EE2016-LAB REPORT

EXPERIMENT 2: AVR INTERRUPT PROGRAMMING

JAWHAR S

EE20B049

TARGET OF THE EXPERIMENT:

The main target of this experiment is to implement interrupts and timers in Atmel AtMega microcontroller using Atmel AVR assembly language programming as well as C-interfacing.

Aims of this experiment are:

(i) Generating an external (logical) hardware interrupt using an emulation of a push button switch.
(ii) Writing an ISR to switch ON an LED for a few seconds (10 secs) and then switch OFF.

(iii) Additionally, using the 16-bit timer to make an LED blink with a duration of 1 second.

QUESTIONS (FROM THE HANDOUT):

Two files, EE2016F21Exp2int1.asm and EE2016F21Exp2.int1.c are given in Moodle. The former one is an assembly program which implements interrupt using int1, while the later one is a 'C' program using int1.

The programs should emulate the following:

Once the switch is pressed the LED should blink 10 times (ON (or OFF) - 1 sec, duty cycle could be 50 %).

The tasks are:

1. Filling in the missing lines of codes given in EE2016F21Exp2int1.asm.

2. Using int0 to redo the same.

3. Filling in the missing lines of codes given in EE2016F21Exp2.int1.c

4. Using int0 to redo the same.

Solutions to the Questions:

(a) Flowchart/Logical Explanation:

This section provides the logic behind the codes written for the questions. Also, it contains two flowcharts for AVR Assembly and C-Interfacing.

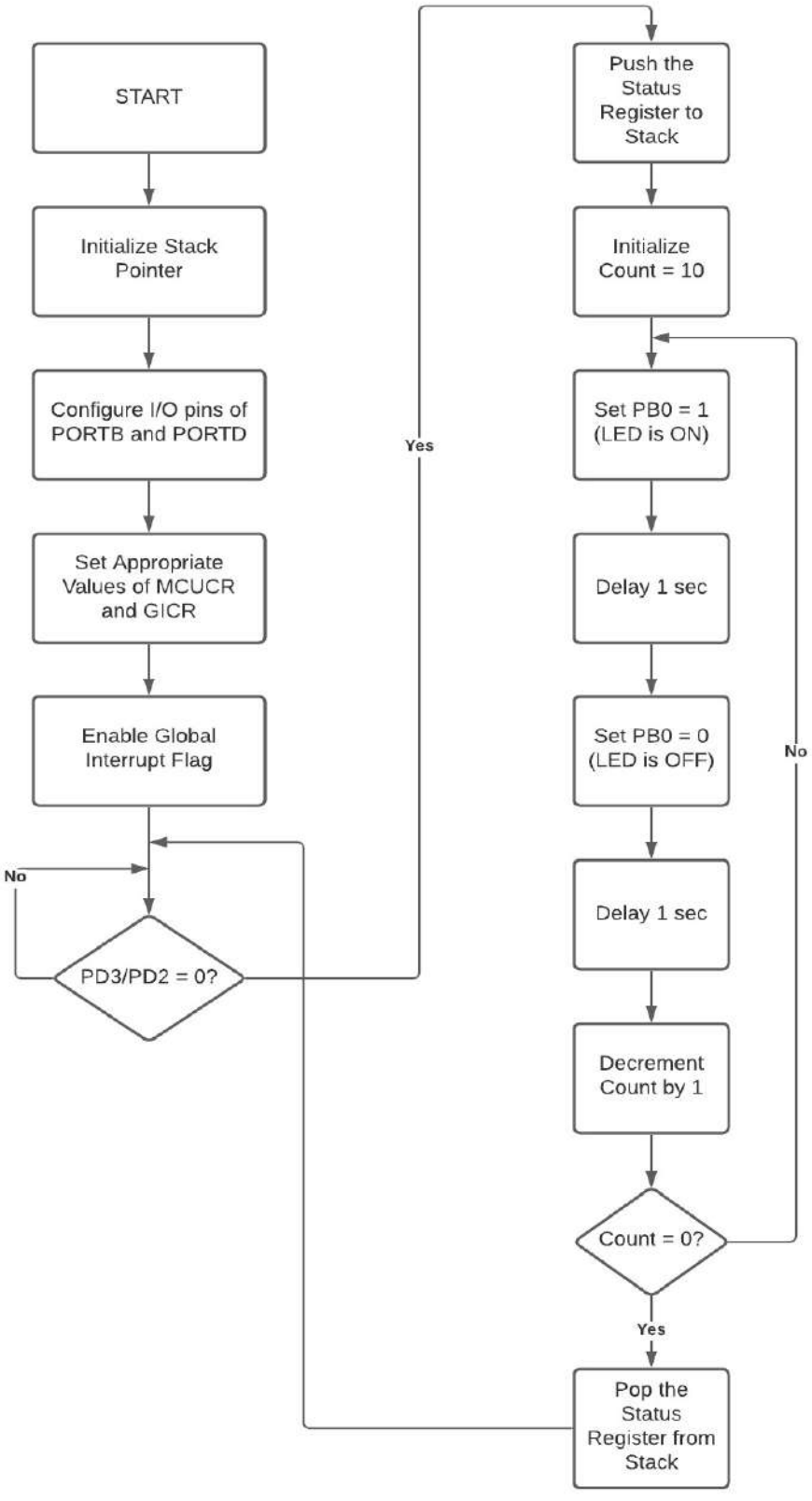
While there are four codes to write, the logic/objective behind the codes is the same. i.e., the codes should generate an Interrupt when the push button is pressed and emulate LED blinking 10 times (ON (or OFF) - 1 sec, duty cycle is 50 %).

The program starts by initializing the Stack Pointer. Then, the I/O pins of PORTB and PORTD are configured by setting DDRB = 0x01 (PB0 as Output) and DDRD = 0x00 (PD3 or PD2 as Input).

For generating low level-triggered interrupts using INT0, ISC01 and ISC00 bits of MCUCR has to be set to 0, and INT0 bit of GICR has to be set to 1. Hence the values of MCUCR and GICR for INT0 is 0x00 and 0x40 respectively.

For generating low level-triggered interrupts using INT1, ISC11 and ISC10 bits of MCUCR has to be set to 0, and INT1 bit of GICR has to be set to 1. Hence the values of MCUCR and GICR for INT0 is 0x00 and 0x80 respectively. The flowchart of AVR Assembly implementation is given in the next page.

Flowchart for AVR Assembly



After setting appropriate values for MCUCR and GICR according to INT0 or INT1, the Global Interrupt Flag is enabled. Then, the program checks whether the push button is pressed (PD2 = 0 for INT0 and PD3 = 0 for INT1). If the Push button is pressed, the contents of the Status Register is pushed into the Stack and the program flow shifts to the ISR (interrupt service routine).

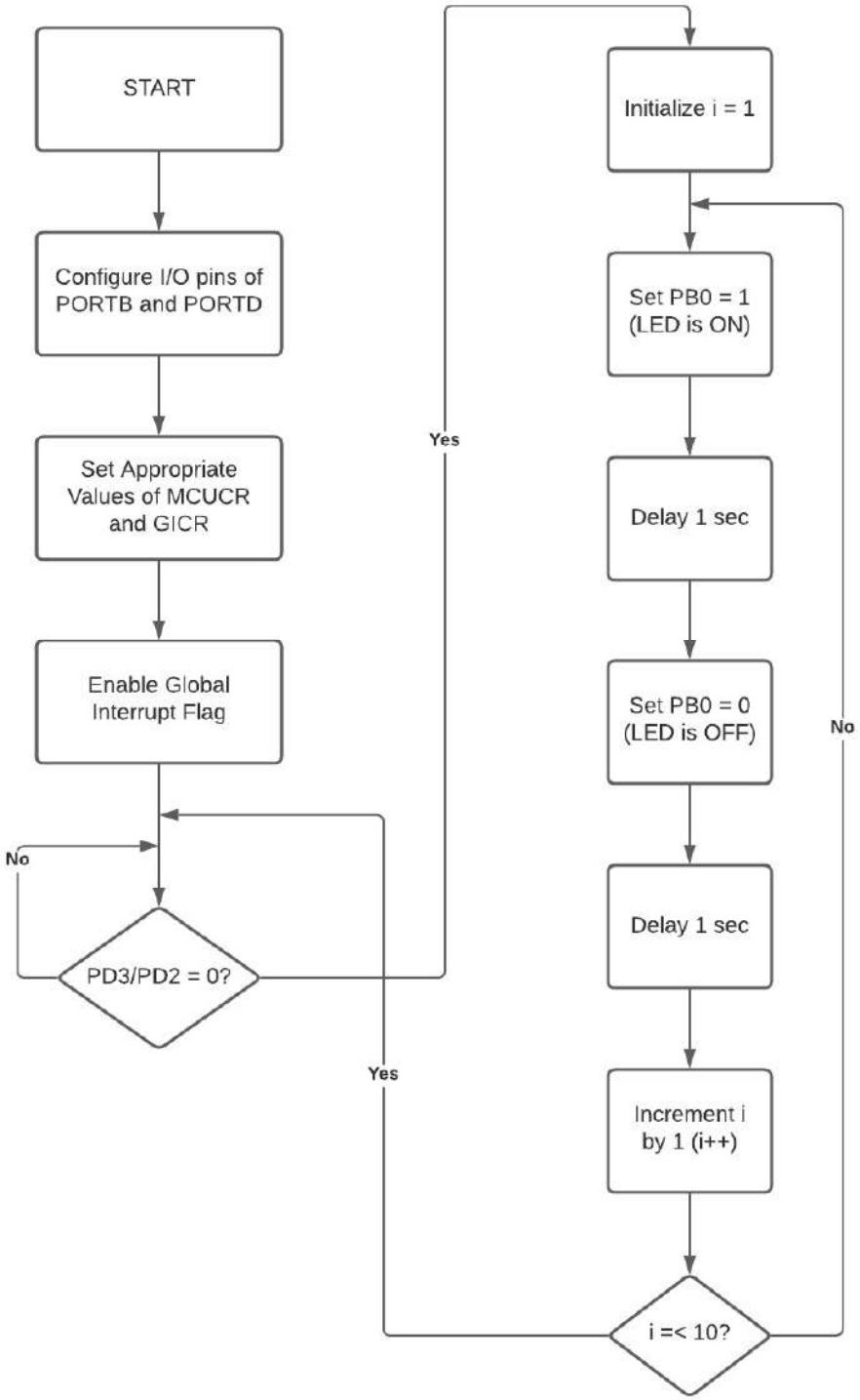
In the ISR, Count is initialized to 10 (Since the LED should blink 10 times). Then a signal is given to switch on the LED (PB0 = 1) and wait for 1 second. Then the signal is turned off (PB0 = 0) and wait for 1 second. Then the Count is decremented by 1.

Now, the program checks whether the LED has blinked 10 times (Count = 0). If the LED did not yet blink 10 times, the program flow shifts to make the LED blink once more. If the LED has blinked 10 times, then the contents of the Status register is popped and the program flow shifts to the main program and it checks for further interrupts.

For C-Interfacing too, the same procedure happens except that the codes for initializing the stack pointer, pushing the contents of status register into stack and popping the contents of status register from stack is not explicit. The C-Compiler does it for us.

The flowchart of C-Interfacing implementation is given in the next page.

Flowchart for C-interfacing



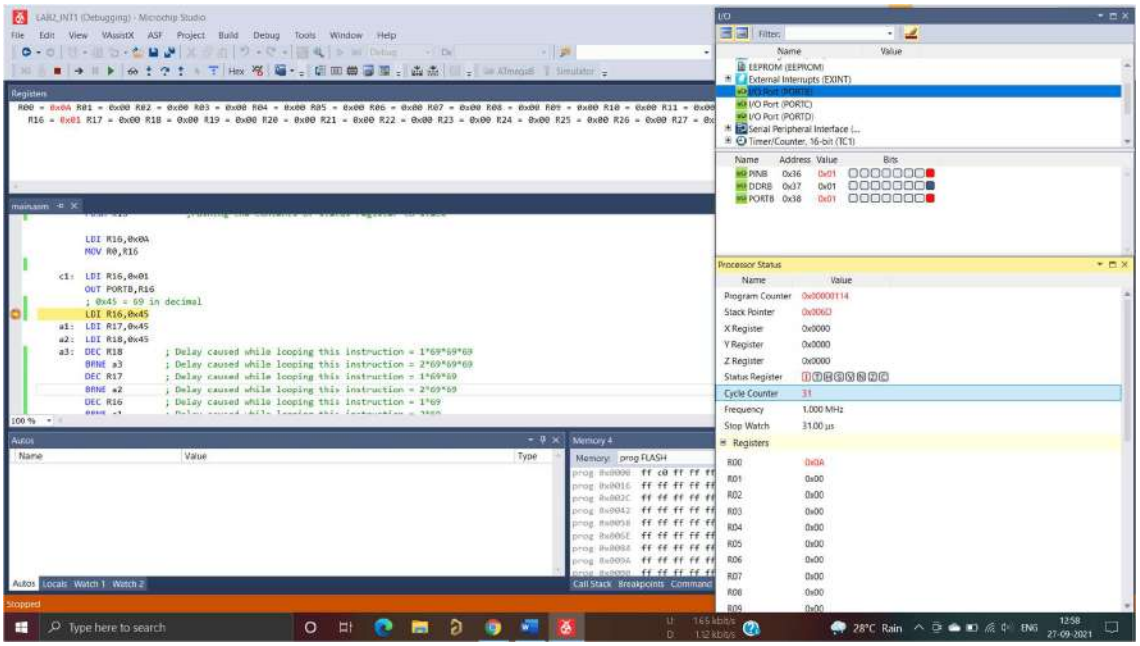
(b) Codes for the Questions:

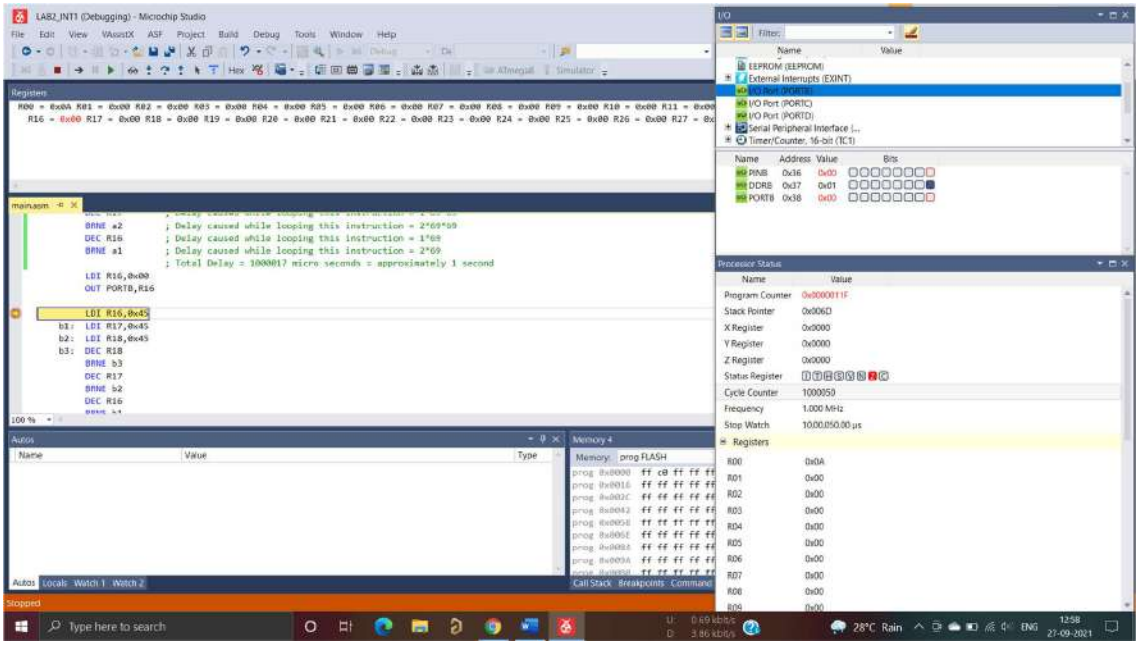
1. AVR Assembly: INT1

```
.org 0x0000
rjmp reset

.org 0x0004
rjmp int1_ISR

.org 0x0100

reset:

      LDI R16,0x70
       OUT SPL,R16
       LDI R16,0x00
       OUT SPH,R16

       LDI R16,0x01
       OUT DDRB,R16

       LDI R16,0x00
       OUT DDRD,R16

       LDI R16,0x00
       OUT MCUCR,R16

       LDI R16,0x80
       OUT GICR,R16

       SEI
ind_loop:rjmp ind_loop

int1_ISR:IN R16,SREG
           PUSH R16

           LDI R16,0x0A
           MOV R0,R16

     c1:   LDI R16,0x01
           OUT PORTB,R16
           LDI R16,0x45
     a1:   LDI R17,0x45
     a2:  LDI R18,0x45
     a3:   DEC R18
           BRNE a3
           DEC R17
           BRNE a2
```

```
        DEC R16
        BRNE a1
        LDI R16,0x00
        OUT PORTB,R16

        LDI R16,0x45
b1:     LDI R17,0x45
b2:   LDI R18,0x45
b3:     DEC R18
        BRNE b3
        DEC R17
        BRNE b2
        DEC R16
        BRNE b1

        DEC R0
        BRNE c1
        POP R16
        OUT SREG, R16
        RETI
```

Below are the screenshots for verifying that an 1 second delay is produced.





2. AVR Assembly: INT0

```
.org 0x0000
rjmp reset

.org 0x0002
rjmp int0_ISR

.org 0x0100

reset:

       LDI R16,0x70
        OUT SPL,R16
        LDI R16,0x00
        OUT SPH,R16

        LDI R16,0x01
        OUT DDRB,R16

        LDI R16,0x00
        OUT DDRD,R16

        LDI R16,0x00
        OUT MCUCR,R16

        LDI R16,0x40
        OUT GICR,R16
```

```
        SEI
ind_loop:rjmp ind_loop

int0_ISR:IN R16,SREG
            PUSH R16

            LDI R16,0x0A
            MOV R0,R16

      c1:   LDI R16,0x01
            OUT PORTB,R16

            LDI R16,0x45
      a1:   LDI R17,0x45
      a2:  LDI R18,0x45
      a3:   DEC R18
            BRNE a3
            DEC R17
            BRNE a2
            DEC R16
            BRNE a1

            LDI R16,0x00
            OUT PORTB,R16

            LDI R16,0x45
      b1:   LDI R17,0x45
      b2:  LDI R18,0x45
      b3:   DEC R18
            BRNE b3
            DEC R17
            BRNE b2
            DEC R16
            BRNE b1

            DEC R0
            BRNE c1
            POP R16
            OUT SREG, R16
            RETI
```

Delay Calculations in AVR Assembly Programs:

For producing a delay of 1 second, we use the delays caused by executing instructions.

```
        LDI R16,0x45   ; 0x45 = 69 in decimal
b1:     LDI R17,0x45
b2:     LDI R18,0x45
b3:     DEC R18        ; DEC is executed 69x69x69 times
        BRNE b3        ; BRNE is executed 69x69x69 times
        DEC R17        ; DEC is executed 69x69 times
        BRNE b2        ; BRNE is executed 69x69 times
        DEC R16        ; DEC is executed 69 times
        BRNE b1        ; BRNE is executed 69 times
```

DEC instruction has 1 microsecond delay and BRNE has 2 microseconds delay.

Hence total delay = 3x69x69x69 + 3x69x69 + 3x69 = 10,00,017 microseconds, which is approximately 1 second.

3. C-Interfacing: INT1

```c
#define F_CPU 1000000

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

ISR (INT1_vect)
{
      int i;
      for (i=1;i<=10;i++)

      {
            PORTB=0x01;
            _delay_ms(1000);
            PORTB=0x00;
            _delay_ms(1000);

      }
```

```
}


int main(void)
{
	DDRD=0x00;
	DDRB=0x01;
	MCUCR=0x00;
	GICR=0x80;
	PORTB=0x00;
	sei();

	while (1)
	{

	}
}
```

4. C-Interfacing: INT0

```
#define F_CPU 1000000

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

ISR (INT0_vect)
{
	int i;
	for (i=1;i<=10;i++)

	{
		PORTB=0x01;
		_delay_ms(1000);
		PORTB=0x00;
		_delay_ms(1000);

	}

}

int main(void)
{
	DDRD=0x00;
	DDRB=0x01;
	MCUCR=0x00;
```

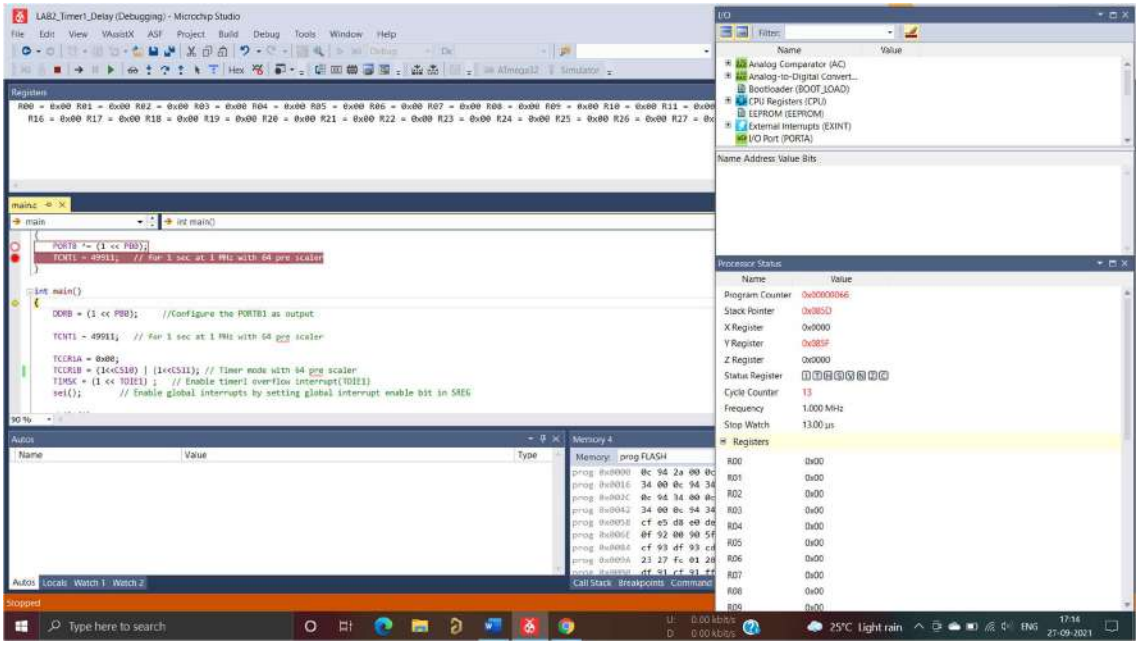
```
	GICR=0x40;
	PORTB=0x00;
	sei();



	while (1)
	{

	}
}
```

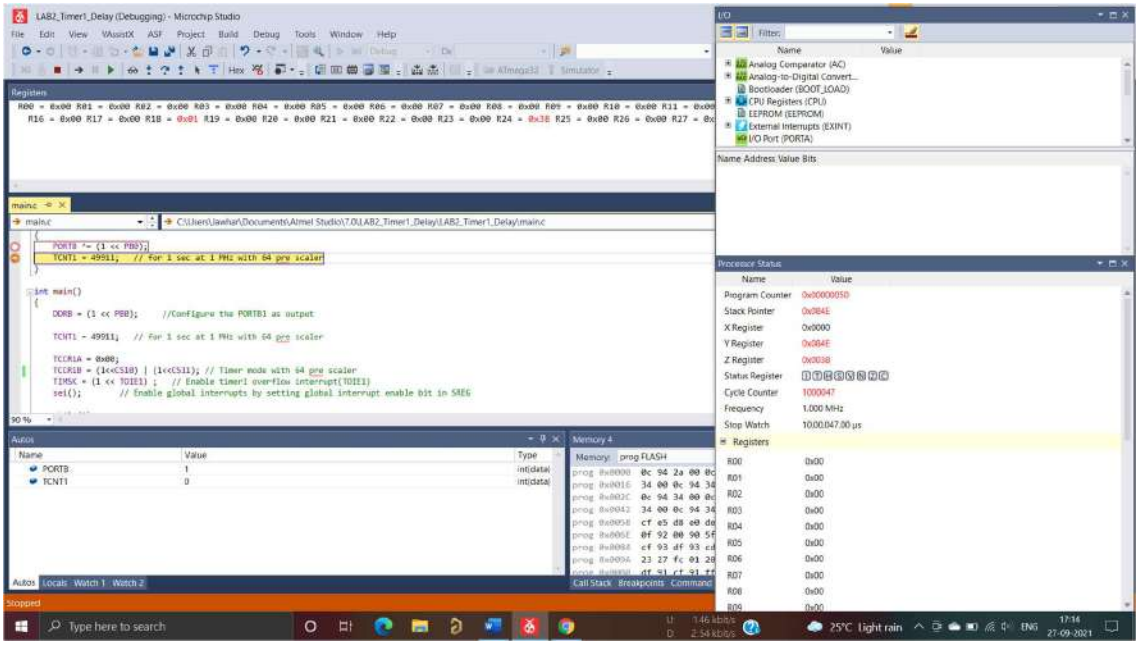
Code for the Additional Problem:

```
#include <avr/io.h>
#include <avr/interrupt.h>

ISR (TIMER1_OVF_vect)    // Timer1 ISR
{
	PORTB ^= (1 << PB0);
	TCNT1 = 49911;   // for 1 sec at 1 MHz with 64 pre scaler
}

int main()
{
	DDRB = (1 << PB0);     //Configure PORTB1 as output

	TCNT1 = 49911;   // for 1 sec at 1 MHz with 64 pre scaler

	TCCR1A = 0x00;
	TCCR1B = (1<<CS10) | (1<<CS11); //Timer mode with 64 prescaler
	TIMSK = (1 << TOIE1) ;   // Enable timer1 overflow interrupt
	sei();        // Enable global interrupt flag

	while(1)
	{

	}
}
```

The frequency of the timer with 64 pre scaler is 15625 Hz. Hence the time delay for incrementing TCNT0 by 1 is 6.4×10^{-5} seconds. Therefore, 15625 such counts will be needed for achieving a delay of 1 second. So TCNT0 is initialized as 65536-15625 = 49911.

Below are the screenshots to verify that the Timer produces a delay of 1 second.





LEARNINGS FROM THE EXPERIMENT:

- Learnt branching, looping and creating delays using looping through instructions in both AVR Assembly and C.
- Learnt programming for interfacing various I/O Ports in Atmel AVR microcontroller.
- Understood Interrupts and programmed them in both AVR Assembly and C to emulate LED blinking when the push button is pressed.
- Learnt about Timers and programmed them to generate delays
- Learnt about various tools in Microchip Studio like breakpoints, I/O panel, Processor Status used for emulating and verifying the code.