第 5 章

CHAPTER 5

单片机 I/O 口及应用

I/O 口是单片机非常重要的组成部分。本章在讲述单片机的 I/O 口结构原理的基础上，讲解了数码管、键盘、液晶显示器的结构原理、编程控制及应用。

键盘、数码管或液晶显示器是构成单片机应用系统的基本部分，并且与单片机的 I/O 口是一种简单连接(没有复杂的时序、复杂的寄存器)，非标准接口(第 9 章为标准接口)，属于 I/O 口的基本应用。通过本章内容的学习，为后面各章的学习及构建单片机应用系统奠定良好的基础。

5.1 单片机 I/O 口结构原理

MCS-51 单片机的 4 个 8 位端口都是准双向口，每个端口的每一位都可以独立地用作输入或输出。每个端口都有一个锁存器(即端口映射寄存器 P0～P3)、一个输出驱动器和一个输入缓冲器。输出时，数据可以锁存，输入时数据可以缓冲。但这 4 个端口功能不完全相同，内部结构也有区别。

当单片机执行输出操作时，CPU 通过内部总线把数据写入锁存器。当单片机执行输入操作时分两种情况，一种情况是读取锁存器原来的输出值，另一种情况是打开端口的缓冲器读取引脚上的输入值，究竟是读取引脚还是读取输出锁存器，与具体指令有关的，后面讨论。

如果单片机系统没有扩展片外存储器，则 4 个端口都可以作为准双向通用 I/O 口使用。在扩展有片外存储器的系统中，P2 口输出高 8 位地址，P0 口为双向总线口，分时输出低 8 位地址、读入指令和进行数据输入/输出。

熟悉单片机的 I/O 口的逻辑电路，不但有利于正确合理使用端口，而且会对设计单片机的外围电路有所启发。下面从结构最简单的 P1 口开始讲解，依次到最复杂的 P0 口。

5.1.1 P1 口

P1 口是一个准双向口，用作通用 I/O 口。从结构上相对来说 P1 口最简单，其端口某一位的原理结构如图 5-1 所示，主要由输出锁存器、场效应管(FET)T 驱动器，控制从锁存器输入的

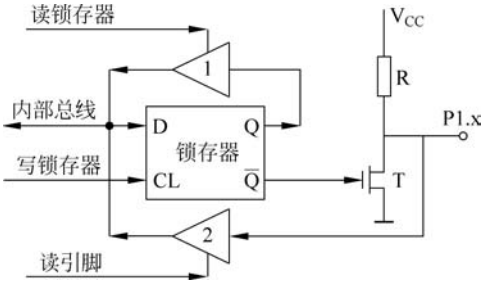

图 5-1 P1 口某一位的原理结构

三态缓冲器 1，控制从引脚输入的三态缓冲器 2，以及 T 上拉电阻 R(实为一 FET)等部分组成。

P1 口的每一位都可以分别定义为输入或输出，既可以对各位进行整体操作，也可以对各位进行分别操作。

1. P1 口输出

输出 1 时，将 1 写入 P1 口某一位的锁存器，使输出驱动器的场效应管 T 截止，该位的引脚由内部上拉电阻拉成高电平，输出为 1。输出 0 时，将 0 写入锁存器，使场效应管导通，则输出引脚为低电平。由于 P1 口各位有上拉电阻，所以在输出高电平时，能向外提供拉电流负载，外部不必再接上拉电阻。

2. P1 口输入

当 P1 口的某位用作输入时，该位的锁存器必须锁存输出 1(该位先写 1)，使输出场效应管 T 截止，才能够正确输入，这时从引脚输入的值决定于外部信号的高低，引脚状态经“读引脚”信号打开的三态缓冲器 2，送入内部总线。

如果输入时不向对应位先写 1，有可能前面的操作使引脚输出 0，场效应管 T 处于导通状态，引脚被箝位为 0，这样，不管外部信号为何状态，从引脚输入的永远为 0。单片机端口输入前必须先向端口输出 1 这种特性，称为准双向口。

对于单片机的 P0、P1、P2、P3 口作为通用 I/O 口使用时，都是准双向口。

P1 口用作输入时，由于片内场效应管 T 的截止电阻很大(数十千欧)，所以不会对输入的信号产生影响。

3. P1 口作“读—修改—写”操作

关于读锁存器问题。在图 5-1 的上部有一个“读锁存器”信号，在 CPU 执行某些指令时，需要先从 P1 口读入数据，经过某些操作后，再从 P1 口输出，这样的操作称为“读—修改—写”操作。如指令 INC　P1，其操作过程为：先把 P1 口原来的值读入(读入的是锁存器中的值，而不是引脚的值)，然后加上 1，最后再把结果从 P1 口输出。表 5-1 给出了 P0～P3 口一些“读—修改—写”指令。对于单片机的 P0、P2、P3 口，都有类似的指令。

表 5-1　Px 口的“读—修改—写”指令

助　记　符	功　　能	实　　例
INC	增 1	INC　P0
DEC	减 1	DEC　P1
ANL	逻辑与	ANL　P2,A
ORL	逻辑或	ORL　P3,A
XRL	逻辑异或	XRL　P1,A
DJNZ	减 1，结果不为 0 转	DJNZ　P2,LABEL
XCH	数据交换	XCH　A,P1
CPL	位求反	CPL　P3.0
JBC	测试位为 1 转并清 0	JBC　P0.1,LABEL

5.1.2　P2 口

P2 口是一个双功能口，一是通用 I/O 口，二是以总线方式访问外部存储器时作为高 8

位地址口。其端口某一位的结构如图 5-2 所示,对比图 5-1 可知,与 P1 口的结构类似,驱动部分基本上与 P1 口相同,但比 P1 口多了一个多路切换开关 MUX 和反相器 3。

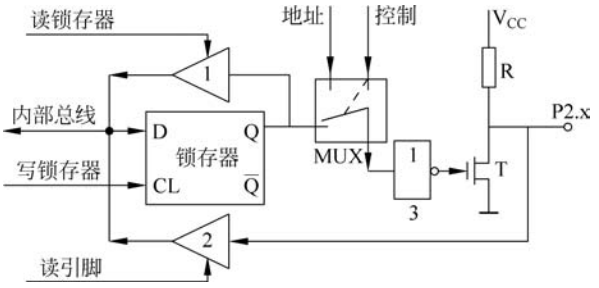

图 5-2 P2 口某一位的原理结构

1. P2 口用作通用 I/O 口

当 CPU 通过 I/O 口进行读/写操作(如执行 MOV A,P2 指令、执行 MOV P2,B 指令)时,由内部硬件自动使开关 MUX 拨向下边,与锁存器的输出端 Q 接通,这时 P2 口为通用 I/O 口,与 P1 口一样,即可随时进行输出,输入时要考虑其准双向口,先输出 1。

2. P2 口输出高 8 位地址

如果系统扩展有片外数据存储器,当进行总线读/写操作(执行 MOVX 指令)时, MUX 开关在硬件控制下拨向上边,P2 口输出高 8 位地址。对于 MOVX A,@Ri 或 MOVX @Ri,A 指令也一样,P2 口始终输出高 8 位地址。在执行 MOVX 指令时,P2 口不能作为一般 I/O 口使用。

如果使用外部程序存储器,CPU 从片外程序存储器每读一条指令,P2 口就输出一次高 8 位地址。由于 CPU 需要一直读取指令,P2 口始终要输出高 8 位地址,因此在这种情况下 P2 口不能够作为通用 I/O 口使用。

5.1.3 P3 口

P3 口是一个多功能口,其某一位的结构见图 5-3。与 P1 口的结构相比不难看出,P3 口与 P1 口的差别在于多了与非门 3 和缓冲器 4。正是这两个部分,使得 P3 口除了具有 P1 口的准双向 I/O 口的功能之外,还可以使用各引脚所具有的第二功能。与非门 3 的作用实际上是一个开关,决定是输出锁存器 Q 端数据,还是输出第二功能 W 的信号。

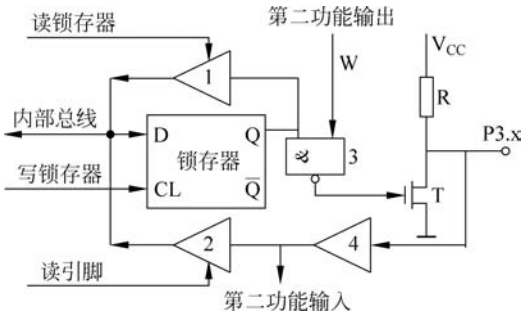

图 5-3 P3 口某一位的原理结构

1. P3 口用作通用 I/O 口

当使用 P3 口作为通用 I/O 口输出时,与非门 3 的 W 信号自动变高,为 Q 信号输出打开与非门,输出信号经过 T 从 P3 引脚输出。

当使用 P3 口作为通用 I/O 口输入时,与 P1 口一样,其准双向的特性应该先输出 1,这时与非门 3 的 W 信号也是自动为高,从 Q 端输出的高电平信号经与非门输出使 FET 截止,P3 口引脚的电位取决于外部信号,这时的读引脚操作打开缓冲器 2,引脚状态经缓冲器

4(常开)、缓冲器 2 后进入内部总线。

2. P3 口用作第二功能

当使用 P3 口的第二功能时,8 个引脚有不同的意义,各个引脚的第二功能见表 2-2。

当某位作第二功能输出时,该位的锁存器输出端被内部硬件自动置 1,使与非门 3 对第二功能的输出是打开的。由表 2-2 可知,第二功能输出可以是 TXD、$\overline{WR}$ 和 $\overline{RD}$。例如,P3.7 被选择为 $\overline{RD}$ 功能时,则该位第二功能输出的 $\overline{RD}$ 信号,通过与非门 3 和 FET 输出到 P3.7 引脚。

当某位作第二功能输入时,该位的锁存器输出端被内部硬件自动置 1,并且 W 在端口不作第二功能输出时保持为 1,则与非门 3 输出低,所以 FET 截止,该位引脚为高阻输入。P3 口的第二输入功能可以是 RXD、$\overline{INT0}$/GATE0、$\overline{INT1}$/GATE1、T0 和 T1 等,此时端口不作通用 I/O 口,因此"读引脚"信号无效,三态缓冲器 2 不导通,这样,从引脚输入的第二功能信号,经缓冲器 4 后被直接送给相关设备做处理。

5.1.4　P0 口

图 5-4 给出了 P0 口某一位的原理结构图。与 P1 口比较,多了一路总线输出(地址/数据)、总线输出控制电路(反相器 3 和与门 4)、两路输出切换开关 MUX 及开关控制 C,并且把上拉电阻换成了场效应管 T1,以增加总线的驱动能力。

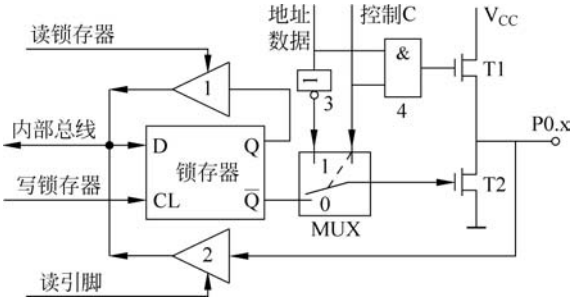

图 5-4　P0 口某一位的原理结构

当 CPU 使控制线 C=0 时,开关 MUX 拨向 $\overline{Q}$ 输出端,P0 口为通用 I/O 口;当 C=1 时,开关拨向反相器 3 的输出端,P0 口作总线使用,分时地输出地址和数据。

1. P0 口用作通用 I/O 口

如果单片机没有扩展程序存储器和数据存储器,CPU 通过 P0 口进行读/写操作(执行 MOV 指令)时,由硬件自动使控制线 C=0,封锁与门 4,使 T1 截止。开关 MUX 处于拨向 $\overline{Q}$ 输出端位置,把输出场效应管 T2 与锁存器的 $\overline{Q}$ 端接通。同时,因与门 4 输出为 0,输出级中的上拉场效应管 T1 处于截止状态,因此,输出级是漏极开路的开漏电路。这时,P0 口可以作通用 I/O 口使用,但应外接上拉电阻,才能输出高电平。

P0 口作为通用 I/O 口时,也是准双向口,在作输入之前,必须先输出 1,使输出场效应管 T2 截止,方能正确输入。

P0 口作为通用 I/O 口时,也有相应的"读—修改—写"指令,与 P1 口类似,不再赘述。

2. P0 口用作地址/数据总线

当单片机扩展有外部程序存储器或数据存储器,CPU 对片外存储器进行读/写(执行

MOVX 指令,或 $\overline{EA}=0$ 时执行 MOVC 指令)时,由内部硬件自动使控制线 C=1,开关 MUX 拨向反相器 3 的输出端。这时,P0 口为总线操作,分时地输出地址和传输数据,具体有两种情况。

1) P0 口作为总线输出地址或数据

在扩展的程序存储器或数据存储器系统中,对于 P0 口分时地输出地址和输出数据,端口的操作是一样的。MUX 开关把 CPU 内部的地址或数据经反相器 3 与驱动场效应管 T2 的栅极接通,输出 1 时,T1 导通而 T2 截止,从引脚输出高电平;输出 0 时,T1 截止而 T2 导通,从引脚输出低电平。

从图 5-4 中可以看出,上下两个 FET 处于反相状态,构成推拉式输出电路(T1 导通时上拉,T2 导通时下拉),大大提高了负载能力。所以只有 P0 口的输出可驱动 8 个 LS 型 TTL 负载。

2) P0 口作为总线输入数据

P0 口作总线操作时,控制线 C=1,总是将开关 MUX 拨向反相器 3 的输出端。这时,为了能够正确读入引脚的状态,CPU 使地址/数据自动输出 1,使 T2 截止,T1 导通。在进行总线输入操作时,"读引脚"信号有效,三态缓冲器 2 打开,引脚上的信号进入内部总线。

5.1.5 端口负载能力和接口要求

综上所述,P0 口的输出级与 P1～P3 口的输出级在结构上是不同的,因此,它们的负载能力和接口要求也各不相同。

1. P0 口

P0 口与其他端口不同,它的输出级无上拉电阻。当把它用作通用 I/O 口时,输出级是开漏电路,故用其输出去驱动 NMOS 输入时要外接上拉电阻。用作输入时,应先向端口锁存器写 1。

把 P0 口用作地址/数据总线时(系统扩展有 ROM 或 RAM),则无须外接上拉电阻。作总线输入时,不必先向端口写 1。P0 口作总线时,每一位输出可以驱动 8 个 LS 型 TTL 负载(每个 LS 型 TTL 负载,输入高电平时,其电流为 20μA,输入低电平时,其电流为 400μA)。

2. P1～P3 口

P1～P3 口的输出级接有上拉负载电阻,它们的每一位输出可驱动 4 个 LS 型 TTL 负载。作为输入口时,任何 TTL 或 NMOS 电路都能以正常的方式驱动 89C51 系列单片机(CHMOS)的 P1～P3 口。由于它们的输出级接有上拉电阻,所以也可以被集电极开路(OC 门)或漏极开路所驱动,而无须外接上拉电阻。

对于 89C51 系列单片机(CHMOS),端口当作输出口去驱动一个普通晶体管的基极(或 TTL 电路输入端)时,应在端口与基极之间串联一个电阻,以限制高电平时输出的电流。

P0～P3 口作为通用 I/O 口时,都是准双向口,作输入时,必须先向对应端口写 1。

5.2 I/O 口输出——数码管及显示控制

单片机应用系统中使用的显示器主要有发光二极管显示器(Light Emitting Diode,LED,也称数码管)和液晶显示器(Liquid Crystal Display,LCD)。本节主要讲述数码管显

示器的工作原理、接口及控制编程，主要应用单片机的端口输出功能。

5.2.1　数码管显示器结构原理

单片机中通常使用 7 段 LED 构成字型“8”，另外，还有一个小数点发光二极管，以显示数字、符号及小数点。这种显示器有共阴极和共阳极两种，如图 5-5 所示。发光二极管的阳极连在一起的（公共端 K0）称为共阳极显示器，阴极连在一起的（公共端 K0）称为共阴极显示器。一位显示器由 8 个发光二极管组成，其中，7 个发光二极管构成字型“8”的各个笔画（段）a～g，另一个小数点为 dp 发光二极管。当在某段发光二极管上施加一定的正向电压时，该段笔画即亮；不加电压则暗。为了保护各段 LED 不被损坏，须外加限流电阻。

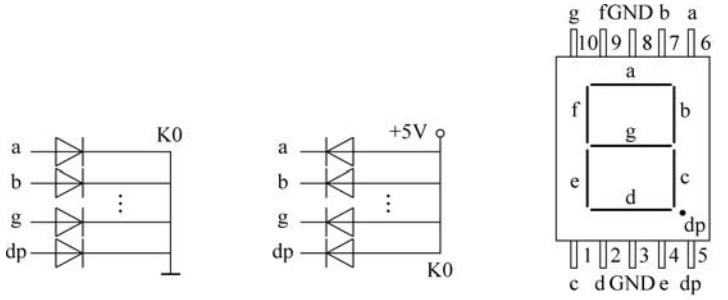

图 5-5　LED 7 段显示器

以共阴极 LED 为例，如图 5-5（a）所示，各 LED 公共阴极 K0 接地。若向各控制端 a，b，…，g，dp 顺次送入 11100001 信号，则该显示器显示“7.”字型。

除上述 7 段“8”字型显示器以外，还有 14 段“米”字型显示器和发光二极管排成 $m \times n$ 个点矩阵的显示器。其工作原理都相同，只是需要更多的 I/O 口线控制。

共阴极与共阳极 7 段 LED 显示数字 0～F、“－”符号及“熄灭”的编码（a 段为最低位，dp 点为最高位）如表 5-2 所示。

表 5-2　共阴极和共阳极 7 段数码管显示字型编码表

显示字符	0	1	2	3	4	5	6	7	8
共阴极段码	3F	06	5B	4F	66	6D	7D	07	7F
共阳极段码	C0	F9	A4	B0	99	92	82	F8	80
显示字符	9	A	B	C	D	E	F	－	熄灭
共阴极段码	6F	77	7C	39	5E	79	71	40	00
共阳极段码	90	88	83	C6	A1	86	8E	BF	FF

注：以上为 8 段，8 段最高位为小数点段。表中为小数点不点亮段码。

5.2.2　数码管显示方式

数码管显示器有静态显示和动态显示两种方式。

1. 数码管静态显示方式

静态显示就是当显示器显示某个字符时，相应的段（发光二极管）恒定地导通或截止，直

到显示另一个字符为止。例如,7 段显示器的 a、b、c 段恒定导通,其余段和小数点恒定截止时显示 7;当显示字符 8 时,显示器的 a、b、c、d、e、f、g 段恒定导通,dp 截止。

数码管显示器工作于静态显示方式时,各位的共阴极(公共端 K0)接地;若为共阳极(公共端 K0),则接+5V 电源。每位的段选线(a～dp)分别与一个 8 位锁存器的输出口相连,显示器中的各位相互独立,而且各位的显示字符一经确定,相应锁存的输出将维持不变。正因如此,静态显示器的亮度较高。这种显示方式编程容易,管理也较简单,但占用 I/O 口线资源较多。因此,在显示位数较多的情况下,一般都采用动态显示方式。

2. 数码管动态显示方式

在多位数码管显示时,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(或共阳)极公共端 K 分别由相应的 I/O 线控制,实现各位的分时选通。如图 5-6 所示为 6 位共阴极数码管动态显示接口电路。

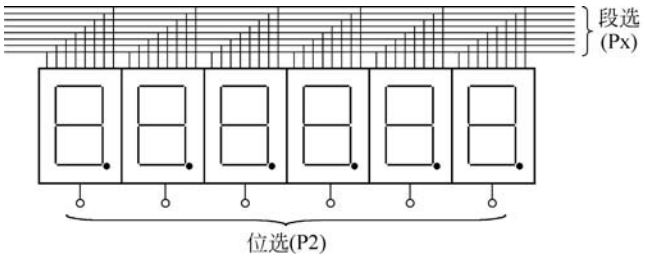

图 5-6　6 位数码管动态显示接口电路

由于 6 位数码管所有段选线皆由 P1 口(或其他口 Px)控制,因此,在每一瞬间,6 位数码管会显示相同的字符。要想每位显示不同的字符,就必须采用扫描方法轮流点亮各位数码管,即在每一瞬间只使某一位显示字符。在此瞬间,P1 口输出相应字符段选码(字形码),而 P2 口在该显示位送入选通电平(因为数码管为共阴,故应送低电平),以保证该位显示相应的字符。如此轮流,使每位分时显示各自应显示的字符。多位数码管的这种显示方式称为动态扫描显示。

段选码、位选码每送入一次后要有一定的延时,使其各段稳定点亮一段时间,延时应不少于 1ms。由于人眼有视觉暂留效应,时间约为 0.1s(100ms),所以只要每位在 0.1s 之内再次扫描点亮,每位显示的内容在人眼中就不会消失,为了确保显示稳定不闪烁,每秒钟每位扫描显示的次数应不少于 20 次(如早期的电视机帧频为 25 帧/秒)。

5.2.3　数码管显示控制

图 5-7 所示为 89C52 P0 口和 P2 口控制的 6 位共阴极数码管动态显示电路。图中,P0 口输出段选码,P2 口输出位选码,位选码占用输出口的引脚数决定于显示器的位数。图中 P0 口的上拉电阻(排电阻 RESPACK-8),使 P0 口能够输出高电位和有一定的驱动能力。

在 Proteus 下做仿真显示可以不用驱动,但是实际使用中,段信号和位信号都需要加驱动,如使用 74LS245,它是一种双向 8 位缓冲驱动器,应加在图 5-7 中虚线框的位置。

在实际应用中,数码管的亮度与排电阻的阻值有关,电阻小,驱动能力强,数码管亮度大,否则数码管亮度小,对上面的仿真电路反映不出来。排电阻参数的修改方法:打开 Edit Component 设置页面,对 Model Type 选择 ANALOG 项,然后在 Part Value 输入电阻值

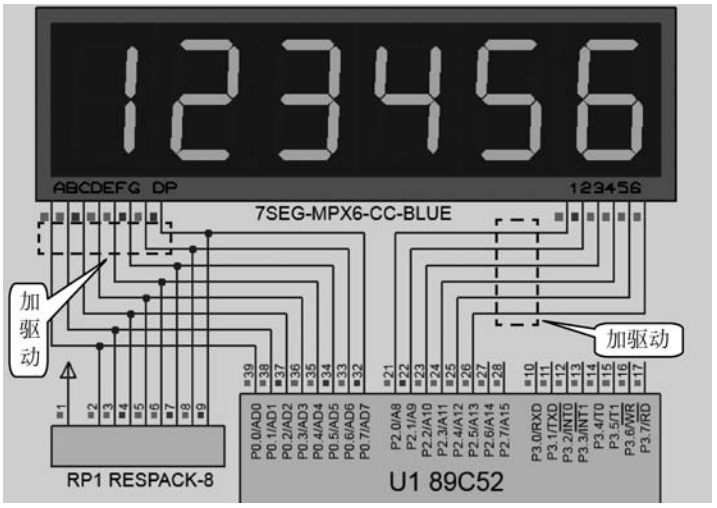

图 5-7　6 位数码管动态显示电路

即可。

C 语言程序如下：

```
#include<reg52.h>
unsigned char code LED[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,
                                                    //共阴数码管显示段码
                            0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x40,0x00};
                                                    //0～9、A～F、-、全灭段码
unsigned char data disBuf[6] = {1,2,3,4,5,6};
                                                    //定义字形码和显示缓冲区
void display()                                      //数码管逐位扫描显示函数
{
    unsigned char i, scan = 0xfe;
    for(i = 0; i<6; i++)                //逐位扫描显示
    {   P2 = 0xff;                      //各位关闭显示
        P1 = LED[disBuf[i]];            //段码送 P1 口
        P2 = scan;                      //位码送 P2 口
        scan = (scan<<1) + 1;           //位码右移 1 位
        delayms(5);                     //延时 5ms,函数定义见例 2-2
}    }
void main()                             //主函数,主要功能就是调用 display()保持数码管显示
{
    while(1)
    {
        display();                      //调用数码管显示函数
        delayms(5);                     //延时 5ms,代表有其他函数时的执行时间
}    }
```

汇编语言程序如下。

```
        DISBUF   EQU   30H                    ; 定义缓冲区(30H～35H)首地址
        ORG      0000H
        LJMP     MAIN
        ORG      0030H
MAIN:                                         ; 主程序
        MOV   SP, #0DFH                       ; 设置堆栈指针,将其放在片内 RAM 高端
        MOV   30H,#1                          ; 给显示缓冲区写显示数据 1～6
        MOV   31H,#2
        MOV   32H,#3
        MOV   33H,#4
        MOV   34H,#5
        MOV   35H,#6
MAINLP:                                       ; 循环体
        LCALL   DISP                          ; 调用数码管扫描显示子程序
        MOV     R7,#5                         ; 准备调用延时子程序入口参数,延时 5ms
        LCALL   DELAYMS                       ; 调用延时子程序,子程序定义见例 3-23
        SJMP    MAINLP                        ; 跳转到 MAINLP 处,作循环
DISP:                                         ; 数码管扫描显示子程序
        MOV   R0,#DISBUF                      ; 显示缓冲区首地址送 R0
        MOV   R2,#0FEH                        ; 位码 1111 1110B 送 R2
        MOV   R3,#6                           ; 6 位显示
        MOV   DPTR,#TAB                       ; DPTR 指向段码表,先点亮最左边 LED
DISLP:
        MOV   P2,#0FFH                        ; 各位关闭显示
        MOV   A,@R0                           ; 取显示数据
        MOV   CA,@A+DPTR                      ; 取出字形码
        MOV   P1,A                            ; 送出显示
        MOV   P2,R2                           ; 位码送 P2 口
        MOV   R7,#5                           ; 准备调用延时子程序入口参数,延时 5ms
        LCALL   DELAYMS                       ; 调用延时子程序,子程序定义见例 3-23
        INC     R0                            ; 数据缓冲区地址加 1
        MOV     A,R2
        RL      A                             ; 位码循环左移一位
        MOV     R2,A
        DJNZ     R3,DISLP                     ; 扫描位数减 1,不为 0 则循环
        RET                                   ; 从数码管扫描显示子程序返回

TAB:                                          ; 共阴数码管显示段码
        DB     3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH    ; 显示 0～8 段码
        DB     6FH,77H,7CH,39H,5EH,79H,71H,40H,00H    ; 显示 9、A～F、-、全灭段码
```

5.3 I/O 口输入——键盘及按键识别

单片机应用系统通常都需要进行人-机对话。这包括人对应用系统的状态干预与数据输入等,所以应用系统大多数都有键盘。本节主要讨论键盘结构、特征及识别,主要应用单片机端口的输入功能。

5.3.1　键盘分类及按键识别

键盘是一组按键的集合，它是最常用的单片机输入设备。操作人员可以通过键盘输入数据或命令，实现简单的人-机通信。按键是一种常开型按钮开关。平时（常态时），按键的两个触点处于断开状态，按下键时它们才闭合（短路）。键盘分编码键盘和非编码键盘。键盘上闭合键的识别由专用的硬件译码器实现，并产生键编号或键值的称为编码键盘，如BCD码键盘、ASCII码键盘等；靠应用程序识别的称为非编码键盘。

在单片机组成的测控系统及智能化仪器中，用得最多的是非编码键盘。本节讨论非编码键盘的原理、接口技术和程序设计。

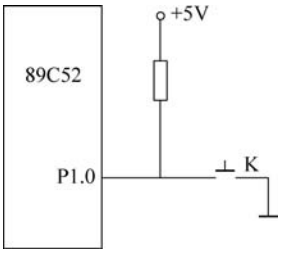

图 5-8　按键电路

键盘中每个按键都是一个常开开关电路，如图5-8所示。

当按键K未被按下时，P1.0输入为高电平；当K闭合时，P1.0输入为低电平。通常按键所用的开关为机械弹性开关，当机械触点断开、闭合时，电压信号波形如图5-9所示。由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会一下子断开。因而在闭合及断开的瞬间均伴随有一连串的抖动，如图5-9所示。抖动时间的长短由按键的机械特性决定，一般为5～10ms。这是一个很重要的时间参数，在很多场合都要用到。

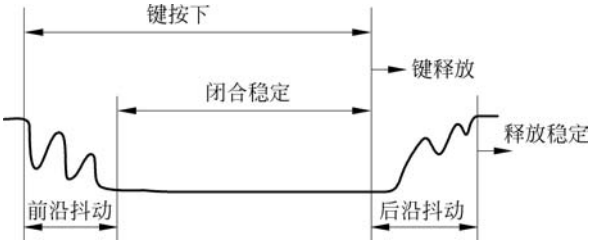

图 5-9　按键时的抖动

按键稳定闭合时间的长短则是由操作人员的按键动作决定的，一般为零点几秒。

键抖动会引起一次按键被误读多次。为了确保CPU对键的一次闭合仅做一次处理，必须去除键抖动。在键闭合稳定时，读取键的状态，并且必须判别；在键释放稳定后，再作处理。按键的抖动，可用硬件或软件两种方法消除。

如果按键较多，常用软件方法去抖动，即检测出键闭合后执行一个延时程序，产生12～20ms的延时，让前沿抖动消失后，再一次检测键的状态，如果仍保持闭合状态电平，则确认为真正有键按下。当确认有键按下或检测到按键释放后，才能转入该键的处理程序。

5.3.2　独立式键盘及按键识别

键盘结构可以根据按键数目的多少分为独立式和行列式（矩阵式）两类，独立式键盘适用于按键数目较少的场合，结构和处理程序比较简单。独立式按键是指各按键相互独立地接通一条输入数据线，如图5-10所示。这是最简单的键盘结构，该电路为查询方式电路。

当任何一个键按下时，与之相连的输入数据线即可读入数据0，即低电平，而没有按下

时读入1,即高电平。要判别是否有键按下,用单片机的位处理指令十分方便。

这种键盘结构的优点是电路简单;缺点是当键数较多时,要占用较多的I/O线。

图5-10所示查询方式键盘的处理程序比较简单。实际应用中,P1口内有上拉电阻,图中电阻可以省去。

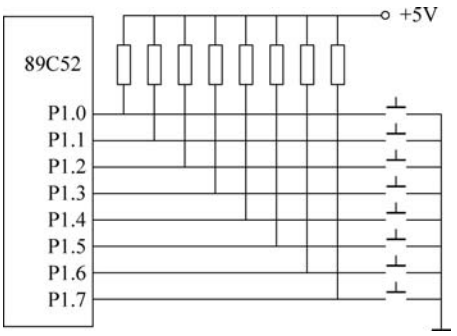

图5-10 独立式键盘

【例5-1】 设计一个独立式按键的键盘接口,并编写键扫描程序,电路原理图如图5-10所示,键号从上到下分别为0～7。

C语言程序如下:

```
#include<reg52.h>
void key()                    //键盘识别函数
{
    unsigned char k;

    P1 = 0xff;                //输入时 P1 口置全 1
    k = P1;                   //读取按键状态
    if(k == 0xff)             //无键按下,返回
      return;
    delayms(20);              //有键按下,延时 20ms 去抖动,函数定义见例 2-2
    k = P1;
    if(k == 0xff)             //确认键按下,抖动引起,返回
        return;
    while(P1!= 0xff);         //等待键释放
    switch(k)
    {   case: 0xfe
        …                     //0 号键按下时执行程序段
        break;
        case: 0xfd
        …                     //1 号键按下时执行程序段
        break;
        …                     //2～6 号键程序省略,读者可自行添上
        case: 0x7f
        …                     //7 号键按下时执行程序段
        break;
}    }
```

汇编语言程序如下:

```
KEY:
    MOV     P1,#0FFH          ; P1 口为输入口
    MOV     A,P1              ; 读取按键状态
```

```
        CPL     A                   ; 取正逻辑,高电平表示有键按下
        JZ      EKEY                ; A = 0 表示无键按下,返回
        MOV     R7,#20              ; 准备调用延时函数的入口参数
        LCALL   DELAYMS             ; 延时 20ms 去抖,函数定义见例 3-23
        MOV     A,P1
        CPL     A
        JZ      EKEY                ; 抖动引起,返回
        MOV     B,A                 ; 存键值
KEY1:
        MOV     A,P1                ; 以下等待键释放
        CPL     A
        JNZ     KEY1                ; 未释放,等待
        MOV     A,B                 ; 取键值送 A
        JB      ACC.0,PKEY0         ; K0 按下转 PKEY0
        JB      ACC.1,PKEY1         ; K1 按下转 PKEY1
        …
        JB      ACC.7,PKEY7         ; K7 按下转 PKEY7
EKEY: RET
PKEY 1:
        LCALL   K0                  ; K0 命令处理程序
        RET
PKEY 2:
        LCALL   K1                  ; K1 命令处理程序
        RET
        …
PKEY 4:
        LCALL   K7                  ; K7 命令处理程序
        RET
```

由程序可以看出,各按键由软件设置了优先级,优先级顺序依次为 0～7。

5.3.3　行列式键盘及按键识别

行列式键盘适用于按键数目较多的场合,其结构排列成行列矩阵形式,如图 5-11 所示。按键的识别有行扫描法、行列对称查找法、行列反转法等方法。

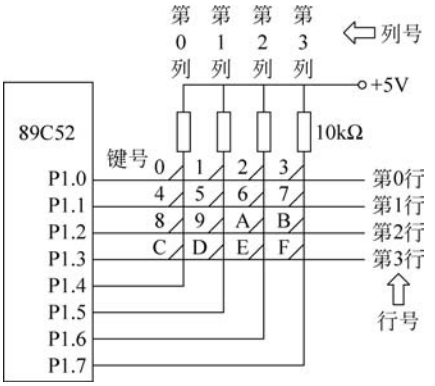

图 5-11　4×4 矩阵键盘接口

行列式键盘的水平线(行线)与垂直线(列线)的交叉处各通过一个按键来连通。利用这种行列矩阵结构只需 M 条行线和 N 条列线,即可组成具有 $M\times N$ 个按键的键盘。

在这种行列矩阵式非编码键盘的单片机系统中,键盘处理程序首先判断是否有键按下,当确认有键按下后,再进一步就要识别是哪一个按键被按下。下面介绍两种常用的按键识别方法:逐行扫描法和行列对称查找法。

1. 行扫描法识别按键

1) 行扫描法原理

行扫描法按键识别的工作过程如下:

(1) 判断键盘中是否有键按下。

(2) 若有则延时去抖动。

(3) 逐行扫描,确定按下键所在的行。

(4) 对按下键的行逐列查找,确定按下键所在的列。

(5) 根据按下键的行、列值,确定按下键的键号(键的顺序编号)0,1,2,…,F。

关于判断键盘中是否有键按下的方法为(以图 5-11 所示的 4×4 键盘为例):由单片机 I/O 口向键盘行线输出扫描字 0000B,把全部行线置为低电平,然后读入列线的电平状态。如果无键按下,则读到列线的值全为高电平,即列线读入值全为 1;如果有按键按下,总会有一根列线被拉至低电平,即列线读入值不全为 1。

当确定有键按下后,需要找到所按下按键的行、列位置,行扫描法的原理为:依次向行线送低电平(如 1110B、1101B、1011B、0111B,第 0 行对应最右端位,第 3 行对应最左端位,从第 0 行向第 3 行依次扫描),然后读取列线状态。如果全为 1,则所按下的键不在此行,接着扫描下一行;如果不全为 1,则所按下的键必在此行,行扫描结束。

找到按键所在行之后,根据上面读取到的列线值,哪一列为 0,则所按下的键必在该列。

找到所按下按键的行、列位置后,对按键进行编码,即求得按键的键号。按图 5-11 所示的行列编号及键号,其键号与行、列号的关系为:

$$键号=列数\times行号+列号 \tag{5-1}$$

需要指出的是,按键的键号并不等于按键实际定义的功能,因此还要进行转换,可以借助查表或其他方法完成。如根据键号,在程序中执行相应的功能子程序,完成按键所定义的功能。

2) 行扫描法键盘识别程序

C 语言程序如下:

```
#include<reg52.h>
unsigned char key()                    //键盘识别函数,有键按下返回键号 0～15,否则返回 0xff
{   unsigned char row,col = 0,scan,k = 0xff;     //定义行、列、行列扫描码、键号变量

    P1 = 0xf0;                         //各行全输出 0
    if(P1 == 0xf0)                     //从 P1 口输入,判断输入值是否为 0xf0
        return k;                      //无键按下,返回
    delayms(15);                       //延时 15ms 去抖动,函数定义见例 2-2
    if(P1 == 0xf0)                     //再从 P1 口输入,判断输入值是否为 0xf0
        return k;                      //抖动引起,返回
    scan = 0xfe;                       //准备行扫描码,从第 0 行开始
    for(row = 0; row<4; row++)         //逐行扫描
    {   P1 = scan;                     //扫描值送 P1,某一行输出 0
        k = P1&0xf0;                   //读 P1 口的值,低 4 位清 0、保留高 4 位的列值
        if(k!= 0xf0)                   //如果各列值不全为 1,所按下键在该行
        {   scan = 0x10;               //准备列扫描码,从第 0 列开始
            for(col = 0; col<4; col++) //查找按下键所在列,逐列扫描
            {   if((k&scan) == 0)      //如果当前列为 0,则已找到按下键的列号
```

```
                    break;          //跳出列扫描循环
                scan<< = 1;         ///列扫描码左移 1 位,查找下一列
            }
            break;                  //跳出行扫描循环
        }
        scan = (scan<<1) + 1;       //行扫描码左移 1 位,查找下一行
    }
    k = 4 * row + col;              //计算按下键的键号,等于行号 * 4 + 列号
    P1 = 0xf0;
    while(P1!= 0xf0);               //等待键释放
    return  k;                      //返回键号
}
```

汇编语言程序如下(返回值：在累加器 A 中,为键号)：

```
KEY:                                ; 有键按下返回键号 0～15,无键按下返回 0FFH
    LCALL  KEYPRESS                 ; 调用查询是否有键按下子程序
    JZ     KEYEXIT                  ; 无键按下,返回
    MOV    R7,＃15                  ; 准备调用延时函数的入口参数
    LCALL  DELAYMS                  ; 延时 15ms 去抖动.函数定义见例 3-23
    LCALL  KEYPRESS
    JZ     KEYEXIT                  ; 抖动引起,返回
KEYSTART:
    MOV    R0,＃0                   ; R0 作为行扫描计数器,开始为 0
    MOV    R3,＃0FEH                ; R3 低 4 位为扫描字,高 4 位输入,为全 1
ROWSCAN:
    MOV    P1,R3                    ; 输出行扫描字
    MOV    A,P1                     ; 读列输入值
    MOV    R1,A                     ; 暂存列输入值
    CPL    A
    ANL    A,＃0F0H
    JNZ    COL                      ; 键在该行,转列处理 COL
    MOV    A,R3
    RL     A
    MOV    R3,A                     ; 进行下一行扫描
    INC    R0
    CJNE   R0,＃4,ROWSCAN           ; 4 次扫描未完成,转 ROWSCAN 否则返回
KEYEXIT:
    MOV    A,＃0FFH                 ; 无键按下时返回值 0FFH
    RET
COL:                                ; 列处理
    MOV    R2,＃0                   ; R2 作为列计数器,开始为 0
    MOV    R3,＃10H                 ; R3 为列扫描字暂存,高 4 位为扫描字
COLSCAN:
    MOV    A,R1                     ; 取列输入值
    ANL    A,R3
    JZ     DCODEKEY                 ; A = 0 则键在该列,转按键编码
    MOV    A,R3
    RL     A
```

```
    MOV     R3,A                    ; 进行下一行扫描
    INC     R2
    CJNE    R2,#4,COLSCAN           ; 4 次扫描未完成,转 COLSCAN 否则返回
    SJMP    KEYEXIT
DCODEKEY:
    MOV     A,R0                    ; 计算键号,行号送 A
    RL      A
    RL      A
    ADD     A,R2                    ; 行号×4+列号=键号,在 A 中
    PUSH    ACC                     ; 键号进栈保存
WAITKEYUP:
    LCALL   KEYPRESS                ; 等待键释放
    JNZ     WAITKEYUP
    POP     ACC                     ; 键号出栈
    RET
KEYPRESS:
    MOV     P1,#0F0H                ; 是否有键按下,有返回非 0,无返回 0
    MOV     A,P1
    XRL     A,#0F0H
    RET
```

2. 行列对称查找法识别按键

1) 行列对称查找法原理

行列对称查找法接口电路和扫描法的接口电路一样。需要注意的是,图 5-11 所使用的 I/O 口为 P1 口,内部有上拉电阻,因此列线的上拉电阻可以不用,如果使用 P0 口,则需要在行线和列线上都加上拉电阻。

行列对称查找法按键识别过程如下:

(1) 判断是否有键按下、延时去抖动、再判断是否有键按下(与行扫描法一样)。

(2) 从键盘端口的各个行线全部输出 0 而各列全输出 1,然后从端口输入,得到列线值,其值只有对应按下键的列为 0,其他列都是 1,据此通过移位找到为 0 的列号。

(3) 从键盘端口的各个列线全部输出 0 而各行全输出 1,然后从端口输入,得到行线值,其值只有对应按下键的行为 0,其他行都是 1,据此通过移位找到为 0 的行号。

(4) 计算按下键的键号,键号=列数×行号+列号。

2) 行列对称查找法识别按键程序

C 语言程序如下:

```
#include<reg52.h>
unsigned char key()                     //行列对称查找法键盘识别函数,有键按下返回 0～15
{                                       //有键按下返回键号 0～15,无键按下返回 0xff
    unsigned char row=0xff,col=4,k=0xff;    //定义行、列、返回值变量

    P1=0xf0;                            //行输出 0,列输出 1
    if(P1==0xf0)                        //读入的列值全为 1 则无键按下
        return k;                       //返回 0xff
    delayms(15);                        //延时 15ms 去抖动,函数定义见例 2-2
    if(P1==0xf0)                        //读入的列值全为 1 则无键按下
        return k;                       //抖动引起,返回 0xff
```

```
    k = P1;                          //读取列值,移到低 4 位
    do                               //查找列值中为 0 的列号
    {   col -- ;                     //查找的列数加 1
        k <<= 1;                     //列值左移 1 位,移出位送到进位标志位 CY
    }while(CY);                      //CY 为 1,未找到则循环; CY 为 0 则找到,退出
    P1 = 0xff;P1 = 0x0f;             //列输出 0,行输出 1
    k = P1;                          //读取行值(在低 4 位,高 4 位为 0)
    do                               //查找行值中为 0 的行号
    {   row++;                       //查找的行数加 1
        k >>= 1;                     //行值右移 1 位,移出位送到进位标志位 CY
    }while(CY);                      //CY 为 1,未找到则循环; CY 为 0 则找到,退出
    k = row * 4 + col;               //计算键号
    while(P1!= 0x0f);                //查询键盘,等待按键释放
    return k;                        //返回按键号 0～15
}
```

汇编语言程序如下(返回值：在累加器 A 中,为键号)：

```
KEY:                                ; 有键按下返回键号 0～15,无键按下返回 0FFH
    LCALL    KEYPRESS               ; 调用查询是否有键按下子程序
    JZ       KEYEXIT                ; 无键按下,返回
    MOV      R7,＃15                ; 准备调用延时子程序的入口参数
    LCALL    DELAYMS                ; 延时 15ms 去抖动. 子程序见例 3-23
    LCALL    KEYPRESS
    JNZ      KEYSTART               ; 有键按下转去处理
KEYEXIT:
    MOV      A,＃0FFH               ; 无键按下返回 0FFH
    RET
KEYSTART:
    MOV      A,P1                   ; 读取列值,在高 4 位,只有按下键对应列为 0
    SWAP     A                      ; 列值交换到低 4 位
    MOV      R1,＃0FFH              ; R1 放查找的列号
KEYCOLLP:                           ; 查找为 0 的列号
    INC      R1                     ; 列号加 1
    RRC      A                      ; 列值带进位右移 1 位,最低位移到进位位 CY
    JC       KEYCOLLP               ; 进位位 CY 为 1 则未找到,继续查找
    MOV      P1,＃0FH               ; P1 口的各列输出 0
    MOV      A,P1                   ; 读取行值,在低 4 位,只有按下键对应行为 0
    MOV      R2,＃0FFH              ; R2 放查找的行号
KEYROWLP:                           ; 查找为 0 的行号
    INC      R2                     ; 行号加 1
    RRC      A                      ; 行值带进位右移 1 位,最低位移到进位位 CY
    JC       KEYROWLP               ; 进位位 CY 为 1 则未找到,继续查找
    MOV      A,R2                   ; R2 中的行号给 A
    RL       A                      ; 行号左移 1 位,相当于行号乘以 2
    RL       A                      ; 行号再乘以 2
    ADD      A,R1                   ; 再加上列号,A 中是键号
    PUSH     ACC                    ; A 中值进栈保存
WAITKEYUP:                          ; 查询按键是否释放
    LCALL    KEYPRESS               ; 等待按键释放
    JNZ      WAITKEYUP              ; 按键未释放,继续查询
```

```
    POP     ACC                 ; 保存的键号出栈
    RET                         ; 键盘扫描子程序返回
KEYPRESS:                       ; 快速扫描,判断是否有键按下
    MOV     P1,#0F0H            ; 是否有键按下,有返回非0,无返回0
    MOV     A,P1                ; 读取列值
    XRL     A,#0F0H             ; 读取值与输出值异或
    RET
```

5.3.4 中断方式扫描键盘

为了提高CPU的效率,可以采用中断扫描工作方式,即只有在键盘有键按下时才产生中断申请,CPU响应中断,进入中断服务程序进行键盘扫描,并做相应处理。也可以采用定时扫描方式,即系统每隔一定时间进行键盘扫描,并做相应处理。

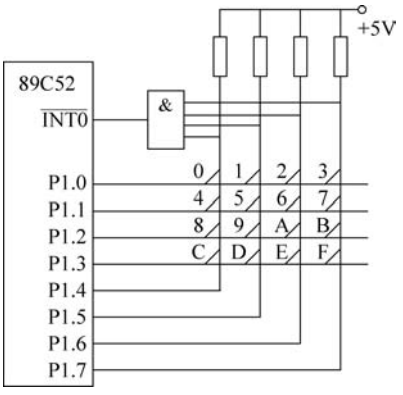

图5-12 中断方式键盘接口

中断扫描工作方式的键盘接口如图5-12所示。该键盘直接由89C52 P1口的高、低4位构成4×4行列式键盘。键盘的行线与P1口的低4位相接,键盘的列线接到P1口的高4位。因此,P1.0～P1.3作行输出线,P1.4～P1.7作列输入线。对P1.0～P1.3各行输出0,当有键按下时,$\overline{\text{INT0}}$端为低电平,向CPU发出中断请求,在中断服务程序调用上面的按键识别程序,得到按下键的键号。

关于中断和中断服务程序,见第6章。

5.3.5 键盘应用举例

本例借用电话机键盘来介绍按键的处理。电话机键盘共有12个按键,0～9为数字键,*和#键为功能键。本例要求实现功能:数字键按下时得到相应的数字值,像计算器按键按下一样显示在数码管上,*键按下将数码管上显示的数字值退格处理,#键按下时将数码管上显示的数字值加倍显示。Proteus模拟的硬件电路如图5-13所示,数码管使用共阳极的7SEG-MPX8-CA-BLUE,键盘使用KEYPAD-PHONE。

数码管的选择分析:对于选用的共阳极数码管,当P0口的某1位P0.x口输出0时,P0.x引脚为低电平,外部电流流入使对应段点亮;当P0.x输出1时,P0x引脚的电位不确定,但端口的驱动三极管是截止的,外部电流无法流入,对应段不亮。可见,使用共阳极数码管P0口不需要接上拉电阻,但使用共阴极数码管需要接上拉电阻。需要说明的是,在实际应用中,数码管的段和位都需要加驱动,P0口就需要接上拉电阻了。

C语言程序如下:

```
#include<reg52.h>
#define BACK  10                    //定义功能键键号
#define DOUBLE  11
unsigned char code LED[] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0xff};
                                    //共阳极段码表
```

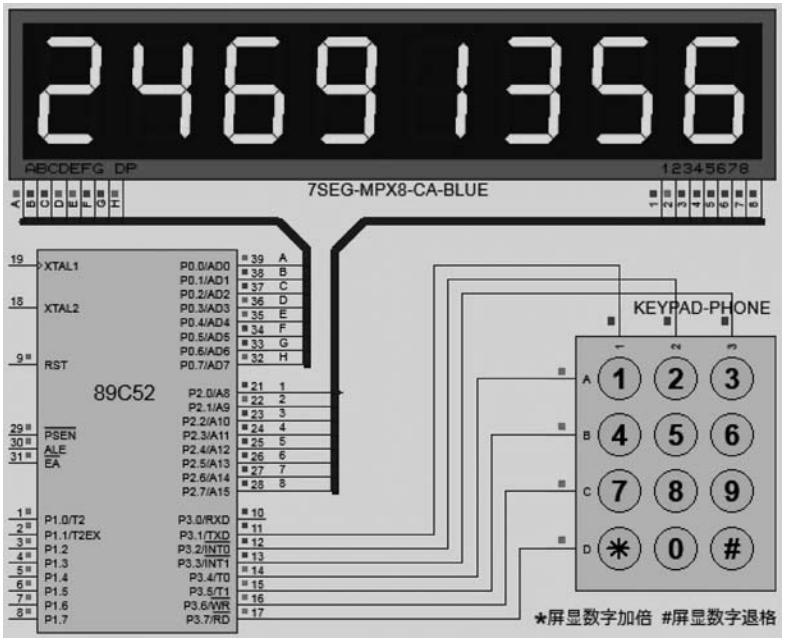

图 5-13　键盘应用举例硬件电路

```
unsigned char disBuf[8];
unsigned char code keyTable[ ] = {1,2,3,4,5,6,7,8,9,DOUBLE,0,BACK 0xff};
                                        //键号对应按键功能表
void writeBuf(unsigned long disData) //待显示数据写入显示缓冲区,disData<= 99999999
{    unsigned char i = 7;
     do
     {    disBuf[ i -- ] = disData % 10;
          disData/ = 10;
     }while(disData);
     while(i<7)
          disBuf[ i -- ] = 10;           //高位 0 存入不显示字符
}
void display()                          //数码管扫描显示函数
{    unsigned char i;
     for(i = 0; i<8; i++)
     {    P2 = 0;                       //关闭显示,避免显示出现乱码
          P0 = LED[disBuf[i]];
          P2 = 1<<i;
          delayms(4);                   //延时 4ms,函数定义见例 2-2
}     }
unsigned char key()                     //键盘识别函数,有键按下返回键功能码,无键按下返回 0xff
{    unsigned char row = 4,col = 0xff,k = 0xff;    //定义行、列、键号变量

     P3 = 0xf0;                         //列送 0,读行
```

```
    if(P3 == 0xf0)
        return k;                    //无键按下,返回 0xff
    delayms(15);                     //延时 15ms 去抖动,函数定义见例 2-2
    if(P3 == 0xf0)
        return k;                    //抖动引起,返回 0xff
    k = P3;                          //读取行值,在高 4 位,低 4 位为 0
    do                               //查找行值中为 0 的行号
    {   row--;                       //查找的行数加 1
        k <<= 1;                     //行值左移 1 位,移出位送到进位标志位 CY
    }while(CY);                      //CY 为 1,未找到则循环; CY 为 0 则找到,退出
    P3 = 0xff;P3 = 0x0f;             //行送 0,读列
    k = P3; k >>= 1;                 /读取列值
    do                               //查找列值中为 0 的列号
    {   col++;                       //查找的列数加 1
        k >>= 1;                     //列值右移 1 位,移出位送到进位标志位 CY
    }while(CY);                      //CY 为 1,未找到则循环; CY 为 0 则找到,退出
    k = row * 3 + col;               //计算键号
    P3 = 0x0f;
    while(P3!= 0x0f)
        display();                   //等待按键释放
    return keyTable[k];              //返回实际按键功能码
}
void main()                          //主函数,循环调用显示函数、键盘函数,对按键做处理
{   unsigned char k;
    unsigned long value;
    bit numFlag = 0;                 //数字输入标志

    writeBuf(0);                     //显示缓冲区填 0
    while(1)
    {   display();
        k = key();
        if(k!= 0xff)                 //有键按下
        {   if(k<10)                 //数字键处理
            {   if(numFlag == 0)
                {   numFlag = 1;     //输入第 1 位数
                    value = 0;
                }
                value = value * 10 + k;
                if(value<= 99999999)
                    writeBuf(value);
                else value/ = 10;    //超过 8 位数则不接受输入
            }
            else if(k == BACK)       //退格键处理
```

```
            {    value/ = 10;
                 writeBuf(value);
            }
            else if(k == DOUBLE)     //加倍键处理
            {    value * = 2;
                 if(value<= 99999999)
                     writeBuf(value);
                 else value/ = 2;     //乘以 2 超过 8 位数则不再乘以 2
                 numFlag = 0;
}   }   }   }
```

5.4　液晶显示器及控制

液晶显示器(LCD)具有功耗低、体积小、重量轻、超薄等许多其他显示器无法比拟的优点,近几年来被广泛用于单片机控制的智能仪器、仪表和低功耗电子产品中。LCD 可分为段位式 LCD、字符式 LCD 和点阵式 LCD。

不同型号的 LCD 接口方式区别较大,这里以 Proteus 仿真软件中 LM016L 为例,讲述字符式 LCD 的显示原理及接口。

LM016L 显示的内容为 16×2,即可以显示两行,每行 16 个字符,和字符型 LCD1602 完全一样。目前市面上字符液晶屏大多采用 HD44780 控制器,因此,基于 HD44780 写的控制程序可以很方便地应用于市面上大部分的字符型液晶显示器。本节不仅用到单片机的整个 8 位端口输入、输出,还会用到按位输入、输出。

5.4.1　LM016L 引脚信号

LM016L 的引脚信号见图 5-14,各引脚功能如下。

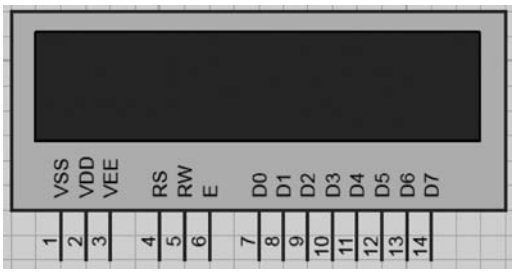

图 5-14　LM016L 的引脚图

VSS：电源地接入端。

VDD：5V 电源正极接入端。

VEE：对比度调整电压接入端。通过一个接 5V 电源和地的 10kΩ 的电位器调节。

RS：指令、数据寄存器选择信号,输入。1 表示选择数据寄存器,D0～D7 输入的应该为数据(显示字符的代码);0 表示选择指令寄存器,D0～D7 输入的应为指令。

R/W：读写控制信号,输入。1 表示从 LCD 读取状态信息(包括光标地址);0 表示向

LCD 写入指令(包括显示地址)或显示的数据。

E：使能信号,输入。读操作时高电平有效,写操作时下降沿有效。

D0～D7：双向 8 位数据线。

A：背光电源正极接入端。

K：背光电源负极接入端。

5.4.2 LM016L 操作指令

LM016L 液晶模块内部的控制器共有 11 条控制指令,如表 5-3 所示。

表 5-3 LM016L 控制命令表

指令编号	指令功能	操作信号		指令代码或参数							
		RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏,光标回到 00H 位置	0	0	0	0	0	0	0	0	0	1
2	光标复位,回到 00H 位置	0	0	0	0	0	0	0	0	1	*
3	光标和显示模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	设置总线宽、显示行、字点阵	0	0	0	0	1	DL	N	F	*	*
7	设置字符发生存储器地址	0	0	0	1	字符发生存储器 CGRAM 地址					
8	设置数据存储器地址	0	0	1	显示数据存储器 DDRAM 地址						
9	读忙标志和地址	0	1	BF	计数器地址						
10	写数据数到 CGRAM 或 DDRAM	1	0	要写的数据内容							
11	从 CGRAM 或 DDRAM 读数据	1	1	读出的数据内容							

LM016L 液晶模块的读写、屏幕和光标的操作,都是通过指令编程来实现的。相关指令说明如下。

指令 3：光标和显示模式设置。I/D：光标移动方向,置 1 右移,清 0 左移。S：屏幕上所有文字左右移动控制,置 1 可移动,清 0 不可移动。

指令 4：显示开关控制。D：控制整体显示的开与关,置 1 开显示,清 0 关显示。C：控制光标的开与关,置 1 显示出光标,清 0 无光标。B：控制光标是否闪烁,置 1 光标闪烁,清 0 光标不闪烁。

指令 5：光标或字符移位。S/C：置 1 表示移动字符,清 0 表示移动光标。

指令 6：功能设置命令。DL：置 1 为 8 位总线,清 0 为 4 位总线。N：置 1 为双行显示,清 0 为单行显示。F：置 1 显示 5×10 的点阵字符,清 0 显示 5×7 的点阵字符。

指令 9：读忙信号和光标地址。BF：为忙标志位,为 1 表示忙,此时模块不能接收命令或者数据,为 0 表示不忙。低 7 位为读出的光标地址。

5.4.3 LM016L 存储器

HD44780 内置了 DDRAM、CGROM 和 CGRAM。

1) 显示数据存储器(DDRAM)

DDRAM 就是显示数据 RAM,用来寄存待显示的字符代码。见指令 8,使用 7 位表示,

共80H个地址，LM016L共有两行，每行16个字符，所以只使用了32个地址，其地址和屏幕的对应关系如表5-4所示。

表5-4　DDRAM地址和屏幕行列的对应关系

	显示位置	1	2	3	4	…	15	16
DDRAM地址	第一行	00H	01H	02H	03H	…	0EH	0FH
	第二行	40H	41H	42H	43H	…	4EH	4FH

在指令8中，D7位为1，所以要想在DDRAM的00H地址处显示数据（即第一行第一列），则必须将00H加上80H，即80H，若要在DDRAM的41H（即第二行第二列）处显示数据，则必须将41H加上80H即C1H，以此类推。

2）常用字符点阵码存储器（CGROM）

CGROM是液晶模块内部的字符发生存储器，LM016L内部已经存储了160个不同的点阵字符图形，这些字符有阿拉伯数字、英文字母的大小写、常用的符号和日文假名等，每一个字符都有一个固定的代码，比如大写的英文字母“A”的代码是01000001B（41H），显示时模块把地址41H中的点阵字符图形显示出来，我们就能看到字母“A”。因为LM016L识别的是ASCII码，所以可以用ASCII码直接赋值，在单片机编程中还可以用字符型常量或变量赋值，如'A'。当我们把字符的ASCII码送入DDRAM相应地址时，就在LCD的相应位置显示该字符。

3）自定义字符点阵码存储器（CGRAM）

CGRAM是用户自定义字符发生存储器，用来存放用户自定义字符的点阵信息，该存储器共有6位地址，64个存储单元（不同厂家的产品数量不同），每个字符使用8个存储单元存放点阵，所以该区域可以存放8个字符的内容，和CGROM中字符的统一编码一样，字符码为0～7。当把该编码送入DDRAM某个地址时，就在LCD的相应位置显示出自定义的字符。

设置自定义字符点阵码的方法是：先用指令7向LCD写入点阵码存于CGRAM的地址（由表5-3中的指令7可知，地址为40H～7FH）；然后以写数据的方式，向LCD写入存于CGRAM的字符点阵码（点阵字节取的方法是按行从上到下，左边为低位数）。

5.4.4　LM016L基本操作函数

LM016L显示器的基本操作函数如下：

```
//lcd.c 文件
#include<reg52.h>
#include<intrins.h>
#define  LCDDATA  P2                        //LM016L 的数据线
#define delay5us(n) {  unsigned char i; \
                     for(i = 0; i<n; i++)\
                       _nop_(); }           //延时 5n(μs)宏定义,12MHz 时钟时误差均为 + 2μs

sbit RS = P3^0;                             //LM016L 的数据/指令选择控制线
sbit RW = P3^1;                             //LM016L 的读写控制线
sbit EN = P3^2;                             //LM016L 的使能控制线
```

```
sbit BF  = LCDDATA^7;                          //读 LM016L 状态线

bit LcdBusy()                                  //读 LM016L 忙状态函数
{                                              //返回值
   RS = 0;                                     //选择指令寄存器
   RW = 1;                                     //选择写
   EN = 1;                                     //使能
   BF = 1;                                     //状态位设置为输入
   delay5us(4);
   F0 = BF;                                    //读状态存于 PSW.5 位
   EN = 0;
   return F0;
}
void LcdWriteCommand(unsigned char com)        //LM016L 写命令函数
{  while(LcdBusy());                           //LM016L 忙时等待
   RS = 0;                                     //选择指令寄存器
   RW = 0;                                     //选择写
   LCDDATA = com;                              //把命令字送入数据口
   EN = 1;                                     //使能线电平变化,命令送入 LM016L 的 8 位数据口
   delay5us (4);                               //延时,让 LM016L 准备接收数据
   EN = 0;
}
void LcdWriteData(unsigned char dat)           //LM016L 写数据函数
{  while(LcdBusy());
   RS = 1;                                     //选择数据寄存器
   RW = 0;                                     //选择写
   LCDDATA = dat;                              //把要显示的数据送入数据口
   EN = 1;                                     //使能线电平变化,数据送入 LM016L 的 8 位数据口
   delay5us(4);                                //延时,让 LM016L 准备接收数据
   EN = 0;
}
void LcdInit()                                 //1602 初始化函数
{
   LcdWriteCommand(0x38);                      //8 位数据,双行显示,5×7 字形
   LcdWriteCommand(0x0c);                      //开启显示屏,关光标,光标不闪烁
   LcdWriteCommand(0x06);                      //显示地址递增,即每写一数据,显示位置后移一位
   LcdWriteCommand(0x01);                      //清屏
   delayms(10);                                //延时,函数定义见例 2-2
}
```

5.4.5 LM016L 应用编程

在 LM016L 上显示字符和数字,显示结果如图 5-15 所示。C 语言程序如下:

```
//main.c 文件
#include<reg52.h>
extern void LcdInit();                         //LCD 操作函数定义见上面 lcd.c 文件
extern void LcdWriteCommand(unsigned char com);
extern void LcdWriteData(unsigned char dat);
unsigned char *disBuf =" I love MCU  0123456789ABCDEF" ;
void main()                                    //主函数
```

```
{    unsigned char i;
     LcdInit();
     LcdWriteCommand(0x80);                    //在 LCD 第一行上显示
     for(i = 0; i<16; i++)
         LcdWriteData(disBuf[i]);              //显示前 16 个字符
     delayms(10);                              //延时 10ms,否则不显示. 函数定义见例 2-2
     LcdWriteCommand(0xc0);                    //在 LCD 第二行上显示
     for(; i<32; i++)                          //显示后 16 个字符
         LcdWriteData(disBuf[i]);
     while(1);
}
```

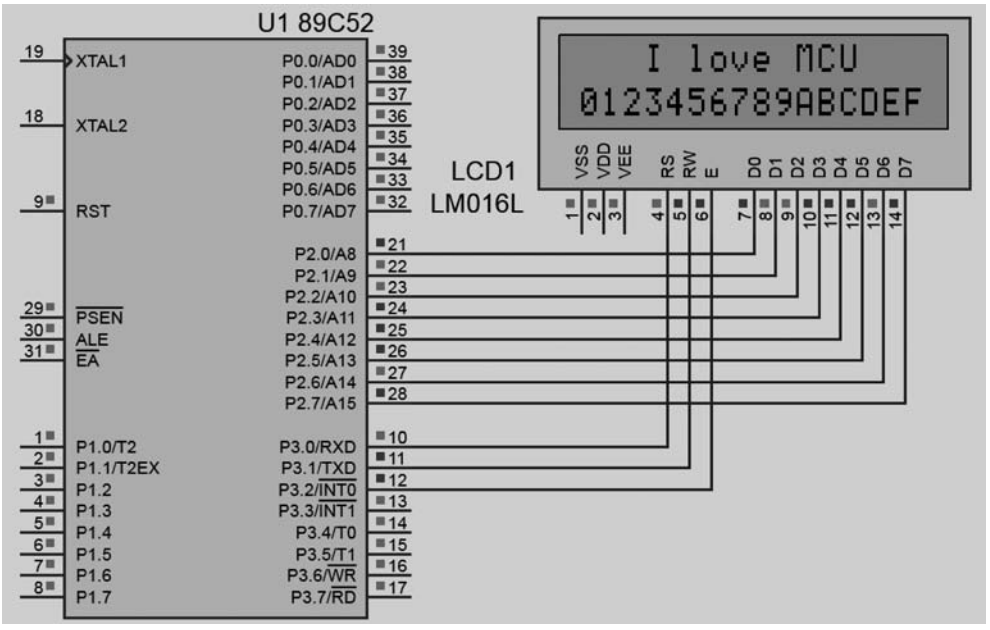

图 5-15　LCD 应用仿真电路

思考题与习题

(1) 对于 MCS-51 单片机,读端口锁存器和“读引脚”有何不同? 各使用哪些指令?

(2) MCS-51 单片机的 P0～P3 口结构有何不同? 用作通用 I/O 口输入数据时应注意什么?

(3) P0 口用作通用 I/O 口输出数据时应注意什么?

(4) 简述多位 LED 数码管动态显示的方法。

(5) 为什么要消除键盘的机械抖动? 消除抖动有哪些方法?

(6) 简述行扫描法识别行列式键盘按键的方法。

(7) 简述行列快速扫描法识别行列式键盘按键的方法。

(8) 参照图 1-34,使用 Proteus 画电路,用单片机的 P0 口控制 8 个 LED,在 Keil C 下创建项目 xiti5-8,对 1.5.3 节中的程序做修改,使点亮两个 LED 循环右移显示,对程序编译链

接产生可执行文件 xiti5-8.hex,装载单片机,模拟运行并观察,如果不显示或不正确,分析原因并做修改,使其正确显示。

(9) 使用 89C52 单片机的 P3 口接 8 个按钮,按钮的另一端接地,单片机的 P0 口接 8 个 LED 的阴极,另一端接电阻、电源,编写程序,读取按钮的状态,从 P0 口输出显示,使用 Proteus 绘制电路、仿真运行,观察运行情况。

(10) 把 89C52 单片机的 P0、P2 口连接起来,编写程序,从 P2 口输出数据(连续的 ASCII 码),从 P0 输入数据。把 F1 位(PSW.1)作为数据的标志,对 P2 口来说,发送数据之前先要查询 F1 是否为 0,为 0 表明发送的数据已经读取,可再次发送,发送之后,对 F1 置 1;对 P0 来说,查询 F1 是否为 1,为 1 表明有数据发送过来,可以读取,读取之后,对 F1 清 0。间隔 1s 查询 1 次,为 P2、P0 交替查询(对 P2 仅查询 F1 是否为 0,对 P0 仅查询 F1 是否为 1)。为了观察传送的数据,可以在 P3 口接 8 个 LED,LED 的阳极接 P3 口,阴极接地。使用 Proteus 绘制电路、仿真运行,观察运行情况。

(11) 用甲乙两个单片机实现通过 I/O 传输数据的功能。把它们的 P2 口连接,甲发送、乙接收数据,并且把它们的 P30、P31、P32 连接,分别作为发送了数据、接收到数据、请求发送数据的联络信号(前者由甲发出,后两个由乙发出),低电平有效。双方联络与数据传输过程如下:

对于甲方,当查询到自己的 P32 为低(有效电平),则通过 P2 口输出 1 个连续的 ASCII 码,并且从 P30 发送出低电平信号;当查询到 P31 为低电平,则得知乙已经读取了 P2 口的数据,从 P30 发送出高电平的无效信号。对于乙方,当查询到 P30 为低电平,则从自己的 P2 口读入数据,然后从 P31 发送出低电平信号,表示已经读取了数据,同时从 P32 发送出高电平的无效信号,表示撤销请求发送信号;当查询到 P30 为高电平,则从 P31 发送出高电平的无效信号,至此,表明一次数据传输结束。每间隔 1s,乙方从 P32 发送出低电平信号,请求甲方发送数据(这就是数据流控制,在实际中的实际间隔由程序控制)。

编写甲乙两个单片机的不同程序,在 Proteus 下绘制电路,把两个程序分别下载到对应的单片机中,仿真运行。为了观察运行情况,可以在两个单片机的 P0 口接 8 个 LED,分别显示出发送和接收的数据。

(12) 对于第(11)题的联络信号,去掉中间那个“收到数据”的应答信号 P31,只要发送了数据信号 P30、请求发送数据信号 P32,其他都一样。编写两个单片机的程序,然后使用 Proteus 绘制电路、仿真运行,观察运行情况。

(13) 仿照第(12)题,再加一个 P37 仲裁信号线,实现两个单片机之间的双向数据传输。当某个单片机需要发送数据时先查询 P37 的状态(先输出 1 再读入),如果为低,则等待,自己处于接收方,按照接收方使用 P30、P32 信号,接收数据;如果查询到 P37 为高,则从 P37 引脚输出 0 使仲裁线变低,自己变为发送方,按照发送方使用 P30、P32 信号,发送数据。可以用 P36 引脚对地接一按钮,改变原来收发双方的角色。使用 Proteus 绘制电路、仿真运行,观察运行情况。这种情况两个单片机程序是一样的,主需要编写一个程序。

(14) 设计一个 89C52 单片机应用系统,使用 6 位共阴极数码管显示器,其段和位分别由 P0 口和 P2 口控制,各段用上拉电阻做限流,各位用 74LS245 驱动,并编写能够显示 0～9、A～F 的十六进制数的函数。使用 Proteus 绘制电路、仿真运行。

(15) 设计一个 89C52 单片机应用系统,使用 6 位共阳极数码管显示器,其段和位分别

由 P0 口和 P2 口控制，各位用 74LS245 驱动，并编写能够显示 0～9、A～F 的十六进制数的函数。使用 Proteus 绘制电路、仿真运行。

(16) 某个 89C52 单片机应用系统的 P1 口连接一个 3 行 5 列的矩阵式键盘，用行扫描法编写识别按键的函数，无键按下返回 0xff，有键按下返回按下键的键号(0～14)。

(17) 某个 89C52 单片机应用系统的 P1 口连接一个 3 行 5 列的矩阵式键盘，用行列快速扫描法编写识别按键的函数，无键按下返回 0xff，有键按下返回按下键的键号(0～14)。

(18) 某个 89C52 单片机应用系统的 P1 口连接一个 m 行 n 列的矩阵式键盘，用行扫描法编写一个通用的识别按键的函数，无键按下返回 0xff，有键按下返回按下键的键号(0～$m\times n-1$)。

(19) 某个 89C52 单片机应用系统的 P1 口连接一个 m 行 n 列的矩阵式键盘，用行列快速扫描法编写一个通用的识别按键的函数，无键按下返回 0xff，有键按下返回按下键的键号(0～$m\times n-1$)。

(20) 某个 89C52 单片机应用系统的按键较多，P2、P3 口分别连接一个 m 行、n 列的矩阵式键盘，用行列快速扫描法编写一个通用的识别按键的函数，无键按下返回 0xff，有键按下返回按下键的键号(0～$m\times n-1$)。设 $4<(m、n)<8$，并且 P2、P3 口都是从最低位开始连续使用。

(21) 使用 Proteus 设计一个 89C52 单片机应用系统，系统包含键盘和数码管显示电路，用 P1 口控制 16 个按键，用 P0 和 P2 口分别控制 6 位数码管的段和位，各段用上拉电阻做限流，各位用 74LS245 驱动。用 Keil C 编写程序，实现扫描识别键盘按键、扫描显示数码管的功能，并且把按下的键的键号显示在数码管上，每按下一次键，显示的数左移 1 位。

(22) 修改第(21)题的应用，将显示改为字符式 LCD 显示，其他要求不变。

(23) 将 5.3.5 节中的例子改为使用 LM016L 液晶显示器显示，其他要求不变，用 Keil C 编写程序，用 Proteus 仿真。