



COMPUTER ARCHITECTURE

Dr. Deyaa Ghazi

Assignment1: Memory, Registers, and ALU implementation

Team # 5

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

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:

```
1  module memory
2      #(
3          parameter A = 12,
4          parameter m = 16
5      ) // parameters help editing the code if you want to change some values
6      (
7          input clk,
8          input [A-1:0] address,
9          input [m-1:0] data_in,
10         input write_enable,
11         output reg [m-1:0] data_out
12     ); // the i/p and o/p in the memory unit
13
14     localparam D = (1 << A);
15     reg [m-1:0] mem [0:D-1];
16     // parameterized memory array
17
18     always @(posedge clk) begin
19         if(address !=0) begin
20             if (write_enable)
21                 mem[address] <= data_in; // write operation
22             else
23                 data_out <= mem[address]; //read operation
24         end
25     end
26 endmodule
```

Memory unit testbench:

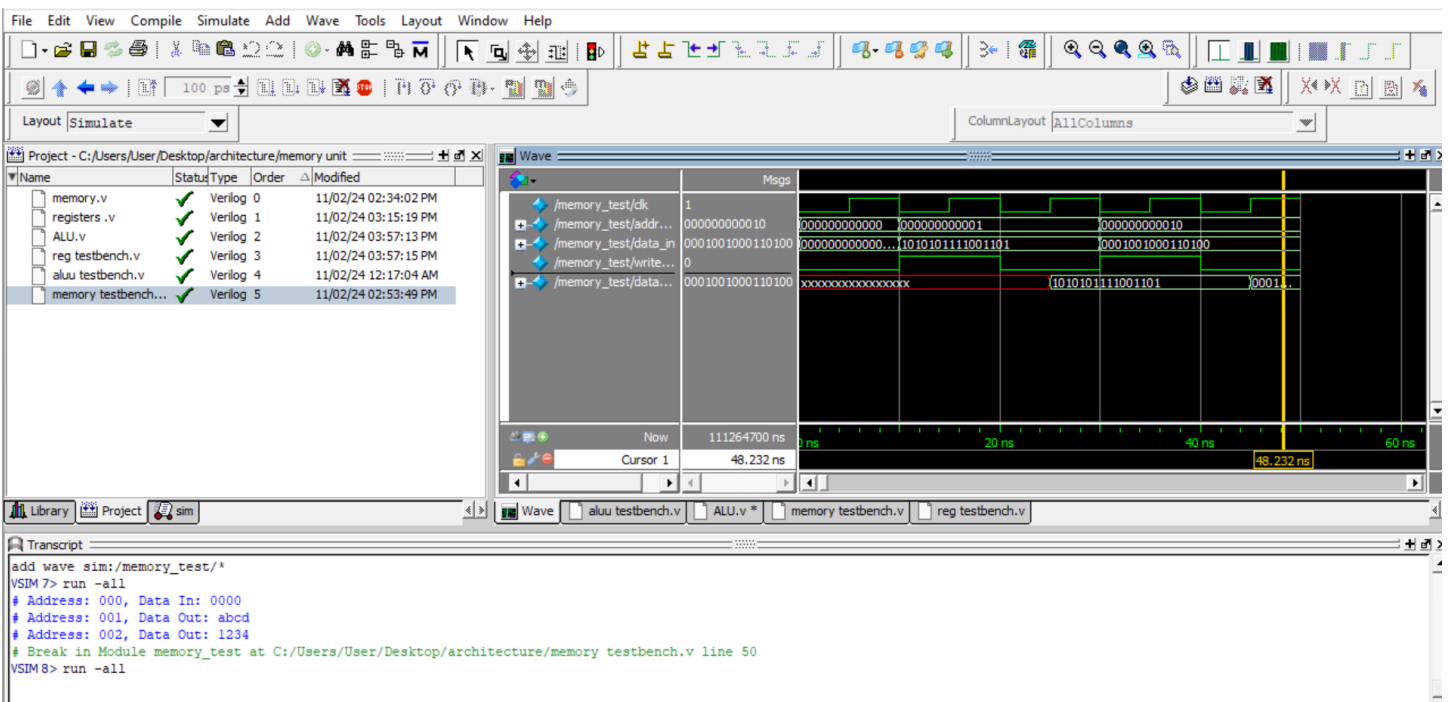
```
1  `timescale 1ns / 1ps
2  module memory_test();
3      parameter A = 12;
4      parameter m = 16;
5
6      reg clk;
7      reg [A-1:0] address;
8      reg [m-1:0] data_in;
9      reg write_enable;
10     wire [m-1:0] data_out;
11
12     // Instantiate the memory module
13     memory memory_test_inst (
14         .clk(clk),
15         .address(address),
16         .data_in(data_in),
17         .write_enable(write_enable),
18         .data_out(data_out)
19     );
20
21     // Clock generation with a period of 10 ns (5 ns per edge)
22     initial begin
23         clk = 0;
24         forever #5 clk = ~clk;
25     end
26
```

```

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:

```
1  module AR_Register
2      #(parameter m = 16) // Parameter 'm' defines the width of the register
3      (
4          input [m-1:0] Data_in, // Input data to load into the register
5          input LD,               // Load control signal
6          input INR,              // Increment control signal
7          input CLR,              // Clear control signal
8          input CLK,              // Clock signal
9          output reg [m-1:0] Data_out // Output data from the register
10     );
11
12     // Always start on the rising edge of the clock (CLK)
13     always @(posedge CLK)
14     begin
15         // If the clear signal (CLR) is high, reset the register to zero
16         if (CLR)
17             Data_out <= {m{1'b0}}; // Set all bits of Data_out to 0
18         // If the load signal (LD) is high, load the input data into the register
19         else if (LD)
20             Data_out <= Data_in; // Assign the input data to Data_out
21         /*If the increment signal (INR) is high,
22         increment the current value in the register:*/
23         else if (INR)
24             Data_out <= Data_out + 1; // Increment the current value of Data_out by 1
25     end
26 endmodule
27
28 /*repeat the same proccess for the rest o the registers,depending on the
29 requirements of each register*/
30 module PC_Register
31     #(parameter m = 16)
32     (input [m-1:0] Data_in,
33      input LD, INR, CLR, CLK,
34      output reg [m-1:0] Data_out);
35
36     always @(posedge CLK)
37     begin
38         if (CLR)
39             Data_out <= {m{1'b0}};
40         else if (LD)
41             Data_out <= Data_in;
42         else if (INR)
43             Data_out <= Data_out + 1;
44     end
45 endmodule
46 module DR_Register
47     #(parameter m = 16)
48     (input [m-1:0] Data_in,
49      input LD, INR, CLR, CLK,
50      output reg [m-1:0] Data_out);
51
52     always @(posedge CLK)
53     begin
54         if (CLR)
55             Data_out <= {m{1'b0}};
56         else if (LD)
57             Data_out <= Data_in;
58         else if (INR)
59             Data_out <= Data_out + 1;
60     end
61 endmodule
```

```

71         Data_out <= {m{1'b0}};
72     else if (LD)
73         Data_out <= Data_in;
74     else if (INR)
75         Data_out <= Data_out + 1;
76     end
77 endmodule
78 module IR_Register
79     #(parameter m = 16)
80     (input [m-1:0] Data_in,
81      input LD, CLK,
82      output reg [m-1:0] Data_out);
83
84     always @(posedge CLK)
85     begin
86         if (LD)
87             Data_out <= Data_in;
88     end
89 endmodule
90 module TR_Register
91     #(parameter m = 16)
92     (input [m-1:0] Data_in,
93      input LD, INR, CLR, CLK,
94      output reg [m-1:0] Data_out);
95
96     always @(posedge CLK)
97     begin
98         if (CLR)
99             Data_out <= {m{1'b0}};
100        else if (LD)
101            Data_out <= Data_in;
102        else if (INR)
103            Data_out <= Data_out + 1;
104    end
105 endmodule

```

Registers test bench:

```

1  `timescale 1ns / 1ps
2  module Register_tb;
3  parameter m = 16; // parameter helps to modify the size easily
4  reg CLK;
5  initial CLK = 0; // initail the clk at 0
6  always #5 CLK = ~CLK; // every 5 ns toggle the clock
7  reg [m-1:0] Data_in;
8  reg LD, INR, CLR;
9  wire [m-1:0] AR_Data_out, PC_Data_out, DR_Data_out, AC_Data_out, IR_Data_out, TR_Data_out;
10 //now instantiate all the modules you want to implement in the testbench
11 AR_Register #(m(m)) ar (
12     .Data_in(Data_in),
13     .LD(LD),
14     .INR(INR),
15     .CLR(CLR),
16     .CLK(CLK),
17     .Data_out(AR_Data_out)
18 );
19 PC_Register #(m(m)) pc (
20     .Data_in(Data_in),
21     .LD(LD),
22     .INR(INR),
23     .CLR(CLR),
24     .CLK(CLK),
25     .Data_out(PC_Data_out)
26 );
27 DR_Register #(m(m))
28 dr(
29     .Data_in(Data_in),
30     .LD(LD),
31     .INR(INR),
32     .CLR(CLR),
33     .CLK(CLK),
34     .Data_out(DR_Data_out)
35 );
36 AC_Register #(m(m)) ac (
37     .Data_in(Data_in),
38     .LD(LD),
39     .INR(INR),
40     .CLR(CLR),
41     .CLK(CLK),
42     .Data_out(AC_Data_out)

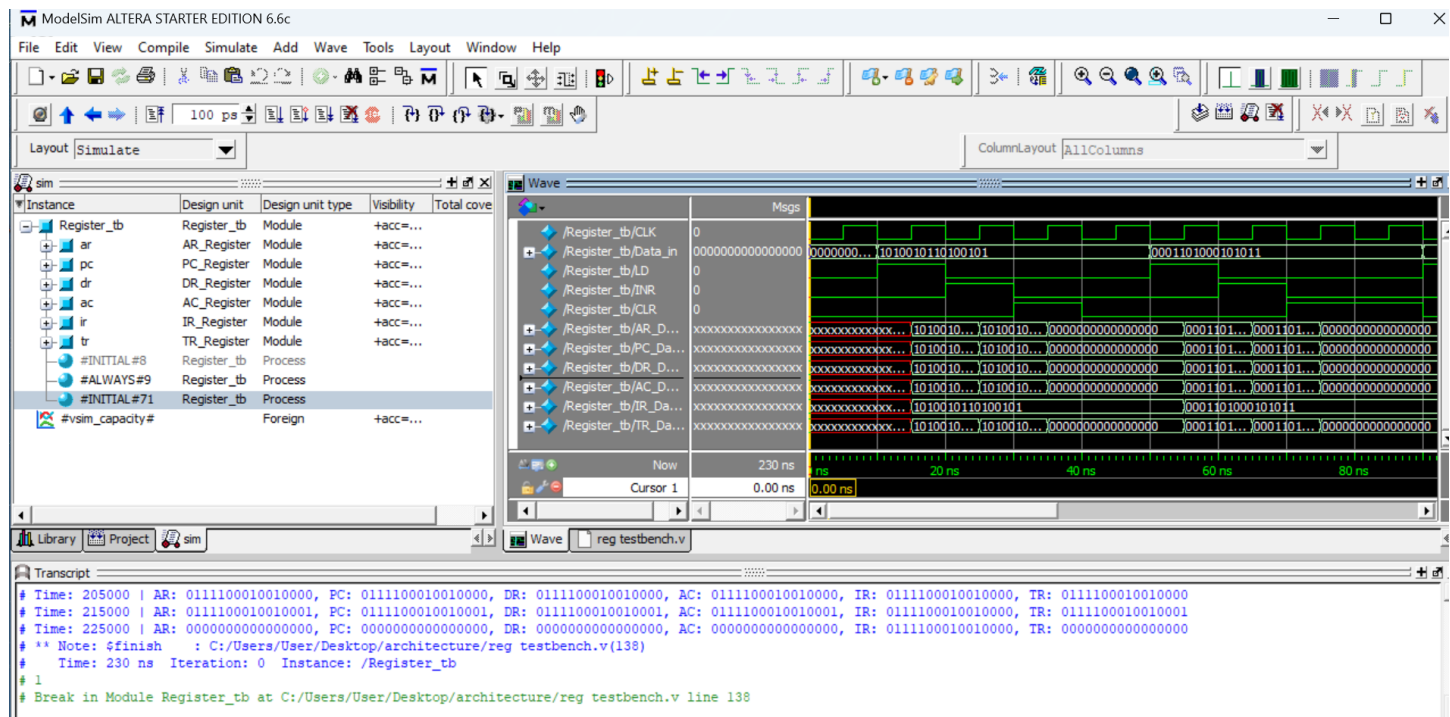
```

```

43 );
44 IR_Register #(.m(m)) ir (
45     .Data_in(Data_in),
46     .LD(LD),
47     .CLK(CLK),
48     .Data_out(IR_Data_out)
49 );
50 TR_Register #(.m(m)) tr (
51     .Data_in(Data_in),
52     .LD(LD),
53     .INR(INR),
54     .CLR(CLR),
55     .CLK(CLK),
56     .Data_out(TR_Data_out)
57 );
58 initial begin
59     Data_in = {m{1'b0}};
60     LD = 0; INR = 0; CLR = 0;
61     $monitor("Time: %0t | AR: %b, PC: %b, DR: %b, AC: %b, IR: %b, TR: %b",
62         $time, AR_Data_out, PC_Data_out, DR_Data_out, AC_Data_out, IR_Data_out, TR_Data_out);
63     #10;
64     Data_in = 16'hA5A5;
65     LD = 1; CLR = 0;
66     #10;
67     CLR = 0;
68     #10;
69     Data_in = 16'h1A2B;
70     LD = 1; CLR = 0;
71     #10;
72     LD = 0; INR = 1;
73     #10;
74     INR = 0; CLR = 1;
75     #10;
76     #10;
77     Data_in = 16'hFF00;
78     LD = 1; CLR = 0;
79     #10;
80     LD = 0; INR = 1;
81     #10;
82     INR = 0; CLR = 1;
83     #10;
84     #10;
85     Data_in = 16'h1234;
86     LD = 1; CLR = 0;
87     #10;
88     LD = 0; INR = 1;
89     #10;
90     INR = 0; CLR = 1;
91     #10;
92     #10;
93     Data_in = 16'hABCD;
94     LD = 1;
95     #10;
96     LD = 0;
97     #10;
98     Data_in = 16'h7890;
99     LD = 1; CLR = 0;
100    #10;
101    LD = 0; INR = 1;
102    #10;
103    INR = 0; CLR = 1;
104    #10;
105    $finish;
106 end
107 endmodule
108
109 /* This testbench ('Register_tb') is designed to test the functionality of multiple register modules
110 (AR_Register, PC_Register, DR_Register, AC_Register, IR_Register, and TR_Register). It defines the
111 necessary parameters and control signals (Load, Increment, Clear) and provides a clock signal. The
112 testbench initializes inputs and control signals, monitors the outputs of the registers, and
113 simulates a series of operations to verify that each register behaves correctly under different
114 conditions (loading data, incrementing values, and clearing). The simulation includes several
115 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	$AC \leftarrow A \& B$	A	B	AC	AND
0001	$AC \leftarrow A + B$	A	B	AC	ADD
0010	$AC \leftarrow 0$ (CLA)	A	-	AC	CLA
0011	$AC \leftarrow \sim A$ (CMA)	A	-	AC	CMA
0100	$AC \leftarrow A \gg 1$ (CIR)	A	-	AC	CIR
0101	$AC \leftarrow A \ll 1$ (CIL)	A	-	AC	CIL
0110	$AC \leftarrow A + 1$ (INC)	A	-	AC	INC
0111	$E \leftarrow 0$ (CLE)	-	-	E	CLE
1000	$E \leftarrow \sim E$ (CME)	E	-	E	CME
1001	$AC \leftarrow 0$ if $A \geq 0$ (SPA)	A	-	AC	SPA
1010	$AC \leftarrow 0$ if $A < 0$ (SNA)	A	-	AC	SNA
1011	$AC \leftarrow 0$ if $A == 0$ (SZA)	A	-	AC	SZA
1100	$AC \leftarrow 0$ if $E == 0$ (SZE)	E	-	AC	SZE
1101	$AC \leftarrow A$ (LDA)	A	-	AC	LDA

ALU CODE :

```
1  module ALU
2  #(
3      parameter A = 16)
4  (
5      input [A-1:0] op1, op2,
6      input [3:0] ALUOP,
7      output reg [A-1:0] data,
8      output reg E
9  );
10
11      parameter
12          AND = 4'b0000,
13          ADD = 4'b0001,
14          CLA = 4'b0010,
15          CMA = 4'b0011,
16          CIR = 4'b0100,
17          CIL = 4'b0101,
18          INC = 4'b0110,
19          CLE = 4'b0111,
20          CME = 4'b1000,
21          SPA = 4'b1001,
22          SNA = 4'b1010,
23          SZA = 4'b1011,
24          SZE = 4'b1100,
25          LDA = 4'b1101;
26
27      always @(*) begin
28          data = {A{1'b0}};
29          E = 1'b0;
30          case (ALUOP)
31              AND: begin
32                  data = op1 & op2;
33              end
34              ADD: begin
35                  {E, data} = op1 + op2;
36              end
37              CLA: begin
38                  data = {A{1'b0}};
39              end
40              CMA: begin
41                  data = ~op1;
42              end
43              CIR: begin
44                  {data, E} = {op1, E} >> 1;
45              end
46              CIL: begin
47                  {E, data} = {E, op1} << 1;
48              end
49              INC: begin
50                  data = op1 + 1;
51              end
52              CLE: begin
53                  E = 1'b0;
54              end
55              CME: begin
56                  E = ~E;
57              end
58              SPA: begin
59                  if (op1[A-1] == 0)
60                      data = {A{1'b0}};
61              end
62              SNA: begin
63                  if (op1[A-1] == 1)
64                      data = {A{1'b0}};
65              end
66              SZA: begin
67                  if (op1 == {A{1'b0}})
68                      data = {A{1'b0}};
69              end
70              SZE: begin
71                  if (E == 1'b0)
72                      data = {A{1'b0}};
73              end
74              LDA: begin
75                  data = op1;
76              end
77              default: begin
78                  data = {A{1'b0}};
79              end
80          endcase
81      end
82  endmodule
```

ALU testbench:

```
1  `timescale 1ns / 1ps
2  module ALU_tb;
3      parameter A = 16;
4      reg [A-1:0] op1;
5      reg [A-1:0] op2;
6      reg [3:0] ALUOP;
7      wire [A-1:0] data;
8      wire E; |
9
10
11      ALU #(A(A)) uut (
12          .op1(op1),
13          .op2(op2),
14          .ALUOP(ALUOP),
15          .data(data),
16          .E(E)
17      );
18
19      initial begin
20          op1 = 0;
21          op2 = 0;
22          ALUOP = 0;
23
24      $monitor("Time: %0d, ALUOP: %b, op1: %b, op2: %b, data: %b, E: %b", $time, ALUOP, op1, op2, data, E);
25      #10
26      op1 = 16'hAAAA; op2 = 16'h5555; ALUOP = 4'b0000;
27      #10
28      op1 = 16'h0001; op2 = 16'h0001; ALUOP = 4'b0001;
29      #10
30      ALUOP = 4'b0010;
31      #10
32      op1 = 16'hAAAA; ALUOP = 4'b0011;
33      #10
34      op1 = 16'h8001; ALUOP = 4'b0100;
35      #10
36      op1 = 16'h4000; ALUOP = 4'b0101;
37      #10
38      op1 = 16'h0005; ALUOP = 4'b0110;
39      #10
40      ALUOP = 4'b0111;
41      #10
42      ALUOP = 4'b1000;
43      #10
44      op1 = 16'h0001; ALUOP = 4'b1001;
45      #10
46      op1 = 16'hFFFF; ALUOP = 4'b1010;
47      #10
48      op1 = 16'h0000; ALUOP = 4'b1011;
49      #10
50      ALUOP = 4'b1100;
51      #10
52      op1 = 16'h1234; ALUOP = 4'b1101;
53      #10
54      $stop;
55      end
56
57  endmodule
```

Test results:

