51 单片机 C 语言学习资料

目 录

基础知识：单片机编程基础

单片机的外部结构：

1、 DIP40 双列直插；
2、 P0，P1，P2，P3 四个 8 位准双向 I/O 引脚；（作为 I/O 输入时，要先输出高电平）
3、 电源 VCC（PIN40）和地线 GND（PIN20）；
4、 高电平复位 RESET（PIN9）；（10uF 电容接 VCC 与 RESET，即可实现上电复位）
5、 内置振荡电路，外部只要接晶体至 X1（PIN18）和 X0（PIN19）；（频率为主频的 12 倍）
6、 程序配置 EA（PIN31）接高电平 VCC；（运行单片机内部 ROM 中的程序）
7、 P3 支持第二功能：RXD、TXD、INT0、INT1、T0、T1

单片机内部 I/O 部件：(所为学习单片机，实际上就是编程控制以下 I/O 部件，完成指定任务)

1、 四个 8 位通用 I/O 端口，对应引脚 P0、P1、P2 和 P3；
2、 两个 16 位定时计数器；（TMOD，TCON，TL0，TH0，TL1，TH1）
3、 一个串行通信接口；（SCON，SBUF）
4、 一个中断控制器；（IE，IP）

针对 AT89C52 单片机，头文件 AT89x52.h 给出了 SFR 特殊功能寄存器所有端口的定义。教科书的 160 页给出了针对 MCS51 系列单片机的 C 语言扩展变量类型。

C 语言编程基础：

1、 十六进制表示字节 0x5a：二进制为 01011010B；0x6E 为 01101110。
2、 如果将一个 16 位二进数赋给一个 8 位的字节变量，则自动截断为低 8 位，而丢掉高 8 位。
3、 ++var 表示对变量 var 先增一；var—表示对变量后减一。
4、 x |= 0x0f;表示为 x = x | 0x0f;
5、 TMOD = (TMOD & 0xf0) | 0x05;表示给变量 TMOD 的低四位赋值 0x5，而不改变 TMOD 的高四位。
6、 While(1)；表示无限执行该语句，即死循环。语句后的分号表示空循环体，也就是{;}

在某引脚输出高电平的编程方法：（比如 P1.3（PIN4）引脚）

```
#include <AT89x52.h>    //该头文档中有单片机内部资源的符号化定义，其中包含 P1.3
void main( void )       //void 表示没有输入参数，也没有函数返值，这入单片机运行的复位入口
{
    P1_3 = 1;           //给 P1_3 赋值 1，引脚 P1.3 就能输出高电平 VCC
    While( 1 );         //死循环，相当 LOOP: goto LOOP;
}
```

注意：*P0 的每个引脚要输出高电平时，必须外接上拉电阻（如 4K7）至 VCC 电源。*

在某引脚输出低电平的编程方法：（比如 P2.7 引脚）

```
#include <AT89x52.h>    //该头文档中有单片机内部资源的符号化定义，其中包含 P2.7
void main( void )       //void 表示没有输入参数，也没有函数返值，这入单片机运行的复位入口
{
    P2_7 = 0;           //给 P2_7 赋值 0，引脚 P2.7 就能输出低电平 GND
```

```
    While( 1 );         //死循环，相当 LOOP: goto LOOP;
}
```

在某引脚输出方波编程方法：（比如 P3.1 引脚）

```
#include <AT89x52.h>    //该头文档中有单片机内部资源的符号化定义，其中包含 P3.1
void main( void )       //void 表示没有输入参数，也没有函数返值，这入单片机运行的复位入口
{
    While( 1 )          //非零表示真，如果为真则执行下面循环体的语句
    {
        P3_1 = 1;       //给 P3_1 赋值 1，引脚 P3.1 就能输出高电平 VCC
        P3_1 = 0;       //给 P3_1 赋值 0，引脚 P3.1 就能输出低电平 GND
    }                   //由于一直为真，所以不断输出高、低、高、低……，从而形成方波
}
```

将某引脚的输入电平取反后，从另一个引脚输出：（ 比如 P0.4 = NOT(P1.1) ）

```
#include <AT89x52.h>    //该头文档中有单片机内部资源的符号化定义，其中包含 P0.4 和 P1.1
void main( void )       //void 表示没有输入参数，也没有函数返值，这入单片机运行的复位入口
{
    P1_1 = 1;           //初始化。P1.1 作为输入，必须输出高电平
    While( 1 )          //非零表示真，如果为真则执行下面循环体的语句
    {
        if( P1_1 == 1 )         //读取 P1.1，就是认为 P1.1 为输入，如果 P1.1 输入高电平 VCC
        {   P0_4 = 0;     }     //给 P0_4 赋值 0，引脚 P0.4 就能输出低电平 GND
        else                    //否则 P1.1 输入为低电平 GND
        //{  P0_4 = 0;     }    //给 P0_4 赋值 0，引脚 P0.4 就能输出低电平 GND
        {   P0_4 = 1;     }     //给 P0_4 赋值 1，引脚 P0.4 就能输出高电平 VCC
    }                   //由于一直为真，所以不断根据 P1.1 的输入情况，改变 P0.4 的输出电平
}
```

将某端口 8 个引脚输入电平，低四位取反后，从另一个端口 8 个引脚输出：（ 比如 P2 = NOT(P3) ）

```
#include <AT89x52.h>    //该头文档中有单片机内部资源的符号化定义，其中包含 P2 和 P3
void main( void )       //void 表示没有输入参数，也没有函数返值，这入单片机运行的复位入口
{
    P3 = 0xff;          //初始化。P3 作为输入，必须输出高电平，同时给 P3 口的 8 个引脚输出高电平
    While( 1 )          //非零表示真，如果为真则执行下面循环体的语句
    {                   //取反的方法是异或 1，而不取反的方法则是异或 0
        P2 = P3^0x0f    //读取 P3，就是认为 P3 为输入，低四位异或者 1，即取反，然后输出
    }                   //由于一直为真，所以不断将 P3 取反输出到 P2
}
```

注意：一个字节的 8 位 D7、D6 至 D0，分别输出到 P3.7、P3.6 至 P3.0，比如 P3=0x0f，则 P3.7、P3.6、P3.5、P3.4 四个引脚都输出低电平，而 P3.3、P3.2、P3.1、P3.0 四个引脚都输出高电平。同样，输入一个端口 P2，即是将 P2.7、P2.6 至 P2.0，读入到一个字节的 8 位 D7、D6 至 D0。

第一节：单数码管按键显示

单片机最小系统的硬件原理接线图：

1、 接电源：VCC（PIN40）、GND（PIN20）。加接退耦电容 0.1uF

2、 接晶体：X1（PIN18）、X2（PIN19）。注意标出晶体频率（选用 12MHz），还有辅助电容 30pF

3、 接复位：RES（PIN9）。接上电复位电路，以及手动复位电路，分析复位工作原理

4、 接配置：EA（PIN31）。说明原因。

发光二极的控控制：单片机 I/O 输出

将一发光二极管 LED 的正极（阳极）接 P1.1，LED 的负极（阴极）接地 GND。只要 P1.1 输出高电平 VCC，LED 就正向导通（导通时 LED 上的压降大于 1V），有电流流过 LED，至发 LED 发亮。实际上由于 P1.1 高电平输出电阻为 10K，起到输出限流的作用，所以流过 LED 的电流小于（5V-1V）/10K = 0.4mA。只要 P1.1 输出低电平 GND，实际小于 0.3V，LED 就不能导通，结果 LED 不亮。

开关双键的输入：输入先输出高

一个按键 KEY_ON 接在 P1.6 与 GND 之间，另一个按键 KEY_OFF 接 P1.7 与 GND 之间，按 KEY_ON 后 LED 亮，按 KEY_OFF 后 LED 灭。同时按下 LED 半亮，LED 保持后松开键的状态，即 ON 亮 OFF 灭。

```
#include <at89x52.h>
#define LED         P1^1            //用符号 LED 代替 P1_1
#define KEY_ON      P1^6            //用符号 KEY_ON 代替 P1_6
#define KEY_OFF     P1^7            //用符号 KEY_OFF 代替 P1_7
void main( void )                   //单片机复位后的执行入口，void 表示空，无输入参数，无返回值
{
    KEY_ON = 1;         //作为输入，首先输出高，接下 KEY_ON，P1.6 则接地为 0，否则输入为 1
    KEY_OFF = 1;        //作为输入，首先输出高，接下 KEY_OFF，P1.7 则接地为 0，否则输入为 1
    While( 1 )          //永远为真，所以永远循环执行如下括号内所有语句
    {
        if( KEY_ON==0 )    LED=1;    //是 KEY_ON 接下，所示 P1.1 输出高，LED 亮
        if( KEY_OFF==0 )   LED=0;    //是 KEY_OFF 接下，所示 P1.1 输出低，LED 灭
    }   //松开键后，都不给 LED 赋值，所以 LED 保持最后按键状态。
        //同时按下时，LED 不断亮灭，各占一半时间，交替频率很快，由于人眼惯性，看上去为半亮态
}
```

数码管的接法和驱动原理

一支七段数码管实际由 8 个发光二极管构成，其中 7 个组形构成数字 8 的七段笔画，所以称为七段数码管，而余下的 1 个发光二极管作为小数点。作为习惯，分别给 8 个发光二极管标上记号：a,b,c,d,e,f,g,h。对应 8 的顶上一画，按顺时针方向排，中间一画为 g，小数点为 h。

我们通常又将各二极与一个字节的 8 位对应，a(D0),b(D1),c(D2),d(D3),e(D4),f(D5),g(D6),h(D7)，相应 8 个发光二极管正好与单片机一个端口 Pn 的 8 个引脚连接，这样单片机就可以通过引脚输出高低电平控制 8 个发光二极的亮与灭，从而显示各种数字和符号；对应字节，引脚接法为：a(Pn.0),b(Pn.1),c(Pn.2)，d(Pn.3)，e(Pn.4)，f(Pn.5)，g(Pn.6)，h(Pn.7)。

如果将 8 个发光二极管的负极（阴极）内接在一起，作为数码管的一个引脚，这种数码管则被称为共阴数码管，共同的引脚则称为共阴极，8 个正极则为段极。否则，如果是将正极（阳极）内接在一起引出的，则称为共阳数码管，共同的引脚则称为共阳极，8 个负极则为段极。

以单支共阴数码管为例，可将段极接到某端口 Pn，共阴极接 GND，则可编写出对应十六进制码的七段码表字节数据如右图：

显示字符	共阴极段选码	共阳极段选码	显示字符	共阴极段选码	共阳极段选码
0	3FH	C0H	C	39H	C6H
1	06H	F9H	D	5EH	A1H
2	5BH	A4H	E	79H	86H
3	4FH	B0H	F	71H	84H
4	66H	99H	P	73H	82H
5	6DH	92H	U	3EH	C1H
6	7DH	82H	r	31H	CEH
7	07H	F8H	y	6EH	91H
8	7FH	80H	8	FFH	00H
9	6FH	90H	“灭”	00H	FFH
A	77H	88H			⋮
B	7CH	83H			

16 键码显示的程序

我们在 P1 端口接一支共阴数码管 SLED，在 P2、P3 端口接 16 个按键，分别编号为 KEY_0、KEY_1 到 KEY_F，操作时只能按一个键，按键后 SLED 显示对应键编号。

```
#include <at89x52.h>
#define SLED  P1
#define KEY_0 P2^0
#define KEY_1 P2^1
#define KEY_2 P2^2
#define KEY_3 P2^3
#define KEY_4 P2^4
#define KEY_5 P2^5
#define KEY_6 P2^6
#define KEY_7 P2^7
#define KEY_8 P3^0
#define KEY_9 P3^1
#define KEY_A P3^2
#define KEY_B P3^3
#define KEY_C P3^4
#define KEY_D P3^5
#define KEY_E P3^6
#define KEY_F P3^7
Code unsigned char Seg7Code[16]= //用十六进数作为数组下标，可直接取得对应的七段编码字节
// 0    1    2    3    4    5    6    7    8    9    A    b    C    d    E    F
{0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
void main( void )
{
    unsigned char i=0; //作为数组下标
```

```
    P2 = 0xff;     //P2 作为输入，初始化输出高
    P3 = 0xff;     //P3 作为输入，初始化输出高
    While( 1 )
    {
        if( KEY_0 == 0 ) i=0;       if( KEY_1 == 0 ) i=1;
        if( KEY_2 == 0 ) i=2;       if( KEY_3 == 0 ) i=3;
        if( KEY_4 == 0 ) i=4;       if( KEY_5 == 0 ) i=5;
        if( KEY_6 == 0 ) i=6;       if( KEY_7 == 0 ) i=7;
        if( KEY_8 == 0 ) i=8;       if( KEY_9 == 0 ) i=9;
        if( KEY_A == 0 ) i=0xA;     if( KEY_B == 0 ) i=0xB;
        if( KEY_C == 0 ) i=0xC;     if( KEY_D == 0 ) i=0xD;
        if( KEY_E == 0 ) i=0xE;     if( KEY_F == 0 ) i=0xF;
        SLED = Seg7Code[ i ];  //开始时显示 0，根据 i 取应七段编码
    }
}
```

第二节：双数码管可调秒表

解：只要满足题目要求，方法越简单越好。由于单片机 I/O 资源足够，所以双数码管可接成静态显示方式，两个共阴数码管分别接在 P1（秒十位）和 P2（秒个位）口，它们的共阴极都接地，安排两个按键接在 P3.2（十位数调整）和 P3.3（个位数调整）上，为了方便计时，选用 12MHz 的晶体。为了达到精确计时，选用定时器方式 2，每计数 250 重载一次，即 250us，定义一整数变量计数重载次数，这样计数 4000 次即为一秒。定义两个字节变量 S10 和 S1 分别计算秒十位和秒个位。编得如下程序：

```
#include <at89x52.h>
Code unsigned char Seg7Code[16]= //用十六进数作为数组下标，可直接取得对应的七段编码字节
// 0    1    2    3    4    5    6    7    8    9    A    b    C    d    E    F
{0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
void main( void )
{
    unsigned int us250 = 0;
    unsigned char s10 = 0;
    unsigned char s1 = 0;
    unsigned char key10 = 0;     //记忆按键状态，为 1 按下
    unsigned char key1 = 0;      //记忆按键状态，为 1 按下
    //初始化定时器 Timer0
    TMOD = (TMOD & 0xF0) | 0x02;
    TH1 = -250;   //对于 8 位二进数来说，-250=6，也就是加 250 次 1 时为 256，即为 0
    TR1 = 1;
    while(1){                                              //----------循环 1
        P1 = Seg7Code[ s10 ];  //显示秒十位
        P2 = Seg7Code[ s1 ];   //显示秒个位
        while( 1 ){                                        //----------循环 2
            //计时处理
            if( TF0 == 1 ){
```

```
                TF0 = 0;
                if( ++us250 >= 4000 ){
                    us250 = 0;
                    if( ++s1 >= 10 ){
                        s1 = 0;
                        if( ++s10 >= 6 ) s10 = 0;
                    }
                    break;    //结束“循环 2”，修改显示
                }
            }
            //按十位键处理
            P3.2 = 1;      //P3.2 作为输入，先要输出高电平
            if( key10 == 1 ){  //等松键
                if( P3.2 == 1 )    key10=0;
            }
            else{              //未按键
                if( P3.2 == 0 ){
                    key10 = 1;
                    if( ++s10 >= 6 ) s10 = 0;
                    break; //结束“循环 2”，修改显示
                }
            }
            //按个位键处理
            P3.3 = 1;      //P3.3 作为输入，先要输出高电平
            if( key1 == 1 )    //等松键
            {    if( P3.3 == 1 )    key1=0;   }
            else {             //未按键
                if( P3.3 == 0 ){   key1 = 1;
                    if( ++s1 >= 10 ) s1 = 0;
                    break; //结束“循环 2”，修改显示
                }
            }
        } //循环 2’ end
    }//循环 1’ end
}//main’ end
```

第三节：十字路口交通灯

如果一个单位时间为 1 秒，这里设定的十字路口交通灯按如下方式四个步骤循环工作：

- λ 60 个单位时间，南北红，东西绿；
- λ 10 个单位时间，南北红，东西黄；
- λ 60 个单位时间，南北绿，东西红；
- λ 10 个单位时间，南北黄，东西红；

解：用 P1 端口的 6 个引脚控制交通灯，高电平灯亮，低电平灯灭。

```
#include <at89x52.h>
//sbit 用来定义一个符号位地址，方便编程，提高可读性，和可移植性
sbit SNRed    =P1^0;        //南北方向红灯
sbit SNYellow =P1^1;        //南北方向黄灯
sbit SNGreen  =P1^2;        //南北方向绿灯
sbit EWRed    =P1^3;        //东西方向红灯
sbit EWYellow =P1^4;        //东西方向黄灯
sbit EWGreen  =P1^5;        //东西方向绿灯
/* 用软件产生延时一个单位时间 */
void Delay1Unit( void )
{
    unsigned int i, j;
    for( i=0; i<1000; i++ )
        for( j<0; j<1000; j++ );     //通过实测，调整 j 循环次数,产生 1ms 延时
//还可以通过生成汇编程序来计算指令周期数，结合晶体频率来调整 j 循环次数，接近 1ms
}
/* 延时 n 个单位时间 */
void Delay( unsigned int n ){ for( ; n!=0; n-- ) Delay1Unit(); }
void main( void )
{
    while( 1 )
    {
        SNRed=0; SNYellow=0; SNGreen=1; EWRed=1; EWYellow=0; EWGreen=0; Delay( 60 );
        SNRed=0; SNYellow=1; SNGreen=0; EWRed=1; EWYellow=0; EWGreen=0; Delay( 10 );
        SNRed=1; SNYellow=0; SNGreen=0; EWRed=0; EWYellow=0; EWGreen=1; Delay( 60 );
        SNRed=1; SNYellow=0; SNGreen=0; EWRed=0; EWYellow=1; EWGreen=0; Delay( 10 );
    }
}
```

第四节：数码管驱动

显示“12345678”

P1 端口接 8 联共阴数码管 SLED8 的段极：P1.7 接段 h,…，P1.0 接段 a

P2 端口接 8 联共阴数码管 SLED8 的段极：P2.7 接左边的共阴极，…，P2.0 接右边的共阴极

方案说明：晶振频率 fosc=12MHz，数码管采用动态刷新方式显示，在 1ms 定时断服务程序中实现

```
#include <at89x92.h>
unsigned char DisBuf[8];     //全局显示缓冲区，DisBuf[0]对应右 SLED，DisBuf[7]对应左 SLED，
void DisplayBrush( void )
{    code unsigned char cathode[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};    //阴极控制码
     Code unsigned char Seg7Code[16]= //用十六进数作为数组下标，可直接取得对应的七段编码字节
     {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
     static unsigned char i=0; //  （0≤i≤7） 循环刷新显示，由于是静态变量，此赋值只做一次。
```

```
    P2 = 0xff;          //显示消隐，以免下一段码值显示在前一支 SLED
    P1 = Seg7Code[ DisBuf[i] ]; //从显示缓冲区取出原始数据，查表变为七段码后送出显示
    P2 = cathode[ i ];          //将对应阴极置低，显示
    if( ++i >= 8 ) i=0;         //指向下一个数码管和相应数据
}
void TimerOIntRoute( void ) interrupt 1
{
    TL0 = -1000;        //由于 TL0 只有 8bits，所以将（-1000）低 8 位赋给 TL0
    TH0 = (-1000)>>8;  //取（-1000）的高 8 位赋给 TH0，重新定时 1ms
    DisplayBrush();
}
void TimerOInit( void )
{   TMOD=(TMOD & 0xf0) | 0x01;  //初始化，定时器 T0，工作方式 1
    TL0 = -1000;  //定时 1ms
    TH0 = (-1000)>>8;
    TR0 = 1;            //允许 T0 开始计数
    ET0 = 1;            //允许 T0 计数溢出时产生中断请求
}
void Display( unsigned char index, unsigned char dataValue ){ DisBuf[ index ] = dataValue; }
void main( void )
{
    unsigned char i;
    for( i=0; i<8; i++ ){ Display(i, 8-i); } //DisBuf[0]为右，DisBuf[7]为左
    TimerOInit();
    EA = 1;             //允许 CPU 响应中断请求
    While(1);
}
```

第五节：键盘驱动

指提供一些函数给任务调用，获取按键信息，或读取按键值。

定义一个头文档 <KEY.H>，描述可用函数，如下：

```
#ifndef _KEY_H_         //防止重复引用该文档，如果没有定义过符号 _KEY_H_，则编译下面语句
#define _KEY_H_         //只要引用过一次，即 #include <key.h>，则定义符号 _KEY_H_
unsigned char keyHit( void );    //如果按键，则返回非 0，否则返回 0
unsigned char keyGet( void );    //读取按键值，如果没有按键则等待到按键为止
void keyPut( unsigned char ucKeyVal );     //保存按键值 ucKeyVal 到按键缓冲队列末
void keyBack( unsigned char ucKeyVal );    //退回键值 ucKeyVal 到按键缓冲队列首
#endif
```

定义函数体文档 KEY.C，如下：

```
＃include “key.h”
#define KeyBufSize 16   //定义按键缓冲队列字节数
```

```
unsigned char KeyBuf[ KeyBufSize ];   //定义一个无符号字符数组作为按键缓冲队列。该队列为先进
                                      //先出，循环存取，下标从 0 到 KeyBufSize-1
unsigned char KeyBufWp=0;    //作为数组下标变量，记录存入位置
unsigned char KeyBufRp=0;    //作为数组下标变量，记录读出位置
//如果存入位置与读出位置相同，则表明队列中无按键数据
unsigned char keyHit( void )
{    if( KeyBufWp == KeyBufRp ) return( 0 ); else return( 1 ); }

unsigned char keyGet( void )
{    unsigned char retVal;   //暂存读出键值
     while( keyHit()==0 );   //等待按键，因为函数 keyHit()的返回值为 0 表示无按键
     retVal = KeyBuf[ KeyBufRp ];      //从数组中读出键值
     if( ++KeyBufRp >= KeyBufSize ) KeyBufRp=0;      //读位置加 1，超出队列则循环回初始位置
     return( retVal );
}

void keyPut( unsigned char ucKeyVal )
{    KeyBuf[ KeyBufWp ] = ucKeyVal;   //键值存入数组
     if( ++KeyBufWp >= KeyBufSize ) KeyBufWp=0;      //存入位置加 1，超出队列则循环回初始位置
}
/*********************************************************************************************
由于某种原因，读出的按键，没有用，但其它任务要用该按键，但传送又不方便。此时可以退回按键队列。
就如取错了信件，有必要退回一样
*********************************************************************************************/
void keyBack( unsigned char ucKeyVal )
{
     /*
     如果 KeyBufRp=0; 减 1 后则为 FFH，大于 KeyBufSize，即从数组头退回到数组尾。或者由于干扰使得
     KeyBufRp 超出队列位置，也要调整回到正常位置，
     */
     if( --KeyBufRp >= KeyBufSize ) KeyBufRp=KeyBufSize-1;
     KeyBuf[ KeyBufRp ] = ucKeyVal;   //回存键值
}
```

下面渐进讲解键盘物理层的驱动。

电路共同点：P2 端口接一共阴数码管，共阴极接 GND，P2.0 接 a 段、P2.1 接 b 段、…、P2.7 接 h 段。

软件共同点：code unsigned char Seg7Code[10] 是七段数码管共阴编码表。

```
Code unsigned char Seg7Code[16]=
// 0     1     2     3     4     5     6     7     8     9     A     b     C     d     E     F
{0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
```

例一：P1.0 接一按键到 GND，键编号为‘6’，显示按键。

```
#include <at89x52.h>
#include “KEY.H”
void main( void )
```

```
{    P1_0 = 1;      //作为输入引脚，必须先输出高电平
     while( 1 )     //永远为真，即死循环
     {    if( P1_0 == 0 )    //如果按键，则为低电平
          {    keyPut( 6 );  //保存按键编号值为按键队列
               while( P1_0 == 0 );     //如果一直按着键，则不停地执行该循环，实际是等待松键
          }
          if( keyHit() != 0 )     //如果队列中有按键
               P2=Seg7Code[ keyGet() ];     //从队列中取出按键值，并显示在数码管上
     }
}
```

例二：在例一中考虑按键 20ms 抖动问题。

```
#include <at89x52.h>
#include “KEY.H”
void main( void )
{    P1_0 = 1;      //作为输入引脚，必须先输出高电平
     while( 1 )     //永远为真，即死循环
     {    if( P1_0 == 0 )    //如果按键，则为低电平
          {    delay20ms();  //延时 20ms，跳过接下抖动
               keyPut( 6 );  //保存按键编号值为按键队列
               while( P1_0 == 0 );     //如果一直按着键，则不停地执行该循环，实际是等待松键
               delay20ms();  //延时 20ms，跳过松开抖动
          }
          if( keyHit() != 0 )     //如果队列中有按键
               P2=Seg7Code[ keyGet() ];     //从队列中取出按键值，并显示在数码管上
     }
}
```

例三：在例二中考虑干扰问题。即小于 20ms 的负脉冲干扰。

```
#include <at89x52.h>
#include “KEY.H”
void main( void )
{    P1_0 = 1;      //作为输入引脚，必须先输出高电平
     while( 1 )     //永远为真，即死循环
     {    if( P1_0 == 0 )    //如果按键，则为低电平
          {    delay20ms();  //延时 20ms，跳过接下抖动
               if( P1_0 == 1 ) continue;   //假按键
               keyPut( 6 );  //保存按键编号值为按键队列
               while( P1_0 == 0 );     //如果一直按着键，则不停地执行该循环，实际是等待松键
               delay20ms();  //延时 20ms，跳过松开抖动
          }
          if( keyHit() != 0 )     //如果队列中有按键
               P2=Seg7Code[ keyGet() ];     //从队列中取出按键值，并显示在数码管上
     }
}
```

例四：状态图编程法。通过 20ms 周期中断，扫描按键。

```
/*********************************************************************************************
```

采用晶体为 12KHz 时，指令周期为 1ms（即主频为 1KHz），这样 T0 工作在定时器方式 2，8 位自动重载。计数值为 20，即可产生 20ms 的周期性中断，在中断服务程序中实现按键扫描

```
*********************************************************************************************/
#include <at89x52.h>
#include “KEY.H”
void main( void )
{
    TMOD = （TMOD & 0xf0 ） | 0x02;  //不改变 T1 的工作方式，T0 为定时器方式 2
    TH0 = -20;                       //计数周期为 20 个主频脉，即 20ms
    TL0=TH0;                         //先软加载一次计数值
    TR0=1;                           //允许 T0 开始计数
    ET0=1;                           //允许 T0 计数溢出时产生中断请求
    EA=1;                            //允许 CPU 响应中断请求
    while( 1 )    //永远为真，即死循环
    {
        if( keyHit() != 0 )     //如果队列中有按键
            P2=Seg7Code[ keyGet() ];     //从队列中取出按键值，并显示在数码管上
    }
}
void timer0int( void ) interrupt 1    //20ms；T0 的中断号为 1
{   static unsigned char sts=0;
    P1_0 = 1;     //作为输入引脚，必须先输出高电平
    switch( sts )
    {
        case 0: if( P1_0==0 ) sts=1; break;   //按键则转入状态 1
        case 1:
            if( P1_0==1 ) sts=0;         //假按错，或干扰，回状态 0
            else{ sts=2; keyPut( 6 ); } //确实按键，键值入队列，并转状态 2
            break;
        case 2: if( P1_0==1 ) sts=3; break;   //如果松键，则转状态 3
        case 3:
            if( P1_0==0 ) sts=2;         //假松键，回状态 2
            else sts=0;                  //真松键，回状态 0，等待下一次按键过程
    }
}
```

例五：状态图编程法。

```
/*********************************************************************************************
```

如果采用晶体为 12MHz 时，指令周期为 1us（即主频为 1MHz），要产生 20ms 左右的计时，则计数值达到 20000，T0 工作必须为定时器方式 1，16 位非自动重载，即可产生 20ms 的周期性中断，在中断服务程序中实现按键扫描

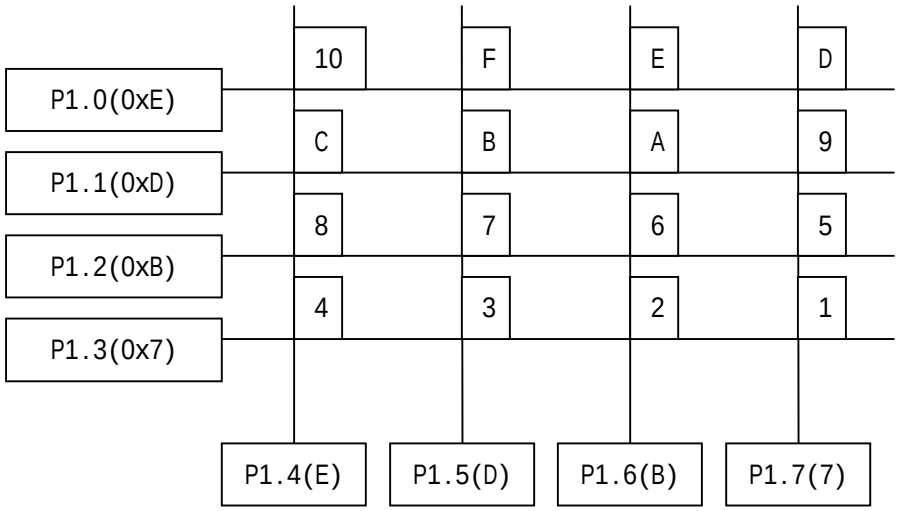
```
*********************************************************************************************/
#include <at89x52.h>
#include “KEY.H”
```

```
void main( void )
{
    TMOD = （TMOD & 0xf0 ） | 0x01;  //不改变 T1 的工作方式，T0 为定时器方式 1
    TL0 = -20000;                    //计数周期为 20000 个主频脉，自动取低 8 位
    TH0 = (-20000)>>8;               //右移 8 位，实际上是取高 8 位
    TR0=1;                           //允许 T0 开始计数
    ET0=1;                           //允许 T0 计数溢出时产生中断请求
    EA=1;                            //允许 CPU 响应中断请求
    while( 1 )    //永远为真，即死循环
    {
        if( keyHit() != 0 )     //如果队列中有按键
            P2=Seg7Code[ keyGet() ];     //从队列中取出按键值，并显示在数码管上
    }
}
void timer0int( void ) interrupt 1    //20ms；T0 的中断号为 1
{   static unsigned char sts=0;
    TL0 = -20000;                    //方式 1 为软件重载
    TH0 = (-20000)>>8;               //右移 8 位，实际上是取高 8 位
    P1_0 = 1;     //作为输入引脚，必须先输出高电平
    switch( sts )
    {
        case 0: if( P1_0==0 ) sts=1; break;  //按键则转入状态 1
        case 1:
            if( P1_0==1 ) sts=0;         //假按错，或干扰，回状态 0
            else{ sts=2; keyPut( 6 ); } //确实按键，键值入队列，并转状态 2
            break;
        case 2: if( P1_0==1 ) sts=3; break;  //如果松键，则转状态 3
        case 3:
            if( P1_0==0 ) sts=2;         //假松键，回状态 2
            else sts=0;                  //真松键，回状态 0，等待下一次按键过程
    }
}
```

例六：4X4 按键。

```
/*********************************************************************************************
```

由 P1 端口的高 4 位和低 4 位构成 4X4 的矩阵键盘，本程序只认为单键操作为合法，同时按多键时无效。这样下面的 X，Y 的合法值为 0x7，0xb，0xd，0xe，0xf，通过表 keyCode 影射变换可得按键值




```
*******************************************************************************************/
#include <at89x52.h>
#include "KEY.H"
unsigned char keyScan( void )    //返回0表示无按键，或无效按键，其它值为按键编码值
{    code unsigned char keyCode[16]=
     /0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF
     { 0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,   2,   0,   3,   4,   0 };
     unsigned char x, y, retVal;
     P1=0x0f;                //低四位输入，高四位输出0
     x=P1&0x0f;              //P1输入后，清高四位，作为X值
     P1=0xf0;                //高四位输入，低四位输出0
     y=(P1 >> 4) & 0x0f;     //P1输入后移位到低四位，并清高四位，作为Y值
     retVal = keyCode[x]*4 + keyCode[y];  //根据本公式倒算按键编码
     if( retVal==0 ) return(0); else return( retVal-4 );
}
//比如按键'1'，得X=0x7，Y=0x7，算得retVal= 5，所以返回函数值1。
//双如按键'7'，得X=0xb，Y=0xd，算得retVal=11，所以返回函数值7。
void main( void )
{
     TMOD = （TMOD & 0xf0 ） | 0x01;  //不改变T1的工作方式，T0为定时器方式1
     TL0 = -20000;                    //计数周期为20000个主频脉，自动取低8位
     TH0 = (-20000)>>8;               //右移8位，实际上是取高8位
     TR0=1;                           //允许T0开始计数
     ET0=1;                           //允许T0计数溢出时产生中断请求
     EA=1;                            //允许CPU响应中断请求
     while( 1 )    //永远为真，即死循环
     {
          if( keyHit() != 0 )     //如果队列中有按键
```

```
            P2=Seg7Code[ keyGet() ];     //从队列中取出按键值，并显示在数码管上
    }
}
void timer0int( void ) interrupt 1    //20ms；T0 的中断号为 1
{   static unsigned char sts=0;
    TL0 = -20000;                     //方式 1 为软件重载
    TH0 = (-20000)>>8;                //右移 8 位，实际上是取高 8 位
    P1_0 = 1;      //作为输入引脚，必须先输出高电平
    switch( sts )
    {
        case 0: if( keyScan()!=0 ) sts=1; break;  //按键则转入状态 1
        case 1:
            if( keyScan()==0 ) sts=0;         //假按错，或干扰，回状态 0
            else{ sts=2; keyPut( keyScan() ); }  //确实按键，键值入队列，并转状态 2
            break;
        case 2: if(keyScan()==0 ) sts=3; break;   //如果松键，则转状态 3
        case 3:
            if( keyScan()!=0 ) sts=2;         //假松键，回状态 2
            else sts=0;                   //真松键，回状态 0，等待下一次按键过程
    }
}
```

第六节：低频频率计

实例目的：学时定时器、计数器、中断应用

说明：选用 24MHz 的晶体，主频可达 2MHz。用 T1 产生 100us 的时标，T0 作信号脉冲计数器。假设晶体频率没有误差，而且稳定不变（实际上可达万分之一）；被测信号是周期性矩形波（正负脉冲宽度都不能小于 0.5us），频率小于 1MHz，大于 1Hz。要求测量时标 1S，测量精度为 0.1%。

解：从测量精度要求来看，当频率超过 1KHz 时，可采用 1S 时标内计数信号脉冲个数来测量信号频，而信号频率低于 1KHz 时，可以通过测量信号的周期来求出信号频率。两种方法自动转换。

对于低于 1KHz 的信号，信号周期最小为 1ms，也就是说超过 1000us，而我们用的定时器计时脉冲周期为 0.5us，如果定时多计或少计一个脉冲，误差为 1us，所以相对误差为 1us/1000us=0.1%。信号周期越大，即信号频率越低，相对误差就越小。

从上面描述来看，当信号频率超过 1KHz 后，信号周期就少于 1000us，显然采用上面的测量方法，不能达到测量精度要求，这时我们采用 1S 单位时间计数信号的脉冲个数，最少能计到 1000 个脉冲，由于信号频率不超过 1MHz，而我们定时脉冲为 2MHz，最差多计或少计一个信号脉冲，这样相对误差为 1/1000，可见信号频率越高，相对误差越小。

信号除输入到 T1（P3.5）外，还输入到 INT1（P3.3）。

```
unsigned int us100;               //对 100us 时间间隔单位计数，即有多少个 100us。
unsigned char Second;
unsigned int K64;                 //对 64K 单位计数，即有多少个 64K
unsigned char oldT0;
unsigned int oldus, oldK64, oldT1;
unsigned long fcy;                //存放频率值，单位为 Hz
```

```
bit HighLow=1;                    //1：表示信号超过 1KHz；0：表示信号低于 1KHz。
void InitialHigh( void )
{
     IE=0; IP=0; HighLow=1;
     TMOD = (TMOD & 0xf0) | 0x02; TH0=-200; TL0=TH0; PX0=1; T0=1;
     TMOD = (TMOD & 0x0f) | 0x50; TH1=0; TL1=0; T1=1; ET1=1;
     Us100=0; Second=0; K64=0;
     oldK64=0; oldT1=0;
     TCON |= 0x50;      //同时置 TR0=1; TR1=1;
     EA = 1;
}
void InitialLow( void )
{
     IE=0; IP=0; HighLow=0;
     TMOD = (TMOD & 0xf0) | 0x02; TH0=-200; TL0=TH0; ET0=1; TR0=1;
     INT1 = 1; IT1=1; EX1=1;
     Us100=0; Second=0; K64=0;
     oldK64=0; oldT1=0;
     EA = 1;
}
void T0intr( void ) interrupt 1
{    if( HighLow==0 ) ++us100;
     else
          if( ++us100 >= 10000 )
          {    unsigned int tmp1, tmp2;
               TR1=0; tmp1=(TH1<<8) + (TL1); tmp2=K64; TR1=1;
               fcy=((tmp2-oldK64)<<16) + (tmp1-oldT1);
               oldK64=tmp1; oldT1=tmp2;
               Second++;
               us100=0;
          }
}
void T1intr( void ) interrupt 3 {    ++K64;    }
void X1intr( void ) interrupt 2
{    static unsigned char sts=0;
     switch( sts )
     {
          case 0: sts = 1; break;
          case 1: oldT0=TL0; oldus=us100; sts=2; break;
          case 2:
          {
               unsigned char tmp1, tmp2;
               TR0=0; tmp1=TL0; tmp2=us100; TR0=1;
               fcy = 1000000L/( (tmp2-oldus)*100L + (256-tmp1)/2 );
```

```
                Second ++;
            }
                Sts = 0;
                break;
        }
    }
void main( void )
{
    if( HighLow==1) InitialHigh(); else InitialLow();
    While(1)
    {
        if( Second != 0 )
        {
            Second = 0;
            //display fcy  引用前面的数码管驱动程序，注意下面对 T0 中断服务程序的修改
            {   unsigned char i;
                for( i=0; i<8; i++ ){ Display(i, fcy%10); fcy /= 10; }
            }
            if( HighLow==1 )
                if( fcy<1000L ){ InitalLow();}
            else
                if( fcy>1000L ){ InitalHigh();}
        }
    }
}
//修改 T0 的中断服务程序，让它在完成时标的功能时，同时完成数码管显示刷新
void T0intr( void ) interrupt 1
{
    static unsigned char ms = 0;
    if( HighLow==0 ) ++us100;
    else
        if( ++us100 >= 10000 )
        {   unsigned int tmp1, tmp2;
            TR1=0; tmp1=(TH1<<8) + (TL1); tmp2=K64; TR1=1;
            fcy=((tmp2-oldK64)<<16) + (tmp1-oldT1);
            oldK64=tmp1; oldT1=tmp2;
            Second++;
            us100=0;
        }
    if( ++ms >= 10 ){ ms=0; DisplayBrush(); } //1ms 数码管刷新
}
```

第七节：电子表

单键可调电子表：主要学习编程方法。
外部中断应用，中断嵌
解：电子表分为工作状态和调整状态。平时为工作状态，按键不足一秒，接键为换屏‘S’。按键超过一秒移位则进入调整状态‘C’，而且调整光标在秒个位开始。调整状态时，按键不足一秒为光标移动‘M’，超过一秒则为调整读数，每 0.5 秒加一‘A’，直到松键；如果 10 秒无按键则自动回到工作状态‘W’。
如果有年、月、日、时、分、秒。四联数码管可分三屏显示，显示格式为“年月.”、“日.时.”、“分.秒”，从小数点的位置来区分显示内容。（月份的十位数也可以用“-”和“-1”表示）。

```
enum status = { Work, Change, Add, Move, Screen }  //状态牧举
//计时和调整都是对下面时间数组 Time 进行修改
unsigned char Time[12]={0,4, 0,6, 1,0, 0,8, 4,5, 3,2};  //04 年 06 月 10 日 08 时 45 分 32 秒
unsigned char cursor = 12;  //指向秒个位，=0 时无光标
unsigned char YmDhMs = 3;   //指向“分秒”显示，=0 时无屏显
static unsigned char sts = Work;
/*
如果 cursor 不为 0，装入 DisBuf 的对应数位，按 0.2 秒周期闪烁，即设一个 0.1 秒计数器 S01，S01 为奇数时灭，S01 为偶数时亮。
小数点显示与 YmDhMs 变量相关。
*/
void DisScan( void )   //动态刷新显示时调用。没编完，针对共阴数码管，只给出控控制算法
{
    //DisBuf 每个显示数据的高四位为标志，最高位 D7 为负号，D6 为小数点，D5 为闪烁
    unsigned char tmp;
    tmp = Seg7Code[?x & 0x1f ];      //设?x 为显示数据，高 3 位为控制位，将低 5 位变为七段码
    if( ?x & 0x40 ) tmp |= 0x80;     //添加小数点
    if( ?x & 0x20 ){ if( S01 & 0x01 ) tmp=0; } //闪烁，S01 奇数时不亮
    //这里没有处理负号位
    //将 tmp 送出显示，并控制对应数码管动作显示
}
void Display( void )        //根据状态进行显示
{
    if( cursor != 0 ){ YmDhMs=(cursor+3)/4; } //1..4=1; 5..8=2; 9..12=3
    for( i=(YmDhMs-1)*4; i<(YmDhMs)*4; i++ )
    {   unsigned char j = i%4;
        Disbuf[j] = Time[i];
        if( i == (cursor-1) ) Disbuf[j] |= 0x20;  //闪烁,cursor!=0 时才闪烁
        if(  (i==9) ||     //小数点：分个位
             (i==7) ||     //小数点：时个位
             (i==5) ||     //小数点：日个位
             (i==3)        //小数点：月个位
        )   Disbuf[j] |= 0x40;
        //if(i==2){ if(Time[2]==1) DisBuf[2]=“-1”; else DisBuf=“-”; }
    }
```

```
    //工作状态：根据 YmDhMs 将屏数据装入 DisBuf
    //调整状态：根据 cursor 将屏数据装入 DisBuf
}
void KeyScan( void )        //根据状态扫描按键
void ProcessKey( void )     //根据状态处理键信息
{
    keyVal = KeyGet();
    if( keyVal == 0 ) return;
    switch( sts )
    {
        case Work:
            if( keyVal == 'S' )
            {
                if( --YmDhMs == 0 ) YmDhMs = 3;  //换屏
            }
            if( keyVal == 'C' )
            {
                sts = Change;
                YmDhMs = 3;
                Cursor = 12;
            }
            break;
        case Change:
            if( keyVal == 'W' )
            if( keyVal == 'A' )
            if( keyVal == 'M' ) //根据 cursor
            break;
    }
}
```

第八节：串行口应用

一、 使用晶体频率为 22.1184MHz 的 AT89C52 单片机，串行口应用工作方式 1，以 9600bps 的波特率向外发送数据，数据为十个数字‘0’到‘9’，循环不断地发送。

解：数字字符为增量进二进制码，‘0’对应 0x30，‘1’= ‘0’+ 1 = 0x31，从‘0’到‘9’对应编码为 0x30 到 0x39，记忆二进制编码较难，实际编程中用单引号括起对应字符表示引用该字符的二进制编码值，如‘？’表示引用？号的编码值。

在用 11.0592MHz 晶体时，9600bps 的初始化分频初值为-6，现晶频加倍，如果其它条件不变，只有分频初始加倍为-12，才能得到 9600bps；如果想得到 2400bps（速率降 4 倍），分频初始自然加大 4 倍，即为-48。根据题意编得如下程序：

```
#include <at89x52.h>
void main( void )
{
```

```
    TMOD = (TMOD & 0x0F) | 0x20;
    TH1 = -12;
    PCON |= 0x80; //SMOD = 1
    TR1 = 1;
    SCON = 0x42;
    while( 1 )
    {
        if( TI==1 )
        {
            static unsigned char Dat= '0' ;
            SBUF = Dat;
            TI = 0;
            If( ++Dat >  '9' ) Dat= '0' ;
        }
    }
}
```

二、 在上题的基础上，改为 2400bps，循环发送小写字母 'a' 到 'z'，然后是大写字母 'A' 到 'Z'。

```
#include <at89x52.h>
void main( void )
{
    TMOD = (TMOD & 0x0F) | 0x20;
    TH1 = -96;          //注意不用倍频方式
    PCON &= 0x7F; //SMOD = 0
    TR1 = 1;
    SCON = 0x42;
    while( 1 )
    {
        if( TI==1 )
        {
            static unsigned char Dat= 'a' ;
            SBUF = Dat;
            TI = 0;
            //If( ++Dat >  '9' )    Dat= '0' ;
            ++Dat;
            if( Dat == ( 'z' +1) )  Dat= 'A' ;
            if( Dat == ( 'Z' +1) )  Dat= 'a' ;
        }
    }
}
```

上述改变值时，也可以再设一变量表示当前的大小写状态，比如写成如下方式：

```
            ++Dat;
            {
                static unsigned char Caps=1;
                if( Caps != 0 )
```

```
                    if( Dat> 'Z' ){ Dat= 'a' ; Caps=0; }
                else
                    if( Dat> 'z' ){ Dat= 'A' ; Caps=1; }
            }
```

如下写法有错误：因为小 b 比大 Z 的编码值大，所以 Dat 总是 'a'

```
            ++Dat;
            if( Dat> 'Z' ){ Dat= 'a' }
            else if( Dat> 'z' ){ Dat= 'A' }
```

三、 有 A 和 B 两台单片机，晶体频率分别为 13MHz 和 14MHz,在容易编程的条件下，以最快的速度进行双工串行通信，A 给 B 循环发送大写字母从 'A' 到 'Z'，B 给 A 循环发送小写字母从 'a' 到 'z'，双方都用中断方式进行收发。

解：由于晶体频率不同，又不成 2 倍关系，所以只有通信方式 1 和方式 3，由于方式 3 的帧比方式 1 多一位，显然方式 3 的有效数据（9/11）比方式 1（8/10）高，但要用方式 3 的第 9 位 TB8 来发送数据，编程难度较大，这里方式 1 较容易编程。

在计算最高速率时，由于单方程，双未知数，又不知道波特率为多少，所以要综合各方面的条件，估算出 A 和 B 的分频常数，分别为-13 和-14 时，速率不但相同，且为最大值。如下给出 A 机的程序：

```
#include <at89x52.h>
void main( void )
{
    TMOD = (TMOD & 0x0F) | 0x20;
    TH1 = -13;          //注意用倍频方式
    PCON |= 0x80; //SMOD = 1
    TR1 = 1;
    SCON = 0x52;  //REN = 1
    ES = 1;
    EA = 1;
    while( 1 );
}
void RS232_intr( void ) interrupt 4       //注意 RI 和 TI 任一位变为 1 都中断
{
    unsigned char rDat;
    if( RI == 1 ){ RI=0; rDat=SBUF; }
    if( TI==1 )
    {
        static unsigned char tDat= 'a' ;
        SBUF = tDat;
        TI = 0;
        If( ++Dat >  'z' ) Dat= 'a' ;
    }
}
```

四、多机通位

λ 在方式 2 和方式 3，SM2 只对接收有影响，当 SM2=1 时，只接收第 9 位等于 1 的帧（伪地址帧），而 SM2=0 时，第 9 位不影响接收。

- λ 多机通信中，地址的确认与本机程序有关，所以可以实现点对点、点对组、以及通播方式的通信。
- λ 如果收发共用一总线，任何时刻只有一个发送源能占用总线发送数据，否则发生冲突。由此可构造无竞争的令牌网；或者多主竞争总线网。