AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Engineering

Lab Report

Experiment # 5

Experiment Title: Familiarization of assembly language program in a microcontroller.

Date of Perform:	Date Month 2023	**Date of Submission:**	26 June 2023
Course Title:	MICROPROCESSOR AND EMBEDDED SYSTEM LAB		
Course Code:	COE3104	**Section:**	N/A
Semester:	Summer 2022-23	**Degree Program:**	BSc in CSE/EEE
Course Teacher:	**Prof. Dr. Engr. Muhibul Haque Bhuyan**		

Group #

Sl No	Name	ID	PROGRAM	SIGNATURE

Faculty use only

FACULTY COMMENTS	**Marks Obtained**	
	Total Marks	

Table of Contents

Experiment Title: Familiarization of assembly language program in a microcontroller.

Objectives:
The objectives of this experiment are to-

- Familiarize with the assembly language program of Arduino.
- Implement assembly language programming code for Arduino.
- Implement a circuit that activates an LED on an Arduino Microcontroller Board upon detecting an input signal from a connected input switch through an I/O port of the microcontroller.
- Familiarize with the operational timeframe of a project powered by the Arduino's built-in Timer.
- Implement a system for controlling a sequential LED light pattern using an input switch and an Arduino Microcontroller Board.

Equipment List:

1) Arduino Microcontroller board
2) PC having an Intel processor
3) LED lights (Red - 1 pcs, Blue – 1 pcs, Green – 1 pcs)
4) Three 100 Ω resistor
5) Three push switches
6) Jumper wires
7) Breadboard

Circuit Diagram:

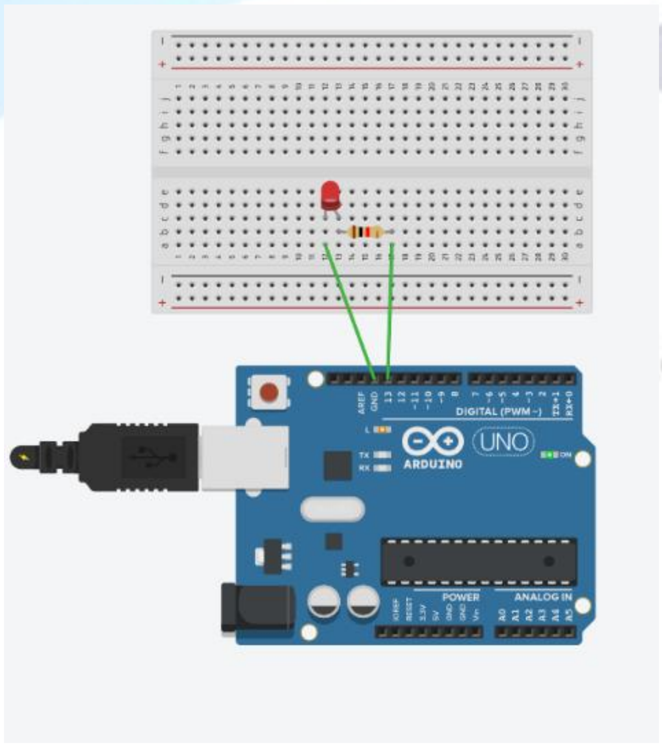

Fig. 1 Experimental setup of an LED blink system using an Arduino Microcontroller Board

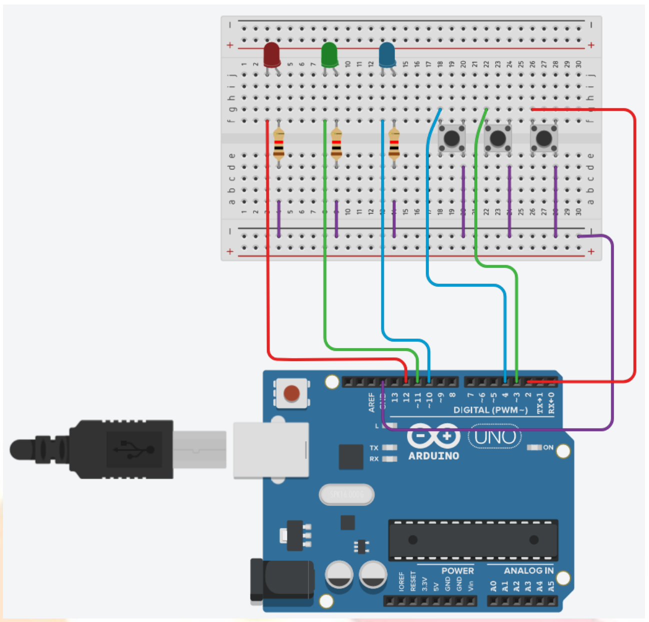
Fig. 2 Experimental setup of an RGB LED ON/OFF using push button on an Arduino Microcontroller Board

Code/Program:

PART 1: Blink an LED [for Fig. 1]

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
RCALL myDelay ; Calling a delay function
RET ; return to loop() function
;---------------------------------------------------------------------
-------
ledOFF:
CBI PORTB, 5 ; set D13 to low
RCALL myDelay
RET ; return to loop() function
;---------------------------------------------------------------------
-------
.equ delayVal, 10000 ; initial count value for the inner loop
;---------------------------------------------------------------------
-------
myDelay:
LDI R20, 100 ; initial count value for the outer loop
outerLoop:
LDI R30, lo8(delayVal) ; low byte of delayVal in R30
LDI R31, hi8(delayVal) ; high byte of delayVal in R31
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 not equal to 0
RET
;---------------------------------------------------------------------
-------
```

PART 2: Push button LED control [for Fig. 2]

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
;==========================================================
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 is not
pressed
RJMP L1
SBI, PORTB, 2 ; Turn ON LED, PB2 if not pressed
CBI, PORTB, 3 ; Turn OFF LED, PB3 if not pressed
```

```
SBIC, PIND, 4 ; Skips below statement if the push button is pressed
RJMP L2
L1: SBI, PORTB, 3 ; Turn ON LED, PB3 if pressed
CBI, PORTB, 2 ; Turn OFF LED, PB2 if pressed
RET
```

Hardware Output Results:

Here is the hardware implementation of LED blink test and the necessary explanation of the implementation:

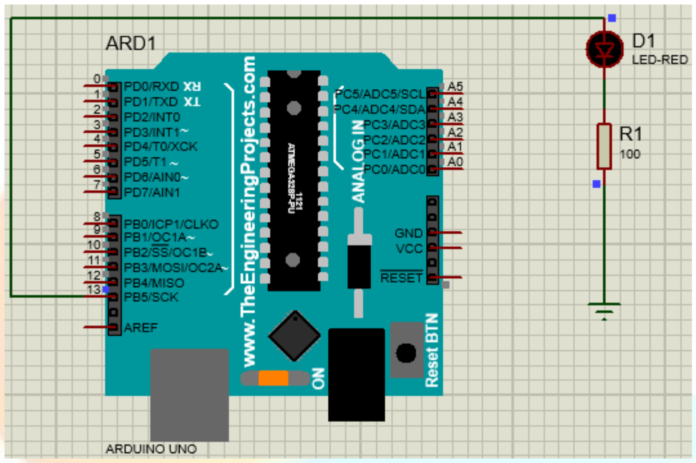

Fig. 3 Hardware implementation setup of LED BLINK TEST

Explanation: In this experiment, an Arduino Uno Microcontroller board was taken. The anode of an LED was connected PIN 13 of the microcontroller. A 100 Ω resistor was connected with the cathode of the LED. The resistor was connected with the GND pin of the microcontroller. Necessary codes were pushed using a serial port via a computer by the Arduino IDE after compilation.

Here is the hardware implementation of RGB LED ON/OFF via buttons and the necessary explanation of the implementation:

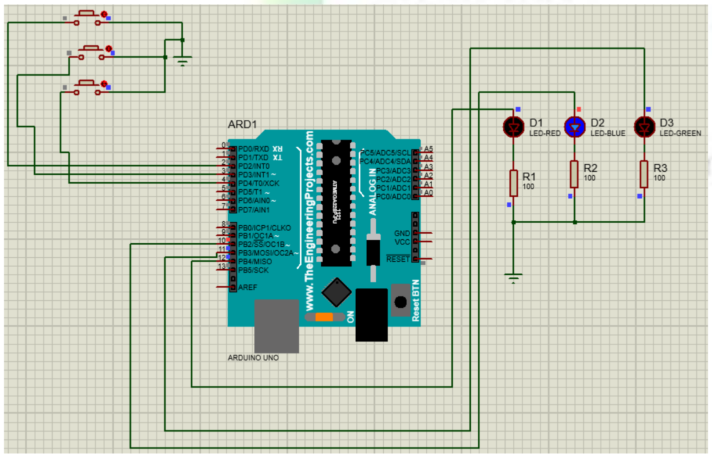

Fig. 4 Hardware implementation setup of RGB LED ON/OFF

Explanation: In this experiment, an Arduino Uno Microcontroller board was taken. The anode of three LEDs – RED, GREEN and BLUE were connected at the following pins accordingly - PIN 10, 11 and 12. Three 100 Ω resistors were connected with the cathode of the LEDs. The resistors were connected with the GND pin of the microcontroller. Three push buttons were used to change the lights. The anodes of the GREEN, BLUE and RED LED BUTTONS were connected in the following serial – PIN 02, 04 and 03. The cathode pole of the push buttons were connected with the GND pin of the Arduino board. Necessary codes were pushed using a serial port via a computer by the Arduino IDE after compilation.

Simulation Output Results:
Here is the simulation implementation of LED blink test and the necessary explanation of the implementation:

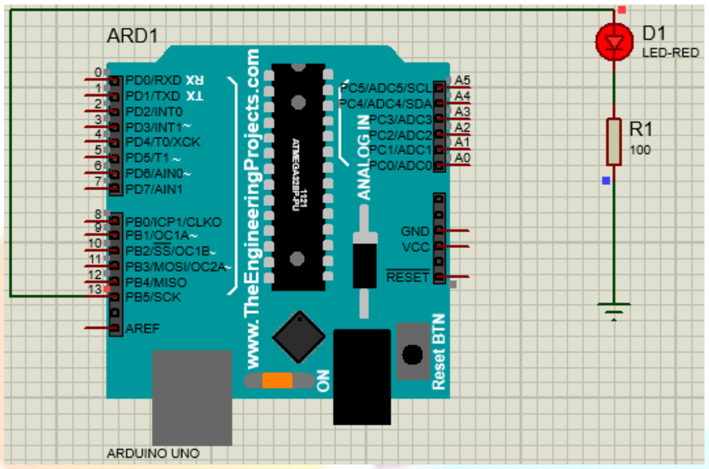

Fig. 5 Simulation of LED ON in LED BLINK TEST

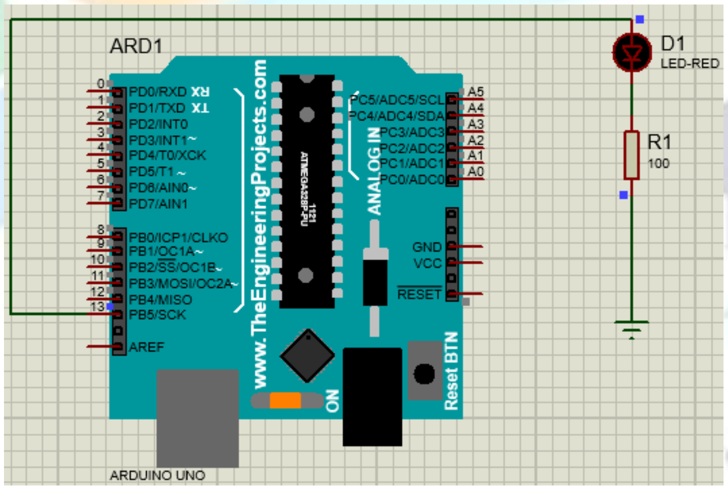

Fig. 6 Simulation of LED OFF in LED BLINK TEST

Explanation: In this simulation, an Arduino Uno Board, Resistor and LED was configured according to the hardware implementation that was performed. The program was made in the Arduino IDE was verified. As a result, a HEX file was generated. The following HEX file was implemented in the Proteus simulation for operation related instructions of the simulation. Afterwards, the simulation was performed and the results that were obtained were noted and compared with the hardware results.

Here is the simulation implementation of RGB LED ON/OFF via buttons and the necessary explanation of the implementation:

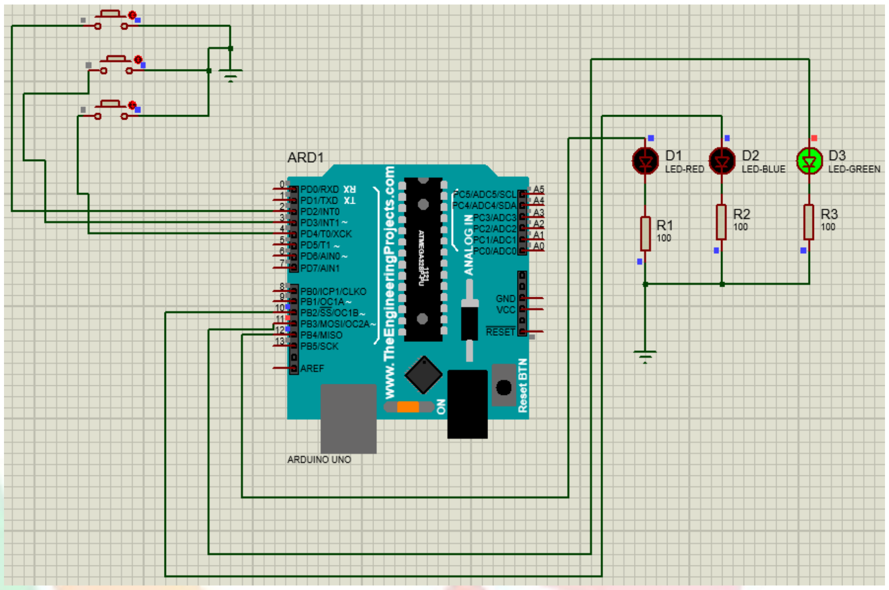

Fig. 7 Simulation of GREEN LED ON in RGB LED ON/OFF Test by pressing Green button

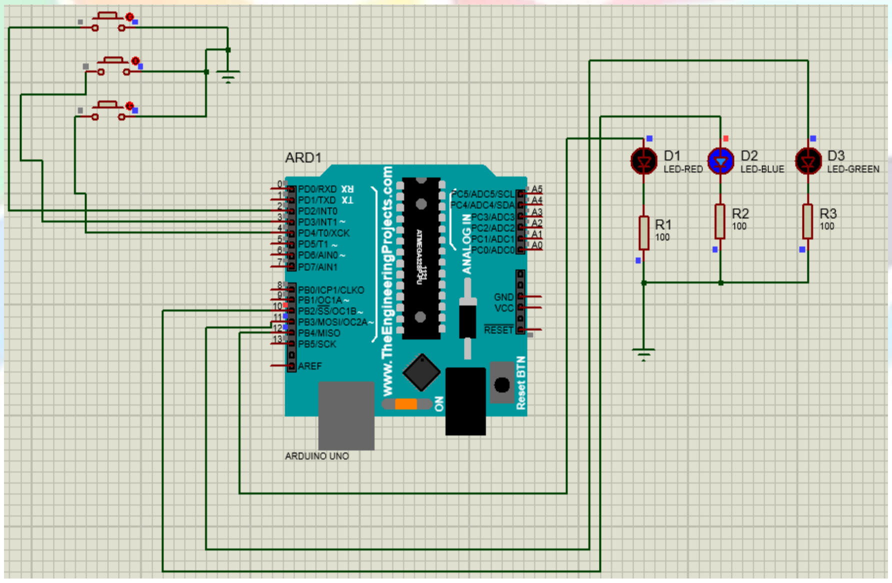

Fig. 8 Simulation of BLUE LED ON in RGB LED ON/OFF Test by pressing Blue button

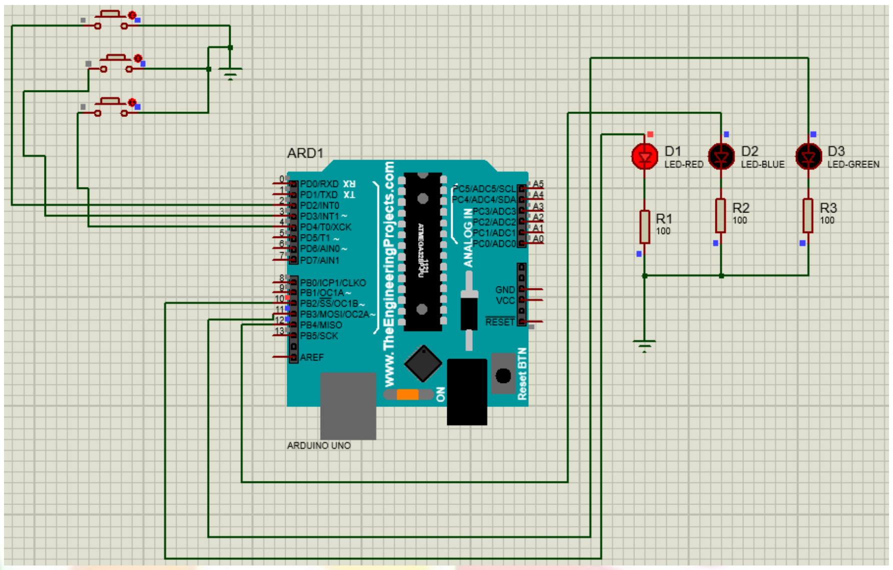

Fig. 9 Simulation of RED LED ON in RGB LED ON/OFF Test by pressing Red button

Explanation: In this simulation, an Arduino Uno Board, Resistor, RED, GREEN and YELLOW LEDs were configured according to the instructions provided in the commented code that was used to perform the task. Buttons to turn on and off the LEDs were also added in the simulation and were connected accordingly based on the assembly code comments. The program was made in the Arduino IDE was verified. As a result, a HEX file was generated. The following HEX file was implemented in the Proteus simulation for operation related instructions of the simulation. Afterwards, the simulation was performed and the results that were obtained were noted and compared with the hardware results.

Discussion:

The purpose of this micro lab experiment was to familiarize ourselves with assembly language programming in a microcontroller. Throughout the experiment, practical experience and insights into the intricacies of programming at the low-level were successfully gained. Assembly language programming in a microcontroller was familiarized during the experiment. The architecture and instruction set of the microcontroller were familiarized to understand the available operations and resources that could be utilized in the programs. Simple assembly language programs were written to perform basic operations, such as arithmetic calculations, conditional branching, and loop structures. The fundamental concepts of assembly programming, including syntax, register usage, and memory management were grasped. Challenges inherent in assembly language programming were encountered, including the need for meticulous attention to detail, manual memory management, and limited high-level abstractions. Assembly language required explicit management of system resources, such as registers and memory, which demanded careful planning and optimization. One of the major advantages discovered through this experiment was the ability to write highly efficient and compact code. Assembly language programs could be optimized to achieve faster execution times and reduced memory usage compared to their high-level language counterparts. This advantage became particularly significant in resource-constrained systems where efficiency was critical. Furthermore, a deeper understanding of the microcontroller's internals was enabled through the familiarity gained with assembly language programming. Insights into the interactions between the hardware components and the software were obtained by working at such a low level. This knowledge proved to be invaluable for debugging and optimizing system performance. While assembly language programming has its merits, it also has its limitations. The intricate nature of assembly code makes it prone to errors, and debugging can be a challenging process. Additionally, the lack of portability restricts the reuse of code across different microcontrollers or processors, necessitating significant modifications when migrating to a different platform. In conclusion, the micro lab experiment titled "Familiarization of assembly language program in a microcontroller" provided us with a valuable opportunity to explore and comprehend the intricacies of assembly language programming. Practical experience was gained in writing efficient and optimized code, as well as a deeper understanding of the microcontroller's inner workings. Despite its challenges and limitations, assembly language programming remains an essential skill for embedded systems development and offers unparalleled control and performance optimizations.

References:

1) Circuit Digest Website, [Online: March 21, 2022], [Cited: June 10, 2023] Available: https://circuitdigest.com/article/everything-you-need-to-know-about-arduino-uno-board-hardware

2) MicroPi, *Program the Arduino Uno in Assembly Language* [Online: March 23, 2021], [Cited: June 24, 2023] Available: https://micropi.wordpress.com/2021/03/23/program-the-arduino-uno-in-assembly-language/

3) Arduino CC Website, [Online] [Cited: June 10, 2023] Available: https://docs.arduino.cc/hardware/uno-rev3

4) AIUB Microprocessor and Embedded Systems Lab Manual 5