LAB 3-C

COUNTER

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

Test the operation of an "up-counter" from $00 - $FF on picsimlab. Connect the LEDs to each pin of the PORTD and perform the following steps:

a) Assemble the following code:

```
;========Press your up-counter code here==========




;=================================================
DELAY:  LDI     R21, 32
DL1:    LDI     R22, 200
DL2:    LDI     R23, 250
DL3:    NOP
        NOP
        DEC     R23
        BRNE    DL3
        DEC     R22
        BRNE    DL2
        DEC     R21
        BRNE    DL1
        RET
```

b) Upload the hex file to the picsimlab.

c) Observe the LEDs counting up from $00 to $FF.

Lab 3-C

Counter

d) Change the time delay in between the counts by changing the value of R21 register (or increase/decrease the number of NOPs) but make sure the time delay is long enough that you can observe the LEDs counting up. Create a hex file then upload to picsimlab to see what differences.

```
Code:
.INCLUDE "m328pdef.inc"

.ORG 0x0000
    RJMP MAIN

MAIN:
    ; 1. Initialize Stack Pointer (Essential for CALL/RET)
    LDI R16, LOW(RAMEND)
    OUT SPL, R16
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16

    ; 2. Set PORTD as Output (for LEDs)
    LDI R16, 0xFF
    OUT DDRD, R16

    ; 3. Initialize Counter
    LDI R16, 0x00

LOOP:
    OUT PORTD, R16      ; Display counter on LEDs
    RCALL DELAY         ; Call delay
    INC R16             ; Increment counter
    RJMP LOOP           ; Repeat (R16 naturally wraps from $FF to $00)

;=====================================================
; Provided Delay Subroutine
;=====================================================
DELAY:
    LDI R21, 32
DL1:
    LDI R22, 200
DL2:
    LDI R23, 250
DL3:
    NOP
    NOP
    DEC R23
    BRNE DL3
    DEC R22
    BRNE DL2
    DEC R21
```

```
   BRNE DL1
   RET
```

Observation: The LEDs connected to PORTD will display a binary count incrementing from 00000000 (00)to'11111111'(FF) and then roll over to zero continuously. To change the speed, you would modify the value loaded into R21 (currently 32) in the delay subroutine.

ACTIVITY 2

In Activity 1, the maximum count was $FF (or 255). Modify the above program to set maximum count to 10.

e) Upload the hex file into the AVR simulator.

f) Observe the LEDs counting up from $00 to $09 (00001001 binary) continuously.

g) Change the maximum count to the value of your age and observe the LED counting up to that number.

```
.INCLUDE "m328pdef.inc"

.ORG 0x0000
   RJMP MAIN

MAIN:
   ; Initialize Stack Pointer
   LDI R16, LOW(RAMEND)
   OUT SPL, R16
   LDI R16, HIGH(RAMEND)
   OUT SPH, R16

   ; Set PORTD as Output
   LDI R16, 0xFF
   OUT DDRD, R16

   ; Initialize Counter
   LDI R16, 0x00

LOOP:
   OUT PORTD, R16      ; Display current value
   RCALL DELAY

   INC R16           ; Increment value

   ; Check if limit (10) is reached
   CPI R16, 10         ; Compare R16 with 10
   BRNE LOOP           ; If R16 != 10, Branch Not Equal (continue loop)
```

LAB 3-C

COUNTER

```
    ; If R16 == 10, reset to 0
    LDI R16, 0
    RJMP LOOP

; the DELAY subroutine from Activity 1
DELAY:
    LDI R21, 32
DL1:
    LDI R22, 200
DL2:
    LDI R23, 250
DL3:
    NOP
    NOP
    DEC R23
    BRNE DL3
    DEC R22
    BRNE DL2
    DEC R21
    BRNE DL1
    RET
```

ACTIVITY 3

For this activity, connect the DIP switches to pins of port B. Now, use the DIP switches on picsimlab to set the maximum count for an up-counter instead of using constant as Activity 1 and 2.

a) Upload the hex file into the AVR simulator.

b) Observe the LEDs counting up from 00 to the value set by the 8 switches continuously.

```
.INCLUDE "m328pdef.inc"

.ORG 0x0000
    RJMP MAIN

MAIN:
    ; Initialize Stack Pointer
    LDI R16, LOW(RAMEND)
    OUT SPL, R16
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16

    ; Configure PORTD as Output (LEDs)
    LDI R16, 0xFF
```

Lab 3-C

Counter

```
    OUT DDRD, R16

    ; Configure PORTB as Input (Switches)
    LDI R16, 0x00
    OUT DDRB, R16
    ; Enable Internal Pull-ups on PORTB (Optional but recommended for switches)
    LDI R16, 0xFF
    OUT PORTB, R16

    ; Initialize Counter
    LDI R20, 0x00       ; R20 holds the current count

LOOP:
    OUT PORTD, R20      ; Display count on LEDs
    RCALL DELAY         ; Wait

    ; Read Max Count from Switches (PORTB)
    IN R21, PINB        ; R21 holds the Max Count limit

    ; Compare Current Count (R20) with Max Count (R21)
    CP R20, R21
    BREQ RESET_COUNT    ; If R20 == R21 (Equal), reset to 0

    INC R20             ; Else, increment
    RJMP LOOP

RESET_COUNT:
    LDI R20, 0          ; Reset counter
    RJMP LOOP

; the DELAY subroutine from Activity 1
DELAY:
    LDI R21, 32
DL1:
    LDI R22, 200
DL2:
    LDI R23, 250
DL3:
    NOP
    NOP
    DEC R23
    BRNE DL3
    DEC R22
    BRNE DL2
    DEC R21
    BRNE DL1
    RET
```

LAB 3-C

COUNTER

Answer question

1) What is the maximum count for register R20? 255 (or $FF)

2) In this Lab, which port was used to display the count? Which one was used to set the maximum count? Can we use one port for both (inputting the maximum count and displaying the count)? In this lab, PORTD was used to display the count on the connected LEDs, while PORTB was utilized to set the maximum count via the DIP switches,. It is not possible to use a single port for both tasks simultaneously in this configuration because the pins of a port must be specifically configured via the Data Direction Register (DDRx) to act as either an input (logic 0) or an output (logic 1) at any given time.

3) In this Lab, we used BRNE (Branch Not Equal) instruction. Explain how it works. The BRNE (Branch if Not Equal) instruction works by checking the state of the Zero Flag (Z) in the Status Register (SREG),. If the result of the preceding operation was not zero (causing the Z flag to be cleared to 0), the program branches to the specified label, but if the result was zero (setting the Z flag to 1), the program ignores the branch and proceeds to the next instruction.