Question 2 (15 points)

Write the program to construct the **"Bit logical operation"** which includes Complement, And, Or, and Exclusive-Or operation.

- The program will receive **2 Bit inputs** from **switch 0 and 1** and show their status in **LEDs 0 and 1** (Turn On = 1, Turn Off = 0), respectively.
- **Switch 2 and 3** are used to define the logic operation as described in **Table 1** and will be displayed in the **7-segment** as shown in **Figure 1**.
- Finally, the result of the operation is displayed in **LED 3**.

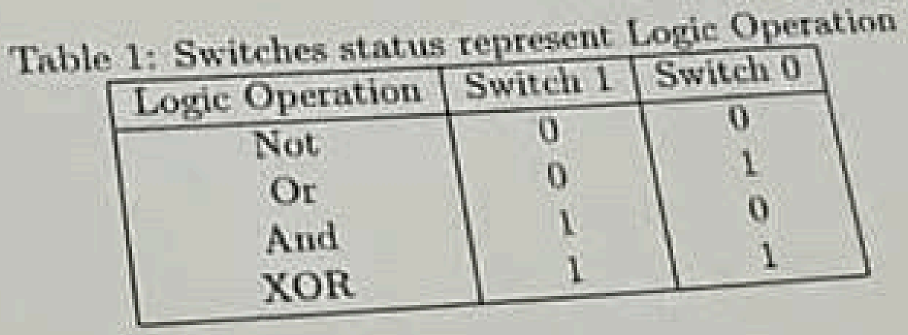

Table 1: Switches status represent Logic Operation

Logic Operation	Switch 1	Switch 0
Not	0	0
Or	0	1
And	1	0
XOR	1	1

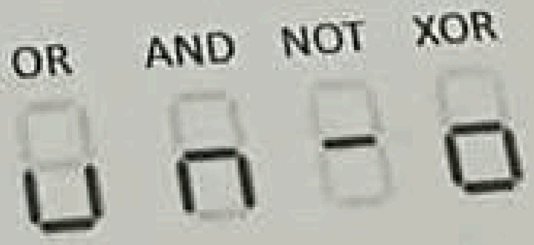

Figure 1: 7-Segment Logic Operation Display

Hints

- Use **PICSimLab** to check your code.
- Recommend to use **"time delay code"** between two outputs to make sure you get correct displays.
-

```
; ==========================================================

; Program: Bit Logical Operation with 7-Seg Display

; Device: ATmega328P

; Inputs:  SW0 (PC0), SW1 (PC1) -> Operands

;          SW2 (PC2), SW3 (PC3) -> Operation Selector

; Outputs: LED0 (PB0), LED1 (PB1) -> Operand Status

;          LED3 (PB3) -> Logic Result

;          7-Seg (PORTD) -> Operation Symbol

; ==========================================================


.include "m328pdef.inc"


; --- 7-Segment Patterns (gfedcba mapping) ---

; Based on Figure 1 shapes:

; NOT: Segment g only (-)       -> 0x40

; OR : Segments c,d,e (u/L)     -> 0x1C

; AND: Segments c,e,g (n)       -> 0x54

; XOR: Segments c,d,e,g (o)     -> 0x5C


.equ SEG_NOT = 0x40

.equ SEG_OR  = 0x1C

.equ SEG_AND = 0x54

.equ SEG_XOR = 0x5C


.cseg

.org 0x0000
```

```
    rjmp start


start:

    ; --- 1. Initialize Stack ---

    ldi r16, low(RAMEND)

    out SPL, r16

    ldi r16, high(RAMEND)

    out SPH, r16


    ; --- 2. Port Configuration ---


    ; Configure PORTB as Output (LEDs)

    ldi r16, 0xFF

    out DDRB, r16


    ; Configure PORTD as Output (7-Segment)

    ldi r16, 0xFF

    out DDRD, r16


    ; Configure PORTC as Input (Switches)

    ldi r16, 0x00

    out DDRC, r16


    ; Enable Internal Pull-ups on PORTC (Optional, for stability)

    ldi r16, 0xFF

    out PORTC, r16
```

```
main_loop:

    ; --- 3. Read Inputs ---

    in r16, PINC        ; Read all switches on PORTC


    ; --- 4. Display Switch Status (LED0, LED1) ---

    ; r16 has format: ... S3 S2 S1 S0

    ; We need to preserve S1 and S0 for LEDs

    mov r20, r16        ; Copy state to r20

    andi r20, 0x03      ; Keep only bits 0 and 1 (S0 and S1)

                        ; r20 now holds the data for LED0 and LED1


    ; --- 5. Extract Operands ---

    ; Operand A (Switch 0)

    mov r22, r16

    andi r22, 0x01      ; r22 = 0 or 1


    ; Operand B (Switch 1)

    mov r23, r16

    andi r23, 0x02      ; Isolate bit 1

    lsr r23             ; Shift right to make it 0 or 1 (r23 = Switch 1
value)


    ; --- 6. Determine Operation (Switch 3 & 2) ---

    ; Mask bits 3 and 2 to check selection

    mov r19, r16
    andi r19, 0x0C      ; Keep S3, S2. Possible values: 0x00, 0x04, 0x08,
0x0C


    ; --- 7. Logic Branching ---

    cpi r19, 0x00       ; S3=0, S2=0 -> NOT

    breq op_not


    cpi r19, 0x04       ; S3=0, S2=1 -> OR

    breq op_or


    cpi r19, 0x08       ; S3=1, S2=0 -> AND

    breq op_and


    cpi r19, 0x0C       ; S3=1, S2=1 -> XOR

    breq op_xor


    rjmp main_loop      ; Safety loop


; --- Operation Subroutines ---


op_not:

    ; Table Row 1: NOT

    ldi r21, SEG_NOT    ; Load 7-Seg Pattern


    ; Perform NOT on Switch 0 (Operand A)

    mov r24, r22        ; Move A to Result
    com r24             ; Complement (Invert bits)

    andi r24, 0x01      ; Mask to keep only the LSB (since com inverts all 8
bits)

    rjmp update_output



op_or:

    ; Table Row 2: OR

    ldi r21, SEG_OR     ; Load 7-Seg Pattern



    mov r24, r22        ; Result = A

    or  r24, r23        ; Result = A OR B

    rjmp update_output



op_and:

    ; Table Row 3: AND

    ldi r21, SEG_AND    ; Load 7-Seg Pattern



    mov r24, r22        ; Result = A

    and r24, r23        ; Result = A AND B

    rjmp update_output



op_xor:

    ; Table Row 4: XOR

    ldi r21, SEG_XOR    ; Load 7-Seg Pattern



    mov r24, r22        ; Result = A
    eor r24, r23        ; Result = A XOR B
    rjmp update_output

update_output:
    ; r21 contains 7-segment pattern
    ; r24 contains Result (0 or 1)
    ; r20 contains LED0/LED1 status (bits 0,1)

    ; 1. Output 7-Segment Pattern
    out PORTD, r21

    ; 2. Prepare LED 3 (Result)
    ; We need result in Bit 3 position (Binary 1000)
    lsl r24
    lsl r24
    lsl r24             ; Shift result left by 3

    ; 3. Combine with LED0/LED1
    or r20, r24         ; Combine LEDs
    out PORTB, r20      ; Update all LEDs

    ; 4. Call Delay
    rcall delay
    rjmp main_loop

; --- Time Delay Code ---
```

```
delay:

    ldi  r25, 20        ; Outer loop counter (Adjust for speed)

d1: ldi  r26, 255       ; Inner loop counter

d2: ldi  r27, 255       ; Innermost loop

d3: dec  r27

    brne d3

    dec  r26

    brne d2

    dec  r25

    brne d1

    ret
```