Experiment Title: Familiarizing of assembly language program in a microcontroller

Obsessive:

The obsessives of this experiment are to:

1) Study the assembly language program of Arduino.
2) Write assembly language programming code for Arduino.
3) Build a circuit to turn on an LED on an Arduino Microcontroller Board an input signal detected by an input switch connected to an I/O port of the microcontroller.
4) Know the working dragon of a project run by an Arduino's built-in Timer.
5) Implement a squeal LED light pattern control system using an input switch and an Arduino Microcontroller Board.

Theory and Methodology:

To study assembly language programming using the Arduino IDE, an Arduino Microcontroller Board with a connecting cable, LEDs, switches, resistors, assembly language programming knowledge, a compiler, a loader, and related components were required. Assembly language was composed within the Arduino IDE and uploaded into the microcontroller via the USB cable after the successful compilation of the code. For this purpose, it was necessary to ensure that two different files were written, one with a. into extension and the other with a. S extension, both sharing the same file name. Both files had to be placed in the same directory.

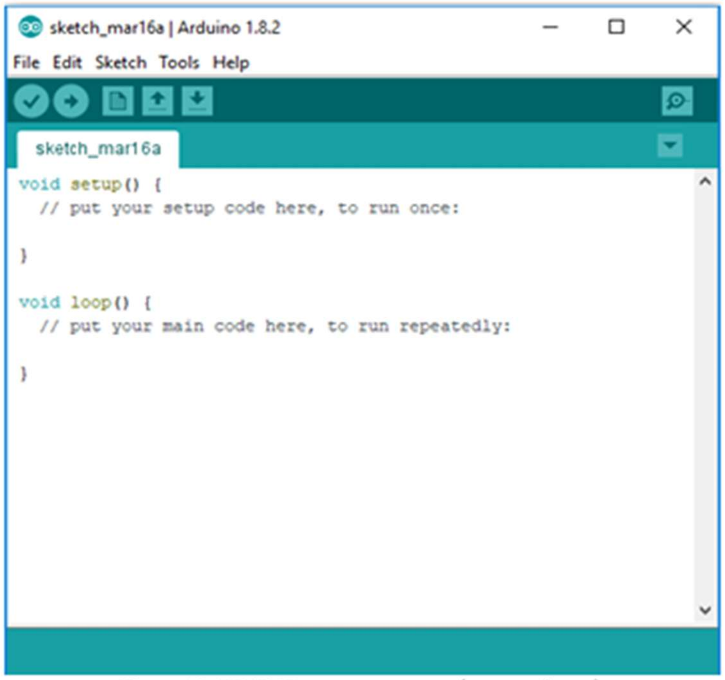
It was noted that the Arduino featured 32 general-purpose registers, with their physical addresses situated on the left side. The Arduino included 4 I/O ports for programming, specifically named ports A, B, C, and D, as displayed in Figure 4. Each of these ports was 8 bits wide and could be configured as either an input or an output port. For each port, there existed a special purpose register known as the data direction register, which defined the direction of signal flow to or from the Arduino microcontroller. These ports were also adaptable for various other functions beyond standard input and output operations.

Equipment:

1. Arduino IDE (version 2.0.1 or any recent iteration).
2. An Arduino Microcontroller board.
3. A PC equipped with an Intel processor.
4. A single red LED light.
5. A 100 Ω resistor.
6. A push switches.
7. Jumper wires for connecting components.

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:
2. To do Assembly Experiments based on programs of Parts 1 and 2: Write the program by



```
void setup() {
  // put your setup code here, to run once:

}

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

}
```

Fig. 10 IDE Environment (text editor)

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.
   f. Create pbintLED.ino file using the codes given below.
   g. Compile this file and upload the .hex file to the Arduino board.
   h. Modify the program to turn on/off an LED at digital PIN 12 with a different delay.
   i. Next, create blinkLED.ino file using the codes given below.
   j. Compile this file and upload the .hex file to the Arduino board.
   k. Modify the program to blink an LED at digital PIN 12 with a different Timer and delay value, say 2 s.

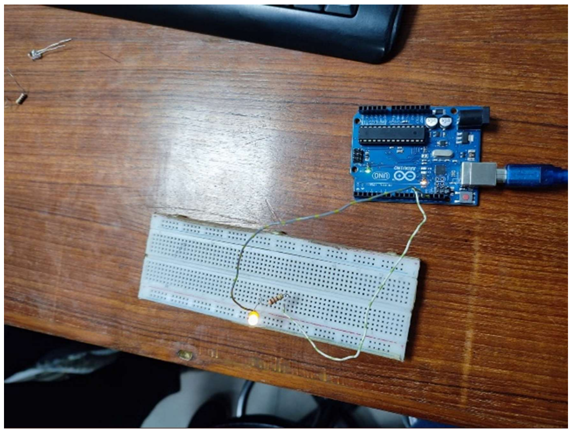
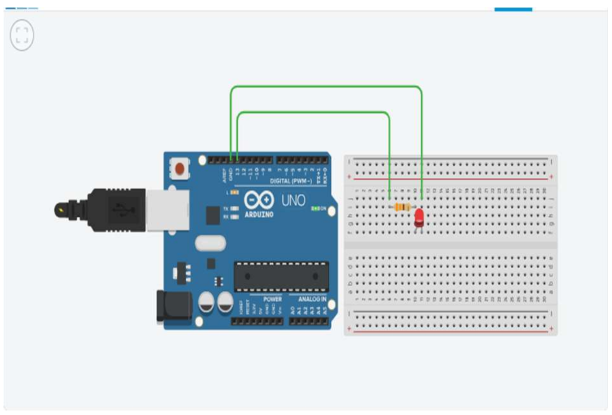
Code of an LED light blink and LED control using a switch:

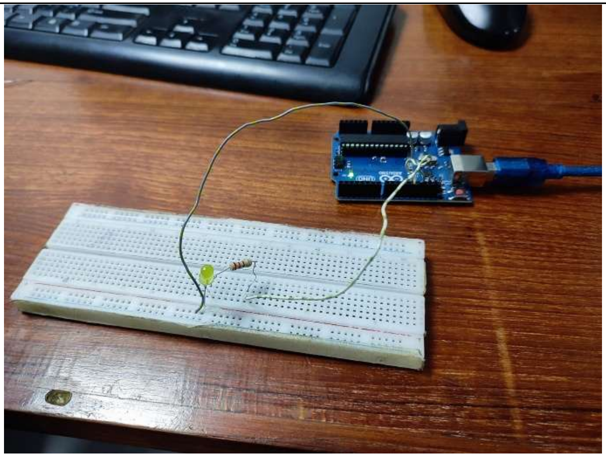
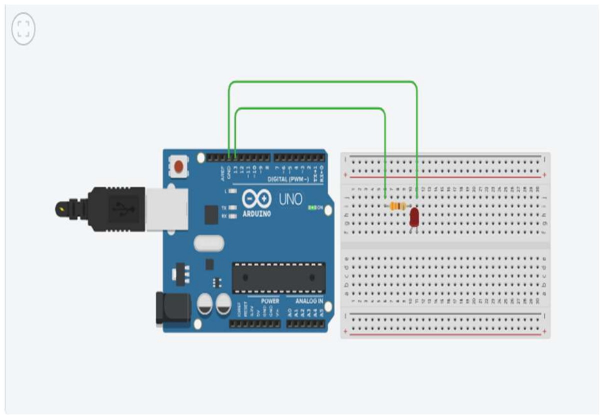
PART 1: Blink an LED

Assembly Code

```
#define __SFR_OFFSET 0x00
#include <avr/io.h>
.global start
.global led
; Definitions
.equ LED_PIN, 5
.equ DELAY_VAL, 10000
; Function Definitions
start:
   SBI DDRB, LED_PIN    ; Set PB5 (D13) as output
   RET                  ; Return to setup() function
led:
   CP R24, R1           ; Compare the parameter with 0
   BREQ ledOFF          ; Jump if equal to 0
   SBI PORTB, LED_PIN   ; Turn on the LED
```

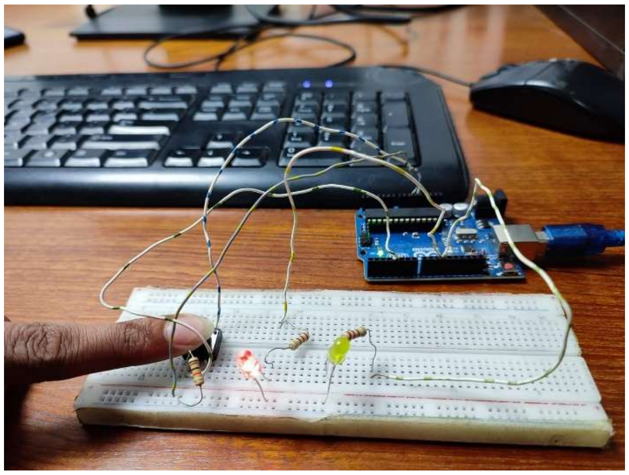
```
    RCALL myDelay        ; Delay for LED ON duration
    RET                  ; Return to loop() function
ledOFF:
    CBI PORTB, LED_PIN   ; Turn off the LED
    RCALL myDelay        ; Delay for LED OFF duration
    RET                  ; Return to loop() function
myDelay:
    LDI R20, 100         ; Outer loop count
outerLoop:
    LDI R30, lo8(DELAY_VAL) ; Initialize delay counter low byte
    LDI R31, hi8(DELAY_VAL) ; Initialize delay counter high byte
innerLoop:
    SBIW R30, 1          ; Decrement delay counter
    BRNE innerLoop       ; Repeat inner loop if not zero
    DEC R20              ; Decrement outer loop counter
    BRNE outerLoop       ; Repeat outer loop if not zero
    RET                  ; Return from delay function
```

Hardware	Simulation
 Led ON	 Led ON

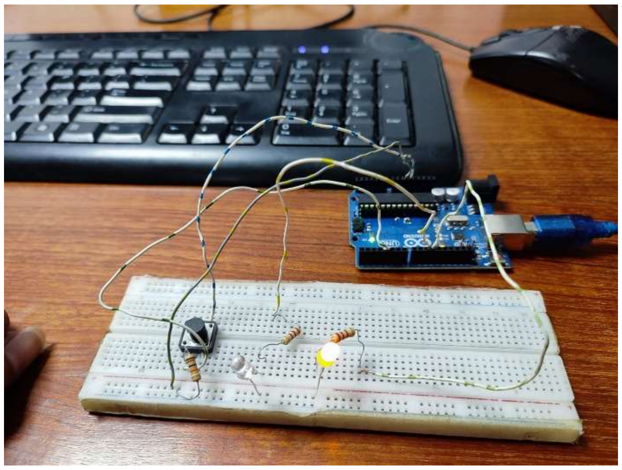
 Led OFF	 Led OFF

PART 2: Push button LED control

```
;------------------------------------------
; Assembly Code: RGB LED ON/OFF via Buttons
;------------------------------------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
.global start
.global btnLED
; Function Definitions
start:
    SBI DDRB, 4     ; Set PB4 (pin D12) as output - red LED
    SBI DDRB, 3     ; Set PB3 (pin D11) as output - green LED
    RET
btnLED:
L2:
    SBIC PIND, 2    ; Check if the red button is pressed
    RJMP L3         ; Jump if not pressed
    SBI PORTB, 4    ; Turn ON red LED (PB4)
    RJMP L4
L3:
   CBI PORTB, 4    ; Turn OFF red LED (PB4)
L4:
    SBIC PIND, 3    ; Check if the green button is pressed
    RJMP L5         ; Jump if not pressed
    SBI PORTB, 3    ; Turn ON green LED (PB3)
    RJMP L6
L5:
    CBI PORTB, 3    ; Turn OFF green LED (PB3)
L6:
    SBIC PIND, 4    ; Check if the blue button is pressed
    RJMP L7         ; Jump if not pressed
    SBI PORTB, 2    ; Turn ON blue LED (PB2)
    RJMP L8
L7:
    CBI PORTB, 2    ; Turn OFF blue LED (PB2)
L8:
    RET
```



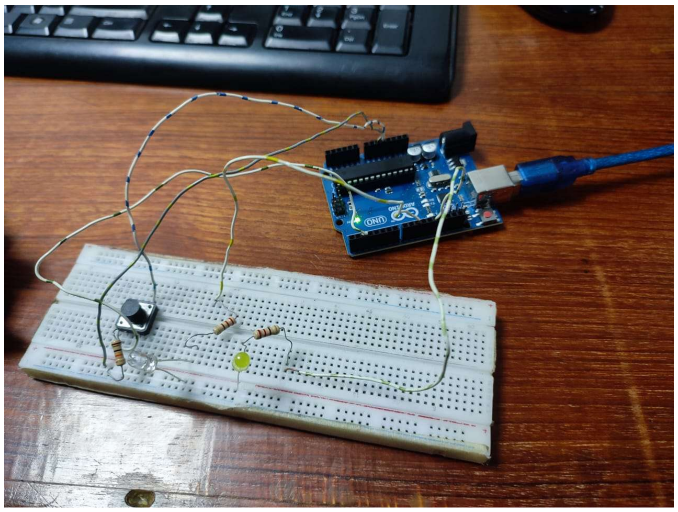
Rad Led ON

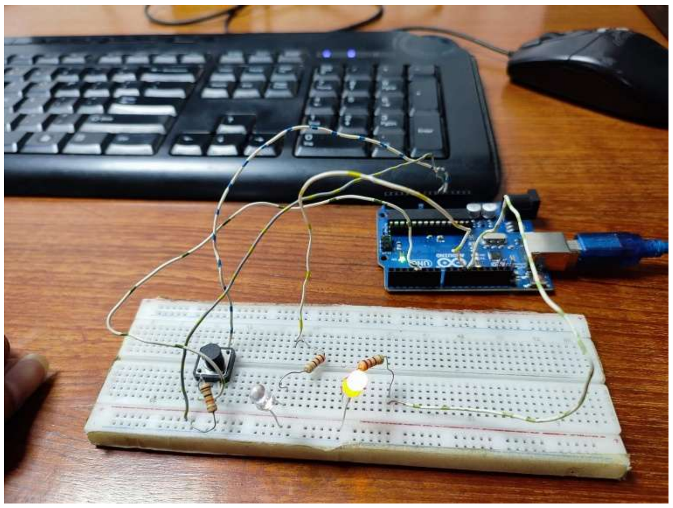


Green Led ON

PART 3: Push button LED control

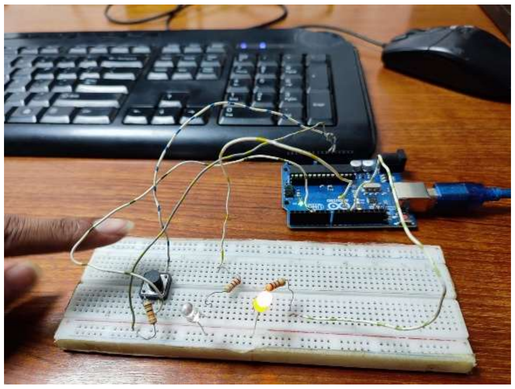
```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP); // Internal pull-up resistor enabled
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);}
void loop() {
  // Nothing to do in the loop; the LED state is updated by the interrupt}
void blink() {
  // Debounce the button
  static unsigned long lastDebounceTime = 0;
  unsigned long debounceDelay = 50; // Adjust this value as needed
  if (millis() - lastDebounceTime > debounceDelay) {
    // Record the time when the button state changes
    lastDebounceTime = millis();
    // Toggle the LED state
    state = !state;
    digitalWrite(ledPin, state);  }}
```

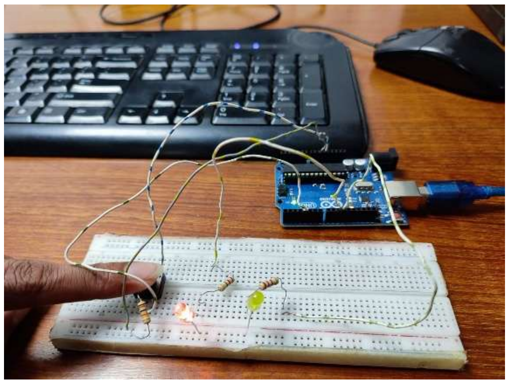

led Off


led On

PART 4: Blinking an LED using a Timer1 ISR

```
bool LED_State = true; // Corrected 'True' to 'true'
void setup() {
  pinMode(13, OUTPUT);
  cli(); // Disable interrupts
  // Reset Timer1 registers
  TCCR1A = 0;
  TCCR1B = 0;
  // Set prescaler value of 256
  TCCR1B |= (1 << CS12); // Set CS12 bit of TCCR1B
  // Enable compare match mode of register A
  TIMSK1 |= (1 << OCIE1A); // Set OCIE1A bit
  // Set the required timer count value in the compare register A
  OCR1A = 31250; // Assuming 16MHz clock, this will trigger every 500 ms
  sei(); // Enable interrupts}
void loop() {
  // No need for code in loop(), everything is handled in the ISR}
// ISR for Timer1 compare match A
ISR(TIMER1_COMPA_vect) {
  TCNT1 = 0; // Reset timer count to 0 for the next interrupt
  LED_State = !LED_State; // Toggle LED state
  digitalWrite(13, LED_State); // Write the new state to pin 13}
```


Yellow Led ON


Red Led On

Discussion:

In this experiment, the utilization of an Arduino Uno Microcontroller board to control an LED was explored. The setup replicated the hardware configuration previously implemented, which included connecting the anode of the LED to PIN 13 of the microcontroller and introducing a 100-ohm resistor in series with the LED's cathode. This resistor was then linked to the GND (Ground) pin on the microcontroller. The process of creating and verifying the code in the Arduino IDE was executed successfully, leading to the generation of a HEX file. This file contained the instructions necessary for the simulation. Subsequently, the Proteus simulation environment was employed to execute the instructions encapsulated in the HEX file, mimicking the operation of the Arduino Uno and the LED. The simulation was completed, and the results obtained were methodically recorded and compared with the outcomes of the hardware implementation. The experiment highlighted the practicality and utility of simulations in testing and verifying hardware related functionalities. It provided valuable insights into the consistency of results between real-world hardware and virtual simulations, enhancing our understanding of microcontroller-based systems.

References:

1) https://www.arduino.cc/
2) ATMega328 manual
3) https://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-mers
4) http://maxembedded.com/2011/06/avr-mers-mer0/