浙江大学实验报告

专业：生物医学工程
姓名：
学号：
日期：2025.5.6
地点：东 1B-416

课程名称：微机原理及其应用　指导老师：陈星　成绩：
实验名称：显示和键盘程序设计　实验类型：微机实验　同组学生姓名：

一、实验目的和要求

1. 实验目的

本实验旨在将实验五串口回复的字符展示于单片机的数码管之上，并且为 其设置一键清除显示的功能。通过这个实验，加深对单片机显示与按键程序设计的理解，掌握相关功能的实现方法，提升在单片机系统中处理输入输出以及功能扩展方面的能力。

2. 实验要求

(1). 通过 PC 电脑与开发板的串行接口连接，实现双向串行通讯，设定波特率为 4800。

(2). 在 PC 端输入“0”-“9”和“A”-“F”中的任何字符，51 处理器将此字符开始的后续 6 个字符显示在单片机的数码显示管上。

(3). 当输入字符达到字母表末尾时，遵循以下跳转规则：大写字母’F’后跳转至数字’0′，数字’9′后跳转至大写字母’A’。

(4). 必须考虑输入数据的合法性，对于不在处理范围内的字符，数码管应显示“ERROR_”。

(5). 任意设置一个独立按键为清除键，用于一键清除显示管内容。

(6). 数据缓存区不得设置在外部数据存储器。

(7). 对编写的程序先在 proteus 上仿真验证，后续用 keil 软件编译程序生成 HEX 文件，下载到开发板进行实机验证。

二、实验原理

1. 数码管动态显示原理

数码管的动态扫描显示技术是一种广泛应用于数字显示领域的驱动方法。在动态驱动方案中，所有数码管的八个显示段（标记为 a, b, c, d, e, f, g, dp）的同名端被连接在一起，而每个数码管的公共极（COM）增加位选通控制电路，位码选通由独立的 I/O 控制。当单片机输出特定的字形码时，所有数码管都会接收到这一字形码，但只有当相应的数码管的选通控制（位选码）被激活时，该数码管才会显示字形；未被选通的数码管则保持熄灭状态。注意：共阴极数码管的位码开关，为高时，数码管全灭，为低时，根据发光二极管阳极（一般称段码或字形码）的状态，高电平时，该段亮，低电平时，该段不亮。

通过快速轮流控制各个数码管的位码开关，实现了各个数码管的轮流显示，这就是动态驱动的基本原理。在这一轮流显示过程中，每个数码管的点亮时间控制在 1-2 毫秒。得益于人眼的视觉暂留效应以及发光二极管的余辉效应，即使各位数码管并非同时点亮，只要扫描速度足够快，人眼所感知到的显示效果将是一组稳

定的数据，不会有闪烁感。因此，动态显示的效果与静态显示相同，但动态显示能够显著节省 I/O 端口资源，并且降低功耗。

2. 多位数码管显示电路原理

由于单片机管脚有限，多位数码管的显示方法一般采用动态显示。选用 2 个 4 位 8 段共阴极数码显示器(7SEG-MPX4-CC)。数码管每段的电压电流参数和单个 LED 的几乎相同可达到 20mA 左右，而单片机管脚的驱动能力较弱，灌电流最大 6mA, 拉电流最大 0.22mA, 因此单片机的 I/O 口无法直接驱动，需要采用 74HC245 驱动器芯片来提高驱动能力。

数码管动态显示时,需要快速切换数码管的选择位,本实验中有 8 位数码管便有 8 个数码管的选择位,如果这些选择位直接和单片机管脚相连进行控制,那么需要 8 个管脚。这对管脚资源不丰富的 51 单片机来说是难以承受的。因此采用 74LS138 译码器来减少单片机管脚的使用数量, 该芯片通过 3 个输入管脚的高低组合可以实现 8 个输出引脚中单独每个引脚的低电平输出。

3. 中断工作方式原理

中断是指计算机在执行程序的过程中，当出现某些事件(如外部设备的数据到达)时，计算机停止当前正在执行的程序，转而去执行处理该事件的程序 (中断服务程序),处理完毕后再返回原来被中断的程序处继续执行。

在本实验的串行通信中，采用中断方式可以及时响应 PC 端发送的数据，提高系统的实时性和效率。当单片机接收到来自 PC 端的字符时，会触发中断，然后在串口中断服务程序中置起串口接收数据标志位，并最终在主程序中进行处理。中断服务程序中的操作要越简单越好，常规的操作 都是中断进来后置标志位然后在主程序查询标志位信息以决定是否去执行相应的操作。

4. 按键消抖工作原理

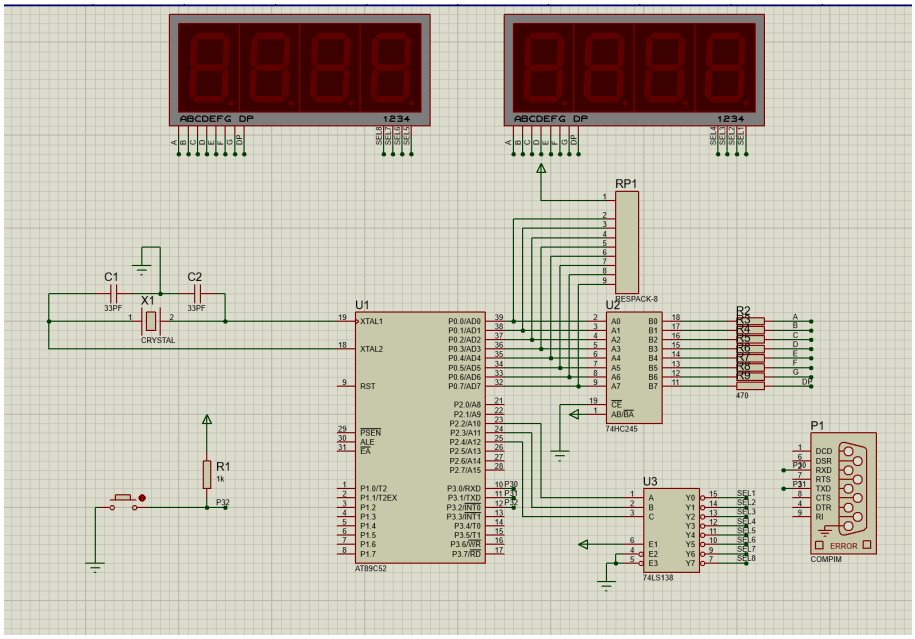
为了克服按键触点机械抖动所致的检测误判，必须采取消抖措施，本次实验依然可以采用实验二和实验三的做法的处理方法，即添加按键状态记录来实现消抖。

三、实验设备及器材

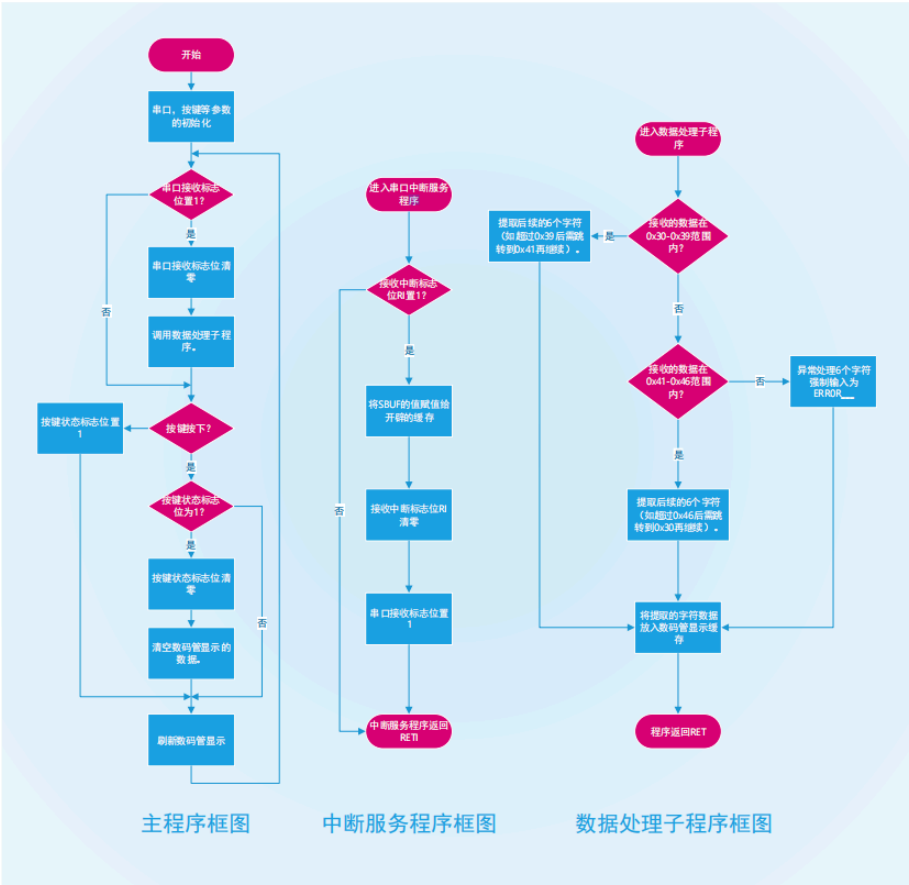
计算机，开发板：HC6800 - ES V2.0 开发板，USB 转串口线（开发板自带）

四、实验步骤和方法

1. 硬件电路设计



2. 软件流程图



3. 程序代码

C 语言版本（在例程基础上完善）

```c
#include <reg52.h>
#include <intrins.h>

unsigned char receive_data;
unsigned char uart_flag = 0;
unsigned char code
smgduan[16]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
sbit KEY = P3^2;   //清除按键连接至P3.2
sbit LSA = P2^2;
sbit LSB = P2^3;
sbit LSC = P2^4;
unsigned char disp[8] = {0};

void delay(unsigned int i)   //自定义延时函数
{
    while(i--);
}

uart_init()
{
  SCON = 0x50;
  TMOD = 0x20;
  PCON = 0x80;
  TH1 = 0xF3;
  TL1 = 0xF3;
  ES = 1;
  EA = 1;
  TR1 = 1;   //与中断相关的参数初始化
}
void ClearDisplay()   //清除按键的函数
{
    unsigned char i;
    for(i = 0;i < 8;i++)
    {
       disp[i] = 0x00;
    }
}
void Usart() interrupt 4
{
    if (RI==1)
    {
        receive_data = SBUF;      //接收的数据放入缓存
        RI = 0;      //清除接收中断标志位
        uart_flag = 1;      //置起串口接收标志位
    }
}
void FrameProcess()   //数据处理函数
{
  unsigned char i,temp;
  unsigned char valid_data;
  valid_data = receive_data;
  temp = valid_data;
  if (valid_data>=0x30 && valid_data<=0x39)
  {
    for (i=0;i<6;i++)
    {
      if (temp>=0x39)
      {
        temp = valid_data+8+i;
        disp[i]=smgduan[temp-0x41+10];
      }else{
        temp = valid_data+1+i;
        disp[i]=smgduan[temp-0x30];
      }
    }
    disp[6]=0x00;
```

```
    disp[7]=0x00;
  }else if (valid_data>=0x41 && valid_data<=0x46)
  {
    for (i=0;i<6;i++)
    {
        if (temp>=0x46 || temp<=0x39)
      {
        temp = valid_data-22+i;
        disp[i]=smgduan[temp-0x30];
      }else{
        temp = valid_data+1+i;
        disp[i]=smgduan[temp-0x41+10];
       }
    }
    disp[6]=0x00;
    disp[7]=0x00;
  }
  else
  {
    disp[0] = 0x79; //数码管字母 E
    disp[1] = 0x77; //数码管字母 R 用 A 代替
    disp[2] = 0x77; //数码管字母 R 用 A 代替
    disp[3] = 0x5C; //数码管字母 o
    disp[4] = 0x77; //数码管字母 R 用 A 代替
    disp[5] = 0x08; //数码管下线
    disp[6] = 0x08; //数码管下线
    disp[7] = 0x08; //数码管下线
  }
}

void DigDisplay() //数码管显示函數
{
    static unsigned char i = 0;
    for(i=0;i<8;i++)
    {
       switch(i) //位选，选择点亮的数码管，
       {
            case(0):
                LSA=0;LSB=0;LSC=0; break;//显示第 1 位
            case(1):
                LSA=1;LSB=0;LSC=0; break;//显示第 2 位
            case(2):
                LSA=0;LSB=1;LSC=0; break;//显示第 3 位
            case(3):
                LSA=1;LSB=1;LSC=0; break;//显示第 4 位
            case(4):
                LSA=0;LSB=0;LSC=1; break;//显示第 5 位
            case(5):
                LSA=1;LSB=0;LSC=1; break;//显示第 6 位
            case(6):
                LSA=0;LSB=1;LSC=1; break;//显示第 7 位
            case(7):
                LSA=1;LSB=1;LSC=1; break;//显示第 8 位
       }
     P0=disp[7-i];    //发送数据至P0口
     delay(100);     //间隔一段时间扫描
     P0=0x00;       //消隐
  }
}

void keyscan()
{
    static unsigned char key_state = 0; // 静态变量记录按键状态
    if (KEY == 0) {            // 检测按键是否按下（低电平有效）
        if (key_state == 0) {   // 首次检测到按下
            key_state = 1;      // 标记为“已按下”
            delay(20);          // 延时20ms消抖（根据实际调整）
            if (KEY == 0) {     // 再次确认按键仍被按下
                ClearDisplay(); // 执行清除显示操作
            }
```

```
        }
    } else {
        key_state = 0;          // 按键释放，重置状态
    }
}

void main()
{
  uart_init();
  while (1)
    {
      keyscan();
      DigDisplay();
      if (uart_flag)
      {
        uart_flag=0;
        FrameProcess();
        DigDisplay();
      }
   }
}
```

汇编代码版本

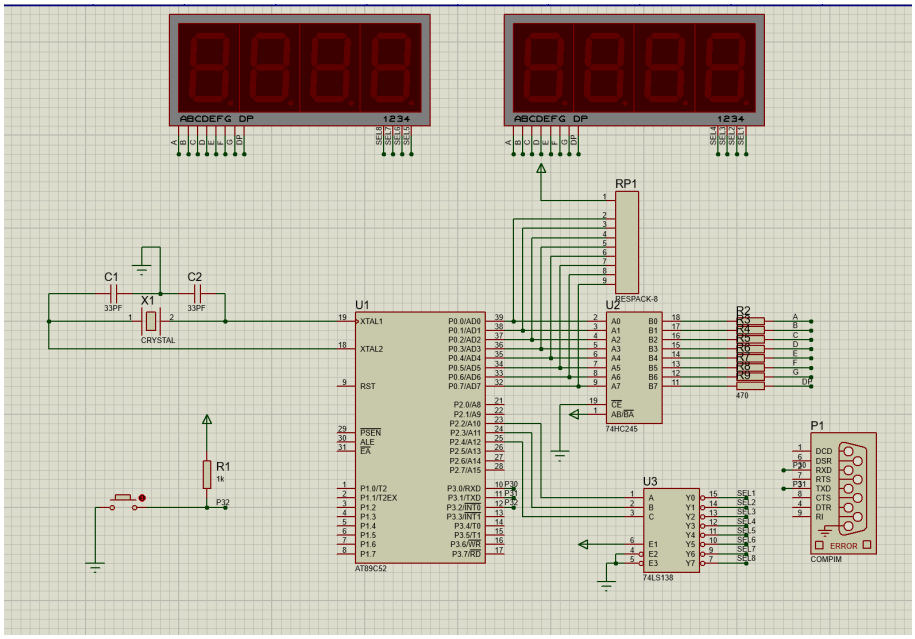
```
        ORG 0000H
        LJMP MAIN
        ORG 0023H
        LJMP PINT


        ORG 0100H
MAIN:   MOV TMOD,#20H   ;定时器设置为模式2，自动装填
        MOV TH1, #0F3H
        MOV TL1, #0F3H
        SETB TR1
        MOV SCON, #50H  ;串口设置为方式1，允许输入
        MOV PCON, #80H
        SETB EA
        SETB ES          ;允许中断
        MOV R1, #0FFH    ;R1储存输入的字符，#0FFH为没有输入，不符合则为#0F0H
        MOV R2, #00H
        MOV R3, #00H   ;以上两个用于辅助判断输入的数字是否违规
        MOV R4, #00H   ;清零状态指示，0为清零，1为未清零
        MOV R5, #0FFH  ;延时
        MOV R0, #00H     ;输出数字时用于暂存
        MOV DPTR, #0A00H


LOOP:   CJNE R4, #00H, SHOW ;清零状态，循环等待
        MOV P2, #1CH
        MOV P0, #00H
        JMP LOOP

SHOW:   MOV P1, #0FFH
        CLR P1.3
        JNB P1.7, CLEAR
        CJNE R1, #0F0H, NORMAL
        JMP ERROR

CLEAR: MOV R4, #00H
        LJMP LOOP

ERROR: MOV P2, #1CH ;E
H0:    MOV P0, #79H
        DJNZ R5, H0
        MOV P2, #18H ;R
H1:    MOV P0, #50H
     DJNZ R5, H1
     MOV P2, #14H ;R
```

```
H2:  MOV P0, #50H
     DJNZ R5, H2
     MOV P2, #10H ;O
H3:  MOV P0, #5CH
     DJNZ R5, H3
     MOV P2, #0CH ;R
H4:  MOV P0, #50H
     DJNZ R5, H4
     MOV P2, #08H ;_
H5:  MOV P0, #08H
     DJNZ R5,H5
     MOV P2, #04H ;_
H6:  MOV P0, #08H
     DJNZ R5, H6
     MOV P2, #00H ;_
H7:  MOV P0, #08H
     DJNZ R5, H7

     LJMP SHOW

NORMAL:MOV A, R1
       MOV R0, A
       MOV P2, #1CH    ;正常显示
       MOV A, R0
       INC R0
       MOVC A, @A+DPTR
H8:  MOV P0, A
     DJNZ R5, H8
     MOV P2, #18H
     MOV A, R0
     INC R0
     MOVC A, @A+DPTR
H9:  MOV P0, A
     DJNZ R5, H9
     MOV P2, #14H
     MOV A, R0
     INC R0
     MOVC A, @A+DPTR
HA:  MOV P0, A
     DJNZ R5, HA
     MOV P2, #10H
     MOV A, R0
     INC R0
     MOVC A, @A+DPTR
HB:  MOV P0, A
     DJNZ R5, HB
     MOV P2, #0CH
     MOV A, R0
     INC R0
     MOVC A, @A+DPTR
HC:  MOV P0, A
     DJNZ R5, HC
     MOV P2, #08H
     MOV A, R0
     INC R0
     MOVC A, @A+DPTR
HD:  MOV P0, A
     DJNZ R5, HD
     LJMP SHOW

;输入中断响应

PINT:  CLR RI            ;禁止输入
       MOV A, SBUF
       LCALL CHOSE       ;判断是否在范围内
       RETI

;判断输入字符

CHOSE: MOV R2, #30H
       MOV R3, #41H

CHOSE1:MOV B, R2
```

```
        CJNE A, B, LOOP1  ;0到9
        SUBB A, #2FH
        MOV R1, A
        JMP RETURN
LOOP1: INC R2
        CJNE R2, #03AH, CHOSE1  ;判断不是数字，跳至判断A到F

CHOSE2:MOV B, R3
        CJNE A, B, LOOP2
        SUBB A, #36H
        MOV R1,A
        JMP RETURN
LOOP2: INC R3
        CJNE R3, #047H, CHOSE2  ;A到F

        MOV R1, #0F0H
RETURN:
        MOV R0, A
        MOV R4, #01H    ;开始显示
        RET

        ORG 0A00H    ;建表
        DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH, 77H, 7CH, 39H, 5EH, 79H, 71H,
3FH, 06H, 5BH, 4FH, 66H, 6DH
```

五、实验结果与分析

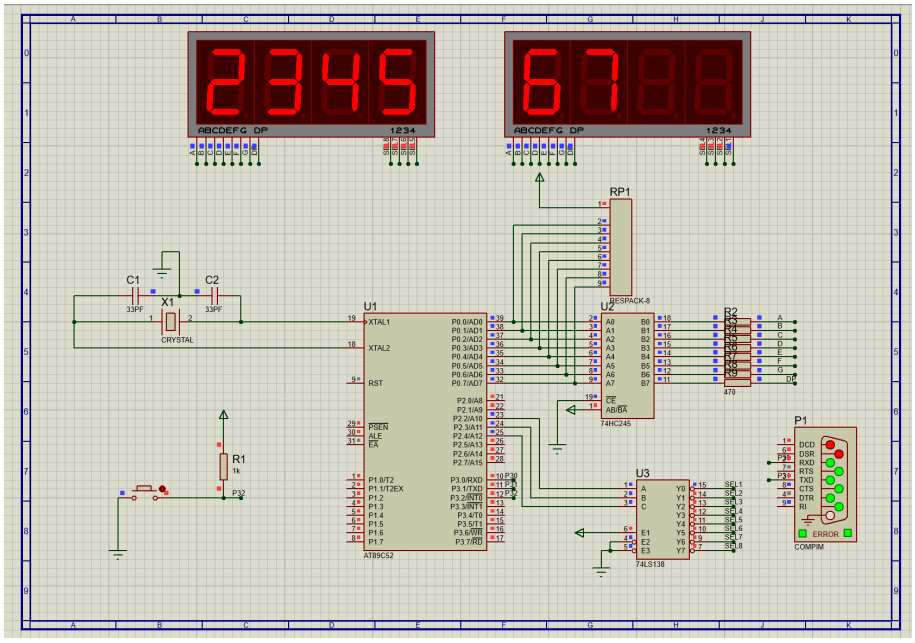
1. 硬件电路设计



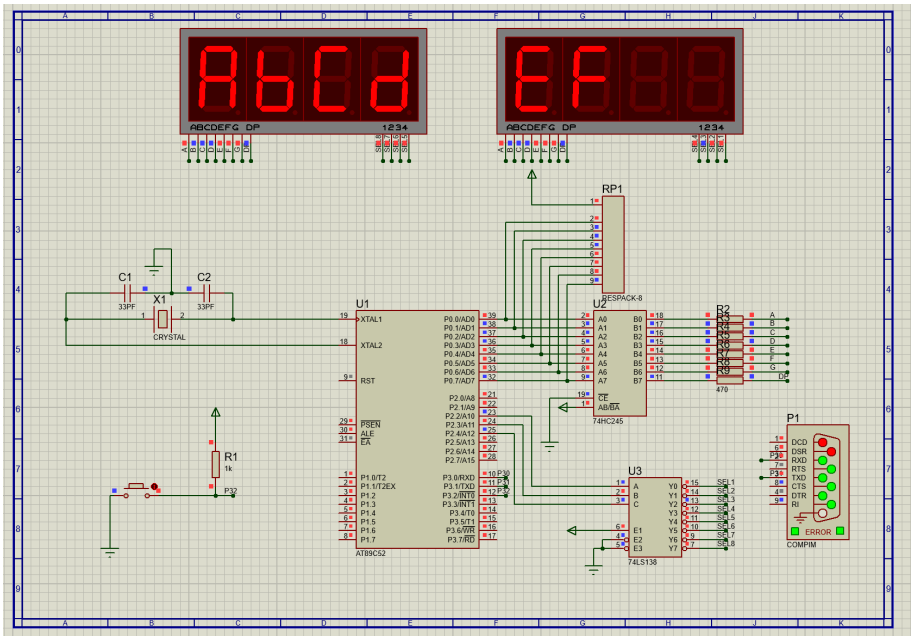
2. 软件仿真运行结果

分别向虚拟串口 COM2 输入 1，9，C，！字符

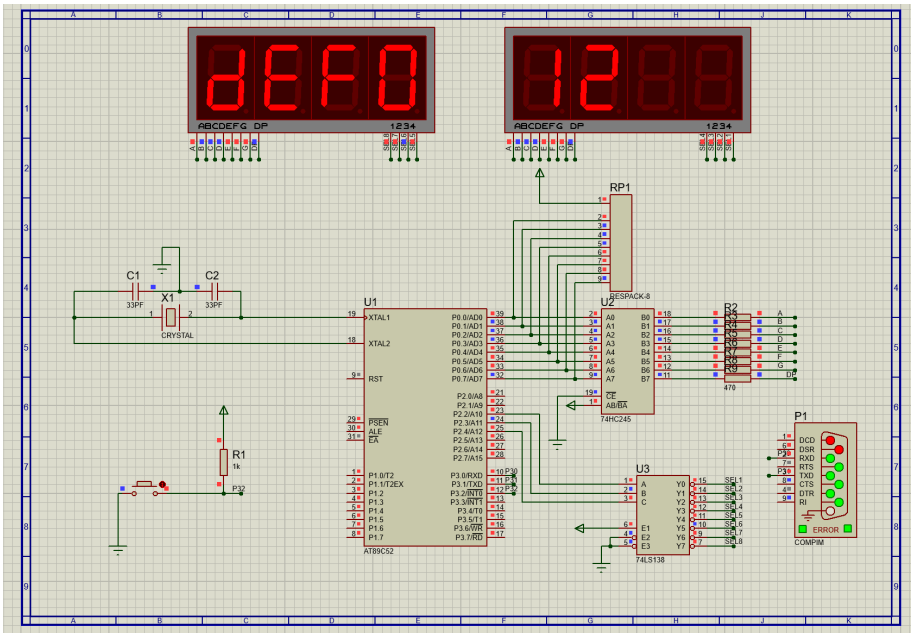
输入 1，仿真结果：



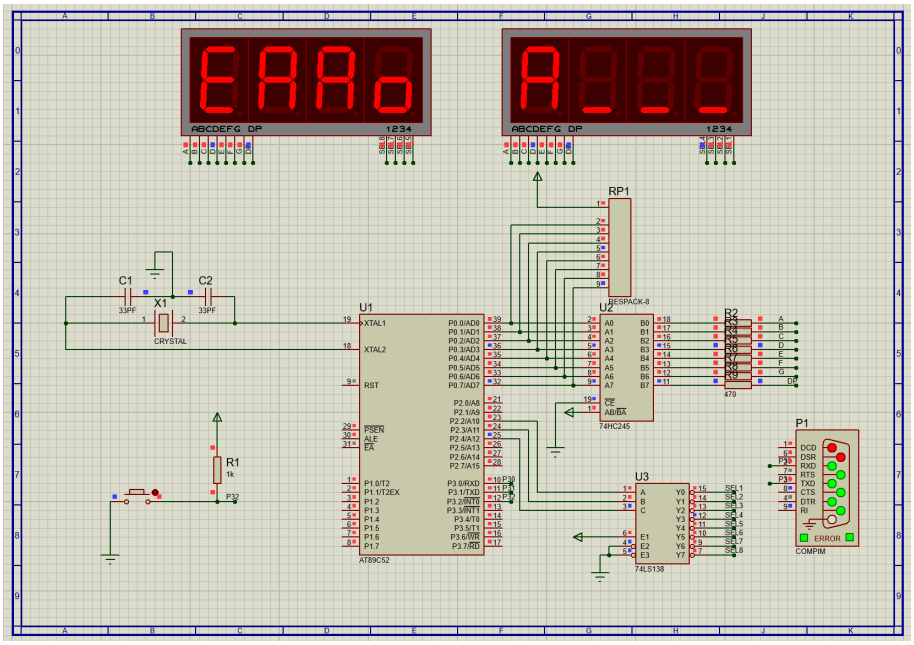
输入 9，仿真结果：



输入 C，仿真结果：

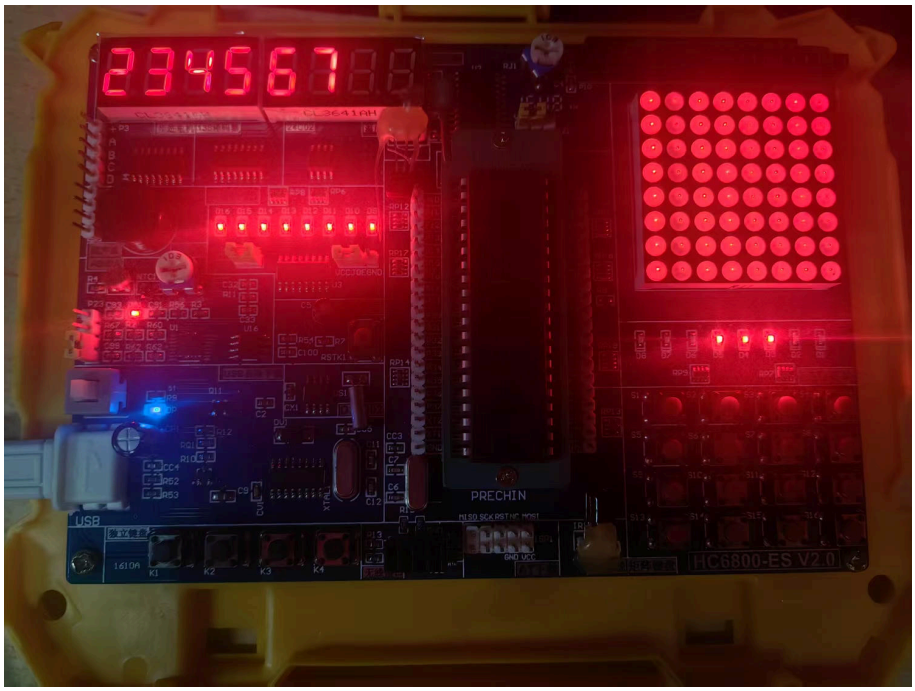


输入!，仿真结果：

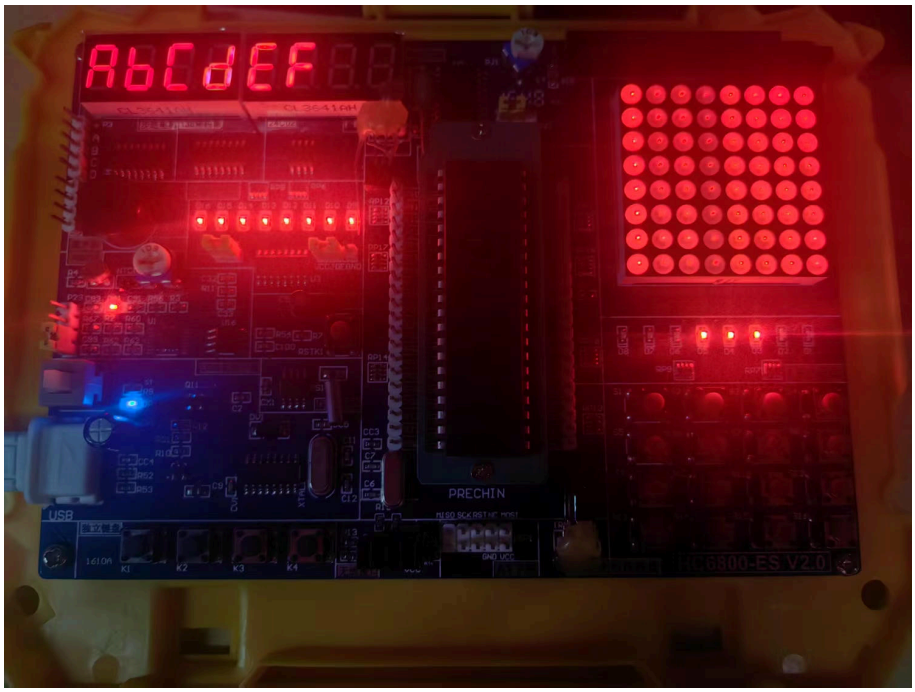


3. 开发板实际运行结果

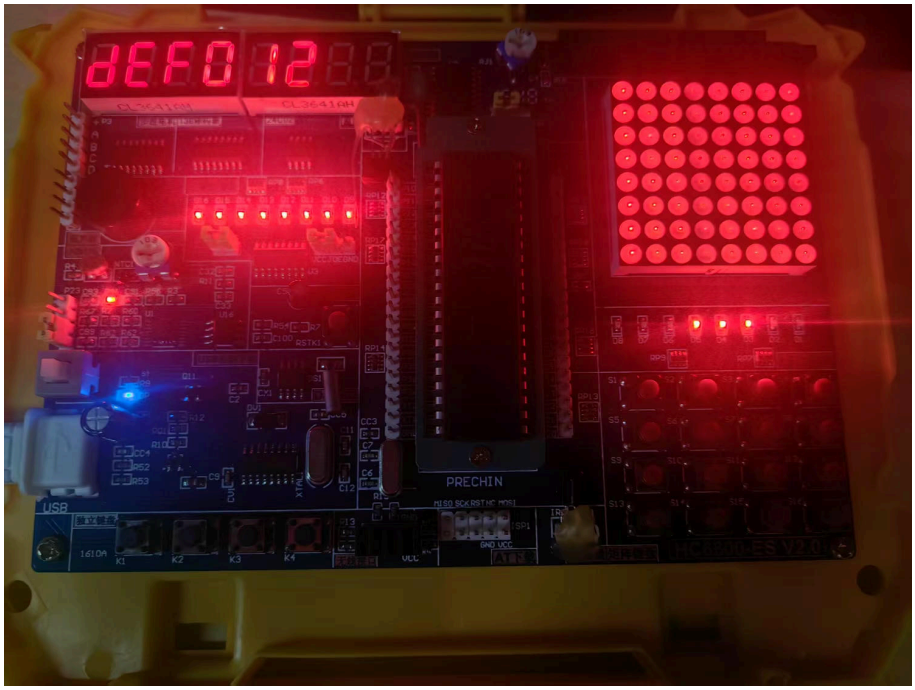
输入 1，开发板运行结果：



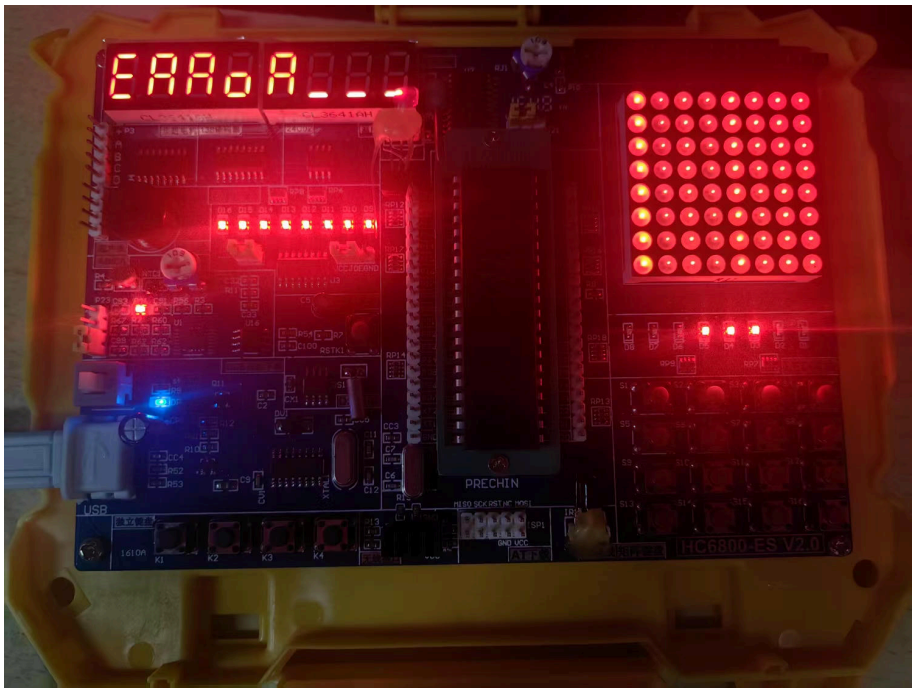
输入 9，开发板运行结果：



输入 C，开发板运行结果：



输入!，开发板运行结果：



六、扩展思考题

实验 2 消抖方法的不足之处在于：未处理抖动时间，依赖单次电平检测。

实验 2 仅通过状态变量判断电平变化的边沿，但未在检测到低电平后等待抖动结束（如延时 10-20ms）。若按键抖动期间电平在高低之间震荡，可能导致闭合时首次低电平被检测为有效按下，但后续抖动的高电平可能误触发状态变量重置，导致计数丢失；断开时抖动的低电平可能被误判为再次按下，尤其是在快速连续按键时易出现误计数。

改进的方法是增加延时函数:

```
void delay(unsigned int i)   //自定义延时函数
{
    while(i--);
}
void keyscan()
{
    static unsigned char key_state = 0; // 静态变量记录按键状态
    if (KEY == 0) {             // 检测按键是否按下（低电平有效）
        if (key_state == 0) {   // 首次检测到按下
            key_state = 1;      // 标记为“已按下”
            delay(20);          // 延时20ms消抖（根据实际调整）
            if (KEY == 0) {     // 再次确认按键仍被按下
                ClearDisplay(); // 执行清除显示操作
            }
        }
    } else {
        key_state = 0;          // 按键释放，重置状态
    }
}
```

导入到开发板测试后消抖效果较好

七、讨论与心得

本次实验最大的收获是了解了动态显示的二极管的操作思路以及相关的代码编写，同时本实验通过虚拟串口调试仿真和实际开发板的运行，熟悉了我关于 51 单片机中断和数码管显示的相关知识，最后将软件程序的运行结果呈现在 51 单片机的开发板上时还是比较有成就感的。最后，本次实验的汇编程序设计多次用到了判断跳转的语句，但是由于微机的结构限制无法像 C 语言那样一视同仁地使用 else-if 语句，而是要灵活使用 JZ、JC、CJNE、DJNZ 等相关语句，对于我而言仍然有比较大的难度。