

EE2016 : Lab Report

Experiment 8

Putta Shravya EE22B032
Maadhav Patel EE22b033

October 2023

1 Aim

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

1. Wire the microcontroller along with the given peripherals in a breadboard.
2. Program the microcontroller to read the DIP switch values and display it in an LED using assembly programming.
3. Program the microcontroller to perform the addition and multiplication of two four-bit numbers which are read from the DIP switches connected to a port and display the result using LED's connected to another port.

Google Drive link

[Click here to view all codes and videos of problems.](#)

2 Control an LED using a DIP switch

2.1 Code

```
.include "m8def.inc" ; Include the necessary AVR microcontroller definitions.

LDI R16, 0xFF        ; Load immediate value 0xFF (all bits set) into register R16.
OUT DDRD, R16        ; Set the data direction register (DDRD) for Port D to output.

LDI R16, 0x00        ; Load immediate value 0x00 (all bits cleared) into register R16.
OUT DDRB, R16        ; Set the data direction register (DDRB) for Port B to input.

LOOP:                ; Start of a loop.
    IN R16, PINB      ; Read the input state of Port B (PINB) into register R16.
    OUT PORTD, R16    ; Output the value in R16 to Port D (PORTD).
    RJMP LOOP         ; Relative jump (loop) back to the 'loop' label.

END: NOP              ; This is the end of the program. The 'NOP' instruction is a no-operation, essentially a
```

2.2 Inferences and Learning Outcomes

Learning Outcomes:

- Understanding of AVR assembly language syntax and directives.
- Knowledge of configuring data direction for I/O ports (setting them as inputs or outputs).
- Familiarity with loading immediate values into registers.
- Mastery of input and output operations for reading from and writing to I/O ports.
- Grasping the concept of loops in assembly language and creating infinite loops for continuous operation.

Inferences:

Register R16 is loaded with a value representing all bits set, which can be used to configure the data direction register (DDR) for Port D as output.

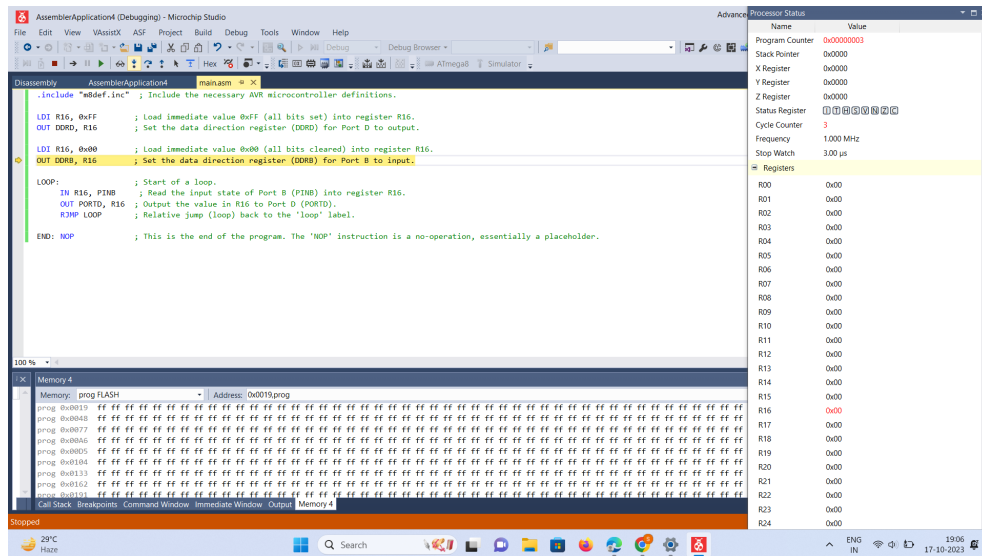
Port D is configured as an output port, meaning you can write data to it to control external devices connected to the pins of Port D.

Register R16 is then loaded with a value representing all bits cleared, which can be used to configure the DDR for Port B as input.

Port B is configured as an input port, meaning you can read data from it, typically used for reading the state of external devices or sensors connected to its pins.

Register R16 then holds the state of the pins on Port B.

The code is creating an infinite loop that repeatedly reads from Port B and mirrors the input to Port D



The screenshot shows the AVR Studio IDE with the following assembly code:

```
.include "mdef.inc" ; Include the necessary AVR microcontroller definitions.

LDI R16, 0xFF ; Load immediate value 0xFF (all bits set) into register R16.
OUT DDRE, R16 ; Set the data direction register (DDRE) for Port D to output.

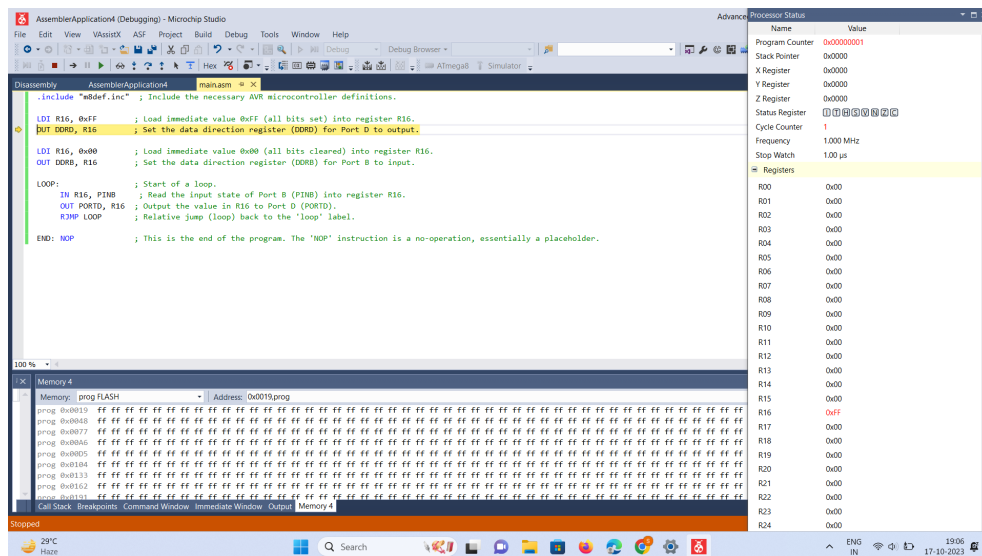
LDI R16, 0x00 ; Load immediate value 0x00 (all bits cleared) into register R16.
OUT DDRE, R16 ; Set the data direction register (DDRE) for Port B to input.

LOOP: ; Start of a loop.
    IN R16, PINB ; Read the input state of Port B (PINB) into register R16.
    OUT PORTD, R16 ; Output the value in R16 to Port D (PORTD).
    RJMP LOOP ; Relative jump (loop) back to the 'loop' label.

END: NOP ; This is the end of the program. The 'NOP' instruction is a no-operation, essentially a placeholder.
```

The Registers window on the right shows the state of the registers:

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0xFF
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00



The screenshot shows the AVR Studio IDE with the following assembly code:

```
.include "mdef.inc" ; Include the necessary AVR microcontroller definitions.

LDI R16, 0xFF ; Load immediate value 0xFF (all bits set) into register R16.
OUT DDRE, R16 ; Set the data direction register (DDRE) for Port D to output.

LDI R16, 0x00 ; Load immediate value 0x00 (all bits cleared) into register R16.
OUT DDRE, R16 ; Set the data direction register (DDRE) for Port B to input.

LOOP: ; Start of a loop.
    IN R16, PINB ; Read the input state of Port B (PINB) into register R16.
    OUT PORTD, R16 ; Output the value in R16 to Port D (PORTD).
    RJMP LOOP ; Relative jump (loop) back to the 'loop' label.

END: NOP ; This is the end of the program. The 'NOP' instruction is a no-operation, essentially a placeholder.
```

The Registers window on the right shows the state of the registers:

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0xFF
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00

Figure 1: Control an LED using a DIP switch

2.3 Flow Chart

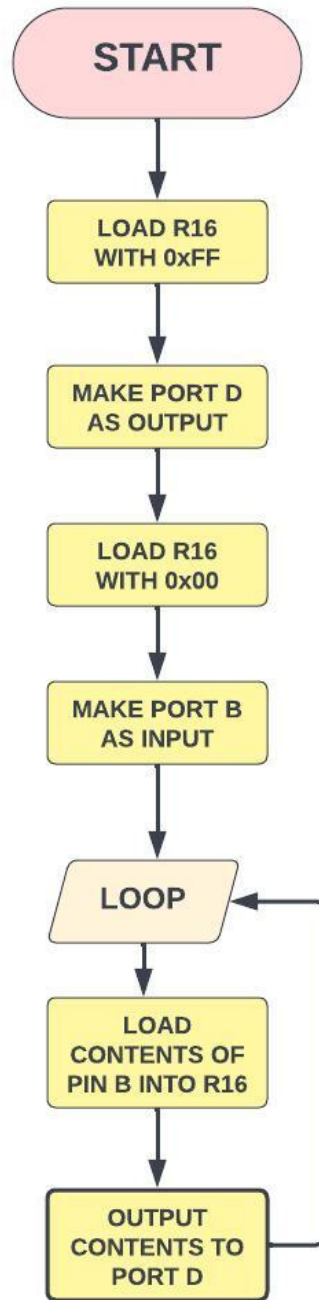


Figure 2: Control an LED using a DIP switch

3 4-bit addition of two unsigned nibbles from an 8-bit dip input switch

3.1 Code

```
.include "m8def.inc" ; Include ATmega8 specific definitions

LDI R16, 0xFF        ; Load immediate value 0xFF (all bits set) into register R16
OUT DDRD, R16        ; Set DDRD (Data Direction Register for Port D) to configure as output

LDI R16, 0x00        ; Load immediate value 0x00 into register R16
OUT DDRB, R16        ; Set DDRB (Data Direction Register for Port B) to configure as output
LDI R16, 0x00        ; Load immediate value 0x00 into register R16
OUT DDRC, R16        ; Set DDRC (Data Direction Register for Port C) to configure as output

LOOP:                ; Start of the loop
    IN R16, PINB      ; Read the input from PINB (Port B Input) into register R16
    IN R17, PINC      ; Read the input from PINC (Port C Input) into register R17
    ADD R16, R17      ; Add the values in R16 and R17 and store the result in R16
    OUT PORTD, R16    ; Output the result (R16) to PORTD (Port D Output)
    RJMP LOOP        ; Relative jump to the LOOP label (looping)

END: NOP             ; End of the program with a No Operation (NOP)
```

3.2 Inferences and Learning Outcomes

Learning Outcomes:

- Understanding AVR Assembly Language
- I/O Port Configuration
- Immediate Value Loading
- Input and Output Operations
- Looping and Control Flow
- Simple Arithmetic Operation
- Understanding I/O Concepts

Inferences:

Register R16 is loaded with a value representing all bits set. This value is likely used to configure a Data Direction Register (DDR) to configure a port as an output.

Port D is set as an output port, and you can write data to it to control external devices connected to the pins of Port D. Register R16 is then loaded with a value representing all bits cleared, which can be used to configure Data Direction Registers for Port B and Port C as input.

Port B is set as an input port, allowing you to read the state of external devices or sensors connected to its pins.

Register R16 is loaded with all bits cleared, preparing it for configuring DDRC as an input.

Port C is set as an input port, similar to Port B.

The program is designed to enter a loop, continuously performing the operations inside the loop.

The code reads the states of Port B and Port C, adds them together, and then outputs the result to Port D, which can be used to control external devices connected to the pins of Port D.

The code is creating an infinite loop, where it repeatedly reads from Port B and Port C, adds the values, and outputs the result to Port D.

3.3 Flow Chart

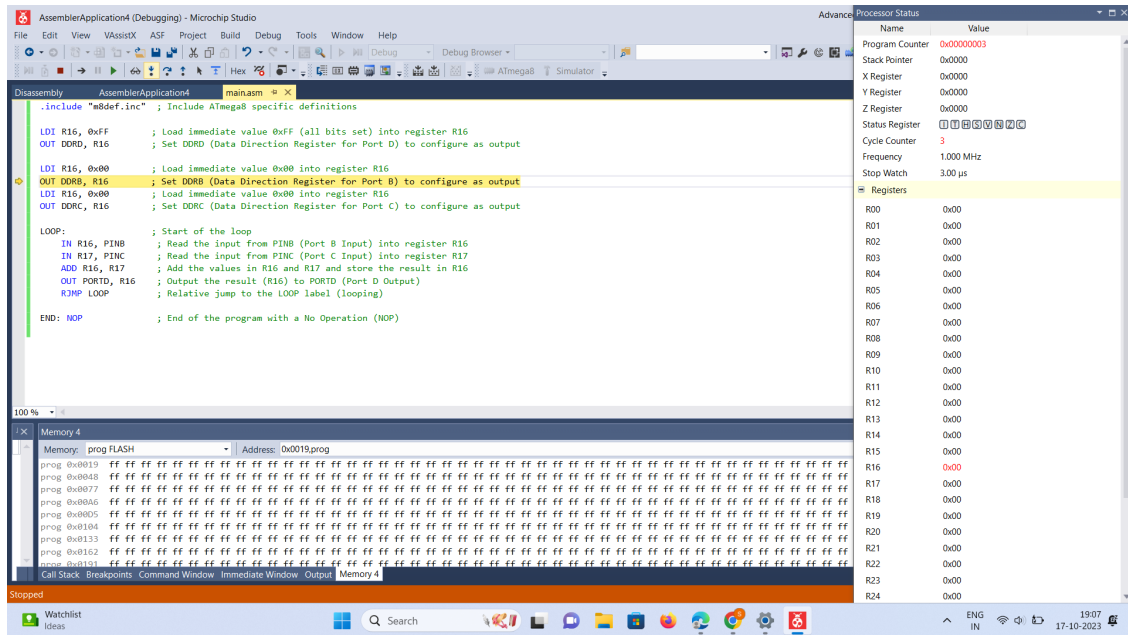
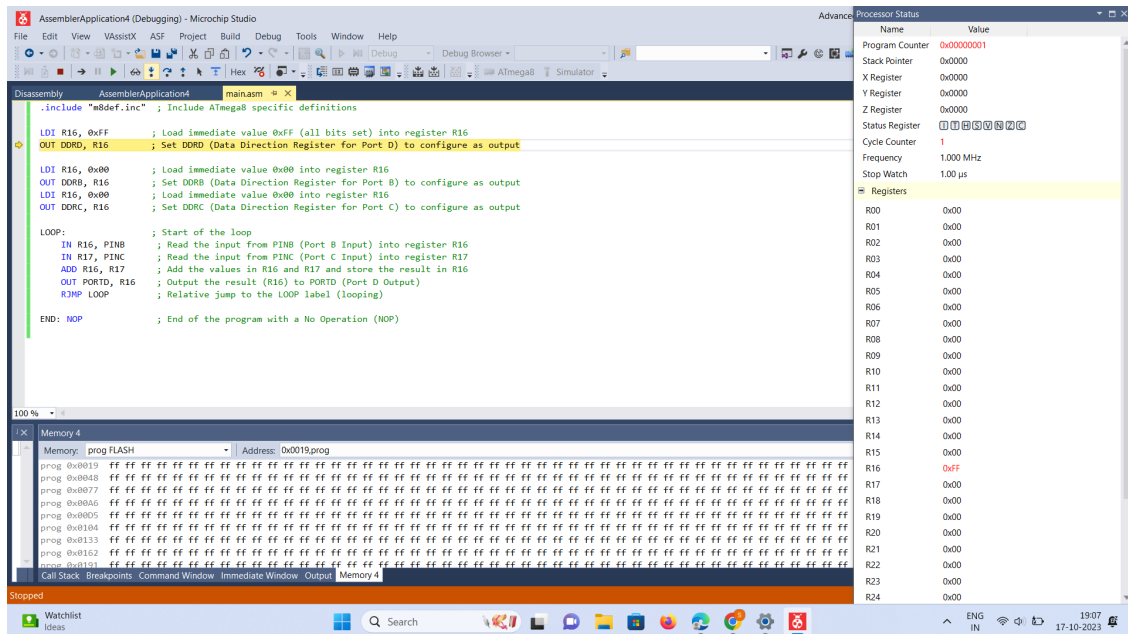


Figure 3: 4-bit addition of two unsigned nibbles from an 8-bit dip input switch

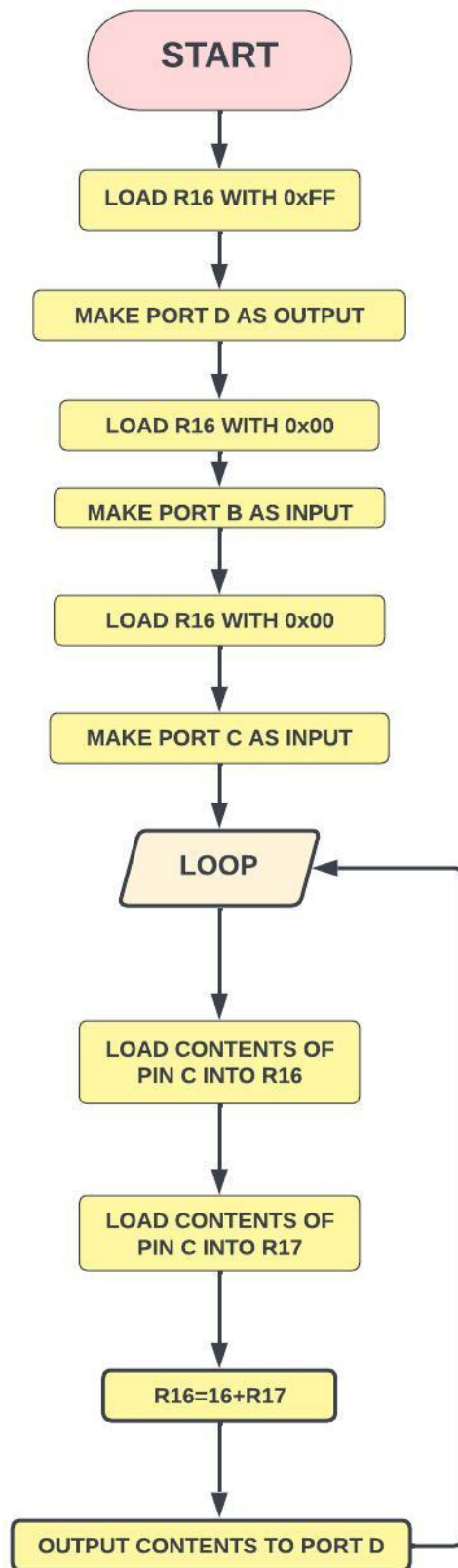


Figure 4: 4-bit addition of two unsigned nibbles from an 8-bit dip input switch

4 4-bit multiplication of two unsigned nibbles

4.1 Code

```
.include "m8def.inc" ; Include ATmega8 specific definitions

LDI R16, 0xFF        ; Load immediate value 0xFF (all bits set) into register R16
OUT DDRD, R16        ; Set DDRD (Data Direction Register for Port D) to configure as output

LDI R16, 0x00        ; Load immediate value 0x00 into register R16
OUT DDRB, R16        ; Set DDRB (Data Direction Register for Port B) to configure as output
LDI R16, 0x00        ; Load immediate value 0x00 into register R16
OUT DDRC, R16        ; Set DDRC (Data Direction Register for Port C) to configure as output

LOOP:                ; Start of the loop
    IN R16, PINB      ; Read the input from PINB (Port B Input) into register R16
    IN R17, PINC      ; Read the input from PINC (Port C Input) into register R17
    MUL R16, R17      ; Multiply the values in R16 and R17 and store the result in R16
    OUT PORTD, R16    ; Output the result (R16) to PORTD (Port D Output)
    RJMP LOOP        ; Relative jump to the LOOP label (looping)

END: NOP              ; End of the program with a No Operation (NOP)
```

4.2 Inferences and Learning Outcomes

Learning Outcomes:

- Understanding AVR Assembly Language
- I/O Port Configuration
- Immediate Value Loading
- Input and Output Operations
- Multiplication Operation
- Looping and Control Flow
- Understanding I/O Concepts

Inferences:

Register R16 is loaded with a value representing all bits set. This value is likely used to configure a Data Direction Register (DDR) to configure a port as an output.

Port D is set as an output port, and you can write data to it to control external devices connected to the pins of Port D. Register R16 is then loaded with a value representing all bits cleared, which can be used to configure Data Direction Registers for Port B and Port C as input.

Port B is set as an input port, allowing you to read the state of external devices or sensors connected to its pins.

Register R16 is loaded with all bits cleared, preparing it for configuring DDRC as an input.

Port C is set as an input port, similar to Port B.

The program is designed to enter a loop, continuously performing the operations inside the loop.

The code reads the states of Port B and Port C, multiplies the values, and stores the result in R16. The code is creating an infinite loop, where it repeatedly reads from Port B and Port C, adds the values, and outputs the result to Port D.

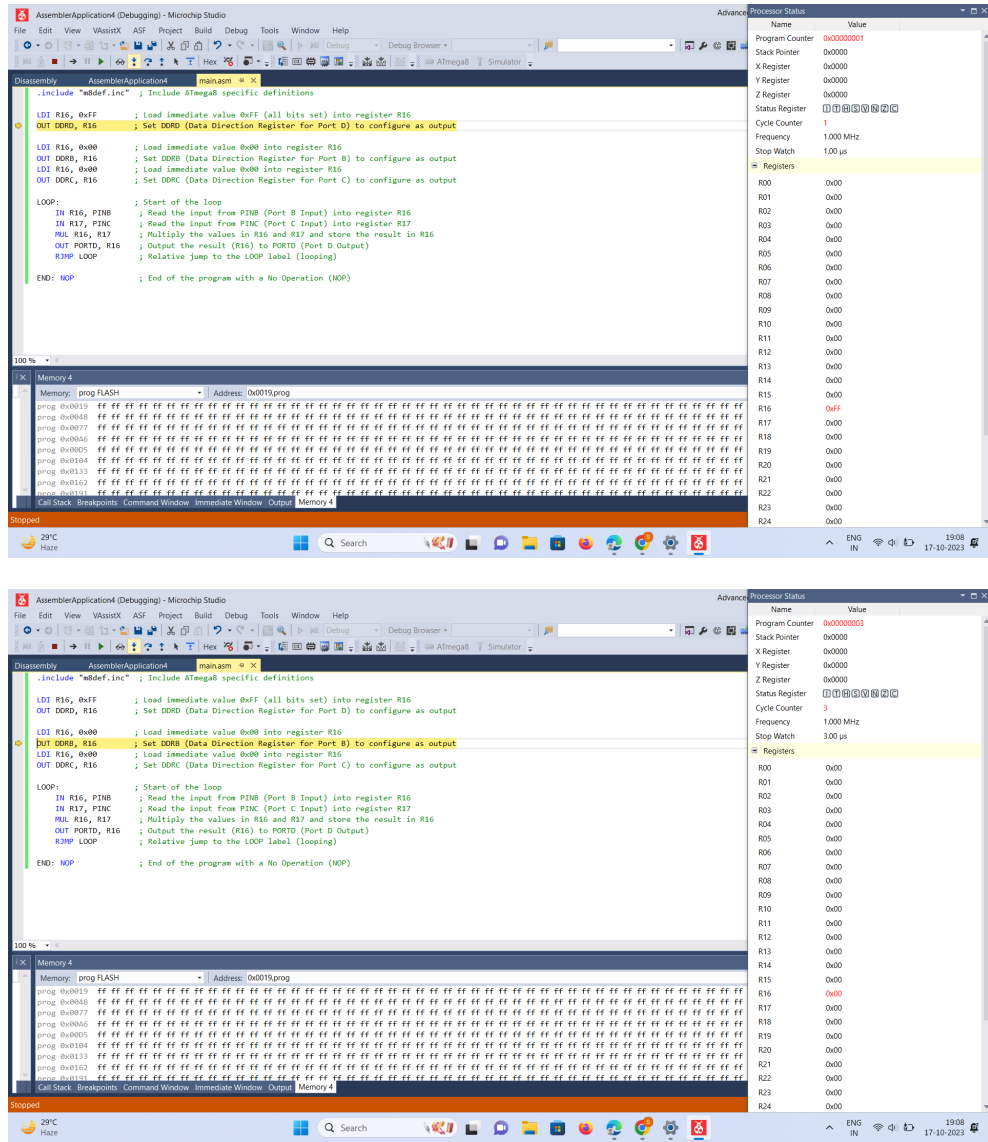


Figure 5: 4-bit multiplication of two unsigned nibbles

4.3 Flow Chart

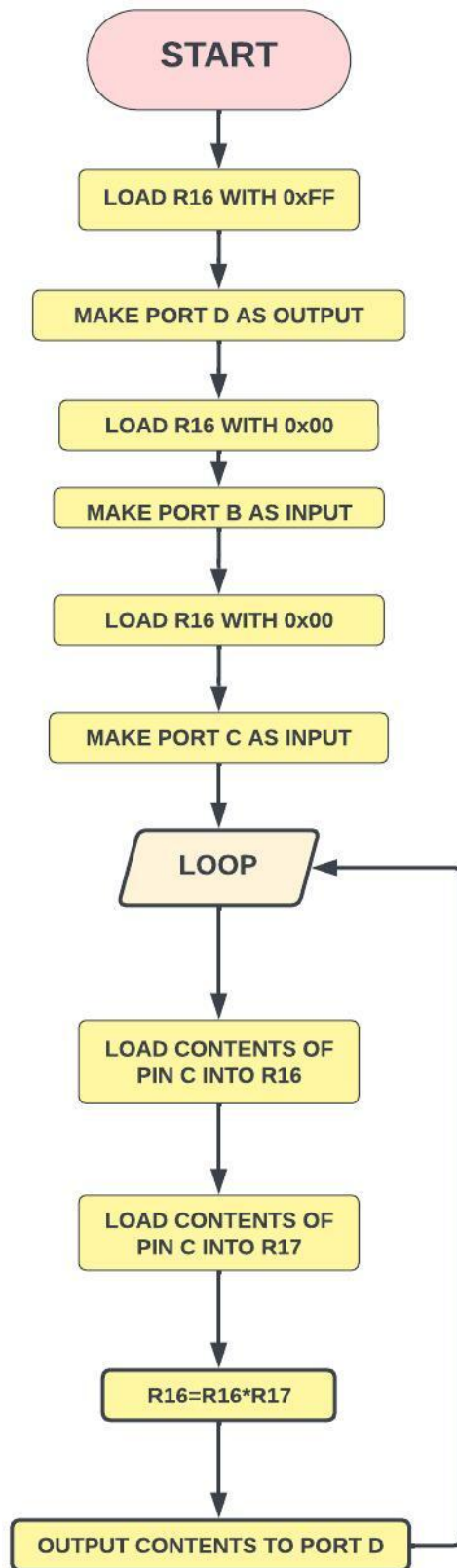


Figure 6: 4-bit multiplication of two unsigned nibbles

5 Individual Contribution

The Experiment was done by both of the team members.

Implimentation of code of LED,addition and their results,flow charts,observations,learning outcomes were done by Maadhav Patel[EE22B033].

Implimentation of code of LED,Multiplication and their results,flow charts,observations,learning outcomes were done by Shravya[EE22B032].

Lab Report was written together by both the team members.