第5章 单片机的开关检测、键盘输入与显示的接口设计

5.1 单片机控制发光二极管显示

发光二极管常用来指示系统工作状态，制作节日彩灯、广告牌匾等。

大部分发光二极管工作电流1~5mA之间，其内阻为20~100Ω。电流越大，亮度也越高。

为保证发光二极管正常工作，同时减少功耗，限流电阻选择十分重要，若供电电压为+5V，则限流电阻可选1~3kΩ。

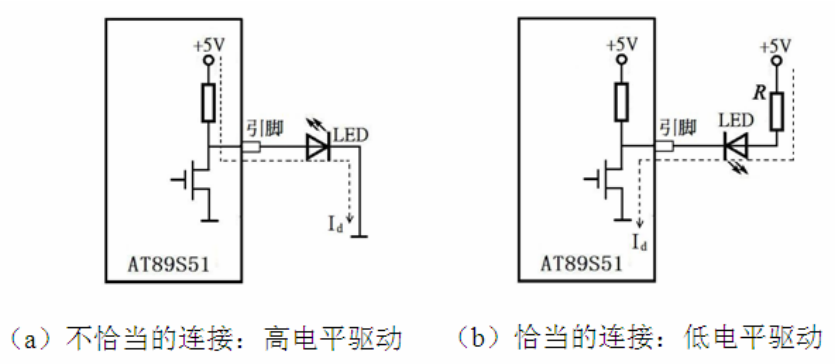
5.1.1 单片机与发光二极管的连接

第2章已介绍，P0口作通用I/O用，由于漏极开路，需外接上拉电阻。而P1～P3口内部有30kΩ左右上拉电阻。

下面讨论P1～P3口如何与LED发光二极管驱动连接问题。

单片机并行端口P1～P3直接驱动发光二极管，电路见图5-1。

与P1、P2、P3口相比，P0口每位可驱动8个LSTTL输入，而P1～P3口每一位驱动能力，只有P0口一半。



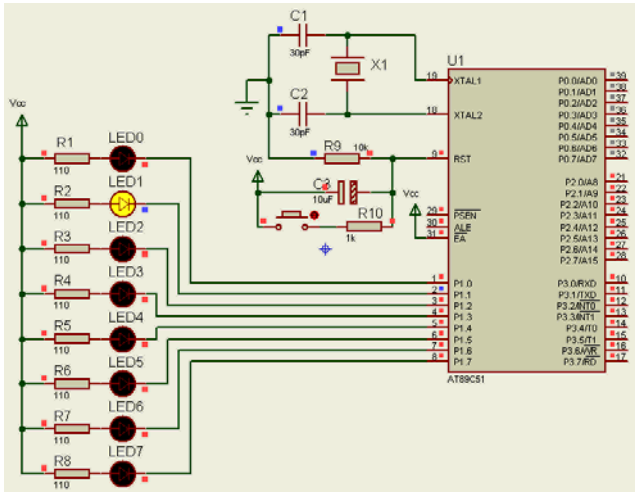
（a）不恰当的连接：高电平驱动　　（b）恰当的连接：低电平驱动

图5-1 发光二极管与单片机并行口的连接

5.1.2 I/O端口的编程举例

对I/O端口编程控制时，要对I/O端口特殊功能寄存器声明，在C51的编译器中，这项声明包含在头文件reg51.h中，编程时，可通过预处理命令#include<reg51.h>，把这个头文件包含进去。下面通过案例介绍如何编程对发光二极管输出控制。

【例5-1】 制作流水灯，原理电路见图5-2，8个发光二极管LED0～LED7经限流电阻分别接至P1口的P1.0～P1.7引脚上，阳极共同接高电平。编写程序来控制发光二极管由上至下的反复循环流水点亮，每次点亮一个发光二极管。



参考程序：

```
#include <reg51.h>
#include <intrins.h>              //包含移位函数_crol_( )的头文件
#define uchar unsigned char
#define uint unsigned int
void  delay(uint i)                       //延时函数
{
        uchar t;
        while (i--)
        {
                for(t=0;t<120;t++);
        }
}
```

```
void  main( )                                  //主程序
{
    P1=0xfe;                                   //向P1口送出点亮数据
    while (1)
    {
        delay( 500 );          //500为延时参数，可根据实际需要调整
        P1=_crol_(P1,1) ;  // 函数_crol_(P1,1)把P1中的数据循环左移1位
    }
}
```


程序说明：

（1）while(1) 两种用法：

“while(1);”： while(1)后有分号，是使程序停留在这指令上；

“while(1) {.....;}”：反复循环执行大括号内程序段，本例用法，即控制流水灯反复循环显示。

（2）C51函数库中的循环移位函数：循环移位函数包括：

- 循环左移函数“_crol_”
- 循环右移函数“_cror_”。

本例用循环左移 “_crol_(P1,1)”，函数。括号第1个参数为循环左移对象，即对P1中的内容循环左移；第2个参数为左移位数，即左移1位。编程中一定要把含有移位函数的头文件intrins.h包含在内，例如第2行“#include <intrins.h>”。

在【例5-1】基础上，编写控制发光二极管反复循环点亮的流水灯。

【例5-2】电路见图5-2，制作由上至下再由下至上反复循环点亮显示的流水灯，3种方法实现。

（1）数组的字节操作实现

建立1个字符型数组，将控制8个LED显示的8位数据作为数组元素，依次送P1口。参考程序：

```
#include <reg51.h>
#define uchar unsigned char
uchar tab[ ]={ 0xfe , 0xfd , 0xfb , 0xf7 , 0xef , 0xdf , 0xbf , 0x7f , 0x7f , 0xbf ,
0xdf , 0xef , 0xf7 ,  0xfb , 0xfd , 0xfe };
                /*前8个数据为左移点亮 数据，后8个为右移点亮数据*/
```

```
void  delay( )
{
        uchar i,j;
        for(i=0; i<255; i++)
        for(j=0; j<255; j++);
}
void  main( )                                  //主函数
{
        uchar i;
        while (1)
        {
                for(i=0;i<16; i++)
                {
                        P1=tab[i];          //向P1口送出点亮数据
                        delay( );           //延时，即点亮一段时间
                }
        }
}
```

（2）移位运算符实现

使用移位运算符“>>”、“<<”，把送P1口显示控制数据进行移位，从而实现发光二极管依次点亮。参考程序：

```
#include <reg51.h>
#define uchar unsigned char
void  delay( )
{
        uchar i,j;
        for(i=0; i<255; i++)
        for(j=0; j<255; j++);
}
void  main( )                          //主函数
{
        uchar i,temp;
        while (1)
```



```
{
        temp=0x01;                     //左移初值赋给temp
        for(i=0; i<8; i++)
        {
          P1=~temp;                    // temp中的数据取反后送P1口
           delay( );                   // 延时
           temp=temp<<1;               // temp 中数据左移一位
        }
        temp=0x80;                     // 赋右移初值给temp
        for(i=0; i<8; i++)
        {
          P1=~temp;                    // temp中的数据取反后送P1口
           delay( );                   // 延时
           temp=temp>>1;               // temp 中数据右移一位
        }
    }
}
```


程序说明： 注意使用移位运算符“>>”、“<<”与使用循环左移函数“_crol_”和循环右移函数“_cror_” 区别。左移移位运算“<<”是将高位丢弃，低位补0 ；右移移位运算、“>>”是将低位丢弃，高位补0。而循环左移函数“_crol_” 是将移出的高位再补到低位，即循环移位；同理循环右移函数“_cror_” 是将移出的低位再补到高位。

（3）用循环左、右移位函数实现

使用C51提供的库函数，即循环左移*n*位函数和循环右移*n*位函数，控制发光二极管点亮。参考程序：

```
#include <reg51.h>
#include <intrins.h>                   //包含循环左、右移位函数的头文件
#define uchar unsigned char
```



```
void  delay( )
{
    uchar i,j;
    for(i=0; i<255; i++)
    for(j=0; j<255; j++);
}
void  main( )                          // 主函数
{
uchar i,temp;
    while (1)
    {
        temp=0xfe;                     // 初值为11111110
        for(i=0; i<7; i++)
```

```
    {
        P1=temp;                    // temp中的点亮数据送P1口，控制点亮显示
        delay( );                                   // 延时
        temp=_crol_( temp,1) ;          //  temp 数据循环左移1位
    }
for(i=0; i<7; i++)
    {
    P1=temp;                                    // temp中的数据送P1口输出
    delay( );                                   // 延时
    temp=_cror_( temp,1) ;                      //temp中数据循环右移1位
    }
  }
}
```


5.2 开关状态检测

读入I/O端口电平，即可检测开关处于闭合状态还是打开状态。

5.2.1 开关检测案例1

用I/O端口来进行开关状态检测，开关一端接到I/O端口引脚上，并通过上拉电阻接+5V上，开关另一端接地，当开关打开时，I/O引脚为高电平，当开关闭合时，I/O引脚为低电平。

【例5-3】 如图5-3，单片机的P1.4～P1.7接4个开关S0～S3，P1.0～P1.3接4个发光二极管LED0～LED3。

编程将P1.4～P1.7上的4个开关状态反映在P1.0～P1.3引脚控制的4个发光二极管上，开关闭合，对应发光二极管点亮。例如P1.4引脚上开关S0状态，由P1.0脚上LED0显示，P1.6引脚上开关S2状态，由P1.2脚的LED2显示。

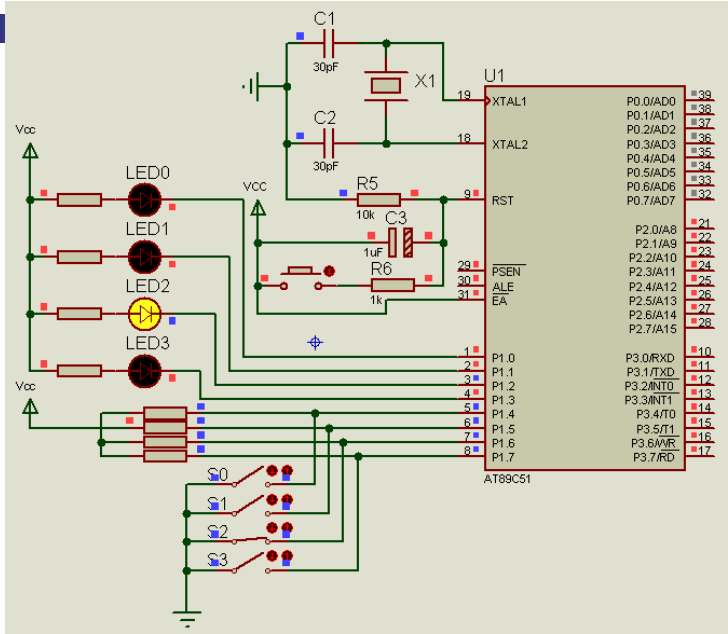


图5-3　开关、LED发光二极管与P1口的连接

参考程序如下：

```
#include <reg51.h>
#define uchar unsigned char
void  delay( )                                    //延时函数
{
        uchar i,j;
        for(i=0; i<255; i++)
        for(j=0; j<255; j++);
}
void  main( )                                     //主函数
{
while (1)
    {
        unsigned char temp;     //定义临时变量temp
        P1=0xff;      //P1口低4位置1，作为输入;高4位置1，发光二极管熄灭
        temp=P1&0xf0;           //读P1口并屏蔽低4位，送入temp 中
        temp=temp>>4;           //temp内容右移4位，P1口高4位移至低4位
                P1=temp;                // temp中的数据送P1口输出
        delay( );
 }
```


5.2.2 开关检测案例2

【例5-4】 如图5-4，P1.0和P1.1引脚接有两只开关S0和S1，两引脚上的高低电平共4种组合，4种组合分别点亮P2.0～P2.3引脚控制的4只LED，即S0、S1均闭合，LED0亮，其余灭；S1闭合、S0打开，LED1亮，其余灭；S0闭合、S1打开，LED2亮，其余灭；S0、S1均打开，LED3亮，其余灭。编程实现此功能。

参考程序：

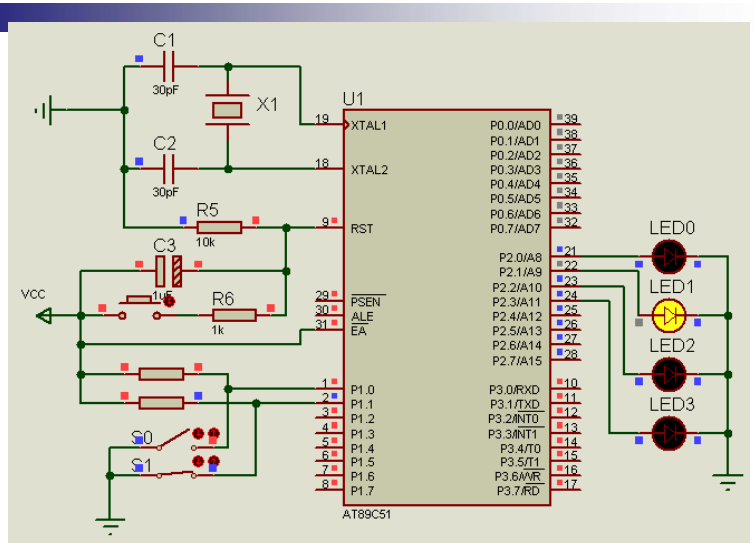


图5-4　开关检测指示器2接口电路与仿真

```
#include <reg51.h>                          // 包含头文件reg51.h
void  main( )                               //主函数main( )
{
    char state;
    do
    {
        P1=0xff;                            // P1口为输入
        state=P1;                           // 读入P1口的状态，送入state
        state=state&0x03;                   // 屏蔽P1口的高6位
        switch (state)                      // 判P1口低2位开关状态
        {
            case 0: P2=0x01; break;// P1.1、P1.0=00，点亮P2.0脚LED
            case 1: P2=0x02; break;// P1.1、P1.0=01，点亮P2.1脚LED
            case 2: P2=0x04; break;// P1.1、P1.0=10，点亮P2.2脚LED
            case 3: P2=0x08; break;// P1.1、P1.0=11，点亮P2.3脚LED
        }
    }while ( 1 );
}
```

程序段中用到循环结构控制语句do-while以及switch-case语句。

5.3 单片机控制LED数码管的显示

5.3.1 LED数码管显示原理

LED数码管：“8”字型，7段（不包括小数点）或8段（包括小数点），每段对应一个发光二极管，共阳极和共阴极两种，见图5-5。共阳极数码管的阳极连接在一起，接+5V；共阴极数码管阴极连在一起接地。

对于共阴极数码管，当某发光二极管阳极为高电平时，发光二极管点亮，相应段被显示。同样，共阳极数码管阳极连在一起，公共阳极接+5V，当某个发光二极管阴极接低电平时，该发光二极管被点亮，相应段被显示。

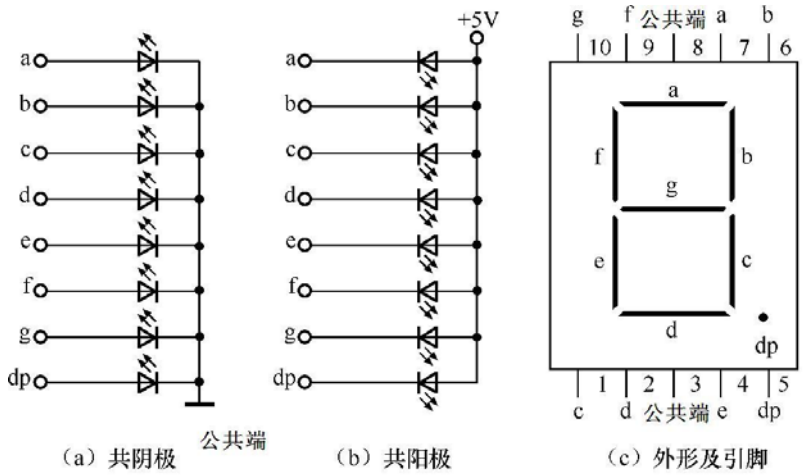
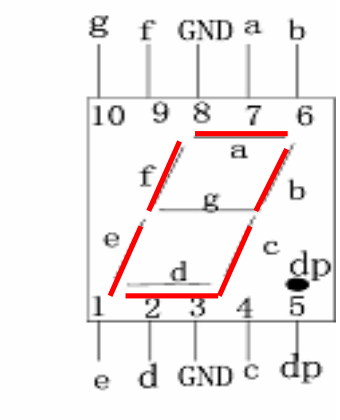


图5-5 8段LED数码管结构及外形



显示0 **a, b, c, d, e, f被点亮**

共阴极段选码(高电平点亮)：3FH

Dp	g	f	e	d	c	b	a
0	0	1	1	1	1	1	1

共阳极段选码(低电平点亮):C0H

Dp	g	f	e	d	c	b	a
1	1	0	0	0	0	0	0

为使LED数码管显示不同字符，要把某些段点亮，就要为数码管各段提供一字节的二进制码，即字型码（也称段码）。习惯上以“a”段对应字型码字节的最低位。各字符段码见表5-1。

表 5-1　　LED 数码管的字型码

显示字符	共阴极字型码	共阳极字型码	显示字符	共阴极字型码	共阳极字型码
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	T	31H	CEH
7	07H	F8H	y	6EH	91H
8	7FH	80H	H	76H	89H
9	6FH	90H	L	38H	C7H
A	77H	88H	“灭”	00H	FFH
b	7CH	83H	…	…	…

如要在数码管显示某字符，只需将该字符字型码加到各段上即可。

例如某存储单元中的数为“02H”，想在共阳极数码管上显示“2”，需要把“2”的字型码“A4H”加到数码管各段。将欲显示字符的字型码作成一个表（数组），根据显示字符从表中查找到相应字型码，然后把该字型码输出数码管各个段上，同时数码管的公共端接+5V，此时在数码管上显示字符“2”。

下面介绍单片机如何控制LED数码管显示字符。

【例5-5】利用单片机控制一个8段LED数码管先循环显示单个偶数：0、2、4、6、8，再显示单个奇数：1、3、5、7、9，如此反复循环显示。

本例原理电路及仿真结果，见图5-6。

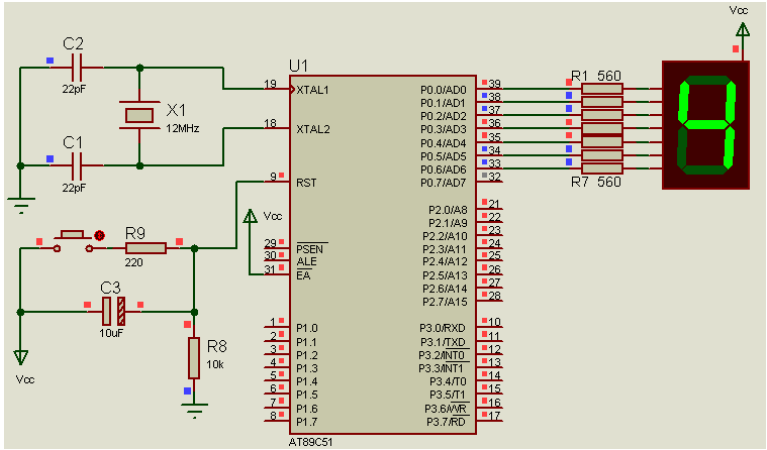


图5-6　　控制数码管循环显示单个数字的电路及仿真

参考程序如下：

```
#include "reg51.h"
#include "intrins.h"
#define uchar unsigned char
#define uint unsigned int
#define out P0
uchar code seg[]= {0xc0,0xa4,0x99,
0x82,0x80,0xf9,0xb0,0x92,0xf8,0x90,0x01};  / /共阳极段码表
void delayms(uint);
void main(void)
{
uchar i;
while(1)
        {
        out=seg[i];
        delayms(900);
        i++;
        if(seg[i]==0x01)i=0;   // 如段码为0x01，表明一个循环显示已结束
        }
}
```



```
void delayms(uint j)                                    // 延时函数
{
uchar i;
for(;j>0;j--)
        {
                i=250;
                while(--i);
                i=249;
                while(--i);
        }
}
```

说明：语句“if(seg[i]==0x01)i=0; ”含义：如果欲送出的数组元素为0x01（数字“9”段码0x90的下一个元素，即结束码），表明一个循环显示已结束，则i=0，则重新开始循环显示，从段码数组表的第一个元素seg[0]，即段码0xc0（数字0）重新开始显示。

5.3.2 LED数码管的静态显示与动态显示

两种显示方式：静态显示和动态显示。

1. 静态显示方式

无论多少位LED数码管，都同时处于显示状态。

多位LED数码管工作于静态显示方式时，各位共阴极（或共阳极）连接在一起并接地（或接+5V）；每位数码管段码线（a～dp）分别与一个8位I/O口锁存器输出相连。如果送往各个LED数码管所显示字符的段码一经确定，则相应I/O口锁存器

锁存的段码输出将维持不变，直到送入下一个显示字符段码。静态显示方式显示无闪烁，亮度较高，软件控制较易。

图5-7为4位LED数码管静态显示电路，各数码管可独立显示，只要向控制各位I/O口锁存器送相应显示段码，该位就能保持相应的显示字符。

这样在同一时间，每一位显示的字符可各不相同。静态显示方式占用I/O口端口线较多。图5-7电路，要占用4个8位I/O口（或锁存器）。如数码管数目增多，则需增加I/O口数目。

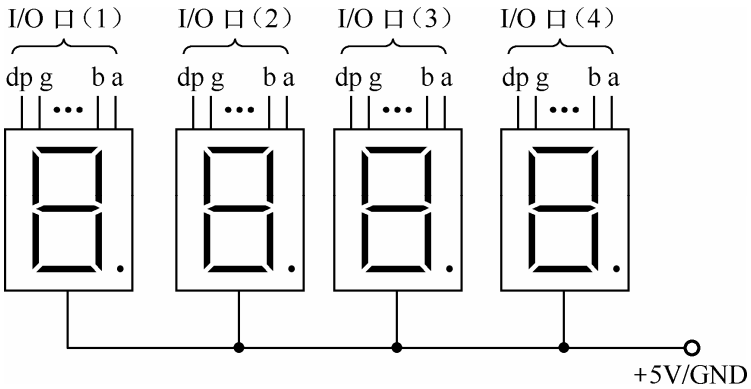


图5-7　4位LED静态显示的示意图

【例5-6】单片机控制2只数码管，静态显示2个数字“27”。 原理电路见图5-8。

单片机用P0口与P1口，分别控制加到两个数码管DS0与DS1的段码，而共阳极数码管DS0与DS1的公共端（公共阳极端）直接接至+5V，因此数码管DS0与DS1始终处于导通状态。利用P0口与P1口带有的锁存功能，只需向单片机P0口与P1口分别写入相应的显示字符“2”和“7”的段码即可。

由于一个数码管就占用一个I/O端口。如果数码管数目增多，则需增加I/O口，但软件编程要简单的多。

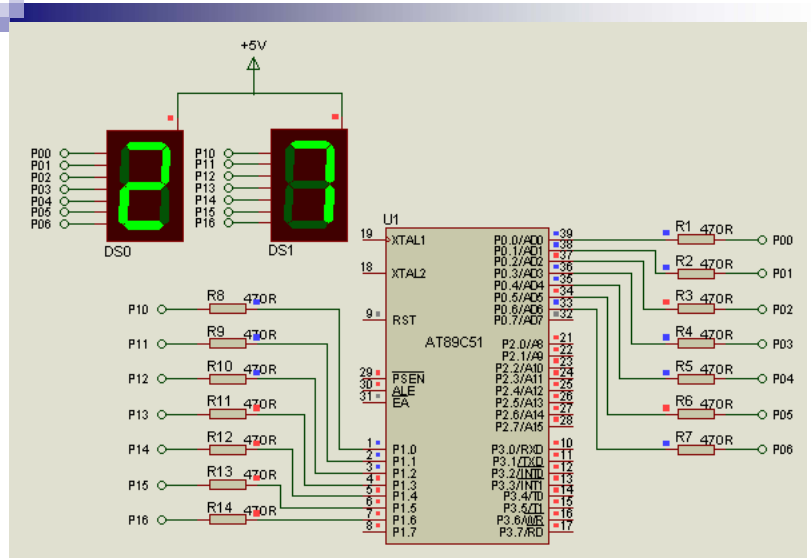


图5-8　2位数码管静态显示的原理电路与仿真

参考程序如下：

```
#include<reg51.h>        //包含8051单片机寄存器定义的头文件
void main(void)
{
   P0=0xa4;              //将数字"2"的段码送P0口
   P1=0xf8;              //将数字"7"的段码送P1口
   while(1)              //无限循环
   ;
}
```

2. 动态显示方式

显示位数较多时，静态显示所占的I/O口多，这时常采用动态显示。为节省I/O口，通常将所有显示器段码线相应段并联在一起，由一个8位I/O口控制，各显示位公共端分别由另一单独I/O口线控制。

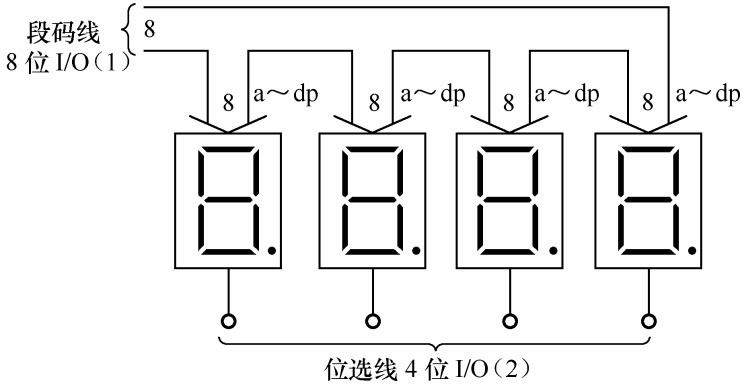


图5-9　4位LED数码管动态显示示意图

【例5-7】 8只数码管，分别滚动显示单个数字1～8。程序运行后，单片机控制左边第1个数码管显示1，其他不显示，延时之后，控制左边第2个数码管显示1，其他不显示，直至第8个数码管显示8，其他不显示，反复循环上述过程。

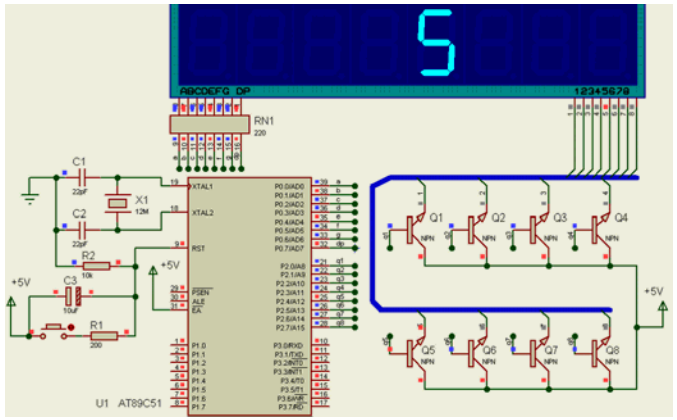


图5-10　8只数码管分别滚动显示单个数字1～8

参考程序如下：

```
#include<reg51.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int
uchar code
    dis_code[]={0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0xc0};
                                        //共阳数码管段码表
void  delay(uint t)                                  //延时函数
{
    uchar i;
    while(t--) for(i=0;i<200;i++);
}
```



```
void  main()
{
        uchar i,j=0x80;
    while(1)
    {
        for(i=0;i<8;i++)
        {
            j=_crol_(j,1);//_crol_(j,1)为将对象j循环左移1位
            P0=dis_code[i];        //P0口输出段码
            P2=j;                  //P2口输出位控码
            delay(180);            //延时，控制每位显示的时间
        }
     }
}
```

5.1　单片机控制发光二极管显示

5.2　开关状态检测

5.3　单片机控制LED数码管的显示

5.4　单片机控制LED点阵显示器显示

5.5　单片机控制液晶显示模块1602 LCD的显示（自学）

5.6　键盘接口设计

5.4　单片机控制LED点阵显示器显示

LED点阵显示器应用非常广泛，在许多公共场合，如商场、银行、车站、机场、医院随处可见。不仅能显示文字、图形，还能播放动画、图像、视频等信号。

LED点阵显示器分为图文显示器和视频显示器，有单色显示，还有彩色显示。下面仅介绍单片机如何来控制单色LED点阵显示器的显示。

5.4.1　LED点阵显示器结构与显示原理

由若干个发光二极管按矩阵方式排列而成。阵列点数可分为5×7、5×8、6×8、8×8点阵；按发光颜色可分为单色、双色、三色；按极性排列可分为共阴极和共阳极。

1.　LED点阵结构

以8×8LED点阵显示器为例，外形见图5-11，内部结构见图5-12，由64个发光二极管组成，且每个发光二极管是处于行线（R0~R7）和列线（C0~C7）之间交叉点上。

2.　LED点阵显示原理

显示的字符由一个个点亮的LED所构成。

由图5-12点亮点阵中一个发光二极管条件：对应行为高电平，对应列为低电平。如在很短时间内依次点亮很多个发光二极管，LED点阵就可显示一个稳定字符、数字或其他图形。控制LED点阵显示器显示，实质就是

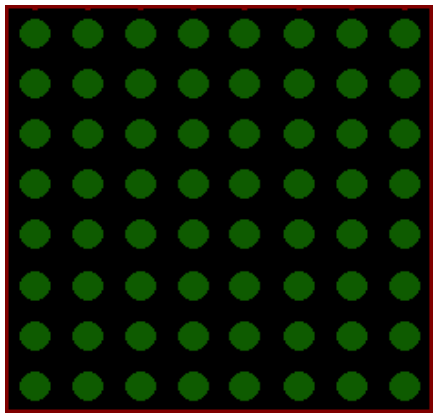


图5-11　8×8 LED点阵显示器外形

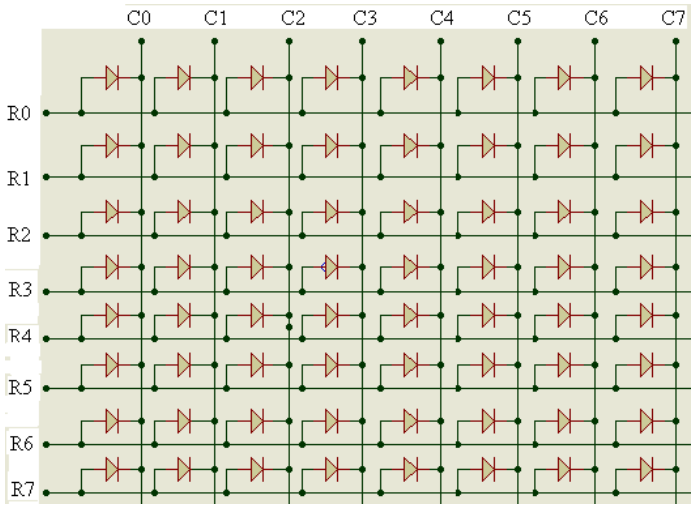


图5-12　8×8LED点阵显示器（共阴极）的结构

控制加到行线和列线上编码，控制点亮某些发光二极管（点），从而显示出由不同发光点组成的各种字符。

16×16 LED点阵显示器的结构与8×8LED点阵显示模块内部结构及显示原理是类似的，只不过行和列均为16。16×16是由4个8×8 LED点阵组成，且每个发光二极管也是放置在行线和列线的交叉点上，当对应某一列置0电平，某一行置1电平时，该发光二极管点亮。

下面以显示字符“子”为例，见图5-13。

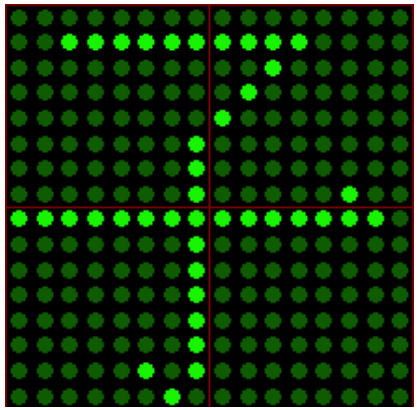


图5-13　16×16 LED点阵显示器显示字符“子”

显示过程如下：

先给LED点阵的第1行送高电平（行线高电平有效），同时给所有列线送高电平（列线低电平有效），从而第1行发光二极管全灭；

延时一段时间后，再给第2行送高电平，同时给所有列线送“1100 0000 0000 1111”，列线为0的发光二极管点亮，从而点亮10个发光二极管，显示出汉字“子”的第一横；

延时一段时间后，再给第3行送高电平，同时加到列线的编码为“1111 1111 1101 1111”，点亮1个发光二极管；

……；

延时一段时间后，再给第16行送高电平，同时给列线送“1111 1101 1111 1111”，显示出汉字“子”的最下面的一行，点亮1个发光二极管。然后再重新循环上述操作，利用人眼视觉暂留效应，一个稳定字符“子”显示出来，见图5-13。

5.4.2 控制16×16 LED点阵显示屏的案例

单片机控制16×16点阵显示屏显示字符案例。

【例5-8】如图5-14，利用单片机及 74LS154（4-16译码器）、74LS07、16×16 LED点阵显示屏来实现字符显示，编写程序，循环显示字符“电子技术”。

图中16×16 LED点阵显示屏16行行线R0~R15电平，由P1口低4位经4-16译码器74HC154的16条译码输出线L0~L15经驱动后的输出来控制。16列列线C0~C15的电平由P0口和P2口控制。剩下问题是如何确定显示字符的点阵编码，以及控制好每一屏逐行显示的扫描速度（刷新频率）。

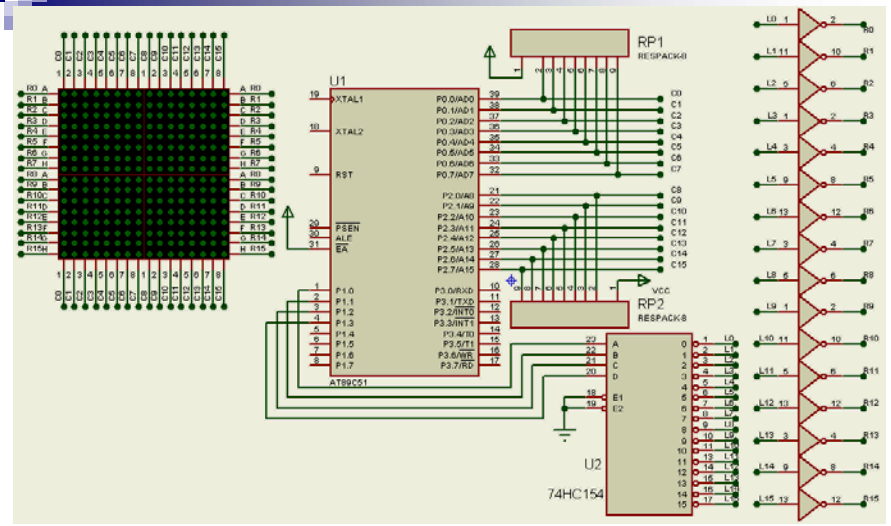


图5-14　控制16×16LED点阵显示器（共阴极）显示字符

参考程序如下：

```
#include<reg51.h>
#define uchar unsigned char
#define uint unsigned int
#define out0 P0
#define out2 P2
#define out1 P1
void delay(uint j)          //延时函数
{
    uchar i=250;
    for(;j>0;j--)
    {
        while(--i);
        i=100;
    }
}
```

```
uchar code string[]= {
//汉字“电” 16×16点阵列码
0x7F,0xFF,0x7F,0xFF,0x7F,0xFF,0x03,0xE0,0x7B,0xEF,0x7B,0xEF,0x03,0x
    E0,0x7B,0xEF,0x7B,0xEF,0x7B,0xEF,0x03,0xE0,0x7B,0xEF,0x7F,0xBF,
    0x7F,0xBF,0xFF,0x00,0xFF,0xFF
//汉字“子” 16×16点阵列码
0xFF,0xFF,0x03,0xF0,0xFF,0xFB,0xFF,0xFD,0xFF,0xFE,0x7F,0xFF,0x7F,0x
    FF,0x7F,0xDF,0x00,0x80,0x7F,0xFF,0x7F,0xFF,0x7F,0xFF,0x7F,0xFF,0x
    7F,0xFF,0x5F,0xFF,0xBF,0xFF
//汉字“技” 16×16点阵列码
0xF7,0xFB,0xF7,0xFB,0xF7,0xFB,0x40,0x80,0xF7,0xFB,0xD7,0xFB,0x67,0x
    C0,0x73,0xEF,0xF4,0xEE,0xF7,0xF6,0xF7,0xF9,0xF7,0xF9,0xF7,0xF6,0x
    77,0x8F,0x95,0xDF,0xFB,0xFF
```

```
//汉字“术”的16×16点阵的列码
0x7F,0xFF,0x7F,0xFB,0x7F,0xF7,0x7F,0xFF,0x00,0x80,0x7F,0xFF,0x3F,0xFE,0
    x5F,0xFD,
0x5F,0xFB,0x6F,0xF7,0x77,0xE7,0x7B,0x8F,0x7C,0xDF,0x7F,0xFF,0x7F,0xFF,
    0xFF,0xFF,
};
void main()
{
    uchar i,j,n;
    while(1)
    {
        for(j=0;j<4;j++)                    //共显示4个汉字
        {
```

```
for(n=0;n<40;n++)                    //每个汉字整屏扫描40次
{
        for(i=0;i<16;i++)            //逐行扫描16行
        {
        out1=i%16;                   //输出行码，
        out0=string[i*2+j*32];       //输出列码到C0~C7，逐行扫描
        out2=string[i*2+1+j*32]; //输出列码到C8~C15，逐行扫描
        delay(4);                    //显示并延时一段时间
        out0=0xff;                   //列线C0~C7为高电平，熄灭发光二极管
        out2=0xff;                   /列线C8~C15为高电平，熄灭发光二极管
        }
    }
}
```

```
    }
}
```

扫描显示时，单片机通过P1口低4位经4-16译码器74HC154的16条译码输出线L0~L15经驱动后的输出来控制，逐行为高电平，来进行扫描。由P0口与P2口控制列码的输出，从而显示出某行应点亮的发光二极管。

以显示汉字“子”为例，说明显示过程。由上面程序可看出，汉字“子”的前3行发光二级管的列码为

“0xFF,0xFF,0x03,0xF0,0xFF,0xFB,……”

第一行列码为：“ 0xff,0xff”，由P0口与P2口输出，无点亮的发光二极管。第二行列码为：“0x03,0xf0”，通过P0口与P2口输出后，由图5-13看出，0x03加到列线C7~ C0的二进制编码为“0000 0011”，这里要注意加到8个发光二极管上的对应位置。

按照图5-12和图5-14连线关系，加到从左到右发光二极管应为C0~ C7的二进制编码为“1100 0000”，即最左边的2个发光二极管不亮，其余的6个发光二极管点亮。

同理，P2口输出的0xF0加到列线C15~ C8的二进制编码为“1111 0000”，即加到C8~ C15的二进制编码为“0000 1111”，所以第二行的最右边的4个发光二极管不亮，如图5-13所示。对应通过P0口与P2口输出加到第3行16个发光二极管的列码为“0xFF,0xFB,”，对应于从左到右的C0~C15的二进制编码为“1111 1111 1101 1111”，从而第3行左边数第11个发光二极管被点亮，其余均熄灭，如图5-13所示。其余各行点亮的发光二极管，也是由16×16点阵的列码来决定。

5.1 单片机控制发光二极管显示

5.2 开关状态检测

5.3 单片机控制LED数码管的显示

5.4 单片机控制LED点阵显示器显示

5.5 单片机控制液晶显示模块1602 LCD的显示（自学）

5.6 键盘接口设计

5.6 键盘接口设计

键盘——向单片机输入数据、命令等功能，是人机对话的主要手段。

由若干按键按照一定规则组成。每一个按键实质上是一个按键开关，按构造可分为有触点开关按键和无触点按键。

有触点开关按键常见的有：触摸式键盘、薄膜键盘、导电橡胶、按键式键盘等，最常用按键式键盘。无触点开关按键有电容式按键、光电式按键和磁感应按键等。下面介绍按键式开关键盘工作原理、方式以及与键盘接口设计与软件编程。

5.6.1 键盘接口设计应解决的问题

1. 键盘的任务

任务3项。

（1）判别是否有键按下？若有，进入第（2）步。

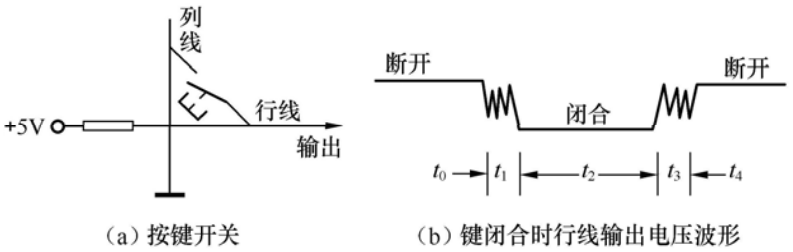
（2）识别哪一个键被按下，并求出相应的键值。

（3）根据键值，找到相应键值处理程序入口。

2. 键盘输入特点

键盘一个按键实质就是一个按钮开关。图5-20（a）所示按键开关的两端分别连接在行线和列线上，列线接地，行线通过电阻接到+5V上。键盘开关机械触点的断开、闭合，其行线电压输出波形如图5-22（b）所示。

图5-22（b）所示的t1和t3分别为键的闭合和断开过程中的抖动期（呈现一串负脉冲），抖动时间长短与开关机械特性有关，一般为5～10ms，t2为稳定的闭合期，其时间由按键动作确定，一般为十分之几秒到几秒，t0、t4为断开期。



（a）按键开关　　（b）键闭合时行线输出电压波形

图5-22　键盘开关及其行线波形

3．按键的识别

按键闭合与否，反应在行线输出电压上就是高电平或低电平，对行线电平高低状态检测，便可确认按键是否按下与松开。为了确保单片机对一次按键动作只确认一次按键有效，必须消除抖动期t1和t3的影响。

4．如何消除按键的抖动

两种去抖动方法。一种是用软件延时来消除按键抖动，基本思想：在检测到有键按下时，该键所对应的行线为低电平，执行一段延时10ms的子程序后，确认该行线电平是否仍为低电平，如果仍为低电平，则确认该行确实有键按下。当按键松开时，行线的低电平变为高电平，执行一段延时10ms的子程序后，检测该行线为高电平，说明按键确实已经松开。

采取以上措施，可消除两个抖动期t1和t3的影响。另一种去除按键抖动的方法是采用专用的键盘/显示器接口芯片，这类芯片中都有自动去抖动的硬件电路。

键盘主要分为两类：非编码键盘和编码键盘。

非编码键盘是利用按键直接与单片机相连接而成，常用在按键数量较少的场合。该类键盘，系统功能比较简单，需要处理任务较少，成本低、电路设计简单。按下键号的信息通过软件来获取。

非编码键盘常见的有：独立式键盘和矩阵式键盘两种结构。

先介绍独立式键盘接口设计。

5.6.2　独立式键盘接口设计案例

独立式键盘特点各键相互独立，每个按键各接一条I/O口线，通过检测I/O输入线的电平状态，易判断哪个按键被按下。

图5-23为一独立式键盘，8个按键k1~k8分别接到单片机的P1.0~ P1.7引脚上，图中上拉电阻保证按键未按下时，保证对应I/O口线为稳定高电平。当某一按键按下时，对应I/O口线就变成低电平，与其他按键相连的I/O口线仍为高电平。

因此，只需读入I/O口线状态，判别是否为低电平，就很容易识别出哪个键被按下。可见独立式键盘优点是电路简单，各条检测线独立，识别按键号的软件编写简单。独立式键盘适于按键数目较少场合，如按键数目较多，要占用较多I/O口线。

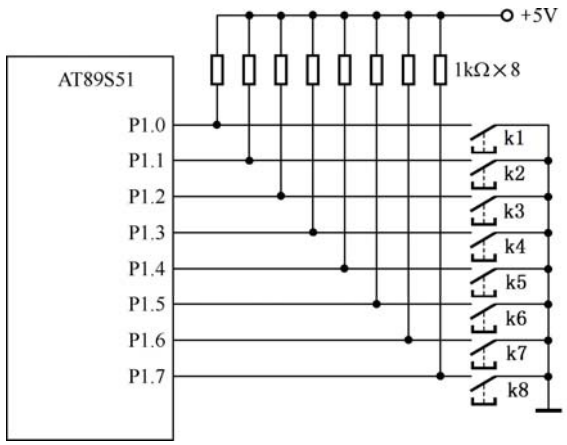


图5-23　独立式键盘的接口电路

1.独立式键盘的查询工作方式

【例5-10】对图5-23所示独立式键盘，用查询方式实现键盘扫描，根据按下不同按键，对其进行处理。扫描程序如下：

```
#include<reg51.h>
void key_scan(void)
{
        unsigned char keyval
        do
        {
                P1=0xff;                    // P1口为输入
                keyval=P1;                  //从P1口读入键盘状态
                keyval=~ keyval;//键盘状态求反
```



```
switch(keyval)
  {
      case 1: ……;          //处理按下的k1键，“……”为处理程序
      break;               //跳出switch语句
      case 2: ……;          //处理按下的k2键
      break;               //跳出switch语句
      case 4: ……;          //处理按下的k3键
      break;               //跳出switch语句
      case 8: ……;          //处理按下的k4键
      break;               //跳出switch语句
```

```
                case 16: ……;          //处理按下的k5键
                break;                //跳出switch语句
                case 32: ……;          //处理按下的k6键
                break;                //跳出switch语句
                case 64: ……;          //处理按下的k7键
                break;                //跳出switch语句
                case 128: ……;         //处理按下的k8键
                break;                //跳出switch语句
                default:
                break;                //无按下键处理
        }
    }
    while(1);
}
```

下面看用Proteus虚拟仿真独立式键盘实际案例。

【例5-11】单片机与4个独立按键k1~k4及8个LED指示灯的一个独立式键盘。4个按键接在P1.0~P1.3引脚，P3口接8个LED指示灯，控制LED指示灯亮与灭，原理电路见图5-24。

按下k1键，P3口8个LED正向（由上至下）流水点亮；

按下k2键，P3口8个LED反向（由下而上）流水点亮；

按下k3键，高、低4个LED交替点亮；

按下k4键，P3口8个LED闪烁点亮。

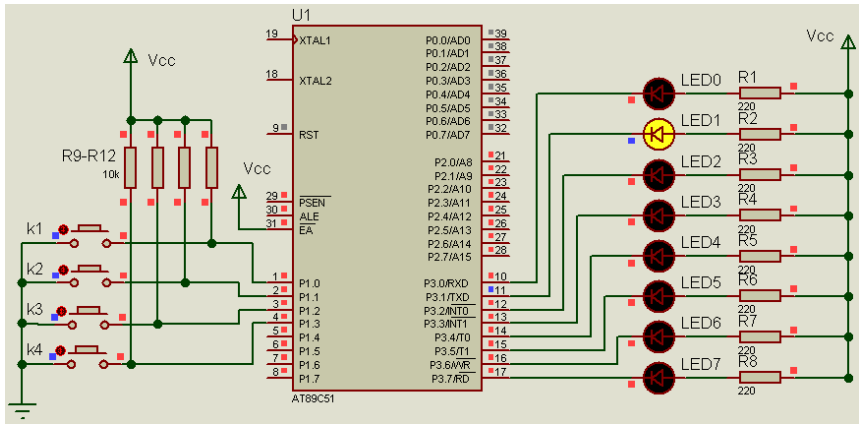


图5-24　　虚拟仿真的独立式键盘的接口电路

由于本案例中的4个按键分别对应4个不同的点亮功能，且具有不同的按键值“keyval”，具体如下：

- 按下K1键时，keyval=1
- 按下K2键时，keyval=2
- 按下K3键时，keyval=3
- 按下K4键时，keyval=4

本独立式键盘工作原理如下：

（1）首先判断是否有按键按下。将接有4个按键的P1口低4位（P1.0~P1.3）写入“1”，使P1口低4位为输入状态。然后读入低4位的电平，只要有一位不为“1”，则说明有键按下。读取方法：

```
P1=0xff;
if((P1&0x0f)!=0x0f);//读P1口低4位按键值，按位“与”运算后结果非
                    // 0x0f，表明低4位必有1位是“0”， 说明有键按下
```

（2）按键去抖动。当判别有键按下时，调用软件延时子程序，延时约10ms后再进行判别，若按键确实按下，则执行相应的按键功能，否则重新开始进行扫描。

（3）获得键值。确认有键按下时，可采用扫描方法，来判哪个键按下，并获取键值。

首先Keil μVision3建立工程，再建立源程序“*.c”文件。

参考程序：

```
#include<reg51.h>
sbit S1=P1^0;                    //将S1位定义为P1.0引脚
sbit S2=P1^1;                    //将S2位定义为P1.1引脚
sbit S3=P1^2;                    //将S3位定义为P1.2引脚
sbit S4=P1^3;                    //将S4位定义为P1.3引脚
unsigned char keyval;            //定义键值储存变量单元
void main(void)                  //主函数
{
        keyval=0;                //键值初始化为0
        while(1)
```



```
{
        key_scan();                      //调用键盘扫描函数
        switch(keyval)
        {
        case 1:forward();        //键值为1，调用正向流水点亮函数
        break;
        case 2:backward();       //键值为2，调用反向流水点亮函数
        break;
        case 3:Alter(); //键值为3，调用高、低4位交替点亮函数
        break;
        case 4:blink ();//键值为4，调用闪烁点亮函数
        break;
        }
   }
}
```



```
void key_scan(void)                  //函数功能：键盘扫描
{
   P1=0xff;
   if((P1&0x0f)!=0x0f)               //检测到有键按下
   {
        delay10ms();            //延时10ms再去检测
        if(S1==0)               //按键k1被按下
        keyval=1;
        if(S2==0)               //按键k2被按下
        keyval=2;
        if(S3==0)               //按键k3被按下
        keyval=3;
        if(S4==0)               //按键k4被按下
        keyval=4;
   }
}
```



```
void forward(void)           //函数功能：正向流水点亮LED
{
   P3=0xfe;                  //LED0亮
   led_delay();
   P3=0xfd;                  //LED1亮
   led_delay();
   P3=0xfb;                  //LED2亮
   led_delay();
   P3=0xf7;                  //LED3亮
   led_delay();
   P3=0xef;                  //LED4亮
   led_delay();
   P3=0xdf;                  //LED5亮
   led_delay();
   P3=0xbf;                  //LED6亮
   led_delay();
   P3=0x7f;                  //LED7亮
```

```
    led_delay();
}
void backward(void)          //函数：反向流水点亮LED
{
     P3=0x7f;                //LED7亮
    led_delay();
    P3=0xbf;                 //LED6亮
    led_delay();
    P3=0xdf;                 //LED5亮
    led_delay();
    P3=0xef;                 //LED4亮
    led_delay();
    P3=0xf7;                 //LED3亮
    led_delay();
    P3=0xfb;                 //LED2亮
    led_delay();
    P3=0xfd;                 //LED1亮
```



```
led_delay();
   P3=0xfe;                  //LED0亮
   led_delay();
 }
void Alter(void)             //函数：交替点亮高4位与低4位LED
{
   P3=0x0f;
   led_delay();
   P3=0xf0;
   led_delay();
}
void blink (void)            //函数：闪烁点亮LED
{
   P3=0xff;
   led_delay();
```



```
P3=0x00;
   led_delay();
}
void led_delay(void)                 //函数：延时
{
   unsigned char i,j;
   for(i=0;i<220;i++)
   for(j=0;j<220;j++)
       ;
}
void delay10ms(void)                 //函数：软件消抖延时10ms
{
   unsigned char i,j;
   for(i=0;i<100;i++)
   for(j=0;j<100;j++)
       ;
}
```


2. 独立式键盘的中断扫描方式

前面介绍查询方式独立式键盘接口设计与程序设计。为提高单片机扫描键盘的工作效率，可采用中断扫描方式，只有在键盘有键按下时，才进行扫描与处理。可见中断扫描方式的键盘实时性强，工作效率高。

【例5-12】设计一采用中断扫描方式独立式键盘，只有在键盘有按键按下时，才进行处理，接口电路见图5-25。当键盘中有键按下时，8输入与非门74LS30输出经过74LS04反相后向单片机外中断请求输入引脚INT0*发出低电平中断请求信号，单片机响应中断，进入外部中断的中断函数，在中断函数中，判断按键是否真按下。如确实按下，则把标志keyflag置1，并得到按下按键键值，然后从中断返回，根据键值跳向该键的处理程序。

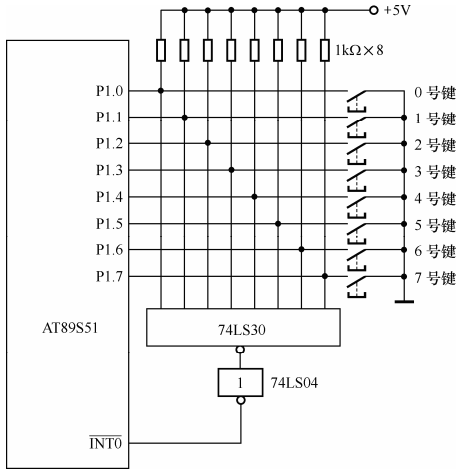


图5-25　中断扫描方式的独立式键盘的接口电路

参考程序如下：

```
#include<reg51.h>
#include<absacc.h>
#define uchar unsigned char
#define TRUE 1
#define FALSE 0
bit keyflag;                    // keyflag为按键按下的标志位
uchar keyval;                   // keyval为键值
void delay10ms(void);           //软件延时10ms函数，见例5-11
void main(void)
{
        IE=0x81;                //总中断允许EA=1，允许中断
        IP=0x01;                //设置外中断0为高优先级
        keyflag=0;              //设置按键按下标志为0
        do
        {
```



```
if(keyflag)             //如果按键按下标志keyflag =1，则有键按下
{
        keyval=~keyval;         //键值取反
        switch(keyval)          //根据按下键的键值进行分支跳转
         {
                case 1:...;             //处理0号键
                        break;
                case 2: ...;            //处理1号键
                        break;
                case 4: ...;            //处理2号键
                        break;
                case 8: ...;            //处理3号键
                        break;
                case 16: ...;           //处理4号键
                        break;
                case 32: ...;           //处理5号键
```



```
                        break;
                case 64: ...;           //处理6号键
                        break;
                case 128: ...;          //处理7号键
                        break;
                default;
                        break;          //无效按键，例如多个键同时按下
                }
                keyflag=0;              //清按键按下标志
                        }
                } while(TRUE);
    }
void int0( )  interrupt 0    //有键按下，则执行的中断函数
{
   uchar reread_key;            // reread_key为重读键值变量;
   IE=0x80;                     // 屏蔽中断
```

```
        keyflag=0;                  // 把按键按下标志keyflag清0
        P1=0xff;                    // 向P1口写1，设置P1口为输入
        keyval=P1;                  // 从P1口读入键盘的状态
        delay10ms(void);            // 延时10ms
        reread_key=P1;              // 再次从P1口读键盘状态，并存reread_key中
    if(keyval==reread_key) // 比较两次读取的键值，如相同，说明键按下
    {
        key_flag=1;                  // 按键按下标志key_flag为1
    }
    IE=0x81;                        // 重新允许中断
}
```


程序中用到了外部中断INT0*，当没有按键按下时，标志keyflag=0，程序一直执行“do{ }while（）”循环。当有键按下时，则74LS04输出端产生低电平，向单片机INT0*脚发中断请求信号，单片机响应中断，执行中断函数。

如果确实按键按下，在中断函数中把keyflag置1，并得到键值。当执行完中断函数后，再进入“do{ }while（）”循环，此时由于“if(keyflag)”中的keyflag=1，则可根据键值“keyval”，采用“switch(keyval)”分支语句，进行按下按键的处理。

5.6.3 矩阵式键盘接口设计案例

矩阵式（也称行列式）键盘用于按键数目较多的场合，由行线和列线组成，按键位于行、列交叉点上，见图5-26，一个4×4的行、列结构可以构成一个16个按键的键盘，只需要一个8位的并行I/O口即可。如果采用8×8的行、列结构，可以构成一个64按键的键盘，只需要两个并行I/O口即可。

在按键数目较多场合，矩阵式键盘要比独立式键盘节省较多I/O口线。

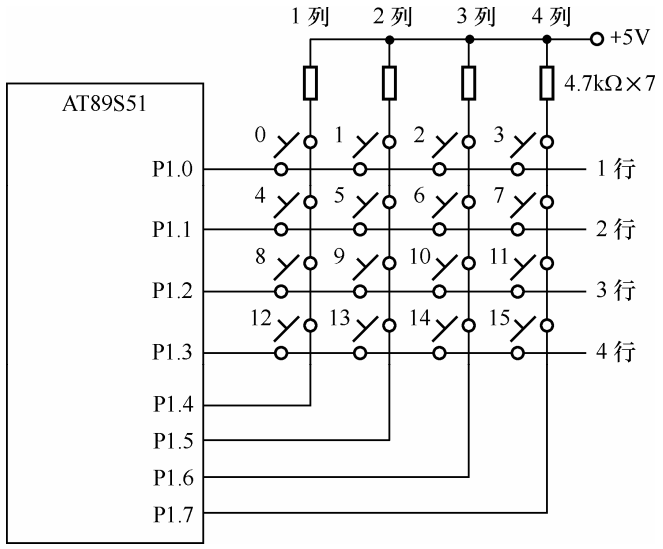


图5-26 矩阵式（行列式）键盘的接口电路

下面介绍查询方式的矩阵式键盘程序设计。

【例5-13】对图5-26矩阵式键盘，编写查询式的键盘处理程序。

先判有无键按下，即把所有行线P1.0～P1.3均置为低，然后检测各列线状态，若列线不全为高电平，则表示键盘中有键被按下；若所有列线列均为高电平，说明键盘中无键按下。

在确认有键按下后，即可查找具体闭合键位置，其方法是依次将行线置为低电平，再逐行检查各列线的电平状态。若某列为低，则该列线与行线交叉处键就是闭合键。

判断有无键按下，以及获取键值的参考程序如下：


```
#include<reg51.h>
#define uchar unsigned char
#define uint unsigned int
void main(void)
{
   uchar key;
   while(1)
   {
     key=keyscan( );     //调用键盘扫描函数，返回的键值送到变量
   key
     delay( );                //延时
   }
 void delay10ms(void);        //延时函数
{
    uchar i;
    for(i=0;i<200;i++){ }
}
```



```
uchar key_scan(void)                    //键盘扫描函数
{
   uchar code_h;                        //行扫描值
   uchar code_l;                  //列扫描值
   P1=0xf0;                       //P1.0～P1.3行线输出都为0，准备读列状态
   if((P1&f0)!=0xf0)              //如果P1.4～P1.7不全为1，可能有键按下
   {
    delay10ms(void);              //延时去抖动，参见例5-11
    if((P1&f0)!=0xf0)             //重读P1.4～P1.7，若不全为1，定有键按下
    code_h=0xfe;          // P1.0行线置为0，开始行扫描
    while((code_h&0x10)!=0xf0); //判是否扫描到最后一行，若不是继续扫描
    {
        P1=code_h;                //P1口输出行扫描值
        if((P1&f0)!=0xf0);        //如果P1.4～P1.7不全为1，该行有键按下
```



```
{
    code_l=(P1&0xf0|0x0f);          //保留P1口高4位，低4位变1，作为列值
   return((～code_h)+(～code_l)); //键扫描值=行扫描值+列扫描值，
                                   //返回主程序
   }
   else                                 //若该行无键按下，往下执行
   code_h=(code_h<<1)|0x01;     //行扫描值左移，准备扫描下一行
   }
   }
 }
        return(0) ;                //无键按下，返回0
}
```

【例5-14】 数码管显示4×4矩阵键盘键号。单片机的P1口的P1.0～P1.7连接4×4矩阵键盘，矩阵中各键编号见图5-27。

数码管显示由P0口控制，当4×4矩阵键盘中的某一按键按下时，数码管上显示对应键号。例如，1号键按下时，数码管显示“1”；E键按下时，数码管显示“E”等等。

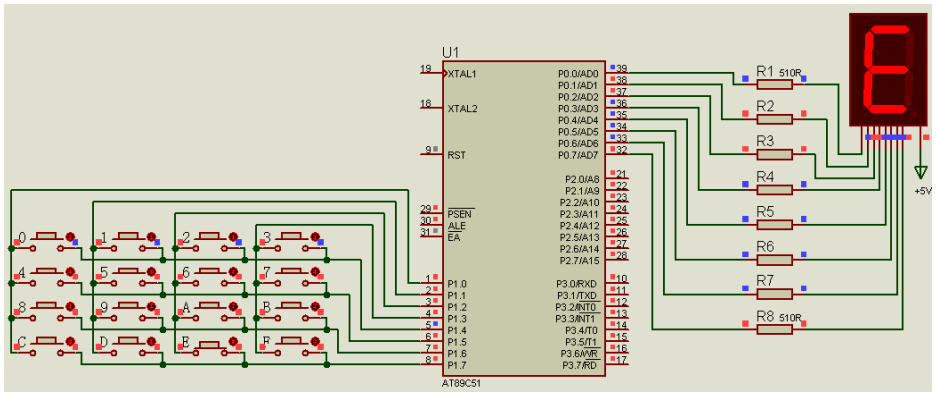


图5-27 数码管显示4×4矩阵键盘键号的原理电路

参考程序如下：

```
#include <reg51.h>
#define uchar unsigned char
sbit L1=P1^0;                    // 定义列
sbit L2=P1^1;
sbit L3=P1^2;
sbit L4=P1^3;
uchar
dis[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0
x83,0xc6,0xa1,0x86,0x8e };          //共阳极字符 0～F 的段码
unsigned int time;
delay(time)                          //延时子程序
{
unsigned int i;
       for(j=0;j<time;j++)
       {}
}
```



```
main()                                          //主程序
{
   uchar temp;
   uchar k,i;
   while(1)
   {
        P1=0xef;              //行扫描初值，P1.4=0，P1.5、P1.6、P1.7=1
        for(i=0;i<=3;i=i++)              //按行扫描,一共4行
        {
        if (L1==0) P0= dis [i*4+0];      //判第1列有无键按下,若有,键值
                                         // 可能为0,4,8.C，送显示
        if (L2==0) P0= dis [i*4+1];      //判第2列有无键按下,若有,键值
                                         //可能为1,5,9,d，送显示
        if (L3==0) P0= dis [i*4+2];      //判第3列有无键按下,若有,键值
                                         //可能为2,6,A,E，送显示
```

```
if (L4==0) P0=dis [i*4+3];        //判第4列有无按键按下,若有，键值可能
                                  //为3,7,b,F，送显示
    delay(500);
    temp11=P1;                    //读入P1口的状态
    temp=temp|0x0f;  //屏蔽掉P1.7~ P1.4
    temp=temp<<1;    //P1.3~ P1.0左移1位，准备下一行扫描
    temp=temp|0x0f;  //屏蔽掉P1.7~ P1.4
    P1=temp;              //下一行行扫描值送P1口,为下一行扫描
                          //做准备
     }
   }
}
```

程序说明：本例关键是如何获取键号，具体采用了逐行扫描。

先驱动行P1.4=0，然后依次读入各列的状态：

第1列对应i=0，

第2列对应i=1，

第3列对应i=2，

第4列对应i=3。

假设4号键按下，此时第2列对应的i=1，又L2=0，执行语句“if (L2==0) P0=dis [i*4+1]”后，i*4+1=5，从而查找到字型码数组dis[ ]中的第5个元素，即显示“4”的段码“0x99”（见表5-1），把段码“0x99”送P0口驱动数码管显示“4”。

5.6.4 非编码键盘扫描方式选择

单片机在忙于其他各项工作任务时，如何兼顾非编码键盘的输入，这取决于键盘扫描的工作方式。键盘扫描工作方式选取的原则是，既要保证及时响应按键操作，又不要过多占用单片机的执行其他任务的工作时间。

通常，键盘的扫描工作方式有3种：查询扫描、定时扫描和中断扫描。

1．查询扫描

利用单片机空闲时，调用键盘扫描子程序，反复扫描键盘，但如果单片机查询频率过高，虽能及时响应键盘输入，但也会影响其他任务的进行。如果查询频率过低，有可能出现键盘输入漏判现象。所以要根据单片机系统的繁忙程度和键盘的操作频率，来调整键盘扫描频率。

2．定时扫描

也可每隔一定的时间对键盘扫描一次，即定时扫描。这种方式中，通常利用单片机内的定时器产生的定时中断，进入中断子程序后对键盘进行扫描，在有键按下时识别出按下的键，并执行相应键的处理程序。

由于每次按键的时间一般不会小于100ms，所以为了不漏判有效按键，定时中断周期一般应小于100ms。

3．中断扫描方式

为进一步提高单片机扫描键盘工作效率，可采用中断扫描方式，即键盘只有在键盘有按键按下时，才会向单片机发出中断请求信号，单片机响应中断，执行键盘扫描中断服务子程序，识别出按下的按键，并跳向该按键的处理程序。如无键按下，单片机将不理睬键盘。该方式优点，只有按键按下时，才进行处理，所以其实时性强，工作效率高。

5.6.5　专用键盘/显示器芯片HD7279的接口设计（自学）