

# 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.

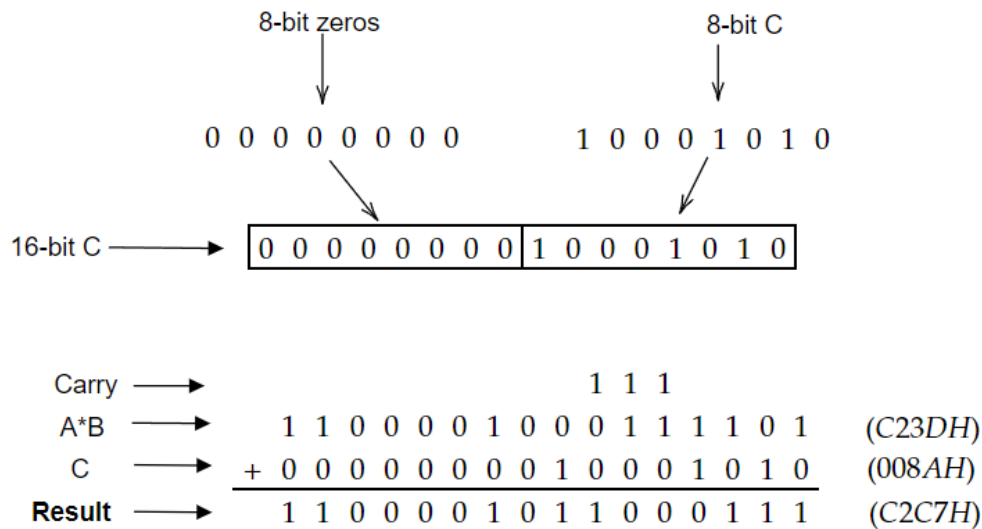
Link: [8085 Jubin Simulator](#)

## 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*.

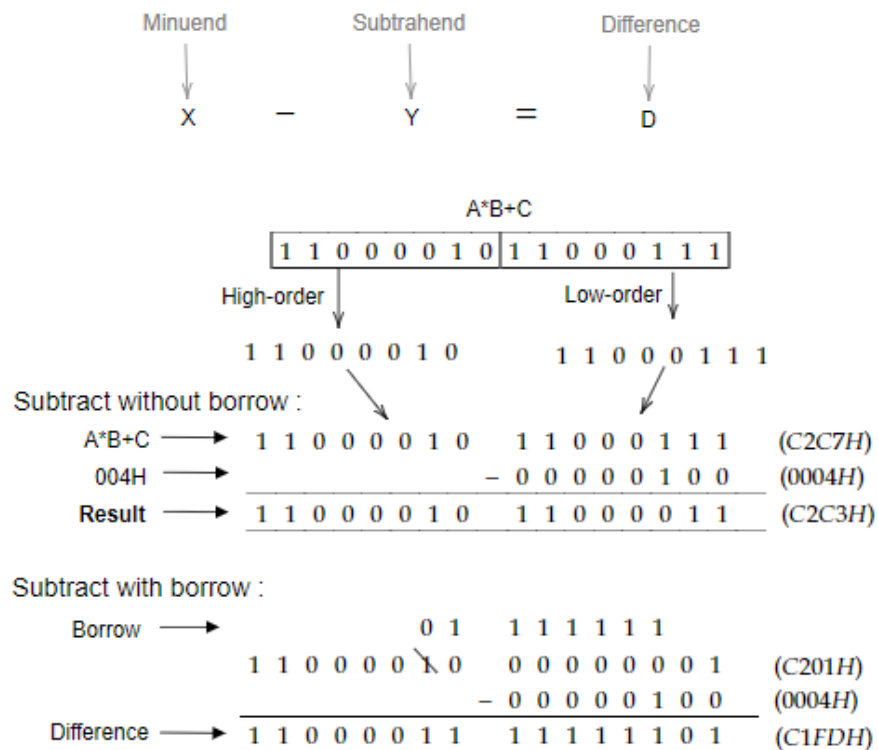
The logic for Addition is relatively simple. We need to perform Addition of 8-bit number (C in this case) with 16-bit result of multiplication (i.e. A\*B in this case). One way to implement this is to add two more 8 zeros in-front of our 8-bit number and then perform binary Addition

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



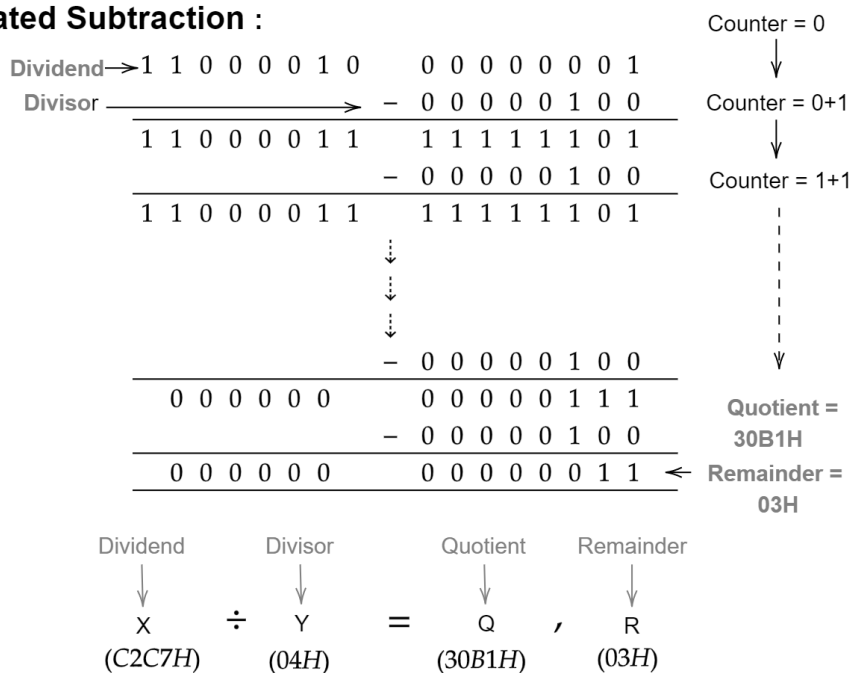
Given below is 16-bit with 8-bit subtraction, the Minuend is a 16-bit number, whereas the Subtrahend is 8-bit, so we need to break the Minuend into two 8-bit High-order and Low-order numbers. The Subtrahend performs subtraction on Low-order number and borrows 1 from the High-order number whenever required.

## 16-Bit with 8-bit Binary Subtraction



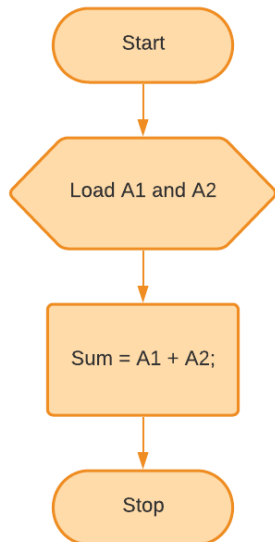
For Division we can use repetitive subtraction of Minuend (A\*B+C in this case) with Subtrahend (0004H in this case) until the Difference is less than the Minuend, here the Minuend act as the Dividend, Subtrahend is the Divisor and the resulting Difference is the Remainder. The number of time subtraction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

### Repeated Subtraction :

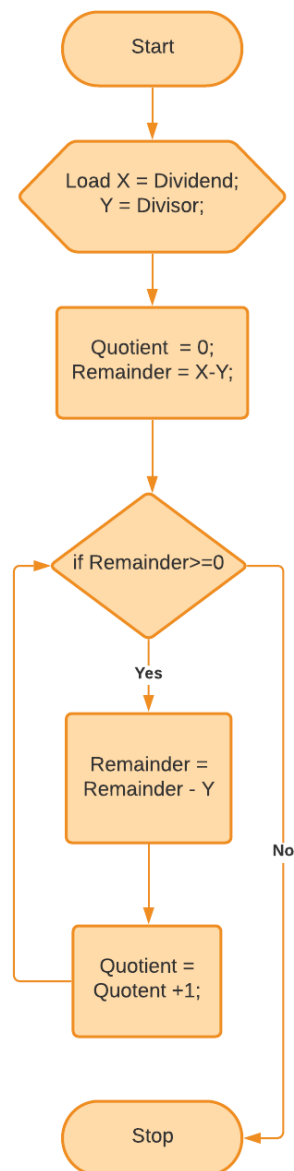


# Flow Chart:

## 1) Addition:



## 2) Division:



# Program:

```
1.  # ORG 5500H
2.  // For initializing respective registers with the multiplicand and multiplier of the first term and calling
    subroutines
3.  INITIALIZE:
4.    LDA 5000    // Loading the Multiplicand into Accumulator
5.    MOV L,A     // Loading HL register pair with H as 00H and L as value in Accumulator
6.    MVI H,00
7.    LXI D,0000  // Initializing DE register pair with 0000H
8.    LDA 5001    // Load Accumulator with Multiplier
9.    MVI C,08    // Loading counter C with 8 for 8-bit multiplication
10.   CALL MULTIPLY // Multiplies A and B
11.   XCHG        // Exchanging value at DE register pair with HL, HL now has the first 16-bit addition
                    term
12.   SHLD 5005    // Load multiplication result at 5007H and 5008H memory location
13.   CALL ADD     // Adds C with A*B
14.   CALL DVIDE   // Divides A*B+C with 4 and stores the Higher 8 bit result in 5006H and lower 8-bit
                    in 5005H
15.   HLT
16.
17. // For Multiplication
18. MULTIPLY:
19.   RAR          // Rotate the content of the accumulator by one bit towards right with carry flag
                    having LSB
20.   JNC CONTINUE // Go to Continue on no carry
21.   PUSH H       // Store current value of HL in stack
22.   DAD D        // Add contents of DE to HL
23.   MOV D,H      // Store the added value in DE
24.   MOV E,L
25.   POP H        // loading HL with the stored stack value
26.   DAD H        // Shifting the contents of HL towards left by one bit
27.   DCR C
28.   RZ
29.   JMP MULTIPLY
30.
31. // Shift the content of HL register pair, decrement counter and send control back to START
32. CONTINUE:
33.   DCR C
34.   RZ           // Exit loop on counter C = 0
35.   DAD H
36.   JMP MULTIPLY // End of Multiplication
37.
38. ADD:
39.   LDA 5002    // Loading second 8-bit addition term into accumulator
40.   ADD L       // Add lower part of first term with accumulator
```

```

41. STA 5007      // Store lower part addition at 5005H memory location
42. MVI A,00      // Load accumulator with 00H data bit
43. ADC H         // Add higher part of first addition term with accumulator with carry of the lower
                    part addition
44. STA 5008      // Store the higher part addition result at 5006H memory location
45. RET
46.
47. DVIDE:
48. LHLD 5007     // Loads the 16-bit first term
49. MOV A,L       // Store lower part of 16-bit number
50. LXI D,0000    // Counter to count number of subtractions
51. CALL LOOP
52. MOV A,B
53. STA 500B      // Store Remainder
54. XCHG
55. SHLD 5009     // Store Quotient
56. RET
57.
58. // Outer loop performs subtraction
59. LOOP:  CPI 04
60.  CC LOOP2     // Calls inner loop when accumulator has value less than 04H
61.  SUI 04       // Subtracting 4 from the accumulator
62.  INX D        // Incrementing counter
63.  JMP LOOP
64.
65. // Inner loop for generating borrow
66. LOOP2: MOV B,A
67.  MOV A,H
68.  CPI 00
69.  JZ EXIT      // Returns to LOOP when H register is zero
70.  DCR A        // Decrementing by one bit the higher 8bit part of the 16-bit number
71.  MOV H,A
72.  MOV A,B
73.  RET
74.
75. EXIT:  POP H      // Pop out Program counter pointing at next instruction after subroutine call to
                    LOOP2 in LOOP
76.  RET // Return to next instruction after call LOOP in DVIDE
77.
78. // Loading 5000H,5001H and 5002H with 8-bit numbers A, B and C
79. # ORG 5000H
80. # DB FFH,C3H,8AH
81.

```

# Screen-grab of Simulator:

8085 Simulator

File Edit Tools Settings Simulation Subroutine View Load Sample Program Help

Editor Assembler Registers Memory Devices

8085 Assembly Language Editor

Assembler Disassembler

```
# ORG 5500H

// For initialise respective registers with the multiplicand and multiplier
// of the first term
INITIALIZE: LDA 5000H // Loading the Multiplicand into Accumulator

MOV L,A // Loading HL register pair with H as 00H and
L as value in Accumulator
MVI H,00H
LXI D,0000H // Initializing DE register pair with 0000H
LDA 5001H // Load Accumulator with Multiplier
MVI C,08H // Loading counter C with 8 for 8-bit
multiplication

CALL MULTIPLY // Multiplies A and B
CALL ADD // Adds C with A*B
CALL DIVIDE // 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
DCR C
RZ // Exit loop on counter C = 0
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

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
JMP MULTIPLY

// End of Multiplication
ADD: XCHG // Exchanging value at DE register pair with
HL,HL now has the first 16 bit addition term

LDA 5002H //Loading second 8 bit addition term into
accumulator
ADD L // Add lower part of first term with
accumulator
STA 5005H //Store lower part addition at 5005H
memory location
MVI A, 00H //Load accumulator with 00H data bit
ADC H //Add higher part of first addition term with
accumulator with carry of the lower part addition
```

Autocorrect Assemble

Registers :

Register	Value	7	6	5	4	3	2	1	0
Accumulator	E0	1	1	1	0	0	0	0	0
Register B	00	0	0	0	0	0	0	0	0
Register C	04	0	0	0	0	0	1	0	0
Register D	00	0	0	0	0	0	0	0	0
Register E	38	0	0	1	1	1	0	0	0
Register H	00	0	0	0	0	0	0	0	0
Register L	40	0	1	0	0	0	0	0	0
Memory(M)	00	0	0	0	0	0	0	0	0

Resister	Value	S	Z	*	AC	*	P	*	CY
Flag Register	10	0	0	0	1	0	0	0	0

Type	Value
Stack Pointer(SP)	FFFE
Memory Pointer (HL)	0040
Program Status Word(PSW)	E010
Program Counter(PC)	5529
Clock Cycle Counter	359
Instruction Counter	45

SOD	SID	INTR	TRAP	R7.5	R6.5	R5.5
0	0	0	0	0	0	0

For SIM instruction

SOD	SDE	*	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

For RIM instruction

SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

No. Converter Tool :

Hexadecimal	Decimal	Binary
0	0	0



# Result:

8085 Simulator							
File Edit Tools Settings Simulation Subroutine View Load Sample Program Help							
Editor Assembler				Registers Memory Devices			
Assembler							
* Address	Label	Mnemonics	Hexcode	Bytes	M-Cycles	T-States	
✓ 5500	INITIA...	LDA 5000	3A	3	4	13	
5501			00				
5502			50				
✓ 5503		MOV L,A	6F	1	1	4	
✓ 5504		MVI H,00	26	2	2	7	
5505			00				
✓ 5506		LXI D,0000	11	3	3	10	
5507			00				
5508			00				
✓ 5509		LDA 5001	3A	3	4	13	
550A			01				
550B			50				
✓ 550C		MVI C,08	0E	2	2	7	
550D			08				
✓ 550E		CALL MULTL...	CD	3	5	18	
550F			1C				
5510			55				
✓ 5511		XCHG	EB	1	1	4	
✓ 5512		SHLD 5005	22	3	5	16	
5513			2F				

Memory Editor	
Memory Range: 0000 ---- FFFF	
Memory Address	Value
5000	FF
5001	C3
5002	8A
5005	3D
5006	C2
5007	C7
5008	C2
5009	B1
500A	30
500B	03
5500	3A
5502	50
5503	6F
5504	26
5506	11
5509	3A
550A	01
550B	50
550C	0E

Input	Address	Value	Output	Address	Value
A	5000H:	FF	A*B	5006H:	C2 C23DH
B	5001H:	C3	A*B+C	5005H:	3D
C	5002H:	8A		5008H:	C2 C2C7H
				5007H:	C7
			<b>Result: (A*B+C)/4</b>	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*