# LAB 3-B

# I/O PORTS

## OBJECTIVES:

- To examine the I/O port operation using a simulator.
- To trace through a CALL subroutine using a simulator.

## MATERIAL:

- Atmel Studio
- https://lcgamboa.github.io/js/picsimlab.html?../picsimlab_examples/ (Simulator)

## WEB SITES:

- www.microchip.com for Atmel Studio Software

## ACTIVITY 1

Write and assemble a program to toggle all the bits of PORTB, PORTC, and PORTD continuously by sending $55 and $AA to these ports. Put a time delay ( between the "on" and "off" states. Then using the simulator, single-step through the program and examine the ports. **Do not single-step through the time delay call.**

```
.equ DELAY_INNER = 100      ; Inner loop count
.equ DELAY_OUTER = 800      ; Outer loop count

delay:
    LDI R18, DELAY_OUTER    ; Load outer loop counter (800)
L1:
    LDI R19, DELAY_INNER    ; Load inner loop counter (100)
L2:
    NOP                     ; Each NOP takes 1 cycle
    DEC R19                 ; Decrement inner loop counter
    BRNE L2                 ; Branch if not zero, takes 2 cycles if branch
                        taken, 1 cycle if not
    DEC R18                     ; Decrement outer loop counter
    BRNE L1                     ; Branch if not zero
    RET                         ; Return from subroutine
```

```
.INCLUDE "m328pdef.inc"

.EQU DELAY_INNER = 100
.EQU DELAY_OUTER = 255   ; Adjusted from 800 to 255 (Max 8-bit value)

.ORG 0x0000
  RJMP MAIN
```

# LAB 3-B

# I/O PORTS

```
MAIN:
   ; 1. Initialize Stack Pointer (Crucial for Subroutines)
   LDI R16, LOW(RAMEND)
   OUT SPL, R16
   LDI R16, HIGH(RAMEND)
   OUT SPH, R16

   ; 2. Configure PORTB, PORTC, PORTD as Outputs
   LDI R16, 0xFF
   OUT DDRB, R16
   OUT DDRC, R16
   OUT DDRD, R16

LOOP:
   ; 3. Output $55 (Binary 01010101)
   LDI R16, 0x55
   OUT PORTB, R16
   OUT PORTC, R16
   OUT PORTD, R16

   CALL DELAY         ; Call delay subroutine

   ; 4. Output $AA (Binary 10101010)
   LDI R16, 0xAA
   OUT PORTB, R16
   OUT PORTC, R16
   OUT PORTD, R16

   CALL DELAY         ; Call delay subroutine

   RJMP LOOP          ; Repeat indefinitely

; --- Delay Subroutine ---
DELAY:
   LDI R18, DELAY_OUTER    ; Load Outer Loop Counter
L1:
   LDI R19, DELAY_INNER    ; Load Inner Loop Counter
L2:
   NOP                ; No Operation (1 cycle delay)
   DEC R19               ; Decrement Inner
   BRNE L2               ; Loop Inner if not zero

   DEC R18               ; Decrement Outer
   BRNE L1               ; Loop Outer if not zero
   RET                ; Return from Subroutine
```

# LAB 3-B

# I/O PORTS

## ACTIVITY 2

Examine the registers of the delay subroutine and make the delay shorter or longer by changing the DELAY_INNER or DELAY_OUTTER value.

- To make the delay shorter, you can decrease the values of DELAY_INNER (e.g., to 50) or DELAY_OUTER.
- To make the delay longer, you can increase these values (up to 255 for 8-bit registers). If you need a delay longer than what 255 allows (like the 800 in the prompt), you would need to implement a third nested loop or use a 16-bit register pair (like X, Y, Z) for the counter.

## ACTIVITY 3

Using a simulator, write a program to get a byte of data from PORTD (Change the value of PORTD during debugging when getting data from it) and send it to PORTB. Also, give a copy of it to registers R20, R21, and R22. Single-step the program and examine the ports and registers.

```
.INCLUDE "m328pdef.inc"

.ORG 0x0000
   ; Configure PORTD as Input ($00) and PORTB as Output ($FF)
   LDI R16, 0x00
   OUT DDRD, R16
   LDI R16, 0xFF
   OUT DDRB, R16

LOOP_COPY:
   IN R16, PIND    ; Read from PORTD (PIND register)
   OUT PORTB, R16  ; Send to PORTB

   ; Copy to registers
   MOV R20, R16
   MOV R21, R16
   MOV R22, R16

   RJMP LOOP_COPY
```

a) Upon reset, all the ports of the AVR are configured as __input____ (input, output).

b) To make all the bits of a port an input port we must write __0x00__ hex to DDRx.

c) Write a program to monitor port B.0 continuously. When it becomes low, it sends $55 to PORTB.

# LAB 3-B

# I/O PORTS

```
.INCLUDE "m328pdef.inc"


.ORG 0x0000
   CBI DDRB, 0     ; Set PORTB.0 as Input
   ; Note: Assuming rest of PORTB is output to display $55
   LDI R16, 0xFE   ; 11111110 (Bit 0 input, others output)
   OUT DDRB, R16

CHECK_PIN:
   SBIC PINB, 0    ; Skip next line if PINB bit 0 is Cleared (Low)
   RJMP CHECK_PIN  ; Jump back if bit 0 is Set (High) - Keep waiting

   ; If we get here, PINB.0 is LOW
   LDI R16, 0x55
   OUT PORTB, R16

HERE: RJMP HERE     ; Stop or loop here
```

## ACTIVITY 4

Test the AVR's ports by using picsimlab for input operation as follows.

a) Connect the pins of PORTx.4-PORTx.7 (PORTD for example) of the AVR to DIP switches. Also connect the pins of PORTy.4-PORTy.7 (e.g. PORTB) to LEDs.

b) Then, write and run a program to get data from PORTx.4-PORTx.7 and send it to PORTy.4-PORTy.7, respectively. Any change of status of the switches connected to PORTx will be instantly reflected on LEDs which are connected to PORTy.

The Code Solution: This program reads the high nibble (bits 4-7) from PORTD (Switches) and sends it to the high nibble of PORTB (LEDs).
**Note:** Verified on PicSimLab: Switches on PD4-PD7 successfully control LEDs on PB4-PB7.

```
.INCLUDE "m328pdef.inc"

.ORG 0x0000
   RJMP MAIN

MAIN:
   ; 1. Configure PORTB (LEDs)
   ; We need PB4-PB7 as Output.
   ; LDI R16, 0xF0  (Binary 11110000) -> 1 is Output
   LDI R16, 0xF0
   OUT DDRB, R16

   ; 2. Configure PORTD (Switches)
```

# LAB 3-B

# I/O PORTS

```
    ; We need PD4-PD7 as Input.
    ; LDI R16, 0x00  (Binary 00000000) -> 0 is Input
    LDI R16, 0x00
    OUT DDRD, R16

    ; Optional: Enable Pull-up resistors on Inputs (PD4-PD7)
    ; This ensures stable HIGH signal when switch is open
    LDI R16, 0xF0
    OUT PORTD, R16

LOOP:
    ; 3. Read from PORTD (Input)
    IN R16, PIND    ; Read status of pins PD0-PD7

    ; 4. Mask Data
    ; We only care about bits 4-7. Force lower bits to 0.
    ANDI R16, 0xF0

    ; 5. Output to PORTB
    ; Send the status directly to LEDs on PB4-PB7
    OUT PORTB, R16

    RJMP LOOP       ; Infinite loop to monitor changes continuously
```

**Note: The main program functions must be in the infinite loop to keep the controller working**