EE2016 : Lab Report
Experiment 9

Putta Shravya EE22B032
Maadhav Patel EE22b033

October 2023

1 Aim

Using Atmel AVR assembly language programming, implement interrupts and DIP switches control in Atmel Atmega microprocessor. Aims of this experiment are:
(i) Generate an external (logical) hardware interrupt using an emulation of a push button switch.
(ii) Write an ISR (Interrupt Service Routine) to switch ON an LED for a few seconds (10 secs) and then switch OFF. (The lighting of the LED could be verifed by monitoring the signal to switch it ON).
(iii) If there is time, you could try this also: Use the 16 bit timer (via interrupt) to make an LED blink with a duration of 1 second.
Also, one needs to implement all of the above, in AVR assembly.

Google Drive link

Click here to view all codes and videos of problems.

2 Interrupt using INT1

2.1 Code

```
.nolist; turn list file generation OFF
.include "m8def.inc"
.list /* turn it ON again */
.org 0 ; set program origin
rjmp reset ; on reset, program starts here
.org 0x0004 ;
rjmp int1_ISR;
reset:
ldi R16, 0x70 ;
out spl, R16
ldi R16, 0x00 /* Guess, why it is done ??? */
out sph, R16
ldi R16, 0x02    ; make PB1 output
out DDRB, R16; /* fill in here  */
ldi R16, 0x00 ; fill in here
out DDRD, R16 ; make PORTD input
ldi R16,0x08 ;  /* enable internal pull-up resistor. This avoids the high impe */
out PORTD, R16 ; /* -dance state while reading data from external world */
in R16, MCUCR ;
ori R16, 0x08 ; why it is Ored?
out MCUCR, R16  ;
in R16, GICR ; enable INT1 interrupt
ori R16, 0x80 ;
out GICR, R16 ;  /* fill in here  */
ldi R16, 0x00 ;
out PORTB, R16  ;
```

```
sei ; /* what does it do? */
indefiniteloop: rjmp indefiniteloop /* Stay here. Expecting interrupt? */
            /* reset { the main - loop ends here */
int1_ISR: ; INT1 interrupt handler or ISR
in R16, SREG ; save status register SREG
PUSH R16 ; /* Fill in here. save the status register contents into the stack memory. */
                  /* StckPntr decremented. StckPntr tracks the lower end of ACTIVE stack. PC is */
                 /* PUSHed automatically, by default. No need for explicit instruction */
ldi R16,0x0a ; blink led 10 times  by storing R0 a value of 10 & decrementing
mov R0, R16 /* to zero realises the LED blinking 10 times */
back5: ldi R16,0x02 ; Turn on LED
out PORTB, R16 /* LED connected to penultimate bit (B1) of PORTB */
delay1:LDI R16,0xFF ; delay
back2: LDI R17,0xFF ; /* back2 loop starts.. adds delay */

back1:
DEC R17 /* fill in. Innermost delay loop. Each execution cycle is of T_clk */
BRNE back1 /* branch if not equal { means go on in loop till R* goes 0 */
DEC R16 /* for each inner loop run, an equal amount of delay in outer loop */
BRNE back2   /* how many clock cycles for ON period? */

            /* Till this time LED is ON. First part of duty cycle ends */
ldi R16,0x00 ; Turn off LED
out PORTB, R16 ;      /* fill-in here */

delay2:LDI R16,0xFF ; delay { OFF period. Second part of duty cycle starts
back3: LDI R17,0xFF
back4:
DEC R7
BRNE back4
DEC R16
BRNE back3
DEC R16 /* Fill in here. Initially R0 = 10. Completes ONE duty cycle */
BRNE back5 ; /* R0-- till 0. 10 times blinking */

POP R16 ; retrieve status register. The stack's lower end is incremented
out SREG, R16 /* meaning stckPntr++; In \pop R16" instruction, the topend
 /* stack location's value is espewed out and is stored in R16 */

RETI ; go back to main program and set I = 1 (enabling interrupts as the current ISR is executed)
```

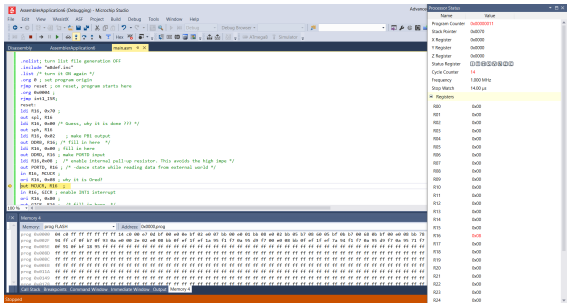
2.2 Inferences

1. ldi R16, 0x70: Load the value 0x70 into register R16.
    - R16 = 0x70
2. out spl, R16: Store the value of R16 into the Stack Pointer Low (SPL) register.
    - SPL = 0x70
3. ldi R16, 0x00: Load the value 0x00 into register R16.
    - R16 = 0x00
4. out sph, R16: Store the value of R16 into the Stack Pointer High (SPH) register.
    - SPH = 0x00
5. ldi R16, 0x02: Load the value 0x02 into register R16.

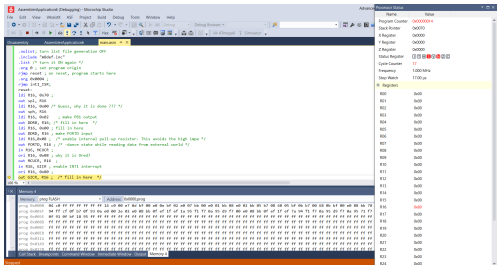
- R16 = 0x02

6. out DDRB, R16: Set the Data Direction Register for Port B (DDRB) to 0x02, making Pin 1 (PB1) an output.
   - DDRB = 0x02
7. ldi R16, 0x00: Load the value 0x00 into register R16.
   - R16 = 0x00
8. out DDRD, R16: Set the Data Direction Register for Port D (DDRD) to 0x00, making all pins on Port D inputs.
   - DDRD = 0x00
9. ldi R16, 0x08: Load the value 0x08 into register R16.
   - R16 = 0x08
10. out PORTD, R16: Enable the internal pull-up resistor for PD3 (INT1) by setting bit 3 in the Port D register (PORTD).
    - PORTD = 0x08
11. in R16, MCUCR: Read the MCU Control Register (MCUCR) into register R16.
    - R16 contains the value of MCUCR
12. ori R16, 0x08: Logically OR R16 with 0x08 (bitwise OR).
    - This sets the third bit of R16 to 1 while leaving other bits unchanged.
13. out MCUCR, R16: Store the modified R16 value back into the MCU Control Register (MCUCR).
14. in R16, GICR: Read the General Interrupt Control Register (GICR) into register R16.
    - R16 contains the value of GICR.
15. ori R16, 0x80: Logically OR R16 with 0x80 (bitwise OR).
    - This sets the seventh bit of R16 to 1 while leaving other bits unchanged.
16. out GICR, R16: Store the modified R16 value back into the General Interrupt Control Register (GICR).
17. ldi R16, 0x00: Load the value 0x00 into register R16.
    - R16 = 0x00
18. out PORTB, R16: Clear all bits of Port B (PORTB), turning off any connected LEDs or outputs.
    - PORTB = 0x00
19. sei: Enable global interrupts by setting the I flag in the Status Register (SREG).
    - The I flag is now set to 1, enabling interrupts globally.

This covers the register transfer values and operations up to the point where the program enters an infinite loop waiting for INT1 interrupts.



(a) INT1



(b) INT1

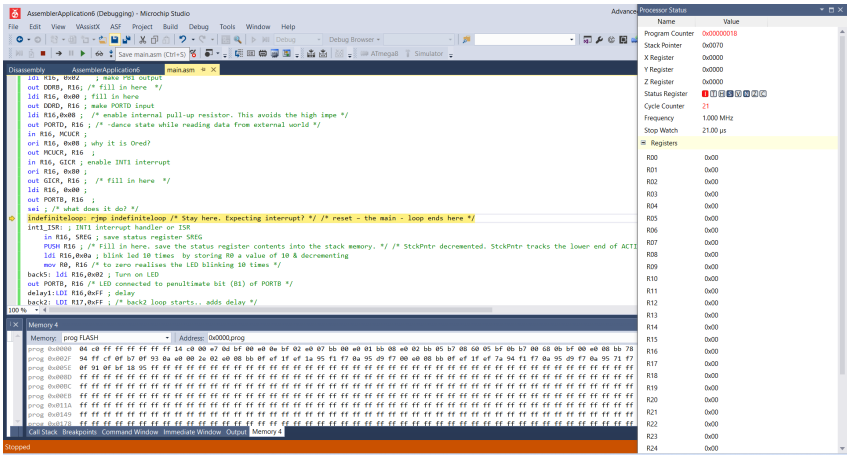


Figure 2: INT1

2.3 Flow Chart

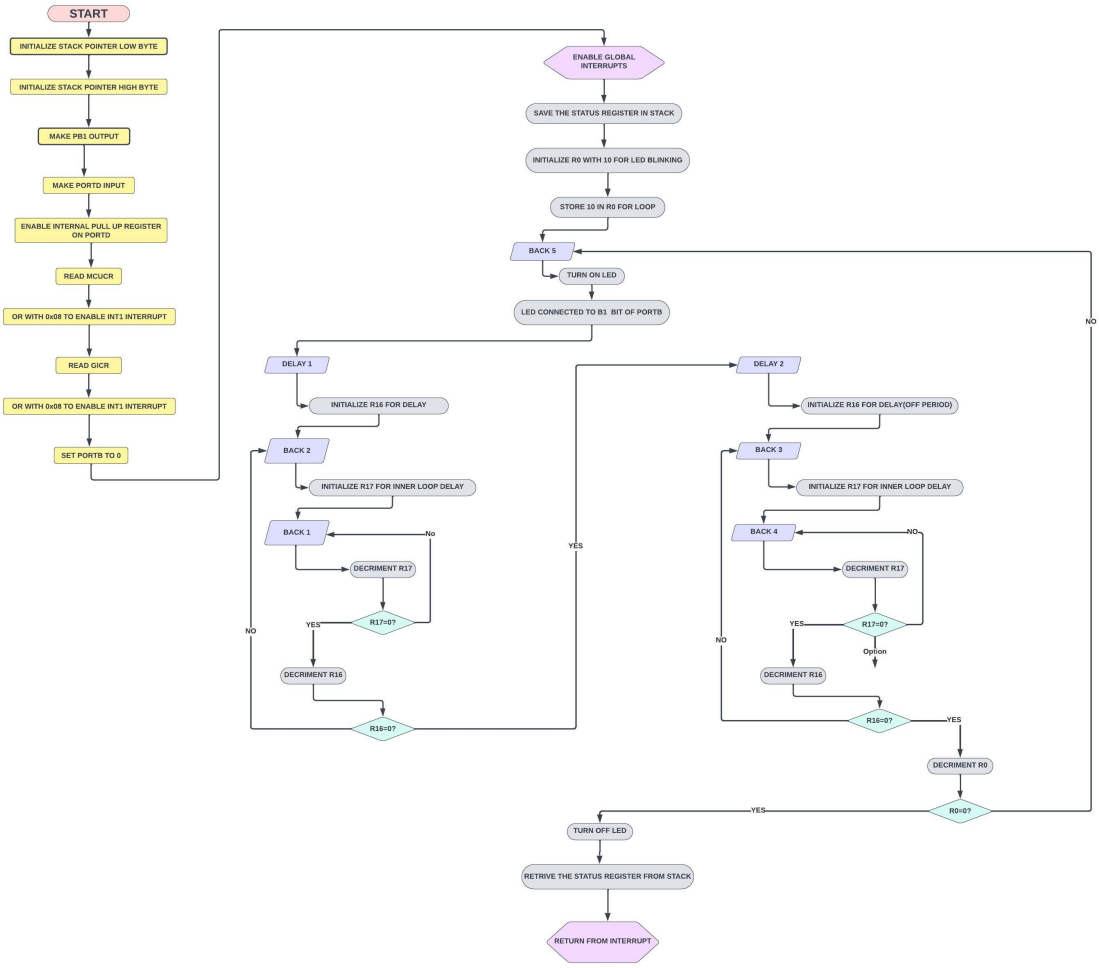


Figure 3: Interrupt Using INT1

3 Interrupt using INT0

3.1 Code

```
.nolist; turn list file generation OFF
.include "m8def.inc"
.list ; turn it ON again

.org 0 ; set program origin
rjmp reset ; on reset, program starts here

.org 0x0002 ; INT0 vector
rjmp int0_ISR;

reset:
ldi R16, 0x70 ; Initialize stack pointer low byte
out spl, R16
ldi R16, 0x00 ; Initialize stack pointer high byte
out sph, R16
ldi R16, 0x02 ; make PB1 output
out DDRB, R16
ldi R16, 0x00 ; make PORTD input
out DDRD, R16
ldi R16, 0x08 ; enable internal pull-up resistor on PORTD
out PORTD, R16
```

```
in R16, MCUCR ; read MCUCR
ori R16, 0x03 ; OR with 0x03 to enable the INT0 interrupt on rising edge
out MCUCR, R16
in R16, GICR ; read GICR
sbr R16, 0x40 ; Set bit 6 to enable INT0 interrupt
out GICR, R16
ldi R16, 0x00 ; Set PORTB to 0
out PORTB, R16
sei ; Enable global interrupts

indefiniteloop:
rjmp indefiniteloop ; Stay in an infinite loop

int0_ISR: ; INT0 interrupt handler or ISR
in R16, SREG ; save status register SREG
PUSH R16 ; Save the status register into the stack

ldi R16, 0x0A ; Initialize R0 with 10 for LED blinking
mov R0, R16 ; Store 10 in R0 for the loop

back5:
ldi R16, 0x02 ; Turn on LED
out PORTB, R16 ; LED connected to penultimate bit (B1) of PORTB

delay1:
ldi R16, 0xFF ; Initialize R16 for delay
back2:
ldi R17, 0xFF ; Initialize R17 for an inner loop delay

back1:
DEC R17 ; Decrement R17
BRNE back1 ; Repeat the inner loop until R17 becomes 0
DEC R16 ; Decrement R16 for the outer loop
BRNE back2 ; Repeat the outer loop until R16 becomes 0

ldi R16, 0x00 ; Turn off LED
out PORTB, R16 ; Turn off the LED

delay2:
ldi R16, 0xFF ; Initialize R16 for delay (OFF period)
back3:
ldi R17, 0xFF ; Initialize R17 for an inner loop delay
back4:
DEC R7 ; Decrement R17
BRNE back4 ; Repeat the inner loop until R17 becomes 0
DEC R16 ; Decrement R16
BRNE back3 ; Repeat the outer loop until R16 becomes 0

DEC R0 ; Decrement R0
BRNE back5 ; Repeat the LED blinking 10 times

POP R16 ; Retrieve the status register from the stack
out SREG, R16 ; Restore the status register

RETI ; Return from interrupt and enable global interrupts
```

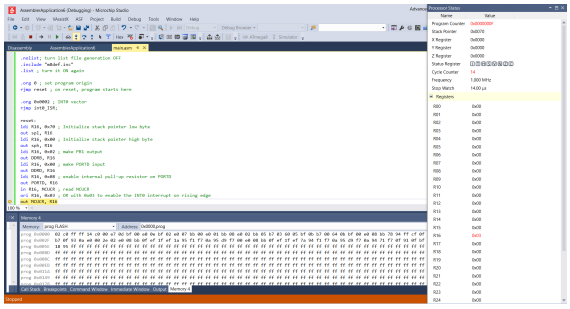
3.2 Inferences

1. ldi R16, 0x70: Load the value 0x70 into register R16.
   - R16 = 0x70
2. out SPL, R16: Store the value of R16 into the Stack Pointer Low (SPL) register.
   - SPL = 0x70
3. ldi R16, 0x00: Load the value 0x00 into register R16.
   - R16 = 0x00
4. out SPH, R16: Store the value of R16 into the Stack Pointer High (SPH) register.
   - SPH = 0x00
5. ldi R16, 0x02: Load the value 0x02 into register R16.
   - R16 = 0x02
6. out DDRB, R16: Set the Data Direction Register for Port B (DDRB) to 0x02, making Pin 1 (PB1) an output.
   - DDRB = 0x02
7. ldi R16, 0x00: Load the value 0x00 into register R16.
   - R16 = 0x00
8. out DDRD, R16: Set the Data Direction Register for Port D (DDRD) to 0x00, making all pins on Port D inputs.
   - DDRD = 0x00
9. ldi R16, 0x08: Load the value 0x08 into register R16.
   - R16 = 0x08
10. out PORTD, R16: Enable the internal pull-up resistor for PD3 (INT0) by setting bit 3 in the Port D register (PORTD).
    - PORTD = 0x08
11. in R16, MCUCR: Read the MCU Control Register (MCUCR) into register R16.
    - R16 contains the value of MCUCR.
12. ori R16, 0x03: Logically OR R16 with 0x03 (bitwise OR).
    - This sets bits 1 and 0 of R16 to 1 while leaving other bits unchanged.
13. out MCUCR, R16: Store the modified R16 value back into the MCU Control Register (MCUCR).
    - MCUCR is updated based on the OR operation.
14. in R16, GICR: Read the General Interrupt Control Register (GICR) into register R16.
    - R16 contains the value of GICR

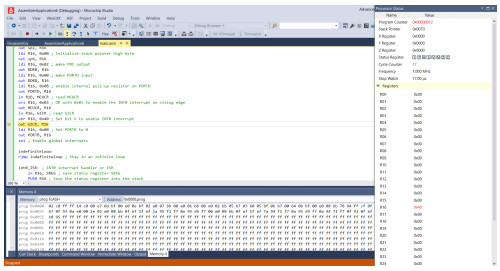
    .
15. sbr R16, 0x40: Set bit 6 of R16 to 1, enabling INT0 interrupt.
    - Bit 6 in R16 is set to 1.
16. out GICR, R16: Store the modified R16 value back into the General Interrupt Control Register (GICR).
    - GICR is updated to enable the INT0 interrupt.
17. ldi R16, 0x00: Load the value 0x00 into register R16.
    - R16 = 0x00

18. out PORTB, R16: Clear all bits of Port B (PORTB), turning off any connected LEDs or outputs.
    - PORTB = 0x00
19. sei: Enable global interrupts by setting the I flag in the Status Register (SREG).
    - The I flag is now set to 1, enabling interrupts globally.

The program then enters an infinite loop waiting for INT0 interrupts, and the subsequent part of the code handles the INT0 interrupt and controls LED blinking.



(a) INT0



(b) INT0

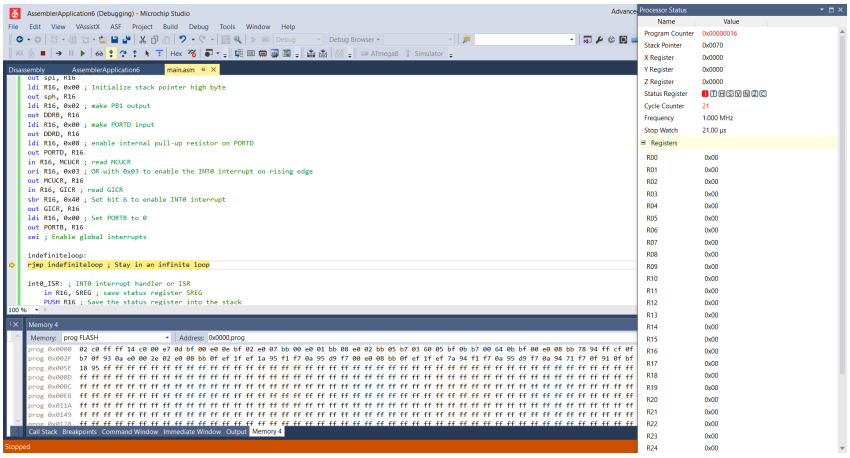


Figure 5: INT0

3.3 Flow Chart

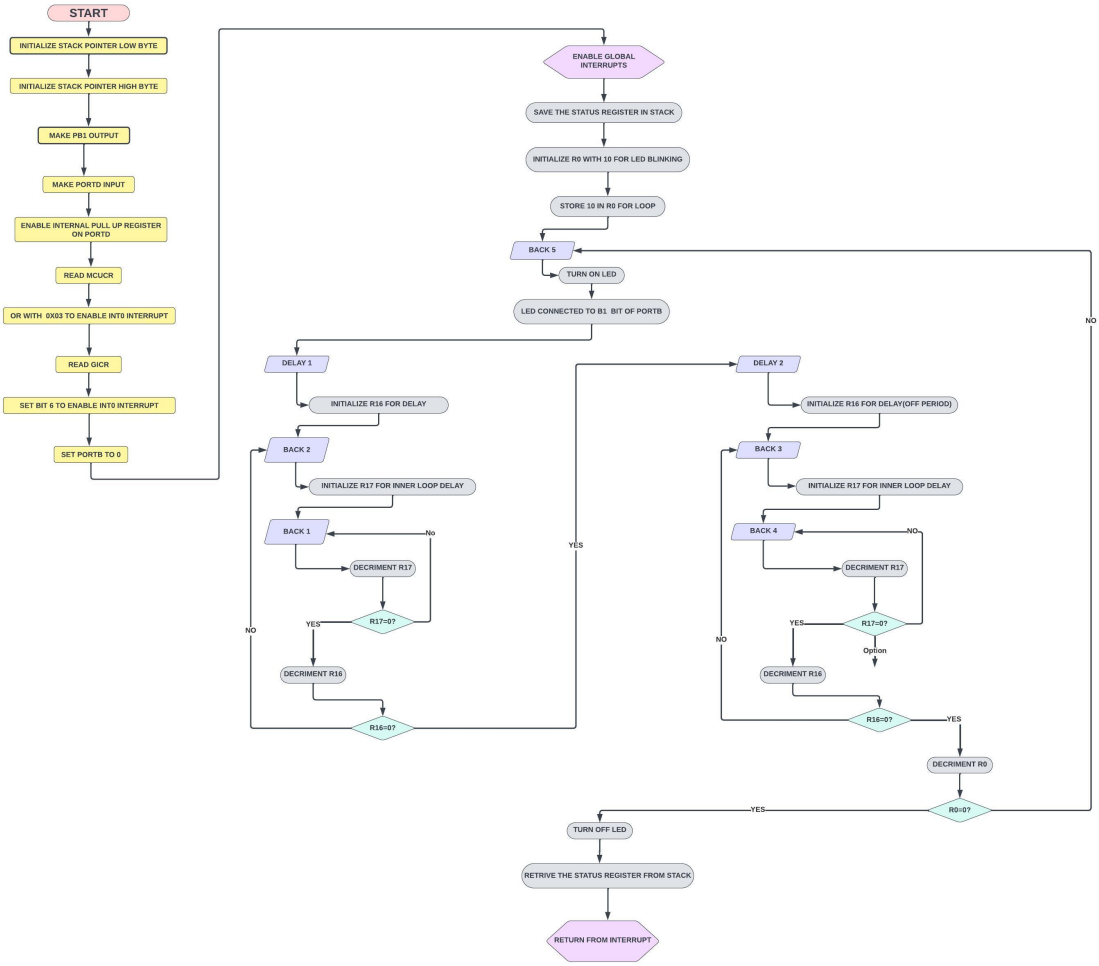


Figure 6: Interrupt Using INT0

4 Questions

1)**ldi R16, 0x00 /* Guess, why it is done ??? */**
A)This instruction loads 0x00 into R16. Its a placeholder for further code.

2)**ori R16, 0x08 ; why it is Ored?**
A)The "ori" instruction performs a bitwise OR operation between R16 and 0x08, setting the ISC11 and ISC10 bits to configure the INT1 interrupt trigger.

3)**sei ; /* what does it do? */**
A)This instruction enables global interrupts by setting the I bit in the Status Register (SREG). This allows the microcontroller to respond to interrupts.

4)**rjmp indefiniteloop /* Stay here. Expecting interrupt? */**
A)It suggests that the code should stay at rjmp indefiniteloop while expecting interrupts, and when an interrupt occurs, it will execute the ISR and return to the loop.

5)**BRNE back2 /* how many clock cycles for ON period? */**
A)The LED remains in the "ON" state for the duration of the combined loops back1 and back2. The total number of clock cycles for the "ON" period is determined by the number of iterations in both the inner and outer loops. Since R16 is initialized to 0xFF (255), and R17 is also initialized to 0xFF (255), the total delay time is determined by the product of these two values, i.e., 255 * 255 = 65,025 clock cycles.

5 Individual Contribution

The Experiment was done by both of the team members.
Implimentation of code of interrupt INT1,flow charts,inferences,learning outcomes were done by Maadhav Patel[EE22B033].
Implimentation of code of interrupt INT0 ,flow charts,inferences,learning outcomes were done by Shravya[EE22B032].
Lab Report was written together by both the team members.