```
1   `timescale 1ns/1ps
2   `define mydelay 1
3
4   //------------------------------------------------------------------
5   // mips.v
6   // David_Harris@hmc.edu and Sarah_Harris@hmc.edu 23 October 2005
7   // Single-cycle MIPS processor
8   //------------------------------------------------------------------
9
10  // single-cycle MIPS processor
11  module mips(input         clk, reset,
12              output [31:0] pc,
13              input  [31:0] instr,
14              output        memwrite,
15              output [31:0] memaddr,
16              output [31:0] memwritedata,
17              input  [31:0] memreaddata);
18
19    wire        signext, shiftl16, memtoreg, branch;
20    wire        pcsrc, zero;
21    wire        alusrc, regdst, regwrite, jump;
22    wire [2:0]  alucontrol;
23
24    // Instantiate Controller
25    controller c(
26      .op         (instr[31:26]),
27        .funct      (instr[5:0]),
28        .zero       (zero),
29        .signext    (signext),
30        .shiftl16   (shiftl16),
31        .memtoreg   (memtoreg),
32        .memwrite   (memwrite),
33        .pcsrc      (pcsrc),
34        .alusrc     (alusrc),
35        .regdst     (regdst),
36        .regwrite   (regwrite),
37        .jump       (jump),
38        .alucontrol (alucontrol));
39
40    // Instantiate Datapath
41    datapath dp(
42      .clk        (clk),
43      .reset      (reset),
44      .signext    (signext),
45      .shiftl16   (shiftl16),
46      .memtoreg   (memtoreg),
47      .pcsrc      (pcsrc),
48      .alusrc     (alusrc),
49      .regdst     (regdst),
50      .regwrite   (regwrite),
51      .jump       (jump),
52      .alucontrol (alucontrol),
53      .zero       (zero),
54      .pc         (pc),
55      .instr      (instr),
56      .aluout     (memaddr),
57      .writedata  (memwritedata),
58      .readdata   (memreaddata));
59
60  endmodule
61
62  module controller(input  [5:0] op, funct,
63                    input        zero,
64                    output       signext,
65                    output       shiftl16,
66                    output       memtoreg, memwrite,
```

```
67                    output       pcsrc, alusrc,
68                    output       regdst, regwrite,
69                    output       jump,
70                    output [2:0] alucontrol);
71
72      wire [1:0] aluop;
73      wire       branch;
74
75      maindec md(
76        .op       (op),
77        .signext  (signext),
78        .shiftl16 (shiftl16),
79        .memtoreg (memtoreg),
80        .memwrite (memwrite),
81        .branch   (branch),
82        .alusrc   (alusrc),
83        .regdst   (regdst),
84        .regwrite (regwrite),
85        .jump     (jump),
86        .aluop    (aluop));
87
88      aludec ad(
89        .funct      (funct),
90        .aluop      (aluop),
91        .alucontrol (alucontrol));
92
93      assign pcsrc = branch & zero;
94
95    endmodule
96
97
98    module maindec(input  [5:0] op,
99                   output       signext,
100                  output       shiftl16,
101                  output       memtoreg, memwrite,
102                  output       branch, alusrc,
103                  output       regdst, regwrite,
104                  output       jump,
105                  output [1:0] aluop);
106
107     reg [10:0] controls;
108
109     assign {signext, shiftl16, regwrite, regdst, alusrc, branch, memwrite,
110             memtoreg, jump, aluop} = controls;
111
112     always @(*)
113       case(op)
114         6'b000000: controls <= #`mydelay 11'b00110000011; // Rtype
115         6'b100011: controls <= #`mydelay 11'b10101001000; // LW
116         6'b101011: controls <= #`mydelay 11'b10001010000; // SW
117         6'b000100: controls <= #`mydelay 11'b10000100001; // BEQ
118         6'b001000,
119         6'b001001: controls <= #`mydelay 11'b10101000000; // ADDI, ADDIU: only difference
    is exception
120         6'b001101: controls <= #`mydelay 11'b00101000010; // ORI
121         6'b001111: controls <= #`mydelay 11'b01101000000; // LUI
122         6'b000010: controls <= #`mydelay 11'b00000000100; // J
123         default:   controls <= #`mydelay 11'bxxxxxxxxxxx; // ???
124       endcase
125
126   endmodule
127
128   module aludec(input      [5:0] funct,
129                 input      [1:0] aluop,
130                 output reg [2:0] alucontrol);
131
```

```
132     always @(*)
133       case(aluop)
134         2'b00: alucontrol <= #`mydelay 3'b010;  // add
135         2'b01: alucontrol <= #`mydelay 3'b110;  // sub
136         2'b10: alucontrol <= #`mydelay 3'b001;  // or
137         default: case(funct)          // RTYPE
138             6'b100000,
139             6'b100001: alucontrol <= #`mydelay 3'b010; // ADD, ADDU: only difference is
    exception
140             6'b100010,
141             6'b100011: alucontrol <= #`mydelay 3'b110; // SUB, SUBU: only difference is
    exception
142             6'b100100: alucontrol <= #`mydelay 3'b000; // AND
143             6'b100101: alucontrol <= #`mydelay 3'b001; // OR
144             6'b101010: alucontrol <= #`mydelay 3'b111; // SLT
145             default:   alucontrol <= #`mydelay 3'bxxx; // ???
146           endcase
147       endcase
148
149   endmodule
150
151   module datapath(input         clk, reset,
152                   input         signext,
153                   input         shiftl16,
154                   input         memtoreg, pcsrc,
155                   input         alusrc, regdst,
156                   input         regwrite, jump,
157                   input  [2:0]  alucontrol,
158                   output        zero,
159                   output [31:0] pc,
160                   input  [31:0] instr,
161                   output [31:0] aluout, writedata,
162                   input  [31:0] readdata);
163
164     wire [4:0]  writereg;
165     wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
166     wire [31:0] signimm, signimmsh, shiftedimm;
167     wire [31:0] srca, srcb;
168     wire [31:0] result;
169     wire        shift;
170
171     // next PC logic
172     flopr #(32) pcreg(
173       .clk   (clk),
174       .reset (reset),
175       .d     (pcnext),
176       .q     (pc));
177
178     adder pcadd1(
179       .a (pc),
180       .b (32'b100),
181       .y (pcplus4));
182
183     sl2 immsh(
184       .a (signimm),
185       .y (signimmsh));
186
187     adder pcadd2(
188       .a (pcplus4),
189       .b (signimmsh),
190       .y (pcbranch));
191
192     mux2 #(32) pcbrmux(
193       .d0  (pcplus4),
194       .d1  (pcbranch),
195       .s   (pcsrc),
```

```
196       .y   (pcnextbr));
197
198     mux2 #(32) pcmux(
199       .d0   (pcnextbr),
200       .d1   ({pcplus4[31:28], instr[25:0], 2'b00}),
201       .s    (jump),
202       .y    (pcnext));
203
204     // register file logic
205     regfile rf(
206       .clk     (clk),
207       .we      (regwrite),
208       .ra1     (instr[25:21]),
209       .ra2     (instr[20:16]),
210       .wa      (writereg),
211       .wd      (result),
212       .rd1     (srca),
213       .rd2     (writedata));
214
215     mux2 #(5) wrmux(
216       .d0  (instr[20:16]),
217       .d1  (instr[15:11]),
218       .s   (regdst),
219       .y   (writereg));
220
221     mux2 #(32) resmux(
222       .d0 (aluout),
223       .d1 (readdata),
224       .s  (memtoreg),
225       .y  (result));
226
227     sign_zero_ext sze(
228       .a       (instr[15:0]),
229       .signext (signext),
230       .y       (signimm[31:0]));
231
232     shift_left_16 sl16(
233       .a         (signimm[31:0]),
234       .shiftl16  (shiftl16),
235       .y         (shiftedimm[31:0]));
236
237     // ALU logic
238     mux2 #(32) srcbmux(
239       .d0 (writedata),
240       .d1 (shiftedimm[31:0]),
241       .s  (alusrc),
242       .y  (srcb));
243
244     alu alu(
245       .a       (srca),
246       .b       (srcb),
247       .alucont (alucontrol),
248       .result  (aluout),
249       .zero    (zero));
250
251   endmodule
252
```