# Embedded System Lab

(ELC3930)

**Experiment No.: 03**

## Object:

Write a program using 8085 simulator to implement the following function: F = (A*B+C)/4

**G. No:** **GL3136**

**S. No:** **A3EL-02**

**F. No:** **19ELB056**

**Name:** **Maha Zakir Khan**

**Date of performing experiment: 24|01|2022**

**Date of report submission: 30|01|2022**

# Simulator Used:

8085 Simulator by Jubin Mitra. It helps in get started easily with example codes, and to learn the architecture playfully. This tool is an integrated software environment for teaching microprocessor concepts. The software is shared under opensource GNU license.

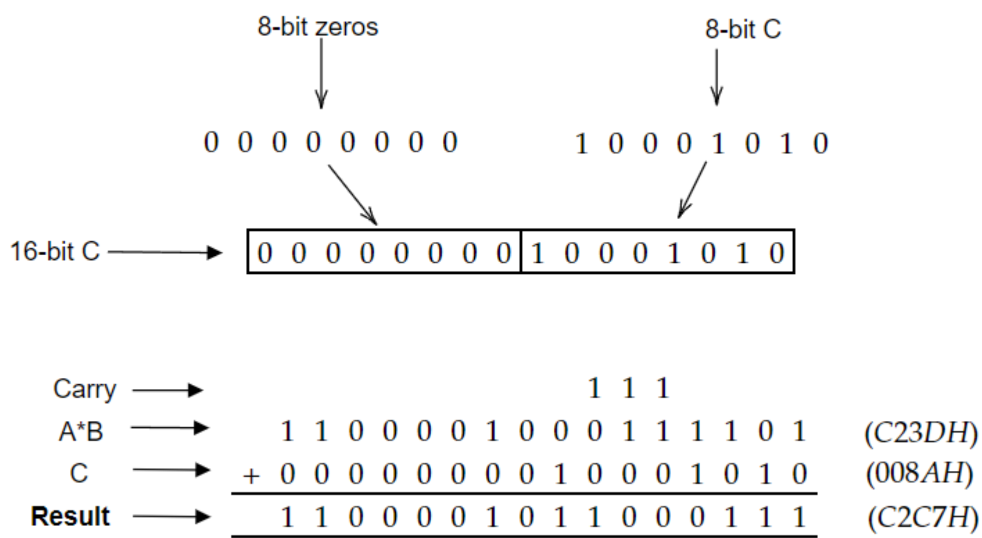
**Link:** [8085 Jubin Simulator]

# Algorithm:

To implement the function F = (A*B+C)/4, three different operations need to be performed: Multiplication, Addition and Division. For Multiplication we can use the same algorithm as in *Experiment-02*.

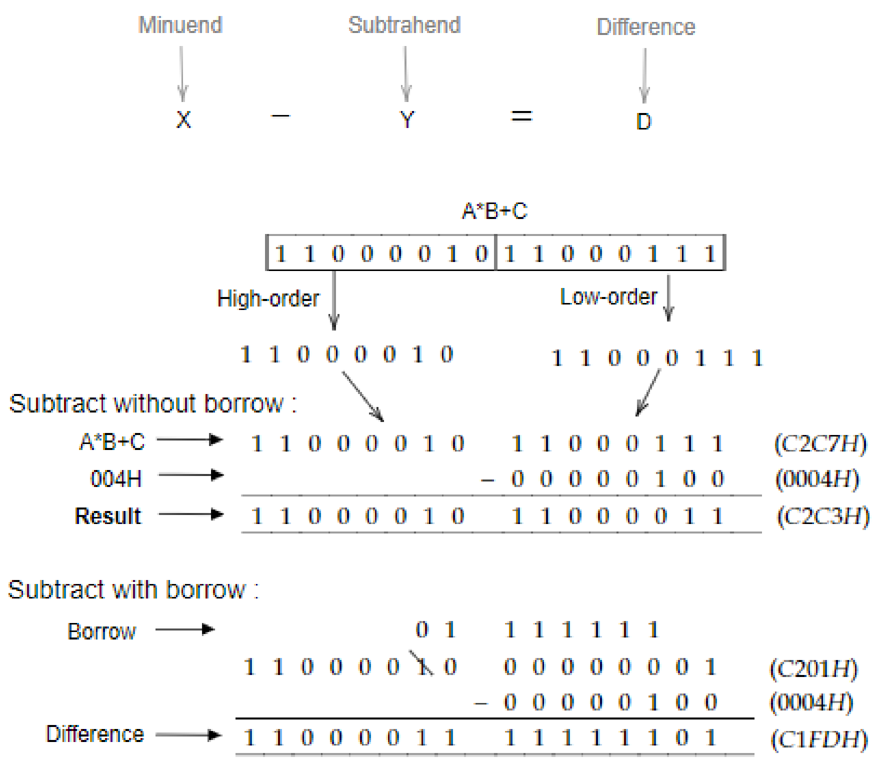
The logic for Addition is relatively simple. We need to perform Addition of 8-bit number (C in this case) with 16-bit result of multiplication (i.e. A*B in this case). One way to implement this is to add two more 8 zeros in-front of our 8-bit number and then perform binary Addition

**16-Bit with 8-bit Binary Addition**

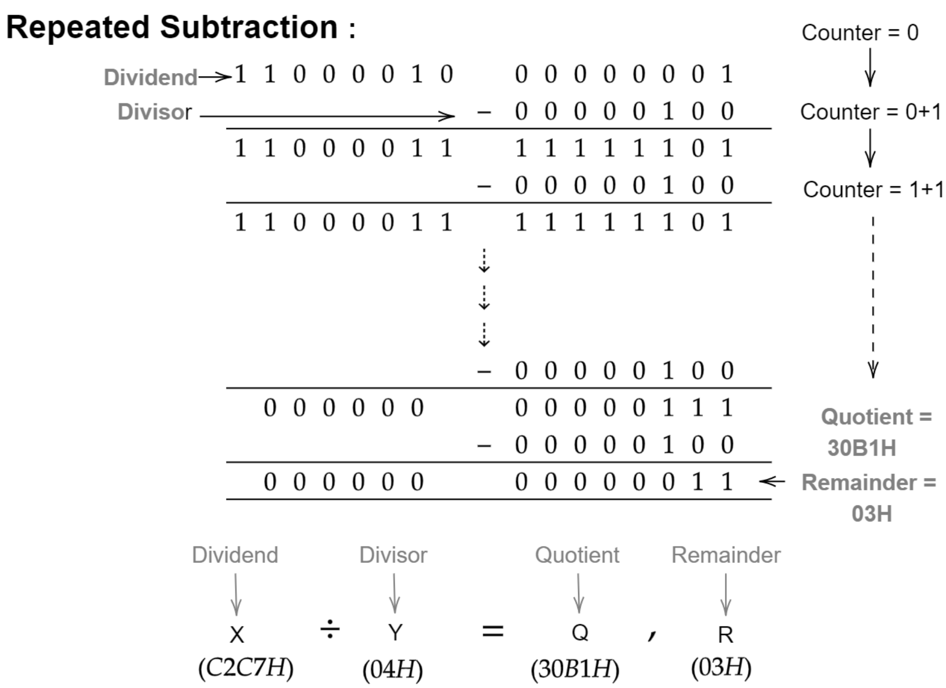


Given below is 16-bit with 8-bit subtraction, the Minuend is a 16-bit number, whereas the Subtrahend is 8-bit, so we need to break the Minuend into two 8-bit High-order and Low-order numbers. The Subtrahend performs subtraction on Low-order number and borrows 1 from the High-order number whenever required.

## 16-Bit with 8-bit Binary Subtraction

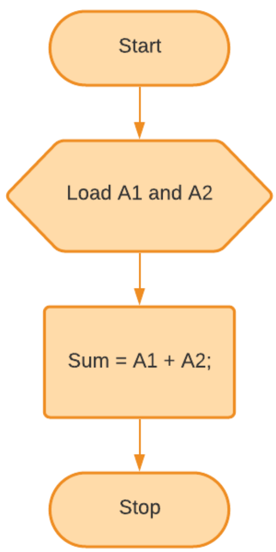


For Division we can use repetitive subtraction of Minuend (A*B+C in this case) with Subtrahend (0004H in this case) until the Difference is less than the Minuend, here the Minuend act as the Dividend, Subtrahend is the Divisor and the resulting Difference is the Remainder. The number of time subtraction occurs is the Quotient. A counter keeps track of the number of times the subtraction is performed.

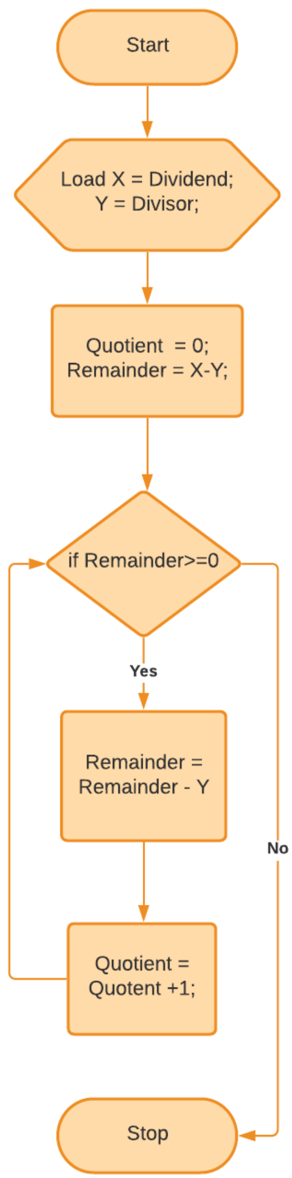


# Flow Chart:

## 1)Addition:




## 2) Division:

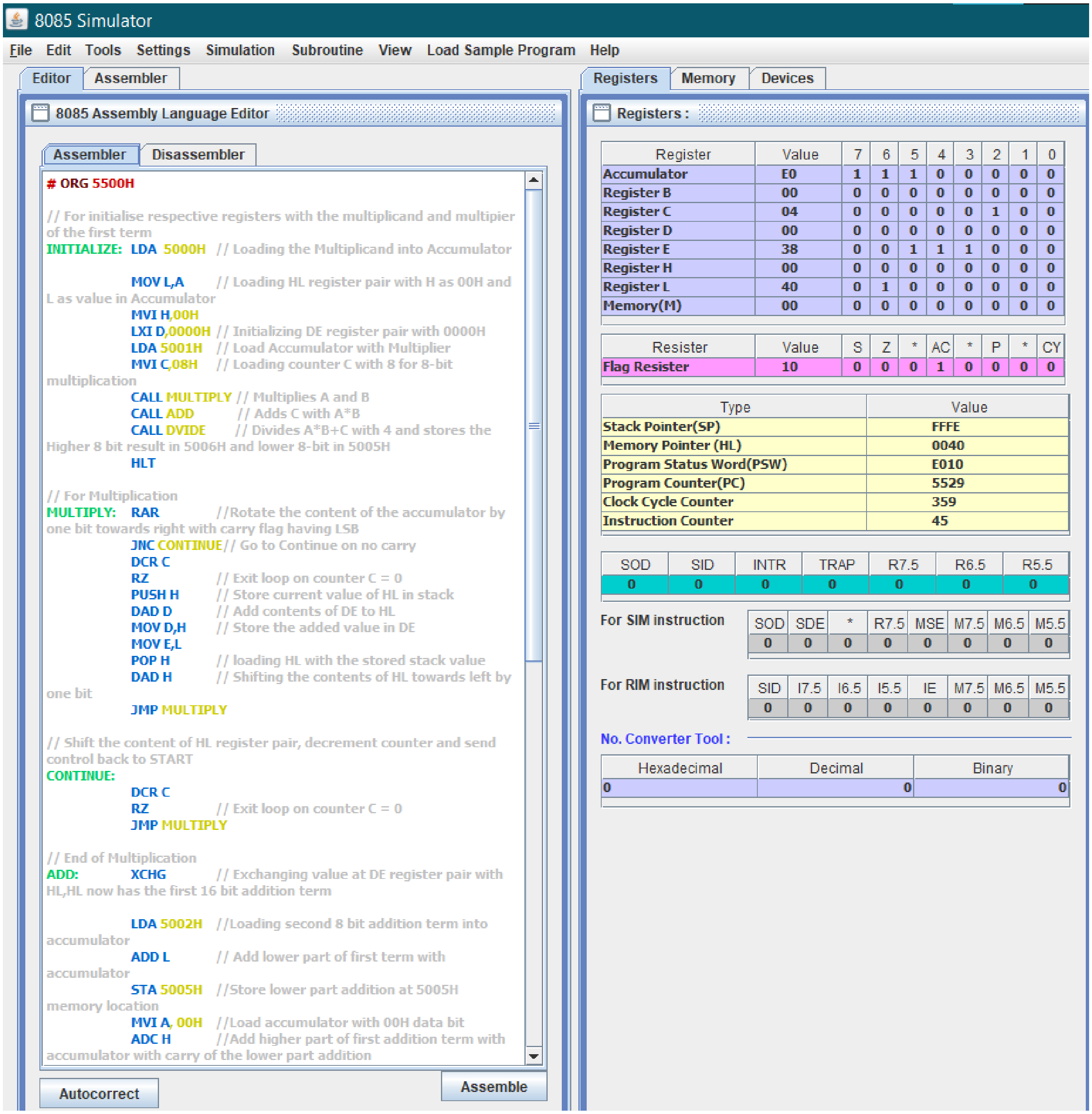


# Program:

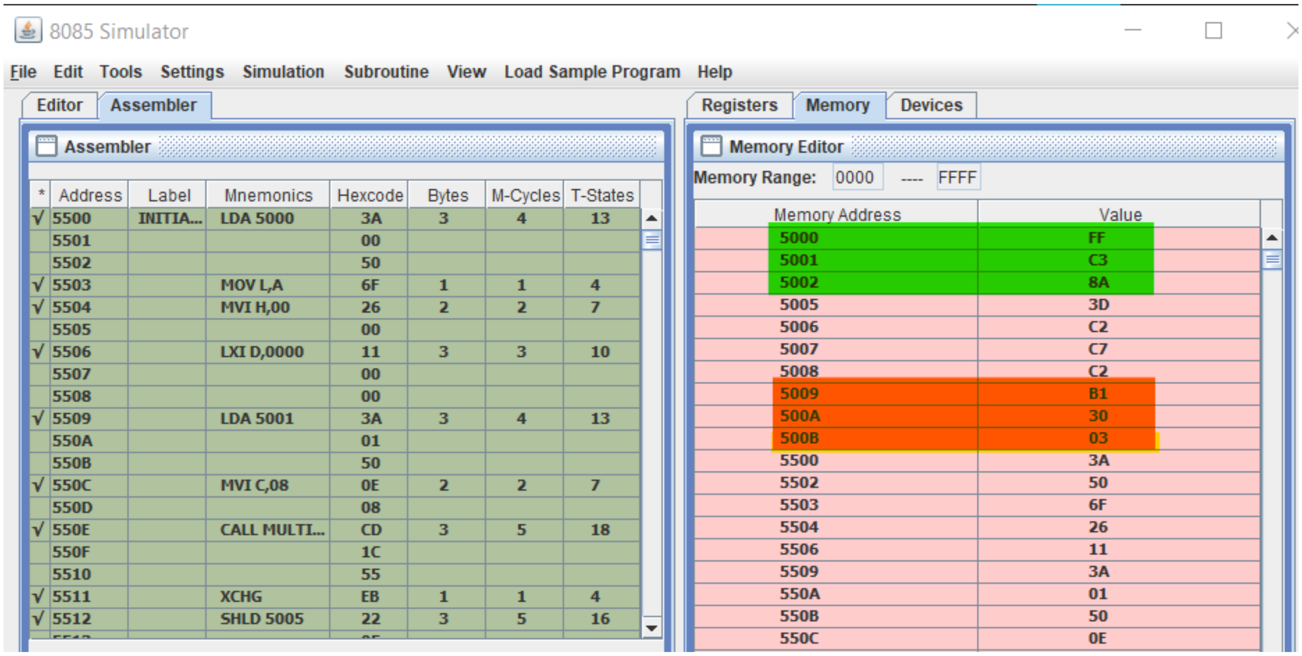
```
# ORG 5500H
 // For initializing respective registers with the multiplicand and multiplier of the first term and calling
subroutines
INITIALIZE:
  LDA 5000       // Loading the Multiplicand into Accumulator
  MOV L,A        // Loading HL register pair with H as 00H and L as value in Accumulator
  MVI H,00
  LXI D,0000     // Initializing DE register pair with 0000H
  LDA 5001       // Load Accumulator with Multiplier
  MVI C,08       // Loading counter C with 8 for 8-bit multiplication
  CALL MULTIPLY        // Multiplies A and B
  XCHG           // Exchanging value at DE register pair with HL, HL now has the first 16-bit addition
                    term
  SHLD 5005      // Load multiplication result at 5007H and 5008H memory location
  CALL ADD       // Adds C with A*B
  CALL DVIDE     // Divides A*B+C with 4 and stores the Higher 8 bit result in 5006H and lower 8-bit
                   in 5005H
  HLT

// For Multiplication
MULTIPLY:
  RAR            // Rotate the content of the accumulator by one bit towards right with carry flag
            having LSB
  JNC CONTINUE         // Go to Continue on no carry
  PUSH H         // Store current value of HL in stack
  DAD D          // Add contents of DE to HL
  MOV D,H        // Store the added value in DE
  MOV E,L
  POP H          // loading HL with the stored stack value
  DAD H          // Shifting the contents of HL towards left by one bit
  DCR C
  RZ
  JMP MULTIPLY

// Shift the content of HL register pair, decrement counter and send control back to START
CONTINUE:
  DCR C
  RZ                   // Exit loop on counter C = 0
  DAD H
  JMP MULTIPLY         // End of Multiplication

ADD:
  LDA 5002       // Loading second 8-bit addition term into accumulator
  ADD L          // Add lower part of first term with accumulator
```

```
    STA 5007      // Store lower part addition at 5005H memory location
    MVI A,00      // Load accumulator with 00H data bit
    ADC H         // Add higher part of first addition term with accumulator with carry of the lower
              part addition
    STA 5008      // Store the higher part addition result at 5006H memory location
    RET

DVIDE:
    LHLD 5007     // Loads the 16-bit first term
    MOV A,L       // Store lower part of 16-bit number
    LXI D,0000    // Counter to count number of subtractions
    CALL LOOP
    MOV A,B
    STA 500B      // Store Remainder
    XCHG
    SHLD 5009     // Store Quotient
    RET

// Outer loop performs subtraction
LOOP:    CPI 04
    CC LOOP2      // Calls inner loop when accumulator has value less than 04H
    SUI 04        // Subtracting 4 from the accumulator
    INX D         // Incrementing counter
    JMP LOOP

// Inner loop for generating borrow
LOOP2:   MOV B,A
    MOV A,H
    CPI 00
    JZ EXIT       // Returns to LOOP when H register is zero
    DCR A         // Decrementing by one bit the higher 8bit part of the 16-bit number
    MOV H,A
    MOV A,B
    RET

EXIT:    POP H          // Pop out Program counter pointing at next instruction after subroutine call to
                    LOOP2 in LOOP
    RET    // Return to next instruction after call LOOP in DVIDE

// Loading 5000H,5001H and 5002H with 8-bit numbers A, B and C
# ORG 5000H
# DB FFH,C3H,8AH

```

# Screen-grab of Simulator:



# Result:




| Input | Address | Value | | Output | Address | Value | |
|---|---|---|---|---|---|---|---|
| A | 5000H: | FF | | A*B | 5006H: | C2 | C23DH |
| | | | | | 5005H: | 3D | |
| B | 5001H: | C3 | | A*B+C | 5008H: | C2 | C2C7H |
| | | | | | 5007H: | C7 | |
| C | 5002H: | 8A | | | | | |
| | | | **Result:** | (A*B+C)/4 | 5009H: | 30 | Quotient (30B1H) |
| | | | | | 500AH: | B1 | |
| | | | | | 500BH: | 03 | Remainder (03H) |

# Discussion:

In this Experiment, I used three different code blocks to compute the value of the Function F = (A*B+C)/4 and made subroutine calls to the respective blocks from the initialize block. The addition block didn't require any looping operation. The division block used two nested loops, outer for lower-order subtraction and inner for borrow from high-order number. More information about multiply block is given in *Experiment-02*