```
1   // lab10_CJ
2   // risc-v multicycle controller
3   // Christian Johnson
4   // chrjohnson@hmc.edu
5   // 11/24/2024
6
7   // Structural Verilog Code for the ALU controller
8
9   module multicycle_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        // Defining internal logic for the multicycle controller
24        logic pcupdate, branch;
25        logic [1:0] aluop;
26
27        // Calling mainFSM module
28        mainFSM fsm(clk, reset, op, pcupdate, branch, regwrite, memwrite, irwrite, resultsrc,
    alusrca, alusrcb, adrsrc, aluop);
29
30
31        /*
32        Calling alu_decoder (from lab 2 implementation)
33        aluop[1]: a
34        aluop[0]: b
35
36        funct3[2]: c
37        funct3[1]: d
38        funct3[0]: e
39
40        op[5]:    f
41        funct7b5: g
42        */
43        alu_decoder aludec(aluop[1], aluop[0], funct3[2], funct3[1], funct3[0], op[5],
    funct7b5, alucontrol[2], alucontrol[1], alucontrol[0]);
44
45        // Calling instr_decoder module
46        instr_decoder id(op, immsrc);
47
48
49        // output logic
50        assign pcwrite = branch & zero | pcupdate;
51  endmodule
52
53
54
55  module mainFSM(input logic clk,
56                 input logic reset,
57                 input logic [6:0] op,
58                 output logic pcupdate,
59                 output logic branch,
60                 output logic regwrite, memwrite,
61                 output logic irwrite,
62                 output logic [1:0] resultsrc,
63                 output logic [1:0] alusrca, alusrcb,
64                 output logic adrsrc,
65                 output logic [1:0] aluop);
66
67
68        // Internal logic for states
```

```
69          typedef enum logic [3:0] {FETCH, DECODE, MEMADR, MEMREAD, MEMWB, MEMWRITE, EXECUTER,
    ALUWB, EXECUTEI, JAL, BEQ} statetype;
70          statetype state, nextstate;
71          logic [13:0] controls;
72
73          // state register
74          always_ff @(posedge clk, posedge reset)
75             if (reset) state <= FETCH;
76             else state <= nextstate;
77
78
79          // next state logic
80          always_comb
81             casez (state)
82                FETCH:      nextstate = DECODE;
83
84                DECODE:     casez(op)
85                               7'b0?00011:       nextstate = MEMADR;
86                               7'b0110011:       nextstate = EXECUTER;
87                               7'b0010011:       nextstate = EXECUTEI;
88                               7'b1101111:       nextstate = JAL;
89                               7'b1100011:       nextstate = BEQ;
90                               default:          nextstate = state;
91                            endcase
92
93                MEMADR:     casez(op)
94                               7'b0000011:       nextstate = MEMREAD;
95                               7'b0100011:       nextstate = MEMWRITE;
96                               default:          nextstate = state;
97                            endcase
98
99                MEMREAD:    nextstate = MEMWB;
100
101               MEMWB:      nextstate = FETCH;
102
103               MEMWRITE:   nextstate = FETCH;
104
105               EXECUTER:   nextstate = ALUWB;
106
107               ALUWB:      nextstate = FETCH;
108
109               EXECUTEI:   nextstate = ALUWB;
110
111               JAL:        nextstate = ALUWB;
112
113               BEQ:        nextstate = FETCH;
114
115               default:    nextstate = state;
116         endcase
117
118
119
120         // setting current state signals
121         always_comb
122            case (state)
123                     //
    pcupdate__branch__regwrite__memwrite__irwrite__resultsrc__alusrcb__alusrca__adrsrc__aluop
124               FETCH:      controls = 14'b1_0_0_0_1_10_10_00_0_00 ;
125
126               DECODE:     controls = 14'b0_0_0_0_0_00_01_01_0_00 ;
127
128               MEMADR:     controls = 14'b0_0_0_0_0_00_01_10_0_00 ;
129
130               MEMREAD:    controls = 14'b0_0_0_0_0_00_00_00_1_00 ;
131
132               MEMWB:      controls = 14'b0_0_1_0_0_01_00_00_0_00 ;
133
134               MEMWRITE:   controls = 14'b0_0_0_1_0_00_00_00_1_00 ;
135
136               EXECUTER:   controls = 14'b0_0_0_0_0_00_00_10_0_10 ;
```

```
137
138               ALUWB:      controls = 14'b0_0_1_0_0_00_00_00_0_00 ;
139
140               EXECUTEI:   controls = 14'b0_0_0_0_0_00_01_10_0_10 ;
141
142               JAL:        controls = 14'b1_0_0_0_0_00_10_01_0_00 ;
143
144               BEQ:        controls = 14'b0_1_0_0_0_00_00_10_0_01 ;
145
146               default:    controls = 14'b0_0_0_0_0_00_00_00_0_00 ;
147         endcase
148
149            assign {pcupdate, branch, regwrite, memwrite, irwrite, resultsrc, alusrcb,
    alusrca, adrsrc, aluop} = controls;
150   endmodule
151
152
153
154
155
156   // Structural Verilog Code for the ALU Decoder adapted from lab 2
157   module alu_decoder(input  logic a, b, c, d, e, f, g,
158                      output logic y2, y1, y0);
159         // Declaring the internal logic signals or local variables
160         // which can only be used inside of this module
161         logic n1, n2, n3, na, nb, nc, nd, ne;
162
163         // Getting negations of each input
164         not g1(na, a);
165         not g2(nb, b);
166         not g3(nc, c);
167         not g4(nd, d);
168         not g5(ne, e);
169
170         // y2 output logic
171         and a1(y2, a, nb, nc, d, ne);
172
173         // y1 output logic
174         and a2(y1, a, nb, c, d);
175
176         // y0 output logic
177         and a3(n1, na, b);
178         and a4(n2, a, nb, nc, nd, ne, f, g);
179         and a5(n3, a, nb, d, ne);
180         or o1(y0, n1, n2, n3);
181   endmodule
182
183
184
185
186   // Strucutural Verilog Code for the instr decoder
187
188   module instr_decoder(input logic [6:0] op,
189                        output logic [1:0] immsrc);
190
191         // Logic for choosing the immsrc
192         always_comb
193            casez (op)
194               7'b0110011: immsrc <= 2'b00; // R-type data processing (dont care but setting
    to 0's)
195               7'b0010011: immsrc <= 2'b00; // I-type data processing
196               7'b0000011: immsrc <= 2'b00; // LW
197               7'b0100011: immsrc <= 2'b01; // SW
198               7'b1100011: immsrc <= 2'b10; // BEQ
199               7'b1101111: immsrc <= 2'b11; // JAL
200             default: immsrc <= 2'b00;      // ???
201         endcase
202   endmodule
```