



普通高等教育电气信息类规划教材



免费电子教案下载

www.cmpedu.com

单片机原理及应用

蔡启仲 主 编

柯宝中 包敬海 陈文辉 副主编



机械工业出版社
CHINA MACHINE PRESS

普通高等教育电气信息类规划教材

单片机原理及应用

蔡启仲 主编

柯宝中 包敬海 陈文辉 副主编



机械工业出版社

本书详细介绍了 MCS-51 单片机的硬件结构、指令系统和外部扩展电路的设计及汇编语言程序设计;对编写的程序作了详细的注释,从第 6 章开始,以中断程序的编写为主,加强了定时中断编程的分析与设计思路的说明、单片机外部扩展接口电路的编址和扩展综合电路的设计介绍;介绍了单片机 C 语言基本知识, C51 编写的程序基本上是与第 6~12 章的例题一一对应;介绍了常用的 I²C 和 SPI 串行总线接口设计、单片机应用系统综合设计举例,并在附录中详细介绍了常用单片机开发环境——uVision2 集成开发环境的使用。本书突出了选取内容的实用性,书中的例子大多来自于工程实践和教学实践,内容丰富。

本书不但适用于不开设《微机原理及接口技术》课程的工科院校本科生单片机课程的教学用书,也适用于将《微机原理及接口技术》作为先修课程的工科院校本科生单片机课程的教学用书,并可以作为从事单片机研发、应用等工作的专业技术人员的参考用书。

本书配套授课电子课件,需要的教师可登录 www.cmpedu.com 免费注册,审核通过后下载,或联系编辑索取(QQ: 308596956, 电话: 010-88379753)。

图书在版编目(CIP)数据

单片机原理及应用/蔡启仲主编. —北京:机械工业出版社, 2016. 8

普通高等教育电气信息类规划教材

ISBN 978-7-111-54515-6

I. ①单… II. ①蔡… III. ①单片微型计算机-高等学校-教材
IV. ①TP368.1

中国版本图书馆 CIP 数据核字(2016)第 186995 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)

策划编辑:时 静 责任编辑:时 静

责任校对:张艳霞 责任印制:

印刷厂印刷

2016 年 8 月第 1 版·第 1 次印刷

184mm×260mm·18.25 印张·443 千字

0001— 册

标准书号: ISBN 978-7-111-54515-6

定价: 元

凡购本书,如有缺页、倒页、脱页,由本社发行部调换

电话服务

网络服务

服务咨询热线:(010)88379833

机工官网:www.cmpbook.com

读者购书热线:(010)88379649

机工官博:weibo.com/cmp1952

教育服务网:www.cmpedu.com

封面无防伪标均为盗版

金书网:www.golden-book.com

前 言

微型计算机自 20 世纪 70 年代诞生以来,得以迅速地发展、普及和应用。随着增强型 8051 单片机和嵌入式微处理器技术的飞速发展,以及单片机 C 语言应用的普及,单片机应用领域越来越广泛。由于 MCS-51 单片机与增强型 8051 单片机和嵌入式微处理器的基本原理和编程方法基本一致,而 MCS-51 单片机的基本原理和应用技术易学易懂,所以学习和掌握 MCS-51 单片机是学习与应用增强型 8051 单片机和嵌入式微处理器技术的基础。

本书在详细阐述单片机原理及结构的基础上,考虑到目前教学改革要求减少课内课时,增加课外学习和实践的时间,以及增强型 8051 单片机和单片机 C51 语言的普遍应用等因素,编写时重点做了如下的思考:

- 1) 对编写的程序作了详细注释,便于自学与理解。
- 2) 针对 C51 语言编程容易解决的算法问题,减少了算术运算等程序的举例。
- 3) 中断技术的应用是微处理器应用系统的基本技能,无论是应用 C51 还是汇编语言编程,对于初学者都是难点,本书加强了中断技术应用举例,尤其是定时中断编程的分析与思路的形成说明,从第 6 章开始,以中断程序的编写为主,提高读者掌握工程实用技术的技能。
- 4) 加强了单片机扩展外部接口电路的编址和扩展综合电路设计的描述,无论是增强型 8051 单片机还是应用 C51 语言编程,它们都是设计微处理器应用系统必须掌握的基本技能之一。
- 5) 第 13 章的例题和应用 C51 编写的程序基本上是与第 6~12 章的例题一一对应,便于读者对照 C51 和汇编程序的异同点,尤其是编程思路的共同点。

本书引入了工程设计中常应用的总线及接口技术(如 I²C、SPI 等)、单片机的 C51 程序设计以及 Keil C51 开发环境等内容,对提高读者应用单片机技术的工程实践综合能力、加强 C 语言的应用能力具有较高的实际应用价值。

全书共 14 章。第 1 章介绍了微型计算机的基本组成与工作原理,以及单片机技术的发展;第 2~7 章详细介绍了 MCS-51 单片机的硬件结构、指令系统和汇编程序设计、片内各部件的功能和编程应用;第 8~11 章介绍了各种类型的外部扩展部件的接口电路及编程设计,如存储器、I/O 接口、键盘及显示、A-D 和 D-A 转换等;第 12 章介绍了 I²C 总线、SPI 总线在内的串行总线接口的时序和接口驱动程序;第 13 章介绍了单片机 C51 语言及其编程方法;第 14 章从实际应用出发详细介绍了单片机应用系统的开发过程、单片机应用系统的可靠性设计及单片机应用系统综合设计举例。最后,在附录中,详细介绍了单片机开发环境——uVision2 集成开发环境的使用及用户程序的调试过程。

本书的参考学时为 40~64 学时。教师可根据实际情况,对书中的内容进行取舍。

本书由广西科技大学蔡启仲担任主编,并完成第 1 章、附录 A 的编写以及全书的统稿

工作；柯宝中、钦州学院包敬海担任副主编，柯宝中完成第 8 章和第 12 章的编写，并参与全书的统稿工作，包敬海完成第 2 章和第 4 章的编写工作。此外，参加本书编写工作的还有陈文辉、刘清正、刘瑞琪、蓝会立和梁喜幸。其中，陈文辉完成第 13 章和附录 B 的编写，并参与全书的统稿工作；刘清正完成第 5 章、第 6 章和第 7 章的编写工作；刘瑞琪完成第 3 章和第 11 章的编写工作；蓝会立完成第 9 章和第 10 章的编写工作；广西科技大学鹿山学院梁喜幸完成第 14 章的编写工作。

书中错误及疏漏之处在所难免，敬请读者批评指正。

编者

目 录

前言

第 1 章 绪论	1
1.1 计算机中常用的编码	1
1.2 微型计算机的分类与组成	3
1.2.1 微型计算机系统的分类	3
1.2.2 微型计算机系统的组成	4
1.3 微型计算机的基本工作原理	5
1.4 单片机的基本概念	9
1.4.1 单片机的特点	9
1.4.2 单片机的发展概况	9
1.4.3 单片机的应用	11
1.4.4 常用的系列单片机	12
习题	15
第 2 章 51 单片机的硬件结构	16
2.1 51 单片机的内部结构与引脚功能	16
2.1.1 51 单片机的基本组成	16
2.1.2 51 单片机的引脚功能	18
2.2 51 单片机的 CPU	19
2.2.1 运算部件	19
2.2.2 控制部件和振荡器	21
2.2.3 CPU 时序	23
2.3 存储器结构	24
2.3.1 程序存储器	25
2.3.2 数据存储器	26
2.3.3 特殊功能寄存器 (SFR)	27
2.3.4 位寻址空间	28
2.4 并行输入/输出端口	30
2.4.1 P0 端口	30
2.4.2 P1 端口	32
2.4.3 P2 端口	32

2.4.4	P3 端口	33
2.4.5	端口电路小结	34
2.5	51 单片机复位操作与复位电路	35
2.5.1	复位操作与复位状态	35
2.5.2	复位电路	36
2.5.3	时钟电路	36
2.5.4	“看门狗”技术	36
2.6	51 单片机的低功耗运行方式	37
2.6.1	方式设定	37
2.6.2	空闲工作方式	38
2.6.3	掉电工作方式	38
	习题	39
第3章	51 单片机的指令系统	40
3.1	51 单片机的汇编指令	40
3.1.1	一个简单的程序示例	40
3.1.2	指令格式	40
3.1.3	指令类型	41
3.1.4	常用符号说明	41
3.2	51 单片机指令的寻址方式	42
3.3	51 单片机的指令集	45
3.3.1	数据传送类指令	45
3.3.2	算术运算类指令	49
3.3.3	逻辑运算及移位类指令	52
3.3.4	控制转移类指令	53
3.3.5	位操作类指令	55
	习题	56
第4章	汇编语言程序设计	57
4.1	单片机汇编语言程序设计概述	57
4.1.1	机器语言、汇编语言和 C 语言	57
4.1.2	汇编语言的特点	59
4.2	伪指令	59
4.3	汇编语言程序的基本结构形式	61
4.3.1	顺序程序	62
4.3.2	分支程序	62
4.3.3	循环程序	62
4.3.4	位操作程序	63
4.4	汇编语言程序设计举例	64
4.4.1	顺序程序设计	64

4.4.2	分支程序设计	65
4.4.3	循环程序设计	67
4.4.4	子程序设计	68
	习题	70
第5章	51 单片机定时器/计数器	72
5.1	51 单片机定时器/计数器的结构	72
5.1.1	定时器/计数器的核心原型	72
5.1.2	定时器/计数器的组成	73
5.1.3	定时器/计数器的工作方式寄存器 TMOD	74
5.1.4	控制寄存器 TCON	75
5.2	定时器/计数器的工作方式	75
5.2.1	方式1	75
5.2.2	方式2	77
5.2.3	方式3	77
5.2.4	方式0	78
5.3	定时器/计数器的应用	79
5.3.1	方式1 的应用举例	79
5.3.2	方式2 的应用举例	82
5.3.3	方式3 的应用举例	83
5.3.4	51 单片机外部输入的计数脉冲信号的要求	85
	习题	85
第6章	中断系统	87
6.1	中断系统概念	87
6.1.1	中断技术	87
6.1.2	中断技术的主要功能特点	88
6.1.3	中断技术的优点	89
6.2	51 单片机的中断系统及其管理	90
6.2.1	中断系统的结构	90
6.2.2	中断源	90
6.2.3	中断请求标志	91
6.2.4	中断控制	93
6.3	51 单片机的中断响应	95
6.3.1	中断响应的条件	95
6.3.2	中断响应的时间	95
6.3.3	中断请求的撤销	96
6.4	中断系统的编程	97
6.4.1	中断服务程序编程的基本步骤	97
6.4.2	外部中断源的编程	98

6.4.3 定时器/计数器中断方法的编程	101
习题	104
第7章 串行通信及应用	106
7.1 数据通信的基本概念	106
7.1.1 根据通信方式分类	106
7.1.2 根据同步方式分类	107
7.1.3 根据数据传输方向分类	108
7.2 51 单片机串行口	109
7.2.1 串行口的结构	109
7.2.2 串行口控制寄存器 SCON	109
7.2.3 电源控制寄存器 PCON	111
7.3 串行口工作方式	111
7.3.1 方式 0	111
7.3.2 方式 1	112
7.3.3 方式 2 和方式 3	113
7.4 单片机串行口波特率	115
7.4.1 波特率与字符传输速率	115
7.4.2 波特率的计算	115
7.5 串行口的应用	117
7.5.1 串行口扩展并行 I/O 口	117
7.5.2 双单片机异步串行通信	117
7.5.3 多机通信	119
7.5.4 RS-232 总线及接口电路	120
习题	122
第8章 51 单片机的系统扩展	123
8.1 三总线与地址编码	123
8.1.1 51 单片机应用系统总线	123
8.1.2 51 单片机系统总线的构成	124
8.1.3 单片机系统总线驱动能力扩展	125
8.1.4 51 单片机系统扩展外部部件的地址空间分配	126
8.2 程序存储器的扩展及应用	129
8.2.1 常用 EPROM 芯片介绍	129
8.2.2 EPROM 芯片的工作方式	130
8.2.3 程序存储器的扩展	131
8.2.4 程序存储器的读控制与时序	133
8.3 数据存储器的扩展及应用	135
8.3.1 常用的静态数据存储器芯片介绍	135
8.3.2 RAM 芯片的工作方式	135

8.3.3 数据存储器的扩展	136
8.3.4 数据存储器的读写控制与时序	137
8.4 程序存储器和数据存储器的综合扩展	138
习题	139
第9章 51 单片机扩展 I/O 接口的设计	141
9.1 并行 I/O 接口电路与编址	141
9.2 可编程 8255 并行 I/O 接口芯片的扩展	142
9.2.1 8255 芯片介绍	142
9.2.2 8255 的工作方式	144
9.2.3 8255 与 51 单片机的接口及应用	148
9.3 可编程 8155 并行 I/O 接口芯片的扩展	150
9.3.1 8155 芯片介绍	150
9.3.2 8155 的 I/O 端口的工作方式	152
9.3.3 8155 与 51 单片机的接口及应用	156
习题	158
第10章 A-D 和 D-A 转换器的接口	160
10.1 D-A 转换器接口	160
10.1.1 D-A 转换器原理	160
10.1.2 51 单片机与 DAC0832 的接口与编程	161
10.1.3 51 单片机与 DAC1208 的接口与编程	166
10.2 A-D 转换器接口	167
10.2.1 A-D 转换器原理	168
10.2.2 51 单片机与 ADC0809 的接口	169
10.2.3 51 单片机与 AD574 的接口	173
习题	176
第11章 51 单片机与键盘、显示器接口设计	177
11.1 键盘接口电路	177
11.1.1 独立式键盘	177
11.1.2 矩阵式键盘	180
11.2 LED 显示接口电路	183
11.2.1 LED 显示器	183
11.2.2 LED 并行显示技术	185
11.3 键盘、LED 显示器、A-D 和 D-A 的综合应用	188
11.4 LCD 显示接口电路	189
11.4.1 LCD 显示器件	189
11.4.2 单片机与液晶显示模块接口电路与驱动	192
习题	192
第12章 单片机的串行总线接口技术	194

12.1	I ² C 总线接口	194
12.1.1	I ² C 总线概述	194
12.1.2	串行 EEPROM 概述	195
12.1.3	24C02 引脚功能	195
12.1.4	工作方式	196
12.1.5	器件寻址、写操作和读操作	197
12.1.6	24C02 芯片与 51 单片机的接口电路及编程	199
12.1.7	ZLG7290 接口电路及编程	202
12.2	SPI 总线接口	205
12.2.1	SPI 总线概述及主要特点	205
12.2.2	12 位串行 A-D TLC2543 介绍	206
12.2.3	TLC2543 与 51 单片机的接口设计	210
12.3	单总线串行扩展	211
	习题	212
第 13 章	51 单片机的 C51 程序设计	214
13.1	C51 编程概述	214
13.1.1	Keil C51 开发工具	215
13.1.2	Keil C51 程序开发过程	215
13.1.3	C51 程序结构	215
13.2	C51 的数据类型及存储类型	216
13.2.1	C51 的基本数据类型	216
13.2.2	C51 的数据存储类型和存储模式	217
13.2.3	单片机特殊功能寄存器 (SFR) 及其 C51 定义	218
13.3	C51 函数	219
13.3.1	函数的分类与定义	219
13.3.2	函数的调用	220
13.3.3	中断服务函数和寄存器组定义	221
13.4	C51 库函数	222
13.5	C51 程序设计举例	225
13.5.1	51 单片机内部资源的 C51 编程	225
13.5.2	51 单片机扩展资源的 C51 编程	233
	习题	240
第 14 章	单片机应用系统的开发与举例	241
14.1	单片机应用系统的开发过程	241
14.1.1	需求分析与可行性论证	241
14.1.2	硬件电路设计原则	242
14.1.3	软件设计原则	243
14.1.4	软硬件测试与程序固化	243

14.2	单片机应用系统的开发工具	245
14.3	应用系统的可靠性设计	245
14.3.1	主要干扰源	245
14.3.2	硬件可靠性设计	245
14.3.3	软件可靠性设计	246
14.4	单片机应用系统设计举例	246
14.4.1	单片机应用系统并行扩展芯片的基本接口电路	246
14.4.2	直流电动机转速控制系统的设计	248
14.4.3	单片机应用系统串行总线接口芯片的扩展应用系统设计	258
附录		262
附录 A	51 单片机指令集	262
附录 B	μ Vision2 集成开发环境使用	266
参考文献		279

第1章 绪 论

本章介绍微型计算机的基本组成与基本工作原理，以及单片机技术的发展，这是学习本书后续章节内容的必要准备。

1.1 计算机中常用的编码

一般来讲，用数字或某种文字、符号或数码串来表示某一对应数字、信号和状态的过程，称为编码（例如，给孩子取名字、给学生编学号、电话号码、各地的邮政编码等，都是编码）。

计算机中采用的是二进制数，因此，在计算机中表示的数、字母和符号等都是以其特定的二进制数来表示，这就是二进制编码，编码的结果称为代码，或二进制码、十六进制码。例如将字符 A 编码为 ASCII 码的 100 0001B，即 41H，则称 A 的代码为 41H。

在计算机中，由于机器只能识别二进制数，因此键盘上所有的数字、字母和符号都事先按一定的规律为它们进行了二进制编码，以便机器加以识别、存储、处理和传送。和日常生活中的编码问题一样，所需编码的数字、字母和符号越多，二进制数的位数也越长。

二进制编码的实质是将二进制数按一定的规律和数字、对象、信号和状态一一对应起来，用不同的二进制数代码串对应数字、对象、信号和状态。

下面就计算机中常用的 BCD（Binary Coded Decimal）码和 ASCII（American Standard Code for Information Interchange）码进行介绍。

1. BCD 码（十进制数的二进制编码）

（1）BCD 码

BCD 码是一种具有十进制权位的二进制编码，最常用的 BCD 码是标准 BCD 码或称为 8421 码，应用十分广泛。BCD 码是将 4 位二进制数组成 1 位十进制数，4 位二进制数有 16 种组合 0000B ~ 1111B，取其 0000B ~ 1001B 按照二进制数的规律排列成十个数字，因为组成它的 4 位二进制数的权位分别为 8、4、2、1 而得名 8421 码，用 0000B ~ 1001B 分别表示十进制的 0 ~ 9，而 BCD 码不使用 1010B ~ 1111B（0AH ~ 0FH）六种状态。BCD 码有压缩 BCD 码和非压缩 BCD 码之分。压缩 BCD 码是 4 位二进制数表示 1 位十进制数，而非压缩 BCD 码是 8 位二进制数表示 1 位十进制数。1 位十进制数的 BCD 码见表 1-1。

表 1-1 BCD 码表

十进制数	压缩的 BCD 码（BCD 码）	非压缩的 BCD 码	十进制数	压缩的 BCD 码（BCD 码）	非压缩的 BCD 码
0	0000	00000000	5	0101	00000101
1	0001	00000001	6	0110	00000110
2	0010	00000010	7	0111	00000111
3	0011	00000011	8	1000	00001000
4	0100	00000100	9	1001	00001001

通常压缩 BCD 码简称为 BCD 码。

例 1-1 写出两位十进制数 97、42 和三位十进制数 125 的二进制数、压缩 BCD 码和非压缩 BCD 码。

十进制数	二进制数	压缩 BCD 码	非压缩 BCD 码
97	0110 0001	1001 0111	00001001 00000111
42	0010 1010	0100 0010	00000100 00000010
125	0111 1101	0001 0010 0101	00000001 00000010 00000101

(2) BCD 码的加法运算

应用 BCD 码做加法时，其结果仍然是一个二进制数。4 位二进制数的加数和被加数之间只能逢 16 进位，不能逢 10 进位，因此，计算机在进行 BCD 码加法时，必须对二进制加法的结果进行调整。十进制调整是利用调整指令予以实现。只有调整后的 BCD 码才能够做到逢 10 进位。51 单片机专门用于十进制调整的指令将在 3.3.2 节中介绍。

例 1-2 已知 $X = 37$ ， $Y = 15$ ，试计算 $(X + Y)_{\text{BCD 码}} = ?$

分析： $(37)_{\text{BCD 码}} = 0011\ 0111\text{B}$ ， $(15)_{\text{BCD 码}} = 0001\ 0101\text{B}$

$$\begin{array}{r}
 0011\ 0111 \\
 +) \ 0001\ 0101 \\
 \hline
 0100\ 1100 \\
 +) \ 0110 \\
 \hline
 0101\ 0010 \rightarrow \text{十进制加6调整}
 \end{array}
 \qquad
 \begin{array}{r}
 37 \\
 + \ 15 \\
 \hline
 52
 \end{array}$$

则有：

$$(X + Y)_{\text{BCD 码}} = (37 + 15)_{\text{BCD 码}} = 0101\ 0010\text{B} = 52$$

BCD 码的优点：

- 1) BCD 码与自然二进制数有很好的对应关系，很容易实现彼此之间的转换。
- 2) BCD 码具有奇偶特性，凡是奇数十进制数的 BCD 码的最低位皆为 1，偶数的 BCD 码的最低位皆为 0，所以采用 BCD 码的十进制数容易判别奇偶性。

2. ASCII 码

ASCII 码是美国国家标准信息交换码，目前它是许多国家通用的一种国际标准码。ASCII 码用一个 8 位二进制数来表示一个字符的代码，其中最高位常用作奇偶校验位，用来检验 ASCII 码在存储和传送过程中是否发生错误，低 7 位是字符的 ASCII 码的代码值。

ASCII 码用 7 位二进制数组成字符的代码进行编码时，可以得到 128 个不同的代码，其中 26 个大写英文字母 A ~ Z 的代码为 41H ~ 5AH，26 个小写英文字母 a ~ z 的代码 61H ~ 7AH，小写字母的代码与大写字母代码相差 20H，十进制数 0 ~ 9 的代码为 30H ~ 39H，文字符 34 个代码，其余代码则分配给各种标点符号和运算符等。

例如：英文字母 A 的 ASCII 码是 41H ($D_6D_5D_4D_3D_2D_1D_0 = 1000001\text{B}$)；

数字 7 的 ASCII 码是 37H ($D_6D_5D_4D_3D_2D_1D_0 = 0110111\text{B}$)；

回车“CR”的 ASCII 码是 0DH ($D_6D_5D_4D_3D_2D_1D_0 = 0001101\text{B}$)；

逗号“,”的 ASCII 码是 2CH ($D_6D_5D_4D_3D_2D_1D_0 = 0101100\text{B}$)。

ASCII 码表见表 1-2。

表 1-2 ASCII 码

编码	字符	编码	字符	编码	字符	编码	字符	编码	字符	编码	字符	编码	字符
00	NUL	12	DC2	24	\$	36	6	48	H	5A	Z	6D	m
01	SOH	13	DC3	25	%	37	7	49	I	5B	[	6E	n
02	STX	14	DC4	26	&	38	8	4A	J	5C	\	6F	o
03	ETX	15	NAK	27	'	39	9	4B	K	5D	]	70	p
04	EOT	16	SYN	28	(	3A	:	4C	L	5E	^	71	q
05	END	17	ETB	29	)	3B	;	4D	M	5F	_	72	r
06	ACK	18	CAN	2A	*	3C	<	4E	N	60	`	73	s
07	BEL	19	EM	2B	+	3D	=	4F	O	61	a	74	t
08	BS	1A	SUB	2C	,	3E	>	50	P	62	b	75	u
09	HT	1B	ESC	2D	-	3F	?	51	Q	63	c	76	v
0A	LF	1C	FS	2E	.	40	@	52	R	64	d	77	w
0B	VT	1D	GS	2F	/	41	A	53	S	65	e	78	x
0C	FF	1E	RS	30	0	42	B	54	T	66	f	79	y
0D	CR	1F	US	31	1	43	C	55	U	67	g	7A	z
0E	SO	20	space	32	2	44	D	56	V	68	h	7B	{
0F	SI	21	!	33	3	45	E	57	W	69	i	7C	
10	DLE	22	"	34	4	46	F	58	X	6A	j	7D	}
11	DC1	23	#	35	5	47	G	59	Y	6B	k	7E	~
										6C	l	7F	DEL

1.2 微型计算机的分类与组成

计算机可以分为巨型、大型、小型和微型计算机，微型计算机应用得极为广泛，已成为人们的日常生活和工作中必备的工具。

1.2.1 微型计算机系统的分类

微型计算机常简称为微型机或微机。微型计算机的分类方式很多。按微处理器的位数，可分为 1 位、4 位、8 位、32 位和 64 位机等。按组装方式可分为单片机、单板机、通用微型计算机和嵌入式微型机。

1) 单片机。利用大规模集成电路工艺将微型机的三大组成部分：CPU（Central Processing Unit，中央处理器）、内存储器和 I/O 接口电路集成在一片硅片上，这就是单片机。目前有 8 位机、16 位机和 32 位机之分。使用简单的开发装置可以对它进行在线开发。单片机在工业过程控制、智能化仪器仪表和家用电器中得到广泛的应用。

2) 单板机。若将微型机的 CPU、内存和 I/O 接口电路的多个芯片安装在一块印制电路板上就组成了单板机。单板机结构简单，价格低廉，性能较好，经过开发后，可用于过程控制、各种仪器仪表、机器的单机控制和数据处理等，且普遍用作学习“微型机原理”的实

验机型。

3) 通用微型计算机。根据需把微型机的 CPU、内存、I/O 接口电路和电源等组装在不同的印制电路板上, 然后装进同一个机箱, 就构成了一个通用微型计算机。目前, 由多个微处理器组成多核微型计算机已得到普及, 与单核计算机比较, 多核微型计算机的运行速度得到了极大地提高。这些通用微型计算机可配置键盘、监视器 (CRT、液晶显示器)、打印机和硬磁盘驱动器等多种外围设备和足够的软件, 成为一个完整的通用微型计算机系统。国内用得非常多的个人微型计算机和笔记本式微型计算机等都是单核和多核微处理器的微型计算机系统, 笔记本式微型计算机还衍生出膝上型、掌上型和平板计算机。

4) 嵌入式微型机。微型机作为各种应用装置的控制核心, 嵌入到装置中组成嵌入式系统, 例如 ARM 系列微处理器、DSP 信息处理芯片等与各种外设组成嵌入式微型机, 使得这些装置成为智能控制系统, 已在通信装置、家电、仪器仪表、信息处理、测控和自动化装置等领域得到广泛的应用。

1.2.2 微型计算机系统的组成

1. 硬件系统

电子数字计算机起初是作为一个计算工具出现的, 那么计算机是如何脱离人的干预, 自动完成计算功能的呢?

作为一个计算工具, 首先要有能进行运算的部件, 这称为运算器; 其次要有能起到纸和笔作用的部件, 它能记忆原始题目、原始数据和中间结果, 以及为了使机器能自动进行运算而编制的各种命令, 这种部件称为存储器; 再次, 要有能代替人的控制作用的部件, 它能根据事先给定的命令发出各种控制信息, 使整个计算过程能一步一步进行, 这种部件称为控制器; 对于原始的数据和命令的输入、数据计算和处理结果的输出, 需要有输入设备和输出设备。这样就构成了一个基本的微型计算机系统。冯·诺依曼结构的微型计算机系统基本组成如图 1-1 所示。

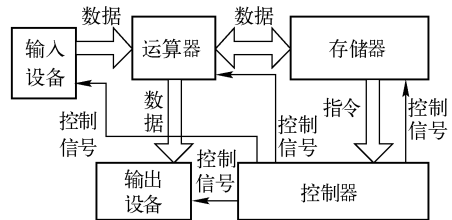


图 1-1 微型计算机系统的基本组成

存储器通常分为内存储器 (简称为内存) 和外存储器 (简称为外存) 两部分。内存也称为主存, 用于存放正在运行的程序或正在处理的数据。换句话说, 也就是所有程序只有调入内存才能执行。内存容量一般为 8 MB、16 MB、32 MB、64 MB、128 MB、256 MB、512 MB、1 GB 和 2 GB 等, 并多采用半导体存储器构成。其特点是: 存储容量小, 存取速度较快。

外存也称为辅助存储器, 其存取速度较慢, 但容量大, 现在一般可达数百、数千 GB。它常被用于存放暂不执行的程序和数据, 在关机或停电时信息不丢失, 能够长期保存程序和数据, 常用的有硬盘、光盘和 U 盘等。

输入设备常用的有键盘、触摸屏和鼠标等; 输出设备常用的有 CRT 和液晶显示器、打印机、扫描仪等。

在图 1-1 中, 人们常把运算器、存储器和控制器合在一起称为微型计算机的主机, 而把各种输入输出设备统称为微型计算机的外围设备。在主机部分, 往往将运算器和控制器合在一起称为中央处理单元 (Central Processing Unit, CPU)。

2. 软件系统

软件泛指计算机中的程序和文件等，是微型计算机必不可少的重要组成部分。从使用的角度看，软件系统可分为两大类：一类是用于管理微型计算机资源，协调其内部工作的程序，称为系统软件；另一类是为解决某些实际问题或用户根据自己的需要而设计的程序，称为应用软件。

(1) 系统软件

所谓系统软件是指用来实现对微型计算机资源进行管理，便于人们使用微型计算机而配置的软件，它一般是由厂家随微型计算机一起配置。系统软件通常包括操作系统（或监控管理程序）；各种语言的处理程序（如汇编、解释或编译程序）；数据库管理程序；网络管理软件；编辑、调试、装配、故障检查和诊断等工具软件。

在系统软件中称为操作系统的软件具有特殊的地位。只要计算机处在工作状态，就有操作系统的有关部分在内存储器中，负责接收、分析并调度执行用户的程序和各种命令。DOS、Windows、Unix、Linux 及 Mac OS 等是目前最流行的微型计算机操作系统。

(2) 应用软件

应用软件是用户根据自己的需要，针对某一实际问题而设计的程序。比如目前流行的各种软件包：文字处理系统、MATLAB 软件平台、计算机辅助设计（CAD）软件、计算机辅助教学（CAI）软件、图形动画软件、病毒防护软件和各种网络应用软件等；在工业自动控制方面有许多数据采集与控制程序；在信息管理方面有许多管理程序，如人事管理、财务管理和图书资料管理等。这些程序都是针对某一具体问题设计的，统称应用软件。

(3) 软、硬件的关系

硬件系统是构成微型计算机系统各功能部件的集合，而软件系统是微型计算机系统的各种程序的集合。软件与硬件的关系可以用图 1-2 描述。

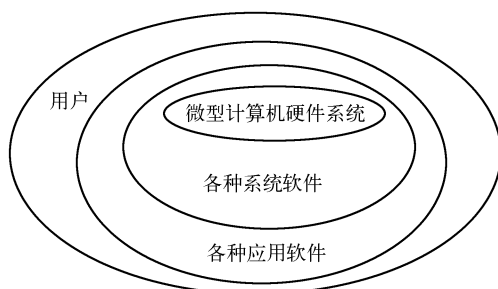


图 1-2 软、硬件的关系

通常情况下，用户使用程序设计语言或其他编程工具编写的应用程序或者直接使用软件商提供的应用软件，在系统软件的干预下，由微型计算机硬件系统执行。所以，硬件系统是人们操作微型计算机的物理基础，而软件系统则是人们与微型计算机系统进行信息交换、通信对话、按人们的思维对微型计算机系统进行控制和管理的工具。

1.3 微型计算机的基本工作原理

计算机能够自动地进行计算，是由于人们把实现计算的一步步操作命令的形式（即

一条条指令) 预先送入到存储器中, 在执行时, 机器把这些命令从存储器中按照预先设定的时序一条条地取出来, 加以翻译和执行。

例如计算 $A + B = ?$, 假定 A、B 的值已存储在存储器中, 进行这个简单运算, 通常需要做以下几步操作:

- 第一步: 从存储器中取数 A;
- 第二步: 从存储器中取数 B 并与 A 相加;
- 第三步: 把 $A + B$ 的结果存储到存储器中;
- 第四步: 停机。

所有这些取数、相加和存数等都是一种基本操作, 把计算机执行的各种操作命令的形式写下来, 这就是指令, 计算机所能执行的各种指令的集合, 就是计算机的指令系统。计算机的指令系统是在设计 CPU 时由设计人员规定的。在使用计算机时, 必须把要解决的问题编成一条条指令, 这些指令的有序集合就称为程序。

指令通常分成操作码和操作数两大部分。操作码表示计算机执行什么操作; 操作数表示参加操作的数本身或操作数存储在存储器的地址。

计算机指令系统中的所有指令都是以二进制编码的形式来表示, 称为机器指令。为了便于记忆和理解, 人们用助记符 (便于记忆的符号) 代替操作码, 用符号表示操作数, 这样形成的指令称为汇编语言指令。例如, MOV A, 09H, 其中 MOV 是操作码助记符, A、09H 是两个操作数。用汇编语言指令编写的程序称为汇编语言源程序, 源程序必须翻译成机器能识别的二进制数编码, 计算机才能执行。

微型计算机的工作过程就是执行程序的过程。程序由指令序列组成, 由用户预先输入到存储器中。所以, 执行程序的过程就是执行指令序列的过程, 也就是周而复始地取指令、执行指令的过程。

图 1-3 是微型计算机的结构图, 它由微处理器 (CPU)、存储器和接口电路组成, 通过三条总线 (BUS) ——地址总线 (AB)、数据总线 (DB) 和控制总线 (CB) 来连接。为简化问题, 认为要执行的程序以及数据, 已存储在存储器中。

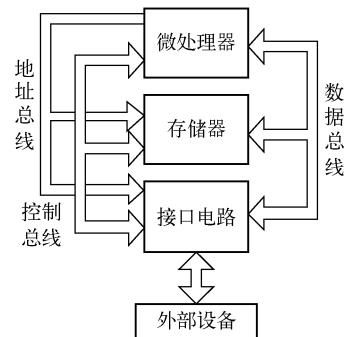


图 1-3 微型计算机结构图

1. CPU 的结构

CPU 的基本结构如图 1-4 中点画线以上部分所示, 图 1-4 中各单元定义如下:

- | | |
|-----------------------|---------------|
| ALU — 算术逻辑单元; | F — 标志寄存器 |
| A — 累加器; | B — 寄存器 |
| AR — 地址寄存器; | DR — 数据寄存器 |
| IR — 指令寄存器; | ID — 指令译码器 |
| PC — 程序计数器 (指令指针寄存器); | PLA — 可编程逻辑阵列 |
| AB — 地址总线; | DB — 数据总线 |

要执行的指令地址由程序计数器 PC 提供, 地址寄存器 AR 将要寻址的存储单元地址通过地址总线 AB 送至存储器, 从该地址存储单元中取出指令后, 由数据寄存器 DR 送至指令译码器 ID 译码, 再通过控制电路发出执行该条指令所需要的各种控制信息。

2. 存储器

存储器的结构如图 1-4 中点画线以下部分所示。

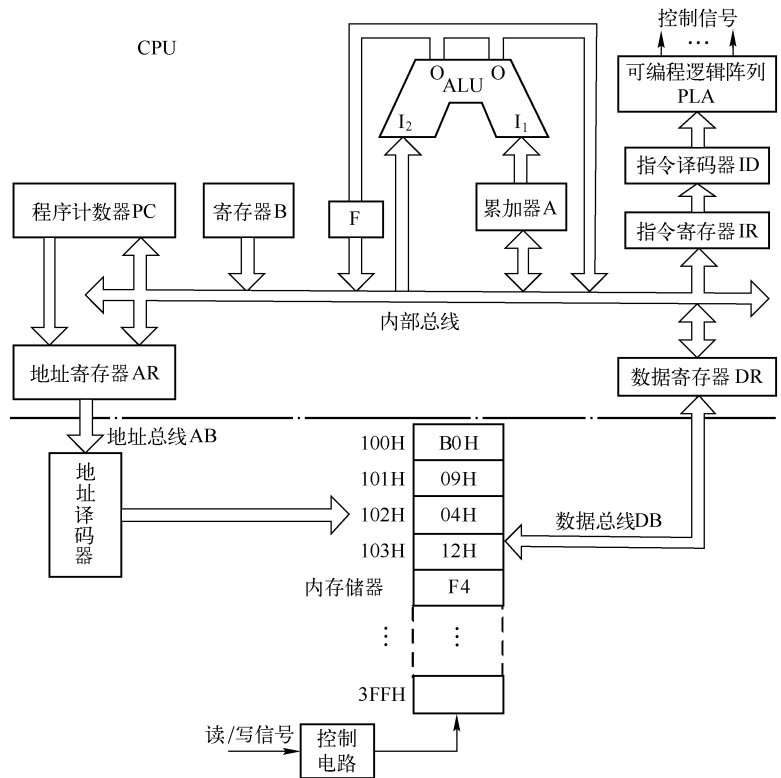


图 1-4 一个模型机的 CPU 结构

存储器以二进制代码的形式存储数据和程序，具有非破坏性读出的特点，即从存储器读出信息后，存储器原来所保存的信息依然不变。

内存储器由许多存储单元（简称单元）组成，每个存储单元可以是 1、4、8、16、32 位等二进制数，其中 1 位为位单元，4 位为半个字节，8 位为一个字节，16 位为一个字，32 位为双字。内存储器中的每个存储单元都对应一个编号，称为该存储单元地址，简称单元地址。内存容量是指地址单元数乘以每个单元的二进制位数，也可用上述乘积即总位数来表示。微处理器就是按照存储单元的地址来访问内存。每次向内存中写入数据或从内存中读出数据时，都首先由微处理器给出地址，然后再发出读或写控制信号，从而把数据写入指定地址的单元或从指定地址的单元读出数据。

3. 简单程序举例

有三条用汇编语言指令表示的程序为：

汇编语言源程序 翻译成的机器代码

- (1) MOV A, 09H B009H
- (2) ADD A, 12H 0412H
- (3) HLT F4H

第一条指令 MOV A, 09H 是把立即数 09H 送入累加器 A。

第二条指令 ADD A, 12H 是把 A 中的内容和立即数 12H 相加, 结果 ($09H + 12H = 1BH$) 存入 A 中。

第三条指令 HLT 为暂停指令。这段程序的功能就是把 09H 和 12H 相加后存放在 A 中。

4. 程序执行的过程

程序执行前, 必须先将程序的机器代码装入内存储器, 同时将第一条指令的起始地址 (如图 1-4 的 100H 存储单元) 赋予程序计数器 PC。程序启动后, 计算机根据 PC 的值从第一条指令开始执行。

(1) 取第一条指令的过程

- 1) PC 的值 100H 送入地址寄存器 AR。
- 2) 程序计数器 PC 的程序指针自动增 1, 指向下一条指令的单元地址, 即 101H。
- 3) 地址寄存器 AR 将地址码 100H 通过地址总线 AB 送到地址译码器, 经译码后选中 100H 单元。
- 4) CPU 发出读控制信号。
- 5) 所选中的 100H 单元内容 B0H 通过数据总线 DB 送到数据寄存器 DR。
- 6) 因是取指令操作阶段, 取出的是指令操作码 B0H, 故 DR 把它送到指令寄存器 IR。
- 7) 指令寄存器 IR 中的操作码 B0H 经指令译码器 ID 译码后, 通过可编程逻辑阵列 PLA 发出执行该指令的有关控制命令。

然后就转入了执行第一条指令阶段。操作码 B0H 经译码后知道这是一条把立即数送累加器 A 的指令, 立即数在指令的第二字节中, 则执行第一条指令就是从内存中取出指令第二字节 (即 09H) 送入 A。

(2) 执行第一条指令的过程

- 1) 将 PC 的内容 101H 送入地址寄存器 AR。
- 2) PC 的内容自动加 1 (变为 102H), 指向下一条指令的单元地址。
- 3) 地址寄存器 AR 通过地址总线 AB 把地址码 101H 送到地址译码器, 经地址译码后选中相应的 101H 存储单元。
- 4) CPU 发出读控制信号。
- 5) 所选中的 101H 单元内容 09H 经数据总线 DB 送到数据寄存器 DR。
- 6) 因已知读出的是操作数, 且指令要求把它送累加器 A, 所以数据寄存器 DR 中的数据 09H 通过内部总线送到 A, 则第一条指令的执行过程全部完成。

接着, 进入第二条指令的取操作码阶段和第二条指令的执行阶段。

和第一条指令类似, 由 PC 的值取得操作码 04H, 经指令译码器 ID 译码后知道这是一条加法指令, 其中第一个操作数已在 A 中, 另一个操作数 (12H) 在指令的第二个字节中。此时, 首先从内存中取出 12H 送 ALU 的 I_2 端, 然后将 A 中的内容与 12H 相加, 结果再送回 A。

至此, 该程序段的主要工作已经完成, 第三条指令只是告诉 CPU 进入“暂停”。这是通过第三条指令 HLT 的机器码 F4H 的取指、译码后实现的。

分析以上模型机执行程序的工作过程之后, 就可以将 CPU 的操作归纳如下:

- 1) 把指令地址放在地址总线上。
- 2) 通过数据总线从存储器中取出指令, 并对指令译码。

- 3) 根据指令中的地址码取出操作数据。
- 4) 执行指令码所规定的操作。
- 5) 在操作过程中提供表示状态的标志信号、控制信号和定时信号。

1.4 单片机的基本概念

微型计算机自 20 世纪 70 年代诞生以来,得以迅速发展、普及和应用,对科学研究和生产生活起到了巨大的推动作用。特别是随着微处理器的出现,单片微型计算机(简称单片机)技术已成为计算机技术中的一个独特的分支,由于具有体积小、价格低、功能强的特点,其应用领域越来越广泛,在工业控制、智能仪器仪表研发等领域中发挥着越来越重要的作用。

1.4.1 单片机的特点

一块单片机芯片具有微型计算机的基本结构,其基本结构如图 1-1 所示,一块单片机芯片还集成了微型计算机具有的串行口、定时器/计数器、中断系统和系统时钟等电路,以及单片机扩展外部电路的数据总线、地址总线和控制总线的系统三总线。

其主要特点如下:

1. 体积小,重量轻,功耗低,性价比好

单片机的高性能、低价格是其显著特点,为提高速度和效率,有的单片机已采用流水线和 DSP (Digital Signal Processing, 数字信号处理) 技术,使单片机性能明显优于同类微处理器;增加了 I²C (Inter - Integrated Circuit) 串行总线、SPI (Serial Peripheral Interface) 串行接口等,进一步优化了系统结构,缩小了单片机体积。而低功耗、低电压的特点又使得它便于生产便携式产品,许多单片机已经可以工作于 2.2 V 的电压,有的甚至能在 1.2 V 或 0.9 V 电压下工作,功耗降为 μA 级,一粒纽扣电池就可以保障长期工作。

2. 可靠性高

单片机本身是根据工业控制环境的要求设计,把各功能部件集成在一块芯片上,内部采用总线结构,减少了单片机内部之间的连线,其信号通道受外界影响小,大大提高了单片机的可靠性与抗干扰能力。另外,由于其体积小,对于强磁场环境易于采取屏蔽措施。单片机分为军用级、工业级及民用级三个等级系列,其中军用级、工业级具有较强的适应恶劣环境工作的能力。

3. 控制功能强

为满足控制的要求,单片机的指令系统中均有极丰富的转移指令、I/O 口的逻辑操作以及位处理功能等控制功能命令,其位逻辑控制功能及运行速度均高于同一档次的微型计算机。

4. 易扩展

单片机的系统配置较典型、规范,与很多外围芯片可以直接连接,容易进行相应的扩展构成各种不同规模的应用系统。

1.4.2 单片机的发展概况

1. 4 位单片机阶段

自 1971 年美国 Intel 公司首先研制出 4 位单片机 4004 以来,各个计算机生产公司竞相

推出各自的 4 位单片机。4 位单片机推出之初，主要应用于家用电器、计算器以及电子玩具等初级电子产品。这一阶段属于单片机的萌芽阶段。

2. 中、低档 8 位机阶段

1976 年 9 月，美国 Intel 公司推出了 MCS - 48 系列 8 位单片机，其特点是采用了专门的结构设计，片内集成了 8 位 CPU、8 位并行 I/O 口、8 位定时器/计数器以及 RAM、ROM 等，可满足一般工业控制的需求，不足之处是没有串行口，中断处理比较简单；此后，单片机发展进入了一个新的阶段，8 位单片机纷纷应运而生。例如，Motorola 公司的 6801 系列，Zilog 公司的 Z8 系列，Rockwell 公司的 6501、6502 等；此外，日本的 NEC 公司、日立公司等也推出了具有特色的初级 8 位单片机产品。

3. 高档 8 位机阶段

Intel 公司在 MCS - 48 系列单片机的基础上于 1980 年推出了 MCS - 51 系列单片机。其技术特点是完善了外部总线，确立了单片机的控制功能。它带有串行接口和多个 16 位定时器/计数器，具有 2 级中断功能，片内的 RAM、ROM 容量增大。虽然它仍然是 8 位单片机，但其功能有很大的增强，属于高档 8 位单片机。

在高档 8 位机的基础上，单片机的功能进一步得到提高，近年来相继推出了超 8 位单片机。如 Intel 公司 8X252、UPI - 45283C152，Motorola 公司的 MC68HC，Zilog 公司的 Super8 等，它们一方面扩大了片内存储器容量，更重要的是还增加了通信、DMA 传输以及高速 I/O 等功能，由于这类单片机性价比较高，因此得到广泛应用。

4. 16 位机阶段

1983 年以后，16 位单片机也逐渐问世。代表产品有 Intel 公司 1983 年推出的 MCS - 96 系列单片机、1987 年推出的 80C96，美国国家半导体公司推出的 HPC16040，NEC 公司推出的 783XX 系列，Siemens 公司的 80C167，Hitachi 公司的 H8 和 Motorola 公司的 M68HC16 等。

此外，还推出了 32 位单片机，除了具有更高的集成度外，其主频更高，从而使单片机数据处理速度得以大幅度提高，性能更加优越，进一步拓展了单片机的应用范围。

5. 单片机全面发展阶段

这一阶段单片机的显著技术特点是全面发展单片机的各种功能，提高性能，应用流水线技术提高单片机执行程序的速度。单片机的首创公司 Intel 将其 MCS - 51 系列单片机中的 8051 单片机的 CPU 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 Atmel、Philips、Motorola、Analog Devices、DALLAS、Siemens、OKI、Dallas、NEC 和 SST 华邦等公司，这些厂家生产以 8051 为内核推出增强型 8051 系列单片机及其派生型单片机，这些兼容产品与 8051 的系统结构相同，8051 单片机事实上已经成为单片机结构标准。这些公司以 8051 单片机的 CPU 为内核推出的增强型 8051 系列单片机产品中，在保持与 8051 单片机兼容的基础上增强了 8051 的许多特性，改善其结构，将外围电路内装化，在芯片内部嵌入集成了测控系统常用的模数、数模转换器，监视定时器 WDT，脉宽调制器，液晶显示控制器，CAN 总线控制器，I²C、SPI、MICROWIRE 等串行总线及其接口，以及较大容量的数据存储单元，实时处理能力更强，大大降低了单片机应用系统的电路规模。

随着单片机在各个领域全面深入的发展和应用，高速、大寻址范围、强运算能力、低电压及低功耗，外围电路内装化、增强 I/O 功能，高度集成化低成本的 8 位、16 位乃至 32 位的通用及专用等多种系列单片机并存，成为当前阶段单片机发展的显著特征。

另一方面，由于 Flash 存储器（Flash Memory，简称闪存）的出现及使用，能够应用在线编程技术 ISP（IN System Programmable）在 PC 上编好程序，并通过所建立的 SPI 或其他串行接口直接传输并且烧录到单片机的闪存上，单片机芯片可以直接焊接到电路板上，免去了调试时由于频繁地插入取出芯片对芯片和电路板带来的不便，大大改进了开发环境。

1.4.3 单片机的应用

单片机的应用范围十分广泛，其中主要的应用领域有：

(1) 家用电器

这是单片机最早应用的领域之一，目前国内外各种家用电器已普遍采用单片机代替传统的控制电路，如微波炉、电视机、电冰箱、空调、洗衣机、录像机、音响设备乃至许多高级电子玩具都配上了单片机，从而提高智能化程度，增强产品功能和性能，使人类生活更加舒适和方便。

(2) 仪器仪表

这是单片机应用最多、最活跃的领域之一。由于单片机体积小、成本低、运用灵活，易于产品化，它能方便地组装成各种智能化的控制设备和仪器仪表，做到机电一体化。其主要用于工业用智能仪器仪表、医疗器械和数字示波器等，不仅能提高测量精度和准确度，简化仪器硬件结构，减小仪器体积便于携带，还具有数据处理、分析和监控等功能，易于实现仪器仪表数字化和智能化。

(3) 工业控制

这是单片机应用的主要领域，单片机本身是按工业测控环境的要求设计的，其面向控制，能针对性地解决从简单到复杂的各种控制任务，获得最佳的性能价格比。其适用温度范围宽，在各种恶劣的环境下都能可靠工作，这是其他类型微处理器无法比拟的。无论过程控制、数据采集还是测控技术，都离不开单片机，单片机可以构成各种工业控制系统、数据采集系统等，同时可以方便地实现多机和分布式控制，使系统保持最佳工作状态，提高了系统工作效率和产品质量。

(4) 计算机外围设备与商用产品

目前，多数计算机外围设备如图形终端机、传真机、复印机、打印机、绘图仪、硬盘驱动器和智能终端机等都使用了单片机，可大大减轻主机负担；此外，单片机在一些商用产品，如自动售货机、电子收款机和电子秤等产品中也得到了广泛应用。

(5) 信息技术领域的应用

随着单片机的全面发展，以单片机为主的嵌入式系统在互联网和 IT 技术领域得到很大发展，被做成嵌入式的网络新产品，移动语言和手持数据、音视频和数字图像等消费类产品，如手机、可视电话、MP3、MP4、数码相机和数码摄像机等已经普遍应用单片机，而调制解调器、程控交换机、无线局域网和无线家居网等各种通信设备和系统也都已经开始应用。

MCS-51 系列单片机划分为民用级、工业级和军用级三个等级，其温度特性分别为：

民用级 0 ~ 70℃

工业用级 -40 ~ +85℃

军用级 -65 ~ +125℃

因此，在使用中应根据工作现场温度来适当选择芯片。

1.4.4 常用的系列单片机

1. MCS-51 系列单片机

MCS 系列单片机是 Intel 公司生产的单片机的总称。Intel 公司是生产单片机的创始者，其产品在单片机的各个发展阶段均具有代表性。但应用最广泛的还属 MCS-51 系列单片机。

MCS-51 系列单片机包括三个基本型 8031、8051、8751 和对应的低功耗型 80C31、80C51、87C51，以及三个增强型 8032、8052、8752 和对应的低功耗型 80C32、80C52、87C52，共有十几种芯片，见表 1-3。

表 1-3 MCS-51 系列单片机分类表

子系列	片内 ROM 形式			片内 ROM 容量/KB	片内 RAM 容量/B	片外存储器 寻址范围/KB	I/O 特性			中断源
	无	ROM	EPROM				计数器	并行口	串行口	
51 子系列	8031	8051	8751	4	128	2×64	2×16	4×8	1	5
	80C31	80C51	87C51	4	128	2×64	2×16	4×8	1	5
52 子系列	8032	8052	8752	8	256	2×64	3×16	4×8	1	6
	80C32	80C52	87C52	8	256	2×64	3×16	4×8	1	6

(1) 51 子系列基本型

51 子系列基本型包括 8031、8051 和 8751，其中 8031 内部无程序存储器，需要外扩 EPROM 芯片；8051 内部集成了 4 KB 的 ROM，但 ROM 内的程序是公司制作芯片时，代用户烧制的，一般用于大批量产品生产；8751 是在 8031 基础上增加了 4 KB 的 EPROM 可供用户修改、调试程序。

(2) 52 子系列增强型

52 子系列增强型包括 8032、8052 和 8752，与 51 子系列基本型相比，片内 ROM 从 4 KB 增加到 8 KB；片内 RAM 从 128 B 增加到 256 B；定时器/计数器从 2 个增加到 3 个；中断源从 5 个增加到 6 个；串行口通信速率提高了 5 倍。

(3) 低功耗型

表 1-3 的芯片型号中凡有字母“C”的表示采用 CMOS 工艺的 CHMOS 芯片，属于低功耗型产品。例如 8051 的功耗为 630 mW，而 80C51 的功耗只有 120 mW。此类单片机有 2 种节电工作方式：一是等待方式，CPU 停止工作，其他部分仍继续工作；二是掉电方式，此时片内振荡器停止工作，单片机内部所有运行状态停止，只有 RAM 继续保持数据，这对于便携式、手提式或野外作业仪器设备具有非常重要的意义。

2. 增强型 8051 单片机

常用的增强型 8051 单片机有以下几种：

- 1) Atmel 公司的 89 系列单片机，将 Flash 存储器技术与 80C5x 内核结合，形成片上带有 Flash 存储器的 AT89C5x、AT89S5x 系列单片机。
- 2) Philips 公司的 8051 内核单片机，如 P89V51RB2/RC2/RD2 等。
- 3) ST 公司的增强型 8051 内核单片机，如 μ PSD32 系列、 μ PSD33 系列单片机等。
- 4) SST 公司的 89 系列单片机，如 SST89E516RD 等。
- 5) Silicon Laboratories 公司的 C8051 系列单片机，如 C8051F020 等。

6) WinBond (华邦) 公司的 8051 内核单片机, 如 W78、W79 系列单片机。

下面介绍几种增强型 8051 单片机的功能及特点。

(1) Philips 系列单片机

Philips 公司生产的 89 系列单片机 (P89V51RB2/RC2/RD2) 是以 8051 为内核的增强型 8051 单片机, 片内包含 16/32/64 KB Flash ROM 和 1024 B 的数据 RAM, 典型特性是它的 X2 方式选项。利用该特性, 可以使应用程序以传统的 80C51 时钟频率 (每个机器周期包含 12 个时钟) 或 X2 方式 (每个机器周期包含 6 个时钟) 的时钟频率运行, 选择 X2 方式可以使单片机在相同时钟频率下获得 2 倍的总线数据的吞吐量。

Philips 的 89 系列增强型 8051 单片机内部的 Flash 程序存储器支持并行和串行在系统可编程 (In - System Programming, ISP)。ISP 允许在软件控制下对成品中的器件进行重复编程。P89V51RB2/RC2/RD2 也支持在应用可编程 (In - Application Programming, IAP), IAP 允许随时对 Flash 程序存储器重新配置, 即使是应用程序正在运行时也不例外。

Philips 的 P89 系列单片机的特性:

- 1) 16/32/64 KB 的片内 Flash 程序存储器, 具有 ISP 和 IAP 功能。
- 2) 通过软件或 ISP 选择支持 12 时钟 (默认) 运行或 6 时钟运行模式。
- 3) SPI (串行外围接口) 和增强型 UART (通用异步收发器 (异步串行通信口))。
- 4) PCA (可编程计数器阵列), 具有 PWM 和捕获/比较功能。
- 5) 4 个 8 位 I/O 口, 含有 3 个高电流 P1 口 (每个 I/O 口的电流为 16 mA)。
- 6) 3 个 16 位定时器/计数器。
- 7) 可编程看门狗定时器 (WDT)。
- 8) 8 个中断源, 4 个中断优先级。
- 9) 2 个 DPTR 寄存器。
- 10) 低 EMI (Electro Magnetic Interference) 方式: 控制 ALE 信号输出以降低 EMI 干扰等。

(2) SST89 系列单片机

SST89 系列是美国 SST 公司推出的高可靠、小扇区结构的 Flash 单片机, 特别是所有产品均带有 IAP (在应用可编程) 和 ISP (在系统可编程) 功能, 不占用户资源, 通过串行口即可在系统仿真和编程, 无须专用仿真开发设备, 3 ~ 5 V 工作电压, 低价格, 在市场竞争中占有较强的优势。

SST89 系列 Flash 单片机主要功能特性:

- 1) 大容量内部数据存储器 RAM。
- 2) 在应用可编程 (IAP) 和在系统可编程 (ISP), 可实现远程升级, 无须编程器。
- 3) 非易失性数据存储 (内部扩展 4 K/8 K EEPROM)。
- 4) 9 个中断源, 4 个中断优先级, 3 个大电流驱动引脚 (可直接驱动 LED)。
- 5) 双倍速, 6 时钟模式, 编程时可选择, 默认为 12 时钟模式。
- 6) 可编程计数器阵列 (PCA, PWM)。
- 7) 增强通用异步通信总线 UART, 支持地址自动识别和帧数据错误检测。
- 8) 看门狗定时器 (WDT)。
- 9) 宽工作电压范围 2.7 ~ 5.5 V, 低功耗。
- 10) 掉电检测功能等。

(3) C8051Fxxx 系列单片机

C8051Fxxx 系列单片机由 Silicon Laboratories 公司生产并具有与 8051 兼容的微控制器内核，与 MCS-51 系列单片机指令集完全兼容。除了具有标准 8052 的体系结构，片内还集成了数据采集和控制系统中常用的模拟部件和其他数字外设及功能部件，包括模拟多路选择器、可编程增益放大器、ADC、DAC、电压比较器、电压基准、温度传感器、SMBus/I²C、UART、SPI、可编程计数器/定时器阵列（PCA）、定时器、数字 I/O 端口、电源监视器、看门狗定时器（WDT）和时钟振荡器等。

C8051Fxxx 系列单片机具有大容量的 Flash 程序存储器和 256 B 的内部 RAM，有些单片机内部还有位于外部数据存储器空间的 RAM。C8051Fxxx 单片机采用流水线结构，机器周期由标准的 12 个系统时钟周期降为 1 个系统时钟周期，处理能力大大提高，最高可达 25MIPS。C8051Fxxx 单片机的 Flash 存储器还具有在系统重新编程能力，可用于非易失性数据存储。片内 JTAG 调试支持功能允许在应用系统上进行非侵入式（不占用片内资源）、全速、在系统调试。C8051Fxxx 系列单片机都可在工业温度范围（-45 ~ +85℃）内用 2.7 ~ 3.6 V（F018/019 为 2.8 ~ 3.6 V）的电压工作。各端口及 $\overline{\text{RST}}$ 和 JTAG 引脚都容许 5 V 的输入信号电压。

C8051Fxxx 系列单片机与标准 8051 相比，有几项关键性的改进，提高了整体性能：

1) 扩展的中断系统提供最高达 22（C8051F3xx 为 12）个中断源，允许大量的模拟和数字外设中断微控制器。

2) 有多达 7 个复位源：一个片内电源监视器、一个看门狗定时器、一个时钟丢失检测器、一个由比较器 0 提供的电压检测器、一个强制软件复位、CNVSTR 引脚及 $\overline{\text{RST}}$ 引脚。 $\overline{\text{RST}}$ 引脚是双向的，可接受外部复位或将内部产生的上电复位信号输出到 $\overline{\text{RST}}$ 引脚。除了 VDD 监视器和复位输入引脚以外，每个复位源都可以由用户用软件禁止。

3) 单片机内部有一个能独立工作的时钟发生器，在复位后被默认为系统时钟。如有需要，时钟源可以在运行时切换到外部振荡器。外部振荡器可以使用晶体、陶瓷谐振器、电容、RC 或外部时钟源产生系统时钟。这种时钟切换功能在低功耗系统中是非常有用的，它允许单片机从一个低频率的节电状态切换到高速（可达 16 MHz）的正常工作状态。

3. 其他单片机

除增强型 8051 单片机外，比较有代表性的单片机还有以下几种：

1) Microchip 公司的 PIC 系列单片机（<http://www.microchip.com>）。

2) Ti 公司的 MSP430 系列 16 位单片机（<http://www.ti.com>）。

3) Atmel 公司的 AVR 单片机（<http://www.atmel.com>）。

4) 台湾凌阳半导体公司的 SP 系列单片机（<http://www.sunplus.com.cn>）。

5) Zilog 公司的 Z8 系列单片机（<http://www.zilog.com>）。

6) 美国 UBICOM 公司的 SX 系列网络单片机。

还有许多其他的产品，在此不一一列举，用户可以根据自己的实际需要选择合适的单片机。尽管单片机的种类很多，但几乎所有单片机的基本工作原理都一样，主要区别在于内部硬件资源的不同，汇编语言的格式不同。

因此，掌握好 MCS-51 系列单片机的 51 子系列基本型是十分重要的，因为它们是 8051

内核单片机的基础。本书描述的 MCS - 51 系列单片机主要包含 8051、8751 和对应的低功耗型 80C51、87C51 单片机。在本书中，将 MCS - 51 系列单片机简称为 51 单片机，增强型 8051 单片机简称为增强型 51 单片机。

习题

1-1 微型计算机系统主要由哪几部分组成？微处理器、微型计算机和微型计算机系统的关系是什么？

1-2 微型计算机的分类方法包括哪几种？各应用在什么领域中？

1-3 说明什么是单片机，单片机有什么特点？

1-4 单片机主要应用在哪些领域？

1-5 上网搜索增强型 8051 内核单片机和专用单片机，解释何为外围电路内装化。

1-6 上网搜索单片机课程或单片机技术的学习网站，选择 1 ~ 2 个认为比较好的网站，列举其特点。

第2章 51 单片机的硬件结构

本章介绍 MCS-51 系列单片机的内部结构、CPU、存储器结构和 I/O 端口等内部硬件资源。学习 51 单片机的硬件组成和工作原理，掌握内部功能部件的作用和操作方法，是学习后续章节的基础，也是单片机应用系统设计的基础。

2.1 51 单片机的内部结构与引脚功能

2.1.1 51 单片机的基本组成

1. 51 单片机的基本组成结构

当前流行的以 8051 为内核的增强型单片机在功能、速度和容量等方面均比传统的 51 单片机有了很大进步，但是它们具有相似的基本组成结构和兼容的指令系统，51 单片机的基本组成结构如图 2-1 所示。

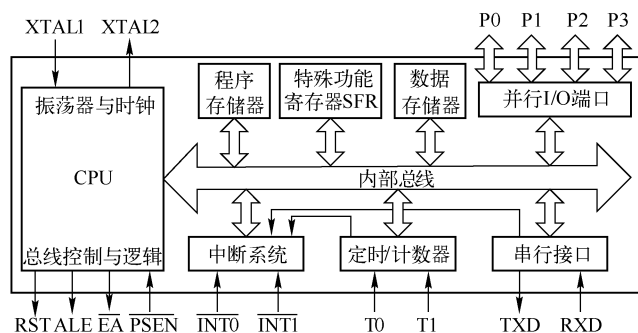


图 2-1 51 单片机的基本组成结构

从图 2-1 可以看出，51 单片机可以划分为三大模块：

(1) CPU（Central Processing Unit，中央处理器）

1) 8 位 CPU，含布尔处理器。

2) 时钟电路。

3) 总线控制与逻辑。

(2) 存储器系统

1) 程序存储器（ROM/EPROM/EEPROM/Flash），一般为 4~64 KB，可以外扩至 64 KB。

2) 数据存储器（RAM），一般为 128 B、256 B（52 单片机）至 4KB（增强型 51 单片机），可以外部扩展至 64 KB。

3) 特殊功能寄存器 SFR。

(3) I/O 口和其他功能单元

1) 4 个并行 I/O 口，共 32 个 I/O 引脚。

- 2) 2 个 16 位定时器/计数器。
- 3) 1 个全双工异步串行口 (UART)。
- 4) 中断系统 (5 个中断源, 2 个优先级)。

2. 51 单片机的内部结构

51 单片机的内部结构如图 2-2 所示。从图 2-2 中可以看出, 51 单片机集成了 CPU (ALU、ACC、B 和 PSW 等)、存储器系统 (程序存储器 ROM/EPROM/EEPROM/Flash 和数据存储器 RAM)、定时器/计数器、并行 I/O 接口、串行接口、中断系统和特殊功能寄存器 (SFR) 等, 各个部件通过内部总线紧密地联系在一起。

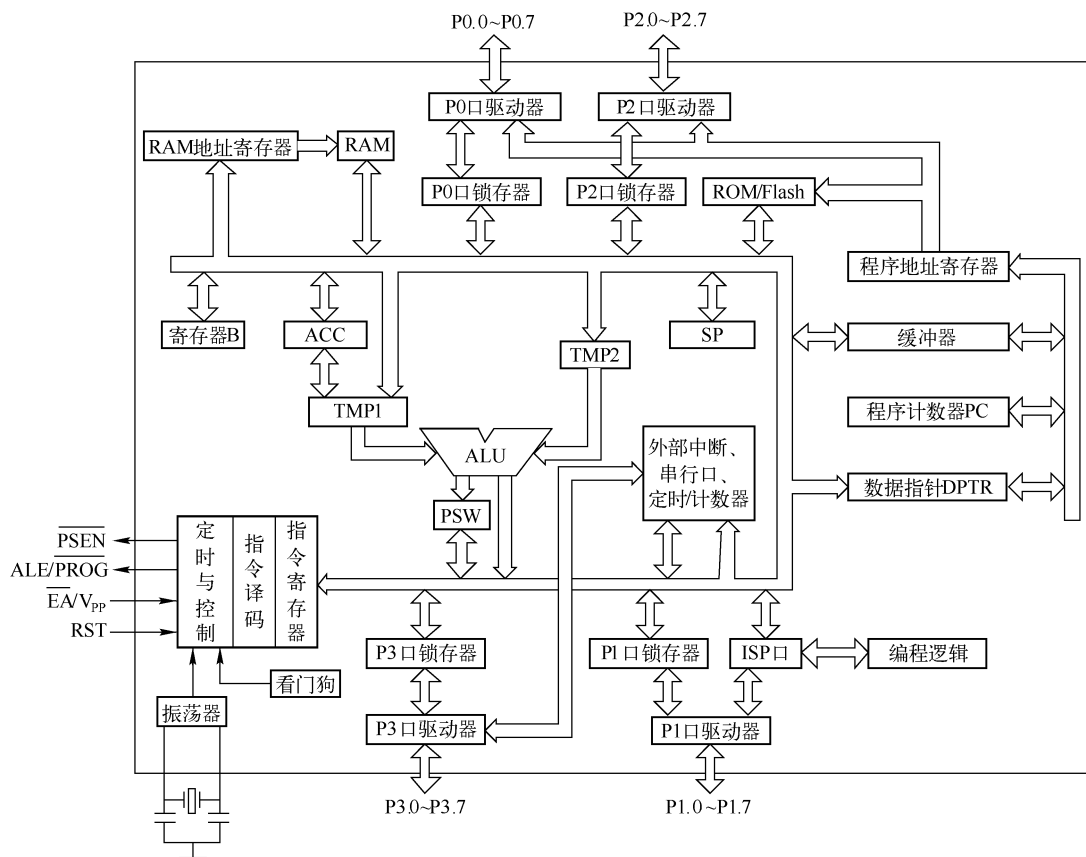


图 2-2 51 单片机的内部结构框图

3. 51 单片机的总线结构

总线是各部件进行信息交流的公共通道, 一般分为数据总线、地址总线和控制总线三大类。采用总线能够有效减少单片机的引脚数, 从而提高集成度和可靠性。51 单片机的总线结构如图 2-3 所示。三类总线所占用的引脚如下:

- 1) 数据总线 DB (Data Bus)。宽度 8 位, 由 P0 口 (D7 ~ D0) 提供。

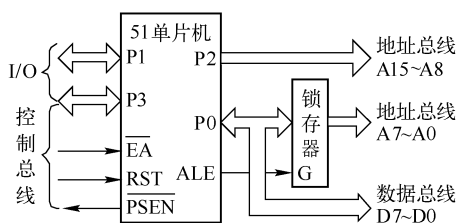


图 2-3 51 单片机的总线结构

2) 地址总线 AB (Address Bus)。宽度 16 位或 8 位, 可寻址 64 KB 的数据存储器和其他外部扩展部件, 以及 64 KB 的程序存储器。如果外部扩展地址空间大于 256 B, 则 P0 口 (P1.7 ~ P1.0; A7 ~ A0) 作为地址总线的低 8 位, P2 口 (P2.7 ~ P2.0; A15 ~ A8) 作为地址总线的高 8 位。如果外部扩展地址空间不大于 256 B, 则只需要 P0 口作为 8 位地址总线。

由图 2-3 可以看出, P0 口具有分时复用的功能。分时复用时, P0 口首先通过 ALE 将地址总线信息送入锁存器 (如 74HC373, G 为锁存器的使能端) 暂存, 然后再输出或输入数据信息, 从而实现分时复用。

3) 控制总线 CB (Control Bus)。控制总线由 P3 口的第二功能 (RXD、TXD、 $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 、T0、T1、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$) 和独立的控制线 (RST、 $\overline{\text{EA}}$ 、ALE、 $\overline{\text{PSEN}}$) 组成, 共有 12 条控制线。

2.1.2 51 单片机的引脚功能

51 单片机主要有 DIP (双列直插式)、QFP (方型扁平式)、SOP (表面贴装式、贴片) 和 LCC (无引脚芯片载体) 等几种形式封装, 如图 2-4 所示。

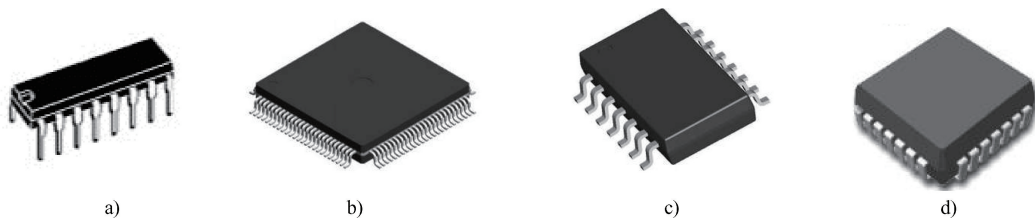


图 2-4 51 单片机的常见封装形式

a) DIP 封装 b) QFP 封装 c) SOP 封装 d) LCC 封装

单片机的引脚数有 40 (51 单片机)、20 (如 AT89S2051)、44 (如 STC15F2K60S2)、32 (如 C8051F930) 以及 28、48 等。DIP40 表示 51 单片机有 40 个引脚, 单片机主要的控制功能就是通过这些引脚来实现的, 51 单片机的 DIP40 封装的引脚图如图 2-5 所示。

从图 2-5 可以看出, 51 单片机的引脚可以划分为三大类:

(1) 电源及时钟引脚, 4 个引脚

1) V_{CC} (40 脚): 电源接入引脚, 一般为 +5 V 电源正端。

2) V_{SS} (20 脚): 接地引脚。

3) XTAL1 (19 脚, External Crystal, 外部晶振): 晶体振荡器接入的 1 个引脚。

4) XTAL2 (18 脚, External Crystal, 外部晶振): 晶体振荡器接入的另外 1 个引脚。

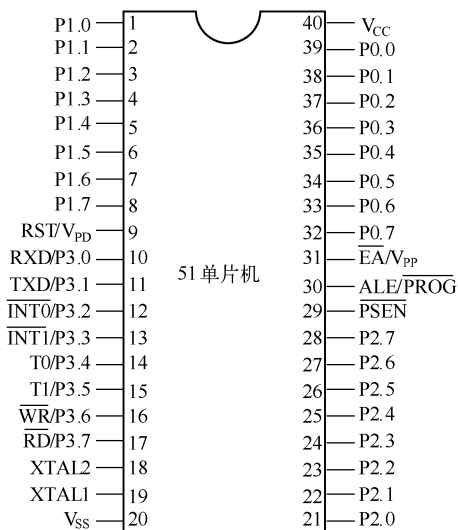


图 2-5 DIP40 封装的 51 单片机的引脚

(2) 控制线引脚, 4 个引脚

1) RST/V_{PD} (9 脚, Reset, 复位): 复位信号输入引脚/备用电源输入引脚。单片机正常工作时, 该引脚为 $\leq 0.5\text{ V}$ 的低电平。如果在该引脚出现 2 个机器周期以上的高电平将使单片机复位。V_{PD} 为该引脚的第二功能, 也就是备用电源输入端。当 V_{CC} 下降到某一规定值或在系统掉电时, 将备用电源 (+5 V) 自动接入 RST 端, 由 V_{PD} 为内部 RAM 提供备用电源, 以保证片内 RAM 的数据不丢失。

2) ALE/ $\overline{\text{PROG}}$ (30 脚, Address Lock Enable 地址锁存允许/ Programme 编程): 地址锁存器允许信号输出引脚/编程脉冲输入引脚。在访问外部程序存储器和外部扩展部件时, 这个输出信号用于锁存低 8 位字节地址。在对 EPROM 或 Flash 编程时, 该引脚用于输入编程脉冲 $\overline{\text{PROG}}$ 。一般情况下, ALE 是振荡器频率的 6 分频。因此可以将 ALE 作为外部定时或时钟使用 (可以驱动 8 个 LS 型的 TTL 负载), 也可以通过示波器观察 ALE 是否有脉冲输出用于判断 51 单片机是否损坏。

3) $\overline{\text{EA}}$ /V_{PP} (31 脚, External Address Enable, 外部地址允许): 外部程序存储器选择引脚/片内 EEPROM 或 Flash 的编程电压输入引脚。 $\overline{\text{EA}} = 0$, 则单片机从外部程序存储器中读取指令; $\overline{\text{EA}} = 1$, 则单片机先从内部程序存储器读取指令, 内部程序存储器的指令读完以后自动转向外部程序存储器。在对程序存储器编程时, 该引脚接编程电压 V_{PP}, 具体编程电压根据不同型号的 51 单片机而确定。

4) $\overline{\text{PSEN}}$ (29 脚, Programmer Saving Enable): 外部程序存储器选通信号输出引脚。读取外部程序存储器时, 该引脚低电平有效。执行读外部程序存储器的指令时, 每个机器周期内 $\overline{\text{PSEN}}$ 被启动 2 次; 在对外部数据存储器的读写指令的执行过程中, $\overline{\text{PSEN}}$ 不出现。 $\overline{\text{PSEN}}$ 同样可以驱动 8 个 LS 型的 TTL 负载。

(3) 并行 I/O 口引脚, 32 个引脚

1) P0.7 ~ P0.0 (32 ~ 39 脚): 一般 I/O 口引脚, 8 位数据/低 8 位地址总线复用引脚。

2) P1.7 ~ P1.0 (8 ~ 1 脚): 一般 I/O 口引脚。

3) P2.7 ~ P2.0 (28 ~ 21 脚): 一般 I/O 口引脚, 高 8 位地址总线引脚。

4) P3.7 ~ P3.0 (17 ~ 10 脚): 一般 I/O 口引脚, P3 口第二功能控制线引脚。

2.2 51 单片机的 CPU

CPU 是单片机的大脑, 它决定了单片机的指令系统和主要功能。CPU 主要由运算器和控制器两大部分组成, 主要完成取指令、分析指令和执行指令的操作, 以及协调各个部件的工作。

2.2.1 运算部件

运算器也称为运算部件, 是以 ALU (Arithmetic Logic Unit, 算术逻辑单元) 为核心, 加上累加器 ACC、B 寄存器、PSW、暂存器 TMP1、暂存器 TMP2 和布尔处理机等组成, 它可以实现数据的算术运算 (加、减、乘、除)、逻辑运算 (逻辑与、或、异或)、位单元处理

（置 1、清 0、取反）和数据传送（如传输、循环移位）等操作。

1. ACC

ACC（Accumulator，累加器）是一个 8 位累加器，它是 CPU 使用最频繁的寄存器。在进行 ALU 运算时，其中 1 个数据基本上来自 ACC。它一般用于存放参加运算的操作数和运算结果。在指令系统中用 A 表示 ACC。

2. B 寄存器

B 寄存器主要用于乘法和除法指令。其使用的情况如下：

- 1) 乘法指令：两个操作数分别取自 A 和 B，乘积结果高 8 位存放在 B，低 8 位存放在 A。
- 2) 除法指令：除数取自 B，被除数取自 A，除法结果的商存放在 A，余数存放在 B。
- 3) 其他情况：如果程序中没有应用乘除运算指令，B 可以作为通用寄存器使用。

3. PSW

PSW（Programmer Status Word，程序状态字）是一个 8 位的程序状态字寄存器，也称为标志寄存器，用于保存指令执行结果的状态信息，以供程序查询和判断。PSW 的字节地址为 D0H，具体各位含义见表 2-1。

表 2-1 PSW 各位的含义

位	PSW. 7	PSW. 6	PSW. 5	PSW. 4	PSW. 3	PSW. 2	PSW. 1	PSW. 0
名称	Cy	AC	FO	RS1	RS0	OV	—	P
含义	进位位	辅助进位	用户定义标志位	寄存器组选择	寄存器组选择	溢出标志	保留位	奇偶标志

Cy（Carry，进位）也称为 C，在指令中用 C 表示。在执行某些算术运算、逻辑操作类指令时，可被硬件或软件置位（SETB）、清 0（CLR）或取反（CPL）。如果运算结果在最高位存在进位或借位时，C = 1，反之 C = 0。

AC（Auxiliary Carry，辅助进位）也称为半进位，表示两个 8 位数运算时，低 4 位是否有进位或借位。当低 4 位相加或相减时，如果 D3 位向 D4 位有进位或借位，则 AC = 1，反之 AC = 0。在 BCD 码运算的十进制调整中，需要应用 AC 位的信息。

FO 为用户自定义标志位，用户可以根据自己的需要对 FO 赋予一定的物理含义（例如某设备的开关状态等）。FO 可以通过软件置位或清 0，并由程序测试 FO 控制程序的流向。

RS1 和 RS0（Register Select，寄存器选择）为寄存器组选择位，可以通过软件置位或清 0。RS1 和 RS0 要联合使用，用于在 4 组寄存器组中选择 1 组作为当前的工作寄存器。选择方式见表 2-2。

表 2-2 当前寄存器组的选择方式

RS1	RS0	当前工作寄存器组 R0 ~ R7
0	0	0 组（00H ~ 07H）
0	1	1 组（08H ~ 0FH）
1	0	2 组（10H ~ 17H）
1	1	3 组（18H ~ 1FH）

OV（Overflow，溢出）为溢出标志位。执行加法或减法指令时，OV 由硬件置位或清 0，以指示运算结果是否溢出。如果运算结果超过累加器 A 所能表示的范围（无符号数 0 ~ 255，

有符号数 $-128 \sim +127$) 时产生溢出, 此时 $OV = 1$ 。如果没有溢出, 则 $OV = 0$ 。另外, 在执行乘法 $MUL\ AB$ 和除法 $DIV\ AB$ 指令时, 如果乘积结果超出 255 或 B 所放的除数为 0, 则 $OV = 1$ 。

保留位为系统暂时没有用到的位, 未用。

P (Parity, 奇偶) 为奇偶标志位。每个指令周期都由硬件来置位或清 0, 以表示累加器 A 中 1 的个数是否为奇数。如果 A 中 1 的个数为奇数, 则 $P = 1$; 如果 A 中 1 的个数为偶数, 则 $P = 0$ 。串行通信中常用奇偶校验的方法来检验数据传输的可靠性。

4. 布尔处理机

布尔处理 (位处理) 机有 17 条位处理指令, 位处理指令中的位存储单元地址指向 51 单片机可位寻址的位地址空间, 以及特殊功能寄存器中能够位寻址的位存储单元。位处理指令通过置位、清 0、取反、逻辑与、逻辑或和位单元状态判断等指令, 实现快速灵活的位操作。

2.2.2 控制部件和振荡器

控制部件也称为控制器, 是单片机的神经中枢, 用于协调单片机各部件的数据传送、数据运算等操作, 并对外发出地址锁存 ALE 信号和外部程序存储器选通信号 $\overline{PSEN}$ 、发出数据存储器读写控制信号 $\overline{RD}$ 和 $\overline{WR}$ 、接收复位信号 RST 、接收外部程序存储器访问控制 $\overline{EA}$ 信号。控制部件包括定时和控制电路、IR (Instruction Register, 指令寄存器)、ID (Instruction Decoder, 指令译码器)、PC、SP、DPTR、缓冲器以及信息传输部件等。它以振荡信号为基准产生 CPU 时序, 从程序存储器中取出指令传输到指令寄存器, 然后在指令译码器中对指令进行译码, 产生指令执行所需的各种控制信号, 这些控制信号送到 51 单片机内部的各部件并控制它们完成相应的操作。以下对控制器的几个主要部件和振荡器进行介绍。

1. PC

PC (Program Counter, 程序计数器) 用于存放 CPU 要执行的下一条指令的地址。程序中的每条指令都有自己存放的地址 (程序存储器的某一地址存储单元), 当 CPU 要执行某条指令时, 就把该条指令的地址值 (PC 中的数值) 送到地址总线, 从程序存储器中该地址值的存储单元中读取指令。当 PC 中的地址值被送上地址总线后, PC 会自动指向 CPU 要执行的下一条指令的地址。系统复位后, PC 的初始值为 0000H, 因此 CPU 首先从程序存储器的 0000H 单元读取指令并译码执行。PC 不属于特殊功能寄存器 SFR, 没有地址, 因此不可寻址, 用户无法对它进行读写操作, 但是可以通过跳转、调用和返回等指令改变其内容, 用以控制程序按要求进行转移。

2. SP

SP (Stack Pointer, 堆栈指针) 是一个 8 位的特殊功能寄存器, 是堆栈地址指针, 即 SP 中的内容是 8 位地址值, 用于指向堆栈顶部在单片机内部 RAM 的位置。堆栈是单片机的临时存储区域, 按照先进后出的原则工作。系统复位后, SP 初始化为 07H, 使得堆栈从 08H 单元开始存放数据。由于 08H ~ 1FH 单元属于 1 ~ 3 组寄存器的存储区域, 因此应该将 SP 的内容改为 20H 或以上的值, 例如经常将 SP 的内容设置为 50H、60H 或 70H。SP 的初始值越小, 堆栈深度就越深。SP 的值可以通过软件改变, 在执行 PUSH、POP、子程序调用、中断

响应、子程序返回（RET）和中断返回（RETI）等指令时也会自动调整 SP 的值。

3. DPTR

DPTR（Data Pointer，数据指针寄存器）是一个 16 位的特殊功能寄存器，是访问 51 单片机外部扩展部件的地址指针，即 DPTR 的内容是 16 位的地址值。当 CPU 访问外部扩展的数据存储器和扩展的其他部件时，DPTR 作为地址指针，指向外部数据存储器或外部其他部件的某一单元的地址；当 CPU 访问程序存储器时，用 DPTR 作为基址寄存器。同时，也可以将 DPTR 分成高 8 位 DPH 和低 8 位 DPL 寄存器单独进行操作。例如：

```
MOV    DPH,#7FH
MOV    DPL,#0F0H
MOVX   A,@DPTR    ;DPTR 单元的内容(7FF0H)作为外部扩展部件的地址,将该地址
                    ;存储单元的内容传送给 A
```

4. 振荡器

51 单片机和所有的微处理器一样在运行时需要按照一定的时钟序列进行操作，也就是需要振荡器产生时钟信号。51 单片机内部有一个用于构成振荡器的高增益反相放大器，但是需要外接晶体振荡器作为反馈元件才能构成自激振荡器。其中，连接外部晶振的引脚 XTAL1 和 XTAL2 分别是该放大器的输入端和输出端。51 单片机产生振荡信号有两种方法：内部振荡和外部振荡，如图 2-6 所示。

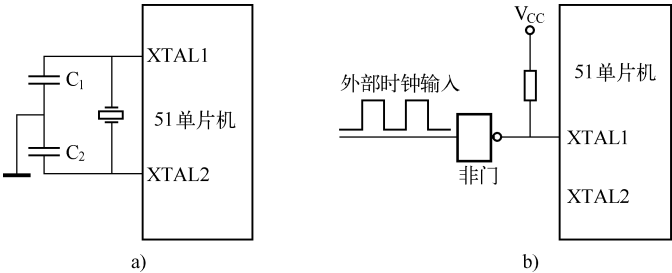


图 2-6 产生振荡信号的方法
a) 内部振荡 b) 外部振荡

内部振荡：单片机外接石英晶体振荡器或陶瓷谐振器，并将两个相同容量的电容接地（石英晶振所用电容取值范围为 22 ~ 33 pF，典型值为 30 pF；陶瓷谐振器所用电容取值范围为 40 ~ 47 pF）。

外部振荡：如果是 89C51 等 CHMOS 型单片机，将外部时钟信号经过上拉电阻后输入 XTAL1，而 XTAL2 悬空。如果是较老的 8051 等 HMOS 型单片机，则 XTAL1 接地，而外部时钟信号经过上拉电阻后输入 XTAL2。该方式经常用于多个单片机同时工作，以实现各块单片机之间的同步运行。此时一般采用 ≤12 MHz 的方波信号。

在实际使用中，基本上采用内部振荡的石英晶体振荡的方式。其中，石英晶振的频率为 1.2 ~ 12 MHz，较高的有 24 MHz、33 MHz 和 40 MHz 等。频率越高，单片机速度越快，但是对存储器、设计 PCB 板等的要求也就越高，容易出现不稳定的情况。因此，当前主要采用 12 MHz 和 11.0592 MHz（该频率主要用于通信）两种。现在，很多增强型 51 单片机内部集成了 RC 振荡源，无须外接晶振即可正常工作，频率误差也很小，简化了硬件设计。

2.2.3 CPU 时序

单片机在执行指令时，是将一条指令分解为若干个基本操作，这些基本操作所对应的脉冲信号的先后次序称为时序。

1. 4 种周期

51 单片机的时序划分为 4 种周期，也就是振荡周期（时钟周期）、状态周期、机器周期和指令周期，如图 2-7 所示。

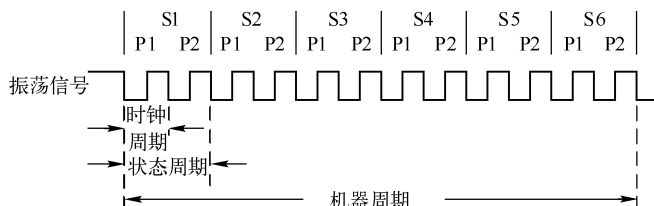


图 2-7 51 单片机的周期

振荡周期（时钟周期）：振荡脉冲的时钟周期，为时钟脉冲频率 f_{osc} 的倒数，是 51 单片机中最基本的时间单位。例如 12 MHz 晶振的时钟周期为 $1/12 \mu s$ 。

状态周期：每 2 个时钟周期为 1 个状态周期，用 S 表示。这 2 个时钟周期分别称为节拍 P1 和节拍 P2。节拍 P1 时通常完成算术逻辑运算，节拍 P2 时通常进行内部寄存器之间的传输。

机器周期：每 6 个状态周期构成 1 个机器周期，这 6 个状态周期分别用 S1、S2、…、S6 表示。每个机器周期包括 12 个节拍，这些节拍分别用 S1P1、S1P2、S2P1、…、S6P2 表示。

指令周期：执行一条指令所占用的全部时间，它以机器周期为单位。51 单片机最常用的指令多为单机器周期，乘法和除法为 4 机器周期，而其他指令为双机器周期。对于时钟脉冲频率在 12 MHz 晶振的条件下，执行一条指令所需的指令周期时间有 $1 \mu s$ 、 $2 \mu s$ 、 $4 \mu s$ 几种情况。

2. 取指令的时序

CPU 工作时，需要进行取指令、分析指令和执行指令几个步骤。下面以典型的取指令为例介绍 CPU 的工作时序。在每个机器周期内，地址锁存信号 ALE 出现 2 次有效信号，即 2 次高电平信号。第 1 次出现在 S1P2 和 S2P1 期间，第 2 次出现在 S4P2 和 S5P1 期间。

单字节单周期指令：在 1 个机器周期内发生 2 次读操作码的操作，但在 S4P2 期间所读的这个字节操作码被丢弃，如图 2-8 中的“读操作码 2”为无用读取。

双字节单周期指令：在 S4 期间读指令的第 2 个字节，图 2-8 中的“读操作码 2”为有效读取。

单字节双周期指令：在 2 个机器周期内发生 4 次读操作码的操作，但是由于是单字节指令，所以后面 3 次被丢弃，如图 2-9 所示。

现在，随着增强型 51 单片机的出现，每个机器周期包括 12 个振荡周期的局面也发生了改变。很多增强型 51 单片机每个机器周期只需要 1 个振荡周期。为了区分它们，传统 51 单片机称为 12T 单片机，只需要 1 个振荡周期的增强型 51 单片机称为 1T 单片机。1T 单片机执行指令的速度比 12T 单片机快得多。

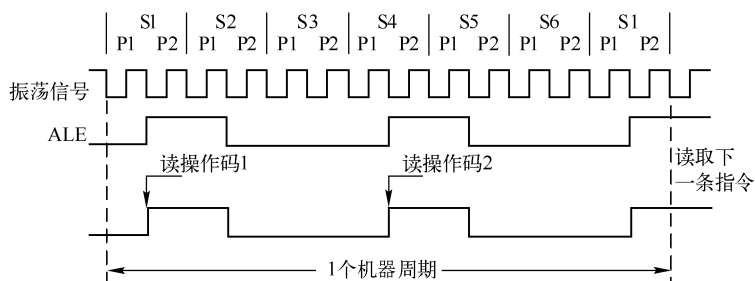


图 2-8 单周期指令的时序

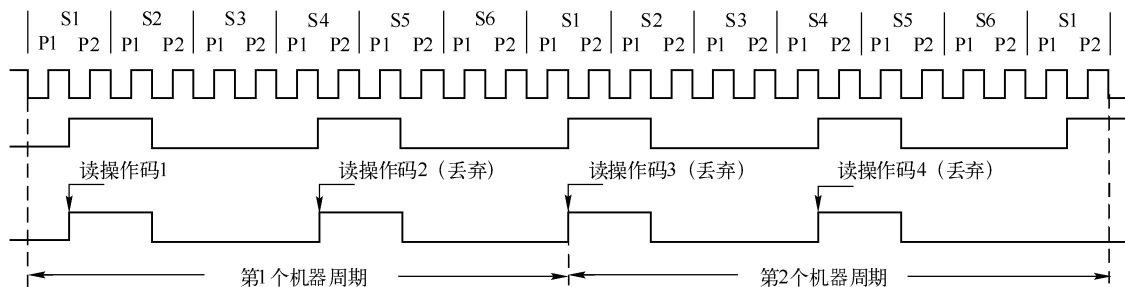


图 2-9 单字节双周期指令的时序

2.3 存储器结构

微处理器的存储器结构通常为两种，一种冯·诺依曼结构（也称普林斯顿结构），另一种是哈佛结构。

冯·诺依曼结构：将程序存储器和数据存储器合并在一起的存储器结构。程序存储地址和数据存储地址指向同一个存储器的不同物理位置，因此指令和数据的宽度相同。

哈佛结构：将程序存储和数据存储分开的存储器结构。CPU 首先到程序存储器中读取程序指令内容，对指令解码后得到数据单元地址，再到相应的数据存储器的该单元中读取数据，并进行下一步的操作（通常是执行）。程序存储和数据存储分开，可以使指令和数据有不同的数据宽度。

哈佛结构与冯·诺依曼结构的最大区别在于冯·诺依曼结构的计算机采用指令代码与数据的统一编址，而哈佛结构是独立编址的，指令代码存储空间与数据存储空间完全分开。

51 单片机采用哈佛结构，也就是程序存储器和数据存储器相互独立，典型的存储器结构如图 2-10 所示。

51 单片机的存储器结构有以下不同的划分方法：

- 1) 按照物理结构，分为片内程序存储器、片外程序存储器、片内数据存储器 and 片外数据存储器 4 个部分。
- 2) 按照功能划分结构，分为程序存储器、片内数据存储器、片外数据存储器、特殊功能寄存器和位寻址区 5 个部分。
- 3) 按照寻址空间划分结构，分为程序存储器、片内数据存储器、片外数据存储器 and 位寻址区 4 个部分。

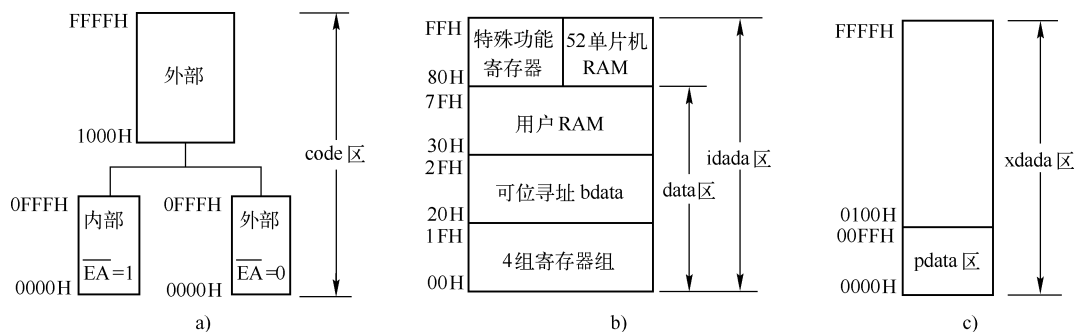


图 2-10 51 单片机典型的存储器结构

a) 程序存储器 b) 内部数据存储器 c) 外部数据存储器

寻址区 4 个独立的地址空间。

从图 2-10 可以看出，各个存储区域都有自己的命名，例如 code 区、dada 区、idata 区、bdata 区、pdata 区和 xdata 区，每个存储区域的容量大小、访问速度有很大差别，在编程时需要注意。

2.3.1 程序存储器

程序存储器（code 区）用于存放编写好的程序、常量和表格等，在 51 单片机执行程序时只能读不能写，掉电后数据不丢失。51 单片机执行程序时，从程序存储器中逐条取出指令并执行，如图 2-2 所示框图中的 16 位程序计数器 PC 用来存储要执行的下一条指令的地址。因此，程序存储器的每个存储单元（Byte，字节）都拥有一个唯一的 16 位地址（Address），可寻址空间为 64 KB（2 的 16 次方，65536），程序的第一条指令存放在地址值为 0000H 的存储单元（简称单元）中，程序存储器的最大地址值为 FFFFH。

程序存储器常被称为 ROM，实际上可以是 ROM、PROM、EPROM、EEPROM 或 Flash 等几种存储器。由于 Flash 具有电可刷写、在线刷写和价格便宜等优点，因此比例逐渐增加。在使用程序存储器时，需要注意以下几点：

1) 片内片外程序存储器统一编址。

2) 用 EA 引脚区分片内、片外程序存储器。EA = 1（接高电平）时，单片机先从内部程序存储器开始执行程序；EA = 0（接地）时，不管是否存在内部程序存储器，单片机都会先从外部程序存储器开始执行程序。

例如常见的 89S52 单片机内部有 8 KB（0000H ~ 1FFFH）Flash 程序存储器，如果 EA = 1，则单片机先从片内 0000H 开始执行程序，CPU 取出 1FFFH 地址单元的指令字节单元内容后，PC 的内容为 2000H，则自动跳转到片外程序存储器的 2000H 地址单元开始执行程序。

3) 采用 MOV C 指令进行查表。

4) 中断向量地址。51 单片机至少有 5 个中断向量地址，中断程序的入口地址被固定在程序存储器的几个单元中，见表 2-3。因此，编写程序时，如果存在中断服务程序，就不能占用这些入口地址，否则会出现错误。这时，应该在 0000H 地址存放一条绝对跳转指令，以便转入系统主程序的入口地址开始执行程序。

表 2-3 中断源的中断服务程序入口地址

中 断 源	入 口 地 址
INT0（外部中断 0）	0003H
T0（定时器/计数器 0）	000BH
INT1（外部中断 1）	0013H
T1（定时器/计数器 1）	001BH
串行口	0023H
T2（定时器/计数器 2）	002BH（52 单片机、STC89C51）

5) 程序存储器的扩展。如果选用的 51 单片机只有 4 KB 程序存储空间，而单片机应用系统需要的程序存储器容量将超过 4 KB 程序存储空间，应改选用内部程序存储器容量更大的增强型 51 单片机。

2.3.2 数据存储器

51 单片机内部的数据存储器与微处理器的内存类似，用于存放程序的变量（中间数据或运算结果），掉电后数据丢失。数据存储器常被称为 RAM，实际上可以是 SRAM、DRAM、EEPROM、Flash 等几种存储器。使用 RAM 时需要注意如下一些要点：

1) 片内 RAM 和片外 RAM 的地址空间互相独立，片内 RAM 只有 8 位地址，片外 RAM 可以是 16 位地址或 8 位地址。

2) 51 单片机片内 RAM 有 128 B（00H ~ 7FH）存储单元（简称单元），特殊功能寄存器区为 128 B（80H ~ FFH），所以划分为低 128 B（00H ~ 7FH）和高 128 B（80H ~ FFH）两个区域。其中低 128 B 存储区包括工作寄存器区、既可以字节寻址又可以位寻址的存储区（位寻址区）和数据缓冲区（或称用户 RAM 区）3 个部分。高 128 B 为特殊功能寄存器区。其分布见表 2-4。

表 2-4 内部数据存储器的分布

字节地址范围		用 途
高 128 B	80H ~ FFH	特殊功能寄存器区
	30H ~ 7FH	数据缓冲区
低 128 B	20H ~ 2FH	可字节寻址又可位寻址区
	18H ~ 1FH	第 3 组寄存器组
	10H ~ 17H	第 2 组寄存器组
	08H ~ 0FH	第 1 组寄存器组
	00H ~ 07H	第 0 组寄存器组

片内 RAM 的 00H ~ 1FH 为寄存器组区，见表 2-4。该区共有 0、1、2、3 组寄存器组，每组都拥有 R0、R1、…、R7 的 8 个通用寄存器，也就是说共有 32 个 8 位的通用寄存器。在任一时刻，单片机只能使用其中 1 组寄存器，此时需要通过设置程序状态字 PSW 寄存器的 RS1、RS0 来选择当前寄存器组（参看表 2-2）。系统复位时 RS1 = 0、RS0 = 0，默认选中的是第 0 组寄存器组。如果单片机程序不需要应用其他的寄存器组，可以将它们作为一般的

RAM 来使用。

片内 RAM 的 30H ~ 7FH 为数据缓冲区，也称为用户 RAM 区，共有 80 个字节单元，用于堆栈和存放数据。

3) 52 单片机除了原有的 256B 片内 RAM 外，还有与特殊功能寄存器地址重叠、物理上独立的 128B (80H ~ FFH) 通用 RAM。此时，如果使用直接地址则访问的是特殊功能寄存器，如果使用间接寻址则访问的是通用 RAM。例如：

```
MOV  90H, #79H    ;访问高 128 B 特殊功能寄存器的 90H 单元,直接寻址
MOV  R0, #90H      ;访问高 128 B 通用 RAM 的 90H 单元。首先将地址值 90H 赋值给 R0
MOV  @R0, #79H     ;需要通过 R0 间接寻址才能够访问高 128 B 通用 RAM 的 90H 单元
```

4) 片外 RAM 用 MOVX 指令访问。并行扩展片外 RAM 时，如果外部扩展的地址空间小于 FFH，只需要 P0 口作为地址总线；如果外部地址空间大于 FFH，则由 P2 口作为高 8 位地址，P0 口作为低 8 位地址组成 16 位地址。

2.3.3 特殊功能寄存器 (SFR)

51 单片机片内 RAM 的高 128B (80H ~ FFH) 为特殊功能寄存器 (Special Function Register, SFR)，也称为专用寄存器，主要有 ACC、B、PSW 等，特殊功能寄存器见表 2-5。

表 2-5 51 单片机 SFR 的名称、地址对照表

符 号		名 称	字 节 地 址	位地址范围
P0		P0 端口	80H	80H ~ 87H
SP		堆栈指针	81H	
DPTR	DPL	数据指针低字节	82H	
	DPH	数据指针高字节	83H	
PCON		电源控制寄存器	87H	
TCON		定时器/计数器控制寄存器	88H	88H ~ 8FH
TMOD		定时器/计数器控制方式寄存器	89H	
TLO		T0 的低字节	8AH	
TL1		T1 的低字节	8BH	
TH0		T0 的高字节	8CH	
TH1		T1 的高字节	8DH	
P1		P1 端口	90H	90H ~ 97H
SCON		串行口控制寄存器	98H	98H ~ 9FH
SBUF		串行数据缓冲寄存器	99H	
P2		P2 端口	A0H	A0H ~ A7H
IE		中断允许控制寄存器	A8H	A8H ~ AFH
P3		P3 端口	B0H	B0H ~ B7H
IP		中断优先级控制寄存器	B8H	B8H ~ BFH
PSW		程序状态字	D0H	D0H ~ D7H

(续)

符 号	名 称	字 节 地 址	位地址范围
ACC (或 A)	累加器	E0H	E0H ~ E7H
B	B 寄存器	F0H	F0H ~ F7H

SFR 的使用需要注意如下一些问题：

1) 在 80H ~ FFH 这 128 B 的字节单元中，SFR 只占用了其中一部分。而剩余的部分单元作为系统扩展的保留字节，用户不能使用，无法写入，读取所得到的数据也只是随机数据。

2) 可以通过直接寻址的方式对 SFR 进行访问，也可以使用寄存器名称（符号地址）进行访问。例如对累加器 ACC，可以使用其符号 A 或物理地址 E0H 进行访问。

3) 部分具有位寻址功能的 SFR 可以对它的某个位进行访问。

表 2-5 列出的是 51 单片机的特殊功能寄存器，增强型 51 单片机在表 2-5 的基础上增加了更多的特殊功能寄存器，比如双数据指针寄存器 DPTR1、辅助寄存器 AUXR1、辅助寄存器 AUXR、看门狗复位寄存器 WDTRST、可编程计数器阵列 PCA、脉宽调制 PWM、时钟分频 CLK_DIV、第 2 串口 S2BUF 和 SPI 控制寄存器等。不同型号的 51 单片机存在一定的差异，因此使用时需要参考具体型号对应的手册。

2.3.4 位寻址空间

51 单片机片内 RAM 可字节寻址又可位寻址的区域（bdata 区）的字节地址范围为 20H ~ 2FH，共有字节单元 16 个、位存储单元（简称位单元）128 个，见表 2-6。

表 2-6 位寻址区域地址表

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

位寻址区域的使用需要注意如下两点：

1) 可字节寻址又可位寻址的区域的字节地址范围为 20H ~ 2FH，位地址范围为 00H ~ 7FH，由于位地址值与字节地址值重叠，初学者容易混淆，需要把握和理解指令是对字节单元操作还是对位单元操作。

例如，MOV 20H, #57H 是字节操作指令，也就是对字节地址 20H 的整个字节进行操作，使得该字节单元所存储的数值变成 57H。

再如，SETB 07H 是位操作指令，也就是将位地址 07H（字节地址 20H 的最高位）的位单元内容设置为 1。

2) 位寻址区域可以通过“字节. 位”的形式访问，也可以直接通过位地址来访问。例如访问 20H 的最高位，可以通过 20H. 7 和 07H 两种形式来实现。

特殊功能寄存器 SFR 中的字节地址能够被 8 整除的有 11 个 SFR，这 11 个 SFR 也能够实现位寻址，位单元 82 个，见表 2-7。

表 2-7 特殊功能寄存器中的位单元地址分布

特殊功能寄存器	符号	字节地址	位地址与位名称							
			D7	D6	D5	D4	D3	D2	D1	D0
P0 口	P0	80H	87H P0. 7	86H P0. 6	85H P0. 5	84H P0. 4	83H P0. 3	82H P0. 2	81H P0. 1	80H P0. 0
定时器/计数器控制	TCON	88H	8FH TF1	8EH TR1	8DH TF0	8CH TR0	8BH IE1	8AH IT1	89H IE0	88H IT0
P1 口	P1	90H	97H P1. 7	96H P1. 6	95H P1. 5	94H P1. 4	93H P1. 3	92H P1. 2	91H P1. 1	90H P1. 0
串行控制	SCON	98H	9FH SM0	9EH SM1	9DH SM2	9CH REN	9BH TB8	9AH RB8	99H TI	98H RI
P2 口	P2	A0H	A7H P2. 7	A6H P2. 6	A5H P2. 5	A4H P2. 4	A3H P2. 3	A2H P2. 2	A1H P2. 1	A0H P2. 0
中断允许控制	IE	A8H	AFH EA	— —	— —	ACH ES	ABH ET1	AAH EX1	A9H ET0	A8H EX0
P3 口	P3	B0H	B7H P3. 7	B6H P3. 6	B5H P3. 5	B4H P3. 4	B3H P3. 3	B2H P3. 2	B1H P3. 1	B0H P3. 0
中断优先级控制	IP	B8H	— —	— —	— —	BCH PS	BBH PT1	BAH PX1	B9H PT0	B8H PX0
程序状态字	PSW	D0H	D7H Cy	D6H AC	D5H F0	D4H RS1	D3H RS0	D2HOV	— —	D0H P
累加器	A	E0H	E7H ACC. 7	E6H ACC. 6	E5H ACC. 5	E4H ACC. 4	E3H ACC. 3	E2H ACC. 2	E1H ACC. 1	E0H ACC. 0
B 寄存器	B	F0H	F7H B. 7	F6H B. 6	F5H B. 5	F4H B. 4	F3H B. 3	F2H B. 2	F1H B. 1	F0H B. 0

表 2-7 中，SFR 中可位寻址的每个位单元有一个位地址，并赋予每个位单元一个位名称。访问位单元主要有 4 种方法：

① 使用位地址。例如 PSW 最高位的位地址，可以用 D7H 表示。

② 使用位名称表示。例如 PSW 最高位的位地址，可以用 Cy 表示。

③ 使用“字节地址. 位”的形式表示。例如 PSW 的字节地址为 D0H，因此 PSW 最高位 Cy 的位地址可以用 D0H. 7 表示。

④ 使用“名称. 位”的形式表示。例如 P1 最高位的位地址，可以用 P1.7 表示。
在实际应用中主要采用第②种方法，以及第④种方法。

2.4 并行输入/输出端口

I/O 端口（Input/Output Port）又称为输入/输出接口，是单片机对外部装置实现控制和信息交换的必经之路。I/O 端口有并行和串行之分，串行口一次只能传送一位二进制数，并行口一次能传送一组二进制数。并行口的效率较高，因此 51 单片机设置有 4 个并行的 8 位双向 I/O 端口，也就是 P0（P0.7 ~ P0.0）、P1（P1.7 ~ P1.0）、P2（P2.7 ~ P2.0）和 P3（P3.7 ~ P3.0）。这 4 个端口都属于可以位寻址的 SFR，因此每个端口既能进行字节操作，也能对各个位进行独立的位操作。4 个端口的工作原理相似，但其功能和电路结构各具特点。

2.4.1 P0 端口

P0 口是一个 8 位漏极开路的三态双向端口，字节地址为 80H，位地址为 87H ~ 80H，能驱动 8 个 TTL（Transistor - Transistor Logic，晶体管 - 晶体管逻辑集成电路）负载。P0 口可以作为通用 I/O 口使用，也可以作为分时复用的低 8 位地址/8 位数据总线使用。

1. P0 口的内部结构

P0 口有 8 个完全一样、互相独立的位逻辑电路，其位逻辑电路内部结构如图 2-11 所示。

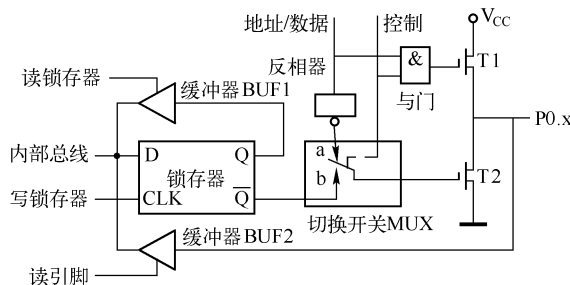


图 2-11 P0 口的位逻辑电路结构

图 2-11 的 P0 口的 1 位电路中，主要包括如下器件：

- 1) 1 个数据输出锁存器，用于数据位的锁存。
- 2) 2 个三态数据输入缓冲器，其中 BUF1 用于读锁存器数据，BUF2 用于读引脚数据。
- 3) 1 个切换开关 MUX，用于控制选择锁存器的输出和“地址/数据”信号之间的转换。
- 4) 1 个输出驱动电路，由 2 个场效应晶体管 T1 和 T2 组成，从而提高带负载能力。
- 5) 1 个输出控制电路，由与门、反相器（非门）和切换开关 MUX 组成。

2. P0 口作通用 I/O 口使用

P0 口作为通用 I/O 口使用时，对应控制信号为低电平，T1 处于截止状态，切换开关 MUX 将锁存器 Q 端与 T2 的栅极连接，T2 形成漏极开路输出，所以 P0 作为通用 I/O 口时，

要外接上拉电阻，通常外接 10 k Ω 左右的上拉电阻，如图 2-12 所示，否则 P0 口作为输出口使用时，无法输出高电平。

P0 口作为输入口使用时，应先向端口写“1”，因为端口引脚在内部直接与场效应晶体管连接。在图 2-13 中，如果锁存器 Q 端为“0”状态， $\bar{Q}$ 端为“1”状态，使得 T2 处于饱和导通状态，也就是 P0.x 的电平被 T2 钳制在“0”，导致引脚外部输入的电平不能得到正确的读取。因此，输入时要先写“1”使得 T1、T2 都处于截止状态，才能通过三态输入缓冲器 BUF2 正确读取引脚信号。

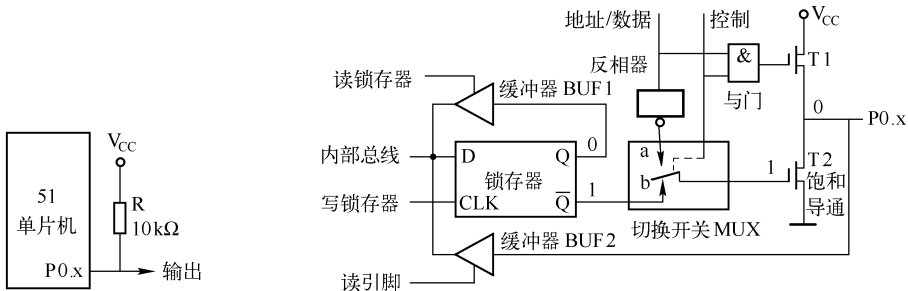


图 2-12 P0 口加上拉电阻输出

图 2-13 输入前锁存器 Q 端的原始状态为“0”

由于在读引脚时必须先用 1 条对端口置 1 的指令，多了一个附加的准备动作，因此这类 I/O 端口被称为准双向口。51 单片机的 4 个端口在作为通用 I/O 口使用时都是准双向口。

3. P0 口作低 8 位地址/数据复用总线使用

如果 51 单片机通过并行方式扩展外部存储器和其他外部部件，则 P0 口分时输入/输出 8 位数据和输出低 8 位地址，此时，P0 口是一个真正的双向口。

P0 口作为地址或数据输出时，控制信号为高电平，切换开关 MUX 将反相器输出端和 T2 的栅极连接，T1 的栅极受控于输出的地址或数据的电平控制。图 2-14 中，若内部总线上的地址或数据信号为“0”，则 T1 截止，T2 饱和，P0.x 引脚输出低电平，即 P0.x 端口输出信号“0”；若内部总线上的地址或数据信号为“1”，则 T1 饱和导通，T2 截止，P0.x 端口输出信号“1”。

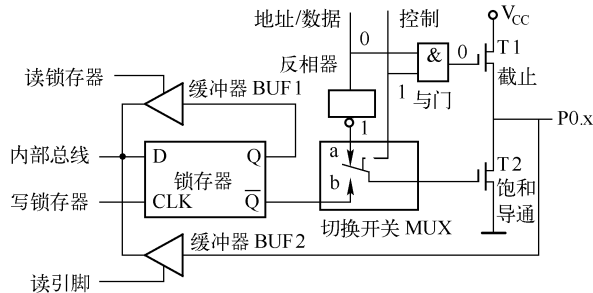


图 2-14 地址/数据输出信号“0”

P0 口作为数据输入时，控制信号为低电平，T1 截止，CPU 将自动向 P0 口写入 FFH，使得 T2 也截止，然后读入数据。

4. 对 Flash ROM 进行编程或校验时输入或输出指令代码

在对 Flash ROM 进行编程下载时，P0 口用于接收程序指令代码；校验时，P0 口输出程

序指令代码，此时需要外加上拉电阻。

2.4.2 P1 端口

P1 口的字节地址为 90H，位地址为 97H ~ 90H，能驱动 4 个 TTL 负载。P1 口的位逻辑电路结构如图 2-15 所示。P1 口内部有上拉电阻，与场效应晶体管 T1 共同组成输出驱动电路。

1. P1 口作通用 I/O 口使用

P1 口作为输出口使用时，由于内部有上拉电阻，可以直接控制输出信号。

P1 口作为输入口使用时，同样应先向端口写“1”，使得 T2 截止，从而正确地完成输入操作。

2. P1 口引脚复用功能

对于 52 单片机和部分增强型 51 单片机，P1 口还具备第二功能，见表 2-8。

表 2-8 52 单片机 P1 口的引脚复用功能

引脚	复用的第二功能
P1.0	T2（定时/计数器 2 的外部输入端）
P1.1	T2EX（定时/计数器 2 的外部触发端和双向控制）
P1.5	MOSI（SPI 总线的串行数据输入）
P1.6	MOSO（SPI 总线的串行数据输出）
P1.7	SCK（SPI 总线的串行时钟输出）

3. 对 Flash ROM 进行编程或校验时作 8 位地址

在对 Flash ROM 进行下载程序和校验程序时，P1 口作为低 8 位地址。

2.4.3 P2 端口

P2 口的字节地址为 A0H，位地址为 A7H ~ A0H，能驱动 4 个 TTL 负载。P2 口的位逻辑电路结构如图 2-16 所示。由于 P2 口是双功能端口，与 P0 口类似，P2 口的位逻辑电路结构也具有切换开关 MUX；P2 口与 P1 口类似，P2 口内部也有上拉电阻，能够直接控制输出信号。

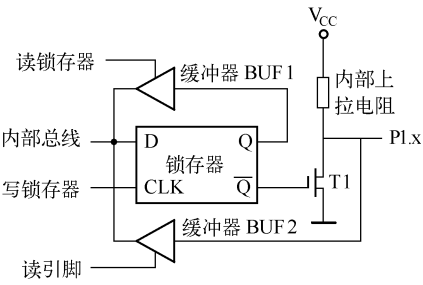


图 2-15 P1 口的位逻辑电路结构

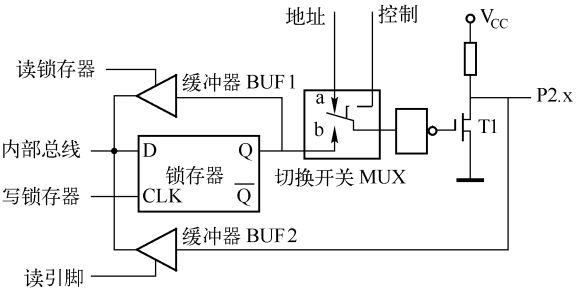


图 2-16 P2 口的位逻辑电路结构

1. P2 口作通用 I/O 口使用

P2 口作为通用 I/O 口使用时，在内部控制信号的作用下，锁存器的 Q 端经过一个非门与 T1 的栅极连接。

P2 口作为输出口使用时，由于内部有上拉电阻，可以直接控制输出信号。

P2 口作为输入口使用时，同样应先向端口写“1”，使得 T2 截止，从而正确地完成输入操作。

2. P2 口作高 8 位地址总线使用

当系统通过并行方式扩展外部存储器时，P2 口用于输出高 8 位地址（A15 ~ A8）。此时在 CPU 内部控制信号的作用下，内部地址总线经非门和 T1 的栅极连接。图 2-16 中，内部地址为“1”时，T1 截止，由于有上拉电阻，P2.x 口输出“1”；内部地址为“0”时，T1 饱和导通，P2.x 口输出“0”。

P2 口作高 8 位地址总线使用时，P2 口原有的数据会一直锁存在锁存器中，不受地址输出控制的影响。P2 口结束高 8 位地址总线使用后，此时在 CPU 内部控制信号的作用下，锁存器的 Q 端经过一个非门与 T1 的栅极连接，锁存在锁存器中原有的数据输出，相当于恢复原来的数据。

如果编写的 51 单片机程序只有访问 8 位外部地址（如 MOVX @ Rn）的指令时，不需要用到 P2 口，P2 口可以作为通用 I/O 口使用；如果编写的 51 单片机程序有访问 16 位的外部地址（如 MOVX @ DPTR）的指令时，P2 口输出高 8 位地址，不宜再作为通用 I/O 口使用。

3. 对 Flash ROM 进行编程或校验时作高 8 位地址

在对 Flash ROM 进行下载程序和校验程序时，P2 口作为高 8 位地址。

2.4.4 P3 端口

P3 口的字节地址为 B0H，位地址为 B7H ~ B0H，能驱动 4 个 TTL 负载。P3 口的特点是具有重要的第二功能，内部也有上拉电阻，它的位逻辑电路结构如图 2-17 所示。

1. P3 口作通用 I/O 口使用

P3 口作通用 I/O 口使用时，第二功能输出信号线保持高电平，图 2-17 的与非门输出状态受控于锁存器的 Q 端状态。

P3 口作为输出口使用时，由于内部有上拉电阻，可以直接控制输出。

P3 口作为输入口使用时，同样应先向端口写“1”，使得 T1 截止，从而正确地完成输入操作。

2. P3 口引脚复用功能

P3 口作第二功能使用时，锁存器的 Q 端必须为“1”状态，T1 的饱和导通和截止状态受控于第二功能输出信号线的状态。

当图 2-17 的 P3.x 口作第二功能的信号输入口时，第二功能输出信号线为“1”状态，T1 截止，通过控制 BUF3 能够正确完成输入操作。

当图 2-17 的 P3.x 口作第二功能的信号输出口时，如果第二功能输出信号线为“0”状

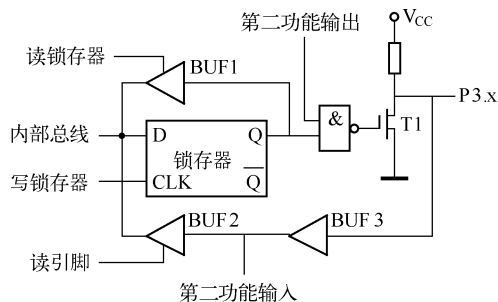


图 2-17 P3 口的位逻辑电路结构

态，T1 饱和导通，输出信号“0”；若第二功能输出信号线为“1”状态，T1 截止，输出信号“1”。

P3 口的第二功能见表 2-9。

表 2-9 P3 口引脚的第二功能

引脚	复用的第二功能
P3.0	RXD（串行输入口）
P3.1	TXD（串行输出口）
P3.2	$\overline{\text{INT0}}$ （外部中断 0 信号输入口）
P3.3	$\overline{\text{INT1}}$ （外部中断 1 信号输入口）
P3.4	T0（定时器 0 的外部计数信号输入口）
P3.5	T1（定时器 1 的外部计数信号输入口）
P3.6	$\overline{\text{WR}}$ （外部扩展部件的写信号输出口）
P3.7	$\overline{\text{RD}}$ （外部扩展部件的读信号输出口）

由于 51 单片机复位时，P3 口初始化状态为 FFH，也就是各锁存器 Q 端为“1”，可以立即作为第二功能使用。第二功能的输入信号 RXD、 $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 、T0、T1 经过缓冲器 BUF3 输入，而作为通用输入端口的输入信号则经过缓冲器 BUF2 输入。

2.4.5 端口电路小结

51 单片机的 4 个端口都能作为通用 I/O 口，但是也存在差异，表 2-10 为 4 个端口的比较。

表 2-10 4 个端口的比较

项 目	P0	P1	P2	P3
字节地址	80H	90H	A0H	B0H
位寻址地址	87H ~ 80H	97H ~ 90H	A7H ~ A0H	B7H ~ B0H
引脚编号	39 ~ 32	8 ~ 1	28 ~ 21	17 ~ 10
端口类型	作通用 I/O 时为准双向口（输入先写入 1） 作地址/数据分时复用时为真正三态双向口（CPU 自动写入 1）	准双向口 （输入先写入 1）	准双向口 （输入先写入 1）	准双向口 （输入先写入 1）
输出特性	需外加上拉电阻	内部有上拉电阻	内部有上拉电阻	内部有上拉电阻
驱动 TTL 负载数量/个	8	4	4	4
场效应晶体管数量/个	2	1	1	1
缓冲驱动器数量/个	2	2	2	3
多路切换开关	有	无	有	无
锁存器输出	$\overline{\text{Q}}$	$\overline{\text{Q}}$	Q 接非门	Q 接与非门
复用功能	8 位数据总线、 低 8 位地址总线	51 单片机没有，52 单片机有扩展的定时器、串口	地址线高 8 位（结束总线操作后，P2 口恢复原有的数据）	串口、外部中断、定时计数、外部存储器 和外部部件读写
Flash ROM 编程	下载程序、读取校验	低 8 位地址	高 8 位地址	无

I/O 端口是单片机应用系统中最重要部件，应该熟练掌握其使用方法。表 2-11 列出 I/O 端口的常用指令。

表 2-11 I/O 端口常用指令

指 令	功 能	例 子
ANL	逻辑与	ANL P0, A
ORL	逻辑或	ORL P1, A
XRL	逻辑异或	XRL P2, A
JBC	测试位为 1 就跳转并清 0	JBC P3.0, LOOP
CPL	位取反	CPL P3.1
INC	加 1	INC P0
DEC	减 1	DEC P1
DJNZ	减 1 后，如果结果不为 0 就跳转	DJNZ P2, LOOP
MOV P _{X.Y} , C	进位位送 P _{X.Y}	MOV P2.5, C
CLR P _{X.Y}	P _{X.Y} 清 0	CLR P2.6
SETB P _{X.Y}	P _{X.Y} 置 1	SETB P2.7

2.5 51 单片机复位操作与复位电路

2.5.1 复位操作与复位状态

复位是单片机的初始化操作，用于将单片机内部各部件恢复到初始状态，与计算机的重启类似。当单片机应用系统刚接通电源或者按下复位键时进入复位状态。复位信号在 RST 引脚上保持 24 个振荡时钟周期（2 个机器周期）以上的高电平，就会执行复位操作。51 单片机复位后，各寄存器和程序计数器 PC 的初始状态见表 2-12。

表 2-12 51 单片机复位后寄存器的初始状态

寄存器	初始状态值	寄存器	初始状态值	寄存器	初始状态值
PC	0000H	ACC	00H	B	00H
PSW	00H	SP	07H	DPTR	0000H
P0 ~ P3	FFH	IP	xxx0 0000B (51) xx00 0000B (52)	IE	0xx0 0000B (51) 0x00 0000B (52)
TMOD	00H	TCON	00H	TH0	00H
TL0	00H	TH1	00H	TL1	00H
SCON	00H	PCON	0xx0 0000B	SBUF	不定

复位后各寄存器大多为 0，但 P0 ~ P3 端口的输出都为 FFH，为信号输入做好了准备。(PC) = 0000H 说明复位后单片机从程序存储器的 0000H 单元开始取第一条指令执行。复位后 (PSW) = 00H，使得 RS1(PSW.4) = 0 和 RS0(PSW.3) = 0，说明默认选择的当前寄存器组是第 0 组。复位后 (SP) = 07H，说明复位后在片内 RAM 的 08H 单元开始建立堆栈。

2.5.2 复位电路

产生复位信号、使得单片机执行复位操作的电路称为复位电路。51 单片机通常采用上电复位和按键复位两种方法实现复位，常用的上电和按键复位电路如图 2-18 所示。

1) 上电复位：当接通电源的瞬间，RST 端与 V_{CC} 同电位，随着电容上的电压逐渐上升，RST 端的电压逐渐下降，于是在 RST 端便形成了一个正脉冲，只要该正脉冲的宽度持续 2 个机器周期的高电平，就可实现 51 单片机自动复位。

2) 按键复位（手动复位）：当人工按下复位按键后，RST 接通 V_{CC} ，只要保持 2 个机器周期的高电平，就可以实现 51 单片机复位。

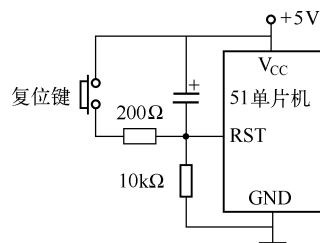


图 2-18 复位电路

2.5.3 时钟电路

通过外接晶振、电容和内部振荡器构成自激振荡电路，产生振荡时钟信号，使单片机能够在统一的时钟脉冲控制下进行有序的工作。同时，对振荡时钟信号进行 2 分频、6 分频、12 分频形成状态时钟信号、ALE 信号和机器周期信号。

振荡时钟信号除了供应单片机内部使用外，还可以通过 XTAL1 和 XTAL2 引脚对外部器件提供时钟信号，但需要增加驱动能力。其输出方式如图 2-19 所示。

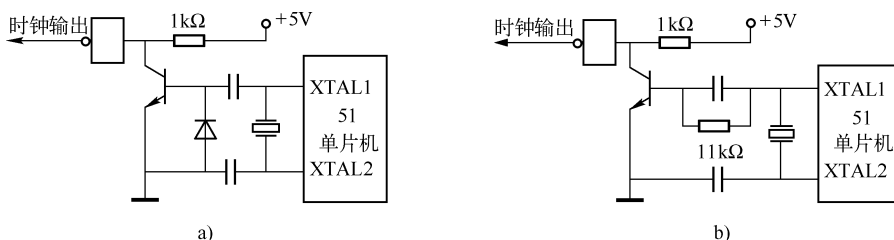


图 2-19 时钟信号输出电路

a) 从 XTAL1 输出 b) 从 XTAL2 输出

2.5.4 “看门狗”技术

WDT (Watch Dog Timer, 看门狗定时器) 是单片机的一个重要组成部分，51 单片机片内没有看门狗电路，很多型号的增强型 51 单片机内部设置了看门狗电路。由于程序设计不完善、雷电或强磁场等外部干扰的影响，单片机应用系统的程序执行可能会进入混乱、程序跑飞等异常状态，从而引发死机和不可预料的后果。为了实时监测单片机的运行状态是否正常，当单片机执行程序出现异常状态时，能够自动产生单片机复位信号，使得单片机重新从程序存储器的 0000H 单元取第一条指令开始执行程序。实现这种功能的技术称为看门狗技术，可用软件或硬件方法实现。

看门狗是一个定时电路，一般有一个输入端（喂狗端），用来复位定时器；一个输出端。当单片机正常执行程序时，每隔一段时间向 WDT 输入端送入一个信号，使看门狗定时器复位重新开始定时，这个过程叫“喂狗”。如果单片机在看门狗定时器的设定时间内不

“喂狗”（一般是程序“跑飞”），WDT 的定时器就会溢出并产生复位信号输出，使单片机复位。

软件看门狗是利用 51 单片机内部的定时器/计数器作为看门狗定时器，在单片机程序中适当地插入“喂狗”指令，当程序运行出现异常、跑飞或进入死循环时，看门狗定时器的定时时间到，利用软件将程序计数器 PC 赋予初始值 0000H，使得单片机恢复正常执行程序的状态。

专用硬件看门狗电路的一个输出端与 51 单片机的 RST 引脚连接。单片机正常执行程序时，每隔一段时间（例如 1 s 或 0.5 s）输出一个信号到喂狗端，使得 WDT 复位重新开始定时。如果程序跑飞或发生死循环时，在规定时间（溢出周期）内单片机自然无法向 WDT 输出定时喂狗信号，专用硬件看门狗没有获得“吃的”，就会跑出来“怒吼”，也就是定时时间到，看门狗输出端发出超时复位信号，使得 51 单片机重新从 0000H 单元开始执行程序。常用的硬件看门狗有 DS1232、MAX801、MAX813、X5045、IMP813 和 SP706 等。图 2-20 为 DS1232 构成的看门狗电路。

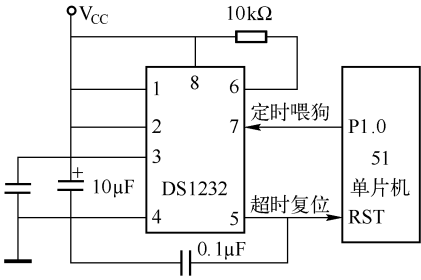


图 2-20 看门狗电路

2.6 51 单片机的低功耗运行方式

便携式或野外作业仪器设备多采用电池供电，对低功耗要求很高。51 单片机有两种节电方式：空闲（等待、待机）方式和掉电（停机）方式。

2.6.1 方式设定

51 单片机进入低功耗运行状态，主要是通过设置特殊功能寄存器 PCON 的 PD 和 IDL 的状态进行控制的。PCON 的格式见表 2-13。

表 2-13 PCON 的格式

位 单 元	名 称	作 用
PCON. 7	SMOD	波特率加倍。0 为不加倍，1 为加倍 T1 产生波特率且串行口工作方式 1、2、3 时，SMOD 有效
PCON. 6	—	保留位
PCON. 5	—	保留位
PCON. 4	—	保留位
PCON. 3	GF0	通用标志位
PCON. 2	GF1	通用标志位
PCON. 1	PD	为 1 则进入掉电方式（优先级别高）
PCON. 0	IDL	为 1 则进入空闲待机方式（优先级别低）

其工作原理如图 2-21 所示。

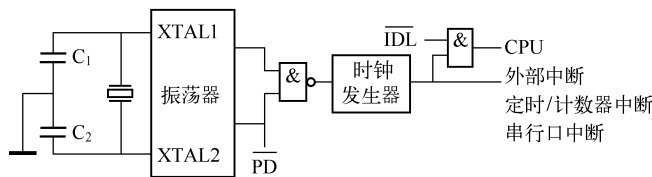


图 2-21 低功耗控制电路

当 $IDL = 1$ ，也就是 $\overline{IDL} = 0$ 时，该信号与时钟发生器的输出信号经过与非门后输出 0，使得 CPU 时钟被切断，CPU 停止工作，进入空闲待机方式。但是，由于振荡器和时钟发生器继续工作，因此中断系统还能够正常运行。

当 $PD = 1$ ，也就是 $\overline{PD} = 0$ 时，该信号与振荡信号经过与非门后输出恒为 1，使得振荡信号输出无效，进入掉电工作方式。

如果 $IDL = 1$ 且 $PD = 1$ ，由于 PD 在前级，优先级别比 IDL 高，因此单片机也进入掉电状态。

如果 $IDL = 0$ 且 $PD = 0$ ，单片机处于正常运行状态。

2.6.2 空闲工作方式

当 IDL 置位时，单片机进入空闲工作方式。此时 CPU 时钟被切断，但中断系统、定时器和串行口的时钟信号继续保持，所有的 SFR 保持空闲待机方式前的状态。

退出空闲工作方式有两种方法：

1) 中断退出。由于空闲待机状态下，中断系统还在正常工作，因此，如果发生定时器/计数器中断或外部中断信号、串行口中断信号，就会使得 IDL 位硬件清 0，从而退出空闲待机状态，进入中断服务程序。

2) 复位退出。单片机复位后，所有的寄存器都恢复为默认状态，PCON 的各位也被清 0。

在空闲状态下，禁止访问片内 RAM，但不禁止访问外部 RAM 和 I/O 端口。为了避免出现错误，在设置进入空闲模式的指令后面不要有写外部 RAM 或写 I/O 端口的指令。

2.6.3 掉电工作方式

当 $PD = 1$ 时，51 单片机进入掉电工作方式。此时振荡器停止工作，单片机的所有功能全部停止，功耗、工作电压和维持电流等都会下降，但是片内 RAM 和 SFR 的内容保持不变。

退出掉电工作方式为硬件复位。复位操作使得 SFR 的内容被初始化，但 RAM 的内容保持不变。退出掉电工作方式时，为了恢复 SFR 的原有数据，也可以在进入掉电工作方式前将这些数据写入外部扩展的 EEPROM（如 AT24C02）中。现在，很多增强型 51 单片机的掉电状态还保留外部中断功能，此时也可以通过外部中断退出掉电状态。

51 单片机的两种节电方式能够大大降低电源消耗。以 STC89C51 为例，正常工作电流为 25 mA，空闲待机时为 6.5 mA，掉电时为 50 μ A。经过不断发展，增强型 51 单片机的功耗也在不断下降。例如，STC15F2K60S2 的工作电流 4 ~ 6 mA，空闲待机时 < 1 mA，掉电时 < 100 nA。

再如，C8051F9xx 工作电压从 5 V 降到 0.9 V，工作电流为 1.5 ~ 4 mA，睡眠模式电流为 10 ~ 300 nA。因此，应该根据设计需求选择合适型号的 51 单片机或增强型 51 单片机。

习题

2-1 51 单片机片内集成了哪些逻辑功能部件？各个逻辑功能部件最主要的功能是什么？

2-2 51 单片机的数据总线、地址总线和控制总线分别有多少根？

2-3 字节单元和该字节的位单元之间有什么关系？位寻址与字节寻址之间有什么关系？

2-4 简述 PSW 各位的含义。

2-5 当 51 单片机晶振为 6 MHz 时，它的振荡周期（时钟周期）、状态周期、机器周期和指令周期分别为多长时间？

2-6 51 单片机的 4 个存储空间分别是什么？片内 RAM 中，包括哪 3 个主要部分，各部分的主要功能是什么？

2-7 单片机的程序存储器和数据存储器的地址空间如何分配？

2-8 52 单片机的 SFR 和高 128B 的用户 RAM 地址重叠了，访问时如何区分？

2-9 51 单片机有多少组寄存器组？每组有多少个通用寄存器？默认使用的是第几组？如何选择某组作为当前寄存器组？

2-10 51 单片机并行扩展外部存储器和外部部件时，P0、P2、P3 口分别传送的是什么信号？

2-11 如果手中有一台示波器，可通过观察哪个引脚的状态，大致判断 51 单片机正在工作？

2-12 请查询 STC89C51 与 MCS-51 单片机之间有何相同点，STC89C51 增加了哪些功能？

2-13 内部 RAM 中，位地址为 33H 的位单元，该位单元所在字节的字节地址为（ ）。

2-14 若 A 中的内容为 66H，那么，P 标志位的值为（ ）。

2-15 判断以下有关 PC 和 DPTR 的结论是否正确？

- A. DPTR 是可以访问的，而 PC 不能访问
- B. 它们都是 16 位的寄存器
- C. 它们都有自动加 1 的功能
- D. DPTR 可以分为两个 8 位的寄存器使用，但 PC 不能

2-16 PC 的值是（ ）。

- A. 当前正在执行指令的前一条指令的地址
- B. 当前正在执行指令的地址
- C. 当前正在执行指令的下一条指令的地址
- D. 控制器中指令寄存器的地址

2-17 51 单片机复位后，R4 所对应的单元地址为（ ），因上电时 PSW =（ ）。这时当前的工作寄存器区是（ ）组工作寄存器区。

第3章 51 单片机的指令系统

学习和设计微处理器应用系统的一个重要环节就是理解和熟练掌握它的指令系统。微处理器的指令系统是由生产厂商定义，对于不同型号系列的微处理器，其指令系统是不同的。本章将详细介绍 51 单片机指令系统的寻址方式及各类指令的格式和功能。

3.1 51 单片机的汇编指令

3.1.1 一个简单的程序示例

例 3-1 将内部 RAM 中的 20H 和 30H 单元的无符号数相加，结果存入 R0（高位）和 R1（低位）中。

程序流程图如图 3-1 所示，按照图 3-1 编写汇编程序源代码如下：

ORG	0000H	;ORG 为伪指令,表示程序从 0000H 开始执行
LJMP	MAIN	;长跳转指令,跳到标号 MAIN 处执行
ORG	0030H	;主程序起始地址 0030H
MAIN:	MOV A,20H	;20H 单元中的内容送至累加器 A 中
CLR	C	;清进位位 Cy
ADD	A,30H	;30H 中的内容与累加器 A 的内容相加, ;结果存放在累加器 A 中
MOV	R1,A	;将累加器 A 中的内容传送给寄存器 R1
CLR	A	;清累加器 A
ADDC	A,#00	;带进位加法指令
MOV	R0,A	;将累加器 A 的内容传送给 R0
SJMP	\$	;相对转移指令,\$是当前行指令的地址
END		;结束,伪指令

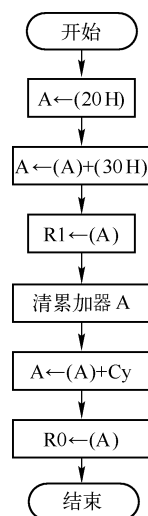


图 3-1 程序流程图

3.1.2 指令格式

指令的表示方法就是指令格式。51 单片机指令主要由操作码（指令助记符）字段和操作数字段组成。

指令格式：操作码 [操作数 1] [,操作数 2] [,操作数 3]

程序是指令的有序结合，指令在汇编程序中可以增加标号字段和注释字段，其格式如下：

[标号:] 操作码 [操作数 1] [,操作数 2] [,操作数 3] [;注释]

指令格式中各项的含义说明如下：

1) []：括号中的内容是可选项，其包含的内容因指令的不同可有可无。

2) 标号：根据编程的需要给指令设定的符号地址，可有可无；通常在子程序入口或转移指令的目标地址处才赋予标号。标号由 1~8 个字符组成，第一个字符必须是英文字母，不能是数字或其他符号，标号后必须有冒号。

3) 操作码：又称为指令助记符，是指令的核心部分，每条指令必须有操作码，由 2~5 个英文字母组成，如 JB、MOV、CJNE 或 LCALL 等。用于指示执行何种操作，如加、减、乘、除和传送等。

4) 操作数：表示指令操作的对象，可以是一个具体的数据，也可以是参加运算的数据所在的地址单元。操作数一般有以下几种形式：

① 没有操作数：操作数隐含在操作码中，如 RET 指令。

② 源操作数：数据的来源，通常指令中的第 2 个操作数是源操作数。

③ 目的操作数：指令执行结果存放的寄存器、存储单元、位单元，或转移的地址。

④ 只有一个操作数：如 INC A 指令，累加器 A 是源操作数，也是目的操作数。

⑤ 有 2 个操作数：如 MOV A, 30H 指令，操作数之间以逗号相隔，30H 是源操作数，累加器 A 是目的操作数。

⑥ 有 3 个操作数：如 CJNE A, #00H, LOOP 指令，累加器 A 和立即数 00H 是源操作数，LOOP 是目的操作数。

⑦ 如果操作数是立即数，并以字符 A~F 开头，则需要在它的前面加一个“0”，用以区别符号。如指令 MOV A, #0FBH，操作数 0FBH 为数据 FBH，如果不加“0”，则 #FBH 被认为是符号 FBH。

5) 注释：对指令的解释说明，用以提高程序的可读性，注释前必须加分号，分号后的所有字符均认为是注释内容。注释内容另起一行时，行前也要加分号。

3.1.3 指令类型

51 单片机汇编语言有 42 种操作码（参见附录 A），用来描述 33 种操作功能。一种操作码可以使用一种以上的数据类型。由于指令助记符规定了其访问的存储器空间，所以一种功能可能有几个助记符，如 MOV、MOVB、MOVW。操作码与寻址方式组合可得到 111 条指令。

1) 按指令所占字节数分类，有单字节指令（49 条）、双字节指令（45 条）和三字节指令（17 条）。

2) 按指令执行时间分类，有单周期指令（64 条）、双周期指令（45 条）和四周期指令（2 条：乘法指令和除法指令）。

3) 按功能分类，51 单片机指令系统可分为数据传送类指令（29 条）、算术运算类指令（24 条）、逻辑运算类指令（24 条）、控制转移类指令（17 条）和位操作类指令（17 条）。

3.1.4 常用符号说明

编写汇编语言指令时必须遵守一定的规则，表 3-1 给出了编写指令时的一些符号的约定含义，这些符号也可以应用于对指令的注释和说明。

表 3-1 指令中常见符号的约定含义

符 号	含 义
Rn	当前选定寄存器组的寄存器 Rn, n = 0 ~ 7
Ri	作为间接寻址的地址指针 Ri, i = 0 或 1
#data	8 位立即数, 8 位立即数的数据范围: 00H ~ FFH
#data16	16 位立即数, 16 位立即数的数据范围: 0000H ~ FFFFH
addr16	16 位地址, 可表示 64 KB 范围内的地址单元的寻址, 用于 LCALL 和 LJMP 指令中
addr11	11 位地址, 可表示 2 KB 范围内的地址单元的寻址, 用于 ACALL 和 AJMP 指令中, 以及 16 位地址指针寄存器 DPTR 的数据传输指令中
direct	8 位直接地址, 可以是片内 RAM 区的某一单元或某一特殊功能寄存器的地址
rel	带符号的 8 位地址偏移量, rel 地址偏移量的范围: -128 ~ +127, 用于 SJMP 和条件转移指令中
bit	位单元的 8 位地址, 位寻址区的直接寻址位, 表示片内 RAM 中可寻址位和 SFR 中可寻址位; 位单元的内容只有“0”和“1”两种状态, 作为源操作数在注释和说明时不加括号
(X)	X 是一个地址, (X) 是 X 地址单元中的内容。(X) 只用于对指令的注释和说明
((X))	X 是一个地址, X 地址单元中的内容作为地址指针, 即 (X) 仍然是一个地址, ((X)) 是 (X) 地址指针所指向的地址单元中的内容。((X)) 只用于对指令的注释和说明
→	按箭头指示方向传送内容, 主要用于对指令的注释和说明
\$	当前指令所在地址, 用于无条件转移指令中
/	在位单元之前, 表示该位状态取反参与操作, 但位单元本身状态不变, 用于位操作指令中
@	间接寻址寄存器或基址寄存器的前缀, 用于寄存器间接寻址和基址 + 变址寻址方式的指令中

3.2 51 单片机指令的寻址方式

在微处理器指令系统中, 操作数是指令的重要组成部分, 它指定了参加操作的数据或数据所在的地址单元。寻找源操作数地址的方式称为寻址方式。一条指令采用什么样的寻址方式是由指令的功能所决定的。寻址方式越多, 指令功能就越强, 灵活性就越大。只有透彻地理解寻址方式, 才能正确地应用指令。

51 单片机指令的操作数寻址方式主要有 7 种, 分别为直接寻址、立即寻址、寄存器寻址、寄存器间接寻址、基址 + 变址寻址、位寻址和相对寻址。

1. 直接寻址

直接寻址方式是指令中的源操作数是操作数的单元地址, 该单元地址指出了参与运算或传送的数据所在的字节单元或位单元地址。

例如, 已知(30H) = FFH, 执行指令:

MOV A,30H ;A←(30H),执行结果:(A) = FFH

该指令表示把内部 RAM 30H 单元中的内容传送给累加器 A。源操作数 30H 采用的是直接寻址方式。直接寻址的操作数在指令中以单元地址的形式出现, 而直接寻址方式只能给出 8 位地址, 因此, 这种寻址方式的寻址范围只限于片内 RAM, 具体来说:

- 1) 低 128B 单元在指令中直接以单元地址形式给出。
- 2) 特殊功能寄存器除可以用单元地址形式给出外, 还可以用特殊功能寄存器符号形式

给出。

2. 立即寻址

立即寻址方式是指令中含有立即数，立即数作为源操作数，即指令中的立即数以指令操作数的形式存放于程序存储器中。为了与直接寻址指令中的直接地址相区别，需要在操作数前面加前缀标志“#”；当立即数是以字母开头的数时，前面必须加0，如#0FEH。

例如，执行指令：

```
MOV    A,#0FEH      ;A←FEH,A 为目的操作数,A 中的内容为 FEH
MOV    DPTR,#8000H   ;DPTR←8000H,DPTR 中的内容为 8000H
```

第1条指令表示将立即数FEH送入累加器A中。第2条指令表示把16位立即数送入数据指针寄存器DPTR中，其中高8位送DPH，低8位送DPL。

3. 寄存器寻址

寄存器寻址方式指令中的源操作数是寄存器，源操作数的内容存放在寄存器内。寄存器寻址一般用于访问选定的通用寄存器R0~R7，以及A、B和DPTR。源操作数和目的操作数均为寄存器寻址方式的指令都是单字节指令。

例如，执行指令：

```
MOV    A,R0          ;A←(R0),指令中的R0为源操作数,A为目的操作数
INC     R0            ;(R0)+1→R0,指令中的R0是源操作数,也是目的操作数
MOV    A,30H         ;A←(30H),指令中的30H为源操作数,A为目的操作数
```

第1条指令的源操作数和目的操作数均为寄存器寻址方式；第2条指令为寄存器寻址方式；第3条指令的源操作数为直接寻址，目的操作数为寄存器寻址。

寄存器寻址的寻址范围如下：

- 1) 当前通用寄存器组：R0~R7。
- 2) 部分特殊功能寄存器。例如，累加器A、寄存器B和数据指针DPTR。

4. 寄存器间接寻址

寄存器间接寻址是指令中的某一个寄存器的内容作为操作数地址的寻址方式，即寄存器的内容作为地址指针，操作数的单元地址是通过寄存器间接得到。

寄存器间接寻址需要以寄存器符号的形式表示，为了区别寄存器寻址和寄存器间接寻址，在寄存器间接寻址方式中，必须在寄存器的名称前面加前缀标志“@”。访问内部RAM或外部数据存储器和其他扩展部件的低256个字节时，只能采用R0或R1作为间接寻址寄存器；访问外部扩展的64K单元地址空间和程序存储器则采用DPTR作为间接寻址寄存器。

例3-2 已知R0中存放片内RAM地址65H，数据指针DPTR中存放片外RAM地址8000H，地址65H单元中的数据为30H，8000H单元中的数据为59H，执行指令：

```
MOV    A,@R0         ;A←((R0)),执行指令:(A)=30H,@R0为源操作数
MOVX   A,@DPTR       ;A←((DPTR)),执行指令:(A)=59H,@DPTR为源操作数
```

寄存器间接寻址的寻址范围如下：

- 1) 51单片机片内RAM的低128B单元，52单片机片内RAM的256B单元：应用R0和R1作为间接寻址寄存器。

2) 片外扩展的 64KB 单元地址空间：使用 DPTR 作为间接寻址寄存器，如果 P2 口输出的高 8 位地址值是一个确定地址值，也可以使用 R0 和 R1 作为间接寻址寄存器。

例 3-3 已知片外 RAM 的 0118H 中的内容为 58H，执行指令：

```
MOV    P2,#01H      ;设置 51 单片机地址总线的高 8 位地址为 01H
MOV    R0,#18H      ;设置 51 单片机地址总线的低 8 位地址为 18H
MOVX   A,@R0        ;A←((R0)+0100H)=(0118H),(A)=58H
```

3) 程序存储器：使用 DPTR 作为间接寻址寄存器。

4) 片外扩展的低 256B 单元地址空间：可以使用 DPTR、R0 和 R1 作为间接寻址寄存器；或只扩展 256B 单元，使用 R0 和 R1 作为间接寻址寄存器。

5) 堆栈区：以堆栈指针 SP 作为间接寻址寄存器。

5. 基址 + 变址寻址方式

基址 + 变址寻址方式是指令中有操作数 @ A + PC 或 @ A + DPTR，程序计数器 PC 或数据指针 DPTR 作为基址寄存器，累加器 A 作为变址寄存器，基址寄存器和变址寄存器的内容相加形成读取程序存储器的 16 位操作数地址，用于访问程序存储器中的数据表格。

例 3-4 执行指令：

```
MOVC   A,@ A + PC    ;((A) + (PC))→A,@ A + PC 为源操作数
MOVC   A,@ A + DPTR  ;((A) + (DPTR))→A,@ A + DPTR 为源操作数
```

@ A + PC 是将 (A) + (PC) 的值作为地址指针指向程序存储器的一个单元，@ A + DPTR 将 (A) + (DPTR) 的值作为地址指针指向程序存储器的一个单元。

6. 位寻址

51 单片机有一个布尔处理器，可以对 8 位字节中的位单元进行操作，对位单元的寻址就是位寻址方式。除了位判断转移指令外，源操作数和目的操作数都是位单元。

例 3-5 执行指令：

```
MOV    C,40H        ;该指令的功能是把 40H 位单元中的状态送进位 Cy
SETB   TR0           ;该指令的功能是把 TR0 位单元中的状态置为“1”状态
```

位寻址的寻址范围如下：

1) 片内 RAM 中的位寻址区，位单元地址范围：00H ~ 7FH。

2) 可位寻址的特殊功能寄存器，可以直接使用位单元的符号，比如第 2 条指令中的 TR0，参见表 2-7 特殊功能寄存器中的位单元地址分布。

7. 相对寻址

相对寻址用于访问程序存储器，只出现在转移指令中。以 PC 的当前值加上指令中给出的相对偏移量 (rel) 形成转移地址。其中，rel 是一个带符号的 8 位二进制数，以补码形式置于操作码之后存放。程序的转移范围以 PC 当前值为中心，介于 -128 ~ +127 之间。

例如，执行指令：

```
JC    rel
```

设 rel = 85H，Cy = 1，这是一条以 Cy 状态为条件的转移指令，指令为两字节指令，CPU 取出第二个字节时，PC 当前值为原 PC 值加 2。由于 Cy = 1，程序转向 (PC) + 2 + rel 程序存

储器的目的地址单元去执行。注意，此时 rel = 85H 为负值。

编写单片机汇编程序时，偏移量 rel 不需要编程者自己计算，而是在程序中将偏移量 rel 用标号取代，比如：JC LOOP，汇编程序自动计算 LOOP 指向转移的目的地址。

3.3 51 单片机的指令集

按照指令的功能分类，51 单片机指令系统的 111 条指令分为数据传送类指令、算术运算类指令、逻辑运算类指令、控制转移类指令和位操作类指令。按照指令的功能分类理解指令便于记忆和查阅按照指令功能分类的指令表。只要熟练掌握了 51 单片机的存储器的结构，7 种寻址方式和每条指令的功能类型，对于理解和掌握每条指令的功能是不困难的。

3.3.1 数据传送类指令

数据传送类指令共有 29 条。这类指令将源操作数传送到指定的目的操作数的地址单元，传送后源操作数保持不变。数据传送类指令可以在累加器 A、R0 ~ R7 工作寄存器与片内数据存储器、片外扩展的数据存储器和其他扩展部件之间进行数据传送，从程序存储器读入数据到累加器 A 中。数据传送类指令一般不影响标志位，但如果目的操作数是累加器 A，将影响 SPW 的奇偶标志位 P，以及堆栈操作 POP 0D0H 指令可以直接修改程序状态字 PSW。

1. 内部数据的传送指令

1) 以累加器 A 为目的操作数的指令，即累加器 A 为目的地址单元，指令见表 3-2。

表 3-2 以累加器 A 为目的操作数的传送指令

指令助记符	操作说明	指令助记符	操作说明
MOV A,Rn	寄存器寻址，(Rn)→A	MOV A,direct	直接寻址，(direct)→A
MOV A,@ Ri	寄存器间接寻址，((Ri))→A	MOV A,#data	立即寻址，data→A

例 3-6 将立即数 35H 送入片内 30H 单元，可执行以下指令：

```
MOV    A,#35H           ;累加器 A 为目的地址单元,35H→A
MOV    R0,#30H          ;R0 为目的地址单元,30H→R0
MOV    @ R0,A            ;@ R0 为目的地址单元,(R0) = 30H,(30H)→A
```

也可执行以下这条指令：

```
MOV    30H,#35H         ;35H→30H
```

2) 以寄存器 Rn 为目的操作数的指令，指令见表 3-3。

表 3-3 以寄存器 Rn 为目的操作数的传送指令

指令助记符	操作说明	指令助记符	操作说明
MOV Rn,A	寄存器寻址，(A)→Rn	MOV Rn,direct	直接寻址，(direct)→Rn
MOV Rn,#data	立即寻址，data→Rn		

例 3-7 若(A) = 20H, (20H) = F0H, 执行以下指令:

```
MOV    R0,A          ;(R0) = 20H
MOV    R0,20H         ;(R0) = F0H
MOV    R6,#0A5H       ;(R6) = A5H
```

3) 以直接地址为目的的操作数的指令, 指令见表 3-4。

表 3-4 以直接地址为目的的操作数的传送指令

指令助记符	操作说明	指令助记符	操作说明
MOV direct,A	寄存器寻址, (A)→direct	MOV direct,Rn	寄存器寻址, (Rn)→direct
MOV direct1,direct2	直接寻址, (direct2)→direct1	MOV direct,@ Ri	寄存器间接寻址, ((Ri))→direct
MOV direct,#data	立即寻址, data→direct		

例 3-8 设 66H 单元中的内容为 EAH, 执行下面指令:

```
MOV    36H,#66H       ;66H→36H,(36H) = 66H
MOV    R1,36H          ;(36H)→R1,(R1) = 66H
MOV    7EH,@ R1        ;((R1))→7EH,(7EH) = (66H) = EAH
MOV    38H,7EH         ;(7EH)→38H,(38H) = (7EH) = EAH
```

4) 以寄存器间接地址为目的的操作数的指令, 指令见表 3-5。

表 3-5 以寄存器间接地址为目的的操作数的传送指令

指令助记符	操作说明	指令助记符	操作说明
MOV @ Ri,A	寄存器寻址, (A)→(Ri)	MOV @ Ri,direct	直接寻址, (direct)→(Ri)
MOV @ Ri,#data	立即寻址, data→(Ri)		

注: @ Ri 为目的操作数, (Ri) 表示一个地址单元。

例 3-9 若(A) = 20H, (R0) = 42H, (32H) = 46H, (42H) = 52H, 执行以下指令:

```
MOV    @ R0,A          ;((R0)) = (42H) = (A) = 20H,42H 单元的内容被刷新
MOV    @ R0,32H        ;((R0)) = (42H) = (32H) = 46H,42H 单元的内容被刷新
MOV    @ R0,#0D5H      ;((R0)) = (42H) = D5H,42H 单元的内容被刷新
```

5) 16 位数据传送指令, 指令如下:

```
MOV    DPTR,#data16    ;立即寻址,16 位地址值 data16→DPTR
                        ;16 位地址值高 8 位送 DPH,低 8 位送 DPL
```

这是 51 单片机中唯一的一条 16 位数据传送指令。

例 3-10 执行指令:

```
MOV    DPTR,#1234H     ;DPH←12H,DPL←34H,(DPTR) = 1234H
```

6) 堆栈操作指令 PUSH、POP, 指令见表 3-6。

堆栈操作是通过堆栈指针 SP 来实现, 分为入栈操作和出栈操作, 先进后出。51 单片机开机或复位后(SP) = 07, 一般需要重新设定 SP 的初始值, SP 的内容就是栈顶的位置 (地址)。堆栈指令有两条, 分别为进栈指令和出栈指令。

表 3-6 堆栈操作指令

指令助记符	入栈指令操作说明	指令助记符	出栈指令操作说明
PUSH direct	首先 $SP \leftarrow (SP) + 1$ ，然后 direct 单元的内容传送到堆栈指针 SP 指向的单元中	POP direct	首先堆栈指针 SP 指向的单元的内容传送到 direct 单元中，然后 $SP \leftarrow (SP) - 1$

例 3-11 已知 $(A) = 10H$ ， $(B) = 40H$ ，执行指令：

```

MOV    SP, #30H    ; 设置堆栈指针:  $(SP) = 30H$ ,
PUSH   ACC          ; ①  $SP \leftarrow (SP) + 1, (SP) = 31H$ , ②  $((SP)) = (31H) = (A) = 10H$ 
PUSH   B            ; ①  $SP \leftarrow (SP) + 1, (SP) = 32H$ , ②  $((SP)) = (32H) = (B) = 40H$ 
POP    ACC          ; ①  $(SP) = (32H) = 40H \rightarrow A$ , ②  $(SP) = (SP) - 1, (SP) = 31H$ 
POP    B            ; ①  $(SP) = (31H) = 10H \rightarrow B$ , ②  $(SP) = (SP) - 1, (SP) = 30H$ 

```

5 条指令的执行过程如图 3-2 所示。其执行的最终结果： $(SP) = 30H$ ； $(A) = 40H$ ； $(B) = 10H$ ，堆栈指针 SP 仍然指向最初的 30H 单元，实现累加器 A 和寄存器 B 中的数据互相交换。

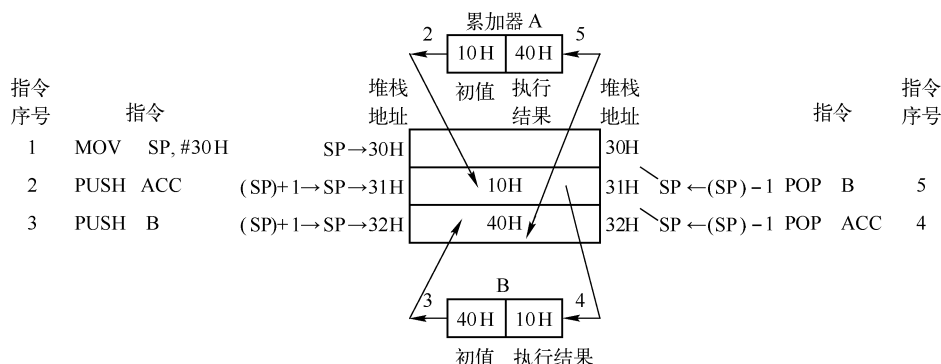


图 3-2 例 3-11 的 5 条指令的执行示意图

7) 字节交换指令和半字节交换指令，指令见表 3-7。

表 3-7 字节交换指令和半字节交换指令

指令助记符	操作说明	指令助记符	操作说明
XCH A, Rn	字节交换, $(Rn) \leftrightarrow (A)$	XCH A, direct	字节交换, $(direct) \leftrightarrow (A)$
XCH A, @ Ri	字节交换, $((Ri)) \leftrightarrow (A)$	XCHD A, @ Ri	半字节交换, $(A[D3 \sim D0]) \leftrightarrow ((Ri)[D3 \sim D0])$
		SWAP A	累加器 A 的高半字节与低半字节交换, $(A[D7 \sim D4]) \leftrightarrow (A[D3 \sim D0])$

例 3-12 已知 $(A) = 80H$ ， $(R0) = 28H$ ， $(28H) = 36H$ ，执行指令：

```

XCH    A, @ R0      ;  $((R0)) \leftrightarrow (A)$ ,  $(R0) = 28H$ , 28H 单元和 A 互相交换数据
                      ; 使得  $(A) = 36H$ ,  $((R0)) = (28H) = 80H$ 
SWAP   A            ;  $(A[D3 \sim D0]) \leftrightarrow (A[D7 \sim D4])$ , 使得  $(A) = 63H$ 

```

例 3-13 已知 $(A) = 80H$ ， $(R0) = 08H$ ， $(08H) = 36H$ ，执行指令：

XCHD A,@ R0 ;((R0)) = (08H) = 36H 的低 4 位与 (A) = 80H 的低 4 位交换,
 ;使得 (08H) = 30H, (A) = 86H
 SWAP A ;(A) = 86H 的高半字节 8 和低半字节 6 交换,使得 (A) = 68H

2. 累加器 A 与片外 RAM 或外部扩展接口部件的数据传送指令

累加器 A 与片外 RAM 或外部扩展接口部件相互之间的数据传送指令见表 3-8。

表 3-8 累加器 A 与片外 RAM 或外部扩展接口部件的数据传送指令

指令助记符	操作说明	指令助记符	操作说明
MOVX A,@ Ri	寄存器间接寻址, $A \leftarrow ((Ri))$	MOVX A,@ DPTR	寄存器间接寻址, $A \leftarrow ((DPTR))$
MOVX @ Ri,A	寄存器间接寻址, $(Ri) \leftarrow (A)$	MOVX @ DPTR,A	寄存器间接寻址, $(DPTR) \leftarrow (A)$

注: @ Ri 和 @ DPTR 分别为目的操作数时, (Ri) 和 (DPTR) 分别表示一个地址单元。

表 3-8 中的这组指令采用 16 位 DPTR 作间接寻址时, 可寻址整个片外的 64 KB 地址空间, 高 8 位地址 (DPH) 由 P2 口输出, 低 8 位地址 (DPL) 由 P0 口输出, 51 单片机 30 引脚输出地址锁存 ALE 信号。采用 R0 和 R1 进行间接寻址时, 8 位地址在 R0 或 R1 中, 由 P0 口输出; 如果有高 8 位地址, 则高 8 位地址由 P2 口输出, 组成 16 位地址。执行向累加器 A 传送数据的指令时, 51 单片机 P3.7 口产生读信号 $\overline{RD}$; 执行累加器 A 向片外 RAM 或外部扩展接口芯片传送数据时, 51 单片机 P3.6 口产生写信号 $\overline{WR}$ 。

51 单片机应用系统扩展的片外 RAM 和外部接口部件只能够通过累加器 A 进行数据传输, 而不能够与 51 单片机内部 RAM 或其他特殊功能寄存器直接进行数据传输。

例 3-14 已知 (DPTR) = 2000H, (2000H) = 10H, 执行指令:

MOVX A,@ DPTR ; $A \leftarrow ((DPTR)) = (2000H)$, (A) = 10H

例 3-15 已知 (P2) = 20H, (R1) = 48H, (A) = 60H, 执行指令:

MOVX @ R1,A ; $(R1) + 2000H \leftarrow (A)$, 则片外 (2048H) = 60H

3. 查表指令

查表指令只有两条, 均采用基址 + 变址寻址方式, 单字节指令, 用于读取程序存储器中的数据, 指令见表 3-9。

表 3-9 查表指令

指令助记符	操作说明	指令助记符	操作说明
MOVC A,@ A+PC	$PC \leftarrow (PC) + 1$, $(A) \leftarrow ((A) + (PC))$	MOVC A,@ A+DPTR	$A \leftarrow ((A) + (DPTR))$

例 3-16 设 (A) = 30H, 执行指令:

MOVC A,@ A+PC

若该指令的单元地址为 1000H, 则下一条指令单元的地址为 1001H, 执行该条指令将 $(A) + (PC) = 1031H$ 作为地址指向程序存储器, 将程序存储器 1031H 单元的内容传送给累加器 A。

这条查表指令以 PC 作为基址寄存器, A 作为变址寄存器, 当前的 PC 值指向该条指令

的下一条指令的地址，而累加器 A 的内容为 0 ~ 255，所以 (A) 和 (PC) 相加所得到的地址只能在该查表指令以下的 256 个地址单元之内，因此所查表格的大小受到限制。

例 3-17 设(DPTR) = 8200H, (A) = 30H, 执行指令：

MOVC A, @ A + DPTR ; (A) + (DPTR) = 8230H, 将程序存储器的(8230H)→A

这条查表指令以 DPTR 作为基址寄存器，A 作为变址寄存器，通常 DPTR 中的内容作为地址指向表格的首地址，而 A 的内容作为变址指向某一数据在表格的位置。表格大小和位置可在 64 KB 程序存储器中任意安排，应用该指令之前对 DPTR 和累加器 A 赋值，就能使一个表格被各个程序共用，因此查表时，常用该查表指令。

3.3.2 算术运算类指令

算术运算类指令有加、减、乘、除法指令，以及加 1 和减 1 指令，共 24 条指令。大多数指令都以累加器 A 作为一个源操作数，另一个源操作数可以存放于任何一个工作寄存器 Rn 或片内 RAM 单元中，也可以是一个立即数，运算结果存放在目的操作数累加器 A 中。

1. 加法指令

1) 不带进位的加法指令，指令见表 3-10。

表 3-10 不带进位的加法指令

指令助记符	操作说明	指令助记符	操作说明
ADD A, Rn	$A \leftarrow (A) + (Rn)$	ADD A, direct	$A \leftarrow (A) + (direct)$
ADD A, @ Ri	$A \leftarrow (A) + ((Ri))$	ADD A, #data	$A \leftarrow (A) + data$

2) 带进位的加法指令，指令见表 3-11。

不带进位的加法指令和带进位的加法指令的执行对状态标志位进行设置，如果 D7 位有进位，则置进位位 Cy 为 1，否则 Cy 清 0；如果 D6 位有进位而 D7 位没有，或者 D7 位有进位而 D6 位没有进位，则置溢出标志 OV 为 1，否则清 0；如果 D3 位有进位，则置辅助进位位 AC 为 1，否则清 0。

表 3-11 带进位的加法指令

指令助记符	操作说明	指令助记符	操作说明
ADDC A, Rn	$A \leftarrow (A) + (Rn) + Cy$	ADDC A, direct	$A \leftarrow (A) + (direct) + Cy$
ADDC A, @ Ri	$A \leftarrow (A) + ((Ri)) + Cy$	ADDC A, #data	$A \leftarrow (A) + data + Cy$

例 3-18 设(A) = 45H, (R0) = 20H, (20H) = ADH, 执行指令：

01000101
ADD A, @ R0 ; + 10101101
11110010

结果：(A) = F2H, Cy = 0, AC = 1, OV = 0, P = 1。

例 3-19 设(A) = 85H, (20H) = FFH, Cy = 1, 执行指令：

ADDC A, 20H ; 10000101 + 11111111 + 1 = [1]10000101, 产生进位 1

结果：(A) = 85H, Cy = 1, AC = 1, OV = 0, P = 1。

3) 加 1 指令，指令见表 3-12。

表 3-12 加 1 指令

指令助记符	操作说明	指令助记符	操作说明	指令助记符	操作说明
INC A	$A \leftarrow (A) + 1$	INC Rn	$Rn \leftarrow (Rn) + 1$	INC direct	$direct \leftarrow (direct) + 1$
INC @Ri	$(Ri) \leftarrow ((Ri)) + 1$	INC DPTR	$DPTR \leftarrow (DPTR) + 1$		

若表 3-12 的指令中的操作数原来为 FFH，执行加 1 指令后其结果为 00H。第一条指令对累加器 A 的加 1 操作将影响奇偶标志位 P，其他加 1 指令的执行都不会对 PSW 中的任何标志产生影响。INC DPTR 指令是对 DPTR 中的 16 位二进制数加 1。

例 3-20 设 $(A) = FFH$ ， $(R3) = 0FH$ ， $(30H) = E2H$ ， $(R0) = 40H$ ， $(40H) = AAH$ ，执行指令：

```

INC    A           ;FFH + 1 → A, (A) = 00H, PSW 的 P 状态设置为 0
INC    R3          ;0FH + 1 → R3, (R3) = 10H, PSW 状态不变
INC    @R0         ;((R0)) = (40H), AAH + 1 → 40H, (40H) = ABH, PSW 状态不变

```

4) 十进制调整指令：计算机的运算以二进制为基础，对于十进制 BCD 码相加应用 ADD 和 ADDC 指令的运算结果都是二进制数，要得到正确的十进制数运算结果，就必须应用十进制调整指令。指令如下：

DA A ;十进制调整指令，必须应用在 ADD 或 ADDC 指令的后面

这条指令对累加器 A 中由上一条加法指令所获得的 8 位结果进行十进制调整，调整过程由 51 单片机自动处理完成。

十进制调整指令执行后，程序状态字 (PSW) 中的进位标志位 (Cy) 表示结果的百位值。

例 3-21 $(A) = 56H$ ， $(30H) = 77H$ ，编程实现 BCD 码 $56 + 77$ 的运算，执行指令：

```

ADD    A, 30H      ;(A) + (30H) = 56H + 77H = CDH
DA     A           ;个位数为 DH, DH + 6 = 3, 进位 1
                     ;十位数为 CH, CH + 6 + 1 = 3, 进位 1 → Cy
MOV    R0, A       ;把和的低字节的十位数和个位数存入 R0
CLR    A           ;0 → A
ADDC   A, #00      ;百位数: 0 + Cy → A
MOV    R1, A       ;和的高字节的百位数存入 R1

```

运算结果为 133，并将其十进制 BCD 码存放在 R1R0 中，R1 为 01H，R0 为 33H。

2. 减法指令

1) 减法指令，指令见表 3-13。

表 3-13 减法指令

指令助记符	操作说明	指令助记符	操作说明
SUBB A, Rn	$A \leftarrow (A) - (Rn) - Cy$	SUBB A, direct	$A \leftarrow (A) - (direct) - Cy$
SUBB A, @Ri	$A \leftarrow (A) - ((Ri)) - Cy$	SUBB A, #data	$A \leftarrow (A) - data - Cy$

减法只有带借位的减法指令。在进行减法操作时，如果运算结果产生借位，Cy 置 1，否则清 0；如果 D3 位需借位，则将辅助进位标志位 AC 置 1，否则将 AC 清 0；如果 D7 需借位

而 D6 位不借位或 D6 位借位而 D7 位不借位，则将溢出标志位 OV 置 1，否则清 0；奇偶标志位 P 随着累加器 A 中 1 的个数而变化。

例 3-22 设(A) = 76H，立即数为 C5H，Cy = 0，执行指令：

$$\begin{array}{r} \text{SUBB A, \#0C5H} \quad ; \quad \begin{array}{r} 01110110 \\ 11000101 \\ - \quad \quad 0 \\ \hline \text{借位 [1]00110001} \end{array} \end{array}$$

结果：(A) = 31H，Cy = 1，AC = 0，OV = 1，P = 0。

2) 减 1 指令，指令见表 3-14。

表 3-14 减 1 指令

指令助记符	操作说明	指令助记符	操作说明
DEC A	$A \leftarrow (A) - 1$	DEC Rn	$Rn \leftarrow (Rn) - 1$
DEC direct	$direct \leftarrow (direct) - 1$	DEC @ Ri	$(Ri) \leftarrow ((Ri)) - 1$

若表 3-14 的指令中的操作数原来为 00H，执行减 1 指令后其结果为 FFH。第一条指令对累加器 A 的减 1 操作将影响奇偶标志位 P，其他减 1 指令的执行都不会对 PSW 中的任何标志产生影响。

例 3-23 设(A) = 0EH，(30H) = 00H，(R1) = 70H，(70H) = FFH，执行指令：

DEC A ; $A \leftarrow (A) - 1$, (A) = 0DH, P = 1, 不影响其他标志
DEC 30H ; $30H \leftarrow (30H) - 1$, (30H) = FFH, PSW 状态不变
DEC @ R1 ; $(R1) \leftarrow ((R1)) - 1$, (70H) = FEH, PSW 状态不变

3. 乘法和除法指令

乘法和除法指令见表 3-15。

表 3-15 乘法和除法指令

指令助记符	操作说明	指令助记符	操作说明
MUL AB	$(A) \times (B)$ ，结果低 8 位→A，高 8 位→B	DIV AB	$(A)/(B)$ ，商的整数→A，余数→B

51 单片机的乘法和除法指令只能对两个 8 位无符号数进行乘法和除法运算。

MUL AB 这条指令将对 Cy、OV 和 P 三个标志位产生影响。如果乘积大于 255 (FFH)，溢出标志 OV 置 1，否则 OV 清 0；进位标志 Cy 总是清 0；P 随着 A 中 1 的个数而变化。

例 3-24 设(A) = 4EH，(B) = 5DH，执行指令：

MUL AB ; 运算结果：(B) = 1CH，(A) = 56H，即乘积结果(BA) = 1C56H

执行 DIV AB 这条指令之前，如果原来 B 中的内容为 0，即除数为 0，则执行结果 A 和 B 中的内容不定，并将溢出标志 OV 置 1，在其他情况下，OV 被复位为 0，表示除法操作是合理的。进位标志 Cy 总是清 0。

例 3-25 设(A) = BFH，(B) = 32H，执行指令：

DIV AB ; 运算结果：(A) = 03H，(B) = 29H，Cy = 0，OV = 0

3.3.3 逻辑运算及移位类指令

逻辑运算指令可以完成两个 8 位二进制数，按位进行与、或、异或的逻辑运算，以及对累加器 A 的内容清 0 和按位取反操作。移位类指令是对累加器 A 的循环移位操作，包括左移一位、右移一位，以及带与不带进位标志位移位的方式。两类指令共有 24 条。

1. 逻辑与指令 ANL

逻辑与指令见表 3-16。

表 3-16 逻辑与指令

指令助记符	操作说明	指令助记符	操作说明	指令助记符	操作说明
ANL A, Rn	$A \leftarrow (A) \wedge (Rn)$	ANL A, direct	$A \leftarrow (A) \wedge (\text{direct})$	ANL A, @ Ri	$A \leftarrow (A) \wedge ((Ri))$
ANL A, #data	$A \leftarrow (A) \wedge \text{data}$	ANL direct, A	$\text{direct} \leftarrow (\text{direct}) \wedge (A)$	ANL direct, #data	$\text{direct} \leftarrow (\text{direct}) \wedge \text{data}$

例 3-26 若 $(A) = 07H$, $(R0) = FCH$, 执行指令:

$$\begin{array}{r} 00000111 \\ \wedge 11111100 \\ \hline 00000100 \end{array}$$

ANL A, R0 ;

结果: $(A) = 04H$ 。

2. 逻辑或指令 ORL

逻辑或指令见表 3-17。

表 3-17 逻辑或指令

指令助记符	操作说明	指令助记符	操作说明	指令助记符	操作说明
ORL A, Rn	$A \leftarrow (A) \vee (Rn)$	ORL A, direct	$A \leftarrow (A) \vee (\text{direct})$	ORL A, @ Ri	$A \leftarrow (A) \vee ((Ri))$
ORL A, #data	$A \leftarrow (A) \vee \text{data}$	ORL direct, A	$\text{direct} \leftarrow (\text{direct}) \vee (A)$	ORL direct, #data	$\text{direct} \leftarrow (\text{direct}) \vee \text{data}$

例 3-27 若 $(A) = 07H$, $(R0) = F0H$, 执行指令:

ORL A, R0 ; $(A) \vee (R0) = 00000111B \vee 11110000B = 11110111B = F7H$

3. 逻辑异或指令 XRL

逻辑异或指令见表 3-18。

表 3-18 逻辑异或指令

指令助记符	操作说明	指令助记符	操作说明	指令助记符	操作说明
XRL A, Rn	$A \leftarrow (A) \oplus (Rn)$	XRL A, direct	$A \leftarrow (A) \oplus (\text{direct})$	XRL A, @ Ri	$A \leftarrow (A) \oplus ((Ri))$
XRL A, #data	$A \leftarrow (A) \oplus \text{data}$	XRL direct, A	$\text{direct} \leftarrow (\text{direct}) \oplus (A)$	XRL direct, #data	$\text{direct} \leftarrow (\text{direct}) \oplus \text{data}$

例 3-28 若 $(A) = 07H$, $(R0) = FCH$, 执行指令:

XRL A, R0 ; $(A) \oplus (R0) = 00000111B \oplus 11111100B = 11111011B = FBH$

在使用中，逻辑与用于实现对指定位清 0，其余位不变；逻辑或用于实现对指定位置 1，其余位不变；逻辑异或用于实现指定位取反，其余位不变。

例 3-29 根据要求写指令:

- 1) 对累加器 A 的 D1、D3、D5 位清 0，其余位不变: ANL A, #0D5H。
- 2) 对累加器 A 中的 D2、D4、D6 位置 1，其余位不变: ORL A, #54H。

3) 对累加器 A 中的 D0、D1 位取反, 其余位不变: `XRL A, #03H`。

4. 清零和求反指令

清零和求反指令见表 3-19。51 单片机只有清零指令 `CLR A` 和求反指令 `CPL A` 是字节指令, 其他清零和求反指令都是位操作指令。

表 3-19 清零和求反指令

指令助记符	操作说明	指令助记符	操作说明
<code>CLR A</code>	$(A) \leftarrow 00H$	<code>CPL A</code>	$A \leftarrow (\overline{A})$

例 3-30 对 R0 中的内容取反:

```
MOV    A,R0
CPL    A
MOV    R0,A
```

5. 循环移位指令

循环移位指令见表 3-20。

表 3-20 循环移位指令

指令助记符	操作说明	指令助记符	操作说明
<code>RL A</code>	循环左移, $D_{i+1} \leftarrow D_i, D_0 \leftarrow D_7$	<code>RR A</code>	循环右移, $D_{i+1} \rightarrow D_i, D_0 \rightarrow D_7$
<code>RLC A</code>	带进位的循环左移 $Cy \leftarrow D_7, D_{i+1} \leftarrow D_i, D_0 \leftarrow Cy$	<code>RRC A</code>	带进位的循环右移 $Cy \rightarrow D_7, D_{i+1} \rightarrow D_i, D_0 \rightarrow Cy$

4 条循环移位指令执行一次, 累加器 A 中的内容向左或向右移位一次, 循环移位指令执行一次的示意图如图 3-3 所示。

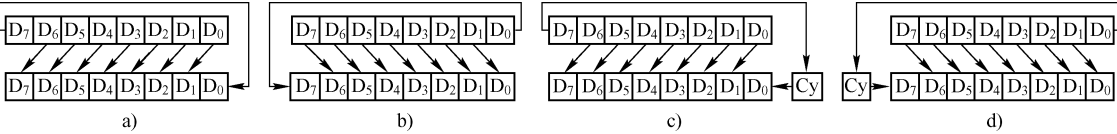


图 3-3 循环移位指令执行一次的结果示意图
a) RL A b) RR A c) RLC A d) RRC A

例 3-31 若累加器 A 中的内容为 99H, $Cy = 0$, 执行带进位的循环左移指令:

```
RLC    A           ;执行结果: (A) = 32H, Cy = 1。
```

3.3.4 控制转移类指令

控制转移类指令通常用于循环结构和分支结构程序, 共有 17 条指令, 包括无条件转移指令、条件转移指令、子程序调用和返回指令, 以及空操作指令。

1. 无条件转移指令

无条件转移指令见表 3-21。

表 3-21 无条件转移指令

指令助记符	操作说明	指令助记符	操作说明
<code>LJMP addr16</code>	$PC \leftarrow \text{addr16}$	<code>AJMP addr11</code>	$PC[D10 \sim D0] \leftarrow \text{addr11}$
<code>SJMP rel</code>	$PC \leftarrow PC + 2 + \text{rel}$	<code>JMP @A + DPTR</code>	$PC \leftarrow (A) + (DPTR)$

无条件转移指令是指当程序执行到该指令时，程序无条件地转移到指令所提供的地址处执行指令。这四条无条件转移指令不影响任何标志。

1) LJMP addr16 是长转移指令，是 64 KB 范围内的无条件跳转指令。该指令在执行时直接将该 16 位地址值送给程序计数器 PC，程序无条件地转移到 PC 指针所指向程序存储器新的目的地址单元取指令执行。

2) AJMP addr11 是绝对转移指令，是 2 KB 范围内的无条件跳转指令。该指令在执行时先将 PC 加 2，然后把指令的 11 位地址 addr11 送给 PC 的低 11 位，其高 5 位不变，执行后程序转移到 PC 指针所指向程序存储器新的目的地址单元取指令执行。

3) SJMP rel 是相对转移指令，操作数 rel 是 8 位带符号补码数，取值范围为 $-128 \sim +127$ ，所以该指令的转移范围是：相对 PC 当前值向前 128 字节，向后 127 字节。执行时先将程序计数器 PC 加 2，然后将 PC 的值与指令中的偏移量 rel 相加得到转移的目的地址。

4) JMP @A + DPTR 是间接转移指令，是 51 单片机中的唯一一条间接转移指令，转移的目的地址由数据指针 DPTR 中的内容与累加器 A 中的 8 位无符号数相加得到，可实现在 64 KB 范围内无条件跳转。指令执行过程中对 DPTR 和 A 中的内容不产生影响。

2. 条件转移指令

条件转移指令是指当条件满足时，程序转移到指定的目的地址；条件不满足时，程序将顺序执行。在 51 单片机中，条件转移指令有 3 种：对累加器 A 判 0 或判非 0 条件转移指令、比较转移指令，以及减 1 不为 0 转移指令。

1) 对累加器 A 判 0 和判非 0 条件转移指令，指令见表 3-22。

表 3-22 对累加器 A 判 0 和判非 0 条件转移指令

指令助记符	操作说明	指令助记符	操作说明
JZ rel	若 A = 0，则转移，否则，顺序执行	JNZ rel	若 A ≠ 0，则转移，否则，顺序执行

对累加器 A 判 0 和判非 0 条件转移指令不影响任何标志。

1) JZ rel 是判 0 转移指令，rel 是 8 位带符号补码数，取值范围为 $-128 \sim +127$ 的偏移量，执行 JZ rel 指令的操作过程：若 A = 0，则 $PC \leftarrow PC + 2 + rel$ ，否则， $PC \leftarrow PC + 2$ 。

2) JNZ rel 是判非 0 转移指令，rel 也是偏移量，执行 JNZ rel 的操作过程：若 A ≠ 0，则 $PC \leftarrow PC + 2 + rel$ ，否则， $PC \leftarrow PC + 2$ 。

2) 比较不相等转移指令，指令见表 3-23。

表 3-23 比较不相等转移指令

指令助记符	操作说明
CJNE A, direct, rel	若 (A) = (direct)，顺序执行，否则转移；如果 (A) > (direct)，Cy ← 0；否则，Cy ← 1
CJNE A, #data, rel	若 (A) = data，顺序执行，否则转移；如果 (A) > data，Cy ← 0；否则，Cy ← 1
CJNE Rn, #data, rel	若 (Rn) = data，顺序执行，否则转移；如果 (Rn) > data，Cy ← 0；否则，Cy ← 1
CJNE @Ri, #data, rel	若 ((Ri)) = data，顺序执行，否则转移；如果 ((Ri)) > data，Cy ← 0；否则，Cy ← 1

比较不相等转移指令满足转移条件时，执行 $PC \leftarrow PC + 3 + rel$ 操作，实现转移。指令中的 rel 是偏移量，取值范围为 $-128 \sim +127$ 。比较不相等转移指令影响标志 Cy，对其他标志无影响。

3) 减 1 不为 0 转移指令，指令见表 3-24。

表 3-24 减 1 不为 0 转移指令

指令助记符	操作说明
DJNZ Rn,rel	若 $(Rn) - 1 = 0$ ，则顺序执行；如果 $(Rn) - 1 \neq 0$ ，则转移
DJNZ direct,rel	若 $(direct) - 1 = 0$ ，则顺序执行；如果 $(direct) - 1 \neq 0$ ，则转移

这两条减 1 不为 0 转移指令满足转移条件时，执行 $PC \leftarrow PC + 2 + rel$ 的操作，实现转移。指令中的 rel 是偏移量，取值范围为 $-128 \sim +127$ 。这两条减 1 不为 0 转移指令不影响任何标志。

编写循环程序时，通常应用 DJNZ 指令来构造循环程序结构，实现程序段的重复处理。

3. 子程序调用与返回指令

子程序调用与返回指令见表 3-25。

表 3-25 子程序调用与返回指令

指令助记符	操作说明
LCALL addr16	将该指令的下一条指令地址压入堆栈，然后 $PC \leftarrow addr16$ ，addr16 为子程序入口地址
ACALL addr11	将该指令的下一条指令地址压入堆栈，然后 $PC \leftarrow addr11$ ，PC 的高 5 位 (D15 ~ D11) 不变
RET	将执行 LCALL addr16 或 ACALL addr11 指令，压入堆栈的 16 位指令地址出栈 $\rightarrow PC$
RETI	将响应中断时压入堆栈的 16 位指令地址出栈 $\rightarrow PC$ ，清内部中断优先级标志

这四条子程序调用与返回指令不影响任何标志。

主程序对子程序的一次完整调用，包括主程序调用子程序指令，子程序的末尾必须有一条返回主程序的指令 RET。

1) LCALL addr16 是长调用指令，无条件地调用位于指定地址的子程序，由于指令直接给出了 16 位子程序入口地址，能够调用 64 KB 范围内的任何子程序。

2) ACALL addr11 是绝对调用指令，无条件地调用位于指定地址的子程序，由于指令给出的是 11 位地址，只能实现调用 2 KB 范围内的任何子程序。

3) RET 是子程序返回指令，当子程序执行到该指令时，表示结束子程序的执行，返回主程序断点处的指令继续执行。

4) RETI 是中断返回指令，中断服务程序的最后一条指令。该指令和 RET 指令的功能相似，但执行 RETI 时，首先清内部中断状态寄存器的中断优先级的标志。

4. 空操作指令

空操作指令如下：

NOP

该指令经取指、译码后不进行任何操作（空操作），占用一个机器周期的时间。

3.3.5 位操作类指令

51 单片机的一个显著的特点就是有一个布尔处理机，具有位单元的操作功能。位操作指令以进位标志位 Cy 作为位累加器，位地址可用以下方式表示：

1) 直接用位地址表示方式，如 20H、D4H。

- 2) 采用字节地址加位的方式，两者之间用“.”隔开，如 20H.0。
- 3) 采用寄存器名称加位的方式，如 ACC.0、PSW.7、P2.7。
- 4) 位名称方式，如 EA、TR0。

位操作指令共有 17 条，见表 3-26，主要分为 4 类，即位传送指令、位状态控制指令、位逻辑运算指令和位条件转移指令。

表 3-26 位操作类指令

指令助记符	操 作 说 明	指令助记符	操 作 说 明
MOV C,bit	位传送: $Cy \leftarrow bit$	MOV bit,C	位传送: $bit \leftarrow Cy$
CLR C	位累加器 Cy 清 0: $Cy \leftarrow 0$	CLR bit	位单元清 0: $bit \leftarrow 0$
SETB C	位累加器 Cy 置 1: $Cy \leftarrow 1$	SETB bit	位单元置 1: $bit \leftarrow 1$
ANL C,bit	位逻辑与: $Cy \leftarrow Cy \wedge bit$	ANL C,/bit	位逻辑与: $Cy \leftarrow Cy \wedge \overline{bit}$
ORL C,bit	位逻辑或: $Cy \leftarrow Cy \vee bit$	ORL C,/bit	位逻辑或: $Cy \leftarrow Cy \vee \overline{bit}$
CPL C	位累加器 Cy 取反: $Cy \leftarrow \overline{Cy}$	CPL bit	位单元取反: $bit \leftarrow \overline{bit}$
JC rel	若 $Cy = 1$ ，则转移；否则，顺序执行	JNC rel	若 $Cy = 0$ ，则转移，否则，顺序执行
JB bit,rel	若 $bit = 1$ ，则转移；否则，顺序执行	JNB bit,rel	若 $bit = 0$ ，则转移，否则，顺序执行
JBC bit,rel	若 $bit = 1$ ，则转移，并执行 $bit \leftarrow 0$ ；否则，顺序执行		

习题

3-1 51 单片机的指令系统包括多少条指令？按字节划分为几种？按执行时间如何划分？按功能如何划分？

3-2 列举 51 单片机的几种寻址方式。

3-3 列举访问外部数据存储器和其他扩展部件的几种方法。

3-4 列举访问程序存储器的几种指令，并理解各个指令的用法。

3-5 简述如何判断字节指令和位操作指令。

3-6 在下列指令中，判断哪些是字节操作指令，哪些是 16 位操作指令，哪些是位操作指令？

- (1) CLR A (2) CLR 20H (3) JNB 7FH (4) ANL A, Rn
 (5) CPL A (6) MOV 4DH, C (7) ORL C, 6EH (8) ANL C, /55H
 (9) JNC rel (10) CPL 50H (11) JZ rel (12) MOV DPTR,#2010H

3-7 在下列指令中，判断哪些是合法指令，哪些是不合法的指令？

- (1) MOV C,R1 (2) JBC 2FH,rel (3) MOV DPTR,#05H
 (4) CJNE R7,7FH,rel (5) MOV R1,R7 (6) ANL 20H,#0FEH
 (7) ADD A,56H (8) MOV 20H,@DPTR (9) MOV @DPTR,P1
 (10) ORL 35H,A (11) SETB R0 (12) DJNZ @R0,rel
 (13) DEC DPTR (14) INC DPTR (15) MOVX B,@R0

第4章 汇编语言程序设计

第3章介绍了51单片机的指令系统，指令也就是汇编语言的命令语句。通过汇编命令语句能够编写出执行效率最高的程序。本章介绍汇编语言的相关概念、汇编语言程序的基本结构形式以及编写汇编语言程序的基本方法。由于汇编语言是面向机器硬件的语言，掌握汇编语言能够更好地理解单片机的内部结构和工作原理。

4.1 单片机汇编语言程序设计概述

4.1.1 机器语言、汇编语言和C语言

计算机所用的程序设计语言可以分为三类：机器语言、汇编语言和高级语言。

1. 机器语言

机器语言，也称为机器码，是一种由二进制码“0”和“1”组成的、能够被计算机直接识别和执行的语言。用机器语言表示的程序，又称目标程序。

单片机程序设计中，不论采用高级语言还是汇编语言编写，最终下载到单片机芯片的Hex文件或Bin文件都是二进制机器码。由于二进制代码较为冗长，通常用一位十六进制数来表示四位二进制数，即应用十六进制表示机器码。例如，51单片机控制P1口实现流水灯效果的某Hex文件，用记事本打开该文件，可以看到如下几行机器码：

从图4-1可以看出，机器码用十六进制数表示，但如果不通过指令表进行对照，难以看出机器码与指令对应的关系，编写、修改和维护困难，容易出错，难以发现和排除错误。

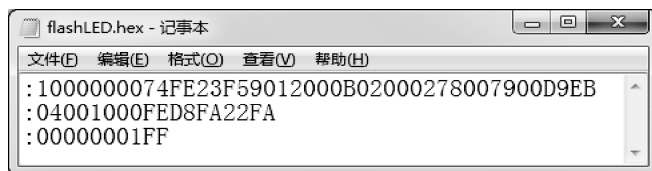


图4-1 51控制P1口实现流水灯的机器码

2. 汇编语言

为了解决机器语言难以编程和阅读的难题，使用汇编语言来表示机器语言。应用汇编语言编写程序的每一条指令与机器码一一对应。例如，图4-1的流水灯机器码的汇编程序为：

MOV	A, #0FEH	;对累加器A赋值, FEH = 1111 1110B, 最低位LED亮
START: RL	A	;自定义代码段START, 对A进行不带进位Cy的左循环移位
MOV	P1, A	;将A的值送入P1口, 点亮某个引脚的LED
LCALL	DELAY	;调用延时子程序
LJMP	START	;跳转回START段, 实现流水灯循环亮灭的效果
DELAY: MOV	R0, #00H	;自定义延时代码段, 对寄存器R0赋值0, 用于自减延时

```

DELAY1: MOV    R1, #00H        ;R1 赋值 0,用于内循环自减延时
          DJNZ   R1, $          ;(R1) - 1 → R1, (R1) ≠ 0 原地跳转,循环 256 次
          DJNZ   R0, DELAY1     ;(R0) - 1 → R0, (R0) ≠ 0 转移到 DELAY1,循环 256 次
          RET                    ;返回主程序
          END                    ;程序结束

```

从该程序可以看出，用助记符表示的汇编语言比机器码容易学习和使用。自 MCS-51 单片机诞生之日起，汇编语言就伴随着其他发展并获得了广泛应用。

由于单片机不能直接识别和执行汇编语言编写的程序，需要将汇编语言源程序转换（翻译）为二进制的机器码（目标程序），这一转换过程称为“汇编”，而用于转换的工具称为汇编程序，如图 4-2 所示。其中，源文件是汇编语言编写的文件，默认后缀名为 ASM；汇编程序为 Keil、IAR 等专门的编译工具，也称为编译器；目标程序是最终烧写到单片机芯片的 Hex 或 Bin 文件。

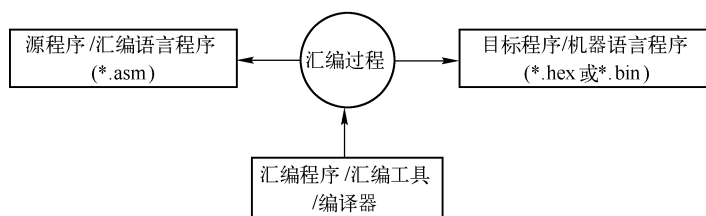


图 4-2 源程序、汇编程序和目标程序的关系

3. C51 语言

汇编语言是直接面向机器的语言，学习和应用汇编语言便于理解和熟悉微处理器的内部结构、存储区域的特点和特殊功能寄存器的功能、指令执行过程的基本工作原理及微处理器应用系统电路设计等。但也存在开发周期长、设计成本高、不易于移植等不足。人们将高级语言引入微处理器领域，其语言编程形式接近人类的自然语言和数学公式，易于移植。目前，各类系列的单片机基本上都采用 C 语言为主要的开发语言，例如 51 单片机的 C51、MSP430 的 C430 等。

C51 作为高级语言，易学易用，成熟稳定，性能已经接近汇编语言，优化后的机器代码长度也能控制在汇编语言的 1.2 ~ 1.5 倍，网上的各种资源极大丰富，获得了市场和单片机爱好者的广泛认可，成为单片机应用系统设计的首选语言。

但 C51 也并不能完全替代汇编语言。在对时间和空间要求较高的场合，汇编语言仍然是不可或缺的。例如，在典型的冒泡排序算法中，C51 需要 1033μs，而汇编语言只需要 376μs，性能的差异依然不小。不少工程师为了保持单片机系统的最佳性能，采用 C51 与汇编混合编程的方法进行程序设计。

另外，即使是高级语言编写的源程序，经过程的编译链接，产生目标代码的 Hex 文件，对编译的 Hex 文件进行反编译后仍然是以汇编程序的形式出现的。以上面的流水灯的 Hex 文件反编译为例，其反编译结果如下：

地址标号	汇编程序	指令地址	指令代码
Q0000:	MOV A, #0FEH	;0000	74 FE

Q0002:	RL	A	;0002	23
	MOV	P1,A	;0003	F5 90
	LCALL	Q000B	;0005	12 00 0B
	LJMP	Q0002	;0008	02 00 02
Q000B:	MOV	R0,#00H	;000B	78 00
Q000D:	MOV	R1,#00H	;000D	79 00
	DJNZ	R1,\$	;000F	D9 FE
	DJNZ	R0,Q000D	;0011	D8 FA
	RET		;0013	22

其中左边第一列为地址标号，第二列是 51 单片机反编译器获得的汇编程序，第三列为指令地址，第四列为指令代码，即机器码（对应图 4-1）。由此可见，要真正掌握单片机技术，学习汇编程序是非常必要的。

C51 的介绍和应用请参阅第 13 章 51 单片机的 C51 程序设计的内容。

4.1.2 汇编语言的特点

汇编语言具有如下一些特点：

- 1) 与机器语言相比，汇编语言接近于书写和说话的形式，更容易阅读和理解。但是与 C51 相比，汇编语言的学习和编程、调试的难度要大，因此汇编语言程序多用于编写 4 KB 以下、运算不太复杂的小程序。
- 2) MOV、CJNE 等指令助记符与机器指令一一对应。因此，程序占用的存储空间小，运行效率高。只有通过汇编语言，才能写出性能最优的程序。
- 3) 面向硬件，能够直接管理和控制硬件设备，例如对 I/O 接口、存储器的访问等。
- 4) 与单片机内部结构紧密结合，例如累加器 A、寄存器 R0 等。只有掌握汇编语言，才能更好地理解单片机的内部结构和工作原理。
- 5) 依赖于机器硬件，缺乏通用性，程序难以移植。
- 6) 高级语言一般用十进制编写，汇编语言多用十六进制来表示二进制数。

4.2 伪指令

第 3 章介绍的每条指令分别对应单片机能够执行的一个机器码。还有一类指令没有对应的机器码，不生成目标程序，也不影响程序的执行，仅用于对汇编过程进行某种控制或操作，这类指令称为伪指令。常用的伪指令有 8 条，下面分别进行介绍。

1. ORG

ORG 用于定义起始地址，也就是紧随其后的程序段或数据块将被设置在所规定的起始地址。ORG 为 origin（源点）的缩写。其格式为：

ORG addr16

其中，addr16 表示程序存储器中的一个 16 位地址，一般是一个十六进制数，或者是定义好的标号。一个源程序中可以多次使用 ORG 指令，以表示不同程序段的起始位置。但是所规定的地址应该按照顺序从小到大，不能交叉或重叠。

```

例 4-1  ORG      1000H

        START:MOV    A,20H
        ...
        ORG      1500H
        NEXT: MOV    R0,#30H

```

当使用编译器汇编完这段程序后，就会自动把“MOV A, 20H”这条指令（双字节指令）放置在程序存储器中的地址为 1000H 和 1001H 单元中。START 这个标号所表示的地址就是 1000H，而 NEXT 所在行的指令首地址是 1500H。

2. END

END 用于表示程序结束。整个源程序中只能有一条 END 命令，且应该位于程序的最后。如果 END 出现在源程序的中间，则后面的程序将会被编译器省略。其格式为：

```
END
```

3. EQU

EQU 用于赋值，equal（等于）的缩写，与 C 语言的#define 类似，相当于“=”。其格式为：

```
常量名 EQU 操作数/符号
```

其中，操作数可以是 8 位、16 位二进制数，也可以用十进制、十六进制表示。符号为其他的汇编符号。

```

例 4-2  X EQU 50           ;X 等于 50,十进制
        Y EQU 1000H        ;Y 等于 1000H,十六进制
        BUTTON2 EQU P1.0   ;BUTTON2 等价于 P1.0

```

编译器进行汇编时，程序中所有的常量 X 就具有 50 的值，Y、BUTTON2 同理。

4. DATA

DATA 用于赋值。其格式为：

```
变量名 DATA 操作数/表达式
```

```
例 4-3  X DATA 2000H      ;X 等于 2000H
```

在编译器进行汇编的时候，程序中所有的变量 X 都用 2000H 代替。DATA 的功能与 EQU 类似，但也存在以下区别：

1) EQU 必须先定义才能使用，而 DATA 不受限制。也就是说 EQU 定义的是常量，DATA 定义变量。

2) EQU 可以把一个汇编符号赋给一个字符名，而 DATA 不能。

3) DATA 可以将一个表达式的值赋给变量，而 EQU 不能。

4) DATA 通常用于定义数据地址。

5. DB

DB 用于从指定的地址单元开始，定义若干单字节内存单元的内容。其格式为：

```
[标号:] DB 单字节操作数
```

其中，单字节操作数可以是 8 位二进制、十进制和十六进制数值，或 ASCII 码。


```

例 4-4  ORG    2000H
         DB     3,41H,0F7H,'B'
         DB     "123Ac"

```

编译器进行汇编后，存储单元的内容分别为：

```

(2000H) = 3                (2001H) = 41H                (2002H) = F7H
(2003H) = 42H ('B')        (2004H) = 31H ('1')          (2005H) = 32H ('2')
(2006H) = 33H ('3')        (2007H) = 41H ('A')          (2008H) = 63H ('c')

```

6. DW

DW 用于从指定的地址单元开始，定义若干双字节存储单元的内容。其格式为：

[标号:] DW 双字节操作数

双字节操作数是 16 位二进制、十进制和十六进制数值，或 ASCII 码，其中高 8 位存入低地址字节，低 8 位存入高地址字节。如果不足 16 位，则高位用 0 填充。

```

例 4-5  ORG 2000H
         TAB: DW 1234H,100,69H

```

编译器进行汇编后，存储单元的内容分别为：

```

(2000H) = 12H              (2001H) = 34H              (2002H) = 00H
(2003H) = 64H (100)        (2004H) = 00H              (2005H) = 69H

```

7. DS

DS 用于从指定的地址单元开始，保留若干字节的存储单元作为备用。其格式为：

[标号:] DS 操作数

```

例 4-6  ORG    0050H
         DS     06H
         DB     5AH

```

编译器进行汇编后，从 0050H 开始保留 6 个字节的单元给程序备用，也就是占用了 0050H ~ 0055H。而 5AH 存放在 0056H 存储单元。

要注意的是，DB、DW 和 DS 伪指令都只对程序存储器起作用，不能对数据存储器的内容进行赋值或初始化。

8. BIT

BIT 用于定义位常量，与 EQU 类似。有些编译器不具备 BIT 命令时，可以用 EQU 来代替 BIT。其格式为：

常量名 BIT 操作数/符号

```

例 4-7  flag BIT 20H
         key  BIT P1.1

```

编译器进行汇编后，位地址 20H 赋值给 flag，key 作为 P1.1 使用。

4.3 汇编语言程序的基本结构形式

与高级语言一样，汇编语言程序设计中广泛应用结构化的程序设计方法，采用结构化设

设计的程序。按照结构化程序设计的方法，汇编语言程序一般有顺序、分支和循环等几种结构。三种结构的程序分别称为顺序程序、分支程序和循环程序，如图 4-3 所示。

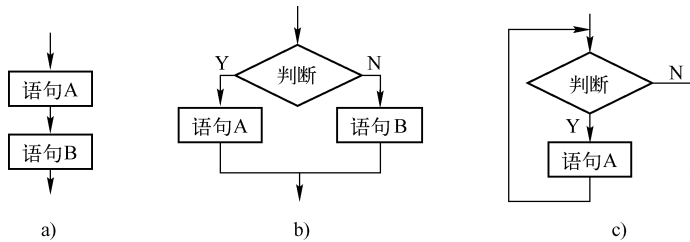


图 4-3 顺序程序、分支程序和循环程序
a) 顺序程序 b) 分支程序 c) 循环程序

4.3.1 顺序程序

顺序程序是指按照指令的顺序逐条执行指令，也称为简单程序或直线程序。其特点是按指令的排列顺序逐条执行，直到全部指令执行完毕。顺序程序主要使用数据传送指令和数据运算类指令。

例 4-8 对寄存器 R1、R2、R3 中存储的数据进行相加，结果存储到 R4。

MOV	A, R1	;R1 的数据复制到累加器 A
ADD	A, R2	;A 与 R2 的数据相加,结果存储在 A
ADDC	A, R3	;A 与 R3 的数据进行带进位相加,结果存储在 A
MOV	R4, A	;将 A 的数据复制到 R4

4.3.2 分支程序

根据不同条件转向不同的处理程序来改变程序执行的顺序，这种结构的程序称为分支程序。分支程序一般是利用条件转移指令来进行判断，通过转移指令的组合，可以形成单分支、双分支和多路分支形式。常用的转移指令有 JZ、JNZ、CJNE、JC、JNC、JB、JNB、JBC 和 DJNZ，多用于正负数判断、位单元状态判断、进借位判断、是否为“0”的判断及大小判断等。

4.3.3 循环程序

在一个程序段要重复多次运行的情况下，可以采用循环方法设计程序。循环程序主要有 5 个部分：

- 1) 初始化部分：设置循环初值，包括循环变量计数、数据指针初值，为循环做准备。
- 2) 循环处理部分：重复执行的程序段，是循环程序的主体，称为循环体。循环体可以是单条指令，也可以是复杂的程序段。
- 3) 循环修改：每循环一次，就要修改循环次数、数据及地址指针等。
- 4) 循环控制部分：控制循环的次数，为进行下一次循环修改计数器和指针的值，并检查循环次数是否达到退出循环的条件。
- 5) 循环结果处理：分析和存储结果。

循环可以是单重循环和多重循环。如果在循环程序的循环体中不再包含循环程序，即为

单重循环；如果在循环体中还包含有循环程序，这种现象就称为循环嵌套，这样的程序就称为多重循环程序。在多重循环中，内外循环不得交叉，也不允许从循环程序的外部跳入循环程序的内部。

通常对循环初始化只执行一次，而对循环处理、循环控制可以执行多次。循环结构可以分为图 4-4 所示的两种形式：计数控制和条件控制。

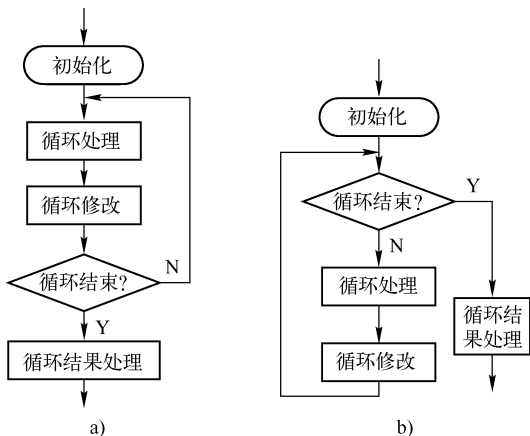


图 4-4 计数控制结构和条件控制结构

a) 计数控制结构 b) 条件控制结构

4.3.4 位操作程序

C、C++、Java、C#和 VB 等主流高级语言主要面向 8 位或 8 位以上的数据处理，而检测和控制应用系统中常常需要对 1 位信息进行处理，51 单片机提供了用于位操作的指令，位操作的指令主要有 MOV（传送）、CLR（清 0）、SETB（置 1）、ANL（与）、ORL（或）、CPL（取反）和位单元状态的判断控制指令等。

例 4-9 单片机应用系统上电和复位后常常要求测试单片机外部扩展各部件是否能够正常工作。编写程序，测试外部扩展 RAM 的地址为 4000H 的存储单元是否损坏。

```

MOV    DPTR,#4000H    ;设置要测试的外部 RAM 的地址
MOV     A,#55H         ;55H 作为测试数据
MOV     B,A            ;(B)=55H,保存测试数据
MOVX    @DPTR,A        ;55H 写入外部 RAM 的 4000H 存储单元中
MOVX    A,@DPTR        ;写入数据 55H 后,再读回数据→A
XRL     A,B            ;A、B 各位进行异或,比较写入和读回的数据是否一致
JNZ     BAD            ;如果(A)≠(B),RAM 可能已损坏,跳转到 BAD 程序段

```

位操作可以直接对 P0、P1、P2 和 P3 等端口的每个引脚进行单独控制，例如 SETB P1.0 将引脚 P1.0 置高电平，而 CPL P1.0 可以将 P1.0 输出的电平取反。

位操作能够对内部 RAM 的 20H~2FH 字节单元中的每个位单元，以及对特殊功能寄存器中能够进行位寻址的位单元进行单独控制。

例 4-10 CLR 20H.3 ;对 20H.3 (位地址为 03H)清零

MOV	C,40H	;将位地址为 40H 单元的内容送入进位位 Cy
SETB	RS0	
CLR	RS1	;选择当前寄存器组为第一组(参见表 2-2)

4.4 汇编语言程序设计举例

汇编语言程序设计的一般步骤有：任务分析、确定算法、算法优化、设计程序流程图、分配字节单元和位单元及有关 I/O 端口、扩展部件的端口地址、编写程序并加上注释、仿真调试及实物调试。

在使用汇编语言进行程序设计时，要注意 3 个问题：模块化设计、循环程序及子程序。模块化设计就是将一个复杂的任务划分为若干个模块（程序段），以提高程序的可读性、可靠性，使得程序层次更为清晰；循环程序能够大大减少对存储空间的占用；子程序可以增加程序段的可重用性，简化程序结构。

4.4.1 顺序程序设计

顺序结构程序是一种最简单、最基本的程序，无分支，按照程序编写的顺序依次执行。

例 4-11 编写程序，将外部 RAM 的 8000H 单元的内容传输到 7FFFH 单元。

分析：51 单片机外部扩展的存储器或部件与单片机内部寄存器或 RAM 之间的数据传输，必须通过累加器 A 来实现，外部扩展的存储器的存储单元或部件相互之间的数据传输也是如此。编程时，需要注意对 16 位地址指针 DPTR 的正确设置。

ORG	0000H	
LJMP	MAIN	;跳转至主程序执行,避开中断服务程序入口地址
ORG	0030H	
MAIN: MOV	DPTR,#8000H	;16 位地址值 8000H→DPTR
MOVX	A,@DPTR	;存储单元(8000H)→A
MOV	DPTR,#7FFFH	;修改 16 位地址指针 DPTR 指向 7FFFH 存储单元
MOVX	@DPTR,A	;(A)→(DPTR),A 中的内容传输给 7FFFH 单元
SJMP	\$	
END		

例 4-12 设某一变量的数据保存在片内 RAM 的 20H 单元中，取值范围为 00H ~ 09H，编写查表程序，查出变量的平方值，并存入片内 RAM 的 21H 单元。

查表就是根据自变量 x，在表格中找 y，使 $y = f(x)$ 。x 一般为连续变化的整数，将对应的 y 放置在一个线性表中，在需要的时候通过查表就能快速获取结果。查表方法广泛应用于显示、打印和数据转换，能够极大地加快程序执行的速度。

ORG	0000H	
LJMP	MAIN	
ORG	0030H	
MAIN: MOV	DPTR,#TAB	;平方值表的起始地址值→DPTR
MOV	A,20H	;需要求平方的值(20H)→A

```

MOV  A,@ A + DPTR      ;(A) + (DPTR)组成平方值单元地址,单元内容→A
MOV  21H,A             ;查表得到的平方值→21H
SJMP $                 ;无限循环
TAB: DB 0,1,4,9,16,25,36,49,64,81 ;00H~09H 的十进制数平方值表
END

```

例 4-12 的程序中，数据指针 DPTR 的地址值为 TAB，即表格中存储数据 0 的单元地址值，指令 SJMP \$ 是一条空循环指令，单片机未作任何实质性工作，处于等待状态，该指令在编写有中断服务程序的主程序时是一条经常被使用的指令。

4.4.2 分支程序设计

分支程序的结构有两种：双分支结构和多分支结构（散转分支结构），如图 4-5 所示。

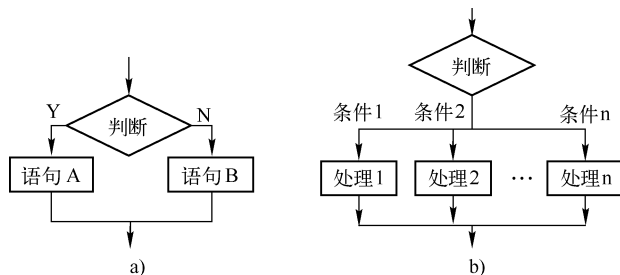


图 4-5 分支程序结构
a) 双分支结构 b) 多分支结构

例 4-13 设变量 x 以无符号数的形式存放在内部 RAM 的 30H

单元中，变量 y 与 x 有如下关系：
$$y = \begin{cases} 0FH, & x > 0AH \\ 0AH, & x = 0AH \\ 01H, & x < 0AH \end{cases}$$

试编写程序，根据 x 的取值求出 y，并将其存入内部 RAM 的 31H 单元。

程序流程图如图 4-6 所示，根据程序流程图编程如下：

```

ORG    0000H
LJMP   MAIN
ORG    0030H
MAIN:  MOV    A,30H      ;变量 x 的值存储在 30H 中
       CJNE   A,#0AH,POSITIVE ;x = (A) = 0AH 顺序执行,否则转移
                                   ;如果(A) > 0AH,Cy←0;如果(A) < 0AH,Cy←1

       SJMP    RESULT
POSITIVE: JC      XIAOYU    ;若 Cy 的状态为 0 顺序执行;否则转移
       MOV     A,#0FH      ;因为(A) = x > 0AH,则 0FH→A
       SJMP    RESULT
XIAOYU: MOV     A,#01H     ;因为(A) = x < 0AH,则 01H→A

```

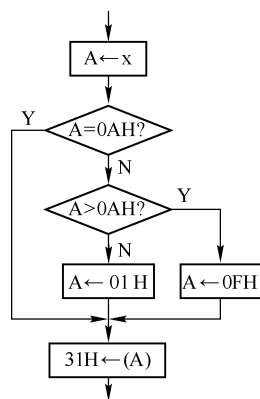


图 4-6 例 4-13 程序流程图

```

RESULT:  MOV    31H,A      ;31H←(A)
          SJMP   $
          END

```

注意例 4-13 程序中有两条 SJMP RESULT 指令，如果没有这两条指令，则无论 x 的值如何，31H 中内容都会是 0AH，所以程序中的两条 SJMP RESULT 指令缺一不可。

例 4-13 是用条件转移指令实现分支，下面介绍利用间接转移指令 JMP 来实现多分支程序转移（称为散转程序），通常有两种设计方法：

- 1) 查转移地址表，用转移地址构成散转表，将表中的内容作为转移的目标地址。
- 2) 查转移指令表，用转移到不同程序的转移指令构成散转表，判断条件后，转到表中指令执行。

下面用例 4-14 和例 4-15 两个例子说明，其中例 4-14 是利用转移地址表实现程序的转移，例 4-15 是利用转移指令表实现程序的转移。

例 4-14 试编写根据 R7 的内容转向对应的入口地址的程序，R7 的内容为 0 ~ n (n < 127)，处理程序的入口地址为 PR0 ~ PRn。

分析：将 PR0 ~ PRn 的入口地址值列在一个表格中，每一个地址为 16 位地址值，占两个单元。假设 PR0 ~ PRn 入口地址表格的首地址为 TAB，则 PR0 的高 8 位地址值在表格的首地址中，低 8 位地址值在首地址 TAB + 1 的存储单元中；而 PRx 在表中的偏移量为 2x，因此将 R7 的内容乘以 2 即得到 PRx 在表中的偏移地址，从偏移地址 2x 和 2x + 1 两个单元分别取出 PRx 的高 8 位地址和低 8 位地址送至 DPTR 寄存器，将 A 清 0，用 JMP @ A + DPTR 指令即转移到 PRx 入口地址。

```

PR0      EQU    0110H
PR1      EQU    0220H
PR2      EQU    0330H
...                               ;定义 PR0 ~ PRn 的入口地址
ORG      0000H
MOV      A,R7                     ;R7 的内容为 0 ~ n(n < 127)
ADD      A,R7                     ;A←(A) × 2( PRx 的高 8 位偏移地址 2x)
MOV      DPTR,#TAB                ;DPTR 指向表格的首地址
PUSH     ACC                      ;累加器 A 的内容压栈,保存 2x 值
MOVC     A,@ A + DPTR             ;取地址表中 PRx 的高 8 位地址
MOV      B,A;                     PRx 的高 8 位地址暂存于 B
POP      ACC                      ;2x 值出栈→A
INC      A                        ;累加器 A 的内容为低 8 位偏移地址 2x + 1
MOVC     A,@ A + DPTR             ;取地址表中 PRx 的低 8 位地址
MOV      DPL,A                    ;PRx 的低 8 位地址值→DPL
MOV      DPH,B                    ;PRx 的高 8 位地址值→DPH,DPTR 指向 PRx 入口地址
CLR      A
JMP      @ A + DPTR               ;转移到 DPTR 指定的 PRx 地址单元执行程序
TAB:     DW      PR1,PR1,PR2,...,PRx
          END

```

例 4-15 要求同例 4-14，试用转移指令表实现转移。

```

PR0      EQU    110H
PR1      EQU    0220H
PR2      EQU    0330H
...
ORG      0000H
LJMP     MAIN
ORG      0030H
MAIN: MOV   DPTR,#TAB          ;转移表首地址送数据指针 DPTR
      MOV   A,R7               ;分支序号→A
      ADD   A,R7               ;分支序号 * 2
      JMP   @ A + DPTR        ;跳转在转移表
TAB:  AJMP  PR0                ;转移指令组成的转移表,(DPTR) = TAB,(A) = 0
      AJMP  PR1                ;(DPTR) = TAB,(A) = 2
      ...
      AJMP  PRx                ;(DPTR) = TAB,(A) = 2 × x
      ...
      AJMP  PRn                ;(DPTR) = TAB,(A) = 2 × n
END

```

4.4.3 循环程序设计

软件延时是最常见的循环程序的应用。

例 4-16 设计一个延时 200 μ s 的子程序，已知单片机使用的晶振为 12 MHz，1 个机器周期为 1 μ s。

```

DELAY200us: MOV   R1,#99      ;子程序开始,初始化设置循环 99 次,指令执行 1  $\mu$ s
              NOP             ;空操作,1  $\mu$ s
              DJNZ  R1,$       ;R1←(R1) - 1,判断(R1) = 0? 不为 0 原地跳转,2  $\mu$ s
              RET             ;如果 R1 = 0 则循环结束,返回主程序,2  $\mu$ s

```

延时程序所消耗的时间： $1 + 1 + 99 \times 2 + 2 = 200\mu\text{s}$ 。

例 4-17 设有一个六位十进制压缩 BCD 码数存放在 32H ~ 30H 中，十万位和万位数存放在 32H 的高四位和低四位中，依次类推，十位数和个位数存放在 30H 中，试编程将六位十进制 BCD 码数据拆分存储在 45H ~ 40H 的低四位中，十万位 BCD 码数在 45H 的低四位，万位数在 44H 的低四位，依次类推，个位数在 40H 的低四位。

分析：3 个字节的压缩 BCD 码数拆分成 6 个字节，每个字节的拆分方法都是一样的，可以采用循环程序，假设首先从十位和个位数开始，则两个 R0、R1 数据指针初值：30H，40H，循环次数 R7：3。循环处理部分：((R0)) 的低四位 → (R1)，((R0)) 的高四位 → ((R1) + 1)。子程序取名：BCDDATA。

```

BCDDATA: MOV   R0,#30H        ;待拆分 BCD 码数的首地址:30H(十位和个位)
              MOV   R1,#40H    ;存放一位 BCD 码数的首地址:40H(个位 BCD 码)

```

	MOV	R7, #03H	;设置循环变量的循环次数
LOOP:	MOV	A, @ R0	;一个字节待拆分 BCD 码数→A
	ANL	A, #0FH	;屏蔽高四位,保留低四位二进制数不变
	MOV	@ R1, A	;存放一个 BCD 码数:个位、百位或万位 BCD 码
	INC	R1	;存放已拆分 BCD 码数的地址指针 +1
	MOV	A, @ R0	;再将该字节的待拆分 BCD 码数→A
	ANL	A, #0F0H	;屏蔽低四位,保留高四位二进制数不变
	SWAP	A	;将保留的 BCD 码数交换至 A 的低四位
	MOV	@ R1, A	;存放一个 BCD 码数:十位、千位或十万位 BCD 码
	INC	R0	;修改待拆分 BCD 码数的地址指针
	INC	R1	;修改已拆分 BCD 码数的存放地址的地址指针
	DJNZ	R7, LOOP	;R7←(R7) - 1, 如果 (R7) 不为 0, 转移到 LOOP
	RET		;如果 (R7) 为 0, 已循环执行 3 次, 子程序返回

4.4.4 子程序设计

在实际问题中, 有很多程序段需要在一个程序中进行多次操作处理, 如多字节的算术运算、代码转换及字符处理等。编程时, 把这样多次重复使用的程序段作为独立的程序段进行编写, 程序可以调用这些独立的程序段。这种能够完成一定功能、可以被其他程序调用的独立程序段称为子程序。调用子程序的程序称为主程序或调用程序。调用子程序的过程称为子程序调用。子程序执行完后返回主程序的过程称为子程序返回。子程序节省占用的存储单元, 使程序简短、清晰。善于灵活地使用子程序, 是程序设计中的重要技巧之一。

子程序是具有某种功能的独立程序段, 子程序的编写方法与一般程序基本相同, 但每个子程序有一个子程序名, 子程序的末尾有一条子程序返回指令 (RET), 调用子程序使用 ACALL 或 LCALL 子程序调用指令。在编写和调用子程序时要把握和注意以下几个问题:

- 1) 每个子程序赋一个该子程序的名字, 作为子程序入口的符号 (标号), 即子程序名作为子程序第一条指令 (语句) 的标号。
- 2) 明确入口条件和出口条件。入口条件表明子程序需要哪些参数, 这些参数存放在哪些寄存器或哪些存储单元; 出口条件则表明子程序处理的结果是如何存放的。只有正确理解并运用这两个条件, 才能完成子程序和主程序之间数据的正确传递。
- 3) 保护现场和恢复现场。若在调用该子程序之前, 主程序已经使用了某些存储单元或寄存器, 执行该子程序过程中, 这些寄存器或存储单元的内容被修改刷新, 需要先把这些被刷新的寄存器或存储单元中的内容压入堆栈 (使用压栈指令 PUSH) 进行保护, 执行完子程序后再从堆栈中弹出 (使用出栈指令 POP), 以便加以恢复。注意: 在 51 单片机中, 堆栈操作遵循“先入后出”的原则, 因此, 先压栈的参数后弹出, 才能保证恢复原来的数据。如果有较多的寄存器要保护, 应使主程序和子程序使用不同的寄存器组。
- 4) 在子程序中可包含对另外子程序的调用, 称为子程序嵌套。
- 5) 主程序中要正确地设置堆栈指针。

例 4-18 编写两个 16 位数据的加法程序, 子程序取名为 ADD16。

ADD16: PUSH ACC


```

MOV    A,@R0
CLR    C                ;执行低 8 位两数相加之前清进位位 C
ADD    A,@R1
MOV    @R0,A            ;低 8 位两数相加结果存储在 R0 指定的存储单元中
INC    R0
INC    R1
MOV    A,@R0
ADDC   A,@R1            ;执行带进位位的高 8 位两数相加
MOV    @R0,A            ;高 8 位两数相加结果存储在 R0 指定的存储单元中
POP    ACC
RET

```

该子程序的调用方式如下：

```

MOV    R0,#VALUE1      ;被加数 16 位数据首地址,相加结果存放的首地址
MOV    R1,#VALUE2      ;加数 16 位数据的首地址
LCALL  ADD16

```

应特别注意的是，若两个 16 位数据相加有进位时，其进位标志位 Cy 会改变为 1。

例 4-19 在工程应用中，需要将二进制数转换成为十进制 BCD 码再予以显示。编程将累加器 A 中二进制数转为 BCD 码，百位存入 43H、十位存入 42H、个位存入 41H。

分析：可以采用一个字节的二进制数除 100 取整得到百位数、余数除 10 取整得到十位数、除 10 得到的余数为个位数的方法，分别获取十进制的百位、十位和个位数。

```

B2BCD:MOV    B,#100      ;子程序取名为 B2BCD,B 赋值 100
DIV    AB                ;A/B,执行该除法指令,商存入 A,余数存入 B
MOV    43H,A            ;商是除 100 取整得到的百位数,存入 43H
MOV    A,B              ;(A)=(B),A 中的内容为除 100 的余数,即十位和个位数
MOV    B,#10
DIV    AB                ;A/B,十位数存入 A,个位数存入 B
MOV    42H,A            ;十位数存入 42H
MOV    41H,B            ;个位数存入 41H
RET                    ;返回主程序

```

B2BCD 子程序的调用方式如下：

```

MOV    A,#XSH           ;需要转为 BCD 码的二进制数→A
LCALL  B2BCD

```

例 4-20 编程实现将累加器 A 中的低 4 位二进制数 0~FH 转换为 ASCII 码。

常用的显示设备，例如字符液晶、计算机屏幕显示，以及打印机等都是采用 ASCII 码。如果直接将数字送进去是无法正确显示的。例如数字 8 送入显示装置，会显示“退格键”，因为退格键的 ASCII 码为 8。要正确显示 8，必须要将数字 8 转为 ASCII 的“8”。

分析：4 位二进制数可以分为 0~9 和 AH~FH 的两种情况，按照表 1-2 的 ASCII 码：

1) 0~9 的 ASCII 码为 30H~39H，30H~39H 恰好与 0~9 相差 30H，0~9 转换为 ASCII 码只需要加上 30H 即可。

2) AH~FH 的 ASCII 码为 41H~46H，41H~46H 与 AH~FH 相差 37H，AH~FH 转换

为 ASCII 码只需要加上 37H (十进制为 55)。

```

B2ASC:ANL    A,#0FH    ;子程序取名为 B2ASC,A 的高 4 位清零,保留低四位
        MOV     B,A      ;(B) = (A),暂存
        CLR     C        ;清借位位 Cy
        SUBB    A,#10    ;如果二进制数为 0~9,不够减,产生借位,借位位 Cy 为 1
        MOV     A,B      ;A = B,为后面加法做准备
        JC      NUM      ;Cy 为 1,说明是 0~9,跳转到 NUM 程序段
        ADD     A,#55    ;AH~FH,A←(A)+55(37H),转换成 ASCII 码字符“A”~“F”
        SJMP    EXIT     ;跳转退出
NUM:    ADD     A,#30H    ;0~9,A = A + 30H,转换成 ASCII 码字符“0”~“9”
        EXIT:  RET      ;退出子程序
    
```

调试汇编程序需要技巧。对于编写的例 4-20 的子程序,是否正确,如何调用呢?需要以下 3 个步骤:

- 1) 在 Keil 编写调用的程序,利用 LCALL 指令,分别将 7 和 A 两个字符作为参数调用子程序。编译完毕后生成 Hex 文件。
- 2) 在 Proteus 菜单中选择“Debug”→“8051 CPU Registers”,显示单片机的基本信息。
- 3) 在 Proteus 中运行,暂停,观察寄存器、累加器 A 的内容是否正确。

如图 4-7 所示,首先以 7 为参数调用子程序,运算的结果放在 R0 中,调试结果显示为 37H,正是 ASCII 字符“7”。然后以 A 为参数调用,放在累加器 ACC 中,41H 也与“A”吻合。

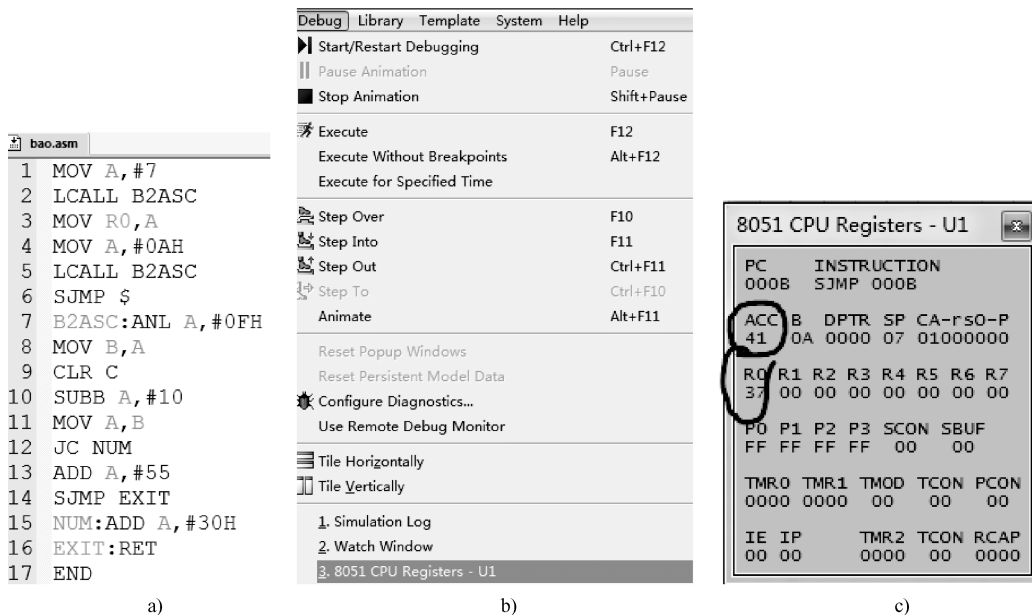


图 4-7 汇编语言程序设计的调试

a) Keil 程序设计 b) Proteus 显示 CPU 寄存器 c) A 和 R0 结果正确

习题

4-1 51 单片机的汇编语言有何特点?

4-2 对下面程序中各条指令做出注释，并分析程序运行的最后结果 A 等于多少？请使用 Keil 编译 Hex、Proteus 调试的方法解决。

```

MOV    R0,#40H
MOV    @R0,#97H
MOV    A,#68H
ANL    A,#17H
ORL    40H,A
XRL    A,@R0
ADD    A,R0
CPL    A
END

```

4-3 设 1 个机器周期为 $1\mu\text{s}$ ，请编写 1 ms 的延时程序并进行计算分析。

4-4 编程将 A 累加器的低四位数据送至 P1 口的高四位，P1 口的低四位保持不变。

4-5 编程将 R0 的内容和 R1 的内容互相交换。

4-6 编写一个软件延时 20 ms 的子程序，子程序取名为 DELAY20 ms。

4-7 设有一个四位十进制 BCD 码数据，千位数存放在 33H 的低四位，百位数在 32H 的低四位，十位数在 31H 的低四位，个位数在 30H 的低四位。试编程将四位十进制 BCD 码数据的千位数和百位数合并存储在 51 单片机片内的 41H 的高四位和低四位中，十位数和个位数合并存储在 40H 高四位和低四位中。

4-8 编程比较累加器 A 和 30H 中的内容，如果 $(A) = (30H)$ ，则设置位单元 20H 为“0”状态，否则设置位单元 20H 为“1”状态；如果 $(A) > (30H)$ ，设置位单元 21H 为“1”状态；如果 $(A) < (30H)$ ，设置位单元 21H 为“0”状态。

4-9 设 R0、R1、R2、R3 分别存储一个字节的无符号数，编写程序实现 $(R0 \times R1 + R2) / R3$ 。

4-10 请根据图 4-8 所示的电路，用汇编语言编写实现流水灯控制的程序。

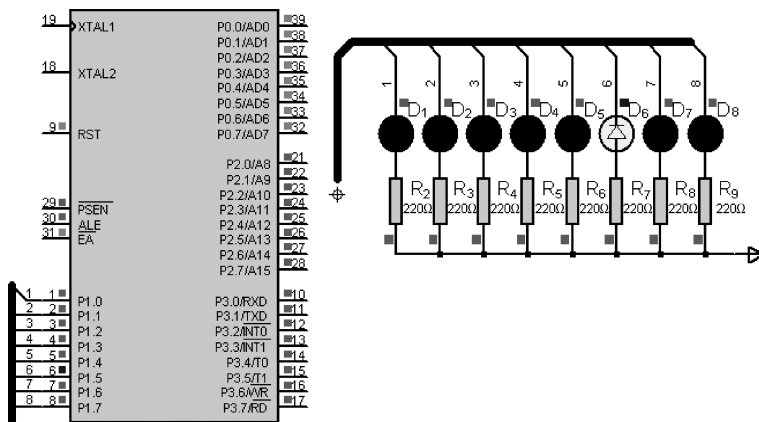


图 4-8 题 4-10 图

第5章 51 单片机定时器/计数器

在微处理器应用系统的设计过程中，针对特定事件的定时控制、脉冲个数的统计等情况经常遇到。要实现这些功能，可以应用微处理器内部的定时器/计数器予以实现。定时器/计数器的核心部件就是数字电子技术中的计数器（时序逻辑电路），定时器/计数器由计数器以及相应的附加控制单元电路组成。

5.1 51 单片机定时器/计数器的结构

51 单片机内部一般有 2~3 个可编程定时器/计数器，其结构基本相同。常规系列 80C51 单片机内部设有 2 个 16 位可编程定时器/计数器，分别是 T0 和 T1。其工作方式和工作状态均可以通过设置相应特殊功能寄存器相关位的状态进行控制。

5.1.1 定时器/计数器的核心原型

51 单片机定时器/计数器的主要功能部件的核心原型是 +1 计数器。51 单片机采用可编程的 +1 计数器，位数可选 8 位、13 位和 16 位。为理解其工作过程，现以简单的 3 位异步 +1 计数器来说明问题。

图 5-1 所示为数字电路的基本时序逻辑电路：二进制计数器。该二进制计数器为 3 位异步 +1 计数器。

触发器——JK 触发器。

CP——时钟脉冲输入信号。触发器作为计数器应用，也可称为计数脉冲输入信号。

JK 端悬空，相当于 $J = K = 1$ ，因此在每一个 CP 下降沿触发下 Q_0 的状态翻转一次。

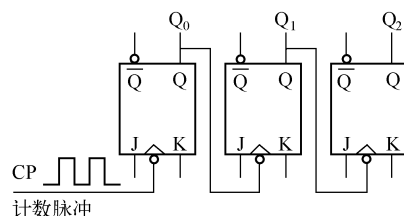


图 5-1 3 位异步 +1 计数器的组成

默认 JK 触发器输出的 Q_0 、 Q_1 、 Q_2 的初始状态为“000”时，在每一个计数脉冲 CP 下降沿触发下 Q_0 、 Q_1 、 Q_2 的输出值 +1，当 Q_0 、 Q_1 、 Q_2 的输出值为“111”时，在一个计数脉冲 CP 下降沿触发下， Q_0 、 Q_1 、 Q_2 的输出值由“111”变换为“000”。图 5-1 中的各 JK 触发器在计数脉冲 CP 的作用下， Q_0 、 Q_1 、 Q_2 输出端输出的波形如图 5-2 所示。

若 CP 计数脉冲输入信号是一个频率固定的时钟脉冲信号，假设其频率为 $f_{CP} = 1 \text{ Hz}$ ，则时钟脉冲周期 $T_{CP} = 1 \text{ s}$ 。当 JK 触发器输出端 Q_0 、 Q_1 、 Q_2 的状态变换由 $000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 110 \rightarrow 111 \rightarrow 000$ 时，总共需要经过 8 个计数脉冲，相当于定时了 8 s 时间。将 Q_0 、 Q_1 、 Q_2 的状态由 $111 \rightarrow 000$ 转换时定义为“溢出”，此时 $\overline{Q_2}$ 的状态由“0” $\rightarrow$ “1”。

假若是定时过程，需要定时 5 s，则设置 Q_0 、 Q_1 、 Q_2 的初始状态为“011”，当 Q_0 、 Q_1 、 Q_2 的状态变换由 $011 \rightarrow 100 \rightarrow 101 \rightarrow 110 \rightarrow 111 \rightarrow 000$ 时，恰好需要 5 个时钟脉冲周期，即定时时间为 5 s。初始状态“011”被称为定时初值，即定时计数从该定时初值开始累加计数。

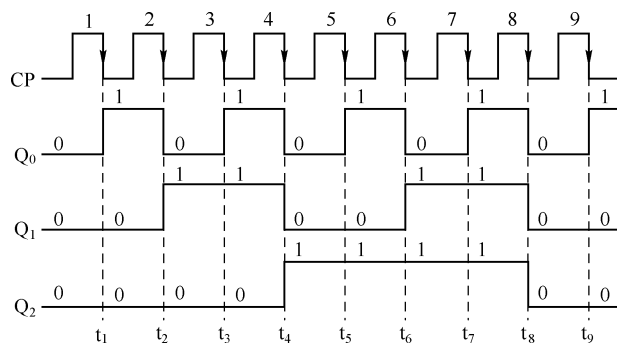


图 5-2 3 位异步 +1 计数器的时序图

假若是计数过程，CP 的输入脉冲不使用固定频率的时钟脉冲，而采用未知的计数脉冲（来源于单片机外部扩展部件或其他方式）。假设需要计数 3 次，则设置 Q_0 、 Q_1 、 Q_2 的计数初始状态为“101”，即计数初值为“101”，在 3 个计数脉冲下降沿的作用下， Q_0 、 Q_1 、 Q_2 的状态变换由 101→110→111→000，即计数次数为 3。

51 单片机内部的定时器/计数器则在类似于图 5-1 这种 +1 计数器的功能上增加了启动、溢出和中断等相应的硬件电路以及寄存器，以达到编程实现有效的控制和便利的操作。

5.1.2 定时器/计数器的组成

定时器/计数器的原理结构框图如图 5-3 所示。该框图给出了单片机内部定时器/计数器 T0、T1 的基本结构。

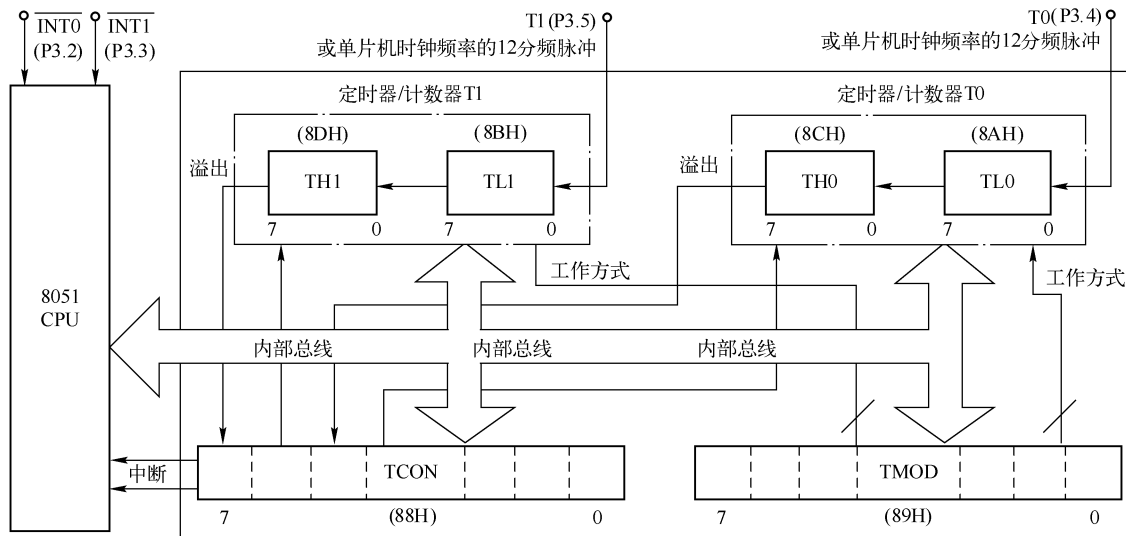


图 5-3 定时器/计数器结构框图

从图 5-3 可以看出：

1) 定时器/计数器独立于单片机内部的 CPU 进行计数操作，即计数过程与单片机执行程序的过程并行进行，互不影响。

- 2) 51 单片机内部有 T0、T1 两个定时器/计数器。
- 3) 每个定时器/计数器均具备独立的 +1 计数器 TH_x 和 TL_x (x 取 0 或 1)。
- 4) 两个定时器/计数器共同受 TMOD (工作方式寄存器) 和 TCON (控制寄存器) 相应位状态的控制。

5.1.3 定时器/计数器的工作方式寄存器 TMOD

定时器/计数器是一种可编程控制的部件，如何实现其工作方式的程序设定是重点内容。

两个定时器/计数器均具备两种基本功能：定时和计数，且具备 4 种工作方式：方式 0、方式 1、方式 2 和方式 3。

特殊功能寄存器 TMOD 就是用于设置定时器/计数器 T0 和 T1 工作方式的专用寄存器。寄存器 TMOD 的字节地址为 89H，不能位寻址。

TMOD 寄存器的定义如图 5-4 所示。

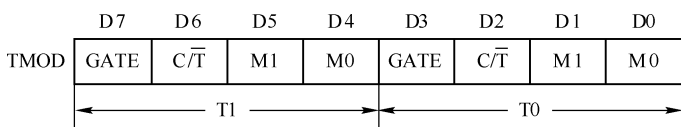


图 5-4 寄存器 TMOD 定义

图 5-4 中，工作方式寄存器 TMOD 的低 4 位设置 T0 的工作模式和工作方式，高 4 位设置 T1 的工作模式和工作方式，各位定义如下：

(1) GATE——门控位

GATE = 0，定时器/计数器的运行启动控制仅由 TR_x (x 取 0 或 1) 位通过置 1 实现。

GATE = 1，定时器/计数器的运行启动控制不但需要 TR_x (x 取 0 或 1) 位设置为“1”，还和外部引脚 $\overline{\text{INT}}_x$ (x 取 0 或 1) 的电平情况相关。只有 $\overline{\text{INT}}_x$ 引脚输入为高电平时，才能启动相应的定时器/计数器开始工作（具体工作原理参看 5.2 节的工作方式介绍）。

(2) C/T——工作模式选择位

C/T = 0，定时器/计数器的工作模式为定时器，固定的定时时钟脉冲来自于 12 分频的单片机时钟脉冲。

C/T = 1，定时器/计数器的工作模式为计数器，计数脉冲来自于 P3.4 (T0) / P3.5 (T1) 引脚输入的计数脉冲。

(3) M1 和 M0——工作方式选择位

M1 和 M0 的 4 种编码对应定时器/计数器的 4 种工作方式，两者具体的对应情况见表 5-1。

表 5-1 定时器/计数器工作方式选择位的配置

M1	M0	工作方式	备 注
0	0	方式 0	13 位计数长度，软件程序装载初值，T0 和 T1 均适用
0	1	方式 1	16 位计数长度，软件程序装载初值，T0 和 T1 均适用
1	0	方式 2	8 位计数长度，自动重新装载初值，T0 和 T1 均适用
1	1	方式 3	特殊情况，仅适用于 T0，分成两个 8 位计数器，T1 停止计数

5.1.4 控制寄存器 TCON

定时器/计数器不仅仅完成简单的定时和计数功能，还具备诸如启动控制、溢出判断和申请中断等功能，这些功能的实现需要依靠控制寄存器 TCON 来完成配置。

特殊功能寄存器 TCON 的字节地址为 88H，可位寻址，位地址为 8FH ~ 88H。寄存器 TCON 的定义如图 5-5 所示。

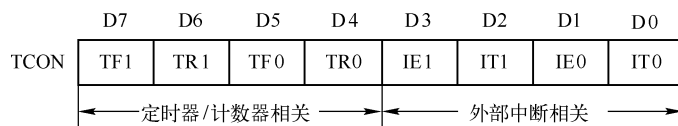


图 5-5 TCON 寄存器定义

在 TCON 寄存器各位的定义中，低 4 位 D3 ~ D0 各位的定义与外部中断源相关，本章不做表述，这里仅对与定时器/计数器相关的高 4 位的定义给予解释。

(1) TF_x (x 取 0 或 1) ——溢出标志位

TF_x = 1，计数值溢出。定时器/计数器核心部件 +1 计数器在每一个输入脉冲作用下计数值加 1。当 +1 计数器的计数值达到最大值时，在下一个输入脉冲作用下产生溢出，由硬件完成 TF_x 的置 1 操作，并同时申请中断。若此时允许定时器/计数器中断，则进入定时器/计数器中断服务程序，由硬件对 TF_x 自动清 0。也可采用软件查询方式，通过程序查询 TF_x 位是否为 1 来判断溢出。由于查询方法的特殊性，查询完成转向处理程序时需通过软件将 TF_x 清 0。

(2) TR_x (x 取 0 或 1) ——运行控制位

TR_x = 1，启动定时器/计数器 T_x 工作（相对控制，是否启动还需要考虑 GATE 位的配置）。

TR_x = 0，停止定时器/计数器 T_x 工作（绝对控制，计数器此时停止工作）。

TR_x 通过软件置 1 或清 0，从而达到对定时器/计数器的启/停控制。

5.2 定时器/计数器的工作方式

5.1 节对定时器/计数器的功能部件结构、工作方式寄存器 TMOD 和控制寄存器 TCON 进行了描述。编程设置 TMOD 的 M1 和 M0 的状态，可以实现定时器/计数器的工作方式选择。本节重点对 4 种工作方式进行描述，以 T0 为例来说明定时器/计数器的工作方式。

5.2.1 方式 1

方式 1 为 16 位定时器/计数器工作方式。T0 工作于方式 1（设置 M1 = 0，M0 = 1）的基本电路结构如图 5-6 所示，T1 工作于方式 1 的电路结构与 T0 相同。该电路结构的核心部件为 16 位 +1 计数器，由特殊功能寄存器 TLO 组成 16 位 +1 计数器的低 8 位，TH0 组成 16 位 +1 计数器的高 8 位。启动 T0 工作之后，16 位 +1 计数器从 TH0 和 TLO 共同组成的初值 X 开始计数，当 16 位 +1 计数器的计数值溢出时（即 FFFFH + 1 = 0000H，产生溢出信号），TLO 和 TH0 复 0，同时 TCON 中的 TF0 被置 1。

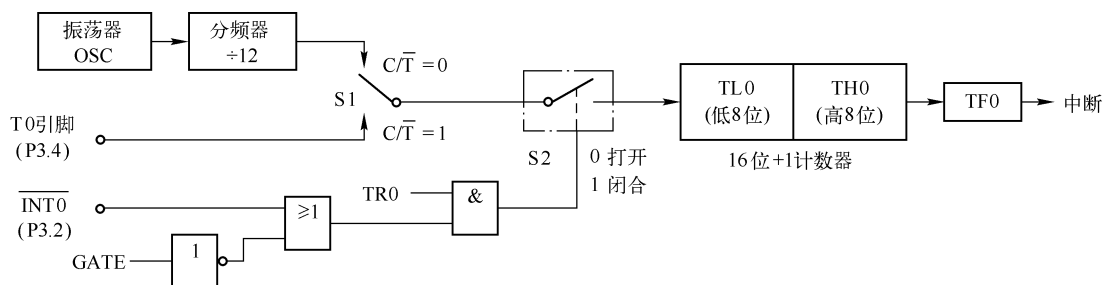


图 5-6 方式 1 基本电路结构框图 (以 T0 为例)

(1) 工作模式选择开关 S1 的控制

TMOD 低四位中的 $C/\bar{T}=0$ 时，图 5-6 中的开关 S1 连接到上侧的位置，此时 T0 工作于定时模式。16 位 +1 计数器的计数脉冲信号来自于 51 单片机时钟振荡器的时钟脉冲的 12 分频。因为时钟振荡器的频率（周期）固定，所以 12 分频后获得的分频信号的频率（周期）也保持固定。以固定周期的脉冲信号触发 16 位 +1 计数器执行 +1 操作，计数次数乘以分频信号周期等于定时时间。

TMOD 低四位中的 $C/\bar{T}=1$ 时，开关 S1 连接到下侧的位置，此时 T0 工作于计数模式。16 位 +1 计数器的计数脉冲信号来自于外部 T0 (P3.4) 引脚输入的计数脉冲信号。因为外部引脚所加载的计数脉冲信号的频率周期未知，所以计数次数代表了外加计数脉冲的个数。

(2) 主通道开关 S2 的控制

与门输出 =0 时，开关 S2 打开，由于 16 位 +1 计数器没有计数脉冲信号的输入，16 位 +1 计数器停止计数，T0 处于停止工作状态。

与门输出 =1 时，开关 S2 闭合，T0 处于工作状态。

分析图 5-6 所示的电路，当 $TR0=0$ 时，无论图 5-6 中与门的另一个输入端是 0 还是 1 状态，与门输出 0，开关 S2 打开，停止计数，T0 处于停止工作状态。

当 $TR0=1$ 时，与门的输出是 0 还是 1 将受控于 GATE 的状态和 $\overline{INT0}$ (P3.2) 引脚的输入状态。

1) $GATE=0$ ，T0 处于非门控状态。则无论 $\overline{INT0}$ (P3.2) 外部引脚的输入状态为 0 还是 1，或门输出 1，T0 是否处于工作状态完全由 $TR0$ 的状态所决定。

2) $GATE=1$ ，T0 处于门控状态。或门输出的状态受控于 $\overline{INT0}$ (P3.2) 外部引脚的输入状态。只有 $\overline{INT0}$ (P3.2) 外部引脚输入状态为“1”时，才使得或门输出 1 状态。这样 $TR1=1$ 时，只有 $\overline{INT0}$ (P3.2) 外部输入状态引脚为“1”时，T0 处于工作状态，否则处于停止工作状态。

综上所述， $M1=0$ ， $M0=1$ ：工作方式 1； $C/\bar{T}=1$ ：计数器； $C/\bar{T}=0$ ：定时器；

$GATE=0$ ， $TR0=0$ ：T0 停止工作； $GATE=0$ ， $TR1=1$ ：T0 处于工作状态；

$GATE=1$ ， $TR1=1$ 和 $\overline{INT0}$ (P3.2) 外部引脚输入状态为“1”时：T0 处于工作状态；

$GATE=1$ ， $TR1=0$ 或 $\overline{INT0}$ (P3.2) 外部引脚输入状态为“0”时：T0 停止工作。

5.2.2 方式2

方式2为自动装载初值的8位定时器/计数器工作方式。T0工作于方式2（设置M1=1，M0=0）的基本电路结构如图5-7所示，T1工作于方式2的电路结构与T0相同。

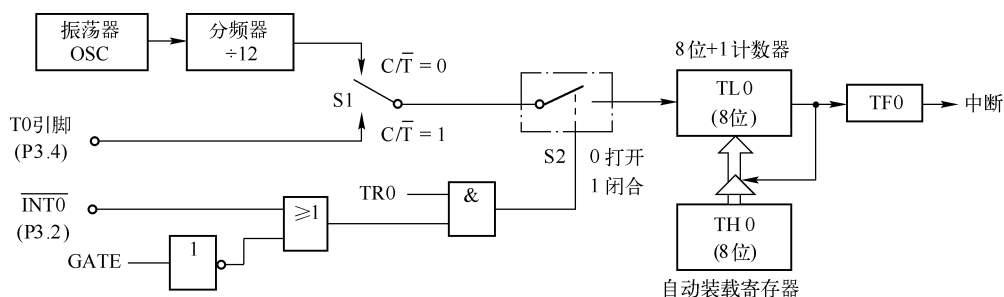


图5-7 方式2基本电路结构框图（以T0为例）

通过对图5-7与图5-6的比较，方式2与方式1的不同在于方式2将方式1的16位+1计数器变为8位+1计数器。特殊功能寄存器TL0组成8位+1计数器，而TH0作为自动装载寄存器存放定时器/计数初值，当8位+1计数器TL0计数溢出时，TH0的定时器/计数初值自动传输给TL0，同时TCON中的TF0被置1。

图5-7的其他部分的电路结构与方式1的电路结构相同，控制方法一致，参阅5.2.1节方式1控制部分的解释，不再赘述。

5.2.3 方式3

只有T0能够工作于方式3。T0工作于方式3（设置M1=1，M0=1）的基本电路结构如图5-8所示。T0工作方式3的引入，目的在于增加一个8位定时器，但由于T0占用了T1的TR1和TF1，导致T0工作于方式3时，T1不能够使用TR1和TF1，使得T1的功能弱化。特别强调，T1不能设置为方式3。

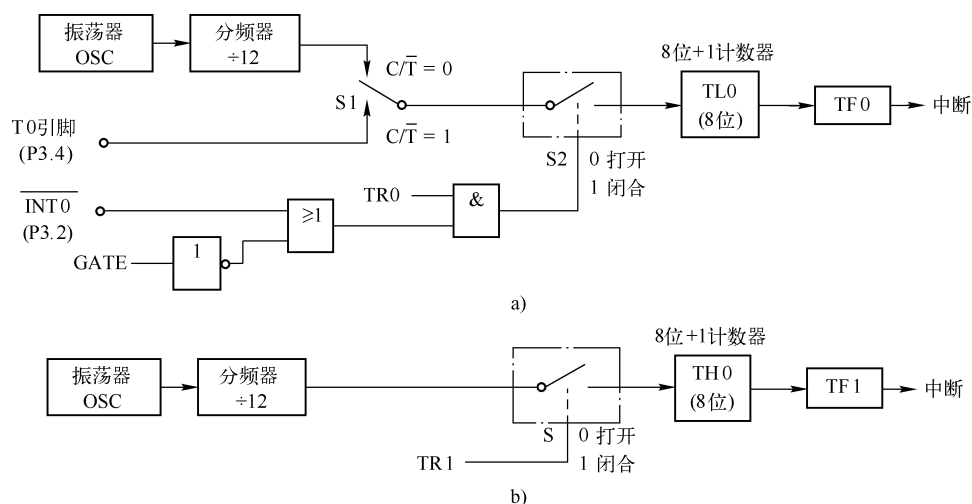


图5-8 方式3基本电路结构框图（仅T0/C0可用）

a) TL0作为8位定时器/计数器 b) TH0作为8位定时器

1. T0 工作于方式 3

从图 5-8 可知, T0 设置为方式 3 时, TL0 和 TH0 分开独立使用。

图 5-8a 与 T0 工作于方式 1 的图 5-6 比较, 唯一的差别在于图 5-8a 的 TL0 作为 8 位 + 1 计数器, 组成 8 位定时器/计数器; 而图 5-6 是将 TL0 和 TH0 组合成 16 位 + 1 计数器使用, 所以两者的控制和应用方法相同。

图 5-8a 的 TL0 作为 8 位定时器/计数器使用 T0 的控制位和引脚: TMOD 的 D3 位 GATE、D2 位 $C/\overline{T}$, TCON 的 TR0、TF0, 引脚 T0 (P3.4) 和引脚 $\overline{INT0}$ (P3.2)。

图 5-8b 的 TH0 作为 8 位 + 1 计数器, 由于计数脉冲只能来自于 12 分频的单片机时钟脉冲, 因此只能作为 8 位定时器使用。图 5-8b 的 TH0 作为 8 位定时器占用了 T1 的控制位: TCON 的 TR1 和 TF1。TR1 控制该定时器的运行和停止, TF1 作为该定时器的溢出信号。

T0 工作于方式 3: 设置 T0 的 M1 = 1, M0 = 1。

1) TL0 作为 8 位定时器/计数器, 其控制方式与 T0 工作于方式 1 相同, 只是 TL0 为 8 位 + 1 计数器, 设置 8 位初值。

2) TH0 能够作为非门控的 8 位定时器, TH0 为 8 位 + 1 计数器, 设置 8 位定时初值, 将 TR1 置 1, 启动 TH0 定时器工作, TF1 作为定时时间到的溢出信号。

2. T0 工作于方式 3 时 T1 的工作状态

由于 T0 被配置为方式 3, TH0 作为 8 位定时器占用了 T1 的 TR1 和 TF1, 但 T1 依然可以设置为其他的工作方式。在实际应用中, T0 工作在方式 3 时, T1 通常作为串行口的波特率发生器使用。

T0 工作在方式 3 时, T1 工作于方式 1 的基本电路框图如图 5-9 所示, T1 工作于方式 2 的基本电路框图如图 5-10 所示。

T1 工作于方式 1: 设置 T1 的 M1 = 0, M0 = 1, 设置 $C/\overline{T}$ = 0, 启动 T1 计数, 其输出为串行口的输出脉冲, T1 作为串行口的波特率发生器使用。

T1 工作于方式 2: 设置 T1 的 M1 = 1, M0 = 0, 设置 $C/\overline{T}$ = 0, 启动 T1 计数, 其输出为串行口的输出脉冲, T1 作为串行口的波特率发生器使用, 注意此时 TL1 为 8 位 + 1 计数器, 由于可以由 TH1 自动装载定时初值, 作为波特率发生器输出的周期脉冲更为精确。

如果设置 T1 的 M1 = 1, M0 = 1, 则 T1 停止工作, 即停止计数操作。

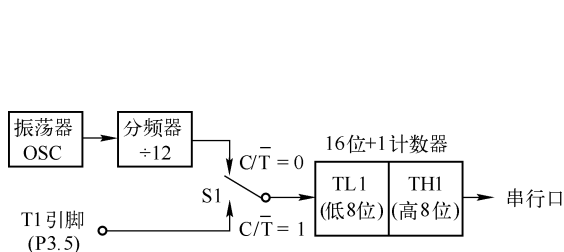


图 5-9 T1 工作于方式 1 的基本电路框图

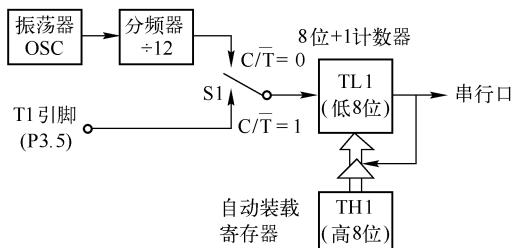


图 5-10 T1 工作于方式 2 的基本电路框图

5.2.4 方式 0

定时器/计数器工作于方式 0 (设置 M1 = 0, M0 = 0) 的基本电路结构与工作于方式 1

的基本电路结构（图 5-6）的区别仅仅在于计数器的位数，方式 1 由 TL0 和 TH0 组成 16 位 +1 计数器，而方式 0 由 TL0 的低 5 位和 TH0 的 8 位共同组成 13 位 +1 计数器。方式 0 和方式 1 的其他控制方法一致。

由于方式 0 为 13 位 +1 计数器，计算初值麻烦且易出错，其功能完全可以由方式 1 替代，因此使用很少，不做讨论。

5.3 定时器/计数器的应用

定时器/计数器是所有的微处理器不可或缺的组成部件，应用极其广泛。在前两节中具体阐述了定时器/计数器的基本概念，本节通过具体的举例说明定时器/计数器的使用方法。

定时器/计数器是一种可编程部件，51 单片机应用定时器/计数器主要有查询和中断两种编程方法，本节主要介绍查询编程的方法，第 6 章将介绍定时器/计数器的中断编程方法。

无论是查询还是中断的方法编程，定时器/计数器的编程主要有以下几个步骤：

1) 计算初值。由于定时器/计数器内部都是 +1 计数器，定时器/计数器都是从设置的初值开始进行 +1 操作，直到定时器/计数器产生溢出或者被停止。

① 计数器。设计数最大值为 M ，计数初值为 X 。

$$\text{方式 1:} \quad X = 2^{16} - M = 65536 - M \quad (5-1)$$

$$\text{方式 2、3:} \quad X = 2^8 - M = 256 - M \quad (5-2)$$

② 定时器。设定时器定时时间为 t ，定时初值为 X 。

设单片机振荡器的时钟频率为 f_{osc} ，则 12 分频后的时钟周期 $T = 12/f_{\text{osc}}$ 。

$$\text{方式 1:} \quad X = 2^n - \frac{t}{T} = 2^{16} - \frac{t \times f_{\text{osc}}}{12} = 65536 - \frac{t \times f_{\text{osc}}}{12} \quad (5-3)$$

$$\text{方式 2、方式 3:} \quad X = 2^n - \frac{t}{T} = 2^8 - \frac{t \times f_{\text{osc}}}{12} = 256 - \frac{t \times f_{\text{osc}}}{12} \quad (5-4)$$

2) 定时器/计数器的初始化编程。初始化编程包括初值的设置、工作模式和工作方式的设置及定时器/计数器的启动控制等，如果采用中断方式，还需要对中断系统进行初始化。

(3) 定时器/计数器所要完成功能的程序设计。

5.3.1 方式 1 的应用举例

例 5-1 某 51 单片机应用系统，系统晶振的时钟频率采用 12 MHz，在 P1.0 引脚输出一个周期为 10 ms 的方波信号，如图 5-11 所示。

分析：根据题目要求，输出方波信号的周期为 10 ms，则半个周期为 5 ms。选用定时器 T1，设定 T1 定时时间为 5 ms。启动 T1 定时后，判断 T1 的溢出标志 TF1 是否为 1，如果 TF1 = 1，表明 5 ms 定时时间已到，软件清除 TF1，取反 P1.0，不断循环即可实现 10 ms 周期方波的输出。

(1) 计算定时初值 X

时钟频率 $f_{\text{osc}} = 12 \text{ MHz}$ ，12 分频后为 1 MHz，周期为 $1 \mu\text{s}$ ，定时时间为 5 ms，按照式 (5-3) 计算定时初值 X 。

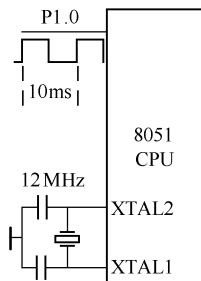


图 5-11 例 5-1 电路

$$X = 65536 - \frac{t \times f_{osc}}{12} = 65536 - \frac{t}{T} = 65536 - \frac{5 \times 10^{-3}}{10^{-6}} = 65536 - 5000 = 60536 = EC78H$$

(2) 初始化定时器 T1

采用查询方法编程。初始化程序主要完成对 TL1、TH1、TMOD 和 TCON 的初始化设置。

(3) 完成功能程序设计

思路：查询 TF1 是否为 1 状态，如果 TF1 = 1，调用定时处理子程序，即 P1.0 引脚的取反操作，编程时，利用 JBC 指令，实现位判断后自动对位单元清 0 的功能。

编写程序如下：

```

ORG      0000H
LJMP     MAIN
ORG      0030H
MAIN:    MOV     SP,#60H      ;开辟堆栈空间,栈底的地址设置为 60H
         MOV     TMOD,#10H    ;初始化设置 T1 为工作方式 1,非门控的定时器模式
         MOV     TCON,#00H    ;初始化设置 T1 相关标志位,TR1 = 0,TF1 = 0
         MOV     TL0,#78H     ;初始化设置定时初值低 8 位
         MOV     TH0,#0ECH    ;初始化设置定时初值高 8 位
         SETB    TR1          ;启动 T1 开始定时
LOOP:    JBC     TF1,TIME1    ;判断 T1 是否溢出,若 TF1 = 0,顺序执行
                                   ;如果 TF1 = 1,清 TF1,转移到 TIME1 执行
         SJMP     LOOP        ;TF1 = 0,返回 LOOP 再继续查询
TIME1:   LCALL   CPLP10       ;TF1 溢出位有效,调用定时处理子程序 CPLP10
         SJMP     LOOP        ;实现产生方波脉冲的循环输出
         ORG      0090H       ;定时处理子程序入口地址
CPLP10:  MOV     TL1,#78H     ;重新设置定时初值低 8 位
         MOV     TH1,#0ECH    ;重新设置定时初值高 8 位
         CPL      P1.0        ;取反 P1.0,输出 10 ms 方波
         RET              ;子程序返回
         END                ;汇编源文件编译结束

```

例 5-2 某 51 单片机应用系统，系统晶振的时钟频率采用 12 MHz，通过定时器 T0 产生 1s 的定时，驱动 P1.0 引脚电平取反操作程序。

分析：根据题目要求，利用定时器 T0，设定 T0 工作于方式 1，定时初值设为 0000H，启动定时器后等待计数溢出。根据核心部件 +1 计数器的特点，每一个时钟脉冲计数值加 1，而方式 1 的最大计数值为 FFFFH，考虑初值为 0000H，即经过 65536 个时钟脉冲周期后 +1 计数器溢出，TF0 置位。

根据例 5-2 分析可知，此时的计数时钟周期为 1 μs，则定时器 T0 最长定时时间为 65536 × 1 μs，即 65.536 ms。

据此归纳，时钟频率为 12 MHz 时，各工作方式最长定时时间如下：

方式 0 最长定时时间：8.192 ms(2¹³ × 1 μs)；

方式 1 最长定时时间：65.536 ms(2¹⁶ × 1 μs)；

方式 2 最长定时时间：0.256 ms(2⁸ × 1 μs)；

方式 3 最长定时时间: $0.256\text{ ms}(2^8 \times 1\text{ }\mu\text{s})$ 。

由上可见,即便选择方式 1,也无法直接实现 1 s 定时功能。此时可采用“硬件(定时器)定时 + 软件计数”的方法达到目标要求。设定定时时间为 50 ms ,通过程序设置循环变量计数 20 次,可得到 $20 \times 50\text{ ms} = 1000\text{ ms} = 1\text{ s}$ 的定时时间。

(1) 计算定时初值 X

时钟频率 $f_{\text{OSC}} = 12\text{ MHz}$, 12 分频后为 1 MHz , 周期为 $1\text{ }\mu\text{s}$, 定时时间 50 ms , 按照式(5-3) 计算定时初值 X。

$$X = 65536 - \frac{t}{T} = 65536 - \frac{50 \times 10^{-3}}{10^{-6}} = 65536 - 50000 = 15536 = 3\text{CB0H}$$

(2) 初始化定时器 T0

采用查询方法编程。初始化程序主要完成对 TLO、TH0、TMOD 和 TCON 的设置,以及循环变量计数值的初始化设置。设循环变量 R0 存放计数值,则初始化其值为 $(R0) = 20$ 。

(3) 完成功能程序设计

思路: 查询 TF0 是否为 1, 如果 $\text{TF0} = 1$, 调用定时处理子程序。此时,此次过程仅仅实现了 50 ms 定时。然后对 R0 循环变量计数值减 1, 当循环 20 次后, R0 中的值为 0 时即实现了 1 s 定时。编程时, 利用 JBC 指令, 实现自动对位单元清 0 的功能。

编写程序如下:

```
ORG    0000H
LJMP   MAIN
ORG    0030H
MAIN:  MOV    SP, #60H        ;开辟堆栈空间
        MOV    TMOD, #01H    ;设置 T0 为工作方式 1, 定时器模式
        MOV    TCON, #00H    ;设置 T0 相关标志位, TR0 = 0, TF0 = 0
        MOV    TLO, #0B0H    ;设置定时初值低 8 位
        MOV    TH0, #3CH     ;设置定时初值高 8 位
        MOV    R0, #20       ;初始化定时 1 s 的循环变量计数值为 20
        SETB   TR0          ;启动 T0 开始定时
LOOP:  JBC     TF0, TIME0     ;判断 T0 是否溢出, 若 TF0 = 0, 顺序执行
                                ;如果 TF0 = 1, 清 TF0, 转移到 TIME0 执行
        SJMP   LOOP          ;返回 LOOP 再继续查询
TIME0: LCALL   MS50          ;TF0 溢出位有效, 调用定时处理子程序 MS50
        SJMP   LOOP          ;实现产生方波脉冲输出的循环控制
ORG    0090H
MS50:  DJNZ   R0, NEXT       ;判断是否已产生了 20 次 50 ms 的定时过程
        CPL    P1.0          ;已产生了 1000 ms 的定时, 取反 P1.0
        MOV    R0, #14H     ;重新设置循环变量计数值为 20 (20 = 14H)
NEXT:  MOV    TLO, #0B0H    ;重新设置定时初值低 8 位
        MOV    TH0, #3CH    ;重新设置定时初值高 8 位
        RET                     ;子程序返回
END                     ;汇编源文件编译结束
```

5.3.2 方式2的应用举例

方式2为自动装载初值的8位定时器/计数器工作方式。因为其初值自动重新装载，避免程序的再次干预耗费时间，所以作为定时器使用时，定时精度比方式1高。

例5-3 编写应用T0实现对T0(P3.4)引脚的方波脉冲个数检测程序。每检测到100个脉冲，驱动P1.0引脚电平取反，如图5-12所示。

分析：根据题目要求，将T0设定为计数器模式，工作于方式2，外部待计数方波脉冲信号由T0(P3.4)引脚引入，每产生一次下降沿跳变，T0的+1计数器的计数值加1，由程序查询TF0状态。

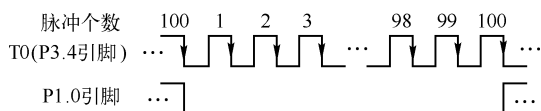


图5-12 例5-3示意图

(1) 计算计数初值X

由于是计数工作模式，作为计数器的+1操作与系统晶振时钟无关。因要求累计计数次数M=100，按照式(5-2)，T0的计数初值X为

$$X = 2^8 - M = 256 - 100 = 156 = 9CH$$

(2) 初始化计数器T0

采用查询方法编程。初始化程序主要完成对TL0、TH0、TMOD和TCON的初始化设置。

(3) 完成功能程序设计

查询TF0是否为1状态，如果TF0=1，调用计数处理子程序，执行P1.0取反操作。

```

ORG    0000H
LJMP   MAIN           ;跳转到主程序
ORG    0030H
MAIN:  MOV    SP, #60H      ;开辟堆栈空间
      MOV    TMOD, #06H    ;初始化T0为工作方式2,计数器模式
      MOV    TCON, #00H    ;初始化T0相关标志位
      MOV    TL0, #9CH     ;初始化8位计数初值
      MOV    TH0, #9CH     ;初始化TH0,必须满足(TH0)=(TL0)
      SETB   TR0           ;启动T0工作
LOOP:  JBC    TF0, CNT100   ;判断T0是否溢出,若TF0=0,顺序执行
                        ;如果TF0=1,清TF0,转移到CNT100执行
                        ;当TF0由0→1时,单片机自动将(TH0)→TL0
      SJMP   LOOP          ;反复查询
CNT100: LCALL  CPLP10       ;TF0溢出位有效,调用计数处理子程序
      SJMP   LOOP          ;实现产生方波脉冲的循环输出
      ORG    0090H         ;计数处理子程序入口地址
CPLP10: CPL    P1.0        ;已计数100个脉冲,取反P1.0
      RET                ;子程序返回
      END                ;汇编源文件编译结束

```

由例5-3可以看出，在自动装载初值方式2的工作方式中，计数溢出之后调用的CPLP10子程序中不需要编写指令重新装载初值，而由单片机硬件自动完成，提高了程序执

行的速度。

5.3.3 方式3的应用举例

例 5-4 某 51 单片机应用系统，要求 P1.0 引脚以 200 μs 的周期输出方波，P1.1 引脚以 482 μs 的周期输出方波，且同时以 T1 作为串行通信的波特率发生器（系统晶振采用 12 MHz，波特率为 9600 bit/s）。

分析：根据题目要求，至少需要三个硬件定时器/计数器完成此项工作（不考虑延时程序定时的编程方法），而 51 单片机内部仅仅具备两个硬件定时器/计数器，不足以解决问题。方式 3 的提出恰好可以解决此问题，应用 T0 实现两个硬件的 8 位定时器（参见图 5-8），将 T1 设置为方式 2（参见图 5-10），构建了三个硬件定时器，满足了本题目的要求。

(1) 计算定时初值 X

1) TL0 作为 8 位定时器。按照式 (5-4)：
$$X_{TL0} = 2^n - \frac{t}{T} = 2^8 - \frac{100 \times 10^{-6}}{10^{-6}} = 256 - 100 = 156 = 9CH。$$

2) TH0 作为 8 位定时器。按照式 (5-4)：
$$X_{TH0} = 2^n - \frac{t}{T} = 2^8 - \frac{241 \times 10^{-6}}{10^{-6}} = 256 - 241 = 15 = 0FH。$$

3) T1 为工作于方式 2 的定时器。定时初值 X_{bps} 计算公式为（具体原理详见第 7 章串行通信及应用）

$$X_{bps} = 2^8 - \frac{12 \times 10^6 \times (0 + 1)}{384 \times 9600} = 256 - 3 = 253 = FDH$$

(2) 初始化定时器工作状态

采用查询方法编程。初始化程序主要完成对 T0、T1 相关寄存器的初始化。

(3) 完成功能程序设计

根据 TF0、TF1 的查询结果，分别执行 P1.0 和 P1.1 的取反操作。

```
ORG    0000H
LJMP   MAIN
ORG    0050H
MAIN:  MOV    SP, #60H
        MOV    TMOD, #23H    ;初始化 T0 定时:方式 3;T1 定时:方式 2
        MOV    TCON, #00H    ;初始化 T0、T1 相关标志位
        MOV    TL0, #9CH     ;设置 TL0 的定时初值 X:t = 100  $\mu\text{s}$ 
        MOV    TH0, #0FH     ;设置 TH0 的定时初值 X:t = 241  $\mu\text{s}$ 
        MOV    TL1, #0FDH    ;设置波特率发生器的定时初值  $X_{bps}$ 
        MOV    TH1, #0FDH    ;设置 TH1 的初值,(TH1) = (TH0) = FDH
        SETB   TR0           ;启动 T0 的 TL0 定时
        SETB   TR1           ;启动 T0 的 TH0 定时
LOOP:   JBC     TF0, TIME100   ;判断 100  $\mu\text{s}$  定时计数是否溢出,溢出转移
        JBC     TF1, TIME241   ;判断 241  $\mu\text{s}$  定时计数是否溢出,溢出转移
```

	SJMP	LOOP	;反复查询
TIME100:	MOV	TL0,#9CH	;100 μ s 定时处理程序,重新装载定时参数初值
	CPL	P1.0	;每个 100 μ s 取反一次 P1.0
	SJMP	LOOP	;实现周期为 200 μ s 的方波脉冲的循环输出
TIME241:	MOV	TH0,#0FH	;241 μ s 定时处理程序,重新装载定时初值
	CPL	P1.1	;每个 241 μ s 取反一次 P1.1
	SJMP	LOOP	;实现周期为 482 μ s 方波脉冲的循环输出
	END		;汇编源文件编译结束

工作于方式 2 的 T1, 设置工作方式之后, 无须附加其他控制即可自动实现串行口输出波特率为 9600 bit/s 的串行通信时钟信号。

例 5-5 利用定时器 T0 测量 $\overline{\text{INT0}}$ (P3.2) 引脚外部输入周期性方波信号的高电平 T_{Hx} 持续时间 (振荡频率 $f_{\text{osc}} = 12 \text{ MHz}$)。

分析: 根据题目要求, 待测方波如图 5-13 所示。由于不对低电平测量, 可以应用 T0 门控 GATE 方式进行测量, 设置 T0 为定时器模式, 工作于方式 1, 定时初值:



图 5-13 例 5-5 待测方波示意图

$(\text{TH0}) = (\text{TL0}) = 00\text{H}$ 。

1) $\text{GATE} = 1$, $\overline{\text{INT0}}$ 输入为低电平时, 设置 $\text{TR0} = 1$, T0 停止定时。

2) $\text{GATE} = 1$, $\text{TR0} = 1$, $\overline{\text{INT0}}$ 输入由 0 $\rightarrow$ 1 的上升沿跳变时, T0 开始定时。即在图 5-13 中高电平 T_{Hx} 的上升沿开始定时, TH0 和 TL0 组成的 16 位 +1 计数器从 0000H 开始 +1 计数。

3) 当 $\overline{\text{INT0}}$ (P3.2) 引脚输入由高电平 $\rightarrow$ 低电平时, 即图 5-13 中高电平 T_{Hx} 的下降沿, T0 停止定时, 读出并存放 TH0 和 TL0 中的定时计数值, 该值乘以 T 就是高电平 T_{Hx} 的持续时间。

	CNT_H	EQU	45H	
	CNT_L	EQU	44H	
	ORG	0000H		;程序开始
	LJMP	MAIN		;跳转到主程序
	ORG	0050H		;主程序入口地址
MAIN:	MOV	SP,#60H		;开辟堆栈空间
	MOV	TMOD,#09H		;设置 T0 门控方式,工作于定时方式 1,GATE = 1
	CLR	A		
	MOV	TCON,A		;初始化 T0 相关标志位
	MOV	TL0,A		;初始化定时低 8 位初值(TL0) = 00H
	MOV	TH0,A		;初始化定时高 8 位初值(TH0) = 00H
	SETB	P3.2		;设置 P3.2 引脚为高电平,为正确读取输入状态做好准备
	JB	P3.2,\$		;P3.2 = 0,顺序执行;P3.2 = 1,等待待测方波低电平的出现
	SETB	TR0		;为启动 T0 定时计数做好准备
LOOP:	SETB	P3.2		;P3.2 输入为 0,设置 P3.2 = 1,为正确读取输入状态做好准备
	JNB	P3.2,LOOP		;P3.2 = 0 转移;P3.2 = 1 顺序执行,并启动 T0 定时计数

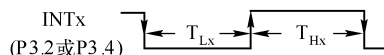
JB	P3.2,\$	;等 P3.2 上待测方波低电平的出现;P3.2 =0,顺序执行
CLR	TR0	;测量完成,清零 TR0,T0 停止定时计数
MOV	CNT_H,TH0	;存放定时计数结果的高 8 位时间数据
MOV	CNT_L,TL0	;存放定时计数结果的低 8 位时间数据
SJMP	\$	;原地踏步
END		

例 5-5 中通过几条 JB 和 JNB 指令避免对检测方波的高电平持续时间的错误测量,保证了测量开始是从一个完整的高电平开始时刻进行。此外,本例也具有其局限性,例如方波高电平的持续时间必须小于 65.636 ms。

5.3.4 51 单片机外部输入的计数脉冲信号的要求

在例 5-3 中 T0 (P3.4) 引脚输入的计数脉冲信号必须满足一定的条件:一是计数脉冲信号不能够有抖动,即计数脉冲信号的上升沿和下降沿不能够有抖动,否则计数不正确。二是计数脉冲信号的最大频率要小于单片机振荡时钟频率的 1/24。例如,如果 51 单片机应用系统是 12 MHz 振荡时钟频率,则计数脉冲信号的最大频率必须小于 500 kHz;如果是 6 MHz,则计数脉冲信号的最大频率必须小于 250 kHz。三是对于计数脉冲占空比的要求:高电平 T_H 和低电平 T_L 分别保证维持一个机器周期以上即可。参见图 5-14,即 $T_{Lx} \geq$ 一个机器周期, $T_{Hx} \geq$ 一个机器周期。

究其原因,主要是由单片机执行指令的具体过程和特点所决定。对于定时器/计数器工作在计数模式下,



触发脉冲信号来自于外部引脚 T0 (P3.4) 或 T1 (P3.5)。图 5-14 计数脉冲信号占空比示意图

定时器/计数器的 +1 计数器的加 1 操作发生在触发脉冲由 1→0 的下降沿跳变过程中。根据单片机执行指令的时序图 (参见图 2-7 ~ 图 2-9), 每个机器周期的 S5P2 期间, 均对外部输入引脚 T0 或 T1 进行电平采样。如果在前一个机器周期的采样过程中, T0 引脚电平为 “1”, 而在当前机器周期的采样中 T0 引脚电平为 “0”, 即表明产生了一个下降沿, 则在下一个机器周期的 S3P1 期间, +1 计数器将进行加 1 操作。

习题

5-1 51 单片机内部有几个定时器/计数器? 它们是由哪些特殊功能寄存器组成? 计数脉冲的来源是什么?

5-2 如果 51 单片机晶振频率 $f_{osc} = 3 \text{ MHz}$, 分析定时器/计数器工作在方式 1 和方式 2 的最大定时时间各为多少?

5-3 简述定时器/计数器的门控信号 GATE 设置为 1 时, 定时器的启动方法。

5-4 编程实现选用 T1 方式 2 产生 $500 \mu\text{s}$ 的定时程序, 在 P1.0 输出周期为 1 ms 的方波 ($f_{osc} = 6 \text{ MHz}$)。

5-5 已知 51 单片机应用系统采用 6 MHz 晶振, 应用 T0 定时, 在 P1.0 输出一个高电平宽度为 100 ms、低电平宽度 50 ms 的方波信号。

5-6 仔细思考例 5-5, 分析是否可以利用定时器实现超宽脉冲的时间测量。如果可以,

说明如何操作；如果不可以，说明为什么。

5-7 利用 51 单片机产生一个实时时钟，上电初始时钟从 00:00:00 开始，以秒为单位。

5-8 使用定时器/计数器编写跑马灯的程序：实现连接在 P1 口上的 8 个 LED 依次亮灭。每次只能一个灯状态变化，且按由 P1.7 ~ P1.0 的顺序依次亮灭，每次变化时间间隔 100 ms，设 51 单片机晶振时钟频率为 6 MHz。

5-9 对于软件延时程序和定时器/计数器定时程序的执行过程，分析比较这两种定时方式的特点和优缺点。

第6章 中断系统

中断系统是微处理器的一个基本功能，任何一种型号的微处理器都设置有中断系统。51 单片机也具备相应的中断系统。中断技术有效地保证了微处理器对随机发生的事件能够及时响应，有利于实时检测与控制。微处理器中断系统的基本原理一致，区别仅在于结构、配置、使用和功能复杂度等具体细节。

本章主要介绍 51 单片机的中断系统及其工作原理，并通过适当的举例加深读者对中断系统概念的理解，掌握中断技术的编程方法。

6.1 中断系统概念

6.1.1 中断技术

中断技术是十分重要而复杂的技术，由微处理器的软、硬件共同完成，称为中断系统。中断技术概念来源于人们的日常生活，来源于人们的活动和理解，并通过抽象转化为一种切实可行、必不可少的技术。比如，A 同学得知 B 同学下午将送一件物品过来，但是不知道具体时间。A 同学不能因此而空等，于是按计划去图书馆读书学习。当 B 同学到达后电话通知 A，A 标记当前学习内容的具体位置后，去处理 B 的事情及安放物品，之后再回到图书馆从已做好标记的位置继续学习。从这一事件中可以抽象出如下的信息处理方法，如图 6-1a 所示。

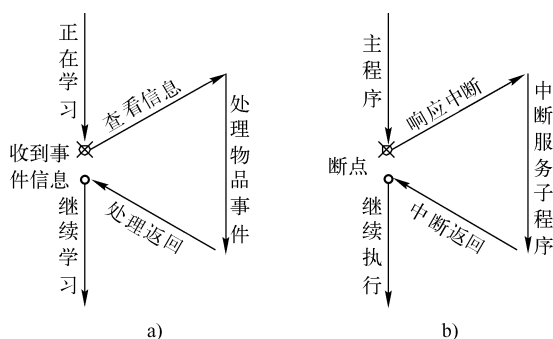


图 6-1 中断处理的基本过程

a) 学习过程中断处理 b) 计算机中断处理

1) 已设置一个事件（同学来送物品），但不知何时会出现，即事件出现的时间是随机的。

2) 随机（中断）事件出现时（同学电话通知已到达），完成已在执行的指令（阅读完一个完整的语句），保存好信息（标记当前学习内容的具体位置），为处理完随机事件后返回现工作状态做好准备。

3) 暂停正在进行的工作，处理随机（中断）事件（处理物品事件）。

4) 处理完随机（中断）事件，返回处理被中断的工作（从已做好标记的位置继续学习）。

对于微处理器的中断技术，有着类似的处理过程。当微处理器正在执行主程序的过程中，得到一个事件的中断请求，执行完当前正在执行的指令，记录下一条指令的地址，转去执行中断事件的中断服务程序，执行完中断服务程序返回已记录的指令地址的位置（中断返回）继续执行程序，如图 6-1b 所示。这个过程，被称为“中断”。

6.1.2 中断技术的主要功能特点

对于中断技术的理解，可以从中断的整个过程着眼加以认识，逐步熟悉中断的主要功能。

1. 设置中断源

中断源：引起中断发生的特殊事件（中断请求源）。中断源的数量决定了微处理器处理突发事件的能力。微处理器中通常具有硬件中断源和软件中断源，51 单片机只设置了 5 个硬件中断源。编程者编写任何应用程序过程中，都会根据需求自主设置中断事件，即根据功能要求分配相应的中断源，也就是说，编程者已知有什么类型的随机事件发生。

2. 甄别中断请求

中断请求：某个中断源相应随机事件发生后向微处理器发出事件产生的中断请求信号。中断请求的甄别就是对已发生中断事件中中断源的识别，即识别出中断请求来自于哪一个中断源。

3. 中断优先级

中断优先级：多个不同的中断源可安排不同的抢占式中断优先级。当正在执行低优先级中断服务程序时，可以被高优先级中断请求抢占打断，待高优先级中断处理完毕之后，再返回低优先级中断服务程序执行，实现多级中断嵌套。但低优先级不能够中断高优先级的中断服务程序的执行；同级中断优先级也不能够中断同级优先级的中断服务程序的执行。51 单片机可以由软件设置成高、低两个抢占式中断优先级，能够实现两级中断嵌套。

51 单片机的硬件还设置了 5 个响应式自然优先级。当若干个中断源同时发出中断请求后，单片机不可能同时响应这些中断源的中断请求，因为单片机在任何时刻只能够执行一个程序模块，所以首先响应自然优先级别最高中断源的中断请求。

微处理器如何设置和识别中断源的不同中断优先级，这是中断系统的一个必备功能。

4. 自动响应中断请求

中断响应：微处理器接到中断请求后响应该中断源的中断请求。操作的过程是首先完成当前正在执行的指令，并保存下一条指令的地址（断点）；暂停当前正在执行的程序，将程序执行的地址转移到该中断源中断事件的中断服务程序的首地址（中断入口地址，也称为中断向量）。这个过程无须程序干预，微处理器硬件自动完成。

5. 中断的处理

中断处理：对请求中断事件的处理过程，也称为中断服务。对于 51 单片机，需要编程者对每一个中断事件编写一个中断服务程序，在中断服务程序中完成中断事件的处理。

6. 中断返回

中断返回：在执行完中断服务程序后，返回到被中断前一刻断开的断点位置，从响应中断时保存的指令地址单元开始执行。中断返回过程与中断响应过程前后呼应，重点解决如下几个事情：

1) 断点：微处理器响应中断请求后被打断的位置。在单片机程序结构中，断点就是被打断程序执行时的下一条指令在程序存储器中的 16 位存储单元地址。

2) 保护现场：中断响应过程中，微处理器除对断点的 16 位地址自动压栈保存外，还应在编写中断服务程序时注意其他相关寄存器和存储单元内容、标志位等重要信息的压栈保存。

3) 恢复现场：中断返回过程中，除了对断点的 16 位地址自动出栈外，在编写中断服务程序时需要将保护现场过程中已压栈的寄存器和存储单元内容、标志位等信息出栈，恢复到执行中断服务程序之前的程序执行状态，从而保证中断服务程序的执行不破坏原数据的完整性。

7. 中断嵌套

中断嵌套：中断过程中的再次中断，与中断源的中断优先级密切相关。当微处理器正在处理一个低优先级的中断事件（执行处理该中断事件的中断服务程序）过程中，发生了高优先级中断源的中断申请，此时，微处理器将中断当前中断服务程序的执行，继而转去执行高优先级中断事件的中断服务程序。这样的过程被称为中断嵌套。中断源的低优先级和高优先级的设定由程序控制中断系统中断优先级寄存器完成。中断嵌套的示意图如图 6-2 所示。

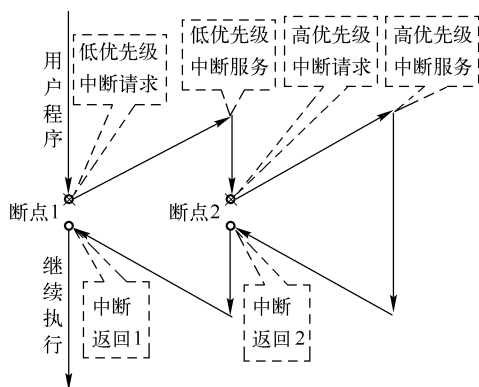


图 6-2 两级中断嵌套示意图

6.1.3 中断技术的优点

中断技术使得微处理器具备了“分时控制，提高工作效率”和“实时控制，及时进行处理”的双重优点。

分时控制，相对于程序查询的控制方式而言，提高了微处理器工作效率。一般程序设计中，主要应用“查询方式”和“中断方式”两种方法进行编程。第 5 章定时器/计数器的应用举例中所有程序均采用了查询的方法。在查询方法中，虽然 51 单片机的定时器/计数器在定时/计数的过程中不占用程序执行时间，但由于需要不断地判断溢出标志位 TF_x 的状态，占用了单片机程序执行时间，直到 TF_x 被置位才转去执行处理定时器/计数器的功能程序。而定时器/计数器如果采用中断技术，不但在定时/计数的过程中不占用程序执行时间，而且 TF_x 状态的变化不需要程序查询，因为当 $TF_x = 1$ 时，硬件自动完成定时器/计数器向 51 单片机发出中断请求，单片机此时若响应中断就会转去执行定时器/计数器的中断服务程序。这样实现了分时处理，极大地提高了 51 单片机的工作效率。

实时控制，及时进行处理。实时控制过程要求快速反应，保证事件及时得到响应。在微处理器应用系统中，经常需要完成诸如输入信息的定时采样、人机界面中的按键检测及随机

发生的故障信息处理等。由于采用中断技术，每个随机事件信息可随时发出中断请求，微处理器快速的中断响应处理，从而保证了对随机事件信息的立刻处理，而无须微处理器的程序查询判断，达到实时处理的目的。

6.2 51 单片机的中断系统及其管理

6.2.1 中断系统的结构

51 单片机的中断系统主要包括 5 个中断源及其相关的控制部分。每个中断源分别对应 1 个固定的中断入口地址（即中断向量）。除硬件电路外，主要与中断系统相关的是 4 个特殊功能寄存器：定时器/计数器的控制寄存器 TCON、串行口控制寄存器 SCON、中断允许寄存器 IE 和中断优先级寄存器 IP。51 单片机具有固有顺序的中断自然优先级，程序可以设定高、低抢占中断优先级。程序还可以独立地设定每一个中断源是被允许中断还是被禁止中断。

51 单片机中断系统结构模型示意图如图 6-3 所示。

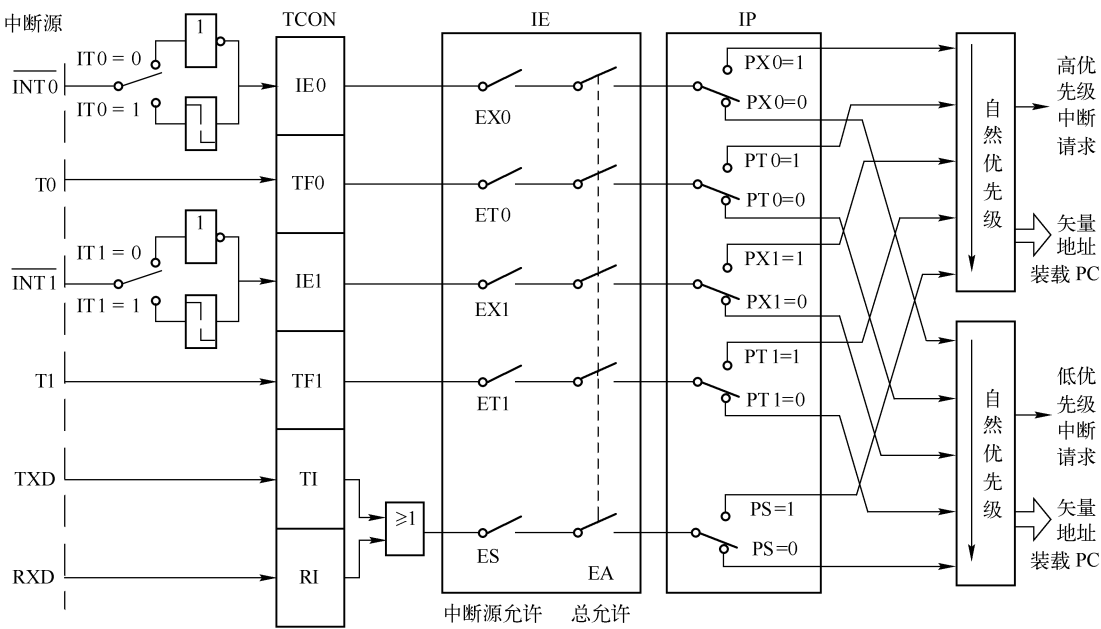


图 6-3 中断系统结构模型示意图

6.2.2 中断源

1. 5 个中断源

由图 6-3 中的中断系统结构模型示意图可知，中断系统共有 5 个中断源。

- 1) 外部中断 0 ($\overline{\text{INT0}}$)：由 P3.2 引脚输入的中断请求信号。
- 2) 定时器 T0 中断 (T0)：定时器/计数器 T0 的定时/计数溢出产生的中断请求信号。

- 3) 外部中断 1 ($\overline{\text{INT1}}$): 由 P3.3 引脚输入的中断请求信号。
- 4) 定时器 T1 中断 (T1): 定时器/计数器 T1 的定时/计数溢出产生的中断请求信号。
- 5) 串行口中断 (ES): 由串行数据发送完成 (TI) 或串行数据接收完成 (RI) 产生的中断请求信号。

5 个中断源可以分为 3 类, 分别是外部中断类、定时/计数中断类和串行口中断类。

2. 中断入口地址

当单片机查询到某个中断源的中断请求时, 该中断源满足中断允许的情况下响应中断, 程序的执行将转向中断服务程序处理中断事件。如何进入中断服务程序? 单片机响应中断时必须根据请求中断的中断源自动转移到该中断源的中断服务程序执行, 因此提出了中断入口地址的概念。

51 单片机硬件系统中, 每一个中断源的中断服务程序入口地址对应程序存储器的一个单元地址, 5 个中断源与 5 个中断服务程序入口地址一一对应, 且固定不变。如果某个中断源发出中断请求被响应, 单片机自动将该中断源对应的中断服务程序入口地址送入 PC, 并控制程序跳转到中断服务程序执行。这些地址被称为“中断入口地址”或“中断向量”。

51 单片机的 5 个中断源及其中断入口地址的对应关系见表 6-1, 从表 6-1 可知:

表 6-1 中断源及其中断入口地址

中 断 源	中断入口地址	自然优先级
外部中断 0	0003H	高
定时器 T0	000BH	↓
外部中断 1	0013H	↓
定时器 T1	001BH	↓
串行口	0023H	低

1) 所有中断源的入口地址按规定分配在程序存储器的 0003H ~ 0023H 之间的 5 个存储单元地址, 表 6-1 称为中断入口地址表, 或称为中断向量表。

2) 51 单片机的中断入口地址按 8 字节间隔的顺序递增往下排列, 此 8 字节保留的存储空间只能编写一些简单的指令。因此, 通常会在中断入口地址的单元编写一条无条件转移指令, 使得程序跳转至中断服务程序执行。

3) 表 6-1 中的中断源自上而下的排列顺序就是中断的自然优先级由高到低的顺序。51 单片机按照该顺序查询中断源的中断请求信号, 当若干个中断源同时申请中断时, 首先响应被优先查询到的中断源的中断请求。中断源自然优先级是一种响应优先级, 只是解决若干个中断源同时发出中断请求的优先响应顺序问题。

6.2.3 中断请求标志

51 单片机查询到某个中断源有中断请求, 由硬件设置该中断源的中断请求标志位。单片机在每个机器周期的 S5P2 期间按照中断自然优先级的顺序采样每个中断源, 检查中断请求信号并置位相应的中断请求标志位, 然后在下一个机器周期按优先级顺序查询中断请求标志位, 若查询到的中断请求标志位为 1 且满足中断响应的条件, 则进行中断响应。

51 单片机的中断请求标志位都集中在 TCON 和 SCON 两个寄存器中。

1. TCON 寄存器

定时器/计数器的控制寄存器 TCON，字节地址 88H，可位寻址，位地址为 8FH ~ 88H。TCON 寄存器各位定义如图 6-4 所示。

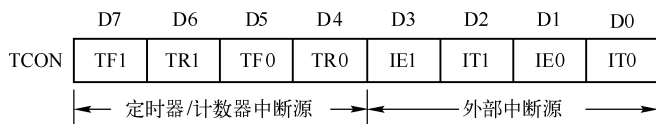


图 6-4 TCON 寄存器的位定义

其中与中断系统相关的各标志位功能如下：

TF1——T1 的溢出中断请求标志位。溢出时由硬件置 1，发出中断请求。单片机响应中断后，TF1 标志位由硬件清 0。

TF0——T0 的溢出中断请求标志位。功能与 TF1 一致，其中断源是 T0。

IT1——外部中断 1 的触发方式选择位。

IT1 = 0，电平触发方式。当 $\overline{\text{INT1}}$ 引脚（P3.3）出现低电平时，外部中断 1 的中断请求信号有效。

IT1 = 1，负跳变（边沿）触发方式。当 $\overline{\text{INT1}}$ 引脚（P3.3）出现高电平向低电平跳变的下降沿（负跳变）时，产生一次外部中断 1 的中断请求。

IE1——外部中断 1 的中断请求标志位。

如果 IT1 = 0，当外部引脚 P3.3 出现低电平时，硬件置位 IE1，并向单片机发出外部中断 1 的中断请求，单片机响应该中断请求后，IE1 标志位硬件清 0。

如果 IT1 = 1，当 $\overline{\text{INT1}}$ 引脚（P3.3）出现高电平向低电平跳变的下降沿（负跳变）时，硬件置位 IE1，并向单片机发出外部中断 1 的中断请求，单片机响应该中断请求后，IE1 标志位硬件清 0。

IT0——外部中断 0 的触发方式选择位。功能与 IT1 类似，其中断源是外部中断 0。

IE0——外部中断 0 中断请求标志位。功能与 IE1 类似，其中断源是外部中断 0。

TR0、TR1 的定义和使用方法涉及定时器/计数器的启动控制，见第 5 章定时器/计数器一章详解，此处不做阐述。

复位操作使得寄存器 TCON 为 00H，默认 2 个外部中断源为电平触发方式。

2. SCON 寄存器

串行口控制寄存器 SCON，字节地址 98H，可位寻址，位地址为 9FH ~ 98H。SCON 寄存器各位定义如图 6-5 所示。

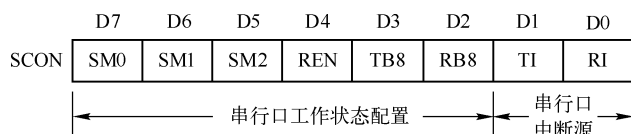


图 6-5 SCON 寄存器的位定义

其中与中断系统相关的各标志位功能如下：

TI——串行口发送完成中断请求标志位。

51 单片机串行口处于允许发送数据状态时，单片机将一个字节的数据写入串行口的数据缓冲器 SBUF，从 TXD 引脚发送数据。一帧数据发送完成后，硬件置 TI 标志位为“1”，单片机响应串行口中断后，必须在中断服务程序中编写 TI 标志位清 0 的指令（软件清 0）。

RI——串行口接收完成中断请求标志位。

51 单片机串行口处于允许接收数据状态时，当 RXD 引脚接收一帧数据存入数据缓冲器 SBUF 后，硬件置 RI 标志位为“1”，单片机响应串行口中断后，RI 标志位必须软件清 0。

SM0、SM1、SM2、REN、TB8、RB8 的定义和使用方法涉及串行口工作状态配置，在第 7 章串行通信一章详解，此处不做阐述。

复位操作使得 SCON 寄存器为 00H，TI 和 RI 中断标志位均为 0。

6.2.4 中断控制

中断系统的中断控制包括中断允许与屏蔽、中断优先级设定两个方面的问题，涉及特殊功能寄存器的中断允许寄存器 IE 和中断优先级寄存器 IP。

1. 中断允许寄存器 IE

中断允许寄存器 IE，字节地址 A8H，可位寻址，位地址为 AFH ~ A8H。IE 寄存器各位定义如图 6-6 所示。

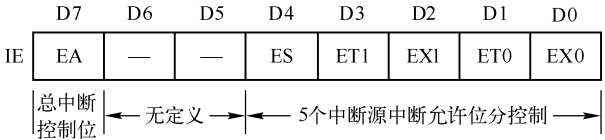


图 6-6 IE 寄存器的位定义

IE 寄存器各位的定义与功能如下：

EA——总中断允许控制位。

EA = 0：屏蔽所有的中断请求；EA = 1：开放所有的中断请求。

ES——串行口中断允许控制位。

ES = 0：屏蔽串行口中断请求；ES = 1：开放串行口中断请求。

ET1——定时器/计数器 T1 中断允许控制位。

ET1 = 0：屏蔽 T1 中断请求；ET1 = 1：开放 T1 中断请求。

EX1——外部中断 1 中断允许控制位。

EX1 = 0：屏蔽外部中断 1 中断请求。EX1 = 1：开放外部中断 1 中断请求。

ET0——定时器/计数器 T0 中断允许控制位。

ET0 = 0：屏蔽 T0 中断请求；ET0 = 1：开放 T0 中断请求。

EX0——外部中断 0 中断允许控制位

EX0 = 0：屏蔽外部中断 0 中断请求；EX0 = 1：开放外部中断 0 中断请求。

由上可见，中断允许寄存器 IE 控制中断源中断请求的开放和屏蔽。需要编程完成对 IE 的相应位置 1 或清 0 操作，进而实现某个中断源的开放与屏蔽。例如设置 EA = 0，则屏蔽所

有的中断，即使 5 个中断源的中断允许控制位都为“1”，单片机都不会响应任何中断源的中断请求；若 EA = 1，ET0 = 1，而其他 4 个中断源的中断允许控制位都为“0”，则单片机仅能够响应 T0 的中断请求。

复位操作使得 IE 寄存器为 00H，屏蔽所有的中断源。

2. 中断优先级寄存器 IP

中断优先级寄存器 IP，字节地址 B8H，可位寻址，位地址为 BFH ~ B8H。IP 寄存器各位定义如图 6-7 所示。

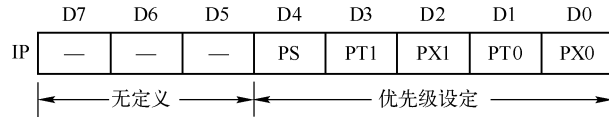


图 6-7 寄存器 IP 的位定义

各位的定义与功能如下：

PS——串行口中断优先级控制位。

PS = 0：低优先级；PS = 1：高优先级。

PT1——T1 中断优先级控制位。

PT1 = 0：低优先级；PT1 = 1：高优先级。

PX1——外部中断 1 中断优先级控制位。

PX1 = 0：低优先级；PX1 = 1：高优先级。

PT0——T0 中断优先级控制位。

PT0 = 0：低优先级；PT0 = 1：高优先级。

PX0——外部中断 0 中断优先级控制位。

PX0 = 0：低优先级；PX0 = 1：高优先级。

51 单片机对中断优先级的管理实行两级管理：高优先级与低优先级，能够实现两级中断嵌套。每个中断源的优先级均可以通过对 IP 的相应位置 1 或清 0，实现该中断源的优先级设定。两级中断嵌套就是 51 单片机处于正在执行低优先级中断服务程序时，高优先级中断事件发生，暂停低优先级中断服务程序的执行，转而执行高优先级中断服务程序，直到该中断服务程序执行完毕再返回低优先级中断服务程序执行。

关于优先级的概念，必须掌握如下 4 点：

1) 每个中断源只能被配置为高优先级或低优先级，复位操作使得 IP 寄存器为 00H，5 个中断源都被设置为同一级的低优先级中断源。

2) 低优先级事件能够被高优先级事件中断，高优先级事件不能被低优先级事件中断。

3) 同级优先级事件间不能互相中断。例如 A 和 B 中断源均被设置为高优先级，C 和 D 中断源均被设置为低优先级。A 不能中断 B 的执行，反之依然；C 不能中断 D 的执行，反之依然。

4) 高、低优先级区别于自然优先级，抢占优先级区别于响应优先级。高、低优先级由用户通过程序指令指定中断源优先级，自然优先级由 51 单片机硬件结构决定中断源优先级。高、低抢占优先级决定了中断嵌套的关系，响应优先级决定了若干个中断源同时中断申请被响应的次序。

6.3 51 单片机的中断响应

中断的处理过程包括中断响应、中断服务和中断返回三个部分。在 6.1.2 节的中断技术的主要功能特点中对中断服务和中断返回已作了描述，本节重点对中断响应作较为深入的阐述。中断源有中断请求时，单片机根据中断响应条件进行判断，满足条件后完成对中断源中断请求的应答、中止现程序的执行、保存断点及进入中断服务程序的整个过程。中断响应解决中断源的识别和中断处理前的准备工作，是由单片机中断系统的硬件装置自动完成的。

6.3.1 中断响应的条件

单片机在程序执行过程中，并不是随时都可以响应中断源的中断请求信号，必须满足以下基本条件单片机才可能响应中断请求：

- 1) 有中断源发出有效的中断请求，相应的中断请求标志位为“1”状态。
- 2) 中断允许寄存器 IE 中总中断允许控制位 $EA = 1$ ，开放所有的中断请求。
- 3) 中断请求的中断源允许控制位为“1”，即该中断源的中断请求被开放。

基本条件仅仅是必要条件，并非充要条件。而中断响应过程遇到如下情况，该中断源的中断请求响应过程将会进入中断的阻塞等待状态：

- 1) 单片机目前正在执行同级或更高优先级中断服务程序。
- 2) 当前查询的机器周期不是一条指令执行的最后一个周期，即正在执行的指令还未完成，中断请求不会得到响应，以确保当前指令的执行过程完整。
- 3) 当前正在执行 RETI 指令或读/写特殊功能寄存器 IE、IP 的指令，中断不能在指令执行结束后立即被响应，需要再执行一条其他指令，中断才会被单片机响应。

一个中断源的中断阻塞等待状态并不等于中断的撤销。阻塞仅仅是延时滞后执行，只要中断标志位有效状态存在，一定会被单片机响应进入中断处理过程。

中断响应条件满足后，中断响应过程主要完成两件事：断点的保存和转入中断服务程序。

断点的保存，即将中断处的 16 位地址（当前程序计数器 PC 的内容）压入堆栈保存，以便中断返回时按此 16 位地址返回中断前程序的指令地址继续执行。

转入中断服务程序前，单片机硬件自动生成一条长调用指令 LCALL addr16, addr16 写入 PC，使程序转向中断服务程序的入口。其中 addr16 就是表 6-1 所列各中断源的 16 位入口地址，即中断入口地址。

例如，外部中断 0 产生中断请求且 $EA = 1$ ， $EX0 = 1$ ，单片机响应外部中断 0 的中断请求时自动将断点保存，执行长调用指令：LCALL 0003H。

6.3.2 中断响应的时间

中断响应过程并非瞬间完成，需要一定的时间。对于实时控制系统，时间要求比较苛刻，中断响应时间的长与短关系到程序的稳定性和功能的可靠性。

51 单片机的中断响应时间存在最短时间，而最长时间由于情况不同各异。

最短响应时间：3 个机器周期。第 1 个机器周期执行查询中断请求标志位（此机器周期恰好为一条非 RETI 或访问 IE、IP 指令的最后一个机器周期）；第 2 个和第 3 个机器周期由

单片机执行硬件生成的长调用指令 LCALL addr16，主要完成 16 位断点地址压入堆栈和程序跳转至中断入口地址处。

最长响应时间：8 个机器周期甚至更长。8 个机器周期的响应时间发生情况比较特殊。在单片机进行中断标志位查询时，当前正在执行 RETI 指令的第 1 个机器周期（RETI 指令为双周期指令）；RETI 指令执行完毕后必须再执行另一条指令才能被响应。按此计算：RETI 指令两个机器周期，如果另一条指令恰好为执行时间为 4 机器周期的乘/除法指令，再考虑执行硬件生成的双机器周期的长调用指令 LCALL addr16，合计响应时间为 $2 + 4 + 2 = 8$ 个机器周期。

最长响应时间严格意义上是无法确定的。例如发出一个低优先级中断请求后，而单片机正在执行一个高优先级中断服务程序，此时阻塞了低优先级中断请求。低优先级中断请求的响应必须要等待执行完高优先级中断服务程序，此时高优先级中断服务程序的执行时间导致低优先级中断请求的响应时间变长。

综上所述，若 51 单片机应用系统仅使用某一个中断源的情况下，中断请求的响应时间为 3 ~ 8 个机器周期之间。

6.3.3 中断请求的撤销

单片机响应一个中断源的中断请求后，必须清除中断请求的标志位，否则，该中断源的中断服务程序执行结束返回后，如果该中断源的中断标志位仍然有效，单片机会再次响应同一中断源的中断请求。下面按照中断源的类型分别说明中断请求标志位的清除方式。

1. 定时器/计数器中断请求标志位的清除方式

定时器/计数器在其核心 +1 计数器溢出时，硬件置位中断标志位 TF0（TF1）。当中断请求被响应后，硬件自动完成该中断标志位的清除。

2. 外部中断请求标志位的清除方式

外部中断源的中断请求被响应后，硬件清除外部中断请求标志位 IE0 或 IE1。

但由于外部中断存在低电平和负跳变两种触发方式，撤销中断标志位后会根据具体情况出现不同的情况。

1) 负跳变触发方式中（ITx = 1），中断响应后硬件自动清除中断标志位，中断响应后负跳变下降沿消失，因此标志位清除有效。

2) 低电平触发方式中（ITx = 0），中断标志位的置 1 和外部中断引脚输入的低电平同步。尽管中断响应后能够自动清除中断标志位，但是如果外部中断引脚输入仍然保持低电平，导致已清除的中断标志位再次被置 1，从而会在中断返回后再次向单片机发出中断请求。因此，撤销低电平触发方式的中断标志位的关键是要可靠地撤销中断响应后外部中断引脚上的低电平。此时，可考虑软件与硬件相结合的方法实现，如图 6-8 所示。

图 6-8 中，D 触发器 CP 端为正边沿触发有效。当外部中断请求信号由高电平变为低电平请求中断时，经过反相器在 CP 输入端产生正边沿触发信号，此时 D 输入端的低电平被锁存到 Q 输出端，加载到 INT0 引脚，发出中断请求。中断响应后，需要使 P1.0 引脚输出一个负脉冲即可使

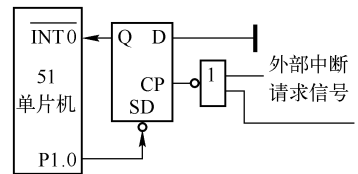


图 6-8 电平触发方式中断请求信号撤销电路

Q 端输出为高电平，撤销中断请求信号的低电平，也就清除了中断标志位 EX0。在中断服务程序开始处安排如下两条指令即可达到撤销INT0引脚的低电平。

```
CLR  P1.0 ;P1.0 引脚输出低电平,D 触发器 SD 端有效,置位 Q 端输出高电平
SETB P1.0 ;P1.0 引脚输出高电平,D 触发器 SD 端无效,Q 端输出锁存等待下次触发
```

3. 串行口中断请求标志位的清除方式

串行口具备发送和接收双重功能，其完成一帧数据发送（接收）时，硬件置位相应的中断标志位 TI（RI）。无论是 TI 置位还是 RI 置位，均会请求串行口中断。由于串行口的内部结构是发送完成和接收完成共享同一个串行口中断，中断系统响应串行口中断后无法直接判断是由 TI 还是由 RI 发出的中断请求，因此，需要在中断服务程序中编写判断是发送中断请求标志位 TI 有效还是接收中断请求标志位 RI 有效，根据判断结果再对中断标志位清 0。所以，串行口中断标志位 TI 和 RI 需要软件方法清除。

6.4 中断系统的编程

中断技术有效地提高了系统的实时性，不同于“查询方法”的程序设计。中断服务程序实质上就是一个子程序，编程方法与子程序基本相同，本节通过几个举例来说明中断系统的编程方法和程序结构，有利于掌握中断系统的概念、原理和编程设计。

6.4.1 中断服务程序编程的基本步骤

中断处理的一个基本过程如图 6-9 所示，整个中断处理过程是软件、硬件的协同处理过程。设计者需要编写中断初始化程序和中断服务程序。

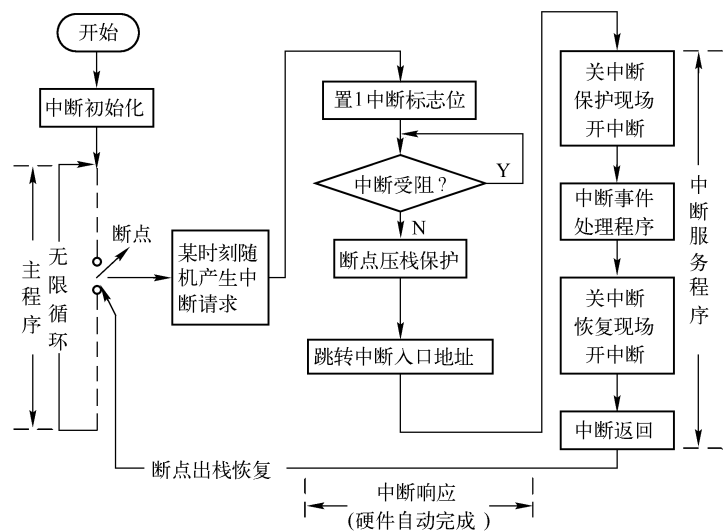


图 6-9 中断处理的基本过程

(1) 中断初始化程序

对于编程者设置的中断源，必须允许该中断源中断，即总中断 EA = 1 和该中断源的中断允许位置 1；中断优先级设置是可选项，根据实际情况配置；对于外部中断源需要设置中

断请求触发方式，例如，外部中断 0 设置为边沿触发方式，则编程设置 $IT0 = 1$ 。

(2) 中断服务程序

包括现场保护、中断事件处理、恢复现场和中断返回等。

1) 中断事件处理程序：编程完成中断事件所要求的处理功能，是中断服务程序的核心。编程者要特别注意中断事件处理程序中所使用的寄存器或存储单元是否与整个应用程序中其他程序段相冲突。对于中断事件处理程序中应用的寄存器或存储单元，如果其他程序段也在使用，为了避免执行中断事件处理程序使得其他程序段的这些寄存器或存储单元的状态、数据丢失，从而引发中断结束后返回断点位置执行原程序的错误发生，需对这些寄存器或存储单元进行现场保护和恢复现场。

2) 保护现场：对中断事件处理程序中使用的寄存器和存储单元进行压栈处理，中断服务程序首先关闭总中断，然后应用 PUSH 指令对使用的寄存器和存储单元的内容压栈，最后开放总中断。

3) 恢复现场：对中断事件处理程序中被压栈的寄存器和存储单元进行出栈操作。首先关闭总中断，然后应用 POP 指令将压栈的寄存器和存储单元的内容出栈，最后开放总中断。

4) 中断返回：中断服务程序的最后一条指令必须是中断返回指令 RETI。该指令完成断点地址出栈传输给 PC，中断返回后，单片机从断点地址处继续执行程序。

单片机响应中断源的中断请求时，断点地址压栈，即程序计数器的 PC 值压栈，该中断源的中断入口地址→PC，中断返回时断点地址出栈→PC，所以，特别注意编写中断服务程序中的 PUSH 指令和 POP 指令必须成对地对应出现，否则程序的执行就会出现混乱。

6.4.2 外部中断源的编程

例 6-1 电路如图 6-10 所示，外部报警信号由“1”→“0”时，控制警示发光二极管亮，单片机复位后警示发光二极管灭。

分析：由图 6-10 电路可知，P1.0 输出高电平，警示发光二极管 D_1 灭；P1.0 输出低电平，警示发光二极管 D_1 亮，发出报警信号。编写外部中断 0 中断服务程序时，应用 CLR P1.0 指令使 P1.0 输出低电平。外部中断 0 中断服务程序只有在外部报警信号事件发生的条件下才会执行。

为了加深对中断响应和保护现场、恢复现场过程中程序计数器 PC、堆栈指针 SP 和堆栈单元内容的变化过程的理解，在程序中增加了一条用于测试的指令，将累加器 A 作为测试现场保护功能的单元，其中断过程分析如图 6-11 所示。

设置 $\overline{INT0}$ 工作在边沿触发方式。参考程序如下：

```
ORG    0000H
LJMP   MAIN

ORG    0003H      ;  $\overline{INT0}$  的中断入口地址
LJMP   LEDS       ; 跳转到中断服务程序 LEDS

ORG    0030H
```

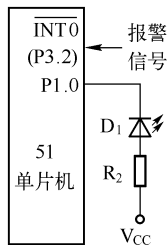


图 6-10 例 6-1 电路图

```

MAIN: MOV    SP, #5FH      ;初始化堆栈指针寄存器
      SETB   P1.0          ;初始化设置 P1.0 = 1, 外部发光二极管为灭
      SETB   IT0           ;设置外部中断 0 为负跳变触发方式, 即下降沿触发
      SETB   EX0           ;外部中断 0 中断允许
      SETB   EA            ;总中断允许
LOOP: INC     A             ;测试现场保护功能, 能够用单步执行程序的方法进行观测
      LJMP   LOOP          ;无限循环, 等待中断
LEDS: CLR     EA           ;外部中断 0 的中断服务程序, 保护现场前关闭中断
      PUSH  ACC            ;保护现场 ACC
      PUSH  0D0H          ;保护现场 PSW
      SETB   EA            ;保护现场后打开中断
      CLR   P1.0          ;警示发光二极管 D1 亮, 发出报警信号
      CLR   A              ;测试现场保护, 此处 A 的变化不影响主程序中
                          ;INC A 指令的正确执行
      CLR   EA            ;恢复现场前关闭中断
      POP   0D0H          ;恢复现场 PSW
      POP   ACC            ;恢复现场 ACC
      SETB   EA            ;恢复现场后打开中断
      RETI
      END

```

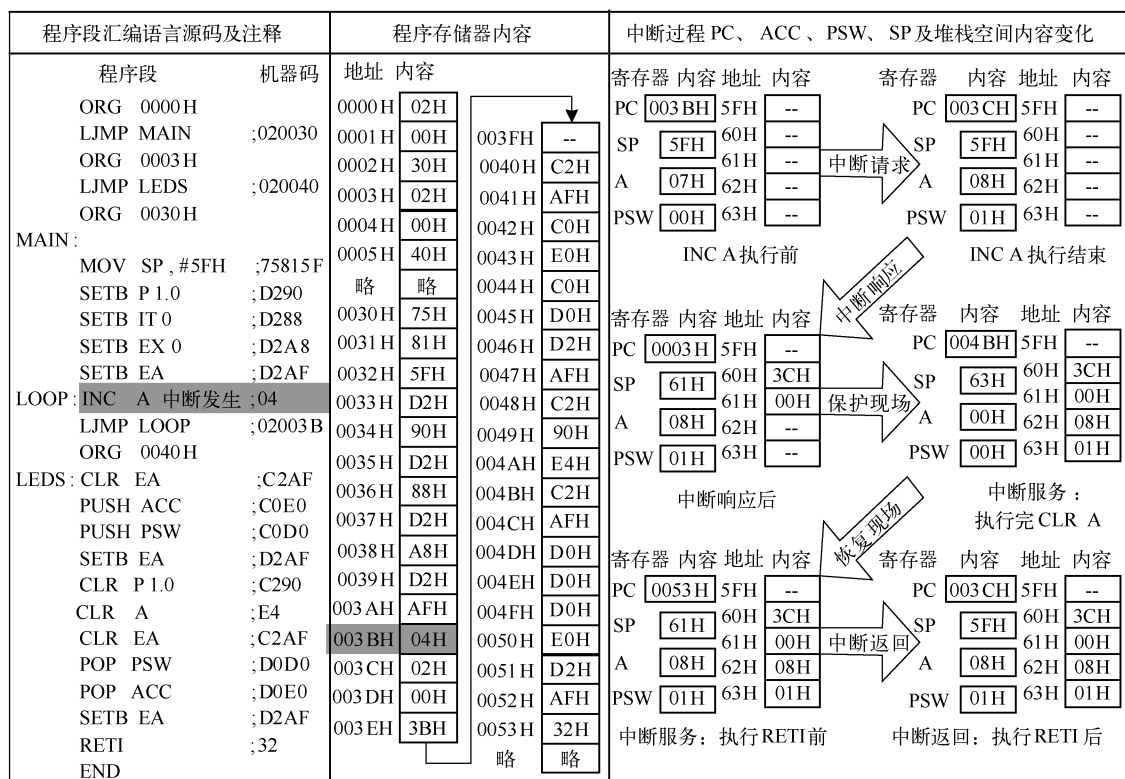


图 6-11 中断过程分析

图 6-11 中，假设外部中断 0 外部报警信号引发的中断发生在单片机正在执行 INC A 指令的过程中，单片机执行完 INC A 指令后中断响应。图 6-11 中列出了 PC、A、PSW、SP 及堆栈空间各单元内容在中断响应、执行中断服务程序和中断返回过程中的变化过程，便于理解中断响应（断点压栈）和中断返回（断点出栈）、中断服务程序的现场保护（保护内容的压栈）和恢复现场（保护内容的出栈）的基本概念。

例 6-2 外部中断源的扩展实现多路输入监控的电路如图 6-12 所示，通过外部中断 0 实现对 CH1、CH2、CH3 三条输入通道的监控。三条通道分别由三台设备提供信号。若设备工作正常，相应通道为高电平；若其中一台设备工作出错，相应的 CHx 通道被拉为低电平。当任何一台设备无法正常工作，需点亮相应的发光二极管提示监控人员。

分析：图 6-12 中，1#设备对应 CH1 通道，故障报警灯为 D₁；2#设备对应 CH2 通道，故障报警灯为 D₂；3#设备对应 CH3 通道，故障报警灯为 D₃。

由于 51 单片机仅有两个外部中断源，因此，应用 2 个外部中断实现三路监控是不可行的。因此，变换思路，通过硬件扩展外部中断源，采用软件程序识别中断请求的外部中断源的办法，巧妙地解决硬件资源不足的问题。设计硬件扩展外部中断源电路如图 6-12 所示，图 6-12 中，当 CH1、CH2、CH3 通道都为高电平时，与门输出为高电平，触发外部中断 0。此时，无法识别是哪一路通道引发的中断，因此，必须在外部中断 0 的中断服务程序中编写识别故障来源的程序进行判别，确定故障来源。

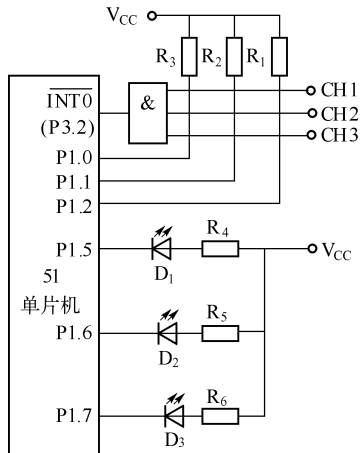


图 6-12 例 6-2 电路图

根据题意要求结合硬件电路，设定 INT0 工作在边沿触发方式，单片机系统复位后三个发光二极管都为灭的状态。

参考程序如下：

```

CH1  BIT  P1.0
CH2  BIT  P1.1
CH3  BIT  P1.2
ERR1 BIT  P1.5
ERR2 BIT  P1.6
ERR3 BIT  P1.7
ORG  0000H
LJMP MAIN
ORG  0003H          ;外部中断 0 入口地址
LJMP LEDS          ;跳转到外部中断 0 的中断服务程序 LEDS
ORG  0030H
MAIN: MOV  SP,#5FH      ;初始化堆栈指针寄存器
      MOV  P1,#0FFH     ;初始化 P1 口,外部发光二极管为灭的状态
      SETB ITO          ;设置外部中断 0 为边沿触发
      SETB EX0          ;外部中断 0 中断允许

```



```

SETB EA          ;总中断允许
SJMP $           ;原地踏步,等待中断
LEDS: JB CH1,NEXT1 ;CH1 通道电平为高,无故障,查询下一路
      CLR ERR1    ;CH1 通道故障,点亮 D1
NEXT1: JB CH2,NEXT2 ;CH2 通道电平为高,无故障,查询下一路
      CLR ERR2    ;CH2 通道故障,点亮 D2
NEXT2: JB CH3,NEXT3 ;CH3 通道电平为高,无故障,查询下一路
      CLR ERR3    ;CH3 通道故障,点亮 D3
NEXT3: RETI
END

```

如果出现有发光二极管被点亮，则只有复位单片机应用系统才能够使发光二极管熄灭。

6.4.3 定时器/计数器中断方法的编程

在实时控制、检测、数据和信号处理系统中，对于数据的定时采样、定时通信和信号脉冲个数计数等，采用定时器/计数器中断编程的方法极其普遍。本节应用三个例题介绍中断服务程序的编写方法。注意中断服务程序的编写方法与子程序编写方法的异同之处，并与第5章相应的例题进行对比，掌握中断与查询编程方法的区别。

例 6-3 某 51 单片机应用系统，系统晶振的时钟频率采用 12 MHz，应用中断技术编写在 P1.0 引脚输出一个周期为 10 ms 方波信号的处理。

分析：该例题与例 5-1 相同，仍然选用定时器 T1，但采用中断技术。例 5-1 解题过程的计算定时初值 X 与本例题完全相同；初始化定时器 T1 的步骤也与本例题类似，但需要增加 T1 开中断的初始化指令。

根据例 5-1，计算已得到定时初值 X = EC78H，初始化设置 TMOD = 10H、TCON = 00H。

按照定时中断技术，程序流程图如图 6-13 所示。从图 6-13b 可以看出，中断服务程序流程图与子程序流程图的绘制方法类似，一是不与主程序流程图混在一起绘制，而是单独绘制；二是最后一条指令是返回指令。两者之间的区别

主要在于：一是中断服务程序的返回指令是中断返回指令 RETI，子程序的返回指令是 RET；二是子程序的调用有专门的子程序调用指令，能够在程序中显现出来；但中断服务程序的调用没有调用指令显现出来，而是单片机响应已开中断的中断源引发的中断请求自动执行调用；三是中断服务程序的初始化是在主程序的初始化程序模块中完成，通常单片机开机之后只执行一次；而子程序的初始化通常是在调用子程序之前的程序段进行初始化，而且调用一次子程序之前一般都会执行一次子程序的初始化程序段。

由于本例简单，中断服务程序中没有应用寄存器和存储器的存储单元，所以在中断服务程序中没有编写保护现场和恢复现场的压栈和出栈指令。

```

ORG 0000H
LJMP MAIN

```

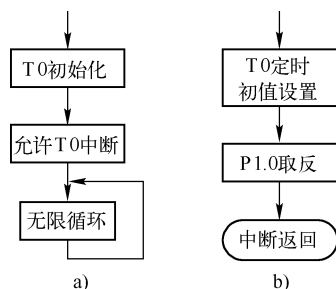


图 6-13 例 6-3 程序流程图
a) 主程序 b) 中断服务程序

```

                ORG    001BH          ;T1 的中断入口地址
                LJMP   TIMER1        ;无条件转移指令,跳转到中断服务程序 TIMER1
                ORG    0030H
MAIN:  MOV     SP,#5FH
        MOV     TMOD,#10H
        MOV     TCON,#00H
        MOV     TL1,#78H
        MOV     TH1,#0ECH
        SETB    ET1                 ;T1 中断允许
        SETB    EA                 ;开总中断
        SETB    TR1                 ;启动 T1 定时
        SJMP    $                  ;主程序无限循环,等待中断
TIMER1: MOV     TL1,#78H            ;程序指令重载定时初值低 8 位
        MOV     TH1,#0ECH          ;程序指令重载定时初值高 8 位
        CPL     P1.0               ;已产生了 5 ms 的定时,取反 P1.0
        RETI
        END

```

例 6-4 某 51 单片机系统,系统晶振的时钟频率采用 6 MHz,应用中断技术编写 P1.0 引脚输出周期为 4 s 方波的处理程序。

分析: 6 MHz 时钟频率的计数脉冲周期为 $2\mu\text{s}$, 采用 16 位定时器 T0 定时 100 ms 溢出中断, 8 位计数器 T1 计数 10 次溢出中断的编程方法, 电路如图 6-14 所示。图 6-14 中, T0 定时 100 ms 溢出使得 P1.1 取反一次, P1.1 端口输出周期为 200 ms 的方波, T1 每 200 ms 计数一次, 计数 10 次为 2 s, 使得 P1.0 取反一次, P1.0 端口输出周期为 4 s 的方波。

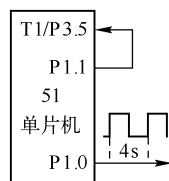


图 6-14 例 6-4 电路图

(1) T0 定时初值计算

选用 T0 工作于方式 1 定时 100 ms, 按照第 5 章式 (5-3), 得

$$X = 65536 - \frac{t \times f_{\text{osc}}}{12} = 65536 - \frac{100 \times 10^{-3} \times 6 \times 10^6}{12} = 65536 - 50000 = 15536 = 3CB0\text{H}$$

(2) T1 计数初值计算

选用 T1 工作于方式 2 计数 10 次, 按照式 (5-2), 得

$$X = 2^8 - M = 256 - 10 = 246 = \text{F6H}$$

参考程序如下:

```

                ORG    0000H
                LJMP   MAIN
                ORG    000BH          ;T0 的中断入口地址
                LJMP   TIMER0        ;跳转到中断服务程序 TIMER0
                ORG    001BH          ;T1 的中断入口地址
                LJMP   COUNT1        ;跳转到中断服务程序 COUNT1
                ORG    0030H

```

```

MAIN:  MOV    SP,#5FH          ;初始化堆栈指针寄存器
        MOV    TMOD,#61H       ;T0 为定时器工作方式 1,T1 为计数器工作方式 2
        MOV    TCON,#00H       ;初始化 T0 和 T1 相关标志位
        MOV    TLO,#0B0H       ;初始化 T0 定时初值低 8 位
        MOV    TH0,#3CH        ;初始化 T0 定时初值高 8 位
        MOV    TL1,#0F6H       ;T1 的 8 位计数初值
        MOV    TH1,#F6H        ;T1 为计数器工作方式 2,(TH1) = (TL1)
        CLR     P1.1           ;初始化 P1.1 引脚的输出电平
        SETB    ET0            ;T0 中断允许
        SETB    ET1            ;T1 中断允许
        SETB    EA            ;开总中断
        SETB    TR0            ;启动 T0 的定时
        SETB    TR1            ;启动 T1 的计数
        SJMP    $              ;无限循环,等待中断
TIMER0: MOV    TLO,#0B0H       ;T0 中断服务程序。重装载定时初值低 8 位
        MOV    TH0,#3CH        ;重装载定时初值高 8 位
        CPL     P1.1           ;已产生了 100 ms 的定时,取反 P1.1
        RETI
COUNT1: CPL    P1.0           ;已计数 10 次,取反 P1.0,方波半周期  $200\text{ ms} \times 10 = 2\text{ s}$ 
        RETI
        END

```

例 6-5 使用定时器/计数器的定时中断技术编写跑马灯的程序,实现连接在 P1 口上的 8 个 LED 灯依次亮灭。每次只能一个 LED 灯状态变化,且按 P1.7 ~ P1.0 的顺序依次亮灭,每次变化时间间隔 1 s,51 单片机晶振时钟频率为 12 MHz。

分析: 设 P1 口输出为低电平时 LED 灯亮,应用 T0 作为定时器。本例题与习题 5-8 比较有三个不同点: 一是 51 单片机晶振时钟频率不同;二是本例题要求每次变化时间间隔为 1 s,而习题 5-8 为 100 ms;三是本例题应用定时中断方式,而习题 5-8 采用定时查询方式。

1) 12 MHz 晶振时钟频率的定时计数时间周期为 $1\text{ }\mu\text{s}$,定时器工作于方式 1 的最长定时时间为 65.536 ms,采用类似于例 5-2 的定时 + 计数的方法实现 1 s 的定时时间,设置定时时间为 50 ms,根据例 5-2 的计算,定时初值 $X = 3\text{CB}0\text{H}$ 。

2) 设置计数变量计数 20 次,得到 $20 \times 50\text{ ms} = 1000\text{ ms} = 1\text{ s}$ 的定时时间。

3) 思路: 定时 50 ms 中断一次,执行 T0 中断服务程序一次,计数变量 + 1,再判断计数变量是否等于 20,不等于则退出 T0 中断服务程序;等于说明 1 s 定时时间到,需要执行与习题 5-8 相同的 8 个 LED 依次亮灭的核心功能指令,即累加器 A 不带进位的右移指令: RR A。

```

        ORG    0000H
        LJMP   MAIN
        ORG    000BH          ;T0 的中断入口地址
        LJMP   TIMER0        ;跳转到中断服务程序 TIMER0
        ORG    0030H
MAIN:    MOV    SP,#5FH       ;初始化堆栈指针寄存器

```

```

MOV    TMOD,#01H    ;T0 为定时器工作方式 1
MOV    TCON,#00H    ;初始化 T0 相关标志位
MOV    TLO,#0B0H    ;初始化 T0 定时初值低 8 位
MOV    TH0,#3CH      ;初始化 T0 定时初值高 8 位
MOV    R2,#00H       ;初始化计数变量(R2)=00H
MOV    30H,#7FH,
MOV    P1,30H        ;初始 P1.7 输出低电平,该 LED 灯亮,其他 LED 灯灭
SETB   ETO           ;T0 中断允许
SETB   EA            ;开总中断
SETB   TR0           ;启动 T0 的定时
SJMP   $             ;无限循环,等待中断
TIMER0: PUSH ACC      ;T0 中断服务程序,每 50 ms 执行一次
MOV    TLO,#0B0H     ;重装定时初值低 8 位
MOV    TH0,#3CH      ;重装定时初值高 8 位
INC     R2            ;计数变量(R2)+1→R2
CJNE   R2,#14H, WAIT ;1 s 定时到,顺序执行,否则退出定时中断服务程序
MOV    R2,#00H       ;清计数变量 R2
MOV    A,30H         ;只有累加器 A 有循环移位指令
RR      A            ;累加器 A 的内容右移一位
MOV    P1,A          ;输出控制 LED 灯的亮灭状态
MOV    30H,A         ;保存 LED 灯亮灭的状态
WAIT:  POP  ACC
      RETI
      END

```

注意中断服务程序的保护现场和恢复现场的编程方法，PUSH 和 POP 指令必须成对出现。

习题

6-1 51 单片机最多可实现 () 级的中断嵌套。

6-2 若设计的 51 单片机应用系统只使用一个中断源，则中断响应时间最短是 () 个机器周期，最长是 () 个机器周期；若有两个以上的中断源，为什么说最长响应时间是无法确定的？

6-3 若软件设置 (IP) = 00011100，高优先级的中断源是 ()，低优先级的中断源是 ()。

6-4 什么是中断，中断系统一般应具备哪些功能？

6-5 在编写中断服务程序时，为什么需要编写保护现场和恢复现场的语句？PUSH 和 POP 指令为什么必须成对地使用？

6-6 分析保护现场和恢复现场与保护断点之间的异同。

6-7 总结中断入口地址的特点。

6-8 简述各个中断源的标志位随着中断响应后撤销的方法。

6-9 分析中断返回指令 RETI 和子程序返回指令 RET 之间的异同。

6-10 中断服务程序和子程序的主要区别是什么？

6-11 一个完整的中断处理的基本过程包括哪些内容？

6-12 将例 6-1 的定时器中断编程方法与例 5-1 的定时器查询编程方法进行比较，指出两种编程方法之间的异同点以及两种情况下程序执行的异同点。

6-13 试编写一段对中断系统进行初始化的程序，使之允许 $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 、T0 和串行口中断，且使串行口中断为高优先级中断。

6-14 试用定时器中断方式设计一个程序，使发光二极管每秒时间内亮 400 ms，灭 600 ms。设 51 单片机的振荡频率为 6 MHz。

6-15 已知 51 单片机应用系统采用 6 MHz 晶振，应用 T0 定时中断，在 P1.0 输出一个高电平宽度为 100 ms、低电平宽度 50 ms 的方波信号。

6-16 应用定时器定时中断编程的方法编写程序：实现连接在 P1 口上的 8 个 LED 依次亮灭。每次只能一个灯状态变化，且按由 P1.0 ~ P1.7 的顺序依次熄灭。每次变化时间间隔 500 ms。

6-17 对于习题 6-16，说明如何修改 LED 灯亮灭变化的间隔时间，采用何种方法最为便利？提示：可以应用伪指令。

第 7 章 串行通信及应用

串行通信是信息交换的一种基本通信方式。绝大多数微处理器均具备不同数量的串行通信功能部件，目前应用较多的串行通信包括 I²C、SPI、USB 和 RS-232 等。本章主要以 51 单片机的串口功能部件为主要学习内容，介绍其基本概念、体系结构、工作模式及程序设计。

7.1 数据通信的基本概念

7.1.1 根据通信方式分类

通信方式的基本分类包括以下两种：并行通信和串行通信。

并行通信：数据以组的方式在多条线路上同时传输的通信方式，不仅仅局限于微处理器芯片间的信息交换，芯片内部也存在大量的并行通信。一般并行通信按照字节长组织传输，例如 51 单片机的内部总线和外部总线的并行通信一般以 1 个字节为宽度进行。并行通信时数据同时出现在并行接口。如图 7-1a 所示，微处理器 A 的 75H（二进制 01110101B）字节数据同时出现在 D7 ~ D0 并行通信线供微处理器 B 读取。并行通信的优点是同一时刻可以传输多位数据，传输效率高，不需要同步。但存在并行线路复杂，长距离传输成本高等缺点。

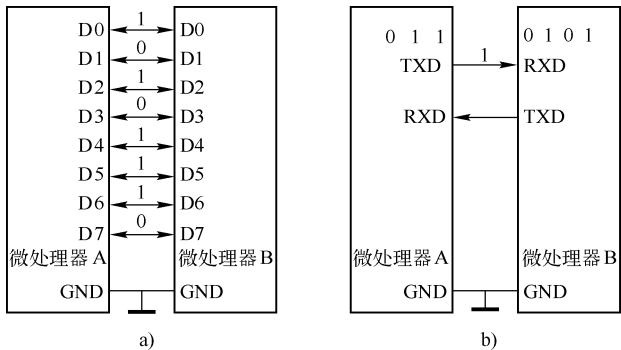


图 7-1 通信的基本方式
a) 并行通信 b) 串行通信

串行通信：数据以位（bit，比特）方式按顺序分时进行收、发数据的通信方式。一般串行通信以位组织传输，并行数据发送前采用并入串出的移位寄存器，将微处理器内部待发送的并行数据转换为串行数据进行发送；串行数据接收后采用串入并出的移位寄存器，将串行数据转换为并行数据在微处理器内部进行处理。如图 7-1b 所示，微处理器 A 的 75H（二进制 01110101B）字节数据进行并/串转换后，通过 TXD 发送通信线以位的形式逐位发送给微处理器 B。串行通信的优点是通信线路简单，仅使用一对（或单条、多条）信号线就可以实现通信，成本相对较低。但相对并行通信而言存在传输效率低，传输过程需要同步等缺点。

7.1.2 根据同步方式分类

串行通信传输过程中，不可避免地存在同步问题。发送端将数据以位的形式不断发送，接收端如何从这一串连续的位信息流中正确地识别通信的开始与结束，这就归属为同步问题。串行通信的同步概念十分重要，如果同步失败，数据接收也就是错误的。

根据串行数据传输过程中的同步方式，串行通信可分为同步通信和异步通信两类。

1. 同步通信

同步通信是一种面向比特（位）同步的通信技术。掌握同步通信，重点理解“同步”的概念。通信双方在数据传输过程中，一起开始，一起结束，双方同步的关键就是发送或接收的时钟同步，也就是同步通信的发送端和接收端采用同频同相的同步时钟信号，保证发送方以某个固定的频率不间断地发送数据，接收方也以此固定的频率不间断地接收数据。这个固定频率的时钟通常采用同步时钟线或同一个时钟源。

同步通信一般以信息块为传输单位，因此又称为区块传输。为了区分开始和结束，同步通信中又将若干信息块组合起来，作为一帧数据进行传输。所以，同步通信传输数据帧由同步字符块、数据块、校验字符块和结束字符块等部分组成。

2. 异步通信

异步通信是一种面向一个字符帧（也称为数据帧）传输的通信技术。掌握异步通信，重点理解“异步”的概念。异步通信在发送字符时，发送端发送的每个字符帧之间的时间间隔可以是任意的，但接收端必须时刻做好接收的准备。异步通信中，在每一帧的数据前后设置控制字符，通信双方的时钟可以彼此独立，但发送数据和接收数据的传输速率必须一致。当然，严格的一致是很难做到的，只要双方保持传输速率的误差不超过一定额度，即可保证传输的正确性。

异步通信中，字符帧具备一定的格式要求，发送或接收一个数据也是以这种帧格式的形式进行，异步通信的字符帧一般包括起始位、数据位、奇偶校验位和停止位4个部分，其格式如图7-2所示。

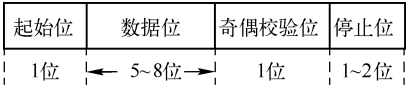


图 7-2 异步通信中字符帧格式

- 1) 起始位：字符帧的开始，逻辑“0”低电平有效，1位长度，表示本字符帧的开始。
- 2) 数据位：紧跟起始位之后，根据实际需求，可取5位、6位、7位或8位，其中低位在前，高位在后。一般情况下，字符帧采用8位数据位。
- 3) 奇偶校验位：紧跟数据位之后，1位长度，可选用奇校验或偶检验，由用户根据实际需要决定。在干扰源少、传输数据短等实际情况下，本位也可以省略。
- 4) 停止位：紧跟奇偶校验位之后，表示结束。逻辑“1”高电平有效。停止位根据约定，可选择1位、1.5位或2位长度。

异步通信传输过程均以字符帧为单位，而字符帧与字符帧之间，可以有空闲位，也可以无空闲位，即帧与帧之间可以存在时间间隔，也可以没有时间间隔。

异步通信由于一帧数据中控制信息位占比较大，相对开销较大，传输效率较低，但由于其帧与帧之间的时间间隔任意，因此灵活性好。

例 7-1 采用异步通信方式连续发送字节数据“25H”（二进制 00100101B），规定1位起始位，8位数据位，1位奇校验位，1位停止位，组成11位的字符帧，无空闲位。绘制字符帧传送波形图。

因为采用奇校验，而待发送字节数据 25H 中“1”的个数为奇数，所以此时奇偶校验位为“0”。字符帧传送波形图如图 7-3 所示。

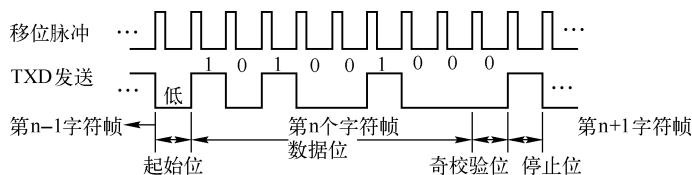


图 7-3 异步通信发送 25H 波形图

7.1.3 根据数据传输方向分类

串行通信的目的就是实现信息的交换和数据的传送，因此必须发生在发送端和接收端之间。按照数据传输的方向，串行通信可分为单工通信、半双工通信和全双工通信。

1. 单工通信

单工通信实现数据的单向传输，即数据通过单向通道，仅能实现由发送端向接收端的传输，并且始终保持单一方向传送。单工通信类似于现实生活中的单向单车道。车辆只能从车道的一端驶入，从另外一端驶出，不允许反向行驶。单工通信的示意图如图 7-4 所示。

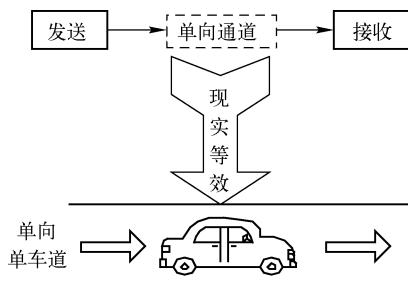


图 7-4 单工通信

2. 半双工通信

半双工通信实现数据的分时双向传输，即数据通过双向通道，满足通信双方发送端向接收端的数据传输，即在任一时刻，只能够传输一个方向的数据。半双工通信类似于现实生活中的双向单车道。车辆可以从车道的任意一端驶入，从另外一端驶出，但是同一时刻，只允许单方向行驶。对向车辆必须等待车道内车辆驶离后才能进入车道行驶。半双工通信的示意图如图 7-5 所示。

3. 全双工通信

全双工通信实现数据的双向同时传输，即数据通过双向通道，满足双方发送端向接收端的数据传输要求，实现真正意义上的两个方向的数据同时传送。从本质上看，全双工通信就是两个单工通信的组合物。全双工通信类似于现实生活中的双向双车道。对向行驶的车辆能够同时行驶在各自方向的车道上。全双工通信的示意图如图 7-6 所示。

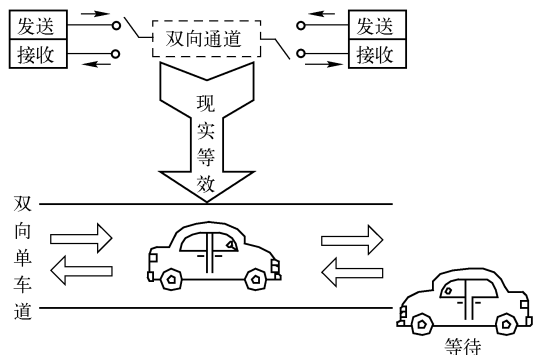


图 7-5 半双工通信

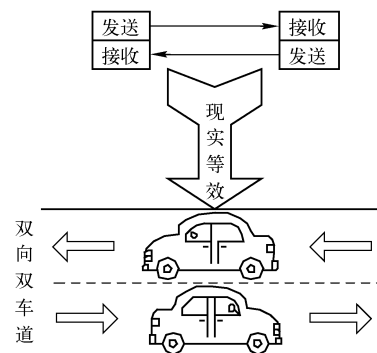


图 7-6 全双工通信

7.2 51 单片机串行口

51 单片机内部有一个可编程的全双工串行通信接口 UART，可同时进行数据的发送和接收。单片机的 TXD 端作为串行数据发送端，RXD 端作为串行数据接收端。

7.2.1 串行口的结构

串行口的内部结构框图如图 7-7 所示。

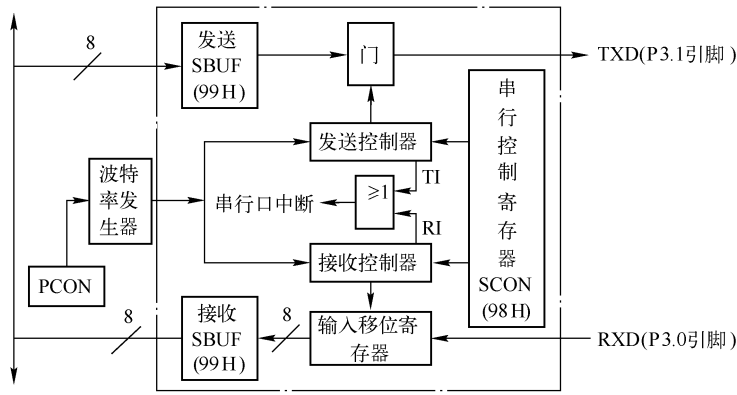


图 7-7 串行口的内部结构

从图 7-7 可以看出：

- 1) 发送部分与接收部分独立控制，分开操作。
- 2) 发送中断标志位 TI 与接收中断标志位 RI 单独产生，但共用一个中断申请。
- 3) 需要依赖于额外的波特率发生器。一般由单片机内部的定时器 T1 产生。
- 4) 具备两个在物理上相互独立的数据缓冲器：发送 SBUF 和接收 SBUF，但是两者的特殊功能寄存器字节地址相同。
- 5) 发送部分将并行数据转换成串行数据通过 TXD (P3.1) 引脚发送；接收部分通过 RXD (P3.0) 引脚接收串行数据，并将串行数据转换为并行数据进行处理。
- 6) 工作状态受 SCON 和 PCON 两个寄存器的配置决定。

7.2.2 串行口控制寄存器 SCON

串行口控制寄存器 SCON 的字节地址为 98H，可以位寻址，决定串行口具体的工作方式。复位后 SCON 所有位均清 0。SCON 的各位定义如图 7-8 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H

图 7-8 TMOD 寄存器定义

其各位定义如下：

(1) SM0 和 SM1——串行口工作方式选择位

两个选择位的编码对应四种工作方式，其编码及对应工作方式选择见表 7-1。

表 7-1 串行口工作方式选择位配置

SM0	SM1	工 作 方 式	特 点	波 特 率
0	0	方式 0	同步移位寄存器	固定 ($f_{osc}/12$)
0	1	方式 1	10 位异步收发通信	可变 (波特率发生器)
1	0	方式 2	11 位异步收发通信	固定 ($f_{osc}/32$ 或 $f_{osc}/64$)
1	1	方式 3	11 位异步收发通信	可变 (波特率发生器)

(2) SM2——多机通信控制位

方式 0 时，SM2 必须配置为 0。

方式 1 时，发送时设置 SM2 无意义；接收时一般设置 $SM2 = 0$ ，此时，不考虑接收到的停止位电平状态。如果 $SM2 = 1$ ，则只有收到有效的停止位时才能置位 $RI = 1$ ，请求中断。

方式 2 和方式 3 时，SM2 的配置在接收状态下起重要作用。

当 $SM2 = 1$ 时，只有接收到第 9 位数据 (RB8) 为 1 时，才可以将已接收的 8 位数据送入 SBUF，并置位 $RI = 1$ ，产生中断请求，等待接收完成的处理。若接收到的第 9 位数据 (RB8) 为 0 时，则接收的 8 位数据不会送入 SBUF，将被丢弃，也不会置位 RI 。

当 $SM2 = 0$ 时，不考虑接收到的第 9 位数据 (RB8) 状态，直接将已接收的 8 位数据送入 SBUF，并置位 $RI = 1$ ，产生中断请求，等待接收完成的处理。

SM2 被定义为多机通信控制位，主要是考虑多机通信协议的规定。多机通信的使用方法参看 7.5.3 节。

(3) REN——允许接收控制位

$REN = 1$ 时，允许串行口接收数据。

$REN = 0$ 时，禁止串行口接收数据。

(4) TB8——发送的第 9 位数据

方式 0 和方式 1 中不使用 TB8。

方式 2 和方式 3 中的 TB8 由软件置 1 或清 0。发送的字符帧中，TB8 即发送的第 9 位数据。通信过程中，该位既可以作为奇偶校验位使用，也可以在多机通信中作为声明地址帧/数据帧使用。

(5) RB8——接收到的第 9 位数据

方式 0 时，RB8 不使用。

方式 1 时，如果 $SM2 = 0$ ，则 RB8 存放的是已接收的停止位。

方式 2 和方式 3 时，由接收到的第 9 位来决定 RB8 是“1”还是“0”。

(6) TI——发送中断标志位

方式 0 时，串行发送完第 8 位数据时 TI 由硬件置 1。

方式 1、2、3 时，发送停止位开始时由硬件置 1。

由于串行口中断标志位的特点是共享一个中断入口地址，所以 TI 置 1 后必须等待软件查询判断是否是 TI 置位，因此不由硬件自动清除，TI 必须通过软件清 0。

TI 清 0 后，直到下次调用 SBUF 写入操作，重启发送完成之后才能再次置位。

(7) RI——接收中断标志位

方式 0 时，串行接收完第 8 位数据时 RI 由硬件置 1。

方式 1、2、3 时，接收停止位中间时刻由硬件置 1。

与 TI 一样，RI 同样不由硬件自动清除，必须通过软件清 0。

7.2.3 电源控制寄存器 PCON

电源控制寄存器 PCON 的字节地址为 87H，不允许位寻址。各位定义如图 7-9 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
PCON	SMOD	—	—	—	GF1	GF0	FD	IDL

图 7-9 PCON 寄存器定义

PCON 寄存器中与串行口工作相关的部分仅包括 SMOD 位，其他的诸如 GF1、GF0、FD、IDL 位定义及使用与电源控制相关，此处不做阐述。

SMOD——波特率倍增位

在方式 0 中，波特率固定模式，与 SMOD 无关。

在方式 1、2、3 中，串行通信的波特率受 SMOD 影响，与 2^{SMOD} 成正比关系。

SMOD = 0：串行通信波特率在设定的基础上不倍增。

SMOD = 1：波特率在设定的基础上提高一倍。

7.3 串行口工作方式

串行口根据实际需要，由 SCON 寄存器中的 SM0 和 SM1 组合配置为 4 种工作方式，分别是方式 0、方式 1、方式 2 和方式 3。

7.3.1 方式 0

当 SM0 SM1 = 00 时，串行口配置为工作方式 0。该方式为同步移位寄存器输入/输出方式。这种方式下，一般不用作单片机间的串行通信，而用作外部扩展移位寄存器实现串并转换，达到扩展并行 I/O 口的目的。

方式 0 的一帧数据包括 8 位信息，没有起始位与停止位，按照先发送/接收低位，后发送/接收高位的顺序通信，波特率固定为 $f_{\text{osc}}/12$ 。方式 0 的帧格式如图 7-10 所示。

...	D0	D1	D2	D3	D4	D5	D6	D7	...
-----	----	----	----	----	----	----	----	----	-----

图 7-10 方式 0 帧格式

1. 方式 0 发送

SM0 SM1 = 00，SM2 = 0，TI = 0，执行指令 MOV SBUF, A，单片机的 TXD 引脚输出移位脉冲，RXD 引脚输出串行数据。串行口将 SBUF 内数据（8 位）以固定的波特率（ $f_{\text{osc}}/12$ ）按位从 RXD 引脚串行输出，低位在先，高位在后，同时 TXD 引脚输出移位脉冲供同步使用。8 位数据发送完成置位发送中断标志位 TI。方式 0 的发送时序如图 7-11 所示。

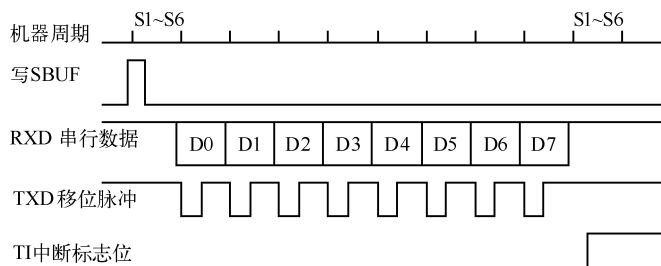


图 7-11 方式 0 发送时序图

2. 方式 0 接收

SM0 SM1 = 00, SM2 = 0, RI = 0, REN = 1 (串行口允许接收), 单片机的 TXD 引脚输出移位脉冲, RXD 引脚输入串行数据。串行口以固定的波特率 ($f_{osc}/12$) 采样 RXD 引脚的电平信息, 完成 8 位数据接收, 硬件置位接收中断标志位 RI, 执行指令 MOV A, SBUF, 将接收的 8 位数据传输给累加器 A。方式 0 的接收时序如图 7-12 所示。

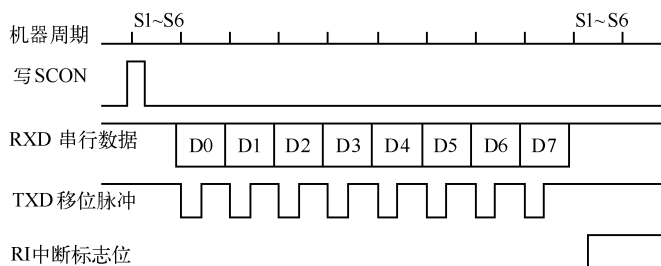


图 7-12 方式 0 接收时序图

7.3.2 方式 1

当 SM0 SM1 = 01 时, 串行口配置为工作方式 1。该方式为波特率可变的固定 8 位数据的通用异步通信方式。

方式 1 字符帧包括 1 位起始位 (低电平)、8 位数据位和 1 位停止位 (高电平), 按照先发送/接收低位, 后发送/接收高位的顺序通信, 波特率由具体的波特率发生器 (定时器 T1 产生) 设置决定。方式 1 的帧格式如图 7-13 所示。

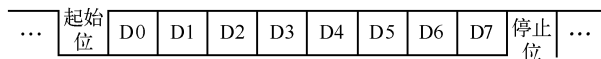


图 7-13 方式 1 帧格式

1. 方式 1 发送

SM0 SM1 = 01, TI = 0, 执行指令 MOV SBUF, A, 内部发送控制信号 $\overline{SEND}$ 变为有效的低电平, 按照设置的波特率, TX 发送时钟脉冲始终存在, 每经过一个 TX 时钟周期, 产生一个移位脉冲, 驱动 TXD 引脚依次输出字符帧的各位信息。在发送起始位之后, 串行口将 SBUF 的 8 位数据以设定的波特率按位从 TXD 引脚串行输出, 低位在先, 高位在后, 8 位数据发送后, 中断标志位 TI 硬件置 1, 然后 $\overline{SEND}$ 失效。图 7-14 为方式 1 的发送时序图。

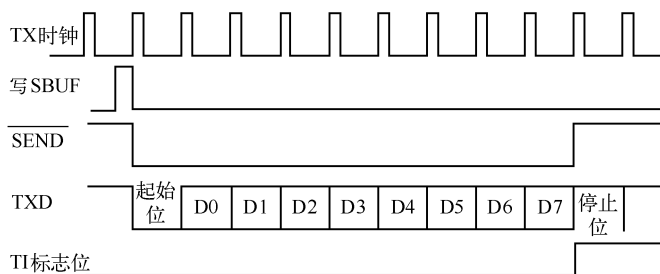


图 7-14 方式 1 发送时序图

2. 方式 1 接收

SM0 SM1 = 01，RI = 0，REN = 1（串行口允许接收），首先检测 RXD（P3.1）引脚上的从 1 到 0 的下降沿跳变，然后持续采样检测，确认是起始位之后开始从 RXD 引脚依次接收各数据位，直到接收到有效的 8 位或检测到有效的停止位，接收结束，满足接收条件硬件置位 RI，执行指令 MOV A, SBUF，将接收的 8 位数据传输给累加器 A。图 7-15 为方式 1 的接收时序图。

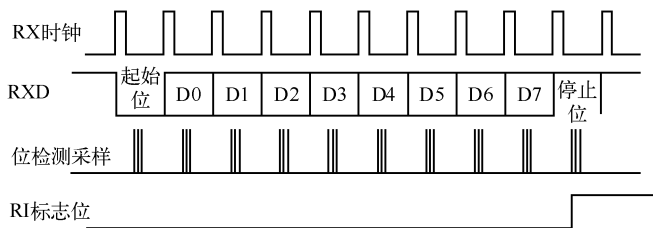


图 7-15 方式 1 接收时序图

方式 1 接收过程需要满足具体的条件才可以保证数据有效，实现写入 SBUF，完成一个完整的接收过程。否则，接收的数据被丢弃，数据无法接收到 SBUF 中。

1) 结束接收时，RI = 0（RI = 0 意味着之前接收的数据已经成功处理，接收缓冲器 SBUF 已空置，保证数据不冲突）。

2) SM2 = 0 或接收到的停止位 = 1（SM2 = 0 意味着无须等待停止位即可保存数据至 SBUF；若 SM2 = 1 则必须接收到可靠的停止位，即收到停止位有效的“1”电平）。

如图 7-15 所示，RX 时钟是接收移位时钟，其频率和波特率相同，位检测采样频率是 RX 时钟的 16 倍，在 1 位数据期间，有 16 个采样脉冲，在第 7、8、9 采样脉冲周期内，分别对 RXD 引脚电平采样，记录其 3 个状态，取其 2 次采样相同的值，则确认电平有效，否则作为干扰信号忽略（三中取二原则）。取第 7、8、9 居中的采样值是为了防止当发送端与接收端的波特率存在差异时，保证通信中不产生错位或漏码（波特率差异允许范围内），三次采样状态值的目的是为了保证接收数据的有效性。

7.3.3 方式 2 和方式 3

SM0 SM1 = 10，配置串行口为工作方式 2。SM0 SM1 = 11，配置串行口为工作方式 3。

两种工作方式均为 11 位通用异步收发通信方式，TXD 作为串行数据发送端，RXD 作为串行数据接收端。两者的不同之处在于产生波特率的方法，方式 2 波特率固定为 $f_{osc}/32$ 或

$f_{osc}/64$ ，而方式3波特率可变，由具体的波特率发生器（定时器T1产生）设置决定。

方式2和方式3的一帧数据包括1位起始位（低电平）、8位数据位、1位程控位（TB8或RB8）和1位停止位（高电平），按照先发送/接收低位，后发送/接收高位的顺序通信，方式2（方式3）的帧格式如图7-16所示。



图 7-16 方式2（方式3）帧格式

1. 方式2（方式3）发送

方式2（方式3）的发送过程与方式1的发送过程类似，只是在方式2（方式3）中，先根据通信协议的约定由程序对TB8进行设置，然后再执行指令MOV SBUF, A，启动发送过程。发送完数据的第8位（D7）之后，由硬件自动将TB8作为第9位数据发送。发送完毕，硬件置位发送中断标志位TI，表示发送已完成。图7-17为方式2（方式3）的发送时序图。

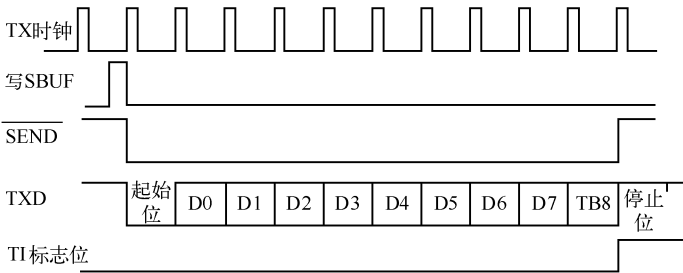


图 7-17 方式2（方式3）发送时序图

2. 方式2（方式3）接收

方式2（方式3）的接收过程与方式1的接收过程类似，只是在方式2（方式3）中，首先设置允许接收控制位REN=1，检测到有效的起始位后，依次按先低位后高位的顺序接收8位数据位D0~D7，再将接收的第9位数据→RB8。如果接收条件具备，硬件置位接收中断标志位RI，表示接收已完成，然后执行指令MOV A, SBUF，将接收的8位数据传输给累加器A。图7-18为方式2（方式3）的接收时序图。

接收完成置位RI需要满足的条件是：①RI=0；②SM2=0或接收到的第9位数据RB8=1。

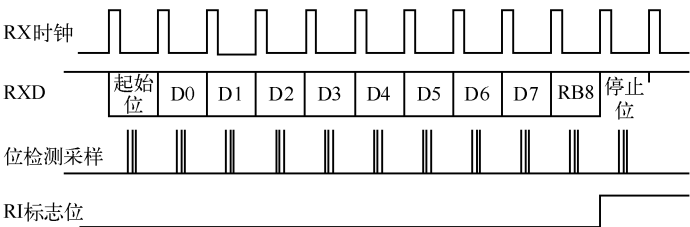


图 7-18 方式2（方式3）接收时序图

7.4 单片机串行口波特率

异步串行通信中，收发双方之间没有直接的同步信号，但是从数据传输快慢的角度看，两者必须步调一致，才能保证数据的正确通信。因此，收发双方对数据传输速率的约定，就归结为波特率的设置。

7.4.1 波特率与字符传输速率

波特率是指串行口每秒发送或接收数据的位数，单位为位/秒 (bit/s)。设发送一位所需要的时间为 T 秒，则波特率为 $1/T$ 。常见的波特率包括 1200 bit/s、4800 bit/s、9600 bit/s 和 19200 bit/s 等。波特率越大，传输速率越快，单位时间内数据传输量就越大，效率越高。

波特率对应的是二进制的位传输速率，字符速率对应的是字符帧的传输速率。在异步通信中，一个字符帧包括起始位、数据位、校验位和停止位等信息，所以字符传输速率不等同于波特率，但两者之间存在具体的关系。

字符传输速率和字符帧格式、波特率有如下关系：

$$\text{字符传输速率} = \frac{\text{波特率}}{n \text{ 位/帧}} \quad (7-1)$$

例 7-2 采用异步通信方式发送 ASCII 码，规定字符帧格式为 1 位起始位、8 位数据位、1 位奇校验位和 2 位停止位。若字符传送速率为每秒 100 个字符，求该异步通信的波特率及传送一位二进制位所需要的时间。

根据题目条件可知，一个字符帧包括 $1 + 8 + 1 + 2 = 12$ 位，即 12 位/帧，代入式 (7-1) 可知

$$\begin{aligned} \text{波特率} &= 100 \text{ 帧/秒} \times 12 \text{ 位/帧} = 1200 \text{ 位/秒} = 1200 \text{ bit/s} \\ \text{传送一位需要时间} &= 1/1200 \text{ bit/s} = 0.833 \text{ ms/位} \end{aligned}$$

7.4.2 波特率的计算

从表 7-1 和 7.3 节内容可知，单片机串行口有四种工作方式，其中方式 0 和方式 2 的波特率固定，而方式 1 和方式 3 的波特率可变。固定的波特率与振荡频率有关，可变的波特率则由定时器 T1 的溢出速率来确定。

1. 方式 0 的波特率

方式 0 的波特率固定，其值为 $f_{\text{osc}}/12$ 。

例如，单片机时钟振荡频率 $f_{\text{osc}} = 12 \text{ MHz}$ ，则波特率为 $f_{\text{osc}}/12 = 1 \text{ MHz}$ 。

2. 方式 2 的波特率

方式 2 的波特率固定，其值为 $f_{\text{osc}}/32$ 或 $f_{\text{osc}}/64$ 。存在两种波特率可供选择的原因是 PCON 寄存器中 SMOD 位的设置。则有

$$\text{波特率} = \frac{2^{\text{SMOD}}}{64} \times f_{\text{osc}} \quad (7-2)$$

SMOD 设置为 0：波特率为 $f_{\text{osc}}/64$ ；SMOD 设置为 1：波特率为 $f_{\text{osc}}/32$ ，波特率倍增。

3. 方式 1 或方式 3 的波特率

方式 1 和方式 3 波特率与 T1 的溢出速率有关。溢出率不同，则波特率不同，因此这两种工作方式的波特率是可变的，两者存在如下比例关系：

$$\text{波特率} = \frac{2^{\text{SMOD}}}{32} \times \text{T1 的溢出率} \quad (7-3)$$

T1 可以工作在 3 种工作方式，其溢出率定义为定时时间的倒数，计算公式如下：

$$\text{T1 溢出率} = \frac{f_{\text{OSC}}}{12} \times \left(\frac{1}{2^n - \text{初值 X}} \right) \quad (7-4)$$

将式 (7-4) 代入式 (7-3) 可得方式 1 或方式 3 的波特率计算公式为

$$\text{波特率} = \frac{2^{\text{SMOD}}}{32} \times \frac{f_{\text{OSC}}}{12} \times \left(\frac{1}{2^n - \text{初值 X}} \right) \quad (7-5)$$

式 (7-4) 和式 (7-5) 中的 n 值是根据定时器 T1 的工作方式决定的，方式 0：n = 13；方式 1：n = 16；方式 2：n = 8。定时初值 X 根据具体的波特率推导。

由于 T1 方式 0 和方式 1 工作中，定时时间到溢出后，需要编程重新装载定时初值 X，这不可避免会影响波特率的准确性，因此实际使用中经常采用自动装载初值的方式 2。这种方式操作方便，避免软件重载初值引入定时误差，导致波特率不稳。

表 7-2 常用波特率与 T1 的定时初值 X 关系表

串行口工作方式	波 特 率	时钟振荡频率 f_{OSC}	SMOD 位	定时器 T1		
				C/T	工作方式	初值 X
方式 1 或方式 3	62500 bit/s	12 MHz	1	0	2	FFH
	19200 bit/s	11.0592 MHz	1	0	2	FDH
	9600 bit/s	11.0592 MHz	0	0	2	FDH
	4800 bit/s	11.0592 MHz	0	0	2	FAH
	2400 bit/s	11.0592 MHz	0	0	2	F4H
	1200 bit/s	11.0592 MHz	0	0	2	E8H
	19200 bit/s	6 MHz	1	0	2	FEH
	9600 bit/s	6 MHz	1	0	2	FDH
	4800 bit/s	6 MHz	0	0	2	FDH

注：当 51 单片机时钟振荡频率 $f_{\text{OSC}} = 6 \text{ MHz}$ 和 12 MHz 时，严格按照定时器 T1 定时初值 X 计算的波特率和标称值存在一定的误差。例如，设置 SMOD = 1，T1 工作方式 2，定时初值 X = FDH，实际计算波特率应为 10416 bit/s，而表 7-2 中波特率为 9600 bit/s，两者相差 816 bit/s。如果 $f_{\text{OSC}} = 11.0592 \text{ MHz}$ ，则波特率和标称值不存在误差。

例 7-3 若 51 单片机时钟振荡频率 $f_{\text{OSC}} = 11.0592 \text{ MHz}$ ，T1 作为波特率发生器，要求波特率为 4800 bit/s，求定时初值 X。

设 T1 工作在方式 2，选择 SMOD = 0。根据题目已知条件，代入式 (7-5) 可知

$$\text{波特率} = \frac{2^0}{32} \times \frac{11.0592 \text{ MHz}}{12} \times \left(\frac{1}{2^8 - \text{初值 X}} \right) = 4800 \text{ bit/s}$$

计算得到定时初值 X = FAH。

7.5 串行口的应用

7.5.1 串行口扩展并行 I/O 口

51 单片机的串行口方式 0 的工作方式一般结合外部的同步移位寄存器，实现串/并数据的转换，达到扩展并行 I/O 口的目的。

例 7-4 利用 51 单片机的串行口与移位寄存器芯片的连接，完成一路 8 位并行输出口的扩展，并行输出数据 37H。

74LS164 就是一款常用的串行输入并行输出的同步移位寄存器芯片，其功能为：

CLR：同步清除输入端，低电平时输出端 QA ~ QH 均锁定为低电平。正常工作情况下要求为高电平。

A 端口和 B 端口：两个串行数据输入端，互为低电平锁定，即当 A 为低电平时，B 输入不起作用，反之亦然。正常工作情况下一般 A 与 B 并联使用或者其中一个固定接高电平。

CLK：时钟输入端，工作条件满足情况下，脉冲上升沿实现 QA ~ QH 输出端状态的移位操作。

根据题目要求，设计具体电路如图 7-19 所示。

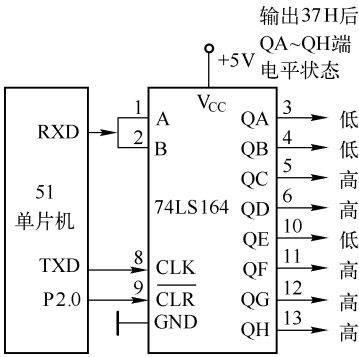


图 7-19 51 单片机与 74LS164 的连接图

```
ORG 0000H
LJMP MAIN
ORG 0030H
MAIN: SETB P2.0      ;P2.0 引脚输出高电平,使能 74LS164 芯片
      MOV SCON,#00H  ;设置串口工作方式 0,禁止接收
      MOV A,#37H      ;准备待发送数据 37H
      MOV SUBF,A      ;TXD 输出移位脉冲,RXD 发送 37H
      SJMP $
      END
```

执行 MOV SUBF, A 命令，8 个移位脉冲后，37H 出现在 74LS164 的 QA ~ QH 输出端。

7.5.2 双单片机异步串行通信

例 7-5 编程实现 1#单片机片内程序存储器中的 0100H ~ 010FH 单元的数据块通过串行口发送给 2#单片机，2#单片机将接收的数据依次存储在 60H ~ 6FH 的单元中。1#和 2#单片机都采用频率为 $f_{osc} = 11.0592 \text{ MHz}$ 的晶振，规定串行口为工作方式 1，波特率为 9600 bit/s。

双单片机串行口的通信电路连接图如图 7-20 所示。

(1) T1 定时初值 X

根据题目要求 $f_{osc} = 11.0592 \text{ MHz}$ 、波特率 9600 bit/s，设 SMOD = 0，查表 7-2 得 X = FDH。

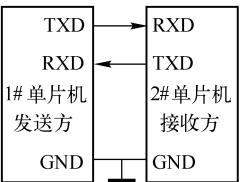


图 7-20 双单片机通信连接图

(2) 1#单片机的发送程序 (查询方式发送)

```
ORG    0000H
LJMP   MAIN
ORG    0030H
MAIN: MOV    TMOD, #20H    ;T1 工作在方式 2
      MOV    TL1, #0FDH    ;定时初值 X
      MOV    TH1, #0FDH    ;定时初值 X
      SETB   TR1           ;启动定时器 T1,即启动波特率发生器
      MOV    SCON, #40H    ;串口工作在方式 1,禁止接收 REN = 0
      MOV    PCON, #00H    ;SMOD = 0,波特率不倍增
      MOV    DPTR, #TAB    ;设置待发送数据区首址
      MOV    R0, #00H      ;清取数计数器
LP:    LCALL  DELAY        ;调用延时子程序
      MOV    A, R0         ;赋值 A 做变址寻址
      MOVC   A, @ A + DPTR ;基址 + 变址方式读取程序存储器中的数据
      INC    R0            ;取数计数器加 1
      MOV    SBUF, A       ;写入发送缓冲器,开始发送
      JNB    TI, $         ;TI = 0 则原地踏步,等待发送完成
      CLR    TI           ;发送已完成,软件清除发送标志位 TI = 0
      CJNE   R0, #10H, LP  ;判断数据区数据是否已经发完,总共需要发 16 个数
      SJMP   $             ;原地踏步循环
DELAY: MOV    R1, #0FFH    ;延时子程序
LOOP: DJNZ   R1, LOOP
      RET
ORG    0100H
TAB: DB 41H,42H,43H,44H,45H,46H,47H,48H,09H,4AH,4BH,4CH,4DH,4EH,4FH,50H
      END
```

(3) 2#单片机的接收程序 (中断方式接收)

```
ORG    0000H
LJMP   MAIN
ORG    0023H
LJMP   REC    ;转移到串行口中断服务程序
ORG    0030H
MAIN: MOV    TMOD, #20H    ;定时器 T1 工作在方式 2
      MOV    TL1, #0FDH    ;定时初值
      MOV    TH1, #0FDH    ;定时初值
      SETB   TR1           ;启动定时器 T1,即启动波特率发生器
      MOV    SCON, #50H    ;串口工作在方式 1,允许接收 REN = 1
      MOV    PCON, #00H    ;SMOD = 0,波特率不倍增
      MOV    R0, #60H      ;设置待接收数据区首址
      SETB   EA            ;开放总中断
```

SETB	ES	;允许串行口中断
SJMP	\$	;原地踏步循环
REC;CLR	RI	;响应串行口中断,接收已完成,清除接收标志位 RI=0
MOV	A,SBUF	;读取接收缓冲器 SBUF 内数据存入 A
MOV	@R0,A	;接收到的数据存入指定内部 RAM 单元
INC	R0	;RAM 单元指针指向下一个单元
RETI		;中断返回
END		

7.5.3 多机通信

多机通信时,一般采用一主多从构建多机通信系统,如图 7-21 所示。多机通信中,一片单片机配置为主机 M,多片单片机配置为从机 S1、S2、...,每个从机 S_x 设置一个地址 A_x。主机 M 发送的信息可以被选中的从机 S_x 接收,任何一个从机 S_x 发送的信息只能被被主机 M 接收。即从机只能与主机通信,从机相互之间不能够通信。

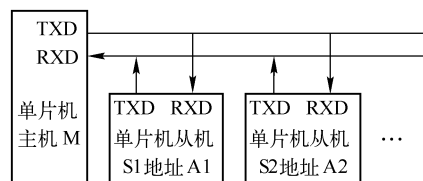


图 7-21 多单片机通信系统

主机 M 发送的字符帧分为地址帧和数据帧,从机 S_x 接收到符合自己地址的地址帧,才能够接收主机 M 之后发送的数据帧。

按照方式 2 或方式 3,若 SM2 = 1,表明工作在多机通信状态下,此时仅当串行口接收到的第 9 位数据 RB8 为 1 时,才能将数据装入接收缓冲器 SBUF,完成一次接收过程,否则,将丢弃数据。如果 SM2 = 0,不管第 9 位数据 RB8 是 0 还是 1,均将数据装入接收缓冲器 SBUF,完成一次接收过程。应用这个特性,便可实现多机通信。

多机通信的关键在于从机 S_x 如何识别主机 M 发送的是地址帧还是数据帧。规定当接收到的第 9 位数据 (RB8) 为 1 时,该字符帧为地址帧;当接收到的第 9 位数据 (RB8) 为 0 时,该字符帧为数据帧。首先,主机 M 先发送地址帧由从机进行地址比对,然后从机根据比对结果进行相应设置。主机 M 再发送数据帧,交由成功比对地址的从机 S_x 处理。

整个过程具体步骤如下:

1) 设置所有从机工作在方式 2 或方式 3,置位 SM2 = 1 和 REN = 1 (主机 M 设置相对自由,SM2、REN 可根据需要开启)。

2) 主机 M 若要发送数据给某个从机 S_x,首先发送一帧以该从机 S_x 地址为内容的地址帧 (地址帧中第 9 位 TB8 = 1)。

3) 由于所有从机的 SM2 = 1,处于接收地址帧状态。此时,所有的从机接收到主机 M 地址帧后,将第 9 位载入 RB8,然后将接收的 8 位地址与本机地址比较,只有地址相同的从机 S_x 设置本机 SM2 = 0,其他保留 SM2 = 1 的状态。

4) 由主机 M 发送的地址帧选中从机 S_x,从机 S_x 对 SM2 清 0,其余从机仍为 SM2 = 1。主机 M 发送地址帧之后,紧接着发送一个或多个数据帧 (数据帧中第 9 位 TB8 = 0),只有被选中的从机 S_x 能够接收这些数据帧,其他从机则丢弃这些数据帧,即不会将这些数据帧

的数据传输给 SBUF。

整个通信过程中，主机 M 发送的地址帧或数据帧均采用的是广播模式，所有从机均能够侦测到，只不过由于工作状态设置不同，被选中的从机成功接收，其他的从机被迫丢弃，这样主机 M 和从机 S_x 建立了一对一的通信。归纳起来就是主机 M 程序设计时需重点考虑 TB8 的值设置（区别本帧是地址帧还是数据帧），从机程序设计重点考虑 REN、SM2 的设置及 RB8 的判别（完成地址比对与数据接收）。

7.5.4 RS-232 总线及接口电路

1. RS-232C 标准接口总线

RS-232C 总线标准接口（协议）的全称是 EIA-RS-232C 标准，定义是“数据终端设备（DTE）和数据通信设备（DCE）之间串行二进制数据交换接口技术标准”。其中 EIA（Electronic Industry Association）代表美国电子工业协会，RS（Recommended standard）代表推荐标准，232 是标识号，C 代表 RS-232 的最新一次修改。RS-232C 标准描述了计算机及相关设备间较低速率的串行数据通信的物理接口及协议。

RS-232C 接口的具体规范包括：

(1) 传输速率与传输距离

RS-232C 接口理论上最大传输速率为 20 kbit/s，线缆最长为 15 m。

(2) 连接形式

RS-232C 接口通常被用于计算机信号输入控制。当通信距离较近时，可不需要 Modem，通信双方可以直接连接，这种情况下，只需使用少数几根信号线。

(3) 信号特性

RS-232C 采用不同逻辑电平定义区分数据线上信号电平和控制线上信号电平。

数据线上信号电平规定：-15 ~ -3 V 为逻辑“1”电平，+3 ~ +15 V 为“0”电平。

控制线上信号电平规定：-15 ~ -3 V 为信号无效，+3 ~ +15 V 为信号有效。

(4) 接口定义

完整的 RS-232C 接口采用两种接口模式，一种是完整的 25 芯 D 型接口，另一种是 9 芯接口。9 芯接口插座的引脚信号功能见表 7-3。

表 7-3 RS-232C 九芯接口信号定义

引脚号	信号名称	方向	信号功能	引脚号	信号名称	方向	信号功能
1	DCD	输入	数据载体检测	2	RXD	输入	接收数据
3	TXD	输出	发送数据	4	DTR	输出	数据终端准备就绪
5	GND		信号地	6	DSR	输入	数据通信设备准备好
7	RTS	输出	请求发送	8	CTS	输入	允许发送
9	RI	输入	振铃指示				

RS-232C 常用的 9 芯插座接口型号为 DB9，主要分为公头和母头两种分类。其引脚编号如图 7-22 所示。

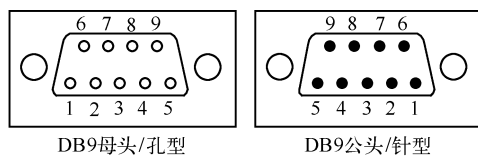


图 7-22 D 型 9 针插头 (DB9) 引脚编号

2. 电平转换

由于 RS-232C 电平采用负逻辑，且电压值较高，而单片机引脚输出电平符合 TTL 标准，两者不能直接连接。因此，单片机与 RS-232C 接口相连时必须借助于电平转换芯片。

MAX232 就是其中一种常用的此类芯片。MAX232 芯片是美信 (MAXIM) 公司专为 RS-232C 标准串口设计的单电源电平转换芯片，其特点为：

- 1) 符合所有的 RS-232C 技术标准。
- 2) 只需要单一 +5 V 电源供电。
- 3) 片载电荷泵具有升压、电压极性反转能力，能够产生 +10 V 和 -10 V 电压 V^+ 、 V^- 。
- 4) 功耗低，典型供电电流为 5 mA。
- 5) 内部集成 2 个 RS-232C 驱动器。
- 6) 高集成度，片外最低只需 4 个电容即可工作。

3. 51 单片机与 MAX232 的电路连接

MAX232 与 51 单片机的接口电路如图 7-23 所示。

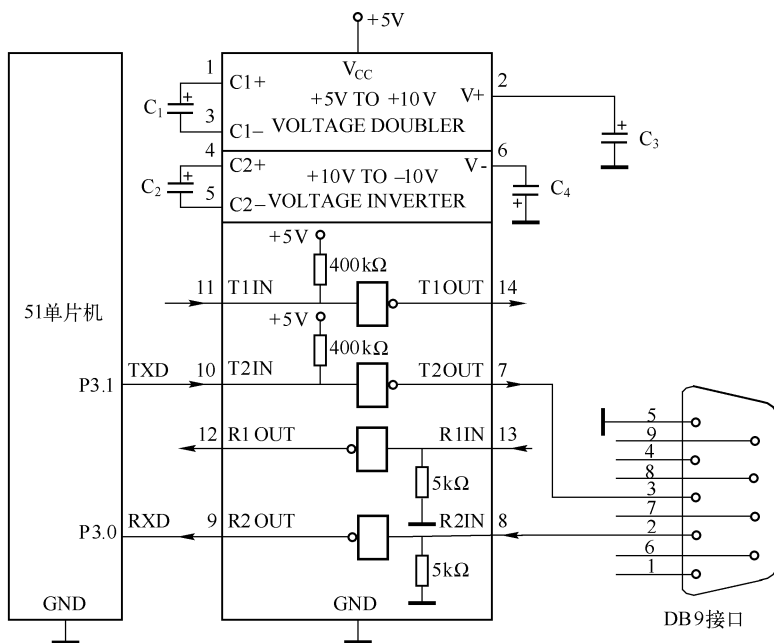


图 7-23 MAX232 与单片机的接口电路图

习题

- 7-1 串行数据传送的主要优点和用途是什么?
- 7-2 简述串行口接收数据和发送数据的过程。
- 7-3 串行口有几种工作方式? 有几种帧格式? 各种工作方式的波特率如何确定?
- 7-4 帧格式为 1 个起始位、8 个数据位和 1 个停止位的异步串行通信方式是方式 ()。
- 7-5 假设某单片机串行口通信, 数据帧格式为起始位 (1 位)、数据位 (7 位)、校验位 (1 位) 和停止位 (1 位), 当该串口每分钟传送 1800 个字符的情况下, 波特率为 ()。
- 7-6 UART 的中文翻译是 (), USART 的中文翻译是 ()。
- 7-7 如何区分 SBUF 是发送缓冲器还是接收缓冲器?
- 7-8 举例说明串行口通信中单片机使用晶振 11.0592 MHz 和晶振 12 MHz 在波特率计算中的差异。
- 7-9 简述串行通信中对传输错误的校验方法。
- 7-10 若晶体振荡器为 11.0592 MHz, 串行口工作于方式 1, 波特率为 4800 bit/s, 写出用 T1 作为波特率发生器的方式控制字和计数初值。
- 7-11 假定串行口串行发送的字符格式为 1 个起始位、8 个数据位、1 个奇效验位和 1 个停止位, 请画出传送字符 “A” 的帧格式。
- 7-12 重新编制例 7-5 的发送程序, 要求使用中断方式编程。
- 7-13 需要传送的字符个数根据需要不确定长度, 如何实现双方的串行通信?
- 7-14 绘制电路图及编写程序, 通过单片机串行口和 74LS164 实现 8 位 LED 发光二极管的交替亮灭, 亮灭间隔 100 ms。

第8章 51 单片机的系统扩展

51 系列单片机内部已经集成了程序存储器、数据存储器 and 基本的 I/O 接口部件，一般情况下可作为单片机的最小系统使用。但根据单片机应用系统的设计和应用需求，51 单片机内部集成的资源常常还不能满足需求，这时就需要在 51 单片机外部扩展相应的部件，扩展的外部部件可以是程序存储器、数据存储器、I/O 接口芯片、A-D、D-A 和其他各种功能部件。本章对 51 单片机应用系统的总线与扩展部件的地址编码方法、常用存储器的扩展方法进行介绍。

8.1 三总线与地址编码

51 单片机具有很强的扩展外部部件的功能，对于常见的外部扩展集成电路芯片，能够按照典型的电路连接，如果扩展两个以上的部件，需要编制每个扩展部件的地址，能方便地构成各种不同需求的 51 单片机应用系统。

8.1.1 51 单片机应用系统总线

所谓总线，是单片机与扩展的各种外部部件连接的公共信号线。在进行系统扩展时，单片机的公共信号线可定义为地址总线（AB）、数据总线（DB）和控制总线（CB）的三总线结构，如图 8-1 所示。单片机各种扩展电路的外部部件（芯片）通过该三总线与单片机连接，因此，学习和掌握单片机的三总线结构是掌握单片机系统扩展的关键。

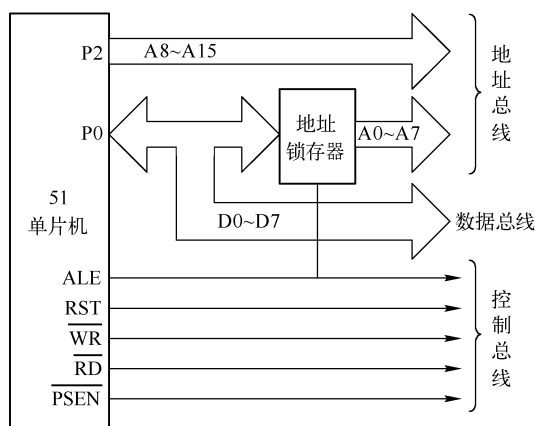


图 8-1 51 单片机系统扩展时的三总线结构

(1) 地址总线

地址总线用于传送单片机发出的地址信号，以便进行外部扩展部件的地址选择。地址总线是单向的，对单片机而言，属于输出信号。地址总线的数目决定了可以直接访问外部扩展

部件的容量。

51 单片机地址总线宽度为 16 位，在片外可扩展的单元地址的最大容量为 2^{16} 字节，即 64 KB 字节，相应的地址空间范围为 FFFFH ~ 0000H。

(2) 数据总线

单片机数据总线是双向的，对单片机而言，通过数据总线可以进行数据、指令和信息等信号的输入和输出操作。数据总线通常同时连接到多个外部扩展的部件（芯片）上，而在同一时间只能有一个有效的数据传送通道，这就由相应的芯片片选信号和具体的控制信号来选择决定。51 单片机是 8 位数据总线。

(3) 控制总线

控制总线用于单片机和外部扩展部件之间进行联络和控制，控制总线是单向的，但对具体不同的控制线，相对单片机来说，既有单片机控制片外扩展部件的输出控制线，也有单片机接收外部扩展部件输出信号的输入控制线。

由于 51 单片机系统采用了上述并行三总线结构形式，使系统扩展简单、规范，提高了系统扩展的可靠性和灵活性。

8.1.2 51 单片机系统总线的构成

由于单片机要兼顾最小系统和系统扩展的应用，其引脚数目受到限制，多数引脚处于多功能复用状态，为此，当必须进行系统扩展时，首先要重新构造系统总线，然后再往系统总线上“挂”接各种存储器、I/O 接口、A-D 和 D-A 等功能部件，从而实现单片机系统的扩展。

1. 地址总线构造

51 单片机的 P0 口是数据总线和低 8 位地址总线分时复用的端口，因此，当构造地址总线时，首先要增加一个 8 位的地址锁存器用于分离地址信号和数据信号。用作单片机地址锁存器的芯片一般有两类：一类是 8D 触发器，如 74LS273、74LS377 等；另一类是 8 位锁存器，如 74LS373 等。不同的地址锁存器由于结构不同，使用时的连接电路也略有不同。

图 8-2 是应用 74LS373 作为地址锁存器构成低 8 位地址总线的扩展电路。单片机向 P0 口送出的低 8 位地址信号在地址锁存允许 ALE 控制线信号的上升沿同步出现，而 74LS373 的 G 端与 ALE 连接，在 ALE 信号的下降沿实现地址锁存。74LS373 的 OE 引脚接地，输出三态门一直处于导通状态，74LS373 的 Q7 ~ Q0 端口输出低 8 位地址总线，从而构成低 8 位地址总线。

51 单片机的 P2 口具有准双向 I/O 口和提供高 8 位地址的功能，当单片机需要扩展高 8 位地址，从而构造 16 位地址总线时，由于 P2 口具有输出锁存功能，直接从 P2 口获得高 8 位地址信号线 A15 ~ A8，与图 8-2 的地址锁存器输出的低 8 位地址 A7 ~ A0 一起构成 51 单片机的 16 位地址总线。P0、P2 口用作地址线后，一般不能再作为 I/O 接口使用。实际应用中，如果只需要 8 位地址总线，P2 口可以作为准双向 I/O 口使用。

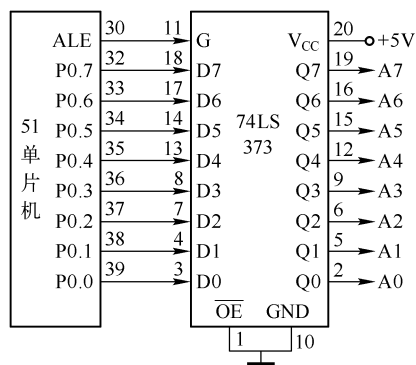


图 8-2 低 8 位地址总线扩展电路

2. 数据总线构造

P0 口直接构成 8 位数据总线，外部扩展部件与数据总线连接的数据端口和 P0 口连接即可。

3. 控制总线构造

控制总线包括单片机对片外扩展部件控制的控制线和片外扩展部件向单片机传输控制信号的控制线。根据扩展功能的不同，51 单片机控制总线主要包括：

1) ALE 信号线：低 8 位地址锁存输出信号，通常在 P0 口输出地址期间用其下降沿控制地址锁存器锁存地址数据。

2) $\overline{\text{PSEN}}$ ：扩展程序存储器的读信号（取指信号）。

3) $\overline{\text{EA}}$ ：片内外程序存储器的选择控制信号，当 $\overline{\text{EA}}=0$ 时，只访问片外程序存储器。

4) $\overline{\text{RD}}$ ：外部扩展部件的读信号，当执行片外数据存储器“读”操作时，读信号自动生效。

5) $\overline{\text{WR}}$ ：外部数据存储器和 I/O 接口的写信号，当执行片外数据存储器“写”操作时，写信号自动生效。

6) $\overline{\text{INT0}}$ ($\overline{\text{INT1}}$)：外部中断 0（外部中断 1）的输入信号。

7) T0 (T1)：定时器/计数器 T0（定时器/计数器 T1）输入信号。

8) RST：系统复位输入信号。

9) RXD (TXD)：串行数据接收（串行数据发送）。

结合第 2 章的介绍可以看出，51 单片机的控制总线有的属于单片机引脚的第一功能信号，有的属于单片机引脚的第二功能信号。

虽然 51 单片机理论上有 P0、P1、P2 和 P3 四个 I/O 接口，但由于应用系统扩展的需要，51 单片机中能真正用于 I/O 口使用的通常就只有 P1 口和 P3 口的部分引脚。

8.1.3 单片机系统总线驱动能力扩展

(1) 总线的驱动能力

在 51 单片机应用系统中，所有扩展的外部部件都要通过单片机的三总线驱动，由于总线的驱动电流总是有限的，只能驱动一定数量的扩展部件。例如，51 单片机的 P0 口可以驱动 8 个 LSTTL 电路，而 P1、P2、P3 口则只能驱动 4 个 LSTTL 电路，当应用系统规模过大，扩展所接的外部芯片多，超过了系统总线的驱动能力时，就必须进行总线驱动，否则，系统将有可能无法正常可靠工作。

(2) 总线的驱动扩展方法

所谓总线驱动，通常是指通过外接一些相应的驱动电路，在电路逻辑不变的前提下，增加总线驱动负载的能力。

由于地址总线和控制总线是单向的，扩展驱动能力时，可采用单向总线驱动器。而数据总线是双向的，必须采用双向三态驱动器进行数据总线驱动能力的扩展。

常用的单向总线驱动器有 74LS244、74LS240（带反向输出）等，其中单向总线驱动器 74LS244 的内部有 8 个三态驱动器，分成两组，每组 4 个，分别由控制端 $\overline{1G}$ 和 $\overline{2G}$ 控制。

图 8-3 为 74LS244 与 51 单片机 P2 口的驱动能力扩展电路，由于 P2 口只作地址输出口，74LS244 的驱动门控制端 $\overline{1G}$ 和 $\overline{2G}$ 接地。对于其他扩展的单向控制总线 $\overline{RD}$ 、 $\overline{WR}$ 等的驱动扩展方法与此相同。

常用的双向总线驱动器有 74LS245，其引脚和 51 单片机的连接如图 8-4 所示，74LS245 双向总线驱动器内部有 16 个三态驱动器，分成两组，共八对，每对由不同方向的驱动器组成，驱动方向由 $\overline{G}$ 和 DIR 两个控制端控制， $\overline{G} = 1$ ，双向端口位高阻态； $\overline{G} = 0$ ，DIR 控制端控制总线驱动器的驱动方向，当 DIR = 0 时，驱动方向从 B 至 A，当 DIR = 1 时则方向相反，从 A 至 B。图 8-4 中的 74LS245 的控制端 $\overline{G}$ 接地，保证驱动器芯片始终处于工作状态，而驱动方向则由 51 单片机控制总线的“读”控制线（ $\overline{RD}$ ）和程序存储器的取指控制线（ $\overline{PSEN}$ ）通过逻辑与后，控制总线驱动器的 DIR 引脚实现。这种逻辑连接方法保证了无论单片机是“读”扩展部件的数据（ $\overline{RD}$ 有效，为低电平），还是从程序存储器取指（ $\overline{PSEN}$ 有效，为低电平），都能保证数据总线 P0 口的输入驱动，数据可从外部传输给 P0 口，从而送进 CPU，而其余时间 DIR = 1（ $\overline{PSEN}$ 和 $\overline{RD}$ 均失效，为高电平），保证 P0 口的输出驱动，数据自 P0 口经总线驱动器 74LS245 向外输出。

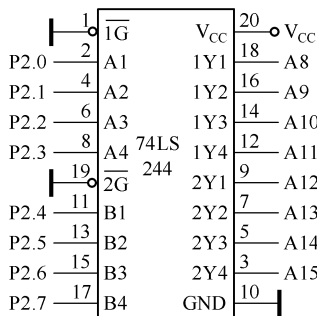


图 8-3 单向总线驱动能力扩展电路

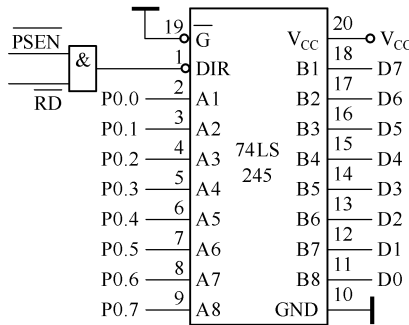


图 8-4 双向总线驱动能力扩展电路

8.1.4 51 单片机系统扩展外部部件的地址空间分配

扩展的外部部件与 51 单片机通过三总线连接，外部部件的地址空间分配是设计单片机应用系统必须掌握的基本技能。由于扩展的外部部件可以是程序存储器、数据存储器、I/O 接口芯片、A-D、D-A 和其他各种功能部件（芯片）。每一块芯片需要有一个唯一的地址标志用于区别其他芯片，而各芯片的端口、内部存储单元和寄存器必须有一个唯一的地址，如何设计、分配相应的地址空间，这就是单片机应用系统的地址空间分配。

1. 51 单片机的扩展能力

由于 51 单片机是按照哈佛结构设计的，即程序存储器的地址空间和数据存储器的地址空间是物理分开，分别寻址的结构，允许二者的地址空间重叠。因此 51 单片机片外可扩展的程序存储器的存储容量是 64 KB；可扩展的数据存储器和其他外部部件的容量共计 64 KB，占用同一个地址空间，统一编址。51 单片机应用系统形成了两个并行的 64 KB 的地址空间，地址范围都是 FFFFH ~ 0000H。

对于程序存储器和数据存储器、其他外部部件的扩展，从硬件上看，它们应用控制总线中不同的控制线，读片外程序存储器的控制信号为 $\overline{\text{PSEN}}$ （取指），读、写外部数据存储器 and 外部扩展部件的控制信号分别为 $\overline{\text{RD}}$ 和 $\overline{\text{WR}}$ ；从软件上看，读片外程序存储器的指令为 MOV_C，读、写外部数据存储器 and 外部扩展部件的指令为 MOV_X。

2. 一般扩展方法

数据存储器 and 扩展的其他各种外部部件尽管有不同的功能、结构和特性，但是与单片机的连接都是按三总线的方式连接，这是在学习 51 单片机应用系统扩展时要始终牢记的问题。

以存储器为例，存储器芯片数据线的数目由存储器芯片的字长决定。常用的数据存储器芯片字长为 8 位，对应有 8 根数据线，8 位数据线（D0 ~ D7）与单片机数据总线（P0.0 ~ P0.7）按低位到高位顺序依次连接即可。

也有极少数存储器芯片是 4 位字长的，应用两块芯片组成 8 位字长的存储器，一块芯片作为低 4 位数据存储器，一块芯片作为高 4 位数据存储器，低 4 位和高 4 位数据线分别与单片机数据总线的低 4 位（P0.3 ~ P0.0）和高 4 位（P0.7 ~ P0.4）连接，而两块芯片的地址线和控制线并联连接。

存储器芯片的控制线：对于程序存储器，一般来说，其控制线为“指令读”信号（ $\overline{\text{OE}}$ ，也称为输出使能端），它与单片机的 $\overline{\text{PSEN}}$ 信号线相连。对于数据存储器和其他外部部件，一般而言，其控制线为“读”（ $\overline{\text{RD}}$ ）、“写”（ $\overline{\text{WR}}$ ），分别与单片机的 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 信号线相连。

存储器芯片的地址线：存储器芯片地址线的数目由该芯片的容量决定。

其他各种外部扩展部件的地址线：通常扩展部件的芯片地址线较少，根据扩展部件的功能而定。

对于程序和数据存储器的扩展，如果选用多片存储器芯片来满足系统所需要的存储器容量时，芯片的数目要根据选用的芯片容量和字长位数以及待扩展的存储容量来确定。

若所选存储器芯片字长与单片机字长一致，则只需扩展容量。所需芯片数目和芯片的片选信号数量相等，按下式确定：

$$\text{芯片数目和片选信号数量} = \frac{\text{系统待扩展容量}}{\text{单片存储器芯片容量}}$$

若所选存储器芯片字长与单片机字长不一致（比如为 4 位字长），则此时不仅需要扩展存储器容量，还需要扩展字长。所需芯片数目按下式确定：

$$\text{芯片数目} = \frac{\text{系统待扩展容量}}{\text{单片存储器芯片容量}} \times \frac{\text{系统字长}}{\text{存储器芯片字长}}$$

对于 4 位字长的存储器，片选信号数量恰好是芯片数目的一半。

3. 地址分配与编址方法

通常能够与 51 单片机总线直接连接的各种扩展部件（芯片）的引脚中，至少有一个引脚可以作为芯片的片选信号（或称为使能信号）输入端，分配扩展芯片的地址时，片选端作为芯片的最高位地址线与单片机总线连接，主要用于区别各扩展部件芯片之间的不同地址。常用的地址分配与编址的方法有两种：线选法和译码法。

(1) 线选法

线选法就是将每一个扩展芯片的片选端直接与单片机地址总线的高位地址线连接，即把

单独的高位地址线接到某一个芯片的片选信号引脚，控制这一根地址线的状态，就选中该芯片。线选法的优点是简单，不需要额外增加硬件，成本低。缺点是可扩展寻址的空间小，扩展芯片较多时，地址资源浪费多，设计时也容易出现错，主要用于扩展芯片少的应用系统。

(2) 译码法

译码法就是利用译码器对 51 单片机的若干根高位地址线进行译码，其输出提供给扩展芯片作为片选信号。这是最常用的地址分配方法。

译码法又分两种：部分译码法和全译码法。

部分译码法是指单片机的高位地址线仅用一部分参加译码，另一部分空余的高位地址线不用，可悬空，作为无关地址线（也称为无关地址位）。其特点与线选法类似，存在地址空间浪费的不足，但是相比线选法，由于进行了部分译码，能够避免各部件芯片的地址出现重叠的现象。

全译码通常用于存储器芯片的扩展，单片机系统的地址总线与存储器芯片的地址线按低位地址线至高位地址线的顺序依次相连后，剩余的所有高位地址线全部参加译码，译出的信号作为存储器芯片的片选信号。全译码方法产生的存储器芯片的地址空间是唯一的，能够扩展的存储器空间也是最大的，各芯片的地址空间是连续的。

4. 地址译码电路

地址译码电路通常应用常用的译码器芯片：74LS138（3－8 线译码器），74LS139（双 2－4 线译码器）和 74LS154（4－16 线译码器）。下面主要介绍 74LS138 及其 51 单片机常用的译码电路。

74LS138 是一种 3－8 译码器，有 3 个数据输入端， $G1$ 、 $\overline{G2A}$ 和 $\overline{G2B}$ 共 3 个使能输入端，当 $G1 = 1$ ， $\overline{G2A} = \overline{G2B} = 0$ 时，选通该译码器，A、B、C 三个输入信号经译码产生 8 种状态输出，输出端 $\overline{Y7} \sim \overline{Y0}$ 中只有一个引脚输出状态为低电平；当 $G1$ 、 $\overline{G2A}$ 、 $\overline{G2B}$ 为其他组合状态时， $\overline{Y7} \sim \overline{Y0}$ 输出端全部为高电平，74LS138 译码器的真值表见表 8-1。

表 8-1 74LS138 真值表

输 入						输 出							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	$\overline{Y7}$	$\overline{Y6}$	$\overline{Y5}$	$\overline{Y4}$	$\overline{Y3}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
其他状态			×	×	×	1	1	1	1	1	1	1	1

74LS138 的引脚如图 8-5 所示，常用的地址译码电路如图 8-6 所示，根据表 8-1，图 8-6 中的 74LS138 译码器始终处于工作状态。单片机高 8 位地址线的最高 3 位 P2.7

(A15)、P2.6 (A14) 和 P2.5 (A13) 分别与 74LS138 译码器的 3 个数据输入端 C、B、A 连接。P2.7、P2.6、P2.5 组成 3 位二进制数，共有 8 个二进制编码，74LS138 译码器的 $\overline{Y7} \sim \overline{Y0}$ 输出端通常分别与单片机扩展的外部存储器和其他各种接口部件的片选（使能）端连接，并作为这些部件（芯片）的最高位地址。图 8-6 中的译码电路最多能够实现单片机与 8 个扩展部件（芯片）的片选（使能）端连接，能够确保 8 个扩展部件（芯片）的地址不重叠。74LS138 译码器的 $\overline{Y7} \sim \overline{Y0}$ 输出端对应的译码输出的地址空间范围分别为：

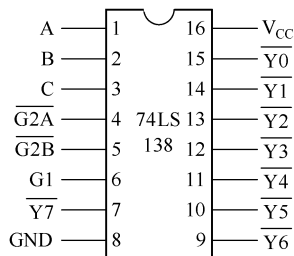


图 8-5 74LS138 引脚图

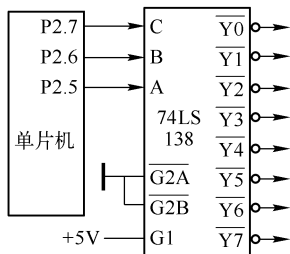


图 8-6 74LS138 地址译码电路

$\overline{Y0}$: 1FFFH ~ 0000H (P2.7、P2.6、P2.5 = 000)

$\overline{Y1}$: 3FFFH ~ 2000H (P2.7、P2.6、P2.5 = 001)

$\overline{Y2}$: 5FFFH ~ 4000H (P2.7、P2.6、P2.5 = 010)

$\overline{Y3}$: 7FFFH ~ 6000H (P2.7、P2.6、P2.5 = 011)

$\overline{Y4}$: 9FFFH ~ 8000H (P2.7、P2.6、P2.5 = 100)

$\overline{Y5}$: BFFFH ~ A000H (P2.7、P2.6、P2.5 = 101)

$\overline{Y6}$: DFFFH ~ C000H (P2.7、P2.6、P2.5 = 110)

$\overline{Y7}$: FFFFH ~ E000H (P2.7、P2.6、P2.5 = 111)

由上述分析可见，存储器的地址空间分配，实际上就是将单片机的地址总线与存储器芯片的地址引脚按一定规律连接，最终达到一个物理存储单元对应一个单元地址的要求。

8.2 程序存储器的扩展及应用

在 51 单片机应用系统中，对于片内有程序存储器的芯片来说，如果容量不够，需要扩展。程序存储器扩展的容量随应用系统的要求可随意配置，最大为 64KB。

程序存储器一般采用只读存储器（ROM），因为这种存储器在电源关断后依然能保存数据信息，适合程序存储器的特点。ROM 分为掩模 ROM、可编程 ROM（PROM）、EPROM、EEPROM 和 Flash ROM 等。

本节以 EPROM 芯片为例，介绍程序存储器扩展的电路原理和编制地址的方法。

8.2.1 常用 EPROM 芯片介绍

EPROM 的典型芯片是 Intel 公司的 27 系列产品，例如，2764（8 KB × 8）、27128

(16 KB × 8)、27256 (32 KB × 8)、27512 (64 KB × 8)；型号名称“27”后面的数字表示 EPROM 的位存储容量，单位为 kbit。随着大规模集成电路技术的发展，大容量存储器芯片的价格不断下降，它们的引脚功能基本类似，均向下兼容。其芯片的引脚如图 8-7 所示。

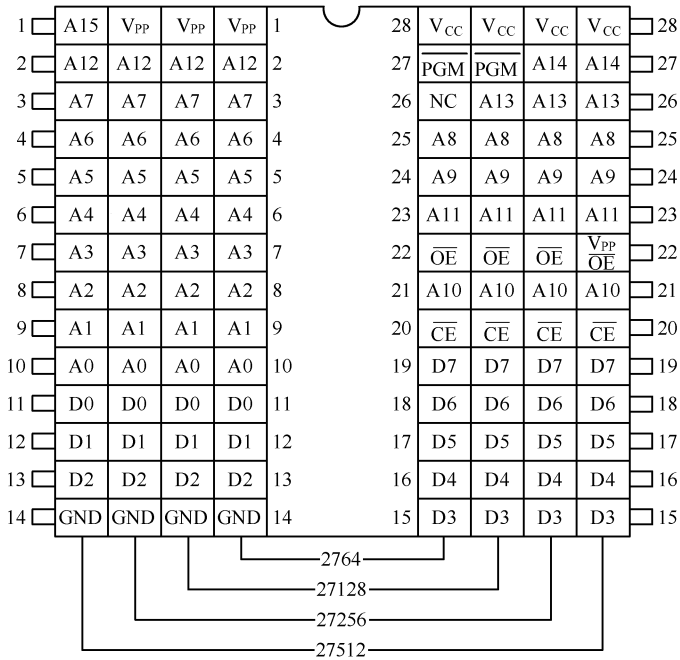


图 8-7 常用的 EPROM 芯片引脚图

图 8-7 中，芯片的引脚功能如下：

A15 ~ A0：地址线引脚。地址线引脚的数目由芯片的存储容量来定，不同容量的存储器，其地址线引脚的数目不同，一个地址值唯一地对应存储器中的一个单元。

D7 ~ D0：8 位数据线引脚。

$\overline{\text{CE}}$ ：片选输入端，低电平有效。

$\overline{\text{OE}}$ ：输出允许控制端，低电平有效。

$\overline{\text{PGM}}$ ：编程时，加编程脉冲的输入端。

V_{PP}：编程时，编程电压（+12 V 或 +25 V）输入端。

V_{CC}：+5 V，芯片工作电压。

GND：数字地。

NC：没有使用的引脚。

8.2.2 EPROM 芯片的工作方式

EPROM 芯片一般都有 5 种工作方式，分别是程序读出、未选中、编程、程序校验和编程禁止，见表 8-2。它们由 $\overline{\text{CE}}$ 、 $\overline{\text{OE}}$ （ $\overline{\text{OE}}$ ）和 $\overline{\text{PGM}}$ 信号的状态组合来确定。单片机应用系统设计时，通常按照程序读出的工作方式设计。

表 8-2 EPROM 的 5 种工作方式

方式 \ 引脚	$\overline{\text{CE}}/\overline{\text{PGM}}$	$\overline{\text{OE}}$	V_{PP}	D7 ~ D0
程序读出	0	0	+5V	程序读出
未选中	1	×	+5V	高阻
编程	50 ms 正脉冲	1	+25 V (或 +12 V)	程序写入
程序校验	0	0	+25 V (或 +12 V)	程序读出
编程禁止	0	1	+25 V (或 +12 V)	高阻

表 8-2 中，×表示任意状态，可以是“0”，也可以是“1”。

8.2.3 程序存储器的扩展

程序存储器的扩展同样是按三总线模式来进行设计的，用到的单片机三总线分别是：

- 1) 数据总线：D7 ~ D0。
- 2) 地址总线：A15 ~ A0（根据程序存储器的不同容量，应用地址总线的数量有所不同）。
- 3) 控制总线：ALE——用于低 8 位地址锁存控制； $\overline{\text{PSEN}}$ ——外部程序存储器取指控制信号，接 EPROM 的 $\overline{\text{OE}}$ 端；EA——访问外部程序存储器接地。

图 8-8 是 28 引脚的 EPROM2764 与 51 单片机的连接电路，扩展 8 KB 的 EPROM 程序存储器。

1. 单片程序存储器的扩展

图 8-8 是 28 引脚的 EPROM2764 与 51 单片机的连接电路，扩展 8 KB 的 EPROM 程序存储器。

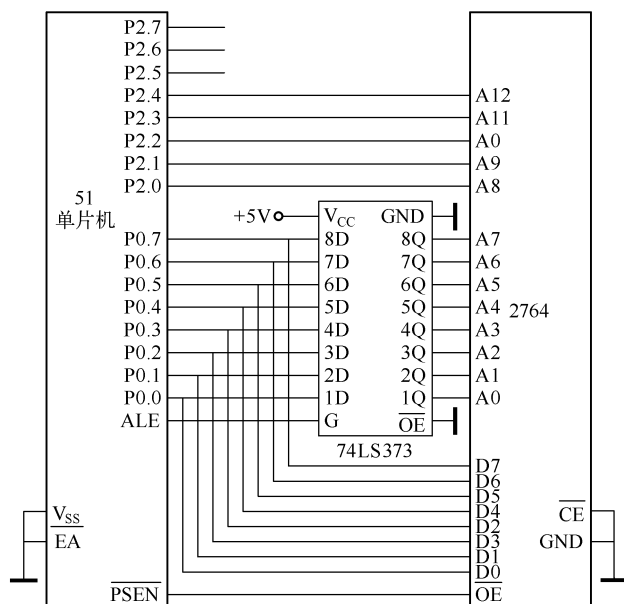


图 8-8 单片 EPROM2764 与 51 单片机的扩展电路

8 KB×8 位的 2764 芯片的地址引脚线有 13 条，顺次和单片机的地址总线 A12 ~ A0 相接。因为只扩展一片程序存储器，所以将 2764 芯片的片选信号直接接地。由于不采用地址

译码器，所以高 3 位地址线 P2.7、P2.6、P2.5 悬空，也称 P2.7、P2.6、P2.5（A15、A14、A13）为无关地址线（位），故有 $2^3 = 8$ 个重叠的 8KB 地址空间。

图 8-8 所示扩展电路中，8 个重叠的 8KB 地址范围为：

0000000000000000 ~ 0001111111111111，即 0000H ~ 1FFFH；
 0010000000000000 ~ 0011111111111111，即 2000H ~ 3FFFH；
 0100000000000000 ~ 0101111111111111，即 4000H ~ 5FFFH；
 0110000000000000 ~ 0111111111111111，即 6000H ~ 7FFFH；
 1000000000000000 ~ 1001111111111111，即 8000H ~ 9FFFH；
 1010000000000000 ~ 1011111111111111，即 A000H ~ BFFFH；
 1100000000000000 ~ 1101111111111111，即 C000H ~ DFFFH；
 1110000000000000 ~ 1111111111111111，即 E000H ~ FFFFH。

通常在应用中，根据个人习惯将无关地址线全部取 0，或全部取 1。

无关地址线全部取 0，地址为：0000H ~ 1FFFH；

无关地址线全部取 1，地址为：E000H ~ FFFFH。

如果扩展 8 KB 的存储容量不够，可将 2764 改为 27128 或 27256 等容量更大的 EPROM，在只扩展 1 片 EPROM 的情况下，每增加一倍的存储容量，增加一根地址线。

2. 采用线选法的多片程序存储器的扩展

例 8-1 使用两片 2764 扩展 16KB 的程序存储器，采用线选法编址选择存储器芯片。

与单片 EPROM 扩展电路相比，多片 EPROM 的扩展除片选线 $\overline{CE}$ 以外，其他的三总线均与单片 EPROM 扩展电路相同。图 8-9 给出了应用 2 片 2764 扩展 16KB 程序存储器的方法。图 8-9 中采用线选法连接，以 P2.5 作为片选，当 P2.5 = 0 时，选中 2764（1）；当 P2.5 = 1 时，选中 2764（2）。

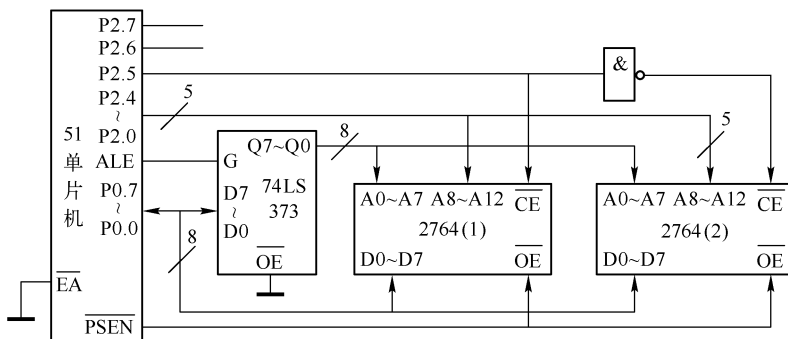


图 8-9 两片 EPROM2764 与 51 单片机连接的扩展电路（线选法编址）

因两根地址线 P2.7、P2.6 未用，故两块芯片各有 $2^2 = 4$ 个重叠的地址空间。

若无关地址线 P2.7、P2.6 全部取 0，2764（1）的地址为 0000H ~ 1FFFH；2764（2）的地址为 2000H ~ 3FFFH。

若无关地址线 P2.7、P2.6 全部取 1，2764（1）的地址为 C000H ~ DFFFH；2764（2）的地址为 E000H ~ FFFFH。

在多片程序存储器扩展时还可以采用译码法编址，这与本节介绍的扩展电路的主要区别

在于程序存储器片选端的连接，采用译码法编址方法将在数据存储器和其他外部芯片扩展的章节中进行介绍。

随着单片 EPROM 容量的逐渐增大，线选法设计简单、硬件少的优点越来越明显，应用也越来越多，对于单片机应用系统来说，多片 EPROM 的扩展已经越来越少用。

8.2.4 程序存储器的读控制与时序

1. 访问程序存储器的控制信号

51 单片机访问片外扩展的程序存储器时，应用了 3 根控制线。

1) ALE：低 8 位地址锁存控制。

2) $\overline{\text{PSEN}}$ ：片外程序存储器“读选通”控制信号。它与扩展的 EPROM 的 $\overline{\text{OE}}$ 引脚连接。

3) $\overline{\text{EA}}$ ：片内、片外程序存储器访问的控制信号。当 $\overline{\text{EA}} = 1$ ，单片机发出的地址值小于片内程序存储器最大地址值时，访问片内程序存储器；当 $\overline{\text{EA}} = 0$ 时，只访问片外程序存储器。

如果从片外 EPROM 中读取指令，除了 ALE 用于低 8 位地址锁存信号之外，控制信号还有 $\overline{\text{PSEN}}$ ，接片外 EPROM 的 $\overline{\text{OE}}$ 引脚。此外，P0 口分时用作低 8 位地址总线 and 数据总线，P2 口用作高 8 位地址线。

2. 操作时序

51 单片机对片外程序存储器的操作时序分两种，即无 MOVX 指令的执行时序和执行有 MOVX 指令的时序。

(1) 单片机应用系统未扩展片外 RAM 和其他外部芯片

51 单片机应用系统未扩展数据存储器（RAM）和其他外部芯片时，编制的程序中不会有 MOVX 指令，即不需要执行 MOVX 指令。在执行非 MOVX 指令时，片外程序存储器的操作时序如图 8-10 所示。

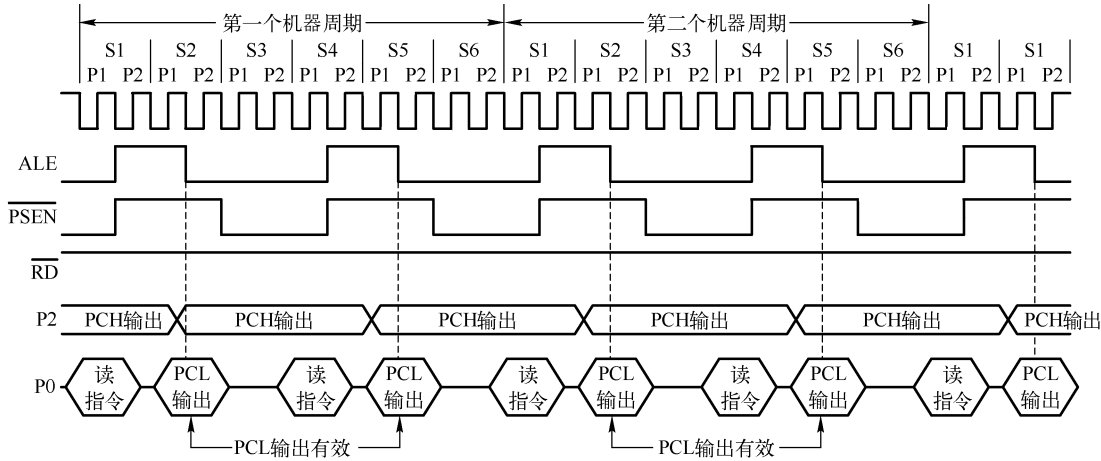


图 8-10 无 MOVX 指令的执行时序

图 8-10 中，PCL：P0 口输出的低 8 位地址；PCH：P2 口输出的高 8 位地址。

由于 P0 口分时复用，故首先将 P0 口输出的低 8 位地址 PCL 锁存在地址锁存器中，然后 P0 口再作为数据口。在每个机器周期中，允许地址锁存 ALE 两次有效，在 ALE 下降沿时，将 P0 口的低 8 位地址 PCL 锁存在地址锁存器中。

$\overline{\text{PSEN}}$ 也是在每个机器周期中两次有效，用于选通片外程序存储器，将指令读入片内。

单片机应用系统未扩展数据存储器（RAM）和其他外部芯片时，ALE 信号以振荡频率的 1/6 出现在引脚上，可用作外部时钟或定时脉冲输出信号。

(2) 单片机应用系统扩展了片外 RAM 和其他外部芯片

在这种情况下，编制的程序中必然会有 MOVX 指令，在执行访问片外芯片的 MOVX 指令时，16 位地址将指向外部芯片，执行 MOVX 指令的时序如图 8-11 所示。

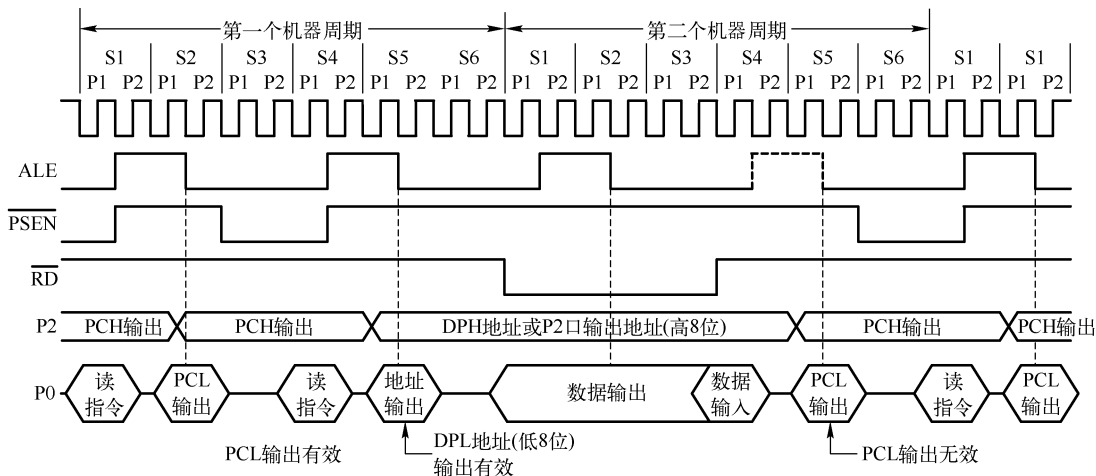


图 8-11 执行 MOVX 指令的时序

执行 MOVX 指令之前，P2 口、P0 口输出的地址 PCH、PCL 指向程序存储器；取出 MOVX 指令并开始执行，ALE 在该机器周期 S5 状态时锁存 P0 口输出的片外 RAM 或其他外部芯片的低 8 位地址。

若执行的是“MOVX A,@ DPTR”或“MOVX @ DPTR,A”指令，则低 8 位地址就是 DPL（DPTR 数据地址指针低 8 位）中的内容，由 P0 口输出；同时，在 P2 口上出现的高 8 位地址就是 DPH（DPTR 数据地址指针的高 8 位）中的内容。

若执行的是“MOVX A,@ Ri”或“MOVX @ Ri,A”指令，则 Ri 为数据地址指针低 8 位，其内容为低 8 位地址 PCL，由 P0 口输出，而 P2 口输出的是 P2 口锁存器的内容。在同一机器周期中将不再出现有效取指信号，下一个机器周期中 ALE 的有效锁存信号也不再出现；当 $\overline{\text{RD}}/\overline{\text{WR}}$ 有效时，P0 口将读/写数据存储器或其他外部芯片寄存器中的数据。

由图 8-11 中可以看出：

1) 执行一次 MOVX 指令就会丢失一个 ALE 脉冲，此时 ALE 输出频率的不再是振荡频率的 1/6。

2) 执行 MOVX 指令的第二个机器周期中，才实现对外部芯片的读/写。

8.3 数据存储器的扩展及应用

扩展数据存储器是设计单片机应用系统的基本技能，本节仅讨论静态数据存储器与 51 单片机的接口与编制地址的方法。

8.3.1 常用的静态数据存储器芯片介绍

51 单片机扩展静态数据存储器的芯片主要有 6116（2 KB×8 位）、6264（8 KB×8 位）、62128（16 KB×8 位）和 62256（32 KB×8 位）等，其双列直插封装的引脚如图 8-12 所示。它们都使用单一的 +5V 电源，除 6116 为 24 引脚外，其余均为 28 引脚，向下兼容。

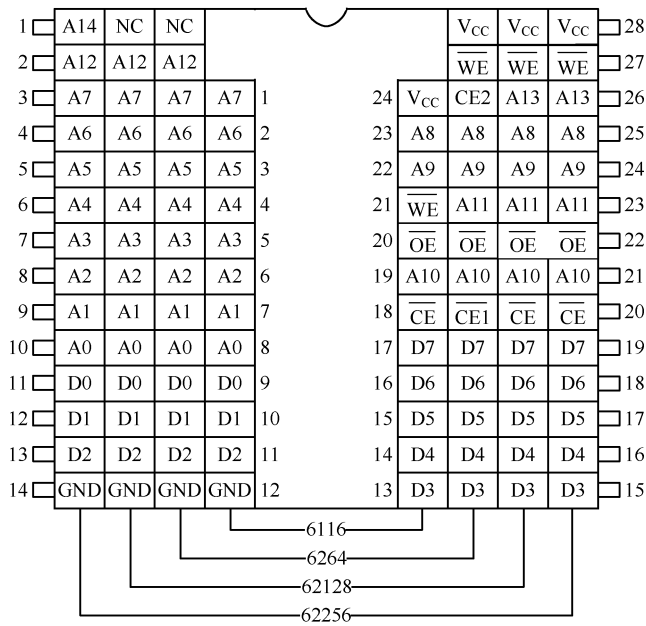


图 8-12 常用的数据存储器芯片引脚

8.3.2 RAM 芯片的工作方式

静态 RAM 存储器有 3 种工作方式，分别是输出、输入和维持。3 种工作方式见表 8-3。

表 8-3 静态数据存储器的工作方式

方式	信号	$\overline{\text{CE}}$	$\overline{\text{OE}}$	$\overline{\text{WE}}$	D0 ~ D7
输出		0	0	1	存储器数据输出
输入		0	1	0	存储器数据输入
维持		1	×	×	高阻态

静态数据存储器芯片的各引脚功能介绍如下：

A14 ~ A0：地址输入线。

D7 ~ D0：双向三态数据线。

$\overline{\text{CE}}$: 片选信号输入线。

对于 6264 芯片, 有两个片选控制端 $\overline{\text{CE1}}$ 和 $\overline{\text{CE2}}$, 只有当 $\overline{\text{CE1}}$ 为低电平和 $\overline{\text{CE2}}$ 为高电平时才能选中该芯片。

$\overline{\text{OE}}$: 读选通信号输入线。 $\overline{\text{WE}}$: 写允许信号输入线。

8.3.3 数据存储器的扩展

与程序存储器扩展类似, 静态数据存储器扩展依然是三总线的扩展, 它们分别是:

1) 数据总线: D7 ~ D0。

2) 地址总线: A15 ~ A0 (根据不同容量, 地址总线数量有所不同)。

3) 控制总线: ALE——用于低 8 位地址锁存控制; $\overline{\text{RD}}$ ——外部数据存储器“读”控制信号, 接 RAM 的 $\overline{\text{OE}}$ 端; $\overline{\text{WR}}$ ——外部数据存储器的“写”控制信号, 接 RAM 的 $\overline{\text{WE}}$ 端。

在扩展数据存储器时, 除了“读”“写”控制信号外, 其他信号线与单片机的连接方法和程序存储器与单片机的连接方法完全相同。

访问片外数据存储器可采用低 8 位地址线寻址, 此时可寻址空间为 256 B, 具体指令为

```
MOVX  A,@ Ri    ;(i=0,1)
```

```
MOVX  @ Ri,A    ;(i=0,1)
```

当采用 16 位地址线寻址时, 可寻址空间为 64 KB, 具体指令为

```
MOVX  A,@ DPTR
```

```
MOVX  @ DPTR,A
```

例 8-2 采用全译码方式用 4 片 62128 芯片扩展 64 KB 数据存储器。

用全译码法扩展 64 KB 外部数据存储器的电路如图 8-13 所示。数据存储器 62128 为 16 KB 容量, 该芯片地址线为 A0 ~ A13, 顺次和单片机的地址总线 A0 ~ A13 相接, 剩余的高 2 位地址线 P2.6、P2.7 提供给 74LS139 译码器译码, 译码后输出的 4 个译码信号作为片选, 分别提供给 4 个存储器芯片。各个 62128 芯片地址分配见表 8-4。

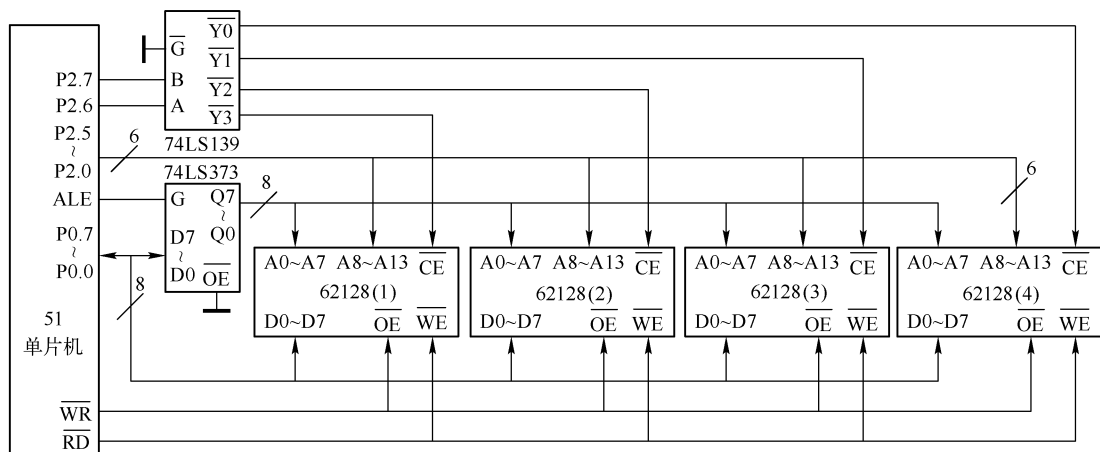


图 8-13 多片数据存储器扩展电路 (全译码方式)

表 8-4 62128 芯片地址分配表

B	A	译码器有效输出	选中芯片	地址范围	有效容量/KB
0	0	$\overline{Y_0}$	62128 (1)	0000H ~ 3FFFFH	16
0	1	$\overline{Y_1}$	62128 (2)	4000H ~ 7FFFFH	16
1	0	$\overline{Y_2}$	62128 (3)	8000H ~ BFFFFH	16
1	1	$\overline{Y_3}$	62128 (4)	C000H ~ FFFFFH	16

由表 8-4 可见，译码法进行地址分配，各芯片的地址空间是连续的。

8.3.4 数据存储器的读写控制与时序

对片外 RAM 的读和写的操作时序的过程基本相同。

1. 读片外 RAM 操作时序

若扩展一片 RAM，应将单片机 $\overline{WR}$ 引脚与 RAM 的 $\overline{WE}$ 引脚连接， $\overline{RD}$ 与芯片 $\overline{OE}$ 引脚连接。

51 单片机读片外 RAM 操作时序如图 8-14 所示。

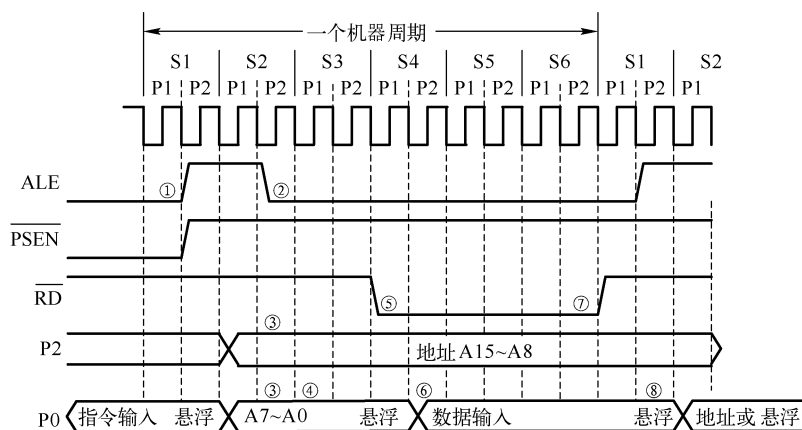


图 8-14 51 单片机读片外 RAM 操作时序图

在一个机器周期的 S1 状态，ALE 信号由低变高（①处），读 RAM 周期开始。在 S2 状态，CPU 把低 8 位地址送到 P0 口总线上，把高 8 位地址传输到 P2 口（执行“MOVX A, @DPTR”指令传输高 8 位；若执行“MOVX A, @Ri”则不传输高 8 位）。

ALE 下降沿（②处）把低 8 位地址信息锁存到地址锁存器 74LS373。而高 8 位地址信息一直锁存在 P2 口锁存器中（③处）。

在 S3 状态，P0 口总线变成高阻悬浮状态④。在 S4 状态，执行指令“MOVX A, @DPTR”使 $\overline{RD}$ 信号为有效（⑤处），稍过片刻， $\overline{RD}$ 信号使片外 RAM 被选中单元的数据传输到 P0 口（数据总线）（⑥处），当 $\overline{RD}$ 回到高电平后（⑦处），P0 口（数据总线）变悬浮状态（⑧处）。

2. 写片外 RAM 操作时序

向片外 RAM 写数据，单片机执行“MOVX @DPTR, A”指令。执行该指令，51 单片机的 $\overline{WR}$ 信号为低有效，此信号使 RAM 的 $\overline{WE}$ 信号输入有效。

写片外 RAM 的时序如图 8-15 所示。开始的过程与读过程类似，但写的过程是 CPU 主动把数据送上 P0 口（数据总线），故在时序上，CPU 先向 P0 口传输 8 位地址，并在 S3 状态就将数据传输到 P0 口（数据总线）（③处）。此间，P0 口（数据总线）不会出现高阻悬浮现象。

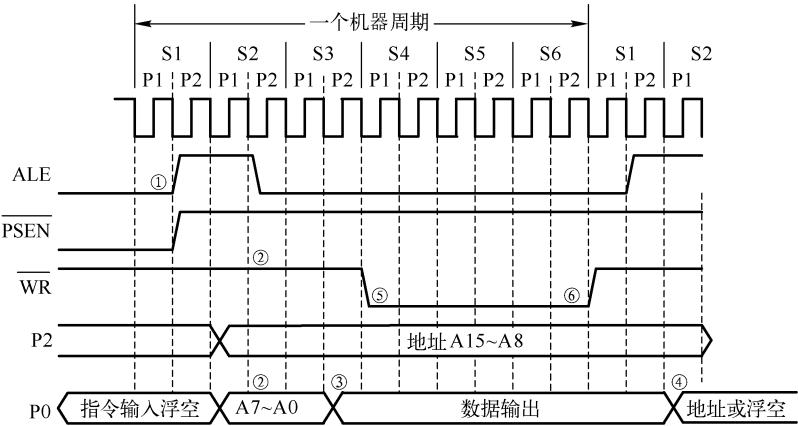


图 8-15 51 单片机写片外 RAM 操作时序图

在 S4 状态，写信号 $\overline{\text{WR}}$ 有效（⑤处），稍过片刻，P0 口（数据总线）的数据就写入到片外 RAM 被选中的单元中，然后写信号 $\overline{\text{WR}}$ 变为无效（⑥处）。

8.4 程序存储器和数据存储器的综合扩展

例 8-3 设在一个 51 单片机应用系统中，应用了 2 片 2764 芯片和 2 片 6264 芯片扩展 16 KB 程序存储器和 16 KB 数据存储器。

这是程序存储器和数据存储器的综合扩展电路，由于数据存储器 and 程序存储器的地址空间是各自独立的，能够分别独立地进行编址，二者的综合扩展可以理解 of 各自独立扩展的简单“并联”，具体扩展电路如图 8-16 所示，图 8-16 中 4 块芯片的地址范围见表 8-5。

图 8-16 中，数据存储器 and 程序存储器的数据总线 and 地址总线公用，在控制总线中，2 片 2764 芯片应用 51 单片机的 $\overline{\text{PSEN}}$ ，2 片 6264 芯片应用 51 单片机的 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ ，2 片 6264 的 CE2 直接与 +5V 单元连接为高电平，由 $\overline{\text{CE1}}$ 为低电平来选中 6264 芯片。

采用全译码方式进行编址，应用 74LS138 译码

器对 P2.7、P2.6、P2.5 译码得到 8 个片选信号 $\overline{\text{Y7}}$ 、 $\overline{\text{Y6}}$ 、 $\cdots$ 、 $\overline{\text{Y0}}$ ，由于只扩展 2 片 2764 和 2 片 6264，片选信号可以只使用 2 根，图 8-16 中使用了 $\overline{\text{Y0}}$ 和 $\overline{\text{Y1}}$ 。其中 $\overline{\text{Y0}}$ 同时与 2764(1) 的 $\overline{\text{CE}}$ 和 6264(1) 的 $\overline{\text{CE1}}$ 连接， $\overline{\text{Y1}}$ 同时与 2764(2) 的 $\overline{\text{CE}}$ 和 6264(2) 的 $\overline{\text{CE1}}$ 连接。从表 8-5

表 8-5 图 8-16 扩展存储器的地址分配

芯 片	地 址 范 围
2764(1)	0000H ~ 1FFFH
2764(2)	2000H ~ 3FFFH
6264(1)	0000H ~ 1FFFH
6264(2)	2000H ~ 3FFFH

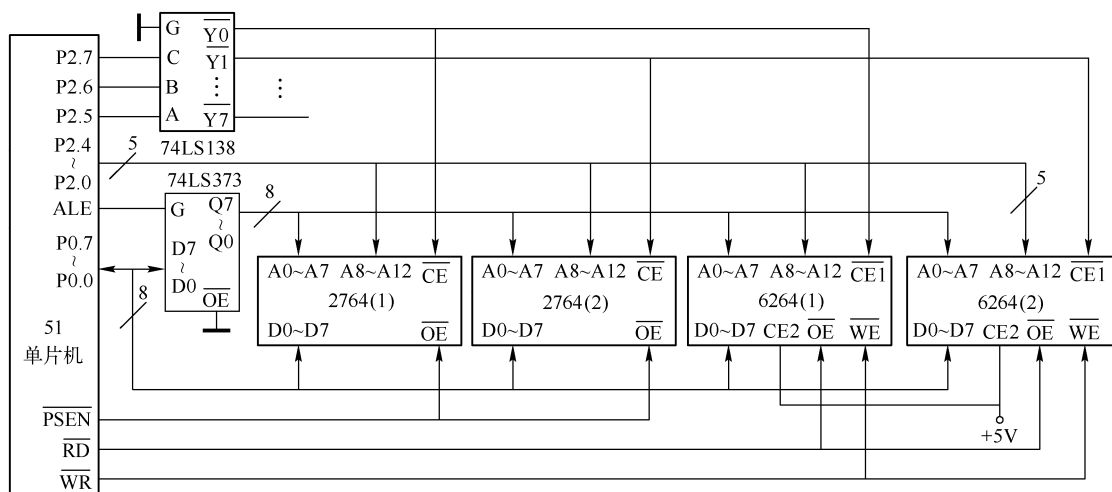


图 8-16 采用译码法的数据存储器和程序存储器综合扩展电路

中可以看出，2 片 2764 芯片和 2 片 6264 芯片的地址范围相同，这样 2764(1) 和 6264(1) 芯片相互之间、2764(2) 和 6264(2) 芯片相互之间都存在一个地址相同的两个单元，将会同时被选通，但两个地址相同的单元一个在程序存储器中，另一个在数据存储器中，其控制信号不同， $\overline{\text{PSEN}}$ 控制程序存储器的数据读出，而 $\overline{\text{RD}}$ 信号控制数据存储器的数据读出， $\overline{\text{WR}}$ 信号控制数据存储器写入。而 $\overline{\text{PSEN}}$ 和 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 信号在任何时刻不会同时有效，所以，即使两个相同地址的单元都被选中，也不会发生总线数据冲突。

如果在图 8-16 中再扩展 RAM 芯片，只需要根据 RAM 芯片的容量，地址线按照顺序与 51 单片机应用系统的地址总线连接，每一块 RAM 芯片的片选信号线与 74LS138 译码器输出的 $\overline{\text{Y7}}$ 、 $\overline{\text{Y6}}$ 、 $\dots$ 、 $\overline{\text{Y2}}$ 中的一个引脚连接，则新增加的 RAM 芯片其他引脚与 51 单片机应用系统的连接与图 8-16 中的 6264(1) 和 6264(2) 相同。但要注意 6264 有两个片选引脚 $\overline{\text{CE1}}$ 和 $\overline{\text{CE2}}$ ，其他 RAM 芯片一般只有一个片选引脚。

习题

8-1 计算机三总线分别为：() 总线；() 总线；() 总线。

8-2 单片机 P0 口为 () 总线和 () 总线的复用端口，74LS373 与 P0 口连接，其输出线作为系统的 ()，74LS373 的 G 端与单片机 () 连接。

8-3 51 系列单片机外扩存储器芯片时，4 个 I/O 口中用作数据总线的是 ()。

A. P0 口和 P2 口 B. P0 口 C. P2 口和 P3 口 D. P2 口

8-4 51 单片机使用的是并行总线结构，按其功能通常把系统总线分为哪几组？并简单描述各总线的构成。在 51 单片机系统中，外接程序存储器和外接数据存储器共用地址总线和数据总线，为何不会发生冲突？

8-5 单片机存储器的主要功能是存储 () 和 ()。

8-6 设外部数据存储器 2000H 单元的内容为 80H，执行下列指令后累加器 A 中的内容

为 ()。

```
MOV P2,#20H
```

```
MOV R0,#00H
```

```
MOVX A,@R0
```

8-7 地址范围为 0000H ~ 3FFFH 的存储器的容量是 () KB。

8-8 在 51 单片机中 PC 和 DPTR 都用于提供地址，但 PC 是为访问 () 存储器提供地址，而 DPTR 是为访问 () 存储器提供地址。

8-9 11 条地址线可选 () 个单元，16 KB 单元需要 () 条地址线。

8-10 4 KB RAM 存储器的首地址若为 0000H，则末地址为 () H。

8-11 区分 51 单片机片外程序存储器和片外数据存储器的最可靠方法是 ()。

A. 看其位于地址范围的低端还是高端

B. 看其离 51 单片机的远近

C. 看其芯片的型号是 ROM 还是 RAM

D. 看其是与 $\overline{RD}$ 还是与 $\overline{PSEN}$ 信号线连接

8-12 按照图 8-16，设两个原始数据存放在片外 RAM 的 1001H 单元和 1002H 单元中，试编写一个程序将按顺序拼装（如将 05H 和 06H 拼为 56H）后的单字节数放入 2002H 单元中。并指出地址为 1001H、1002H 和 2002H 单元分别在图 8-16 中的哪一块 6264 芯片中。

8-13 编写程序，将外部数据存储器中的 3000H ~ 30FFH 单元全部清 0。

8-14 现有 51 单片机、74LS73 锁存器、1 片 2764 EPROM 和 3 片 6264 RAM，请用它们组成一个单片机应用系统，要求如下：

1) 画出硬件电路连线图并标注主要引脚。

2) 指出该 51 单片机应用系统的程序存储器和数据存储器各自存储空间地址范围。

第9章 51 单片机扩展 I/O 接口的设计

51 单片机有 4 个 8 位并行输入/输出(I/O)端口,在实际的单片机应用系统中,如果需要扩展存储器或其他接口芯片,则只有 P1 口能够作为通用的 I/O 口使用。在设计单片机应用系统时,虽然能够选择增强型 51 单片机,尽可能地避免扩展单片机的外部接口芯片,但对于功能较为复杂的单片机应用系统,仍然会需要扩展 I/O 接口芯片,其接口电路设计与编程应用是设计单片机应用系统的基本技能之一。本章主要介绍并行 I/O 接口芯片 8255 和 8155 与 51 单片机的接口电路与编程方法。

9.1 并行 I/O 接口电路与编址

1. I/O 接口功能

I/O 接口即输入/输出接口,是实现外围设备(外设)与单片机之间的连接电路,也是单片机与外设交换信息的桥梁。I/O 接口从数据传送形式上可以分为并行 I/O 接口和串行 I/O 接口。并行 I/O 接口的数据传送是并行的,一次可以传送一个字节数据;串行 I/O 接口的数据传送是串行的,一次只能传送一位二进制信息。I/O 接口主要功能如下:

(1) 实现 CPU 与外设速度匹配

由于外围设备(如打印机等)的工作速度慢,与单片机微秒级速度相差甚远,单片机只有在确认外设已为数据传送做好准备的前提下才能进行数据传送。而要知道外设是否准备好,就需要 I/O 接口电路与外设之间传送状态信息,以实现单片机与外设之间的速度匹配。

(2) 输出数据锁存

与外设相比,由于单片机的工作速度快,数据在数据总线上保留的时间十分短暂,无法满足慢速外设的数据接收。所以在扩展的 I/O 接口电路中应有输出数据锁存器,以保证输出数据能为慢速的接收设备所接收。

(3) 输入数据三态缓冲

单片机需要经过数据总线从外设输入数据,但数据总线上可能“挂”有多个数据源,为使传送数据时不发生冲突,只允许当前时刻正在输出或输入数据的外部接口芯片使用数据总线,其他外部接口芯片应处于隔离状态,为此要求与数据总线连接的外部接口芯片的电路向单片机输出数据的部件必须具有三态缓冲功能。

2. I/O 接口电路的编址

I/O 接口是单片机与外设之间连接电路的总称。I/O 接口电路中具有单元地址的寄存器或缓冲器称为 I/O 端口(简称 I/O 口)。一个 I/O 接口芯片可以有多个 I/O 端口,传送数据的称为数据口,传送命令的称为命令口,传送状态的称为状态口。单片机通过读写端口来和外设交换信息。常用的 I/O 端口编址有两种方式,一种是独立编址方式,另一种是统一编址方式。

(1) 独立编址

I/O 端口地址空间和存储器地址空间分开编址。优点是 I/O 地址空间和存储器地址空间

相互独立，界限分明。但需要设置一套专门的读写 I/O 端口的指令和控制信号。

(2) 统一编址

把 I/O 端口与外部数据存储器的存储单元同等对待。I/O 端口和外部数据存储器 RAM 统一编址。因此外部数据存储器的地址空间和 I/O 端口的地址空间属于同一个地址空间。优点是不需专门的 I/O 指令。缺点是需要把数据存储器单元地址与 I/O 端口地址进行地址配置，划分清楚，避免数据冲突。51 单片机采用的是统一编址的方式。

3. I/O 并行接口的特点

能够用于 I/O 接口的芯片主要分为两种，一是应用不可编程的锁存器集成芯片，比如 74LS373 作为 8 位输出口扩展，74LS244 作为 8 位输入口扩展，每一个 I/O 口都必须编址，在 51 单片机的 $\overline{RD}/\overline{WR}$ 信号的控制下完成数据的输入/输出。对扩展的 I/O 口的操作只能使用 MOVX 类型的数据传输指令。用于扩展 8 位输入口的芯片与 51 单片机数据总线连接的 8 位端口必须具有三态功能，在选择扩展 8 位数据输入接口芯片时尤其要注意这一点。二是应用多功能可编程并行 I/O 接口芯片，其特点是能够直接与 51 单片机的总线连接，有多个工作方式可以选择，在任何时刻，只能工作于其中的一种工作方式，工作方式的选择通过编程设置命令寄存器的每一位的状态来实现。

9.2 可编程 8255 并行 I/O 接口芯片的扩展

9.2.1 8255 芯片介绍

8255 (82C55) 是 Intel 公司生产的通用可编程并行 I/O 接口芯片，能够直接与单片机应用系统的总线连接，它有 1 个命令寄存器、3 个 8 位并行 I/O 端口，有 3 种工作方式，通过对命令寄存器进行设置选择其工作方式。8255 使用方便，通用性强，获得了广泛的应用。

1. 8255 的引脚功能

8255 是一个具有 3 个 8 位并行 I/O 端口的可编程接口芯片，双列直插式 DIP 封装的 8255 的 40 个外部引脚如图 9-1 所示。

1) D7 ~ D0: 8 位三态双向数据线，与单片机数据总线连接，实现单片机与 8255 之间数据的读出与写入，以及控制字写入 8255 命令寄存器。

2) PA7 ~ PA0: 端口 A 输入/输出线，与外设之间的数据（或控制、状态信号）的传送。

3) PB7 ~ PB0: 端口 B 输入/输出线，与外设之间的数据（或控制、状态信号）的传送。

4) PC7 ~ PC0: 端口 C 输入/输出线，与外设之间的数据（或控制、状态信号）的传送。

5) A1、A0: 地址信号线，用来选择 8255 内部 3 个端口和命令寄存器。A1 和 A0 经片内译码产生 4 个有效地址分别选择对应 3 个独立的端口 A、端口 B 和端口 C，以及 1 个命令寄存器。在实际使用中，

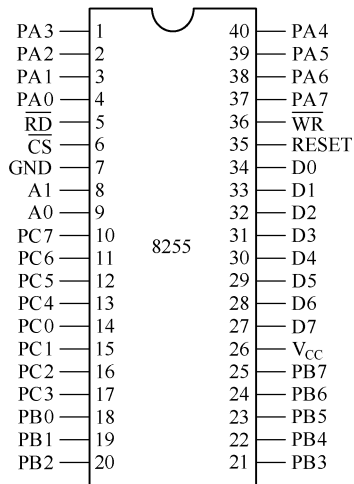


图 9-1 8255 引脚图

A1、A0 地址信号线直接与单片机地址总线的 A1、A0 连接。

6) $\overline{CS}$: 片选信号, 低电平有效, 通常作为 8255 芯片的最高位地址线。

7) $\overline{RD}$: 读控制信号, 低电平有效, 通常直接与单片机控制总线的 $\overline{RD}$ 信号线连接。

8) $\overline{WR}$: 写控制信号, 低电平有效, 通常直接与单片机控制总线的 $\overline{WR}$ 信号线连接。

9) RESRT: 复位信号, 高电平有效。通常直接与单片机控制总线的 RST 信号线连接。

8255 复位后, 端口 A、端口 B、端口 C 的 3 个端口都被设置为基本输入方式。

2. 8255 的内部结构

图 9-2 为 8255 的内部结构图, 由 3 个并行数据 I/O 端口、2 组控制电路、1 个读/写控制逻辑电路和 1 个 8 位数据总线缓冲器组成。

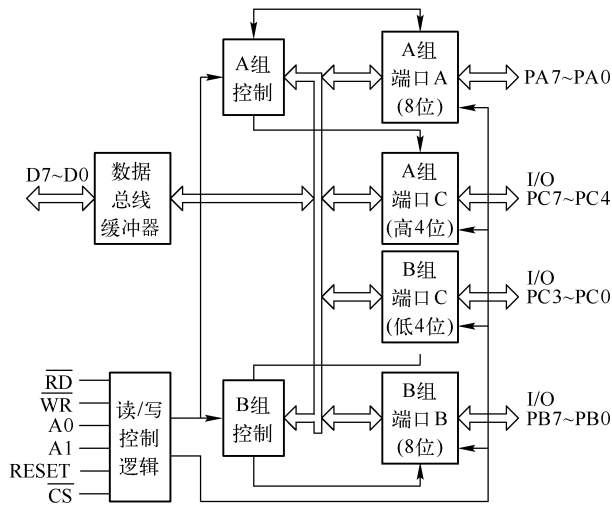


图 9-2 8255 的内部结构

(1) I/O 端口的端口 A、端口 B 和端口 C

端口 A、端口 B 和端口 C 都是 8 位端口, 可以选择作为输入或输出口。端口 C 还可分开设置为高 4 位和低 4 位的两个 4 位端口, 分别作为端口 A、端口 B 在选通工作方式操作时的状态控制信号。

(2) 内部控制逻辑电路

两组内部控制逻辑电路包括 A 组控制与 B 组控制两部分。A 组控制逻辑用来控制端口 A(PA 口)的 PA7 ~ PA0 和端口 C(PC 口)的高 4 位 PC7 ~ PC4; B 组控制逻辑用来控制端口 B(PB 口)的 PB7 ~ PB0 和端口 C(PC 口)的低 4 位 PC3 ~ PC0。用一个命令寄存器来设置 A 组控制和 B 组控制的功能, 还可应用命令寄存器对端口 C 实现按位置位或按位复位的操作。

(3) 数据总线缓冲器

数据总线缓冲器是一个 8 位三态数据缓冲器, 作为 8255 与单片机系统总线之间的接口, 用来传送数据、指令、控制命令以及外部状态信息。

(4) 读/写控制逻辑

读/写控制逻辑电路的功能是负责管理 8255 与单片机之间的数据传送过程。根据片选信

号 $\overline{CS}$ 、读控制信号 $\overline{RD}$ 、写控制信号 $\overline{WR}$ 、复位信号 RESET 及地址总线信号 A1、A0 的组合，得到 A 组控制和 B 组控制命令，实现对 8255 端口的读写操作，各端口读写操作与对应的控制信号之间的关系见表 9-1。

表 9-1 8255 各端口工作状态选择表

$\overline{CS}$	A1	A0	$\overline{RD}$	$\overline{WR}$	所选端口	操作
0	0	0	0	1	PA 口	读端口 A (PA 口数据→数据总线)
0	0	1	0	1	PB 口	读端口 B (PB 口数据→数据总线)
0	1	0	0	1	PC 口	读端口 C (PC 口数据→数据总线)
0	0	0	1	0	PA 口	写端口 A (数据总线→PA 口)
0	0	1	1	0	PB 口	写端口 B (数据总线→PB 口)
0	1	0	1	0	PC 口	写端口 C (数据总线→PC 口)
0	1	1	1	0	命令寄存器	写命令 (数据总线→命令寄存器)
1	×	×	×	×		数据总线缓冲器的输出为高阻态

9.2.2 8255 的工作方式

1. 工作方式选择控制字

8255 共有 3 种基本工作方式：①方式 0——基本输入/输出；②方式 1——选通输入/输出；③方式 2——双向传送（仅限于 PA 口）。PA、PB、PC 的 3 个端口工作方式是通过 8255 的命令寄存器写入一个工作方式控制字来设置的，工作方式控制字的格式如图 9-3 所示。最高位 D7 = 1 是工作方式控制字的标识。

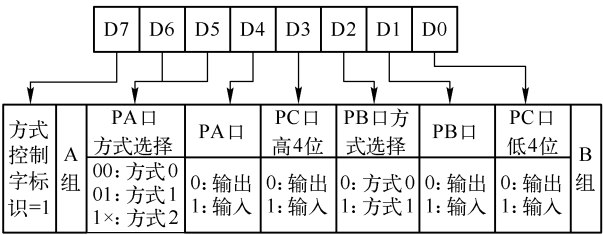


图 9-3 8255 工作方式控制字格式

例 9-1 设 8255 命令寄存器的地址为 FF7FH，试编写其初始化程序，使 PA 口工作于方式 0 输出，PB 口工作于方式 0 输入，PC 口的 PC7 ~ PC4 为输出，PC 口的 PC3 ~ PC0 为输入。

依题意，按照图 9-3，设置命令寄存器的 D7 = 1，D6D5 = 00，D4 = D3 = 0，D2 = 0，D1 = D0 = 1；则 8255 的控制字为 10000011B = 83H，初始化程序如下：

```
MOV  DPTR,#0FF7FH ;命令寄存器的地址 FF7FH→DPTR
MOV  A,#87H       ;方式控制字 83H 送 A
MOVX @DPTR,A      ;方式控制字送命令寄存器
```

2. PC 口按位置位/复位控制字

8255 的 PC 口具有对任意一位的位控功能, 即单片机向 8255 命令寄存器写入一个按位置位/复位控制字对 PC 口的某一位进行置 1 或复 0, 而 PC 口的其他位状态不变。PC 口按位置位/复位控制字格式如图 9-4 所示。

例 9-2 设 8255 命令寄存器的地址为 FF7FH, 编写 8255 初始化程序, 使 PC 口的 PC2 引脚置成高电平输出, 其他位不变。

依题意, 按照图 9-4, 控制字设置为 00000101B = 05H, 初始化程序段为

```
MOV  DPTR,#0FF7FH    ;命令寄存器端口地址 FF7FH→DPTR
MOV  A,#05H           ;控制字 05H→累加器 A
MOVX @DPTR,A          ;控制字 05H→命令寄存器,PC2 置 1
```

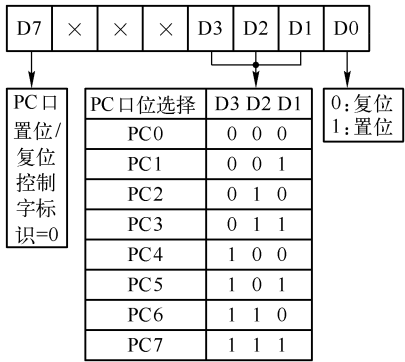


图 9-4 PC 口按位置位/复位控制字格式

3. 8255 的 3 种工作方式

(1) 方式 0——基本输入/输出方式

8255 工作于方式 0 和外设交换数据时, 无须选通信号。如果设置命令寄存器 D6D5 = 00, 则 PA 口和 PC 口的 PC7 ~ PC4 设置成基本输入/输出方式; 如果 D2 = 0, 则 PB 口和 PC 口的 PC3 ~ PC0 设置成基本输入/输出方式, 基本输入/输出方式适用于同步或无条件数据传送应用场合。若作为输出端口, 输出数据被锁存; 若作为输入端口, 输入数据没有锁存仅有缓冲。

(2) 方式 1——选通输入/输出方式

方式 1 是一种采用应答联络的输入/输出工作方式, 只有 PA 口和 PB 口可工作于这种方式, 可以设置为输入或输出。选通输入/输出方式能够实现中断方式向单片机传送 I/O 数据。

1) 方式 1 输入工作过程。PA 口工作于方式 1 输入时, 用 PC5 ~ PC3 作联络信号线。PB 口工作于方式 1 输入时, 用 PC2 ~ PC0 作联络信号线, 剩余的 PC7 和 PC6 工作于方式 0, 通过方式控制字设置为输入或输出。方式 1 输入的各联络信号线如图 9-5 所示, 其中 $\overline{STB}$ 与 IBF 为一对应答联络信号, 各联络信号线的功能如下:

$\overline{STB}$: 选通输入端, 低电平有效。该信号由外设输入, 当 $\overline{STB}$ 有效时, 将外设发送至端口的数据送入 8255 的相应输入缓冲器。PA 口 $\overline{STB}_A$ 使用 PC4 引脚, PB 口 $\overline{STB}_B$ 使用 PC2 引脚。

IBF: 输入缓冲器满指示信号, 输出高电平有效。当 $\overline{STB}$ 信号的下降沿时 IBF 有效, 表示 8255 输入缓冲器已满, 数据尚未被单片机取走, 外设不能送新的数据。PA、PB 口的输入缓冲器满信号 IBF_A 和 IBF_B 分别由 PC5 和 PC1 输出。

INTR: 中断请求信号, 输出高电平有效。在中断允许 INTE = 1 及 IBF = 1 条件下, 由 $\overline{STB}$ 信号的上升沿产生中断请求。PA 口的 INTR_A 由 PC3 输出, PB 口的 INTR_B 由 PC0 输出。

INTE: 中断允许信号, INTE = 1 允许中断, INTE = 0 禁止中断, 用 PC 口按位控制字控制对应位进行置位/复位。

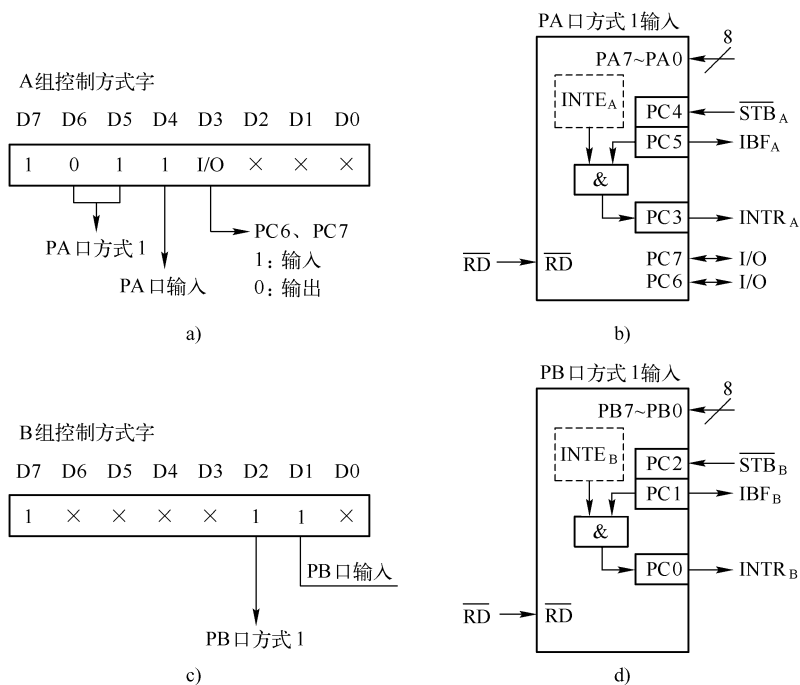


图 9-5 方式 1 输入时的应答联络信号

- a) A 组工作方式控制字 b) PA 口工作于方式 1 输入
c) B 组工作方式控制字 d) PB 口工作于方式 1 输入

INTE_A: PA 口输入中断允许信号, 控制 PA 口是否允许中断, 由 PC4 的置位/复位来控制。

INTE_B: PB 口输入中断允许信号, 控制 PB 口是否允许中断, 由 PC2 的置位/复位来控制。

方式 1 输入过程如图 9-6 所示, 以 PA 口的方式 1 输入为例。

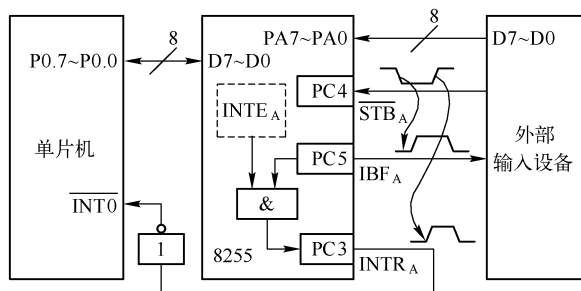


图 9-6 PA 口方式 1 输入工作过程示意图

- 当外设准备好数据并送到 8255 的 PA 口时, 外设自动在 STB_A 上向 8255 发送一个低电平选通信号。
- 8255 收到 STB_A 信号后, 将 PA7 ~ PA0 输入的数据锁存到 8255 的数据输入缓冲器中, 然后 IBF_A 变为高电平, 向外设发送输入缓冲器满的信号, 表示已收到传来的数据,

但禁止外设再送下一个数据。在允许中断的情况下，由 INTR_A 向单片机发中断请求，通知单片机取走数据。

- 在中断方式下，单片机响应中断，在执行中断服务程序时，将 PA 口的输入缓冲器数据取走，撤销 INTR_A 的中断请求，缓冲器空并设置 $\text{IBF}_A = 0$ ，通知外设可送下一个数据到 8255，可以进入下一个传输数据的过程。

2) 方式 1 输出工作过程。8255 工作于方式 1 输出时，各端口应答联络信号如图 9-7 所示。 $\overline{\text{OBF}}$ 和 $\overline{\text{ACK}}$ 构成一对应答联络信号，各应答联络信号含义如下：

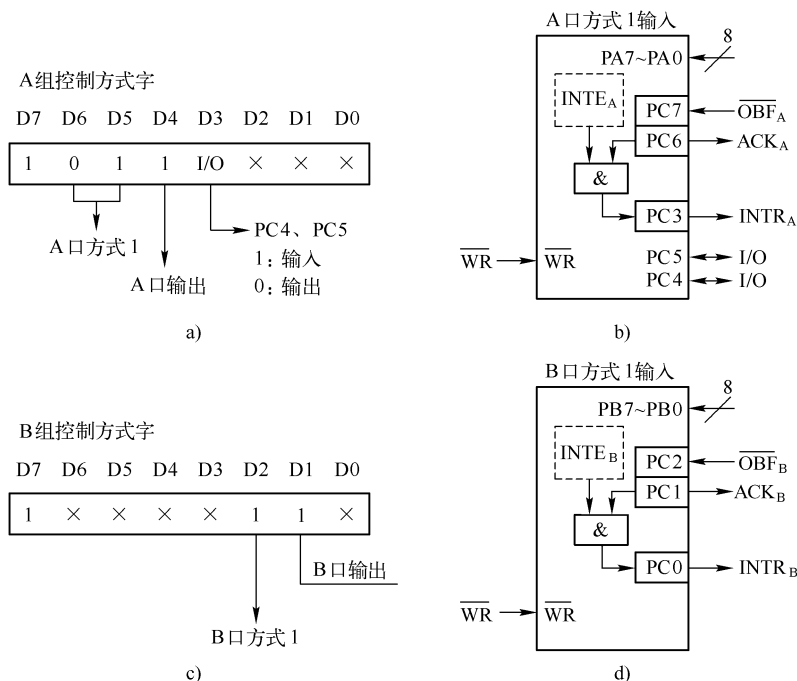


图 9-7 方式 1 输出时的应答联络信号

- a) A 组工作方式控制字 b) PA 口工作于方式 1 输出
c) B 组工作方式控制字 d) PB 口工作于方式 1 输出

$\overline{\text{OBF}}$ ：端口输出缓冲器满信号，低电平有效。表示单片机给 8255 端口的输出缓冲器已写入数据，外设可以将数据取走。PA、PB 端口的 $\overline{\text{OBF}}$ 信号分别由引脚 PC7 及 PC1 输出。

$\overline{\text{ACK}}$ ：外设应答输入信号，低电平有效。表示外设已从端口取走数据。PA、PB 端口的 $\overline{\text{ACK}}$ 信号分别由引脚 PC6 及 PC2 输出。

INTR ：中断请求信号，高电平有效。表示该数据已被外设取走，向单片机发出中断请求，如果单片机响应该中断，则可在中断服务程序中向 8255 写入要输出的下一数据。

INTE_A ：PA 口输出中断允许信号，控制 PA 口是否允许中断，由 PC6 的置位/复位来控制。

INTE_B ：PB 口输出中断允许信号，控制 PB 口是否允许中断，由 PC2 的置位/复位来控制。方式 1 输出工作过程如图 9-8 所示。以 PB 口的方式 1 输出为例。

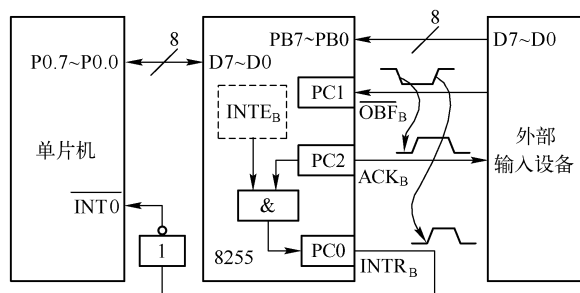


图 9-8 B 口方式 1 输出工作过程示意图

- 单片机向 8255 的 PB 口发送数据，8255 收到后置 $\overline{\text{OBF}}_B = 0$ ，通知外设取数据。
- 外设收到 $\overline{\text{OBF}}_B = 0$ 信号后，从 PB 口取走数据，使 $\overline{\text{ACK}}_B$ 变为低电平，通知 8255 外设已收到数据，输出缓冲器空，8255 使 $\overline{\text{OBF}}_B$ 信号变为高电平。
- 8255 查询到 $\overline{\text{OBF}}_B$ 为高电平时，如果 PB 口中断允许 $\text{INTE}_B = 1$ ，则 INTR_B (PC0) 被置为高电平向单片机发出中断请求。单片机响应 INTR_B 中断请求后通过中断服务程序把下一个输出数据送到 PB 口。重复上述过程，完成一次数据的输出。

(3) 方式 2——选通双向输入/输出方式

只有 PA 口可以工作于方式 2，且要占用 PC 口的 5 根 I/O 信号线作为输入或输出的联络信号。在该工作方式下，PA7 ~ PA0 的 I/O 信号线既可以输入也可以输出，是方式 1 输入和方式 1 输出的组合。当输入数据时，PA 口受 $\overline{\text{STB}}$ (PC4)、IBF (PC5) 信号控制，工作过程与方式 1 输入时相同；当输出数据时，PA 口受 $\overline{\text{OBF}}$ (PC7)、 $\overline{\text{ACK}}$ (PC6) 信号控制，工作过程与方式 1 输出时相同。

9.2.3 8255 与 51 单片机的接口及应用

8255 与 51 单片机的接口和扩展数据存储器与单片机连接电路类似，不同点主要在于 8255 有一个需要直接与单片机控制总线 RST 信号线连接的输入复位信号 $\overline{\text{RESRT}}$ ，而输入的地址线仅只有 A1、A0，单片机与 8255 的接口电路如图 9-9 所示。

对于 8255 与 51 单片机接口的编址，由于 8255 只有 $\overline{\text{CS}}$ 、A1 和 A0 的引脚能够与单片机地址总线连接，图 9-9 的单片机应用系统外部扩展只使用了 8 位地址线，应用线选法确定 8255 的 PA、PB、PC 口及命令寄存器的地址，8255 的 $\overline{\text{CS}}$ 引脚与地址锁存器 74LS373 的 Q7(A7) 连接，A1 和 A0 分别与 Q1(A1) 和 Q0(A0) 连接，A6 ~ A2 地址线不与 8255 发生连接关系，为 8255 芯片编址的无关地址位，编址时，每个无关地址位可以取 0，也可以取 1，如果无关地址位全部取“1”，则 8255 的 3 个端口和命令寄存器地址分别为 7CH、7DH、7EH、7FH。

如果图 9-9 的单片机应用系统外部扩展使用了 16 位地址线，则 A15 ~ A8、A6 ~ A2 不与 8255 发生连接关系，都是 8255 芯片编址的无关地址位，如果无关地址位全部取“1”，则 8255 的 3 个端口和命令寄存器地址分别为 FF7CH、FF7DH、FF7EH、FF7FH。

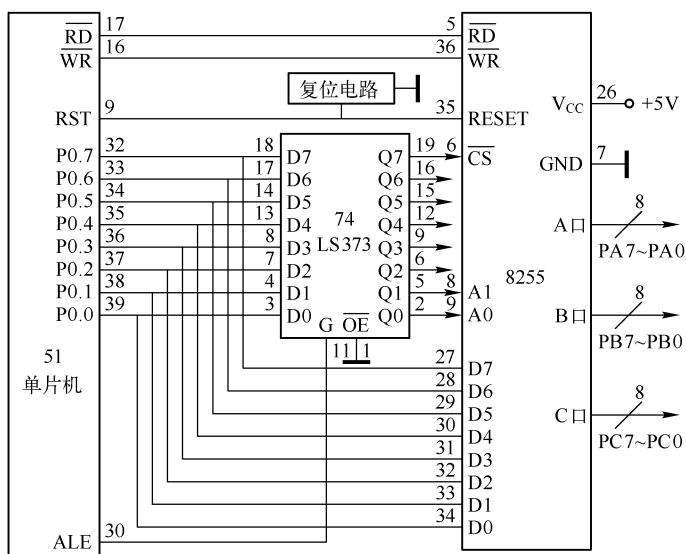


图 9-9 51 单片机与 8255 的接口电路

例 9-3 根据图 9-9 电路，要求从 8255 的 PC 口的 PC3 输出一个正脉冲，试编写该功能的子程序。

分析：假设单片机应用系统外部扩展使用了 16 位地址线，PC3 输出低电平的控制字为 06H = 00000110B；PC3 输出高电平的控制字为 07H = 00000111B，子程序如下：

```
MAICH:MOV    DPTR,# 0FF7FH
          MOV    A,#06H
          MOVX   @ DPTR,A
          MOV    A,#07
          MOVX   @ DPTR,A
          MOV    A,#06H
          MOVX   @ DPTR,A
          RET
```

例 9-4 根据图 9-9 电路，假设 PA 口接一组开关，PB 口接一组指示灯，将单片机工作寄存器 R7 的内容送指示灯，将开关状态读入工作寄存器 R2，试编写实现该功能的子程序。

分析：依题意，PA 口、PB 口均为不需应答联络信号的基本输入/输出，故可初始化 8255 的 PA 口为方式 0 输入，PB 口为方式 0 输出，PC 口的输入/输出方式没有要求，控制字相应位置 0，因此 8255 工作方式控制字为 10010000B = 90H。假设单片机应用系统外部扩展只使用了 8 位地址线，编制的子程序如下：

```
ZHSHD:MOV    R0,#7FH          ;命令寄存器地址→R0
          MOV    A,#90H        ;8255 工作方式控制字 90H→累加器 A
          MOVX   @ R0,A        ;方式控制字写入 8255 命令寄存器
          MOV    R0,#7CH       ;PA 口地址→R0
          MOVX   A,@ R0        ;读 PA 口开关状态
          MOV    R2,A          ;将 PA 口内容→R2
```

```
MOV    R0,#7DH           ;PB 口地址→R0
MOV    A,R7
MOVX   @R0,A             ;将 R7 内容→PB 口
RET
```

9.3 可编程 8155 并行 I/O 接口芯片的扩展

9.3.1 8155 芯片介绍

8155 (81C55) 是 Intel 公司生产的可编程接口芯片，具有 40 个引脚，与单片机的接口简单，功能较强，是单片机应用系统中广泛使用的 I/O 接口芯片。

1. 8155 芯片内部结构

8155 芯片的内部逻辑结构如图 9-10 所示，它主要由以下三部分构成：

- 1) I/O 接口：两个可编程的 8 位并行 I/O 端口 A（A 口：PA7 ~ PA0）、端口 B（B 口：PB7 ~ PB0）和一个可编程的 6 位并行 I/O 端口 C（C 口：PC5 ~ PC0）。
 - 2) 随机存储器：256 B 的静态 RAM。
 - 3) 定时/计数器：1 个 14 位二进制减法计数器，可以用于定时、计数或分频。
- 8155 内部还有一个命令/状态寄存器，命令寄存器只能够写入，状态寄存器只能够读出。

2. 8155 引脚功能

双列直插式 DIP 封装的 8155 芯片外部 40 个引脚如图 9-11 所示，各引脚功能定义如下：

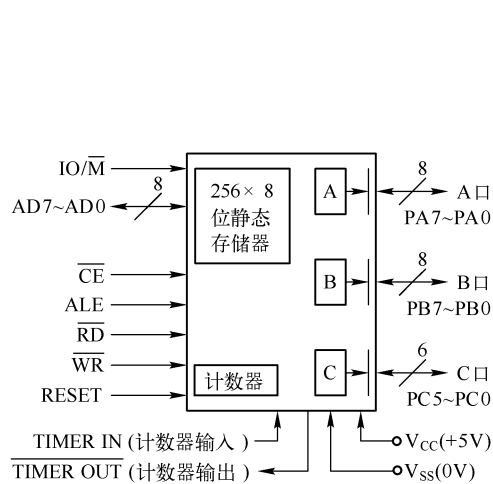


图 9-10 8155 芯片逻辑结构图

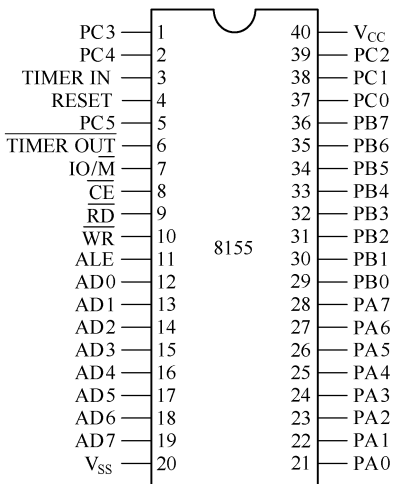


图 9-11 8155 芯片引脚结构图

- 1) RESET：复位信号输入端，高电平有效，芯片复位后，3 个 I/O 端口均被置为输入工作方式。
- 2) CE：片选信号线，低电平有效，表示 8155 芯片被选中。
- 3) RD：读信号线，低电平有效，用来读出 8155 的 I/O 端口数据、状态寄存器的状态信息或 8155 的 RAM 中某单元内容的读控制信号。

4) $\overline{\text{WR}}$: 写信号线, 低电平有效, 用来向 8155 的 I/O 端口、命令寄存器、定时/计数器端口或 RAM 某单元写入数据的写控制信号。

5) $\text{IO}/\overline{\text{M}}$: 8155 的 RAM 存储器和端口选择线。当 $\text{IO}/\overline{\text{M}} = 0$ 时, 则选择 8155 的片内 RAM, AD7 ~ AD0 上的地址信号为 8155 中 RAM 单元的地址 (FFH ~ 00H); 当 $\text{IO}/\overline{\text{M}} = 1$ 时, 选择 8155 的端口, AD7 ~ AD0 上的地址信号为 8155 的端口地址。

6) ALE: 允许地址锁存信号。8155 内部设有地址锁存器, 在 ALE 的下降沿将单片机 P0 口输出的低 8 位地址信息、片选信号 $\overline{\text{CE}}$ 及 $\text{IO}/\overline{\text{M}}$ 信号都锁存到 8155 内部锁存器。

7) AD7 ~ AD0: 三态地址/数据总线。直接与单片机的低 8 位地址/数据总线 (P0 口) 相连, 单片机与 8155 之间的地址、数据、命令与状态信息都是通过这个总线口传送。作为地址总线时将根据输入的 $\text{IO}/\overline{\text{M}}$ 信号状态决定是存储器的 8 位地址还是端口地址; 作为数据总线时由控制信号 $\overline{\text{WR}}$ 和 $\overline{\text{RD}}$ 来决定数据是写入还是读出。

8) PA7 ~ PA0: 8 位通用 I/O 线, 用于传送 A 口上的外设数据, 数据传送方向由写入 8155 的控制字决定。

9) PB7 ~ PB0: 8 位通用 I/O 线, 用于传送 B 口上的外设数据, 数据传送方向由写入 8155 的控制字决定。

10) PC5 ~ PC0: 数据/控制线, 共有 6 条, 在基本 I/O 方式下, 用作传送 I/O 数据; 在选通 I/O 方式下, 用作 A 口和 B 口的控制信号线。

11) TIMER IN: 定时/计数器的计数脉冲输入线, 脉冲上跳沿对 14 位计数器进行减 1 操作。

12) TIMER OUT: 定时/计数器输出线, 当 14 位计数器减为 0 时, 在该引线上输出脉冲或方波, 输出脉冲或方波与所选的定时/计数器工作方式有关。

3. 8155 端口及 RAM 单元地址编码

在单片机应用系统中, 如果系统为 16 位地址, 8155 的片选线信号作为最高地址位参与编址。当 $\overline{\text{CE}} = 0$, $\text{IO}/\overline{\text{M}} = 0$ 时, 选中 8155 片内 RAM, 8155 片内 RAM 作为单片机片外 RAM 使用, 其 RAM 的低 8 位地址范围为 FFH ~ 00H; 当 $\overline{\text{CE}} = 0$, $\text{IO}/\overline{\text{M}} = 1$ 时, 选中 8155 的端口, 8155 内部共有 6 个端口, 需要单片机低 8 位地址中的 A2 ~ A0 的不同组合的编码来加以区分, 8155 端口低 8 位地址由 AD7 ~ AD0 确定。

8155 的高 8 位地址由片选线信号 $\overline{\text{CE}}$ 与 51 单片机地址总线的连接方式所决定。

假设将单片机低 8 位地址 (A7 ~ A0) 中的无关地址位置 1, 端口地址及 RAM 地址编码见表 9-2。

表 9-2 8155 的 6 个端口及 RAM 地址分配

控制信号线		AD7 ~ AD0								所选端口及其低 8 位地址 (地址无关位设置为 1)
$\overline{\text{CE}}$	$\text{IO}/\overline{\text{M}}$	A7	A6	A5	A4	A3	A2	A1	A0	
0	1	×	×	×	×	×	0	0	0	命令/状态寄存器 (F8H)
0	1	×	×	×	×	×	0	0	1	A 口 (F9H)
0	1	×	×	×	×	×	0	1	0	B 口 (FAH)
0	1	×	×	×	×	×	0	1	1	C 口 (FBH)

(续)

控制信号线		AD7 ~ AD0								所选端口及其低 8 位地址 (地址无关位设置为 1)
$\overline{\text{CE}}$	IO/ $\overline{\text{M}}$	A7	A6	A5	A4	A3	A2	A1	A0	
0	1	×	×	×	×	×	1	0	0	定时/计数器的低 8 位 T_L (FCH)
0	1	×	×	×	×	×	1	0	1	定时/计数器的高 6 位 T_H (FDH)
0	0	×	×	×	×	×	×	×	×	RAM 单元 (FFH ~ 00H)

9.3.2 8155 的 I/O 端口的工作方式

8155 的 A 口、B 口可工作于基本 I/O 方式或选通 I/O 方式。C 口可工作于基本 I/O 方式，也可作为 A 口、B 口在选通工作方式时的状态控制信号线。

1. 8155 的控制字

8155 芯片的 I/O 端口和定时/计数器的工作方式通过对 8155 的命令寄存器写入控制字来实现。命令寄存器只能写入，不能读出，命令寄存器的控制字格式见表 9-3。

表 9-3 8155 控制字格式

D7	D6	D5	D4	D3	D2	D1	D0
TM2	TM1	IEB	IEA	PC2	PC1	PB	PA

1) TM2(D7)、TM1(D6)位：设置定时/计数器的操作。

TM2、TM1 = 00：空操作，不影响计数操作。

TM2、TM1 = 01：停止定时/计数器计数。

TM2、TM1 = 10：若定时/计数器正在计数，计数值减为 0 时停止计数。

TM2、TM1 = 11：装入方式和计数值后，立即启动定时/计数器，若定时/计数器正在运行，则达到计数值后，按新的方式和计数初值予以启动。

2) IEB(D5)位：确定端口 B 以选通输入/输出方式工作时是否允许中断请求。

“0” — 禁止；“1” — 允许。

3) IEA(D4)位：确定端口 A 以选通输入/输出方式工作时是否允许中断请求。

“0” — 禁止；“1” — 允许。

4) PC2(D3)、PC1(D2)位：设置端口 A、B、C 的工作方式。

PC2、PC1 = 00：方式 1，A 口、B 口作为基本输入/输出，C 口作为输入。

PC2、PC1 = 11：方式 2，A 口、B 口作为基本输入/输出，C 口作为输出。

PC2、PC1 = 01：方式 3，A 口作为选通输入/输出，B 口作为基本输入/输出，PC0 作为 A 口中断信号线 (AINTR)，PC1 作为 A 口缓冲器满信号线 (ABF)，PC2 作为 A 口选通信号线 ($\overline{\text{ASTB}}$)，PC5 ~ PC3 作为输出。

PC2、PC1 = 10：方式 4，A 口、B 口作为选通输入/输出，PC0 作为 A 口中断信号线 (AINTR)，PC1 作为 A 口缓冲器满信号线 (ABF)，PC2 作为 A 口选通信号线 ($\overline{\text{ASTB}}$)；PC3 作为 B 口中断信号线 (BINTR)，PC4 作为 B 口缓冲器满信号线 (BBF)，PC5 作为 B 口选通信号线 ($\overline{\text{BSTB}}$)。

5) PB(D1): 设置端口 B 的输入/输出工作方式。

“0” —输入; “1” —输出。

6) PA(D0): 设置端口 A 的输入/输出工作方式。

“0” —输入; “1” —输出。

2. 8155 状态字格式

8155 片内的 8 位状态寄存器用于锁存 A 口、B 口的状态标志和定时/计数器的状态标志, 最高位为任意位, 其端口地址与命令寄存器地址相同, 状态寄存器的内容只能读出不能写入。状态寄存器的格式见表 9-4。

表 9-4 8155 状态寄存器格式

D7	D6	D5	D4	D3	D2	D1	D0
—	TIMER	INTEB	BBF	INTRB	INTEA	ABF	INTRA

1) TIMER(D6)位: 定时/计数器中断标志。

“0” —在读状态字后或硬件复位后; “1” —定时/计数器溢出, 中断发生。

2) INTEB(D5)位: B 口允许中断标志。

“0” —禁止; “1” —允许。

3) BBF(D4)位: B 口缓冲器状态标志。

“0” —空; “1” —满。

4) INTRB(D3)位: B 口中断请求标志。

“0” —无中断请求; “1” —有中断请求。

5) INTEA(D2)位: A 口允许中断标志。

“0” —禁止; “1” —允许。

6) ABF(D1)位: A 口缓冲器状态标志。

“0” —空; “1” —满。

7) INTRA(D0)位: A 口中断请求标志。

“0” —无中断请求; “1” —有中断请求。

3. 8155 工作方式

8155 可工作于存储器方式和 I/O 方式。

(1) 存储器方式

$\overline{\text{IO/M}}=0$, $\overline{\text{CE}}=0$, 8155 工作于存储器方式, 用于对 8155 片内 RAM 的 256 个单元进行读/写操作, 单片机可通过 AD7 ~ AD0 上的地址选择 RAM 存储器中的任意单元进行读和写。

(2) I/O 方式

$\overline{\text{IO/M}}=1$, $\overline{\text{CE}}=0$, 8155 工作于 I/O 方式, I/O 方式分为基本输入/输出和选通输入/输出两种方式。在 I/O 方式下, 8155 可选择片内任意端口寄存器进行读/写, 端口地址由 A2、A1 和 A0 三位决定。

1) 基本 I/O 方式。端口 A、端口 B 和端口 C 均用作输入/输出, 通过设置控制字决定。例如, 若把 00001101B = 0DH 的控制字写入 8155 命令寄存器, 则 8155 的 B 口设定为输入方式, A 口和 C 口设定为输出方式。

2) 选通 I/O 方式。A 口和 B 口都可独立工作于这种方式，常用作数据口，C 口用作 A 口和 B 口的应答联络信号控制。C 口各位应答联络信号线的定义见表 9-5。

表 9-5 C 口在选通 I/O 方式下各位的定义

C 口	基本 I/O 方式		选通 I/O 方式	
	方式 1	方式 2	方式 3	方式 4
PC0	输入	输出	AINTR (A 口中断请求)	AINTR (A 口中断请求)
PC1	输入	输出	ABF (A 口缓冲器满信号)	ABF (A 口缓冲器满信号)
PC2	输入	输出	$\overline{\text{ASTB}}$ (A 口选通信号)	$\overline{\text{ASTB}}$ (A 口选通信号)
PC3	输入	输出	输出	BINTR (B 口中断请求)
PC4	输入	输出	输出	BBF (B 口缓冲器满信号)
PC5	输入	输出	输出	$\overline{\text{BSTB}}$ (B 口选通信号)

选通 I/O 方式又可分为选通 I/O 数据输入和选通 I/O 数据输出两种方式。

① 选通 I/O 数据输入。A 口和 B 口都可设定为选通数据输入工作方式。若设置控制字中 D0(PA) = 0 且 D3(PC2)、D2(PC1) = 01 (工作方式 3) 或 10 (工作方式 4)，则 A 口设定为该工作方式；若控制字中 D1(PB) = 0 且 D3、D2 = 10 (工作方式 4)，则 B 口设定为该工作方式。以 A 口为例，说明 8155 选通 I/O 数据输入工作过程，A 口选通数据输入工作过程示意图如图 9-12 所示。

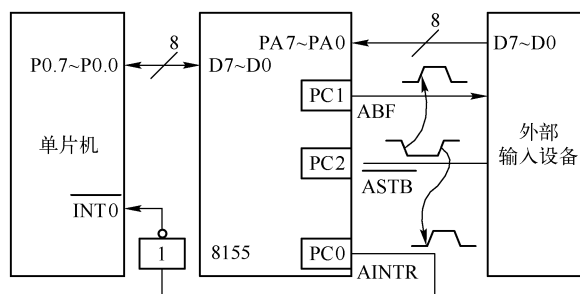


图 9-12 A 口选通 I/O 输入工作过程示意图

图 9-12 中， $\overline{\text{ASTB}}$ 与 ABF 为一对应答联络信号，具体输入过程如下：

- 当外设向 8155 的 A 口发送一个数据时，外设自动在 $\overline{\text{ASTB}}$ 上向 8155 发送一个低电平选通信号。
- 8155 收到 $\overline{\text{ASTB}}$ 低电平信号后，把输入的数据存入到 A 口的输入数据缓冲/锁存器，然后使输出应答线 ABF 变为高电平，通知外设 A 口已收到数据。
- $\overline{\text{ASTB}}$ 由低电平变为高电平，AINTR(PC0) 被置为高电平，向单片机发出中断请求。
- 单片机响应 8155 的中断请求后，读取 A 口的输入数据。当输入数据被单片机读走后，8155 撤销 AINTR 上的中断请求，并使 ABF 变低，通知外设可传送下一个输入数据。

② 选通 I/O 数据输出。A 口和 B 口都可设定为选通数据输出工作方式。若设置控制字中 D0(PA) = 1 且 D3(PC2)、D2(PC1) = 01 (工作方式 3) 或 10 (工作方式 4)，则 A 口设定为该工作方

式；若控制字中 D1(PB) = 1 且 D3、D2 = 10（工作方式4），则 B 口设定为该工作方式。以 A 口为例，说明 8155 选通 I/O 数据输出工作过程，A 口选通数据输出工作过程示意图如图 9-13 所示。

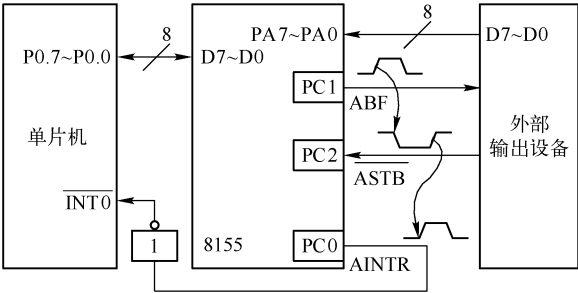


图 9-13 PA 口选通 I/O 输出工作过程示意图

图 9-13 中， $\overline{\text{ASTB}}$ 与 ABF 为一对应答联络信号，具体输出过程如下：

- 单片机向 8155 端口 A 发送数据，8155 收到后置 ABF = 1，通知外设数据已在 A 口的 PA7 ~ PA0 上。
- 外设收到 ABF 高电平信号后，从 A 口取走数据，并置 $\overline{\text{ASTB}}$ = 0，通知 8155 外设已收到数据。
- $\overline{\text{STBA}}$ 由低电平变为高电平，8155 将 AINTR(PC0)置为高电平，向单片机发出中断请求。
- 单片机响应 8155 中断请求后，便可通过中断服务程序把下一个输出数据送到 A 口。重复上述过程，完成所有数据的输出。

4. 8155 的定时/计数器

8155 内部的定时/计数器是一个 14 位的减法计数器，它对 TIMER IN 端输入脉冲进行减 1 计数，当计数结束（即减 1 计数至“0”）时，由 TIMER OUT 端输出方波或脉冲。当 TIMER IN 接外部脉冲时，为计数方式；连接单片机振荡时钟信号时，可作为定时方式。

定时/计数器由两个 8 位定时/计数器的寄存器 T_H 和 T_L 构成， T_H 和 T_L 的低 14 位组成 14 位 - 1 计数器， T_H 剩下的两个高位（M2、M1）用于定义输出方式。其格式见表 9-6 和表 9-7。

表 9-6 8155 计数器计数长度高 8 位 T_H (05H)

D7	D6	D5	D4	D3	D2	D1	D0
M2	M1	T13	T12	T11	T10	T9	T8

表 9-7 8155 计数器计数长度低 8 位 T_L (04H)

D7	D6	D5	D4	D3	D2	D1	D0
T7	T6	T5	T4	T3	T2	T1	T0

表 9-6 和表 9-7 中，T13 ~ T0 为计数器的计数位；M2、M1 设置 8155 定时/计数器的输出方式，共有 4 种工作方式，其对应的引脚输出波形如图 9-14 所示。

使用 8155 的定时/计数器时，首先设置其工作方式和计数初值，然后启动定时/计数器，启动和停止定时/计数器都是通过设置控制字来实现。

M2	M1	方式	定时/计数器输出波形
0	0	单方波	
0	1	连续方波	
1	0	单脉冲	
1	1	连续脉冲	

图 9-14 8155 计数器工作方式及其引脚输出波形

启动定时/计数器的步骤为：

- 1) 根据要求确定计数初值，即 14 位减法计数器的计数初值。
- 2) 设置定时/计数器工作方式并按先高 8 位后低 8 位顺序将计数初值写入计数寄存器。
- 3) 向命令寄存器最高两位 TM2(D7)、TM1(D6) 写入 11，启动定时/计数器。

停止定时/计数器工作的方法为：

- 1) 向命令寄存器最高两位 TM2(D7)、TM1(D6) 写入 01，使定时/计数器立即停止计数。
- 2) 向命令寄存器最高两位 TM2(D7)、TM1(D6) 写入 10，定时/计数器溢出时停止计数。

9.3.3 8155 与 51 单片机的接口及应用

8155 可以直接与 51 单片机总线连接，接口电路如图 9-15 所示。图 9-15 的 51 单片机与 8155 的接口电路和数据存储器、8255 与 51 单片机接口电路比较，由于 8155 内部有地址锁存器，所以 8155 的接口电路不需要与 74LS373 芯片连接，单片机 P0 口直接与 8155 的 AD7~AD0 相连，既作为低 8 位地址总线，又可作为数据总线，但 8155 的地址锁存信号 ALE 引脚需要与单片机的 ALE 引脚连接。

图 9-15 中，8155 的片选信号 $\overline{CE}$ 作为最高位地址线与单片机的 P2.7 引脚连接，8155 的 RAM 存储器和端口选择线 $\overline{IO/\overline{M}}$ 与 P2.0 相连。当 P2.7 = 0，P2.0 = 0 时，则访问 8155 的片内 RAM；当 P2.7 = 0，P2.0 = 1 时，则访问 8155 端口。根据图 9-15 可知，假设 8155 未用的无关地址位置为 1，则 8155 的命令/状态寄存器口地址为 7FF8H，A 口、B 口和 C 口地址分别为 7FF9H、7FFAH、7FFBH，RAM 地址空间范围为 7E00H~7EFFH，定时/计数器低 8 位和高 8 位地址分别为 7FFCH 和 7FFDH。

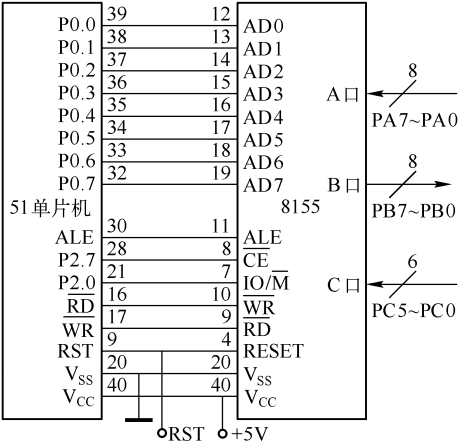


图 9-15 51 单片机与 8155 的接口电路

例 9-5 根据图 9-15 电路，将 51 单片机片内 RAM 30H~3FH 单元的内容，按照地址顺序传送到 8155 片内 RAM 的 7E40H~7E4FH 单元中。设置 8155 的 A 口为基本输入方式，B 口为基本输出方式，C 口为输入方式，定时/计数器对输入脉冲进行 48 分频，输出方波。

设置 8155 的控制字见表 9-8，控制字为 11000010B = C2H。

表 9-8 例 9-5 的 8155 控制字

TM2	TM1	IEB	IEA	PC2	PC1	PB	PA
1	1	0	0	0	0	1	0

根据 8155 定时/计数器的计数与控制输出方式格式，48 分频的计数初值为 48 = 0030H，还需要在最高两位（M2、M1）加上控制定时/计数器输出方式的 01（连续方波），则定时/计数器的计数初值及控制定时/计数器输出方式的 16 位参数应为 0100 0000 0011 0000B =

4030H, 见表 9-9 和表 9-10。

表 9-9 例 9-5 的 8155 定时/计数器的 T_H (05H) 表 9-10 例 9-5 的 8155 定时/计数器的 T_L (04H)

M2	M1	T13	T12	T11	T10	T9	T8
0	1	0	0	0	0	0	0

T7	T6	T5	T4	T3	T2	T1	T0
0	0	1	1	0	0	0	0

程序可以分为三个模块：数据块传输、设置定时/计数器的计数初值和 8155 的控制字。

```
ORG      1000H
Li9_5:MOV    R0,#30H           ;R0 为地址指针,指向单片机片内 RAM 30H 单元
MOV      DPTR,#7E40H          ;DPTR 数据指针指向 8155 内部 RAM 7E40H 单元
LP:MOV      A,@R0              ;((R0))→累加器 A
MOVX     @DPTR,A              ;(A)→((DPTR));8155 内部 RAM 单元
INC      DPTR                  ;指向下一个 8155 内部 RAM 单元
INC      R0                    ;指向单片机下一个内部 RAM 单元
CJNE     R0,#40H,LP            ;数据未传送完返回。以上为数据块传输程序段
MOV      DPTR,#7FFCH          ;DPTR 数据指针指向  $T_L$ 
MOV      A,#30H                ;分频系数  $48 = 30H$ 
MOVX     @DPTR,A              ;装入低 8 位计数初值
INC      DPTR                  ;DPTR 数据指针指向  $T_H$ 
MOV      A,#40H                ;设定定时/计数器工作方式连续方波( $40H = 0100\ 0000\ B$ )
MOVX     @DPTR,A              ;装入定时/计数器工作方式及高 6 位计数初值
;以上为定时/计数器时间参数初值设置的程序段
MOV      DPTR,#7FF8H          ;DPTR 数据指针指向命令寄存器
MOV      A,#0C2H              ;设定 A 口、B 口、C 口工作方式( $C2H = 1100\ 0010\ B$ )
MOVX     @DPTR,A              ;启动定时/计数器
RET
END
```

例 9-6 图 9-16 是 51 单片机扩展 8155 与微型打印机的接口电路，编写打印一个字符“A”的程序。

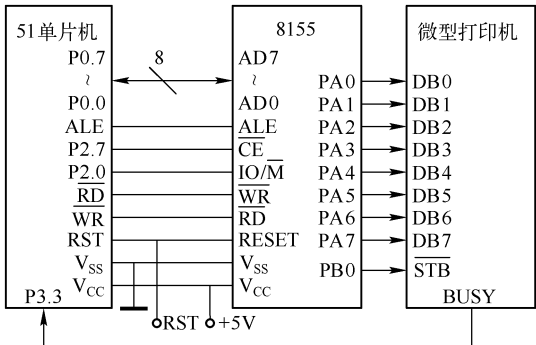


图 9-16 微型打印机与 51 单片机的接口电路

分析：图 9-16 中，BUSY 与 $\overline{STB}$ 为一对应答联络信号。BUSY = 1，表示微型打印机忙；当 51 单片机检测到 P3.3 为“0”，即 BUSY = 0 时，51 单片机向 8155 的 A 口发送“A”的

ASCII 码字符，并向 PBO 发送一个负跳变，微型打印机 $\overline{\text{STB}}$ 输入端接收到该负跳变信号，将 8155 的 PA7 ~ PA0 输出的数据锁存并打印。

图 9-16 中，8155 的 $\overline{\text{CE}}$ 片选端与 P2.7 连接， $\text{IO}/\overline{\text{M}}$ 与 P2.0 连接，则 8155 命令寄存器地址为 0100H，A 口地址为 0101H，B 口地址为 0102H，A 口、B 口均为输出状态，微型打印机“BUSY”信号采用查询方式识别，控制字：03H；PC2(D3)、PC1(D2) = 00；方式 1；PB(D1) = PA(D0) = 1，A 口、B 口为基本输出方式。

```

START: MOV    DPTR, #0100H    ;命令寄存器地址值→DPTA
        MOV    A, #03H        ;设置 8155 为方式 1, A 口、B 口为基本输出方式
        MOVX   @DPTR, A       ;控制字→命令寄存器
WAIT:   JB     P3.3, WAIT      ;查询 BUSY, 如果 BUSY = 1, 等待
        INC    DPTR           ;DPTR 地址指针指向 A 口: 0101H
        MOV    A, #'A'        ;“A”的 ASCII 码字符代码→累加器 A
        MOVX   @DPTR, A       ;A 口锁存输出“A”字符
        INC    DPTR           ;DPTR 地址指针指向 B 口: 0102H
        MOV    A, #0
        MOVX   @DPTR, A       ;向 PBO 发送一个选通信号
        NOP
        NOP                   ;微型打印机锁存“A”并开始打印
        MOV    A, #1
        MOVX   @DPTR, A       ;向 PBO 输出高电平, 产生一个负脉冲信号
        RET

```

习题

9-1 I/O 接口和 I/O 端口有什么区别？I/O 接口的功能是什么？

9-2 常用的 I/O 接口编址有哪两种方式？它们各有什么特点？51 单片机的 I/O 端口编址采用的是哪种方式？

9-3 简述 8255 的基本组成和功能，8255 芯片编址的特点，编址时片选 $\overline{\text{CS}}$ 端的作用，无关地址位如何处理。

9-4 8255 分别有哪几种工作方式？怎样进行选择？

9-5 编写程序，将 8255 的 PC 口的 PC7 置“0”，PC4 置“1”，已知 8255 各端口的地址为 7FFCH ~ 7FFFH。

9-6 根据图 9-9 的接口电路图，试对 8255 进行初始化编程，设置 PA、PB、PC 都是基本输入/输出方式，PA 口为输入，PB 口为输出，PC 口的高四位为输出，低四位为输入。

9-7 图 9-9 的接口电路图中，如果无关地址位全部取“0”，试列出 8255 的 PA 口、PB 口、PC 口和命令寄存器的 16 位地址。

9-8 8155 有哪些端口？哪些引脚决定端口的地址？引脚 TIMER IN 和 TIMER OUT 的作用是什么？

9-9 简述 8155 的基本组成和功能，8155 芯片编址的特点，编址时片选 $\overline{\text{CS}}$ 、 $\text{IO}/\overline{\text{M}}$ 的作

用，无关地址位如何处理。

9-10 假设 8155 的 TIMER IN 引脚输入的频率为 4 MHz，问 8155 的最大定时时间是多少？

9-11 试编程对 8155 进行初始化，设 A 口为选通输出，B 口为选通输入，C 口作为控制联络口，并启动定时/计数器按方式 1 工作，工作时间为 10 ms，定时/计数器的计数脉冲频率为单片机的时钟频率 24 分频， $f_{osc} = 12 \text{ MHz}$ 。

9-12 图 9-15 的接口电路图中，如果无关地址位全部取“0”，试列出 8155 的命令寄存器、A 口、B 口和 C 口的 16 位地址。

9-13 如图 9-17 所示，8155 的 A 口用于输出，B 口用于输入，C 口用于输出。

1) 地址无关位取 0 或 1，写出 8155 的各 I/O 口、命令寄存器地址和 8155 片内 RAM 的地址空间范围。

2) 编写 8155 初始化程序。

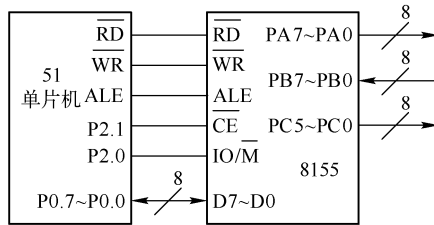


图 9-17 题 9-13 和题 9-15 图

9-14 设在图 9-9 的接口电路图中，8255 的 PA 口外接了 8 个发光二极管电路，低电平时发光二极管亮，应用定时中断编程方法，编写 8 个发光二极管跑马灯的程序。

9-15 设在图 9-17 的接口电路图中，8155 的 B 口外接了 8 个发光二极管电路，低电平时发光二极管亮，应用定时中断编程方法，编写 8 个发光二极管跑马灯的程序。

第 10 章 A - D 和 D - A 转换器的接口

微处理器只能处理二进制数据，而计算机处理的物理量绝大多数都是模拟量，对于微处理器输出的二进制数据需要通过数模转换电路将二进制数据转换为模拟量输出，对于输入的模拟量需要通过模数转换电路将模拟量转换为二进制数据给微处理器进行数据处理。

本章介绍 D - A 转换器和 A - D 转换器的基本原理和技术指标，重点介绍 D - A 转换器和 A - D 转换器与 51 单片机的接口电路，以及 D - A 转换器和 A - D 转换器的编程方法。

10.1 D - A 转换器接口

10.1.1 D - A 转换器原理

1. D - A 转换器基本工作原理

D - A 转换器 (Digital to Analog Converter, 简称 DAC), 又称为数模转换器, 是一种将二进制数字量转换成以参考量为基准的模拟量的转换器。最常见的 D - A 转换器是将并行二进制数字量转换为直流电压或直流电流, 即 D - A 转换器的输出电压 V_O 应与输入的 n 位数字量 M 成正比, 即 $V_O = MV_R$, 其中 V_R 为参考电压。D - A 转换器的输入数字量采用二进制形式, 每个二进制位的权为 2^i , 对应一定大小的模拟量。为了将数字量转换成模拟量, 需要先把数字量的每一个二进制位按权位转换成相应的模拟量, 然后求和得到与数字量成正比的模拟量。所以 D - A 转换器的原理可以总结为“按权展开, 然后相加”。D - A 转换器的类型很多, 常用的是二进制加权电阻网络型 D - A 转换器和 T 型电阻网络型 D - A 转换器。下面以 4 位二进制加权电阻网络型 D - A 转换器为例来说明 D - A 转换器的工作原理, 转换原理图如图 10-1 所示。

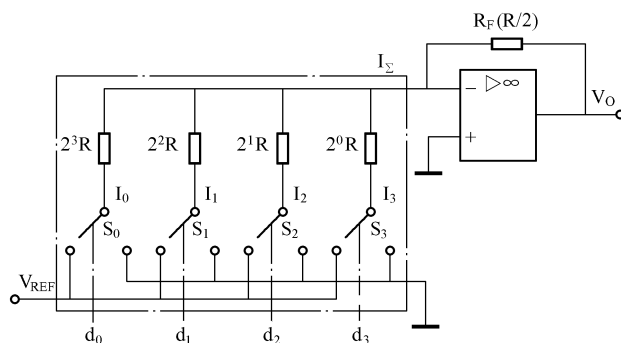


图 10-1 二进制加权电阻网络型 D - A 转换器

由图 10-1 可见, 权电阻网络 D - A 转换器由电阻网络、4 个数码开关和 1 个求和运算放大器组成。其中 V_{REF} 为基准电压, $S_0 \sim S_3$ 为数码开关, 受二进制位状态控制, 相应二进制位为“0”, 表示开关接地; 相应二进制位为“1”, 表示开关接 V_{REF} 。点画线框内电阻 $2^n R$ 称为“权电阻”, 是指二进制数的每一位代表的数值, 电阻值越大, 在“权电阻”上流过的

“权电流”越小，经过运放后得到的 V_0 电压值越小，4 个数码开关的不同组合，将产生 16 个不同的 V_0 电压值。“权电阻”数量越多， V_0 端输出的电压值数量越多，并且电压值之间的差值越小，这就是 D-A 转换器的基本原理。数码开关和电阻网络是 D-A 转换器的基本部分，不同类型的 D-A 转换器，这两部分的结构也各不相同。下面推导图 10-1 所示的二进制加权电阻网络型 D-A 转换器的输入数字量 M 与输出量 V_0 之间的关系。

根据运算放大器“电压虚短”和“电流虚断”的概念，可以得到

$$V_0 = -\frac{V_{REF}}{2^4}(2^3 \times d_3 + 2^2 \times d_2 + 2^1 \times d_1 + 2^0 \times d_0) = -\frac{V_{REF}}{2^4} \times M$$

一般地， n 位二进制加权电阻网络型 D-A 转换器的输出电压为

$$V_0 = -\frac{V_{REF}}{2^n}(2^{n-1} \times d_{n-1} + 2^{n-2} \times d_{n-2} + \cdots + 2^1 \times d_1 + 2^0 \times d_0) = -\frac{V_{REF}}{2^n} \times M$$

由此可见，D-A 转换器的输出电压 V_0 正比于输入数字量 M ，无论 D-A 转换器的内部结构如何，最终得到计算输出电压 V_0 的公式均相同。

2. D-A 转换器的技术指标

(1) 分辨率

分辨率是指输入数字量的最低有效位 (LSB) 发生变化时，所对应的输出模拟量 (电压或电流) 的变化量。它反映了输出模拟量的最小变化值，通常定义为输出满刻度值 FS 与 2^n 之比 (n 为 D-A 转换器的二进制位数)。对于 5 V 的满量程，采用 8 位的 DAC 时，分辨率为 $5 \text{ V}/256 = 19.5 \text{ mV}$ ，即输入的二进制数最低位的变化可引起输出模拟电压的变化量为 19.5 mV ，常用符号 1LSB 表示；当采用 12 位的 DAC 时，分辨率则为 $5 \text{ V}/4096 = 1.22 \text{ mV}$ 。显然，位数越多，分辨率就越高，即 D-A 转换器对输入量变化的敏感程度越高。

(2) 转换精度

转换精度是指满量程时 DAC 实际的模拟输出值与理论值的接近程度。理想情况下，转换精度与分辨率基本一致，位数越多，精度越高。但由于制造工艺等各种因素存在着误差，严格讲，转换精度与分辨率并不完全一致。只要位数相同，分辨率则相同，但相同位数的不同转换器转换精度会有所不同。例如，满量程时理论输出为 10 V，实际输出值在 9.99 ~ 10.01 V 之间，其转换精度为 $\pm 10 \text{ mV}$ 。

(3) 建立时间

建立时间是指输入数字量变化后，输出模拟量稳定到终值误差 $\pm (1/2)\text{LSB}$ 时所经历的时间，是描述 DAC 转换速度快慢的一个重要参数。电流输出型 DAC 的建立时间短。电压输出型 DAC 的建立时间主要取决于运算放大器的响应时间。根据建立时间的长短，可以将 DAC 分成超高速 ($<1 \mu\text{s}$)、高速 ($10 \sim 1 \mu\text{s}$)、中速 ($100 \sim 10 \mu\text{s}$) 和低速 ($\geq 100 \mu\text{s}$) 几档。

3. D-A 转换器的输出及接口形式

D-A 转换器的输出有两种形式：一种是电压输出型；另一种是电流输出型，电流输出型的 D-A 转换器可在其输出端再将电流输出转换为电压输出。

D-A 转换器的接口主要有并行接口和串行接口 (如通用 UART 串行口、 I^2C 串行口和 SPI 串行口等)，因此在选择 D-A 转换器时，要考虑单片机与 D-A 转换器的接口形式。

10.1.2 51 单片机与 DAC0832 的接口与编程

DAC0832 是 8 位电流输出型 D-A 转换器，转换时间为 $1 \mu\text{s}$ ，逻辑电平输入与 TTL 电平

兼容，单一电源供电（+5 ~ +15 V），功耗为 20 mW，可以直接与 51 单片机应用系统总线连接。DAC0832 的主要特性有：数据输入可采用双缓冲、单缓冲或直通方式；电流输出，输出电流线性度可在满量程下调节。

1. DAC0832 的内部结构

DAC0832 的内部结构如图 10-2 所示，它由一个 8 位输入寄存器、一个 8 位 DAC 寄存器、一个 8 位 D - A 转换电路及控制电路构成。8 位输入寄存器用于存放单片机送来的数字量，使输入数字量得到缓冲和锁存，由 $\overline{LE1}$ 控制；8 位 DAC 寄存器用于存放待转换的数字量，由 $\overline{LE2}$ 控制；8 位 D - A 转换电路受 8 位 DAC 寄存器输出的数字量控制，能输出和数字量成正比的模拟电流。DAC0832 可以根据需要连接成两级输入锁存的双缓冲方式或一级输入锁存的单缓冲方式，或完全直通的无缓冲方式。

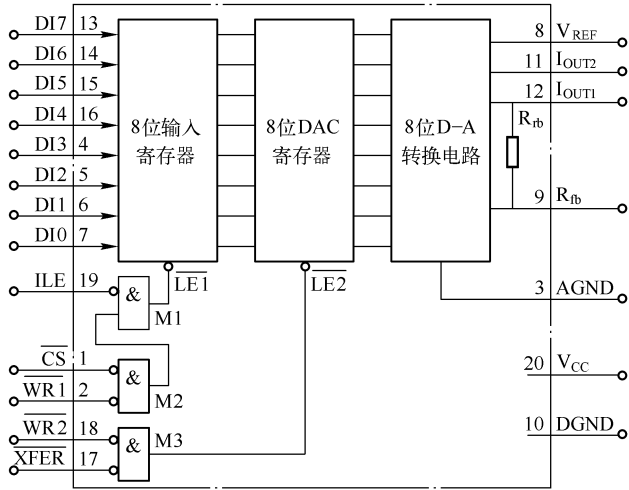


图 10-2 DAC0832 的内部结构

2. DAC0832 的引脚

图 10-3 为 DAC0832 的引脚分布图，各引脚功能定义如下：

1) DI7 ~ DI0：8 位数据输入引脚，其中 DI7 是高位，DI0 是低位。DI7 ~ DI0 与 51 单片机的数据总线 P0 口连接，用于接收 51 单片机送来的待转换为模拟量的数字量。

2) $\overline{CS}$ ：片选端，低电平有效时，表示 DAC0832 芯片被选中。

3) ILE：输入锁存允许信号，高电平有效。

4) $\overline{WR1}$ ：8 位输入寄存器写选通控制，低电平有效。当 $\overline{CS}=0$ ， $ILE=1$ ， $\overline{WR1}=0$ 时，把待转换的数据信号锁存到 8 位输入寄存器中。

5) $\overline{XFER}$ ：数据传送控制，低电平有效。

6) $\overline{WR2}$ ：8 位 DAC 寄存器写选通控制端，低电平有效。当 $\overline{XFER}=0$ ， $\overline{WR2}=0$ 时，8 位输入寄存器中待转换的数据传入 8 位 DAC 寄存器中。

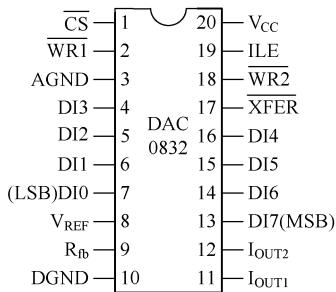


图 10-3 DAC0832 引脚分布

7) I_{OUT1} : 8 位 D-A 转换器电流输出 1 端, 输入数字量全为“1”时, I_{OUT1} 最大, 输入数字量全为“0”时, I_{OUT1} 最小。

8) I_{OUT2} : 8 位 D-A 转换器电流输出 2 端, $I_{OUT1} + I_{OUT2} = \text{常数}$ 。

9) R_{fb} : 外部反馈信号输入端, 为外部运算放大器提供一个反馈电压。根据需要还可以外接另外一个反馈电阻。

10) V_{CC} : 电源输入端, 在 +5 ~ +15 V 范围内。

11) V_{REF} : 参考电压输入端, DAC 要求外部连接一个精密的电源。如果直接与 V_{CC} 连接, 适应于对转换精度要求一般的单片机应用系统。

12) DGND: 数字信号地。

13) AGND: 模拟信号地, 最好与基准电压共地。

3. DAC0832 与 51 单片机的连接

(1) 单缓冲连接方式

DAC0832 与 51 单片机接口的单缓冲连接电路如图 10-4 所示。图 10-4 中, $\overline{WR2}$ 和 $\overline{XFER}$ 直接接地, 由图 10-2 可知, 8 位 DAC 寄存器处于直通状态; $\overline{ILE}$ 接 +5 V, 8 位输入寄存器受控于 $\overline{WR1}$ 和 $\overline{CS}$ 信号, 当 $\overline{CS}$ 为“0”时, 即选中 DAC0832, 在 $\overline{WR1}$ 信号的作用下将 51 单片机数据总线的 8 位数据 $\rightarrow$ 8 位输入寄存器 $\rightarrow$ 8 位 DAC 寄存器 $\rightarrow$ 8 位 D-A 转换电路 $\rightarrow$ 输出。所谓的“单缓冲”是指单片机只需执行一条写操作指令, 即可完成 D-A 转换。单缓冲工作方式适用于只有一路模拟量输出, 或有几路模拟量输出但不要求同步输出的系统。设 DPTR 中的内容为 DAC0832 地址值, A 中的内容为待转换的数据, 执行指令 `MOVX @DPTR, A` 即可实现 DAC0832 的数字到模拟量的转换。

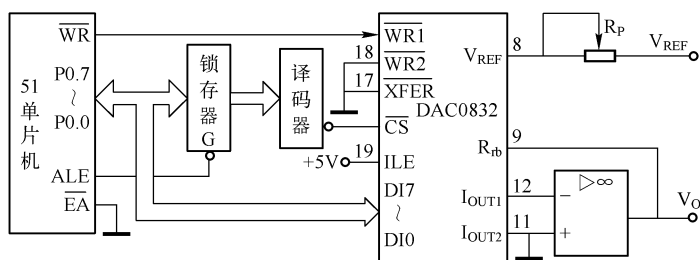


图 10-4 DAC0832 与单片机接口的单缓冲连接电路

例 10-1 试编写 DAC0832 以单缓冲方式输出如图 10-5 所示的锯齿波、三角波的 D-A 转换输出程序。设 DAC0832 的译码地址为 FFFE_H, 51 单片机晶振频率为 12 MHz, $T_a = 100 \text{ ms}$, $T_b = 200 \text{ ms}$ 。

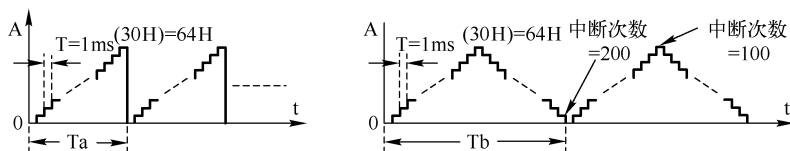


图 10-5 锯齿波和三角波输出波形

1) 锯齿波。按照图 10-5，锯齿波的周期 $T_a = 100\text{ ms}$ ，最大幅值 $(30\text{H}) = 64\text{H} = 100$ ，每个待转换的数字量维持 $T = 1\text{ ms}$ 时间。每间隔 1 ms ，DAC0832 的数据输入值加 1 一次，应用 T0 工作于方式 1 定时 1 ms ，采用定时中断方法编程实现。

按照式 (5-3) 计算 T0 的定时初值：

$$X = 65536 - 1 \times 10^{-3} \times 10^6 = 65536 - 1000 = 64536 = \text{FC18H}$$

```

ORG    0000H
LJMP   START
ORG    000BH           ;定时器/计数器 T0 的中断入口地址
LJMP   TIMER0          ;跳转到定时 1 ms 的中断服务程序 TIMER0
ORG    0030H
START: MOV    TMOD, #01H      ;T0 为定时器工作方式 1
      MOV    TCON, #00H      ;初始化 T0 相关标志位
      MOV    TL0, #18H       ;初始化 T0 定时初值低 8 位
      MOV    TH0, #0FCH      ;初始化 T0 定时初值高 8 位
      CLR    A               ;DAC 转换数据初值 0 送累加器 A
      MOV    30H, A          ;30H 作为输出锯齿波数据的计数单元
      SETB   ET0             ;T0 中断允许
      SETB   EA              ;开总中断
      SETB   TR0             ;启动 T0 定时
      SJMP   $               ;无限循环, 等待中断
TIMER0: PUSH   ACC           ;累加器 A 的内容压栈
      PUSH   82H             ;DPL 的地址值为 82H
      PUSH   83H             ;DPH 的地址值为 83H
      MOV    DPTR, #OFFFEH   ;数据指针 DPTR 指向 DAC0832
      MOV    A, 30H          ;待转换数据→A
      MOVX   @DPTR, A        ;DAC0832 转换数据, 并输出
      INC    30H             ;待转换数据加 1
      CJNE   A, #64H, LOOP   ;(A) 与三角波最大值 100 比较, 相等顺序执行
      CLR    A               ;已输出一个周期的锯齿波, 清转换数据初值
      MOV    30H, A          ;清计数单元的初始数据
LOOP:  MOV    TL0, #18H       ;重载 T0 定时初值低 8 位
      MOV    TH0, #0FCH      ;重载 T0 定时初值高 8 位
      POP    83H
      POP    82H
      POP    ACC
      RETI

```

2) 三角波。按照图 10-5，三角波的周期 $T_a = 200\text{ ms}$ ，最大幅值 $A = 64\text{H} = 100$ ，每个待转换的数字量维持 $T = 1\text{ ms}$ 时间，DAC0832 的输入数据每间隔 1 ms 加 1 一次，应用 T0 工作于方式 1 定时 1 ms ，采用定时中断方法编程。

根据图 10-5，比较三角波和锯齿波的波形，不同的地方在于 1 ms 的中断服务程序的编制，一是 30H 单元内容从 0 每 1 ms 加 1 直到 $64\text{H}(100)$ ，然后再从 64H 每 1 ms 减 1 直到 0，

(2) 双缓冲工作方式

```

graph TD
    Start([中断服务程序]) --> Protect[保护现场]
    Protect --> Output[输出波形参数(30H) → A  
(A) → DAC0832转换]
    Output --> Decision1{中断次数 > 200?}
    Decision1 -- Y --> Clear[中断次数清0]
    Clear --> Decision1
    Decision1 -- N --> Decision2{中断次数 > 100?}
    Decision2 -- Y --> Dec30H["(30H) - 1 → 30H"]
    Decision2 -- N --> Inc30H["(30H) + 1 → 30H"]
    Dec30H --> IncCount[中断次数 + 1]
    Inc30H --> IncCount
    IncCount --> Reload[重装载定时初值]
    Reload --> Restore[恢复现场]
    Restore --> End([中断返回])
  
```

图 10-6 中断服务程序流程图

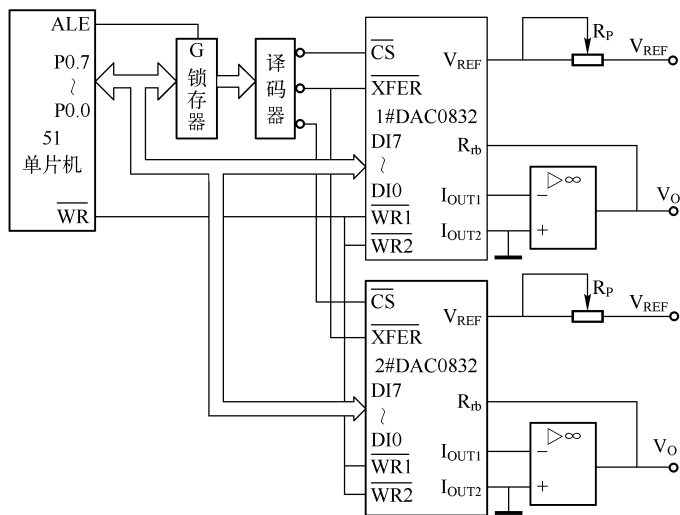


图 10-7 两块 DAC0832 与 51 单片机接口的双缓冲电路连接方式

(3) 直通工作方式

根据图 10-2，如果将 DAC0832 的 $\overline{\text{ILE}}$ 接 +5 V， $\overline{\text{CS}}$ 、 $\overline{\text{WR1}}$ 、 $\overline{\text{WR2}}$ 和 $\overline{\text{XFER}}$ 直接与系统的数字地连接， $\text{DI0} \sim \text{DI7}$ 直接与 51 单片机的 P1 口连接，则 8 位 DAC 寄存器和 8 位输入寄存器处于输入数据与输出数据直通状态，51 单片机 P1 口输出的数据直接传输到 8 位 D-A 转换电路进行 D-A 转换，输出模拟信号。

10.1.3 51 单片机与 DAC1208 的接口与编程

DAC1208 数模转换器是美国国家半导体公司生产的一种电流输出型高速并行 D-A 转换器，分辨率为 12 位，转换时间为 $1\ \mu\text{s}$ ，功耗为 20 mW。由于其片内有输入数据寄存器，能够与 51 单片机接口连接。

1. DAC1208 内部结构

DAC1208 内部结构图如图 10-8 所示，由一个 8 位输入锁存器、一个 4 位输入锁存器、一个 12 位 DAC 寄存器和一个 12 位 D-A 转换器构成，与 DAC0832 结构相似，都能够连接成两级输入锁存的双缓冲方式或一级输入锁存的单缓冲方式，或完全直通的无缓冲方式。DAC1208 的 8 位和 4 位输入锁存器由 $\overline{\text{CS}}$ 和 $\overline{\text{WR1}}$ 控制， $\overline{\text{XFER}}$ 和 $\overline{\text{WR2}}$ 控制 DAC1208 的 12 位 DAC 寄存器。与 DAC0832 结构的区别在于，DAC1208 的 DAC 寄存器和 D-A 转换器为 12 位。而 $\overline{\text{BYTE1}}/\overline{\text{BYTE2}}$ 用来区分是 8 位输入锁存器还是 4 位输入锁存器。在向 DAC1208 输入 12 位数字量时，应先输入高 8 位，然后再输入低 4 位，最后使 $\overline{\text{XFER}}$ 和 $\overline{\text{WR2}}$ 为低电平使 $\overline{\text{LE3}}$ 有效，12 位数据同时进入 12 位 DAC 寄存器，启动 D-A 转换。

2. DAC1208 的引脚功能

DAC1208 的标准双列直插式芯片为 24 引脚，如图 10-9 所示。各引脚功能如下：

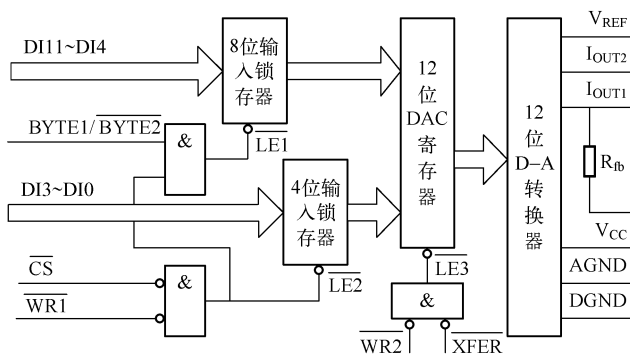


图 10-8 DAC1208 内部结构图

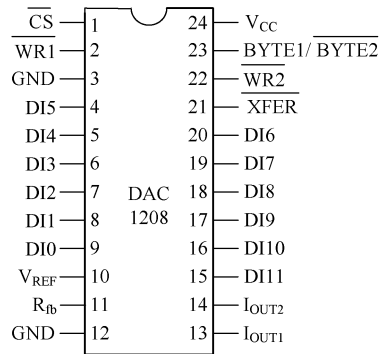


图 10-9 DAC1208 引脚分布图

- 1) $\text{DI11} \sim \text{DI0}$ ：12 位数据输入引脚。
- 2) $\overline{\text{CS}}$ ：片选端，低电平有效，8 位和 4 位输入锁存器被选中。
- 3) $\overline{\text{BYTE1}}/\overline{\text{BYTE2}}$ ：字节顺序控制信号。当 $\overline{\text{BYTE1}}/\overline{\text{BYTE2}} = 1$ ，则 8 位和 4 位两个输入锁存器都被选中；当 $\overline{\text{BYTE1}}/\overline{\text{BYTE2}} = 0$ 时，则只选中 4 位输入锁存器。
- 4) $\overline{\text{WR1}}$ ：写信号，低电平有效，当信号变高电平时，8 位和 4 位输入锁存器都进入锁

存状态。

5) $\overline{\text{WR2}}$: 写信号, 低电平有效。

6) $\overline{\text{XFER}}$: 数据传送控制信号, 低电平有效, 与 $\overline{\text{WR2}}$ 配合。当 $\overline{\text{XFER}} = 0$, $\overline{\text{WR2}} = 0$ 时, 把 8 位和 4 位两个输入锁存器组成的 12 位待转换的数据传入 12 位 D - A 寄存器中。

7) I_{OUT1} : D - A 转换器电流输出 1 端, 输入数字量全为 “1” 时, I_{OUT1} 最大, 输入数字量全为 “0” 时, I_{OUT1} 最小。

8) I_{OUT2} : D - A 转换器电流输出 2 端, $I_{\text{OUT2}} + I_{\text{OUT1}} = \left(1 - \frac{1}{4096}\right) \times \frac{V_{\text{REF}}}{R}$, $R = 15\text{k}\Omega$ 。

9) R_{fb} : 外部反馈信号输入端, 为外部运算放大器提供一个反馈电压。根据需要还可以外接另外一个反馈电阻。

10) V_{CC} : 电源输入端, 在 $+5 \sim +15\text{V}$ 范围内。

11) V_{REF} : 参考电压输入, 在 $-10 \sim +10\text{V}$ 范围内。

12) GND: 信号地。

3. DAC1208 与 51 单片机的接口电路及转换方法

图 10-10 为 DAC1208 与 51 单片机的接口电路图。假设单片机地址总线与 DAC1208 地址的无关位取 0, 则 DAC1208 的高 8 位输入锁存器地址为 6000H, 低 4 位输入锁存器地址为 4000H, 12 位 DAC 寄存器地址为 8000H, 采用双缓冲方式编程, 先送高 8 位数据 DI11 ~ DI4, 再送入低 4 位数据 DI3 ~ DI0, 在 12 位数据分别写入两个输入锁存器后, 再打开 12 位 DAC 寄存器, 进行 D - A 转换。

例 10-2 根据图 10-10, 假设待转换的 12 位数字量高 8 位存放在 30H, 低 4 位数据存放在 31H, 试编写相应的 D - A 转换子程序。

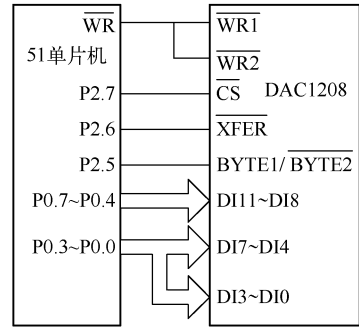


图 10-10 DAC1208 与单片机的接口电路

```
DA: MOV    DPTR, #6000H    ;高 8 位输入锁存器地址
      MOV    R1, #30H      ;高 8 位数据的地址→R1
      MOV    A, @R1        ;取出高 8 位数据
      MOVX   @DPTR, A      ;高 8 位数据写入 DAC1208 的高 8 位和低 4 位输入锁存器
      MOV    DPTR, #4000H  ;低 4 位输入锁存器地址
      MOV    R1, #31H      ;低 4 位数据的地址→R1
      MOV    A, @R1        ;取出低 4 位数据
      MOVX   @DPTR, A      ;低 4 位数据写入 DAC1208 的低 4 位输入锁存器
      MOV    DPTR, #8000H  ;12 位 DAC 寄存器的地址→DPTA
      MOVX   @DPTR, A      ;12 位数据同步进行 D - A 转换, 并输出
      RET
```

10.2 A - D 转换器接口

在微处理器应用系统中, 常常需要对物理量的模拟信号如电压、电流等进行采集和处

理，需要把模拟信号转换成数字信号，微处理器才能够进行数据处理，A-D转换器（Analog to Digital Converter，简称ADC）能够实现这一功能。由于实现这种转换的工作原理和采用工艺技术不同，因此生产出种类繁多的A-D转换芯片。A-D转换器按分辨率分为4位、8位、10位、12位、14位、16位和输出BCD码的 $3\frac{1}{2}$ 位、 $5\frac{1}{2}$ 位等；按照转换速度大致可分为超高速（转换时间 $\leq 1\text{ ns}$ ）、次超高速（转换时间 $\leq 1\text{ }\mu\text{s}$ ）、高速（转换时间 $\leq 330\text{ }\mu\text{s}$ ）、中速（ $\leq 1\text{ ms}$ ）和低速（转换时间 $\leq 1\text{ s}$ ）等；按照转换原理可分为逐次比较型A-D转换器、双积分型A-D转换器、并行比较型/串并行型和 $\Sigma-\Delta$ 调制型等，目前最常用的是逐次比较型A-D转换器和双积分型A-D转换器。

10.2.1 A-D转换器原理

1. 逐次比较型A-D转换器工作原理

以8位逐次比较型A-D转换器为例，其内部结构如图10-11所示。它由8位寄存器、8位D-A转换器、电压比较器、控制逻辑和三态锁存缓冲器等部分组成。8位寄存器用来存放8位二进制数码，当模拟量 V_{IN} 送入电压比较器后，启动信号有效，通过控制逻辑电路启动A-D转换，在时钟脉冲信号CLK的作用下进行A-D转换。首先，将8位寄存器清零，然后置其最高位（D7）为1，8位寄存器输出为10000000B，8位寄存器的内容经D-A转换器转换为相应的模拟电压 V_N ，并与模拟输入电压 V_{IN} 进行比较，如果 $V_{\text{IN}} \geq V_N$ ，则8位寄存器中最高位的1保留，否则就将最高位清零。然后再使次高位（D6）置1，进行相同的过程直到8位寄存器中所有位都被确定，控制逻辑输出 $\text{EOC}=1$ ，转换过程结束，8位寄存器中的二进制数码就是A-D转换器的结果，在OE信号的作用下，A-D转换的结果通过三态锁存缓冲器输出到微处理器的数据总线。

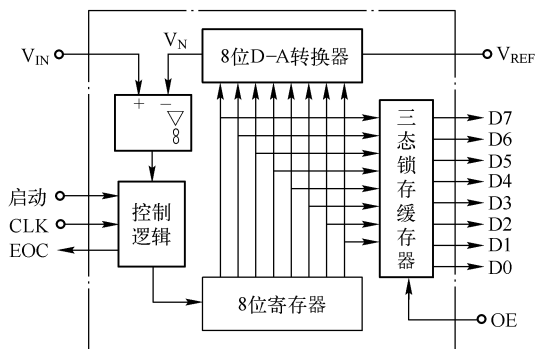


图 10-11 逐次比较型 A-D 转换器的结构框图

逐次比较型A-D转换器是一种转换速度较高（转换时间在几微秒到几百微秒之间）、精度较高、功耗低的转换器，在低分辨率（ ≤ 12 位）时价格便宜，高精度（ > 12 位）时价格很高。

2. 双积分型A-D转换器的转换原理

双积分型A-D转换器的电路原理如图10-12a所示，由电子开关S、标准电压源 V_{REF} 、积分器、比较器、计数器和控制逻辑等部件组成。双积分型A-D转换的原理就是进行一次

A-D 转换需要二次积分。转换时，控制逻辑通过电子开关 S 把被测电压 V_{IN} 加到积分器的输入端，积分器从零开始，在固定的时间 T 内对 V_{IN} 积分，这个积分过程被称为定时积分，积分输出终值与 V_{IN} 成正比。接着控制逻辑将电子开关切换到极性与 V_{IN} 相反的基准电压 V_{REF} 上进行反向积分，由于基准电压 V_{REF} 恒定，因此积分输出将按 T 期间积分的值以恒定斜率下降，当比较器检测积分输出为零时，积分器停止工作，这个积分过程被称为定值积分。对标准电压 V_{REF} 的反相积分时间 T_1 与定值积分的初值成比例关系，即 T_1 正比于输入模拟电压 V_{IN} 。图 10-12b 给出了两种不同输入电压 ($V_{IN} > V'_{IN}$) 的积分情况，可知 V'_{IN} 小，在 T 定时积分期间积分器输出的终值也小，而下降斜率相同，因此反相积分时间 T_2 也小。利用计数器对时钟脉冲的个数进行计数来测量反相积分时间 T_1 (或 T_2)，即可得到对应于模拟电压 V_{IN} 的数字量。

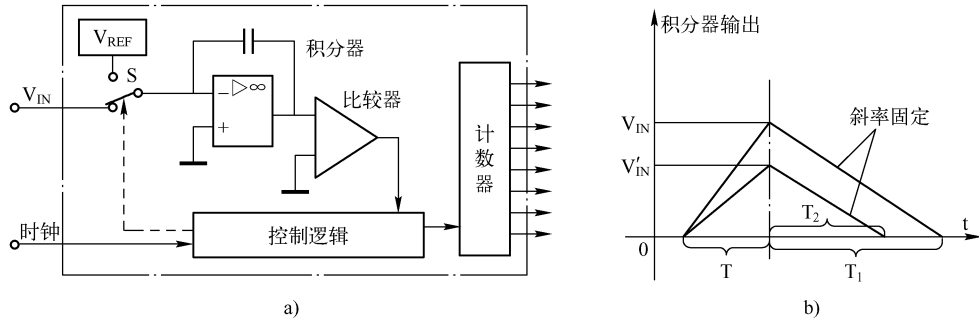


图 10-12 双积分型 A-D 转换器的电路原理框图

a) 电路工作原理图 b) 双积分过程

双积分型 A-D 转换器的优点是用简单电路就能获得高分辨率，抗干扰能力强，转换精度高，但缺点是由于转换精度依赖于积分时间，转换速率低。

3. A-D 转换器的主要技术指标

(1) 分辨率

ADC 的分辨率是指使输出数字量变化一个相邻数码所需输入模拟电压的变化量，分辨率是衡量 A-D 转换器能够分辨出输入模拟量最小变化程度的技术指标。分辨率取决于 A-D 转换器的位数，习惯上用输出的二进制位数表示。例如，12 位 A-D 转换器的满量程输入电压为 5 V，可输出 12 位二进制数，即用 2^{12} 个数进行量化，其分辨率为 1LSB，也即 $5\text{ V}/2^{12} = 1.22\text{ mV}$ ，或者说 A-D 转换器能分辨出输入电压 1.22 mV 的变化。

(2) 转换速率和转换时间

ADC 的转换时间则是完成一次 A-D 转换所需的时间 (包括稳定时间)，转换速率是转换时间的倒数，即每秒转换的次数。

(3) 转换精度

ADC 的转换精度定义为一个实际的 A-D 转换器与一个理想 A-D 转换器在量化值上的差值，可用绝对误差或相对误差表示。

10.2.2 51 单片机与 ADC0809 的接口

ADC0809 是逐次比较型 A-D 转换器，分辨率为 8 位，采用单电源 +5 V 供电，片内具

有 8 路模拟开关，可对 8 路输入模拟电压实现分时转换，转换时间约为 $100\ \mu\text{s}$ ，片内带有三态输出锁存器，可直接与 51 单片机的数据总线相连接。

1. ADC0809 的内部逻辑结构

ADC0809 的内部逻辑结构如图 10-13 所示，ADC0809 由一个 8 路模拟开关、一个地址锁存与译码器、一个 8 位 A-D 转换器和一个三态输出锁存器组成。8 路模拟开关可以对 8 路模拟输入通道选择 1 路模拟输入信号进行 A-D 转换，由 C、B、A 输入的地址值译码决定所选择的 A-D 转换的模拟输入通道。地址锁存器与译码电路完成对 C、B、A 的 3 个地址位的锁存和译码，三态输出锁存器用于锁存 A-D 转换完的数字量，当 EOC 端为高电平时，表明一次 A-D 转换已完成，51 单片机可以从 ADC0809 的三态输出锁存器读出转换完的数据。

2. ADC0809 的引脚

双列直插式封装的 ADC0809 芯片为 28 引脚，如图 10-14 所示。主要引脚功能如下：

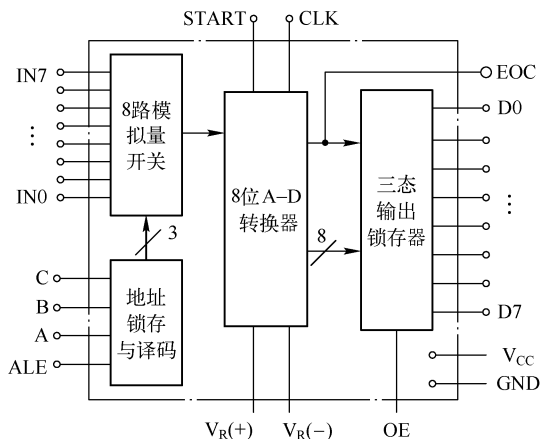


图 10-13 ADC0809 的内部逻辑结构

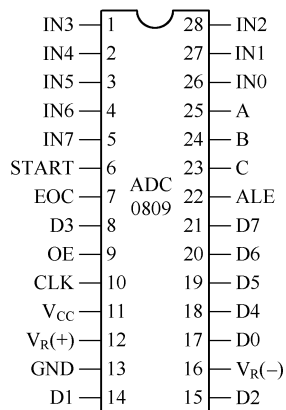


图 10-14 ADC0809 的引脚图

- 1) D7 ~ D0：8 位数字量输出引脚。
- 2) IN7 ~ IN0：8 位模拟量输入引脚。电压范围为 0 ~ 5 V，若信号过小还需进行放大。另外输入的模拟量在 A-D 转换过程中应保持不变，因此对变化速度较快的模拟量，在输入前应增加采样保持电路。
- 3) V_{CC} ：+5 V 工作电压。GND：地。
- 4) $V_R(+)$ 、 $V_R(-)$ ：基准电压正负端，用来与输入的模拟信号进行比较，作为逐次比较型 A-D 转换器的基准电压。
- 5) START：A-D 转换启动信号输入引脚，正脉冲启动 ADC0809，对由 C、B、A 选中的模拟通道输入信号进行转换。
- 6) ALE：地址锁存允许信号输入引脚，上升沿将 C、B、A 地址信号锁存到地址锁存器。
- 7) EOC：转换结束信号输出引脚，启动转换时为低电平，当转换结束时为高电平。
- 8) OE：输出允许控制信号输入引脚，高电平时打开三态输出缓冲器，使转换后的数字量从 D7 ~ D0 引脚输出。
- 9) CLK：时钟信号输入引脚，ADC0809 内部没有时钟电路，所需的时钟信号由外部提供。

时钟频率越高，A-D 转换速度越快。通常使用 500 kHz 频率，此时 A-D 转换时间约为 100 μ s。

10) C、B、A：模拟通道选择的地址输入引脚，其取值为 111 ~ 000，分别对应 IN7 ~ IN0。

3. ADC0809 与 51 单片机的接口设计

ADC0809 与 51 单片机的接口电路如图 10-15 所示，ADC0809 的通道选择线 C、B、A 分别与锁存器 74LS373 输出的低 8 位地址总线的 A2、A1、A0 连接。由于 ADC0809 芯片的引脚中没有一般接口芯片所具有的片选信号 $\overline{CS}$ 端，其芯片最高地址的确定采用了一种特殊的方法予以解决。图 10-15 中，ADC0809 的 ALE 输入引脚和 START 引脚并联再与一个或非门的输出端连接，该或非门的两个输入端分别与 51 单片机的 $\overline{WR}$ 和 P2.7 引脚连接，当 P2.7 为“0”时， $\overline{WR}$ 信号对 ADC0809 才是有效的写信号， $\overline{WR}$ 信号能够将 C、B、A 地址值锁存，并按照该地址译码值选中的模拟通道输入信号启动转换；同理，当 P2.7 为“0”时， $\overline{RD}$ 信号对 ADC0809 才是有效的读信号。ADC0809 的 CLK 引脚经由 74LS74 触发器构成的二分频电路与单片机的 ALE 引脚连接。假设单片机的振荡频率为 6 MHz，单片机引脚 ALE 输出 1 MHz 信号，则 ADC0809 的 CLK 引脚的输入频率为 500 kHz。显然，图 10-15 的这种接口电路连接方法中，两个或非门的各自一个输入端并联连接作为 ADC0809 的最高地址位，所以 P2.7 是 ADC0809 芯片的最高地址位，这样 8 个模拟输入通道 IN7 ~ IN0 的地址分别为 7FFFH ~ 7FF8H（地址无关位置 1）。51 单片机读取 ADC0809 转换后的数据常用以下两种方式：

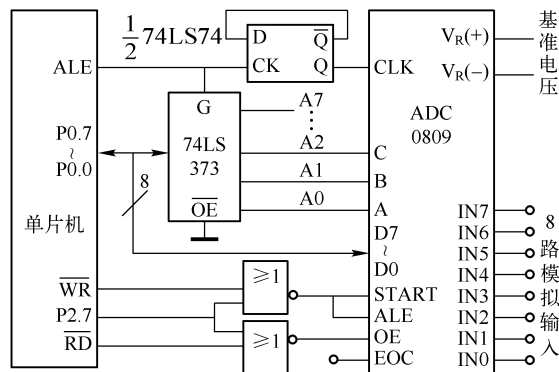


图 10-15 ADC0809 与 51 单片机的接口电路

(1) 程序查询方式

将 ADC0809 的 EOC 端与单片机的某个 I/O 端口连接，首先由单片机发出 A-D 转换器的启动信号，然后单片机通过程序不断读取 EOC 端的状态，判断 EOC 状态，如果为“1”，表示转换结束，从 A-D 转换器读取转换后的数据，如果 EOC 为“0”，表示转换未完成，继续查询。

(2) 中断方式

把转换完成的状态信号 EOC 作为中断请求信号，采取中断方式进行数据传送。如在图 10-15 中，ADC0809 的 EOC 输出端增加一个非门与单片机的 $\overline{INT0}$ 引脚连接，当 A-D 转换结束后，EOC 状态由 0→1 时，向单片机发出中断请求信号，单片机响应中断，在中断服务程序中将 A-D 转换的结果读出传输给累加器 A，然后再启动下一次 A-D 转换。

例 10-3 在图 10-15 中，将 ADC0809 的 EOC 端经过一个非门和单片机的 $\overline{INT0}$ 引脚连

接。采用中断方式对 ADC0809 的 IN0 路模拟信号进行采样一次，并把转换结果存储在 A 中。

```
ORG    0000H
AJMP   MAIN
ORG    0003H
AJMP   LINT0           ;EOC 由 0→1,发出中断请求,单片机响应中断
ORG    0030H
MAIN:MOV    DPTR,#7FF8H      ;主程序,将 IN0 路的地址值→DPTR
      SETB   IT0             ;置IT0为边沿触发
      SETB   EA
      SETB   EX0            ;允许INT0中断
      MOVX   @ DPTR,A        ;执行该指令,P2.7=0,产生WR信号,ALE 和 START
                                ;信号有效,启动 IN0 路的 A-D 转换;(A)可以是任意值
      SJMP   $               ;等待中断
LINT0:MOV    DPTR,#7FF8H      ;将 IN0 路的地址值→DPTR
      MOVX   A,@ DPTR        ;中断服务程序:读入 IN0 路的转换结果存储在 A 中
      MOVX   @ DPTR,A        ;再次启动对 IN0 路的 A-D 转换
      RETI
```

例 10-4 在数据处理和测控系统中，经常需要对多路模拟信号进行定时采样，在图 10-15 中，将 ADC0809 的 EOC 输出端增加一个非门与单片机 P1.0 引脚连接，采用定时 100 ms 的中断方式对 ADC0809 的 8 路模拟信号进行巡回采样一次，而每一路模拟信号的采样应用查询方式，并把每次 8 路转换结果按照顺序存储在 51 单片机内部 RAM 的 30H~37H 单元中。设 51 单片机时钟振荡频率为 6 MHz。

6 MHz 的周期 $T = 2 \mu s$ ，定时时间 $t = 100 ms$ ，定时初值 $X = 65536 - \frac{t}{T} = 65536 - 100 \times 0.5 \times 10^3 = 15536 = 3CB0H$ ，选用 T0 定时器。

分析：根据本例要求，T0 定时中断服务程序流程图如图 10-16 所示。主程序的初始化程序中主要完成 T0 的初始化、开 T0 的中断和启动 T0 定时。按照图 10-16 的 T0 定时中断服务程序流程图编写 T0 定时中断服务程序如下：

```
TIME0: PUSH    ACC
      MOV      TL0,#0B0H
      MOV      TH0,#3CH      ;重置 T0 定时初值
      MOV      DPTR,#7FF8H   ;选中 IN0 通道
      MOV      R1,#30H        ;置数据存储区首地址
      MOV      R7,#08H        ;置通道数
```

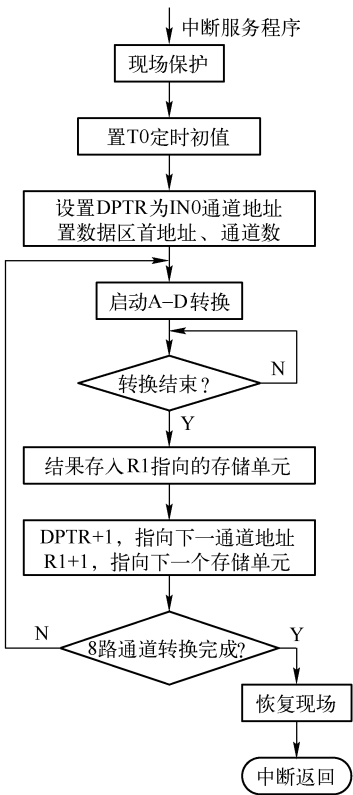


图 10-16 T0 定时中断服务程序流程图


```

LOOP: MOVX    @ DPTR,A      ;启动 A - D 转换
TEST: JB      P1.0,TEST     ;判断转换是否已完成
      MOVX    A,@ DPTR     ;转换结束,读入转换结果
      MOV     @ R1,A        ;存入内部 RAM 存储区
      INC     DPTR          ;指向下一个通道
      INC     R1            ;修改地址指针,指向下一个采集通道存放数据的地址
      DJNZ    R7,LOOP       ;8 个通道是否转换完成
      POP     ACC
      RETI

```

10.2.3 51 单片机与 AD574 的接口

AD574 是 Analog Devices 公司生产的 12 位逐次比较型快速 A - D 转换器, 转换时间 $\leq 35 \mu\text{s}$, 转换精度 $\leq 0.05\%$ 。可进行 12 位或 8 位转换, 可以一次输出转换好的 12 位数据, 也可分两次输出, 先高 8 位输出, 后低 4 位输出。内部有三态输出缓冲电路, 输出电平与 TTL 和 CMOS 电平兼容, 无须外加时钟电路, 适合在高精度快速数据采集系统中使用。

1. AD574 的引脚功能

AD574 为 28 脚双列直插式封装, 引脚如图 10-17 所示, 各引脚功能如下:

- 1) $\overline{\text{CS}}$: 片选信号, 低电平有效。
- 2) CE: 片允许信号, 高电平有效。
- 3) $\text{R}/\overline{\text{C}}$: 读出/转换控制信号。当 $\text{CE} = 1$ 、 $\overline{\text{CS}} = 0$ 、 $\text{R}/\overline{\text{C}} = 0$ 时启动 A - D 转换; 当 $\text{CE} = 1$ 、 $\overline{\text{CS}} = 0$ 、 $\text{R}/\overline{\text{C}} = 1$ 时, 可以读出 A - D 转换结果。
- 4) $12/\overline{8}$: 数据输出格式选择端, 当 $12/\overline{8} = 1$ 时, 12 位数据同时输出; 当 $12/\overline{8} = 0$ 时, A - D 转换结果分两次输出, 先高 8 位输出, 后低 4 位输出。该引脚信号与 A0 引脚配合使用, $12/\overline{8}$ 引脚只能接 V_{CC} 或 GND。
- 5) A0: 字节选择控制线。A0 = 1, 8 位 A - D 转换, 完成一次 A - D 转换的时间是 $10 \mu\text{s}$; A0 = 0, 12 位 A - D 转换, 完成一次 A - D 转换的时间是 $35 \mu\text{s}$ 。

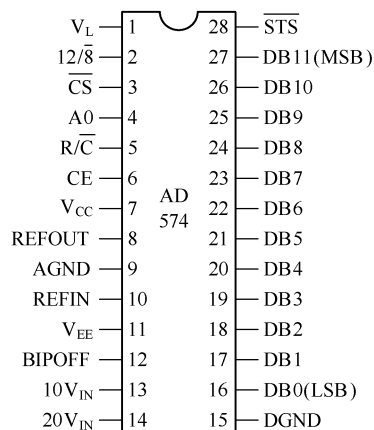


图 10-17 AD574 封装引脚图

当 $12/\overline{8} = 0$ 时, 在读出 A - D 转换结果期间, A0 = 0 读出高 8 位数据, A0 = 1 读出低 4 位数据。两次读出的 12 位数据遵循左对齐原则, 12 位数据排列见表 10-1。

表 10-1 AD574 的 A - D 转换结果的 12 位输出数据的排列

A - D 转换结果的高 8 位								A - D 转换结果低 4 位 + 尾随 4 位补零							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	0	0	0	0

AD574 控制信号 $\overline{\text{CS}}$ 、CE、 $\text{R}/\overline{\text{C}}$ 、 $12/\overline{8}$ 和 A0 组合功能见表 10-2。

表 10-2 AD574 控制功能表

CE	$\overline{\text{CS}}$	$\text{R}/\overline{\text{C}}$	$12/\overline{8}$	A0	操 作
0	×	×	×	×	禁止
×	1	×	×	×	禁止
1	0	0	×	0	启动 12 位 A-D 转换
1	0	0	×	1	启动 8 位 A-D 转换
1	0	1	V_{CC}	×	允许 12 位 A-D 转换结果并行输出
1	0	1	GND	0	允许 A-D 转换结果高 8 位输出
1	0	1	GND	1	允许 A-D 转换结果低 4 位输出

6) 电源线。

V_{L} : 数字逻辑电源 +5 V; V_{CC} : +12 V ~ +15 V 电源线; V_{EE} : -15 V ~ -12 V 电源线;

REFOUT: 参考电压输出, AD574 向外提供 +10 V 基准电压输出。

REFIN: 参考电压输入, 一般通过一个 50 Ω 电阻与 REFOUT 引脚连接, 或用于满量程调节。

AGND 和 DGND: 模拟信号地和数字信号地。

7) BIPOFF: 双极性偏差调整, 用于调零。

8) $10 V_{\text{IN}}$ 和 $20 V_{\text{IN}}$: 模拟信号输入引脚。可以工作在单极性输入或双极性输入方式。单极性输入时, 允许输入的信号范围为 0 ~ +10 V 或 0 ~ +20 V; 双极性输入时, 允许输入的信号范围为 -5 ~ +5 V 或者 -10 ~ +10 V。

9) $\overline{\text{STS}}$: 转换结束信号, 转换时为高电平, 当转换结束时为低电平。

2. 51 单片机与 AD574 的接口设计

AD574 工作在双极性输入方式时与 51 单片机的接口电路如图 10-18 所示, 可实现对 -5 V ~ +5 V 或 -10 V ~ +10 V 模拟信号进行转换。图 10-18 中电位器 R_2 用于调整双极性输入电路的零点, $10 V_{\text{IN}}$ 引脚输入 -5 V ~ +5 V 模拟信号, $20 V_{\text{IN}}$ 引脚输入 -10 V ~ +10 V 模拟信号。双极性输入时, 输出的转换结果 D 与模拟输入电压 V_{IN} 之间的关系为

$$D = 2048 \times \left(1 + \frac{2V_{\text{IN}}}{V_{\text{FS}}}\right) \quad \text{或} \quad V_{\text{IN}} = \left(\frac{D}{2048} - 1\right) \times \frac{V_{\text{FS}}}{2}$$

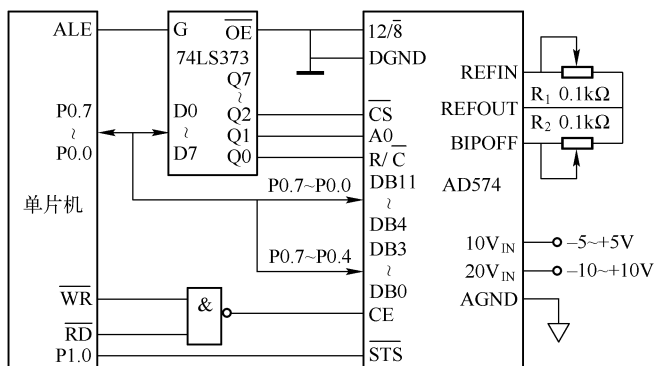


图 10-18 51 单片机与 AD574 的接口电路

式中, V_{FS} 为满量程电压。上式求出的 D 为 12 位偏移二进制码, 把 D 的最高位求反便得到补码, 补码对应输入模拟量的符号和大小。例如, 当模拟信号从 $10 V_{IN}$ 引脚输入时, $V_{FS} = 10 V$, 若读得 $D = FFFH$, 即 $1111\ 1111\ 1111B = 4095$, 代入上式, 可求得 $V_{IN} = 4.9976 V$ 。

$A-D$ 转换结果的高 8 位从 $DB11 \sim DB4$ 输出, 低 4 位从 $DB3 \sim DB0$ 输出, 即 $A0 = 0$ 时, 读取结果的高 8 位; 当 $A0 = 1$ 时, 读取结果的低 4 位。遵循左对齐的原则, $DB3 \sim DB0$ 应接单片机的 $P0.7 \sim P0.4$ 。

$\overline{STS}$ 引脚接单片机的 $P1.0$ 引脚, 采用查询方式读取转换结果。当单片机执行对外部数据存储器写指令, 使 $\overline{CS} = 0$ 、 $CE = 1$ 、 $R/\overline{C} = 0$ 、 $A0 = 0$ 时, 启动 $A-D$ 转换。当单片机查询到 $P1.0$ 引脚为低电平时, 表示 $A-D$ 转换结束, 单片机使 $CE = 1$ 、 $\overline{CS} = 0$ 、 $R/\overline{C} = 1$ 、 $A0 = 0$, 读取 $A-D$ 转换结果高 8 位; 使 $CE = 1$ 、 $\overline{CS} = 0$ 、 $R/\overline{C} = 1$ 、 $A0 = 1$, 读取 $A-D$ 转换结果的低 4 位。

例 10-5 根据图 10-18 接口电路, 采用查询方式编写 $A-D$ 转换程序, 并把转换结果的 12 位数字量高 8 位存入单片机内部 RAM 的 $30H$ 单元, 低 4 位存入 $31H$ 单元, 遵循左对齐原则。

图 10-18 中, $P0.2$ 、 $P0.1$ 和 $P0.0$ 通过地址锁存器与 $AD574$ 的 $\overline{CS}$ 、 $A0$ 和 $R/\overline{C}$ 连接, 因此, 51 单片机对 $AD574$ 的 12 位数据采集的操作处理过程如下:

1) 启动转换。 $\overline{CS} = 0$, $A0 = 0$ 且 $R/\overline{C} = 0$, 则按 12 位转换, 地址线无关位置 1, 则 12 位 $A-D$ 转换的地址为 $1111\ 1111\ 1111\ 1000B = FFF8H$ 。

2) 读取转换结果。 $\overline{CS} = 0$, $A0 = 0$, $R/\overline{C} = 1$ 时读高 8 位, $A0 = 1$ 时读低 4 位, 因而, 读高 8 位数据的地址为 $FFF9H$, 读低 4 位数据的地址为 $FFFBH$ 。

3) $P1.0$ 用于单片机查询 $A-D$ 转换是否结束。

AD574: MOV	DPTR, #0FFF8H	; 12 位 $A-D$ 转换的地址 $\rightarrow$ DPTR
	MOVX	@ DPTR, A ; 启动 AD574 进行转换
	SETB	P1.0 ; 置 $P1.0$ 为 1, 为读取 $P1.0$ 输入状态做准备
LOOP: JB	P1.0, LOOP	; 查询转换是否结束
	INC	DPTR ; 读高 8 位数据的地址 $\rightarrow$ DPTR, 准备读取结果
	MOVX	A, @ DPTR ; $R/\overline{C} = 1$, 读取高 8 位转换结果
	MOV	30H, A ; 高 8 位转换结果存入 $30H$ 单元
	INC	DPTR
	INC	DPTR ; 使 $R/\overline{C} = 1$, $A0 = 1$, 读低 4 位数据的地址 $\rightarrow$ DPTR
	MOVX	A, @ DPTR ; $R/\overline{C} = 1$, $A0 = 1$, 读取低 4 位转换结果
	ANL	A, #0F0H
	MOV	31H, A ; 低 4 位转换结果存入 $31H$ 单元
	RET	

上述程序是按查询方式设计的, 图 10-18 中的 $\overline{STS}$ 引脚也可接单片机的外部中断输入引脚, 即采用中断方式读取转换结果, 读者可自行编制相应程序。

习题

10-1 什么是 A-D 转换和 D-A 转换?为何要进行转换?各在什么应用情况下使用?

10-2 简述 D-A 转换器的主要性能指标,设某二进制 12 位的 DAC 满量程输出电压为 5 V,试问它的分辨率是多少?

10-3 说明 DAC0832 的单缓冲、双缓冲和直通工作方式的接口电路、工作过程与特点。

10-4 上网查询,说明目前应用较广泛的 A-D 转换器主要有哪几种类型。

10-5 ADC0809 器件有哪些主要技术指标?其含义是什么?

10-6 等待 A-D 转换结束有哪几种方式?各有何特点?

10-7 DAC 和 ADC 的主要技术指标中,量化误差、分辨率和精度有何区别?

10-8 按照图 10-4,设 51 单片机晶振为 6 MHz,要求 DAC0832 输出周期为 10 ms 的方波,方波上限电平数据为 data1,下限电平为 data2,请应用定时器中断 5 ms 的方法编制程序。

10-9 在图 10-7 中,如果译码器是 74LS138,其 C、B、A 输入端分别与 51 单片机的 P2.6、P2.5、P2.4 连接,74LS138 的 $\overline{Y0}$ 与 1# DAC0832 的 $\overline{CS}$ 连接, $\overline{Y1}$ 与 1# 和 2# DAC0832 的 $\overline{XFER}$ 连接, $\overline{Y2}$ 与 2# DAC0832 的 $\overline{CS}$ 连接,写出 1# 和 2# DAC0832 的 8 位输入寄存器的地址、1# 和 2# DAC0832 的 8 位 DAC 寄存器的地址,并绘制电路图。

10-10 应用 ADC0809 设计一个与 51 单片机接口的采用译码法编址的电路,每隔 5 s 对 8 个模拟通道各采样一次,将采样结果保存在内部 RAM 的 30H 开始单元。画出原理图并编制采样程序。

第 11 章 51 单片机与键盘、显示器接口设计

在微处理器应用系统中，人机界面设计必不可少。人机界面的输入器件常用的有按键、开关和触摸屏；输出显示器件的种类比较多，比如发光二极管、数码显示（LED、LCD）、彩色液晶显示器等。目前，通常将触摸屏贴在彩色液晶显示屏上作为人机界面，但触摸屏 + 彩色液晶显示器作为人机界面的编程对于初学者而言较为复杂，本章将主要介绍键盘和数码显示的接口电路和编程方法。

11.1 键盘接口电路

键盘是多个按键的集合，按其结构形式可分为非编码键盘和编码键盘，

编码键盘采用硬件方法产生按键代码（简称键码）。按下一个或同时按下多个按键，键盘能自动生成按键代码，按键数较多，且具有去抖动功能。这种键盘使用方便，但硬件较复杂，个人计算机所用键盘即为编码键盘。

非编码键盘与单片机的连接方式可分为独立式键盘与矩阵式键盘。非编码键盘仅提供按键开关工作状态，其键码由软件确定，这种键盘按键数较少，电路简单，广泛应用于微处理器应用系统中，本节主要介绍非编码键盘的设计与应用。

11.1.1 独立式键盘

独立式键盘的每个按键占用一根 I/O 口线，每根 I/O 口线上的按键工作状态不会影响其他按键的工作状态。这种按键处理程序简单，但占用 I/O 口线较多，适用于需要按键数量较少的微处理器应用系统中。

图 11-1 为 4 个独立式按键与 51 单片机的连接电路。对按键的处理需要编程判断是否有键按下，若有键按下，判断是哪一个键按下，并求出其键码，根据键码执行处理该按键的程序。每个按键的键码是由编程者自己设定的。图 11-1 中，无键按下时，其四根线都是高电平，与门输出也是高电平；有键按下时，与门输出为低电平。

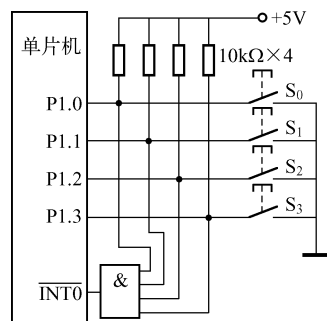


图 11-1 独立式键盘应用电路

1. 按键的抖动与处理

由于按键类属于开关，存在按键闭合和断开时短暂的机械弹性作用，按键闭合时不会立即稳定地被接通，按键断开时也不会立即可靠地被断开，因此在按键闭合与断开的瞬间会出现电压抖动现象，如图 11-2 所示，按键闭合和断开时的抖动时间一般为 5 ~ 10 ms，抖动现象会引起微处理器对一次按键操作多次进行处理的现象，从而产生错误。当微处理器测试到有按键闭合或断开时，需要进行消除抖动处

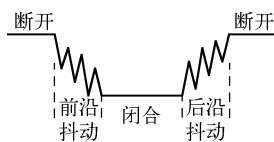


图 11-2 按键抖动示意图

理，才能够得到按键闭合与断开的稳定状态。

消除按键抖动有硬件与软件两种方法：硬件方法是加消抖电路，例如，可通过 RS 触发器或阻容滤波实现用硬件消除抖动；软件方法是在第一次检测到键盘有按键按下后，延时 10 ~ 20 ms 再次检测判断键盘是否确实有按键闭合，如果有按键闭合，再进行键盘扫描确定闭合的按键。判断按键是否断开的过程与判断按键闭合的过程类似。消除按键抖动的程序通常采用定时中断或软件延时的方法来消除按键抖动的影响。

2. 独立式键盘的编程

(1) 是否有键按下的处理子程序

例 11-1 以图 11-1 的 4 个独立式键盘与 51 单片机的连接电路为例。图 11-1 中，若有键按下，则 P1.0 ~ P1.3 不会全为高电平。是否有键按下的处理子程序主要是对读入的按键状态进行预处理，若有键闭合，则(A) ≠ 0，若无键闭合，则(A) = 0。

```
KCS: MOV    P1, #0FFH    ;读 P1 口之前,先设置 P1 口的所有端口为 1 状态
      MOV    A, P1
      CPL    A            ;对 A 的每一位取反,无键按下,低四位全部为 0
      ANL    A, #0FH      ;确保 A 中的高四位为 0,保留低四位的值
      MOV    30H, A        ;读入的按键结果保存在 30H 中
      RET
```

(2) 键识别与键处理子程序

例 11-2 键识别程序就是判断是哪一个键按下的程序，图 11-1 中，若某键闭合则对应单片机 P1.0 ~ P1.3 某一个引脚输入为低电平，但子程序 KCS 中已将按键的输入状态取反，则某键闭合对应的状态为“1”，采取逐个按键判断确定被按下的按键，然后再调用该按键的处理子程序。

键识别与键处理子程序如下：

```
KEY: MOV    A, 30H        ;按照子程序 KCS,按键结果存放在 30H 中
KEY0: JNB    ACC. 0, KEY1  ;P1.0 = 1 是 0 号键,顺序执行,否则转移
      ACALL  FUN0          ;调用处理 0 号键功能的子程序
      SJMP   RETURN        ;键识别子程序返回
KEY1: JNB    ACC. 1, KEY2  ;P1.1 = 1 是 1 号键,顺序执行,否则转移
      ACALL  FUN1          ;调用处理 1 号键功能的子程序
      SJMP   RETURN        ;键识别子程序返回
KEY2: JNB    ACC. 2, KEY3  ;P1.2 = 1 是 2 号键,顺序执行,否则转移
      ACALL  FUN2          ;调用处理 2 号键功能的子程序
      SJMP   RETURN        ;键识别子程序返回
KEY3: JNB    ACC. 3, RETURN ;P1.3 = 1 是 3 号键,顺序执行,否则转移
      ACALL  FUN3          ;调用处理 3 号键功能的子程序
RETURN: RET                ;键识别子程序返回
```

(3) 消抖动与独立式键盘程序

对于键盘消抖动的滤波时间通常设置为 10 ~ 20 ms，编制软件延时的方法对测控和信息处理系统的处理速度产生极大的影响，在实际的微处理器应用系统设计中通常采用定时中断

的方式来实现消抖动滤波的延时；另外，通常设计的键盘处理程序对于按键闭合一次只进行一次键功能处理操作，即需要确保按键闭合时只调用一次按键处理功能的子程序。

对于独立式键盘应用电路，可以采用外部中断和 10 ms 定时中断的方式编制独立式键盘程序。在初始化程序中，设置外部中断 0 为边沿触发，并允许中断。

例 11-3 图 11-1 中，设系统晶振的时钟频率采用 6 MHz，应用 T0 作为定时器，当无键按下时， $\overline{\text{INT0}}$ 输入端为高电平；有键按下时，产生一个下降沿发出中断申请信号；在外部中断 0 服务程序中，设置 T0 为工作方式 1，启动 T0 定时和允许 T0 中断。

在 T0 的 10 ms 定时中断服务程序中，调用是否有键按下的处理子程序，如果判断按键是有效的按下，则调用键识别与键处理子程序进行键处理；如果无按键被有效地按下，说明按键按下引起 $\overline{\text{INT0}}$ 产生的中断是无效的，则直接退出定时中断服务程序。

消抖动与独立式键盘程序如下：

```

ORG    0000H
SJMP   MAIN
ORG    0003H
SJMP   INT00
ORG    000BH
SJMP   T0INT
ORG    0030H
MAIN:MOV    SP,#6FH
      SETB   EX0
      SETB   IT0           ;设置外部中断 0 负跳变触发方式
      SETB   EA           ;设置外部中断 0 的中断允许
      SJMP   $
INT00:CLR    EX0           ;外部中断 0 服务程序,关外部中断 0
      MOV    TLO,#0ECH
      MOV    TH0,#78H      ;T0 定时 10 ms,按照式(5-3)计算定时初值
      SETB   ET0           ;允许定时器 0 中断
      SETB   TR0           ;启动定时器 0 定时
      RETI
T0INT:PUSH   ACC           ;滤波 10 ms 进入定时 0 中断服务程序
      CLR    TR0
      CLR    ET0           ;定时器 0 停止定时,并关定时器 0 的中断
      LCALL  KCS           ;调用按键读入预处理子程序 KCS
      JZ     RTN           ;A=0 无键有效按下,转移退出中断服务程序
      LCALL  KEY          ;(A)≠0,有键按下,调用键识别与键处理子程序
RTN:POP     ACC
      SETB   EX0           ;外部中断 0 允许,为下一次按键处理做好装备
      RETI

```

图 11-1 的 4 个独立式按键与 51 单片机的连接电路中，有键按下时，与门输出由 1→0，产生一次外部中断 0 申请，在外部中断 0 服务程序中关闭外部中断 0 的中断允许，确保键按

下只会执行一次外部中断 0 服务程序。由于 T0 是在外部中断 0 服务程序中被启动，滤波 10 ms 才进入定时中断服务程序，在定时中断服务程序中，调用按键状态进行预处理的子程序 KCS，然后判断按键是否是有效按下，如果不是则不作按键处理；如果按键是有效的被按下，才调用执行一次键识别子程序 KEY，确保按键闭合时只调用一次处理按键功能的子程序。

11.1.2 矩阵式键盘

矩阵式键盘又称行列式键盘，行列式键盘与 51 单片机的连接电路如图 11-3 和图 11-4 所示。两个图中，P1 口的 P1.4 ~ P1.7 作为 4 根行线，P1.0 ~ P1.3 作为 4 根列线，在其行、列交汇点连接有 16 个按键。

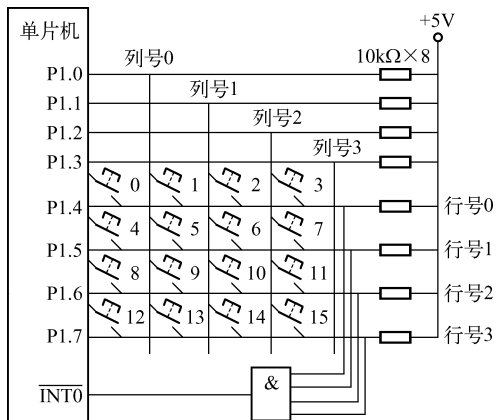


图 11-3 线反转法矩阵式键盘

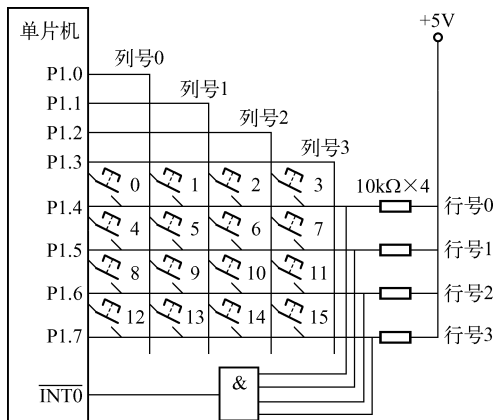


图 11-4 扫描法矩阵式键盘

如果 51 单片机的 P1 口需要连接其他外部装置，则可以应用 I/O 接口芯片来连接矩阵式键盘，图 11-5 中利用 8155 的 PC0 ~ PC3 作为行线，PA0 ~ PA3 作为列线，组成有 $4 \times 4 = 16$ 个按键的矩阵式键盘。也可以应用 8255 来连接矩阵式键盘，其电路读者可以自行设计。

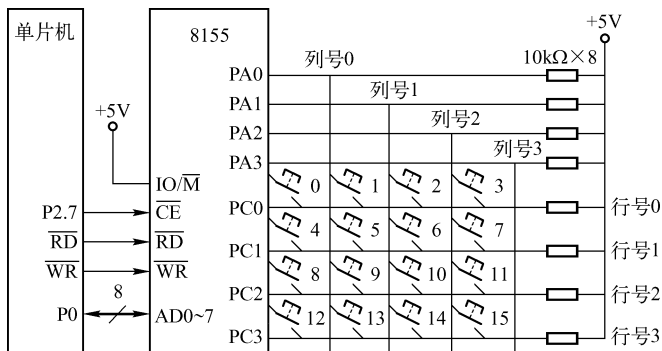


图 11-5 与 8155 连接的线反转法矩阵式键盘电路

1. 线反转法键盘的按键识别原理

图 11-3 和图 11-5 是线反转法矩阵式键盘电路。以图 11-3 的线反转法矩阵式键盘为例来说明其线反转法按键识别的工作原理。

1) 4 根列线 P1.0 ~ P1.3 全部输出低电平, 4 根行线 P1.4 ~ P1.7 作为输入, 若 16 个键中无键按下, P1.4 ~ P1.7 输入全部为高电平; 若某一行中有一个按键被按下, 则对应该行的输入线电平为低电平, 这样能够确定哪一行有按键按下, 并记录行号。

2) 输入线和输出线反转, 即 4 根列线 P1.0 ~ P1.3 作为输入, 4 根行线 P1.4 ~ P1.7 全部输出低电平, 某一列有按键按下, 则对应该列的输入线电平为低电平, 这样能够确定哪一列有按键按下, 并记录列号。

3) 对于图 11-3 和图 11-5 的线反转法矩阵式键盘电路, 行号 $\times 4 +$ 列号的值恰好是其键值。

一般地, 无论矩阵式键盘由多少行和列的按键组成, 只要识别被按下的这个按键在矩阵中的行号和列号, 即可根据设计者对键盘按键编码的方法得到该按键的键值。

2. 扫描法键盘的按键识别原理

图 11-4 是扫描法矩阵式键盘电路, 4 根列线 P1.0 ~ P1.3 为输出, 4 根行线 P1.4 ~ P1.7 作为输入, 键盘扫描法子程序的按键识别的工作原理如下:

1) 获得按下键的行号。4 根列线 P1.0 ~ P1.3 全部输出低电平, 4 根行线 P1.4 ~ P1.7 作为输入, 若 16 个键中无键按下, P1.4 ~ P1.7 输入全部为高电平; 若某一行中有一个按键被按下, 则对应该行的输入线电平为低电平, 这样获得被按下的按键的行号, 并记录行号。

2) 判断被按下按键的列号, 需要采用逐列扫描的方式, 直到找到该按键的列号。

① 第 0 列键扫描。设置 P1.0 输出低电平, P1.1 ~ P1.3 输出高电平, 读入 P1.4 ~ P1.7 的输入状态, 若 P1.4 ~ P1.7 输入不全为高电平, 则获得被按下的按键的列号 0, 并记录列号为 0, 键扫描过程结束, 进行键值计算, 调用该键处理子程序, 键扫描子程序返回; 若 P1.4 ~ P1.7 输入全部为高电平, 则第 0 列无键按下, 则进入第 1 列键扫描过程。

② 第 1 列键扫描。设置 P1.1 输出低电平, 其他列线输出高电平, 参照第①步第 0 列扫描方法的过程, 获得列号 1 及对该键进行键处理, 键扫描子程序返回; 或该列无键按下进入第 2 列键扫描过程。

③ 第 2 列键扫描。设置 P1.2 输出低电平, 其他列线输出高电平, 参照第①步第 0 列扫描方法的过程, 获得列号 2 及对该键进行键处理, 键扫描子程序返回; 或该列无键按下进入第 3 列键扫描过程。

④ 第 3 列键扫描。设置 P1.3 输出低电平, 其他列线输出高电平, 参照第①步第 0 列扫描方法的过程, 获得列号 3 及对该键进行键处理, 键扫描子程序返回; 或判断该列无键按下, 键扫描子程序返回。

扫描法的矩阵式键盘的编程读者可以自己编制, 或参考其他教材关于这方面的内容描述。

3. 线反转法键盘的中断控制方式编程

(1) 确定按下键行号的子程序

例 11-4 图 11-3 中, 4 根列线 P1.0 ~ P1.3 输出全部为 0, 若有键按下, 则 P1.4 ~ P1.7 的输入不会全为高电平。连接的某一行有按键被按下, 则对应输入端口的输入状态为 0。

一个按键被按下, P1.4 ~ P1.7 取反后的结果将是 10H、20H、40H、80H 中的一个值, 高四位与低四位交换则转换为 01H、02H、04H、08H, 再通过查表来获得行号。

```

HKCS:MOV    P1,#0F0H      ;P1.0~P1.3 输出全部为 0,设置 P1.4~P1.7 全部为 1 状态
        MOV     A,P1
        ORL     A,#0FH      ;A 的低四位全部为 1,高四位与 P1.4~P1.7 的状态相同
        CPL     A           ;对 A 取反,低四位为 0,高四位与 P1.4~P1.7 的状态相反
        SWAP    A           ;A 的高四位与低四位交换
        MOV     DPTR,#TBL    ;表头的基址地址送入 DPTR
        MOVC    A,@ A + DPTR
        MOV     30H,A        ;行号保存在 30H 中
        RET
TBL:DB      0FFH,00H,01H,0FFH,02H,0FFH,0FFH,0FFH,03H

```

此时, 30H 存储单元中的值: 按下键的行号 0 为 00H; 行号 1 为 01H; 行号 2 为 02H; 行号 3 为 03H。比如, 图 11-3 中, 10 号键被按下, 查表指令的 (A) = 04H, 则 (30H) = 02H。

(2) 确定按下键列号的子程序

例 11-5 图 11-3 中, 设置 4 根行线 P1.4~P1.7 输出全部为 0, 若有键按下, 则 4 根列线 P1.0~P1.3 的输入不会全为高电平。连接的某一列有按键被按下, 则对应的列线输入状态为 0。

一个按键被按下, P1.0~P1.3 取反后的结果将是 01H、02H、04H、08H 中的一个值, 可以通过查表来获得列号。

```

LKCS:MOV    P1,#0FH      ;P1.4~P1.7 输出全部为 0,设置 P1.0~P1.3 全部为 1 状态
        MOV     A,P1
        ORL     A,#0F0H    ;A 的高四位全部为 1,低四位与 P1.0~P1.3 的状态相同
        CPL     A          ;对 A 取反,高四位为 0,低四位与 P1.0~P1.3 的状态相反
        MOV     DPTR,#TBL  ;表头的基址地址送入 DPTR
        MOVC    A,@ A + DPTR
        MOV     31H,A      ;列号保存在 31H 中
        RET
TBL:DB      0FFH,00H,01H,0FFH,02H,0FFH,0FFH,0FFH,03H

```

(3) 外部中断 0 和定时中断服务程序

例 11-6 设系统晶振的时钟频率采用 6 MHz, 设置 T0 为工作方式 1。

外部中断 0 服务程序

```

INT00:CLR    EX0          ;外部中断 0 服务程序,关外部中断 0
        MOV     TLO,#0ECH
        MOV     TH0,#78H   ;T0 定时 10 ms,按照式(5-3)计算定时参数初值
        SETB    ET0        ;允许定时器 0 中断
        SETB    TR0        ;启动定时器 0 定时
        RETI

```

T0 中断服务程序

```

T0INT:PUSH    ACC          ;滤波 10 ms 进入定时 0 中断服务程序

```

CLR	TR0	
CLR	ET0	;定时器 0 停止定时,并关定时器 0 的中断
LCALL	HKCS	;调用确定按下键行号的子程序
CJNE	A,#00H,LLKCS	;滤波 10 ms 后,如果(A)=00H,按键是无效按下
SJMP	RTN	;退出 T0 中断服务程序
LLKCS:LCALL	LKCS	;按键是有效按下,调用确定按下键列号的子程序
MOV	B,#04H	
MUL	AB	;列号×4,乘积→A,高 8 位为 0
ADD	A,30H	;行号+列号×4=键值
MOV	30H,A	;键值存放在 30H 中
LCALL	KEY	;调用键值识别处理子程序
RTN: MOV	P1,#0FFH	;设置 P1.0~P1.7 全部为 1,为下一次按键处理做好准备
POP	ACC	
SETB	EX0	;外部中断 0 允许,为下一次按键处理做好装备
RETI		

其主程序的编制与独立式键盘编程的主程序类似,设置外部中断 0 为边沿触发,并允许中断,再增加一条 MOV P1, #0FH 指令即可,使得列线输出低电平。

由于 T0 是在外部中断 0 服务程序中被启动,滤波 10 ms 才进入定时中断服务程序,在定时中断服务程序中,如果按键不是有效的被按下,则不作按键处理,并退出定时中断服务程序;如果按键是有效的被按下,才执行一次键值识别处理子程序,能够确保按键闭合时只调用一次按键功能的子程序。对于键值识别处理子程序,判断(30H)中的键值,根据键值调用相应的按键处理程序,然后返回,读者可以自行编制。

对于图 11-5 的线反转法矩阵式键盘与 8155 连接电路的编程,确定好 8155 命令寄存器、A 口和 C 口的地址,设置 8155 的命令字。初始化程序中,定义 A 口和 C 口为基本的输入输出方式,A 口为输出,输出值为 0,C 口定义为输入;在按下键行号和列号的子程序中,重新定义 C 口和 A 口的输入输出状态,定时器中断服务程序的 RETI 指令之前,再次定义 A 口为输出,输出值为 0,C 口为输入。编程时要特别注意 DPTR 中地址值的变换。

11.2 LED 显示接口电路

11.2.1 LED 显示器

LED (Light Emitting Diode) 显示器是由发光二极管作为显示字段的数码型显示器件,具有显示清晰、成本低廉、配置灵活、与单片机接口简单易行的特点,在单片机应用系统中得到了广泛的应用。

1. LED 显示器结构与分类

LED 显示器内部由 7 段发光二极管组成,因此亦称为 7 段 LED 显示器,由于主要用于显示各种数字符号,故又称为 LED 数码管。每个显示器还有一个圆点型发光二极管(用符号 DP 表示),用于显示小数点,图 11-6 为 LED 显示器的符号与引脚图。根据其内部结构,

LED 显示器可分为共阴极与共阳极两种 LED 显示器，如图 11-7 所示。

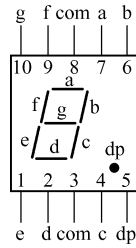


图 11-6 LED 显示器的
符号与引脚图

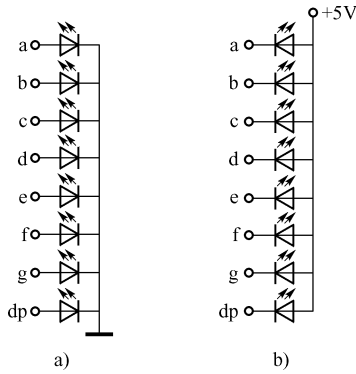


图 11-7 LED 显示器内部结构图
a) 共阴极结构 b) 共阳极结构

1) 共阴极 LED 显示器的内部结构如图 11-7a 所示。图中各二极管的阴极连接在一起，公共端接低电平时，若某段阳极加上高电平则该段发光二极管就导通发光，而输入低电平的段则不发光。

2) 共阳极 LED 显示器的内部结构如图 11-7b 所示。图中各二极管的阳极连在一起，公共端接高电平时，若某段阴极加上低电平则该段发光二极管就导通发光，而输入高电平的段则不发光。

LED 数码管通常有红色、绿色和黄色三种。由于二极管的发光材料不同，数码管有高亮与普亮之分，应用时根据数码管的规格与显示方式等决定是否加驱动电路。

2. LED 显示器的段码

7 段 LED 显示器可采用硬件译码与软件译码两种方式。在数字电路教材中已介绍硬件译码显示方法，如利用 74LS47 等实现译码显示，本节主要介绍软件方式实现译码显示。加在显示器上对应各种显示字符的二进制数据称为段码。数码管中，七段发光二极管加上一个小数点位共计 8 段，因此段码为 8 位二进制数。由于点亮方式不同，共阴与共阳两种 LED 数码管的段码是不同的，详见表 11-1。针对表 11-1 的 LED 数码管的段码，图 11-6 中的 LED 显示器的 dp、g、f~a 段是按照顺序分别与 8 位数据线的 D7~D0 连接。

表 11-1 LED 数码管段码表

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

11.2.2 LED 并行显示技术

LED 显示的设计方式主要有动态显示和静态显示。

动态显示方式是利用单片机控制 LED 显示器逐位点亮，而不是同时点亮。由于人眼的视觉残留效应，仍然感觉显示器是同时点亮的，这不但要求单片机按照顺序逐个点亮每个 LED 显示器，而且要求不断地定时对整个 LED 显示器进行动态显示处理，并且动态处理的速度应足够快，否则会有闪烁感。

静态显示方式是指控制输出到 LED 显示器的 8 段数据具有锁存功能，只有需要刷新显示数据时才调用执行一次显示子程序。

静态显示方式电路规模较大，每一位 LED 显示器需要一个具有驱动显示的锁存器，但显示处理占用微处理器时间少。动态显示方式电路规模小，但占用微处理器时间多，如果动态显示处理程序设计不当，容易导致设计的微处理器应用系统难以满足实时处理数据的需求。

1. LED 显示器动态显示

(1) LED 显示器动态显示并行接口电路

图 11-8 是应用 8155 扩展 6 位 LED 显示器的动态显示接口电路。8155 的 C 口作为位控输出口。7 路达林顿反向驱动输出芯片 MC1413 作为位控驱动；A 口作为段控输出口，输出 8 位段码。为增加显示亮度，采用单向驱动芯片 74LS244 进行段码显示输出驱动。

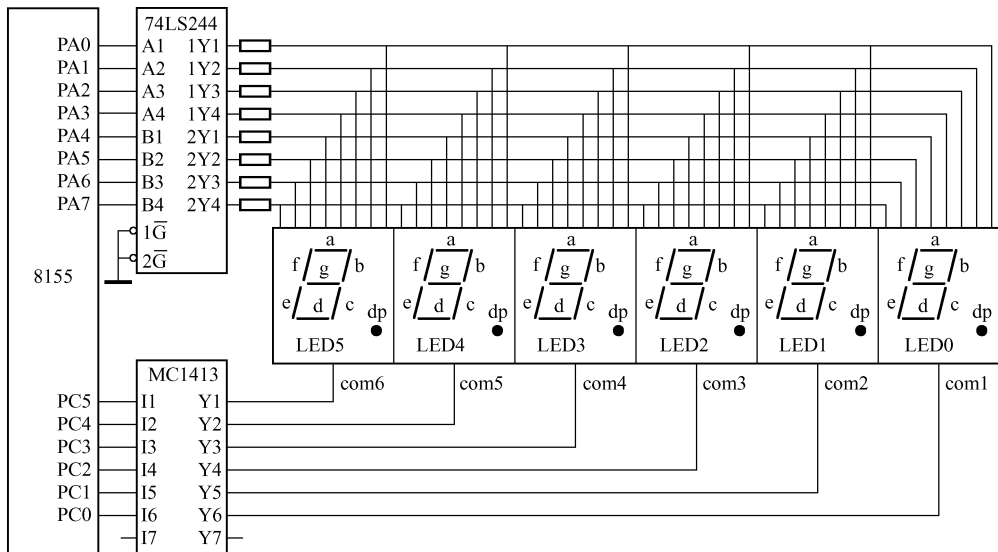


图 11-8 8155 芯片扩展 6 位动态 LED 显示器

在实现动态显示时，除了必须给各位数码管提供段码外，还必须对各位显示器进行位的控制，即进行段控与位控。

(2) 建立显示缓冲区

显示缓冲区是指在内存中开辟的一块存储区域，专门用于存放待显示的数据，存储区域的大小应与 LED 数码管位数相同。以图 11-8 的电路为例，设显示缓存单元为 35H ~ 30H，图 11-8 中最左边的 LED5 为最高位，其显示数据存放在 35H 中，LED4 ~ LED0 的显示数据依次存放在 34H ~ 30H 存储单元中，最右边的 LED 显示器 LED0 为最低位，其显示数据存放在 30H 中。

(3) 动态显示子程序设计

例 11-7 设图 11-8 中的 LED 数码管为共阴极结构的 LED 数码管, 参考图 11-5 中的 8155 与单片机的接口电路连接, 8155 命令寄存器的地址为 7FF8H, A 口 (即段控口) 地址为 7FF9H, C 口 (即位控口) 地址为 7FFBH。寄存器 R0 作为显示数据单元的地址指针, 寄存器 R3 存放当前选通的位控码, DELAY 为延时子程序, 延时时间大约为 1 ms。

```
DISPLAY: MOV    DPTR, #7FF8H      ;8155 命令寄存器地址→DPTR
          MOV    A, #0DH          ;控制字:00001101, 方式 2: A 口、C 口输出方式
          MOVX   @DPTR, A
          MOV    R0, #30H         ;建立显示缓冲区首址, 指向 LED0 的显示数据地址
          MOV    R3, #01H         ;LED0 的位控码, MC1413 反向后为 FEH
          MOV    A, R3            ;从右边第一位数码管 LED0 开始显示
LD0:      MOV    DPTR, #7FFBH     ;置位控口(C 口)地址
          MOVX   @DPTR, A        ;输出当前选通的位控码
          MOV    A, @R0           ;取待显示数据
DIS0:     MOV    DPTR, #TAB       ;置共阴段码表首地址
          MOVC   A, @A + DPTA     ;查表法得到对应 LED 显示位的显示段码
DIS1:     MOV    DPTR, #7FF9H     ;置段控口(A 口)地址
          MOVX   @DPTR, A        ;输出显示段码
          ACALL  DELAY            ;延时以使该位 LED 显示器作显示停留
          INC    R0               ;指向下一缓冲单元, 即指向下一位 LED 显示数据地址
          MOV    A, R3
          JB     ACC. 6, LD1       ;最高一位 LED5 显示了吗? 已显示则子程序返回
          RL     A                ;修改选通的位控码, 即指向高一位 LED 显示位控位
          MOV    R3, A            ;下一位 LED 显示的选通位控码送 R3 保存
          AJMP   LD0              ;继续下一位 LED 显示
LD1:      RET
TAB: DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH, 77H, 7CH, 39H, 5EH, 79H, 71H, 00H
```

如果图 11-8 中的 LED 数码管为共阳极结构的 LED 数码管, 只需要将该动态显示子程序稍作修改即可, 读者可以自己修改程序编制。

2. LED 显示器静态显示

(1) LED 显示器静态显示并行接口电路

图 11-9 是应用 8155 扩展 6 位共阳 LED 显示器的静态显示接口电路。每个 LED 显示器的位控口 com 直接与 +5V 电源连接; 每个 LED 显示器的段控口与一个锁存器 74LS373 的输出端口连接, A 口作为段控输出口输出 8 位段码与 6 个锁存器 74LS373 的 8 位数据输入端连接, 每个 74LS373 的 G 端作为 LED 显示的选通位分别由 PC5 ~ PC0 控制, 高电平有效, 在选通位信号的下降沿实现显示段码的锁存。

(2) 静态显示子程序设计

例 11-8 设显示缓存单元为 45H ~ 40H, 图 11-9 中最左边的 LED5 为最高位, 其显示数据存放在 45H, LED4 ~ LED0 的显示数据依次存放在 44H ~ 40H 存储单元中, 最右边 LED0 为最低位, 其显示数据存放在 40H 中。

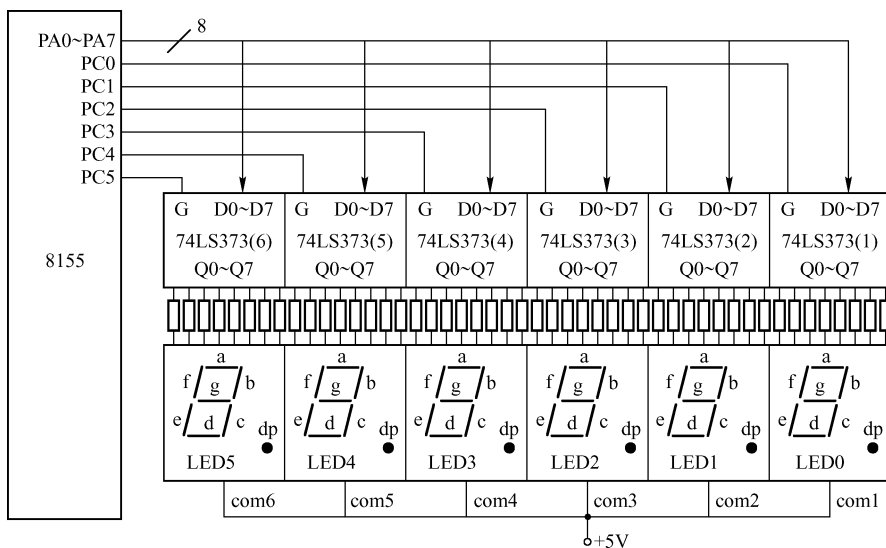


图 11-9 8155 芯片扩展 6 位静态 LED 显示器

参考图 11-5 中的 8155 与单片机的接口电路连接，8155 命令寄存器的地址为 7FF8H，A 口（即段控口）地址为 7FF9H，C 口（即显示选通位的位控口）地址为 7FFBH。寄存器 R0 作为显示数据单元的地址指针，寄存器 R3 存放当前位的位选通值。

```

JDISPLAY: MOV    DPTR, #7FF8H        ;8155 命令寄存器地址→DPTR
           MOV    A, #0DH            ;控制字:00001101,方式 2;A 口、C 口输出方式
           MOVX   @DPTR, A
           MOV    R0, #40H           ;建立显示缓冲区首址,指向 LED0 的显示数据地址
           MOV    R3, #01H           ;置显示选通位的位控初值,指向 LED0
DIS:      MOV    A, @R0              ;取待显示的数据
           MOV    DPTR, #TAB         ;置共阳段码表首地址
           MOVC   A, @A + DPTA       ;查表法得到对应段码
DIS1:     MOV    DPTR, #7FF9H        ;置段控口地址
           MOVX   @DPTR, A          ;输出段码→A 口
DIW:      MOV    A, R3              ;显示位选通的位控值→A
           MOV    DPTR, #7FFBH       ;置显示位选通的位控口地址
           MOVX   @DPTR, A          ;显示位选通的位控值→C 口,刷新该 LED 显示内容
           JB     ACC.6, LD1          ;最高一位显示了吗? 已显示则返回
           RL     A                  ;修改显示位选通的位控值,指向高一位 LED 显示器
           MOV    R3, A              ;显示位选通的位控值送→R3 保存
           INC    R0                 ;指向高一位待显示数据的缓冲单元
           AJMP   DIS                ;继续显示下一位

LD1: RET
TAB: DB  0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H, 88H, 83H, 0C6H, 0A1H
      DB  86H, 8EH, 0FFH

```


ADC0809：最高位地址线与 $\overline{Y7}$ 连接，P2.7 ~ P2.4 为 1111；8 路模拟量输入 IN7 ~ IN0 的地址：FFFFH ~ FFF8H。

8155：片选端 $\overline{CE}$ 与 $\overline{Y6}$ 连接，P2.7 ~ P2.4 为 1101。

命令寄存器：DFF8H；A 口：DFF9H；B 口：DFFAH；C 口：DFFBH；定时/计数器低 8 位地址：DFFCH；定时/计数器高 6 位地址：DFFDH。

ADC0809 的 EOC 与 51 单片机的 P3.4 连接。

2. 键盘、LED 显示器、A-D 和 D-A 的综合应用编程

按照图 11-10 电路的结构以及四个功能模块的功能要求，按照以下步骤编程：

1) 确定各扩展接口芯片的地址，包括芯片内部寄存器和 I/O 端口以及模拟量输入地址。

2) 确定中断源。图 11-10 的键盘处理应用外部中断 0 和定时器 0 的 10 ms 定时中断；ADC0809 的数据采集应用定时器 1 的定时中断。

3) 存储空间的配置。配置好四个功能模块的单片机内部存储区域，包括位单元和通用寄存器的配置，一是避免子程序中需要编制过多的压栈和出栈的指令；二是避免存储单元配置不当，导致出错。

4) 确定程序模块。

① 线反转法键盘：参考 11.1.2 的第 3 节的描述：初始化程序，外部中断 0 和定时器 0 中断服务程序，定时器 0 中断服务程序还包括确定按下键行号和列号、键处理子程序。

② ADC0809 的数据采集。参考 10.2.2 的第 3 节的描述和例 10-4：初始化程序，定时器 1 中断服务程序，包括对多路模拟信号的定时采样。

③ DAC0832 的单缓冲方式输出。在测控系统中，通常是在键盘的按键处理和数据采集的数据处理计算后，根据系统功能要求通过 DAC0832 输出模拟量。

④ 静态 LED 显示：参考 11.2.2 第 2 节的描述，通常只在需要刷新显示数据时调用静态 LED 显示子程序。

5) 编程。

① 确定程序结构。初始化程序的主程序模块；1 个外部中断 0 的中断服务程序；2 个定时器中断服务程序；各个功能模块的子程序，包括功能模块子程序中的子程序。绘制程序流程图。

② 按照程序模块和程序流程图编制程序。注意主程序通常是无限循环程序，且比较简单。也要注意各个功能模块之间的数据传递，DPTR 中的地址内容的变换。

11.4 LCD 显示接口电路

11.4.1 LCD 显示器件

液晶显示器（LCD）具有功耗低、体积小、重量轻、超薄等许多其他显示器无法比拟的优点，已被广泛用于单片机控制的智能仪器、仪表和低功耗电子产品中。LCD 可分为段位式 LCD、字符式 LCD 和点阵式 LCD。其中，段位式 LCD 和字符式 LCD 只能用于字符和数字

的显示，不能显示图形、曲线和汉字；而点阵式 LCD 不仅可以显示字符、数字，还可以显示各种图形、曲线及汉字，并且可以实现屏幕上下左右滚动、动画等功能，用途十分广泛。本节主要介绍点阵式液晶显示器 MGLS12864 模块的基本结构、指令系统，以及与 51 单片机的接口电路设计。

1. MGLS12864 液晶显示模块内部结构与引脚功能

图 11-11 为 MGLS12864 液晶显示模块逻辑电路图。该模块内部使用 2 片 KS0108B 或其兼容控制器 HD61202 作为列驱动器，使用一片 KS0107B 或其兼容芯片 HD61203 作为行驱动器。KS0108B 芯片具有内置 64 位×64 位的显示存储器，显示屏上各像素点的显示状态与显示存储器的各位数据一一对应，显示存储器的数据直接作为图形显示的驱动信号。当某二进制位为“1”时，相应的像素点显示；为“0”时，相应的像素点不显示。表 11-2 为 MGLS12864 液晶显示模块引脚说明。

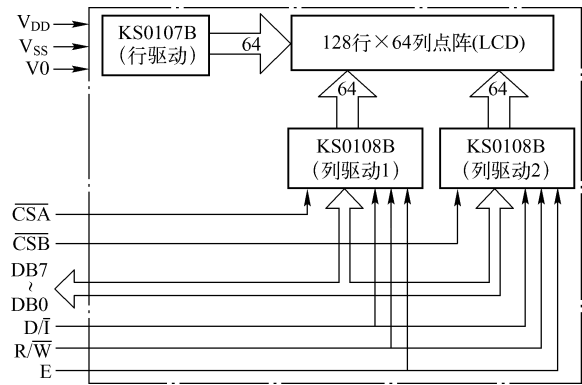


图 11-11 MGLS12864 液晶显示模块逻辑电路图

表 11-2 MGLS12864 液晶显示模块引脚说明

引 脚 号	引 脚 名 称	引 脚 功 能	
1	$\overline{\text{CSA}}$	片选 1	$\overline{\text{CSA}} \ \overline{\text{CSB}} = 01$ ：选择列驱动芯片 1（左芯片） $\overline{\text{CSA}} \ \overline{\text{CSB}} = 10$ ：选择列驱动芯片 2（右芯片）
2	$\overline{\text{CSB}}$	片选 2	
3	V_{SS}	数字地	
4	V_{DD}	逻辑电源 +5 V	
5	V_0	对比度调节，电压变化范围为 $-10 \sim 0 \text{ V}$ （视不同型号）	
6	$\text{D}/\overline{\text{I}}$	指令、数据通道选择。为 1 时：数据操作；为 0 时：指令操作	
7	$\text{R}/\overline{\text{W}}$	读写选择。为 1 时：读操作；为 0 时：写操作	
8	E	使能信号，高电平时读出数据，下降沿时写入数据	
9 ~ 16	$\text{DB}_0 \sim \text{DB}_7$	数据线	
17	LED +	LED 背光正电源端。底背光：<4.0V；边背光：≤4.1V	
18	LED -	LED 背光接地端	

2. KS0108B 列驱动器显示 RAM 地址结构

MGLS12864 液晶显示模块为 128 × 64 点阵液晶显示，模块内部有两片列驱动芯片 KS0108B。该芯片内部有 64 × 64 = 4096 位显示 RAM，RAM 中的每一位数据对应 LCD 屏上一

个点的亮、暗状态。

表 11-3 为 KS0108B 芯片显示 RAM 的地址结构。该存储器总计分 8 页、64 列，每页对应显示屏上 8 行 64 列。显示器上每 8 点固定对应显示 RAM 中的一个字节的数据，例如，第 0 页第 0 列的 DB0 ~ DB7 为一个字节。单片机写入或读出显示存储器的数据代表显示屏上某一点列上的垂直 8 点行的数据，例如，单片机写入第 6 页地址为 3EH (62) 单元的显示数据，则液晶显示屏的第 62 列的第 48 ~ 55 行显示该 3EH (62) 单元的显示数据。从表 11-3 中可以看出，DB0 代表某页最上一行的点数据，DB1 为某页第二行的点数据，…，DB7 为某页第八行的点数据。

表 11-3 KS0108B 芯片显示 RAM 的地址结构

页 地 址	列地址						行 地 址
	0	1	2	...	62	63	
0	DB0	DB0	DB0	...	DB0	DB0	0
	...	...	...	...	...	...	...
	DB7	DB7	DB7	...	DB7	DB7	7
1	DB0	DB0	DB0	...	DB0	DB0	8
	...	...	...	...	...	...	...
	DB7	DB7	DB7	...	DB7	DB7	15
⋮	⋮						⋮
6	DB0	DB0	DB0	...	DB0	DB0	48
	...	...	...	...	...	...	...
	DB7	DB7	DB7	...	DB7	DB7	55
7	DB0	DB0	DB0	...	DB0	DB0	56
	...	...	...	...	...	...	...
	DB7	DB7	DB7	...	DB7	DB7	63

3. 液晶显示模块指令系统

液晶控制器 KS0108B 一共有七条指令，从作用上可分为两类：显示状态设置指令和数据读/写操作指令，现分别介绍如下：

(1) 显示开/关指令

R/ $\overline{W}$	D/ $\overline{I}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	1	1	1	1/0

例如，当 DB0 = 1 时，LCD 显示 RAM 中的内容；当 DB0 = 0 时，关闭显示。

(2) 显示起始行设置指令

R/ $\overline{W}$	D/ $\overline{I}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	1	显示起始行号(0~63)					

(3) 页设置命令

R/ $\overline{W}$	D/ $\overline{I}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	1	1	1	页号(0~7)		

(4) 列地址设置命令

R/ $\overline{W}$	D/ $\overline{I}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	显示列地址(0~63)					

(5) 读状态指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	BUSY	0	ON/OFF	RESET	0	0	0	0

读状态指令用来查询液晶显示模块内部控制器的工作状态，各参量意义如下：

BUSY：该位为 1，表示内部在工作；该位为 0，表示处于正常状态。

ON/OFF：该位为 1，显示器关闭；该位为 0，显示器打开。

RESET：该位为 1，处于复位状态；该位为 0，处于正常状态。

在 BUSY 和 REST 为 1 状态时，除可进行读指令操作外，其他指令均不对液晶显示模块产生作用。

(6) 写数据指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	待写数据							

(7) 读数据指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	读显示数据							

读写数据指令每执行完一次读、写操作，列地址就自动增一。需注意在进行读操作之前，必须有一次空读操作，紧接着再读才会读出所要读单元中的数据。

11.4.2 单片机与液晶显示模块接口电路与驱动

液晶显示模块与 51 单片机有两种接口方式：直接控制方式和间接控制方式，图 11-12 为间接控制方式连接电路，直接控制方式请参阅有关资料。

目前各种型号的液晶模块生产厂家一般会提供与微处理器的接口参考电路，包括与单片机接口的电路，并提供相应的驱动程序，读者可以查阅相关资料。

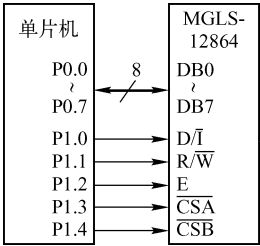


图 11-12 间接控制方式接口电路

习题

- 11-1 说明产生键盘抖动的原因，消除按键抖动的方法有哪几种？原理是什么？
- 11-2 什么是独立式按键？什么是行列式按键？
- 11-3 说明矩阵式键盘按键按下的识别原理，键盘程序通常由几部分构成？
- 11-4 对于图 11-4 扫描法矩阵式键盘，按照扫描法按键识别原理编写键识别并得到其键号的子程序。
- 11-5 对于图 11-5 的 51 单片机与 8155 连接的线反转法矩阵式键盘电路，按照线反转法按键识别原理编写键识别并得到其键号的子程序。
- 11-6 共阳极数码管与共阴极数码管有什么不同？可否在电路中互换使用？
- 11-7 说明为什么待显示的数据必须查表转换成段码值才能够在 LED 数码管中得到正确的显示？
- 11-8 表 11-1 的 LED 数码管段码表中未列出小数点的显示段码值，假设需要显示 5.7

两个数据，请列出 5.7 的共阴和共阳的段码值。

11-9 LED 数码管动态显示原理是什么？与静态显示有何不同？

11-10 如果将图 11-8 中的共阴 LED 数码管改为共阳极结构的 LED 数码管，该电路图需要作何修改，请说明为什么？

11-11 如果将图 11-8 中的 LED 数码管改为共阳极结构的 LED 数码管，参照例 11-7 共阴 LED 数码管动态显示的子程序，将该子程序修改为共阳 LED 数码管动态显示子程序。

11-12 如果将图 11-9 中的 6 位静态共阳 LED 数码管改为共阴极结构的 LED 数码管，该电路图需要作何修改，请说明为什么？

11-13 如果将图 11-9 中的 6 位静态共阳 LED 数码管改为共阴极结构的 LED 数码管电路，参照例 11-8 的 6 位静态共阳 LED 数码管显示的子程序，将该子程序修改为 6 位静态共阴 LED 数码管显示的子程序。

11-14 对于图 11-10 和第 8 章图 8-16 的两个 51 单片机扩展电路，结合这两个扩展电路图，说明如何用一块 51 单片机来扩展这些芯片，各个芯片的引脚与 51 单片机接口的连接是否会有变化？列出每块芯片的地址。

11-15 简述 MGLS12864 液晶显示模块的内部结构及工作原理。

11-16 在网络中查阅 MGLS12864 液晶显示模块与 51 单片机的接口电路，简述该接口电路原理。

11-17 根据在网络中查阅到的 MGLS12864 液晶显示模块与 51 单片机的接口电路，修改图 11-9，将图 11-9 中的 6 位静态 LED 显示器电路用 MGLS12864 液晶显示模块取代，编制 MGLS12864 液晶显示模块的地址。

第 12 章 单片机的串行总线接口技术

单片机的并行总线扩展（利用三总线 DB、AB、CB 进行的系统扩展）已不再是单片机应用系统唯一的扩展结构。目前，串行总线扩展技术已得到广泛地应用。例如，DALLAS 公司的单总线（1-Wire）接口、Philips 公司的 I²C 串行总线接口、Motorola 公司的 SPI 串行外设接口以及 Microwire 总线三线同步串行接口。

本章介绍上述串行扩展接口总线的工作原理及特点，重点介绍 I²C 串行扩展技术和 SPI 总线接口。

单片机的串行扩展技术与并行扩展技术相比具有显著的优点，串行接口器件与单片机接口时需要的 I/O 口线很少（仅需 1~4 条），串行接口器件体积小，因而占用电路板的空间小，约为并行接口器件的 10%，明显减少了电路板空间和成本。

除上述优点，还有工作电压宽、抗干扰能力强、功耗低和数据不易丢失等特点。串行扩展技术在 IC 卡、智能仪器仪表以及分布式控制系统等领域得到广泛应用。

12.1 I²C 总线接口

12.1.1 I²C 总线概述

I²C 总线使用广泛，是很有发展前途的芯片间串行扩展总线。

I²C 总线只有两条信号线，一是数据线 SDA，二是时钟线 SCL。两条信号线都是双向传输的信号线，所有连接到 I²C 上的器件的数据线都与 SDA 线连接；各器件时钟线均与 SCL 线连接。I²C 系统的基本结构如图 12-1 所示。I²C 总线单片机（如 PHILIPS 公司的 8xC552）可以直接与 I²C 接口的各种扩展器件（如存储器、I/O 芯片、A-D、D-A、键盘、显示器和日历/时钟）连接。

由于 I²C 总线的寻址采用纯软件的寻址方法，无须片选线的连接，这样就大大简化了总线数量。

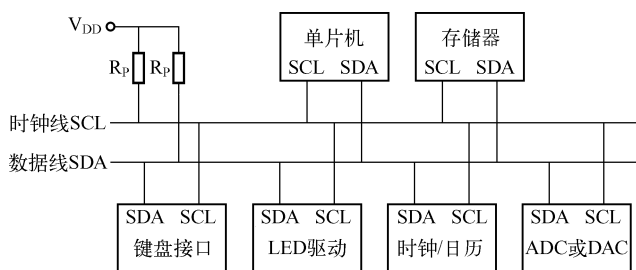


图 12-1 I²C 串行总线系统的基本结构

I²C 的运行由主器件（主机）控制。主器件是指启动数据的发送（发出起始信号）、发

出时钟信号、传送结束时发出终止信号的器件，主器件通常由单片机来担当。

从器件（从机）可以是存储器、LED 或 LCD 驱动器、A - D 或 D - A 转换器、时钟/日历器件等，从器件必须带有 I²C 串行总线接口。

当 I²C 总线空闲时，SDA 和 SCL 两条线均为高电平。由于连接到总线上器件（节点）输出级必须是漏极或集电极开路，只要有一器件任意时刻输出低电平，都将使总线上的信号变低，即各器件的 SDA 及 SCL 都是“线与”关系。

由于各器件输出端为漏极开路，故必须通过上拉电阻接正电源（见图 12-1 中的两个电阻），以保证 SDA 和 SCL 在空闲时被上拉为高电平。

SCL 线上的时钟信号对 SDA 线上的各器件间的数据传输起同步控制作用。SDA 线上的数据起始、终止及数据的有效性均要根据 SCL 线上的时钟信号来判断。

在标准 I²C 模式中，数据的传输速率为 100 kbit/s，高速模式下可达 400 kbit/s。

总线上扩展的器件数量不是由电流负载决定的，而是由电容负载确定的。I²C 总线上每个节点器件的接口都有一定的等效电容，连接的器件越多，等效电容值越大，这会造成信号传输的延迟。总线上允许的器件数以所有器件的等效电容量不超过 400 pF（通过驱动扩展可达 4000 pF）为宜，据此可计算出总线长度及连接器件的数量。

连到 I²C 总线上的每个器件有一个唯一的地址，扩展器件时也要受器件地址数目的限制。

I²C 系统允许多主器件，究竟哪一主器件控制总线要通过总线仲裁来决定。如何仲裁，可查阅 I²C 仲裁协议。但在实际应用中，经常遇到的是以一个单片机为主机，其他外围接口器件为从机情况，在这种情况下，时钟线 SCL 为单向传输信号线，由单一主机发出。

12.1.2 串行 EEPROM 概述

EEPROM（Electrically Erasable Programmable Read - Only Memory），电可擦除可编程只读存储器，是一种掉电后数据不丢失的存储芯片。EEPROM 可以应用微处理器或专用设备擦除已有信息，重新写入数据，写入的数据在常温下可以保持十年。EEPROM 芯片有两类接口：并行接口和串行接口。并行 EEPROM 相对容量大、速度快、功耗大、价格贵，但读/写方法简单；串行 EEPROM 体积小、功耗低、价格便宜，使用中占用系统信号线少，但读/写速度相对慢，读/写方法稍复杂一些。对于系统中那些需要较少修改的代码或参数，通常所需要的字节数和写入的次数不会很多，写入速度也没有太高要求，这种情况下采用串行 EEPROM 是非常合适的。本节以 24C02 为例介绍二线制串行 EEPROM 的使用。

12.1.3 24C02 引脚功能

24C02 是一块有 2048 位，即 256 个 8 位字节存储容量的串行 CMOS EEPROM 芯片。该芯片有 8 个引脚，封装有 PDIP、SOIC、TSSOP 和 MSOP，图 12-2 为 PDIP 封装引脚图。

SCL——串行时钟输入引脚，输入发送或接收数据的时钟信号。

SDA——串行数据/地址引脚，双向信号传输，用于器件所有数据的发送或接收，SDA 是一个开漏输出引脚，可与其他器件的开漏输出或集

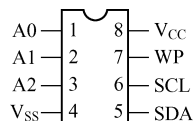


图 12-2 24C02 引脚图

电极开路输出端进行线或（Wire-OR），使用时需要连接一个上拉电阻。

A0 A1 A2——器件地址定义输入引脚，输入器件地址，最多可级联 8 个同类型 EEPROM 器件。如果只有一块 24C02 芯片连接在 I²C 总线上，A0 A1 A2 三个引脚可悬空或连接到 V_{SS}。

WP——写保护，如果 WP 引脚连接到 V_{CC}，所有的内容都被写保护，只能读。当 WP 引脚连接到 V_{SS}或悬空时，允许器件进行正常的读/写操作。

V_{CC}——电源输入，+1.8~6.0V 的工作电压。如果 V_{CC} 低于 3.8V，禁止内部的擦/写操作。

V_{SS}——接系统地。

12.1.4 工作方式

24C02 的 I²C 总线协议定义如下：

- 1) 只有在总线空闲时才允许启动数据传送。
- 2) 在数据传送过程中，当时钟线 SCL 为高电平时，数据线 SDA 必须保持稳定状态，不允许有跳变。因为 SCL 为高电平时，SDA 的任何电平变化将被看作总线的起始或停止信号。

24C02 的 I²C 总线的时序状态定义如图 12-3 所示。

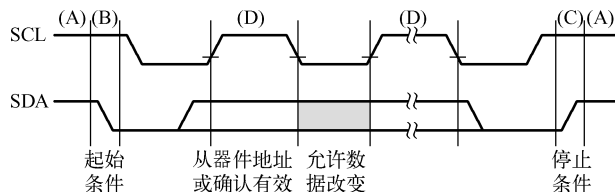


图 12-3 串行总线状态定义

- ① 总线非忙状态（A 段）。数据线 SDA 和时钟线 SCL 都保持为高电平。
- ② 启动数据传输（B 段）。当时钟线 SCL 为高电平状态时，数据线 SDA 由高电平变为低电平的下降沿被认为是“启动”信号。只有出现“启动”信号以后，其他的命令才有效。
- ③ 停止数据传输（C 段）。当时钟线 SCL 为高电平状态时，数据线 SDA 由低电平变为高电平的上升沿被认为是“停止”信号。随着“停止”信号的出现，所有的外部操作都结束。
- ④ 数据有效（D 段）。出现“启动”信号以后，时钟线为高电平状态时数据线的状态必须是稳定的，这时数据线的状态就表示要传送的有效数据。

数据线上数据的改变必须在时钟线为低电平期间完成，每一位数据传输占一个时钟周期脉冲。每组数据的传输都是由“启动”信号开始，结束于“停止”信号。在“启动”与“停止”信号之间传输的每组数据字节数由微处理器确定，从理论上来说字节数没有限制。

- 3) 应答信号。I²C 总线数据传送的确认应答时序如图 12-4 所示。I²C 总线数据传送时，每成功地传送一个 8 位字节数据后，接收器都必须产生一个应答信号，应答的器件在第 9 个时钟周期将数据线 SDA 拉低，表示其已收到一个 8 位字节数据。

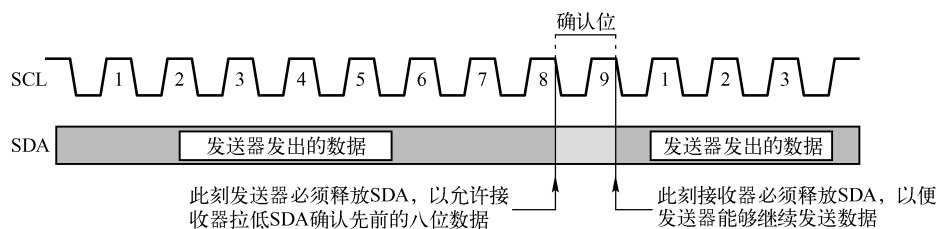


图 12-4 确认应答时序图

24C02 在接收到起始信号和从器件地址，并确认被主器件选中之后，要发出一个应答信号，如果主器件已选择了写操作，则 24C02 在每接收一个 8 位字节数据之后发出一个应答信号。如果主器件已选择了读操作，24C02 发送一个 8 位数据后，释放 SDA 线并监视主器件的应答信号，一旦接收到主器件的应答信号，24C02 继续发送数据，如主器件没有发送应答信号，24C02 停止传送数据且等待一个停止信号。主器件对 24C02 发送的最后一个字节数据不产生应答信号，但是发给 24C02 一个结束信号。

12.1.5 器件寻址、写操作和读操作

1. 器件寻址

24C02 作为 I²C 总线的从器件有一个从器件地址。24C02 是内部有 256 个字节的存储单元的 EEPROM，每个字节存储单元也有一个字节的 8 位地址。主器件发送一个起始信号启动发送过程，然后发送 24C02 的控制字节（从器件地址字节），其格式如图 12-5 所示，控制字节包括主器件所要寻址的从器件的地址，高 4 位固定为 1010，接下来的 3 位 A₂ A₁ A₀ 为主器件选中某一个 24C02 器件的地址，I²C 总线能够挂接 8 个 24C02，可单独被主器件寻址，控制字的最低位（R/ $\overline{W}$ ）为读写控制位，“1”表示对从器件进行读操作，“0”表示对从器件进行写操作。在主器件发送起始信号和从器件地址字节（控制字节）之后，24C02 接收从器件地址字节并与本身的从器件地址进行比较，从器件地址相符时发出一个应答信号（通过 SDA 线），然后 24C02 再根据读写控制位（R/ $\overline{W}$ ）的状态进行读或写操作。

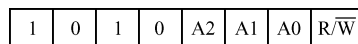


图 12-5 24C02 控制字节
（从器件地址字节）

24C02 内部存储单元的地址字节格式为 A₇ ~ A₀，A₇ 为 D₇ 位，A₀ 为 D₀ 位。

2. 写操作

（1）字节写操作

要能够进行字节写操作，24C02 的 WP 引脚应为低电平，或与 V_{ss} 连接，处于禁止写保护功能状态。

字节写操作示意图如图 12-6 所示，在字节写模式下，主器件发送起始位、4 位控制码 1010 紧随其后，接下来的 3 位是从器件地址 A₂、A₁、A₀，然后是 R/ $\overline{W}$ 位（写操作该位为低电平），如果 A₂A₁A₀ 符合 24C02 引脚 A₂A₁A₀ 连接的地址值，A₂A₁A₀ 就发出应答信号，即在第 9 个时钟周期产生一个确认位。

主器件在收到 24C02 发出的确认位的应答信号后，发送的第二个字节是待写入 24C02 内部存储单元的数据的地址字节，24C02 把地址字节锁存到内部的地址计数器，24C02 器件

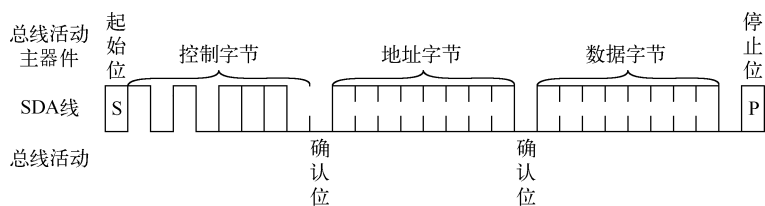


图 12-6 字节写操作示意图

再次发出确认信号 ACK。

主器件在接收到该确认信号后即发送数据字节，24C02 接收数据字节写入被选中地址的存储单元中，之后主器件发出停止条件信号，24C02 收到停止条件信号后，启动内部擦/写周期，在内部擦/写过程中 24C02 不再应答主器件的任何请求。

如果在 WP 引脚保持高电平时进行存储器写操作，从器件会确认命令，但不会启动写周期，也不会写入数据，而会立即接受新的命令。

(2) 写保护

通过把 WP 引脚连接到 V_{CC} ，允许用户设置 24C02 的写保护功能。把 WP 引脚连接到 V_{SS} 将禁止写保护功能。对于每个写命令，24C02 对 WP 引脚在停止位之前进行采样，在停止位之后改变 WP 引脚电平不会对写周期的执行产生任何影响。

3. 读操作

除了控制字节（从器件地址字节）的 R/W 位设置为“1”外，读操作与写操作基本相同。24C02 有三种基本的读操作：当前地址的读操作、随机读操作和连续读操作。

(1) 当前地址单元内容的读操作

当前地址单元内容读操作的示意图如图 12-7 所示，24C02 内置一个自动加“1”地址计数器，该计数器保留最后一次访问的地址。因此，如果先前已对地址“n”（n 为任意合法地址）的存储单元进行了读或写操作，则下一条读操作命令将自动对地址 $n + 1$ 的存储单元访问。

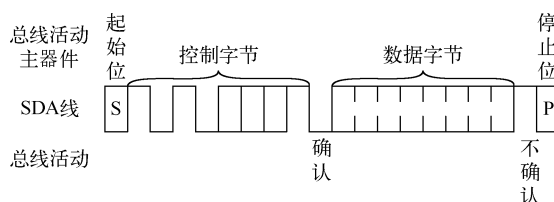


图 12-7 当前地址读操作示意图

24C02 接收到 R/W 位设置为“1”的控制字节后，24C02 发出确认信号，并发送当前地址单元的 8 位数据字节。主器件不会对数据传输做出确认，但会产生停止条件，24C02 即停止数据发送。

(2) 随机读操作

随机读操作允许主器件以随机方式访问 24C02 中的任意一个存储单元。随机读操作示意图如图 12-8 所示，发送的控制字节 R/W 设置为“0”，执行该指令前还需先设置 24C02 的内部待读出数据的存储单元的地址字节。主器件接收到 24C02 发出的字节地址的确认信

号再次产生起始条件，主器件再次发送控制字节，而该字节中 R/W 位设置为“1”，之后 24C02 发出确认信号，并发送被选中的地址单元的 8 位数据字节。主器件不会对数据传输做出确认，但会产生停止条件，24C02 即停止数据发送。在随机读取命令执行结束之后，24C02 内部地址计数器加 1 指向下一个存储单元的地址。

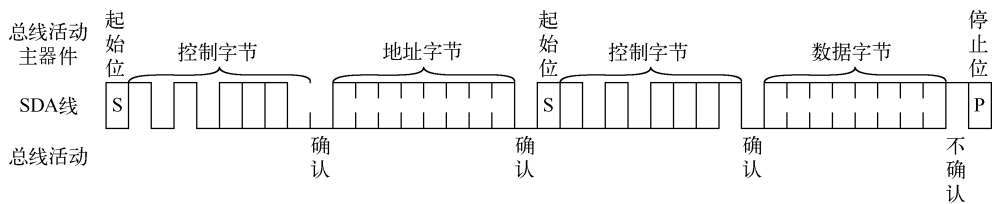


图 12-8 随机读操作示意图

(3) 连续读操作

连续读操作示意图如图 12-9 所示，连续读操作的启动过程和随机读操作相同，只是在 24C02 发送完第一个数据字节后，主器件发出确认信号。该确认信号指示 24C02 器件发送下一个连续地址的存储单元的数据字节。在 24C02 器件向主器件发送完最后一个数据字节后，主器件不再产生确认信号，而是产生停止条件。为了可以进行连续读操作，24C02 器件内置了一个地址指针，在每次操作完成后该指针加 1。地址指针达到最后一个地址字节 FFH 后，地址指针将翻转到地址 00H。

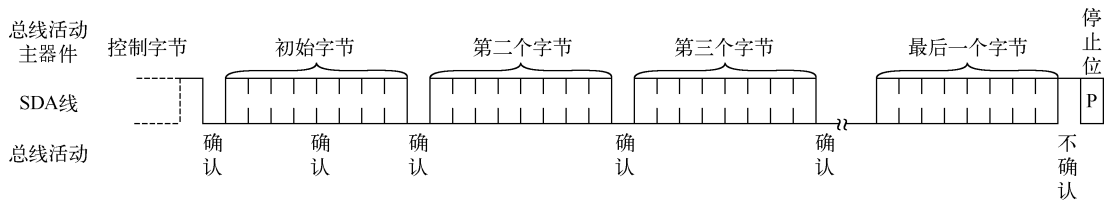


图 12-9 连续读操作示意图

12.1.6 24C02 芯片与 51 单片机的接口电路及编程

一块 24C02 芯片与 51 单片机的接口电路如图 12-10 所示。图 12-10 中，24C02 的三个地址引脚直接接地，串行时钟 SCL 引脚与 51 单片机的 P1.0 连接，双向串行数据 SDA 引脚与 P1.1 连接，SDA 和 SCL 都需要一个上拉电阻和 +5V 连接，WP 写保护引脚直接接地，允许 24C02 执行一般的读写操作。

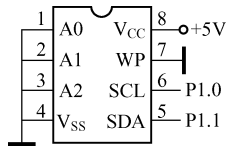


图 12-10 24C02 与 51 单片机的接口电路

24C02 内部有地址寄存器。读或写一个字节数据后，该地址寄存器自动加 1，实现对下一个存储单元的读/写。为降低总的写入时间，一次操作可连续读/写多达 8 个字节的数据。

例 12-1 将程序存储器首地址为 1000H 的 8 个数写到 24C02 中，24C02 的首址单元为 01H；然后将 24C02 中的首地址为 10H 的 8 个单元读到 51 单片机内部 RAM 中的首地址为 30H 的 8 个单元中，参考程序如下：

```

ORG      0000H
SCL      BIT    P1.0          ;定义 24C02 的串行时钟线
SDA      BIT    P1.1          ;定义 24C02 的串行数据线
LJMP     WSTART
ORG      0030H
;对从器件 24C02 写字节操作,参见图 12-3 的起始条件,图 12-6 的字节写操作
WSTART:  MOV     R2,#08H      ;一串数据有 8 个字节
        MOV     DPTR,#1000H  ;定义源数据的位置
        MOV     R1,#01H      ;定义存放到 24C02 中的数据的首址单元位置
LOOPW:   LCALL   STAR        ;调用启动,参见图 12-3
        MOV     R3,#10100000B ;写控制字节:1010 + 器件 3 位地址(000) +  $\overline{R/\overline{W}}$ (0)
        MOV     A,R3         ;送控制字节
        LCALL   SDATA        ;调用主器件移位输出(写)8 位数据操作子程序
        LCALL   ACK          ;调用确认子程序,参见图 12-6
        JC      LOOPW        ;判断从器件是否已确认,未确认返回再传输一次
        MOV     A,R1         ;送从器件 24C02 的首址单元为 01H 的地址字节
        LCALL   SDATA        ;调用主器件移位输出(写)8 位数据操作子程序
        LCALL   ACK          ;调用确认子程序
        JC      LOOPW        ;判断从器件是否已确认,未确认返回再传输一次
        MOV     A,#00H
        MOVC    A,@ A + DPTR ;提取 51 单片机程序存储器存储单元的数据传输
        LCALL   SDATA        ;调用写操作子程序
        LCALL   ACK          ;调用应答子程序
        JC      LOOPW        ;判断从器件是否已确认,未确认返回再传输一次
        INC     DPTR
        INC     R1           ;存放地址增一
        DJNZ    R2,LOOPW
        LCALL   STOP         ;调用停止子程序
;对从器件 24C02 连续读操作
;(R7) = 欲读数据存放地址指针,(R1) = 片内字节地址,(R2) = 连续读字节数
;(R3) = 10100001(读控制字节:1010 + 器件 3 位地址(000) +  $\overline{R/\overline{W}}$ (1))
RSTART:  MOV     R2,#08H      ;一串数据有 8 个字节,连续读字节数
        MOV     R7,#10H      ;定义从器件 24C02 源数据的首地址存储单元
        MOV     R1,#30H      ;定义读入数据存放在 51 单片机中的首地址存储单元
LOOPR:   LCALL   STAR        ;调用启动
        MOV     R3,#10100000B ;写控制字节:1010 + 器件 3 位地址(000) +  $\overline{R/\overline{W}}$ (0)
        MOV     A,R3         ;送控制字节
        LCALL   SDATA        ;调用写操作子程序
        LCALL   ACK          ;调用确认子程序,参见图 12-8
        JC      LOOPR        ;判断从器件是否已确认,未确认返回再传输一次

```

	MOV	A,R7	;送地址字节
	LCALL	SDATA	;调用写操作子程序
	LCALL	ACK	;调用从器件确认子程序
	JC	LOOPR	;C=1 转移,C=0 顺序执行,表示从器件已确认
	MOV	R3,#10100001B	;读控制字节:1010 + 器件 3 位地址(000) + $R/\overline{W}$ (1)
	MOV	A,R3	;送读控制字节
	LCALL	SDATA	;调用写操作子程序,参见图 12-8
	LCALL	ACK	;调用确认子程序
	JC	LOOPR	
MORE:	ACALL	SDATAR	;调用包括应答处理的接收单字节子程序,参见图 12-9
	MOV	@R1,A	;读入的数据存储在 51 单片机内部 RAM 中
	INC	R1	;51 单片机内部 RAM 存储单元地址 + 1
	DJNZ	R2,MORE	
	LCALL	STOP	;调用停止子程序
HERE:	AJMP	HERE	
SDATAR:	MOV	R4,#08H	;读单字节子程序
LOOPR:	SETB	SCL	
	ACALL	DELAY	
	MOV	C,SDA	
	RLC	A	
	CLR	SCL	
	ACALL	DELAY	
	DJNZ	R4,LOOPR	
	CJNE	R2,#01H,LOW	
	SETB	SDA	;若是最后一个字节不确认
	AJMP	SETOK	
LOW:	CLR	SDA	;否则主器件发出确认信号
SETOK:	ACALL	DELAY	
	SETB	SCL	
	ACALL	DELAY	
	CLR	SCL	
	ACALL	DELAY	
	SETB	SDA	;应答毕,SDA 置 1,P 口回到接收状态
	RET		
STAR:	SETB	SDA	;启动子程序,参见图 12-3,读/写操作都用该子程序
	ACALL	DELAY	;延时以满足传输速率要求
	SETB	SCL	
	ACALL	DELAY	;延时以满足传输速率要求
	CLR	SDA	
	ACALL	DELAY	;延时以满足传输速率要求
	CLR	SCL	

```

                RET
SDATA:  MOV    R0,#08H           ;写单字节操作子程序
LOOP0:  RLC     A
        MOV    SDA,C
        ACALL  DELAY             ;延时以满足传输速率要求
        SETB   SCL
        ACALL  DELAY             ;延时以满足传输速率要求
        CLR    SCL
        DJNZ   R0,LOOP0
        RET
ACK:    SETB    SDA              ;SDA 与 P1.1 连接,读 SDA 首先置 P1.1 为“1”
        ACALL  DELAY             ;延时以满足传输速率要求
        SETB   SCL
        ACALL  DELAY             ;延时以满足传输速率要求
        MOV    C,SDA             ;读从器件的确认信号
        CLR    SCL
        RET
STOP:   CLR     SDA              ;参见图 12-3
        ACALL  DELAY             ;延时以满足传输速率要求
        SETB   SCL
        ACALL  DELAY             ;延时以满足传输速率要求
        SETB   SDA
        ACALL  DELAY             ;延时以满足传输速率要求
        CLR    SCL
        RET
DELAY:  NOP
        NOP
        RET
        ORG    1000H
        DB 0A0H,10H,01H,02H,03H,04H,05H,06H
        END

```

程序中多处调用了 DELAY 子程序，这是为了满足 I²C 总线上数据传送速率的要求，只有当 SDA 数据线上的数据稳定下来之后才能进行读写（即 SCL 线发出正脉冲）。另外，在读最后一个字节数据时，置应答信号为“1”，表示读操作即将完成。

12.1.7 ZLG7290 接口电路及编程

ZLG7290 采用 I²C 串行接口，提供键盘中断信号，方便与微处理器接口。在显示方面，能够直接驱动 8 位共阴数码管或 64 只独立的 LED；控制扫描数码管的位数，控制任一数码管进行闪烁。可以实现数据译码和循环、移位、段寻址等控制。在键盘处理方面，可直接驱动 64 个按键，有 8 个功能键，可检测任一按键的连击次数，无须外接元件就可扩展驱动电

流和驱动电压。

采用 PDIP24 封装的 ZLG7290 引脚图如图 12-11 所示，其引脚功能参见表 12-1。

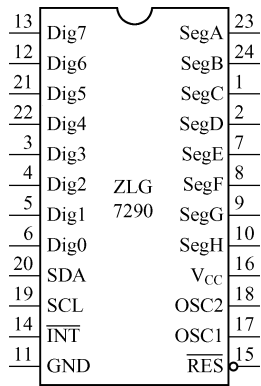


图 12-11 ZLG7290 引脚图

表 12-1 ZLG7290 引脚功能表

引脚	引脚名称	功能描述	引脚	引脚名称	功能描述
1	SC/SegC	数码管 c 段/键盘行信号 2	13	Dig7/KC7	数码管位选信号 7/键盘列信号 7
2	SD/SegD	数码管 d 段/键盘行信号 3	14	INT	键盘中断请求信号，低电平（下降沿）有效
3	Dig3/KC3	数码管位选信号 3/键盘列信号 3	15	RST	复位信号，低电平有效
4	Dig2/KC2	数码管位选信号 2/键盘列信号 2	16	V _{CC}	电源，+3.3 ~ 5.5 V
5	Dig1/KC1	数码管位选信号 1/键盘列信号 1	17	OSC1	晶振输入信号
6	Dig0/KC0	数码管位选信号 0/键盘列信号 0	18	OSC2	晶振输出信号
7	SE/SegE	数码管 e 段/键盘行信号 4	19	SCL	I ² C 总线时钟信号
8	SF/SegF	数码管 f 段/键盘行信号 5	20	SDA	I ² C 总线数据信号
9	SG/SegG	数码管 g 段/键盘行信号 6	21	Dig5/KC5	数码管位选信号 5/键盘列信号 5
10	DP/SegH	数码管 dp 段/键盘行信号 7	22	Dig4/KC4	数码管位选信号 4/键盘列信号 4
11	GND	接地	23	SA/SegA	数码管 a 段/键盘行信号 0
12	Dig6/KC6	数码管位选信号 6/键盘列信号 6	24	SB/SegB	数码管 b 段/键盘行信号

1. 功能描述

ZLG7290 系统功能框图如图 12-12 所示，系统功能框图即寄存器映像图。

(1) 键盘部分

ZLG7290 可采样 64 个按键或传感器，可检测每个按键的连击次数。其基本功能如下：

1) 键盘自动去抖动处理，无须编制键盘消抖动的程序。

2) 双键互锁处理。当有两个以上按键被同时按下时，ZLG7290 只采样优先级高的按键（优先顺序为 $S_1 > S_2 > \dots > S_{64}$ ，例如，同时按下 S_2 和 S_{18} 按键时，只能够采样到 S_2 按键被按下）。

3) 连击键处理。当某个按键按下时，输出一次键值后，如果该按键还未释放，该键值连续有效，就像连续按压该键一样，这种功能称为连击。ZLG7290 的连击次数计数器（RepeatCnt）可区别出单击（某些功能不允许连击，如开/关）或连击。判断连击次数可以检测

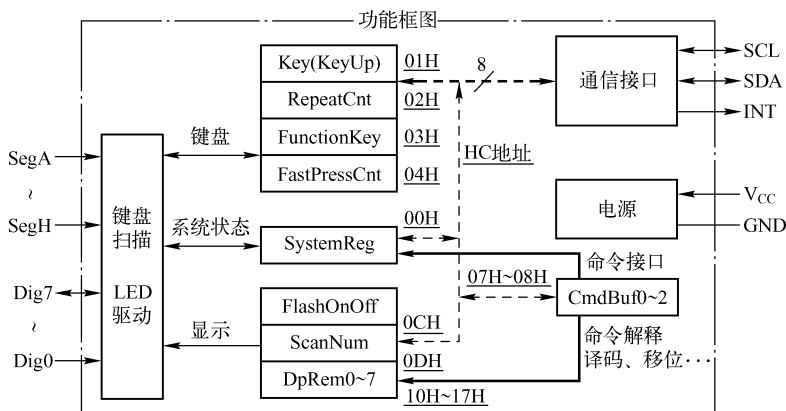


图 12-12 ZLG7290 系统功能框图

被按时间，以防止某些功能误操作（如连续按 5 s 进入参数设置状态）。

4) 功能键处理。如图 12-13 所示，ZLG7290 典型应用电路原理图中的按键 S57 ~ S64 是功能键，功能键能够实现 2 个以上按键同时按下来扩展按键数目或实现特殊功能。类似于 PC 上的〈Ctrl〉、〈Shift〉、〈Ctrl〉、〈Alt〉键，以及〈Ctrl〉+〈Shift〉键构成的组合键等。

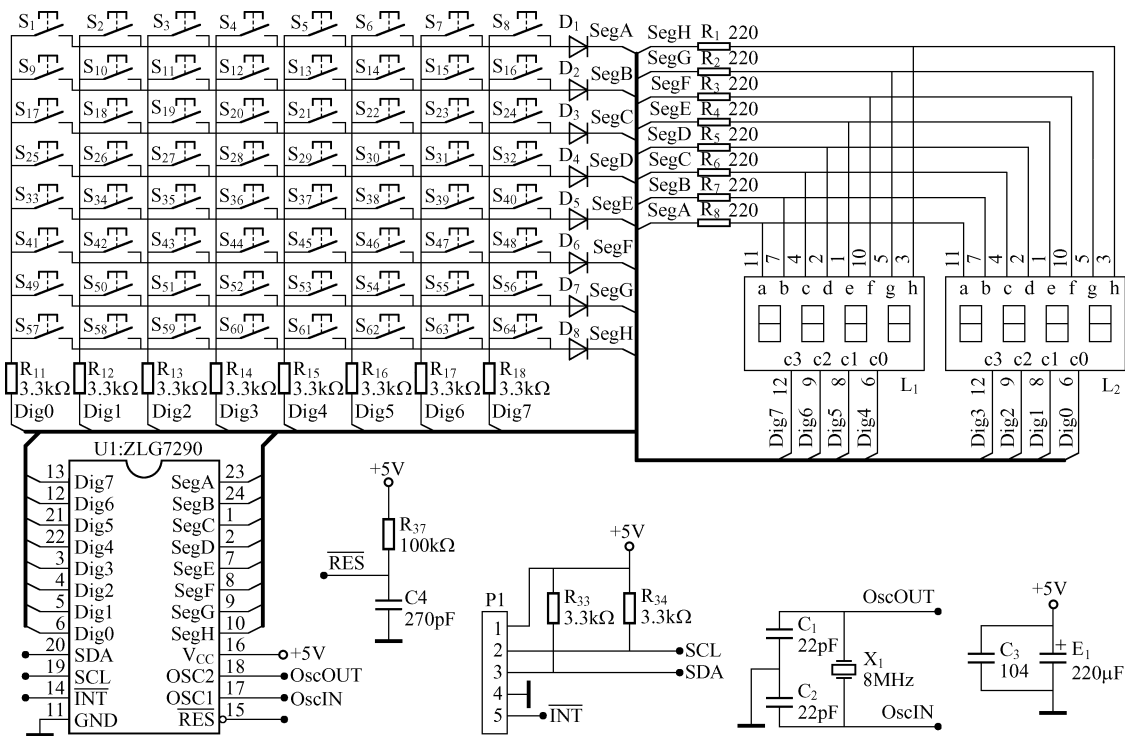


图 12-13 ZLG7290 典型应用电路原理图

(2) 显示部分

在每个显示刷新周期，ZLG7290 按照扫描位数寄存器（ScanNum）指定的显示位数 N，把显示缓存 DpRam0 ~ DpRamN 的内容按先后顺序送入 LED 驱动器实现动态显示，减少 N 值

可提高每位显示扫描时间的占空比,以提高 LED 亮度,显示缓存中的内容不受影响。修改闪烁控制寄存器 (FlashOnOff) 可改变闪烁频率和占空比 (亮和灭的时间)。

ZLG7290 提供两种控制方式:寄存器映像控制和命令解释控制。寄存器映像控制是指直接访问底层寄存器,实现基本控制功能,这些寄存器须字节操作。命令解释控制是指通过解释命令缓冲区 (CmdBuf0 CmdBuf1) 中的指令,间接访问底层寄存器实现扩展控制功能。如实现寄存器的位操作,对显示缓存循环、移位,对操作数译码等操作。

寄存器详解和指令详解请参阅相关手册。

(3) 通信接口

ZLG7290 作为 I²C 总线的从器件的接口传输速率可达 32kbit/s,其从地址 (Slave Address) 为 70H (01110000B)。

ZLG7290 内部可通过 I²C 总线访问的寄存器地址范围为 00H ~ 17H,任一寄存器都可按字节直接读写,也可以通过命令接口间接读写或按位读写。支持自动增址功能 (访问一寄存器后,寄存器子地址 (Sub Address) 自动加一) 和地址翻转功能 (访问最后一寄存器 (子地址 17H) 后,寄存器子地址翻转为 00H)。ZLG7290 的控制和状态查询全部是通过读/写寄存器实现的,用户只需像读写 24C02 内的单元一样,即可实现对 LG7290 的控制。

2. 典型应用电路

图 12-13 中, U1 就是 ZLG7290。为了使电源更加稳定,一般要在 V_{cc} 到 GND 之间接入 47 ~ 470 μ F 的电解电容 E₁。P1 是 ZLG7290 与微控制器的接口,按照 I²C 总线协议的要求,信号线 SCL 和 SDA 上必须要分别连接上拉电阻,其典型值是 10 k Ω 。晶振 X₁ 通常取值 8 MHz,调节电容 C₁ 和 C₂ 通常取值在 10pF 左右。复位信号是低电平有效,一般只需外接简单的 RC 复位电路,也可以通过直接拉低 RST 引脚的方法进行复位。

数码管必须是共阴结构的 LED 数码管, L₁ 和 L₂ 是 4 位联体式数码管,共同组成 8 位数码管显示电路。数码管在工作时要消耗较大的电流, R₁ ~ R₈ 是限流电阻,典型值是 220 Ω 。64 个按键中,前 56 个按键是普通按键 S₁ ~ S₅₆,最后 8 个 S₅₇ ~ S₆₄ 为功能键 F0 ~ F7。键盘电阻 R₁₁ ~ R₁₈ 的典型值是 3.3 k Ω 。数码管扫描线和键盘扫描线是共用的,所以二极管 D₁ ~ D₈ 是必需的,有了它们就可以防止按键干扰数码管显示的情况发生。

12.2 SPI 总线接口

12.2.1 SPI 总线概述及主要特点

SPI (Serial Peripheral Interface) 是 Motorola 公司推出的同步串行外设接口,允许单片机与多个厂家生产的带有标准 SPI 接口的外围设备直接连接,以串行方式交换信息。

图 12-14 为 SPI 串行总线外围器件扩展结构图。图中, 51 单片机作为主器件, 外围器件 1# ~ n# 作为从器件。对于每一个从器件使用 SPI 总线 4 条线: 串行时钟 SCK、主器件输入/从器件输出数据线 MISO、主器件输出/从器件输入数据线 MOSI 以及每一个从器件的一条选择线。

SPI 的典型应用是单主系统,一台主器件,从器件通常是外围接口器件,如存储器、I/O 接口、A - D、D - A、键盘、日历/时钟和显示驱动等。扩展多个外围器件时, SPI 总线无法

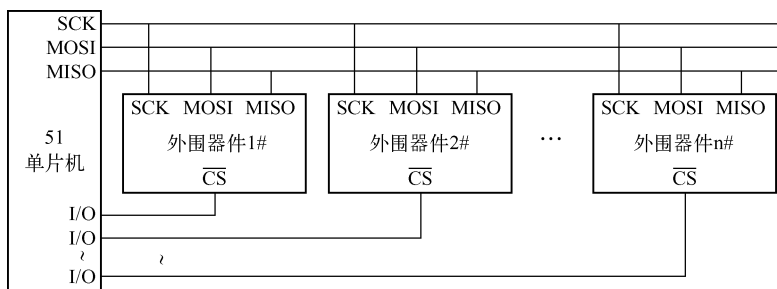


图 12-14 SPI 外围串行扩展结构图

通过数据线译码选择外围器件，故外围器件都有片选端。在扩展单个 SPI 器件时，外围器件的片选端 $\overline{CS}$ 可以接地或通过 I/O 口控制；在扩展多个 SPI 从器件时，单片机通过 I/O 口线来分时选通外围器件。如果 SPI 串行总线只扩展一个从器件，该从器件只作输入（如键盘）或只作输出（如显示器），则可省去一条数据输出（MISO）或一条数据输入（MOSI）线，构成二线系统， $\overline{CS}$ 可以接地。

SPI 总线系统与 I²C 总线选通比较，每个从器件多两根连接线，从器件扩展比较多时，主器件需要控制较多的从器件 $\overline{CS}$ 端，连线较多。但由于主器件单片机对从器件的选通是直接控制其 $\overline{CS}$ 端，省去 I²C 总线数据传输时的从器件地址字节（控制字节），而且 SPI 总线应用 MISO 和 MOSI 线，数据传送软件的编制比 I²C 总线简单得多。

SPI 总线的 8 位数据传输时序如图 12-15 所示。图 12-15 中，主器件单片机将选中的从器件的 $\overline{CS}$ 信号由“1”→“0”，发出 SCK 时钟信号，启动一次 8 位数据传输，SCK 时钟信号作为从器件的同步时钟，控制数据的输入和输出。传输格式是高位（MSB）在前，低位（LSB）在后。输出数据的变化以及输入数据时的采样，都取决于 SCK。但对不同外围芯片的从器件，可能是 SCK 的上升沿起作用，也可能是 SCK 的下降沿起作用。SPI 总线有较高的数据传输速度，最高可达 1.05Mbit/s。

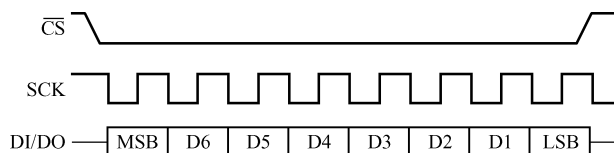


图 12-15 SPI 数据传送格式

Motorola 提供了一系列具有 SPI 接口的单片机和外围接口芯片，如存储器 MC2814、显示驱动器 MC14499 和 MC14489 等各种芯片。

目前已有许多增强型 51 单片机和其他机型的单片机都带有 SPI 接口。但对 51 单片机，由于不带 SPI 接口，要实现 SPI 接口，可采用软件控制 I/O 口来模拟 SPI 的接口时序。

12.2.2 12 位串行 A-D TLC2543 介绍

1. 功能特点与引脚

TLC2543 是美国 TI 公司的 12 位串行 SPI 接口的 A-D 转换器，使用开关电容逐次逼近

技术完成 A - D 转换过程，转换时间为 10 μ s。片内有 1 个 14 路模拟开关，用来选择 11 路模拟输入以及 3 路内部测试电压中的 1 路进行采样。由于是串行输入结构，能够节省 51 单片机 I/O 资源；且价格适中，分辨率较高，因此在仪器仪表中有较为广泛的应用。

TLC2543 具有如下特点：

- 1) 12 位分辨率 A - D 转换器。
- 2) 在工作温度范围内 10 μ s 转换时间。
- 3) 11 个模拟输入通道。
- 4) 3 路内置自测试方式。
- 5) 采样率为 66 kbit/s。
- 6) 线性误差 $\pm 1\text{LSB}_{\text{max}}$ 。
- 7) A - D 转换结束输出 EOC。
- 8) 具有单、双极性输出。
- 9) 可编程的 MSB 或 LSB 前导。
- 10) 可编程输出数据长度。

TLC2543 的引脚如图 12-16 所示，引脚说明见表 12-2。

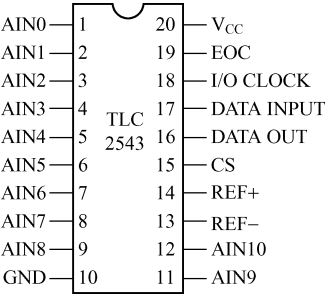


图 12-16 TLC2543 引脚图

表 12-2 TLC2543 引脚说明

引 脚 号	名 称	I/O	说 明
1 ~ 9, 11, 12	AIN0 ~ AIN10	I	模拟量输入端。模拟量输入通道由 8 位控制寄存器 D7 ~ D4 位选通
15	$\overline{\text{CS}}$	I	片选端
17	DATA INPUT	I	串行数据输入端
16	DATA OUT	O	A - D 转换结果的三态串行输出端。 $\overline{\text{CS}}$ 为高时处于高阻抗状态， $\overline{\text{CS}}$ 为低时处于激活状态
19	EOC	O	转换结束端。在最后的 I/O CLOCK 下降沿之后，EOC 从高电平变为低电平并保持到 A - D 转换完成和数据准备传输为止
10	GND		地。GND 是内部电路的地回路端
18	I/O CLOCK	I	输入/输出时钟端
14	REF +	I	正基准电压端。基准电压的正端（通常为 V_{CC} ）被加到 REF +，最大的输入电压范围由加于本端与 REF - 端的电压差决定
13	REF -	I	负基准电压端。基准电压的低端（通常为地）被加到 REF -
20	V_{CC}		电源

2. 工作原理

TLC2543 内部有一个 8 位控制寄存器、输出数据寄存器、A - D 转换器和模拟量采样器。8 位控制寄存器规定了 TLC2543 要转换的模拟量通道、转换后的输出数据长度和输出数据的格式，上电后 8 位控制寄存器初始状态被置为 00H。输出数据寄存器锁存 A - D 转换的结果，模拟量采样器实现对 8 位控制寄存器所规定的模拟量通道的输入模拟量的采样，A - D 转换器完成对采样值的 A - D 转换。

TLC2543 完成模拟量通道的选择、模拟量的采样、输出数据寄存器串行输出数据和 A - D 转换的时间称为一个工作周期，其中，输出数据寄存器串行输出数据的过程和模拟量通道的选择、模拟量的采样过程并行执行。

片选 $\overline{CS}$ 输入端为高电平时，I/O CLOCK、DATA INPUT 被禁止，DATA OUT 呈高阻状态，EOC 端输出为高电平。

一次工作周期的开始，TLC2543 的片选 $\overline{CS}$ 输入端必须从高电平变换到低电平，I/O CLOCK、DATA INPUT 使能，DATA OUT 脱离高阻状态，EOC 输出为高电平，8 位控制寄存器被置为 0。

I/O CLOCK 输入端依次输入 12 或 16 个时钟信号，随着时钟信号的输入，首先控制字从 DATA INPUT 一位一位地在时钟信号的上升沿时被送入 TLC2543（高位先送入）的 8 位控制寄存器，同时存放在输出数据寄存器中的上一个工作周期 A - D 转换数据的结果从 DATA OUT 一位一位地移出。第 4 个时钟信号后，TLC2543 已收到模拟通道号，此时 TLC2543 开始对选定的通道模拟量进行采样，并保持到第 12 或 16 个时钟的下降沿。在第 12 或 16 个时钟下降沿，EOC 变为低电平，开始对本次采样的模拟量进行 A - D 转换，转换时间约需 $10\mu s$ ，转换结束后 EOC 变为高电平，转换的数据锁存在输出数据寄存器中，待下一个工作周期输出。

（1）内部控制寄存器

8 位内部控制寄存器各位的基本功能如下：

1) 高 4 位 D7 ~ D4 决定模拟量输入通道号。0 通道至 10 通道的地址分别为 0000 ~ 1010H，该 4 位为 1011 ~ 1101 时，用于对 TLC2543 的自检，分别测试 ($V_{REF+} + V_{REF-}$)/2、 V_{REF-} 和 V_{REF+} 的值，当为 1110 时，TLC2543 进入休眠状态。

2) D3、D2 决定输出数据长度。00 和 10 表示输出数据长度为 12 位（标准位数，12 个时钟的工作周期），01 为 8 位（精度较低），11 为 16 位（16 个时钟的工作周期，低四位为零，便于 16 位串行数据传输）。

3) D1：决定输出数据是高位先送出，还是低位先送出。0：表示输出数据的最高位导前；1：最低位导前。

4) D0：单极性（无符号二进制）还是双极性（有符号的补码二进制）输出控制位。0：单极性；1：双极性。

（2）采样过程

TLC2543 的工作周期可以分为二个子周期：I/O 周期和转换周期 t_{conv} 。

1) I/O 周期完成对内部控制寄存器的置数、模拟量的采样，并同时完成数据寄存器的数据（上一个工作周期 A - D 转换的结果）从 DATA OUT 端的串行输出，工作在 I/O 周期时，EOC 输出高电平；I/O 周期又可以分为访问周期 A 和采样周期 B。

① 访问周期 A：4 个时钟脉冲，使得 TLC2543 接收到模拟量输入通道号，然后进入采样周期 B。

② 采样周期 B：3 个并行处理的周期，一是 5 ~ 8 个时钟脉冲，8 位控制寄存器的 D3 ~ D0 的数据输入；二是输出数据寄存器数据的串行输出；三是对选定的模拟量输入通道的输入信号进行采样。采样开始于 I/O 时钟的第 4 个下降沿。采样方式取决于内部控制寄存器的 D3、D2 的设置值，01：采样直到第 8 个 I/O CLOCK 下降沿；00 和 10：12 个时钟周期的采样；11：16 个时钟周期的采样。采样周期 B 结束，EOC 由输出高电平变换为输出低电平，进入转换周期。

2) 转换周期完成模拟量采样值的 A-D 转换, 是由 I/O 时钟同步的内部时钟来控制的。在转换周期开始时, EOC 输出变为低电平; 当转换完成时变为高电平, A-D 转换的数据结果由输出数据寄存器锁存。

上电后的第一个工作周期从输出数据寄存器串行输出的转换结果不准确, 应忽略。

TLC2543 可以用四种传输方法使 TLC2543 得到 12 位分辨率, 每次转换和数据串行输出可以使用 12 或 16 个时钟周期。

一个片选 ($\overline{CS}$) 脉冲要插到每次转换的开始处, 或者在 A-D 转换时序的开始处变化一次直到工作周期时序结束。

图 12-17 是使用 $\overline{CS}$, MSB 在前的 I/O 周期为 16 个时钟的一个工作周期的时序图, 图 12-18 是不使用 $\overline{CS}$, MSB 在前的 I/O 周期使用 16 个时钟的一个工作周期的时序图。

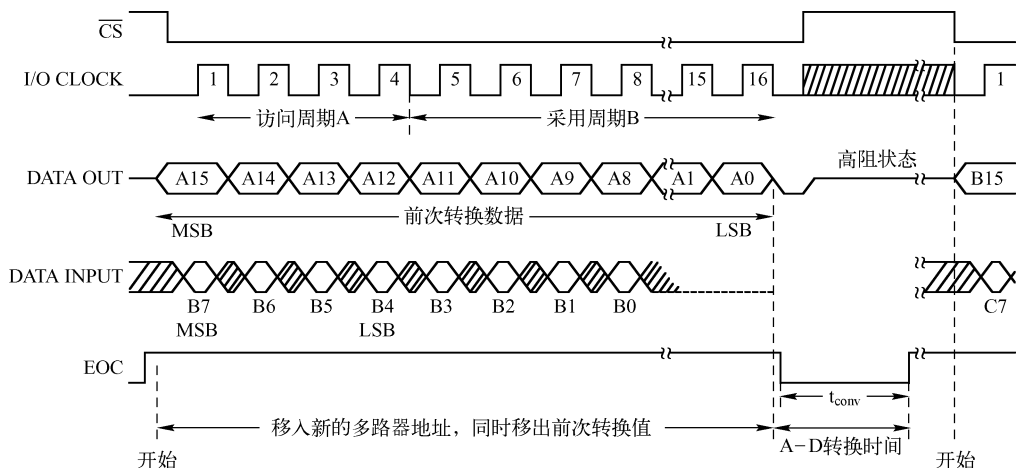


图 12-17 I/O 周期为 16 时钟的一个工作周期时序图 (使用 $\overline{CS}$, MSB 在前)

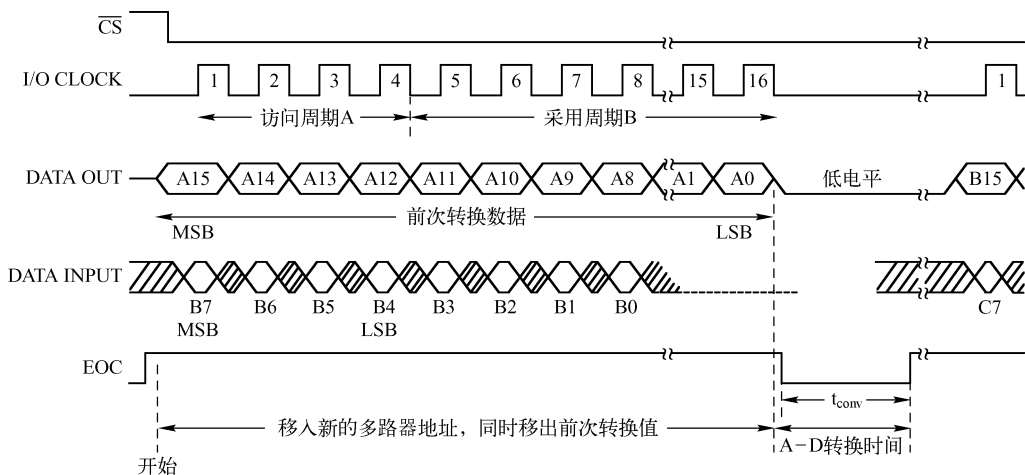


图 12-18 I/O 周期为 16 时钟的一个工作周期时序图 (不使用 $\overline{CS}$, MSB 在前)

12.2.3 TLC2543 与 51 单片机的接口设计

51 单片机与串行 A-D 转换器 TLC2543 的 SPI 接口原理图如图 12-19 所示。TLC2543 的 I/O CLOCK、DATA INPUT 和 $\overline{\text{CS}}$ 端由 51 单片机的 P1.0、P1.1 和 P1.3 来控制。转换结果的输出数据 (DATA OUT) 由 51 单片机的 P1.2 串行接收, 51 单片机将命令字通过 P1.1 输入到 TLC2543 的输入寄存器中。

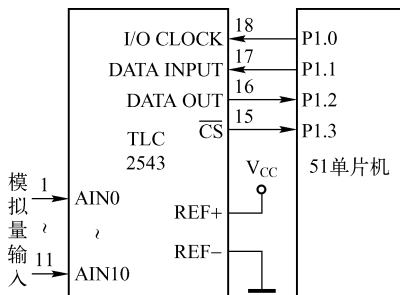


图 12-19 51 单片机与 TLC2543 的 SPI 接口

TLC2543 的数据采集程序结构可以参照第 10 章例 10-4 的编程方法, 应用定时器定时中断采样的方法设计程序结构。主要不同点在于定时中断服务程序。

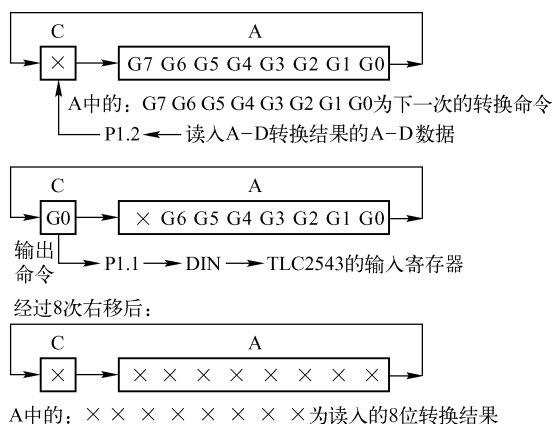
例 12-2 编写 51 单片机选择 TLC2543 的某一通道 (例如 AIN0 通道) 进行 1 次数据采集的子程序, A-D 转换结果共 12 位, 分两次读入。先读入 TLC2543 中的 8 位转换结果到 51 单片机中, 同时写入下一次转换的命令, 然后再读入 4 位的转换结果到 51 单片机中。

	ADCOMD	DATA 6FH	;定义命令存储单元
	ADOUTH	DATA 6EH	;定义存储转换结果高 4 位单元
	ADOUTL	DATA 6DH	;定义存储转换结果低 8 位单元
ADCV:	CLR	P1.0	;时钟引脚为低电平,P1.0 模拟 SCK 信号
	CLR	P1.3	;片选 $\overline{\text{CS}}$ 有效,选中 TLC2543,P1.3 模拟选择信号
	MOV	R2,#08H	;送出下一次 8 位转换命令和为读 8 位转换结果做准备
	MOV	A,ADCOMD	;下一次转换命令将 ADCOMD 单元中的内容送 A
LOOP1:	MOV	C,P1.2	;读入 1 位转换结果;P1.2 模拟 MOSI 信号
	RRC	A	;1 位转换结果带进位右移
	MOV	P1.1,C	;送出命令字节中的 1 位;P1.1 模拟 MOSI 信号
	SETB	P1.0	;产生 1 个时钟
	NOP		
	CLR	P1.0	
	NOP		
	DJNZ	R2,LOOP1	;是否完成 8 次转换结果读入和命令输出?未完则转移
	MOV	ADOUTL,A	;读 8 位转换结果存入 ADOUTL 单元
	MOV	A,#00H	;A 清 0
	MOV	R2,#04H	;为读入 4 位转换结果做准备
LOOP2:	MOV	C,P1.2	;读入高 4 位转换结果中的 1 位

RRC	A	; 带进位位循环右移
SETB	P1.0	; 产生 1 个时钟
NOP		
CLR	P1.0	
NOP		
DJNZ	R2, LOOP2	; 是否完成 4 次读入? 未完转移
MOV	ADOUTH, A	; 高 4 位转换结果存入 ADOUTH 高 4 位
SWAP	ADOUTH	; ADOUTH 中的高 4 位与低 4 位互换
SETB	P1.0	; 1 次数据采集过程结束, 时钟无效
RET		

执行上述程序中的循环程序段 8 次, 执行“RRC A”指令 8 次, 每次读入转换结果 1 位, 然后送出 ADCOMD 单元中的下一次转换命令字节“G7 G6 G5 G4 G3 G2 G1 G0”中的 1 位, 进入 TLC2543 的内部控制寄存器。

经 8 次右移后, 8 位 A-D 转换结果数据“××××××××”读入累加器 ACC 中, 上述的具体数据交换过程如图 12-20 所示。子程序中的 4 次循环, 只是读入转换结果的 4 位数据, 图中没有给出, 读者可自行画出 4 次移位的过程。



由本例见, 单片机与 TLC2543 接口十分简单, 只需用软件控制 4 条 I/O 引脚按规定时序对 TLC2543 进行访问即可。

12.3 单总线串行扩展

单总线（也称 1-Wire Bus）只有一条数据输入/输出线 DQ，总线上的所有器件都挂在 DQ 上，电源也通过这条信号线 DQ 供给。使用一条信号线的串行扩展技术，称为单总线技术。

单总线系统应用的各种器件是由 DALLAS 公司提供的专用芯片实现。每个芯片内部都有一个 64 位 ROM，每个芯片的 ROM 已被厂家用激光烧写编码，其中存有 16 位十进制编码序列号，它是器件的地址编号，确保它挂在总线上后，可唯一被确定。

除地址编码外，每个专用芯片内部还包含收发控制和电源存储电路，如图 12-21 所示。这些芯片的耗电量都很小（空闲时几微瓦，工作时几毫瓦），从单总线上馈送电能到能量存储大电容中就可以工作，故一般不需另加电源。

例 12-3 图 12-22 为一个由单总线构成的分布式温度监测系统。图 12-22 中有多个带有单总线接口的数字温度传感器 DS18B20，每个芯片都挂在单片机的 1 根 I/O 口线（即 DQ 线）上。对每个 DS18B20 通过总线 DQ 寻址。DS18B20 芯片的 DQ 都是漏极开路端口，须加上拉电阻。

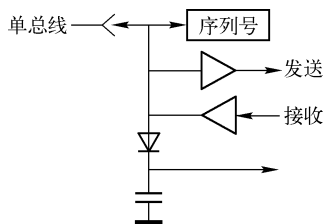


图 12-21 单总线芯片的内部结构示意图

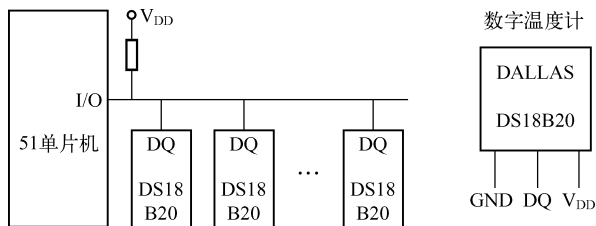


图 12-22 单总线构成的分布式温度监测系统

DS18B20 封装形式多样，其中的一种封装形式如图 12-22 所示。在该单总线数字温度传感器系列中还有 DS1820、DS18S20、DS1822 等其他型号，工作原理与特性基本相同。主要特点有：

- 1) 体积小、结构简单、使用方便。
- 2) 每芯片都有唯一的 64 位光刻 ROM 编码，家族码为 28H。
- 3) 温度测量范围为 $-55 \sim +125^{\circ}\text{C}$ ，在 $-10 \sim +85^{\circ}\text{C}$ 范围内，测量精度可达 $\pm 0.5^{\circ}\text{C}$ 。
- 4) 分辨率为可编程的 9 ~ 12 位（其中包括 1 位符号位），对应的温度变化量分别为 0.5°C 、 0.25°C 、 0.125°C 和 0.0625°C 。
- 5) 转换时间与分辨率有关。当设定为 9 位时，转换时间 93.75 ms；设定为 10 位时，转换时间为 187.5 ms；当设定为 11 位时，转换时间 375 ms；当设定为 12 位时，转换时间 750 ms。

6) 片内含有 SRAM、EEPROM，单片机写入 EEPROM 的报警的上下限温度值以及对 DS18B20 的设置，在芯片掉电的情况下不丢失。

功能命令包括两类：1 条启动温度转换命令（44H），5 条读/写 SRAM 和 EEPROM 命令。

图 12-22 所示电路中如果再扩展几位（根据需要）LED 数码管显示器，即可构成简易的数字温度计系统。也可在图 12-22 的电路基础上，自行扩展功能设计。

1-Wire 总线传输的是数字信号，数据传输均采用 CRC 码校验。DALLAS 公司为单总线的寻址及数据的传送制定了总线协议，具体内容读者可查阅相关资料。

1-Wire 协议的不足在于传输速率稍慢，故 1-Wire 总线协议特别适用于测控点多、分布面广、种类复杂，而又需集中监控、统一管理的应用场合。

习题

12-1 单片机的串行扩展技术与并行扩展技术相比具有哪些显著的优点？

12-2 I²C 总线为什么需要设置起始信号和终止信号？起始信号和终止信号是如何定义的？

12-3 I²C 总线的数据传输方向如何控制？

12-4 I²C 总线中的从器件有从器件地址和从器件内部器件的地址，单片机如何实现对 I²C 总线中的从器件和从器件内部器件的寻址？

12-5 I²C 总线在数据传送时，应答是如何进行的？

12-6 24C02 有三种基本的读操作：当前地址的读操作、随机读操作和连续读操作。简述这三种基本读操作的异同点和特点，哪种读操作速度最慢？

12-7 上网查阅作为 I²C 总线从器件的 ZLG7290B 应用程序，说明应用 51 单片机编制 ZLG7290B 应用程序的基本思路和程序结构。

12-8 什么是 SPI 总线接口？SPI 总线接口由哪几条线组成？SPI 总线电路结构与 I²C 总线电路结构比较，有哪些主要特点？

12-9 51 单片机如何实现对 SPI 总线中的从器件的寻址？

12-10 上网查阅作为 SPI 总线从器件的 MC2814 存储器的资料，说明 SPI 总线如何实现对从器件内部器件的寻址？

12-11 绘制作为 SPI 总线从器件的 3 块串行 A-D 转换器 TLC2543 与 51 单片机的接口电路。

12-12 如果串行 A-D 转换器 TLC2543 设置为 12 位的 A-D 转换器，如何设置内部控制寄存器 D3、D2 位，12 个时钟周期的采样方式好还是 16 个时钟周期的采样方式好？为什么？

12-13 I²C 总线和 SPI 总线的数据传输速度各为多少？在实际选用中如何考虑？

12-14 上网查询片内集成 I²C 总线控制部件的增强型 8051 内核单片机，列出具有这种功能的增强型单片机型号及应用特点。

12-15 上网查询片内集成 SPI 总线控制部件的增强型 8051 内核单片机，列出具有这种功能的增强型单片机型号及应用特点。

第 13 章 51 单片机的 C51 程序设计

13.1 C51 编程概述

C 语言是一种通用的计算机程序设计语言，它既可以用来编写计算机的系统程序，也可以用来编写一般的应用程序。由于汇编语言程序的可读性和可移植性都存在难度，采用汇编语言编写单片机应用程序的周期长，调试和排错的难度大。为了提高编写单片机应用程序的效率，改善程序的可读性和可移植性，采用高级语言无疑是一种最好的选择。目前，各类系列的单片机基本上都采用了 C 语言为主要开发语言，例如 51 单片机的 C51，MSP430 的 C430 等。C51 语言既具有一般高级语言的特点，又能直接对单片机的硬件进行操作，表达和运算能力比较强，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 C51 语言来解决。

C51 与标准 C 语言的主要不同如下：

1) C51 语言的学习与应用必须掌握 51 单片机的基本原理和组成结构、外部扩展芯片的编址和电路设计，硬件和软件设计能力都需要具备，而标准 C 语言对硬件的关联度小。

2) 标准 C 语言定义的变量不需要直接与计算机硬件的地址发生关联。C51 应用单片机内部的寄存器组，特殊功能寄存器，包括特殊功能寄存器可位寻址的位单元，以及扩展的 I/O 芯片端口和各种功能部件，其定义的变量必须定位到这些硬件每个寄存器和端口的地址。

3) 针对单片机位存储空间和位操作指令，C51 增加了位数据类型的定义和位处理功能。

4) 头文件的不同。由于 51 系列单片机生产厂家的不同，以及增强型单片机内部资源的不同，其头文件也不同，头文件集中体现了各类系列单片机的不同资源及功能。例如，C51 使用 8051 系列单片机内部资源的头文件：reg51.h；扩展资源使用：absacc.h。

5) 针对单片机所特有的存储空间结构，C51 增加了各种存储空间变量的定义。

6) 对于单片机的中断系统，寄存器组的选择，C51 专门增加了处理的语句。

7) Keil C51 与标准 C 语言的库函数有较大的不同。Keil C51 的部分库函数是结合单片机硬件特点而开发的，其构成和应用方法有很大的不同。例如，库函数 printf 和 scanf，标准 C 语言中是用于屏幕打印和接收数据；Keil C51 中主要用于串行口数据的接收与发送。

8) 针对 51 单片机通常的存储资源最多为 64KB 的情况，C51 不允许过多的程序嵌套，对于标准 C 语言的函数递归调用的功能必须用 reentrant 进行声明方可使用，否则不被支持。

C51 语言作为一种非常方便的编程语言已得到广泛的应用，C51 程序本身虽然依赖于单片机的硬件系统，但只需要针对不同型号的单片机选用与之对应的头文件，修改与特定的单片机应用系统相关变量的存储区域和地址，就可以在不同的单片机应用系统中移植使用。

13.1.1 Keil C51 开发工具

德国 Keil Software 公司多年来致力于单片机 C 语言编译器的研究，该公司开发的 Keil C51 是一种专为 51 单片机设计的高效率 C 语言编译器，符合 ANSI 标准，产生的程序代码运行速度快，所需要的存储器空间小，可与汇编语言相媲美。

Keil 公司推出的 C51 编译器，为 51 单片机软件开发提供了全新的 C 语言环境，同时保留了汇编代码高效、快速的特点。C51 已被完全集成到一个功能强大的全新的集成开发环境 μ Vision2 中，其中包括项目管理器、Cx51 编译器、Ax51 宏汇编器、BL51/Lx51 链接定位器、RTX51 实时操作系统、Simulator 软件模拟器和 Monitor51 硬件目标调试器，所有这些功能均可在 μ Vision2 提供的单一而灵活的开发环境中极为简单地进行操作。开发工具套件可用于编译 C 源程序、汇编源程序，链接和定位目标文件和库，创建 Hex 文件以及调试目标程序。

- μ Vision2 for windows：一个集成开发环境。它将项目管理、源代码编辑和程序调试等组合在一个功能强大的环境中。
- Cx51 国际标准优化 C 交叉编译器：从 C 源程序产生可重定位的目标模块。
- Ax51 宏汇编器：从 8051 汇编源代码产生可重定位的目标模块。
- BL51 链接/定位器：组合由 Cx51 和 Ax51 产生的可重定位的目标模块，生成绝对目标模块。
- LIB51 库管理器：从目标模块生成链接器可以使用的库文件。
- OH51 目标文件至 Hex 格式的转换器：从绝对目标模块生成 Intel Hex 文件。
- RTX-51 实时操作系统：简化了复杂的实时应用软件项目的设计。

13.1.2 Keil C51 程序开发过程

采用 Keil C51 开发 51 单片机应用程序一般需要以下步骤：

- 1) 在 μ Vision2 集成开发环境中创建一个新项目 (Project)，为该项目选择合适的单片机 CPU 器件，并配置工具软件的设置。
- 2) 利用 μ Vision2 的文件编辑器编写 C 语言（或汇编语言）源程序文件，并将文件添加到项目中。一个项目可以包含多个文件，除源文件外还可以有库文件或文本说明文件。
- 3) 通过 μ Vision2 的各种选项，配置 Cx51 编译器、Ax51 宏汇编器、BL51/Lx51 链接定位器以及 Debug 调试器的功能。
- 4) 利用 μ Vision2 的构造 (Build) 功能对项目中的源程序文件进行编译连接，生产绝对目标代码和可选的 Hex 文件，如果出现编译连接错误则返回第②步，修改源程序中的错误后重新构造 (Build) 整个项目。
- 5) 将没有错误的绝对目标代码装入 μ Vision2 调试器进行仿真调试，调试成功后将 Hex 文件写入单片机应用系统的程序存储器中。

13.1.3 C51 程序结构

C51 的程序结构与标准 C 语言相同。一个 C51 程序就是函数的集合，在这个集合中，有且只有一个名为 main 的函数（主函数）。主函数是程序的入口，主函数中的所有语句执行完毕，则总的程序执行结束。

C51 函数定义的一般格式如下：

```
类型 函数名(参数表)
参数说明;
{
    数据说明部分;
    执行语句部分;
}
```

一个函数在程序中可以有三种形态：函数定义、函数调用和函数说明。函数定义和函数调用不分先后，但若调用在定义之前，那么在调用前必须先进行函数说明。函数说明是一个没有函数体的函数定义，而函数调用则要求有函数名和实参数表。

C51 中函数分为两大类，一类是库函数，另一类是用户定义函数，这与标准 C 语言是一致的。库函数是 C51 在库文件中已定义的函数，其函数说明在相关的头文件中。对于这类函数，用户在编程时只要用 include 预处理指令将头文件包含在用户文件中，直接调用即可。用户函数是用户自己定义和调用的一类函数，从某种意义上来看，C51 编程实际上就是对一系列用户函数的定义和调用。

13.2 C51 的数据类型及存储类型

13.2.1 C51 的基本数据类型

任何程序设计都离不开对数据的处理，数据在计算机内存中的存放情况由数据结构决定。C51 语言的数据结构是以数据类型出现的，数据类型可分为基本数据类型和复杂数据类型，而复杂数据类型由基本数据类型构造而成。对于 C51 编译器来说，short 类型与 int 类型相同，double 类型与 float 类型相同。

C51 语言中的数据类型有：

(1) 位类型 (bit)

位类型是 Keil C51 编译器的一种扩充数据类型，利用它可以定义一个位变量，但不能定义位指针，也不能定义位数组。位类型数据的值是 1 (true) 或 0 (false)。与 51 单片机硬件特性操作有关的位变量必须定位在单片机片内数据存储区 (RAM) 的可位寻址空间中，51 单片机内部可位寻址的特殊功能寄存器的位变量已在头文件 reg51.h 中定义。

(2) 字符型 (char)

字符型有 unsigned char (无符号字符型) 和 signed char (有符号字符型) 之分，默认值为 signed char。它们的长度均为一个字节，用于存放一个单字节的数据。对于 signed char 类型数据，数据的最高位表示该数据的符号，“1”表示负数，“0”表示正数。对于一个 unsigned char 类型数据，数据的取值范围为 0 ~ 255；对于一个 signed char 类型数据，数据的取值范围为 -128 ~ 127。

(3) 整型 (int)

整型有 unsigned int (无符号整型) 和 signed int (有符号整型) 之分，默认为 signed int。它们的长度均为两个字节，用于存放一个双字节数据。signed int 类型数据的最高位是该数

据的符号位，“1”表示负数，“0”表示正数。对于一个 unsigned int 类型数据，数据的取值范围为 0 ~ 65535；对于一个 signed int 类型数据，数据的取值范围为 -32768 ~ 32767。

(4) 长整型 (long)

长整型有 unsigned long (无符号长型) 和 signed long (有符号长型) 之分，默认为 signed long。它们的长度均为 4 个字节。signed long 类型数据的最高位是该数据的符号位，“1”表示负数，“0”表示正数。对于一个 unsigned long 类型数据，其取值范围为 0 ~ 4294967295；对于一个 signed long 类型数据，其取值范围为 -2147483648 ~ 2147483647。

(5) 浮点型 (float)

符合 IEEE-754 标准的单精度浮点型数据，在十进制数中具有 7 位有效数字。float 类型数据占用 4 个字节。

(6) 指针类型 (*)

指针类型数据不同于其他数据类型，它本身是一个变量，但这个变量中存放的不是普通数据，而是指向另一数据的地址。指针变量要占用一定的内存单元，在 C51 中指针变量的长度一般为 1 ~ 3 个字节。指针变量也具有数据类型，其表示方法是在指针符号“*”的前面加数据类型符号，如 unsigned char * pt 表示 pt 是一个无符号字符型的指针变量。指针变量的数据类型表示该指针所指向地址中数据的类型。使用指针型变量可以方便地对 51 单片机各部分的物理地址单元直接进行操作。

(7) 空类型

在调用函数时，通常应向调用者返回一个函数值。这个返回的函数值是具有一定数据类型的，应在函数定义及函数声明中予以说明。但也有一类函数，调用后并不需要向调用者返回函数值，这种函数可以定义为“空类型”，类型说明符为 void。

(8) 特殊功能寄存器 (sfr)

这也是 Keil C51 编译器的一种扩充数据类型，利用它可以定义 51 单片机的所有内部 8 位特殊功能寄存器。sfr 型数据占用一个内存单元，其取值范围是 0 ~ 255。

(9) 16 位特殊功能寄存器 (sfr16)

16 位特殊功能寄存器 DPTR 占用两个字节，取值范围是 0 ~ 65535，利用它可以定义 51 单片机内部 16 位特殊功能寄存器 DPTR。

13.2.2 C51 的数据存储类型和存储模式

在讨论 Keil C51 的数据类型的时候，必须同时提及它的存储类型以及它与 51 单片机存储器结构的关系，因为 Keil C51 是面向 51 单片机及其硬件控制系统的开发工具。它所定义的任何数据类型必须以一定存储类型的方式定位在单片机的某一存储区中，否则就没有任何实际意义。

Keil C51 编译器完全支持 51 单片机的硬件结构，可以访问 51 硬件系统的所有部分。该编译器通过将变量、常量定义成不同的存储类型 (data、bdata、idata、pdata、xdata 和 code) 的方法，将它们定位在单片机系统的不同存储区中。

存储类型与 51 单片机实际存储空间的对对应关系见表 13-1，并参见图 2-10。

存储模式决定了变量的默认存储类型、参数传递区和无明确存储类型说明的变量的存储类型。定义变量时如果省略存储类型，则按照编译时使用的存储模式 SMALL、COMPACT 或

表 13-1 C51 存储类型与 51 单片机存储空间的对对应关系

存 储 类 型	说 明
data	直接寻址的片内数据存储区，访问速度最快（00H ~ 7FH）
bdata	可位寻址的片内数据存储区，允许位与字节混合访问（字节单元：20H ~ 2FH；位单元：00H ~ 7FH）
idata	间接寻址的片内数据存储区，可访问片内全部 RAM 地址空间（51：00H ~ 7FH；52：00H ~ FFH）
pdata	分页寻址的片外数据存储区（256 B），由 MOVX @ R0 访问的存储器空间
xdata	片外数据存储区（64 KB），由 MOVX @ DPTR 访问的存储器空间
code	程序存储区（64 KB），由 MOVC @ A + DPTR 访问的存储器空间

LARGE 来规定默认存储类型，确定变量的存储空间，函数中不能采用寄存器传递的参数变量和过程变量也保存在默认的存储空间。在 SMALL 模式下，参数传递是在片内数据存储区中完成的。LARGE 和 COMPACT 模式允许参数在外部存储器中传递。同时，Keil C51 也支持混合模式，例如在 LARGE 模式下，生成的程序可以将一些函数放入 SMALL 模式中，从而加快执行速度。Keil C51 编译器的三种存储模式（默认的存储类型）对变量的影响如下：

（1）SMALL 模式

变量被定义在 51 单片机的片内数据存储区中，对这种变量的访问速度最快。另外，所有的对象，包括堆栈，都必须位于片内数据存储器中，而堆栈的长度很重要，实际栈长取决于不同函数的嵌套深度。

（2）COMPACT 模式

变量被定义在分页寻址的片外数据存储器中，每一页片外数据存储器的长度为 256 B。这时对变量的访问是通过寄存器间接寻址（MOVX @ Ri）进行的，堆栈位于单片机片内数据存储器中。采用这种编译模式时，变量的高 8 位地址由 P2 口确定，低 8 位地址由 R0 或 R1 的内容确定。采用这种模式时，必须适当修改 STARTUP.A51 文件中的参数：PDASTART 和 PDATALEN；在用 BL51 进行链接时还必须采用链接命令“PDATA”对 P2 口地址进行定位，才能确保 P2 口是所需要的高 8 位地址。

（3）LARGE 模式

变量定义在片外数据存储器中，使用数据指针 DPTR 来进行寻址（MOVX @ DPTR）。用此数据指针进行访问效率较低，尤其是对两个或多个字节的变量，这种数据类型的访问机制直接影响代码的长度。需要指出的是，变量的存储类型与存储器类型是完全无关的。

13.2.3 单片机特殊功能寄存器（SFR）及其 C51 定义

为了能直接访问 51 单片机内部的特殊功能寄存器（SFR），Keil C51 编译器扩充了关键字 sfr 和 sfr16，利用这些扩充关键字可以在 C 语言源程序中直接对 51 单片机的特殊功能寄存器进行定义。定义方法如下：

sfr 特殊功能寄存器名 = 地址常数；例如：

```
sfr TMOD = 0x89;           /* 定时/计数器方式控制寄存器地址 89H */
sfr P0 = 0x80;             /* 定义 I/O 口 P0, P0 的地址为 80H */
```

注意：sfr 后面必须跟一个特殊功能寄存器名，“=”后面的地址必须是常数，不允许带

有运算符的表达式，这个常数值范围必须在特殊功能寄存器地址范围内，位于 0x80 ~ 0xFF 之间。

在 51 单片机应用系统中经常需要访问特殊功能寄存器中的某些位，Keil C51 编译器为此提供了一个扩充的 sbit 关键字，利用它定义可位寻址对象。定义方法有如下三种：

(1) sbit 位变量 = 位地址

这种方法将位的绝对地址赋给位变量，位地址必须位于 0x80 ~ 0xFF 之间。例如：

```
sbit PSW = 0xD0;           /* 定义 PSW 寄存器的地址为 0xD0 */
sbit CY = 0xD7;
```

(2) sbit 位变量名 = 特殊功能寄存器名 ^ 位位置

当可寻址位位于特殊功能寄存器中时可采用这种方法，“位位置”是一个 0 ~ 7 之间的常数。

(3) sbit 位变量名 = 字节地址 ^ 位位置

这种方法以一个常数（字节地址）作为基地址，该常数必须在 0x80 ~ 0xFF 之间。“位位置”是一个 0 ~ 7 之间的常数。例如：

```
sbit OV = 0xD0^2;
sbit CY = 0xD0^7;
```

除了通常的 C 数据类型外，C51 编译器支持 bit 数据类型。需要注意的是，sbit 是一个独立的关键字，不要将它与关键字 bit 相混淆。bit 用来定义一个普通的位变量。一个函数可以有 bit 类型的参数，函数的返回值也可以是 bit 类型。如果在函数中禁止使用中断（#pragma disable）或函数中包含明确的寄存器组定义（using n），则该函数不能返回位型值，否则编译时会产生编译错误。另外，不能定义位指针，也不能定义位数组。

上面介绍了变量及其定义方法，这在编写 C 语言程序时是十分重要的。从变量的作用范围来看，还有全局变量和局部变量之分。全局变量是指在程序开始处或各个功能函数的外面定义的变量，在程序开始处定义的全局变量对于整个程序都有效，可供程序中所有函数共同使用；而在各功能函数外面定义的全局变量只对定义处往后的各个函数有效，只有从定义处往后的那些功能函数才可以使用该变量，定义处前面的函数则不能使用它。

本章节主要介绍与 51 单片机直接相关的 C 语言程序部分，至于 C 语言程序设计的基本语法不是本书的重点，在此略过，如需要，请参考有关 C 语言程序设计基础方面的书籍。

13.3 C51 函数

函数是 C 语言中的一个基本模块，实际上，一个 C 语言程序就是由若干个模块化的函数构成的。C 语言程序总是由主函数 main() 开始，main() 函数是一个程序流的特殊函数，它是程序的起点。当设计较大的程序时，一般应将程序分解成若干个子程序模块，每个模块完成一种特定的功能。在 C 语言中，子程序是用函数来实现的。由于采用模块化的结构，C 语言易于实现结构化的程序设计，使程序的层次清晰，便于编写、阅读和调试。

13.3.1 函数的分类与定义

从用户的角度来看有两种函数：标准库函数和用户自定义函数。标准库函数是由 Keil

C51 编译器提供的，不需要用户进行定义，可以直接调用。用户自定义函数是根据用户需要编写的能实现特定功能的函数，它必须先进行定义之后才能调用。函数定义的一般形式是：

```
函数类型  函数名(形式参数表)
形式参数说明
{
    局部变量定义;
    函数体语句;
}
```

其中，“函数类型”说明了自定义函数返回值的类型。

“函数名”是由用户定义的标识符。

“形式参数表”中列出的是在主调函数和被调函数之间传递数据的形式参数，形式参数的类型必须加以说明。如果定义的是无参函数，可以没有形式参数，但圆括号不能省略。

如果定义函数时，只给出一对{}而不给出其局部变量和函数体语句，则该函数为“空函数”，这种函数也是合法的。

(1) 函数的参数

函数的参数分为形参和实参两种。形参和实参的功能是数据传递。发生函数调用时，主调函数把实参的值传递给被调函数的形参，从而实现主调函数向被调函数的数据传递。

(2) 函数的值

函数的值是指函数被调用后，执行函数体程序所取得的并返回给主调函数的值，对于函数的值（或函数的返回值）有以下说明：

- 1) 函数的值只能通过 return 语句返回给主调函数。
- 2) 函数值的类型和函数定义中函数的类型应保持一致。如果不一致，则以函数类型为准，自动进行转换。
- 3) 不返回函数值的函数，应明确定义为“空类型”，类型说明符为“void”。

13.3.2 函数的调用

C 语言程序中函数是可以相互调用的。所谓函数调用就是在一个函数体内引用另外一个已经定义的函数，前者称为主调函数，后者称为被调函数。在 C 语言中可以采用三种方法完成函数的调用。

(1) 函数语句

在主调函数中将函数调用作为一条语句。如：fun1(a,i)。

(2) 函数表达式

在主调函数中将函数调用作为一个运算对象直接出现在表达式中，这种表达式称为函数表达式。如：c = fun1(a,i) + fun2(b,j)。

(3) 函数参数

在主调函数中将函数调用作为另一个函数调用的实际参数。如：X = fun1(fun1(a,b),k)。

1. 函数的嵌套调用

在 C 语言中，函数的定义都是相互独立的，即定义函数时，一个函数的内部不能包含

另一个函数。尽管 C 语言中函数不能嵌套定义，但允许嵌套调用，也就是说，在调用一个函数的过程中，允许调用另一个函数。

有些 C 语言编译器对函数嵌套的深度有一定的限制，对于 51 单片机来说，对函数嵌套调用层次的限制是由于 51 单片机片内 RAM 中堆栈空间不够所致。

2. 函数的递归调用和再入函数

在调用一个函数的过程中，又直接或间接地调用该函数本身。这种情况称为函数的递归调用。

C 语言的强大优势之一就在于它允许函数的递归调用，通常用于问题的求解，可以把一个解法逐次地用于问题的子集表示的场合。

Keil C51 编译器采用一个扩展关键字 `reentrant`，作为定义函数时的选项，只要在函数名后面加上关键字 `reentrant`，就可以将该函数定义为再入函数。再入函数的一般形式如下：

函数类型函数名(形式参数表) [`reentrant`]

再入函数可被递归调用，无论何时，包括中断服务函数在内的任何函数都可调用再入函数。和非再入函数的参数传递和局部变量的存储区分配方法不同，C51 编译器为再入函数产生一个模拟栈，通过模拟栈来完成参数传递和存放局部变量。模拟栈所在的存储空间根据再入函数的存储类型的不同，可以是 DATA、PDATA 或 XDATA 存储器空间。当程序中包含有多种存储器模式的再入函数时，C51 编译器为每种模式单独建立一个模拟栈。

13.3.3 中断服务函数和寄存器组定义

为了在 C 语言源程序中直接编写 51 单片机的中断服务函数程序，Keil C51 编译器对函数的定义进行了扩展，增加了一个扩展关键字 `interrupt`，它是函数定义时的一个选项，加上这个选项之后就可以将一个函数定义成中断服务函数。定义中断服务函数的一般形式为

函数类型函数名(形式参数表) `interrupt n` [`using n`]

关键字 `interrupt` 后面的 `n` 是中断号，`n` 的取值范围为 0 ~ 31。编译器从 `8n + 3` 处产生中断向量，中断号和中断向量取决于 51 单片机的型号，常用的中断源和中断向量见表 13-2。

表 13-2 常用中断号与中断向量

中 断 源	中 断 号	中 断 向 量
外部中断 0	0	0x0003
定时中断 0	1	0x000B
外部中断 1	2	0x0013
定时中断 1	3	0x001B
串行口中断	4	0x0023

51 单片机可以在片内 RAM 中使用 4 个不同的工作寄存器组，每个寄存器组中包含 8 个工作寄存器（R0 ~ R7）。Keil C51 编译器扩展了一个关键字 `using`，专门用来选择 51 单片机中不同的工作寄存器组。`using` 后面的 `n` 是一个 0 ~ 3 的常整数，用于分别选择 4 个不同的寄

寄存器组。定义一个函数时 `using` 是一个选项，如果不用该选项，则由编译器自动选择一个寄存器组作为绝对寄存器组访问。需注意的是，关键字 `using` 和 `interrupt` 后面的表达式都不允许有运算符。

关键字 `interrupt` 也不允许用于外部函数，它对中断函数目标代码的影响有：在进入中断函数时，51 单片机的特殊功能寄存器 ACC、B、DPH、DPL 和 PSW 将被保护到堆栈；如果不使用关键字 `using` 进行工作寄存器组切换，则将中断函数中所用到的全部工作寄存器都入栈保存；函数退出之前将所有寄存器内容出栈恢复。

关键字 `using` 对函数目标代码的影响有：在函数的入口将当前工作寄存器组保护到堆栈中，指定的工作寄存器内容不会改变；函数退出之前将被保护的工作寄存器组从堆栈中恢复。

例 13-1 应用定时器中断的方式产生延时，使 P1.0 输出方波。

```
#include <reg51.h>           //51 单片机 C51 必须有该头文件
sbit P10 = P1^0;            //定义位变量 P10,符号 P10 指向 P1.0 端口
void main(void)              //主函数
{
    TMOD = 0x01;             //TMOD 的地址已在 reg51.h 中定义,0x01 为 T0 定时模式方式 1
    TH0 = 0x3c;              //TH0 和 TL0 的地址已在 reg51.h 中定义,设置 T0 的定时初值
    TL0 = 0xb0;
    TR0 = 1;                 //位单元 TR0、ET0 和 EA 的位地址已在 reg51.h 中定义,启动 T0 定时
    ET0 = 1;                 //允许 T0 中断
    EA = 1;                  //开总中断
    while(1);                //无限循环,等待 T0 中断
}                             //主函数编写结束

void timer0_ISR() interrupt 1 //定时器 T0 中断服务函数,中断号为 1
{
    TR0 = 0;                 //停止定时器 T0 的定时计数
    TH0 = 0x3c;
    TL0 = 0xb0;
    P10 = ~ P10;             //51 单片机 P1 口的 P1.0 引脚输出电平取反
    TR0 = 1;
}                             /* 定时器 T0 中断服务函数编写结束 */
```

注：编写的程序中，//符号后面的是注释内容；/* */符号中间的是注释内容。

13.4 C51 库函数

Cx51 编译器的运行库中包含有丰富的库函数，用户可在自己编写的 C51 语言程序中使用这些预定义的函数和宏。用户利用 Cx51 编译器的库中所提供的字符串操作、缓冲区操作、数据转换以及浮点运算等函数来执行一般的编程任务，可以使嵌入式软件的开发更容易。

使用库函数可以大大简化用户的程序设计工作，提高编程效率。由于 51 单片机本身的

特点，某些库函数的参数和调用格式与 ANSI C 标准有所不用。Keil Cx51 软件包提供了如下库函数文件：

C51S. LIB	不包含浮点运算的小型库函数
C51FPS. LIB	包含浮点运算的小型库函数
C51C. LIB	不包含浮点运算的紧凑型库函数
C51FPC. LIB	包含浮点运算的紧凑型库函数
C51L. LIB	不包含浮点运算的大型库函数
C51FPL. LIB	包含浮点运算的大型库函数
80751. LIB	应用于 Philips 8xC751 系列单片机的库函数

每个库函数都在相应的头文件中给出了函数原型申明，用户如果需要使用库函数，必须在源程序的开头将相应的头文件包含进来。如果省略了头文件，将不能保证函数的正确运行。C51 库函数中类型的选择考虑到了 51 单片机的结构特性，用户在自己的应用程序中应尽可能地使用最小的数据类型，以最大限度地发挥 51 单片机的性能，同时可以减少应用程序的代码长度。

下面将 C51 库函数中一些常用的函数列出并作必要的解释。

(1) 字符函数 CTYPE. H

在 C51 函数库中，字符函数的原型申明包含在头文件 CTYPE. H 中。

(2) 标准 I/O 函数 STDIO. H

C51 库函数中标准 I/O 函数的原型申明包含在头文件 STDIO. H 中，标准 I/O 函数通过 51 单片机的串行接口工作，如果希望支持其他 I/O 接口，只需修改 `_getkey()` 和 `_putchar()` 函数，库中的所有其他 I/O 支持函数都依赖于这两个函数模块。

(3) 字符串函数 STRING. H

字符串函数的原型申明包含在头文件 STRING. H 中，字符串函数通常接收字符串作为输入值。一个字符串包括 2 个或多个字符，字符串的结尾以空字符表示。在函数 `memcmp`、`memcpy`、`memchr`、`memccpy`、`memset` 和 `memmove` 中，字符串的长度由调用者明确规定，这些函数可以工作在任何模式。

(4) 标准函数 STDLIB. H

标准函数的原型申明包含在头文件 STDLIB. H 中，利用标准函数可以完成数据类型转换以及存储器分配操作。

函数原型：`void * calloc (unsigned int n, unsigned int size);`

功能：`calloc` 函数为 `n` 个元素的数组分配内存空间，数组中每个元素的大小为 `size`，所分配的内存区域用 0 进行初始化。返回值为已分配的内存单元的起始地址，如不成功则返回 0。

函数原型：`void free (void xdata * p);`

功能：`free` 释放指针 `p` 所指向的存储器区域，如果 `p` 为 NULL，则该函数无效，`p` 必须是以前用 `calloc`、`malloc` 或 `realloc` 函数分配的存储器区域。

函数原型：`void * malloc (unsigned int size);`

功能：`malloc` 函数在内存中分配一个大小为 `size` 的存储器空间，返回值为一个内存指针，分配内存指针的存储器空间与 `size` 所规定的大小相同。如果返回 NULL，则说明没有足

够的内存空间可用。

(5) 数学函数 MATH. H

数学函数的原型申明包含在头文件 MATH. H 中。

(6) 绝对地址访问 ABSACC. H

进行绝对地址访问的宏定义包含在头文件 ABSACC. H 中。

函数原型：
`#define CBYTE(( unsigned char volatile code * )0)`
`#define DBYTE(( unsigned char volatile idata * )0)`
`#define PBYTE(( unsigned char volatile pdata * )0)`
`#define XBYTE(( unsigned char volatile xdata * )0)`

功能：上述宏定义用来对 51 单片机的存储器空间进行绝对地址访问，可以作字节寻址。CBYTE 寻址 CODE 区，DBYTE 寻址 DATA 区，PBYTE 寻址分页 XDATA（用 MOVX @ Ri 指令访问的存储器空间）区，XBYTE 寻址 XDATA 区（用 MOVX @ DPTR 指令访问的存储器空间）。

从 `#define XBYTE(( unsigned char volatile xdata * )0)` 可以看出 XBYTE 是一个地址指针（可当成一个数组名或数组的首地址），它在文件 ABSACC. H 中由系统定义，指向外部 RAM（包括 I/O 口）的 0000H 单元，XBYTE 后面的中括号[]是指数组首地址 0000H 的偏移地址，即用 XBYTE[0x2000]可访问偏移地址为 0x2000 的 I/O 端口。

可以通过使用 `#define` 预处理命令，采用其他的符号来定义绝对地址。例如：

```
#define PA8155 XBYTE[0x7f00] /* PA8155 定义为 8155 命令寄存器,地址:0x7f00 */
```

下面介绍几种 C51 编程过程中常用的访问片外数据存储器空间地址的方法：

1) 使用扩展关键字 “_at_” 来对指定变量的存储器空间绝对地址。一般格式如下：

```
[存储类型] 数据类型 标识符 _at_ 地址常数  
xdata unsigned char pa _at_ 0x7f00; /* 变量 pa 定位于 xdata 空间,地址为 0x7f00 */  
pa = 0x01;
```

2) 使用基于存储器的指针来指定变量的存储器绝对地址。例如：

```
unsigned char xdata * pa; /* 定义一个指向 xdata 空间的指针变量 */  
pa = 0x7f00;  
* pa = 0x01;
```

3) 直接定义指向外部存储器的指针。例如：

```
(unsigned char xdata *)0x7f00 /* 定义指向 xdata 空间,地址为 0x7f00 的指针 */  
* (unsigned char xdata *)0x7f00 = 0x01;
```

以上三种方法都可以实现同一功能：指向片外数据存储器空间，地址为 0x7f00 的存储单元（或 I/O 口）写入数据 0x01。

(7) 内部函数 INTRINS. H

内部函数的原型申明包含在头文件 INTRINS. H 中。

函数原型：
`unsigned char _crol_(unsigned char val,unsigned char n);`
`unsigned char _irol_(unsigned int val,unsigned char n);`

```
unsigned char _lrol_(unsigned long val,unsigned char n);
```

功能：_crol_、_lrol_和_lrol_将变量 val 循环左移 n 位，它们与 51 单片机的“RL A”指令相对应。这三个函数的不同之处在于参数和返回值的类型不同。

如：temp = _crol_(temp,2)将变量 temp 的值循环左移 2 位后再赋值给 temp。

函数原型：unsigned char _cror_(unsigned char val,unsigned char n);

```
unsigned int _iror_(unsigned int val,unsigned char n);
```

```
unsigned long _lror_(unsigned long val,unsigned char n);
```

功能：_cror_、_iror_和_lror_将变量 val 循环右移 n 位，它们与 51 单片机的“RR A”指令相对应。

如：temp = _cror_(temp,2)将变量 temp 的值循环右移 2 位后再赋值给 temp。

函数原型：void _nop_(void);

功能：_nop_()函数，产生一个 51 单片机的 NOP 指令，该函数可以用于 C51 程序中的时间延时。C51 编译器对程序中出现_nop_()函数的地方，直接产生一条 NOP 指令。

函数原型：bit _testbit_(bit x);

功能：_testbit_产生一条 51 单片机的 JBC 指令，该函数对字节中某一位进行测试。如果该位为 1 则函数返回 1，同时将该位清零，否则返回 0。

13.5 C51 程序设计举例

本节的例题和应用 C51 编写的程序基本上与第 6 ~ 12 章的例题一一对应，便于读者对照 C51 和汇编程序的异同点，尤其是把握编程思路的共同点。由于本节例题的电路图与第 6 ~ 12 章例题的电路图基本相同，所以基本上没有绘制本节例题的电路图，请读者查阅与本节例题相关的第 6 ~ 12 章例题的电路图。

13.5.1 51 单片机内部资源的 C51 编程

1. 定时器/计数器的应用实例

在 51 单片机应用系统中，定时时间通常采用定时器的定时中断技术来产生，这与软件循环延时和定时器的定时查询编程方法完全不同，尽管最终都依赖系统的时钟，但在使用时中断技术进行定时计数时，单片机系统的其他事件可以同时进行；而使用软件定时计数时，只能等到软件循环完成后才能去执行其他事件，定时查询也需要用判断指令不断查询定时计数是否溢出，延时和定时过程中不能响应其他事件。

例 13-2 在振荡频率为 12 MHz 的 51 单片机系统上，用定时器 T1 产生 10 kHz 定时器滴答中断。

分析：产生 10 kHz 定时器滴答中断需要的定时周期为 100 μ s，由于单片机的系统时钟频率为 12 MHz，则机器周期为 1 μ s，所以定时器需计数 100 次。在定时器 T1 的工作方式选择时，选用方式 2 作为定时器 T1 的工作方式。

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

```
static unsigned long overflow_count=0; /* 定义全局滴答计数变量 */
```

```
void timer1_ISR(void)interrupt 3 /* 定时器 T1 中断服务程序 */
```

```

{
    overflow_count ++ ;
}
void main(void)
{
    TMOD = 0x20;                /* 设置定时器 T1 工作在方式 2 */
    TH1 = 256 - 100;            /* 设置定时初值 */
    TL1 = TH1;
    ET1 = 1;                    /* 允许定时器 T1 中断 */
    TR1 = 1;                    /* 启动定时器 T1 */
    EA = 1;                     /* 开总中断 */
    while(1);                   /* 程序主循环(无限循环),等待 T0 定时溢出并产生中断 */
}

```

每次定时器滴答中断都会使变量 overflow_count 加 1。

例 13-3 设 51 单片机的振荡频率为 12 MHz，要求在单片机的 P1.0 引脚上输出周期为 10 ms 的方波。

该例题与第 6 章例 6-3 相同，请参阅图 6-13 所示的程序流程图。

分析：在一个周期的方波中，包含高电平阶段和低电平阶段，且高低电平的宽度相等；周期为 10 ms 的方波，一个周期内其高低电平的维持时间各是 5 ms。因此，只需控制单片机的 P1.0 引脚每隔 5 ms 做一次电平翻转就能达到要求。单片机的振荡频率为 12 MHz，5 ms 的定时时间需要定时器计数 5000 次，定时器的工作方式只能采用方式 1，即 16 位的定时器/计数器。

```

#include <reg51.h>
sbit P10 = P1^0;
void timer0_ISR(void) interrupt 1
{
    TH0 = (65536 - 5000)/256;
    TL0 = (65536 - 5000)%256;
    P10 = ~P10;                /* 定时时间到,将 P10 反相 */
}
void main(void)
{
    TMOD = 0x01;                /* 选用定时器 T0 的方式 1 */
    TH0 = (65536 - 5000)/256;    /* 设置 T0 的定时初值 */
    TL0 = (65536 - 5000)%256;
    ET0 = 1;                    /* 允许定时器 T0 中断 */
    TR0 = 1;                    /* 启动定时器 T0 */
    EA = 1;                     /* 开总中断 */
    while(1);
}

```

例 13-4 某 51 单片机应用系统，系统晶振的时钟频率采用 12 MHz，通过定时器 T0 产

生 1s 的定时，驱动 P1.0 引脚电平取反操作。

该例题与第 6 章例 6-4 要求基本类似，但编程思路有所不同，请参阅该例题的分析说明和图 6-14。

分析：要求通过定时器 T0 产生 1s 的定时；由于 51 单片机系统的时钟频率为 12 MHz，定时器最大计数长度为 16 位（65536），最大定时时间为 65 ms 左右，不能满足 1s 要求，因此需要增加一个软件计数器，设定定时器 T0 定时时间为 50 ms，定时器 T0 中断 1 次，软件计数器加 1，当软件计数器计数 20 次时，时间到 1s，再驱动 P1.0 电平取反。

```
#include <reg51.h>

void main(void)
{
    static unsigned char cnt=0;
    TMOD=0x01;
    TH0=0x3c;
    TL0=0xb0;
    ET0=1;
    TR0=1;
    EA=1;
    while(1);
}

void timer0_ISR(void) interrupt 1
{
    if(cnt >= 20)                /* 判断定时时间是否到了 1s,时间到取反 P1.0 */
    {
        cnt=0;                  /* 清零软件计数器 */
        P1.0 = ~P1.0;
    }
    else cnt = cnt + 1;          /* 时间未到,软件计数器继续加 1 */
    TH0=0x3c;
    TL0=0xb0;
}
```

例 13-5 如图 13-1 所示，系统振荡频率为 12 MHz。在 P1 端口上连接 8 个发光二极管，要求使用定时器/计数器中断方式，控制 8 个发光二极管循环闪亮（即“流水灯显示”），每个发光二极管亮 1 s 后熄灭，同时控制下一个发光二极管亮 1 s，如此反复，使 8 个发光二极管像流水似的流动显示，并不断循环。

该例题与第 6 章例 6-5 基本相同，编程思路也基本相同，请参阅该例题的分析说明。

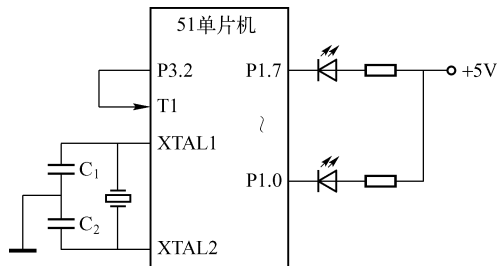


图 13-1 P1 口与 LED 的连接电路

分析：要求控制发光二极管显示 1 s 后熄灭，然后控制下一个发光二极管显示 1 s，因此需要使用定时器产生周期为 1 s 的定时。系统振荡频率为 12 MHz，机器周期为 1 μ s。采用复合定时的方法，即 T0 定时 50 ms + T1 计数的方法将定时时间扩展至 1 s。

这里使用定时器 T0 工作在方式 1，定时时间为 50 ms，定时时间到后控制 P3.2 反相，使 P3.2 端输出周期为 100 ms 的方波信号；并将这个方波信号送入定时器 T1 的输入端 T1，使定时器/计数器 T1 工作在计数方式，当计数满 10 次时，便产生 1 s 的定时时间，再控制发光二极管显示并移位。

```
#include <reg51.h>
#include <intrins.h>          /* 需使用内部函数 */
sbit P32 = P3^2;
unsigned char led = 0xfe;      /* 显示数据的初始值 */
void timer0_ISR(void) interrupt 1 using 1
{
    TH0 = (65536 - 50000)/256;  /* 设置定时初值,产生 50 ms 定时 */
    TL0 = (65536 - 50000)%256;
    P32 = ~P32;                /* 使 P3.2 端口产生周期为 100 ms 的方波 */
}
void timer1_ISR(void) interrupt 3 using 2
{
    P1 = led;                  /* 使某一位发光二极管亮 */
    led = _crol_(led,1);       /* 准备下次显示数据,“流水灯”从左至右流动 */
}
void main(void)
{
    TMOD = 0x21;               /* T0 工作在方式 1;T1 工作在方式 2 */
    TH0 = (65536 - 50000)/256;  /* 设置定时初值,产生 50 ms 定时 */
    TL0 = (65536 - 50000)%256;
    TH1 = 256 - 10;            /* 设置计数初值,计数 10 次 */
    TL1 = TH1;
    ET0 = 1;                   /* 开定时器 T0 中断 */
    ET1 = 1;                   /* 开定时器 T1 中断 */
    TR0 = 1;                   /* 启动定时器 T0 */
    TR1 = 1;                   /* 启动定时器 T1 */
    EA = 1;                   /* 开总中断 */
    while(1);
}
```

例 13-6 设 51 单片机系统振荡频率为 12 MHz，在 P1.0 脚上输出周期为 2.5 s，占空比为 20% 的脉冲信号。

分析：使定时器 T0 工作在定时方式，产生 10 ms 定时，则 2.5 s 的定时需要中断 250 次，占空比为 20%，高电平的时间应维持 50 次中断。


```

#include <reg51.h>
#define uchar unsigned char
sbit P10 = P1^0;
uchar time;
uchar period = 250;           /* 定时器的中断次数 */
uchar high = 50;              /* 高电平维持的定时器中断次数 */
void timer0_ISR( void) interrupt 1 using 1
{
    TH0 = (65536 - 1000)/256;
    TL0 = (65536 - 1000)%256;
    if( ++time == high) P10 = 0;    /* 高电平时间到,P1.0 变低 */
    else if( time == period)        /* 周期时间到,P1.0 变高 */
    {
        time = 0;                  /* 中断计数清零 */
        P10 = 1;
    }
}
void main( void)
{
    P10 = 1;
    TMOD = 0x01;
    TH0 = (65536 - 1000)/256;
    TL0 = (65536 - 1000)%256;
    ET0 = 1;
    TR0 = 1;
    EA = 1;
    while(1);
}

```

2. 51 单片机的外部中断应用编程

例 13-7 电路如图 6-10 所示，外部报警信号由“1”→“0”时，控制警示发光二极管亮，单片机复位后警示发光二极管灭。

该例题与第 6 章例 6-1 相同，请参阅该例题的分析说明和图 6-10 所示的电路图。

```

sbit P10 = P1^0;
void leds( void) interrupt 0
{
    EA = 0;
    P10 = 0;
    EA = 1;
}
void main( void)
{
}

```

```

P10 = 1;
IT0 = 1;                                /* 设置外部中断 0 为边沿触发方式 */
EX0 = 1;                                /* 允许外部中断 0 中断 */
EA = 1;                                /* 开总中断 */
while(1);                               /* 主循环,等待中断 */
}

```

例 13-8 外部中断源的扩展实现多路输入监控的电路如图 6-12 所示,通过外部中断 0 实现对 CH1、CH2、CH3 三条输入通道的监控。三条通道分别由三台设备提供信号。若设备工作正常,相应通道为高电平;若其中一台设备工作出错,相应的 CH_x 通道被拉为低电平。当任何一台设备无法正常工作,需点亮相应的发光二极管提示监控人员(1#设备对应 CH1 通道,故障报警灯为 D₁; 2#设备对应 CH2 通道,故障报警灯为 D₂; 3#设备对应 CH3 通道,故障报警灯为 D₃)。

该例题与第 6 章例 6-2 相同,请参阅该例题的分析说明和图 6-12 所示的电路图。

```

void leds( void) interrupt 0
{
    unsigned char temp = 0;
    temp = P1;                            /* 读 P1 口 */
    temp = temp & 0x07;                    /* 保留 P1 口数据低三位 */
    temp = (temp << 5) | 0x1f;             /* 将 P1 口数据低三位左移至高三位,并将低五位位置 1 */
    P1 = temp;                             /* 将 P1 口数据的第三位送到 P1,从而控制连接在 P1 */
                                           /* 高三位的 LED,实现三个故障报警灯指示三个不同的设备 */
}

void main( void)
{
    IT0 = 1;                              /* 设置外部中断 0 为边沿触发方式 */
    EX0 = 1;                              /* 允许外部中断 0 中断 */
    EA = 1;                              /* 开总中断 */
    while(1);                             /* 主循环,等待中断 */
}

```

3. 51 单片机的片内串行口应用编程

例 13-9 两个 51 单片机系统之间的串行通信,两系统振荡频率均为 11.0592 MHz。

甲机发送:将首地址为 ADDR1 的 128 B 片外数据存储器中数据块按顺序向乙机发送。

乙机接收:将接收到的 128 B 数据,按顺序存放放到以 ADDR2 为首地址的片外数据存储器中。

该例题与第 7 章例 7-5 要求基本类似,编程思路基本相同,请参阅该例题的分析说明和图 7-20。

分析:将两个单片机系统的串行通信波特率设为 9600 bit/s。则定时器的初值为 TL1 = TH1 = 0xfd。

发送端的 C51 程序如下:

```

#include <reg51.h>
unsigned char xdata ADDR[128];
unsigned char num = 0;
unsigned char * p;
void serial_ISR( void) interrupt 4
{
    TI = 0;
    num ++ ;
    if( num == 0x7f) ES = 0
    else
    {
        p ++ ;
        SBUF = * p;
    }
}

void main( void)
{
    SCON = 0x40;
    TMOD = 0x20;
    TH1 = 0xfd;
    TL1 = TH1;
    TR1 = 1;
    ES = 1;
    EA = 1;
    p = ADDR;
    SBUF = * p;
    while(1);
}

```

/* 在片外 RAM 中定义 128 个单元 */
/* 声明计数变量 */
/* 指向片外存储器的指针 */
/* 串行中断函数 */
/* 清发送中断标志 */
/* 修改计数变量的值 */
/* 判断是否发送完,若发完,则关串行中断 */
/* 否则,修改指针,发送下个数据 */
/* 置串行口工作在方式 1 */
/* 定时器 T1 工作在方式 2 */
/* 波特率为 9600 bit/s 时的定时器初值 */
/* 启动定时器 T1 */
/* 允许串行口中断 */
/* 将指针指向发送数据区首地址 */
/* 发送第一个数据 */

接收端的 C51 程序如下:

```

#include <reg51.h>
unsigned char xdata ADDR[128];
unsigned char num = 0;
unsigned char * p;
void serial_ISR( void) interrupt 4
{
    RI = 0;
    num ++ ;
    if( num == 0x7f) ES = 0
    else
    {
        p ++ ;
        * p = SBUF;
    }
}

```

```

    }
}
void main(void)
{
    SCON = 0x50;
    TMOD = 0x20;
    TH1 = 0xfd;
    TL1 = TH1;
    TR1 = 1;
    ES = 1;
    EA = 1;
    P = ADDR;
    while(1);
}

```

例 13-10 利用 51 单片机串行口实现多机通信。

下面给出一个利用 51 单片机串行口进行多机通信的 C51 程序。一个主机与多个从机进行单工通信，主机发送，从机接收。多机通信的过程是：主机先向从机发送一帧地址信息，然后再发送 10 位的数据信息。从机接收到地址后，与本机的地址相比较，若不相同则保持 SM2 = 1 状态不变，就接收不到数据帧。若地址相同，则使 SM2 = 0，准备接收主机发送来的数据信息。通信双方均采用 11.0592 MHz 的晶振。

发送程序代码如下：

```

#include <reg51.h>
#define count 10 /* 定义缓冲区大小 */
#define NODE_ADDR 64 /* 定义目的节点地址 */
unsigned char buffer[count]; /* 定义发送缓冲区 */
unsigned char num = 0; /* 定义计数变量 */
void main(void)
{
    SCON = 0xc0; /* 初始化串行口和波特率发生器 */
    TMOD = 0x20;
    TH1 = 0xfd;
    TL1 = 0xfd;
    TR1 = 1;
    ES = 1;
    EA = 1;
    TB8 = 1;
    SBUF = NODE_ADDR; /* 发送地址帧 */
    while(num < count); /* 等待全部数据发送完 */
}
void serial_ISR(void) interrupt 4
{
}

```

```

    TI = 0;
    num ++ ;                                /* 修改计数变量的值 */
    if ( num >= count) return;              /* 如果发送完毕则返回 */
    else
    {
        TB8 = 0;                            /* 设置数据帧标志 */
        SBUF = buffer[ num ];                /* 发送数据帧 */
    }
}
}

```

接收程序代码如下：

```

#include <reg51.h>
#define count 10                            /* 定义缓冲区大小 */
#define NODE_ADDR 64                        /* 定义目的节点地址 */
unsigned char buffer[ count ];              /* 定义接收缓冲区 */
unsigned char num = 0;                      /* 定义计数变量 */
void main( void )
{
    SCON = 0xf0;                            /* 初始化串行口和波特率发生器 */
    TMOD = 0x20;
    TH1 = 0xfd;
    TL1 = 0xfd;
    TR1 = 1;
    ES = 1;
    EA = 1;
    while( num < count );                    /* 等待全部数据接收完 */
}

void serial_ISR( void ) interrupt 4
{
    RI = 0;
    if ( RB8 = 1 )                          /* 如果是本节点的地址帧 */
    {
        if ( SBUF == NODE_ADDR) SM2 = 0; /* 则清零 SM2, 准备接收数据帧 */
    }

    num ++ ;                                /* 修改计数变量的值 */
    buffer[ num ] = SBUF;                    /* 将接收到的数据存入接收缓存区 */
    if ( num >= count) SM2 = 1;              /* 若接收完毕, 结束本次通信, 置 SM2 = 1, 准备下次通信 */
}

```

13.5.2 51 单片机扩展资源的 C51 编程

1. 并行接口 8155 的 C51 编程实例

例 13-11 51 单片机扩展并行接口 8155 与微型打印机的接口电路如图 9-16 所示。编写

打印字符“welcome2016”的程序。

该例题与第9章例9-6要求基本类似，接口电路相同，编程思路基本相同，请参阅该例题的分析说明和图9-16。

例9-6已得到8155各端口的地址：

命令寄存器地址：0x0100；A口地址：0x0101；B口地址：0x0102。

源程序代码如下：

```
#include <reg51.h>
#include <absacc.h>
#define uchar unsigned char
#define COM8155 XBYTE[0x0100]
#define PA8155 XBYTE[0x0101]
#define PB8155 XBYTE[0x0102]
sbit BUSY = P3^3;
uchar code line[] = {0x57,0x45,0x4c,0x43,0x4f,0x4d,0x45}; /* welcome 的 ASCII 码表 */
/***** 字符打印函数 *****/
void prnchar(uchar a)
{
    PA8155 = a;
    PB8155 = 0x00; /* STB = 0 */
    PB8155 = 0x01; /* STB = 1, 产生 STB 的上升沿 */
    while(BUSY); /* 等待打印结束 */
}
/***** 行打印函数 *****/
void prnline(void)
{
    uchar i;
    for(i = 0; i < 6; i++)
    {
        PA8155 = line[i];
        PB8155 = 0x00;
        PB8155 = 0x01;
        while(BUSY);
    }
}
/***** 主函数 *****/
void main(void)
{
    COM8155 = 0x01;
    prnline(); /* 打印输出“welcome” */
    prnchar(0x0d); /* 换行 */
    prnchar(0x32); /* 打印输出“2016” */
    prnchar(0x30);
```

```

        prnchar(0x31);
        prnchar(0x36);
    }

```

2. 线反转法矩阵式键盘编程实例

例 13-12 以图 11-3 的线反转法键盘电路为例，设系统晶振的时钟频率采用 12 MHz，定时器工作于方式 1，初始化程序中，设置 4 根列线 P1.0 ~ P1.3 输出全部为 0，4 根行线 P1.4 ~ P1.7 输出全部为 1，设置定时器 T0 工作于方式 1，不启动定时器 T0 定时和关定时器 T0 中断，外部中断 0 为边沿触发，并允许中断，当无键按下时， $\overline{\text{INT0}}$ 输入端为高电平；有键按下时，产生一个下降沿触发中断；若再多按下一个或多个键， $\overline{\text{INT0}}$ 输入端维持低电平，不会产生中断申请信号。

该例题来自于 11.1.2 节中“线反转法键盘的中断控制方式编程”这部分的内容，请参阅该节内容及例题与分析。

(1) 确定按下键行号的函数

```

unsigned char tbl[8] = {0xff,0,1,0,2,0,0,0,3};
unsigned char hkcs(void)
{
    unsigned char temp;
    P1 = 0xf0;          /* P1 低四位输出 0 */
    temp = P1;          /* 读 P1,将 P1 口第四位置 1,保留 P1 口高四位 */
    temp = temp | 0x0f;
    temp = ~temp;       /* 取反 */
    temp = _crol_(temp,4); /* 高低四位交换 */
    return tbl[temp];    /* 查表得出行号,并返回行号 */
}

```

(2) 确定按下键列号的函数

图 11-3 中，设置 4 根行线 P1.4 ~ P1.7 输出全部为 0，若有键按下，则 P1.0 ~ P1.3 的输入不会全为高电平。连接的某一列有按键被按下，则对应输入端口的输入状态为 0。

一个按键被按下，P1.0 ~ P1.3 取反后的结果将是 01H、02H、04H、08H 中的一个值，可以通过查表来获得列号。

```

unsigned char lkcs(void)
{
    unsigned char temp;
    P1 = 0x0f;          /* P1 低四位输出 0 */
    temp = P1;          /* 读 P1,将 P1 口第四位置 1,保留 P1 口高四位 */
    temp = temp | 0xf0;
    temp = ~temp;       /* 取反 */
    return tbl[temp];    /* 查表得出列号,并返回列号 */
}

```

(3) 外部中断 0 和定时中断服务程序

```

void int0_ISR( void) interrupt 0          /* 外部 INT0 中断服务程序 */
{
    TH0 = 0xd8;
    TL0 = 0xf0;
    ET0 = 1;                             /* 允许定时器 T0 中断 */
    TR0 = 1;                             /* 启动定时器 T0 */
}

void timer0_ISR( void) interrupt 1
{
    unsigned char htemp, ltemp;
    htemp = hkcs( );                     /* 调用确定按下键行号的函数读出行号 */
    if( temp != 0xff)                     /* 键盘防抖 */
    {
        ltemp = lkcs( );                 /* 调用确定按下键列号的函数读出列号 */
        keyCode = ltemp * 4 + htemp;      /* 列号 * 4 加上行号为实际键值 */
        ...                             /* 根据不同键值调用不同处理程序,此略 */
    }
    P1 = 0xf0;
    EX0 = 1;
}

```

3. LED 显示器动态显示编程实例

例 13-13 LED 数码管动态显示子程序设计。参考图 11-8 中的 8155 与单片机的接口电路连接, 8155 A 口 (即段控口) 地址为 0001H, C 口 (即位控口) 地址为 0003H。delay() 延时子程序, 延时时间大约为 10 ms。

该例题与第 11 章例 11-7 要求基本类似, 接口电路相同, 编程思路基本相同, 请参阅该例题的分析说明和图 11-8。

```

unsigned char tab[ ] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
                        0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,0 }; /* 显示译码表 */
unsigned char dispbuf[ ] = {1,2,3,4,5,6}; /* 显示缓冲区 */
void display( void)
{
    unsigned char disp;
    unsigned char temp = 0x20;           /* 定义位码变量,从最左边一位开始显示 */
    for( unsigned char i = 0, i < 6; i ++ ) /* 6 位数码管,进行 6 次显示循环 */
    {
        disp = dispbuf[ i ]              /* 取显示缓冲区数据 */
        * ( unsigned char xdata *) 0x0001 = tab[ disp ]; /* 译码后送至段控口 */
        * ( unsigned char xdata *) 0x0003 = temp;      /* 控制位控口,先显示最左边的数码管 */
        delay( );
        temp = temp >> 1;                 /* 位码移位,进行位扫描 */
    }
    temp = 0x20;
}

```


4. DAC0832 数码转换器编程实例

例 13-14 DAC0832 与单片机接口的单缓冲连接电路如图 10-4 所示，试编写 DAC0832 以单缓冲方式输出如图 10-5 所示锯齿波的程序。设 DAC0832 的译码地址为 0xfffe，单片机晶振频率 12 MHz， $T_a = 100\text{ ms}$ 。

该例题与第 10 章例 10-1 要求相同，接口电路相同，编程思路相同，请参阅该例题的分析说明、图 10-4 和图 10-5。

分析：按照图 10-5，锯齿波的周期 $T_a = 100\text{ ms}$ ，最大幅值 $A = 100$ ，则 DAC0832 的数据输入每 1 ms 加 1 一次，应用 T0 工作于方式 1 定时 1 ms，采用定时中断方法编程实现。

```
#include <reg51.h>
#include <absacc.h>
#define DAC0 XBYTE[0xfffe]
unsigned char count;
void timer0_ISR( void) interrupt 1
{
    TH0 = 0xfc;
    TL0 = 0x18;
    DAC0 = count++;
    If( count >= 100) count = 0;
}
void main( void)
{
    TMOD = 0x01;          /* 定时器 T0 工作在方式 1 */
    TH0 = 0xfc;           /* 设置定时初值 0xfc18 */
    TL0 = 0x18;
    ET0 = 1;              /* 允许定时器 T0 中断 */
    TR0 = 1;              /* 启动定时器 T0 */
    EA = 1;               /* 开总中断 */
    while(1);
}
```

5. ADC0809 的 C51 编程实例

例 13-15 ADC0809 与 8051 的接口电路如图 10-15 所示，ADC0809 的 EOC 输出端经过一个反相器后送单片机的 $\overline{\text{INT0}}$ 引脚，模数转换结束后，引起 EOC 引脚电平变化，可以采用软件查询或中断方式读取 A-D 转换后的值。模拟量输入通道 IN0 的地址为 0x7ff8。

该例题与第 10 章例 10-3 和 10-4 要求基本相同，接口电路相同，编程思路基本相同，请参阅例 10-3 和 10-4 的分析说明、图 10-15 和 10-16。

(1) 中断方式

```
#include <reg51.h>
#include <absacc.h>
#define IN0 XBYTE[0x7ff8]
#define uchar unsigned char
```

```

uchar in0;          /* 全局变量,用于保存模拟量的转换结果 */
void IN0_ISR( void) interrupt 0
{
    in0 = IN0;      /* 读转换结果,并保存 */
    IN0 = 0;        /* 启动 ADC0809 */
}
void main( void)
{
    EX0 = 1;        /* 允许外部 INTO 中断 */
    EA = 1;
    IN0 = 0;        /* 启动 ADC0809 */
    while(1);      /* 主循环,等待中断 */
}

```

(2) 定时采样编程实例

在实际的 51 单片机应用系统中,模拟量数据采集总是需要设置采样频率,单片机利用两次采样之间的时间间隔去执行其他任务。假设采样频率为 1 kHz。

```

#include <reg51.h>
#include <absacc.h>
#define IN0 XBYTE[0x7ff8]
#define uchar unsigned char
uchar in0;          /* 全局变量,用于保存模拟量的转换结果 */
void IN0_ISR( void) interrupt 0
{
    TH0 = 0xfc;
    TL0 = 0x18;
    in0 = IN0;      /* 读转换结果,并保存 */
}
void timer0_ISR( void) interrupt 1
{
    IN0 = 0;        /* 启动 ADC0809 */
}
void main( void)
{
    TMOD = 1;
    TH0 = 0xfc;
    TL0 = 0x18;     /* 设置定时器 T0 工作方式和初值 */
    ET0 = 1;
    TR0 = 1;
    EX0 = 1;
    EA = 1;        /* 允许定时器 T0 中断和外部 INTO 中断 */
    while(1);      /* 主循环,等待中断 */
}

```

(3) 多路采集编程实例

采用查询方式采集 8 路模拟量的程序如下：

```
#include <reg51.h>
#include <absacc.h>
#define IN0 XBYTE[0x7ff8]
#define uchar unsigned char
uchar x[8];          /* 定义用于保存 8 通道模拟量转换结果的数组 */
sbit BUSY = P3.2;
void adc0809 (void)
{
    uchar i;
    uchar xdata *ad_addr;
    ad_addr = 0x7ff8;
    for (i = 0; i < 8; i++)
    {
        *ad_addr = 0;      /* 启动 ADC0809 转换 */
        i = i;
        i = i;              /* 延时,等待 EOC 变低 */
        while( BUSY == 0); /* 等待 A-D 转换结束 */
        x[i] = *ad_addr;    /* 保存转换结果 */
        ad_addr++;          /* 下一个通道 */
    }
}
```

例 13-16 编写 51 单片机选择 TLC2543 的某一通道（例如 AIN0 通道）进行 1 次数据采集的子程序。51 单片机与 TLC2543 的 SPI 接口电路图参考图 12-19。

该例题与第 12 章例 12-2 要求相同，接口电路相同，编程思路相同，请参阅该例题的分析说明和图 12-19。

```
#define uint unsigned int
sbit CLK = P3^2;
sbit ADIN = P3^3;
sbit DOUT = P3^4;
sbit CS = P3^5;
uint Rd_2543(unsigned char ch)      /* ch 为模拟量通道号 */
{
    uint i;
    uint ad_value = 0;
    CLK = 0;
    CS = 0;
    ch <<= 4;
    for(i = 0; i < 12; i++)
    {
```

```

        if(DOUT)  ad_value |= 0x01;
        ADIN = (bit)(port & 0x80);
        CLK = 1;
        _nop_();
        _nop_();
        _nop_();
        CLK = 0;
        _nop_();
        _nop_();
        _nop_();
        ch = ch << 1;
        ad_value = ad_value << 1;
    }

    CS = 1;
    ad_value = ad_value >> 1;
    return ad_value;
}

```

习题

- 13-1 哪些变量类型是 51 单片机直接支持的?
- 13-2 简述 C51 的数据存储类型。
- 13-3 简述 C51 对 51 单片机特殊功能寄存器的定义方法。
- 13-4 C51 的 data、bdata、idata 有什么区别?
- 13-5 C51 中的中断函数和一般的函数有什么不同?
- 13-6 C51 采用什么形式对绝对地址进行访问?
- 13-7 在 51 单片机系统中, 已知振荡频率是 12 MHz, 用定时器/计数器 T0 实现 P1.1 产生周期是 2 s 的方波, 试编程。
- 13-8 51 单片机系统应用 8155 扩展 I/O 口, 设其 A 口为方式 1 输入, B 口为方式 1 输出, C 口余下的引脚用于输出, 设 8155 命令端口的地址为 0x7f00; 试写出其初始化程序。
- 13-9 编程实现由 DAC0832 输出的幅度和频率都可以控制的三角波, 即从 0 上升到最大值, 再从最大值下降到 0, 并不断重复。假设 DAC0832 的地址为 0x7f00。

第 14 章 单片机应用系统的开发与举例

本章介绍单片机应用系统的开发过程和设计原则，通过单片机应用系统的举例，使读者了解单片机应用系统开发的一般过程和系统设计方法。

14.1 单片机应用系统的开发过程

在单片机应用系统的开发过程中，为了完成某一项任务，单片机系统需要包含硬件部分和软件部分。软件和硬件的设计密切相关，协调一致，不能脱离软件去设计硬件，软件也需要根据硬件进行设计和调试。某些硬件可以用软件替代其功能，例如按键的消抖动、电源待机电指示灯控制等。软件的处理需要硬件的配合才能更好地完成功能，如模拟数据的采集，为了在软件中能采集到更准确、稳定的数据，硬件的采集电路可以在滤波、采样上替软件做好前级准备。这就是所谓的“软硬结合”。

单片机应用系统的开发基本过程如图 14-1 所示。图 14-1 中的每个阶段都是相互关联的，不是绝对分开的，并且在实际设计开发过程中经常是交叉或者同时进行的。例如在硬件设计的过程中，软件设计可以同时进行。对于硬件电路测试，可以专门编写简单的测试程序予以测试，需要确保硬件电路能够正常运行。对于软件可以按照模块化程序分别进行测试，先仿真，后下载到单片机程序存储器中进行运行测试，成功之后逐步增加程序模块进行联调，这样便于找出问题和修改程序。

14.1.1 需求分析与可行性论证

在进行单片机应用系统开发时应首先根据应用系统的设计要求，分析系统的工作原理，划分功能模块，提取技术指标，通过调研与资料查阅确定能否采用以单片机为核心的应用系统达到设计目标，在此基础上确定单片机应用系统的总体开发方案，在制定总体方案时应注意以下两点：

(1) 确定单片机型号

在完成可行性分析之后，即可进行总体方案设计，设计时首先应根据系统要实现的功能、规模和复杂程度，确定其核心部分——单片机型号。目前国内外单片机种类很多，指令位数、内部存储器容量、定时器和中断等内部资源的配置相差很大，在选择时应根据以下几个原则：①性价比高，在满足性能指标要求的基础上，性价比要高，尤其是在大批量生产的时候；②在条件允许的情况下，尽量选择配置高的单片机，以减少外围电路，提高系统的可靠性，缩短研制周期；③资源充足，技术成熟，性能可靠，有成熟的开发工具；④在研制阶段可选用带 Flash ROM 的 CPU 芯片，如 89cxx 系列，无须擦除器，便于调试，研制成功后，再换上相应型号而价格较低的芯片。

(2) 合理划分软硬件功能

系统软硬件功能的划分应根据系统要求来定，多用硬件可提高系统的运行速度，减少程序的复杂性，但会增加成本，降低系统的灵活性；相反多用软件实现相应的功能，可提高灵

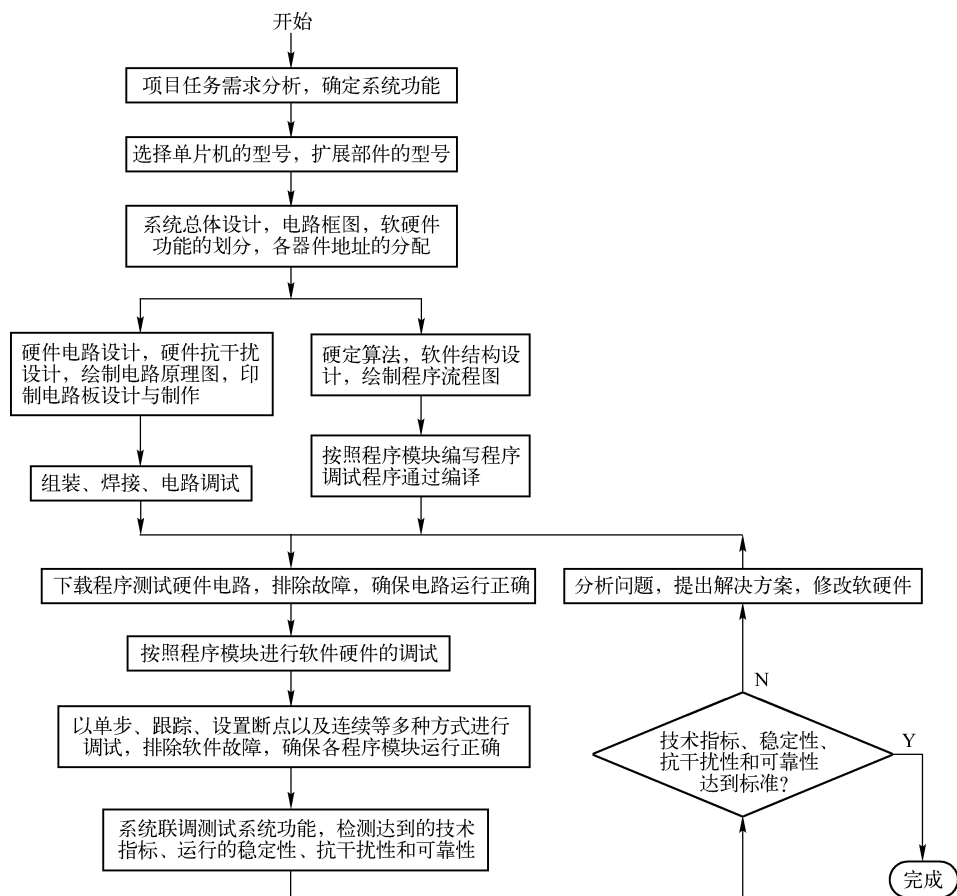


图 14-1 项目开发流程图

活性，但又会降低系统的运行速度，增加程序设计的复杂性，因此，要合理划分软硬件功能。

14.1.2 硬件电路设计原则

单片机应用系统的硬件电路设计包含两部分内容：一是系统功能扩展，即单片机内部的资源不能满足应用系统的要求时，必须在片外进行扩展；二是系统外围设备配置，即按照系统功能要求配置外围设备，如键盘、显示器、打印机、A-D 和 D-A 转换器等，要设计合适的接口电路，系统的扩展和配置应遵循以下原则：

- 1) 尽可能选择标准化、模块化的电路。
- 2) 系统扩展与外围设备的配置水平应充分满足应用系统的功能要求，应留有冗余以备设计改动与功能扩展之需。
- 3) 硬件结构应结合应用软件方案一并考虑，软件能实现的功能尽可能由软件实现，以简化硬件结构，但必须注意，由软件实现的硬件功能，一般响应时间比硬件时间长，且占用 CPU 时间。
- 4) 系统中的相关器件要尽可能做到性能匹配。如选用 CMOS 芯片单片机构成低功耗系统时，系统中所有芯片都应尽可能选择低功耗产品。

5) 单片机外围电路较多时, 必须考虑其驱动能力, 可通过增设驱动器增强驱动能力或减少芯片功耗来降低总线负载。

6) 根据系统的规模选择功能强的单片机以尽量减少外围电路, 提高系统的可靠性。系统器件越多, 器件之间相互干扰也越强, 功耗也增大, 也不可避免地降低了系统的稳定性。

7) 可靠性及抗干扰设计是硬件设计必不可少的一部分, 它包括芯片、器件选择, 去耦滤波, 印制电路板布线和通道隔离等。

14.1.3 软件设计原则

单片机应用系统中的应用软件应根据系统功能进行设计, 要求可靠地实现系统的各种功能。应用系统种类繁多, 应用软件各不相同, 基本设计原则如下:

1) 软件结构清晰、简洁、流程合理, 应加程序注释, 以便于阅读与修改。

2) 各种功能程序模块化、子程序化, 这样既便于调试、链接, 又便于移植、修改。

3) 合理规划程序存储区、数据存储区以节约内存容量, 操作方便。

4) 运行状态采集标志化管理。各个功能程序运行状态、运行结果以及运行要求都设置状态标志以便查询, 程序的转移、运行和控制尽量用状态标志条件来控制。

5) 软件抗干扰设计。软件抗干扰是单片机应用系统提高可靠性的有力措施。

6) 为了提高运行的可靠性, 在应用软件中设置自诊断程序, 在系统工作运行前先运行自诊断程序, 用以检查系统各特征状态参数是否正常。

14.1.4 软硬件测试与程序固化

单片机应用系统软硬件测试是应用系统开发的一个重要步骤, 包括硬件调试和软件调试。一般来说, 先检测硬件, 然后再进行软硬件综合调试, 测试正常后方可进行程序固化和试运行。目前市场上仿真器的型号很多, 使用方法大同小异, 可根据单片机的型号选择适当的仿真器进行系统调试。

(1) 元器件安装与调试

1) 在未焊上各元器件管座或元件之前, 首先用眼睛和万用表直接检查线路板各处是否有明显的断路、短路的地方, 尤其是要注意电源是否短路。

2) 元器件在焊接过程中要逐一检查, 例如二极管、晶体管、电解电容的极性, 电容的容量及耐压, 元件的数值是否正确等。

3) 管座、元件焊接完毕, 还要仔细检查元件面各元件之间裸露部分有无相互接触现象, 焊接面的各焊点间、焊点与近邻线有无连接。对布线密集或未加阻焊处理的印制电路板更应注意检查这些可能造成短路的原因。

4) 先空载上电, 检查电路板各引脚及插件上的电位是否正常, 特别是单片机引脚上的各点电位, 若一切正常, 将芯片插入各管座, 再通电检查各点电压是否达到要求, 逻辑电平是否符合电路或器件的逻辑关系。

(2) 功能模块的软件调试

静态测试正常后, 即可进行软硬件调试, 测试系统功能。调试的方法是首先将整个应用系统按其功能分成若干模块, 如系统扩展模块、输入模块、输出模块和 A-D 模块等, 可先

编写一些测试程序对其进行测试，确保硬件电路功能正常，然后将应用软件按模块分段进行仿真，调试，最后再进行联调。

(3) 程序固化

用户程序经过仿真调试，并连续运行一段时间无故障后，即可进行程序固化，固化的方法有两种：一是使用 ISP 电缆进行在线调试与固化；二是采用专门的编程器进行固化。

例如 STC 系列单片机程序的下载软件人机界面如图 14-2 所示，程序下载步骤如下：



图 14-2 STC 系列单片机下载软件人机界面

- 1) 打开下载软件，并且在“单片机型号”一项中选择实际使用的单片机型号。
- 2) 将下载器与计算机串口连接好，然后在“串口号”中选择当前端口号。可以右击“我的电脑”→“管理”→“设备管理器”→“端口”查看当前连接的端口编号。
- ③ 单击“打开程序文件”，选择即将要烧录进单片机的程序文件，通常为 .hex 文件。

④ 单击“下载/编程”，然后给单片机上电，即可将程序下载到单片机中。STC 系列的单片机，在程序下载的时候需要单片机冷启动，即在单击“下载/编程”的时候，如果单片机是处于通电状态，则需要先断电，然后再重新上电，程序才会下载到单片机中。因为此系列单片机内部分为两部分，一部分是 ISP 引导程序，另一部分是用户程序区，在单片机上电的时候，单片机会先运行 ISP 引导程序，如果此时有程序下载的指令，则下载程序，如果没有，则运行用户程序。

程序固化后即可进行组装与运行，一般来说，经过仿真调试后的程序固化后能够正常运行，如工作不正常，可重点检查复位电路与系统晶振。由于单片机多用于工业现场，环境复杂，干扰信号较多，因此，仿真成功的应用系统还需经过现场的进一步测试完善后方才完成了整个开发任务。

14.2 单片机应用系统的开发工具

单片机系统的开发需要经历调试阶段，并且只有在调试过程中才能发现问题，修正BUG，最终完成产品的功能和技术要求。甚至某些隐藏得比较深的BUG在开发调试过程中难以发现，而需要到客户那里或者经过长时间测试之后问题才会暴露出来。所以在相对较复杂的项目中，程序很难一次就能调试成功，即使是资深的程序员或者工程师也是如此。

在单片机系统的开发过程中，需要使用到一些辅助的工具才能完成。首先是软件开发工具，常用的开发工具有 Keil、IAR 和 AVRstudio 等。不同内核的单片机，用到的开发软件是不一样的。其次是编程器或者称下载器，编程器的功能主要是在程序的编写完成并且生成机器代码（通常是 Hex 文件或者 Bin 文件）之后，将机器代码烧录到单片机中。程序的烧录，通常还需要烧录软件，例如 STC 单片机的 STC_ISP、AVR 单片机的 AVR_fighter。也有直接从程序开发软件中下载的，比如 STM32 的芯片，通过使用 J-link 等下载器，可以直接从开发软件中下载程序到单片机中，开发软件自带下载和在线调试功能，提高了开发的效率。

14.3 应用系统的可靠性设计

14.3.1 主要干扰源

单片机系统的应用场合非常广泛，涉及医疗、汽车、工业生产和农业等，应用现场中会存在着不同的干扰源，这些干扰源会影响到单片机系统运行的稳定性，严重的甚至导致系统无法运行。通常会对单片机系统产生影响的条件有：

- 1) 外部电气条件：如电源电压的稳定性，强电场、磁场的影响。
- 2) 外部空间条件：温度、湿度和空气洁净度等。
- 3) 外部机械条件：震动、冲击、放置不好等。

14.3.2 硬件可靠性设计

为了设计出稳定、可靠的单片机系统，增强系统的抗干扰能力，应该遵循的原则：元器件选择是根本；合理安装调试是基础；系统设计是手段；外部环境是保证。

1) 精选元器件，保证精度。元器件的选择要控制精度，特别是核心部分的器件。尽量避免同一台设备中使用不同厂家的同一型号器件，尽量避免因使用批次不同产品带来的影响。

2) 电路设计中，要求元件的布局要合理，布线要规范，减少器件直接的相互干扰。特别是高频电路部分，PCB 的布局更加需要谨慎、讲究。在关键的核心部件中，条件允许的情况下，可以采用冗余结构，使得当某些器件发生故障时能切换到备用器件上，不影响整个系统的安全运行。

3) 采取屏蔽、隔离措施，远离产生强磁场干扰的场合，必要时可以模块化设计，独立分离处理。同时，安装注意避免震动、潮湿和高温等环境影响。

- 4) 选用集成度高的器件。因为集成度高的器件，稳定性更好，PCB 布局也方便简单，

大大地减少了干扰的产生。同时，电路器件少了，电路简单了，在后期的维护和维修过程中也更加方便，大大减少了工作量。

5) 降额使用。即在额定电压和电流的条件下使用元器件，这样可以调元器件的可靠性。多使用无源元件（如电阻、电容）、大功率器件、电源模块或大电流高压开关器件等。尽量不采用 TTL 器件，因为 TTL 电路对工作电压范围要求较严格，不能降额使用。MOS 电路也可以降额使用，但因其工作电流十分微小，失效主要不是由功耗发热引起的，故效果不明显。

14.3.3 软件可靠性设计

1) 软件冗余技术，也称“容错技术”或“故障掩盖技术”。主要是通过软件的调试，允许一定程度上的误差或错误出现，同时在软件中进行处理。例如，通过 ADC 采样的电压值作为某个状态执行的条件判断，由于外部传感器的精度和电源的稳定性因素的影响，会造成 ADC 读取到的电压值有一定的波动和不稳定。此时可以将判断的电压范围适当增大，以确保执行的条件判断能够准确获取而不会造成漏判和误判。

2) 时间冗余技术，也称“重复检测技术”。可采用重复执行某一操作或某一程序，并将执行结果与前一次的结果进行比较来确认系统工作是否正常。只有两次结果相同时，才允许执行下一步操作；否则重复执行前一次操作。如果第三次结果与前两次结果的其中一次相同，则认为另一个结果是偶然故障引起的，应当删除。如果三次均不同，则初步判定为硬件故障，需要进一步检查并排除。用时间为代价来换取可靠性，不增加硬件投资，但降低了运行速度，只能用在执行时间比较宽余、操作步骤比较重要的场合。

3) 数据冗余。保护数据对提高系统可靠性十分重要，常将重要数据存储于不同的空间中。一旦某些存储空间或某一备份被损坏，控制程序即自动从其他空间中重新复制数据。

4) 故障自动检测与诊断技术。系统运行中，可以增加一些自动检测或诊断的措施，当出现故障的时候软件能第一时间检测到，并作出相应的反应，避免造成更大的损失。例如，可以在程序运行的过程中，从某个器件中读取数据，判断数据是否正确，如果不正确则说明硬件系统已经出现问题，应该向客户发出故障警报，问题严重的应该及时停止系统工作，确保安全。

14.4 单片机应用系统设计举例

单片机应用系统的硬件电路设计可以分为三种基本形式：一是以扩展并行接口芯片为主的应用系统；二是以串行总线接口芯片为主的应用系统；三是既扩展并行接口芯片，也扩展串行总线接口芯片的应用系统；四是选用单片机内装能够满足应用系统需求的功能部件单片机，只需要在单片机外部连接简单的扩展电路的应用系统。

无论单片机应用系统的硬件电路采用何种方法设计，其软件主要由初始化程序（C51 的主函数）和各功能模块的子程序（C51 的各功能模块的函数）组成。

本节重点介绍以扩展并行接口芯片为主的应用系统和以串行总线接口芯片为主的应用系统的软硬件设计。

14.4.1 单片机应用系统并行扩展芯片的基本接口电路

前面章节的例题都是针对一个问题提出硬件电路模块和软件模块的解决方法，而一个单

[illegible]

对于图 14-3 的综合应用电路图,如果需要设计的单片机应用系统的外部扩展部件以并行接口芯片电路为主,则可以应用图 14-3 的综合应用电路图为基础进行设计。图 14-3 中,74LS138 的译码输出还有 $\overline{Y4} \sim \overline{Y0}$ 未使用,还可以扩展 5 块单片机接口芯片,扩展的 8 块芯片的地址之间不会出现交叉重叠的现象。对于 51 单片机,一般需要扩展一块 RAM 芯片,如果选用内部有 2 KB 以上的增强型 51 单片机可以不扩展 RAM 芯片。

1) 如果其他扩展部件需要应用 P1 口, 则需要扩展一片 82C55 或 81C55 与矩阵式键盘连接。

2) 图 14-3 中的 6 位静态 LED 显示器电路是第 11 章图 11-9 所示的电路, 如果 LED 是高亮度的发光二极管, 则 74LS373 锁存器可以选用驱动能力强的锁存器, 或者在每块

74LS373 输出端增加一块功率驱动芯片，例如 ULN2003 或 ULN2803 芯片。

3) 键盘和 LED 显示还可以选用专用可编程键盘/显示器接口芯片 8279，可编程键盘/显示器接口芯片 8279 能够直接与 51 单片机总线连接，请读者参考 8279 的相关资料。

14.4.2 直流电动机转速控制系统的设计

1. 设计要求

设计一个小型直流电动机的转速控制系统，要求具有正转、反转，转速控制，转速的显示，电动机电流的显示与报警。按键操作能够控制电动机的停机、正反转，以及转速的设置。

2. 电路设计

根据设计要求，电路由多个电路模块组成：电动机驱动电路，电动机电流和转速检测电路，键盘、显示电路，存储器电路等。

(1) 电动机驱动电路

选用具有 DMOS 输出能力的直流电动机驱动芯片 TLE5206，TLE5206 内部集成了功率 H 桥，能够提供 5 A 的直流电流连续电路，6 A 的峰值电流；电源电压 DC40 V；输出短路保护和过温保护与滞后诊断；内部续流二极管，使用温度范围广： $-40^{\circ}\text{C} < T_j < 150^{\circ}\text{C}$ ；控制引脚可以与单片机的 I/O 端口直接连接。TLE5206 的封装与引脚如图 14-4 所示，内部功能框图如图 14-5 所示。

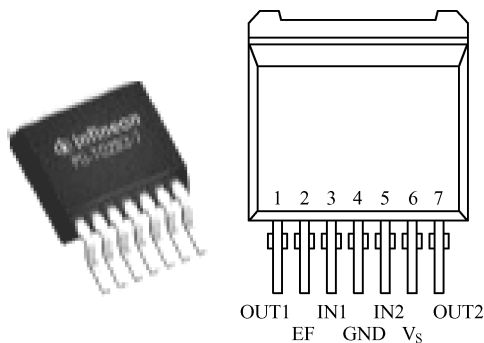


图 14-4 TLE5206 的封装与引脚图

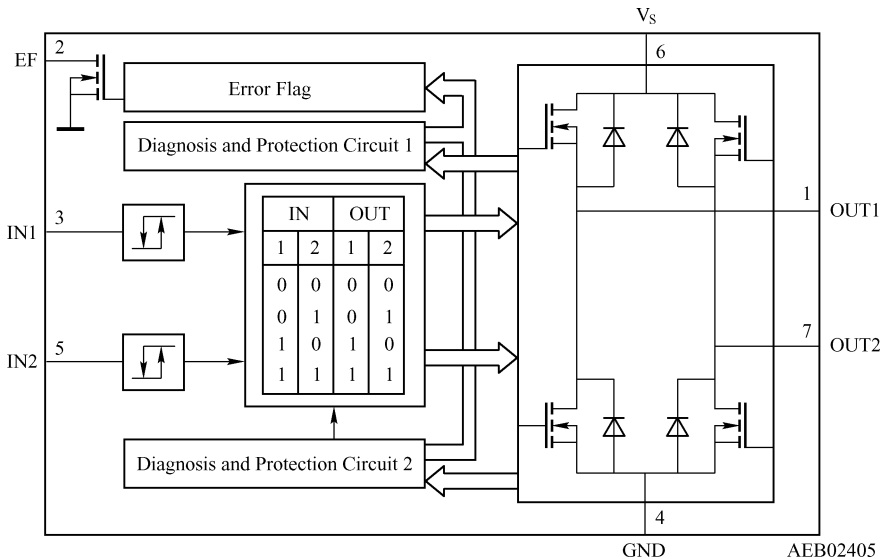


图 14-5 TLE5206 内部功能框图

TLE5206 的引脚 IN1 和 IN2 输入电动机控制信号，IN1、IN2 = 00 和 11 时，电动机停转；IN1、IN2 = 10 时，电动机向一个方向转动，假设为正转；IN1、IN2 = 01 时，电动机反转。如果保持 IN1、IN2 = 10 不变，电动机则高速正转；如果 IN1 输入脉冲宽度调制（PWM）信号，则可以实现电动机正转的调速，反转调速控制亦然。

在图 14-6 的小型直流电动机单片机控制体系电路结构图中，将 IN1 和 IN2 两个引脚分别和两个与门的输出端连接，两个与门的一个输入端与 51 单片机的 P1.7、P1.6 连接，两个与

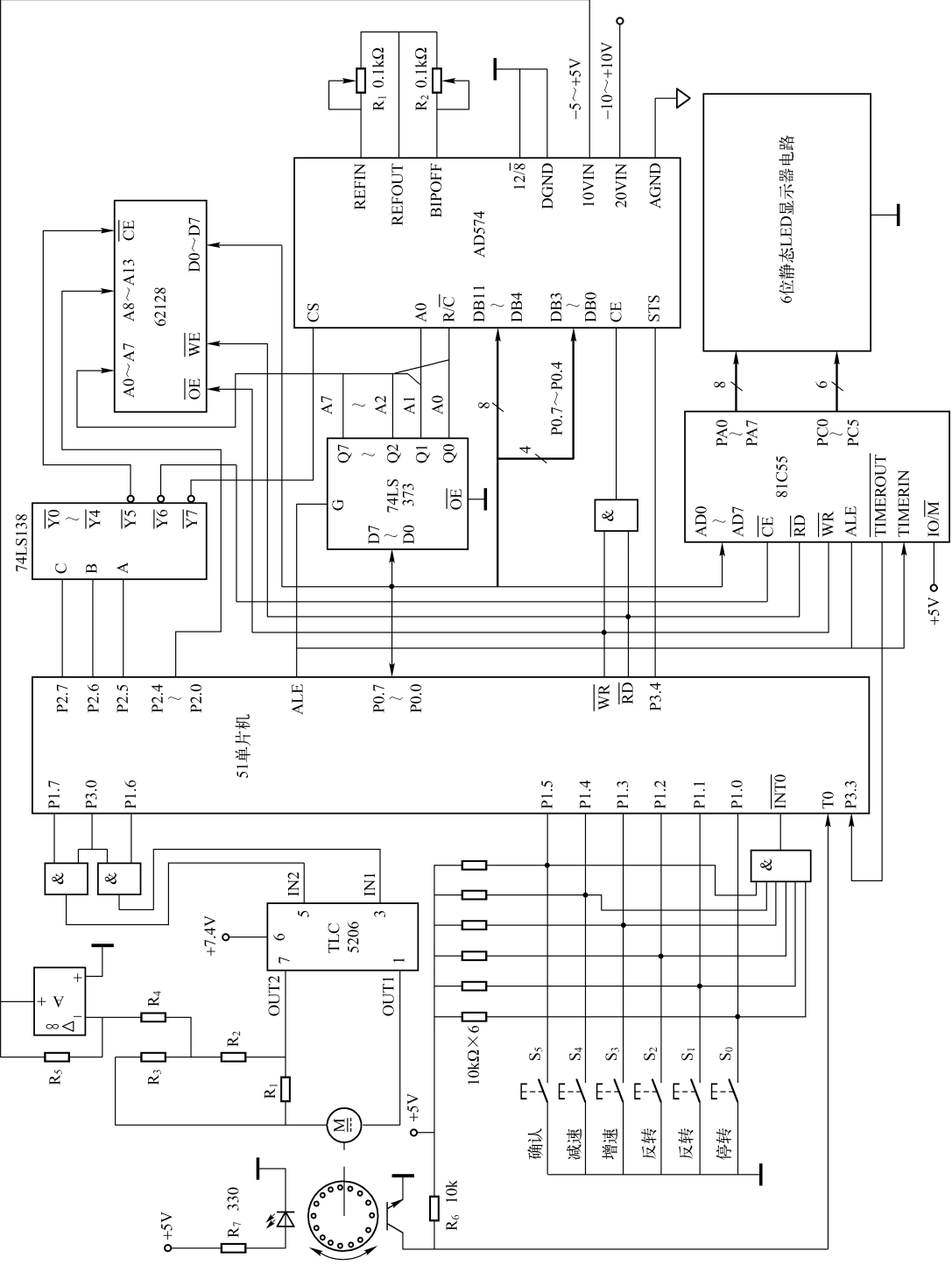


图 14-6 小型直流电动机的 51 单片机控制体系电路结构图

门的另一个输入端并联与 51 单片机的 P3.0 连接。P1.7、P1.6 的输出控制电动机的正反转和停止电动机转动, P3.0 输出脉冲宽度调制 (PWM) 信号, 控制电动机的转速。

TLE5206 的 V_s 端电压应远大于其工作电压, 图 14-6 中的 V_s 为 7.4 V, 而且应将 TLE5206 芯片的电源正极与单片机电源正极分开做滤波处理, 防止驱动电动机时, 电压波动过大使单片机复位。

TLE5206 的输出驱动端 OUT1 与电动机的一个驱动输入端连接, 输出驱动端 OUT2 通过一个电流取样电阻 R_1 与电动机的另一个驱动输入端连接。

(2) 电动机电流和转速检测电路

电动机电流检测的目的是为了实时监测显示电动机的电流, 当电动机电流超过 5 A 时使电动机停机, 起到保护功能。

电动机电流检测电路由取样电阻 R_1 , 分压电路 R_2 和 R_3 , 运算放大器 A 和 R_4 、 R_5 组成的信号放大器, 以及 12 位 A-D 转换器 AD574 组成。设电动机电流为 ± 5 A 时, 从 R_1 两端得到的取样电压为 ± 0.5 V, 确定好分压比, 信号放大器的放大倍数, 使信号放大器在电动机电流为 ± 5 A 时输出电压范围为 $-9 \sim +9$ V 或 $-4.5 \sim +4.5$ V, 运算放大器 A 的输出端与 AD574 的 $-10 \sim +10$ V 或 $-5 \sim +5$ V 输入端连接。图 14-6 中, 运算放大器 A 的输出端与 AD574 的 $-10 \sim +10$ V 输入端连接。

电动机转速检测电路的目之一是实时监测显示电动机的转速, 二是获得电动机转速闭环控制的转速反馈信号。转速检测可以选择非接触式的红外光电传感器, 在电动机转轴上安装一个转盘, 沿着转盘边缘打 16 个小圆孔, 红外对管对着小孔时, 图 14-6 中的三极管输出低电平, 红外发光二极管未对着小孔时, 三极管输出高电平, 这样转盘旋转一圈发出 16 个脉冲, 设 M 为脉冲个数/秒, 则转速计算公式为 $n = \frac{M}{16}$ 。如果电动机转速为 1000 rad/s, 则单片机每秒接收到 16000 个脉冲。

(3) 直流电动机转速控制系统硬件电路设计

单片机应用系统硬件电路设计必须与软件相结合, 根据直流电动机转速控制系统设计的要求, 以及电动机驱动电路和电动机电流和转速检测电路的结构, 首先所需要的定时、计数和外部中断的需求如下:

- 1) 按键的识别与处理: 参照第 11 章例 11-3, 需要一个外部中断, 以及滤波定时。
- 2) 电动机转速的计数输入: 作为转速反馈信号, 需要定时读取计数值, 所以需要一个计数器和一个定时器。
- 3) 电动机电流的定时采样: 需要一个定时器。
- 4) 电动机调速的脉冲宽度调制信号输出: 需要一个定时器。

很明显由于 51 单片机只有 T0 和 T1, 不能够满足设计的需要。解决的方法可以利用 81C55 的定时/计数器, 采取定时 + 软件计数的方式, 即按键的识别与处理、电动机转速计数值的读取和电动机电流的定时采样都使用 81C55 的定时/计数器: 电动机转速的计数输入使用 T0 计数; 电动机调速的脉冲宽度调制信号输出应用 T1。设计的小型直流电动机的 51 单片机控制系统电路结构图如图 14-6 所示。

从图 14-6 可以看出, 电动机的单片机控制系统电路是在图 14-3 的综合应用电路图基础之上进行的设计, 图 14-6 中将 ADC0809 置换为 12 位 A-D 转换器 AD574, DAC0832 用

RAM61128 替换,对键盘电路进行了修改,增加了电动机控制和电动机电流的采样电路。

3. 系统分析

软件设计必须按照设计要求和图 14-6 的电路结构进行设计,并对系统的运行状态、停机等进行分析。

(1) 确定 51 单片机扩展外部部件的地址

1) RAM61128 的地址范围: A000H ~ BFFFH。

2) AD574: 启动 A-D 转换地址: FFFCH; 读取结果高 8 位的地址: FFFE_H; 读取结果低 4 位的地址: FFFF_H; A-D 转换是否结束: 查询 P3.1 的状态,为“0”转换结束。

3) 81C55: 命令寄存器: DFF8H; A 口(段控口): DFF9H; C 口(位控口): DFFBH; 定时/计数器低 8 位地址: DFFCH; 定时/计数器高 6 位地址: DFFDH。

(2) 确定小型直流电动机的转速控制系统开机的初始状态

系统开机和复位时,电动机处于停机状态。

1) 51 单片机内部部件的初始状态: 启动 T0、T1 工作,允许 T1 和 $\overline{\text{INT0}}$ 中断。

2) 外部部件的初始状态: AD574 停止数据采集; 启动 81C55 的定时/计数器工作; LED 显示器最低位显示停机标志:“-”,最高位显示:“┐”,默认为正转停机状态。

3) 系统复位时,不影响电动机正反转的转速设置值。

(3) 小型直流电动机的转速控制系统的运行过程分析

图 14-6 中共有 6 个按键 $S_0 \sim S_5$,可以分为 2 组, $S_0 \sim S_2$ 为电动机运行控制按键, $S_3 \sim S_5$ 为正反转的转速设置按键。

1) 电动机正反转控制。按下正转 S_1 按键,设置 20 H 位单元为“1”状态,电动机正转;按下反转 S_2 按键,设置 20 H 位单元为“0”状态,电动机反转。

2) 停机控制。按下停转 S_0 按键,电动机停机。停机 LED 显示器的最高位显示:“┐”,低 5 位处于全灭的状态。

3) 电动机在正转时,按一次增速 S_3 按键,电动机转速增加 10 rad/s;按一次减速 S_4 按键,电动机转速减少 10 rad/s;按下一次确认 S_5 按键,增速或减速的操作才有效。电动机在反转时, S_3 、 S_4 和 S_5 按键功能相同。

4) LED 数据显示。LED 显示器的最高位显示的是电动机正反转标志:“┐”表示正转参数,“-”表示反转参数。电动机运行时,间隔 1 s 交替显示电动机的转速和电流;转速设置操作时,显示转速设置值,按下确认 S_5 按键,显示 1 s 时间的转速确认值,再恢复正常的显示过程。

停机状态可以进行转速增减的设置,LED 显示器的最高位显示“┐”时,进行正转的转速设置;显示“-”时,进行反转的转速设置。按下 S_0 按键进行正反转设置的切换。

图 14-6 的小型直流电动机的转速控制系统是一个转速闭环控制系统,其控制框图如图 14-7 所示,给定值 r 是转速设置值,反馈信号 f 是转换 T0 计数值的转速值,81C55 的定时/计数器的定时时间 T 到,读取 T0 的计数值并转换为转速值,给定值 r 与反馈值 f 进行比较,根据比较值计算设置 T1 的占空比,通过 P3.0 输出脉冲宽度调制信号 u ,TLC5206 控制电动机的运行。

1) 设 51 单片机的晶振时钟频率采用 12 MHz,81C55 的定时/计数器的计数脉冲来自于 51 单片机的 ALE 引脚输出的 1 MHz 脉冲,设置采样周期 $T = 100 \text{ ms}$,定时时间设置为采样周期 T 的 1/10,即定时时间 = 10 ms,在主程序中不断检测 P3.3 的状态,判断 10 ms 定时时间是否已到,并应用软件计数 10 次,满足 100 ms 采样周期 T 的定时。

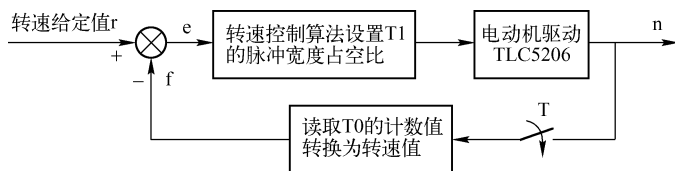


图 14-7 小型直流电动机转速控制框图

- 2) 时间 T 到，读取 T0 的计数值，先低 8 位，后高 8 位，计算转换为转速值反馈值 f。
- 3) 计算 $e = r - f$ ，根据 f 值作控制算法的运算，设置 T1 的脉冲宽度占空比。
- 4) 设脉冲宽度调制信号的周期 T1 为 100 ms，定时器 T1 工作于定时模式工作方式 2，允许中断，设置定时时间 100 μ s，应用软件计数方式确定 P3.0 端口输出高低电平的占空比。
- 5) 每个采样周期 T 启动 AD574 对电动机电流采样，进行 A - D 转换一次。
- 6) 在电动机正常运行过程中，每 1 s 刷新一次 LED 显示内容，转速、电流交替显示。
- 7) 当有按键操作时，按照按键的功能刷新一次 LED 显示内容。
- 8) 按键的滤波。由于没有定时器专门用于按键的滤波，只能够借助于 81C55 的定时/计数器的 10 ms 定时时间进行滤波，但按键的按下是随机的，所以设置滤波时间为 10 ~ 20 ms。

4. 程序流程图设计

首先将必备的字节单元和位单元与数据、标志之间的关系列出现表 14-1。

表 14-1 字节单元和位单元应用情况一览表

字节单元	应用说明	位单元	应用说明
30H	100 ms 计数单元	20H	1: 电动机运行; 0: 停机
31H	1 s 计数单元	21H	1: 正转; 0: 反转
33H, 32H	高电平计数个数, 最大值 1000, 单位: 100 μ s	22H	1 s 显示标志, 1: 转速; 0: 电流
35H, 34H	100 μ s 的计数单元, 最大 1000	23H	转速增标志
37H, 36H	存放 1000 = 3E8H, 调制周期	24H	转速减标志
39H, 38H	存放读取 T0 的计数值	25H	1: 正转设置 0: 反转设置
3BH, 3AH	转速值 rad/min, 即反馈信号 f	26H	转速设置标志
3DH, 3CH	电动机电流	27H	正转转速设置标志
45H ~ 40H	待显示数据存储单元	28H	反转转速设置标志
44H ~ 40H	显示代码数据	29H	设置值确认标志
46H, 45H	二进制待显示数据单元	30H	1: 外部中断 0 标志
48H, 47H	正转转速设置值	31H	1: 按键滤波标志
4AH, 49H	反转转速设置值		
4CH, 4BH	正转转速设置暂存值		
4EH, 4DH	反转转速设置暂存值		
51H	KCS 子程序读按键结果单元		

在设计程序流程图、编写程序和调试修改程序过程中不断增加和完善表 14-1 的内容。

(1) 主程序模块

主程序模块由初始化程序和无限循环程序组成，主程序流程图如图 14-8 所示。

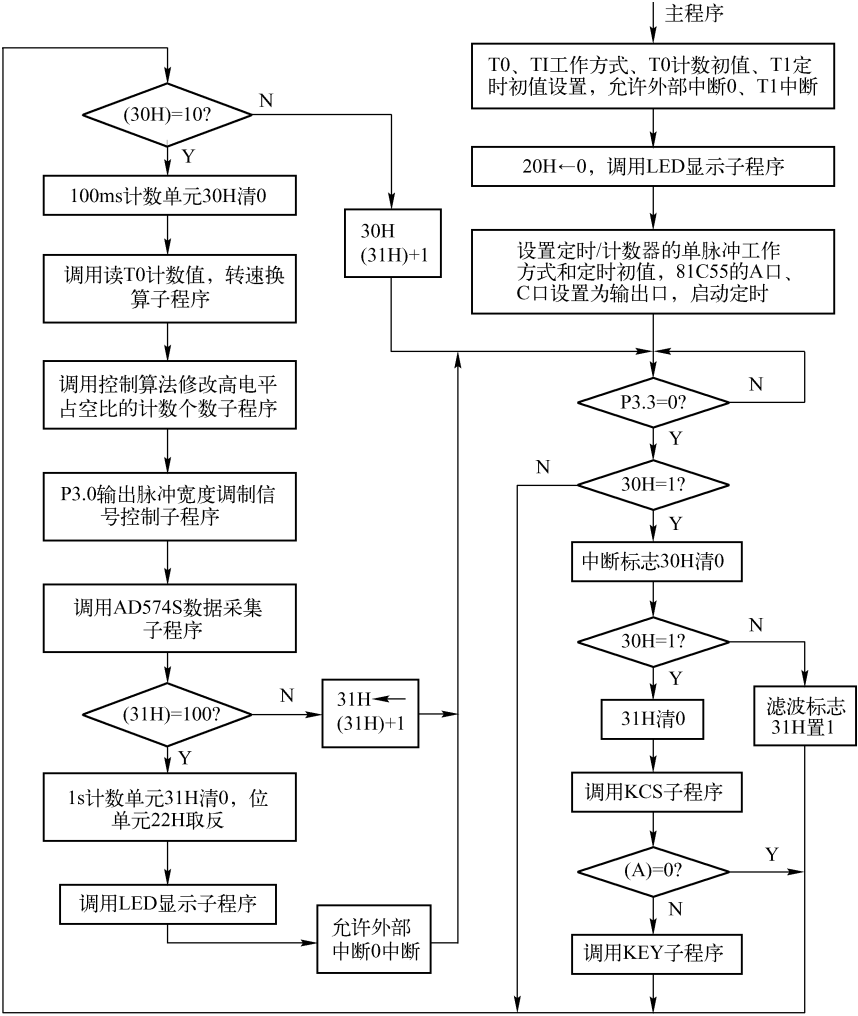


图 14-8 主程序流程图

无限循环程序由判断 10 ms 定时是否到和判断采样周期 T 的 100 ms 定时是否到两部分组成。判断 10 ms 定时到时，完成键盘功能的处理；判断采样周期 T 的 100 ms 定时到时，完成对电动机运行控制的功能处理。

(2) 读 T0 计数值，转速换算子程序

每个周期 T 读取一次 T0 的计数值：(TL0)→38H，(TH0)→39H，再对 T0 设置一次计数初值 00H。计算 $M = (39H)(38H) \times 10$ ，将 M 存放在 3BH 和 3AH 中，再按照公式 $n = \frac{M}{16}$ ，将 (39H)(38H) 右移 4 位即得到转速值，也就是图 14-7 的反馈信号 f。读者可以自己设计该

子程序流程图。

(3) 控制算法修改高电平占空比的计数个数子程序

控制算法有多种选择, 可以选择 PID 算法, 也可以采用智能控制算法, 该子程序的最终结果为高电平占空比计数个数值, 高电平占空比计数个数值 $\times 100 \mu\text{s}$ = 高电平持续时间, 即脉冲宽度调制信号 100 ms 周期 T1 中的高电平持续时间。

(4) P3.0 输出脉冲宽度调制信号控制子程序

P3.0 输出脉冲宽度调制信号控制子程序流程图如图 14-9 所示。

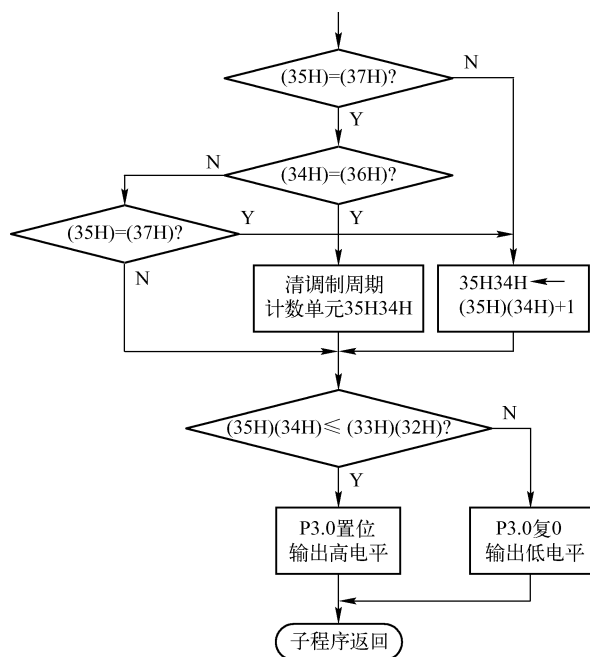


图 14-9 脉冲宽度调制信号输出控制子程序流程图

(5) AD574 的数据采集子程序

参阅第 10 章例 10-5 的子程序, 但要注意修改 AD574 的地址、转换结束信号的位单元地址, 以及采集转换的值需要存放在 3DH、3CH 中。在例 10-5 的子程序 RET 指令之前, 还需要增加将 A-D 转换的双字节结果转换为双字节的 BCD 码程序。

(6) LED 显示子程序

参阅第 11 章例 11-8 的子程序, 但要注意修改 81C55 的地址, 以及该子程序 45H ~ 40H 单元存放的是已编辑好的待显示数据, 即每个单元高 4 位都为 0, 应用低 4 位数据查显示段码表。所以需要在例 11-8 的子程序前面增加二进制待显示数据的一段程序。根据设计要求, 图 14-6 的 LED 显示器的共阳段码表见表 14-2。

由于显示状态还受到按键操作的控制, 在例 11-3 的子程序基础之上还需要增加对按键处理结果的判断处理程序。LED 显示子程序的程序流程图如图 14-10 所示。

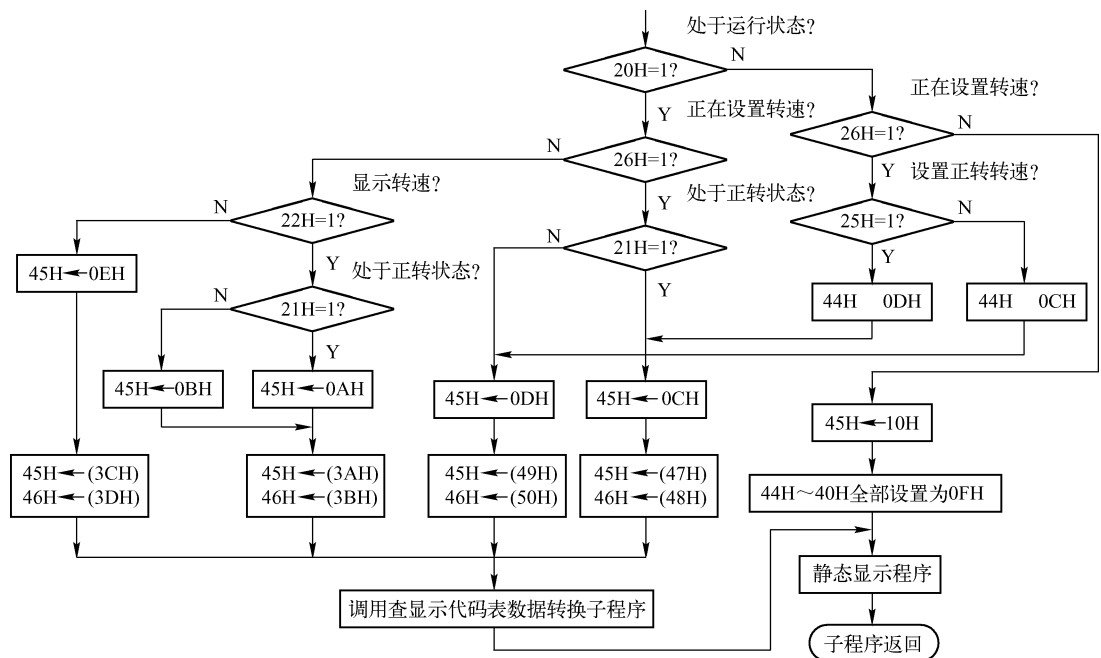


图 14-10 LED 显示子程序流程图

表 14-2 LED 数码管共阳段码表

显示字型	段码	显示字型	段码	显示字型	段码	查表数据
0	C0H	5	92H	└	9FH	0AH
1	F9H	6	82H	—	BFH	0BH
2	A4H	7	F8H	┌	8FH	0CH
3	B0H	8	80H	├	B9H	0DH
4	99H	9	90H	┘	BCH	0EH
				灭	FFH	0FH
				三	C6H	10H

注：└：正转；—：反转；┌：正转转速设置；├：反转转速设置；┘：电流。

(7) 按键处理程序

按键处理程序包括外部中断 0 的中断服务程序、按键滤波处理子程序 KCS（参阅第 11 章例 11-1）以及键识别与键处理子程序 KEY（参阅第 11 章例 11-2）。在键识别与键处理子程序 KEY 中，调用有效按下键的功能处理子程序。

1) 外部中断 0 的中断服务程序 INT00。参阅第 11 章例 11-3 的外部中断 0 的中断服务程序 INT00，但根据本系统的设计思路，INT00 中只需要关外部中断 0，设置已响应外部中断 0 的中断标志，即位单元 30H 置 1。

2) 按键滤波处理子程序 KCS。图 14-8 主程序流程图中, 位单元 30H 为 1 时, 调用按键滤波处理子程序 KCS, 但对第 11 章例 11-1 的 KCS 程序需要按照本系统的要求进行修改, 将 30H 单元改为 51H (见表 14-1), ANL A, #0FH 改为 ANL A, #3FH, 这是因为本系统有 6 个按键, 而例 11-1 只有 4 个按键。

3) 键识别与键处理子程序 KEY。图 14-8 主程序流程图中, 判断累加器 A 中的内容是否为 0, 不为 0 表示有按键是有效的按下, 调用键识别与键处理子程序 KEY。因为本系统有 6 个按键, 而例 11-2 只对 4 个按键的处理, 所以需要在例 11-2 的子程序 KEY 再增加 2 个按键判断处理程序, 即

```

      :
KEY3: JNB  ACC. 3, KEY4      ;P1. 3 = 1 是 S3 号键
      ACALL FUN3             ;调用 S3 号键子程序
KEY4: JNB  ACC. 4, KEY5      ;P1. 4 = 1 是 S4 号键
      ACALL FUN4             ;调用 S4 号键子程序
KEY5: JNB  ACC. 5, RETURN    ;P1. 5 = 1 是 S5 号键
      ACALL FUN5             ;调用 S5 号键子程序
RETURN: RET

```

本系统共有 6 个按键 $S_0 \sim S_5$, $S_0 \sim S_2$ 为电动机运行控制按键, $S_3 \sim S_5$ 为正反转的转速设置按键。设计时需要对每个按键的功能编写一个功能处理子程序。

① 停转 S_0 号键功能子程序 FUN0。根据本节前面的描述、图 14-6 和表 14-1, S_0 号键子程序 FUN0 的功能很简单: 设置停机/运行标志位单元 20H 为 “0” 状态, 设置 P1.6 和 P1.7 输出都为低电平, 使电动机停转。

② 正转 S_1 号键功能子程序 FUN1。根据本节前面的描述、图 14-6 和表 14-1, S_1 号键子程序 FUN0 的功能: 设置停机/运行标志位单元 20H 为 “1” 状态, 设置正转/反转标志位单元 21H 为 “1” 状态, 设置 P1.6 为 1, P1.7 为 0, 则 P3.0 输出的脉冲宽度调制信号施加在 TLC5206 的 IN1 引脚, 使电动机正转。

③ 反转 S_2 号键功能子程序 FUN2。 S_2 号键子程序 FUN2 的功能与 S_1 号键子程序 FUN0 相反: 设置停机/运行标志位单元 20H 为 “1” 状态, 设置正转/反转标志位单元 21H 为 “0” 状态, 设置 P1.6 为 0, P1.7 为 1, 则 P3.0 输出的脉冲宽度调制信号施加在 TLC5206 的 IN2 引脚, 使电动机反转。

④ 增速 S_3 号键功能子程序 FUN3。根据本节前面的描述、图 14-6 和表 14-1, S_3 号键子程序 FUN3 的功能: 设置增速标志和转速设置标志, 清减速标志, 判断是正转转速增速还是反转转速增速, 执行一次 FUN3 子程序, 正转或反转的转速设置值加 10 (0AH), 设置值就是图 14-7 的转速给定值 r 。子程序 FUN3 的程序流程图如图 14-11 所示。

⑤ 减速 S_4 号键功能子程序 FUN4。 S_4 号键子程序 FUN4 的功能与 S_3 号键子程序 FUN3 的功能类似: 设置减速标志和转速设置标志, 清增速标志, 判断是正转转速减速还是反转转速减速, 执行一次 FUN4 子程序, 正转或反转的转速设置值减 10 (0AH)。设置值就是图 14-7 的转速给定值 r 。子程序 FUN4 的程序流程图如图 14-12 所示。

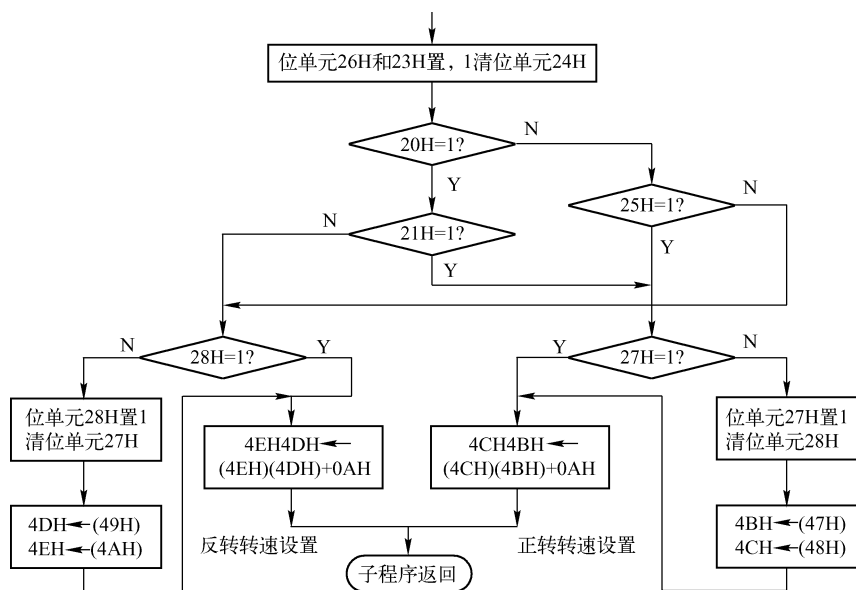


图 14-11 增速 S_3 号键功能 FUN3 子程序流程图

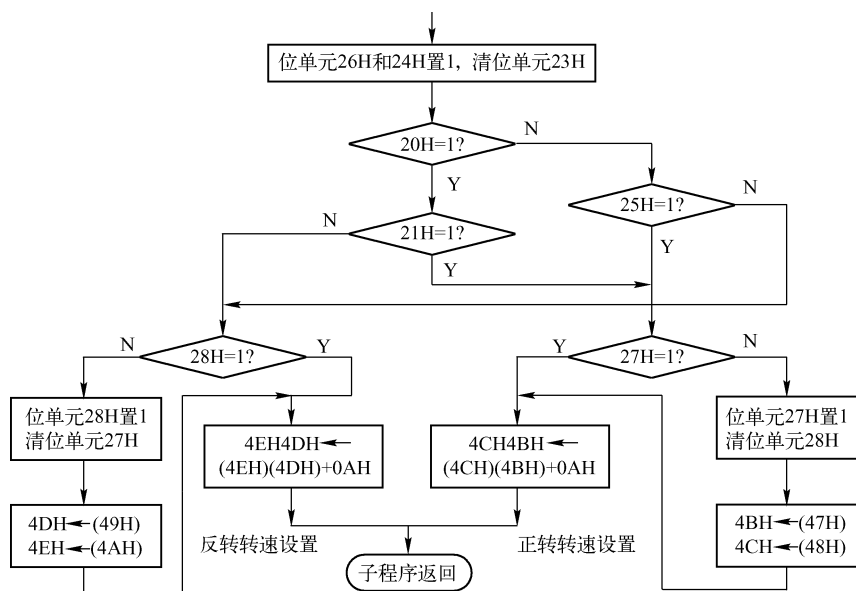


图 14-12 减速 S_4 号键功能 FUN4 子程序流程图

⑥ 设置确认 S_5 号键功能子程序 FUN5。确认键 S_5 号键有两个功能，一是在电动机转速设置状态时，根据正转还是反转转速设置的标志，将设置的转速暂存值存放在正转或反转速度设置值的存储单元中，确认的设置值就是图 14-7 的转速给定值 r ；二是在非电动机转速设置状态时，取反设置正转/反转转速的标志，即实现下次进入电动机转速设置状态时，由正转速度设置转换为反转转速设置状态，或由反转速度设置转换为正转转速设置状态。子程序 FUN5 的程序流程图如图 14-13 所示。

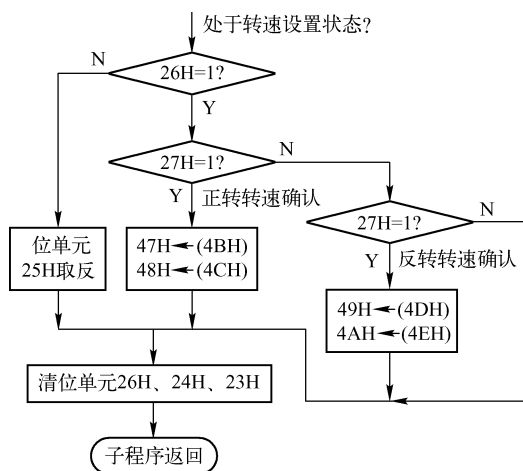


图 14-13 确认 S₄ 号键功能 FUN5 子程序流程图

14.4.3 单片机应用系统串行总线接口芯片的扩展应用系统设计

1. 设计要求

设计一个电子时钟，使用单片机 I/O 口模拟串行总线 I²C 和 SPI 通信协议，分别驱动时钟芯片 DS3231 和 OLED 显示模块。要求能在 OLED 显示屏上显示出年、月、日、时、分、秒和星期，并且能通过按键对年、月、日、时、分、秒和星期进行调节。

2. 电路设计

根据设计要求，设计的电路如图 14-14 所示。图 14-14 中 U1 是时钟芯片 DS3231，51 单片机通过 I/O 口模拟 I²C 通信方式对 DS3231 进行控制和数据读取；U3 是 SPI 方式的 OLED 显示屏，51 单片机通过 I/O 口模拟 SPI 通信协议即可驱动屏幕显示内容。

DS3231 是低成本、高精度 I²C 实时时钟，具有集成的温补晶体振荡器和晶体。该器件包含电池输入端，断开主电源时仍可保持精确的计时。集成晶体振荡器提高了器件的长期准确度，并减少了产品的元件数量。DS3231 能够保持年、月、日、时、分、秒和星期信息。少于 31 天的月份，将自动调整月末日期，包括闰年补偿。时钟的工作格式可以是 24 小时或者带 AM/PM 指示的 12 小时格式。提供两个可编程日历闹钟和一路可编程方波输出。地址与数据通过 I²C 双向串行传输。图 14-14 中，SCL 为 DS3231 芯片 I²C 的时钟引脚，SDA 为数据端口，SCL 和 SDA51 分别与 51 单片机的 P1.0 和 P1.1 引脚连接，可以对 DS3231 芯片进行控制和数据读取。VBAT 为备用电源接口，图中 BT1 为纽扣电池，电池通过二极管隔离连接到时钟芯片。当时钟芯片的主电源断开时，自动切换到备用电池，用以保持芯片内部时钟的继续运行。直到下次重新接上主电源之后，时钟即可继续运行，而不需要重新调整时钟。

DS3231 的 PIN4 引脚为中断输出端口，可以通过配置，当秒钟时间到之后，自动输出一个低电平脉冲。图 14-14 中将 PIN4 引脚连接到 51 单片机的外部中断 0 信号输入引脚 P3.2，编写外部中断 0 的中断服务程序，在外部中断 0 的中断服务程序中读取时钟芯片内部的时间信息，并且驱动显示屏刷新时间。DS3231 操作时序图如图 14-15 所示。

本设计使用的显示模块为 SPI 接口的 OLED 显示屏，2.42 寸的显示屏，分辨率为 128 × 64，有蓝、黄、白和绿色四种颜色供选择，OLED 显示屏如图 14-16 所示。屏幕内部驱动 IC 为 SSD1309，对外接口为 SPI 模式。单片机应用 SPI 协议对显示屏进行控制。

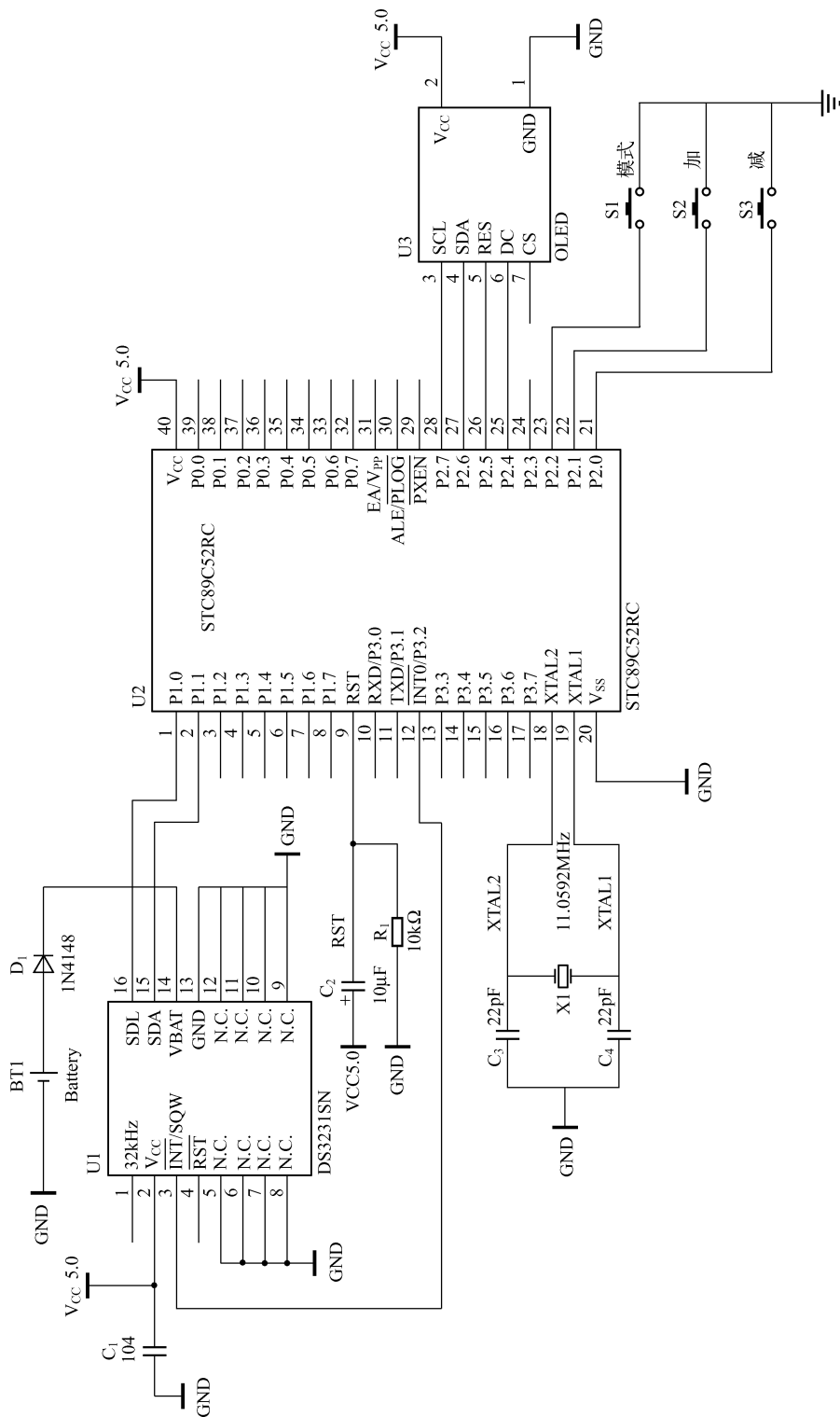


图14-14 应用系统串行总线接口的电子时钟电路结构图

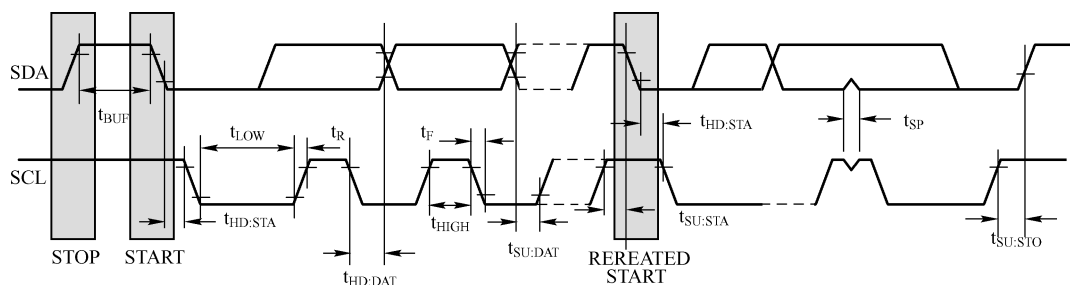


图 14-15 DS3231 操作时序图

显示屏 SPI 接口定义：

- 1) 引脚 1：GND（电源地）。
- 2) 引脚 2：VCC（电源正）。
- 3) 引脚 3：SCL（时钟线）。
- 4) 引脚 4：SDA（数据线）。
- 5) 引脚 5：RES（复位线）。
- 6) 引脚 6：DC（数据/命令）。
- 7) 引脚 7：CS（片选）。

3. 软件设计

软件设计主要涉及两部分，即时钟数据读取、显示和时钟数据调节。时钟数据的读取主要是通过 I²C 对 DS3231 的操作。时间的读取是时钟芯片 DS3231 发出秒中断信号，51 单片机响应中断后执行 INT0 中断服务程序读取时间数据，然后刷新时间显示，主程序流程图如图 14-17 所示，时钟芯片 DS3231 的中断服务程序的流程图如图 14-18 所示。

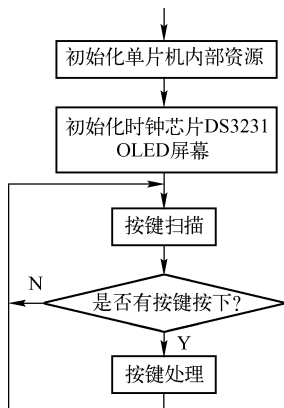


图 14-17 主程序流程图

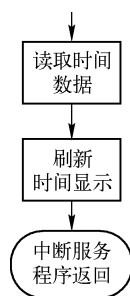


图 14-18 中断服务程序流程图

在时间调节部分，需要进行操作的内容相对较多，涉及年、月、日、星期、时、分、秒的数据调整。本例电路设计中，设置了 3 个按键，分别是“模式”“加”和“减”。“模式”键用于切换当前模式，“加”和“减”两个按键用于在不同的模式状态下，对数据进行“加 1”或“减 1”操作。此部分编程的中心思想为“状态机”机制，将程序的运行分为多个状态，不同的状态下，按键的功能各有不同，处理的事件也不同。而状态的切换，由“模式”键进行操作。状态机的编程思想，可以将复杂的事件分成多个简单的小事件分别处理，使用此编程方式能大大地提高程序的处理复杂事件的编程效率，程序思路清晰，易读，易修改，易维护。按键处理程序流程图如图 14-19 所示。

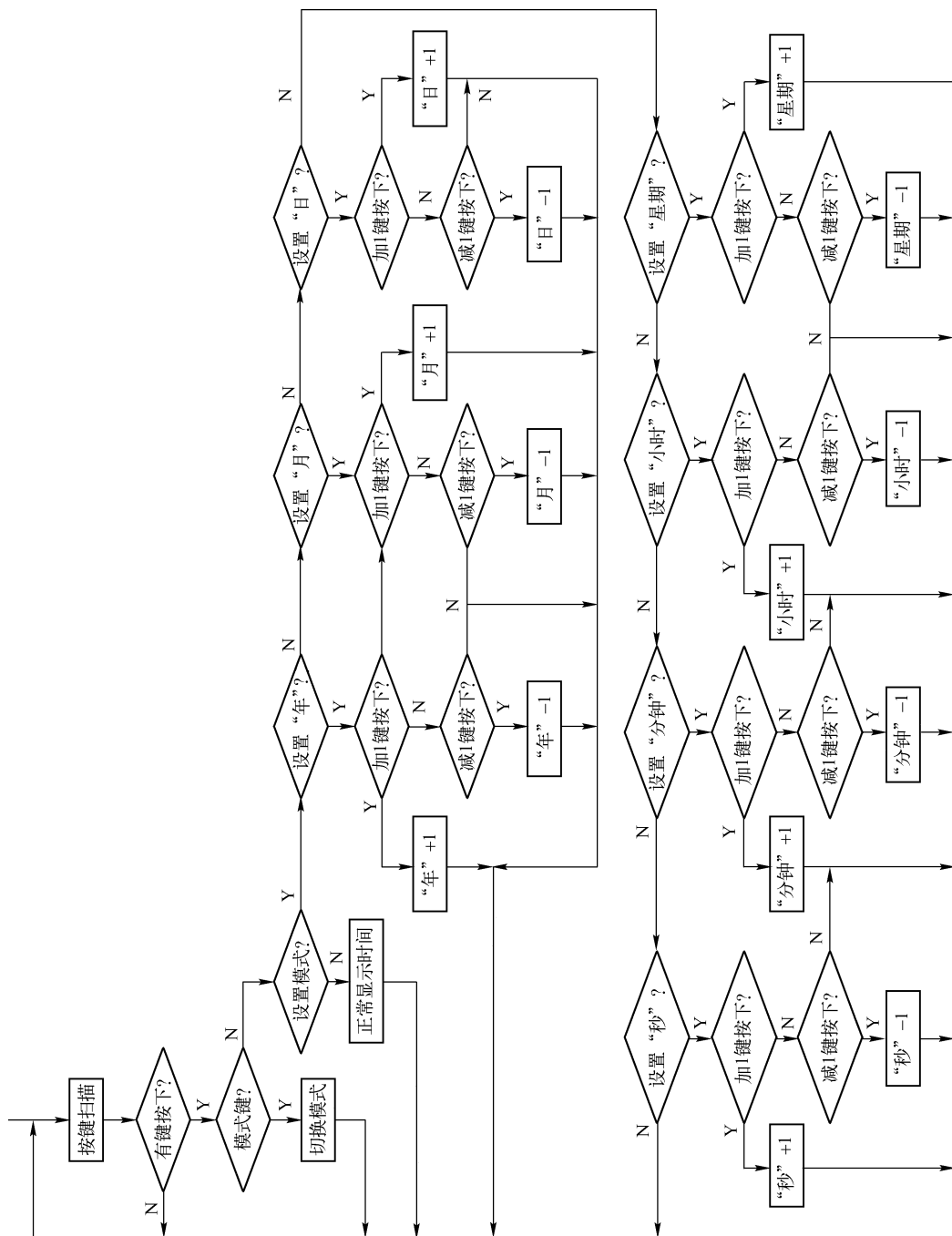


图14-19 按键处理流程图

附录

附录 A 51 单片机指令集

表 A-1 MCS51 指令系统所用符号和含义

符 号	含 义
addr11	11 位地址
addr16	16 位地址
bit	位地址（位单元；8 位地址）
rel	相对偏移量，8 位符号数（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，n=0，1，2，3，4，5，6，7
A	累加器
Ri	i=0，1。指数数据指针 R0 或 R1
X	片内 RAM 中的直接地址或寄存器
@	间接寻址方式中，表示寄存器间接寻址的符号
(X)	表示存储单元 X 中的内容
((X))	表示以存储单元 X 中的内容作为地址，该地址单元的内容
→	数据传送方向
∧	逻辑与
∨	逻辑或
⊕	逻辑异或
/	在位地址之前，表示该位状态取反参与操作，位单元本身状态不变，用于位操作指令
✓	对标志产生影响
×	不影响标志

表 A-2 51 单片机指令集

十六进制 代码	助 记 符	功 能	对标志影响				字 节 数	周 期 数
			P	OV	AC	CY		
数据传送类指令								
E8H ~ EFH	MOV A, Rn	(Rn)→A	✓	×	×	×	1	1
E5H, direct	MOV A, direct	(direct)→A	✓	×	×	×	2	1
E6H ~ E7H	MOV A, @ Ri	(((Ri))→A	✓	×	×	×	1	1
74H, data	MOV A, #data	data→A	✓	×	×	×	2	1
F8H ~ FFH	MOV Rn, A	(A)→Rn	×	×	×	×	1	1
A8H ~ AFH, direct	MOV Rn, direct	(direct)→Rn	×	×	×	×	2	2
78H ~ 7FH, data	MOV Rn, #data	data→Rn	×	×	×	×	2	1
F5H, direct	MOV direct, A	(A) →direct	×	×	×	×	2	1
88H ~ 8FH, direct	MOV direct, Rn	(Rn) →direct	×	×	×	×	2	2
85H, direct2, direct1	MOV direct1, direct2	(direct2) →direct1	×	×	×	×	3	2

(续)

十六进制 代码	助记符	功 能	对标志影响				字 节 数	周 期 数
			P	OV	AC	CY		
86H ~ 87H, direct	MOV direct, @ Ri	((Ri)) → direct	×	×	×	×	2	2
75H, direct, data	MOV direct, #data	data → (direct)	×	×	×	×	3	2
F6H ~ F7H	MOV @ Ri, A	A → (Ri)	×	×	×	×	1	1
A6H ~ A7H, direct	MOV @ Ri, direct	direct → (Ri)	×	×	×	×	2	2
76H ~ 77H, data	MOV @ Ri, #data	data → (Ri)	×	×	×	×	2	1
90H, data16	MOV DPTR, #data16	data16 → DPTR	×	×	×	×	3	2
C0H, direct	PUSH direct	(SP) + 1 → SP, (direct) → (SP)	×	×	×	×	2	2
D0H, direct	POP direct	((SP)) → direct, (SP) - 1 → SP	×	×	×	×	2	2
C8H ~ CFH	XCH A, Rn	(A) ↔ (Rn)	✓	×	×	×	1	1
C5H, direct	XCH A, direct	(A) ↔ (direct)	✓	×	×	×	2	1
C6H ~ C7H	XCH A, @ Ri	(A) ↔ ((Ri))	✓	×	×	×	1	1
D6H ~ D7H	XCHD A, @ Ri	(A [D3 ~ D0]) ↔ ((Ri) [D3 ~ D0])	✓	×	×	×	1	1
C4H	SWAP A	(A [D7 ~ D4]) ↔ (A [D3 ~ D0])	×	×	×	×	1	1
E2H ~ E3H	MOVX A, @ Ri	((Ri)) → A	✓	×	×	×	1	2
E0H	MOVX A, @ DPTR	((DPTR)) → A	✓	×	×	×	1	2
F2H ~ F3H	MOVX @ Ri, A	(A) → (Ri)	×	×	×	×	1	2
F0H	MOVX @ DPTR, A	(A) → (DPTR)	×	×	×	×	1	2
93H	MOVC A, @ A + DPTR	((A + DPTR)) → A	✓	×	×	×	1	2
83H	MOVC A, @ A + PC	(PC) + 1 → PC, ((A) + (PC)) → A	✓	×	×	×	1	2

算术运算类指令

28H ~ 2FH	ADD A, Rn	(A) + (Rn) → A	✓	✓	✓	✓	1	1
25H, direct	ADD A, direct	(A) + (direct) → A	✓	✓	✓	✓	2	1
26H ~ 27H	ADD A, @ Ri	(A) + ((Ri)) → A	✓	✓	✓	✓	1	1
24H, data	ADD A, #data	(A) + data → A	✓	✓	✓	✓	2	1
38H ~ 3FH	ADDC A, Rn	(A) + (Rn) + Cy → A	✓	✓	✓	✓	1	1
35H, direct	ADDC A, direct	(A) + (direct) + Cy → A	✓	✓	✓	✓	2	1
36H, 37H	ADDC A, @ Ri	(A) + ((Ri)) + Cy → A	✓	✓	✓	✓	1	1
34H, data	ADDC A, #data	(A) + data + Cy → A	✓	✓	✓	✓	2	1
04H	INC A	(A) + 1 → A	✓	×	×	×	1	1
08H ~ 0FH	INC Rn	(Rn) + 1 → Rn	×	×	×	×	1	1
05H, direct	INC direct	(direct) + 1 → direct	×	×	×	×	2	1
06H ~ 07H	INC @ Ri	((Ri)) + 1 → (Ri)	×	×	×	×	1	1
A3H	INC DPTR	(DPTR) + 1 → DPTR	×	×	×	×	1	2
D4H	DA A	对 A 进行十进制调整	✓	✓	✓	✓	1	1
98H ~ 9FH	SUBB A, Rn	(A) - (Rn) - Cy → A	✓	✓	✓	✓	1	1
95H, direct	SUBB A, direct	(A) - (direct) - Cy → A	✓	✓	✓	✓	2	1
96H, 97H	SUBB A, @ Ri	(A) - ((Ri)) - Cy → A	✓	✓	✓	✓	1	1
94H, data	SUBB A, #data	(A) - data - Cy → A	✓	✓	✓	✓	2	1
14H	DEC A	(A) - 1 → A	✓	×	×	×	1	1
18H ~ 1FH	DEC Rn	(Rn) - 1 → Rn	×	×	×	×	1	1
15H, direct	DEC direct	(direct) - 1 → (direct)	×	×	×	×	2	1
16H, 17H	DEC @ Ri	((Ri)) - 1 → (Ri)	×	×	×	×	1	1

(续)

十六进制 代码	助记符	功 能	对标志影响				字 节 数	周 期 数
			P	OV	AC	CY		
A4H	MUL AB	$(A) \times (B)$, 结果低 8 位 $\rightarrow A$, 高 8 位 $\rightarrow B$	✓	✓	×	0	1	4
84H	DIV AB	$(A)/(B)$, 商的整数 $\rightarrow A$, 余数 $\rightarrow B$	✓	✓	×	0	1	4
逻辑运算类指令								
58H ~ 5FH	ANL A, Rn	$(A) \wedge (Rn) \rightarrow A$	✓	×	×	×	1	1
55H, direct	ANL A, direct	$(A) \wedge (\text{direct}) \rightarrow A$	✓	×	×	×	2	1
56H, 57H	ANL A, @ Ri	$(A) \wedge ((Ri)) \rightarrow A$	✓	×	×	×	1	1
54H, data	ANL A, #data	$(A) \wedge \text{data} \rightarrow A$	✓	×	×	×	2	1
52H, direct	ANL direct, A	$(\text{direct}) \wedge (A) \rightarrow \text{direct}$	×	×	×	×	2	1
53H, direct, data	ANL direct, #data	$(\text{direct}) \wedge \text{data} \rightarrow \text{direct}$	×	×	×	×	3	2
48H ~ 4FH	ORL A, Rn	$(A) \vee (Rn) \rightarrow A$	✓	×	×	×	1	1
45H, direct	ORL A, direct	$(A) \vee (\text{direct}) \rightarrow A$	✓	×	×	×	2	1
46H ~ 47H	ORL A, @ Ri	$(A) \vee ((Ri)) \rightarrow A$	✓	×	×	×	1	1
44H, data	ORL A, #data	$(A) \vee \text{data} \rightarrow A$	✓	×	×	×	2	1
42H, direct	ORL direct, A	$(\text{direct}) \vee (A) \rightarrow \text{direct}$	×	×	×	×	2	1
43H, direct, data	ORL direct, #data	$(\text{direct}) \vee \text{data} \rightarrow \text{direct}$	×	×	×	×	3	2
68H ~ 6FH	XRL A, Rn	$(A) \oplus (Rn) \rightarrow A$	✓	×	×	×	1	1
65H, direct	XRL A, direct	$(A) \oplus (\text{direct}) \rightarrow A$	✓	×	×	×	2	1
66H ~ 67H	XRL A, @ Ri	$(A) \oplus ((Ri)) \rightarrow A$	✓	×	×	×	1	1
64H, data	XRL A, #data	$(A) \oplus \text{data} \rightarrow A$	✓	×	×	×	2	1
62H, direct	XRL direct, A	$(\text{direct}) \oplus (A) \rightarrow \text{direct}$	×	×	×	×	2	1
63H, direct, data	XRL direct, #data	$(\text{direct}) \oplus \text{data} \rightarrow \text{direct}$	×	×	×	×	3	2
E4H	CLR A	$0 \rightarrow A$	✓	×	×	×	1	1
F4H	CPL A	$\neg(A) \rightarrow A$	×	×	×	×	1	1
23H	RL A	A 循环左移一位	×	×	×	×	1	1
03H	RR A	A 循环右移一位	×	×	×	×	1	1
33H	RLC A	A 带进位循环左移一位	✓	×	×	×	1	1
13H	RRC A	A 带进位循环右移一位	✓	×	×	×	1	1
控制转移类指令								
02H, addr (15 ~ 8) ^① addr (7 ~ 0)	LJMP addr16	无条件长转移, 64 KB 范围内的无条件跳转指令	×	×	×	×	3	2
a ₁₀ a ₉ a ₈ 00001 ^① , addr (7 ~ 0)	AJMP addr11	无条件绝对转移指令, 2 KB 范围内的无条件跳转指令	×	×	×	×	2	2
80H, rel	SJMP rel	相对转移, 转移范围: 相对 PC 当前值向前 128, 向后 127 字节	×	×	×	×	2	2
73H	JMP @ A + DPTR	$PC \leftarrow ((A) + (DPTR))$	×	×	×	×	1	2
60H, rel	JZ rel	若(A) = 0, 转移; 否则顺序执行	×	×	×	×	2	2
70H, rel	JNZ rel	若(A) ≠ 0, 转移; 否则顺序执行	×	×	×	×	2	2
B5H, direct, rel	CJNE A, direct, rel	若(A) ≠ (direct), 转移; 否则顺序执行; 若(A) < (direct), 则 1 $\rightarrow$ Cy; 若 A > (direct), 则 0 $\rightarrow$ Cy	×	×	×	✓	3	2

(续)

十六进制代码	助记符	功 能	对标志影响				字 节 数	周 期 数
			P	OV	AC	CY		
B4H, data, rel	CJNE A, #data, rel	若 (A) ≠ data, 转移; 否则顺序执行; 若 (A) < data, 则 1→Cy; 若 (A) > data, 则 0→Cy	×	×	×	√	3	2
B8H ~ BFH, data, rel	CJNE Rn, #data, rel	若 (Rn) ≠ data, 转移; 否则顺序执行; 若 (Rn) < data, 则 1→Cy; 若 (Rn) > data, 则 0→Cy	×	×	×	√	3	2
B6H ~ B7H	CJNE @ Ri, #da- ta, rel	若 ((Ri)) ≠ data, 转移; 否则顺序执行; 若 ((Ri)) < data, 则 1→Cy; 若 ((Ri)) > data, 则 0→Cy	×	×	×	√	3	2
D8H ~ DFH, rel	DJNZ Rn, rel	(Rn) - 1 → Rn, 若 (Rn) ≠ 0, 转移; 否 则顺序执行	×	×	×	×	2	2
D5H, direct, rel	DJNZ direct, rel	(direct) - 1 → direct, 若 (direct) ≠ 0, 转移; 否则顺序执行	×	×	×	×	3	2
12H, addr (15 ~ 8) ^① addr (7 ~ 0)	LCALL addr16	子程序长调用。下一条指令地址压入 堆栈, PC ← addr16, 能够调用 64KB 范围 内的任何子程序	×	×	×	×	3	2
a ₁₀ a ₉ a ₈ 10001 ^① , addr (7 ~ 0)	ACALL addr11	子程序绝对调用。下一条指令地址压 入堆栈, PC ← addr11, PC 的高 5 位不 变, 能够调用 2KB 范围内的任何子程序	×	×	×	×	2	2
22H	RET	子程序返回	×	×	×	×	1	2
32H	RETI	中断返回指令	×	×	×	×	1	2
00H	NOP	空操作	×	×	×	×	1	1
位操作类指令								
A2H, bit	MOV C, bit	bit → Cy	×	×	×	√	2	2
92H, bit	MOV bit, C	Cy → bit	×	×	×	×	2	2
C3H	CLR C	0 → Cy	×	×	×	√	1	1
C2H, bit	CLR bit	0 → bit	×	×	×	×	2	1
D3H	SETB C	1 → Cy	×	×	×	√	1	1
D2H, bit	SETB bit	1 → bit	×	×	×	×	2	1
82H, bit	ANL C, bit	Cy ∧ bit → Cy	×	×	×	√	2	2
B0H, bit	ANL C, /bit	Cy ∧ /bit → Cy	×	×	×	√	2	2
72H, bit	ORL C, bit	Cy ∨ bit → Cy	×	×	×	√	2	2
A0H, bit	ORL C, /bit	CY ∨ /bit → Cy	×	×	×	√	2	2
B3H	CPL C	/Cy → Cy	×	×	×	√	1	1
B2H, bit	CPL bit	/bit → bit	×	×	×	×	2	1
40H, rel	JC rel	若 Cy = 1, 转移; 否则顺序执行	×	×	×	×	2	2
50H, rel	JNC rel	若 Cy = 0, 转移; 否则顺序执行	×	×	×	×	2	2
20H, bit, rel	JB bit, rel	若 bit = 1, 转移; 否则顺序执行	×	×	×	×	3	2
30H, bit, rel	JNB bit, rel	若 bit = 0, 转移; 否则顺序执行	×	×	×	×	3	2
10H, bit, rel	JBC bit, rel	若 bit = 1, 转移, 0 → bit; 若 bit = 0, 顺序执行	×	×	×	×	3	2

^①其中 a₁₀a₉a₈...a₂a₁a₀ 是 addr11 的各位, addr (7 ~ 0) 表示 a₇ ~ a₀; addr (15 ~ 8) 表示 a₁₅ ~ a₈。

附录 B μ Vision2 集成开发环境使用

μ Vision2 集成开发环境 IDE 是一个基于 Windows 的软件开发平台，包含一个高效的编辑器、一个项目管理器和一个 MAKE 工具。

μ Vision2 支持所有的 8051 Keil 工具，包括 C 编译器、宏汇编器、连接器/定位器和目标文件至 Hex 格式的转换器。

μ Vision2 软件提供了各种操作菜单和快捷图标，比如编辑器操作、项目维护、开发工具选项设置、程序调试、窗口选择和操作以及在线帮助等。工具菜单栏可以快速执行 μ Vision2 命令，键盘快捷键也可以执行 μ Vision2 命令。

在 μ Vision2 集成开发环境下是使用工程项目的的方法来管理文件的，而不是单一文件的模式。所有文件包括源文件（C 语言程序、汇编程序）、头文件，甚至说明性的技术文档都可以放在工程项目文件中统一管理。

对于刚刚使用 Keil C51 的用户，一般可以按照以下步骤来创建一个自己的 Keil C51 应用程序：

- 1) 启动 μ Vision2，创建一个工程项目文件。
- 2) 为工程项目选择目标器件（如选择 ATMEL AT89C51）。
- 3) 为工程项目设置软/硬件调试环境。
- 4) 创建源程序文件，并输入源程序代码。
- 5) 保存创建的源程序文件。
- 6) 将源程序文件添加到项目中。
- 7) 添加配置启动代码（适用于用 C 语言编写的源程序）。

下面以创建一个新的工程文件 My_Project1. UV2 为例，详细介绍建立一个 Keil C51 应用程序的过程。

1. 创建工程

1) 双击桌面上的 μ Vision2 快捷图标，进入图 B-1 所示的 KEIL C51 集成开发环境。 μ Vision2 启动后，总是打开用户前一次处理的工程。

2) 选择工具菜单栏的 Project 选项，在弹出图 B-2 所示的下拉菜单中选择 New Project 命令，建立一个新的 μ Vision2 工程，这时会弹出图 B-3 所示的项目文件保存对话框。

为工程项目取名并确定工程项目存放的路径后，单击保存按钮退出新建工程项目的窗口。这里应注意：

① 用户在给工程项目取名称时，工程名应便于记忆，但名称不宜过长。

② 在选择工程存放的路径时，建议为每个工程单独建立一个文件夹，并且将工程中所有的文件都放在这个文件夹中。

3) 在工程建立完毕后， μ Vision2 会立即弹出图 B-4 所示的目标器件选择窗口。目标器件选择的目的是告诉 μ Vision2 所使用的 8051 单片机芯片的型号，因为不同型号的 8051 单片机内部资源是不同的。 μ Vision2 根据选择的器件进行特殊功能寄存器预定义，在软/硬件仿真中提供易于操作的外设窗口。



图 B-1 Keil C51 集成开发环境界面

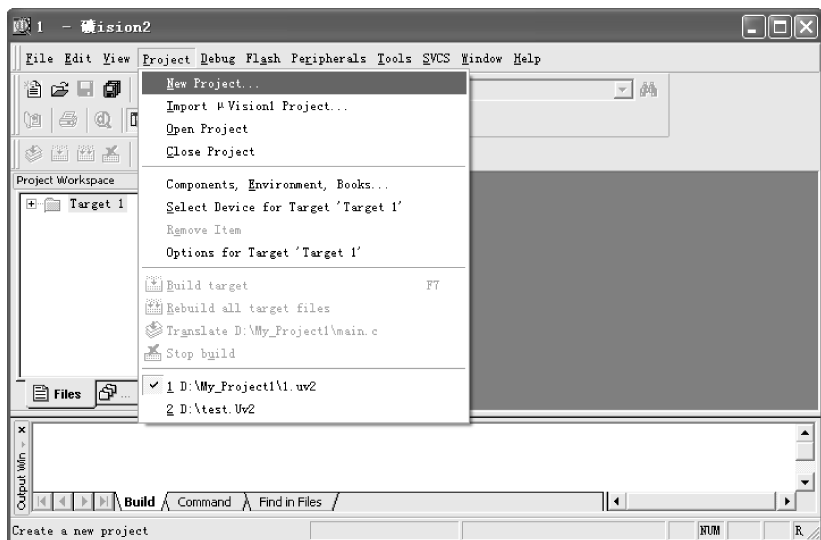


图 B-2 新建工程项目的菜单

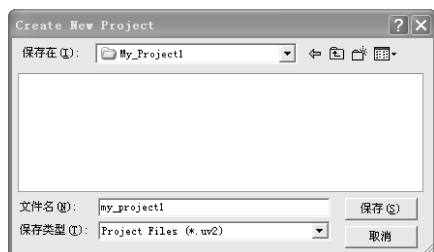


图 B-3 新建工程项目的对话框

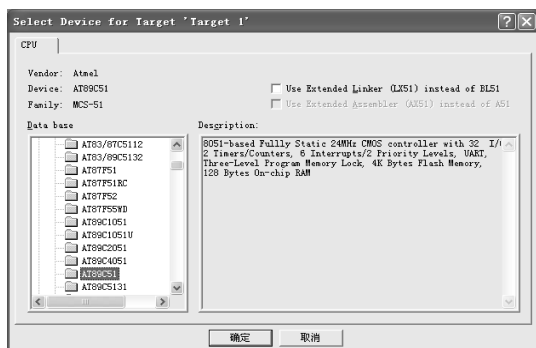


图 B-4 目标器件选择窗口

图 B-4 中选择的是 ATMEL 公司的 AT89C51 芯片。如果用户在选择完目标器件后想重新更改目标器件，可以通过工具菜单栏中的 Project 选项，在弹出图 B-5 所示的下拉菜单中选择 Select Device for Target 'Target1'，在弹出图 B-4 所示的窗口中重新选择目标器件。由于不同厂家的许多芯片型号相同或相近，当用户的目标器件型号在 μ Vision2 中找不到时，用户可以选择其他公司相近的芯片型号。

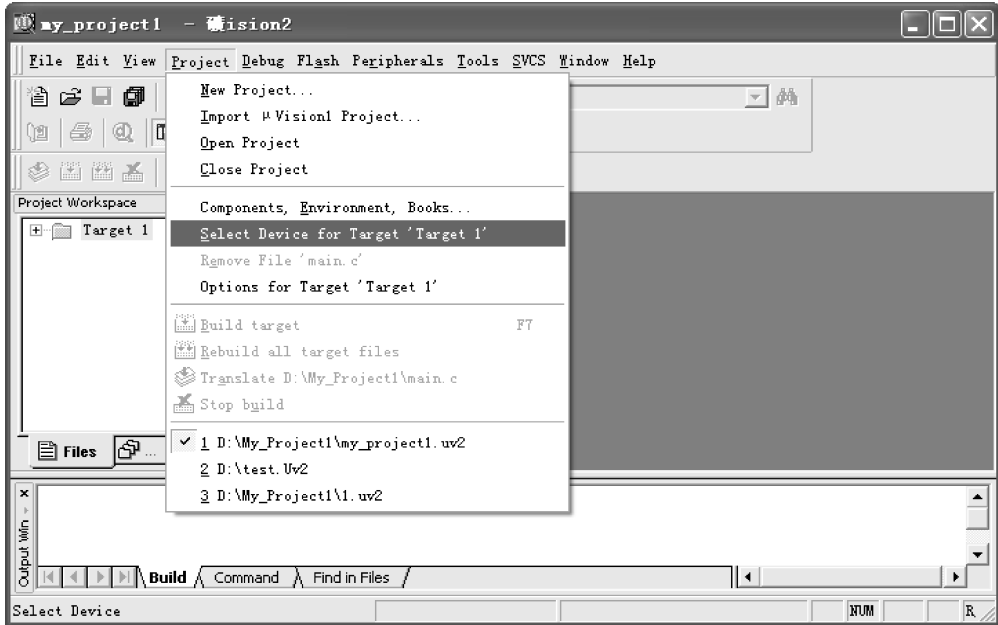


图 B-5 目标器件选择命令的菜单

选择目标器件并确定后，会立即弹出图 B-6 所示的复制并添加启动代码对话框，用户可以选择是否在工程项目中复制并添加启动代码 STARTUP.A51 文件。当用户采用 C 语言编写 MCS51 单片机应用程序时，应添加此文件；否则不添加此文件。

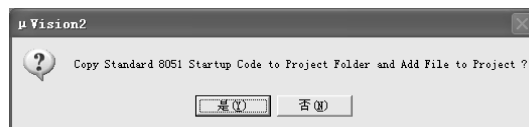


图 B-6 复制并添加启动代码对话框

4) 用户已经建立了一个空白的工程项目文件，并选择好了目标器件。下一步就可以创建源文件和输入源程序代码。

选择工具菜单栏的 File 选项，在弹出的图 B-7 所示的菜单中选择 New 命令，这时会出现一个图 B-8 所示的新建源编辑文件窗口 Text1。

5) 现在可以在新建的源文件编辑窗口中输入用户自己的源程序（如 main.c）。由于 μ Vision2 是一个标准的 Windows 应用程序，源文件的编辑方法同其他的文本编辑器是一样的，用户可以执行如输入、删除、选择、复制和粘贴等基本的文字处理命令。当然也可以在其他文本编辑器中编写源程序，如记事本等。通过其他文本编辑器编写源程序时，应注意在保存源文件时应确定该文件的扩展名为 .ASM 或 .C，而不能保存为默认的 .txt 或 .doc 等

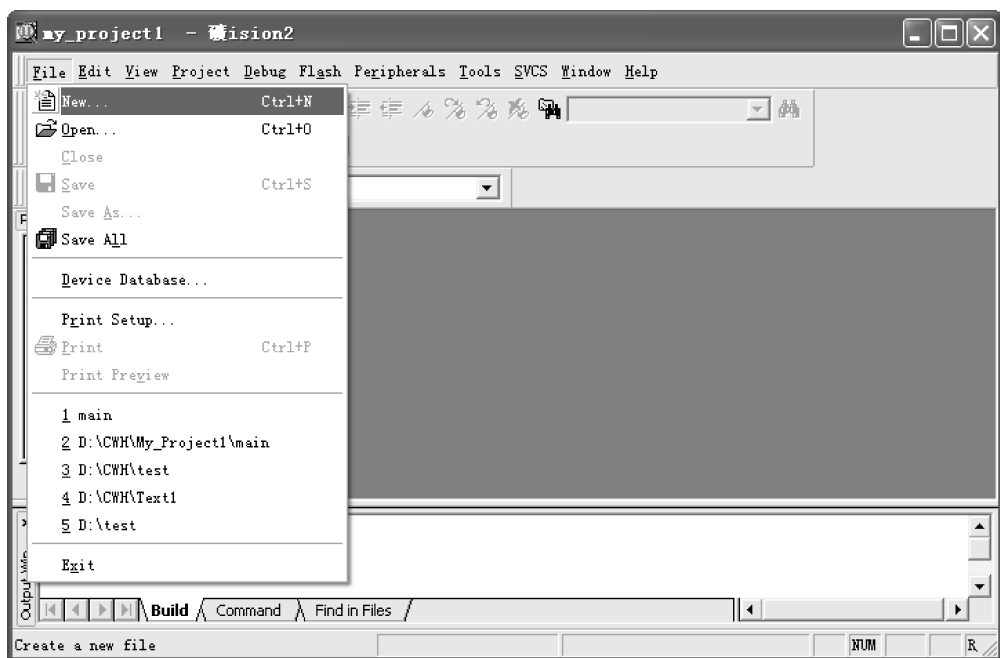


图 B-7 新建源文件的菜单

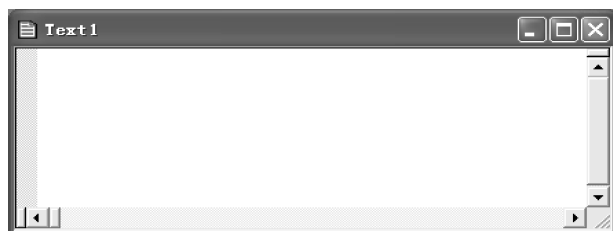


图 B-8 新建源文件的编辑窗口

文件格式。

6) 源程序输入完毕后选择工具菜单栏的 File 选项，在弹出菜单中选择 Save 命令保存源程序文件，这时会弹出图 B-9 所示的对话框。在文件名栏中输入源程序文件名，图中示范的是将源文件保存为 main.c。注意由于 μVision2 只支持汇编和 C 语言，因此在保存源文件时扩展名应是 .ASM 或 .C。

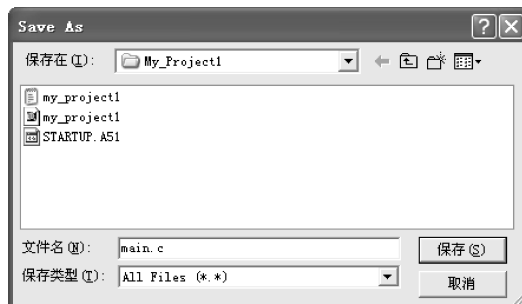


图 B-9 保存源程序文件窗口

在源程序文件正确保存后，源程序文件中的关键字就变成了蓝色。这是用户检查源程序中各关键字的一个好方法。

在创建源文件后，main.c 源程序文件只是一个独立的文件，和 My_Project1. UV2 工程项目还没有建立起任何关系。此时，用户应该把源程序文件添加到 My_Project1. UV2 工程中，从而构成一个完整的工程项目。在图 B-10 所示的 Project Workspace 窗口中，先选中 Source Group1，然后单击鼠标右键，在弹出菜单中选择 Add Files to Group ‘Source Group1’ 选项，此时会弹出一个图 B-11 所示的添加源文件的窗口。至此，My_Project1. UV2 工程已经建立完毕。

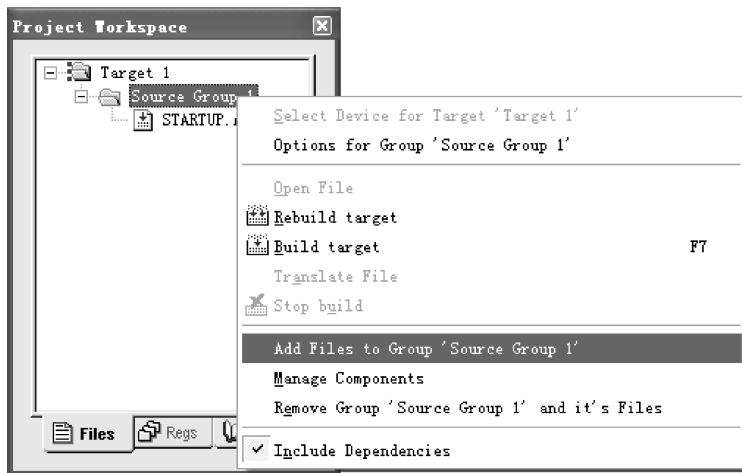


图 B-10 添加源程序文件的菜单

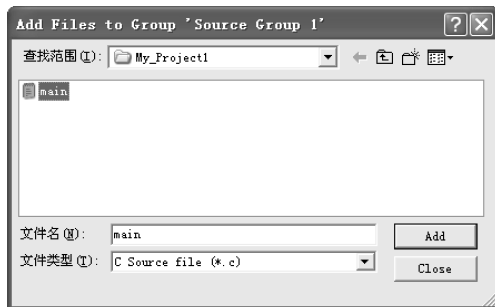


图 B-11 添加源程序文件的窗口

2. 添加和配置启动代码

如果用户采用 C 语言编写源程序代码，建立的工程项目中应添加启动代码文件 STARTUP.A51。当然如果用户采用汇编语言编写源程序代码，则不需要添加这个启动代码文件。

STARTUP.A51 文件是大多数 8051 CPU 及其派生产品的启动代码。启动代码用于清零数据存储寄存器，并初始化硬件和堆栈指针。如果要按照目标硬件的要求来修改 STARTUP.A51 文件，那么应将它从安装目录（如 C:\KEIL\C51\LIB）中复制到项目文件夹中。添加启动代码的方法有两种：

1) 在添加目标器件后，会弹出图 B-6 所示的对话框，询问是否将标准的 STARTUP.A51 文件复制到工程文件夹中，并将其添加到工程项目中，选择“是”按钮就可以完

成添加启动代码的工作。

2) 手动先将启动代码 STARTUP. A51 文件从安装目录（如 C:\KEIL\C51\LIB）中复制到项目文件夹中，然后使用添加源程序文件到工程项目的方法将 STARTUP. A51 文件添加工程项目中，如图 B-12 所示。

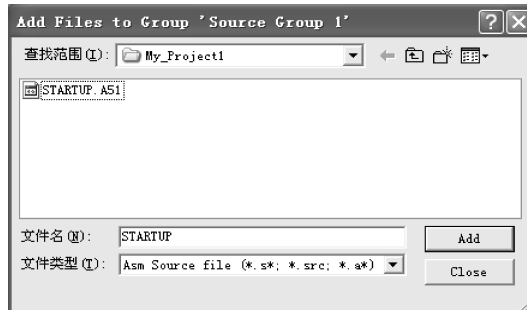


图 B-12 添加启动代码 STARTUP. A51 的窗口

3. 程序文件的编译、链接

(1) 为项目设置工程选项

μVision2 可以为工程项目设置选项。通过单击快捷图标栏的 Options for Target 图标 , 或通过选择工具菜单栏 Project 选项，在弹出的下拉菜单中选择 Option For Target ‘Target 1’ 命令，此时将弹出图 B-13 所示的 μVision2 调试环境设置窗口，为工程项目设置工程选项。

在 Target 选项卡中，可以指定目标硬件以及所选器件片内部件的所有相关参数，如图 B-13 所示。

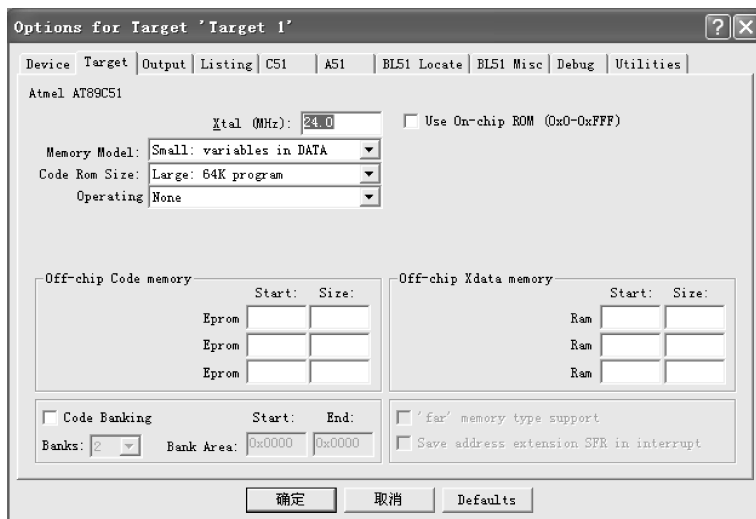


图 B-13 μVision2 调试环境设置窗口

单击 Output 选项卡，在打开的页面中勾选 Create Hex File 复选框，如图 B-14 所示；在工程编译完成后系统将自动生成以工程项目名为名字的目标代码文件 *.hex（如 My_Project1.hex），如图 B-15 所示。

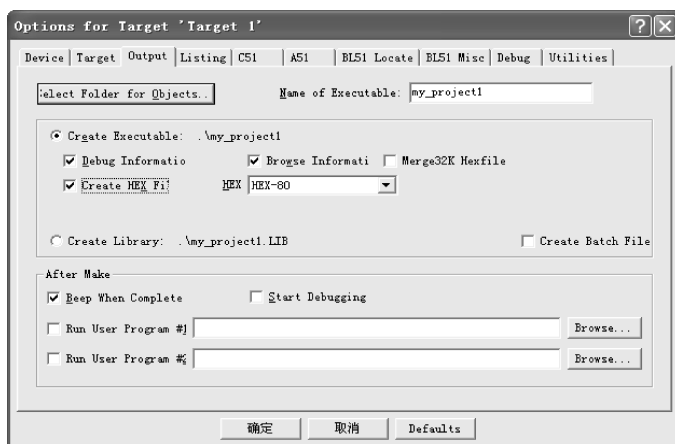


图 B-14 Option for Target—Output 选项页面

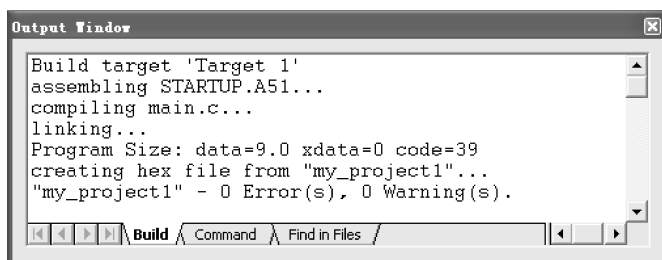


图 B-15 Output Windows 输出提示信息

选择 Debug 选项卡会出现图 B-16 所示的工作模式选择页面，在此页面中，用户可以设置不同的仿真模式。从图 B-16 可以看出， μ Vision2 有两种调试模式：Use simulator（软件模拟）和 Use（硬件仿真）。



图 B-16 Debug 设置窗口

Use simulator（软件模拟）选项将 μ Vision2 调试器设置成软件模拟仿真模式，在此模式下不需要实际的目标硬件设备可以模拟 8051 单片机的许多功能。用户可以在准备硬件之前，通过软件模式仿真调试用户程序，如控制算法程序等。

Use（硬件仿真）选项中有多种驱动，用户可以根据自己所用的开发工具选择对应的驱

动程序。当然用户可以为自己的开发工具添加驱动。运用此功能用户可以将 Keil C51 嵌入到自己的系统中，从而实现在目标硬件上调试程序。如果要使用硬件仿真，则应选择 Use 单选按钮，并在该栏后的驱动方式选择框中选择与开发工具对应的驱动程序。

(2) 程序编译与链接

通过单击快捷图标栏的 Build Target 图标或选择工具菜单栏 Project 选项后在弹出的下拉菜单中选择 Build Target 命令，可以对源程序文件进行编译，当然也可以选择 Rebuild ALL Target Files 来编译工程项目中的所有文件，此时会在 Output Windows 信息输出窗口中显示一些相关信息，如图 B-15 所示。Build Target 命令只会编译修改过的源文件或新的源文件。Rebuild All Target 命令将编译工程中所有文件，而不考虑文件是否被修改过。

若在编译过程中出现错误，系统会在输出窗口中给出错误所在的行和该错误的提示信息，用户可以根据这些提示信息，修改源程序中出现的错误，保存并重新编译直至没有错误信息。当出现错误信息时，用户可以用鼠标双击 Output Windows 窗口中的错误信息，此时 μ Vision2 会在编辑器窗口用颜色箭头指示出有错误的语句，这种方法可以快速定位有错误的语句。

至此一个完整的工程项目 My_Project1. UV2 已经完成。然而一个符合要求的、好的工程项目需要经软件调试、硬件调试和现场系统调试等反复修改、更新的过程。

4. 调试程序

μ Vision2 调试器可以调试用 C 语言和汇编语言开发的应用程序。 μ Vision2 调试有两种工作模式，即 Use Simulator（软件模拟）和 Use（硬件仿真）。

(1) 启动调试

在工具菜单栏 Debug 选项的下拉菜单中选择 Start/Stop Debug Session 命令可以启动或停止 μ Vision2 的调试模式。按照图 B-16 所示 Debug 设置窗口的设置， μ Vision2 会载入应用程序，并执行启动代码。 μ Vision2 保存编辑器窗口，并恢复最后一次调试时的调试窗口布局。在调试窗口中，下一条将要执行的语句用黄色箭头标出。如果退出调试模式， μ Vision2 会返回到源程序编辑窗口。

(2) 反编译窗口

反编译窗口用源程序和汇编程序的混合代码或汇编代码来显示用户应用程序，如图 B-17 所示。如果选 Disassembly 反编译窗口作为活动窗口，则程序的单步（Step）命令会工作在 CPU 的指令级而不是源代码的语句。

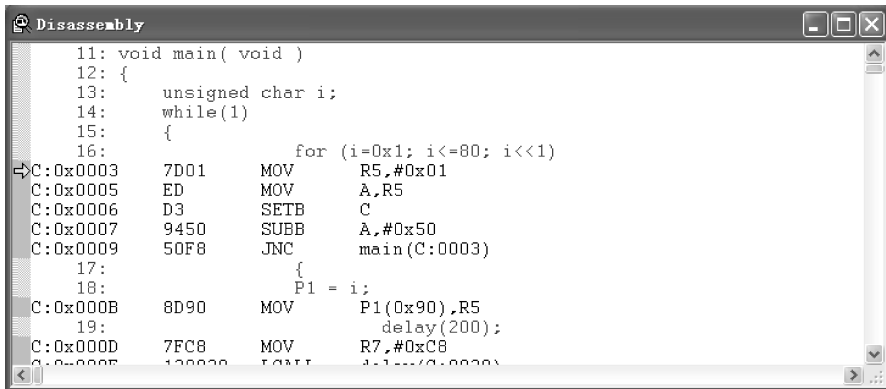


图 B-17 反汇编窗口

(3) 断点

μVision2 有几种设置断点的方法，在设置断点前，应将光标设置在需设置断点的程序行。

1) 通过鼠标双击需设置或取消断点的程序行。此时会在该程序行的前面添加或删除红色的断点标志。

2) 在有效程序行的任意位置，用鼠标的右键打开快捷菜单，选择 Insert/Remove Breakpoint 命令来设置或取消断点。

3) 通过单击快捷图标栏的 Insert/Remove Breakpoint 图标  或选择工具菜单栏的 Debug 选项，在弹出的下拉菜单中选择 Insert/Remove Breakpoint 命令来设置或取消断点。

(4) 目标程序的执行

1) 单步跟踪 (Step Into) 。用工具菜单栏 Debug 选项中的 Step 或快捷图标的 Step Into 命令按钮可以单步跟踪程序。每执行一次单步跟踪命令，程序将运行一条指令。当前的指令用黄色的箭头标出，每执行一步箭头移动一次，已执行的语句呈现绿色。单步跟踪是以指令为执行单元的。

2) 单步运行 (Step Over) 。用工具菜单栏 Debug 选项中的 Step Over 或快捷图标的 Step Over 命令，即可实现单步运行程序。此时的单步运行命令将函数和函数调用当成一个实体来看待，因此单步运行是以语句为基本执行单元的，而不管该语句是单一命令还是函数调用。

3) 执行返回 (Step Out) 。在使用单步跟踪命令跟踪到函数或子程序的内部时，可以使用 Step Out 命令来实现程序的 PC 指针返回到调用此子程序或函数的下一条语句。

4) 执行到光标 (Run to Cursor Line) 。用工具菜单栏 Debug 选项中的 Run to Cursor Line 命令或快捷图标 Run to Cursor Line 命令，使程序执行到光标所在的程序行，但不包括此行。此命令的实质是在光标所在行设置有临时断点。

5) 全速运行 (Go) 。用工具菜单栏 Debug 选项中的 Go 命令或快捷图标 Run 命令可以实现程序的全速运行。当然如果程序中设置有断点，程序将全速运行至断点所在的程序行，不包括该行。如果程序中没有设置断点，在程序的全速运行期间，μVision2 不允许查看任何资源，也不接受任何命令。

(5) Watch 窗口

Watch 窗口可以查看和修改用户程序中变量的值。Watch 窗口的内容在程序停止运行后自动更新。通过工具菜单栏的 View 选项，单击下拉菜单中的 Watch & Call Stack Window 命令，μVision2 调试窗口中将出现图 B-18 所示的 Watch 窗口。

其中，Locals 页中显示当前函数的所有局部变量。Watch 页中显示的是用户所指定的变量。将变量添加到 Watch 窗口的方法有：

1) 选择 Watch 窗口的 Watches 页，按 F2 键后输入变量名即可，如图 B-19 所示。

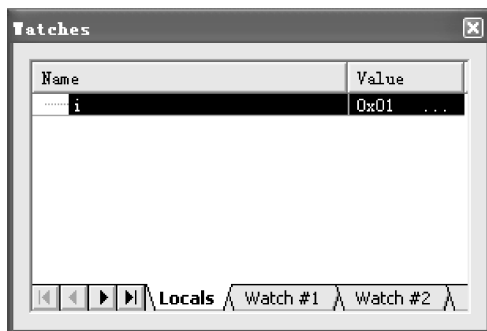


图 B-18 Watch 窗口

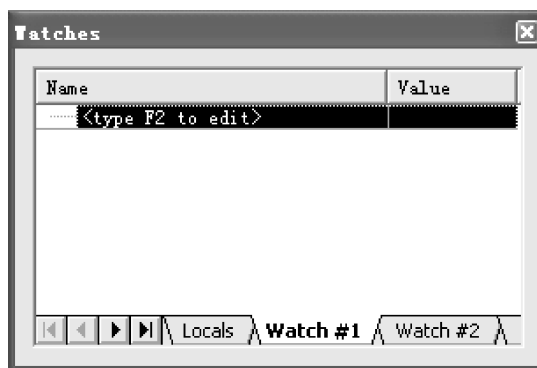


图 B-19 Watch 窗口的 Watches 页

2) 在源程序的变量名上用鼠标右键打开快捷菜单，选择 Add “变量名” to Watch Window...，变量便添加到 Watch 窗口中。

3) 在 Output Window 的 Command 页中，使用 WatchSet 命令输入变量名。

要删除 Watch 窗口中的变量，选择该变量用 Delete 键删除。

(6) Memory 窗口

Memory 窗口能显示单片机系统各存储区的内容，如图 B-20 所示。在 Memory 窗口的 Address 选项中可以输入表达式，表示要显示区域的起始地址。查看各存储区的内容的方法如下：

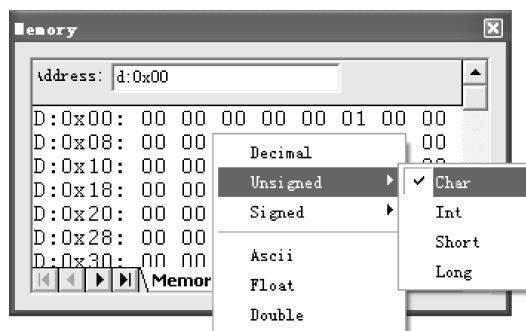


图 B-20 Memory 窗口

1) 查看片内数据存储区的内容。在 Address 选项中输入：d: 0x00 并回车，便可以查看起始地址为 0x00 的片内数据区的数据，d 代表 data 存储类型。

2) 查看程序存储器区域的内容。在 Address 选项中输入：c: 0x0000 并回车，便可以查看起始地址为 0x0000 的程序存储器区域的数据，c 代表 code 存储类型。

3) 查看片外数据存储区的内容。在 Address 选项中输入：x: 0x0000 并回车，便可以查看起始地址为 0x0000 的片外数据存储区的数据，x 代表 xdata 存储类型。

在数据区域通过鼠标右键打开快捷菜单可以修改数据区数据的显示格式。

下面用一个实例来介绍 Keil C51 程序的开发与调试过程。

1) 创建 My_Project1. UV2 工程项目文件。将 My_Project1. UV2 的工程文件放在 D:\ My_

Project1 目录中。选择 ATMEL AT89C51 为目标器件，复制并添加标准的启动代码文件 STARTUP.A51。

2) 编辑 main.c 文件。在 μ Vision2 的编辑窗口中，编译 main.c 源程序文件，如图 B-21 所示，并将该文件添加到工程中。My_Project1.UV2 工程的文件结构如图 B-22 所示。main.c 源程序的功能是实现 8051 单片机 P1 端口的“走马灯”功能，即先控制 P1 口的最低位 P1.0 为“1”（高电平），其他位为“0”（低电平）；经一段延时后，控制 P1.1 为“1”，而其他位为“0”，随着程序的继续执行，当 P1 的最高为 P1.7 为“1”时，经延时后，再次控制 P1 的 P1.0 为“1”，如此循环不断。

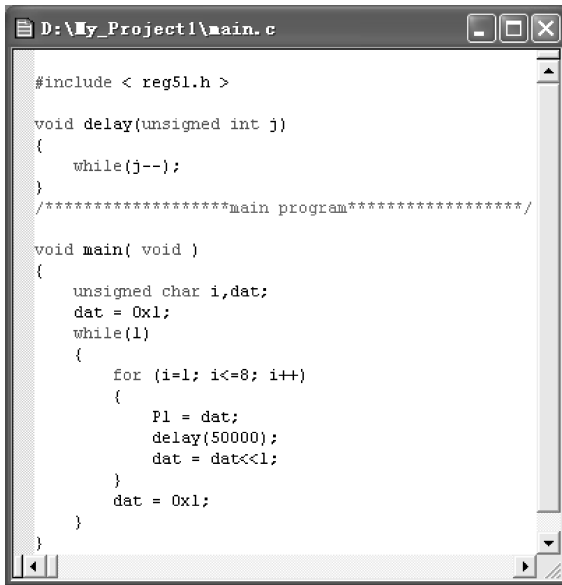


图 B-21 编辑窗口中的源文件

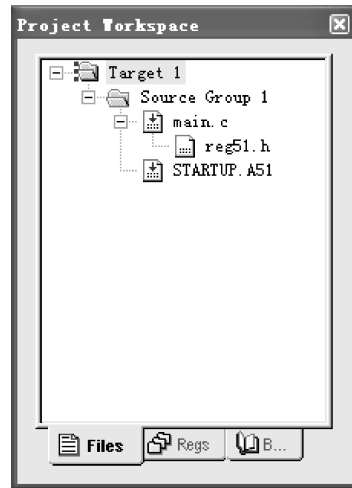


图 B-22 工程的文件结构

程序实例中的延时时间由软件延时函数来产生。

3) 编译和链接。用工具菜单栏 Project 选项中的或快捷图标栏的 Build Target 命令编译和链接工程项目。编译链接的状态信息在 Output Window 的 Build 页中显示出来。如图 B-23 所示。编译和链接的结果是：没有错误和警告。下一步就可以调试程序。此实例采用软件模拟的调试模式。

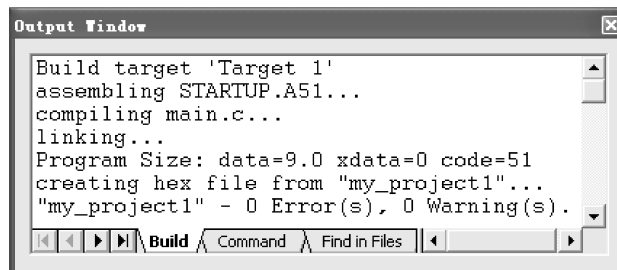


图 B-23 Output Window 中显示的编译链接状态信息

4) 调试 main.c。通过工具菜单栏 Debug 选项中或快捷图标栏的 Start/Stop Debug Session 命令来启动 μ Vision2 的调试画面。 μ Vision2 初始化调试器并运行启动代码，且运行到主函数 main 函数。如图 B-24 所示，黄色箭头指在 main 函数的第一条有效语句上，指明下一条要执行的语句。

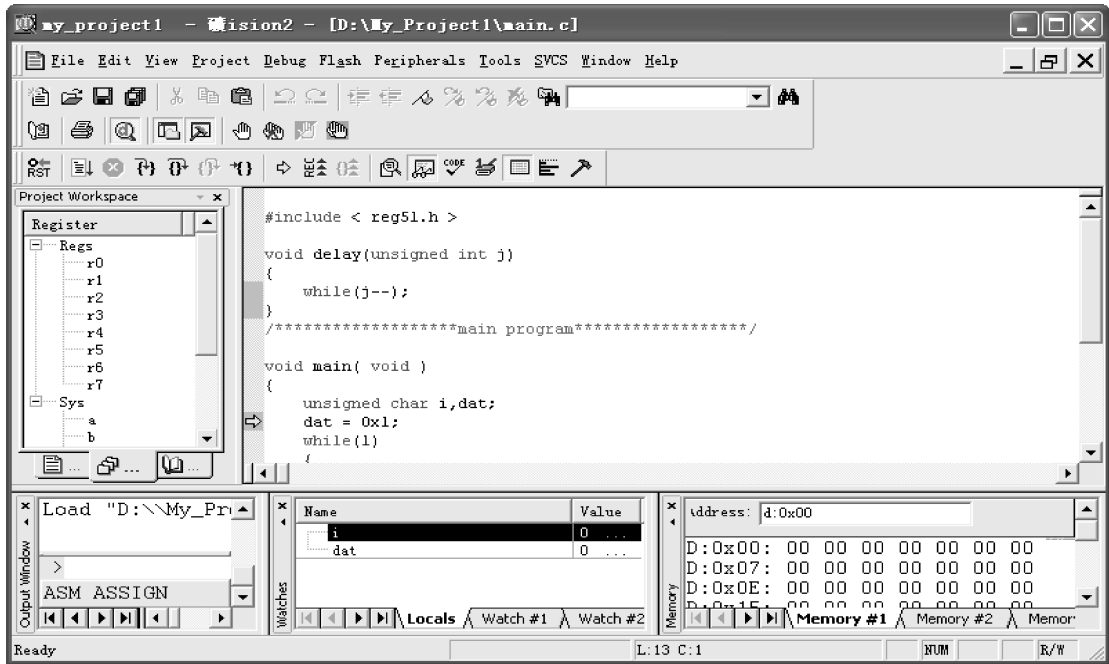


图 B-24 μ Vision2 的调试画面

① 在工具菜单栏 Peripherals 选项中的 I/O - Ports 下拉菜单中，选择 Port 1，此时弹出 8051 单片机 P1 端口的控制窗口，它可以控制和监视 P1 端口的状态。如图 B-25 所示，P1 的初始状态为 0xff，它显示的是 8051 单片机复位后 P1 端口的状态。

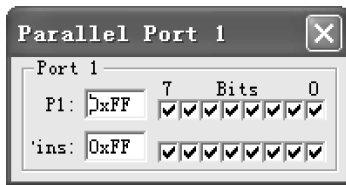


图 B-25 P1 端口的控制窗口

② 调试过程。使用 Step Over 或 Run to Cursor Line 命令使程序运行到图 B-26 所示的位置，此时已经将变量 dat 中的数据 0x01 赋值给 P1 端口。观察 P1 端口控制窗口的状态变化，此时 P1 口的第 0 位标记有“√”，表示该位为高电平“1”。继续使用 Step Over 或 Run to Cursor Line 命令，变量 dat 经移位赋值后它的值变为 0x02，当程序运行到图 B-27 所示的位置时，P1 端口的控制窗口的状态由在 P1 口的第 0 位为“1”变成 P1 口的第 1 位为“1”，如图 B-27 所示。随着程序的继续运行，P1 口的控制窗口的状态也随着改变，如此循环 8 次，直到 P1 口的最高为高电平“1”后，将变量 dat 重新赋初值 0x01。由于 main.c 源程序代码中采用的是 while 无限循环结构，随着程序的循环运行，高电平“1”状态从 P1 口的第 0 位逐步向最高位流动，并反复不断。

用 Run 或 Go 全速运行命令，就可以从 P1 口的控制窗口看出高电平“1”状态从 P1 口的低位不断地向高位流动，并循环不断。

用户可以在任何时间用 Start/Stop Debug Session 命令停止调试。

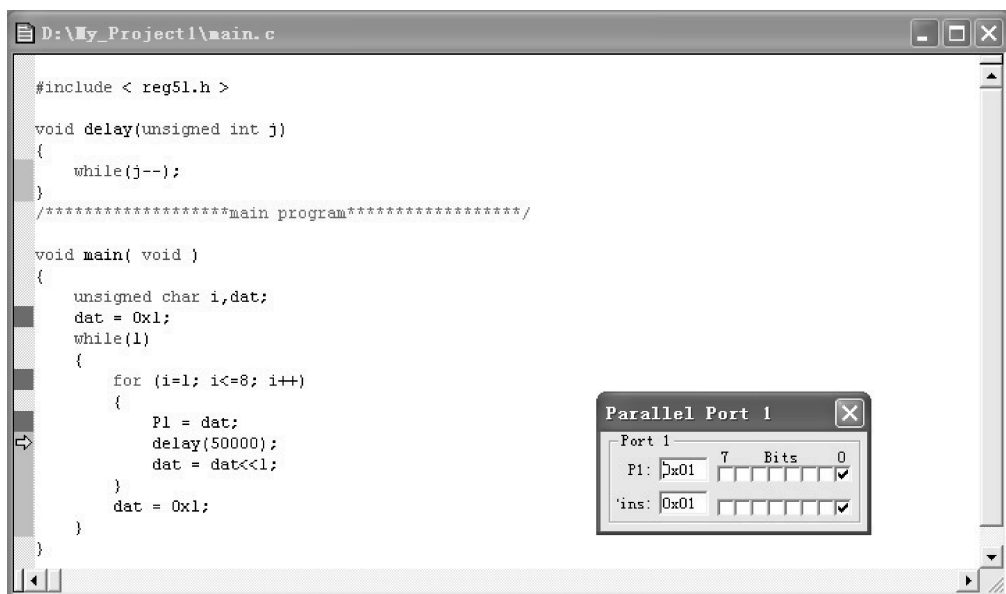


图 B-26 μVision2 的调试画面 1

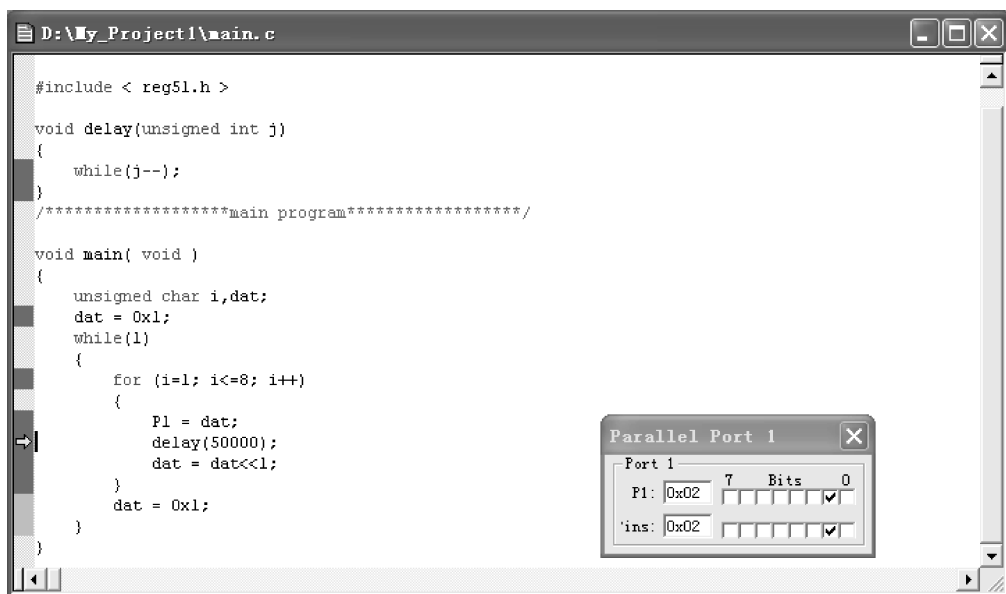


图 B-27 μVision2 的调试画面 2

参考文献

- [1] 张毅刚, 彭喜元, 彭宇. 单片机原理及应用 [M]. 2 版. 北京: 高等教育出版社, 2010.
- [2] 张毅刚, 彭喜元, 彭宇. 单片机原理及应用 [M]. 北京: 高等教育出版社, 2004.
- [3] 文武松, 王璐, 杨贵恒. 单片机原理及应用 [M]. 北京: 机械工业出版社, 2015.
- [4] 丁元杰. 单片微机原理及应用 [M]. 北京: 机械工业出版社, 2005.
- [5] 陈桂友, 孙同景. 单片机原理及应用 [M]. 北京: 机械工业出版社, 2013.
- [6] 李念强, 崔世耀, 何敬银, 等. 单片机原理及应用 [M]. 2 版. 北京: 机械工业出版社, 2013.
- [7] 万隆, 巴奉丽, 尹丽菊, 等. 单片机原理及应用技术 [M]. 2 版. 北京: 清华大学出版社, 2014.
- [8] 何立民. 单片机应用系统设计 [M]. 北京: 北京航空航天大学出版社, 1990.
- [9] 张毅刚, 彭喜元. 单片机原理及接口技术 [M]. 北京: 人民邮电出版社, 2008.
- [10] 赵全利, 肖兴达. 单片机原理及应用教程 [M]. 2 版. 北京: 北京大学出版社, 2010.
- [11] 马忠梅, 籍顺心, 张凯, 等. 单片机的 C 语言应用程序设计 [M]. 5 版. 北京: 北京航空航天大学出版社, 2013.
- [12] 张培仁, 孙占辉, 张欣, 等. 基于 C 语言编程 MCS-51 单片机原理与应用 [M]. 北京: 清华大学出版社, 2003.
- [13] 徐爱钧, 彭秀华. Keil Cx51 V7.0 单片机高级语言编程与 μ Vision2 应用实践 [M]. 北京: 电子工业出版社, 2007.
- [14] 高卫东, 辛友顺, 韩彦征. 51 单片机原理与实践 [M]. 北京: 北京航空航天大学出版社, 2008.
- [15] 周广兴, 张子红. 单片机原理及应用教程 [M]. 北京: 北京大学出版社, 2010.
- [16] 何立民. MCS-51 系列单片机应用系统设计——系统配置与接口技术 [M]. 北京: 北京航空航天大学出版社, 2000.
- [17] 李朝青. 单片机原理及接口技术 [M]. 3 版. 北京: 北京航空航天大学出版社, 2005.
- [18] 汪贵平. 新编单片机原理及应用 [M]. 北京: 机械工业出版社, 2011.
- [19] Cx51 Compiler User's Guide. Keil software Inc., 2001.
- [20] Getting Started with μ Vision2 User's Guide. Keil software Inc., 2001.
- [21] Marco Assembler and Utilities User's Guide. Keil software Inc., 2001.
- [22] 广州周立功单片机发展有限公司. I²C 总线规范 [S/OL]. <http://www.zlgmcu.com/download/downs.asp?ID=780>.
- [23] 韩峻峰, 海涛, 陈文辉. 单片机原理及应用 [M]. 北京: 机械工业出版社, 2012.
- [24] 高峰. 单片微机应用系统设计及实用技术 [M]. 2 版. 北京: 机械工业出版社, 2004.
- [25] 王幸之, 王雷, 翟成, 等. 单片机应用系统抗干扰技术 [M]. 北京: 北京航空航天大学出版社, 2000.
- [26] 严天峰. 单片机应用系统设计与仿真调试 [M]. 北京: 北京航空航天大学出版社, 2005.
- [27] 李群芳. 单片机原理与应用——嵌入式系统开发基础 [M]. 武汉: 武汉大学出版社, 2006.

在线互动交流平台

官方微博: <http://weibo.com/cmpjsj>

读者教师QQ群: 427066270

33811098

单片机原理及应用



普通高等教育电气信息类规划教材

- ◎ C语言程序设计
- ◎ 微机原理及接口技术
- ◎ 单片机原理及应用
- ◎ 单片机应用技术
- ◎ 单片机原理与应用
- ◎ 单片机原理及应用
- 单片机原理及应用
- ◎ 电机与拖动基础
- ◎ 电力电子技术 第3版
- ◎ 机电传动控制
- ◎ 数据采集技术与系统设计
- ◎ 自动化专业英语教程
- ◎ 信号与控制基础
- ◎ CPLD/FPGA控制系统设计
- ◎ PLC基础及应用教程(三菱FX_{2N}系列)
- ◎ 西门子S7-200 PLC应用教程
- ◎ 传感器技术实用教程
- ◎ EDA技术与应用

蔡启先
蔡启仲
韩骏峰
陈桂友
杭和平
王景景
蔡启仲
徐胜军
周渊深
张志义
李念强
王彩霞
张家海
周京华
秦春斌
赵全利
吕勇军
孙志雄

地址: 北京市百万庄大街22号

邮政编码: 100037

电话服务

服务咨询热线: 010-88379833

读者购书热线: 010-88379469

网络服务

机工官网: www.cmpbook.com

机工官博: weibo.com/cmp1952

金书网: www.golden-book.com

教育服务网: www.cmpedu.com

封面无防伪标均为盗版



机工教育微信服务号 IT有得聊微信服务号

机工社本科电气类教师群
扫一扫二维码, 加入社群

上架指导 电气信息/单片机

ISBN 978-7-111-54515-6

策划编辑◎时 静/封面设计◎



ISBN 978-7-111-54515-6



定价: 45.00元