Electrical and Computer Engineering

ENCS2380 – Computer Organization and Microprocessors - Spring 2023

Course Project (I)

Team:

- Name: Qusay Taradeh
  ID: 1212508
  Instructor: Dr. Abualseoud Hanani
  Section: 1

- Name: Mosa Sbeih
  ID: 1211250
  Instructor: Dr. Abualseoud Hanani
  Section: 2

Contribution:

The team worked each module and task together.

Simulation (1)

Memory Address	Content	Content interpretation
0	0x180A	LOAD [10]
1	0X580B	MUL [11]
2	0X3005	ADD 5
3	0X280C	STORE [12]
10	0X0009	9
11	0XFFFC	-4
12	0X0000	0

1) Load the value of Memory[10] = 9 to AC.

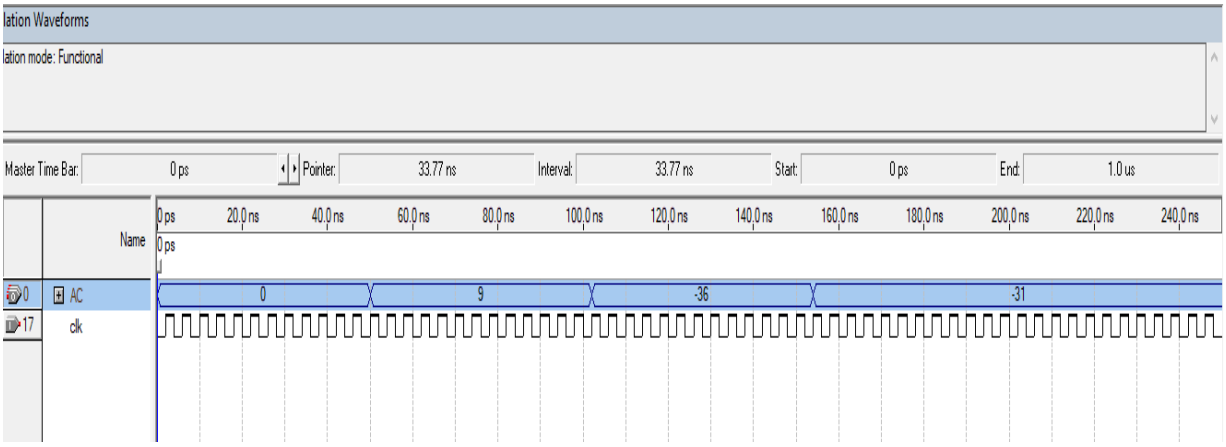
2) Multiply AC by the value of Memory[11]

= -4, then AC = -36

3) Add 5 to AC, then AC = -31

4) Store the value of AC to Memory[12], then Memory[12] = -31, instead of 0

WaveForm Simulation of the above program:



Code of RAM:

```
1   module RAM(input clk, input [10:0] address, input [15:0] data_in, input rd, input wr, output reg[15:0] data_out);
2
3    reg[15:0] Memory[0:63];
4
5   initial begin
6     Memory[0] = 16'h180A;
7     Memory[1] = 16'h580B;
8     Memory[2] = 16'h3005;
9     Memory[3] = 16'h280C;
10
11    Memory[10] = 16'h0009;
12    Memory[11] = 16'hFFFC;
13    Memory[12] = 16'h0000;
14   end
15
16  always @(posedge clk) begin
17   if (rd) begin
18     data_out <= Memory[address];
19    end
20
21   else if (wr) begin
22     Memory[address] <= data_in;
23    end
24   end
25
26
27  endmodule
28
```

Simulation (2)

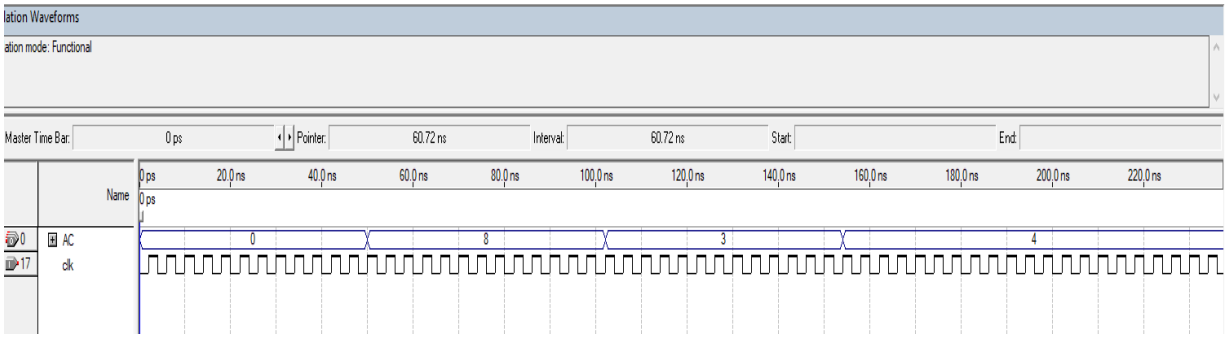
$Y=(A+B*C-5)/(D+E+1)$

Address	Content	Assembly
0	0x1817	Load [23]
1	0x3818	Add [24]
2	0x3001	Add 1
3	0x2819	Store [25]
4	0x1815	Load [21]
5	0x5816	Mul [22]
6	0x4005	Sub 5
7	0x3814	Add [20]
8	0x6819	Div [25]
9	0x2819	Store [25]
10	0x1819	Load [25]
20 (A)	0x0002	2
21 (B)	0x0003	3
22 (C)	0x0005	5
23 (D)	0x0008	8
24 (E)	0xFFFB	-5
25 (Y)	0x0000	0

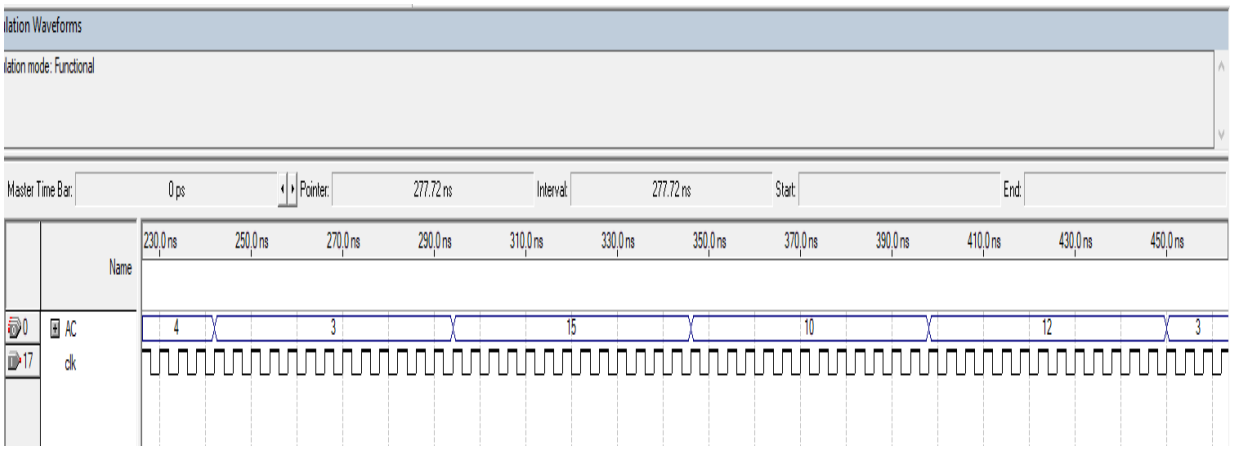
1) Load value of Memory[23] = 8 to AC
2) Add value of Memory[24] = -5 to AC , then AC = 3
3) Add 1 to AC, then AC = 4
4) Store AC to Memory[25]
5) Load value of Memory[21] = 3 to AC
6) Multiply the value of Memory[22] = 5 by AC,
   then AC = 15
7) Subtract 5 from AC, then AC = 10
8) Add value of Memory[20] = 2 to AC,
   then AC = 12
9) Divide AC by the Memory[25] = 4,
   then AC = 3
10) Store AC into Memory[25], the Memory[25] = 3

WaveForm Simulation of the above program:

1)



2)



Code of RAM:

```
module RAM(input clk, input [10:0] address, input [15:0] data_in, input rd, input wr, output reg[15:0] data_out);

 reg[15:0] Memory[0:63];

 initial begin
  Memory[0] = 16'h1817; // Load D
  Memory[1] = 16'h3818; // Add E
  Memory[2] = 16'h4001; // Sub 1
  Memory[3] = 16'h2819; // Store Y
  Memory[4] = 16'h1815; // Load B
  Memory[5] = 16'h5816; // Mul C
  Memory[6] = 16'h4005; // Sub 5
  Memory[7] = 16'h3814; // Add A
  Memory[8] = 16'h6819; // Div Y
  Memory[9] = 16'h2819; // Store Y
  Memory[10] = 16'h1819; // Load Y

  Memory[20] = 16'h0002; // A
  Memory[21] = 16'h0003; // B
  Memory[22] = 16'h0005; // C
  Memory[23] = 16'h0008; // D
  Memory[24] = 16'hFFFB; // E
 end

 always @(posedge clk) begin
  if (rd) begin
   data_out <= Memory[address];
  end

   else if (wr) begin
    Memory[address] <= data_in;
   end
 end


endmodule
```

```
module CPU(clk, memoryData, memoryAddress, dataOut, readSignal, writeSignal, AC);

input clk;
input [15:0] memoryData;
reg [3:0] state;
reg [5:0] PC;
reg [10:0] MAR;
reg signed [15:0] MBR;
reg [3:0] opcode;
reg M;
reg [15:0] IR;
reg zeroFlag, overflowFlag, carryFlag, negativeFlag;
output reg readSignal = 1;
output reg writeSignal = 0;
output reg signed [15:0] AC;
output reg [15:0] dataOut;
output reg [10:0] memoryAddress;
reg signed [15:0] result;

parameter Load = 4'd1, Store = 4'd2, Add = 4'd3, Sub = 4'd4, Mul = 4'd5, Div = 4'd6 , Branch = 4'd7, BRZ = 4'd8;

/*
 =============Instrucion Format==============
                    "16-bit"
    [15:12]  [11]          [10:0]
 ____________ _____ ____________________________
|    4-bit   | 1-bit |          11-bit           |
|     Op     |   M   |1-Memory Address/0-Constant|
|____________|_______|___________________________|

*/

initial begin
    PC = 0;
    state = 0;
    AC = 0;
    dataOut = 0;
    zeroFlag = 0;
    overflowFlag = 0;
    carryFlag = 0;
    negativeFlag = 0;
end
```

Declaration of input/output and registers

Initializing relevant variables.

```

always @ (posedge clk) begin
    case (state)
        0: begin // intial state
            MAR <= PC;
            state = 1;
        end
  /*****************************************************************************/
        1: begin // fetching from memory
            writeSignal = 0;
            readSignal = 1;
            memoryAddress <= MAR;
            PC <= PC + 1;
            state = 2;
        end
  /*****************************************************************************/
            2:state = 3; // waiting for next clock
  /*****************************************************************************/
        3: begin
            MBR <= memoryData;
            state = 4;
        end
  /*****************************************************************************/
        4: begin
            IR <= MBR;
            state = 5;
        end
  /*****************************************************************************/
        5: begin // decoding
            opcode <= IR[15:12];
            M <= IR[11];
            MAR <= IR[10:0];
            state = 6;
        end
```

Always statement that depends on positive edge clk

State 1: Sending address of first inst. to MAR

State 2: Waiting for next clock

State 3: Retrieving Data from memory to MBR in CPU

State 4: Putting the value of MBR to the IR register

State 5: IR decoding the instruction

```
78  /*****************************************************************************/
79          6: begin
80                  if (opcode == Branch) begin // Branch operation
81                      PC <= MAR;
82                      state = 0;
83                  end
84                  else if (opcode == BRZ) begin // BRZ operation if zero flag == 1
85                      if(zeroFlag == 1 && M == 1) begin
86                          PC <= MAR;
87                          state = 0;
88                      end
89                      else begin
90                          state = 0;
91                      end
92                  end
93                  else begin
94                      state = 7;
95                  end
96          end
97  /*****************************************************************************/
98          7: begin
99              if (opcode == Store) begin
100                 memoryAddress <= MAR;
101             end
102             state = 8;
103         end
104 /*****************************************************************************/
105         8: begin
106             if (opcode == Store) begin //store operation
107                 writeSignal = 1;
108                 readSignal = 0;
109                 dataOut <= AC;
110                 state = 0;
111             end
112             else begin
113                 state = 9;
114             end
115         end
116 /*****************************************************************************/
117         9: begin
118             writeSignal = 0;
119             readSignal = 1;
120             if(M == 1) begin
121                 memoryAddress <= MAR;
122             end
123             state = 10;
124             end
125 /*****************************************************************************/
```

State 6: Checking if opcode is Branch or BRZ before executing.

State 7: Sending the address to Memory if opcode is Store

State 8: Executing the Store operation if opcode is Store

State 9: Sending the Address of operand to memory if it’s not a constant

```
126         10: begin
127             state = 11;
128         end
129  /*****************************************************************************/
130         11: begin
131             if (M == 1) begin
132                 MBR <= memoryData;
133             end
134             else if (M == 0) begin
135                 MBR <= $signed(IR[10:0]);
136             end
137             state = 12;
138         end
139  /*****************************************************************************/
140         12: begin // executing
141             if (opcode == Load) begin //load operation
142                 AC <= MBR;
143                 state = 0;
144             end
145             else begin
146                 case(opcode)
147                 Add : // Add operation
148                     begin
149                     {carryFlag, result} = AC + MBR;
150                     zeroFlag = (result==0)? 1:0;
151                     if(AC[15]==MBR[15]) begin
152                       if(MBR[15]==result[15])begin
153                       overflowFlag = 0;
154                       end
155                       else begin
156                       overflowFlag = 1;
157                       end
158                     end
159                     else begin
160                     overflowFlag = 0;
161                     end
162                     negativeFlag =(result[15]==1)? 1:0;
163                     end
164
```

State 10: waiting for next clk

State 11: if operand is in memory set MBR to that data in memory, else set MBR to the constant operand

State 12: if opcode is Load , load the MBR to AC

If opcode is Arithmetic i.e. Add, Sub, Mul and Div.

calculate the operation and set it to AC and calculate the flags

After executing the opcode state is set to 0

```
164
165                 Sub : // Sub operation
166                     begin
167                     {carryFlag, result} = AC - MBR;
168                     zeroFlag = (result==0)? 1:0;
169                     if(AC[15]==MBR[15]) begin
170                       if(MBR[15]==result[15])begin
171                       overflowFlag = 0;
172                       end
173                       else begin
174                       overflowFlag = 1;
175                       end
176                     end
177                     else begin
178                     overflowFlag = 0;
179                     end
180
181                     negativeFlag =(result[15]==1)? 1:0;
182                     end
183
184                 Mul : // Mul operation
185                     begin
186                     result = AC * MBR;
187
188                     zeroFlag = (result==0)? 1:0;
189
190                     if(result > 16'h0FFF || result < 16'hF000) begin
191                     overflowFlag = 1;
192                     end
193
194                     else begin
195                         overflowFlag = 0;
196                     end
197
198                     negativeFlag =(result[15]==1)? 1:0;
199                     end
200
201                 Div : // Div operation
202                     begin
203                     result = AC / MBR;
204                     zeroFlag = (result==0)? 1:0;
205                     negativeFlag =(result[15]==1)? 1:0;
206                     end
207
208             endcase //end of arithmetic
209             AC <= result;
210             state = 0;
211             end
212         end
213     endcase
214 end
```

The above is the code of the module CPU

THE END