Embedded System Lab

(ELC3930)

Experiment No.: 04

Object:

Write a program using 8085 simulator to Divide a 16-bit number by an 8-bit number using rotation method.

G. No: GL3136

S. No: A3EL-02

F. No: 19ELB056

Name: Maha Zakir Khan

Date of performing experiment: 07|02|2022

Date of report submission: 13|02|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:

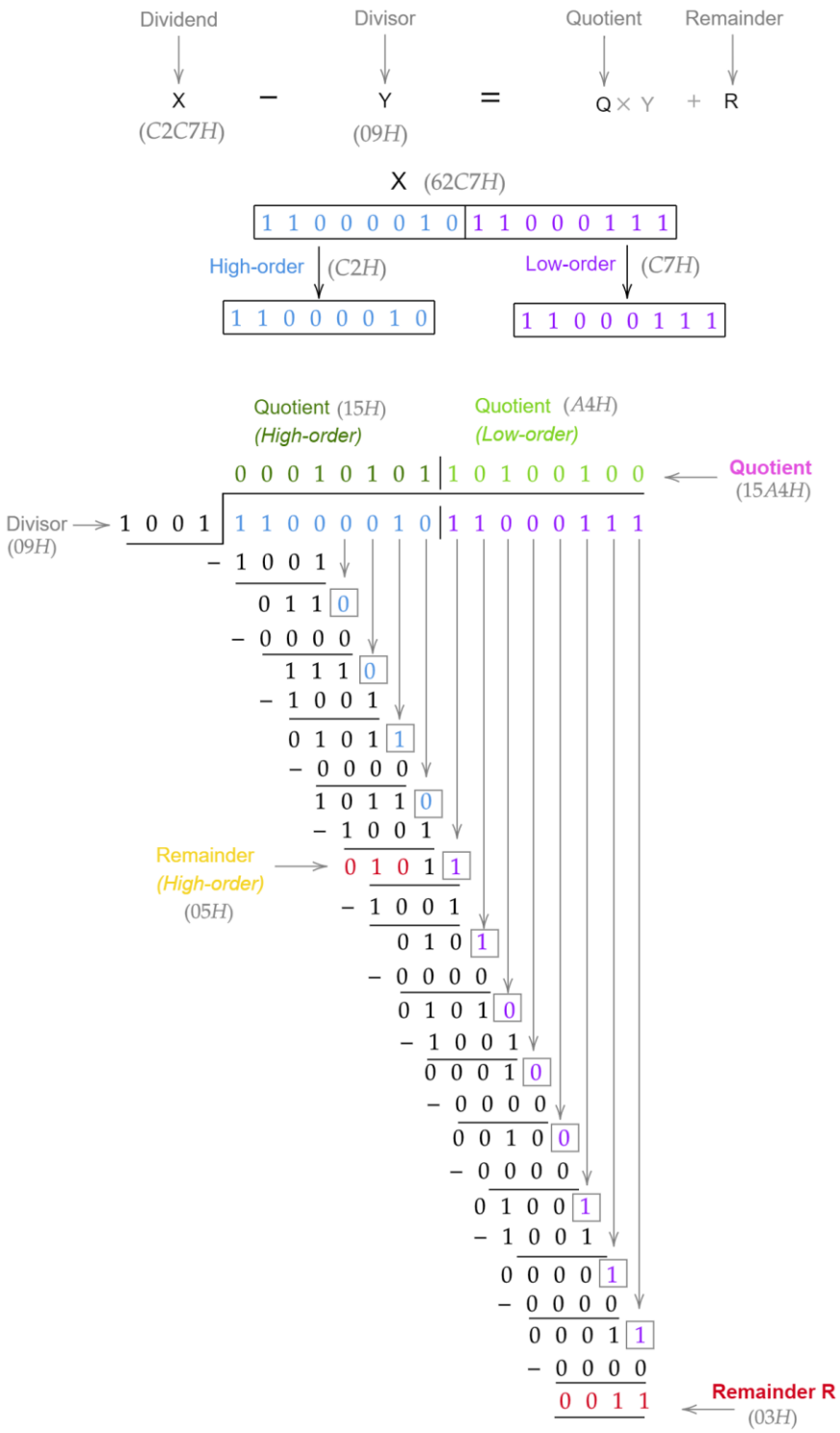
Given below is the algorithm I used to implement 16-bit with 8-bit number division, by modifying the 4-bit divisor to 8-bit, the algorithm remains the same. First the dividend is divided into high-order and low-order parts, then the divisor performs binary division as follow:

STEP 1: If MSB of Extracted-dividend > Divisor then subtract divisor from Extracted-dividend and increment Quotient by 1

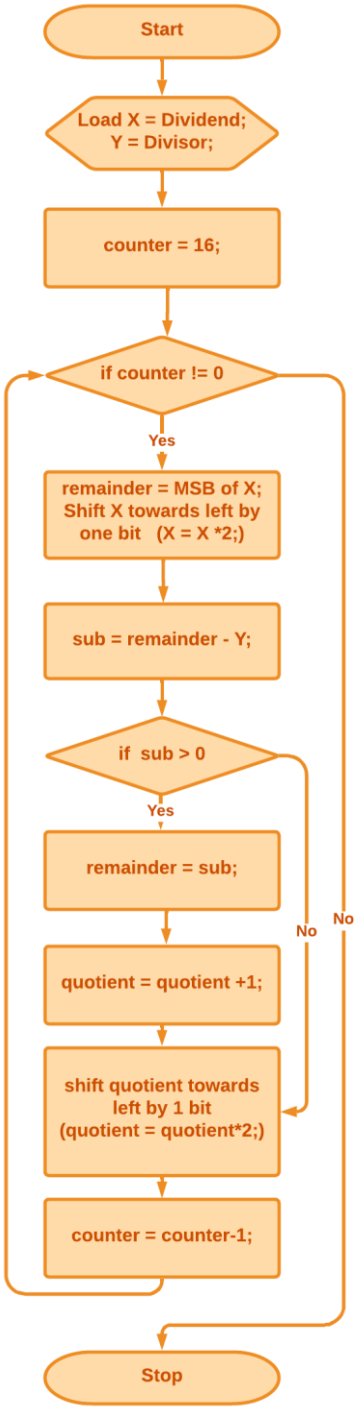
STEP 2: Shift Dividend and Quotient towards left by one-bit, Extract MSB of dividend and add into Extracted-dividend

STEP 3: Repeat 1 and 2 till all 16 bits of dividend are extracted

Using 8-bit with 4-bit Binary division to divide 16-bit number



Flow Chart:


Start
Load X = Dividend;
Y = Divisor;
counter = 16;
if counter != 0
Yes
No
remainder = MSB of X;
Shift X towards left by
one bit (X = X *2;)
sub = remainder - Y;
if sub > 0
Yes
No
remainder = sub;
quotient = quotient +1;
shift quotient towards
left by 1 bit
(quotient = quotient*2;)
counter = counter-1;
Stop

Program:

```
# ORG 5000H
INITIALIZE:     LDA 4000
MOV L,A         // Load L register with High-order part of dividend
MVI H,00        // Load H register with 00H
LDA 4002        // Load Accumulator with Divisor
MOV B,A         // Load B with Divisor
MVI C,08        // Load counter with 8 for 8-bit division
CALL START      // Loop to divide high-order dividend with divisor
LDA 4001
    MOV L,A     // Load L register with Low-order part of dividend
    MVI C,08    // Load counter with 8 for 8-bit division
    XCHG
    DAD H       // Shifting Quotient towards left by one-bit
                   before next division
    XCHG
    CALL START  // Loop to divide high-order dividend with divisor
    MOV A,H
    STA 4006    // Load memory location 4006H with Remainder
    XCHG
    SHLD 4003   // Load memory location 4003H with low-order
                   Quotient and 4004H with High-order Quotient
    HLT

// Loop to perform 8-bit with 8-bit division
START:
    DAD H// Shift HL register pair towards left by 1
    MOV A,H
    SUB B
    CNC INCREMENT   // If Accumulator >= Divisor go to INCREMENT
    DCR C
    RZ
    XCHG
    DAD H           // Shift DE register pair towards left by 1
    XCHG
    JMP START
INCREMENT:
    MOV H,A
    INX D
    RET
# ORG 4000H
// Dividend: C2C7H, Divisor: 09H
# DB C2H, C7H,09H
```

Screen-grab of Simulator:

8085 Simulator

File Edit Tools Settings Simulation Subroutine View Load Sample Program Help

Editor | Assembler

8085 Assembly Language Editor

Assembler | Disassembler

```
# ORG 5000H

INITIALIZE:   LDA 4000
              MOV L,A     // Load L register with High-order part of
dividend
              MVI H,00    // Load H register with 00H
              LDA 4002    // Load Accumulator with Divisor
              MOV B,A     // Load B with Divisor
              MVI C,08    // Load counter with 8 for 8-bit division
              CALL START            // Loop to divide high-order
dividend with divisor
              LDA 4001
              MOV L,A     // Load L register with Low-order part of
dividend
              MVI C,08    // Load counter with 8 for 8-bit division
              XCHG
              DAD H       // Shifting Quotient towards left by one-bit
before next division
              XCHG
              CALL START            // Loop to divide high-order
dividend with divisor
              MOV A,H
              STA 4006    // Load memoery location 4006H with
Remainder
              XCHG
              SHLD 4003 // Load memoery location 4003H with low-order
Quotient and 4004H with High-order Quotient
              HLT

// Loop to perform 8-bit with 8-bit division
START:        DAD H       // Shift HL register pair towards left by 1
              MOV A,H
              SUB B
              CNC INCREMENT         // I f Accumulator >= Divisor go to
INCREMENT
              DCR C
              RZ
              XCHG
              DAD H       // Shift DE register pair towards left by 1
              XCHG
              JMP START

INCREMENT:    MOV H,A
              INX D
              RET
# ORG 4000H

// Dividend: C2C7H, Divisor: 09H
# DB C2H,C7H,09H
```

Autocorrect | Assemble

Registers | Memory | Devices

Registers :

Register	Value	7	6	5	4	3	2	1	0
Accumulator	03	0	0	0	0	0	0	1	1
Register B	09	0	0	0	0	1	0	0	1
Register C	00	0	0	0	0	0	0	0	0
Register D	03	0	0	0	0	0	0	1	1
Register E	00	0	0	0	0	0	0	0	0
Register H	15	0	0	0	1	0	1	0	1
Register L	A4	1	0	1	0	0	1	0	0
Memory(M)	00	0	0	0	0	0	0	0	0

Resister	Value	S	Z	*	AC	*	P	*	CY
Flag Resister	55	0	1	0	1	0	1	0	1

Type	Value
Stack Pointer(SP)	0000
Memory Pointer (HL)	15A4
Program Status Word(PSW)	0355
Program Counter(PC)	5023
Clock Cycle Counter	1480
Instruction Counter	189

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

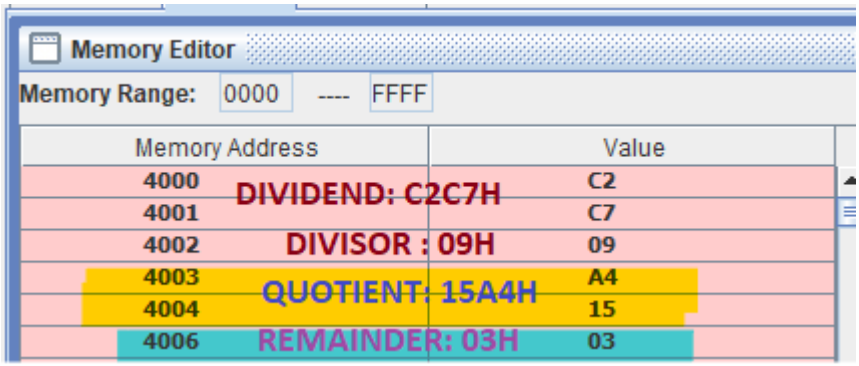
For RIM instruction

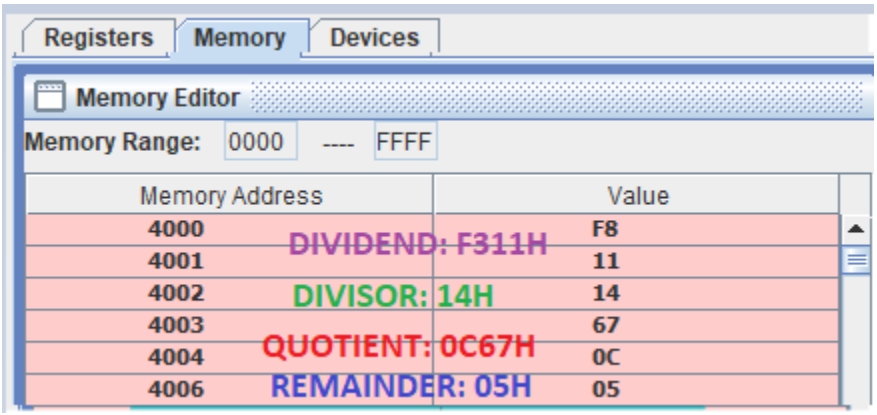
SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

No. Converter Tool :

Hexadecimal	Decimal	Binary
9	9	00001001

Result:





Discussion:

In this program, I implemented 16-bit with 8-bit division using DAD H instruction. In the process of comparing H register with Divisor, I observed a bug in the simulator, the instruction sets CMP and CPI are setting carry flag to 1 instead of 0 on comparing accumulator with 00H. For this reason, I avoided the use of compare instruction and instead used SUB instruction which was working fine. In the last class Hasan sir discussed the use of DSUB instruction for division, however the simulator doesn't support the use of this instruction set, therefore I implemented the algorithm using SUB instruction only.