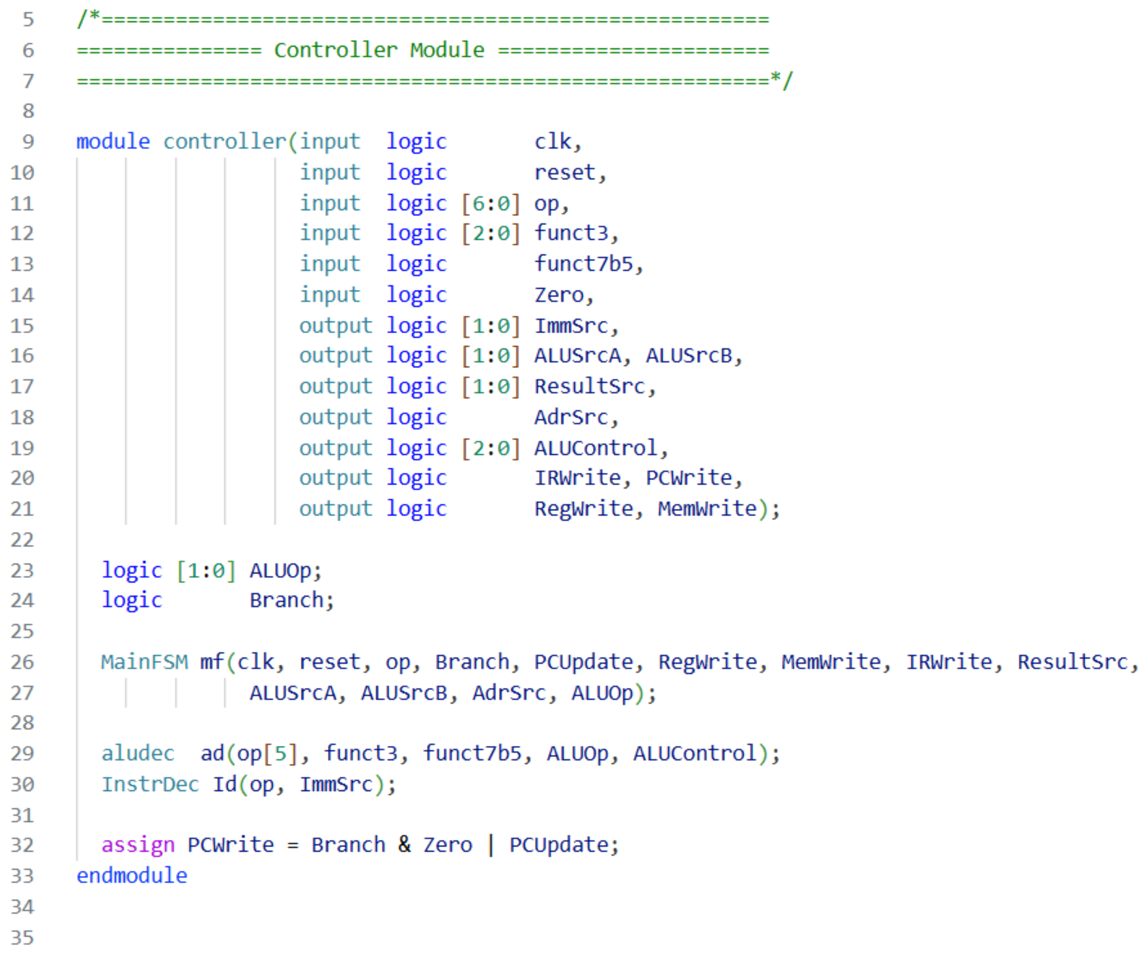
UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	**CPE300L - Digital System Architecture and Design Lab**	Semester:	**Fall 2025**
Points		Document author:	**Darryll Mckoy**
		Author's email:	**Mckoyd1@unlv.nevada.edu**
		Document topic:	**Postlab 8**
Instructor's comments:			

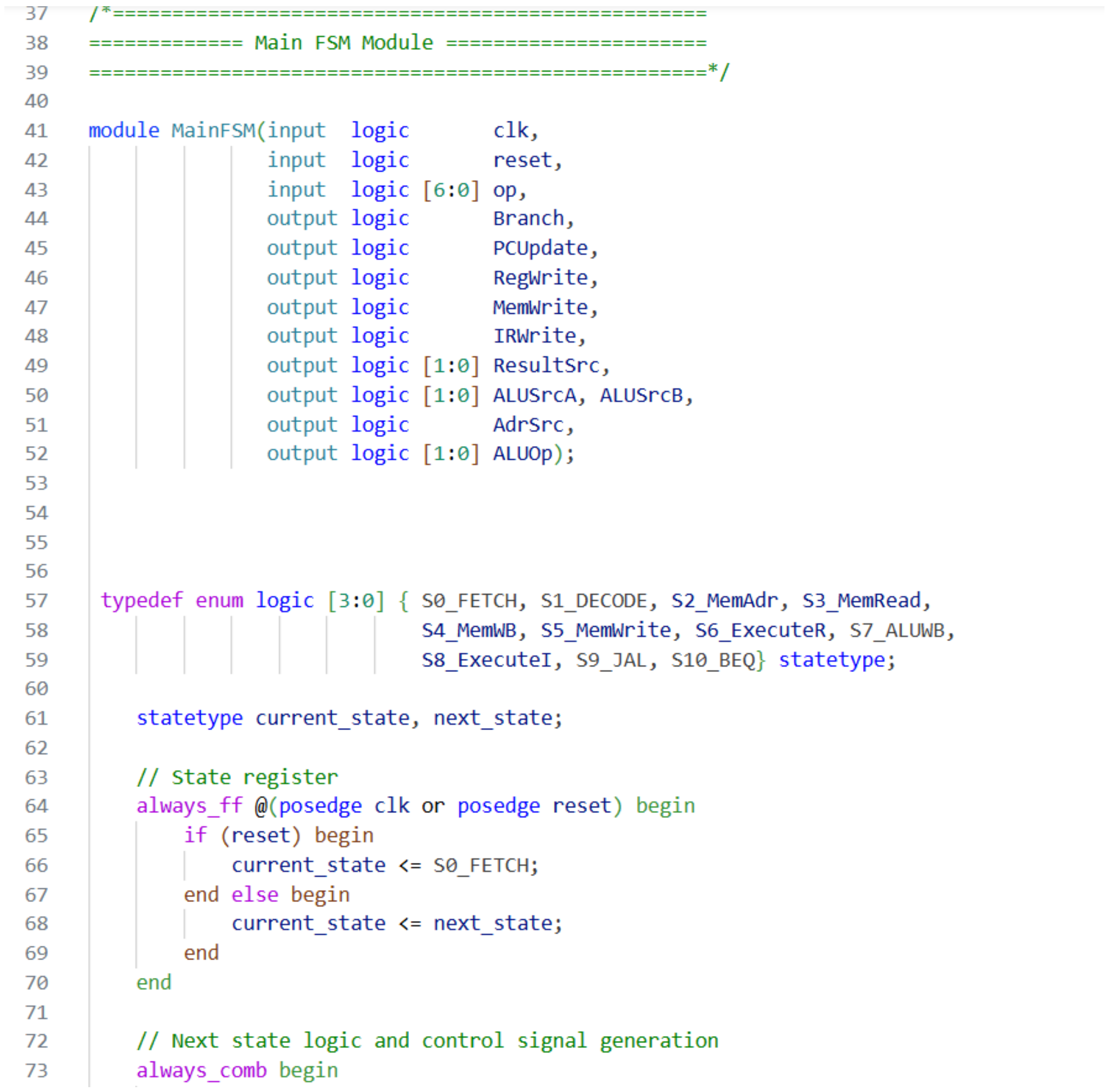
1. Hours spent on this lab

About 7 hours

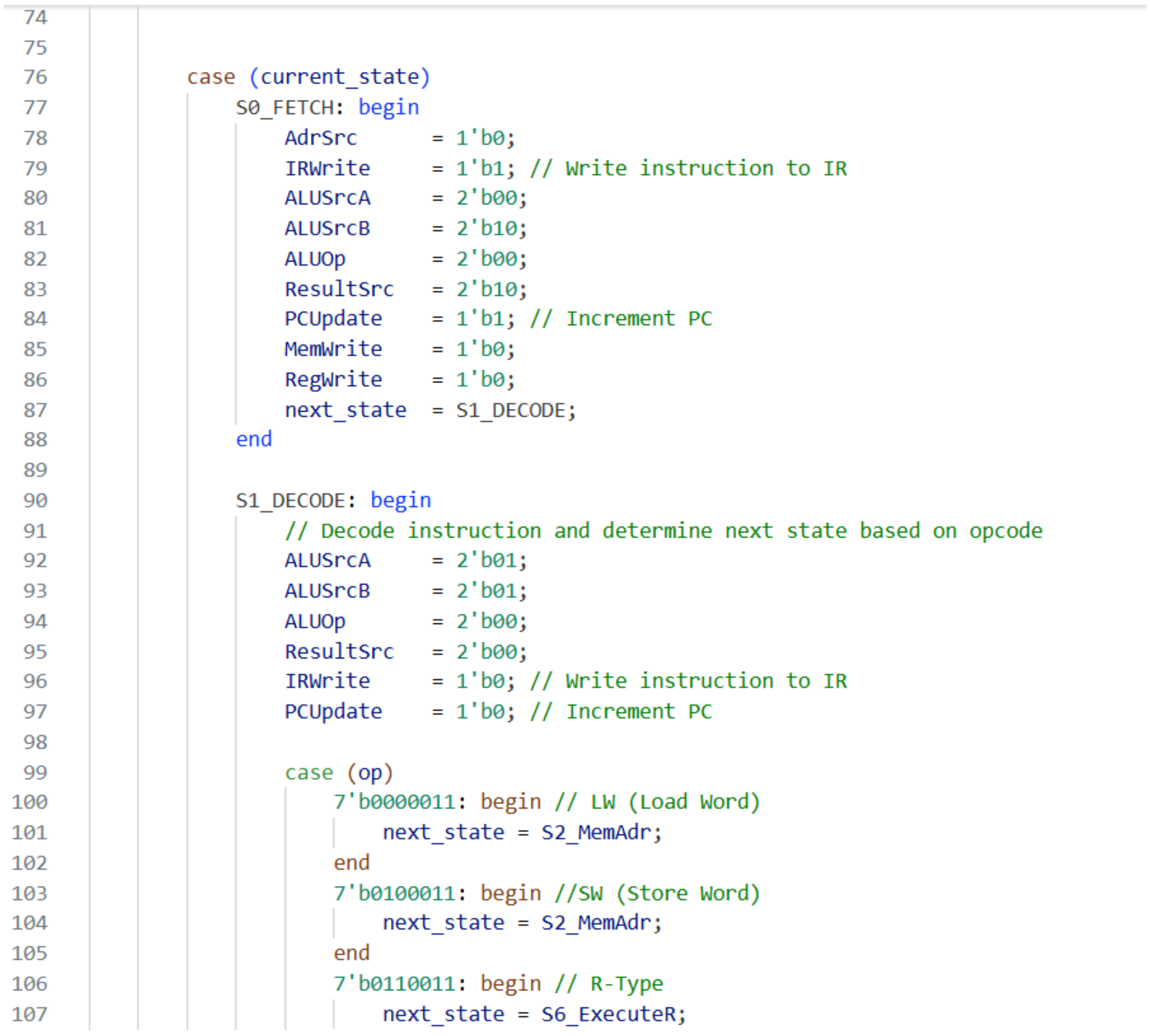
2. System Verilog code for controller module



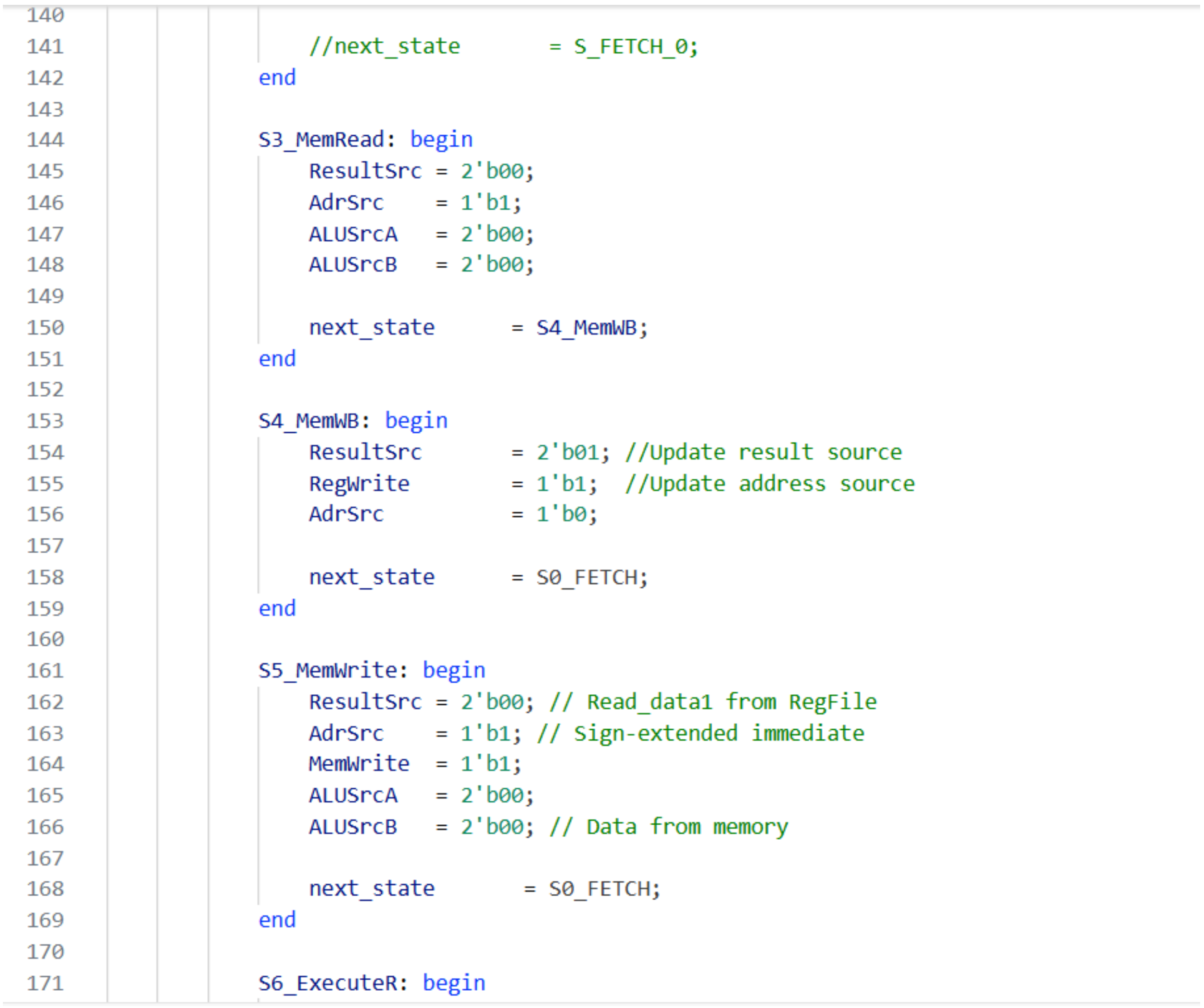
```
/*=======================================================
=============== Controller Module ======================
=======================================================*/

module controller(input  logic       clk,
                  input  logic       reset,
                  input  logic [6:0] op,
                  input  logic [2:0] funct3,
                  input  logic       funct7b5,
                  input  logic       Zero,
                  output logic [1:0] ImmSrc,
                  output logic [1:0] ALUSrcA, ALUSrcB,
                  output logic [1:0] ResultSrc,
                  output logic       AdrSrc,
                  output logic [2:0] ALUControl,
                  output logic       IRWrite, PCWrite,
                  output logic       RegWrite, MemWrite);

  logic [1:0] ALUOp;
  logic       Branch;

  MainFSM mf(clk, reset, op, Branch, PCUpdate, RegWrite, MemWrite, IRWrite, ResultSrc,
             ALUSrcA, ALUSrcB, AdrSrc, ALUOp);

  aludec  ad(op[5], funct3, funct7b5, ALUOp, ALUControl);
  InstrDec Id(op, ImmSrc);

  assign PCWrite = Branch & Zero | PCUpdate;
endmodule
```



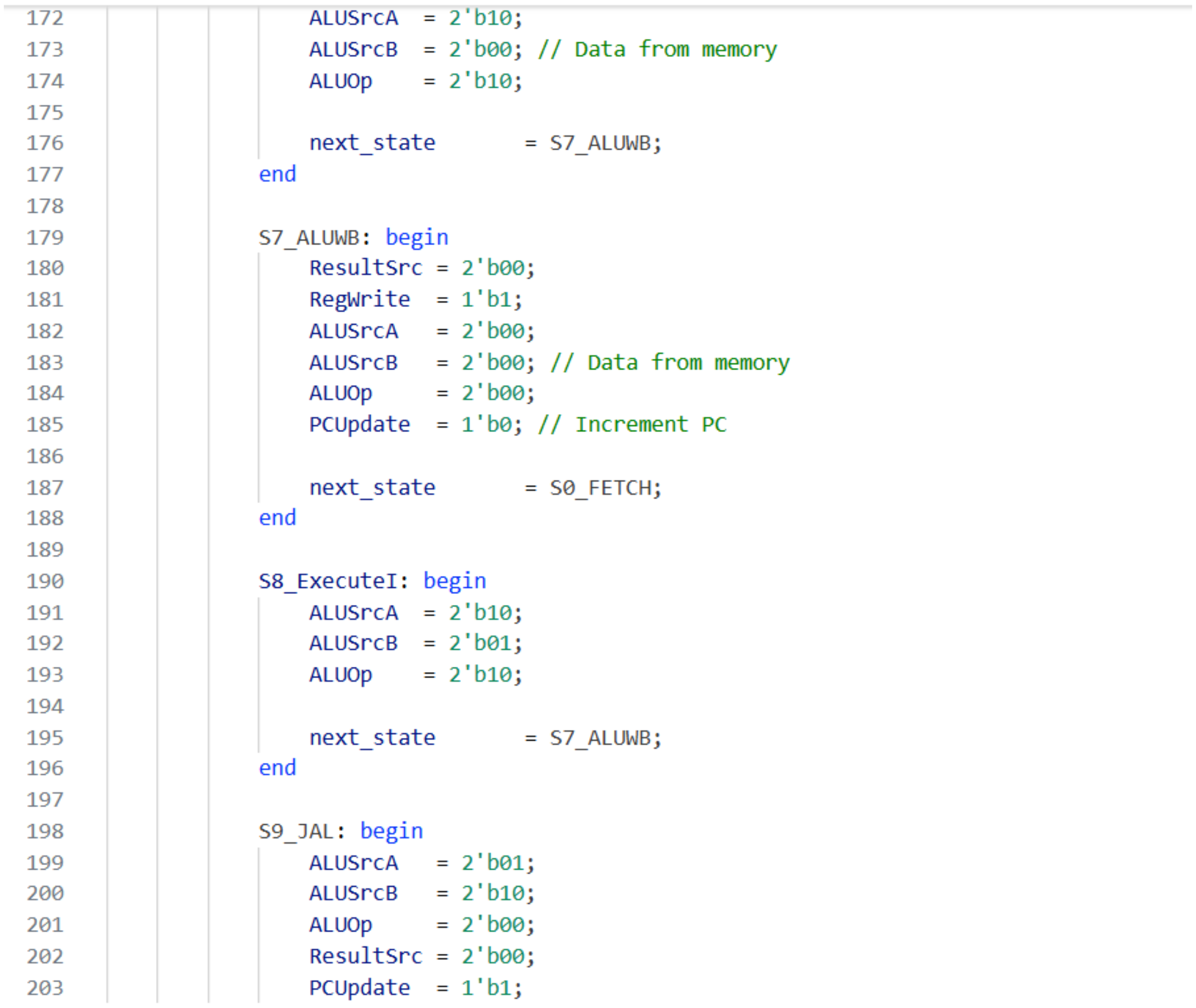
```
/*==================================================
============= Main FSM Module =====================
==================================================*/

module MainFSM(input  logic       clk,
               input  logic       reset,
               input  logic [6:0] op,
               output logic       Branch,
               output logic       PCUpdate,
               output logic       RegWrite,
               output logic       MemWrite,
               output logic       IRWrite,
               output logic [1:0] ResultSrc,
               output logic [1:0] ALUSrcA, ALUSrcB,
               output logic       AdrSrc,
               output logic [1:0] ALUOp);




 typedef enum logic [3:0] { S0_FETCH, S1_DECODE, S2_MemAdr, S3_MemRead,
                            S4_MemWB, S5_MemWrite, S6_ExecuteR, S7_ALUWB,
                            S8_ExecuteI, S9_JAL, S10_BEQ} statetype;

    statetype current_state, next_state;

    // State register
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            current_state <= S0_FETCH;
        end else begin
            current_state <= next_state;
        end
    end

    // Next state logic and control signal generation
    always_comb begin
```



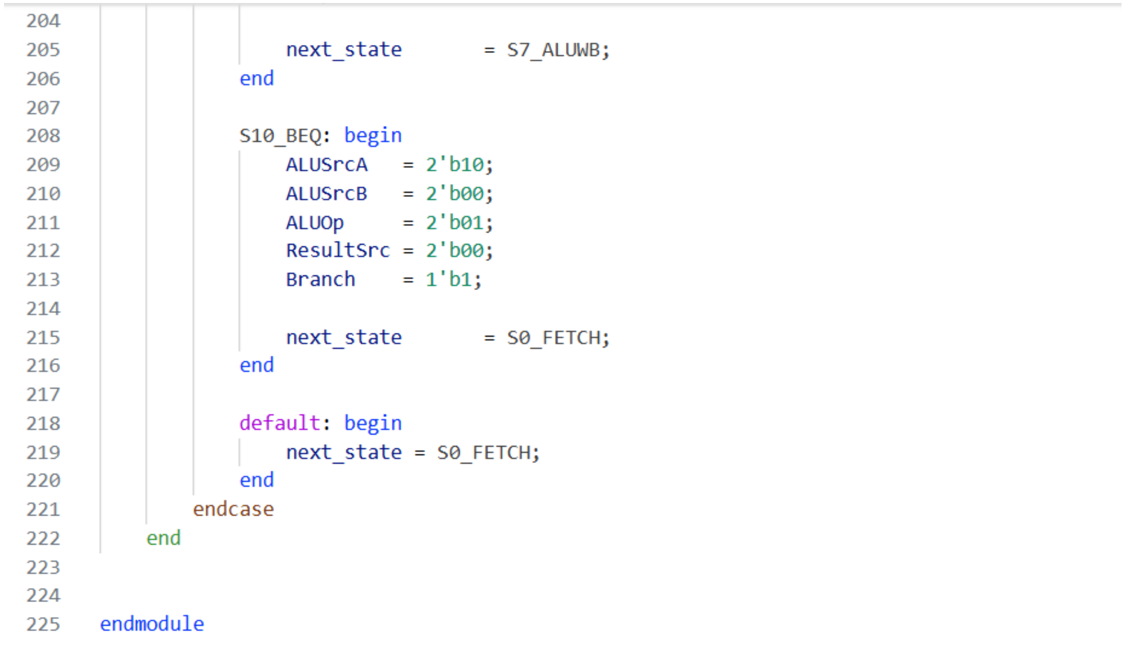
```verilog


        case (current_state)
            S0_FETCH: begin
                AdrSrc      = 1'b0;
                IRWrite     = 1'b1; // Write instruction to IR
                ALUSrcA     = 2'b00;
                ALUSrcB     = 2'b10;
                ALUOp       = 2'b00;
                ResultSrc   = 2'b10;
                PCUpdate    = 1'b1; // Increment PC
                MemWrite    = 1'b0;
                RegWrite    = 1'b0;
                next_state  = S1_DECODE;
            end

            S1_DECODE: begin
                // Decode instruction and determine next state based on opcode
                ALUSrcA     = 2'b01;
                ALUSrcB     = 2'b01;
                ALUOp       = 2'b00;
                ResultSrc   = 2'b00;
                IRWrite     = 1'b0; // Write instruction to IR
                PCUpdate    = 1'b0; // Increment PC

                case (op)
                    7'b0000011: begin // LW (Load Word)
                        next_state = S2_MemAdr;
                    end
                    7'b0100011: begin //SW (Store Word)
                        next_state = S2_MemAdr;
                    end
                    7'b0110011: begin // R-Type
                        next_state = S6_ExecuteR;
                    end
                    7'b0010011: begin // I-Type ALU
                        next_state = S8_ExecuteI;
                    end
                    7'b1101111: begin // JAL
                        next_state = S9_JAL;
                    end
                    7'b1100011: begin // beq
                        next_state = S10_BEQ;
                    end
                    default: begin // Handle unsupported opcodes
                        next_state = S0_FETCH; // Or an error state
                    end
                endcase
            end

            S2_MemAdr: begin
                ALUSrcA = 2'b10;
                ALUSrcB = 2'b01;
                ALUOp   = 2'b00;

                case (op)
                    7'b0000011: begin // LW (Load Word)
                        next_state = S3_MemRead;
                    end
                    7'b0100011: begin //SW (Store Word)
                        next_state = S5_MemWrite;
                    end
                    default: begin // Handle unsupported opcodes
                        next_state = S0_FETCH; // Or an error state
                    end
                endcase
```



```
                    //next_state       = S_FETCH_0;
                end

                S3_MemRead: begin
                    ResultSrc = 2'b00;
                    AdrSrc    = 1'b1;
                    ALUSrcA   = 2'b00;
                    ALUSrcB   = 2'b00;

                    next_state      = S4_MemWB;
                end

                S4_MemWB: begin
                    ResultSrc       = 2'b01; //Update result source
                    RegWrite        = 1'b1;  //Update address source
                    AdrSrc          = 1'b0;

                    next_state      = S0_FETCH;
                end

                S5_MemWrite: begin
                    ResultSrc = 2'b00; // Read_data1 from RegFile
                    AdrSrc    = 1'b1; // Sign-extended immediate
                    MemWrite  = 1'b1;
                    ALUSrcA   = 2'b00;
                    ALUSrcB   = 2'b00; // Data from memory

                    next_state       = S0_FETCH;
                end

                S6_ExecuteR: begin
```



```verilog
                ALUSrcA  = 2'b10;
                ALUSrcB  = 2'b00; // Data from memory
                ALUOp    = 2'b10;

                next_state       = S7_ALUWB;
            end

            S7_ALUWB: begin
                ResultSrc = 2'b00;
                RegWrite  = 1'b1;
                ALUSrcA   = 2'b00;
                ALUSrcB   = 2'b00; // Data from memory
                ALUOp     = 2'b00;
                PCUpdate  = 1'b0; // Increment PC

                next_state       = S0_FETCH;
            end

            S8_ExecuteI: begin
                ALUSrcA  = 2'b10;
                ALUSrcB  = 2'b01;
                ALUOp    = 2'b10;

                next_state       = S7_ALUWB;
            end

            S9_JAL: begin
                ALUSrcA   = 2'b01;
                ALUSrcB   = 2'b10;
                ALUOp     = 2'b00;
                ResultSrc = 2'b00;
                PCUpdate  = 1'b1;
```



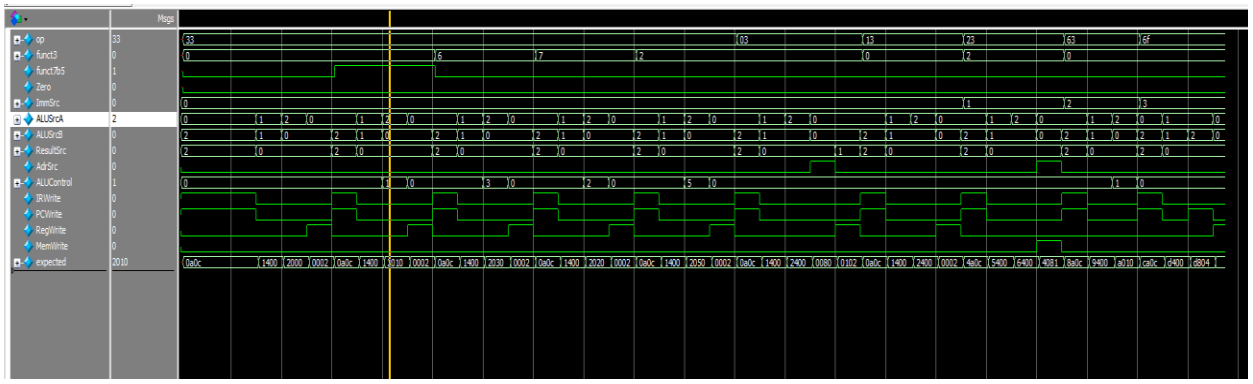
```verilog

                next_state       = S7_ALUWB;
            end

            S10_BEQ: begin
                ALUSrcA   = 2'b10;
                ALUSrcB   = 2'b00;
                ALUOp     = 2'b01;
                ResultSrc = 2'b00;
                Branch    = 1'b1;

                next_state       = S0_FETCH;
            end

            default: begin
                next_state = S0_FETCH;
            end
        endcase
    end


endmodule
```

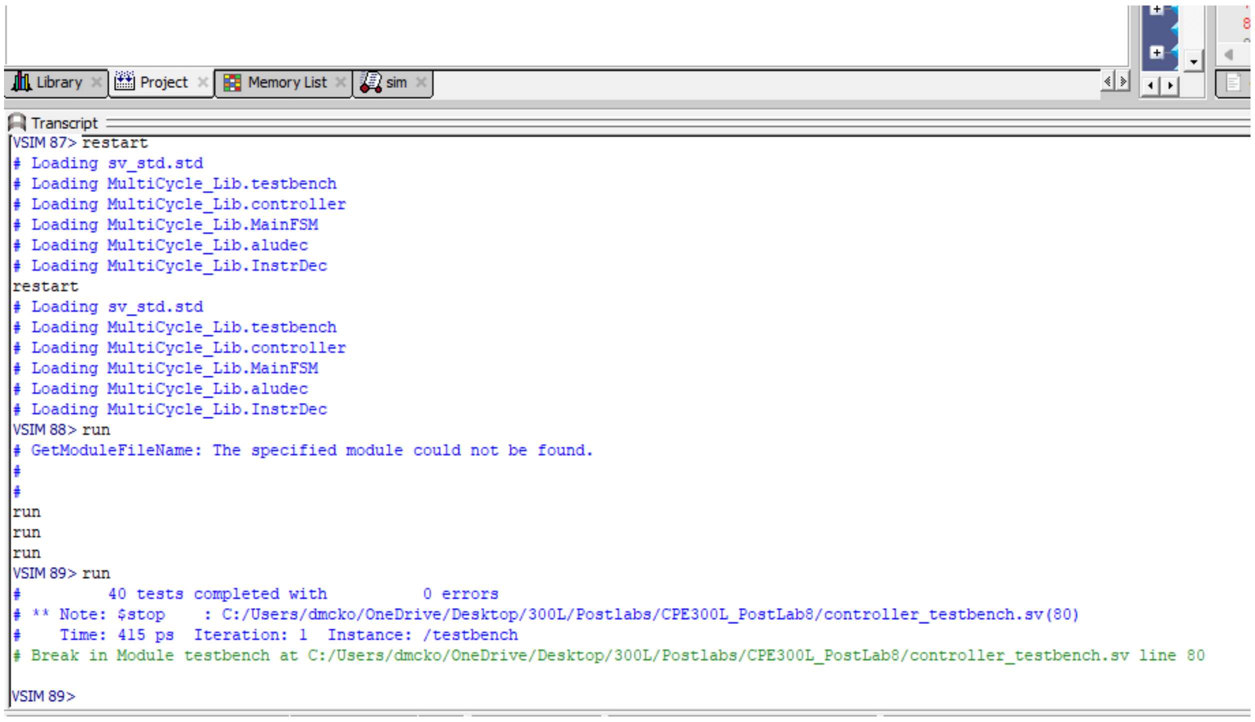
```
/*=======================================================
================ ALU Decoder Module ==================
======================================================*/

module aludec(input  logic       opb5,
              input  logic [2:0] funct3,
              input  logic       funct7b5,
              input  logic [1:0] ALUOp,
              output logic [2:0] ALUControl);

  logic  RtypeSub;
  assign RtypeSub = funct7b5 & opb5;  // TRUE for R-type subtract instruction

  always_comb
    case(ALUOp)
      2'b00:                ALUControl = 3'b000; // addition
      2'b01:                ALUControl = 3'b001; // subtraction
      default: case(funct3) // R-type or I-type ALU
                 3'b000:  if (RtypeSub)
                            ALUControl = 3'b001; // sub
                          else
                            ALUControl = 3'b000; // add, addi
                 3'b010:    ALUControl = 3'b101; // slt, slti
                 3'b110:    ALUControl = 3'b011; // or, ori
                 3'b111:    ALUControl = 3'b010; // and, andi
                 default:   ALUControl = 3'bxxx; // ???
               endcase
    endcase
endmodule
```

```
/*=====================================================
============== Instruction Decoder Module ============
=====================================================*/

module InstrDec(input  logic  [6:0] op,
                output logic  [1:0] ImmSrc);

logic [10:0] controls;

    assign ImmSrc = controls;

  always_comb
    case(op)
    // RegWrite_ImmSrc_ALUSrc_MemWrite_ResultSrc_Branch_ALUOp_Jump
      7'b0000011: controls = 11'b1_00_1_0_01_0_00_0; // lw
      7'b0100011: controls = 11'b0_01_1_1_00_0_00_1; // sw
      7'b0110011: controls = 11'b1_00_0_0_00_0_10_0; // R-type
      7'b1100011: controls = 11'b0_10_0_0_00_1_01_0; // beq
      7'b0010011: controls = 11'b1_00_1_0_00_0_10_0; // I-type ALU
      7'b1101111: controls = 11'b1_11_0_0_10_0_01_1; // jal
      default:    controls = 11'b0_00_0_0_00_0_00_0; // non-implemented instruction
    endcase


endmodule
```

3. Initially the controller module did not pass the test vectors. After simulating we found several errors that needed to be corrected one by one based on the error messages that printed to the terminal. Several corrections required within the FSM states, also made changes to the R-type instructions in the test vector text file (replaced all x's with 0's).

Screenshot of test vector wave file and successful compilation running 40 tests with zero errors.





```
VSIM 87> restart
# Loading sv_std.std
# Loading MultiCycle_Lib.testbench
# Loading MultiCycle_Lib.controller
# Loading MultiCycle_Lib.MainFSM
# Loading MultiCycle_Lib.aludec
# Loading MultiCycle_Lib.InstrDec
restart
# Loading sv_std.std
# Loading MultiCycle_Lib.testbench
# Loading MultiCycle_Lib.controller
# Loading MultiCycle_Lib.MainFSM
# Loading MultiCycle_Lib.aludec
# Loading MultiCycle_Lib.InstrDec
VSIM 88> run
# GetModuleFileName: The specified module could not be found.
#
#
run
run
run
VSIM 89> run
#          40 tests completed with           0 errors
# ** Note: $stop    : C:/Users/dmcko/OneDrive/Desktop/300L/Postlabs/CPE300L_PostLab8/controller_testbench.sv(80)
#    Time: 415 ps  Iteration: 1  Instance: /testbench
# Break in Module testbench at C:/Users/dmcko/OneDrive/Desktop/300L/Postlabs/CPE300L_PostLab8/controller_testbench.sv line 80

VSIM 89>
```