NAME: ROHIT KUMAR

ROLL NO.: EE20B111

EE2016 Microprocessor Lab & Theory Jul - Nov 2021

EE Department, IIT Madras.

(Lab Experiment-2 Report)

Experiment 2: Interrupts and Timers in Atmel AVR Atmega (Title)

1.) Aim: -

Using Atmel AVR assembly language programming, implement interrupts and timers in Atmel Atmega microprocessor. The main constraint is that, it should be emulation only (due to ongoing pandemic). Aims of this experiment are:

(i) Generate an external (logical) hardware interrupt using an emulation of a push button switch.
(ii) Write an ISR to switch ON an LED for a few seconds (10 secs) and then switch OFF. (The lighting of the LED could be verified by monitoring the signal to switch it ON).

Also, one needs to implement all of the above using C-interface.

2.) Equipment Required: -

Since this is an emulation-based experiment, we need only a PC with the following software: Microchip studio simulation software.

3.) Tasks: -

You are given two files (apart from this handout in the directory EE2016F21Exprmnt2AVR.IntrrptCIntrfcng in moodle): EE2016F21Exp2int1.asm and EE2016F21Exp2.int1.c The former one is an assembly program which implements interrupt using int1, while the later one is a 'C' program using int1.c

1. Fill in the blanks in the assembly code.

2. Use int0 to redo the same in the demo program (duly filled in). Once the switch is pressed the LED should blink 10 times (ON (or OFF) - 1 sec, duty cycle could be 50 %). Demonstrate both the cases.

3. Rewrite the program in 'C' (int1). Rewrite the C program for int0.

4. Demonstrate both the cases (of assembly and C).

4.) Demo code (/Questions from handouts): -

# Given demo code for assembly programming: - (INT1 ASM.txt)

```
.org 0x0000

rjmp reset

.org 0x0004

rjmp int1_ISR

.org 0x0100
```

```
reset:
            ; Loading stack pointer address
             LDI R16,0x70
            OUT SPL, R16
            LDI R16,0x00
            OUT SPH, R16

            ; Interface port B pin0 to be output
            ; so, to view LED blinking
             LDI R16,0x00
            OUT DDRD, R16

            ; Set MCUCR register to enable low level interrupt
            OUT MCUCR, R16

            ; Set GICR register to enable interrupt 1
            OUT MCUCR, R16

            LDI R16,0x00
            OUT PORTB, R16

            SEI
ind_loop:
            rjmp ind_loop
int1_ISR:           IN R16, SREG
                    PUSH R16
                    LDI R16,0x0A
                    MOV R0, R16
                    ; Modify below loops to make LED blink for 1 sec
          c1:       LDI R16,0x01
                    OUT PORTB, R16
```

```
        LDI R16,0xFF
a1:     LDI R17,0xFF
a2:     DEC R17
        BRNE a2
        DEC R16
        BRNE a1

        LDI R16,0x00
        OUT PORTB, R16
```

# Given demo code for C-interfacing: - (INT1 C PROG.c)

```c
#define F_CPU 1000000  // clock frequency

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT1_vect)
{
    int i;
    for (i=1;i<=10;i++) // for 10 times LED blink

    {
        PORTB=0x01;
        _delay_ms(1000);   // delay of 1 sec
        PORTB=0x00;
        _delay_ms(1000);

    }


}
int main(void)
{
    //Set the input/output pins appropriately
    //To enable interrupt and port interfacing
    //For LED to blink
    DDRD=0x00;   //Set appropriate data direction for D
    DDRB=0x00;  //Make PB0 as output
    MCUCR=0x00;  //Set MCUCR to level triggered
    GICR=0x00;   //Enable interrupt 1
    PORTB=0x00;
    sei();        // global interrupt flag

    while (1) //wait
    {

    }
}
```

5.) Solutions to the questions: - (Logic/flowchart, code)

(i) Logic/Flowchart: (for assembly program)

We need to use the demo code to blink an LED for 10 times with a delay of 1 second.

We define our program space by using ".org" directives. This directive is usually used to indicate the beginning of the address. In this code, .org directive tells where it has to store the interrupt vector and reset and at what memory locations. The memory locations are clearly defined for our microcontroller Atmega8(in the given handout). So, for interrupt0
(int0 shortly) it should be stored at 0x0001 and for interrupt1(int1 shortly) it should be at 0x0002 for our microcontroller (according to manual).

Then stack pointer is initialised to a memory location 0x0070 by loading 0x70 into SPL and 0x00 into SPH.

Make PORT B PIN0 as an output to view the LED blink by loading 0x01 into DDRB. Next DDRD is loaded with 0x00 which will make PIND bits as input to receive the interrupts at PD2(for INT0) and PD3(for INT1).

We next enable low level interrupts for int0 and int1, for this the ISC01 and ISC00 bits of MCUCR should be loaded with 0x00. Therefore, 0x00 is loaded into the MCUCR to enable low level interrupts. And then setting up the GICR register to enable interrupts. (Loading 0x80 into GICR for int1 and loading 0x40 into GICR for int0).

In order to receive interrupts globally, the D7 bit of the SREG register must be set high. Can be achieved using instruction "SEI".

Next step is to define our ISR for the interrupt (INT1 or INT0). For this, we save the current context (SREG register) into the stack using "PUSH". We then load the counter into R0 with a value of 0x0A (decimal equivalent is 10) to count for the number of blinks.

Next, we turn "ON" the LED by loading PORTB with value 0x01 indicating that PB0 pin is set to HIGH i.e., digital level HIGH. Since, the XTAL frequency is 1MHz (one cycle takes 1µs) therefore we need approximately 10^6 machine cycles to create a time delay of 1s when LED is "ON".

```
    LDI R16, 0x63
a1: LDI R17, 0x63
a2: LDI R18, 0X21
a3: DEC R18
    BRNE a3
    DEC R17
    BRNE a2
    DEC R16
    BRNE a1
```

Total delay in (we use the beside generator code to generate a delay of 1 second) given code =

[(5 + (((3*33-1) +4)99 – 1))*99 -1]x1 µs = 1000097 µs ≈ 1 second (~10^6 µs)

Then next we set the PORTB to 0(low level) to turn "OFF" the LED.

Total delay (between ON and OFF now equals to (1000097+2)x1µs = 1000099µs ≈ 10^6µs = 1 second

Similarly, again 1s delay is done when the LED turns ON again (from the OFF state).

In order for LED to blink 10 times, we decrement the counter by one each time the bulb has blinked once. When the counter becomes 0x00(zero) i.e., it has blinked for 10 times, we stop and pop out the contents from stack and store it back in SREG. Upon executing the RETI instruction, the microcontroller returns to the position where it was interrupted i.e., to the main loop(reset).

(ii) Codes:

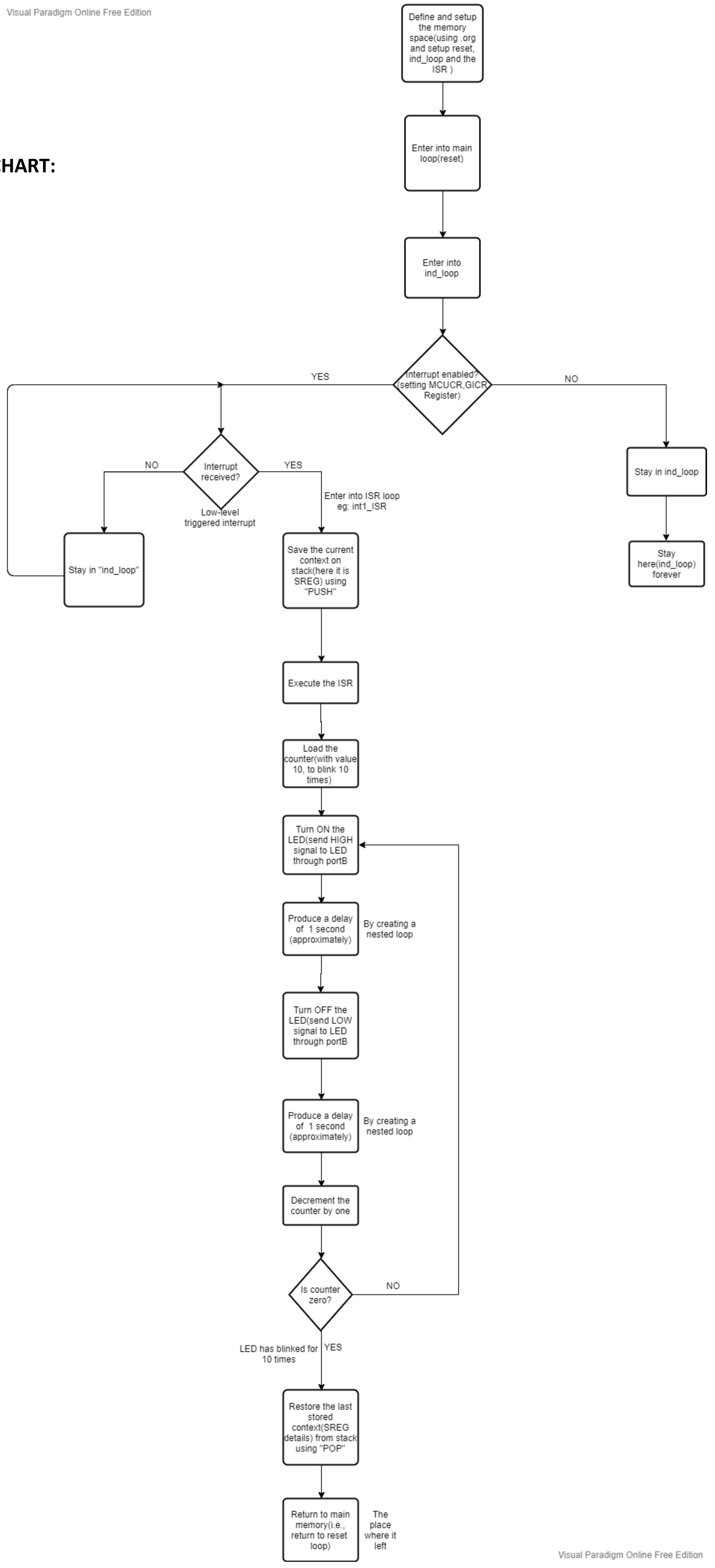
ASSEMBLY INTERRUPT PROGRAM USING INT1

```
.include "m8def.inc"

.org 0x0000;
rjmp reset;

.org 0x0002;
rjmp int1_ISR;

.org 0x0100;

reset:
        ; loading stack pointer address
        LDI R16,0x70;
        OUT SPL, R16;
        LDI R16,0x00;
        OUT SPH, R16;

        LDI R16,0x01;
        OUT DDRB, R16; interface port B pin 0(PB0) to be output

        LDI R16,0x00;
        OUT DDRD, R16;

        ; Set MCUCR register to enable low level interrupt
        LDI R16, 0x00;
        OUT MCUCR, R16;

        ; Set GICR register to enable interrupt 1
        LDI R16, 0x80;
        OUT GICR, R16;

        LDI R16, 0x00;
        OUT PORTB, R16;

        SEI;

ind_loop:
    RJMP ind_loop;

int1_ISR:
    IN R16, SREG;
    PUSH R16; push R16 on stack

    LDI R16, 0x0A; to blink led 10 times
    MOV R0, R16; i.e., R0 is the counter
```

```
; loops to make LED blink for 1 second
c1: LDI R16, 0x01
    OUT PORTB, R16;

    LDI R16, 0x63
a1: LDI R17, 0x63
a2: LDI R18, 0X21
a3: DEC R18
    BRNE a3
    DEC R17
    BRNE a2
    DEC R16
    BRNE a1

c2: LDI R16, 0x00
    OUT PORTB, R16

    LDI R16, 0x63
b1: LDI R17, 0x63
b2: LDI R18, 0X21
b3: DEC R18
    BRNE b3
    DEC R17
    BRNE b2
    DEC R16
    BRNE b1


   DEC R0; Each decrement corresponds to one blink in led
   BRNE c1; blink until R0 != 0, i.e., blink 10 times
   POP R16; pop R16 from stack (when led has blinked 10 times)
   OUT SREG, R16;

   RETI; after POPing I = 0
```

- ➔ Below flowchart (for an idea) is attached which is applicable for both int1 and int0(with just slight modifications).
- ➔ Same flowcharts can also be used to make C-interface programs.

FLOWCHART:


Define and setup the memory space(using .org and setup reset, ind_loop and the ISR)
Enter into main loop(reset)
Enter into ind_loop
Interrupt enabled? (setting MCUCR,GICR Register)
YES
NO
Stay in ind_loop
Stay here(ind_loop) forever
Interrupt received?
NO
YES
Low-level triggered interrupt
Stay in "ind_loop"
Enter into ISR loop eg: int1_ISR
Save the current context on stack(here it is SREG) using "PUSH"
Execute the ISR
Load the counter(with value 10, to blink 10 times)
Turn ON the LED(send HIGH signal to LED through portB
Produce a delay of 1 second (approximately)
By creating a nested loop
Turn OFF the LED(send LOW signal to LED through portB
Produce a delay of 1 second (approximately)
By creating a nested loop
Decrement the counter by one
Is counter zero?
NO
YES
LED has blinked for 10 times
Restore the last stored context(SREG details) from stack using "POP"
Return to main memory(i.e., return to reset loop)
The place where it left

ASSEMBLY INTERRUPT PROGRAM USING INT0

```
.include "m8def.inc"

.org 0x0000;
rjmp reset;

.org 0x0001;
rjmp int0_ISR;

.org 0x0100;

reset:
        ; loading stack pointer address
        LDI R16,0x70;
        OUT SPL, R16;
        LDI R16,0x00;
        OUT SPH, R16;

        LDI R16,0x01;
        OUT DDRB, R16; interface port B pin 0(PB0) to be output

        LDI R16,0x00;
        OUT DDRD, R16;

        ; Set MCUCR register to enable low level interrupt
        LDI R16, 0x00;
        OUT MCUCR, R16;

        ; Set GICR register to enable interrupt 0
        LDI R16, 0x40;
        OUT GICR, R16;

        LDI R16, 0x00;
        OUT PORTB, R16;

        SEI;

ind_loop:
    RJMP ind_loop;

int0_ISR:
    IN R16, SREG;
    PUSH R16; push R16 on stack

    LDI R16, 0x0A; to blink led 10 times
    MOV R0, R16; i.e., R0 is the counter

 ; loops to make LED blink for 1 second
 c1: LDI R16, 0x01
     OUT PORTB, R16;

     LDI R16, 0x63
 a1: LDI R17, 0x63
 a2: LDI R18, 0X21
 a3: DEC R18
     BRNE a3
     DEC R17
     BRNE a2
     DEC R16
     BRNE a1
 c2: LDI R16, 0x00
     OUT PORTB, R16

     LDI R16, 0x63
 b1: LDI R17, 0x63
 b2: LDI R18, 0X21
 b3: DEC R18
     BRNE b3
     DEC R17
     BRNE b2
     DEC R16
     BRNE b1


    DEC R0; Each decrement corresponds to one blink in led
    BRNE c1; blink until R0 != 0, i.e., blink 10 times
    POP R16; pop R16 from stack (when led has blinked 10 times)
    OUT SREG, R16;

    RETI; after POPing I = 0
```

<u>(i) Logic/Flowchart:</u> (for C program)

We define F_CPU with a value of 10^6(in Hz) which is the value of clock frequency for Atmega8 microcontroller. Also include avr library, delay library and interrupt library. Then we code our ISR for corresponding interrupt (INT0/INT1) and start with a for loop that does 10 iterations (10 times for led to blink).

Setting PORTB with a value of 0x01 to turn “ON” the LED and then we add a delay of 1000ms (i.e., 1 second) with the help of ***_delay_ms()*** function. After this, PORTB is set to 0x00 to turn “OFF” the LED and we again add a delay of 1000ms before it glows again.

In the main loop, DDRD is set to 0x00 which gives the direction to make PIND bits as input and where we receive the interrupts. To make PB0 (PORTB PIN 0) as output, DDRB is set as 0x01. For low level triggered interrupt to occur, we need both ISC01 and ISC00 to be 0 by setting MCUCR with 0x00.

Corresponding GICR values are loaded for the interrupt (GICR = 0x80 for INT1 and GICR = 0x40 for INT0). PORTB being set to 0x00 indicates that LED is turned “OFF” initially.

Global interrupt bit I is enabled by using ***sei ()*** function and an infinite loop (while (1)) is defined indicating that the microcontroller is doing some work continuously (until and unless interrupted).

So, shortly the program will be implemented in a way that LED blinks for 10 times when the interrupt is received. It (microcontroller) stops what it’s doing and jumps to the ISR loop and executes the ISR by blinking of LED for 10 times, and then it returns to the main loop.

NOTE: The ind_loop in above assembly program is analogous to the infinite loop (while (1)) in C-Program. Also, the same flowchart above (in assembly one) can be used to code in C-interface too (logic remains the same, just some change in name of variables and implementation).

(ii) Codes: -

C-PROGRAM USING INT1

```
#define F_CPU 1000000 // clock frequency

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT1_vect)
{
       int i;
       for (i=1;i<=10;i++) // for 10 times LED blink

       {
              PORTB=0x01;
              _delay_ms(1000); // delay of 1 sec
              PORTB=0x00;
              _delay_ms(1000);

       }


}
int main(void)
{
       //Set the input/output pins appropriately
       //To enable interrupt and port interfacing
       //For LED to blink
       DDRD=0x00; //Set appropriate data direction for D
       DDRB=0x01; //Make PB0 as output
       MCUCR=0x00; //Set MCUCR to level triggered
       GICR=0x80; //Enable interrupt 1
       PORTB=0x00;
       sei(); // global interrupt flag

       while (1) //wait
       {

       }
}
```

C-PROGRAM USING INT0

```
#define F_CPU 1000000 // clock frequency

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT1_vect)
{
	int i;
	for (i=1;i<=10;i++) // for 10 times LED blink

	{
		PORTB=0x01;
		_delay_ms(1000); // delay of 1 sec
		PORTB=0x00;
		_delay_ms(1000);

	}


}
int main(void)
{
	//Set the input/output pins appropriately
	//To enable interrupt and port interfacing
	//For LED to blink
	DDRD=0x00; //Set appropriate data direction for D
	DDRB=0x01; //Make PB0 as output
	MCUCR=0x00; //Set MCUCR to level triggered
	GICR=0x40; //Enable interrupt 0
	PORTB=0x00;
	sei(); // global interrupt flag

	while (1) //wait
	{

	}
}
```

6.) Inferences/Learning: -

→ We learnt how to generate and manage hardware interrupts using a push button in AVR microcontroller (atmega8) in both assembly program and C language.

→ We learnt how to set (reset, change etc.) the various registers related to interrupts (MCUCR, GICR etc.) to implement our task.

→ We also learnt AVR I/O Programming, i.e., how enable the port as input or output.

→ We learnt about the instruction delays for various instructions (like LDI, DEC, BRNE etc.) which generate time delays.

→ Using the above (point 3) instruction delays we learnt how to add a time delay in assembly program by manipulating the number of instructions used. Whereas a simple delay function is available in C-language to generate required delays.