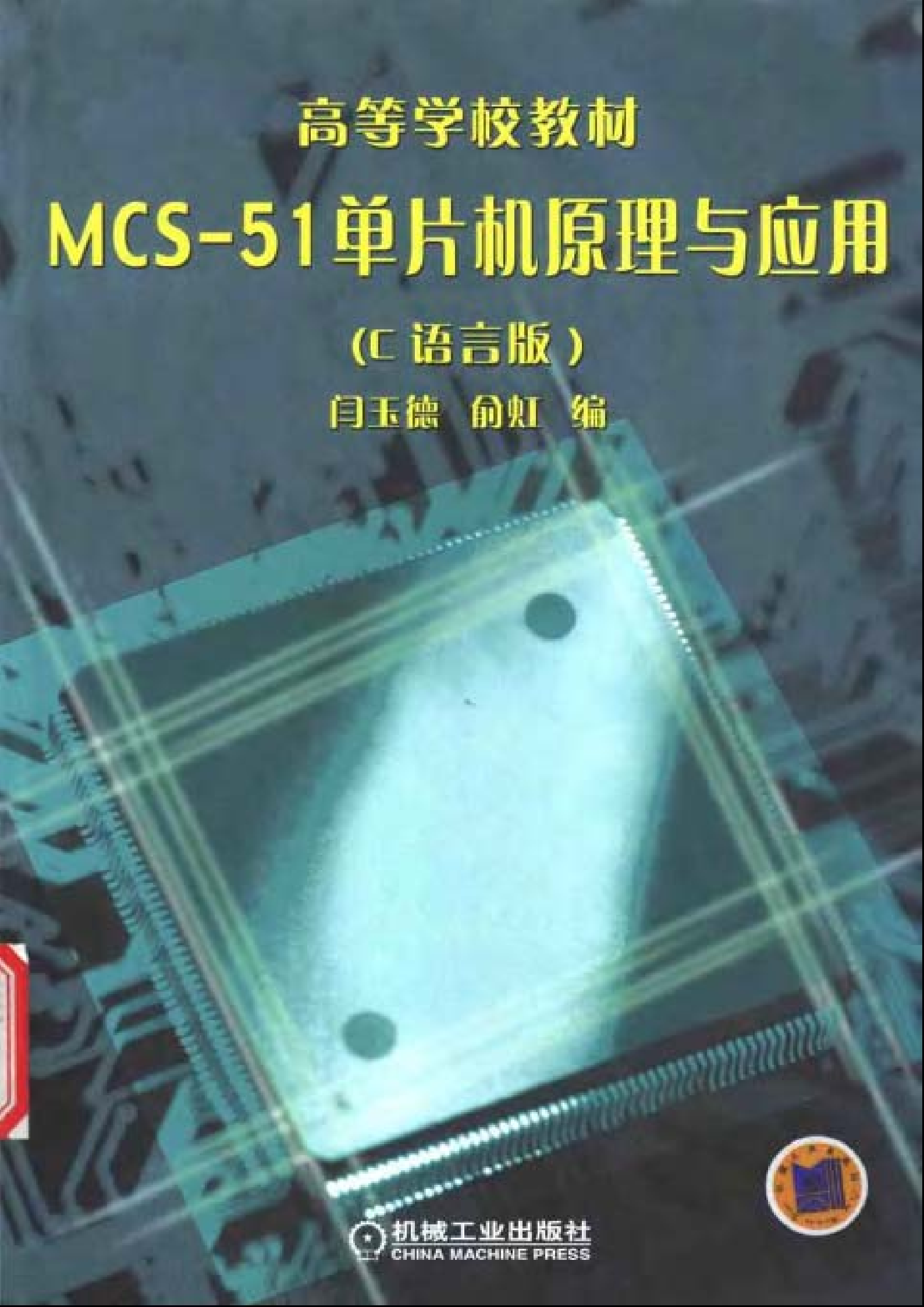
高等学校教材

MCS-51单片机原理与应用

（C语言版）

闫玉德 俞虹 编



机械工业出版社
CHINA MACHINE PRESS

高等学校教材

MCS—51单片机原理与应用

（C语言版）

闫玉德
俞　虹　编

机械工业出版社

本书以目前国内最流行的MCS—51系列单片机为讲述对象，全面叙述其系统结构、工作原理、内部功能器件的特性及组成单片机应用系统时的设计技术和方法。本书共分六章。第一章介绍MCS—51单片机的组织结构和功能，包括CPU的组成和与其有关的寄存器、控制信号；存储器的结构、类型和使用特点；I/O接口的结构、功能和使用特点；MCS—51单片机的引脚、时钟、复位电路、CPU时序及最小系统组成等。第二章介绍MCS—51单片机的指令系统。首先给出MCS—51单片机的寻址方式，然后按指令功能分类简要叙述MCS—51单片机的汇编语言指令。第三章介绍MCS—51单片机的C语言程序设计。首先介绍MCS—51的数据存储器结构和寻址方式及MCS—51的汇编程序指令系统，在此基础上重点介绍MCS—51单片机的C语言程序设计，主要介绍与MCS—51硬件相关的、与PC的C语言程序设计不同的内容，其他内容只是简略说明并给出与硬件相关的例子。第四章介绍MCS—51单片机内含的功能部件的结构、功能、工作方式及其相关的特殊功能寄存器，它们的应用编程都用C语言程序实现。第五章介绍MCS—51单片机的系统扩展方法。首先概述系统扩展的任务、系统扩展的总线结构，包括并行总线和串行（I²C）总线。其次介绍程序存储器和数据存储器的扩展方法。最后，重点介绍并行I/O接口的扩展。第六章首先说明单片机应用系统的组成结构、单片机的任务，然后依次介绍单片机应用系统的前向通道的特点、数据采集子系统的组成结构和A/D变换技术，后向通道的特点和D/A变换技术，人机通道的特点，LED数码显示技术和键盘处理技术。

图书在版编目（CIP）数据

MCS—51单片机原理与应用：C语言版/何卡德，俞虹编．—北京：机械工业出版社，2002

高等学校教材

ISBN 7-111-11141-9

Ⅰ.M… Ⅱ.①何…②俞… Ⅲ.单片微型计算机，MCS—51—高等学校—教材 Ⅳ.TP368.1

中国版本图书馆CIP数据核字（2002）第086293号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）

责任编辑：韩雪清 卢若薇 版式设计：张世琴 责任校对：李秋荣

封面设计：鞠 杨 责任印制：路 琳

北京机工印刷厂印刷·新华书店北京发行所发行

2003年1月第1版·第1次印刷

1000mm×1400mm B5·7.25印张·278千字

0 001—4 000册

定价：18.00元

前　言

在许多领域中，嵌入式系统的应用对产品的技术水平和生产过程自动化水平的提高发挥了重要的作用。单片机是嵌入式系统中普遍使用的核心器件，其应用系统开发人才很受欢迎。单片机原理与应用课程是自动化、机电一体化、仪器仪表等专业的学生的必修课程。

MCS　51 系列单片机是国内最流行的机型，许多学校的单片机与应用课程都以该机型为讲授对象，系统的程序设计都以 MCS　51 指令系统汇编语言为工具。目前，在单片机应用系统开发中，大多数的仿真开发系统支持 C51 程序设计语言，开发者普遍使用 C51 进行系统开发的程序设计。

在各大学中，自动化、机电一体化、仪器仪表等专业的学生大多进修过 C 语言课程，对 C 程序设计有一定的基础，学习 C51 容易理解和掌握。因此，我们对单片机的课程做了改革：简要介绍单片机的指令系统，着重讲述 C51 中对硬件依赖较强的内容，对 C51 中与一般 C 语言相同的内容也只做简要讲述，对 C51 的使用在讲解内部功能器件和系统扩展的例子中加以强化。

本教材是对我们的讲义进行整理后推出的，请兄弟院校的同行们和各位热心的读者提出宝贵意见。

本书可供大专院校作教材使用，也可作为单片机应用系统设计的开发人员的参考资料。

本书第一、四、五、六章由闫玉德编写，第二、三两章由俞虹编写。

目　录

绪　论

随着技术的飞速发展，国民经济各个领域对自动化的要求越来越迫切。计算机在自动化技术中发挥着极其重要的作用。从计算机外围设备、民用电器、医用仪器设备、机电仪一体化产品到航空航天技术，从人工智能、工业机器人到人体工程等领域中，开发计算机应用系统成为一个热门技术。

目前，8位、16位、32位单片机以及具有各种优异性能、特殊功能类型的单片机，如信号处理单片机、USB接口控制单片机、网络通信控制单片机等，可作为广大科技工作者的开发工具。

与通用计算机不同，单片机是专为智能仪器仪表与自动化领域设计开发的专用计算机，在一片芯片上集成了组成一个计算机所需的五个主要部件：运算器、控制器、存储器、输入接口和输出接口，具有体积小、功能强、可靠性好、容易扩展、使用简单、价格便宜等特点。

有两种结构的单片机体系。一种是单总线结构，如Intel公司、Motorola公司和Zilog公司的系列产品。另一种是双总线哈佛结构，如Microchip公司的PIC系列产品和Atmel公司的AVR系列产品。单总线结构的单片机大多是复杂指令集计算机，而双总线哈佛结构的单片机大多是精简指令集计算机。

在国内，主流产品是Intel公司的MCS　51系列单片机。PIC单片机和AVR单片机由于速度快、功耗低、采用精简指令集，受到许多开发者的重视。

从20世纪70年代初期开始，Intel公司开始开发单片机产品。MCS－48和MCS－51系列产品奠定了Intel公司在单片机领域的主导地位。随后，PHILIP公司和ATMEL等公司对MCS—51产品作了进一步的发展，丰富了MCS－51产品，提高了产品的功能和速度，增强了MCS—51产品在智能仪器仪表与自动化领域应用的主导地位。

本书主要介绍Intel公司的单片机产品MCS—51系列单片机产品的结构、功能、系统扩展与应用。表0-1是MCS—51系列单片机的主要机型性能表。

表 0-1　MCS—51 系列单片机主要机型性能表

器件号	OTP EPROM	ROM	RAM	振荡频率 f_{osc}/MHz	8 位并行口	串行 I/O 接口	定时器	A/D		特　点
								位数	通道数	
8031/8X51	1KB	4KB	128	12	1	UART	2			
8032/8X52	8KB	8KB	256	12	1	UART	3			
8XC51GB	8KB		256	12	6	UART+SEP	3+2PCAs	8	8	PWM 15 中断
8XC552	8KB	8KB	256	16	6	UART+I²C	2+PCA	10	8	PWM WDT
8XC752	2KB	2KB	64	16	2+5/8	UART	1	8	3	PWM
8XCE598	32KB	32KB	512	16	6	UART	3	10	8	CANbus WDT
8XC750	1KB	1KB	64	40	2+3/8	—	1	—	—	24 引脚
AT89C51	4KBFLASH		128	24	4	UART	2	—	—	
AT89C52	8KBFLASH		256	24	4	UART	3	—	—	
AT89C2051	2KBFLASH		128	24	1+7/8	UART	2	—	—	

80CXXX 为无 ROM　83CXXX 为 ROM　87CXXX 为 EPROM　FLASH 为 FLASH ROM

OTP 为一次性写入芯片　WDT 为看门狗电路　PWM 为脉宽调制输出　I²C 为串行接口　PCA 为可编程定时/计数器阵列

第一章　MCS—51 单片机的组成及结构分析

本章首先给出单片机的硬件结构框图，然后分别介绍其各组成部分：CPU，存储器，I/O 口，最后给出 MCS -51 单片机引脚信号及 CPU 时序。除串行口和定时器在后面章节中介绍外，学完本章，应对 MCS—51 单片机的硬件结构及各部分的工作原理有一个基本的了解。

1.1　MCS—51 单片机的内部结构框图

8051 单片机的内部总体结构框图如图 1-1 所示。其基本特性如下：

8 位 CPU，片内振荡器。

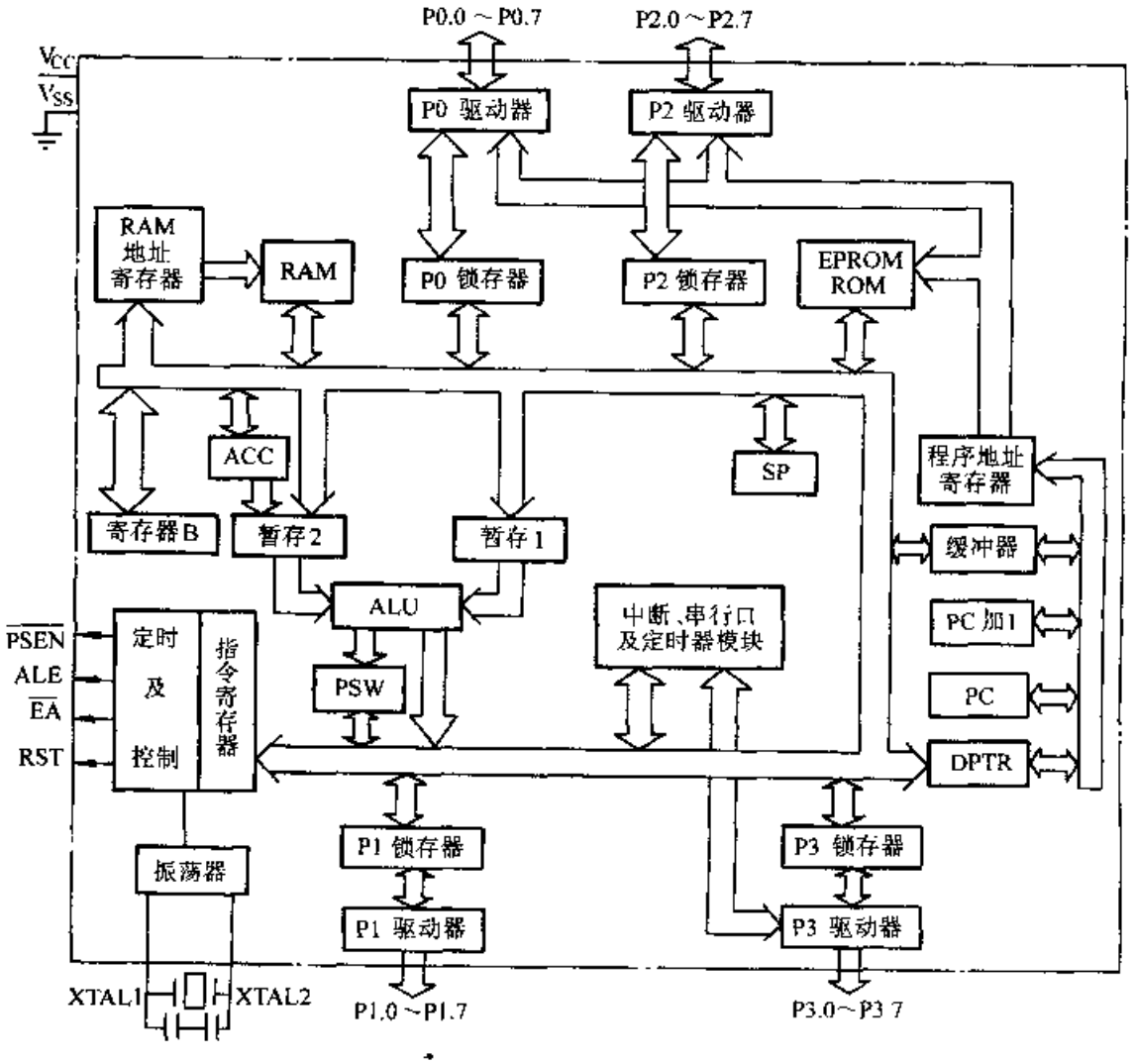


图 1-1　MCS—51 总体结构框图

4K 字节 ROM，128 字节 RAM。

21 个特殊功能寄存器。

32 根 I/O 线。

可寻址各 64K 字节的外部数据、程序存储器空间。

2 个 16 位的定时器/计数器。

中断结构：五个中断源，两个优先级。

一个全双工串行口。

有位寻址功能，适于布尔运算的位处理机。

由框图可知，除 128 字节×8 的片内 RAM，4K 字节×8 的 ROM 和中断、串行口及定时器模块外，就是分布在框图上、下两侧的 4 个 I/O 口 P0～P3。剩余部分则是中央处理器 CPU 的全部组成。而 CPU、存储器、I/O 口（输入输出接口）这三部分则由内部总线紧密地联系在一起。由图可以看出，单片机的基本组成和一般微型计算机是相同的，它只不过是多片微型计算机的单片化而已。

把框图中 4K 字节 ROM 换为 EPROM，就是 8751 的结构框图，如去掉 ROM/EPROM 部分即为 8031 的框图。

1.2 CPU 结构

单片机的中央处理器 CPU 由运算器和控制器组成。与一般多片微机中的 CPU 不同，该 CPU 运算器内包含有一个专门进行位数据操作的布尔处理机，所以下面分三部分介绍。

1.2.1 运算器

图 1 1 中以 8 位的算术/逻辑运算部件 ALU 为核心，加上通过内部总线而挂在其周围的暂存器 TEMP1、TEMP2、累加器 ACC、寄存器 B、程序状态标志寄存器 PSW 以及布尔处理机，就组成了整个运算器的逻辑电路。

算术逻辑单元 ALU 用来完成二进制数的四则运算和逻辑运算。累加器 ACC 是一个 8 位的寄存器，它是 CPU 中工作最频繁的寄存器，在算术逻辑类操作时，累加器 ACC 往往在运算前暂存一个操作数，而运算后又保存其结果。B 寄存器用于乘法和除法操作，对于其他指令，它只能作一个暂存器用。标志寄存器 PSW 也是一个 8 位的寄存器，用来存放运算结果的一些特征。其每位的具体含义见表 1-1。

程序状态字寄存器 PSW

D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV	—	P

表 1-1 PSW 寄存器各位功能、标志符号、位地址

进位标志	CY	PSW.7
辅助进位标志	AC	PSW.6
溢出标志	OV	PSW.2
奇偶标志	P	PSW.0
用户标志	F0	PSW.5
寄存器区选择	MSb RS1	PSW.4
寄存器区选择	LSb RS0	PSW.3
保留	……	PSW.1

对用户来讲，最关心的是以下 4 位。

1）进位标志 CY：它表示了运算是否有进位（或借位）。如果操作结果在最高位有进位（在加法时）或有借位（在减法时），则该位为'1'状态，否则清'0'。

2）辅助进位标志 AC：即所谓半进位标志。它反映了两个 8 位数运算低四位是否有进位，即低 4 位相加（或减）有否进位（或借位），如有，则 AC 为'1'状态，否则 AC 清'0'。辅助进位标志 AC 主要用于 BCD 码加法后的调整。

3）溢出标志位 OV：反映运算结果是否溢出，溢出时 OV 为'1'状态，否则为'0'。溢出和进位标志 CY 是两种不同性质的标志。溢出是指在两个有符号正数相加时，得到负的结果，或两个有符号负数相加时，得到正的结果，OV 等于第六位向第七位的进位与第七位向上的进位（进位位）相异或。而进位位是指两个无符号数作加减运算时有否进位（或借位）。用此两个标志位时应注意场合。

4）奇偶标志 P：反映累加器 ACC 的奇偶性。如果累加器的 8 位的模 2 和是 1（奇），则 P 为'1'状态，否则 P 为'0'（偶）。它完全由 A 累加器中运算结果'1'的个数为偶数还是奇数来决定。

运算器 ALU 主要完成下列功能：

算术运算：加、带进位加、带借位减、乘、除、加 1、减 1 及 BCD 加法的十进制调整。

逻辑运算：与、或、异或、求反、清 0。

移位功能：对累加器 ACC 或带进位位 C 进行逐位的循环左、右移位。

微处理器指令系统的优劣决定了运算器功能的强弱，MCS－51 系列单片机较之 MCS－48 系列单片机功能更为强大，其原因在于它的运算器功能更全面、更丰富。

1.2.2 布尔处理机

布尔处理机是单片机 CPU 中运算器的一个重要组成部分。它有相应的指令系统，可提供 17 条位操作指令，硬件有自己的"累加器"（进位位 CY）、自己的

位寻址 RAM 和位 I/O 空间，所以是一个独立的位处理机。和 8 位操作指令相同，大部分位操作均围绕着位累加器——进位位 C 完成。位操作指令允许直接寻址内部数据 RAM 里的 128 个位和特殊功能寄存器里的位地址空间。对任何可直接寻址的位，布尔处理机可执行置位、取反、等于 1 转移、等于 0 转移、等于 1 转移并清 0 和送入/取自进位位的位操作。在任何可寻址的位（或该位内容取反）与进位位标志之间，可执行逻辑与、逻辑或操作，其结果送回到进位标志 C。

由于布尔处理机给用户提供了丰富的位操作功能，用户在编程时可以利用指令完成原来单凭复杂的硬件逻辑所完成的功能以及可方便地设置标志等。

1.2.3 控制器

控制器是 CPU 的大脑中枢，它包括定时控制逻辑，指令寄存器，译码器，数据地址指针 DPTR，程序计数器 PC，堆栈指针 SP，以及 RAM 地址寄存器，16 位地址缓冲器等。

8051CPU 从程序存储器取出的指令字节放在指令寄存器中寄存，使整个分析执行过程一直在该指令控制下。指令寄存器中的指令代码被分析译码成一种或几种电平信号，这些电平信号与外部时钟脉冲（系统时钟）在 CPU 定时与控制电路中组合，形成各种按一定时间节拍变化的电平和脉冲，即是控制信息，在 CPU 内部协调寄存器之间进行数据传送、数据运算等操作，对外部发出地址锁存 ALE，外部程序存储器选通（$\overline{\text{PSEN}}$）以及外部数据存储器的读（$\overline{\text{RD}}$）、写（$\overline{\text{WR}}$）等控制信号。

控制器的定时功能是由时钟和定时电路完成的，它产生 CPU 的操作时序。8051 的时钟可以由两种方式产生，一种是内部方式，另一种是外部时钟方式，如图 1-2a、b 所示。

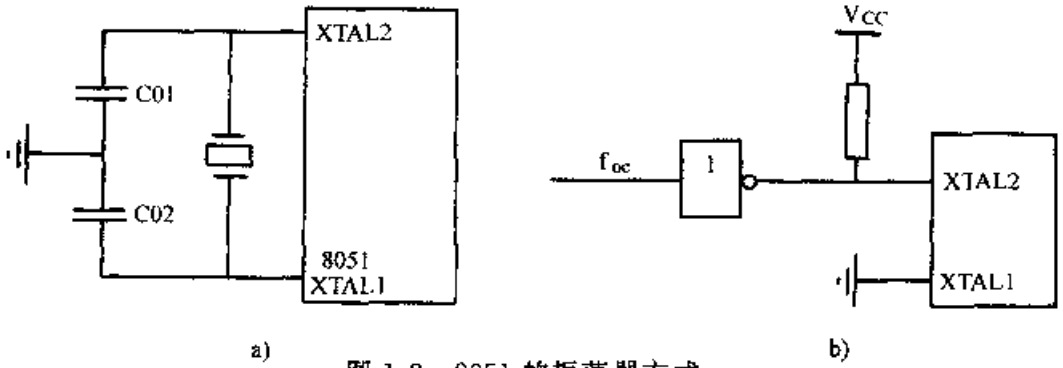


图 1-2 8051 的振荡器方式

a）内部振荡器方式 b）外部振荡器方式

图中 XTAL1 为芯片内部振荡电路（单级反相放大器）输入端，XTAL2 为芯片内部振荡电路输出端。若采用内部方式，则利用芯片内反相器和电阻组成的振荡电路，在 XTAL1、XTAL2 引脚上外接定时元件，如压电晶体和电容组成的并联谐振回路，则在内部可产生与外加晶体同频率的振荡时钟。一般晶体可以在 1.2

～12MHz 之间任选，电容 C01、C02 在 5～30pF 之间选择，对时钟频率有微调作用。若采用外部时钟方式，此时把 XTAL1 接地，振荡频率由 XTAL2 引脚提供，一般当整个单片机系统已有时钟源或者在多机系统中为取得时钟上的同步，可考虑使用此方式。

至于 16 位的程序计数器 PC 和 8 位的堆栈指针 SP 等，其作用和一般微机是相同的。CPU 利用程序计数器 PC 指出下一条指令字节所在程序存储器地址。而用堆栈指针 SP 在片内 RAM128 个字节中开辟栈区，并随时跟踪栈顶地址。MCS—51 的堆栈是向上生成的，即将一字节数据压入堆栈时，堆栈指针 SP 增一，将一字节数据弹出堆栈时，堆栈指针 SP 减一。初始化时，单片机栈底地址（即 SP 的内容）为 07H。由于 MCS--51 系列单片机存储器组织将程序存储器和数据存储器分开，所以又增设了一个 16 位的地址指针 DPTR，用以指示外部数据存储器或 I/O 口的地址。通过后面结合指令系统介绍，对它们的作用将会有更深的体会。

1.3 存储器空间

单片机存储器结构的主要特点是程序存储器和数据存储器的寻址空间是分开的。对 MCS—51 系列（8031 和 8032 除外）而言，有 4 个物理上相互独立的存储器空间：内、外程序存储器，内、外数据存储器，如图 1-3 所示。

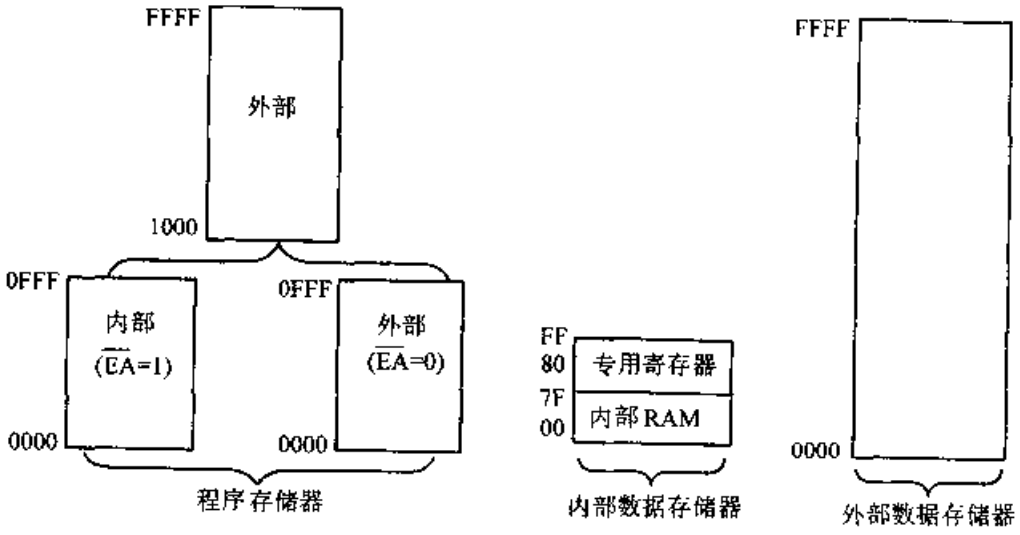


图 1-3 MCS—51 存储器的配置图

程序存储器是用于存放程序指令的。其中 8051 片内有 4K 字节 ROM，8751 片内则为 4K 字节 EPROM，而 8052、8752 的片内存储器容量则为 8K 字节。8031、8032 无片内程序存储器。所以片内程序存储器的有无和种类是区别 MCS—51 系列产品 8031，8051 和 8751 的主要标志。至于片外程序存储器的容量，用户可根据需要任意选择，但片内片外的总容量合起来不得超过 64K 字节。

用于存放数据的片内随机存储器容量为 128 字节（8032、8052 等为 256 字

节)；片外随机存储器的容量一般为 64K 字节。这对于 MCS—51 系列中的 8031、8051 和 8751 单片机都是相同的。

从逻辑空间上看，实际上存在三个独立的空间。片内片外的程序存储器在同一逻辑空间中，它们的地址从 0000H～FFFFH 是连续的。片内、片外的数据存储器各占一个逻辑空间，其中片内数据存储器为 00H～FFH，而片外为 0000H～FFFFH。单片机存储器空间的这种配置方法与一般微处理机是不相同的，对一般微机比如 Z—80 微计算机来说，只有一个统一的逻辑空间为 0～64K 字节，在这统一的空间中任意分配程序存储器和数据存储器的地址及其容量。所以，有必要对单片机三个独立的逻辑空间分别加以讨论。

1.3.1 程序存储器

程序是指挥计算机动作的一系列命令，计算机所能认识并执行的命令是一串由'0'、'1'代码组成的所谓机器指令。在计算机处理问题之前，必须事先把编好的程序和所需表格常数存入机器内存之中，单片机中完成这一存储任务的物理器件就是程序存储器。

各类单片机的程序存储器对比见本书绪论的表 0-1。对 8051、8751 等产品可以由片内和片外两部分组成，若要使用片内程序存储器，则必须占用由 0000H～0FFFH 的最低 4K 字节。这时片外扩充的程序存储器地址编号应由 1000H 开始。单片机程序存储器的这种结构并未限制用户一定要使用片内程序存储器，它提供的$\overline{EA}$信号，使得用户可作出两种选择：若$\overline{EA}$脚保持高电平（接 Vcc），在地址小于 4K 字节时，CPU 访问内部的程序存储器，在地址大于 4K 字节时（对 8052 来说则为 8K 字节），CPU 自动访问外部程序存储器。在$\overline{EA}$脚为高电平情况下，用户既可使用内部程序存储器，也可使用外部程序存储器。由于由片内到片外程序存储器的总线扩展是自动进行的，所以地址空间是连续的；若$\overline{EA}$引脚接地，CPU 的取指操作只对外部程序存储器进行，不用内部程序存储器。显然，对 8031 来说，由于无片内程序存储器，所以$\overline{EA}$信号引脚必须接地。

由于程序计数器 PC 为 16 位，使得程序存储器可使用 16 位地址，这就决定了外部扩充程序存储器的容量。若使用内部程序存储器，则外部可扩充容量最大为 60K 字节，若不使用内部存储器，则外部扩充程序存储器容量可达 64K 字节。实际应用中，程序存储器的容量，用户可根据需要扩展。而其地址空间原则上也可由用户任意安排。但由于单片机复位后程序计数器的内容为 0，使得单片机必然从 0 单元取指令，执行程序。而 0003H 到 0023H 单元又分别固定用于 5 个中断服务子程序的入口地址。所以，往往把该存储区间作为保留单元，而由 0000H 开始存放一条绝对跳转指令，用户设计的程序则由跳转后的地址开始安放。

外部程序存储器一般由 EPROM、EEPROM、FLASH ROM 组成，单片机访问时，至少需要提供两类信号，一类是地址信号，用来确定选中某一单元，另一

类是控制信号，一般接在外部程序存储器的数据允许输出端$\overline{OE}$和片选端$\overline{CE}$。由于单片机无专门的地址总线和数据总线，一般用口 P2 输出地址的高 8 位，而用口 P0 分时输出地址的低 8 位和数据。并由 ALE 信号把低 8 位地址锁存在地址锁存器中。而单片机提供的另一程序存储器输出允许信号$\overline{PSEN}$，则往往与存储器芯片的数据允许输出端相连。关于这个信号的有关说明及使用在后面 CPU 时序引脚信号中还要做详细介绍。由于程序存储器和数据存储器在逻辑上截然分开，所以尽管 CPU 可通过 MOVC 指令遍访 64K 字节程序存储器空间，但没有指令可控制程序由程序存储器空间转到数据存储器空间。也没有任何其他指令可以用来更改程序区的内容，即向程序存储器执行写入操作。因此，单片机的程序存储器是只读的，这也是和多片机系统的不同点。

1.3.2 外部数据存储器

单片机的外部数据存储器由随机存取存储器 RAM 组成。其最大可扩展到 64K 字节，用于存储数据。实际使用时应首先充分利用内部数据存储器空间，只有在实时数据采集和处理或数据存储量较大的情况下，方才扩充数据存储器。最常采用的存储器芯片是半导体静态 RAM 电路。

访问外部数据存储器，可以用 16 位数据存储器地址指针 DPTR，同样用口 P2 输出地址高 8 位，用口 P0 输出地址低 8 位，用 ALE 作为地址锁存信号。但和程序存储器不同，数据存储器的内容既可读出也可写入。在时序上则产生相应的$\overline{RD}$和$\overline{WR}$信号，并以此来选通存储器。也可以用 8 位地址访问外部数据存储器，这不会与内部数据存储器空间发生重叠。单片机指令中设置了专门访问外部数据存储器的指令 MOVX，使得这种操作既区别于访问程序存储器的指令 MOVC，也区别于访问内部数据存储器的 MOV 指令，这在时序上和相应的控制信号上都得到了保证。

显然，当外部数据存储区不超过 256 字节（一个页面）时，8 位地址已足够使用，若要扩展较大的 RAM 区域，则应在使用 8 位地址时预先设置 P2 口的内容，以确定页面地址（高 8 位），而再用 8 位地址指令执行对该页面内某存储单元的操作。

1.3.3 内部数据存储器

从应用的角度讲，搞清内部数据存储器的结构和地址空间分配是十分重要的。因为读者在将来学习指令系统和程序设计中会经常接触到它们。内部数据存储器由地址 00H～FFH 共有 256 个字节的地址空间组成。这 256 个字节的空间被分为两部分，其中内部数据 RAM 地址为 0～127（即 00H～7FH），特殊功能寄存器（SFR）的地址为 128～255（即 80H～FFH）。此外，在这 256 个字节中，还有一个可以按位寻址的"位地址"空间，供布尔处理机执行位操作指令时使用。所以在内部数据存储器中，存在着这样一些单元，CPU 既可以对其执行按字节的操作，

布尔处理机也可以对其中每个单元的 8 位二进制数据执行按位的操作。整个内部数据存储器地址空间分配如图 1-4 所示。(8052/8032 片内 RAM 为 256 个单元)。由图可见，在内部数据 RAM128 个单元中，由 32～48 (20H～2FH) 共计 16 个单元为位地址空间，共计 16×8＝128 个位。而特殊功能寄存器中有 12 个专用寄存器（这些寄存器的直接地址可被 8 除尽）的各位（除 IP.7、IP.6 和 IE.6 以外）是可位寻址的，计有 93 个可寻址位。所以，内部数据存储器共可提供进行位操作的 221 个位地址。

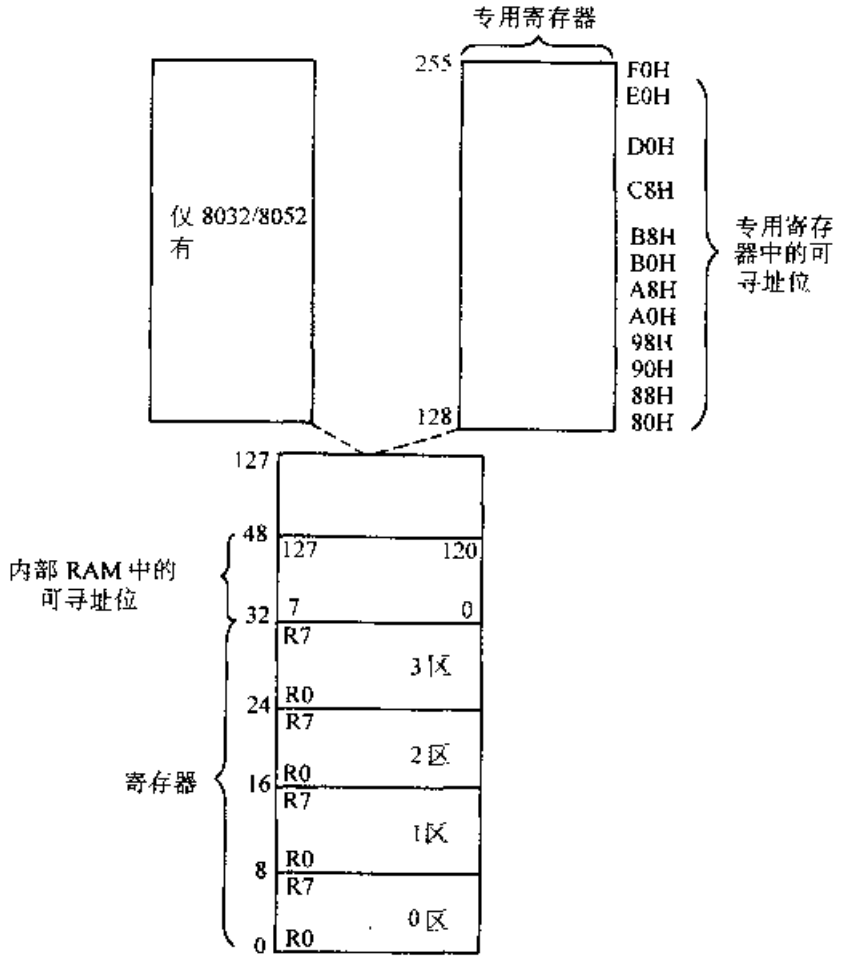


图 1-4　内部数据存储器

1. 内部数据 RAM 单元

8051 单片机内部有 128 个字节的随机存取存储器 RAM。CPU 为其提供了丰富的操作指令，它们均可按字节操作。用户既可以当数据缓冲区，也可以在其中开辟自己的栈区，还可以利用所提供的工作寄存器区进行数据的快速交换和处理。

MCS—51 单片机的特点之一是内部工作寄存器以 RAM 形式组成，它没有专门的寄存器阵列。在 MCS—51 单片机中，那些与 CPU 直接有关或表示 CPU 状态的寄存器如堆栈指针 SP，累加器 ACC，程序状态寄存器 PSW 等则归并于特殊功能寄存器 SFR 之中。在 RAM 存储区的 00H～1FH 区域，划分出四组，每组有工

作寄存器R0～R7，共32个工作寄存器，可用来暂存运算的中间结果以提高运算速度，也可以用其中的R0、R1来存放8位的地址值，去访问一个256字节块中的单元。另外，R0～R7也可以用作计数器，在指令作用下加1或减1。但它们不能组成所谓的寄存器对，因而也不能当作16位地址指针使用。

单片机工作寄存器很多，当需要快速保护现场时，不需交换寄存器内容，只需改变标志寄存器PSW中的RS0、RS1这两位，就可完成一组8个寄存器的切换，这就给用户程序保护寄存器内容提供了极大方便。只要CPU执行一条单周期指令，就可改变程序状态字PSW的第3、第4位，即PSW.3和PSW.4，如表1-2所示。

表1-2　PSW.3、PSW.4的含义

RS1	RS0		R0～R7所占用单元的地址
0	0	0组（BANK0）	00H～07H
0	1	1组（BANK1）	08H～0FH
1	0	2组（BANK2）	10H～17H
1	1	3组（BANK3）	18H～1FH

单片机只有8位的堆栈指针SP，单片机的堆栈限制在内部数据存储区，由于堆栈指针为8位，所以原则上堆栈可由用户分配在片内RAM的任意区域，只要对堆栈指针SP赋以不同的初值就可指定不同的堆栈区域。但在具体应用时，栈区的设置应和RAM的分配统一考虑。工作寄存器和位寻址区域分配好后，再指定堆栈区域。由于MCS—51系列机复位以后，SP为07H，指向工作寄存器区1，因此用户初始化程序都应对SP设置初值，一般设在30H以后的范围为宜。建议RAM空间的分配使用情况如表1-3所示。MCS　51系列机的堆栈是向上生成的，例如，若SP＝40H，则CPU执行一条调用指令或响应中断，程序计数器PC的内容会自动保护进栈，PC（L）保护到41H，PC（H）保护到42H，此时SP的内容会自动修改为（SP）＝42H。

表1-3　内部数据存储器分配表

<table>
<tr><td>7FH

30H</td><td>数据缓冲区
堆栈区
工作单元</td><td>只能字节寻址</td></tr>
<tr><td>2FH
20H</td><td>位地址：00H～7FH</td><td>可位寻址的区域（16个字节128位，也可按字节寻址）</td></tr>
<tr><td>1FH</td><td>3区</td><td rowspan="4">4组工作寄存器R0～R7
（也可以作为RAM单元使用，以字节地址寻址）</td></tr>
<tr><td></td><td>2区</td></tr>
<tr><td></td><td>1区</td></tr>
<tr><td>00H</td><td>0区</td></tr>
</table>

2. 特殊功能寄存器

特殊功能寄存器（SFR）的地址空间是128～255，其地址范围为80H～FFH。在MCS--51中，除程序计数器PC和四组工作寄存器区外，其余21个寄存器（3个只属于8032/8052）都存于SFR块中，其中5个是双字节寄存器，它们共占用了26个字节，如表1-4所列。所有这些特殊功能寄存器的地址分配示于表1-5中。

表1-4 特殊功能寄存器的名称、标识符和地址

标识符	名　　称	地　　址
ACC	累加器	0E0H
B	B寄存器	0F0H
PSW	程序状态字	0D0H
SP	堆栈指针	81H
DPTR	数据指针（包括DPH和DPL）	83H和82H
P0	口0	80H
P1	口1	90H
P2	口2	0A0H
P3	口3	0B0H
IP	中断优先级控制	0B8H
IE	允许中断控制	0A8H
TMOD	定时/计数器方式控制	89H
TCON	定时/计数器控制	88H
T2CON*	定时/计数器2控制	0C8H
TH0	定时/计数器0（高位字节）	8CH
TL0	定时/计数器0（低位字节）	8AH
TH1	定时/计数器1（高位字节）	8DH
TL1	定时/计数器1（低位字节）	8BH
TH2*	定时/计数器2（高位字节）	0CDH
TL2*	定时/计数器2（低位字节）	0CCH
RLDH*	定时/计数器2自动再装载（高位字节）	0CBH
RLDL*	定时/计数器2自动再装载（低位字节）	0CAH
SCON	串行控制	98H
SBUF	串行数据缓冲器	99H
PCON	电源控制	87H

注：带*号的寄存器仅在8052系列机有。

特殊功能寄存器反映了8051的状态，实际上是8051的状态字及控制字寄存器，大体可分为两类，一类与芯片的引脚有关，另一类作芯片内部功能的控制用。8051中的一些中断屏蔽及优先级控制，不是采用硬件优先链方式解决，而是用程序在特殊功能寄存器中设定。定时器、串行口的控制字等全部以特殊功能寄存器出现，这就使得单片机有可能把I/O口与CPU、存储器集成在一起，代替多片机

表 1·5　特殊功能寄存器地址分配表

寄存器名	位地址（十）		字节地址		
	255	248	（十）	（十六）	
B	B		240	(F0H)	←
	247	240			
ACC	A		224	(E0H)	←
	231	224			
PSW			208	(D0H)	←
	215	208			
IP			184	(B8H)	←
	191	184			
P3			176	(B0H)	←
	183	176			
IE			168	(A8H)	←
	175	168			
P2			160	(A0H)	←
	167	160			
SBUF			153	(99H)	
SCON			152	(98H)	←
	159	152			
P1			144	(90H)	←
	151	144			
TH1			141	(8DH)	
TH0			140	(8CH)	
TL1			139	(8BH)	
TL0			138	(8AH)	
TMOD			137	(89H)	
TCON	143	136	136	(88H)	←
PCON			135	(87H)	
DPH			131	(83H)	
DPL			130	(82H)	
SP			129	(81H)	
P0	135	128	128	(80H)	←

可位寻址

中多个芯片连接在一起完成的功能。因为在一般微机中（比如 Z-80），其标准的接口芯片都是可编程的，即其功能可由程序变更，所以许多接口芯片必然都带有控制寄存器或数据寄存器、计数器等，而在单片机中，则由相应特殊功能寄存器取代，这也是 MCS—51 单片机设计的一大特色。

与芯片引脚有关的特殊功能寄存器是 P0～P3，它们实际上是 4 个锁存器，每个锁存器附加上相应的一个输出驱动器和一个输入缓冲器就构成了一个并行口。MCS-51 单片机共有 4 个这样的并行口，可提供 32 根 I/O 线，每根线都是双向的，并且具有第二功能。其余用于芯片内控制的寄存器中，算术寄存器 A、B、PSW，堆栈指示器 SP，数据指示器 DPTR 的功能前已述及，另一些寄存器的功能在后面有关章节再做详细介绍。

3. 位地址空间

8051 具有功能很强的布尔处理机，有着丰富的位操作指令，而且硬件上有自己的累加器和位地址空间。在 MCS—51 系列中，共有 221 个位，位地址范围在 00H～FFH 之间。其中低 128 位处于内部数据 RAM 的 20H～2FH 字节单元，其位地址编码从 00H～7FH，如表 1-6 所示。高 93 位则分布在特殊功能寄存器区中，但不是连续的。从 80H 开始每 8 个单元有一个可以位寻址的专用寄存器。目前已定义了 12 个（其中 0C8H 是 8032/8052 的 T2CON），尚有 0C0H、0D8H、0E8H 及 0F8H 保留未用，如表 1-7 所示。位处理的数据仅为一位二进制数，所以在指令系统中，位地址和字节地址是不难区别的。大多数的传送和逻辑类位操作均围绕着进位位 C 进行，此外一些如位置位、位清 0、位求反等，其位地址在指令中是显而易见的。

顺便指出，在内部数据存储器 128～255 共 128 个字节单元中，特殊功能寄存器只占用了其中 26 个字节，其余单元现无定义，用户不能对这些单元进行读/写操作。若对其进行访问，则将得到一个不确定的随机数。

表 1-6　RAM 位地址

RAM 字节 (MSB)								(LSB) 位
7FH								127
2FH	7F	7E	7D	7C	7B	7A	79	78	47
2EH	77	76	75	74	73	72	71	70	46
2DH	6F	6E	6D	6C	6B	6A	69	68	45
2CH	67	66	65	64	63	62	61	60	44
2BH	5F	5E	5D	5C	5B	5A	59	58	43
2AH	57	56	55	54	53	52	51	50	42

（续）

RAM

字节（MSB）								（LSB） 位
7FH								127
29H	4F	4E	4D	4C	4B	4A	49	48 41
28H	47	46	45	44	43	42	41	40 40
27H	3F	3E	3D	3C	3B	3A	39	38 39
26H	37	36	35	34	33	32	31	30 38
25H	2F	2E	2D	2C	2B	2A	29	28 37
24H	27	26	25	24	23	22	21	20 36
23H	1F	1E	1D	1C	1B	1A	19	18 35
22H	17	16	15	14	13	12	11	10 34
21H	0F	0E	0D	0C	0B	0A	09	08 33
20H	07	06	05	04	03	02	01	00 32
1FH	BanK3							31
18H								24
17H	BanK2							23
10H								16
0FH	BanK1							15
08H								8
07H	BanK0							7
00H								0

表 1-7　特殊功能寄存器位地址

字节地址	D7～D0								寄存器符号名
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	CY	AC	F0	RS1	RS0	OV		P	PSW
B8			PT2	PS	PT1	PX1	PT0	PX0	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3

（续）

字节地址	D7～D0								寄存器符号名
	EA		ET2	ES	ET1	EX1	ET0	EX0	
A8	AF	--	AD	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	91	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
88	8F	8E	8D	8C	8B	8A	89	88	TCON
80	87	86	85	84	83	82	81	80	P0

1.4 I/O 口及相应的特殊功能寄存器

在图 1-1 的 8051 框图中，只表示出了四个双向通道口 P0～P3 的 32 根引脚，而没有像一般微机如 8086 一样明确表示出地址线和数据线。MCS -51 系列单片机，数据线和地址低 8 位分时复用通道口 P0、P3 在程序控制下有第二功能，从而使有限的引脚能完成更多的功能。

四个通道口都具有一种特殊的线路结构，每个口都包含一个锁存器，即特殊功能寄存器 P0～P3，一个输出驱动器和两个（P3 口为 3 个）三态缓冲器。这种结构在数据输出时可以锁存，即在重新输出新的数据之前，口上的数据一直保持不变。但对输入信号是不锁存的，所以外设欲输入的数据必须保持到取数指令执行（把数据读取）完毕为止。为了叙述方便，以下把 1 个端口和其中的锁存器（即特殊功能寄存器）都笼统地表示为 P0～P3，

1.4.1 P0 口

在访问外部存储器时，P0 口是一个真正的双向数据总线口，并分时送出地址的低 8 位和数据。图 1-5 是 P0 口一位的结构图，它包括一个输出锁存器，两个三态缓冲器，一个输出驱动电路和一个输出控制电路。其中输出驱动电路由一对 FET（场效应管）组成，其工作状态受输出控制电路的控制。

当从 P0 输出地址或数据时，控制信号应为高电平'1'，模拟转换开关（MUX）把地址/数据信息经反相器和下拉 FET 接通，同时与门开锁。输出的地址或数据信号既通过与门去驱动上拉 FET，又通过反相器去驱动下拉 FET。例如，

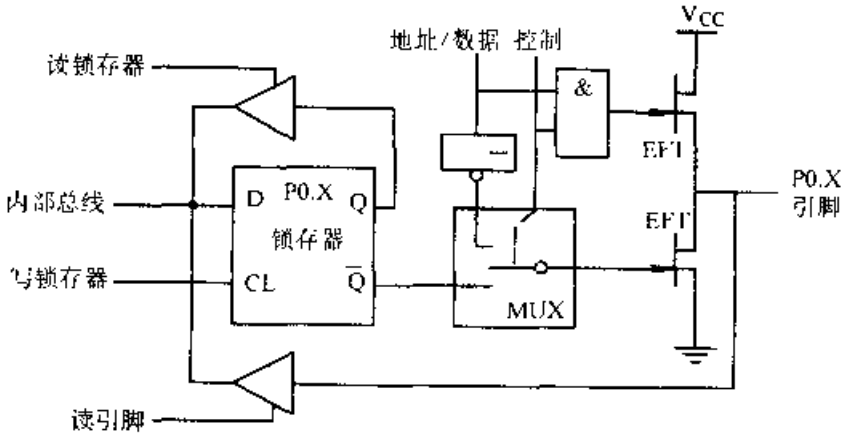


图 1.5 P0 口的位结构

若地址/数据信息为'0'，该'0'信号一方面通过与门使上拉 FET 截止，另一方面经反相器使下拉 FET 导通，从而使引脚上输出相应的'0'信号。反之，若地址/数据信息为'1'，将会使上拉 FET 导通而下拉 FET 截止，引脚上将出现相应的'1'信号。

若 P0 口作为一般 I/O 口使用，在 CPU 向端口输出数据时，对应的输出控制信号应为'0'，模拟转换开关将把输出级与锁存器 $\overline{Q}$ 接通。同时，因与门输出为'0'，使上拉 FET 处于截止状态，因此输出级是漏极开路的开漏电路，必须加上拉电阻。这样，当写脉冲加在触发器时钟端 CL 上时，则与内部总线相连的 D 端数据取反后就出现在 $\overline{Q}$ 端，再经 FET 反相，在 P0 引脚上出现的数据正好是内部总线的数据。

不难看出，P0 口在输出地址/数据信息和作为一般 I/O 口输出数据时，其输出驱动电路的工作状态是有差别的。P0 口做地址/数据总线口使用时，两个 FET 工作于推挽方式，不必外加提升电阻。而作为一般 I/O 口使用时，由于上拉 FET 截止，下拉 FET 处于开漏状态，故需外接上拉电阻。

当 P0 口作普通 I/O 口用于输入数据时，此时上拉 FET 一直处于截止状态。引脚上的外部信号既加在下面的一个内部三态缓冲器的输入端，又加在下拉 FET 的漏极，假定在此之前曾输出锁存过数据'0'，则下拉 FET 是导通的，这样引脚上的电位就始终被钳位在 0 电平，使输入高电平无法读入。因此作为一般 I/O 口使用时，P0 口也是一个准双向口，即输入数据时，应先向锁存器写入'1'，使下拉 FET 截止，然后方可作高阻抗输入。在复位时，P0 口的锁存器的值为 0FFH，所以，对用户而言，P0 口用作普通 I/O 口时可以直接用作输入口。

上面所述为数据由引脚输入的情况，称为"读引脚"操作。但在有些情况下，例如用一根口线去驱动一个晶体管的基极，则向此口线写'1'时，晶体管导通，并把引脚上的电平拉低，这时若从引脚上读取数据，会把此数据错读为'0'。为了避免错读引脚上电平的可能性，单片机中还提供了另一类所谓"读锁存器"操作。这类操

作的特点是：先读口锁存器，随之可对读入的数据进行修改，然后再写到端口上。例如执行指令 P0—P0 x 时，则先把 P0 锁存器的内容读入CPU，然后与变量 x 按位进行逻辑'或'操作，最后把'或'的结果送回 P0 口。能使单片机产生这种读 修改写操作的指令，其目的操作数一般为某 I/O 口或口的某一位，这些指令是：位与（&），位或（ ），取反（～），增 1，减 1 等。

综上所述，P0 口既可作地址/数据总线口，这时它是真正的双向口；也可作通用 I/O 口，但只是一个准双向口，且要加上拉电阻。准双向口工作的特点是：在某引脚由输出状态变为输入时，则应先往对应锁存器写入'1'，以免读错引脚上的信息。一般情况下，若 P0 口已当作地址/数据总线口使用时，就不能再作为通用 I/O 口使用。

复位时，P0 口锁存器均置'1'，8 根引脚可当一般输入线使用。

1.4.2 P2 口

当系统中接有外部存储器时，P2 口可用于输出高 8 位地址，若当作通用 I/O 口用，P2 口则是一个准双向口。通道口 2 的一位线路结构示于图 1-6 中。

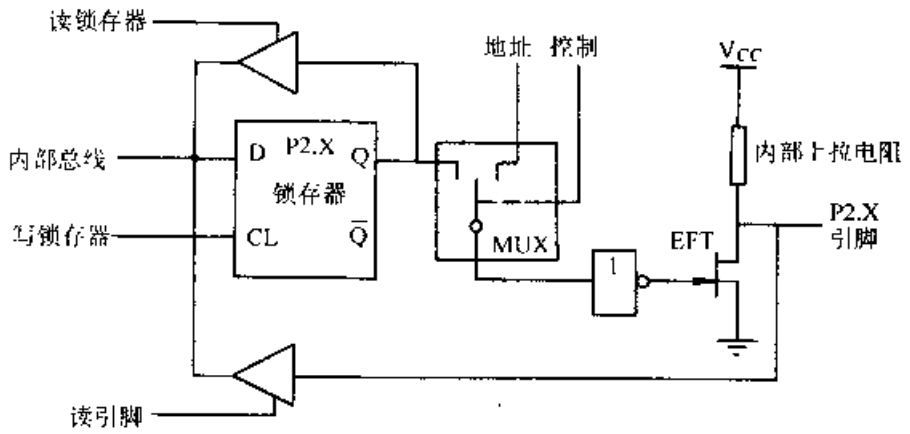


图 1-6 P2 口的位结构

由图可见，线路有一转换器（MUX），由控制信号控制，如果接通右边位置，则地址信号加到输出线路（反相器和输出 FET），并在输出脚上出现地址信号，这相当于 P2 口用于地址输出时的情况。如果转换器接通左边位置，则锁存器与输出线路接通，需要输出的数据通过内部总线锁存入 P2（符合通常输出锁存的常规），这个数据并将出现在口 2 的输出脚上。P2 锁存器中的数据也可以由"读锁存器"接通到内部总线，读回 CPU。

当把口 2 的某一位作为输入位时，首先要在对应锁存器中写入'1'，它使得输出 FET 截止。从而使图中下面一个三态缓冲器输入端逻辑电平随输入信号而变，由"读引脚"信号将外部输入信号接通到内部总线，并读入 CPU。否则和 P0 口作通用 I/O 口使用时相同，不预先在锁存器中写入'1'，则输出 FET 有可能处于导通状

态，从而将口线钳位在低电平上，而不可能把高电平输进来，这就是为什么把其称为准双向口的原因。线路中输入没有锁存，输出为三态，符合通常的习惯。

复位时，8 个锁存器的状态均为高电平，可以当作输入线使用。

由上所述可见，P2 口输出有锁存功能，作准双向口使用输入时要先向口写 1。每根引脚既可以作地址输出，也可以作数据输出和输入。例如：口 2 可以作为高 8 位程序计数器（PCH）输出（程序地址），高 8 位数据地址指针（DPH）输出或者作通用 I/O 口使用。但一般来说，若 P2 口已外接程序存储器，由于访问外部存储器的操作不断，P2 口不断送出高 8 位地址，故这时 P2 口不可能再作通用 I/O 口使用。只有在仅连接外部数据存储器的系统中，可视访问外部数据存储器的频繁程度或扩充外部数据存储器容量的大小，在一定限度内作一般 I/O 口使用。这要结合外部存储器执行时序予以具体分析。

由图还可看出，在输出驱动器部分，P2 口也有别于 P0 口，它接有内部上拉电阻。实际中的上拉电阻是由作阻性元件使用的场效应管 FET 组成的，可分为固定部分和附加部分，其附加部分是为加速输出由"0～1"的跳变过程而设置的。对于 P1、P2、P3 口，输出级结构是相同的。这种驱动部分接有内部上拉电阻的结构，使 P1～P3 口输出时能驱动 3 个 LSTTL 输入，而不必外接提升电阻就可驱动任何 CMOS 输入。

1.4.3 P3 口

P3 口是一个双功能口，第一功能和 P2 口一样可作为通用 I/O 口，每位可定义为输入或输出，且是一个准双向口。P3 口工作于第二功能时，各位的定义如表 1-8 所示。

表 1-8 P3 口各位第二功能定义

P3.0	RXD（串行输入通道）
P3.1	TXD（串行输出通道
P3.2	$\overline{INT0}$（外中断 0）
P3.3	$\overline{INT1}$（外中断 1）
P3.4	T0（定时器 0 外部输入）
P3.5	T1（定时器 1 外部输入）
P3.6	$\overline{WR}$（外部数据存储器写选通）
P3.7	$\overline{RD}$（外部数据存储器读选通）

由图 1-7 中 P3 口的位结构可以看出，实现第一功能作通用输入/输出口时，选择输出功能端应保持高电平，使与非门对锁存器 Q 端是畅通的。同理，实现第二功能做专用信号输出时（如送出$\overline{WR}$、$\overline{RD}$等信号），则该位的锁存器应置 1，使与非门对选择输出功能端是畅通的。但对输入而言，无论该位是作通用输入口或作第二功能输入口，相应的输出锁存器和选择输出功能端都应置 1。实际上，由于

MCS—51 系列单片机所有口锁存器在上电复位时均置为'1'，自然满足了上述条件，所以用户不必做任何工作，就可以直接使用 P3 口的第二功能。而在确信某一引脚第二功能所可提供的信号不用（或不会产生）时，该引脚才可作 I/O 线使用，这同一般准双向口的 I/O 引脚使用方法相同。

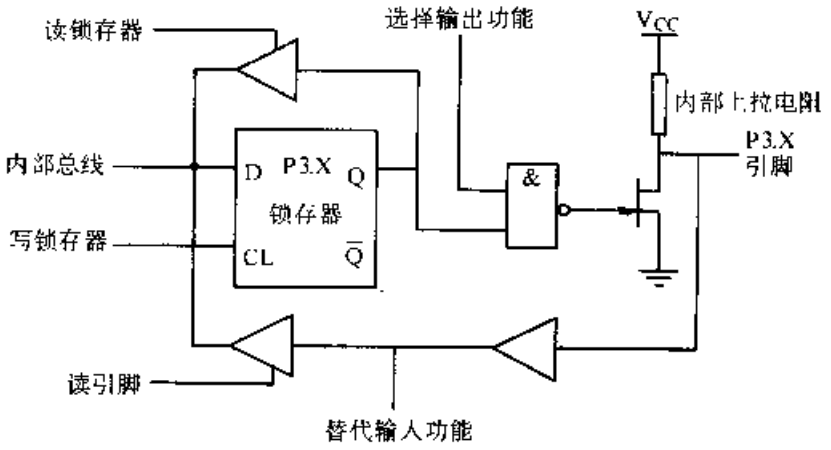


图 1-7　P3 口的位结构

图下面的输入通道中有两个缓冲器，第二功能的专用输入信号取自第一个缓冲器输出端，通用输入信号取自"读引脚"缓冲器的输出端。

1.4.4　P1 口及各端口的使用特点

P1 口是一个标准的准双向口，其位结构见图 1-8。在组成应用系统时，它往往作通用的 I/O 口使用。只是在 8032/8052 中，P1.0 和 P1.1 是多功能的。除作一般双向 I/O 口使用外，P1.0 还用来作定时器/计数器 2 的外部输入端，并以标识符 T2 表示；P1.1 则可作为定时器/计数器 2 的外部控制输入，以 T_{2EX}表示。

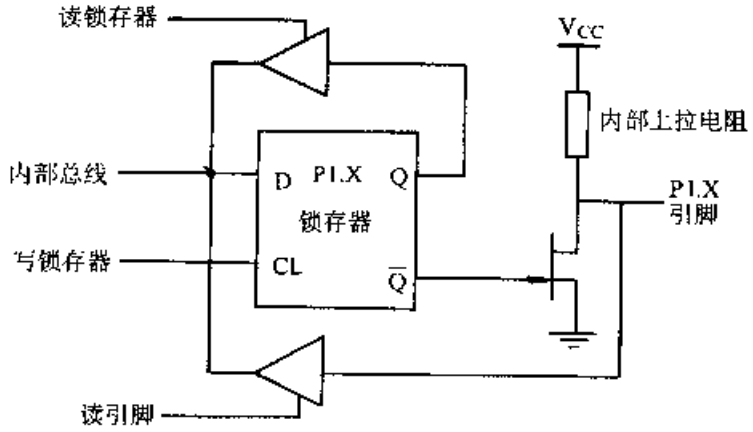


图 1-8　P1 口的位结构

至此，可对组成一般单片机应用系统时各个口的分工小结如下：

P0 口：地址低 8 位与数据线分时使用口。

P1 口：按位可编程的输入输出口。

P2 口：PC 高 8 位、DPTR 高 8 位或 I/O 口。

P3 口：双功能口，第二功能定义如上所述，若不用第二功能，也可作一般 I/O 口。

组成应用系统时，端口最常用来进行系统的扩展。例如实现单片机和存储器及输入/输出接口的连接，也可直接利用端口进行单片机和外设间的信息传送，这就必须考虑端口的负载能力。一般 P1、P2、P3 口的输出能驱动 3 个 LSTTL 输入，口 P0 的输出能驱动 8 个 LSTTL 输入。它们可直接驱动固态继电器工作。对 CMOS 输入而言，口 P1、口 P2 和口 P3 无须外加提升电阻就可直接驱动，口 P0 需外加提升电阻。但当口 P0 用作地址/数据线时，它可直接驱动 CMOS 输入而不必外加提升电阻，这些在实际使用时都应予以注意。

新型 8×51 单片机产品的端口 I/O 驱动能力有较大提高，如 ATMEL 公司的 89C51、89C52 等产品，其端口输出电流达到 20mA，可以直接驱动 LED 显示。

1.4.5 利用端口组成的 8031 应用系统举例

作为端口使用的一个实例，先介绍一下应用 8031 的最小系统。

由于 8031 无片内程序存储器，使用时需外加 EPROM 作为外部程序存储器，所以 8031 的 P0 口和 P2 口实际上只能作为外部程序存储器、外部数据存储器和 I/O 扩展电路的地址/数据信号输出，而不能直接用来作为输入/输出口外接 I/O 设备。对于 8031 应用系统来说，P0 口和 P2 口相当于地址和数据总线接口，P2 口作为外部存储器地址的高 8 位输出口，P0 口作为外部存储器地址低 8 位输出和数据总线口，按照上述原则组成的 8031 最小系统如图 1-9 所示。

图中由于只用一片 2716（EPROM）作为程序存储器，所以剩余的 P1 口和 P3 口均可作为 I/O 口使用。控制信号 ALE 和$\overline{\text{PSEN}}$均由单片机输出产生，其中地址锁存信号 ALE 用来把 P0 口地址低 8 位打入外部地址锁存器 74LS373。由图 1-10a 的时序图可以看出，ALE 信号在每一个机器周期中两次有效，在 ALE 由高变低时，有效地址 PCL 出现在 P0 总线上，低 8 位地址锁存器此时应在 ALE 控制下把地址锁存起来。同时$\overline{\text{PSEN}}$也是每个机器周期两次有效，用于选通外部程序存储器，使指令送到 P0 总线上，由 CPU 取入。

由图可见，ALE 信号利用其由高变低的下降沿对出现在 P0 口上的地址低 8 位进行锁存，而在其低电平（ALE 无效）期间读进指令数据，这就实现了 P0 口地址/数据的分时传送。而程序存储器允许信号$\overline{\text{PSEN}}$只在访问外部程序存储器时产生，由于程序存储器是只读的，所以$\overline{\text{PSEN}}$信号连接在 EPROM 的输出允许端$\overline{\text{OE}}$上，而把其片选$\overline{\text{CE}}$接地，使其常通。这样，PCH 由 P2 口输出锁存（P2 口本身即是锁存器），PCL 由 P0 口分时送出并经 74LS373 输出锁存，它们均分别接至外部存储器的地址端，这就使得指令字节的读出只受程序计数器 PC 提供的地址信号和单片机产生的时序信号$\overline{\text{PSEN}}$的控制。

如果 8031 单片机要扩充外部数据存储器和 I/O 口，由于数据 RAM 是可读可

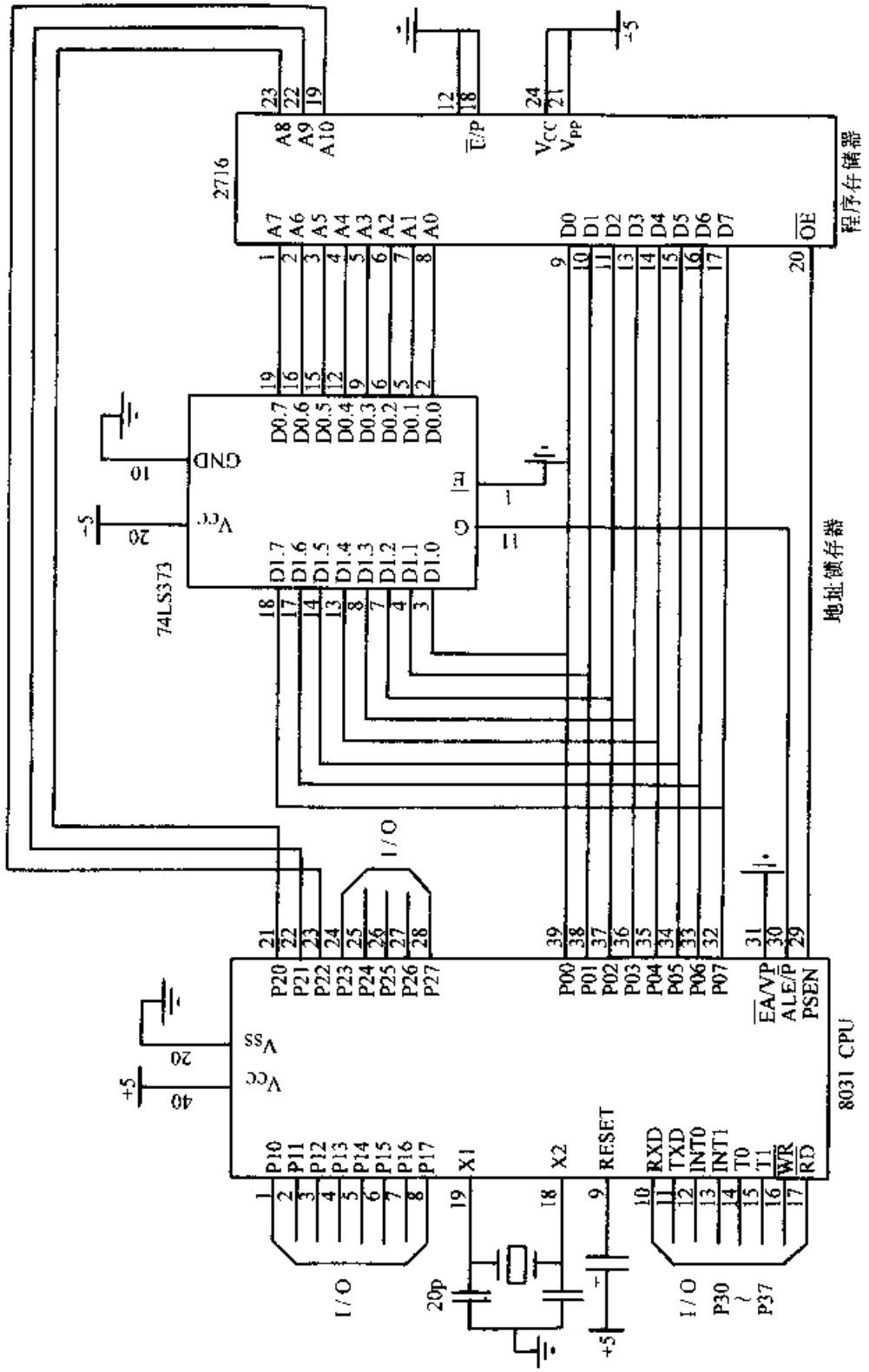


图 1-9　8031 最小系统

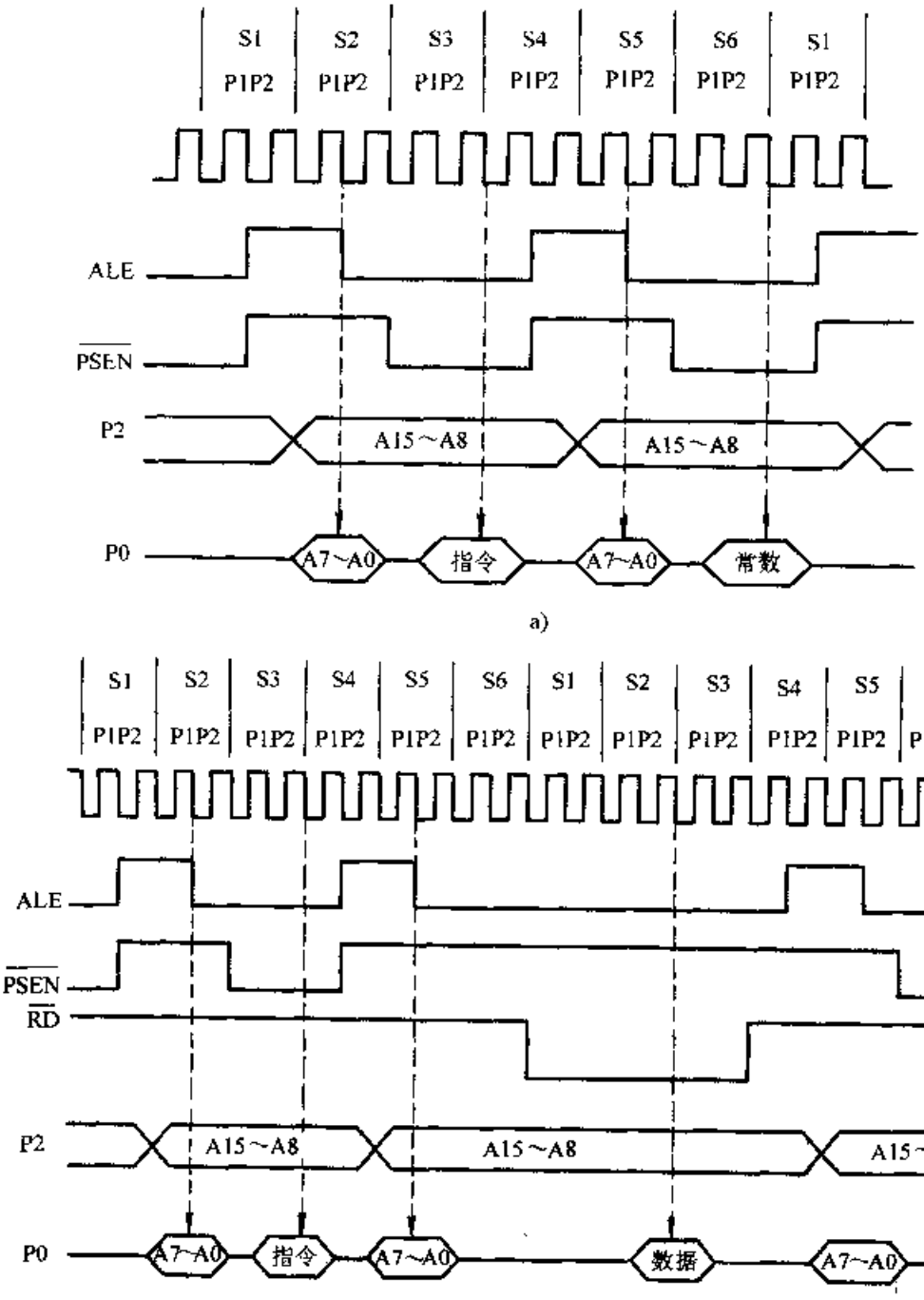


图 1-10　8051 程序、数据存储器操作时序

a) 访问外部程序存储器　b) 访问外部数据存储器

写的，单片机和 I/O 口亦要进行信息交换，所以提供了专门的读写信号$\overline{RD}$和$\overline{WR}$予以选通，8031 应用系统的一般组成如图 1-11 所示。

对程序存储器的操作如前所述，而对外部数据存储器的操作时序（见图 2-10b）可简述如下：在同一周期的 S5 状态，ALE 利用其下跳沿锁存 P0 总线上出现的地址低 8 位（由 DPTR 低 8 位或 R0、R1 间接提供的地址低 8 位值），而 P2

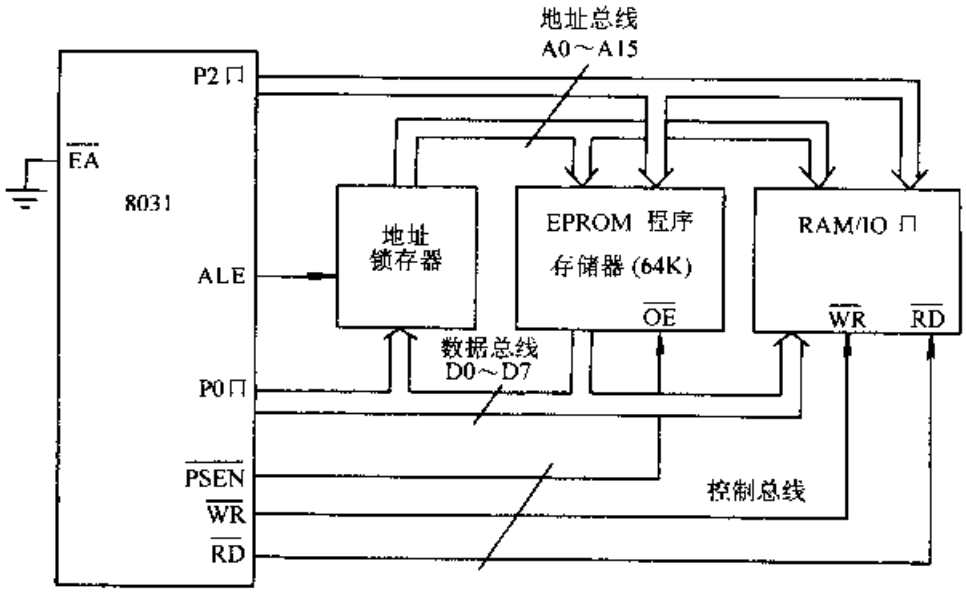


图 1-11　8031 应用系统的典型连接

口上将出现地址高 8 位值（DPTR 的高 8 位或 P2 锁存器内容）。由于执行 MOVX 指令，会产生相应的$\overline{RD}$（或$\overline{WR}$）有效信号，在 P0 总线将出现有效的输入数据（或输出数据）供 CPU 存取。对外部数据存储器操作期间，在同一机器周期的 S6 状态将不再出现$\overline{PSEN}$有效信号，下一个机器周期的第一个 ALE 也不再出现。

1.5　MCS—51 单片机的引脚信号和 CPU 时序

用单片机组成应用系统，往往需要对存储容量和 I/O 接口加以扩充，为保证连接无误和速度匹配，就需要熟悉和了解单片机的引脚信号和 CPU 的时序过程。

1.5.1　8051 单片机引脚功能说明

MCS—51 系列单片机是一个具有 40 根引脚的双列直插式器件，4 个并行口共有 32 根引脚，可分别用作地址线，数据线和 I/O 线，另外还有 6 根控制信号线，两根电源线。其芯片管脚图见图 1-12。

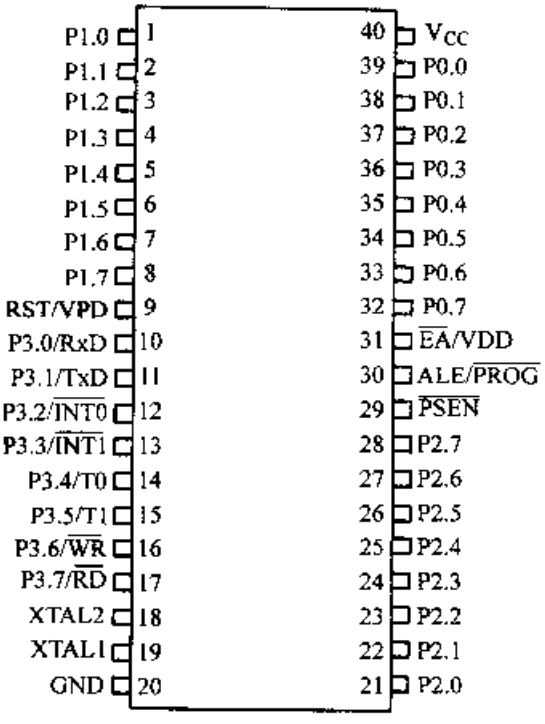


图 1-12　8031、8051、8751 芯片管脚图

图中各引脚的功能如下：

1. P0～P3 口的引线

通道 0：通道 0 是一个 8 位漏极开路的双向 I/O 通道。在存取外存储器时用作低 8 位地址及数据总线（此时内部上拉电阻有效）。在程序检验时也用作输出指令字节（在程序检查时需要外部上拉电阻），通道 0 能连接 8

个 LSTTL 输入。

通道 1：通道 1 是一个带内部上拉电阻的 8 位双向 I/O 通道。在 8751 或 8051 的程序检验时，它接收低 8 位地址字节。

通道 1 能吸收或供给 3 个 LSTTL 输入，它不用附加上拉电阻即可驱动 MOS 输入。

通道 2：它是一个带内部上拉电阻的 8 位双向 I/O 通道。在存取外存储器时，它是高位地址字节的出口。在 8051 或 8751 的程序检查中，它也能接受高位地址和控制信号。

通道 2 能吸收或供给三个 LSTTL 输入，它不用外加上拉电阻即可驱动 MOS 输入。

通道 3：是一个带内部上拉电阻的 8 位 I/O 通道。它还能用于实现 MCS—51 系列的各种特殊功能，如表 1-4 所示。

通道 3 能够吸收或供给三个 LSTTL 输入，不用外加电阻即可驱动 MOS 输入。

2. 控制信号

ALE/PROG：地址锁存允许输出。在存取片外存储器时，锁存低位地址字节。为了达到这个目的，甚至在片外存储器不作存取时，也以时钟振荡频率 1/6 的固定频率激发 ALE。因此它可以用于外部时钟和定时（然而，在每一次存取片外数据存储器时，会丢失一个 ALE 脉冲）。在进行 EPROM 编程时，该端线还是编程脉冲输入端（PROG）。

$\overline{\text{PSEN}}$：程序存储器输出允许。是片外程序存储器的读选通信号。从片外程序存储器取数时，每个机器周期内$\overline{\text{PSEN}}$激发两次（然后，当执行片外程序存储器的程序时，$\overline{\text{PSEN}}$在每次存取片外数据存储器时，有两个脉冲是不出现的）。从内部程序存储器读取指令时，不激发$\overline{\text{PSEN}}$。

对 8031 而言，访问外部程序存储器时，将 PC 的十六位地址输出到 P2 口和 P0 口外部的地址存储器后，$\overline{\text{PSEN}}$产生负脉冲允许片外程序存储器输出数据。相应的存储单元的指令字节送到 P0 口，供 8051 读取。

$\overline{\text{PSEN}}$、ALE 和 XTAL2 输出端是否有信号输出可以判断出 8051 是否在工作。

$\overline{\text{EA}}$/VDD：当$\overline{\text{EA}}$为高电平时，CPU 执行片内程序存储器指令（除非程序计数器超过 0FFFH）。当$\overline{\text{EA}}$为低电平时，CPU 只执行片外程序存储器指令。在 8031 中$\overline{\text{EA}}$必须外接到 GND，在 8751 中，当 EPROM 编程时，它也接收 21V 的编程电源电压（VDD）。

XTAL1：作为振荡器倒相放大器的输入。使用外振荡器时，必须接地电位。

XTAL2：作为振荡器的倒相放大器的输出和内部时钟发生器的输入。当使用

外振荡器时，接收外振荡器信号。

RST/VPD：复位输入。当振荡器工作时，在此端线持续给出两个机器周期的高电平可以完成复位。由于有一个内部的下拉电阻，只需要在本端和 V_{CC} 端之间加一个电容，便可以做到上电复位。

复位以后，P0～P3 口输出高电平，SP 指针重新赋值为 07H，其他特殊功能寄存器和程序计数器 PC 被清 0。复位后各内部寄存器初态如表 1-9 所示。

表 1-9　MCS—51 复位后内部寄存器初态

特殊功能寄存器	初始状态	特殊功能寄存器	初始状态
ACC	00H	TMOD	00H
B	00H	TCON	00H
PSW	00H	TH0	00H
SP	07H	TL0	00H
DPL	00H	TH1	00H
DPH	00H	TL1	00H
P0～P3	0FFH	SCON	00H
IP	×××00000B	SBUF	不定
IE	0××00000B	PCON	0×××××××B

只要 RESET 保持高电平，8051 就会循环复位。RESET 由高电平变为低电平后，8051 从 0 地址开始执行程序。8051 初始复位不影响内部 RAM 的状态，包括工作寄存器 R0～R7。

常见的复位电路连接如下：

1）上电复位电路如图 1-13 所示。在通电瞬间，由于 C_R 通过 R_R 充电，在 RST 端出现正脉冲，8051 加电后自动复位。C_R、R_R 随 CPU 时钟频率而变化，可由实验调整。当采用 6MHz 时钟时 C_R 为 22μF，R_R 为 1kΩ 时，便能可靠复位。

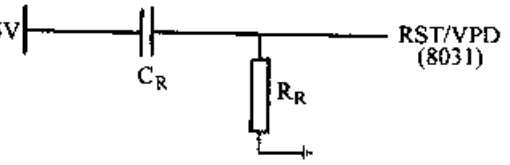


图 1-13　上电复位电路

2）电平方式开关复位电路和脉冲方式开关复位电路如图 1-14 所示。复位电路中的电阻、电容参数和 CPU 采用的时钟频率有关，由实验调整。在实际的 8051 应用系统中，外部扩展的 I/O 口电路也需初始复位，如果和 8051 的复位端相连也将影响复位电路中的 RC 参数。也可以采用独立的外围接口上电自动复位电路。

为了可靠实现复位信号，并且能在计算机受到干扰使程序不能正常运行时自动产生复位信号，各集成电路厂家研制出微机监控电路 WDT（看门狗）。现以 DALLAS 产品 DS1232 为例说明其功能和使用。

DS1232 为 8 引脚芯片，如图 1-15 所示。

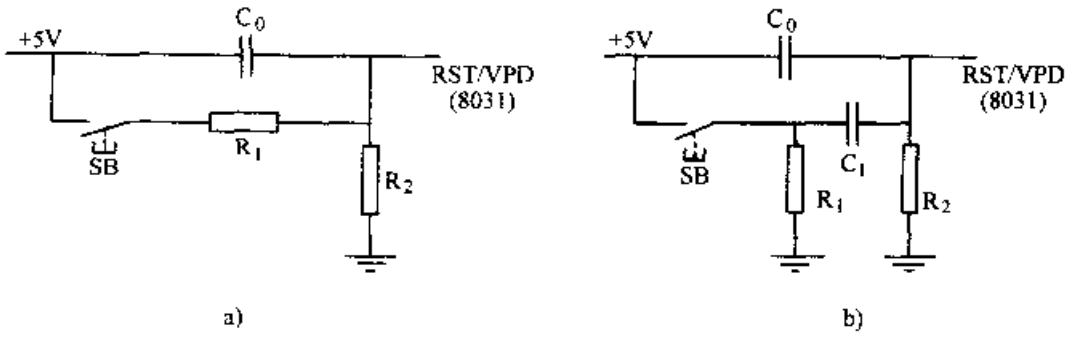


图 1-14　开关复位电路

a）电平方式开关复位　b）脉冲方式开关复位

DS1232 的主要功能：提供高电平和低电平两种复位信号；可以产生上电复位和手动复位；可以监视电源电平，当电平低于一定值时产生复位信号；可以监视软件运行状态，当程序运行出现飞车时，产生复位信号。

DS1232 引脚说明：

TD（2）：WDT 定时器超时周期设置，接地为 150ms，悬空为 600ms，接 V_{CC}为 1.2s。

TOL（3）：电源电平监测门限 5%或 10%选择，接地为 5%，接 V_{CC}为 10%。

ST（7）：WDT 定时器复位信号输入，在选定的超时周期内，在 ST 引脚产生一下降沿信号，就复位 WDT 定时器；若在选定的超时周期内在 ST 引脚没有产生一下降沿信号，就产生复位信号输出。

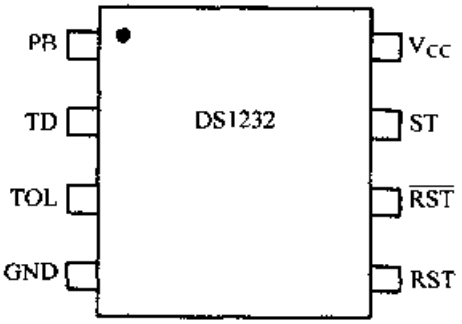


图 1-15　DS1232 管脚图

PB（1）：从该引脚接一开关接地，构成手动复位开关。开关闭合时，PB 接地，产生复位输出信号。

DS1232 的使用：由于 8051 系列单片机需要高电平复位信号，将 DS1232 的 RST 接 8051 的 RESET（9），将 TD 和 TOL 接 V_{CC}，PB 通过一开关接地，8051 的一个 I/O 引脚接 ST。RST 与 V_{CC}之间接一上拉电阻。

软件设计时，在程序的主循环中和较长的函数中，要保证在设定的超时周期时间内在 ST 输出一个脉冲信号。

3. 电源线

V_{CC}：在编程（用于 8751）检验（8051 或 8751）和正常运行时的电源，接+5V。

V_{SS}：接地端。

一般 V_{CC}和 V_{SS}之间应接有高频和低频滤波电容。

1.5.2　CPU 时序

单片机的基本操作周期为机器周期，一个机器周期可分 6 个状态，每个状态由两个脉冲组成（所谓两相）。前一个脉冲叫 P1（相位 1），后一个脉冲叫 P2（相

位 2)，所以，一个机器周期共有 12 个振荡脉冲，如图 1-16 所示。一般情况下，算术和逻辑操作发生在 P1 期间，而内部寄存器到寄存器传输发生在 P2 期间。

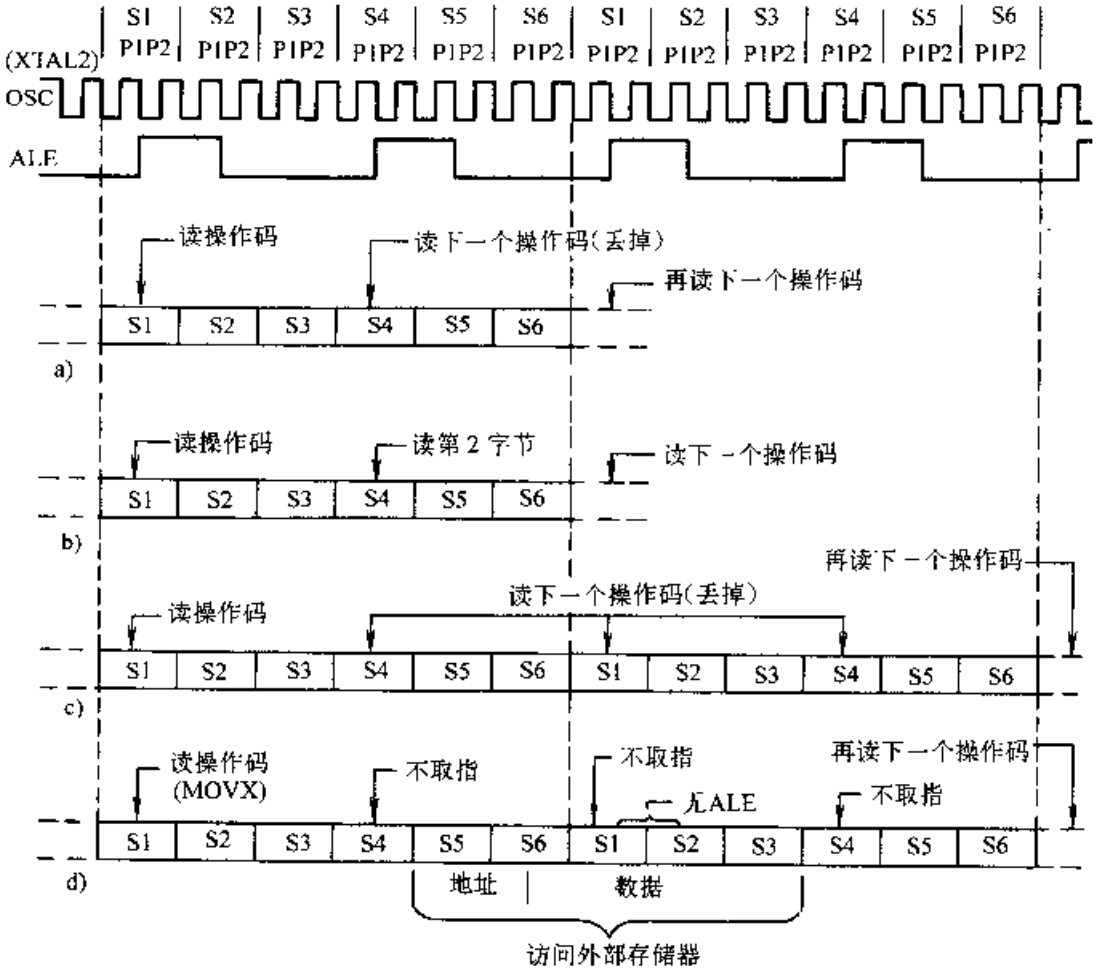


图 1-16　典型指令的取指/执行时序

a）单字节单周期指令，如 INC A　b）双字节单周期指令，如 ADD A，#data

c）单字节双周期指令，如 INC DPTR　d）MOVX（单字节双周期）

图 1-16 所示的是存取/执行时序的内部状态和相位。由于内部时钟信号在外而是无法观察到的，所以画出了外部 XTAL2 的振荡信号和 ALE（地址锁存允许）信号供参考。如图，一个机器周期包含 12 个振荡周期，编号为 S1P1（1 状态 1 相）到 S6P2（6 状态 2 相），每一相持续一个振荡周期，每一个状态持续两个振荡周期。在每个机器周期中，ALE 信号两次有效，一次在 S1P2 和 S2P1 期间，还有一次在 S4P2 和 S5P1 期间。

执行一条单周期指令时。在 S1～S3 期间读入操作码并把它锁存到指令寄存器中。如果是一条双字节指令，第二个字节在同一机器周期的 S4～S6 期间读出。如果是一条单字节指令，在 S4～S6 期间仍然有一个读操作，但这时读出的字节（下一条指令的操作码）是不加处理的，而且程序计数器也不加 1。不管上述哪一

种情况，指令都在S6P2期间执行完毕。图1-16a和b分别显示了单字节、单周期指令和双字节、单周期指令的时序。

绝大多数的8051指令是在一个周期内执行的。只有MUL（乘）和DIV（除）指令需用4个周期来完成。通常在每一个机器周期中，从程序存储器中取两个字节码，仅在执行MOVX指令时是例外。MOVX是一条单字节、双周期指令，用于存取片外数据存储器数据。执行MOVX指令时，仍在第一个机器周期的S1～S3期间读入其操作码，而在S4～S6期间也执行读操作，但读入的下一个操作码不予处理（因为MOVX是单字节指令）。由第一机器周期的S5开始，送出片外数据存储器的地址，随后读或写数据，直到第二个机器周期的S3结束，此期间不产生ALE有效信号。而在第二机器周期S4期间，由于片外数据存储器已被寻址和选通，所以也不产生取指操作。图1-16c和d显示了通常的单字节、双周期和MOVX指令的时序。

1.5.3 程序和数据存储空间的重合

最后，作为引脚连接和CPU时序的一个应用例子，介绍实际中常常遇到并要加以解决的一个问题：如何实现程序和数据存储器空间的重合。

8051单片机程序和数据存储器地址空间互相独立，可达128K字节。但在程序调试阶段，需要把指令送入存储器中，执行时又要能方便读出，同时，调试过程中指令也可能要随时修改，这就需要可读可写的存储器。然而8051中的程序存储器却是只读的，无法满足要求。为了解决这一问题，使同一地址区域中的EPROM和RAM芯片能够互换使用，可以采用如图1-17所示的办法，即将8051的片外程序存储器读选通线（$\overline{PSEN}$）与片外数据存储器读选通线（$\overline{RD}$）采用物理组合的方法，用一与门电路来解决上述问题。

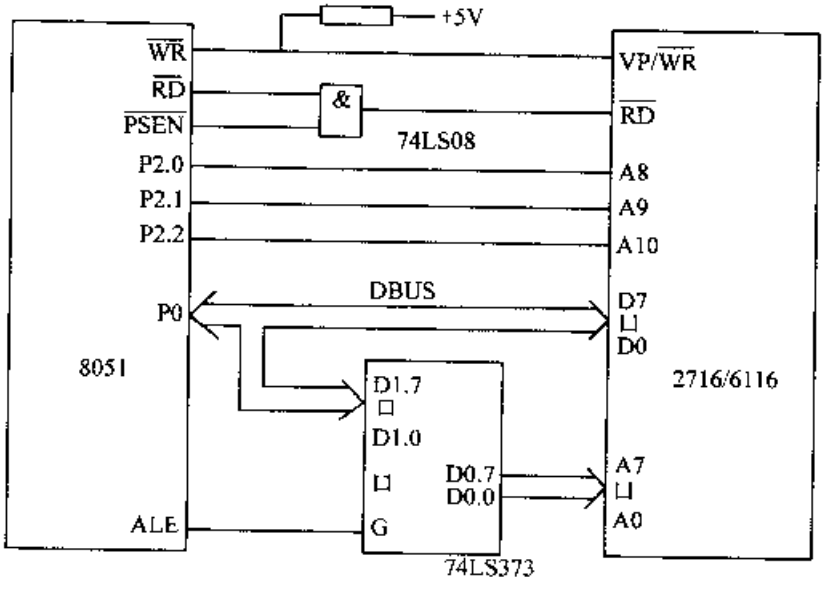


图1-17 程序和数据存储空间的重合

按图中的接线方式，在存储器插座上既可插入EPROM，又可不做任何变动插入静态RAM，同时对两种存储器的取指令也可任意使用，给软件编写及调试带来了方便。但需要注意的是因为$\overline{\text{PSEN}}$周期比$\overline{\text{RD}}$短，故在选择芯片时要使其工作频率快到足以适应$\overline{\text{PSEN}}$周期。

习题和思考题

1. 请你分别写出一个MCS--51中ROM、EPROM、无ROM型单片机的型号和内部资源。其中哪个产品内部具有固化的软件？该软件能否被所有的用户所利用？怎样使用市售的该种产品？
2. 试根据P1口和P3口的结构特性，指出它们作为输入口或第二功能输入/输出的条件。
3. MCS—51中无ROM型单片机，在应用中P2口和P0口能否直接作为输入/输出I/连接开关、指示灯之类的外围设备？为什么？
4. 什么是堆栈？8032的堆栈区可以设在什么地方？一般应设在什么区域？如何实现？试举例说明。
5. 8031的内部RAM中，哪些可以作为数据缓冲区？

第二章　MCS—51 单片机的指令系统

2.1　MCS—51 的寻址方式

2.1.1　MCS　51 汇编语言的指令格式

MCS　51 汇编语言指令格式如下：

[标号：]　　操作码 [第一操作数，] [第二操作数] [；注释]

其中：[] 中的项为任选项，操作码与操作数之间用空格分隔，操作数与操作数之间用逗号分隔。

如：标号：　　操作码　　第一操作数，第二操作数　　；注释

LOOP：　　MOV　　A，　　30H　　；(A)←(30H)

1. 标号

标号是由字母开头的字母数字序列，如 line1。MCS—51 汇编语言程序中，一条指令占一行。每一行可以有一个标号。每一个子程序的第一条语句都必须有一个标号，以标识该子程序的入口地址，也充当该子程序的名字。

2. 操作码

每一条汇编指令都完成指定功能，这由指令的操作码表示，即操作码是指令的助记符，如 ADD 表示加法指令，MOV 指令表示数据传送等。

3. 操作数

指令中的操作数是指令的操作对象，有的指令没有操作数，有的指令只有一个操作数，而有的指令需要两个操作数。如 RET、RETI 指令为子程序返回指令，指令的操作是明确的，即从堆栈中将程序计数器的值弹出，让程序返回断点处继续执行，因而指令不需要操作数；INC　A 指令是将累加器的值加 1，它的操作数只有一个；ADD　A，＃23H 将数 23H 加到累加器 A 中，它就需要两个操作数。

4. 注释

每一条指令都可以有注释，加注释对以后阅读和修改程序有很大帮助。一条指令的后面可以有一个';'，';'后的内容是注释，汇编程序在编译程序时忽略注释。

2.1.2　MCS—51 的寻址方式

寻址方式就是汇编语言指令中表示操作数的方式。MCS—51 单片机有七种寻址方式。

1. 寄存器寻址

用某个工作寄存器存放操作数，即指令中的操作数为某个工作寄存器的内容。

```
如：    MOV   A，R0          ;(A)←(R0)，寄存器 R0 的内容是操作数
        ADD   A,R5           ;(A)←(A)+(R5)寄存器 R5 的内容是操作数
```

使用寄存器寻址方式时，工作寄存器在指令中以 Rn 的形式出现，n=0～7，由 PSW 的 RS0 和 RS1 两位确定是哪一组工作寄存器。

2. 直接寻址

指令中直接给出操作数的内部 RAM 单元地址。

```
如：    MOV   A，45H         ；(A) ← (45H)
```

此时内部 RAM 的 45H 单元的内容是操作数。

可用于直接寻址的空间是：内部 RAM 的低 128 字节（包括可位寻址区），特殊功能寄存器区（SFR）。对特殊功能寄存器区，这是惟一的寻址方式。

3. 寄存器间接寻址

用某个工作寄存器 Ri 存放操作数的地址，i 的值只能是 0 或 1，即能用于这种寻址方式的工作寄存器只有 R0 和 R1。

```
如：    MOV   A，@R1         ；(A) ←((R1))
```

此时 R1 的内容是操作数的地址。若 R1 的内容为 35H，内部 RAM35H 单元的内容是 45，该指令将 45 送入累加器 A。

使用 R0 和 R1 可以寻址外部数据寄存器 0～255 空间的单元，而使用 DPTR 可以寻址外部数据寄存器 64K 空间的单元。

```
如：    MOVX  A，@R0         ；(A) ←((R0))
        MOVX  A，@DPTR       ；(A) ←((DPTR))
```

对于 8052 系列单片机，其高 128 字节内部 RAM 的地址与特殊功能寄存器地址重合，故必须用这种寻址方式寻址。而特殊功能寄存器必须用直接寻址方式寻址。靠寻址方式的不同可以区别操作数所处的空间。

4. 立即寻址

指令中直接给出操作数。

```
如：    MOV   A，#2FH        ；(A) ←2FH
        ADD   A，#0E3H       ；(A) ← (A) +E3H
```

注意其中的'#'符号，数字前冠以'#'表示立即数。直接寻址方式时，地址操作数前没有'#'符号。

5. 变址寻址

用于对存放于程序存储器中的数组的操作。使用时通常用 DPTR 存放数组首址，A 中存放数组元素的偏移量（无符号数），两者的和作为实际操作数的地址。

如：　　MOVC　A，@A+DPTR　　；(A) ←((A)+(DPTR))

变址寻址的作用空间是程序存储器中存放的数据表（数组）。

DPTR 称为基址寄存器，能用作基址寄存器的还有程序计数器 PC。

如：　　MOVC　A，@A+PC　　；(A) ←((A)+(PC))

6. 相对寻址

以程序寄存器 PC 作为基址寄存器，指令中给出偏移量（有符号数）rel，PC 的当前内容（源地址）与 rel 之和给出了操作数的实际地址。相对寻址主要用于跳转指令。

如：　　SJMP　rel　　；PC← (PC) +2+rel

7. 位寻址

对内部 RAM 的可位寻址空间及特殊功能寄存器中可位寻址的单元采用直接寻址方式寻址。

2.2　MCS—51 的指令说明

MCS—51 分为数据传送类、逻辑操作类、算术运算类、位操作和控制转移类指令。

2.2.1　数据传送类指令

数据传送类指令是最常用、最基本的一类指令。数据传送类指令分为两类，一类为单纯的数据传送，即把源操作数传送给目的操作数而源操作数不变，表示为源操作数→目的操作数。另一类为数据交换，即源操作数和目的操作数相互交换位置，表示为源操作数←→目的操作数。而按作用区域可将数据传送类指令分为内部数据传送指令和外部数据传送指令。

1. 内部数据传送指令

指令格式：　　MOV〈目的字节〉,〈源字节〉

功能：传送字节数据

(1) 立即数送累加器 A 和内部数据存储器（Rn，内部 RAM，SFR）

```
MOV    A,      #data     ;(A)←#data
MOV    direct, #data     ;(direct)←#data
MOV    @Ri,    #data     ;((Ri))←#data,i 取 0 或 1
MOV    Rn,     #data     ;(Rn)←#data,n 取 0～7
```

(2) 内部数据存储器（Rn、内部 RAM、SFR）与累加器 A 之间的数据传送

```
MOV    A,      direct    ;(A)←(direct)
MOV    A,      @Ri       ;(A)←((Ri)),i 取 0 或 1
MOV    A,      Rn        ;(A)←(Rn),n 取 0～7
```

```
MOV     direct,   A          ;(direct)←(A)
MOV     @Ri,      A          ;((Ri))←(A),i 取 0 或 1
MOV     Rn,       A          ;(Rn)←(A),n 取 0～7
```

(3) 内部数据存储器中 Rn、SFR 和片内数据 RAM 之间的数据传送

```
MOV     direct2   direct1    ;(direct2)←(direct1)
MOV     direct,   @Ri        ;(direct)←((Ri)),i 取 0 或 1
MOV     direct,   Rn         ;(direct)←(Rn)
MOV     @Ri,      direct     ;((Ri))←(direct),i 取 0 或 1
MOV     Rn,       direct     ;(Rn)←(direct),n 取 0～7
```

指令中，direct 表示直接地址。#data 表示立即数。A 是累加器，它是一个特殊功能寄存器 (SFR)，因此它表示直接地址。@Ri 表示寄存器间接寻址，@符号表示间接寻址，((Ri)) 表示把立即数送到由 Ri 寄存器的内容所指出的那个 RAM 单元中去。

(4) 目标地址传送

```
MOV     DPTR, #data16    ; (DPTR) ←#data16
```

2. 数据交换指令

(1) 字节交换指令

```
XCH     A,      direct       ; (A) ← (direct)
XCH     A,      @Ri          ; (A) ←((Ri)), i 取 0 或 1
XCH     A,      Rn           ; (A) ← (Rn), n 取 0～7
```

(2) 半字节交换指令

```
XCHD    A, @Ri;              ; (A3～0) ←→((Ri)3～0)
```

3. 栈操作指令

在 8051 单片机内部 RAM 中有一先进后出、后进先出的区域称为堆栈。特殊功能寄存器 SP 作为堆栈指针，始终指向栈顶。

(1) 进栈指令

```
PUSH   direct                ;(SP)←(SP)+1,(SP)←(direct)
```

功能：先将堆栈指针 SP 的内容加 1，然后将直接地址 direct 中的内容送到 SP 所指的内部 RAM 单元。

(2) 出栈指令

```
POP     direct               ; (SP) ← (direct), (SP) ← (SP) －1
```

功能：先将 SP 所指的内部 RAM 单元的内容送到直接地址 direct 指向的单元中，然后堆栈指针 SP 的内容减 1。

如：　　PUSH　ACC

```
POP     ACC
```

4. 外部数据传送指令

外部数据传送指令是指累加器 A 和外部 RAM 之间以及累加器 A 和外部 ROM 之间的数据传送。外部 RAM 各单元之间，以及外部存储器与内部存储器之间的数据传送只能通过累加器 A 间接进行。

(1) 外部 RAM 与累加器 A 之间的数据传送指令

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

(2) 外部 ROM 向累加器 A 传送指令

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

2.2.2 逻辑操作类指令

逻辑操作类指令有单操作数指令和双操作数指令。单操作数指令以 ACC 为操作对象，主要有清零、取反及移位指令。双操作数指令包括与、或及异或。

(1) 累加器清零及取反指令

```
CLR     A       ; (A) ←0        (清零)
CPL     A       ; (A) ←～ (A)(累加器取反)
```

(2) 移位指令

```
RL      A       ; 累加器左环移
RLC     A       ; 累加器带进位左环移
RR      A       ; 累加器右环移
RRC     A       ; 累加器带进位右环移
SWAP    A       ; 交换累加器 ACC 的高低半字节内容
```

RL、RLC 指令示意见图 2-1 所示。

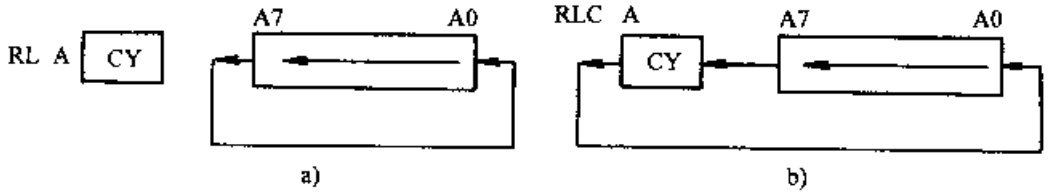


图 2-1 RL、RLC 指令示意图

a) RL 指令 b) RLC 指令

(3) 逻辑与指令

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

(4) 逻辑或指令

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

(5) 逻辑异或指令

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

逻辑操作是按位进行的。所以，逻辑与指令常用来屏蔽字节中的某些位，该位欲清除用'0'去'与'，该位欲保留用'1'去'与'；逻辑或指令常用来对字节中某些位置'1'，欲保留（不变）的位用'0'去'或'，欲置位的位用'1'去'或'；逻辑异或指令用来对字节中某些位取反，欲取反位用'1'去"异或"，欲保留位用'0'去"异或"。

当用逻辑指令对 P0、P1、P2 或 P3 进行操作时，是从它们的锁存器中读出数据，再将处理后的结果写回锁存器。

2.2.3 算术运算类指令

算术运算类指令主要完成加、减、乘、除四则运算，以及加 1、减 1 和二-十进制调整。四则运算指令会影响状态标志寄存器 PSW，而加 1、减 1 指令不会影响状态标志寄存器 PSW。

(1) 不带进位的加法指令

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

(2) 带进位加法指令

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

(3) 带借位减法指令

```
SUBB    A,    #data     ;(A)←(A)－#data－(C)
SUBB    A,    data      ;(A)←(A)--(data)－(C)
SUBB    A,    @Ri       ;(A)←(A)－((Ri))－(C)
SUBB    A,    Rn        ;(A)←(A)－(Rn)－(C)
```

加减指令会影响状态寄存器 PSW 中的进位位 CY、辅助进位位 AC、溢出位 OV 和奇偶位 P。OV 溢出标志取决于带符号数加减运算时的第 6、7 位中有一位产生进位（借位）而另一位不产生进位（借位），则使 OV 置 1，否则 OV 被清 0。OV＝1 表示两正数相加，和变成负数，或两负数相加，和变为正数的错误结果。

(4) 乘法指令

$$\text{MUL}\quad AB\ ;\quad \left.\begin{matrix}(A)\ 0\sim7\\(B)\ 8\sim15\end{matrix}\right\}\leftarrow (A)\times(B)$$

该指令把累加器 A 和寄存器 B 中的 8 位无符号整数相乘，16 位乘积的低字节存放在累加器 A 中，高字节存放在寄存器 B 中，如乘积大于 255 (FFH)，则使溢出标志位 OV 置 1，否则清 0。运算结果总使进位标志 CY 清 0。

(5) 除法指令

$$\text{DIV}\quad AB\quad ;\quad \left.\begin{matrix}(A)\ 商\\(B)\ 余数\end{matrix}\right\}\leftarrow (A)\ /\ (B)$$

该指令把累加器 A 中的 8 位无符号整数除以寄存器 B 中 8 位无符号整数，所得商存在累加器 A 中，余数存在寄存器 B 中，标志位 CY 和 OV 均清 0。若除数（B 中内容）为 00H，则执行后结果为不定值，并置位溢出标志 OV。在任何情况下，进位标志 CY 总清 0。

(6) 加 1 指令

```
INC     A         ;(A)←(A)＋1
INC     Direct    ;(direct)←(direct)＋1
INC     @Ri       ;((Ri))←((Ri))＋1
INC     Rn        ;(Rn)←(Rn)＋1
INC     DPTR      ;(DPTR)←(DPTR)＋1
```

(7) 减 1 指令

```
DEC     A         ;(A)←(A)－1
DEC     Direct    ;(direct)←(direct)－1
DEC     @Ri       ;((Ri))←((Ri))－1
DEC     Rn        ;(Rn)←(Rn)－1
```

加 1 和减 1 指令不影响 PSW 标志位。

(8) 二-十进制调整指令

DA　　A

本指令是对二-十进制的加法进行调整的指令。两个压缩型 BCD 码按二进制数相加，必须经过本条指令调整后才能得到压缩型的 BCD 码和数。由于指令要利用 AC、CY 等标志位才能起到正确的调整作用，因此它必须跟在加法（ADD、ADDC）指令后面才能使用。对用户而言，只要保证参加运算的两数为 BCD 码，并先对 BCD 码执行二进制加法运算（用 ADD、ADDC 指令），然后紧跟一条 DA　A 指令即可。使用时应注意：DA　A 指令不能对减法进行十进制调整。

指令的操作过程为：若相加后累加器低 4 位大于 9 或半进位位 AC=1，则低 4 位进行加 6 调整，即 A←A+06H；若累加器高 4 位大于 9 或进位位 CY=1，则高 4 位进行加 6 调整，即 A←A+60H；若两者同时发生或高 4 位虽等于 9 但低 4 位修正有进位，则应进行加 66 修正。

2.2.4　位操作指令

位操作指令是 MCS-51 单片机的显著特点，它是以位为单位进行运算和操作的。

（1）位数据传送指令

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

（2）位状态控制指令

```
CLR     bit         ; (bit) ←0
CLR     C           ; (C) ←0
SETB    bit         ; (bit) ←1
SETB    C           ; (C) ←1
```

这组指令是对位累加器 C 及位地址指定的位 bit 进行置位及清零，不影响其他标志。

（3）位逻辑操作指令

```
ANL     C, bit      ; (C) ← (C) ∧ (bit)
ANL     C, /bit     ; (C) ← (C) ∧ (bit)
ORL     C, bit      ; (C) ← (C) ∨ (bit)
ORL     C, /bit     ; (C) ← (C) ∨ (/bit)
CPL     bit         ; (bit) ←! (bit)
CPL     C           ; (C) ←! (C)
```

上述指令包括逻辑与、逻辑或和逻辑非三种。指令中的"/bit"表示对该寻址位内容取反后再进行运算，但是（bit）的内容并不改变。

（4）位条件转移指令

```
JC      rel         ;若(C)=1,则(PC)←(PC)+rel,否则顺序执行
```

```
JNC     rel          ;若(C)=0,则(PC)←(PC)+rel,否则顺序执行
JB      bit,rel      ;若(bit)=1,(PC)←(P)+rel,否则顺序执行
JNB     bit,rel      ;若(bit)=0,(PC)←(PC)+rel,否则顺序执行
JBC     bit,rel      ;若(bit)=1,(PC)←(PC)+rel,(bit)←0,
                     ;否则顺序执行
```

上述指令通过判断进位 C 或位地址的内容(bit)的状态来决定程序的走向。当条件满足时就转移,否则就顺序执行程序。

2.2.5 控制转移类指令

在程序设计中,有时要修改程序计数器 PC 的值才能满足设计要求,控制转移类指令即用来实现这一要求。控制转移指令包括无条件转移指令、条件转移指令及子程序调用、返回指令三种。

(1) 无条件转移指令

```
LJMP    addr16       ;(PC)←Addr0~15
AJMP    addr11       ;(PC)←(PC)+2,(PC0~10)←addr0~10,(PC11~15)不变
SJMP    rel          ;(PC)←(PC)+2,(PC)←(PC)+rel
JMP     @A+DPTR      ;(PC)←(A)+(DPTR)
```

上述四条指令都可使程序无条件地转移到指令所提供的地址上,它们的不同之处在于它们所提供的转移范围不同。

1) 长转移指令 LJMP:指令提供 16 位目标地址,所以执行这条指令可以使程序从当前地址转移到 64K 字节程序存储器地址空间的任何单元。

2) 短转移指令 AJMP:指令第二字节存放的是低 8 位地址,第一字节 5、6、7 位存放着高 3 位地址 a8~a10。指令执行时分别把高 3 位和低 8 位地址值取出送入程序计数器 PC 的低 11 位,维持 PC 的高 5 位((PC)+2 后的)地址值不变,可实现 2K 范围内的程序转移。

由于 AJMP 为双字节指令,当程序真正转移时 PC 值已加 2 了,因此转移的目标地址应与 AJMP 下相邻指令第一字节地址在同一 2K 字节范围内。AJMP 指令示意如图 2-2 所示。

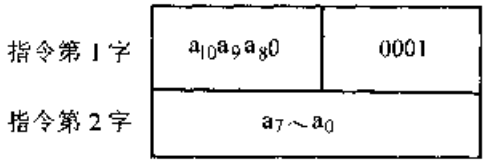


图 2-2 AJMP 指令示意图

3) 相对转移指令 SJMP:该指令地址由指令第二字节的相对地址和程序计数器 PC 的当前值(执行 SJMP 前的 PC 值加 2)相加形成。因而转向地址可以在这条指令首地址 −128 字节~+127 字节之间。

这条指令的突出优点是指令中只给出了相对转移地址 rel(8 位带符号的数),这样当程序修改时只要相对地址不发生变化,该指令就不需做任何改动。对于前两条指令(LJMP、AJMP),由于直接给出转移地址,在程序修改时就可能需要修

改该地址。

4）间接转移指令 JMP：这是一条极有用的多分支选择转移指令，其转移地址是在程序运行时动态决定的，这也是和前三条指令的主要区别。所以可在 DPTR 中装入多分支转移程序的首地址，而由累加器 A 的内容来动态选择其中的某一个分支予以转移，这就可用一条指令代替众多的转移指令，实现以 DPTR 内容为起始的 256 个字节范围的选择转移。

（2）条件转移指令

1）累加器判零转移指令：

```
JZ    rel        ；累加器为 0 转移
JNZ   rel        ；累加器不为 0 转移
```

JZ 表示若累加器全'0'，则转向指定的地址，否则顺序执行下条指令。JNZ 指令刚好相反，只需累加器"非零"，则转向指定地址，否则顺序执行下条指令。

2）比较转移指令：

格式：　　CJNE　〈目的字节〉，〈源字节〉，rel

CJNE 类指令比较两个操作数之大小，如果它们的值不相等则转移，相等则继续执行。这些指令均为三字节指令，PC 当前值((PC) +3→ (PC)) 与指令第三字节带符号的偏移量相加即得到转移地址。如果目的字节的无符号整数值小于源字节的无符号整数值，则置位进位标志，否则清零进位位，指令执行不影响任何一个操作数。

具体的比较转移指令有 4 条：

```
CJNE  A，#data，rel
CJNE  A，direct，rel
CJNE  @Ri，#data，rel
CJNE  Rn，#data，rel
```

3）循环转移指令：

格式：　　DJNZ〈字节〉，rel

这是一条减 1 与零比较指令，指令先对源操作数做减 1 操作，然后判断结果是否为 0，不为 0 则转移，否则顺序执行。一般可将循环次数放入某个工作寄存器 Rn 或直接地址单元中，利用该指令即可实现循环。

共有两条指令：

```
DJNZ  direct，rel
DJNZ  Rn，rel
```

（3）子程序调用、返回指令

1）长调用指令：

LCALL addr16 ；
- (PC)←(PC)+3
- (SP)←(SP)+1
- ((SP))←($PC_{0\sim7}$)
- (SP)←(SP)+1
- ((SP))←($PC_{8\sim15}$)

2）绝对调用指令：

ACALL
- (PC)←(PC)+2
- (SP)←(SP)+1
- ((SP))←($PC_{0\sim7}$)
- (SP)←(SP)+1
- ((SP))←($PC_{8\sim15}$)
- ($PC_{0\sim10}$)←$addr_{0\sim10}$
- ($PC_{11\sim15}$)不变

3）返回指令：

RET
- ($PC_{8\sim15}$)←((SP))
- (SP)←(SP)−1
- ($PC_{0\sim7}$)←((SP))
- (SP)←(SP)−1

RETI ；中断服务程序返回指令

4）空操作指令

NOP ；(PC)←(PC)+1

2.3 伪指令

伪指令是在机器汇编过程中，用来对汇编过程进行某种控制或对符号和标号进行赋值的指令。这些指令不属于指令系统中的指令，汇编时也不产生机器代码。常用的伪指令有：

(1) ORG (Origin)

一般形式： ORG 16位地址

ORG伪指令出现在程序块或数据块的开始，用以指明此语句后面的目标程序或数据块存放的起始地址。在一个源程序文件中，可以多次使用ORG，规定不同程序段的起始位置，但定义的地址顺序要从小到大，并不能重叠。

(2) DB (Define Byte)

一般形式： [标号:] DB 字节数据项表

数据项表从标号指定的地址连续存放，可为十进制数或十六进制数，也可以是由单引号括起来的一个字符串，每个字符串元素为一个ASCII码。各数据项用

逗号分隔。

(3) DW　(Define Word)

一般形式：　[标号:]　DW　双字节数据项表

DW 的功能与 DB 类似。通常 DB 用于定义数据表，DW 用于定义 16 位地址表。

(4) EQU　或＝ (Equal)

一般形式：　名字　EQU　表达式

或：　名字　＝　表达式

用于给一个表达式的值或一个字符串起一个名字。程序中，该名字可以用作程序地址、数据地址或立即数使用。表达式可以是 8 位或 16 位数值。名字必须是以字母打头的字母数字串，名字必须惟一。

如：
```
START  EQU  100H
PORT   EQU  2301H
ORG    START
MOV    DPTR, #PORT
```

(5) DATA

一般形式：　名字　DATA　直接字节地址

该伪指令给一个 8 位内部 RAM 单元起一个名字，相当于定义一个变量。一个单元可以有多个名字。

如：
```
ERR    DATA  32H
MOV    ERR, #23H
```

(6) END

END 伪指令指出源程序到此结束，汇编器对其后的程序语句不予处理。一个源文件只能使用一个 END 伪指令。

2.4　MCS—51 程序设计举例

2.4.1　简单程序设计

简单程序是程序结构中最简单、基本的一种程序结构，它按程序的书写顺序依次执行，程序中没有转移指令。

例 1　将 40H 单元的两个 BCD 码拆开并变成 ASCII 码，存入 41H、42H 单元 (0～9 的 ASCII 码为 30H～39H)。

可先将 40H 单元的 BCD 码除以 10H，A 和 B 中分别为 BCD 码的高 4 位 (商) 和低 4 位 (余数)，然后使 A 和 B 分别与 30H 相或，即可得到 ASCII 码。

程序如下：

```
ORG  4000H
```

```
MOV  A，40H          ；(40H) →A
MOV  B，#10H         ；10H→B
DIV  AB              ；拆开两个 BCD 数，高 4 位在 A 中，低 4 位在 B 中
ORL  A，#30H         ；高 4 位转换成 ASCII 码
MOV  42H，A          ；存结果
ORL  B，#30H         ；低 4 位转换成 ASCII 码
MOV  41H，B          ；存结果
END
```

2.4.2 分支程序设计

分支程序是根据条件的成立与否来决定执行哪个程序段。在程序中条件转移指令常用来实现分支，无条件转移指令用来控制每个分支的转移方向。

例 2 编制计算函数 Y＝f（X）的程序。设 X、Y 均为带符号数，其方程如下：

$$Y=\begin{cases}2 & \text{当 } X>0 \text{ 时}\\ 0 & \text{当 } X=0 \text{ 时}\\ -2 & \text{当 } X<0 \text{ 时}\end{cases}$$

因为 X 为带符号数，故不能用比较转移指令 CJNE。该程序有三个分支，要做两次判断，第 1 次用累加器判零指令判断 X 是否为零，第 2 次用位条件转移指令判断 X＞0 还是 X＜0。

```
        ORG    3000H
X       EQU    30H          ；设 X 存放在 30H
Y       EQU    31H          ；设 Y 存放在 31H
        MOV    A，X         ；(30H) →A
        JZ     LOOP2        ；若 X＝0，则转 LOOP2
        JNB    ACC.7，LOOP1 ；若 X＞0，则转 LOOP1
        MOV    A，#0FEH     ；若 X＜0，则 A＝－2
        LJMP   LOOP2        ；转 LOOP2
LOOP1：  MOV   A，#02H      ；若 X＞0，则 A＝2
LOOP2：  MOV   Y，A         ；存结果
        END
```

2.4.3 循环程序设计

循环程序将要重复执行的部分做为一个程序段，然后利用指令来实现循环，直到满足要求为止。

例 3 设 8051 单片机的时钟频率为 12MHz，要求设计一个软件延时程序，延时时间为 100ms。

出题可知单片机的机器周期为 1μs，可以设计一个延时 1ms 的循环程序作为内循环，共循环 100 次，就可以实现 100ms 的延时。

```
        ORG   4000H
        MOV   R6，#100        ；外循环 100 次
LOOP1：  MOV，R7，#200        ；内循环 1ms
LOOP2：  NOP
        NOP
        NOP
        DJNZ  R7，LOOP2
        DJNZ  R6，LOOP1
        END
```

内循环指令的机器周期数为：1＋1＋1＋2＝5，故内循环每循环一次的时间为：（1＋5×200）μs。

2.4.4 数据转换程序设计

在单片机应用系统中，数据的输入输出常采用十进制数，直观、方便；内部运算时，要使用二进制数，运算简便、存储量小。程序中，经常使用数制转换子程序。

例 4 单字节 BCD 码到二进制数的转换。

算法原理：BCD 码高位乘 10 加 BCD 码低位。

（1）入口参数：R2（BCD 码）

（2）出口参数：R2（8 位无符号二进制数）

（3）子程序：

```
BCD2B：  MOV   A，R2
        ANL   A，#0F0H        ；取 BCD 码高位
        SWAP  A
        MOV   B，#10
        MUL   AB
        MOV   R3，A           ；缓存
        MOV   A，R2
        ANL   A，#0FH
        ADDA，R3
        MOV   R2，A
        RET
```

例 5 双字节 BCD 码到二进制数的转换。

算法原理：双字节 BCD 码可以表示为

$(d_3d_2d_1d_0)_{BCD}=(d_3*10+d_2)*100+(d_1*10+d_0)$

这里，(d_1*10+d_0) 的运算可由子程序 BCD2B 完成。

1）入口参数：R5（千、百位）R4（十、个位）BCD 码

2）出口参数：R5R4（无符号二进制数）

3）子程序：

```
BCD4B:  MOV     A，R5
        MOV     R2，A
        ACALL   BCD2B
        MOV     A，R2
        MOV     B，#100
        MUL     AB
        MOV     R6，A           ；乘积低字节
        XCH     B，A
        MOV     R5，A           ；乘积高字节
        MOV     A，R4
        MOV     R2，A
        ACALL   BCD2B
        MOV     A，R2
        ADD     A，R6
        MOV     R4，A
        MOV     A，R5
        ADDC    A，#0
        MOV     R5，A
        RET
```

2.4.5 查表程序设计

查表是单片机应用程序设计时常用的算法，根据不同的表格结构，有多种查表算法。

例 6 设有一个巡回检测报警装置，需对 16 路输入节点进行检测。每路输入有一个最大允许值，为双字节整数。检测时，需根据测量的节点号，找出该节点的最大允许值，看测量值是否大于最大允许值，如大于就报警。

这个问题所进行的操作是，给出表格的项号（节点号），查出该项表项的值，即求解 $X_i=f(i)$ 的过程。这种结构的程序是将一系列函数值编成表格存放在 ROM 中，当需要时再按 i 查找 f（i）。

查表时，R2 中存放节点号，查到的该路最大值放在 R3R4 中。

```
LTB1:   MOV     A，R2
```

```
        ADD     A，R2           ；每个表项为两字节
        MOV     R3，A
        ADD     A，＃6          ；表首偏移量
        MOVC，  A，@A＋PC       ；查第一字节
        XCH     R3，A
        ADD     A，＃3
        MOVC    A，@A＋PC       ；查第二字节
        MOV     R4，A
        RET
TAB：   DW      1520，3721，42645，7850
        DW      3483，32657，883，9943
        DW      10000，49511，6758，8931
        DW      4468，5871，13284，27808
```

这里用 PC 作为基址，表的位置和表的长度都须合适。当表的长度较大时，必须用 DPTR 作基址。若例中有 1024 个表项，每个表项为两字节，考虑程序该如何修改。

例 7 有时表格中的一个表项包含有若干个数据值，例如每项有两个数据值 ai 和 bi。对于这种表格，一般的查表运算为求 $b_i=f(a_i)$，即给出 a_i，找到对应的 b_i。

设有一控制系统，输入一个 ASCII 字符，要按输入的字符转去执行对应的处理程序。命令字符为'A'、'D'、'E'、'L'、'X'、'Z'六种，对应的处理程序入口标号为 XA、XD、XE、XL、XX、XZ。

入口：命令字符在 A 中。查到时，转相应处理子程序，否则转查不到处理子程序。

与前一例相同，表的结束标志仍为-- 0 数值。

子程序：

```
LTB3：  MOV     DPTR，＃TABEL
        MOV     B，A            ；缓存输入字符
LTB3A： CLR     A
        MOVC    A，@A＋DPTR
        JZ      LTB3N           ；到表尾
        INC     DPTR            ；指向下一字节
        CJNE    A，B，LTB3B     ；不符
        CLR     A               ；找到
        MOVC    A，@A＋DPTR     ；取入口地址高字节
```

```
        MOV     B，A
        INC     DPTR
        CLR     A
        MOVC    A，@A+DPTR  ；取入口地址低字节
        MOV     DPL，A
        MOV     DPH，B
        CLR     A
        JMP     @A－DPTR    ；按（DPTR）跳转
LTB3B：  INCDPTR
        INC     DPTR        ；准备查下一表项
        SJMP    LTB3A
LTB3N：  AJMP    UNFOUND     ；转查不到处理子程序
TABEL：  DB      'A'
        DW      XA
        DB      'D'
        DW      XD
        DB      'E'
        DW      XE
        DB      'L'
        DW      XL
        DB      'X'
        DW      XX
        DB      'Z'
        DW      XZ
        DB      0
```

这里使用的方法实质上是一种散转程序设计方法，另一种较简捷的方法如下：

```
LTB4：   MOV     DPTR，#TABEL
        MOV     B，A
LTB4A：  CLR     A
        MOVC    A，@A+DPTR
        JZ      LTB4N
        INC     DPTR
        CJNE    A，B，LTB4B
        CLR     A
        JMP     @A+DPTR
```

```
LTB4B:  INCDPTR
        INC     DPTR
        SJMP    LTB4A
LTB4N:  AJMP    UNFOUND
TABEL:  DB      'A'
        AJMP    XZ
        DB      'D'
        AJMP    XD
        DB      'E'
        AJMP    XE
        DB      'L'
        AJMP    XL
        DB      'X'
        AJMP    XX
        DB      'Z'
        AJMP    XZ
        DB      0
```

第三章　单片机的C程序设计

8051系列单片机作为工业标准，从1985年开始就有8051单片机的C语言编译器，简称C51。

3.1　C51的数据与运算

3.1.1　C51的数据类型

C51的数据类型与一般C语言的数据类型大多相同，其基本数据类型有：

1. 字符型数据

标识符为char、unsigned char（无符号字符型）。字符型数据占用一个字节存储单元，存放的内容为－128～127或0～255（无符号型）的数据。8051系列单片机是8位机，在程序中大多使用字符型变量存放数据。

2. 整型数据

标识符为int、unsigned int（无符号整型）。整型数据占用两个字节存储单元，存放的内容为－32768～32767或0～65535（无符号数）的数据。

3. 长整型数据

标识符为long、unsigned long（无符号整型）。整型数据占用四个字节存储单元。

4. 实型数据

标识符为float（单精度）、double（双精度）。float型数据占用四个字节存储单元，double型数据占用八个字节存储单元。由于8051系列单片机没有硬件乘法器，其实型数据的运算非常慢，因而一般在程序中不做实型数据的运算。

在基本数据类型基础上，C51也有派生数据类型，如数组、结构体类型、共用体类型、枚举类型、指针类型等，这些数据类型在后面讲述。

除了一般的C语言数据类型外，为了智能仪器仪表和工业自动化系统的需要，C51增加了位变量数据类型。

一般位变量类型，标识符为bit，bit型变量只占用一位存储单元，位于内部RAM的可位寻址区。

对位于可位寻址的SFR空间和RAM空间的字节型变量，可以定义特殊位变量标识该变量的某一位，这种位数据类型的标识符为sbit。

bit和sbit类型的区别和用法将在后面讲述。

3.1.2　C51数据的存储类型与8051存储器结构

C51 定义的任何数据类型必须以一定的存储类型定位在 8051 的某一存储区中，否则便没有任何实际意义。

8051 系列单片机在物理上有三个存储空间：程序存储器空间、片内数据存储器空间、片外数据存储器空间。

程序存储器空间是只读的，只能存放固定不变的数据，如对热电偶采集数据的校正参数表格、LED 显示的字符编码数据等。

片外数据存储器空间只能用 R0、R1 和 DPTR 寄存器的寄存器间接寻址方式，用指令 MOVX 进行数据读写操作。

片内数据存储器空间分为四个区域：四组通用工作寄存器、可位寻址区、堆栈区和用户数据区。对 8×51 系列单片机，这四个区域都处于低 128 字节 RAM 区内。用户数据区可用两种寻址方式进行访问（直接、寄存器间接）。对 8×52 系列单片机，用户数据区还包括高 128 字节的 RAM 区域，但该区域只能用寄存器间接寻址方式进行访问，以与对 SFR 的直接访问相区别。

C51 在定义变量和常量时，需说明它们的存储类型，将它们定位在不同的存储区中。

C51 对单片机的不同存储区域定义了不同的存储类型，它们的关系如表 3-1 所示。

表 3-1　存储类型与对应的存储区域

存储类型	对应的存储区域
data	直接寻址片内 RAM（128 字节）
bdata	可位寻址的片内 RAM（16 字节），允许位与字节混合访问
idata	间接寻址片内 RAM，可访问全部片内 RAM（256 字节）
pdata	分页寻址片外 RAM（256 字节）
xdata	片外 RAM（64K 字节）
code	程序存储区（64K 字节）

如：

```
#define uchar unsigned char   /* 定义符号常量 uchar */
uchar data a1;     /* 字符变量 a1 定位在 8051 的片内数据存储区中 */
bit bdata flag;    /* 位变量 flag 定位在 8051 的片内数据存储区中的可位寻
                      址区 */
float idata x, y;  /* 浮点变量 x，y 定位在 8051 的片内数据存储区并只能
                      通过间接寻址来访问 */
uchar xdata s [] = {3, 4, 7, 2, 12, 8};   /* 无符号字符数组 s 定位在
                                             片外 RAM */
uchar code table [10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
```

```
                    0x7f,0x6f};   /* 无符号字符数组 table 定位在片外
                    ROM */
```

当没有指定存储类型时，由编译系统的存储模式将其存于缺省（默认）存储空间。当编译系统的存储模式为小模式（small）时，缺省存储类型为 data，当编译系统的存储模式为其他模式时，缺省存储类型为 xdata。C51 编译器的存储模式一般为小模式。

3.1.3 8051 特殊功能寄存器及其 C51 定义

8051 单片机片内有 21 个特殊功能寄存器（SFR），它们分布在片内 RAM 的高 128 字节中，对特殊功能寄存器只能用直接寻址方式访问。特殊功能寄存器中还有 11 个可位寻址的寄存器。

在 C51 中，特殊功能寄存器及其可位寻址的位是通过关键字 sfr 和 sbit 来定义的，这种定义方法与标准 C 不兼容，只适用于 C51。

如：

```
sfr PSW=0xD0;
sfr TMOD=0x89; /* 定义定时/计数器方式控制寄存器 TMOD（其地址为
                  89H) */
sfr P1=0x90;   /* 定义 I/O 口 P1(其地址为 90H) */
```

PSW 是可位寻址的 SFR，其中各位的定义用 sbit。

如：

```
sbit CY=0xD7;   /* 定义位 CY(其位地址为 D7H) */
sbit AC=0xD0^6;   /* 定义位 AC(其位地址为 D6H) */
sfr PSW=0xD0;   /* 定义程序状态字寄存器 PSW(其地址为 D0H) */
sbit RS0=PSW^3;   /* 定义位 RS0(其位地址为 D3H) */
```

注意：sfr 和 sbit 只能在函数外使用，一般放在程序的开头。

实际上大部分特殊功能寄存器及其可位寻址的位的定义在 reg51.h、reg52.h 等相应的头文件中已给出，使用时只需在源文件中包含相应的头文件，即可使用 SFR 及其可寻址的位；而对于未定义的位，使用之前必须先定义。

如：

```
#include "reg51.h"
sbit  P10=P1^0;
sbit  P12=P1^2;
main ()
{
  P10=1; P12=0;
  PSW=0x08;          /* 等价的定义 RS0=1; RS1=0; */
```

```
    if (OV==1) ……;
     ⋮
}
```

3.1.4 8051 并行接口及其 C51 定义

8051 的 I/O 接口有 P0、P1、P2 和 P3 四个，除此之外 8051 还可以在片外扩展硬件 I/O 口和其他功能芯片，它们与外部数据存储器是统一编址的，即 8051 把它们当作外部数据存储器的一个单元。

P0、P1、P2 和 P3 的定义在头文件 reg51.h 和 reg52.h 中，扩展的外部 RAM 单元、外部硬件 I/O 口和功能接口芯片需用户自定义。

如：

```
#include  "absacc.h"
#define  PA  XBYTE [0xffec]
main ()
{
  PA=0x3A; /*向外部数据存储器的地址为 ffecH 的单元中写入 3AH */
}
```

以上程序中用 C 中的编译预处理命令 #define 将 PA 定义为外部 I/O 口，地址为 0xffec，是单字节量。其中 XBYTE 是一个指针，指向外部数据存储器的零地址单元，它是在头文件 absacc.h 中定义的，详见后面关于 C51 的指针类型部分。

3.1.5 位变量及其 C51 定义

8051 系列单片机具有位运算器，C51 相应地设置了 bit 数据类型。

1. 位变量的定义

位变量用关键字"bit"来定义，它的值是一个二进制位。

```
如：bit  lock_pt;              /*将 lock_pt 定义为位变量*/
    bit  direction_bit;        /*将 lock_pt 定义为位变量*/
```

2. 函数可以有 bit 类型的参数，也可以有 bit 类型的返回值

```
bit func (bit  b0, bit b1)
{
  bit, a;
  …
  return  a;
}
```

使用禁止中断宏命令 # progma disnable 或指定明确的寄存器切换 (using n) 的函数不能返回位值。

3. 对位变量定义的限制

不能定义位变量指针，如：bit * bit point；

不能定义位数组，如：bit bit array [5]；

位变量说明中可以指定存储类型，位变量的存储类型只能是 bdata。

在程序设计时，对于可位寻址对象，即可以字节寻址又可以位寻址的变量，则其存储类型只能是 bdata。

使用时，先说明字节变量的数据类型和存储类型：

如：

```
int bdata a;              /* 整型变量 a 定位在片内数据存储区中的可位寻址区 */
char bdata b [4];         /* 字符数组 b 定位在片内数据存储区中的可位寻址区 */
```

然后，使用 sbit 关键字定义其中可独立寻址访问的位变量：

```
sbit a0=a^0;              /* 定义 a0 为 a 的第 0 位 */
sbit a12=a^12;            /* 定义 a12 为 a 的第 12 位 */
sbit b03=b [0]^3;         /* 定义 b03 为 b [0] 的第 3 位 */
sbit b36=b [3]^6;         /* 定义 b36 为 b [3] 的第 6 位 */
```

sbit 定义要求基址对象的存储类型为 bdata。

使用 sbit 类型位变量时，基址变量和其对应的位变量的说明必须在函数外部进行。

3.1.6 C51 运算符、表达式及其规则

C51 的运算符主要有：算术运算符、关系运算符、逻辑运算符、位运算符、赋值及复合赋值运算符等。

1. 算术运算符和算术表达式

(1) 基本的算术运算符

＋ （加法运算符）

－ （减法运算符）

* （乘法运算符）

/ （除法运算符）

% （模运算符或取余运算符）

这五个运算符都是双目运算符，即运算需两个操作数。

对于除法运算符：若两个整数相除，结果为整数（即取整）。

对于取余运算符：要求%两侧的操作数均为整型数据，所得结果的符号与左侧操作数的符号相同。

如：7/5=1，5/7=0，－93%23=－1，－93%－23=－1，93%－23=1

(2) 增、自减运算符（都是单目运算符）

＋＋为自增运算符，－－为自减运算符。

如：＋＋j、j＋＋、－－i、i－－

说明：①＋＋和－－运算符只能用于变量，不能用于常量和表达式。

②当＋＋j时，是先将变量的值加 1（j=j+1），再取变量值；

当 j＋＋时，是先取变量值，再把变量的值加 1（j=j+1）。

如：若变量 a=2；则分别执行 b=＋＋a 与 b=a＋＋后变量 b 的值是不同的。前者 b=3，后者 b=2（a 的值都等于 2）。

（3）算术表达式和运算符的优先级与结合性

算术表达式：用算术运算符和括号将操作数（常量、变量、函数等）连接起来的式子。如：a＊b/c－1.5＋'d'

C51 规定了运算符的优先级和结合性。当一个操作数的两侧都有运算符时，哪一个运算符先执行，哪一个运算符后执行，是由两个运算符的优先级和结合性决定的。一般先由优先级的高低来定运算顺序，当优先级相同时，再看结合性。（表 3-2 列出了所有运算符的优先级别和结合性）

表 3-2　运算符的优先级别和结合性

优先级	运算符	含　义	操作数个数	结合性
1	（　）	圆括号		从左至右
	[　]	下标运算符		
	．与	结构体成员运算符		
2	！	逻辑非运算符	1（单目运算符）	从右至左
	～	按位取反运算符		
	＋＋	自增运算符		
	－－	自减运算符		
	（类型）	类型转换运算符		
	＊	指针运算符		
	&	取地址运算符		
	sizeof	长度运算符		
3	＊	乘法运算符	2（双目运算符）	从左至右
	/	除法运算符		
	%	取余运算符		
4	＋	加法运算符	2（双目运算符）	从左至右
	－	减法运算符		
5	≪	左移运算符	2（双目运算符）	从左至右
	≫	右移运算符		
6	＜　＜＝　＞　＞＝	关系运算符	2(双目运算符)	从左至右
7	－－	等于运算符	2（双目运算符）	从左至右
	！－	不等于运算符		

（续）

优先级	运算符	含　义	操作数个数	结合性
8	&	按位与运算符	2(双目运算符)	从左至右
9	^	按位异或运算符	2(双目运算符)	从左至右
10	|	按位或运算符	2(双目运算符)	从左至右
11	&&	逻辑与运算符	2(双目运算符)	从左至右
12	||	逻辑或运算符	2(双目运算符)	从左至右
13	? :	条件运算符	3(三目运算符)	从右至左
14	= += -= *= /= %= >>= <<= &= ^= |=	赋值及复合赋值运算符	2（双目运算符）	从右至左
15	,	逗号运算符		从左至右

如：a－b＊c 等价于 a－(b＊c)

如：a＊b/c 等价于(a＊b)/c

(4) 强制类型转换运算符

可以利用强制类型转换运算符强行将一个表达式转换成所需类型。

其一般形式为：　（类型名）（表达式）

如：(double)a　　将 a 强制转换成 double 类型

(int)(x＋y)　　将 x＋y 强制转换成 int 类型

(int)a%(int)b　　将 a 与 b 强制转换成 int 类型后，做取余运算

说明：表达式必须用括号括起

如：(double) x＋y＝ (double) (x) ＋y≠ (double) (x＋y)

2. 关系运算符和关系表达式

(1) 关系运算符及其优先级（关系运算符都是双目运算符）

关系运算即比较运算，就是将两个数值进行比较。C51 提供了 6 种关系运算符：

＜　（小于）

＜＝　（小于等于）

＞　（大于）

＞＝　（大于等于）

＝＝　（等于）

！＝　（不等于）

优先级：①前四个运算符的优先级相同，后两个运算符的优先级相同。

②关系运算符的优先级低于算术运算符的优先级，高于赋值运算符

的优先级。

(2) 关系表达式

用关系运算符将两个表达式连接起来的表达式称为关系表达式。

如：a>b，a+b>=c+d，(a=3) < (b=2)

关系表达式的值为逻辑值：假和真。C51 中用 0 表示假，用 1 表示真。

如有关系表达式 a>=b，若 a 的值是 4，b 的值是 3，则给定的关系满足，所以关系表达式的值为 1，即逻辑真；若 a 的值是 2，则给定的关系不满足，故关系不等式的值为 0，即逻辑假。

3. 逻辑运算符和逻辑表达式

(1) 逻辑运算符及其优先级

逻辑运算是对逻辑量进行的运算。C51 提供三种逻辑运算符：

&&　　(逻辑与)

||　　(逻辑或)

!　　(逻辑非)

&& 和||是双目运算符，　! 是单目运算符。

! 的优先级高于 &&，&& 的优先级又高于||。逻辑运算符与其他运算符的优先级参见前面表 3-2。

逻辑运算的真值表见表 3-3。

表 3-3　逻辑运算的真值表

a	b	! a	! b	a&&b	a |b
真 (1)	真 (1)	假 (0)	假 (0)	真 (1)	真 (1)
真 (1)	假 (0)	假 (0)	真 (1)	假 (0)	真 (1)
假 (0)	真 (1)	真 (1)	假 (0)	假 (0)	真 (1)
假 (0)	假 (0)	真 (1)	真 (1)	假 (0)	假 (0)

(2) 逻辑表达式

用逻辑运算符将两个表达式或逻辑量连接起来构成的表达式。表达式可以是算术表达式、关系表达式或逻辑表达式。逻辑表达式的值也是逻辑量（真或假）。

对于算术表达式，其值若为 0，则认为是逻辑假；若不为 0，即非 0，则认为是逻辑真。

如：有逻辑表达式 a&&b，若 a 的值是 4，b 的值是 0，因为 a 的值不为 0，所以为真；而 b 的值为 0，所以为假，真和假做逻辑与运算，结果为假，即该逻辑表达式的值为 0。

逻辑表达式的执行规则：逻辑表达式是不完全执行的，即一个逻辑表达式中的所有运算符并不都被执行，只有当一定要执行下一个逻辑运算符才能确定表达

式的值时，才执行该运算符。

如： a&&b&&c

若 a 的值为 0，则不需判断 b 和 c 的值就可确定表达式的值为 0。

而： a||b||c

若 a 的值是 0，则还需判断 b 的值，若 b 的值为 1，则不需判断 c 的值就可确定表达式的值为 1。

4. 位运算符及其表达式

C 语言提供六种位运算符：

按位与 &	按位或|	按位异或^
按位取反～	左移≪	右移≫

注：位运算的操作对象是整型和字符型数据，不能是实型数据。

(1) 按位与运算符 &

参加运算的两个量，如果两个相应的位都为 1，则该位的结果值为 1，否则为 0。

即： 1&0=0；0&1=0；0&0=0；1&1=1。

如：char a=3，b=6；

```
        0 0 0 0 0 0 1 1   (a)
&       0 0 0 0 0 1 1 0   (b)
--------------------------
        0 0 0 0 0 0 1 0
```

则结果：a&b 的值为 2

按位与的作用：

1) 清零：让要清零的数与 0 按位与即可。

2) 一个数的某些位，而将其余的位清零。

如：int a； 如只想要保留 a 的低字节，高字节清零。

处理：将 a 与 0x00ff 按位与即可。 (a=0x6539，b=0x00ff)

```
        0 1 1 0  0 1 0 1    0 0 1 1  1 0 0 1   (a)
&       0 0 0 0  0 0 0 0    1 1 1 1  1 1 1 1   (b)
-----------------------------------------------
        0 0 0 0  0 0 0 0    0 0 1 1  1 0 0 1   (c)
```

结果：c=a&b (0x0039)，c 只取 a 的低字节，高字节为 0

如果只想取两个字节中的高字节，取 b=0xff00，c=a&b 即可。

(2) 按位或运算符|

参加运算的两个量，两个相应的位中只要有一个为 1，则该位的结果值为 1，否则为 0。

即：0|0=0；0|1=1；1|0=1；1|1=1。

如：060|017 (八进制)

```
  0  0  1  1  0  0  0  0
| 0  0  0  0  1  1  1  1
  0  0  1  1  1  1  1  1
```

按位与的作用是将不需要的位清零，按位或的作用是将指定的位置1。

(3) 异或运算符^

异或运算又称XOR运算，它的运算规则是：参加运算的两个量的相应位若相同，则该位的结果为0；否则结果为1。

即：0^0=0；1^1=0；0^1=1；1^0=1。

异或运算的性质：

1) 与1异或，使特定位翻转：任何数与1异或都会变成相反数，利用这一点，就可以把一个量的指定位翻转而不影响其他位的值。

2) 与0异或，使指定位保留原值：任何数与0异或都保持不变，利用这一点，就可以把一个量的指定位保留原值。

(4) 位取反运算符～

取反运算符是单目运算符。它的运算规则是：将操作数的各位分别翻转为相反值。

即：～1=0；　　～0=1

如：若unsigned char a=0x9a，b；则经过b=～a后，a的值不变，b=0x65。

a：1 0 0 1 1 0 1 0

b：0 1 1 0 0 1 0 1

(5) 位左移运算符≪

将一个数的各位顺序左移若干位（左移运算中，高位移出舍弃不用，低位补零。）

如：int　a=15；（即00001111）　a=a≪1；

将变量a的各位左移1位，左移1位后a=30。（相当于乘2）

(6) 位右移运算符≫

将一个数的各位顺序右移若干位。

如：int a=15；a=a≫2；

将变量a的各位右移2位，右移2位后a=3。（相当于除2^2）

注：右移运算中，低位移出舍弃不用，对无符号数：高位补0；对有符号数：高位补符号位（即保持原数的符号不变）。

5. 赋值运算符和赋值表达式

(1) 赋值运算符

赋值运算符就是赋值符号"="，赋值运算符的优先级较低，结合性是右结合性。

(2) 赋值表达式

将一个变量与表达式用赋值号连接起来就构成赋值表达式。

1) 一般形式：　　变量名=表达式

赋值表达式中的表达式包括变量、数学表达式、关系表达式、逻辑表达式等，甚至可以是另一个赋值表达式。

2) 求解过程：将"="右边表达式的值赋给"="左边的一个变量，赋值表达式的值就是被赋值变量的值。

如：a=b=5 等价于 a= (b=5)，该表达式的值为 5

如：a= (b=4) + (c=6)，　　该表达式的值为 10

(3) 赋值的类型转换规则

在赋值运算中，当"="两侧类型不一致时，系统自动将右边表达式的值转换成左侧变量的类型，再赋给该变量。转换的规则如下：

1) 实型数据赋给整型变量时，舍弃实数的小数部分。

2) 整型数据赋给实型变量时，数值不变，但以浮点数形式存储在变量中。

3) 长字节整型数据赋给短字节整型变量时，实行截断处理。如将 long 型数据赋给 int 型变量时，将 long 型数据的低两字节数据赋给 int 型变量，而将 long 型数据的高两字节的数据丢弃。

4) 短字节整型数据赋给长字节整型变量时，进行符号扩展。如将 int 型数据赋给 long 型变量时，将 int 型数据赋给 long 型变量的低两字节，而将 long 型变量的高两字节的每一位都设为 int 型数据的符号值。

6. 复合赋值运算符

赋值号前加上其他运算符构成复合赋值运算符。

C51 提供了下列十个复合赋值运算符：

+=，-=，*=，/=，%=，&=，|=，^=，≪=，≫=。

如：a+=b　　等价于 a= (a+b)

如：x*=a|b　　等价于 x= (x* (a+b))

如：a&=b　　等价于 a= (a&b)

如：a≪=4　　等价于 a= (a≪4)

例 1　P1.0 接一开关，P1.1 接一发光二极管，如图 3-1 所示。开关打开时，二极管不亮，开关闭合时，二极管亮。程序如下：

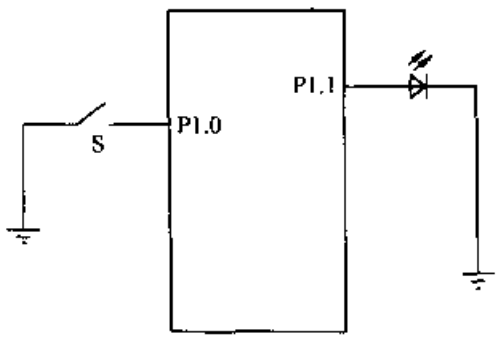


图 3-1　例 1 附图

```
#include<reg51.h>
sbit  p10=P1^0;
sbit  p11=P1^1;
```

```
void  main (  )
{
  p11=! p10;
}
```

3.2 C51 流程控制语句

3.2.1 C 程序的基本结构及流程图

C 语言是一种结构化的程序设计语言，任何程序都可用三种基本结构即顺序结构、选择结构、循环结构来表示。

1. 顺序结构

顺序结构是一种最简单的编程结构，这种结构的程序流程是按语句的顺序依次执行的。如图 3-2 所示，程序先执行 A 操作，然后执行 B 操作，两者是顺序执行的关系。

2. 选择结构

选择结构是根据给定的条件进行判断，由判断的结果决定执行两支或多支程序段中的一支。如图 3-3 所示，根据给定的条件 P 选择执行 A 或 B，若 P 为真则执行 A 操作，若 P 为假则执行 B 操作。这是具有两个分支的选择结构。

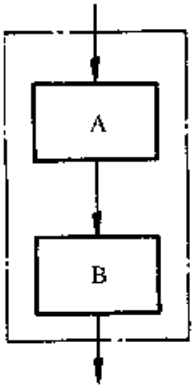


图 3-2　顺序结构示意图

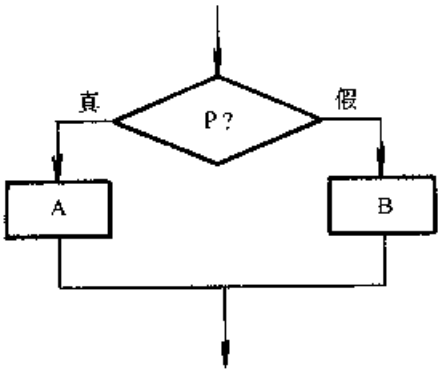


图 3-3　选择结构示意图

3. 循环结构

循环结构一般是在给定的条件为真时，反复执行某个程序段。循环结构有当型和直到型两类循环结构。

(1) 当型循环结构

如图 3-4a 所示，当给定的条件 P 为真时，重复执行操作 A，直到条件为假时，则退出循环。

(2) 直到型循环结构

如图 3-4b 所示，先执行 A 操作，再判断给定的条件 P，若 P 为真时，重复执行操作 A，直到条件为假时，则退出循环。

C 语言是一种很好的结构化程序设计语言，它提供了实现结构化程序所需的丰富的流程控制语句。它们分类如下：

流程控制语句
- 选择
  - if～else（条件语句）
  - switch～case（多分支选择语句）
  - goto（转向语句）
- 循环
  - while
  - do～while
  - for

辅助控制语句
- break
- continue
- return

下面讨论除 return 语句之外的所有流程控制语句的功能，return 语句将在函数中介绍。

3.2.2 选择语句

选择语句即条件判断控制语句，它首先判断给定的条件是否满足，然后根据判断的结果决定执行给出的若干种操作之一。C51 中的选择语句有 if 语句、switch～case 语句。

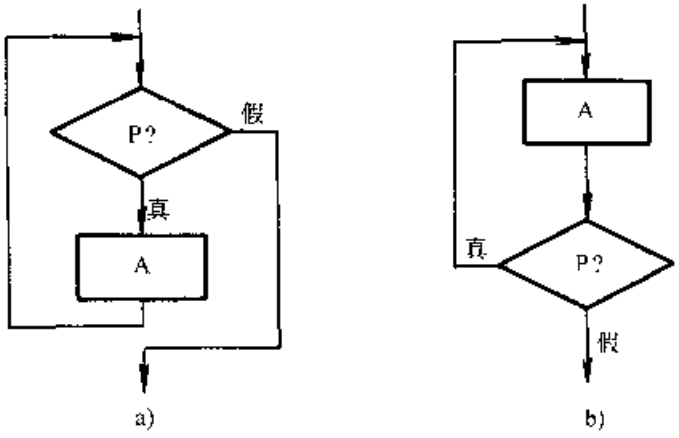


图 3-4

a）当型循环结构 b）直到型循环结构

1. if 语句

C 语句有两种类型的 if 语句。

（1）Ⅰ型 if 语句

一般形式：

```
if（表达式）
    S
```

其中〈表达式〉可以是符合 C 语法规则的任一表达式，如：算术表达式、关系表达式、逻辑表达式等；将 S 称为 if 语句的内嵌语句，S 一般是单条语句或是一个复合语句。执行流程如图 3-5 所示。

（2）Ⅱ型 if 语句

一般形式：

```
if（表达式）
    S1
else
```

S2

其中〈表达式〉可以是符合 C 语法规则的任一表达式。将 S1、S2 称为 if 语句的内嵌语句，它们可以是简单语句或是复合语句。

执行流程如图 3-6 所示。

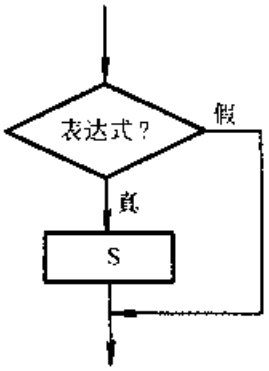


图 3-5　Ⅰ型 if 语句执行流程图

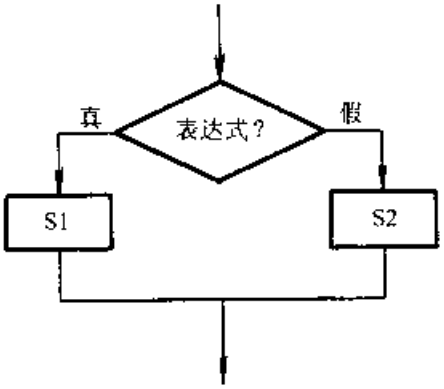


图 3-6　Ⅱ型 if 语句执行流程图

例 2　将例 1 用Ⅱ型 if 语句来改写。

程序如下：

```
#include (reg51. h)
sbit p10=P1^0;
sbit p11=P1^1;
void main (  )
{
  if (p10==0)    p11=1;
  else p11=0;
}
```

2. 条件运算符

经常用到一种简单的 if 语句，即

```
if (表达式) max=a;
else max=b;
```

它的特点是：语句 1 和语句 2 都是赋值语句，且都是给同一个变量赋值。

此时，可以用条件运算符来处理：

```
max= (表达式)? a : b;
```

条件运算符（?:）是唯一的一个三目运算符，即需要三个操作数。它的优先级是 13，是右结合性。

用条件运算符可以构成条件表达式，其一般形式是：

表达式 1? 表达式 2：表达式 3

它的执行过程：先计算表达式 1 的值，若非 0，就计算表达式 2 的值并作为条件表达式的值；否则，就计算表达式 3 的值并作为条件表达式的值。

例：max=a>b? a : b;

max=a>b? a : c>d? c : d;

例 3 将例 1 用条件运算符来改写。

程序如下：

```
#include <reg51. h>
sbit p10=P1^0;
sbit p11=P1^1;
void main ( )
{
  p11= (p10==0)? 1 : 0;
}
```

3. if 语句的嵌套

当 if 语句的内嵌语句是一个或多个 if 语句时，称为 if 语句的嵌套。if 语句的嵌套是有一定的条件的。if 语句嵌套的规则实际是说一个 else 与其前面的哪一个 if 配对的规则。C 规定 else 只能与其前面还未与其他 else 配对的 if 组成一个Ⅱ型 if 语句。

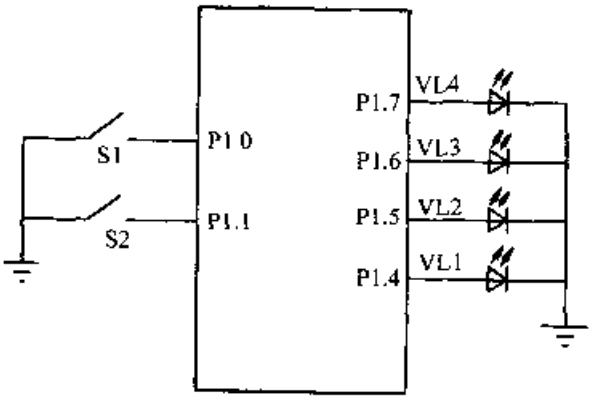


图 3-7 例 4 附图

例 4 P1 口的 P1.0 和 P1.1 各接一开关 S1、S2，P1.4、P1.5、P1.6 和 P1.7 各接一发光二极管，如图 3-7 所示。由 S1 和 S2 的不同状态，确定哪个发光二极管亮，真值表如下表所示：

S1	S2	亮的管子
1	1	VL1
0	1	VL2
1	0	VL3
0	0	VL4

程序如下：

```
#include <reg51. h>
void main ( )
{
```

```
char a;
a=P1;
a=a&0x03;              /*屏蔽高6位*/
if (a==3)    P1=0x13;
  else if (a==2)    P1=0x23;
        else if (a==1)    P1=0x43;
              else P1=0x83;
```

该例就是if语句嵌套的一种形式，其中每一个Ⅱ型if语句的S2语句是另一个Ⅱ型if语句。

4. switch语句

switch语句是多分支选择语句，它的一般形式是：

```
switch（表达式）
      { case 常量表达式1：语句1 break;
        case 常量表达式2：语句2 break;
              ⋮
        case 常量表达式n：语句n break;
        default          ：语句n+1
}
```

说明：

1）表达式可以是任意类型的。

2）常量表达式的类型要与表达式的类型相同。

3）各常量表达式的值必须互不相同。

4）各个case的出现次序可以任意。

5）switch语句的执行过程：先计算表达式的值，当它与某个常量表达式的值相等时，就执行此case后面的语句，然后执行break语句，退出switch语句，继续执行后面的语句；若表达式的值不与任何的常量表达式的值相等，就执行default后面的语句，然后退出switch语句，继续执行后面的语句。

6）若没有break语句，则当与表达式相等的常量表达式后面的语句执行之后，会继续执行后面case中的语句，以及default后的语句。

7）这里的break语句是中断语句，它中断switch语句的执行。

例5 将例4用switch语句改写。

程序如下：

```
#include〈reg51. h〉
void main（  ）
{
```

```
    char a;
    a=P1;
    a=a&0x03;
    switch (a) {
      case3: P1=0x13; break;
      case2: P1=0x23; break;
      case1: P1=0x43; break;
      case0: P1=0x83;
    }
}
```

3.2.3 循环语句

C 语句中有四种循环实现方法：

1）用 if 语句和 goto 语句。

2）用 while 语句。

3）用 do-while 语句。

4）用 for 语句。

循环的种类：当型循环和直到型循环。

1. if 语句和 goto 语句

goto 语句是无条件转向语句，它的一般形式是：

```
  goto 语句标号;
```

语句标号是一个标识符。

C 程序中的任何一个语句都可以有一个语句标号，其一般形式是：

```
    语句标号：语句
```

goto 语句的执行：无条件地转到语句标号后面的语句处执行。

用 goto 语句可以与 if 语句一起构成循环结构。

（1）构成当型循环

```
loop：if（表达式）
        {语句
          goto loop;
        }
```

执行过程见图 3-8。

（2）构成直到型循环

```
loop：{语句
        if（表达式）goto loop;
      }
```

执行过程见图 3-9。

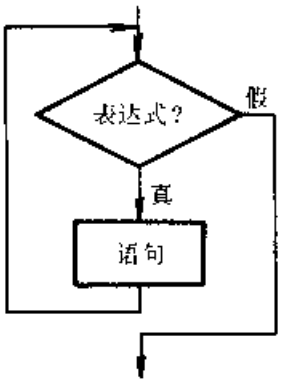


图 3-8　构成当型循环执行流程图

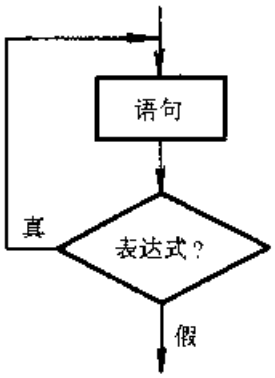


图 3-9　构成直到型循环执行流程图

前面例 4 的程序只能执行一次，我们需要的是在程序执行过程中可以随时改变开关的状态，进而改变发光二极管的状态。这就要程序不能停止，即用到循环结构。程序见例 6。

例 6

```
#include <reg51. h>
void main (  )
 {
char a;
loop: a=P1;
a=a&0x03;
switch (a)
   {case3: P1=0x13; break;
    case2: P1=0x23; break;
    case1: P1=0x43; break;
    case0: P1=0x83;
  }
goto loop;
}
```

该例用 goto 语句构成死循环，单片机的程序基本是这种结构。程序一直处理主循环中的语句。

2. while 语句

while 语句用来实现当型循环。其一般格式是：

while（表达式）　语句

表达式可以是任何表达式，语句可以是复合语句。

While 语句的执行过程：

1）计算表达式的值；

2）若其值为非 0，则执行内嵌语句，转 1；若表达式的值为 0，则退出 while 循环，执行 while 下面的语句（见图 3-8）。

例 7 求 $\sum_{n=1}^{20} n!$（即求 1! +2! +…+20!）

```
main (  )
{
  double f=1, sum=0; int n=1;
  while (n (=20)
  {
      f * =n;
      sum+ =f;
      n++;
  }
}
```

例 8 将例 6 用 while 语句改写。

```
#include (reg51. h)
void main (  )
{
char a;
while (1)
  {
    a=P1;
    a=a&0x03;
    switch (a)
       {case3: P1=0x13; break;
        case2: P1=0x23; break;
        case1: P1=0x43; break;
        case0: P1=0x83;
        }
  }
}
```

3. do-while 语句

do-while 语句实现直到型循环结构。其一般形式是：

do 语句 while（表达式）；

其特点是：先执行语句，后判断表达式。执行过程：

1）执行内嵌的语句；

2）计算表达式，当表达式的值为非 0（真）时，转 1；当表达式的值为 0（假）时，结束循环，执行 do-while 语句下面的语句（见图 3-9）。

例 9 求 $\sum_{n=1}^{20} n!$（即求 1! +2! +…+20!）

```
main ( )
{
  double f=1, sum=0; int n=1;
  do {
      f*=n;
      sum+= f;
      n++;
      } while (n<=20);
}
```

例 10 将例 6 用 do-while 语句改写。

```
#include (reg51. h)
void main ( )
{
char a;
do {
    a=P1;
    a=a&0x03;
    switch (a)
       {case3: P1=0x13; break;
        case2: P1=0x23; break;
        case1: P1=0x43; break;
        case0: P1=0x83;
        }
  } while (1);
}
```

4. for 语句

一般形式：

for（表达式 1；表达式 2；表达式 3）语句

它的执行过程：

1）求解表达式 1；

2）求解表达式 2，若其值非 0（真），则执行内嵌语句，转 3；若其值为 0（假），转 4；

3）求解表达式 3，转 2；

4）结束循环，执行 for 语句下面的语句（见图 3-10）。

for 语句最简单的应用形式是：

for（循环变量赋初值；循环条件；循环变量改变）语句

如：for（i=1；i<=100；i++）sum+=i；

它的等价形式：

```
i=1;
while (i<=100)
 {
  sum+=i;
  i++;
 }
```

循环变量是决定循环条件的变量。

用 while 语句表示 for 语句的一般形式：

```
表达式 1;
while（表达式 2）
 {
  语句
  表达式 3;
 }
```

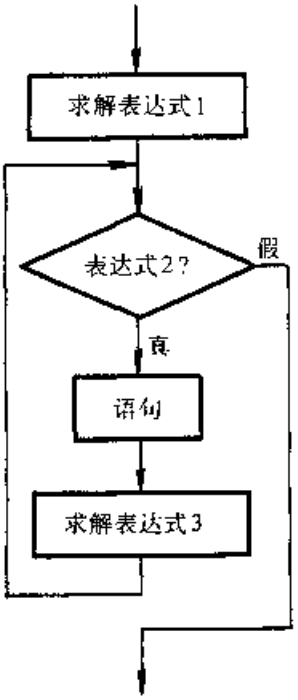


图 3-10　for 语句执行流程图

例 11　求 $\sum_{n=1}^{20} n!$（即求 1！+2！+…+20！）

```
main (  )
  {
   double f=1, sum=0; int n;
   for (n=1; n<=20; n++)
      {f*=n;
       sum+=f;
      }
  }
```

for 语句中，可以没有表达式 1、表达式 2 或表达式 3；若三个表达式都没有，则相当于一个无限循环：while（1）语句。

例 12　将例 6 用 for 语句改写。

```
#include <reg51. h>
void main ( )
{
char a;
for (;  ;)
   {
    a=P1;
    a=a&0x03;
    switch (a)
       {case3: P1=0x13; break;
        case2: P1=0x23; break;
        case1: P1=0x43; break;
        csae0: P1=0x83;
        }

  }
}
```

5. 循环的嵌套

一个循环体内又包含另一个循环结构，称为循环的嵌套。内嵌的循环中还可以嵌套循环，形成多层循环嵌套。

三种形式的循环（while 循环、do-while 循环、for 循环）可以互相嵌套。

例 13 P1 口接八个发光二极管，分别为 VL0、VL1…VL7，如图 3-11 所示。程序控制点亮 VL0，延迟一段时间，再点亮 VL1，…，VL7，然后又是 VL0。同时只能有一个灯亮。

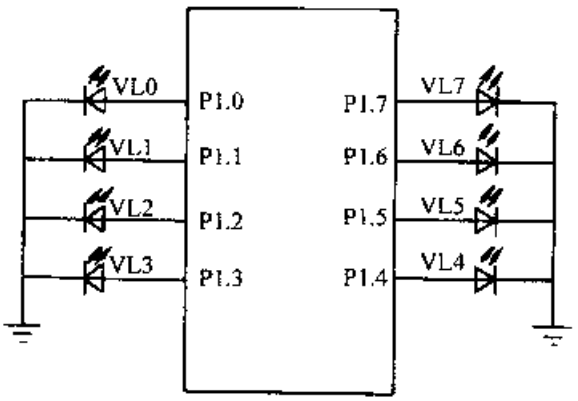


图 3-11　例 13 附图

```
#include <reg51. h>
void main ( )
{
  char a=1;
  unsigned int b;
  do {
     P1=a;
```

```
        a=a≪1;
        if (a==0) a=1;
        b=50000;
        while (--b);
        } while (1);
}
```

该例中有两层循环，do-while 循环（外层循环）嵌套了一个 while 循环（内层循环）。其中外层循环是死循环，使程序始终在执行；而内层循环的循环体是一空语句，即内层循环为空循环，不做其他事，当执行内循环时，只是让 CPU 等待了一段时间，起延时的作用，这就是所谓的软件延时。

上面我们讨论了三种循环结构，它们都以某个表达式的结果值作为循环条件，当此表达式的值为假时，就结束循环流程。除了这种正规结束循环的方式之外，还可以从循环中途退出而结束循环。实现该功能的语句是 break 语句和 continue 语句。

(1) break 语句

break 语句称为中断语句，只能用于 switch 语句和循环语句。在 switch 语句中，break 语句终止 switch 语句的执行，转去执行 switch 语句下面的一条语句。在循环语句中，break 语句可以中断本层的循环，转去执行该循环语句下面的语句。

如：

```
for (r=1; r<=8; r++)
 {
  area=3.1416*r*r;
  if (area>120) break;
  printf ("area=%f", area);
 }
```

该例是计算 r=1 到 r=8 时的圆面积 area，直到 area 大于 120 为止。从上面的 for 循环可以看到：当 area>120 时，就执行 break 语句，提前结束循环，即不再执行其余的几次循环，流程见图 3-12。

(2) continue 语句

continue 语句称为接续语句，功能是结束本次循环，即跳过本次循环尚未执行的语句，把程序流程转移到当前循环语句的下一个循环周期，并接着进行下一次是否继续循环的判断。

例 14 输出 100～200 之间的能被 3 整除的整数。

```
main ()
 {
```

```
    int n;
    for (n=100; n<=200; n++)
     {
      if (n%3! =0) continue;
      printf ("%d", n);
     }
  }
```

本例中，当 n 不能被 3 整除时，就执行 continue 语句，结束本次循环（即跳过 printf 语句），只有当 n 能被 3 整除时，才执行 printf 语句，流程见图 3-13。

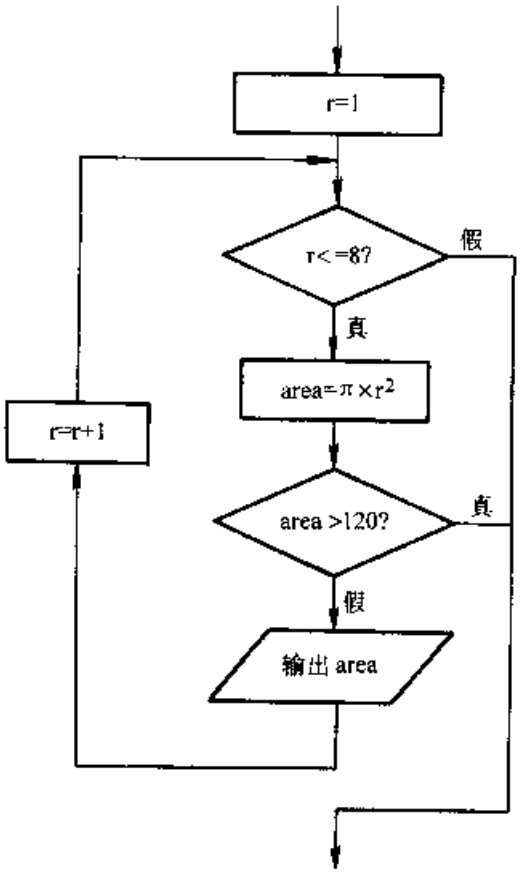


图 3-12　break 语句执行流程图

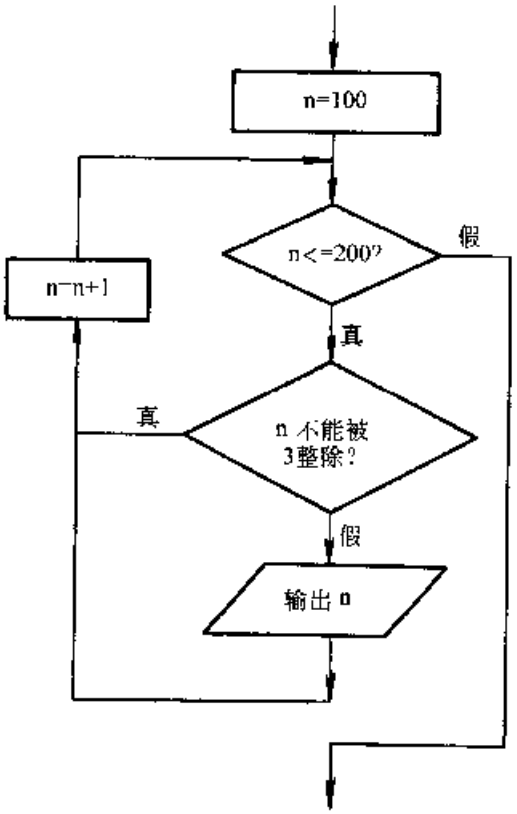


图 3-13　continue 语句执行流程图

continue 语句与 break 语句的区别：continue 只结束本次循环，break 则结束整个循环。注意，break 语句只结束本层循环，而不影响外层循环。continue 语句只能用于循环语句中。

3.3　C51 构造数据类型

3.3.1　数组

数组是相关数据的一个有序集合，数组中的每一个元素都是同一类型的数据。数据集合用一个名字来标识，称为数组名。数组中元素的顺序用下标表示，下标

表示该元素在数组中的位置。数组的下标放在方括号内，下标为 n 的元素可以表示为数组名 [n]。改变 [ ] 中的下标就可以访问数组中所有的元素。一个数组元素等同于一个变量，因此又可以说数组是一组相同数据类型的相关变量的有序集合。

1. 一维数组

由具有一个下标的数组元素组成的数组称为一维数组。

(1) 一维数组的定义

定义的一般形式：

类型说明符　数组名 [元素个数];

其中：数组名是一个标识符，元素个数是一常量表达式，不能是含有变量的表达式。

如：int a [50]; /* 定义一个数组名为 a 的数组，数组包括 50 个整型的元素 */

(2) 一维数组的使用

1) 数组与变量一样要先定义后使用。

2) 数组不能整体使用，只能逐个使用数组元素。

数组元素的一般形式：数组名 [下标]

其中：下标可以是整常量或整型表达式，其值从 0～（元素个数－1）。

在上例中，数组 a 的元素是 a [0]，a [1]，…，a [49]，而 a [50] 是不能使用的。

(3) 一维数组的赋值

在定义数组时可以对数组整体初始化，若定义后想对数组赋值，则只能对每个数组元素分别赋值。

如：int a [5] = {1, 2, 3, 4, 5}; /* 给全部元素赋值，结果：a [0] =1, a [1] =2, a [2] =3, a [3] =4, a [4] =5 */

如：int b [6] = {1, 2, 6}; /* 给部分元素赋值，结果：b [0] =1, b [1]=2, b [2] =6, b [3] =b [4] = b [5] =0 */

如：int d [10]; d [0] =4; d [1] =－6; …/* 在数组定义后要对数组赋值，则只能对每个数组元素分别赋值。若写成：int d [10]; d [10] = {4, －6}; 则是错误的。 */

例 15　求 Fibonacci 数列的前 40 项。

该数列的通项为：F1=1，F2=1，Fn=Fn-1+Fn-2（n≥3）

```
main ( ) {
  int n;
  long f [40];
  f [0] =f [1] =1;
  for (n=2; n<40; n++)
    f [n] =f [n-1] +f [n-2];
  for (n=0; n<40; n++)
     {if (n%5--0) printf ("\n");
      printf ("%12ld", f [n]);
     }
  }
```

2. 二维数组

只有一维数组是不够用的，C 语言可以使用多维数组，经常用的是二维数组。由具有两个下标的数组元素组成的数组称为二维数组。

（1）二维数组的定义

定义的一般形式：

类型说明符　数组名 [行数] [列数];

数组名是一个标识符，行数和列数都是常量表达式。

如：float a [3] [4];　　/*a 数组有 3 行 4 列共 12 个实型元素*/

a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]
a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]
a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

二维数组可以看成是一个一维数组，它的每个元素都是一个一维数组。因此，数组 a [3] [4] 有三个元素：a [0]，a [1]，a [2]，而每个元素又是一个有 4 个元素的一维数组，如 a [1] 的 4 个元素分别是：a [1] [0]，a [1] [1]，a [1] [2]，a [1] [3]。下横线表示 a [1] 是数组名。

（2）二维数组的使用

二维数组的元素表示形式：数组名 [下标] [下标]

其中：下标可以是整常量或整型表达式，其值从 0～（行或列-1）。

（3）二维数组的初始化

```
int a [2] [3] = { {1, 2, 3}, {4, 5, 6}};
```

例 16　将一个二维数组的行和列元素互换，存到另一个数组中。

实现行列互换，即 b [i] [j] =a [j] [i]。（设将 a 数组中行和列元素互换→b 数组）

```
main (  ) {
  int a [3] [4], b [4] [3], n, m;
  printf ("Input 12 numbers\n");
  for (n=0; n<3; n++)
    for (m=0; m<4; m++)
        scanf ("%d", &a [n] [m]);
  for (n=0; n<3; n++;)
    for (m=0; m<4; m++)
        b [m] [n] =a [n] [m];
    for (n=0; n<3; n++)
     {
      for (m=0; m<4; m++)
        printf ("%5d", b [n] [m]);
    printf ("\n");
    }
}
```

3. 字符数组

若一个数组的元素是字符类型的，该数组就是一个字符数组。

(1) 字符数组的定义

如：char c [8], b [3] [6]; char b [20] ="I am a student.";

(2) 字符串和字符串结束符

C 语言中没有字符串变量，需用字符数组来处理字符串。当数组中存放的实际字符个数（有效字符串长度）与数组的长度不一样时，为了测定字符串的实际长度和使用系统提供的各种字符串函数，C 语言规定了一个字符串结束标志'\0'，它是一个 ASCII 码值为 0 的字符。在一个字符数组中，一旦遇到字符'\0'，就表示字符串结束，其后的字符忽略不计。上面 b 数组的后 5 个元素皆为'\0'，字符串常量中也自动包含一个字符串结束符'\0'。

4. 查表

数组的一个很有用的用途就是查表。在单片机应用中，常需要对数学公式进行计算以及对一些传感器的非线性特性进行补偿，这时，采用查表的办法就比较简单有效。因为单片机的计算能力有限，可以将复杂的数学公式或补偿算法事先计算好表格，存入程序存储器中，而其中的表就是数组。

注意：数组是连续存放的一块存储区域，特别是位于内部 RAM 中的数组，如不能有效利用，就会浪费大量的存储空间。因此，在开发 C51 程序时，要仔细地根据需要选择数组的大小。

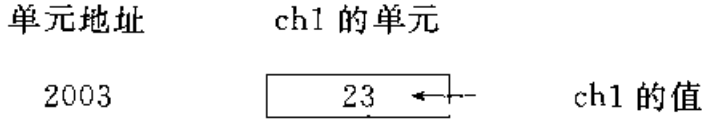
3.3.2 指针

指针是C语言中的一种重要的数据类型。指针就是对象的地址，如变量的地址、数组的地址、数组元素的地址等。用来存放指针的变量称为指针变量。指针的值是整型数据，但有其特殊的运算规律。与其他类型变量一样，可以有指针数组、甚至有存放指针变量的指针变量。

1. 变量的地址

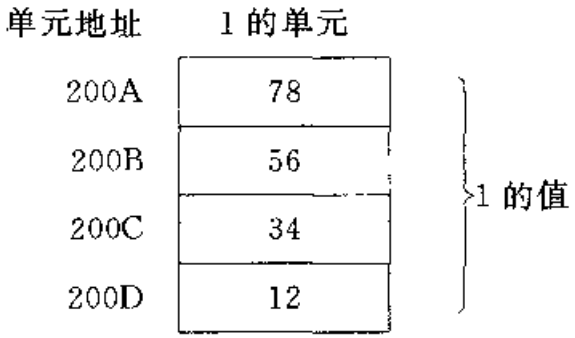
变量在内存中占用一定的内存单元，如：char 变量占用一个字节，int 变量占用连续的两个字节，long 变量占用连续的四个字节，float 变量占用连续的四个字节，double 变量占用连续的八个字节。所谓变量的地址即它们所占连续的内存单元的最低字节单元的地址。

如：char ch1=23;



变量 ch1 的地址为 2003

如：long 1=0x12345678;



变量 1 的地址为 200A

2. 指针变量的概念

指针变量是一种特殊的变量，特殊在它只能存放地址值。

(1) 指针变量的定义

定义的一般形式：

类型说明符 * 指针变量名;

其中：类型说明符指定该指针变量中只能存放这种类型变量的地址；

* 号说明这个变量是指针变量。

如：int *a; float *b;

(2) 指针变量的使用

可以通过取地址运算符和指针运算符使指针变量与变量建立某种联系。

1）&：取地址运算符，当把某变量的地址赋给一个指针变量后，就说该指针变量指向该变量。

如：int a，*pointer，b；

pointer＝&a；　　/*指针变量 pointer 指向变量 a*/

对指针变量，只能把一个变量的地址赋给它，而不能赋给它一个数值。

如：pointer＝100；是非法的。但有一个值可以赋给指针变量，记为 NULL（空指针），它的值是 0。pointer＝NULL；是合法的，意思是 pointer 指向地址为 0 的单元。

2）*：取指向运算符，即取指针变量所指向的变量的值。

如：b＝*pointer；＝＞　b＝a；

通过对指针变量取指向运算使用变量的值称为间接寻址。即指针变量 pointer 的值为 a 的地址，先由 pointer 得到 a 的地址，再从该地址单元中取出 a 的值赋给变量 b。

&和*都是单目运算符，优先级相同，且其结合顺序是自右至左的。

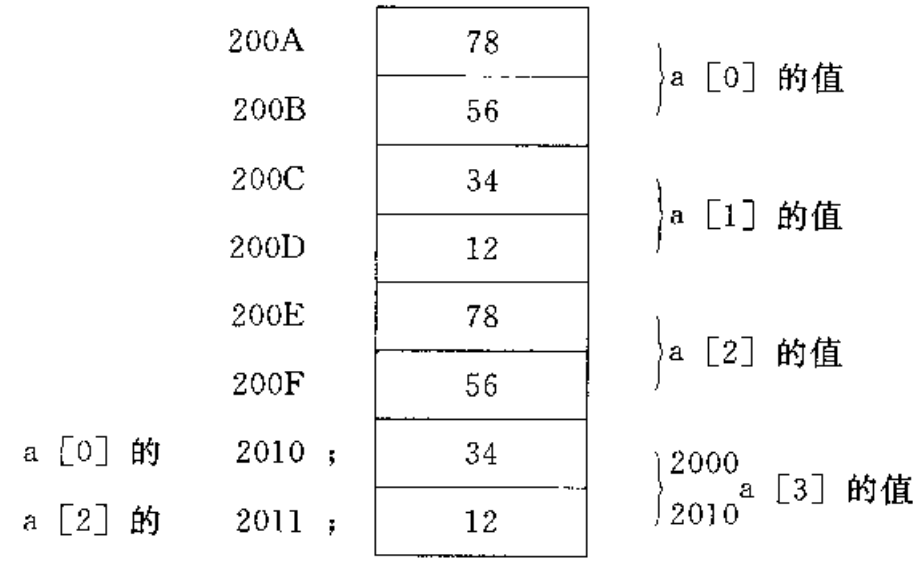
如：*pointer^{-+}等价于*（pointer^{+-}）而不是（*pointer）$^{-+}$。

3. 指针和一维数组

（1）一维数组的地址

一维数组在内存中的存放：从下标为 0 的元素开始，连续存放。下标为 0 的元素的地址即为整个数组的地址。

如：int a［4］＝｛0x5678，0x1234，0x5678，0x1234｝；



数组 a 的地址为 a［0］的地址＝200A。

（2）指针的使用

要用指针变量来处理一维数组只需将一维数组的地址赋给一个指针变量即可。

如：int *p，a [4]；

p=a；　　/*数组 a 的地址赋给指针变量 p*/

这样，指针变量 p 中存放了数组 a 的地址，即 p 指向了数组 a，则 *p=a [0]。由上式可以看出，数组名就代表数组的地址，又称为数组的首地址。

指针的值虽是整型数据，但有其特殊的运算规律。对指针变量，其算术运算的特殊性表现在：

1）只能加减，不能乘除。

2）当某指针变量指向一个数组时，对该指针变量加 1，则不管这个数组是何类型，这个指针变量都指向数组的下一个元素（只要没越界）。如：做完 p++运算后，p 就指向数组 a 的下一个元素，即 a [1]。

综上所述，数组元素的引用可以采用两种方法：下标法和指针法。

① 下标法：

如：int a [10]，*p；p=a；

数组的第 n 个元素可表示为：a [n] 或 p [n]

② 指针法：

数组的第 n 个元素可表示为：*（a+n）或 *（p+n）

注意：数组名可看成是一个指针常量，其值不能改变。

例 17 用选择法对数组进行排序（从小到大）。

[方法思路] 设数组长度为 5，令 i=0

(1) 寻找 a [i] 及后面的元素中最小元素，a [k]；

(2) 若 k==i，转 (4)；

(3) 若 k!=i，交换 a [i] 与 a [k] 的值；

(4) i++，若 i<4，转 (1)；否则结束。

程序如下：

```
main ( ) {
    int a [5] = {15, 10, 2, -78, 5}, i, k, j, temp;
  for (i=0; i<4; i++)
      {k=i;
   for (j=k+1; j<5; j++)              /* [1] 步 */
     if (a [j] <a [k]) k=j; }
    if (k! =i)                        /* [3] 步 */
      {temp=a [i]; a [i] =a [k];
       a [k] =temp;
      }
```

```
        }
    for (i=0; i<5; i++)
      printf ("%5d", a [i]);
}
```

4. 指针与二维数组

(1) 二维数组的地址

int a [3] [4] = { {23, 32, 13, 45}, {1268, 231, 2, 87}, {38, 8, 63, 981}};

该二维数组在内存中的存放：

地址：	1000	1002	1004	1006
值：	23	32	13	45
元素：	a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]
地址：	1008	100A	100C	100E
值：	1268	231	2	87
元素：	a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]
地址：	1010	1012	1014	1016
值：	38	8	63	981
元素：	a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

一个 n 行 m 列的二维数组 a [n] [m]，可以看作是一个有 n 个元素的一维数组，其每个元素又是一个具有 m 个元素的一维数组。

数组名只负责寻址数组中的行，即 a+i 指向数组的第 i 行，这称为宏观管理。若要寻找第 i 行的第 j 个元素，必须是指针先指向该行的第 0 个元素，然后再寻找其第 j 个元素，即 * (a+i) +j，取它的指向就是 a [i] [j] = * (* (a+i) + j)，这称为微观管理。

(2) 指向二维数组的指针变量

可以定义指针变量指向二维数组的元素。

例 18 用指针变量输出数组元素的值。

程序如下：

```
main ( )
{
  int a [3] [2] = { {1, 5}, {6, 9}, {5, 7}}, *p;
  for (p=a [0]; p<a [0] +6; p++)      /*指针 p 指向 a [0] [0], p++
     后指向 a [0] [1] */
    printf ("%5d", *p);
```

```
}
```

5. 指针数组和指向指针的指针变量

(1) 指针数组

与普通变量一样，也可以定义指针数组，即每个元素都是同类型的指针变量。常令指针数组指向若干个字符串，这使得字符串的处理更为方便。

其定义的一般形式是：

类型说明符 * 数组名 [元素个数];

例 19 用选择法对 10 个字符串排序。

程序如下：

```
#include"stdio. h"
main ( ) {
  char ch [10] [81], *p [10], *p1; /*p 为指针数组，包含 10 个指针元
    素*/
  int i, j, k;
  printf ("Enter 10 strings: \n");
  for (i=0; i<10; i++)    /*输入 10 个字符串*/
      gets (ch [i]);
  for (n=0; n<10; n''') /*使 p 的 10 个指针元素分别指向上面 10 个字符
     串*/
      p [n] =ch [n];
  for (i=0; i<9; i++)
     {k=i;
  for (j=i+1; j<10; j++)
      if (strcmp (p [j], p [k]) <0) k=j;
  if (k! =i)
       {p1=p [k]; p [k] =p [i]; p [i] =p1;}
    }
for (i=0; i<10; i++) puts (p [i]);
}
```

(2) 指向指针变量的指针

指针变量在内存中占用一定的单元，即也有地址。如果一个指针变量用来存放另一个指针变量的地址，这个变量就称为指向指针变量的指针变量。

其定义的一般形式是：

类型说明符 * * 变量名;

其中：类型说明符是指被指向的指针变量的类型。

如：char * * p，*p1；

p=&p1；

6. 关于C51的指针类型

C51支持“基于存储器”的指针和“一般”指针。当定义一个指针变量时，若未指定它所指向的对象的存储类型，则该指针变量被认为是一般指针；反之若指定了它所指对象的存储类型，则该指针变量被认为是基于存储器的指针。

基于存储器的指针类型由C源代码中指定的存储器类型决定，并在编译时确定，这种指针只需1～2个字节，且高效。

一般指针需占用3个字节：第一个字节为存储器类型的编码（由编译模式的默认值确定），剩余两个字节为地址偏移量。存储器类型决定了对象所用的8051存储空间，偏移量指向实际地址。一个“一般”指针可以访问任何变量而不管它在8051存储器空间的位置。这就允许一般函数，如memcpy（）等将数据从任意一个地址拷贝到另一个地址空间。

（1）基于存储器的指针

基于存储器的指针是在说明一个指针变量时，指定它所指向的对象的存储类型。

如：char xdata* px；

px为指向一个定义在xdata存储器中的字符变量的指针变量。px本身在默认的存储器区域（由编译模式决定），其长度为2个字节。

C51有三种存储模式：SMALL、COMPACT和LARGE。C51的存储模式确定函数的参数与局部变量的存储区域。在SMALL模式下，函数的参数与局部变量位于8051的内部RAM，在COMPACT和LARGE模式下，函数的参数与局部变量位于8051的外部RAM。

如：char xdata* data py；

py为指向一个定义在xdata存储器中的字符变量的指针变量。px本身在内部RAM中，与编译模式无关，其长度也为2个字节。

（2）一般指针

在函数调用中，函数的指针参数需要用“一般”指针。一般指针的说明形式为：

char *pz；

这里没有指定指针变量pz所指向的变量的存储类型，pz处于编译模式默认的存储区，长度为3个字节，格式为：

地　址	+0	+1	+2
内　容	存储类型的编码	高位地址偏移量	低位地址偏移量

其中，存储类型由编译模式决定，不同的存储区域的编码如下：

存储类型	idata	xdata	pdata	data	code
编码值	1	2	3	4	5

在用常量作指针时，如定义外部端口的地址，必须注意正确定义存储类型和偏移。

如：要将数值 0x41 写入地址为 0x8000 的外部数据存储器。

则这样写即可：#include "absacc. h"

XBYTE [0x8000] =0x41;

其中 XBYTE 是一个指针，它是在头文件 absacc. h 中定义的，定义如下：

#define XBYTE ((unsigned char*) 0x20000L)

XBYTE 被定义为 (unsigned char*) 0x20000L，为一般指针，其存储类型为 2，即为 xdata 类型，偏移量是 0000，这样，XBYTE 成为指向外部数据存储器的零地址单元的指针。而 XBYTE [0x8000] 则表示外部数据存储器的 0x8000 单元。

3.3.3 结构体

数组是构造类型数据，其中数组中的各元素必须是同一类型的，但在描述一个实体时，只有一种数据类型往往是不够的。如描述一个学生一个学期的成绩，就会有如下的数据：姓名，性别，班级，学号，各科成绩等，这些数据的类型各不相同，有字符串类型，字符类型及整型数据，显然用数组是解决不了的。

C 语言提供了一种数据结构，称为结构体类型，它是将若干个不同类型的数据元素组合在一起构成的一种数据类型。显然一种结构体类型可以有若干个元素，每个元素的类型可以不同，每个元素称为结构体的成员。

由于描述不同的实体需要不同的属性（即不同的成员个数和类型），因此，当使用结构体时，必须先定义结构体类型，再说明该类型的变量。这是与数组的使用不同的地方。

1. 结构体类型及结构体变量

(1) 结构体类型的定义

定义的一般形式：

```
struct  结构体类型名
   {    数据类型    成员名 1;
        数据类型    成员名 2;      /*  成员列表  */
             ⋮
   };
```

其中：struct 是定义结构体类型的关键字；

结构体类型名是一个标识符；

成员列表是对若干成员的说明。

如：struct student /* struct student 为定义的结构体类型名 */
{ char name [10]; /* 以下都是该结构体类型所包含的成员 */
char sex;
unsigned long num;
int eng;
int math;
int phys;
};

如：struct Birthday /* 结构体类型 struct birthday 包括以下三个成员 */
{ int year, month, int day;};

(2) 结构体变量的定义

定义了需要的结构体类型之后，就可以定义该类型的结构体变量了。结构体变量的定义形式：

struct 结构体类型名 变量名列表;

如：struct student
{ char name [10];
char sex;
unsigned long num;
int eng, math, phys;
};
struct student stu1, stu2;

该例是先定义结构体类型 struct student，再定义该类型的两个结构体变量 stu1，stu2。

如：struct student
{ char name [10]; char sex;
unsigned long num;
int eng; int math; int phys;
} stu1, stu2;

该例是在定义结构体类型 struct student 的同时定义了该类型的两个结构体变量 stu1，stu2，此时 student 可省略。

2. 结构体变量的引用

在定义了一个结构体变量之后，就可以对它进行引用，即赋值、存取及运算。一般，结构体变量的引用是通过对其结构体成员项数据的引用来实现的。结构体成员项可用取成员运算符“.”来表示，此时对成员的操作就像对变量操作一样。

如：struct student
{ char name [10];
char sex;
unsigned long num;
int eng, math, phys;
} stu1, stu2;

结构体变量 stu1 的 name 成员、sex 成员、num 成员及 eng 成员分别表示为：stu1.name、stu1.sex、stu1.num、stu1.eng

如：struct Birthday
{ int year; int month; int day;};
struct person
{ char name [10];
char sex;
char addre [50];
struct Birthday birth;
} birth1, birth2;

对于结构体变量 birth1 和 birth2，由于其成员 birth 又是一个结构体变量，因此对于它们的成员 year、month、day 的使用就需要使用两次取成员运算符：

birth1.birth.year、birth1.birth.month

说明：

1）结构体变量不能整体使用，这与数组是相同的。但有一点例外，即可以把一个结构体变量整体赋给另一个同类型的结构体变量。

如：若有结构体变量 stu1、stu2，则可以将 stu1 整体赋给 stu2，即 stu2=stu1；是合法的。

结构体变量的如下引用显然是错误的：

printf ("%s,%c,%1d,%d,%d,%d", stu1);

只能通过成员项来引用，即 printf("%s,%c,%1d,%d,%d,%d",stu1.name, stu1.sex, stu1.num, stu1.eng, stu1.math, stu1.phys);

2）结构体的成员名可以与程序中的变量同名。

3. 结构体变量的初始化

可以对结构体变量的各成员分别赋值，也可以在定义结构体变量时整体初始化。

如：struct student stu1= {"Li ming",'m', 90101203, 89, 76, 93};

说明：对结构体变量不能整体赋值。

如：struct student stu1;
stu1= {"Li ming",'m', 90101203, 89, 76, 93};

显然这样赋值是非法的，只能对它的每一个成员单独赋值。正确的赋值如下：

```
stu.sex='m';
stu.num=90101203;
strcpy (stu.name,"Li ming");
stu.eng=89;
stu.math=76;
stu.phys=93;
```

例 20 结构体变量的初始化及引用。

```
main ( )
{ struct student
   { char name [10];
     char sex;
     unsigned long num;
     int eng, math, phys;
  } stu1= {"He Ling",'m', 99103321, 73, 90, 88};
printf ("No.:        %lud \n",   stu1.num);
printf ("name:       %s   \n",   stu1.name);
printf ("sex:        %c   \n",   stu1.sex);
printf ("English:    %d   \n",   stu1.eng);
printf ("Math:       %d   \n",   stu1.math);
printf ("Physics:    %d   \n",   stu1.phys);
}
```

4. 结构体数组

在具体应用中，通常要用到多个类型相同的结构体变量，这时就可以把这些相同类型的结构体变量集合在一起，构成结构体数组。结构体数组的特点就是数组中的每个元素都具有相同的结构体类型。结构体数组的定义与初始化都和结构体变量类似。

```
如：struct student stu1[30];   /*定义一个 struct student 类型的数组 stu1，
                                 即该数组的 30 个元素都为 struct student 类型
                                 */
如：struct student stu2 [3] =      /*在定义结构体数组时整体初始化*/
     {  {"Li ming",'m', 90103303, 89, 76, 93},
        {"Luo jun",'m', 90103305, 83, 62, 76},
        {"Li ping",'f', 90103310, 93, 91, 89}};
```

5. 指向结构体类型数据的指针

(1) 指向结构体变量的指针变量

可以定义指向结构体变量的指针变量，如：struct student stu1， *p；这里变量 p 就是一个指向结构体变量的指针变量，只需把某个结构体变量的地址赋给它，该指针变量就指向这个结构体变量。

如：p=&stu1；　　/*p 指向结构体变量 stu1，即 *p 就是 stu */

取结构体变量 stu1 的成员可有三种方法：

1) stu1.num，stu1.eng

2) (*p).num，(*p).eng

注意：'.'的优先级高于'*'的优先级，故要加括号。

3) 利用运算符'->'

p->num，p->eng

(2) 指向结构体数组的指针变量

结构体指针除了可以指向结构体变量，还可以指向结构体数组。

如：struct student *p，stu2 [20]；

p=stu2；　　/*　p 指向结构体数组 stu2 */

这样 p 指向数组 stu2 的第一个元素 stu2 [0]，即 *p=stu2 [0]。此时若 p^{-+}，则 p 指向数组 stu2 的第二个元素，即 *p=stu2 [2]。

3.3.4　共用体

1. 共用体的概念

共用体是另一类需用户自定义的数据类型。共用体类型数据可以有若干个数据成员，每个数据成员的类型可以不同，这与结构体类型数据相同。两者不同的是：结构体类型数据的各成员在内存中分别占有自己的内存单元，并连续存放；而共用体类型数据的各成员占用同一块内存单元，即，各成员的地址是相同的。这种技术可使不同的数据分时使用同一个内存空间，提高内存的利用效率。

共用体数据类型的定义形式：

union　共用体类型名 {成员列表}；

其中，union 是定义共用体数据类型的关键字；

共用体类型名是一个标识符；

成员列表与结构体的成员列表相同。

如：union data
{ char c1；
char c2 [2]；
int k；
} x；

2000H 地址单元	2001H 地址单元
x.c1	
x.c2 [0]	x.c2 [1]
x.k 的低字节	x.k 的高字节

共用体类型 union data 有三个数据成员：字符变量 c1、字符数组 c2、整型变

量 k、x 是该共用体类型的一个变量。各数据成员 c1、c2 及 k 都从同一地址（假设为 2000H）开始存放，这样各成员相互覆盖，占据相同的内存单元。即 c1、c2 [0] 及 k 的低字节占相同的内存单元，c2 [1] 与 k 的高字节占相同的内存单元。而该共用体变量 x 所占的内存单元数是 2 个字节，即共用体中数据长度最长的成员所需单元数（int 类型所需单元数）。

由于共用体类型数据的特点，当一个成员的值改变时，就同时改变了其他成员的值。

2. 共用体变量的引用方式

共用体变量的引用方式与结构体变量的引用方式相同，要使用成员引用运算符'.'。

对上述共用体变量 x 可以这样引用：x.c1、x.k、x.c2 [0]、x.c2 [1]

也可以定义指向共用体变量的指针变量。

如：union x, *p;

```
p=&x;                    /*指针变量 p 指向共用体变量 x*/
p->c1; 或 (*p).k;        /*取共用体变量 stu1 的成员*/
```

3. 共用体类型数据的特点

1）共用体各成员的地址都相同。由于共用体类型数据的特点，当一个成员的值改变时，就同时改变了其他成员的值。

例 21 分析以下程序的结果：

```
main ( )
{ union
    { char c1;
      char c2 [2];
      int k;
    } x;
    x.c1=0x61;
    x.c2 [1] =0x62;
    printf ("%x,%x", x.k, x.c2 [0]);
}
```

由于 c1、c2 [0] 及 k 的低字节占相同的内存单元，而 c2 [1] 与 k 的高字节占相同的内存单元，所以由 x.c1=0x61，x.c2 [1] =0x62→x.c2 [0] =0x61，x.k =0x6261。

2）共用体变量不能作函数的参数，函数也不能返回共用体类型数据，但可以用指向共用体变量的指针变量作函数的参数。

3）共用体类型可以出现在结构体类型定义中，结构体类型也可以作为共用体

类型的成员，即两者可以相互嵌套。

例 22 分析以下程序的结果：

现有 union

```
{ int x;              /* 整型变量 x 为共用体变量 a 的一个成员 */
  struct              /* 结构体变量 y 为共用体变量 a 的一个成员 */
  { char c1;
    char c2;          /* 结构体变量 y 又有两个字符型成员 c1、c2 */
    } y;
} a;
```

若 a.x＝0x1234，则 a.y.c1＝0x34，a.y.c2＝0x12。

3.3.5 枚举

通常的变量取值都是无穷多个，没办法都列举出来。但有些变量的取值是有限的，如逻辑量，只有真和假两个；一个星期的天数只有 7 天；在描述某些物体时，它的颜色种类是有限的等等。对这些取值有限的变量，在程序中可以把它们一一列举出来，形成一个集合，便于使用，这就是枚举类型。

枚举类型的定义形式：

enum 类型名 { 枚举常量列表 };

其中：enum 是定义枚举类型的关键字；

类型名是要定义的枚举类型的名称；

枚举常量列表是这个枚举类型数据可能取的所有值。

如：enum weekday {sun, mon, tue, wed, thu, fri, sat};

当定义了枚举类型后，就可以定义这种类型的变量了。

如：enum weekday workday, week_end;

这些枚举类型变量的值只能是 sun 到 sat 之一。

如：weekday＝mon;　　week_end＝sat;

枚举类型定义中的 sun，mon，…，sat 等都是枚举常量，它们的值按定义的顺序分别为 0，1，…，6。也可以在定义枚举类型时改变这些常量的值，

如：enum weekday {sun, mon＝3, tue, wed, thu, fri, sat};

这时 sun 的值为 0，mon 的值为 3，其后的几个常量的值依次递增，分别为 3，4，5，6，7，8。

枚举量可以相互比较，比较的是它们在类型定义时的位置顺序。

3.4 函数

3.4.1 概述

在前面我们看到，可以将一个 C 程序的所有功能都放在一个主函数中实现，

但是当程序较大时，它的阅读、调试和修改都比较困难。一般做法是将一个较大的程序分成若干个子程序模块，每个子程序模块实现一种基本功能。在C语言中，子程序是通过函数来实现的，可以将自己所写的各功能函数做成一个专门的函数库，由不同的程序、甚至可以由不同的用户调用，就像C提供的系统函数一样。这种模块化的程序设计方法可以大大提高编程效率。

例23 P1口接八个发光二极管，分别为VL0、VL1…VL7。程序先点亮VL0，延迟一段时间，再顺序点亮VL1…VL7，然后又是VL0。同时只能有一个灯亮(见图3-11)。

该例在前面是用两层循环来实现的，其中的内层循环为空循环，不做其他事，只起延时的作用。此时我们可以自定义一个函数，其功能就是延时。在主程序中调用该函数就可以实现延时。

```
# include <reg51.h>
void delay ( )                          /* 延时函数 delay ( ) 的定义 */
{
    int a=50000;
    while (--a);
}
void main ( )
{
    char a=1;
    do {
        P1=a;
        a=a<<1;
        if (a==0) a=1;
        delay ( );                      /* 延时函数的调用 */
        } while (1);
}
```

C程序的主函数是一个特殊的函数，每个程序必须有而且只能有一个主函数，但可以有若干个其他函数。这些函数的关系是：程序的运行总是从main ( ) 函数开始的，主函数可以调用其他函数，调用完毕后回到主函数，在主函数中结束整个程序的运行。

注意：其他函数之间可以互相调用，但它们不能调用主函数。

函数可分为无参函数和有参函数，它们的区别在于被调用时，是否有参数输入。如：系统函数sqrt ( ) 是一个有参函数，而系统函数getchar ( ) 是一个无参函数。

调用函数时，有的函数会返回一个数值，如 sin（）、sqrt（）等函数，这些函数称为有返回值的函数，而有的函数只完成一定功能是没有数值结果的，这些函数称为无返回值的函数。

3.4.2 函数的定义

函数定义的一般形式为：

返回值类型　函数名（形式参数列表）

{

函数体

}

其中：

1）关于返回值类型：

① 可以是基本数据类型（int，char，float，double 等）及指针类型。

② 当函数没有返回值时，用标识符 void 说明该函数没有返回值。

③ 若没有指定返回值类型，默认返回值为整型类型。

④ 一个函数只能有一个返回值，该返回值是通过函数中的 return 语句获得的。

2）函数名必须是一个合法标识符。

3）形参参数表列包括了函数所需全部参数的定义。此时函数的参数称为形式参数，简称形参。形参可以是基本数据类型的数据、指针类型数据、数组等。在没有调用函数时，函数的形参和函数内部的变量未被分配内存单元，即它们是不存在的。

4）函数体由两部分组成：函数内部变量定义和函数体其他语句。

5）各函数的定义是独立的。

函数的定义不能在另一个函数的内部，即函数的定义不能嵌套。

3.4.3 函数的调用

函数调用的一般形式为：

函数名（实际参数列表）；

在一个函数中需要用到某个函数的功能时，就调用该函数。调用者称为主调函数，被调用者称为被调函数。如：在 a 函数中调用 b 函数，则 a 函数为主调函数，b 函数为被调函数。若被调函数是有参函数，则主调函数必须把被调函数所需的参数传递给被调函数，传递给被调函数的数据称为实际参数，简称实参。若被调函数是无参函数，则调用该函数时，可以没有实参列表，但括号不能省。被调函数执行完后再返回主调函数继续执行剩余程序。

注：① 实参与形参在数量、类型和顺序上都要一致。

② 实参可以是常量、变量或表达式。

③ 实参对形参的数据传递是单向的值传递，即只能实参→形参，而不能形参→实参。

函数的参数可以是多种类型，基本数据类型、指针类型、数组等都能作函数的参数。下面来讨论各种类型的数据作函数参数的情况。

(1) 普通变量作函数参数

例 24 编写函数求三个整数中的最大值。

```
int max (int x, int y, int z)
  { int a=x;
    if (y>x)    a=y;
    if (z>a)    a=z;
    return a;
  }

main ()
{ int x1, y1, z1;
  printf ("Enter 3 numbers\n");
  scanf ("%d %d %d", &x1, &y1, &z1);
  printf ("The max is %d\n", max (x1, y1, z1));
}
```

例中：① 函数 max 有三个形参 x、y、z，都是整型参数；
函数 main 有三个实参 x1、y1、z1，也都是整型。

② 变量 a 是函数 max 的内部变量，只在函数 max 内有效。

③ 函数的返回值是通过语句 return a；实现的。

④ 一个函数只能返回一个值，可以有多个 return 语句，但只有一个被执行。

⑤ 函数的形参和内部变量可以与其他函数的形参和内部变量同名。

例 25 有如下的函数定义：

```
void exchange (int x, int y)
{ int temp;
  temp=x; x=y; y=temp;    }

main (  )
{ int a, b;
  scanf ("%d%d", &a, &b);
  printf ("a=%d, b=%d\n", a, b);
  exchange (a, b);
  printf ("a=%d, b=%d\n", a, b);
```

}

该例中，主函数调用 exchange 函数，希望交换变量 a 和 b 的值。但由于参数传递是单方向的，实参可以传递给形参，但形参不能传递给实参。所以形参 x、y 的改变并不会影响主函数的变量 a 和 b。程序的执行过程见图 3-14。

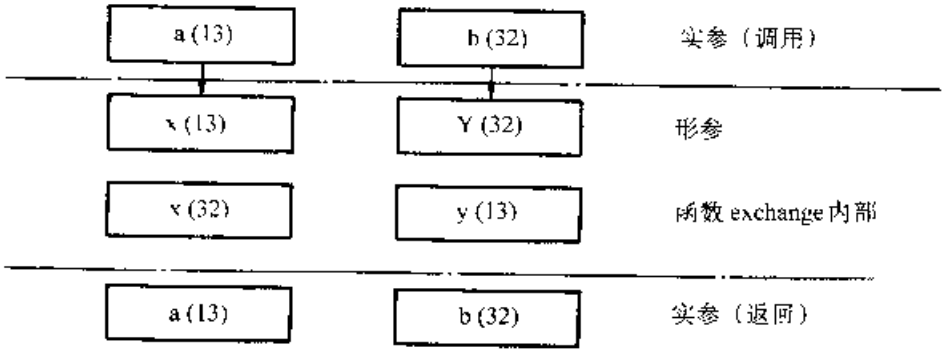


图 3-14　例 25 的程序执行过程

(2) 指针变量作函数参数

当调用函数时，若实参是指针类型，与之对应的形参也必须是指针类型。

例 26　若对例 25 做如下修改：

```
void exchange (int *x, int *y)
{  int temp;
   temp=*x; *x=*y; *y=temp;
}

main ( )
{ int a, b;
  scanf ("%d %d", &a, &b);
  printf ("a=%d, b=%d\n", a, b);
  exchange (&a, &b);
  printf ("a=%d, b=%d\n", a, b);
}
```

这里，函数的形参 x、y 是指针变量，主函数传递给函数的是两个变量 a、b 的地址。修改的是指针变量所指向的变量值，而不是指针变量本身。因此，当函数调用返回后，主函数中的变量 a、b 的值被交换了。

主函数中：

地址	a	地址	b
1000	13	1002	32

函数 exchange 被调用时：

地址	x	地址	y
100A	1000（a 的地址）	100B	1002（b 的地址）

exchange 函数中交换 * x 与 * y 的值时：

地址	* x	地址	* y
1000	32	1002	13

函数 exchange 返回后，主函数中：

地址	a	地址	b
1000	32	1002	13

若把上述函数修改一下：

```
void exchange (int * x, int * y)
 {  int * temp;
    temp=x; x=y; y=temp;
 }
```

则就不能完成交换变量值的功能了。

(3) 数组作函数参数

除了可以用变量作函数参数外，数组也可作函数参数。这里分两种情况：一是数组元素作函数参数，其用法与变量相同；还有一种是数组名作函数参数，则调用函数时，传递的是数组的首地址。

例 27 编写函数求一数组中的最大值。

```
int max (int y [ ], int n)                    /* 数组名 y 作形参之一 */
{  int m, a=y [0];
   for (m=1; m<n; m++)
       if (y [m] >a)      a=y [m];
   return a;
}

main ( )
{  int x [5] = {3, 6, 4, 28, 35};
   printf ("The max item is %d\n", max (x, 5));
                                              /* 数组名 x 作实参之一 */
}
```

注：① 形参数组可不指定大小，只在定义形参数组时在数组名后跟一个空的方括号。为了方便在被调函数中处理数组元素，可另设一个形参以传递实参数组的大小（如上例中的 n）。

② 用数组名作函数参数时，并不是把数组的值传递给形参，而是把实参数组的起始地址传递给形参数组，这样就使得两个数组占用同一段内存单元，一旦形参数组的某元素的值发生变化，将会导致实参数组对应的元素的值也随之变化。

③ 由指针变量和一维数组的关系，上述函数也可写为：

```
int max (int *p, int n)
{  int a=*p, m;
   for (m=1; m<=n; m++)
       if (*(p+m)>a)      a=*(p+m);
   return a;
}
```

总结函数调用的过程：在函数没有被调用时，函数的形参和内部变量没有分配内存。当一个主调函数调用函数时，为该函数的形参和内部变量分配内存单元，并把主调函数传递的实际参数赋给形参。因为形参和实参不占用相同的内存单元，所以形参的变化不会影响实参。当函数执行到 return 语句时，系统释放函数形参和内部变量占用的内存单元，并返回到主调函数中继续执行。

3.4.4 函数的嵌套调用与递归调用

1. 函数的嵌套调用

C 语言不能嵌套定义函数，但可以嵌套调用函数，即在调用一个函数的过程中，又调用另一个函数。

例 28 分析以下程序的执行过程。

```
int f2 (int x2)
{ return 2 * x2+1; }

int f3 (int * x3)
{ int y3;
  y3=*x3;
  *x3=1;
  return y3 * y3;
}

int f1 (int * x1)
{ int x;
    x=f3 (x1);
return f2 (x);
```

```
}

main ( )
   { int x=2, y;
     y=f1 (&x);
     printf ("%d, %d\n", x, y);
   }
```

本程序的执行过程见图 3-15。

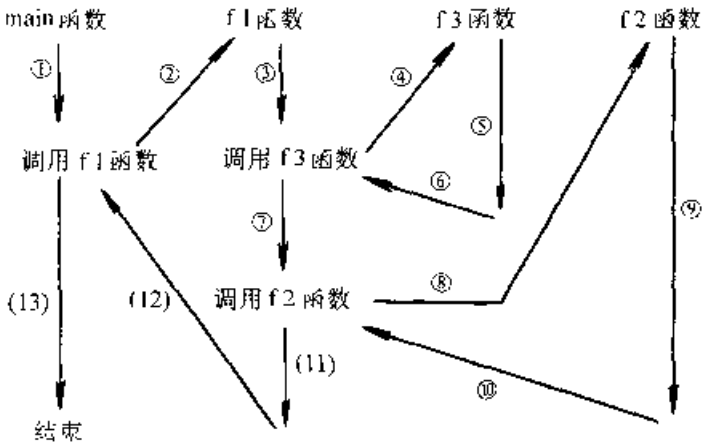


图 3-15　例 28 程序的执行过程

2. 函数的递归调用

一个函数在其函数体内调用自己，称递归调用。递归调用是一种特殊的循环结构。在 C51 编程中，递归函数必须是可重入的，可重入的函数必须加关键字 reentrant。

例 29　求 n!

设函数 f1 (n) =n!，则 f1 (n) 可用如下的递归公式来表示：

$$f1(n)=\begin{cases} n\times f1(n-1) & (n>1) \\ 1 & (n=0,\ 1) \end{cases}$$

由递归公式，可以很快得到下面的递归程序：

```
float f1 (int n) reentrant
{ if (n==0 || n==1)    return 1;                    /* 递归结束条件 */
  return n *f1 (n-1);
}

main ( )
```

```
{ printf ("%d\n", f1 (4));}
```

若采用正常的循环来求 n!，设 n! 用函数 f2 表示，则程序如下：

```
float f2 (int n)
{   int m; float a=1;
    for (m=1; m<=n; m++)    a*=m;
    return a;
}
```

比较两个函数的工作过程：

① 使用正常循环结构的函数：从已知条件 f2 (0) 或 f2 (1) =1 开始，由 f2 (2) =2*f2 (1)，求得 f (2)；再由 f (3) =3*f (2) 求得 f (3)，由此一直求到指定的 n 为止。即从已知初始条件递推到要求的值。

② 使用递归结构的函数：根据 f1 (n) =n*f1 (n-1) 的规律，用 f (n-1) 求 f (n)，但 f (n-1) 也是未知的，于是又使用 f (n-1) = (n-1) *f (n-2) 的规律用 f (n-2) 确定 f (n-1)。如此进行，直到要由 f (1) 去确定 f (2) 时，f (1) 的值是已知的，因而 f (2) 就被求出了。再按相反方向，由得出的 f (2) 的值求出 f (3)，…，直到得出 f (n-1)，并由 f (n) =n*f (n-1) 就确定了 f (n) 的值。递归函数 f (n) 的执行过程见图 3-16。

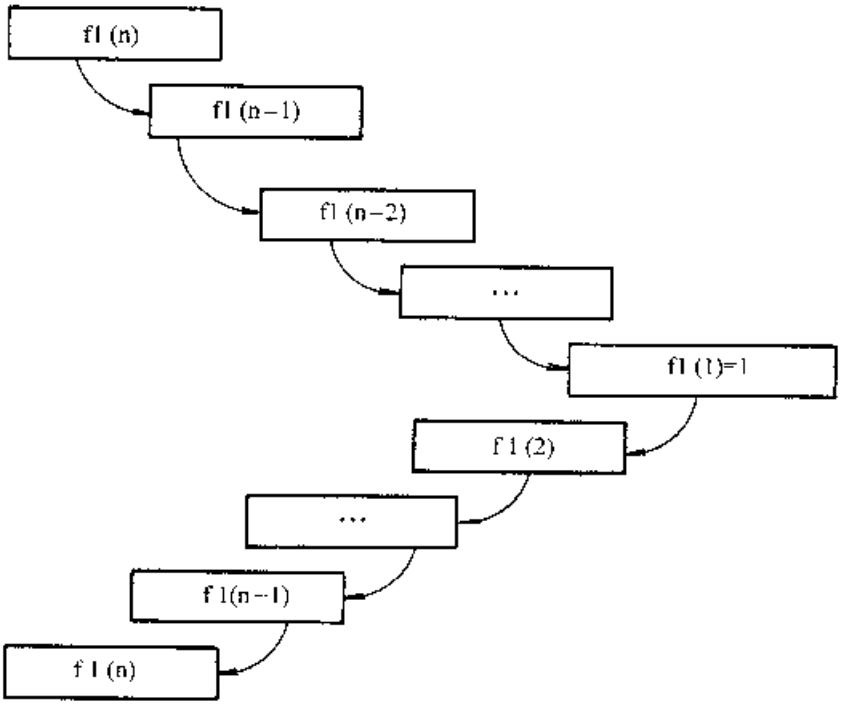


图 3-16 递归函数 f (n) 的执行过程

所以，递归函数的工作原理就是，先按一定的规律从要求的未知量逐步递推到已知量，这个过程实际上就是确定递推关系；再从已知量反向递推到要求的未知量。这就要求递归函数要包含两个要素：递归规律和递归结束条件。在上面的递归函数中，递归规律是：f1（n）=n*f1（n-1），递归结束条件是：当n==0或n==1时，f1（n）=1。

例30 一只猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天又将吃剩下的桃子吃掉一半，又多吃了一个。以后每天都吃了前一天剩下的一半零一个。到第十天早上，只剩下一个桃子。求第一天共摘多少桃子？

按题意：第n+1天早上的桃子数与前一天早上的桃子数的关系是：

$$peach(n) = (peach(n+1)+1)*2$$

现知道第十天早上的桃子数是1，即peach（10）=1，要求第一天早上的桃子数。程序：

```
int peach (int n) reentrant
{  if (n==10)      return 1;
   return (peach (n+1) +1) *2;
}

main (  )
{ printf ("The number of peach is:%d\n", peach (1));}
```

3.4.5 指向函数的指针变量

在把程序调入内存运行时，每一个函数都被分配了内存单元。我们将函数的第一条指令所在单元的地址称为该函数的（入口）地址，可以定义一个指针变量用来存放函数的地址，然后通过该指针变量就可调用此函数。

指向函数的指针变量的定义的一般形式：

类型说明符 （*指针变量名） （形参表列）；

其中：类型说明符指定了指针所指函数的返回值类型；

形参表列指定了指针所指函数的参数个数及类型。

若有如下的函数定义：

```
double max (double x, double y)
{
        函数体
}
```

可以定义指针变量p指向该函数。定义如下：

```
double (*p) (double, double);
```

一旦定义了一个指向某类函数的指针变量后，这个指针变量就只能指向该类函数，即返回值相同、参数的个数、类型、顺序都相同的一类函数，而不能是任意的函数。

例 31 实现一个简单的计算器，能完成加减乘除和取余运算。

```
int add (int x, int y)    /* add 函数实现加法功能 */
{  return x + y;   }

int sub (int x, int y)    /* sub 函数实现减法功能 */
{  return x   y;   }

int mul (int x, int y)    /* mul 函数实现乘法功能 */
{  return x * y;   }

int div (int x, int y)    /* div 函数实现除法功能 */
{  return x / y;   }

int mod (int x, int y)    /* mod 函数实现取余功能 */
{  return x % y;   }
                          /* 上面五个函数的返回值类型及参数都相同 */
main ( )
{  int x, y, c; char z;
   int (*p) (int, int); /* p 定义为指向函数的指针 */
while (1)
{  printf ("Input the first number: \n");
   scanf ("%d", &x);  /* 输入第一个操作数 */
   printf ("Input the second number: \n");
   scanf ("%d", &y);  /* 输入第二个操作数 */
   printf ("Choice the operator: 1  +; 2  -; 3  *; 4  /; 5  %\n");
   scanf ("%d", &z);  /* 输入运算符代码 */
   switch (z)
       {case 1: p=add; c='+'; break; /* 将 add 函数地址赋给 p */
        case 2: p=sub; c='-'; break; /* 将 sub 函数地址赋给 p */
        case 3: p=mul; c='*'; break; /* 将 mul 函数地址赋给 p */
        case 4: p=div; c='/'; break; /* 将 div 函数地址赋给 p */
        case 5: p=mod; c='%'; break; /* 将 mod 函数地址赋给 p */
```

```
        }
        printf ("%d %c %d = %d\n", x, c, y, (*p)(x, y)); /* 调用 p 所指
          函数 */
        printf ("Continue? 1 YES; 0  NO\n");
        scanf ("%d", &x);
        if (! x)      break;
    }
}
```

3.4.6 局部变量和全局变量

在前面的各个例子中，变量都是在函数内部定义的，各个函数内部的变量可以同名而互不影响，这些变量称为内部变量，又称局部变量。函数的形式参数也属于局部变量。

C 程序中，还允许在函数外面定义变量，在函数外面定义的变量称为外部变量，又称全局变量。

全局变量与局部变量的区别在于它们的作用域。每个函数都能使用全局变量；而局部变量只能被定义它的函数使用，不能被其他函数使用。也就是说，全局变量对所有函数都是可见的，局部变量只对定义它的函数才是可见的。

全局变量和局部变量的另一个区别在于：全局变量被定义时若没有初始化，其值为 0；而局部变量被定义时若没有初始化，则它的值是不确定的。

在一个函数内部，当一个局部变量与一个全局变量同名时，全局变量不起作用，局部变量起作用。

例 32 分析以下程序：

```
int a=3, b=5;       /* 全局变量 a、b */
int max (int a, int b)/* 局部变量 a、b，全局变量 a、b 被屏蔽 */
{ int c;
  c=a>b? a: b;    /* a=8, b=5, c=8 */
  return c;
}

main ( )
{ int a=8;          /* 局部变量 a，全局变量 b=5，全局变量 a 被屏蔽 */
  printf ("%d", max (a, b));      /* 输出结果：8 */
}
```

局部与全局的概念是相对的。在一个函数内部，也有局部与全局之别。（可以在复合语句中定义变量，这些变量只在复合语句中有效）

例 33 分析以下程序：

```
main ( ) {
  int a=1, b=2, c=3;          /* 全局变量 a、b、c */
  +-a;                        /* a=2 */
  c      |b; /* b=3, c=3+3=6 */
  {
    int b=4, c;               /* 局部变量 b、c，全局变量 b、c 被屏蔽 */
    c=b*3;                    /* c=4*3=12 */
    a -  c;                   /* a=2+12=14 */
    printf ("first:%d,%d,%d\n", a, b, c);      /* 输出 14, 4, 12 */
    a|   c;                                    /* a=14+12=26 */
    printf ("second:%d,%d,%d\n", a, b, c);     /* 输出 26, 4, 12 */
  }
    printf ("third:%d,%d,%d\n", a, b, c);      /* 输出 26, 3, 6 */
  }
```

当一个函数要使用一个全局变量，而这个变量是在该函数的后面定义的，应该在函数中对该变量做如下声明，以向编译程序说明该变量是一个已被定义的外部变量。

extern 类型说明符 变量名；

当一个文件中的函数要使用另一个文件中定义的全局变量时，也必须用 extern 作外部变量声明。当然，这两个文件都是同一个程序的模块。

例 34 分析以下程序：

```
int max (int a, int b)            /* 局部变量 a、b */
{ int c;
  c=a>b? a : b;
  return c;
}

main ( )
{ extern int a, b;                /* 全局变量 a、b 的申明 */
  printf ("%d", max (a, b));
}
int a=13, b=5;                    /* 全局变量 a、b 的定义 */
```

全局变量提供了函数之间数据交换的途径。但是，由于各函数都能修改全局变量的值，就会产生隐患。一般不提倡使用全局变量。

3.5 C51的库函数

C51编译器提供了丰富的库函数，使用库函数大大提高了编程的效率，用户可以根据需要随时调用。每个库函数都在相应的头文件中给出了函数的原型，使用时只需在源程序的开头用编译预处理命令#include将相关的头文件包含进来即可。下面就一些常用的C51库函数分类做一些介绍。

3.5.1 字符函数库CTYPE.H

一组关于字符处理的函数。主要的函数原型和功能如下：

(1) extern bit isalpha (char)；

检查参数字符是否为英文字母，是则返回1，否则返回0。

(2) extern bit isalnum (char)；

检查参数字符是否为英文字母或数字字符，是则返回1，否则返回0。

(3) extern bit iscntrl (char)；

检查参数字符是否为控制字符，即ASCII值为0x00～0x1f或0x7f的字符，是则返回1，否则返回0。

(4) extern bit islower (char)；

检查参数字符是否为小写英文字母，是则返回1，否则返回0。

(5) extern bit isupper (char)；

检查参数字符是否为大写英文字母，是则返回1，否则返回0。

(6) extern bit isdigit (char)；

检查参数字符是否为数字字符，是则返回1，否则返回0。

(7) extern bit isxdigit (char)；

检查参数字符是否为16进制数字字符，是则返回1，否则返回0。

(8) extern char toint (char)；

将ASCII字符的0～9、a～f（大小写无关）转换为16进制数字。

(9) extern char toupper (char)；

将小写字母转换成大写字母，如果字符不在'a'～'z'之间，则不作转换直接返回该字符。

(10) extern char tolower (char)；

将大写字母转换成小写字母，如果字符不在'A'～'Z'之间，则不作转换直接返回该字符。

3.5.2 标准函数库STDLIB.H

(1) extern float atof (char *s)；

将字符串s转换成浮点数值并返回它。参数字符串必须包含与浮点数规定相符的数。

(2) extern long atol (char *s);

将字符串 s 转换成长整型数值并返回它。参数字符串必须包含与长整型数规定相符的数。

(3) extern int atoi (char *s);

将字符串 s 转换成整型数值并返回它。参数字符串必须包含与整型数规定相符的数。

(4) void *malloc (unsigned int size);

返回一块大小为 size 个字节的连续内存空间的指针。如返回值为 NULL，则无足够的内存空间可用。

(5) void free (void *p);

释放由 malloc 函数分配的存储器空间。

(6) void init_mempool (void *p, unsigned int size);

清零由 malloc 函数分配的存储器空间。

3.5.3 数学函数库 MATH.H

一组数学函数。

(1) extern int abs (int val);

extern char abs (char val);

extern float abs (float val);

extern long abs (long val);

计算并返回 val 的绝对值。这四个函数的区别在于参数和返回值的类型不同。

(2) extern float exp (float x);

返回以 e 为底的 x 的幂，即 e^x。

(3) extern float log (float x);

extern float log10 (float x);

log 返回 x 的自然对数，即 lnx；log10 返回以 10 为底的 x 的对数，即 $\log_{10}x$。

(4) extern float sqrt (float x);

返回 x 的正平方根，即 $\sqrt{x}$。

(5) extern float sin (float x);

extern float cos (float x);

extern float tan (float x);

sin 返回值为 sin (x)；cos 返回值为 cos (x)；tan 返回值为 tg (x)。

(6) extern float pow (float x, float y);

返回值为 x'。

3.5.4 绝对地址访问头文件 ABSACC.H

(1) #define CBYTE ((unsigned char *) 0x50000L)

#define DBYTE ((unsigned char *) 0x40000L)

#define PBYTE ((unsigned char *) 0x30000L)

#define XBYTE ((unsigned char *) 0x20000L)

用来对 8051 系列单片机的存储器空间进行绝对地址访问，以字节为单位寻址。

CBYTE 寻址 CODE 区；

DBYTE 寻址 DATA 区；

PBYTE 寻址 XDATA 的 00H～FFH 区域（用指令 MOVX @R0，A）；

XBYTE 寻址 XDATA 区（用指令 MOVX @DPTR，A）。

(2) #define CWORD ((unsigned int *) 0x50000L)

#define DWORD ((unsigned int *) 0x40000L)

#define PWORD ((unsigned int *) 0x30000L)

#define XWORD ((unsigned int *) 0x20000L)

与前面的宏定义相同，只是数据为双字节。

3.5.5 内部函数库 INTRINS.H

(1) unsigned char _crol_ (unsigned char val, unsigned char n);

unsigned int _irol_ (unsigned int val, unsigned char n);

unsigned long _lrol_ (unsigned long val, unsigned char n);

将变量 val 循环左移 n 位。

(2) unsigned char _cror_ (unsigned char val, unsigned char n);

unsigned int _iror_ (unsigned int val, unsigned char n);

unsigned long _lror_ (unsigned long val, unsigned char n);

将变量 val 循环右移 n 位。

(3) void _nop_ (void);

该函数产生一个 8051 单片机的 NOP 指令，用于延时一个机器周期。

如：

```
P10=1;
_nop_ ();        /*等待一个机器周期*/
P10=0;
```

(4) bit _testbit_ (bit x);

测试给定的位参数 x 是否为 1，若为 1，返回 1，同时将该位复位为 0；否则返回 0。

3.5.6 访问 SFR 和 SFR_bit 地址头文件 REGxxx.H

头文件 reg51.h、reg52.h 等文件中定义了 8051 单片机中的 SFR 寄存器名和相关的位变量名。

3.6 编程举例

循环队列是一种先入先出（FIFO）存储结构，在单片机应用程序中经常使用。队列需要队头指针 listhead、队尾指针 listtail、队列长度 listlen、队列空标志 listempty 和队列满 listfull 标志。队头指针 listhead 总指向队列中的第一个数据，队尾指针 listtail 总指向队列当前的空位置。队列可以用数组实现，数组长度为 listlen。listempty 标志若为 1，表示队列中没有数据，即队列为空；若为 0，表示队列有数据，即队列非空。listfull 标志若为 1，表示队列已满；若为 0，表示还能向队列写入数据，即队列非满。

初始时，listhead=0，listtail=0，listempty=1，listfull=0。需要定义两个函数 listwrite（）和 listread（），分别对队列进行读写操作。

listwrite（）函数的操作思路：

```
if（队列满）退出；
else {
    将数据写入 listtail 指向的数组单元；
    listtail++；
    if (listtail==listlen)    listtail=0；
    listempty=0；
    if (listhead==listtail)    listfull=1；
    }
```

listread（）函数的操作思路：

```
if（队列空）退出；
else {
    将 listtail 指向的数组单元的内容读出；
    listhead++；
    if (listhead==listlen) listhead=0；
    listfull=0；
    if (listtail==listhead) listempty=1；
    }
```

函数的实现：

```
#define listlen 10；
unsigned char list [listlen]；
```

```
char listwrite (char x)
{
    if (listfull) return 0;
    list [listtail] =x;
    listtail++;
    if (listhead==liselen) listtail=0;
    listempty=0;
    if (listhead==listtail) listfull=1;
    return 1;
}
char listread (char*x)
{
    if (listempty) return0;
    *x=list [listhead];
    listhead++;
    if (listtail==liselen) listhead=0;
    listfull=0;
    if (listtail==listhead) listempty=1;
    return 1;
}
```

第四章　MCS—51的功能部件

单片机是为工业控制设计的计算机，在工控系统中常使用定时器、计数器；在分布式控制系统中，各控制单元之间需要通信功能；为了快速响应人工干预、外部事件及迅速处理意外故障，计算机必须具有中断能力。MCS　51单片机在芯片内部集成了中断处理系统、定时/计数器和串行通信接口，增强了它在工业控制系统中的应用能力。

4.1　中断

4.1.1　中断的概念

现代的计算机都具有实时处理功能，能对外部发生的事件如人工干预、外部事件及意外故障做出及时的响应或处理，这是依靠它的中断系统来实现的。

在MCS—51应用系统中，经常需要处理如下问题。

1. 定时器问题

在温度控制系统中，需对受控对象的温度进行定时采样，两次采样之间的时间间隔是固定的，如每秒一次。在电机恒速控制系统中，需对受控电机的转速进行定时采样，两次采样之间的时间间隔也是固定的，如每秒两次。为了定时采样，就必须使用定时器。当CPU启动定时器后，就要等待定时器的定时超时标志、然后就进行采样，周而复始，循环不止。

2. 键盘按键问题

键盘是计算机系统操作者对系统进行参数设置和状态控制的常用设备，操作者何时对键盘进行操作是无法事先确定的。单片机应快速响应键盘操作。

3. 串行通信问题

一个单片机控制系统可能与另一个计算机系统有联系，它们之间的数据交换是通过异步串行通信接口RS-232C进行的。MCS—51单片机有一个串行通信控制器，当CPU将要发送的一个字节数据提交给串行通信控制器后，需要等串行通信控制器把这个字节数据发送完毕，才能发送下一个字节数据。这时，CPU要等待串行通信控制器的一个标志，表明串行通信控制器的发送缓冲器空闲，才能把下一个要发送字节数据提交给串行通信控制器。CPU除了发送数据之外，还要接收对方发送来的数据，而对方什么时候要发送数据是无法确定的。MCS　51单片机的串行通信控制器会自动处理数据接收，一旦接收到一个字节的数据，串行通信控制器会设置数据接收完成标志，CPU检测到该标志后，就从串行通信控制器

中将数据读出。

上述三个问题中，一个共性的问题是：CPU 需要对一个标志进行检测判断，以决定是否进行一项预定的工作（即执行一个特定的程序段）。对一个定时出现的或随机出现的标志进行检测判断，可以采用两种方法：查询和中断。

查询是指 CPU 在程序流程中循环判断标志的改变，如，启动定时器时，定时器的定时超时标志 TF 为 0，定时间隔到时，定时器将定时超时标志 TF 置为 1，程序中 CPU 用循环结构判断该标志是否为 1 等待定时结束：while（TF==0）；TF=0；调用采样函数，或者，在主函数的流程中按顺序判断各个标志的状态，以确定要做的工作，while（1）{if（TI）调用发送函数；if（RI）调用接受函数；if（keypress）调用按键处理函数；……}。这里，TF、TI、RI、keypress 分别为定时到标志、发送缓冲器空标志、接收缓冲器满标志和有键按下标志。

所谓中断是指单片机内部有一个中断管理系统，它对内部的定时器事件、串行通信的发送和接收事件及外部事件（如键盘按键动作）等进行自动的检测判断，当有某个事件产生时，中断管理系统会置位相应标志通知 CPU，请求 CPU 迅速去处理。CPU 检测到某个标志时，会停止当前正在处理的程序流程，转去处理所发生的事件（针对发生的事件，调用某一特定的函数，称为该事件的中断服务函数），处理完以后，再回到原来被中断的地方，继续执行原来的程序。这个过程称为中断。CPU 对中断标志的检测是在程序指令执行的周期中顺带进行的，不影响指令的连续执行。

查询和中断两种工作方式的特点，可以用电话的例子来说明：一个人有一部只有指示灯而没有铃的电话，为了不遗漏来电，在他工作的时候，必须时时查看电话的指示灯。这就是查询工作方式，时时查看指示灯势必会降低他的工作效率。另一个人的电话有铃，当有来电时，电话铃就会自动响起来，他会停下手中的工作去接电话，电话打完之后，他又继续他的工作。这就是中断工作方式。这种工作方式比查询工作方式的工作效率要高得多，而且能更快地处理引起中断的事件。

当有中断标志时，CPU 会调用一段特定的函数，称为中断服务函数，与程序中函数的调用是不同的。程序中的一般函数是由主函数或其他函数调用的，而中断服务函数不能被其他函数调用；一般函数的调用在程序中是固定的，而中断服务函数的执行完全是随机的。

MCS--51 单片机有一个中断管理系统，它可以自动处理内部定时器、内部串行通信设备的事件及外部事件的检测，当有某个事件发生时，它会自动置位中断标志向 CPU 发出警报，请求 CPU 对该事件进行处理。中断管理系统可以处理的事件称为中断源。

一般计算机系统允许有多个中断源，当几个中断源同时向 CPU 请求中断，要求为它们服务的时候，就存在 CPU 优先响应哪一个中断请求源的问题，一般根据

中断源（所发生的实时事件）的轻重缓急排队，优先处理最紧急事件的中断请求，于是规定每一个中断源都有自己的中断优先级别。

当 CPU 正在处理一个中断源请求时，又发生了另一个优先级比它高的中断请求，如果 CPU 能够暂时中止执行当前的中断服务程序，转而去处理优先级更高的中断请求，待处理完以后，再继续执行原来的低级中断处理程序，这样的过程称为中断嵌套，这样的中断系统称为多级中断系统。没有中断嵌套功能的中断系统称为单级中断系统。

4.1.2 MCS—51 的中断系统

MCS—51 系列中不同型号单片机的中断源数量是不同的（5～11 个），最典型的 8051 单片机有 5 个中断源，具有两个中断优先级，可以实现两级中断服务程序嵌套。每一个中断源可以编程为高优先级或低优先级中断，允许或禁止向 CPU 请求中断。与中断系统有关的特殊功能寄存器有中断允许寄存器 IE、中断优先级控制寄存器 IP、中断控制寄存器 TCON 和 SCON 中有关位。MCS 51 单片机基本的中断系统结构如图 4-1 所示。

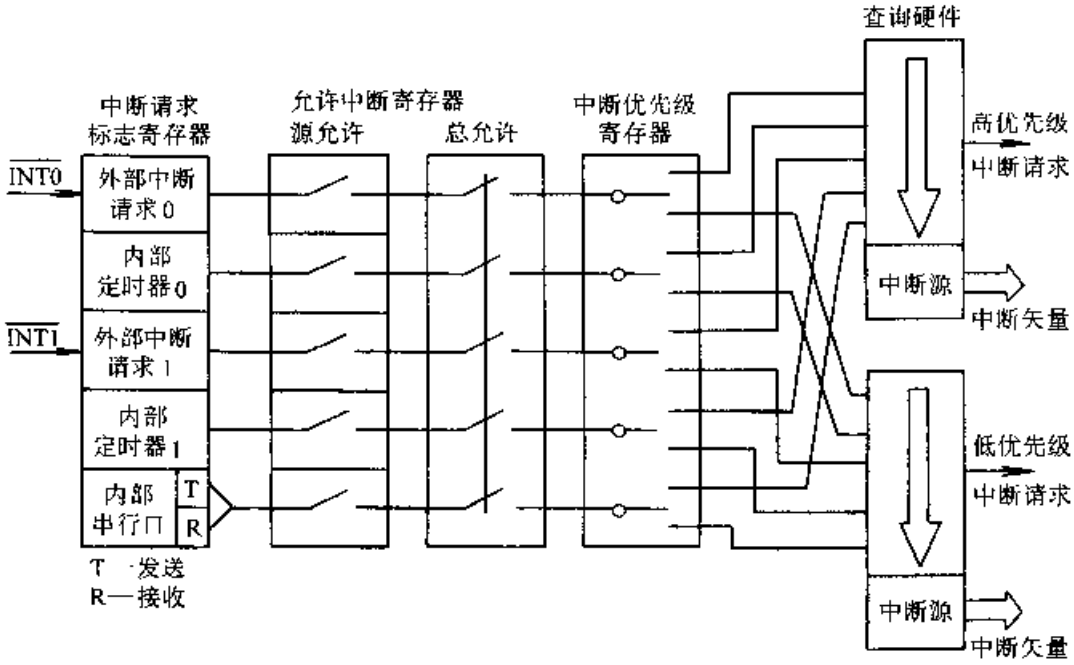


图 4-1 MCS 51 的中断系统

1. 中断源与中断标志

MCS- 51 中典型的 8051 单片机有 5 个中断源：两个外部中断是$\overline{\text{INT0}}$、$\overline{\text{INT1}}$（P3.2、P3.3）上输入的外部事件中断源，低电平或负跳变有效，在每个机器周期的 S5P2 状态采样，并置位 TCON 中的 IE0 和 IE1 中断请求标志位；三个内部的中断源，它们是定时器/计数器 T0、T1 的溢出中断源和串行口的发送接收中断，对 T0 和 T1 中断，当定时计数回'0'溢出时，由硬件自动置位 TCON 中的 TF0 或

TF1 中断请求标志位；对串行接收/发送中断，当完成一串行帧的接收/发送时，由硬件自动置位 SCON 中的中断请求标志位 TI（发送）或 RI（接收），必须由用户在中断服务程序中复位 T1 或 RI；8052 型单片机增加了一个定时器 T2 中断。

中断控制寄存器 TCON 的各位（可为寻址）：

位：	D7	D6	D5	D4	D3	D2	D1	D0
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

IE0：外部中断 0 请求源（INT0，P3.2）标志。IE0=1，外部中断 0 正在向 CPU 请求中断，当 CPU 响应该中断时由硬件清'0'IE0（边沿触发方式）。

IT0：外部中断源 0 触发方式控制位。IT0=0，外部中断 0 程控为电平触发方式，当$\overline{\text{INT0}}$(P3.2)输入低电平时，置位 IE0。中断系统在每一个机器周期的 S5P2 采样$\overline{\text{INT0}}$的输入电平，当采样到低电平时，置'1'IE0。在 CPU 调用中断服务程序之前，$\overline{\text{INT0}}$上的电平必须保持为低，否则当中断系统检测到$\overline{\text{INT0}}$的输入为高电平时，就清除 IE0 标志。IT0=1，外部中断 0 程控为边沿触发方式，中断系统在每一个机器周期的 S5P2 采样$\overline{\text{INT0}}$（P3.2）的输入电平。如果相继的两次采样，一个周期中采样到$\overline{\text{INT0}}$为高电平，接着的下个周期中采样到$\overline{\text{INT0}}$为低电平，则置'1'IE0。IE0 为 1，表示外部中断 0 正在向 CPU 申请中断，直到该中断被 CPU 响应时，才由硬件清'0'IE0。因为每个机器周期采样一次外部中断输入电平，因此，采用边沿触发方式时，外部中断源输入的高电平和低电平时间必须保持 12 个振荡周期以上，才能保证 CPU 检测到高到低的负跳变。

IE1：外部中断 1 请求源（$\overline{\text{INT1}}$，P3.3）标志。IE1=1 外部中断 1 向 CPU 请求中断，当 CPU 响应外部中断时，由硬件清'0'IE1（边沿触发方式）。

IT1：外部中断 1 触发方式控制位。IT1=0，外部中断 1 程控为电平触发方式，IT1=1，外部中断 1 为边沿触发方式。其功能和 IT0 类似。

TR0：定时/计数器 T0 运行控制位。

TF0：定时/计数器 T0 溢出中断标志位，CPU 执行中断服务程序时由硬件复位。

TR1：定时/计数器 T1 运行控制位。

TF1：定时/计数器 T1 溢出中断标志位，CPU 执行中断服务程序时由硬件复位。

串行口中断：串行口的接收中断标志 RI（SCON.0）和发送中断标志 TI（SCON.1）逻辑或以后作为内部的一个中断源。串行口发送完一个字符后由内部硬件置位发送中断标志 TI，接收到一个字符后也由内部硬件置位接收中断标志 RI。应该注意，CPU 响应串行口的中断时，并不清'0'TI 和 RI 中断标志，TI 和 RI

必须由软件清 0（中断服务程序中必须有清 T1、R1 的指令）。

系统复位时，TCON 的各位均被清 0。

2. 中断控制

(1) 中断允许寄存器 IE

MCS 51 的 CPU 对中断源的开放或屏蔽，即每一个中断源是否被允许中断，是由内部的中断允许寄存器 IE（IE 为特殊功能寄存器，它的字节地址 A8H，可位寻址）控制的，其格式如下：

位：	D7	D6	D5	D4	D3	D2	D1	D0
	EA	—	—	ES	ET1	EX1	ET0	EX0

EA：CPU 的中断开放标志。EA=1，CPU 开放中断；EA=0，CPU 屏蔽所有的中断申请。

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

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

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

ET0：T0 的溢出中断允许位。ET0=1，允许 T0 中断；ET0=0，禁止 T0 中断。

EX0：外部中断 0 中断允许位。EX0=1，允许中断；EX0=0，禁止中断。

(2) 中断优先级控制

MCS 51 有两个中断优先级，每一中断请求源可编程为高优先级中断或低优先级中断，实现二级中断嵌套。一个正在被执行的低优先级中断服务程序能被高优先级中断所中断，但不能被另一个同级的或低优先级中断源所中断。若 CPU 正在执行高优先级的中断服务程序，则不能被任何中断源所中断，一直执行到结束，遇到返回指令 RETI，返回主程序后再执行一条指令才能响应新的中断源申请。为了实现上述功能，MCS—51 的中断系统有两个不可寻址的优先级状态触发器，一个指出 CPU 是否正在执行高优先级中断服务程序，另一个指出 CPU 是否正在执行低级中断服务程序。这两个触发器的'1'状态分别屏蔽所有的中断申请和同一优先级的其他中断源申请。另外，MCS—51 的片内有一个中断优先级寄存器 IP（IP 为特殊功能寄存器，它的字节地址为 B8H，可位寻址），其格式如下：

位：	D7	D6	D5	D4	D3	D2	D1	D0
	—	—	—	PS	PT1	PX1	PT0	PX0

PS：串行口中断优先级控制位。PS=1，串行口中断定义为高优先级中断；PS

=0，串行口中断定义为低优先级中断。

PT1：定时器 T1 中断优先级控制位。PT1=1，定时器 T1 中断定义为高优先级中断；PT1=0，定时器 T1 中断定义为低优先级中断。

PX1：外部中断 1 中断优先级控制位。PX1=1，外部中断 1 中断定义为高优先级中断；PX1=0，外部中断 1 中断定义为低优先级中断。

PT0：定时器 T0 中断优先级控制位。PT0=1，定时器 T0 中断定义为高优先级中断；PT0=0，定时器 T0 中断定义为低优先级中断。

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

在 CPU 接收到同样优先级的几个中断请求源时，一个内部的硬件查询序列确定优先服务于哪一个中断申请，这样在同一个优先级里，由查询序列确定了优先级结构，其优先级别排列如下：

中　断　源	中断优先级
外部中断 0	最　高
定时器 T0 中断	↓
外部中断 1	
定时器 T1 中断	
串行口中断	最低

MCS—51 复位以后，特殊功能寄存器 IE、IP 的内容均为 0，由初始化程序对 IE、IP 编程，以开放中央处理器 CPU 中断、允许某些中断源中断和改变中断的优先级。整个中断系统结构如图 4-1 所示。

3. 中断响应过程

MCS—51 的 CPU 在每一个机器周期顺序检查每一个中断源。在机器周期的 S6 采样并按优先级处理所有被激活的中断请求，如果没有被下述条件所阻止，将在下一个机器周期的状态 S1 响应激活了的最高级中断请求。

- CPU 正在处理相同的或更高优先级的中断。
- 现行的机器周期不是所执行指令的最后一个机器周期。
- 正在执行的指令是中断返回指令（RETI）或者是对 IE、IP 的写操作指令（执行这些指令后至少再执行一条指令才会响应中断）。

如果上述条件中有一个存在，CPU 将丢弃中断查询的结果；若一个条件也不存在，将在紧接着的下一个机器周期执行中断服务程序。

处理机响应中断时，先置位相应的优先级状态触发器（该触发器指出 CPU 开始处理的中断优先级别），然后执行一条硬件子程序调用，清 0 中断请求源申请标志（TI 和 RI 除外）。接着把程序计数部 PC 的内容压入堆栈（但不保护 PSW），将被响应的中断服务程序的入口地址送程序计数器 PC，各中断源服务程序的入口地

址为：

中　断　源	入 口 地 址
外部中断 0	0003H
定时器 T0	000BH
外部中断 1	0013H
定时器 T1	001BH
串行口中断	0023H

通常在中断入口安排一条跳转指令，以转移到用户设计的中断处理程序入口。

CPU 执行中断处理程序一直到 RETI 指令为止。RETI 指令是表示中断服务程序的结束，CPU 执行完这条指令后，清 0 响应中断时所置位的优先级状态触发器，然后从堆栈中弹出顶上的两个字节到程序计数器 PC，CPU 从原来打断处重新执行被中断的程序。由此可见，用户的中断服务程序末尾必须安排一条返回指令 RETI，CPU 现场的保护和恢复必须由用户的中断服务程序实现。

4. 外部中断响应时间

$\overline{\text{INT0}}$和$\overline{\text{INT1}}$电平在每一个机器周期的 S5P2 被采样并锁存到 IE0、IE2 中，这个新置入的 IE0、IE1 状态等到下一个机器周期才被查询电路查询到。如果中断被激活，并且满足响应条件，CPU 接着执行一条硬件子程序调用指令以转到相应的服务程序入口，该调用指令本身需两个机器周期。这样，在产生外部中断请求到开始执行中断服务程序的第一条指令之间，最少需要三个完整的机器周期。

如果中断请求被前面列出的三个条件之一所阻止，则需要更长的响应时间。如果已经在处理同级或更高级中断，额外的等待时间明显地取决于别的中断服务程序的处理过程。当没有处理同级或更高级中断时，如果正在处理的指令没有执行到最后的机器周期，所需的额外等待时间不会多于 3 个机器周期，因为最长的指令（乘法指令 MUL 和除法指令 DIV）也只有 4 个机器周期，如果正在执行的指令为 IE、IP 的指令，额外的等待时间不会多于 5 个机器周期（最多需一个周期完成正在处理的指令，完成下一条指令（设为 MUL 或 DIV）4 个机器周期）。这样，在一个单一中断优先级的系统里，外部中断响应时间总是在 3～8 个机器周期之间。

4.1.3　外部中断触发方式选择

1. 电平触发方式

若外部中断定为电平触发方式，外部引脚中断输入必须有效（保持低电平），直到 CPU 实际响应该中断时为止，同时在中断服务程序返回之前，外部中断输入必须无效（高电平），否则 CPU 返回后会再次引起中断。所以电平触发方式适合于外部中断输入以低电平输入的、而且中断服务程序能清除外部中断输入请求信号的情况。在用户系统中，可将中断输入信号经一个 D 触发器接入，并使 D 触发器的 D 端接地，当外部中断请求的正脉冲信号出现在 D 触发器的 CLK 端时，D 触

发器的Q端产生负电平，$\overline{\text{INTx}}$有效，发出中断请求，CPU执行中断服务程序时，利用一根口线，如P1.0，输出一负电平脉冲使D触发器置位，撤消中断请求。

2. 边沿触发方式

外部中断若定义为边沿触发方式，外部中断申请触发器能锁存外部中断输入线上的负跳变，即使CPU暂时不能响应，中断申请标志也不会丢失。在这种方式里，如果相继连续两次采样，一个周期采样到外部中断输入为高电平，下个周期采样到低电平，则置位中断申请触发器，直到CPU响应此中断时才清0。这样不会丢失中断，但输入的脉冲宽度至少保持12个时钟周期（若晶振频率为6MHz，即2μs）才能被CPU采样到。外部中断的边沿触发方式适合于以脉冲形式输入的外部输入请求，如ADC0809的A/D转换结果的标志信号EOC为正脉冲，取反后连到8031的$\overline{\text{INT0}}$，就可以中断方式 读取A/D的转换结果。

4.1.4 中断服务程序及例程

使用MCS—51的中断，要为使用到的中断源编写中断服务程序。C51为中断服务程序的编写提供了方便的方法。C51的中断服务程序是一种特殊的函数，它的说明形式为：

```
void  函数名（void）  interrupt n using m
{  函数体语句  }
```

这里，interrupt和using是为编写C51中断服务程序而引入的关键字，interrupt表示该函数是一个中断服务函数，interrupt后的整数n表示该中断服务函数是对应哪一个中断源。每个中断源都有系统指定的中断编号：

中 断 源	中断编号
外部中断0	0
定时器T0	1
外部中断1	2
定时器T1	3
串行口中断	4
定时器T2	5

using指定该中断服务程序要使用的工作寄存器组号，m为0～3。

关键字interrupt和usiug只能用于中断服务函数的说明而不能用于其他函数。

若不使用关键字usiug，则编译系统会将当前工作寄存器组的8个寄存器都压入堆栈。

程序中的任何函数都不能调用中断服务函数，中断服务函数是由系统调用

的。

例 1 INT0端口接一开关，P1.0 接一发光二极管。开关闭合（接地）时，发光二极管改变一次状态。

```
#include " reg51.h"
#include " intrins. h"
void delay (void) {
    int a=5000;
    while (a--)_nop_ ( );    /* INTRINS.H 中说明的内部函数 */
}
void int0_srv (void) interrupt 0 using 1
{
    delay ( );
    if (INT0==0)
    {P10=! P10; while (INT0==0);}
}
void main ( ) {
    P10=0;
    EA=1;
    EX0=1; while (1);
}
```

4.2 定时/计数器

由于在大多数的微机应用系统中都要使用定时/计数器，所以几乎所有单片机内部都集成有定时/计数器。MCS—51 系列单片机 8051 系列有两个十六位定时/计数器 T0、T1，8052 系列有三个十六位定时/计数器 T0、T1 和 T2。它们都可以用作定时器或外部事件计数器，并有 4 种工作方式。

MCS—51 系列单片机的定时/计数器有几个相关的特殊功能寄存器：方式控制寄存器 TMOD，加计数寄存器高八位 TH0、TH1，加计数寄存器低八位 TL0、TL1；定时/计数到标志位 TF0、TF1(TCON)；定时/计数器启停控制位 TR0、TR1(TCON)；定时/计数器中断允许位 ET0、ET1 (IE)；定时/计数器中断优先级设定位 PT0、PT1 (IP)。

4.2.1 定时/计数器方式控制寄存器

定时/计数器的工作方式由方式寄存器 TMOD 的各位控制，TMOD 的格式为：

位：

D7	D6	D5	D4	D3	D2	D1	D0
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
T1 方式字段 ←						→ T0 方式字段	

TMOD 的低四位为 T0 的方式字，高四位为 T1 的方式字。TMOD 不能位寻址，必须整体赋值。TMOD 各位的含义如下。

1. 工作方式选择位 M1、M0

M1、M0 的状态决定定时器的工作方式：

M1	M0	功能说明
0	0	方式 0，为 13 位的定时/计数器
0	1	方式 1，为 16 位的定时/计数器
1	0	方式 2，为常数自动重装入的 8 位定时/计数器
1	1	方式 3，T0 在该方式时成为两个 8 位定时/计数器，T1 在该方式时停止计数。

2. 定时和外部事件计数方式选择位 $C/\overline{T}$

$C/\overline{T}=0$ 为定时器方式。在定时方式中，以振荡器输出时钟脉冲的十二分频信号（即机器周期）作为计数信号，也就是每一个机器周期定时器加'1'，若晶振为 12MHz，则定时器计数频率为 1MHz，计数的脉冲周期为 1μs。定时器从初值开始加'1'计数直至定时器溢出所需的时间是固定的，所以称为定时方式。

$C/\overline{T}=1$ 为外部事件计数器方式，这种方式采用外部引脚（T0 为 P3.4、T1 为 P3.5）上的输入脉冲作为计数脉冲。内部硬件在每个机器周期的 S5P2 采样外部引脚的状态，当一个机器周期采样到高电平，接着的下一个机器周期采样到低电平时计数器加'1'，也就是外部输入电平发生负跳变时加'1'。外部事件计数时最高计数频率为晶振频率的 1/24，外部输入脉冲高电平和低电平时间必须在一个机器周期以上。对外部输入脉冲计数的目的通常是为了测试脉冲的周期、频率或对输入的脉冲数进行累加。

3. 门控位 GATE

GATE 为 1 时，定时器的计数受定时器运行控制位和外部引脚输入电平的控制（TR0 和$\overline{INT0}$控制 T0 的运行，TR1 和$\overline{INT1}$控制 T1 的运行），GATE 为 0 时定时器计数不受外部引脚输入电平的控制，而只受定时器运行控制位（TR0、TR1）的控制。

4.2.2 定时器运行控制位

在特殊功能寄存器 TCON 中存放着定时器的运行控制位和溢出标志位。

1. 定时器 T0 运行控制位 TR0

TR0（TCON.4）由软件置位和清零。当门控位 GATE 为 0 时，T0 的计数仅由 TR0 控制，T0 为 1 时允许 T0 计数，TR0 为 0 时禁止 T0 计数，这时，定时器仅由软件控制。门控位 GATE 为 1 时，仅当 TR0 等于 1 且$\overline{INT0}$（P3.2）的输入信号为高电平时 T0 才计数，TR0 为 0 或$\overline{INT0}$输入低电平时都禁止 T0 计数，这时，置 TR0 为 1，则定时器仅由引脚信号$\overline{INT0}$的状态控制启停，因而是硬件控制的。用 TR0 和$\overline{INT0}$一起控制定时器的启停，则为软、硬件配合控制。

2. 定时器 T1 运行控制位 TR1

TR1 由软件置位和清零。当门控位 GATE 为 0 时，T1 的计数仅由 TR1 控制，TR1 为'1'时允许 T1 计数，TR1 为'0'时禁止 T1 计数。门控位 GATE 为'1'时，仅当 TR1 为'1'且$\overline{INT1}$（P3.3）输入为高电平时才允许 T1 计数，TR1 为 0 或输入低电平都将禁止 T1 计数。

4.2.3 定时/计数器的工作方式

MCS -51 的定时器有方式 0、方式 1、方式 2 和方式 3 这 4 种工作方式。下面对各种工作方式的定时器结构和功能加以详细讨论。

1. 方式 0

当 M1M0 为 00 时，定时器工作于方式 0。定时器 T1 方式 0 的结构框图如图 4-2 所示。方式 0 为 13 位的计数器，由 TL1 的低 5 位和 TH1 的 8 位组成，TL1 低 5 位计数溢出时向 TH1 进位，TH1 计数溢出时置位溢出标志 TF1。若 T1 工作于定时方式，计数初值为 a，晶振频率为 12MHz，则 T1 从初值计数到溢出的定时时间为 $t=(2^{13}-a)\ \mu s$。

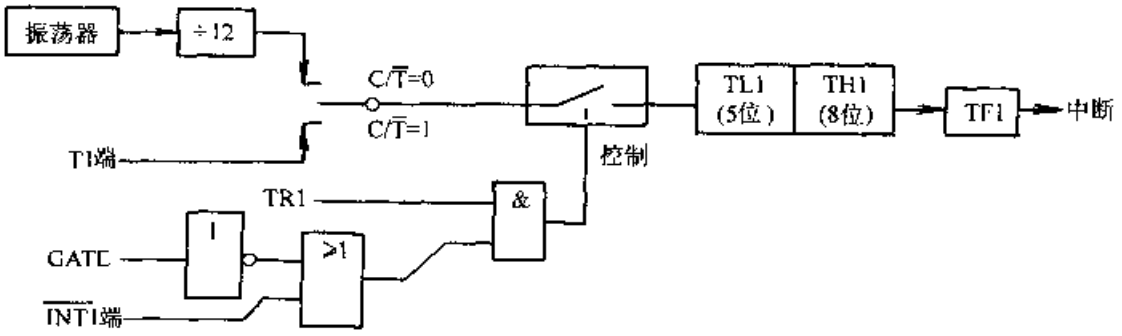


图 4-2 定时/计数器工作方式 0 的结构框图

在图 4-2 中的 T1 计数脉冲控制电路中，有一个方式电子开关和计数控制电子开关。$C/\overline{T}=0$ 时，方式电子开关打在上面，以振荡器的十二分频信号作为 T1 的计数信号，$C/\overline{T}=1$ 时，方式电子开关打在下面，此时以 T1（P3.5）引脚上的输入脉冲作为 T1 的计数脉冲。当 GATE 为 0 时，只要 TR1 为'1'，计数控制开关的控制端即为高电平，使开关闭合，计数脉冲加到 T1，允许 T1 计数。当 GATE 为'1'时，仅当 TR1 为'1'且$\overline{INT1}$脚上输入高电平，控制端才为高电平，才使控制开

关闭合，允许 T1 计数，TR1 为'0'或$\overline{\text{INT1}}$输入低电平都使控制开关断开，禁止 T1 计数。

2. 方式 1

当 M1M0 为 01 时，定时器工作于方式 1。方式 1 和方式 0 的差别仅仅在于计数器的位数不同，方式 1 为 16 位的定时器/计数器。定时器 T1 工作于方式 1 的逻辑结构框图如图 4-3 所示。T1 工作于方式 1 时，由 TH1 作为高 8 位，TL1 作为低 8 位，构成一个十六位的计数器。若 T1 工作于定时方式 1，计数初值为 a，晶振频率为 12MHz，则 T1 从计数初值计数到溢出的定时时间为 $t=(2^{16}-a)\ \mu s$。

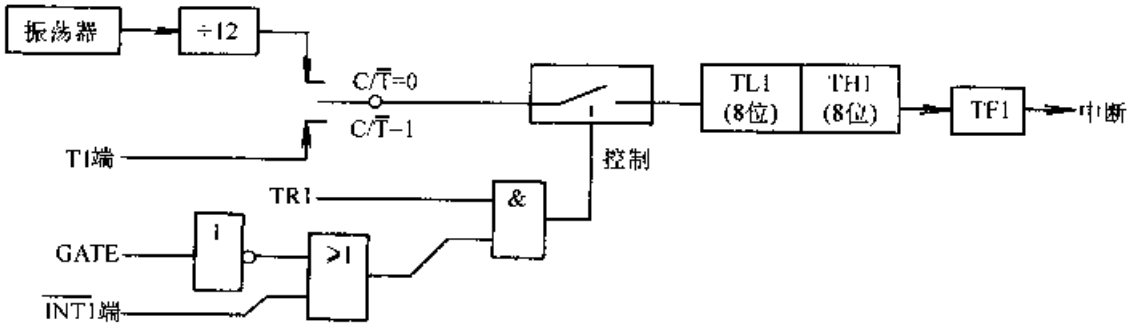


图 4-3　定时/计数器工作方式 1 的结构框图

3. 方式 2

M1M0 为 10 时，定时器/计数器工作于方式 2，方式 2 为自动恢复初值的 8 位计数器。定时器 T1 工作于方式 2 时的逻辑结构如图 4-4 所示。T1 工作于方式 2 时，TL1 作为 8 位计数器，TH1 作为计数初值寄存器。当 TL1 计数溢出时，一方面置'1'溢出标志 TF1，同时打开三态门，将 TH1 中的计数初值送至 TL1，使 TL1 从初值开始重新加'1'计数。若 T1 工作于定时方式 2 时，计数初值为 a，晶振频率为 12MHz 时，则定时时间为 $t=(2^{8}-a)\ \mu s$。

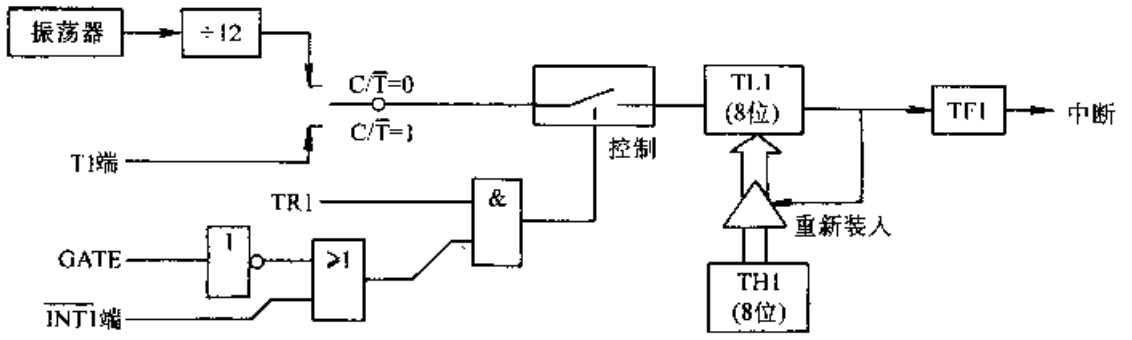


图 4-4　定时/计数器工作方式 2 的结构框图

上面以 T1 为例，说明了定时器/计数器方式 0、1、2 的工作原理，T0 和 T1 的这 3 种方式是完全相同的。

4. 方式 3

若 T1 设置为工作方式 3 时，则使 T1 停止计数。T0 方式字段中置 M1M0 为 11 时，T0 被设置为方式 3，此时 T0 的逻辑结构如图 4-5 所示，T0 分为两个独立

的 8 位计数器 TL0 和 TH0。TL0 使用 T0 的所有状态控制位、GATE、TR0、$\overline{INT0}$ (P3.2)、T0 (P3.4)、TF0 等，TL0 可以作为 8 位定时器或外部事件计数器。TL0 计数溢出时置位溢出标志 TF0，TL0 计数初值必须由软件每次设定。

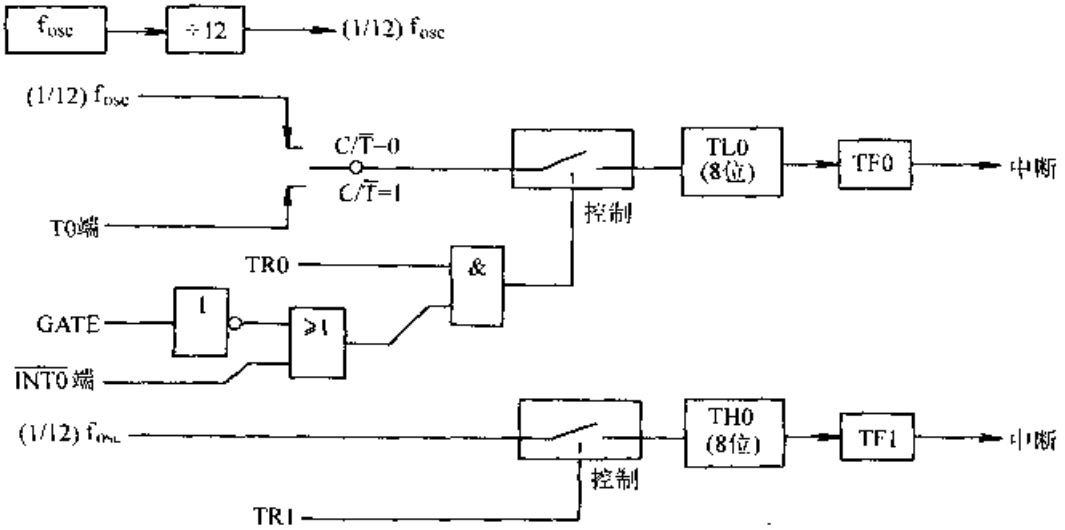


图 4-5　定时/计数器工作方式 3 的结构框图

TH0 被固定为一个 8 位定时器方式，并使用 T1 的状态控制位 TR1、TF1。TR1 为'1'时，允许 TH0 计数，当 TH0 计数溢出时，置'1'溢出标志 TF1。一般情况下，只有当 T1 用于串行口的波特率发生器时，T0 才在需要时用于方式 3，以增加一个计数器。这时 T1 的启停也受 TR1 控制，当 T1 计数溢出时不置位 TF1。

4.2.4　定时/计数器应用举例

使用 MCS—51 单片机的定时/计数器的步骤是：

1）设定 TMOD，即确定定时/计数器的工作状态：是用作定时器还是计数器，定时/计数器的工作方式字，定时/计数器的控制方式。如 T1 用于定时器，方式 1，T0 用于计数器，方式 2，均用软件控制。则 TMOD 的值应为：0001 0110，即 0x16。

2）设置合适的计数初值，以产生期望的定时间隔。由于定时/计数器在方式 0、方式 1 和方式 2 时的最大计数间隔取决于使用的晶振频率 f_{osc}（如下表所示），所以当需要的定时间隔较大时，要采用适当的方法，即将定时间隔分段处理。

晶振频率/MHz	6	8	10	12
方式 0	16.384ms	12.288ms	9.83ms	8.192ms
方式 1	131.072ms	98.304ms	78.643ms	65.536ms
方式 2	0.512ms	0.384ms	0.307ms	0.256ms

计数初值的计算方法如下：设晶振频率为 f_{osc}，则定时/计数器计数频率为 f_{osc}/

12，定时/计数器的计数总次数 T_all 在方式 0、方式 1 和方式 2 时分别为 $2^{13}=8192$、$2^{16}=65536$ 和 $2^8=256$，定时间隔为 T，计数初值为 a，则有

$$T=12\times(T\_all-a)/f_{osc}$$

$$a=T\_all-T\times f_{osc}/12$$

$$a=-T\times f_{osc}/12$$

将计数初值 a 分别赋给加 1 计数器 TH0、TL0 或 TH1、TL1：

```
TH0=a/256;
TL0=a%256;
```

3）确定定时/计数器工作于查询方式还是中断方式，若工作于中断方式，则在初始化时开放定时/计数器的中断及总中断：

```
ET0=1;
EA=1;
```

还需要编写中断服务函数：

```
void T0_srv (void) interrupt 1 using 1
 {
    TL0=a%256;
    TH0=a/256;
    中断服务程序段
}
```

4）启动定时器：TR0（TR1）=1。

例 2 从 P1.0 输出方波信号，周期为 50ms。

采用定时/计数器 T0 用于定时器，定时间隔为 25ms，软件控制，方式 1，中断方式。设 $f_{osc}=6MHz$。

定时计数初值为：

$$a=-0.025*6000000/12$$

$$=-12500$$

```
#include " reg51. h"
void main ( ) {
    TMOD=0x01;
    TH0=-12500/256;
    TL0=-12500%256;
    ET0=1;
    EA=1;
    TR0=1;
    While (1);
```

```
}
void T0_srv (void) interrupt 1 using 1
{
    TL0=-12500%256;
    TH0=-12500/256;
    P10=! P10;
}
```

例 3 交通灯控制问题。

设东西方向和南北方向均有红、黄、绿灯，控制规则为：南北绿灯亮 10s，黄灯亮 3s；绿灯亮 8s，黄灯亮 3s。

端口分配：东西方向 P1.0 红灯，P1.1 黄灯，P1.2 绿灯

南北方向 P1.3 红灯，P1.4 黄灯，P1.5 绿灯

使用定时/计数器 T0 作定时器，设 $f_{osc}=6MHz$，定时间隔为 0.1s。定时常数为

$$a=-0.1*6000000/12=-50000$$

由于定时/计数器无法定时 1s，故将 1s 定时分为 10 段，每段 0.1s，在定时中断服务函数中使用一个计数变量．当计数变量达到 10 时，就完成了 1s 的定时，置标志 flag。

```
#include " reg51. h"
char count;        /*计数变量*/
bit flag;
void T0_srv (void) interrupt 1 using 1
{
    TL0=-50000%256;
    TH0=-50000/256;
    count++;
    if (count==10)
     {
        count=0;
        flag=1;
    }
}
void main ( )
{
    char sum=0;
```

```
TMOD=0x01;
TH0=-50000/256;
TL0=-50000%256;
ET0=1;
EA=1;
TR0=1;
P1=0;
While (1)
{
  P10=1; P15=1;    /*东西：红；南北：绿*/
  while (sum<10)
   {
      while (! flag);
      flag=0;
      sum++;
  }
  sum=0;
  P15=0;
  P14=1;
  while (sum<3)
  {
    while (! flag);
    flag=0;
    sum++;
}
sum=0;
P14=0; P10=0; P12=1;
while (sum<8)
{
    while (! flag);
    flag=0;
    sum++;
}
sum=0;
P12=0; P11=1;
```

```
        while (sum<3)
        {
            while (! flag);
            flag=0;
            sum++;
        }
        sum=0;
        P11=0;
      }
    }
```

例 4 有救护车和消防车控制的交通控制问题。

在上例基础上，增加救护车和消防车控制情况，当有救护车和消防车通过时，两个方向均亮红灯，救护车和消防车通过后，恢复原来状态。

在上例端口使用基础上，增加一个手动控制开关，接在$\overline{\text{INT0}}$输入端，当开关接通时，保存 P1 口状态，使两个方向红灯亮，等开关断开后，恢复 P1 口状态。

上例程序中增加一个$\overline{\text{INT0}}$中断服务程序：

```
void int0_srv (void) interrupt 0 using 2
{
    char a;
    delay ( );
    if (! INT0)
    {
        a=P1;          /*保留 P1 状态*/
        P1=0;
        P10=1;
        P13=1;
        while (! INT0);          /*等待开关断开*/
        P1=a;                    /*恢复 P1 状态*/
    }
}
delay ( )
{
    int x=10000;
```

```
    while (x--);
}
```

主程序初始化时，增加设置$\overline{\text{INT0}}$为高优先级中断，并开放$\overline{\text{INT0}}$：

```
PX0=1;
EX0=1;
```

例 5 上面的交通灯控制程序中，并没有完全使用中断方式，面是混合使用了中断与查询方法。这里给出完全使用中断方式的中断服务程序流程。

设置位变量 dire 和 color，标识当前亮黄或绿灯的方向即是黄灯或绿灯。dire=0 为东西方向，dire=1 为南北方向；color=0 为绿灯亮，color=1 为黄灯亮。

定时器 T0 产生 0.1s 定时。

```
void T0_svr () interrupt 1 using 1
{
    ms++;
    if (ms==5) {
      s++;
      if (color) {   /*黄灯亮*/
        if (s==8) {   /*4 秒定时到*/
          if (dire) {   /*南北方向的黄灯*/
            红绿灯改变方向;
          }
          else 红绿灯改变方向;
          s=0; dire=! dire; color=0
        }
        else 黄灯闪烁处理;
      }
      else     /*绿灯亮*/
        if (s==12) {   /*6 秒定时到*/
          if (dire) 南北绿灯改黄灯;
          else 东西绿灯改黄灯;
          color=1; s=0;
        }
    }
}
```

例 6 波形展宽程序。

设 P3.4 输入低频的窄脉冲信号，要求在 P3.4 输入发生负跳变时，输出一个 500μs 的同步脉冲。若晶振频率为 6MHz，500μs 相当于 250 个机器周期。采用如图 4-6 所示的设计方法。P1.0 的初态为高电平，T0 选为方式 2 对外部事件计数，初值为 0FFH；这样 P3.4 输入发生负跳变时，T0 产生溢出，程序查询到 TF0 为 1 时，T0 改变为 500μs 的定时器的工作方式，并使 P1.0 输出低电平，T0 溢出后恢复 P1.0 高电平，T0 又工作于外部事件计数方式。

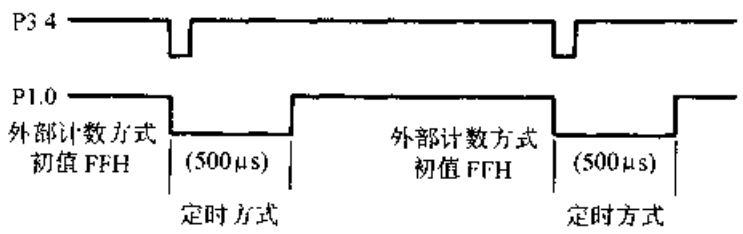


图 4-6 I/O 波形和 T0 方式变换

程序：

```
#include " reg51. h"
void main ( )
 {
  TMOD=0x06;        /*T0 计数器，方式 2*/
  TH0=-250;
  TL0=255;
  TR0=1;
  while (1)
   {
    while (! TF0);
    TF0=0;
    TMOD=0x02;        /*T0 定时器，方式 2*/
    P10=0;
  while (! TF0);
  TF0=0;
  P10=1;
  TMOD=0x06;
  TL0=255;
 }
}
```

4.3 串行通信接口

4.3.1 数据通信概述

1. 模拟通信与数字通信

在通信过程中，发送方要发送的数据是模拟的，并以模拟电信号的形式直接发送，如电话通信系统，则称为模拟通信系统。

如果将模拟数据经量化处理成数字电信号的形式进行传输，接收方将接收到的数字信号进行数字模拟变换得到模拟数据，这样的系统称为数字通信系统。

如要发送的数据就是数字形式的，并采用计算机进行通信处理的通信系统称为数据通信系统。

2. 并行通信与串行通信

数据通信可以采用并行通信或串行通信方式。

并行通信是指计算机之间的数据通道至少有 8 路，可以同时将一个字节的 8 位二进制代码发送到对方。计算机与打印机、CPU 与光驱、CPU 与硬盘之间的通信就是并行通信。

当通信距离较长时，要采用串行通信方式。串行通信用两根传输线进行数据传输，一次只能发送一位二进制位，一个字节的 8 位数据必须一位一位地顺序发送。

在串行通信系统中，根据数据流动方向可以分为三种方式：

单工通信，数据只能由一方发送到另一方，即数据的流动方向是固定的。

半双工通信，数据的流动方向是双向的，但每一时刻，数据只能在一个方向流动。

全双工通信，允许数据同时在两个方向流动，即通信双方的数据发送和接收是同时进行的。

在串行通信过程中，通信双方必须采用相同的通信速率，即每秒钟发送或接收的位数，称为波特率，如 110，300，600，1200，2400，4800，9600，19200b/s 等。发送的一位数据称为一个码元，每个码元的宽度是相等的。

3. 异步串行通信的帧结构

串行通信一次发送一个二进制位，一个字节有 8 个二进制位，在多个字节连续发送时，接收方必须从接收到的位序列中正确地区分出不同的字节，这需要采用合适的规则。异步串行通信采用如下的帧结构：

起始位＋8 位数据位＋停止位

或

起始位＋9 位数据位＋停止位

起始位为低电平，停止位为高电平。

第一种格式中，8 位数据可以是 8 位二进制数据，这时无法进行错误校验；也可以是字符的 ASCII 码，由于 ASCII 码只有 7 位，于是，最高位可以是一位奇偶校验位，以进行简单的错误校验。

第二种格式中，9 位数据的构成为 8 位数据位、1 位校验位或多机通信的地址数据标志位。

4. 错误校验

在串行通信过程中，传输的电信号会受到各种干扰而产生畸变，因而使接收方接收到错误的数据。为了能正确地进行数据传输，往往采用错误校验的方法使接收方能判断出接收到的数据中是否有误码。

在发送的数据信号中加入多余的信息位而达到错误校验的目的。

在异步串行通信系统中，奇偶校验是一种简单的错误校验编码。奇偶校验分为奇校验和偶校验。奇校验是指数据各位与校验位中'1'码的个数总为奇数个，偶校验是指数据各位与校验位中'1'码的个数总为偶数个。当传输 ASCII 字符数据时，可以利用每个字节的最高位作奇偶校验位。若发送的是 8 位的数据信息，则可采用 9 位的帧结构，使用第 9 位作奇偶校验位。

5. 异步串行通信过程和 UART

基本的计算机异步串行通信系统中，两台计算机之间通过三根信号线 TxD、RxD 和 GND 连接起来，TxD 与 GND 构成发送线路，RxD 与 GND 构成接收线路。一台计算机的 TxD、RxD 线分别与另一台计算机的 RxD、TxD 线相连。

发送过程：

CPU 将要发送的一个字节并行数据转换为串行数据序列，加一个起始位和一个停止位，根据需要加入校验位构成一个帧。按事先约定的通信波特率发送起始位，然后发送数据位和校验位，最后发送停止位。

接收过程：

在双方都没有数据发送时，通信线路处于空闲状态。异步串行通信系统在空闲时，接收线 RxD 为高电平。CPU 以 64 或 16 倍于波特率的频率对 RxD 进行采样，当采样到低电平后，延时半个码元时间再采样，若为低电平，确定接收到起始位。然后，延时一个码元时间接收一位数据，直至接收到停止位。判断无误后，将串行的字节数据转换为并行数据，交程序处理。在接收一位数据时，采用三次采样的多数判决方法。

由于在串行通信过程中的串并转换、并串转换、线路检测、采样判决、组帧、拆帧、定时发送和接收二进制位等操作需消耗 CPU 大量时间，以至 CPU 无法处理其他工作，因而开发出专用于处理异步串行通信发送和接收工作的芯片 UART（通用异步串行通信接收发送器）。CPU 只需将要发送的字节数据交给 UART，其

他发送工作由 UART 自动完成；UART 自动监测线路状态并完成数据接收工作，当接收到一个字节数据后，UART 会通知 CPU 以并行数据形式读取。采用 UART 后，CPU 的负担大大减轻了。

6. MCS—51 的串行通信接口

MCS—51 单片机内部集成有一个 UART，用于全双工方式的串行通信，可以同时发送、接收数据。它有一个发送缓冲器和一个发送移位寄存器，发送缓冲器是发送移位寄存器的并行输入，定时器 T1 作波特率发生器，其溢出信号作发送移位寄存器的移位时钟，发送移位寄存器的输出是 TxD 引脚。当 CPU 将一个字节的数据写入发送缓冲器后，UART 根据指定的帧结构将该数据存入发送移位寄存器，然后启动发送移位寄存器的输出操作，当把停止位发出之后，停止发送移位寄存器的输出工作，置位 TI，向 CPU 发中断请求。UART 还有一个接收缓冲器和一个接收移位寄存器，RxD 引脚是接收移位寄存器的输入，接收移位寄存器的并行输出是接收缓冲器的输入，波特率发生器的溢出信号作接收移位寄存器的移位时钟。UART 一直检测接收线 RxD 的状态，当接收完一帧数据后，将数据写入接收缓冲器，置位 RI 标志，向 CPU 发中断请求。UART 的发送缓冲器与接收缓冲器同名（SBUF），并共用一个地址号（99H），为区别操作对象，规定发送缓冲器只能写入，不能读出，接收缓冲器只能读出，不能写入。要发送的字节数据直接写入发送缓冲器，SBUF—a。当 UART 接收到数据后，CPU 从接收缓冲器中读取数据，a=SBUF。

串行接口有四种工作方式，有的工作方式时其波特率是可变的。用户可以用软件编程的方法在串行控制寄存器 SCON 中写入相应的控制字就可改变串行口的工作方式。

串行接口可以用查询方式处理串行接口，也可以用中断方式处理串行接口。

4.3.2 串行接口的控制寄存器

串行接口的控制寄存器有两个，串行控制寄存器 SCON 和能改变波特率的特殊功能寄存器 PCON。其作用如下：

（1）串行控制寄存器 SCON，字节地址 9BH，可位寻址。

SCON 用于确定串行通道的操作方式和控制串行通道的某些功能。也可用于发送和接收第九个数据位（TB8、RB8），并有接收和发送中断标志（RI 及 TI）位。SCON 各位的意义如下：

位：	D7	D6	D5	D4	D3	D2	D1	D0
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

这里 SM0、SM1 指定了串行通道的工作方式，若设振荡器频率为 f_{osc}，则规定如下：

SM0	SM1	方式	说　明	波特率
0	0	0	移位寄存器工作方式	$f_{osc}/12$
0	1	1	8 位数据位的 UART 工作方式	可变
1	0	2	9 位数据位的 UART 工作方式	$f_{osc}/64$ 或 $f_{osc}/32$
1	1	3	9 位数据位的 UART 工作方式	可变

SM2：在方式 2 和方式 3 时，进行主-从式多微机通信操作的控制位。SM2＝1，该机为从机，SM2＝0，该机为主机。

在方式 1 时，如 SM2＝1，则只有在接收到有效停止位时才能激发中断标志（RI＝1），如没有接收到有效停止位则 RI 仍然为 0。SM2＝0，则不进行帧的完整性检验，只把数据接收完就置位 RI。

在方式 0 时，SM2 应为 0。

REN：允许串行接口接收控制位。用软件置 REN＝1 时为允许接收状态，可启动串行口的接收器 RxD，开始接收数据。用软件复位（REN＝0）时，为禁止接收状态。

TR8：在方式 2 和方式 3 时，它是要发送的第九个数据位，按需要由软件进行置位或清零。例如可用作数据的奇偶校验位，或在多机通信中表示是地址帧/数据帧标志位（TB8＝1/0）。

RB8：在方式 2 和方式 3 时，它是接收到的第九位数据，作为奇偶校验位或地址帧/数据帧标志位。在方式 1 时，若 SM2＝0，则 RB8 是接收到的停止位，在方式 0 时，不使用 RB8。

TI：发送中断标志位。在方式 0 时，当串行发送数据字节第八位结束时，由内部硬件置位（TI＝1），向 CPU 申请发送中断。CPU 响应中断后，必须用软件清零，取消此中断标志。在其他方式时，它在停止位开始发送时由硬件置位。同样，必须用软件使其复位。

RI：接收中断标志位。在方式 0 时，串行接收到第八位数据时由内部硬件置位。在其他方式中，它在接收到停止位的中间时刻由硬件置位（例外情况见 SM2 说明），也必须用软件来复位。

SCON 的所有位在复位之后均为 0。

SCON 的内容可由 SCON＝a；指令来设定。如选择工作方式 0 并启动接收，可由指令 SCON＝0x10；来设定。应指出，当由指令改变 SCON 的内容时，改变的内容是在下一条指令的第一个周期的 S1P1 状态期间才锁存入 SFR 中并开始生效的。如果此时已经开始进行一次串行发送，那么 TR8 中送出去的仍是原有的值，而

不是新值。

当一幅行帧发送完成时，发送中断标志 TI 被置位，接着发生串行口中断，当接收完一幅行帧时，接受中断标志 RI 被置位，同样产生串行口中断，如 CPU 允许中断，则进入串行口中断服务程序。但 CPU 事先并不能分辨是由 TI 还是由 RI 引起的中断请求，而必须在中断服务程序中用位测试指令加以判别。两个中断标志位 TI 及 RI 均不能自动复位，故必须在中断服务程序设置清中断标志位指令，撤消中断请求状态，否则原先的中断标志位状态又将表示有中断请求。

(2) 特殊功能寄存器 PCON，字节地址 87H，没有位寻址。

格式	D7	D6	D5	D4	D3	D2	D1	D0
	SMOD	—	—	—	—	—	—	—

PCON 寄存器中的 D7 位为串行口波特率选择位。当用软件使 SMOD=1 时(如使用 PCON=0x80 指令)，则使方式 1、方式 2、方式 3 的波特率加倍。SMOD=0，各工作方式时，波特率不加倍。整机复位时，SMOD 为 0。

单片机串行通道内设有数据寄存器。在所有的串行方式中，在写 SBUF 信号的控制下把数据装入 9 位的发送移位寄存器，前面 8 位为数据字节，其最低位就是移位寄存器的移位输出位。根据不同的工作方式会自动将'1'或 TB8 的值装入移位寄存器的第九位，并进行发送。

其串行通道的接收寄存器是一个输入移位寄存器。在方式 0 时移位寄存器字长为 8 位，在其他方式时其字长为 9 位。当一个字符接收完毕，移位寄存器中的数据字节装入串行数据缓冲器 SBUF 中，其第九位则装入 SCON 寄存器的 RB8 位。如果 SM2 使得已接收的数据无效，则 RB8 位和 SBUF 缓冲器中的内容不变。

4.3.3 串行接口的四种工作方式

1. 方式 0——移位寄存器方式

串行口输出端可直接与移位寄存器相连，可用作扩展 I/O 口，或外接同步输入输出设备。

发送过程：当 CPU 将数据写入到发送缓冲器 SBUF 时，串行口即将 8 位数据以 $f_{osc}/12$ 的波特率由 RxD 引脚输出，同时由 TxD 引脚输出同步脉冲。字符发送完毕，置中断标志 TI 为'1'。

接收过程：控制字除置方式 0 外，还应置允许接收控制位 REN=1，清除 RI 中断标志。接收器启动后 RxD 为数据输入端，TxD 为同步信号输出端。接收器以 $f_{osc}/12$ 波特率采样 RxD 引脚输入的数据信息。当接收完 8 位数据时又重新置中断标志 RI=1。

方式 0 工作时必须使 SCON 控制字的 SM2 位(多机通信控制位)为'0'，从而不影响 TB8 和 RB8 位。由于波特率固定，无须用定时器提供。但以中断方式传送

数据时，CPU 响应中断并不会自动清除 TI、RI 标志，所以在中断服务程序中必须由指令清'0'，例如用 TI＝0 及 RI＝0 指令。

2. 其他方式 ——UART 方式

发送过程：CPU 执行一条将数据写入发送缓冲器 SBUF 的指令即可启动发送（如 SBUF＝a），发送完毕将中断标志位置'1'。

接收过程：UART 检测到起始位后，就按程序设定的帧格式接收一帧代码，并把此码的数据位变换成并行码送入接收缓冲寄存器中（在方式 1 时，把停止位、在方式 2、3 时把第九位数据送入 RB8），置接收中断标志 RI＝1。CPU 响应中断后必须在中断服务程序中由软件使 RI 复位。

方式 1、方式 2 与方式 3 的区别之一是：方式 1 其数据字是 8 位异步通信格式，串行口发送/接收共 10 位信息，第 1 位为起始位'0'，其后的 8 位是数据位，最后是停止位'1'；方式 2、3 其数据字为 9 位的异步通信。1 位起始位'0'，8 位数据位，第 9 位是可程控位'1'或'0'，最后是停止位'1'，共有 11 位信息。

方式 1、方式 2 与方式 3 的区别之二是：方式 1、3 的波特率是可变的，其波特率取决于定时器 1 的溢出率和特殊功能寄存器 PCON 中的 SMOD 位的值，方式 1、3 的波特率为

$$\text{波特率}=\frac{2^{\text{SMOD}}}{32}\times\text{（定时器 1 的溢出率）}$$

方式 2 的波特率为

$$\text{波特率}=\frac{2^{\text{SMOD}}}{64}\times\text{（振荡器频率）}$$

在方式 1 和方式 3 工作时，定时器 1 用作波特率发生器用，其中断应无效。因振荡器的频率是固定的，故方式 2 的波特率只能取为（振荡器频率）/32 或（振荡器频率）/64 两种，这靠在编程时使 PCON 寄存器的"SMOD"位写入'1'或'0'来决定。故也可称方式 2 的波特率是可编程的。

显然，方式 2 的波特率变化范围比方式 1、3 要小，这也是方式 2 和方式 3 的惟一区别。

对于输出方式，三者设置控制字的差别仅在于 SCON 寄存器选择方式的两个控制位不同。此外，在方式 2、3 中还可通过程控 TR8 位的方法，使其传送附加的第九位数据作为多机通信中的地址、数据标志位，或作为数据的奇偶校验位。

若以 TB8 作为奇偶校验位，处理方法为数据写入 SBUF 之前，先将数据的奇偶位写入 TB8（设工作寄存器区 2 的 R0 作为发送数据区地址指针）。

对输入方式而言，除选不同的方式控制外，均应使 REN＝1，允许串行接收。只有在最后的移位脉冲产生并同时满足下列两个条件时，才会产生接收数据装入 SBUF 和 RB8 及置位 RI 的信号。

对方式 1　　对方式 2、3

1) RI＝0　　1) RI＝0

2) SM2＝0 或接收到的停止位　　2) SM2＝0 或已接收到的第九个数据＝1

如果不满足上述条件，接收到的数据将不可避免地被丢失。由此可见，中断标志必须由用户在中断服务程序中设置清'0'指令。否则，将有可能产生另一次中断而造成混乱。表 4-1 给出了串行口工作方式一览表。

表 4-1　串行口工作方式

有关信号 \ 方式		方式 0	方式 1	方式 2、3
SM0　SM1		0　0	0　1	方式 2：10　方式 3：11
输出（发送）	TB8	没　用	没　用	
	发送位	8　位	10　位 （加上起始位和停止位）	
	数据	8　位	8　位	
	RxD	输出串行输出		
	TxD	输出同步脉冲	输出发送数据	
	波特率	$f_{osc}/12$	可变： $\frac{2^{SMOD}\times 定时器1溢出率}{32}$	方式 2： $\frac{2^{SMOD}\times 振荡频率}{64}$ 方式 3 同方式 1
	中断	发送完，置中断标志 TI＝1，响应后必须由软件清零		
输入（接收）	RB8	没　有	若 SM2＝0，接收停止位	接收发送的第九位数据
	REN	允许串行接收　REN＝1		
	SM2	SM2＝0	通常 SM2＝0	串行通信时置"1" 正常接收时置"0"
	接收位	8　位	10　位	11　位
	数据	8　位	8　位	8　位
	波特率	同发送情况		
	接收条件	无要求	(1) RI＝0 且 (2) SM2＝0 或停止位＝1	(1) RI＝0 且 (2) SM2＝0 或接收的第九个数据＝1
	中断	接收完，置中断标志 RI＝1，响应后必须由软件清零		
	RxD	串行数据输入端		
	TxD	同步信号输出端		

4.3.4 多处理机通信

如前所述，串行口控制寄存器 SCON 中的 SM2 位为串行口工作于方式 2 和方式 3 时进行多机通信的控制位。这种多机通信方式为一台主机，多台从机系统：主机发送的地址信息可被各从机接收，而主机发送的数据信息从机无法接收。从机间相互不能直接通信。

多机系统中，从机串行口必须在方式 2 或方式 3 下工作，而且应使 SM2 及 REN 控制位置'1'。从而使从机先处于只能接收地址帧信息（第 9 数据位为 1）的状态，当从机接收到主机发出的地址帧信息后，串行口可向 CPU 申请中断。设有一个由主机（由 8031 或其他具有串行接口的系统机）和三个 8031 组成的从机系统，如图 4-7 所示。从机系统由初始化程序（或相关处理程序）将串行口置成工作方式 2 或 3，SM2＝1，REN＝1，处于接收状态。当主机和某一从机通信时，主机应先发出一帧某从机的地址信息给各从机，接着才能送数据或命令。

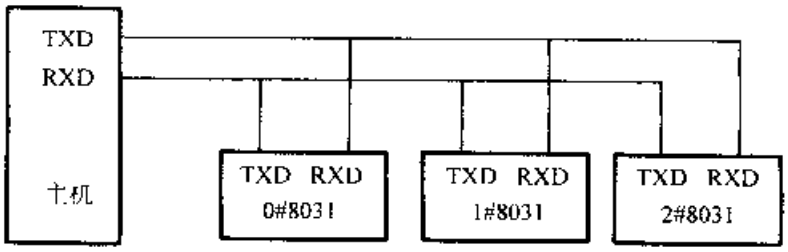


图 4-7 多机通信系统示意图

当各从机接收到主机发出的地址帧信息后，自动将第 9 数据位状态'1'送到 SCON 控制寄存器的 RB8 位，激发中断标志 RI＝1，分别中断 CPU。各 CPU 响应中断后均进入中断服务程序，在服务程序中把主机送来的地址号与本、从机的地址号相比较，若地址相符，则把本机之 SM2 置'0'，升格为主机，为接收主机接着发送来的数据帧（第 9 数据位为 0）作准备。而地址号不符的其他从机仍然维持 SM2＝1 状态，对主机以后发出的数据帧信息不予理采，不激发中断标志 RI＝0。从而实现了主从一对一通信（点-点通信）。从上述说明中看出，在多机通信时 SM2 控制位起着极为重要的作用。

4.3.5 波特率的设定

串行口工作于方式 0 时，波特率为振荡频率的 1/12，是固定不变的；工作于方式 2 时，波特率是可编程的，有两种波特率可选择，它取决于电源控制寄存器 PCON 中 SMOD 的值，当取 SMOD＝0 时，为振荡频率的 1/64，当取 SMOD＝1 时，为振荡频率 1/32；工作于方式 1 和 3 时，波特率是可变的，它是通过编程改变定时器 1 的溢出率来实现的。

若已知定时器 1 的溢出率，则工作在方式 1、3 时的串行口波特率可由下式求得

$$\text{波特率}=\frac{2^{SMOD}}{32}\times\text{（定时器 1 的溢出率）}$$

编程时还应注意，当定时器 T1 作为波特率发生器用时，应不允许定时器 T1 产生中断，即禁止中断，使中断允许寄存器的 IE.3（ET1）=0。定时器 1 作为波特率发生器应用时，最典型的用法是定时器 T1 工作在自动再装入时间常数的定时方式 2（即定时器的方式控制寄存器 TMOD 的高四位为 0010B 状态。定时器的控制寄存器 TCON 的 TCON.6（TR1）=1 启动定时器 T1）。这时溢出率取决于 TH1 中的自动重新再装入值。定时器 T1 的溢出率可由下式算出

$$\text{溢出率}=\text{振荡频率 } f_{osc}/(12\times[256-TH1])$$

将此值代入求波特率的算式，可求得

$$\text{串行口的波特率}=\frac{2^{SMOD}}{32}\times\frac{f_{osc}}{12\times[256-(TH1)]}$$

常用的串行口波特率以及与定时器 T1 各参数间的关系如表 4-2 所示。

表 4-2　常用波特率以及与定时器 1 各参数的关系

常用波特率	振荡频率 f_{osc} /MHz	SMOD	定时器 1		
			C/T	方式	重新装入值
方式 0　MAX：1MHz	12	×	×	×	×
方式 2　MAX：35Kb	12	1	×	×	×
方式 1、3：62.5Kb	12	1	0	2	FFH
19.2Kb	11.059	1	0	2	FDH
	11.059				
9.6Kb	11.059	0	0	2	FDH
4.8Kb	11.059	0	0	2	FAH
2.4Kb	11.059	0	0	2	F4H
1.2Kb	11.059	0	0	2	E8H
137.5b	6	0	0	2	1DH
110b	12	0	0	2	72H
110b		0	0	2	FEEBH

4.3.6　应用举例

例 7　将字符串"MCS—51 Serial Communication Bus."发送出去。

设 f_{osc}=11.0592MHz，波特率=2400，串行口工作于方式 1。

```
#include "reg51. h"
#include "string. h"
char s [] ="MCS—51" Serial Communication Bus.";
main ()
```

```
{
  char a, b=0;
  TMOD=0x20;
  SCON=0x50;                    /*SM0=SM2=0, SM1=1, REN=1*/
  TH1=0xF4;
  TL1=0xF4;
  a=strlen (s);
  for (; b<a; b++)
   {
    SBUF=s [b];
    while (! TI);
    TI=0;
   }
}
```

例 8 带奇偶校验的发送程序。

设 f_{osc}=11.0592MHz，波特率=2400，串行口工作于方式 1。

```
#include "reg51. h"
#include "string. h"
char s [] ="MCS-51" Serial Communication Bus.";
char bdata c;
sbit c7=c^7;
main ()
{
  char a, b=0;
  TMOD=0x20;
  SCON=0x50;                    /*SM0=SM2=0, SM1=1, REN=1*/
  TH1=0xF4;
  TL1=0xF4;
  a=strlen (s);
  for (; b<a; b++)
   {
    c=s [b];
    ACC=c;
    c7=P;                       /*奇偶位，偶校验*/
    SBUF=c;
```

```
    while (! TI);
    TI=0;
  }
}
```

例 9 将上例串行口的工作方式改为方式 3。

设 f_{osc}=11.0592MHz，波特率=2400，串行口工作于方式 3。

```
#include "reg51. h"
#include "string. h"
char s [] ="MCS-51 Serial Communication Bus.";
main ()
{
  char a, b=0;
  TMOD=0x20;
  SCON=0x50;              /*SM0=SM2=0, SM1=1, REN=1*/
  TH1=0xF4;
  TL1=0xF4;
  TR1=1;
  a=strlen (s);
  for (; b<a; b++)
   {
    ACC=s [b];
    TB8=P;                /*奇偶位，偶校验*/
    SBUF=s [b];
    while (! TI);
    TI=0;
  }
}
```

例 10 一多机通信系统，主机发送的数据格式为

报头	地址	命令	数据长度	数据	校验和	报尾

其中：

报头为　0xFF　0xFF　0xFF（至少三个）

地址　一个字节

命令　发送数据：　0xA5　要求数据：　0x5A

数据长度　命令字为 0xA5 时，没有此项与数据项

校验和　　数据长度与数据各字节的异或和

报尾　　0x0D　0x0A

编写从机的接收中断服务程序。

设 f_{osc}=11.0592MHz，波特率为 4800，串行口工作方式 3，TB8 作字节奇偶校验。奇偶校验采用偶校验。

主程序为：

```
#include "reg51. h"
#define ADREE 0x12
char s [15];                /*数据存储缓冲*/
bit rev. succ;              /*一帧数据接收成功标志*/
char rev .len, data_len; index;
char address=0x12;
bit want;
main ()
{
  TMOD=0x20;        /*T1 方式 2，波特率发生器*/
  SCON=0xD0;        /*串行口方式 3，REN=1*/
  TH1=0xFA;
  TL1=0xFA;
  TR1=1;
  ES=1;             /*允许串行口中断*/
  EA=1;
  while (1);
}
void sir_srv () interrupt 4 using 1
{
  char t, t1=0, t2;
  if (RI)
   {
    RI=0;
    t=SBUF;
ACC=t;
if (P! =RB8)
{
  rev_len=0; rev_succ=0; data_len=0; index=0;
```

```
}
else
{
  switch (rev_len)
   {
    case 0: if (t==0xFF) rev_len++; break;
    case 1: if (t==0xFF) rev len++;
         else rev_len=0;
         break;
    case 2: if (t==0xFF) rev_len++;
         else rev_len=0;
         break;
    case 3: if (t==0xFF) break;
         else if (t==address) rev_len++;
         else rev_len=0;
         break;
    case 4: if (t==0xA5) rev_len++;
         else if (t==0x5A) {want=1; rev_len=0;}
         else rev_len=0;
         break;
    case 5: data_len=t; rev_len++; break;
    case 6: s [index++] =t;
         if (index==data_len) rev_len++;
         break;
    case 7: for (t2=0; t2<data_len; t2++)
         t1=t1^s [t2];
    t1^=data_len;
    if (t1==t) rev_succ=1;
    rev_len=0; data_len=0; index=0;
      }
    }
  }
}
```

习题和思考题

1. MCS-51 T0、T1 的定时器和计数器方式的差别是什么？试举例说明这两种方式的用途。
2. 若晶振为 11.0592MHz，用 T0 产生 1μs 的定时，可以选择哪几种方式？分别写出定时器的方式字和计数初值。
3. 若晶振为 12MHz，如何用 T0 来测试 2～1kHz 之间的方波周期？又如何测试频率为 0.5MHz 左右的脉冲频率。
4. 若晶振为 11.0592MHz，串行口工作于方式 1，波特率为 4800，分别写出用 T1、T2 作为被持率发生器的方式字和计数初值。
5. 串行口方式 0 输出时能否外接多个 74LS164？若不可以说明其原因，若可以画出逻辑框图并说明数据输出方法。
6. MCS—51 的中断处理程序能否存储在 64K 字节程序存储器的任意区域？若可以则如何实现？
7. 在一个 8031 系统中，晶振为 12MHz，一个外部中断请求信号是一个宽度为 500ns 的负脉冲，则应用哪种触发方式？如何实现？
8. 若外部中断请求信号是一个低电平有效的信号，是否一定要选择电平触发方式？为什么？
9. 试设计一个测试 N 个脉冲平均周期的方案，其误差在一个机器周期之内。
10. 若晶振为 12MHz，现需要 1s 的定时，应如何实现？
11. 试编制一段程序，其功能为当 P1.2 上跳时对 P1.1 的输入脉冲进行计数，当 P1.2 下跳时停止计数，并将计数值写入 R6R7。
12. 设 f_{osc}=12MHz，试编写一段程序，其功能为对定时器 T0 初始化，使之工作于方式 2，产生 200μs 定时，并用查询 T0 溢出标志的方法，控制 P1.0 输出周期为 2ms 的方波。
13. 设 f_{osc}=11.0592MHz，试编写一段程序，其功能为对串行口初始化，使之工作于方式 1，波特率为 1200，并用查询串行口状态的方法，读出接收缓冲器的数据并回送到发送缓冲器。
14. 试编写一段对中断系统初始化程序，使之允许 INT0、INT1、T0、串行口中断，且使 T0 中断为高优先级中断。
15. 设输入到 P1.0～P1.3 上的脉冲信号周期在 3s 以上，脉冲的上跳变和下跳变都有 30ms 的抖动期。设 f_{osc}=12MHz，试编写一个初始化程序和 T0 中断服务程序，使 T0 工作于方式 1，产生 10ms 的定时中断，由 T0 中断服务程序滤除 P1.0～P1.3 的跳变抖动信号，读取 P1.0～P1.3 的有效输入状态输出到 P1.4～P1.7。
16. 试编制一个子程序，对串行口初始化，使串行口以方式 1、1200 波特率（晶振为 11.0592MHz）发送字符串："MCS—51 Emulator Intel COP."。
17. 若晶振为 11.0592MHz，串行口工作于方式 3，波特率为 2400，第 9 位数据为奇校验位。试编制一个程序，对串行口初始化，并用查询方式接收串行口上输入的 16 个字符存于内部 RAM 中 30H 开始的区域，若对 RB8 校验出错则停止接收，并清"0"P1.0，若正确地接收到 16 个字符则停止接收，并清"0"P1.1。
18. 若晶振为 12MHz，用 T0 和外部中断来测试 128 个脉冲（频率在 1kHz 左右变化，测试误差在 1 个机器周期之内）的平均周期，请编写出有关的初始化程序和中断服务程序。
19. 试编写一个子程序，其功能为将（R0）指出的两个 RAM 单元中的数转换为四个 ASCII 字符，并用查询方式从串行口上发送出去（设串行口由主程序初始化）。
20. 若晶振为 6MHz，用 T0 产生 500μs 的定时中断，试编写出有关的初始化程序和对时钟进行计数的 T0 中断服务程序。时钟计数单元为：30H、31H、32H，分别存储压缩 BCD 码的时、

分、秒参数。

21. 在一个 8031 系统中，晶振为 12MHz，P1 口上输入 8 路脉冲，频率为 0.1～3Hz，现用 T0 产生 1ms 定时，由 T0 中断服务读 P1 口的状态，若发生上跳则该路软件计数单元加 1，每到一分钟将各路计数值拆成 2 位十六进制数送显示缓冲区 70H～7FH，并清"0"各计数器。试编写出具有上述功能的有关程序。

22. 试用定时器 T0 产生 500μs 定时，在 P1.0 上产生频率为 1kHz 的方波（设 f_{osc}=12MHz，输出脉冲周期误差在 8 个机器周期之内）。试说明实现上述功能的条件，并编制出有关的程序。

23. 简易顺序控制器监控程序。在一个简易顺序控制器中，用 8031 P1 口上的八个继电器来控制一个机械装置的八个机械动作，要求 P1 口输出如图 4-8 所示的波形，为这个控制器编写一个监控程序。

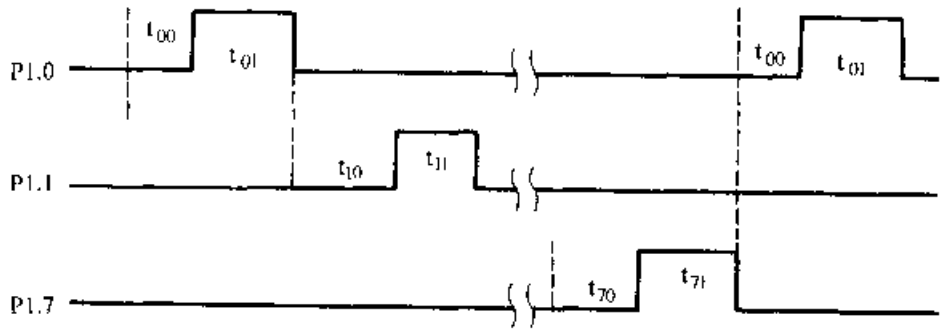


图 4-8　简易顺序控制器输出波形

24. 脉冲宽度测试程序。该程序的功能是测试 P3.3 上输入的正脉冲宽度（图 4-9），将测试的结果送内部 RAM 缓冲器中。

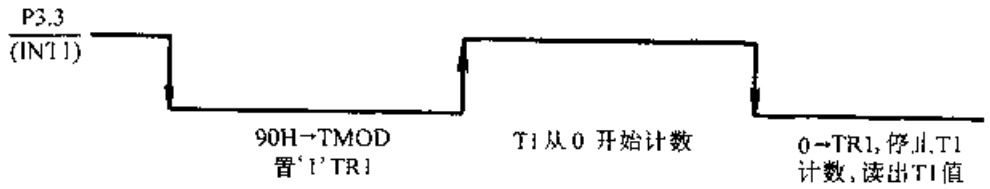


图 4-9　脉冲宽度测试原理

25. 双机通信程序。设甲乙二机为同一机柜之内的两个单片机系统，它们各自完成规定的功能，又通过串行通信协同工作。如图 4-10 所示，将它们的串行口直接相连。甲机向乙机发送数据时，在 P1.0 上产生一个负脉冲，向乙机请求中断，乙机响应后，在 P1.0 输出负脉冲。接着便进行数据传送。乙机向甲机发送数据的过程与之相同。编写发送和接收程序。

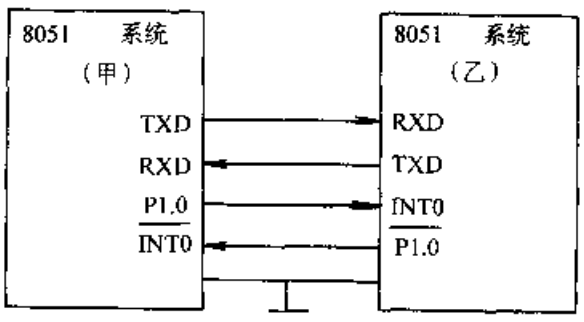


图 4-10　双机通信接口电路

第五章　MCS—51 的系统扩展

从单片机的本意来说，它集成了计算机系统的所有基本功能模块：CPU、存储器、I/O 接口。在智能仪器仪表、家电智能化和一些小型控制系统中，可以直接使用单片机而不必再扩展外围芯片，使用极为方便。但对于较大的应用系统，单片机片内所具有的功能将显得不足，这就必须在片外扩展一些外围芯片以增强和/或增加单片机的功能。

5.1　系统扩展概述

用单片机组成应用系统时，如单片机所具有的各种功能能满足应用系统的要求，这样的系统称为最小系统。最小系统包括必须外接的晶振电路，复位电路和电源部分等，对于 8031 单片机，其最小系统还包括外接的程序存储器芯片。

当单片机最小系统不能满足系统的功能要求时，就要扩展 EPROM、RAM、I/O 接口电路芯片或其他所需的外部芯片。为了使单片机能方便的与各种扩展芯片连接，应把单片机的 I/O 口看作为一般的总线结构，总线有并行总线和串行总线。

5.1.1　并行总线

单片机进行并行扩展时应将其 I/O 口看作为一般的微型机三总线结构形式，其三总线的构成如下。

1. 地址总线

MCS—51 单片机可以提供十六位地址线。高八位地址线由 P2 口提供。P2 口具有锁存功能，可以和外部芯片的高位地址线直接相连。低八位地址线由 P0 口提供。P0 口为地址/数据分时复用的 I/O 口，所以，为保持整个取指周期内低八位地址的稳定，需外加地址锁存器以锁存低八位的地址信息，如 74LS373 或 74LS273（八路 D 触发器），锁存器的锁存触发信号用 ALE（74LS373）或将 ALE 取反（73LS273），这样，在 P0 口低位地址输出有效时，由 ALE 的下降沿将低八位地址锁存在地址锁存器的输出端。

2. 数据总线

由 P0 口提供。当 P0 口用作地址/数据口时，是双向的具有输入三态控制的通道口，可与外部芯片的数据口直接相连。

3. 控制总线

系统扩展时常用的扩展信号为 ALE、$\overline{\text{PSEN}}$、$\overline{\text{WR}}$和$\overline{\text{RD}}$等。其中，ALE 是地址锁存信号，$\overline{\text{PSEN}}$是程序存储器输出允许信号，$\overline{\text{WR}}$是数据存储器或外部功能器件

写信号，$\overline{RD}$是数据存储器或外部功能器件读信号。

MCS—51 总线结构见图 5-1。

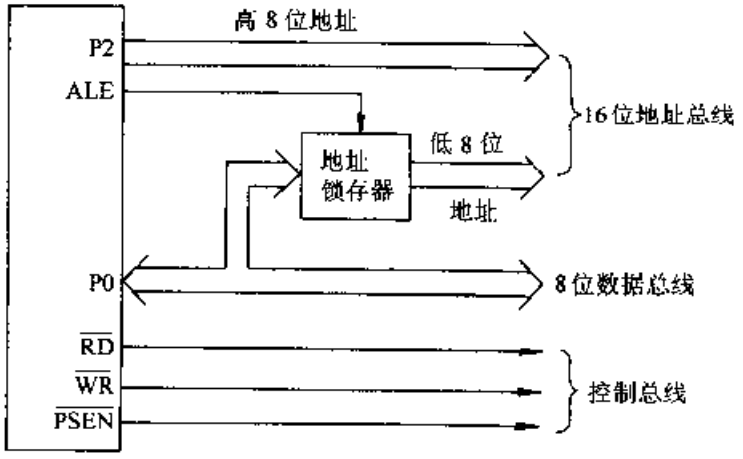


图 5-1　MCS—51 总线结构

5.1.2　串行总线

使用并行口方式进行系统扩展时应采用上述三总线结构。进行系统扩展也可以使用串行接口方式。常用的串行接口总线有：UART 的工作方式 0 和 I^2C 总线标准。UART 接口前面已有介绍，这里讲述 I^2C 总线接口。

I^2C 总线是一种双向二线制同步串行总线，其电器结构图见图 5-2。

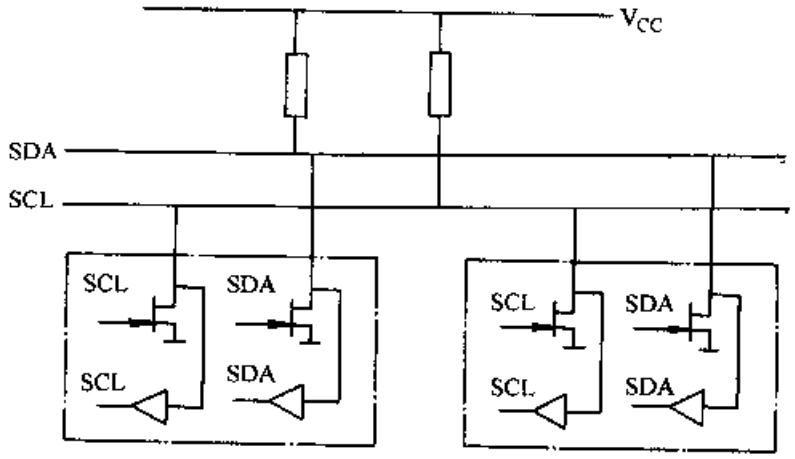


图 5-2　I^2C 总线电器结构图

SDA 是串行数据线，SCL 是串行时钟线。每个连到总线上的器件都有自己的地址。由于具有竞争检测和仲裁电路，可实现多主通信系统。具有 CPU 的 I^2C 器件都可以是主器件，每个主器件即可作为发送器也可以作为接收器。

同步时钟允许通过总线以不同的波特率进行通信。同步时钟又是停止和重新启动串行发送的握手信号。

目前具有 I^2C 总线的 8051 系列单片机有 8×C550、8×C552、8×C652、8×C654、8×C751、8×C752 等。还有具有 I^2C 总线的 LED 驱动器、LCD 驱动器、A/

D、D/A 转换器、RAM、EPROM、键盘显示器接口电路及 I/O 接口等上百种电路芯片。

I^2C 总线可以构成多主数据传送系统。主器件发送时钟、启动位、停止位、数据传送方向，从器件接收时钟及数据传送方向。接收或发送则根据数据传送方向决定。I^2C 总线上数据传送时的启动、停止和有效状态都由 SDA、SCL 的电平状态决定。在 I^2C 总线规约中启动和停止条件规定如下：

启动条件：在 SCL 为高电平时，SDA 出现一个下降沿则启动 I^2C 总线。

停止条件：在 SCL 为高电平时，SDA 出现一个上升沿则停止 I^2C 总线。

除了启动和停止状态，在正常数据传送状态下，SCL 的高电平都对应于 SDA 的稳定数据状态。

每一个被传送的数据位由 SDA 上的高、低电平表示，每一个被传送的数据位都在 SCL 线上产生一个时钟脉冲。在时钟脉冲为高电平期间，SDA 线上的数据必须稳定，否则被认为是启停控制信号。SDA 只能在 SCL 时钟脉冲为低电平期间改变。在启动和停止条件之间可传送的数据长度不受限制，以 8 位一个字节为单位，首先传送最高位。在每个字节后必须跟一个响应位。接收器接收数据时，每接收一个字节数据后，发送一个应答信号(ACK)，在这期间，发送器必须保证 SDA 为高电平，由接收器将 SDA 拉低。主器件在接收了最后一个字节之后不发送应答信号，也称为非应答信号(NOT ACK)。当从器件不能再接收另外的字节时也会出现这种情况。

主器件发动一次数据传送的过程为：

1) 主器件发送启动信号，发送的第一个字节格式如下：

D6	D5	D4	D3	D2	D1	D0	$R/\overline{W}$

其中：D6～D0 为被寻址的从机地址，高位在前；

$R/\overline{W}$ 为数据传送方向，1：由从机到主机；0：由主机到从机。

该字节发送后，接收方发送一应答位；

2) 循序发送数据字节，每一个字节后由接收方发送一应答位；

3) 发送最后一个字节数据，接收方发送非应答信号；

4) 主机发送停止信号，结束此次数据传送。

I^2C 总线的数据传送波特率是不固定的，在器件允许速率范围内由主机控制。

对于内部没有集成 I^2C 总线接口的 8051 系列单片机，当与具有 I^2C 总线接口的器件连接时，可以用软件模拟 I^2C 总线接口功能。下面给出通用的 I^2C 总线读写程序，程序中定义 P1.6 和 P1.7 作为 I^2C 总线的 SCL 和 SDA 信号。

程序包括如下 I^2C 总线功能函数：

I_init( ),初始化。

delay( ),延时。

I_start( ),启动信号。

I_stop( ),结束信号。

I_sendbit( ),发送一位。

I_readbit( ),接收一位。

I_sendbyte( ),发送一个字节。

I_readbyte,接收一个字节。

I_sendack,应答信号。

I_sendnak,非应答信号。

B_ack,判应答。

```
#include"reg51.h"
#include"intrins"
sbit  SCL  P1^6;
sbit  SDA  P1^7;
void delay(unsigned char n)
{
  char j=0;
  while(j<n)
  {
    _nop_( );
    j--;
}
}
void I_start( )
{
  SCL=1;
  Delay(3);
  SDA=0;
  delay(3);
  SCL=0;
  SDA=1;
  delay(3);
}
```

```
void I_stop( )
{
  SDA=0;
  delay(1);
  SCL=1;
  dekay(3);
  SDA=1;
  dekay(3);
  SCL=0;
  delay(3);
}
  void I_init( )
{
  SCL=1;
  I_stop( );
}
  void I_sendbit(bit x)
{
  SDA=x;
  delay(1);
  SCL=1;
  delay(7);
  SCL=0;
  delay(3);
}
bit I_readbit( )
{
  bit s;
  SCL=1;
  delay(3);
  s=SDA;
  delay(2);
  SCL=0;
  delay(3);
  return s;
```

```
}
  void I_sendbyte(char x)
{
  bit s;
  char x=8;
  while(x--)
  {
s=x&0x80;
        I  sendbit(s);
        x=_crol_(x,1);     /*左环移一位*/
}
}
char I_readbyte( )
{
  char x=8,y=0;
  bit s;
  while(x--)
  {
  y=_crol_(y,1);
  s=I_readbit( );
  y=y|s;
}
return y;
}
void I_sendack( )
{
  I_sendbit(1);
}
void I_sendnak( )
{
  I_sendbit(0);
}
bit B_ack( )
{
```

```
    return I_readbit( );
}
```

5.1.3 系统扩展的内容和方法

1. MCS—51 系统扩展的内容

1) 外部程序存储器扩展。

2) 外部数据存储器扩展。

3) 输入输出接口扩展。

4) 管理功能器件扩展。如定时/计数器、中断优先编码器等。

2. 系统扩展的基本途径

1) 使用 TTL、CMOS 中小规模集成电路:这是一种常用的简单的扩展方法。与总线相连应符合"输出锁存、输入三态"的原则。

2) 采用 Intel MCS—80/85 微处理器外围芯片。

3) 采用为 MCS -48 单片机设计的外围芯片。

4) 采用与 MCS -80/85 外围器件兼容的其他通用标准芯片。

3. 系统扩展多片电路的方法

在扩展外部程序存储器、数据存储器和各种功能器件时,各器件的地址属于两个不同的地址空间,程序存储器芯片属于程序地址空间,数据存储器和各种功能器件的地址属于外部数据存储器空间。在一个地址空间中扩展多片芯片时,为保证只对一个芯片进行操作,必须对各芯片的片选信号进行控制。有两种方法实现片选控制。

(1) 线选法

使用 P2 口未使用的口线作片选控制线。

如使用多片容量为 2K 字节的数据存储器 6116 进行系统扩展,每片 RAM 需要 11 根地址线,高 8 位地址总线 P2 口只用了三位:P2.0、P2.1 和 P2.2,其余 5 根口线用于控制各片的片选信号。被操作的芯片的片选线为低电平,其余芯片的片选线为高电平。这样可以扩展 5 片 RAM 共 10K 字节容量。但这 10K 字节数据存储器的单元地址是不连续的,它们的地址范围是:第一片:0xF000～0xF7FF;第二片:0xE800～0xEFF;第三片:0xD800～0xDFF;第四片:0xB800～0xBFF;第五片:0x7800～0x7FF。

(2) 译码法

为了使用较少的口线选择更多的芯片,可以使用译码法。TTL 集成电路或 CMOS 集成电路中有 2-4 译码器、3-8 译码器、4-10 译码器和 4-6 译码器。

如使用 4-16 译码器对多片容量为 2K 字节的数据存储器 6116 进行系统扩展,P2 口的 P2.3、P2.4、P2.5 和 P2.6 作译码器的输入信号,P2.7 作译码器的片选控制信号。这时可以扩展 16 片 6116,即 32K 字节容量的 RAM,而且,它们的地址是

连续的，由 0x0000～0x7FF。当 P2.7 为 1 时，各 RAM 芯片都未被选中。再添加一片 4-16 译码器，仍然用 P2 口的 P2.3、P2.4、P2.5 和 P2.6 作译码器的输入信号，P2.7 取反后作译码器的片选控制信号。则这片译码器控制的 16 片 RAM 的地址范围是：0x8000～0xFFFF，扩展的 RAM 容量达到极限容量 64K 字节。

5.2 程序存储器的扩展

程序存储器常采用 EPROM、EEPROM 和 FLASH ROM 芯片。程序存储器的最大扩展量是 64K 字节。

程序存储器的引脚信号的类型有：地址信号，信号线的位数由芯片的容量决定；数据信号，一般是 8 位；输出允许，$\overline{OE}$；片选信号，$\overline{CE}$；电源，V_{CC}和 GND。程序存储器的片内单元数有 1K、2K、…、32K 字节不等。

扩展程序存储器时，只需将程序存储器所需的高 8 位地址线与单片机的 P2 口相连，低 8 位地址线与地址锁存器 74LS373 的输出相连，将程序存储器的数据线与单片机的数据总线 P0 相连，单片机的指令输出允许线$\overline{PSEN}$与程序存储器的读允许线($\overline{OE}$)相连。一般系统中只需扩展一片程序存储器，程序存储器的片选引脚($\overline{CE}$)直接接地即可。接线方式见 8031 最小系统图。

当要扩展两片程序存储器时，各程序存储器的片选引脚必须分别与 P2 口未用的口线相连。例如，要扩展两片 EPROM，2716(2K*8)，每片需使用 11 根地址线，即使用 P0 口经锁存后输出的低 8 位地址线和 P2 口的 P2.0、P2.1 和 P2.2。为了区分两片 EPROM，用 P2.3 接一片 EPROM 的片选引脚，将 P2.3 取反接另一片 EPROM 的片选引脚。这时，两片程序存储器单元的地址从 0x0000 到 0x0FFF 是连续的。若要扩展三片以上的程序存储器，为保证地址连续，必须使用译码法进行片选。

5.3 数据存储器的扩展

5.3.1 并行扩展方式

采用并行扩展方式扩展外部数据存储器时，其地址线、数据线的连接方法与扩展程序存储器的方法相同。

数据存储器一般采用半导体随机读写存储器(RAM)，其引脚类型与程序存储器基本相同，只是为写操作而设置了一个写控制信号$\overline{WR}$，其输出允许信号表示为$\overline{RD}$。

与数据存储器的读写信号线相连的 8051 的控制线是$\overline{WR}$(P3.6)和$\overline{RD}$(P3.7)。数据存储器的地址线和数据线的连接方法与程序存储器相同。

MCS-51 单片机的内部数据存储器空间和外部数据存储器空间是独立编址的。

外部数据存储器通常分为两个区域：

(1) 字节地址 00H～FFH

共 256 个字节，可以用 8 位地址寻址。对这个区域的单元操作的汇编语言指令为：MOVX A，@Ri 和 MOVX @Ri，A，i 为 0 或 1。这个区域的 C 存储类型为 pdata，定义这个区域的变量的格式为：

```
char pdata x1,x2;
```

(2) 字节地址 0000H～0FFFFH

共 64K 字节。对这个区域的单元操作的汇编语言指令为：MOVX A，@DPTR 和 MOVX @DPTR，A。这个区域的 C 存储类型为 xdata，定义这个区域的变量的格式为：

```
char xdata x3,x4;
```

单片数据存储器的扩展，其地址、数据线与片选信号的连接与单片程序存储器扩展方法相同，读、写控制需$\overline{RD}$和$\overline{WR}$信号，参见图 1-11。

当扩展多片外部数据存储器时，其片选信号的处理可以用线选法或译码法。

5.3.2 串行扩展方式

24C01B/02B/04B/08/16/32 是 128 字节/256 字节/512 字节/1K 字节/2K 字节/4K 字节×8 位的 I^2C 总线串行 EEPROM 器件，其芯片引脚见图 5-3。

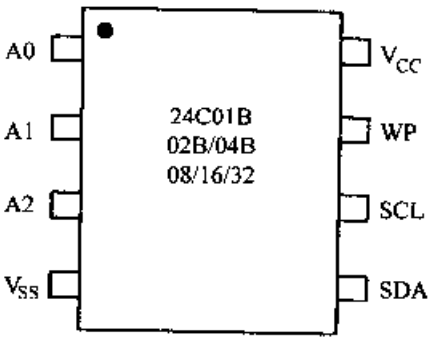


图 5-3 24C 系列 I^2C 总线串行 EEPROM

这里介绍 24C02B 的使用方法。

芯片引脚说明：

WP：写保护。只能接 V_{CC}或 GND，接 V_{CC}时，芯片只读；接 GND 时，可读写。

A0、A1、A2：芯片选择。一条 I^2C 总线上可以连接 8 片 24C02B。

24C 系列芯片的串行数据传送速率为 100～400K 字节/s。

24C02B 的操作有两种方式：字节读写方式和页读写方式。

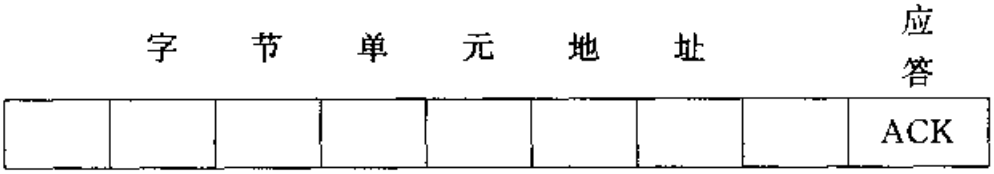
1. 字节读写方式

在这种方式下，主器件发送寻址控制字(寻址从机地址和数据传送方向)，然后发送单元地址，其后是传送的字节数据。SDA 引线上的数据格式为：

控制节字：

启动	控制字节								应答
S	1	0	1	0	A2	A1	A0	R/$\overline{W}$	ACK

单元地址字节：

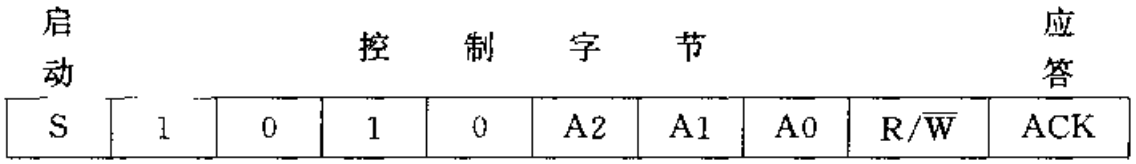


数据字节：

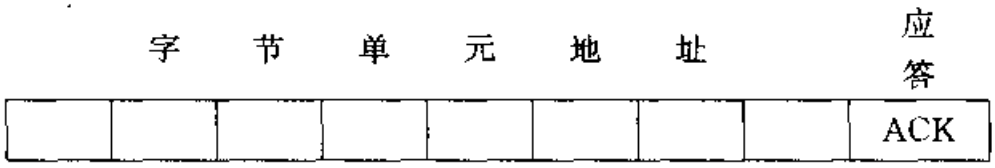
2. 页操作方式

对 24C02B 而言，一页是 8 个字节。在页操作方式下，前三个字节的传送与字节方式相同，只是数据字节的应答之后，不是发送停止信号，而是再继续传送至多 7 个数据字节和应答位，最后是停止信号。这时所发送的字节单元地址是页数据单元的首地址。页操作的数据格式为：

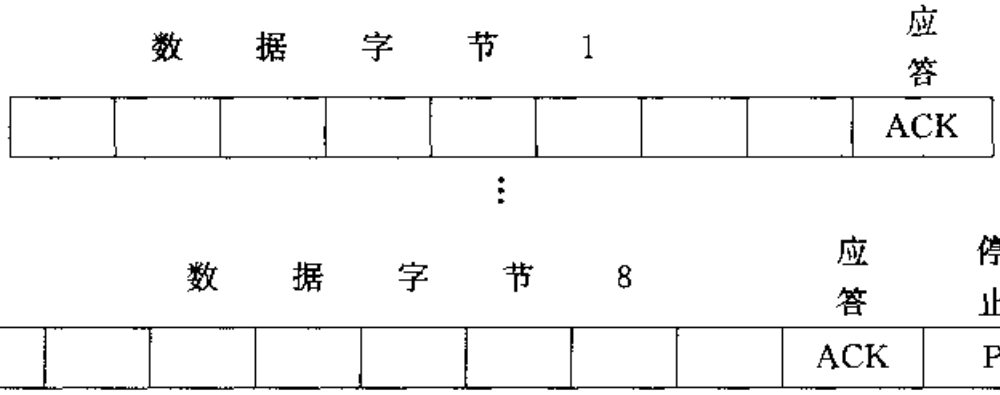
控制字节：



单元地址字节：



数据字节：



5.4 I/O 口的扩展

当单片机应用系统中扩展了程序存储器或数据存储器或 A/D、D/A 等功能器

件时，单片机的 P2 口和 P0 口被用于地址线和数据线，P3 口的部分口线（$\overline{WR}$，P3.6，$\overline{RD}$，P3.7）也已被使用，作为整体能被使用的 I/O 口只有 P1 口。当采用系统较复杂时，I/O 口就不够用了，必须加以扩展。I/O 口的扩展可以采用并行 I/O 口扩展方法，也可以采用串行 I/O 口扩展方法，常用的是并行 I/O 口扩展。

并行 I/O 口扩展方法主要有两种：一是利用通用 TTL 或 CMOS 总线扩展接口；另一种是采用专用的 I/O 扩展接口，如 8155、8255 等。

5.4.1 简单的并行 I/O 接口扩展

根据输入三态、输出锁存的原则，采用 TTL 或 CMOS 电路组成简单的并行 I/O 接口。可以使用的芯片有 74LS373、74LS273、74LS241 和 74LS245 等。图 5-4 所示为某一实用电路。

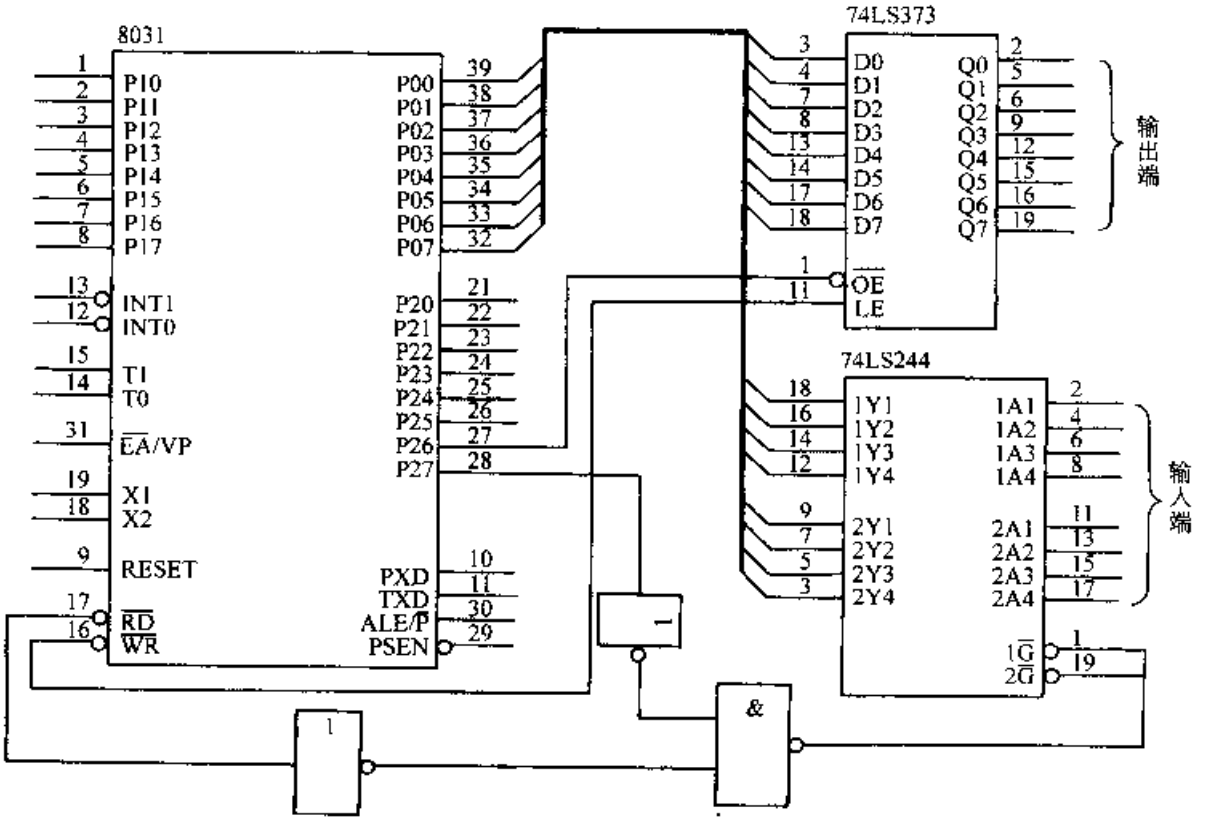


图 5-4 简单 I/O 扩展

如图所示，用具有三态缓存器的 74LS244 构成输入口，用具有锁存功能的 74LS373 组成输出口。

扩展 I/O 接口时，I/O 扩展芯片的片选引脚接 P2 口的某个引脚，即将其看作地址线。每一 I/O 芯片安排惟一的地址。I/O 扩展芯片的地址与外部 RAM 共同编址，对它们的访问都用 MOVX 指令。

当使用多个芯片扩展 I/O 接口时，可以使用译码法。

I/O 扩展接口与外部数据存储器统一编址。对图 5-4 中的输出端口，其地址编

码为:P2.6=0,其他 P2 口引脚和 P0 口任意,令其均为 1,即 1011111111111111=0xBFFF。图 5-4 中的输入端口的地址编码为:0x7FFF。编程时可定义如下:

```
#include <absacc.h>
#include <reg51.h>
#define OUT XBYTE[0xBFFF]
#define IN XBYTE[0x7FFF]
```

要输出数据时,可用指令:

```
OUT=x;
```

输入数据时,可用指令:

```
a=IN;
```

5.4.2 用 8155 扩展并行 I/O 接口

1.8155 的性能特点

1) 带有 256 字节 RAM。

2) 两个可编程 8 位输入输出口,一个可编程 6 位输入输出口。

3) 一个 14 位可编程定时/计数器。

2.8155 的功能框图

8155 的功能框图见图 5-5。

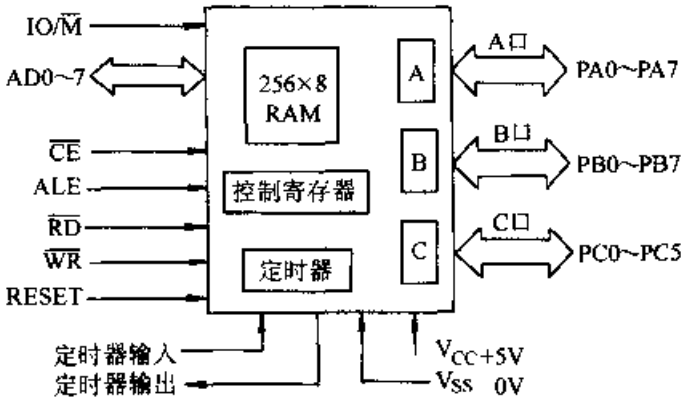


图 5-5 8155 的功能框图

3.8155 的引脚功能

8155 的引脚排列如图 5-6 所示。

AD0~7:地址/数据总线。与 P0 口直接相连。8155 内部有八位地址锁存器,该地址作为内部 RAM 或输入输出口的地址。传送数据时,由$\overline{RD}$或$\overline{WR}$信号决定是读出还是写入。

RESET:复位。宽度不小于 600ns。复位后,3 个 I/O 口被置为输入工作方式。

ALE:地址锁存允许。与 8051 的 ALE 相连,在 ALE 的下降沿将 AD0~7 8 位地址、使能信号 CE 和 IO/$\overline{M}$ 信号锁存在片内锁存器中。

IO/$\overline{M}$：IO/MEMORY 选择。区别对 8155 的操作是对 RAM 还是对 I/O 口。IO/$\overline{M}$ 为高电平时，操作对象是输入输出口或命令/状态寄存器。IO/$\overline{M}$ 为低电平时，操作对象是对内部 RAM。

$\overline{CE}$：芯片使能。低电平有效。

$\overline{RD}$：读信号。与 8051 的$\overline{RD}$相连。

$\overline{WR}$：写信号。与 8051 的$\overline{WR}$相连。$\overline{RD}$和$\overline{WR}$的操作对象由 IO/$\overline{M}$ 的状态决定。

PA0～7：8155A 口引脚。

PB0～7：8155B 口引脚。

PC0～7：8155C 口引脚。可作 I/O 线或 A、B 口的控制信号线。

TMRIN：定时/计数器输入端。

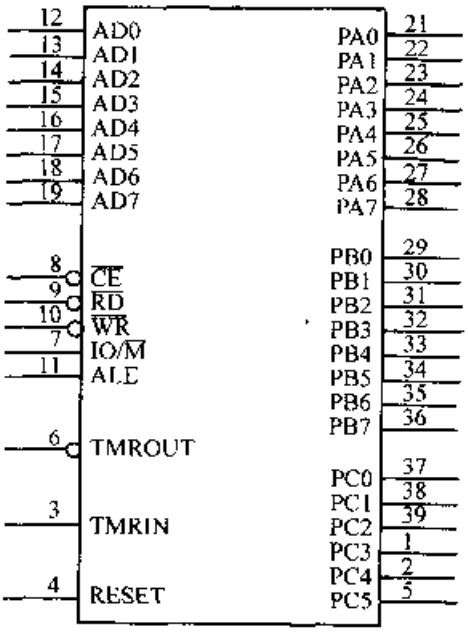


图 5-6　8155 的引脚排列

TMROUT：定时/计数器输出端。输出波形可编程为方波或脉冲波形。

4. 8155 的工作原理

(1) 一般说明

8155 内部具有 256 字节 RAM，最快存取时间为 400ns。三个 IO 通道，其中 C 口可编程用以决定另外两个端口 A 和 B 的信息交换方式。8155 还包含一个 14 位定时/计数器。

(2) RAM

当 CPU 对 8155 中的 RAM 进行存取时，应置低 IO/$\overline{M}$ 引脚，在 A0～7 引脚上给出单元地址信息。

(3) 定时/计数器

在 TMRIN 引脚加入时钟脉冲后，在 TMROUT 引脚上根据设置的方式可输出四种波形。

(4) I/O 口

8155 有 A、B、C 三个输入输出口。A 口和 B 口可以工作于基本输入输出方式或选通输入输出方式。C 口可以工作于基本输入输出方式或选通控制方式(此时 A 口或 B 口为选通输入输出方式)。

1) 基本输入/输出方式：以 A 口为例，当预先设定其为基本输入方式后，CPU 就可以用输入指令(MOVX A，@DPTR)从 PA0～7 读取 8 位数据。CPU 向 8155 发出的控制信号有$\overline{RD}$=0、IO/$\overline{M}$=1 以及 A 口的地址编码。若预先设定 A 口为基本输出方式，CPU 即可用输出指令(MOVX @DPTR，A)向 A 口输出 8 位数据。这时 CPU 要发出的信号是：$\overline{WR}$=0、IO/$\overline{M}$=1、A 口的地址编码以及要输出的数据。

2）选通输入方式：若 A 口设置为选通输入方式，则相应地 C 口的 PC0、PC1、PC2 就作为 A 口的选通控制引线。此时，PC0 作为 INTR（中断）、PC1 作为 BF（缓冲器满）、PC2 作为$\overline{\text{STB}}$（选通）。这些引线与 8051 和外设的连接如图 5-7 所示。

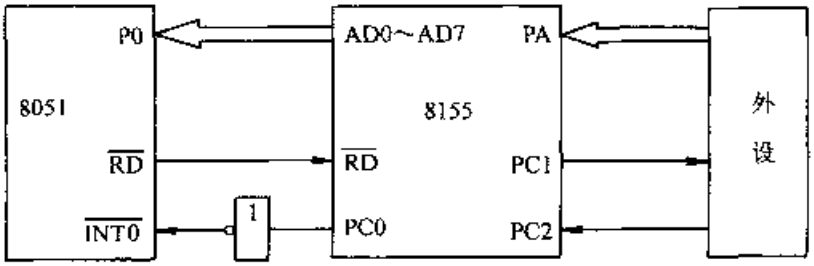


图 5-7　8155A 口选通输入方式

A 口的动作过程如下：

①　外部设备可提供数据时，就把数据送至 PA0～PA7 引脚，然后发出$\overline{\text{STB}}$信号。

②　8155 接收到数据后，就将 BF 置高，表示 A 口缓冲器已满，同时，INTR 引脚向 CPU 发出中断请求。

③　CPU 响应中断请求，读取 8155A 口数据。

④　8155 在数据被取走后，就清零 BF 引脚，恢复到①之前的状态。

3）选通输出方式：A 工作于选通输出方式时，C 口的 PC0、PC1、PC2 的作用与选通输入方式时相同，这些引线与 8051 和外设的连接如图 5-8 所示。

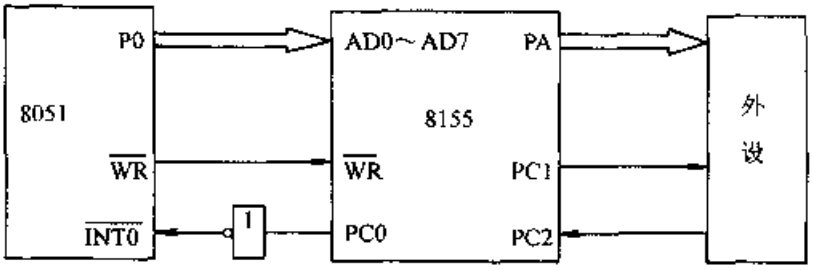


图 5-8　8155A 口选通输出方式

其工作过程如下：

①　当 8155 可接收数据时，就置高 INTR 引线，向 CPU 发出中断请求。

②　CPU 响应中断，向 8155A 口写数据。

③　8155 接收数据后，就向外设发出 BF 信号。

④　外设若可以接收数据，就从 8155 读取数据，并发出$\overline{\text{STB}}$信号。

⑤　8155 在$\overline{\text{STB}}$的后沿（上升沿）再将 INTR 置高，即回到①的状态。

（5）命令字及状态寄存器

8155 的三组 I/O 口以及定时/计数器有几种工作方式，在使用之前必须设置好工作方式。8051 通过设置 8155 的命令字寄存器的内容来设置工作方式，此后

8155 就按设定的方式动作。8155 命令字格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
TM2	TM1	IEB	IEA	PC2	PC1	PB	PA

其中：

PA： A 口功能　0：输入　1：输出

PB： B 口功能　0：输入　1：输出

PC2PC1	0 0	0 1	1 0	1 1
方式	1	2	3	4
PC0	输入口	输出口	A INTR	A INTR
PC1	输入口	输出口	A BF	A BF
PC2	输入口	输出口	A $\overline{\text{STB}}$	A $\overline{\text{STB}}$
PC3	输入口	输出口	输出口	B INTR
PC4	输入口	输出口	输出口	B BF
PC5	输入口	输出口	输出口	B $\overline{\text{STB}}$

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

TM2 和 TM1 与定时器有关，放到后面讲。

8155 的状态寄存器用来记录 8155 的当前状态。其格式为：

D7	D6	D5	D4	D3	D2	D1	D0
X	TIMER	INTE B	BF B	INTR B	INTE A	BF A	INTR A

INTR(A)：　A 口中断请求

BF(A)：　A 后缓冲器满/空

INTE(A)：　A 口中断允许

X：　保留

8155 的命令寄存器只能写入，不能读出。状态寄存器只能读出，不能写入。当对 8155 的 RAM 进行存取时，置 IO/$\overline{\text{M}}$ 为 0，AD0～AD7 8 位地址即可对所有 RAM 单元进行寻址。

当对 8155 的 I/O 端口、命令字状态字寄存器或定时/计数器进行操作时，则要区分所操作的对象。首先，置 IO/$\overline{\text{M}}$ 为 1，命令字寄存器、状态字寄存器、计数器高

低字节、A 口、B 口和 C 口的地址编码分别为：

A7	A6	A5	A4	A3	A2	A1	A0	选中的寄存器
×	×	×	×	×	0	0	0	命令/状态寄存器(写/读)
×	×	×	×	×	0	0	1	A 口
×	×	×	×	×	0	1	0	B 口
×	×	×	×	×	0	1	1	C 口
×	×	×	×	×	1	0	0	定时器低 8 位寄存器
×	×	×	×	×	1	0	1	定时器高 6 位寄存器

注：×表示任意值，一般取 1。

这时 8155 的片选引脚必须为低电平。

5. 8155 与 8051 的连接与应用

由图 5-9，8155 可直接与 MCS—51 系列单片机相连，不需要任何外加逻辑电路。

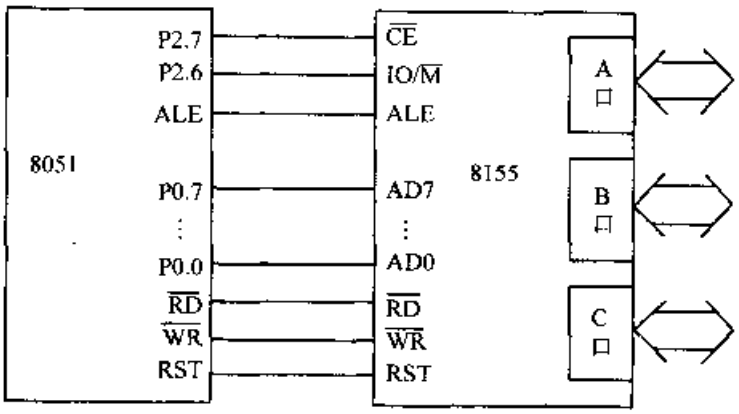


图 5-9　8155 与 8051 基本连接图

通常选用 P2 口的高位地址作 8155 的片选信号和 IO/$\overline{M}$ 选择信号。8051 的 ALE 与 8155 的 ALE 相连，利用 8051 ALE 的下降沿把 P0 口的地址锁存在 8155 内的锁存器中。P0. 0～P0. 7 与 AD0～AD7 相连，$\overline{RD}$和$\overline{WR}$也有对应相连端口。

按如图连接方法，8155 的地址分配如下：

0111 1111 1111 1000～0111 1111 1111 1101

即，7FF8H～7FFDH。

对 8155 内部 RAM 的地址分配如下：

0011 1111 0000 0000～0011 1111 1111 1111

即，3F00H～3FFFH。

对 8155 的操作可分述如下：

```
#include <absacc.h>
#define STATE8155 XBYTE[0x7FF8]
#define IOA XBYTE[0x7FF9]
```

```
#define IOB XBYTE[0x7FFA]
#define IOB XBYTE[0x7FFB]
```

(1) 写命令字寄存器

```
STATE8155=x;
```

(2) 读状态寄存器

```
x=STATE8155;
```

(3) 写 A 口

```
IOA=x;
```

(4) 读 B 口

```
x=IOB;
```

(5) 写 8155 的 RAM 单元(20H)

```
XBYTE[0x3F20]=35;
```

(6) 读 8155 的 RAM 单元(65H)

```
x=XBYTE[0x3F65];
```

6. 8155 定时器的使用

8155 的定时器是一个 14 位减法计数器，有四种工作方式。8155 内部有两个寄存器用来存放定时器的工作方式和计数初值。它们的结构分别为：

(1) 定时器方式和计数值高 6 位(地址 FDH)

D7	D6	D5	D4	D3	D2	D1	D0
M2	M1	T13	T12	T11	T10	T9	T8

(2) 计数值低 8 位(地址 FCH)

D7	D6	D5	D4	D3	D2	D1	D0
T7	T6	T5	T4	T3	T2	T1	T0

8155 定时器的四种工作方式有 M2 和 M1 的取值决定：

M2	M1	方　　式
0	0	在计数的后半部分，输出低电平
0	1	矩形波，周期为计数长度(自动重装入)
1	0	计数到零时，输出一个单脉冲
1	1	自动重装入，每次计数到零，输出一个单脉冲

T0~T13 用于控制输出定时，由程序设置计数器的计数长度。14 位长度需两

次写入。对定时器四种工作方式的说明如下：

方式 1：启动定时器后，定时器只计数一次。在给定的计数长度的前半部分，定时器输出高电平；计数的后半部分输出低电平。当计数值减到零时，定时器输出高电平。

方式 2：启动定时器后，定时器开始连续计数，当计数值减到零时，系统自动重新装入计数初值。在一个计数周期（计数长度）内，前半部分输出高电平，后半部分输出低电平。如此形成周期矩形波。

方式 3：启动定时器后，定时器只计数一次。当计数值减到零时，定时器输出一个低脉冲。宽度为定时器输入时钟周期。

方式 4：启动定时器后，定时器开始连续计数，当计数值减到零时，系统自动重新装入计数初值，并输出一个低脉冲。

8155 命令寄存器中有两项与定时器有关：

TM2	TM1	定时/计数器工作方式
0	0	不影响计数器操作
0	1	若计数器未启动，无操作；否则，停止计数
1	0	达到当前计数 TC 值后，立即停止；若未启动，无操作
1	1	装入方式和计数值后立即启动计数器；若计数器已在运行，则达到当前计数值后，按新的方式和计数值予以启动

8155 定时器对 TMRIN 引脚输入的外部事件的计数脉冲或定时脉冲进行计数，减法计数器减至零时，TMROUT 引脚输出定时到矩形波或脉冲信号。使用定时器时，应先将计数长度写入定时器寄存器中，设置定时器工作方式，然后再装入命令字并启动定时器。

定时器在计数期间，可以将新的计数长度值和计数方式写入计数长度寄存器中，它不影响定时器原来的操作，只有装入了新的启动控制命令到命令寄存器，等原来的计数值到零后才按新的工作方式及计数长度进行工作。

5.4.3 用 8255A 扩展并行 I/O 接口

8255A 有三个 8 位的并行口，端口既可以编程为普通 I/O 口，也可以编程为选通 I/O 口和双向传输口。

1. 8255A 的引脚

8255A 的引脚见图 5-10。

D0～D7：　双向数据线，与 8031 的 P0 口直接相连。

RESET：　复位输入。

$\overline{CS}$：　片选。

$\overline{WR}$：　写允许。

$\overline{RD}$：　　读允许。

PA7～PA0：端口 A。

PB7～PB0：端口 B。

PC7～PC0：端口 C。

A1、A0：　　端口地址线，选择如下：

A1	A0	选通的端口
0	0	口 A
0	1	口 B
1	0	口 C
1	1	命令字口

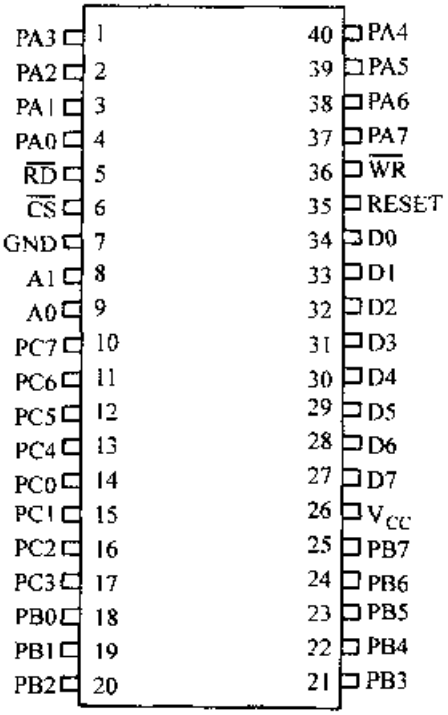


图 5-10　8255A 芯片管脚图

2. 8255A 的命令字和工作方式

8255A 有两个命令字：方式选择字和口 C 指定位的置位/复位命令字。这两个命令字占用同一地址，由最高位的状态加以区别。

(1) 工作方式命令字

8255A 有三种工作方式：方式 0、方式 1 和方式 2。方式命令字的格式见图 5-11。

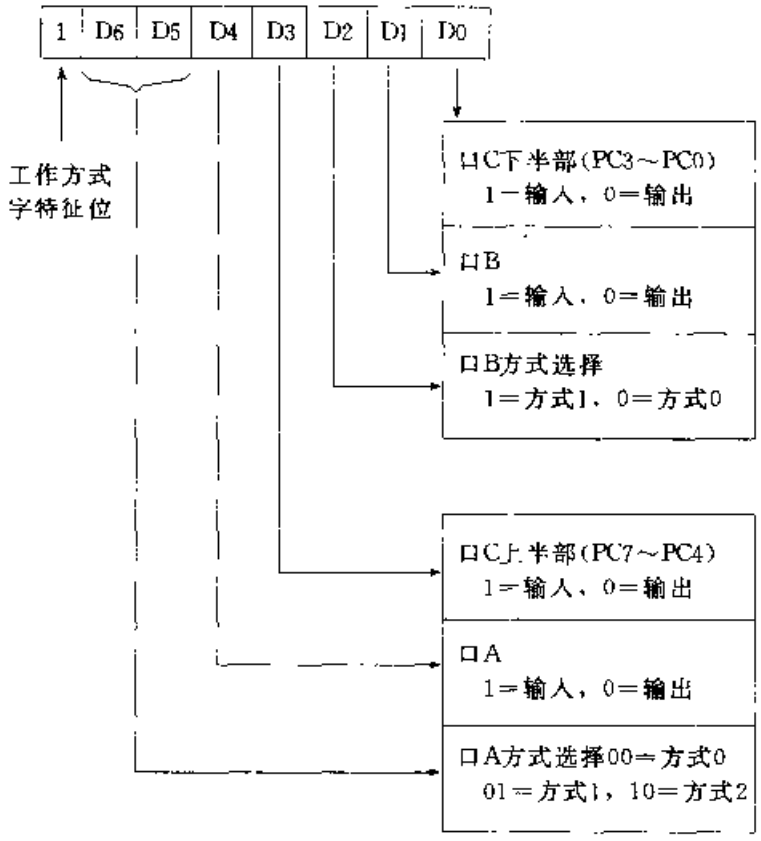


图 5-11　8255A 的方式选择命令字

由图可以看出，口 A 有 3 种工作方式，口 B 只有 2 种工作方式。

(2) 口 C 置位/复位命令字

8255A 的口 C 的输出具有位控制功能：按位置位或复位。其操作由口 C 的置位/复位命令字控制，其格式如下：

0	×	×	×	D3	D2	D1	D0

D7：命令字标识位。D7 为 0 是置位/复位命令字。

D3、D2、D1：口 C 的 8 个位选择。000～111 的 8 种状态对应于 PC0～PC7 的 8 个位。

D0：置位/复位选择位。D0＝1 指定位置 1；D0＝0 指定位清 0。

(3) 8255A 的工作方式

1) 方式 0：方式 0 是基本输入/输出方式。在这种方式下，端口按方式选择命令字指定的方式作输入口或输出口，作输出口时端口具有锁存功能；作输入口时，只有口 A 有锁存功能。口 C 的高、低四位可以分别定义为输入或输出。

2) 方式 1：方式 1 是选通输入/输出方式。在方式 1 下，8255A 的三个端口被分成 A 组和 B 组。A 组中，口 A 为 I/O 口，口 C 的三位为其提供联络信号。B 组中，口 B 为 I/O 口，口 C 的三位为其提供联络信号。方式 1 的使用方法与 8155 的选通输入/输出方式相同。

3) 方式 2：方式 2 为双向传输方式，只适用口 A。口 A 工作于方式 2 时，口 C 提供 5 个联络信号。

方式 1 和方式 2 时口 C 的功能见下表：

口 C	方式 1		方式 2
	输入	输出	
PC0	$INTR_B$	$INTR_B$	I/O
PC1	IBF_B	$\overline{OBF_B}$	I/O
PC2	$\overline{STB_B}$	$\overline{ACK_B}$	I/O
PC3	$INTR_A$	$INTR_A$	$INTR_A$
PC4	$\overline{STB_A}$	I/O	$\overline{STB_A}$
PC5	IBF_A	I/O	IBF_A
PC6	I/O	$\overline{ACK_A}$	$\overline{ACK_A}$
PC7	I/O	$\overline{OBF_A}$	$\overline{OBF_A}$

表中：

$\overline{STB}$：选通信号输入，低电平有效。当外围设备将数据送到口线上时，将$\overline{STB}$置低，端口数据打入输入缓冲器。

IBF：输入缓冲器满信号，高电平有效，当$\overline{STB}$为低时，8255 将 IBF 置高，通知

外围设备输入缓冲器已满。外围设备将$\overline{STB}$置高。

INTR：中断请求信号，高电平有效。当$\overline{STB}$信号结束且 IBF 有效，即$\overline{STB}=1$且 IBF=1 时 INTR 为高电平，向 CPU 发中断请求。当 CPU 响应中断，从 8255A 读取数据后，8255A 将 INTR 和 IBF 信号置低。

$\overline{OBF}$：输出缓冲器满信号，低电平有效。通知外围设备读数据。

$\overline{ACK}$：外围设备读取数据后，向 8255A 的应答信号，表示输出缓冲器空，这时，8255A 将$\overline{OBF}$置低，将 INTR 置高，向 CPU 发中断请求。

3. 8255 与 8051 的连接和使用

(1) 8255 与 8051 的连接

8255 与 8051 的连接主要有：

数据总线 D0～D7：它们分别与 8051 的 P0.0～P0.7 相连。

地址线 A0、A1：分别与地址锁存器 74LS373 的输出线 Q0、Q1 相连。

片选线$\overline{CS}$：与 8051P2 口的一条未使用的地址线相连，若为 P2.7，则 8255 各口及命令字寄存器的地址分别是 A 口：0x7FFC；B 口：0x7FFD；C 口：0x7FFE；命令字寄存器：0x7FFF。

8255 的中断请求信号要经过取反后接 8051 的外部中断输入引脚。

(2) 8055 的编程

首先定义 8255 各口和命令字寄存器的地址：

```
#include<absacc.h>
#define COM8255 XBYTE[0x7FFF]
#define PA8255 XBYTE[0x7FFC]
#define PB8255 XBYTE[0x7FFD]
#define PC8255 XBYTE[0x7FFE]
```

在主函数的初始化部分，对 8255 的工作方式进行定义：

```
COM8255=0xXX;  /*XX 为 8255 方式字*/
```

其后，就可以用 PA8255、PB8255 或 PC8255 对 8255 的各 I/O 口进行操作了。

习题和思考题

1. 有一个 8031 单板机，它用一片 EPROM2732 存放监控核心程序(地址为 0000H～0FFFH)，用一片 E^2PROM2817 作为可由用户程序在线改写的用户程序存储器(地址为 1000H～1FFFH)，试画出这个系统的程序存储器框图。如果需要将此 2817E^2PROM 中开头的 256 个单元内容在线改写为零，请编写有关程序。
2. 在一个 8031 应用系统中，接有一片 8155 和 2K 字节 RAM(称为工作 RAM，地址为 7800H～7FFFH)，再通过 8155 接 16K 字节的后备 RAM 存储器。试画出这个系统的逻辑框图，并说明 8031 对后备 RAM 的读、写原理。

3. 在一个 f=11.0592MHz 的 8031 应用系统中，接有一片 8255(地址为 0BFFCH～0BFFFH)，其 PA 口接 8 个输入继电器，输入幅度为 5V、宽度不小于 50ms、前后沿有 1ms 抖动的脉冲信号，PB 口接 8 个指示灯。欲用 8031 的 T0 产生定时中断，将 PA 口输入的继电器稳态(去抖动)实时地送 PB 口显示，请编写有关的初始程序和 T0 中断服务程序。
4. 有一个 MCS--51 应用系统，需要 4K 字节程序存储器、两个 8 位输入口、两个 8 位输出口、四个外部中断源，试分别设计符合下列要求的系统框图：
   (1) 体积小、程序修改方便。
   (2) 价格低、程序修改方便。
5. 现有两个 8031 系统，若用两片 74LS373 作为双机通信数据口，用 8031 的 P1.0 和 INT0 作为通信联络线，试画出此双机通信的接口电路，并说明其通信原理。

第六章　单片机的功能扩展

单片机在众多的领域中得到了广泛的应用，如智能仪表，自动检测报警系统，家电自动化控制，自动机床，通信设备，生产过程自动化系统，交通管理控制系统等。在单片机应用系统中，单片机系统与被控制系统是相对独立的，它需要实时检测被控制系统的运行状态，按照一定的算法对被控制系统的状态进行分析处理，并按照一定的规则产生各种控制信号施加在被控制系统上，使被控制系统按规定的方式运行。

因此，单片机系统可以看作是一个信息处理系统。它的输入是被控制系统的状态数据，需要一个功能组件，对这些数据进行采集和预处理，这称为单片机系统的前向通道。单片机对被控制系统的状态数据分析处理之后，产生所需的控制信号，作进一步的处理后施加于被控制系统上，以改变被控制系统的状态。通常是对控制信号进行功率放大，并作隔离处理，这称为单片机系统的后向通道。单片机对被控制系统状态的处理，不仅要对被控制系统的状态进行控制，在很多情况下，还需要向操作人员显示报告系统的运行状态，在许多系统中，还需要为操作者设置键盘，用于对系统进行参数设置，这称为单片机系统的人机接口。

单片机处理系统的前向通道、后向通道和人机接口中，需要很多功能器件以完成相应的操作，这些功能器件是单片机所不具备的。增加这些功能器件，构建具有指定功能的单片机系统，称为单片机的功能扩展。

6.1　前向通道的配置与接口技术

6.1.1　单片机应用系统中的前向通道

1. 前向通道的含义

在单片机测、控系统中，外界系统总要有被测信号输出通道，由计算机拾取必要的输入信息。作为测试或监测系统，对被测对象拾取必要的原始参量信号是系统的核心任务；对控制系统来说，对被控对象状态的测试以及对控制条件的监测也是不可缺少的环节。

对被测状态的测试一般都离不开传感器或敏感元件，因为被测对象的状态参数大多是一些非电物理量，如温度、压力、位移等。因此，在前向通道中，传感器、敏感元件及其相关电路占有重要地位。

对传感器和敏感元件输出的模拟信号，需要把它们转换成满足计算机输入接口电平要求的数字信号。对外界已存在的开关量信号，如各种电源开关、触点、晶

体管开关、继电器等部件产生的信号，则应转换成 TTL 电平供计算机检测。

单片机应用系统的前向通道，是信号输入、传感和信号调节、交换的过程，是被测对象与系统相互联系的原始参数输入通道。

前向通道的主要任务就是忠实地反映被测对象的真实状态，包括两个方面：实时性和测量精度，同时使这些测量信号满足计算机输入接口的电平要求。

2. 前向通道的特点

1）检测电路因距被测对象很近，因而工作条件差，易受外界的干扰，是单片机系统中最重要的一个干扰进入渠道。

2）大多数传感器的输出是微弱的模拟信号，为将其转换成计算机接口电路的电平要求，需采用模拟电路技术，这处理起来有一定难度。

3. 前向通道的结构类型

前向通道是被测对象信号输出到单片机数据总线的输入通道，因此，其结构形式取决于被测对象的环境、输出信号的类型、数量、大小等。

传感器输出的信号种类可分为：电压信号、电流信号、频率信号、开关信号，如图 6-1 所示。

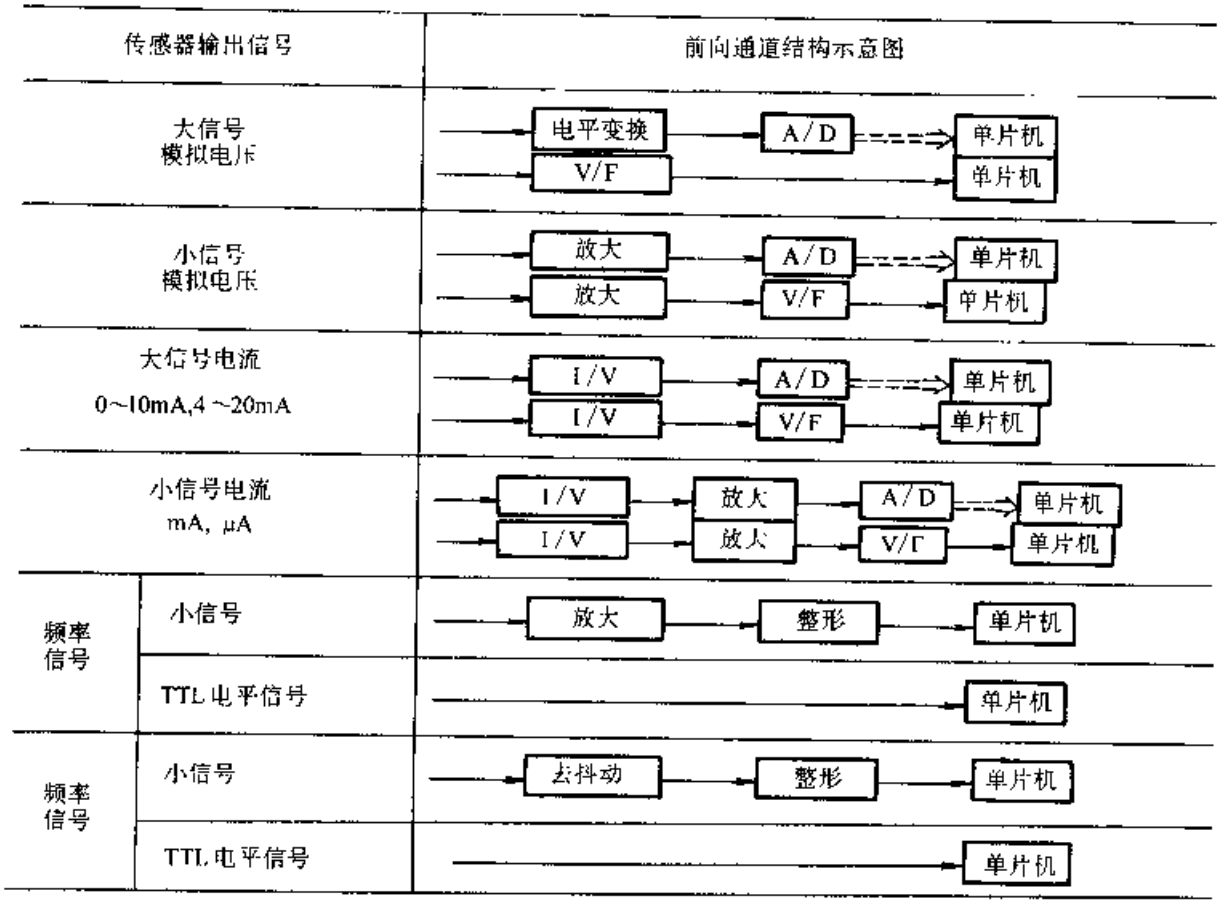


图 6-1　前向通道结构示意图

对于大信号模拟电压信号，只需经电平变换使之满足 A/D 输入的电压范围，即可输入给 A/D 转换器，经 A/D 转换后送入单片机，或经 V/F 变化成频率信号送入单片机。但 V/F 转换速度慢，多用于慢变过程参量的测量，其特点是抗干扰能力强，传输距离远。

对于小信号模拟电压信号，则需经放大电路进行放大、滤波，再经 A/D 转换后送入单片机，或经 V/F 转换后送入单片机。

当传感器所输出的电信号为电流信号时，则首先应通过 I/V 转换，放大到能满足 A/D 转换、V/F 转换要求的输入电压。最简单的 I/V 转换电路就是一个精密电阻。对于标准的 0～10mA 或 4～20mA 电流信号，选择合适阻值就可以直接获得能满足 A/D、V/F 转换输入要求的模拟电压信号，对于小电流信号则 I/V 变化后还需经放大才能送入 A/D 或转换电路。

对于频率信号或开关信号，只需把非 TTL 电平转换成 TTL 电平即可，而对于 TTL 电平的信号，则可直接送入单片机。

对于多路的模拟信号，则需使用多路开关使多个输入源共有一片 A/D 电路，如图 6-2 所示。

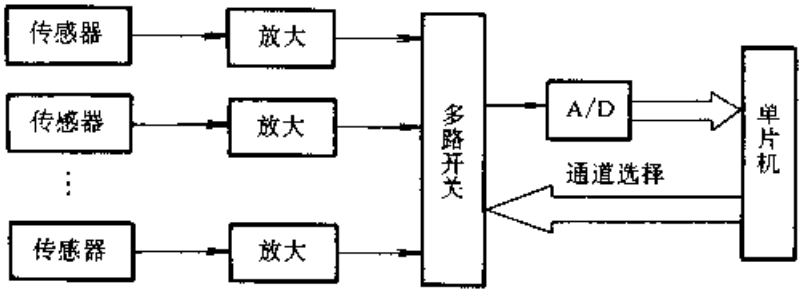


图 6-2　多路 A/D 转换结构图

4. 前向通道设计中应考虑的问题

前向通道是单片机应用系统的信号采集通道，包括信号的传感、转换到单片信号输入。故前向通道设计中必须考虑到信号拾取、信号调节、A/D 转换以及抗干扰问题。

(1) 信号拾取方式

在前向通道中，经常会遇到将压力、温度、速度、位移等非电量转换为电量的问题，也会有将强电量或微弱电量变换成适合计算机接口电路电平要求的信号的情况。

信号拾取这一环节可通过敏感元件、传感器和测量仪表来实现。

敏感元件体积小，随使用环境特点做成各种形状的探头。

传感器是用敏感元件及相应的测量电路、传递机构配以适当的外壳做成各种不同形状、适用于不同测量范围的传感器。传感器的输出一般为模拟电量或频率量。

近来的发展趋势是传感器的制作以适应计算机应用系统的要求为目标。

在热工、化工等部门，有各种调节仪表，一般都有大信号输出端，但售价高。

随着智能仪表的不断发展，新型传感器大都具有数字通信接口，与计算机的接口非常方便。

(2) 信号调节

信号调节的任务是将传感器和敏感元件输出的初次电信号转换成能满足单片机或 A/D 输入要求的标准电平信号。

信号调节的任务较复杂，有小信号放大、滤波、零点校正、线性化处理、温度补偿、误差修正、量程切换等。在单片机应用系统中，这些调节大多可用软件实现。因此，信号调节的重点为小信号放大、信号滤波以及对频率量的放大整形等。

(3) A/D 转换方式选择

选择芯片的主要要求有：转换精度、转换速度、模拟信号输入通道数及成本等。

不同的 A/D 芯片，其与单片机的接口电路要求不同，必须使接口电路满足 A/D 芯片的要求。

5. 抗干扰

在前向通道中，由于采取不同性质的器件，有时对电源的要求较为严格，一般前向通道配置的电源具有以下特点：

1) 小功率。

2) 高稳定度、高纯净度。

3) 有干扰隔离与抑制措施。

由于前向通道是系统干扰的主要渠道，因此，对其抗干扰有较高的要求，一般措施有：

1) 电源隔离，采用 DC/DC 转换器实现。

2) 模拟通道隔离，采用隔离放大器。

3) 数字通道隔离，通过电耦合器实现。

6.1.2 前向通道中的 A/D 转换与 A/D 转换器

A/D 转换接口是前向通道中的一个环节(具有模拟信号采集的应用系统)。数据采集和转换系统从一个或几个信号源中采集模拟信号，并将这些信号转换为数字形式，以便输入计算机。因此，一个模拟信号转换成数字信号所要求的基本部件有：

1) 模拟多路转换器与信号调节。

2) 采样/保持放大器。

3) 模拟/数字 (A/D) 转换器。

4）通道控制电路。

1. A/D 转换器的主要参数

（1）量化误差与分辨率

A/D 转换器的分辨率习惯上以输出二进制位数或 BCD 码位数表示。12 位 A/D 转换器 AD574A 的分辨率为

$$\frac{1}{2^n}\times 100\%=\frac{1}{2^{12}}\times 100\%=\frac{1}{4096}\times 100\%=0.0244\%$$

量化误差是由于用有限数字对模拟量进行离散取值（量化）而引起的误差。理论上量化误差等于分辨率。

（2）转换精度

A/D 转换器转换精度反映了一个实际 A/D 转换器在量化值上与一个理想 A/D 转换器进行模/数转换的差值，可表示为绝对误差和相对误差，其中不包含量化误差。

实际上对应于同一数字量，其模拟量输入不是固定值，而是一个范围。如理论上 5V 对应数字量 800H，而实际上 4.997V 到 4.99V 都产生数字量 800H，则绝对误差将是（4.997+4.99）V/2−5=−2mV。绝对误差经常用最小有效位 LSB 的分数值表示。

设 Δ 定义为数字量 D 的最小有效位（LSB）的当量。则如果模拟量 A 在 ±Δ/2 的范围内都产生相对惟一的数字量 D，则其绝对精度为 ± （1/2）LSB。

如在 A−3Δ/4～A+3Δ/4 范围都产生数字量 D，则其绝对精度为 ± （1/4）LSB。

图 6-3 是对上述说明的示意性图示。

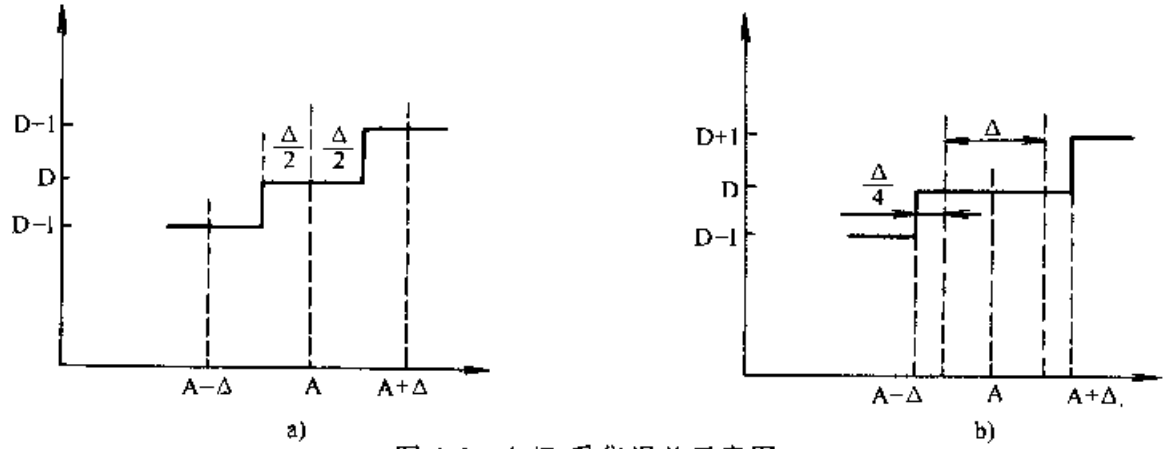


图 6-3 A/D 采集误差示意图

a）绝对精度为 ± （1/2）LSB b）绝对精度为 ± （1/4）LSB

（3）转换时间与转换速率

A/D 转换器完成一次转换所需要的时间为 A/D 转换时间，转换速率是转换时间的倒数。

通常衡量 A/D 转换速率的量称为通过率，它是 A/D 转换时间加上软件将 A/D 转换完的结果取入计算机内供使用的时间之和的倒数。

2. A/D 转换器及其应用特性

A/D 转换芯片种类繁多，按其变换原理分类，主要有逐次比较式、双积分式、量化反馈式和并行式 A/D 转换器。

(1) 逐次比较式转换器

逐次比较式转换器是目前种类最多、数量最大、应用最广的转换器件。主要有单片集成和混合集成两类，后者的性能指标高于前者。这里只介绍单片集成化逐次比较式转换器。

目前流行的单芯片集成式逐次比较式转换器有两种：一种是以双极型微电子工艺为基础的产品；另一种是以 CMOS 工艺为基础的产品。前者转换速度较高，一般在 1～40μs 范围内。单片集成式的分辨率通常为 8～13 位二进制量级。

单芯片集成化逐次比较式芯片主要有：

1) ADC0801～0805 型八位 CMOS A/D 转换器：片内有三态数据输出锁存器，与单片机可直接连接，单通道输入，转换时间约 100μs，精度最高的为 ADC0801，非线性误差为±(1/4) LSB，最差的为 ADC0804 和 ADC0805，非线性误差为±1LSB，单一 5V 电源电压供电。

2) ADC0808 系列多通道 8 位 CMOS A/D 转换器：ADC0808 系列的结构、性能与 ADC0801～0805 近似，芯片内设置了多路模拟开关及通道地址译码和锁存电路，能对多路模拟信号进行分时采集与转换。

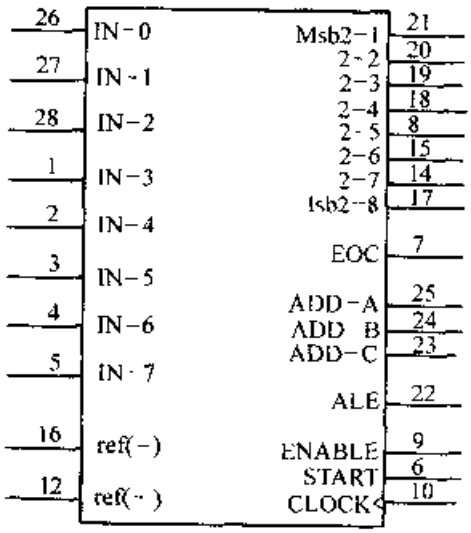


图 6-4 ADC0808 引脚图

ADC0808 系列芯片主要有 8 通道的 ADC0808/0809 和 16 通道的 ADC0816/0817。

ADC0808 引脚图见图 6-4。

图 6-4 各管脚名称及功能：

IN0～IN7 (26～28，1～5)　(I)：八路模拟信号输入端；

ADDA (25)，ADDB (24)，ADDC (23)；(I)：地址码输入；

D0 (LSB) (17)，D1 (14)，D2 (15)，

D3 (8)，D4～D7 (18～21)　(O)：数据输出；

ALE (22)　(I)：地址锁存信号，上升沿有效(与单片机 ALE 相反)；

OE (9)　(I)：输出有效控制，高有效；

CLOCK（10）　　(I)：时钟信号输入（典型为 640kHz)；

REF+（12）　　(I)：参考电压+端；

REF−（16）　　(I)：参考电压−端；

START（6）　　(I)：启动 A/D 转换；

EOC（7）　　(O)：A/D 转换结束，高有效；

VCC（11）　　(I)：+5V；

GND（13）　　(I)：地。

ADC0808 每一通道的转换时间为 66～73 个时钟脉冲，在 640kHz 时钟信号时，约 100μs。

ADC0808 的最大不可调整误差小于±（1/2）LSB，ADC0809 为±1LSB。

ADC0808 系列操作时序：ADC0808 系列内部含有三-八译码电路以控制分别选通八个模拟输入通道，见右表。

地址码			选通模拟通道
A	B	C	
0	0	0	IN0
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

故首先要确定 ADDA、ADDB 和 ADDC（一般用 P0 口的 P0.0、P0.1 和 P0.2，在 ALE 信号下将该地址信号锁存在 0808 内部的地址锁存器中，经译码以选通指定的模拟通道。然后在 START 引脚上给一个正脉冲以启动 A/D 开始转换。EOC 端上的电平在 A/D 转换期间为低，转换完成后变成高电平，可作为查询或中断信号使用。当 EOC 变高后，在 OE 引脚上给高电平使转换结果出现在数据总线 D0～D7 上，当 OE 为低电平时，D0～D7 为高阻状态。

这里，从启动 A/D 和读取 A/D 结果的信号组合（图 6-5）中可以看到，尽管 ADC0808 系列没有片选信号，但 P2.7 控制了$\overline{RD}$和$\overline{WR}$信号的传输，起到了片选的作用。这时写 A/D（启动 A/D 转换）的地址高八位 P2 为 7FH，低八位的 P0 中的高五位为全 1，低 3 位依八路模拟通道依次为 000～111，即地址为 7FF8H～7FFFH。读 A/D 转换结果地址为 7FFFH（低八位任意）。

```
启动 A/D：XBYTE [0x7FF8] =0;          /* 第 0 通道 */
                                      /* 只需WR信号与 P2.7 */
读 A/D：   x=XBYTE [0x7FFF];
```

3）12 位 A/D 变换器 AD574A

AD574A 是一种快速 12 位逐次比较式 A/D 变换器，是单通道变换器。具有三态数据输出锁存器。非线性误差小于±（1/2）LSB，一次转换时间为 25μs，电源

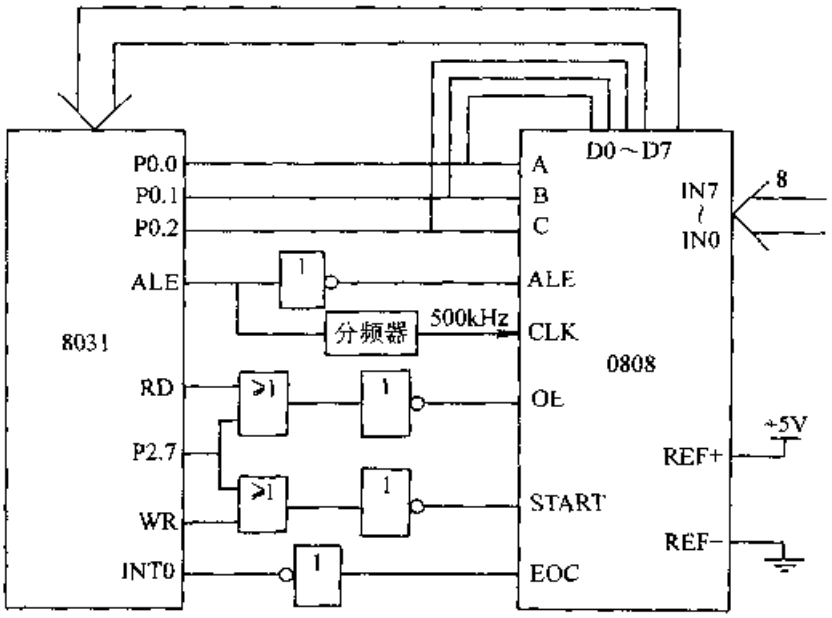


图 6-5　ADC0808 与 8031 的连接

供电为±15V（±12V）和+5V。

AD574A 是由模拟芯片和数字芯片混合封装而成的，模拟芯片为一 12 位快速 A/D 转换器，数字芯片包括高性能比较器、逐次比较逻辑寄存器、时钟电路、逻辑控制电路以及三态输出数据锁存器等。

AD574A 引脚排列及功能：

$\overline{CS}$	(3)：	片选；
$\overline{CE}$	(6)：	使能；
R/$\overline{C}$	(5)：	启动/读数控制　0:启动　1:读数；
A0	(4)：	转换字长/数据格式控制。A0＝0 时启动 A/D 变换，则按 12 位 A/D 方式工作；A0＝1 时启动 A/D，则按 8 位 A/D 方式工作；
12/$\overline{8}$	(2)：	数据格式控制，与 A0 共用。12/$\overline{8}$＝1 时，按 12 位并行输出；12/$\overline{8}$＝0 时，按 8 位双字节输出。A0＝0 时输出高 8 位；A0＝1 时输出低 4 位，并以 4 个 0 补足尾随的 4 位。该位不与 TTL 电平兼容，必须直接接地或+5V。另外，在数据输出期间，A0 不能变化；
STS	(28)：	转换/完成状态输出。STS＝1 表示正处于转换中，STS＝0 表示转换完成；
REFIN	(10)：	参考输入；
REFOUT	(8)：	参考输出。输出+10V 电压；
BIPOFF	(12)：	双极性输入偏置；
$10V_{IN}$	(13)：	

$20V_{IN}$　(14)：　10V 档输入端。供 0～10V 和－5～＋5V 输入用；

D0～D11　(16～27)：20V 档输入端。供 0～20V 和－10～＋10V 输入用；

－12(－15V)　(11)：　12 根输出数据线；

12(15V)　(7)

V_{CC}(＋5V)　(1)

DGND　(15)　数字地；

AGND　(9)　模拟地。

AD574A 具有四种输入方式：

1）单极性 0～10V　$10V_{IN}$入

2）单极性 0～20V　$20V_{IN}$入

3）双极性－5～＋5V　$10V_{IN}$入

4）双极性－10～＋10V　$20V_{IN}$入

AD574A 在单极性和双极性输入时其参数输入具有不同的形式，分别见图 6-6 和图 6-7。

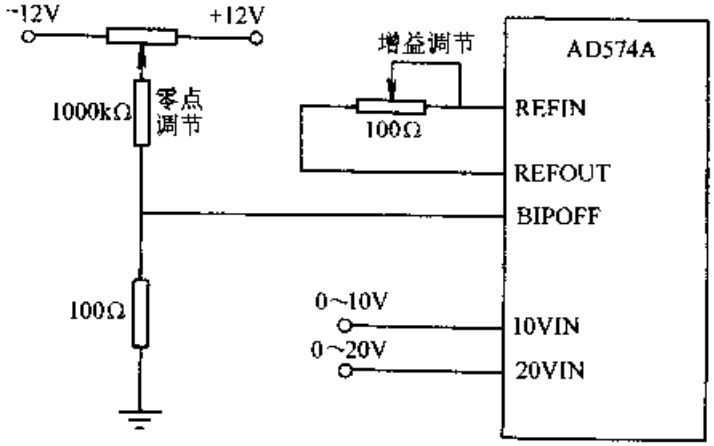


图 6-6　单极性参数输入时

AD574A 的逻辑控制输入信号有$\overline{CS}$、$\overline{CE}$、R/$\overline{C}$、12/$\overline{8}$、A0，用以对 AD574A 控制启动和输出。其控制真值表为：

$\overline{CS}$	$\overline{CE}$	R/$\overline{C}$	12/$\overline{8}$	A0	工作状态
×	0	×	×	×	禁止
1		×	×	×	禁止
0	1	0	×	0	启动 12 位转换
0	1	0	×	1	启动 8 位转换
0	1	1	接 1 脚（＋5V）	×	12 位并行输出
0	1	1	接 15 脚（0V）	0	高 8 位输出
0	1	1	接 15 脚（0V）	1	低 4 位输出

AD574A 与 8031 的连接见图 6-8，图中启动变换的地址为 0x7FFFC，当 P1.0 由高转低时，A/D 变换结束。

读 A/D 高 8 位的地址为 0x7FFD。

读 A/D 低 8 位的地址为 0x7FFFH。

(2) 双积分式 A/D 转换器

双积分式 A/D 变换是一种间接 A/D 转换技术，先将模拟电压转换成积分时间，再用数字脉冲计时方法转换成计数脉冲数，最后将计数用二进制或 BCD 码形式输出。其转换时间较长：大于 40～50ms。

在数字万用表中，使用的就是双积分式 A/D 转换器。

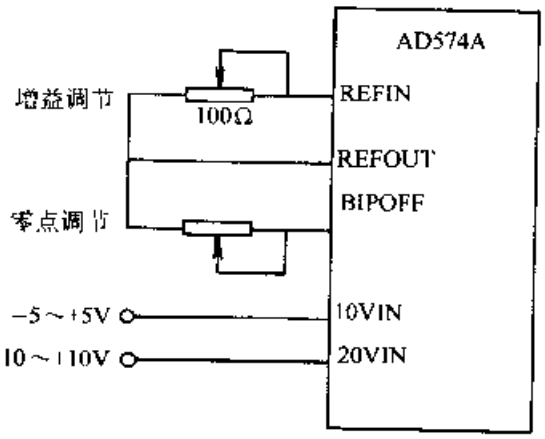


图 6-7　双极性参数输入时

常用的双积分式 A/D 转换器有：

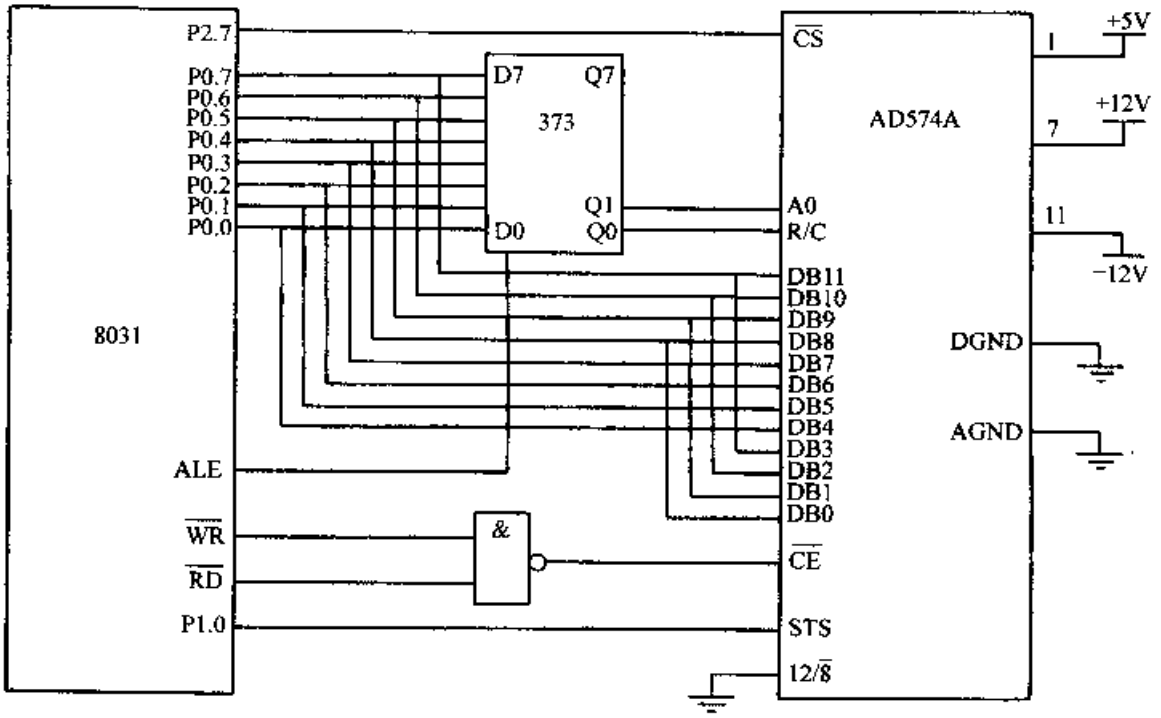


图 6-8　AD574A 与 8031 的连接

MC14433　(3 (1/2) 位)

ICL7106　(3 (1/2) 位) BCD 输出　个十百千 (千位只能输出 1，0)

ICL7135　(4 (1/2) 位)

AD7555　(5 (1/2) 位)

(3) 并行式 A/D 转换

并行式 A/D 转换又称瞬时比较编码式 A/D 变换。这是一种转换速度较快、转

换原理较直观的 A/D 变换技术。目前主要产品有：

		采样速率（每秒采样兆次数）
TDC1007J	8 位	20MS/s
TDC1019J	9 位	20MS/s
TDC1029J	6 位	100MS/s
CA3308	8 位	15MS/s
MC10315L	7 位	15MS/s

3. A/D 转换中的应用问题

前向通道中，A/D 转换直接通过信号调节电路与信号对象相连。由于信号特征不同，A/D 转换器存在着各种应用问题。如极性变换、输入保护、多路采集、采样保持、高精度化问题。

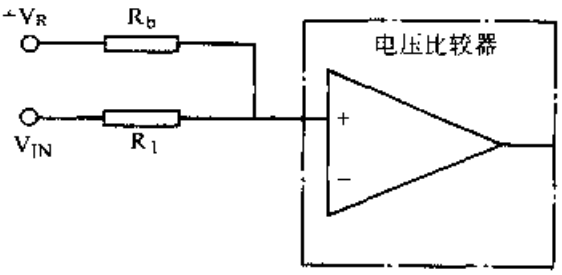


图 6-9　双极性输入偏置电路

（1）模拟信号的输入极性变换

A/D 转换器输入的模拟信号有单极性（0～5V 或 0～10V）和双极性（±2.5V，±5V，±10V 等）。因此要求 A/D 转换器能够处理双极性模拟信号。

通常双积分式 A/D 转换器中都设有正负参考电压，适当地设置参考电压的极性，可设定输入为单极性或双极性。图 6-9 所示为双极性输入偏置电路。

对于逐次比较式 A/D 转换器，其模拟输入通道形式较多，如 ADC0801～0805 具有完全差分输入形式；0809 则为多通道单极性输入；AD574A 设置有输入偏置电阻，因而可以工作于单极性输入形式和双极性输入方式。

对于只能工作于单极性模拟电压的 A/D 转换器，可以在外部提供偏置电路，因为在其模拟输入端允许进行电流或电压的叠加。

设 A/D 转换器最大输入模拟电压值为 V_{NFS}，则外加偏置电压 V_R 和电阻 R_b 的大小按下式计算

$$V_R \frac{R_1}{R_1+R_b}=\frac{V_{NFS}}{2}$$

其中 R_1 包含信号源的输出阻抗。

例如 ADC0809 的输入电压范围为 0～5V，为了能用于±5V 的模拟输入电压，其偏置电路如图 6-10 所示。

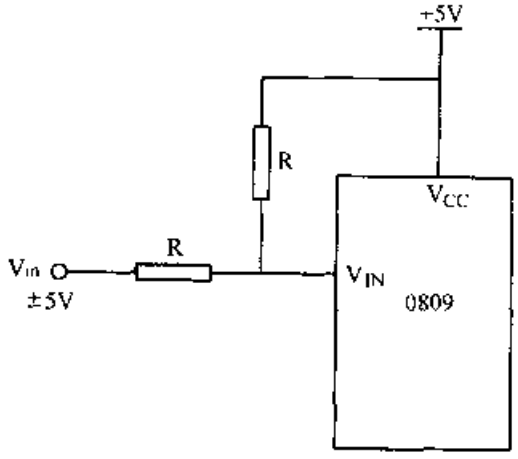


图 6-10　ADC0809 的双极性偏置电路

这时，对于－5V 输入，V_{IN}端电压为 0V，采样编码为 00H；0V 输入时，V_{IN}端电

压为 2.5V，采样编码为 80H；5V 输入时，V_{IN}端为 5V，采样编码为 FFH。

（2）模拟信号的多路输入

在实际应用系统中，常常要使用一个 A/D 转换器实现对多个模拟通道的转换。有些 A/D 转换器本身就具有多路模拟输入通道，如 ADC0808、ADC0816 等。这样的 A/D 转换器内部具有多路开关。

不是所有的 A/D 转换器都具有多路模拟输入能力。对于单输入通道的 A/D 转换器，要用于多路模拟信号采集时，必须使用多路模拟开关电路来扩展模拟输入通道。

在前向通道中常使用的模拟多路开关是 CMOS 多路开关。典型的多路开关有四选一、双四选一、八选一、双八选一和十六选一五种类型。CMOS 模拟开关的等效电路如图 6-11 所示。图中开关 S 由其内部的译码电路控制。流经检流计的电流为纳安（nA）级。

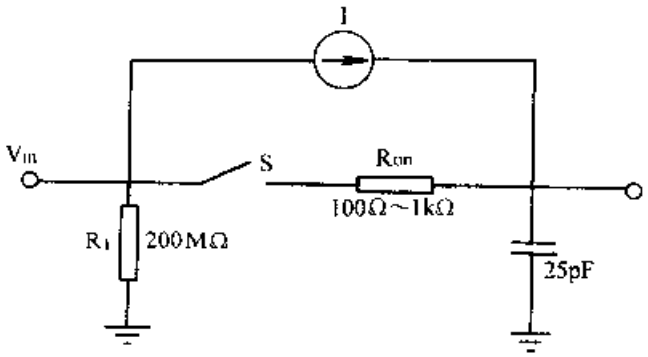
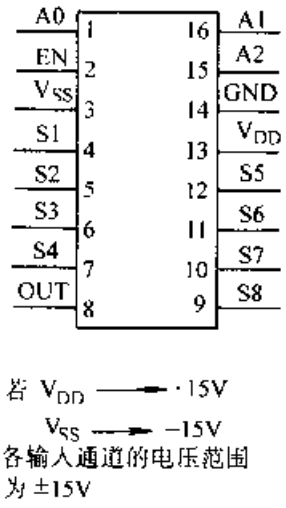


图 6-11　CMOS 模拟开关等效电路

常用的多路开关有 CD4051、AD7503、LF13508、AD7506 等。CD4051 的模拟信号范围为±7.5V。LF13508 能适合更大范围的模拟信号。LF13508 引脚图及操作真值表见图 6-12。它由三根地址线的译码值控制，EN 为使能端。



EN	A0	A1	A2	通道
0	×	×	×	无通道
1	0	0	0	S1
1	0	0	1	S2
1	0	1	0	S3
1	0	1	1	S4
1	1	0	0	S5
1	1	0	1	S6
1	1	1	0	S7
1	1	1	1	S8

图 6-12　LF13508 引脚图及操作真值表

（3）采样/保持器

采样/保持器是在输入逻辑电平控制下，处于“采样”或“保持”两种状态的电路。在采样状态下，电路的输出跟踪输入模拟电压，转为保持状态时，电路输出保持状态改变时的模拟信号电平，直至进入下一次采样状态为止。

由于模拟量转换成数字量有一过程，对于 A/D 转换器，存在一孔径时间，即启动 A/D 至 A/D 转换器捕捉到模拟信号的时间。对于动态模拟输入信号，由于转换器孔径时间的存在，会产生孔径误差，如图 6-13 所示。

图 6-13　孔径误差示意图

最大孔径误差总是发生在信号变化率最大的地方。

为了减少孔径时间，对于 12 位转换器，在采样信号的频率大于 10Hz 时，都应该采用采样/保持器。

最基本的采样/保持器由模拟开关 S、保持电容 C_H 和缓冲放大器组成，如图 6-14 所示。当控制信号 V_C 为采样电平时，S 导通，保持电容充电，这时，输出电压 V_O 跟踪输入电压变化；当控制信号 V_C 为保持电平时，S 断开，输出电压保持在开关断开瞬间的输入信号值。

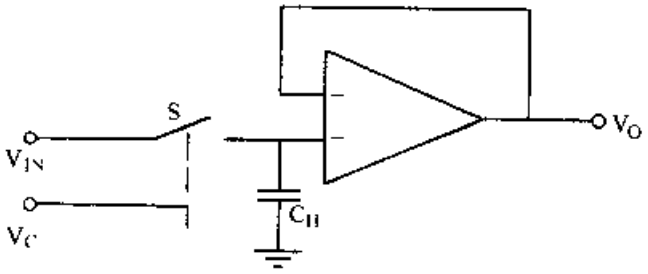


图 6-14　采样/保持器原理图

LF398 是一种常用的采样/保持器，是一种反馈型采样/保持电路，其引脚排列如图 6-15 所示。

LF398 输入阻抗高、转换速度快，当逻辑控制 V_C 为高电平时，处于跟随状态，V_C 为低电平时，处于保持状态。LF398 典型应用电路如图 6-16 所示。

(4) 软件编程的特点

A/D 转换是对观测变量进行有规律的采样，采样周期是固定的，因而必须使用定时器。当定时器溢出中断时，启动 A/D 转换器开始 A/D 转换，A/D 转化完成后，A/D 转换器产生中断信号，CPU 读取 A/D 转换结果。由处理程序对数据作处理，这就完成一次 A/D 采样。随着定时器的有规律的溢出中断，A/D 采样周而复始地进行。

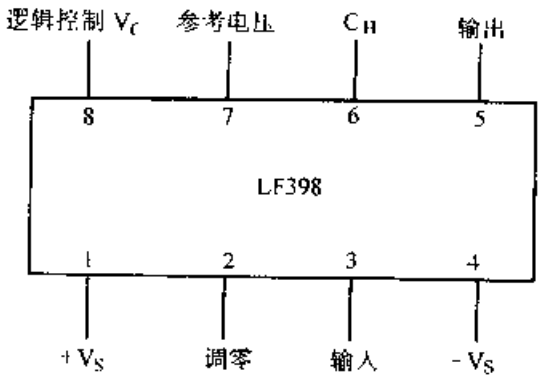


图 6-15　LF398 引脚排列

A/D 转换器的转换完成信号在转换期

间是一种状态，转换完成后变成另一种状态，并保持到下一次转换开始。因而，外部中断应采用下降沿触发方式，即令 IT0 或 IT1 为 1。

对于多路数据采集，对每一路数据通道应使用采样保持器，当定时器溢出中断时，使能采样保持，使各路数据都保持同一时刻的数据值，然后启动 A/D 转换器，对各路数据依次采样。

[课程实践]：采用 ADC0809A/D 转换器对 8 路模拟信号进行采样，采样周期为 0.5s，对各路信号采用采样保持器。各路信号的采样值存入一数组中，每次采样完毕设置一标志，通知主流程。画出系统硬件线路图，编写程序。

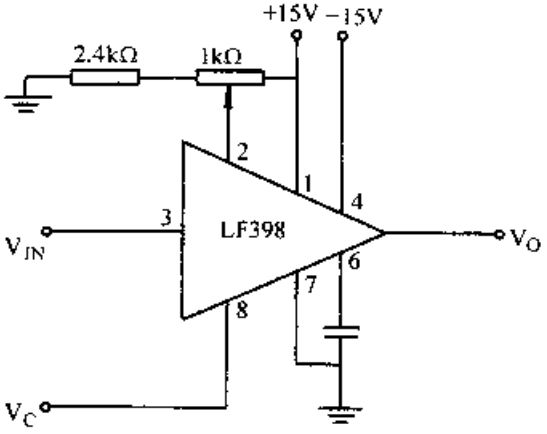


图 6-16　LF398 典型应用电路

6.2　后向通道的配置与接口技术

6.2.1　单片机应用系统中的后向通道

在工业控制系统中，单片机总要对控制对象实施控制操作。后向通道是计算机完成控制运算处理后，对控制对象的控制输出通道接口。它的结构与特点和控制对象密切相关。

1. 后向通道的特点

根据单片机的输出和受控对象对控制信号的要求，后向通道具有以下特点：

1）小信号输出、大功率控制。

2）它是一个输出通道，输出伺服驱动控制信号。而伺服驱动系统中的状态反馈信号作为检测对象输入到前向通道。

3）接近受控对象，环境中干扰较为严重，且易从伺服驱动系统的状态信号检测电路进入前向通道。

2. 后向通道的结构

根据单片机的输出信号形式和受控对象的特点，后向通道的结构如图 6-17 所示。

单片机在完成控制处理后，总是以数字信号通过 I/O 口或数据总线送给控制对象。这些数字信号形态主要有开关量、二进制数字量和频率量，可直接用于开关量、数字量系统及频率调制系统，但对于一些模拟量控制系统，则应通过数/模转换或 F/V 转换变换成模拟量控制信号。

3. 后向通道应解决的问题

1）功率驱动，将单片机输出的信号进行功率放大，以满足伺服驱动的功率要

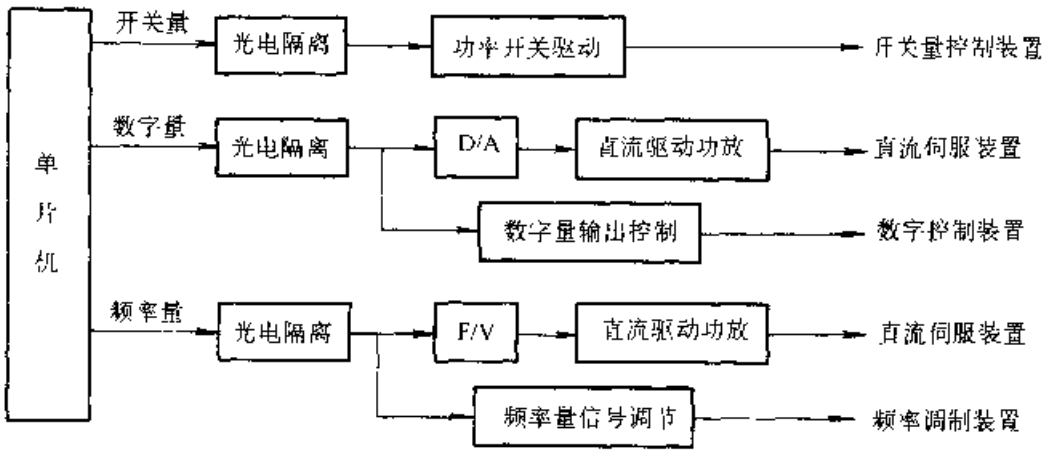


图 6-17　后向通道的结构图

求。

2）数据类型转换，D/A 或 F/V。

3）干扰防治，通常采用信号隔离、电源隔离和对大功率开关实现过零切换等方法。

6.2.2　后向通道中的常用器件及电路

后向通道中常用的器件及电路主要有数/模转换、功率驱动和干扰防治器件和电路。

1. 功率开关接口器件及电路

在实际的单片机应用系统中，大量使用的是开关型驱动控制。常用的功率开关接口器件及电路有：功率开关驱动电路、功率型光耦合器、集成驱动芯片及固态继电器等。

（1）大功率 I/O 接口电路

在单片机应用系统中，开关量都是通过单片机的 I/O 口或扩展 I/O 口输出的，这些 I/O 口的驱动能力有限。对于标准 TTL 门电路，在 0V 电平时吸收电流的能力约为 16mA，而采用集电极开路驱动器的 74 系列芯片，如 7438，可吸收 48mA 电流，其驱动能力依然有限。在后向通道的一些大功率开关量控制接口中，常采用下列功率开关电路：

1）功率晶体管驱动：当晶体管用作开关元件时，其输出电流为输入电流乘以晶体管的增益。典型的开关功率晶体管具有高速、中等功率特性，其驱动电流可达 800mA，击穿电压小于 60V，在输出 500mA 时，典型正向电流增益为 30，因此要开关 500mA 负载电流时，其基极电流至少需要供给 17mA，一般基极驱动电流的取法为所需基流的两倍，见图 6-18。

2）达林顿驱动电路：用单个开关晶体管驱动时，有时其基极驱动电流还是较大，为了减小基极驱动电流，可以把多个开关管级联起来提高正向电流增益。达

林顿晶体管即是用两个晶体管构成一个整体，见图 6-19。这种结构形式具有高的输入阻抗和高的电流增益。

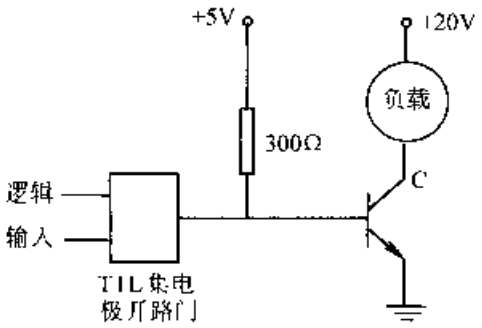


图 6-18　简单晶体管驱动器

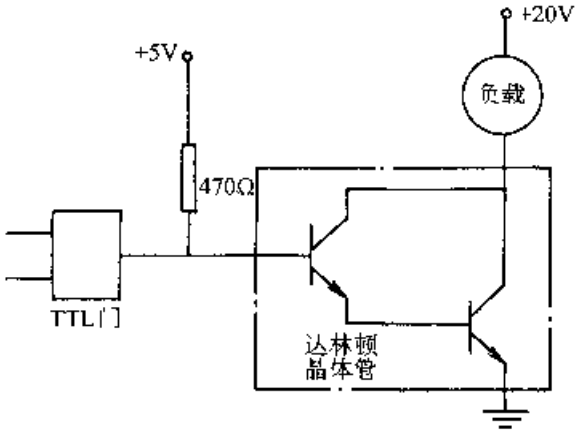


图 6-19　达林顿驱动器

3）晶闸管驱动电路：晶闸管整流器（VH）是一种三端固态器件，其阳极相当于晶体管的集电极，阴极相当于发射极，门控极相当基极。晶闸管整流器只工作在导通或截止状态，故可作为开关器件。

使 VH 导通只需要极小的驱动电流，一般输出负载电流和输入驱动电流之比大于 1000。VH 一旦导通便无法截止，除非切断负载电流。因而 VH 一般只用在交流功率开关电路中，因为交流电在每半个周期过零一次，VH 截止不成问题。

由于 VH 通常用于控制交流大电压负载，故不宜直接与数字逻辑电路相连，在实际应用中一般采用光耦合隔离措施，见图 6-20。

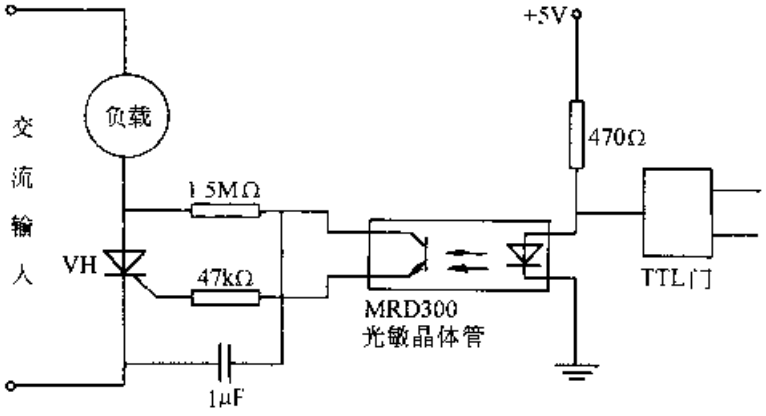


图 6-20　光隔离晶闸管

4）机械继电器：在数字逻辑电路中最常使用的机械继电器有簧式继电器。簧式继电器由一个线圈和两个磁性簧片组成。当线圈通电时，产生磁场，受磁场作用，两个簧片相接而导通。这种簧式继电器控制电流较小，而簧式触点可开关较大电流。如控制线圈内阻为 380Ω 时，由 5V 电源提供电流，则驱动电流为 13mA，而簧片则可流过近 1A 的电流。其控制电路如图 6-21 所示。

触点两端的稳压管用来防止产生触点电弧。线圈上并接的二极管为限制反向电路的产生。

机械继电器的开关响应时间较大，单片机程序设计时应考虑开关响应时间的影响。

5）功率场效应管(VMOS)：功率场效应管的特点是：属电压驱动型元件，驱动电流小，输出电流大，开关频率高。其电路如图 6-22 所示。

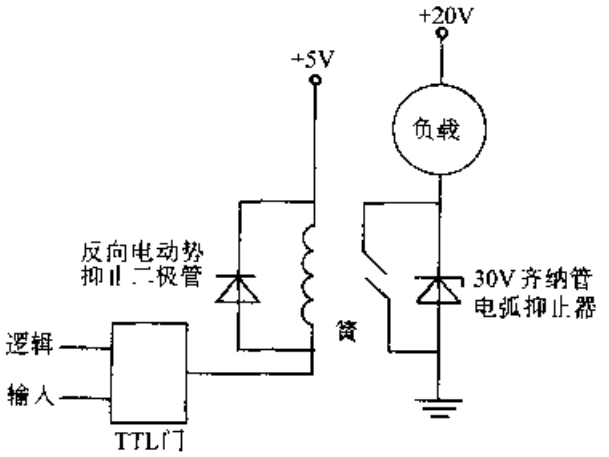


图 6-21　簧式继电器

（2）固态继电器（SSR）

固态继电器是一种无触点通断功率型电子开关，又名固态开关。当施加触发信号后其主回路呈导通状态，无信号时呈阻断状态。它利用了分离器件集成器件及微电子技术实现了控制回路（输入端）与负载回路（输出端）之间的电隔离及信号耦合，没有任何部件或触点，实现了具有相当于电磁继电器一样的功能。

固态继电器通常是一个四端组件，两个为输入端，两个为输出端。它由输入电路、隔离部分和输出电路组成。

固态继电器有许多种类可供选择：按输出功能分有直流型、过零型、非过零型；按隔离方式分有光隔离和变压器隔离；按封装形式分有塑封型和金属壳密封型，还有各种特殊用途的固态继电器。

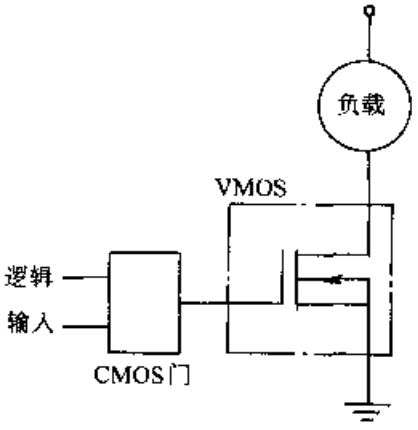


图 6-22　MOSFET 驱动器

固态继电器的特性：

固态继电器是由固态元件组成的无触点开关器件，因而比电磁继电器工作可靠、寿命长，对外界干扰小，能与逻辑电路兼容，抗干扰能力强，开关速度快。

使用固态继电器时，应注意以下特性：

1）根据产品的不同功能，输出电路可接直流或交流。对交流负载的控制有过零和不过零控制功能。其控制波形如图 6-23 所示。

2）由于固态继电器是一种电子开关，故有一定的通态压降和断态漏电流，其数值与型号规格有关。

3）负载短路易造成 SSR 的损坏，应特别注意避免。

4）必须考虑瞬态过电压和断态 dv/dt 对 SSR 的影响。

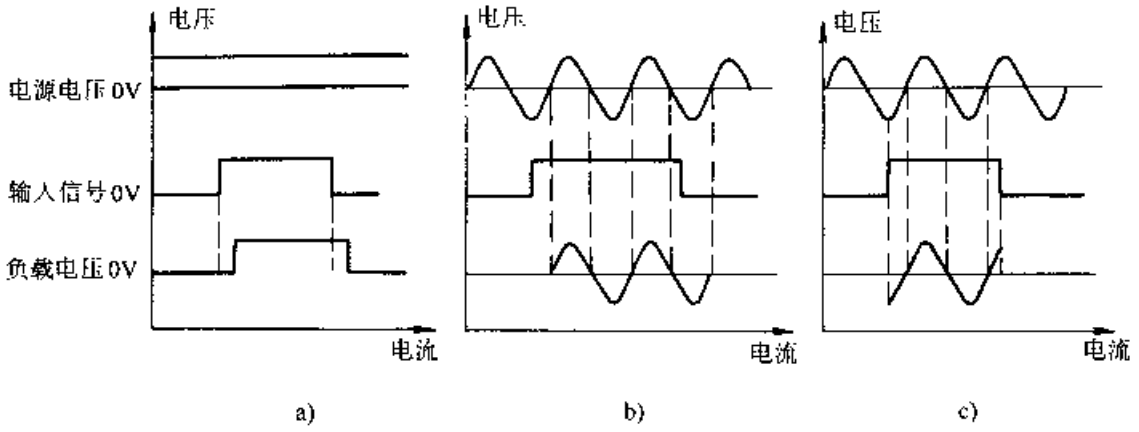


图 6-23　不同功能的固态继电器控制波形

a）直流型　b）过零型　c）非过零型

固态继电器的应用：

许多固态继电器输入回路要求电压低电流小，可用 TTL 门直接驱动（图 6-24a），用 CMOS 门时应加驱动（图 6-24b）。

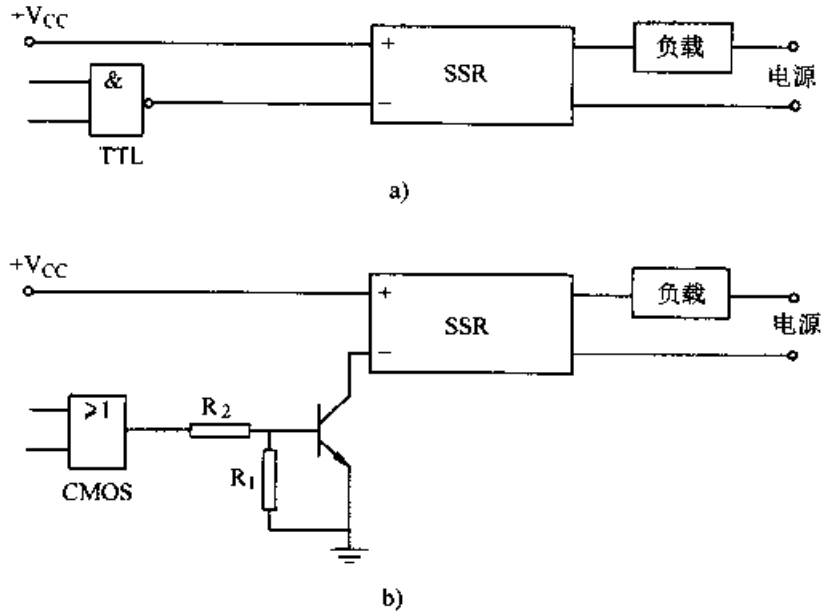


图 6-24　集成电路输入控制方式

2. 光隔离与接口驱动器件

光耦合器件能可靠地实现信号的隔离，并易构成各种功能状态，如信号隔离、隔离驱动、远距离传送等，故在应用系统的后向通道中经常使用各种类型的光耦合器件。

（1）信号隔离用光耦合器件

最普通的信号隔离用光耦合器件如图 6-25 所示。

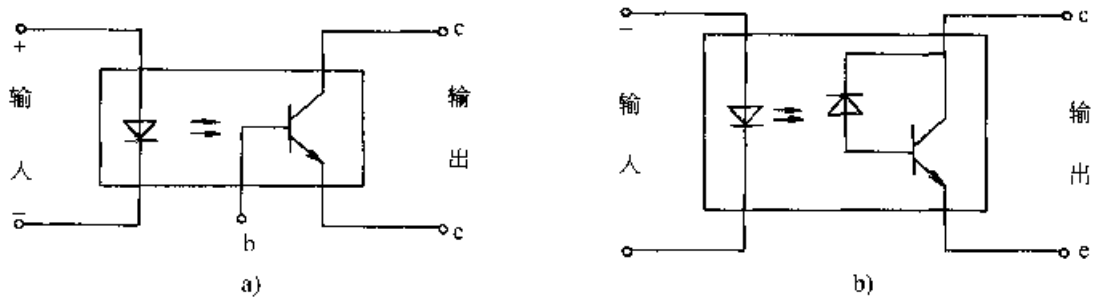


图 6-25　信号隔离用光耦合器件

a）普通型　b）高速型

以发光二极管为输入端，光敏三极管为输出端构成普通型光耦合器，如图 6-25a 所示。这种器件一般用在 100kHz 以下的频率信号。如果基极有引出线则可用于温度补偿、检测及调制。

高速光耦的输出部分采用 PUV 光敏二极管和高速开关管组成复合结构，具有较高的响应速度，如图 6-25b 所示。

（2）隔离驱动用光耦合器件

常用两种形式如图 6-26 所示。

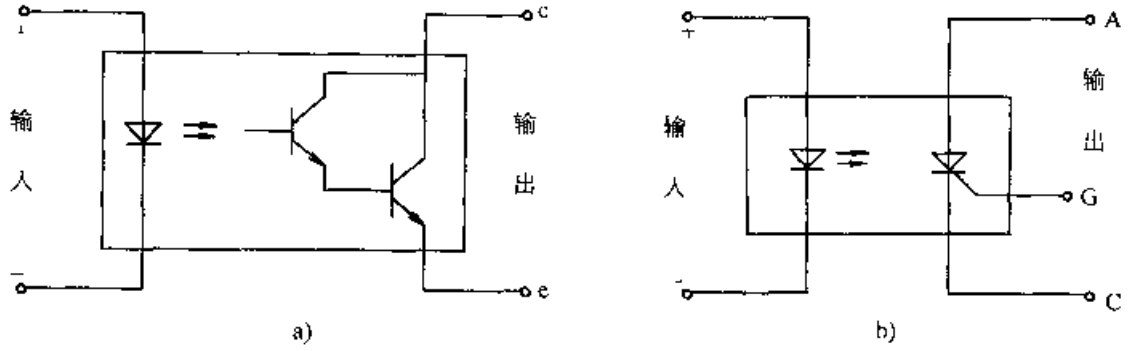


图 6-26　隔离驱动用光耦合器

a）达林顿输出　b）晶闸管输出

1）达林顿输出光耦合器件（图 6-26a），用于较低频率的负载。

2）晶闸管输出光耦合器件（图 6-26b）。光控晶闸管有单向双向两种形式，常用在交流大功率的隔离驱动。

（3）远距离的隔离传送

当要远距离控制一受控对象时，采用光耦合的隔离传送十分方便，其电路如图 6-27 所示。

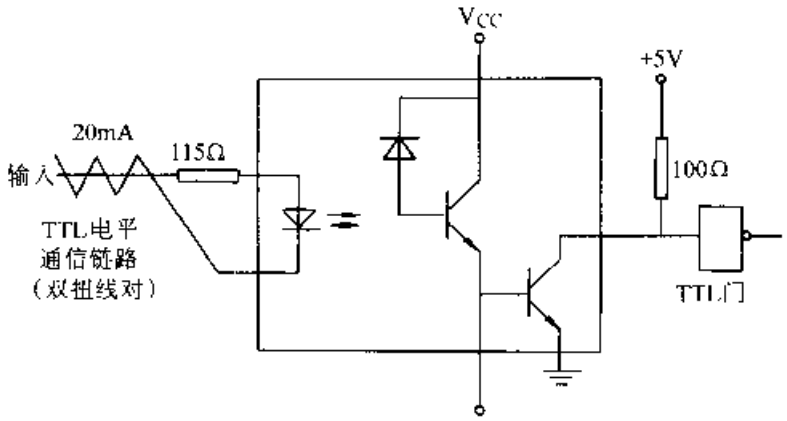


图 6 27　远距离的光隔离传送

6.2.3　后向通道中的 D/A 转换技术及其接口芯片

D/A 转换是应用系统后向通道的典型接口技术，实现数字量至模拟量的线性转换。

D/A 转换接口设计的一般性问题。在 D/A 转换接口设计中主要考虑的问题是 D/A 转换芯片的选择、数字量的码输入及模拟量的极性输出，参考电流源、模拟电量输出的调整与分配等。

1. D/A 转换芯片的选取原则

主要考虑性能结构及应用特性。

(1) 主要性能指标

给定工作条件下的静态指标，包括各项精度指标、动态指标建立时间和尖峰参数、环境指标、温度系数（手册上会给出）。使用时主要考虑以位数表现的转换精度和转换时间。

(2) 主要结构特性与应用特性

1) 数字输入特性：目前的转换芯片一般都只接收自然二进制数字代码。输入数据格式一般为并行码，有些能接收串行码输入。

2) 模拟输出特性：目前大多数 D/A 转换器件均属电流输出器件。手册上通常给出在规定输入参考电压及参考电阻之下的满码（全 1）输出电流，最大输出短路电流以及输出电压允许范围。在输出端的电压在输出允许范围以内时，输出电流和输入数字之间保持正确的转换关系，而与输出端的电压无关。

3) 锁存特性及转换控制：有没有对输入量的锁存功能会直接影响其与 CPU 的接口设计。

有些 D/A 需有外部信号控制才开始进行 D/A 转换，这样 CPU 可以控制多路 D/A 的同步输出。

4) 参考源：D/A 转换中，参考电压源是惟一影响输出结果的模拟参数。使用内部带有低漂移精密参考电压源的 D/A 转换能保证有较好的转换精度，并且可以

简化接口电路。

2. 参考电压源的配置

目前在D/A转换接口中常用的D/A转换器大多不带参考电压源，为了方便地改变输出模拟电压范围、极性，需配置相应的参考电压源。常用电路见图6-28。

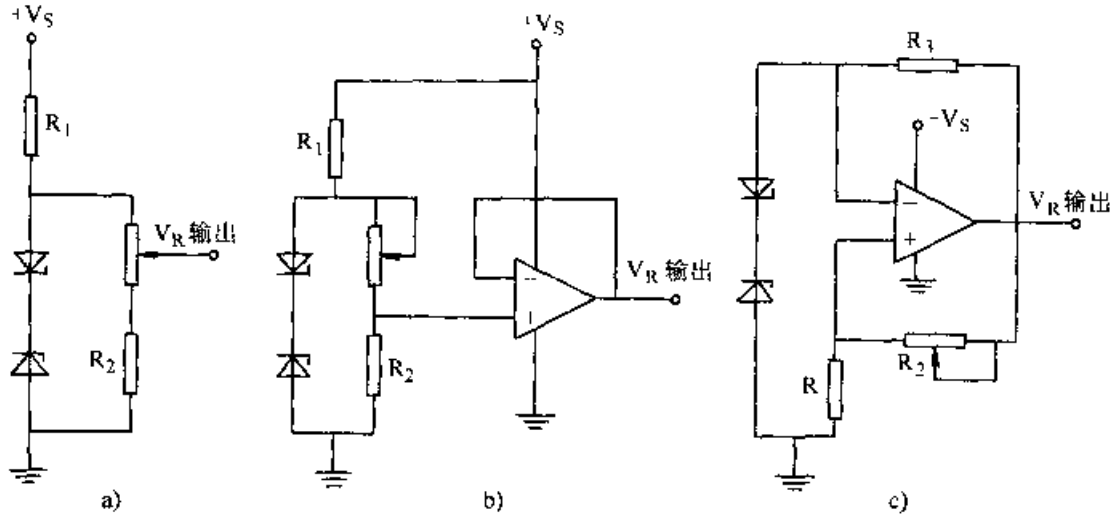


图6-28 D/A转换接口中常用的几种参考电压源电路

a）简单稳压电路 b)、c）带运算放大器的稳压电路

3. 数字输入码与模拟输出电压的变换特性

所有的D/A变换器件的输出模拟电压V_O，都可以表达成为输入数字量D（数字代码）和模拟参考电压的乘积：$V_O=D\times V_R$。

二进制代码可以表示为

$$D=a_1\times 2^{-1}+a_2\times 2^{-2}+\cdots+a_n\times 2^{-n} \quad (a_i=0,\ 1)$$

式中，a_1为最高有效位（MSB）；a_n为最低有效位（LSB）。

由于目前大多数D/A输出的模拟量均为电流量，这个电流量要通过一个反相输入的运算放大量才能转换成模拟电压输出，如图6-29所示（图中以7520为例）。

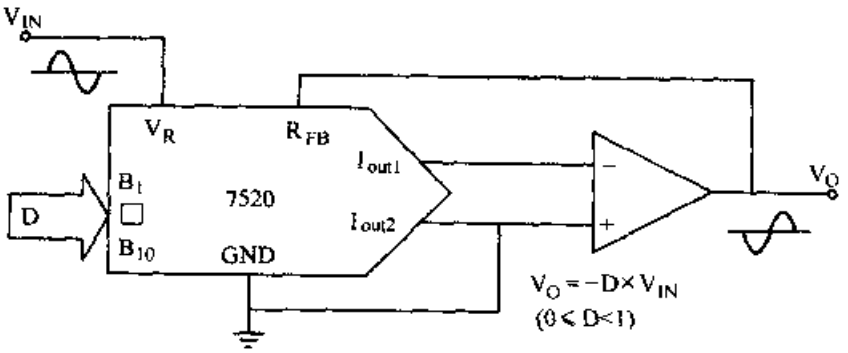


图6-29 两象限工作的D/A转换器的输出特性

在这种情况下，模拟输出电压 V_O 表示为

$$V_O = -D \times V_R \quad (0 \leqslant D \leqslant 1)$$

这种工作方式称为二象限方式，即单值数字量 D 和正负参考电压 V_R。输出电压 V_O 的极性完全取决于模拟参考电压的极性。当参考电压极性不变时，只能得到单极性的模拟电压输出。

当参考电压 V_R 极性不变时，要想得到双极性的模拟电压输出，必须采用四象限工作方式，见图 6-30 所示。

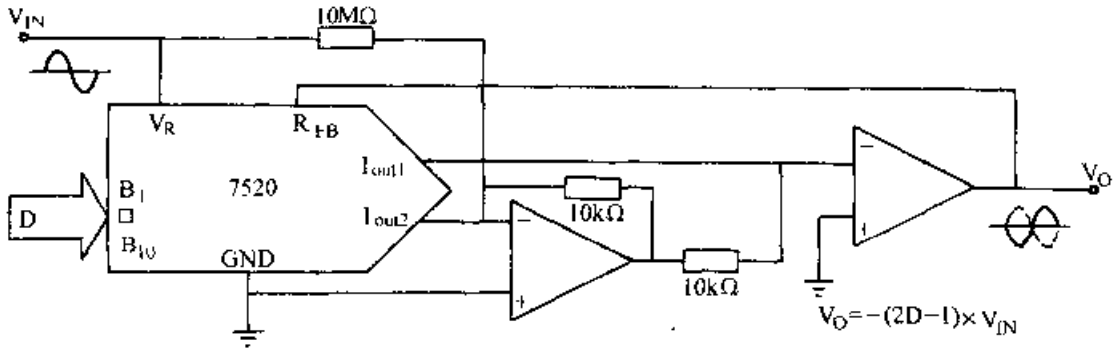


图 6-30　四象限工作的 D/A 接口电路

这时，当 V_R 为单极性时，输出电压 V_O 的极性为双极性的。

4．8 位 D/A 转换芯片 DAC0832

DAC0832 是较常使用的 8 位 D/A 变换芯片，其结构框图如图 6-31 所示。

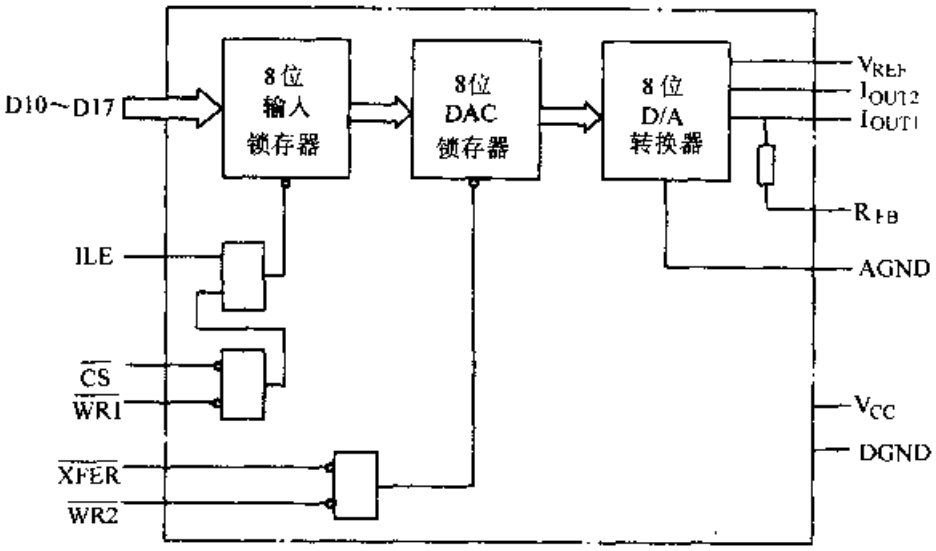


图 6-31　DAC0832 结构框图

DAC0832 有两级锁存控制功能，能够实现多通道 D/A 的同步输出。当只有一路 D/A 输出时，可以将$\overline{CS}$和$\overline{XFER}$相连作片选信号，将$\overline{WR1}$与$\overline{WR2}$相连作输出控

制，这样，两级锁存就成为一级锁存。

DAC0832 芯片为 20 引脚双列直插式封装结构，其引脚分配见图 6-32。

各管脚的功能如下：

D10～7：8 位数据输入总线。

ILE：数据允许锁存信号，一般接 V_{CC}。

$\overline{CS}$：输入锁存器选择信号。

$\overline{WR1}$：输入锁存器写控制信号。

$\overline{XFER}$：DAC 锁存器选择信号。

$\overline{WR2}$：DAC 锁存器写控制信号。一旦数据进入 DAC 锁存器，D/A 转换即开始。

V_{REF}：基准参考电源输入。一般接 V_{CC}。

R_{FB}：电流/电压转换放大器反馈信号输入端。

I_{OUT1}：电流输出端 1，其值随 DAC 锁存器内容线性变化。

I_{OUT2}：电流输出端 2，$I_{OUT1}+I_{OUT2}=$常数。

V_{CC}：电源输入端。

AGND：模拟信号地。

DGND：数字信号地。

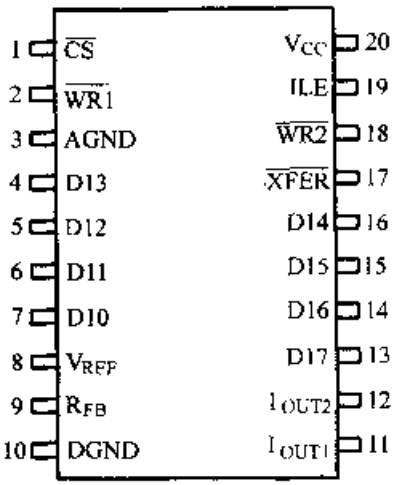


图 6-32　DAC0832 管脚图

5. D/A 转换的软件设计特点

D/A 转换器的使用比较简单，在硬件连接好之后，就确定了该器件的 I/O 地址。采用定时器设置输出间隔，在定时器中断服务程序中将要输出的数据写入 D/A 端口即可。

[课程实践]：采用 DAC0832D/A 转换芯片构成数模转换电路，输出采用四象限工作方式，输出电平为－5V～＋5V。画出电路线路图并编程使实现输出三角波和锯齿波信号。

6.3　人机通道配置与接口技术

单片机应用系统中，通常都要有人机对话功能。它包括操作者对应用系统状态的干预与数据输入，以及应用系统向操作者报告运行状态与运行结果。

6.3.1　单片机应用系统中的人机通道

1. 人机通道配置的类型

单片机应用系统中，人机对话配置水平、规模与应用系统的规模、特点有关。单片机应用系统的类型多种多样，如智能仪表、控制单元、数据采集系统、分布式监测系统等。对于各种类型的单片机应用系统，其人机通道配置的集合，如图 6-33 所示。

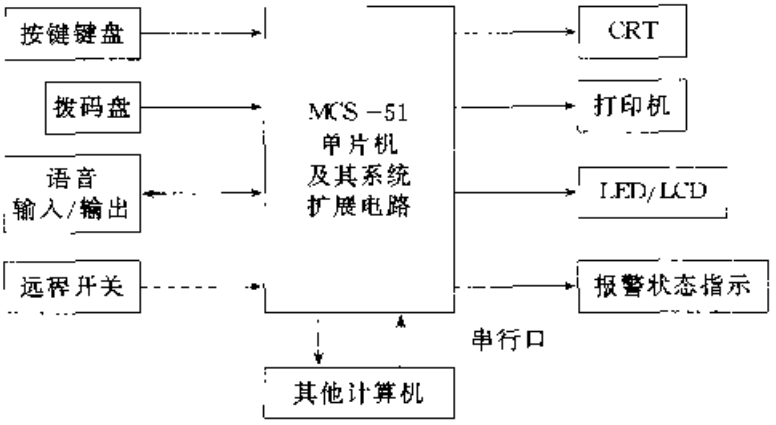


图 6-33　人机通道配置的集合

单片机应用系统中，人对系统状态的干预和数据输入的外围设备最常用的是键和键盘，有对系统状态实现干预的功能键和向系统输入数据的数字键等。拨码盘是对系统置入数据的一种较价廉、可靠的工具。这些都是目前单片机应用系统中最常用的办法。近年来，针对单片机应用系统的结构特点和应用环境，也开始发展非接触的人→机接口，如遥控键盘、远程开关以及语音输入接口等。

单片机应用系统中，系统向人报告运行状态及运行结果最常用的有各种报警指示灯、LED/LCD 显示器以及能永久保持结果数据、状态的打印机等。近年来，单片机应用系统根据需要也开始配置简易、廉价的 CRT 接口以及语音对话接口。

作为一个独立的单片机应用系统，根据功能需要选择图 6-31 中的一部分人机接口配置即可。但是有一些应用系统，如分布式监测系统，常常要配置较强水平的人机对话接口，而且常常采用集中控制，故通常与通用计算机系统连机。在这样的应用系统中，人机对话的配置主要依靠通用计算机系统实现。在控制总站及各控制子站中只配置一些简单的人机外设接口。

2. 人机对话接口的特点

单片机应用系统的人机对话是在应用系统与人之间的信息传递渠道。因此，其接口特点与单片机应用系统的特点以及用户的特点有关。

1）专用性：一般来说单片机应用系统都是专用计算机应用系统。人机通道外部设备配置水平完全根据系统功能要求而定，例如显示器显示位数、键盘数量、报警指示灯数量以及选用何种规模的微型打印机等。

2）小型价廉：单片机应用系统本身的特点是低成本，中小规模、环境适应性强、配置灵活。因此，相应的外围设备以配置小型、微型、廉价型为原则。

3）在 MCS-51 单片机应用系统中，外围设备与外部数据存储器统一编址，与外围设备的数据通信使用 MOVX 指令。

4）人机接口中一般都是数字逻辑控制电路，许多外围设备都有标准的接口与

通信要求。

6.3.2 显示及显示器接口

单片机应用系统中，使用的显示器主要有 LED（发光二极管显示器）和 LCD（液晶显示器）。这两种显示器成本低廉，配置灵活，与单片机接口方便。近年来也开始配置简易形式的 CRT 接口，可以较方便地进行图形显示。

1. LED 显示及显示器接口

(1) LED 显示器的结构与原理

LED 显示器是由发光二极管显示字段的显示器件。在单片机应用系统中通常使用的是七段 LED。这种显示器有共阴极与共阳极两种，如图 6-34 所示。共阴极 LED 显示器的发光二极管阴极共地，如图 a 所示，当某个发光二极管的阳极为高电平时，发光二极管点亮；共阳极 LED 显示器的发光二极管阳极并接，如图 b 所示。

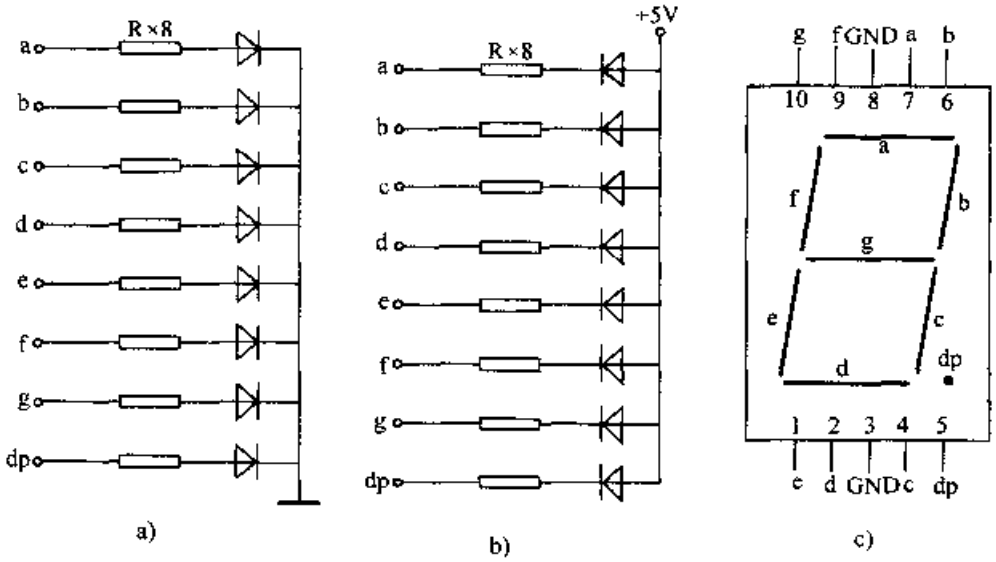


图 6-34 七段 LED 显示器

a) 共阴极 b) 共阳极 c) 管脚配置

通常的七段 LED 显示器中有八个发光二极管，故也有人叫做八段显示器。其中七个发光二极管构成七笔字形'8'，一个发光二极管构成小数点。

七段显示器与单片机接口非常容易。只要将一个 8 位并行输出口与显示器的发光二极管引脚相连即可。8 位并行输出口输出不同的字节数据即可获得不同的数字或字符，如下表所示。通常将控制发光二极管的 8 位字节数据称为段选码。共阳极与共阴极的段选码互为补数。

(2) LED 显示器与显示方式

在单片机应用系统中使用 LED 显示器构成 N 位 LED 显示。图 6-35 是 N 位 LED 显示器的构成原理图。

七段 LED 的段选码

显示字符	共阴段选码	共阳段选码	显示字符	共阴段选码	共阳段选码
0	3FH	C0H	C	39H	C6H
1	06H	F9H	D	5EH	A1H
2	5BH	A4H	E	79H	86H
3	4FH	B0H	F	71H	8EH
4	66H	99H	P	73H	8CH
5	6DH	92H	U	3EH	C1H
6	7DH	82H	Γ	31H	CEH
7	07H	F8H	Y	6EH	91H
8	7FH	80H	8.	FFH	00H
9	6FH	90H	"灭"	00H	FFH
A	77H	88H	…	…	…
B	7CH	83H			

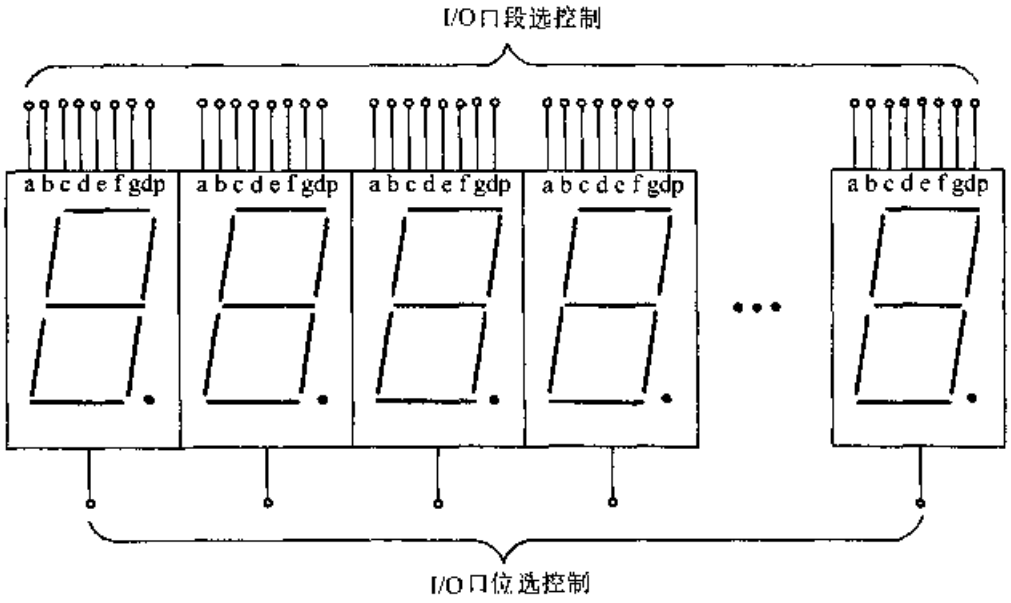


图 6-35　N 位 LED 显示器的构成原理

N 位 LED 显示器有 N 根位选线和 8×N 根段选线。根据显示方式不同，位选线与段选线的连接方法不同。段选线控制字符选择，位选线控制显示位的亮、暗。

LED 显示器有静态显示与动态显示两种方式。

1）LED 静态显示方式：LED 显示器工作在静态显示方式下，共阴极或共阳极

连接在一起接地或接+5V；每位的段选线（a—dp）与一个8位并行口相连。图6-36表示了一个4位静态LED显示器电路。该电路每一位可独立显示，只要在该位的段选线上保持段选码电平，该位就能保持相应的显示字符。由于每一位由一个8位输出口控制段选码，故在同一时间里每一位显示的字符可以各不相同。

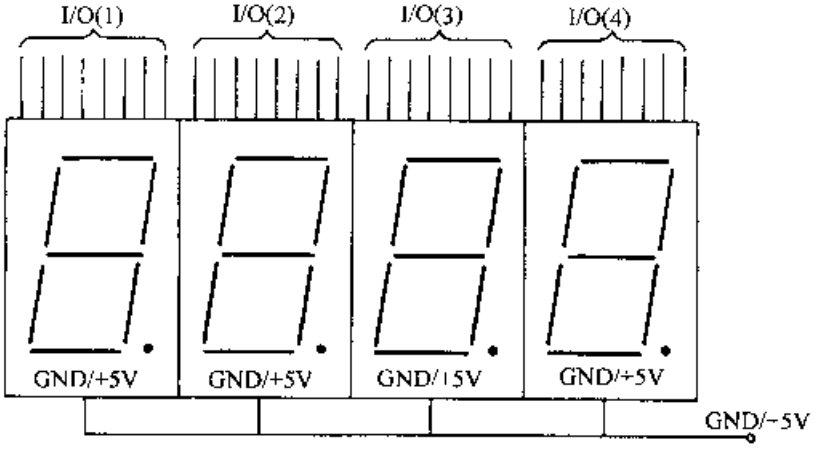


图6-36　4位静态LED显示器电路

N位静态显示器要求有N×8根I/O口线，占用I/O资源较多。故在位数较多时往往采用动态显示方式。

2）LED动态显示方式：在多位LED显示时，为了简化电路、降低成本，将所有位的段选线并联在一起，由一个8位I/O口控制，而共阴极点或共阳极点分别由相应的I/O口线控制。图6-37就是一个8位LED动态显示器电路。

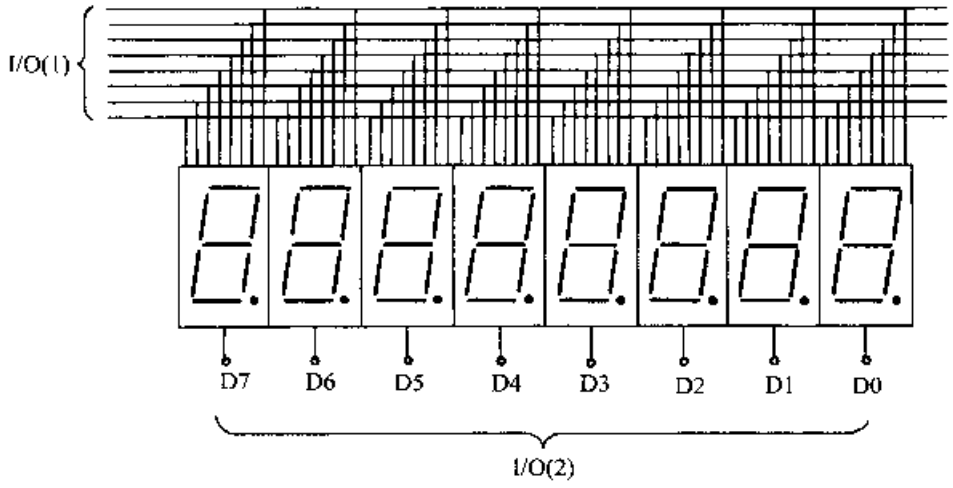


图6-37　8位LED动态显示器电路

8位LED动态显示电路只需要两个8位I/O口。其中一个控制段选码，另一个控制位选。由于所有位的段选码皆由一个I/O控制，因此，在每个瞬间，8位LED只可能显示相同的字符。要想每位显示不同的字符，必须采用动态扫描显示方式。即在每一瞬间只使某一位显示相应字符。在此瞬间，位选控制I/O口在该显示位

送入选通电平（共阴极送低电平、共阳极送高电平）以保证该位显示相应字符，段选控制 I/O 口输出相应字符段选码。如此轮流，使每位显示该位应显示的字符，并保持延时一段时间，以造成视觉暂留效果。

不断循环送出相应的段选码、位选码，就可以获得视觉稳定的显示状态。由人眼的视觉特性，每一位 LED 在一秒钟内点亮不少于 30 次，其效果和一直点亮相差不多。

2. LED 显示器接口实例

从 LED 显示器的显示原理可知，为了显示字母数字，必须最终转换成相应段选码。这种转换可以通过硬件译码器或软件进行译码。

下面介绍使用译码器或软件译码的一些接口电路。

（1）硬件译码显示器接口——BCD-七段十六进制译码驱动显示接口

单片机应用系统中，通常要求 LED 显示器能显示十六进制及十进制带小数点的数。因此，在选择译码器时，要能够完成 BCD 码至十六进制的锁存、译码，并具有驱动功能，否则就不如采用软件译码接口。

图 6-38 是 MOTOROLA 公司生产的 CMOS BCD-七段十六进制锁存、译码驱动芯片 MC14495 的逻辑图和引脚及字形显示图。该电路的特点是可用字母 A、B、C、D、E、F 来显示二进制数 10、11、12、13、14、15，同时还有译码器输入大于等于 10 时的指示端（h）。当输入数据≥10 时，h 端输出 1 电平。另外还有输入数据为 15 时，电路输出端 VCR 为 0 电平（其他输入状态时为高阻）的功能。电

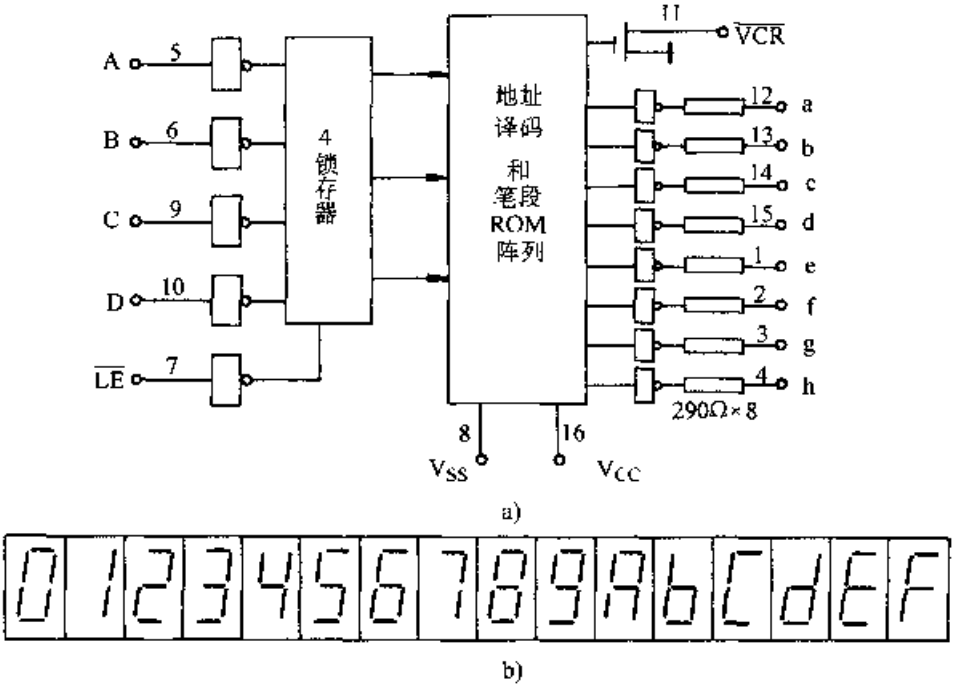


图 6-38　MC14495 BCD-七段十六进制锁存译码驱动器

a）逻辑图和引脚　b）字形显示图

路内部还有一个 290Ω 的限流电阻。LE 为选通端，电路中的锁存器在 LE＝0 时输入数据，在 LE＝1 时锁存数据。

图 6-39 是使用 MC14495 的多位静态 LED 显示接口电路，该电路中可直接显示多位十六进制数。若要显示带小数点的十进制数，则只要在 LED 的 dp 端另加驱动控制即可。LED 显示块采用共阴极。因 MC14499 有输出限流电阻，故 LED 不需外加限流电阻。

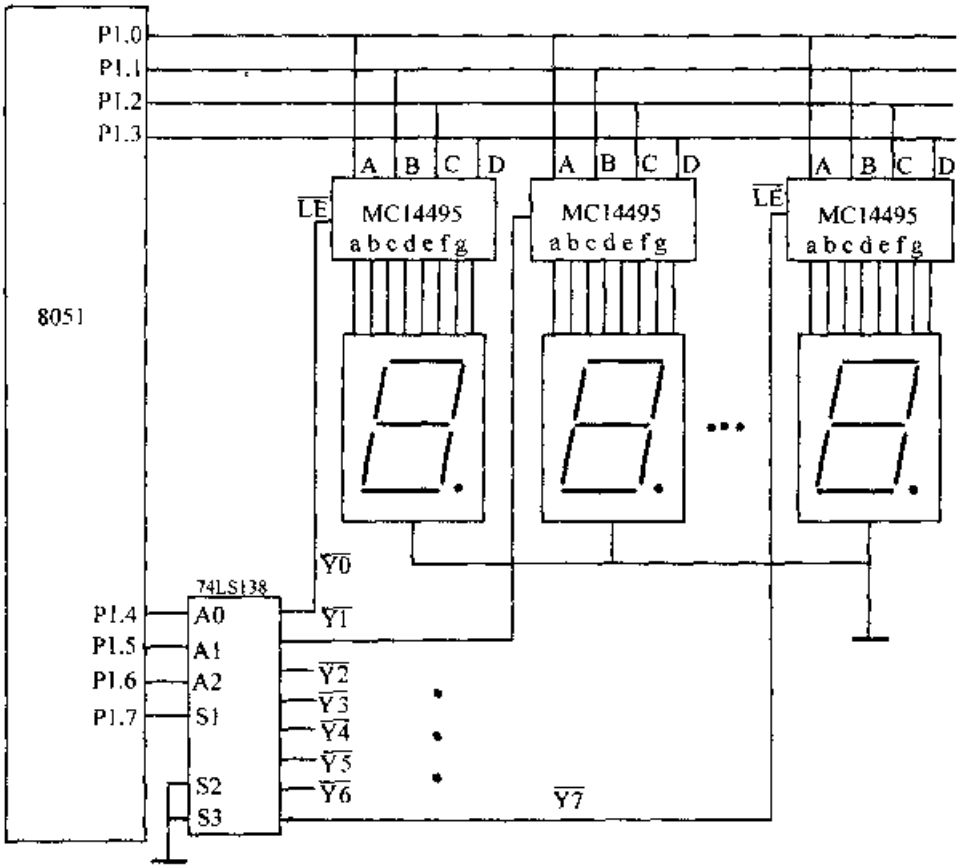


图 6-39　使用 MC14495 的多位静态 LED 显示接口

该接口软件十分简单。当给 P1.7 高电平时，开始显示，由 P1.4、P1.5、P1.6 控制 LE 依次选中一位 LED 然后由 P1.0～P1.3 送入 BCD 码，在使 LE 转高电平时锁存该位数据并译码、驱动显示。

(2) 软件译码的显示器接口

应该指出，在单片机应用系统中，由于单片机本身有较强的逻辑控制能力，采用软件译码并不复杂。而且软件译码其译码逻辑可随意编程设定，不受硬件译码逻辑限制。采用软件译码还能简化硬件电路结构。因此，在单片机应用系统中，使用得最广的还是软件译码的显示器接口。

1) 软件译码的静态显示接口：图 6-40 是 8031 通过 8255 扩展 I/O 口控制的三位静态 LED 显示接口。图中 LED 为共阴极。若为共阳极时，公共极接＋5V。如

果显示位数较多时，可再增加 8255 或其他并行输出口。

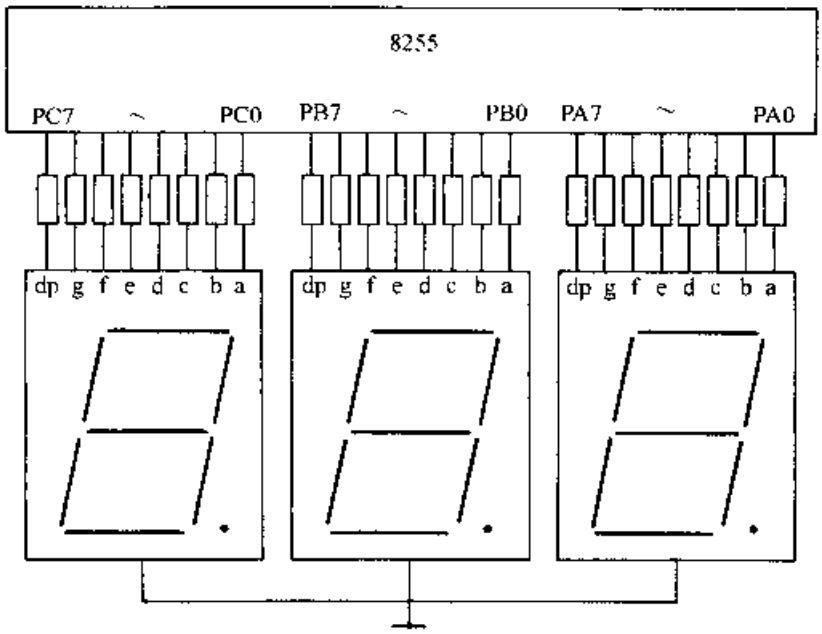


图 6-40　用 8255 构成 3 位静态 LED 显示接口

2）软件译码的动态显示接口：图 6-41 是 8031 通过 8155 扩展 I/O 控制的 8 位 LED 动态显示接口。只需要 8155 提供 2 个 8 位输出口即可。图中 PB 口输出段选码，PA 口输出位选码，位选码占用输出口线数取决于显示器位数。74S437 为 8 位集成驱动芯片。

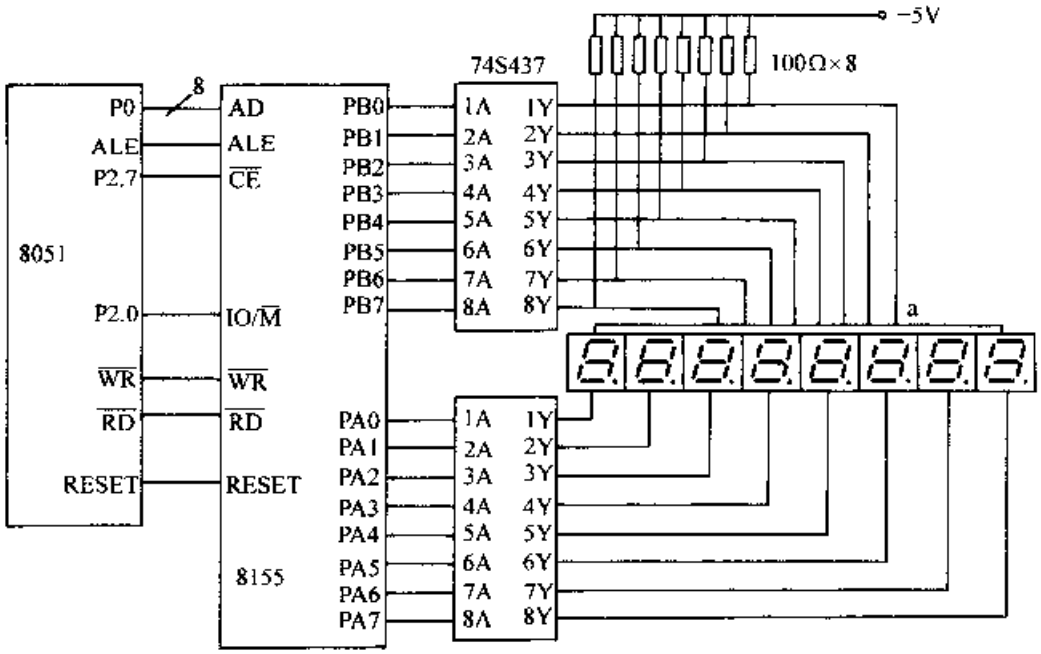


图 6-41　用 8155 构成的 8 位 LED 动态显示接口

3）动态显示程序设计：对于图 6-41 的动态显示接口，其动态显示子程序框图如图 6-42 所示。

设 fosc=6MHz。定时器 T0 工作于方式 1。

程序清单

```
#include<reg51.h>
#include<intrins.h>
#define COM8155 XBYTE [0x7FF8]
#define PA XBYTE [0x7FF9]
#define PB XBYTE [0x7FFA]
char code dispdata [] = {0x3F, 0x06, 0x5B,
0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F,
0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};
char dis dat [8];
void delay () /*延时 1ms*/
{
    TL0=--500%256;
    TH0=--500/256;
    TR0=1;
    while (! TF0);
    TF0=0;
    TR0=0;
}
void disp (char ch1)
{
    static char ch=0xFE;
    PA=ch;
    PB=dis dat [ch1];
    ch=~ch;
    ch=ch<<1;
    if (ch==0) ch=1;
    ch=~ch;
}
main ()
{
    char ch1, ch2;
    TMOD=0x01;
    COM8155=3;
```

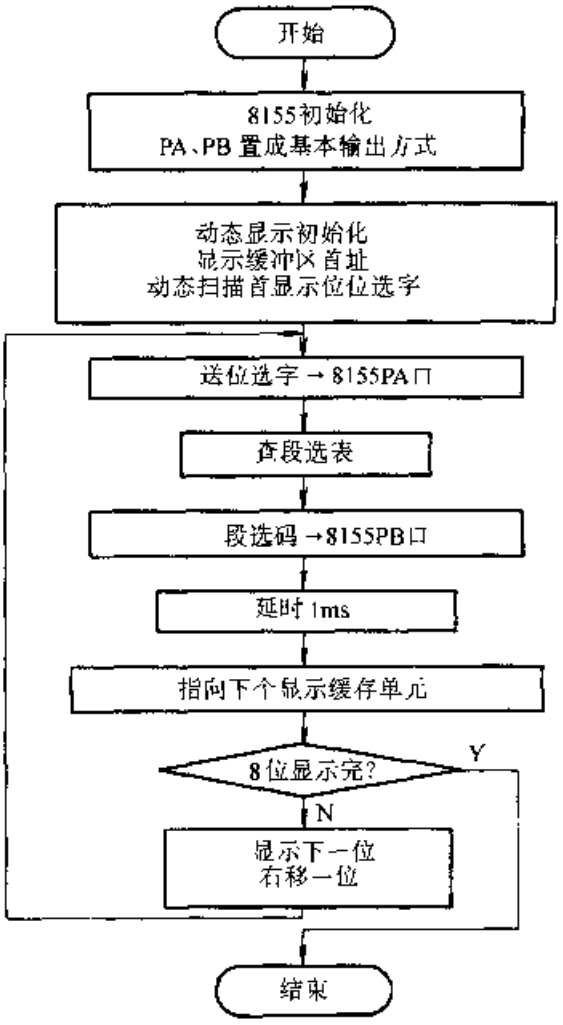


图 6-42　动态显示子程序框图

```
    While (1)
     {
        for (ch1=0; ch1<8; ch1++)
         {
            disp (ch1);
            delay ();
        }
        delay ();
        delay ();
    }
}
```

6.3.3 按键、键盘及其接口

在单片机应用系统中，为了控制系统的工作状态以及向系统中输入数据，应用系统应设有按键或键盘。例如复位用复位键，功能转换用的功能键以及数据输入用的数字键盘等。

1. 单片机应用系统中的键输入

单片机应用系统中除了复位按键有专门的复位电路，以及专一的复位功能外，其他按键或键盘都是以开关状态来设置控制功能或输入数据的。因此，这些开关不只是简单的用于电平输入。

(1) 键输入过程与软件结构

当所设置的功能键或数字键按下时，计算机应用系统应完成该按键所设定的功能。因此，键信息输入是与软件结构密切相应的过程。对某些应用系统，例如智能仪表来说，键输入程序是整个应用程序的核心部分。

图 6-43 是 MCS—51 单片机应用系统的键输入软件框图。

对一组键，或一个键盘，总有一个接口电路与 CPU 相连。通过软件了解键输入信息，CPU 可以采用中断方式或查询方式了解有无键输入并检查是哪一个键按下，当有键按下时，执行该键的功能程序。

(2) 键输入接口与软件应完成的任务

键输入接口与软件应可靠而快速地实现键信息输入与键功能任务。为此，应解决下列问题。

1) 键开关状态的可靠输入：目前，无论是按键或键盘都是利用机械触点的合、断作用。一个电压信号通过机械触点的闭合、断开过程，其波形如图 6-44 所示。由于机械触点的弹性作用，在闭合及断开瞬间均有抖动过程，会出现一系列负脉冲。抖动时间长短与开关的机械特性有关，一般为 5～10ms。

按键的稳定闭合期，由操作人员的按键动作所确定，一般为十分之几秒至几

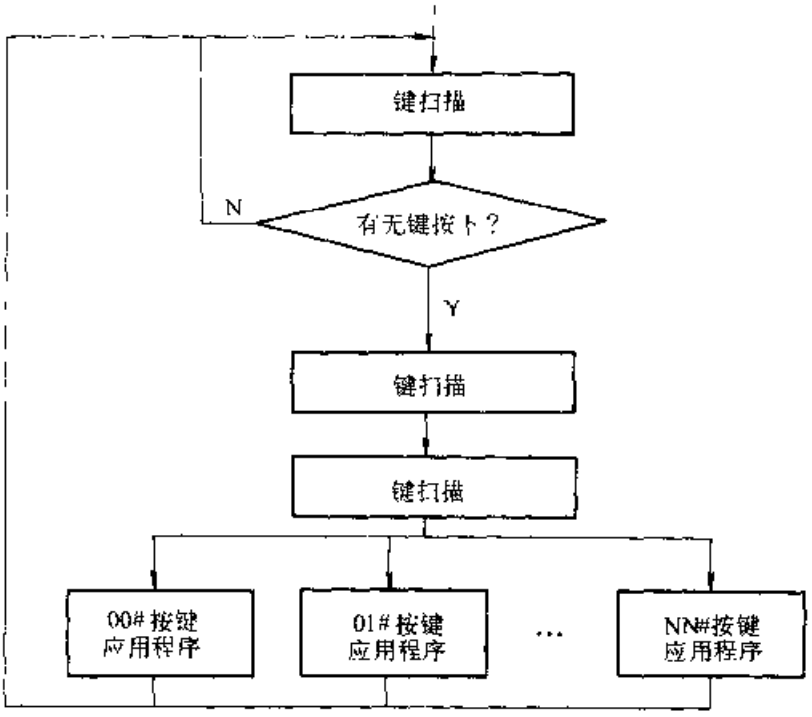


图 6 43　键输入过程框图

秒时间。为了保证CPU对键的一次闭合，仅作一次键输入处理，必须去除抖动影响。

通常去抖动影响的措施有硬、软件两种。图 6-45 是用 R-S 触发器或单稳态电路构成的硬件去抖动电路。采用软件去除抖动影响的办法是在检测到有键按下时，执行一个 10ms 的延时程序后再确认该键电平是否仍保持闭合状态电平，如保持闭合状态电平则确认为真正键按下状态，从而消除了抖动影响。

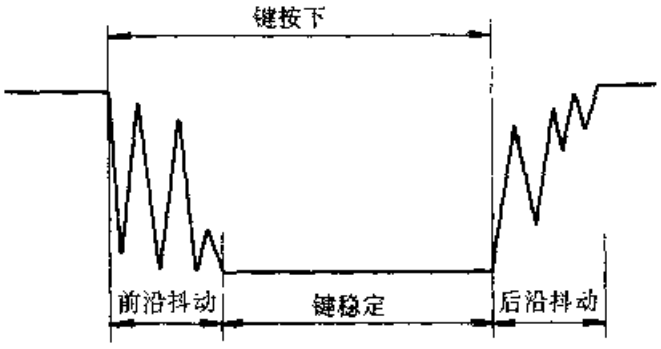


图 6-44　键闭合及断开时的电压抖动

2）选择键盘监测方法：对于计算机应用系统，键盘扫描只是CPU工作的一部分，键盘处理只是在有键按下时才有意义。对是否有键按下的信息输入方式有中断方式与查询方式两种。

3）编制好键盘程序：一个完善的键盘控制程序应具备下列功能：

① 监测有无按键按下。

② 有键按下后，在无硬件去抖动电路时，应有软件延时方法去除抖动影响。

③ 有可靠的逻辑处理办法。如 n 键锁定，即只处理一个键，其间任何按下

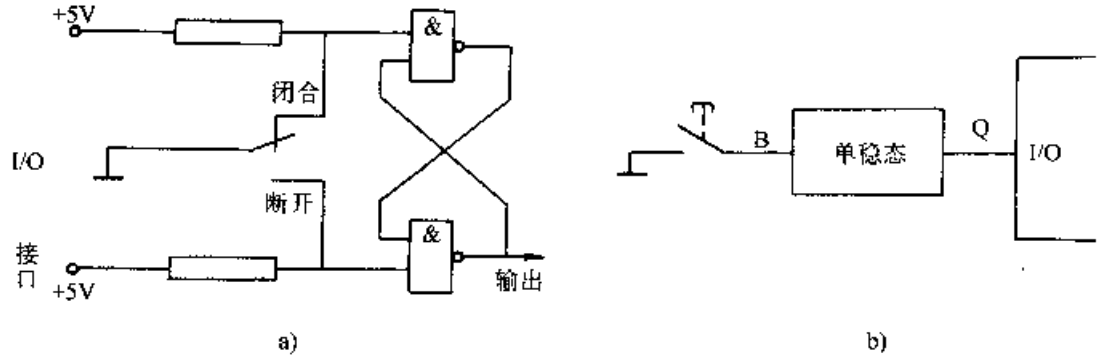


图 6-45　去抖动电路

a）R-S 触发器　b）单稳态电路

又松开的键不产生影响；不管一次按键持续有多长时间，仅执行一次按键功能程序等。

2. 独立式按键

（1）独立式按键的结构

独立式按键是指直接用 I/O 口线构成的单个按键电路。每个独立式按键单独占有一根 I/O 口线，每根 I/O 口线上的按键工作状态不会影响其他 I/O 口线的工作状态。独立式按键电路如图 6-46 所示。

独立式按键电路配置灵活，软件结构简单，但每个按键必须占用一根 I/O 口线，在按键数量较多时，I/O 口线浪费较大。故在按键数量不多时，常采用这种按键电路。

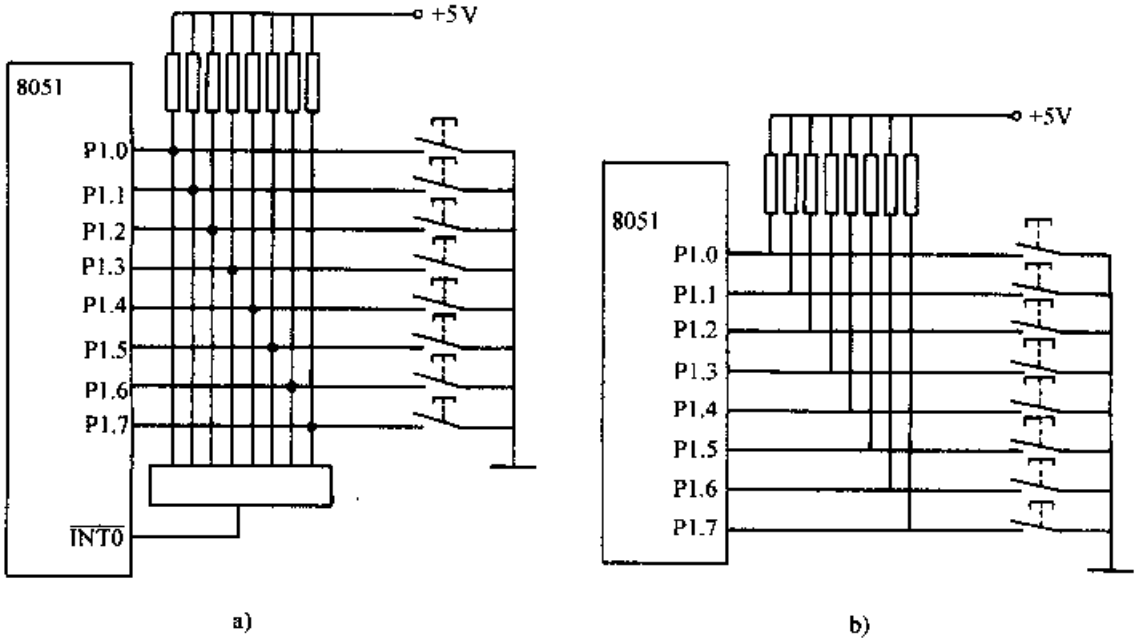


图 6-46　独立式按键电路

a）中断方式　b）查询方式

图 6-46a 为中断方式的独立式按键电路，图 b 为查询方式电路。通常按键输入都采用低电平有效。上拉电阻保证了按键断开时，I/O 口线有确定的高电平。当 I/O 口内部有上拉电阻时，外电路可以不配置上拉电阻。

(2) 独立式按键的软件结构

下面是查询方式的键盘程序。程序中没有软件防抖动措施。它只包括键查询、键功能程序转移。

程序清单（设 I/O 为 P1 口）：

```
void keyproc ()
{
    char x;
    x=P1;
    switch (x)
     {
        case 0xFE: prom0 ();          /*处理 0 号键*/
            break;
        case 0xFD: prom1 ();          /*处理 1 号键*/
            break;
        case 0xFB: prom2 ();      break;
        case 0xF7: prom3 ();      break;
        case 0xEF: prom4 ();      break;
        case 0xDF: prom5 ();      break;
        case 0xBF: prom6 ();      break;
        case 0x7F: prom7 ();
    }
}
```

其中，prom0 () ～prom7 () 分别是 8 个按键所对应的功能函数。

由此程序可以看出，按键由软件设置了优先级，优先级的顺序依次为 0～7。

3. 行列式键盘

行列式键盘又叫矩阵式键盘。用 I/O 口线组成行、列结构，按键设置在行、列的交点上。例如用 2×2 的行、列结构可构成 4 个键的键盘，4×4 的行列结构可构成 16 个键的键盘。因此，在按键数量较多时，可以节省 I/O 口线。

(1) 键盘工作原理

行列式键盘电路原理如图 6-47 所示。

按键设置在行、列线交点行，行、列线分别连接到按键开关的两端。当行线通过上拉电阻接＋5V 时，被钳位在高电平状态。

键盘中有无按键按下是由列线送入全扫描字、行线读入行线状态来判断的。其方法是：给列线的所有 I/O 口线均置成低电平，然后将行线电平状态读入。如果有键按下，总会有一根行线电平被拉至低电平，从而使行输入不全为'1'。

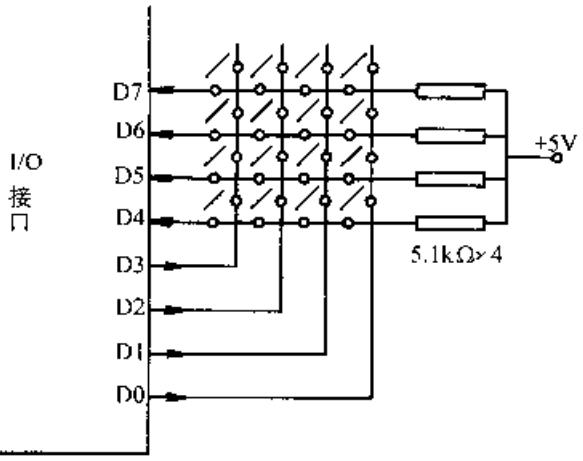


图 6 47　行列式键盘原理电路

键盘中哪一个键按下是由列线逐列置低电平后，检查行输入状态而定的。其方法是：依次给列线送低电平，然后查所有行线状态，如果全为'1'，则所按下之键不在此列。如果不全为'1'，则所按下的键必在此列。而且是在与'0'电平行线相交的交点上的那个键。

（2）键盘工作方式

单片机应用系统中，键盘扫描只是CPU 工作的一个内容之一。CPU 在忙于各项工作任务时，如何兼顾键盘扫描，即既保证不失时机地响应键操作，又不过多占用 CPU 时间。因此，要根据应用系统中 CPU 的忙、闲情况，选择好键盘的工作方式。键盘的工作方式有编程扫描方式、定时扫描方式和中断扫描方式三种。

1）编程扫描工作方式：编程扫描工作方式是利用 CPU 在完成其他工作的空余，调用键盘扫描子程序，来响应键输入要求。在执行键功能程序时，CPU 不再响应键输入要求。下面以图 6-48 的 8155 扩展 I/O 口组成的行列式键盘为例，介绍编程扫描工作方式的工作过程与键盘扫描子程序。在该键盘中，共 32 个键，由 1

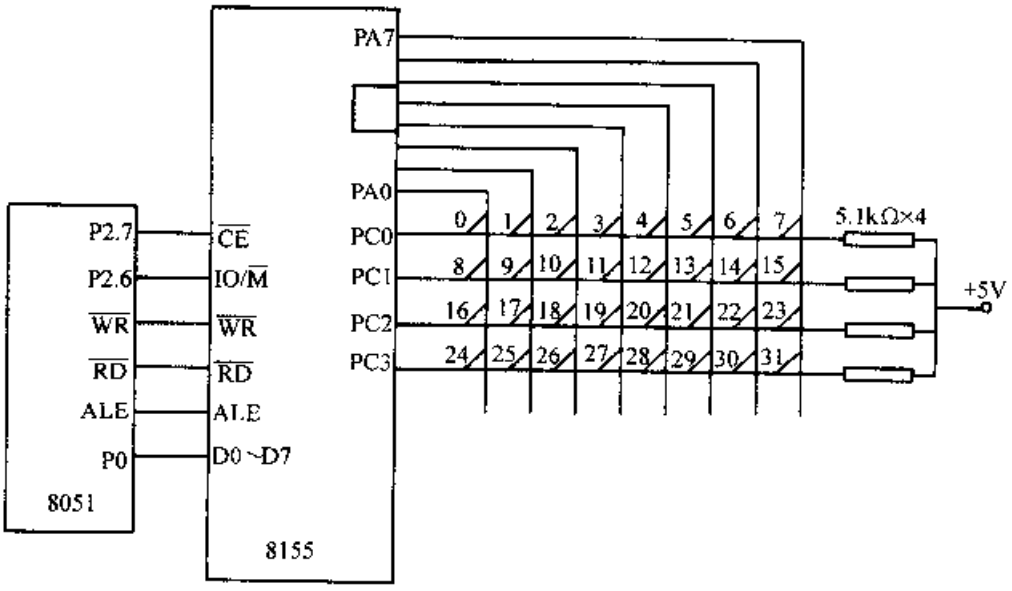


图 6-48　8155 扩展 I/O 口组成的行列式键盘

个 8 位口和 1 个 4 位口组成 4×8 的行列式键盘。

在键盘扫描子程序中完成下述几个功能：

① 判断键盘上有无键按下。其方法为，PA 口输出全扫描字 00H，读 PC 口状态，若 PC0～3 为全'1'则键盘无键按下，若不全为'1'则有键按下。

② 去键的机械抖动影响。其方法为，在判断有键按下后，软件延时一段时间再判断键盘状态，如果仍为有键按下状态，则认为有一个确定的键按下，否则按键抖动处理。

③ 求按下键的键号。按照行列式键盘工作原理，图中 32 个键的键值应对应作如下分布（按 PA、PC 口二进制码，X 为任意值）：

FEXE	FDXE	FBXE	F7XE	EFXE	DFXE	BFXE	7FXB
FEXD	FDXD	FBXD	F7XD	EFXD	DFXD	BFXD	7FXD
FEXB	FDXB	FBXB	F7XB	EFXB	DFXB	BFXB	7FXB
FEX7	FDX7	FBX7	F7X7	EFX7	DFX7	BFX7	7FX7

④ 键闭合一次仅进行一次键功能操作。其方法为，等待键释放以后再将键号返回。

图 6-49 为键扫描子程序框图。

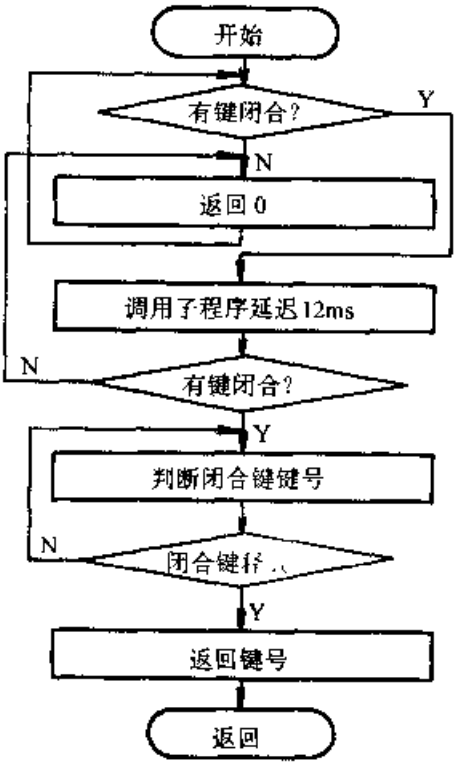


图 6-49 键扫描子程序框图

键号扫描函数清单（8155 的初始化，置 PA 口为基本输出口、PC 为基本输入口，放在主程序中）：

```
void keyproc ()
{
   PA=0;
   if (PC & 0x0F! =0x0F)
   {
      delay (); /* 12ms 延时 */
      if (PC & 0x0F==0x0F)
         return;
      PA=0xFE;
      switch (PC & 0x0F) {
         case 0x0E: proc0 (); break;
         case 0x0D: proc8 (); break;
         case 0x0B: proc16 (); break;
         case 0x07: proc24 ();
      }
      PA=0xFD;
```

```
        switch (PC & 0x0F) {
            case 0x0E: proc1 (); break;
            case 0x0D: proc9 (); break;
            case 0x0B: proc17 (); break;
            case 0x07: proc25 ();
        }
        ...
    }
    PA=0xFF;
}
```

编程扫描工作方式只有在CPU空闲时才调用键盘扫描子程序。因此，在应用系统软件方案设计时，应考虑这种键盘扫描子程序的编程调用能满足键盘响应要求。

2）定时扫描工作方式：定时扫描工作方式是利用单片机内部定时器产生定时中断（例如10ms，CPU响应中断后对键盘进行扫描，并在有键按下时转入键功能处理程序。定时扫描工作方式的键盘硬件电路与编程扫描工作方式相同。其软件框图如图6-50所示。

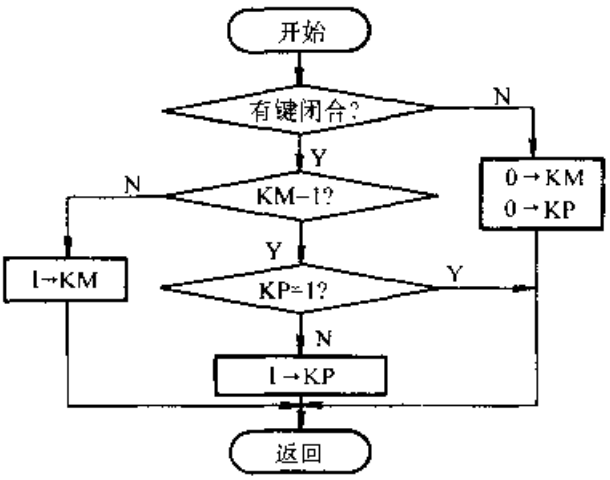


图 6-50　定时扫描方式程序框图

定时扫描工作方式在本质上是中断方式，因此，图6-50是一个中断服务程序框图。按照程序要求，在单片机的片内RAM位寻址区设置去抖动标志KM和处理标志KP两个标志位。

当键盘中无键按下，KM、KP置零，返回。由于定时开始后一般不会立即有键按下，故相当于KM、KP初始化置零。

当键盘中有键按下时，先检查KM标志，KM＝0时，表示尚未作去抖动影响处理，此时中断返回同时KM置'1'。因为中断返回后要经10ms才可能再次中断，相当于实现了10ms延时效果，因而程序中不需要延时。当再次定时中断后检查KP标志，由于开始时KP＝0，程序进入查找键号，并使KP置'1'，执行键功能程序，然后返回。在KM、KP均为'1'时，表示键处理完毕，再次定时中断时，将KM、KP清0。

3）中断工作方式：计算机应用系统工作时，并不经常需要键输入，因此，无论是编程工作或定时工作，CPU经常处于空扫描状态。为了进一步提高CPU效

率，可以采用中断扫描工作方式。即可在键盘上有键按下时，才执行键盘扫描，执行该键功能程序。中断扫描工作方式的键盘接口如图 6-51 所示。

该键盘直接由 8031 的 P1 口的高、低字节构成 4×4 行列式键盘。键盘的列线与 P1 口的低 4 位相连，键盘的行线通过二极管接到 P1 口的高 4 位。因此，P1.4～P1.7 作键输出线，P1.0～P1.3 作扫描输入线。初始化时，使 P1.0～P1.3 置零。当有键按下时，$\overline{INT0}$/$\overline{INT1}$端为低电平，向 CPU 发出中断申请，若 CPU 开放外部中断，则响应中断请求，进入中断服务程序。在中断服务程序中除完成键识别、键功能处理外，还须有消除键抖动影响、防止多次重复执行键功能操作等措施。

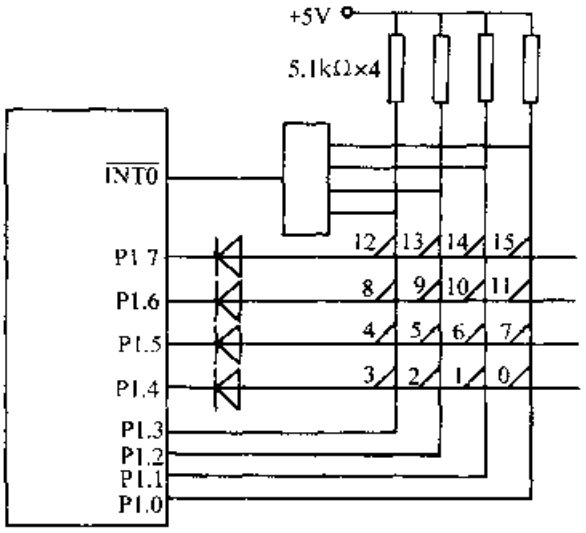


图 6-51　中断方式键盘接口

（3）键盘扫描方式

当键盘有键按下时，要逐行或逐列扫描，以判定是哪一个按键按下。通常扫描方式有两种，即扫描法和反转法。

1）扫描法：扫描法是在判定有键按下后逐列（或行）置低电平，同时读入行（或列）状态，如果行（或列）状态出现非全'1'状态，这时'0'状态的行、列交点的键就是所按下的键。

扫描法的特点就是逐行（或列）扫描查询。这时，相应的行（或列）应有上拉电阻接高电平。

图 6-47 的行列式键盘扫描程序就是采用扫描法来确定哪个键按下的，图中行线上拉接＋5V，列线逐列扫描。

2）反转法：扫描法要逐行（列）扫描查询，当所按下的键在最后行（列）时，则要经过多次扫描才能获得键值/键号。而采用反转法时，只要经过两个步骤即可获得键值。反转法原理如图 6-52 所示。

图中硬件采用中断工作方式。用一个 8 位 I/O 口构成 4×4 键盘。假定图中所画的键为所按下的键。下面介绍反转法的两个步骤：

第一步：将 D3～D0 编程为列输入线，D7～D4 编程为行输出线，并使 I/O 输出数据为 0FH（即保证行输出信号 D7～D4 为 0000）。若有键按下，与门的输出端变为低电平，向 CPU 申请中断，表示键盘中有键按下。与此同时，D3～D0 的数据用一个变量 x 保存。其中 0 位对应的是被按下键的列位置，然后转入第二步。

第二步：将第一步中的传送方向反转过来，即将 D7～D4 编程为输入线，D3

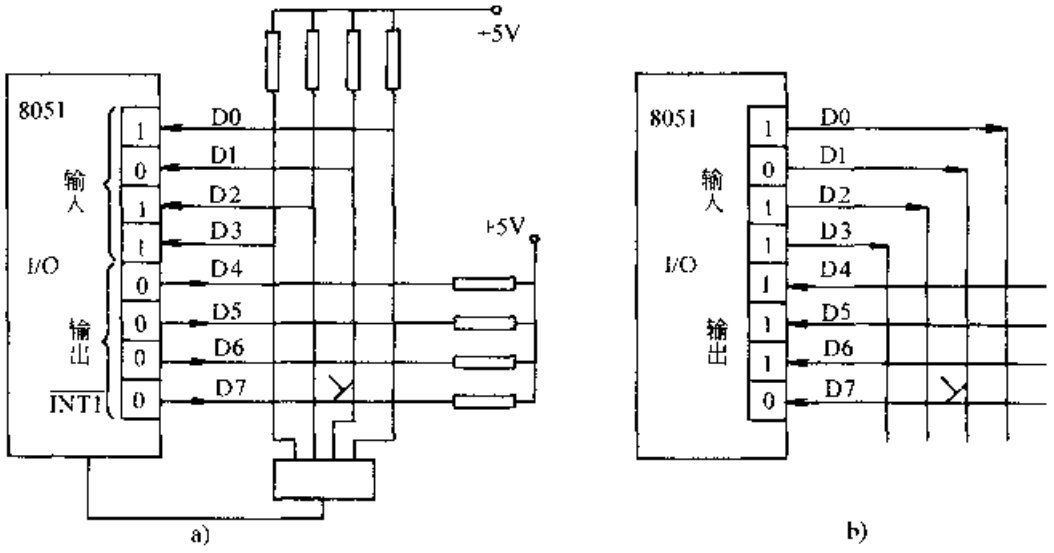


图 6-52　线反转法原理

a）线反转法第一步　b）线反转法第二步

～D0 编程为输出线。使 I/O 口输出数据为 x 的值（即 D3～D0 为按下键的列位置），然后读入 I/O 数据，存入变量 y，该数据的 D7～D4 位中 0 电平对应的位是按下键的行位置。

最后，将 x 的 D3～D0 与 y 的 D7～D4 拼接起来就是按下键的键值。如图 6-50 中按下键的键值为 01111101＝0x7D。

（4）行列式键盘接口

单片机应用系统中，任何 I/O 口或扩展 I/O 口均可构成行列式键盘。MCS-51 单片机用于系统扩展时，可提供用户直接使用的 I/O 口线很少。故在 MCS-51 单片机应用系统中，大多采用扩展 I/O 口来构成行列式键盘。典型的键盘接口有通用 I/O 扩展口、串行 I/O 扩展口、专用键盘芯片构成的行列式键盘。由于带有行列式键盘的应用系统中通常都有显示器，为节省 I/O 口线，往往把显示器电路与行列式键盘做在一个接口电路中。

1）通用并行扩展 I/O 口键盘接口：这种键盘一般通过通用 I/O 扩展芯片的 I/O 口线构成行列式结构。如 8155、8255 等。图 6-48 是由 8155 扩展 I/O 口构成的 4×8 的行列式键盘接口。如果改成 8255 扩展口，则其键盘电路接口以及键盘扫描子程序结构完全相似。

硬件部分换成 8255，同样可使用 PA 口、PC 口或其他口线，并在主程序中把列线设置成输出，行线设置成输入。与 8051 的接口部分遵循 8255 扩展要求。软件部分可以完全套用图 6-49 的 8155 行列式键盘的扫描子程序，只要把程序中 8155 的 A 口、C 口地址改成 8255 的相应口线地址即可。

2）8051 串行口 I/O 口扩展的键盘接口：由于 8031 的串行口在方式 0 工作状态下，可以方便地通过移位寄存器扩展并行输出口。因此，可以将这些并行口线

作为列线可与 P3 口的行线构成行列式键盘，如图 6-53 所示。每占用一根 P3 口线可增加八个按键，图中为 2×8 键盘。用户可根据需要增减。

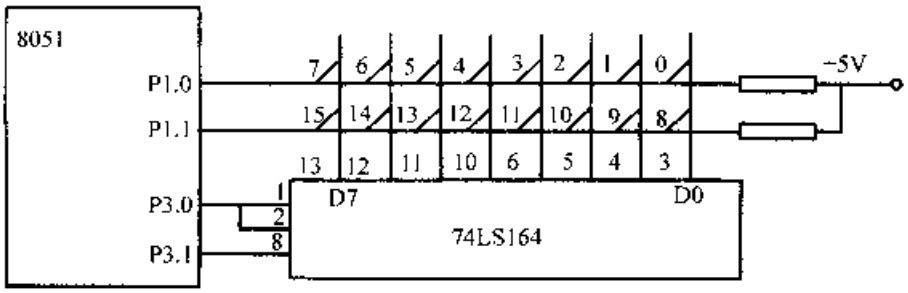


图 6-53　串行口 I/O 扩展的行列式键盘接口

4. 8279 键盘、显示接口芯片

Intel8279 芯片是一种通用的可编程序的键盘、显示接口器件，单个芯片就能完成键盘输入和 LED 显示控制两种功能。其内部结构如图 6-54 所示。

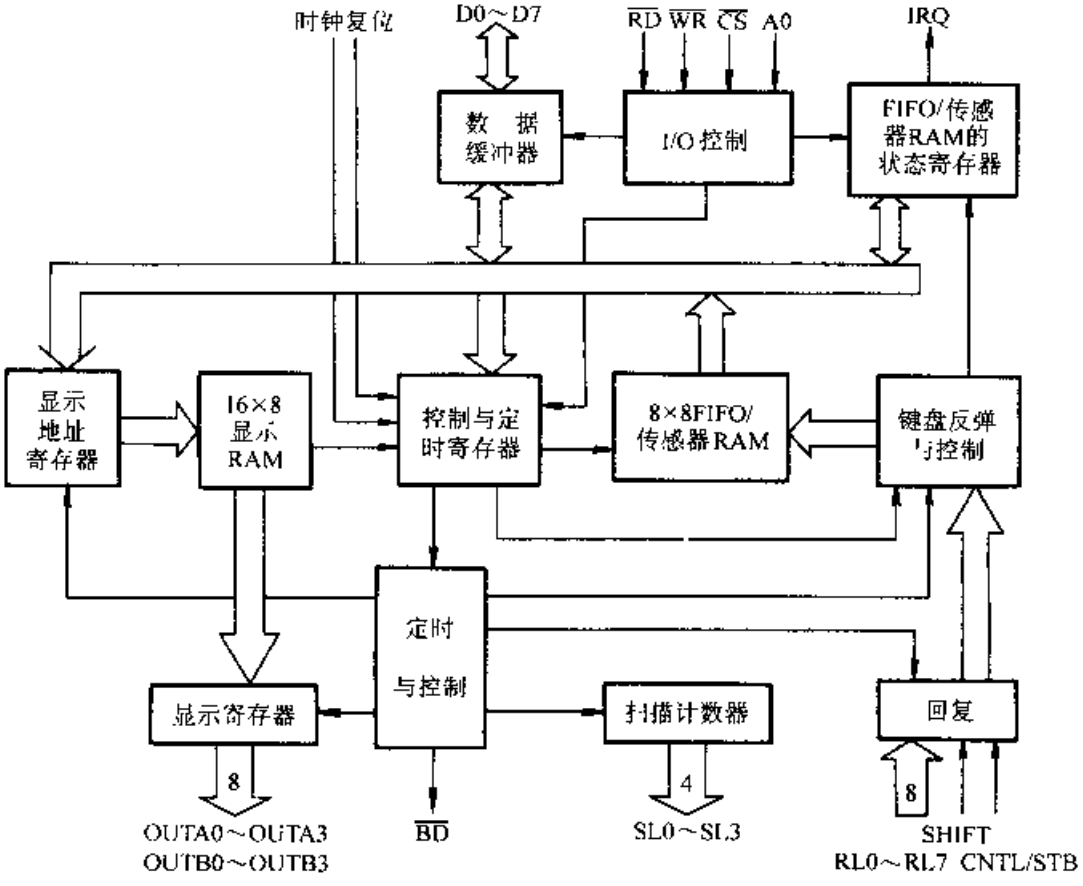


图 6-54　8279 结构框图

8279 包括键盘输入和显示输出两个部分。

键盘部分提供的扫描方式，可以和具有 64 个按键或传感器的阵列相连。能自动消除开关抖动并实现 N 键同时按下的保护。

显示部分按扫描方式工作。可以显示 8 或 16 位 LED 显示块。

(1) 8279 电路工作原理

根据结构框图，分别介绍各部分电路的作用。

1) I/O 控制及数据缓冲器：数据缓冲器是双向缓冲器，连接内、外部总线，用于传送 CPU 和 8279 之间的命令或数据。I/O 控制线是 CPU 对 8279 进行控制的引线。$\overline{CS}$是 8279 的片选信号，当$\overline{CS}=0$ 时，8279 才被允许读出或写入信息。$\overline{WR}$、$\overline{RD}$为来自 CPU 的读、写控制信号。

A0 用于区别信息特性：A0=1 时，表示数据缓冲器输入为指令、输出为状态字；A0=0 时，输入、输出皆为数据。

2) 控制与定时寄存器及定时控制：控制与定时寄存器用来寄存键盘及显示的工作方式，以及由 CPU 编程的其他操作方式。这些寄存器一旦接收并锁存送来的命令，就通过译码产生相应的信号，从而完成相应的控制功能。

定时控制包含基本计数器。首级计数器是一个可编程的 N 级计数器。N 在 2～31 之间由软件编程，以便从外界时钟 CLK 分频得到内部所需要的 100kHz 时钟。然后再经过分频，为键盘扫描提供适当的逐行扫描频率和显示扫描时间。

3) 扫描计数器：扫描计数器有两种工作方式。按编码方式工作时，计数器作二进制计数。4 位计数状态从扫描线 SL0～SL3 输出，经外部译码器译码后，为键盘和显示器提供扫描线；按译码方式工作时，扫描计数器的最低二位被译码后，从 SL0～SL3 输出。因此，SL0～SL3 提供了 4 中取 1 的扫描译码。

4) 回复缓冲器、键盘去抖及控制：来自 RL0～RL7 的 8 根回复线的回复信号，由回复缓冲器缓冲并锁存。

在键盘工作方式中，回复线作为行列式键盘的行列输入线。在逐行列扫描时，回复线用来搜寻每一行列中闭合的键。当某一键闭合时，去抖电路被置位，延时等待 10ms 后，再检验该键是否继续闭合，并将该键的地址和附加的移位、控制状态一起形成键盘数据被送入 8279 内部 FIFO（先进先出）存储器。键盘数据格式如下：

D7	D6	D5 D4 D3	D2 D1 D0
控制	移位	扫描	回复

控制和移位（D7、D6）的状态由两个独立的附加开关决定，而扫描（D5、D4、D3）和回复（D2、D1、D0）则是被按键置位的数据。D5、D4、D3 来自扫描计数器，是按下键的行列编码，而 D2、D1、D0 则来自行/列计数器，它们是根据回复信息而确定的行/列编码。

在传感器开关状态矩阵方式中，回复线的内容直接被送往相应的传感器 RAM（即 FIFO 存储器）。

在选通输入方式工作时，回复线的内容在 CNTL/STB 线的脉冲上升沿被送

入 FIFO 存储器。

5）FIFO/传感器 RAM 及其状态寄存器：FIFO/传感器 RAM 是一个双重功能的 8×8RAM。

在键盘或选通方式工作时，它是 FIFO 存储器，其输入或读出遵循先入先出的原则。FIFO 状态寄存器用来存放 FIFO 的工作状态。例如，FIFO 存储器是满还是空；其中存有多少数据；是否操作出错等。当 FIFO 存储器不空，状态逻辑将产生 IRQ=1 信号向 CPU 申请中断。

在传感器矩阵方式工作时，这个存储器又是传感器存储器。它存放着传感器矩阵中的每一个传感器状态。在此方式中，若检索出传感器的变化，IRQ 信号变为高电平，向 CPU 申请中断。

6）显示 RAM 和显示地址寄存器：显示 RAM 用来存储显示数据，容量为 16×8 位。在显示过程中，存储的显示数据轮流从显示寄存器输出。显示寄存器分为 A、B 两组，OUTA0～A3 和 OUTB0～B3 可以单独送数，也可以组成一个 8 位的字。显示寄存器的输出与显示扫描配合，不断从显示 RAM 中读出显示数据，同时轮流驱动被选中的显示器件，以达到多路复用的目的，使显示器件呈现稳定的显示状态。

显示地址寄存器用来寄存由 CPU 进行读/写显示 RAM 的地址，它可以由命令设定，也可以设置成每次读出或写入之后自动递增。

（2）管脚、引线与功能

8279 采用 DIP40 引脚封装，其管脚、引线功能如图 6-55 所示。

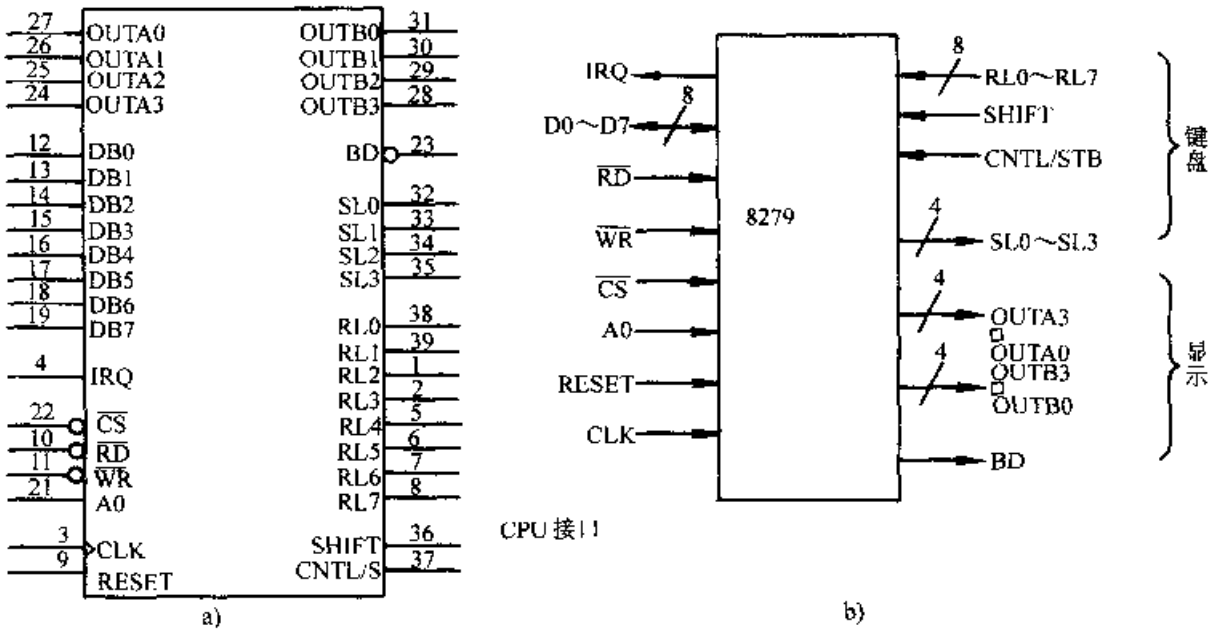


图 6-55　8279 管脚及引脚功能

a）管脚配置　b）引线功能

其引脚功能分述如下：

·D0～D7（数据总线）：双向、三态总线，和系统数据总线相连；用于CPU和8279间的数据/命令传送。

·CLK（系统时钟）：输入线，为8279提供内部时钟的输入端。

·RESET（复位）：输入线，当RESET＝1时，8279复位，其复位状态为：

16个字符显示；

编码扫描键盘——双键锁定；

程序时钟编程为31。

·$\overline{CS}$（片选）：输入线，当$\overline{CS}$＝0时8279被选中，允许CPU对其读、写，否则被禁止。

·A0（数据选择）：输入线。当A0＝1时CPU写入数据为命令字，读出数据为状态字；A0＝0时CPU读、写的字节均为数据。

·$\overline{RD}$、$\overline{WR}$（读、写信号）：输入线。低电平有效，来自CPU的控制信号，控制8279的读、写操作。

·IRQ（中断请求）：输出线。高电平有效。

在键盘工作方式中，当FIFO/传感器RAM存有数据时，IRQ为高电平。CPU每次从RAM中读出数据时，IRQ变为低电平。若RAM中仍有数据，则IRQ再次恢复为高电平。

在传感器工作方式中，每当检测到传感器状态变化时，IRQ就出现高电平。

·SL0～SL3（扫描线）：输出线。用来扫描键盘和显示器。它们可以编程设定为编码（16中取1）或译码输出（4取1）。

·RL0～RL7（回复线）：输入线。它们是键盘矩阵或传感器矩阵的列（或行）信号输入线。

·SHIFT（移位信号）：输入线、高电平有效。该输入信号是8279键盘数据的次高位（D6），通常用来扩充键开关的功能，可以用作键盘上、下档功能键。在传感器方式和选通方式中，SHIFT无效。

·CNTL/STB（控制/选通）：输入线。高电平有效。

在键盘工作方式时，该输入信号是键盘数据的最高位（D7），通常用来扩充键开关的控制功能，作为控制功能键用。

在选通输入方式时，该信号的上升沿可从将来自RL0～RL7的数据存入FIFO RAM中。

在传感器方式下，该信号无效。

·OUTA0～OUTA3（A组显示信号）：输出线。

·OUTB0～OUTB3（B组显示信号）：输出线。

这两组引线都是显示数据输出线，与多位数字显示的扫描线SL0～SL3同步。

两组可以独立使用，也可以合并使用。

· $\overline{BD}$（显示消隐）：输出线。低电平有效。该信号在数字切换显示或使用消隐命令时，将显示消隐。

(3) 命令格式与命令字

8279 的操作方式是通过 CPU 对 8279 送入命令实现编程。当数据选择端 A0 置 1 时，CPU 对 8289 写入的数据为命令字，读出的数据为状态字。

8279 共有八条命令，其功能及命令字定义分述如下。

1) 键盘/显示方式设置命令字

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	D	D	K	K	K

其中：

· D7 D6 D5＝000　　为方式设置命令特征位。

· D D（D4、D3）：用来设定显示方式，其定义如下：

0 0　　8 个字符显示，左入口

0 1　　16 个字符显示，左入口

1 0　　8 个字符显示，右入口

1 1　　16 个字符显示，右入口

所谓左入口，即显示位置从最左一位（最高位）开始，以后逐次输入的显示字符逐个向右顺序排列。所谓右入口，则是显示位置从最右一位（最低位）开始，以后逐次输入显示字符时，已有的显示字符依次向左移动。

· K K K（D2、D1、D0）：用来设定七种键盘、显示工作：

0 0 0　　编码扫描键盘，双键锁定

0 0 1　　译码扫描键盘，双键锁定

0 1 0　　编码扫描键盘，N 键轮回

0 1 1　　译码扫描键盘，N 键轮回

1 0 0　　编码扫描传感器矩阵

1 0 1　　译码扫描传感器矩阵

1 1 0　　选通输入，编码显示扫描

1 1 1　　选通输入，译码显示扫描

双键锁定与 N 键轮回是多键按下时的两种不同的保护方式；双键锁定为两键同时按下时提供的保护方法。在消抖动周期里，如果有两键同时被按下，则只有其中一个键弹起，而另一个键保持在按下位置时，才被认可。N 键轮回为 N 键同时按下时的保护方法。当有若干键按下时，键盘扫描能够根据发现它们的顺序，依

次将它们的状态送入 FIFO RAM 中。

2）程序时钟命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	P	P	P	P	P

其中：

• D7 D6 D5＝001 为时钟命令特征位。

• PPPPP（D4、D3、D2、D1、D0）用来设定对外部输入 CLK 端的时钟进行分频的分频数 N。N 取值为 2～31。例如外部时钟频率为 2MHz，PPPPP 被置成为 10100（N＝20），则对输入的外部时钟 20 分频，以获得 8279 内部要求的 100kHz 的基本频率。

3）读 FIFO/传感器 RAM 命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	AI	×	A	A	A

其中：

• D7 D6 D5＝010 为读 FIFO/传感器 RAM 命令特征位。该命令只在传感器方式时使用。在 CPU 读传感器 RAM 之前，必须用这条命令来设定传感器 RAM 中的 8 个地址（每个地址一个字节）。

• AAA（D2、D1、D0）为传感器 RAM 中的八个字节地址。

• AI（D4）为自动增量特征。当 AI＝1 时，每次读出传感器 RAM 后地址自动加 1 使地址指针指向下一个存储单元。这样，下一个数据便从下一个地址读出，而不必重新设置读 FIFO/传感器 RAM 命令。

在键盘工作方式中，由于读出操作严格按照先入先出顺序，因此，不需使用这条命令。

4）读显示 RAM 命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	AI	A	A	A	A

其中：

• D7 D6 D5＝011 为读显示 RAM 命令字的特征位。该命令字用来设定将要读出的显示 RAM 地址。

• AAAA（D3、D2、D1、D0）用来寻址显示 RAM 中的存储单元。由于显示 RAM 中有 16 个字节单元，故需要 4 位寻址。

• AI（D4）为自动增量特征位。AI=1 时，每次读出后地址自动加 1，指向下一地址。

5）写显示 RAM 命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	AI	A	A	A	A

其中：

• D7 D6 D5=100 为写显示 RAM 命令字特征位。在写显示 RAM 之前用这个命令字来设定将要写入的显示 RAM 地址。

• AAAA（D3、D2、D1、D0）为将要写入的显示 RAM 中的存储单元地址。

• AI（D4）为自动增量特征位，AI=1 时，每次写入后地址自动加 1，指向下一次写入地址。

6）显示禁止写入/消隐命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	×	IW A	IW B	BL A	ML B

其中：

• D7 D6 D5=101 为显示禁止写入/消隐命令特征位。

• IW/A、IW/B（D3、D2）为 A、B 组显示 RAM 写入屏蔽位。由于显示寄存器分成 A、B 两组，可以单组送数，故用两位来分别屏蔽。当 A 组的屏蔽位 D3=1 时，A 组的显示 RAM 禁止写入。因此，从 CPU 写入显示器 RAM 数据时，不会影响 A 组的显示。这种情况通常在采用双 4 位显示器时使用。因为两个四位显示器是相互独立的。为了给其中一个四位显示器输入数据而又不影响另一个四位显示器，因此必须对另一组的输入实行屏蔽。

• BL/A、BL/B（D1、D0）为消隐设置位。用于对两组显示输出消隐。若 BL=1，对应组的显示输出被消隐。当 BL=0，则恢复显示。

7）清除命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	C_D	C_D	C_D	C_F	C_A

其中：

• D7 D6 D5=110 为清除命令特征位。

• C_D C_D C_D（D4、D3、D2）用来设定清除显示 RAM 方式，共有下表所列的

四种消除方式。

D4	D3	D2	清除方式
1	0	×	将显示 RAM 全部清零
	1	0	将显示 RAM 清成 20H（A 组＝0010；B 组＝0000）
	1	1	将显示 RAM 全部置 1
0	不清除（若 C_A＝1，则 D3、D2 仍有效）		

· C_F(D1)用来置空 FIFO 存储器，当 C_F＝1 时，执行清除命令后，FIFO RAM 被置空，使中断输出线复位。同时，传感器 RAM 读出地址也被置为 0。

· C_A（D0）为总清的特征位。它兼有 C_D 和 C_F 的联合效能。在 C_F＝1 时，对显示的清除方式由 D3、D2 的编码决定。

清除显示 RAM 约需 160μs。在此期间 FIFO 状态字的最高位 Du＝1，表示显示无效。CPU 不能向显示 RAM 写入数据。

8）结束中断/错误方式设置命令

命令格式：

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	E	×	×	×	×

其中：

· D7 D6 D5＝111 为该命令的特征位。此命令有下列两种不同的作用：

① 作为结束中断命令，在传感器工作方式中使用。每当传感器状态出现变化时，扫描检测电路就将其状态写入传感器 RAM，并启动中断逻辑，使 IRQ 变高，向 CPU 请求中断，并且禁止写入传感器 RAM。此时，若传感器 RAM 读出地址的自动递增特征没有置位（AI＝0），则中断请求 IRQ 在 CPU 第一次从传感器 RAM 读出数据时就被清除。若自动递增特征已置位（AI＝1），则 CPU 对传感器 RAM 的读出并不能清除 IRQ，而必须通过给 8279 写入结束中断/错误方式设置命令才能使 IRQ 变低。因此，在传感器工作方式中，此命令用来结束传感器 RAM 的中断请求。

② 作为特定错误方式设置命令。在 8279 已被设定为键盘扫描 N 键轮回方式以后，如果 CPU 给 8279 又写入结束中断/错误方式设置命令（E＝1），则 8279 将以一种特定的错误方式工作。这种方式的特点是：在 8279 的消颤周期内，如果发现多个按键同时按下，则 FIFO 状态字中的错误特征位 S/E 将置'1'，并产生中断请求信号和阻止写入 FIFO RAM。

上述八种用于确定 8279 操作方式的命令字皆由 D7D6D5 特征位确定，8279 能自动寻址相应的命令寄存器。因此，写入命令时惟一的要求是使数据选择信号 A0＝1。

（4）状态格式与状态字

8279 的 FIFO 状态字，主要用于键盘和选通工作方式，以指示 FIFO RAM 中的字符数和有无错误发生。其格式为：

D7	D6	D5	D4	D3	D2	D1	D0
Du	S/E	O	U	F	N	N	N

其中：

• Du（D7）为显示无效特征位。当 Du＝1 时表示显示无效。当显示 RAM 由于清除显示或全清命令尚未完成时，Du＝1。

• S/E（D6）为传感器信号结束/错误特征位。该特征位在读出 FIFO 态状字时被读出，而在执行 CF＝1 的清除命令时被复位。当 8279 工作在传感器工作方式时，若 S/E＝1，表示传感器的最后一个传感器信号已进入传感器 RAM；而当 8279 工作在特殊错误方式时，若 S/E＝1，则表示出现了多键同时按下的错误。

• OU（D5、D4）为超出、不足错误特征位。对 FIFO RAM 的操作可能出现两种错误：超出或不足。当 FIFO RAM 已经充满时，其他的键盘数据还企图写入 FIFO RAM，则出现超出错误，超出错误特征位 O（D5）置'1'；当 FIFO RAM 已经置空时，CPU 还企图读出，则出现不足错误，不足错误特征位 U（D4）置'1'。

• F（D3）表示 FIFO RAM 中是否已满。F＝1 表示 FIFO RAM 中已满。

• NNN（D2、D1、D0）表示 FIFO RAM 中的字符个数。

5. 8279 键盘、显示器配置结构

8279 是专用键盘、显示控制芯片，能对显示器自动扫描；能识别键盘上按下键的键号，可充分提高 CPU 工作效率。8279 与 MCS-51 接口方便，由它构成的标准键盘、显示接口在单片机应用系统中使用愈来愈广泛。

8279 的键盘、显示电路及与 8031 接口的一般连接方法如图 6-56 所示。

6.3.4 单片机应用系统中的典型键盘、显示接口技术

在单片机应用系统中，同时需要使用键盘与显示器接口时。为了节省 I/O 口线，常常把键盘和显示电路做在一起，构成实用的键盘、显示电路。下面介绍几个典型的键盘、显示器接口及软件清单。

1. 8155 扩展 I/O 口的键盘、显示器接口

（1）接口电路

图 6-57 是一个典型实用的、采用 8155 并行扩展口构成的键盘显示器电路。

图中只设置了 32 个键。如果增加 PC 口线，可以加多按键，最多可达 64 个键。可根据需要设置。

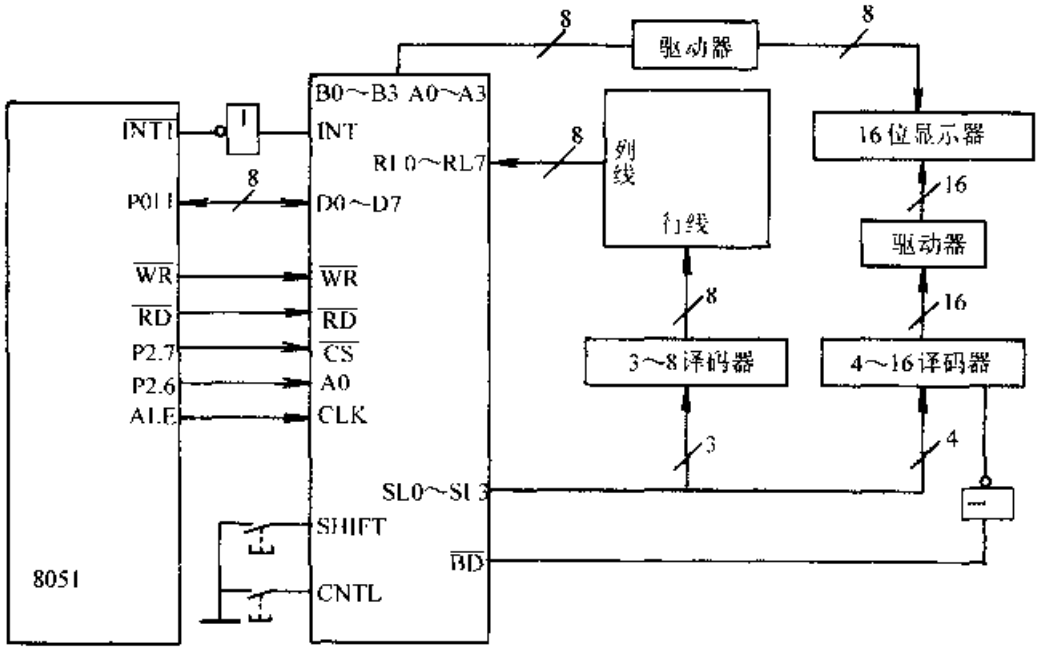


图 6-56　8279 的键盘、显示配置结构图

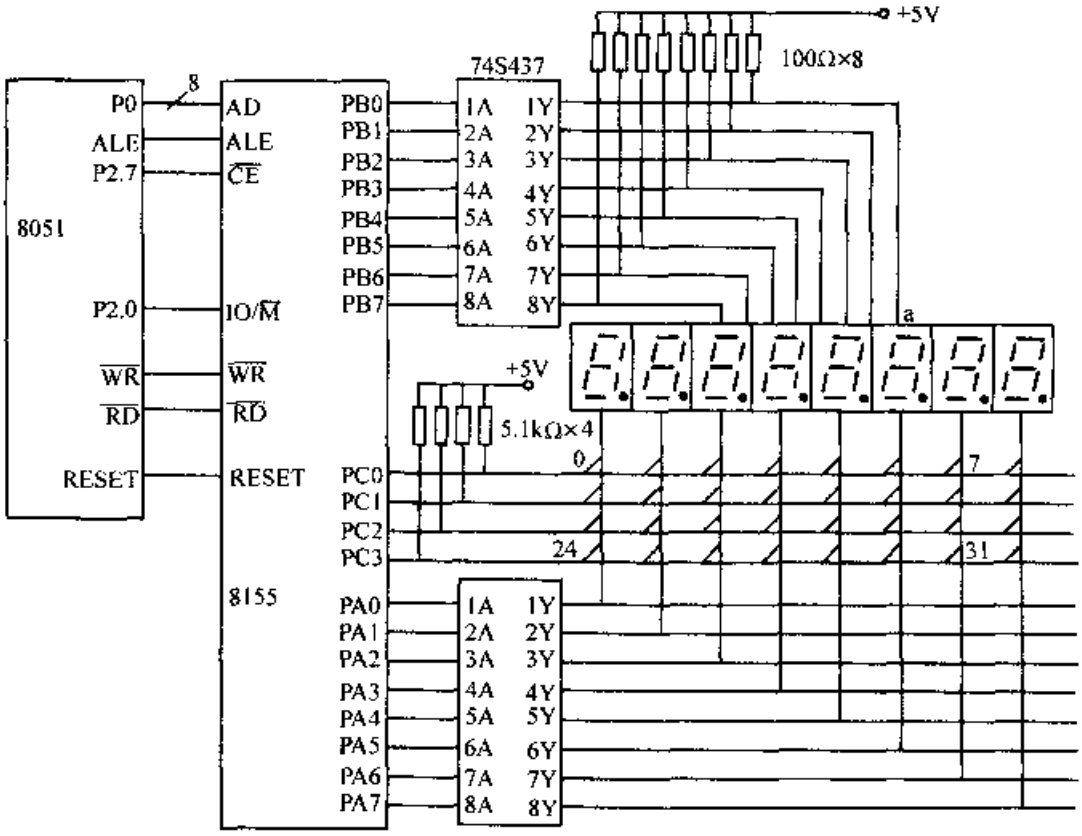


图 6-57　8155 扩展 I/O 口的键盘、显示器接口电路

LED 显示器采用共阴极。段选码由 8155PB 口提供，位选码由 PA 口提供。键盘的列输入由 PA 口提供，行输出由 PC0～PC3 提供。

LED 采用动态显示软件译码，键盘采用逐列扫描查询工作方式。

LED 的驱动采用输出八位驱动器 74S437。

(2) 软件设计

由于键与显示器作成一个接口电路，因此在软件中合并考虑键盘查询与动态显示，键盘消颤的延时子程序用显示程序替代。

8155 地址安排：

命令字寄存器　0x7FF8

口 A　0x7FF9

口 B　0x7FFA

口 C　0x7FFB

口 A 和口 B 为基本输出方式，口 C 为基本输入方式，不使用中断。

命令字为：0x03。

```
#include"reg51.h"
#include"absacc.h"
#define COMM XBYTE[0x7FF8]
#define PA XBYTE[0x7FF9]
#define PB XBYTE[0x7FFA]
#define PC XBYTE[0x7FFB]
char code dispdata[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,
0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
char dis  dat[8];
void   delay()   /*延时 1ms*/
{
    TL0=-500%256;
    TH0=-500/256;
    TR0=1;
    while(! TF0);
    TF0=0;
    TR0=0;
}
void disp(char ch)
{
    char ch1,ch2;
```

```
    switch(ch){
        case 0xFE:ch2=0;break;
        case 0xFD:ch2=1;break;
        case 0xFB:ch2=2;break;
        case 0xF7:ch2=3;break;
        case 0xEF:ch2=4;break;
        case 0xDF:ch2=5;break;
        case 0xBF:ch2=6;break;
        case 0x7F:ch2=7;
    }
    for (ch1=0;ch1<8;ch1++){
        PA=ch;
        PB=dis_dat[ch2++];
        If(ch2==8)ch2=0;
        ch=~ch;
        ch=ch<<1;
        if(ch==0)ch=1;
        ch=~ch;
        delay();
    }
}
void main()
{
    char ch=0xFE,ch1,ch2;
    TMOD=0x02;
    TH0=-500/256;
    TL0=-500%256;
    for(ch1=0;ch1<8;ch1++)
    {
        PA=ch;
        PB=dis_dat[ch1];
        ch2=PC&0x0F;
        if(ch2!=0x0F)
        {
            disp(ch);
```

```
            ch2=PC&0x0F;
            if(ch2!=0x0F)keyproc(ch,ch2);
        }
        ch=~ch;
        ch=ch<<1;
        if(ch==0)ch=1;
        ch=~ch;
        delay();
    }
}
```

程序中的 keyproc () 函数的功能是根据键盘列线和行线的值确定键值并作相应的处理。

[**课程实践**]：时间显示电路设计。对图 6-54 的电路实现一个电子时钟，显示器的低两位显示时间的秒，第 4、5 两位显示分钟，高两位显示小时，第 3 位和第 6 位显示中间的横杠，分隔小时、分钟和秒。

键盘的安排。需要的按键有：0～9 数字键；时间设定键；设定位左移键和右移键。键号分配：0～9 号键为数字 0～9；31 号键为设定键；30 号和 29 号键为左移键和右移键。

程序设计思路：

设置 LED 数字 0～9 的显示编码数组和 8 位 LED 显示数据数组，前者存于 code区域，后者存于内部 RAM 区域。

定时器 T0 产生 1ms 定时，用查询方式。用于 LED 动态刷新。

键盘扫描在 LED 动态刷新期间完成。当按下设定键后，时间计数仍然工作，但不再更新显示数组，设定从分钟的个位开始，被设定的位闪烁，数字键 0～9 按下时，就存入时间计数器的对应位，并被显示。用左右移动键选择要设定的位。当再次按下设定键时，设定结束，回到正常计时状态。

定时器 T1 产生 100ms 定时，作时间计数器的计数时钟。

动态刷新程序的操作步骤：

PA＝0x7F；输出秒个位的显示编码数据，查询 PC0～PC3 是否全 1，若是延时 1ms（启动定时器 T0，查询溢出标志，然后关闭定时器），改变位选信号，显示下一位。若 PC0～PC3 不为全 1，调用一个将 8 位 LED 刷新显示一遍的函数，函数中延时时间总和不小于 8ms，作用是去除抖动影响，再查询 PC0～PC3 的状态，以确定是否有键按下，并做适当处理。

键盘处理程序思路：

设置时钟设定标志 bit　inset 和当前设定位标志 char　set_bit。正常时间计

数时，inset 值为 0，当按下设定键时，将 inset 置为 1，将 set bit 设为 3（秒的个位序号为 0），对应位 LED 闪烁显示；按下左移键时，set_bit 增 1，按下右移键时，set_bit 减 1，但 set_bit 的值不能为 2 和 5；按下数字键时，对应位改为按下的键值。当再一次按下设定键时，将 inset 清 0，结束时间设定。

时间计数器工作思路：

设置小时、分钟、秒计数变量，char hour，mini，seco，s。s 用于定时器 T1 作计数器，T1 每产生一次中断，s 增 1，当 s 等于 10 时，一秒定时到，修改计数变量hour、mini 和 seco 的值，并改写显示数组的内容。显示数组中存放要显示数据的 LED 编码值。

2. 8279 键盘、显示器接口电路

(1) 8279 接口与编程的一般方法

1) 接口电路的一般连接方法：8279 的键盘、显示电路及与 8051 接口的一般连接方法如图 6-56 所示。

8279 键盘配置最大为 8×8。扫描线由 SL0～SL2 通过 3～8 译码器提供，接入键盘列线（设扫描线为列线）；查询线由反馈输入线 RL0～RL7 提供，接入键盘线（设定查询线为列线）。

8279 显示器最大配置为 16 位显示，位选线由扫描线 SL0～SL3 经 4～16 译码器、驱动器提供；段选线 B0～B3、A0～A3 通过驱动器提供。BD 信号线可用来控制译码器，实现显示器的消隐。

与 8051 联接无特殊要求，除数据线 P0 口、$\overline{WR}$、$\overline{RD}$可直接连接外，$\overline{CS}$由 8051 地址线选择。时钟由 ALE 提供，A0 选择线也可由地址线选择。8279 的 RESET 按图 6-54 连接，为上电复位方式。使用 SHIFT 和 CNTL/STB 时，可按图 6-53 连接，SHIFT 和 CNTL/STB 内部有上拉电阻。

8279 的中断请求线需经反相器与 8051 的$\overline{INT0}$/$\overline{INT1}$相连。

ALE 可直接与 8279 CLK 相连，由 8279 设置适当的分频数，分频至 100kHz。

2) 8279 键盘、显示接口的应用特性：对 8279 键盘、显示器编程时，必须充分了解典型键盘、显示接口的应用特征。它包括操作命令、操作状态、输入数据的字节格式定义；显示器在不同输入方式下的移位/填入方式；使用外部译码和内部译码时键盘、显示器的结构等。下面分别介绍。

· 8279 的操作命令

8279 的操作命令已在前面详细介绍，此处不再重复。下页的表是 8279 操作命令字及其功能特征速查表，供编程时查对。

8279 键盘、显示器编程时，命令字的入口地址由$\overline{CS}$和 A0 决定。$\overline{CS}$=0，A0=1 时为命令字输入口地址。

· 8279 的 FIFO 状态查询

8279 命令功能一览表

命令特征位			功能特征位				
D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0
键盘/显示器工作方式			左端送入	8×8 显示	双键锁定 0	1	编码扫描
			1	1	N 键轮回 1	0	1
			右端送入	16×8 显示	传感器矩阵 1	1	译码扫描
					选通输入显示扫描		
0	0	1	×	×	×	×	×
程序时钟			2 — 31 分频				
0	1	0	1	×	×	×	×
读 FIFO/传感器 RAM			传感器 RAM 自动加 1	—	传感器 RAM 的 8 个字节地址		
0	1	1	1	×	×	×	×
读显示 RAM				显示 RAM16 个字节地址			
1	0	0	1	×	×	×	×
写显示 RAM			自动加 1	显示 RAM16 个字节地址			
1	0	1	×	1	1	1	1
显示器写禁止/消隐			—	禁止写 A 口	禁止写 B 口	消隐 A 口	消隐 B 口
1	1	0	1	0	×	1	1
清除（清除显示寄存器 A、B 组输出）			允许清除	A、B 全部清零 1	0	FIFO 成空状态；中断复位；传感器读出地址置 0	总清零
				A、B 清成 20H 1	1		
				A、B 皆置 1			
1	1	1	1	×	×	×	×
结束中断/错误方式设置			特殊工作方式				

在键输入或选通输入方式中，可以查询 FIFO RAM 的状态，以了解输入数据缓冲器 FIFO 中的字符个数和是否出错。状态字节的读出地址和命令字输入地址相同（$\overline{CS}=0$，A0=1）。

· 8279 的数据输入/输出

对 8279 输入数据（如显示数据，键输入数据、传感器矩阵数据等）时，要选择数据输入输出口地址。8279 的数据输入/输出口地址由$\overline{CS}=0$，A0=0 确定。

在键盘扫描方式中，8279 中键输入数据按下列格式存放：

D7	D6	D5 D4 D3	D2 D1 D0
CNTL	SHIFT	SCAN	RETURN

RETURN 为键所在的行号（由 RL0～RL7 状态确定）。

SCAN 为键所在的列号（由 SL0～SL2 确定）。

SHIFT 为控制键 SHIFT 的状态位。

CNTL 为控制键 CNTL 的状态位。

控制键 CNTL、SHIFT 为单独的开关键。CNTL 与其他键连用作特殊命令键，SHIFT 可用作上、下档控制键。

在传感器方式或选通方式中，8 位输入数据为 RL0～RL7 的状态。

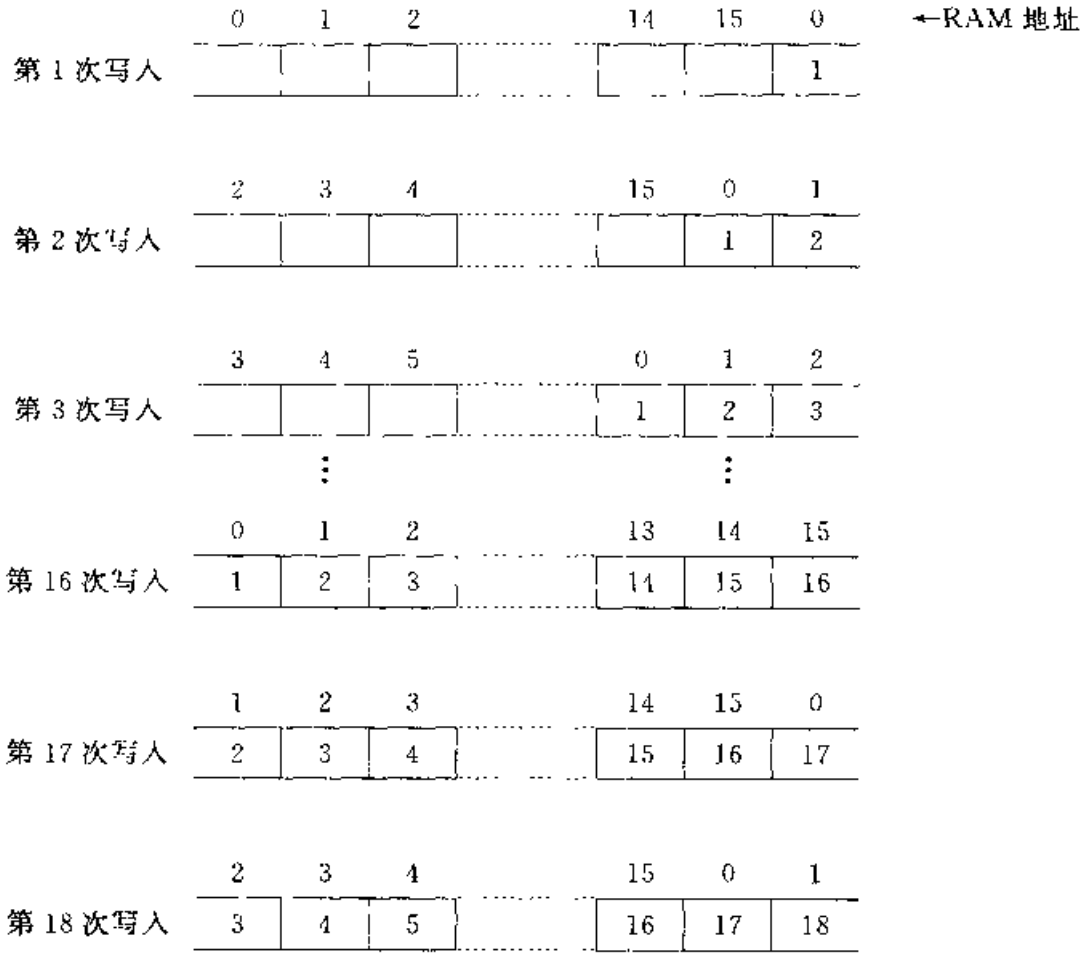
D7	D6	D5	D4	D3	D2	D1	D0
RL7	RL6	RL5	RL4	RL3	RL2	RL1	RL0

· 显示器的填入/移位方式

8279 可外接 8 位或 16 位 LED 显示器，每位显示器对应一个 8 位（段选码）显示缓冲器。CPU 将显示数据写入缓冲器时，有左端送入和右端送入两种方式。左端送入为依次填入方式，显示缓冲区 RAM 地址。0～15 分别对应于显示器的 0 位～15 位，CPU 依次从 0 地址或某一地址开始将段选数据写入显示缓冲区。地址大于 15 时，再从 0 地址写入，如下所示：

	0	1	2	…	14	15	←RAM 地址
第 1 次写入	1						
第 2 次写入	1	2					
⋮	⋮				⋮		
第 16 次写入	1	2	3		15	16	
第 17 次写入	17	2	3		15	16	
第 18 次写入	17	18	3		15	16	

右端送入为移位方式。输入数据总是写入右边的显示缓冲器。数据写入显示缓冲器后，原来缓冲器内容左移一个字节，写入过程如下：



在右端送入方式中，显示器位置和缓冲器 RAM 地址不相对应。

·8279 的内部译码与外部译码

在键盘、显示器工作方式中，SL0～SL3 为键盘的列扫描线和动态显示的位选线。

当选择内部译码时（D0＝1），SL0～SL3 每一时刻只能有一位为低电平输出。此时 8279 只能外接 4 位显示器和 4×8 的键盘。

当选择外部译码时（D0＝0），SL0～SL3 呈计数分频式波形输出。外接 4～16 译码器可外接 16 位显示器位选。外接 3～8 译码器时，与 RL0～RL7 构成 8×8 矩阵键盘（键输入数据格式中只能计入 SL0～SL2 的 8 种状态）。

·键盘键值的给定

对于图 6-55 中的 8×8 键盘，如果规定扫描线（SL0～SL2）为列线，回复线（RL0～RL7）为行线。则在数据输入格式中，用 D5、D4、D3 表示 SL0～SL2 的 8 个译码状态，用 D2、D1、D0 表示 RL0～RL7 的 8 个状态。因此，8×8

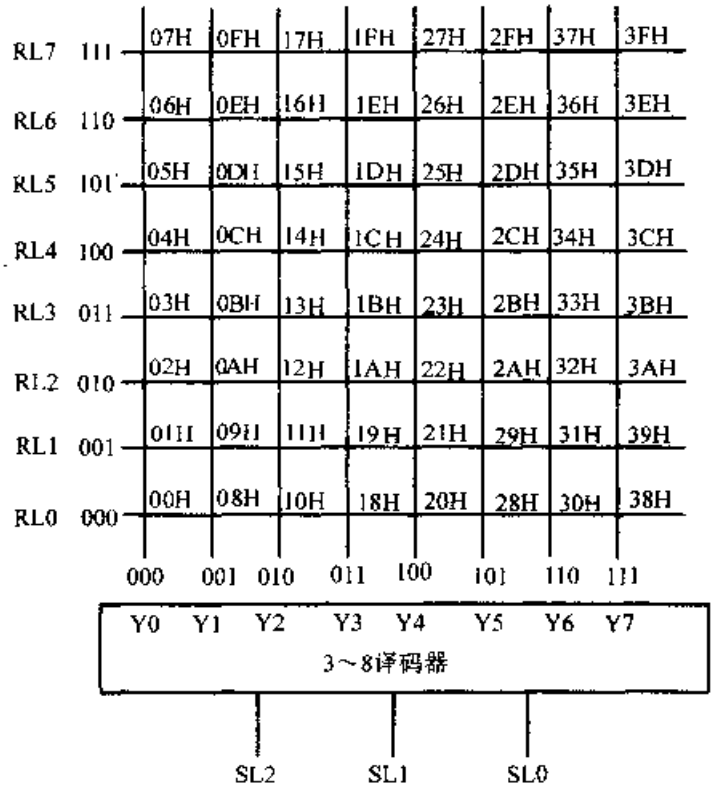


图 6-58　8×8 键盘的键值与键号

键盘的键值如图 6-58 所示。由于 64 个键的键值均依次排列，也可作为键号使用。

3）8051 和 8279 键盘、显示器接口的编程方法：对于图 6-55 所示的一般接口电路，键盘的读出既可用中断方式，也可用查询方式。

图 6-59 是一个实际的 8279 键盘、显示器接口电路。

对图 6-59 所示的 8279 键盘、显示器接口电路，8 位 LED 显示，16 个键盘，键盘采用查询方式读出。8051 晶振为 6MHz。

8279 地址分配：

命令字/状态寄存器　　0x7FFF　　（A0＝1）

数据口　　0x7FFE　　（A0＝0）

8279 工作方式：左端送入，8×8 显示，双键锁定，编码扫描。命令字为　0x08。

fosc 为 6MHz，ALE 为 1MHz，需 10 分频得到 8279 所需的 100kHz 的时钟信号，命令字为　0x2A。

程序首先清除 8279 的显示和 FIFO 队列，清除命令字为　0xD3。

对键盘的查询使用读 FIFO 状态命令，即读状态寄存器。

当 8279 FIFO 队列中有数据（有按键键值）时，需读 FIFO，命令字为　0x40。送入命令字后，读 8279 数据口，即可得到键值，在没有使用 CONTROL 键和 SHIFT 键时，数据的低 6 位为键值。键值与键号相同，如图中所示为

00H～10H。

显示的数据要写入显示 RAM，写显示 RAM 的命令字为 0x90。

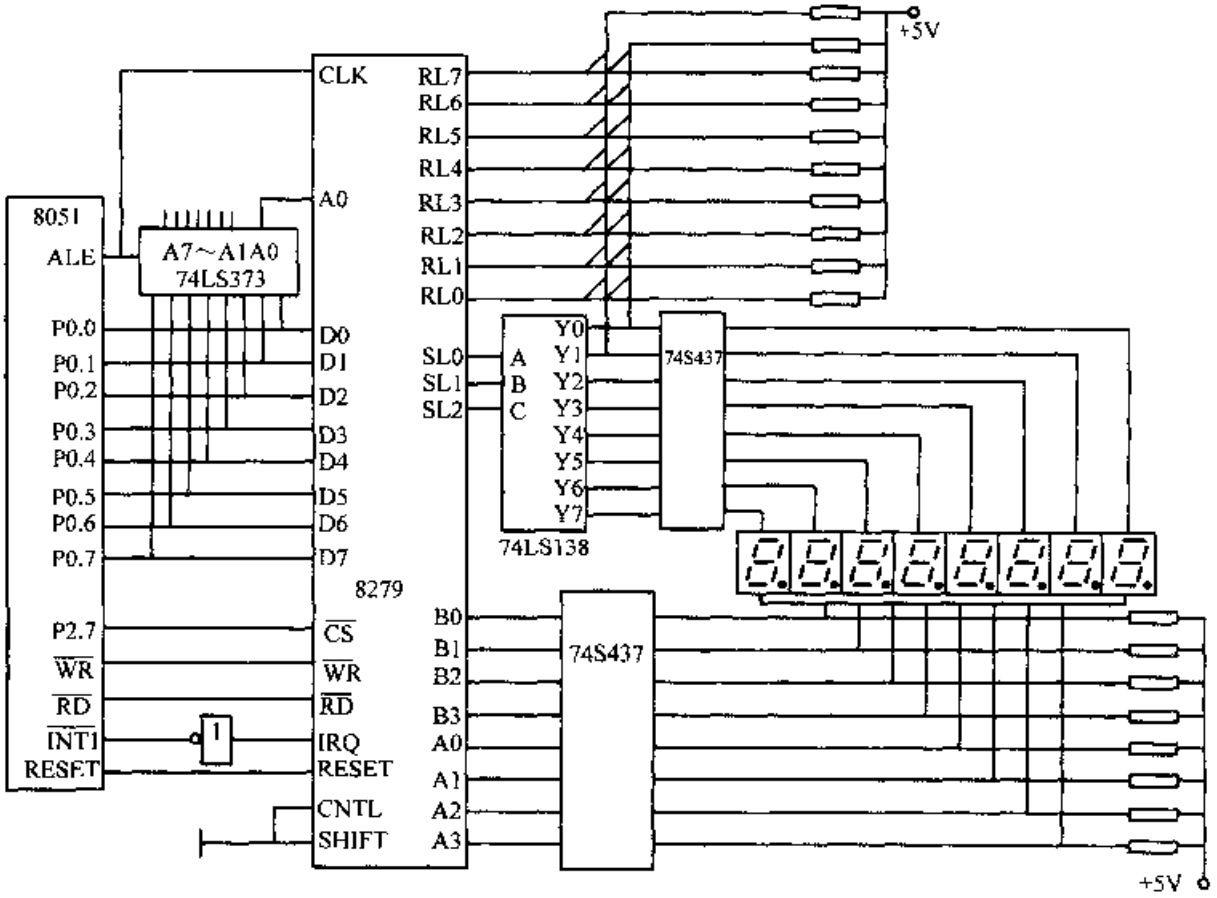


图 6-59 实际的 8279 键盘、显示器接口电路

程序清单如下：

```
#include"reg51.h"
#include"absacc.h"
#define COMM XBYTE[0x7FFF]
#define DATA XBYTE[0x7FFE]
char code dispdata[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,
0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
char dis_dat[8];
void main()
{
    char ch;
    COMM=0xD3;
    do {ch=COMM};
    while((ch &0x80)==0x80);   /*等待清除完成*/
```

```
    COMM=0x2A;
    COMM=0x08;
    COMM=0x90;
    For(ch=0;ch<8;ch++)
        DATA=dis_dat[ch];
    do{
        ch=COMM;
        ch=ch&0x0F;
    }while(ch==-0);
    COMM=0x40;
    ch=DATA;
    ch=ch&0x3F;
    …
}
```

上述程序中，由于对键按下是采用查询方式，故设有等待键输入指令。

[**课程实践**]：采用 8279 芯片实现键盘、LED 显示器电路。画出线路图并编程。与采用 8155 或 8255 的键盘显示器电路相比，使用 8279 要简单可靠。

习题和思考题

1. 在一个 8031 应用系统中，8031 以中断方式通过并行接口 74LS244 读取 A/D 器件 ADC0808 的转换结果。试画出有关逻辑电路，并编写读取 A/D 结果的中断服务程序。
2. 在一个 fosc 为 12MHz 的 8031 系统中接有一片 D/A 器件 DAC0832，它的地址为 7FFFH，输出电压为 0～5V。请画出有关逻辑框图，并编写一个程序，使其运行后能在示波器上显示出锯齿波（设示波器 X 向扫描频率为 50μs，Y 向扫描频率为 1V/格）。
3. 在一个 fosc 为 12MHz 的 8031 系统中接有一片 A/D 器件 ADC0809，它的地址为 7FF8H～7FFFH。试画出有关逻辑框图，并编写 ADC0809 初始化程序和定时采样通道 2 的程序（假设采样频率为 1ms 一次，每次采样 4 个数据，存于 8031 内部 RAM70H～73H 中）。
4. 在一个 8031 系统中，8031 的串行口级联了三片 D/A 器件 MC144111。试画出有关的逻辑图，并编写一个程序，使 8031 内部 RAM 61H～69H 单元内容转换成 12 路模拟量。
5. 在一个 fosc 为 12MHz 的 8031 系统中，扩展了一片 8155，用 8155 产生 100ms 的定时中断。试画出有关逻辑框图，并编制 8031 的外中断处理程序框图，使 8155 的 PA 口、PB 口的 16 位分别产生周期为 0.2、0.4、0.5、1、2、4、8、16、32、64、96、128、160、192、224、256s 的方波信号。
6. 试设计一个 8031 应用系统，它除了扩展有 32K 字节 EPROM、8K 字节 RAM 外，还扩展了一片并行接口电路 8255、一片串行接口电路 8251、一片键盘显示器接口芯片 8279、一片 RAM/IO 扩展器 8155、一片 A/D 电路 ADC0809、一片 D/A 电路 ADC0832，组成了一个具

有实时控制、数据采集和处理以及远程通信等功能的 MCS—51 系统（提示：在设计时应考虑到 8031 P0 口的负载能力而酌情使用总线驱动电路 74244、74245 等）。

7. 在一个 8031 系统中扩展一片 8155，8155 外接 6 位 LED 显示器和 2×4 键盘，试画出电路图，并编写出相应的显示子程序和读键盘子程序。
8. 试画出 6×6 键盘、6 位共阴极显示器和 8279 的接口电路图，并编写一个子程序，用查询方法将键盘上输入键的键号送显示器显示。

参 考 文 献

1 薛钧义等．MCS-51/96 系列单片微型计算机及其应用．西安：西安交通大学出版社，1990

2 孙涵芳等．MCS 51/96 系列单片机原理及应用．北京：北京航空航天大学出版社，1988

3 张友德等．单片微型计算机原理、应用与实验．上海：复旦大学出版社，1992

4 何立民．MCS—51 系列单片机应用系统设计．北京：北京航空航天大学出版社，1990

5 孙育才．MCS 51 系列单片微型计算机及其应用．南京：东南大学出版社，1987

6 王秀玲等．A/D D/A 转换接口技术及数据采集系统设计．北京：清华大学出版社，1984

7 马忠梅等．单片机的 C 语言应用程序设计．北京：北京航空航天大学出版社，2001

ISBN 7-111-11141-9/ TP·2657(课)

封面设计：电脑制作 蒋杨

ISBN 7-111-11141-9
9 787111 111412 >

定价：18.00 元

地址：北京市百万庄大街22号 邮政编码：100037
联系电话：(010) 68326294 网址：http://www.cmpbook.com
E-mail:online@cmpbook.com