项目三

篮球记分牌

3.1 项目设计

3.1.1 项目要求

设计一个篮球电子记分牌，符合以下要求：①采用单片机 AT89C52 进行控制；②利用 LED 数码管显示参赛球队比分；③具备加分、减分功能。

通过本项目的设计开发，了解中断的基本概念、中断的执行过程以及中断的控制方法，能利用中断技术编写单片机应用程序；了解单片机的键盘接口技术并能加以应用。

3.1.2 电路设计

1. 设计思路

篮球电子记分牌主要包括单片机控制、比分显示和按键 3 个模块。通过这 3 个模块的协调工作，完成相应的计分控制和显示功能，如图 3-1 所示。

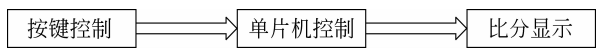

图 3-1　篮球记分牌设计思路

2. 器件清单

篮球记分牌器件清单如表 3-1 所示。

表 3-1　篮球记分牌器件清单

功能块	器件标号	器件名称	KEYWORDS	参数值	数量
主控	U1	微处理器	AT89C52		1
比分	LED1、LED2	LED 数码管	7SEG-MPX2-CA		2
	Q1～Q4	三极管	2N2905		4
	R_1～R_8	电阻	MINRES270R	270Ω	8
	R_{11}～R_{18}	电阻	MINRES5K1	5.1kΩ	8

续表

功能块	器件标号	器件名称	KEYWORDS	参数值	数量
按键	K1～K8	按键	BUTTON		8
	R_9～R_{10}	电阻	MINRES5K1	5.1kΩ	2
	U2	逻辑门	74LS08		1
复位	K0	按键	BUTTON		1
	R_0	电阻	RES	270Ω	1
	C_1	电容	PCELECT10U50V	10μF	1
时钟	X1	晶振	CRYSTAL	12MHz	1
	C_2～C_3	电容	CAP	30pF	2

注：表中 Q1～Q4 三极管 2N2905 仅用于仿真，实际应采用 9015 三极管。

3. 电路原理

篮球记分牌硬件电路图如图 3-2 所示。

提示：

① 为了画图方便，本电路图中没有画出时钟电路，但在做实物时，时钟电路不可或缺。

② 若采用片内程序存储器 ROM，虽然仿真时$\overline{EA}$没有要求，但在做实物时，必须接高电平＋5V。

3.1.3　程序设计

1. 程序流程

篮球记分牌程序流程图如图 3-3 所示。

2. 程序代码

1）汇编代码

```
SEG_OUT  EQU  P2          ;定义数码管段码端口
BIT_OUT  EQU  P1          ;定义数码管位选码端口
KEY_IO   EQU  P0          ;定义按键扫描端口
SCORE_A  EQU  50H         ;定义 SCORE_A 及其存储位置
SCORE_B  EQU  51H         ;定义 SCORE_B 及其存储位置
SCORE_A1 EQU  52H         ;定义 SCORE_A1 及其存储位置
SCORE_A0 EQU  53H         ;定义 SCORE_A0 及其存储位置
SCORE_B1 EQU  54H         ;定义 SCORE_B1 及其存储位置
SCORE_B0 EQU  55H         ;定义 SCORE_B0 及其存储位置
KEY_SAVE EQU  56H         ;定义 KEY_SAVE 及其存储位置
FLAG     BIT  00H         ;定义按键处理标志位 FLAG 及其存储位置

ORG  0000H                ;程序开始入口
LJMP START                ;跳至 START 主程序设置
ORG  0003H                ;中断服务子程序入口
LJMP INT0                 ;跳至中断服务子程序
```

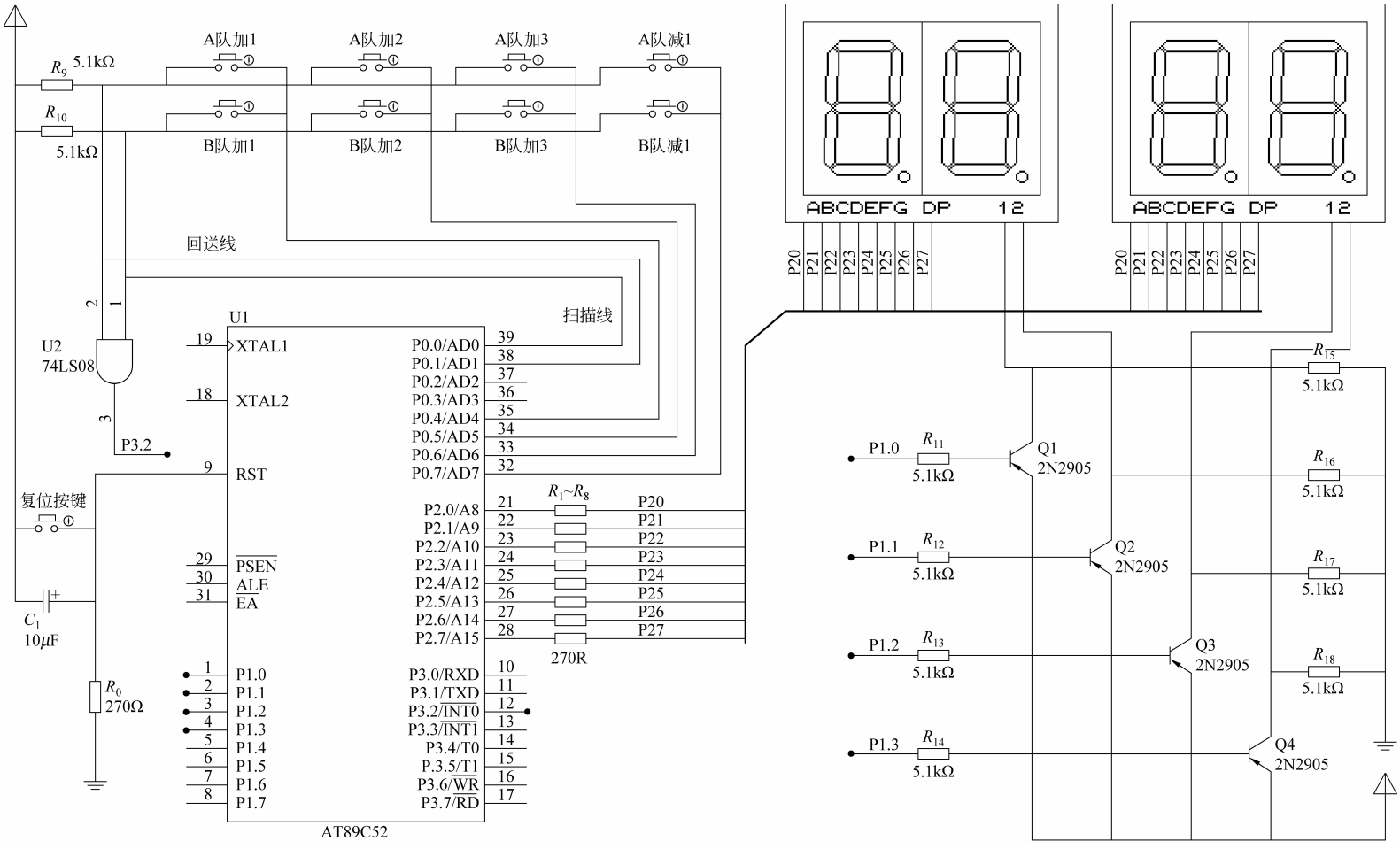

图 3-2　篮球记分牌硬件电路图

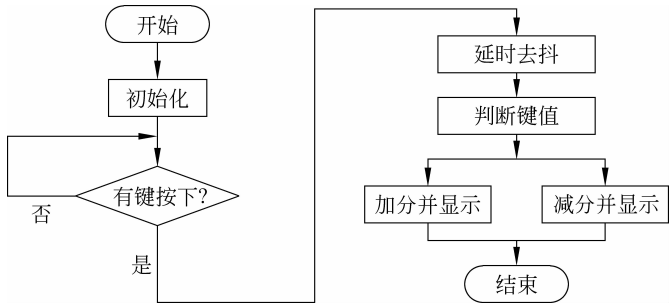

图 3-3　程序流程图

```
;*********************************************************************************************
;主程序
;*********************************************************************************************
          ORG   0030H              ;主程序开始位置
START:    SETB  EX0                ;开外部中断 0
          SETB  EA                 ;开全局中断
          CLR   IT0                ;外部中断 0 下降沿触发
          MOV   SP, #60H           ;初始化堆栈
          MOV   SCORE_A, #0        ;SCORE_A 初始化为“0”
          MOV   SCORE_B, #0        ;SCORE_B 初始化为“0”
MAIN:     MOV   KEY_IO, #03H       ;按键端口初始化赋值为 0000 0011, 便于矩阵键扫描
          LCALL DISP               ;调用显示函数
          SETB  EX0                ;开外部中断 0
          JNB   P3.2, MAIN         ;P3.2 为“0”时调至 MAIN, 为“1”时执行下面的指令
          MOV   KEY_SAVE, #100     ;赋值 KEY_SAVE 为“100”
          CLR   FLAG               ;清除按键处理标准位
          SJMP  MAIN               ;循环跳至 MAIN
;*********************************************************************************************
;显示模块
;*********************************************************************************************
DISP:
          MOV  A, SCORE_A          ;将 SCORE_A 的值赋给 A
          MOV  B, #10              ;将 B 的值赋为 10
          DIV  AB                  ;A 除以 B, 结果存于 A,余数存于 B
          MOV  SCORE_A1, A         ;将 A 即十位, 赋给 SCORE_A1
          MOV  SCORE_A0, B         ;将 B 即个位, 赋给 SCORE_A0
          MOV  A, SCORE_B          ;将 SCORE_B 的值赋给 A
          MOV  B, #10              ;将 B 的值赋为 10
          DIV  AB                  ;A 除以 B, 结果存于 A,余数存于 B
          MOV  SCORE_B1, A         ;将 A 即十位, 赋给 SCORE_B1
          MOV  SCORE_B0, B         ;将 B 即个位, 赋给 SCORE_B0
          MOV  R1, #0FEH           ;R1 寄存器赋值为 1111 1110
          MOV  R0, #SCORE_A1       ;将 R0 赋值为 SCORE_A1
          MOV  R2, #4              ;将 R2 赋值为 4, 用于后面 4 位数码管扫描显示
DIS_LOOP:
          MOV  BIT_OUT, #0FFH      ;先将位选端口赋值为 1111 1111, 即全灭
          MOV  DPTR, #TAB          ;将 TAB 的地址赋给数据指针 DPTR
```

```
        MOV   A, @R0            ;读取 R0 里面的内容, 赋给 A
        MOVC  A, @A+DPTR        ;读取 A+DPTR 地址里面的内容, 赋给 A
        MOV   SEG_OUT, A        ;将 A 送到段码端口
        MOV   A, R1             ;将 R1 的数赋给 A
        MOV   BIT_OUT, A        ;将 A 送到位选码端口
        LCALL DELAY             ;调用延时函数
        MOV   A, R1             ;将 R1 赋值给 A
        RL    A                 ;左移 A, 低位补 1
        MOV   R1, A             ;再将 A 赋值给 R1
        INC   R0                ;R0 自加 1
        DJNZ  R2, DIS_LOOP      ;R2 自减 1,不为 0 时,跳至 DIS_LOOP;为 0,就往下执行
        RET                     ;子程序返回
TAB:    DB    0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H
                                ;0~9 的数码管段码存储位置
;*******************************************************************************
;延时子程序
;*******************************************************************************
DELAY:  MOV   45H, #03H         ;45H 寄存器赋值为 0x03
DEL1:   MOV   46H, #0FFH        ;46H 寄存器赋值为 0xff
DEL2:   NOP                     ;空指令
        NOP                     ;空指令
        DJNZ  46H, DEL2         ;46H 的内容自减 1,不为 0,跳至 DEL2;为 0,执行下一步
        DJNZ  45H, DEL1         ;45H 的内容自减 1,不为 0,跳至 DEL1;为 0,执行下一步
        RET                     ;子程序返回
;*******************************************************************************
;键盘扫描模块
;*******************************************************************************
KEYSCAN:
        MOV    KEY_IO, #03H     ;先将按键口赋值为 0000 0011
        MOV    A, KEY_IO        ;读取键盘状况
        CJNE   A, #03H, KS1     ;有键按下
        LJMP   KEYSCAN_RET      ;无键按下
KS1:    MOV    B, A             ;存行值
        MOV    KEY_IO, #0E3H    ;扫描第 0 列
        MOV    A, KEY_IO        ;将按键状态赋给 A
        CJNE   A, #0E3H, KS2    ;判断 A 是否为 0xe3, 不是就转移到 KS2
        MOV    KEY_IO, #0D3H    ;扫描第 1 列
        MOV    A, KEY_IO        ;将按键状态赋给 A
        CJNE   A, #0D3H, KS2    ;判断 A 是否为 0xd3, 不是就转移到 KS2
        MOV    KEY_IO, #0B3H    ;扫描第 2 列
        MOV    A, KEY_IO        ;将按键状态赋给 A
        CJNE   A, #0B3H, KS2    ;判断 A 是否为 0xb3, 不是就转移到 KS2
        MOV    KEY_IO, #73H     ;扫描第 3 列
        MOV    A, KEY_IO        ;将按键状态赋给 A
        CJNE   A, #73H, KS2     ;判断 A 是否为 0x73, 不是就转移到 KS2
        LJMP   KEYSCAN_RET      ;出错
KS2:    ANL    A, #0F0H         ;列
        ORL    A, B             ;按下键的行列值
        MOV    B, A             ;暂存键值到 B
```

```
        MOV    R4, #8            ;8 个键
        MOV    R3, #0            ;键码初值
        MOV    DPTR, #KEYTAB     ;KEYTAB 地址赋给 DPTR
KS3:    MOV    A, R3             ;A 的值赋为 R3
        MOVC   A, @A+DPTR        ;从顺序码表中取键值
        CJNE   A, B, KS5         ;与按下键键值比较
        MOV    A, R3             ;A 的值赋为 R3
        CJNE   A, KEY_SAVE, KEYSCAN_INI    ;KEY_SAVE 与 A 比较
        JB     FLAG, KEYSCAN_RET  ;判断 FLAG 是否转移。FLAG=1,则转移
        LCALL  KEY_FUNC          ;调用按键功能处理程序
        SETB   FLAG              ;处理标志位置“1”
        MOV    KEY_IO, #03H      ;相等, 则完成以下步骤
        AJMP   KEYSCAN_RET       ;跳至 KEYSCAN_RET
KS5:    INC    R3                ;不相等,则继续访问键值表
        DJNZ   R4, KS3           ;R4 减 1。不为 0,跳转继续扫描
KEYSCAN_INI:
        MOV    KEY_SAVE, R3      ;R3 赋值给 KEY_SAVE
        CLR    FLAG              ;清零按键处理标志位
KEYSCAN_RET:                     ;按键函数返回
        RET                      ;按键函数返回
KEYTAB:
        DB  0E1H, 0D1H, 0B1H, 71H    ;按键顺序码
        DB  0E2H, 0D2H, 0B2H, 72H
;*****************************************************************************
;按键处理
;*****************************************************************************
KEY_FUNC:
        MOV    A, R3             ;得键码
        ADD    A, R3             ;A 与 R3 相加, 赋值给 A
        MOV    DPTR, #KEY_JMP    ;KEY_JMP 赋给地址指针 DPTR
        JMP    @A+DPTR           ;跳至 A+DPTR 的地址位置
KEY_JMP:
        AJMP   KEY_0             ;跳至 KEY_0
        AJMP   KEY_1             ;跳至 KEY_1
        AJMP   KEY_2             ;跳至 KEY_2
        AJMP   KEY_3             ;跳至 KEY_3
        AJMP   KEY_4             ;跳至 KEY_4
        AJMP   KEY_5             ;跳至 KEY_5
        AJMP   KEY_6             ;跳至 KEY_6
        AJMP   KEY_7             ;跳至 KEY_7
        AJMP   KEY_FUNC_RET      ;跳至 KEY_FUNC_RET
KEY_0:
        MOV    A, SCORE_A        ;SCORE_A 赋值给 A
        ADD    A, #1             ;A 加 1
        MOV    SCORE_A, A        ;将 A 赋给 SCORE_A
        CJNE   A, #100, KEY_FUNC_RET    ;当 A 不等于 100 时,跳至 KEY_FUNC_RET
        MOV    SCORE_A, #0       ;SCORE_A 赋值为 0
        AJMP   KEY_FUNC_RET      ;跳至 KEY_FUNC_RET
KEY_1:
```

```
        MOV    A, SCORE_A       ;SCORE_A 赋值给 A
        ADD    A, #2            ;A 加 2
        MOV    SCORE_A, A       ;将 A 赋给 SCORE_A
        SUBB   A, #100          ;A 减 100
        JC     KEY_FUNC_RET     ;A 不为 0 时,转移到 KEY_FUNC_RET
        MOV    SCORE_A, A       ;将 A 赋给 SCORE_A
        AJMP   KEY_FUNC_RET     ;跳至 KEY_FUNC_RET
KEY_2:
        MOV    A, SCORE_A       ;SCORE_A 赋值给 A
        ADD    A, #3            ;A 自加 3
        MOV    SCORE_A, A       ;将 A 赋给 SCORE_A
        SUBB   A, #100          ;A 减 100
        JC     KEY_FUNC_RET     ;A 不为 0 时,转移到 KEY_FUNC_RET
        MOV    SCORE_A, A       ;将 A 赋给 SCORE_A
        AJMP   KEY_FUNC_RET     ;跳至 KEY_FUNC_RET
KEY_3:
        MOV    A, SCORE_A       ;SCORE_A 赋值给 A
        SUBB   A, #1            ;A 自减 1
        MOV    SCORE_A, A       ;将 A 赋给 SCORE_A
        JNC    KEY_FUNC_RET     ;A 不为 0 时,转移到 KEY_FUNC_RET
        MOV    SCORE_A, #0      ;将 SCORE_A 赋值为 0
        AJMP   KEY_FUNC_RET     ;跳至 KEY_FUNC_RET
KEY_4:
        MOV    A, SCORE_B       ;SCORE_B 赋值给 A
        ADD    A, #1            ;A 自加 1
        MOV    SCORE_B, A       ;将 A 赋给 SCORE_B
        CJNE   A, #100, KEY_FUNC_RET    ;当 A 不等于 100 时,跳至 KEY_FUNC_RET
        MOV    SCORE_B, #0      ;将 0 赋给 SCORE_B
        AJMP   KEY_FUNC_RET     ;跳至 KEY_FUNC_RET
KEY_5:
        MOV    A, SCORE_B       ;SCORE_B 赋值给 A
        ADD    A, #2            ;A 自加 2
        MOV    SCORE_B, A       ;将 A 赋给 SCORE_B
        SUBB   A, #100          ;A 减 100
        JC     KEY_FUNC_RET     ;A 不为 0 时,转移到 KEY_FUNC_RET
        MOV    SCORE_B, A       ;将 A 赋给 SCORE_B
        AJMP   KEY_FUNC_RET     ;跳至 KEY_FUNC_RET
KEY_6:
        MOV    A, SCORE_B       ;SCORE_B 赋值给 A
        ADD    A, #3            ;A 自加 3
        MOV    SCORE_B, A       ;将 A 赋给 SCORE_B
        SUBB   A, #100          ;A 减 100
        JC     KEY_FUNC_RET     ;A 不为 0 时,转移到 KEY_FUNC_RET
        MOV    SCORE_B, A       ;将 A 赋给 SCORE_B
        AJMP   KEY_FUNC_RET     ;跳至 KEY_FUNC_RET
KEY_7:
        MOV    A, SCORE_B       ;SCORE_B 赋值给 A
        SUBB   A, #1            ;A 自减 1
        MOV    SCORE_B, A       ;将 A 赋给 SCORE_B
```

```
        JNC     KEY_FUNC_RET      ;A 不为 0 时,转移到 KEY_FUNC_RET
        MOV     SCORE_B, #0       ;将 0 赋给 SCORE_B
KEY_FUNC_RET:
        RET                       ;按键处理子程序返回
;*******************************************************************************
;外部中断 0 中断服务子程序
;*******************************************************************************
INT0:
        PUSH    ACC               ;累加器的内容保存
        PUSH    B                 ;B 寄存器的内容保存
        PUSH    PSW               ;PSW 寄存器的内容保存
        CLR     EX0               ;关闭外部中断
        LCALL   KEYSCAN           ;调用按键扫描函数
        POP     PSW               ;读出 PSW 寄存器的内容
        POP     B                 ;读出 B 寄存器的内容
        POP     ACC               ;读出累加器的内容
        RETI                      ;中断返回
        END                       ;程序执行结束
```

2）C 代码

```
#include "reg51.h"                //51 头文件
#include "intrins.h"              //_nop_();延时函数用
#define  uchar  unsigned char     //无符号字符型宏定义
#define  uint   unsigned int      //无符号整型宏定义
#define  SEG_OUT    P2            //段码输出口
#define  BIT_OUT    P1            //扫描口
#define  KEY_IO     P0            //键盘接口
sbit   dot=P2^7;                  //LED 小数点控制
uchar code tab[12]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0xFF,
0xBF};
/*共阳 LED 段码表 "0"  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9" "不亮" "-" */
uchar data  score_a=0,score_b=0;          //双方分数变量
uchar data  dis_buff[4]={0,0,0,0};        //待显示单元数据,共 4 个数据,即显示数组
uchar data  keytemp,key=100,keysave=100; //键值存放
bit  data  flag=0;                        //按键处理标志位
//-------------------------------------------------------------------------
//1ms 延时函数
//-------------------------------------------------------------------------
delay1ms(int t)
{
      int i,j;                    //延时变量
      for(i=0;i<t;i++)            //t 表示循环次数
            for(j=0;j<120;j++);   //for 循环 120 次,共 1ms
}
//-------------------------------------------------------------------------
//LED 显示动态扫描函数
//-------------------------------------------------------------------------
display()
{
```

```
    char k;                          //for 扫描显示中的变量 k
    char m=0xfe;                     //m 初值为 1111 1110
    dis_buff[0]=score_a/10;          //送 score_a 的十位到显示数组里面
    dis_buff[1]=score_a%10;          //送 score_a 的个位到显示数组里面
    dis_buff[2]=score_b/10;          //送 score_b 的十位到显示数组里面
    dis_buff[3]=score_b%10;          //送 score_b 的个位到显示数组里面
    for(k=0;k<4;k++)                 //6 位 LED 扫描控制
    {
        BIT_OUT=0xff;                //先在位选端口送 1111 1111, 即全灭
        SEG_OUT=tab[dis_buff[k]];    //将数码管段码送到 8 个段码 IO 口
        BIT_OUT=m;                   //将 m 送到位选端口, 第一次时点亮第一位数码管
        delay1ms(3);                 //延时 3ms
        m=(m<<1)|0x01;               //将 m 左移 1 位, 高位补"1",在循环中依次点亮数码管
    }
}
//-------------------------------------------------------------------------------
//按键功能处理函数
//-------------------------------------------------------------------------------
void keyproc()
{
    switch(key)          //用 switch 函数判断当前 key 值的状态, 不同状态执行不同指令
    {
        case 0:score_a++;if(score_a==100)score_a=0;break;
                                                    //key=0 时, score_a 增 1
        case 1:score_a+=2;if(score_a>=100)score_a=100;break;
                                                    //key=1 时, score_a 增 2
        case 2:score_a+=3;if(score_a>=100)score_a-=100;break;
                                                    //key=2 时, score_a 增 3
        case 3:if(score_a!=0)score_a--;break;       //key=3 时, score_a 减 1
        case 4:score_b++;if(score_b==100)score_b=0;break;
                                                    //key=4 时, score_b 增 1
        case 5:score_b+=2;if(score_b>=100)score_b-=100;break;
                                                    //key=5 时, score_b 增 2
        case 6:score_b+=3;if(score_b>=100)score_b-=100;break;
                                                    //key=6 时, score_b 增 3
        case 7:if(score_b!=0)score_b--;break;       //key=7 时, score_b 减 1
        default:break;                              //都不符合时跳出循环
    }
}
//-------------------------------------------------------------------------------
//按键扫描函数
//-------------------------------------------------------------------------------
void keyscan()
{
    uchar i,t=0xef;                  //1110 1111
    KEY_IO=0x03;                     // KEY_IO 赋值为 0000 0011
    keytemp=(~KEY_IO)&0x03;          //将按键端口取反,再与上 0000 0011 赋值给 keytemp
                                     //即当有按键按下时, keytemp≠0
    if(keytemp!=0)                   //当 keytemp≠0 时
```

```
    {
        for(i=0;i<4;i++)                    //分 4 次循环扫描 4 列
        {
            KEY_IO=t;                       //先将第一条列线赋为“0”
            keytemp=(~KEY_IO)&0x03;         //读取按键端口状态
            if(keytemp!=0)                  //当 keytemp≠0 时
            {
                switch(keytemp)             //判断 keytemp 状态
                {
                    case 0x02:key=0*4+i;break;
                        //第一行按下时,keytemp 为 0x02,根据 i 判断第几列
                    case 0x01:key=1*4+i;break;
                        //第二行按下时,keytemp 为 0x01,根据 i 判断第几列
                    default:break;
                }
                if(key>=0&key<=7&key==keysave)
                                            //当 key 为 0~7, 并且 key=keysave 时
                {
                    if(flag!=1)             //当按键处理标准位不为“1”时
                    {
                        keyproc();          //调用按键处理函数
                        flag=1;             //按键处理标志位为“1”, 相当于有键按下
                    }
                    break;                  //跳出 if 循环
                }
                else
                {
                    keysave=key;            //将 key 的值赋给 keysave
                    flag=0;                 //按键处理标志位为“0”,相当于没有键按下
                    break;                  //跳出 if 循环
                }
            }
            t=(t<<1)|0x01;                  //t 左移 1 位, 高位补“1”, 扫描下一列
        }
    }
}
//-------------------------------------------------------------------------------
//外部中断 0 中断服务子函数
//-------------------------------------------------------------------------------
void ext0(void) interrupt 0 using 0         //通过一个与非门用中断扫描按键
{
    EX0=0;                                  //关闭外部中断
    keyscan();                              //按键扫描函数
}
//-------------------------------------------------------------------------------
//主函数
//-------------------------------------------------------------------------------
main()
{
```