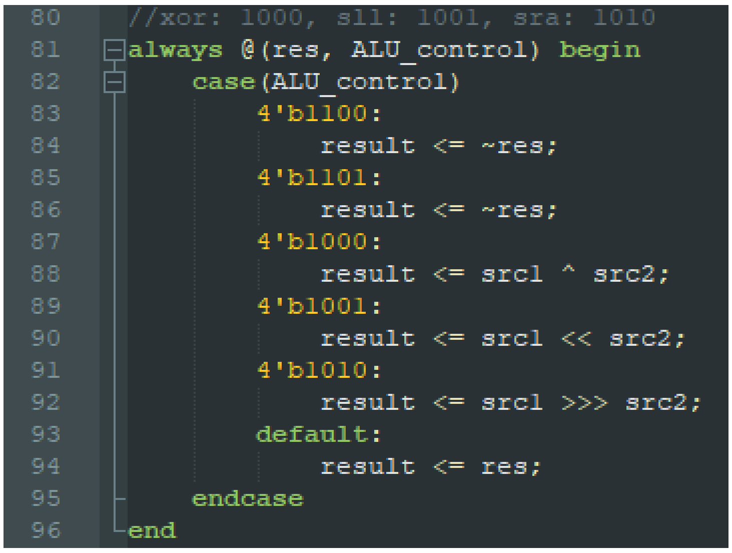
Report

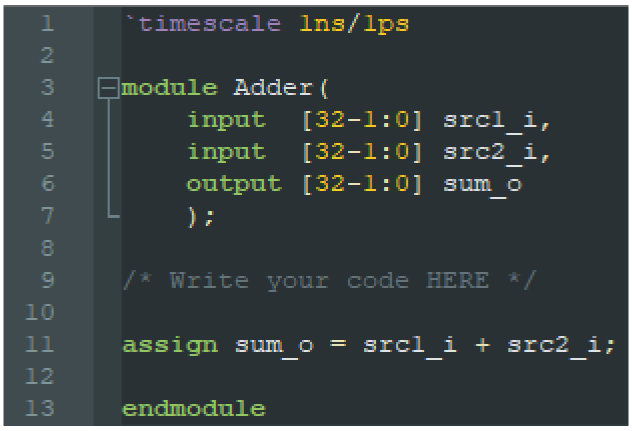
1. Description of the implementation

- Alu.v: added three additional operations.



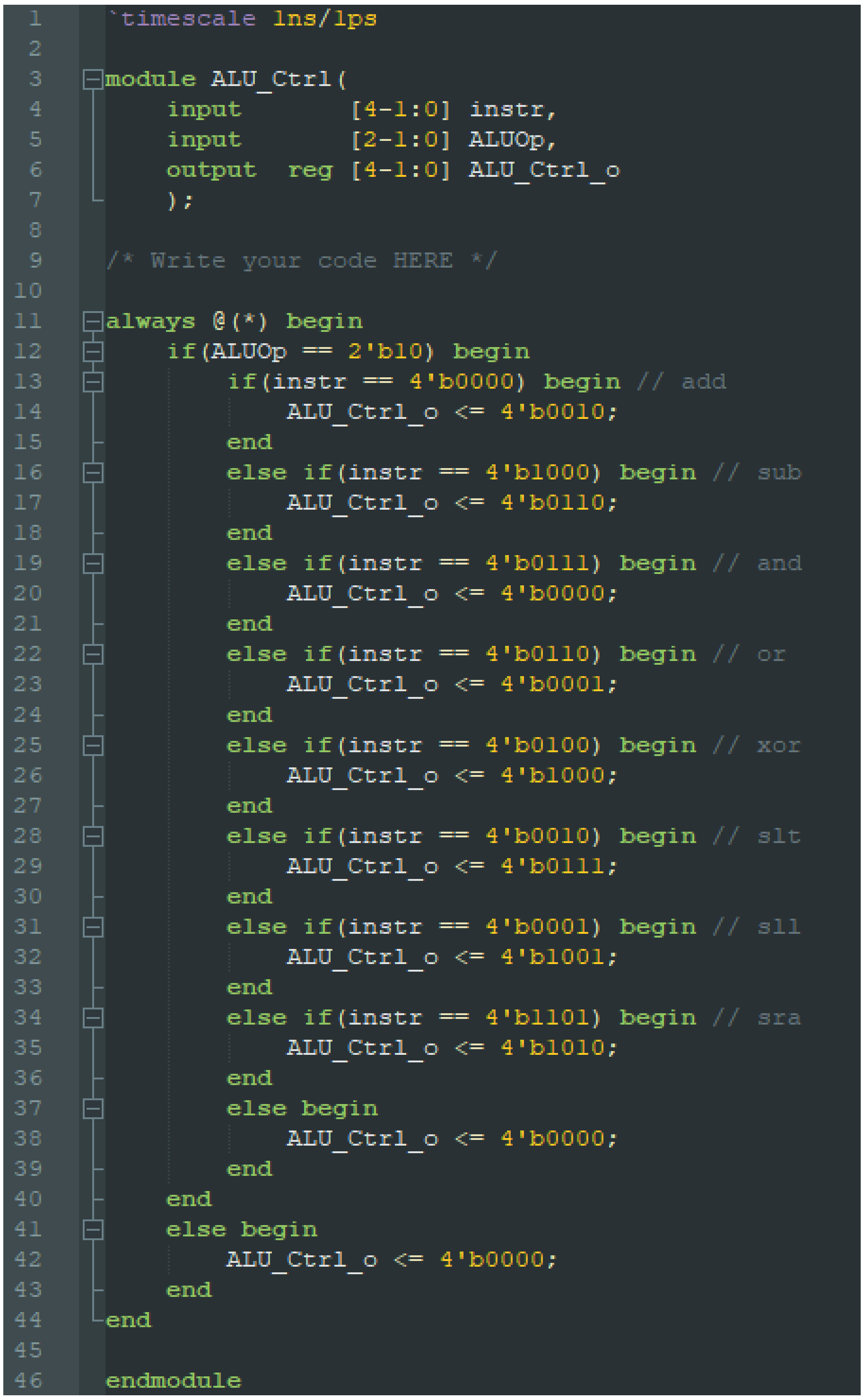
```
//xor: 1000, sll: 1001, sra: 1010
always @(res, ALU_control) begin
    case(ALU_control)
        4'b1100:
            result <= ~res;
        4'b1101:
            result <= ~res;
        4'b1000:
            result <= src1 ^ src2;
        4'b1001:
            result <= src1 << src2;
        4'b1010:
            result <= src1 >>> src2;
        default:
            result <= res;
    endcase
end
```

- Adder.v: assign the sum of two input to output.



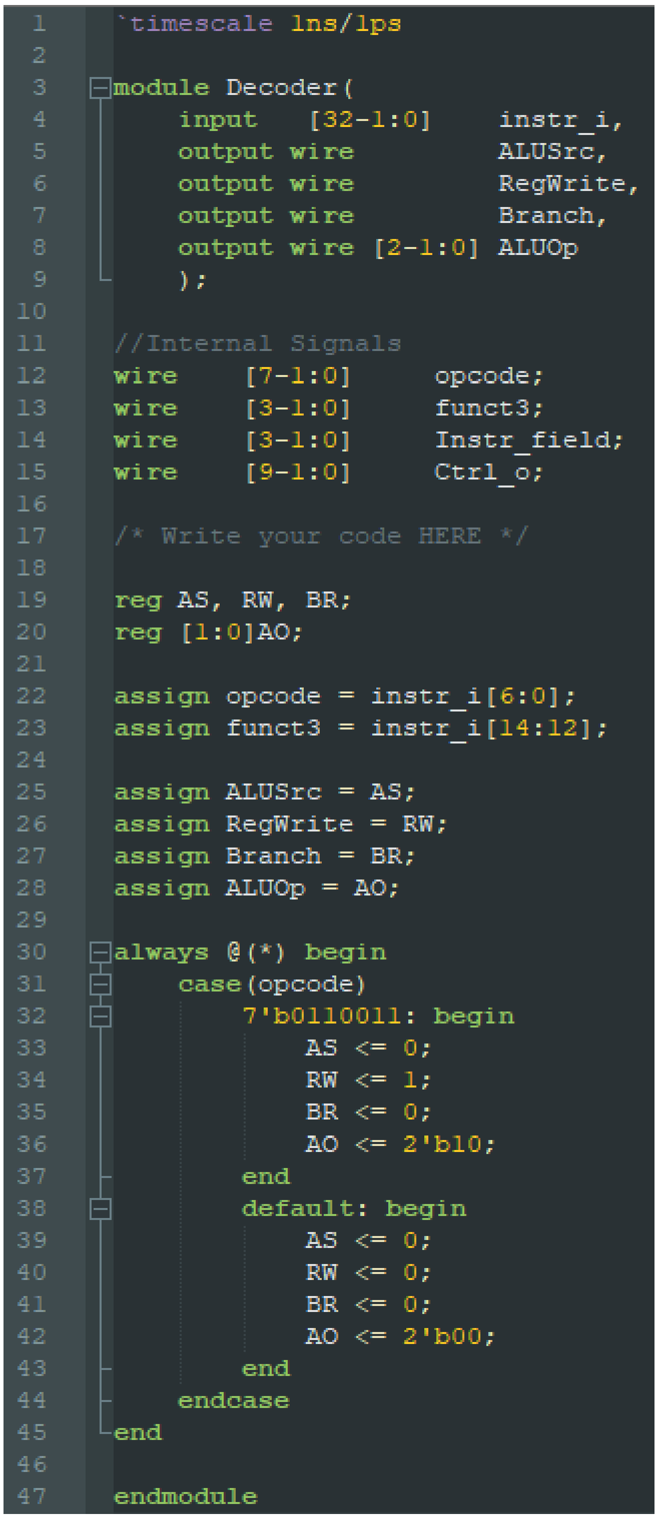
```
`timescale 1ns/1ps

module Adder(
    input  [32-1:0] src1_i,
    input  [32-1:0] src2_i,
    output [32-1:0] sum_o
    );

/* Write your code HERE */

assign sum_o = src1_i + src2_i;

endmodule
```

- ALU_Ctrl.v: use ALUOp and instruction (I30 + func3) to identify each operation.



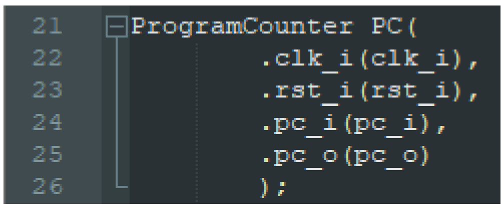
```
`timescale 1ns/1ps

module ALU_Ctrl(
        input        [4-1:0] instr,
        input        [2-1:0] ALUOp,
        output   reg [4-1:0] ALU_Ctrl_o
        );

/* Write your code HERE */

always @(*) begin
        if(ALUOp == 2'b10) begin
                if(instr == 4'b0000) begin // add
                        ALU_Ctrl_o <= 4'b0010;
                end
                else if(instr == 4'b1000) begin // sub
                        ALU_Ctrl_o <= 4'b0110;
                end
                else if(instr == 4'b0111) begin // and
                        ALU_Ctrl_o <= 4'b0000;
                end
                else if(instr == 4'b0110) begin // or
                        ALU_Ctrl_o <= 4'b0001;
                end
                else if(instr == 4'b0100) begin // xor
                        ALU_Ctrl_o <= 4'b1000;
                end
                else if(instr == 4'b0010) begin // slt
                        ALU_Ctrl_o <= 4'b0111;
                end
                else if(instr == 4'b0001) begin // sll
                        ALU_Ctrl_o <= 4'b1001;
                end
                else if(instr == 4'b1101) begin // sra
                        ALU_Ctrl_o <= 4'b1010;
                end
                else begin
                        ALU_Ctrl_o <= 4'b0000;
                end
        end
        else begin
                ALU_Ctrl_o <= 4'b0000;
        end
end

endmodule
```

- Decoder.v: identify each type of instruction by its opcode, which is I[6:0]. Assign corresponding control signal to each output.



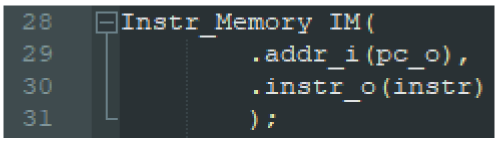
```verilog
`timescale 1ns/1ps

module Decoder(
	input	[32-1:0]	instr_i,
	output wire		ALUSrc,
	output wire		RegWrite,
	output wire		Branch,
	output wire [2-1:0] ALUOp
	);

//Internal Signals
wire	[7-1:0]		opcode;
wire	[3-1:0]		funct3;
wire	[3-1:0]		Instr_field;
wire	[9-1:0]		Ctrl_o;

/* Write your code HERE */

reg AS, RW, BR;
reg [1:0]AO;

assign opcode = instr_i[6:0];
assign funct3 = instr_i[14:12];

assign ALUSrc = AS;
assign RegWrite = RW;
assign Branch = BR;
assign ALUOp = AO;

always @(*) begin
	case(opcode)
		7'b0110011: begin
			AS <= 0;
			RW <= 1;
			BR <= 0;
			AO <= 2'b10;
		end
		default: begin
			AS <= 0;
			RW <= 0;
			BR <= 0;
			AO <= 2'b00;
		end
	endcase
end

endmodule
```

- Simple_Single_CPU.v:
  i. Program Counter: input the calculated pc and the circuit will check if the reset signal is marked, then output the result.



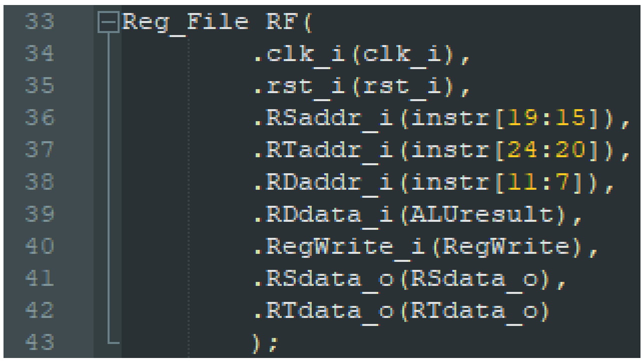
```
ProgramCounter PC(
        .clk_i(clk_i),
        .rst_i(rst_i),
        .pc_i(pc_i),
        .pc_o(pc_o)
        );
```

ii. Instruction Memory: input current pc and get instruction code.



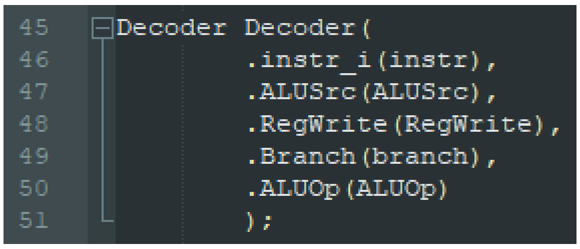
```
Instr_Memory IM(
        .addr_i(pc_o),
        .instr_o(instr)
        );
```

iii. Register File: RSaddr is rs1(I[19:15]), RTaddr is rs2(I[24:20]), and RDaddr is rd(I[11:7]). The write data should be the output of the ALU and controlled by RegWrite signal. The data in RS and RT will be sent to RSdata_o and RDdata_o.



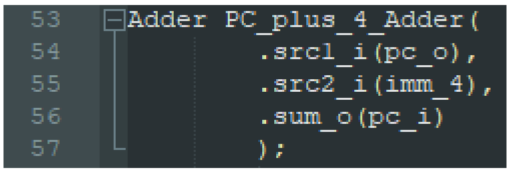
```
Reg_File RF(
        .clk_i(clk_i),
        .rst_i(rst_i),
        .RSaddr_i(instr[19:15]),
        .RTaddr_i(instr[24:20]),
        .RDaddr_i(instr[11:7]),
        .RDdata_i(ALUresult),
        .RegWrite_i(RegWrite),
        .RSdata_o(RSdata_o),
        .RTdata_o(RTdata_o)
        );
```

iv. Decoder: Read the instruction and decode to corresponding control signals.



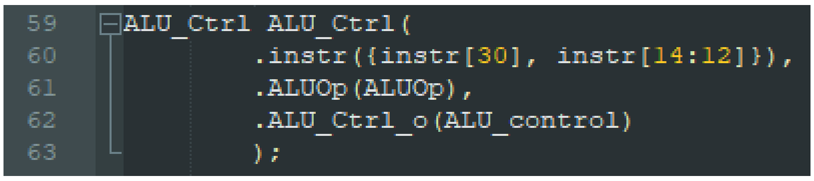
```
Decoder Decoder(
        .instr_i(instr),
        .ALUSrc(ALUSrc),
        .RegWrite(RegWrite),
        .Branch(branch),
        .ALUOp(ALUOp)
        );
```

v. PC plus 4: Input current pc and constant 4 to get pc of next loop.



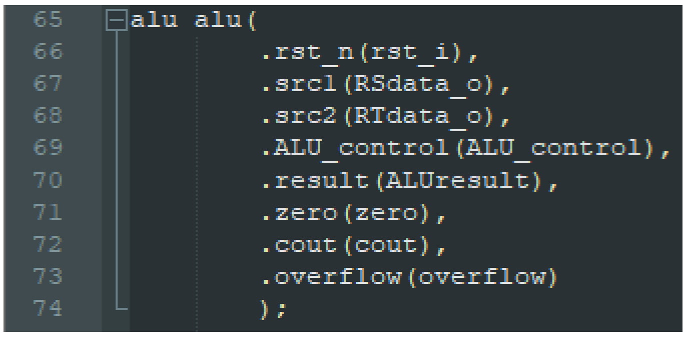
```
Adder PC_plus_4_Adder(
        .src1_i(pc_o),
        .src2_i(imm_4),
        .sum_o(pc_i)
        );
```

vi. ALU Control: Input I30+func3 and ALUOp to get ALU control signal.



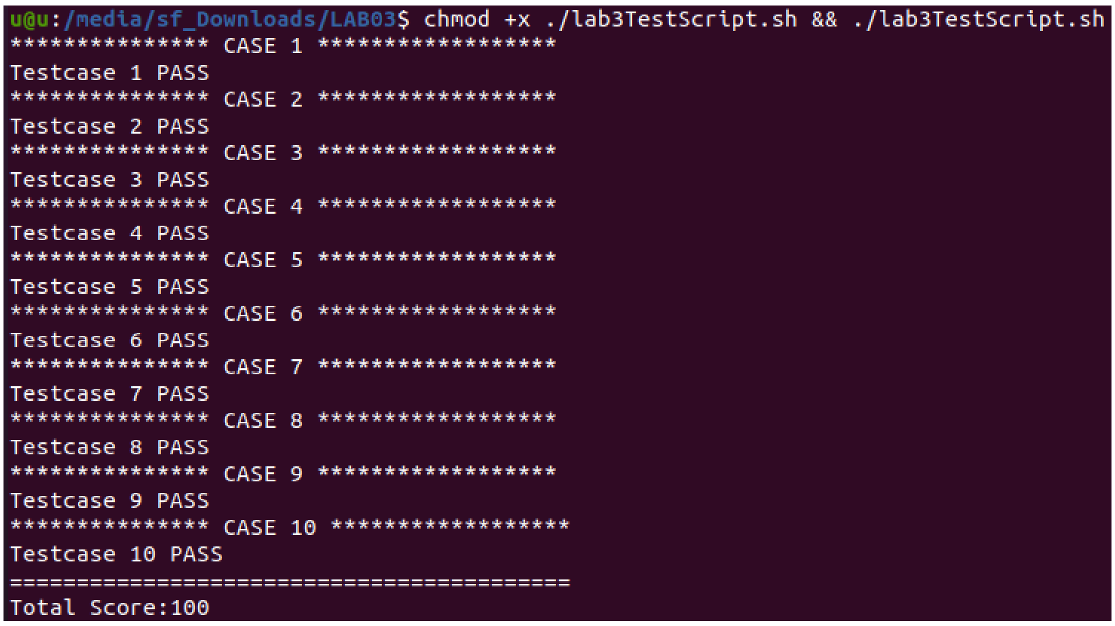
```
ALU_Ctrl ALU_Ctrl(
        .instr({instr[30], instr[14:12]}),
        .ALUOp(ALUOp),
        .ALU_Ctrl_o(ALU_control)
        );
```

vii. ALU: Reads in RSdata, RTdata, and ALU control signal then outputs the ALUresult and ZCV flag.



```
alu alu(
        .rst_n(rst_i),
        .src1(RSdata_o),
        .src2(RTdata_o),
        .ALU_control(ALU_control),
        .result(ALUresult),
        .zero(zero),
        .cout(cout),
        .overflow(overflow)
        );
```

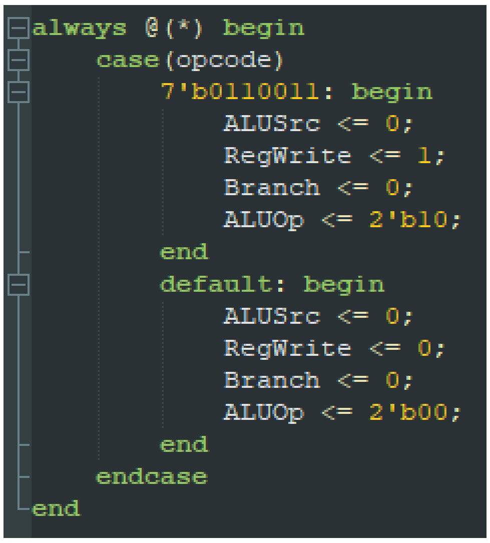
2. Implementation results



```
u@u:/media/sf_Downloads/LAB03$ chmod +x ./lab3TestScript.sh && ./lab3TestScript.sh
*************** CASE 1 ******************
Testcase 1 PASS
*************** CASE 2 ******************
Testcase 2 PASS
*************** CASE 3 ******************
Testcase 3 PASS
*************** CASE 4 ******************
Testcase 4 PASS
*************** CASE 5 ******************
Testcase 5 PASS
*************** CASE 6 ******************
Testcase 6 PASS
*************** CASE 7 ******************
Testcase 7 PASS
*************** CASE 8 ******************
Testcase 8 PASS
*************** CASE 9 ******************
Testcase 9 PASS
*************** CASE 10 ******************
Testcase 10 PASS
==========================================
Total Score:100
```

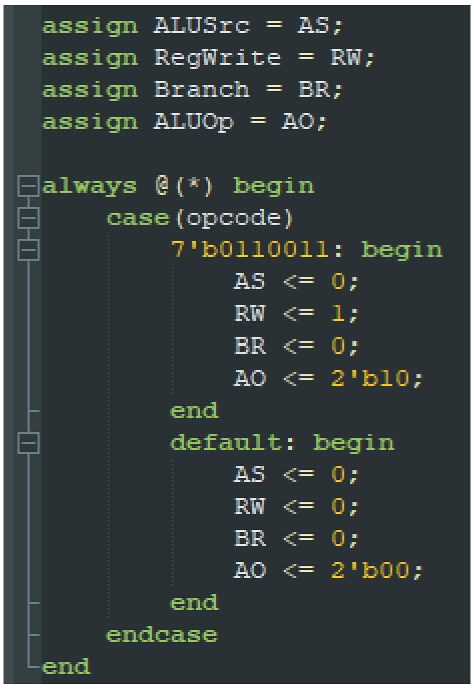
3. Problems encountered

At first, my code in decoder is like the code below and it produces a lot of error.



```
always @(*) begin
    case(opcode)
        7'b0110011: begin
            ALUSrc <= 0;
            RegWrite <= 1;
            Branch <= 0;
            ALUOp <= 2'b10;
        end
        default: begin
            ALUSrc <= 0;
            RegWrite <= 0;
            Branch <= 0;
            ALUOp <= 2'b00;
        end
    endcase
end
```

Solution: I added some intermediate registers to be assigned in the always block, and assign the registers to the output wire, then the problem is solved.



```verilog
assign ALUSrc = AS;
assign RegWrite = RW;
assign Branch = BR;
assign ALUOp = AO;

always @(*) begin
    case(opcode)
        7'b0110011: begin
            AS <= 0;
            RW <= 1;
            BR <= 0;
            AO <= 2'b10;
        end
        default: begin
            AS <= 0;
            RW <= 0;
            BR <= 0;
            AO <= 2'b00;
        end
    endcase
end
```