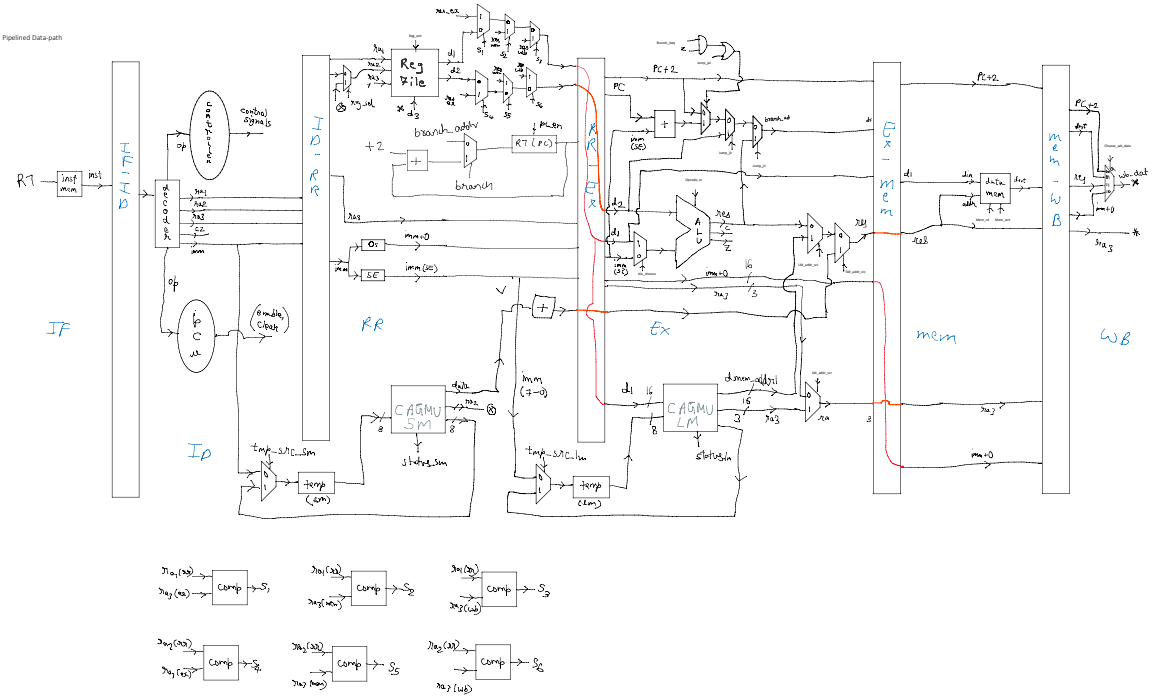
IITB RISC-22 Pipelined

K Akhilesh Rao (213079018)
Siddhant Singh Tomar (213079010)
Dhruva S Hegde (213079022)

Pipelined Datapath



Control Signals

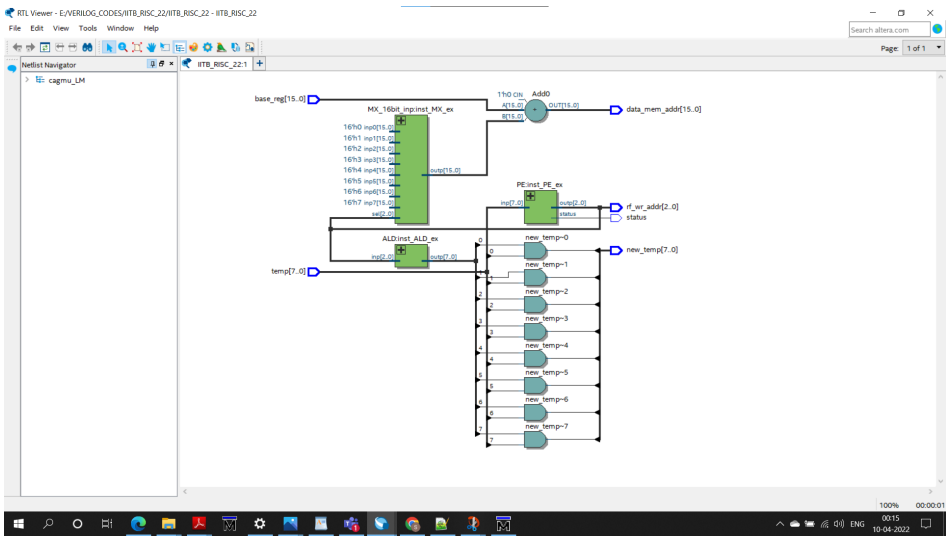
The following list consists of all control signals generated by the Controller and their purpose.

1. pc_en : Enable PC increment
2. reg_wrt : Enable Register write
3. alu_choose : Choose either immediate data or register file content as ALU operand
4. mem_wrt : Enable Memory write
5. mem_rd : Enable Read write
6. $choose\_wb\_data$: Choose among the 4 write back options (ALU result, data from memory, immediate data, PC+1)
7. $branch\_beq$: Indicates BEQ
8. $jump\_jal$: Indicates JAL
9. $jump\_jlr$: Indicates JLR
10. $jump\_jri$: Indicates JRI
11. $temp\_LM_src$: Select source for LM, LA instructions
12. $temp\_SM\_src$: Select source for SM, SA instructions
13. LM_addr_src : Select address for LM, LA instructions
14. SM_addr_src : Select address for SM, SA instructions
15. reg_SM : Select register for SM, SA instructions

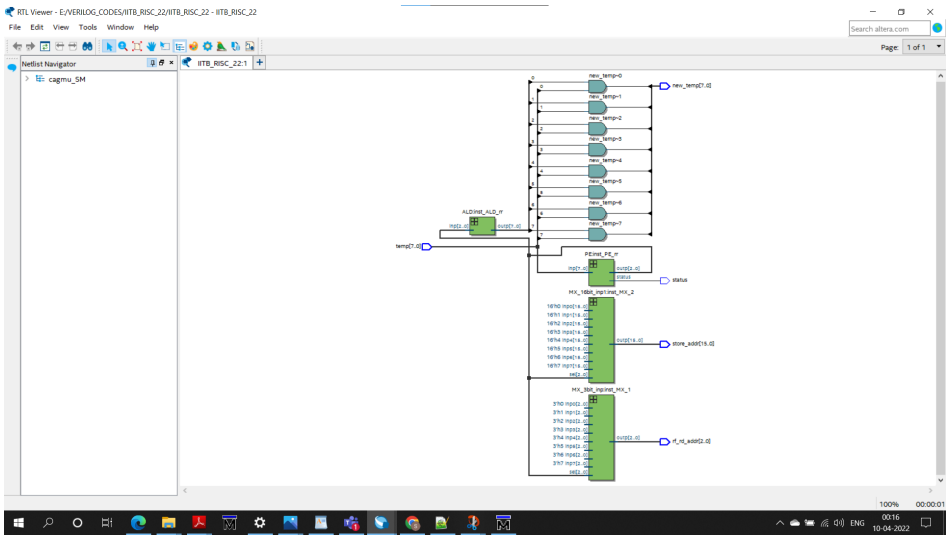
5 enable signals and 5 clear signals for each pipelined register are given from "Pipeline Control Unit (PCU)" for either disabling or flushing their contents.

Implementation of LM, SM, LA and SA

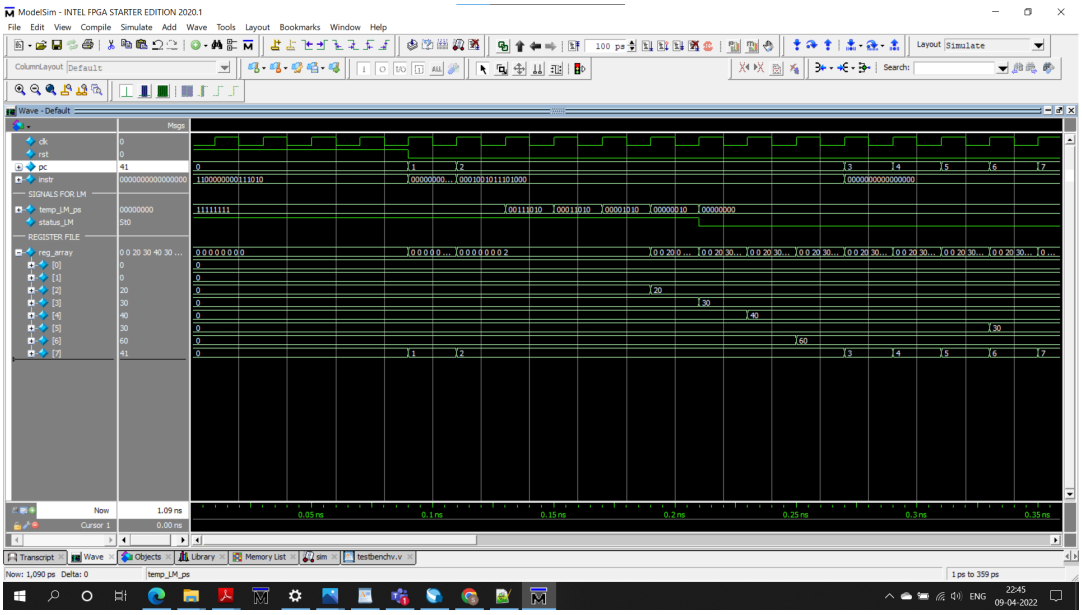
The image below shows the RTL view of the combinational logic used to implement LM and LA instructions.

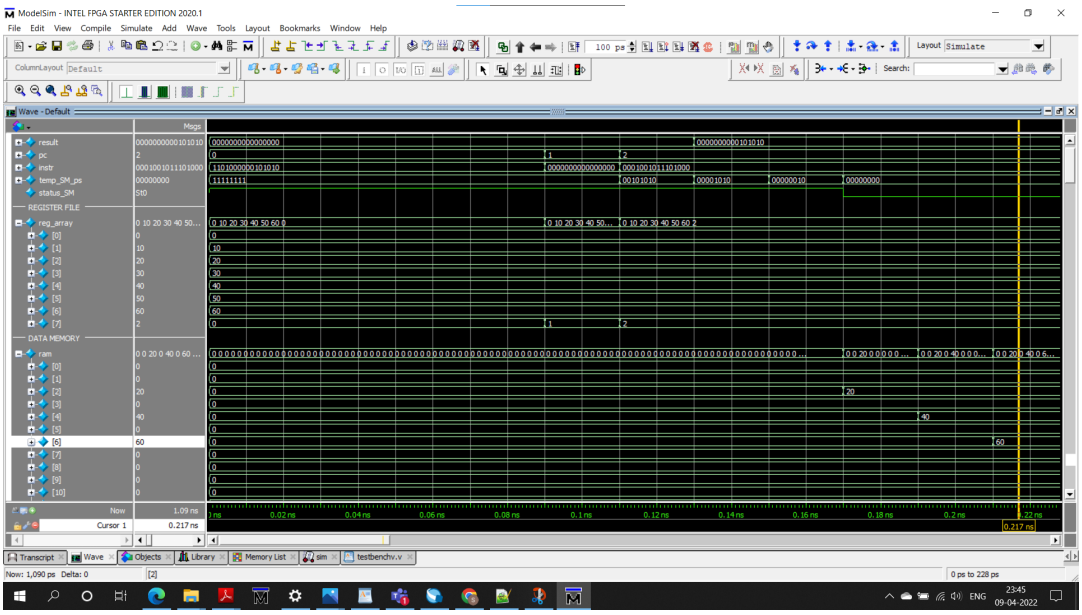


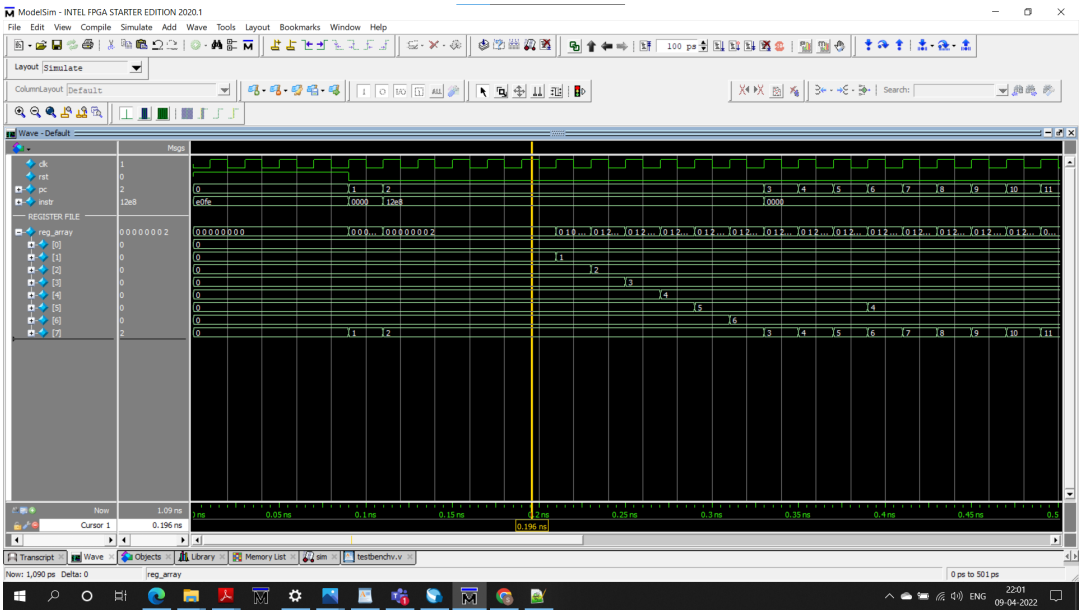
The image below shows the RTL view of the combinational logic used to implement SM and SA instructions.

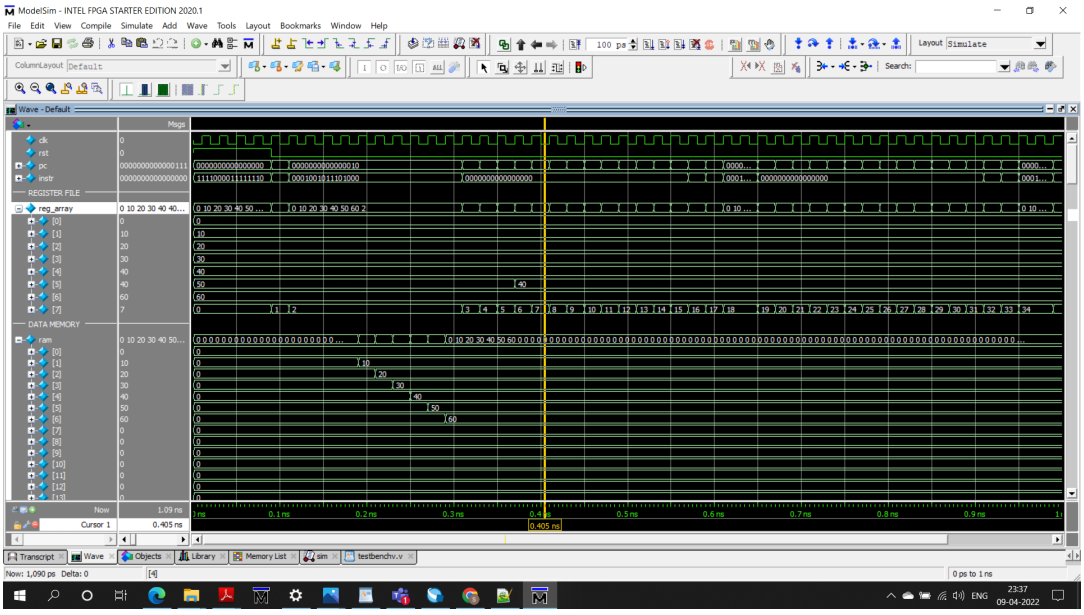


The following images show the simulation results of LM, SM, LA and SA instructions respectively.






ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Msgs
clk
rst
pc
instr
REGISTER FILE
reg_array
Now
Cursor 1
Transcript
Wave
Objects
Library
Memory List
sim
testbenchv.v
reg_array
ENG
22:01
09-04-2022


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Msgs
clk
rst
pc
instr
REGISTER FILE
reg_array
DATA MEMORY
ram
Now
Cursor 1
Transcript
Wave
Objects
Library
Memory List
sim
testbenchv.v
ENG
23:37
09-04-2022

Branch Prediction

An FSM for 2 bit branch prediction for BEQ instruction has been made.
Also, look up tables for PC as well as branch target address have been created.

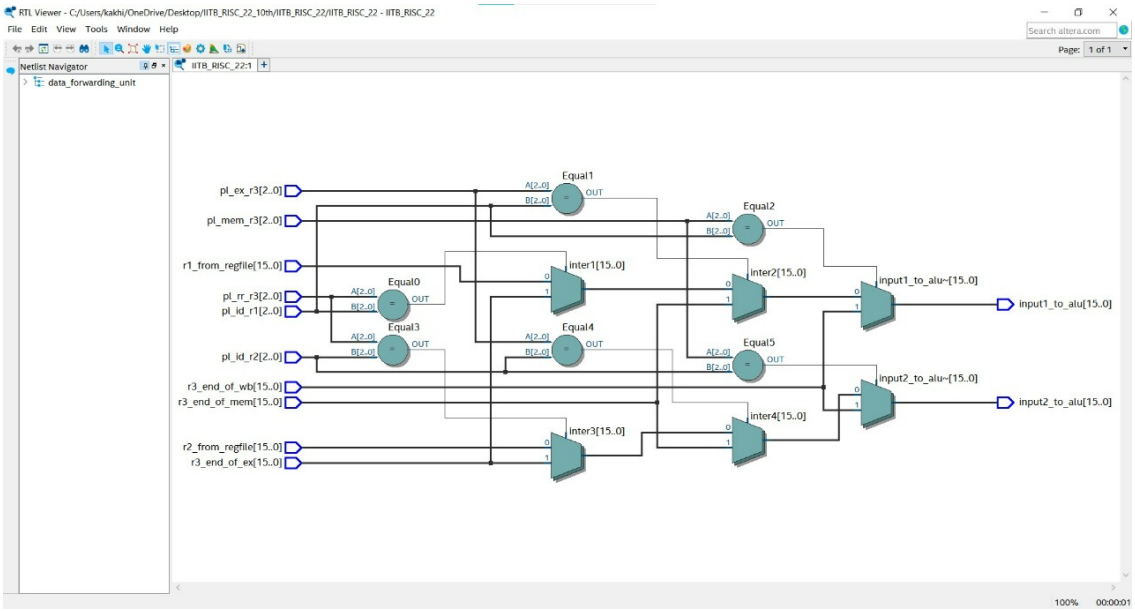
These units have not been included in the processor design.

Data Forwarding

Data forwarding has been implemented in order to resolve Data Hazards.

As illustrated in the data-path, comparators have been used to compare both the sources registers present in RR stage with the destination register in Ex stage, Mem stage and WB stage and detect data dependency.
If data dependency is detected between RR stage and any of the succeeding stages, the result present in that stage will be directly forwarded to RR stage using multiplexers.

RTL view of Data Forwarding Unit:

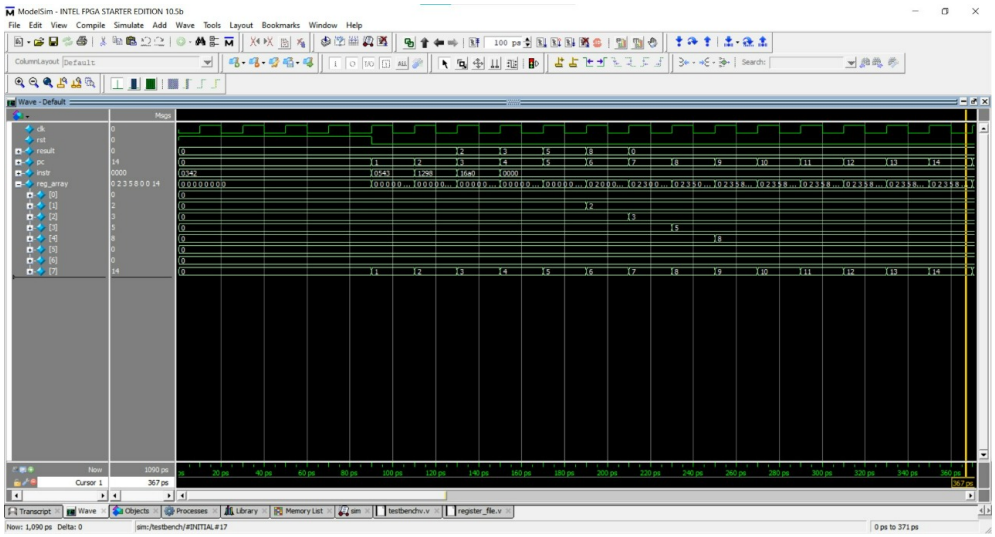


The below images show the instructions and the results obtained to illustrated data dependency and forwarding.

IITB_RISC_22.v | inst_mem.v | Compilation Report - IITB_RISC_22

```
1  module inst_mem (input [15:0] pc, output wire [15:0] instruction);
2
3     wire [3:0] rom_addr = pc [3:0];
4     reg [15:0] rom [15:0];
5
6     initial
7     begin
8        rom[0] = 16'b0000_001_101_000010;  //adi r1 r4 2
9        rom[1] = 16'b0000_010_101_000011;     //adi r2 r4 3
10       rom[2] = 16'b0001_001_010_011_0_00;   //add r3 r1 r2
11       rom[3] = 16'b0001_011_010_100_0_00;   //add r5 r3 r2
12       rom[4] = 16'b0000000000000000;
13       rom[5] = 16'b0000000000000000;
14       rom[6] = 16'b0000000000000000;
15       rom[7] = 16'b0000000000000000;
16       rom[8] = 16'b0000000000000000;
17       rom[9] = 16'b0000000000000000;
18       rom[10] = 16'b0000000000000000;
19       rom[11] = 16'b0000000000000000;
20       rom[12] = 16'b0000000000000000;
21       rom[13] = 16'b0000000000000000;
22       rom[14] = 16'b0000000000000000;
23       rom[15] = 16'b0000000000000000;
24    end
25
26    assign instruction = rom [rom_addr];
27
28 endmodule
```

Note that there are 3 dependencies in the above program.
This also illustrates the usage of R-type instructions.



Simulation of other simple instructions

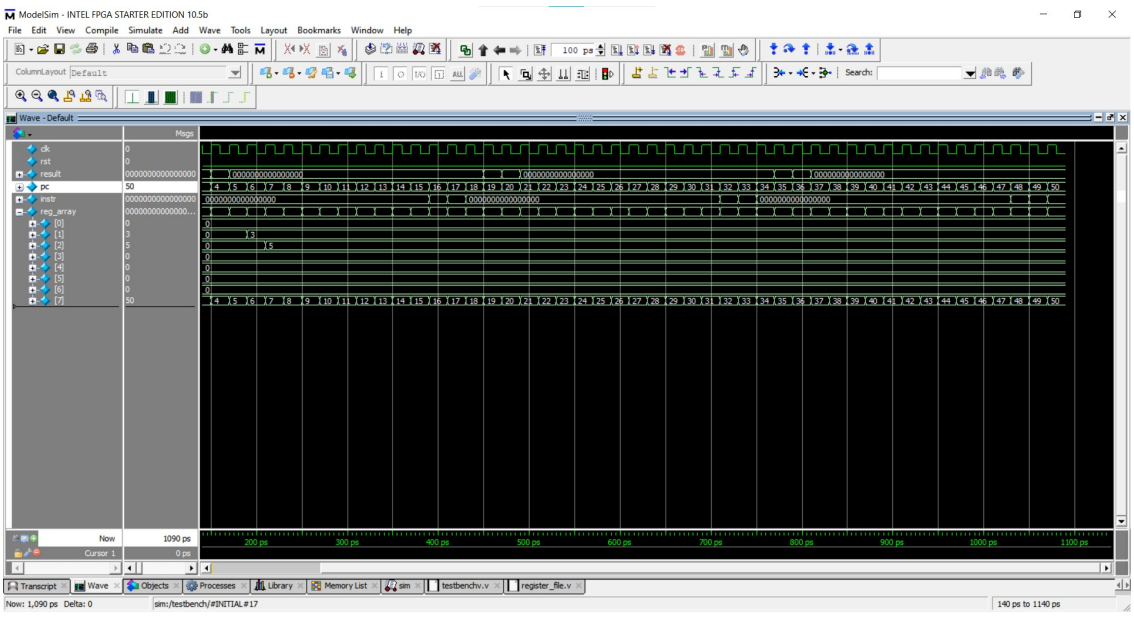
ADI and LW:

Instructions used-

IITB_RISC_22.v | inst_mem.v | Compilation Report - IITB_RISC_22 | inst_decoder.v

```
1  module inst_mem (input [15:0] pc, output wire [15:0] instruction);
2
3     wire [3:0] rom_addr = pc [3:0];
4     reg [15:0] rom [15:0];
5
6     initial
7     begin
8        rom[0] = 16'b0000_001_000_000011;  //adi r1 3 3
9        rom[1] = 16'b0100_010_001_000010;  //lw r2 r1 2
10       rom[2] = 16'b0000000000000000;
11       rom[3] = 16'b0000000000000000;
12       rom[4] = 16'b0000000000000000;
13       rom[5] = 16'b0000000000000000;
14       rom[6] = 16'b0000000000000000;
15       rom[7] = 16'b0000000000000000;
16       rom[8] = 16'b0000000000000000;
17       rom[9] = 16'b0000000000000000;
18       rom[10] = 16'b0000000000000000;
19       rom[11] = 16'b0000000000000000;
20       rom[12] = 16'b0000000000000000;
21       rom[13] = 16'b0000000000000000;
22       rom[14] = 16'b0000000000000000;
23       rom[15] = 16'b0000000000000000;
24
```

Result obtained-



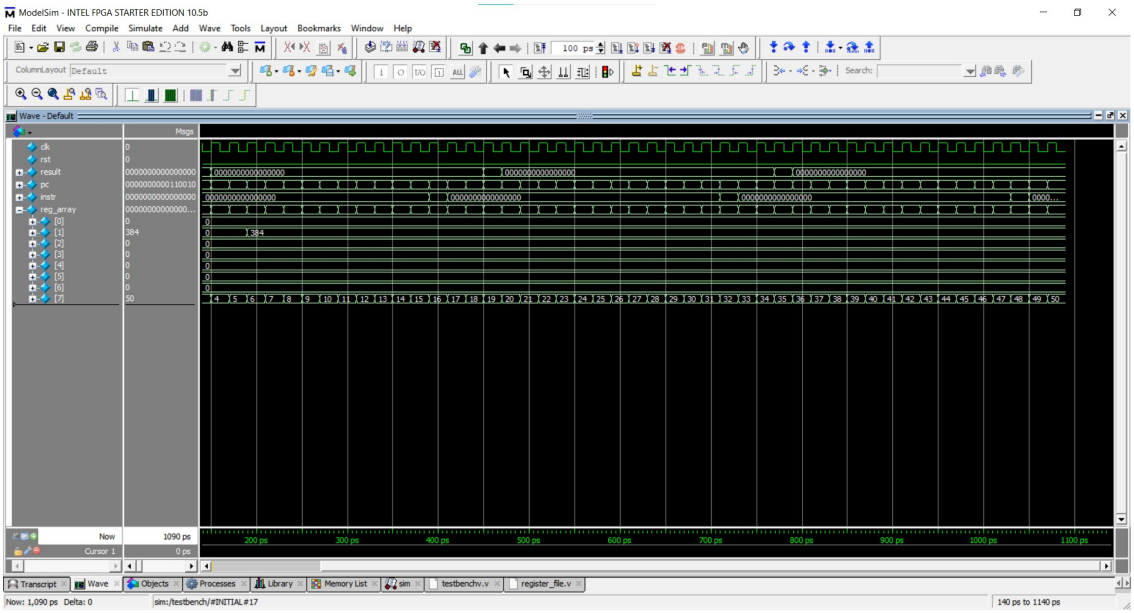
LHI:

Instructions used-

IITB_RISC_22.v | inst_mem.v | Compilation Report - IITB_RISC_22 | inst_decoder.v

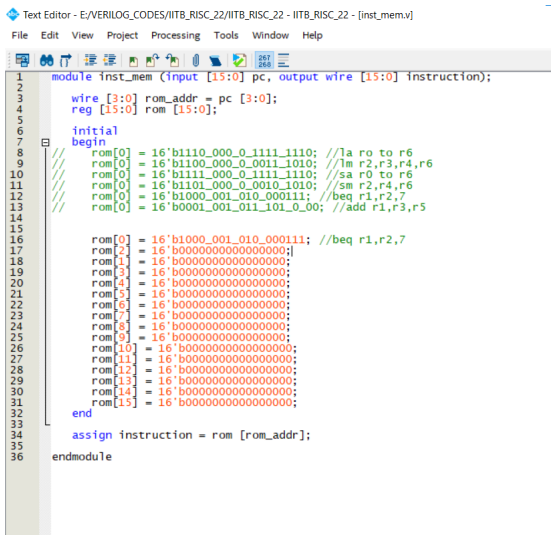
```
1  module inst_mem (input [15:0] pc, output wire [15:0] instruction);
2
3     wire [3:0] rom_addr = pc [3:0];
4     reg [15:0] rom [15:0];
5
6     initial
7     begin
8        rom[0] = 16'b0011_001_000000011;  //lhi r1 3
9        rom[1] = 16'b0000000000000000 ;
10       rom[2] = 16'b0000000000000000;
11       rom[3] = 16'b0000000000000000;
12       rom[4] = 16'b0000000000000000;
13       rom[5] = 16'b0000000000000000;
14       rom[6] = 16'b0000000000000000;
15       rom[7] = 16'b0000000000000000;
16       rom[8] = 16'b0000000000000000;
17       rom[9] = 16'b0000000000000000;
18       rom[10] = 16'b0000000000000000;
19       rom[11] = 16'b0000000000000000;
20       rom[12] = 16'b0000000000000000;
21       rom[13] = 16'b0000000000000000;
22       rom[14] = 16'b0000000000000000;
23       rom[15] = 16'b0000000000000000;
24
```

Result obtained-

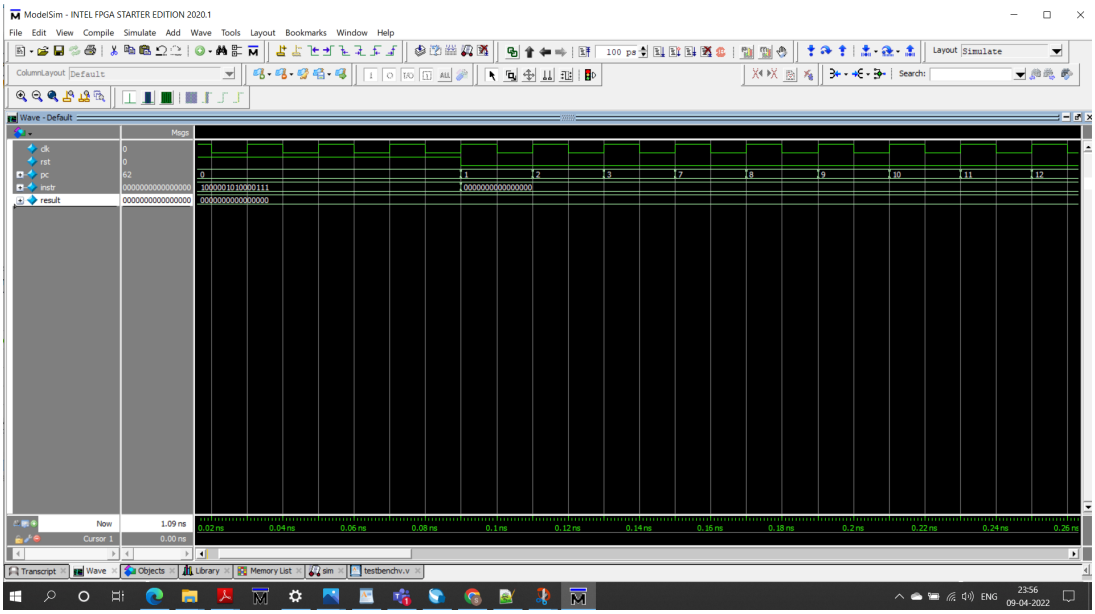


BEQ:

Instructions used-



Result obtained-



SW:

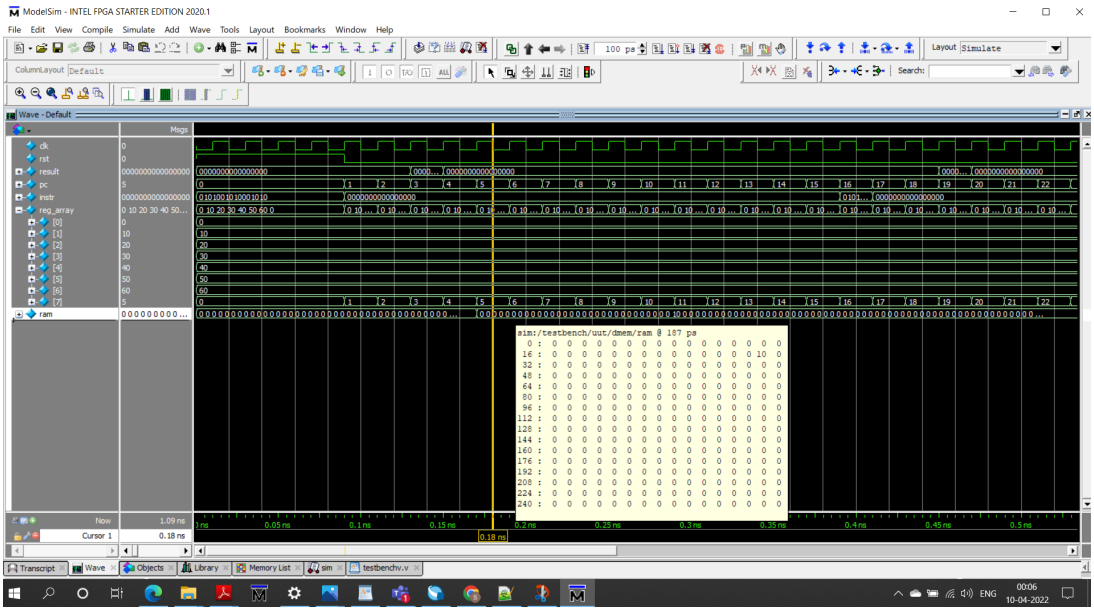
Instructions used-

Text Editor - E:/VERILOG_CODES/IITB_RISC_22/IITB_RISC_22 - IITB_RISC_22 - [inst_mem.v]

File Edit View Project Processing Tools Window Help

```
module inst_mem (input [15:0] pc, output wire [15:0] instruction);

   wire [3:0] rom_addr = pc [3:0];
   reg [15:0] rom [15:0];

   initial
   begin
//    rom[0] = 16'b1110_000_0_1111_1110; //la r0 to r6
//    rom[0] = 16'b1100_000_0_0011_1010; //lm r2,r3,r4,r6
//    rom[0] = 16'b1111_000_0_1111_1110; //sa r0 to r6
//    rom[0] = 16'b1101_000_0_0010_1010; //sm r2,r4,r6
//    rom[0] = 16'b1000_001_010_000111; //beq r1,r2,7
//    rom[0] = 16'b0001_001_011_101_0_00; //add r1,r3,r5


      rom[0] = 16'b0101_001_010_001010; //sw r1,r2,10
      rom[2] = 16'b0001_001_011_101_0_00; //add r1,r3,r5
      rom[1] = 16'b0000000000000000;
      rom[3] = 16'b0000000000000000;
      rom[4] = 16'b0000000000000000;
      rom[5] = 16'b0000000000000000;
      rom[6] = 16'b0000000000000000;
      rom[7] = 16'b0000000000000000;
      rom[8] = 16'b0000000000000000;
      rom[9] = 16'b0000000000000000;
      rom[10] = 16'b0000000000000000;
      rom[11] = 16'b0000000000000000;
      rom[12] = 16'b0000000000000000;
      rom[13] = 16'b0000000000000000;
      rom[14] = 16'b0000000000000000;
      rom[15] = 16'b0000000000000000;
   end

   assign instruction = rom [rom_addr];

endmodule
```

Result obtained-



JRI:

Instructions used-

IITB_RISC_22.v | inst_mem.v | Compilation Report - IITB_RISC_22 | inst_decoder.v

```
1  module inst_mem (input [15:0] pc, output wire [15:0] instruction);
2
3     wire [3:0] rom_addr = pc [3:0];
4     reg [15:0] rom [15:0];
5
6     initial
7     begin
8        rom[0] = 16'b1011_001_000000111;  //jri r1 7
9        rom[1] = 16'b0000000000000000 ;
10       rom[2] = 16'b0000000000000000;
11       rom[3] = 16'b0000000000000000;
12       rom[4] = 16'b0000000000000000;
13       rom[5] = 16'b0000000000000000;
14       rom[6] = 16'b0000000000000000;
15       rom[7] = 16'b0000000000000000;
16       rom[8] = 16'b0000000000000000;
17       rom[9] = 16'b0000000000000000;
18       rom[10] = 16'b0000000000000000;
19       rom[11] = 16'b0000000000000000;
20       rom[12] = 16'b0000000000000000;
21       rom[13] = 16'b0000000000000000;
22       rom[14] = 16'b0000000000000000;
23       rom[15] = 16'b0000000000000000;
24
```

Result obtained-