第 3 章　P0～P3 口输入/输出的 C51 编程

通过第 2 章内容的介绍，我们知道 51 系列单片机内部有 4 个并行的 I/O 口，分别为 P0(P0.7～P0.0)、P1(P1.7～P1.0)、P2(P2.7～P2.0)、P3(P3.7～P3.0)。这 4 个并行的 I/O 口除具有基本的输入/输出功能外，还具有和中断系统、定时器/计数器、串行口及外部扩展时的地址总线、数据总线、控制总线等有关的第二功能。关于第二功能的内容在后续相应的章节中讨论，本章主要介绍 51 单片机片内并行口的原理及输入/输出操作。

3.1　51 单片机的 P0～P3 口基础知识

在介绍 51 单片机的 P0～P3 口之前，先看两个概念。

双向口：单片机的 I/O 口是 CPU 与片外设备进行信息交换的通道，是为提高接口的驱动能力，具有由场效应管组成的输出驱动器。当驱动器场效应管的漏极具有开路状态时，该口就具有高电平、低电平和高阻抗 3 种状态，称为双向口。

准双向口：单片机 I/O 口的输出场效应管的漏极接有上拉电阻，该口具有高电平、低电平两种状态，称为准双向口。

▶▶ 3.1.1　P0～P3 口结构

在 51 单片机内部包含 4 个并行的 I/O 接口，分别称为 P0 口、P1 口、P2 口和 P3 口，每一个口都是 8 位的，每个口的位都有一个输出锁存器和一个输入缓冲器。输出锁存器用于存放需要输出的数据，每个端口的 8 位输出锁存器构成一个特殊功能寄存器，且冠名与端口相同；输入缓冲器用于对端口引脚上输入的数据进行缓冲，因此各引脚上输入的数据必须一直保持到 CPU 把它读走为止。图 3-1 所示是 P0～P3 口的位结构示意图，其中图 3-1(a)是 P0 口位结构示意图，图 3-1(b)是 P1 口位结构示意图，图 3-1(c)是 P2 口位结构示意图，图 3-1(d)是 P3 口位结构示意图。

P0、P1、P2 和 P3 端口的电路形式不同，其功能也不同，但作为基本的输入/输出口，其操作大同小异，现在主要以 P1 为例进行说明。

在图 3-1(b)中，锁存器起输出作用，场效应管 VT1 与上拉电阻组成输出驱动器，以增大负载能力。三态门 1 是输入缓冲器，三态门 2 在端口操作时使用。

P1 口作为通用的 I/O 接口使用，具有输出、读引脚、读锁存器三种工作方式。

1．输出方式

P1 口工作于输出方式，此时数据经内部总线送入锁存器存储。如果某位的数据为 1，则该位锁存器输出端 $Q=1$，$\overline{Q}=0$ 使 VT1 截止，从而在引脚 P1.X 上出现高电平；反之，如果数据为 0，则 $Q=0$，$\overline{Q}=1$ 使 VT1 导通，在引脚 P1.X 上出现低电平。

2．读引脚方式

读引脚时，控制器打开三态门 1，引脚 P1.X 上的数据经三态门 1 进入芯片的内部总线，并送到累加器 A。输入时无锁存功能。

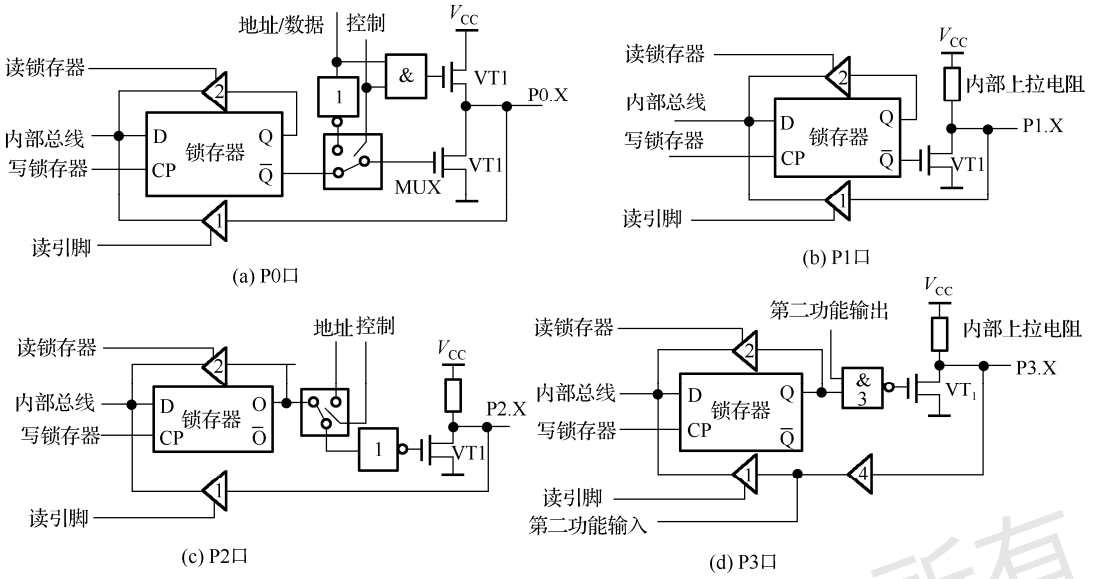


图 3-1　P0～P3 口位结构示意图

在单片机执行读引脚操作时，如果锁存器原来寄存的数据 Q = 0，那么由于 $\overline{Q}$ = 1，将使 VT1 导通，引脚被始终钳位在低电平上，不可能输入高电平。为此，使用读引脚指令前，必须先用输出指令置 Q = 1，使 VT1 截止。这就是 P1 被称为准双向口的原因。“准双向”的含义为输出直接操作，输入前需先置 1，再输入。

3．读锁存器方式

51 系列单片机有很多指令可以直接进行端口操作，这些指令的执行过程分成“读—修改—写”三步，即先将端口的数据读入 CPU，在 ALU 中进行运算，运算结果再送回端口。执行“读—修改—写”类指令时，CPU 实际是通过三态门 2 读回锁存器 Q 端的数据的。

这种读锁存器的方式是为了避免可能出现的错误。例如，用一根口线去驱动端口外的一个晶体管基极，当向口线写“1”时，该晶体管导通，导通后的 PN 结会把端口引脚的高电平拉低，这样直接读引脚就会把本来的“1”误读为“0”。但若从锁存器 Q 端读，就能避免这样的错误，得到正确的数据。也就是说，如果某位输出为 1，有外接器件拉低电平，就有区别了，读锁存器状态是 1，读引脚状态是 0，锁存器状态取决于单片机企图输出什么电平。引脚状态则是引脚的实际电平。

是读引脚还是读锁存器，其过程 CPU 内部会自动处理，读者不必在意。但应注意，当作为读引脚方式使用时，应先对该口写“1”，使场效应管截止，再进行读操作，以防止场效应管处于导通状态，使引脚为“0”，从而引起误读。

▶▶ 3.1.2　P0～P3 口特点总结

① 若要执行输入操作，P0～P3 口都必须先输出高电平，才能读取该端口所连接的外部设备的数据。

② P0 口 8 位皆为漏极开路输出，每个引脚可以驱动 8 个 LS 型 TTL 负载（通常把 100μA 的电流定义为 1 个 LSTTL 负载的电流）；P1～P3 口的 8 位类似于漏极开路输出，但已接上拉电阻，每个引脚可驱动 4 个 LS 型 TTL 负载。

③ P0 口内部无上拉电阻，执行输出功能时，外部必须接上拉电阻（一般 10kΩ 即可）；P1～P3

口内部具有约 30kΩ 的上拉电阻，执行输出操作时，无须连接外部上拉电阻。

④ 若系统连接外部存储器或 I/O 口芯片，P0 口作为地址总线（A7～A0）及数据总线（D7～D0）的复用引脚，P2 可作为地址总线（A15～A8）引脚；P3 口的 8 个引脚各具有其他功能，如表 2-2 所示。

强调：输出操作时，在相应的口线输出高、低电平，低电平时的电流（灌电流）比高电平时的电流（拉电流）大；输入操作就是读入相应引脚的高、低电平，在读入引脚高、低电平之前必须先将相应引脚进行置 1 操作。

3.2　输出操作

51 单片机的 P0～P3 口可以作为基本的输出口使用（注意：P0 口与 P1～P3 口不同），其输出的高、低电平可以控制外部设备。在输出控制时可以采用字节操作，也可以采用位操作。

▶▶ 3.2.1　基本输出操作举例——字节输出与位输出

1．字节输出

在字节输出时，只要通过赋值操作符对相应的口线输出高、低电平即可。例如：

```
P1=0X0F;          //P1 口高 4 位输出低电平，低 4 位输出低电平
```

在输出操作时，灌电流（输出低电平时）要比拉电流（输出高电平时）大。

【例 3-1】 电路如图 3-2 所示，P1 口接 8 个发光二极管作为输出指示，编程实现使 8 个发光二极管按一定的频率亮、灭闪烁。

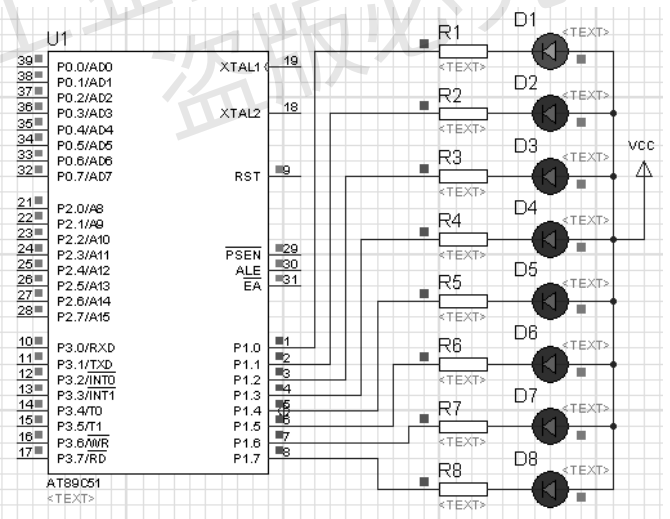


图 3-2　例 3-1 电路图

分析：P1 口输出“高电平”时灯灭，输出“低电平”时灯亮；亮、灭闪烁可以通过一个软件延时程序实现。

```
#include<reg51.h>                    //特殊功能寄存器声明
void  msec(unsigned  int  x)         //延时函数
{   unsigned  char  j;
```

```
    while(x--)
    {   for(j=0;j<125;j++);
    }
}
void  main(  )                              //主函数
{  while(1)
    {  P1=0x00;      delay(500);            //亮
       P1=0xff;      delay(500);            //灭
    }
}
```

本例说明：

① delay(500)是定性地延时 500 毫秒，准确时间不定。

② while(1)是无限循环。

2．位输出

位输出操作需要对使用的位进行“位定义”。位定义采用 sbit。

【例 3-2】 电路如图 3-2 所示，编程实现 P1.3 所接的发光二极管亮、灭闪烁。

```
#include<reg51.h>
sbit  LED3=P1^3;                          //对 P1.3 进行位定义
void  delayxms(unsigned  int  xms)    //延时函数
{  unsigned  int  t1, t2;
   for(t1=xms;t1>0;t1--)
   for(t2=120;t2>0;t2--);
}
void  main(  )
{   while(1)
    {  LED3=0;   delayxms (500);
       LED3=1;   delayxms (500);
    }
}
```

修改：

① P1.1、P1.3、P1.5 对应的灯亮、灭闪烁。

② P1.0 对应的灯亮时，P1.7 对应的灯灭；P1.0 对应的灯灭时，P1.7 对应的灯亮。

③ P1.0 对应的灯以 1 秒周期、P1.1 对应的灯以 500 毫秒周期亮、灭闪烁。

强调：例 3-1 和例 3-2 都以 P1 口为输出口，实际应用时也可以采用 P0、P2、P3 口，但采用 P0 口时要外接上拉电阻。

▶▶ 3.2.2　扩展输出操作举例——流水灯与霹雳灯

流水灯和霹雳灯是指一定数量的灯按一定的规律进行亮、灭闪烁。

【例 3-3】流水灯示例。电路如图 3-2 所示，编程实现 8 个灯从低到高流水灯的显示闪烁。

分析：流水灯闪烁规律：11111110B—11111101B—11111011B—…—01111111B，从初值循环左移 1 位就可以。

```
#include<reg51.h>
#include<intrins.h>                    //内部函数库，请参看第 2 章函数部分的介绍
void  delay( unsigned  int  d )        //延时函数
{  while(--d>0);   }
void  main(  )
{  unsigned  char  sel;
   sel=0xfe;                           //初值为 P1.0 为 0
   while(1)
  {  P1=sel;  delay(50000);
     sel=_crol_(sel,1);                //循环左移 1 位
  }
}
```

说明：

① intrins.h 称为内部函数库，提供循环左移、循环右移等函数。

② sel=_crol_(del,1)是将变量 sel 循环向左移 1 位。

也可以这样编程：

```
#include<reg51.h>
void  delay(unsigned  int  d)
{  while(--d>0);  }
void  main( )
{  unsigned  char  i,  sel,  a;
   while(1)
  {  sel=0xfe;
     for( i=0; i<8; i++)
     {  P1=sel;   delay(50000);
        a=sel<<1;  sel=a|0x01;         //“<<”为左移 1 位，空出的位补“0”
      }
  }
}
```

修改：

① 两个灯左流水。

② 从左到右，一个一个亮保持到全亮，然后再重复。

【例 3-4】 霹雳灯示例。电路如图 3-2 所示，由 P1 口驱动 8 个 LED 灯，编程实现霹雳灯闪烁。

分析：所谓的霹雳灯是指一排 LED 里，任何一个时间只有一个 LED 亮，而亮灯的顺序为由左而右再由右到左，感觉就像一个 LED 由左跑到右再由右跑到左。霹雳灯规律：

11111110B — 11111101B — … — 01111111B — 10111111B — 11011111—…— 11111110—…。在程序设计上，有很多方法可以达到这个目的，例如采用计数循环方式，首先左移 7 次，再右移 7 次，如此循环不停。

程序框图如图 3-3 所示。

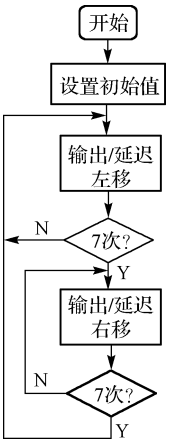


图 3-3　霹雳灯程序框图

C51 程序：

```
//==声明区=============================================
#include<reg51.h>
#define LED P1                          //定义 LED 接至 P1
void  delayxms(int);                    //声明延迟函数
//==主函数=============================================
main()
{   unsigned  char  i;                  //无符号字符型变量 i
    LED=0xfe;                           //初值=1111 1110,只有最右 1 灯亮
    while(1)
    {   for( i=0; i<7; i++)             //左移 7 次
        {   delayxms(500);              //延迟 100×5ms=0.5s
            LED=(LED<<1)|0x01;          //左移 1 位,并设定最低位为 1
        }                               //左移结束,只有最左 1 灯亮
        for( i=0; i<7; i++)             //右移 7 次
        {   delayxms(500);              //延迟 100×5ms=0.5s
            LED=(LED>>1)|0x80;          //右移 1 位,并设定最高位为 1
        }                               //结束右移,只有最右 1 灯亮
    }
}
/*延迟函数,延迟约 xms 毫秒*/
void  delayxms(unsigned  int  xms)      //延迟函数开始
{   unsigned  int  t1,t2;               //声明整数变数 t1,t2
    for( t1=0; t1<xms; t1++)            //计数 xms 次,延迟 xms 毫秒
    for( t2=0; t2<110; t2++);
}
```

修改：

① 实现双灯的霹雳灯功能。

② 花样流水灯的实现——8 个灯的亮、灭情况没有规律可循，不像例 3-3 和例 3-4。

▶▶ 3.2.3　扩展输出操作举例——8 段 LED 数码显示器

对于 51 单片机的 P0～P3 口，可以输出控制 8 段 LED 数码显示器。

1．认识 8 段 LED 数码显示器

（1）结构与原理

通常的七段 LED 数码显示器中有 8 个发光二极管，引脚如图 3-4(a)所示。a、b、c、d、e、f、g、dp 称为 LED 的段，公共端 com 称为 LED 的位。从引脚 a～dp 输入不同的 8 位二进制数，可显示不同的数字或字符。根据公共端的连接情况有共阴极和共阳极两种，如图 3-4(b)和(c)所示。对共阴极 LED 显示器的发光二极管的公共端 com 接地，当某发光二极管的阳极为高电平时，相应的发光二极管点亮；共阳极 LED 显示器则相反。

（2）显示器的驱动问题

显示器中的每个段是一个发光二极管，要其正常发光必须提供足够的电流。不同的 LED 显示器具有不同的正常发光电流范围，因此在设计硬件电路时要为显示器提供驱动电路，以保证其正常工作。

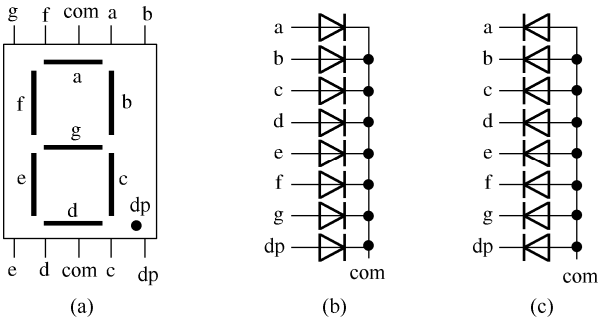


图 3-4　七段 LED 数码显示器引脚与结构图

2．显示函数的编写问题

硬件电路设计好之后，编写显示函数对显示进行管理非常关键。显示函数的功能是将显示缓冲区中的内容查表送到相应的显示器上完成显示。在对显示进行软件管理的过程中，要注意以下几点。

（1）根据硬件电路的结构建立一个显示的代码表

建立代码表的目的是解决好显示内容和显示代码转换问题。

显示的内容就是在显示器上要显示出来的内容，如 1、4、A 等。如要显示 1，对共阴极显示器来说，需要 b、c 段亮，其余的段不亮。在单片机中，段是由数据线上的内容来控制的，如果数据线 D7、D6、D5、D4、D3、D2、D1、D0 对应着 LED 的段 dp、g、f、e、d、c、b、a，要在显示器上显示 1，则在数据线需要输出 00000110B（06H），这个 06H 就称为 1 的显示代码。如果硬件电路确定了，每一个要显示的内容都有一个固定的显示代码和它对应。在显示过程中，实际上是把要显示的内容的代码送到数据线，由数据线将显示的代码送到显示器的段上，这样在显示器上就显示出相应的内容了。

表 3-1 列出了 LED 显示器的七段码，展示了数据线 D7、D6、D5、D4、D3、D2、D1、D0 对应 LED 的段 dp、g、f、e、d、c、b、a 的硬件连接时的显示内容和显示代码之间的关系。如果硬件连接发生变化，对应关系将发生变化。

表 3-1　LED 显示器的七段码

显示字符	共阴极七段码	共阳极七段码	显示字符	共阴极七段码	共阳极七段码
0	3FH	C0H	9	6FH	90H
1	06H	F9H	A	77H	88H
2	5BH	A4H	B	7CH	83H
3	4FH	B0H	C	39H	C6H
4	66H	99H	D	5EH	A1H
5	6DH	92H	E	79H	86H
6	7DH	82H	F	71H	8EH
7	07H	F8H	P	73H	8CH
8	7FH	80H	-	40H	BFH

在 code 区域中，将所有要显示的内容（提前是已知的）的显示代码按照一定的顺序建立一个表格，这个表格称为显示的代码表，例如：

```
unsigned char code table[18]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x73,0x40};       //显示的代码表
```

（2）开辟显示缓冲区

在片内 RAM 中开辟出一块特殊区域—显示缓冲器。显示缓冲区中存放要显示的内容所对应的代码在代码表中的相对位置。显示缓冲区的位数和硬件电路中显示器的位数相同，每个显示缓冲器对应着一位显示器。例如 6 位显示器，显示缓冲区可如下：

```
unsigned char data dis_buf[6];                    //显示缓冲区
```

（3）查表并操作相应的显示器

根据显示缓冲区中的内容（相应的显示器要显示的内容所对应的显示代码在代码表中的相对位置），在代码表中得到相应的显示代码，送到相应的显示器上进行显示。

（4）显示函数的调用

在主函数中对显示缓冲区送相应的数据，然后调用显示函数就可以了。在主函数中，显示缓冲区的内容变化了，显示的内容就随着变化了。对显示函数的调用因不同的显示方式而不同。对于静态显示，只要显示缓冲区的内容发生变化就可以调用显示函数；对于动态显示方式，可以分为随机调用和定时调用两种方式。下面通过例题加深理解。

3．8 段 LED 静态显示技术

所谓静态显示，就是当显示器显示某一个字符时，相应的发光二极管恒定地导通或截止。例如七段数码显示器的 f、e、d、c、b、a 导通，dp、g 截止，则显示 0。这种显示方式中，每一位显示器都需要一个 8 位输出口控制，所以占用硬件多，一般用于显示器位数较少的场合。

【例 3-5】 利用 51 单片机的并行口作为静态显示的控制口的示例。

电路如图 3-5 所示，通过 AT89C51 单片机的 P1 口、P3 口作为两位共阳极数码管静态显示的控制口。编程实现显示“Ab”的程序。

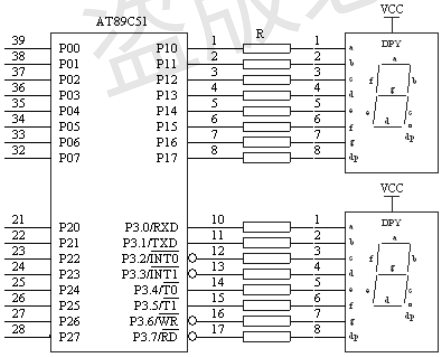


图 3-5　例 3-5 的电路原理图

说明：共阳极显示器、两位静态显示；本程序只能显示 A、b，不能显示别的内容。

C51 程序如下：

```
#include<reg51.h>
#define  uchar  unsigned  char
#define  uint  unsigned  int
uchar  data  dis_buf[2];                          //显示缓冲区
uchar  code  table[ ]={0x88,0x83};                //显示的代码表 A、b
void  display(void)                               //显示函数
```

```
{   uchar  segcode;
    segcode=dis_buf[0];                          //P1 口显示
    segcode=table[segcode];
    P1=segcode;
    segcode=dis_buf[1];                          //P3 口显示
    segcode=table[segcode];
    P3=segcode;
}
void  main(void)                                 //主函数
{   dis_buf[0]=0;    dis_buf[1]=1;               //显示缓冲器赋值
    display( );                                  //调用显示函数
    while(1);
}
```

示例中的显示函数 display( )可以再简单一些，如下面程序段：

```
void  display(void)
{   P1=table[dis_buf[0]];                        //P1 口显示
    P3=table[dis_buf[1]];                        //P3 口显示
}
```

修改：

① 显示“12”。

② 静态轮流显示“12”“- -”和“Ab”。延时时间采用软件延时。

③ 秒表：从 00 显示开始，1 秒显示器加 1。

4. 8 段 LED 动态显示技术

LED 动态显示是将所有显示器的相同的段连接在一起，接在一个 I/O 口上，称为段口，共阴极端或共阳极端分别由相应的 I/O 口线控制，称为位口。图 3-6 所示为一个 8 段 LED 动态显示器电路图（具体电路在设计时需要考虑 LED 段口和位口的输出驱动问题）。

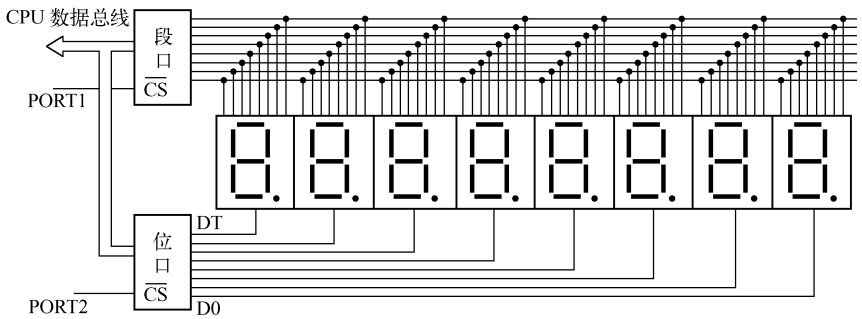


图 3-6　8 段 LED 动态显示器电路图

动态显示就是一位一位地轮流点亮各位显示器（扫描方式），对每一位显示器每隔一定时间点亮一次。即从段口上按位次分别送所要显示字符的段码，在位控制口上也按相应的次序分别选通相应的显示位码（共阴极送低电平，共阳极送高电平），选通的位就显示相应字符，并保持几毫秒的延时，未选通位不显示字符（保持熄灭）。这样，对各位显示就是一个循环过程。从计算机的工作过程来看，在

一个瞬时只有一位显示字符，而其他位都是熄灭的，但因为人的“视觉滞留”和显示器的“余辉”，这种动态变化是觉察不到的。从效果上看，各位显示器都能连续而稳定地显示不同的字符，这就是动态显示。

动态接口技术的硬件连接比较简单，但显示函数相对麻烦。显示函数是对相应的段口、位口进行输出操作，编程框图如图 3-7 所示。

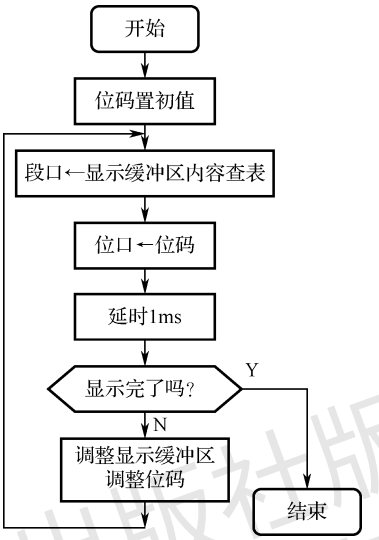


图 3-7　动态显示函数框图

【例 3-6】 利用 51 单片机的并行口作为动态显示的段口与位口的示例。

电路如图 3-8 所示，通过 51 单片机的 P1 口作为段口，P3 口作为位口构成 6 位 LED 动态显示的硬件电路。

说明：74LS245 是段驱动，7407 是位驱动。

6 位数码管动态显示“123456”的 C51 程序如下。

```
#include<reg51.h>
#define  uchar  unsigned  char
uchar  data  dis_buf[6];                    //显示缓冲区
uchar  code  table[18]={0x06,0x5b,0x4f,0x66,0x6d,0x7d,};
                                            //代码表 1、2、3、4、5、6
void  dl_ms()                               //延时 1ms 函数
{   unsigned  int  j;
    for( j=0; j<150; j++)    ;
}
void  display(void)                         //显示函数
{   uchar  segcode, bitcode, i;
    bitcode=0xfe;                           //位码赋初值
    for( i=0; i<6; i++)
    {   segcode=dis_buf[i];                 //显示缓冲器内容查表
        P1=table[segcode];                  //端口输出操作
        P3=bitcode;                         //位口输出操作
        dl_ms( );                           //延时 1 毫秒
```

```
            P3=0xff;                          //关闭显示
            bitcode=bitcode<<1;               //调整位码
            bitcode=bitcode|0x01;
        }
    }
    void  main(void)
    {   dis_buf[0]=0; dis_buf[1]=1;           //显示缓冲区赋初值
        dis_buf[2]=2; dis_buf[3]=3;
        dis_buf[4]=4;dis_buf[5]=5;
        while(1)
        {   display( );                       //调用显示
        }
    }
```

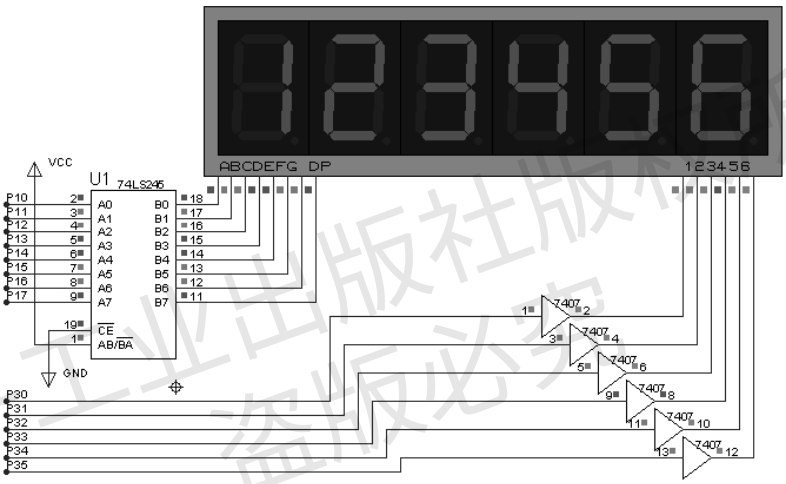


图 3-8　例 3-6 的电路原理图

对于显示函数的调用可以有两种方式：随机调用、定时调用。

随机调用是在主函数中，当显示缓冲区的内容发生变化后，就需要对显示函数进行调用，两次调用之间的时间间隔不能太长，间隔时间太长将发生显示的闪烁现象。上面的程序是随机调用。关于定时调用将在介绍定时器/计数器一节后讨论。

修改：

① 显示“ABCDEF”。

② 静态轮流显示“123456”和“ABCDEF”。延时时间采用软件延时。

③ 时钟：从 00 时 00 分 00 秒开始，1 秒显示器加 1。

强调：理解显示代码表的关键是“顺序问题”；理解显示缓冲区的关键是，显示缓冲器中存放的是要显示的内容所对应的代码在显示代码表中的相对位置。

3.3　输入操作

51 单片机的 P0～P3 口可以作为基本的输入口使用，其输入信号是由外部电路决定的，可以分为两大类：电平信号、脉冲信号。这两类信号可以通过闸刀开关、按钮开关两类开关来模拟。如图 3-9

所示。图 3-9(a)是闸刀型开关示意图；图 3-9(b)是按钮型开关示意图；图 3-9(c)是闸刀型开关输入电路示意图；图 3-9(d)是按钮型开关输入电路示意图。

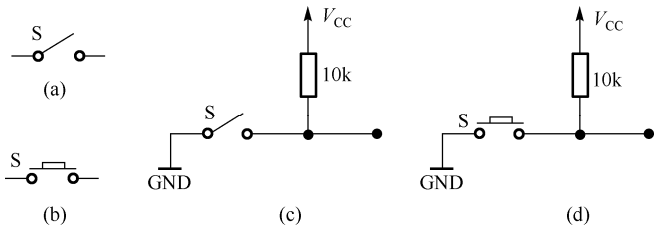


图 3-9　闸刀型与按钮型开关示意图

▶▶ 3.3.1　闸刀型开关输入信号

闸刀型开关输入信号具有残留功能，也就是不会主动恢复（弹回）。当我们按一下开关（或切换开关）时，其中的接点接通（或断开），若要恢复接点状态，则需再按一下开关（或切换）。常见的闸刀型开关如拨码开关、自锁按钮开关、面板用数字式拨码开关、电路板用数字式拨码开关等。

【例 3-7】 闸刀型开关输入信号例子。电路如图 3-10 所示，编程实现相应的开关闭合时，相应的灯亮。

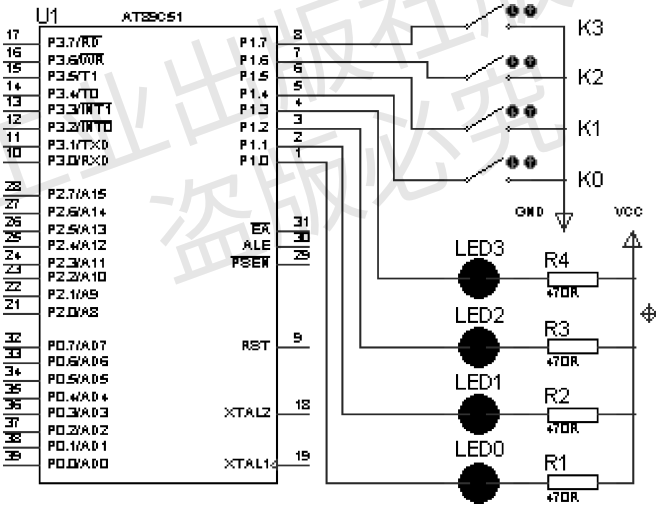


图 3-10　例 3-7 电路原理图

分析：P1 口高 4 位接开关，低 4 位接指示灯，因此 P1 口高 4 位为输入口，低 4 位为输出口；对于输入部分，键按下时输入口为低电平，抬起时输入口为高电平；对于输出部分，输出高电平灯灭，输出低电平灯亮；所以只要将相应的键的状态送到相应的输出位置就可以了。

```
#include<reg51.h>
sbit  LED0=P1^0;                                    //输出灯位定义
sbit  LED1=P1^1;
sbit  LED2=P1^2;
sbit  LED3=P1^3;
sbit  K0=P1^4;                                      //输入按键位定义
sbit  K1=P1^5;
```

```
sbit  K2=P1^6;
sbit  K3=P1^7;
void  main(  )
{  K0=1;  K1=1;  K2=1;  K3=1;                    //读之前先置 1
   while(1)
   {  LED0=K0;  LED1=K1;  LED2=K2;  LED3=K3;
   }
}
```

思考：为什么在读键状态之前要先置 1？

修改：

① 开关闭合时灯灭。

② K0 控制 LED3，K1 控制 LED2，K2 控制 LED1，K3 控制 LED0。

③ K0 打开时，灯全亮；K0 闭合时，灯按一定频率亮、灭闪烁。

强调：闸刀型开关一般用在需要两种状态的选择上，其状态要么是断开，要么是闭合。

▶▶3.3.2　单个按钮型开关输入信号

按钮型开关输入信号具有自动恢复（弹回）功能，当我们按下按钮，其中的接点接通，放开按钮后，接点恢复为断开。

开关在动作时并不是理想的状态，将可发生许多非预想状态，这种非预想状态称为抖动，如图 3-11 所示。

开关抖动的处理可以分为硬件去抖动和软件去抖动。硬件去抖动增加硬件投入，在单片机应用电路中，一般采用软件去抖动。

软件去抖动就是执行一段软件延时程序（8ms 左右）。这样，开关的处理程序框图如图 3-12 所示。在图 3-12 中，关注两个问题：去抖动、判断按键是否抬起。

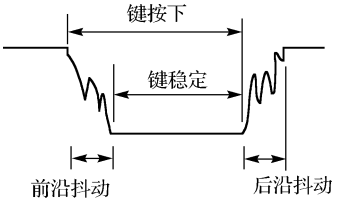


图 3-11　开关的抖动

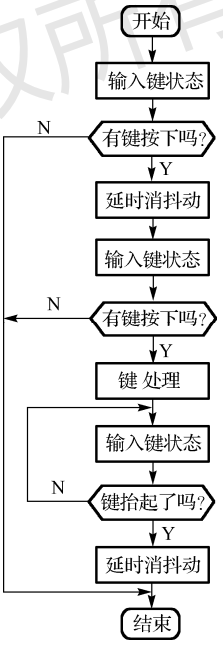


图 3-12　开关的处理程序框图

【例 3-8】 按钮开关模拟输入。电路如图 3-13 所示，开始高 4 位的灯亮，低 4 位的灯灭，编程实现 S1 按钮按一下，4 个灯一组亮、灭交替。

分析：按键按下时为低电平，抬起时为高电平；按键的消抖动需延时 8ms；P1 口作为输出口控制灯，初值为 0FH，亮、灭交替取反即可。

```
#include<reg51.h>
#define  uchar  unsigned  char
#define  uint  unsigned  int
sbit  S1=P3^2;                          //按键位定义
void  dlxms( uint  xms)                 //延时 xms 毫秒
```

```
{  uint  t1, t2;
   for( t1=0;  t1<xms; t1++)
   for( t2=0; t2<110; t2++);
}
void  main( )
{   P1=0x0f;                              //P1 口赋初值
    while(1)
    {   S1=1;                             //读之前先置 1
        if( S1==0)                        //如果按键按下
        {   dlxms(10);                    //延时，去抖动
            if( S1==0)                    //键继续按下吗?
            {   dlxms(10);                //去抖动
                P1=~P1;
                while( S1==0);            //按键没抬起，等待
                dlxms(10);                //抬起，延时
            }
         }
    }
}
```

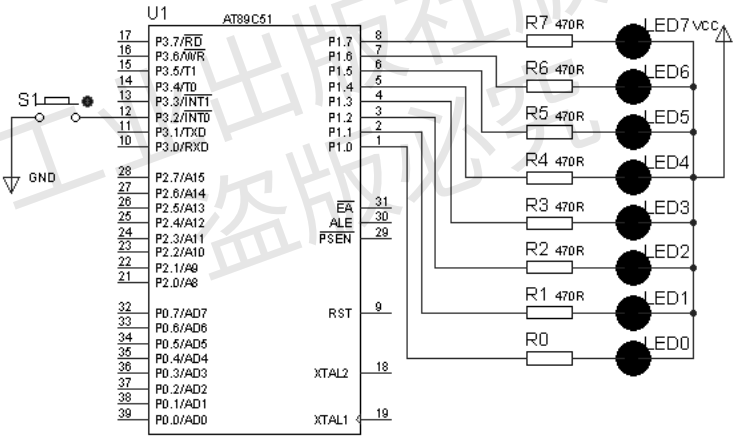


图 3-13　例 3-8 电路原理图

说明：开关接在 P3.2 上是为以后介绍中断的时间准备的；本例题是属于查询方式，第 4 章介绍中断时可以模拟外部中断输入。

【例 3-9】 电路如图 3-13 所示，开始是所有的灯都亮，按一下 S1，灯变为 500ms 闪烁，再按一下，变为全亮。（相当于 S1 为一个控制开关，控制着灯的亮、灭闪烁。）

分析：注意和例题 3-8 的不同。定义一个位单元，按键每动作一次，该位单元取反：该单元为 0 时，灯全亮；该单元为 1 时，灯闪烁。

```
#include<reg51.h>
#define  uchar  unsigned char
#define  uint  unsigned int
sbit  S1=P3^2;
```

```
bit  key=0;                          //定义一个位，存储按键的动作（偶、奇）
void  dlxms( uint  xms)              //延时函数
{  uint  t1, t2;
   for( t1=0; t1<xms; t1++)
   for( t2=0; t2<110; t2++);
}
void  keyscan()                      //键扫描函数
{  S1=1;                             //先置 1
   if( S1==0)                        //键按下吗?
   {  dlxms(10);                     //延时消抖动
      if( S1==0)
      {  key=~key;                   //取反
         while( S1==0);              //键抬起了吗?
      }
   }
}
void  main( )
{   P1=0x00;                         //灯全亮
    while(1)
    {   keyscan();                   //调用键扫描函数
        if(key==0)
        {  P1=0x00;
        }
        else
        {   dlxms(500);
            P1=~P1;
        }
    }
}
```

强调: 对按键的处理关键是两件事——去抖动、按键动作一次只能处理一次。在处理过程中，一般采用例 3-9 的处理方式，对按键单独设计一键扫描函数。

▶▶ 3.3.3　多个按钮型开关输入信号——键盘

键盘是由多个按钮开关组成的，单个按钮开关的处理（去抖动、键识别及抬起）在键盘中都是适用的，但在软件上也有其不同的地方。

1．键号、键值、键值表

（1）键号

用户在设计键盘程序时，为每一个按键定义了一个号码，称为键号。如 0 号键、1 号键等，找到了某个键的键号，就确定了该键的功能。

（2）键值

用户在设计键盘程序时，每一个按键根据某种算法，可以得到和其他按键不一样的值，该值称为该按键的键值。

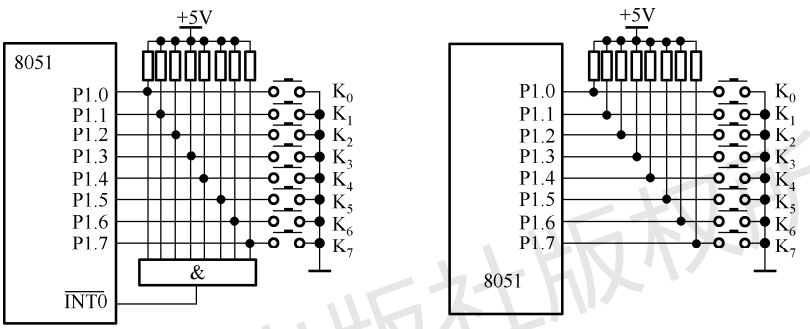
（3）键值表

用户在设计键盘程序时，将所有按键的键值，按照一定的顺序，在 code 区建立一个表格，该表格称为键值表。

2．独立式键盘接口技术

当按键的数量比较少（≤8）时，可采用独立式按键的硬件结构。独立式按键是指直接用一根 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 口线上的按键的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 3-14 所示。

图 3-14(a)所示为采用中断方式的独立式按键接口电路，图 3-14(b)所示为采用查询方式的独立式按键接口电路。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 口线上有确定的高电平。



(a) 中断方式　　(b) 查询方式

图 3-14　独立式按键的接口示意图

【例 3-10】 独立式按键示例。

电路如图 3-15 所示，P1 口作为并行接口按键的输入口，用 P3 口接一共阳极 LED 显示器，编程显示按键的键号 0～7，开始时显示 8。

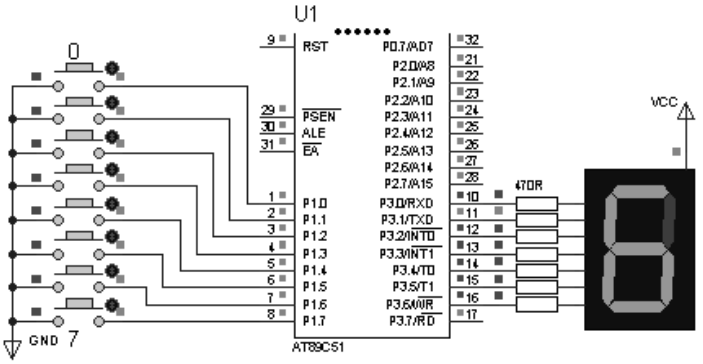


图 3-15　例 3-10 的电路原理图

说明：1 位共阳极静态显示；0 号键按下时，P1 口的内容为 11111110B；……；7 号键按下时，P1 口的内容为 01111111B。

C51 程序代码如下：

```
#include<reg51.h>
#define  uchar  unsigned  char
```

```
#define  uint  unsigned  int
uchar  data  keycode=8;                          //键值的初值设为 8
uchar  data  dir_buf;                            //显示缓冲区
code  uchar  dirtab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0xbf};
                                                 //显示的代码表
code  uchar  keytab[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
                                                 //键值表
void  dl_xms(uint  xms)                          //延时 xms 毫秒
{   uint  t1,t2;
    for(t1=xms;t1>0;t1--)
    for(t2=110;t2>0;t2--);
}
void  dir()                                      //显示函数
{   P3=dirtab[dir_buf];
}
void  keyscan( )                                 //键盘扫描函数
{  uchar  key1;
   P1=0xff; key1=P1;                             //读键盘的值
   if(key1!=0xff)                                //如果有键按下
   {  dl_xms(8);                                 //延时消抖动
      P1=0xff;  key1=P1;                         //再读键盘的值
      if(key1!=0xff)                             //继续按下
      {  keycode=0;
         while(key1!=keytab[keycode])            //查表得键号
         {  keycode=keycode+1;
            if(keycode==8)
            break;
         }
         while(P1!=0xff);                        //等待键抬起
      }
   }
}
void  main( )
{  dir_buf=8;                                    //缓冲区送 8，显示-
   dir( );
   while(1)
   {  keyscan( );                                //调用键扫描函数
      dir_buf=keycode;                           //键号送显示缓冲区
      dir( );
   }
}
```

修改：电路如图 3-15 所示：两个输入按键（如 P1.6、P1.7）一个为“+1”键，一个为“–1”键，开始显示器显示“5”，然后根据按键显示后面的内容。

2．矩阵键盘接口

当按键的数量比较多时，必须采用行列式键盘。行列式键盘又称为矩阵式键盘。

（1）行列式键盘的硬件结构

行列式键盘的硬件结构比较简单，由行输出口和列输入口构成行列式键盘，按键设置在行、列的交点上。如图 3-16 所示为一个 4×4 的行列式键盘的硬件结构。

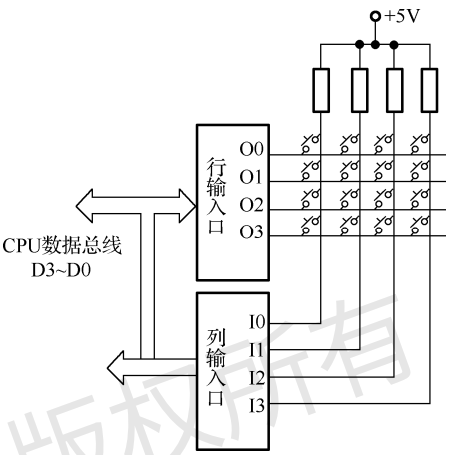


图 3-16　4×4 的行列式键盘的硬件连接

（2）行列式键盘的软件管理

对行列式键盘的软件管理分三步：

① 判断整个键盘是否有键按下。

采用粗扫描的办法。让所有的行为 0，读列的数值。如果读得的列值为全 1，说明无键按下，否则说明有键按下。

② 判断被按键的具体位置。

采用细扫描的办法。逐行输出 0，读列的数值。如果读得的列值为全 1，说明被按键不在该行上，再让下一行为 0；否则说明被按键在该行上。

③ 计算被按键的键值，以确定要完成的功能。

采用某种算法，将行和列的信息合并为一个信息，该信息称为该键的键值，并按一定的顺序形成一个键值表。

在计算键值时应注意所有按键的键值应采用同一种算法并且计算出来的键值应该各不相同。

行列式键盘软件管理流程图如图 3-17 所示。

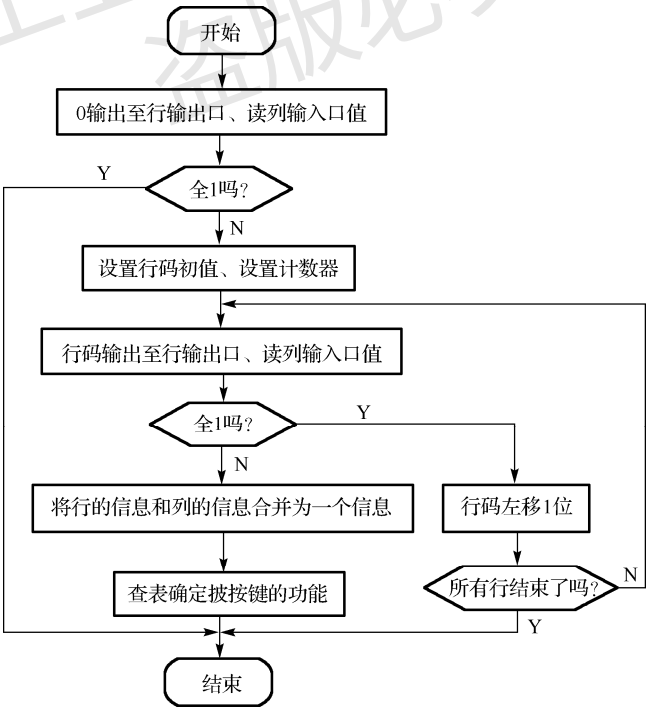


图 3-17　行列式键盘软件管理流程图

【例 3-11】 4×4 矩阵键盘示例。

电路如图 3-18 所示，P1.4～P1.7 为列输入线，P1.0～P1.3 为行输出线。P2 口接一个共阳极 LED 显示器，编程显示按键的号码为 0～F。

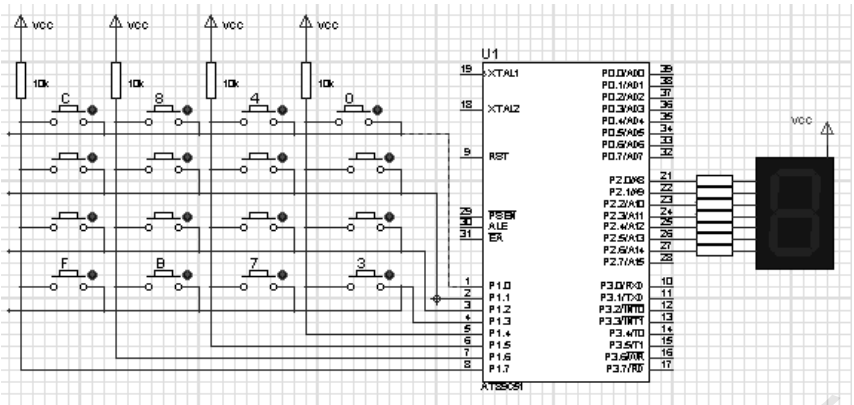


图 3-18　例 3-11 的电路原理图

C51 程序代码如下：

```
#include<reg51.h>
#define  uchar  unsigned  char
#define  uint  unsigned  int
uchar  data  dir_buf;                          //显示缓冲区
uchar  key;                                    //计算所得键值
uchar  i2;                                     //键号
code uchar dirtab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,
            0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e,0xbf};   //显示的代码表
code uchar keytab[]={0xee,0xed,0xeb,0xe7,0xde,0xdd,0xdb,0xd7,
               0xbe,0xbd,0xbb,0xb7,0x7e,0x7d,0x7b,0x77};     //键值表
void  delay(uint) ;                            //延时函数
void  keyscan( ) ;                             //键盘扫描函数
void  dir( ) ;                                 //显示函数
void  main( )
{   i2=16;
    dir_buf=16;                                //调用显示-
    while(1)
    {   keyscan( );                            //调用键扫描函数
        dir_buf=i2;                            //显示-
        dir( ) ;
  }
}
void  dir( )                                   //显示函数
{   P2=dirtab[dir_buf] ;
}
void  delay(uint  xms)                         //延时 xms
```

```
{    uint  t1, t2;
    for( t1=xms; t1>0; t1--)
    for( t2=110; t2>0; t2--) ;
}
void  keyscan( )                          //键扫描
{   uchar  code_h, code_l,  i1;           //code_h 为行输出值，code_l 为列输入值
    P1=0xf0;                              //所有的行输出 0
    code_l=P1;                            //读列值
    code_l=code_l&0xf0;                   //屏蔽掉低 4 位
    if(code_l!=0xf0)                      //如果有键按下
   {    delay(6);                         //延时，消抖动，再读
        code_l=P1;
        code_l=code_l&0xf0;               //屏蔽掉低 4 位
        if(code_l!=0xf0)                  //有键按下吗?
        {   code_h=0xfe;
            for(i1=0; i1<4; i1++)
            {   P1=code_h;
                code_l=P1;
                code_l=code_l&0xf0;
                if(code_l==0xf0)
                {   code_h=(code_h<<1)|0x01;
                }
                else  break;
            }
            P1=0xff;                      //等待键抬起
            while(P1!=0XFF);
            code_h=code_h&0x0f;           //行的信息屏蔽掉高 4 位
            key=code_h+code_l;   //得到计算键值（列值在高 4 位上，行值在低 4 位上）
            for(i2=0; i2<16; i2++)        //i2 是键号
            {    if(key==keytab[i2])
                {   break;
                }
            }
        }
    }
}
```

在键盘的软件管理过程中，比较关键的是采用某种算法来计算键值。当键盘的结构在行和列的数量之和小于等于 8 时，算法比较简单：把行的信息放在高位（或低位），列的信息放在低位（或高位），二者组成 1 字节就可以了。当按键的数量比较多时，一种通用的算法是：将行的信息转变为行号（在 0000～1111 之间），将列的信息转变为列号（在 0000～1111 之间），这样就可以将行号作为高 4 位（或低 4 位），列号作为低 4 位（或高 4 位），二者组成 1 字节。这种方法管理的键盘结构可达到 16×16，具体的程序读者可自行编写。

强调：在按键的数量不超过 8 个时，采用并行接口按键结构，当超过 8 个时，一般采用矩阵键盘结构形式。

3.4　实验与设计

▶▶ 实验 1　闸刀型开关输入/8 段 LED 静态显示输出

【实验目的】掌握 51 单片机片内并行 I/O 口的输入/输出的基本操作；掌握闸刀型开关输入信号的程序管理；掌握 8 段 LED 显示器的程序管理。

【实验电路与内容】电路如图 3-19 所示，P1.0 接一闸刀型开关 K0，P3 口接一共阳极显示器，编程实现 K0 闭合时显示“H”，K0 断开时显示“F”。

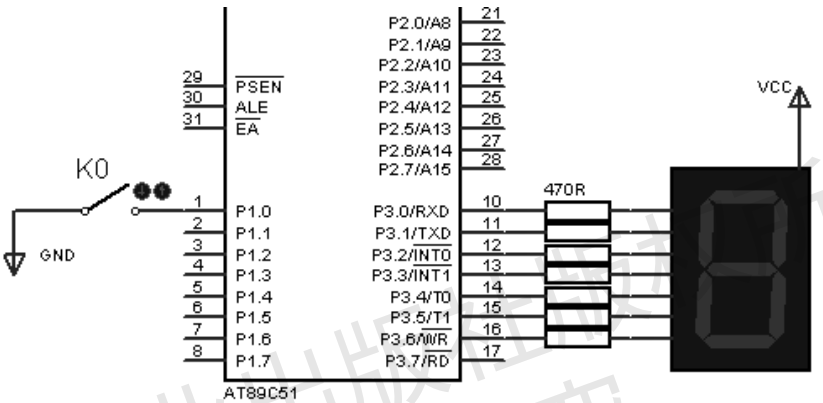


图 3-19　P0～P3 口输入/输出实验 1 电路

【参考程序】

```
#include<reg51.h>
sbit K0=P1^0;                                  //定义 K0
unsigned char data dir_buf;                    //显示缓冲区
code unsigned char dirtab[]={0x8E,0x89}; //F 与 H 的显示代码
void dir()                                     //显示函数
{   P3=dirtab[dir_buf];
}
void main( )
{   while(1)
    {   K0=1;
        if(K0==0)  dir_buf=1;                  //K0 闭合
        else  dir_buf=0;
        dir();
    }
}
```

▶▶ 实验 2　按钮型开关输入/8 段 LED 静态显示输出

【实验目的】掌握 51 单片机片内并行 I/O 口的输入/输出的基本操作；掌握按钮型开关输入信号的程序管理；掌握 8 段 LED 显示器的程序管理。

【实验电路与内容】电路如图 3-20 所示，P3.2 和 P3.3 接两个按钮开关 K0、K1，P1 口、P2 口接了两个共阴极 LED 显示器，编程实现：开始实现数字 50，定义 K0 和 K1 分别为+1 和–1 键，按 K0 显示的数字是+1；按 K1 显示的数字是–1。K0 和 K1 的管理采用查询的方式。

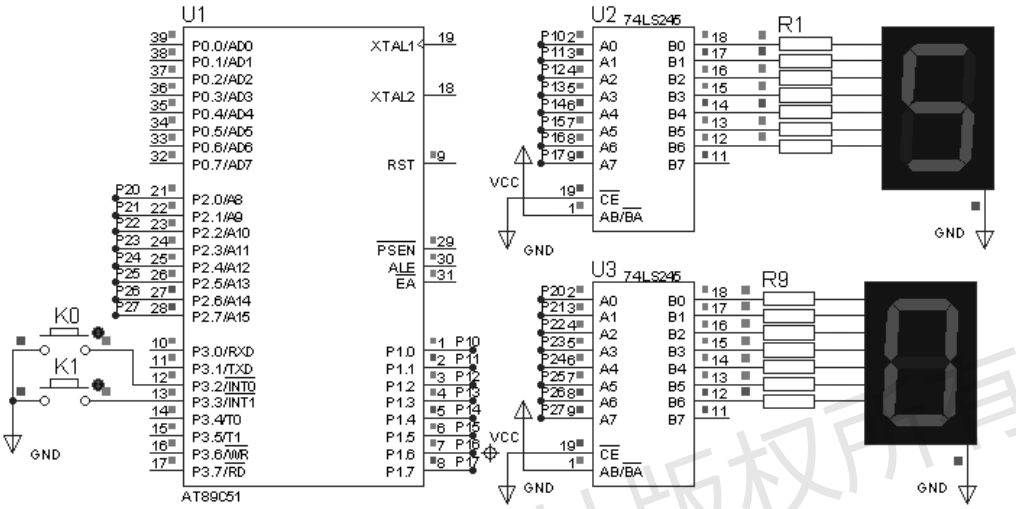


图 3-20　P0～P3 口输入/输出实验 2 电路

【参考程序】

```
#include<reg51.h>
#define uchar unsigned char
#define uint unsigned int
sbit K0=P3^2;
sbit K1=P3^3;
uchar data i;                                   //定义一个 RAM 单元
uchar data dir_buf[2];                          //显示缓冲区
uchar code dirtable[18]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
                                                //显示的代码表
void delay(uint xms)                            //延时 xms 毫秒
{   uint t1,t2;
    for(t1=0;t1<xms;t1++)
    for(t2=0;t2<120;t2++);
}
void display( )                                 //显示函数
{   P1=dirtable[dir_buf[0]];                    //P1 口显示
    P2=dirtable[dir_buf[1]];                    //P2 口显示
}
void key()                                      //按键管理函数
{   uchar key1;
    K0=1;K1=1;                                  //读按键
    key1=P3;
    key1=key1&0x0c;                             //屏蔽掉无用位
    while(key1!=0x0c)                           //有键按下吗?
```

```
        {   delay(6);                          //消抖动
            key1=P3;                           //再读
            key1=key1&0x0c;
            while(key1!=0x0c)
            {   if(K0==0)   i=i+1;             //加 1 键吗?
                else i=i-1;
                K0=1;K1=1;                     //判断键是否抬起
                key1=P3;
                key1=key1&0x0c;
                while(key1!=0x0c)
                {   key1=P3;
                    key1=key1&0x0c;
                }
            }
    }
    }
    void main()
    {   i=50;
        while(1)
        {   key();                             //调用键管理函数
            dir_buf[0]=i/10;                   //十位
            dir_buf[1]=i%10;                   //个位
            display();
        }
    }
```

▶▶ 设计 1：计时秒表的设计

① 两位 LED 显示，可以显示 00～99 秒。

② 两个按键，分别为启动/停止键、清 0 键。

要求：功能描述、硬件电路设计、软件程序设计（时间由软件延时）、总结等。

▶▶ 设计 2：模拟交通信号灯控制装置的设计

① 6 个发光二极管模拟交通灯。

南北：黄、红、绿；东西：黄、红、绿。

② 两个应急开关：南北绿东西红或东西绿南北红。

要求：功能描述、硬件电路设计、软件程序设计（延时由软件形成）、总结等。

本章小结

51 单片机有 4 个并行的 I/O 口 P0～P3 口，P0、P2、P3 口除具有基本的输入/输出功能外，还具有第二功能。本章主要介绍其作为输入/输出口的基本应用。共包含了以下 4 部分内容。

（1）51 单片机 P0～P3 口的基本知识

主要从应用的角度介绍 P0～P3 口的基本结构、基本操作、特点，目的是使读者对其结构与特点

有个基本的了解，为后续的应用打下基础。

（2）输出操作

在输出操作过程中，就是在相应的引脚输出高、低电平。目的是通过相应的例子使读者掌握字节输出、位输出的基本操作。

（3）输入操作

在输入操作时，就是在相应的引脚输入高、低电平，该高、低信号可以由两种开关来模拟：闸刀型开关输入、按钮型开关输入。目的是通过相应的例子，使读者掌握闸刀型输入信号和按钮型输入信号的“读”操作。应该注意是在读输入信号时，一定要先将相应的引脚置高电平，然后再读该引脚的状态。

（4）实验与设计题

通过两个实验操作和两个设计题使读者对 P0～P3 口的基本操作有一个更好的掌握。实验 1 的目的是练习闸刀型开关输入的操作；实验 2 的目的是练习按钮型开关输入的操作；设计 1 考核的目标是并行口的输入及输出设计及软件查表程序的设计；设计 2 的考核目标是并行口的输入及输出设计、位操作等知识。

本章主要是 P0～P3 口的输入/输出操作，要求在了解其基本结构的基础上掌握其输入/输出操作。

习题

1．P0～P3 除具有一般输入/输出端口的功能外，P0、P2、P3 引脚还有什么其他功能？

2．在 51 系列单片机的输入/输出端口中，哪个输入/输出端口执行输出功能时没有内部上拉电阻？

3．在 51 系列单片机中，若输入/输出端口执行输入操作时，为何要先送“1”到该输入端口？

4．试编写一个延迟 1s 的延迟函数。

5．开关抖动现象如何处理？

6．简述 51 单片机的 P0～P3 口各有什么特点，以 P1 口为例说明准双向 I/O 端口的意义。