Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: Fall 2022, B.Sc. in CSE (Day)

Course Title: Microprocessor & Microcontroller Lab
Course Code: CSE 304 Section: 203D1

Lab Project Name: Calculator using Assembly Language.

Student Details

	Name	ID
1.	Joy Pal	201002418
2.	Md. Aminul Islam	183002137

Submission Date : 07 January, 2023
Course Teacher's Name : Amena Zahan

[For Teachers use only: Don't Write Anything inside this box]

Lab Project Status

Marks: …… **Signature: .....**

Comments: ..... **Date: .....**

Table of Contents

Chapter 1

Introduction

1.1 Introduction

An assemble (or assembler) language, often abbreviated .asm, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to -one) correspondence between the language and the language and the architecture's machine code instructions. Each assembly language is specific to particular computer architecture. In contrast, most high-level.
Assembly language may also be called symbolic machine code. Assembly language is converted into executable machine code by a utility program referred to as an assembler. The conversion process is referred to as assembly, or assembling the source code. Assembly time is the computational step where an assembler is run. Assembly language uses a mnemonic to represent each low-level machine instruction or opcode, typically also each architectural register, flag, etc.

Many operations require one or more operands in order to form a complete instructions and most assembler can take expressions of numbers and named constants as well as registers and labels as operands, freeing the programmer from tedious repetitive calculations. Depending on architecture, these elements are also be combined for specific instructions or addressing mode using offsets or other data as well as fixed addresses.
Calculators were created in order to give people a simple, fast, and error free method of doing these calculations. The program is designed to act like a "16–bit Decimal Calculator" with the usual standard functions (addition, subtraction, multiplication, division, modulo, and power). This calculator will have the capability of performing arithmetic operations on 16-bit decimal numbers.

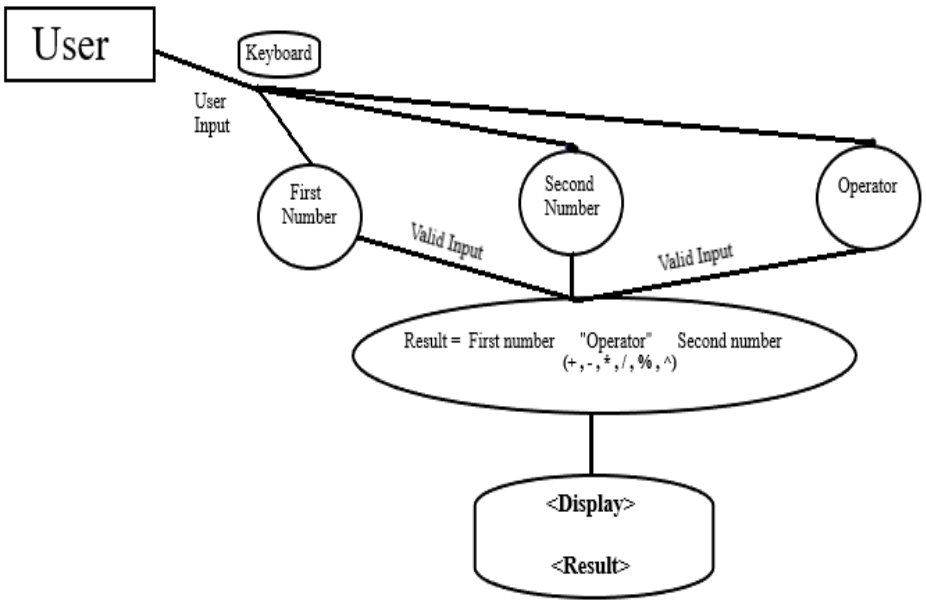
1.2 Design Goals/Objective

- To learn instructions to syntax and structure of assembly language.
- To learn implementation of various basic arithmetic operations and conditional statements in assembly language.
- To gather knowledge how to use loop and array in assembly language.
- To learn 8086 instructions related to procedure using Assembly Language Program.

Chapter 2

Design/Development/Implementation of the Project

2.1 A simple Block Diagram to using this project



2.2 Implementation

```
org 100h

.DATA
   msg1 DB 0AH,0DH, "Enter first  Number :  $"
   msg2 DB 0AH,0DH, "Enter second Number :  $"

   msg3 DB 0AH,0DH, "Enter operation (+,-,*,/,%,^) :  $"
```

```
    result DB 0AH,0DH, "Result is : $"

    proj  DB 0AH,0DH,   "              -------- AJ CALCULATOR -------$"
    done  DB 0AH,0DH, "--------------------------------$"
    done2  DB 0AH,0DH,
          "*********************************************************************************$"
    invalid_message DB 0AH,0DH, "INVALID INPUT$"
    num1 dw 00h
    num2 dw 00h
    overflow db 00h

 .CODE
        include 'emu8086.inc'
        LEA DX,proj
        MOV AH,09H
        INT 21H
        LEA DX,done2
        MOV AH,09H
        INT 21H

calculator:
        MOV AX,@DATA
        MOV DS,AX

        CALL input
        CALL parser
        CALL operation
        MOV [SI],'&'
        call reverse_parser
        call print_result
```

```
input  PROC                ;************   input procedure   **************

        MOV [SI],'&'
        LEA DX,msg1
        MOV AH,09H
        INT 21H

input1:
          MOV AH,01H
          INT 21H
          CMP AL,13d
          JZ  print_message2
          MOV AH,AL
          SUB AH,'0'
          JC invalid
          MOV AH,AL
          MOV DH,'9'
          SUB DH,AH
          JC invalid
          SUB AL,'0'
          INC SI
          MOV [SI],AL
          JMP input1

print_message2:
          INC SI
          MOV [SI],'&'
          LEA DX,msg2
          MOV AH,09H
          INT 21H
```

```
input2:
        MOV AH,01H
        INT 21H
        CMP AL,13d
        JZ exit
        MOV AH,AL
        SUB AH,'0'
        JC invalid
        MOV AH,AL
        MOV DH,'9'
        SUB DH,AH
        JC invalid
        SUB AL,'0'
        INC SI
        MOV [SI],AL
        JMP input2
    exit:
        ret

    invalid: LEA DX,invalid_message
        MOV AH,09H
        INT 21H
        hlt
ENDP                    ;END of input procedure



 parser PROC            ;parser procedure

        MOV CX,01d
        MOV BX,00H
```

```
parse2:
    MOV AX,00H
    MOV AL,[SI]
    MUL CX
    ADD BX,AX
    MOV AX,CX
    MOV CX,10d
    MUL CX
    MOV CX,AX
    DEC SI
    CMP [SI],'&'
    JNZ parse2

    MOV [num2],BX
    MOV BX,00H
    MOV DX,00h
    DEC SI
    MOV CX,01d

parse1:
    MOV AX,00H
    MOV AL,[SI]
    MUL CX
    ADD BX,AX
    MOV AX,CX
    MOV CX,10d
    MUL CX
    MOV CX,AX
    DEC SI
    CMP [SI],'&'
```

```
        JNZ parse1

        MOV [num1],BX
        MOV AX,[num1]
        MOV BX,[num2]


    ret
ENDP   ;END of parser procedure


operation proc              ;operation procedure
        MOV CX,AX
        LEA DX,msg3
        MOV AH,09H
        INT 21H


        MOV AH,01H
        INT 21H

        CMP AL,'+'
        JZ addition

        CMP AL,'-'
        JZ subtraction

        CMP AL,'*'
        JZ multiplication

        CMP AL,'/'
```

```
        JZ division

        CMP AL,'%'
        JZ mod

        CMP AL,'^'
        JZ pow

        LEA DX,invalid_message
        MOV AH,09H
        INT 21H
    hlt

addition:
        MOV AX,CX
        MOV DX,00h
        ADD AX,BX
        ADC AX,DX
        RET
subtraction:
        MOV AX,CX
        SUB AX,BX
        JC ov
        JNC nov
     ov:NEG AX
        MOV [overflow],01h
        RET
    nov:RET

multiplication:
        MOV AX,CX
```

```
        MOV DX,00H
        MUL BX
        RET
division:
        MOV AX,CX
        MOV DX,00H
        ADD BX,DX
        JZ DbyZ
        DIV BX
        RET
 DbyZ:  print '   ERROR : DIVIDE BY ZERO'
        JMP calculator
mod:
        MOV AX,CX
        MOV DX,00H
        ADD BX,DX
        JZ DbZ
        DIV BX
        MOV AX,DX
   DbZ: RET
pow:
        MOV AX,CX
        MOV CX,BX
        ADD CX,00h
        JZ Lc
        SUB CX,01h
        JZ La
        JNZ Lb
   La:  ret
   Lb:  MOV BX,AX
        MOV DX,00h
```

```
    L1:  MUL BX
         LOOP L1
         ret
    Lc:  MOV AX,01h
         ret

ENDP                    ;END OF operation procedure


reverse_parser PROC          ;reverse_parser procedure

   r_parse:
         MOV DX,00h
         MOV BX,10d
         DIV BX
         ADD DL,'0'

         INC SI
         MOV [SI],DL
         ADD AX,00h
         JNZ r_parse


ENDP                    ;END of reverse_parser procedure


print_result PROC            ;print_result procedure

        LEA DX,result
        MOV AH,09H
        INT 21H
```

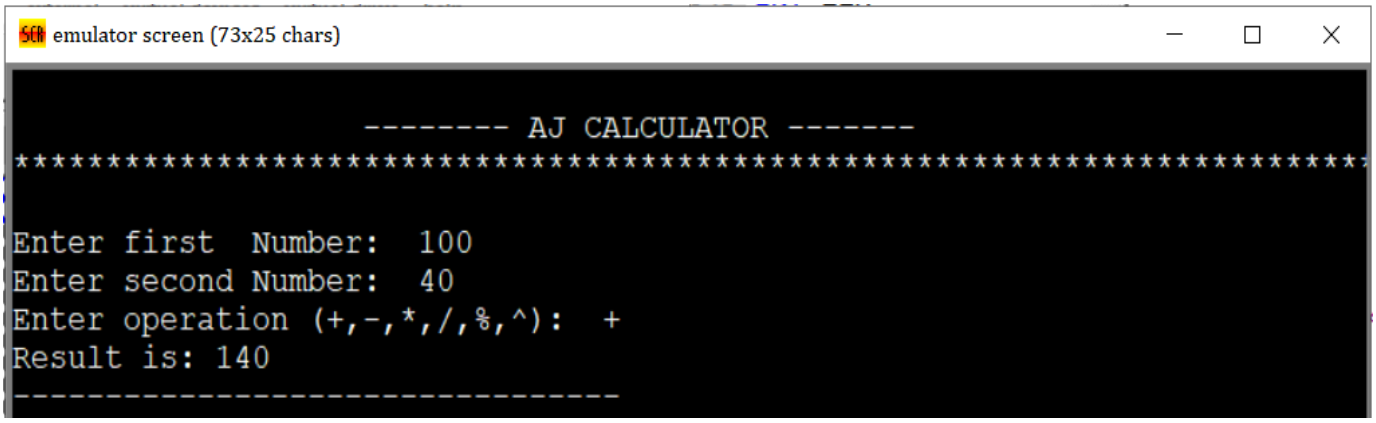
```
        MOV CL,01h
        CMP CL,[overflow]
        MOV [overflow],00h
        JZ print_minus
        JNZ print
print_minus: MOV DL,'-'
        MOV AH,02H
        INT 21H

    print:
        MOV DL,[SI]
        MOV AH,02H
        INT 21H
        DEC SI
        CMP [SI],'&'
        JNZ print

        LEA DX,done
        MOV AH,09H
        INT 21H

        JMP calculator

ENDP                    ;END of print_result procedure
```

Chapter 3

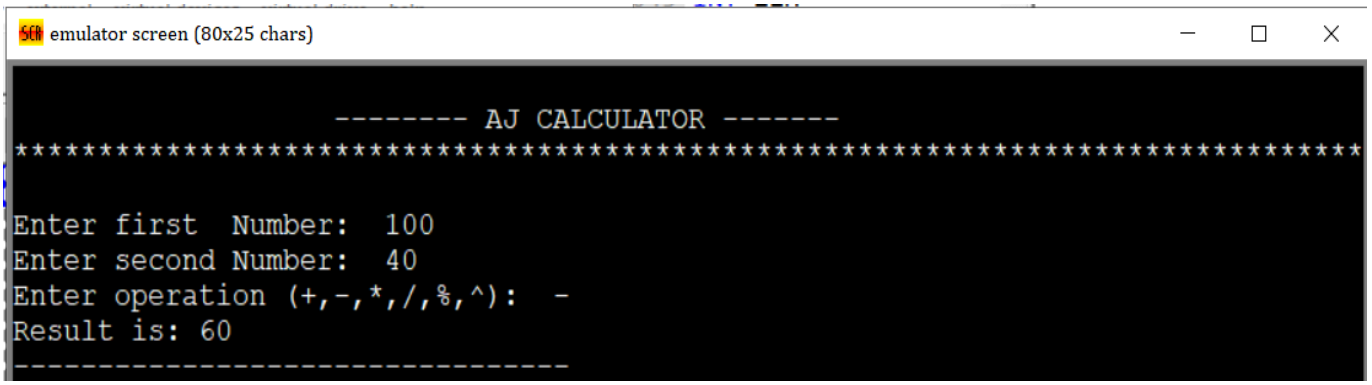
Performance Evaluation

3.1 Results and Output

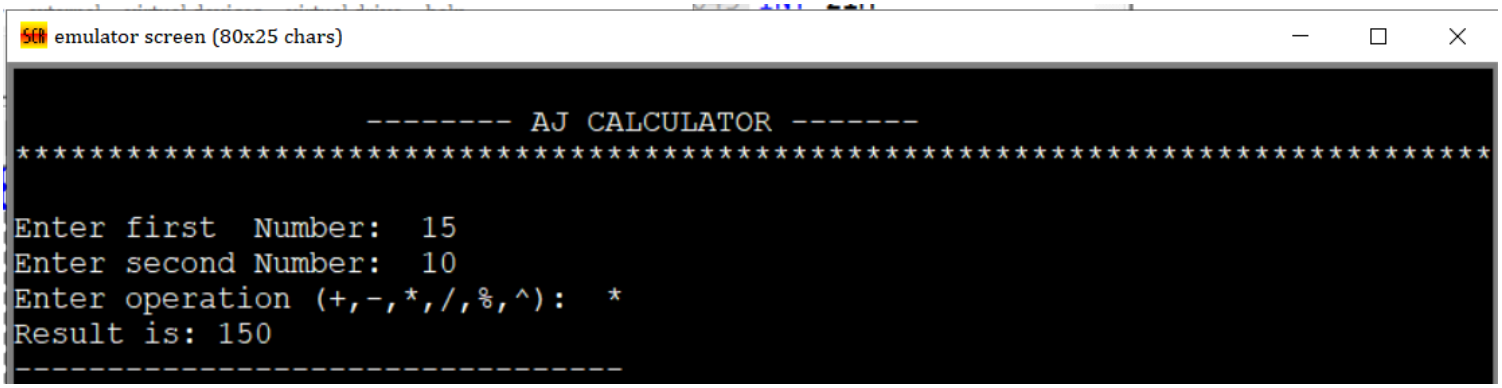
Addition ~



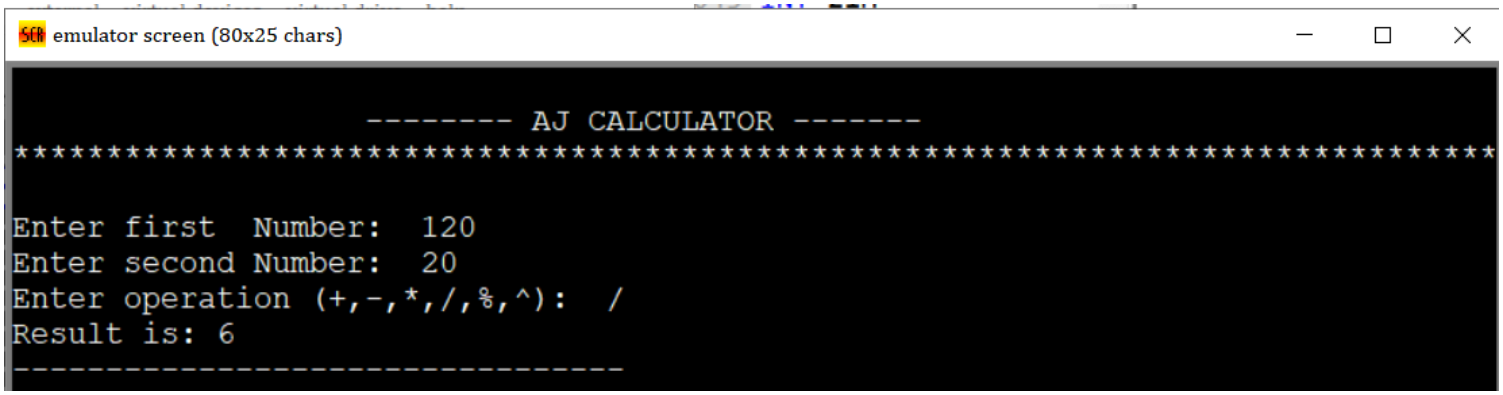
Subtraction ~



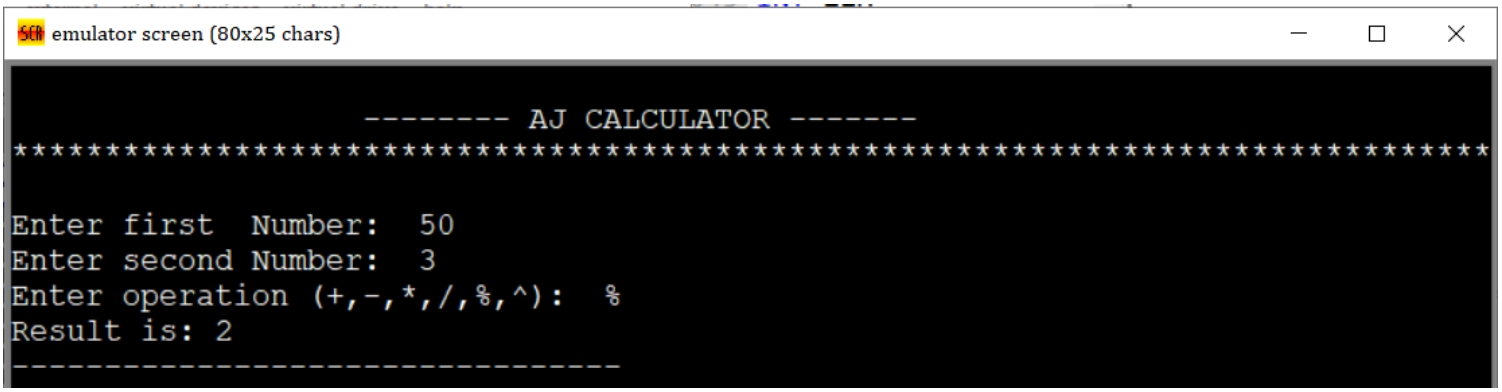
Multiplication ~



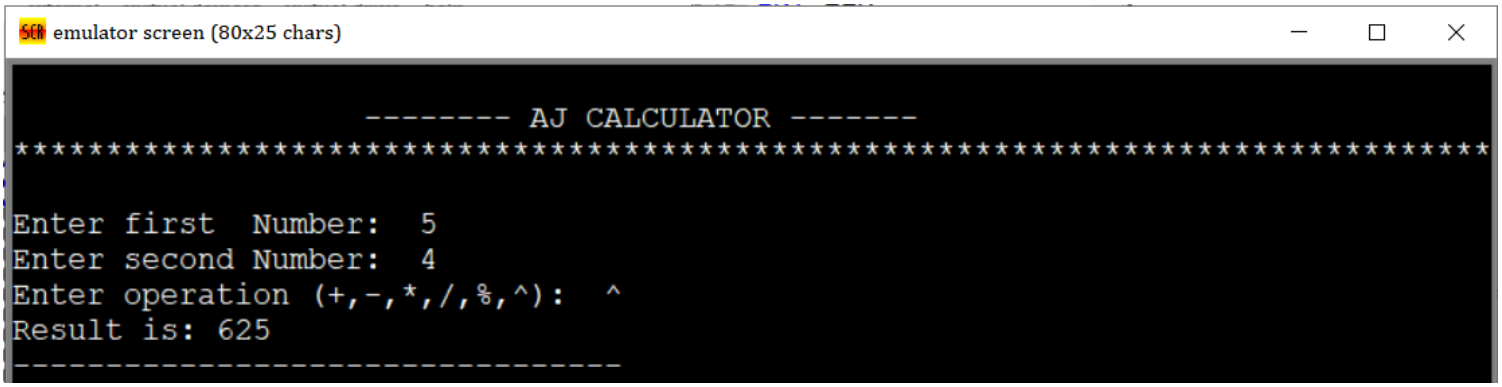
Division ~



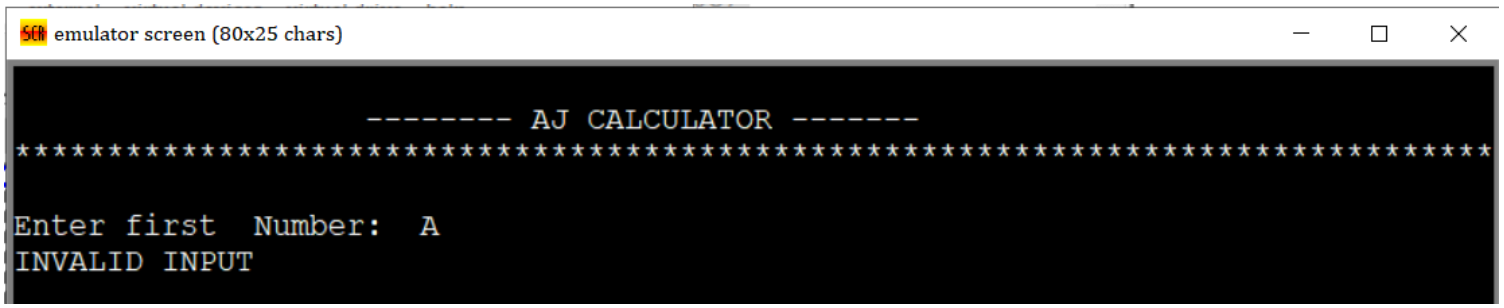
Modulus ~



Power ~



Invalid Output ~



Chapter 4

Conclusion

4.1 Discussion

The project has been successfully completed by having established the user friendly interface with the help of Emu8086.It allows user to perform basic arithmetic operations on 16-bit decimal number (range 0-65535) in an easy way. If we are able to introduce friendly interface for complicated tasks then it gives user what he wants, that will be ultimate success of our attempts.

4.2 Scope of Future Work

At the same time there is some scope for improvement in the feature. It can be possible to make it more user friendly by adding more variety of functions to it and also by increasing its range (Ex. 32-bit ,64-bit etc).

References

[1] Amar sharma, India. Github: github.com/amarsharma441.

[2] Author: Amena Zahan , Lab manuals of Microprocessor & Microcontroller Lab , Green University of Bangladesh.