实验报告

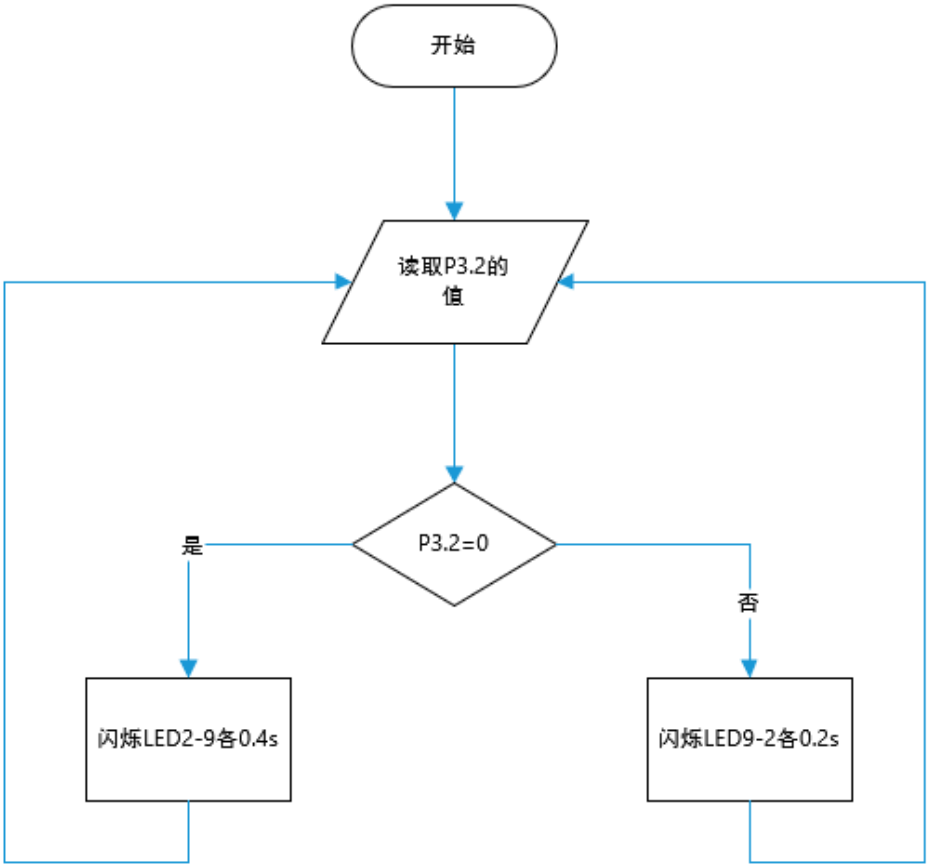
课后完成：根据3.3中的要求进行编程，简单描述程序的整体结构和各子函数的功能，并对程序的每一句进行简单备注；拍摄一到两张程序运行中的图片作为报告附图。

第一种情况：一次完整循环完毕后，来判断P3.2的状态，以决定循环的方向；

设计思路：

考虑程序需跑完一个循环再检测P3.2的值，故设计两个子程序，一个用于K0断开时的情形，一个用于K0闭合时的情形。以及一个延时程序。K0断开时，从将11111110赋给P0，保证灯从最低位也就是LED2开始闪烁，闪烁一次做一次左移，循环八次到达01111111即LED9，再回到开始检测P3.2的值，进行下一个循环。K1断开时，从将01111111赋给P0，保证灯从最高位也就是LED9开始闪烁，闪烁一次做一次右移，循环八次到达11111110即LED2，再回到开始检测P3.2的值，进行下一个循环。考虑到，两种不同的循环有着不同的闪烁时间，故需两种不同的延时循环，这里设计用一个地址存储延时函数最外层的循环次数，来改变闪烁时间。

流程图如下：



具体代码如下：

```
        ORG     0000H    ;  定位伪指令
        LJMP    START    ;  转移到主程序
        ORG     0030H    ;  定位伪指令
START:  MOV     P0, #0FFH;  将11111111赋值给P0口 关闭LED
        MOV     P1, #0EEH;  将11101110赋值给P1口 打开LEDS6 保证三极管导通
START1: MOV     R0, #8   ;  将8赋给R0 为循环次数
        MOV     A, P3    ;  把P3口的值赋给A
```

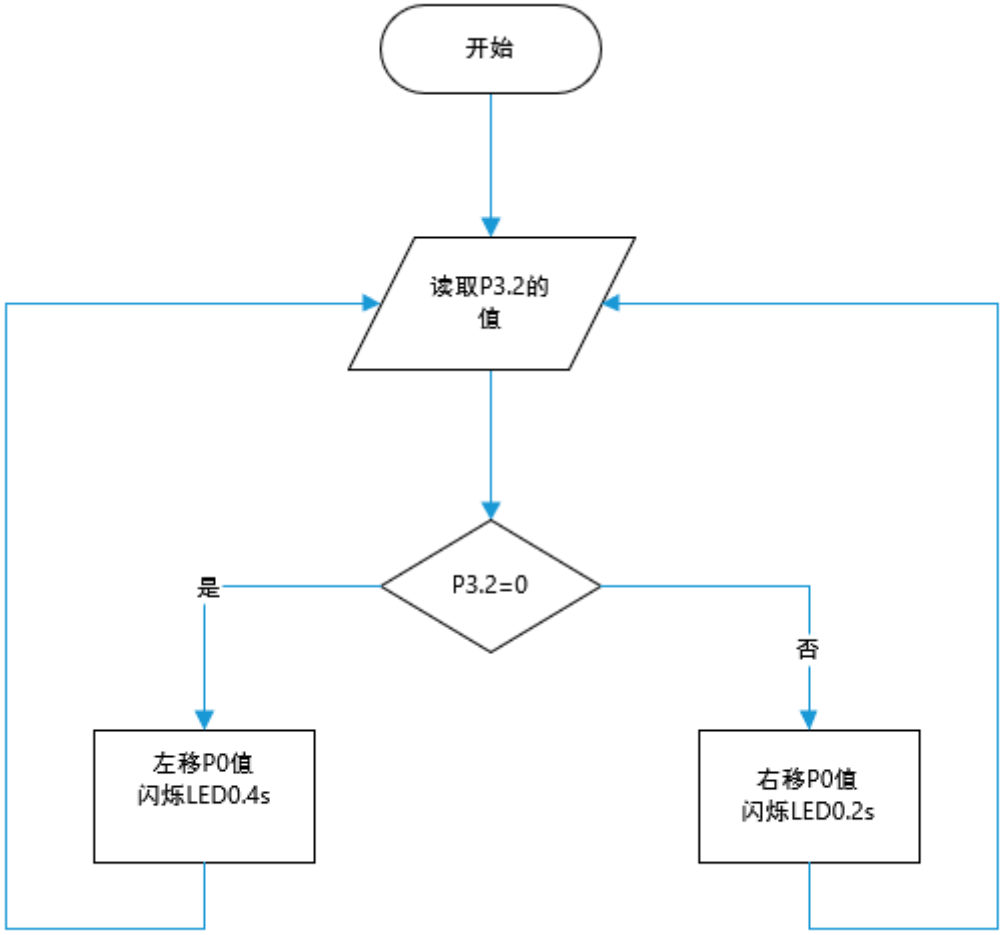
```
        ANL     A, #04H  ;  与00000100做位与运算 获得P3.2的值
        RR      A        ;  右移一位
        RR      A        ;  右移一位 得到0000000X X为P3.2的值
        MOV   DPTR, #TAB ;  将表格起始地址赋给DPTR
        MOVC  A, @A+DPTR ;  找到TAB中所需的地址
        JMP    @A+DPTR   ;  跳转至所需位置
TAB:    DB      PRG0-TAB ;  跳到PRG0
        DB      PRG1-TAB ;  跳到PRG1
PRG0:   MOV     A,#7FH   ;  将01111111赋给A
MLOOP0: MOV     P0,A     ;  将A的值赋给P0 开LED9
        MOV     31H,#2   ;  确定闪烁时间 延时函数最外层循环两次 即0.2s
        LCALL   DELAY    ;  调用延时函数
        RR      A        ;  右移A 接着亮右边的灯
        DJNZ R0,MLOOP0   ;  循环8次MLOOP0
        LJMP    START1   ;  回到主函数
PRG1:   MOV     A,#0FEH  ;  将11111110赋给A
MLOOP1: MOV     P0,A     ;  将A的值赋给P0 开LED2
        MOV     31H,#4   ;  确定闪烁时间 延时函数最外层循环四次 即0.4s
        LCALL   DELAY    ;  调用延时函数
        RL      A        ;  左移A 接着亮左边的灯
        DJNZ R0,MLOOP1   ;  循环8次MLOOP1
        LJMP    START1   ;  回到主函数
DELAY:  MOV    R1, 31H   ;  延时函数的第三重循环 循环次数由分支函数决定
DEL1:   MOV    R2, #200  ;  延时函数的第二重循环
DEL2:   MOV    R3, #229  ;  延时函数的第一重循环
DEL3:   DJNZ   R3, DEL3  ;  自己循环229次
        DJNZ   R2, DEL2  ;  循环DEL2 229*200次
        DJNZ   R1, DEL1  ;  循环DEL3 229*200*(31H)次
        RET              ;  回到调用处
        END              ;  程序结束
```

第二种情况：在一次还未做完时，实时的根据P3.2的状态来调整循环的方向。

考虑程序需每闪烁一次就需检测P3.2的值，同样设计两个子程序，一个用于K0断开时的情形，一个用于K0闭合时的情形。以及一个延时程序。这里考虑到进行移位操作时需对ACC进行操作，而在检测P3.2的值时也需对ACC的值进行改变，故设计利用两个地址去存储暂时ACC的值。为什么用两个地址而不用一个地址是因为在程序开始运行前，无法知晓K0是断开还是闭合。而K0断开和闭合对应的起始P0值不同（一个为11111110，一个为01111111）而只使用一个地址时，不能保证在小灯开始闪烁时位于我们需要的位置即LED9或者LED2.具体而言，假设只用一个地址32H存储A值时，若32H起始为11111110，则当整个单片机开始运行前，K0为闭合状态，则此时第一个亮起的LED为LED2而非LED9，不满足从高位起的要求。故选择多占用一条地址进行设计。

K0断开时，从将11111110赋给P0，保证灯从最低位也就是LED2开始闪烁，闪烁一次做一次左移，再回到开始检测P3.2的值，进行下一个循环。K1断开时，从将01111111赋给P0，保证灯从最高位也就是LED9开始闪烁，闪烁一次做一次右移，再回到开始检测P3.2的值，进行下一个循环。考虑到，两种不同的循环有着不同的闪烁时间，故需两种不同的延时循环，这里设计用一个地址存储延时函数最外层的循环次数，来改变闪烁时间。

流程图如下:



具体代码如下:

```
        ORG   0000H         ;  定位伪指令
        LJMP   START        ;  转移到主程序
        ORG   0030H         ;  定位伪指令
START:  MOV   P0, #0FFH     ;  将11111111赋值给P0口 关闭LED
        MOV   p1, #0EEH     ;  将11101110赋值给P1口 打开LEDS6 保证三极管导通
        MOV   32H,#0FEH     ;  用32H这个地址存储循环中A的值 起始为11111110 用于程序运行前K0为断开的情况
        MOV   33H,#7FH      ;  用33H这个地址存储循环中A的值 起始为01111111 用于程序运行前K0为闭合的情况
START1: MOV   A, P3         ;  将P3口的值赋给A
        ANL   A, #04H       ;  与00000100做位与运算 获得P3.2的值
        RR    A             ;  右移一位
        RR    A             ;  右移一位 得到0000000X X为P3.2的值
        MOV     DPTR, #TAB; 将表格起始地址赋给DPTR
        MOVC    A, @A+DPTR; 找到TAB中所需的地址
        JMP     @A+DPTR     ;  跳转至所需位置
TAB:    DB        PRG0-TAB  ;  跳到PRG0
        DB        PRG1-TAB  ;  跳到PRG1
PRG0:   MOV       A,33H     ;  将暂存的A值还给A
MLOOP0: MOV   P0,A          ;  根据A的值开灯
        MOV       31H,#2    ;  确定闪烁时间 延时函数最外层循环两次 即0.4s
        LCALL     DELAY     ;  调用延时函数
        RR   A              ;  右移A 接着亮右边的灯
        MOV   32H,A         ;  存储A值
        MOV   33H,A         ;  存储A值
```

```
        LJMP START1       ;  回到主函数
PRG1:   MOV     A,32H     ;  将暂存的A值还给A
MLOOP1: MOV  P0,A         ;  根据A的值开灯
        MOV     31H,#4    ;  确定闪烁时间  延时函数最外层循环四次  即0.4s
        LCALL   DELAY     ;  调用延时函数
        RL   A            ;  左移A  接着亮左边的灯
        MOV  32H,A        ;  存储A值
        MOV  33H,A        ;  存储A值
        LJMP START1       ;  回到主函数
DELAY:  MOV    R1, 31H    ;  延时函数的第三重循环  循环次数由分支函数决定
DEL1:   MOV    R2, #200   ;  延时函数的第二重循环
DEL2:   MOV    R3, #229   ;  延时函数的第一重循环
DEL3:   DJNZ   R3, DEL3   ;  自己循环229次
        DJNZ   R2, DEL2   ;  循环DEL2  229*200次
        DJNZ   R1, DEL1   ;  循环DEL3  229*200*(31H)次
        RET               ;  回到调用处
        END               ;  程序结束
```