多地占用 CPU 的工作时间，就要根据应用系统中 CPU 的忙闲情况，选择适当的键盘工作方式。键盘的工作方式一般有循环扫描方式和中断扫描方式两种。下面分别加以介绍。

(1) 循环扫描方式

循环扫描方式是利用 CPU 在完成其他工作的空余，调用键盘扫描子程序，来响应键输入要求。在执行键功能程序时，CPU 不再响应键输入要求。

键盘扫描程序一般应具备下述几个功能。

- 判断键盘上有没有键按下 其方法为 P1 口输出全扫描字“0”（即低电平）时，读 P2 口状态，若 P2.0~P2.3 全为“1”，则键盘无键按下，若不全为“1”，则有键按下。
- 去除键的抖动影响 其方法为在判断有键按下后，软件延时一段时间（一般为 10ms

左右)后,再判断键盘状态,如果仍为有键按下状态,则认为有一个确定的键被按下,否则以键抖动处理。

- 扫描键盘,得到按下键的键号 按照行列式键盘的工作原理,图 10-3 中 32 个键的键值从左上角的数字“0”键开始对应为如下分布(键值用十六进制数码表示):

00H,	01H,	02H,	03H,	04H,	05H,	06H,	07H
08H,	09H,	0AH,	0BH,	0CH,	0DH,	0EH,	0FH
10H,	11H,	12H,	13H,	14H,	15H,	16H,	17H
18H,	19H,	1AH,	1BH,	1CH,	1DH,	1EH,	1FH

其对应的键号如图 10-3 所示,这种顺序排列的键号按照行首键号与列号相加的办法处理,即每行的行首键号给以固定编号,0,8,16,24;列号依列线顺序为 0~7。

行扫描法的基本原理是:先使一条列线为低电平,如果这条列线上有闭合键,则相应的那条行线即为低电平,否则各行线状态都为高电平。这样就可以根据行线号和列线号求得闭合键的键码。

获取这 32 个键值时,P1 口和 P2 口输出与输入的相应值为如下分布(键值用十六进制数码表示):

FE×E	FD×E	FB×E	F7×E	EF×E	DF×E	BF×E	7F×E
FE×D	FD×D	FB×D	F7×D	EF×D	DF×D	BF×D	7F×D
FE×B	FD×B	FB×B	F7×B	EF×B	DF×B	BF×B	7F×B
FE×7	FD×7	FB×7	F7×7	EF×7	DF×7	BF×7	7F×7

上述分布的意义表示当行值与列值同时满足要求时,则选中该键。例如,0 号键的表达式为 FE×E,其表示当列值为 11111110B,行值为 1110B 时,选中 0 号键,以下依此类推。

行扫描的过程是:先使输出口输出 FEH(首列扫描字),即使 P1.0 为 0,然后读入行状态,判断行线中是否有低电平。如果没有低电平,再使输出口输出 FDH(第二列扫描字),依此类推,当行线中有状态为低电平时,则找到闭合键。根据此时零电平所在的行号和扫描列的列号得出闭合键的键码值。

闭合键的键码值 = 行首键号 + 列号

例如,当 P1 口的输出为 FDH(11111101B),即其第 1 列有输出时,读出 P2 口低 4 位的值为 0BH(1011B),说明是第 2 行与第 1 列相交的键闭合,则键号=16+1=17。

- 判别闭合的键是否释放。键闭合一次仅进行一次键功能操作。等键释放后去除键的抖动再将键值送入累加器 A 中,然后执行键功能操作。

设在主程序中已把 AT89S51 初始化为 P1 口做基本输出口,接键盘列线,P2 口做基本输入口,接 4 根行线,设计键扫描子程序框图如图 10-4 所示。键盘扫描子程序如下(程序中 KS 为查询有无按键按下子程序,DELAY 为延时子程序,延时时间为 5~10ms):

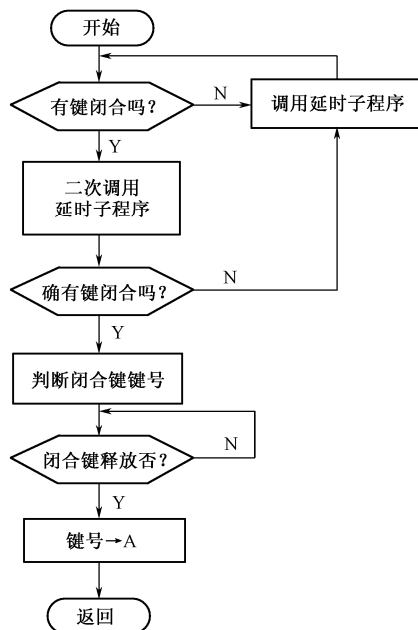


图 10-4 键扫描子程序框图

汇编语言程序如下：

```

KEY:   LCALL  KS           ; 调用 KS 判别有键按下吗
       JNZ    K1           ; 有键按下转移
       LCALL  DELAY        ; 无键按下，调延时子程序（省略）
       LJMP   KEY

K1:    LCALL  DELAY        ; 加长延时时间，消除键抖动
       LCALL  DELAY
       LCALL  KS           ; 调用 KS 子程序再次判别有无键闭合
       JNZ    K2           ; 键按下，转逐列扫描
       LJMP   KEY          ; 误读键，返回

K2:    MOV    R2,    # 0FEH ; 首列扫描字送 R2
       MOV    R4,    # 00H ; 首列号送 R4

K3:    MOV    A,      R2
       MOV    P1,    A     ; 列扫描字送 P1 口
       MOV    A,      P2   ; 读取行扫描值
       JB     ACC.0,   L1   ; 第 0 行无键按下，转查第 1 行
       MOV    A,      # 00H ; 第 0 行有键按下，该行的行首键号# 0H 送 A
       LJMP   LK          ; 转求键号

L1:    JB     ACC.1,   L2   ; 第 1 行无键按下，转查第 2 行
       MOV    A,      # 08H ; 第 1 行有键按下，该行行首键号# 08H 送 A
       LJMP   LK          ; 转求键号

L2:    JB     ACC.2,   L3   ; 第 2 行无键按下，转查第 3 行
       MOV    A,      # 10H ; 第 2 行有键按下，该行的行首键号# 10H 送 A
       LJMP   LK          ; 转求键号

L3:    JB     ACC.3,   NEXT ; 第 3 行无键按下，改查下一列
       MOV    A,      # 18H ; 第 3 行有键按下，该行的行首键号# 18H 送 A

LK:    ADD    A,      R4    ; 形成键码送入 A
       PUSH   ACC         ; 键码入栈保护

K4:    LCALL  DELAY
       LCALL  KS           ; 等待键释放
       JNZ    K4           ; 未释放，等待
       POP    ACC         ; 键释放，弹栈送 ACC
       RET              ; 键扫描结束，返回

NEXT:  INC    R4          ; 修改列号，指向下一列
       MOV    A,      R2
       JNB    ACC.7,   KEY ; 第 7 位为 0，已扫描完最高列转 KEY
       RL     A           ; 未扫描完，扫描字左移一位，变为下列扫描字
       MOV    R2,    A     ; 扫描字暂存 R2
       LJMP   K3          ; 转下列扫描

KS:    MOV    A,      # 0
       MOV    P1,    A     ; 全扫描字# 00H 送 P1 口
       MOV    A,      P2   ; 读入 P2 口行状态
       CPL    A           ; 变正逻辑，以高电平表示有键按下
       ANL    A,      # 0FH ; 屏蔽高 4 位
       RET              ; 出口状态：A≠0 时有键按下

```

C 语言程序清单：

```

..... //预处理
void delay(uint m) //延时程序

```

```

{   uint k;
    for(k=0;k<m;k++){};
}

uchar scan(void)                                //键扫描程序
{
uchar i,j,k,n,key_vaule ;
    P1 = 0;                                       //全扫描字#00H 送 P1 口
    do{} while((P2&0x0f)==0x0f);                //等待有键按下
    delay(1000);                                 //延时消除键抖动, 延时值自定
    if ((P2&0x0f)!=0x0f)                         //再次判别有无键闭合
    {                                             //键按下, 逐列扫描
        k = 0xfe;                               //首列扫描字
        for(i = 0;i<8;i++)
        {   P1 = k ;
            j  = P2&0x0f ;
            if (j!=0x0f)
            {   switch (j)
                {
                    case 0x0e:n = 0;break;        //第 0 行无键按下, 转查第 1 行
                    case 0x0d:n = 8;break;        //第 1 行无键按下, 转查第 2 行
                    case 0x0b:n = 16;break;       //第 2 行无键按下, 转查第 3 行
                    case 0x07:n = 24;break;       //第 3 行无键按下, 转查下一列
                    default: n = 0xff;break;
                }
            }
            break ;
        }
        k = (k <<1)+1;                          //扫描字左移一位, 变为下列扫描字
    }
}

key_vaule = n + i ;                             //行首键号加列号形成键值送入 key_vaule
P1 = 0;                                          //全扫描字#00H 送 P1 口
do{} while((P2&0x0f)!=0x0f);                  //等待键释放
delay(1000);                                   //延时消除键抖动
return (key_vaule);
}

```

在配有键盘的应用系统中,一般都相应地配有显示器,因而,在系统初始化后,CPU 必须反复不断地轮流调用扫描式显示子程序和键盘输入程序。在识别有键闭合后,执行规定的操作再重新进入上述循环。

(2) 中断工作方式

采用上述扫描键盘的工作方式,虽然也能响应键入的命令或数据,但是这种方式不管键盘上有无按键按下,CPU 总要定时扫描键盘,而应用系统在工作时,并不经常需要键输入,因此 CPU 经常处于空扫描状态。为了提高 CPU 的工作效率,可采用中断扫描工作方式。即只有在键盘有键按下时,才发中断请求。CPU 响应中断请求后,转中断服务程序,进行键盘扫描,识别键码。中断扫描工作方式的一种简易键盘接口如图 10-5 所示。该键盘直接由 AT89S51 P1 口的高、低字节构成 4×4 行列式键盘。键盘的列线与 P1 口的低 4 位相接,键盘的行线接到 P1 口的高 4 位。

图 10-5 中的四输入端与门就是为中断扫描方式而设计的，其输入端分别与各列线相连，输出端接单片机的外部中断输入 $\overline{\text{INT0}}$ 。初始化时，使键盘行输出全部置零。当有键按下时， $\overline{\text{INT0}}$ 端为负脉冲，向 CPU 发出中断申请，若 CPU 开放外部中断，则响应中断请求，进入中断服务程序。在中断服务程序中执行前面讨论的扫描式键盘输入子程序，注意返回指令要改用 RETI，此外还要注意保护与恢复现场。

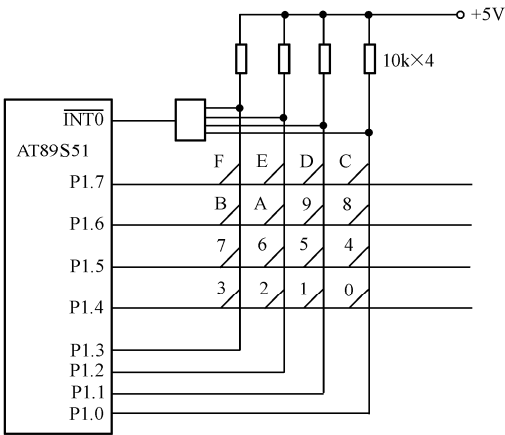


图 10-5 中断方式键盘电路

由于 P1 口为双向 I/O 口，可以采用一种称为“线路反转”的方法识别键值。步骤如下：

- (a) P1.0~P1.3 输出 0，由 P1.4~P1.7 输入并保存数据到 A 中；
- (b) P1.4~P1.7 输出 0，由 P1.0~P1.3 输入并保存数据到 B 中；
- (c) A 的高 4 位与 B 的低 4 位组合成为键码值；
- (d) 查表求得键号；

线路反转汇编程序如下：

```

ORG      0000H
LJMP     START
ORG      0003H
LJMP     FZH           ; 转读键值程序
ORG      0030H

START:
MOV      P1,    # 0FH
MOV      IE,    # 81H   ; CPU 开中断，外部中断 0 开中断
        :
ORG      0080H           ; 读键值中断程序
FZH:     SETB    RS0      ; 保护第 0 组工作寄存器（主程序中默认用第 0 组，
                           ; 以下不再说明）

MOV      P1,    # 0F0H   ; 设 P1.0~P1.3 输出 0
MOV      A,     P1       ; 读 P1 口
ANL      A,     # 0F0H   ; 屏蔽低 4 位，保留高 4 位
MOV      B,     A        ; 将 P1.4~P1.7 的值存入 B
MOV      P1,    # 0FH   ; 反转设置，设 P1.4~P1.7 输出 0
MOV      A,     P1       ; 读 P1 口
ANL      A,     # 0FH   ; 屏蔽高 4 位，保留低 4 位
ORL      A,     B        ; 与 P1.4~P1.7 的值相或，形成键码

```

```

MOV     B,      A
MOV     R0,     # 00H      ; 置键号初值
MOV     DPTR,   # TAB      ;
LOOP:
MOV     A,      R0
MOVC    A,      @A+DPTR    ; 取键码值
CJNE    A,      B, NEXT2   ; 与按键值相比较, 如果不相等, 继续
SJMP    RR0        ; 相等返回, 键码值在 A 中
NEXT2:
INC     R0        ; 键值加 1
CJNE    R0,     # 10H, LOOP
RR0:    CLR     RS0      ; 恢复第 0 组工作寄存器
RETI
TAB:
DB 0EEH, 0EDH, 0EBH, 0E7H ; 0, 1, 2, 3 的键码值
DB 0DEH, 0DDH, 0DBH, 0D7H ; 4, 5, 6, 7 的键码值
DB 0BEH, 0BDH, 0BBH, 0B7H ; 8, 9, 10, 11 的键码值
DB 07EH, 07DH, 07BH, 077H ; 12, 13, 14, 15 的键码值
END

```

线路反转法 C51 语言程序如下:

```

..... //预处理
uchar keycode;
uchar code key_value[16]= {0xee,0xed,0xeb,0xe7,0xde,0xdd,0xdb,0xd7,
    0xbe,0xbd,0xbb,0xb7,0x7e,0x7d,0x7b,0x77
    }; //键码值
void main(void)
{ P1 = 0x0F ;
  IT0 = 0; //外部中断 0 采用边沿触发方式
  IE = 0x81 ; //CPU 开中断, 允许外部中断 0 中断
  while(1) //等待键值处理
  { ..... }
}
void int0_pro() interrupt 0 using 1 //定义外部中断 0 中断函数, 用第 1 组工作寄存器
{
  uchar key,i;
  keycode = 0x00; //置键号初值
  P1 = 0xf0; //设 P1.0~P1.3 输出 0
  key = P1&0xf0; //保存 P1.4~P1.7 的值
  P1 = 0x0f; //反转设置, 设 P1.4~P1.7 输出 0
  key += P1&0x0f ; //与 P1.4~P1.7 的值相或, 形成键码
  for (i =0;i<16;i++)
  {
    if (key == key_value[i]) //查表得到键码值
    {
      keycode = i ; //返回键号
      break;
    }
  }
}
}

```

如果采用中断方式，最好采用硬件方法去除抖动。

10.2 显示器接口

为方便人们观察和监视单片机的运行情况，通常需要用一种显示器作为单片机的输出设备，用来显示单片机的键输入值、中间信息及运算结果等。

在单片机应用系统中，常用的显示器主要有 LED（发光二极管显示器）和 LCD（液晶显示器）。这两种显示器具有耗电省、配置灵活、线路简单、安装方便、耐振动、寿命长等优点。两者相比，LED 显示器价格更低廉，寿命更长，结构更简单，LCD 显示器功耗更低，显示清晰度更高。用户可根据实际需要选择，本节将对这两种显示器分别予以介绍。

10.2.1 LED 显示器的结构与原理

LED 显示器是由发光二极管显示字段的显示器件，也可称为数码管。其外形结构如图 10-6（a）所示，由图可见它由 8 个发光二极管（以下简称字段）构成，通过不同的组合可用来显示 0~9、A~F 及小数点“.”等字符，图中 D_p 表示小数点，COM 表示公共端。

数码管通常有共阴极和共阳极两种型号，见图 10-6（b）和（c）。图中电阻为外接的，一般共阳极数码管必须外接电阻，共阴极不一定外接电阻。共阴极数码管的发光二极管阴极必须接低电平，当某一发光二极管的阳极接高电平（一般为+5V）时，此二极管点亮；共阳极数码管的发光二极管是阳极并接到高电平，对于需点亮的发光二极管使其阴极接低电平（一般为地）即可。显然，要显示某字形就应使此字形的相应字段点亮，实际上就是送一个用不同电平组合代表的数据至数码管。这种装入数码管中显示字形的数据称字形码。

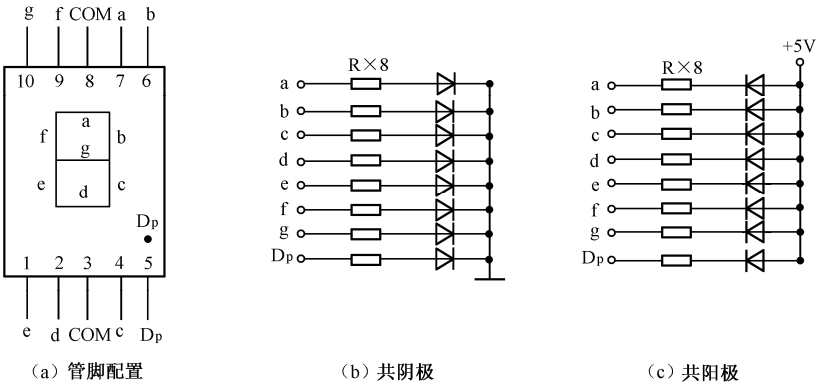


图 10-6 “8” 字型数码管

下面以共阴极数码管为例说明字形与字形码的关系。

对照图 10-6（a）字段，字形码各位定义如下。

D7	D6	D5	D4	D3	D2	D1	D0
D_p	g	f	e	d	c	b	a

数据位 D_0 与 a 字段对应， D_1 与 b 字段对应……，依此类推。参考图 10-6（a）和（b）可以看出，如要显示“7”字形，a、b、c 三个字段应点亮，所以对应的字形码为

00000111B。

如要显示“E”字形，对应的 a、f、g、e、d 字段应点亮，所以其字形码为 01111001B。

共阴极常用的显示字形如表 10-1 所示，按照显示字符顺序排列。通常显示代码存放在程序存储器的固定区域中，构成显示代码表。当要显示某字符时，可根据地址及显示字符查表。

表 10-1 常用字形表（共阴极）

字 符	字 形	D7	D6	D5	D4	D3	D2	D1	D0	字 形 码
0	0	0	0	1	1	1	1	1	1	3FH
1	1	0	0	0	0	0	1	1	0	06H
2	2	0	1	0	1	1	0	1	1	5BH
3	3	0	1	0	0	1	1	1	1	4FH
4	4	0	1	1	0	0	1	1	0	66H
5	5	0	1	1	0	1	1	0	1	6DH
6	6	0	1	1	1	1	1	0	1	7DH
7	7	0	0	0	0	0	1	1	1	07H
8	8	0	1	1	1	1	1	1	1	7FH
9	9	0	1	1	0	1	1	1	1	6FH
A	A	0	1	1	1	0	1	1	1	77H
B	b	0	1	1	1	1	1	0	0	7CH
C	C	0	0	1	1	1	0	0	1	39H
D	d	0	1	0	1	1	1	1	0	5EH
E	E	0	1	1	1	1	0	0	1	79H
F	F	0	1	1	1	0	0	0	1	71H

参考表 10-1 可以得到共阳极常用的显示字形 0~FH，只要对应的字形码取反即可。其顺序为 C0H, F9H, A4H, B0H, 99H, 92H, 82H, F8H, 80H, 90H, 88H, 83H, C6H, A1H, 86H, 8EH。

LED 显示器显示方式有静态和动态两种，下面分别予以介绍。

10.2.2 LED 静态显示方式

静态显示是指在显示器显示某个字符时，相应的字段（发光二极管）一直导通或截止，直到变换为其他字符。数码管工作在静态显示方式下时，其位选线统一接地（共阴极）或高电平（共阳极）。数码管段选线各位相互独立，通常是与一个锁存器的输出口相接。只要在各位的段选线上保持段选码电平，该数码管就能保持相应的显示字符。如果要显示新的字符，

则重新发送新字符的段选码电平。

静态显示通常有以下 2 种方法：

1. 用并行口控制显示器

并行输出接口可以采用 74 系列的锁存器（如 74373 等），也可以采用可编程 I/O 接口芯片等，还可以采用专门用于驱动显示的硬件译码器件等与段选线相连。显示时只需把待显示字符的段选码电平送入输出口。图 10-7 所示为一个由 3 个数码管相连的三位静态显示器。显然这个显示器至少要占用 24 位 I/O 口线。由于这种方法需要占用较多 I/O 口线，当要显示的数码管超过 4 个，这种方法对于 80C51 单片机就不再适用。

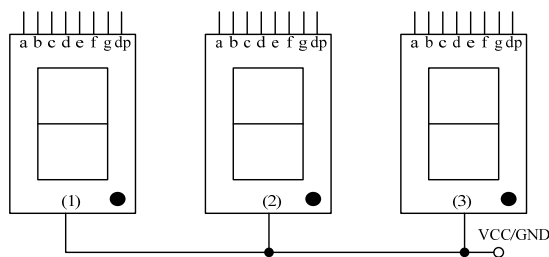


图 10-7 三位静态显示电路

2. 用串行口控制显示器

静态显示方式常采用串行口设定为方式 0 输出方式，采用串行输入/并行输出的移位寄存器构成显示电路，可参考第 7 章的例 2。

这种方式下要显示某字符，首先要把这个字符转换为相应的字形码，然后再通过串行口发送到串行输入/并行输出的移位寄存器，例如 74HC164 芯片，一个数码管需要一片 74HC164。74HC164 把串行口收到的数转换为并行输出，并加到数码管上，详见文献 1。这种方式虽然节约了 I/O 口线，却增加了电路的复杂程度和移位寄存器芯片，在位数较多时，字符更新速度慢。

通常静态显示方式亮度较高，软件编程较简单。但由于他需要占用较多 I/O 口线，或者需要较多的芯片，线路较复杂，所以在位数较多时常采用动态显示方式。

10.2.3 LED 动态显示方式

动态扫描显示方式是把各显示器的相同段选线并联在一起，并由一个 8 位 I/O 口控制，而其公共端由其他相应的 I/O 口控制，然后采用扫描方法轮流点亮各位 LED，使每位分时显示该位应该显示的字符，这是最常用的显示方式之一。

图 10-8 所示电路即为单片机应用系统中的一种动态显示方式示意图。为简化问题，该例中直接用单片机的 I/O 口输出相应的字形码和位选扫描电平。其中 P1 口输出与选通的数码管相对应的字形码信号，P2 口的 6 位口作为位选信号，每次仅有 1 路输出是 1 电平（其余为 0）。

通常采用动态显示字形码输出及位选信号输出时，应经驱动后，再与数码管相连。图中字形驱动选用 8 路三态同相缓冲器 74HC244，位选驱动使用 ULN2803 反相驱动芯片（参见

附录 E)。采用 8 段共阴极数码管，发光时，字形驱动输出 1 有效，位选驱动输出 0 有效。

由于 8 路段选线都由 P1 口控制，因此，每个要显示的字符都会同时加到这 6 个数码管上。要想让每位显示不同的字符，就必须采用如下的扫描工作方式。

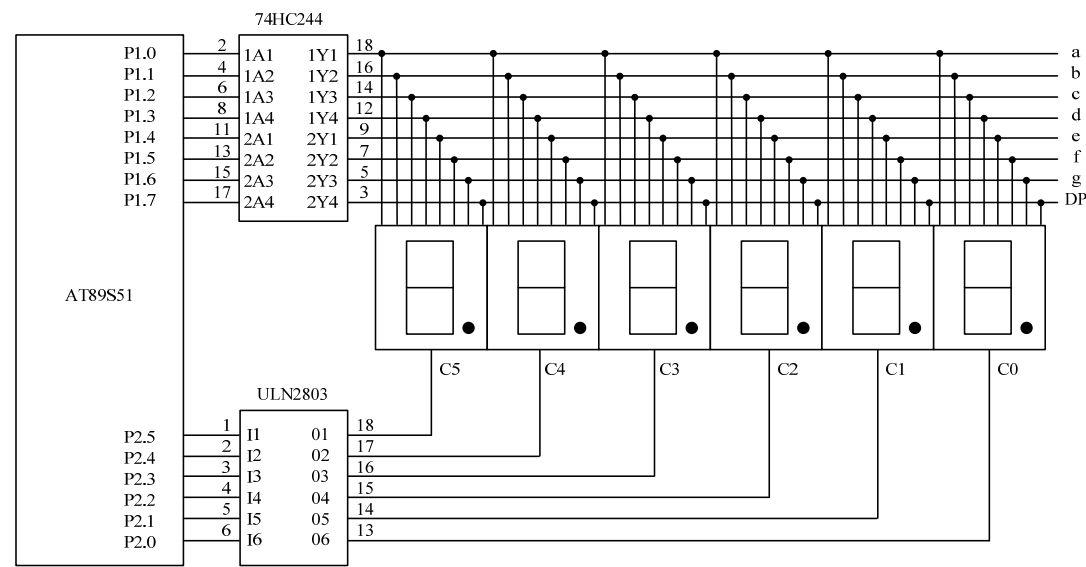


图 10-8 扫描式显示电路

这种工作方式是分时轮流选通数码管的公共端，使得各数码管轮流导通，即各数码管是由脉冲电流导电的（循环扫描 1 次的时间一般为 10ms）。当所有数码管依次显示一遍后，软件控制循环，使每位显示器分时点亮。例如：若要显示“123DEF”，则位选码、段选码扫描一遍的相应显示内容如表 10-2 所示。

表 10-2 6 位动态扫描显示内容

段选码	位选码	显示器显示内容					
06H	20H	1					
5BH	10H		2				
4FH	08H			3			
5EH	04H				D		
79H	02H					E	
71H	01H						F

这种方式不但能提高数码管的发光效率，而且由于各数码管的字段线并联使用，从而大大简化了硬件线路。

各数码管虽然是分时轮流通电，但由于发光数码管的余辉特性及人眼的视觉暂留作用，因此，循环扫描频率选取适当时，看上去所有数码管是同时点亮的，察觉不出有闪烁现象。不过，采用这种方式时，数码管不宜太多，否则每个数码管所分配到的实际导通时间就会太短，从而使得亮度不够。

按照图 10-8 所示电路编写一段 6 位数码管的显示子程序。设 DIS0~DIS5 是片内显示缓

冲区，共 6 个单元，对应 6 个数码管的显示内容。程序中，先取 DIS5 中的数据，对应选中图 10-8 所示扫描式显示电路中最左边的数码管，其余以此类推。

汇编语言程序清单如下：

```

DIR:    PUSH    ACC
        PUSH    DPH
        PUSH    DPL
        MOV     R0,    #DIS5      ;指向显示缓冲区首单元
        MOV     R6,    #20H      ;选中最左边的数码管
        MOV     R7,    #00H      ;设定显示时间
        MOV     DPTR,  #TAB1     ;指向字形表首地址
DIR1:   MOV     A,    #00H
        MOV     P2,A             ;关断显示
        MOVC    A,    @R0        ;取要显示的数据
        MOVC    A,    @A+DPTR    ;查表得字形码
        MOV     P1,    A         ;送字形码
        MOV     A,    R6         ;取位选字
        MOV     P2,    A         ;送位选字
HERE:   DJNZ    R7,    HERE      ;延时
        INC     R0              ;更新显示缓冲单元
        CLR     C
        MOV     A,    R6
        RRC     A               ;位选字右移
        MOV     R6,    A
        JNZ     DIR1           ;未扫描完，继续循环
        POP     DPL
        POP     DPH
        POP     ACC             ;恢复现场
        RET
TAB1:   DB      3FH,06,5BH,4FH,66H,6DH,7DH,07    ;0H~7H
        DB      7FH,6FH,77H,7CH,39H,5EH,79H,71H ;8H~0FH

```

C 语言程序清单如下：

```

#include<reg51.h>
#define uchar  unsigned char
uchar TABLE1[]={0x20,0x10,0x08,0x04,0x02,0x01}; //位选码
uchar TABLE2[]={0x06,0x5b,0x4f,0x5e,0x79,0x71}; //段选码,显示 1, 2, 3, D,E,F
void delay()
{
    uchar i,j;
    for(i=0;i<200;i++)
    {
        for(j=0;j<5;j++)
        {;}
    }
}
main()
{
    uchar i;
    for(;;)
    {
        for(i=0;i<6;i++)

```

```

    {
        P2=0X00;           //关断显示
        P1=TABLE2[i];      //送字形码，显示某个字符
        P2=TABLE1[i];      //送位选字，选中某一位
        delay();
    }
}
}

```

采用此显示子程序，每调用 1 次，仅扫描 1 遍；要得到稳定的显示，则必须不断地调用显示子程序。

动态扫描式显示接口硬件虽然简单，但在使用时必须反复循环显示，若 CPU 须作其他操作，则只能插入循环程序中，这就降低了 CPU 的工作效率。因此，在实际应用中，要根据具体情况来选用显示方式。

10.2.4 液晶显示器概述

液晶显示器 LCD (Liquid Crystal Display) 是一种极低功耗的显示器。由于其具有清晰度高，信息量大等特点，从而使得它越来越广泛地应用在小型仪器的显示中。本节将介绍 LCD 的工作原理以及与单片机的接口和应用技术。

液晶是一种介于固体和液体之间的有机化合物，它是利用液晶特殊的折射性进行显示的。由于液晶工作时需要加一种固定的交流电压，因而驱动其工作的过程较繁琐。为了简化对 LCD 器件的操作，现在已经配置了专门的驱动器和控制器。在进行信息显示时，由计算机对控制器进行操作，使控制器为驱动器提供扫描时序信号和准备显示的内容，然后，驱动器驱动液晶器件进行显示。此外，现在有一些专用单片机本身增加了直接驱动 LCD 的功能，此时可不用驱动器。

把 LCD 与驱动器组装在一起的部件的英文名称为 LCD Module，简称 LCM。LCM 一般分为 3 类，即段码型液晶模块、点阵字符液晶模块和点阵图形液晶模块。

- 段码型液晶模块是由数显液晶显示器件和集成电路组装成的模块，也称为“笔段型液晶模块”。其段码显示形式与 LED 显示器类似，是应用最简单的一类。
- 点阵字符液晶模块是由点阵字符液晶显示器件和专用的驱动器、控制器、结构件等装配成的模块，可以显示数字和英文字符。这种模块本身具有字符发生器，显示容量大于段码型液晶模块。
- 点阵图形液晶模块的点阵像素是连续排列的。因此，可以显示连续、完整的图形，也可以显示中文、数字等，是功能最全面，但控制也最复杂的一类，其价格也高于前两类模块。

本节选择字符型液晶显示模块 LCM 为例，是因为目前 LCM 应用较广泛，使用方法也不太复杂，并且它们的接口方法比较统一，各制造商所采用的模块控制器都是 HD44780U 或其兼容产品，虽然形成的最终产品型号较多，如有 MDLS81809、MDLS16163、MDLS24265 等，但它们的使用方法大同小异。因此学会一种型号，即可收到触类旁通的效果。为叙述方便，在下面的介绍中统称这类模块为 LCM。

10.2.5 字符型液晶显示模块 LCM 的组成及原理

字符型液晶显示模块显示的是点阵字符，有 5×8 和 5×11 这 2 种点阵字符供选择；显示行数有 1、2、4 行共 3 种，每行有 8、16、24、40、80 等多种字位长度。每 1 个字符位都可显示 1 个字符，通过软件编程可显示要求的字符。LCM 主要由指令寄存器、数据寄存器、AC 地址计数器、DDRAM 显示数据存储器等组成。下面介绍各主要部分的功能。

(1) 指令寄存器与解码器

指令寄存器用于存放计算机送来的指令代码，指令解码器用于翻译准备执行的指令。

(2) 时序产生器

时序产生器用于产生逻辑电路的工作时序。LCM 的工作时序易于与单片机配合。

(3) 地址指针计数器 AC

AC 是显示数据寄存器 DDRAM 和字符发生器 CGRAM 共同的地址指针计数器。指示当前 DDRAM 和 CGRAM 的地址，也可以指示当前光标和闪烁的位置地址。

(4) DDRAM 显示数据存储器

DDRAM 显示数据存储器用于存放 LCD 当前要显示的数据，其容量为 80×8 位，即可存放 80 个字符。这 80 个字符的地址由地址计数器 AC 提供，DDRAM 各单元对应显示屏上的各字符位。其地址定义有两种方式，即 1 行显示的地址定义和 2 行显示的地址定义，通常为后者。表 10-3 所列 40×2 显示在 LCD 上的显示位置与 DDRAM 地址的对应关系。在 2 行显示时，不论一行的字符是多少个，第 2 行的首地址总是从 DDRAM 地址的 40H 单元开始。

表 10-3 显示位置与 DDRAM 地址的对应关系

显示位置	1	2	3	...	19	...	38	39	40
第 1 行的 DDRAM 地址	0	1	2	...	13H	...	25H	26H	27H
第 2 行的 DDRAM 地址	40H	41H	42H	...	53H	...	65H	66H	67H

由表 10-3 可以看出：对于 20×2 的 LCD 显示器 2 行各为 20 个字符，第 1 行的地址为 0~19 (13H)，第 2 行是从 40H 单元开始至 53H 结束；1 个 DDRAM 地址对应 1 个显示位置，写入不同的字符即可显示不同的字形。

(5) CGROM 字符产生器 ROM

在 CGROM 字符产生器的 ROM 中存放已经固化好的字符库。只要通过软件写入某个字符的字符代码，控制器即将它作为字符库的地址，并把该字符输出到驱动器显示。例如：当把 41H 字符码写入 DDRAM 时，CGROM 会自动把相应的字符“A”送至 LCD 显示器显示。

(6) CGRAM 字符产生器 RAM

在 CGRAM 字符产生器的 RAM 中，可存放 8 个用户设计的 5×8 点阵图形。

5×8 点阵字符代码与 CGRAM 地址的对应关系如下：

字符代码	0	1	2	...	6	7
CGRAM 地址	00H~07H	08H~0FH	10H~17H	...	30H~37H	38H~3FH

图 10-9 所示为字符代码地址为 00 的 CGRAM 地址及字模数据与显示效果的对应关系。

本例要显示的字为“月”，当要显示字符的某一位为1时，表示该点亮，为0则表示不亮。每个字符均由8个字节组成，前7个字节为字符体，最后1个字节留作光标位置。在编程时，要先向 CGRAM 地址的 0~7 送入 00H~1FH 之间的数字（仅低5位有效），然后再将对应码的代码写入 DDRAM 即可显示。

在图 10-9 所示的例子中，因为已经把字符“月”的点阵字符存入 00H~07H，所以它的字符代码就是 0；如果把“月”的点阵字符存入 20H~27H，则它的字符代码就变为 4。

代码0的CGRAM地址	“月”的CGRAM单元数据	显示屏的显示效果
0	0FH	□ ■ ■ ■ ■
1	09H	□ ■ □ □ ■
2	0FH	□ ■ ■ ■ ■
3	09H	□ ■ □ □ ■
4	0FH	□ ■ ■ ■ ■
5	09H	□ ■ □ □ ■
6	13H	■ □ □ ■ ■
7	00H	□ □ □ □ □

图 10-9 自定义字符设计举例

(7) 忙标志触发器 BF

忙标志触发器 BF 指示 LCD 是否正在作内部处理工作。在写指令前，必须首先检查 BF 标志。当 BF=0 时，才可以将指令写入 LCD 控制器；当 BF=1 时，表示 LCD 正在作内部处理工作，不能将指令写入 LCD 控制器。处理完毕，BF 自动清 0。

(8) 光标/闪烁控制器

HD44780U 具有光标和闪烁功能、由计算机控制光标和闪烁电路的工作，可以通过软件设定在 DDRAM 当前地址产生 1 个光标和 1 个闪烁的字符。

上述各部件的工作是通过计算机对 RS、R/、E 等引脚的控制实现的。DB0~DB7 是外界与 LCM 的传输引脚。

10.2.6 字符型液晶显示模块 LCM 的引脚及说明

LCM 一般采用 16 引脚封装，某常用型号的外形与引脚分布如图 10-10 所示。

各引脚的名称及功能如下。

1 脚 VSS——电源地。

2 脚 VCC——+5V。

3 脚 VO——调节电压端，用以改变显示对比度，通常此端接可调电阻活动端。如果不调节可直接接地。

4 脚 RS——寄存器选择端。当 RS 为 0 时，选择指令寄存器；当 RS 为 1 时，选择数据寄存器。

5 脚 R/W——读/写控制信号。当 R/W 为 0 时，选择写操作；当 R/W 为 1 时，选择读操作。由 RS 和 R/W 控制读/写操作的格式如表 10-4 所列。

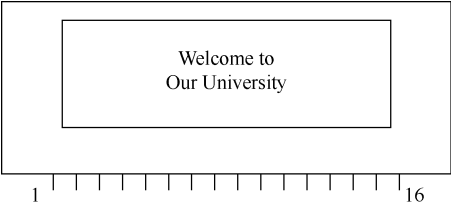


图 10-10 LCM 外形及引脚分布

表 10-4 RS 和 R/W 控制读写操作的格式

RS	R/W	操作
0	0	写指令寄存器
0	1	读忙标志位 BF 和地址计数器 AC
1	0	写数据寄存器
1	1	读数据寄存器

6 脚 E——使能控制端，E 是正脉冲信号（脉冲宽度为 $0.5\mu\text{s}$ 左右），下降沿有效。

7~14 脚 DB0~7——8 位数据总线。

15、16 脚——不同厂家生产的 LCM 其 15、16 脚的功能设置不同（具体对应哪种功能，须查看其型号说明书），通常有以下 3 种情况：

- 15 脚 NC——空；16 脚 NC——空。
- 15 脚 LED+——LED 背光电源正；16 脚 LED- ——LED 背光电源负。
- 15 脚 E1——使能信号（控制第 1、2 行）；16 脚 E2——使能信号（控制第 3、4 行）。

10.2.7 LCM 的指令

LCM 提供了 11 条控制指令，通过这些指令可实现基本控制显示功能。指令一览表参见表 10-5。由表可知，除了“清除屏幕”和“归位”2 条指令的执行时间为 1.64ms 之外，其余指令的执行时间均为 $40\mu\text{s}$ 。每条指令的执行时间为每一次读/写 LCD 寄存器后要等待的时间，之后 CPU 才可发下一条指令。在编程时，要考虑这个细节。对表 10-5 中的控制指令说明如下：

（1）清除屏幕指令

该指令清除显示内容，光标返回 00H 位置（显示屏左上角）。其命令字为 01。

（2）归位指令

该指令置 DDRAM 地址为 0,即将 AC 清 0；使光标和光标所在位的字符回到原点，但 DDRAM 的内容不变。其命令字为 02。

表 10-5 LCM 指令一览表

指令名称	控制信号		控制代码								运行时间
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
清除屏幕	0	0	0	0	0	0	0	0	0	1	1.64ms
归位	0	0	0	0	0	0	0	0	1	*	1.64ms
输入方式	0	0	0	0	0	0	0	1	I/D	S	$40\mu\text{s}$
显示状态	0	0	0	0	0	0	1	D	C	B	$40\mu\text{s}$
光标/画面移位	0	0	0	0	0	1	S/C	R/L	*	*	$40\mu\text{s}$
工作方式	0	0	0	0	1	DL	N	F	*	*	$40\mu\text{s}$
CGRAM 的地址设置	0	0	0	1	A5	A4	A3	A2	A1	A0	$40\mu\text{s}$
DDRAM 的地址设置	0	0	1	A6	A5	A4	A3	A2	A1	A0	$40\mu\text{s}$
读取忙标志/地址计数器	0	1	BF	A6	A5	A4	A3	A2	A1	A0	$40\mu\text{s}$
写数据	1	0	数据								$40\mu\text{s}$
读数据	1	1	数据								$40\mu\text{s}$

(3) 输入方式指令

该指令选择显示时的输入方式。

其命令字高 6 位的固定值为 000001B。其最低 2 位的作用如下：

- 当 I/D=1 时，为增量方式，在数据读写操作后，AC 可自动加 1，光标右移一个字符位；
- 当 I/D=0 时，为减量方式，在数据读写操作后，AC 可自动减 1，光标左移一个字符位。
- 当 S=1 时，显示画面移位；
- 当 S=0 时，显示画面不移位。

(4) 显示状态控制指令

该指令控制画面、光标和闪烁的开与关。其命令字高 5 位的固定值为 00001B。其低 3 位的作用如下：

- 当 D=1 时，为开显示方式；
当 D=0 时，为关显示方式，不出现画面，但显示数据仍保存在 DDRAM 中。
- 当 C=1 时，显示光标，光标的位置由地址指针寄存器 AC 确定；
当 C=0 时，关光标。
- 当 B=1 时，显示字符闪烁，闪烁字符的位置由地址指针寄存器 AC 确定，在控制器频率为 250kHz 时，闪烁频率为 2.4Hz；
当 B=0 时，显示字符不闪烁。

(5) 光标或画面移位指令

该指令选择光标或画面（即显示字符）向左或向右移动 1 个字符位。其命令字高 4 位的固定值为 0001B。其低 4 位的作用如下：

- 当 S/C=1 时，光标与字符同时移位；
当 S/C=0 时，仅光标移位。
- 当 R/L=1 时，右移；
当 R/L=0 时，左移。

位 1 和位 0 的 X 可为任意值。

(6) 工作方式设置指令

该指令选择数据长度与显示格式。其命令字高 3 位的固定值为 001B。位 1 和 0 无作用，位 4~2 的作用如下：

- 当 DL=1 时，采用 8 位数据总线；
当 DL=0 时，采用 4 位数据总线。
- 当 N=1 时，显示双行字符行；
当 N=0 时，显示单行字符行。
- 当 F=1 时，采用 5×10 点阵字符体；
当 F=0 时，采用 5×7 点阵字符体。

(7) CGRAM 的地址设置指令

该指令把 6 位 CGRAM 的地址写入地址指针寄存器 AC，随后计算机对数据的操作就是对 CGRAM 的读/写操作。该指令高 2 位的固定值为 01B。A5~A0 位用于选择 CGRAM 的地址，其范围为 00~63。

(8) DDRAM 的地址设置指令

该指令把 7 位 DDRAM 的地址写入地址指针寄存器 AC，随后计算机对数据的操作就是对 DDRAM 的读/写操作。

指令位 7 的固定值为 1。A6~A0 位用于选择 DDRAM 的地址，并存放于 AC 中。写入本指令后，随后必须是读/写 DDRAM 数据的指令。

(9) 读取忙标志/地址计数器 AC

该操作用于读取忙标志位 BF 及地址计数器 AC 的内容。

在读/写数据之前，一定要检查 BF 位的状态。当 BF=0，可以存取 LCD；当 BF=1，则不能存取 LCD。A6~A0 位为 DDRAM 或 CGRAM 的地址（取决于计算机最近向 AC 写入的是哪类地址）。

(10) 写入 DDRAM/CGRAM

在地址设定指令后，该操作把 D0~D7 表示的字符码写入 DDRAM 以显示相应的字符，或把用户设计的字符写入 CGRAM。

(11) 读取 DDRAM/CGRAM 中数据

在地址设定指令后，该操作读取 DDRAM/CGRAM 中的数据。

10.2.8 LCM 的复位及初始化

在 LCM 内部有一个复位电路，如果电源符合 LCM 的时序要求（当电源从 0.2 V 上升到 4.5 V 所需要的时间在 0.1~10 ms 之内，或者电源的低电平（<0.2 V）瞬间抖动持续时间大于 1 ms 时），则 LCM 上电即可自动复位。复位后的默认状态如下：

- ① 清除显示，即清 DDRAM；
- ② 功能设定为 8 位数据长度，单行显示，5×7 点阵字符；
- ③ 显示屏、光标、闪烁功能均为关闭；
- ④ 输入模式为 AC 地址自动加 1，显示画面不移动。

如果电源不符合 LCM 的时序要求，则需要采用指令对其热启动，一般操作步骤如下：

- ① 写入指令代码 30H 或 38H。
- ② 延时时间>4.1ms。
- ③ 写入指令代码 30H 或 38H。
- ④ 延时时间>100μs。
- ⑤ 写入指令代码 30H 或 38H。

注意：在步骤①~⑤之间不能检测 BF 标志位。

如果电源符合 LCM 的时序要求或已经用指令进行了热启动，则可根据 LCM 的时间要求和各指令的功能，根据实际直接设置功能初始化指令，通常为如下几步：

- ① 设置工作方式，指令为 001DLNFxx；
- ② 清除显示，指令为 01；
- ③ 设定输入方式，指令为 0000011/DS；
- ④ 设置显示状态，指令为 00001DCB。

在输入上述指令前，及对数据进行读取时都要检测 BF 标志位，如果为 1 则等待，为 0 则执行下一条指令。

10.2.9 LCM 的接口及应用举例

【例 10-1】以 AT89S51 单片机为主机，实现与字符型 LCM 的接口，编程显示 2 行字母数字，且第 1 行显示“WELCOME TO”，第 2 行显示“OUR UNIVERSITY”。此例中的 LCM 为 20×2 显示模块，主机频率为 6MHz，其接口电路如图 10-11 所示。要求设定为 2 行显示，8 位数据长度， 5×7 点阵字形。

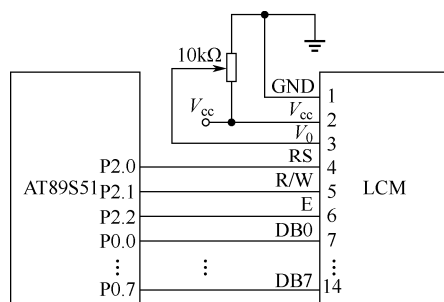


图 10-11 接口电路原理图

汇编语言编程如下：

```

RS      BIT      P2.0          ; 寄存器选择信号
RW      BIT      P2.1          ; 读/写选择信号
E       BIT      P2.2          ; 使能控制
ORG     0000H
LJMP    MAIN
ORG     60H
MAIN:
MOV     SP,      #60H          ; 设堆栈指针
LCALL   INIT                 ; 调 LCM 初始化程序
LCALL   FIRST                ; 调设定显示地址为第 1 行第 1 个位置的子程序
MOV     DPTR,    #TAB1        ; 设置第 1 行字符的首地址指针
LCALL   DISPLAY              ; 调用显示字符程序
LCALL   SECOND            ; 调设定显示地址为第 2 行第 1 个位置的子程序
MOV     DPTR,    #TAB2        ; 设置第 2 行字符的首地址指针
LCALL   DISPLAY              ; 调用显示字符程序
SJMP    $
; LCM 初始化程序（本程序采用了用于热启动的指令段）
INIT:
LCALL   DELAY1                ; 调延时 5ms 子程序，省略
LCALL   DELAY1                ; 延时等待电源稳定
MOV     A,       #30H
MOV     P2,      #11111000B    ; E、RS、R/W 为 0
NOP
SETB    E              ; E 为高
MOV     P0,      A      ; 写入指令代码
CLR     E              ; E 为低
LCALL   DELAY1                ; 延时 5ms 子程序，省略
MOV     A,       #30H
MOV     P2,      #11111000B    ; E、RS、R/W 为 0
NOP
SETB    E              ; E 为高
MOV     P0,      A      ; 写入指令代码
CLR     E              ; E 为低
LCALL   DELAY2              ; 延时 120μs 子程序，省略
MOV     A,       #30H
MOV     P2,      #11111000B    ; E、RS、R/W 为 0
NOP

```

SETB	E		; E 为高
MOV	P0,	A	; 写指令
CLR	E		; E 为低
LCALL	DELAY2		; 延时 120μs
MOV	A,	#38H	
MOV	P2,	#11111000B	; E、RS、R/W 为 0
NOP			
SETB	E		; E 为高
MOV	P0,	A	; 写入指令代码
CLR	E		; E 为低
LCALL	DELAY2		; 延时 120μs 子程序, 省略
MOV	A,	#38H	; 功能设置为 2 行显示, 8 位数据总线, 5×7 点阵
ACALL	WRC		; 判读 BF 和写命令
MOV	A,	#01H	; 清除显示
ACALL	WRC		
LCALL	DELAY1		
MOV	A,	#06H	; 设置输入方式为 AC 加 1, 光标右移, 画面不动
ACALL	WRC		
MOV	A,	#0EH	; 设置显示状态为开显示, 显示光标, 不闪烁
ACALL	WRC		
RET			
; 判读 BF 和写命令			
WRC:	ACALL	BUSY	; 调判读 BF 子程序
MOV	P2,	#11111000B	; E、RS、R/W 为 0, 准备写
NOP			
SETB	E		; E 为高
MOV	P0,A		; 写入指令代码
CLR	E		; E 为低
RET			
BUSY:	PUSH	ACC	
W:	MOV P2,	#11111010B	; RS=0,R/W=1,E=0, 准备读忙标志
NOP			
SETB	E		
MOV	A,P0		; 读 BF 和 AC 值
JB	ACC.7,	W	; BF 不为 0, 等待
CLR	E		
POP	ACC		
RET			
; 写显示数据子程序			
WRTD:	ACALL	BUSY	
MOV	P2,	#11111001B	;RS=1,R/W=0,E=0,准备写数据
SETB	E		
MOV	P0,	A	
CLR	E		
LCALL	DELAY2		
RET			
; 显示字符程序			
DISPLAY :			

```

MOV    R1,    #00
NEXT:
MOV    A,     R1
MOVC   A,     @A+DPTR    ; 将 DPTR 所指的字符码逐一送到 LCD 显示
CJNE   A,     #21H,DSL   ; 到结束符“!”返回, 否则继续显示
RET
DSL:    LCALL  WRTD       ; 调写显示数据子程序
INC     R1
SJMP   NEXT
FIRST:
MOV     A,     #10000000B ; DDRAM 的地址设为 80H, 即要显示的字符
                                从第 1 行的第 1 个位置开始

LCALL   WRC
RET
SECOND:
MOV     A,     #11000000B ; 设要显示的字符从第 2 行的第 1 个位置开始
LCALL   WRC
RET
TAB1:DB'  WELCOME TO'    ; LCD 第 1 行显示的字符串
DB'!'; 结束码为“!”
TAB2:DB' OUR UNIVERSITY' ; LCD 第 2 行显示的字符串
DB'!'; 结束码为“!”
END

```

DB 后面的字符码也可以直接写为它的代码形式, 例如 TAB1 也可以表示如下:

```

DB 20H,20H,20H,57H,45H,4CH,43H,4FH,4DH,45H
DB 20H,54H,4FH,20H,20H,21H; 20H 为空格符, 21H 为“!”符

```

C 语言程序如下。

```

#include <reg51.h>
#include <intrins.h>
sbit  RS=P2^0;    //寄存器选择信号
sbit  RW=P2^1;    //读写选择信号
sbit  E =P2^2;    //使能信号
#define uchar unsigned char
uchar code TAB1[]={"  WELCOME  TO! "}; //LCD 第 1 行显示的字符串
uchar code TAB2[]={" OUR UNIVERSITY! "}; //LCD 第 2 行显示的字符串
void lcm_init(void); //LCM 初始化程序
void delay(void);    //延时 5ms
void delay2(void);   //延时 120us
void display(uchar temp[]); //显示子函数
void WR_DAT(uchar dat); //写数据
void WR_COM(uchar command); //写指令
void READ_BUSY(void);  //读判“忙”信号子函数
void main()
{
    lcm_init();
    WR_COM(0x80); //设定显示地址为第 1 行第 1 个位置
    display(TAB1); //调用显示字符程序
    WR_COM(0xc0); //设定显示地址为第 2 行第 1 个位置
}

```

```

        display(TAB2);                                //调用显示字符程序
        while(1);                                     //待机
    }
    void display(uchar temp[])
    {   uchar i=0;
        do{
            WR_DAT(temp[i]);                           //调显示数据子程序
            i++;
        }
        while(temp[i]!='!');                           //到结束符“!”返回，否则继续显示
    }
    void READ_BUSY(void)                               //读判“忙”标志子函数
    {   uchar flag;
        do {
            P2=0xfa;                                   // RS=0,  RW=1,  E=0, 准备读“忙”标志
            _nop_();
            E=1;
            flag=P0;
            E=0; }
        while((flag&0x80)!=0);                           //标志位不为0，则继续读
    }
    void WR_DAT(uchar dat)                             //写数据
    {
        READ_BUSY();
        P2=0xf9;                                       // RS=1,  RW=0,  E=0, 准备写数据
        _nop_();
        E=1;
        P0=dat;
        E=0;
        delay2();
    }
    void WR_COM(uchar command)                         //写命令
    {
        READ_BUSY();
        P2=0xf8;                                       // RS=0,  RW=0,  E=0, 准备写命令
        _nop_();
        E=1;
        P0=command;
        E=0;
    }
    void lcm_init()                                    //LCM 初始化程序（本程序采用了用于热启动的指令段）
    {
        delay();                                       //延时 5ms
        delay();                                       //延时 5ms，等待电源稳定
        P2=0xf8;                                       // RS=0,  RW=0,  E=0, 准备写命令
        _nop_();
        E=1;
        P0=0X30;
        E=0;
    }

```

```

    delay();                //延时 5ms
    P0=0xF8;
    _nop_();
    E=1;
    P0=0X30;
    E=0;
    delay2();              //延时 120us
    P0=0xF8;
    _nop_();
    E=1;
    P0=0X30;
    E=0;
    delay2();              //延时 120us
    P0=0xF8;
    _nop_();
    E=1;
    P0=0X38;
    E=0;
    delay2();              //延时 120us
    WR_COM(0x38);          //功能设置为 2 行显示, 8 位数据总线, 5×7 点阵
    WR_COM(0x01);          //清除显示
    delay();               //调用延时时间大于 1.64ms
    WR_COM(0x06);          //设置输入方式为 AC 加 1, 光标右移, 画面不动
    WR_COM(0x0e);          //设置显示状态为开显示, 显示光标, 不闪烁
}
void delay2(void)          //延时大于 120us, 晶振为 12MHz
{
    int i;
    for(i=0;i<30;i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}
void delay(void)           //延时大于 5ms, 晶振为 12MHz
{
    int i;
    for(i=0;i<1000;i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

```

例 10-2 利用例 10-1 的显示系统，显示“2017 年 06 月 16 日”。

解：本例主要为了说明 CGRAM 的使用方法，并使读者学会用 LCD 显示自己创造的字符。一般的操作步骤如下：

1. 按照图 10-11 所示的方法对要显示的字符编码；
2. 设定 CGRAM 的起始地址，存入自己创造的字符，它们对应的字符码为 0~7；
3. 设定 DDRAM 的地址，把要显示的字符码写入 DDRAM。

本例需要用到自己创造的字符“年”、“月”和“日”。对它们编码后的点阵图形码表，放在 TAB1 中；要显示的“2017 年 06 月 16 日”代码放在 TAB2 中。

汇编语言编程如下：

```
RS      BIT      P2.0
RW      BIT      P2.1
E        BIT      P2.2
ORG     0000H
LJMP    MAIN
ORG     60H
MAIN:
MOV     SP,                #60H; 设堆栈指针
LCALL   INIT               ; 调 LCM 初始化程序, 同例 10-1 (省略)
LCALL   WORD               ; 调把自创字符写入 CGRAM 的子程序
LCALL   FIRST              ; 调设定显示地址为第 1 行第 1 个位置的子程序, 同例 1 (省略)
MOV     DPTR, #TAB2        ; 设置第 1 行字符的首地址指针
LCALL   DISPLAY            ; 调用显示字符程序, 同例 10-1 (省略)
SJMP $
; 把自创字符写入 CGRAM 程序
WORD:
MOV     A, #40H            ; 设定 CGRAM 的首地址为 00
LCALL   WRC                ; 写命令, 同例 10-1, 省略
MOV     R5, #24            ; 写入 24 个字符码
MOV     DPTR, #TAB1        ; 24 个字符码的首地址送入 DPTR
MOV     R6, #0             ; 偏移量初始值
NEXT:
MOV     A, R6              ; 取查表偏移量
MOVC    A, @A+DPTR         ; 写显示数据, 同例 10-1, 省略
LCALL   WRD                ; 写显示数据, 同例 10-1, 省略
INC     R6
DJNZ    R5, NEXT
RET
TAB1: DB 08H,0FH,12H,0FH,0AH,1FH,02H,00H ; “年” 的点阵图形码
      DB 0FH,09H,0FH,09H,0FH,09H,13H,00H ; “月” 的点阵图形码
      DB 0FH,09H,09H,0FH,09H,09H,0FH,00H ; “日” 的点阵图形码
TAB2: DB '2017',0, '06',01, '16',02      ; 0、01、02 分别是年、月、日的字符码
      DB '!' ; 结束码
END
```

C 语言程序如下：

```
#include <reg51.h>
#include <intrins.h>
```

```

sbit RS=P2^0;    //寄存器选择信号
sbit RW=P2^1;    //读写选择信号
sbit E =P2^2;    //使能信号
#define uchar unsigned char
uchar code TAB1[]={0X08,0X0F,0X12,0X0F,0X0A,0X1F,0X02,0X00,    //“年”的点阵图形码
                   0X0F,0X09,0X0F,0X09,0X0F,0X09,0X13,0X00,    //“月”的点阵图形码
                   0X0F,0X09,0X09,0X0F,0X09,0X09,0X0F,0X00};    //“日”的点阵图形码
uchar code TAB2[]={ '2','0','1','7','0','0','6','0','1','1','6','2','!'}; //0、01、02 分别是年、月、日的字符码，
                    '!'为结束码
void WORD(void);    //自创字符写入 CGRAM 的子程序
void main()
{
    lcm_init();    //调 LCM 初始化程序，同例 10-1（省略）
    WORD();    //调把自创字符写入 CGRAM 的子程序
    WR_COM(0x80);    //在液晶第一行显示 2017 年 06 月 16 日
    display(TAB2);    //显示数据，同例 10-1，省略
}
void WORD(void)
{
    uchar i;
    WR_COM(0x40);    //设定 CGRAM 的首地址为 00，写命令同例 10-1，省略
    for(i=0;i<24;i++)    //写入 24 个字符码
    {
        WR_DAT(TAB1[i]);    //写显示数据，同例 10-1，省略
    }
}

```

在上述章节分别介绍了键盘与显示器的工作原理及应用实例，但在单片机应用系统中，有时需要同时使用键盘与显示器。此时，为了节省 I/O 口线，常常把键盘和显示电路做在一起，构成实用的键盘、显示电路。实现的方法可以是采用并行扩展，也可以是采用串行扩展，见参考文献[1]。但当要扩展的键盘或显示器较多时，这两种方法的硬件电路都较复杂。为简化系统的软/硬件设计，提高显示质量，充分提高 CPU 的工作效率，陆续出现了一些专用于键盘与显示器接口的芯片。这些芯片通过编程设置键盘与显示功能，可大大简化硬件电路并减少软件工作量，是微处理器仪表理想的键盘与显示驱动电路，如早期出现的 Intel8279 等。随着串行技术的发展，近年出现的 HD7279、MAX7219/7221 和 ZLG7290 等芯片，具有外部占用引线少，显示方式可调节等优点，并逐渐取代了 8279。限于篇幅，这里不予介绍。

10.3 功率驱动器件及接口电路

单片机的主要作用之一就是利用 I/O 口对外部设备进行控制，但因为单片机的 I/O 口驱动能力有限，不可能直接驱动大功率开关及设备，如电磁阀、电动机、电炉及接触器等，所以在输出通道端口必须配接输出驱动电路控制功率开关器件，再通过功率开关器件控制大功率设备的工作。此外，许多大功率设备在开关过程中会产生强电磁干扰，可能会造成系统的误动作或损坏，所以在强电情况下还要考虑电气隔离问题。本节介绍常用的隔离技术及常见的几种功率开关器件及接口电路。

10.3.1 输出接口的隔离技术

为防止大功率设备在开关过程中产生的强电磁干扰，在单片机的输出端口常采用隔离技术，现在最常用的就是光—电隔离技术，因为光信号的传输不受电场、磁场的影响，可以有效地隔离电信号，根据这种技术生产的器件称为光电隔离器，简称光隔。

目前生产的光电隔离器型号品种很多，性能参数也不尽相同，但它们的基本工作原理是相同的。如图 10-12 (a) 所示即为一个三极管型的光电隔离器原理图。当发光二极管中通过一定值的电流时，二极管发出的光使光敏三极管导通；当发光二极管中无电流时，则光敏三极管截止，由此达到控制开关的目的。

利用光电隔离器实现输出通道的隔离时，一定要注意，被隔离的通道必须单独使用各自的电源，即驱动发光二极管的电源与驱动光敏三极管的电源必须是各自独立的，不能共地，否则外部干扰信号可能会通过电源进入系统，就起不到隔离作用了。图 10-12 (b) 为正确接法，图 10-12 (c) 为错误接法。

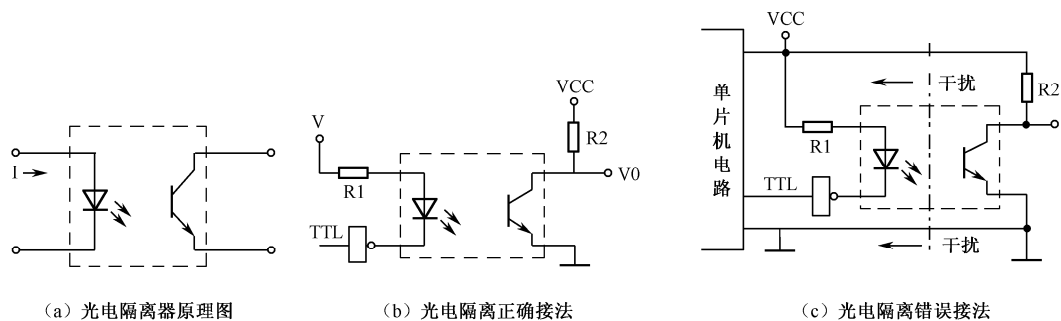


图 10-12 光电隔离器原理及接法

一般单片机的 I/O 口可以直接驱动光电隔离器，对于有些驱动能力有限的 I/O 口可以采用集电极开路的门电路，如 7406、7407 等去驱动光电隔离器。

10.3.2 直流负载驱动电路

常见的直流负载驱动电路有如下几种。

1. 晶体管

晶体管通常用于控制电流不太大的直流负载（大约几百 mA）。

当晶体管处于放大状态时，基极输入一个 mA 级的小电流 I_B ，集电极可输出一个放大的电流，达到放大功率的目的。单片机的 I/O 管脚即可驱动一般的晶体管，如图 10-13 (a) 所示。

2. 达林顿开关驱动电路

把两只晶体管接成复合型，做成一只管子叫达林顿管。达林顿管的特点是用很小的输入电流，即可得到比晶体管大的输出电流，可以直接推动较大的负载，如图 10-13 (b) 所示。

现在有一些把多个达林顿管集成在一起的芯片，如 ULN2068、ULN2803 等驱动器均可直接与单片机接口，驱动多路负载。

3. 功率场效应管

功率场效应管在制造中多采用 V 沟槽工艺, 简称 VMOS 场效应管, 这种场效应管只要求微安级输入电流即可控制中功率或大功率负载, 一般单片机的 I/O 口均可直接驱动大功率负载。其控制电路如图 10-13 (c) 所示。

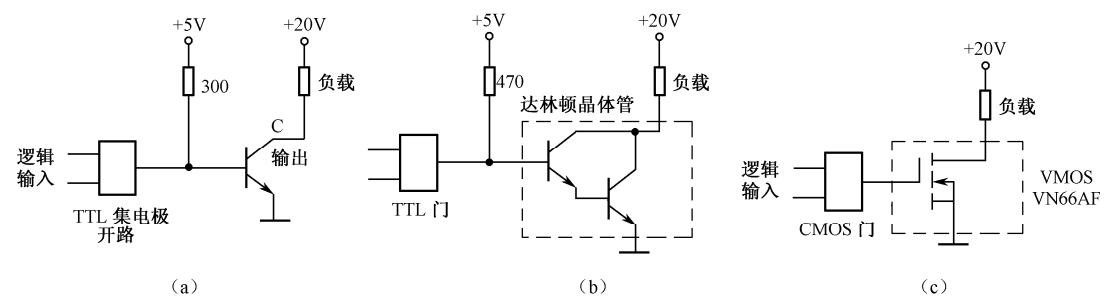


图 10-13 直流电源负载驱动电路

10.3.3 晶闸管负载驱动电路

晶闸管（也称为可控硅）是一种可控的半导体功率器件。它具有容量大、效率高、体积小、寿命长、便于控制等许多优点，因此用途很广。它在可控整流、逆变、斩波、变频、大功率开关电路中已被广泛采用。

1. 晶闸管及主要特性

晶闸管按导通方式可分为单向晶闸管（也称为单向可控硅）和双向晶闸管（也称为双向可控硅），其符号如图 10-14 所示。

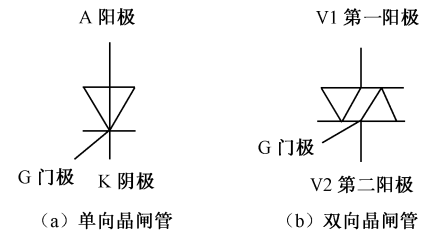


图 10-14 晶闸管符号

下面分别简单介绍这两种晶闸管的结构特性。

(1) 单向晶闸管

单向晶闸管的结构简图如图 10-14 (a) 所示, 显然, 其导通方向是由 A 到 K, G 为控制端。单向晶闸管导通条件有两个, 二者必须同时具备:

- 阳极电压必须大于等于阴极电压 ($U_A \geq U_K$);
- 门极 (G) 必须加正向电压。二者同时满足, 晶闸管才可导通。

单向晶闸管关断条件如下:

单向晶闸管一旦导通, 即使去掉控制极电压, 仍能继续维持导通。要使晶闸管关断, 必须把阳极电压减少到不足以维持其导通; 为了加速其关断, 常在阳极与阴极之间加一反向电压。

(2) 双向晶闸管

双向晶闸管的结构简图如图 10-14 (b) 所示, 由图可见, 一只双向晶闸管相当于两只反向并联的单向晶闸管, 它常用于控制交流负载, 若为电阻负载在交流电源过零时自动关断, 不必采取专门关断措施。

晶闸管最大优点是可控性, 通过控制极的小电流可控制大电流负载的电源通断, 这种电路称为开关电路, 很容易实现单片机控制。单向晶闸管可作为直流开关, 双向晶闸管或两只单向晶闸管反向并联可作为交流开关。

2. 晶闸管与单片机接口电路

在用晶闸管做开关时, 由于交流电路属强电, 为防止交流电对单片机的干扰, 一般要用光电耦合器隔离。现在已经生产出很多种光电耦合触发的可控硅器件, 例如 Motorola 公司的 MOC3020/1/2/3 等, 这种器件可以直接由单片机的 I/O 脚控制交流电, 如图 10-15 所示。图中双向可控硅和交流负载串联, 当 AT89S51 的 P1.0 输出高电平时, 光耦导通, 从而使双向晶闸管导通, 接通负载回路。

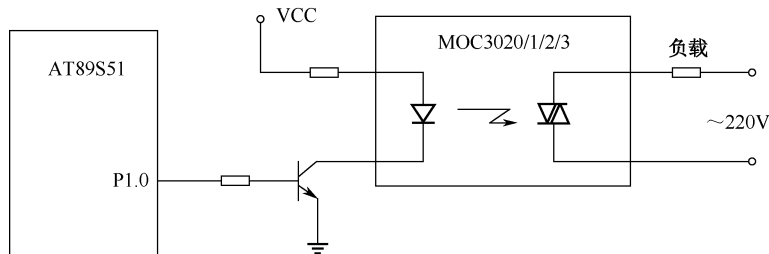


图 10-15 单片机控制的可控硅接口电路

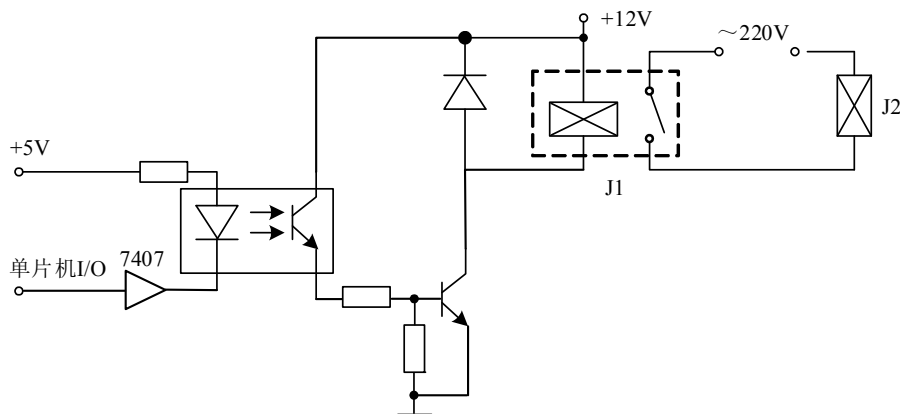
10.3.4 继电器接口电路

继电器在家电、国防及工业控制等领域应用非常广泛, 这是一种历史悠久, 较成熟的功率开关器件, 由于其具有接触电阻小, 流通电流大、耐高压等特点, 得到广泛应用。其种类非常多, 如果按工作原理分类有电磁继电器、固态继电器、温度继电器、时间继电器和舌簧继电器等。篇幅关系本节仅介绍电磁继电器和固态继电器。

电磁式继电器的工作原理是通过采用控制电流通过继电器线圈后产生的电磁吸力, 控制大电流通过的触点闭合或断开, 从而控制大型设备电路的通断。例如用十几毫安电流接通线圈, 可使能通过几十个安培的触点接通。它控制的负载可以是直流也可以是交流。

注意: 继电器线圈是电感性负载, 所以线圈两端要并联续流二极管, 以保护驱动器不被浪涌电压损坏。

图 10-16 所示是一个继电器与单片机的接口电路。为简化电路, 在此图中假设 J1 线圈所需电流很小, 由光隔输出就可带动。不同大小的继电器所需要的驱动电流不相同, 但一般情况单片机的 I/O 口都是不能直接驱动这个线圈的, 所以通常是在 I/O 口和线圈之间接晶体管或 7407 等驱动器。在这个电路中的继电器 J1 是用直流电源励磁的, 通过直流继电器 J1 对需要用交流电源工作的交流负载 J2 间接控制。



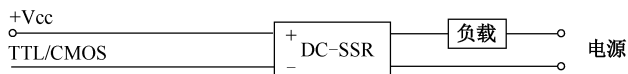
J1 一中间继电器；J2 一交流负载

图 10-16 单片机控制的继电器接口电路

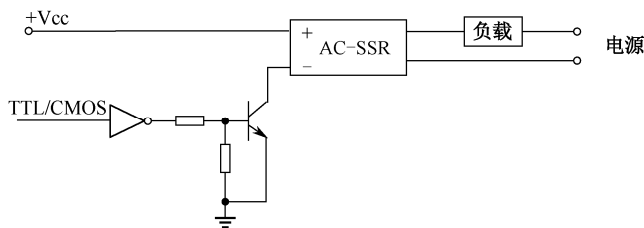
10.3.5 固态继电器接口电路

固态继电器简称 SSR (Solid State Relay)，是一种无触点的功率开关器件，固态继电器是一个四端口的器件，其器件内部有光电耦合器将输入与输出隔离开，在输出回路中有功放电路（主要采用双向可控硅或功率场效应管）。它的两个输入端，为控制端，两个输出端用于接通和切断负载电流。SSR 特别适用于在测控系统中作为输出通道的控制元件，与普通的电磁式继电器相比，其有很多显著优点，如寿命长，功耗小，体积小、可靠性高，开关速度快（比电磁继电器响应速度快），耐冲击，特别是它的输入端控制电流小，用计算机的 I/O 输出端即可直接驱动，便于计算机控制，且输出端电流大，易于大功率设备。因此，在很多场合，它已经逐渐取代传统的电磁式继电器和磁开关作为开关量输出控制。但由于固态继电器漏电流较大，触点单一，过载能力差，使用温度范围窄等缺点，所以它并不能完全取代电磁继电器。

固态继电器有直流和交流 2 类，2 种接口电路如图 10-17 所示，DC-SSR 的输入电流一般小于 15mA，所以 DC-SSR 可以用 TTL 电路、OC 门或晶体管直接驱动，在图 10-17 (a) 中，计算机的 I/O 口可直接控制 DC-SSR。而 AC-SSR 的输入电流通常大于 DC-SSR，小于 500mA，一般要加接晶体管驱动，见图 10-17 (b)。在此电路中单片机的低电平输出使三极管导通，



(a) DC-SSR 接口电路



(b) AC-SSR 接口电路

图 10-17 固态继电器接口电路

从而接通固态继电器 SSR 的输入电源, SSR 导通后输出端接通负载回路。因为固态继电器的输入电压一般为 4~32 伏, 因此在使用时要注意选择适当的电压 V_{cc} 和限流电阻。

在控制电路中到底采用什么功率开关器件, 要具体问题具体分析。

10.4 打印机接口

打印机是计算机系统常备外设之一, 打印机品种很多, 打印原理也不完全相同。在单片机控制的各种仪表中, 为了打印票据、表格、曲线等, 常常使用微型打印机。因为其体积小, 便于携带, 也便于安装在仪器中。这些打印机都是一种由单片机控制的智能型打印机, 在仪器中使用较广泛, 本节将以微型打印机为例介绍。

10.4.1 微型打印机简介

“微”是微型打印机的一个显著特点, 其具有最小的体积、最大的纸仓空间, 最经济的打印效果。早期常使用的微型打印机有 TP μ P-16A/40A 和 GP-16 等, 这类打印机因为功能落后前几年已经被淘汰。目前较流行的微型打印机品牌有爱普生、映美、富士通、兄弟、斯普瑞特等。早期生产的微型打印机的接口均为并口, 打印方式均为针打。现在的微型打印机接口有并口+USB 口和串口+USB 口, 为使用方便还配置了串、并转换线, 还有 Wi-Fi 方式。打印方式增加了热敏方式。针打是利用打印头把字符通过色带打印在普通卷纸上, 其信息保存时间比较长。热敏打印是利用能发热的打印头把字符打印到专用热敏打印纸上, 这种方式信息保存时间一般是 2~3 个月, 但这种打印机价格低。

本文以斯普瑞特 (SPRT) 公司生产的 SP-RMDIHD 型热敏微打为例, 介绍微型打印机的一般应用。SPRT 热敏微型打印机有串口或并口两种接口模式, 应用较广泛。

一般微型打印机的接口与时序要求完全相同, 操作方式相近, 硬件电路及插脚完全兼容, 只是指令代码不完全相同, 打印宽度不相同。

1. 一般微型打印机的主要技术性能

① 采用单片机控制, 具有控制打印程序及标准的接口, 便于和各种计算机应用系统或智能仪器仪表联机使用。

② 具有较丰富的打印命令, 命令代码均为单字节, 格式简单。

③ 可产生全部标准的 ASCII 代码字符, 以及 1 万个以上的非标准字符和图符。用户可通过程序自行定义 96 个代码字符 (6×7 点阵), 并可通过命令用此 96 个代码字符去更换任何驻留代码字型, 以便用于多种文字的打印。

④ 打印时只需将相应的代码送入打印机即可实现打印。可混合打印代码字符和点阵图样。

⑤ 字符、图符和点阵图可以在宽和高的方向放大为×2、×3、×4 倍。打印宽度不同型号不同, 一般在 54~88mm 之间。

⑥ 可用命令更换每行字符的点行数 (包括字符的行间距), 即字符行间距空点行可在 0~256 间任选。

⑦ 带有水平和垂直制表命令, 便于打印表格。

2. 打印接口举例

目前打印接口有多种,限于篇幅,在此以串行接口举例。串行接口的打印机一般支持 TTL 电平和 RS232 电平两种接口方式。

SP-RMDIIID 型热敏微打的串行接口有 5 针和 10 针两种形式。这里仅介绍 10 针引脚信号,引脚排列如图 10-18 所示。图中引脚名称及功能如下:

- ① RXD: 数据线,打印机接收从计算机发来的数据。
- ② TXD: 数据线,打印机向计算机发送数据。低电平有效, $\overline{\text{STB}}$ 宽度应大于 $0.5\mu\text{s}$ 。在该信号的上升沿,数据线上的 8 位并行数据被打印机读入机内锁存。
- ③ CTS: 打印机“忙”状态信号。当该信号有效(高电平)时,表示打印机正忙于处理数据。此时,主计算机不能向打印机送入新的数据,否则将丢失。
- ④ DCD: 功能同 CTS。
- ⑤ DSR: 该信号指示打印机在线与否。

2	4	6	8	10
TXD		DSR	CTS	
DCD	RXD	GND		
1	3	5	7	9

图 10-18 SP-RMDIIID 串口插脚排列

10.4.2 字符代码及打印命令

SP-RMDIIID 打印机在西文工作方式下兼容字符集 1 和字符集 2,在中文方式下按汉字点阵的不同可以打印 24 点阵高的 12×24 半角字符、16 点阵高的 8×16 半角字符,汉字有 24×24 、 16×16 的国标一、二级字库选择,以及大量的数学符号、专用符号、图形、曲线,条形码等。可通过命令更换打印行距和字符的大小,还可以自定义部分代码字符。

1. 命令代码

SP-RMDIIID 打印机提供的打印命令与传统 ESC 打印命令完全兼容。表 10-6 给出了一些常用的打印命令。命令代码通常是由 1~2 个字节组成,其第一个字节范围是 00H~1FH,例如表 10-6 中代码的第一列数;第二个代码字节范围是 04~77H,例如选择字符集 1 的命令代码是 1B 36。有些命令还有第 3 个字节 n,例如表 10-6 中 1C 57 n 命令表示设置放大倍数,如果 n 为 2,则表示放大 2 倍。

表 10-6 常用打印命令代码及功能

命令代码(十六进制)	符号和格式	命令功能
0A	LF	换行
0D	CR	回车
1B 36	ESC 6	选择字符集 1
1B 37	ESC 7	选择字符集 2
1B 40	ESC @	初始化打印机

续表

命令代码（十六进制）	符号和格式	命令功能
1C 26	FS &	设置国标一、二级汉字库打印
1C 2E	FS .	取消汉字打印方式
1C 57 n	FS W n	设置放大倍数
1D 68 n	GS h n	设置条码高度

2. 字符代码

SP-RMDIII 中全部字符代码编号范围为 20H~FFH，分为以下几类情况：

- 编号 20H~7FH 为标准 ASCII 代码；
- 编号 80H~FFH 为非 ASCII 代码，其中包括少量汉字、希腊字母、块图图符和一些特殊的字符。编号范围非 ASCII 代码表详见打印机说明书。
- 西文方式下字符码由两个字符集组成，每个字符集中的字符码范围都是 20H~0FFH。可以参见厂家提供的详细说明书。

● 中文方式下字符码范围也是 20H~0FFH，其他中文汉字符合 GB2312 汉字编码表，汉字区域，高位为 0xB0~0xF7，低位为 0xA1~0xFE。用 2 个字节能确定一个汉字，如“年”对应着“0xC4 0xEA”。

字符串的回车换行代码为 0AH。但是，当输入代码满 32 个字符或 12 个汉字时，打印机自动回车。字符代码串实例如下：

① 打印字符串“\$3265.37”

输送代码串为：24，33，32，36，35，2E，33，37，0A。

② 打印“32.8cm”

输送代码为：33，32，2E，38，63，6D，0A。

③ 打印“2017 年 8 月 8 日”

输送代码在西文方式下时：32，30，31，37，8C，38，8D，38，8E，0A。

输送代码为中文方式下时：32，30，31，37，C4，EA，38，D4，C2，38，C8，D5，0A。

10.4.3 打印机与单片机接口举例

本节以 SP-RMDIII 智能打印机为例介绍与单片机的连接与编程。打印机与单片机的连接

如图 10-19 所示。该智能打印机默认的串行口波特率为 9600，也可以通过厂家给定的上位机软件更改波特率，本例采用厂家默认设置。

按照图 10-19 所示电路，编制一个程序。要求打印机先打印片内 50H~5FH 单元内的数据，此数据区内的数据已是分离的 BCD 码，均放在低半字节，然后再打印时间“2017 年 8 月 8 日”。

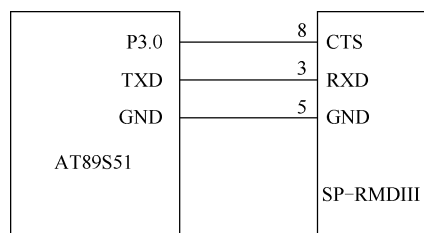


图 10-19 SP-RMDIII 打印机与单片机的连接图

汇编语言程序清单:

```

        ORG     0000H
        LJMP    MAIN

        ORG     0200H
MAIN:    MOV     SCON,    #40H
        MOV     TMOD,    #20H
        MOV     TH1,     #0FDH           ; 串口波特率设置为 9600
        SETB    TR1
        MOV     R0,      #50H           ; 送打印数据区首地址
        MOV     R7,      #16           ; 送数据长度
LOOP1:   MOV     A,       @R0
        ADD     A,       #30H           ; 变换为 ASCII 码
        LCALL   PRT       ; 打印一个数据或字符
        INC     R0
        DJNZ    R7,      LOOP1         ; 是否打印完数据
        MOV     A,       #0AH           ; 送命令结束代码
        LCALL   PRT
        CLR     A
        MOV     R3,      A
LP2:     MOV     DPTR,    #TAB           ; 指向表首
        MOVC    A,       @A+DPTR       ; 取待打印字符
        LCALL   PRT       ; 调打印程序
        INC     R3
        MOV     A,       R3
        XRL     A,       #9
        JZ      LP3       ; 打印完 9 个字符转 LP3
        MOV     A,       R3
        SJMP    LP2
LP3:     MOV     A,       #0AH           ; 送回车换行符
        LCALL   PRT
HERE:    SJMP    HERE
PRT:     PUSH    DPH
        PUSH    DPL
PRT1:    JB      P3.0,    PRT1          ; 没准备好等待
        MOV     SBUF,    A             ; 打印数据送串口缓冲区
WAIT:    JNB     TI,      WAIT         ; 等待发送完成
        CLR     TI             ; 清发送完成标志
        POP     DPL
        POP     DPH
        RET
TAB:     DB 32H,30H,31H,37H,8CH,38H,8DH,38H,8EH

```

C51 语言程序清单:

```

#include<reg51.h>
#define uchar unsigned char
sbit busy=P3^0;
uchar code prt1[16]= {5,2,6,8,5,9,4,2,2,1,0,0,9,6,3,0}; //待打印数据
uchar code prt2[9]= {'2','0','1','7','0x8c','8','0x8d','8','0x8e'}; //待打印字符 ‘2017 年 8 月 8 日’

```

```

void send_char_com(unsigned char ch)
{
    while(busy);
    SBUF=ch;
    while(TI==0);
    TI=0;
}
void main(void)
{
    uchar i,temp;
    SCON=0x40;
    TMOD=0X20;
    TH1=0xFD;           //串口波特率设置为 9600，与打印机的波特率相匹配
    TR1=1;
    for(i=0;i<16;i++)
    {
        temp = prt1[i]+0x30 ;    //取出数据并变换为 ASCII 码
        send_char_com(temp);     //打印一个数据或字符
    }
    send_char_com(0x0A);         //送换行命令代码
    for(i=0;i<9;i++)
    {
        temp = prt2[i];          //取出字符或数据
        send_char_com(temp);     //打印一个数据或字符
    }
    send_char_com(0x0A);         //送换行命令代码
    while(1){};
}

```

思考与练习

1. 用串行口扩展 4 个发光数码管显示电路，编程使数码管轮流显示“ABCD”和“EFGH”，每秒钟变换一次。
2. 试说明非编码键盘的工作原理。为何要消除键抖动？又为何要等待键释放？
3. 设计一个用 AT89S51 单片机控制的显示与键盘应用系统，要求外接 4 位显示器、4 个按键，试画出该部分的接口逻辑电路，并编写相应的显示子程序和读键盘的子程序。
4. 常见的功率驱动器件有哪几种？各有什么特点？
5. 单向可控硅与双向可控硅的导通条件、关断条件有什么不同？
6. SSR 与电磁式继电器相比有哪些优点？怎样用单片机控制 SSR 与电磁式继电器工作？

第 11 章 单片机应用系统的设计与开发

单片机嵌入式系统现在已越来越广泛地应用于人类生活、工作的众多领域，这说明它和我们每一个人的工作、生活都密切相关。现在是一个万众创新、大众创业的时代，也说明我们每一个人都有可能和有机会利用嵌入式系统去改造身边的仪器、产品、工作与生活环境，这是一个可以让你的创意得到充分发挥的平台。由于它的应用领域很广，因此应用系统的硬件与软件设计差别很大，但总体设计方法和研制步骤却基本相同。本章将针对大多数应用场合，简要介绍单片机应用系统的一般开发、研制方法及其所用的开发工具，并结合实例说明开发设计过程。

11.1 应用系统研制过程

所谓应用系统，就是利用单片机为某应用目的所设计的专门的单片机系统（在调试过程中通常称为目标系统）。

像一般的计算机系统一样，单片机的应用系统也是由硬件和软件所组成的。硬件指由单片机，扩展的存储器，输入、输出设备，控制设备，执行部件等组成的系统。软件是各种工作程序的总称。硬件和软件只有紧密配合，协调一致，才能组成高性能的单片机应用系统。在系统的研制过程中，软、硬件的功能总是在不断地调整，以便相互适应，相互配合，达到最佳性能价格比。

单片机应用系统的研制过程包括总体设计、硬件设计、软件设计、在线调试等几个阶段，但它们不是绝对分开的，有时是交叉进行的。

11.1.1 总体方案设计

总体方案设计主要包括如下几方面内容。

1. 确定系统的设计方案和技术指标

在开始设计前，必须根据应用系统的任务要求和系统的工作环境状况等，综合考虑系统的先进性、可靠性、可维护性和成本、经济效益，再参考国内外同类产品的资料，提出合理可行的设计方案和技术指标，以达到最高的性能价格比。

2. 选择机型

机型选择要根据应用系统的要求，选择技术先进，且最容易实现产品技术指标的机种，尽可能把要求的功能集成在一个单片机芯片上，即尽可能选择存储器容量、I/O 口数量、定时器数量及 A/D、D/A 等片上外设能同时满足要求的单片机，当然，还要考虑有较高的性能价格比和有稳定、充足的货源。同时考虑开发团队本身的实际特点，对机型的熟悉程度、时间要求和芯片开发商的技术支持等。

3. 硬件和软件功能的选择与配合

系统硬件的配置和软件设计是紧密联系在一起的，通常软件是建立在硬件基础上的，但在某些场合，同样的功能可以通过硬件也可以通过软件实现，即它们具有一定的互换性。例如日历时钟的产生可以用独立的时钟电路芯片，也可以用单片机内部的定时器通过软件设计为时钟，还可以选择具有片上实时时钟的单片机。当用硬件完成某种功能时，可以减少软件研制的工作量，但增加了硬件成本。若用软件代替某些硬件的功能，可以节省硬件开支，提高可靠性，却增加了软件的复杂性，且要占用 CPU 的工作时间。由于软件是一次性投资，因此在研制产品批量比较大的情况下，能够用软件实现的功能尽量由软件来完成，以便简化硬件结构，降低生产成本。到底采取哪种方法更合适，需具体问题具体分析。

11.1.2 硬件设计

硬件设计的任务是根据总体设计要求，在所选择机型的基础上，确定系统扩展所要用的存储器、I/O 电路、A/D 电路以及有关外围电路等，然后设计出系统的电路原理图。

下面介绍硬件设计的各个环节。

1. 存储器

由于目前单片机片内存储器的容量越来越大，已能满足用户程序容量的要求，所以现在已不再用外扩程序存储器。对于数据存储器的容量要求，各个系统之间差别比较大。有的测量仪器和仪表只需少量的 RAM 即可，此时应尽量选用片内 RAM 容量能符合要求的单片机。对于要求较大容量 RAM 的系统，对 RAM 芯片的选择原则是尽可能减少芯片的数量。例如，选一片 62 256 (32KB) 比选 4 片 6264 (8KB) 价格低得多，连线也简单。也可以选择采用串行总线的 EEPROM 保存需要掉电保护的数据。

2. 选择扩展外围芯片

在很多情况一个单片机芯片不能满足应用系统的要求，此时需要对系统进行扩展，应根据系统总的输入输出要求选择适当的外围芯片，并设计接口电路。扩展方法见第 9 章。

随着集成电路的发展，出现了多种复杂可编程逻辑器件如 CPLD (Complex Programmable Logic Device)、FPGA (Field Programmable Gate Array) 等。这些器件可以由用户根据自己的需要定义其逻辑功能，是一种专用集成电路领域中的硬件可编程电路。使用这类器件可提高系统集成度，减小系统体积，在设计外围电路时要注意尽可能选择这样的器件。

对于 A/D 和 D/A 电路芯片的选择原则应根据系统对它的通道数、速度、精度和价格的要求而确定。在要求不高的情况，可直接选择本身有此功能的单片机。

3. 测控通道外围设备和电路的配置

除了单片机系统以外，按系统功能要求，系统中可能还需要配置键盘、显示器、打印机等外部设备。这些部件的选择应符合系统精度、速度、体积和可靠性等方面的要求。

在测量和控制系统中，经常需要对一些现场物理量进行测量或者将其采集下来进行信号处理之后再反过来去控制被测对象或相关设备。在这种情况下，嵌入式应用系统的硬件设计就应包括与此有关的传感器、隔离电路、驱动电路和输入输出接口电路等。

图 11-1 为一个典型的比较全面的单片机测控系统，图中，左边为单片机扩展的外设、功能芯片及存储器等。它们各自都通过相应的接口与单片机的内部总线相连，如果单片机接口够用也可以直接相接。右边是输入输出通道，为被测控对象，总称为用户。按被测物理量的特点它们可分为如下三种形式：

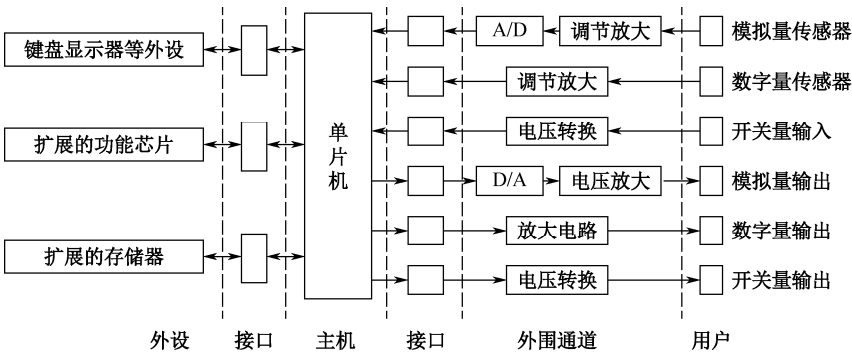


图 11-1 典型单片机测控系统框图

（1）模拟量

模拟量是连续变化的物理量。这些物理量可能是电信号，如电压、电流等，或非电信号，如压力、拉力、温度等。对于非电信号首先要转换为电信号，此时就要用传感器来实现转换（传感器是把其他非电信号转换成相应比例关系的电信号的仪表或器件）。详见有关非电量测量的书籍。

（2）数字量

数字量一般是常见的频率信号或脉冲发生器所产生的电脉冲，这些信号如果不符合 TTL 电平的要求，则需要先进行转换，然后才能输入。但是像串行口信号及某些数字式传感器计数的数字量，所传输的信息为有序组合的“0”、“1”两种 TTL 电平状态，则可以不经转换直接输入。

（3）开关量

开关量是指如按键开关、行程开关等接点通、断时产生的突变电压信号。

图 11-1 右上方的 3 条外围通道是作为输入到单片机去的通道。第 1 条因为是模拟量，要送到单片机中去，所以外围通道中的主要器件是模/数转换器（A/D），此信号一般经信号调节放大处理使之符合 A/D 输入的要求，才能送入 A/D 转换器。第 2 条从用户来的信息已是数字量，则可不用 A/D 转换器，此时只需将数字量信号调节为与接口电路（通常为计数器）的要求相适配即可。第 3 条用户来的信息是开关量，必须将其转换成稳定的接口能接收的直流电平。

图 11-1 右下方的 3 条外围通道是由单片机输出去控制用户（控制对象）的，视被控制装置的类型，可以有模拟量输出、数字量输出以及开关量输出。

这些信号在送到用户装置以前，一般也都要经过信号调节，才能驱动外部设备。

由此可见，当设计一个单片机的测控系统时，还需设计相关的外围电路，如信号调节放大电路、驱动电路等。

4. 硬件可靠性设计

单片机应用系统的可靠性是一项最重要最基本的技术指标，这是硬件设计时必须考虑的

一个指标。

可靠性通常是指在规定的条件下，在规定时间内完成规定功能的能力。

规定的条件包括环境条件（如温度、湿度、振动等）、供电条件等；规定时间一般指平均故障时间、平均无故障时间、连续正常运转时间等。所规定的功能随单片机应用系统的不同而异。

单片机应用系统在实际工作中，可能会受到各种外部和内部的干扰，特别是单片机的测控系统常常工作在环境恶劣的工业现场，此时非常容易受到电网电压、电磁辐射、高频干扰等的影响，使系统工作产生错误或故障。为减少这种错误和故障，需要采取各种提高硬件可靠性的措施，使系统有较强的抗干扰能力。常用以下措施。

（1）提高元器件的可靠性

在系统硬件设计和加工时应注意选用质量好的电子元器件、接插件，并进行严格的测试、筛选和老化处理。设计时技术参数（如负载）应留有余量。

（2）提高印制电路板和组装的质量，设计电路板时布线及接地方法要符合要求

特别要注意以下两点。

- 地线与电源线应适当加粗，数字地、模拟地应尽量远离，且单独走线，最后在一点共地，地线最好设计成网格状。
- 在印制电路板的各个关键部位和芯片上应配置去耦电容，例如，在电源输入端跨接一个 $100\mu\text{F}$ 的电容，在每个集成电路芯片的电源端配置一个 $0.01\mu\text{F}$ 的电容。

（3）对供电电源采取抗干扰措施

- 选带屏蔽层的电源变压器。
- 加电源低通滤波器。
- 可根据情况选择滤波效果更好的开关电源供电。
- 电源变压器的容量应留有余地。

（4）输入、输出通道采取抗干扰措施

- 采用光电隔离电路，用光电隔离器作为数字量、开关量的输入、输出，这种隔离电路效果很好。
- 采用正确的接地技术。
- 采用双绞线或者屏蔽电缆作为信号传输线，可以获得较强的抗共模干扰能力。

硬件电路系统设计主要包括电子线路的原理图设计和 PCB 图的设计。市场上有多种软件可以完成电路设计，如 Altium Designer(Protel)、PowerPCB、AutoCAD 等。近年在国内开始推广的 Proteus 软件，不仅具有其他 EDA 工具软件的仿真功能，还能仿真单片机及外围器件，是目前比较好的仿真单片机及外围器件的工具，已经在 5.5 节做过介绍。

11.1.3 软件设计

在单片机应用系统研制中，软件设计一般是工作量最大、任务最重的环节。下面介绍软件设计的一般方法与步骤。

1. 系统定义

系统定义是指在软件设计前，首先要进一步明确软件所要完成的任务，然后结合硬件结构，进一步厘清软件承担的任务细节。

- 定义和说明各输入/输出口的功能。
- 在程序存储器区域中，合理分配存储空间。
- 在数据存储器区域中，定义数据暂存区标志单元等。
- 面板开关、按键等控制输入量的定义与软件编制密切相关，所以事先也必须给以定义，作为编程的依据。

2. 软件结构设计

合理的软件结构是设计出一个性能优良的单片机应用系统软件的基础，必须予以充分重视。

由系统的定义，可以把整个工作分解为几个相对独立的操作，根据这些操作的相互联系及时间关系，设计出一个合理的软件结构。

在程序设计方法上，模块程序设计是单片机应用中最常用的程序设计技术。这种方法是把一个完整的程序分解为若干个功能相对独立的较小的程序模块，对各个程序模块分别进行设计、编制和调试，最后将各个调试好的程序模块连成一个完整的程序。

3. 程序设计

在软件结构设计确定之后就可以进入程序设计了，一般设计过程首先根据问题的定义，描述出各个输入变量和输出变量之间的数学关系，即建立数学模型。然后根据系统功能及操作过程，先列出程序的简单功能流程框图（粗框图），再对粗框图进行扩充和具体化，即对存储器、寄存器、标志位等工作单元做具体的分配和说明。把功能流程图中每一个粗框转变为具体的存储单元、寄存器和 I/O 口的操作，从而绘制出详细的程序流程图（细框图）。

在完成流程图设计以后，便可编写程序。单片机应用程序可以采用汇编语言，也可以采用某些高级语言，编写完后均须汇编成 80C51 的机器码，经调试正常运行后，再固化到非易失性存储器中，完成系统的设计。

4. 软件可靠性设计

软件可靠性设计通常也称为软件抗干扰设计，是系统抗干扰设计的重要一环。在很多情况下，系统的干扰是不可能完全靠硬件来解决的，因而软件可靠性设计是不可缺少的一环。单片机在系统运行过程所受到的干扰，多数情况都通过软件执行的混乱反映出来，通常简称此现象为程序跑飞。为解决此问题，除了要采取一些硬件措施解决之外，还可采用以下软件可靠性设计方法。

（1）指令冗余技术

在软件设计时，应多采用单字节指令，并在关键的地方人为地插入一些单字节指令（NOP），或将有效单字节指令重复书写，这就是指令冗余。

其一般插入原则为：

- 在跳转或多字节指令前插入，保证指令的正确执行。
- 在比较重要的指令前插入，如中断、堆栈等。
- 在程序中每隔若干条指令插 1 次。

（2）软件陷阱

若 CPU 受干扰，造成程序跑飞到非程序区，此时软件冗余无能为力。可在非程序区设置

拦截措施，使程序进入陷阱，强迫程序进入一个指定的地址，执行一段专门对程序出错进行处理的程序。软件陷阱由 3 条指令构成，ERR 为指定出错处理程序的入口地址：

```
NOP  
NOP  
LJMP ERR
```

软件陷阱安排在下列 3 种地方：

- 未使用的中断区。
- 未使用的 ROM 空间。
- 程序区。当程序执行到 LJMP、SJMP、RET、RETI 等无条件转移类指令时，PC 的值应产生正常的跳变，此时程序不可能继续往下顺序执行。若在这些指令之后设置软件陷阱，就是可拦截跑飞到这儿的程序，而又不影响正常执行流程。

软件陷阱安排在正常程序执行不到的地方，不影响程序执行的效率，在存储器容量允许的条件下，多设置软件陷阱有百利而无一害。

(3) 看门狗（WDT）技术

看门狗是一个通过软、硬件相结合的重要的常用抗干扰技术。

当程序跑飞到一个临时构成的死循环时，冗余指令和软件陷阱都将无能为力，系统将完全陷入瘫痪。看门狗能监视系统的运行状况，并在干扰使程序跑飞的情况下，退出死循环，并使程序转向出错处理程序。

11.2 开发工具和开发方法

一个单片机应用系统从提出任务到正式投入运行的过程，称为单片机的开发。开发过程所用的设备即开发工具。

11.2.1 开发工具

如前所述，单片机本身只是一个电子元件，只有当它和其他的器件、设备有机地组合在一起，并配置适当的工作程序后，才能构成一个单片机应用系统，完成特定的功能，因此单片机的开发包括硬件和软件两个部分。通常，新研制的系统是不可能一次就成功的，多少都会有一些错误。此时就需要通过一步一步调试程序发现系统在硬件和软件上的错误，但是单片机本身没有自开发功能，必须借助于开发工具来排除应用系统（指调试中的目标系统）样机中的硬件故障，生成目标程序，并排除程序错误。当目标系统调试成功以后，还需要用开发工具把目标程序固化到单片机内部或外部程序存储器中。

现代计算机系统的硬件和软件调试，仅靠万用表和示波器等常规工具是不够的，通常要采用自动化调试手段，即用计算机来调试单片机，单片机的开发调试技术主要有仿真技术和监控程序调试技术，通常简称这 2 种开发工具为仿真器和调试器。下面分别予以介绍。

1. 仿真器

仿真器通常是一个特殊的计算机系统，通称单片机仿真系统，简称仿真器，从单片机诞生开始就出现了仿真器。图 11-2 就是一个典型的单片机仿真系统连接示意图。图中的编程器部分不是每一个仿真器必带的，有很多编程器是单独出售的。图中的串行线可以是 RS-232，

也可以是 USB。

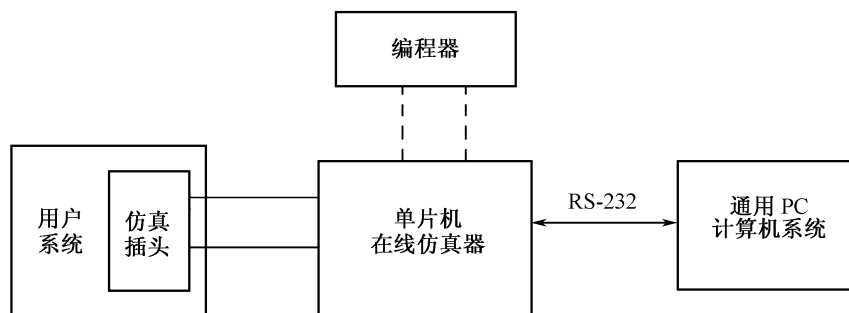


图 11-2 单片机仿真系统连接示意图

单片机仿真系统和一般通用计算机系统相比，在硬件上增加了目标系统的在线仿真器、编程器等部件，所提供的软件除有类似一般计算机系统的简单操作系统之外，还增加了目标系统的汇编和调试程序等。由图 11-2 可见，仿真器是通过串行口与 PC 机相连的，用户利用仿真软件可以在 PC 机上编辑、修改源程序，然后通过汇编软件生成目标码，再传输到仿真器，之后就可以开始调试了。在调试用户系统时，必须把仿真插头插入用户系统的单片机插座上。调试完毕，把仿真插头拔出，换上已经装入用户程序的单片机。

单片机仿真系统的功能主要有：程序编辑与编译，在线仿真，调试功能。下面主要介绍在线仿真功能。

在线仿真器的英文名为 In Circuit Emulator，简称 ICE，是由一系列硬件构成的设备。开发系统中的在线仿真器应能仿真应用系统（也称为目标系统）中的单片机，并能模拟应用系统的 ROM、RAM 和 I/O 口以及外部可扩充的数据存储器地址空间，使应用系统能根据单片机固有的资源特性进行硬件和软件的设计。

在线仿真时，开发系统应能将在线仿真器中的单片机完整地出借给应用系统，不占用应用系统单片机的任何资源，使应用系统在联机仿真和脱机运行时的环境（工作程序、使用的资源和地址空间）完全一致，即运行环境完全“逼真”，实现完全的一次性仿真（需注意，有些仿真器不能 100%地仿真），使用户在应用系统样机还未完全配置好以前，便可以借用开发系统提供的资源进行软件的开发。

因为在研制应用系统开始的初级阶段，应用程序还未生成，更谈不上已固化的应用程序。因此，用户的应用程序必须存放在仿真器的 RAM 存储器内，以便于在调试过程中对程序进行修改。仿真器能出借给用户的作为应用系统程序存储器的 RAM，我们称之为仿真 RAM。仿真器中仿真 RAM 的容量和地址映射和应用机系统完全一致，并保持原有复位入口和中断入口地址不变。

2. 调试器

仿真器的应用比较成熟也比较广泛，通常是针对单片机的某一种具体型号，所以没有通用性，一旦更换单片机就需要更换仿真器。仿真器的价格是比较昂贵的，此外，仿真器并不能完全仿真目标单片机的所有行为，例如时钟和复位特性。另外，仿真插头一般只适用于直插式芯片，而现在已经出现的很多表面贴装式封装的芯片很难组合使用。为了解决上述问题，出现了基于监控程序的调试技术。这种技术可实现直接在目标机上进行调试和运行程序，通常称为在线调试。监控程序通常要占用一定的单片机资源，一般为几 K 字节闪存和几十字节

的 RAM。每次 CPU 启动后，监控程序将首先获得 CPU 的控制权。这种技术可实现直接在目标机上进行调试和运行程序，通常称为在线调试。监控程序的主要功能是通过某种通信方式（不同的单片机是不同的，且所采用的引脚及数量也不同，例如 ST 公司的 ST7 系列采用并行口通信，而 Freescale 公司的 HC08 是通过 RS-232 串口通信，还有用 USB 口的）从 PC 机把程序传入单片机的存储器，并能实现仿真器的主要调试功能。其实，调试器主要是由一个 PC 机和单片机相连的专用连接器和相关的软件组成的（通常厂家为用户使用方便都配备了一些外设），因而其价格很便宜，且在 PC 机软件的支持下，可直接对目标系统的单片机进行在系统动态仿真调试，不再需要仿真插座，可以直接配置与修复目标单片机内部资源（即定时器、I/O 口等），直接方便地擦除与下载单片机应用程序。图 11-3 即为单片机调试器的连接示意图，图中的插头是专用连接器的插头，尽管这种调试器的功能不如一般的通用仿真器强（例如目前多数调试器都只能设一个断点），且监控程序通常要占用单片机内一定的系统资源（一般为几 KB 闪存和几十字节的 RAM），但它价格低廉，可以直接把程序固化到单片机的闪存中，节省了编程器，因而随着这种技术的完善和发展，仿真器将逐步退出单片机的调试市场。

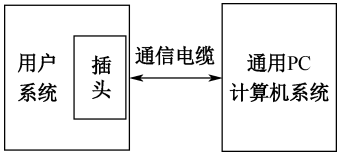


图 11-3 单片机调试器连接示意图

不管是采用仿真器还是采用调试器，在使用前都需要在通用 PC 机中装入用于集成开发环境的软件平台。

除此还可以采用 ISP（In System Programming）技术，在 PC 机上运行一个下载软件，提过串口通信把程序目标码传输到已经安装到应用系统的单片机内部的 Flash 存储器，运行程序观察结果。只要保留单片机对外的这个串口，对于已经编程的单片机就可以利用 ISP 技术反复擦除和再编程，直至运行结果正确。这种方法可以不用仿真器和调试器，但因为不能对用户程序采取单步、断点及跟踪等方法，所以不容易发现程序中的问题。

11.2.2 开发方法

在完成了用户系统样机的组装和软件设计以后，便进入系统的调试阶段。用户系统调试步骤和方法基本是相同的，但具体细节则和所采用的开发工具以及用户系统选用的单片机型号有关。单片机的开发方法实际上就是如何对一个新的应用系统进行调试，一般调试包括硬件调试和软件调试，通常是先排除系统中明显的硬件故障后才和软件结合起来调试。

在正式调试前可以先利用 Proteus 软件设计出系统的硬件原理电路，然后编写程序，在 Proteus 环境下仿真调试通过，根据仿真结果设计实际的硬件电路。这种方法可以在没有硬件的情况下比较快捷的发现设计中的大部分问题，节约了调试时间。但由于该软件不能对硬件电路的各部件进行诊断，也不能进行实时在线仿真，所以最后还是需要采用仿真器、调试器等方法做最后调试。

1. 硬件调试

在进行硬件调试时首先要排除常见的硬件故障，包括逻辑错误、元器件失效、电源故障、可靠性差等问题，然后再进行脱机调试和联机调试。脱机调试是在样机加电之前，先用万用表等工具，根据硬件电气原理图和装配图仔细检查样机线路的正确性，并核对元器件的型号、

规格和安装是否符合要求。应特别注意电源的走线，防止电源之间的短路和极性错误，并重点检查扩展系统总线是否存在相互间的短路或与其他信号线的短路。联机前先断电，把仿真器的仿真插头插到样机的单片机插座上，检查开发机与样机之间的电源、接地是否良好。一切正常，即可接通电源。

通电后仿真器开始工作，对用户样机的存储器、I/O 端口进行读写操作、逻辑检查，如有故障，可用示波器观察有关波形（如选中的译码输出波形、读写控制信号、地址数据波形以及有关控制电平）。通过对波形的观察分析，寻找故障原因，并进一步排除故障。可能的故障有：线路连接上有逻辑错误、有断路或短路现象、集成电路失效等。在用户系统的样机（主机部分）调试好后，可以插上用户系统的其他外围部件，如键盘、显示器、输出驱动板、A/D 板、D/A 板等，再对这些部件进行初步调试。

2. 软件调试

软件调试与所选用的软件结构和程序设计技术有关。如果采用模块程序设计技术，则逐个模块调好以后，再进行系统程序总调试。

对于模块结构程序，要一个个子程序分别调试。调试子程序时，一定要符合现场环境，即入口条件和出口条件。调试的手段可采用单步运行方式和断点运行方式（详见 5.4 节），通过检查用户系统 CPU 的现场、RAM 的内容和 I/O 口的状态，检测程序执行结果是否符合设计要求。通过检测，可以发现程序中的故障、软件算法及硬件设计错误等。

在调试过程中不断调整用户系统的软件和硬件，逐步通过一个个程序模块。各程序模块通过后，可以把有关的功能模块联合起来一起进行整体程序综合调试。在这阶段若发生故障，可以考虑各子程序在运行时是否破坏现场，缓冲单元是否发生冲突，标志位的建立和清除在设计上是否有失误，堆栈区域是否有溢出，输入设备的状态是否正常等等。

通过单步和断点调试后，还应进行连续调试，这是因为单步运行只能验证程序的正确与否，而不能确定定时精度、CPU 的实时响应等问题。待全部调试完成后，应反复运行多次，除了观察稳定性之外，还要观察用户系统的操作是否符合原始设计要求、安排的用户操作是否合理等，必要时还要做适当修正。

在全部调试和修改完成后，将用户软件固化于程序存储器中，对于采用仿真器的系统需要把编程后的单片机插入用户样机后，用户系统才能脱离开发工具独立工作。对于采用调试器的系统，则系统可以立即正常工作，至此单片机应用系统开发研制完成。

当熟练掌握了一种单片机的开发和应用之后，在开发其他型号单片机时也可以不再专门购买这种型号单片机的仿真器，因为现在生产的单片机都具有 Flash 存储器，且大多数都具有 JTAG 接口，因而可以方便地多次在线擦除和下载程序。本书介绍的 AT89S51/52 就具有 JTAG 接口，但均没有在线调试功能，所以在开发初期还需要用仿真器。

11.3 恒温箱温度控制监测系统

温度是工作和日常生活中经常遇到的物理量，例如环境温度、人体温度等。本节将介绍一个简单的由电炉加热的恒温箱温度测控报警系统，要求温度控制在 80℃ 左右，当温度在 78℃~82℃ 时绿色指示灯亮，当温度超过 82℃ 或低于 78℃ 时，红色指示灯亮，并且发出报警声。当温度超过 82℃ 关闭电炉，而当温度低于 78℃，接通电炉加热。要求用 4 个数码管显示，

前三位数码管显示温度，温度显示范围为 00.0℃~99.9℃，数码管的最后一位显示“C”。

11.3.1 题目分析

根据本题的要求，选用 AT89S51 单片机为控制器组成温度测控报警系统。本例题中温度信号首先要经过温度传感器和信号调理器变换为 0~5V 的电压信号，然后此信号再送入 ADC0809 芯片进行模/数转换，转换后的数字量送入单片机，测量结果经过处理后用 LED 显示。如果温度传感器的线性工作范围为 0℃~100℃，工作范围内对应的输出电压是 0~5V，则当采用 8 位的 A/D 转换器对传感器的输出电压量化时，量化值 0~255 对应的输入电压范围为 0~5V，也就是对应于温度 0℃~100℃。如果输入量是线性变化的，则 50℃时传感器的输出是 2.5V，对应 A/D 转换器的输出值大约为 7FH；100℃时传感器的输出是 5V，对应 A/D 转换器的输出值为 FFH。如果要显示实际温度值，经过 A/D 转换后的值还需要进行物理量与数字量的变换，通常称为标度变换。对于本例题的情况，标度变换值应该为 $B=99.9^{\circ}\text{C}/255$ ，如果 A/D 采集的数字值用 D 表示，则变换后的温度值为 $T=D\times B$ ，这就是准备显示的数字量。

温度超过 82℃或低于 78℃时要求报警，因而这是两个用于比较的值。按照上述公式，在 82℃时，相应的数字量为 209，在 78℃时，相应的数字量为 199。

11.3.2 硬件设计

根据本项目的要求，硬件电路设计如图 11-4 所示，图中用 AT89S51 的 P1.4 接绿色指示灯，P1.3 接红色指示灯，P1.1 输出驱动蜂鸣器报警，P1.2 接继电器，控制电炉开关的通断。

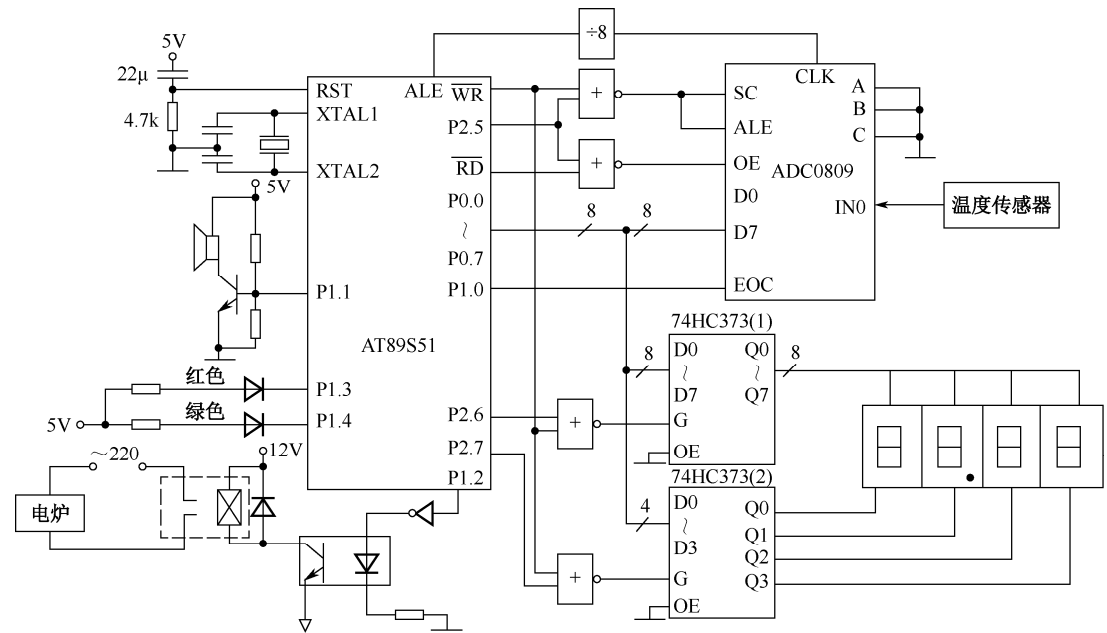


图 11-4 电炉温度监控系统原理图

选择 ADC0809 的 IN0 通道输入温度传感器的信号，因为温度信号只有一路，因此，采用将 ADC0809 的地址 A、B、C 端全部接地的方法，用硬件选择 0 通道可以简化电路。转换

启动信号（SC）和地址锁存信号（ALE）连接在一起，由 $\overline{\text{WR}}$ 信号和 A/D 转换器地址共同控制 ADC0809 IN0 的选通。按图 11-4 中的连接情况，通道 IN0 地址为 0DFF8H（因为 A、B、C 端已经共地，其低 4 位也可以为其他值）。

本例题采用查询方式判断 A/D 是否转换完毕，查询到 A/D 转换完成后，进行数据的读操作。当 A/D 转换器地址和 $\overline{\text{RD}}$ 信号同时有效时选通 OE，转换数据送至数据总线，由 AT89S51 读入。

测量结果采用共阴极数码管动态扫描显示，其中两片 74HC373 分别用来锁存 LED 的段码和位选通信号，74HC373（1）选通地址为 0BFFFH，74HC373（2）选通地址为 7FFFH。因为 74HC373 的驱动能力不太大，所以在对显示亮度要求高的场合还需要再加驱动。

4 位数码管显示缓冲区的存储单元设为单片机内部 RAM 40H~43H。

设晶振频率为 6MHz，单片机的 ALE 脉冲经 8 分频后接入 ADC0809 的 CLK 引脚。

11.3.3 软件设计

程序设计工作的主要任务是已把已经转换为电压量的温度信号经 A/D 转换变成数字量，进行判断和报警等处理，然后再通过编程计算得到温度值的 BCD 码，最后送 LED 显示。按上述工作原理和硬件结构设计的主程序框图如图 11-5 所示。

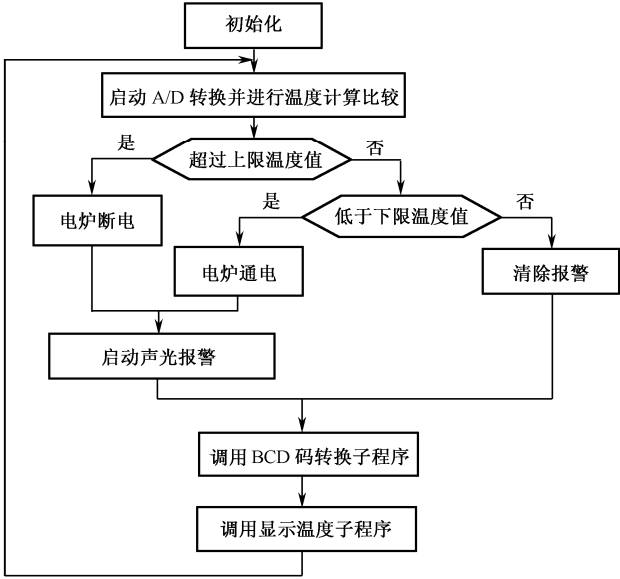


图 11-5 温度采集控制系统主程序框图

C 语言主程序清单如下：

```
#include <reg51.h> //51 库函数定义
#include <absacc.h> //宏定义，可访问绝对地址
#define uchar unsigned char //定义变量
#define TMAX 209 //82 度时对应的 AD 转换器值
#define TMIN 199 //78 度时对应的 AD 转换器值

#define ADC0809_IN0 XBYTE[0x0DFF8] //ADC0809INT0 地址
```

```

#define LEDC  XBYTE[0xBFFF]           //LED 段码地址
#define LEDS  XBYTE[0x7FFF]           //LED 位码地址
uchar DISBUF[4] _at_ 0x40;            //显示数据缓存
uchar xdata *adc_ADR;                 //定义 Ad 地址指针
unsigned int temperature;              //定义整数型温度变量
//位定义
bdata FLAG _at_ 0x21;                 //位地址空间变量
sbit  ALARMU = FLAG^0;                //温度超过上限报警标志位
sbit  ALARMD = FLAG^1;                //温度超过下限报警标志位
sbit  EOC = P1^0;                     //AD 转换完毕标志位
sbit  ALARM = P1^1;                   //报警控制位
sbit  HEAT = P1^2;                    //电炉控制位
sbit  RED = P1^3;                     //红色发光管控制位
sbit  GREEN = P1^4;                  //绿色发光管控制位
unsigned int adc_pro(void);            //数据采集程序
void HextoBCD(unsigned int hex_result); //二进制转换成 BCD 码程序
void disp(void);                      //显示程序
void delay(void);                     //延时程序
unsigned char code Discode[17]={0x3f,0x06,0x5b,0x4f,
                                0x66,0x6d,0x7d,0x07,
                                0x7f,0x6f,0x77,0x7c,
                                0x39,0x5e,0x79,0x71,
                                0x40}; // 0~F 字形码表, 最后一个为 "C"

void main(void)                       //主函数
{
    ALARM = 0;                        //清除声音报警
    ALARMU = 0;                       //清除上限报警
    ALARMD = 0;                       //清除下限报警
    HEAT = 0;                         //不加热
    GREEN = 0;                        //绿灯亮
    RED = 1;                          //红灯灭
    SP = 0x5F;                        //设置堆栈值
    while(1)
    {
        temperature=adc_pro();        //采集温度值, 并进行判断
        if (ALARMU ==1)                //低于温度下限处理
        {
            HEAT = 1;                 //开始加热
            RED = 0;                   //红灯亮
            ALARM = 1;                 //声音报警
        }
        if (ALARMD ==1)                //超过温度上限进行处理
        {
            ALARM = 1;                 //声音报警
            RED = 0;                   //红灯亮
        }
        if(ALARMD&ALARMU ==0)          //温度没有超范围进行处理

```

```

    {
        HEAT = 0;                //不加热
        RED = 1;                 //红灯灭
        GREEN = 0;               //绿灯亮
    }
    HextoBCD(temperature);      //十六进制温度值转为 BCD 码
    DISBUF[3] = 0x10;           //把字母 C 在字形码表位置放在显示缓存 3 中
    disp();                     //数码管上显示温度值
}
while(1);
}

```

下面分别介绍各子函数。

1. 温度采集子函数

温度采集子函数主要完成数据采集、转换，对温度上、下限的比较，根据比较结果设置相应的报警标志位。本函数还要完成物理量到温度数字值的变换，即标度变换。在本例题中，标度变换值应该为 $B=99.9^{\circ}\text{C}/255$ ，为达到转换精度，即为能准确显示到小数点后一位的值，则计算公式应修改为 $T=D\times 10\times B=D\times 10\times 99.9/255$ （ D 为采集的数字量），数据扩大 10 倍计算可避免小数运算，所以变换后的温度实际最大值为 999。

```

unsigned int adc_pro(void)
{
    unsigned char temp1;        //定义一个无符号字符型变量
    unsigned int temp_adult;     //定义一个无符号整型变量
    adc_ADR = &ADC0809_IN0;     //取 ADC 零通道地址
    *adc_ADR = 0;               //向 ADC 写入数据，启动 AD 采集
    while(EOC==1);              //等待 EOC 为低
    while(EOC==0);              //等待 A/D 转换完毕
    temp1 = *adc_ADR;            //读取采集结果放入 temp1 中
    if (temp1 > TMAX) ALARMU = 1; //如果温度值超过上限，高温报警位置 1
    else ALARMU = 0;             //否则清零
    if (temp1 < TMIN) ALARMD = 1; //如果温度值低于下限，低温报警置 1
    else ALARMD = 0;            //否则清零
    temp_adult = temp1*999;       //温度电压值变为温度值
    temp_adult = temp_adult/255;
    return(temp_adult);          //返回温度的十六进制值
}

```

2. 二进制变换为 BCD 码子函数

经过标度变换后的温度值还是十六进制数，需要变换为 BCD 码用于显示。因为测量的温度最大可能值为 99.9°C ，实际需要转换的字长不超过 2 字节，相应的 BCD 码不超过百位。转换完之后，压缩 BCD 码的结果需要占用 3 字节，即温度显示值占用 3 个单元。

```

void HextoBCD(unsigned int hex_result)
{
    DISBUF[0] = (uchar)((hex_result%1000)/100); //取得百位数
    DISBUF[1] = (uchar)((hex_result%100)/10);   //取得十位数
    DISBUF[2] = (uchar)(hex_result%10);         //取得个位数
}

```

3. 显示子函数

由于要求显示的数据精确到小数点后一位，所以测量结果用三位数码管显示，显示格式为 $\times\times.\times C$ （其中 $\times\times.\times$ 为温度值）。第2位数码管在显示时应带有小数点。

在数码管上显示小数点有几种方法，最常用的有3种方法。第一种是硬件方法，即把固定需要显示小数点的那一个数码管的小数点位，根据数码管是共阳极还是共阴极与低电平或高电平相接。第二种是软件方法，即把固定需要显示小数点的那一个数码管的小数点位，根据数码管是共阳极还是共阴极把要显示的数字与3FH相“与”，或者与80H相“或”。第三种也是软件方法，即在显示时采用2个字型表，一个是不带小数点的，另一个是带小数点的，根据需求去取数。本例采用第二种方法。

```
void disp(void)
{
    unsigned char i,j,m,dis_data;
    for (j=0;j<250;j++)
    {
        m = 0xef;                                //显示位码值
        for(i=0;i<4;i++)
        {
            LEDS = m;                             //输出位码
            dis_data = Discode[DISBUF[i]];         //取段码值
            if (i==1) dis_data += 0x80;           //如果为第一位加上小数点
            LEDC =dis_data;                       //输出段码
            delay();                               //延时
            m = (m>>1)+ 0x80;                    //准备下一位显示
            LEDC =0x00;                           //关闭显示
        }
    }
}
```

延时程序略

11.4 PC 机与单片机间的串行通信设计

单片机的主要优点是抗干扰能力强、价格低廉、功耗极小等。它的明显不足是运算功能和显示功能较差。而PC机的特点正好与其相反，因而，在很多工业自动控制场合，把单片机放在环境较恶劣的远程现场，作为从机（也称为下位机），而把PC机放在条件较好的环境中作为中央控制机（也称为上位机）。利用单片机的串行口与PC机的串行口进行串行通信，PC机可对远程前沿单片机进行控制，将单片机采集的数据传送到PC机中，由PC机对数据进行处理和显示，同时把反馈信号发到单片机，实现闭环控制和管理。这样可充分发挥它们的各自优点形成一个优良的控制采集系统，从而具有很广泛的实际应用价值。下面介绍PC机与单片机间串行通信的硬件和软件设计。

11.4.1 RS-232C 标准接口

实现在计算机与计算机（包括单片机）、计算机与外设间的串行通信时，通常采用标准

通信接口。所谓标准通信接口，是明确定义若干信号线的机械、电气特性，使接口电路标准化、通用化，能方便地把不同的计算机、外设等有机地连接起来，进行串行通信。最早出现的标准通信接口 RS-232C 是由美国电子工业协会（EIA）正式公布的，它包括按位串行传输的电气和机械方面的规定，适用于短距离或带调制解调器的通信场合，是在异步串行通信中应用最广的标准总线。本节以 RS-232 为例予以介绍。

RS-232C 接口的具体规定如下。

1. 范围

RS-232C 标准最高的数据速率为 19.2Kb/s。如果不增加其他设备，RS-232C 标准的电缆长度最大为 15m。

2. RS-232C 的信号特性

为了保证二进制数据能够正确传送，有必要使所用的信号电平保持一致。为满足此要求，RS-232C 标准规定了数据和控制信号的电压范围。由于 RS-232C 的电平不是+5V 和地，而是采用负逻辑，规定在+3~+15V 之间的任意电压表示逻辑 0 电平，-3~-15V 之间的任意电压表示逻辑 1 电平，因而采用 TTL 电平正逻辑的器件不能直接与其接口，中间必须进行电平转换。

3. RS-232C 接口信号说明

RS-232C 串行标准接口信号引脚共有 22 个，采用标准 25 芯插头座，但在实际异步串行通信中，并不要求用全部的 RS-232C 信号，现在 PC 机上通常只引出 9 芯插座。因此本节不对全部信号做详细解释。实际上单片机与 PC 机通信时只需要三根线就够了。这三根线分别是信号地（5 脚）、发送数据输出端 TXD（2 脚）和接收数据输入端 RXD（3 脚）。其余信号主要用于双方设备通信过程中的联络（握手信号），而且有些信号仅用于和 MODEM（数据通信装置）的联络。

11.4.2 单片机与 PC 机之间的电平转换芯片

由于 RS-232C 标准采用负逻辑，而一般单片机均为 0~5V 的正逻辑，为了实现单片机与 PC 机的通信，在电路需要采用电平转换芯片，现在一般采用一块芯片实现正电压、负电压的转换。常用美国 MAXIM 公司生产的 MAX232 和 MAX202 等芯片来实现。

MAX232/202 芯片简介如下。

MAX232/202 芯片是包含两路接收器和驱动器的 IC 芯片，适用于各种通信接口。芯片内部有一个电源电压变换器，可以把输入的+5V 电源电压变换成 RS-232C 输出电平所需的±10V 电压。所以，采用此芯片接口的串行通信系统只需单一的+5V 电源就可以了。图 11-6 为 MAX232 芯片的典型工作电路连接方法。由图 11-6 可见，芯片的上半部分为电源变换电路部分，引脚 C1+、C1-等分别与电容的正、负端相接，V+为变换出的+10V 电压，V-为变换出的-10V 电压，通过此两引脚可检测芯片工作是否正常。

在实际应用中，器件对电源噪声很敏感。因此，VCC 必须对地加去耦电容 C5，其值为 1μF。电容 C1，C2，C3，C4 取同样数值的钽电解电容 1μF/16V，用以提高抗干扰能力，在连接时必须尽量靠近器件。与 MAX232 功能基本相同的 MAX202 接法与使用也相同，只是

所用电容参数为 0.1μF。

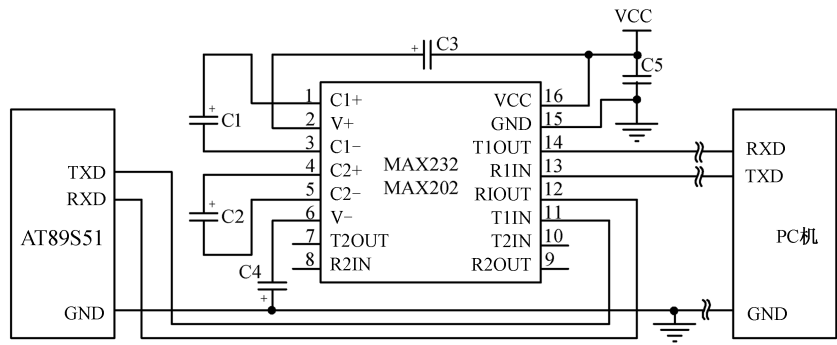


图 11-6 采用 MAX202 接口的串行通信电路图

11.4.3 PC 机与单片机串行通信应用实例

本例题的接口电路如图 11-6 所示。单片机采用 AT89S51，PC 机可以是一般的通用机。从 MAX202 芯片中两路发送、接收中任选一路作为接口。要注意其发送、接收的引脚要对应。如使 T1IN 接单片机的发送端 TXD，则 PC 机的 RS-232 接收端 RXD 一定要对应接 T1OUT 引脚。同时，R1OUT 接单片机的 RXD 引脚，PC 机的 RS-232 发送端 TXD 对应接 R1IN 引脚。

在软件设计时一定要注意单片机与 PC 机之间应该遵守相同的通信协议，主要包括波特率、传输帧格式、校验位等。除此之外，如果要想实现 PC 机与多个单片机的通信，PC 机还应该向单片机发送欲寻单片机的地址编码，而单片机中要编写地址识别程序段。

本例题的通信协议约定如下。

波特率：2400b/s；

帧格式：1 位起始位，8 位数据位，1 位停止位，无奇偶校验；

传送方式：PC 机采用中断方式接收，单片机也采用中断方式接收；

数据长度：1 字节；

校验方式：累加和校验；

握手方式：软件握手。

本例题要求实现一个简单的通信过程，首先 PC 机向单片机发送一个数据请求信号“Q”，单片机收到该请求信号后，开始发送已经准备好的 10 字节数据，最后发送 10 个数据的累加和校验。PC 机在收到发过来的数据和累加和后，与自己的累加和相比较，相同则发送一个“J”的 ASCII 码作为应答信号，表示本次通信结束；不同则在 PC 机上显示出错对话框。如需再次发送，则再向单片机发送数据请求信号“Q”。

1. PC 机的通信软件设计

实现 PC 机串口通信的软件采用 VC++6.0 语言编程，VC 提供了一组系统函数用于支持 Windows 平台下的串口通信，在 msdn 帮助文档中提供了 TTY 方式通信的例子。以这组通信函数为基础，可实现一个用于串口通信的类 CComm，此通信类能够与指定串口关联，向串口发送数据，并且可以检测串口，一旦有数据到来，就会以中断方式将数据读入用户缓冲并向主窗口发送消息，利用它可以方便地编制串口应用程序。

源程序清单如下：

通信类头文件 Comm.h

#define BUFLen 1024

//缓冲长度

#define WM_COMM_READ WM_USER+1000

//用户自定义的数据读入消息

class CComm : public CObject

{

public:

 BOOL VerifyRbuf();

//校验和检查

 void dowithRbuf();

//通信协议应答

 void Write(char cmd);

//发送命令字

 void WriteFormat(char * sbuf, BYTE length);

//打包发送数据

 void Close();

//关闭串口

 void Read();

//从串口读数据

 void Open(UINT com);

//打开串口

 HANDLE m_hCom;

//串口句柄

 CWinThread * m_hThread;

//线程句柄

 BOOL m_bRun;

//是否运行标志

 UINT m_com;

//串口编号

 char m_rbuf[BUFLen];

//输入缓冲

 int m_rbuflen;

//输入数据长度

 char m_sbuf[BUFLen];

//输出缓冲

 DWORD dwLength;

//实际读入（发送）的数据长度

 OVERLAPPED osWrite, osRead;

//读写操作结果

 CComm();

//构造函数

 virtual ~CComm();

//析构函数

};

通信类实现 Comm.cpp

UINT CommWatchProc(VOID* pcom);

CComm::CComm()

{

 m_bRun = FALSE;

 m_hThread = NULL;

 m_rbuflen = 0;

}

CComm::~~CComm()

{

}

void CComm::Open(UINT com)

//初始化串口，参数 com 为打开的串口编号

{

 memset(&osRead, 0, sizeof(OVERLAPPED));

 memset(&osWrite, 0, sizeof(OVERLAPPED));

 //创建读操作系统事件

 osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

 //创建写操作系统事件

 osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

```

if(osRead.hEvent == NULL || osWrite.hEvent == NULL)
    return;                                //创建事件失败返回
m_com = com;                                //记住串口编号
CString str;
str.Format("COM%d", m_com);
m_hCom = CreateFile((LPCTSTR)str,           //打开指定的串口
                    GENERIC_READ | GENERIC_WRITE, //允许读写
                    0,                        //此项必须为 0
                    NULL,                    //无安全属性
                    OPEN_EXISTING,          //设置产生方式
                    FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
                    //使用异步通信
                    NULL );

ASSERT(m_hCom != INVALID_HANDLE_VALUE);     //检测打开串口操作是否成功
SetCommMask(m_hCom, EV_RXCHAR );           //设置事件驱动的类型
SetupComm( m_hCom, 1024,512);              //设置输入、输出缓冲区的大小
PurgeComm( m_hCom, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR
           | PURGE_RXCLEAR );              //清输入、输出缓冲区

DCB dcb;                                   //定义数据控制块结构
GetCommState(m_hCom, &dcb);                //读串口原来的参数设置
dcb.BaudRate = 2400;                        //设置波特率为 2400b/s
dcb.ByteSize = 8;                          //数据位 8 位
dcb.Parity = NOPARITY;                     //无奇偶校验位
dcb.StopBits = ONESTOPBIT;                 //1 位停止位
dcb.fParity = FALSE;                       //不进行奇偶校验
SetCommState(m_hCom, &dcb);                //串口参数设置
m_bRun = TRUE;                             //启动串口读检测
m_hThread = AfxBeginThread(CommWatchProc, this, THREAD_PRIORITY_NORMAL, 0,
                           CREATE_SUSPENDED, NULL);
m_hThread->m_bautoDelete=FALSE;
m_hThread->ResumeThread();
}

void CComm::Read()    //从串口读入数据，存放到了用户自定义缓冲 rbuf 中，并向主窗口发送消息
{
    DWORD nBytesRead, dwEvent, dwError;
    COMSTAT cs;
    while(m_bRun)                                //是否停止检测
    {
        if(WaitCommEvent(m_hCom, &dwEvent, NULL)) //等待数据到来
        {
            if((dwEvent & EV_RXCHAR) == EV_RXCHAR) //是否为数据到来事件
            {
                ClearCommError(m_hCom, &dwError, &cs); //获得数据长度
                if(cs.cbInQue != 0)
                { //读数据
                    ReadFile(m_hCom, m_rbuf + m_rbuflen, cs.cbInQue, &nBytesRead, &osRead);
                    m_rbuflen += cs.cbInQue;           //设置读入数据长度
                }
            }
        }
    }
}

```

```

//向主窗口发送接收到数据的消息
::SendMessage(AfxGetMainWnd()->m_hWnd, WM_COMM_READ, (UINT)this, NULL);
    }
}
}
}
PurgeComm(m_hCom, PURGE_RXCLEAR); //清输入、输出缓冲区
}

void CComm::Close() //关闭串口
{
    m_bRun = FALSE; //置停止检测标志
    if(m_hThread)
    {
        WaitForSingleObject(m_hThread->m_hThread, 1000); //INFINITE); // 等待子线程停止
        m_hThread = NULL;
    }
    CloseHandle(m_hCom); //释放串口句柄
    CloseHandle(osRead.hEvent); //释放读事件句柄
    CloseHandle(osWrite.hEvent); //释放写事件句柄
}

void CComm::Write(char cmd) //发送命令字
{
    WriteFile(m_hCom, &cmd, 1, &dwLength, &osWrite);
}
//向串口发送数据，sbuf 为数据缓冲，length 为数据长度
void CComm::WriteFormat(char * sbuf, BYTE length)
{
    m_sbuf[0] = length; //设置数据长度字
    char sum = 0;
    for(int i=1; i<length+1; i++) //计算累加和
    {
        m_sbuf[i] = sbuf[i];
        sum = sum + sbuf[i];
    }
    m_sbuf[length + 1] = sum; //设置累加和
    WriteFile(m_hCom, m_sbuf, length + 2, &dwLength, &osWrite); //发送数据
}

void CComm::dowithRbuf() //处理读入数据，实现通信协议
{
    if(m_rbuflen < m_rbuf[0] + 2 ) //读入数据是否完整
        return;
    if(VerifyRbuf()) //检验和是否正确
    {
        Write('J'); //向单片机发送检验和正确回应
        //请在此处加入对数据的处理，其中 m_rbuflen 为长度，m_rbuf 为数据缓冲指针
    }
    else printf("The sum is wrong !"); //报错
}

```

```

BOOL CComm::VerifyRbuf()
{
    char sum = 0;
    for(int i=1; i<m_rbuflen - 1; i++)
        sum += m_rbuf[i];
    if(sum != m_rbuf[m_rbuflen-1])           //校验和错误
        return FALSE;
    else
        return TRUE;                         //校验和正确
}
UINT CommWatchProc(VOID* pcom)              //串口读检测线程
{
    CComm * pCom =(CComm *) pcom;
    pCom->Read();
    return TRUE;
}

```

在 Windows 下将上述通信类加入到应用程序中的过程如下。

① 定义通信类对象，调用 open 方法初始化串口。

在你的主窗口类中定义通信类对象：CComm m_com;

在主窗口类的初始化函数中打开串口：m_com.Open(1);

② 向串口发送数据：m_com.Write('S')。

③ 在主窗口消息处理函数中处理接收到的数据。

为主窗口类头文件增加消息响应函数声明。

```
void OnCommRead(WPARAM wParam,LPARAM lParam);
```

在主窗口类的实现中增加消息映射（在 END-MESSAGE-MAP()宏之前）

```
ON_MESSAGE(WM-COMM-READ, OnCommRead)
```

该消息响应函数实现如下：（其中 CMainWnd 为你的主窗口类）

```

void CMainWnd::OnCommRead(WPARAM wParam, LPARAM lParam)
{
    m_com.dowithRbuf();
}

```

2. 单片机的通信软件设计

单片机的通信软件采用 80C51 的 C 语言编写，此软件适用于 51 系列的任何一种型号。单片机的发送和接收采用中断程序。单片机晶振是 11.0592MHz，准备发送的数据存放在代码区。

程序清单如下：

```

#include<reg51.h>
#define uchar unsigned char
//待发送的数据
uchar code send_buf[10]={0x12,0x23,0x34,0x45,0x56,0x67,0x78,0x89,0x90,0x01};
void com_test() interrupt 4      // 串口中断服务函数
{
    uchar addtest,temp,i;

```

```

if(RI==1)                //接收到数据判断
{
    temp=SBUF;            //取数据
    RI=0;                 //接收标志位清零
    addtest=0;            //校验和清零
    if(temp=='Q')         //如果接收到的数据是请求数据的标志
    {
        for(i=0;i<10;i++)
        {
            addtest=addtest+send_buf[i]; //求校验和
            SBUF=send_buf[i];           //发送数据
            while(TI==0);                //等待发送完毕
            TI=0;                        //发送标志位清零
        }
        SBUF=addtest;                    //发送校验和
        while(TI==0);                    //等待发送完毕
        TI=0;                            //发送标志位清零
    }
}
}
}

void main(void)           //主函数
{
    SCON=0x50;             //串口工作在方式 1，接收允许
    TMOD=0X20;             //定时器 1 工作在方式 2，用于串口的波特率发生器
    TH1=0xF4;              //波特率 2400
    TL1=0XF4;
    IE=0X90;               //允许串口中断，开总中断
    TR1=1;                 //启动定时器 1，
    for(;;){ }
}

```

由于 RS-232C 标准在传输距离上的局限性，当距离超过 20m 时必须选用 RS-422A 或者 RS-485 等标准。这些标准在性能上有很大提高，比 RS-232C 标准传输信号距离长、速度快，传输率最大为 10Mb/s，在此速率下，电缆允许长度为 120m；如采用较低传输速率，例如为 90 000b/s 时，最大距离可达 1200m。并且 RS-422A 等标准具有更强的抗干扰能力，还能进行一对多点的通信，虽然在硬件使用上有一点区别，但在软件应用上与 RS-232C 是兼容的，所以在工业控制领域应用广泛。

11.5 步进电机控制设计

步进电机是一种数控电动机，可以对位移量、旋转角度和转动速度等进行高精度的控制，因而在自动控制和精密机械等领域广泛应用步进电机作为单片机控制应用系统中的执行部件。

11.5.1 步进电机的工作原理

步进电机是将电脉冲信号转换成机械角位移的执行部件。步进电机电源一般是采用单极性直流电。当电机某一相绕组通电时，产生磁场，并与转子形成磁路。在磁场的作用下，转子将转动一定的角度，使转子齿与定子齿对齐，从而使步进电机向前“走”一步。每当电机绕组接受一个电脉冲，转子就转过一个相应的步距角，步进电机由此而得名。通过对步进电机的各相绕组按恰当的时序方式通电，就可使其执行步进转动。转子的角位移大小及转速分别与输入的电脉冲数及频率成正比，并在时间上与输入的脉冲同步。只要能正确控制输入的电脉冲数、频率以及电机各相绕组通电的相序，就可以得到所需要的转角、转速及转向。通过单片机很容易实现对步进电机的数字控制。

11.5.2 步进电机的控制方法

通过步进电机的工作原理就可以知道,如果通过单片机按顺序给绕组施加有序的脉冲电流，就可以控制电机的转动，从而实现了数字向角度的转换。转动的角度大小与施加的脉冲频率成正比（不同电机可用频率范围不同），而转动方向则与脉冲的顺序有关。下面以四相步进电机为例，说明步进电机的控制方法。四相步进电机电流脉冲的施加一般有单四拍、双四拍及单、双八拍三种方式。“单”的意思是指每次切换前后只有一相绕组通电；“拍”是指从一种通电状态到另一种通电状态；“双”是指每次有两相绕组通电。设四相步进电机的四相分别为 A、B、C、D。这三种方式的通电时序示意图见图 11-7。图 11-7（a）为单四拍方式，它是按单相绕组顺序通电，有四种通电状态；（b）为双四拍方式，它是按双相绕组顺序通电，有四种通电状态；（c）为单、双八拍方式，它有八种通电状态，是接单、双相绕组轮流交替的方式通电，如图示，其首先是 DA 两相同时通电,然后 A 相单独通电，再使 AB 相同时通电，然后 A 相断电，B 相单独通电，接着 BC 两相同时通电……。

（b）、（c）两种方式在转换时始终保持一个绕组通电，因此步进电机的运行平稳柔和，减少抖动。

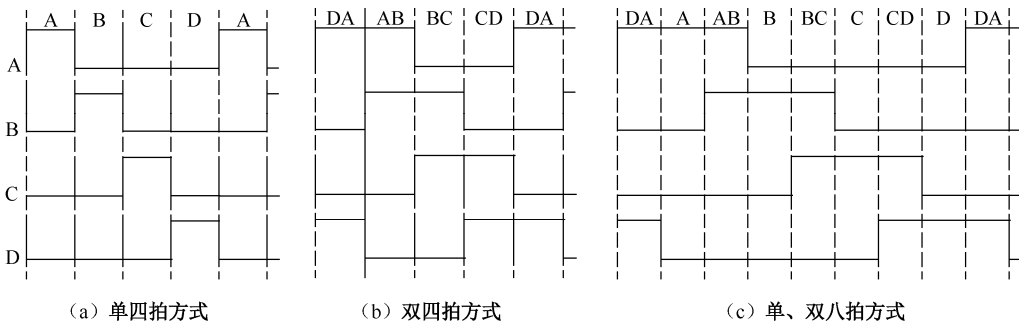


图 11-7 四相步进电机的三种工作方式通电时序

通过单片机用软件方法可方便灵活地控制步进电机的运行状态，且简化了电路，降低了成本。

由于单片机的驱动电流一般都比较小，不能直接驱动电机工作，所以单片机的 I/O 口输出必须接驱动电路才能控制电机工作。步进电机常用的驱动方式有几种，例如，衡电压驱动，高低压驱动、双电压驱动、单电压功率放大电路等多种，具体选择那种驱动方式要根据电机

的性能特点和功率大小而定。

11.5.3 步进电机控制应用举例

本例题要求用步进电机控制一个光电检测器在一螺杆上按某功能要求上、下移动。图 11-8 为此装置示意图。在螺杆上、下各装有一个限位器。当检测器上、下移动至限位器时，则自动停止。

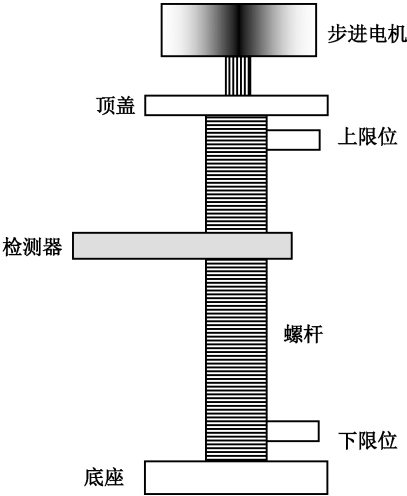


图 11-8 检测器示意图

1. 硬件设计

为简化说明，本例采用一种控制方案最简单的 4 相步进电机，电流脉冲的施加用单 4 拍方式，即上述第 1 种。根据本题目要求设计单片机控制电路如图 11-9。对控制电路作如下说明：

用 AT89S51 单片机作为主机，外加 1 片 ULN2803 用作步进电机的驱动芯片。本系统所使用的晶振为 6MHz。用单片机的 P1.0~P1.3 作为步进电机的控制口，P1.6 和 P1.7 作为上、下限位的输入检测引脚，光电检测器没有到达限位点时为高电平。当光电检测器行进到其中一个限位点时，相应的限位点输入低电平，单片机控制电机停止运动。

2. 软件设计

在定时中断服务程序中控制电机改变步进控制字，因为 ULN2803 是高电平时导通，因而当控制信号为高时，此相有信号输出，电机工作时 4 相控制字将按顺序设为高。电机的上升、下降状态由外部中断程序设定（电路图上的 INT0 和 INT1）。为减少篇幅，本程序中有关的子程序未全列出，只列出定时中断服务程序。图 11-10 为主程序框图，定时中断服务程序框图省略。

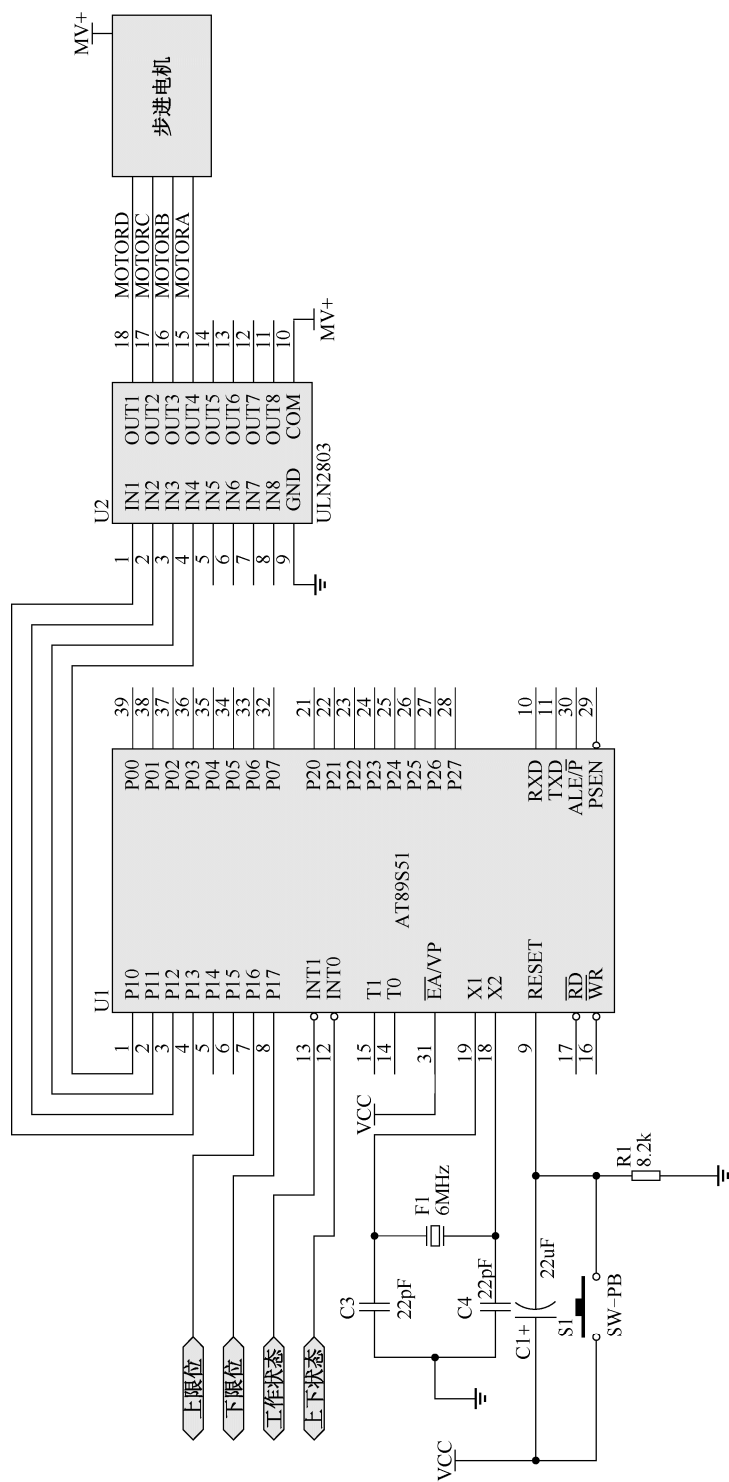


图 11-9 液位检测器控制电路

主程序清单如下：

```
#include<reg51.h>
//定义控制电机的 A、B、C、D 相
sbit MOTORA=P1^0;
sbit MOTORB=P1^1;
sbit MOTORC=P1^2;
sbit MOTORD=P1^3;
//定义上下两个限位的输入口
sbit UP=P1^6;
sbit DOWN=P1^7;
unsigned char CURRENT_M=0;           //记录当前相位
unsigned char MOTOR_WORK=0;         //电机是否工作标志
unsigned char work_dir=0;           //工作方向标志
main()
{
    TMOD=0x01;                       //定时器工作在定时方式 1
    TH0=0xEC;                         //6MHz 晶振，定时 10ms
    TL0=0x78;
    ET0=1;EA=1;                       //开定时器中断，开总中断
    while(1)
    {
        ... ..                        //其他程序
        while(MOTOR_WORK==0);         //等待电机工作
        MOTOR_WORK=0;                //清标志，为下次做准备
        TR0=1;                        //启动定时器
        if(work_dir==1)               //上升
        {
            while(UP==1);             //如上升未到上限位则继续等待
        }
        else                          //下降
        {
            while(DOWN==1);           //如下降未到达下限位则继续等待
        }
        P1=0xff;                     //步进电机停止
        TR0=0;                        //关闭定时器
        ... ..                        //其他程序
    }
}
//定时器 0 中断服务程序
void T0_int() interrupt 1
{
    if(work_dir==1)                   //如果上升，则 4 项依次为 A-B-C-D-A
    {
        switch(CURRENT_M)
        {
            case 0:                   //A 相通电
                MOTORD=0;
                MOTORA=1;
```

```

        break;
    case 1:                                     //B 相通电
        MOTORA=0;
        MOTORB=1;
        break;
    case 2:                                     //C 相通电
        MOTORB=0;
        MOTORC=1;
        break;
    case 3:                                     //D 相通电
        MOTORC=0;
        MOTORD=1;
        break;
    default:
        break;
}
}
else                                           //如果为下降，则电机一次通电为 A-D-C-B-A
{
    switch(CURRENT_M)
    {
        case 0:
            MOTORB=0;
            MOTORA=1;
            break;
        case 1:
            MOTORA=0;
            MOTORD=1;
            break;
        case 2:
            MOTORD=0;
            MOTORC=1;
            break;
        case 3:
            MOTORC=0;
            MOTORB=1;
            break;
        default:
            break;
    }
}
CURRENT_M++;
if(CURRENT_M==4)                             //4 相轮回
    CURRENT_M=0;
TH0=0xEC;                                     //10ms 定时初值
TL0=0x78;
}

```

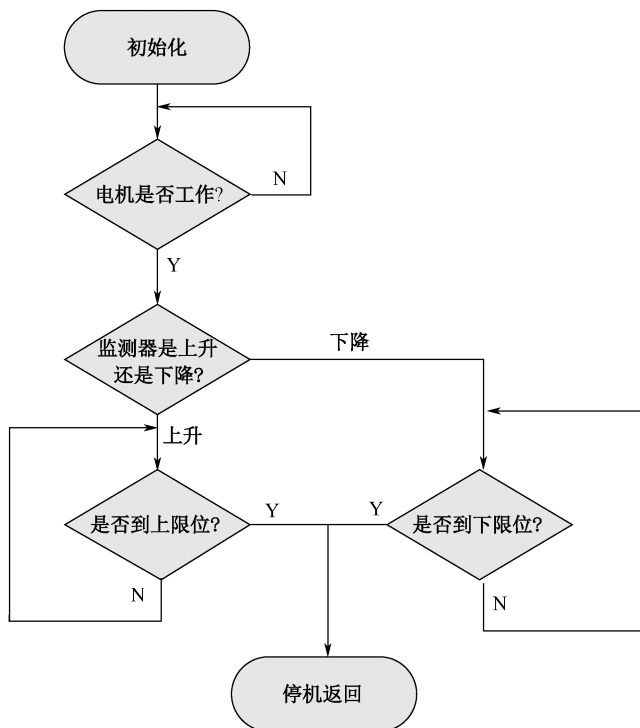


图 11-10 电机控制主程序框图

思考与练习

1. 在单片机应用系统总体设计中，应考虑哪几方面的问题？简述硬件设计和软件设计的主要过程。
2. 如何提高应用系统的抗干扰性？可采取哪些措施？
3. 请自行设计一个节日彩灯循环闪烁的应用系统。
4. 请自行设计一个交通灯控制系统，此系统还要求显示秒倒计时时间，每当还差 10s 该换指示灯时（例如红灯换绿灯），该指示灯变为闪烁点亮。
5. 请自行设计一个温度采集系统，要求按一路/秒的速度顺序检测八路温度点，测温范围为 $+20^{\circ}\text{C} \sim +100^{\circ}\text{C}$ ，测量精度为 $\pm 1\%$ 。要求用 5 位数码管显示温度，最高位显示通道号，次高位显示“—”，低三位显示温度值。
6. 用 PC 机向 AT89S52 单片机发一个启动开始的信号，单片机收到信号后，即开始把内存中 90~A0H 之间的数连续发送到 PC 机中，发送完毕发一个停止信号。要求波特率为 9600b/s，采用 11 位帧格式，用中断方式发送和接收。

附录 A 80C51 指令表

80C51 指令系统所用符号和含义：

addr11	11 位地址
addr16	16 位地址
bit	位地址
rel	相对偏移量，为 8 位有符号数（补码形式）
direct	直接地址单元（RAM、SFR、I/O）
#data	立即数
Rn	工作寄存器 R0~R7
A	累加器
Ri	i=0, 1, 数据指针 R0 或 R1
X	片内 RAM 中的直接地址或寄存器
@	在间接寻址方式中，表示间址寄存器的符号
(X)	表示 X 中的内容
((X))	在间接寻址方式中，表示间址寄存器 X 指出的地址单元中的内容
→	数据传送方向
^	逻辑与
∨	逻辑或
⊕	逻辑异或
√	对标志产生影响
×	不影响标志

十六进制 制代码	助 记 符	功 能	对标志的影响				字节数	周期数
			P	OV	AC	CY		
算 术 运 算 指 令								
28~2F	ADD A, Rn	(A)+(Rn)→A	✓	✓	✓	✓	1	1
25	ADD A, direct	(A)+(direct)→A	✓	✓	✓	✓	2	1
26, 27	ADD A, @Ri	(A)+((Ri))→A	✓	✓	✓	✓	1	1
24	ADD A, #data	(A)+#data→A	✓	✓	✓	✓	2	1
38~3F	ADDC A, Rn	(A)+(Rn)+CY→A	✓	✓	✓	✓	1	1
35	ADDC A, direct	(A)+(direct)+CY→A	✓	✓	✓	✓	2	1
36, 37	ADDC A, @Ri	(A)+((Ri))+CY→A	✓	✓	✓	✓	1	1
34	ADDC A, #data	(A)+#data+CY→A	✓	✓	✓	✓	2	1
98~9F	SUBB A, Rn	(A)-(Rn)-CY→A	✓	✓	✓	✓	1	1
95	SUBB A, direct	(A)-(direct)-CY→A	✓	✓	✓	✓	2	1
96, 97	SUBB A, @Ri	(A)-((Ri))-CY→A	✓	✓	✓	✓	1	1
94	SUBB A, #data	(A)-#data-CY→A	✓	✓	✓	✓	2	1
04	INC A	(A)+1→A	✓	×	×	×	1	1

续表

十六进制 制代码	助 记 符	功 能	对标志的影响				字节数	周期数
			P	OV	AC	CY		
算 术 运 算 指 令								
08~0F	INC Rn	(Rn)+1→Rn	×	×	×	×	1	1
05	INC direct	(direct)+1→(direct)	×	×	×	×	2	1
06, 07	INC @Ri	((Ri))+1→(Ri)	×	×	×	×	1	1
A3	INC DPTR	(DPTR)+1→DPTR					1	2
14	DEC A	(A)-1→A	√	×	×	×	1	1
18~1F	DEC Rn	(Rn)-1→Rn	×	×	×	×	1	1
15	DEC direct	(direct)-1→(direct)	×	×	×	×	2	1
16, 17	DEC @Ri	((Ri))-1→(Ri)	×	×	×	×	1	1
A4	MUL AB	(A)·(B)→BA	√	√	×	0	1	4
84	DIV AB	(A)/(B)高位→A, 余数→B	√	√	×	0	1	4
D4	DA A	对 A 进行十进制调整	√	×	√	√	1	4
逻 辑 运 算 指 令								
58~5F	ANL A, Rn	(A)^(Rn)→A	√	×	×	×	1	1
55	ANL A, direct	(A)^(direct)→A	√	×	×	×	2	1
56, 57	ANL A, @Ri	(A)^((Ri))→A	√	×	×	×	1	1
54	ANL A, #data	(A)^ #data→A	√	×	×	×	2	1
52	ANL direct, A	(direct)^(A)→(direct)	×	×	×	×	2	1
53	ANL direct, #data	(direct)^#data→(direct)	×	×	×	×	3	2
48~4F	ORL A, Rn	(A)∨(Rn)→A	√	×	×	×	1	1
45	ORL A, direct	(A)∨(direct)→A	√	×	×	×	2	1
46, 47	ORL A, @Ri	(A)∨((Ri))→A	√	×	×	×	1	1
44	ORL A, #data	(A)∨ #data→A	√	×	×	×	2	1
42	ORL direct, A	(direct) ∨(A)→(direct)	×	×	×	×	2	1
43	ORL direct, #data	(direct)∨ #data→(direct)	×	×	×	×	3	2
68~6F	XRL A, Rn	(A)⊕(Rn)→A	√	×	×	×	1	1
65	XRL A, direct	(A)⊕(direct)→A	√	×	×	×	2	1
66, 67	XRL A, @Ri	(A)⊕((Ri))→A	√	×	×	×	1	1
64	XRL A, #data	(A)⊕ #data→A	√	×	×	×	2	1
62	XRL direct, A	(direct) ⊕(A)→(direct)	×	×	×	×	2	1
63	XRL direct, #data	(direct)⊕#data→(direct)	×	×	×	×	3	2
E4	CLR A	0→A	√	×	×	×	1	1
F4	CPL A	(\overline{A})→A	×	×	×	×	1	1
23	RL A	(A)循环左移一位	×	×	×	×	1	1
33	RLC A	(A)带进位循环左移一位	√	×	×	√	1	1
03	RR A	(A)循环右移一位	×	×	×	×	1	1
13	RRC A	(A)带进位循环右移一位	√	×	×	√	1	1
C4	SWAP A	(A)半字节交换	×	×	×	×	1	1

续表

十六进制 制代码	助 记 符	功 能	对标志的影响				字节数	周期数
			P	OV	AC	CY		
数 据 传 送 指 令								
E8~EF	MOV A, Rn	(Rn)→A	√	×	×	×	1	1
E5	MOV A, direct	(direct)→A	√	×	×	×	2	1
E6, E7	MOV A, @Ri	((Ri))→A	√	×	×	×	1	1
74	MOV A, #data	#data→A	√	×	×	×	2	1
F8~FF	MOV Rn, A	(A)→Rn	×	×	×	×	1	1
A8~AF	MOV Rn, direct	(direct)→Rn	×	×	×	×	2	2
78~7F	MOV Rn, #data	# data→Rn	×	×	×	×	2	1
F5	MOV direct, A	(A)→(direct)	×	×	×	×	2	1
88~8F	MOV direct, Rn	(Rn)→(direct)	×	×	×	×	2	2
85	MOV direct1, direct2	(direct2)→(direct1)	×	×	×	×	3	2
86, 87	MOV direct, @Ri	((Ri))→(direct)	×	×	×	×	2	2
75	MOV direct, #data	#data→(direct)	×	×	×	×	3	2
F6, F7	MOV @Ri, A	(A)→(Ri)	×	×	×	×	1	1
A6, A7	MOV @Ri, direct	(direct)→(Ri)	×	×	×	×	2	2
76, 77	MOV @Ri, #data	#data→(Ri)	×	×	×	×	2	1
90	MOV DPTR, #data16	#data16→DPTR	×	×	×	×	3	2
93	MOVC A, @A+DPTR	((A)+(DPTR))→A	√	×	×	×	1	2
83	MOVC A, @A+PC	(PC)+1→PC, ((A+PC))→A	√	×	×	×	1	2
E2, E3	MOVX A, @Ri	((Ri))→A	√	×	×	×	1	2
E0	MOVX A, @A+DPTR	((DPTR))→A	√	×	×	×	1	2
F2, F3	MOVX @Ri, A	(A)→(Ri)	×	×	×	×	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	(cy)→cy	×	×	×	√	1	1
B2	CPL bit	(bit)→bit	×	×	×		2	1
82	ANL C, bit	(cy)∧(bit)→cy	×	×	×	√	2	2

续表

十六进制 制代码	助 记 符	功 能	对标志的影响				字节数	周期数
			P	OV	AC	CY		
位 操 作 指 令								
B0	ANL C, /bit	$(cy) \wedge (\overline{(bit)}) \rightarrow cy$	×	×	×	√	2	2
72	ORL C, bit	$(cy) \vee (bit) \rightarrow cy$	×	×	×	√	2	2
A0	ORL C, /bit	$(cy) \vee (\overline{(bit)}) \rightarrow cy$	×	×	×	√	2	2
A2	MOV C, bit	$(bit) \rightarrow cy$	×	×	×	√	2	1
92	MOV bit, C	$(cy) \rightarrow bit$	×	×	×	×	2	2
控 制 转 移 指 令								
*1	ACALL addr11	$(PC)+2 \rightarrow PC, (SP)+1 \rightarrow SP$ $(PCL) \rightarrow (SP), (SP)+1 \rightarrow SP,$ $(PCH) \rightarrow (SP), \text{addr11} \rightarrow PC10 \sim 0$	×	×	×	×	2	2
12	LCALL addr16	$(PC)+3 \rightarrow PC, (SP)+1 \rightarrow SP,$ $(PCL) \rightarrow (SP), SP+1 \rightarrow SP,$ $(PCH) \rightarrow (SP), \text{addr16} \rightarrow PC$	×	×	×	×	3	2
22	RET	$(SP) \rightarrow PCH, SP-1 \rightarrow SP,$ $(SP) \rightarrow PCL, (SP)-1 \rightarrow SP$	×	×	×	×	1	2
32	RETI	$(SP) \rightarrow PCH, (SP)-1 \rightarrow SP,$ $(SP) \rightarrow PCL, (SP)-1 \rightarrow SP,$ 从中断返回	×	×	×	×	1	2
*1	AJMP addr11	$(PC)+2 \rightarrow PC,$ $\text{addr11} \rightarrow PC10 \sim 0$	×	×	×	×	2	2
02	LJMP addr16	$\text{Addr16} \rightarrow PC$	×	×	×	×	2	2
80	SJMP rel	$(PC)+2 \rightarrow PC, (PC)+\text{rel} \rightarrow PC$	×	×	×	×	2	2
73	JMP @A+DPTR	$((A+DPTR)) \rightarrow PC$	×	×	×	×	1	2
60	JZ rel	$(A)=0: PC+2+\text{rel}=PC$ $(A) \neq 0: PC+2=PC$	×	×	×	×	2	2
70	JNZ rel	$(A) \neq 0: (PC)+2+\text{rel}=PC$ $(A)=0: (PC)+2=PC$	×	×	×	×	2	2
40	JC rel	$(C)=1: (PC)+2+\text{rel}=PC$ $(C)=0: (PC)+2=PC$	×	×	×	×	2	2
50	JNC rel	$(C)=0: (PC)+2+\text{rel}=PC$ $(C)=1: (PC)+2=PC$	×	×	×	×	2	2
20	JB bit,rel	$(bit)=1: (PC)+3+\text{rel}=PC$ $(bit)=0: (PC)+3=PC$	×	×	×	×	3	2
30	JNB bit, rel	$(bit)=0: (PC)+3+\text{rel}=PC$ $(bit)=1: (PC)+3=PC$	×	×	×	×	3	2
10	JBC bit, rel	$(bit)=1: (PC)+3+\text{rel}=PC$ $(bit)=0: (PC)+3=PC$ 若(A)= (direct), 则 $(PC)+3 \rightarrow PC$	×	×	×	×	3	2
B5	CJNE A, direct, rel	若(A)≠(direct), 则 $(PC)+\text{rel} \rightarrow PC$ 若(A)<(direct), 则 $1 \rightarrow cy$; 若(A)>(direct), 则 $0 \rightarrow cy$	×	×	×	√	3	2

续表

十六进制 制代码	助 记 符	功 能	对标志的影响				字节数	周期数
			P	OV	AC	CY		
控 制 转 移 指 令								
B4	CJNE A, #data, rel	若(A)=#data, 则 (PC)+3→PC 若(A)≠#data, 则 (PC)+rel→PC, 若(A)>#data, 则 0→cy 若(A)<#data, 则 1→cy 若(Rn)=#data, 则 (PC)+3→PC	×	×	×	√	3	2
B8～BF	CJNE Rn,#data,rel	若(Rn)≠#data, 则 (PC)+rel→PC, 若(Rn)>#data, 则 0→cy 若(Rn)<#data, 则 1→cy 若((Ri))=#data, 则 (PC)+3→PC	×	×	×	√	3	2
B6～B7	CJNE @Ri, #data, rel	若(Ri)≠#data, 则 (PC)+rel→PC, 若((Ri))> #data, 则 0→cy 若((Ri))<#data, 则 1→cy	×	×	×	√	3	2
D8～DF	DJNZ Rn, rel	(Rn)－1→Rn,PC+2→PC, 若(Rn)≠0, 则 PC+rel→PC (PC)+2→PC,	×	×	×	×	2	2
D5	DJNZ direct, rel	(direct)－1→(direct), 若(direct)≠0, 则 (PC)+rel→PC	×	×	×	×	3	2
00	NOP	空操作	×	×	×	×	1	1




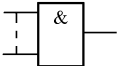



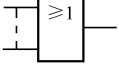

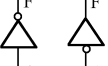

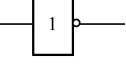



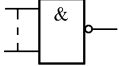



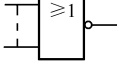


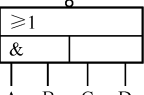
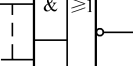



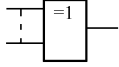
附录 B 各数制对照表

十	十六	二	二-十	十	十六	二	二-十
0	0	0000	0000	8	8	1000	1000
1	1	0001	0001	9	9	1001	1001
2	2	0010	0010	10	A	1010	00010000
3	3	0011	0011	11	B	1011	00010001
4	4	0100	0100	12	C	1100	00010010
5	5	0101	0101	13	D	1101	00010011
6	6	0110	0110	14	E	1110	00010100
7	7	0111	0111	15	F	1111	00010101

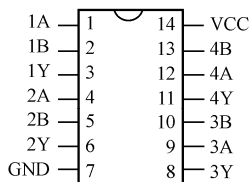
附录 C ASCII（美国标准信息交换码）表

$\begin{matrix} & b_6b_5b_4 \\ b_3b_2b_1b_0 \end{matrix}$	000	001	010	011	100	101	110	111
0 0 0 0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	\$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(8	H	X	h	x
1 0 0 1	HT	EM)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[k	{
1 1 0 0	FF	FS	,	<	L	\	l	
1 1 0 1	CR	GS	-	=	M]	m	}
1 1 1 0	SO	RS	.	>	N	↑	n	~
1 1 1 1	SI	US	/	?	O	←	o	DEL

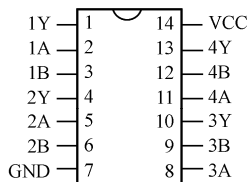
附录 D 二进制逻辑单元图形符号对照表

序号	名称	原电子部标准	原国际通用符号	原国家标准	国家标准	备注
1	与门					$F = A \cdot B$
2	或门					$F = A + B$
3	非门					$F = \overline{A}$
4	与非门					$F = \overline{A \cdot B}$
5	或非门					$F = \overline{A + B}$
6	与或非门					$F = \overline{A \cdot B + C \cdot D}$
7	异或门					$F = A \oplus B$

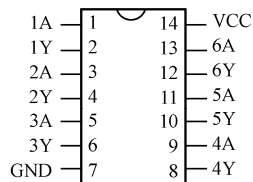
附录 E 常用芯片引脚图



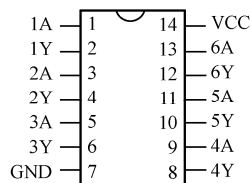
四 2 输入与非门 74HC00



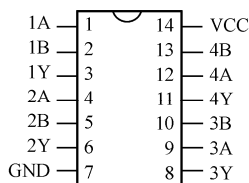
四 2 输入或非门 74HC02



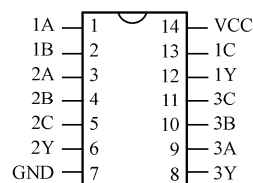
6 反相器 74HC04



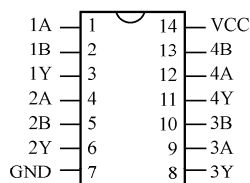
6 同相缓冲器/驱动器
(OC 高压输出) 7407



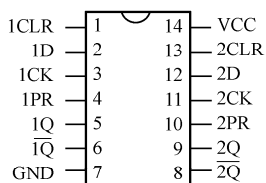
四 2 输入与门 74HC08



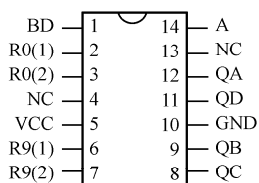
三 3 输入与非门 74HC10



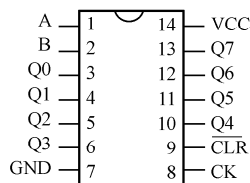
四 2 输入或门 74HC32



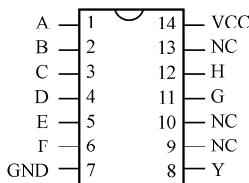
正沿触发双 D 锁存器 74HC74



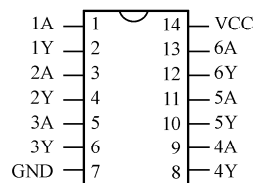
十进制计数器 74HC90



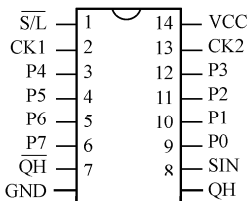
8 位串入/并出移位寄存器
74HC164



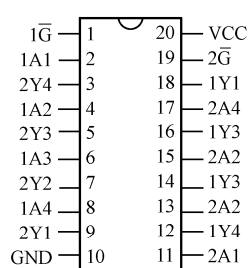
8 输入与非门 74HC30



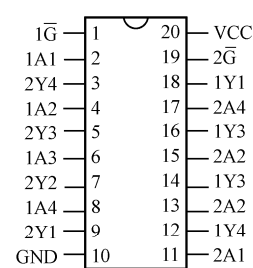
6 反相缓冲器/驱动器
(OC 高压输出) 7406



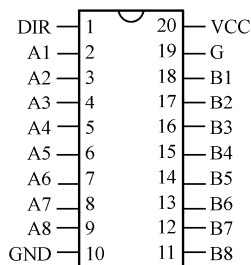
8 位并入/串出移位寄存器
74HC165



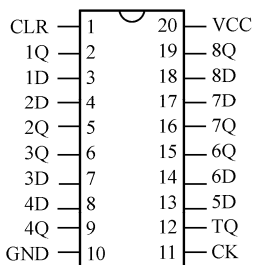
8 缓冲/线驱动/线接收器
(原码三态输出) 74HC244



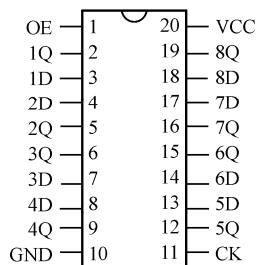
8 缓冲/线驱动/线接收器
(反码三态输出) 74HC240



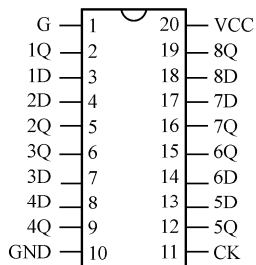
8 总线接收/发送器
74HC245



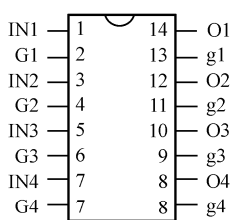
8D 锁存器 74HC273



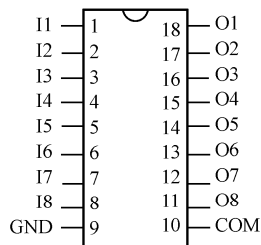
8D 透明锁存器
(三态输出) 74HC373



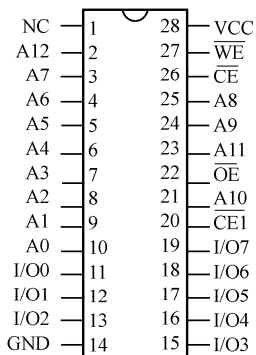
带使能端的 8D 锁存器
74HC377



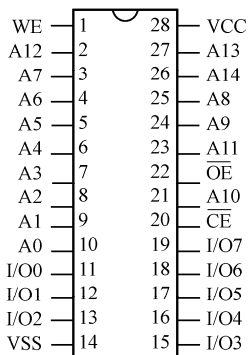
四路光隔电路 TLP521-4



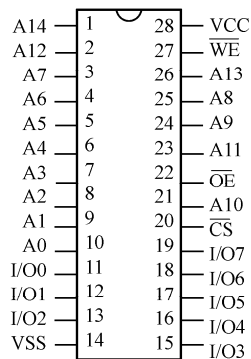
达林顿驱动电路 ULN2803



8K×8 位静态 RAM6264



32K×8 位闪存 AT29C256



32K×8 位静态 RAM62256

参 考 文 献

- [1] 张迎新, 等. 单片机原理及应用[M]. 第2版. 北京: 北京电子工业出版社, 2008.
- [2] 张迎新. 单片微型计算机原理、应用及接口技术[M]. 第1版. 北京: 国防工业出版社, 1993.
- [3] 张迎新, 王盛军, 等. 单片机初级教程[M]. 第3版. 北京: 北京航空航天大学出版社, 2015.
- [4] Atmel Corporation. Microcontroller Data Book. 2004.
- [5] 何立民. MCS-1 系列单片机应用系统设计配置与接口技术[M]. 北京: 北京航空航天大学出版社, 1990.
- [6] 李朝青. 单片机原理及接口技术[M]. 北京: 北京航空航天大学出版社, 2005.
- [7] 张毅刚, 等. 单片机原理及应用. [M]. 北京: 高等教育出版社, 2016
- [8] 万光毅, 严义. 单片机实验与实践教程(一)[M]. 北京: 北京航空航天大学出版社, 2003.
- [9] 何立民. 单片机高级教程[M]. 北京: 北京航空航天大学出版社, 2000.
- [10] 李维促, 郭强. 液晶显示应用技术[M]. 北京: 电子工业出版社, 2000.
- [11] Philips Semiconductors and Electronics North America Corporation. Data Hanndbook 80C51Based 8Bit Microcontrollers. Printed in U S A, 2005.
- [12] 王幸之, 等. AT89 系列单片机原理与接口技术[M]. 北京: 北京航空航天大学出版社, 2004.
- [13] 马忠梅, 等. 单片机的 C 语言应用程序设计(第5版). 北京: 北京航空航天大学出版社, 2013.
- [14] 张俊谟. 单片机中级教程[M]. 北京: 北京航空航天大学出版社, 2000.
- [15] 余永权, 等. ATMEL 系列单片机应用技术[M]. 北京: 北京航空航天大学出版社, 2003.
- [16] 万隆, 巴奉丽. 单片机原理及应用技术(第2版). 北京: 清华大学出版社, 2014.
- [17] 谢维成, 杨加国. 理与应用及 C51 程序设计(第3版). 北京: 清华大学出版社, 2014.
- [18] 李群芳, 等. 单片微型计算机与接口技术(第4版). 北京: 电子工业出版社, 2012.

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任 and 行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396; (010) 88258888

传 真：(010) 88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036

