American International University- Bangladesh
Department of Electrical and Electronic Engineering
EEE 4103: Microprocessor and Embedded Systems Laboratory

Title: Familiarization of assembly language program and Interrupts in a microcontroller.

Introduction:

The objectives of this experiment are to
1. Study the assembly language program of an Arduino.
2. Write assembly language programming code for an Arduino.
3. Build a circuit to turn on and off an LED on an Arduino Microcontroller Board connected to an I/O port of the microcontroller.
4. Study of the external interrupts of an Arduino using its digital I/O port.
5. Build a circuit to turn on and off an LED on an Arduino Microcontroller Board connected to an I/O port of the microcontroller due to the external interrupt.

Theory and Methodology:

A. Assembly Language

To study assembly language programming using Arduino IDE, we need an Arduino Microcontroller Board with connecting cable, LEDs, switches, resistors, assembly language programming knowledge, compiler, loader, etc. Assembly language is written in the Arduino IDE and uploaded into the microcontroller via the USB cable after successfully compiling the code. This is shown in Fig. 1.

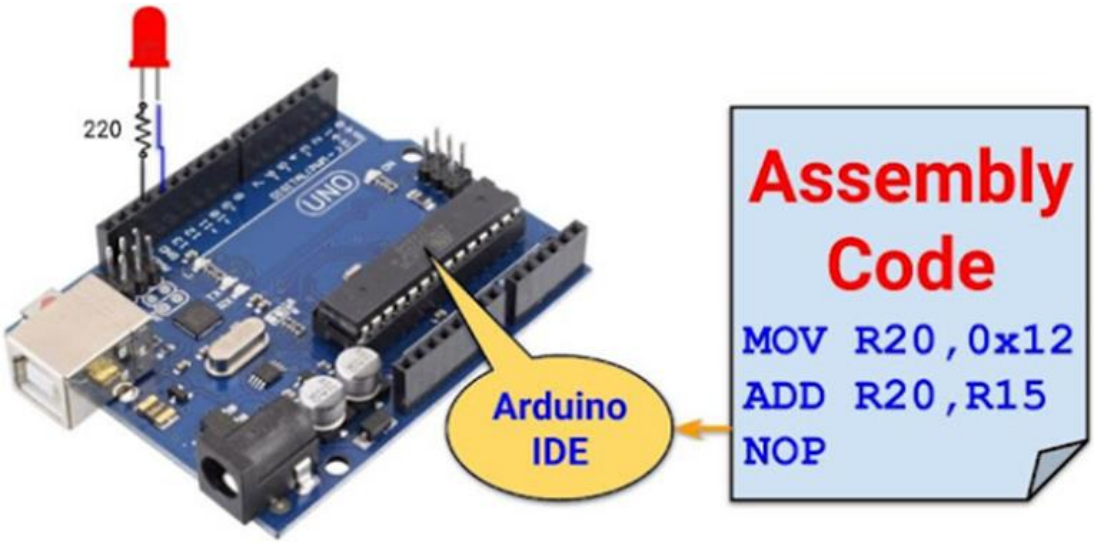


Fig. 1 Assembly programming with Arduino IDE

For this purpose, we need to know that the codes should be written in two different files with the file extension being .ino and .S with the same file name as shown in Fig. 2. Both files must be in the same directory. We know that the Arduino has 32 general-purpose registers as shown in Fig. 3 with their physical address being on the left side. Arduino has 4 I/O ports for programming, namely ports A, B, C, and D. This is shown in Fig. 4. Each port is 8-bit wide and can be configured as an input and output port. For each port, there is a special purpose register called the data direction register, which defines the direction of signal flow to or from the Arduino microcontroller. These ports can be used to perform other functions as well. This is explained in Fig. 5.

Fig. 2 Assembly programming via Arduino IDE

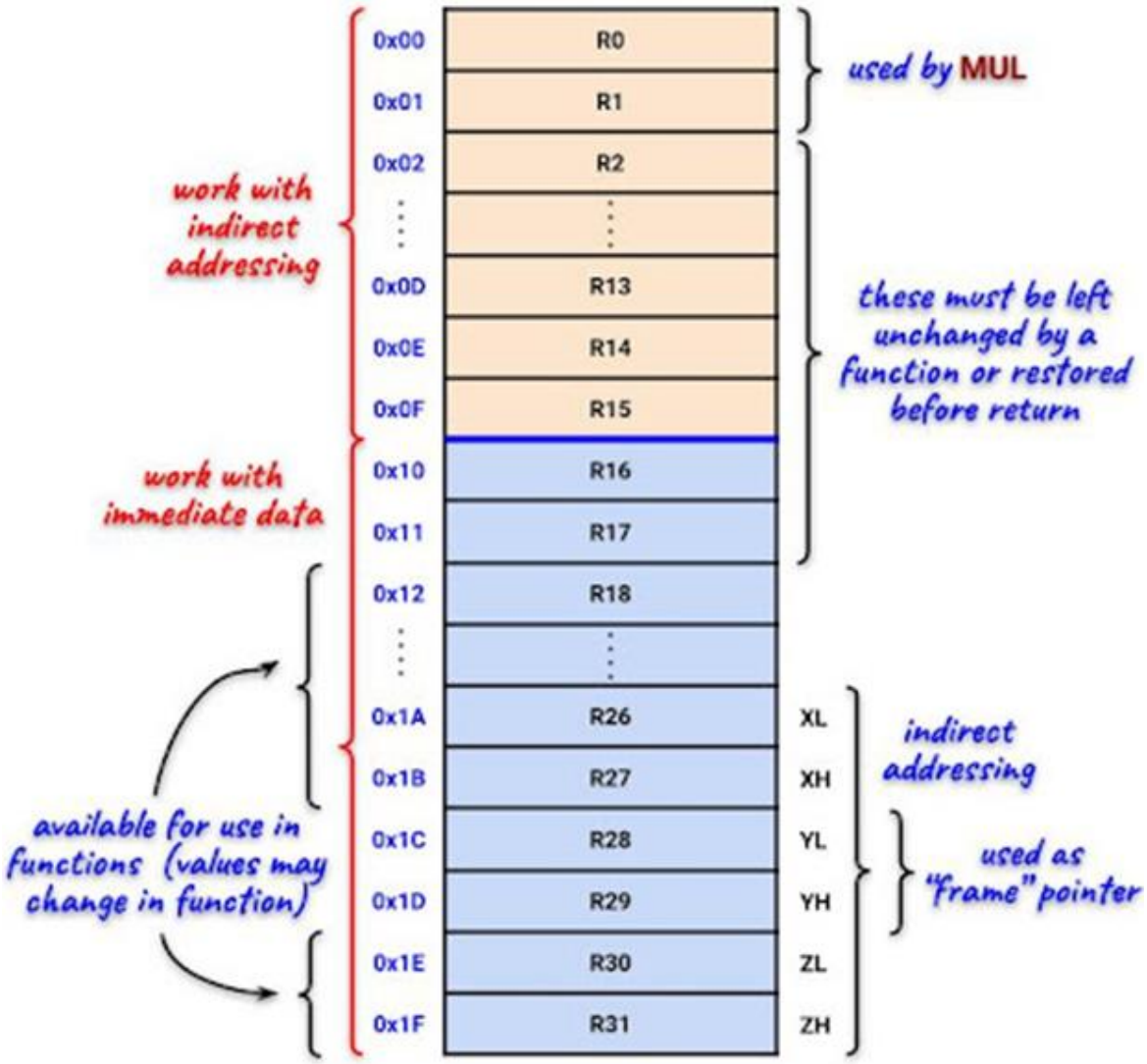


Fig. 3 General-purpose registers of the Arduino (ATMega328P)

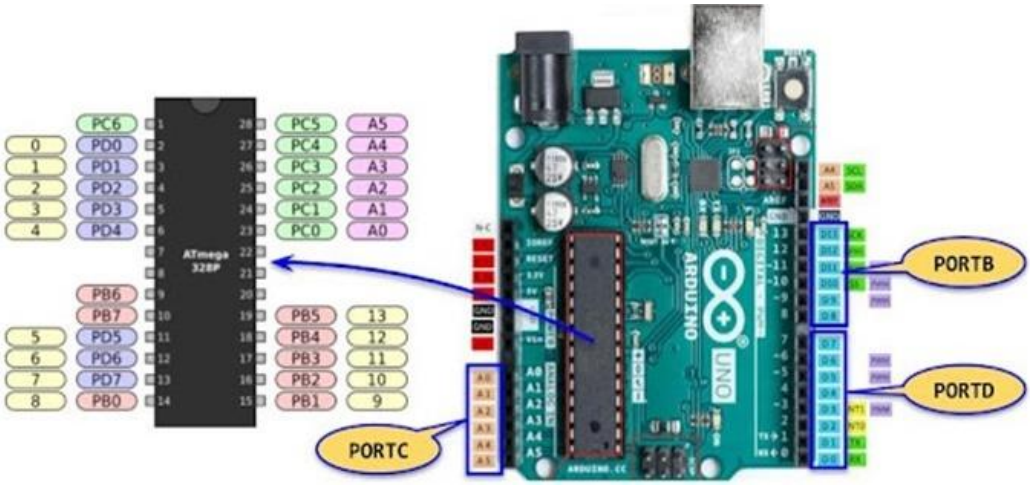


Fig. 4 I/O ports of the Arduino (ATMega328P)

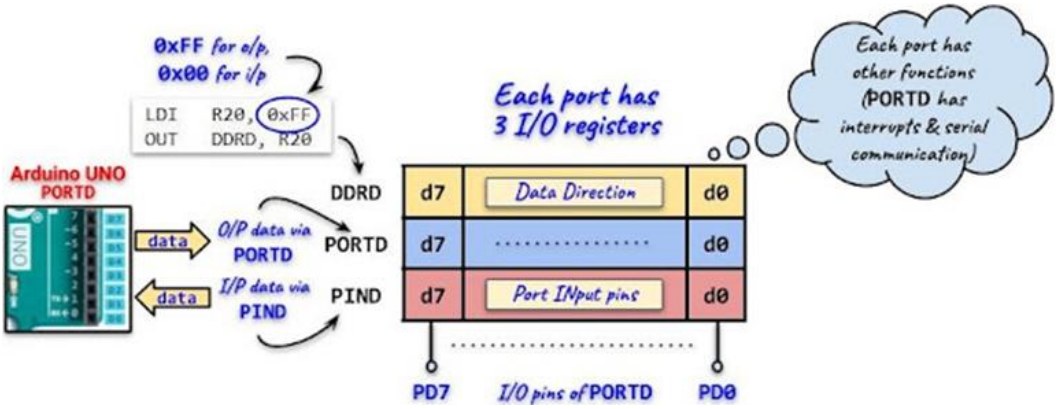


Fig. 5 Assembly programming of I/O ports of the Arduino (ATMega328P)

B. Interrupts

An interrupt is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution. An interrupt is a signal emitted by a device attached to a computer or from a program within the computer. It requires the operating system (OS) to stop and figure out what to do next. An interrupt temporarily stops or terminates a service or a current process. The hardware event can either be a busy-to-ready transition in an external I/O device (like the UART input/output) or an internal event (like a bus fault, memory fault, or a periodic timer).

Microcontroller normally executes instructions in an orderly fetch-execute sequence as dictated by a user-written program.

However, a microcontroller must also be ready to handle unscheduled events that might occur inside or outside the microcontroller. The interrupt system onboard a microcontroller allows it to respond to these internally and externally generated events. By definition, we do not know when these events will occur. When an interrupt event occurs, the microcontroller will normally complete the instruction it is currently executing and then will transfer the program control to an Interrupt Service Routine (ISR) that handles the interrupt. Once the ISR is complete, the microcontroller will resume processing where it left off before the interrupt event occurred.

A request for the processor to 'interrupt' the currently executing process, so the event can be processed on time. It is also referred to as a 'trap'. If the request is accepted, the processor suspends current processes, saves its states, and executes the Interrupt Service Routine (ISR), Interrupt Handler (IH), or Interrupt Service Procedure (ISP). The concept of an 'Interrupt' is very useful when implementing any of the switch debouncing methods though the scheduled events are not termed as interrupts.

The interrupt vectors and vector table are crucial to the understanding of hardware and software interrupts. Interrupt vectors are addresses that inform the interrupt handler as to

where to find the ISR. Misspelling the vector name (even wrong capitalization) will result in the ISR not being called and it will not also result in a compiler error. Interrupt Service Routines are functions with no arguments or parameters. Some Arduino libraries are designed to call these functions, so the user just needs to supply an ordinary function, e.g.,

```
// Interrupt Service Routine (ISR)
void pinChange ()
{
    flag = true;
}  // end of pinChange
```

As per the datasheet, the minimal amount of time to service an interrupt is 82 clock cycles in total. Assuming a 16 MHz clock, there would be 62.5 ns time needed per clock cycle. So, the total time required is $62.5 \times 82 = 5125$ ns = 5.125 μs.
When an ISR is entered, interrupts are disabled. Naturally, they must have been enabled in the first place, otherwise, the ISR would not be entered. However, to avoid having an ISR itself be interrupted, the processor turns interrupts off.
When an ISR exits, then interrupts are enabled again. The compiler also generates code inside an ISR to save registers and status flags so that whatever you were doing when the interrupt occurred will not be affected. However, you can turn interrupts on inside an ISR if you absolutely must.

```
// Interrupt Service Routine (ISR)
void pinChange ()
{
    // handle pin change here
    interrupts ();  // allow more interrupts
 }  // end of pinChange
```

Re-enables interrupts (after they've been disabled by noInterrupts()). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of a code, however, and may be disabled for particularly critical sections of a code.

Syntax: interrupts() and noInterrupts(), all are opposite of interrupts()
Parameters: None
Returns: Nothing
Example Code: The code enables Interrupts.

```
void setup() {}

void loop() {
    noInterrupts();
    // critical, time-sensitive code here
    interrupts();
    // other code here
}
```

The main interrupt flag, global interrupt is used to turn all the interrupts on or off. For example, sei() is a globally enabled interrupt. This is available in the Bit7 of the Status Register (SREG) of the Arduino.

The ATmega328P provides support for 25 different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector located at the lowest addresses in the Flash program memory space. The complete list of "Reset and Interrupt Vectors" in the ATMega328P is shown in Table 1. Each Interrupt Vector occupies two instruction words. The list determines the priority levels of the different interrupts. The lower the address the higher the priority level. RESET has the highest priority, and the next one is INT0, i.e., the External Interrupt Request 0.

Table 1 Atmega328P Interrupt Vector Table

VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

A trigger is an asynchronous event that causes the interrupt. Once the interrupt is triggered and processed, the interrupt flag is cleared, and clearing the interrupt flag is called acknowledgment. The module that is executed when hardware requests an interrupt. There may be one large ISR handling all the interrupt requests, or many small ISRs handling the many interrupts (interrupt vectors). For example, if the Timer0 of an Arduino enables an interrupt due to the overflow of Timer0 when its TCNT0 is counting, and overflow occurs then its TOV0 bit of the TIFR0 register is set and at the same time TOIE0 bit of the TIMSK0 register is set along with the I-bit of the SREG register is set. In such a case, the following ISR is called:

ISR(TIMER0_OVF_vect) // enabling overflow vector inside Timer0 using an ISR

When its TCNT0 is counting, and its value matches with a value stored in its OCR0A register then its OCF0 bit of the TIFR0 register is set and at the same time OCIE0A bit of

the TIMSK0 register is set along with the I-bit of the SREG register is set. In such a case, the following ISR is called:

ISR(TIMER0_COMPA_vect) // This is the Timer0 Compare 'A' interrupt service routine.

Remember, the ISR is a separate routine and requires a separate flowchart to represent. Some of the ISR names are given in Table 2.

Table 2 List of interrupts, in priority order, for the ATmega328P

1	Reset	
2	External Interrupt Request 0 (pin D2)	(INT0_vect)
3	External Interrupt Request 1 (pin D3)	(INT1_vect)
4	Pin Change Interrupt Request 0 (pins D8 to D13)	(PCINT0_vect)
5	Pin Change Interrupt Request 1 (pins A0 to A5)	(PCINT1_vect)
6	Pin Change Interrupt Request 2 (pins D0 to D7)	(PCINT2_vect)
7	Watchdog Time-out Interrupt	(WDT_vect)
8	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)
9	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)
10	Timer/Counter2 Overflow	(TIMER2_OVF_vect)
11	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
12	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
13	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
14	Timer/Counter1 Overflow	(TIMER1_OVF_vect)
15	Timer/Counter0 Compare Match A	(TIMER0_COMPA_vect)
16	Timer/Counter0 Compare Match B	(TIMER0_COMPB_vect)
17	Timer/Counter0 Overflow	(TIMER0_OVF_vect)
18	SPI Serial Transfer Complete	(SPI_STC_vect)
19	USART Rx Complete	(USART_RX_vect)
20	USART, Data Register Empty	(USART_UDRE_vect)
21	USART, Tx Complete	(USART_TX_vect)
22	ADC Conversion Complete	(ADC_vect)
23	EEPROM Ready	(EE_READY_vect)
24	Analog Comparator	(ANALOG_COMP_vect)
25	2-wire Serial Interface (I2C)	(TWI_vect)
26	Store Program Memory Ready	(SPM_READY_vect)

The registers for Timer0 are shown in Figs. 6-8.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 6 TIMSK0 – Timer/Counter0 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	-	-	-	-	-	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 7 TIFR0 – Timer/Counter0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 8 SREG – Status Register

In this experiment, we will study the use of the Timer1 interrupt vector to make an LED blink every 500 ms and use an external switch to call an interrupt.

The values of TCCR1A and TCCR1B registers are reset to 0 to make sure everything is clear. TCCR1B was set equal to 00000100 for a prescalar of 256. If you want to set ones, an OR operation can be used. If you want to set zeros, an AND operation can be used.
The compare match mode for the OCR1A register is enabled. For that, the OCIE1A bit was set to be a 1 and that's from the TIMSK1 register. So, we equal that to OR and this byte 00000010.
The value of the OCR1A register was set to 31250 so that we will have an interruption each 500 ms. Each time the interrupt is triggered, we go to the related ISR vector. Since we have 3 timers, we have six ISR vectors, two for each timer and their names are:

TIMER1_COMPA_vect, TIMER2_COMPA_vect, TIMER0_COMPA_vect, TIMER1_COMPB_vect, TIMER2_COMPB_vect, TIMER0_COMPB_vect

Since in this experiment, **Timer1** and output compare register A (OCR1A) are used; so, we need to use the ISR named **TIMER1_COMPA_vect**. As such, below the void loop function, the interruption routine is defined. Inside this interruption, the state of the LED is inverted, and a digitalWrite() function is used. But at first, the timer's current value of the TCNT1 register is reset. Otherwise, it will continue to count up to its maximum value from its matching value of 31250. So, for each 500 ms, this code will run and invert the LED state and, in this way, a blink of the LED connected to pin D5 is obained.
We will use a **pre-scalar** value of 256 to reduce the timer clock frequency to 16 MHz/256 = 62.5 kHz, with a **new timer clock period = 1/62.5 kHz = 16 µs**.
Thus, the required timer count,

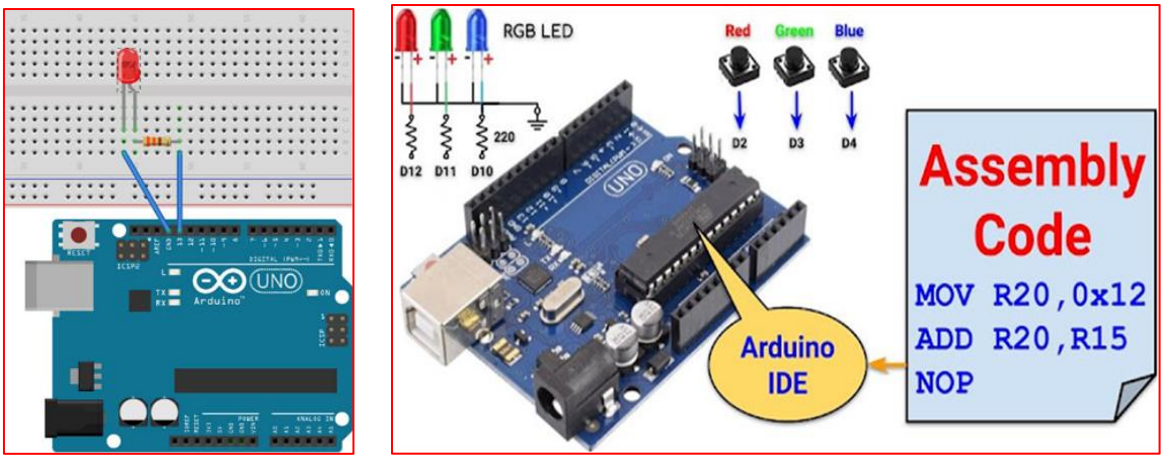
$$TC = \frac{Required\ Delay}{Timer\ Clocked\ period\ after\ scaled} - 1 = \frac{500\times10^3}{16} - 1 = 31250 - 1 = 31249.$$

So, this is the value that we should store in the OCR1A register.

Apparatus:

1) Arduino IDE (2.0.1 or any recent version)
2) Arduino Microcontroller board
3) PC having an Intel processor
4) LED lights (Red, Green, Yellow, 1 pc each)
5) One 100 Ω resistor
6) Three push switches
7) Jumper wires

Experimental Setup:



(a) (b)

Fig. 9 Experimental setup of an LED control system using an Arduino Microcontroller Board.

Experimental Procedure:

The main task of this experiment is to implement an LED light blinks and control system using a push switch. Connect the circuits as per the diagram of Figs. 9 (a) and 9 (b). Then plug the Arduino microcontroller board into the PC.

Using Arduino IDE to write code:

1. Open the Arduino Uno IDE 2.3.2 (version may be updated automatically on your computer) and a blank sketch will open. The window as in Fig. 10 will come up on your PC:

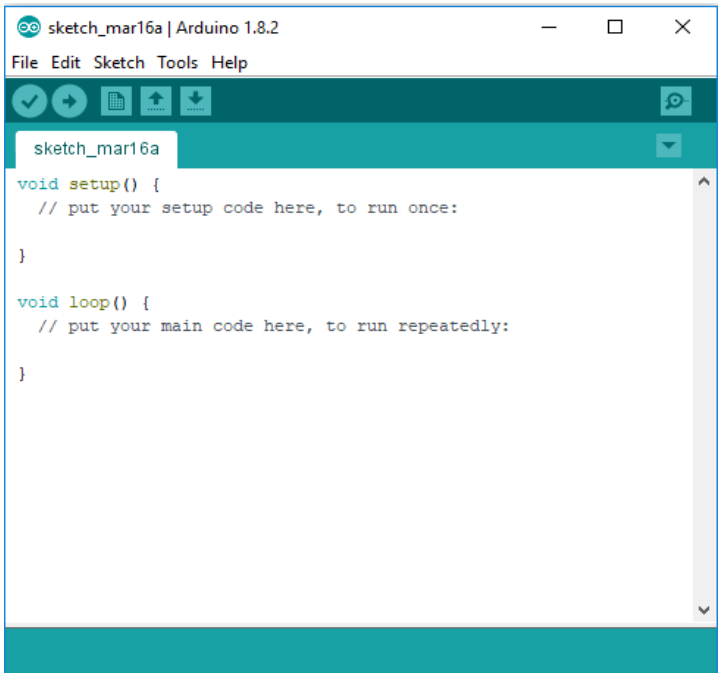


Fig. 10 IDE Environment (text editor)

2. **To do Assembly Experiments based on programs of Parts 1 and 2:** Write the program by copying the codes given below boxes in sequences on the blank sketch of Fig. 10 for the LED blink and control by using assembly language programs.
    a. Create *led.ino* and *led.S* files using the codes given below.
    b. Create a folder named LED and place the above two files in that folder.
    c. Open *led.ino* using Arduino IDE.
    d. Compile and upload to the hardware.
    e. Modify the program to blink an LED at digital PIN 12 with a different delay.

3. **To do Interrupts Experiments based on programs of Parts 3 and 4:** Write the program by copying the codes given below boxes in sequences on the blank sketch of Fig. 10 for the LED blink and control by using Interrupts.
    a. Create *pbintLED.ino* file using the codes given below.
    b. Compile this file and upload the .hex file to the Arduino board.
    c. Modify the program to turn on/off an LED at digital PIN 12 with a different delay.
    d. Next, create *blinkLED.ino* file using the codes given below.
    e. Compile this file and upload the .hex file to the Arduino board.
    f. Modify the program to blink an LED at digital PIN 12 with a different Timer and delay value, say 2 s.

Note: Please use the appropriate digital pins of the Arduino board according to the pins used in the program. Please note that the switch state must match the switch state used in the program for its proper operation.

Code of an LED light blink and LED control using a switch:

PART 1: Blink an LED [for Fig. 9 (a)]

The .ino file:

```
//-------------------------
// C Code for Blinking LED
//-------------------------
extern "C"
{
  void start();
  void led(byte);
}
//--------------------------------------------------
void setup()
{
  start();
}
//--------------------------------------------------
void loop()
{
  led(1);
  led(0);
}
```

The .S file:

```
;---------------
; Assembly Code
;---------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;------------------------
.global start
.global led
;------------------------
start:
    SBI   DDRB, 5;          set PB5 (D13) as o/p
    RET;                    return to setup() function
;-----------------------------------------------------------------------
led:
    CPI       R24, 0x00;        value in R24 passed by caller compared with 0
    BREQ    ledOFF;             jump (branch) if equal to subroutine ledOFF
    SBI      PORTB, 5;          set D13 to HIGH, i.e., the LED will turn ON
    RCALL myDelay;              Calling a delay function to determine the ON duration of LED
    RET;                        return to loop() function
;-----------------------------------------------------------------------
ledOFF:
    CBI       PORTB, 5;         set D13 to LOW, i.e., the LED will turn OFF
    RCALL myDelay;              Calling a delay function to determine the OFF duration of LED
    RET;                        return to loop() function
```

```
;-----------------------------------------------------------------------
.equ  delayVal, 10000;          initial count value for the inner loop (0010011100010000)
;-----------------------------------------------------------------------
myDelay:
   LDI   R20, 100;                initial count value for the outer loop
outerLoop:
   LDI   R30, lo8(delayVal);   low byte of delayVal in R30 (00010000)
   LDI   R31, hi8(delayVal);     high byte of delayVal in R31 (00100111)
innerLoop:
   SBIW  R30, 1;                   subtract 1 from 16-bit value in R31, R30
   BRNE  innerLoop;              jump if countVal not equal to 0
   ;--------------
   SUBI  R20, 1;                   subtract 1 from R20
   BRNE  outerLoop;              jump if R20 is not equal to 0
   RET
;-----------------------------------------------------------------------
```

PART 2: Push button LED control [for Fig. 9 (b)]

The btnLED.ino file:

```
//-----------------------------------
// C Code: RGB LED ON/OFF via Buttons
//-----------------------------------
extern "C"
{
  void start();
  void btnLED();
}
//----------------------
void setup()
{
  start();
}
//----------------------
void loop()
{
  btnLED();
}
```

The btnLED.S file:

```
;----------------------------------------
; Assembly Code: RGB LED ON/OFF via Buttons
;----------------------------------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;----------------------
.global start
.global btnLED
;=============================================================
start:
```

```
   SBI   DDRB, 4;          set PB4 (pin D12 as o/p - red LED)
   SBI   DDRB, 3;          set PB3 (pin D11 as o/p - green LED)
   SBI   DDRB, 2;          set PB2 (pin D10 as o/p - blue LED)
   CBI   DDRD, 2;          clear PD2 (pin D02 as i/p - red button)
   CBI   DDRD, 3;          clear PD3 (pin D03 as i/p - green button)
   CBI   DDRD, 4;          clear PD4 (pin D04 as i/p - blue button)
RET

btnLED:
L2:     SBIS    PIND, 4      ; Skips below statement if the push button of D04 is not pressed
        RJMP    L1
        SBI     PORTB, 2     ; Turn ON LED, PB2(D10), if SW of D04 is not pressed
        CBI     PORTB, 3     ; Turn OFF LED, PB3(D11), if SW of D04 is not pressed
        SBIC    PIND, 4      ; Skips below statement if the push button of D04 is pressed
        RJMP    L2

L1:     CBI     PORTB, 2     ; Turn OFF LED, PB2(D10), if SW of D04 is pressed
        SBI     PORTB, 3     ; Turn OFF LED, PB3(D11), if SW of D04 is pressed
        RET
```

Note: In this program one switch of D04 pin controls two LEDs connected to two pins of D10 and D11.

PART 3: Push button LED control [for Fig. 9 (a)]

The pbintLED.ino file:

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

Note: INT0 and INT1 are mapped with the pins PD2 and PD3 on ATMega328P.

PART 4: Blinking an LED using a Timer1 ISR [for Fig. 9 (b)]

The blinkLED.ino file:

```
bool LED_State = 'True';

void setup() {
```

```
  pinMode(13, OUTPUT);
  cli();                // stop interrupts till we make the settings
  TCCR1A = 0;           // Reset the entire A and B registers of Timer1 to make sure that
  TCCR1B = 0;           // we start with everything disabled.
  TCCR1B = 0b00000100; // Set CS12 bit of TCCR1B to 1 to get a prescalar value of 256.
  TIMSK1 = 0b00000010; // Set OCIE1A bit to 1 to enable compare match mode of A reg.
  OCR1A = 31250;    // We set the required timer count value in the compare register, A
  sei();                // Enable back the interrupts
}

void loop() {
        // put your main code here, to run repeatedly.
}

// With the settings above, this ISR will trigger each 500 ms.
ISR(TIMER1_COMPA_vect) {
        TCNT1 = 0;          // First, set the timer back to 0 so that it resets for the next interrupt
        LED_State = !LED_State;      // Invert the LED State
        digitalWrite(13, LED_State); // Write this new state to the LED connected to pin D5
}
```

Questions for report writing:

1) Include all codes and scripts in the lab report following the lab report writing template.
2) Show the output/results in the form of images. Give their captions and descriptions.
3) Configure the port numbers for outputs and inputs according to your ID. Consider the last six digits from your ID (if your ID is XY-PQABC-Z then consider Port B's and Port D's bits of PQABC and Z as the inputs and outputs, respectively). Include all the programs and results within your lab report.
4) Write a program that will control three LEDs from two switches using the assembly language program.
5) Write an LED blink program using Timer2 with a delay of 1 second (the computed value must be loaded onto the OCR2B register) using its Interrupt.
6) Include the **Proteus simulation** of the LED blink program and LED control system using push buttons. Explain the simulation methodology. You may learn the simulation from the following video link: *https://www.youtube.com/watch?v=yHB5it0s2oU*

Reference(s):

[1] https://www.arduino.cc/.
[2] ATMega328 manual
[3] https://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-timers
[4] http://maxembedded.com/2011/06/avr-timers-timer0/