First Semester 2022/2023

COMPUTER ORGANIZATION AND MICROPROCESSOR

Simple computer project

Dr. Abualseoud Hanani

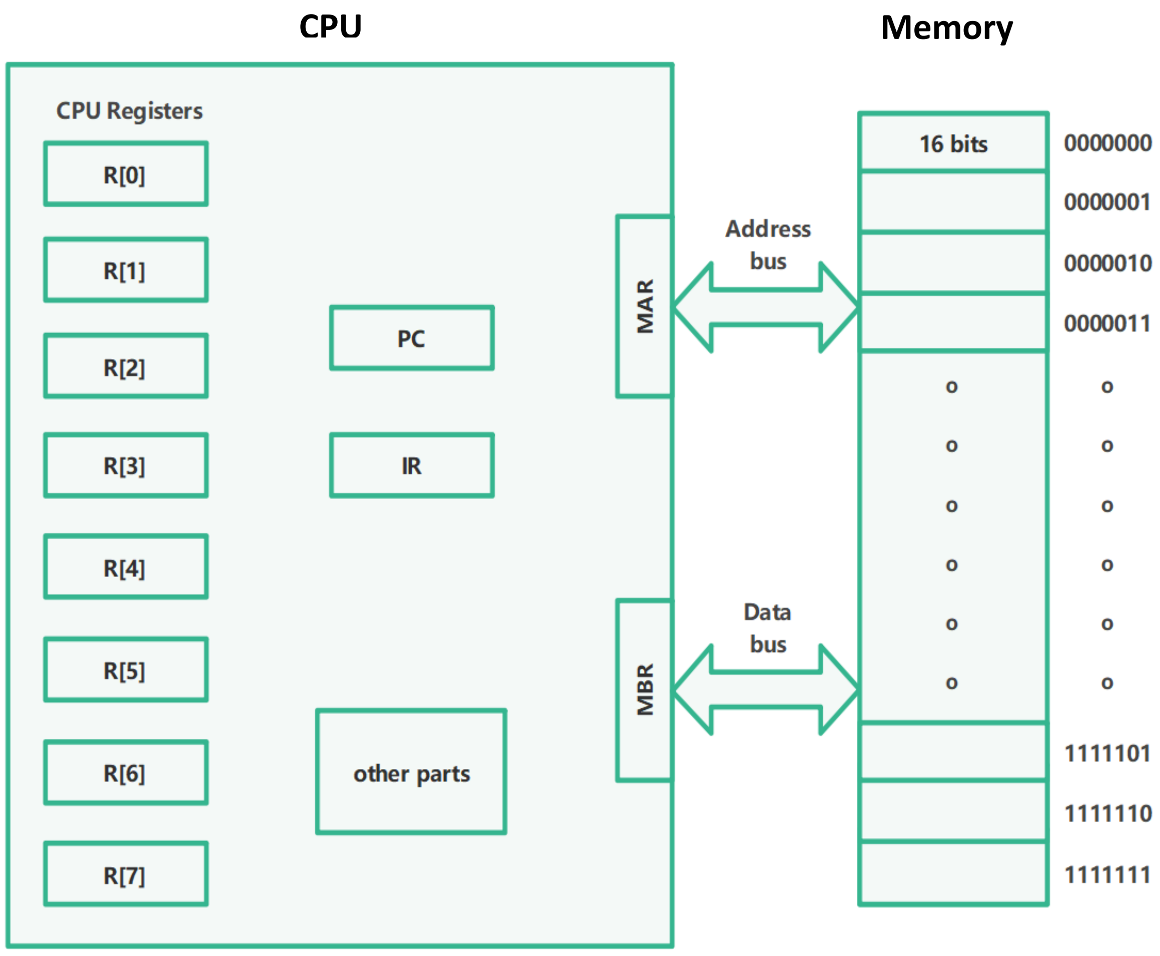
Section (2)

QOSSAY RIDA (1211553)

Contents:

Introduction about the project:

In this project , we implement simple computer by using Verilog language , this computer has memory with size 128 cells as shown in the figure below and each cell has a length of 16 bits (2 byte), and has CPU contains program counter (PC), instruction register (IR) , memory address register (MAR), memory buffer register (MBR) for can accessed to memory and set of registers R[0] to R[7] (general purpose registers), the memory is synchronous to the CPU, and the CPU can read/write one cell in a single clock cycle.



The CPU contains Arithmetic Logic Unit & Floating Point Unit, but we don’t implement this unit in project.

Instruction format for this computer:

1- Instruction format

15 13	12 10	9 8	7 0
Opcode #3bits	Dst. Register #3bits	Mode #2bits	Constant, Src. Register / Memory Address #8bits

[15:13] : Using this code To specify the operation, all operations will be explained on the next page

[12:10] : Using this code To specify the destination register (1^{st}) operand, this computer has 8 register has code from 000 to 111

[9:8] : Using this code To specify the Addressing Mode for second operand as follow :

00 ->	**Direct memory address**
01 ->	**Register**
10 ->	**Register Indirect**
11 ->	**Constant**

[7:0] : Using this code To specify the (2^{nd}) operand after look at mode bit

2- Sign-magnitude format

Sign #1bit	Magnitude #15bits

This computer use sign-magnitude method to store a integer in 16-bits as shown in the figure above

>> sign-bit=0 then negative integer
>> sign-bit=1 then positive integer

Range for integer:

$$-2^{15}+1 \qquad to \qquad 2^{15}-1$$

Operations that this computer can perform:

Load operation (000): this operation will load the value from (Memory cell , Register, Register Indirect, Constant) to Destination Register.

Store operation (001): this this operation will store the value of Register in the memory (just has mode equal to 00 because it just store to memory directly).

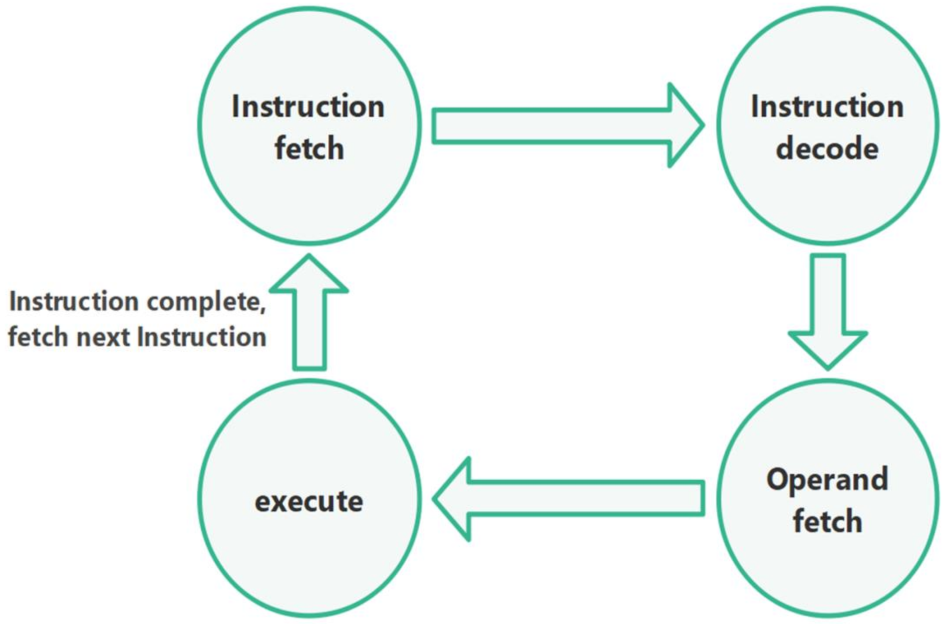
Add operation (010): Add the value of Register to the operand and store the result at Register.

Sub operation (011): Subtract the value of Register to the operand and store the result at Register.

Mul operation (100): Multiplies the value of Register to the operand and store the result at Register.

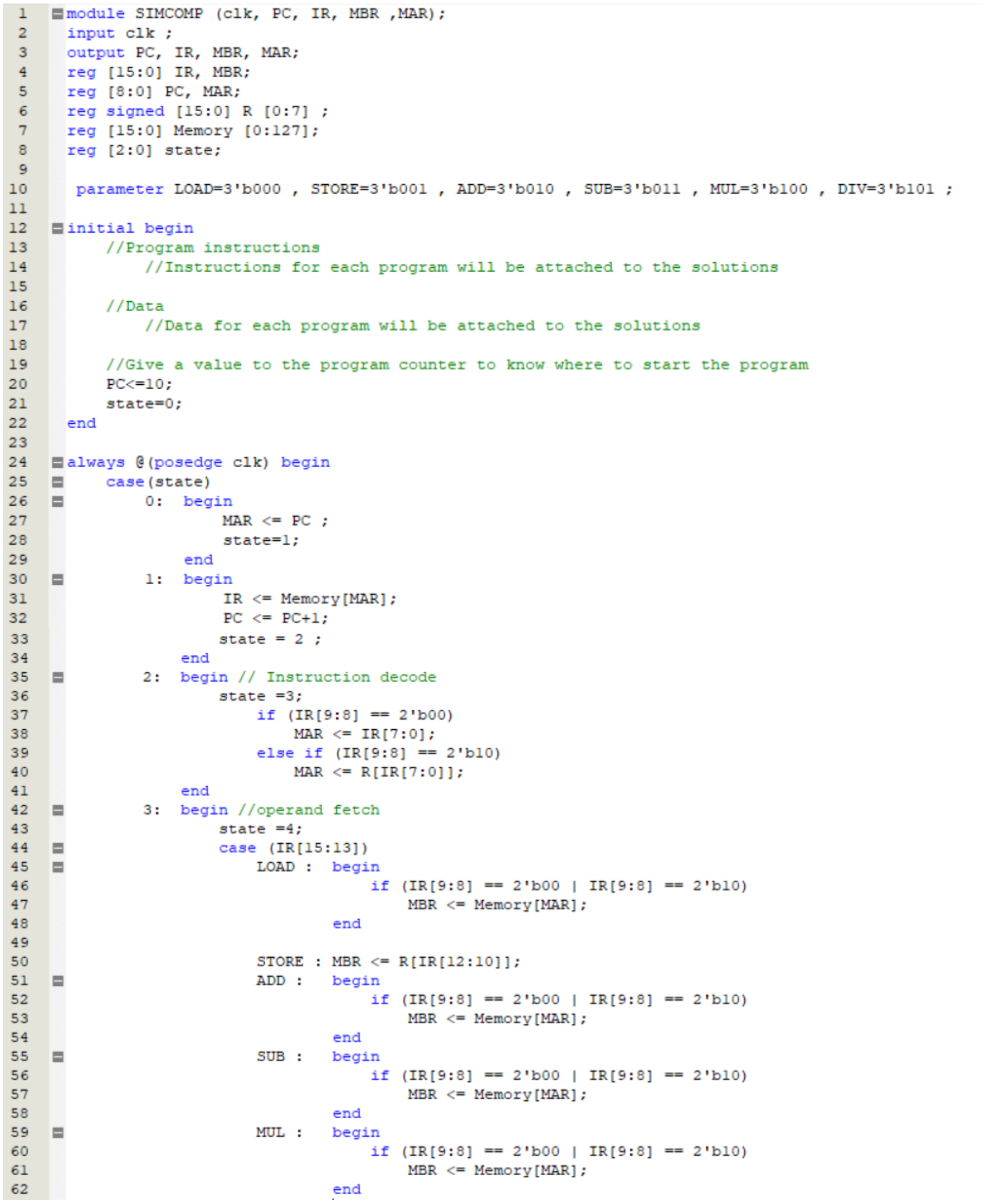
Div operation (101): Divides the value of Register to the operand and store the result at Register.

Instruction Cycle State Diagram:

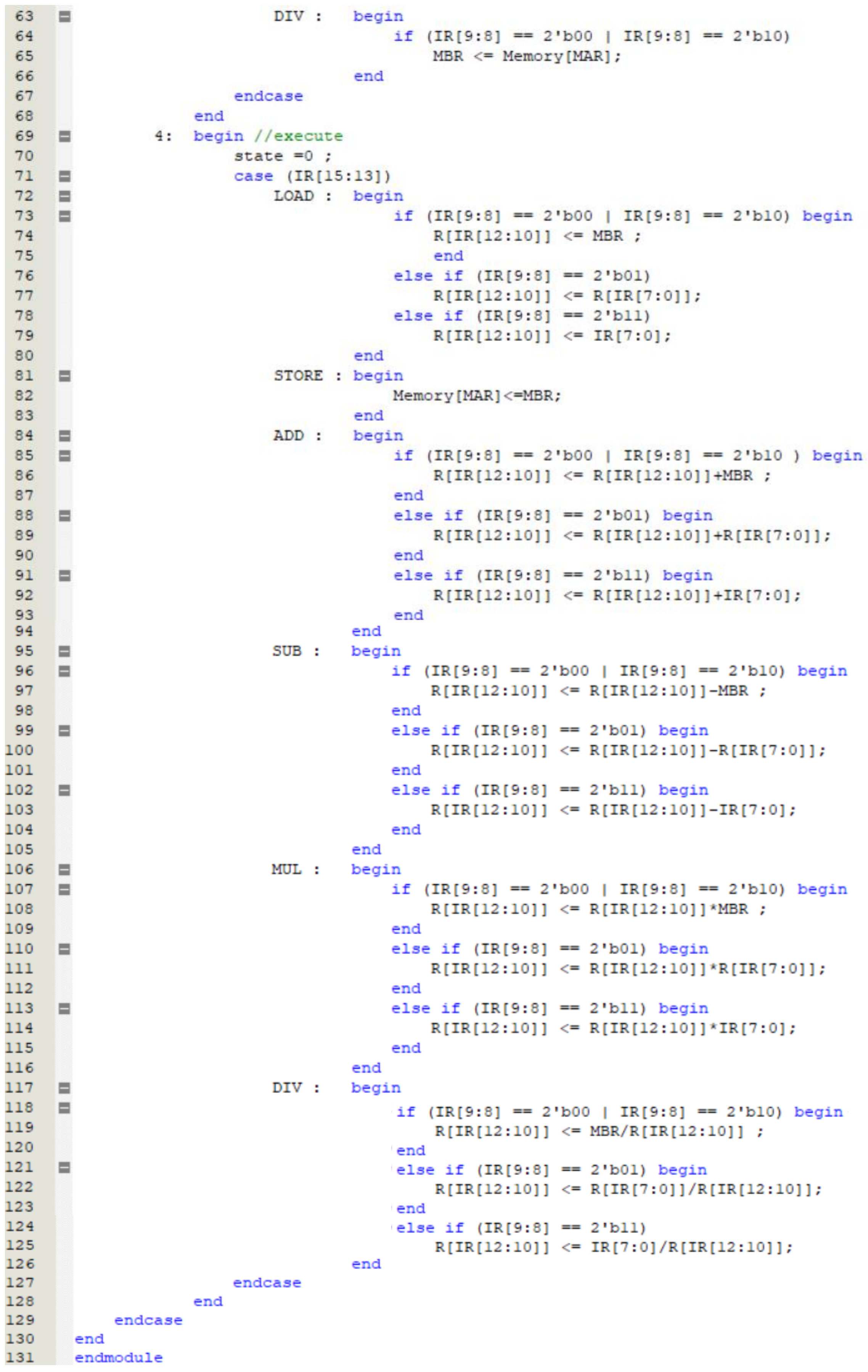


The code in Verilog:

Instructions and Data for each program will be attached to the solutions



```verilog
module SIMCOMP (clk, PC, IR, MBR ,MAR);
input clk ;
output PC, IR, MBR, MAR;
reg [15:0] IR, MBR;
reg [8:0] PC, MAR;
reg signed [15:0] R [0:7] ;
reg [15:0] Memory [0:127];
reg [2:0] state;

 parameter LOAD=3'b000 , STORE=3'b001 , ADD=3'b010 , SUB=3'b011 , MUL=3'b100 , DIV=3'b101 ;

initial begin
    //Program instructions
        //Instructions for each program will be attached to the solutions

    //Data
        //Data for each program will be attached to the solutions

    //Give a value to the program counter to know where to start the program
    PC<=10;
    state=0;
end

always @(posedge clk) begin
    case(state)
        0:  begin
                MAR <= PC ;
                state=1;
            end
        1:  begin
                IR <= Memory[MAR];
                PC <= PC+1;
                state = 2 ;
            end
        2:  begin // Instruction decode
                state =3;
                    if (IR[9:8] == 2'b00)
                        MAR <= IR[7:0];
                    else if (IR[9:8] == 2'b10)
                        MAR <= R[IR[7:0]];
            end
        3:  begin //operand fetch
                state =4;
                case (IR[15:13])
                    LOAD :  begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10)
                                    MBR <= Memory[MAR];
                            end

                    STORE : MBR <= R[IR[12:10]];
                    ADD :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10)
                                    MBR <= Memory[MAR];
                            end
                    SUB :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10)
                                    MBR <= Memory[MAR];
                            end
                    MUL :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10)
                                    MBR <= Memory[MAR];
                            end
```



```
                    DIV :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10)
                                    MBR <= Memory[MAR];
                            end
                endcase
            end
        4:  begin //execute
                state =0 ;
                case (IR[15:13])
                    LOAD :  begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10) begin
                                    R[IR[12:10]] <= MBR ;
                                    end
                                else if (IR[9:8] == 2'b01)
                                    R[IR[12:10]] <= R[IR[7:0]];
                                else if (IR[9:8] == 2'b11)
                                    R[IR[12:10]] <= IR[7:0];
                            end
                    STORE : begin
                                Memory[MAR]<=MBR;
                            end
                    ADD :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10 ) begin
                                    R[IR[12:10]] <= R[IR[12:10]]+MBR ;
                                end
                                else if (IR[9:8] == 2'b01) begin
                                    R[IR[12:10]] <= R[IR[12:10]]+R[IR[7:0]];
                                end
                                else if (IR[9:8] == 2'b11) begin
                                    R[IR[12:10]] <= R[IR[12:10]]+IR[7:0];
                                end
                            end
                    SUB :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10) begin
                                    R[IR[12:10]] <= R[IR[12:10]]-MBR ;
                                end
                                else if (IR[9:8] == 2'b01) begin
                                    R[IR[12:10]] <= R[IR[12:10]]-R[IR[7:0]];
                                end
                                else if (IR[9:8] == 2'b11) begin
                                    R[IR[12:10]] <= R[IR[12:10]]-IR[7:0];
                                end
                            end
                    MUL :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10) begin
                                    R[IR[12:10]] <= R[IR[12:10]]*MBR ;
                                end
                                else if (IR[9:8] == 2'b01) begin
                                    R[IR[12:10]] <= R[IR[12:10]]*R[IR[7:0]];
                                end
                                else if (IR[9:8] == 2'b11) begin
                                    R[IR[12:10]] <= R[IR[12:10]]*IR[7:0];
                                end
                            end
                    DIV :   begin
                                if (IR[9:8] == 2'b00 | IR[9:8] == 2'b10) begin
                                    R[IR[12:10]] <= MBR/R[IR[12:10]] ;
                                end
                                else if (IR[9:8] == 2'b01) begin
                                    R[IR[12:10]] <= R[IR[7:0]]/R[IR[12:10]];
                                end
                                else if (IR[9:8] == 2'b11)
                                    R[IR[12:10]] <= IR[7:0]/R[IR[12:10]];
                            end
                endcase
            end
    endcase
end
endmodule
```

Requirements solutions:

>> Question 1

A- How many instructions this machine can support?

This machine has 8 kind of instruction because the opcode has 3 bit

B- What is the range of unsigned and signed constant numbers this machine supports?

$$-2^{15}+1 \qquad to \qquad 2^{15}-1$$

Because it represent in sign-magnitude way

C- What is the length (in bits) of the following registers in this machine?

PC: 8 bits

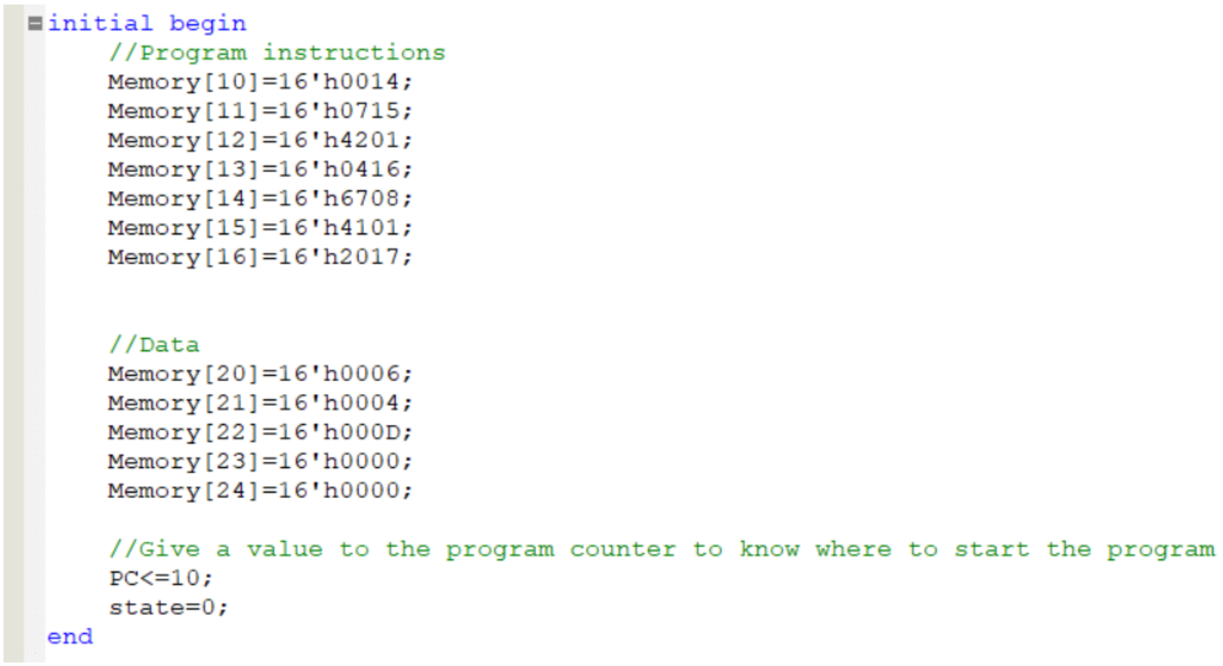
IR: 16 bits

MAR: 8 bits

MBR: 16 bits

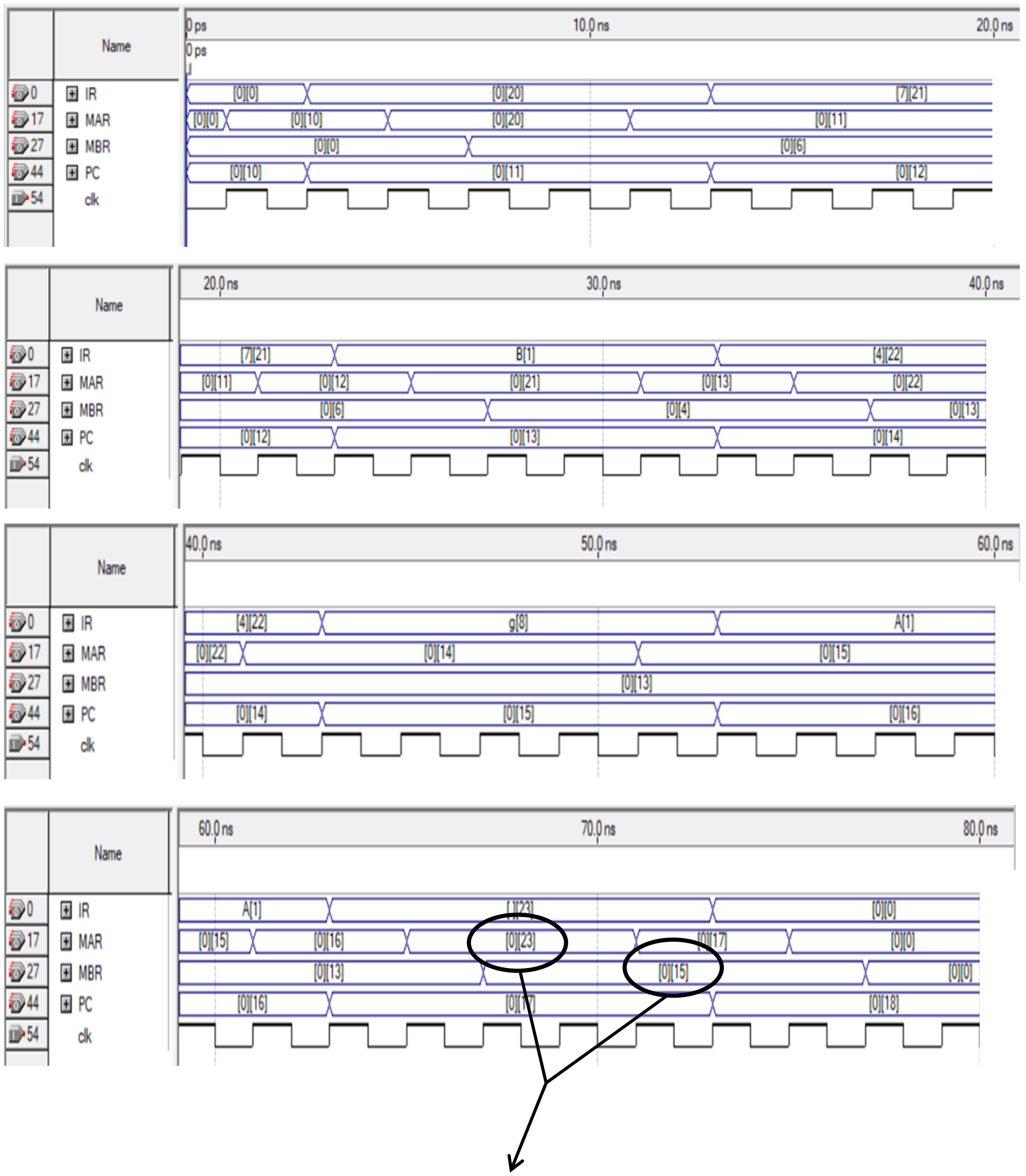
>> Question 2

Instructions and Data for this program:



```
initial begin
     //Program instructions
     Memory[10]=16'h0014;
     Memory[11]=16'h0715;
     Memory[12]=16'h4201;
     Memory[13]=16'h0416;
     Memory[14]=16'h6708;
     Memory[15]=16'h4101;
     Memory[16]=16'h2017;


     //Data
     Memory[20]=16'h0006;
     Memory[21]=16'h0004;
     Memory[22]=16'h000D;
     Memory[23]=16'h0000;
     Memory[24]=16'h0000;

     //Give a value to the program counter to know where to start the program
     PC<=10;
     state=0;
end
```

Simulation waveform:



When this computer executes the instruction in Memory [16], will store Register[1] to Memory[23], by send the value of register to MBR then store

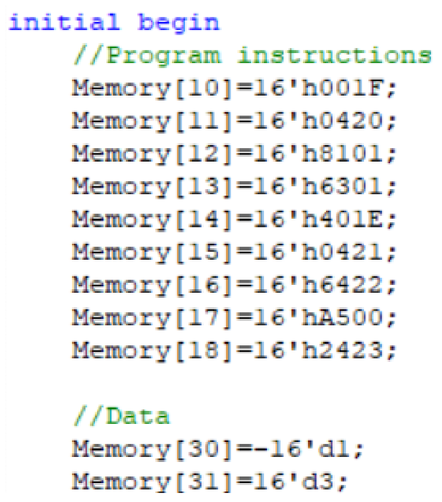
It to the memory, from simulation the value will store to Memory[23] will equal 15 .

Memory Address	Contents	R0	R1
10	Load R0, [20] (instruction)	6	
11	Load R1,21 (instruction)	6	21
12	Add R0, [R1] (instruction)	10	21
13	load R1, [22] (instruction)	10	13
14	Sub R1, +8 (instruction)	10	5
15	Add R0,R1 (instruction)	15	5
16	Store R0, [23] (instruction)	15	5
…..			
20	6 (Data)		
21	4 (Data)		
22	13 (Data)		
23	0 (Data)	Will be 15	

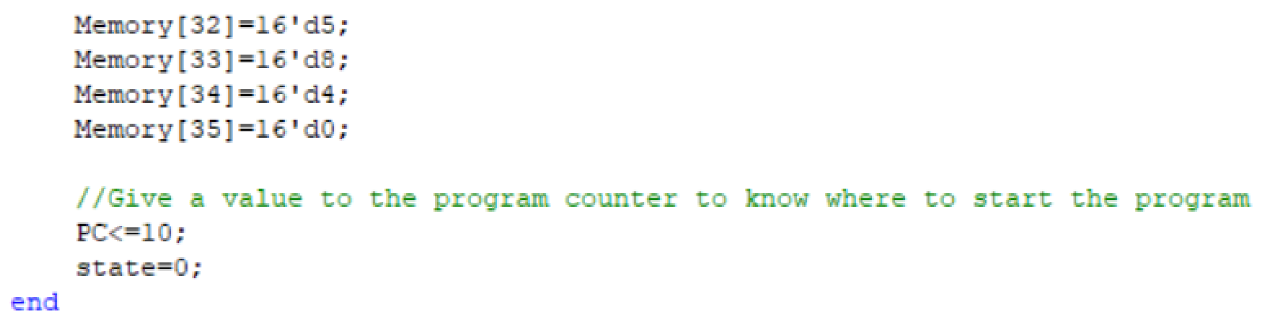
After executing the previous instructions theoretically, we find that the value in Memory[23] is equal to 15, and equal the value will be in Memory[23] experimentally

>> Question 3

Instructions and Data for this program:



```
initial begin
    //Program instructions
    Memory[10]=16'h001F;
    Memory[11]=16'h0420;
    Memory[12]=16'h8101;
    Memory[13]=16'h6301;
    Memory[14]=16'h401E;
    Memory[15]=16'h0421;
    Memory[16]=16'h6422;
    Memory[17]=16'hA500;
    Memory[18]=16'h2423;

    //Data
    Memory[30]=-16'd1;
    Memory[31]=16'd3;
```



```
        Memory[32]=16'd5;
        Memory[33]=16'd8;
        Memory[34]=16'd4;
        Memory[35]=16'd0;

        //Give a value to the program counter to know where to start the program
        PC<=10;
        state=0;
    end
```

Equation:

$$Y = \frac{A+B*C-1}{D-E}$$

Solution:

>> Write assembly code for implementing the above arithmetic expression?

The assembly code

Memory Address	Contents	R0	R1
10	LODE R0 , [31] ;	3	
11	load R1 , [32]	3	5
12	MUL R0 , R1 ;	15	5
13	SUB R0 , 1 ;	14	5
14	ADD R0 , [30] ;	13	5
15	LODE R1 , [33] ;	13	8
16	SUB R1 , [34] ;	13	4
17	DIV R1 ,R0 ;	13	3
18	STORE R1,[35] ;	13	3
……			
30	-1		
31	3		
32	5		
33	8		
34	4		
35	0	Will be 3	

After executing the previous instructions theoretically, we find that the value in Memory[35] is equal to 3 .

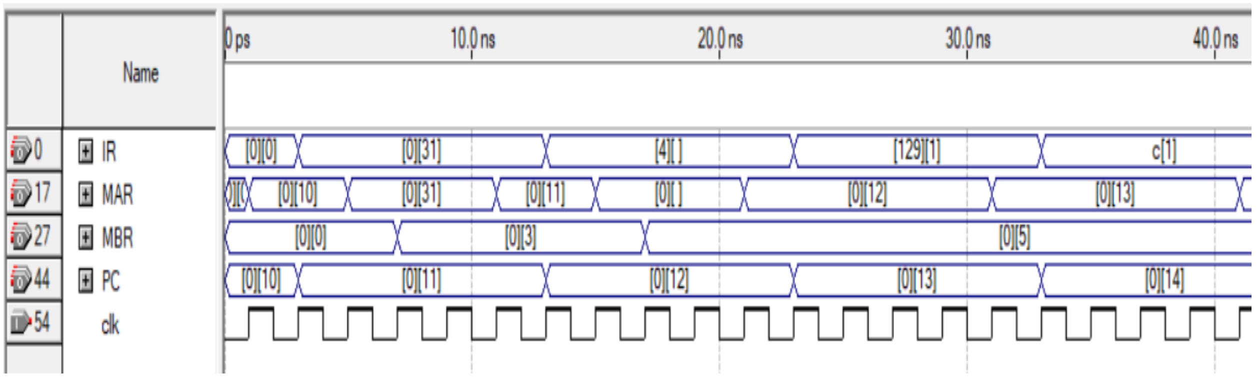
>> Convert the above assembly instructions into machine code and store them in the memory starting at address 10.

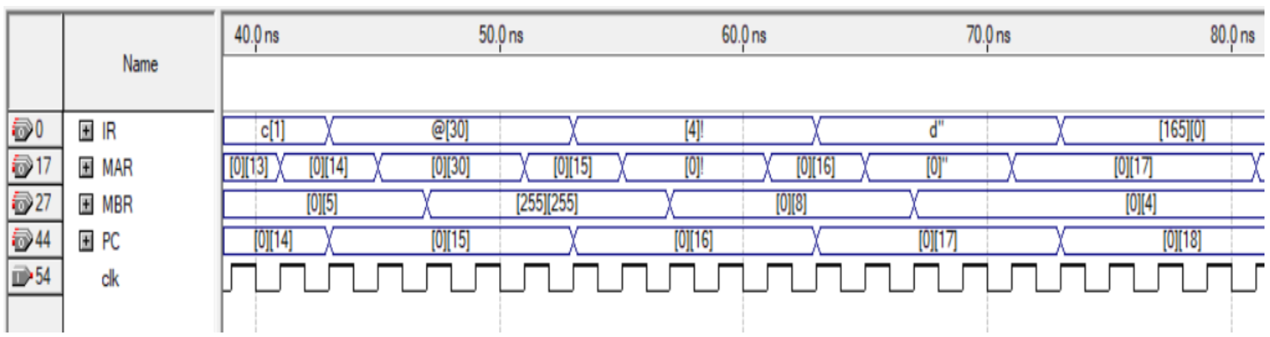
Memory Address	Contents
10	0000 0000 0001 1111
11	0000 1000 0100 0000
12	1000 0001 0000 0001
13	0110 0011 0000 0001
14	0100 0000 0001 1110
15	0000 0100 0010 0001
16	0110 0100 0010 0010
17	1010 0101 0000 0000
18	0010 0100 0010 0011

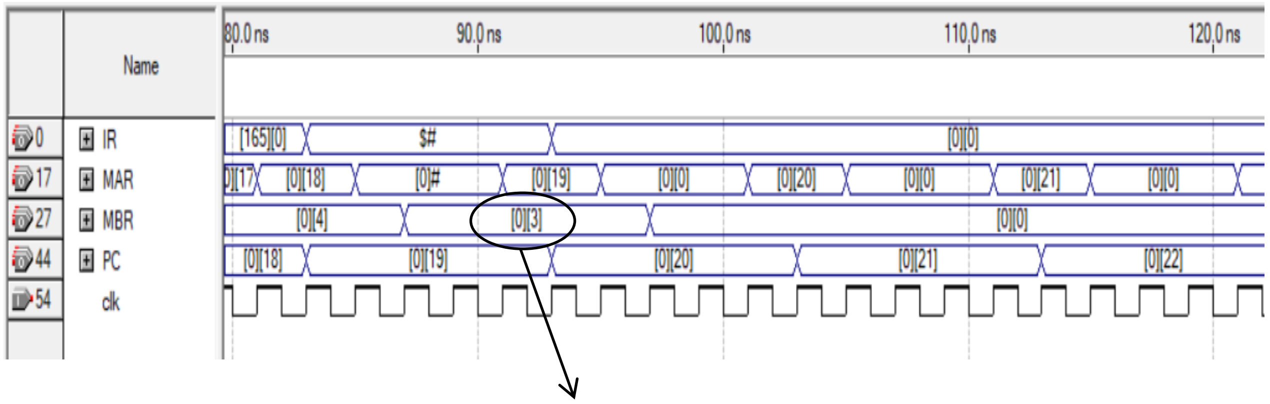
Memory Address	Contents
30	1111 1111 1111 1111
31	0000 0000 0000 0011
32	0000 0000 0000 0101
33	0000 0000 0000 1000
34	0000 0000 0000 0100
35	0000 0000 0000 0000

>> Set PC=10 and simulate the above program. Verify that it works correctly and the result stored at memory variable Y is correct. Attach simulation waveform and the Verilog source file. Assume A, B, C, D and E have the values -1, 3, 5, 8, and 4, respectively.

Simulation waveform:







When this computer executes the instruction in Memory [18], will store Register[1] to Memory[35], by send the value of register to MBR then store It to the memory, from simulation the value will store to Memory[35] will equal 3.