EE2016 : Microprocessor Theory and Lab

Lab Report # 3

Hardware wiring and programming for interrupts by ASM and C-Programming using Atmel Atmega(8) AVR

Mandla Siva Manoj, EE21B083
Ajoe George, EE21B009
Group 3

October 8, 2022

Contents

List of Figures

1 Aim of the Experiment

To implement basic arithmetic and logical manipulation programs using Atmel Atmega8 microcontroller in assembly program emulation, including addition, multiplication and comparison.

This experiment introduces assembly programming and interaction with peripherals in Atmel Atmega8 microcontroller.

- This experiment introduces assembly programming and interaction with peripherals in Atmel Atmega8 microcontrollers.
- This experiment introduces assembly programming and interaction with peripherals in Atmel Atmega8 microcontroller.
- This experiment introduces assembly programming and interaction with peripherals in Atmel Atmega8 microcontroller.

2 Problems

1. Make a blinking LED program.
2. Make a program to control a LED using a push button
3. Problem 2 4 bit addition of two unsigned nibbles from the 8 bit dip input switch and display the result obtained in LEDs.
4. Problem 2 4 bit addition of two unsigned nibbles from the 8 bit dip input switch and display the result obtained in LEDs.

3 Blinking LED Experiment

3.1 Code

```
.CSEG
 LDI R16, 0x01
 OUT DDRB, R16

 again: LDI R16, 0x01
OUT PORTB, R16

LDI R16, 0xFF
loop1:  LDI R17, 0xFF
loop2:  DEC R17
BRNE loop2
DEC R16
BRNE loop1

LDI R16, 0x00
OUT PORTB, R16

LDI R16, 0xFF
back3:  LDI R17, 0xFF
back4:  DEC R17
BRNE back4
DEC R16
BRNE back3
rjmp again
```

Listing 1: Code for Blinking LED

3.2 Outputs

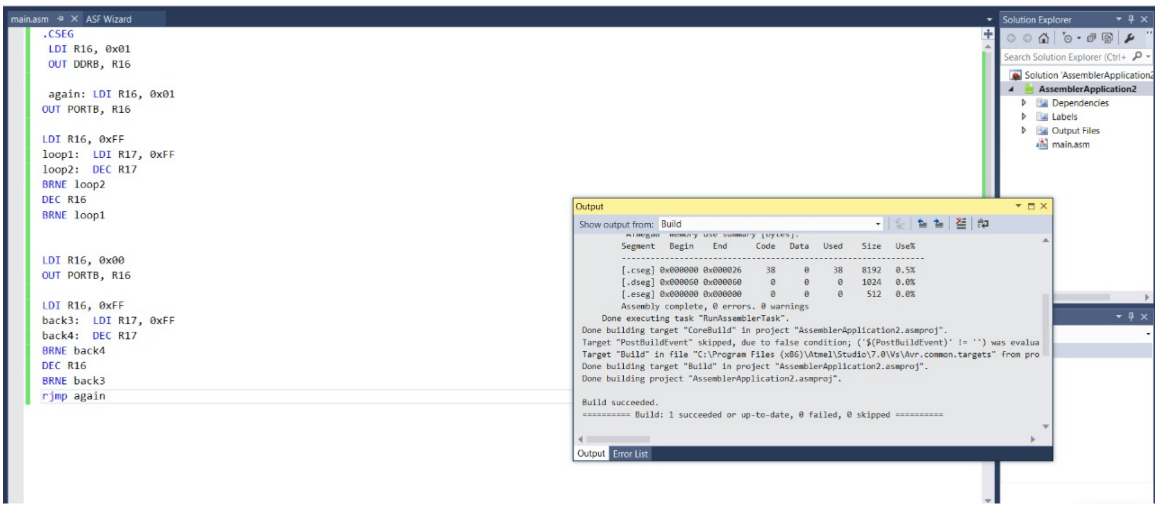
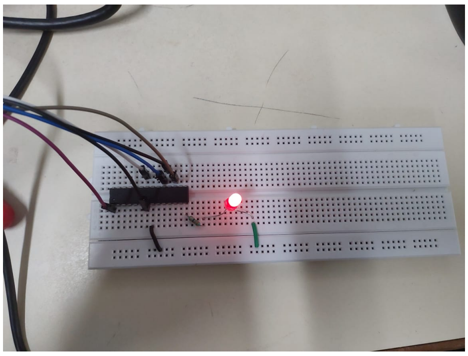
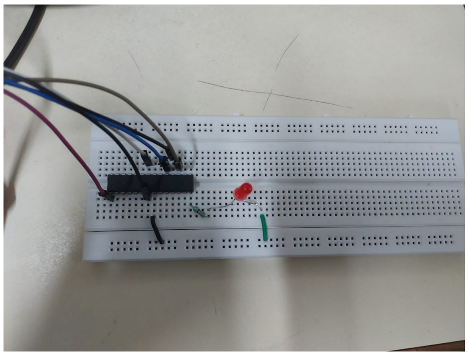


Figure 1: Output of Blinking LED



(a) Blink ON



(b) Blink OFF

Blinking Video:
https://drive.google.com/file/d/1BeScRXJdqsWOVwoSAh4Zb8gVUhhKLgdL/view?usp=drivesdk

3.3 Explanation

The LED will begin to glow if P0 is grounded because current will flow from it. So, Port should be set to LOW. Now, since we don't want current to pass through LED, P1 shouldn't be grounded if we want to turn off the LED. Port P1.0 should therefore be HIGH. The programme continuously commands the IC to carry out the aforementioned two actions, which causes the LED to blink.

4 LED with push button

4.1 Code

```
.CSEG
LDI R16, 0x01
OUT DDRB, R16
LDI R16, 0x00
OUT DDRD, R16
again: LDI R16, 0x00
       OUT PORTB, R16
       IN R16,PIND
       COM R16
       ANDI R16, 0x01
       OUT PORTB, R16
       rjmp again
```

Listing 2: Code for LED with push button

4.2 Outputs

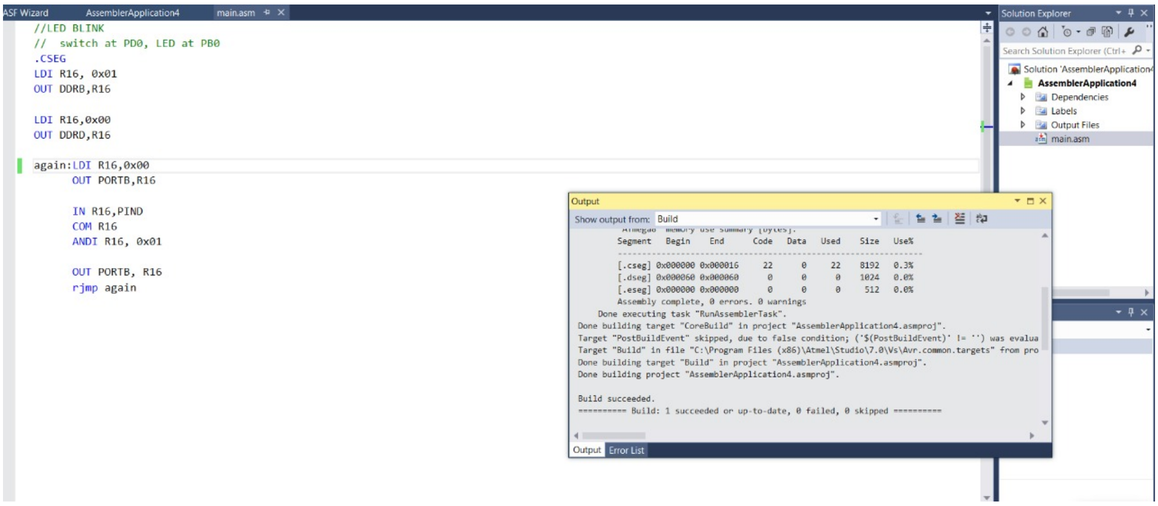
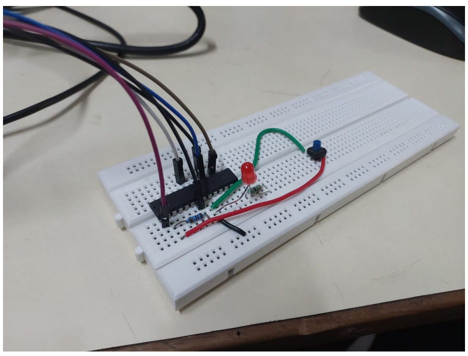


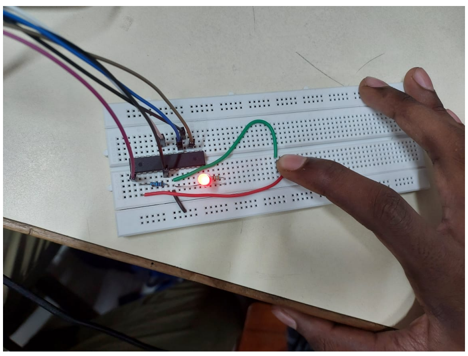
Figure 3: Output of LED with push button

4.3 Explanation

The hexadecimal number is a four bit number in binary (for example, F=1111,0=0000,1=0001). The last two digits of the number 01 are in hexadecimal format, and the 0x in 0x01 indicates that the number is in that format. This 8-bit binary number is assigned to the port 1 bits because it is equivalent to the binary value of 01, which is 00000001. As a result, port 1's pin 0 functions as an input pin and the remaining 7 pins as an output. We can use the button's connection to port 1 pin 0, which is designated as input by the statement P1=0x01, as input for our programme. The logic declared in the code is continuously run using a continuous loop. According to the logic, whenever the button is high, i.e. the button is pressed , the led glows and is switched off when released.



(a) Switch Open



(b) Switch Close

5 Addition and results displayed in LEDs

5.1 Code

```
#include "m8def.inc"

START:
        LDI R16, 0x00;
        OUT DDRD, R16;  Setting PORTD to INPUT

        LDI R16, 0xFF;
        OUT DDRC, R16;  Setting PORTC to OUTPUT

ADDITION:
        IN R21, PIND;   R21 <-- (<NUM2><NUM1>)
        MOV R20, R21;   Making a copy of R21 in R20 for having the 2
        ↪  numbers in separate registers
        ANDI R20, 0xF0; Assigning R20 as "<NUM2>0000"
        SWAP R20;  Interchanging higher and lower nibbles of R20. R20 <--
        ↪  "0000<NUM2>"
        ANDI R21, 0x0F; Assigning R21 as "0000<NUM1>"
        ADD R20, R21;        R20 <-- R20 + R21

END:
    OUT PORTC, R20;        PORTC <-- R20
        NOP; End of program
```

Listing 3: Code for Addition using LEDs

5.2 Outputs

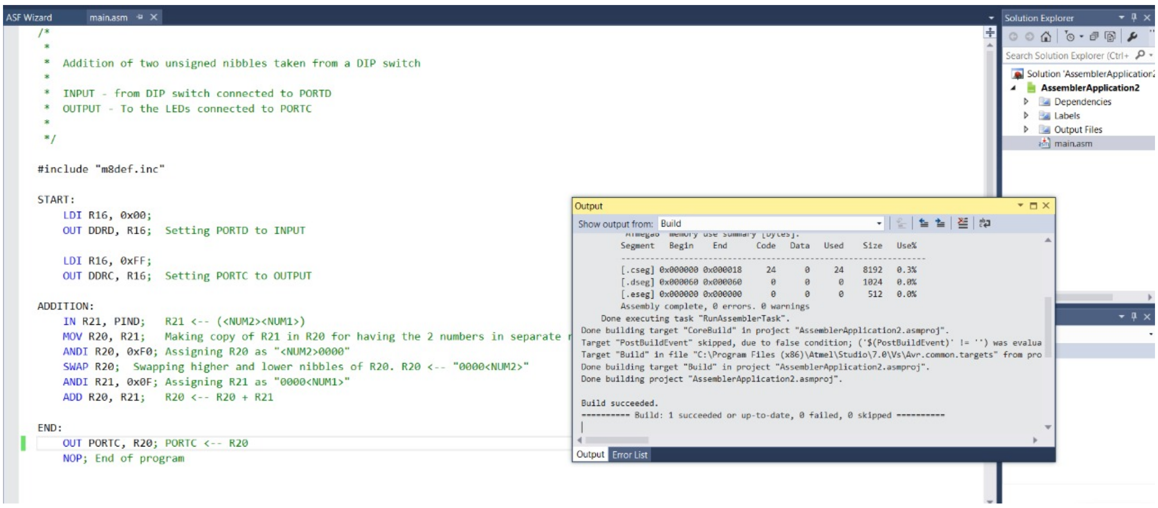
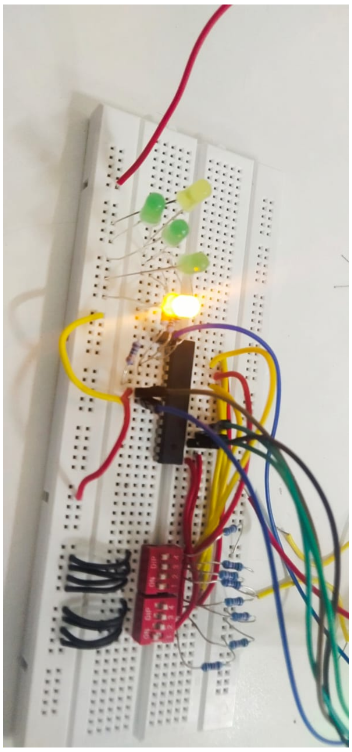
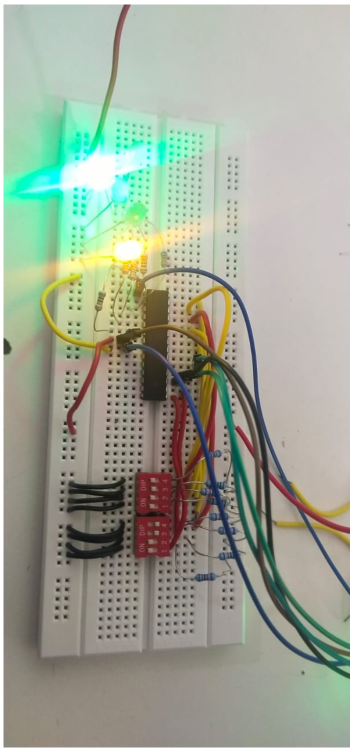


Figure 5: Output of Addition using LEDs



(a) Addition of 15 and 1



(b) Addition of 15 and 3

5.3 Explanation

At portD, which has 8 pins and 4 for one number and 4 for another, we are accepting external inputs.The desired output is obtained at register R20 and we get the practical result at portC, which is connected to LEDs.

6 Multiplication and results displayed in LEDs

6.1 Code

```
#include "m8def.inc"

START:
    LDI R16, 0x00;
    OUT DDRD, R16;  Setting PORTD to INPUT

    LDI R16, 0xFF;
    OUT DDRB, R16;  Setting PORTC to OUTPUT

        LDI R16, 0x00;        clearing productL register
        LDI R17, 0x00;        clearing productH register
        LDI R18, 0x00;        clearing temporary register

INPUT:
    IN R21, PIND;   R21 <-- (<NUM2><NUM1>)
    MOV R20, R21;   Making copy of R21 in R20 for having the 2 numbers in separate registers
    ANDI R20, 0xF0; Assigning R20 as "<NUM2>0000"
    SWAP R20;  Interchanging higher and lower nibbles of R20. R20 <-- "0000<NUM2>"
    ANDI R21, 0x0F; Assigning R21 as "0000<NUM1>"

MULTIPLY1:
        CLC;        clear Carry Bit
        ROR R21;        Right rotation of R21
        BRCC MULTIPLY2;    go to next step when last bit (carry now) is cleared.
        ADD R16, R20;
        ADC R17, R18;

MULTIPLY2:
        CLC;
        ROL R20;
        ROL R18;
        TST R21;
        BRNE MULTIPLY1;

END:
        OUT PORTB, R16;
        NOP;        End of Program
```

Listing 4: Code for Multiplication using LEDs

6.2 Outputs

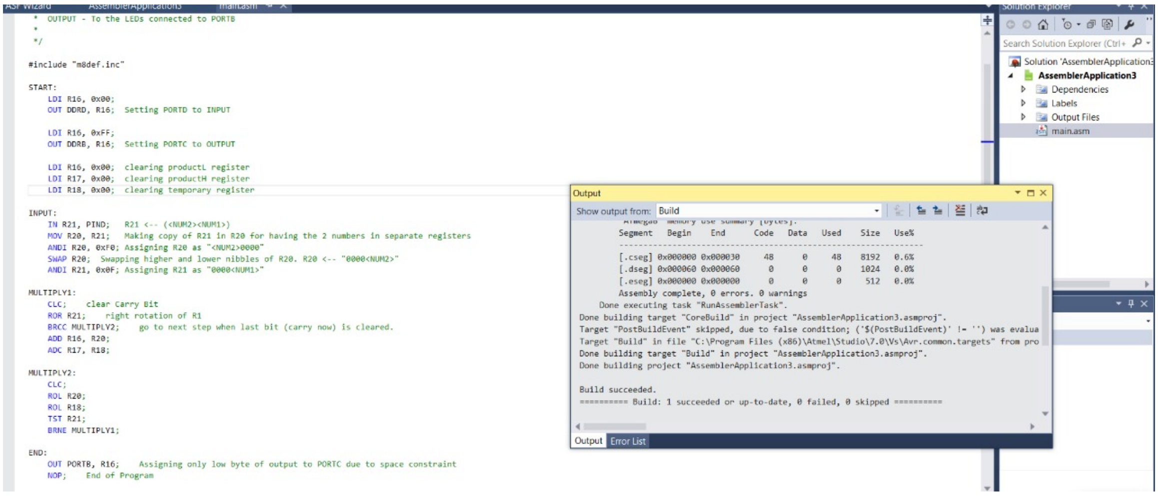


Figure 7: Output of Multiplication using LEDs

6.3 Explanation

At portD, which has 8 pins and 4 for one number and 4 for another, we are accepting external inputs.The desired output is obtained at register R0, copied to register R24, and the actual outcome is obtained at portC, which is connected to LEDs.

7 Learning Outcomes

By doing this experiment we were able to:

- Learn how to program in assembly language and how to burn code into board.
- Get familiar with the software Atmel Studio and AVR Burn-OMAT.
- Learn how to display outputs of addition and multipication in LEDs