

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

```

5  /*=====
6  ===== Controller Module =====
7  =====*/
8
9  module controller(input  logic      clk,
10                  input  logic      reset,
11                  input  logic [6:0] op,
12                  input  logic [2:0] funct3,
13                  input  logic      funct7b5,
14                  input  logic      Zero,
15                  output logic [1:0] ImmSrc,
16                  output logic [1:0] ALUSrcA, ALUSrcB,
17                  output logic [1:0] ResultSrc,
18                  output logic      AdrSrc,
19                  output logic [2:0] ALUControl,
20                  output logic      IRWrite, PCWrite,
21                  output logic      RegWrite, MemWrite);
22
23      logic [1:0] ALUOp;
24      logic      Branch;
25
26      MainFSM mf(clk, reset, op, Branch, PCUpdate, RegWrite, MemWrite, IRWrite, ResultSrc,
27                ALUSrcA, ALUSrcB, AdrSrc, ALUOp);
28
29      aludec ad(op[5], funct3, funct7b5, ALUOp, ALUControl);
30      InstrDec Id(op, ImmSrc);
31
32      assign PCWrite = Branch & Zero | PCUpdate;
33  endmodule
34
35

```

```

37  /*=====
38  ===== Main FSM Module =====
39  =====*/
40
41  module MainFSM(input logic clk,
42                input logic reset,
43                input logic [6:0] op,
44                output logic Branch,
45                output logic PCUpdate,
46                output logic RegWrite,
47                output logic MemWrite,
48                output logic IRWrite,
49                output logic [1:0] ResultSrc,
50                output logic [1:0] ALUSrcA, ALUSrcB,
51                output logic AdrSrc,
52                output logic [1:0] ALUOp);
53
54
55
56
57  typedef enum logic [3:0] { S0_FETCH, S1_DECODE, S2_MemAdr, S3_MemRead,
58                            S4_MemWB, S5_MemWrite, S6_Executer, S7_ALUWB,
59                            S8_ExecuteI, S9_JAL, S10_BEQ} statetype;
60
61  statetype current_state, next_state;
62
63  // State register
64  always_ff @(posedge clk or posedge reset) begin
65      if (reset) begin
66          current_state <= S0_FETCH;
67      end else begin
68          current_state <= next_state;
69      end
70  end
71
72  // Next state logic and control signal generation
73  always_comb begin

```

```

74
75
76     case (current_state)
77     S0_FETCH: begin
78         AddrSrc      = 1'b0;
79         IRWrite       = 1'b1; // Write instruction to IR
80         ALUSrcA       = 2'b00;
81         ALUSrcB       = 2'b10;
82         ALUOp         = 2'b00;
83         ResultSrc     = 2'b10;
84         PCUpdate      = 1'b1; // Increment PC
85         MemWrite      = 1'b0;
86         RegWrite      = 1'b0;
87         next_state    = S1_DECODE;
88     end
89
90     S1_DECODE: begin
91         // Decode instruction and determine next state based on opcode
92         ALUSrcA       = 2'b01;
93         ALUSrcB       = 2'b01;
94         ALUOp         = 2'b00;
95         ResultSrc     = 2'b00;
96         IRWrite       = 1'b0; // Write instruction to IR
97         PCUpdate      = 1'b0; // Increment PC
98
99         case (op)
100         7'b0000011: begin // LW (Load Word)
101             |         next_state = S2_MemAdr;
102         end
103         7'b0100011: begin //SW (Store Word)
104             |         next_state = S2_MemAdr;
105         end
106         7'b0110011: begin // R-Type
107             |         next_state = S6_Executer;

```

```

108     end
109     7'b0010011: begin // I-Type ALU
110         |
111         next_state = S8_ExecuteI;
112     end
113     7'b1101111: begin // JAL
114         |
115         next_state = S9_JAL;
116     end
117     7'b1100011: begin // beq
118         |
119         next_state = S10_BEQ;
120     end
121     default: begin // Handle unsupported opcodes
122         |
123         next_state = S0_FETCH; // Or an error state
124     end
125 endcase
126 end
127
128 S2_MemAdr: begin
129     ALUSrcA = 2'b10;
130     ALUSrcB = 2'b01;
131     ALUOp   = 2'b00;
132
133     case (op)
134         7'b0000011: begin // LW (Load Word)
135             |
136             next_state = S3_MemRead;
137         end
138         7'b0100011: begin //SW (Store Word)
139             |
140             next_state = S5_MemWrite;
141         end
142         default: begin // Handle unsupported opcodes
143             |
144             next_state = S0_FETCH; // Or an error state
145         end
146     endcase
147 end

```

```

140
141     //next_state      = S_FETCH_0;
142 end
143
144 S3_MemRead: begin
145     ResultSrc = 2'b00;
146     AdrSrc    = 1'b1;
147     ALUSrcA   = 2'b00;
148     ALUSrcB   = 2'b00;
149
150     next_state = S4_MemWB;
151 end
152
153 S4_MemWB: begin
154     ResultSrc    = 2'b01; //Update result source
155     RegWrite     = 1'b1;  //Update address source
156     AdrSrc       = 1'b0;
157
158     next_state   = S0_FETCH;
159 end
160
161 S5_MemWrite: begin
162     ResultSrc = 2'b00; // Read_data1 from RegFile
163     AdrSrc    = 1'b1; // Sign-extended immediate
164     MemWrite  = 1'b1;
165     ALUSrcA   = 2'b00;
166     ALUSrcB   = 2'b00; // Data from memory
167
168     next_state = S0_FETCH;
169 end
170
171 S6_Executer: begin

```

```

172     ALUSrcA = 2'b10;
173     ALUSrcB = 2'b00; // Data from memory
174     ALUOp   = 2'b10;
175
176     next_state = S7_ALUWB;
177 end
178
179 S7_ALUWB: begin
180     ResultSrc = 2'b00;
181     RegWrite  = 1'b1;
182     ALUSrcA   = 2'b00;
183     ALUSrcB   = 2'b00; // Data from memory
184     ALUOp     = 2'b00;
185     PCUpdate  = 1'b0; // Increment PC
186
187     next_state = S0_FETCH;
188 end
189
190 S8_ExecuteI: begin
191     ALUSrcA = 2'b10;
192     ALUSrcB = 2'b01;
193     ALUOp   = 2'b10;
194
195     next_state = S7_ALUWB;
196 end
197
198 S9_JAL: begin
199     ALUSrcA = 2'b01;
200     ALUSrcB = 2'b10;
201     ALUOp   = 2'b00;
202     ResultSrc = 2'b00;
203     PCUpdate = 1'b1;

```

```

204
205     next_state = S7_ALUWB;
206 end
207
208 S10_BEQ: begin
209     ALUSrcA = 2'b10;
210     ALUSrcB = 2'b00;
211     ALUOp   = 2'b01;
212     ResultSrc = 2'b00;
213     Branch  = 1'b1;
214
215     next_state = S0_FETCH;
216 end
217
218 default: begin
219     next_state = S0_FETCH;
220 end
221 endcase
222 end
223
224
225 endmodule

```

```

229  /*=====
230  ===== ALU Decoder Module =====
231  =====*/
232
233  module aludec(input  logic      opb5,
234               input  logic [2:0] funct3,
235               input  logic      funct7b5,
236               input  logic [1:0] ALUOp,
237               output logic [2:0] ALUControl);
238
239  logic RtypeSub;
240  assign RtypeSub = funct7b5 & opb5; // TRUE for R-type subtract instruction
241
242  always_comb
243  case(ALUOp)
244      2'b00:          ALUControl = 3'b000; // addition
245      2'b01:          ALUControl = 3'b001; // subtraction
246      default: case(funct3) // R-type or I-type ALU
247          3'b000: if (RtypeSub)
248                  ALUControl = 3'b001; // sub
249                  else
250                  ALUControl = 3'b000; // add, addi
251          3'b010: ALUControl = 3'b101; // slt, slti
252          3'b110: ALUControl = 3'b011; // or, ori
253          3'b111: ALUControl = 3'b010; // and, andi
254          default: ALUControl = 3'bxxx; // ???
255      endcase
256  endcase
257 endmodule
258

```

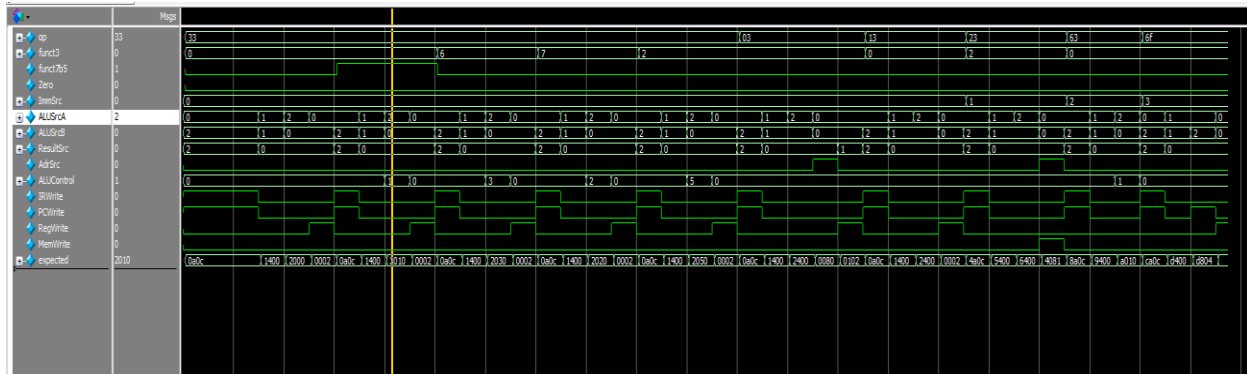
```

260  /*=====
261  ===== Instruction Decoder Module =====
262  =====*/
263
264  module InstrDec(input logic [6:0] op,
265                |   |   |   |   output logic [1:0] ImmSrc);
266
267  logic [10:0] controls;
268
269      assign ImmSrc = controls;
270
271  always_comb
272      case(op)
273  // RegWrite_ImmSrc_ALUSrc_MemWrite_ResultSrc_Branch_ALUOp_Jump
274      7'b0000011: controls = 11'b1_00_1_0_01_0_00_0; // lw
275      7'b0100011: controls = 11'b0_01_1_1_00_0_00_1; // sw
276      7'b0110011: controls = 11'b1_00_0_0_00_0_10_0; // R-type
277      7'b1100011: controls = 11'b0_10_0_0_00_1_01_0; // beq
278      7'b0010011: controls = 11'b1_00_1_0_00_0_10_0; // I-type ALU
279      7'b1101111: controls = 11'b1_11_0_0_10_0_01_1; // jal
280      default:    controls = 11'b0_00_0_0_00_0_00_0; // non-implemented instruction
281      endcase
282
283
284  endmodule

```


- 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.



```
Library | Project | Memory List | sim
Transcript
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>
```