Introduction to Digital Logic Design Lab
EECS 31L

Lab #5

Steven Tabilisma
16326339

03/13/22

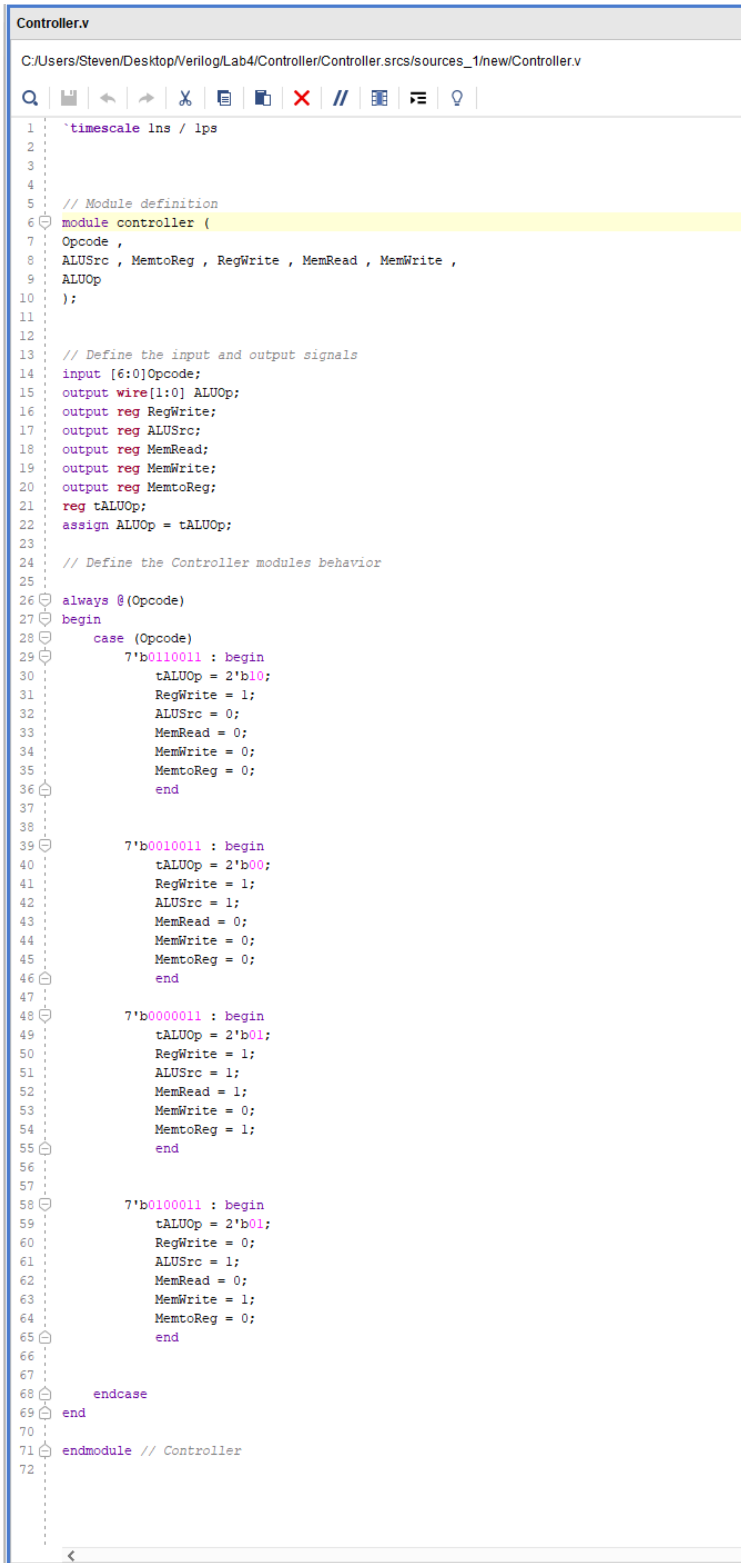
1 Objective

The objective of this lab was to create the final two lower level controller modules for the processor and ALU, and then implement them in order to finally complete the RISC-V processor.

2 Procedure

Controller

The design of the lower level modules in this lab were very simple. While it was suggested to use concurrent assignments, I made use of a procedural block of case statements to implement this module. Using the simple table provided to us, I read the opcode then output the correct values to the 6 outputs.

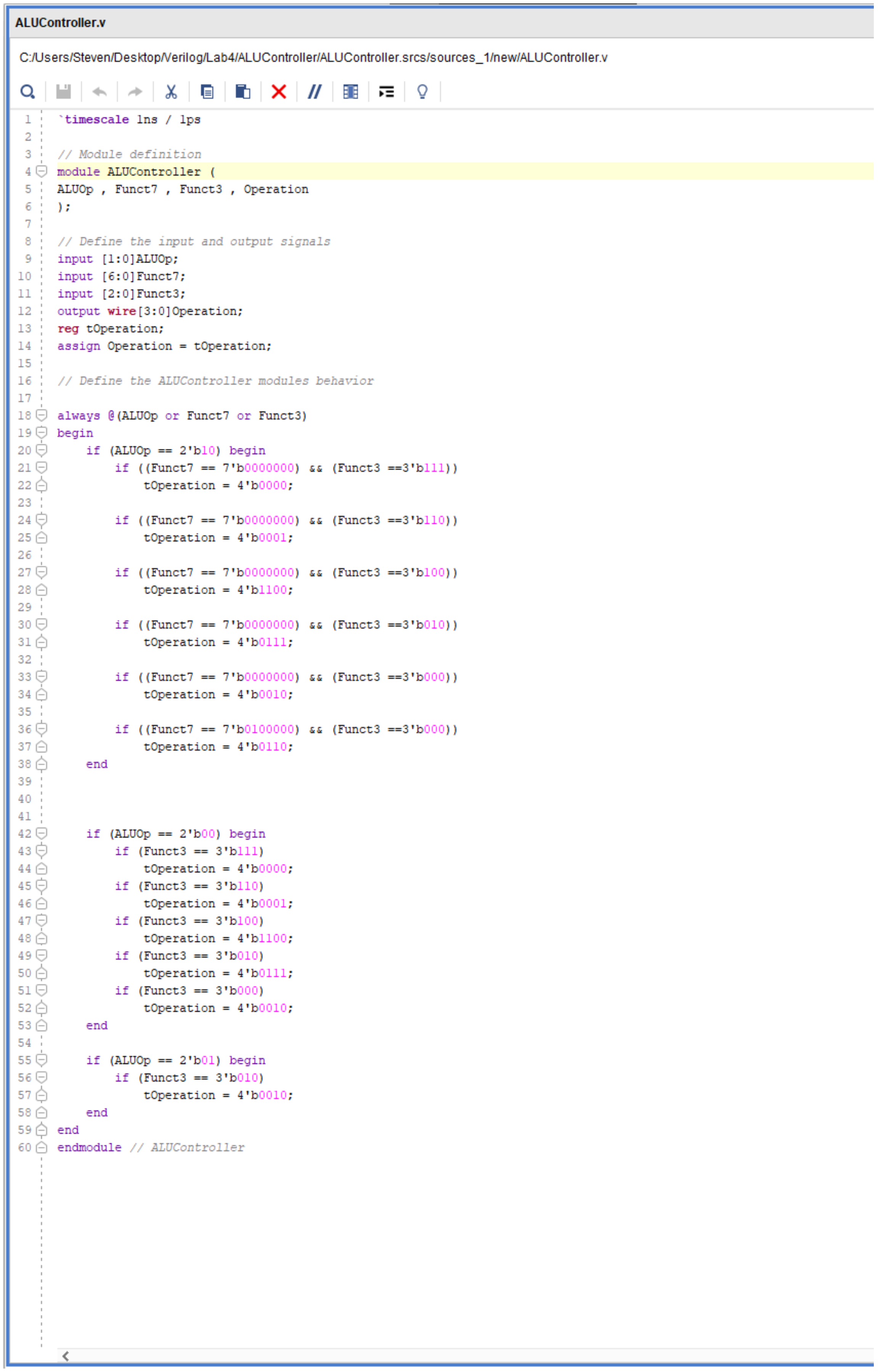

Controller.v

C:/Users/Steven/Desktop/Verilog/Lab4/Controller/Controller.srcs/sources_1/new/Controller.v

```
`timescale 1ns / 1ps



// Module definition
module controller (
Opcode ,
ALUSrc , MemtoReg , RegWrite , MemRead , MemWrite ,
ALUOp
);


// Define the input and output signals
input [6:0]Opcode;
output wire[1:0] ALUOp;
output reg RegWrite;
output reg ALUSrc;
output reg MemRead;
output reg MemWrite;
output reg MemtoReg;
reg tALUOp;
assign ALUOp = tALUOp;

// Define the Controller modules behavior

always @(Opcode)
begin
    case (Opcode)
        7'b0110011 : begin
            tALUOp = 2'b10;
            RegWrite = 1;
            ALUSrc = 0;
            MemRead = 0;
            MemWrite = 0;
            MemtoReg = 0;
            end


        7'b0010011 : begin
            tALUOp = 2'b00;
            RegWrite = 1;
            ALUSrc = 1;
            MemRead = 0;
            MemWrite = 0;
            MemtoReg = 0;
            end

        7'b0000011 : begin
            tALUOp = 2'b01;
            RegWrite = 1;
            ALUSrc = 1;
            MemRead = 1;
            MemWrite = 0;
            MemtoReg = 1;
            end


        7'b0100011 : begin
            tALUOp = 2'b01;
            RegWrite = 0;
            ALUSrc = 1;
            MemRead = 0;
            MemWrite = 1;
            MemtoReg = 0;
            end


    endcase
end

endmodule // Controller
```

ALU Controller

The ALU controller was also very similar to the Controller module in terms of simplicity. Rather than using case statements however, I simply designed use three if conditionals with nested conditions within them. The outer tree was based on the ALUop input and based on this I would check for which funct3 input was given, and then funct7 if necessary. I would simply output the corresponding operation code.

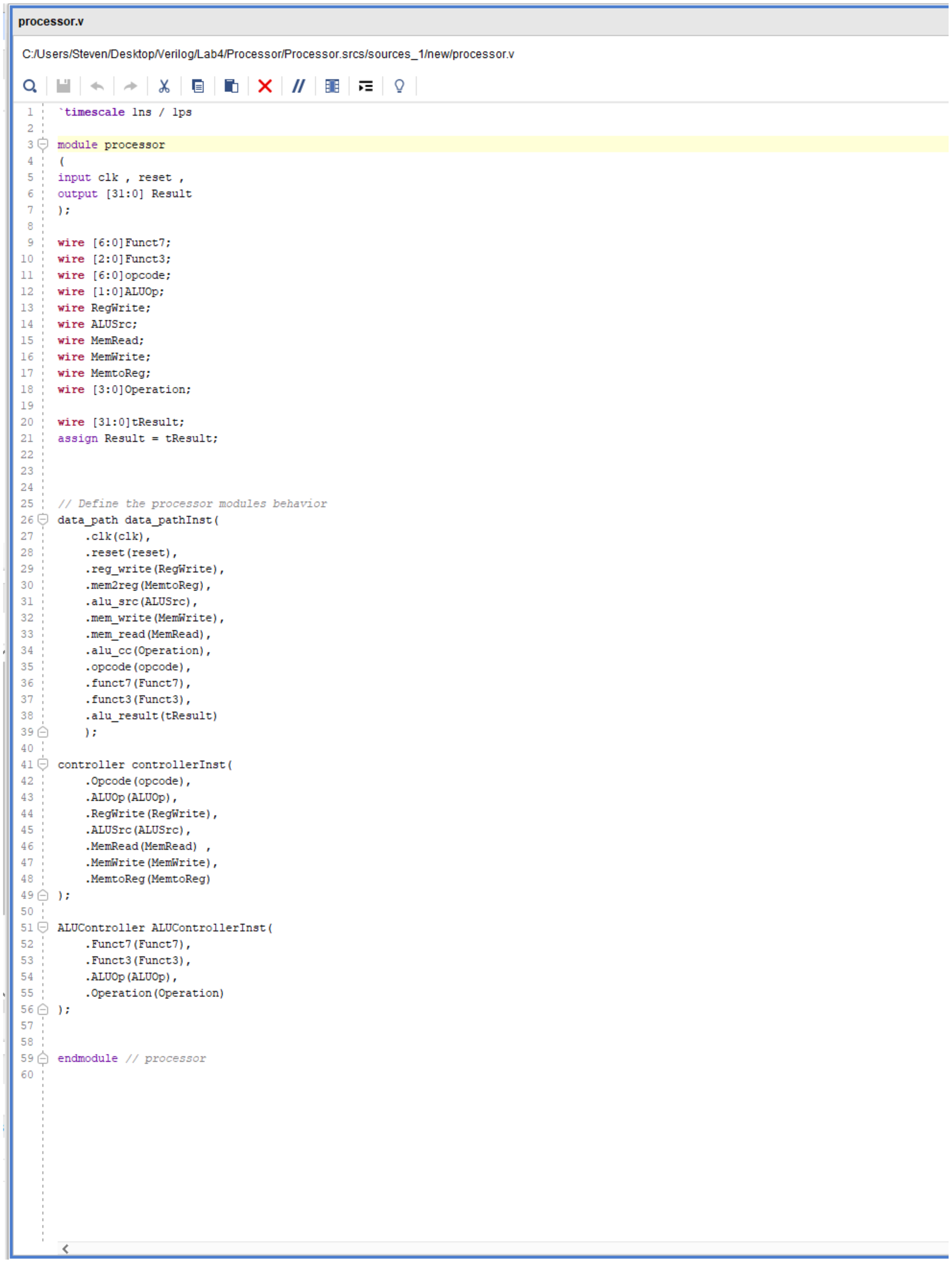

ALUController.v

C:/Users/Steven/Desktop/Verilog/Lab4/ALUController/ALUController.srcs/sources_1/new/ALUController.v

```verilog
`timescale 1ns / 1ps

// Module definition
module ALUController (
ALUOp , Funct7 , Funct3 , Operation
);

// Define the input and output signals
input [1:0]ALUOp;
input [6:0]Funct7;
input [2:0]Funct3;
output wire[3:0]Operation;
reg tOperation;
assign Operation = tOperation;

// Define the ALUController modules behavior

always @(ALUOp or Funct7 or Funct3)
begin
    if (ALUOp == 2'b10) begin
        if ((Funct7 == 7'b0000000) && (Funct3 ==3'b111))
            tOperation = 4'b0000;

        if ((Funct7 == 7'b0000000) && (Funct3 ==3'b110))
            tOperation = 4'b0001;

        if ((Funct7 == 7'b0000000) && (Funct3 ==3'b100))
            tOperation = 4'b1100;

        if ((Funct7 == 7'b0000000) && (Funct3 ==3'b010))
            tOperation = 4'b0111;

        if ((Funct7 == 7'b0000000) && (Funct3 ==3'b000))
            tOperation = 4'b0010;

        if ((Funct7 == 7'b0100000) && (Funct3 ==3'b000))
            tOperation = 4'b0110;
    end



    if (ALUOp == 2'b00) begin
        if (Funct3 == 3'b111)
            tOperation = 4'b0000;
        if (Funct3 == 3'b110)
            tOperation = 4'b0001;
        if (Funct3 == 3'b100)
            tOperation = 4'b1100;
        if (Funct3 == 3'b010)
            tOperation = 4'b0111;
        if (Funct3 == 3'b000)
            tOperation = 4'b0010;
    end

    if (ALUOp == 2'b01) begin
        if (Funct3 == 3'b010)
            tOperation = 4'b0010;
    end
end
endmodule // ALUController
```

Processor

This was the final and complete processor essentially but interestingly enough it was not as hard to implement as the data path was. I suppose this was because it contained far less modules. I simply wired the necessary components together and connected the clock and reset inputs, then made sure datapath was connected to the output of the module. After that the processor was complete and after running the provided testbench it was done.

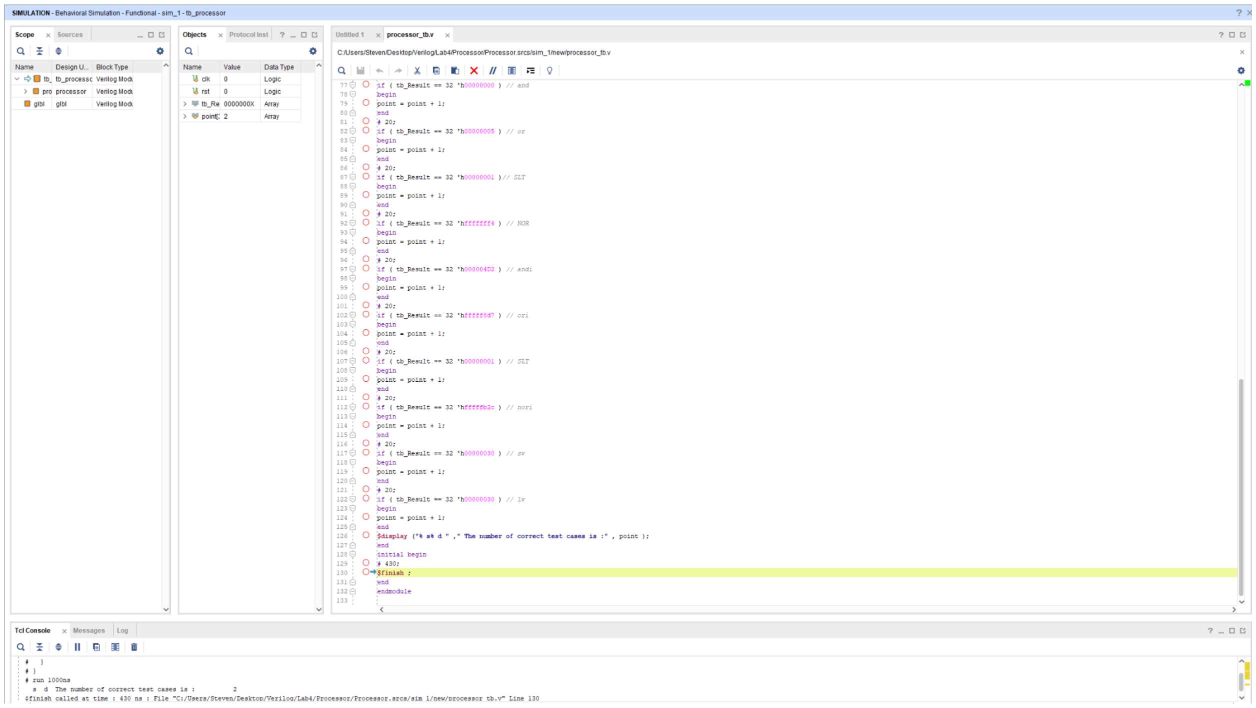


processor.v

C:/Users/Steven/Desktop/Verilog/Lab4/Processor/Processor.srcs/sources_1/new/processor.v

```verilog
`timescale 1ns / 1ps

module processor
(
input clk , reset ,
output [31:0] Result
);

wire [6:0]Funct7;
wire [2:0]Funct3;
wire [6:0]opcode;
wire [1:0]ALUOp;
wire RegWrite;
wire ALUSrc;
wire MemRead;
wire MemWrite;
wire MemtoReg;
wire [3:0]Operation;

wire [31:0]tResult;
assign Result = tResult;



// Define the processor modules behavior
data_path data_pathInst(
    .clk(clk),
    .reset(reset),
    .reg_write(RegWrite),
    .mem2reg(MemtoReg),
    .alu_src(ALUSrc),
    .mem_write(MemWrite),
    .mem_read(MemRead),
    .alu_cc(Operation),
    .opcode(opcode),
    .funct7(Funct7),
    .funct3(Funct3),
    .alu_result(tResult)
    );

controller controllerInst(
    .Opcode(opcode),
    .ALUOp(ALUOp),
    .RegWrite(RegWrite),
    .ALUSrc(ALUSrc),
    .MemRead(MemRead) ,
    .MemWrite(MemWrite),
    .MemtoReg(MemtoReg)
);

ALUController ALUControllerInst(
    .Funct7(Funct7),
    .Funct3(Funct3),
    .ALUOp(ALUOp),
    .Operation(Operation)
);


endmodule // processor

```

3 Simulation Results



SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_processor

Name	Design U...	Block Type
tb_	tb_processo	Verilog Modu
pro	processor	Verilog Modu
glbl	glbl	Verilog Modu

Name	Value	Data Type
clk	0	Logic
rst	0	Logic
tb_Re	0000000X	Array
point[	2	Array

C:/Users/Steven/Desktop/Verilog/Lab4/Processor/Processor.srcs/sim_1/new/processor_tb.v

```
 77 if ( tb_Result == 32 'h00000000 ) // and
 78 begin
 79 point = point + 1;
 80 end
 81 # 20;
 82 if ( tb_Result == 32 'h00000005 ) // or
 83 begin
 84 point = point + 1;
 85 end
 86 # 20;
 87 if ( tb_Result == 32 'h00000001 )// SLT
 88 begin
 89 point = point + 1;
 90 end
 91 # 20;
 92 if ( tb_Result == 32 'hfffffff4 ) // NOR
 93 begin
 94 point = point + 1;
 95 end
 96 # 20;
 97 if ( tb_Result == 32 'h000004D2 ) // andi
 98 begin
 99 point = point + 1;
100 end
101 # 20;
102 if ( tb_Result == 32 'hfffff8d7 ) // ori
103 begin
104 point = point + 1;
105 end
106 # 20;
107 if ( tb_Result == 32 'h00000001 ) // SLT
108 begin
109 point = point + 1;
110 end
111 # 20;
112 if ( tb_Result == 32 'hfffffb2c ) // nori
113 begin
114 point = point + 1;
115 end
116 # 20;
117 if ( tb_Result == 32 'h00000030 ) // sw
118 begin
119 point = point + 1;
120 end
121 # 20;
122 if ( tb_Result == 32 'h00000030 ) // lw
123 begin
124 point = point + 1;
125 end
126 $display ("% s% d " ," The number of correct test cases is :" , point );
127 end
128 initial begin
129 # 430;
130 $finish ;
131 end
132 endmodule
133
```

Tcl Console

```
# }
# }
# run 1000ns
  s  d  The number of correct test cases is :          2
$finish called at time : 430 ns : File "C:/Users/Steven/Desktop/Verilog/Lab4/Processor/Processor.srcs/sim 1/new/processor tb.v" Line 130
```

Processor Implementation

As seen in the screenshot I was only able to pass 2 test cases. The two that I passed were any zero outputs from the datapath. This shows that my higher level implementation is correct, but that one of my lower level modules is incorrect. However, I know that the reason for this failure is in my datapath module and more specifically the regfile. When running the test bench for the datapath module, only two cases successfully pass. Both cases have the result of the ALU as zero. Although I am unsure why, my regfile refuses to store data to the registers, meaning that nothing is able to be read except for the default values of the registers which is zero. Overall, the structure of the processor is correctly implemented, and the controllers are properly designed.