بسم الله الرحمن الرحیم

ساختار کامپیوتر

امیر حسین رستمی

96101635

گزارش کار فاز اول پروژه

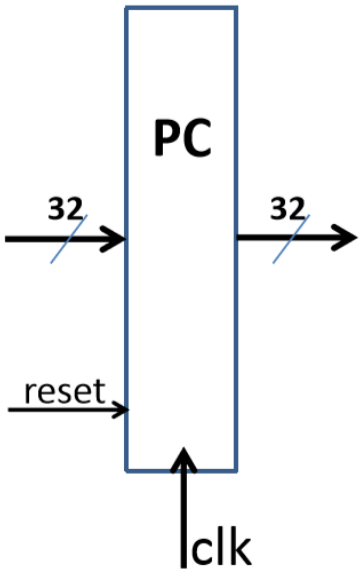
دکتر موحدیان عطار

دانشگاه صنعتی شریف بهار 98

گزارش فاز اول :

ماژول PC :

اول اینکه من ماژول PC تعریف کردم برای اینکه اگر ورودی ریست وارد شد پی سی صفر شود و اگرنه در هر پالس کلاک دستور کد بعدی را از حافظه لود کرده و پردازش کند.




```
always @( posedge clk )
    if ( reset )
        PC <=       32'h00000000;
    else
        PC <=       nxt_PC
```

هدف از ماژول فوق :

هدف module:

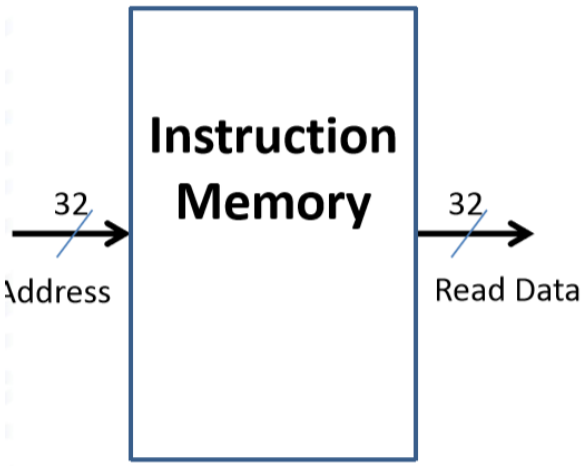
هدف طراحی ماژولی با ورودی و خروجی های ذکر شده و حافظه RAM تعریف شده به شکل بالا است که با دریافت یک آدرس 32 بیتی به عنوان ورودی (Address)، رجیستر 32 بیتی خروجی را با مقدار RAM[Address] مقدار دهی کند.

- طبیعتا از <= استفاده کردم چون در کلاک بعدی می خواهم که پی سی با مقدار مطلوب ست شود.

```
module PC(

input clk,
input reset,
input [31:0] nxt_pc,
output reg [31:0]out
    );

always@(posedge clk)
begin

    if(reset)
    begin
    out <= 32'h0000_0000;
    end

    else
    begin
    out <= nxt_pc;
    end

end

endmodule
```

ماژول InstructionMemory :

هدف از این ماژول این است که از آدرس داده شده برود داده بخواند و داده ی خوانده شده را به خروجی برگرداند ...




```
`timescale 1ns/1ps

module instr_mem(
  input [31:0] address,
  output [31:0] read_data
);

  reg [31:0] mem_data [0 : (1 << 30)-1];

  assign        read_data =
                mem_data[ address[31:2] ];

endmodule
```

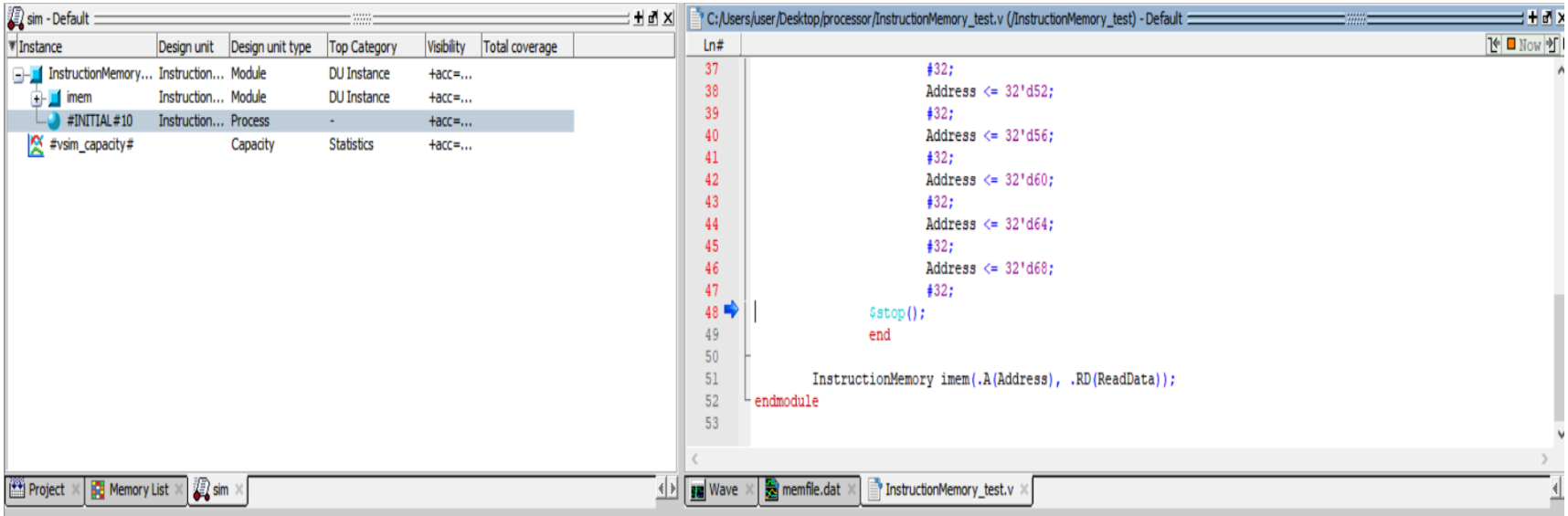
هدف از ماژول :

هدف طراحی ماژولی با ورودی و خروجی های ذکر شده و حافظه RAM تعریف شده به شکل بالا است که با دریافت یک آدرس 32 بیتی به عنوان ورودی (Address)، رجیستر 32 بیتی خروجی را با مقدار RAM[Address] مقدار دهی کند.

این ماژول از خانه های موجود در memfile.dat داده ها را می خواند ...

- خروجی تست بنچ این قسمت در صفحه ی بعد آمده است و در این تست بنچ داده ها از فایل memdat. برداشته شده و بروی ممووری نوشته می شوند.

```
module InstructionMemory(

input [31:0] A,
output [31:0] RD
);

reg [31:0] RAM [1023:0];
initial begin
$readmemh("memfile.dat",RAM);
end
assign RD = RAM[A[31:2]];

endmodule
```

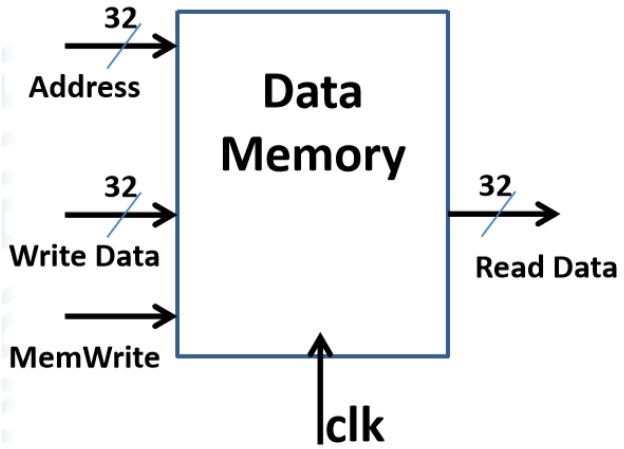


Transcript

```
VSIM 10> quit -sim
# End time: 17:04:57 on May 09,2019, Elapsed time: 0:00:27
# Errors: 0, Warnings: 0
ModelSim> vsim work.InstructionMemory_test -novopt
# vsim work.InstructionMemory_test -novopt
# Start time: 17:05:19 on May 09,2019
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cause your simulation to run very slowly. If you are using this switch to preserve visibility for Debug o
r PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
#
# Refreshing C:/Users/user/Desktop/processor/work.InstructionMemory_test
# Loading work.InstructionMemory_test
# Refreshing C:/Users/user/Desktop/processor/work.InstructionMemory
# Loading work.InstructionMemory
VSIM 12> run -all
# ** Note: $stop    : C:/Users/user/Desktop/processor/InstructionMemory_test.v(48)
#    Time: 5760 ns  Iteration: 0  Instance: /InstructionMemory_test
# Break in Module InstructionMemory_test at C:/Users/user/Desktop/processor/InstructionMemory_test.v line 48
```

ماژول

: DataMemory




```
`timescale 1ns/1ns

module async_mem(
  input clk,
  input write,
  input [31:0] address,
  input [31:0] write_data,
  output [31:0] read_data
);

  reg [31:0] mem_data [0 : (1 << 30)-1];

  assign #0.25 read_data =
            mem_data[ address[31:2] ];

  always @( posedge clk )
    if ( write )
      mem_data[ address[31:2] ] <=
                              write_data;

endmodule
```

هدف از ماژول :

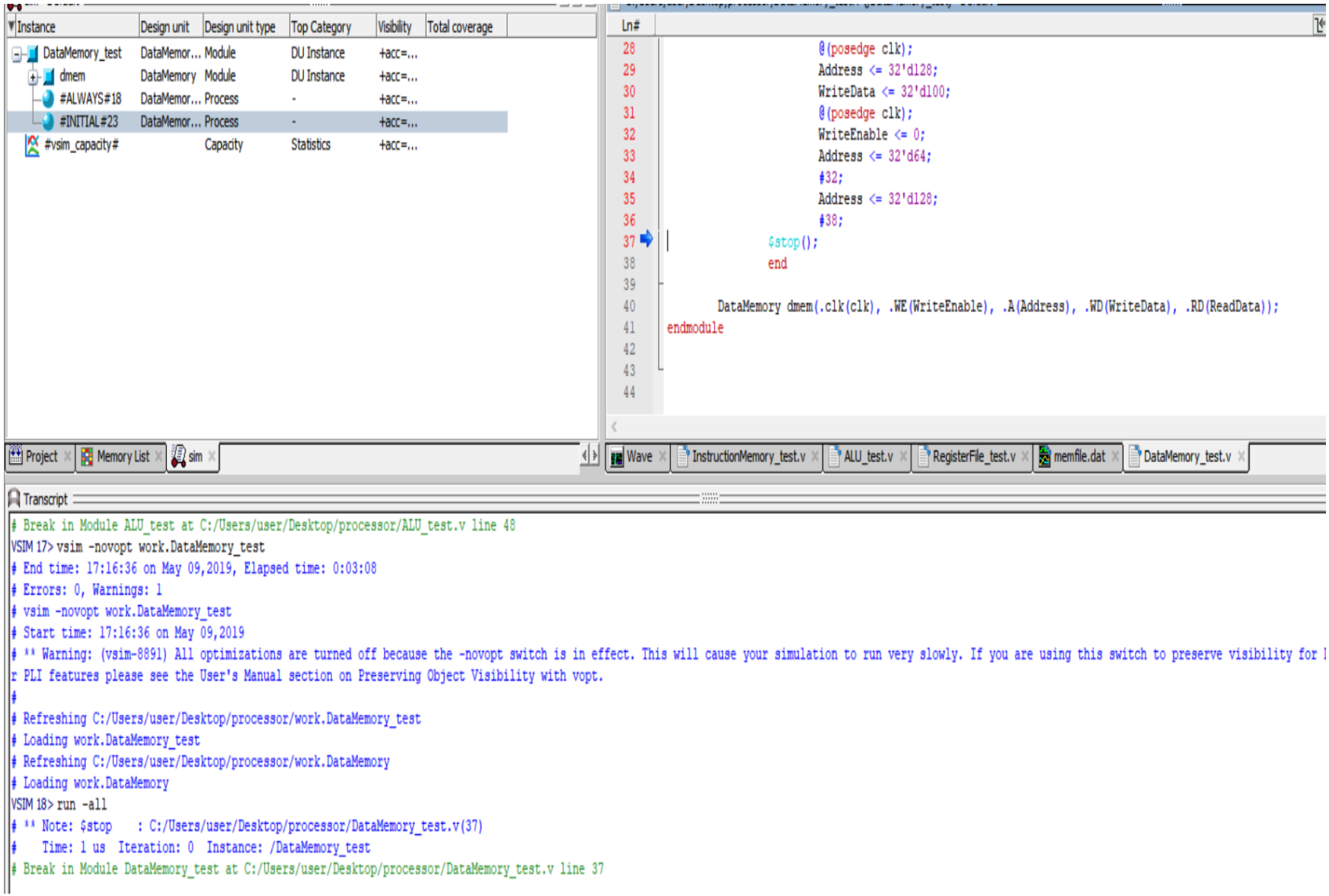
هدف طراحی ماژولی با ورودی و خروجی های ذکر شده و یک حافظه RAM دو بعدی با 1024 خانه 32 بیتی می باشد که با دریافت یک آدرس 32 بیتی word aligned به عنوان ورودی (Address)، همواره مقدار موجود در خانه Address را برای خواندن در رجیستر خروجی قرار می دهد. (مدار ترکیبی و بدون نیاز به clk)

و هم چنین، در صورت فعال بودن بیت MemWrite، در انتهای کلاک بعد مقدار موجود در ورودی Write Data را در خانه حافظه RAM با آدرس 32 بیتی Address قرار می دهد.

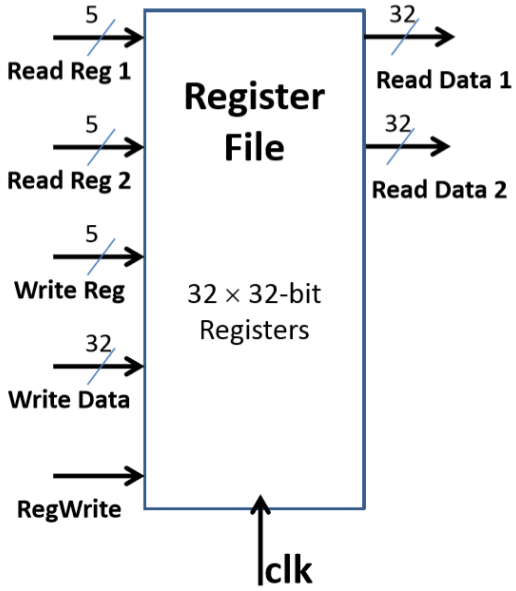
داریم که کد زده شده به صورت زیر می باشد:

نتیجه تست بنچ این قسمت در صفحه ی بعد قرار داده شده است.

```
module DataMemory(
input clk,
input WE,
input [31:0] A,
input [31:0] WD,
output [31:0] RD
);

reg [31:0] RAM[1023:0];
assign RD = RAM[A[31:2]];

always @(posedge clk)
    if(WE)
    RAM[A[31:2]] <= WD;

endmodule
```


Instance
Design unit
Design unit type
Top Category
Visibility
Total coverage
DataMemory_test
DataMemor...
Module
DU Instance
+acc=...
dmem
DataMemory
Module
DU Instance
+acc=...
#ALWAYS#18
DataMemor...
Process
-
+acc=...
#INITIAL#23
DataMemor...
Process
-
+acc=...
#vsim_capacity#
Capacity
Statistics
+acc=...
Project
Memory List
sim
Ln#
28 @(posedge clk);
29 Address <= 32'd128;
30 WriteData <= 32'd100;
31 @(posedge clk);
32 WriteEnable <= 0;
33 Address <= 32'd64;
34 #32;
35 Address <= 32'd128;
36 #38;
37 $stop();
38 end
39
40 DataMemory dmem(.clk(clk), .WE(WriteEnable), .A(Address), .WD(WriteData), .RD(ReadData));
41 endmodule
42
43
44
Wave
InstructionMemory_test.v
ALU_test.v
RegisterFile_test.v
memfile.dat
DataMemory_test.v
Transcript
Break in Module ALU_test at C:/Users/user/Desktop/processor/ALU_test.v line 48
VSIM 17> vsim -novopt work.DataMemory_test
End time: 17:16:36 on May 09,2019, Elapsed time: 0:03:08
Errors: 0, Warnings: 1
vsim -novopt work.DataMemory_test
Start time: 17:16:36 on May 09,2019
** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cause your simulation to run very slowly. If you are using this switch to preserve visibility for
r PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
#
Refreshing C:/Users/user/Desktop/processor/work.DataMemory_test
Loading work.DataMemory_test
Refreshing C:/Users/user/Desktop/processor/work.DataMemory
Loading work.DataMemory
VSIM 18> run -all
** Note: $stop : C:/Users/user/Desktop/processor/DataMemory_test.v(37)
Time: 1 us Iteration: 0 Instance: /DataMemory_test
Break in Module DataMemory_test at C:/Users/user/Desktop/processor/DataMemory_test.v line 37

ماژولRegisterFile :




```
`timescale 1ns/1ps

module reg_file(
  input clk,
  input write,
  input [4:0] WR,
  input [31:0] WD,
  input [4:0] RR1,
  input [4:0] RR2,
  output [31:0] RD1,
  output [31:0] RD2
);

  reg [31:0] rf_data [0:31];

assign        RD1 = rf_data[ RR1 ];

assign        RD2 = rf_data[ RR2 ];

always @( posedge clk ) begin

    if ( write )
      rf_data[ WR ] <= WD;

    rf_data[0] <= 32'h00000000;

  end

endmodule
```

هدف از این ماژول این است که :

هدف طراحی ماژولی با ورودی و خروجی های ذکر شده و یک رجیستر فایل دو بعدی با 32 رجیستر 32 بیتی می باشد که با دریافت سه آدرس 5 بیتی به عنوان ورودی، همواره رجیستر ها با آدرس های Read Reg 1 و Read Reg 2 را در خروجی های Read Data 1 و Read Data 2 قرار می دهد. (مدار ترکیبی و بدون نیاز به clk)

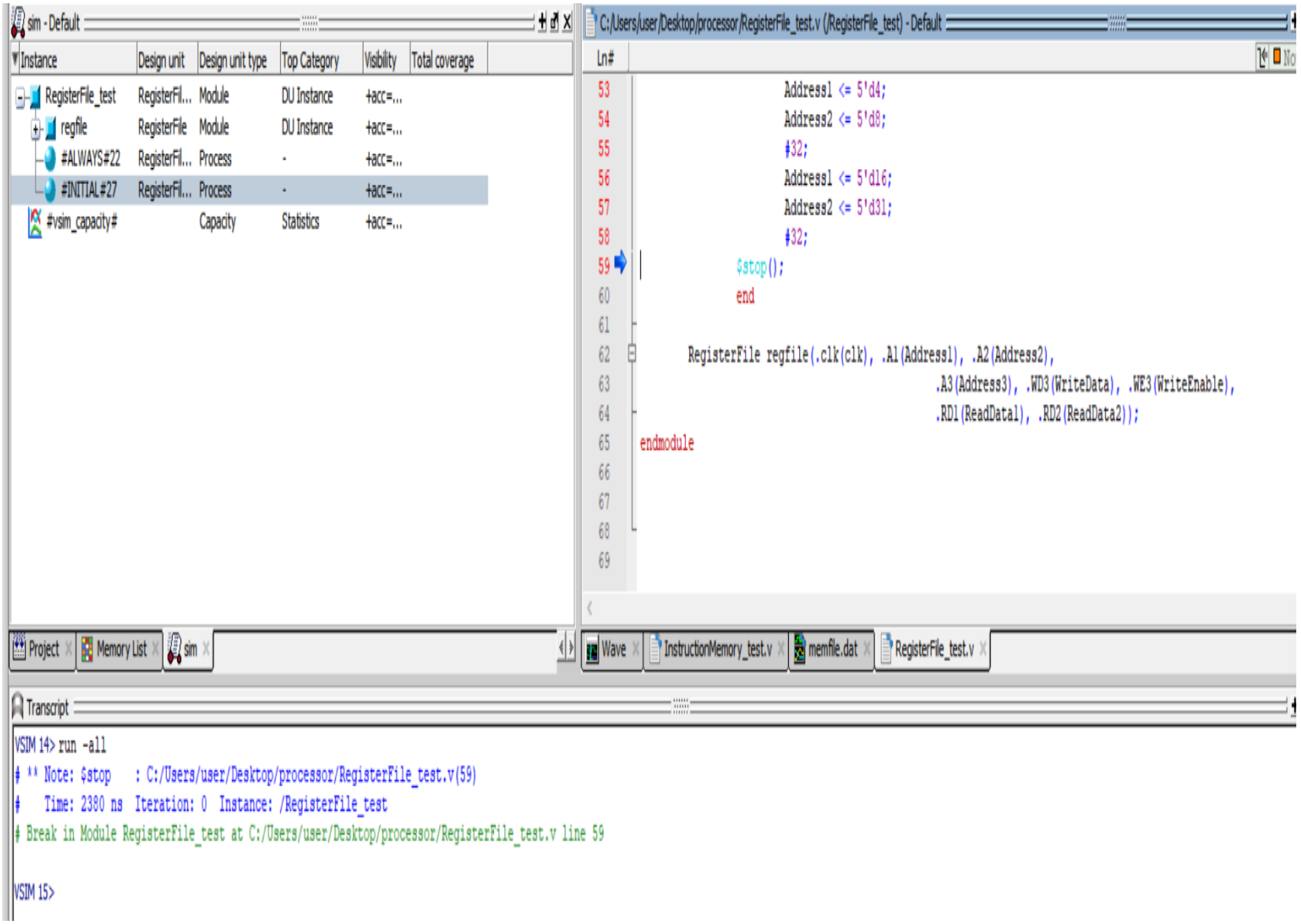
و هم چنین، در صورت فعال بودن بیت RegWrite، در انتهای کلاک بعد مقدار موجود در ورودی Write Data را در رجیستر با آدرس 5 بیتی Write Reg قرار می دهد.

```
module RegisterFile(
input clk,
input [4:0] A1,
input [4:0] A2,
input [4:0] A3,
input WE3,
input [31:0] WD3,
output[31:0] RD1,
output[31:0] RD2
);

reg[31:0] rf_data[0:31];
assign RD1 = rf_data[A1];
assign RD2 = rf_data[A2];

always@(posedge clk) begin
    if(WE3)
    rf_data[A3] <= WD3;
    rf_data[0] <= 32'h0000_0000;
    end

endmodule
```

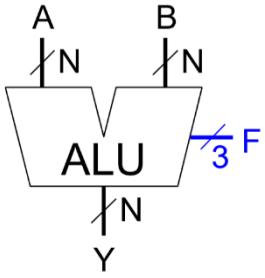
کد زده شده برای این قسمت : <----------

در صفحه ی بعد خروجی تست بنچ این قسمت نهادینه شده است :

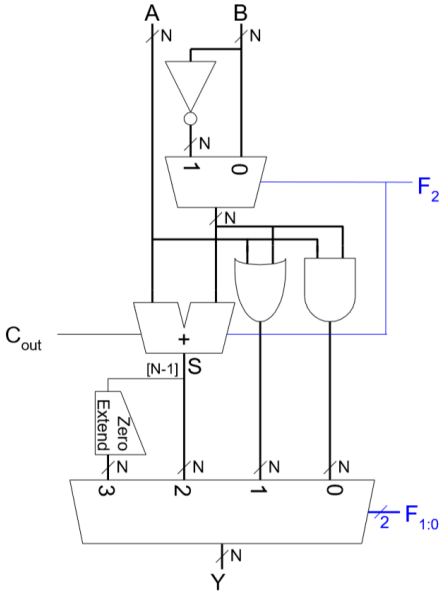
و ملاحظه می کنید که بدون مشکلی این قسمت ران می شود و ارتباط ورودی خروجی انجام میگیرد.


sim - Default
Instance
Design unit
Design unit type
Top Category
Visibility
Total coverage
RegisterFile_test
RegisterFil...
Module
DU Instance
+acc=...
regfile
RegisterFile
Module
DU Instance
+acc=...
#ALWAYS#22
RegisterFil...
Process
-
+acc=...
#INITIAL#27
RegisterFil...
Process
-
+acc=...
#vsim_capacity#
Capacity
Statistics
+acc=...
C:/Users/user/Desktop/processor/RegisterFile_test.v (/RegisterFile_test) - Default
Ln#
53 Address1 <= 5'd4;
54 Address2 <= 5'd8;
55 #32;
56 Address1 <= 5'd16;
57 Address2 <= 5'd31;
58 #32;
59 $stop();
60 end
61
62 RegisterFile regfile(.clk(clk), .A1(Address1), .A2(Address2),
63 .A3(Address3), .WD3(WriteData), .WE3(WriteEnable),
64 .RD1(ReadData1), .RD2(ReadData2));
65 endmodule
66
67
68
69
Project
Memory List
sim
Wave
InstructionMemory_test.v
memfile.dat
RegisterFile_test.v
Transcript
VSIM 14> run -all
** Note: $stop : C:/Users/user/Desktop/processor/RegisterFile_test.v(59)
Time: 2380 ns Iteration: 0 Instance: /RegisterFile_test
Break in Module RegisterFile_test at C:/Users/user/Desktop/processor/RegisterFile_test.v line 59
VSIM 15>

ماژول ALU :



$F_{2:0}$	Function
000	A & B
001	A | B
010	A + B
011	not used
100	A & ~B
101	A | ~B
110	A - B
111	SLT



هدف :

هدف طراحی ماژولی با ورودی و خروجی های ذکر شده می باشد که با دریافت دو رجیستر 32 بیتی به عنوان ورودی، با توجه به ALUControl همواره عملیات مورد نظر را که در زیر آمده اند بر روی دو رجیستر انجام داده و نتیجه خروجی را در رجیستر با نام Y قرار می دهد.

نکته: در صورت صفر شدن Y، خروجی یک بیتی Z فعال می شود.

جدول عملیات های ALU با توجه به ۳ بیت ALUControl:

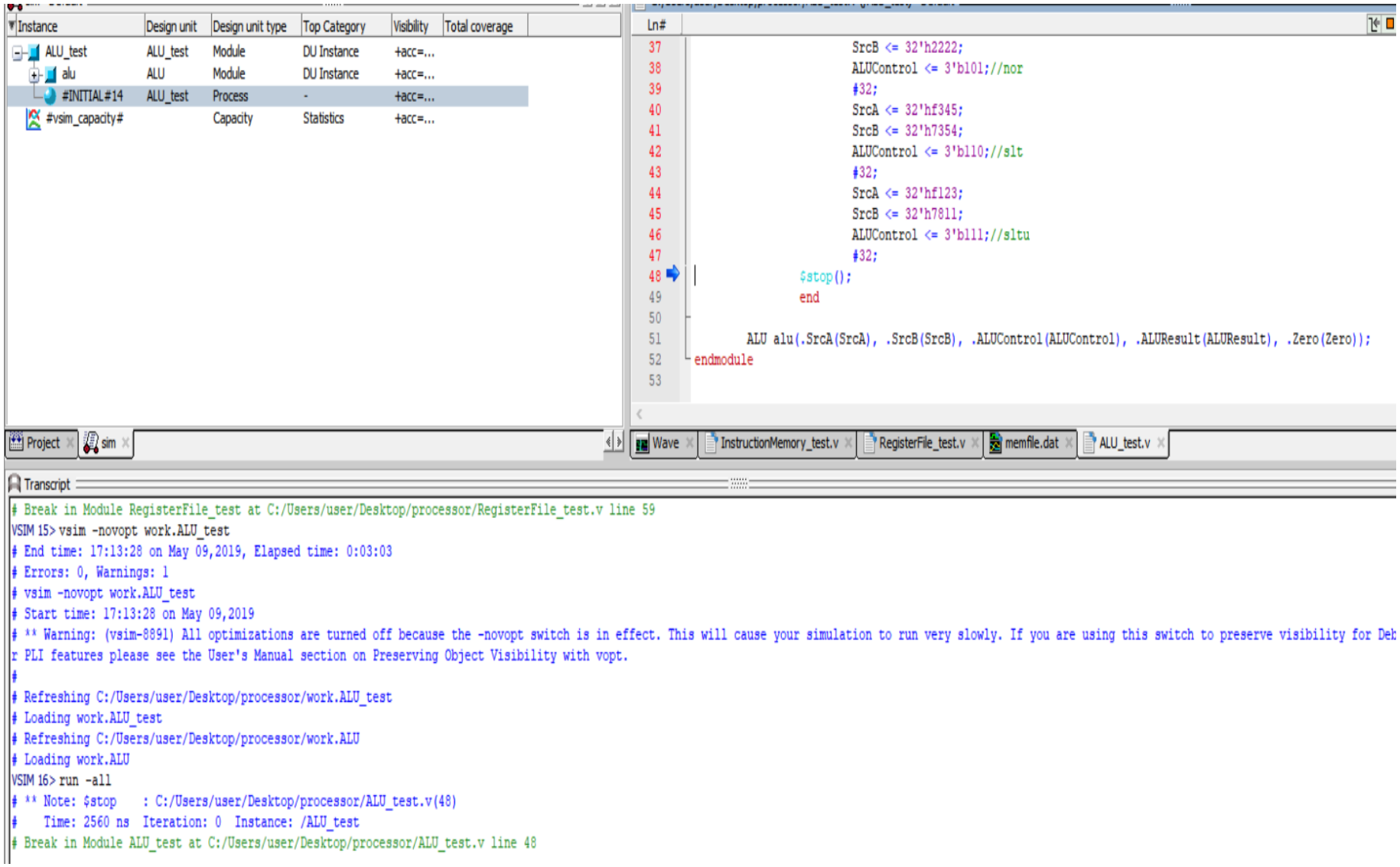
Operation	ALUControl[2:0]
ADD	3'b000
SUB	3'b001
AND	3'b010
OR	3'b011
XOR	3'b100
NOR	3'b101
SLT	3'b110
SLTU	3'b111

پیاده سازی ماژول ...

```
module ALU(
input [31:0]SrcA,
input [31:0]SrcB,
input [2:0]ALUControl,
output[31:0]ALUResult,
output Zero
);

assign ALUResult = ( ALUControl == 3'b000 ) ? SrcA + SrcB :
                   ( ALUControl == 3'b001 ) ? SrcA - SrcB :
                   ( ALUControl == 3'b010 ) ? SrcA & SrcB :
                   ( ALUControl == 3'b011 ) ? SrcA | SrcB :
                   ( ALUControl == 3'b100 ) ? SrcA ^ SrcB :
                   ( ALUControl == 3'b101 ) ? ~(SrcA | SrcB) :
                   ( ALUControl == 3'b110 ) ? $signed(SrcA) < $signed(SrcB):
                   ( ALUControl == 3'b111 ) ? SrcA < SrcB : 32'h00000000;

assign Zero = ( ALUResult == 32'h00000000 )? 1 : 0;

endmodule
```

خروجی تست بنچ این قسمت در صفحه ی بعد قرار داده شده است ....

همانطور که ملاحظه می کنید بدون اشکالی تبادل داده با ALU انجام میگیرد.


Instance
Design unit
Design unit type
Top Category
Visibility
Total coverage
ALU_test
ALU_test
Module
DU Instance
+acc=...
alu
ALU
Module
DU Instance
+acc=...
#INITIAL#14
ALU_test
Process
-
+acc=...
#vsim_capacity#
Capacity
Statistics
+acc=...
Ln#
37 SrcB <= 32'h2222;
38 ALUControl <= 3'b101;//nor
39 #32;
40 SrcA <= 32'hf345;
41 SrcB <= 32'h7354;
42 ALUControl <= 3'b110;//slt
43 #32;
44 SrcA <= 32'hf123;
45 SrcB <= 32'h7811;
46 ALUControl <= 3'b111;//sltu
47 #32;
48 $stop();
49 end
50
51 ALU alu(.SrcA(SrcA), .SrcB(SrcB), .ALUControl(ALUControl), .ALUResult(ALUResult), .Zero(Zero));
52 endmodule
53
Project
sim
Wave
InstructionMemory_test.v
RegisterFile_test.v
memfile.dat
ALU_test.v
Transcript
Break in Module RegisterFile_test at C:/Users/user/Desktop/processor/RegisterFile_test.v line 59
VSIM 15> vsim -novopt work.ALU_test
End time: 17:13:28 on May 09,2019, Elapsed time: 0:03:03
Errors: 0, Warnings: 1
vsim -novopt work.ALU_test
Start time: 17:13:28 on May 09,2019
** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cause your simulation to run very slowly. If you are using this switch to preserve visibility for Deb
r PLI features please see the User's Manual section on Preserving Object Visibility with vopt.
#
Refreshing C:/Users/user/Desktop/processor/work.ALU_test
Loading work.ALU_test
Refreshing C:/Users/user/Desktop/processor/work.ALU
Loading work.ALU
VSIM 16> run -all
** Note: $stop : C:/Users/user/Desktop/processor/ALU_test.v(48)
Time: 2560 ns Iteration: 0 Instance: /ALU_test
Break in Module ALU_test at C:/Users/user/Desktop/processor/ALU_test.v line 48