LAB 4 - A :

ACTIVITY 1

Write a program that copies bits 1, 2, and 3 of PIND to bits 0, 1, and 2 of port B.

```
LDI R16, 0x00        ; Input Value
OUT DDRD, R16        ; PORTD = INPUT
LDI R18, 0xFF        ; Output Value
OUT DDRB, R18        ; PORTB = OUTPUT

MAIN :
        IN R17, PIND         ; Store all values of PORTD
        ANDI R17, 0x0E       ; Bitwise Digit 0, 4, 5, 6, and 7 → 0000 xxx0

        ; Shift Bit to Right & Send R17 to PORTB
        LSR R17;             ; 0000 0111
        OUT PORTB, R17

        RJMP MAIN
```

ACTIVITY 2

Write a program that copies bits 1 and 6 of PINC to bits 0, and 2 of port B, respectively.

```
   ; 1. Set Up Ports

LDI R16, 0x00

LDI R17, 0x05   ; 0000 0101 (Only need bits 0 and 2 as output)

OUT DDRC, R16   ; PORTC = INPUT

OUT DDRB, R17   ; PORTB = OUTPUT



MAIN:

    IN R18, PINC      ; Read the whole Port C



    ; --- Handle Bit 1 -> Bit 0 ---

    MOV R19, R18    ; Copy Port C value to R19
```

```
ANDI R19, 0x02  ; Mask bit 1 (0000 0010)
LSR R19         ; Shift Right once: bit 1 moves to bit 0


; --- Handle Bit 6 -> Bit 2 ---
MOV R20, R18    ; Copy Port C value to R20
ANDI R20, 0x40  ; Mask bit 6 (0100 0000)
; We need to move bit 6 down to bit 2 (shift right 4 times)
LSR R20
LSR R20
LSR R20
LSR R20         ; Bit 6 is now at bit 2


; --- Combine and Output ---
OR R19, R20     ; Combine both bits into R19
OUT PORTB, R19  ; Send result to Port B


RJMP MAIN
```

ACTIVITY 3

1) Write a program that inverts bit 3 of port C and sends it to bit 5 of port B.

```
; 1. Setup
LDI R16, 0x00
OUT DDRC, R16    ; Port C = Input
LDI R17, 0x20    ; 0010 0000 (Bit 5 as output)
OUT DDRB, R17    ; Port B = Output


MAIN:
    IN  R18, PINC    ; Read Port C
    ANDI R18, 0x08   ; Isolate Bit 3 (0000 1000)

    ; 2. Invert Bit 3
    ; Using XOR with 0x08 flips only bit 3
    XORI R18, 0x08

    ; 3. Move Bit 3 to Bit 5
    LSL R18          ; Shift left (Bit 3 -> Bit 4)
    LSL R18          ; Shift left (Bit 4 -> Bit 5)

    OUT PORTB, R18   ; Send to Port B
    RJMP MAIN
```

2) Find the value in R16 after the following code.

```
LDI   R16, $45     ; 64+5 = 69 = 0100 0101 | C = 0

ROR   R16          ; 0010 0010 | C = 1

ROR   R16          ; 1001 0001 | C = 0

ROR   R16          ; 0100 1000 = $48
```

R16 = **$48** in hex

3) Find the value in R16 after the following code.

```
LDI   R16, $45     ; 0100 0101 | C = 0

ROL   R16          ; 1000 1010 | C = 0

ROL   R16          ; 0001 0100 | C = 1

ROL   R16          ; 0010 1001
```

R16 = **$29** in hex

4) In the absence of the "SWAP Rn" instruction, how does the operation perform?

How does "SWAP" work?
The SWAP instruction takes an 8-bit register (Rn) and exchanges its low nibble (bits 0-3) with its high nibble (bits 4-7).

Before: [High Nibble] [Low Nibble]
After: [Low Nibble] [High Nibble]

Without "SWAP":
We would perform the operation using **four shifts**:

- To swap the high nibble to the low, you `LSR` four times.
- To swap the low nibble to the high, you `LSL` four times.
- You then `OR` the results together.

5) Can the SWAP instruction work on any register?

Yes. The SWAP Rd instruction works on any of the 32 general-purpose registers (R0 through R31).

6) Find the value in R2 after the following code.

```
CLR   R2          ; 0000 0000

LDI   R21, $FF    ; 1111 1111

EOR   R2, R21     ; $00 XOR $FF = $FF
```

R2 = **$FF** in hex

7) Find the value in A after the following code.

```
CLR   R10         ; 0000 0000

COM   R10         ; COMPLEMENT R10 = 1111 1111

LDI   R16, $AA    ; 1010 1010

EOR   R10, R16    ; 0101 0101 = $55 in HEX
```

R10 = **$55** in hex

LAB 4 - B :

ACTIVITY 1

Write a program to transfer a string of data from **program memory** starting at address $200 to RAM locations starting at $140. Using the simulator, single-step through the program and examine the data transfer and registers.

```
.ORG $0000
   RJMP MAIN

; 1. Define the Data in Program Memory
.ORG $0200
MY_STRING: .DB "Ming", 0  ; A string with a null terminator (0) at the end

MAIN:
   ; 2. Initialize Z-pointer to point to the source (Flash $200)
   ; We shift left by 1 because Flash is word-addressed but LPM is byte-addressed.
   LDI ZL, LOW(MY_STRING << 1)
   LDI ZH, HIGH(MY_STRING << 1)

   ; 3. Initialize X-pointer to point to the destination (RAM $140)
   LDI R26, $40    ; Low byte of $0140
   LDI R27, $01    ; High byte of $0140

TRANSFER_LOOP:
   ; 4. Read byte from Program Memory using Z-pointer
   LPM R16, Z+     ; Load byte into R16 and increment Z-pointer

   ; 5. Check for Null Terminator (0) to know when to stop
   TST R16         ; Check if the byte in R16 is zero
   BREQ DONE       ; If it's zero, we finished copying the string

   ; 6. Store byte into RAM using X-pointer
   ST X+, R16      ; Store into SRAM and increment X-pointer

   RJMP TRANSFER_LOOP ; Repeat for the next character

DONE:
   HERE: RJMP HERE ; End of program
```

Note :

- **Source:** Flash memory accessed via LPM and the Z-pointer.
- **Destination:** SRAM memory accessed via ST and the X-pointer.
- **The Shift:** Always remember that MY_STRING << 1 trick; without it, we'll be reading the wrong bytes from memory.

ACTIVITY 2

Add the subroutine to the program in Activity 1. After data has been transferred from program memory into RAM, the subroutine function should copy the data from **RAM** locations starting at $140 to **RAM** locations starting at $160. Use single-step through the subroutine and examine the operations.

```
.ORG $0000
   RJMP MAIN

; 1. Define the Data in Program Memory
.ORG $0200
MY_STRING: .DB "Ming", 0  ; Null-terminated string

MAIN:
   ; --- PART 1: FLASH TO RAM ($140) ---
   LDI ZL, LOW(MY_STRING << 1)
   LDI ZH, HIGH(MY_STRING << 1)
   LDI R26, $40    ; X-pointer Low (RAM $140)
   LDI R27, $01    ; X-pointer High

FLASH_TO_RAM:
   LPM R16, Z+     ; Read from Flash
   ST X+, R16      ; Store to RAM $140
   TST R16         ; Check for end of string
   BRNE FLASH_TO_RAM

   ; --- PART 2: CALL SUBROUTINE ---
   CALL RAM_COPY_SUB ; Copy from $140 to $160

HERE: RJMP HERE     ; End of main program

; --- SUBROUTINE: RAM TO RAM COPY ---
RAM_COPY_SUB:
   ; 1. Set X-pointer to Source (RAM $140)
   LDI R26, $40
   LDI R27, $01

   ; 2. Set Y-pointer to Destination (RAM $160)
   LDI R28, $60    ; Y-pointer Low (RAM $160)
   LDI R29, $01    ; Y-pointer High

COPY_LOOP:
   LD R16, X+      ; Load from $140 using X-pointer
   ST Y+, R16      ; Store to $160 using Y-pointer
```

```
TST R16         ; Check for end of string
BRNE COPY_LOOP  ; Repeat if not null

RET             ; Return to MAIN
```

Note:

- **X and Y Pointers:** We use the X-pointer ($R27:R26$) and Y-pointer ($R29:R28$) because they are specifically designed for indirect addressing in the SRAM.
- **Post-Increment (+):** Using X+ and Y+ is the most efficient way to handle strings, as the hardware automatically moves to the next memory address for you.
- **Subroutine Isolation:** By putting the copy logic in a subroutine, you can reuse this code whenever you need to move a block of RAM data without rewriting the entire loop.

ACTIVITY 3

1. Write a program to calculate y where $y = x^2 + 2x + 9$. Where x is the number between 0 and 9 and the look-up table for x^2 is located at the address $200 of **program memory**. Register R20 keeps the value of x, and at the end of the program R21 should contain the value of y. Use the simulator to change the x value and single-step through the program, examining the changes.

```
.ORG $0000
   RJMP MAIN

; 1. Look-up Table for x^2 (0^2, 1^2, ..., 9^2)
.ORG $0200
SQUARE_TABLE: .DB 0, 1, 4, 9, 16, 25, 36, 49, 64, 81

MAIN:
   ; --- Step 1: Initialize x ---
   LDI R20, 5      ; Example: Let x = 5

   ; --- Step 2: Get x^2 from the Table ---
   LDI ZL, LOW(SQUARE_TABLE << 1)
   LDI ZH, HIGH(SQUARE_TABLE << 1)

   ADD ZL, R20     ; Add x to the base address to point to x^2
   LDI R16, 0
   ADC ZH, R16     ; Add carry if necessary

   LPM R21, Z      ; R21 now contains x^2 (e.g., 25)

   ; --- Step 3: Add 2x ---
   MOV R17, R20    ; Copy x
   LSL R17         ; Shift Left = Multiply by 2 (2x)
   ADD R21, R17    ; R21 = x^2 + 2x

   ; --- Step 4: Add 9 ---
   SUBI R21, -9    ; R21 = x^2 + 2x + 9 (Using the SUBI trick for addition)

HERE: RJMP HERE
```

2. Explain the difference between the following two instructions:

    a. LPM R16, Z

    Load Program Memory reads a byte from the Flash memory (where the code and .DB constants are stored).

    b. LD R16, Z

    Load reads a byte from the SRAM (Data Memory), looking for variables or temporary data stored in RAM, not the permanent code space

3. Circle the invalid instructions.

    a. LDS R20, 60

    b. LD R30, Z

    c. LD R25, Z+

    d. LPM R25, Z+4

    AVR pointers do not support "Pointer + Constant" offsets in a single instruction like this. We can use Z or Z+, but you cannot add an immediate value like 4 inside the instruction. We would have to use “ADIW ZL, 4” before the LPM if you wanted to skip 4 bytes.

4. Explain the difference between the following two instructions:

    a. LDS R20, $40

    Load Direct from Data Space goes to **SRAM** memory address $40, look at what is stored inside that "box," and copy that value into R20.

    - R20 = The *contents* of the address $40.

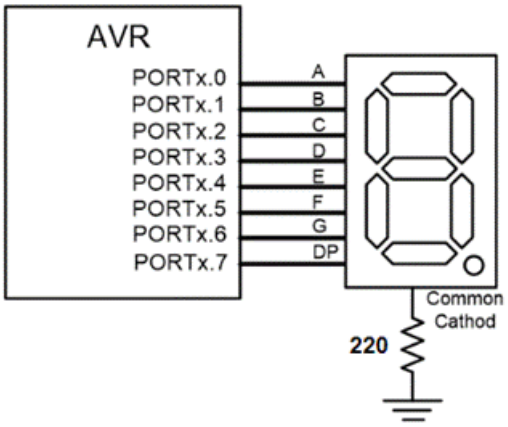
    b. LDI R20, $40

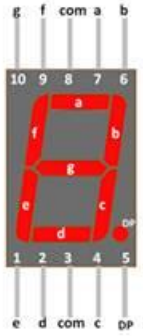
    Load Immediate takes the actual number $40 (64 in decimal) and put it directly into R20.
    - R20 = $40

LAB 4 - B :

ACTIVITY 1

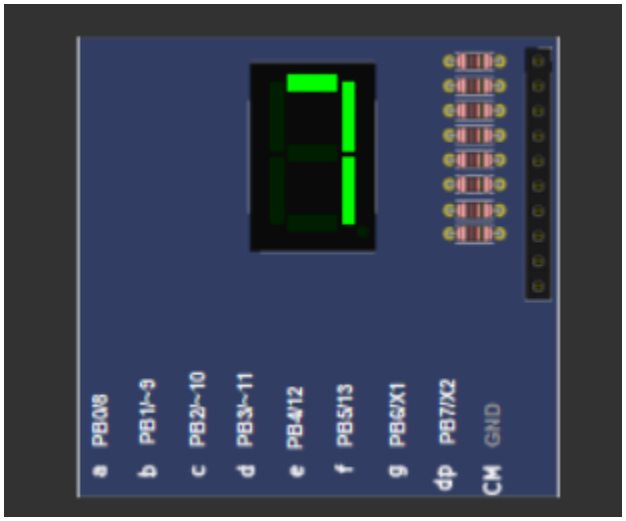
a) Connect a common cathode 7-segment directly to PORTD.





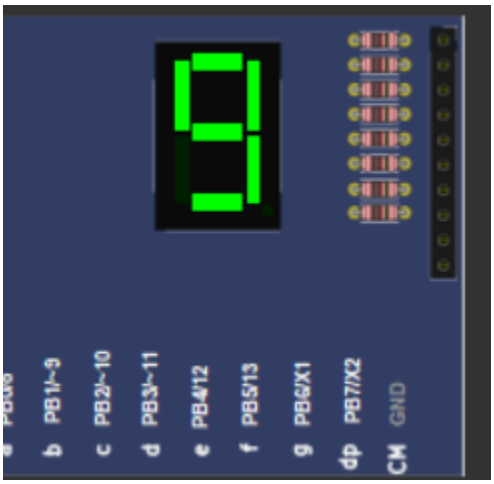
b) Write the following program in the AVR Studio, build and download to the picsimlab.

```
          LDI    R20, 0xFF

          OUT    DDRD, R20

          LDI    R20, 0b00000111

          OUT    PORTD, R20

HERE: RJMP       HERE
```



ACTIVITY 2

Modify the previous program to display 9 on the 7-segment.



by Modifying the following line :

```
LDI     R20, 0b01101111 ; from 0b00000111
```

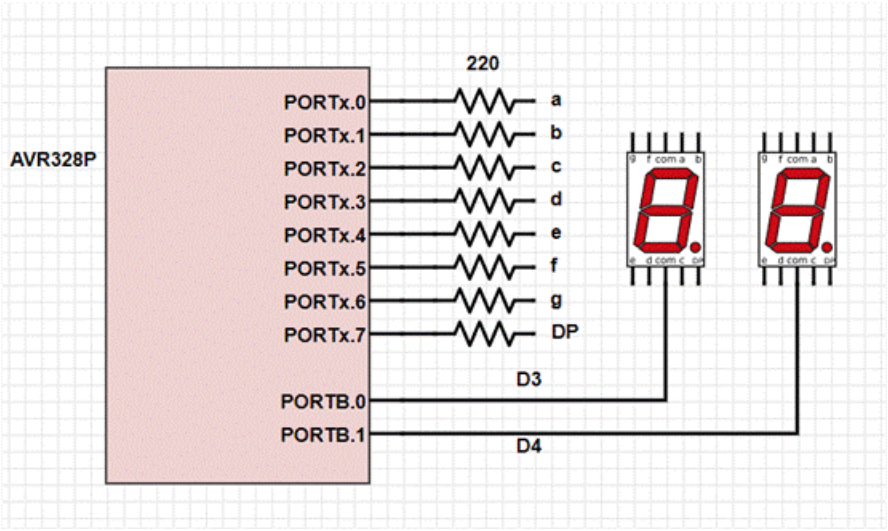
ACTIVITY 3

Use a **look-up table** to write a subroutine that displays the value stored in R21 on the 7-segment.

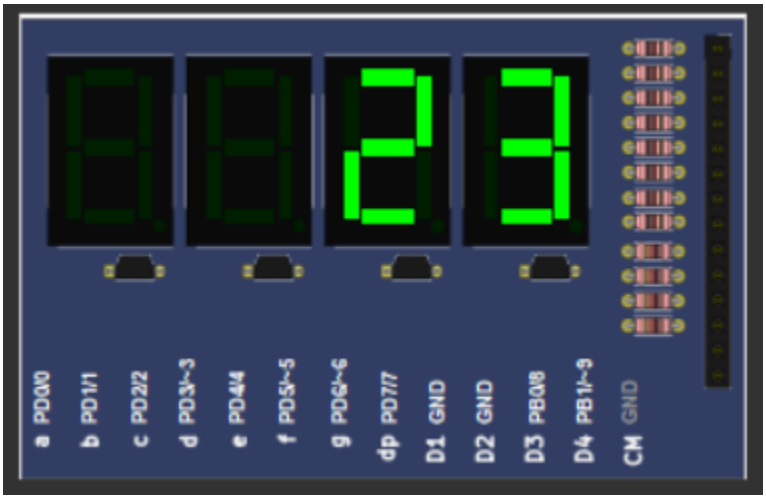
```
.ORG $0000
    RJMP MAIN

; 1. Look-up Table for 7-Segment Patterns (0-9)
.ORG $0200
SEVEN_SEG_TABLE:
    .DB $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F

MAIN:
    ; Setup Port B as output for the 7-segment display
    LDI R16, $FF
    OUT DDRB, R16

    ; Example: Value to display is 5
    LDI R21, 5

    ; Call the subroutine
    CALL DISPLAY_HEX

HERE: RJMP HERE

;================================================
; Subroutine: DISPLAY_HEX
; Input: R21 (The value to display)
; Output: PORTB (The segment pattern)
;================================================
DISPLAY_HEX:
    ; 1. Point Z to the start of the table
    LDI ZL, LOW(SEVEN_SEG_TABLE << 1)
    LDI ZH, HIGH(SEVEN_SEG_TABLE << 1)

    ; 2. Add the offset (the value in R21) to the Z-pointer
    ADD ZL, R21
    CLR R16
    ADC ZH, R16     ; Add carry to high byte if ZL overflowed

    ; 3. Load the pattern from Flash memory
    LPM R16, Z

    ; 4. Send the pattern to the display
    OUT PORTB, R16

    RET
```

ACTIVITY 4

a) Connect two 7-segments to the same port of AVR, as shown below.



b) Display 23 by scanning on those two 7-segments.




```
.ORG $0000
    RJMP MAIN

.ORG $0200
SEVEN_SEG_TABLE:
    .DB $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F

MAIN:
    ; 1. Setup Ports
    LDI R16, $FF
    OUT DDRD, R16     ; Port D = Output (Segments)
    LDI R16, $03
    OUT DDRB, R16     ; Port B.0 and B.1 = Output (Digit Select)

DISPLAY_LOOP:
    ; --- Display "2" (Tens Digit) ---
    LDI R21, 2        ; Value to show
    RCALL GET_PATTERN
    OUT PORTD, R16    ; Send pattern for '2' to Port D
    LDI R17, 0x01     ; 0000 0001
    OUT PORTB, R17    ; Turn ON Digit 3 (B.0), Turn OFF Digit 4
```

```
    RCALL MUX_DELAY  ; Wait ~5-10ms

    ; --- Display "3" (Units Digit) ---
    LDI R21, 3       ; Value to show
    RCALL GET_PATTERN
    OUT PORTD, R16   ; Send pattern for '3' to Port D
    LDI R17, 0x02    ; 0000 0010
    OUT PORTB, R17   ; Turn OFF Digit 3, Turn ON Digit 4 (B.1)
    RCALL MUX_DELAY  ; Wait ~5-10ms

    RJMP DISPLAY_LOOP

; --- Subroutine: Get Segment Pattern from LUT ---
GET_PATTERN:
    LDI ZL, LOW(SEVEN_SEG_TABLE << 1)
    LDI ZH, HIGH(SEVEN_SEG_TABLE << 1)
    ADD ZL, R21
    LDI R16, 0
    ADC ZH, R16
    LPM R16, Z
    RET

; --- Subroutine: Small Delay for Multiplexing ---
MUX_DELAY:
    LDI R18, 100
D1: LDI R19, 100
D2: DEC R19
    BRNE D2
    DEC R18
    BRNE D1
    RET
```