MCS-51 单片机原理、接口及应用

主编：王质朴　吕运朋

参编：杨勇 王丽霞　李文方 李海霞　王二萍　乐丽琴

内容简介

本书以 MCS-51 系列单片机为对象机型，介绍了 8 位单片机及其应用系统的理论和技术。全书分为原理、接口和应用三个层次，共 12 章，全面讲述了 MCS-51 的硬件结构、功能部件、指令系统、A51 汇编语言、C51 语言等基础知识，介绍了外围接口技术和典型接口部件如 ADC、DAC、键盘、显示器、IC 卡、微型打印机等，并介绍了单片机应用系统设计的一般方法和步骤。全书力求反映近年单片机及外围接口技术的最新发展，如目前迅速流行的串行总线 SPI、IIC、1-WIRE 技术及其典型 IC 芯片。每章附有习题。第 11 章选编了典型的试验单元和课程设计。

本书可作为高等学校机械设计与自动化、电子信息工程、测控技术与仪器、电气自动化等专业的单片机课程教材，也适于单片机爱好者自学和工程技术人员参考之用。

前　言

现代计算机技术的发展出现了两大分支，即通用计算机系统和嵌入式计算机系统。单片机是嵌入式系统中的典型代表。单片机最本质的功能特性是控制。它将典型的计算机功能资源制作于一片集成电路之中，然后嵌入到各类具体设备内部，形成了可以实现人类智能的普遍意义上的控制器件，因此也称为微型控制器（MCU）。在众多的单片机中，MCS-51 系列机型的出现是 MCU 产业发展中的里程碑。它历经近 30 年发展，已经形成了一个品种多、功能全、性价比高、用户群庞大的系列产品，成了是事实上 8 位单片的技术标准，也成了国内高校最为流行的单片机教学机型。

单片机普遍意义上的控制功能使得它的应用范围非常的广泛，因而也成为当今工科类学生所应必须掌握的基本专业基础知识和技能。培养基本知识扎实、掌握实际运用技能的单片机应用型人才，正是编写本书所致力的目标。

本书以 MCS-51 系列为对象机型，系统介绍了单片机应用的硬件软件的知识和应用，其主要特点是：

（1）层次分明、布局全面、系统性强。全书分为原理、接口和应用三个层次，分别讲述了 MCS-51 的硬件和软件的基本结构、特殊功能单元系统和工作原理；MCS-51 系统扩展的基本原理、方法和通用外围接口电路；在此基础上又讲述了实用中迅速发展的 C51 语言编程技术和应用系统的一般设计方法，最后介绍了几个典型的试验单元和课程设计。

（2）力求反映单片机应用领域最新技术的发展。本书介绍了近年实用中发展很快、带来单片机应用系统技术进步的串行总线技术 SPI、IIC 和单总线及其典型的 IC 芯片，如 A/D、D/A、LCD 模块、温度测量、IC 卡等，介绍了提高编程效率 C51 编程技术，使教材内容尽可能反映实践技术的最新发展。

（3）力求方便读者自学。本书多以文图结合的形式讲述知识；每条指令后面都加以注释；对于较难理解的外围扩展技术都附件以具体连接实例讲解；A51、C51 编程结合 KEIL 开发系统的编程和调试运行界面讲解等，尽力创设一种易于自学、便于理解、能够上机操作的学习环境。

（4）力求培养读者的实际应用能力。在讲解 MCS-51 基本知识的基础上，以较多的篇幅介绍了单片机应用系统的主要组成环节、应用系统的主要组成方法及典型的应用电路和源程序，以及从教学实际中挑选的效果好的典型试验单元，从而使读者能够从实用系统的整体上来构建 MCS-51 的知识结构，掌握应用设计的基本思路、方式和技能。

本书将教学大纲要求的内容排在每章的前面，选学内容在目录中和各节标题的后面标以“(*)”符号，可根据教学实际适当选用。

本书由王质朴、吕运朋任主编，张勇、王丽霞、李文方、李海霞、乐丽琴、王二萍参与编写。全书共分 12 章，其中第 0 章、11 章由张勇编写，第 1 章由王丽霞编写，第 2 章由李文方编写，第 3 章由李海霞编写，第 4 章由王二萍编写，第 5 章由乐丽琴编写，第 6、7、8 章由王质朴编写，第 9、10 章由吕运朋编写。王质朴统审了全稿。

编写中我们参考了大量的著作和资料，其中主要的列于后面的参考文献中，在此对作者一并致以衷心的敬意和谢忱。感谢刘杨参与制作了部分插图。

限于编者知识水平，也限于编写时间，书中错误和不足在所难免，恳请读者批评指正。

编者

2009 年 7 月

目　录

第 0 章 单片机概述

0.1 单片机的定义及应用领域简介

1）单片机的定义

单片机的全称是单芯片微型计算机(single chip microcomputer)，也称作微控制器（MicroController Unit)，它是将中央处理单元 (Center Processing Unit，CPU，也称作微处理器)、数据存储器RAM(random access memory,随机读写存储器)、程序存储器ROM (read only memory，只读存储器)以及 I/O（input/output，输入输出）接口集成在一块芯片上，构成的一个计算机系统。

2）单片机的诞生

单片机诞生于 20 世纪 70 年代末，具有代表性的事件是 1976 年 Intel 公司推出了 MCS-48 单片机系列的第一款产品：8048。这款单片机在一个芯片内集成了超过 17000 个晶体管，包含一个 CPU（central processing United，中央处理器），1KB 的 EPROM（erasable programmable read only memory，可擦可编程只读存储器），64 字节的 RAM，27 个 I/O 端口和一个 8 位的定时器。8048 很快就成为了控制领域的工业标准，它们起初被广泛用来替代诸如洗衣机和交通灯等产品中的控制部分。

1980 年，Intel 公司在 MCS-48 的基础上推出了 MCS-51 系列的第一款单片机 8051，单片机的功耗、大小和复杂程度都提高了一个数量级。与 8048 相比，8051 集成了超过 60000 个晶体管，拥有 4KB 的 ROM，128B 的 RAM，32 个 I/O 接口，一个串行通信接口和 2 个 16 位的定时器。经过二十多年的发展，MCS-51 系列单片机已经形成了一个规模庞大、功能齐全、资源丰富的产品群。

3）单片机的应用领域

单片机在我们的日常生活和工作中无处不在、无处不有：家用电器中的电子表、洗衣机、电饭褒、豆浆机、电子秤；住宅小区的监控系统、电梯智能化控制系统；汽车电子设备中的 ABS、GPS、ESP、TPMS；医用设备中的呼吸机，各种分析仪，监护仪，病床呼叫系统；公交汽车、地铁站的 IC 卡读卡机、滚动显示车次和时间的 LED 点阵显示屏；电脑的外设，如键盘、鼠标、光驱、打印机、复印件、传真机、调制解调器；计算机网络的通讯设备；智能化仪表中的万用表，示波器，逻辑分析仪；工厂流水线的智能化管理系统，成套设备中关键工作点的分布式监控系统；导弹的导航装置，飞机上的各种仪表等等。有资料表明：2007 年全球单片机的产值达到 151 亿美元，我国单片机的销售额达到 400 亿元人民币，我国每年单片机的需求量达 50 至 60 亿片，是全球单片机的最大市场。可以说单片机已经渗透到了我们生活的各个领域。

4）单片机与嵌入式系统

所谓嵌入式系统，就是嵌入到对象体系中的专用计算机系统。“嵌入性”、“专用性”与“计算机系统”是嵌入式系统的三个基本要素。对象体系则是指嵌入式系统所嵌入的宿主系统。按照上述嵌入式系统的定义，只要满足定义中三要素的计算机系统，都可称为嵌入式系统。嵌入式系统按形态可分为设备级（工控机）、板级（单板、模块）、芯片级（MCU、SoC）。单片机是嵌入式系统使用的一种核心元件。

嵌入式系统是现代计算机的两大分支之一，另一大分支是通用计算机，通用计算机的代表性产品是台式计算机。这两大计算机分支的发展方向不同：通用计算机的发展方向是总

线速度的无限提升，存储容量的无限扩大，嵌入式系统的发展方向是体积更小、控制能力与控制的可靠性更高。

0.2 MCS-51 系列单片机及其主要类型

MCS-51 系列单片机指的是 Intel 公司生产的一个系列的单片机的总称。20 世纪 80 年代中期以后，由于 Intel 公司将重点放在高档微处理器芯片的开发上，所以将其 MCS–51 系列中的 80C51 内核使用权以专利互换或出售的形式转让给了全世界许多著名 IC 设计厂商，如 AMTEL、PHILIPS、ANALOG DEVICES、DALLAS 等。这些厂家生产的单片机是 MCS-51 系列单片机的兼容产品，或者说是与 MCS-51 指令系统兼容的单片机。MCS-51 系列单片机是商业化的单片机的鼻祖，多年来积累的技术资料和开发经验是其他系列单片机所不能比拟的，MCS-51 系列单片机事实上已经成为 8 位单片机的行业标准。所以，本课程以 MCS-51 系列单片机为对象进行讲授。

MCS-51 系列单片机按照功能可以划分为以下主要类型：

（1） 基本型

基本型主要包括 8031、8051 和 8751 等通用产品，其基本特性如下：

- 8 位 CPU
- 4kbytes 片内程序存储器(ROM)
- 128bytes 的片内数据存储器(RAM)
- 32 条并行 I/O 口线
- 21 个专用寄存器
- 2 个可编程定时/计数器
- 5 个中断源，2 个优先级
- 一个全双工串行通信口
- 外部数据存储器寻址空间为 64kB
- 外部程序存储器寻址空间为 64kB
- 逻辑操作位寻址功能
- 一个片内时钟振荡器和时钟电路
- 单一+5V 电源供电

（2） 增强型

增强型有 8052、8032、8752、89C52、89S52 等。这些单片机内部的 ROM、RAM 容量比基本型增大了一倍，同时定时器增为 3 个。87C54 内部 ROM 为 16KB，87C58 增加到 32KB。另外，诸如中断源、A/D、SPI、IIC 接口等等也越来越多地集成到了 MCS-51 单片机中。

（3） 低功耗型

低功耗型有 80C5X、80C3X、87C5X 和 89C5X 等。型号中的“C”字样的单片机采用 CHMOS 工艺，特点是低功耗。

（4） ISP 型

ISP（In System Programming），在线编程。是 Lattice 半导体公司首先提出来的一种让我们能在产品设计、制造过程中的每个环节，甚至在产品卖给最终用户以后，具有对其器件、电路板或整个电子系统的逻辑和功能随时进行重组或重新编程的技术。具有代表性的产品有 ATMEL 公司的 AT89S51、AT89S52 等 S 系列的产品。

（5） IAP 型

IAP(In Appplication Program)，即在应用中可编程。就是在系统运行的过程中动态编程，这种编程是对程序执行代码的动态修改，而且毋须借助于任何外部力量，也毋须进行任何机械操作。这一点有别于 isp。一般来说，isp 在进行加载程序以前，需要设置某些功能引脚，

迫使 IC 转入自举状态。而 IAP 则不需要作硬件上的任何动作，只要有合法的数据来源。数据源既可以是内部程序运行的结果，也可以来自 UART，I/O 口或者总线。IAP 不仅提供现场或者远程软件修改升级，也可以把它理解成 idate，pdate 或者 xdate，替代 I2C 之类的外部 E2PROM，存储并加密数据。典型芯片如 SST 公司开发的 C51 系列单片机：SST89C54/58。

单片机的主要类型有：

1）MOTOROLA 单片机

MOTOROLA 是世界上最大的单片机厂商，品种全、选择余地大、新产品多是其特点。在 8 位机方面有 68HC05 和升级产品 68HC08。68HC05 有 30 多个系列，200 多个品种，产量已超过 20 亿片。16 位机 68HC16 也有十多个品种。32 位单片机的 683XX 系列也有几十个品种。MOTOROLA 单片机特点之一是在同样速度下所用的时钟频率较 Intel 类单片机低得多，因而使得高频噪声低、抗干扰能力强，更适合用于工业控制领域及恶劣的环境。

2）Microchip 单片机

由美国 Microchip 公司推出的 PIC 单片机系列产品，已有三种系列多种型号的产品问世，从电脑的外设，家电控制，电讯通信，智能仪器，汽车电子到金融电子的各个领域都得到广泛的应用。它的主要产品是 16C 系列 8 位单片机，CPU 采用 RISC 结构，仅 33 条指令，其高速度，低电压，低功耗，大电流 LCD 驱动能力和低价位 OTP 技术等都体现出单片机产业的发展新趋势。且以低价位著称，一般单片机价格都在一美元以下。Microchip 单片机没有掩膜产品，大都是 OTP（一次性可编程） 器件，近年已推出 FLASH 型单片机。Microchip 强调节约成本的最优化设计、使用量大、档次低、价格敏感的产品。

3）Atmel 单片机

AVR 单片机是 1997 年由 ATMEL 公司研发出的增强型内置 Flash 的精简指令集高速 8 位单片机。AVR 单片机具有这样一些特点，如：运行速度快；芯片内部的 Flsah、EEPROM、SRAM 容量较大并且全部支持在线编程烧写(ISP)；上电自动复位；每个 IO 口都可以以推换驱动的方式输出高、低电平，驱动能力强；内部资源丰富等。目前支持 AVR 单片机编译器的语言主要有汇编语言、C 语言、BASIC 语言等。

0.3 单片机的发展趋势

目前单片机的发展有以下几个主要的趋势。

（1）低功耗。单片机多数是采用 CMOS（金属栅氧化物）半导体工艺生产。CMOS 芯片除了低功耗特性之外，还具有功耗的可控性。CMOS 电路的特点是低功耗、高密度、低速度、低价格。采用双极型半导体工艺的 TTL 电路速度快，但功耗和芯片面积较大。随着技术和工艺水平的提高，又出现了 HMOS（高密度、高速度 MOS）和 CHMOS 工艺。CHMOS 和 HMOS 工艺的结合。目前生产的 CHMOS 电路已达到 LSTTL 的速度，传输延迟时间小于 2ns，它的综合优势已优于 TTL 电路。因而，在单片机领域 CMOS 正在逐渐取代 TTL 电路。几乎所有的单片机都有 WAIT、STOP 等省电运行方式。允许使用的电压范围越来越宽，一般在 3～6V 范围内工作。8051F9XX 单片机的最低电压可到 0.9V，Atmel 公司最新发布的 0.7V TinyAVR 甚至可以使用一个钮扣电池供电。

（2）外围电路内装。目前单片机的集成度不断提高，除了一般必须具有的 CPU、ROM、RAM、定时器/计数器等以外，片内集成的部件还有模/数转换器、人机界面、通信接口、I^2C、SPI 、CAN、USB 总线等等。人机界面技术与开始只在高端单片机产品，现在已经延伸到中低端单片机上了，这就是工业产品的消费化趋势。AVR、PIC 单片机都支持 LCD、触摸传感功能。Atmel 的 Qtouch 技术与 PicoPowerMCU 和触摸软件库形成低成本方案。伴随着互联网的广泛应用，各种有线和无线的通信方式与单片机结合的越发紧密。CAN、USB、Ethernet 已经成为 32 位单片机的基本组成部分。无线技术在工业和消费电子产品中的应用越来越多。

如 TI 公司的 CC2430，称为无线单片机，它是一种集成了单片机和无线收发模块的 SOC。

（3）大容量。为了适应复杂控制领域的要求，须运用新的工艺，使片内存储器大容量化。目前，单片机内 ROM 最大可达 64KB，RAM 最大为 2KB。

（4）高速化。主要是指进一步改变 CPU 的性能，加快指令运算的速度和提高系统控制的可靠性。采用精简指令集（RISC）结构和流水线技术，可以大幅度提高运行速度。现指令速度最高者已达 100MIPS（Million Instruction Per Seconds，即兆指令每秒），并加强了位处理、中断和定时控制功能。这类单片机的运算速度比标准的单片机高出 10 倍以上。由于这类单片机有极高的指令速度，可以使用软件模拟其 I/O 功能，由此引入了虚拟外设的新概念。

（5）低价格、小容量。以 4 位、8 位机为中心的小容量、低价格化也是发展动向之一。这类单片机的用途是把以往用数字逻辑集成电路组成的控制电路单片化，可广泛用于家电产品。

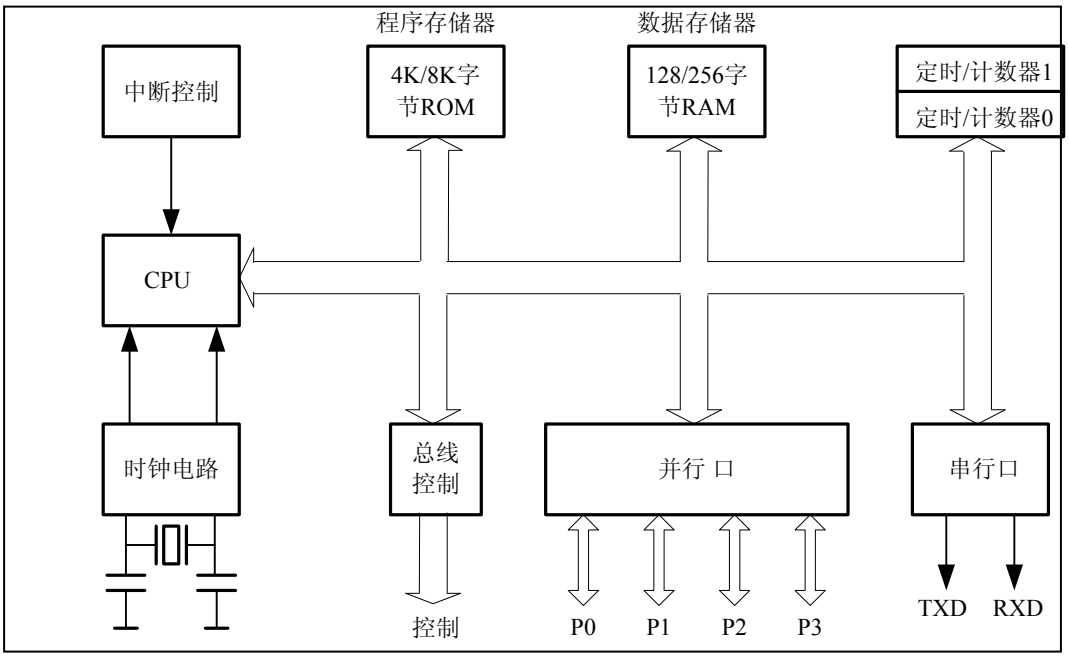
第 1 章　MCS-51 单片机的硬件结构

1.1　MCS-51 单片机的内部结构、引脚定义及外部总线

1.1.1　内部结构

MCS-51 系列单片机内部采用模块式结构，其结构组成框图如图 1.1-1 所示。

图 1.1-1　MCS-51 系列单片机组成框图



由图 1.1-1 可见，MCS-51 系列单片机主要由以下部件通过片内总线连接而成：中央处理器（CPU）、数据存储器（RAM）、程序存储器（ROM）、并行输入/输出口（P0 口~P3 口）、串行口、定时器/计数器、中断控制、总线控制及时钟电路。

1.1.2　引脚定义

引脚是单片机和外界进行通信的通道连接点，用户只能通过引脚组建控制系统。从应用的角度来看，引脚的应用是单片机应用的一个重要基础。因此熟悉引脚是学习应用单片机的基础。

MCS-51 系列单片机的引脚封装主要有： PDIP40、PLCC44 和 PQFP/TQFP44。不同封装的芯片其引脚的排列位置有所不同，但他们的功能和特性都相同。方形封装（PLCC44 和 POFP/TQFP44）有 44 引脚，其中 4 个 NC 为空引脚。采用 40 引脚 PDIP 封装的 80C51 单片机的引脚排列及逻辑符号如图 1.1-2 所示。

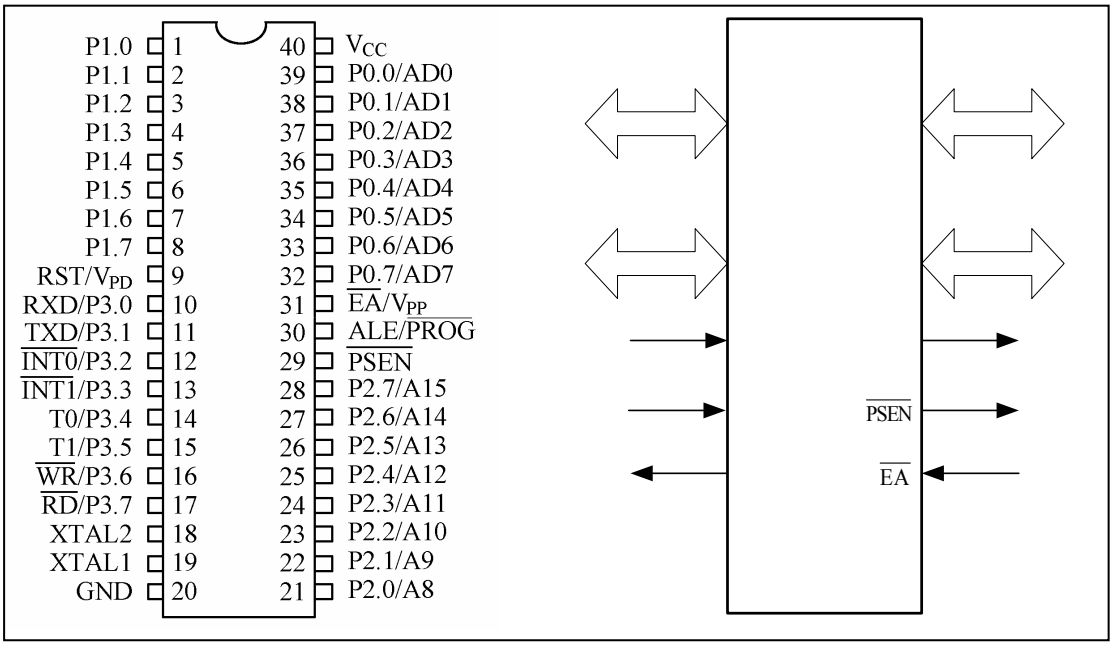
由于工艺及标准化等原因，芯片的引脚数量是有限的，但单片机为实现控制所需要的信号数目却远远超过其引脚数目。为解决这一矛盾，单片机的某些信号引脚被赋以双重功能。

1）电源及电源复位引脚：

（1）V_{CC}（40 脚）：正常操作时接+5V 直流电源。

（2）V_{SS} (20 脚)：接地端。

图 1.1-2　40 引脚 PDIP 封装的 80C51 单片机的引脚排列及逻辑符号图



（3）RST/V_{PD}（9 脚）：复位信号输入端。在该引脚上输入一定时间（约两个机器周期）的高电平将使单片机复位。该引脚的第二功能是 V_{PD}，即备用电源输入端。当主电源发生故障，降低到低电平规定值时，可将+5V 备用电源自动接入 V_{PD} 端，以保护片内 RAM 中的信息不丢失，使复电后能继续正常运行。

（4）$\overline{EA}$ /V_{PP}（31 脚）：访问程序存储器控制信号/编程电源输入。当 $\overline{EA}$ 保持高电平时，访问内部程序存储器，访问地址范围在 0~4KB 内；当 PC（程序计数器）值超过 0FFFH，即访问地址超出 4KB 时，将自动转向执行外部程序存储器内的程序；当 $\overline{EA}$ 保持低电平时，不管单片机内部是否有程序存储器，则只访问外部程序存储器（从 0000H 地址开始）。由此可见，对片内有可用程序存储器的单片机而言，$\overline{EA}$ 端应接高电平，而对片内无程序存储器的单片机，可将 $\overline{EA}$ 接地。

对于 EPROM 型单片机，在 EPROM 编程期间，此引脚用于施加 21V 的编程电源（V_{PP}）。

2）时钟振荡电路引脚 XTAL1 和 XTAL2：

（1）XTAL1（19 脚）：外接石英晶体和微调电容引脚 1。它是片内振荡电路反向放大器的输入端。采用外部振荡器时此引脚接地。

（2）XTAL2（18 脚）：外接石英晶体和微调电容引脚 2。它是片内振荡电路反向放大器的输出端。采用外部振荡器时此引脚为外部振荡信号输入端。

3) ALE/$\overline{PROG}$（30 脚）：低 8 位地址锁存控制信号/编程脉冲输入。在系统扩展时，ALE 用于把 P0 口输出的低 8 位地址锁存起来，以实现低 8 位地址和数据的隔离。在访问外部程序存储器期间，ALE 信号两次有效；而在访问外部数据存储器期间，ALE 信号一次有效。对于 EPROM 型单片机，在 EPROM 编程期间，此引脚用于输入编程脉冲 $\overline{PROG}$。

4）$\overline{PSEN}$（29 脚）：外部程序存储器的读选通信号输出端，低电平有效。在从外部程序存储器取指令（或常数）期间，此引脚定时输出负脉冲作为读取外部程序存储器的信号，每个机器周期 $\overline{PSEN}$ 两次有效，此时地址总线上送出的地址为外部程序存储器地址；在此期间，如果访问外部数据存储器和内部程序存储器，不会产生 $\overline{PSEN}$ 信号。

5）并行双向输入/输出(I/O)口引脚：

（1）P0 口的 P0.0~P0.7 引脚（39~32 脚）：8 位通用输入/输出端口和片外 8 位数据/低 8 位地址复用总线端口。

（2）P1 口的 P1.0~P1.7 引脚（1~8 脚）：8 位通用输入/输出端口。

（3）P2 口的 P2.0~P2.7 引脚（28~21 脚）：8 位通用输入/输出端口和片外高 8 位地址总线端口。

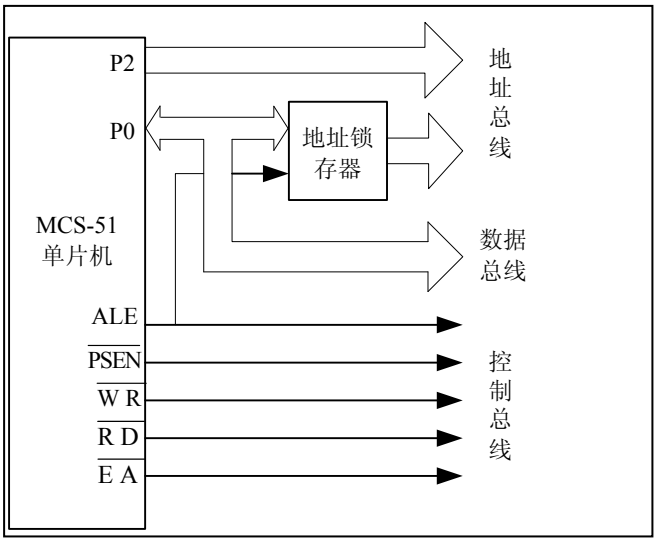
（4）P3 口的 P3.0~P3.7 引脚（10~17 脚）：8 位通用输入/输出端口，具有第二功能。

1.1.3 外部总线构成

所谓总线，就是连接单片机与各外部器件的一组公共的信号线。当系统要求扩展时，单片机要与一定数量的外部器件和外围设备连接。如果各部件及每一种外围设备都分别用各自的一组线路与 CPU 直接连接，那么连线将会错综复杂，甚至难以实现。为了简化硬件电路的设计和系统结构，常用一组线路，并配以适当的接口电路来与各个外部器件和外围设备连接，这组共用的连接线路就是总线。采用总线结构便于扩展外部器件和外围设备，而统一的总线标准则使不同设备间的互连更容易实现。

利用片外引脚可以构造 MCS-51 系列单片机的三总线结构。单片机的引脚除了电源端 VCC、接地端 VSS、复位端 RST、晶振接入端 XTAL1 和 XTAL2、通用 I/O 口的 P1.0~P1.7 以外，其余的引脚都是为实现系统扩展而设置的。用这些引脚构造的单片机系统的三总线结构如图 1.1-3 所示。

图 1.1-3　MCS-51 系列单片机片外三总线结构



1）地址总线（Address Bus，AB）：MCS-51 系列单片机总共有 16 根地址线 A15~ A0，片外存储器可寻址范围达 64KB（2^{16}=65536 字节），由 P2 口直接提供高 8 位地址 A15~ A8，P0 口经地址锁存器提供低 8 位地址 A7~ A0。

2）数据总线（Data Bus，DB）：MCS-51 系列单片机总共有 8 根数据线 D7~D0，全由 P0 口提供。由于 P0 口是分时复用总线，分时输送低 8 位地址（通过地址锁存器锁存）和高 8 位数据信息。

3）控制总线（Control Bus，CB）：控制总线由 P3 口的第二功能 $\overline{\text{WR}}$ (P3.6)、$\overline{\text{RD}}$ (P3.7) 和 3 根独立的控制线 $\overline{\text{EA}}$ 、ALE、$\overline{\text{PSEN}}$ 组成。

1.2 MCS-51 单片机的中央处理器

1.2.1 CPU的结构组成

CPU 是单片机的核心部件，由运算器和控制器组成，用以进行各种算术和逻辑运算，并实现数据的传送。

运算器包括算术逻辑单元部件 ALU（Arithmetic and Logic Unit）、位处理器、8 位累加器 A、寄存器 B、两个 8 位暂存寄存器 TMP1 和 TMP2 及程序状态字寄存器 PSW（8 位）等。ALU 不仅可以实现 8 位数据的加、减、乘、除、增量、减量、十进制调整、比较等算术运算和与、或、异或、求补等逻辑运算，同时还具有一般微机所不具备的位处理功能，可对位变量进行置位、清零、求补及与、或等操作。

控制器是 CPU 的神经中枢，它识别指令并根据指令性质协调单片机内部各组成单元自动协调地工作。主要包括程序计数器 PC、PC 增量器、指令寄存器、指令译码器、定时及控制逻辑电路等。其功能是以主振频率为基准产生时钟信号，向单片机内部各组成单元发出各种微控制信号，控制指令的读入、译码和执行，并对指令执行过程进行定时和逻辑控制。

1.2.2 指令执行的基本步骤

必须先在存储器中载入程序，单片机才可以开始工作。所谓程序，就是为了完成某项工作，将一系列指令有序地组合，而指令则是要求单片机执行某种操作的命令。指令分为操作码和地址码两部分。操作码部分规定了单片机的操作类型；而地址码部分一般是直接或间接地给出了参与操作的数据的存放地址，所以地址码也可以称为操作数。

单片机执行一条指令一般分为两个步骤：取指令阶段和指令执行阶段。CPU 从程序存储器中取出指令操作码，送到指令寄存器，再经指令译码器译码，产生一系列控制信号，以明确该指令执行什么样的操作，以及操作数的存放地址，根据这个地址获取操作数，这是取指令阶段；然后 CPU 按操作码指明的操作类型对获取的操作数进行操作（也可称为运算），这是指令执行阶段。

那么 CPU 如何完成取指令和执行指令操作的全过程呢？

CPU 复位后，程序计数器 PC 会自动地指向第一条指令存放的存储单元的首地址（16 位 PC 中的内容总是 CPU 将要执行的那条指令所在的存储单元的首地址）。在控制信号的控制下，CPU 从该存储单元中取出指令，暂存在指令寄存器中。指令的操作码部分进入指令译码器译码，译码结果通知控制电路发出相应的控制信号来控制 CPU 各部件，以完成这条指令的操作。指令的地址码部分送往操作数地址形成电路，以便形成实际的操作数地址，然后 CPU 再通过总线从该地址所在单元取出操作数送入暂存寄存器，从暂存寄存器送入算术逻辑运算部件 ALU 中。运算结果送指令所指定的单元，同时将运算结果的有关状态送入程序状态字寄存器 PSW 中。

指令的实际执行过程比上述要复杂的多，并且所有的操作都是在精确的时序控制下进行的。在学习时需要注意以下两点：

1）PC 是一个有自动加 1 功能的 16 位计数器。CPU 从存储单元取指令的过程中，每取一个字节的内容，程序计数器 PC 就自动加 1。在取完这条指令后，PC 中的内容就是下一条要执行的指令所在存储单元的首地址了。PC 没有地址，是不可寻址的，因此，用户不能对它进行读和写操作。

2）以上是 CPU 顺序执行指令的操作过程，然而在实际应用中，CPU 有时还要执行程序的转移、子程序的调用和中断响应等操作，那时 PC 中的内容不再是上述情况中简单地加 1，而是根据不同的情况自动地被置入或修改成新的目的地址，从而改变程序的执行顺序。

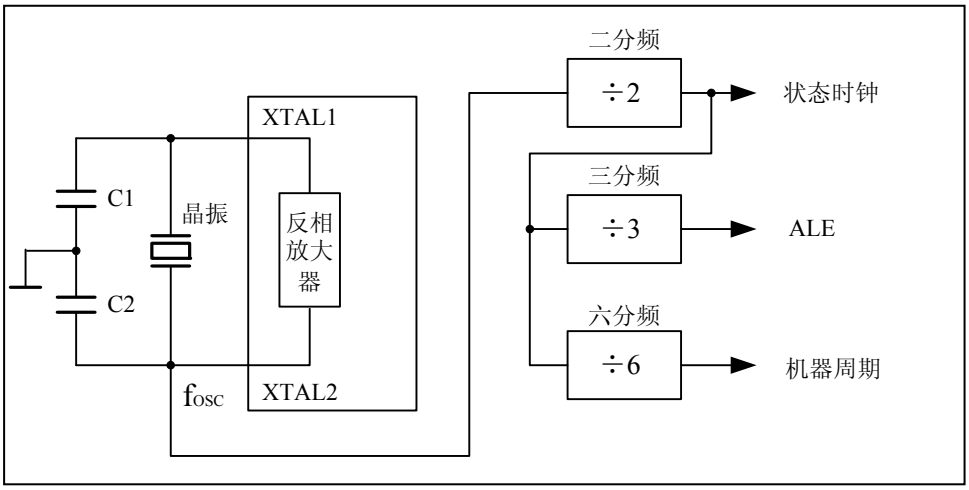
这在后继章节中将详细叙述。

1.2.3 时钟电路及时钟时序单位

1) 时钟电路

单片机本身如同一个复杂的同步时序电路，为了保证同步工作，电路应在唯一的时钟信号控制下，严格地按规定时序工作。而时钟电路就用于产生单片机工作所需要的时钟信号。MCS-51 单片机时钟电路示意图如图 1.2-1 所示。

图 1.2-1 MCS-51 单片机时钟振荡电路示意图



在 MCS-51 芯片内部有一个高增益反相放大器，用于构成振荡器。反相放大器的输入端为引脚 XTAL1，输出端为引脚 XTAL2，在芯片的外部通过这两个引脚跨接晶体振荡器和微调电容 C1、C2 形成反馈电路，可构成稳定的自激振荡器，振荡频率范围通常是 1.2~12MHz。晶体振荡频率高，则系统的时钟频率也高，单片机的运行速度也就快。

在图 1.2-1 中，使用晶体振荡器时，C1、C2 取值 30±10pF；使用陶瓷振荡器时，C1、C2 取值 40±10pF。C1、C2 的取值虽然没有严格的要求，但电容的大小影响振荡电路的稳定性和快速性，通常取值 20~30pF。在设计印制电路板时，晶振和电容等应尽可能靠近芯片，以减少分布电容，保证振荡器振荡的稳定性。

振荡电路产生的振荡脉冲并不直接使用，而是经分频后再为系统所用。振荡脉冲在片内通过一个时钟发生电路二分频后才作为系统的时钟信号。片内时钟发生电路实质上是一个二分频的触发器，其输入来自振荡器，输出为二相时钟信号，即状态时钟信号，其频率为 $f_{osc}/2$；状态时钟三分频后为 ALE 信号，其频率为 $f_{osc}/6$；状态时钟六分频后为机器周期，其频率为 $f_{osc}/12$。

也可以由外部时钟电路向片内输入脉冲信号作为单片机的振荡脉冲。这时外部脉冲信号是经 XTAL1 引脚引入的，而 XTAL2 引脚悬空或接地。对外部信号的占空比没有要求，但高低电平持续的时间不应小于 20ns。这种方式常用于多块芯片同时工作，便于同步。其外部脉冲接入方式如图 1.2-2 所示。

2）时钟时序单位

所谓时序，是指在指令执行过程中，CPU 的控制器所发出的一系列特定的控制信号在时间上的先后关系。CPU 发出的控制信号有两类：一类是用于单片机内部的，用户不能直接接触此类信号，不必对它作过多了解；另一类是通过控制总线送到片外的，人们通常以时序图的形式来表示相关信号的波形及出现的先后次序。为了说明信号的时间关系，需要定义时序单位。MCS-51 的时序单位共有四个，从小到大依次是拍节、状态、机器周期和指令周期。如图 1.2-3 所示。

图 1.2-2　MCS-51 单片机外部时钟输入接线图

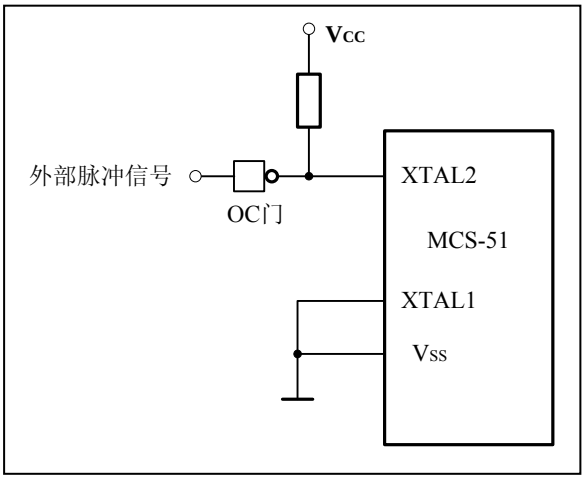
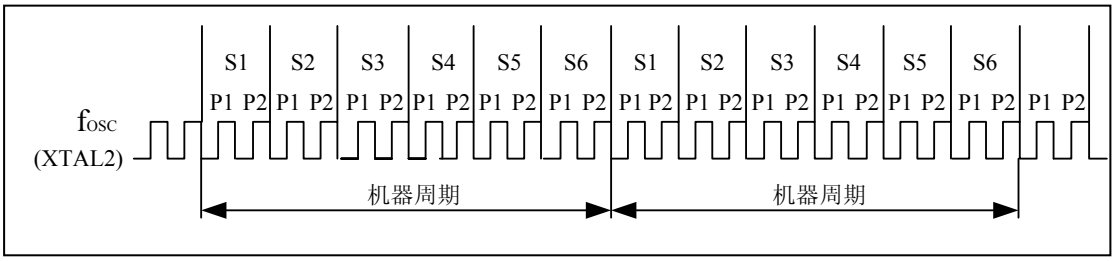


图 1.2-3　拍节、状态、机器周期的关系



（1）拍节与状态

把振荡脉冲的周期（晶振周期）定义为拍节（用“P”表示），振荡脉冲经过二分频后，就是单片机的状态信号（用“S”表示）。这样，一个状态信号包含两个振荡脉冲，即两个拍节。在每个状态的前半周期拍节 1（P1）有效；后半周期拍节 2（P2）有效。

每个状态 S 有两个节拍（相）P1 和 P2，CPU 就以两相时钟 P1 和 P2 为基本节拍“指挥”单片机各部件协调工作。

（2）机器周期

一个机器周期是指 CPU 访问存储器一次所需要的时间。MCS-51 单片机有固定的机器周期，规定一个机器周期为 6 个状态，相当于 12 个拍节，即 12 个振荡脉冲的周期，例如，若振荡脉冲频率为 6MHz，则振荡脉冲周期为 1/6000000s=1/6μs，状态为 1/6×2=1/3μs，机器周期为 1/6×12=2μs。

（3）指令周期

单片机执行一条指令所需要的时间称之为指令周期，它是最大的时序定时单位。指令周期以机器周期的数目来表示。在 MCS-51 系统中，有单周期指令、双周期指令和 4 周期指令。4 周期指令只有乘、除两条指令，其余都是单周期或双周期指令。指令的运算速度和它的机器周期数直接相关，机器周期数少则执行速度快。在编程时，应注意选用具有同样功能而机器周期数少的指令。

1.3　MCS-51 单片机的内部存储器

1.3.1　存储器结构及地址分配

存储器是组成单片机的主要部件之一，其功能是存储信息（数据和程序）。存储器按其存储方式可以分成两大类：一类为随机存储器 RAM；另一类是只读存储器 ROM。

CPU 在运行过程中可对 RAM 随时进行数据的写入和读出，但在关闭电源时，RAM 中

所存储的信息会丢失，所以 RAM 只能用来存放暂时性的输入/输出数据、运算中的结果等。RAM 也因此常被称为数据存储器。

而 ROM 是写入数据后不能更改只能读出的存储器。在断电后，ROM 中的信息保留不变，所以 ROM 用来存放固定的程序和固定的数据，如系统程序、常数、表格等。ROM 也因此常被称为程序存储器。常用的 ROM 根据其编程方式不同可分为以下 5 种类型：掩膜型 ROM(MROM)、一次性可编程 ROM（PROM，又称为 OTPROM)、紫外线可擦 ROM（EPROM）、电可擦 ROM（EEPROM）、闪速 ROM（Flash ROM）。

多数单片机系统的存储器配置与通用微机的不同，程序存储器地址空间和数据存储器地址空间相互独立，并通过各自的数据总线与 CPU 相连，以加快程序的执行速度。而通用微机系统的程序存储器和数据存储器往往共用同一存储区，统一编址。

MCS-51 单片机的存储器可分为 5 类：

1）片内程序存储器；

2）片外程序存储器；

3）片内数据存储器；

4）特殊功能寄存器；

5）片外数据存储器；

MCS-51 单片机存储器的地址空间可分为 3 个：

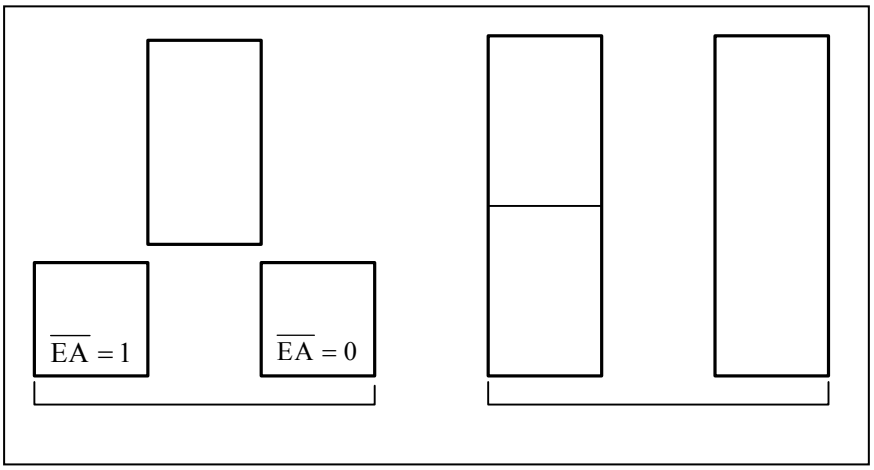
1）片内片外统一编址的 64KB 的程序存储器地址空间（16 位地址 0000H~FFFFH）

2）片内数据存储器与特殊功能寄存器统一编址的 256B 内部数据存储器地址空间（8 位地址 00H~7FH，80H~FFH）；

3）64KB 片外数据存储器地址空间（16 位地址 0000H~FFFFH）。

MCS-51 单片机的存储结构有两个重要的特点：一是把数据存储器和程序存储器截然分开，采用哈佛结构，不同于通用微机的普林斯顿结构；二是存储器有内外之分。在以后的学习中我们将会了解到，在访问不同的地址空间时，采用的控制信号和存取指令也是不同的。

图 1.3-1　MCS-51 单片机存储器地址结构



对于复杂一些的单片机应用系统，当片内所固有的程序存储器和数据存储器不能满足需要时，往往还要在单片机芯片外另行扩展存储器，称之为片外存储器。外部存储器的扩展是构建单片机系统的重要内容。我们将在后继章节中对其进行详细的介绍。

1.3.2　内部程序存储器

MCS-51 单片机有片内有程序存储器和片内无程序存储器之分。如 80C51 单片机片内设置有 4KB 的掩膜型 ROM。其地址范围是 0000H~0FFFH。当 $\overline{EA}$ 接高电平时，指令寻址

地址在 0000H~0FFFH 时，CPU 访问内部存储单元，如果指令寻址地址大于 0FFFH 时，CPU 自动转向对片外存储器的访问。当 $\overline{EA}$ 接低电平（接地）时，CPU 只能访问外部程序存储器，这时外部程序存储器的地址从 0000H 开始编址。80C31、80C32 单片机因其片内无程序存储器，所以只能采用将 $\overline{EA}$ 接地的方法。

程序存储器的一些低端地址单元是用作存储特定程序的入口地址的，其中 0000H~0002H 这 3 个单元是系统的启动单元。系统复位后，程序计数器 PC 中的复位值是 0000H，所以系统是从 0000H 单元开始取指令执行程序的。在编程时，应当在这 3 个单元中存放一条无条件转移指令，以转向执行指定的程序。而地址为 0003H~002AH 的 40 个单元是中断服务程序地址区，分成 5 段，每段 8 个单元，分别分配给 5 个中断源，具体分配如下：

- 外部中断 0（$\overline{INT0}$）服务程序地址区　　0003H~000AH
- 定时器/计数器 0 中断服务程序地址区　　000BH~0012H
- 外部中断 1（$\overline{INT1}$）服务程序地址区　　0013H~001AH
- 定时器/计数器 1 中断服务程序地址区　　001BH~0022H
- 串行口中断服务程序入口地址区　　0023H~002AH

中断响应后，系统按中断类型自动转到各中断服务程序的首地址单元去执行程序。在中断服务程序地址区应当存放中断服务程序，但在实际应用中，8 个存储单元往往难以存下一个完整的中断服务程序，所以，通常是从该中断服务程序的首地址开始，存放一条无条件转移指令，以便响应中断后，通过该地址区转到中断服务程序的实际入口地址单元中去。

1.3.3　内部数据存储器

MCS-51 单片机内部有 128 或 256 字节的 RAM 用作数据存储器（不同的型号有区别），它们均可读写，部分单元还可以位寻址。

80C51 单片机的片内数据存储器共有 256 个字节，地址范围为 00H~7FH 单元的 128 个字节作为用户 RAM 区，分成工作寄存器区、位寻址区、通用 RAM 区三部分；地址范围为 80H~FFH 单元的 128 个字节作为特殊功能寄存器区（SFR）。如图 1.3-2 所示。

图 1.3-2　80C51 片内 RAM 地址空间

1.3.3.1 用户 RAM 区

1）工作寄存器区

地址范围在 00H~1FH 的 32 个字节可分成 4 个工作寄存器组，每组占 8 个字节，具体划分如图 1.3-2 所示。

每个工作寄存器组都有 8 个寄存器，均分别称为 R0、R1、R2、R3、R4、R5、R6、R7。但在程序运行时，只允许一个工作寄存器组工作，称为当前工作寄存器组，所以每组之间不会因为名称相同而混淆出错。

用哪组工作寄存器作为当前工作寄存器组，可由特殊功能寄存器中程序状态字寄存器 PSW 中的 RS1、RS0 共 2 位二进制数组合来决定，见表 1.3-1 所示。

表 1.3-1　工作寄存器组的选择

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

注意以下几点：

（1）RS1、RS0 的状态可由写指令来进行设置，以选择不同的工作寄存器组；

（2）复位后自然选中第 0 组工作寄存器；

（3）从某一工作寄存器组换至另一个字寄存器组时，原工作寄存器中的内容将被屏蔽保护；

（4）若程序中不需要用 4 组工作寄存器，那么不用的工作寄存器可作为一般 RAM 使用。

2）位寻址区

内部 RAM 的 20H~2FH 地址范围共 16 个字节单元为位寻址区，既可作为一般的 RAM 进行字节寻址，也可以对单元中的每一位进行位寻址（称 MCS-51 具有布尔处理功能），这种位寻址能力是一般微机所没有的。16 个字节单元共 128 位，每位有位地址，以便对位进行操作，所以共有 128 个位地址（00H~7FH）。CPU 能直接寻址这些位，执行置 1、清 0、求“反”、转移、传送等操作。位地址分配见表 1.3-2 所示，其中 MSB（Most Significant Bit）表示最高有效位，LSB（Lost Significant Bit）表示最低有效位。

00H~7FH 共 128 个位地址与内部 RAM 中 128 个字节地址 00H~7FH 重叠，在应用中可通过指令的类型来区分它们是位地址还是字节地址。

3）通用 RAM 区

在内部数据存储器的 128 个字节单元中，工作寄存器区和位寻址区占去了 48 个单元，剩下的 80 个单元以存储单元的形式来用，没有其它的规定和限制，称为通用 RAM 区，用来存放用户数据或作为堆栈区使用。CPU 对该区中的每个 RAM 单元只能实现字节寻址。

1.3.3.2 特殊功能寄存器

MCS-51 中的特殊功能寄存器（Special Function Registers：SFR），也称为专用寄存器，它们离散地分布在内部 RAM 地址为 80H~FFH 的 128 个字节的存储空间中，构成了 SFR 存储块。SFR 反映了 MCS-51 单片机的运行状态，其功能已做了专门的规定，用户不能修改其结构。

MCS-51 中的特殊功能寄存器共有 21 个，在这 21 个 SFR 中，字节地址能被 8 整除（即

十六进制数的地址码尾数为 0 或 8 的）的 11 个单元具有位寻址能力。有效的位地址共有 82 个，可用位地址、位符号、单元地址.位序和寄存器名.位序 4 种方法来表示，如 D7H、C、D0H.7、PSW.7 都表示同一个位，即程序状态字寄存器的最高位。但一般用位符号来表示。各特殊功能寄存器的符号及位地址见表 1.3-3 所示。

表 1.3-2　内部 RAM 位寻址区的位地址分配表

字节地址	MSB←位地址→LSB							
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
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

这些特殊功能寄存器分别用于以下各功能单元：

1)CPU：累加器 A、B 寄存器、程序状态字寄存器 PSW、堆栈指针 SP、数据指针 DPTR(由 DPH 和 DPL 两个 8 位寄存器组成)。

2）并行口：P0、P1、P2、P3。

3）串行口：串行口控制寄存器 SCON、串行数据缓冲器 SBUF、电源控制寄存器 PCON。

4）中断系统：中断允许控制寄存器 IE、中断优先级控制寄存器 IP。

5）定时器/计数器：定时器/计数器方式控制寄存器 TMOD、定时器/计数器控制寄存器 TCON、定时器/计数器 0（TH0、TL0）、定时器/计数器 1（TH1、TL1）。

下面介绍几个常用的特殊功能寄存器，其余的将在后继章节中陆续介绍。

1）累加器 A(或 ACC：Accumulator)

累加器 A 是 8 位寄存器，地址为 E0H。它是程序中最繁忙的特殊功能寄存器，用于向 ALU 提供操作数、存放运算的中间结果、数据传送的中间站等。大部分的数据操作都会通过累加器 A 进行，就像一个交通要道。由于它的功能特殊，地位重要，因此近年来出现的单片机，有的集成了多累加器结构，或者使用寄存器阵列来代替累加器，即赋予更多寄存器以累加器的功能，以解决累加器的“交通堵塞”问题，提高单片机程序的执行效率

2）B 寄存器

寄存器 B 也是 8 位寄存器，地址为 F0H。它主要用于乘、除运算。乘法指令中，两个因数分别取自累加器 A 和寄存器 B，其运算结果低 8 位存放于累加器 A 中，高 8 位存放于寄存器 B 中；除法指令中，被除数取自累加器 A，除数取自寄存器 B，运算结果的商存放于累加器 A 中，余数存放于寄存器 B 中。不做乘除运算时，寄存器 B 也可以作为通用寄存

器用。

表 1.3-3　特殊功能寄存器的地址及符号表

SFR	位地址/位符号（寄存器名.位序）								字节地址
B	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H	F0H
	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	
A	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H	E0H
	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	
PSW	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H	D0H
	CY	AC	F0	RS1	RS0	OV	/	P	
IP	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H	B8H
	/	/	/	PS	PT1	PX1	PT01	PX0	
P3	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H	B0H
	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	
IE	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H	A8H
	EA	/	/	ES	ET1	EX1	ET0	EX0	
P2	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H	A0H
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
SBUF									99H
SCON	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H	98H
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
P1	97H	96H	95H	94H	93H	92H	91H	90H	90H
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	
TH1									8DH
TH0									8CH
TL1									8BH
TL0									8AH
TMOD	GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0	89H
TCON	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H	88H
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
PCON	SMOD	/	/	/	GF1	GF0	PD	IDL	87H
DPH									83H
DPL									82H
SP									81H
P0	87H	86H	85H	84H	83H	82H	81H	80H	80H
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	

3）程序状态字寄存器 PSW（Program Status Word）

PSW 为 8 位寄存器，地址为 D0H，用于存放程序运行的状态信息，供程序查询和判别时用。其中有些位状态是根据指令执行的结果，由硬件自动设置的，而有些位状态是使用软件方法设定的。位状态可以用专门的指令进行测试，也可用指令读出。PSW 的各位定义如下：

位序	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
位标志	C_y	AC	F0	RS1	RS0	OV	/	P

① Cy：进位/借位标志位。此位有两个功能：一是存放执行算术运算时的进位/借位标志，可被硬件或软件置位或清 0，如进行加、减运算时，若运算结果在最高位有进位或借位时，Cy 被硬件自动置 1，反之则自动置 0；二是在位操作中做位累加器使用。

② AC：辅助进位/借位标志位，又称为半进位标志位。当进行加、减运算时如果由低 4 位向高 4 位进位或借位，则 AC 被硬件自动置 1，反之则自动置 0。AC 常用于二~十进制调整。

③ F0 ：用户标志位，可由用户自定义其含义。F0 通常不是单片机在执行指令过程中自动形成的，而是用户根据执行程序的需要通过传送指令设置的。用户通过对 F0 置 1 或置 0 来设定程序的走向。

④ RS1、RS0：工作寄存器组（区）选择标志位。8051 共有 4 组 8×8 位工作寄存器，每组均命名为 R0~R7，但每组在 RAM 中的物理地址不同，用户可通过软件改变 RS1 和 RS0 的组合内容来选择 R0~R7 在片内 RAM 中的实际物理地址，见表 1.3-4 所示。

表 1.3-4　工作寄存器组（区）选择标志位

RS1（PSW.4）	RS0（PSW.3）	选定的当前使用的工作寄存器组（区）	片内 RAM 地址	通用寄存器名称
0	0	第 0 组	00H-07H	R0-R7
0	1	第 1 组	08H-0FH	R0-R7
1	0	第 2 组	10H-17H	R0-R7
1	1	第 3 组	18H-1FH	R0-R7

⑤ OV：溢出标志位，带符号数加减运算中，若结果超出了累加器 A 所能表示的有效范围（-128~+127），即产生溢出。如果运算结果错误，则 OV=1；若运算结果正确，则 OV=0。

⑥ P：奇偶校验位。用于指示运算结果（存放在累加器 A 中）中“1”的个数的奇偶性，若累加器 A 中 1 的个数为奇数个则 P=1，若累加器 A 中 1 的个数为偶数个则 P=0；

⑦ 位 1（PSW.1）是无效位，此位未定义。

4）数据指针 DPTR（Data Pointer，16 位）

由高字节 DPH 和低字节 DPL 两个 8 位特殊功能寄存器组成，它们的地址分别是 83H 和 82H。称它为地址指针，是因为可用它存放一个 16 位地址，以便用特定的指令形式对片外数据存储器或程序存储器进行 64KB 范围内的数据操作。DPTR 的使用比较灵活，它既可以当作 16 位寄存器使用，也可以作为两个 8 位寄存器使用。

5）I/O 口专用寄存器（P0、P1、P2、P3）

MCS-51 片内有四个 8 位并行 I/O 口 P0、P1、P2、P3，每个 I/O 口内部都有一个 8 位数据输出锁存器和一个 8 位数据输入缓冲器，4 个数据输出锁存器与端口号同名，皆为特殊功能寄存器中的一个，即为 4 个 I/O 口寄存器 P0、P1、P2、P3。MCS-51 单片机并没有专门的 I/O 口操作指令，而是把 I/O 口也当作一般的寄存器来用，通过 MOV 指令来传送数据，其优势在于 4 个并行 I/O 口还可以当作寄存器直接寻址，参与其他操作。

6）串行数据缓冲器（SBUF）

SBUF 用来存放需要发送和接收的数据。它由两个独立的寄存器组成，一个是发送缓冲器，另一个是接收缓冲器，要发送和接收的操作其实都是对串行数据缓冲器 SBUF 进行的。

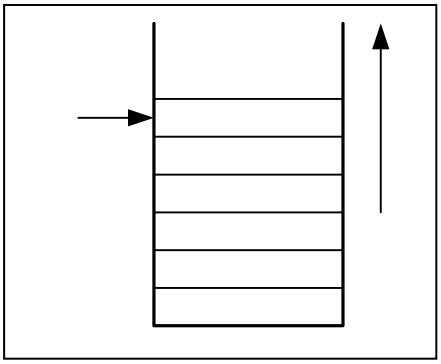
1.3.3.3　堆栈

1）堆栈的概念

堆栈就是在单片机内部 RAM 中，从某个选定的存储单元开始划定的一个地址连续的区域。设立堆栈的目的是用于数据的暂存，中断、子程序调用时断点和现场的保护与恢复。这

个区域本身没有任何特殊之处，它就是内部 RAM 的一部分，也是用来存放数据的，不同的是这个区域以选定的某个存储单元作为栈底，只允许向一个方向写入数据，最后一个写入数据的存储单元称为栈顶。堆栈的生成有两种情况：向高地址方向写入数据生成的堆栈称为向上生长型堆栈；反之称为向下生长型堆栈。MCS-51 单片机设置的是向上生长型堆栈。数据写入堆栈为插入（PUSH）运算，通常称为入栈；数据从堆栈中读出为删除（POP）运算，通常称为出栈。按堆栈的规定，入栈和出栈只能在栈顶一端进行。因此，向上生长型的堆栈栈顶存储单元的地址必定随着数据的入栈而递增，随着数据的出栈而递减，并由此规定可知，对栈顶进行入栈和出栈操作必定是“先进后出”或“后进先出”。向上生长型堆栈如图 1.3-3 所示。

图 1.3-3　向上生长型堆栈



2）堆栈指针 SP（Stack Pointer，8 位）

堆栈指针 SP 是一个 8 位特殊功能寄存器，用于指示堆栈的栈顶地址， SP 总是指向堆栈栈顶,它决定了栈顶在内部 RAM 中的物理位置。所以，每当执行一次入栈操作（PUSH 指令）时，SP 就会在原来值的基础上自动加 1；每当执行一次出栈操作（POP 指令）时，SP 就会在原来值的基础上自动减 1。当堆栈中位空（无数据）时，栈顶地址等于栈底地址，两者重合，SP 的内容即为栈底地址。栈底地址一旦设定，就固定不变，直至重新设置。每当数据进栈或出栈时，SP 的内容都随之变化，即栈顶随之浮动。

数据入栈的操作过程为：先将 SP 加 1（SP←SP+1），然后将要入栈的数据存放在 SP 指定的存储单元中。将数据从堆栈中弹出时，先将 SP 寄存器指定的存储单元的内容传送到 POP 指令给定的寄存器或内部 RAM 单元中，然后 SP 减 1（SP←SP-1）。

3）堆栈的开辟

单片机的堆栈只能开辟在片内数据存储器中，称为内堆栈形式。系统复位后，SP 中的复位值是 07H，即 SP 指向片内数据存储器中地址为 07H 的存储单元，这个存储单元正好在单片机片内数据存储器的工作寄存器区。为了避免占用宝贵的工作寄存器区和位寻址区，所以，在程序设计时，常用指令对 SP 中的复位值进行修改，把堆栈开辟在通用 RAM 区，即在 30H~7FH 地址范围内开辟一个地址连续的存储区域来实现堆栈。

1.4　并行端口

1.4.1　端口功能

MCS-51 单片机有 4 个 8 位并行双向输入/输出（I/O）口，每个端口都包括有一个输出锁存器（属于特殊功能寄存器，分别记作 P0、P1、P2、P3）、一个输出驱动器和输入缓冲器。通常把 4 个端口笼统地称为 P0 口、P1 口、P2 口、P3 口。每个端口占 8 个引脚，共 32 个引脚。每个端口均可按字节输入、输出，也可以按位进行输入、输出。它们是典型的 CPU 与外部设备的输入/输出端口，可以很方便地实现 CPU 与外部设备及芯片的信息交换。各端口

功能如下：

1）P0 口（P0.0~P0.7）：双功能 8 位 I/O 口，字节地址为 80H，位地址为 80H~87H。它既可以作为输入/输出端口使用，又可以在单片机扩展外部设备或芯片时作为地址/数据总线使用。在作为地址/数据总线使用时，可分时传送低 8 位地址和 8 位数据。

2）P1 口（P1.0~P1.7）：唯一的一个单功能并行 I/O 口，字节地址为 90H，位地址为 90H~97H。只能用作通用的数据输入/输出端口。在 EPROM 编程和程序验证时，它接收低 8 位地址。（注：在增强型 MCS-51 芯片中，P1.0 和 P1.1 引脚具有第二输入/输出功能，即除了可作为一般的 I/O 引脚使用外，P1.0 引脚还作为定时/计数器 T2 的计数输入端或定时时钟输出端，而 P1.1 引脚作为定时/计数器 T2 外部触发输入端 T2EX。）

3）P2 口（P2.0~P2.7）：双功能 8 位并行 I/O 口，字节地址为 A0H，位地址为 A0H~A7H。它可以作为通用的输入/输出端口使用，在单片机扩展外部设备或芯片时，它又可以用作高 8 位地址总线，与 P0 口的低 8 位地址线一起构成 16 位片外地址总线。

4）P3 口（P3.0~P3.7）：双功能 8 位并行 I/O 口，字节地址为 B0H，位地址为 B0H~B7H。

作为通用 I/O 口时，功能与 P1 口相同。在 MCS-51 单片机中，这 8 个引脚还兼有专用功能，见表 1.4-1 所示：

表 1.4-1　P3 口的第二功能定义

引脚名	符号：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 输入
P3.3	$\overline{\text{INT1}}$：外部中断 1 输入
P3.4	T0：定时器/计数器 0 的外部计数脉冲输入
P3.5	T1：定定时器/计数器 1 的外部计数脉冲输入
P3.6	$\overline{\text{WR}}$：片外数据存储器写选通信号输出
P3.7	$\overline{\text{RD}}$：片外数据存储器读选通信号输出

1.4.2　端口原理及操作

MCS-51 单片机的 4 个 I/O 端口的线路设计非常巧妙。了解端口的逻辑电路，不但有利于正确合理地使用端口，而且会给设计单片机的外围电路带来启发。

1）P0 口

（1）结构组成：P0 口某位的结构如图 1.4-1 所示。

P0 口的位电路结构由以下几个部分构成：

① 一个输出锁存器，用于进行输出数据的锁存；

② 两个三态输入缓冲器，分别用于锁存器和引脚数据的输入缓冲；

③ 一个多路开关，它的一个输入来自锁存器，另一个输入是地址/数据信号的反相输出。多路开关在控制信号的控制下能实现对锁存器输出端和地址/数据线之间的切换；

④ 由两个场效应管组成的输出驱动电路。

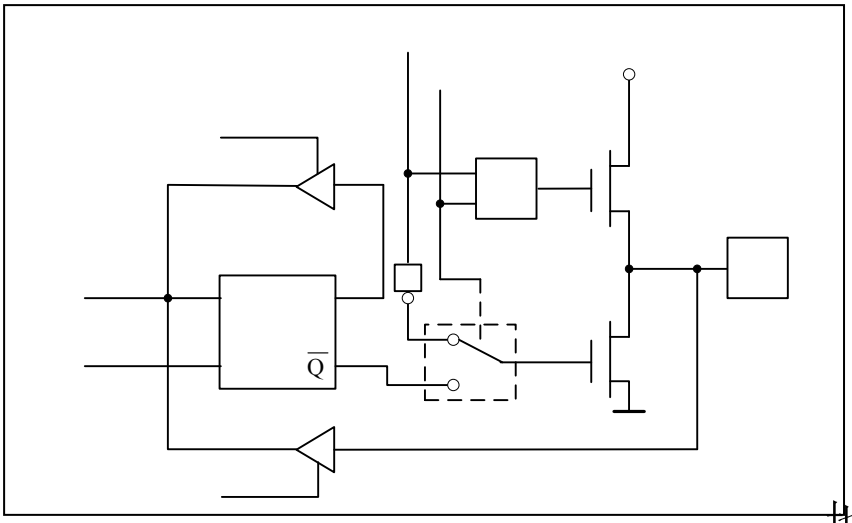
（2）对于内置了 ROM、EPROM、OTPROM 或者 Flash ROM 的 MCS-51 单片机来说，当不使用外部存储器（程序存储器和数据存储器）时，P0 口可作为通用输入/输出口使用，当需要扩展外部存储器时，P0 口用作地址/数据总线。

① P0 口作为通用输入/输出口使用：

当 P0 口用作通用输入/输出口时，电路中控制线上的开关控制信号为 0，多路开关受控

制信号的作用将输出锁存器 $\overline{Q}$ 端与驱动器的上拉场效应管 VT1 的栅极接通，并封锁与门，使与门输出为 0，造成驱动器的上拉场效应管 VT1 截止，使输出驱动器工作在需要外接上拉电阻的漏极开路方式。

图 1.4-1　P0 口的位结构图



◆ P0 口用作输出口：当 P0 口用作输出口时，内部数据总线上的数据在 CPU 写锁存器信号的作用下，由 D 端进入锁存器，经锁存器的 $\overline{Q}$ 端送至场效应管 VT2 的栅极，再经 VT2 反相，使 P0.X 引脚输出状态正好与内部总线上的状态相同（需外接上拉电阻才能有高电平输出）。

◆ P0 口用作输入口：当 P0 口用作输入口时，根据指令的不同，有读锁存器 Q 端的数据（读锁存器）和读引脚上的数据（读引脚）两种读入方式。在执行指令时，究竟是读引脚还是读锁存器，CPU 内部会自行判断发读引脚信号还是读锁存器信号。

② P0 口用作地址/数据总线：

在扩展系统中，P0 口作为地址/数据总线使用时，用作低 8 位地址总线和 8 位数据总线。这时单片机内控制信号为 1，控制多路开关接向反相器的输出端，并使与门的输出状态只由地址/数据的状态决定。

◆ 分时输出低 8 位地址及 8 位数据。P0 口分时输出低 8 位地址及 8 位数据时，若地址/数据总线的状态为 1，则场效应管 VT1 导通，VT2 截止，引脚状态为 1；若地址/数据总线的状态为 0，则场效应管 VT1 截止，VT2 导通，引脚状态为 0。P0.X 的状态正好与地址/数据的状态相同。

◆ 分时输出低 8 位地址及输入 8 位数据。P0 口分时输出低 8 位地址及输入 8 位数据时，先按上述原理输出地址，然后控制信号自动转为 0，使多路开关接向锁存器 $\overline{Q}$ 端，CPU 自动向 P0 口写入 FFH，这时 VT1 和 VT2 同时截止，引脚悬空（高阻状态），在读引脚信号的作用下，传送到引脚上的外部数据经缓冲器三态门 2 读入内部数据总线。

（3）P0 口的特点：

① P0 口是一个双功能的端口：地址/数据复用口和通用 I/O 口。

② 具有高阻抗输入的 I/O 口称为真正的双向口，真正的双向口是具有高电平、低电平和高阻抗三种状态的端口。因此，P0 口作为地址/数据总线复用口时，相当于一个真正的双向 I/O 口。而用作通用 I/O 口时，由于引脚上需要外接上拉电阻，端口不存在高阻（悬空）状态，此时 P0 口只是一个准双向口。

③ 为保证引脚上的信号能正确读入，在读入操作前应首先向锁存器写 1。单片机复位后，

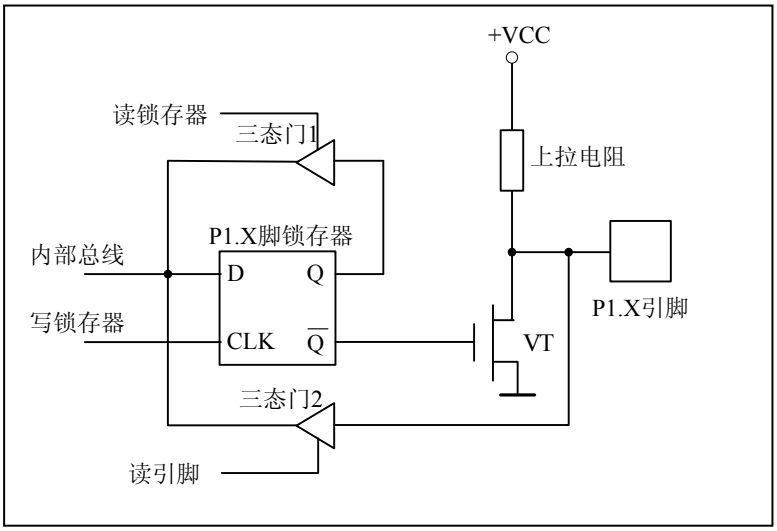
锁存器被自动置 1，但如果 P0 口由原来的输出状态变为输入状态时，也应首先将锁存器置 1 方可执行输入操作。

④ P0 口能驱动 8 个 TTL 负载。

2) P1 口

（1）结构组成：P1 口的位结构如图 1.4-2 所示。

图 1.4-2 P1 口的位结构图



P1 口的位电路结构由以下几个部分组成：

① 一个输出锁存器，用于进行输出数据的锁存；

② 两个三态输入缓冲器，三态门 1 用于读锁存器，三态门 2 用于读引脚；

③ 数据输出驱动电路，由场效应管 VT 和片内上拉电阻组成。

（2）作为通用输出口使用：P1 口作为通用输出口使用时，数据经内部总线、锁存器反相输出端 $\overline{Q}$、VT 管栅极和漏极到 P1.X 引脚。若 CPU 输出 1，在写锁存器时钟作用下，锁存器反相输出端 $\overline{Q}$ 为低电平，场效应管 VT 截止，漏极输出高电平，P1.X 引脚输出 1；若 CPU 输出 0，在写锁存器时钟作用下，锁存器反相输出端 $\overline{Q}$ 为高电平，场效应管 VT 导通，漏极输出低电平（场效应管漏、源之间的导通电阻很小，仅为几十欧到几百欧，而漏极等效上拉电阻一般为数十千欧，分压后，P1.X 引脚电位近似为 0，故 P1.X 引脚输出低电平），P1.X 引脚输出 0。

（3）作为通用输入口使用：P1 口作为通用输入口使用时，根据指令的不同，也有读锁存器和读引脚两种输入方式。读锁存器时，锁存器输出端 Q 的状态经输入缓冲器三态门 1 进入内部总线；读引脚时，必须先向锁存器写 1，使场效应管 VT 截止，P1.X 引脚上的电平状态经输入缓冲器三态门 2 进入内部总线。

（4）P1 口的特点：

① P1 口由于有内部上拉电阻，没有高阻态，所以称为准双向口。作为输出口时，不需要在片外接上拉电阻。

② P1 口读引脚输入时，必须先向锁存器写入 1，其原理与 P0 口相同。

③ P1 口引脚能驱动 4 个 TTL 负载。

3） P2 口

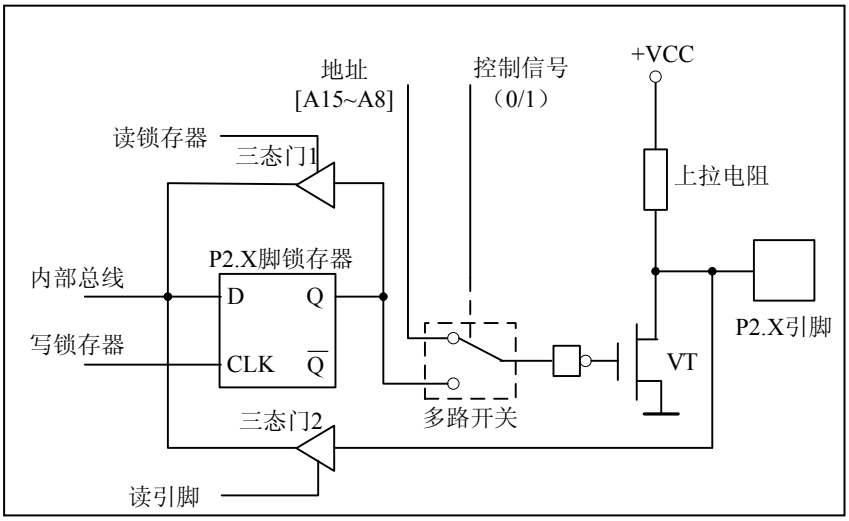
（1）结构组成：P2 口的位结构如图 1.4-3 所示。

P2 口的位电路结构由以下几个部分组成：

① 一个数据输出锁存器，用于进行输出数据的锁存；

② 两个三态输入缓冲器，三态门 1 用于读锁存器，三态门 2 用于读引脚；

图 1.4-3　P2 口的位结构图



③ 一个多路开关，它的一个输入来自锁存器的 Q 端，另一个输入来自内部地址的高 8 位；

④ 数据输出驱动电路，由反相器、场效应管 VT 和片内上拉电阻组成。

（2）P2 口可用作通用输入/输出口，也是一个准双向口。P2 口用作通用输入/输出口时，电路中控制线上的开关控制信号为 0，多路开关受开关控制信号的作用，将开关拨向锁存器的 Q 端。

① P2 口用作输出口时，若 CPU 输出 1，内部数据总线上的数据在写锁存器信号的作用下由 D 端进入锁存器，则 Q=1，Q 端的状态通过多路开关经反相器反相后送至场效应管 VT，使场效应管 VT 截止，所以 P2.X 引脚的输出为 1；同理，若 CPU 输出 0，则 Q=0，场效应管 VT 导通， P2.X 引脚的输出为 0。

② P2 口用作输入口时，根据指令的不同也有读锁存器和读引脚两种输入方式。读锁存器时，锁存器的输出端 Q 的状态经输入缓冲器进入内部总线；读引脚时，必须先向锁存器写 1，使场效应管 VT 截止，外部传送到 P2.X 引脚上的电平状态经输入缓冲器三态门 2 进入内部总线。

（3）P2 口作为高 8 位地址输出线使用时（与 P0 口输出的低 8 位地址一起构成 16 位地址总线），开关控制信号为 1，这时多路开关拨向地址线位置，地址信号经反相器和场效应管两次反相送到 P2.X 引脚。因此，若地址线为 1，则 Q=1，使场效应管 VT 截止，所以 P2.X 引脚的输出为 1；若地址线为 0，则 Q=0，场效应管 VT 导通， P2.X 引脚的输出为 0。

应当注意：当 P2 口的 8 位不需要全部用作地址总线时（根据片外存储器的扩展容量而定），剩余的口线不可以再用作通用输入/输出端口。

（4）P2 口的特点：

① P2 口作为高 8 位地址输出线应用时，与 P0 口输出的低 8 位地址一起构成 16 位地址总线，可寻址 64KB 的地址空间。

② 当 P2 口作为高 8 位地址输出口时，其输出锁存器原锁存的内容保持不变。

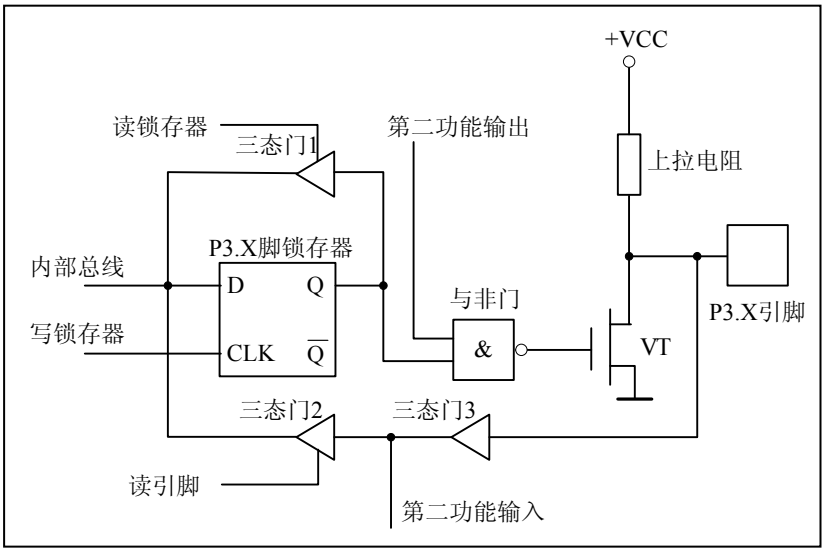
③ 作为通用 I/O 口使用时，P2 口为一准双向口，功能与 P1 口一样。

④ P2 口能驱动 4 个 TTL 负载。

4） P3 口

（1）结构组成：P3 口是双功能 8 位输入/输出端口，内部结构中增加了第二输入/输出功能。其结构组成如图 1.4-4 所示。

图 1.4-4　P3 口的位结构图



P3 口的位电路结构由以下几个部分组成：

① 一个数据输出锁存器，用于进行输出数据的锁存；

② 3 个三态输入缓冲器，三态门 1 用于读锁存器，三态门 2、三态门 3 用于读引脚和第二功能数据缓冲的输入；

③ 数据输出驱动电路，由与非门、场效应管 VT 和片内上拉电阻组成。

（2）当用作通用输入/输出口时，单片机内部硬件自动将第二功能输出端置 1，这时与非门相当于一个反相器，锁存器的输出可通过与非门反相后送至场效应管，再输出到引脚。

当实现输出时，若 CPU 输出 1，则 Q=1，场效应管 VT 截止， P3.X 引脚的输出为 1；若 CPU 输出 0，则 Q=0，场效应管 VT 导通， P3.X 引脚的输出为 0。

当实现读引脚输入时，也需要先对锁存器写 1，使场效应管 VT 截止，传至引脚上的信息通过缓冲器三态门 3 和三态门 2 进入内部总线。

当实现读锁存器输入时，锁存器 Q 端的信息通过缓冲器三态门 1 进入内部总线。

（3）在作为第二功能使用时，单片机内部硬件自动将锁存器 Q 端置 1，打开与非门。

① 输出时，第二功能输出端的信息通过与非门和场效应管 VT 送到引脚。若第二功能输出端输出为 1，则 Q=1，场效应管 VT 截止， P3.X 引脚的输出为 1；若第二功能输出端输出为 0，则 Q=0，场效应管 VT 导通， P3.X 引脚的输出为 0。

② 输入时，第二功能输出端与 Q 端均为高电平 1，场效应管 VT 截止，P3.X 引脚的第二功能信号通过缓冲器三态门 3 送到第二功能输入端。

P3 口第二功能的信号是单片机的主要控制信号，在实际使用时，总是按需要优先选用它的第二功能，剩下不用的才作为输入/输出口线使用。

（4）P3 口的特点：

① P3 口内部有上拉电阻，不存在高阻态，是一个准双向口。

② P3 口作为第二功能的输出/输入或作为通用输入时，均需将相应的锁存器置 1。实际应用中，由于复位后 P3 口锁存器自动置 1，已满足第二功能运作条件，所以可以直接进行第二功能操作。

③ P3 口的某位不作为第二功能使用时，则自动处于通用输出/输入口的功能，可作为通用输出/输入端口使用。

④ 作为通用输出/输入端口使用时，输入信号取自缓冲器三态门 2 的输出端；作为第二功能使用时，输入信号取自缓冲器三态门 3 的输出端。

⑤ 能驱动 4 个 TTL 负载。

1.5 复位

1.5.1 复位状态

复位是单片机的初始化操作，其主要功能是将程序计数器 PC 初始化为 0000H，使单片机从 0000H 单元开始执行程序。除了进入系统的正常初始化外，当程序运行出错或操作错误使系统处于死锁状态时，也须重新启动单片机，使其复位。

单片机刚接上电源时，其内部各寄存器处于随即状态，复位可使 CPU 和系统中的其它部件都处于一个确定的初始状态，并以此初始状态开始工作。RST 为外部复位信号的输入引脚，当在 MCS-51 单片机的 RST 引脚上输入高电平并至少保持两个机器周期（即 24 个振荡周期）以上时，复位过程即可完成。如果 RST 引脚持续保持高电平，单片机就处于循环复位状态；当 RST 从高电平转为低电平后，CPU 才从初始状态开始工作。

复位也可使单片机退出低功耗工作方式而入正常工作状态。

单片机复位后，除 P3~P0 的端口锁存器被设置成 FFH、堆栈指针 SP 设置成 07H 和串行口的 SBUF 无确定值外，其它各专用寄存器包括程序计数器 PC 均被设置成 00H。片内 RAM 不受复位的影响，上电后 RAM 中的内容是随机的。记住这些特殊功能寄存器的复位状态，对熟悉单片机操作，简短应用程序中的初始化部分是十分必要的。

1.5.2 复位电路

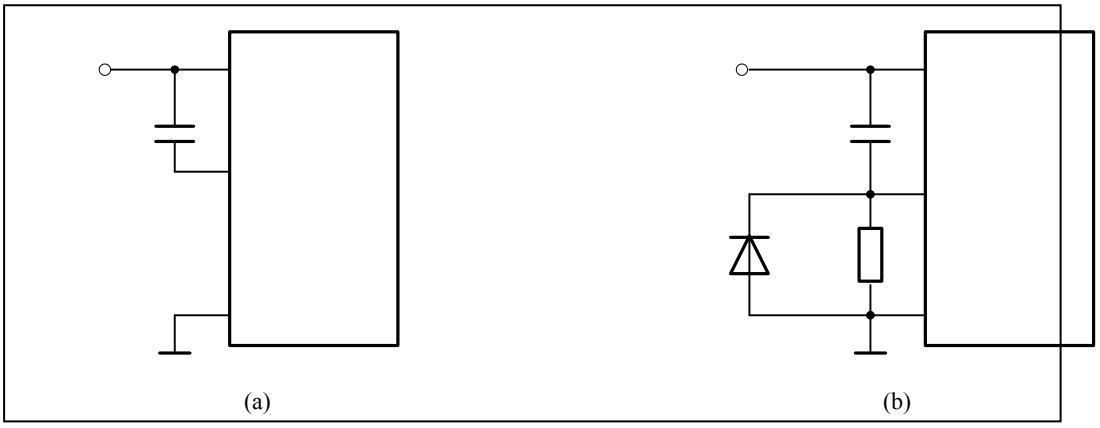
单片机的复位操作有上电自动复位和手动按键复位两种方式。

上电自动复位操作要求接通电源后自动实现复位操作。如图 1.5-1 所示。

图 1.5-1（a）所示为最简单的复位电路。上电瞬间由于电容 C 上无储能，其端电压近似为零，RST 获得高电平，随着电容器 C 的充电，RST 引脚上的高电平将逐渐下降，当 RST 引脚上的电压小于某一数值后，单片机就脱离复位状态，进入正常工作模式。只要高电平能保持复位所需要的时间（约两个机器周期），单片机就能实现复位。

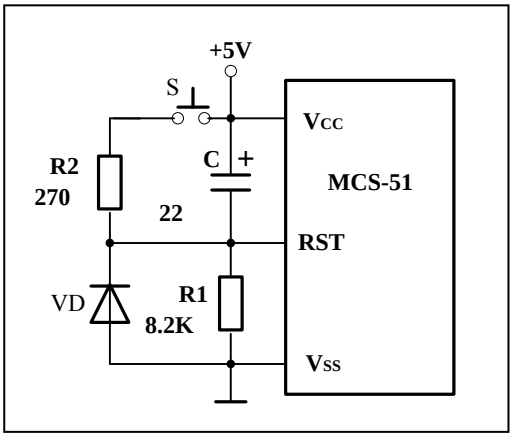
相比于图 1.5-1（a），图 1.5-1（b）所示的电路只是增加了外接二极管 V_D 和电阻 R。其优越性在于停电后，二极管 V_D 给电容 C 提供了快速放电通路，保证再上电时 RST 为高电平，从而保证单片机可靠复位。正常工作时，二极管反偏，对电路没影响。断电后，V_{CC} 逐渐下降，当 V_{CC}=0 时，电容 C 通过 V_D 迅速放电，恢复到无电量的初始状态，为下次上电复位做好准备。

图 1.5-1 上电自动复位电路



手动按键复位要求在电源接通的条件下，用按钮开关操作使单片机复位，如图 1.5-2 所示。其工作原理为：复位键按下后，电容 C 通过 R2 放电，放电结束后，RST 引脚的电位由 R1 和 R2 分压决定，由于 R2<<R1，因此，RST 引脚为高电平，单片机进入复位状态，松开按键后，电容充电，RST 上的电位降低，经过一定的延时，单片机就脱离复位状态，进入正常工作模式。R2 的作用在于限流，避免按键按下的瞬间电容 C 放电产生火花，保护按键的触点。

图 1.5-2　手动按键复位电路



系统上电运行后，若需要复位，一般是通过手动复位来实现的。通常采用手动复位和上电自动复位结合。复位电路虽然简单，但其作用十分重要。一个单片机系统能否正常运行，首先要检查是否能复位成功。初步检查可用示波器探头监视 RST 引脚，按下复位键，观察是否有足够幅度的波形输出（瞬时的），还可以通过改变复位电路阻容值的方法进行检测。

1.6　MCS-51 单片机的工作方式

1.6.1　执行指令程序方式

单片机上电复位后，从程序存储器的 0000H 单元开始执行程序，这就是执行指令程序方式，是单片机的基本工作方式。它又分为连续执行工作方式和单步执行工作方式。

1）连续执行工作方式

这种工作方式是所有单片机都需要的一种方式。由于单片机复位后，PC 值为 0000H，因此复位后单片机总是转到 0000H 处执行程序，但是用户程序并不在 0000H 开始的存储器单元中，为此，需要在 0000H 处放入一条无条件转移指令，以便跳转到用户程序的实际入口地址处执行程序。单片机按照程序事先编排的任务，自动连续地执行下去。

2）单步执行工作方式

这是用户调试程序的一种工作方式，在单片机开发系统上有一专用的单步按键。按一次，单片机就执行一条指令，这样就可以逐条检查程序，发现问题进行修改。

单步执行工作方式是利用单片机外部中断功能实现的。单步执行键相当于外部中断的中断源，当它被按下时，相应电路就产生一个负脉冲（即中断请求信号），送到单片机的 $\overline{\text{INT0}}$（或 $\overline{\text{INT1}}$）引脚，MCS-51 单片机在 $\overline{\text{INT0}}$ 上的负脉冲作用下，便能自动执行预先安排在中断服务程序中的单步执行指令，执行完毕后中断返回。

1.6.2　掉电保护方式

单片机在运行过程中，如果发生突然掉电故障，将会使系统数据丢失，造成严重后果。

MCS-51 单片机设置有掉电保护措施，其保护措施是先保存重要数据，然后启用备用电源维持供电。

数据保护是通过执行一段中断服务程序来实现的。单片机的电源端都接有滤波电容，掉电后电容储存的电能对单片机来说仍可维持有效电压达几个毫秒，足以使单片机完成一次中断操作。因此，在硬件上可在单片机系统中设置一个电压检测电路，一旦检测到电源电压下降，立即向单片机内的中断机构发出中断请求信号，CPU 响应中断请求后执行中断服务程序，把有用的数据送到单片机的片内 RAM 中保护起来。

如果掉电，RAM 中保存的数据仍将丢失，所以在单片机系统中要实现掉电保护方式，还需在引脚 RST/V_{PD} 端连接备用电源及备用电源与 V_{CC} 电源的自动切换电路，此电路在 V_{CC} 电源下降时，自动启用备用电源供电，使单片机进入掉电保护方式。掉电工作方式中，时钟电路和 CPU 皆停止工作，只有内部 RAM 和专用寄存器继续工作，以保持其存储的数据。当 V_{CC} 电源恢复正常时，此电路使 RST/V_{PD} 端连接的备用电源维持一段时间（约 10ms）后，再结束单片机的掉电保护状态。

掉电保护可由以下两个过程描述：

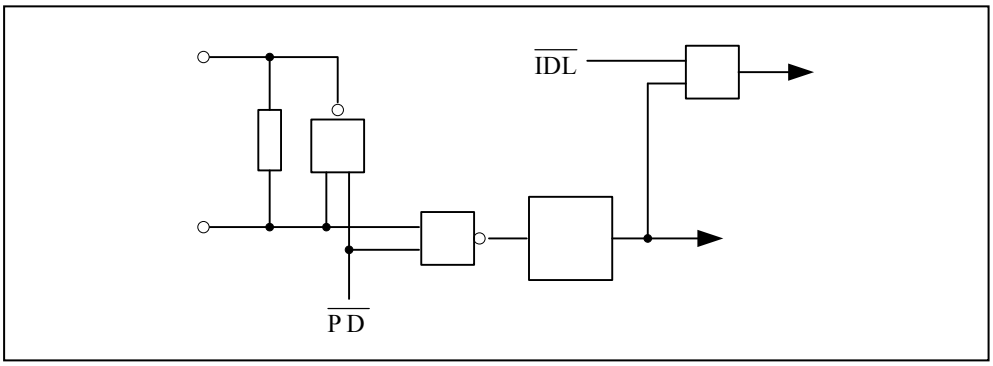
保护过程：V_{CC} 掉电（或低于下限值）→产生外部中断→CPU 响应→将必须保护的数据送入内部 RAM→V_{CC} 继续下降，V_{PD} 继续增加→V_{PD}＞V_{CC}，由 V_{PD} 供电。

恢复过程：V_{CC} 来电，V_{PD} 继续供电→单片机复位→V_{CC} 供电，V_{PD} 撤出。

1.6.3 低功耗方式

在便携式、手提式或野外作业仪器设备上，低功耗是非常有意义的。MCS-51 有两种低功耗节电工作方式：空闲（等待）方式（Idle Mode）和掉电（停机）方式（Power Down Mode）。这两种方式都是由特殊功能寄存器中的电源控制寄存器 PCON 的有关控制位来控制的。它们的工作原理如图 1.6-1 所示。

图 1.6-1 低功耗工作方式原理图



若 $\overline{IDL}$ =0，则单片机进入空闲运作方式。在这种方式下，振荡器仍继续运行，但 $\overline{IDL}$ 封锁了去 CPU 的与门，故 CPU 此时得不到时钟信号。而中断、串行口和定时器等仍在时钟控制下正常运行。掉电方式下（$\overline{PD}$ =0）,振荡器停止工作。

电源控制器 PCON 是不可位寻址的特殊功能寄存器，字节地址为 87H，其每位的定义如下：

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

◆SMOD：为串行口波特率倍率控制位，用于串行通信。若此位为 1，则串行口方式 1、方式 2 和方式 3 的波特率加倍。

◆/：保留位。

◆GF0, GF1：通用标志位，描述中断是来自正常运行还是来自空闲方式，用户可通过指令设定它们的状态。

◆PD： 掉电方式控制位，为 1 时，单片机进入掉电工作方式。由图 1.6-1 可见，此时时钟“冻结”。

◆IDL：空闲方式控制位，该位为 1 时，单片机进入空闲待机工作方式。这时 CPU 因无时钟控制而停止工作。

上面的 IDL、PD 同时为 1，则进入掉电工作方式，同时为 0，则工作在正常运行状态。

1）空闲方式

只要执行将 IDL 位置 1 的指令（MOV PCON, #01H），单片机系统就可进入空闲工作方式。在空闲方式下，振荡器仍然运行，CPU 进入睡眠状态，所有外围电路（中断系统、串行口和定时器/计数器）仍继续工作，但内部 RAM 和特殊功能寄存器中的数据保持在原状态不变，端口状态也保持不变。ALE 和 $\overline{\text{PSEN}}$ 保持逻辑高电平。

空闲方式的退出有两种方式：中断方式和硬件复位方式。

在空闲方式下，中断功能仍然保留。若引入一个中断请求信号，则在单片机响应中断的同时，IDL 位被片内硬件自动清零，单片机就退出空闲方式而进入正常工作状态。在中断服务程序中，只需安排一条 RETI 指令，就可使单片机恢复正常工作后返回断点继续执行程序。

当通过硬件复位方式退出空闲方式时，在内部复位系统发生作用前，单片机要从它转入空闲方式时的断点恢复执行程序达两个机器周期的时间。在这期间，单片机硬件禁止对内部 RAM 的访问，但不禁止对端口引脚的访问。为了避免在退出空闲方式时出现对端口的不希望的写入，跟随在设置空闲方式指令（IDL 置 1 的指令）后面的不应是对端口引脚或外部 RAM 的写入指令。

2） 掉电方式

只要执行一条将 PD 置为 1 的指令（MOV PCON, #02H），可使单片机进入掉电工作方式。在掉电方式下，振荡器停止运行，但内部 RAM 和特殊功能寄存器中的数据保持在原状态不变，直到退出掉电方式。

退出掉电方式的唯一方法是硬件复位，复位时所有特殊功能寄存器都将被初始化，但片内 RAM 的内容保持不变。

在掉电工作方式下，V_{CC} 可以下降到 2V，但在进入掉电方式之前 V_{CC} 不能降低。而在准备退出掉电方式之前，V_{CC} 必须恢复正常的工作电压值，并维持足够的时间（约 10 毫秒），使振荡器重新启动并达到稳定后方可退出掉电方式。

本章小结

1）MCS-51 单片机的内部结构主要包括 CPU、只读存储器 ROM、随机存储器 RAM、定时器/计数器、I/O 接口电路、中断系统等部件。MCS-51 单片机的存储结构的特点之一是将程序存储器和数据存储器分开，并有各自的寻址空间和寻址方式。这种结构称为哈佛结构。

2）MCS-51 单片机在物理上有 4 个存储空间：片内程序存储器和片外程序存储器、片内数据存储器和片外数据存储器。MCS-51 单片机内有 256B 的数据存储器 RAM 和 4KB 的程序存储器 ROM（8031/80C31/8032 除外）。除此以外，还可以在片外扩展 RAM 和 ROM，并且各有 64KB 的寻址范围，也就是最多可以在外部扩展 2×64KB 的存储器。

3）MCS-51 单片机有 3 个存储器地址空间：片内/片外统一的 64KB 的程序存储器地址空间、256B 内部数据存储器地址空间及 64KB 外部数据存储器地址空间，这三个空间的地址要么是完全重叠要么是部分重叠，因此，在访问这三个不同的地址空间时应采用不同形式的指令。

4）MCS-51 单片机属于向上生长型堆栈，堆栈的操作规则是在执行指令时由单片机硬件自动进行的：入栈时，先 SP 中的内容加 1，后写入数据；出栈时，先读出数据，后 SP 中的内容减 1。

5）MCS-51 单片机共有 4 个 8 位并行 I/O 端口，分别记为 P0、P1、P2、P3，共有 32 根 I/O 线，每一根 I/O 线都能够独立地用作输入/输出线。每个并行口主要由 4 个部分组成：端口锁存器、输入缓冲器、输出驱动器和外部引脚，P0~P3 口被归入特殊功能寄存器之列，具有字节寻址和位寻址能力。

思考题与习题

1.1　MCS-51 单片机内部包含哪些主要逻辑部件？

1.2　MCS-51 单片机的存储器有几种类型？可划分为几个地址空间？各地址空间的地址范围和容量是多少？

1.3　MCS-51 单片机的存储器结构与一般微机有何不同？程序存储器和数据存储器各有何功能？

1.4　MCS-51 单片机的$\overline{\text{EA}}$信号有何功能？在使用 80C51 单片机时，$\overline{\text{EA}}$信号的引脚应该如何处理？

1.5　MCS-51 单片机的内部数据存储器可分为哪几个不同的区域？说明各区域的使用特点。

1.6　程序计数器 PC 是不可寻址的寄存器，它有何特点？

1.7　程序状态字 PSW 中各位的含义是什么？

1.8　MCS-51 单片机引脚中有多少输入/输出线？它们与单片机片外的地址总线、数据总线和控制总线有什么关系？地址总线和数据总线各是几位？

1.9　MCS-51 单片机对外有几条专用控制总线？其功能怎样？

1.10　P1 口某位锁存器置“0”，其相应的引脚能否作为输入用？为什么？

1.11 什么是堆栈？堆栈指针 SP 有何作用？在程序设计时为什么要重新对 SP 赋值？

1.12 使单片机复位有几种方式？复位后机器的初始状态如何？

第 2 章　MCS-51 单片机指令系统

2.1　MCS-51 单片机指令概述

指令是使算机内部执行一定的动作的一种控制命令，由构成计算机的电子器件特性所决定。一台计算机所能识别、执行的指令的集合就是它的指令系统，不同类型的 CPU 有不同的指令系统。计算机只能识别二进制代码，因此可以用二进制代码来描述指令功能，这样的语言称之为机器语言。

机器语言是计算机唯一能识别和执行的指令，由于机器语言不便被人们识别、记忆、理解和使用，为了方便书写和记忆，人们采用助记符来进行控制命令的描述，这样的系统称为汇编语言。计算机的指令系统一般都是利用汇编语言描述的，汇编语言是由计算机硬件设计所决定的，不同指令系统没有通用性。

MCS-51 单片机机指令系统具有功能强、指令短、执行快等特点，共有 111 条指令。从功能上可划分成数据传送、算术操作、逻辑操作、程序转移、位操作等五大类；从指令字节上可以分为单字节指令(49 条)、双字节指令(46 条)和最长的三字节指令(只有 16 条)；从执行时间上可分成单机器周期指令(64 条)、双机器周期指令(45 条)和乘、除法两条 4 个机器周期的指令。可见，MCS-51 单片机指令系统在存储空间和执行时间方面具有较高的效率。

2.1.1　MCS-51 单片机汇编语言指令格式

指令的表示方式称为指令格式，它规定了指令的长度和内部信息的安排。完整的指令格式如下：

[标号:] 操作码 [操作数] [,操作数] [；注释]

其中：[ ]项是可选项，下面分别做简要介绍。

1）标号：指令的符号地址。

其要求为:①用于一段功能程序的识别标记或控制转移地址；②指令前的标号代表该指令的地址，是用符号表示的地址；③一般用英文字母和数字组成；④标号必须用冒号“ ：”与操作码分隔。

2) 操作码：又称助记符，它是由英文缩写构成，规定了指令的操作性质，是指令的核心部分。

3）操作数：参与操作的数据或数据地址。

操作数有以下特点：① 操作数可以是数据，也可以是数据的地址、数据地址的地址或操作数的其他信息；②操作数可分为目的操作数和源操作数；③操作数可用二进制数、十进制数或十六进制数表示；④操作数的个数可以是 0～3 个；⑤操作数与操作码之间用空格分隔，操作数与操作数之间用逗号“，”分隔。

4) 注释：注释也是指令语句的可选项，它是为增加程序的可读性而设置的，是针对某指令而添加的说明性文字，不产生可执行的目标代码。

2.1.2　指令中的常用符号

MCS-51 单片机的指令系统中定义了一些常用的符号，这些符号本身有其特定的含义，它们或用于操作数，或用于指令注释。使用这些符号，不仅增加了指令形式的规范性，也使

得指令更加便于理解和记忆。下面列出这些常用的符号及其意义。

1）#：立即数符。#data：8 位立即数；#data16：16 位立即数。

2）direct：8 位直接地址，代表内 RAM 00H～7FH 或 SFR 的 80H～FFH 。

3）@：间接寻址符。在间接寻址方式中，表示间接寻址寄存器指针的前缀标志。如@Ri，@DPTR，@A+PC，@A+DPTR。

4）addr11：11 位目的地址。主要用于 ACALL 和 AJMP 指令中。

5）addr16：16 位目的地址。主要用于 LCALL 和 LJMP 指令中。

6）rel：带符号的 8 位偏移地址。主要用于相对转移指令，以形成转移的目的地址,其范围是相对于下一条指令第 1 字节地址的-128～+127 个字节。

7）bit：位地址。代表片内 RAM 中的可寻址位 00H～7FH 及 SFR 中的可寻址位。

8）Rn（n=0～7）：表示当前工作寄存器 R0～R7 中的任一个寄存器。

9）Ri（i=0 或 1）：表示通用寄存器组中用于间接寻址的两个寄存器 R0, R1。

10) A（或 ACC）、 B ：表示累加器、 B 寄存器。（A 是寄存器方式，ACC 是直接方式的累加器）

11) C：表示 PSW 中的进位标志位 Cy。

12) $：表示当前的指令地址。

13) /：在位操作指令中，表示对该位先求反后再参与操作。

14)（X）：表示由 X 所指定的某寄存器或某单元中的内容。

15) ((X))：表示由 X 间接寻址单元中的内容。

16) ←：表示指令的操作结果是将箭头右边的内容传送到左边。

17) →：表示指令的操作结果是将箭头左边的内容传送到右边。

18) ∧、 ∨、 ⊕ ：表示逻辑与、或、异或。

2.2 MCS-51 单片机的寻址方式

计算机中说明操作数所在地址的方法称为指令的寻址方式。计算机执行程序的过程实际上是在不断寻找操作数并进行操作的过程。对于不同的计算机，它的寻址方式在设计时已经确定，其寻址方式越多，计算机的灵活性越强，指令系统的功能也就越强。

MCS-51 单片机的指令系统有 7 种寻址方式，分别为立即寻址、直接寻址、寄存器寻址、寄存器间接寻址、变址寻址、相对寻址和位寻址。

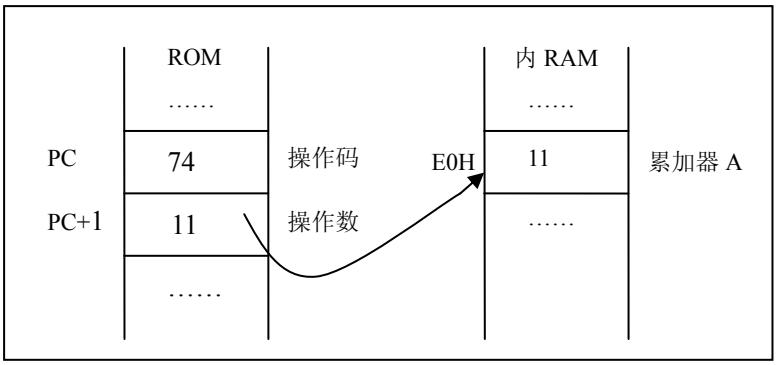
2.2.1 立即寻址

将参与操作的数据直接写在指令中，这种寻址方式称为立即寻址。立即寻址方式特点是第二操作数（源操作数）为 8 位或 16 位常数。立即数通常使用#data 或#data16 表示，在立即数前面加“#”标志（也正因为如此，该操作数被称为“立即数”，并把这一寻址方式形象地称为“立即寻址”）。

例如：对指令“MOV A,#11H”，该指令的机器码为“74H 11H”，指令在程序存储器中存放的位置及执行结果如图 2.2-1 所示。

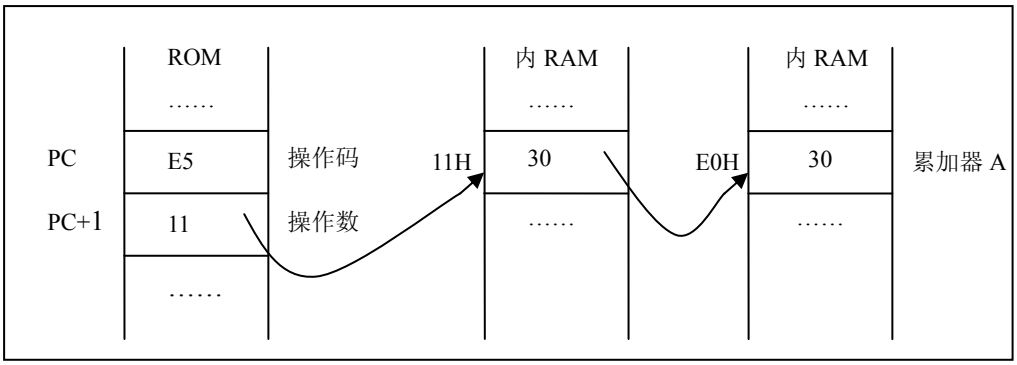
在使用本指令时要注意，只有源操作数能使用立即寻址方式，此外，作为源操作数的立即数的长度必须小于或等于目的操作数的长度。例如“MOV A,#1001H”是错误的，因为累加器 A 的字长为 8 位，无法装入 16 位的二进制数据，这个指令在汇编时会出现错误。

图 2.2-1 立即寻址示意图



例如：指令“MOV A,11H”,已知 11H 单元的内容为 30H。该指令的机器码为“0E5H 11H”,指令在存储器中存放的位置及执行结果如图 2.2-2 所示。

图 2.2-2 直接寻址示意图



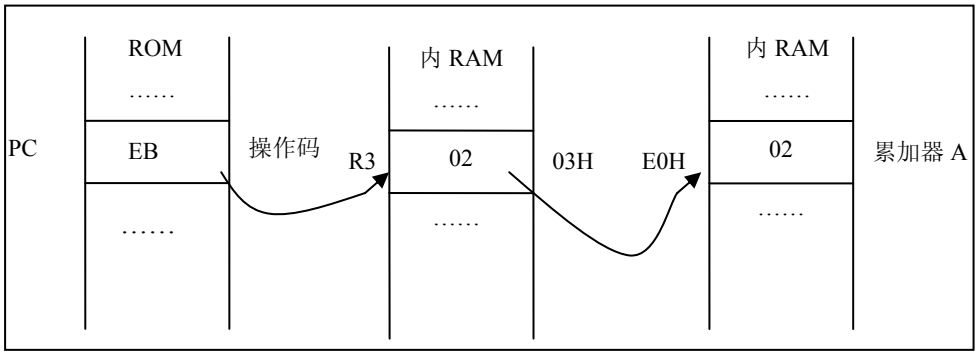
直接寻址方式的寻址范围为 ROM、片内 RAM 区、SFR。

2.2.3 寄存器寻址

操作数存放在 MCS-51 单片机内部的某个工作寄存器 Rn（R0～R7）或部分专用寄存器中，这种寻址方式称为寄存器寻址。寄存器寻址的特点是以一个寄存器的内容作为操作数。存放操作数的寄存器在指令代码中不占据单独的一个字节，而是嵌入（隐含）到操作码字节中。这种寻址方式，可以适用于源操作数，也可以适用于目的操作数。

寄存器寻址方式的寻址范围包括四组通用寄存器 Rn（R0～R7）、部分专用寄存器（ A, B, DPTR。例如：执行指令“MOV A, R3”，设当前工作寄存器为第一组，R3 中存放的数据为 02H，该指令执行过程如图 2.2-3 所示，指令执行完后，（A）= (R3) = 02H。

图 2.2-3 寄存器寻址示意图



2.2.4 寄存器间接寻址

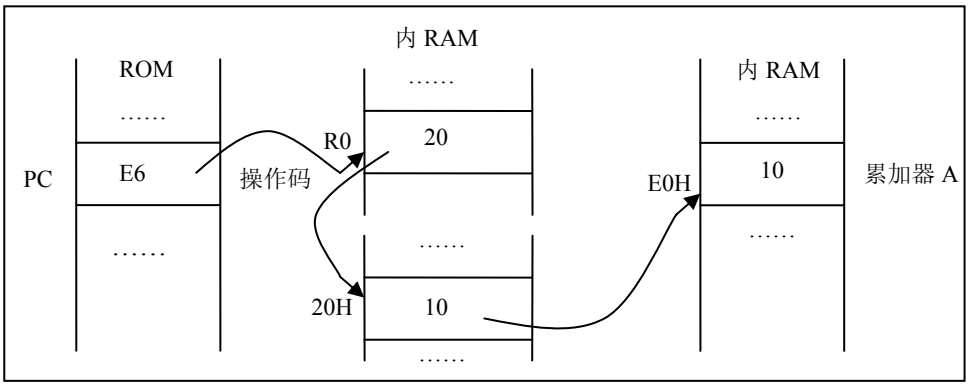
指令中的寄存器中存放的是操作数据的地址。这种寻址方式称为寄存器间接寻址，简称为寄存器间址。

寄存器间接寻址是一种二次寻找操作数地址的寻址方式，先从指令中的寄存器找到其中存放的数，然后用找到的这个数再做地址从而找到真正的操作数。寄存器间接寻址的标志是寄存器前边必须加前缀符号“@”。

寄存器间址的寻址范围：内部 RAM 低 128B（只能使用 R0 或 R1 作间址寄存器）和外部 RAM 的所有区间；对于外部 RAM 低 256 单元的访问，除可以使用 DPTR 外，还可以使用 R0 或 R1 作间址寄存器。注：寄存器间址不能用于寻址特殊功能寄存器 SFR。

例如：“MOV A ,@R0”,其中设（R0）=20H,(20H)=10H,该指令的执行过程如图 2.2-4 所示。

图 2.2-4 寄存器寻址示意图



指令执行时，首先由操作码找到 R0，再取出 R0 中存放的操作数 20H，然后以 20H 做地址，取出 20H 单元存放的操作数 10H 并存入累加器 A，指令执行完后（A）=（(R0)）=（20H）=10H。

2.2.5 变址寻址

操作数存放在变址寄存器（累加器 A）和基址寄存器（DPTR 或 PC）相加形成的 16 位地址单元中。这种寻址方式称为基址加变址寄存器间接寻址，简称为变址寻址。

在执行变址寻址指令时，MCS-51 单片机先把基地址（ DPTR 或 PC 的内容）和地址偏移量（A 的内容）相加，以形成操作数地址，再由操作数地址找到操作数，并完成相应的操作。变址寻址的指令操作码中隐含作为基址寄存器用的 DPTR（或 PC ）和作为变址用的累加器 A，因此变址寻址方式是单字节指令。

变址寻址只能对程序存储器 ROM 进行寻址，主要用于查表性质的访问。

2.2.6 相对寻址

将程序计数器 PC 的当前值与指令给出的偏移量（rel）相加，形成新的转移目标地址，此种方式称为相对寻址方式。相对寻址方式是为实现程序的相对转移而设计的，为相对转移指令所使用，其指令码中含有相对地址偏移量，能生成浮动代码。

如： SJMP rel ； （PC）←（PC） + 2 + rel

相对转移指令的目的地址＝指令地址＋指令字节数＋偏移量

相对寻址方式只能对程序存储器 ROM 进行寻址。相对地址偏移量（rel）是一个带符号

的 8 位二进制补码，其取值范围为–128～＋127（以 PC 为中间的 256 个字节范围），在实际编程中 rel 可以用标号来代替，系统会自动计算偏移量。

2.2.7 位寻址

操作数是一个可单独寻址的位地址，这种寻址方式称为位寻址方式。位寻址是直接寻址方式的一种，其特点是对存储器中可位寻址的某一位进行操作。

可位寻址的位地址的表示形式如下：

1）直接使用位地址形式。如：

MOV 00H, C ；（00H）←（Cy）

其中：00H 是片内 RAM 中 20H 地址单元的第 0 位。

2）字节地址加位序号的形式。如：

MOV 20H.0, C ；（20H.0）←（Cy）

其中：20H.0 是片内 RAM 中 20H 地址单元的第 0 位。

3）位的符号地址（位名称）的形式。对于部分特殊功能寄存器，其各位均有一个特定的名字，所以可以用它们的位名称来访问该位，如：

ANL C, P ；（C）← （C）∧（P）

其中：P 是 PSW 的第 0 位，C 是 PSW 的第 7 位。

4）字节符号地址（字节名称）加位序号的形式。对于部分特殊功能寄存器（如状态标志寄存器 PSW），还可以用其字节名称加位序号形式来访问某一位。如：

CLR PSW.6 ；（AC）← 0，其中： PSW.6 表示该位是 PSW 的第 6 位。

3）寻址范围：片内 RAM 低 128B 中位寻址区、部分 SFR（其中有 83 位可以位寻址）。

以上是 MCS-51 单片机的 7 种寻址方式，在使用的时候要注意：

（1）指令中的源操作数可以使用上面 7 种寻址方式中的任何一种，但目的操作数只能是寄存器寻址、寄存器间址、直接寻址和位寻址这 4 种中的任意一种；

（2）片内 SFR 区只能采用直接寻址方式；

（3）片外 RAM 区只能采用寄存器寻址、寄存器间接寻址（以 R0,R1 或 DPTR 作间址寄存器）的寻址方式访问；

2.3 MCS-51 单片机的指令系统

MCS-51 指令系统使用 44 种助记符，它们代表着 33 种功能，可以实现 51 种操作。指令助记符与操作数各种可能的寻址方式的结合一共可构造出 111 条指令。

一般地，在指令中，操作码占 1 字节；操作数中，直接地址 direct 占 1 字节，#data 占 1 字节，#data16 占两字节；操作数中的 A、B、R0～R7、@Ri、@DPTR、@ A+ DPTR、 @ A+ PC 等均隐含在操作码中。

2.3.1 数据传送指令

数据传送是一种最基本、最主要的操作。在通常的应用程序中，传送指令占有极大的比例，数据传送的灵活性对整个程序的编写和执行都起着很大的作用。

所谓数据“传送”，就是把源地址单元的内容传送到目的地址单元中去，指令执行后一般源地址单元内容不变，属于“复制”而不是“剪切”；或者源地址单元与目的地址单元内容互换。

数据传送指令共 29 条，可分为内部 RAM 数据传送、外部 RAM 数据传送、程序存储器数据传送、数据交换和堆栈操作等五类。数据传送指令对标志位的影响：除以累加器 A 为目的操作数的数据传送指令对 P 标志位有影响外，其余均不影响标志位。

1）内部 RAM 数据传送指令（16 条）

格式：MOV　<dest> , <src>

其中 <dest> 表示目的操作数，<src>表示源操作数。内 RAM 数据传送指令按目的操作数可分为以下几类。

（1）　以累加器 A 为目的操作数的指令（4 条）

这组指令的作用是把对应的源操作数内容存放累加器 A 中，以累加器 A 为目的操作数的指令，只影响 PSW 中的 P 标志位，不影响其他标志位，其指令格式如表 2.3-1 所示。

表 2.3-1 数据传送到累加器 A 的指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOV A,#data	0111 0100　data	A ← data	2	1
MOV A,direct	1110 0101　direct	A ← (direct)	2	1
MOV A,Rn	1110 1rrr	A ← (Rn)	1	1
MOV A,@Ri	1110 011i	A ← ((Ri))	1	1

注：rrr 对应的范围为 000-111，对应着 R0～R7，i 的范围为 0、1，对应着 R0、R1，下同。

例 2-1 把存放在片内 RAM 50H 单元的数据送到累加器 A 中

解法一：　MOV A,50H

解法二：　MOV R0,#50H

　　MOV A,@R0

例 2-2 若(R6)=40H，(20H)=30H，(R0)=30H，(30H)=20H 分析下列每条指令执行后寄存器的内容。

```
MOV　A , R6　　；　(A)=40H，(R6)=40H 不变
MOV　A, 20H　　；执行后(A)=30H，(20H)=30H 不变
MOV　A, @R0　　；执行后(A)=20H，(R0)=30H
MOV　A, #11H　　；执行后(A)=11H
```

（2）　以工作寄存器 Rn 为目的操作数的指令（3 条）

这组指令的作用是把源操作数内容存放到当前工作寄存器组的 R0—R7 中的某个寄存器中，源操作数内容不变，指令格式如表 2.3-2 所示。

表 2.3-2 数据传送到工作寄存器的指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOV Rn,#data	0111 1rrr　data	Rn ← data	2	1
MOV Rn,direct	1010 1rrr　direct	Rn ← (direct)	2	2
MOV Rn, A	1111 1rr	Rn ← (A)	1	1

例 2-3 若（A）=20H，（20H）=60H，分析下面每条指令执行后寄存器的内容。

```
MOV　R2 ,　A　　；(R2)=20H
MOV　R2 ,　20H　　；(R2)=60H
MOV　R2 ,　 #20H ；(R2)=20H
```

（3）　以直接地址单元为目的操作数的指令（5 条）

本组指令的作用是把源操作数内容存放到片内 RAM 的某个直接地址单元中，源操作数内容不变,其指令格式如表 2.3-3 所示。

表 2.3-3 数据传送到直接地址单元的指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOV direct , A	1111 0101 direct	direct ← (A)	2	1
MOV direct , #data	0111 0101 direct data	direct ← data	3	2
MOV direct1, direct2	1110 0101 direct1 direct2	direct1 ← (direct2)	3	2
MOV direct , Rn	1000 1rrr direct	direct ← (Rn)	2	2
MOV direct , @Ri	1000 011i direct	direct ← ((Ri))	2	2

例 2-4 若（A）=40H , (R2)= 46H , (30H)= 70H , (R0)=30H, 分析下列程序执行后寄存器的内容。

```
MOV   R1 , A        ；( R1)=(40H), (A)=40H
MOV   40H , R2      ；(40H)=46H ,(R2)= 46H
MOV   41H , @R0     ；(41H)=70H , (R0)=30H
```

（4） 以间址寄存器@Ri 为目的操作数的指令（3 条）

本组指令的作用是把对应的源操作数的内容存放到间址寄存器 Ri 中去，而源操作数不变。指令格式如表 2.3-4 所示。

表 2.3-4 数据传送到间址寄存器的指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOV @Ri, A	0111 1rrr data	(Ri) ← (A)	2	1
MOV @Ri, direct	1010 1rrr direct	(Ri) ← (direct)	2	2
MOV@Ri, #data	1111 1rr	(Ri) ← data	1	1

例 2-5 求执行下述指令后，寄存器及 50H 单元的内容有什么变化？

```
MOV R0, #50H
MOV @R0, #00H
```

解：执行完上述指令后，（R0）=50H, (50H)=00H。

（5）16 位数据传送指令（1 条）

16 位数据传送指令格式如表 2.3-5 所示。

表 2.3-5 16 位数据传送指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOV DPTR , #data16	1001 0000 data 高 8 位 data 低 8 位	DPH ← $Data_{15-8}$ DPL ← $Data_{7-0}$	3	2

通过以上内 RAM 数据传送指令我们可以得到内部数据传送关系如图 2.3-1 所示。

2）外部 RAM 数据传送指令（4 条）

CPU 与外部数据存储器之间进行数据传送时，必须使用外部传送指令，外部数据传送必须通过累加器 A，且采用寄存器间接寻址（用 R0, R1 和 DPTR 三个间接寻址的寄存器）方式完成，其指令格式如表 2.3-6 所示。

以上 4 条指令中，以累加器 A 为目的操作数的指令为外 RAM 的读指令，以累加器 A 为源操作数的指令为外 RAM 写指令。由于 Ri 是 8 位的寄存器，所以只能访问片外 RAM 的低 256 个单元（00H～0FFH）；DPTR 可以访问片外 RAM 的全部 64KB 的空间。

图 2.3-1　内 RAM 数据传送关系图

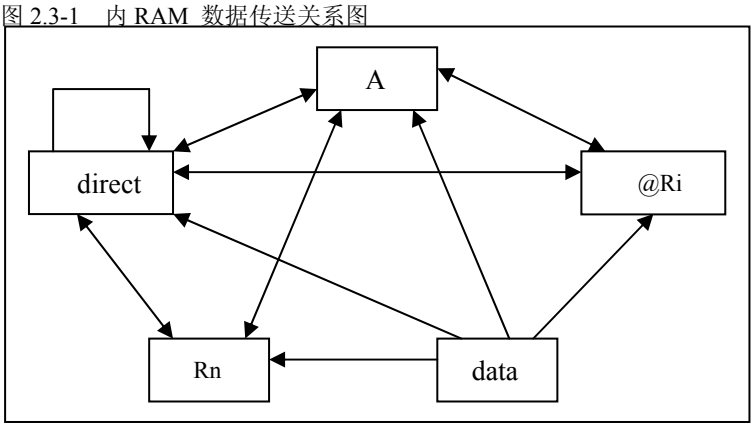


表 2.3-6 外 RAM 数据传送指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOVX　A, @Ri	1110 001i	A ← ((Ri))	1	2
MOVX　A, @DPTR	1110 0000t	A ← ((DPTR))	1	2
MOVX　@Ri,　A	1111 001i	(Ri) ← (A)	1	2
MOVX　@DPTR,　A	1111 0000	(DPTR) ← (A)	1	2

本组指令中，外 RAM 的读指令对标志位有影响，其余的指令对标志位没有影响。从上面的指令可以看出，要与外部数据存储器 RAM 打交道必须通过累加器 A，所有需要送入外部 RAM 的数据要通过累加器送出去，而所有要读入外部 RAM 的数据也必须通过累加器读入。由此可以看出内 RAM 和外 RAM 的区别：内 RAM 之间可以直接进行数据传送；而外 RAM 之间以及外 RAM 和内 RAM 之间不能直接进行数据传送。

例 2-6 下列指令执行完之后，累加器 A 和外 RAM 1234H 单元内容分别是多少？

```
MOV 30H, #44H          ；(30H)=44H
MOV   R1 ,#30H         ；(R1)=30H
MOVX   A , @R1         ；(A)=44H
MOV DPTR , #1234H      ；(DPTR)=1234H
MOVX   @DPTR , A       ；(1234H)=(A)=44H
```

由注释可知，执行完上述指令之后，（A）=44H, (1234H)=44H

3) 程序存储器（ROM）数据传送指令（查表指令）（2 条）

MCS-51 单片机的程序存储器 ROM 分为内 ROM 和外 ROM，程序存储器内部的数据是只读的，因此程序存储器的数据传送是单向的，并且只能读到累加器 A 中。这类指令在查表中经常用到，又称为查表指令。查表指令一共有 2 条，都属于变址寻址，指令格式如表 2.3-7 所示。

表 2.3-7 ROM 数据传送指令表

编语言指令	机器码	指令功能	指令字节数	机器周期数
MOVC A , @A+DPTR	1001 0011	A ← ((A)+(DPTR))	1	2
MOVC A , @A+PC	1000　0011	A ← ((A)+(PC)+1)	1	2

上面这两条指令功能相同，但在使用方式上有区别，这主要体现在：

（1）查表的位置不同

采用 DPTR 作为基地址寄存器，表可以放在 64KB 程序存储器空间的任何地址，使用方便，故称为远程查表；采用 PC 作为基地址寄存器，具体的表在程序存储器中只能在查表指令后的 256B 的地址空间中，使用有限制，故称为近程查表。

（2）偏移量的计算方法不同

采用 DPTR 作为基地址寄存器，查表目标地址为（DPTR）+(A)，A 为欲查数值距离表首地址的值；采用 PC 作为基地址寄存器，查表目标地址为（A）+当前指令的 PC 值+1，因此 A 的值必须预先设置为：A 的值=查表的目标地址－当前指令的 PC 值－1。

例 2-7 指出下面程序执行后累加器 A 中的数据

```
ORG 1000H
MOV A ,#20H       ；双字节指令
MOVC A ,@A+PC ；单字节指令
┆
ORG 1020H
DB 0C0H, 0F9H ,0A4H, 0A5H,30H, 44H ,50H
```

解：查表指令的当前 PC 值为 1002H,上述查表指令执行后，查表目标地址为(1002+1)H+20H=1023H，而 1023H 单元存储的数据 0A5H，故指令执行完后，累加器 A 的内容为 0A5H。

4）数据交换指令（5 条）

数据传输时，若需要保存目的操作数，则经常采用数据交换指令。数据交换指令有 3 条整字节交换和 2 条半字节交换指令，下面分别介绍。

（1）半字节数据交换指令（2 条）

半字节交换指令主要有 A 的高低字节交换和 A 与间址寄存器低 4 位交换，指令格式如表 2.3-8 所示。

表 2.3-8 半字节数据交换指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
SWAP A	1100 0100	$(A)_{0\sim3} \longleftrightarrow (A)_{4\sim7}$	1	1
XCHD A, @Ri	1101 011i	$(A)_{0\sim3} \longleftrightarrow ((Ri))_{0\sim3}$	1	1

注：第二条半字节交换指令对 P 标志位有影响。

（2）整字节交换指令（3 条）

整字节交换指令如表 2.3-9 所示。

表 2.3-9 整字节数据交换指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
XCH A ,Rn	1100 1rrr	(A) ⟷ (Rn)	1	1
XCH A ,direct	1100 0101	(A) ⟷ (direct)	1	1
XCH A , @Ri	1100 011i	(A) ⟷ ((Ri))	1	1

5) 堆栈操作指令（2 条）

堆栈操作在执行中断、子程序调用、参数传递等程序时用于保护断点和恢复现场。其功能是实现 RAM 单元数据送入栈顶或由栈顶取出数据至 RAM 单元，因此堆栈操作指令实际上是以堆栈指针 SP 为间址寄存器的间址寻址方式。堆栈操作有两种：压栈和出栈，指令符分别为 PUSH、POP，堆栈操作指令如表 2.3-10 所示。

在使用堆栈指令时要注意，堆栈区应避开使用的工作寄存器区和其他需要使用的数据区，系统复位后，SP 的初始值为 07H。为了避免重叠，一般初始化时要重新设置 SP。

另外，在书写的时候还要注意，堆栈操作指令是直接寻址指令，直接地址不能是寄存器名，因此应注意指令的书写格式。例如：

PUSH ACC (不能写成 PUSH A)

POP　00H　(不能写成 POP　R0)

表 2.3-10 堆栈操作指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
PUSH direct	1100 0000　direct	SP ← (SP)+1 (SP) ← (direct)	2	2
POP direct	1101 0000 direct	direct ← ((SP)) SP ← (SP)-1	2	2

2.3.2 算术运算指令

算术运算指令主要完成加、减、乘、除四则运算，以及加 1、减 1、BCD 码的运算和调整等。算术运算指令的两个参与运算的操作数，一个存放在累加器 A 中（此操作数也为目的操作数）；一个存放在 R0～R7 或@Ri（片内 RAM）以及片内 RAM 某个直接地址中，或是#data（立即数），这类指令大多数都要影响到程序状态字 PSW，乘除指令不影响 AC 标志位。

1） 不带进位的加法指令（4 条）

不带进位的加法指令是将源操作数的内容与累加器 A 的内容相加，结果存入累加器 A。不带进位的加法指令（ADD）对 PSW 中的所有标志位均产生影响，具体的指令格式如表 2.3-11 所示。

表 2.3-11 不带进位的加法指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
ADD　A , #data	0010 0100 data	A ← (A)+data	2	1
ADD　A, direct	0010 0101 direct	A ← (A)+ (direct)	2	1
ADD　A , Rn	0010 1rrr	A ← (A)+(Rn)	1	1
ADD　A , @Ri	0010 011i	A ← (A)+((Ri))	1	1

例 2-8 如果程序执行前（A）=29H, (20H)=88H , (R7)=0A0H , (R0)=30H , (30H)=0C2H, 求执行完下列每条指令的结果(包括标志位的状态)。

```
ADD A , #10H        ;  (A)=39H, (Cy)=0 , (AC)=0 , (OV)=0
ADD A , 20H         ;  (A)=0C1H, (Cy)=0 , (AC)=1 , (OV)=0
ADD A , R7          ;  (A)=61H, (Cy)=1 , (AC)=0 , (OV)=1
ADD A , @R0         ;  (A)=23H, (Cy)=1 , (AC)=0 , (OV)=0
```

2） 带进位的加法指令（4 条）

带进位的加法指令是将源操作数的内容与累加器 A 的内容、进位标志位的内容一起相加，结果存入累加器 A 中。带进位的加法指令（ADDC）对 PSW 中的所有标志位均产生影响，具体的指令格式如表 2.3-12 所示。

表 2.3-12 带进位的加法指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
ADDC　A , #data	0011 0100 data	A ← (A)+data+(Cy)	2	1
ADDC　A, direct	0011 0101 direct	A ← (A)+ (direct)+(Cy)	2	1
ADDC　A , Rn	0011 1rrr	A ← (A)+(Rn) +(Cy)	1	1
ADDC　A , @Ri	0011 011i	A ← (A)+((Ri)) +(C_y)	1	1

例 2-9 如果程序执行前（A）=9EH,(C_y)=0 , (R0)=32H, 求执行指令“ADDC A , R0”后的结果(包括标志位的状态)。

解：执行完本指令后（A）=0D1H, (C_y)=0 , (AC)=1 , (OV)=0

例 2-10 将内 RAM 20H 单元，21H 单元，22H 单元中的三个无符号数相加，并将和存入 R0(高位)与 R1（低位）。

```
解：MOV   A,20H          ；把 20H 单元内容存入 A
ADD   A , 21H            ；把 21H 单元的内容和 A 中内容（即 20H）单元内容相加
MOV   R1 , A             ；将前两个数相加的低位存入 R1
MOV   A , #00H           ；给 A 赋值为 0
ADDC  A ,#00H            ；将前两个数相加的进位存入 A
MOV   R0 , A             ；将前两个数相加的高位（进位）存入 R0
MOV   A ,R1              ；将前两个数相加的低位存入 A
ADD   A , 22H            ；前两个数相加的低位与第三个数相加
MOV   R1 , A             ；三个数和的低位存入 R1
MOV   A , #00H           ；给 A 赋值为 0
ADDC  A , R0             ；第二次加法的进位与第一次加法的高位（R0 中的数）相加
MOV   R0 , A             ；高位和存入 R0
```

3）带借位的减法指令（4 条）

带借位的减法指令是将累加器 A 的内容减去源操作数的内容，再减去进位标志位的内容，最后将结果存入累加器 A 中。带借位的减法指令（SUBB）对 PSW 中的所有标志位均产生影响，具体的指令格式如表 2.3-13 所示。

表 2.3-13 带借位的减法指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
SUBB A , #data	1001 0100 data	A ← (A)-data-(C_y)	2	1
SUBB A, direct	1001 0101 direct	A ← (A)- (direct)-(C_y)	2	1
SUBB A , Rn	1001 1rrr	A ← (A)-(Rn) -(C_y)	1	1
SUBB A , @Ri	1001 011i	A ← (A)-((Ri)) -(C_y)	1	1

MCS-51 指令系统中没有不带进位的减法，欲实现不带借位的减法，需在执行 SUBB 指令前，使（Cy）清零。

例 2-11 完成 2 字节减法运算，设被减数低位高位分别存放在 40H ,41H 单元中，减数低位高位分别存放在 30H，31H 中，差的低位和高位分别存放在 50H，51H 单元中。

```
MOV   A   , 40H          ；被减数低字节送 A
CLR   C                  ；Cy 清零
SUBB  A ，30H            ；低位字节相减
MOV   50H , A            ；低位字节的差送入 50H
MOV   A , 41H            ；被减数高字节送入 A
SUBB  A ,31H             ；高字节相减
MOV 51H ,A               ；高字节差送入 51H
```

4） 十进制调整指令（1 条）

十进制调整指令是一条专门的指令，它跟在加法指令 ADD 或 ADDC 后面，对运算结果的十进制数进行 BCD 码修正。因为压缩的 BCD 码存在 6 个无效码，BCD 加法时，若结果进入无效码区，会出现错误结果，使用十进制调整指令可以使运算结果调整为压缩的 BCD 码数，以完成十进制加法运算功能。十进制调整指令格式如表 2.3-14 所示。

BCD 码调整的原则为：（1）若累加器 A 的第 4 位大于 9 或（AC）=1，则（A）=(A)+06H；(2)若累加器 A 的高 4 位大于 9 或（C_y）=1，则（A）=(A)+60H；（3）若累加器 A 的低 4 位和高 4 位都大于 9 或（AC）和（C_y）都等于 1，则（A）=(A)+66H。

表 2.3-14 十进制调整指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
DA A	1101 0100	将 A 的内容转换为 BCD 码	1	1

注：DA A 指令对 PSW 中除 OV 之外的所有标志位均有影响。

另外，十进制调整指令不能对减法指令进行修正。BCD 码减法必须采用 BCD 补码运算法则，变减法为补码加法（被减数+减数的补码，减数的补码=9AH－减数），然后对其进行十进制调整来实现。

例 2-12 试编写程序求 68H 和 99H 这两个数的 BCD 码之和。

```
解：MOV   A ,#68H    ；(A)=68H
    MOV R0, #99H     ；(R0)=99H
    ADD   A, R0      ；如果不调整则（A）=01H, (Cy)=1, (AC)=1
    DA    A          ；调整之后（A）=67H, (CY)=1,结果为 167，正确
```

5） 加 1 指令（5 条）

加 1 指令又称为增量指令，其功能是使操作数所指定单元的内容加 1，指令格式如表 2.3-15 所示。

表 2.3-15 加 1 指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
INC A	0000 0100	A ← (A)+1	1	1
INC direct	0000 0101 direct	direct ← (direct)+1	2	1
INC Rn	0000 1rrr	Rn ← (Rn) +1	1	1
INC @Ri	0000 011i	(Ri) ← ((Ri))+1	1	1
INC DPTR	1010 0011	DPTR ← (DPTR)+1	1	2

注：INC 指令除对累加器 A 操作影响 P 标志位外，其他操作均不影响 PSW 的各标志位。

例 2-13 将 50H，51H 单元的内容清零。

```
解： MOV R0 ,#50H    ； 给 R0 赋值
    MOV @R0, #00H    ；利用间址寻址，把 50H 单元清零
    INC R0           ；(R0)=51H
    MOV @R0 ,#00H    ；把 51H 单元清零
```

6）减 1 指令（4 条）

减 1 指令又称为减量指令，其功能是使操作数所指定的单元的内容减 1，指令格式如表 2.3-16 所示。

表 2.3-16 减 1 指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
DEC A	0001 0100	A ←(A)-1	1	1
DEC direct	0001 0101 direct	direct ←(direct)-1	2	1
DEC Rn	0001 1rrr	Rn ←(Rn)- 1	1	1
DEC @Ri	0001 011i	(Ri) ←((Ri))-1	1	1

注：DEC 指令除对累加器 A 操作影响 P 标志位外，其他操作均不影响 PSW 的各标志位。

例 2-14 编程实现 DPTR 减 1 的运算。

解：由于减 1 指令中没有 DPTR 减 1 的指令，因此需要将 DPTR 拆分为 DPH 和 DPL 来进行操作。

程序如下：

```
CLR   C              ；将 Cy 清零
MOV   A , DPL        ；将数据指针的低 8 位送入 A
SUBB A, #01H         ；将 A 的内容减 1 送入 A
MOV DPL , A          ；将 A 的内容减 1 送入 DPL
MOV A , DPH          ；将数据指针的高 8 位送入 A
SUBB A,#00H          ；将 A 的内容减 0，再减去低位的借位送入 A
MOV DPH ,A           ；将 A 中得到的最终结果送入 DPH
```

7） 乘除指令（2 条）

乘除指令实现乘法或除法操作，这两条指令是 MCS-51 指令系统中执行时间最长的指令，各占 4 个机器周期。乘法可以完成 2 个 8 位无符号数相乘，乘数和被乘数分别放在 A 和 B 寄存器中，乘积的高 8 位放在 B 寄存器中，低 8 位放在累加器 A 中；除法指令中，被除数和除数分别存放在 A 和 B 寄存器中，商存放在 A 中，余数存放在 B 中。指令格式如表 2.3-17 所示。

表 2.3-17 乘除指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MUL AB	1010 0100	(B)(A) ← (A)×(B)	1	4
DIV AB	1000 0100	(A) ← (A)/(B)	1	4

注：乘除指令影响 PSW 的 P, OV 和 C_y 标志位,C_y 总是被清零的。乘法运算中，若乘积大于 0FFH，则 OV=1；除法运算中，若除数为 0，则 OV=1。

2.3.3 逻辑运算和移位指令

常用的逻辑运算和移位类指令有：逻辑与、逻辑或、逻辑异或、循环移位、清 0、求反等指令，它们的操作数都是 8 位的。逻辑运算都是按位进行的，除用于逻辑运算外，还可用于模拟各种数字逻辑电路的功能，进行逻辑电路的设计。逻辑运算和移位指令中，除了以 A 为目的操作数的指令影响 P 标志位以及带进位的循环移位指令影响 C_y 和 P 之外，其余均不影响 PSW 各标志位。

1）逻辑与运算指令（6 条）

本指令能实现两个操作数的逻辑与，指令格式如表 2.3-18 所示。

表 2.3-18 逻辑与运算指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
ANL A ,#data	0101 0100 data	A ←(A) ∧ data	2	1
ANL A , direct	0101 0101 direct	A ←(A) ∧ (direct)	2	1
ANL A , Rn	0101 1rrr	A←(A) ∧ (Rn)	1	1
ANL A , @Ri	0101 011i	A ←(A) ∧ ((Ri))	1	1
ANL direct ,A	0101 0010 direct	direct←(A) ∧ (direct)	2	1
ANL direct,#data	0101 0011 direct data	direct←(direct) ∧ (data)	3	2

逻辑与运算主要可以使操作数的某些位不变(这些位与“1”),某些位置 0(这些位与“0”)。

例 2-15 取 P 口的 D0，D4 位，结果存放到 30H 单元

```
解：MOV P1, #11H          ；P1.0,P1.4 口写 1
MOV A ,P1                 ；读 P1 口的值
ANL A ,#11H               ；另 P1.0, P1,4 口不变，其余位清零
MOV 30H ,A                ；结果存放到 30H 单元
```

2）逻辑或运算指令（6 条）

本指令能实现两个操作数的逻辑或，指令格式如表 2.3-19 所示。逻辑或运算主要可以使操作数的某些位不变（这些位或“0”），某些位置 1（这些位或“1”）。

表 2.3-19 逻辑或运算指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
ORL A ,#data	0100 0100 data	A ← (A) ∨ data	2	1
ORL A , direct	0100 0101 direct	A ← (A) ∨ (direct)	2	1
ORL A , Rn	0100 1rrr	A ← (A) ∨ (Rn)	1	1
ORL A , @Ri	0100 011i	A ← (A) ∨ ((Ri))	1	1
ORL direct ,A	0100 0010 direct	direct ← (A) ∨ (direct)	2	1
ORL direct,#data	0100 0011 direct data	direct ← (direct) ∨ (data)	3	2

例 2-16 将 P1 口输入的低 3 位信号，与 P2 口输入的高 5 位合并成一个字节，然后从 P3 口输出，试编程实现之。

```
解：MOV  P1, #0FFH          ；P1 口为准双向口，做输入时先置 1
    MOV P2, #0FFH           ；P2 口为准双向口，做输入时先置 1
    MOV A , P1              ；读 P1 口的值
    ANL A ,#07H             ；屏蔽 P1 口高 5 位，取低 3 位
    MOV R1, A               ；将结果暂存如 R1
    MOV  A ,P2              ；读 P2 口的值
    ANL A ,#0F8H            ；屏蔽 P2 口低 3 位，取高 5 为
    ORL  A ,R1              ；组合 P1 口的低 3 位和 P2 口的高 5 位
    MOV P3 ,A               ；结果从 P3 口输出
```

3）逻辑异或运算指令（6 条）

本指令能实现两个操作数的逻辑或，指令格式如表 2.3-20 所示。

表 2.3-20 逻辑异或运算指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
XRL A ,#data	0100 0100 data	A ← (A) ⊕ data	2	1
XRL A , direct	0110 0101 direct	A ← (A) ⊕ (direct)	2	1
XRL A , Rn	0110 1rrr	A ← (A) ⊕ (Rn)	1	1
XRL A , @Ri	0110 011i	A ← (A) ⊕ ((Ri))	1	1
XRL direct ,A	0110 0010 direct	direct ← (A) ⊕ (direct)	2	1
XRL direct,#data	0110 0011 direct data	direc ← (direct) ⊕ (data)	3	2

逻辑异或运算按位操作，相同为 0，相异为 1，可以实现某些位取反（用这些位异或“1”），某些位不变（用这些位异或“0”）。

4) 循环移位指令（4 条）

MCS-51 单片机的循环移位指令共有不带进位的循环左、右移位（操作码为 RL, RR）和带进位的循环左、右移位（操作码为 RLC, RRC）指令 4 条，只能对累加器 A 进行移位，移位指令可以相当于乘法或除法（左移一位相当于乘 2，右移一位相当于除 2），其指令格式如表 2.3-21 所示。

例 2-17 对存放在 R1R0 的 16 位数（高 8 位在 R1 中）右移一位，最高位补“1”

```
解：MOV A , R1          ；读取高 8 位的值
    SETB C              ；值 Cy 为 1
    RRC A               ；循环右移，是最高位为 1
```

```
MOV R1, A          ；把右移后的结果给高 8 位
MOV A ,R0          ；读取低 8 位的值
RRC   A            ；循环右移，最低位给 Cy
MOV   R0,   A      ；结果给低 8 位
```

表 2.3-21 循环移位指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
RL A	0110 0011	$A_{n+1} \leftarrow (A_n)$（n=0-6）,$A_0 \leftarrow (A_7)$	1	1
RR A	0000 0011	$A_n \leftarrow (A_{n+1})$（n=0-6）,$A_7 \leftarrow (A_0)$	1	1
RLC A	0011 0011	$A_{n+1} \leftarrow (A_n)$（n=0-6）,$Cy \leftarrow (A_7)$, $A_0 \leftarrow (Cy)$	1	1
RRC A	0001 0011	$(A_n) \leftarrow (A_{n+1})$（n=0-6）, $A_7 \leftarrow (Cy)$, $Cy \leftarrow (A_0)$	1	1

5）累加器清 0 与取反指令（2 条）

在 MCS-51 指令系统中，专门提供了累加器 A 的清零和取反指令，利用取反，可以进行求补操作，即对要求补的数先取反再加 1。累加器清零和取反指令均是单字节单周期指令，与逻辑运算或数据传送实现累加器 A 清零或取反相比，效率较高，而且节约存储空间。其指令格式如表 2.3-22 所示。

表 2.3-22 累加器清零与取反指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
CLR A	1110 1100	$A \leftarrow 0$	1	1
CPL A	1111 0100	$A \leftarrow (\overline{A})$	1	1

6）空操作指令

空操作指令的汇编语言格式如表 2.3-23 所示。

表 2.3-23 空操作指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
NOP	0000 0000	PC ← PC+1,空操作	1	1

空操作指令不进行任何操作，但执行时要占用一个机器周期，多用于延时等待或在抗干扰设计中做指令冗余，以提高软件的可靠性。

2.3.4 控制转移指令

计算机在运行的过程中，一般通过程序计数器 PC 的自动加 1 实现顺序执行，有时候对于较复杂的操作，需要通过强迫改变 PC 值的方法来进行程序的分支转移，这就需要用到控制转移指令。控制转移指令的功能就是通过改变程序计数器 PC 中的内容，控制程序执行的流向，实现程序分支转向，MCS-51 单片机提供了 17 条控制转移指令。这些控制转移指令中，除了 CJNE 影响 PSW 的进位标志位 Cy 外，其余均不影响 PSW 的各标志位。

1）无条件转移指令（4 条）

无条件转移指令就是提供不规定条件的程序转移，共有 4 条，指令格式如表 2.3-24 所示。

(1) 长转移指令（LJMP addr16）

长转移指令提供了 16 位的目标转移地址，其功能是把指令码中的 16 位转移目标地址送入 PC，因此这条指令可以跳转到 64KB ROM 的任意位置。为了方便程序设计，addr16 通常采用符号地址表示，程序执行时再被汇编成 16 位二进制地址。

表 2.3-24 无条件转移指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
LJMP addr16	0000 0010 $Addr_{15-8}$ $Addr_{7-0}$	PC←addr15～0	3	2
AJMP addr11	$A_{10}A_9A_8$0 0001 A_7-A_0	PC←(PC)+2,PC_{10}～$_0$←addr11	2	2
SJMP rel	1000 0000 rel	PC←(PC)+2+rel	2	2
JMP @A+DPTR	0111 0011	PC←(DPTR)+(A)	1	2

例 2-18 编写初始化程序,使单片机复位后自动执行 MAIN 主程序，已知 MAIN 主程序存放在 ROM8000H 处。

解： ORG 0000H

LJMP MAIN

ORG 8000H

MAIN:……

(2) 绝对转移指令（AJMP addr11）

这条指令可以在该指令的下一条指令的同一个 2KB 区域（2^{11}）内跳转。PC_{15}～PC_{11}用于把 64K 的 ROM 划分为 32 个区域，相当于 32 个页（每页 2KB，由 PC_{10}～PC_0确定）。指令执行时，PC 首先加 2 指向当前 PC 值（即形成新的 PC 值），然后用指令中的 11 位地址替换当前 PC 值的低 11 位，当前 PC 值的高 5 位确定了是哪一页，替代后的低 11 位代表了该页的页内地址，在实际程序设计中为了方便，addr11 也常用标号代替，程序执行时再被自动汇编成对应的地址。

例 2-19 分析下面绝对转移指令的执行情况。

ORG 2000H

MAIN: AJMP 101 1000 0111B

解： 指令执行前：(PC)=MAIN 的标号地址(2000H)

指令执行后：(1)(PC)=(PC)+2=2002H=0010 0000 0000 0010B

(2)用指令中给出的 11 位地址取代当前 PC 的低 11 位，

得转移目标地址= 0010 0101 1000 0111（2587H）

即程序将转到 2587H 处去执行。

（3）相对（短）转移指令（SJMP rel ）

本指令的目标地址是（PC）=(PC)+2+rel，rel 是一个用补码表示的 8 位带符号的二进制数，范围为－128～+127，负数表示向后(地址变小方向)移，正数表示向前(地址变大方向)移。rel 常用符号地址表示，在汇编时计算机能自动计算出 rel 的值，手工汇编时，需要人工计算 rel 的值，rel 值=转移的目的地址-2-指令执行前的 PC 值。

这条指令的优点是只给出相对的转移地址，不具体指出地址值，这样，当程序地址发生变化时，只要相对地址不发生变化，该指令就不需要做任何改动。

例 2-20 指出下面转移指令执行后程序跳转的位置。

ORG 24A5H

SJMP 30H

解：偏移量 rel 为正数，因此程序向前转移，

转移目标地址为（PC）+2+rel=24A5H+02H+30H=24D7H，

故程序转移到 24D7H 处开始执行。

（4）间接（散）转移指令（JMP @A+DPTR）

这条指令的功能是把累加器 A 中的 8 位无符号数与数据指针 DPTR 中的 16 位数相加，

送入 PC，作为下条指令的地址，利用这条指令能实现程序的散转。

例 2-21 分析下面程序的执行过程。

```
        MOV DPTR ,#TAB      ；将 TAB 所代表的地址送入 DPTR
        MOV   A, R1         ；  从 R1 中取数送入 A
        MOV   B,#2          ；给 B 赋值为 2
        MUL   AB            ；A 中的数（即原来 R1 中的数）乘以 2
        JMP @A+DPTR         ；跳转
TAB: SJMP   S1              ；跳转表格，S1 处理程序
        SJMP   S2           ；S2 处理程序
        SJMP   S3           ；S3 处理程序
```

解：下面分析一下本程序的执行过程：

第一句执行完之后，DPTR 中的值就是 TAB 的首地址；

第二句是从 R1 获得一个值，假设这个值为按键处理程序获得的键值，比如按下第一个键，键值为 0，需执行 S1 处理程序；按下第二个键，键值为 1，需执行 S1 处理程序……依次类推。假设按下的是第三个键， 则从 R1 中取到的键值为 2；

执行第 3、第 4 句指令之后，A 中的值为键值的 2 倍，即(A)=4；

接着执行第 5 句，程序将跳转到(DPTR)+(A)=TAB 标号地址+4 处执行，由于 SJMP 指令为 2 周期指令，程序跳至 SJMP S3 处执行，这即是键 3 的处理程序。

由此可见，散转指令可以实现程序的分支转移。

在编程中，经常使用短转移指令 SJMP 和绝对转移指令 AJMP，以便生成浮动代码。另外，MCS-51 单片机无专用的停机指令，若要求动态停机可用 SJMP 指令，常用方法有以下两种：

方法 1: HERE :SJMP HERE

方法 2：SJMP $

2）条件转移指令（8 条）

条件转移指令的功能就是在规定的条件满足时进行程序转移，否则程序往下顺序执行。MCS-51 单片机中，条件转移指令有累加器 A 判零转移指令，比较转移指令和减 1 不为 0 转移指令等，下面分别介绍。

（1）累加器 A 判零转移指令(2 条)

指令格式如表 2.3-25 所示。

表 2.3-25 累加器 A 判零转移指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
JZ rel	0110 0000 rel	若(A)=0,则转移(PC) ←(PC)+2+rel； 若(A)≠0,则顺序执行(PC) ← (PC) +2	2	2
JNZ rel	0111 0000 rel	若(A)≠0,则(PC) ←(PC)+2+rel； 若(A) =0,则顺序执行(PC) ← (PC) +2	2	2

rel 的取值范围是在执行当前转移指令后的 PC 值（PC+2）基础上的－128～+127（用补码表示），实际编程时可以采用符号地址表示。

偏移量 rel 的计算方法：rel = 转移目标地址－转移指令地址(当前 PC 值) － 2

例 2-22 判 A 口有无数据输入，若有则把 A 口数据送内部 RAM50H 单元，若无数据输入，则送 0 到内部 60H 单元。

```
MAIN: MOV P1, #0FFH     ；  P1 口准双向口，读端口前先置 1
      MOV   A , P1      ；读取 P1 口的值至 A
      CPL A             ；A(P1 口的值)取反
```

```
        JZ     ZERO              ；若（A）=0，则 P1 口无输入，转移
        MOV 50H, A               ；（A）不为 0，数据存入内 RAM 50H 单元
      AJMP LOOP                  ；跳转
ZERO: MOV 60H, A                 ；无数据输入，则 0 送入 60H
LOOP :SJMP     $                 ；停机
        END                      ；结束
```

(2) 比较转移指令（4 条）

比较转移指令是把两个操作数做比较，以比较的结果作为控制条件来控制程序的转移方向，共有 4 条指令，指令格式如表 2.3-26 所示。

比较转移指令的转移范围 rel 的取值是在执行当前转移指令后的 PC 值基础上的－128～+127（用补码表示），可以采用符号地址表示。

表 2.3-26 比较转移指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
CJNE A, direct, rel	10110101 direct rel	若(A)=(direct),则(PC)←(PC)+3； 若(A)>(direct),则(PC)←(PC)+3+rel,Cy= 0； 若(A) <(direct),则(PC)←(PC)+3+rel,Cy= 1	3	2
CJNE A, #data , rel	1011 0100 data rel	若(A)=data,则(PC)←(PC)+3； 若(A)>data,则(PC)←(PC)+3+rel,Cy= 0； 若(A) <data,则(PC)←(PC)+3+rel,Cy= 1	3	2
CJNE Rn, #data , rel	1011 1rrr data rel	若(Rn)=data,则(PC)←(PC)+3； 若(Rn)>data,则(PC)←(PC)+3+rel,Cy= 0； 若(Rn) <data,则(PC)←(PC)+3+rel,Cy= 1	3	2
CJNE @Ri, #data ,rel	1011 011i data rel	若((Ri))=data,则(PC)←(PC)+3； 若((Ri))>data,则(PC)←(PC)+3+rel,Cy= 0； 若((Ri))<data,则(PC)←(PC)+3+rel,Cy= 1	3	2

比较转移指令对两个无符号数进行的比较是通过两操作数的减法实现的，影响 Cy 标志位，但不保存最后的差值，两个操作数的内容不变；若用比较转移指令比较两个有符号数，则单纯看 Cy 的值是无法比较这两个数大小的，需先判断其符号位，在符号位相同的情况下可以按照比较条件产生的 Cy 值进一步判断大小。

（3）减 1 不为 0 转移指令（2 条）

循环转移指令是通过对寄存器或地址单元的数据进行减 1 并按条件进行转移的指令，主要用于循环程序当中，实现对循环次数的控制，指令格式如表 2.3-27 所示。

表 2.3-27 减 1 不为 0 转移指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
DJNZ Rn, rel	1101 1rrr rel	(Rn)←(Rn)-1,若(Rn)≠0 则 (PC) ←(PC)+2+rel 若(Rn)=0 则 (PC) ←(PC)+2	2	2
DJNZ direct,rel	1101 0101 direct rel	(direct)←(direct)-1,若(direct)≠0 则(PC) ←(PC)+3+rel 若((direct)=0 则 (PC) ←(PC)+3	3	2

减 1 不为 0 转移指令在执行时，先将操作数内容减 1，并保存减 1 后的结果，然后对该结果进行判断，若为 0 则跳转，否则顺序执行。

例 2-23 编程计算 1+2+3+……+10 的值。

解：本题目为自然数依次相加，从 1 加到 10，故可以用循环指令来控制循环次数。

程序如下：

```
        MOV R0 ,#10          ；设置循环次数，也即加数的最大值
        CLR A                ；累加器清零
        CLR  C               ；进位标志位清零
LOOP: ADDC  A, R0            ；循环主题，用 R0 中的值（加数）加之前的运算和，并存入 A
        DJNZ R0, LOOP        ；控制循环次数
         SJMP  $             ；结束
```

3）子程序调用与返回指令（4 条）

在程序编写时，有时候会需要一些执行相同操作的程序段，为了减少编写和调试的工作量，人们常常把这些程序段定义为子程序。在程序运行时，可以通过调用指令来调用并执行子程序；子程序执行完后，再用返回指令从子程序返回到主程序。

图 2.3-2 主程序与子程序关系图

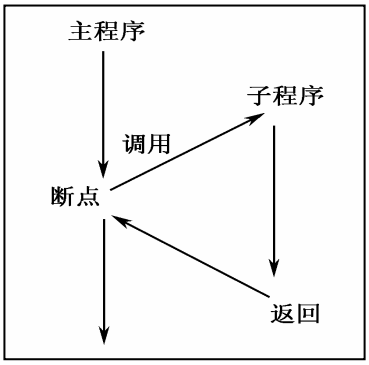
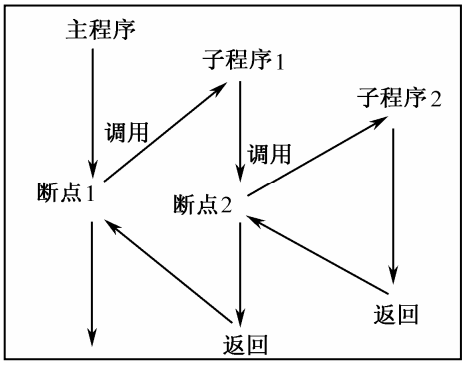


图 2.3-3 子程序嵌套示意



子程序可以嵌套，即在一个子程序执行过程中可以调用另外的子程序，形成子程序的嵌套；嵌套的子程序返回时，先返回后调用的，再返回先调用的，保证调用指令和返回指令成对出现。子程序嵌套有利于模块化程序设计。主程序与子程序之间的调用关系如图 2.3-2 所示，两级子程序嵌套的示意图如图 2.3-3 所示。

为了实现子程序的完整调用，子程序调用指令和返回指令必须成对出现。子程序调用指令在主程序中使用，在需要调用子程序时，调用指令能自动把程序 PC 的当前值（断点地址）保护到堆栈中，并将子程序入口地址自动送入程序计数器 PC 中，从而使程序转去执行子程序；子程序执行完后，需要执行子程序返回指令，它是子程序的最后一条指令，返回指令能自动把堆栈中的断点地址送入 PC，使程序能自动返回到主程序中调用指令的下一条指令处（即断点处）。

子程序调用和返回指令格式如表 2.3-28 所示。

表 2.3-28 子程序调用和返回指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
LCALL addr16	0001 0010 $addr_{15-8}$ $addr_{7-0}$	(PC)← (PC)+3,SP←(SP)+1,(SP)←（PC）$_{7-0}$； SP←(SP)+1,(SP)←$(PC)_{15-8}$,$(PC)_{15-0}$←addr16	3	2
ACALL addr11	A_{10-8}10001 A_{7-0}	(PC)←(PC)+2,SP←(SP)+1，(SP)←$(PC)_{7-0}$ SP←(SP)+1,(SP)←$(PC)_{15-8}$,,$(PC)_{10-0}$←addr11	2	2
RET	0010 0010	(PC) $_{15-8}$←((SP)),SP← (SP) −1, (PC) $_{7-0}$← ((SP)), SP ← (SP) −1	1	2
RETI	0011 0010	(PC) $_{15-8}$←((SP)), SP← (SP) −1， (PC) $_{7-0}$← ((SP)) , SP ← (SP) −1	1	2

在使用子程序调用和返回指令时，子程序调用时应注意入口参数设置，子程序返回时应注意出口参数的传递。

(1)长调用指令 LCALL

长调用指令，用于调用存放在 64KB 空间任意地方的子程序，本指令不影响 PSW 的各标志位。

（2）短调用指令 ACALL

该指令的调用范围为 2KB，执行该指令时共完成 2 项操作：①断点保护，即通过自动堆栈，把断点地址（(PC）+2 的值）保存起来；② 构造目的地址，即在 PC 加 2 之后，用 addr11 代替 PC 的低 11 位，而 PC 的高 5 位不变。

使用 ACALL 指令时需注意，子程序必须存放在该指令执行后第一个字节开始的 2KB 范围内，即同一页内。

（3）子程序返回指令 RET

该指令与子程序调用指令成对出现，其功能是从堆栈中取出断点地址，送入 PC，使主程序从断点处继续执行。

（4）中断返回指令 RETI

中断服务程序是一种特殊的子程序，它是在计算机响应中断时，由硬件完成调用而进入相应的中断服务程序。RETI 指令与 RET 指令相仿，区别在于 RET 是从子程序返回，RETI 是从中断服务程序返回。无论是 RET 还是 RETI 都是子程序执行的最后一条指令。

例 2-25 从 30H～3FH 单元中寻找特征字 80H，如找到，则计数单元 R3 加 1，同时使 P1.0 口驱动发光二极管发光并延时一定时间（发光二极管低电平驱动）。

解：

```
        ORG 1000H
MAIN:   MOV   R3 ,#00H              ；计数单元清零
        MOV   R1 , #10H             ；设置查找单元个数
        MOV   R0, #30H              ；设置查找单元首地址
BIJIAO: CJNE @R0 , #80H , LOOP      ；从第一个单元开始与特征字比较
        INC R3                      ；若找到特征字，则计数单元加 1
        CLR P1.0                    ；驱动发光二极管发光
        LCALL   DELAY               ；调用延时字程序
        SETB P1.0                   ；发光二极管灭
LOOP:   INC R0                      ；查找单元地址加 1
        DJNZ   R1 , BIJIAO          ；是否查找结束，若没有结束，则从下个单元接着找
        SJMP $                      ；查找完，“原地踏步”
DELAY:  MOV R7 , #200               ；延时子程序
     D1: MOV R6 ,#250               ；设置内循环次数
     D2: DJNZ   R6 ,D2              ；内循环
        DJNZ   R7 , D1              ；外循环转移语句
        RET                         ；子程序返回
        END                         ；程序结束
```

2.3.5 位操作指令

位操作指令的操作对象不是字节，而是某个可寻址的位，由于位的取值只能是 0 或 1，故位操作指令又称为布尔变量操作指令。位操作指令操作的对象为片内 RAM 的 128 个可寻址的位和 SFR 中 11 个可位寻址的特殊功能寄存器中的 83 个可寻址位。

位操作指令以进位标志 Cy 作为位累加器，可以实现布尔变量的传送、运算和控制转移

等功能。

1） 位数据传送指令（2 条）

位数据传送可以在可寻址位与位累加器 Cy 之间进行，以 Cy 做中介可以实现两个位之间的数据传送，指令格式如表 2.3-29 所示。

表 2.3-29 位数据传送指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
MOV C ,bit	1010 0010 bit	Cy← (bit)	2	1
MOV bit , C	1001 0010 bit	bit ← (Cy)	2	1

2) 位状态控制指令（4 条）

位状态控制主要是对位置位或复位，其指令格式如表 2.3-30 所示。

表 2.3-30 位状态控制指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
CLR C	1100 0011	Cy ← 0	1	1
CLR bit	1100 0010 bit	bit ← 0	2	1
SETB C	1101 0011	Cy ← 1	1	
SETB bit	1101 0010 bit	bit ← 1	2	1

3) 位逻辑操作指令（6 条）

位逻辑操作有位与、位或、位异或、位取反等，除了位取反指令 CPL bit 外，其余指令均以位累加器 Cy 为目的操作数，执行结果存入 Cy. 原位的内容不发生变化。指令格式如表所 2.3-31 示。

表 2.3-31 位逻辑操作指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
ANL C ,bit	1000 0010 bit	Cy ←(Cy) ∧ (bit)	2	2
ANL C ,/bit	1011 0000 bit	Cy ←(Cy) ∧ $\overline{(bit)}$	2	2
ORL C,bit	0111 0010 bit	Cy ←(Cy) ∨ (bit)	2	2
ORL C ,/bit	1010 0000 bit	Cy ←(Cy) ∨ $\overline{(bit)}$	2	2
CPL C	1011 0011	Cy ← $\overline{(c_y)}$	1	2
CPL bit	1011 0010 bit	(bit) ← $\overline{(bit)}$	2	2

位逻辑操作指令除了用于位逻辑操作外，还可用于对组合逻辑电路的模拟。采用位操作指令进行组合逻辑电路的设计比采用字节型逻辑指令节约存储空间，运算操作十分方便。

例 2-26 设 x, y, z 为不同的位地址，编程实现（z）=(x) ⊕ (y)

```
解：  MOV C, y          ；  (Cy)=(y)
      ANL C , /x        ；  (Cy)= (y) ∧ ( x̄ )
      MOV  z ,C         ；  (z)=(Cy)= (y) ∧ ( x̄ )
      MOV C , x         ；(Cy)=(x)
      ANL  C , /y       ；  (Cy)= (x) ∧ ( ȳ )
      ORL C , z         ；(Cy)= (y) ∧ ( x̄ )+ (x) ∧ ( ȳ )
      MOV z , C         ；  (z)= (y) ∧ ( x̄ )+ (x) ∧ ( ȳ ) =(x) ⊕ (y)
```

4) 位条件（控制）转移指令（5 条）

位条件（控制）转移指令和前面介绍过的条件转移指令类似，只不过是以位的状态作为实现程序转移的判断条件，可以使程序设计更加方便、灵活。指令格式如表 2.3-32 所示。

表 2.3-32 位条件转移指令表

汇编语言指令	机器码	指令功能	指令字节数	机器周期数
JC rel	0110 0000 rel	若(Cy)=1，则转移(PC)←(PC)+2+rel ；否则顺序执行(PC)←(PC)+2	2	2
JNC rel	0101 0000 rel	若(Cy)=0，则转移(PC)←(PC)+2+rel；否则顺序执行(PC)←(PC)+2	2	2
JB bit , rel	0010 0000 bit rel	若(bit)=1，则转移(PC)← (PC)+3+rel；否则顺序执行(PC)←(PC)+2	2	2
JNB bit , rel	0011 0000 bit rel	若(bit)=则转移(PC)← (PC)+3+rel；否则顺序执行(PC)←(PC)+2	2	2
JBC bit , rel	0001 0000 bit rel	若(bit)=1，则转移(PC)← (PC)+3+rel,且(bit)←0；否则顺序执行(PC)←(PC)+2	1	2

例 2-27 求出内 RAM30H 单元中的数据含 1 的个数，并将结果存入 31H 单元。

```
解：    CLR C                    ；Cy 清零
        MOV R2, #8               ；设置循环次数
        MOV R1 ,#00              ；计数单元清零
        MOV A ,30H               ；把 30H 单元内容读入 A
LOOP:   RRC A                    ；执行带进位的右循环，使(ACC.0)送入 Cy 中
        JNC NEXT                 ；判断 Cy 中是否为 1，若不为 1 则转至 NEXT
        INC R1                   ；若 Cy 为 1，则令计数单元内容加 1
NEXT:   DJNZ R2 ,LOOP            ；判断是否够 8 次，若不够则继续执行移位
        MOV 31H, R1              ；将计数结果送入 31H 单元
        SJMP $                   ；停机
        END                      ；结束
```

本章小结

本章主要介绍了单片机的寻址方式、指令系统常用符号及 MCS-51 单片机的指令系统。重点在于寻址方式以及各种指令的灵活应用，难点在于各种控制转移指令和位操作指令的理解和应用。

本章主要内容如下：

1) MCS-51 单片机共有 7 种寻址方式，对每一种寻址方式，需要掌握其工作原理，即在指令执行时寻找操作数的过程。另外转移指令中偏移量 rel ，转移地址 addr11、addr16 等在实际编程时都可以用标号来代替，不需要手动计算偏移量，汇编时由汇编程序自动计算。

2) MCS-51 单片机指令的格式，由[标号：] 操作码 [操作数] [,操作数] [；注释]几个部分组成，其中带[]的表示可有可无的，不同的指令可能操作数的有无或者个数不同，但是对任何指令而言，操作码都是必不可少的，它代表着指令的操作性质。

3）MCS-51 单片机的指令系统组成。MCS-51 单片机指令系统一共有 111 条指令，是由助记符和操作数的各种寻址方式组合而成，包含了数据传送、算术运算、逻辑运算、控制转移、位操作等功能，要掌握每一类指令的助记符及其功能，并根据实际的需要灵活应用。在实际使用当中，往往需要把各种功能的指令融合到一起完成某个特定的功能，因此，必须熟练掌握这些基本的指令及其用法。

思考题与习题

2.1　MCS-51 单片机的指令系统具有哪些特点?

2.2　MCS-51 单片机指令系统中有哪些寻址方式？

2.3　SJMP，AJMP，LJMP 指令在功能上有何不同？

2.4　设（SP）=60H，子程序 LOOP 的首地址为 0340H，现执行位于 0123 处的“LCALL LOOP”双字节指令后，（PC）=_____，(61H)=_____，(62H)=_____。

2.5　已知（1000H）=45H，（1001H）=77H，执行下列指令后（A）=_____,(DPTR)=_____。

```
MOV DPTR，  #1000H
MOVX A, @DPTR
INC DPTR
MOVX @DPTR ,A
```

2.6　已知（SP）=30H，执行下列指令后，（SP）=_____，((SP))= _____。

```
MOV 50H,#0F0H
PUSH 50H
```

2.7　执行下列指令后，A 的内容为_____。

```
MOV A ,#50H
MOV R0,#0FFH
XCH A,R0
```

2.8　执行下列指令后，（A）=_____，(OV)=_____ ，(CY)=_____。

```
CLR C
MOV A ,#12H
MOV B, #0FH
SUBB A, B
```

2.9　执行下列程序后，（P1）= _____。

```
MOV P1，#6EH
CPL P1.0
CPL P1.2
CLR P1.1
SETB P1.7
```

2.10　编程实现将外部数据存储器 200DH 单元中的内容传送到 280DH 单元中。

2.11　编写程序，将片外数据存储器中 2000H 单元中的内容和 2100H 单元的内容相乘，并将结果存放在内部 RAM 的 22H 和 23H 单元中，高位存放在 23H 中。

2.12　请将片外数据存储器地址为 2040H～2060H 区域的数据块，全部搬移到片内 RAM 的 40H～60H 地址区域，并将原数据区全部填为 FFH。

2.13　试编写子程序，使间址寄存器 R0 所指的片外 RAM 单元中的两个连续单元的低 4 位二进制数，合并为一个字节，装入累加器 A 中。已知 R0 指向低地址，并要求该单元低 4 位放在 A 中的高 4 位。

2.14 编写计算 Y 的程序，设乘积结果均小于 255，a、b 为分别存在片外 RAM 3001H 和 3002H 单元中的两个无符号数，结果 Y 存于片外 RAM 3003H 单元中。

$$Y=\begin{cases}16, a=b\\ a+b, a<b\\ a\times b, a>b\end{cases}$$

2.15 某一位的 16 进制数存放在片内 RAM 的 50H 单元，试用查表法将其转换成 ASCII

码并存入该单元。

2.16 某班有 56 个学生，某次考试的成绩存放在内 RAM50H 开始的单元内，试统计其中不及格学生的人数，将结果存放在外部 RAM60H 单元。

第 3 章　汇编语言程序设计

程序设计就是编制计算机的程序，即应用计算机所能识别的、接受的语言把要解决的问题的步骤有序地描述出来。

程序设计语言的种类：

（1）机器语言：机器语言是用二进制代码表示的计算机惟一能识别和执行的最原始的程序设计语言。

（2）汇编语言：利用指令助记符来描述的程序设计语言。

（3）高级语言：高级语言接近于人的自然语言，是面向过程而独立于机器的通用语言。

单片机只能识别用二进制数表示的指令，这就是机器指令（机器码），它的缺点是难记忆。而汇编语言与机器码有着以一一对应的关系，非常有利于用户记忆，它是既面向机器又面向用户的语言。

在进行单片机应用系统设计时，除了需要熟悉单片机的硬件原理外，还需要掌握单片机的汇编语言指令系统，掌握程序设计的基本方法和技巧，设计出质量高、可读性好、程序长度最短和执行速度最快的优秀程序。汇编语言程序设计不仅关系到单片机控制系统的特型和效率，而且还与控制系统本身的硬件结构紧密相关。

MCS-51 单片机汇编语言，包含两类不同性质的指令：

（1）基本指令：即指令系统中的指令。它们都是机器能够执行的指令，每一条指令都有对应的机器码。

（2）伪指令：汇编时用于控制汇编的指令。它们都是机器不执行的指令，无机器码。

3.1　程序设计基础

程序就是为计算某一算式或完成某一工作的若干指令的有序集合。计算机的全部工作概括起来，就是执行指令序列的过程。为计算机准备这一系列指令前的过程称为程序设计。目前，可用于程序设计的语言基本上可分为 3 种：机器语言、汇编语言和高级语言。前面已经对这三种语言做了简要介绍，这里重点介绍汇编语言。

在程序设计自动化的第一阶段，就是用英文字符来代替机器语言，这些英文字符被称为助记符。用这种助记符表示指令系统的语言称为汇编语言或符号语言，用汇编语言编写的程序称为汇编语言程序。

汇编语言具有以下几个特点：

（1）助记符指令和机器指令一一对应，所以用汇编语言编写的程序效率高，占用存储空间小，运行速度快，因此汇编语言能编写出最优化的程序。

（2）使用汇编语言编程比使用高级语言困难。因为汇编语言是面向计算机的，汇编语言的程序设计人员必须对计算机硬件有相当深入的了解。

（3）汇编语言能直接访问存储器、输入与输出接口及扩展的各种芯片(比如 A/D、D/A 等)，也可直接处理中断，因此汇编语言能直接管理和控制硬件设备。

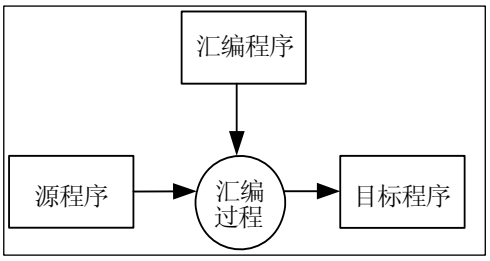
（4）汇编语言缺乏通用性，程序不易移植，各种计算机都有自己的汇编语言，不同的单片微机具有不同的指令系统，并且不能通用。

但是，计算机不能直接识别在汇编语言中出现的字母、数字和符号，需要将其转换成用二进制代码表示的机器语言程序，才能够识别和执行。通常把这一转换（翻译）工作称为汇编。汇编可以由程序员通过查指令表把汇编指令程序转换为机器语言程序，这个过程称为人工汇编。目前基本上由专门的程序来进行汇编，这种程序称为汇编程序。经汇编程序汇编

而得到的机器语言程序，计算机能够识别和执行，因此这一机器语言程序称为目的程序或目标程序，而汇编语言程序称为源程序。这三者之间的关系如图 3.1-1 所示。

3.1.1 汇编语言源程序设计的步骤

图 3.1-1 汇编过程示意图



根据任务要求，采用汇编语言编制程序的过程称为汇编语言程序设计。在进行程序设计时，首先应根据需要解决的实际问题的要求和所使用计算机的特点，决定所采取的计算方法和计算公式，然后结合计算机指令系统特点，本着节省存储单元和提高执行速度的原则编制程序。一个应用程序的编制，通常可以分为以下几步：

1）分析问题，确定算法

这是程序设计中最重要的一步。设计人员必须认真、仔细地考虑系统需要解决的各种问题以及将来系统功能的进一步扩展，明确知道程序要解决的问题和接收、处理、发送的数据范围以及使用什么样的算法。具体可分为：

（1）拟订设计任务书。设计者应根据现场的实际情况，写出比较明确的设计任务书，设计任务书包括：程序功能、技术指标、精度等级、实施方案、所需设备、工程进度、人员分配和研制经费等。

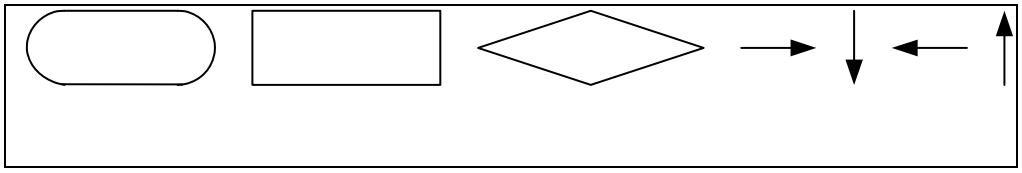
（2）建立数学模型。在弄清楚设计任务的基础上，根据对象的特性建立数学模型。数学模型可以是多种多样的。可以是一系列的数学表达式，可以是运算状态的模拟，也可以是数学的推理和判断等。

（3）确定算法。设计者应根据对象的特性和逻辑关系确定算法。算法是进行程序设计的依据，它决定了程序设计的正确性和程序的质量，确定算法时，不但要根据问题的具体要求，还要考虑指令系统的特点，决定所采用的计算方法和计算公式。

2）编制程序框图

这是程序的结构设计阶段，也是程序设计前的准备阶段。程序流程图是用各种图形、符号、有向线段来直观地表示程序执行的步骤和顺序。它可使人们通过流程图的基本线索，对全局有完整的了解。程序流程图中各部分的规定画法如图 3.1-2 所示。

图 3.1-2 程序流程图规定画法示意图



3）编写源程序代码

根据流程图和指令系统应用的相应软件，用汇编语言指令实现流程图的每一个步骤，从而编写出汇编语言的源程序。

4）调试、测试程序

调试是利用仿真器等开发工具，采用单步、设断点、连续运行等方法排除程序中的错误，直至正确为止。在调试程序时，一般将各模块分调完成后，再进行整个程序的联调。

5）程序优化

程序的优化是以能够完成实际问题要求为前提的，以质量高、可读性好、节省存储单元和提高执行速度为原则。程序设计中经常采用循环和子程序的形式来缩短程序的长度，通过改进算法和择优使用指令来节省工作单元和减少程序的执行时间。

3.1.2 汇编语言的语法结构

各种计算机汇编语言的语法规则是基本相同的，且具有相同的语句格式，现结合MCS-51汇编语言作具体说明。

MCS-51汇编语言的语句格式表示如下：

[<标号>]：<操作码>[<操作数>]；[<注释>]

即一条汇编语句是由标号、操作码、操作数和注释4个部分所组成，其中方括号括起来的是可选择部分，可有可无，视需要而定。

例 3.1 编写程序把片外存储器2200H单元中的数送入片内70H单元中。

解：编程如下：

标号	操作码	操作数	注释
BEGIN：	MOV	DPTR，#2200H	；（DPTR）=2200H
	MOV	R_0，#70H	；（R_0）=70H
	MOVX	A，@ DPTR	；（A）=（(DPTR)）
	MOV	@ R_0，A	；(((R_0)) =（A）

下面分别介绍各个字段的含义：

1）标号

标号是语句地址的标志符号，有了标号，程序中的其它语句才能访问该语句。有关标号的规定如下：

（1）标号是由1—8个ASCII字符组成，但头一个字符必须是字母，其余字符可以是字母、数字或其它特定字符。

（2）不能使用本汇编语言已经定义了的符号作为标号，如指令助记符、伪指令记忆符以及寄存器的符号名称等。

（3）标号后边必须跟以冒号（：）。

（4）同一标号在一个程序中只能定义一次，不能重复定义。

（5）一条语句可以有标号，也可以没有标号，标号的有无取决于本程序中的其它语句是否需要访问这条语句。

下面给出一些常见的错误标号和正确标号，以加深理解。

错误的标号	正确的标号
1BT：（以数字开头）	BT1：
BEGIN（无冒号）	BEGIN：
TA+TB：（“+”号不能出现在标号里）	TATB：
ADD：（指令助记符）	ADD1：

2）操作码

操作码用于规定语句执行的操作内容，操作码是以指令助记符或伪指令助记符表示的，操作码是汇编指令格式中唯一不能空缺的部分。

3）操作数

操作数用于给指令的操作提供数据或地址。在一条语句中，操作数可能是空白，也可能只包括一项，还可能包括二项、三项，各操作数之间以逗号分隔。MCS-51的操作数可能有寄存器、直接、间接等7种不同的寻址方式。

4）注释

注释不属于语句的功能部分，它只是对语句的解释说明，只要用“；”开头，即表明以下为注释内容。使用注释可使程序的文件编制显得更加清楚，帮助程序人员阅读程序，简化软件的维护。注释的长度不限，一行不够时可以换行接着书写，但换行时应注意在开头使用“；”

号。

5）分界符（分隔符）

分界符用于把语句格式中的各部分隔开，以便于区分，包括空格、冒号、分号或逗号等多种符号。这些分界符号在 MCS-51 汇编语言中使用情况如下：

冒号（:）　用于标号之后

空格（ ）　用于操作码和操作数之间

逗号（,）　用于操作数之间

分号（;）　用于注释之前

3.1.3　汇编语言的伪指令

汇编语言程序的机器汇编是由计算机自动完成的。为此，在源程序中应有向汇编程序发出指令信息，告诉它应该如何完成汇编工作，这一任务是通过使用伪指令来实现的。伪指令具有控制汇编程序的输入输出、定义数据和符号、条件汇编、分配存储空间等功能，不同汇编语言的伪指令也有所不同，但一些基本的东西却是相同的。

伪指令是程序员发给汇编程序的命令，也称为汇编命令或汇编程序控制指令。只有在汇编前的源程序中才有伪指令。汇编得到目标程序后，伪指令已无存在的必要，所以，伪指令没有相应的机器代码，在目标程序中见不到与伪指令相对应的机器码。

下面介绍 MCS-51 汇编语言程序中常用的伪指令。

1）汇编起始地址伪指令

汇编起始地址伪指令的一般格式如下：

ORG　表达式

该指令的功能是向汇编程序说明，下述程序段的起始地址由表达式指明。表达式通常为十六进制的地址码。

在一个源程序中，可以多次使用 ORG 指令，以规定不同的程序段的起始位置。但所规定的地址应该是从小到大，而且不允许重叠，即不同的程序代码段之间不能有重叠地址。一个源程序如果不从 ORG 指令开始，则从 0000H 开始存放目标代码。

例 3.2 分析以下程序的执行情况：

```
        ORG   3000H
START:  MOV   A，#7AH
```

解：这段程序汇编后目标代码在存储器中存放的起始地址是 3000H。

2）汇编结束伪指令

汇编结束伪指令的一般格式如下：

格式 1：<字符名称>END<表达式>

格式 2：<字符名称>END　或者　END

只有主程序模块才具有<表达式>项，且<表达式>的值等于该程序模块的入口地址。而其它程序模块就没有<表达式>项。

本指令用于终止源程序的汇编工作。END 是汇编语言源程序的结束标志，因此，在整个源程序中只允许出现一个 END 语句，它必须放在整个程序的最后。如果 END 指令出现在中间，则其后面的源程序，汇编程序将不予处理。

3）赋值伪指令

赋值伪指令的一般格式如下：

<字符名称>EQU<表达式>

这里使用的“字符名称”不是标号，不能用“：”来作分隔符。用 EQU 指令赋值以后的字符名称可以用作数据地址、代码地址、位地址或者当作一个立即数来使用。因此，给字符名

称所赋的值可以是 8 位数，也可以是 16 位数。

本指令用于给标号赋值。赋值以后，其标号值在整个程序中有效。

例 3.3 分析以下程序的执行情况：

```
LOOP    EQU   2007H
```

解：程序执行后，LOOP 的值为 2007H。

说明：用 EQU 指令给一个字符名称赋值之后，在整个程序中该字符名称的值都是固定的，不能更改。若需要改，需用伪指令 DL 重新定义赋值。

4）数据地址定义指令

数据地址定义指令的一般格式如下：

<字符名称>DATA<表达式>

DATA 伪指令的功能和 EQU 有些相似，使用时要注意它们有以下区别：

（1）EQU 伪指令必须先定义后使用，而 DATA 伪指令可以后定义先使用。

（2）用 EQU 伪指令可以把一个汇编符号赋给一个字符名称，而 DATA 只能把数据赋给字符名称；

（3）DATA 伪指令可将一个表达式的值赋给一个字符名称，所定义的字符名称也可以出现在表达式中，而 EQU 定义的字符则不能这样使用。DATA 伪指令在程序中常用来定义数据地址。

5）定义标号值伪指令

定义标号值伪指令的一般格式如下：

<字符名称>DL<表达式>

例 3.4 分析以下程序的执行情况：

```
COUNTDL   4064H              ；定义标号 COUNTDL 的值为 4064H
COUNTDL   COUNT+3            ；重新定义 COUNTDL 的值为 4064H+3
```

解：第 1 条语句执行后，标号 COUNTDL 的值为 4064H，第 2 条语句执行后，COUNTDL 的值重新定义为 4064H+3。

说明：DL 和 EQU 的功能都是将表达式值赋予标号，但两者有差别：可用 DL 语句在同一源程序中给同一标号赋予不同的值，即可更改已定义的标号值；而用 EQU 语句定义的标号在整个源程序中不能更改。

6）定义字节伪指令

定义字节伪指令的一般格式如下：

<字符名称>DB<表达式或表达式列表>

本指令用于从指定的地址开始，在程序存储器的连续单元中定义字节数据。

字节数据可以是一个字节常数或字符，或用逗号分开的字符串，或用引号括起来的字符串。例如：

```
DB“how are you?”
```

把字符串中的字符按 ASCII 码存于连续的 ROM 单元中。

又例如：

```
DB   -2，-4，-6，10，11，17
```

把 6 个数转换为十六进制表示（即 0FEH，0FCH，0FAH，0AH，0BH，11H），并连续存放在 6 个程序存储单元中。

常使用本命令存放数据表格，例如存放数码管显示的十六进制数的字形码，可使用多条 DB 命令定义：

```
DB   0C0H，0F9H，0A4H，0B0H
DB   99H，92H，82H，0F8H
```

```
DB   80H，90H，88H，83H
DB   0C6H，0A1H，86H，84H
```

查表时，为确定数据区的起始地址，可采用两种方法：

（1）根据 DB 命令前一条指令的地址确定。把该地址加上它的字节数，就是 DB 所定义的数据字节的起始地址。例如：

```
8100H   MOV    A，#49H
TAB：   DB     0C0H，0F9H，0A4H，0B0H
        ⋮
```

定义的数码管字行码从 8101H 地址开始存放。因为 MOV　A，#49H 指令是一字节指令。

（2）使用 ORG 命令专门规定

例如：

```
        ORG    8100H
TAB：   DB     0C0H，0F9H，0A4H，0B0H
        ⋮
```

定义的数码管字形码从 8100 地址开始存放。

例 3.5 分析以下程序的执行情况：

```
ORG    4000H
TABLE  DB  45H，67，100，32，00，-2
```

解：以上程序表示字节串数据存入由 TABLE 标号为起始地址的连续存储器单元中，即以 4000H 存储单元开始依次连续存放数据为 45H，67，100，32，00，-2。

7）定义字伪指令

定义字伪指令的一般格式如下：

<字符名称>DW<表达式或表达式表>

DW 是从指定的地址开始定义若干 16 位数据，且把字的高字节数存入低地址单元，低字节数存入高地址单元，按顺序连续存放（数据字的高 8 位在前（低地址），低 8 位在后（高地址））。例如：

```
DW   “AA”              ；存入 41H，41H
DW   “A”               ；存入 00H，41H
DW   “ABC”             ；不合法，因超过两个字节
DW   100H，1ACH，-804  ；按顺序存入 01H，00H，
                        ；01H，0ACH，0FCH，0DCH
```

例 3.6 分析以下程序的执行情况：

```
DW7064H，1234H，209H
```

解：以上程序表示按顺序存入 70H，64H，12H，34H，02H，09H。

注意：DB 和 DW 定义的数表，数的个数不得超过 80 个。如数据的数目较多时，可使用多个定义命令。在 MCS—51 程序设计应用中，常以 DB 来定义数据，以 DW 来定义地址。

8）定义存储区伪指令

定义存储区伪指令的一般格式如下：

<字符名称>DS<表达式>

本指令用于从指定地址开始，保留指定数目的字节单元作为存储区，供程序运行使用。汇编时，对这些单元不赋值。

例如：

```
ADDRTABL：DS   20
```

从标号 ADDRTABL 代表的地址开始，保留 20 个连续的地址单元。

又例如：

```
ORG   8100H
DS    08H
```

从 8100H 地址开始，保留 8 个连续的地址单元。

例 3.7 分析以下程序的执行情况：

```
ORG   4000H
DS    9H
DB    86H，A7H
```

解：汇编后，从 4000H 开始保留 9 个字节的内存单元，然后从 4009H 开始，按照下一条 DB 命令给内存单元赋值，即（4009H）=86H，（400AH）=A7H。保留的空间将由程序的其他部分决定它们的用途。

注意：对 MCS—51 单片机来说，DB，DW，DS 命令只能对程序存储器使用，而不能对数据存储器使用。

9）位地址符号伪指令

位地址符号伪指令的一般格式如下：

<字符名称>BIT<位地址>

位地址符号伪指令是对位地址赋予所规定的字符名称。

说明：其中，位地址可以是绝对地址，也可以是符号地址。

例 3.8 分析以下程序的执行情况：

```
A1   BIT   P2.1
A2   BIT   P3.0
```

解：程序执行后，把 P2.1 和 P3.0 位地址赋值给 A1 和 A2，在后面的编程中，A1、A2 就可作为位地址代替 P2.1、P3.0 使用。

例 3.9 8051 单片机常用伪指令的应用举例。说明以下指令的运行结果：

```
ORG       6070H
BUFFER   DS   45H
DW   “EF”
DW   1100H，209H，-814
```

解：（1）汇编后程序将 6070H 至 60B4H 空间作为缓冲区空间；

（2）（60B5H）=‘E’，（60B6H）=‘F’；

（3）60B7H 单元起存放 11H，00H，02H，09H，FCH，D2H。

3.2 单片机汇编语言程序的基本结构形式

单片机程序设计和其他程序设计一样，程序结构一般也采用 3 种基本控制结构，即顺序结构、分支结构和循环结构。再加上使用广泛的子程序及中断服务子程序，共有 5 种基本结构。

3.2.1 顺序结构程序设计

顺序程序是最简单的程序结构，在顺序程序中既无分支、循环，也不调用子程序，程序执行时一条一条地按顺序执行指令。

由于 MCS—51 指令系统中只有单字节加法指令，因此多字节相加时，必须从低位字节开始分字节进行运算。除最低字节可以使用 ADD 指令之外，其它字节相加时要把低字节的

进位考虑进去，因此应使用 ADDC 指令。

例 3.10 将片内 RAM 30H 单元中的两位压缩 BCD 码转换成二进制数送到片内 RAM 40H 单元中。

解：两位压缩 BCD 码转换为二进制数的算法为：$(a_1a_0)_{BCD}=10\times a_1+a_0$，程序流程图如图 3.2-1 所示。程序如下：

```
        ORG   1000H
START:  MOV   A，30H          ；取两位 BCD 压缩码 a1a0 送 A
        ANL   A，#0F0H        ；取高 4 位 BCD 码 a1
        SWAP  A               ；高 4 位与低 4 位换位
        MOV   B，#0AH         ；将二进制数 10 送入 B
        MUL   AB              ；将 10×a1 送入 A 中
        MOV   R0，A           ；结果送入 R0 中保存
        MOV   A，30H          ；再取两位 BCD 压缩码 a1a0 送 A
        ANL   A，#0FH         ；取低 4 位 BCD 码 a0
        ADD   A，R0           ；求 10×a1+ a0
        MOV   40H，A          ；结果送入 40H 保存
        SJMP  $               ；程序执行完，“原地踏步”
        END
```

3.2.2 分支结构程序设计

分支结构程序设计的特点是根据不同的条件，确定程序的走向。它主要靠条件转移指令、比较转移指令和位转移指令来实现。

分支程序体现了计算机执行程序时的分析判断能力。若某种条件满足，则计算机就转移到另一分支上执行程序；若条件不满足，则计算机就按原程序继续执行。如图 3.2-2 所示。

图 3.2-1 例 3.10 程序流程图

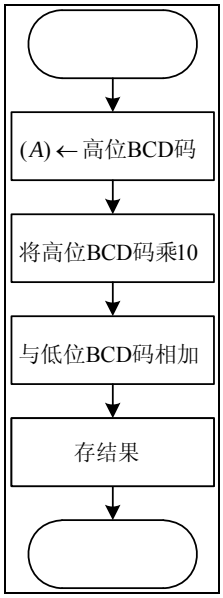
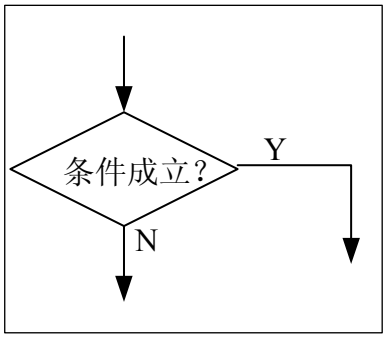


图 3.2-2 分支程序结构示意图



分支程序的设计要点如下：

（1）先建立可供条件转移指令测试的条件。

（2）选用合适的条件转移指令。

（3）在转移的目的地址处设定标号。

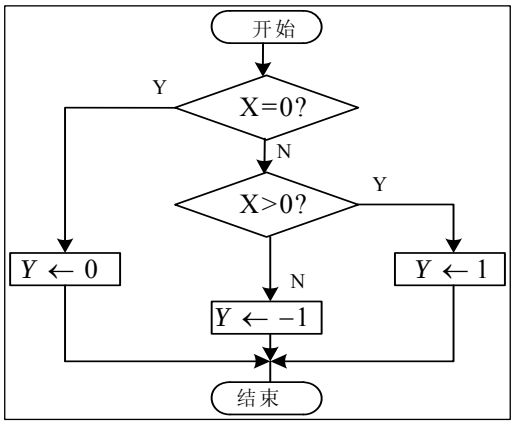
例 3.11 求符号函数的值。已知片内 RAM 的 40H 单元内有一自变量 X，编制程序按如

下条件求函数 Y 的值，并将其存入片内 RAM 的 41H 单元中。

$$Y=\begin{cases}1 & X>0\\0 & X=0\\-1 & X<0\end{cases}$$

解：此题有三个条件，所以有三个分支程序。这是一个三分支归一的条件转移问题。 X 是有符号数，判断符号位是 0 还是 1 可利用 JB 或 JNB 指令。判断 X 是否等于 0 则直接可以使用累加器 A 的判 0 指令。程序流程图如图 3.2-3 所示。

图 3.2-3 例 3.11 程序流程图




```
        ORG   1000H
START:  MOV   A, 40H          ；将 X 送入 A 中
        JZ    COMP            ；若 A 为 0，转至 COMP 处
        JNB ACC.7, POST       ；若 A 第 7 位不为 1（X 为正数），则程序转到 POST 处，否则
                                （X 为负数）程序往下执行
        MOV   A, #0FFH        ；将−1（补码）送入 A 中
        SJMP  COMP            ；程序转到 COMP 处
POST:   MOV   A, #01H         ；将+1 送入 A 中
COMP:   MOV   41H, A          ；结果存入 Y
        SJMP  $               ；程序执行完，“原地踏步”
        END
```

3.2.3 循环结构程序设计

循环程序的特点是程序中含有可以重复执行的程序段（循环体），采用循环程序可以有效地缩短程序，减少程序占用的内存空间，使程序的结构紧凑、可读性好。循环程序一般由下面四部分组成：

（1）循环初始化。位于循环程序开头，用于完成循环前的准备工作，如设置各工作单元的初始值以及循环次数。

（2）循环体。循环程序的主体，位于循环体内，是循环程序的工作程序，在执行中会被多次重复使用。要求编写得尽可能简练，以提高程序的执行速度。

（3）循环控制。位于循环体内，一般由循环次数修改、循环修改和条件语句等组成，用于控制循环次数和修改每次循环时的参数。

（4）循环结束。用于存放执行循环程序所得的结果，以及恢复各工作单元的初值。

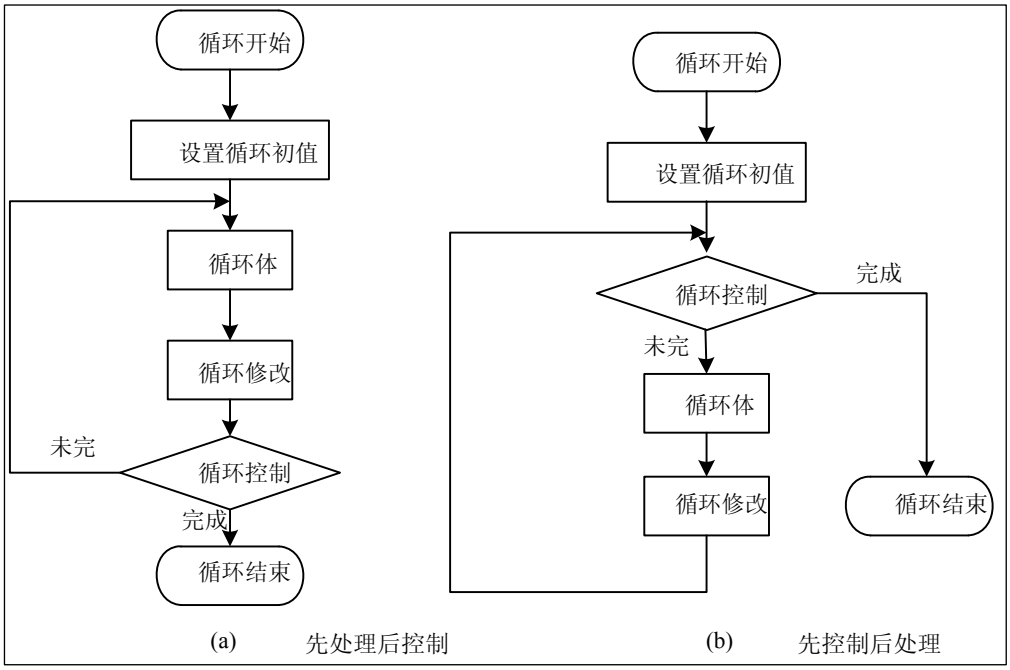
常见的循环结构有两种，一种是先循环处理，后循环控制（即先处理后控制），另一种是先循环控制，后循环处理（即先控制后处理）。如图 3.2-4 所示。

循环程序按结构形式，有单循环与多重循环。

1）单循环程序

循环体内部不包括其他循环的程序称为单循环程序。

图 3.2-4　循环程序结构类型



例 3.12　已知片内 RAM 30H～3FH 单元中存放了 16 个二进制无符号数，编制程序求它们的累加和，并将其和数存放在 R_4, R_5 中。

解：每次求和的过程相同，可以用循环程序实现。16 个二进制无符号数求和，循环程序的循环次数应为 16 次（存放在 R_2 中），它们的和放在 R_4, R_5 中（R_4 存高 8 位，R_5 存低 8 位）。

程序如下：

```
        ORG   1000H
START:  MOV   R0, #30H
        MOV   R2, #10H        ; 设置循环次数（16）
        MOV   R4, #00H        ; 和高位单元 R4 清 0
        MOV   R5, #00H        ; 和低位单元 R5 清 0
LOOP:   MOV   A, R5           ; 和低 8 位的内容送 A
        ADD   A, @R0          ; 将@R0 与 R5 的内容相加并产生进位 Cy
        MOV   R5, A           ; 低 8 位的结果送 R5
        CLR   A               ; A 清 0
        ADDC  A, R4           ; 将 R4 的内容和 Cy 相加
        MOV   R4, A           ; 高 8 位的结果送 R4
        INC   R0              ; 地址递增（加 1）
        DJNZ  R2, LOOP        ; 若循环次数减 1 不为 0，则转到 LOOP 处循环，否则，
                              循环结束
        SJMP  $
        END
```

2）多重循环程序

若循环中还有循环，则称为多重循环，也叫循环嵌套。最简单的多重循环为由 DJNZ 指令构成的软件延时程序。

例 3.13　编制程序设计 50ms 延时程序。

解：延时程序与 MCS-51 指令执行时间（机器周期数）和晶振频率 f_{OSC} 有直接的关系。

当 f_{OSC}=12MHz 时，机器周期为 1μs，执行一条 DJNZ 指令需要 2 个机器周期，时间为 2μs。50ms÷ 2μs>255，因此单重循环程序无法实现，可采用双重循环的方法编写 50ms 延时程序。

程序如下：

```
        ORG   1000H
DELAY：MOV   R7, #200     ；设置外循环次数（此条指令需要 1 个机器周期）
DLY1：  MOV   R6, #123     ；设置内循环次数
DLY2：  DJNZ  R6, DLY2     ；（R6）－1=0，则顺序执行，否则转回 DLY2 继续循环，延时时间
                              为 2μs×123=246μs
        NOP                ；  延时时间为 1μs
        DJNZ  R7，DLY1     ；（R7）－1=0，则顺序执行，否则转回 DLY1 继续循环，延时时间为
                              （246＋2＋1＋1）×200＋2＋1=50.003ms
        RET                ；  子程序结束
        END
```

注意；应用软件延时的程序不允许有中断，否则将严重影响延时时间（定时）的准确性。

3）设计循环程序时应注意的问题

（1）循环程序是一个有始有终的整体，它的执行是有条件的，所以要避免从循环体外直接转到循环体内部。

（2）多重循环程序是从外层向内层一层一层进入，循环结束时是由内层到外层一层一层退出的。在多重循环中，只允许外重循环嵌套内重循环。不允许循环相互交叉，也不允许从循环程序的外部跳入循环程序的内部。

（3）编写循环程序时，首先要确定程序结构，处理好逻辑关系。一般情况下，一个循环体的设计可以从第一次执行情况入手，先画出重复执行的程序框图，然后再加上循环控制和置循环初值部分，使其成为一个完整的循环程序。

（4）循环体是循环程序中重复执行的部分，应仔细推敲，合理安排，应从改进算法、选择合适的指令入手对其进行优化，以达到缩短程序执行时间的目的。

4）排序程序设计（冒泡法）

例 3.14 设 MCS-51 单片机内部 RAM 起始地址为 30H 的数据块中共存有 64 个无符号数，编制程序使它们按从小到大的顺序排列。

解：设 64 个无符号数在数据块中的顺序为：$e_{64}, e_{63},\ldots, e_2, e_1$，使他们从小到大顺序排列的方法很多，现以冒泡法为例进行介绍。

冒泡法又称两两比较法。它先使 e_{64} 和 e_{63} 比较，若 $e_{64}> e_{63}$，则两个存储单元中的内容交换，否则就不交换。然后使 e_{63} 和 e_{62} 比较，按同样的原则决定是否交换。一直比较下去，最后完成 e_2 和 e_1 的比较及交换，经过 N–1=63 次比较（常用内循环 63 次来实现）后，e_1 的位置上必然得到数组中的最大值，犹如一个气泡从水低冒出来一样，如图 3.2-7 所示（图中只画出了 6 个数的比较过程）。

第二次冒泡过程和第一次完全相同，比较次数也可以是 63 次（其实只需要 62 次，因为 e_1 的位置上是数据块中的最大数，不需要再比较），冒泡后在 e_2 的位置上得到数组中的次大数，如图 3.2-5 所示。如此冒泡（即大循环）共 63 次（内循环 63×63 次）便可完成 64 个数的排序。

实际编程时，可通过设置“交换标志”用来控制是否再需要冒泡，若刚刚进行完的冒泡中发生过数据交换（即排序尚未完成），应继续进行冒泡；若进行完的冒泡中未发生过数据交换（即排序已经完成），冒泡应该停止。例如：对于一个已经排好序的数组：1, 2, 3,…, 63, 64，排序程序只要进行一次循环便可根据“交换标志”的状态而结束排序程序的再执行，这自然可以减少 63–1=62 次的冒泡时间。冒泡法程序流程图如图 3.2-6 所示。

图 3.2-5　冒泡法排序过程

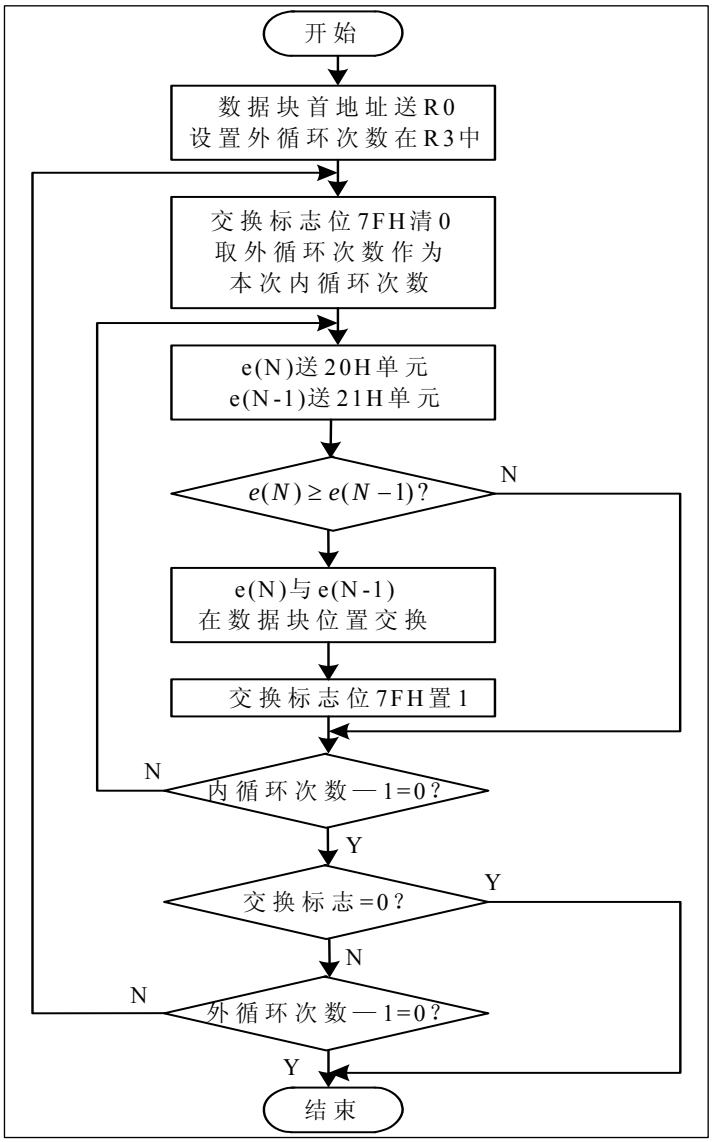
第一次冒泡排序（比较5次）

N=6时		比较1	比较2	比较3	比较4	比较5
e_1	8	8	8	8	8	256
e_2	4	4	4	4	256	8
e_3	0	0	0	256	4	4
e_4	87	87	256	0	0	0
e_5	26	256	87	87	87	87
e_6	256	26	26	26	26	26

第二次冒泡排序（比较5次，实际上只需要4次即可）

N=6时		比较1	比较2	比较3	比较4	比较5
e_1	8	256	256	256	256	256
e_2	4	8	8	8	87	8
e_3	0	4	4	87	8	4
e_4	87	0	87	4	4	0
e_5	26	87	0	0	0	87
e_6	256	26	26	26	26	26

图 3.2-6 冒泡法程序流程图



程序如下：

```
        ORG   1000H
        MOV   R0, #30H        ；数据区首地址送 R0
        MOV   R3, #63H        ；设置外循环次数在 R3 中
LP0：   CLR   7FH             ；交换标志位 2FH.7 清 0
        MOV   A, R3           ；取外循环次数
        MOV   R2, A           ；设置内循环次数
LP1：   MOV   20H, @R0        ；数据区数据送 20H 单元中
        MOV   A, @ R0         ；20H 内容送 A
        INC   R0              ；修改地址指针（R0+1）
        MOV   21H, @ R0       ；下一个地址的内容送 21H
        CLR   C               ；Cy 清 0
        SUBB  A, 21H          ；前一个单元的内容与下一个单元的内容比较
        JC    LP2             ；若有借位（Cy =1），前者小，程序转移到 LP2 处执行，若无借位
                              （Cy =0），前者大，不转移，程序往下执行
        MOV   @ R0, 20H       ；前、后内容交换
        DEC   R0
        MOV   @ R0, 21H
        INC   R0              ；修改地址指针（R0+1）
        SETB  7FH             ；置位交换标志位 2FH.7 为 1
LP2：   DJNZ  R2, LP1         ；修改内循环次数 R2（减少），若 R2≠0，则程序转到 LP1 处仍执行
                              循环，若 R2=0，程序结束循环，程序往下执行
        JNB   7FH, LP3        ；交换标志位 2FH.7 若为 0，则程序转到 LP3 处结束循环
        DJNZ  R3, LP0         ；修改外循环次数 R3（减少），若 R3≠0，程序转到 LP0 处， 执行
                              仍循环，若 R3=0，程序结束循环，往下执行
LP3：   SJMP  $               ；程序执行完，“原地踏步”
        END
```

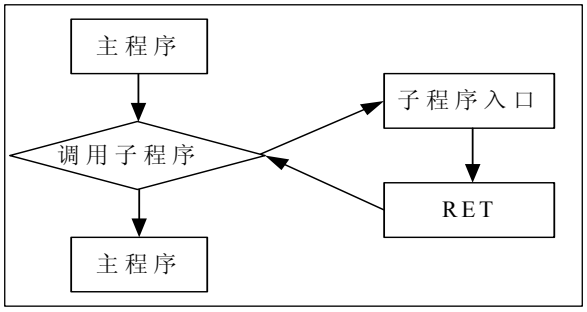
在双重循环（或多重循环中），外循环执行一次内循环执行一圈。因此，在双重循环和多重循环的程序设计中，内循环体前应注意安排循环初始化，内外循环不应相互交叉。此外，为了减少多重循环指令的执行时间，内循环指令应尽可能简捷。

3.2.4 子程序设计

在实际程序编制中，常常会遇到多次进行一些相同操作的程序计算，如数码转换、乘除运算、延时等。如果每次都重新编制一段这样的程序，不仅浪费存储空间、增加程序长度，而且十分烦琐。因此可以将经常使用的程序段从程序中单独取出来，供其他程序随时调用。这种独立的、有一定功能的程序段称为子程序，调用子程序的源程序则称为主程序。

子程序在结构上与主程序的结构没有根本的区别，子程序也可以由简单结构、分支结构或循环结构构成。不同的是在于程序在操作的过程中需要由其他程序来调用，执行完以后又需要将执行流程返回到调用该子程序的程序中。子程序的执行过程如图 3.2-7 所示。

图 3.2-7 子程序调用过程示意图



1）子程序调用的规则和应该注意的地方在汇编语言源程序中调用子程序时，

应注意解决好以下问题：

（1）主程序与子程序之间的参数传递。

（2）保护现场和恢复现场。当然对每个具体的子程序是不是需要现场保护，哪些参数应当保护，还应视实际情况确定。

在 MCS—51 系列单片机的指令系统中，向用户提供了两条调用子程序的指令：ACALL、LCALL 以及一条返回主程序的指令 RET。

此外，子程序在运行过程中，还可以再调用其他的子程序，这种操作称之为程序嵌套。

MCS—51 系列单片机对于子程序嵌套的层数没有限制（只要为堆栈容量所允许）。在编制子程序时，需要注意以下几个方面的问题：

（1）子程序的首地址必须用符号地址，该符号是子程序的名称。子程序最后一定要有一条返回指令。

（2）为了方便使用，每个子程序都有适当的使用说明，如子程序功能的说明、出口条件、所占用的存储单元和寄存器等。

（3）子程序尽量编写成浮动地址程序，采用相对转移指令。

（4）子程序入口条件。在调用子程序之前，必须先将数据或参数送到主程序与子程序的某一共享存储单元或寄存器中。调用子程序后，子程序从共享存储单元或寄存器中取得数，在返回主程序之间，子程序还必须把计算结果送到共享存储单元或寄存器中。这样在返回主程序之后，主程序才能从共享存储单元或寄存器中得到执行子程序后的结果。

（5）保护现场与恢复现场。在调用子程序时，单片机只是自动保护断点地址。但由调用程序转入子程序执行时，往往会破坏住程序或调用程序的有关寄存器（如工作寄存器和累加器等）的内容，也很可能破坏程序状态字 PSW 中的标志位，从而在子程序返回后引起出错。因此，必要时应将这些单元内容保护起来，即保护现场。对于 PSW、A、B 等可通过压栈指令进栈保护。工作寄存器 Rn 采用选择不同工作寄存器组的方式来达到保护的目的。一般主程序选用工作寄存器组 0，而子程序选用工作寄存器的其他组。

2）常用子程序举例

例 3.15 编制程序实现 $c=a^2+b^2$，（a, b 均为 1 位十进制数）。

解：计算某数的平方可采用查表的方法实现，并编写成子程序。只要两次调用子程序，并求和就可得运算结果。设 a, b 分别存放于片内 RAM 的 30H, 31H 两个单元中，结果 c 存放于片内 RAM 的 40H 单元。程序流程图如图 3.2-8 所示。

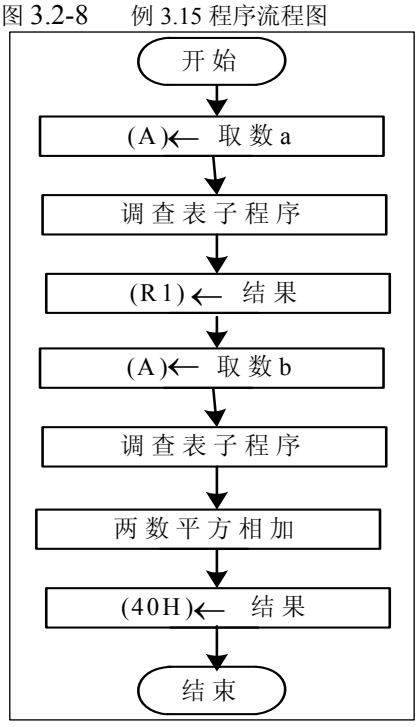
主程序如下：

```
        ORG   1000H
SR：MOV   A, 30H        ；  将 30H 中的内容 a 送入 A
    ACALL  SQR          ；  转求平方子程序 SQR 处执行
    MOV   R1, A         ；  将 a² 结果送 R1
    MOV   A, 31H        ；  将 31H 中的内容 b 送入 A
    ACALL  SQR          ；  转求平方子程序 SQR 处执行
    ADD   A, R1         ；  a²+b² 结果送 A
    MOV   40H, A        ；  结果送 40H 单元中
    SJMP   $            ；  程序执行完，“原地踏步”
```

求平方子程序如下（采用查平方表的方法）：

```
SQR：    INC   A
         MOVC   A, @A+PC
         RET
TABLE：  DB   0, 1, 4, 9, 16
         DB   25, 36, 49, 64, 81
         END
```

图 3.2-8　例 3.15 程序流程图



3.3 MCS-51 单片机汇编语言程序设计举例

3.3.1 数据传送程序设计

例 3.16 编程：现有两个双字节无符号数，分别存放在 R3、R4、R5、R6 中，高字节在前，低字节在后，通过编程使两数相加，和分别存放在 20H、21H、22H 单元中。

解：求和的方法与笔算类似，先加低位后加高位，其程序段如下：

```
        ORG     4000H
        CLR     C                   ；清 C
        MOV     A，R4               ；把被加数的低位放到 A
        ADD     A，R6               ；将加数和被加数的低位相加
        MOV     22H，A              ；把结果的低位存入 22H 单元
        MOV     A，R3               ；把被加数的高位放到 A
        ADDC    A，R5               ；将加数和被加数的高位相加并加低位和进位
        MOV     21H，A              ；把结果的高位存入 21H 单元
        MOV     A，#00H             ；清 A
        ADDC    A，#00H             ；加进位
        MOV     20H，A              ；存和的进位
        END
```

3.3.2 算术运算程序设计

1）加减法运算

（1）不带符号的多个单字节数加法

例如有多个单字节数，依次存放在外部 RAM 21H 开始的连续单元中，要求把计算结果存放在 R1 和 R2 中（假定相加的和为二字节数），其中 R1 为高位，R2 为低位。

```
        MOV     R0，#21H            ；设置数据指针
        MOV     R3，#N              ；字节个数
        MOV     R1，#00             ；和的高位清“0”
        MOV     R2，#00H            ；和的低位清“0”
LOOP：  MOVX    A，@ R0             ；取一个和数
        ADD     A，R2               ；单字节数相加
        MOV     R2，A               ；和的低位送 R2
        JNC     LOOP1
        INC     R1                  ；有进位则和的高位加 1
LOOP1： INC     R0                  ；指向下一单元
        DJNZ    R3，LOOP
```

（2）不带符号的两个多字节数减法

设有两个 N 字节无符号数分别存放在内部 RAM 的单元中，低字节在前，高字节在后，分别由 R0 指定被减数单元地址，由 R1 指定减数单元地址，其差存放在原被减数单元中。

```
        CLR     C                   ；清进位位
        MOV     R2，#N              ；设定字节数
LOOP：  MOV     A，@ R0             ；从低位取被减的一个字节
        SUBB    A，@ R1             ；两数相减
        MOV     @ R0，A             ；存字节相减的差
```

```
        INC     R0
        INC     R1
        DJNZ    R2，LOOP          ；两数相减完否
        JC      QAZ               ；最高字节有借位转溢出处理
        RET
```

（3）带符号数加减运算

对于符号数的减法运算，只要将减数的符号位取反，即可把减法运算按加法运算的原则来处理。

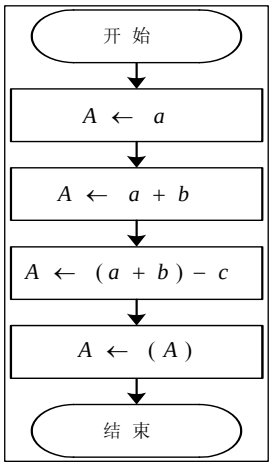
对于符号数的加法运算，首先要进行两数符号的判定。如果两数符号相同，应进行两数相加，并以被加数符号为结果符号。

如果两数符号不同，应进行两数相减。如果相减的差数为正，则该差数即为最后结果，并以被减数符号为结果符号；如果相减的差数为负，则应将其差数取补，并把被减数的符号取反作为结果符号。

例 3.17 若 a、b、c 三个数分别存放在存储器 40H、41H、42H 三个单元中，试编写计算 Y=a＋b－c 的程序。

解：根据题意要求，可先做 a＋b 的运算，然后再做（a＋b）－c 的运算，计算结果送入存储器 Y 的单元中，由算法分析先画出程序执行的流程图，如图 3.3-1 所示。编写 Y=a＋b－c 的源程序如下：

图 3.3-1 程序的流程图




```
        ORG     1000H
START:  MOV     A，40H
        ADD     A，41H
        CLR     C
        SUBB    A，42H
        MOV     43H，A
        END
```

2）乘法运算

由于乘法指令（MUL AB）是对单字节的，因此单字节数的乘法运算使用一条指令就可直接完成，但对于多字节数的乘法就必须通过程序实现。

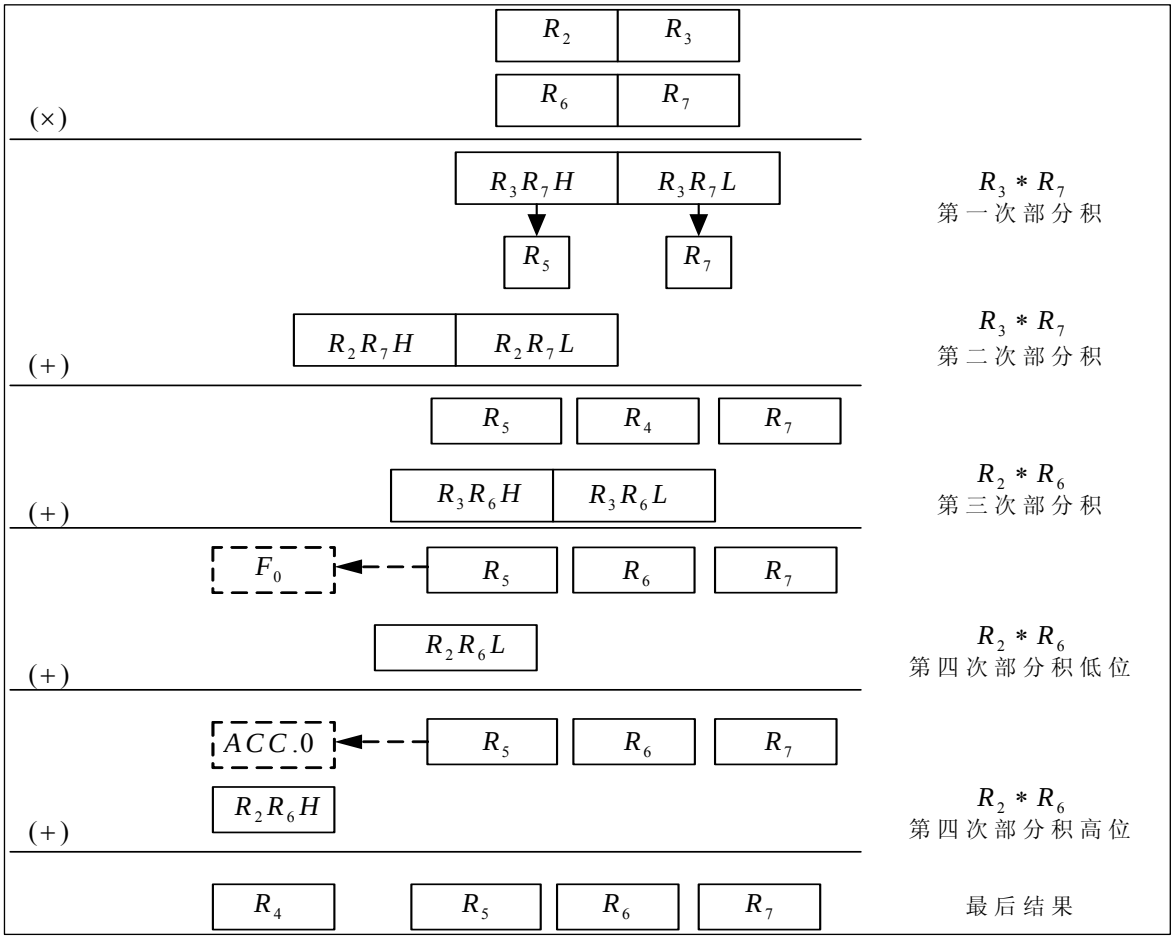
例 3.18 要进行两个双字节无符号数乘法运算，被乘数和乘数分别存放于内部 RAM 的 R2、R3 单元和 R6、R7 单元中（其中 R2 和 R6 分别为高位字节），相乘结果（积）依次存放在 R4、R5、R6、R7 单元中。

因为乘数和被乘数各为二个字节，因此需要进行 4 次乘法运算，得到 4 个部分积，假定部分积的高字节以“H”标志，部分积的低字节以“L”标志。此外，不要忘记对部分积相加产生的进位的处理。为了帮助大家对程序的了解，结合下列程序把乘法运算的实现过程用示意方法表示出来，如图 3.3-2 所示。

乘法程序如下：

```
DBMUL:  MOV     A，R3
        MOV     B，R7
        MUL     AB                ；R3×R7（得第一次部分积）
        XCH     A，R7             ；原 R7 内容送 A，R7←R3 R7L（在 R7 中得到乘积
                                    的第四字节）
        MOV     R5，B             ；R5←R3 R7H
        MOV     B，R2
        MUL     AB                ；R2×R7（得第二次部分积）
```

图 3.3-2 两个双字节无符号数乘法示意图




```
ADD     A，R5                ；  R2 R7L+ R3 R7H
MOV     R4，A                ；  R4←和
CLR     A
ADDC    A，B                 ；R2 R7H+（R2 R7L+ R5 时产生的进位）
MOV     R5，A                ；R5←和
MOV     A，R6
MOV     B，R3
MUL     AB                   ；R3×R6（得第三次部分积）
ADD     A，R4                ；R3 R6L+ R4
XCH     A，R6                ；A←R6, R6←R3 R6L+ R4 (在 R6 中得到积的第三
                               字节)
ADDC    A，R5                ；R3R6H+ R5+ (R3R6L+ R4 时产生的进位）
MOV     R5，A                ；R5←和
MOV     F0，C                ；F0←进位
MOV     A，R2
MUL     AB                   ；R2×R6（得第四次部分积）
ADD     A，R5                ；R2R6L+ (R3R6H+ R5 时产生的进位）
MOV     R5，A                ；在 R5 中得到乘积的第二字节
CLR     A
MOV     ACC.0，C             ；累加器最高位←进位
```

```
        MOV   C，F0
        ADDC   A，B                    ；R2R6H+ F0+ ACC.0
        MOV   R4，A                    ；在 R4 中得到乘积的第一字节
        RET
        XCH    A，B                    ；A←R3 R6H，B←R6
```

3）除法运算

除法指令（DIV AB）也是对单字节的，单字节数的除法运算可直接使用该指令完成。而多字节数据的除法需编程实现。

例 3.19 按“移位相减”这一基本方法，通过编写程序实现两个双字节无符号数的除法运算。

相关数据的单元分配如下：

R_6R_7—执行前存被除数，程序执行后存商数（其中 R7 为高位字节）；

R_5R_4—存除数（其中 R_5 为高位字节）；

R_3R_2—存放每次相除的余数，程序执行后即为最终余数；

3AH—溢出标志单元；

R_1—循环次数计数器（16 次）。

为阅读程序方便，再说明以下几个问题：

（1）除法运算需要对被除数和除数进行判定，若被除数为 0，而除数不为 0，则商为 0；若除数为 0，则除法无法进行，置标志单元 3AH 为“0”。

（2）除法运算是按位进行的，每一位是一个循环，一个循环都要做三件事，即：被除数左移一位、余数减除数、根据是否够减使商位得“1”或“0”。对于双字节被除数，如果循环共进行 16 次，除法就完成了。

（3）移位是除法运算的重要操作，最简单的方法是把被除数向余数单元左移，然后把被除数移位后腾出来的低位用于存放商数。这样，除法完成后，被除数已全部移到余数单元并逐次被减得到余数，而被除数单元却为商数所代替。

（4）除法结束后，可根据需要对余数进行四舍五入。为简单起见，这里我们把它省略了。

双字节数除法程序如下：

```
        MOV     3AH，#00H              ；清溢出标志单元
        MOV     A，R5
        JNZ     ZERO                   ；除数不为 0 转
        MOV     A，R4
        JZ      OVER                   ；除数为 0 转设置溢出标志
ZERO：  MOV     A，R7
        JNZ     START                  ；被除数高字节不为 0 开始除法运算
        MOV     A，R6
        JNZ     START                  ；被除数低字节不为 0 开始除法运算
        RET                            ；被除数为 0 则结束
START：CLR      A                      ；开始除法运算
        MOV     R2，A                  ；余数单元清“0”
        MOV     R3，A
        MOV     R1，#10H
LOOP：  CLR     C                      ；进行一位除法运算
        MOV     A，R6
```

```
        RLC     A                       ；被除数左移一位
        MOV     R6，A
        MOV     A，R7
        RLC     A
        MOV     R7，A
        MOV     A，R2                   ；移出的被除数高位移入余数单元
        RLC     A
        MOV     R2，A
        MOV     A，R3
        RLC     A
        MOV     R3，A
        MOV     A，R2                   ；余数减除数
        SUBB    A，R4                   ；低位先减
        JC      NEXT                    ；不够减转移
        MOV     R0，A
        MOV     A，R3
        SUBB    A，R5                   ；再减高位
        JC      NEXT                    ；不够减转移
        INC     R6                      ；够减商为 1
        MOV     R3，A                   ；相减结果送回余数单元
        MOV     A，R0
        MOV     R2，A
NEXT：  DJNZ    R1，LOOP                ；不够 16 次返回
          ⋮                             ；四舍五入处理（省略）
OVER：  MOV     3AH，#0FFH              ；置溢出标志
        RET
```

3.3.3 数制转换程序设计

通常数制转换都采用子程序调用方法进行，即把具体的转换功能由子程序完成，而由主程序来做组织数据和安排结果等工作。

1）十六进制数转换为 ASCII 码

例 3.20 在内部 RAM 的 hex 单元中存有 2 位十六进制数，试将其转换为 ASCII 码，并存放于 asc 和 asc+1 两个单元中。

主程序（MAIN）：

```
        MOV     SP，#3FH
MAIN：  PUSH    hex                     ；十六进制数进栈
        ACALL   HASC                    ；调用转换子程序
        POP     asc                     ；第一位转换结果送 asc 单元
        MOV     A，hex                  ；再取原十六进制数
        SWAP    A                       ；高低半字节交换
        PUSH    ACC                     ；交换后的十六进制数进栈
        ACALL   HASC
        POP     asc+1                   ；第二位转换结果送 asc+1 单元
```

子程序（HASC）：

```
HASC:   DEC     SP                  ；跨过断点保护内容
        DEC     SP
        POP     ACC                 ；弹出转换数据
        ANL     A，#0FH             ；屏蔽高位
        ADD     A，#7               ；修改变址寄存器内容
        MOVC    A，@A+PC            ；查表
        PUSH    ACC                 ；查表结果进栈
        INC     SP                  ；修改堆栈指针回到断点保护内容
        INC     SP
        RET
ASCTAB：DB“0，1，2，3，4，5，6，7”      ；ASCII 码表
        DB“8，9，A，B，C，D，E，F”
```

这是一个很典型的程序，阅读本程序时请注意以下两个问题：

（1）本程序的一个特点就是堆栈的使用，这对于读者加深堆栈的概念十分有利。在本程序中两种使用堆栈的方法都涉及到了，一种是通过堆栈传送数据，被转换的数据在主程序中进栈而在子程序中出栈，最后再把转换结果返回主程序；另一种使用方法是系统自动的，即调用子程序要用堆栈来保护断点。由于是被转换的数据在主程序中先进栈，而断点地址是在调用子程序时才进栈，为此在子程序中要取出转换数据，就得修改堆栈指针 SP，以指向该数据。

（2）在 ASCII 码表中，以字符串形式列出十六进制数，但在汇编时是以 ASCII 码形式写入存储单元的，因此读出来的是被转换数据的 ASCII 码，再压入堆栈返回主程序。

2）ASCII 码转换为十六进制数

例如把外部 RAM 30H—3FH 单元中的 ASCII 码依次转换为十六进制数，并存入内部 RAM 60H—67H 单元之中。

转换算法：把转换的 ASCII 码减 30H。若小于 0 则为非十六进制数；若为 0—9 之间，即为转换结果；若大于等于 0AH，应再减 7。减 7 后，若小于 0AH，则为非十六进制数；若在 0AH—0FH 之间，即为转换结果；若大于 0FH，还是非十六进制数。转换流程如图 3.3-3 所示。

因为一个字节可装两个转换后得到的十六进制数，即两次转换才能拼装为一个字节。为了避免在程序中重复出现转换程序段，因此通常采用子程序结构，把转换操作编写为子程序，主程序流程如图 3.3-4 所示。

主程序（MAIN）：

```
MAIN:   MOV     R0，#30H            ；设置 ASCII 码地址指针
        MOV     R1，#60H            ；设置十六进制数地址指针
        MOV     R7，#08H            ；需拼装的十六进制数字节个数
AB:     ACALL   TRAN                ；调用转换子程序
        SWAP    A                   ；A 高低 4 位交换
        MOVX    @ R1，A             ；存放外部 RAM
        INC     R0
        ACALL   TRAN                ；调用转换子程序
        XCHD    A，@ R1             ；十六进制数拼装
        INC     R0
        INC     R1
        DJNZ    R7，AB              ；继续
```

HALT： AJMP HALT

图 3.3-3 ASCII 码→十六进制数转换程序流程

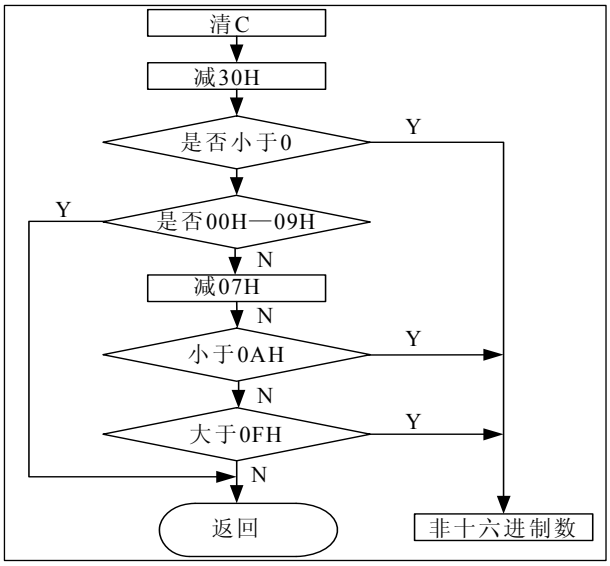
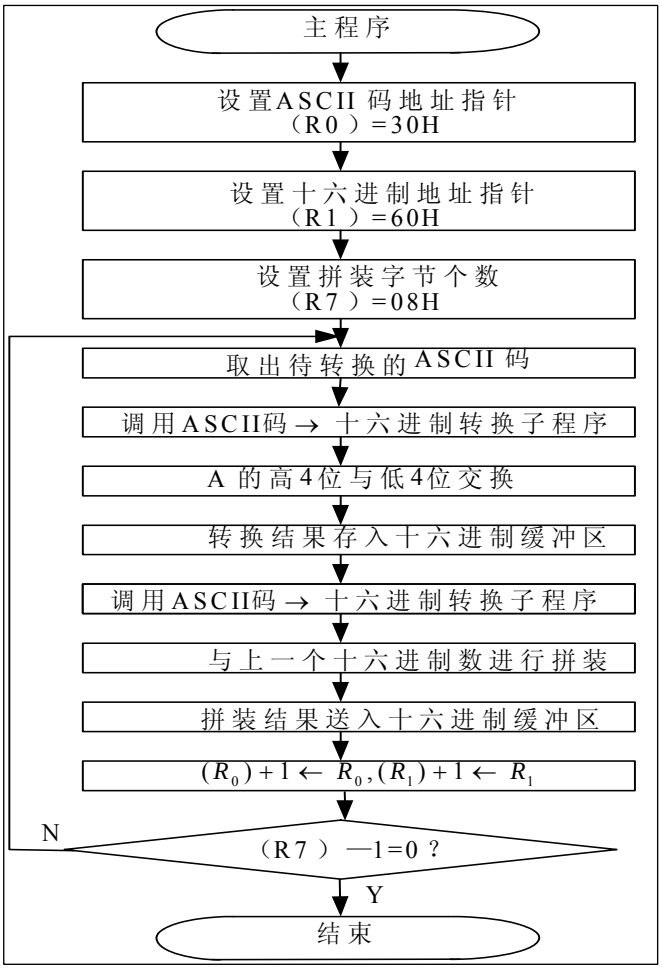


图 3.3-4 ASCII 码→十六进制数转换主程序流程



子程序(TRAN)：

```
TRAN:   CLR     C                   ；清进位位
        MOVX    A，@ R0             ；取 ASCII 码
        SUBB    A，#30H             ；减 30H
        CJNE    A，#0AH，BB
        AJMP    BC
BB:     JC      DONE
BC:     SUBB    A，#07H             ；大于等于 0AH，再减 07H
DONE:   RET                         ；返回
```

3.3.4 查表程序设计

在许多情况下，本来可以通过计算才能解决的问题也可以采用查表的方法解决，而且还要简便的多。因此，在实际的单片机应用中，常常需要编制查表程序以缩短程序的长度和提高程序的执行效率。查表程序主要应用于数码显示、打印字符的转换、数据转换等场合。

查表是根据存放在 ROM 中数据表格的项数来查找与它对应的表中值。MCS—51 指令系统专门设置了两条查表指令。

1）采用 MOVC A, @A+DPTR 指令查表程序的设计方法

（1）在程序存储器中建立相应的函数表（设自变量为 X）。

（2）计算出这个表中所有的函数值 Y。将这群函数值按顺序存放在起始（基）地址为 TABLE 的程序存储器中。

（3)将表格首地址 TABLE 送入 DPTR，X 送入 A，采用查表指令 MOVC　A, @A+DPTR 完成查表，就可以得到与 X 相对应的 Y 值于累加器 A 中。

2）采用 MOVC　A, @A+PC 指令查表程序的设计方法

当使用 PC 作为基址寄存器时，由于 PC 本身是一个程序计数器，与指令的存放地址有关，查表时其操作有所不同。

（1）在程序存储器中建立相应的函数表（设自变量为 X)。

（2）计算出这个表中所有的函数值 Y。将这群函数值按顺序存放在起始（基）地址为 TABLE 的程序存储器中。

（3）X 送入 A，使用 ADD　A, #data 指令对累加器 A 的内容进行修正，偏移量 data 由公式 data=函数数据表首地址–PC–1 确定，即 data 值等于查表指令和函数表之间的字节数。

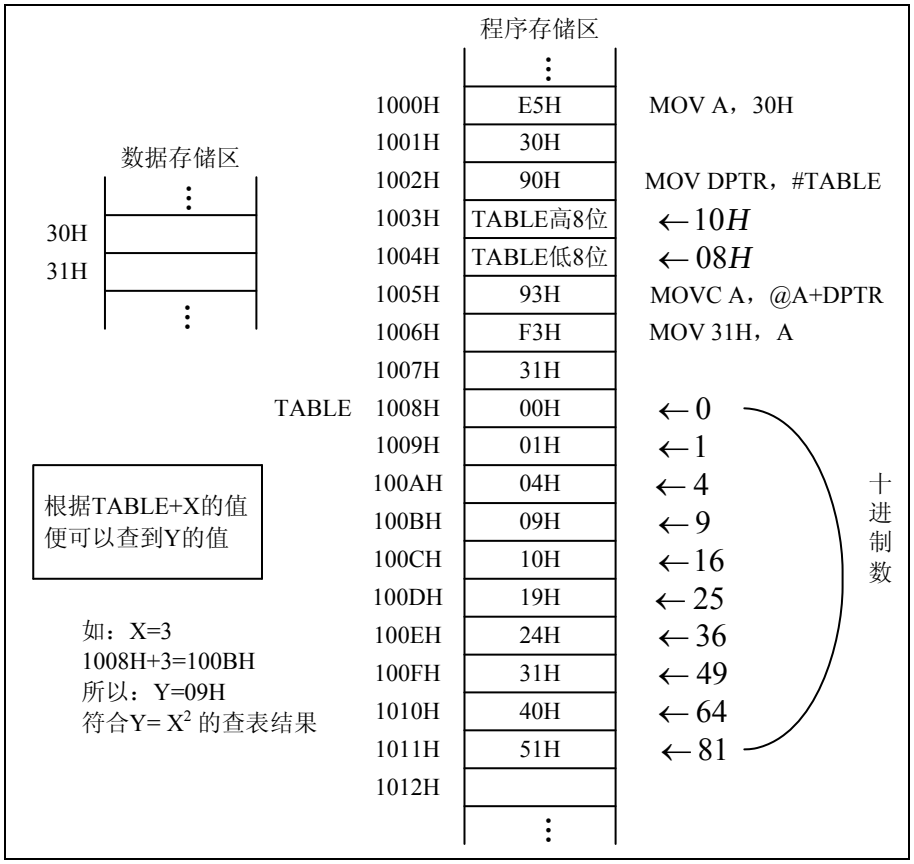
（4）采用查表指令 MOVC　A, @A+PC 完成查表，就可以得到与 X 相对应的 Y 值于累加器 A 中。

例 3.21 利用查表的方法编写 $Y=X^2$（X=0, 1, 2,…, 9）的程序。

解：设变量 X 的值存放在内存 30H 单元中，求得的 Y 的值存放在内存 31H 单元中。平方表存放在首地址为 TABLE 的程序存储器中。

方法一：采用 MOVC　A, @A+DPTR 指令实现，查表过程如图 3.3-5 所示。

图 3.3-5　例 3.21 方法一查表过程示意图

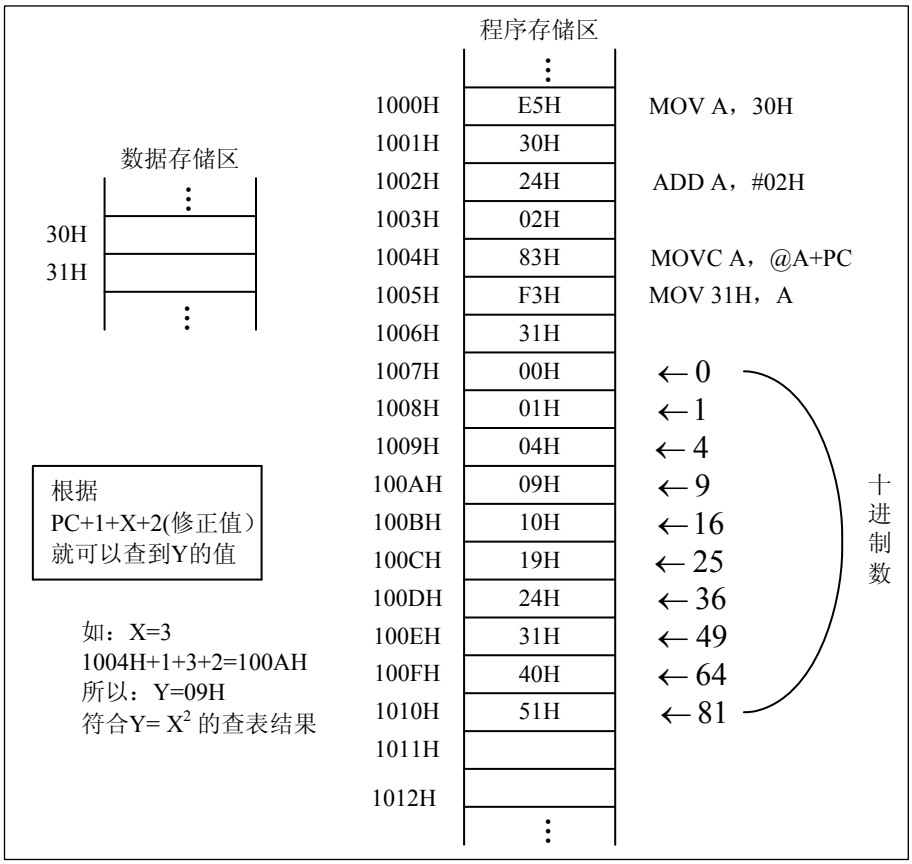


程序如下：

```
        ORG     1000H
START:  MOV     A, 30H              ；将查表的变量 X 送入 A
        MOV     DPTR, #TABLE        ；将查表的 16 位基地址 TABLE 送 DPTR
        MOVC    A, @A+DPTR          ；将查表结果 Y 送 A
        MOV     31H, A              ；Y 值最后放入 31H 中
TABLE:  DB    0, 1, 4, 9, 16
        DB    25, 36, 49, 64, 81
        END
```

方法二：采用 MOVC　A，@A+PC 指令实现，查表过程如图 3.3-6 所示。

图 3.3-6 例 3.21 方法二查表过程示意图



程序如下：

```
        ORG     1000H
START：MOV      A，30H              ；将查表的变量 X 送入 A
        ADD     A，#02H             ；定位修正
        MOVC    A，@A+PC            ；将查表结果 Y 送 A
        MOV     31H，A              ；Y 值最后放入 31H 中
TABLE：DB    0, 1, 4, 9, 16
        DB    25, 36, 49, 64, 81
        END
```

3.3.5 数据检索程序设计

数据检索是在数据区中查找关键字的操作。有两种数据检索方法，即顺序检索和对分检索，下面分别介绍。

1）顺序检索

所谓顺序检索就是把关键字与数据区中的数据从前向后逐个比较，判断是否相等。

例 3.22 假定数据区首地址是内部 RAM 20H，数据区长度为 8，关键字放在 2BH 单元，把检索成功的数据序号放在 2CH 单元中。

检索开始时应把 2CH 单元初始化为 00H。程序运行结束后，如 2CH 单元的内容仍为 00H，则表示没有检索到关键字；否则即为检索成功，2CH 单元的内容即为关键字在数据区中的序号（从 1 开始）。

程序设计：

```
        MOV   R0，#20H          ；数据区首地址
        MOV   R7，#08H          ；数据区长度
        MOV   2CH，#00H
        MOV   R2，#00H
        MOV   2BH，#KEY         ；关键字送 2BH 单元
NEXT：  INC   R2
        MOV   2AH，@ R0         ；数据区取数
        CLR   C
        MOV   A，2BH
        SUBB  A，@ R0           ；与关键字比较
        JZ    ENDP
        INC   R0
        DJNZ  R7，NEXT          ；继续
        MOV   R2，#00H
ENDP：  MOV   2CH，R2           ；送检索是否成功标志
HERE：  AJMP  HERE              ；结束
```

2）对分检索

对分检索的前提是数据已排好序，以便于按对分原则取数进行关键字比较，具体过程是：取数组中间位置的数与关键字比较，如果相等则检索成功。如果取数大于关键字，则下次对分检索的范围是从数据区起点到本次取数。如果取数小于关键字，则下次对分检索的范围是从本次取数到数据区终点。依次类推，逐次缩小检索范围，直到最后。

对分检索可以减少检索次数，大大提高了检索速度。但对分检索是一种递归算法，具体实现时首先要确定检索范围，范围的起点是 0，而终点是把最后一个数的序号加 1，这样才能使最后一个数也处在有效的检索范围之内，因为在程序中对分序号是通过起点与终点相加，然后除 2 取整而得到的。

例 3.23 假定检索数据区在内部 RAM 中，首地址为 data，其数据为无符号数，并以按升序排序。工作单元定义如下：

2AH—存放检索范围的起点。

2BH—存放检索关键字。

R0—先指向数据区首地址。检索开始后，则为对分读数地址。

R2—检索成功标志。如检索成功，则数据序号放入其中；否则置为 0FFH 状态。

R3—检索次数计数器。

R4—存放检索到的数据。

R7—存放检索范围的终点。

对分检索程序为：

```
        MOV   2AH，#00H          ；检索范围起点
        MOV   R7，#DVL           ；检索范围终点
        MOV   2BH，#KEY          ；关键字
        MOV   R3，#01H           ；检索次数初值
LOOP1： MOV   R0，DATA           ；数据区首址
        MOV   A，2AH
        ADD   A，R7              ；起点加终点
        CLR   C
        RRC   A                  ；除 2 取整
        MOV   R2，A              ；存放取数的序号
        CLR   C
        SUBB  A，2AH             ；判是否到范围边缘
        JZ    LOOP3              ；是边缘则转
        MOV   A，R2
        ADD   A，R0              ；形成取数地址
        MOV   R0，A
        MOV   A，@ R0            ；取数
        MOV   R4，A              ；取数放 R4 中
        CLR   C
        SUBB  A，2BH             ；与关键字比较
        JZ    LOOP5              ；相等则检索成功
        JNC   LOOP2              ；取数大则转
        MOV   2AH，R2            ；取数小，修改检索范围起点
        INC   R3                 ；检索次数加 1
        SJMP  LOOP1              ；继续
LOOP2： MOV   A，R2              ；取数大，修改检索范围终点
        MOV   R7，A
        INC   R3
        SJMP  LOOP1              ；继续
LOOP3： MOV   R0，DATA           ；达到边缘，比较数据是否为关键字
        MOV   A，@ R0
        CJNE  A，2BH，LOOP4
        MOV   R4，A              ；是关键字
        SJMP  LOOP5
LOOP4： MOV   A，#FFH            ；不是，送检索不成功标志
        MOV   R2，A
LOOP5： SJMP  LOOP5              ；结束
```

本章小结

本章主要介绍了 MCS-51 系列单片机汇编语言的程序结构、汇编语言的伪指令，以及汇编语言程序的编辑、汇编、仿真调试和目标代码的写入和运行。

在程序结构中主要介绍了顺序程序、循环程序、分支程序，以及子程序调用。

（1）简单程序又称为顺序程序，是一种顺序执行的程序。在这种程序中，没有分支，没有循环，没有子程序的调用。其结构简单，能实现一定的功能，是组成复杂程序的基础。

（2）循环程序是一种按照某种控制规律重复执行若干次，完成大量相同任务的程序。采用循环程序，不仅可以缩短程序长度，节约存储空间，而且可以提高编程效率和质量。

（3）分支程序是一种根据对某种条件的判断结果，决定程序不同走向的一种程序。分支的选择是通过条件转移指令、转移指令表和转移地址表等方式来实现的，分支程序从结构上可以分成单分支和多分支结构。

（4）子程序的特点是执行过程中需要有其他程序来调用，等程序运行结束又需要返回指令使其回到调用该程序的程序中。在调用子程序过程中常常需要解决好主程序和子程序的参数传递和现场保护、现场恢复等问题。

思考题与习题

3.1 编程实现将片内 35H—55H 单元中内容送到以 3000H 为首的存储区中。

3.2 设 5AH 单元中有一变量 X，请编写程序计算下述函数式，结果存入 5BH 单元。

$$\begin{cases} X^2-1, & X<0 \\ Y=X^2+8, & 10\le X\le 15 \\ 41, & X>0 \end{cases}$$

3.3 编程计算片内 RAM 区 50H—57H 8 个单元中数的算术平均值，结果存放于 54H 开始的单元中。

3.4 编写对一个多字节数做乘 10 的运算子程序。

3.5 设有两个长度均为 15 的数组，分别存放在以 2000H 和 2100H 为首的存储区中，试编程求其对应项之和，结果存放到以 2200H 为首的存储区中。

3.6 试编程把以 2000H 为首地址的连续 50 个单元的内容按升序排列，存放到以 3000H 为首地址的存储区中。

3.7 设有 100 个无符号数，连续存放在以 2000H 为首地址的存储区中，试编程统计奇数和偶数的个数。

3.8 将片内数据存储器地址为 1000H—1030H 的数据块，全部搬迁到片内 RAM 的 30H—60H 中，并将原数据块区域全部清零。

3.9 从 20H 单元开始有一个无符号数据块，其长度在 20H 单元中。求出数据块中的最小值，并存入 21H 单元。

3.10 在以 2000H 为首地址的存储区中，存放着 20 个用 ASCII 码表示的 0—9 之间的数，试编程将它们转换成 BCD 码，并以压缩 BCD 码（即一个单元存放二位 BCD 码）的形式存放在 3000H—3009H 单元中。

第 4 章　单片机的中断系统

前述内容中我们对单片机的结构有了一定的了解，虽说单片机五脏俱全，但要实现单片机系统的正常运行，则同其他微机系统一样，CPU 需不断地与外部 I/O 设备交换信息。这些具体的交换是如何实现呢?本章将简要介绍单片机系统中信息交换过程的控制，在此基础上讨论单片机的中断系统控制过程。

4.1　输入输出的控制方式

CPU 与外部设备交换信息的方式通常有以下 3 种：

- 无条件传送方式。
- 程序控制的查询传送方式。
- 中断控制传送方式。

无条件传送方式是指计算机不考虑外围设备的状态，CPU 可随时执行输入或输出指令，立即进行数据传送的一种方式。可以认为外围设备处于随时准备就绪的状态。这是一种最简单、最直接的传送方式，所需硬件少，编程简单，一般用于外围设备始终处于待命状态的场合。本节主要介绍程序控制的查询传送方式和中断控制传送方式。

4.1.1　程序控制的查询传送方式

1．程序按制的查询传送方式定义

CPU 与外部 I/O 设备交换信息时，通常存在 CPU 与 I/O 设备之间的速度匹配问题。为此程序控制的查询传送方式是通过 CPU 执行程序，查询 I/O 设备状态以解决传送问题。在此传送方式中，以 CPU 为主动方，I/O 设备为被动方。为了保证数据传送的正确性，单片机系统在进行数据传送之前，CPU 首先要查询外部设备是否处于“准备就绪”状态。对于输入操作，只需要知道外设是否已把要输入的数据准备好了；对于输出操作，则需要知道外设是否已把上次 CPU 输出的数据接收完毕。只有通过查询，CPU 才能确定是否发出访问外设的指令，以实现数据的交换。此查询方式就称为程序控制的查询传送方式，由于是在特定的条件下才能进行数据传送的，因此又称为程序控制的条件传送方式。

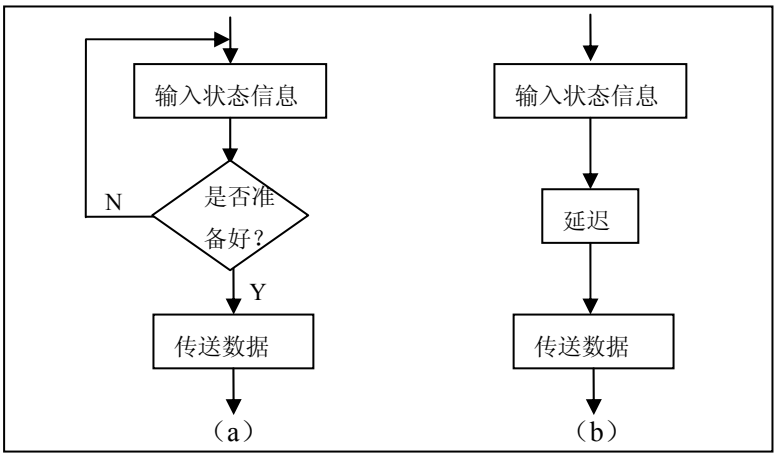
2．程序按制的查询传送方式的过程及流程图

程序按制的查询传送方式的执行过程为：查询→等待→数据传送，等到下一次数据传送时则重复上述过程。在此过程中，CPU 先查询外设所提供的状态信息，若状态满足则进行数据传送，反之则等待。等待期间，CPU 采用执行循环等待程序或通过调用延时子程序的方法来实现，循环等待和延迟等待的过程分别如图 4.1-1（a）图和（b）图所示。

3．程序控制的查询传送方式特点

从流程图 4.1-1 中，不难看出，（a）图中循环等待的方式和（b）图延迟等待的方式在一般的系统里都很容易实现，因此程序控制的查询传送方式的优点就是通用性好，可用于各类外部设备和 CPU 的数据传送，但是这种方式同时又有着一个很大的缺点是需要有一个等待过程，尤其在连续进行数据传送时，外设工作速度一般比 CPU 慢得多，所以 CPU 每完成一次数据传送后要等待较长的时间才能进行下一次的传送。而且在等待过程中，CPU 不能做其他操作，效率较低。为此，想要提高 CPU 的工作效率，人们通常采用中断控制的传送方式。

图 4.1-1 程序控制的查询传送方式流程图。(a) 循环等待；(b) 延迟等待。



4.1.2 中断控制传送方式

当 CPU 正在处理某项事务的时候，如果外界或内部发生了紧急事件，要求 CPU 暂停正在处理的工作转而去处理这个紧急事件，待处理完以后再回到原来被中断的地方，继续执行原来被中断了的程序，这样的过程称为中断。向 CPU 提出中断请求的源称为中断源。MCS-51 单片机一般允许有多个中断源。当几个中断源同时向 CPU 发出中断请求时，CPU 应优先响应最需紧急处理的中断请求。为此，需要规定各个中断源的优先级，使 CPU 在多个中断源同时发出中断请求时能找到优先级最高的中断源，响应它的中断请求。在优先级高的中断请求处理完了以后再响应优先级低的中断请求。

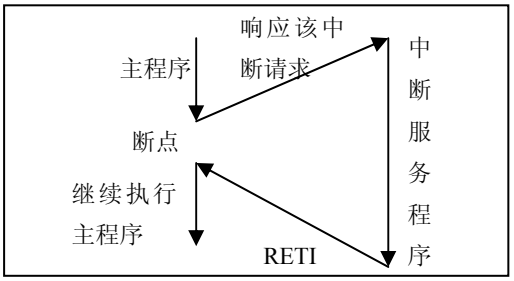
1. 中断、中断源、中断系统的概念

当 CPU 正在处理某件事情时，外部或内部发生的某一事件，如一个变化了的电平信号，一个脉冲沿的发生或定时器的计数溢出等请求提供给 CPU，CPU 在条件允许的情况下，中止当前的工作，转去处理所发生的事件。处理完毕后，CPU 再回到原来被中止的地方继续进行原来的工作，这个过程称为中断。产生中断的请求源称为中断源。实现中断功能的部件称为中断系统或中断机构。中断源向 CPU 提出的处理请求被称为中断请求。而 CPU 中止当前的事务，转去处理事件的过程，称为 CPU 的中断响应过程。对事件的处理过程称为中断服务或中断处理。处理完后，再回到被中止的地方，称为中断返回。

2. 中断过程

中断过程：当 CPU 正在执行主程序时，每隔一固定时间检测一次中断标志位，若有中断源发出了请求，CPU 中止主程序的执行，并自动把断点地址及相关寄存器的信息保存起来，去响应中断源的请求，执行中断服务程序。中断服务程序处理完毕，CPU 通过执行一条中断返回指令回到断点处，恢复相关寄存器的信息后，继续执行主程序。中断流程如图 4.1-2 所示。

图 4.1-2 中断流程图



3．中断的功能及特点

中断是单片机的一个重要功能。采用中断技术能实现如下功能。

(1)分时操作。单片机的中断系统可以使 CPU 与外设同时工作。系统上电，CPU 可在启动多个外设后，继续执行主程序，而被启动的外设开始进行准备工作。若多个外设同时向 CPU 发出请求，CPU 则根据请求信号的级别，分时为各外设提供服务，从而大大提高了 CPU 的利用率和输入输出的速度。

(2)实时处理。当单片机用于实时监控时，请求 CPU 提供服务的中断源是随机产生的。有了中断系统，CPU 就可以立即响应并给予处理。所谓实时控制就是要求微机能及时地响应被控对象提出的分析、计算和控制等请求，使被控对象保持在最佳工作状态，以达到预定的控制效果。

(3)故障处理。对 PC 机而言，在运行时往往会遇见一些故障，如电源断电、存储器奇偶校验出错、运算出错等。有了中断系统，CPU 就可以及时转去执行故障处理程序，自行处理故障而不需停机。

将程序控制的查询传送方式与中断控制传送方式进行比较，得出前者是 CPU 主动要求传送数据，而它又不能控制外设的工作速度，只能用原地“踏步”的等待方式来缓解速度匹配的问题；后者是外设主动提出数据传送的请求，CPU 在收到外设请求之后，才中止原来主程序的执行，去与外设交换数据。CPU 工作速度相对很快，交换数据所花费的时间很短，对于主程序来讲，只中断了一个瞬间，不会影响主程序的整体运行。

可见，中断方式的特点是 CPU 与外设之间的并行工作，能大大提高 CPU 的运行效率，使 CPU 能实现实时控制。

4.2 MCS-51 的中断控制系统

MCS-51 系列单片机的中断与定时系统简单且实用，其中 51 子系列具有 3 类共 5 个中断源，其中包括 2 个外部中断请求源，2 个定时/计数器中断请求源和 1 个串行口中断请求源。每个中断源都可以选择两个优先级。为了实现对各中断源的控制，MCS-51 还有 4 个用于中断控制的寄存器 IE、IP、TCON 和 SCON 等。当 CPU 支持中断屏蔽指令后，可将一部分或所有的中断关断，只有打开相应的中断控制位后，方可接收相应的可屏蔽中断请求。可通过程序设置中断控制寄存器的允许或屏蔽，设置中断的优先级。52 子系列具有 3 类共 6 个中断源。

4.2.1 MCS-51 的中断源

MCS-51 系列单片机的中断系统结构中含有 5 个中断源，它们是两个外部中断 INT0（P3.2）、INT1（P3.3），三个内部中断，其中包括两个片内定时/计数器溢出中断 IT0（P3.4）、IT1（P3.5）和一个串行口中断请求源 TX/RX。如图 4.2-2 所示。

- INT0：外部中断 0 请求，低电平有效，引脚为 P3.2。
- INTl：外部中断 1 请求，低电平有效，引脚为 P3.3。
- IT0：定时器/计数器 0 溢出中断请求。若对外部事件进行计数，则其引脚为 P3.4。
- IT1：定时器/计数器 1 溢出中断请求。若对外部事件进行计数，则其引脚为 P3.5。
- TX/RX：串行口中断请求。MCS-51 有一个全双工的串行接口，既可作为串行异步通信接口(UART)，也可以作为同步移位寄存器方式下的串行扩展接口。当串口完成一帧数据的发送或接收时，便请求中断。

各中断源中断服务程序的入口地址如表 4.2-1 所示。

表 4.2-1 中断源中断服务程序入口地址

中断源	中断服务程序入口地址
外部中断 0（INT0）	0003H
定时器/计数器 T0 中断	000BH
外部中断 1（INT1）	0013H
定时器/计数器 T1 中断	001BH
串行口中断 TX/RX	0023H

4.2.2 MCS-51 的中断控制

1．中断请求标志

(1)定时器控制寄存器 TCON 的中断请求标志位。

TCON 寄存器的格式如表 4.2-2 所示。

表 4.2-2 TCON 寄存器的格式

TCON	D7	D6	D5	D4	D3	D2	D1	D0
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H

TCON 被分成两部分，高 4 位用于定时器/计数器的中断控制，低 4 位用于外部中断的控制。各位含义如下。

● IT0：外部中断 0 的中断触发方式控制位。

当 IT0＝0 时，外部中断 0 程控为电平触发方式。CPU 在每一个机器周期 S5P2 期间采样引脚 P3.2 的输入电平。若外部中断 0 请求为低电平，则置 IE0 位为 1；若外部中断 0 请求为高电平，则置 IE0 位为 0。

当 IT0＝1 时，外部中断 0 程控为边沿触发方式。CPU 在每一个机器周期 S5P2 期间采样引脚 P3.2 的输入电平。如果在相继的两个机器周期采样过程中，一个机器周期采样到 P3.2 引脚为高电平，接着的下一个机器周期采样到 P3.2 引脚为低电平，则置 IE0 位为 1。直至 CPU 响应此中断时，硬件使 IE0 位清 0。

该位可以位寻址，可以被应用程序清零或置位。如：

```
…
SETB   IT0;   IT0 被置位，设定 INT0 为下降沿触发模式
CLR    IT0;   IT0 被清零，设定 INT0 为低电平触发模式
…
```

● IE0：外部中断 0 的中断请求标志位。当 CPU 检测到外部中断引脚 P3.2 上存在有效的中断请求信号时，硬件自动将 IE0 置 1。当 CPU 响应此中断时，硬件自动使 IE0 位清 0。

● ITl：外部中断 1 的中断触发方式控制位。其含义与 IT0 相同。

● IEl：外部中断 1 的中断请求标志位。其含义与 IE0 相同。

● TF0：定时器计/数器 T0 的溢出中断请求标志位。当启动 T0 计数以后，T0 从初值开始加 l 计数，计数器计到最高位产生溢出时，由硬件自动将 TF0 置 1，并向 CPU 发出中断请求。当 CPU 响应此中断时，硬件自动使 TF0 位清 0。

● TF1：定时器计/数器 T1 的溢出中断请求标志位。其含义与 TF0 相同。

(2)串行口控制寄存器 SCON 的中断请求标志位

串行口控制寄存器 SCON 的格式如表 4.2-3 所示。

表 4.2-3 SCON 寄存器的格式

SCON	D7	D6	D5	D4	D3	D2	D1	D0
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
位地址							99H	98H

表中的 D2～D7 位用于串行口方式设置和串行口发送/接收控制，将在第五章介绍，D1～D0 两位是设置串口的接收中断和发送中断标志 TI 和 RI，其意义如下：

● TI：串行口发送中断请求标志位。当 CPU 将每一个数据写入发送缓冲器 SBUF 时，就启动发送。每发送一帧串行数据后，硬件自动将 TI 置 1。当 CPU 响应此中断时，并不清除 TI，必须在中断服务程序中由软件对 TI 清零。

需要注意的是：MCS-51 系列单片机的发送数据过程的启动没有专用的控制位，而是执行到下面的语句：

```
MOV SBUF，A    ；将累加器 A 中的 8 位数据通过串行口发送出去
```

发送数据的过程即被自动启动。

● RI：串行口接收中断请求标志位。在串行口允许接收时，每接收完一帧串行数据后，由硬件置位 RI。与 TI 位相同，CPU 响应中断时不自动清除 RI，用户必须用软件对 RI 清 0。

定时器控制寄存器 TCON 和串行口控制寄存器 SCON 在中断系统中的位置如图 4.2-2 所示。

2．中断允许控制

在 MCS-51 系列单片机的中断系统中，中断的允许或禁止是由片内可进行位寻址的 8 位中断允许寄存器 IE 来控制的，如表 4.2-4 所示。

中断允许控制寄存器 IE 对中断源实施开放式或屏蔽式两级控制。所谓两级控制，是指有一个总的开关中断控制位 EA(或 IE- 7)，当 EA＝0 时，屏蔽所有的中断申请，即任何中断申请都不接受；当 EA＝1 时，CPU 开放中断，但 5 个中断源还要由 IE 低 5 位的各对应控制位的状态进行中断允许控制。

表 4.2-4 IE 寄存器的格式

IE	D7	D6	D5	D4	D3	D2	D1	D0
	EA	-	-	ES	ET1	EX1	ET0	EX0
位地址	AFH			ACH	ABH	AAH	A9H	A8H

表 4.2-4 中 D5，D6 两位未用，其余各位意义如下。

● EX0：外部中断 0 的中断允许位。EX0＝0，禁止外部中断 0 中断；EX0＝1，允许外部中断 0 中断。

● ET0：定时器/计数器 T0 的溢出中断允许位。ET0＝0，禁止 T0 中断；ET0＝1，允许 T0 中断。

● EXl：外部中断 l 的中断允许位。EX1＝0，禁止外部中断 1 中断；EX1＝1，允许外部中断 1 中断。

● ETl：定时器/计数器 T1 的溢出中断允许位。ET1＝0，禁止 T1 中断；ET1＝1，允许 T1 中断。

● ES：串行口中断允许位。ES＝0，禁止串行口中断；ES＝1，允许串行口中断。

● EA：全局中断控制位。EA＝0，屏蔽所有的中断申请；EA＝1，CPU 开放中断。对各中断源的中断请求是否允许，还要取决于各中断源的中断允许控制位的状态。

比如允许外部中断 0、1 的中断，禁止其他中断，可设置 IE：

（1）用字节操作指令。

```
MOV  IE，  #85H   或  MOV  A8H，  #85H
```

（2）用位操作指令

```
SETB   EX0    ；  外部中断 0 允许中断
SETB   EX1    ；  外部中断 1 允许中断
SETB   EA     ；  CPU 开放中断
```

3．中断优先级控制

MCS-51 有两个中断优先级，即高优先级和低优先级，每个中断源都可设置为高或低中断优先级，以便 CPU 对所有的中断实现两级中断嵌套。在响应中断时，CPU 先响应高优先级中断，然后响应低优先级中断。如果有一低优先级的中断正在执行，那么高优先级的中断出现中断请求时，CPU 则会响应这个高优先级的中断，即高优先级的中断可以打断低优先级的中断。而若 CPU 正在处理一个高优先级的中断，此时，即便是有低优先级的中断发出中断请求，CPU 也不会理会这个中断，而是继续执行正在执行的中断服指令后，才会响应新的中断请求。

中断优先级控制寄存器 IP 是用户对中断优先级控制服务程序，直到程序结束，执行最后一条返回指令返回主程序，然后再执行另一条的基础。若 IP 中某位设为 1，相应的中断就设置为高优先级，否则就设置为低优先级。中断优先级控制寄存器的格式如表 4.2-5 所示。

表 4.2-5 IP 寄存器的格式

IP	D7	D6	D5	D4	D3	D2	D1	D0
	-	-	-	PS	PT1	PX1	PT0	PX0
位地址				BCH	BBH	BAH	B9H	B8H

IP 寄存器中的有效控制位的含义如下。

● PX0：外部中断 0 的中断优先级控制位，PX0=1，外部中断 0 被定义为高优先级中断，PX0=0，外部中断 0 被定义为低优先级中断。

● PT0：定时器/计数器 T0 的中断优先级控制位，PT0=1，定时器/计数器 T0 被定义为高优先级中断，PT0=0，定时器/计数器 T0 被定义为低优先级中断。

● PXl：外部中断 1 的中断优先级控制位，其作用与设置同 PX0。

● PTl：定时器/计数器 T1 的中断优先级控制位，其作用与设置同 PT0。

● PS：串行口中断优先级控制位。PS=1，串行口中断被定义为高优先级中断，PS=0，串行口中断被定义为低优先级中断。

MCS-51 内部中断系统对各中断源的中断优先级有一个统一的规定，称为自然优先级（也称为缺省优先级）。

表 4.2-6 MCS-51 内部各中断源中断优先级的顺序

中断源	中断标志	缺省优先级
外部中断 0	IE0	最高
定时器/计数器 T0	TF0	↓
外部中断 1	IE1	↓
定时器/计数器 T1	TF1	↓
串行口中断	TI，RI	最低

CPU 在中断优先级 IP 寄存器的配合下，可实现如下控制功能。

（1）按内部查询顺序排队。

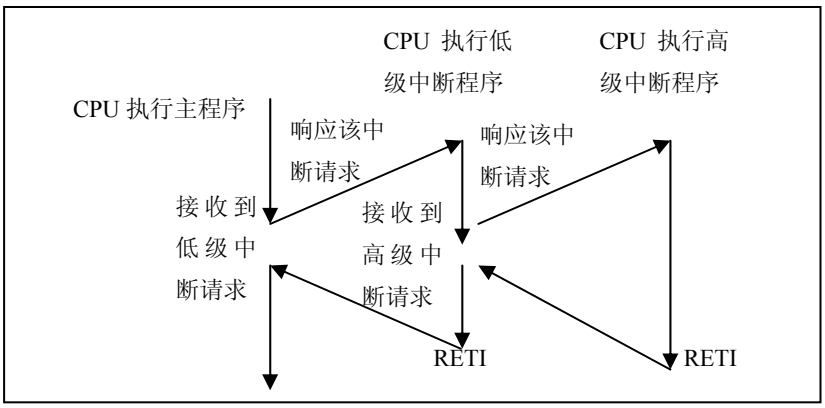
因为单片机系统中有多个中断源，时常会出现数个中断源同时提出中断请求的情况。这就要求用户事先根据各中断源的轻重缓急，为每一个中断源确定一个可供 CPU 为其服务的顺序号。当多个中断源同时产生中断请求时，CPU 则依据顺序号的次序给予响应。

（2）实现中断嵌套。

若 CPU 正在响应一个中断请求时，又出现了一个优先级比它高的中断请求，则 CPU 中

止正在处理的中断服务程序，保护当前断点，转去响应优先级更高的中断请求，并为其服务。待服务结束后，再继续执行原来被中止的优先级较低的中断服务程序。这种处理中断的过程称为中断嵌套，此中断系统称为多级中断系统。中断嵌套的流程如图 4.2-1 所示。

图 4.2-1 中断嵌套的流程



例： 试根据要求设置 IP 寄存器。设 MCS-51 的片外中断为高优先级（INT0、INT1），片内中断为低优先级（T0、T1、串行口）。

解：（1）用字节操作指令。

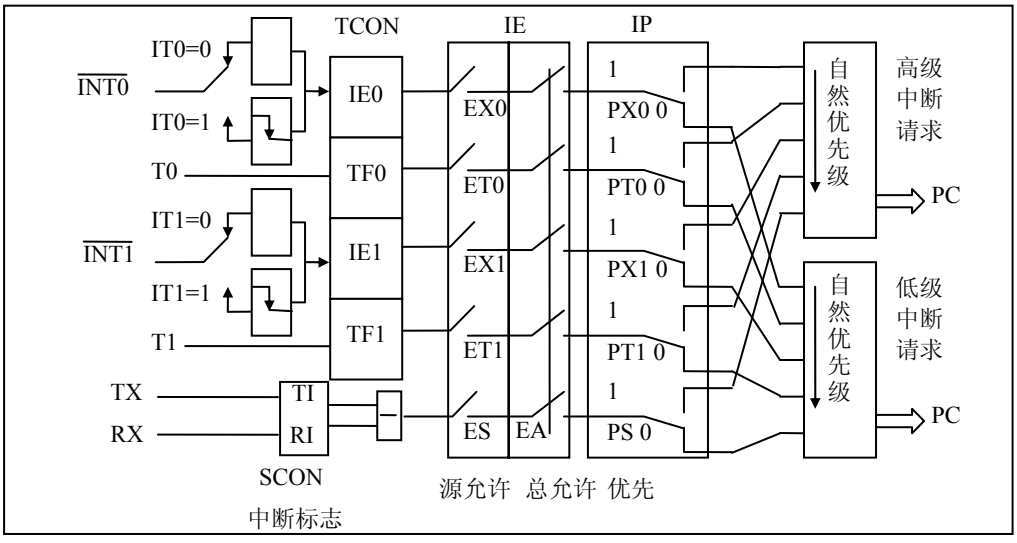
MOV IP，#05H 或 MOV 0B8H，#05H

（2）用位操作指令。

```
SETB    PX0     ；置外部中断 0 为高优先级
SETB    PX1     ；置外部中断 1 为高优先级
CLR     PT0     ；置 T0 的溢出中断为低优先级
CLR     PT1     ；置 T1 的溢出中断为低优先级
CLR     PS      ；置串行口的中断为低优先级
```

在 51 系列单片机中，5 个中断源发起中断以后，中断允许控制寄存器 IE 的相应位的设置情况决定着是否接受这些中断，在接受的情况下 CPU 结合中断优先级控制寄存器 IP 的设置情况和缺省优先级做出相应的处理，也就是中断响应。MCS-51 中断系统的整体结构如图 4.2-2 所示。

图 4.2-2 MCS-51 系列单片机中断系统结构



4.2.3 MCS-51 的中断响应过程

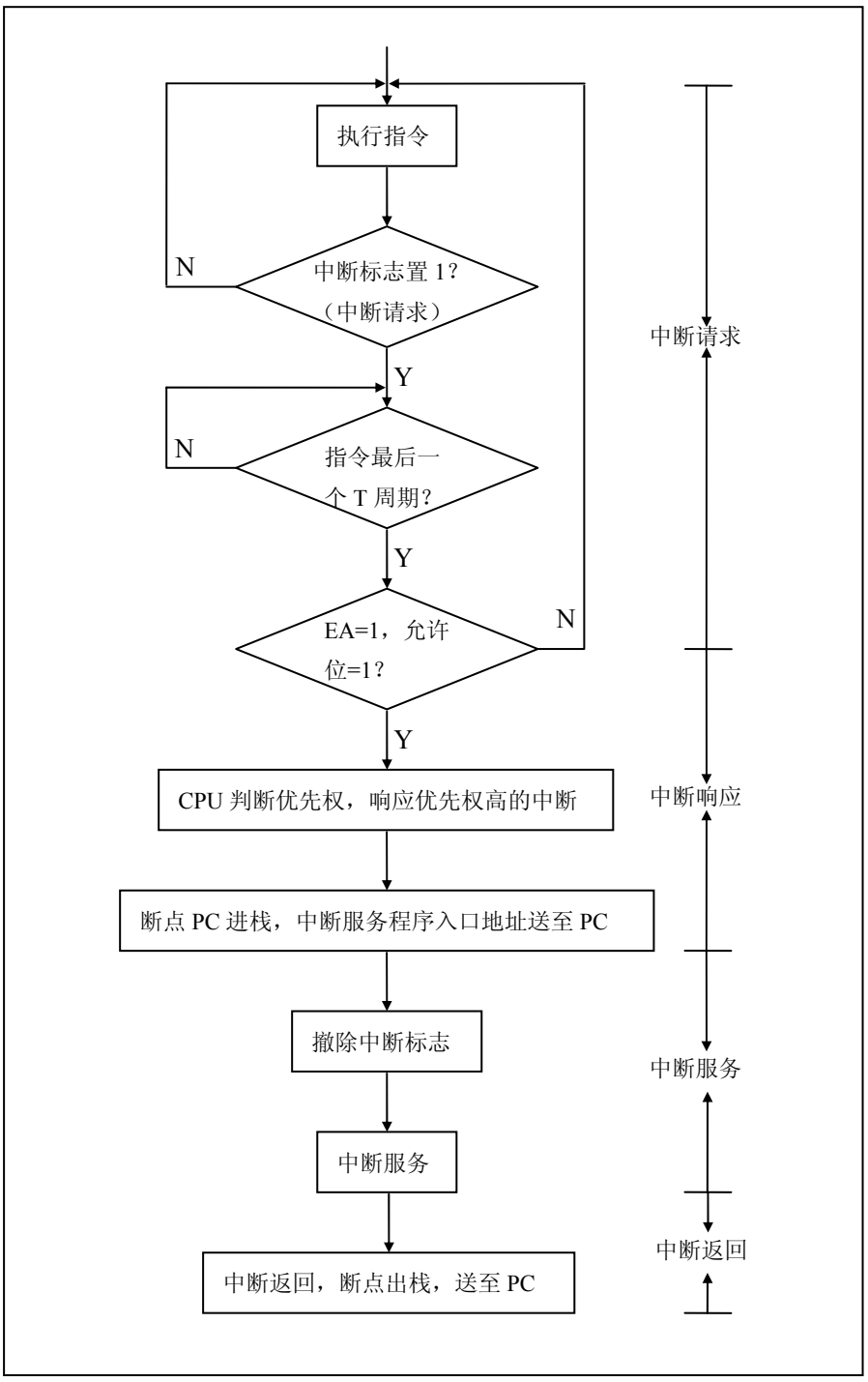
1.中断处理过程

中断处理过程分为四个阶段：中断请求、中断响应、中断服务、中断返回。

MCS-51 系列单片机的中断处理流程如图 4.2-3 所示。

CPU 执行程序时，在每一个指令周期的最后一个 T 周期都要检查是否有中断请求；如果有中断请求，寄存器 TCON 的相应位会置 1；CPU 查到 1 标志后，如果允许，进入中断响应阶段；如果中断被禁止，或没有中断请求，继续执行下条指令。

图 4.2-3 中断处理流程图



在中断响应阶段，如果有多个中断源，CPU 判断哪个的优先级高，优先响应优先级高的中断请求，阻断同级或低级的中断，硬件产生子程序调用指令，将断点 PC 压入堆栈，将所响应的中断源的矢量地址送主 PC 寄存器，转到中断服务程序执行。

中断服务是完成中断处理的事务，用户根据需要编写中断服务程序，程序中应注意将主程序中需要保护的寄存器内容进行保护，中断服务完毕后应恢复这些寄存器的内容。保护现场和恢复现场的过程可以通过堆栈操作完成。

中断返回是通过执行一条 RETI 中断返回指令完成的，该指令使堆栈中被压入的断点地址弹到 PC，从而返回主程序的断点，继续执行主程序。另外 RETI 指令还有恢复优先级状态触发器的作用，因此不能以 RET 指令代替 RETI 指令。

中断请求和中断响应过程都是由硬件完成的。

由上可见，MCS-51 系列单片机响应中断后，不会自动保护标志寄存器（PSW 程序状态字）、不会自动保护现场、不会自动关中断、不会自动发中断响应信号，这些是和 8086CPU 有差别的。

若某个中断源通过编程设置，处于被打开的状态，并满足中断响应的条件，CPU 就会响应中断。然而下面三种情况单片机不响应此中断：

（1） 当前正在执行的那条指令没执行完。

（2） 当前响应了同级或高级中断。

（3） 正在操作 IE，IP 中断控制寄存器或执行 RETI 指令。

在正常的情况下、从中断请求信号有效开始，到中断得到响应，通常需要 3～8 个机器周期。

2.中断请求的撤除

MCS-51 的 5 个中断源也可分为三种类型，即外部中断、定时器/计数器溢出中断和串行口中断。这三种类型的中断请求撤除的方法是不同的，现分别介绍如下。

(1)外部中断请求的撤除

外部中断请求有两种触发方式，电平触发和负边沿触发。对于这两种不同的触发方式，其中断请求撤除的方法是不同的。

在负边沿触发方式下，外部中断标志位 IE0 或 IE1 是依靠 CPU 两次检测 $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$) 上的负边沿触发电平状态而置位的。CPU 在响应中断时，由硬件自动复位 IE0 或 IE1，用户也不必专门为它们撤除。另外，外部中断源在得到 CPU 中断服务时是不可能再在 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 上产生负边沿而使相应的中断标志位置位的。

在电平触发方式下，外部中断标志位 IE0 或 IE1 是依靠 CPU 检测 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 上的低电平而置位的。尽管 CPU 在响应中断时能由硬件自动复位 IE0 或 IEl，但若外部中断源不能及时撤除它在 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 上的低电平，就会使已经复位的 IE0 或 IEl 再次置位，这是绝对不允许的。因此，电平触发型外部中断请求的撤除必须使 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 上的低电平随着其中断被响应而变为高电平。一种可供采用的电平触发型外部中断请求撤除的电路如图 4.2-4 所示。

图 4.2-4 电平触发型外部中断请求撤除的电路

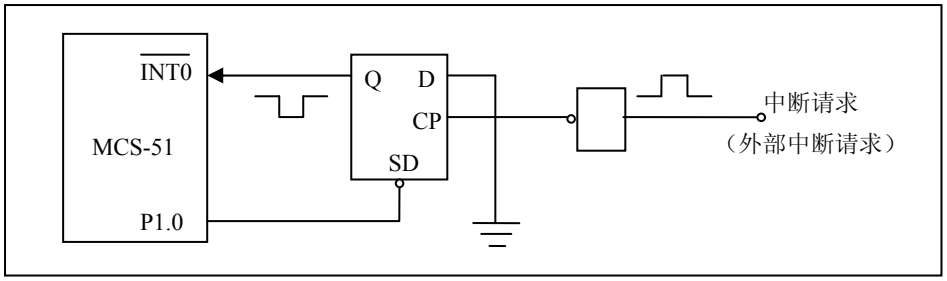


图 4.2-4 中，D 锁存器的作用是锁存外部中断请求的低电平信号，并由 Q 端输出至 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ 端，供 CPU 检测。D 触发器的异步置 1 端接 MCS-51 的一条 I/O 口线(如 P1.0)，此口线平时为“1”，对 D 触发器的输出状态无影响。当中断响应后，为了撤除中断请求，只要在 P1.0 口输出一个负脉冲，使触发器置 1，从而撤除了低电平的中断请求。负脉冲信号可以在中断服务程序中用如下指令来实现。

(2) 定时器/计数器溢出中断请求的撤除

定时器/计数器溢出中断请求的标志位 TF0 和 TF1 因定时器/计数器溢出中断源的中断请求的输入而置位，因定时器/计数器溢出中断得到响应而由硬件自动复位成“0”状态，用户不必专门为它们撤除。

(3)串行口中断请求的撤除

串行口中断请求的标志位 TI 和 RI 不能由硬件自动复位。这是因为 MCS-51 进入串行口中断服务程序后需要对它们进行检测，以测定串行口正在接收中断还是发送中断。为了防止 CPU 再次重复响应这类中断，用户需要在中断服务程序的适当位置通过如下指令将它们撤除：

```
CLR   TI   ；撤除发送中断请求标志
CLR   RI   ；撤除接收中断请求标志
```

或采用字节型指令：ANL　SCON，#0FCH。

4.2.4　MCS-51 的中断程序设计

用户对中断的控制和管理，实际上是对 4 个与中断有关的寄存器 IE、TCON、IP、SCON 进行控制或管理。这几个寄存器在单片机复位时是清零的，因此必须根据需要对这几个寄存器的有关位进行预置。在中断程序的编制中应注意：

（1）开中断总控开关 EA，置位中断源的中断允许位。

（2）对于外部中断 INT0、INT1，应选择中断触发方式是低电平触发，还是下降沿触发。

（3）对于多个中断源中断，应设定中断优先级，预置 IP。

1.怎样编写中断服务程序

中断程序的结构及内容与 CPU 对中断的处理过程密不可分，它通常分为两大部分，即主程序和中断服务程序。

(1) 主程序。

1)设置主程序起始地址。

51 系列单片机复位后，PC＝0000H，而各中断源的入口地址如表 4.2-1 所示。因 0000H—0003H 只有 3byte 单元，无法写主程序，所以编程时应在 0000H 处写一条转移指令，使 CPU 在执行程序时，从 0000H 跳过各中断源的入口地址。主程序是以跳转的目的地址为起始地址开始编写，一般可从 0100H 开始。如表 4.2-7 所示。

2)初始化内容。

初始化是对将要用到的单片机内部部件或所扩展芯片进行初始工作状态设置。当 51 系列单片机复位后，中断控制寄存器 IE 和中断优先级寄存器 IP 的内容均为 00H，所以应对 IE、IP 进行初始化编程，以开放中断，即允许某些中断源中断和设置中断优先级等。

(2)中断服务程序。

1) 中断服务程序的起始地址。

当 CPU 接收到中断请求信号并给予响应后，CPU 把断点处的 PC 内容压入栈中保存，之后转入相应的中断服务程序入口处执行。MCS-51 单片机的中断系统内 5 个中断源的入口地址彼此相距很近，仅为 8byte，如果中断服务程序很短，且少于 8byte，则可从系统规定的

中断服务程序入口地址开始，直接编写中断服务程序。通常中断服务程序的容量是远远大于8byte 的，那么应采取与主程序相同的方法．只在对应的入口地址处写入一条转移指令，并以转移指令的目的地址为中断服务程序的起始地址进行程序编写。该目的地址不应落在主程序存储空间中，如表 4.2-8 所示。

2)中断服务程序编制中的注意事项。

首先，需要进行现场保护；其次，要及时清除那些不能被硬件自动清除的中断请求标志位，以避免产生错误的中断；然后，在中断服务程序中，PUSH 指令与 POP 指令必须成对使用，否则不能正确地返回断点处；最后，需注意主程序和中断服务程序之间的参数传递与主程序和子程序的参数传递方式相同。

表 4.2-7 主程序地址在程序存储器中的安排

0000H	LJMP
	00
	60
	⋮
	⋮
0060H	主程序
	⋮
	⋮

表 4.2-8 中断服务程序入口地址

0000H	LJMP
	03
	00
	⋮
	⋮
0300H	中断服务程序
	⋮
	⋮

2.中断服务程序举例

例：中断加查询扩展外部中断源。图 4.2-5 所示电路中 P1.0～P1.3 为外部中断源的输入信号位，P1.4～P1.7 为输出信号用来驱动 LED 显示。该电路要求系统正常工作时，LED 不点亮，而出现故障时相应的输入线由低电平变为高电平，对应的输出线置为高电平，相应的 LED 被点亮。（对应关系为 P1.0 对 P1.4，其他类推。）

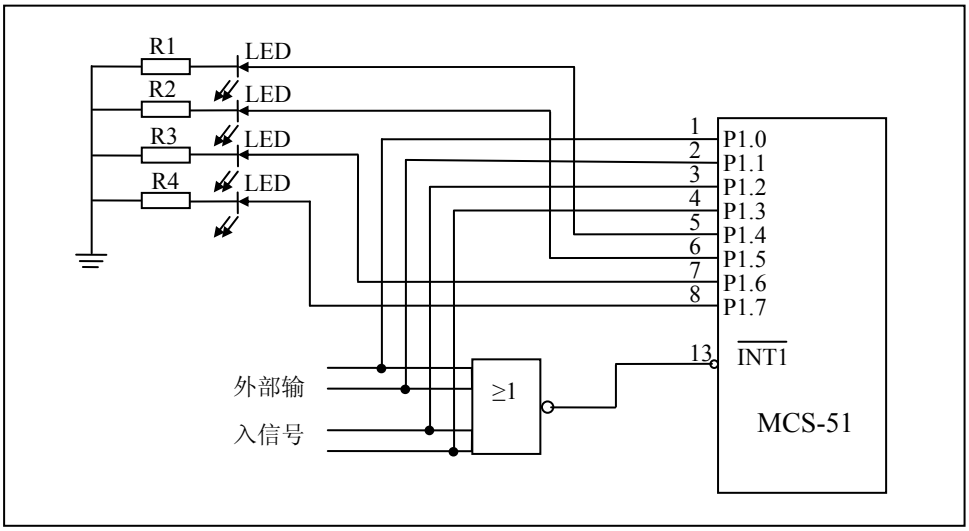
解：当图 4.2-5 中某部分出现故障时，相应的输入线由低电平变为高电平，经或非门送到 $\overline{\text{INT1}}$ 引脚端，以产生向 MCS-51 的中断请求信号（设为边沿触发方式）。

在中断服务程序中，应先将各故障信号读入后进行查询，再做出相应的显示。

```
        ORG     0000H       ；PC=0000H
        AJMP    MAIN        ；系统上电，执行主程序
        ORG     0013H       ；外部中断 1 入口地址
        AJMP    LOOP        ；转移至中断服务程序
MAIN：  WMOV    P1，#00H    ；P1 口复位，清中断和显示
        SETB    EX1         ；允许 INT1 中断
        SETB    IT1         ；INT1 中断选用边沿触发方式
        SETB    EA          ；CPU 开中断
HALT：  SJMP    HALT        ；等待中断
LOOP：  JNB     P1.0，L1    ；查询中断源，若 P1.0=0，则转移至 L1
        SETB    P1.4        ；当 P1.0=1，则置 P1.4=1，使相应的 LED 点亮
L1：    JNB     P1.1，L2    ；查询中断源，若 P1.1=0，则转移至 L2
        SETB    P1.5        ；当 P1.1=1，则置 P1.5=1，使相应的 LED 点亮
L2：    JNB     P1.2，L3    ；查询中断源，若 P1.2=0，则转移至 L3
        SETB    P1.6        ；当 P1.2=1，则置 P1.6=1，使相应的 LED 点亮
L3：    JNB     P1.3，L4    ；查询中断源，若 P1.3=0，则转移至 L4
```

```
        SETB    P1.7            ；当 P1.3=1，则置 P1.7=1，使相应的 LED 点亮
L4:     RETI                    ；中断返回
        END
```

图 4.2-5 中断加查询扩展外部中断电路



本章小结

MCS-51 的 5 个中断源中，两个为外部中断，$\overline{\text{INT0}}$（P3.2 脚）和 $\overline{\text{INT1}}$（P3.3 脚）输入中断请求，两个为片内定时器/计数器溢出中断请求 IT0（P3.4 脚）和 IT1（P3.5 脚），一个为片内串行口中断请求 TX（发送中断）或 RX（接收中断）。这些中断请求标志分别由特殊功能寄存器 TCON 和 SCON 的相应位锁存。TCON 还可以选择外中断的中断触发方式——电平触发还是边沿触发。

中断允许控制寄存器 IE 决定了哪个中断请求被接受，而在优先级控制寄存器 IP 中可以设置各中断源的优先级，每个中断源都可设置为高或低中断优先级，再结合缺省优先级，CPU 能够对所有的中断实现两级中断嵌套。一个正在执行的低优先级中断服务程序能被高优先级中断申请所中断，但不能被另一个低优先级中断源所中断。正在执行的高优先级中断服务程序不能被其他中断申请所打断。

MCS-51 的中断响应过程分为中断请求、中断响应、中断服务、中断返回四个阶段，缺一不可。如果需要撤除中断请求还应该根据中断源的不同选择相应的方式。

思考题与习题

4.1 什么是中断?什么是中断源?

4.2 微型机引进中断技术后有什么好处?

4.3 MCS-51 提供了哪几种中断源?在中断管理上各有什么特点?

4.4 MCS-51 响应中断的条件是什么，CPU 响应中断时，不同的中断源，其中断入口地址各是多少？

4.5 MCS-51 的外部中断有哪两种触发方式?应如何选择和设定？

4.6 MCS-51 单片机的中断系统中有几个优先级?如何设定?

第 5 章　MCS-51 单片机的定时/计数器与串行口

5.1　MCS-51 单片机的定时/计数器

5.1.1　定时/计数器的功能概述

在工业检测、控制系统及智能仪器等应用中，经常需要用到外部时钟，以实现定时或者延时控制，也经常需要用到外部计数器，以实现对外部事件的计数。为满足这方面的需要，几乎所有的单片机系统都集成了定时/计数器。MCS-51 单片机可提供两个 16 位的定时/计数器，即 T0 和 T1，它们均可作定时器或计数器使用，为单片机提供计数或定时功能。

对于定时/计数器来说，不管是独立的定时器芯片还是单片机内集成定时器，一般都具有以下共性：

1） 定时/计数器有多种工作模式，可以是定时模式，也可以是计数模式。

2）定时/计数器的计数值是可变的，但计数的最大值是有限的，取决于计数器的位数。计数的最大长度限定了定时器的最大值。

3）当到达设定的定时或计数值时发出中断请求，以便实现定时控制。

MCS-51 单片机中的定时/计数器有两种工作模式，即计数器工作模式和定时器工作模式。当工作在计数模式时，计数器记录的是外界发生的事件，当工作在定时模式时，定时器由单片机内部提供的一个非常稳定的计数源进行定时。这两种工作模式并没有本质的区别，只是计数脉冲的来源不同而已。

1）计数器工作模式。

当计数源来自 CPU I/O 引脚的外部信号时，称为计数模式。计数功能是对外来脉冲进行计数。MCS-51 单片机芯片有 T0（P3.4）和 T1（P3.5）两个输入引脚，分别是这两个计数器的计数输入端。每当计数器的计数输入引脚的脉冲发生负跳变时，计数器加 1。

2）定时器工作模式。

当计数源来自相对稳定的系统时钟信号时称为定时模式。这个计数源是由单片机的晶振经过 12 分频以后获得的一个脉冲源。设单片机的晶振是 12MHz，则它提供给计数器的脉冲时间间隔是 1μs。定时器计数脉冲的时间间隔与晶振有关。

MCS-51 单片机的定时/计数器有 4 种工作方式，即方式 0、方式 1、方式 2、方式 3，其控制字均在相应的特殊功能寄存器中，通过对它的特殊功能寄存器的编程，用户可以方便地选择定时/计数器的 2 种工作模式和 4 种工作方式（如是定时还是计数；硬件启动还是软件启动；计数长度，即作为 16 位计数器使用还是作为 8 位计数器使用；溢出后是重装初值还是从 0 开始计数等）。

在了解了 MCS-51 单片机内定时/计数器的基本功能后，下面具体介绍 MCS-51 单片机内定时/计数器的结构、功能、工作方式、初始化，有关的特殊功能寄存器、状态字、控制字的含义，以及工作模式和工作方式的选择等。

5.1.2　定时/计数器的结构、控制及工作方式

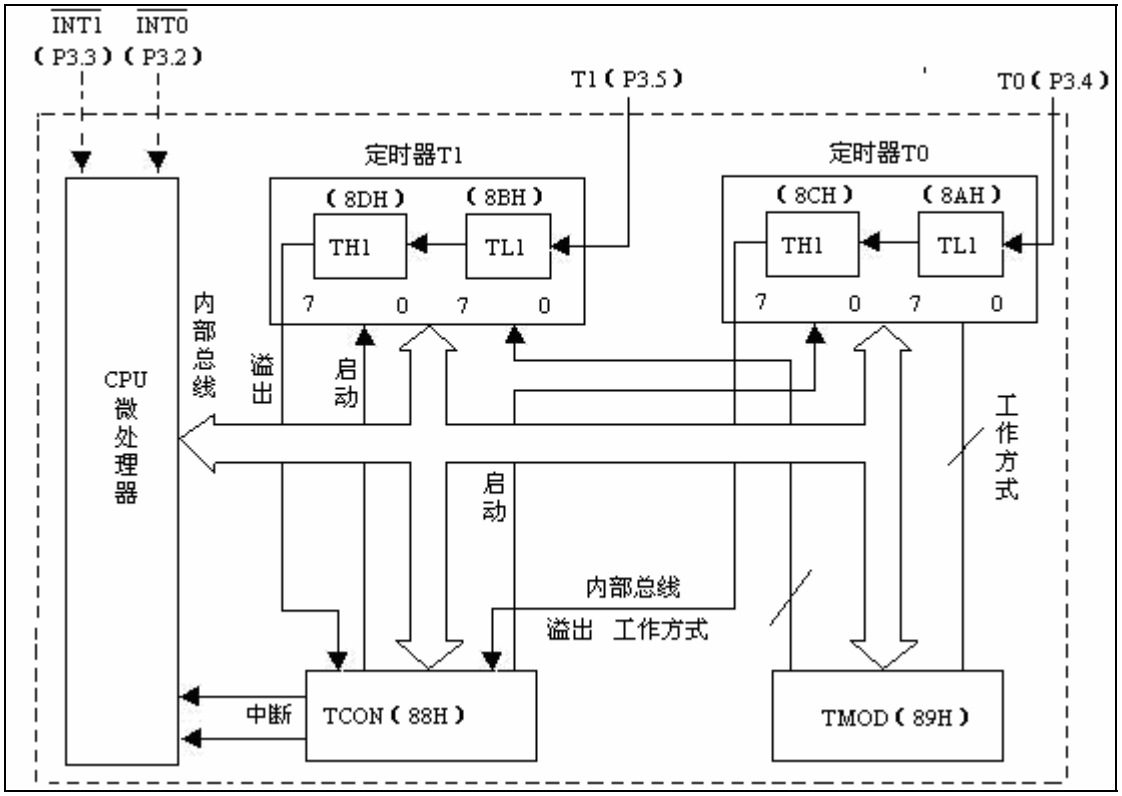
MCS-51 单片机内两个 16 位的可编程定时/计数器 T0、T1，均是二进制加法计数，当计数器计满回零时能自动产生溢出中断请求，表示设定时间已到或者计数已满。两个定时/计数器是可编程的，即可通过编程设定其作为定时器用还是作为计数器用。每种模式下的工作方式也由程序设定，其控制字和状态字均在相应的特殊功能寄存器中，通过对特殊功能寄存

器的编程，就可方便地选择适当的工作方式。此外，定时模式下的定时时间和计数模式下的计数值都可通过编程实现。

5.1.2.1 定时/计数器内部结构

MCS-51 单片机内定时/计数器 T0、T1 的结构如图 5.1-1 所示。

图 5.1-1 定时/计数器 T0、T1 的逻辑结构图



由图可知，MCS-51 单片机定时/计数器由定时器 T0、定时器 T1、定时器方式控制寄存器 TMOD 和定时器控制寄存器 TCON 组成。

定时/计数器的核心是 16 位的加法计数器。定时/计数器由两个 8 位的特殊功能寄存器 TH0（高 8 位）和 TL0（低 8 位）构成 16 位加法计数器。同样，T1 也是由两个 8 位的特殊功能寄存器 TH1（高 8 位）和 TL1（低 8 位）构成 16 位加法计数器。TH0 和 TL0 的访问地址分别为 8CH、8AH，TH1 和 TL1 的访问地址分别为 8DH 和 8BH。每个寄存器可单独访问。在 MCS-51 单片机中与定时/计数器 T0、T1 有关的寄存器为 TMOD 和 TCON。其中，工作方式控制寄存器 TMOD 用于设定定时/计数器 T0、T1 的工作方式，控制寄存器 TCON 用于控制定时/计数器 T0、T1 的启动与停止，同时管理定时器的溢出标志。

TMOD 和 TCON 与定时器 T0、定时器 T1 通过内部总线及逻辑电路连接。

当设置了定时器的工作方式并启动定时器工作后，定时器将按照设定的工作方式独立工作，不再占用 CPU 的操作时间，只有在计数器计满溢出时才可能中断 CPU 当前的操作。

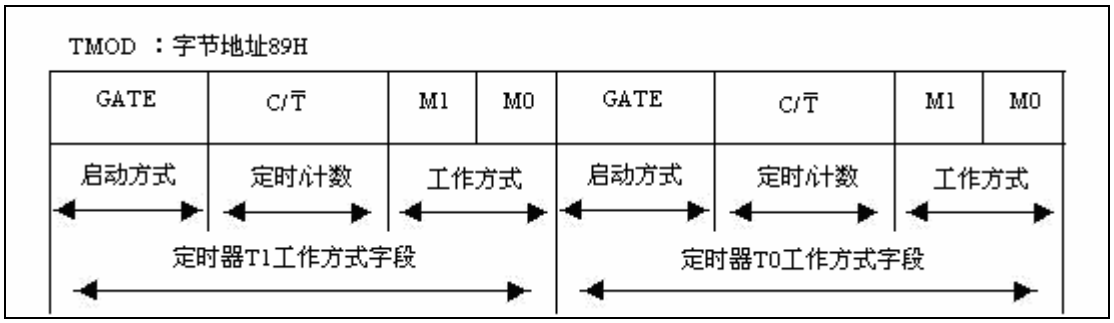
5.1.2.2 定时/计数器控制

在启动定时/计数器工作之前，CPU 必须将一些命令(称为控制字)写入定时/计数器中，这个过程称为定时/计数器的初始化。定时/计数器的初始化通过定时/计数器的方式寄存器 TMOD 和控制寄存器 TCON 完成。

（1）工作方式控制寄存器 TMOD

TMOD 的功能是用于控制定时/计数器 T0、T1 的工作方式。TMOD 寄存器不能位寻址，只能用字节传送指令设置其内容，字节地址为 89H，其格式及每位的含义如图 5.1-2 所示。

图 5.1-2 TMOD 寄存器格式及各位含义



由图可知，TMOD 分成两部分，每部分 4 位，低 4 位(D3--D0)控制定时/计数器 T0 的工作方式，而高 4 位(D7--D4)则控制定时/计数器 T1 的工作方式。

M1M0 是工作方式选择位，定时/计数器一共有 4 种工作方式，如表 5.1-1 所示。

表 5.1-1 定时/计数器工作方式

M1M0	工作方式	方式说明
00	方式 0	13 位定时/计数器，此种方式由于装入初值容易出错，故不推荐使用方式 0
01	方式 1	16 位定时/计数器
10	方式 2	具有自动重装初值功能的 8 位定时/计数器
11	方式 3	定时/计数器 T0 可以工作在这一方式，相当于两个 8 位的定时/计数器。但 T0 工作于方式 3 时，占用了定时/计数器 T1 的部分资源，限制了 T1 的使用范围（此时 T1 可作为串行口发送/接收波特率发生器）

$C/\overline{T}$ 是定时/计数器模式选择位。当 $C/\overline{T}=0$ 时，计数脉冲来自 CPU 内，计数脉冲频率是系统时钟信号频率的 12 分频，为定时模式；当 $C/\overline{T}=1$ 时，计数脉冲来自 P3.4 或 P3.5 引脚，为计数模式。定时/计数器到底作为哪一种功能用，可由用户根据需要通过编程自行设定。

GATE 是启动方式控制位，用于控制定时/计数器的启动是否受外部中断请求信号的影响，如果 GATE=1，则定时/计数器 T0 的启动受芯片引脚 $\overline{INT0}$ (P3.2)控制，定时/计数器 T1 的启动受芯片引脚 $\overline{INT1}$ (P3.3)控制；如果 GATE=0，则定时/计数器的启动与引脚 $\overline{INT0}$、$\overline{INT1}$ 无关。

（2）控制寄存器 TCON

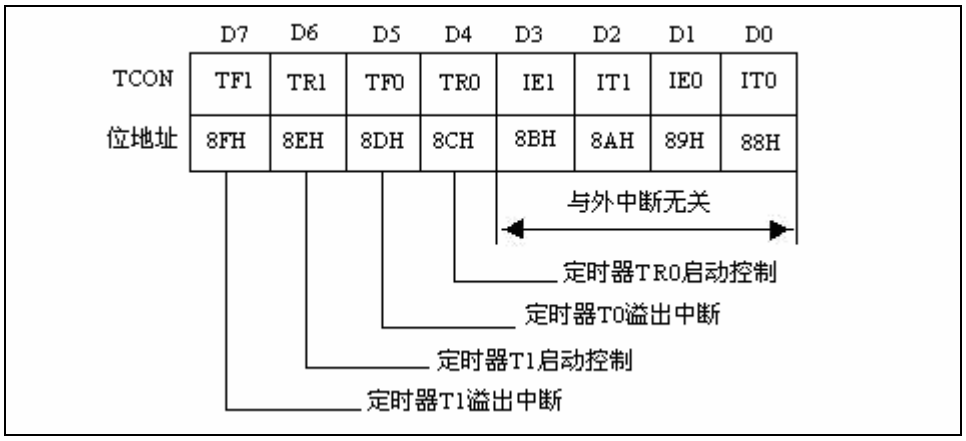
TCON 的功能是控制定时/计数器 T0、T1 的启动与停止，以及管理定时器的溢出标志。该寄存器的格式及各位含义如图 5.1-3 所示。

TCON 有两部分，每部分 4 位，低 4 位(D3~D0)用于外部中断的控制（已在中断控制中讲过），高 4 位(D7--D4)用于定时/计数器的中断控制。

TR0 和 TR1：TR0 为定时/计数器 T0 的启动/停止控制位，其状态可通过软件设定。若 TR0=1，则定时/计数器 T0 启动立即开始计数；若 TR0=0，则定时/计数器 T0 停止计数。例如，可用指令 SETB TR0 置位 TR0 以启动定时/计数器 T0 运行；用指令 CLR TR0 复位指令关闭定时/计数器 T0 的工作。TR1 为定时/计数器 T1 的启动/停止控制位，其功能与同 TR0。

TF0 和 TF1：TF0 为定时/计数器 T0 的溢出中断标志位。当 T0 计数溢出（由全“1”变为全“0”）时，TF0 由硬件自动置位，即 TF0=1，并在允许中断的情况下，向 CPU 发出中断请求信号，当 CPU 响应中断并转向中断服务程序时，TF0 被硬件自动复位（TF0=0）。TF1 为定时/计数器 T1 的溢出中断标志位，其功能同 TF0。

图 5.1-3　TCON 寄存格式及各位含义



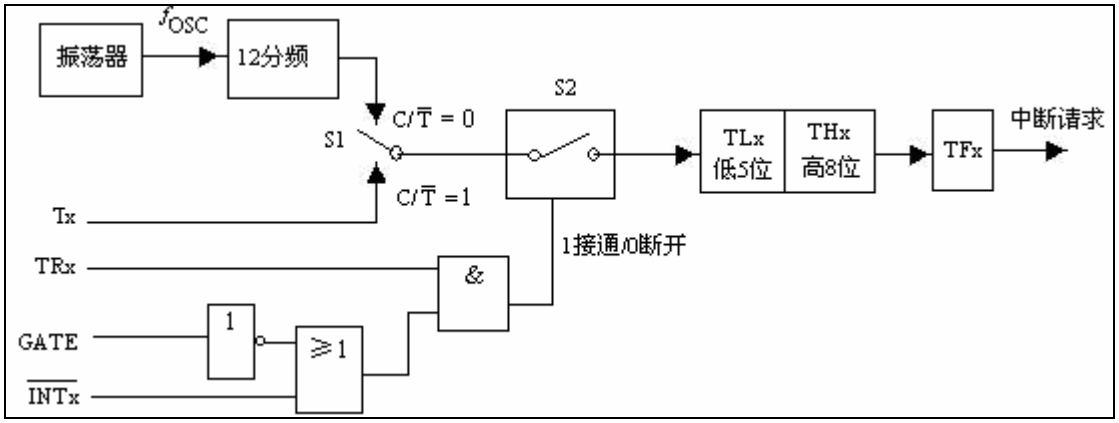
5.1.2.3　定时/计数器的工作方式

MCS-51 单片机的定时/计数器有 4 种工作方式，即方式 0、方式 1、方式 2、方式 3，主要用于定时和计数。但在方式 3 下 T1 通常作为串行异步通信口的波特率发生器。需注意的是，初始化时如果错将定时/计数器 T1 置为方式 3，则 T1 将停止工作。

1）工作方式 0

当 M1M0 初始化为 00（M1M0＝00）时，定时/计数器设定为工作力式 0，构成 13 位定时/计数器，即计数长度为 13 位。其逻辑结构如图 5.1-4 所示（图中 x 为 0 或 1，分别代表 T0 或 T1 的有关信号）。

图 5.1-4 定时/计数器工作于方式 0 的结构



THx 是高 8 位加法计数器，TLx 是低 5 位加法计数器，TLx 的高 3 位未用。TLx 加法计数器溢出时向 THx 进位，THx 加法计数器溢出时 TFx=1，最大计数值为 2^{13}。可用程序将 0~8191(2^{13}-1)的某一个数送入 THx、TLx 作为初值。THx、TLx 从初值开始计数，直到溢同。故初值不同，定时时间或计数值不同。需注意：加法计数器 THx 溢出后，必须用程序对 THx、TLx 设置初值，否则下一次 THx、TLx 将从零开始计数。

当 $C/\overline{T}=1$ 时，开关 S1 自动接在下面，定时/计数器工作在计数状态，13 位加法计数器对 Tx 引脚上的外部脉冲计数。计数值由下式确定

$$N = 2^{13} - x = 8192 - x$$

式中 N 为计数值，x 是 THx、TLx 的初值。x＝8191 时为最小计数值 1，x＝0 时为最大计数值 8192，即计数范因为 1~8192。

当 $C/\overline{T}=0$ 时，开关 S1 自动接在上面，定时/计数器工作在定时状态，加法计数器对机

器周期脉冲 T_{cy} 计数，每个机器周期 TLx 加 1。定时时间由下式确定

$$T=N\times T_{cy}=(8192-x)\,T_{cy}$$

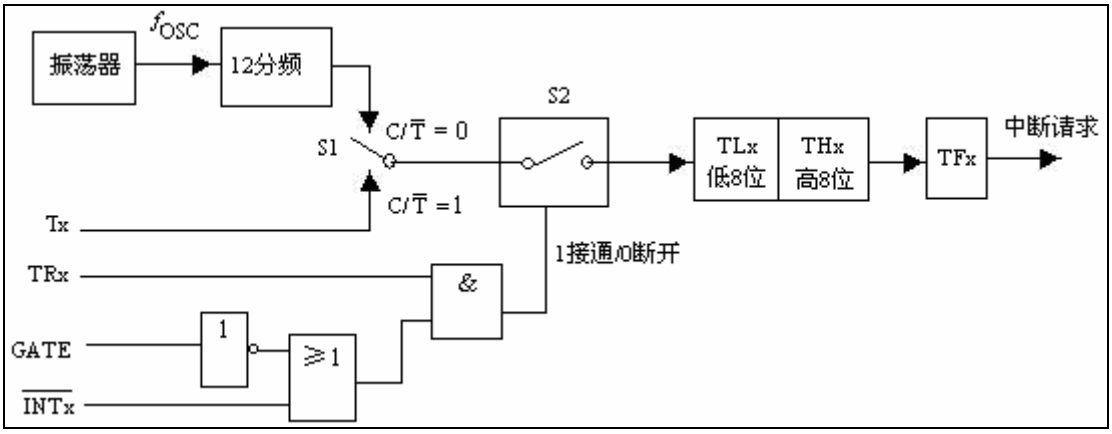
式中 T_{cy} 为单片机的机器周期，x 是 THx、TLx 的初值。如果振荡频率是 12MHz，则 T_{cy}＝1μs，定时范围为 1~8192μs。

定时/计数器的启动与停止由 TRx 控制。当 GATE＝ 0 时，只需用软件置 TRx＝ 1 ，开关 S2 闭合，定时/计数器开始工作；置 TRx＝0，S2 打开，定时/计数器停止工作。GATE＝1 为门控方式。此时，仅当 TRx＝ 1 且 INTx 引脚上出现高电平(即无外部中断请求信号)，S2 才闭合，定时/计数器开始工作。如果 INTx 引脚上出现低电平(即有外部中断请求信号)，则定时/计数器停止工作。所以，在门控方式下，定时/计数器的启动受到外部中断请求的影响，可用来测量 INTx 引脚上出现的正脉冲宽度。

2）工作方式 1

当 M1M0＝01 时，定时/计数器设定为工作力式 1，构成 16 位定时/计数器，即计数长度为 16 位，其结构如图 5.1-5 所示。此时，THx、TLx 都是 8 位加法计数器。其他与工作方式 0 相同。

图 5.1-5 定时/计数器工作于方式 1 的结构



在工作方式 1 下，计数器的计数值由下式确定

$$N=2^{16}-x=65536-x$$

计数范围为 1~65536。

定时器的定时时间由下式确定

$$T=N\times T_{cy}=(65536-x)\,T_{cy}$$

如果振荡频率是 12MHz，则 T_{cy}＝1μs，定时范围为 1~65536μs。

3）工作方式 2

当 M1M0＝10 时，定时/计数器设定为工作方式 2，构成具有自动重装初值功能的 8 位定时/计数器，其逻辑结构如图 5.1-6 所示。方式 2 的 16 位定时/计数器被拆成两个 8 位的寄存器 TLx 和 THx，CPU 在对它们初始化时必须装入相同的定时/计数器初值。

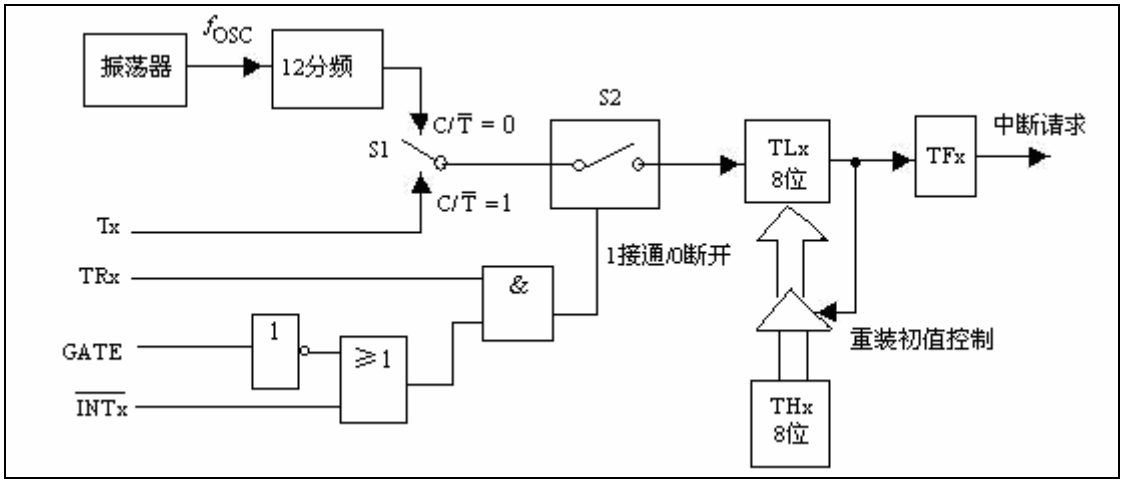
TLx 作为 8 位加法计数器使用，THx 作为初值寄存器用。TLx、THx 的初值都由软件设置。TLx 计数溢出时，不仅置位 TFx，而且发出重装载信号，使三态门打开，将 THx 中的初值自动送入 TLx，重新获得初值，并从初值开始重新计数。重装初值后，THx 的内容保持不变。如此反复，省去了程序需不断给计数器赋值的麻烦，而且计数准确度也提高了。

在工作方式 2 下，计数器的计数值由下式确定

$$N=2^{8}-x=256-x$$

计数范围为 1~256。

图 5.1-6 定时/计数器工作于方式 2 的结构



定时器的定时时间由下式确定

$$T=N\times T_{cy}=(256-x)\,T_{cy}$$

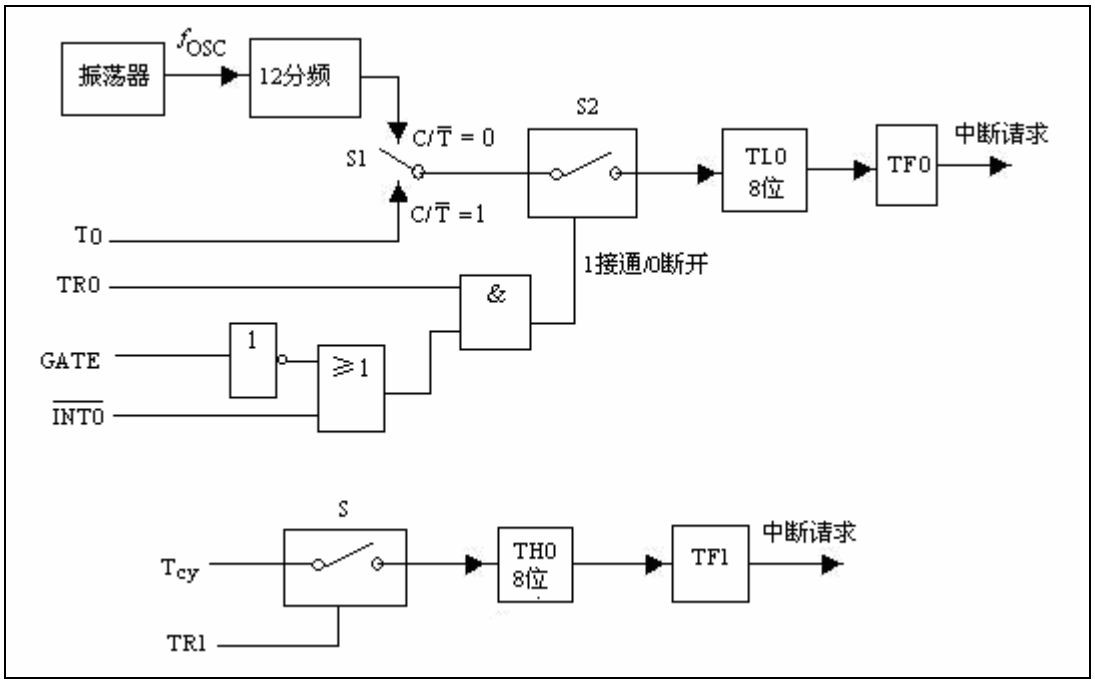
如果振荡频率是 12MHz，则 T_{cy}＝1μs，定时范围为 1~256μs。

定时/计数器工作方式 2 也有其局限性，即只有 8 位，计数值有限，最大计数值只有 256。所以这种工作方式一般适合在那些重复计数的应用场合。如可以通过这样的计数方式产生中断，从而产生一个固定频率的脉冲，也可当做串行数据通信的波特率发生器使用。

4）工作方式 3

当 M1M0＝11 时，定时/计数器设定为工作方式 3。在方式 3 时，定时/计数器 T0 和定时/计数器 T1 的工作方式不同。工作方式 3 下的定时/计数器 T0 的逻辑结构如图 5.1-7 所示。

图 5.1-7　定时/计数器 T0 工作于方式 3 的结构



在工作方式 3 时，定时/计数器 T0 被拆成两个独立的 8 位计数器 TL0、TH0。其中，TL0 既可作计数器用也可用定时器用，定时/计数器 T0 的各控制位和引脚信号全归它使用。当 TL0 溢出时，定时器 T0 溢出中断标志位 TF0 置 1。而 TH0 只能作简单的定时器使用。当 TH0 溢出时，定时器 T1 溢出中断标志位 TF1 置 1，而且还借用了定时/计数器 T1 的启动控

制位 TR1 作为 TH0 的启动控制位，TR1 负责控制 TH0 定时器的启动和停止。由于 TL0 既能作为计数器用也能作为定时器用，而 TH0 只能作为定时器用，故方式 3 下的定时/计数器 T0 可以构成两个定时器或者一个定时器一个计数器。

由于工作方式 3 下的定时/计数器 T0 占用了 T1 的启动控制位 TR1 和溢出中断标志位 TF1，使定时/计数器 T1 的功能受到限制。这种情况下，定时/计数器 T1 虽仍可用于方式 0、1、2，但不能使用中断方式。通常将 T1 作为串行口的波特率发生器使用，以确定串行通信的速率，由于 TF1 已被 T0 借用，所以只能把 T1 计数溢出直接送给串行口。当做波特率发生器使用时只需设置好工作方式，即可自动运行。如要停止工作，只需送入一个把它设置为方式 3 的方式控制字就可以了。由于定时/计数器 T1 本身不能工作于方式 3，当强行把它设置为方式 3，自然就会停止工作。

方式 3 下定时/计数器的定时、计数的范围和定时/计数值的确定同方式 2。

5.1.3 定时/计数器的编程及应用

5.1.3.1 定时/计数器的初始化步骤及初值的计算

1）定时/计数器的初始化步骤

定时/计数器在应用时，其工作方式和工作过程均可通过程序设定和控制，因此，定时/计数器在工作前必须对其进行初始化，计算和设置初值。定时/计数器的初始化顺序如下：

（1）根据任务要求，通过设置方式寄存器 TMOD 设置定时/计数器的工作方式。

（2）根据要求计算定时/计数器的初值 TC。

（3）初始化计数器，将初值 TC 送定时/计数器高、低位（即 TH、TL）。

（4）如果允许定时器溢出中断，则初始化定时/计数器的中断优先级（需要设置 IPH 及 PI 寄存器）；初始化中断控制寄存器 IE，能使相应定时/计数器中断，开中断。

（5）启动定时器。

2）定时/计数器的定时、计数范围

由于定时/计数器的定时、计数范围与计数器的模值 M 有关，而 M 的值又与定时/计数器的工作方式有关，故不同工作方式的定时、计数范围是不同的。

（1）工作方式 0：13 位定时/计数器方式，因此最多可以计到 2^{13}，即 1~8192 次。

（2）工作方式 1：16 位定时/计数器方式，因此最多可以计到 2^{16}，即 1~65536 次。

（3）工作方式 2 和工作方式 3：都是 8 位的定时/计数器方式，因此最多可以计到 2^{8}，即 1~256 次。

3）定时/计数器初值的计算

MCS-51 单片机定时/计数器采用增量式计数，即加法计数，其内部的计数器在定时器方式下对机器周期加“1”计数，在计数器方式下对外部引脚上的脉冲加“1”计数，计数器加满回零溢出时，置中断请求标志 TF。如果允许定时/计数器中断，则可进入中断服务程序，并作相应的操作。

在不同的工作方式下，定时/计数器初值的计算方法基本相同，只是采用了不同长度的计数器，设置时间常数时略有不同。

（1）计数器初值的计算

定时/计数器在计数器模式下工作时，必须给计数器预置初值，并通过程序送入 TH（TH0/TH1）和 TL（TL0/TL1）中。预置初值的计算方法是用最大计数量减去需要的计数次数，即

$$TC = M - C$$

其中，TC：计数器需要预置的初值；

M：计数器的模值（最大计数值），方式 0 时，$M=2^{13}$；方式 1 时，$M=2^{16}$，方式 2、3 时，$M=2^8$；

C：计数器计满回零所需的计数值，即设计任务要求的计数值。

（2）定时器初值的计算

定时/计数器在定时模式下，计数器的计数脉冲是由单片机系统主频经 12 分频后提供的。因此，定时器的定时时间的计算公式为

$$T=(M-TC)T_{cy}$$

或

$$TC=M-\frac{T}{T_{cy}}$$

其中，T：定时器的定时时间，即设计任务要求的定时时间。

T_{cy}：计数器计数脉冲的周期，即单片机系统主频周期的 12 倍。

M：计数器的模值，意义同上。TC：定时器需要预置的初值。

若 TC=0，则定时器定时时间为最大，由于 M 的值与定时器的工作方式有关，因此，不同的工作方式，定时器的最大定时时间 T_{max} 也不一样。若设单片机系统主频为 12MHz，则各种工作方式定时器的最大定时时间为：

工作方式 0：$T_{max}=2^{13}\times 1\mu s=8.192ms$

工作方式 1：$T_{max}=2^{16}\times 1\mu s=65.536ms$

工作方式 2、3：$T_{max}=2^8\times 1\mu s=0.256ms$

5.1.3.2 定时/计数器的溢出校准和实时读取

1）定时/计数器的溢出校准

当 MCS-51 内部定时/计数器的计数器回零溢出后，一方面置位 TF0/TF1 中断标志，申请中断；另一方面，方式 0、方式 1、方式 3 在未再次装入计数初值时，计数器会自动从 0 值重新开始计数。如果要求反复计数或定时，由于 CPU 响应中断和重新装入初值都需要时间，这样会给计数或定时带来误差。对于单级中断系统，一般中断延时可以忽略，但在时间上要求确实很严格的应用场合，必须精确计入这些延时。这时需要对定时器溢出校准。

若在单级中断系统中，CPU 响应中断至少用 3 个机器周期。若有其他指令延误，则延误时间不易计算，可采用以下程序段来校准。例如，T1 方式 1，定时 1ms 中断，$f_{OSC}=12MHz$，则定时初值 TC 为

TC=65536-1ms/1μs=65536-1000=0FC18H

即是[-1000]的补码。中断服务程序如下：

```
    ⋮
CLR   EA                          ；禁止所有的中断
CLR   TR1                         ；停止 T1 计数
MOV   A,# LOW（-1000+7）          ；取低 8 位校准码
ADD   A TL1                       ；校准码加 TL1 中的当前值
MOV   TL1,A                       ；要重装入的校准后的低 8 位初值
MOV   A,# HIGH（-1000+7）         ；取高 8 位校准码
ADDC   A,TH1                      ；校准码加 TH1 中的当前值
MOV   TH1, A                      ；要重装入的校准后的高 8 位初值
SETB   TR1                        ；启动 T1
    ⋮
```

这里考虑处理过程中该定时器停止了 7 个机器周期。关 CPU 中断的目的是禁止高级中断请求干扰这个时间上不能有差错的代码段的执行。

2）实时读取定时/计数器

在实际应用中，有时需要在不干扰定时器计时的条件下读取其当前值。MCS-51 定时/计数器的寄存器都可以在它运行时被读和写，但必须采取一些措施。一般可使用下面的子程序段：

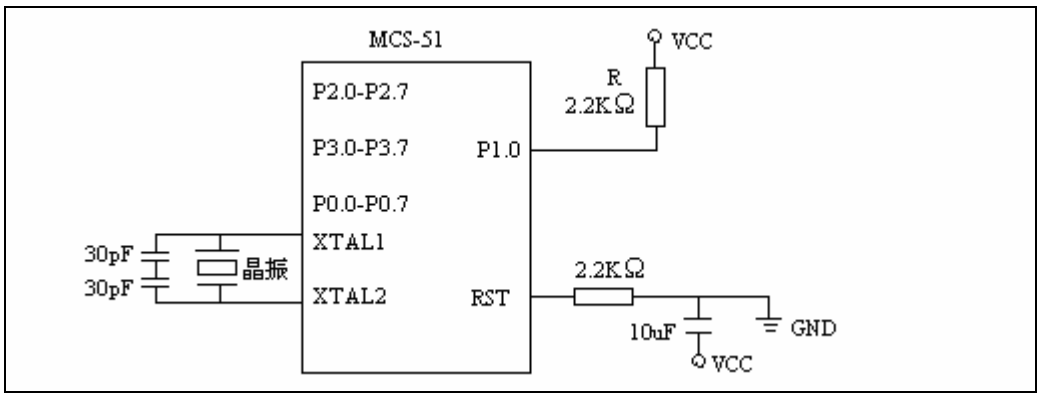
```
RDTIME：MOV   A，TH0
        MOV   R0，TL0
        CJNE  A，TH0，RDTIME
        MOV   R1，A
        RET
```

先读高字节，后读低字节。在读完低字节后，再进一步证实高字节有无改变。如果有改变则重复整个过程。

5.1.3.3 定时/计数器应用举例

例 5.1 设一只发光二极管 LED 和 51 单片机的 P1.0 脚相连。当 P1.0 脚是高电平时，LED 不亮；当 P1.0 脚是低电平时，LED 发亮。试编程用定时器来实现发光二极管 LED 的闪烁功能。已知单片机系统主频为 12MHz。假设控制 LED 每 60ms 闪烁一次，电路如图 5.1-8 所示。

图 5.1-8 例 5.1 图



解： ① 计算计数初值 TC。

采用定时器 T0 的工作方式 1，单片机系统主频为 12MHz，则单片机的机器周期 T_{cy}=单片机的时钟周期×12=1μs，则定时初值 TC 为

$$TC=M-T/T_{cy}=2^{16}-60ms/1\mu s=65536-60000=5536$$

对应的二进制数形式 TC=0001010110100000，高 8 位放入 TH1，即 TH1=00010101=15H；低 8 位放入 TL1，即 TL1=10100000=A0H。

② 程序设计（利用查询方式实现）

```
        ORG 0000H
        AJMP START                ；转入主程序
        ORG   0030H
START：MOV   SP，#60H             ；设置堆栈指针
        MOV   P1，#0FFH           ；关发光二极管 LED
```

```
        MOV   TMOD，#01H          ；定时/计数器 T0 工作于方式 1
        MOV   TH0，#15H           ；设置定时器初值
        MOV   HL0，#0A0H
        SETB  TR0                 ；启动定时/计数器 T0
LOOP：JBC   TF0，NEXT             ；如果 TF0=1，则清 TF0 并转 NEXT 处
        AJMP  LOOP                ；否则跳转到 LOOP 处运行
NEXT：CPL   P1.0                  ；若 LED 原来灭，P1.0 取反后为亮
                                  ；若 LED 原来灭，P1.0 取反后为亮
        MOV   TH0，#15H           ；重置定时/计数器 T0 的初值
        MOV   TL0，0A0H
        AJMP  LOOP                ；返回等待
        END
```

例 5.2 设单片机晶振频率 f_{OSC}＝6MHz，用定时器 T1 以方式 0、查询方式产生周期为 500μs 的等宽方波脉冲，由 P1.0 口输出。

解：①计算计数初值 TC。

本题可在 P1.0 口以 250μs 交替输出高、低电平实现所需的脉冲，定时时间为 250μs。晶振频率 6MHz 对应的机器周期为 2μs。则初值 TC 为：

$$TC=M-T/T_{cy}=2^{13}-250/2=8067$$

对应的二进制数形式 TC=1111110000011，高 8 位放入 TH1，即 TH1=11111100=FCH；低 5 位放入 TL1，即 TL1=00011=03H。

② 寄存器初始化

包括定时器初始化和中断系统初始化，主要对 IP、IE、TCON、TMOD 的相应位进行正确的设置，并将时间常数送入定时器。本例中，IE、IP、TCON、TMOD 均应初始化为 00H。

③ 程序设计

本例假设系统是从复位开始运行，则 IE、TCON、TMOD 均不需要操作。

```
    MOV    TH1，#0FCH          ；T1 置初值
    MOV    TL1，#03H
    SETB   TR1                 ；启动 T1
LOOP：JBC   TF1，LOOP1         ；T1 溢出转 LOOP1
    SJMP   LOOP                ；T1 未溢出，继续查询
LOOP1：MOV  TH1，#0FCH         ；T1 重新置初值
     MOV   TL1，#03H
     CLR   TF1                 ；清 T1 溢出标志位
     CPL    P1.0               ；输出取反
     SJMP  LOOP                ；继续查询
```

例 5.3 试对例 5.2 改以方式 1 实现。

解：①计算计数初值 TC。

$$TC=M-T/T_{cy}=2^{16}-250/2=65411$$

对应的二进制数形式 TC=1111111110000011，高 8 位放入 TH1，即 TH1=11111111=FFH；低 8 位放入 TL1，即 TL1=10000011=83H。

② 程序设计

与例 5.2 程序对应，仅 T1 置初值的程序不同，将对应的程序修改即可。

```
        MOV TH1，#0FFH          ；T1 置初值
        MOV TL1，#83H
```

例 5.5　试用定时/计数器测量低频脉冲信号频率及脉冲宽度。

解：由于被测信号频率低，可令定时/计数器 T0 处于定时方式，测出信号相邻两下降沿间的时间，即可知道被测信号的周期，再利用例 5.4 中的方法，测出信号宽度即可。被测信号从 P3.2 引脚输入。

程序设计如下：

```
        ORG   0000H
        AJMP  START             ；转入主程序
        ORG   100H
START：MOV   SP，#5FH           ；初始化堆栈指针 SP
        ；初始化定时器 T0（工作方式 0）
        MOV   TL0，#00H         ；计数器初值为 0
        MOV   TH0，#00H
        MOV   TMOD，#00000001B  ；设置定时器 T0 工作方式 1
        ；定时器 T0 在定时方式，C/T=0，M1M0=01，即方式 1
        ；由 TR0 控制计数器开关，GATE 位为 0
        ；初始化 INT0 中断
        SETB  IT0               ；外中断 INT0 定义为下降触发方式
        JNB   IE0，$            ；等待 P3.2 引脚出现下降沿
        SETB  TR0               ；开计数器
        CLR   IE0               ；清除 INT0 中断标志位
        JNB   IE0 ，$           ；再次等待 INT0 中断
        CLR   TR0               ；停止 TR0 计数
        ；至此已经测出了被测信号的周期
        MOV   R2，TL0           ；保存结果
        MOV   R3，TH0
        ；测被测信号的宽度
        MOV   TL0，#00H         ；计数器初值为 0
        MOV   TH0，#00H
        ANL   TMOD，#0F0H       ；与 0F0H 相与，使低 4 位清零，高 4 位保持不变
        ORL   TMOD，#00001001B  ；由 INT0 和 TR0 共同控制计数器开和关，GATE 位为 1
                                ；定时方式，即C/T位为 0，M1M0 为 01，即方式 1
WAITL：JB   P3.2，WAITL         ；等待 P.2 引脚为低电平，即等待下降沿
        SETB  TR0               ；在下降沿开启计数器
WAITH：JNB P3.2，WAITH          ；等待 P.2 引脚为高电平
WAITHL：JB P3.2，WAITHL         ；等待 P.2 引脚出现正脉冲下降沿
        CLR   TR0               ；在第二个下降沿关闭计数器 T0
```

5.2　MCS-51 单片机的串行通信

随着单片机的发展，单片机应用已从单机通信转向多机通信或联网通信，需要实现多机之间的数据交换功能。本节将介绍单片机串行通信基本知识、单片机串行口的结构、特点、工作方式，并简单介绍单片机双机、多机、单片机与 PC 机之间的通信技术。

5.2.1 串行通信概述

5.2.1.1 基本通信方式及分类

单片机与外界进行信息交换的过程统称为通信。

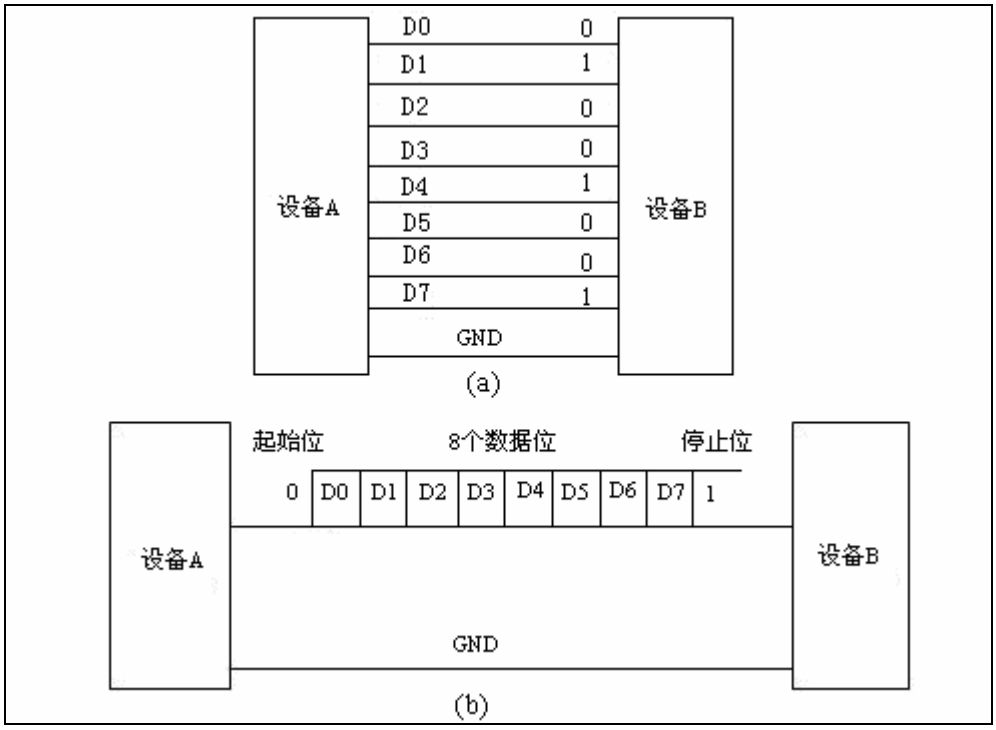
1）两种基本通信方式

根据 CPU 与外设之间连线结构和数据传送方式的不同，可将通信分为并行通信和串行通信两种基本方式。

并行通信是将数据以成组的方式在多条并行通道中传输，如图 5.2-1(a)所示。

图 5.2-1 两种基本通信方式

(a)并行通信　(b)串行通信



并行通信可同时传输一组数据位，每个数据位使用单独的一条导线。并行通信的特点是硬件上有多根数据线，各数据位同时传送，速度快，效率高，但成本高，需要的传输线多（例如对于 8 位数据传输来说，至少需要 8 根数据线以及一些其他的控制信号线等，干扰大，可靠性差，一般只适合于短距离数据传输，如计算机和外围设备之间的通信，CPU、存储器模块和设备控制器之间的通信等。

串行通信中各数据位一位接一位按顺序传送，如图 5-9(b)所示。串行通信的特点是所需传输线少，只要一对传输线即可实现通信，成本低，但传送速度慢。假如传送 8 位二进制数据所需时间为 T，在发送速率相同的条件下，串行传送则至少需要 8T。且在实际的串行通信系统中，还需要在数据位前、后分别插入起始位和停止位，以保证数据可靠地接收，故实际传输时间大于 8T。串行通信适合于远距离传输。

2）串行通信的分类

串行通信按照数据传送方式不同可分为同步通信和异步通信两种。

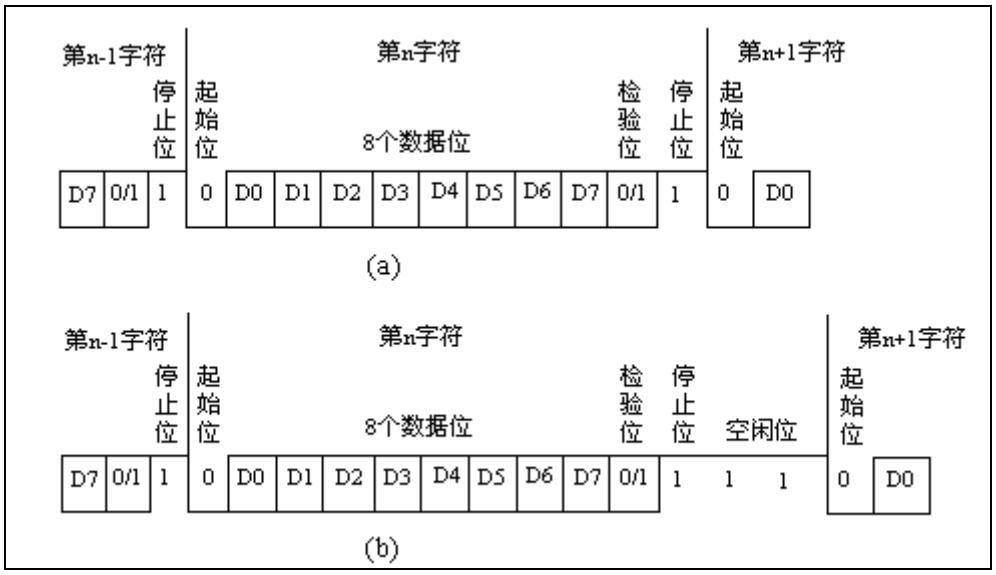
（1）异步通信

异步通信中，每个设备都有自己的时钟信号，通信中这些时钟频率必须保持一致，当传输一个字节时，通常会有一个起始位来同步时钟。当接收端接收到停止位时，将定时器复位，

准备接收下面的数据。异步通信用一帧来表示一个字符，在一帧格式中，每个字符由起始位（低电平）、数据位、奇偶检验位、停止位（高电平）组成，其典型的异步通信数据帧格式如图 5.2-2 所示。

图 5.2-2 异步通信数据帧格式

(a)无空闲位字符帧　　(b)有空闲位字符帧



起始位：通信线在没有数据传输时处于逻辑“1”状态，即呈现高电平，称其为空态。当发送器要发送一个字节的数据时，首先发送一个逻辑“0”信号，表示其后所传输的为数据。因此，起始位表示异步传输开始。

数据位：起始位之后为数据位。数据位的个数是 5、6、7 或 8 位，低位在前，高位在后

奇偶检验位：奇偶检验位用于有限差错检测，是通信双方一致约定的检错方式。奇偶检验位为冗余位，可由用户根据需要选择使用。

停止位：在奇偶检验位或数据位(当无奇偶检验时)之后是停止位。可以是 1 位、2 位，表示一个字符传输的结束。在发送的间隙，即空闲时，通信线路总处于逻辑“1”状态。

空闲位：空闲位在停止位之后，可以是多位。表示一帧数据传送完毕下一帧数据还没有到来，下一帧数据到来时则空闲位结束。

异步通信中，CPU 与外设之间必须有两项规定，即字符帧格式和波特率。字符帧格式的规定是双方能够对同一种 0 和 1 的串的理解达成一致。原则上字符格式或以由通信双方自由制定，但从通用、方便的角度出发，一般还是使用统一标准。

（2）同步通信

同步通信中，所有设备都使用同一个时钟，是以数据块方式传输，每个数据块通过同步字符使收/发双方同步。这里的数据块也称为帧，但与异步通信中的帧格式不同，它通常含有若干个数据字符。同步通信中的数据帧由同步字符、数据字符、检验字符 CRC（Cyclic Redundancy Check，循环冗余检验）等三部分组成，如图 5.2-3 所示。

图 5.2-3 同步通信数据帧格式

同步字符位于帧结构开头，用于确认数据字符的开始；接收时，接收端不断对传输线采

样，并把采样到的字符与双方约定的同步字符比较，只有比较成功后才会把后面接收到的字符加以存储。

数据字符在同步字符之后，个数不受限制，由所需传输的数据块长度决定。

检验字符有 1～2 个，位于帧结构末尾，用于接收端对接收到的数据字符的正确性检验。

同步通信中的同步字符可以采用统一标准格式，也可以由用户约定。在单同步字符帧结构中，同步字符常采用 ASCII 码中规定的 SYN 代码 16H，在双同步字符帧结构中，同步字符一般采用国际通用标准代码 EB90H。

与异步通信相比，同步通信传输速度快、效率高，通常可达 56Mb/s，同步通信的缺点是要求发送时钟与接收时钟保持严格同步，故发送时钟除应和发送波特率保持一致外，还要求把它传送到接收端去。

5.2.1.2 串行数据传送方式及差错检测

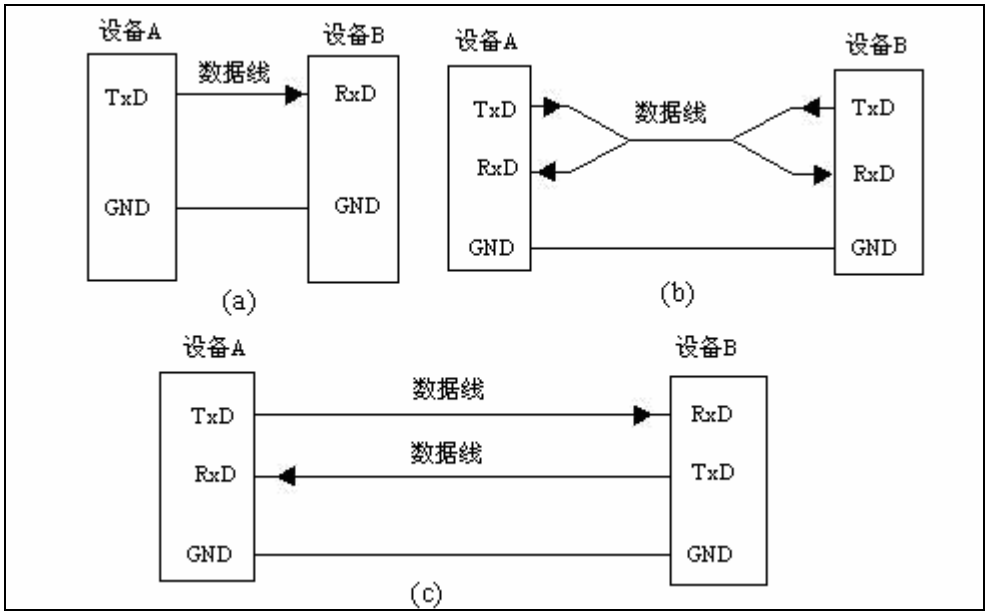
1）串行通信数据传送方式

串行通信的数据传送方式有单工、半双工、全双工以及多工方式。

单工方式：两串行通信设备 A、B 之间的数据传送仅按一个方向传送，一个发送，另一个接收，即数据只能由发送设备单向传输到接收设备，如图 5-11(a)所示。单工方式用途有限，常用于串行口的打印数据传输与简单系统间的数据采集。

图 5.2-4 数据传输方式

(a) 单工方式　　(b)半双工方式　　(c)全双工方式



注：TxD：发送数据端　RxD：接收数据端　GND：地线

半双工方式：两串行通信设备 A、B 之间的数据可双向传送，但不能同时进行，如 A 发送数据时就不能接收数据，A 接收数据时就不能发送数据，如图 5-11(b)所示。实际的应用采用某种协议实现收/发开关转换。

全双工方式：通信双方的数据可双向传送，且可同时进行。由于允许同时发送和接收，就需要两根数据线，A 设备的发送端接 B 设备的接收端，A 的接收端接 B 的发送端，如图 5.2-4(c)所示。

以上三种传输方式都是用同一线路传输一种频率信号，为了充分利用线路资源，可通过使用多路复用器或多路集线器，采用频分、时分或码分复用技术，可实现同一线路上资源共享功能，称为多工传输方式。

2）通信数据的差错检测和校正

通信的关键不仅是能够传输数据，更重要的是能准确传送、检错并纠错。检错有三种基本方法，即奇偶校验、校验和、循环冗余码校验。纠错方法主要有两种，即海明码校验、交叉奇偶校验。

（1）海明码校验

在所有字符上附加冗余码，即可检出并纠单错。对一个字节包含 8 位的情况，需要在字节上附加 $\log_2 8+1$ 位用作海明位。这意味着对于 8 位数据来说，需要用 12 位字。4 个附加位就是奇偶位，分别属于 8 个原始位的不同分组。

b_0 b_3 b_6 → h_2

b_1 b_4 b_3 → h_2

b_2 b_5 → ：行奇偶校验

↓ ↓

b_0 h_1 ↓ ：列奇偶校验

$b_0b_1b_2b_3b_4b_5b_6b_7$=字节

$h_0h_1h_2h_3$=海明位

如果纠错电路发现所产生的 h_0 位与读出的 h_0 位不同，而其他的都正确，则表明 b_2 反了；如果 h_1 和 h_3 是错的，则表明 b_6 反了。所以可以纠单错，检双错。

（2）交叉奇偶校验

将海明码概念扩展到数据块，可以穿过一个字节块应用奇偶校验以及对每一个字节应用奇偶校验，以找出单错。分组如下：

	h_0	b_0	h_0	…	b_0	10
	b_1	b_2	h_1	…	b_1	11
	⋮	⋮	⋮		⋮	⋮
	b_7	b_7	b_7	…	b_7	17
	P_0	P_1	P_2	…	P_n	18

9 位

穿过块的奇偶校验（n 个字节的同一位的奇偶绞验位）

数据块

沿块向下的奇偶校验（每个字节自身的奇偶校验位）

如果位 11 和 P1 是错的，则表明第 2 字节的 b1 反了。

（3）奇偶校验

这种校验方法是，发送时在每一个字符的最高位之后都附加一个奇偶校验位。这个校验位可为“1”或“0”，以保证整个字符(包括校验位)为“1”的位数为偶数(偶校验)或为奇数(奇校验)。接收时，按照发送方所规定的同样的奇偶性，对接收到的每一个字符进行校验．若二者不一致，便说明出现了差错。

奇偶校验是一个字符校验一次，是针对单个字符进行的校验。奇偶校验只能提供最低级的错误检测，尤其是能检测到那种影响了奇数个位的错误，通常只用在异步通信中。

（4）校验和

校验和方法是针对数据块进行的校验方法。在数据发送时，发送方对块中数据简单求和，产生一单字节校验字符(校验和)附加到数据块结尾。接收方对接收到的数据算术求和后，所得的结果与接收到的校验和比较，如果两者不同，即表示接收有错。

需要指出，校验和不能检测出排序错。也就是说，即使数据块是随机、无序地发送，产生的校验和仍然相同。

（5）循环冗余码 CRC

CRC 校验是一个数据块校验一次，同步串行通信中几乎都使用 CRC 校验，例如对磁盘的读/写等。

3）波特率

串行通信系统中常用波特率来衡量通信的快慢程度。数字通信所传输的是一个接一个按节拍传送的数字信号单元，即码元。波特率定义为每秒钟传输的码元个数，单位是波特(Baud)，简记为 B。而每秒钟传送的二进制数码的位数则定义为比特率，单位是比特每秒，即 b/s 或 bps。在串行通信系统中，传送的信号可能是二进制、八进制、十进制等，只有在二进制通信系统中波特率和比特率在数值上才是相等的。本教材所描述的串行通信传输信号都是采用二进制信号传送的，故比特率和波特率相等。

例如，通信双方每秒钟所传送的信息量是 480 字节，每一字节包含 10 位（1 个起始位、8 个数据位、1 个停止位），则波特率为

$$480 \times 10 = 4800\text{b/s} = 4800\text{B}$$

波特率的倒数即为每位传输所需的时间。相互通信的甲乙双方必须具有相同的波特率，否则无法成功地完成串行数据通信。

4）串行通信接口种类

根据串行通信格式及约定（如同步方式、通信速率、信号电平等）的不同，派生出不同的串行通信接口标准，如常见的 RS-232C、RS-422A、RS-485、IEEE1394、I^2C 总线、SPI（同步通信）总线、USB（通用串行总线接口）、CAN 总线接口、UART 接口等。本教材主要介绍 MCS-51 单片机内置的通用异步收发器 UART 接口及使用规则。

5.2.2 51 单片机的串行口及工作方式

串行数据通信主要有两个技术问题。一个是数据传送，另一个则是数据转换。数据传送主要解决传送中的标难、格式及工作方式等问题。而数据转换则一般是数据的串并转换。因为在计算机中使用的数据都是并行数据，因此在发送端，要把并行数据转换为串行数据；而在接收端，却要把串行数据转换为并行数据。为了实现数据的串并转换，可以采用软件或硬件方法实现。

软件实现如：假设要求传送的字符帧为 1 位起始位、7 位数据位、1 位奇偶校验位和 2 位停止位，长度为 11 位。可以编程控制使 P1.0 输出串行字符帧，先输出一位低电平作起始位；然后，利用移位指令把要发送的并行数据转换为串行数据依次从 P1.0 口输出；在发送完数据位后，输出一位奇偶校验位，其后，输出停止位。它的位时间可以用软件延时程序控制。由于实现串并转换的程序和电路都比较简单，此处略。

硬件实现串并转换，通常采用 MCS-51 单片机内置的一个全双工串行口 UART（Universal Asynchronous Receiver/Transmitter）部件。下面着重介绍单片机串行口 UART 的结构、特点、工作方式及简单应用。

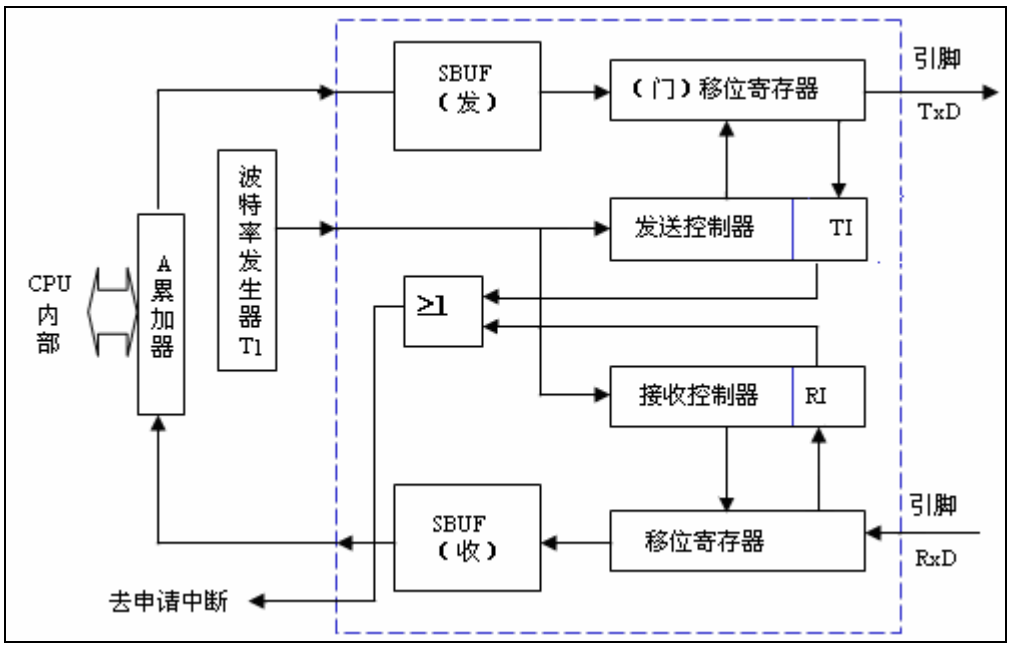
1）MCS-51 单片机的串行口

MCS-51 单片机内置了一个可编程的全双工串行通信口部件 UART，它主要由串行接收缓冲器 SBUF 和串行发送缓冲器 SBUF、输入移位寄存器、接收控制器、发送控制器和门电路等部分组成，其结构如图 5.2-5 所示。串行数据从 TxD(P3.1)引脚输出，从 RxD(P3.0)引脚输入。

串行通信接口 UART 的发送、接收缓冲器使用同一特殊功能寄存器名 SBUF（字节地址都是 99H），其发送和接收数据是彼此独立的，可同时进行。发送缓冲器只能写入数据不可以读出数据，接收缓冲器只可以读出数据不可以写入数据，用读、写指令加以区分。如：

```
MOV   SBUF，A          ；  启动一次数据发送
MOV   A，SBUF          ；  完成一次数据接收
```

图 5.2-5 MCS-51 串行通信接口内部结构



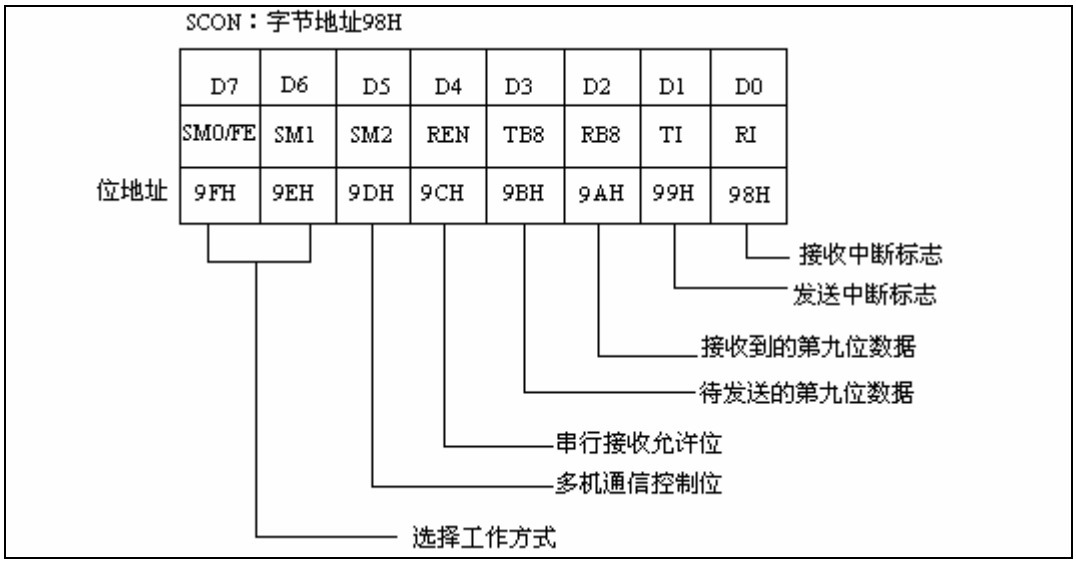
由于串行口接收部分由输入移位寄存器和接收缓冲器构成双缓冲结构，所以在接收缓冲器读出数据之前，串行口可以开始接收第二个字节。但是如果第二个字节已接收完毕时，第一个字节还没有读出，则将丢失其中一个字节。

MCS-51 单片机串行口除了用于数据通信外，还可以通过外接移位寄存器非常方便地构成一个或多个并行 I/O 口，或实现串并转换功能用来驱动键盘或显示器。在 MCS-51 中，与串行通信控制有关的寄存器为串行通信接口控制寄存器 SCON（选择串行通信接口工作方式）和电源控制寄存器 PCON 的 SMOD1 位（发送、接收波特率倍增控制位）。

2）串行口控制寄存器 SCON

SCON 用于定义串行口工作方式和实施接收/发送控制，字节地址为 98H，可按位寻址，位地址从 98H 到 9FH，SCON 的格式及各位的含义如图 5.2-6 所示。

图 5.2-6 SCON 的格式及含义



SCON 寄存器各位理解如下：

（1）SM0SM1：串行口工作方式控制位。其状态组合所对应的工作方式如表 5.2-1 所示。

表 5.2-1 串行通信接口工作方式

SM0 SM1	工作方式	功能说明
00	方式0 （扩展I/O口方式）	同步移位寄存器输入/输出方式（仅用于扩展I/O引脚，不能用于串行通信），此时SM2必须为0；波特率固定为$f_{OSC}/12$
01	方式1 （常用）	8位UART（即8位异步串行通信方式）；当SM2=1时，接收条件（RI=1）是必须收到停止位；波特率可变，由定时器控制
10	方式2 （不常用）	9位UART（即9位异步串行通信方式）；当SM2=1时，接收条件（RI=1）是第9位（RB8）数据必须为1；波特率固定为 $f_{OSC}/64$或 $f_{OSC}/32$
11	方式3 （常用）	9位UART（即9位异步串行通信方式）；当SM2=1时，接收条件（RI=1）是第9位（RB8）数据必须为1；波特率可变，由定时器控制

注：表中 fosc 为单片机系统晶振频率。

串行口有 4 种工作方式，其中，方式 0 并不用于通信，而是通过外接移位寄存器芯片实现扩展 I/O 口的功能，该方式又称为移位寄存器方式。方式 1、2、3 都是异步通信方式。方式 1 是 8 位异步通信接口，一帧信息由 10 位组成，用于双机通信。方式 2 和 3 都是 9 位异步通信接口，其区别仅在于波特率不同。方式 2 和 3 主要用于多机通信，也可用于双机通信。

（2）SM2：为多机通信控制位，允许工作在方式 2 和方式 3 的单片机实现多机通信。

在工作方式 2 或方式 3 时，若 SM2＝1，当接收到的第 9 位数据(RB8)为 0 时，不启动接收中断标志 RI，即 RI＝0，并将接收到的前 8 位数据丢弃；当 RB8＝1 时，把接收到的前 8 位数据送入 SBUF，且置 RI＝1，发出中断申请，接收数据有效。当 SM2＝0，不管第 9 位是 0 还是 1，都将接收到的前 8 位数据送入 SBUF，并发出中断申请。在工作方式 1 时，若 SM2＝1，当接收有效停止位时，置 RI＝1，数据有效；没有接收到有效停止位时，RI＝0，数据无效。在工作方式 0 时，SM2 不用，应设置为 0。

（3）TI：发送中断标志位，用于指示一帧信息发送是否完成，可寻址标志位。在工作方式 0 时发送完第 8 位数据后由硬件置位 TI，在其他方式下，开始发送停止位时硬件置位 TI。TI 置位表示一帧信息发送完成，同时申请中断。TI 在发送数据前必须由软件清零。

（4）RI：接收中断标志位，用于指示一帧信息是否接收完成，可寻址标志位。在串行接收(不考虑 SM2)时，在方式 0 时接收完第 8 位数据后或在其他方式时，接收到停止位的中间时刻由硬件置位 RI，RI 置位表示一帧信息接收完毕，并发出中断申请。它也必须由软件清零。

可通过软件方式查询 TI 或 RI，也可通过中断方式判断发送、接收过程是否完成。

（5）REN：接收允许控制位，用于控制是否允许接收数据。REN=0 时，表示禁止接收数据；REN=1 时表示允许接收数据。该位的置位/清零由软件控制。

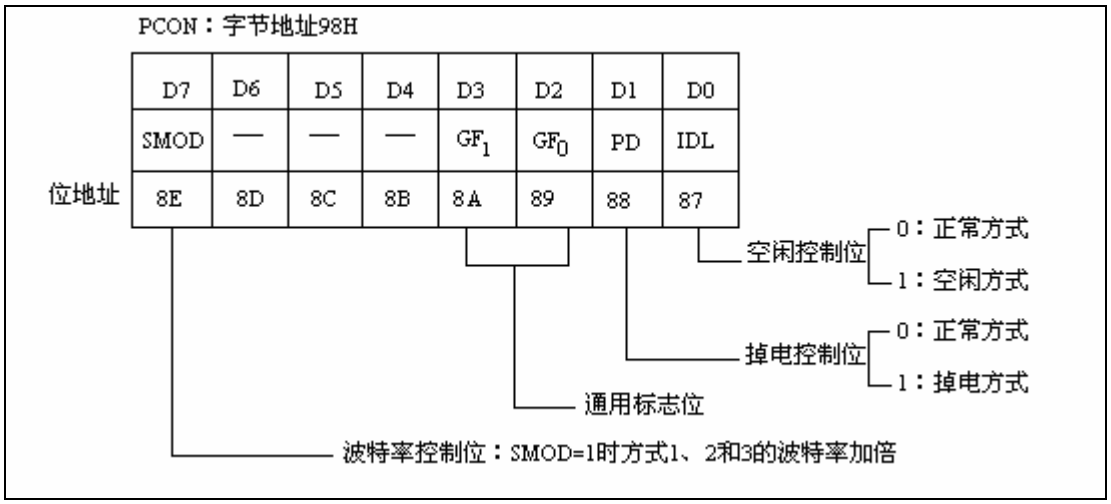
（6）TB8：在方式 2 和方式 2 中要发送的第 9 位数据位，可根据需要由软件置 1 或清 0。在双机通信中，TB8 一般作为奇偶检验位来使用。在多机通信中，TB8 代表传输的是地址还是数据，TB8=0，传输的是数据，TB8=1，传输的是地址。

（7）RB8：在方式 2 和方式 2 中要接收的第 9 位数据位，在方式 1 时，SM2＝0，RB8 接收的是停止位。在方式 0 时，RB8 不用。

3）电源控制寄存器 PCON

PCON 是为了在 CHMOS 型单片机上实现电源控制而设置的专用寄存器，字节地址为 87H，不可位寻址，PCON 的格式及各位含义如图 5.2-7 示。

图 5.2-7 PCON 的格式及含义



SMOD：波特率加倍位。在计算串行方式 1、2、3 的波特率时，SMOD＝0，波特率不加倍；SMOD＝1，波特率加倍。系统复位时默认为 SMOD＝0。PCON 中的其余各位与串行口无关，它们用于单片机的电源控制，已在前面讲过。

4）波特率设置

串行通信中，收发双方的波特率需有一定的约定，否则无法完成正常通信。

MCS-51 单片机串行口有 4 种工作方式，其中方式 0 和方式 2 的波特率是固定的，方式 1 和方式 3 的波特率是可变的，由定时器 T1 的溢出率（T1 溢出信号的频率）控制。

方式 0 的波特率是固定的，为系统晶振频率的 1/12，其值为 $f_{OSC}/12$。

方式 2 的波特率也是固定的，由 PCON 的选择位 SMOD 来决定，表示为：

$$\text{波特率}=(2^{SMOD}/64)\times f_{OSC}$$

由上式可知，当 SMOD=1 时，波特率为 $f_{OSC}/32$；当 SMOD=0 时，波特率为 $f_{OSC}/64$。

方式 1 和方式 3 的波特率由定时器 T1 的溢出率控制，是可变的，表示为：

$$\text{波特率}=(2^{SMOD}/32)\times \text{定时器T1溢出率}$$

$$\text{T1溢出率}=\text{T1计数率}/\text{产生溢出所需的周期数}=(f_{OSC}/12)/(2^{K}-TC)$$

式中，K 为定时器 T1 的位数；TC 为定时器 T1 的预置初值。T1 计数率取决于它工作在定时器状态还是计数器状态。当工作在定时器状态时 T1 计数率为 $f_{OSC}/12$；当工作在计数器状态时 T1 计数率为外部输入脉冲频率，此频率应小于 $f_{OSC}/24$。产生溢出所需周期与定时器 T1 的工作方式和 T1 的预置初值有关。

定时器 T1 工作在方式 0 时溢出所需周期为

$$2^{13}-TC=8192-TC$$

定时器 T1 工作在方式 1 时溢出所需周期为

$$2^{16}-TC=65535-TC$$

定时器 T1 工作在方式 2 时溢出所需周期为

$$2^{8}-TC=256-TC$$

串行口的波特率发生器就是利用定时器提供一个时间基准。定时器计数溢出后只需要做一件事情就是重新装入初值，再开始计数，而且中间不要任何延迟。因为 MCS-51 单片机定时/计数器的方式 2 就是自动重装初值的 8 位定时/计数器模式，所以用它用做波特率发生器

最适合不过了。当时钟频率选用 11.0592MHz 时，容易获得标准的波特率，所以很多单片机系统选用这个晶振频率。定时器 T1 在工作方式 2 时的常用波特率及初值如表 5-3 所示。

表 5.2-2 定时器 T1 在方式 2 时的常用波特率和初值

常用波特率/bps	f_{OSC} / MHz	SMOD	TH1 初值
19200	11.0592	1	FDH
9600	11.0592	0	FDH
4800	11.0592	0	FAH
2400	11.0592	0	F4H
1200	11.0592	0	E8H

通信波特率的选用，不仅和所选通信设备、传输距离、调制解调器（Moden）型号以及传输线状况有关，用户应根据需要正确选择。

5）串行口的工作方式

MCS-51 单片机的可编程全双工串行通信口部件 UART 有 4 种工作方式，分别为方式 0、方式 1、方式 2、方式 3。

（1）工作方式 0

串行口在工作方式 0 下为 8 位同步移位寄存器输入输出方式，用于通过外接移位寄存器扩展 I/O 接口，也可以外接同步输入输出设备。方式 0 下的波特率固定为 $f_{OSC}/12$。此时，串行口本身相当于“并入串出”（发送状态）或“串入并出”（接收状态）的移位寄存器。串行移位脉冲（也称同步移位脉冲）CLOCK 从 TxD（P3.1）引脚输出，频率为系统时钟频率 f_{OSC} 的 1/12，串行数据为 RxD 引脚输入或输出，发送、接收的每帧信息都是 8 位数据，低位在前、高位在后。串行口工作方式 0 的结构示意图如图 5.2-8 所示。

发送：

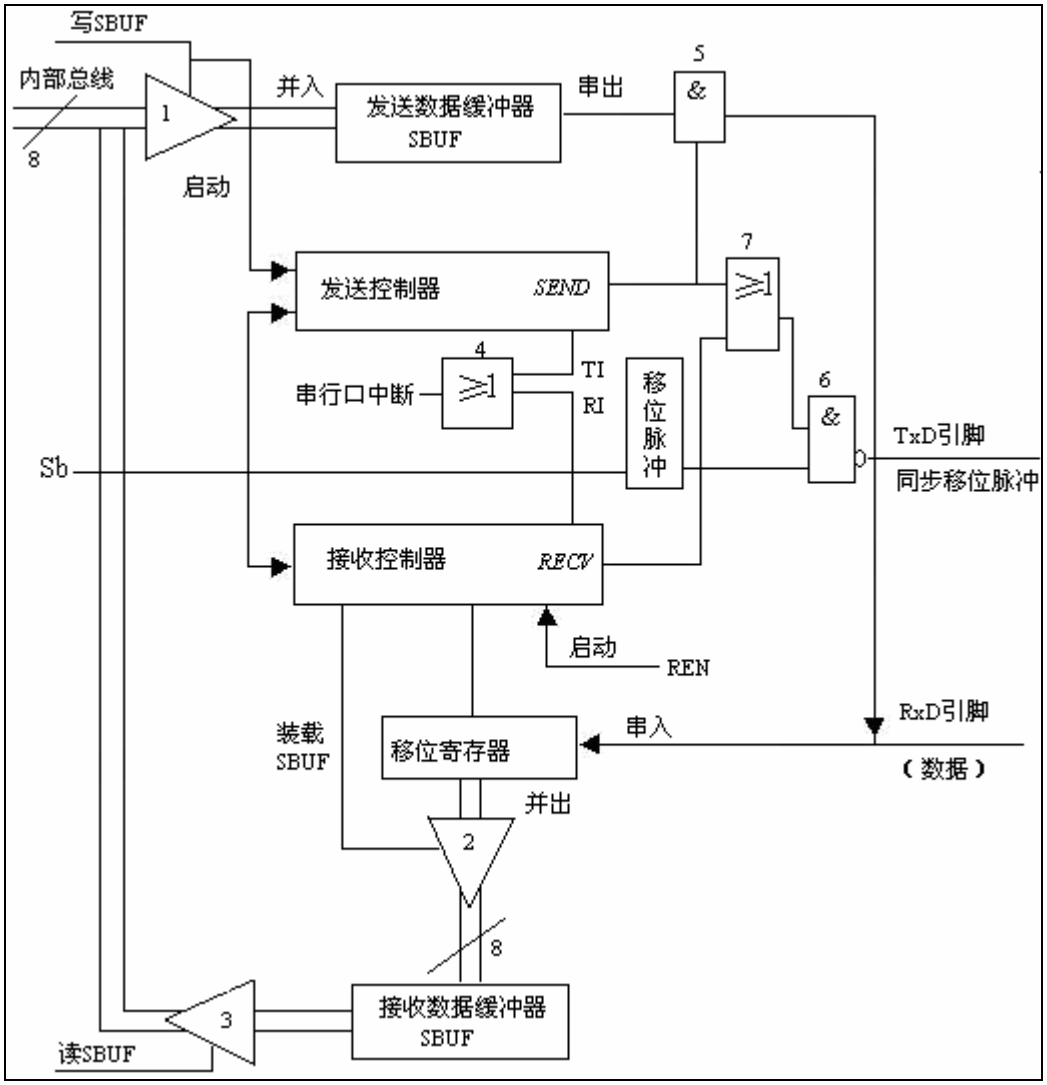
发送操作是在 TI=0 下进行的，此时发送缓冲寄存器“SBUF（发送）”相当于一个并入串出的移位寄存器。CPU 执行一条写 SBUF 的指令，如 MOV SBUF ，A，就启动了发送过程。指令执行期间送来的写信号打开三态门 1，将经内部总线送来的 8 位并行数据写入发送缓冲寄存器 SBUF，写信号的同时启动发送控制器。此后，CPU 与串行口并行工作，经过一个机器周期，发送控制端 SEND 有效（高电平），打开门 5 和门 6，允许 RxD 引脚发送数据，TxD 输出同步移位脉冲。在时钟信号 S6 触发产生的内部移位脉冲作用下，发送数据缓冲器中的数据逐步串行输出，每一个机器周期从 RxD 上发送一位数据。S6 同时形成同步移位脉冲，一个机器周期从 TxD 上输出。8 位数据（一帧）发送完毕后，SEND 恢复低电平状态，停止发送数据，且发送控制器硬件置发送中断标志 TI=1，向 CPU 申请中断。若中断开放，CPU 响应中断，在中断服务程序中，需用指令 CLR TI 先将 TI 清零，然后向发送缓冲寄存器“SBUF（发送）”送下一个欲发送的数据，以重复上述过程。

接收：

接收的初始化条件为 REN=1 和 RI=0，此时接收缓冲寄存器“SBUF（接收）”相当于一个串入并出的移位寄存器。此进，RxD 为串行数据接收端，TxD 依然输出同步移位脉冲。REN 置 1 将启动接收控制器。经过一个机器周期，接收控制端 RECV 有效（高电平），打开门 6，允许 TxD 输出同步移位脉冲。该脉冲控制外接芯片逐位输入数据，波特率为 $f_{OSC}/12$。在内部移位脉冲的作用下，RxD 上的串行数据逐位移入移位寄存器。当 8 位数据（一帧）全部移入移位寄存器后，接收控制器使 RECV 失效，停止输出移位脉冲，并发出“装载 SBUF”信号，打开三态门 2，将 8 位数据并行送入接收数据缓冲器 SBUF 中保存。与此同时，接收控制器硬件置接收中断标志 RI=1，向 CPU 申请中断。CPU 响应中断后，用软件使 RI=0，使移位寄存器开始接收一帧信息，然后通过读接收缓冲器的命令，例如 MOV A，SBUF ，

将“SBUF（接收）”接收到的数据读入累加器 A，读取 SBUF 中的数据。在执行这条指令时，CPU 发出的“读 SBUF”信号打开三态门 3，数据经内部总线进入 CPU。

图 5.2-8 串行口工作方式 0 结构示意图



利用串行口工作方式 0 可扩展 MCS-51 单片机芯片的输出和输入口。

在“串入并出”芯片（如 74LS164、 CD4094、74HC595 等）的配合下，扩展 MCS-51 单片机芯片的输出口。当使用芯片 74LS164 扩展时输出口时，MCS-51 芯片的 RxD 引脚接 74LS164 芯片的串行数据输入端（1、2 脚），TxD 引脚接 74LS164 芯片的移位脉冲输入端 CLK（8 脚），如图 5.2-9 示。

在“并入串出”芯片（如 74LS165、CD4014、74HC597 等）的配合下，扩展 MCS-51 单片机芯片的输出口。当使用芯片 74LS165 扩展时输入口时，MCS-51 芯片的 RxD 引脚接 74LS165 芯片的串行数据输出端（9 脚），TxD 引脚接 74LS165 芯片的移位脉冲输入端 CLK（2 脚），如图 5.2-10 所示。

串行口在方式 0 下的工作并非是一种同步通信方式，它的主要用途是与外部同步移位寄存器连接，以达到扩展一个并行口的目的。

在方式 0 中，串行口控制寄存器 SCON 内的 TB8、RB8 两位没有定义，可设为 0，但 SM2 必须为 0，电源控制寄存器 PCON 中的波特率倍增位 SMOD 也没有定义，可以是任意值。由于方式 0 不能自动插入及检测起始位、停止位，因此工作方式 0 下的串行口不能作为

串行通信口，只能用于扩展 I/O 口。

图 5.2-9 串行口方式 0 扩展输出口电路

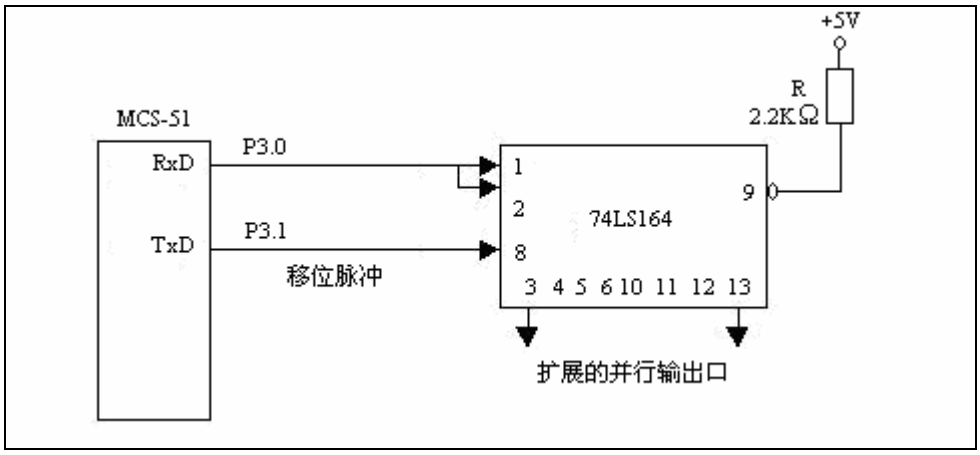
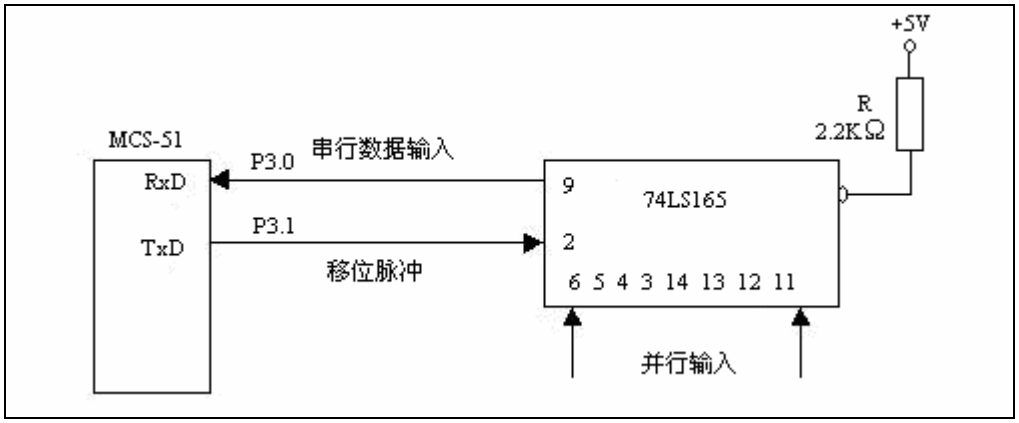


图 5.2-10 串行口方式 0 扩展输入口电路



（2）工作方式 1

当 SM0 SM1=01 时，串行口工作在方式 1。方式 1 为 8 位异步串行通信接口方式，RxD 为接收端，TxD 为发送端。发送或接收一帧信息由起始位（固定为 0）、8 位串行数据位（低位在前、高位在后）和停止位（固定为 1）共 10 位组成。方式 1 的波特率可变，由定时/计数器 Tl 的溢出率以及 SMOD(PCON.7)决定，且发送波特率与接收波特率可以不同。串行口工作方式 1 的结构示意图如图 5.2-11 所示。

发送：

CPU 执行一条写 SBUF 指令并启动了串行口发送，数据从 TxD 输出。在指令执行期间，CPU 送来“写 SBUF”信号，将并行数据送入 SBUF（发送），并启动发送控制器，此时发送控制器的 $\overline{SEND}$、DATA 有效。即串行口自动在 8 个串行数据位的前、后分别插入 1 位起始位（0）和 1 位停止位（1），构成 10 位数据帧，然后按设定的波特率依次从 TxD 上输出起始位、数据位、停止位。一帧信息发送完毕之后，发送控制端的 $\overline{SEND}$、DATA 失效，发送控制器硬件置发送中断标志 TI＝1，表示发送缓冲区内容已发送完毕，并向 CPU 申请中断。

接收：

接收过程是在 TI=0 且 REN=1 条件下启动，此时接收器开始工作。平时，接收电路跳变检测器对高电平的 RxD 进行采样（采样频率是移位脉冲的 16 倍）。当采样到从 1 到 0 的负跳变时，确认是开始位 0，就启动接收控制器接收数据，由于发送、接收双方各自使用自

己的时钟，两者的频率总有少许差异。为了避免这种差异的影响，控制器将位的传送时间分成 16 等份，位检测器在 7、8、9 三个状态，也就是在信号中央采样 RxD3 次。而且，3 次采样中至少 2 次相同的值被确认为数据，这是为了减少干扰的影响。如果接收到的起始位的值不是 0，则起始位无效，复位接收电路。如果起始位为 0，则开始接收本帧其他各位数据。控制器发出内部移位脉冲将 RxD 上的数据逐位移入移位寄存器，当 8 位数据及停止位全部移入后，根据以下状态，进行相应操作。

①若 RI=0、SM2＝0，则接收控制器发出“装载 SBUF”信号，将 8 位数据装入接收数据缓冲器 SBUF，停止位装入 RB8，并置 RI＝l，向 CPU 发出中断请求信号。

②若 RI=0、SM2＝1，则只有在停止位为 1 时才发生上述操作。

③若 RI=0、SM2＝1，且停止位为 0，则所接收的数据不装入 SBUF，即数据丢失。

④若 RI=1，则所接收的数据在任何情况下都不装入 SBUF，即数据丢失。

无论出现哪一种情况，跳变检测器将继续采样 RxD 引脚上的负跳变，以便接收下一帧信息。移位器采用移位寄存器和 SBUF 双缓冲结构，以避免在接收后一帧数据之前，CPU 尚未及时响应中断而将前一帧数据取走，造成两帧数据重叠。采用双缓冲结构后，前、后两帧数据进入 SBUF 的时间间隔至少有 10 个机器周期。在后一帧数据送入 SBUF 之前，CPU 有足够的时间将前一帧数据取走。

（3）工作方式 2 和工作方式 3

当 SM0 SM1=10 时，串行口工作在方式 2；当 SM0 SM1=11 时，串行口工作在方式 3。方式 2、方式 3 都是 9 位异步串行通信接口方式，其结构示意图如图 5.2-11 所示。

方式 2 的波特率固定为 MCS-51 系统主频 f_{OSC} 的 32 分频或者 64 分频，不可调；而方式 3 的波特率与定时器 T1 的溢出率、电源控制寄存器 PCON 的 SMOD 有关（即由定时器 T1 溢出信号的 32 分频，并由 SMOD 倍频得到），可调。选择不同的初值或晶振频率，即可获得不同的波特率，故方式 3 较常用。

发送或接收的一帧信息由 11 位组成。其中，1 位起始位、9 位数据位和 1 位停止位。方式 2、方式 3 发送、接收数据的过程，与方式 1 类似，所不同的是在于对第 9 位数据的处理上。发送时，第 9 位数据由 SCON 中的 TB8 位提供。接收数据时，当第 9 位数据移入移位寄存器后，将 8 位数据装入 SBUF，第 9 位数据装入 SCON 中的 RB8 位。

5.2.3 串行口的应用

1）方式 0 应用编程

串行口在方式 0 下主要作为 8 位同步寄存器使用，波特率为 $f_{OSC}/12$，数据从 P3.0 输入或输出，同步移位脉冲由 P3.1 输出，这种方式常用做扩展 I/O 口。

例 5.6 基于 CD4094 串入并出移位寄存器利用单片机串行口扩展 I/O 口，实现与 CD4094 并行输出连接的 8 个 LED 发光二极管循环点亮、熄灭。

解：串行口方式 0 的 I/O 口扩展电路如图 5.2-12 示。程序流程图如图 5.2-13 所示。

程序如下：

```
        ORG     0000H
        AJMP    main
        ORG     0100H
Main:   CLR     P1.0                ；CD4094 清零
        NOP                         ；空指令
        NOP                         ；空指令
        SETB    P1.0                ；撤除清零信号
        MOV     R3，#09H            ；置灯的循环次数
```

```
        MOV   A，#00H                ；A 清零
        SETB  CY                     ；CY 置 1
        MOV   SCON，#00H             ；设置串行口工作方式 0
XXT：   RRC   A                      ；A 带进位右移一位
        MOV   SBUF ，A               ；将 A 的内容送 SBUF 启动一次发射
        JNB   TI，$                  ；SBUF 的 8 位数据是否全送 CD4094，若没送完则等
                                       待
```

图 5.2-11 串行口工作方式 1、2、3 结构示意图

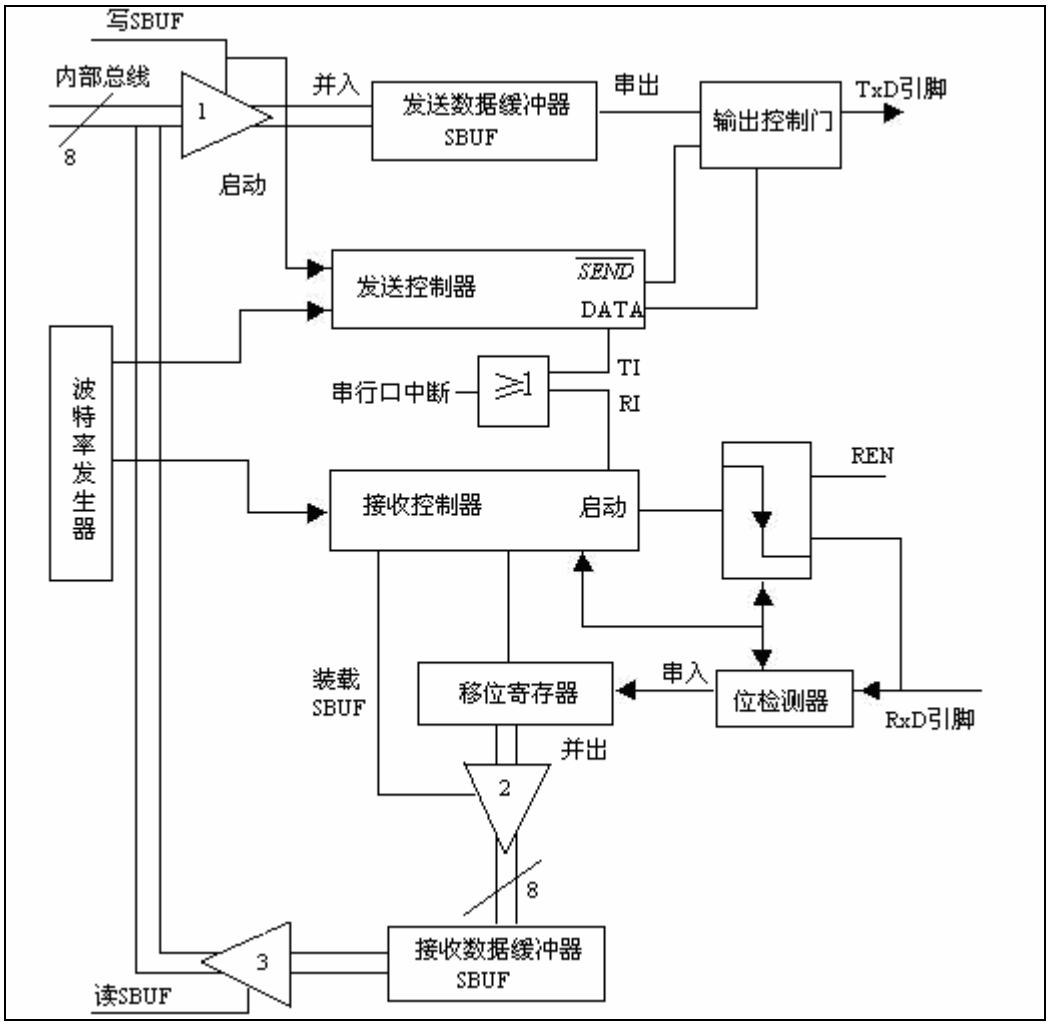


图 5.2-12 CD4094 驱动 LED 硬件连接图

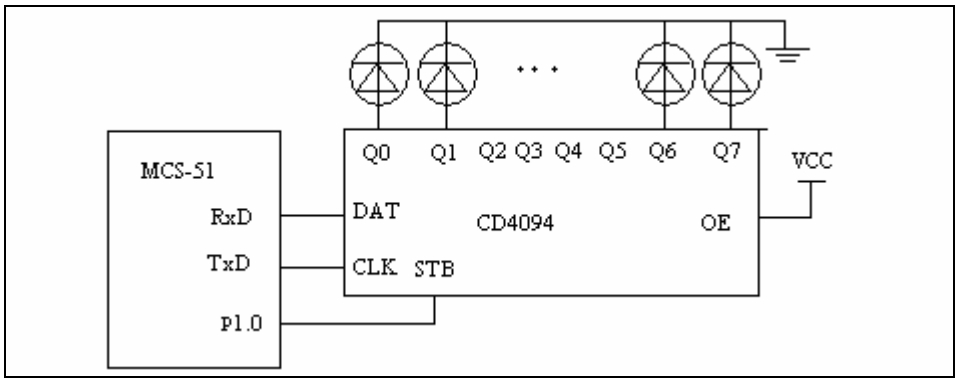
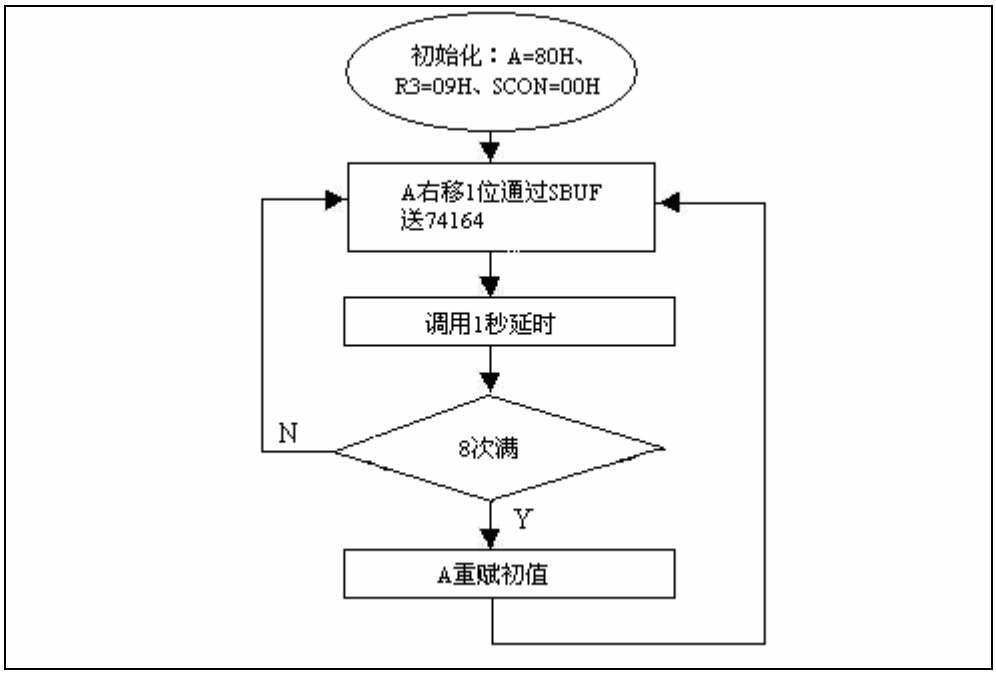


图 5.2-13 例 5.6 的程序流程图




```
        CLR   TI                    ；若发送完毕则清发送中断标志
        LCALL  delayls              ；调用 1 秒延迟程序
        DJNZ   R3，XXT              ；循环 8 次（顺序 8 位点亮）否？没有则转 XXT
        AJMP   main                 ；循环点亮 8 位后重新下一次大循环
Delayls：MOV   TMOD，#01H           ；利用 T0 产生 50MS 的定时循环 20 次
        MOV   R7，#20               ；循环 20 次数
LOOP1：MOV   TH0，#00H
        MOV   TL0，#4CH             ；计数初始值
        SETB   TR0                  ；开 T0 计数器
LOOP2：JNB   TF0，LOOP2             ；查询中断标志
        CLR   TF0                   ；清 T0 溢出标志位
        DJNZ   R7，LOOP1            ；控制循环
        RET                         ；子程序返回
        END                         ；结束
```

2）双机通信

单片机之间的通信，除了采用相同的波特率，通信双方还必须遵循同一协议，其中简单的通信协议可以自己设计，并按设计的协议编写通信程序。

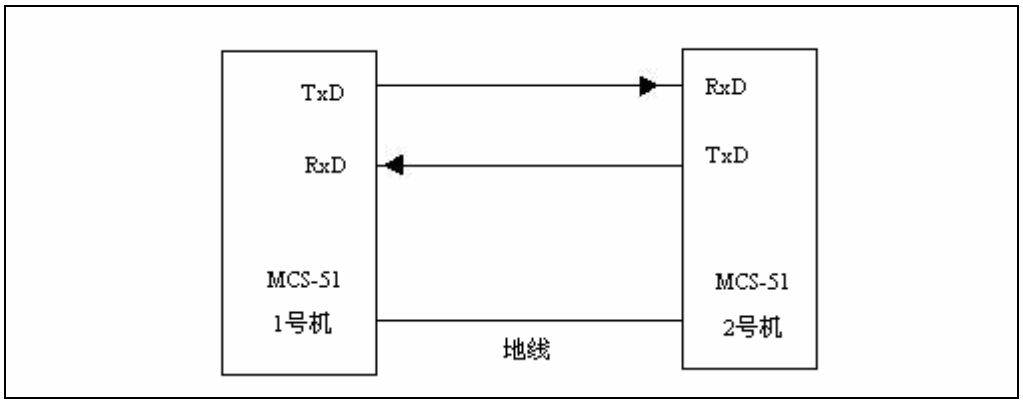
例 5.7 由 MCS-51 构成的双机通信系统如图 5.2-14 所示。#1 机将内部 RAM 的 30H~3FH 中的 16 个无符号随机数通过串行口发送到#2 机，#2 机将接收到的#1 机发送过来的数据存放在 RAM 的 30H~3FH 单元，要求采用累加校验。设单片机的晶振频率为 11.0592MHz，波特率为 9600b/s，采用串口方式 1，试编写程序。

解：现分析通信双方约定如下：

设 1 号机是发送方，2 号机是接收方，当 1 号机发送时，先发送一个“E1”联络信号，2 号机收到后回答一个“E2”应答信号，表示同意接收。当 1 号机收到应答信号“E2”后，开始发送数据，每发送一个字节数据都要计算“校验和”，假定数据块长度为 16 个字节，起始地址为 40H，一个数据块发送完毕后立即发送“校验和”。2 号机接收数据并转存到数据缓冲区，

起始地址为 40H，每接收到一个字节数据都要计算一次“校验和”，当收到一个数据块后再接收 1 号机发来的“校验和”，并将它与 2 号机求出的校验和进行比较。若两“校验和”相等，说

图 5.2-14 CS-51 构成的双机通信系统



明接收正确，2 号机回答 00H；若两者不相等，说明接收不正确，2 号机回答 0FFH，请求重发。1 号机接收到 00H 后结束发送，若收到 0FFH，则重新发送数据一次。双方约定采用串行口方式 1 进行通信，一帧信息由 1 个起始位、8 个数据位、1 个停止位共 10 位组成，波特率为 2400bps，T1 工作在定时器方式 2，单片机系统晶振频率选用 11.0592MHz，查表 5-3 可得，TH1=TL1=0F4H，PCON 寄存器的 SMOD 位为 0。程序流程图如图 5.2-15 示。

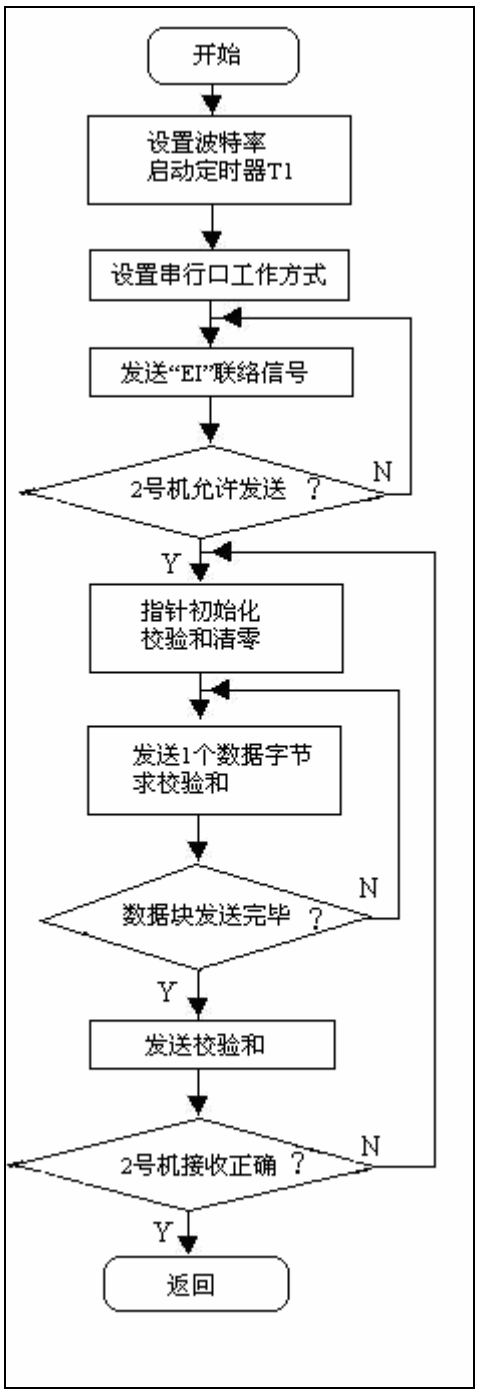
程序如下：

（1）发送程序清单

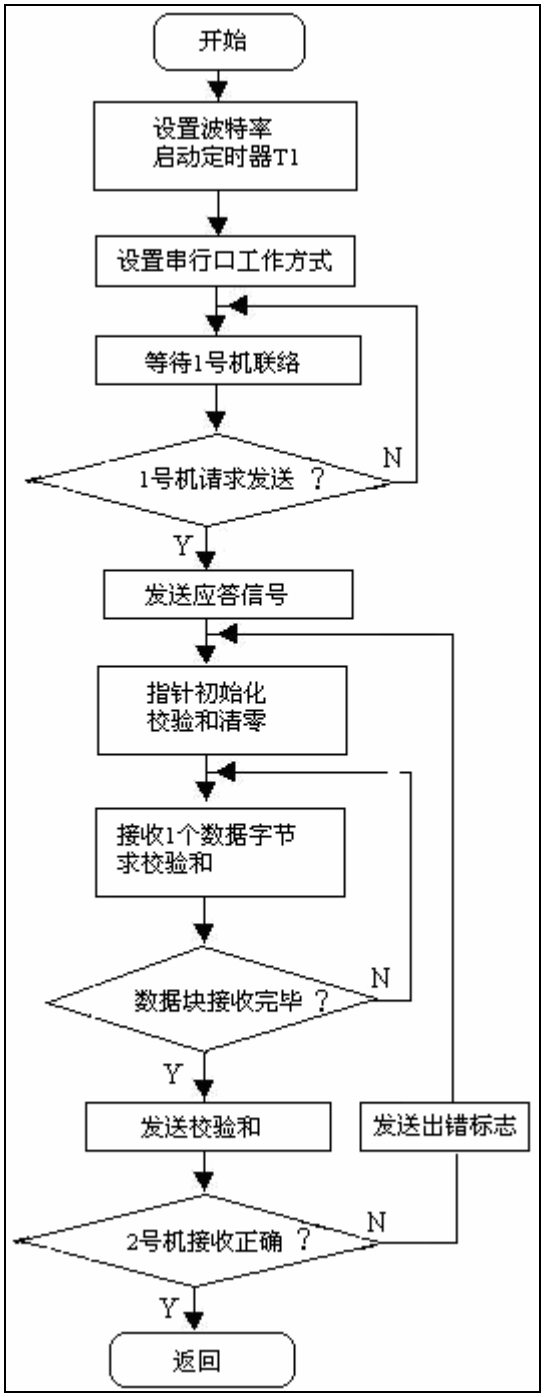
```
        ORG   1000H
ASTART: CLR   EA
        MOV   TMOD，#20H     ；定时器 1 工作于方式 2
        MOV   TH1，#0F4H     ；装载定时器初值，波特率为 2400bps
        MOV   TL1，#0F4H
        MOV   PCON，#00H     ；波特率不加倍
        SETB  TR1            ；启动定时器 T1
        MOV   SCON，#50H     ；串行口工作于方式 1，允许接收
ALOOP1: MOV   SBUF，#0E1H    ；发送联络信号
        JNB   TI，$          ；等待一帧发送完毕
        CLR   TI             ；允许再发送
        JNB   RI，$          ；等待 2 号机的应答信号
        CLR   RI             ；允许再接收
        MOV   A，SBUF        ；2 号机应答后，读至 A
        XRL   A，#0E2H       ；判断 2 号机是否准备完毕
        JNZ   ALOOP1         ；2 号机未准备好，继续联络
ALOOP2: MOV   R0，#40H       ；2 号机准备好，设定数据块地址指针初值
        MOV   R7，#10H       ；设定数据块长度初值
        MOV   R6，#00H       ；清校验和单元
ALOOP3: MOV   SBUF，@R0      ；发送一个数据字节
        MOV   A，R6          ；检验字送累加器 A
        ADD   A，@R0         ；求校验和
        MOV   R6，A          ；保存校验和
```

图 5.2-15 由 MCS-51 构成的双机通信程序流程图

(a)发送程序流程图



(b)接收程序流程图




```
INC   R0              ；地址单元加 1
JNB   TI，$           ；等待发送完
CLR   TI              ；清发送标志位
DJNZ  R7，ALOOP3      ；整个数据块是否发送完毕
MOV   SBUF，R6        ；发送校验和
JNB   TI，$           ；等待发送完
CLR   TI              ；清发送标志
JNB   RI，$           ；等待 2 号机的应答信号
```

```
        CLR   RI                ；清接受标志
        MOV   A，SBUF           ；2 号机应答，读至 A
        JNZ   ALOOP2            ；2 号机应答“错误”，转重新发送
        RET                     ；2 号机应答“正确”，返回
        END
```

（2）接收程序清单

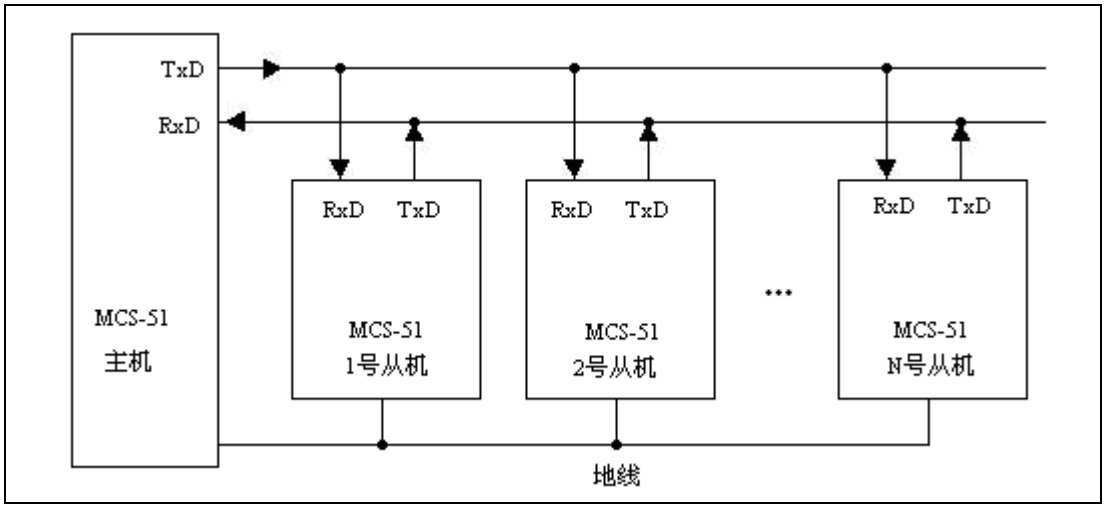
```
        ORG   1000H
BSTART: CLR   EA
        MOV   TMOD ,#20H        ；设置定时器 1 工作方式 2
        MOV   TH1，#0F4H        ；设置 T1 初值
        MOV   TL1，#0 F4H
        MOV   PCON，#00H        ；波特率不加倍
        SETB  TR1               ；启动 T1
        MOV   SCON，#50H        ；设定串口工作方式 1，且准备接收
BLOOP1: JNB   RI ,$             ；等待 1 号机的联络信号
        CLR   RI                ；清接收标志位
        MOV   A，SBUF           ；收到 1 号机的信号
        XRL   A，#0E1H          ；判断是否为 1 号机联络信号
        JNZ   BLOOP1            ；不是 1 号机联络信号，再等待
        MOV   SBUF，#0E2H       ；是 1 号机联络信号，发应答信号
        JNZ   TI，$             ；等待发送完毕
        CLR   TI                ；清发送标志
        MOV   R0，#40H          ；设定数据块地址指针初值
        MOV   R7，#10H          ；设定数据块长度初值
        MOV   R6，#00H          ；清校验和单元
BLOOP2：JNZ   RI，$             ；等待接收信息
        CLR   RI                ；清接收标志位
        MOX   A，SBUF           ；读取接收缓冲区内容
        MOX   @R0，A            ；接收数据转储
        INC   R0                ；存储单元加 1
        ADD   A，R6             ；求校验和
        MOV   R6，A             ；校验和存入 R6
        DJNZ  R7，BLOOP2        ；判断数据块是否接收完毕
        JNB   RI，$             ；完毕，接收 1 号机发来的校验和
        CLR   RI                ；清接收标志位
        MOV   A，SBUF           ；发送
        XRL   A，R6             ；比较校验和
        JZ    EDN1              ；校验和相等，跳至发正确标志
        MOV   SBUF，#0FFH       ；校验和不相等，发错误标志
        JNB   TI，$             ；转重新接收
        CLR   TI                ；清发送标志位
END1：  MOV   SBUF，#00H        ；缓冲区清零
        RET                     ；子程序返回
        END
```

3）多机通信（*）

在实际应用系统中，经常需要多个单片机芯片之间协调工作，即多机通信。主从式多机通信是多机通信中应用最广泛的一种，也是最简单的一种。利用 MCS-51 单片机串行口可实现多机通信。串行口用于多机通信时必须使用具有主从通信功能的方式 2 或方式 3。

所谓主从式多机通信，即在数个单片机中，只有一个是主机，其余都是从机。主机发出的信息只能传送到所有从机或指定的从机，而从机发出的信息只能被主机接收，各从机之间不可以直接通信，各从机的之间的通信必须通过主机进行。在由 MCS-51 组成的主从式多机通信系统中，主机只有一台，从机最多有 256 台。由 MCS-51 构成的多机通信系统如图 5.2-16。

图 5.2-16 由 MCS-51 构成的主从式多机通信系统



在主从式多机通信系统中，主机和从机只能工作在方式 2 或方式 3。主机发出的信息有两类，一类是地址信息（用于确定与主机通信的从机地址，特征是串行发送的第 9 位数据 TB8 为 1），一类是数据信息（特征是串行发送的第 9 位 TB8 为 0）。即在多机通信系统中，方式 2、3 只有 8 位数据，第 9 位（即 TB8）是地址/数据标志位。

主从式多机通信过程中，主机的 SM2 位必须为 0，以确保主机能够接收从机发送的地址信息（第 9 位为 1）和数据信息（第 9 位为 0）。对从机来说，要利用 SCON 寄存器的 SM2 位的控制功能，接收时，若 RI=0，则只要 SM2=1，总能实现接收；而若 SM2=0 时，则发送的第 9 位数据 TB8 必须为 0 接收才能进行。因此，对于从机来说，在接收地址时，应使 SM2=1，以便接收主机发出的地址帧信息，当发现主机送出的地址与本机地址相同时，即认为主机要与自己通信，一经确认，从机应使 SM2=0，以便接收主机随后送出的数据信息，并把本站地址发回主机作为应答。

主从式多机通信过程如下：

（1）使所有的从机的 SM2 位置 l，以便接收主机发来的地址；

（2）主机发出一帧地址信息，其中包括 8 位需要与之通信的从机地址，第 9 位为 l，然后进入接收状态，接收从机应答信号（即相应从机的地址信息）；

（3）所有从机接收到主机发出的地址帧后，将该地址与本机地址相比较，当接收到的地址信息与本机地址相符时，表示主机要与本机通信，本机被选中。被选中的从机将本机地址信息发回给主机，然后执行“CLR SM2”指令，使 SM2=0；当接收到的地址信息与本机不符时，表示未被选中，未被选中的主机应仍保持 SM2＝1 的状态，对主机随后发来的数据信息不予理睬，直至发送新的地址帧；

（4）主机收到从机的应答信号后，给已被寻址的从机发送控制指令和数据(数据帧的第

9 位为 0)；

（5）从机正确接收主机发来的数据后，发送应答信号给主机，将 SM1 置位，主机与从机通信过程结束。

与双机通信一样，单片机的多机通信中，主机和从机之间的通信必须采用同一个通信协议，设计者根据需要设计或选择合适的通信协议，并按照协议的规定编写主机和从机的通信程序。

本章小结

本章主要介绍了 MCS-51 单片机非常实用的部分——定时/计数器。当需要对外部时间进行计数或定时输出一个信号用于控制外部事件时，可以充分利用 T0 或 T1 的功能加以实现。对于定时/计数器的工作模式和 4 种工作方式的选择，可以通过灵活设置 TMOD 和 TCON 专用寄存器的相应位来实现。在本章中，对定时/计数器的设计提供了实例，供读者参考。

对于 MCS-51 单片机而言，片内提供 UART 异步串行通信接口，实现单片机与外部设备之间的信息传输。本章也详细介绍了串口通信的基本概念和单片机串行通信接口的结构、特点、工作方式以及控制寄存器设置，最后还简单介绍了单片机串口通信的应用。

思考题与习题

5.1　定时/计数器的作用是什么，有什么特点?

5.3　MCS—51 单片机有几个定时/计数器，各是多少位，计数的来源有哪些?

5.4　MCS—51 单片机的定时/计数器有哪几种工作方式，各有什么特点?

5.5　若单片机晶振为 11.0592MHz。用 T0 产生 1μs 的定时，可以选择几种方式？分别写出定时器的方式字和计数初值。

5.6　设 MCS—51 单片机的晶振频率为 6MHz，使用 T1 对外部事件进行计数，每计数 200 次后，T1 转为定时工作方式，定时 5ms 后，又转为计数方式，如此周而复始地工作，试编程实现。

5.8　当定时器 T1 用作方式 3 时，由于 TRl 已被 T0 占用，如何控制 T1 的开启与关闭?

5.9　以 T1 进行外部事件计数。每计数 2000 个脉冲后，转为定时方式，定时 20 ms 后又转为计数方式，如此循环不止。假定单片机晶振频率为 6MHz，请使用方式 1 编程实现。

5.12 已知 805l 单片机的 f_{OSC}＝12MHz，用 t1 定时，试编程由 P1.0 和 P1.1 分别输出周期为 2ms 和 500μs 的方波。

5.13 RI 标志位的功能是什么？

5.14 MCS—51 串口有几种工作方式，它们各自的特点是什么?

5.15 如果单片机系统晶振频率为 11.059MHz，要采用 9600 的波特率，应该如何设置？

5.16 并行通信与串行通信的主要区别是什么，各自有什么优缺点?

5.17 异步通信与同步通信的主要区别是什么?

5.18 试述 MCS—51 单片机串行口的四种工作方式、工作原理、字符格式及波特率的产生方法。

5.19 试用查询法编写串行口方式 3 下的接收程序。设波特率为 2400b/s，f_{OSC}＝6MHz，接收数据区在片外 RAM，起始地址为 RTAB，块长度为 40 字节，采用奇校验，放在接收数据第 9 位上。

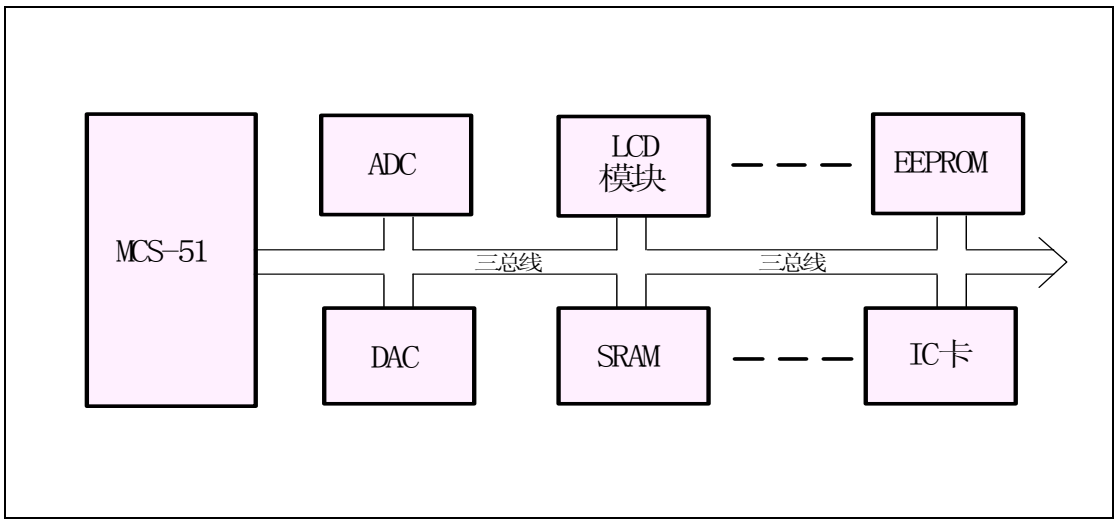
5.20 串行口多机通信的原理是什么?其中 SM2 的作用是什么?与双机通信的区别是什么?

第 6 章　MCS-51 单片机系统扩展技术

我们学习 MCS-51 的最终目的，是要用它来组成实际应用系统。MCS-51 具有体积小、功能强的特点，对于简单的控制，最小系统即可满足要求。所谓最小系统是指利用 MCS-51 自身的资源，无需外加其它功能性的器件所形成的控制系统。如一片 89S51（加上自身需要的晶体振荡器部分）就构成了一个最小应用系统。但实际使用中由于测控对象要求的多样性和复杂性，许多情况下 MCS-51 自身的资源和功能还不够，需要连接外部器件进行资源和功能的扩充，才能构成符合实用要求的 MCS-51 单片机应用系统。

MCS-51 应用系统的一般性结构如图 6.1-1 所示。图中以 MCS-51 为控制核心，通过三总线——数据总线、地址总线和控制总线——扩展各种不同的外围器件芯片，组成了功能各异的应用系统。一个完整的 MCS-51 应用系统，就是由 MCS-51 系列芯片扩展外围器件芯片组成的。这种通过扩展外围器件形成应用系统的原理和方法，我们称之为接口技术，接口技术是我们学习和使用单片机所必须掌握的知识和技能。

图 6.1-1 MCS-51 应用系统一般性结构图



接口技术涵盖的内容较多，从功能上上可分为两大类：MCS-51 自身资源的扩展技术和外部功能器件的扩展技术；从扩展的方式上可分为并行扩展方式和串行扩展方式；从实用上又可分为多种专门的典型应用，如打印机、IC 卡等。因此，一些教材对接口技术的分类也不尽相同。为突出特点，本教材将它们分别安排在第 6、7、8 三个章节中叙述。本章主要讨论 MCS-51 单片机自身资源的扩展技术，包括存储器、I/O 和中断源的扩展方法；外部功能器件的扩展技术分别在第 7 章和第 8 章介绍。

6.1　扩展技术的基本内容、原理和方法

MCS-51 通过扩展外部电路而组成实际应用系统。应用系统有三个组成部分：单片机（也叫最小系统）、三总线和外围电路。扩展技术的基本内容就是以单片机为核心，以三总线为接口，连接相关外围集成电路（IC）芯片而形成的符合 MCS-51 指令时序要求的应用系统。

应用系统中所有外围集成电路（IC）芯片都可看作是 MCS-51 的检测和控制对象。MCS-51 对它们的检测和控制是通过传送数据来实现的。也就是说，解决了和外围 IC 芯片之间的数据传送问题，就实现了单片机对外围电路的测控，也就实现了单片机的系统扩展。

应用系统数据传送是通过三总线来进行的。简单说来，数据总线是传送数据的载体，数据在其中传送；地址总线指定了数据传送的位置，保证数据传送给指定的对象；控制总线决定了数据传送的时刻和方向。

要保证数据的正确的交换，需要解决的问题是：

● 数据传送对象的唯一性。必须保证和 MCS-51 交换数据的 IC 芯片及其内部的存储单元或 I/O 单元是唯一的,即在任一时刻只有一个外围对象取得了和 MCS-51 交换数据的资格，未获得资格的其余 IC 芯片及其内部的存储单元或 I/O 单元不能参与和 MCS-51 的数据交换。

● 数据传送方向的确定性。在任一时刻数据的传送只能是一个方向，对于传送对象来说，或者是输出数据,或者是输入数据。

● 数据传送时刻的可控性。即在规定的时刻进行数据的发送或接收。

三总线解决这些问题的原理和方法是：

（1）用“地址总线”来确定数据传送对象的唯一性。我们通过地址总线为外围 IC 芯片分配地址编码，16 位二进制地址总线可编码的范围是 0000H～FFFFH,共 64K（65536）个存储（I/O）单元。所有的外围 IC 芯片存储单元，被分成了两个大类：程序存储器类和数据存储器类，每各大类自占据了 64KB 的地址空间。我们使用 MOVX 和 MOVC 两种指令来对这两个 64KB 区间进行寻址操作，合计可寻址 128KB 个单元。其中数据存储器区还包含了非存储器类的其它 IC 器件（如 A/D、D/A、LCD 模块等），这些单元地址统称为 I/O 地址，和数据存储单元一起被编址。由此可见，16 条地址总线编码了 64KB 的地址空间，在此范围之内的两大类的外围 IC 芯片的每个地址单元，都有自己唯一的地址码，如同唯一的名字。MCS-51 通过地址总线的编码来指定这些器件单元，就可以准确定位数据交换的对象——被指定为交换对象的地址单元允许和 MCS-51 交换数据；未被指定为交换对象的其余地址单元则被禁止交换数据。

（2）用“控制总线”来确定传送方向和传送时刻。在 MCS-51 和外围器件组成的系统中，数据传送的方向一般用“输入（I）”和“输出(O)”表示，而 I/O 一般是针对 MCS-51 主机而言的：从 MCS-51 向外围器件传送数据叫输出（O），也叫“写”；反之，从外围器件向 MCS-51 传送数据叫输入（I），也叫“读”。我们用控制总线 $\overline{RD}$ 和 $\overline{WR}$ 信号来读写外围 RAM 器件和 I/O 器件的数据，使用的指令是 MOVX；用控制总线中 $\overline{PSEN}$ 信号来读取外围 ROM 器件的固定常数和表格数据，使用的指令是 MOVC。

由此可见：

扩展技术的基本内容就是用三总线连接外围电路组成应用系统。

扩展技术的基本原理就是用三总线时序信号控制外围电路的数据交换。

6.1.1 MCS-51 的三总线信号接口

MCS-51 单片机的三总线信号接口如图 6.1-2 所示。图中可见：

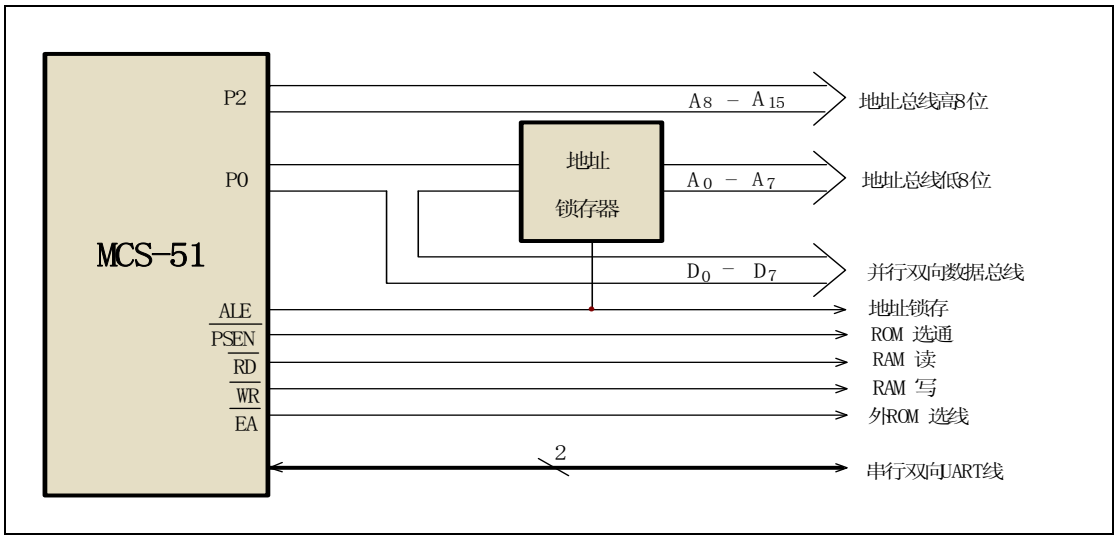
（1） P2 口输出地址总线的 A_8~A_{15}，是高 8 位地址总线信号的接口；

（2） P0 口为低 8 位地址总线信号和数据总线信号的复用接口，分时传送低 8 位地址总线信息 A_0~A_7 和 8 位的数据信息 D_0~D_7。其中 A_0~A_7 在控制总线 ALE 控制信息的作用下，经“地址锁存器”锁存后输出，和 P2 口一起构成了连接外部器件的 16 位的地址总线信号 A_0~A_{15}；

（3）控制总线信号中，ALE 作为 A_0~A_7 的地址锁存信号，用来将 A_0~A_7 从 P0 端口中分离出来；它也可作为频率信号供外部器件使用； $\overline{PSEN}$ 是程序存储器的选通信号，在执行 MOVC 指令时有效（出现），用来读出 ROM 中的常数和表格； $\overline{RD}$ 和 $\overline{WR}$ 是数据存储器的选通信号，在执行 MOVX 指令时有效（出现），分别用来读出和写入数据存储器的数据。 $\overline{EA}$

用来选择外部 ROM 的低 4K 区域，该信号有效时（接地），程序存储器 0000~0FFFH 存储单元在外部 ROM 芯片中。

图 6.1-2　MCS-51 单片机的三总线信号接口示意



6.1.2　MCS-51 系统扩展的三总线方法

为了和 MCS-51 相连接，外围 IC 芯片也都具有自己三总线——数据线、地址线和控制线——的引脚配置。其中“数据线”引脚用来和 MCS-51 交换数据；“地址线”用来接收 MCS-51 的地址编码数据，译码后指向内部的对应存储单元。IC 芯片内部的存储单元（或 I/O 单元）的数量决定了该 IC 芯片地址线引脚的数量。 N 条地址线可编码的存储单元数量为：存储单元数量=2^N。比如，某 IC 芯片内部的存储单元数目是 1024(1K)=2^{10}，则需要 10 条地址线来对这些存储单元进行地址编码；“控制线”用来接收 MCS-51 的控制信息，如读/写允许引脚等。IC 芯片还有一个“片选”（常用$\overline{CS}$或$\overline{CE}$符号）引脚，用来决定该 IC 芯片是否允许工作：该引脚信号有效时允许和 MCS-51 交换数据，无效时禁止交换数据。

MCS-51 与外部 IC 芯片扩展的基本方法，就是将两者的三总线引脚正确连接。具体方法是：

1）数据线的连接。 在 P0 口负载能力满足外围电路输入电流要求的情况下，将 MCS-51 的 P0 口和外围器件芯片的数据线对应引脚直接相连即可。否则应在两者之间添加电流驱动器件；

2）控制线的连接。将$\overline{PSEN}$连接到程序存储器件芯片的输出有效信号引脚$\overline{OE}$；$\overline{RD}$连接数据存储器件芯片的$\overline{OE}$或$\overline{RD}$；$\overline{WR}$连接数据存储器间芯片和 I/O 芯片的$\overline{WE}$或$\overline{WR}$端。即在地址编码相同时，用不同的控制信号来区分数据和程序两类存储芯片；

3）地址线的连接。地址线连接需满足两个条件：一是要指定某个芯片工作；二是要指定该芯片内部的存储单元。前者我们叫做“片选”（选择芯片），即使该芯片的$\overline{CS}$引脚信号有效；后者我们叫做“字选”（选择字节单元）。

字选：MCS-51 地址总线的最低位 A_0 连接到外部芯片的最低位地址线引脚，然后依次连接 A_1、A_2……，直到芯片的最高位地址线引脚。比如，对应于某 8K 存储器芯片，MCS-51 的 A_0~A_{12} 和芯片的最低地址引脚 A_0 到最高地址引脚 A_{12} 相连，即如前所说，N 条地址总线 A_0~A_{N-1} 的编码，对应了芯片内部的 2^N 个存储单元。

片选：字选余下的高位地址线，可用来作为选择芯片的地址总线。常用的有“线选” 和

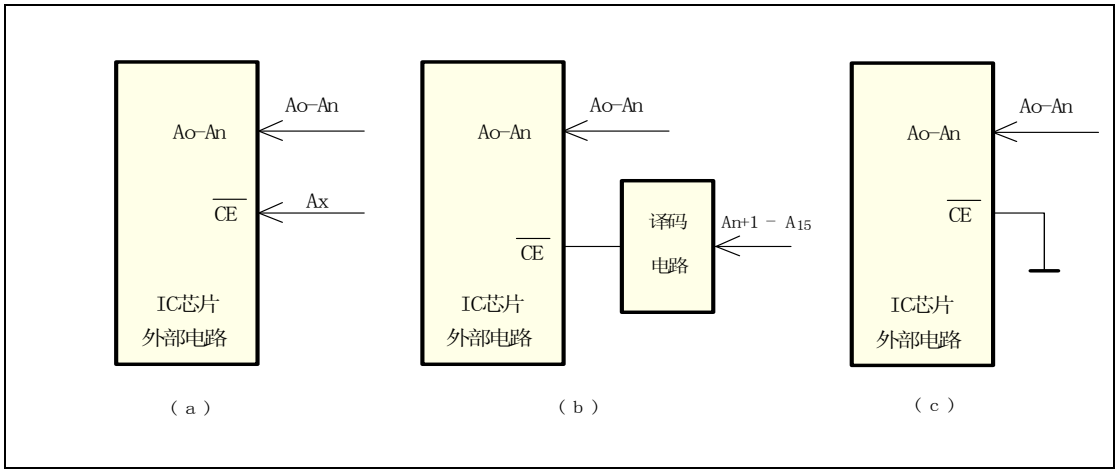
“译码”两种方式。

（1）“线选”法。在外围器件较少、MCS-51 高位地址总线剩余较多的情况下，可以使用某条高位地址总线（一般从最高位 A_{15}（P2.7）开始向下取），作为某器件的“片选”信号，这种连接最简单。如图 6.1-3（a）所示。

（2）“译码”法。地址总线输出地址编码，经过译码电路产生出片选信号。 “译码”可分为全译码和部分译码。全译码是将字选后剩余的所有高位地址线都参与译码；部分译码是用剩余的一部分地址线译码。前者的优点是地址唯一，缺点是译码电路多。译码电路可以使用普通逻辑门电路，如图 6.1-3（b）所示；也可以使用后文将要介绍的专用译码芯片。

此外，在某类 IC 芯片只有一片时，由于不会发生和其它同类芯片的地址冲突，常将其片选端接固定电平，使之一直有效。如图 6.1-3（c）所示。

图 6.1-3　外部芯片“片选”的方式



综上所述，我们把 MCS-51 应用系统外部扩展方法的原则表述为：

（1）使用相同控制信号的 IC 芯片，片选地址不能相同。比如两片程序存器 2764,使用相同的控制信号 $\overline{PSEN}$,两者的片选地址必须有区别；

（2）使用相同片选地址的 IC 芯片，各自的控制信号不能相同。比如同是 8K 存储容量的程序存储器 2764 和数据存储器 6264,由于其控制信号不同，可使用相同的片选地址。

6.1.3 地址锁存器和地址译码器

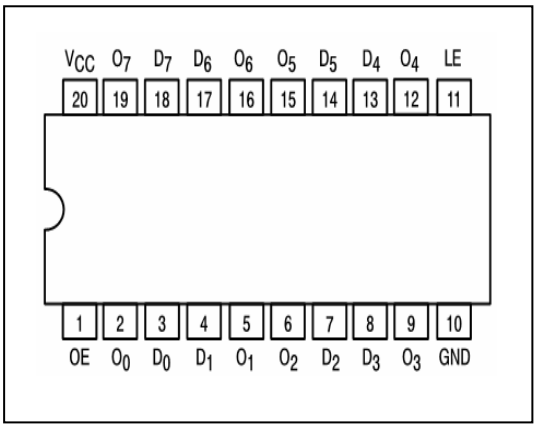
6.1.3.1 地址锁存器

地址锁存器用来将地址/数据复用端口 P0 的地址总线 A_0~A_7 分离出来并保持，形成对外部 IC 芯片编码的低 8 位地址总线。地址锁存器可用 8 位 D 触发器如 74LS373、74LS573 等 IC 芯片。74LS373 的外型图、引脚图和功能表如图 6.1-4 所示。

由 LS373 功能表知，在拴锁允许控制端 LE(引脚 11) =高电平时，输出端 On 的电平随输入端 Dn 而变化，LE=低电平时，输出端 On 的电平被保持住(O_0)而不再变化。即 On“锁存”了 LE=0 开始瞬间 Dn 端的数据。

我们把 P0 口和 74LS373 的 Dn 端按二进制位从低到高对应连接，地址锁存允许信号 ALE 和 74LS373 的拴锁允许控制端 LE 相连，在 P0 端口输出低 8 位地址总线编码数据时，用 ALE 的下降沿将它们锁存存在 74LS373 的 Dn 端。从而实现了分离 A_0~A_7 总线数据并锁存的目的。

图 6.1-4　(a) 74LS373 的引脚图



(b) 373 功能表

输入 D_n	锁存允许 LE	使能 OE	输出 O_n
H	H	L	H
L	H	L	L
X	L	L	O_0
X	X	H	高阻

6.1.3.2 地址译码器

地址译码器是对地址总线中“字选”所剩余的高位地址线的编码数据进行译码，译出地址线编码所对应的 IC 芯片。译码器的有效输出是低电平，连接到 IC 芯片的“片选”引脚。因此，地址译码器也可以理解为“片选译码器”。它可以用普通逻辑门电路组成，也可以使用专用的译码器件。

（1）用普通逻辑门电路组成译码电路

普通逻辑门电路，如 TTL 74LS 系列和 CMOS 4000 系列门电路，都可组成地址译码电路。设某外围芯片的“字选”地址线为 A_0~A_{11}（容量 4KB），“片选”地址编码为 $A_{15}\overline{A_{14}}A_{13}\overline{A_{12}}$，使用普通逻辑门器件组成的该译码电路如图 6.1-5 所示，该器件的地址排列如表 6.1-1。其中，“字选”部分的地址可变，从 A_{11}~A_0 全“0”到全“1”共 4096 个（4KB）存储单元；“片选”部分的编码地址固定，即 $A_{15}\overline{A_{14}}A_{13}\overline{A_{12}}$=1010B。在 16 位地址线的高 4 位等于“1010B” 时，才选中该芯片工作。因此，该片内字节单元的地址范围是 A000H~AFFFH，共 4K 字节。该译码电路使用了所有的地址总线，所以叫做“全译码”。如果该芯片使用 3 条地址线 $A_{15}\overline{A_{14}}A_{13}$ 作为“片选”，那就是“部分译码”。“部分译码”的地址会有重复的现象，原因是未参与译码的地

图 6.1-5 用逻辑门器件的译码电路

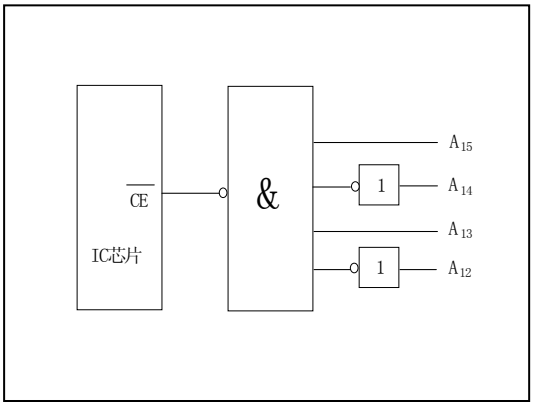


表 6.1-1

片　选 $A_{15}A_{14}A_{13}A_{12}$	字　选 $A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$		
1010	0000	0000	0000
1010	1111	1111	1111

址线 A12 的状态不会对片选产生影响：若 A12=0，则片内字节单元的地址范围是 A000H~AFFFH，若 A12=1，则片内字节单元的地址范围是 B000H~BFFFH。在使用时取其

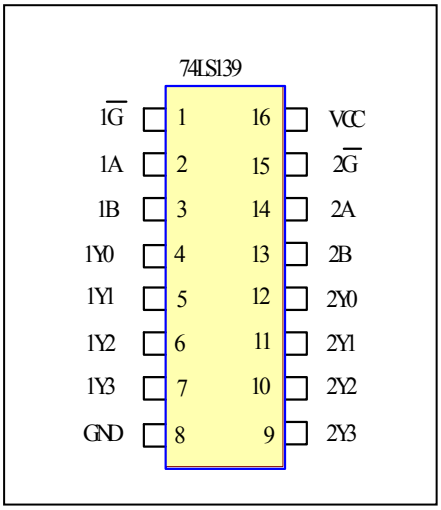
任意一组即可。在一般情况下，选用未参与译码的地址线=1 的地址范围。

（2）专用译码器

常用的专用译码器芯片有 74LS139（双 2—4 译码器）、74LS138（3—8 译码器）、74LS154（4—16 译码器）等，现以 74LS 139 为例介绍。

图 6.1-6 是 74LS139 的引脚图和功能表。它是两个 2-4 译码器。每个有 2 个编码输入端 B、A，4 个译码输出端 Y0~Y3。在允许端 $\overline{G}$=L（0）时，B 和 A 的输入编码被译码，译码结果对应的输出引脚被置为低电平，其余未被译码的引脚保持高电平。 B A 按 2 位二进制编码 00~11 顺序排列，对应的译码输出顺序为 Y0~Y3。我们用 2 条高位地址线接 B A，则 Y0~Y3 即是 2 条地址线的 4 个译码输出。显然，这种专用的译码电路使用很方便，特别是在扩展外 IC 芯片数量较多的时候更是如此。

图 6.1-6 74LS139 引脚和功能图



功能表

输入			输出			
$\overline{G}$	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L
X	X	X	H	H	H	H

例 6.1　试用 16KB 的存储器芯片，组成容量 64KB 的存储器区，请问：

① 需用多少个存储器芯片？多少条地址线？其中“字选”用哪些地址线、“片选” 用哪些地址线？

② 若用 74LS139 译码器，试画出译码电路，并标出其输出引脚的选址范围。

③ 若该用线选法，能组成存最大的储空间是多少？

解：① 64KB÷16KB = 4（片）　　需要 4 片 16KB 的存储器芯片

64KB = 2^{16}　　需要 16 条地址线

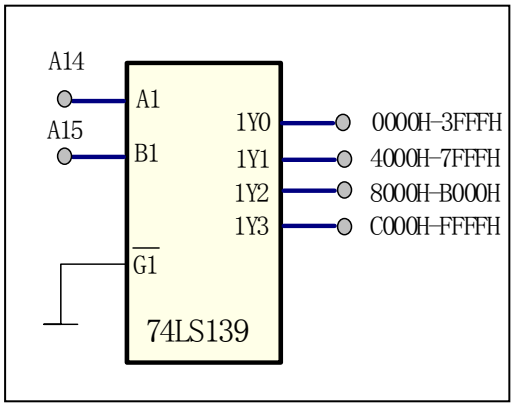
16KB = 2^{14}　　需要条 14 条“字选”线

4 片 = 2^2　　需要 2 条“片选线”

② 低位 14 地址线 A_0~A_{13} 为“字选”，高 2 位地址 $A_{15}A_{14}$ 线为“片选”，用 74LS139 专用译码器组成的全译码电路如图 6.1-7（a）。139 各译码输出引脚的片的地址范围如 6.1-7（b）中表格所示。

③ 若用“线选”法，A15、A14 各自只能选一片 16KB 的芯片，所以组成的存储空间为 16KB×2 = 32KB，其地址范围是：芯片 1：A15=0，4000H~7FFFH；芯片 2： A14=0，8000H~BFFFH。

图 6.1-7 （a）例 6.1 题 139 译码地址范围



（b） 各存储芯片编址范围

139	片选	位选
译码	A_{15} A_{14}	A_{13} $A_{12}A_{11}A_{10}A_9A_8$ $A_7A_6A_5A_4A_3A_2A_1A_0$
1Y0	0 0	00 0000 0000 0000
	0 0	11 1111 1111 1111
1Y1	0 1	00 0000 0000 0000
	0 1	11 1111 1111 1111
1Y2	1 0	00 0000 0000 0000
	1 0	11 1111 1111 1111
1Y3	1 1	00 0000 0000 0000
	1 1	11 1111 1111 1111

6.2 存储器扩展

我们知道，从使用者的角度看，MCS-51 的存储器有三个空间：片内 256B 的数据存储器、片外 64KB 的数据存储器和片内外统一编址的 64KB 程序存储器。在 MCS-51 的内部存储资源区间不能满足应用系统要求的时候，需要进行外部存储器的扩展。这是 MCS-51 资源扩展的主要内容之一。

扩展存储器是通过外接存储器芯片实现的。存储器芯片的种类很多，传统的区分是按使用功能分为只读存储器（ROM）和随机存储器（RAM）两大类器件。ROM 器件用来存储程序代码和固定的表格数据，称为程序存储器或只读存储器；RAM 器件用来存放实时变化的变量数据，称为数据存储器或随机存储器。如 EPROM 类型的芯片作程序存储器用，SRAM 类型的芯片作数据存储器用。近年来随着存储器芯片制造技术的进步，传统意义上的这种分类界限正被打破，比如 EEPROM、Flash 类型的芯片，就既能作程序存储器，也可作数据存储器使用。该技术的意义在于扩展了器件的功能和降低了使用成本。

本节通过三种常用类型的器件，静态型数据存储器 SRAM、电擦除可编程只读存储器 EEPROM 和闪速存储器 PEROM(Flash)，介绍 MCS-51 外部存储器扩展的一般方法。

6.2.1 静态数据存储器SRAM的扩展

MCS-51 内部的 RAM 区间是 256 字节。除去 SFR 占用的 128 字节，可供用户使用的只有 128 字节。通过外接芯片，可在外部扩展至 64KB 的数据存储器空间。MCS-51 应用系统扩展的数据存储器采用 SRAM（静态型）类型芯片。

实现外部数据存储器的扩展，必须符合 MCS-51 的三总线对外部 RAM 区进行读写操作的时序要求。对外部 RAM 区进行读写操作的时序由 CPU 控制，在执行指令 MOVX 时出现。

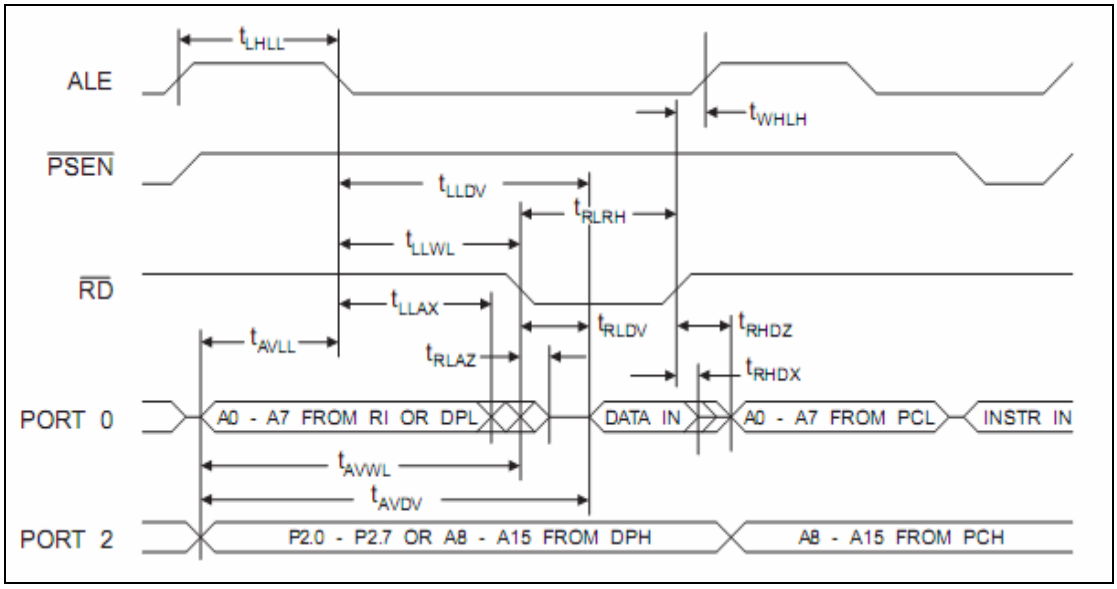
6.2.1.1 外部 RAM 存储器的读写时序

图 6.2-1 分别给出了 MCS-51 读外部 RAM 区时序和写外部 RAM 区时序。

1）读外部数据存储器

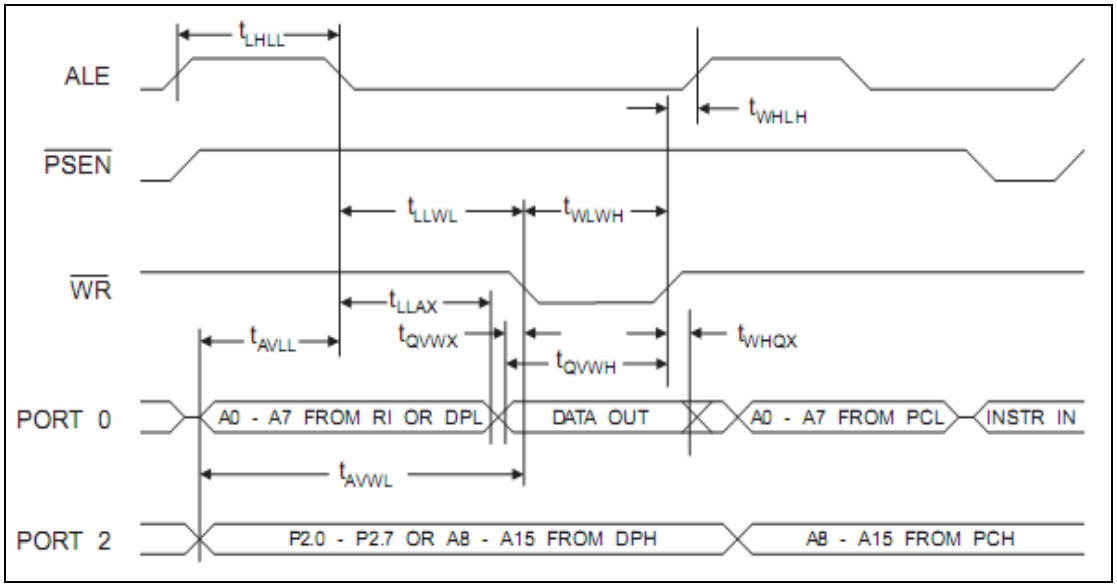
执行 MOVX A，@Ri 和 MOVX A,@DPTR 指令时进入读外部 RAM 时序。在 ALE 高电平的后半部分，P0 口输出来自 Ri（或 DPL）的低 8 位地址编码数据 A_0~A_7 有效，并进入稳定的状态；在 ALE 的下降沿将稳定的 A_0~A_7 编码数据锁存；在读信号 RD 有效时，其

图 6.2-1 （a）外部 RAM 读时序图



下降沿将 P2 口输出来自 DPH 的高 8 位地址编码数据 A_8~A_{15} 和被锁存的低 8 位地址编码数据 A_0~A_7 合在一起，作为完整的 16 位地址编码信号（片选和字选），使相应存储器存储单元被译码；该存储单元的数据被传送到数据总线上；在 $\overline{RD}$ 有效的中部时刻，数据被读到 P0 口，送入 CPU。

图 6.2-1（b）外部 RAM 写时序图



② 写外步数据存储器。

执行 MOVX @Ri， A 和 MOVX @DPTR， A 指令时进入写读外部 RAM 时序。在 ALE 高电平的后半部分，P_0 口输出来自 Ri（或 DPL）的低 8 位地址编码数据 A_0~A_7 有效，并进入稳定的状态；ALE 的下降沿将稳定的 A_0~A_7 编码数据锁存；然后写信号 $\overline{WR}$ 有效，其下降沿将 P2 口输出来自 DPH 的高 8 位地址编码 A_8~A_{15} 和被锁存的低 8 位地址编码据 A_0~A_7 合在一起，作为完整的 16 位地址编码信号（片选和字选），外部 RAM 相应的存储单元被译码；此时数据从 P0 口稳定在数据总线上，在 $\overline{WR}$ 的中部位置，该数据写入到被译码

存储单元中。

以上时序告诉我们，对外部数据存储器 RAM 的读写，都是通过以下三个步骤进行的：

（1） ALE 下降沿将地址编码 A_0~A_7 锁存；

（2） $\overline{RD}$ 和 $\overline{WR}$ 的下降沿时地址编码被译码，选中外部 RAM 的存储单元；

（3） $\overline{RD}$ 和 $\overline{WR}$ 的中间位置时刻进行数据的传送，写数据由 P0 口输出到 RAM 存储单元；读数据由 RAM 存储单元输入到 P0 口。

6.2.1.2 SRAM（静态存储器）IS65C256 的扩展

MCS-51 应用系统常用的静态数据存储器，按存储容量有 8K×8Bit、16K×8Bit、32K×8Bit、64K×8Bit 等，现以 IS65C256 为例讲述。IS65C256 是 ISSI 公司的产品，CMOS 静态随机存储芯片，与 TTL 电平信号兼容。主要性能参数是：

- 存储容量 32KB×8 位，即 32K 字节
- 存取时间 25ns，45ns
- 5V 电源
- 低功耗 200mW，待机功耗 150uW

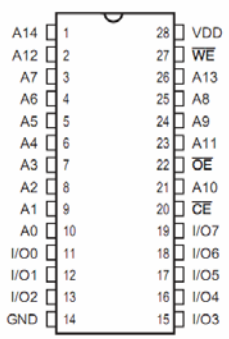
其引脚和真值表如图 6.2-2 示。

表中片选端 $\overline{CE}$ 用来选择芯片是否工作，$\overline{CE}$=H 时,芯片处于待机状态，I/O 线输出为高阻抗。$\overline{OE}$ 为数据输出允许控制引脚，可连接 $\overline{RD}$，在该引脚信号有效时，数据从 IS65C256 芯片的指定存储单元被读出到 P0 口；$\overline{WE}$ 为写入允许引脚，可连接 $\overline{WR}$，在该引脚信号有效时，数据从 P0 口被写入到 IS65C256 芯片的指定存储单元中。I/O_0~I/O_7 是数据线，连接至

图 6.2-2 IS65C256 静态随机存储芯片真值表

TRUTH TABLE

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	I/O Operation
Not Selected (Power-down)	X	H	X	High-Z
Output Disabled	H	L	H	High-Z
Read	H	L	L	DOUT
Write	L	L	X	DIN



P0 口；32K 字节存储单元有 15 条地址编码线，可接 MCS-51 的地址总线 A_0~A_{14}。MCS-51 与 IS65C256 连接如图 6.2-3 示。

图中，地址线的连接：ALE 接到 8D 锁存器 373 的 LE 端，其下降沿将 P_0 口的 A_0~A_7 锁存在 373 的 O_0~O_7 端，373 的 O_0~O_7 连接到 IS65C256 的 A_0~A_7 作为低 8 位的地址编码。IS65C256 的“字选”地址需要 15 条，A_8~A_{14} 和 MCS-51 P2 口的 A_8~A_{14} 对应相连。因只扩展一片,片选脚 $\overline{CE}$ 接地。

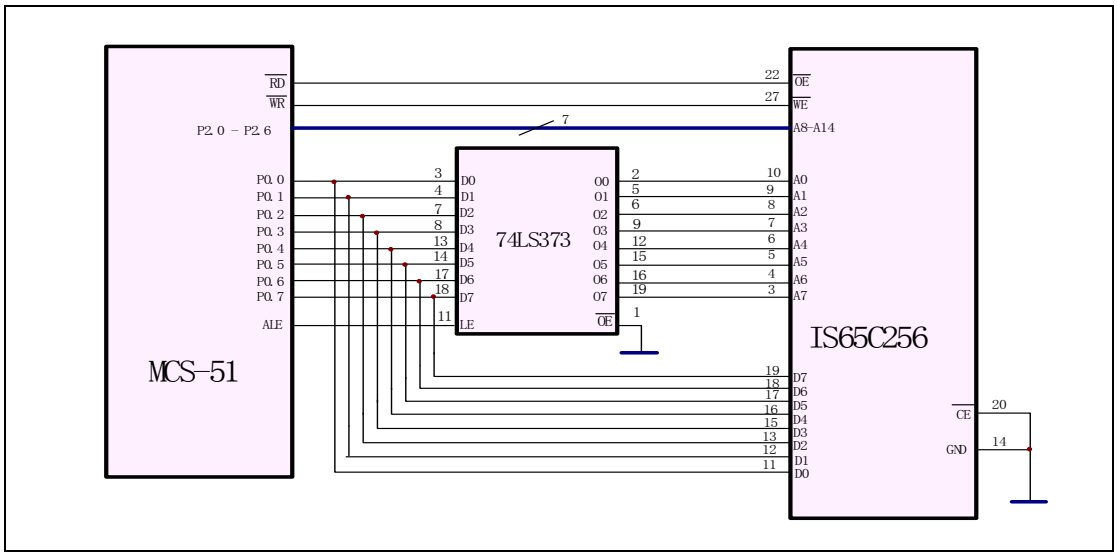
控制线的连接：$\overline{RD}$ 连 $\overline{OE}$，$\overline{WR}$ 连 $\overline{WE}$；

数据线的连接： $P0._0$ ~$P0._7$ 连 I/O_0~I/O_7。

该图中，IS65C256 的 32KB 个存储单元所对应的编码地址是：0000H～7FFFH。即

MCS-51 在外部扩展了 32KB 的静态存储器空间。

图 6.2-3　IS65C256 和 MCS-51 的连接



若写数据#54H 到 IS65C256 的 1090H 单元，可用 2 种编程方法实现：

```
方法 1:    MOV   DPTR,#1090H        ；送 16 位地址编码
           MOV   A，#54H            ；待写数据到 A
           MOVX  @DPTR,A            ；写数据到外 RAM 1090H 单元
方法 2：   MOV   P2，#10H           ；送高 8 位地址编码
           MOV   R0, #90H           ；送低 8 位地址编码
           MOV   A，#54H            ；待写数据
           MOVX      @R0, A         ；写数据到外 RAM 1090H 单元
```

同样，把 1090H 单元的数据读入 A，也有 2 种方式：

```
方法 1：   MOV   DPTR,#1090H        ；送 16 位地址编码
           MOVX  A，@DPTR           ；1090 存储单元数据读到 A
方法 2：   MOV   P2，#10H           ；送高 8 位地址编码
           MOV   R0, #90H           ；送低 8 位地址编码
           MOVX  A , @R0            ；1090 存储单元数据读到 A
```

6.2.2　EEPROM存储器的扩展

EEPROM 存储器，又叫 E^2PROM 存储器，是一种电擦除的可在线编程的只读存储器，主要作为程序存储器使用。所谓在线编程，是该器件无需使用专门的程序擦除设备，在它实际的工作线上（“在线”）就可以做程序的改写操作。和“在线”相对应的方式是“离线”，即离开工作线。EEPROM 存储器的程序写入过程，无需像紫外线擦除的可编程存储器 EPROM 那样，需要专用程序擦除器和外加的程序写入电源；而是用对数据存储器 RAM 的写入指令，就可以在线进行芯片的编程写入操作。而且写入新程序的时候，也无需先进行擦除芯片的指令操作，在新数据写入的同时，旧数据即被覆盖掉；还有，它的程序写入时间只有几个毫秒，而采用专用程序擦除写入器一般要用几十分钟的时间。——显而易见，这些特点要比紫外线擦除方式的 EPROM 类型的程序存储器芯片方便得多。因此，作为程序存储器的扩展技术，

本章将介绍 EEPROM 和 Flash 存储器，而不再介绍离线擦除的 EPROM 器件。

由于 EEPROM 类型的程序存储器具有在线写入方式，且写入时间只有几个毫秒，所以在某些场合，它也可以作为随机存储器 RAM 使用。但读者要注意，在把它们作为随机存储器 RAM 使用时，其最大读出时间为 150ns，和 SRAM 大致相当；而字节写（包括页面写）的最大时间是 5ms，则要比 SRAM 器件慢得多。

EEPROM 的型号以数字“28”开头，表 6.2-1 为安森美半导体公司该类芯片的部分型号。

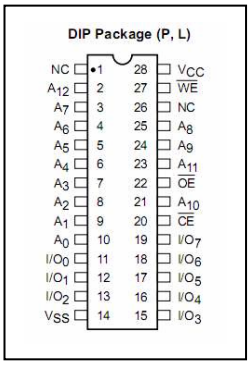
EEPROM 其主要特点是：

- 无须专门擦除，新数据写入即替换旧数据
- 读出时间为 ns 秒级
- 写入时间为 ms 秒级
- 擦写寿命 100,000 次
- 数据保留时间 100 年
- 使用单一的 5V 电源供电
- 芯片的引脚排列同 SRAM（6xxx）、EPROM（27xxx）兼容

表 6.2-1 常用 EEPROM 型号规格举例

型号	存储密度	VIN(V)	ICC(读写/待机)	读取时间(ns)	封装					温度范围
					PDIP	SOIC-JEDEC	SOIC-EIAJ	PLCC	TSOP	
CAT28LV64	64KBits	3.3-3.6	8mA/100μA	150,200,250	L(28)	W(28)	X(28)	G(32)	H13(28)	-55℃-125℃
CAT28LV65	64KBits	3.3-3.6	8mA/100μA	150,200,250	L(28)	W(28)	X(28)	G(32)	H13(28)	-55℃-125℃
CAT28LV256	256KBits	3.3-3.6	15mA/150μA	200,250,300	L(28)			G(32)	H13(28)	-55℃-125℃
CAT28C16A	16KBits	4.5-5.5	25mA/100μA	90,120,200	L(24)	W(24)	X(24)	G(32)		-55℃-125℃
CAT28C17A	16KBits	4.5-5.5	25mA/100μA	200	L(24)	W(24)	X(24)	G(32)		-55℃-125℃
CAT28C64B	64KBits	4.5-5.5	25mA/100μA	90,120,150	L(28)	W(28)	X(28)	G(32)	H13(28)	-55℃-125℃
CAT28C65B	64KBits	4.5-5.5	25mA/100μA	90,120,150	L(28)	W(28)	X(28)	G(32)	H13(28)	-55℃-125℃
CAT28C256	256KBits	4.5-5.5	25mA/150μA	120,150	L(28)			G(32)	H13(28)	-55℃-125℃
CAT28C257	256KBits	4.5-5.5	25mA/150μA	120,150	L(28)			G(32)	H13(28)	-55℃-125℃
CAT28C512	512KBits	4.5-5.5	50mA/200μA	120,150	L(28)			G(32)	H14(32)	-55℃-125℃
CAT28C513	512KBits	4.5-5.5	50mA/200μA	120,150				G(32)		-55℃-125℃

图 6.2-428C64A 引脚



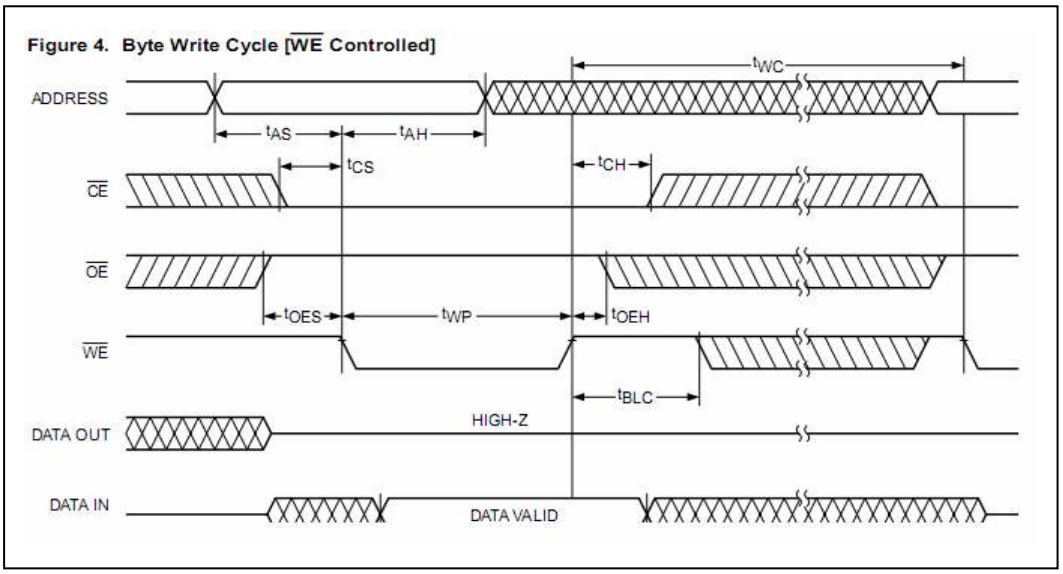
现以安森美半导体公司的 CAT28C64B 芯片为例，说明其工作原理和同 MCS-51 的连接。2864A 的封装有多种形式，采用双列直插形式的引脚见图 6.2-4。

2864A 的数据写入方式有 2 种：“字节写”和“页写”。字节写是对芯片的某一个指定存储字节单元的内容进行写入操作；页写则是对指定的一页存储字节单元的内容进行写入操作，一页的数量是 32 个字节。我们在对 EEPROM 进行页写入操作时，可以一次连续对页内 32 个字节进行写入操作，即每次最大的写入单元是 32 个字节。下面我们通过具体介绍这两种写的时序来理解这两种数据写入方式的意义。

（1）字节写。字节写的时序如图 6.2-5。一个字节写周期发生的条件是$\overline{OE}$=H，$\overline{CE}$和$\overline{WE}$=L。写周期的开始可用$\overline{CE}$或$\overline{WE}$控制，取决于哪个信号最后出现。6.2-5 图示中用$\overline{WE}$作为控制端。被写单元的地址在$\overline{WE}$的下降沿被芯片获取，被写单元要写入的数据在$\overline{WE}$的上升沿被芯片获取。$\overline{WE}$的上升沿后的 t_{WC} 时段内，是 EEPROM 芯片将获取的数据写入指定的地址单元的时间，该时间 EEPROM 芯片完成内部旧字节擦除和新字节写入工作，最大用时

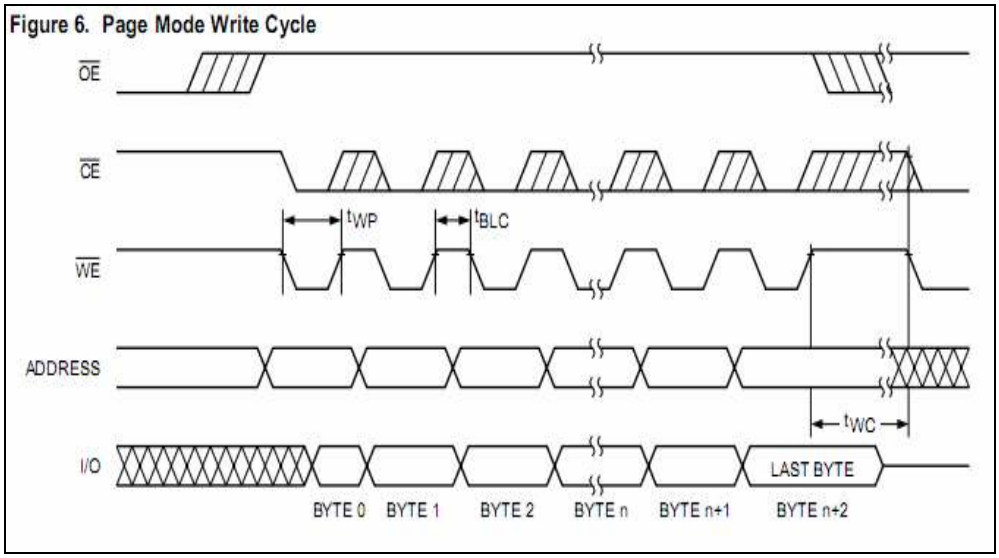
时间为 5ms。

图 6.2-5　CAT28C64B 芯片字节写时序图



（2）页写。页写的时序如图 6.2-6。该方式允许在一个页写周期内写入 1 到 32 个字节的数据。因此，字节写是页写的特例。和字节写一样，一个页写周期发生的条件也是$\overline{OE}$=H，$\overline{CE}$和$\overline{WE}$=L。开始条件可用$\overline{CE}$或$\overline{WE}$控制，取决于哪个最后出现。和字节写不同，页写由一串$\overline{WE}$脉冲组成，每个待写的字节的地址和数据被存入 32 字节的缓冲区。页地址由 A_5~A_{12} 决定，页内地址由 A_0~A_4 决定。在页写的过程中，A_5~A_{12} 的编码数据保持稳定。在待写的字节全部写入到 32 字节的缓冲区后，EEPROM 开始内部擦除和写入过程，最大时间和字节写一样，也是 5ms。

图 6.2-6　CAT28C64B 芯片页节写时序图



EEPROM 芯片的读操作。若作为 ROM 器件使用，用 MOVC 指令读出；若作为 RAM 器件使用，则用 MOVX 指令读出。

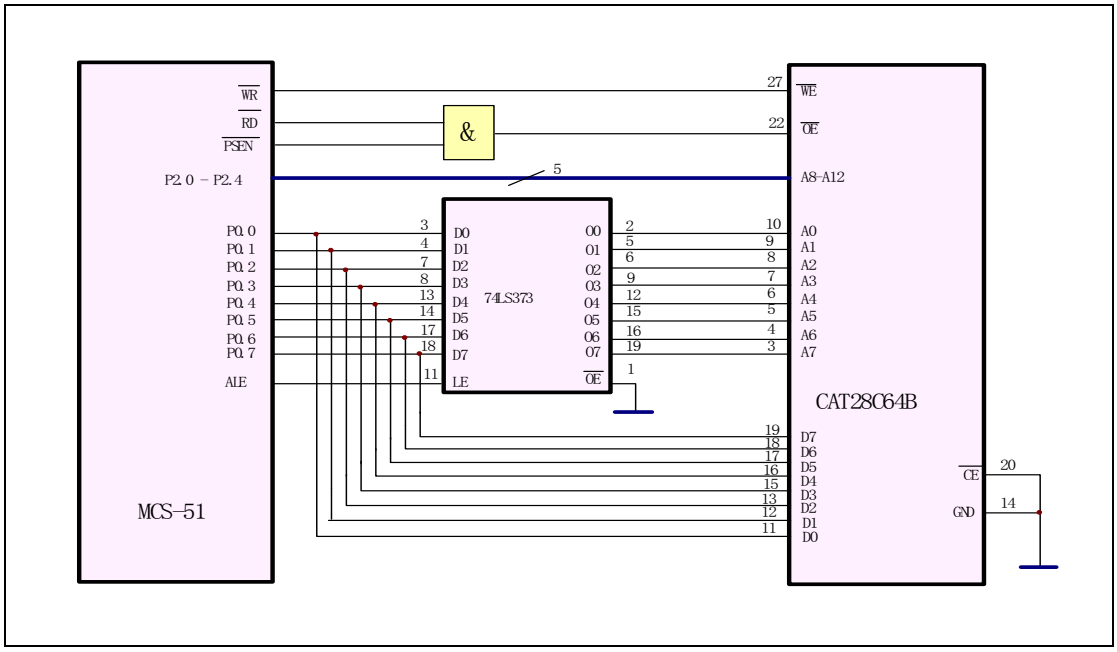
CAT28C64B 和 MCS-51 的连接见图 6.2-7。为了增加功能，图中的 2864 也能作为数据存储器使用，我们可以安排其中的某些单元作为 RAM 区。作为 RAM 使用时应注意执行写入指令的时间要求。图中：

数据线连接：28C64 的数据线接引脚 MCS-51 的 P0 端口；

地址线连接：28C64 为 8K 字节存储器，13 条地址线，接 MCS-51 A_0～ A_{12}，其中 A_0 ～ A_7 须经 373 锁存；设没有连接其它同类芯片，$\overline{CE}$ 端接地。

控制线连接：28C64 的 $\overline{WE}$ 接 $\overline{WR}$，$\overline{OE}$ 接 $\overline{RD}$ & $\overline{PSEN}$，用 MOVX 和 MOVC 可读出不同存储区的 RAM 数据或 ROM 表格常数。改写指令用 MOVX @DPTR，A，实际应用中在写入多个字节时，应注意字节写和页内地址的概念和 32 字节数量的要求。

图 6.2-7　CAT28C64B 和 MCS-51 的连接图



6.2.3　Flash存储器的扩展

Flash 存储器,又叫 PEROM (Programmable and Erasable Read Only Memory)存储器，也是一种可在线编程擦除写入的只读存储器件。通常作为程序存储器使用。Flash 的主要性能指标是：

（1）最大读取时间 150ns

（2）页编程时间 10ms

（3）擦写寿命 10，000 次

（4）数据保存时间 10 年

（5）功耗低，待机电流 300uA,工作电流 50mA

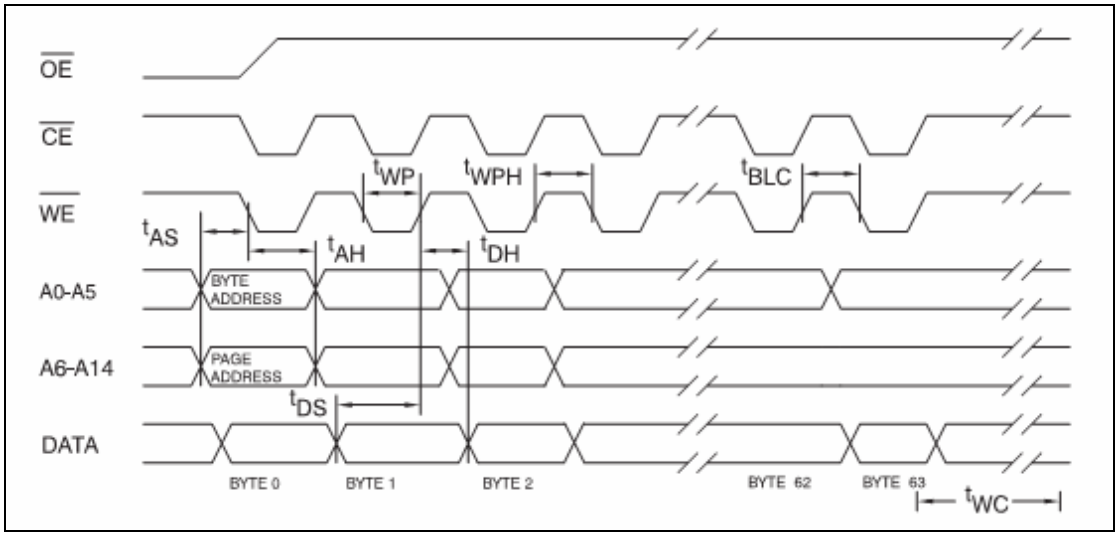
PEROM 器件的型号以数字“29”开头。和以“28”开头的 EEPROM 器件相比，Flash 存储器的最大特点在于它的成本低。市场上同类产品的价格要比 28 系列低 20%左右，因此近年来得到了广泛的应用。在性能上 PEROM 器件有不及 EEPROM 器件的地方，它的擦写寿命没有 EEPROM 长（100，000 次），写入时间不及 EEPROM 快（页写 5ms），最主要的是，PEROM 器件不能进行字节写的操作，只能进行页写操作，也就是说，要改写一个字节内容，需将该字节所在的页内存储单元都改写。尽管有这样的一些不及之处，但价格的低廉是 PEROM 器件广为流行的最大推力，取得了越来越多的市场份额。与此同时，许多单片机的生产厂家还把 PEROM 器件集成到单片机内部，替代原先的 EPROM 存储器，可以实现单片机程序的在线改写，大大方便了单片机的程序改写和固化过程。如 Atmel 公司的 AT89 系列单片机、Winbond 公司的 W77、W78 系列单片机、Philips 公司的 P89、P87 系列单片机、SST 公司的 ST89C、ST89F 单片机等，都用 PEROM 作为片内程序存储器。

现以 Atmel 公司的 32K 字节的 PEROM 芯片 AT29C256 为例（图 6.2-8），说明 PEROM 芯片的编程原理和与 MCS-51 的连接方式。

AT29C256 的数据写入（编程）是以页为基本单位的，每页的字节数是 64 个，页的基本地址由 A_6~A_{14} 决定，页内的地址由 A_0~A_5 决定。在页写的过程中，页基本地址 A_6~A_{14} 是不能改变的，若要该写页内的某一个字节的内容，则必须把该页内其它所有的 63 个字节的内容按原样重新装入，也就是必须安排该这些字节的写指令（MOVX @DPTR，A）操作。否则未安排重写的字节在页写之后的数据将是随机的。图 6.2-9 是 AT29C256 的页写编程时序。

图 6.2-8 AT29C256 外形

图 6.2-8 AT29C256 页写编程时序图



由图可见，在引脚 $\overline{OE}$=H，引脚 $\overline{CE}$ 和 $\overline{WE}$ 都等于 L，且在两者中最后出现的下降沿时刻，A_6~A_{14} 引脚的地址编码数据被芯片获取，随后，在 $\overline{WE}$ (或 $\overline{CE}$)出现的脉冲串的下降沿，依次将页内的 0~63 个字节的数据，按照 A_0~A_5 的地址装入缓冲区。一个字节的数据被装入后，$\overline{WE}$ 的下降沿应在 150us 内出现，进行下一字节的装入。如果 150us 内没有出现下降沿，则外部数据装入过程结束，进入内部编程的过程，内部编程时间 t_{WC} 最大不超过 10ms。

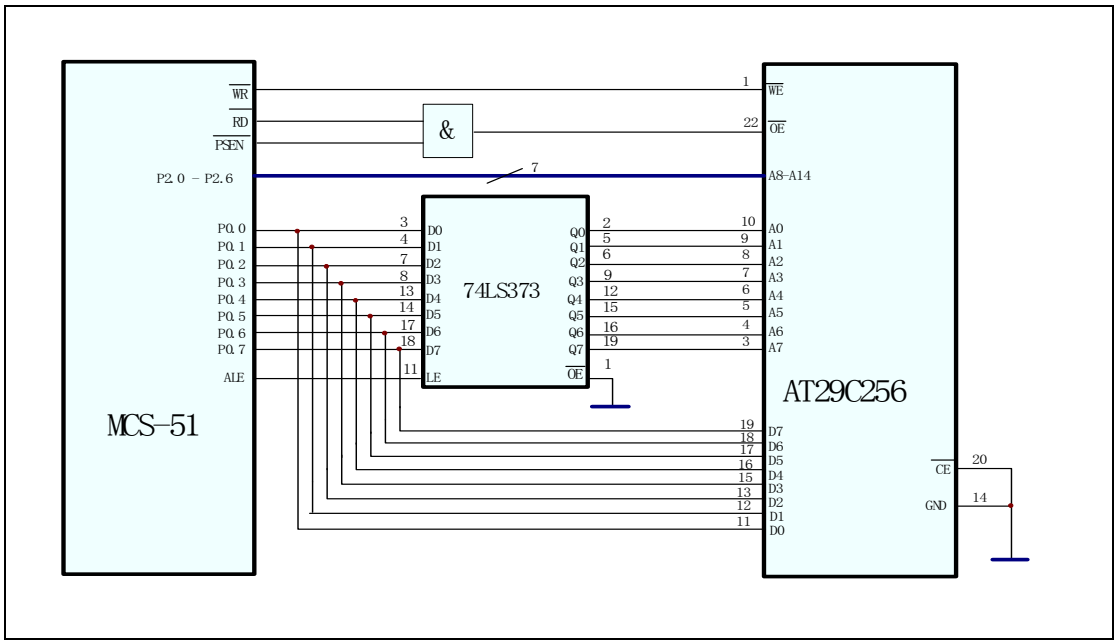
对 AT29C256 的控制，在写数据时可将 $\overline{CE}$ 和 $\overline{WE}$ 并联作为一个控制引脚连接到 MCS-51 的 $\overline{WR}$，或将两者中的任一个接固定低电平，另一个接 MCS-51 的 $\overline{WR}$；在读数时，$\overline{OE}$ 端接读数据控制信号。AT29C256 和 MCS-51 的连接见图 6.2-9。

数据线连接：29C256 的数据线接引脚 MCS-51 的 P0 端口；

地址线连接：29C256 为 32K 字节存储器，15 条地址线，接 MCS-51 A_0 ～A_{15}，其中 A_0 ～A_7 须经 373 锁存；设没有其它同类芯片，$\overline{CE}$ 端接地。

控制线连接：29C256 的 $\overline{WE}$ 接 MCS-51 $\overline{WR}$，$\overline{OE}$ 接 $\overline{RD}$ & $\overline{PSEN}$，在 $\overline{RD}$ 或 $\overline{PSEN}$ 任一个信号有效时，均可读出存储器数据，即 29C256 可作为 ROM 和 RAM 两种存储器使用。在不同的存储区间放置随机数据和固定数据，用 MOVX 和 MOVC 指令可读出不同存储区的 RAM 数据或 ROM 表格常数。改写 ROM 数据用 MOVX @DPTR，A 指令，实际应用中，应注意字页内地址的概念和 64 字节数量的要求。

图 6.2-9　AT29C256 和 MSC-51 的连接



6.3　并行I/O端口扩展

MCS-51 的 4 个并行 I/O 端口 P0~P3；在扩展外部存储器或其他芯片后，P0 和 P2 口作为数据和地址总线用使用，P3 口的一些位作为控制总线用，此时可作为完整 8 位并行 I/O 端口使用的只有 P1。在实际应用系统有更多并行端口需求的情况下，需要进行并行 I/O 端口的扩展。

MCS-51 将片外的并行 I/O 接口地址和片外的存储器统一编址，就是把片外的 I/O 端口看成是片外数据存储器的单元。因此片外的 I/O 接口的扩展方法和片外的数据存储器完全相同，即两者的读写操作时序一致、三总线连接方法相同。

I/O 的扩展可分为简单扩展和可编程扩展两种方式。

6.3.1　并行I/O端口的简单扩展

并行 I/O 端口，是在 MCS-51 和外部 IC 芯片之间，以并行传送 8 位数据的方式，实现输入/输出操作的端口。和存储单元一样，I/O 端口有自己的编码地址线、读写控制线和数据线。并行 I/O 端口可细分为输入端口（I）、输出端口（O）和双向端口（I/O）。其中，输入端口(I)由外部 IC 芯片向 MCS-51 输入数据，用指令“MOVX A，@Ri 或 MOVX A,@DPTR”操作；输出端口（O）由 MCS-51 向外部 IC 芯片输出数据，用指令“MOVX　@Ri，A 或 MOVX @DPTR，A”操作。输入（I）端口无需锁存功能，输出（O）端口一般情况需要锁存功能；输入（I）端口无需驱动功能，输入（O）端口视具体情况，可添加驱动功能。双向端口具有 I 和 O 的双重功能，使用同一个编码地址。

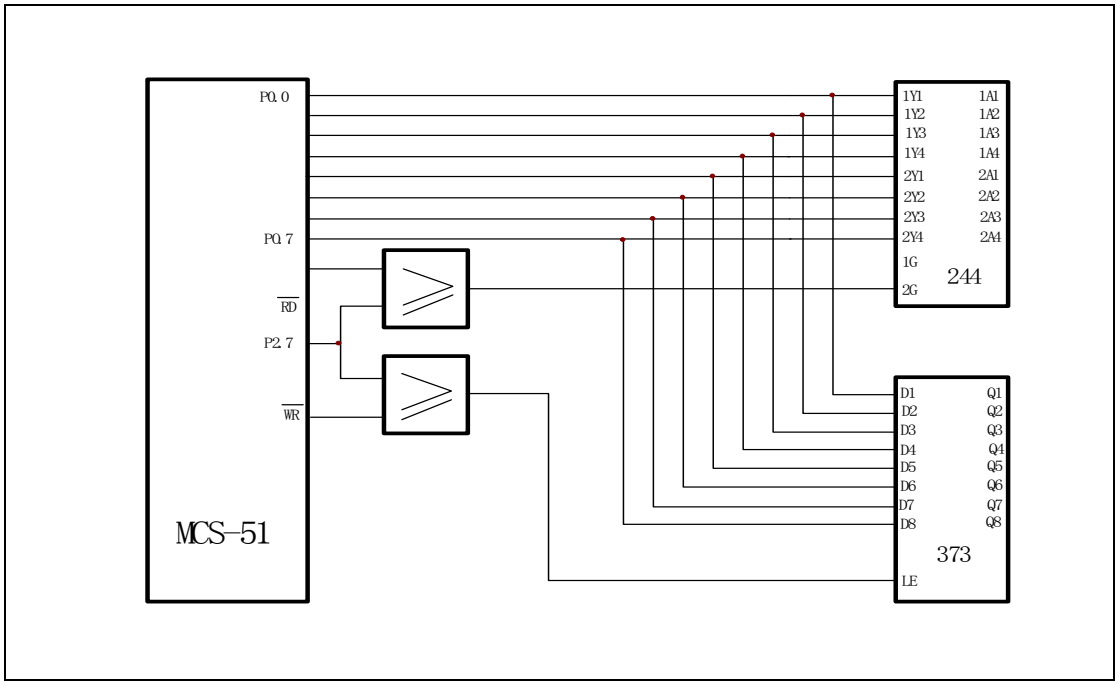
简单并行 I/O 端口，是指 MCS-51 可以随时用 MOVX 指令对它们进行读写操作。这些端口总是处于准备好接收 MCS-51 读写指令的状态，交换数据之前无需通信联络信号（问答信号）询问状态，其读写操作过程与外部数据存储单元完全相同。

扩展简单并行 I/O 端口，就是扩展可以用 MOVX 指令直接操作、并行输入输出数据、具有一定驱动能力和输出锁存功能的 I/O 单元。

扩展简单并行 I/O 端口的方式，通常采用普通的 TTL 电路和 CMOS 门电路实现，如八线驱动器 74LS244\40244、八双向总线发送\接收器 74LS245\40245、八 D 锁存器 74LS373\40373、八 D 触发器 74LS377\40377 等。

如图 6.3-1，是采用 74LS244 扩展的 8 位输入口和采用 74LS373 扩展的 8 位输出口。

图 6.3-1 简单并行 I/O 端口的扩展



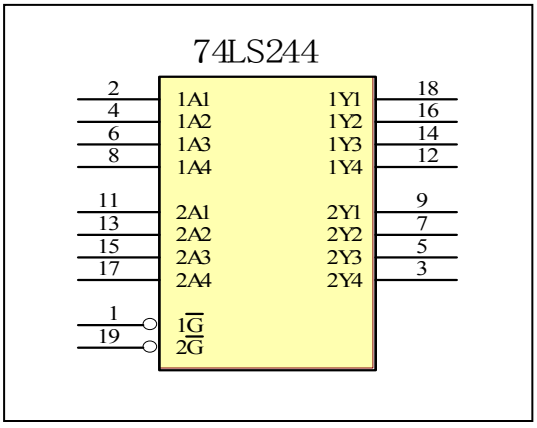
图中 74LS244 是八位线驱动器，20 引脚，引脚及功能表如图 6.3-2。

74LS244 内部由 2 组 4 位三态缓冲器组成，一组输入端 $1A_1$~$1A_4$，输出端 $1Y_1$~$1Y_4$，控制端 $1\overline{G}$；另一组输入端 $2A_1$~$2A_4$，输出端 $2Y_1$~$2Y_4$，控制端 $2\overline{G}$ 。由功能表见，当控制端 G=L 时，输出端 Y=A，即输出和输入联通；当控制端 G=H 时，输出端 Y=Z，即输出端是高阻状态。我们对 G 端进行控制，即可对 244 的输入端 $1A_1$~$1A_4$ 和 $2A_1$~$2A_4$ 的数据进行读入操作。由此例道理可推及一般，即凡作为并行输入（I）端口连接到数据总线的器件，必须具备三态门的功能。

74LS373 是八 D 触发器，作为地址锁存器已在前面介绍过其性能。

图中 74LS244 的输入控制引脚 $1\overline{G}$ $2\overline{G}$，由 $\overline{RD}$ 和 $P_{2.7}$ 相“或”控制，在执行外部数据访问指令 MOVX A, @DPTR 或 MOVX A,@R_i（$P_{2.7}$=0）时，$P_{2.7}$ 和 $\overline{RD}$ 均为低电平，$1\overline{G}$ $2\overline{G}$ 被

图 6.3-2　74LS244 引脚示意图及及真值表



244 真值表

输入		输出
$\overline{G}$	A	Y
L	L	L
L	H	H
H	X	Z

选通，$1A_1$~$1A_4$ 和 $2A_1$~$2A_4$ 的数据通过 $1Y_1$~$1Y_4$ 和 $2Y_1$~$2Y_4$ 被送到数据总线， MCS-51 从 P0 口读入数据。

74LS377 的输出控制信号 LE 引脚由 P2.7 和 $\overline{WR}$ 相“或”控制，在执行外部数据访问指令 MOVX @DPTR， A 或 MOVX @R_i ，A（$P_{2.7}$=0）时，$P_{2.7}$ 和 $\overline{WR}$ 均为低电平，将数据 D_0 ~D_7 送到 Q_0~Q_7，由于 D 触发器的锁存功能，该输出数据被保持住，直到下一个新的输出数据的到来。

该电路从 244 读入数据并从 377 输出数据的程序如下：

```
MOV     DPTR，#7FFFH        ；DPTR 为地址指针（P2.7=0）
MOVX    A，@DPTR            ；244 数据读入
MOVX    @DPTR，A            ；377 输出数据
SJMP    $
```

从此例看到：（1）并行输入（I）端口，输入数据在执行读外部数据存储器指令时被读入 CPU，故该端口不需要锁存功能；并行输出（O）的端口，输出数据在执行写外部数据存储器指令时由 P0 口送出，指令过后数据随即消失，故该端口须需要锁存功能。（2）输入和输出端口使用同一个编码地址。

6.3.2 可编程并行 I/0 端口芯片扩展

与并行 I/O 端口的简单扩展方式相对应的，是并行 I/O 端口的可编程芯片扩展方式。

MCS-51 在同外部设备的信息交换过程中，一般情况下外设的传送速度慢，CPU 需要与其联络，询问其工作的状态，然后才能在恰当的时机进行数据交换。这些询问联络工作若由 MCS-51 承担，会占用 CPU 大量的时间，影响处理能力。若有一种接口芯片，在 MCS-51 和外设之间作为“二传手”的角色，替 CPU 承担和外设的信息联络数据交换，并将处理结报告 MCS-51, 那么将会节省 CPU 大量的时间，提高单片机的工作效率。能够充当这种“二传手”角色的，就是可编程的 I/O 并行接口芯片。

可编程的 I/O 并行接口芯片自身具有多个并行 I/O 端口，这些端口能和外设以联络问答方式来交换数据。 MCS-51 通过对这些 IC 芯片的操作，也就实现了并行 I/O 的扩展。

可编程 I/O 并行接口芯片 8255(A)、8155 等，就是专为此等用途而制作的。

6.3.2.1 可编程并行接口芯片 8155

不言而喻，在满足应用系统要求的前提下，使用的外接器件越少越好。芯片数量的增加不仅会增加成本，也增加了功耗、噪声以及印制板的面积。可编程并行接口芯片 8155 除具有 I/O 扩展接口外，还有 RAM 和计数器，一个芯片而三种资源，增加了功能和用途。

1） 8155 芯片介绍

（1）8155 的性能、结构及引脚。

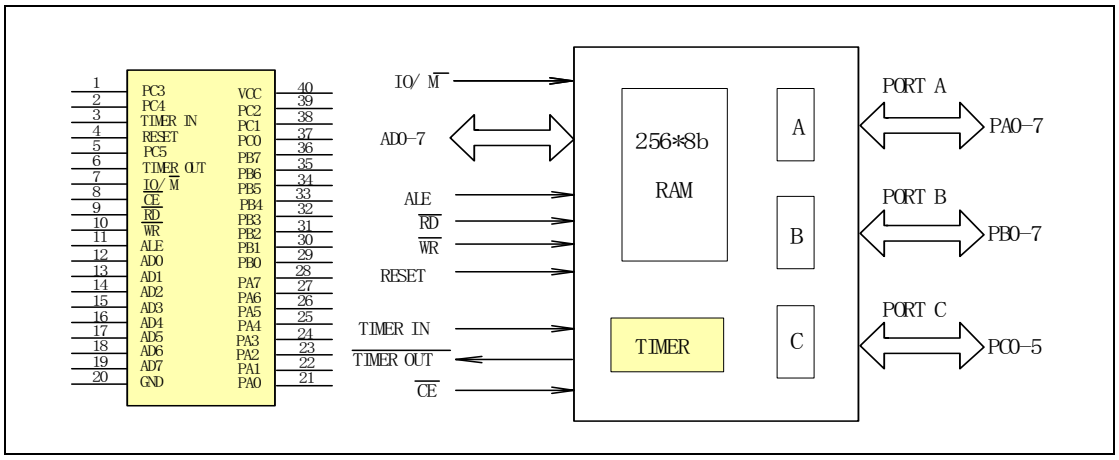
8155 是一种复合型的可编程 I/O 芯片,40 个引脚，含三种扩展功能：（1）三个 I/O 端口, A 口和 B 口各有 8 位 I/O,C 口有 6 位 I/O；（2）256 字节 RAM 存储区；（3）一 个 14 位减法计数器。内部结构和外部引脚示意如图 6.3-5。

引脚定义：

- AD_0～AD_7: 地址/数据复用线，连接到 MSC-51。8155 内部具有地址锁存器，在 ALE 的下降沿可锁存 MSC-51 的地址编码数据，因此无需外加 373 之类的锁存器件，AD_0～AD_7 可直接连 P0 口，接收编码地址和数据。
- I/O 口线： 外设 I/O 线，连接到外部 IC 芯片。PA0～PA7 和 PB0～PB7 分别是 A 口

和 B 口 I/O 线，PC0～PC5 是 C 口 I/O 线；PC 口也可作为 A、B 口的控制联络线。

6.3-5　8155 内部结构和外部引脚示意图



- ALE：地址锁存信号，连接到 MSC-51 的 ALE，锁存 P0 口的地址编码。因此，MCS-51 和 8155 连接时无需 373 芯片。
- $\overline{RD}$、 $\overline{WR}$ 、$\overline{CE}$：读、写、片选线。
- IO/ $\overline{M}$：功能选择信号。用于区分 8155 的内部功能，由 MSC-51 控制。该引脚=1，选择 8155 的 I/O 功能；该信号=0，选择内部 RAM 和定时器功能。
- TIMER IN：计数时钟脉冲输入，1 个脉冲使 8155 的 14 位定时器当前值减 1。
- $\overline{TIMEROUT}$：计数器回 0（计数满或定时到）输出信号，输出波形由对计数器设定的工作方式所决定。
- RESET：复位引脚。A、B、C 三个端口复位成输入方式。

（2）8155 的地址分配

8155 的地址分配如表 6.3-2 所示。当 IO/ $\overline{M}$=1 时，三个 I/O 端口的地址由 AD0～AD2 决定，当 IO/ $\overline{M}$ =0 时，片内的 RAM 区地址由 AD0～AD7 决定。

表 6.3-2　8155 的地址分配

IO/ $\overline{M}$	AD2	AD1	AD0	内容
1	0	0	0	命令/状态口
	0	0	1	PA
	0	1	0	PB
	0	1	1	PC
	1	0	0	TIMER 低 8 位
	1	0	1	TIMER 高 8 位
0 (存储器)	AD7	～	AD0	
	00	～	FFH	内部 RAM 区

（3）8155 的命令控制字

MCS-51 对 8155 的操作要通过对 8155 的命令口写控制字实现。控制字各位定义见表 6.3-3。

6.3-3　8155 命令控制字定义

数据位	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
位定义	TM_2	TM_1	IEB	IEA	PC2	PC1	PB	PA

表中：
- PB、PA　选择 B 口和 A 口 I/O: “0”为输入，“1”为输出
- PC2 PC1　4 种方式：

PC2 PC1= 00　A 口 B 口基本 I/O，C 口输入

PC2 PC1= 11　A 口 B 口基本 I/O，C 口输出

PC2 PC1= 01　A 口选通 I/O、B 口基本 I/O，C0～C2　A 口联络信号，C3～C5 输出

PC2 PC1= 10　A 口 B 口选通 I/O，C0～C2　A 口联络信号，

C3～C5　B 口联络信号

- IEA　0：禁止 A 口中断，1：允许 A 口中断
- IEB　0：禁止 B 口中断，1：允许 B 口中断
- TM_2　TM_1　定时器工作方式：

　　TM_2　TM_1= 00　空操作

　　TM_2　TM_1= 01　停止定时器工作

　　TM_2　TM_1= 10　若定时器正在计数，则计数值减为 0 时停止工作

　　TM_2　TM_1= 11　启动定时器计数

（4）8155 的状态字

MCS-51 对 8155 的运行情况可通过读状态字来查询。状态字和命令字用同一地址，各位定义见表 6.3-4。

表 6.3-4 8155 状态字定义（“1”有效）

数据位	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
位定义	×	TIMER	$INTE_B$	BF_B	$INTR_B$	$INTE_A$	BF_A	$INTR_A$

表中：
- TIMER　计数器状态：“1”计数满，“0”计数未满
- $INTE_B$　B 口中断允许标志
- BF_B　B 口缓冲器满空标志
- $INTR_B$　B 口中断请求标志
- $INTE_A$　A 口中断允许标志
- BF_A　A 口缓冲器满空标志
- $INTR_A$　A 口中断请求标志

2）8155 芯片的使用方法

（1）8155 的 I/O 操作。

当$\overline{CE}$=0，IO/ $\overline{M}$=1 时，用命令控制字的 D_3D_2 的不同编码，可设置 8155 的三个端口的工作方式。A、B 端口可工作于基本 I/O 方式和选通 I/O 方式，C 口只能工作于基本 I/O 方式。在 A、B 端口工作于选通 I/O 方式时，C 口用来作为它们的联络线。工作方式及联络线定义见表 6.3-5。

联络线含义如下：

- $INTR_{(AB)}$：中断请求，输出。高电平有效。当 A 口 B 口输出数据到外设或外设输入数据进入 A 口 B 口后，该信号有效，反相后可作为 MCS-51 的中断请求。中断响应后信号自动撤销。
- $BF_{(AB)}$：I/O 缓冲器满（空）信号，输出。高电平有效。提供给外设，作为应答信号。
- $\overline{STB}_{(AB)}$：选通信号，输入。输入数据时为外设将数据送入 A 口 B 口的锁存信号，输出数据时为外设取走数据的应答信号。

（2）8155 的 RAM 区操作。

当$\overline{CE}$=0，IO/$\overline{M}$=0 时，可读写 8155 的 256 个 RAM 单元内容。寻址编码由$\overline{CE}$、IO/$\overline{M}$和 AD_0～AD_7 引脚共同决定。使用 MOVX 指令读写。

（3）8155 的计数器操作。

8155 内部的 14 位减法计数器可对输入脉冲进行减法计数。外部脉冲从 8155 的 TIMER IN 引脚（3）输入，计数器计满溢出后从 TIMER OUT 引脚（6）输出信号，信号的波形可编程设定。使用 8155 的计数器时，分两步进行：

① 将输出方式和初始值写入计数器的低 8 位和高 8 位，这 2 个字节的格式如表 6.3-5。其中 M_2M_1 用来设置定时器/计数器的输出方式，也就设置是在$\overline{TIMEROUT}$ 引脚输出脉冲或

方波的形式，见表 6.3-6。

② 对命令口写控制字，启动或停止计数器工作。

表 6.3-5 8155 I/O 口工作方式

D3 D2 / 端口	00	11	01	10
	方式 0	方式 1	方式 2	方式 3
A 口	基本 I/O	基本 I/O	选通 I/O	选通 I/O
B 口	基本 I/O	基本 I/O	基本 I/O	选通 I/O
PC0	输入	输出	$INTR_A$	$INTR_A$
PC1	输入	输出	BF_A	BF_A
PC2	输入	输出	$\overline{STB}_A$	$\overline{STB}_A$
PC3	输入	输出	输出	$INTR_B$
PC4	输入	输出	输出	BF_B
PC5	输入	输出	输出	$\overline{STB}_B$

3）MCS-51 和 8155 的连接

数据线连接：8155 的 AD0～AD7 接 MCS-51 的 P0 口；

地址线连接：8155 的 AD0～AD7 接 MCS-51 的 P0 口无需经过锁存器；$\overline{CE}$ 和 IO/$\overline{M}$可接 P2 口高位地址线，也可接其它 I/O 口线；

控制线连接：8155 的 ALE、$\overline{WR}$、$\overline{RD}$ 接 MCS-51 的对应控制线。

表 6.3-5　8155 计数器 2 字节格式

计数器高 8 位（$A_2A_1A_0$=100）			计数器低 8 位（$A_2A_1A_0$=101）
	M_2 M_1	计数器高 6 位数值	计数器低 8 位数值

表 6.3-6　8155 计数器输出方式设置

M_2 M_1	输出方式	输出波形
0 0	一个计数周期输出单次方波	
0 1	连续方波	
1 0	计满回 0 输出单个脉冲	
1 1	连续脉冲	

MCS-51 和 8155 的连接示意如图 6.3-7。由图可见，片选地址 P2.7=1, IO/$\overline{M}$选择地址=P2.6，各部分的地址（部分译码）是：

- 命令/状态寄存器：　F000H
- A 口：　F001H
- B 口：　F002H
- C 口：　F003H
- TIMER 低 8 位：　F004H
- TIMER 高 8 位：　F005H
- RAM 区：　B000H～B0FFH

举例：如图 6.3-7，要求：

（1）A 口输出,B 口输入，读入 B 口的开关状态，然后通过 A 口输出到发光二极管显示。

（2）从 TIMER IN 引脚输入脉冲串，经 1000 分频后，由 $\overline{\text{TIMEROUT}}$ 引脚输出脉冲串。

分析：

（1）设定计数器初值和计数器输出方式。初始值 1000，16 进制数为 03E8H；计数器输出为连续方波，M_2 M_1= 01,则计数器高位=43H,低位=E8H。

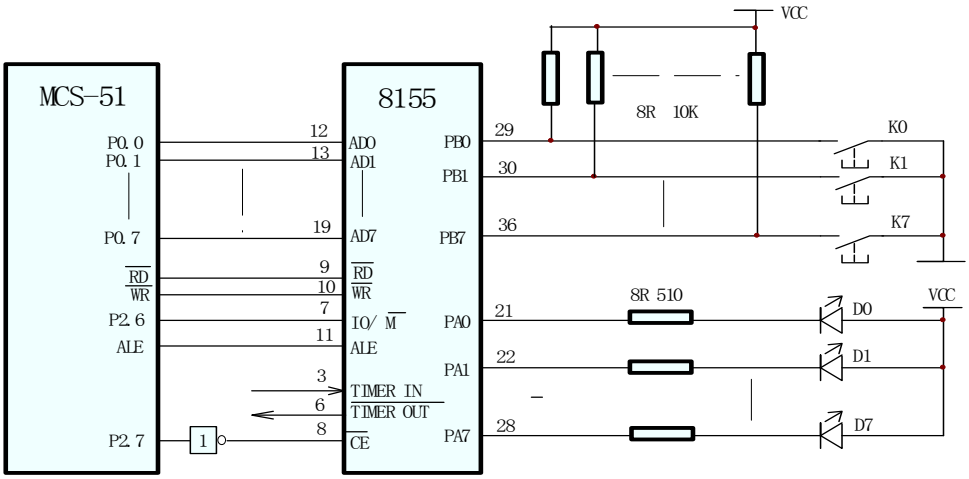
（2）设定命令控制字。列表分析，命令控制字=C1H

定时器控制 启动	中断控制 禁止	工作方式 0	B 口输入	A 口 输出
D_7D_6= 11	D_5D_4 = 00	D_3D_2 = 00	D_1= 0	D_0 = 1

汇编源程序如下：

```
PA      EQU     0F001H   ；A 口地址
PB      EQU     0F002H   ；B 口地址
PC      EQU     0F003H   ；C 口地址
CM      EQU     0F000H   ；命令口地址
TL      EQU     0F004H   ；计数器低字节地址
TH      EQU     0F005H   ；计数器高字节地址
```

图 6.3-7　MCS-51 和 8155 的连接示意图




```
        ORG     0100H            ；程序开始地址
SRART:  MOV     A,#0E8H          ；计数器低字节地始值
        MOV     DPTR,#TL         ；计数器低字节地址
        MOVX    @DPTR,A          ；写计数器低字节地址初始值
        MOV     A,#03H           ；计数器高字节初始值
        MOV     DPTR,#TH         ；计数器高字节地址
        MOVX    @DPTR,A          ；写计数器低字节地址初始值
        MOV     A,#0C1H          ；命令字
        MOV     DPTR,#CM         ；命令口地址
        MOVX    @DPTR,A          ；写命令字
LOOP:   MOV     DPTR,#PB         ；B 口地址
        MOVX    A,@DPTR          ；B 口开关状态读入
        MOV     DPTR,#PA         ；A 口地址
```

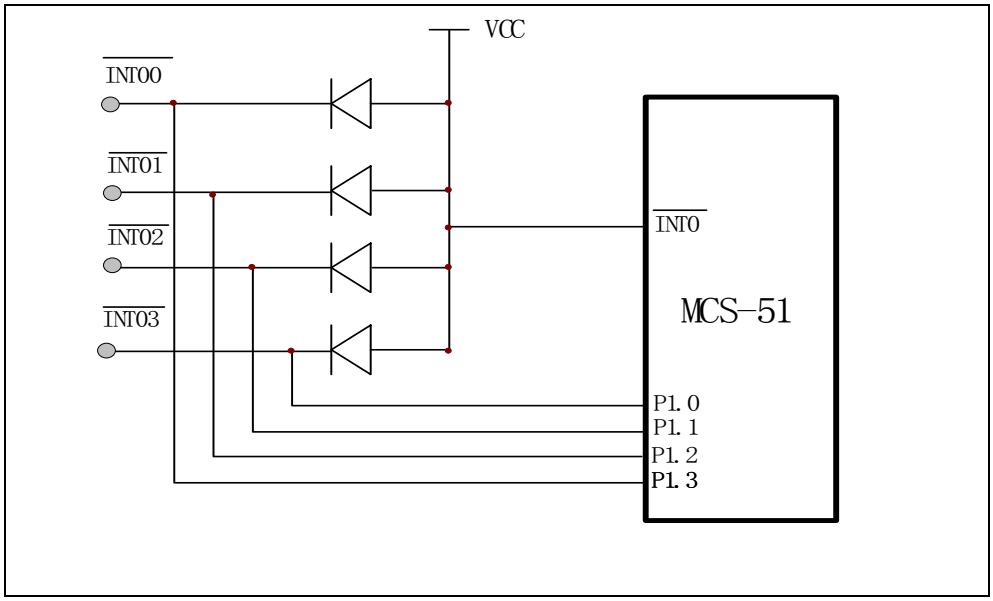
```
        MOVX    @DPTR,A          ；B 口开关状态从 A 口输出
        ACALL   DELAY            ；延时
        AJMP    $                ；
DELAY:  MOV     R6,#0            ；延时子程序
D1:     MOV     R7,#0
        DJNZ    R7,$
        DJNZ    R6,DI
        RET
        END
```

6.4 中断源扩展（*）

基本型的 MCS-51 单片机只有两个外中断源。当实际使用中需要更多的外部中断时，则须采用中断源的扩展技术来满足要求。中断源扩展的基本原理是，通过复用 MCS-51 基本中断源（一级中断），来扩展若干个二级中断，在 MCS-51 响应一级中断后，再找出二级中断来源并对其进行服务。

如图 6.3-8，欲扩展 $\overline{\text{INT10}}$、$\overline{\text{INT11}}$、$\overline{\text{INT12}}$、$\overline{\text{INT13}}$ 四个二级中断，通过与逻辑电路，形成对一级中断 $\overline{\text{INT1}}$ 的申请。当某个二级中断有申请（低电平）时，$\overline{\text{INT1}}$（一级中断）有效，单片机进入 $\overline{\text{INT0}}$ 中断入口，在中断服务程序中，读入 P1 口内容，判别二级中断来源并进行处理。

图 6.3-8　门电路扩展中断源示意图



汇编程序如下：

```
        ORG     0003H           ;/INT0 中断矢量入口
        SJMP    INT2            ;转入中断源判断程序
        ……
INT2:   MOV     P1,#0FH         ；先写“1”
        JNB     P1.0,INT00      ；是 INT00 申请，转至相应中断服务子程序
        JNB     P1.1,INT01      ；是 INT01 申请，转至相应中断服务子程序
        JNB     P1.2,INT02      ；是 INT02 申请，转至相应中断服务子程序
```

```
        ……
        RETI
INT00:  ……                          ;INT00 中断服务子程序
        RETI
INT01:  ……                          ;INT01 中断服务子程序
        RETI
INT02:  ……                          ;INT02 中断服务子程序
        RETI
```

在扩展中断源数量较多时，可采用编码器电路实现，如 8-3 优先编码器 74LS148、10-4 优先编码器 74LS147 等。74LS148 的引脚和功能图见图 6.3-9。

用 74LS148 构成的 8 个二级中断源扩展电路见图 6.3-10。外部中断源 INT00、INT01 … INT07 作为二级中断源分别接 148 的输入引脚 0、1 … 7 ，148 的$\overline{GS}$输出引脚接 MCS-51 的$\overline{INT0}$。当二级中断源有中断申请时，$\overline{GS}$引脚电平变低，对 MCS-51 产生中断申请，并按预定的优先顺序在 148 的 A0～A2 引脚出现相应的编码输出，MCS-51 通过 P1 口读入该编码数据，即可判断并响应相应的二级中断。

图 6.3-9 8—3 优先编码器 148 的功能表、引脚图

INPUTS								OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

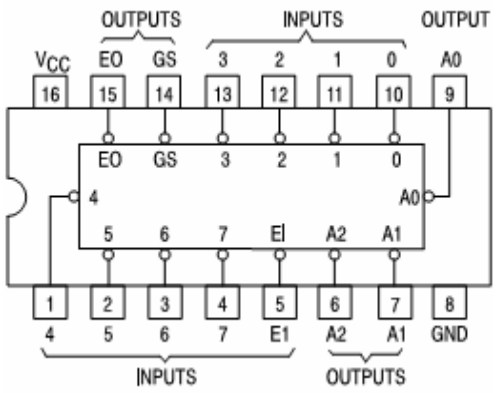
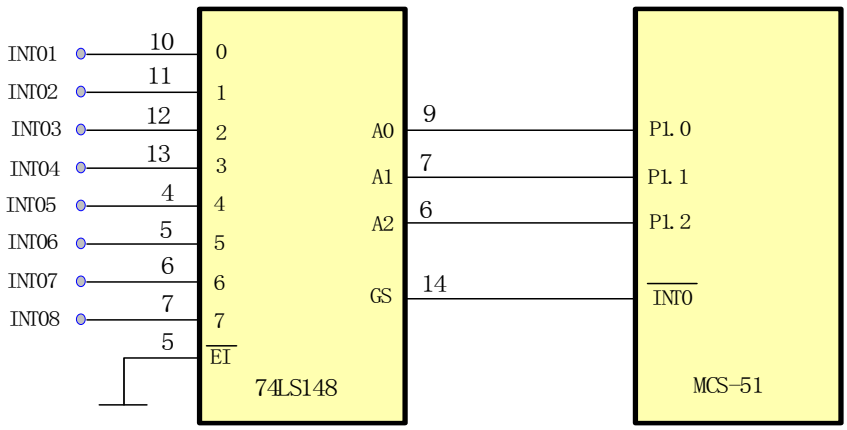


图 6.3-10　74LS148 和 MCS-51 的连接



本章小结

（1）扩展技术是组成 MCS-51 应用系统的基本手段。在 MCS-51 的片内资源不够用、以及需要增加外围功能器件的时候，都要用到扩展技术。

扩展技术的基本内容就是以单片机为核心，以三种线为接口，连接相关外围集成电路（IC）芯片而形成的符合 MCS-51 指令时序要求的应用系统。

MCS-51 的应用系统中，单片机对外围 IC 器件的测量和控制，其实质是通过数据交换实现的。

扩展技术的基础是 MCS-51 的三总线系统。理解并正确使用三总线是实现扩展的基础。其中包括三总线硬件正确连接和三总线软件时序的正确理解。

（2）MCS-51 应用系统所扩展的所有外部 IC 器件，按照“哈佛结构”被排列成两大系列：程序存储器系列和数据存储器系列。两大系列各独立编址，地址空间从 0000H 到 FFFFH。I/O 器件被排列在数据存储器系列内。三总线的硬件端口区分了 64K 的地址并送出了控制信号；软件指令中的控制时序区分了两个系列。我们用 MOVC（$\overline{\text{PSEN}}$）指令来操作程序存储器，用 MOVX（$\overline{\text{RD}}$、$\overline{\text{WR}}$）指令来操作数据存储器。

（3）三总线是连接 MCS-51 和外部 IC 器件的桥梁。其中地址总线的连接是扩展技术中的重点。对外部 IC 芯片存储单元的选择，要涉及“片选”和“字选”，字选通常由低位地址线承担，片选通常由字选余下的高位地址线承担。片选的原则是：使用相同控制信号的 IC 芯片， 片选地址不能相同；使用相同片选地址的 IC 芯片，各自的控制信号不能相同。

（4）EPROM、EEPROM 和 Flash 都可用作程序存储器。EEPROM 和 Flash 的主要优点是可以在线重写。EEPROM 和 Flash 也可作数据存储器使用，但应注意写入的速度比 SRAM 慢得多。

（5）8255、8155 是可编程的 I/O 扩展芯片。使用它们不仅可以扩展 I/O、RAM、计数器，而且可以和慢速的外围设备通过联络通信的方式交换数据。

（6）SRAM、EPROM、EEPROM 和 Flash 芯片的容量越做越大，市场上已经很难买到 8K 以下的存储器芯片。使用大容量的存储器比使用多个小容量的节省成本、节省连线。若测控条件允许，尽可能将 RAM 和 ROM 合并在一个芯片上使用，这更加节省。

思考题与习题

6.1 试叙述单片机应用系统扩展技术的基本内容、基本原理和基本方法。

6.2 在应用系统的外接器件中，作为 I/O 接口的芯片（如 A/D、8155、8255 等），在哪个存储器地址空间被编址？MCS-51 是如何在硬件和软件两方面区分程序存储器地址空间和数据存储器地址空间的？

6.3 试分别叙述 MCS-51 三总线的引脚、作用和时序。

6.4 试叙述 EEPROM 器件的主要特点和参数。

6.5 试叙述 Flash 器件的主要特点和参数。

6.6 MCS-51 扩展地址外部存储器时，为什么 P0 口的低 8 位地址需要锁存，而 P2 口高 8 位地址却不需要锁存？在锁存过程中，锁存信号是什么？在锁存信号的什么位置发生？

6.7 在 MCS-51 上扩展 1 片 EEPROM 28C16（2K×8 位）和 1 片 SRAM 6264（8K×8 位）。要求 28C16 既能做程序存储器，又能做数据存储器。

6.8 在 MCS-51 上扩展 1 片 EEPROM 28C16（2K×8 位）、1 片 74LS244 和 1 片 74LS373，要求 244 的地址为 7FFFH,373 的地址为 0BFFFH。

6.9 8155 的 TIMER IN 引脚输入脉冲频率为 100KHz，编写程序，使 TIMER OUT 输出脉冲序列，频率为 1KHZ。

第 7 章　MCS-51 单片机串行 I/O 总线扩展技术

我们知道，MCS-51 应用系统的 IC 芯片都是安装在印制电路板上面的。IC 芯片之间需要进行数据交换，数据交换使用的连接线越少，芯片的体积和印制电路板的面积也会越小。这不仅会带来诸如节省安装空间、减少装配工序、缩小产品体积、降低设备成本等一系列优点，也会使系统在便于调试维护、抗御电磁干扰等方面的内在品质得到提升。

与并行总线 I/O 扩展技术相比，串行 I/O 总线扩展技术使用的连接线要少得多。进几年串行通信技术在 MCU 的外围 IC 芯片领域发展迅速，诸如存储器、A/D、D/A、V/F、时钟、测温、LCD 驱动、接触式 IC 卡等所有的接口器件领域内，都推出了串行通信芯片。

本章讲述的串行 I/O 总线扩展技术，所涉及的内容是 MCU 和外围串行芯片之间的通信技术，是 MCU 串行 I/O 数据交换接口和功能的扩展。串行 I/O 总线扩展技术和本教材第 5 章介绍的 MCS-51 串行口(UART)技术的重要区别在于：

（1）MCS-51 串行口(UART)用于 MCU 之间的通信，即 MCU 无法和不具备 UART 的 IC 芯片交换数据；而本章内容则是 MCU 与外围串行 IC 芯片之间的通信技术；

（2）MCS-51 串行口(UART)一般用于不同设备之间或同一设备但不同部件之间的通信，其显著特征是通信的距离较远；而本章内容则是在同一块印制板内 MCU 和 IC 芯片之间的通信，其通信的直线距离在几个～几十个厘米之内。

因此，本章串行 I/O 总线扩展技术是特指在同一块印制板上、MCS-51 和外围的串行 IC 芯片之间的数据交换技术。该技术共包含三种总线方式：SPI(Serial Peripheral Interface)总线、I^2C (IC TO IC BUS)总线和 One-Wire 总线。

在串行通信 IC 芯片迅速发展的同时，MCU 自身也在适应这些技术的发展，一些单片机如 C8051F 系列、8XC552、8XC652 等内部就集成了 SPI 或 I^2C 的总线接口。

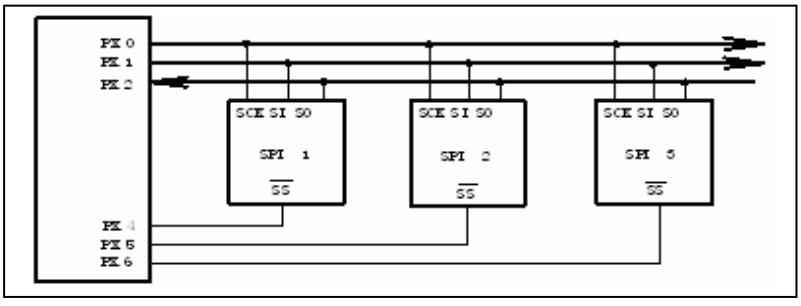
尽管 MCS-51 系列的多数 MCU 内部没有集成这些总线接口，但 MCS-51 可以很方便地通过编程来模拟 I/O 端口的电平时序，使之符合这些总线规定的通信协议，从而构成各种所需的应用系统。

本节将介绍 SPI、 I^2C 和 One-Wire 三种串行总线的原理及应用。

7.1　SPI串行接口总线技术

串行外设接口总线(Serial Peripheral Interface,SPI) 是 Motorola 公司推出的一种同步串行总线技术。现已广泛用于 EEPROM、A/D、D/A、移位寄存器、显示驱动器等多种 IC 器件。在 MCS-51 应用系统中，典型的 SPI 总线应用是单主多从形式，即一个主机、多个从机组成系统。如图 7.1-1 所示，MCS-51 作为主机和 3 个从机芯片进行数据交换。

图 7.1-1　SPI 串行总线系统示意图



7.1.1 SPI 串行总线协议

SPI 总线系统中，IC 芯片之间的数据交换是由主机控制的，数据交换的起停、节拍和方向均由主机发出相应的控制时序信号。数据传送的单位是字节（帧），先传送字节的最高位，依次到字节的最低位。最高传送速率 1.05Mbps。

SPI 器件使用 4 条线和单片机交换数据，这 4 条线的定义是：

（1）SCK —— 同步时钟信号线，简称 SK

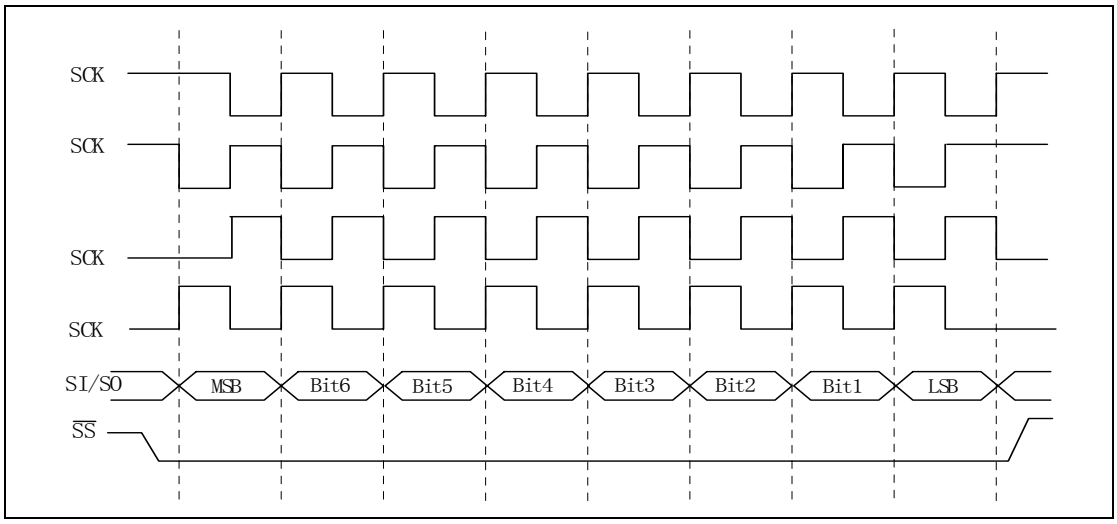
（2）MOSI —— 主机输出/从机输入数据线，简称 SI

（3）MISO —— 主机输入/从机输出数据线，简称 SO

（4）$\overline{SS}$ —— 芯片有效选择信号线，用来选择和主机通信的芯片。

其中 SK、SI、SO 为三条总线，所以也有资料称其为三线式总线。SPI 系统数据交换的时序信号定义见图 7.1-2.

图 7.1-2 SPI 总线数据/时钟时序示意图



图中，在/SS 信号有效后，SPI 芯片被选中，数据传送在 MOSI/MISO 总线上进行， 在 SCK 的控制下，从高位至低位逐位传送。8 位送完，$\overline{SS}$信号复位，传送结束。

需要注意的是，对应不同的芯片，SCK 信号的触发方式不同，图 7.1-2 中列出了 4 种形式，选用时需查阅具体 SPI 芯片的使用说明。另外，若使用的 SPI 芯片只需单向的数据传送，可省去不用的 SI 或 SO；若只有单个芯片，可将$\overline{SS}$引脚接地，此时 SPI 和 MCS-51 的连接只需 2 条线。

MCS-51 系列中，多数单片机内部没有 SPI 接口，我们可以用软件控制 I/O 引脚的电平时序，来模拟 3 条总线。图 7.1-1 中 PX.0 模拟 SCK 线，PX.1 模拟 SI 线，PX.2 模拟 SO 线，PX.4 ～ PX.6 选择芯片。

7.1.2 SPI 总线应用实例——EEPROM AT93C64 的连接

7.1.2.1 AT93C64 简述

AT93C64 是 ATMEL 公司 SPI 总线方式的同步串行 EEPROM 芯片，采用 CMOS 技术，具有 1KB 的存储容量，即 128 字节，能够重复写 100 万次，字节写入时间最大 10ms。和并行的

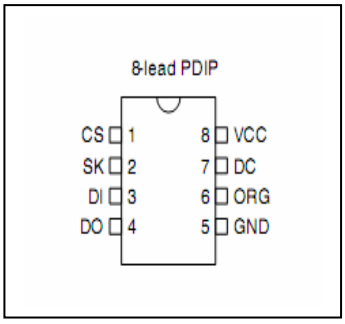
EEPROM 芯片相比，串行 EEPROM 芯片的数据传送速度较低，但其引脚少、体积小、抗干扰能力强，所以特别适用于存放非挥发数据、占用资源少、面积小、传送速度要求不高的 MCU 应用系统中。

AT93C64 的引脚排列和内部原理如图 6.25 所示。

（1）引脚功能

- CS —— 片选，高有效
- SK —— 同步时钟，上升沿触发
- DI —— 数据输入
- DO —— 数据输出
- ORG—— 接地输出 8 位数据，接高电平或悬空输出 16 位数据
- DC —— 空脚
- VCC —— +5V
- GND—— 0V

图 7.1-3 AT93C64 引脚排列



(2) 控制指令

AT93C64 的 7 条控制指令见表 7.1-1。

表 7.1-1 AT93C64 控制指令集

指令	起始位	操作码	字节地址	字地址	字节数据	字数据	内容
ERAL	1	00	10XXXXX	10XXXX	空		所有存储单元擦除
ERASE	1	11	A6-A0	A5-A0	空		指定单元擦除
EWDS	1	00	00XXXXX	00XXXX	空		写禁止
EWEN	1	00	11XXXX	11XXX	空		写允许
READ	1	10	A6-A0	A5-A0	空		指定单元读
WRAL	1	00	01XXXXX	01XXXX	D7-D0	D15-D0	所有存储单元写入
WRITE	1	01	A6-A0	A5-A0	D7-D0	D15-D0	指定单元写

指令格式排列分三部分：（1）起始位和操作码。SB 是指令起始位，该位为逻辑“1”，起始位之后跟操作码；（2）操作地址部分。操作码后面接操作地址，若操作的对象是字节，则地址位 A_6～A_0,共 7 位；若操作的对象是字，则地址位 A_5～A_0,共 6 位；（3）操作数据部分。

7 条指令的含义是：

1）写允许指令（EWEN）。允许对芯片进行擦/写操作。芯片上电后自动进入禁止擦写状态，故在进行擦/写操作之前，须先执行该指令。

2）指定单元擦指令(ERASE)。对指定地址进行擦除操作，即使指定单元内容为“FFH”。擦除期间，DO 引脚电平为“0”，擦除结束，DO 引脚电平为“1”。

3） 指定单元写（WRITE）指令。 对指定地址进行写操作，即将数据通过 DI 写入使指定单元，先写最高位，最后写最低位。写字节最大用时 10ms。

4） 指定读（READ）指令。对指定地址进行读操作，即将指定单元数据通过 DO 读出，先读出最高位，最后读出最低位。

5） 所有存储单元擦除（ERAL）指令。 将芯片整体擦除。即使全体单元内容为“FFH”。

6）所有存储单元写（WRAL）指令。 对芯片整体进行写操作。

7）写禁止（EWDS）指令。 该指令可对以写入的数据进行写保护。

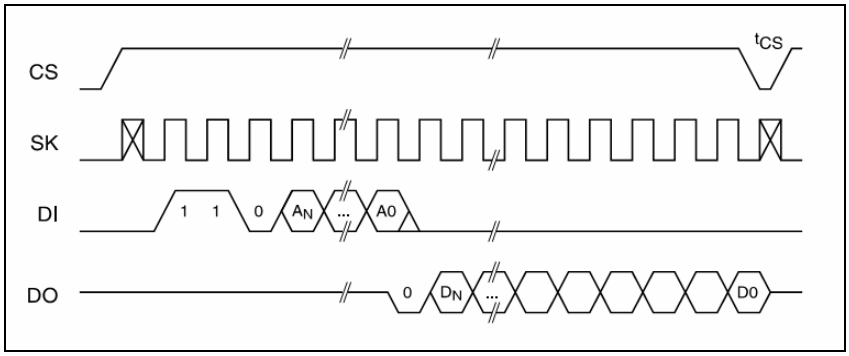
（3）读写时序

① 读字节。如图 7.1-4，CS 线有效（H）后，SK 发出 1 个起始脉冲，随后在 SK 移位时钟控制下，在 DI 线上输出指令起始位和操作码 110B，然后输出要读单元的地址（7 位，编码 128 字节）。随之 DO 线出现 1 位连接“0”，接下来指定存储单元的数据串从最高有效

位到最低有效位依次出现；每位的变化在 SK 的上升沿时刻。

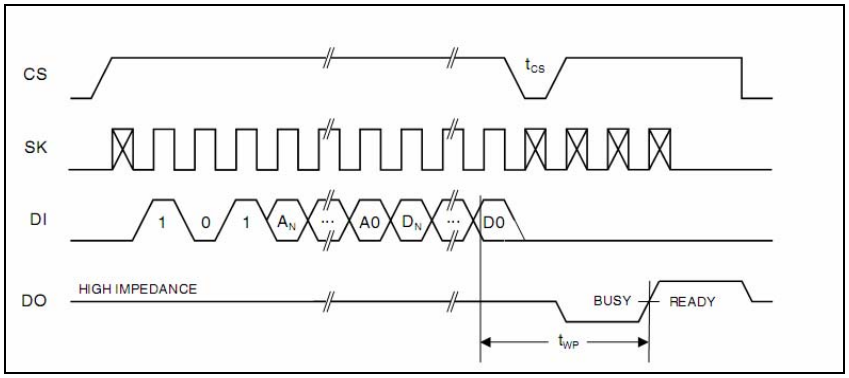
② 写字节。如图 7.1-5， CS 线有效（H）后，SK 发生 1 个起始脉冲，随后在 SK 移位时钟控制下，在 DI 线上输出指令起始位和操作码 101B，接下来输出要写单元的地址，然后

图 7.1-4 AT93C64 读时序



输出数据，从 MSB(DN）依次到 LSB (D0)； AT93C64 芯片接收到 LSB 位后，开始内部擦写过程，该过程最大时间 10ms，典型值 3ms。DI 线上出现最低数据位 LSB 位后，若 CS 线上出现最小间隔为 250ns(t_{CS}）的负脉冲，则芯片会在 DO 线送出 BUSY/READY 信号，逻辑“0”表示芯片正在内部写，处于“忙”状态；逻辑“1”表示内部写完成，可以接收新的指令。若 CS 线上不出现最小间隔为 250ns(t_{CS}）的负脉冲，则 DO 线不送出 BUSY/READY 信号。

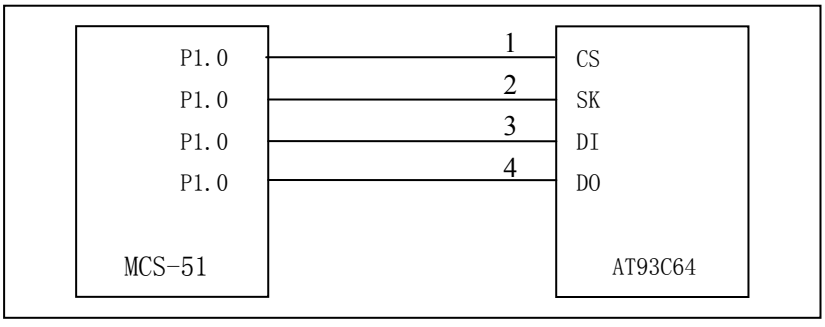
图 7.1-5 AT93C64 写时序图



7.1.2.2 AT93C64 和MCS-51 的连接

AT93C64 和 MCS-51 的连接有两种方式，一种是四线制，将 SPI 芯片的 CS、SK、DI 和 DO 引脚分别连接到 MCS-51 的 4 个 I/O 端口位；另一种是三线制，是将 SPI 芯片的 DI 和 DO 并联接后作为一条线连接到 MCS-51 的一个端口位。AT93C64 和 MCS-51 的四线制连接如图 7.1-6。

图 7.1-6 AT93C64 和 MCS-51 的连接



举例。将 MCS-51 RAM 区首地址 40H 开始的 8 个字节数据写入到 AT93C64 中首地址 40H 开始的存储区中。汇编程序如下。

```
；-------------------------赋值定义-----------------------------------
        DATA   EQU   60H              ；操作数据存放单元 DATA
        ADDR   EQU   62H              ；操作 EEPROM 地址存放单元 ADDR
        CS     BIT   P1.0             ；SPI 片选
        SK     BIT   P1.1             ；移位时钟线
        DI     BIT   P1.2             ；SPI 输入线
        DO     BIT   P1.3             ；SPI 输出线
；-------------------------AT93C64 初始化-----------------------------
        CLR           CS
        CLR           SK
        SET           DI
        SET           DO
        LCALL         WRNUM          ；调用“写数据设置”子程序
        LCALL         EWEN           ；调用“写允许”子程序
        MOV           ADDR,#40H      ；  EEPROM 数据首地址，40H
        MOV           B,#08H         ；待写数据个数
        MOV           R0,#40H        ；MCS-51 数据首地址，40H
LOOP:   MOV           A,@R0          ；取出待写数据
        MOV           DATA,A         ；待写数据送入 DATA 寄存器
        LCALL         WRITE          ；调用字节写子程序
        INC           R0             ；待写数据地址加
        INC           ADDR           ；EEPROM 被写数据地址加
        DJNZ          B,LOOP         ；写循环
        SJMP          $
；-----------------写数据设置子程序-----------------------------------
；此处添入要写到 AT93C64 的数据
WRNUM:  MOV     40H,#00H
        MOV     41H,#01H
        MOV     42H,#02H
        MOV     43H,#03H
        MOV     44H,#04H
        MOV     45H,#05H
        MOV     46H,#06H
        MOV     47H,#07H
        RET
；----------------------写允许子程序-----------------------------------
EWEN:   SETB    CS
        MOV     A,#100 B          ；写允许指令（高字节位）
        MOV     B,#03             ；写允许指令高字节 3 位
        LCALL   OUTDATA           ；调用数据输出子程序
        MOV     A,#01100   000B   ；写允许指令（低字节位）
        MOV     B,#07             ；写允许指令低字节 7 位
```

```
                                        ；写允许指令格式#100 1100000B
          LCALL     OUTDATA             ；调用数据输出子程序
          CLR       CS                  ；芯片等待
          RET
；----------------------数据输出子程序---------------------------------
；MCS-51输出字节数据到AT93C4，输出数据在A,输出位数在B
OUTDATA:  PUSU      B                   ；保护B
EE3：     RR        A
          DJNZ      B,EE3               ；待输出数据最高位到ACC.7
EE4:      LCALL     DOOUT               ；调用字节输出执行子程序
          SETB      DO                  ；释放DO线
          POP       B                   ；恢复B
          RET
；----------------------位输出子程序-----------------------------------
；长度由B内容决定
 DOOUT:   CLR       SK
          RLC       A                   ；待输出字节最高位到C
          MOV       DI,C                ；DI总线输出数据
          NOP                           ；延时
          SETB      SK
          DJNZ      B,DOOUT
          RET
；------------------------ 字节写子程序-------------------------------
；  EEPROM 被写地址在ADDR,待写数据在DATA，字节长度由B的内容决定。
 WRITE:   PUSH      B                   ；保存B
          SET       CS                  ；片选
          MOV       A,#101B             ；写指令高3位
          MOV       B,#3                ；“写指令” 3位码
          LCALL     OUTDATA             ；调用“数据输出子程序”
          MOV       A,ADDR              ；地址
          MOV       B,#7                ；7位地址
          LCALL     OUTDATA             ；调用“数据输出子程序”，
          MOV       A,DATA              ；待写数据送DPL
          MOV       B,#8                ；8位数据
          LCALL     OUTDATA             ；调用“数据输出子程序”
          CLR       CS                  ；EEPROM芯片内部写，等待
          LCALL     DELAY10ms           ；调用延时子程序10ms,略
          POP       B
          RET
          END
```

7.2 串行I^2C总线接口技术

I^2C总线(Inter Integrate Circuit BUS,或 IC TO IC BUS),又叫IIC总线,是Philips

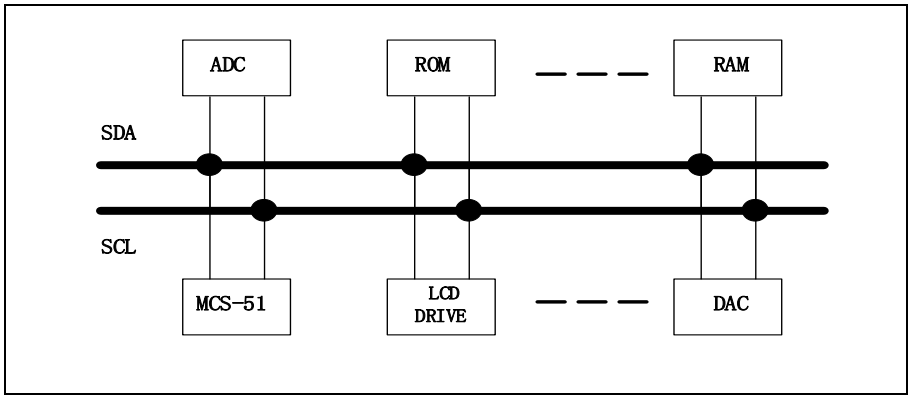
公司设计的一种 IC 芯片之间的全双工同步串行总线技术。和 SPI 总线相比，它用的通信连线更少，采用 2 条线实现数据通信。具有 I^2C 总线接口的 MCS-51 单片机，如 P87LPCXXX\C8051FXX 等，可通过自身专用接口和 I^2C 总线连接；没有 I^2C 总线接口的 MCS-51 单片机可以通过编写软件，在 I/O 端口位模拟通信电平时序和 I^2C 总线连接。实用上，I^2C 总线由于其连线少、时序模拟方便、寻址方式容易等特点，近年来在 MCU 应用系统中得到了快速的发展，各种类型的 MCU 外围器件，如存储器、A/D、D/A、LCD 驱动器、逻辑门阵列电路等，都有 I^2C 总线型芯片的身影。

7.2.1 I^2C 串行总线接口技术简述

1 I^2C 串行总线的特点

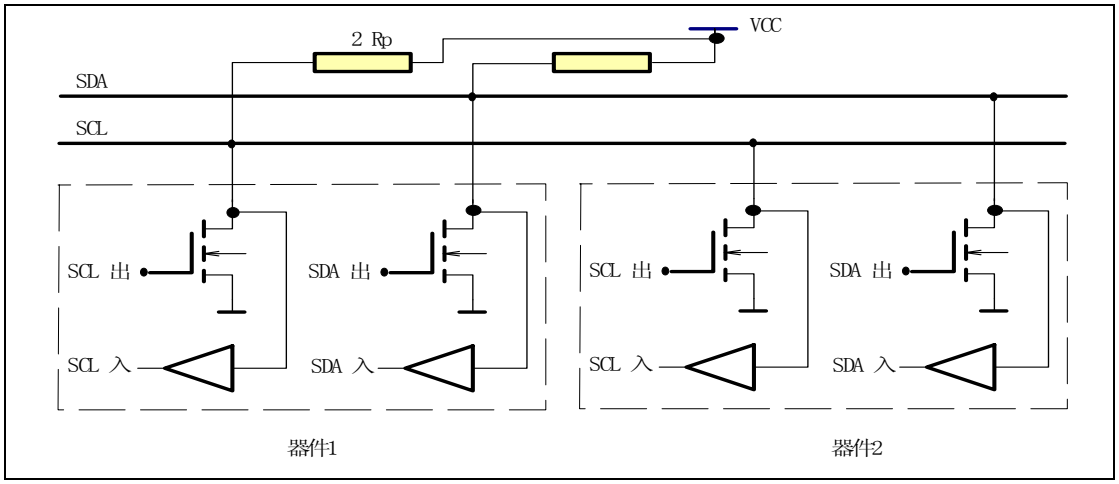
1）使用 2 条通信总线交换数据。I^2C 通信系统如图 7.2-1 所示。I^2C 串行总线由 2 条总线构成，一条是时钟线 SCL，另一条是数据线 SDA。数据线 SDA 上传送数据，数据传送以帧为单位，每帧含一个字节数据和一位应答信号位，数据字节的传送次序为先高位后低位；时钟线 SCL 提供数据传送的位同步信号。所有的 I^2C 器件都挂接在这 2 条总线上，形成 I^2C 通信系统。I^2C 器件分为主机器件和从机器件，主机器件负责数据传送的控制，从机器件按照主机器件的控制时序进行操作。

图 7.2-1 I^2C 总线系统的示意



2） 2 条总线的数据传送都是双向的，挂接在总线上的 I^2C 器件，接口为开漏极形式，须接上拉电阻。上拉电阻 R_P 取值一般在 5K～10K，使用时可查阅具体器件的技术手册。如图 7.2-2 所示。

图 7.2-2 I^2C 总线器件的接口结构



总线空闲时为高电平状态，任一芯片输出低电平，都会使总线信号变低。挂接总线上器件的数量（总线的负载能力）受总线电容量 400pF 上限的限制。总线在标准模式下的数据传输速率为 100Kbps，快速模式下为 400Kbps，高速模式下为 3.4Mbps。

（3）I^2C 芯片的寻址方式采用引脚设置、软件寻址，和 MSC-51 的地址总线无关。在此之前我们接触的外围器件芯片，在对它们进行寻址时，总是采用地址总线进行译码来选择片选的方式实现的， MCS-51 用指令在 P2P0 口输出不同的地址编码来选择不同的芯片，或者是用其它 I/O 端口选择芯片；I^2C 芯片只有 2 条连线，不能和 MCS-51 的地址总线连接，也没有片选端可供 I/O 端口选择；因此，芯片地址的识别采用了全新的方式——用“引脚电平、软件寻址”实现。“引脚电平”是芯片有 3 个地址引脚，可接固定的“0”“1”电平而设置成不同的引脚地址。软件寻址指令的编码内容包括“器件标识”、“引脚电平”和“方向位”三部分，如表 7.1-1 所示。“器件标识”（ D_7～D_4）用来区别不同种类的 I^2C 芯片，如 EEPROM、A\D 等，厂家在制作时已将数据固化在芯片中；“引脚电平”（ D_3 ～D_1）是使用者编程时对应的该芯片的 3 个引脚连接的固定电平状态。若寻址指令编码数据若和某 I^2C 芯片的“器件标识”和“引脚电平”相一致，则该芯片被选中。“方向位”（ D_0）表示对该芯片进行的操作，“1”表示读操作，“0”表示写操作。I^2C 芯片的这种特性，使得它的“片选”既不必象并行芯片那样通过数据线“译码”，也不必象 SPI 芯片那样用专门的 I/O 口的位来选择，只要在 SDA 线上传送寻址指令就可以了。

表 7.2-1　I^2C 芯片寻址指令编码格式

	器 件 标 识				引 脚 电 平			方 向 位
位　序	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
寻址编码	DA_3	DA_2	DA_1	DA_0	A_2	A_1	A_0	1=读，0=写

常用的 I^2C 芯片的寻址指令的格式如表 7.2-2。

表 7.2-2 常用 I^2C 芯片的寻址格式

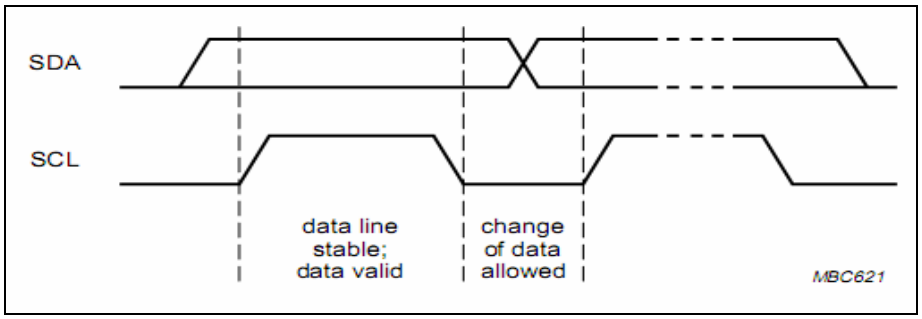
器件	类型	寻址指令格式
PCF8570	256BRAM	1010$A_2A_1A_0$R/W
PCF8582	256BEEPROM	1010 $A_2A_1A_0$R/W
PCF8574	8 位 I/O	0100 $A_2A_1A_0$R/W
PCFSAA1064	4 位 LED 驱动器	0111 $A_2A_1A_0$R/W
PCF8591	8 位 A/D、D/A	1001 $A_2A_1A_0$R/W
PCF8583	RAM、日历	1010 $A_2A_1A_0$R/W

（4）I^2C 是多主总线结构，具有仲裁功能。I^2C 总线系统可以接入多个 MCU，每个 MCU 都可以具有主机资格，即总线是多主机系统。从机一般为外围芯片。总线的操作由主机控制，即由主机发出数据传送的起始信号、停止信号和同步信号。在多主机情况下，总线规定了相应的仲裁协议，可保障在任何时刻经公平竞争后只有一个主机控制总线。

2　I^2C 总线的数据传输协议及方式

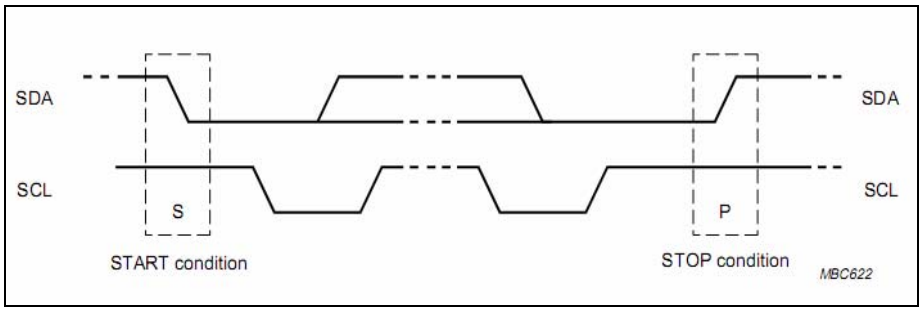
1）数据位的有效性的规定。I^2C 总线的技术条件规定，在时钟线 SCL 为高电平期间，数据线 SDA 上的数据状态必须保持稳定。只有在时钟线 SCL 为低电平期间，SDA 线上的数据才允许发生变化。见图 7.2-3。

图 7.2-3 I²C 总线数据位有效性的规定



2）起停控制和应答信号的规定。数据传送的起始信号和停止信号时序规定见图 7.2-4。

图 7.2-4 I²C 总线系统的起停条件规定



①“起始”信号（S）—— 在 SCL 为高期间，SDA 线由高到低的变化表明数据传送开始。

②“停止”信号（T）—— 在 SCL 为高期间，SDA 线由低到高的变化表明数据传送停止。

需要说明的是，I²C 总线的启动(S)和停止(T)信号，都是由主机发出的。在启动信号出现后，总线就处于“忙”状态；在停止信号发出后，表示该主机放弃总线，总线处于空闲状态。连接在总线上的芯片，若内部具有 I²C 总线接口，则能够及时检测到 S 和 T 信号；对于内部没有 I²C 总线接口的单片机（设处于从机状态），则需要在一个 SCL 时钟周期内至少 2 次采样 SDA 线来读取 S 和 T 信号。

③ 应答信号（A）。 应答信号是接收方接收到一个字节数据后，给予发送方的回应，表示接收正常。I²C 总线上传送一个字节的数据后，发送方在第 9 个 SCL 脉冲高电平期间，释放 SDA 线（高电平），接收方使该线变为低电平，作为应答信号。发送方在收到应答信号后，才能继续进行后续的数据发送。

④ 非应答信号（$\overline{A}$）。如果接收方未能收到数据字节，在第 9 个 SCL 脉冲高电平期间，它将在数据线 SDA 上发出“非应答” 信号，即高电平。发送方在收到该信号后，发出停止信号或新的起始信号。当主机接收来自从机的数据时，在接收最后一个数据帧后，须发出非应答信号，使从机释放 SDA 线，以便随后主机发出停止信号。

应答信号（A）和非应答信号（$\overline{A}$）见图 7.2-5 。

3）数据传送形式

数据传送以数据帧为单位，每帧含 1 个字节 8 位数据和 1 个应答信号位，共 9 位。帧内字节的传送顺序是先最高位（MSB），依次到最低位（LSB），传送数据帧的数量没有限制，直到停止信号为止。见图 7.2-6。

图 7.2-5 I²C 总线系统的应答信号和非应答信号

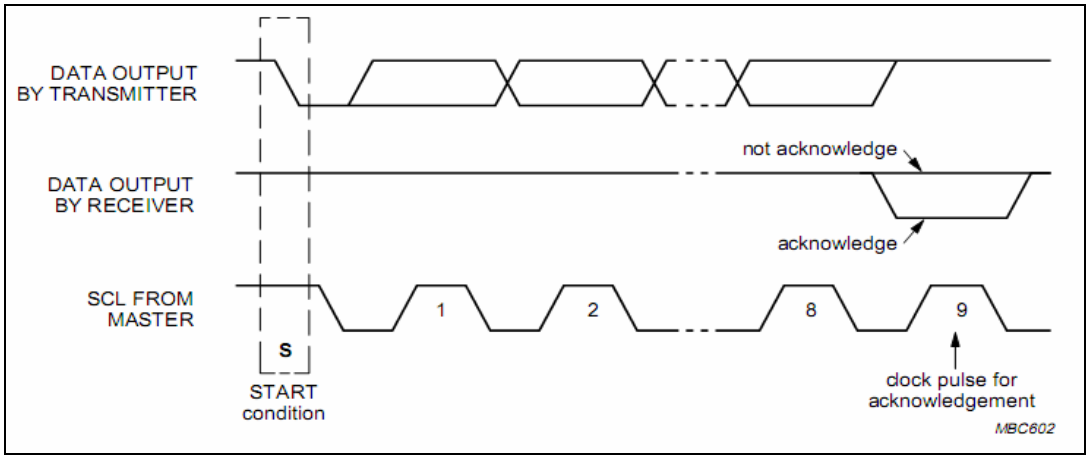
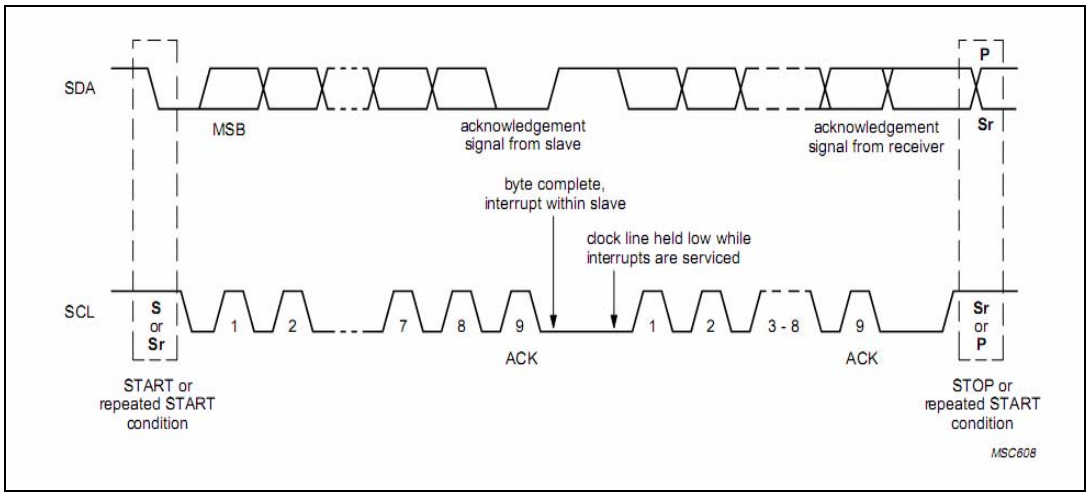


图 7.2-6 I²C 总线系统的字节（帧）传送形式

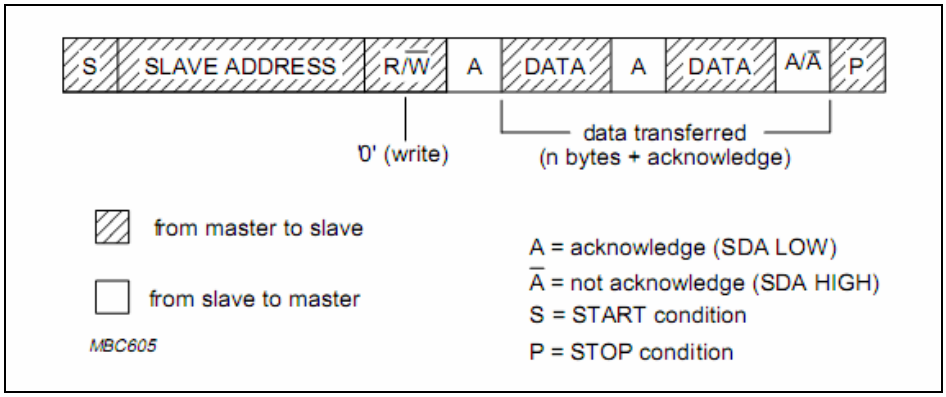


I²C 协议规定，数据传送的基本步骤是：主机发出起始信号后，先发出从机的 8 位地址信息，该信息前 7 位是从机芯片的内部地址，第 8 位是读写信息（R/$\overline{W}$），“1”为读，“0”为写；然后进行和从机之间的读写数据传送；最后由主机发出停止信号，结束数据传送。

概括起来，I²C 总线上的数据传送有三种形式：

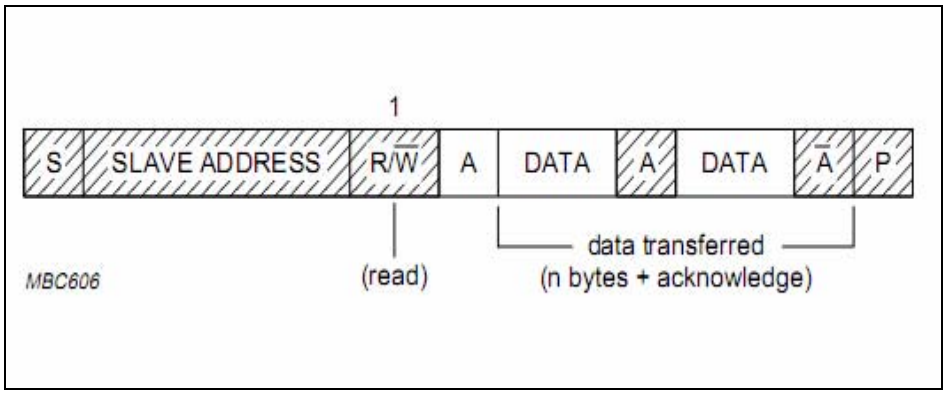
① 主机向从机发送数据。主机发送从机内部地址并收到从机应答信号后，即可开始 N 个数据帧的发送。见图 7.2-7。

图 7.2-7 主机向从机发送数据的操作过程



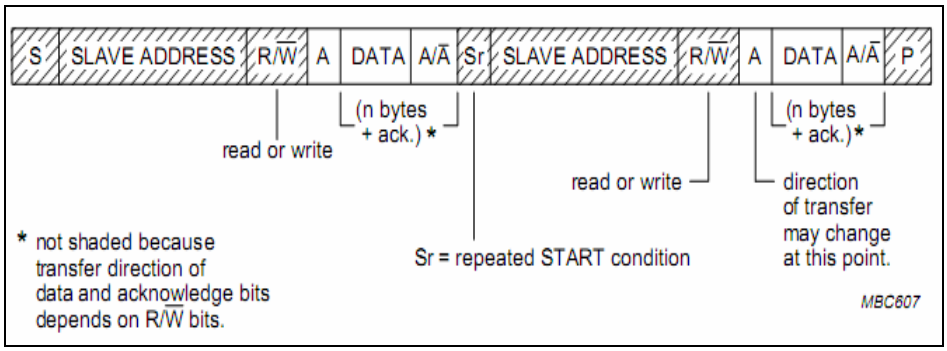
② 主机接收从机数据。主机发送从机内部地址并收到从机应答信号后，即可开始 N 个数据帧的接收。见图 7.2-8。

7.2-8 主机从从机接收数据的操作过程



③ 主机对从机读写操作。数据传送的方向需要改变时，起始位和从机地址都被重复发送一次。见图 7.2-9。

图 7.2-9 主机对从机发送数据和接收数据的混合操作过程

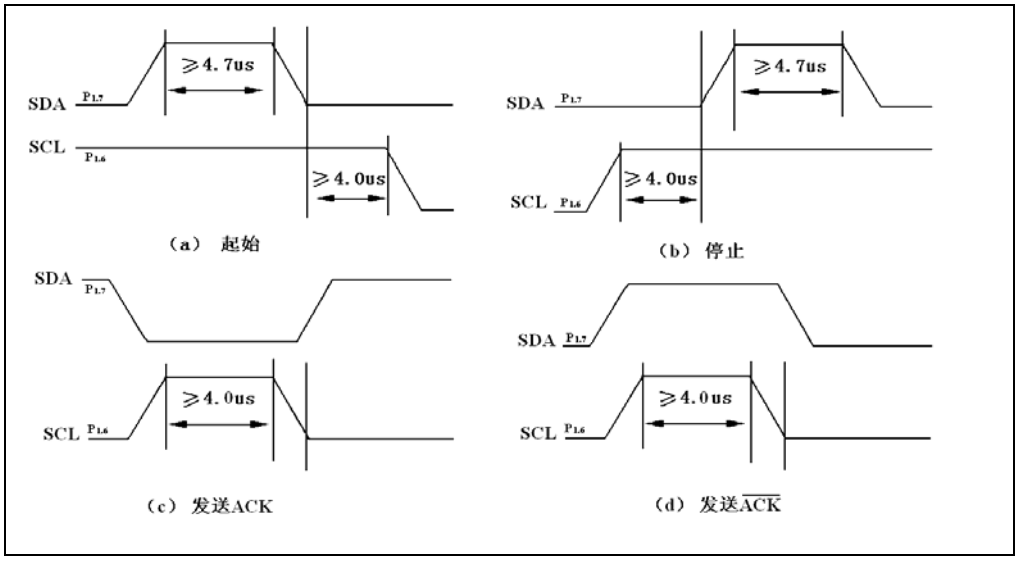


由此可见，数据传送是在主机的控制下进行的，起始信号、停止信号和从机寻址信号均由主机发出，传送的方向由方向位（R/$\overline{W}$）确定；数据接收方须提供应答或非应答信号。

（4）I^2C 总线的信号时序定义

图 7.2-10 给出了 I^2C 总线的起始、停止、应答、非应答信号的时序规定。图中是 MCS-51 用 $P1._7$和 $P1._6$端口位进行软件模拟波形。

图 7.2-10 I^2C 总线信号的时序



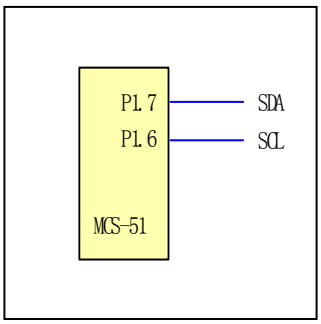
7.2.2 MCS-51 模拟 I²C 总线时序

1、硬件连接。对于内部集成了 I²C 总线接口的 MCS-51 系列单片机，如 P89CXXX、P87LPCXX 等，可用其专用的接口位（SDA\SCL）和 I²C 总线连接；而对于多数的内部并未集成 I²C 总线接口 MCS-51 系列单片机，我们可用 P0～P3 的 I/O 端口模拟 I²C 总线的 SDA 和 SCL。注意，若使用 P1～P3 端口，由于它们已经具有内部上拉电阻，故无需再外接上拉电阻；若使用 P0 口，则须接 5～10K 的上拉电阻。

2、信号时序的模拟。即依据 I²C 总线的通信规约（图 7.2-10 中的信号时序），用软件来控制 SDA 和 SCL 线的电平变化，使其满足起始、停止、应答和非应答以及数据传送的时序要求。下面以图 7.2-11 的硬件连接为例，实现起始、停止、应答和非应答信号的软件模拟。

```
程序中：   SDA  EQU     P1.7
           SCL  EQU     P1.6
```

图 7.2-11 MCS-51 模拟 SDA SCL



（1）起始信号的模拟

起始信号由主机发出（设机器周期为 1 us，下同）。信号规约参见图 7.2-10(a)。

```
；------STR 子程序，模拟起始信号（S）----------
STR:     SETB    SDA
         SETB    SCL
         DB      0,0,0,0,0      ；延时
         CLR     SDA            ；SDA “1”大于 4.7 us
         DB      0,0,0,0        ；延时
         CLR     SCL            ；SCL “1”大于 4us
         RET
```

(2) 停止信号的模拟

停止信号由主机发出。信号规约参见图 7.2-10(b)。

```
；--------------------------STP 子程序，模拟停止信号(T)-----------------------
STP:     CLR     SDA
         SETB    SCL
         DB      0,0,0,0        ；延时
         SETB    SDA            ；SCL “1”大于 4 us
         DB      0,0,0,0,0      ；延时
         CLR     SDA            ；SDA “1”大于 4.7 us
         RET
```

(3) 应答信号的模拟

应答信号由信息接收方在正常接收一个字节的信息后，从 SDA 线上发出。信号规约参见图 7.2-10(c)。

```
；--------------------------ACK 子程序，模拟应答信号(A)-----------------------
ACK:     CLR     SDA
```

```
        SETB    SCL
        DB      0,0,0,0         ；延时
        CLR     SCL             ；SCL“1”大于 4 us
        NOP
        SETB    SDA             ；SDA“0”大于 4 us
        RET
```

(3) 非应答信号的模拟

非应答信号由信息接收方在不能正常接收信息、或希望终止接收数据时，从 SDA 线上发出。信号规约参见图 7.2-10(d)。

```
；-----------------------------NACK 子程序，模拟非应答信号----------------------
NACK:   SETB    SDA
        SETB    SCL
        DB      0,0,0,0         ；SCL 大于 us
        CLR     SCL
        NOP
        SETB    SDA             ；SDA“1”大于 4 us
        RET
```

7.2.3 I²C 应用实例—— AT240XX 芯片连接（*）

AT240XX 系列芯片是 ATMEL 公司的产品，是一种采用 I²C 总线接口技术的 EEPROM 器件，该系列常见的型号有 AT2401A/02/04/08/16 等 5 种，其存储容量分别为 1024/2048/4096/8192/16384 位，也就是 128/256/512/1024/2048 字节。本节以 AT24C02 为例介绍。

1 AT24C02 的工作原理简述

1）引脚。如图 7.2-12

① VCC ： +5V 电源

② GND ： 电源地

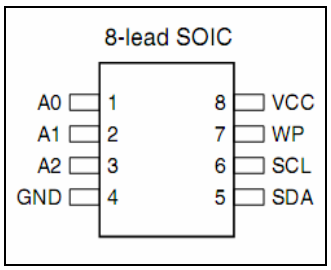
③ SCL ： 时钟输入

④ SDA ： 数据 I/O

⑤ A0 A1 A2 ： 引脚地址

⑥ WP ： 写保护。接高电平时禁止写操作

图 7.2-12 AT24C02 引脚



2）运行状态

① 起始状态。符合 I²C 总线的起始信号时序规定，该状态位于所有的操作命令之前。

② 停止状态。符合 I²C 总线的停止信号时序规定。

③ 应答/非应答信号。由接收数据的器件发出，可以是 AT24C02，也可以是 MCS-51；符合 I²C 总线的应答信号时序规定，在第 9 个 SCL 时钟周期出现。

④ 备用方式（stanby mode）。该方式保证 AT24C02 在没有读写方式时处于低功耗状态。在两种情况下芯片会自动进入该状态：（1）上电复位；（2）在接到了停止信号并完成了内部写操作后。

3）器件寻址。AT24C02 芯片的寻址指令字节编码格式如下。

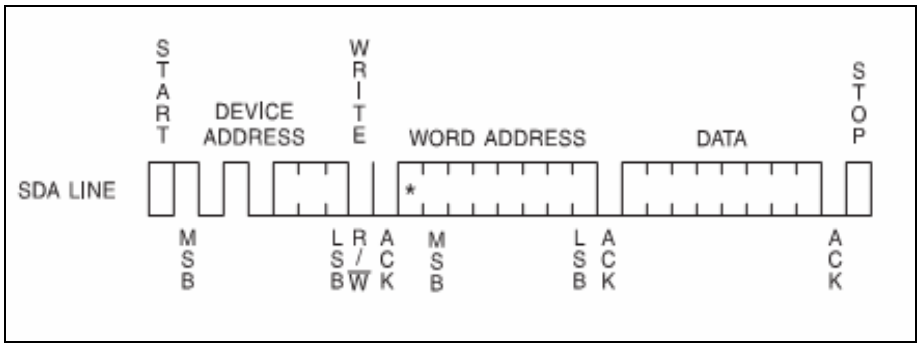
位	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
内容	1	0	1	0	A_2	A_1	A_0	R/W

寻址指令字节由三部分内容组成：（1）D_7～D_4 =1010 为标识位，表明芯片是 AT24XX 类型芯片；（2）D_3～D_1 = A_2 A_1 A_0 为引脚电平，由芯片的引脚接固定“1”、“0”电平，编码排序为 000～111，即在一个系统中，AT24CXX 芯片的最大数目是 8；（3）D_0= R/$\overline{W}$ 是读/写控制位，为“0”则 AT24C02 被写入数据；为“1”则 AT24C02 被读出数据。

4）写操作。该操作是单片机向 AT24C02 写入数据的过程。分两种形式，一是字节写，二是页面写。

（1）字节写。如图 7.2-13 示。该操作完成对 1 个字节数据的写入操作，即把 1 个字节数据写入 1 个指定的存储单元。其过程是：起始状态后，MCS-51 先送出芯片寻址指令，该

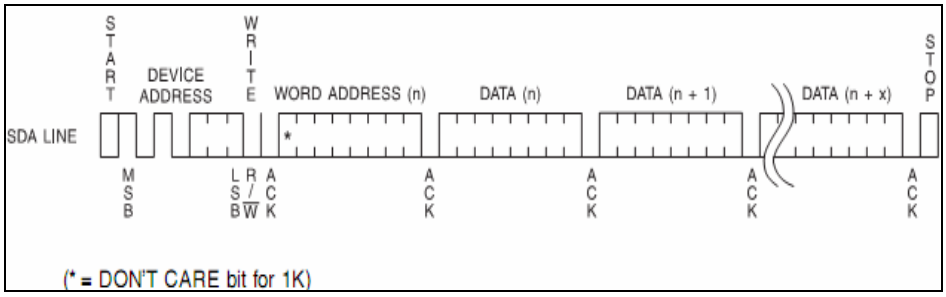
图 7.2-13 字节写时序



指令的 R/$\overline{W}$ 位为“0”，表示是向 I^2C 器件传送命令；在接到 AT24C02 芯片应答信号后，再送出该芯片的指定字节地址；得到应答后，送出要写的数据内容，AT24C02 芯片接收后送应答信号，MCS-51 发停止信号。至此，AT24C02 芯片的外部字节写操作完成。接下来 AT24C02 芯片进入一个内部擦写过程，将收到的数据写入指定单元。内部写最长用时为 5ms。

（2）页写。该操作完成对指定 1 页数据的写入操作，即把 1 页数据写入指定的 1 页存储单元中。1 页的数量和具体芯片有关：AT24CXX 系列中，容量为 1K、2K 的芯片，1 页为 8 个字节；4K、8K、16K 的芯片，1 页为 16 个字节。AT24C02 芯片页写操作是一次写 1 页，即对 8 个存储单元写数据。页写过程的起始条件和字节写一样，不同的是，在送出字节 1 并收到应答信号后，MCS-51 并不发送停止信号，而是继续发送剩余的 7 个字节的数据，送完这 1 页数据后才发送停止信号。收到停止信号后，芯片开始内部写过程，其用时和字节写一样，也是最长 5ms。如图 7.2-14 示。

图 7.2-14 页写时序

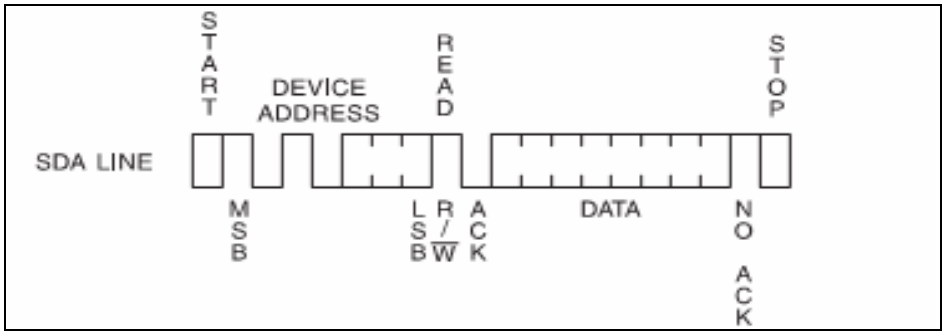


5）写保护。 WP 端接高电平时芯片禁止写操作，此时芯片仅作为 ROM 使用。

6）读操作。读操作是 MCS-5 从 AT24C02 读出数据的过程。读操作和写操作类似，也需要启动、写芯片地址等步骤，不同的是在写芯片地址时将 R/W 位置为“1”；另外在停止信号之后，也没有内部操作时间。读操作分三种形式：现行地址读、随机地址读和顺序地址读。

（1） 现行地址读。在上次读或写操作结束后，芯片内部字地址会自动加 1，产生“现行地址”。一旦本芯片被选中且 R/$\overline{W}$位=“1”,则芯片应答后即将现行地址单元的数据送出。MCS-51 收到数据后，产生非应答信号，然后发出停止条件结束现行读。见图 7.2-14。

图 7.2-14 现行地址读操作时序



（2） 随机地址读。这种方式允许单片机读出 AT24C02 的任意字节内容。它由两个步骤组成：第一步，选择要读的数据地址。单片机发出芯片地址和字节地址，芯片应答后即在 AT24C02 的内部产生了“现行地址”。第二步，执行“现行地址读”段的内容。见图 7.2-15。

（3）顺序地址读。和前两种方式相比，这种方式的最大特点是可以连续地读出一批数据。顺序地址读用“现行地址读”或“随机地址读”启动，在 MCS-51 接收第 1 个字节数据后，不是发送非应答而是发送应答（ACK）信号，AT24C02 收到应答（ACK）信号后就会对字地址加 1，并送出该地址单元的数据，该过程一直持续到单片机发出非应答（$\overline{ACK}$）和停止信号为止。见图 7.2-16。

图 7.2-15 随机地址读操作时序

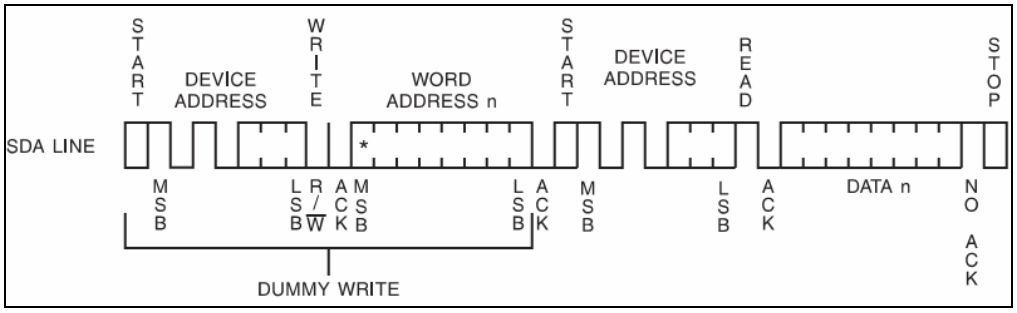
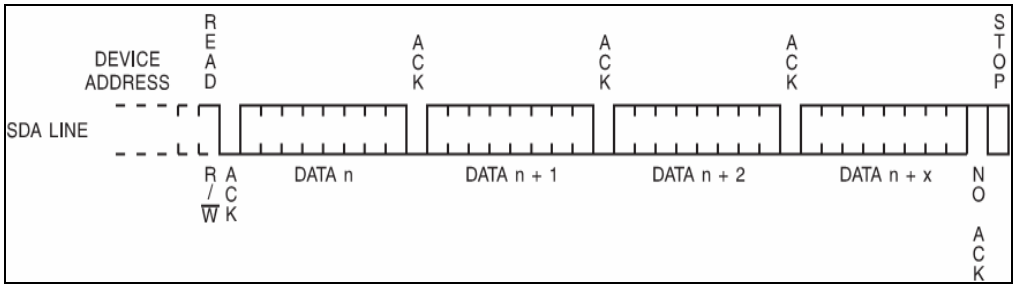


图 7.2-16 顺序地址读操作时序

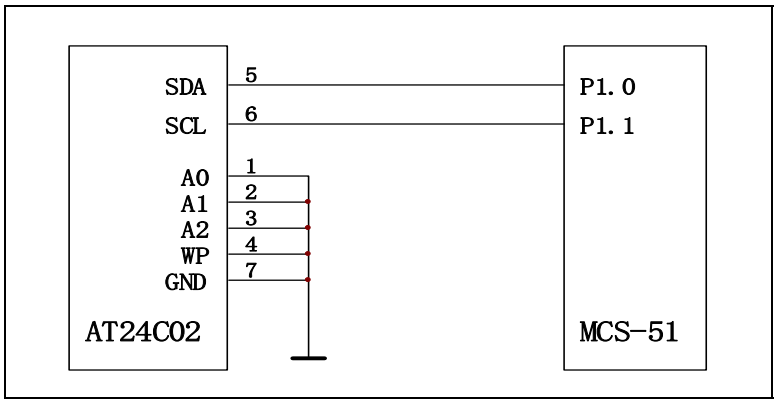


2　单片机和 AT24C02 的连接

本例将 MCS-51 RAM 区首地址为 30H 的 16 个单元内容写入 AT24C02 的首地址为 30H 的 16 个单元；然后再将这些内容从 AT24C02 读到内部 RAM 区首地址为 30H 的 16 个单元中。

（1）硬件连接：见图 7.2-17。SDA、SCL 接 P1.0 和 P1.1，因 P1 口内有上拉电阻，故无需再接上拉电阻；WP 接地，允许写操作；A2～A0 接地，该“芯片地址”=1010 000R/$\overline{W}$。

图 7.2-17　MCS-51 和 AT24C02 的连接



（2）汇编程序：

```
；------------------------------主程序-----------------------------------------
            SDA  BIT   P1.0
            SCL  BIT   P1.1
            ORG  0000H
            MOV  R1,  #30H          ；RAM 首地址
            MOV  R2,  #16           ；16 个数据
            MOV  A,   #0            ；
IN1:        MOV  @R1, A             ；RAM 区,30H 首地址，写 16 个数据
            INC  A                  ；
            INC  R1                 ；
            DJNZ R2,  IN1           ；
            NOP                     ；此处设断点，观察 30H 内容为 0H、1H、2H…0FH
            LCALL     WR24C02       ；调用写 240C2 子程序
            MOV  R1,  #30H          ；RAM 首地址
            MOV  R2,  #16           ；16 个数据
            MOV  A,   #60H          ；立即数#60H
IN2:        MOV  @R1, A             ；RAM 区,30H 首地址，写 16 个数据
            INC  A                  ；
            INC  R1                 ；
            DJNZ R2,  IN2           ；
            NOP                     ；此处设断点，观察 30H 内容为 60H、61H、62H…6FH
            LCALL     RD240C2       ；调用读 24C02 子程序
            DB   00                 ；此处设断点，观察 30H 内容为 0H、1H、2H…0FH
            SJMP $
；------------------------------写 24C02 子程序-------------------------------
；将 MCS-51 首地址 30H 的 16 个字节内容写入 24C02 首地址 30H 的 16 个字节中。
WR24C02:    MOV       R0,  #30H     ；待写数据首地址
            MOV       R1,  #10H     ；待写数据数量
WRNEXT:     LCALL     WROP          ；调用字节写操作子程序
            LCALL     DELAY5MS      ；调用延时 5 ms 子程序
```

```
                INC     R0              ；待写数据地址加 1
                DJNZ    R1,WRNEXT       ；判写完否
                RET
；-------------------------字节写操作子程序-----------------------------------
；采用字节写方式，写 1 个字节，包括向 24C02 送起始信号、芯片地址、字节地址、写数据
；和停止信号
WROP:           ACALL   STR             ；调用起始信号子程序
                MOV     A,#0A0H         ；芯片地址，写控制
                ACALL   WRBYTE          ；调用字节写输出子程序
                JC      WRFAIL          ；判 24C02 的 ACK，无，转移
                MOV     A,R0            ；有 ACK，24C02 字节地址
                ACALL   WRBYTE          ；调用字节写执行子程序
                JC      WRFAIL          ；判 24C02 的 ACK，无，转移
                MOV     A,@R0           ；有 ACK，待写数据到 A
                ACALL    WRBYTE         ；写数据
                ACALL   STP             ；送停止信号
                RET
WRFAIL:         ACALL   STP             ；无 ACK，送停止信号
                RET
；-------------------------字节写输出子程序-----------------------------------
；具体操作 SDA、SCL 线，完成字节的数据送出。被写数据在 A。若写入成功，复位 C；反之置位 C。
WRBYTE:         MOV     R7,#08H         ；8 位
WBY0:           RLC     A               ；送出最高位
                JC      WBY-1           ；为“1”，转移
                CLR     SDA             ；为“0”，
                SJMP    WBY-0
WBY-1:          SETB    SDA             ；SDA 输出“1”
                DB      0,0             ；延时
WBY-0:          DB      0,0             ；延时
                SETB    SCL             ；时钟信号 1
                DB      0,0,0,0         ；延时
                CLR     SCL             ；时钟信号 0
                DJNZ    R7,WBY0         ；判 8 位送完否
                MOV     R6,#5           ；等待 24C02 ACK 信号
WAIT:           SETB    SDA             ；将 SDA 线拉高
                DB      0,0,0,0         ；延时
                SETB    SCL             ；将 SCL 线拉高
                DB      0,0,0,0         ；延时
                JB      SDA,NOACK       ；无 ACK
                CLR     C               ；有 ACK，(C) = 0
                CLR     SCL             ；SCL 线低
                RET
NOACK:          DJNZ    R6,WAIT         ；再延时等待 24C02 ACK 信号
                SETB    C               ；等待失败，(C) = 1
```

```
                CLR     SCL
                RET
;  ---------------------------------读 24C02 子程序---------------------------------
;  将 24C02 首地址 30H 的 16 个字节数据，读存到 RAM 首地址 30 区。
RD24C02:        MOV     R0,  #30H           ；R0,读出数据存放 RAM 区首地址
                MOV     R1,  #10H           ；10H 字节量
RDNEXT:         LCALL   RDOP                ；调用读操作子程序
                INC     R0
                DJNZ    R1,RDNEXT
                RET
;  ----------------------------读操作子程序-----------------------------------
;  采用随机地址读形式，送起始信号、送芯片地址、送字节地址、读 24C02  1 个字节数据
;  并送出停止信号。24C02 字节数据被读出后，送 R0 指针。
RDOP:           ACALL   STR             ；调用起始信号
                MOV     A,#0A0H         ；芯片地址，写控制
                ACALL   WRBYTE          ；调用字节写子程序，发芯片地址
                JC      WRFAIL          ；判 24C02 的 ACK，无，转移
                MOV     A,R0            ；有 ACK，  R0 中是地址
                ACALL   WRBYTE          ；送字节地址
                JC      WRFAIL          ；判 24C02 的 ACK，无，转移
                ACALL   STR             ；调用起始信号
                MOV     A,#0A1H         ；芯片地址，读控制
                ACALL   WRBYTE          ；送读指令
                JC      WRFAIL          ；判 24C02 的 ACK，无，转移
                ACALL   RDBYTE          ；调用字节读子程序
                MOV     @R0,A           ；R0 指针，MCU RAM 地址
                ACALL   STP             ；读 1 字节，送停止信号
                RET
WRFAIL：        ACALL   STP             ；无 ACK，送停止信号
                RET
；字节读子程序。操作 SDA、SCL 线，完成字节的数据读入；读出字节存 A。读出后发非应答信号。
RDBYTE：        SETB    SDA             ；释放 SDA 线
                MOV     R7,#08H         ；8 位
CY1:            DB      0,0             ；
                CLR     SCL             ；
                DB      0,0,0,0         ；
                SETB    SCL             ；准备读
                DB      0,0,0,0         ；延时
                CLR     C               ；进位标志置“0”
                JNB     SDA,RBY-0       ；读 1 位数据。为“0”，  转移
                SETB    C               ；数据为 1，C=1
RBY-0:          RLC     A               ；1 位数据移入 A（最高位先入）
                DB      0,0,0,0         ；延时
                DJNZ    R7,CY1          ；8 位数据移完否
```

```
        CLR     SCL             ;
        DB      0,0,0,0         ; 准备发非应答信号
        CLR     SDA             ;
        SETB    SDA             ; 非应答
        DB      0,0,0,0         ; 延时
        SETB    SCL
        DB      0,0,0,0,0
        CLR     SCL
        DB      0,0,0,0         ; 非应答信号完
        CLR     SDA             ; 释放 SDA
        RET
; ----------------------STR 子程序。见 7.2.2 节---------------------------------
; ----------------------STP 子程序。见 7.2.2 节---------------------------------
; ----------------------DELAY5MS 子程序，略。-----------------------------------
        END
```

7.3 串行单总线(1-Wire)技术（*）

单总线（ 1-Wire）是 Dallas 公司设计的串行总线技术，和 SPI、I^2C 等总线不同，它采用一条总线线进行双向的数据通信，该线既传送控制信号，又传送数据数据信号。因此更能节省单片机的 I/O 接口资源和减少印制线路板面积。

单总线适合于单主机系统。MCS-51 作为系统中的主机。其它具有单总线接口的芯片作为系统的从机。当只有一个从机芯片连接在总线时，该系统叫单节点系统；当多个从机芯片连接在总线时，该系统叫多节点系统。

7.3.1 单总线的工作原理

具备单总线通信功能的集成电路芯片叫单总线芯片。单总线芯片通过漏极开路引脚并联在单总线上，总线通过一个约 5K 的上拉电阻接电源。MCS-51 系列单片机可通过 I/O 口和单总线连接。当连接在总线上的某芯片不使用总线时，它输出高电平以释放总线，因此总线的闲置状态为高电平。

总线中数据的交换是在主机的控制下进行的。MCS-51 和其它单总线芯片交换数据过程，一般分为三个步骤：一是主机对总线的初始化，包括呼叫从机芯片和从机芯片回答；二是单片机发出芯片寻址指令，通过和每个芯片固有的 64 位 ROM 地址代码相比较，使指定的芯片成为数据交换的对象，而其余的芯片则处于等待状态；三是单片机发送具体操作指令进行读写操作。在只有一个从机芯片的情况下，步骤二可以省略其中的寻址的过程，仅执行一条“跳跃”命令，然后进入步骤三。

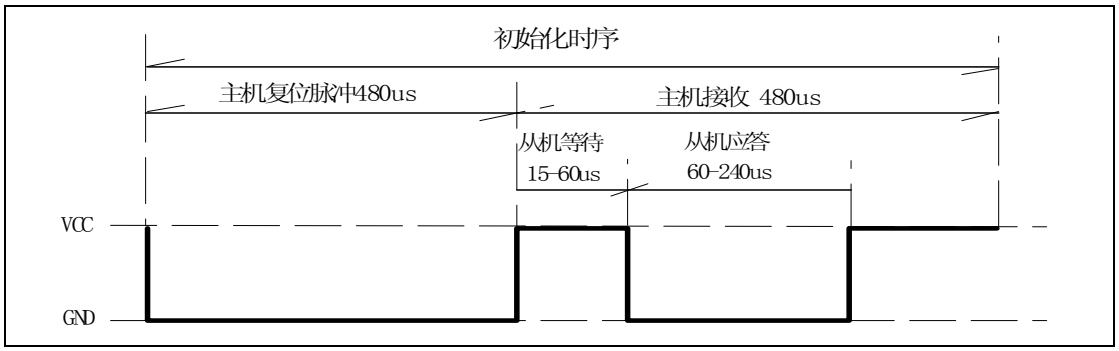
单总线硬件连接简单，而相应的软件控制过程则比较复杂。除了从机芯片的寻址过程复杂外，MCS-51 对系统的操作必须严格单总线通信协议的时序要求。这些时序规定是：

1）初始化序列时序。该时序由主机发出，对单总线系统进行复位，并由从机发出应答信号。单总线的所有通信过程都以初始化时序开始，初始化时序包括主机发出的复位脉冲和从机的应答脉冲，该过程至少需要 960 us。如图 7.3-1，主机在总线上输出“0”电平并保持至少 480us 作为复位脉冲，表示主机对系统复位并呼叫从机，然后主机释放总线，总线在上拉电阻的作用下变为“1”电平，至此复位脉冲完成；从机在接到主机的复位脉冲后，先对

自己内部复位，然后对总线输出“0”电平，并保持 60us～240 us，作为对主机呼叫的应答信号，主机检测到该信号，即可确认总线上有从机存在。

2）写时序。如图 7.3-2。在“写时序”中，主机对从机写 1 位数据。一个写时序至少 60us，在两个写时序之间要有 1us 的恢复时间。

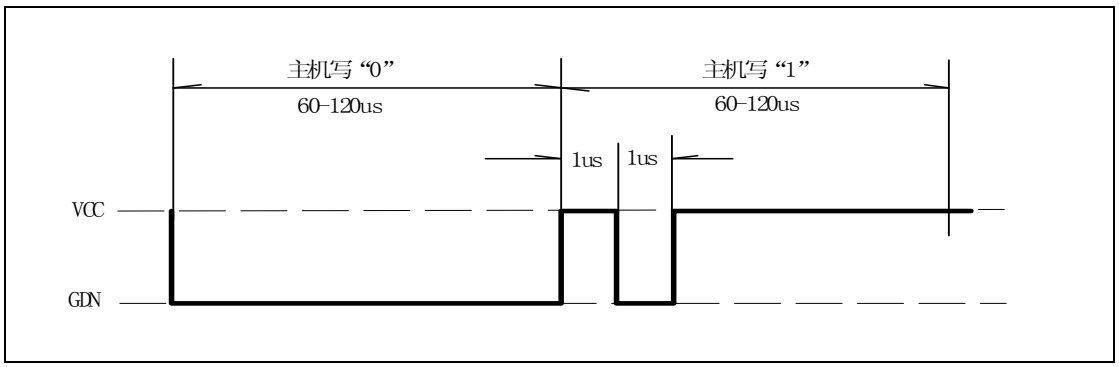
图 7.3-1　单总线初始化时序



（1）写“0”。主机向总线输出“0”，并保持 60us 后释放总线。从机在写时序开始 15us 后开始对总线采样，读入总线数据。

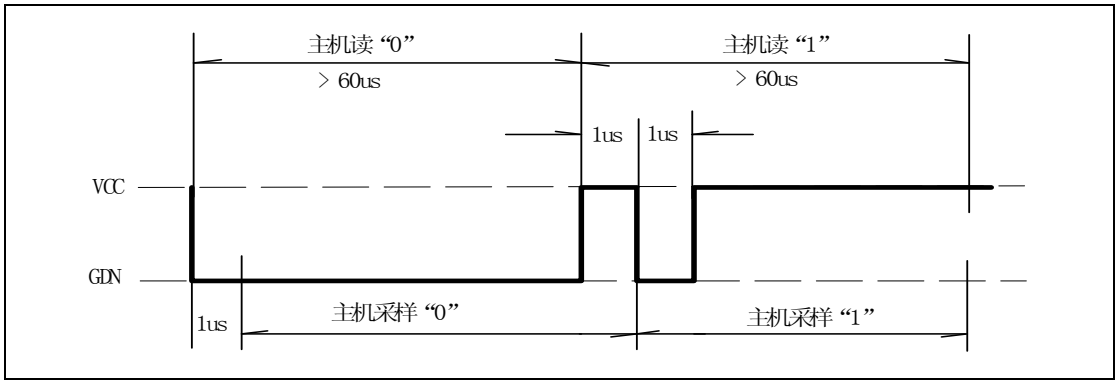
（2）写“1”。 主机向总线输出“0”，1us 后输出“1”并保持 60us。从机在写时序开始 15us 后开始对总线采样，读入总线数据。

图 7.3-2　主机写时序



3）读时序。如图 7.3-3。单总线器件仅在主机发出读时序时，才向主机送出数据，所以在主机发出读数据命令后，必须立即产生读时序，以便从机开始传送数据。每个读时序也至少需要 60us 时间。且两个读时序的间隔也至少为 1us。读时序由主机发起，拉低总线至少 15us，然后从机接管总线，开始发送“0”或“1”数据，主机在 1us 后采样总线接收数据。

图 7.3-3　主机读时序



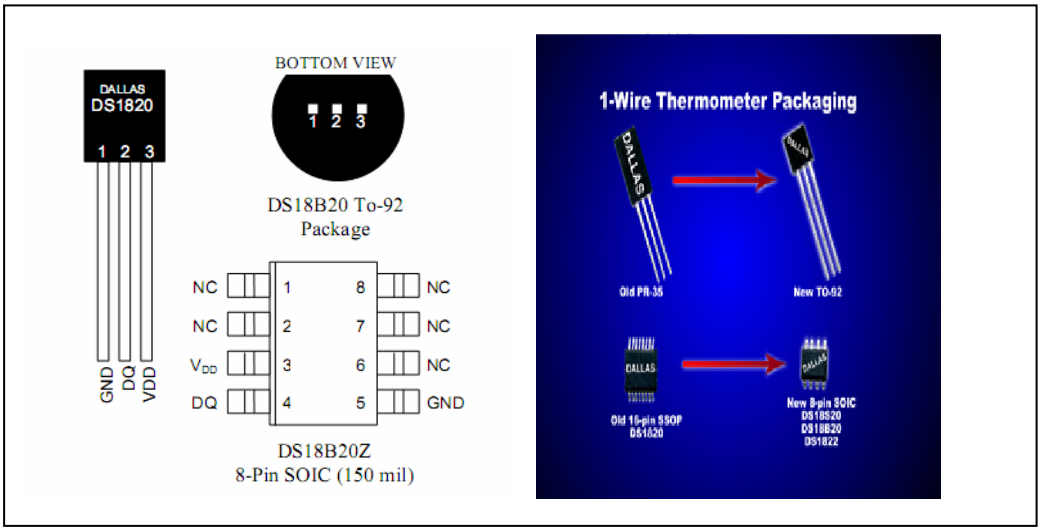
7.3.2　单总线（1-Wire）应用实例——数字温度测量与控制

本节介绍用单总线的应用实例。用 MCS-51 控制数字温度传感器芯片 DS18B20，实现温度的测量和控制。

7.3.2.1　单总线数字化温度传感器DS18B20 芯片

数字化温度传感器 DS18B20 芯片是世界上第一片采用单总线方式的温度传感器。如图 7.3-4 为芯片的外形图和引脚图。图中显示了该芯片的两种封装形式，SOIC 为小外形集成电路封装，另一种为三极管外形封装。

图　7.3-4　　DS18B20 芯片外形及引脚



该芯片测量物体的温度，并在单总线上传送测量数据。和传统的模拟信号测量方式相比，它提高了抗御干扰的能力，适用于环境控制、设备控制、过程控制以及测温类消费电子产品等领域。

三极管外形封装的 DS18B20，外形如同一只小功率三极管，其引脚定义是：

1、GND　　接地

2、DQ　　单总线接口

3、V_{DD}　　电源

1）　DS18B20 芯片的主要特点

- 工作电压 3.0V～5.5V
- 温度测量范围 -55° C～125° C
- 在－10° C～ +85° C 范围内，测量精度为±0.5° C。
- 待机状态下无功率消耗。
- 可编程分辨率 9～12 位，每位分别代表 0.5° C、0.25° C、0.125° C 和 0.0625° C。
- 温度测量时间 200ms。

2）DS18B20 芯片内部结构

DS18B20 芯片内部中主要部件是 64 位光刻 ROM 和温度传感器。

64 位的光刻 ROM，其中存放着 64 位的序列号代码，在出厂前被厂家制作固化在芯片中，是该芯片的地址序列代码。64 位代码的排序是：开始 8 位(28H)是产品类型标号，接下来的 48 位是该芯片自身的序列号，最后 8 位是前面 56 位数字的循环冗余校验码（CRC=$X^8+X^5+X^4+X+1$）。该序列代码在主机发出读 ROM 指令后可被读出，主机可由此确定芯片的身份，如同

I^2C 芯片引脚地址。64 位的序列号代码的作用是可允许在一个单总线系统中连接多个 DS18B20 芯片。有资料介绍，一个系统中最多可连接 8 个 DS18B20 芯片；若再增加数量，则需要扩大 MCU 端口的总线驱动能力。

温度传感器是芯片的核心部分，它连续地对物体温度进行测量，并连续地将新测量结果存放在高速暂存器 RAM 中，存放形式如下：

低字节（LS Byte）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

高字节（MS Byte）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
S	S	S	S	S	2^6	2^5	2^4

测量温度值被放在两个字节中，高字节的高 5 位是符号位，代表一位符号。若这 5 位均为“0”，表示符号为正，测量温度为正值；若这 5 位均为“1”，则表示符号为负，测量的温度为负值。高字节的低 3 位和低字节的 8 位，共 11 位，是测量的数值部分。测量值为正时，将数值乘以 0.0625 即可得到实际测量温度数；测量值为负时，将数值其变补再乘以 0.0625 即可得到实际测量温度的绝对值。比如温度+125° C 对应的转换数字为 07D0H，温度－55° C 对应的转换数字为 FC90H。

3）DS18B20 的 ROM 指令和 RAM 指令

ROM 指令用来确认 DS18B20 的身份，即在众多的单总线芯片或多个 DS18B20 中指定某一个芯片作为操作对象。确定的基本方式是核对各芯片的 64 位的序列号代码，该过程比较复杂，需要若干条 ROM 指令的配合；在仅用 1 个 DS18B20 芯片的场合，只需用“跳过”指令（CCH），就可省略确认身份的过程。

DS18B20 的 RAM 指令见 7.3-1。RAM 指令用来对已经确认身份、被指定为操作对象的 DS18B20 芯片进行具体的读写操作。

4） DS18B20 的读写操作过程

DS18B20 属于单总线芯片，对其进行数据交换必须符合单总线操作时序和三个操作步骤（见 7.3.1 节叙述）。MCS-51 对 DS18B20 的读写过程的三个步骤是：

① 发出初始化时序。MSC-51 将总线拉低 480us，DS18B20 收到复位信号后即开始自身复位，并发出应答信号，然后进入测温状态，并将测得温度随时装入高速暂存器 RAM 中，等待读出；

② 发出读 ROM 指令，即选择进行数据交换的具体芯片。若只有一只 DS18B20 芯片，可用“跳跃”指令越过芯片选择而直接指定该芯片；

③ 发出对 RAM 读操作的指令，读出温度数据。

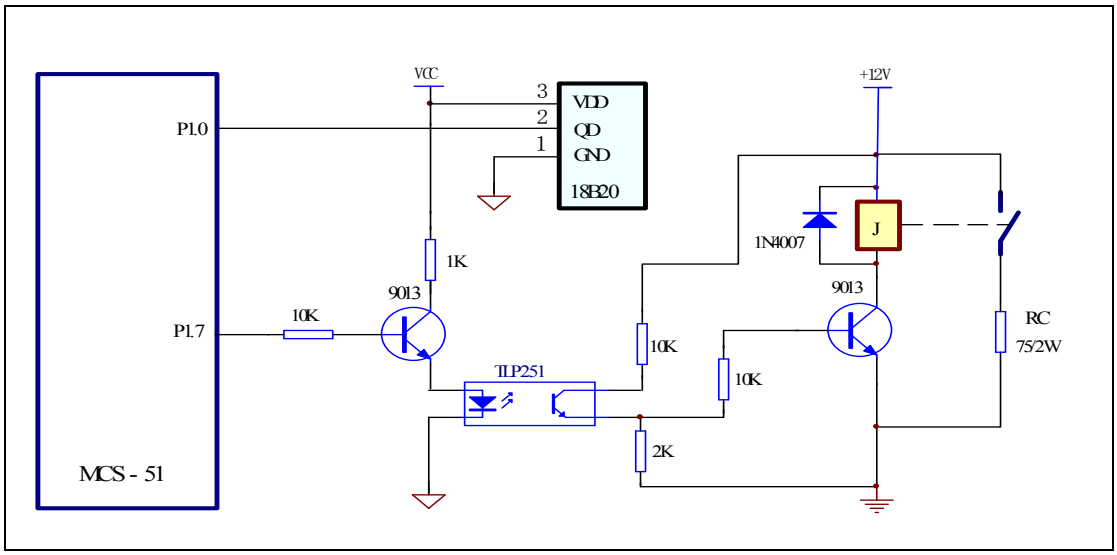
表 7.3-1 DS18B20 的 RAM 指令

指令	代 码	功 能
温度变换	44H	启动温度转换，12 位转换时最长为 750ms。结果存入内部 9 字节 RAM 中
读暂存器	BEH	读 DS18B20RAM 中 9 字节内容
写暂存器	4EH	发出向内部 RAM 的 2、3、4 字节写上、下限温度数据和配置寄存器命令，紧跟该命令之后，是传送三字节的数据
复制暂存器	48H	将 RAM 中 2、3 字节的内容复制到 E^2PROM 中
重调 E^2PROM	B8H	将 E^2PROM 中内容恢复到 RAM 中的第 2、3 字节
读供电方式	B4H	读供电模式。寄生供电模式时发送“0”， 外接电源供电发送“1”

7.3.2.2　MCS-51 和DS18B20 组成的温度控制单元

本例用 MCS-51 和 1 片 DS18B20 组成温度控制单元。硬件原理图见 7.3-5。该单元模拟实现空调中恒温控制功能，使被测物体的温度保持在预先设置的范围之内。原理是由 DS18B20 测量环境温度，然后将测量温度送给 MCS-51，MCS-51 将测量温度与设置温度的上

图 7.3-5　模拟恒温控制原理示意图



下限值进行比较，若测量温度小于设置温度下限值，则进行加热处理；若测量温度大于设置温度上限值，则进行制冷处理。为简便起见，（1）设测量温度均为正值，（2）用电阻 R_C 模拟环境温度，DS18B20 测量该电阻温度，（3）R_C 加热由继电器 J 控制，J 接点闭合对电阻通电加热；J 接点断开对电阻自然冷却。

汇编程序如下：

```
;  ---------------------------主程序-----------------------------------------------
                BUS     BIT     P1.0    ;   定义 1-Wire
                CTL     BIT     P1.7    ;   温度加热继电器控制 I/O 位
                TEML    EQU     29H     ;   转换后读出温度值低 8 位
                TEMH    EQU     28H     ;   转换后读出温度值高 8 位
                STEMH   EQU     27H     ;   设置温度低限值
                STEML   EQU     26H     ;   设置温度高限值
                FLAG    BIT     38H     ;   检测 DS18B20 存在标志，“1”=存在
                        ORG     0100H
MAIN:           LCALL   GET-TEM         ;    调用读温度转换子程序
                MOV     A,      TEML    ;  温度低字节
                MOV     C,      40H     ;  温度高字节位 0（28H.0）
                RRC     A               ;
                MOV     C,41H           ;  温度高字节位 1
                RRC     A               ;
                MOV     C,42H           ;  温度高字节位 2
                RRC     A               ;
                MOV     C,43H           ;  温度高字节位 3
                RRC     A               ;  温度高、低字节合并，乘 0.0625，存 A
```

```
PW:         MOV     TEML,A          ；温度测量实际值存 29H
            CJNE    A,STEMH，S1     ；  比较设置温度高限值
S1:         JNC     SH              ；  大于“设置温度高限值”，转移
            CJNE    A,STEML,S2      ；
S2:         JC      SL              ；  小于“设置温度低限值”，转移
            SJMP    DL              ；  没有越线
SH:         CLR     CTL             ；  大于“设置温度高限值”，冷却
            SJMP    DL
SL:         SETB    CTL             ；  小于“设置温度低限值”，加热
            SJMP    DL
DL:         LCALL   DELAY           ；   调用延时 16ms 子程序
            SJMP    MAIN
；----------------------温度转换子程序-------------------------------
；初始化总线、成功后再发出温度转换指令(44H)、等待 18B20 转换完成；
；再复位总线、发出跳跃指令（CCH)、发出读指令（BEH)，读入数据。

GET-TEM:    SET     BUS             ；  使总线变高
            LCALL   INT-1820        ；  调用总线初始化子程序
            JB      FLAG,TSS2       ；判总线初始化是否成功,是，转移；
            RET                     ；  否，返回
TSS2:       MOV     A,#0CCH         ；  初始化成功，发出跳过指令
            LCALL   WR-1820         ；  写指令
            MOV     A,#44H          ；发出温度转换指令
            LCALL   WR-1820         ；写指令
            LCALL   DELAY1          ；延时 800ms 子程序（从略）
            LCALL   INT-1820        ；准备读，先调用总线初始化子程序
            MOV     A,#0CCH         ；发出跳过指令
            LACLL   WR-1820         ；写指令
            MOV     A,#0BEH         ；发出读温度指令
            LACLL   WR-1820         ；写指令
            LACLL   RD-1820         ；读出温度值，保存在 TEML/TEMH
            RET
；-----------------------------初始化（复位）子程序--------------------------
；发出复位脉冲，等待从机回应。该过程须大于 960us（设 Fosc=12MHz）
    INT-1820:SET        BUS         ；  使总线变高
            NOP
            CLR         BUS         ；  初始化开始
            LCALL       DELAY500    ；  调用延时 500us 子程序(该程序省略)
            SETB        BUS         ；  释放总线，准备进入等待从机回应
            DB          0,0,0
            MOV         R0,#25H     ；  延时计数
    TSR2:   JNB         BUS,TSR3    ；  检测从机回应，有回应，转移
            DJNZ        R0,TSR2     ；延时 150us
            LJMP        TSR4        ；无回应，转移
```

```
TSR3:   SETB    FLAG        ；有回应，标志 1
        LJMP    TSR5
TSR4:   CLR     FLAG        ；无回应，标志 0
        LJMP    TSR7
TSR5:   MOV     R0,#117     ；延时 200 us
TSR6:   DJNZ    R0,TSR6
TSR7:   SETB    BUS         ；释放总线
        RET
；--------------------------------写字节子程序-----------------------------------
；被写数据在 A。先写最低位，共 8 位，时序符合单总线写要求
WR-1820:MOV     R2,#8       ；8 位数据
        CLR     C           ；
WR1:    CLR     BUS         ；总线拉低
        MOV     R3,#7       ；总线拉低延时
        DJNZ    R3,$
        RRC     A           ；
        MOV     BUS,C       ；送最低位到总线
        MOV     R3,#23      ；延时
        DJNZ    R3,$
        SETB    BUS         ；拉高总线
        NOP
        DJNZ    R2,WR1      ；8 位送完?
        SETB    BUS         ；释放总线
        RET
；---------------------------读温度子程序-------------------------------------
；从 18B20 读出测量的温度值，高字节存 TEMH,低字节存 REML
RD-18B20:MOV    R4,#2       ；  2 字节
        MOV     R1,#TEML    ；低字节
RD0：   MOV     R2,#8       ；8 位
RD1:    CLR     C           ；清零
        SETB    BUS         ；释放总线
        DB      0,0
        CLR     BUS         ；  拉低总线
        DB      0,0,0
        SETB    BUS         ；释放总线
        MOV     R3,#9
        DJNZ    R3,$        ；   延时
        MOV     C,BUS       ；读入总线数据
        MOV     R3,#23      ；延时
        DJNZ    R3,$
        RRC     A           ；读入数据移入 A
        DJNZ    R2,RD1      ；1 字节读完?
        MOV     @R1,A       ；存高、低字节
        DEC     R1          ；高字节
```

```
DJNZ        R4,RD0        ；2 字节读完？
RET
```

本章小结

（1） 串行 I/O 总线扩展技术用于 MCS-51 应用系统中印制电路板内芯片之间的数据交换。它的最大特点和优势是节省连线和面积。

（2） 本章介绍了三种串行 I/O 总线，SPI、IIC 和 1-WIRE,分别使用三条（片选$\overline{CS}$不算在内的话)、两条和一条总线进行通信。

（3） SPI 芯片有专用的片选端$\overline{CS}$，SCK 是时钟线，MOSI 和 MISO 是两条单向的数据线。在只有 1 个芯片时，可省去$\overline{CS}$，在只有 1 个传送方向时，可省去一条数据线。

（4）IIC 总线用两条线传送数据。有 4 种时序。芯片有引脚地址。

（5）单总线使用连线最少。但芯片的寻址相对复杂，通信占用时间也更多。因此该类芯片的品种比前两种要少得多。

（6）实用中，串行方式的芯片越来越多，是一种技术的发展趋势。人们更愿意使用引脚少、连线简单的串口芯片，而放弃并行器件。

思考题与习题

7.1 比较 UART 和串行总线 SPI、IIC、1-WIRE 的区别。

7.2 试叙述三种串行总线各自的芯片寻址方式。

7.3 画出 IIC 总线的起停、应答和非应答时序图。编写模拟时序的子程序。

7.4 简述 IIC 的数据传输过程和数据传送的三种基本形式。

7.5 SPI 使用几条线进行数据通信？在什么情况下可以省略 MISO、MOSI 和$\overline{CS}$ 线?

7.6 在 MCS-51 上扩展 2 片 AT24C04,画出连接图并说出 2 片 AT24C04 的地址。

7.7 选择一种具有 SPI 接口的 IC 器件和 MCS-51 连接。

7.8 叙述 1-WIRE 总线系统中，MCS-51 和从机芯片交换数据的三个典型步骤。

7.9 叙述图 7.3-5 中 MCS-51 和 18B20 数据交换的过程，并画出其汇编程序的流程图。

第 10 章　MCS-51 单片机应用系统设计

10.1　MCS-51 单片机应用系统的结构

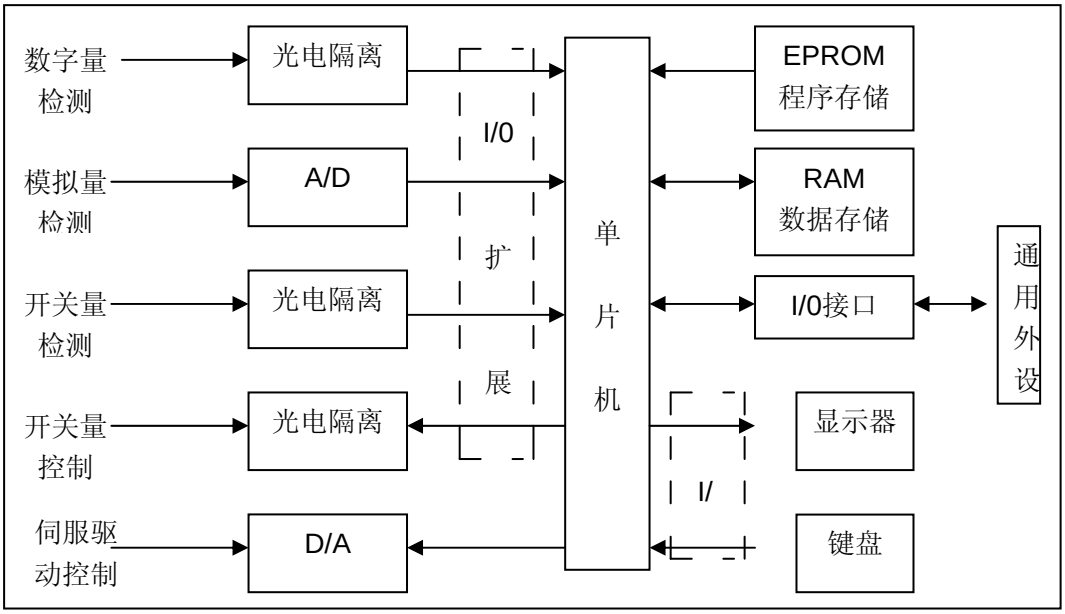
由于单片机具有体积小、功耗低、功能强、可靠性高、实时性强、简单易学、使用方便灵巧、易于维护和操作、性能价格比高、易于推广应用、可实现网络通信等技术特点，因此，单片机在自动化装置、智能仪表、家用电器，乃至数据采集、工业控制、计算机通信、汽车电子、机器人等领域得到了日益广泛的应用。

从系统的角度来看，单片机应用系统是由硬件系统和软件系统两部分组成的。硬件系统包括单片机及其扩展部分和各功能模块部分，如信号测量模块、人机交互模块等；软件系统包括监控程序和各种应用程序。

由于单片机多用于测控领域，因而其典型应用系统包括单片机系统、基本输入输出通道以及基本的人及对话通道，对大型的多机测控系统，还包括机与机之间进行通信的交互通道。图 10.1-1 是一个典型的单片机应用系统的结构框图。

1）前向通道及其特点

图 10.1-1 典型单片机应用系统结构



前向通道一般包括数字量检测输入、模拟量检测输入、开关量检测输入等，前向通道与现场采集对象相连，根据现场采集对象的不同，这些输入量都是由安放在现场的传感、变换装置产生的，是一个模拟、数字混合的电路系统，一般功耗较小，但是它却是现场干扰进入的主要通道，也是整个系统抗干扰的重点。

2）后向通道及其特点

后向通道是应用系统的伺服驱动通道，作为应用系统的输出通道，大多数需要功率驱动，根据输出控制的不同，后向通道电路多种多样，输出信号形式有电流输出、电压输出、开关量输出等。

3）人机交互通道及其特点

单片机应用系统中的人机通道是为了人机对话而设置的，主要有键盘、显示器、打印机等通道接口，可以实现人工干预系统，设置参数等。

人机通道具有以下特点：

（1）人机通道一般都是小规模的；（2）单片机应用系统的人机对话通道及接口大多采用内总线形式，与计算机系统扩展密切相关；（3）人机通道接口一般都是数字电路，可靠性高，结构简单。

10.2 MCS-51 单片机应用系统设计

单片机应用系统设计应当考虑其主要技术性能(速度 精度 功耗 可靠性 驱动能力等)，还应当考虑功能需求，应用需求，开发条件，市场情况，可靠性需求，成本需求等，一旦这些指标确定下来，系统将在这些指标限定下进行。

10.2.1 总体方案设计

1）明确设计任务

认真进行目标分析，根据应用场合、工作环境、具体用途，考虑系统的可靠性、通用性、可维护性、先进性，以及成本等，提出合理的、详尽的功能技术指标。

2）器件选择

（1）单片机选择

不同类型、不同系列的单片机内部结构、外部总线特征等均不相同，对一个系统而言，单片机的型号和结构直接决定其总体结构，因此，在确定系统结构时，首先要选择单片机型号或系列。单片机选择主要从性价比及开发周期方面考虑：

①单片机性价比。应根据应用系统的要求和各种单片机的性能，选择最容易实现产品技术指标的机型，而且能达到较高的性能价格比。这样既能实现系统的技术指标，又不会造成浪费。②开发周期。选择单片机时，要考虑具有新技术的新机型；更重要的是应考虑选用技术成熟，有较多软件支持，可得到相应的开发工具的机型。这样可借鉴许多现成的技术，移植一些现成软件，以节省人力、物力、缩短开发周期，降低开发成本。

（2）外围器件的选择

外围器件应符合系统的精度、速度和可靠性、功耗、抗干扰等方面的要求。应考虑功耗、电压、温度、价格、封装形式等其他方面的指标，应尽可能选择标准化、模块化、功能强、集成度高的典型电路。

3）总体设计

总体设计就是根据设计任务、指标要求和给定条件，设计出符合现场条件的软、硬件方案，并进行方案优化。应划分硬件、软件任务，画出系统结构框图。要合理分配系统内部的硬件、软件资源。包括以下几个方面：

（1）从系统功能需求出发设计功能模块。包括显示器、键盘、数据采集、检测、通信、控制、驱动、供电方式等。

（2）从系统应用需求分配元器件资源。包括定时器/计数器、中断系统、串行口、I/O 接口、A/D、D/A、信号调理、时钟发生器等。

（3）从开发条件与市场情况出发选择元器件。包括仿真器、编程器、元器件、语言、程序设计的简易程度等。

（4）从系统可靠性需求确定系统设计工艺。包括去耦、光隔、屏蔽、印制板、低功耗、散热、传输距离/速度、节电方式、掉电保护、软件措施等。

10.2.2 硬件设计

由总体设计所给出的硬件功能，在确定单片机类型的基础上进行各个功能电路模块的设计，最后综合成一个完整的硬件系统，并进行必要的工艺结构设计，制作出印刷电路板，组装后即完成了硬件设计。

1）硬件电路设计的一般原则

在进行单片机应用系统的硬件设计时应注意以下问题：

（1）采用新技术，尽量选用标准化，模块化的选择典型电路。

（2）在条件允许的情况下，尽量选用功能强、集成度高的电路或芯片。

（3）选择通用性强、市场供应足的元器件。

（4）满足应用系统的功能要求，并留有适当余地，以便进行二次开发。

（5）充分考虑系统各部分的驱动能力及电源的带负载能力，并注意抗干扰设计。

（6）工艺设计时要考虑安装、调试、维修的方便。

2）硬件电路各模块设计的原则

硬件部分各模块电路主要包括：存储器扩展、I/O 扩展、输入输出通道、通信电路、人机交互通道等，各模块电路设计时应考虑以下几个方面：

（1）存储器扩展：应考虑存储器的类型、容量、速度和接口，尽量减少芯片的数量。

（2）I/O 接口的扩展：由于外设多种多样，使得单片机与外设之间的接口电路也各不相同，因此，I/O 接口通常是单片机应用系统中设计最困难也最复杂的部分之一。选择 I/O 接口，要考虑其体积、价格、负载能力、功能，合适的地址译码方法。

（3）输入通道的设计：输入通道设计包括模拟量输入通道和开关量输入通道设计。开关量要考虑接口形式、电压等级、隔离方式、扩展接口等；模拟输入通道信号检测环节（传感器、信号处理电路等）结合，根据系统的速度和精度等来进行适当的设计。

（4）输出通道的设计：输出通道设计包括开关量和模拟量输出通道设计。开关量要考虑功率、控制方式等，模拟量输出通道要考虑 D/A 转换器的参数（精度、转换速度等）和输出信号的形式。

（5）人机界面的设计：人机界面的设计主要包括键盘、开关、拨码盘、启/停操作、复位、显示器、打印、指示、报警等的设计，要考虑界面的易操作性、人性化、布局合理，另外还要考虑各人机界面的扩展口。

（6）其他方面的设计：如提高负载容限的设计，通信电路的设计，印制板的设计与制作、信号逻辑电平兼容性的考虑及抗干扰的设计等。

10.2.3 软件设计

在进行应用系统的总体设计时，软件设计和硬件设计应统筹兼顾、协调进行，一旦硬件设计确定，相应的软件任务也就确定了。单片机应用系统的软件一般是由系统的监控程序和应用程序两部分组成。其中，应用程序是用来完成诸如测量、计算、显示、打印、输出控制能各种实质性功能的软件；监控程序是控制单片机系统按照预定操作方式运行的程序，它负责组织调度各应用程序模块，完成程序自检、初始化、处理各种命令等功能。

软件设计流程图如图 10.2-1 所示，软件设计可分为以下几个方面。

1）总体规划

结合硬件结构，明确软件任务，确定具体实施的方法，合理分配资源。定义输入/输出、确定信息交换的方式（数据速率、数据格式、校验方法、状态信号等）、时间要求，检查与纠正错误。

2）确定软件结构

确定软件的总体任务之后，需要确定软件结构，使各功能程序模块化、子程序化。合理的软件结构是设计一个性能优良的单片机应用系统软件的基础，一般有以下两种程序设计设计方法：

（1）模块程序设计：模块程序设计是把一个较长的程序分解为若干个功能相对独立的较小的程序模块，它的优点是单个功能明确的程序模块的设计和调试比较方便，容易完成，一个模块可以为多个程序所共享。其缺点是各个模块的连接有时有一定难度。

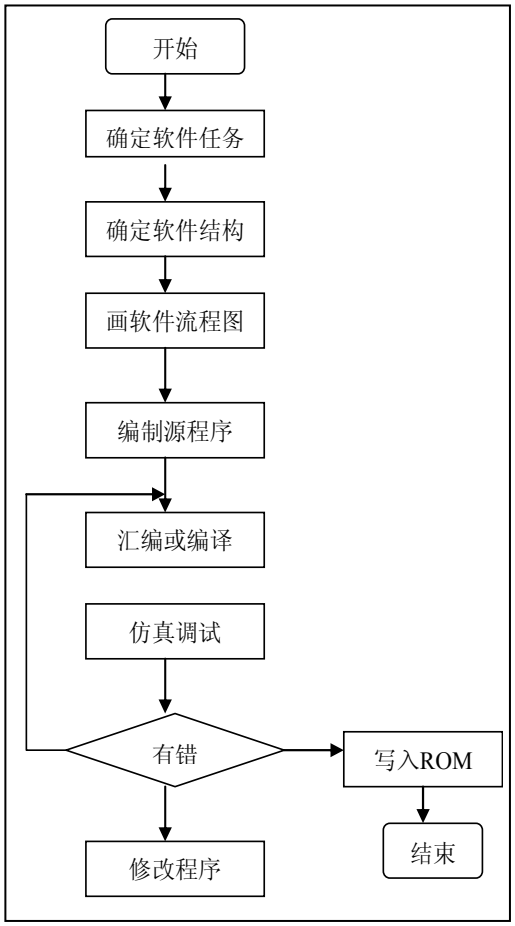
（2）自顶向下的程序设计：自顶向下的程序设计时，先从主程序开始设计，其从属的子程序可以用符号代替，主程序编制好后再编制各从属程序。它的优点是比较符合于人们的日常思维，设计、调试和连接同时按一个线索进行，程序错误可以较早的发现。缺点是上一级的程序错误将对整个程序产生影响，一处修改可能引起对整个程序的全面修改。

3）程序设计

确定好软件结构之后，便可着手进行程序设计，程序设计分为以下几个步骤：

（1）建立数学模型：根据设计任务，描述出各输入变量和各输出变量之间的数学关系。

图 10.2-1 软件设计流程图



（2）绘制程序流程图：通常在编制程序之前先绘制程序流程图，以简明直观的方式对任务进行描述。画程序流程图时先画粗框图，然后画每一部分的细致流程。

（3）程序的编制：程序编制之前要先选择合适的程序编制语言，汇编语言是较为常用的单片机程序语言，用汇编语言编写程序代码精简，执行速度快，但进行大量数据运算时，编写难度大大增加，因此，在有大量数据运算时，可以采用 C51 编写。

程序编写时，应选择合适的数据结构和控制算法，并注意系统的存储空间分配、系统硬件资源的合理分配与使用以及子程序的入/出口参数的设置与传递。

4）软件装配

各程序模块编辑之后，需进行汇编或编译、调试，当满足设计要求后，将各程序模块按照软件结构设计的要求连接起来，即为软件装配。在软件装配时，应注意软件接口。

10.2.4 可靠性设计

产品的可靠性通常是指在规定的条件（环境条件如温度、湿度、振动，供电条件等）下，在规定的时间内（平均无故障时间）完成规定功能的能力。由于测控系统的工作环境往往比较恶劣和复杂，单片机应用系统的可靠性、安全性就成为一个非常突出的问题，因此，必须注意系统的抗干扰设计。

1）干扰产生原因

凡是能产生一定能量，可以影响到周围电路正常工作的媒体都可认为是干扰源。干扰有的来自外部，有的来自内部。一般来说，干扰源可分为以下三类：

（1）自然界的宇宙射线，太阳黑子活动，大气污染及雷电因素造成的；（2）物质固有

的，即电子元器件本身的热噪声和散粒噪声；（3）人为造成的，主要是由电气和电子设备引起。

单片机系统的噪声干扰产生的原因主要有以下几个：

（1）电路性干扰。电路性干扰是由于两个回路经公共阻抗耦合而产生的，干扰量是电流；（2）电容性干扰。电容性干扰是由于干扰源与干扰对象之间存在着变化的电场，从而造成了干扰影响，干扰量是电压；（3）电感性干扰。电感性干扰是由于干扰源的交变磁场在干扰对象中产生了干扰感应电压。而产生感应电压的原因则是由于在干扰源中存在着变化电流；（4）波干扰。波干扰是传导电磁波或空间电磁波所引起的。空间电磁波的干扰量是电场强度和磁场强度。传导波的干扰量是传导电流和传导电压。

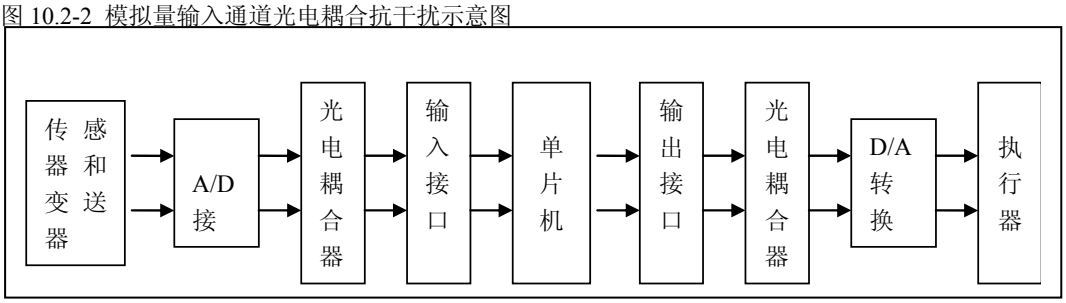
单片机应用系统的主要干扰渠道：空间干扰、过程通道干扰、供电系统干扰。应用于工业生产过程中的单片机应用系统中，空间干扰一般远小于后两者，因此应重点防止供电系统与过程通道的干扰。

提高单片机系统本身的可靠性，可以通过模拟输入通道抗干扰技术、过程通道抗干扰技术、供电系统抗干扰技术以及软件抗干扰技术实现。

2）模拟输入通道的抗干扰技术

（1）模拟量输人回路加入 RC 滤波器，以减小工频干扰信号对输入信号的影响；（2）光电耦合器隔离，在模拟通道使用光电耦合器要按照如图 10.2-2 的安排设计；（3）适当选用 A/D 芯片。在干扰严重的场合，可选用双积分式 A/D 转换器。要求转换速度快的场合，要选用逐次逼近方式的转换器。

图 10.2-2 模拟量输入通道光电耦合抗干扰示意图



3）过程通道的干扰与抑制

过程通道的干扰主要来自于长线传输。单片机应用系统中，从现场信号输出的开关信号或从传感器输出的微弱模拟信号，经传输线送入单片机，信号在传输线上传输时，会产生延时、畸变、衰减及通道干扰。

对过程通道的干扰，可以采取以下抗干扰措施：

（1）采用隔离技术：常用的隔离技术有光电隔离、变压器隔离、继电器隔离和布线隔离等。典型的信号隔离是光电隔离。其优点是能有效地抑制尖峰脉冲及各种噪声干扰，从而使过程通道上的信噪比大大提高。

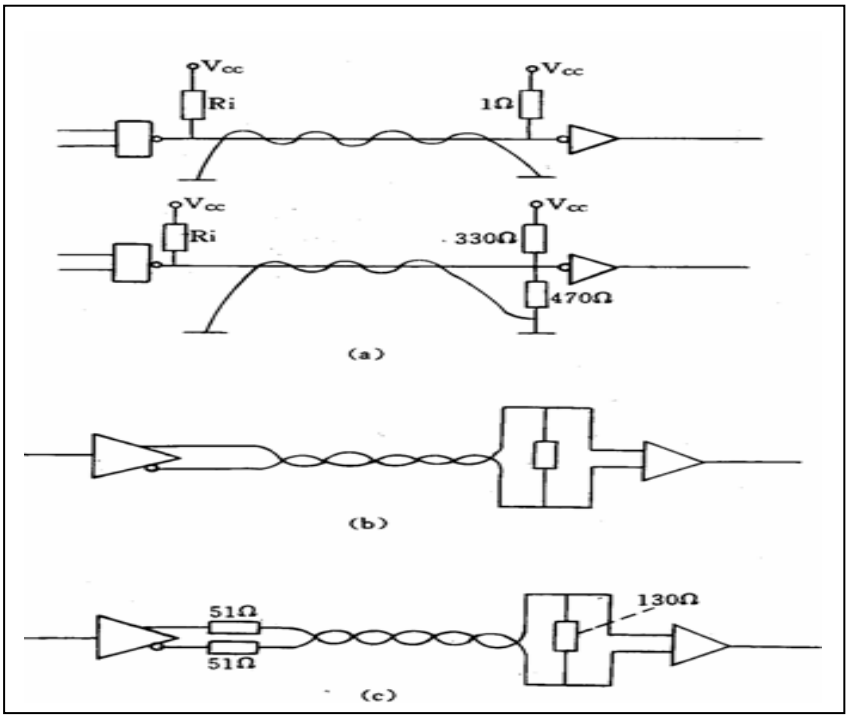
（2）采用屏蔽措施：屏蔽是用来隔离空间辐射的，把噪声特别大的器件用金属罩罩起来可以减少噪声源对单片机系统的干扰，对于容易受干扰的模拟电路（如高灵敏度的弱信号放大电路），也可以屏蔽起来。

（3）采用双绞线传输：双绞线能使各个小环路的电磁感应干抗相互抵消。其特点是波阻抗高、抗共模噪声能力强，但频带较差。在数字信号传输过程中，根据传送距离的不同，双绞线使用方法也有所不同，如图 10.2-3 所示。

当传送距离在 5m 以下时，发送和接收端连接负载电阻，如图 10.2-3(a)，若发送侧为集电极开路驱动，则接收侧的集成电路用施密特型电路，抗干扰能力更强。

当用双绞线作远距离传送数据时，或有较大噪声干扰时，可使用平衡输出的驱动器和平衡输入的接收器。发送和接收信号端都要接匹配电阻，如图 10.2-3(b)、（c）所示。

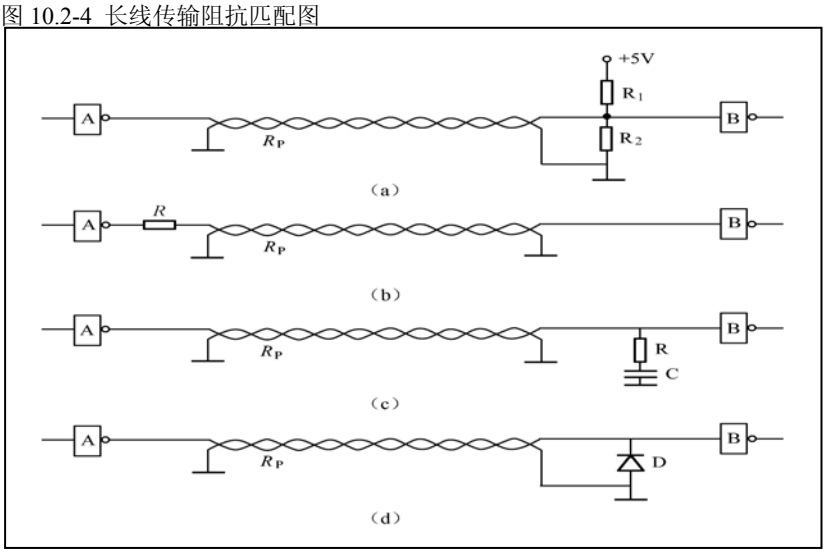
图 10.2-3 双绞线平衡传输图



注释：（a）传输距离在 5 米以下的双绞线， （b）传输距离在 10 米左右的双绞线，（c）传输距离在几十米的双绞线

（4）采用长线传输的阻抗匹配：长线传输时如匹配不好，会使信号产生反射，从而形成严重的失真。传输线阻抗的匹配如图 10.2-4 所示，有 4 种形式。

图 10.2-4 长线传输阻抗匹配图



注释：（a）终端并联阻抗匹配，（b）始端串联阻抗匹配，（c）终端并联隔直流匹配，（d）终端接钳位二极管匹配

① 终端并联阻抗匹配：如图 10.2-4（a）所示， RP= R1//R2 ，其特点是终端阻值低，降低了高电平的抗干扰能力。

② 始端串联匹配：如图 10.2-4（b）所示，匹配电阻 R 的取值为 RP 与 A 门输出低电平

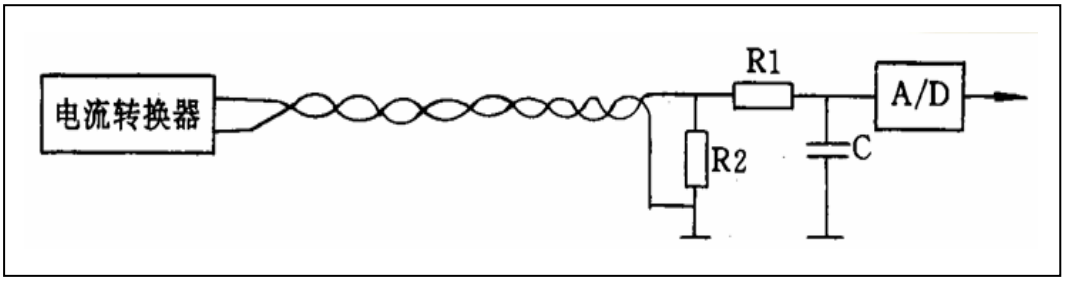
的输出阻抗 ROUT（约 20Ω）之差值，其特点是终端的低电平抬高，降低了低电平的抗干扰能力。

③ 终端并联隔直流匹配：如图 10.2-4（c）所示，R=Rp，其特点是增加了对高电平的抗干扰能力。

④ 终端接钳位二极管匹配：如图 10.2-4（d）所示，利用二极管 D 把 B 门输入端低电平钳位在 0.3V 以下。其特点是减少波的反射和振荡，提高动态抗干扰能力。

另外，长线传输时，用电流传输代替电压传输，可获得较好的抗干扰能力。如图 10.2-5 所示，从电流转换器输出 0-10mA（或 4-20mA）电流，在接收端并上 500Ω（或 lkΩ）的精密电阻，将此电流转换为 0-5V（或 1-5V）的电压，然后送人 A/D 转换器。若在输出端采用光电耦合器输出驱动，也会获得同样的效果。此种方法可减少在传输过程中的干扰，提高传输的可靠性。

图 10.2-5 长线电流传输示意图



4）供电系统干扰及其抑制

供电系统干扰分为：

（1）过压、欠压、停电，使用各种稳压器和不间断电源 UPS；（2）浪涌、下陷、降出快速响应的交流电压调压器；（3）尖峰电压　使用具有噪声抑制能力的交流稳压器或隔离变压器；（4）射频干扰。

为了提高供电系统的抗干扰，可以从以下几个方面考虑：（1）交流进线端加交流滤波器，可滤掉高频干扰，如电网上大功率设备启停造成的瞬间干扰；（2）要求高的系统加交流稳压器；（3）采用具有静电屏蔽和抗电磁干扰的隔离电源变压器；（4）采用集成稳压块两级稳压；（5）主电路板采取独立供电，其余部分分散供电，避免一处电源有故障引起整个系统颠覆；（6）直流输出部分采用大容量电解电容进行平滑滤波。

5）其他硬件抗干扰措施

（1）对信号整形，可采用斯密特电路整形。

（2）组件空闲输入端的处理

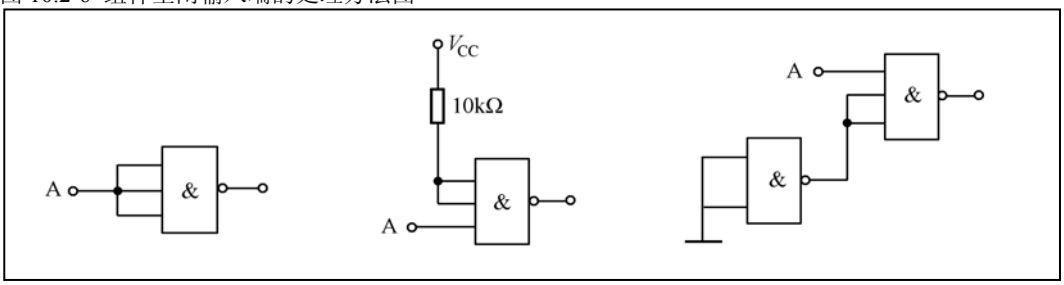
组件空闲输入端的处理方法如 10.2- 6 图所示。其中，图（a）所示的方法最简单，但增加了前级门的负担。图（b）所示的方法适用于慢速、多干扰的场合。图（c）利用印刷电路板上多余的反相器，让其输入端接地，使其输出去控制工作门不用的输入端。

（3）机械触点，接触器、可控硅的噪声抑制

① 开关、按钮、继电器触点等在操作时应采取去抖处理；② 在输入/输出通道中使用接触器、继电器时，应在线圈两端并接噪声抑制器，继电器线圈处要加装放电二极管；③ 可控硅两端并接 RC 抑制电路，可减小可控硅产生的噪声。

（4）印刷电路板（PCB）设计中的抗干扰问题

图 10.2-6 组件空闲输入端的处理方法图



合理选择 PCB 板的层数，大小要适中，布局、分区应合理，把相互有关的元件尽量放得靠近一些。印刷导线的布设应尽量短而宽，尽量减少回路环的面积，以降低感应噪声。导线的布局应当是均匀的、分开的平行直线，以得到一条具有均匀波阻抗的传输通路。应尽可能地减少过孔的数量。在 PCB 板的各个关键部位应配置去耦电容。要将强、弱电路严格分开，尽量不要把它们设计在一块印刷电路板上。电源线的走向应尽量与数据传递方向一致，电源线、地线应尽量加粗，以减小阻抗。双面布线时，应尽量使两面的线条互相垂直，另外，不要在印制板上留下大片空白的铜箔曾，要用地线将没有走线的地方铺满。

（5）地线设计

地线结构大致有保护地、系统地、机壳地（屏蔽地）、数字地、模拟地等。

在设计时，数字地和模拟地要分开，分别与电源端地线相连；屏蔽线根据工作频率可采用单点接地或多点接地；保护地的接地是指接大地。不能把接地线与动力线的零线混淆。

此外，应提高元器件的可靠性，注意各电路之间的电平匹配，总线驱动能力要符合要求，单片机的空闲端要接地或接电源，或者定义成输出。室外使用的单片机系统或从室外架空引入室内的电源线、信号线，要防止雷击，常用的防雷击器件有：气体放电管，TVS（瞬态电压抑制器）等。

6）软件的抗干扰设计

在硬件抗干扰的基础上，采用软件抗干扰技术加以补充，作为硬件措施的辅助手段，可以起到良好的抗干扰效果。由于软件抗干扰方法简单、灵活、节省硬件资源，因此它成为系统抗干扰设计必不可少的手段。常用的软件抗干扰技术有软件陷阱、指令冗余、设置特征标志和软件数字滤波等

（1） 实时数据采集系统的软件抗干扰采用软件数字滤波。

常用的方法有以下几种：① 算术平均值法：对一点数据连续采样多次（可取 3～5 次），以平均值作为该点的采样结果。这种方法可以减少系统的随机干扰对采集结果的影响；② 比较舍取法：对每个采样点连续采样几次，根据所采样数据的变化规律，确定取舍办法来剔除偏差数据。例如，“采三取二”，即对每个采样点连续采样三次，取两次相同数据作为采样结果；③ 中值法：对一个采样点连续采集多个信号，并对这些采样值进行比较，取中值作为该点的采样结果；④ 一阶递推数字滤波法：利用软件完成 RC 低通滤波器的算法。

其公式为：

$$Yn=QXn+（1－Q）Yn－1$$

其中：Q ——数字滤波器时间常数；

Xn——第 n 次采样时的滤波器的输入；

Yn－1——第 n－1 次采样时的滤波器的输出。

Yn——第 n 次采样时的滤波器的输出。

注意：选取何种方法必须根据信号的变化规律予以确定。

（2）开关量控制系统的软件抗干扰可采取指令冗余、设置当前输出状态寄存单元等软件抗干扰措施。软件冗余技术，就是多次使用同一功能的软件指令，以保证指令执行的可靠性，这从以下几个方面考虑。

①采取多次读入法，确保开关量输人正确无误，重要的输人信息利用软件多次读入，比较几次结果一致后再让其参与运算。对于按钮和开关状态读入时，要配合软件延时可消除抖动和误动作；②不断查询输出状态寄存器，及时纠正输出状态。设置输出状态寄存器，利用软件不断查询，当发现和输出的正确状态不一致时，及时纠正，防止由于干扰引起的输出量变化导致设备误动作；③对于条件控制系统，把对控制条件的一次采样、处理控制输出改为循环地采样、处理输出。这种方法对于惯性较大的控制系统具有良好的抗偶然干扰作用；④为防止计算错误，可采用两组计算程序，分别计算，然后将两组计算结果进行比较，如两次计算结果相同，则将结果送出。如出现误差，则再进行一次运算，重新比较，直到结果相同。

（3）程序运行失常的软件对策

程序运行失常是指由于系统干扰可能破坏程序指针 PC，PC 一旦失控，使程序“乱飞”，可能进人非程序区，造成系统运行的一系列错误。通过设置软件陷阱或加入程序监控定时器，可防止程序“乱飞”。

软件陷阱是指将捕获的“跑飞”程序引向复位入口地址 0000H 的指令。设置方法：

① 在 EPROM 中，非程序区设置软件陷阱，软件陷阱一般 1KB 空间有 2～3 个就可以进行有效拦截。指令如下：

```
NOP
NOP
LJMP 0000H
```

② 在未使用的中断服务程序中设置软件陷阱，能及时捕获错误的中断。指令如下：

```
NOP
NOP
RETI
```

利用设置软件陷阱的办法虽在一定程度上解决了程序“飞出”失控问题，但不能有效地解决死循环问题。设置程序监视器（Watchdog 看门狗）可比较有效地解决死循环问题。程序监视器系统有的采用软件解决，大部分都是采用软硬件相结合的办法。下面简要介绍两种方法：

①在程序一开始就启动定时器工作，在主程序中增设定时器赋值指令，使该定时器维持在非溢出工作状态。定时时间要稍大于程序一次循环的执行时间。程序正常循环执行一次给定时器送一次初值，使其不能溢出。但若程序失控，定时器则计满溢出中断，在中断服务程序中使主程序自动复位又进入初始状态。

②利用单稳触发器构成程序监视器方法：利用软件经常访问单稳态电路，一旦程序有问题，CPU 不能照常访问，单稳电路则产生翻转脉冲使单片机复位，程序重新开始执行。

10.3 单片机应用系统的调试、测试

单片机应用系统的软、硬件制作完成后，必须反复进行调试、修改，直至完全正常工作，经过测试，功能完全符合系统性能指标要求，应用系统设计才算完成。单片机应用系统的调试包括硬件调试和软件调试，且两者应该协调统一，一般的方法是先排除明显的硬件故障，再进行综合调试，排除可能的软、硬件故障。

10.3.1 硬件调试

硬件调试是利用开发系统、基本测试仪器（万用表、示波器等），通过执行开发系统的有关命令或运行适当的测试程序（或是与硬件有关的部分程序段）来检查用户硬件系统中存在的问题。硬件调试分为静态检查和动态调试两步。

1）静态调试

静态调试是在系统未工作时的一种硬件检查，分为静态观察、万用表测试和通电检查三步。

（1）静态观察

单片机应用系统中的大部分电路安装在印制电路板上，因此对每一块加工好的印制电路板都要进行仔细检查，检查印制板是否有断线，是否有毛刺，是否有焊盘粘连或脱落，过孔金属化程度是否足够等，发现问题及时弥补。如印制电路板无质量问题，则需要根据硬件电路图进行元器件的安装和焊接，在元器件装配过程中，一定要核对元器件的型号、极性、安装是否正确，并检查有无漏焊、虚焊等现象。

（2）万用表测试

用万用表进行焊接后的线路检查，排除断路或短路，并检查电源和地线之间有无短路。短路现象一定要在系统加电之前查出并解决，如果电源与地之间短路，一旦加电，系统所有器件或设备都有可能被毁坏，因此，对电源与地的处理，在整个系统调试及今后的运行中都要格外小心。

（3）通电测试

通电检查时，可以模拟各种输入信号分别送入电路的各有关部分，观察 I/O 口的动作情况，查看电路板上有无元件过热、冒烟、异味等现象，各相关设备的动作是否符合要求，整个系统的功能是否符合要求。

2）动态调试

在上述静态检查都无误的情况下，即可将用户系统与单片机开发系统用仿真电缆连接起来，利用单片机开发系统完成对用户系统的测试，这个过程也称为联机仿真或联机调试。

动态调试的一般方法是由近及远，由分到合。由近及远是按照信号的流向进行逐级调试，由分到合是按照逻辑功能将用户系统硬件电路分为若干模块，然后进行各个模块分别调试，最后进行综合调试。

10.3.2 软件调试

软件调试是通过对用户程序的汇编、连接、执行来发现程序中存在的语法错误与逻辑错误并加以排除纠正的过程。

软件调试的一般方法是先独立后联机、先分块后组合、先单步后连续。

1）先独立后联机

单片机应用系统中的软件与硬件是密切相关、相辅相成的。但是，当软件对被测试参数进行加工处理或作某项事务处理时，往往是与硬件无关的，这样，就可以通过对用户程序的仔细分析，把与硬件无关的、功能相对独立的程序段抽取出来，形成与硬件无关和依赖于硬件的两大类用户程序块。

调试时，先调试与硬件无关的程序块，此时可以通过开发系统进行相应的参数设置，通过观察端口或存储器数据判断程序执行结果的正确与否；当与硬件无关的程序块全部调试完成后，就可以将仿真机与主机、用户系统连接起来，进行系统联调。在系统联调中，先对依赖于硬件的程序块进行调试，调试成功后，再进行两大程序块的有机组合及总调试。

2）先分块后组合

如果用户系统规模较大、任务较多，即使先行将用户程序分为与硬件无关和依赖于硬件两大部分，这两部分程序仍较为庞大，采用笼统的方法从头至尾调试，既费时间又不容易进行错误定位，所以常规的调试方法是分别对两类程序块进一步采用分模块调试，以提高软件调试的有效性。

在分模块调试时所划分的程序模块应基本保持与软件设计时的程序功能模块或任务一

致。每个程序模块调试完后，将相互有关联的程序模块逐块组合起来加以调试，以解决在程序模块连接中可能出现的逻辑错误。对所有程序模块的整体组合是在系统联调中进行的。

由于各个程序模块通过调试已排除了内部错误，所以软件总体调试的错误就大大减少了，调试成功的可能性也就大大提高了。

3）先单步后连续

调试好程序模块的关键是实现对错误的正确定位。准确发现程序(或硬件电路)错误的最有效方法是采用单步加断点运行方式调试程序。单步运行可以了解被调试程序中每条指令的执行情况，分析指令的运行结果，以便知道该指令执行的正确性，并进一步确定是由于硬件电路错误、数据错误还是程序设计错误等引起了该指令的执行错误，从而发现、排除错误。

但是，所有程序模块都以单步方式查找错误的话，又比较费时费力，所以为了提高调试效率，一般采用先使用断点运行方式将故障定位在程序的一个小范围内，然后针对故障程序段再使用单步运行方式来精确定位错误所在，这样就可以做到调试的快捷和准确。一般情况下，单步调试完成后，还要做连续运行调试，以防止某些错误在单步执行的情况下被掩益。

有些实时性操作(如中断等)利用单步运行方式无法调试，必须采用连续运行方法进行调试。为了准确地对错误进行定位，可使用连续加断点运行方式调试这类程序，即利用断点定位的改变，一步步缩小故障范围，直至最终确定出错误位置并加以排除。

10.3.3 系统联合调试

当硬件和软件调试完成之后，就可以进行系统软、硬件联合调试。对于有电气控制负载的系统，应先试验空载，空载正常后再试验负载情况。系统调试的任务是排除软、硬件中的残留错误，使整个系统能够完成预定的工作任务，达到要求的性能指标。

系统调试成功之后，就可以将程序通过专用程序固化器固化到 ROM 中。将固化好程序的 ROM 插回到应用系统电路板的相应位置，即可脱机运行。系统试运行要连续运行相当长的时间（也称为烤机），以考验其稳定性。并要进一步进行修改和完善处理。

10.3.4 现场调试及性能测试

一般情况厂，通过系统联调，用户系统就可以按照设计要求常工作了。但在某些情况下，由于系统实际运行的环境较为复杂(如环境干扰较为严重、工作现场含腐蚀性气体等)，在实际现场工作之前，环境对系统的影响无法预料，只能通过现场运行调试来发现问题，找出相应的解决方法。另外，有些用户系统的调试是在用模拟设备代替实际监测、控制对象的情况下进行的，这就更有必要进行现场调试，以检验系统在实际工作环境中工作的正确性。

总之，现场调试对用户系统的调试来说足最后必须的一个过程，只有经过现场调试的系统才能保证其可靠地工作。现场调试仍需利用开发系统来完成，其调试方法与联合调试类似。

整个调试过程进行完毕后，一般需进行单片机系统功能的测试，上电、掉电测试，老化测试，静电放电（ElectroStatic Discharge，ESD）抗扰度和电快进瞬变脉冲群（Electrical Fast Transient，EFT）抗扰度等测试。可以使用各种干扰模拟器来测试单片机系统的可靠性，还可以模拟人为使用中可能发生的破坏情况。

经过调试、测试后，若系统完全正常工作，功能完全符合系统性能指标要求，则一个单片机应用系统的研制过程全部结束。

10.4 单片机应用系统举例

10.4.1 单片机在控制系统中的应用

单片机的一个广泛应用领域就是控制系统，包括室内控制，室外控制等。本小节以单片机在大棚环境控制系统中的应用为例，简要介绍单片机在此类控制系统中的应用。

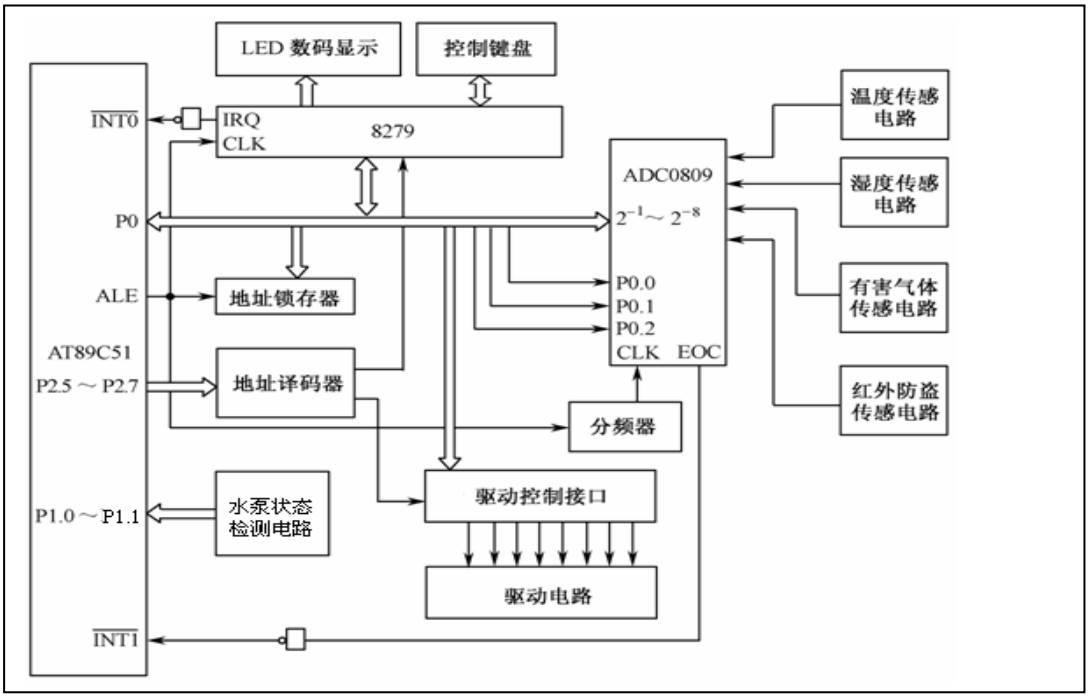
1） 设计思想

随着人们对生活质量的要求不断提高和科技的发展，利用大棚进行蔬菜种植、花卉栽培以及各种植物繁育变得越来越普及，对大棚内的温湿度、有害气体等的监测也也变得比较重要。大棚环境控制系统一般通过传感电路不断循环检测大棚内温度、土壤湿度、有害气体（如CO_2）浓度等环境参数，然后与由控制键盘预置的参数临界值相比较，从而作出开/关水泵、启/停换气扇、升/降温（湿）等判断，再结合水泵状态检测电路所检测到的水泵状态，发出一系列的控制命令，完成土壤过干则自动浇水、室内 CO_2 超标则自动开启换气扇、恒温（湿）等自动控制功能。用户还可通过控制键盘，手动完成上述的各个控制作，并可以选择所显示参数的种类等。

2）系统组成和部分电路设计

控制系统主要由控制器、数据检测电路、A/D 转换电路、驱动控制接口电路以及驱动电路等组成。其系统原理图如图 10.4-1 所示。

图 10.4-1 控制系统原理图



单片机用美国 Atmel 公司的 AT89C51 单片机，利用 89C51 的 P0 口采集数据，利用 P1.0～P1.3 作为水泵状态检测端口，检测水泵目前的工作状态。

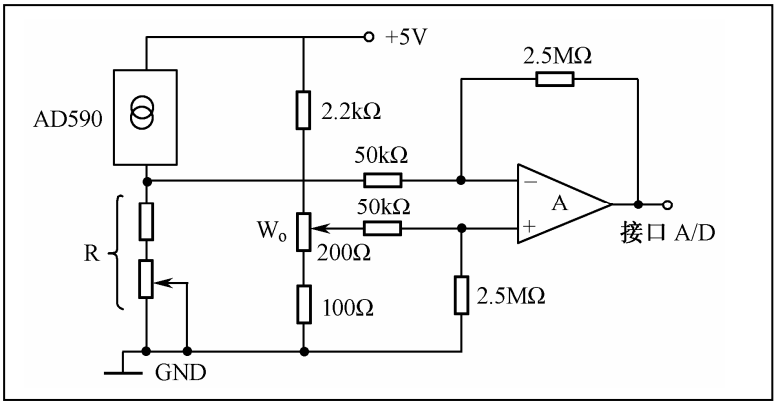
数据检测电路由温度传感电路、湿度传感电路、气体传感电路、红外防盗传感器四个部分组成。在此只以温度传感电路为例进行设计。

根据温度检测的要求，温度传感器选用集成温度传感器 AD590（测温范围为—55℃～+150℃）。测量电路如图 10.4-2 所示。

传感器的采集信号经过数据处理电路，必须通过 A/D 转换器由模拟信号转换为数字信号

才能与单片机连接。本系统中有 4 路模拟输入，加之对系统的采样速度和精度要求不是很高，因此 A/D 转换器选用了 8 位 8 输入的 ADC0809，89C51 通过中断方式读取 A/D 转换的数据。通过 A/D 转换实现的数据采集电路如图 10.4-3 所示，设定 A/D 转换器 8 路模拟输入的地址

图 10.4-2 测量电路图



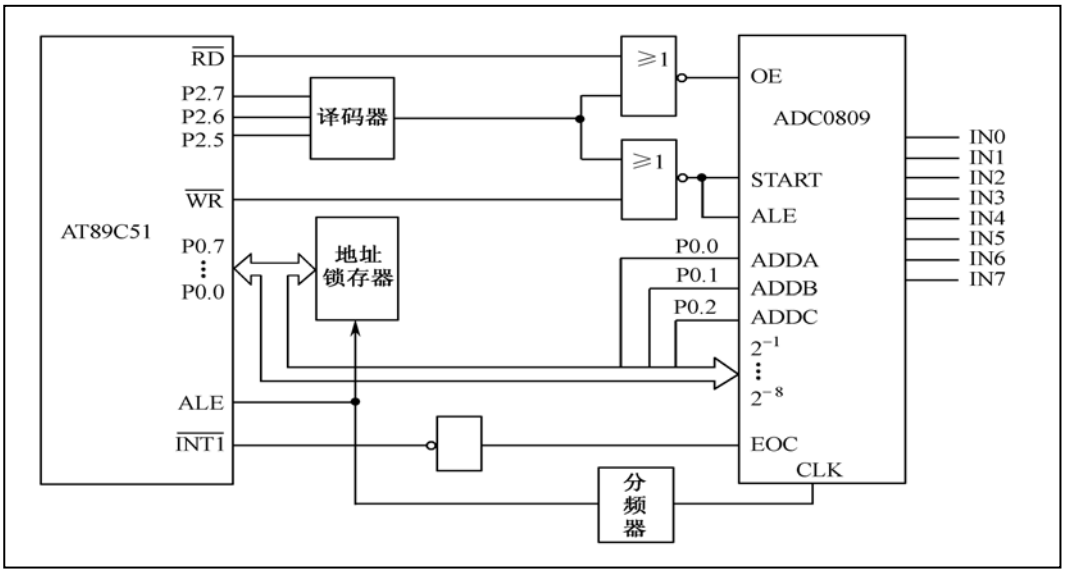
分别为 7FF8H—7FFFH，本设计中温度检测、湿度检测、有害气体检测和红外防盗报警检测 4 路传感器输入通道的地址分别为 7FF8H，7FF9H，7FFAH，7FFBH。

根据驱动信号与所控对象的关系，将系统的驱动电路分解为水泵驱动电路、换气扇驱动电路、温度调节驱动电路、湿度调节驱动电路和报警驱动电路等，分别用它们去控制 1 个对象。

水泵状态检测电路是通过检测对应控制管脚的电平，判断控制水泵的继电器的状态，从而得到目前水泵是抽水还是排水的信息，再根据湿度传感器的检测结果，对水泵下一步的动作进行控制。

键盘输入及显示电路采用 Intel 公司生产的 8279 通用可编程键盘、显示器接口芯片。可实现对键盘和显示器的自动扫描，并识别键盘上闭合键的键号。

图 10.4-3 数据采集电路图



对于控制键盘，采用微动开关制作，通过控制键盘，用户可设置各环境参数的临界值、随意选择所显示参数的种类、直接控制继电器从而控制水泵的的开/关或抽水/排水、换气扇的启/停、温（湿）度的升/降等。

3）软件设计

控制系统的软件主要由一个主程序和两个中断服务程序等组成。主程序的主要作用是在系统复位后对系统进行初始化，设置 8279，ADC0809 等的工作方式和初始状态，设置各中断的优先级别并开中断，首次启动 A/D 转换等，然后向 8279 循环送显示字符，进行显示。程序框图如图 10.4-4 所示。

键中断服务程序的主要作用是在 AT89C51 响应 $\overline{INT0}$ 中断（有键按下，则产生该中断）后，读出键值，并根据键值依序发出相应的控制命令字，完成相应的控制功能。该中断应设为高优先级。程序框图如图 10.4-5 所示。

循环检测中断服务程序的主要作用是在 89C51 响应 $\overline{INT1}$ 中断后，将 A/D 转换结果送相应缓冲区，然后判断该转换结果是否在该参数的上、下限值之间，并根据判断结果按序发出相应的控制命令字，完成相应的控制、报警功能。然后重新选择被转换量，再次启动 A/D 转换后，返回主程序。该中断应设为低优先级，并设为电平触发方式。程序流程图如图 10.4-6 所示。

图 10.4-4 主程序流程图

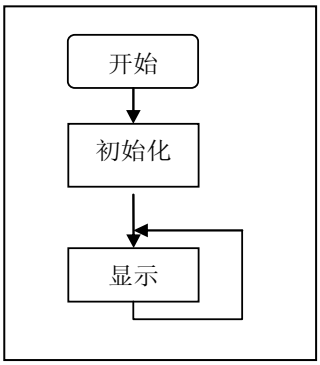


图 10.4-5 键中断服务子程序流程图

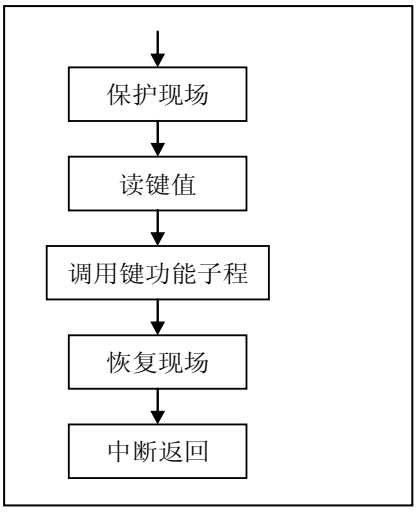
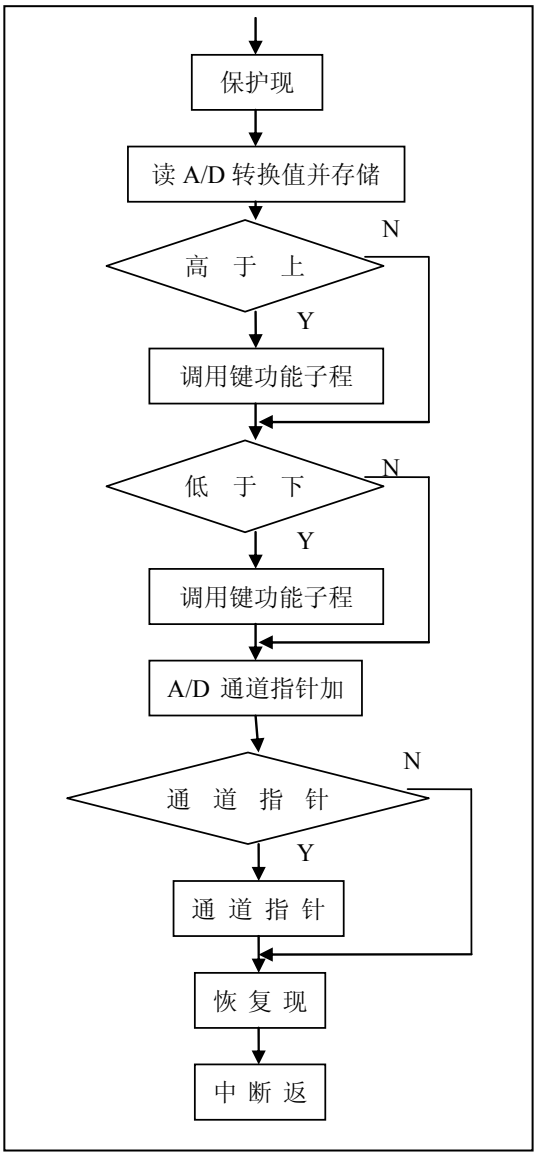


图 10.4-6 循环检测中断服务子程序流程图



10.4.2 单片机在里程、速度计量中的应用

设计要求：利用单片机实现的自行车里程/速度计能自动显示自行车行驶的总里程数及

自行车行驶速度，具有超速信号提醒功能，里程数据自动记忆。也可应用于电动自行车、摩托车、汽车等机动车仪表上。

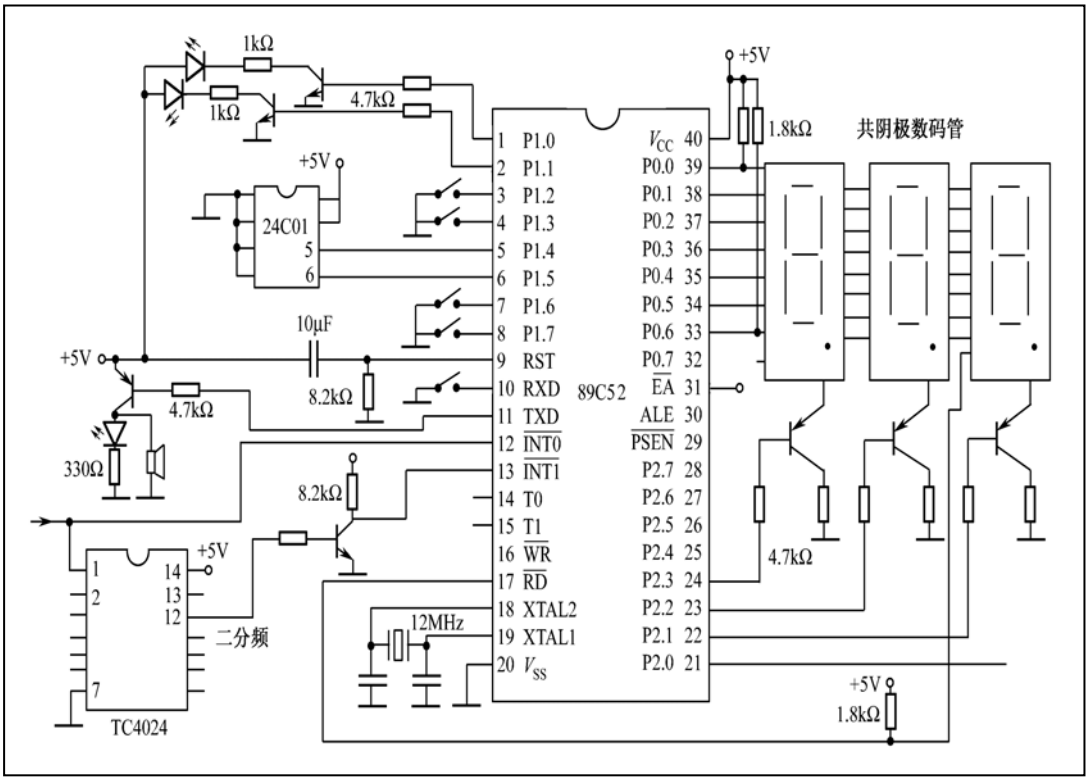
1)总体设计

控制器采用 AT89C52 单片机，速度及里程传感器采用霍尔元件，显示器通过 AT89C52 的 P0 口和 P2 口扩展。外部存储器采用 E2PROM 存储器 AT24C01，用于存储里程和速度等数据。并用控制器来控制里程/速度指示灯，里程指示灯亮时，显示里程；速度指示灯亮时，显示速度。超速报警采用扬声器，用一个发光二极管来配合扬声器，扬声器响时，二极管亮，表明超速。

2)硬件电路设计

电路原理图如图 10.4-7 所示。P0 口和 P2 口用于七段 LED 显示器的段码及扫描输出。在显示里程时，第三位小数点用 P3.7 口（）控制点亮。P1.0 口和 P1.1 口分别用于显示里程状态和速度状态。P1.2，P1.3，P1.6 和 P1.7 口分别用于设置轮圈的大小。P3.0 口的开关用于确定显示的方式。当开关闭合时，显示速度；断开时，显示里程。外中断用于对轮子圈数的计数输入，轮子每转一圈，霍尔传感器输出一个低电平脉冲。外中断用于控制定时器 T1 的启停，当输入为 0 时关闭定时器。此控制信号是将轮子圈数的计数脉冲经二分频后形成，这样，每次定时器 T1 的开启时间正好为轮子转一圈的时间，根据轮子的周长就可以计算出自动车的速度。P1.4 口和 P1.5 口用于 E2PROM 存储器 24C01 的存取控制。11 脚（TXD）输出用于速度超速时的报警。

图 10.4-7 硬件电路原理图



3)软件设计

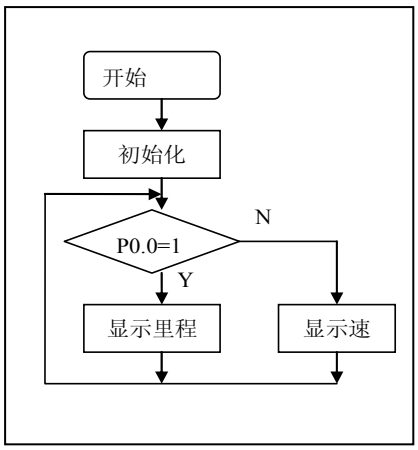
软件主要包括：主程序、初始化程序、里程计数子程序、数据处理子程序、计数器中断服务程序、E2PROM 存取程序、显示子程序。

（1）主程序

根据 P0 口的开关状态切换显示状态，即选择里程显示和速度显示。其流程图如图 10.4-8

所示。

图 10.4--8 主程序流程图



程序如下：

```
    ORG   0000H
          LJMP  START              ;  跳至主程序
     ORG   0003H                   ;  外中断 0 中断程序入口
          LJMP   INTEX0            ;  跳至 INTEX0 中断服务程序
          ORG   000BH              ;  定时器 T0 中断程序入口
          RETT1                    ;  中断返回
    ORG    0013H                   ;  外中断 1 中断入口
          LJMP   INTEX1            ;  跳至 INTEX1 中断服务程序
          ORG    001BH             ;  定时器 T1 中断程序入口
          LJMP   INTT1             ;  跳至 INTT1 中断服务程序
          ORG    0023H             ;  串口中断入口地址
          RETI                     ;  中断返回
          ORG    002BH             ;  定时器 T2 中断入口地址
          RETI                     ;  中断返回
          ORG       0050H
 START:  LCALL   CLEARMEN          ;  上电初始化
 START1: JB   P0.4, DISPLAYS       ;  P0.4=1，则显示里程
          LCALL    DISPLAYV        ;  P0.4=0，显示速度
START2:  SJMP    START             ;  转 START 循环
```

（2）初始化程序

初始化程序主要功能是将 T1 设为外部控制定时器方式，外中断 $\overline{INT0}$ 及 $\overline{INT1}$ 设为边沿触发方式，将部分内存单元清 0，设置车轮周长值，开中断、启动定时器，将 AT24C01 中的数据调入内存中，设置车轮圈出错处理程序。

程序如下：

```
CLEARMEN:  MOV   TMOD, #90          ;  T1 为 16 位外部控制定时器
           MOV       SP, #75H       ;  堆栈在 75H 开始
           SETB      PX0            ;  外中断 0 优先级为 1
           SETB      IT0            ;  外中断 0 用边沿触发
           SETB      IT1            ;  外中断 1 用边沿触发
           MOV       A, #00H        ;  清 A
```

```
        MOV     20H, A          ;  清内存中特定单元
        MOV     6CH, A
        MOV     6DH, A
        MOV     70H, A
        MOV     71H, A
        MOV     72H, A
        MOV     73H, A
        MOV     60H, A
        MOV     61H, A
        MOV     62H, A
        MOV     63H, A          ;  清内存中特定单元
        DEC     A               ;  A 为#0FFH
        MOV     68H,A           ;  内存置数据#0FFH
        MOV     69H,A           ;  内存置数据#0FFH
        MOV     6AH,A           ;  内存置数据#0FFH
        MOV     6BH,A           ;  内存置数据#0FFH
        MOV   P1, A             ;  P1 口置 1
CLEAR1:JB     P1.2, KEY1        ;  根据 P1.2, P1.3, P1.6, P1.7 设置状态
        MOV     21H,#0FH        ;  22 英寸自行车周长系数
        LJMP    CLEAR2          ;  转 CLEAR2
KEY1:  JB    P1.3, KEY2
        MOV     21H, #12H       ;  24 英寸自行车周长系数
        LJMP    CLEAR2          ;  转 CLEAR2
KEY2:  JB     P1.6,KEY3
        MOV     21H,#14H        ;  26 英寸自行车周长系数
        LJMP    CLEAR2          ;  转 CLEAR2
KEY3:  JB  P1.7,ERR
        MOV     21H,#19H        ;  28 英寸自行车周长系数
CLEAR2:  SETB   TR1             ;  开定时器开关 T1
       SETB     EA              ;  开中断允许
          SETB  EX0             ;  开外中断
        SETB    ET1             ;  开定时中断 T1
        SETB    P3.1            ;  关报警器
             LCALL    VIICREAD  ;  将 E2PROM 中原里程数据调入内存
        RET                     ;  子程序返回
           ERR: CLP      P3.1   ;  轮周长设置出错，LED 灯闪烁提醒
      LCALL     DLSS5           ;  延时
      LJMP      CLEAR1          ;  重新初始化，等待轮周长设置开关合上
```

（3）里程计数子程序

外中断$\overline{INT0}$服务程序用于对输入的车轮圈数脉冲进行计数，为十六进制计数，用片内 RAM 的 60H 单元存储计数值的低位，62H 存储高位，计数一次后，对里程数据进行一次存储。

程序如下：

```
INTEX0:  PUSH  ACC              ;  累加器堆栈保护
         PUSH  PSW              ;  状态字堆栈保护
```

```
        INC     60H                 ;  圈加 1
        MOV    A, #00H              ;  清 A
        CJNE    A, 60H, INTEX00UT   ;  计数没溢出转 IN00UT
        INC     61H                 ;  溢出进位（61H 加 1）
        CJNE    A, 61H, INTEX00UT   ;  计数没溢出转 IN00UT
        INC     62H                 ;  溢出进位（62H 加 1）
IN00UT: LCALL   VIICWRITE           ;  里程数据存入 E2PROM
        SETB    EX1                 ;  开外中断 1
        POP     PSW                 ;  状态字恢复
        POP     ACC                 ;  累加器恢复
        RET1
```

（4）数据处理子程序

外中断服务程序用于处理轮子转动一圈后的计时数据，当标志位（00H）为 1 时，说明计数器溢出，放入最大值 0FFH；当标志位为 0 时，将计数单元（TL1,TH1,6CH, 60H）的值放入 68H～6BH 单元。

程序如下：

```
INTEX1:  PUSH   ACC                 ;  累加器堆栈保护
         PUSH   PSW                 ;  状态字堆栈保护
         CLR    EX1                 ;  关外中断 1
         JNB    00H, INTEX11        ;  溢出标志为 0 转 INTEX11
         MOV   TL1, #0FFH           ;  溢出时，计时单元赋#FFH（显示速度为 0）
         MOV   TH1, #0FFH
         MOV   6CH, #0FFH
         MOV   6DH, #0FFH
INTEX11: MOV   68H, TL1             ;  将时间计数值存入暂存单元 68H～6BH
         MOV   69H, TH1
         MOV   6AH, 6CH
         MOV   6BH, 6DH
         MOV   A, #00H              ;  清 A
         MOV   TL1, A               ;  计时单元置 0
         MOV   TH1, A
         MOV   6CH, A
         MOV   6DH, A
         CLR    00H                 ;  清溢出标志
         POP    PSW                 ;  状态字恢复
         POP    ACC                 ;  累加器恢复
         RET1                       ;  中断返回
```

（5）计数器中断服务程序

T1 计数单元由外中断进行控制，当计数器溢出时置溢出标志，不溢出时，使计时单元计数，存入存储器。程序略。

（6）E2PROM 存取程序

将外部信息写入 AT24C01 存储器，存入从 50H 起的单元中；把外部信息从 AT24C01 存储器中读出，送 CPU 进行处理。该段程序略。

（7）显示子程序

当显示里程时，先要将计数器中的数据进行运算，求出总里程，并送入里程显示缓冲区；当要显示速度时，要将轮子的周长和转一圈的时间相除，然后换算成 km/h（千米/小时），存入 70H～73H 单元，进行数据显示,该段程序略。

10.4.3 单片机简单应用示例

例 10.1 如图 10.4-9 所示，单片机扩展可编程接口芯片 8155，8155PA 口控制 8 只发光二极管，形成走马灯，每位点亮的时间为 0.1 秒。

8155 的端口地址如下：

命令口地址（COM8155）：FEF8H

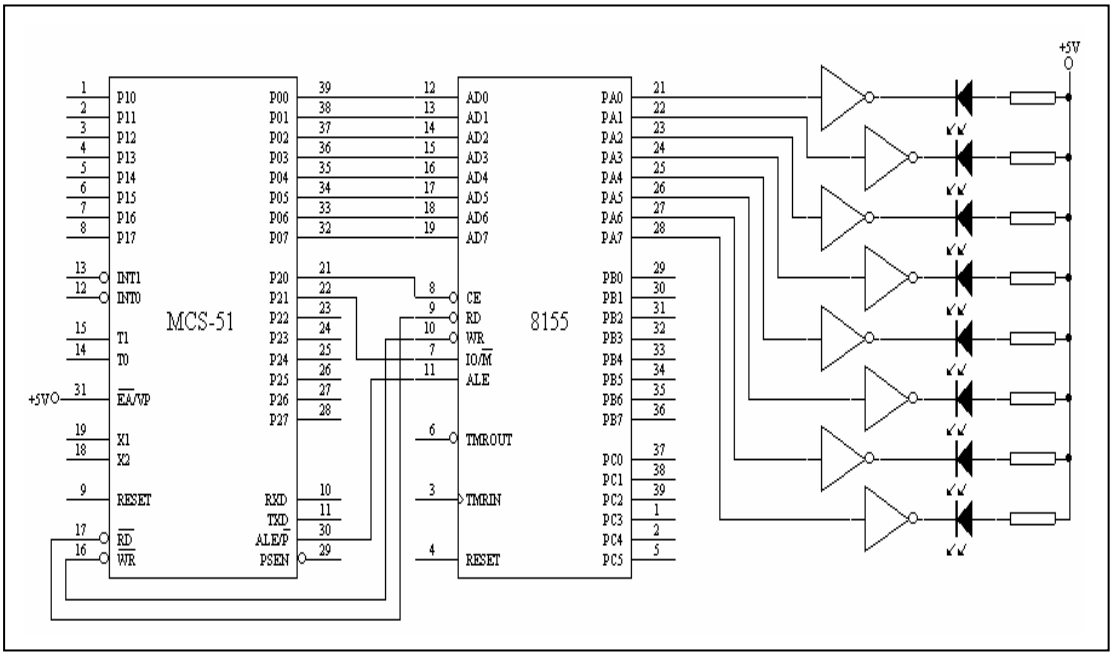
PA 口地址（PA8155）：FEF9H

PB 口地址（PB8155）：FEFAH

PC 口地址（PC8155）：FEFBH

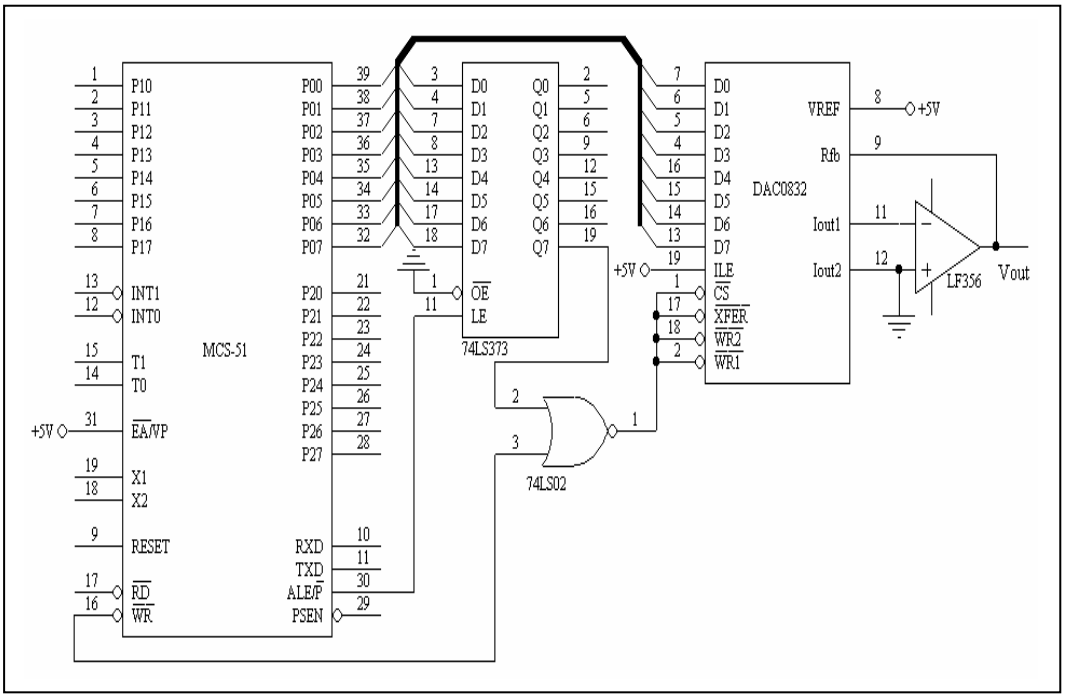
```
    #include “reg51.h”
    #include “absacc.h”
    #define COM8155 XBYTE[0xfef8]
    #define PA8155 XBYTE[0xfef9]
    void delay（void）              /*延时 1ms*/
    {
      TH0=－500/256；
      TL0=－500%256；
      TR0=1；
      while（!TF0）；
      TF0=0；
      TR0=0；
    }
main（）
    {
      char i；
      disp_word=0x01；    /*从第 1 位开始点亮*/
      COM8155=0x01；    /*初始化 8155*/
      do {
      PA8155= disp_word；      /*输出去点亮一位*/
      for（i=0；i<100；i++）    /*点亮 0.1 秒 */
{delay（）；}
        disp_word= disp_word<<1；    /*左移控制字，准备点亮下一位*/
        if（disp_word==0）disp_word=0x01；
      } whlie（1）；
    }
```

图 10.4-9 可编程接口芯片 8155 接口控制图



例 10.2 如图 10.4-10 所示，要求 DAC0832 输出锯齿波电压信号，信号周期自由，用 C 语言写出相应程序。

图 10.4-10 DAC0832 输出锯齿波电路图




```
#include “reg51.h”
#include “absacc.h”
#define DA0832    PBYTE[0x7f]
main（）
{
```

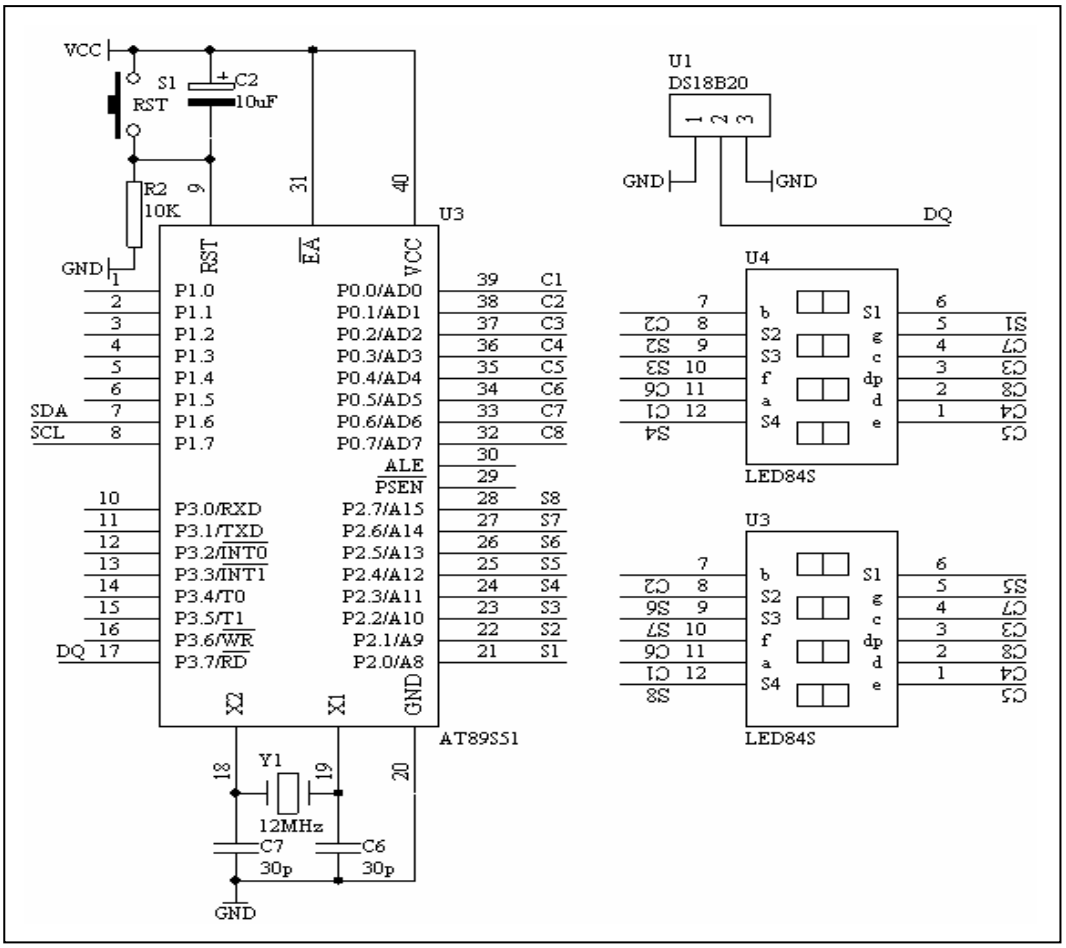
```
    char i;
    do {
    for（i=0；i<255；i++）
        {DAC0832=i；}
    } while（1）;
}
```

例 10.3 如图 10.4-11 所示。DS18B20 数字温度计使用，用一片 DS18B20 构成测温系统，测量的温度精度达到 0.1 度，测量的温度的范围在－20 度到＋100 度之间，用 8 位数码管显示出来。

表 10.4-1 DS18B20 详细引脚功能描述

序号	名称	引脚功能描述
1	GND	地信号
2	DQ	数据输入/输出引脚。开漏单总线接口引脚。当被用着在寄生电源下，也可以向器件提供电源。
3	VDD	可选择的 VDD 引脚。当工作于寄生电源时，此引脚必须接地。

图 10.4-11 DS18B20 数字温度计电路



（1）把“单片机系统”区域中的 P0.0－P0.7 用 8 芯排线连接到“动态数码显示”区域中的 ABCDEFGH 端子上。

（2）把“单片机系统”区域中的 P2.0－P2.7 用 8 芯排线连接到“动态数码显示”区域中的 S1S2S3S4S5S6S7S8 端子上。

（3）把 DS18B20 芯片插入“四路单总线”区域中的任一个插座中，注意电源与地信号不要接反。

（4）把“四路单总线”区域中的对应的 DQ 端子连接到“单片机系统”区域中的 P3.7/RD 端子上。

C 语言源程序

```
#include <AT89X52.H>
#include <INTRINS.h>
unsigned char code displaybit[]={0xfe,0xfd,0xfb,0xf7,
                                 0xef,0xdf,0xbf,0x7f};
unsigned char code displaycode[]={0x3f,0x06,0x5b,0x4f,
                                  0x66,0x6d,0x7d,0x07,
                                  0x7f,0x6f,0x77,0x7c,
                                  0x39,0x5e,0x79,0x71,0x00,0x40};
unsigned char code dotcode[32]={0,3,6,9,12,16,19,22,
                                25,28,31,34,38,41,44,48,
                                50,53,56,59,63,66,69,72,
                                75,78,81,84,88,91,94,97};
unsigned char displaycount;
unsigned char displaybuf[8]={16,16,16,16,16,16,16,16};
unsigned char timecount;
unsigned char readdata[8];
sbit DQ=P3^7;
bit sflag;
bit resetpulse(void)
{
  unsigned char i;
  DQ=0;
  for(i=255;i>0;i--);
  DQ=1;
  for(i=60;i>0;i--);
  return(DQ);
  for(i=200;i>0;i--);
}
void writecommandtods18b20(unsigned char command)
{
  unsigned char i;
  unsigned char j;
  for(i=0;i<8;i++)
    {
```

```
        if((command & 0x01)==0)
          {
            DQ=0;
            for(j=35;j>0;j--);
            DQ=1;
          }
          else
            {
              DQ=0;
              for(j=2;j>0;j--);
              DQ=1;
              for(j=33;j>0;j--);
            }
        command=_cror_(command,1);
      }
}
unsigned char readdatafromds18b20(void)
{
   unsigned char i;
   unsigned char j;
   unsigned char temp;
   temp=0;
   for(i=0;i<8;i++)
      {
        temp=_cror_(temp,1);
        DQ=0;
        _nop_();
        _nop_();
        DQ=1;
        for(j=10;j>0;j--);
        if(DQ==1)
          {
            temp=temp | 0x80;
          }
          else
            {
              temp=temp | 0x00;
            }
        for(j=200;j>0;j--);
      }
   return(temp);
}
void main(void)
{
```

```
    TMOD=0x01;
    TH0=(65536-4000)/256;
    TL0=(65536-4000)%256;
    ET0=1;
    EA=1;
    while(resetpulse());
    writecommandtods18b20(0xcc);
    writecommandtods18b20(0x44);
    TR0=1;
    while(1)
        {
            ;
        }
}
void t0(void) interrupt 1 using 0
{
    unsigned char x;
    unsigned int result;
    TH0=(65536-4000)/256;
    TL0=(65536-4000)%256;
    if(displaycount==2)
        {
            P0=displaycode[displaybuf[displaycount]] | 0x80;
        }
        else
            {
                P0=displaycode[displaybuf[displaycount]];
            }
    P2=displaybit[displaycount];
    displaycount++;
    if(displaycount==8)
        {
            displaycount=0;
        }
    timecount++;
    if(timecount==150)
        {
            timecount=0;
            while(resetpulse());
            writecommandtods18b20(0xcc);
            writecommandtods18b20(0xbe);
            readdata[0]=readdatafromds18b20();
            readdata[1]=readdatafromds18b20();
            for(x=0;x<8;x++)
```

```
            {
              displaybuf[x]=16;
            }
          sflag=0;
          if((readdata[1] & 0xf8)!=0x00)
            {
              sflag=1;
              readdata[1]=~readdata[1];
              readdata[0]=~readdata[0];
              result=readdata[0]+1;
              readdata[0]=result;
              if(result>255)
                {
                  readdata[1]++;
                }
            }
          readdata[1]=readdata[1]<<4;
          readdata[1]=readdata[1] & 0x70;
          x=readdata[0];
          x=x>>4;
          x=x & 0x0f;
          readdata[1]=readdata[1] | x;
          x=2;
          result=readdata[1];
          while(result/10)
            {
              displaybuf[x]=result%10;
              result=result/10;
              x++;
            }
          displaybuf[x]=result;
          if(sflag==1)
            {
              displaybuf[x+1]=17;
            }
          x=readdata[0] & 0x0f;
          x=x<<1;
          displaybuf[0]=(dotcode[x])%10;
          displaybuf[1]=(dotcode[x])/10;
          while(resetpulse());
          writecommandtods18b20(0xcc);
          writecommandtods18b20(0x44);
        }
}
```

本章小结

本章是对前面所学章节的一个综合运用。在前面的章节中，我们掌握了单片机的结构、特点，了解了它的硬件资源，掌握了单片机程序设计的语言和方法，本章综合这些知识，介绍了单片机应用系统的结构和组成，并从总体设计、硬件设计、软件设计、可靠性设计以及调试和测试几个方面详细介绍了单片机应用系统设计的方法和过程，并给出了典型实例设计，使读者在对单片机应用系统设计的各个阶段应完成的任务有一个清晰的认识，并对整个设计过程有一个全面的了解和把握。

本章的重点是掌握单片机应用系统设计的方法和过程，对每个阶段应完成的任务有明确的认识，难点是如何根据实际需要完成一个单片机应用系统的设计，并将其应用于工程实际中。

思考题与习题

10.1 一般单片机应用系统由哪几部分组成？

10.2 简述单片机应用系统设计中软件、硬件的设计原则

10.3 单片机应用系统设计包括哪些方面的设计？

10.4 选择单片机的原则是什么？

10.5 在单片机应用系统调试的基本步骤是什么？

第 11 章　实验及课程设计

单片机是一门实践性很强的课程，在课程的最后专门安排一个章节进行实际电路及其工作程序的设计、制作和调试工作，可以加强对单片机知识的理解，培养实践技能，提高分析和解决实际问题的能力。

本章针对课程内容安排了 5 个实验和 1 个课程设计,每个实验安排 2 个学时(90 分钟)，课程设计安排一周（30 个学时）。对于实验部分，主要要求学生在仿真器上完成实验电路的连接，并且完成一份实验报告。实验报告内容有如下要求：

（1）画出实验原理图。

（2）画出实验程序流程图。

（3）写出实验中调试通过汇的编语言或者 C 语言程序代码。程序部分要有详细的说明：如主程序、子程序的功能；关键指令、寄存器、存储器的功能等。

对于课程设计报告内容有如下要求：

（1）使用 PROTEL 画出实验原理图。

（2）画出课程设计程序流程图。

（3）写出实验中调试通过汇的编语言或者 C 语言程序代码。程序部分要有详细的说明：如主程序、子程序的功能；关键指令、寄存器、存储器的功能等。

（4）写出一份课程设计电路的使用说明书。包括该电路的功能，详细的操作使用说明。

（5）写出一份课程设计总结。主要叙述自己对该课程设计的时间分配、实际完成情况；在设计过程中遇到的问题及解决方法；对于自己的设计的优点、缺点及改进的设想等等。

11.1　P1 口实验

1 实验目的

掌握 MCS-51 单片机编译软件及仿真器的使用方法。掌握 P1 口的编程方法。

2 实验内容

按图 11.1 接线，编程实现 8 只发光二极管在 P1 口循环点亮。要求循环左移 8 次，再循环右移 8 次，如此不断循环。

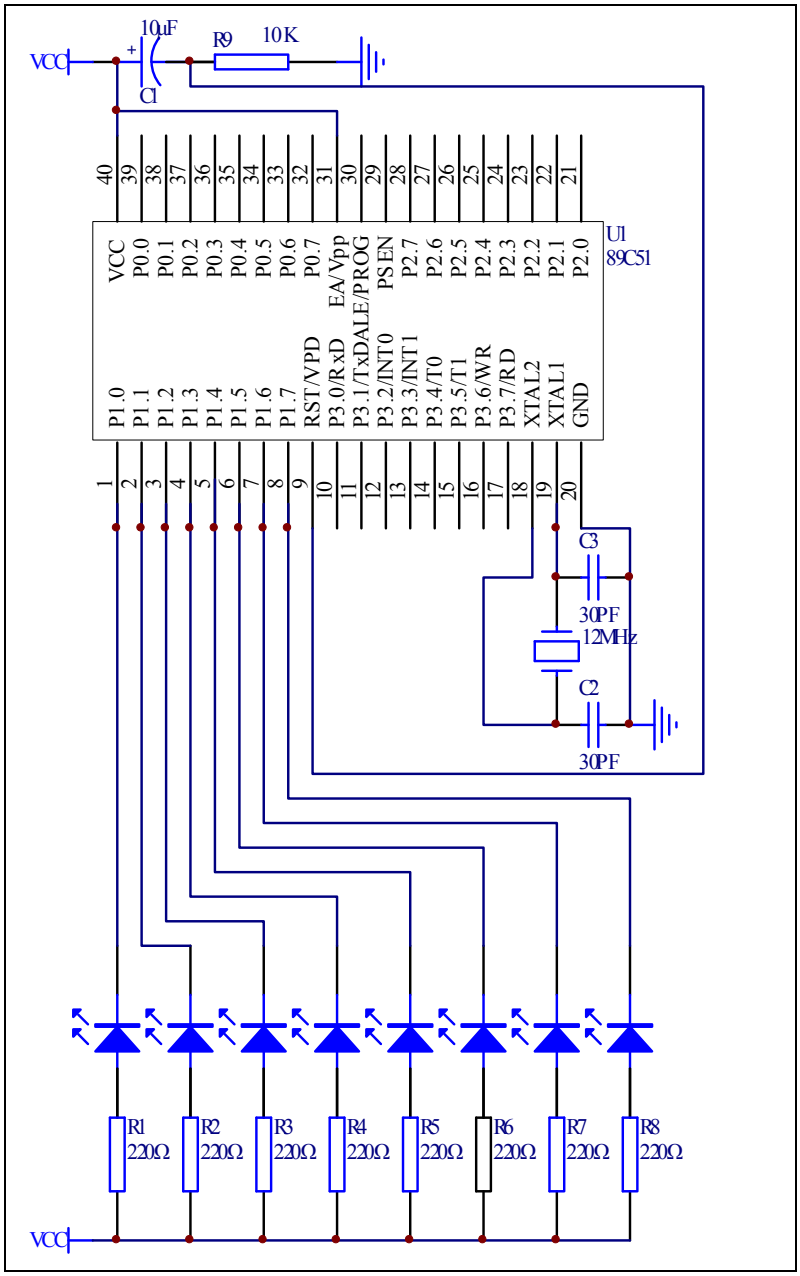
3 参考程序

汇编程序：

```
ORG 0000H                    ；程序存放起始地址为 ROM 的 0000H
START: MOV R2,#8             ；左移循环次数 8 送 R2 寄存
MOV A,#0FEH                  ；从 P1.0 开始点亮 LED
LOOP: MOV P1, A              ；送显示码到 P1 口
LCALL DELAY                  ；调用延时子程序
RL A                         ；循环左移
DJNZ R2,LOOP                 ；左移循环是否结束判断
MOV R2,#8                    ；右移循环次数 8
LOOP1: MOV P1,A              ；显示码 FE 送 P1 口
LCALL DELAY                  ；调用延时子程序
```

```
RR A                                ；右移循环
DJNZ R2,LOOP1                       ；右移循环是否结束判断
AJMP START                              ；左移、右移循环各一次结束，重新循环
DELAY: MOV R5，#20                  ；延时 200 毫秒子程序
D1: MOV R6,#20
D2: MOV R7,#248
DJNZ R7,$
DJNZ R6,D2
DJNZ R5,D1
RET
END                                 ；程序结束
```

图 11.1 P1 口实验



C 语言程序：

```
#include <AT89X51.H>
unsigned char i;
unsigned char temp;
unsigned char a,b;

void delay(void)                          //延时 200 毫秒子程序
{
unsigned char m,n,s;
for(m=20;m>0;m--)
for(n=20;n>0;n--)
for(s=248;s>0;s--);
}

void main(void)                               //主程序
{
while(1)
{
temp=0xfe;                                    //从低位开始显示
P1=temp;
delay();
for(i=1;i<8;i++)                          //先左移循环 8 次
{
a=temp<<i;
b=temp>>(8-i);
P1=a|b;
delay();
}
for(i=1;i<8;i++)                          //右移循环 8 次
{
a=temp>>i;
b=temp<<(8-i);
P1=a|b;
delay();
}
}
}
```

4 思考题

参考 C 语言程序中的移位操作是怎样完成的，它和汇编程序中的移位操作比较哪个更简洁？

11.2　独立式按键实验

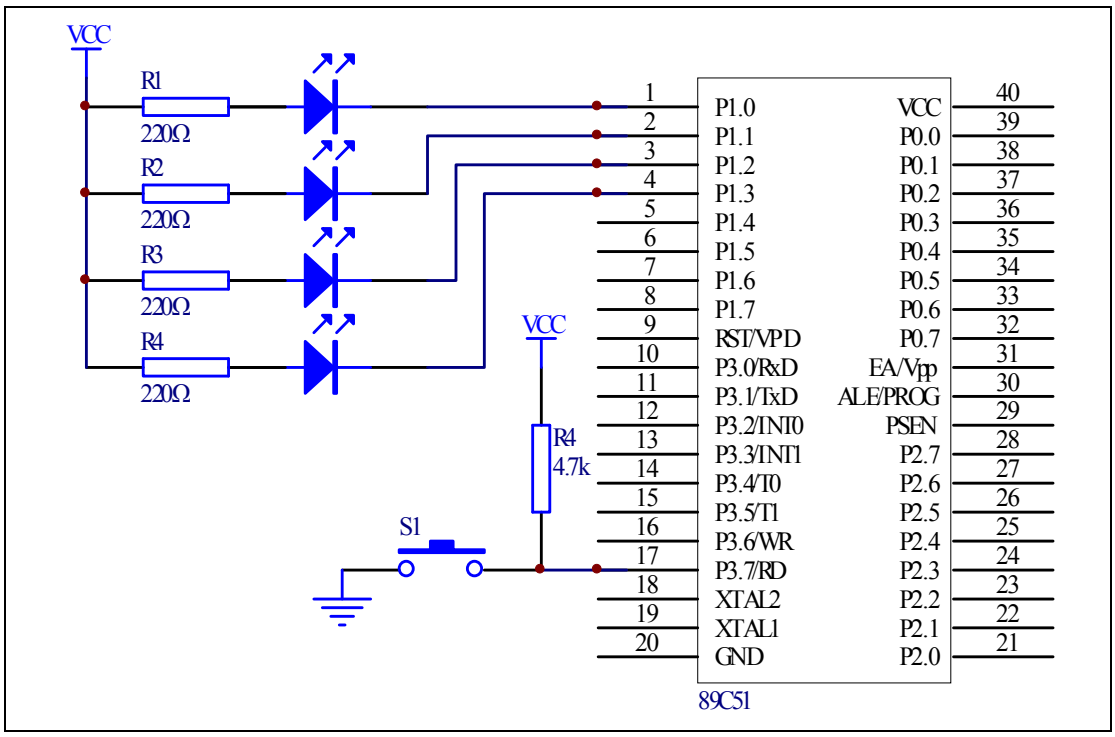
1 实验目的

了解单片机独立式按键的工作原理及编程方法。

2 实验内容

如图 11.2 在单片机的 P1.0 至 P1.3 口线接 4 个 LED，P3.7 口线接一个按键。编程实现每按一次按键，数值加一，4 个 LED 显示对应的 2 进制数。

图 11.2 独立式按键实验



3 参考程序

汇编程序：

```
ORG 0000H                    ;  程序存放起始地址为 ROM 的 0000H
START: MOV R1,#00H           ;  初始化 R 1 为 0 ，表示从 0 开始计数
MOV A,R1                     ;  R1 内容送到 A
CPL A                        ;  取反指令
MOV P1,A                     ;  送出 P1 端口由发光二极管显示
REL: JNB P3.7,REL            ;  判断 SP1 是否按下
LCALL DELAY10MS              ;  若按下，则延时 10 毫秒左右
JNB P3.7,REL                 ;  再判断 SP1 是否真得按下
INC R1                       ;  若真得按下，则进行按键处理，使
MOV A,R1                     ;  计数内容加 1 ，并送出 P1 端口由
CPL A                        ;  发光二极管显示
MOV P1,A                     ;  A 的内容送到 P1
JNB P3.7,$                   ;  等待 SP1 释放
SJMP REL                     ;  继续对 K1 按键扫描
```

```
DELAY10MS: MOV R6,#20                    ;  延时 10ms 子程序
L1: MOV R7,#248
DJNZ R7,$
DJNZ R6,L1
RET
END
```

C 语言程序：

```
#include <AT89X51.H>
unsigned char count;
void delay10ms(void)                     ; 延时 10 毫秒子程序
{
unsigned char i,j;
for(i=20;i>0;i--)
for(j=248;j>0;j--);
}
void main(void)                              ; 主程序
{
while(1)
{
if(P3_7==0)                                  ; 判断按键是否按下
{
delay10ms();
if(P3_7==0)                                  ; 如果按键按下
{
count++;                                 ; 显示数值加一
if(count==16)                            ; 显示数值等于 16 则清零
{
count=0;
}
P1=~count;                                   ; 显示数值取反后送 P1 口
while(P3_7==0);                              ; 等待按键释放
}
}
}
}
```

4 思考题

参考汇编程序与 C 语言程序在处理按键释放时各使用了什么语句?

11.3　7 段LED数码管实验

1 实验目的

掌握 7 段 LED 数码管动态显示的程序编写技术。

2 实验内容

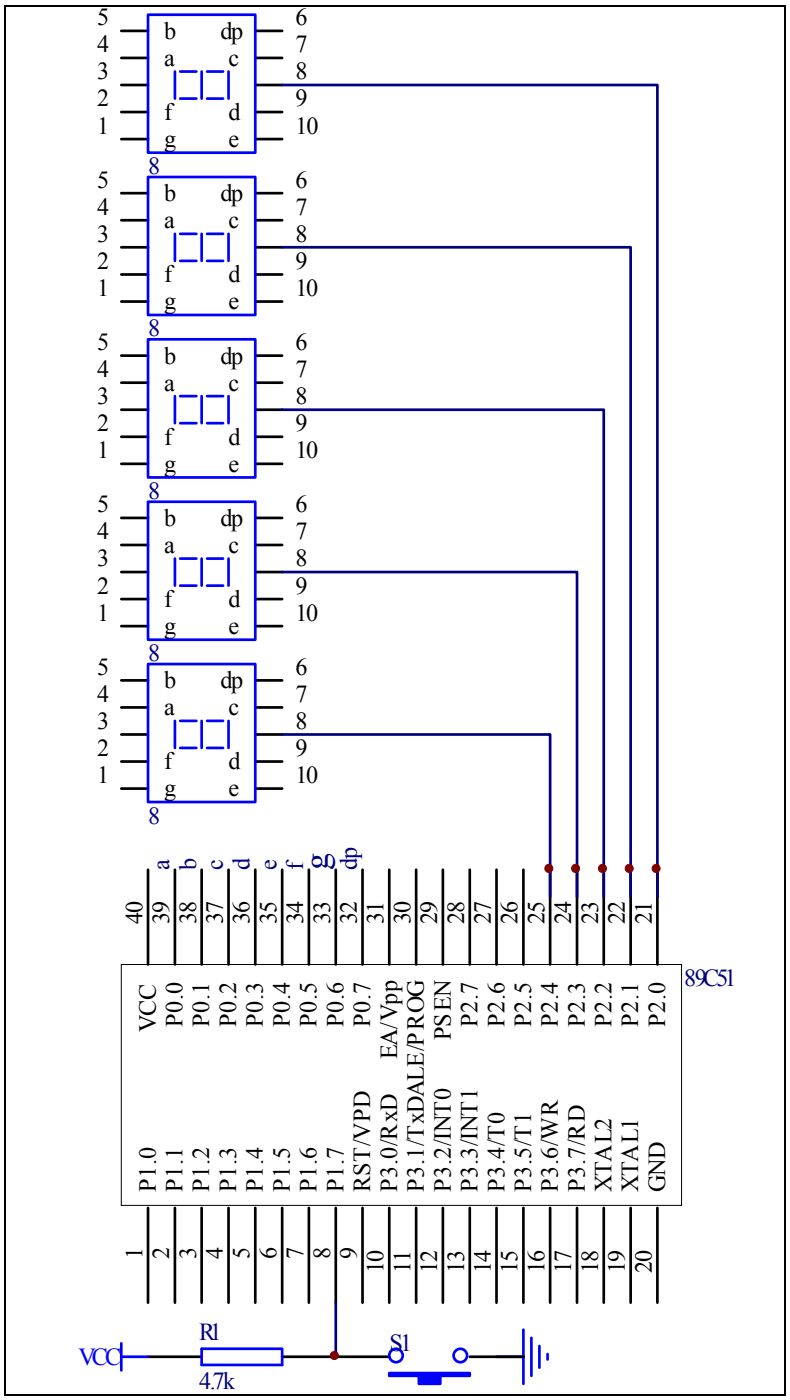
P0 口接 LED 显示器的字型码输入端，P2 口接位选端，P1.7 接一个按键。编程实现当按键按下时，显示“12345”，松开时显示“HELLO”。

3 参考程序

汇编程序：

```
ORG 0000H
START: JB P1.7,DIR1                ；若按键未按下则跳转到 DIR1
MOV DPTR,#TABLE1                   ；“12345”表格首地址送数据指针
SJMP DIR                           ；跳转到 DIR 处
```

图 11.3 7 段 LED 显示器动态显示实验



```
DIR1: MOV DPTR,#TABLE2          ;“HELLO”表格首地址送数据指针
DIR: MOV R0,#00H                ; 清零 R0
MOV R1,#0FEH                    ; 位选码，从右向左显示
NEXT: MOV A,R0                  ; 表格中第一个数据的偏移地址 0 送到 A
MOVC A,@A+DPTR                  ; 查表
MOV P0,A                        ; 查表得到的数据送到 A
MOV A,R1                        ; 位选码送到 A
MOV P2,A                        ; 位选码送到 P0
LCALL DAY                       ; 调用延时程序
INC R0                              ; 表格中下一个数据的偏移地址
RL A                            ; 位选码循环左移一次
MOV R1,A                        ; 位选码送到 R1 寄存
CJNE R1,#0DFH,NEXT              ; 若 5 位显示一遍后跳转到 NEXT
SJMP START                          ; 若 5 位未显示一遍则跳转到 START
DAY: MOV R6,#4                  ; 延时 2ms 子程序
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
TABLE1: DB 06H,5BH,4FH,66H,6DH  ;“12345”
TABLE2: DB 78H,79H,38H,38H,3FH  ;“HELLO”
END                                 ; 程序结束
```

C 语言程序：

```
#include <AT89X51.H>
unsigned char code table1[]={0x06,0x5b,0x4f,0x66,0x6d};//数组 12345
unsigned char code table2[]={0x78,0x79,0x38,0x38,0x3f};//数组 HELLO
unsigned char i;
unsigned char a,b;
unsigned char temp;
void main(void)                                    //主程序
{
while(1)
{
temp=0xfe;                                         //位选码
for(i=0;i<5;i++)                                  //循环显示
{
if(P1_7==0)                                        //按键未按下
{
P0=table1[i];                                      //显示 12345
}
else
{
P0=table2[i];                                      //按键按下显示 HELLO
```

```
}
P2=temp;                                        //由此开始 4 条指令用来位选码移位
a=temp<<(i+1);
b=temp>>(7-i);
temp=a|b;
for(a=4;a>0;a--);                               //延时
for(b=248;b>0;b--);
}
}
```

4 思考题

LED 数码管的亮度和程序中的延时时间有什么关系？

11.4 8051 内部定时器实验

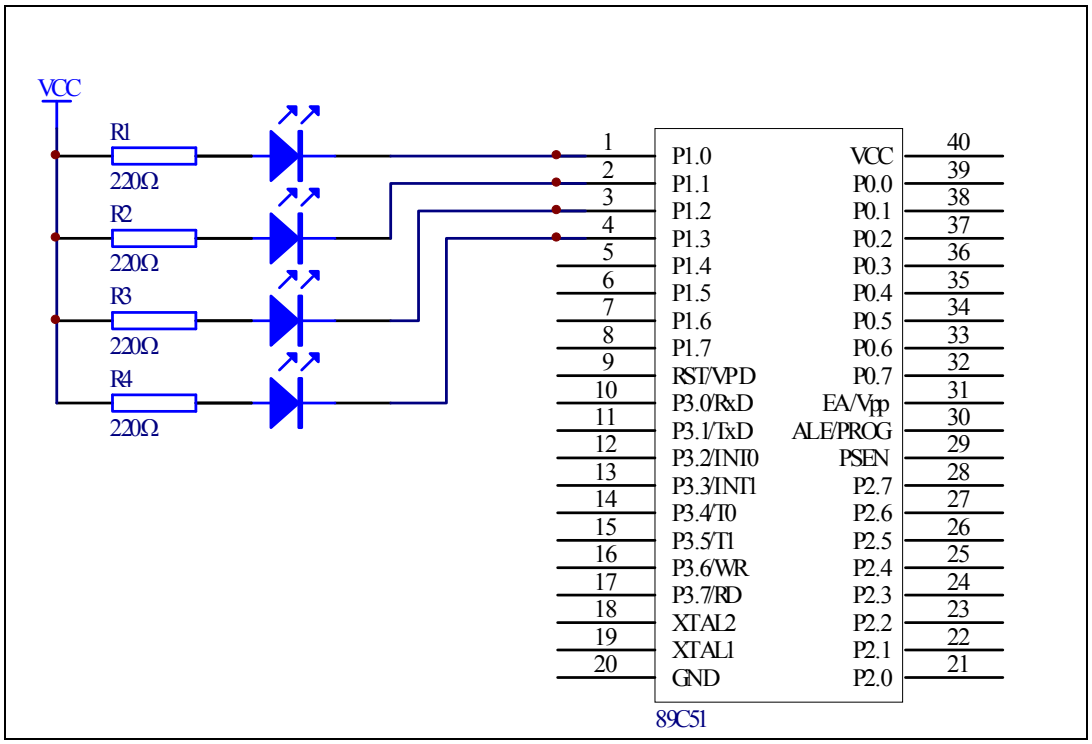
1 实验目的

掌握 MCS51 单片机内部定时器的工作原理、掌握 MCS51 单片机的中断系统工作原理、掌握 MCS51 单片机的中断程序的结构。

2 实验内容

P1.0 至 P1.3 接 4 个 LED。用 MCS-51 的定时 / 计数器 T0 产生 2 秒钟的定时，每当 2 秒定时到来时，更换指示灯闪烁，每个指示闪烁的频率为 5 赫兹。也就是说，开始 L1 指示灯以 5 赫兹的频率闪烁，当 2 秒定时到来之后， L2 开始以 5 赫兹的频率闪烁，如此不断循环。 如图 11.4 所示。

图 11.4 8051 内部定时器实验



3 实验原理

定时 2 秒，采用 16 位定时器方式定时 50 毫秒 ，循环 40 次即可达到 2 秒。定时采用中断方式，每 50 毫秒产生一中断，2 秒定时到来的判断在中断子程序中完成。5 赫兹

的闪烁频率即每秒钟闪烁 5 次，每次用时 0.2 秒，该 0.2 秒的定时，需要 50 毫秒的定时循环 4 次。5 赫兹的闪烁频率也由定时 / 计数器 T0 来完成。对于中断子程序，在主程序中要对中断开中断。由于每次 2 秒定时到时， L1 到 L4 要交替闪烁，采用 ID 号来识别。当 ID ＝ 0 时， L1 在闪烁，当 ID ＝ 1 时， L2 在闪烁；当 ID ＝ 2 时， L3 在闪烁；当 ID ＝ 3 时， L4 在闪烁。

4 参考程序

汇编程序：

```
TCOUNT2S EQU 30H                    ；2 秒钟循环次数存放单元 TCOUNT2S 赋值 30H
TCNT02S EQU 31H                        ；0.2 秒循环次数存放单元 TCNT02S 赋值 31H
ID EQU 32H                             ；当前显示 LED 的 ID 赋值 32H
ORG 0000H                           ；指令存放首地址为 ROM 的 0000H 单元
LJMP START                             ；跳转到 START
ORG 000BH                           ；定时器 T0 中断入口地址
LJMP INT_T0                            ；跳转到 INT_T0 子程序
START: MOV TCOUNT2S,#00H            ；清零 TCOUNT2S
MOV TCNT02S,#00H                    ；清零 TCNT02S
MOV ID,#00H                            ；清零 ID
MOV TMOD,#01H                       ；选择 T0 定时器工作方式 1
MOV TH0,#(65536-50000) / 256      ；50 毫秒定时初值高 8 位送到 TH0
MOV TL0,#(65536-50000) MOD 256      ；50 毫秒定时初值低 8 位送到 TL0
SETB TR0                            ；启动 T0
SETB ET0                            ；开放 T0 中断
SETB EA                                ；开放总中断允许
SJMP $                                 ；原地等待
INT_T0: MOV TH0,#(65536-50000)/ 256 ；T0 中断子程序
MOV TL0,#(65536-50000) MOD 256      ；50 毫秒定时初值重装
INC TCOUNT2S                        ；TCOUNT2S 内容加一
MOV A,TCOUNT2S                      ；TCOUNT2S 内容送到 A
CJNE A,#40,NEXT                        ；若不到 2 秒钟则跳转到 NEXT
MOV TCOUNT2S,#00H                   ；若到 2 秒钟则清零 TCOUNT2S
INC ID                                 ；ID 指向下一个 LED
MOV A,ID                            ；ID 单元内容送到 A
CJNE A,#04H,NEXT                    ；ID 单元内容不为 4 跳转到 NEXT
MOV ID,#00H                            ；ID 单元内容等于 4 则清零 ID 单元
NEXT: INC TCNT02S                   ；TCNT02S 内容加一
MOV A,TCNT02S                       ；TCNT02S 内容送到 A
CJNE A,#4,DONE                      ；若不到 0.2 毫秒则跳转到 DONE
MOV TCNT02S,#00H                    ；若到 0.2 毫秒则清零 TCNT02S
MOV A,ID                            ；ID 单元内容送到 A
CJNE A,#00H,SID1                    ；若 ID 单元内容不为 0 跳转到 SID1
CPL P1.0                            ；ID 单元内容等于 0，取反 P1.0 显示第一个 LED
SJMP DONE                           ；跳转到 DONE
SID1: CJNE A,#01H,SID2              ；若 ID 单元内容不为 1 跳转到 SID2
```

```
CPL P1.1                              ；ID单元内容等于1，取反P1.1显示第二个LED
SJMP DONE                             ；跳转到DONE
SID2: CJNE A,#02H,SID3                ；若ID单元内容不为2跳转到SID3
CPL P1.2                              ；ID单元内容等于不为2，显示第三个LED
SJMP DONE                             ；跳转到DONE
SID3: CJNE A,#03H,DONE                ；若ID单元内容不为3跳转到DONE
CPL P1.3                              ；若ID单元内容等于3，显示第四个LED
DONE: RETI                            ；中断子程序返回
END                                   ；程序结束
```

C语言程序：

```
#include <AT89X51.H>
unsigned char tcount2s;               //2秒钟计数器
unsigned char tcount02s;              //0.2毫秒计数器
unsigned char ID;                     //LED的ID号
void main(void)                           //主程序
{
TMOD=0x01;                                //设置 T0定时器工作方式1
TH0=(65536-50000)/256;                //计算50毫秒定时初值
TL0=(65536-50000)%256;
TR0=1;
ET0=1;
EA=1;
while(1);
}
void t0(void) interrupt 1 using 0//T0中断子程序
{
tcount2s++;
if(tcount2s==40)                      //2秒钟到则清零计数器
{
tcount2s=0;
ID++;                                 //LED的ID号加一
if(ID==4)                             //如果显示一遍则从头显示
{
ID=0;
}
}
tcount02s++;
if(tcount02s==4)                      //0.2毫秒到清零计数器
{
tcount02s=0;
switch(ID)                                //显示对应ID号的LED
{
case 0:
```

```
P1_0=~P1_0;                                    //明暗闪烁
break;
case 1:
P1_1=~P1_1;
break;
case 2:
P1_2=~P1_2;
break;
case 3:
P1_3=~P1_3;
break;
}
}
}
```

5 思考题

参考汇编程序中的当前 LED 显示是如何判断的，采用了什么方法，再设计一种方法，并比较二者的优缺点。

11.5 4×4 矩阵式键盘实验

1 实验目的

掌握矩阵式键盘的工作原理及编程方法。

2 实验内容

如图 11.6 所示，P3 口接 4×4 矩阵式键盘的行线和列线，P0 口接 7 段 LED 显示器的字型码输入端。编程实现每按下一个按键，在 LED 数码管上显示每个按键的序号。对应的按键的序号排列如图 11.5 所示。

3 参考程序

汇编程序：

```
KEYBUF EQU 30H                          ；键值存放单元
ORG 0000H                               ；程序存储起始地址
START: MOV KEYBUF,#2                    ；键值存放单元赋初值
WAIT:
；从第 0 行开始扫描
MOV P3,#0FFH                            ；P3 口送 FFH
CLR P3.4                                ；从第 0 行开始扫描
MOV A,P3                                ；读 P3 口值
ANL A,#0FH                              ；清零高 4 位
XRL A,#0FH                              ；取反低 4 位
JZ NOKEY1                               ；第 0 行无按键按下跳转到 NOKEY1
LCALL DELY10MS                          ；有按键按下延时 10 毫秒去抖动
MOV A,P3                                ；读 P3 口值
ANL A,#0FH                              ；清零高 4 位
```

图 11.5 矩阵式键盘

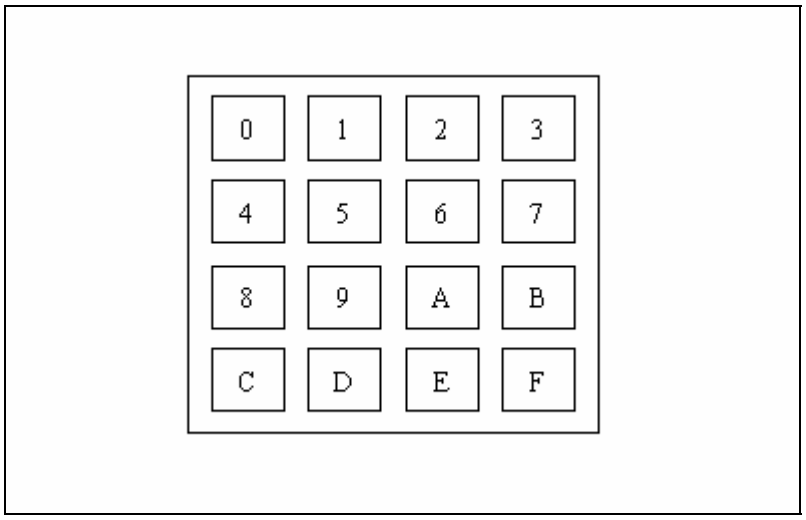
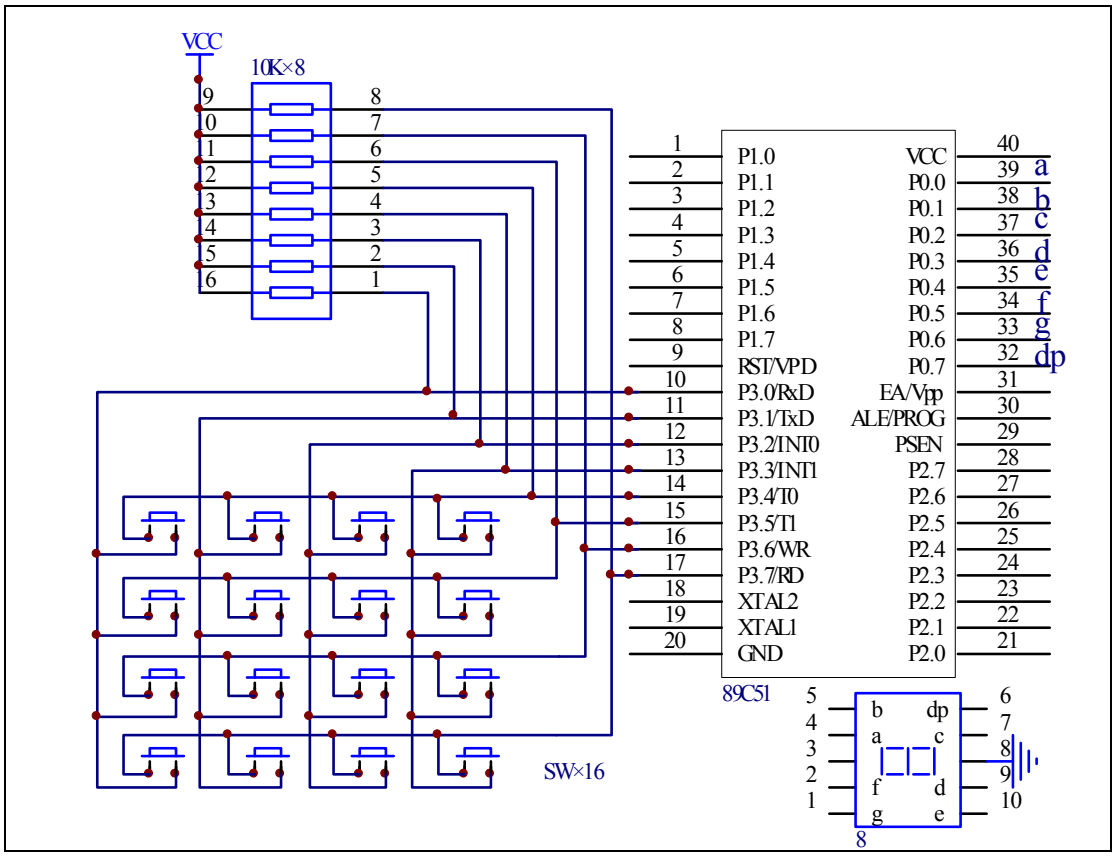


图 11.6 4×4 矩阵式键盘实验




```
XRL  A,#0FH                    ；取反低 4 位
JZ  NOKEY1                     ；第 0 行无按键按下扫描第一行
MOV  A,P3                      ；有按键按下则由此开始计算键值
ANL  A,#0FH                    ；P3 口值高 4 位清零
CJNE  A,#0EH,NK1               ；不是第 0 列跳转到 NK1
MOV  KEYBUF,#0                 ；是第 0 列将键值 0 送到 KEYBUF
```

```
LJMP DK1                    ; 跳转到 DK1 处
NK1: CJNE A,#0DH,NK2        ; 不是第 1 行跳转到 NK2
MOV KEYBUF,#1               ; 是第 1 行将 1 送到 KEYBUF
LJMP DK1                    ; 跳转到 DK1
NK2: CJNE A,#0BH,NK3        ; 不是第 2 行跳转到 NK3
MOV KEYBUF,#2               ; 是第 2 行将 2 送到 KEYBUF
LJMP DK1                    ; 跳转到 DK1
NK3: CJNE A,#07H,NK4        ; 不是第 3 行跳转到 NK4
MOV KEYBUF,#3               ; 是第 3 行将 3 送到 KEYBUF
LJMP DK1                    ; 跳转到 DK1
NK4: NOP                    ; 空指令
DK1:                        ; 标号 DK1
MOV A,KEYBUF                ; 将 KEYBUF 的值送到 A
MOV DPTR,#TABLE             ; 表格首地址送到数据指针
MOVC A,@A+DPTR              ; 查表得按键对应的字型码
MOV P0,A                    ; 字型码送 P0 口显示
DK1A: MOV A,P3              ; P3 口数据送 A
ANL A,#0FH                  ; A 高 4 位清零取反
XRL A,#0FH                  ; A 低 4 位取反
JNZ DK1A                    ; 等待按键释放

NOKEY1:                     ; 由此开始扫描第 1 行
MOV P3,#0FFH                ; P3 口送 FFH
CLR P3.5                    ; 扫描第 1 行
MOV A,P3                    ; 读 P3 口数据
ANL A,#0FH                  ; 清零高 4 位
XRL A,#0FH                  ; 取反低 4 位
JZ NOKEY2                   ; 本行无按键按下则扫描下一行
LCALL DELY10MS              ; 延时 10 毫秒
MOV A,P3                    ; 读 P3 口数据
ANL A,#0FH                  ; 清零高 4 位
XRL A,#0FH                  ; 取反低 4 位
JZ NOKEY2                   ; 本行无按键按下则扫描下一行
MOV A,P3                    ; 本行有按键按下由此开始计算键值
ANL A,#0FH                  ; P3 口数据高 4 位清零
CJNE A,#0EH,NK5             ; 不是第 0 列跳转到 NK5
MOV KEYBUF,#4               ; 是第 0 列将键值送 KEYBUF
LJMP DK2                    ; 跳转到 DK2
NK5: CJNE A,#0DH,NK6        ; 不是第 1 列跳转到 NK6
MOV KEYBUF,#5               ; 是第 1 列将键值送 KEYBUF
LJMP DK2                    ; 跳转到 DK2
NK6: CJNE A,#0BH,NK7        ; 不是第 2 列跳转到 NK7
MOV KEYBUF,#6               ; 是第 2 列将键值送 KEYBUF
LJMP DK2                    ; 跳转到 DK2
```

```
NK7: CJNE A,#07H,NK8          ; 不是第 3 列跳转到 NK8
MOV KEYBUF,#7                 ; 是第 3 列将键值送 KEYBUF
LJMP DK2                      ; 跳转到 DK2
NK8: NOP                      ; 空指令
DK2:                          ; 由此开始查表得键值对应的字型码
MOV A,KEYBUF                  ; 键值送 A
MOV DPTR,#TABLE               ; 字型码表格首地址送数据指针
MOVC A,@A+DPTR                ; 查表
MOV P0,A                      ; 字型码送 P0 口
DK2A: MOV A,P3                ; 由此开始等待按键释放
ANL A,#0FH                    ; 清零 A 高四位
XRL A,#0FH                    ; 将 A 低四位取反
JNZ DK2A                      ; 等待按键释放

NOKEY2:                       ; 第二行扫描开始
MOV P3,#0FFH                  ; P3 口送 FFH
CLR P3.6                      ; 从第三行扫描
MOV A,P3                      ; 读 P3 口数据
ANL A,#0FH                    ; 清零高四位
XRL A,#0FH                    ; 取反低四位
JZ NOKEY3                     ; 若第二行没有按键按下跳到 NOKEY3
LCALL DELY10MS                ; 有按键按下延时 10 毫秒
MOV A,P3                      ; 读 P3 口数据
ANL A,#0FH                    ; 清零高四位
XRL A,#0FH                    ; 取反低四位
JZ NOKEY3                     ; 若第二行没有按键按下跳到 NOKEY3
MOV A,P3                      ; 读 P3 口数据
ANL A,#0FH                    ; 清零高四位
CJNE A,#0EH,NK9               ; 不是第 0 列按键跳转到 NK9
MOV KEYBUF,#8                 ; 是第一列按键则得到键值
LJMP DK3                      ; 跳转到 DK3
NK9: CJNE A,#0DH,NK10         ; 不是第 1 列按键跳转到 NK10
MOV KEYBUF,#9                 ; 是第 1 列按键则得到键值
LJMP DK3                      ; 跳转到 DK3
NK10: CJNE A,#0BH,NK11        ; 不是第 2 列按键跳转到 NK11
MOV KEYBUF,#10                ; 是则得到键值
LJMP DK3                      ; 跳转到 DK3
NK11: CJNE A,#07H,NK12        ; 不是第 3 列按键跳转到 NK11
MOV KEYBUF,#11                ; 是则得到键值
LJMP DK3                      ; 跳转到 DK3
NK12: NOP                     ; 空指令
DK3:                          ; 由此开始查表得到按键字型码
MOV A,KEYBUF                  ; 键值送 A
MOV DPTR,#TABLE               ; 表格首地址送数据指针
```

```
MOVC A,@A+DPTR                  ; 查表
MOV P0,A                        ; 字型码送 P0 口
DK3A: MOV A,P3                  ; 由此开始等待按键释放
ANL A,#0FH                      ; 清零 A 高四位
XRL A,#0FH                      ; 取反 A 低四位
JNZ DK3A                        ; 等待按键释放
NOKEY3:                         ; 由此开始第 3 行按键扫描
MOV P3,#0FFH                    ; FF 送 P3 口
CLR P3.7                        ; 扫描第三行
MOV A,P3                        ; P3 数据送 A
ANL A,#0FH                      ; 清零高四位
XRL A,#0FH                      ; 取反低四位
JZ NOKEY4                       ; 若此行无按键按下跳转到 NOKEY4
LCALL DELY10MS                  ; 若有按键按下则延时 10 毫秒去抖动
MOV A,P3                        ; 读 P3 口数据
ANL A,#0FH                      ; 清零高四位
XRL A,#0FH                      ; 取反低四位
JZ NOKEY4                       ; 若此行无按键按下跳转到 NOKEY4
MOV A,P3                        ; 读 P3 口数据
ANL A,#0FH                      ; 清零高四位
CJNE A,#0EH,NK13                ; 不是第 0 列按键跳转到 NK13
MOV KEYBUF,#12                  ; 是第 0 列则得到键值
LJMP DK4                        ; 跳转到 NK4
NK13: CJNE A,#0DH,NK14          ; 不是第 1 列按键跳转到 NK14
MOV KEYBUF,#13                  ; 是第一列按键则得到键值
LJMP DK4                        ; 跳转到 NK4
NK14: CJNE A,#0BH,NK15          ; 不是第二列按键跳转到 NK15
MOV KEYBUF,#14                  ; 是第二列按键则得到键值
LJMP DK4                        ; 跳转到 NK4
NK15: CJNE A,#07H,NK16          ; 不是第三列按键跳转到 NK16
MOV KEYBUF,#15                  ; 是第三列按键则得到键值
LJMP DK4                        ; 跳转到 DK4
NK16: NOP                       ; 空指令
DK4:                            ; 由此开始查表得到键值对应的字型码
MOV A,KEYBUF                    ; 键值送 A
MOV DPTR,#TABLE                 ; 表格首地址送数据指针
MOVC A,@A+DPTR                  ; 查表
MOV P0,A                        ; 字型码送 P0 口
DK4A: MOV A,P3                  ; 由此开始等待按键释放
ANL A,#0FH                      ; 高四位清零
XRL A,#0FH                      ; 低四位取反
JNZ DK4A                        ; 未释放则继续等待
NOKEY4:                         ; 扫描结束
LJMP WAIT                       ; 重新扫描
```

```
DELY10MS:                                    ; 延时 10 毫秒子程序
MOV R6,#10
D1: MOV R7,#248
DJNZ R7,$
DJNZ R6,D1
RET
TABLE: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H; 共阴极 LED 数码管“0-F”字型码列表
DB 7FH,6FH,77H,7CH,39H,5EH,79H,71H
END                                          ; 程序结束
```

4 思考题

参考程序中 4 行按键的扫描各写出了一段代码，试写出一个通用子程序完成按键的行扫描以缩减代码数量。

11.6 课程设计：带闹钟功能的电子时钟

1 设计题目

带闹钟功能的电子时钟

2 设计任务和要求

使用 MCS-51 单片机做为控制器，6 个 7 段 LED 数码管作为显示器，按 24 小时制显示时间，用户可以设置时钟和闹钟时间。要求使用 protel 画出电气原理图，画出程序流程图，设计出电子钟程序并在仿真器实现走时和闹钟功能，写出电子时钟详细功能及使用说明书。

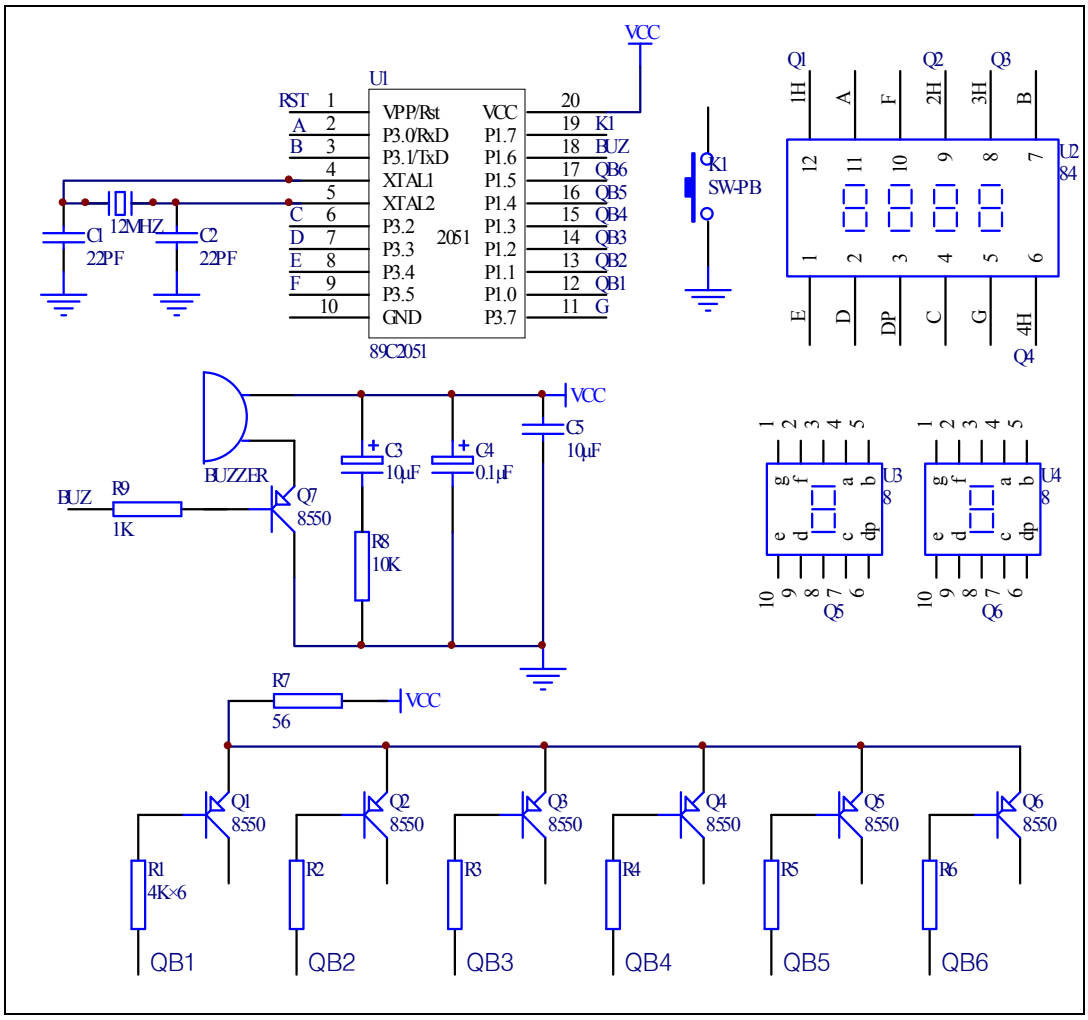
3 参考电路及参考程序

如图 11.7 的参考电子时钟以一片 AT89C2051 单片机为控制单元，采用动态显示方式，显示数据的字型码从 P3 口输出，P1 口输出对应的六位位选信号。LED 数码管位驱动用 8850。发声元件采用 5V 小型蜂鸣器，驱动蜂鸣器采用 PNP 型三极管 8550。该电子时钟只用一个轻触式按键来完成所有的设置。根据实际情况可以调整 R8 的阻值来改变数码管亮度。参考电路如图所示，图中省略了复位电路。

参考程序如下：

```
ORG       0000H              ;程序开始入口
LJMP      START
ORG       0003H              ;外中断 0 中断程序入口
RETI
ORG       000BH              ;定时器 T0 中断程序入口
LJMP      INTT0              ;跳至 INTT0 执行
ORG       0013H              ;外中断 1 中断程序入口
RETI
ORG       001BH              ;定时器 T1 中断程序入口
LJMP      INTT1
ORG       0023H              ;串行中断程序入口地址
RETI
;---------------主程序----------------------;
START:MOV      R0,#70H       ;70 给 R0,清 70-7FH 显示内存
```

图 11.7　带闹钟功能的电子钟




```
MOV     R7,#0FH
CLEARDISP:MOV     @R0,#00H    ;0 给 R0 中的数为地址的内存中存放(70H)
INC     R0
DJNZ    R7,CLEARDISP
MOV     78H,#2              ;默认时间为 12:00,闹钟 5:00
MOV     79H,#1
MOV     74H,#2              ;防止上电时数码管显示 00 小时
MOV     75H,#1
MOV     66H,#1              ;中断退出时 66H 为 1 时,分、时计时单元数据移入显存标志
MOV     68H,#1              ;上电默认闹钟开状态
MOV     7DH,#6              ;闹钟时十位
MOV     7CH,#3
MOV     69H,#0
MOV     7AH,#0AH            ;放入"熄灭符"数据
MOV     TMOD,#11H           ;设 T0、T1 为 16 位定时器
MOV     TL0,#0B0H           ;50MS 定时初值（T0 计时用）
MOV     TH0,#3CH
SETB    EA                  ;总中断开放
```

```
SETB    ET0                 ;允许 T0 中断
SETB    TR0                 ;T0 定时器开始计时
MOV     R4,#14H             ;1 秒定时用初值（50MS×20）
START1:LCALL   DISPLAY      ;显示子程序
LCALL   BEEP                ;闹钟查询
JNB     P1.7,SETTIME        ;P1.7 口为 0 时转时间调整程序
JMP     START1              ;P1.7 口为 1 时跳回 START1
;----------------------时间闹钟调整系统-------------------------;
NFLAG:MOV          A,68H    ;设置闹钟开关状态
CJNE          A,#1,BEE68
MOV           68H,#0
JMP           E6
BEE68:MOV          68H,#1
E6:MOV          66H,#8      ;中断退出 66H 不为 1 时分、时计时单元数据移入显存标志
MOV           72H,7BH
MOV           73H,7CH
MOV           74H,7DH
MOV           75H,7EH
LCALL         DDL
LCALL         DDL
MOV           72H,68H
MOV           73H,69H
MOV           74H,7AH
MOV           75H,7AH
LCALL         DDL
LCALL         DDL
MOV           66H,#1
LJMP      START1
SETTIME:LCALL        DDL
JB            P1.7,NFLAG
;按下时间小于 1 秒，置闹钟开关状态并查看闹钟时间,不关走时,确保准确,大于 1 秒调时
MOV           TL1,#0B0H     ;T1 闪烁定时初值
MOV           TH1,#3CH
MOV           R2,#06H       ;进入调时状态，赋闪烁定时初值
MOV           66H,#8        ;调闹钟时保持走时,关闭时钟显示数据
SETB          TR1           ;开启定时器 T1
SETB          ET1           ;允许 T1 中断
SET1:LCALL         DISPLAY  ;调用显示,防止键按下无显示
JNB        P1.7,SET1        ;P1.7 口为 0 等待键释放
MOV        R5,#00H          ;清设置类型闪烁标志
SETN1:INC      R5           ;闹钟分调整
SET5:LCALL         DISPLAY
JB         P1.7,SET5
SEETN1:LCALL        DDL
```

```
;有键按下大于 1 秒分时间连续加(0.5 秒加 1),小于 1 秒转调时状态
JB          P1.7,SET6                ;键释放查询,键释放自动转调时
MOV         R0,#7CH
LCALL       ADD1
MOV         A,R3
CLR         C
CJNE  A,#60H,HHN1
HHN1:JC          SEETN1
LCALL       CLR0
JMP             SEETN1
SET6:INC         R5                  ;闹钟时调整
SEET6:LCALL          DISPLAY
JB          P1.7,SEET6
SEETNH1:LCALL          DDL
JB          P1.7,SETF
MOV             R0,#7EH
LCALL           ADD1
MOV             A,R3
CLR         C
CJNE  A,#24H,HOUU1
HOUU1:JC         SEETNH1
LCALL       CLR0
JMP   SEETNH1
SETF:LCALL          DISPLAY
JB          P1.7,SETF
LCALL           DDL
JNB         P1.7,SETOUT              ;短按调时,长按退出
CLR             ET0
CLR             TR0
MOV         70H,#00H                 ;设定后的时间从 00 秒开始走时
MOV         71H,#00H
INC   R5
SET3:LCALL          DISPLAY
JB          P1.7,SET3
SETMM:LCALL          DDL
JB          P1.7,SET4
MOV         R0,#77H
LCALL       ADD1
MOV         A,R3
CLR         C
CJNE  A,#60H,MMH
MMH:JC           SETMM
LCALL       CLR0
AJMP            SETMM
```

```
SET4:INC            R5
SEET4:LCALL          DISPLAY
JB          P1.7,SEET4
SETHH:LCALL          DDL
JB          P1.7,SETOUT1
MOV             R0,#79H
LCALL          ADD1
MOV             A,R3
CLR         C
CJNE  A,#24H,HOUU
HOUU:JC          SETHH
LCALL       CLR0
AJMP        SETHH
SETOUT1:SETB    ET0
SETB        TR0                     ;计时开始
SETOUT:MOV      R5,#00H                 ;清设置类型闪烁标志
CLR         TR1                     ;关闭 T1
CLR         ET1                     ;关 T1 中断
MOV             66H,#1
SETOUT2:LCALL         DISPLAY
JNB         P1.7,SETOUT2
LJMP  START1
;--------------------------延时 1 秒钟----------------------;
DDL:MOV             18H,#36
DDL0:MOV             17H,#239
DDL1:LCALL          DISPLAY
DJNZ           17H,DDL1
DJNZ           18H,DDL0
RET
;----------------------------T0 中断程序-------------------------;
INTT0: PUSH   ACC                ;打包
PUSH   PSW
CLR    ET0
CLR    TR0
MOV    A,#0B7H
ADD    A,TL0
MOV    TL0,A
MOV    A,#3CH
ADDC   A,TH0
MOV    TH0,A
SETB   TR0
DJNZ   R4,  OUTT0               ;20 次中断未到中断退出
ADDSS:   MOV     R4,#14H                ;20 次中断到（1 秒）重赋初值
MOV     R0,#71H                 ;指向秒计时单元（70-71H）
```

```
ACALL ADD1                    ;调用加 1 程序（加 1 秒操作）
MOV    A,R3                   ;秒数据放入 A（R3 为 2 位十进制数组合）
CLR   C                       ;清进位标志
CJNE   A,#60H,ADDMM
ADDMM:  JC       OUTT01       ;小于 60 秒退出
ACALL CLR0                    ;等于或大于 60 秒清 0
MOV   R0,#77H                 ;指向分计时单元（76H-77H）
ACALL ADD1
MOV    A,R3
CLR    C
CJNE   A,#60H,ADDHH
ADDHH: JC OUTT0
ACALL CLR0
MOV   R0,#79H                 ;指向小时计时单元（78H-79H）
ACALL ADD1
MOV    A,R3
CLR    C
CJNE   A,#24H,HOUR
HOUR: JC     OUTT0
ACALL CLR0
OUTT0: MOV    A,66H           ;查询标志
CJNE   A,#1,OUTT01
MOV    72H,76H                ;中断退出时将分、时计时单元数据移入对应显示单元
MOV    73H,77H
MOV    74H,78H
MOV    75H,79H
OUTT01:POP   PSW              ;出栈
POP    ACC
SETB   ET0                    ;开放 T0
RETI                          ;中断返回
;---------------------------加 1 程序--------------------------;
ADD1: MOV          A,@R0      ;取当前计时单元数据到 A
DEC            R0             ;指向前一地址
SWAP           A              ;A 中数据高四位与低四位交换
ORL            A,@R0          ;前一地址中数据放入 A 中低四位
ADD            A,#01H         ;A 加 1 操作
DA             A              ;十进制调整
MOV            R3,A           ;移入 R3 寄存器
ANL            A,#0FH         ;高四位变 0
MOV            @R0,A          ;放回前一地址单元
MOV            A,R3           ;取回 R3 中暂存数据
INC            R0             ;指向当前地址单元
SWAP           A              ;A 中数据高四位与低四位交换
ANL            A,#0FH         ;高四位变 0
```

```
MOV             @R0,A              ;数据放入当削地址单元中
RET
CLR0: CLR            C
CLR         A                     ;清累加器
MOV         @R0,A                 ;清当前地址单元
DEC         R0                    ;指向前一地址
MOV         @R0,A                 ;前一地址单元清 0
RET                               ;子程序返回
;-------------------------调时闪烁程序----------------------;
INTT1:            PUSH     ACC
PUSH     PSW
MOV      TL1,#0B0H
MOV      TH1,#3CH
DJNZ     R2,INTT1OUT              ;0.3 秒未到退出中断（50MS 中断 6 次）
MOV      R2,#06H                  ;重装 0.3 秒定时用初值
CPL      02H                      ;0.3 秒定时到对闪烁标志取反
JB       02H,FLASH1               ;02H 位为 1 时显示单元"熄灭"
MOV      A,R5
CJNE     A,#01H,NL
JMP      NNN1
NL:CJNE     A,#02H,NL1
LJMP     NNN1
NL1:CJNE     A,#03H,NL2
LJMP     NLL
NL2:CJNE     A,#04H,INTT1OUT
LJMP     NLL
NLL:MOV      72H,76H
MOV      73H,77H
MOV      74H,78H
MOV      75H,79H
AJMP     INTT1OUT
NNN1:MOV      72H,7BH
MOV      73H,7CH
MOV      74H,7DH
MOV      75H,7EH
INTT1OUT:POP      PSW              ;恢复现场
POP      ACC
RETI                              ;中断退出
FLASH1:MOV      A,R5
CJNE     A,#01H,ML
LJMP     MNN1
ML:CJNE     A,#02H,ML1
LJMP     MNN2
ML1:CJNE     A,#03H,MN
```

```
LJMP    MLL
MN:CJNE    A,#04H,INTT1OUT
LJMP    MHL
MLL:MOV     72H,7AH
MOV     73H,7AH              ;显示单元（72-73H），将不显示分数据
MOV     74H,78H
MOV     75H,79H
AJMP    INTT1OUT
MHL:MOV     72H,76H
MOV     73H,77H
MOV     74H,7AH              ;显示单元（74-75H），小时数据将不显示
MOV     75H,7AH
AJMP    INTT1OUT
MNN1:MOV     72H,7AH
MOV     73H,7AH              ;显示单元（72-73H），将不显示闹钟分数据
MOV     74H,7DH
MOV     75H,7EH
AJMP    INTT1OUT
MNN2:MOV     72H,7BH
MOV     73H,7CH
MOV     74H,7AH
MOV     75H,7AH              ;显示单元（74-75H），闹钟小时数据将不显示
AJMP    INTT1OUT
;-----------------------显示-------------------------------;
DISPLAY:MOV    DPTR,#DISDATA
MOV    A,70H
MOVC   A,@A+DPTR
MOV    P3,A
CLR    P1.0
NOP
NOP
NOP
SETB   P1.0
MOV    A,71H
MOVC   A,@A+DPTR
MOV    P3,A
CLR    P1.1
NOP
NOP
NOP
SETB   P1.1
MOV    A,72H
MOVC   A,@A+DPTR
MOV    P3,A
```

```
CLR     P1.2
NOP
NOP
NOP
SETB    P1.2
MOV     A,73H
MOVC    A,@A+DPTR
MOV     P3,A
CLR     P1.3
NOP
NOP
NOP
SETB    P1.3
MOV     A,74H
MOVC    A,@A+DPTR
MOV     P3,A
CLR     P1.4
NOP
NOP
SETB    P1.4
MOV     A,75H
MOVC    A,@A+DPTR
MOV     P3,A
CLR     P1.5
NOP
NOP
SETB    P1.5
RET
BEEP:MOV    A,68H               ;查询标志
CJNE   A,#1,BEERE
MOV    A,78H                    ;查询闹钟时个位
CJNE   A,7DH,BEERE
MOV    A,79H                    ;查询闹钟时十位
CJNE   A,7EH,BEERE
MOV    A,76H                    ;查询闹钟分个位
CJNE   A,7BH,BEERE
MOV    A,77H                    ;查询闹钟分十位
CJNE   A,7CH,BEERE
BB:CLR    P1.6
LCALL DDL
SETB   P1.6
LCALL DDL
JNB    P1.7,BEERR
BEERE:RET
```

```
BEERR:SETB  P1.6
MOV    68H,#0
LCALL DISPLAY
JNB    P1.7,BEERR
JMP    BEERE
DISDATA:DB 0C0H,0F9H,64H,70H,59H,52H,42H,0F8H,40H,50H,0FFH    ;0123456789 空白
END
```