

```

1  // Eugene Ngo
2  // 4/20/2023
3  // CSE 469
4  // Lab 2, Task 1 and 2
5
6  /* arm is the spotlight of the show and contains the bulk of the datapath and control
   logic. This module is split into two parts, the datapath and control.
7  */
8
9  // clk - system clock
10 // rst - system reset
11 // Instr - incoming 32 bit instruction from imem, contains opcode, condition, addresses and
   or immediates
12 // ReadData - data read out of the dmem
13 // WriteData - data to be written to the dmem
14 // MemWrite - write enable to allowed WriteData to overwrite an existing dmem word
15 // PC - the current program count value, goes to imem to fetch instruciton
16 // ALUResult - result of the ALU operation, sent as address to the dmem
17
18 module arm (
19     input logic clk, rst,
20     input logic [31:0] Instr,
21     input logic [31:0] ReadData,
22     output logic [31:0] WriteData,
23     output logic [31:0] PC, ALUResult,
24     output logic MemWrite
25 );
26
27 // datapath buses and signals
28 logic [31:0] PCPrime, PCPlus4, PCPlus8; // pc signals
29 logic [3:0] RA1, RA2; // regfile input addresses
30 logic [31:0] RD1, RD2; // raw regfile outputs
31 logic [3:0] ALUFlags; // alu combinational flag outputs
32 logic [3:0] FlagsReg; // register for storing flag outputs
33 logic [31:0] ExtImm, SrcA, SrcB; // immediate and alu inputs
34 logic [31:0] Result; // computed or fetched value to be written into
   regfile or pc
35
36 // control signals
37 logic PCSrc, MemtoReg, ALUSrc, RegWrite, FlagWrite;
38 logic [1:0] RegSrc, ImmSrc, ALUControl;
39
40
41 /* The datapath consists of a PC as well as a series of muxes to make decisions about
   which data words to pass forward and operate on. It is
42 ** noticeably missing the register file and alu, which you will fill in using the
   modules made in lab 1. To correctly match up signals to the
43 ** ports of the register file and alu take some time to study and understand the logic
   and flow of the datapath.
44 */
45 //-----
46 //                                     DATAPATH
47 //-----
48
49     assign PCPrime = PCSrc ? Result : PCPlus4; // mux, use either default or newly
   computed value
50
51     assign PCPlus4 = PC + 'd4; // default value to access next instruction
52     assign PCPlus8 = PCPlus4 + 'd4; // value read when reading from reg[15]
53
54 // update the PC, at rst initialize to 0
55 always_ff @(posedge clk) begin
56     if (rst) PC <= '0;
57     else PC <= PCPrime;
58 end
59
60 // determine the register addresses based on control signals
61 // RegSrc[0] is set if doing a branch instruction
62 // RefSrc[1] is set when doing memory instructions
63 assign RA1 = RegSrc[0] ? 4'd15 : Instr[19:16];
64 assign RA2 = RegSrc[1] ? Instr[15:12] : Instr[3:0];
65
66 // Takes the control signals from the control module, and combines them with the

```

```

67 // different input signals RA1, RA2, Instr[15:12], Result. It outputs the read data
68 // signals: RD1, RD2 and writes in and saves data into its own memory if the
69 // wr_en signal is enabled.
70 reg_file u_reg_file (
71     .clk          (clk),
72     .wr_en        (RegWrite),
73     .write_data    (Result),
74     .write_addr    (Instr[15:12]),
75     .read_addr1    (RA1),
76     .read_addr2    (RA2),
77     .read_data1    (RD1),
78     .read_data2    (RD2)
79 );
80
81 // two muxes, put together into an always_comb for clarity
82 // determines which set of instruction bits are used for the immediate
83 always_comb begin
84     if (ImmSrc == 'b00) ExtImm = {{24{Instr[7]}}, Instr[7:0]}; // 8 bit
immediate - reg operations
85     else if (ImmSrc == 'b01) ExtImm = {20'b0, Instr[11:0]}; // 12 bit
immediate - mem operations
86     else ExtImm = {{6{Instr[23]}}, Instr[23:0], 2'b00}; // 24 bit
immediate - branch operation
87 end
88
89 // WriteData and SrcA are direct outputs of the register file, whereas SrcB is chosen
between reg file output and the immediate
90 assign WriteData = (RA2 == 'd15) ? PCPlus8 : RD2; // substitute the 15th
regfile register for PC
91 assign SrcA = (RA1 == 'd15) ? PCPlus8 : RD1; // substitute the 15th
regfile register for PC
92 assign SrcB = ALUSrc ? ExtImm : WriteData; // determine alu operand to
be either from reg file or from immediate
93
94 // Forwards the outputs from the register files and the selection logic above,
95 // to the ALU. The ALU computes a mathematical operation on the incoming values
96 // based on the inputs and outputs the result and the Flags that are triggered.
97 alu u_alu (
98     .a          (SrcA),
99     .b          (SrcB),
100     .ALUControl (ALUControl),
101     .Result      (ALUResult),
102     .ALUFlags    (ALUFlags)
103 );
104
105 // FlagsReg setting logic
106 // sets the flags if the Flagwrite control signal is enabled
107 always_ff @ (posedge clk) begin
108     if (FlagWrite) begin
109         FlagsReg <= ALUFlags;
110     end
111 end
112
113 // determine the result to run back to PC or the register file based on whether we used
a memory instruction
114 assign Result = MemtoReg ? ReadData : ALUResult; // determine whether final
writeback result is from dmemory or alu
115
116
117 /* The control consists of a large decoder, which evaluates the top bits of the
instruction and produces the control bits
** which become the select bits and write enables of the system. The write enables
(RegWrite, MemWrite and PCSrc) are
** especially important because they are representative of your processors current
state.
120 */
121 //-----
122 // CONTROL
123 //-----
124
125 always_comb begin
126     casez (Instr[31:20])
127

```

```

128 // ADD (Imm or Reg)
129 12'b111000?01000 : begin // note that we use wildcard "?" in bit 25. That bit
    decides whether we use immediate or reg, but regardless we add
130     PCSrc = 0;
131     MemtoReg = 0;
132     MemWrite = 0;
133     ALUSrc = Instr[25]; // may use immediate
134     RegWrite = 1;
135     RegSrc = 'b00;
136     ImmSrc = 'b00;
137     ALUControl = 'b00;
138     FlagWrite = 0;
139 end
140
141 // SUB (Imm or Reg)
142 12'b111000?00100 : begin // note that we use wildcard "?" in bit 25. That bit
    decides whether we use immediate or reg, but regardless we sub
143     PCSrc = 0;
144     MemtoReg = 0;
145     MemWrite = 0;
146     ALUSrc = Instr[25]; // may use immediate
147     RegWrite = 1;
148     RegSrc = 'b00;
149     ImmSrc = 'b00;
150     ALUControl = 'b01;
151     FlagWrite = 0;
152 end
153
154 // AND
155 12'b111000000000 : begin
156     PCSrc = 0;
157     MemtoReg = 0;
158     MemWrite = 0;
159     ALUSrc = 0;
160     RegWrite = 1;
161     RegSrc = 'b00;
162     ImmSrc = 'b00; // doesn't matter
163     ALUControl = 'b10;
164     FlagWrite = 0;
165 end
166
167 // ORR
168 12'b111000011000 : begin
169     PCSrc = 0;
170     MemtoReg = 0;
171     MemWrite = 0;
172     ALUSrc = 0;
173     RegWrite = 1;
174     RegSrc = 'b00;
175     ImmSrc = 'b00; // doesn't matter
176     ALUControl = 'b11;
177     FlagWrite = 0;
178 end
179
180 // LDR
181 12'b111001011001 : begin
182     PCSrc = 0;
183     MemtoReg = 1;
184     MemWrite = 0;
185     ALUSrc = 1;
186     RegWrite = 1;
187     RegSrc = 'b10; // msb doesn't matter
188     ImmSrc = 'b01;
189     ALUControl = 'b00; // do an add
190     FlagWrite = 0;
191 end
192
193 // STR
194 12'b111001011000 : begin
195     PCSrc = 0;
196     MemtoReg = 0; // doesn't matter
197     MemWrite = 1;
198     ALUSrc = 1;

```

```

199         RegWrite = 0;
200         RegSrc   = 'b10;    // msb doesn't matter
201         ImmSrc   = 'b01;
202         ALUControl = 'b00;  // do an add
203         FlagWrite = 0;
204     end
205
206     // B
207     12'b11101010???? : begin
208         PCSrc = 1;
209         MemtoReg = 0;
210         MemWrite = 0;
211         ALUSrc = 1;
212         RegWrite = 0;
213         RegSrc = 'b01;
214         ImmSrc = 'b10;
215         ALUControl = 'b00;  // do an add
216         FlagWrite = 0;
217     end
218
219     // CMP
220     12'b111000?00101 : begin
221         PCSrc = 0;
222         MemtoReg = 0;
223         MemWrite = 0;
224         ALUSrc = Instr[25]; // may use immediate
225         RegWrite = 1;
226         RegSrc = 'b00;
227         ImmSrc = 'b00;
228         ALUControl = 'b01;
229         FlagWrite = 1;
230     end
231
232     // B EQ
233     12'b00001010???? : begin
234         PCSrc = FlagsReg[2];
235         MemtoReg = 0;
236         MemWrite = 0;
237         ALUSrc = 1;
238         RegWrite = 0;
239         RegSrc = 'b01;
240         ImmSrc = 'b10;
241         ALUControl = 'b00;  // do an add
242         FlagWrite = 0;
243     end
244
245     // B NE
246     12'b00011010???? : begin
247         PCSrc = ~ FlagsReg[2];
248         MemtoReg = 0;
249         MemWrite = 0;
250         ALUSrc = 1;
251         RegWrite = 0;
252         RegSrc = 'b01;
253         ImmSrc = 'b10;
254         ALUControl = 'b00;  // do an add
255         FlagWrite = 0;
256     end
257
258     // B GE
259     12'b10101010???? : begin
260         PCSrc = ~ FlagsReg[3] | FlagsReg[2];
261         MemtoReg = 0;
262         MemWrite = 0;
263         ALUSrc = 1;
264         RegWrite = 0;
265         RegSrc = 'b01;
266         ImmSrc = 'b10;
267         ALUControl = 'b00;  // do an add
268         FlagWrite = 0;
269     end
270
271     // B GT

```

```

272 12'b11001010???? : begin
273     PCSrc      = ~ FlagsReg[3];
274     MemtoReg    = 0;
275     MemWrite    = 0;
276     ALUSrc      = 1;
277     RegWrite    = 0;
278     RegSrc      = 'b01;
279     ImmSrc      = 'b10;
280     ALUControl  = 'b00; // do an add
281     FlagWrite   = 0;
282 end
283
284 // B LE
285 12'b11011010???? : begin
286     PCSrc      = FlagsReg[3] | FlagsReg[2];
287     MemtoReg    = 0;
288     MemWrite    = 0;
289     ALUSrc      = 1;
290     RegWrite    = 0;
291     RegSrc      = 'b01;
292     ImmSrc      = 'b10;
293     ALUControl  = 'b00; // do an add
294     FlagWrite   = 0;
295 end
296
297 // B LT
298 12'b10111010???? : begin
299     PCSrc      = FlagsReg[3];
300     MemtoReg    = 0;
301     MemWrite    = 0;
302     ALUSrc      = 1;
303     RegWrite    = 0;
304     RegSrc      = 'b01;
305     ImmSrc      = 'b10;
306     ALUControl  = 'b00; // do an add
307     FlagWrite   = 0;
308 end
309
310 default: begin
311     PCSrc      = 0;
312     MemtoReg    = 0; // doesn't matter
313     MemWrite    = 0;
314     ALUSrc      = 0;
315     RegWrite    = 0;
316     RegSrc      = 'b00;
317     ImmSrc      = 'b00;
318     ALUControl  = 'b00; // do an add
319     FlagWrite   = 0;
320 end
321 endcase
322 end
323 endmodule

```