COMPUTER ARCHITECTURE

Dr. Deyaa Ghazi

Assignment1: Memory, Registers, and ALU implementation

Team # 5

Moath Yousef AL-Tahrawi	*2141099*
Omar Tareq AL-Shamasneh	*2139919*
Mohammed AL-Tamimi	*2042979*
Abdullah Nasha'at	*1937045*
Ahmed Maen Barham	*2330132*

Introduction

This project involves designing a functioning computer architecture, by implementing a simplified computer system in Verilog.

In the 1st assignment, our objective is to implement a memory module, register modules, and an ALU that has multiple instructions.

Using Verilog simulation in modelsim, we will showcase the outputs in multiple scenarios to ensure the system is working properly and accurately as it should.

Memory implementation:

Memory unit code:

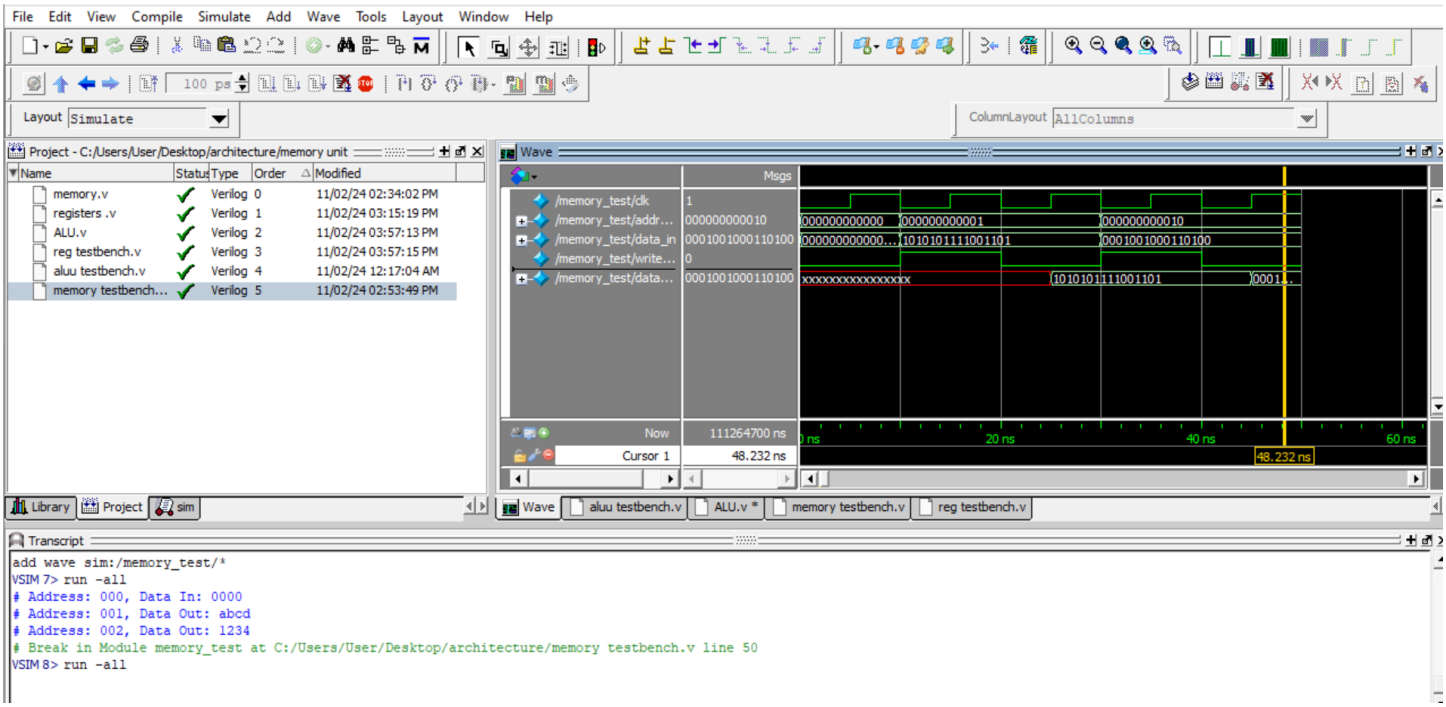
```
module memory
    #(
    parameter A = 12,
    parameter m = 16
    ) // parameters help editing the code if you want to change some values
    (
    input clk,
    input [A-1:0] address,
    input [m-1:0] data_in,
    input write_enable,
    output reg [m-1:0] data_out
    ); // the i/p and o/p in the memory unit

    localparam D = (1 << A);
    reg [m-1:0] mem [0:D-1];
    // parameterized memory array

    always @(posedge clk) begin
      if(address !=0) begin
        if (write_enable)
            mem[address] <= data_in; // write operation
        else
            data_out <= mem[address]; //read operation
    end
  end
endmodule
```

Memory unit testbench:

```
`timescale 1ns / 1ps
module memory_test();
  parameter A = 12;
  parameter m = 16;

  reg clk;
  reg [A-1:0] address;
  reg [m-1:0] data_in;
  reg write_enable;
  wire [m-1:0] data_out;

  // Instantiate the memory module
  memory memory_test_inst (
    .clk(clk),
    .address(address),
    .data_in(data_in),
    .write_enable(write_enable),
    .data_out(data_out)
  );

  // Clock generation with a period of 10 ns (5 ns per edge)
  initial begin
    clk = 0;
    forever #5 clk = ~clk;
  end

```

```
27      // Test sequence
28      initial begin
29        write_enable = 0; // Read operation
30        address = 12'h000;
31        data_in = 16'h0000;
32        #10;
33        $display("Address: %h, Data In: %h", address, data_in);
34        address = 12'h001;
35        data_in = 16'hABCD;
36        write_enable = 1; // Write operation
37        #10;
38        write_enable = 0;
39        #10;
40        $display("Address: %h, Data Out: %h", address, data_out);
41        address = 12'h002;
42        data_in = 16'h1234;
43        write_enable = 1;
44        #10;
45        write_enable = 0;
46        #10;
47    /*the #10 means give 10 ns for each operation, which is splitted into two
48    periods, 5ns for the falling edge, and 5 ns for the rising edge */
49        $display("Address: %h, Data Out: %h", address, data_out);
50        $stop;
51      end
52    endmodule
```

Testing results :



Register Modules:

Register code:

```
module AR_Register
    #(parameter m = 16)  // Parameter 'm' defines the width of the register
    (
     input [m-1:0] Data_in,  // Input data to load into the register
     input LD,               // Load control signal
     input INR,              // Increment control signal
     input CLR,              // Clear control signal
     input CLK,              // Clock signal
     output reg [m-1:0] Data_out  // Output data from the register
    );

// Always start on the rising edge of the clock (CLK)
always @(posedge CLK)
begin
    // If the clear signal (CLR) is high, reset the register to zero
    if (CLR)
        Data_out <= {m{1'b0}};  // Set all bits of Data_out to 0
    // If the load signal (LD) is high, load the input data into the register
    else if (LD)
        Data_out <= Data_in;    // Assign the input data to Data_out
    /*If the increment signal (INR) is high,
    increment the current value in the register*/
    else if (INR)
        Data_out <= Data_out + 1; // Increment the current value of Data_out by 1
end
endmodule
/*repeat the same proccess for the rest o the registers,depending on the
requirements of each register*/
module PC_Register
    #(parameter m = 16)
    (input [m-1:0] Data_in,
     input LD, INR, CLR, CLK,
     output reg [m-1:0] Data_out);

    always @(posedge CLK)
    begin
        if (CLR)
            Data_out <= {m{1'b0}};
        else if (LD)
            Data_out <= Data_in;
        else if (INR)
            Data_out <= Data_out + 1;
    end
endmodule
module DR_Register
    #(parameter m = 16)
    (input [m-1:0] Data_in,
     input LD, INR, CLR, CLK,
     output reg [m-1:0] Data_out);

    always @(posedge CLK)
    begin
        if (CLR)
            Data_out <= {m{1'b0}};
        else if (LD)
            Data_out <= Data_in;
        else if (INR)
            Data_out <= Data_out + 1;
    end
endmodule
```

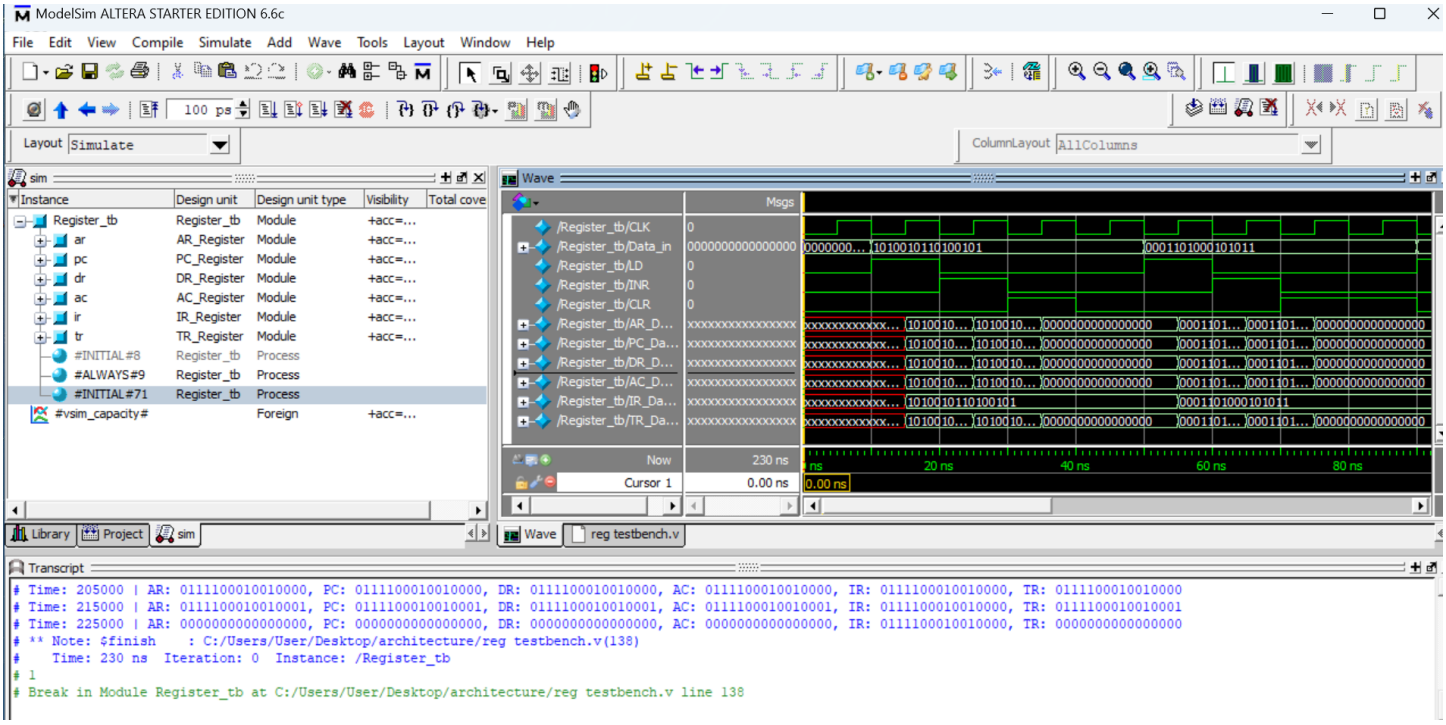
```
            Data_out <= {m{1'b0}};
        else if (LD)
            Data_out <= Data_in;
        else if (INR)
            Data_out <= Data_out + 1;
    end
endmodule
module IR_Register
    #(parameter m = 16)
    (input [m-1:0] Data_in,
     input LD, CLK,
     output reg [m-1:0] Data_out);

    always @(posedge CLK)
    begin
        if (LD)
            Data_out <= Data_in;
    end
endmodule
module TR_Register
    #(parameter m = 16)
    (input [m-1:0] Data_in,
     input LD, INR, CLR, CLK,
     output reg [m-1:0] Data_out);

    always @(posedge CLK)
    begin
        if (CLR)
            Data_out <= {m{1'b0}};
        else if (LD)
            Data_out <= Data_in;
        else if (INR)
            Data_out <= Data_out + 1;
    end
endmodule
```

Registers test bench:

```
`timescale 1ns / 1ps
module Register_tb;
parameter m = 16;// parameter helps to modify the size easily
reg CLK;
initial CLK = 0; // initail the clk at 0
always #5 CLK = ~CLK; // every 5 ns toggle the clock
reg [m-1:0] Data_in;
reg LD, INR, CLR;
wire [m-1:0] AR_Data_out, PC_Data_out, DR_Data_out, AC_Data_out, IR_Data_out, TR_Data_out;
//now instantiate all the modules you want to implement in the testbesnch
AR_Register #(.m(m)) ar (
    .Data_in(Data_in),
    .LD(LD),
    .INR(INR),
    .CLR(CLR),
    .CLK(CLK),
    .Data_out(AR_Data_out)
);
PC_Register #(.m(m)) pc (
    .Data_in(Data_in),
    .LD(LD),
    .INR(INR),
    .CLR(CLR),
    .CLK(CLK),
    .Data_out(PC_Data_out)
);
DR_Register #(.m(m))
dr(
    .Data_in(Data_in),
    .LD(LD),
    .INR(INR),
    .CLR(CLR),
    .CLK(CLK),
    .Data_out(DR_Data_out)
);
AC_Register #(.m(m)) ac (
    .Data_in(Data_in),
    .LD(LD),
    .INR(INR),
    .CLR(CLR),
    .CLK(CLK),
    .Data_out(AC_Data_out)
```

```
);
IR_Register #(.m(m)) ir (
    .Data_in(Data_in),
    .LD(LD),
    .CLK(CLK),
    .Data_out(IR_Data_out)
);
TR_Register #(.m(m)) tr (
    .Data_in(Data_in),
    .LD(LD),
    .INR(INR),
    .CLR(CLR),
    .CLK(CLK),
    .Data_out(TR_Data_out)
);
initial begin
    Data_in = {m{1'b0}};
    LD = 0; INR = 0; CLR = 0;
    $monitor("Time: %0t | AR: %b, PC: %b, DR: %b, AC: %b, IR: %b, TR: %b",
             $time, AR_Data_out, PC_Data_out, DR_Data_out, AC_Data_out, IR_Data_out, TR_Data_out);
    #10;
    Data_in = 16'hA5A5;
    LD = 1; CLR = 0;
    #10;
     CLR = 0;
     #10;
     Data_in = 16'h1A2B;
     LD = 1; CLR = 0;
     #10;
     LD = 0; INR = 1;
     #10;
     INR = 0; CLR = 1;
     #10;
     #10;
     Data_in = 16'hFF00;
     LD = 1; CLR = 0;
     #10;
     LD = 0; INR = 1;
     #10;
     INR = 0; CLR = 1;
     #10;
     #10;
     Data_in = 16'h1234;
     LD = 1; CLR = 0;
     #10;
     LD = 0; INR = 1;
     #10;
     INR = 0; CLR = 1;
     #10;
     #10;
     Data_in = 16'hABCD;
     LD = 1;
    #10;
    LD = 0;
    #10;
    #10;
    Data_in = 16'h7890;
    LD = 1; CLR = 0;
    #10;
    LD = 0; INR = 1;
    #10;
    INR = 0; CLR = 1;
    #10;
    $finish;
end
endmodule
/* This testbench (`Register_tb`) is designed to test the functionality of multiple register modules
 (AR_Register, PC_Register, DR_Register, AC_Register, IR_Register, and TR_Register). It defines the
 necessary parameters and control signals (Load, Increment, Clear) and provides a clock signal. The
 testbench initializes inputs and control signals, monitors the outputs of the registers, and
 simulates a series of operations to verify that each register behaves correctly under different
 conditions (loading data, incrementing values, and clearing). The simulation includes several
 test cases to check how the registers responds to different input values and control signals.*/
```

Testing results :



ALU implementation :

Explanation of the Code:

- **Module Declaration**: The module is defined as ALU, with a parameter A that sets the width of the operands and output to 16 bits.
- **Inputs and Outputs**: It takes two operands (op1 and op2), a 4-bit operation code (ALUOP).
- **Operation Codes**: Various operation codes are defined as parameters to specify which operation the ALU will perform.
- **Behavior**: The always block is sensitive to changes in inputs, and it uses a case statement to determine the operation based on ALUOP.
- **Default Case**: If ALUOP does not match any defined operation, the output is set to an unknown state (16'bx), indicating an error or undefined operation.
- **Extend Bit:** the extend bit is used for detecting overflow

ALU truth table :

ALUOP	Microoperation	SELA	SELB	SELD	OPR
0000	AC ← A & B	A	B	AC	AND
0001	AC ← A + B	A	B	AC	ADD
0010	AC ← 0 (CLA)	A	-	AC	CLA
0011	AC ← ~A (CMA)	A	-	AC	CMA
0100	AC ← A >> 1 (CIR)	A	-	AC	CIR
0101	AC ← A << 1 (CIL)	A	-	AC	CIL
0110	AC ← A + 1 (INC)	A	-	AC	INC
0111	E ← 0 (CLE)	-	-	E	CLE
1000	E ← ~E (CME)	E	-	E	CME
1001	AC ← 0 if A ≥ 0 (SPA)	A	-	AC	SPA
1010	AC ← 0 if A < 0 (SNA)	A	-	AC	SNA
1011	AC ← 0 if A == 0 (SZA)	A	-	AC	SZA
1100	AC ← 0 if E == 0 (SZE)	E	-	AC	SZE
1101	AC ← A (LDA)	A	-	AC	LDA

ALU CODE :

```
module ALU
#(
    parameter A = 16)
(
    input [A-1:0] op1, op2,
    input [3:0] ALUOP,
    output reg [A-1:0] data,
    output reg E
);
    parameter
        AND = 4'b0000,
        ADD = 4'b0001,
        CLA = 4'b0010,
        CMA = 4'b0011,
        CIR = 4'b0100,
        CIL = 4'b0101,
        INC = 4'b0110,
        CLE = 4'b0111,
        CME = 4'b1000,
        SPA = 4'b1001,
        SNA = 4'b1010,
        SZA = 4'b1011,
        SZE = 4'b1100,
        LDA = 4'b1101;

    always @(*) begin
        data = {A{1'bx}};
        E = 1'bx;
        case (ALUOP)
            AND: begin
                data = op1 & op2;
            end
            ADD: begin
                {E, data} = op1 + op2;
            end
            CLA: begin
                data = {A{1'b0}};
            end
            CMA: begin
                data = ~op1;
            end
            CIR: begin
                {data, E} = {op1, E} >> 1;
            end
            CIL: begin
                {E, data} = {E, op1} << 1;
            end
            INC: begin
                data = op1 + 1;
            end
            CLE: begin
                E = 1'b0;
            end
            CME: begin
                E = ~E;
            end
            SPA: begin
                if (op1[A-1] == 0)
                    data = {A{1'b0}};
            end
            SNA: begin
                if (op1[A-1] == 1)
                    data = {A{1'b0}};
            end
            SZA: begin
                if (op1 == {A{1'b0}})
                    data = {A{1'b0}};
            end
            SZE: begin
                if (E == 1'b0)
                    data = {A{1'b0}};
            end
            LDA: begin
                data = op1;
            end
            default: begin
                data = {A{1'bx}};
            end
        endcase
    end
endmodule
```

ALU testbench:

```
`timescale 1ns / 1ps
module ALU_tb;
    parameter A = 16;
    reg [A-1:0] op1;
    reg [A-1:0] op2;
    reg [3:0] ALUOP;
    wire [A-1:0] data;
    wire E;


    ALU #(.A(A)) uut (
        .op1(op1),
        .op2(op2),
        .ALUOP(ALUOP),
        .data(data),
        .E(E)
    );

    initial begin
        op1 = 0;
        op2 = 0;
        ALUOP = 0;

$monitor("Time: %0d, ALUOP: %b, op1: %b, op2: %b, data: %b, E: %b", $time, ALUOP, op1, op2, data, E);
#10
op1 = 16'hAAAA; op2 = 16'h5555; ALUOP = 4'b0000;
#10
op1 = 16'h0001; op2 = 16'h0001; ALUOP = 4'b0001;
#10
ALUOP = 4'b0010;
#10
op1 = 16'hAAAA; ALUOP = 4'b0011;
#10
op1 = 16'h8001; ALUOP = 4'b0100;
#10
op1 = 16'h4000; ALUOP = 4'b0101;
#10
op1 = 16'h0005; ALUOP = 4'b0110;
#10
ALUOP = 4'b0111;
#10
ALUOP = 4'b1000;
#10
op1 = 16'h0001; ALUOP = 4'b1001;
#10
op1 = 16'hFFFF; ALUOP = 4'b1010;
#10
op1 = 16'h0000; ALUOP = 4'b1011;
#10
ALUOP = 4'b1100;
#10
op1 = 16'h1234; ALUOP = 4'b1101;
#10
$stop;
end

endmodule
```

Test results: