

```

1  /* arm is the spotlight of the show and contains the bulk of the datapath and control
2  */
3
4  // clk - system clock
5  // rst - system reset
6  // Instr - incoming 32 bit instruction from imem, contains opcode, condition, addresses and
   or immediates
7  // ReadData - data read out of the dmem
8  // WriteData - data to be written to the dmem
9  // MemWrite - write enable to allowed WriteData to overwrite an existing dmem word
10 // PC - the current program count value, goes to imem to fetch instrucion
11 // ALUResult - result of the ALU operation, sent as address to the dmem
12
13 module arm (
14     input logic      clk, rst,
15     input logic [31:0] Instr,
16     input logic [31:0] ReadData,
17     output logic [31:0] WriteData,
18     output logic [31:0] PC, ALUResult,
19     output logic      MemWrite
20 );
21
22     // datapath buses and signals
23     logic [31:0] PCPrime, PCPlus4, PCPlus8; // pc signals
24     logic [ 3:0] RA1, RA2; // regfile input addresses
25     logic [31:0] RD1, RD2; // raw regfile outputs
26     logic [ 3:0] ALUFlags; // alu combinational flag outputs
27     logic [31:0] ExtImm, SrcA, SrcB; // immediate and alu inputs
28     logic [31:0] Result; // computed or fetched value to be written into
   regfile or pc
29
30     // control signals
31     logic PCSrc, MemtoReg, ALUSrc, RegWrite, FlagWrite;
32     logic [1:0] RegSrc, ImmSrc, ALUControl;
33     logic [3:0] FlagsReg, cond;
34
35
36     /* The datapath consists of a PC as well as a series of muxes to make decisions about
37     which data words to pass forward and operate on. It is
38     ** noticeably missing the register file and alu, which you will fill in using the
39     modules made in lab 1. To correctly match up signals to the
40     ** ports of the register file and alu take some time to study and understand the logic
41     and flow of the datapath.
42     */
43     //-----
44     //                                     DATAPATH
45     //-----
46
47     assign PCPrime = PCSrc ? Result : PCPlus4; // mux, use either default or newly
   computed value
48     assign PCPlus4 = PC + 'd4; // default value to access next instruction
49     assign PCPlus8 = PCPlus4 + 'd4; // value read when reading from reg[15]
50
51     // update the PC, at rst initialize to 0
52     always_ff @(posedge clk) begin
53         if (rst) PC <= '0;
54         else PC <= PCPrime;
55     end
56
57     // determine the register addresses based on control signals
58     // RegSrc[0] is set if doing a branch instruction
59     // RefSrc[1] is set when doing memory instructions
60     assign RA1 = RegSrc[0] ? 4'd15 : Instr[19:16];
61     assign RA2 = RegSrc[1] ? Instr[15:12] : Instr[ 3: 0];
62
63     // Instantiates a register file to hold values.
64     reg_file u_reg_file (
65         .clk (clk),
66         .wr_en (RegWrite),
67         .write_data (Result),
68         .write_addr (Instr[15:12]),
69         .read_addr1 (RA1),
70         .read_addr2 (RA2),
71         .read_data1 (RD1),

```

```

70         .read_data2(RD2)
71     );
72
73     // two muxes, put together into an always_comb for clarity
74     // determines which set of instruction bits are used for the immediate
75     always_comb begin
76         if (ImmSrc == 'b00) ExtImm = {{24{Instr[7]}}, Instr[7:0]}; // 8 bit
77         immediate - reg operations
78         else if (ImmSrc == 'b01) ExtImm = {20'b0, Instr[11:0]}; // 12 bit
79         immediate - mem operations
80         else ExtImm = {{6{Instr[23]}}, Instr[23:0], 2'b00}; // 24 bit
81         immediate - branch operation
82     end
83
84     // WriteData and SrcA are direct outputs of the register file, whereas SrcB is chosen
85     // between reg file output and the immediate
86     assign WriteData = (RA2 == 'd15) ? PCPlus8 : RD2; // substitute the 15th
87     regfile register for PC
88     assign SrcA = (RA1 == 'd15) ? PCPlus8 : RD1; // substitute the 15th
89     regfile register for PC
90     assign SrcB = ALUSrc ? ExtImm : WriteData; // determine alu operand to
91     be either from reg file or from immediate
92
93     // Instantiates an alu module to do arithmetic operations.
94     alu u_alu (
95         .a (SrcA),
96         .b (SrcB),
97         .ALUControl (ALUControl),
98         .Result (ALUResult),
99         .ALUFlags (ALUFlags)
100     );
101
102     // determine the result to run back to PC or the register file based on whether we used
103     // a memory instruction
104     assign Result = MemtoReg ? ReadData : ALUResult; // determine whether final
105     writeback result is from dmemory or alu
106
107     always_ff @(posedge clk) begin
108         if (FlagWrite) FlagsReg = ALUFlags;
109     end
110
111     /* The control consists of a large decoder, which evaluates the top bits of the
112     instruction and produces the control bits
113     ** which become the select bits and write enables of the system. The write enables
114     (RegWrite, MemWrite and PCSrc) are
115     ** especially important because they are representative of your processors current
116     state.
117     */
118     //-----
119     //----- CONTROL -----
120     //-----
121     assign cond = Instr[31:28];
122
123     always_comb begin
124         casez (Instr[27:20])
125             // ADD (Imm or Reg)
126             8'b00?_0100_0 : begin // note that we use wildcard "?" in bit 25. That bit
127                 decides whether we use immediate or reg, but regardless we add
128                 PCSrc = 0;
129                 MemtoReg = 0;
130                 MemWrite = 0;
131                 ALUSrc = Instr[25]; // may use immediate
132                 RegWrite = 1;
133                 RegSrc = 'b00;
134                 ImmSrc = 'b00;
135                 ALUControl = 'b00;
136                 FlagWrite = 0;
137             end
138             // SUB (Imm or Reg)
139             8'b00?_0010_0 : begin // note that we use wildcard "?" in bit 25. That bit
140                 decides whether we use immediate or reg, but regardless we sub
141                 PCSrc = 0;
142                 MemtoReg = 0;

```

```

132         MemWrite = 0;
133         ALUSrc = Instr[25]; // may use immediate
134         RegWrite = 1;
135         RegSrc = 'b00;
136         ImmSrc = 'b00;
137         ALUControl = 'b01;
138         FlagWrite = 0;
139     end
140
141     // AND
142     8'b000_0000_0 : begin
143         PCSrc = 0;
144         MemtoReg = 0;
145         MemWrite = 0;
146         ALUSrc = 0;
147         RegWrite = 1;
148         RegSrc = 'b00;
149         ImmSrc = 'b00; // doesn't matter
150         ALUControl = 'b10;
151         FlagWrite = 0;
152     end
153
154     // ORR
155     8'b000_1100_0 : 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 = 'b11;
164         FlagWrite = 0;
165     end
166
167     // LDR
168     8'b010_1100_1 : begin
169         PCSrc = 0;
170         MemtoReg = 1;
171         MemWrite = 0;
172         ALUSrc = 1;
173         RegWrite = 1;
174         RegSrc = 'b10; // msb doesn't matter
175         ImmSrc = 'b01;
176         ALUControl = 'b00; // do an add
177         FlagWrite = 0;
178     end
179
180     // STR
181     8'b010_1100_0 : begin
182         PCSrc = 0;
183         MemtoReg = 0; // doesn't matter
184         MemWrite = 1;
185         ALUSrc = 1;
186         RegWrite = 0;
187         RegSrc = 'b10; // msb doesn't matter
188         ImmSrc = 'b01;
189         ALUControl = 'b00; // do an add
190         FlagWrite = 0;
191     end
192
193     // B
194     8'b1010_???? : begin
195         if((cond == 1110) ||
196            (cond == 0000 && FlagsReg[2]) ||
197            (cond == 0001 && !FlagsReg[2]) ||
198            (cond == 1010 && !FlagsReg[3]) ||
199            (cond == 1100 && !FlagsReg[3] && !FlagsReg[2]) ||
200            (cond == 1101 && (FlagsReg[3] || FlagsReg[2])) ||
201            (cond == 1011 && FlagsReg[3]))
202         ) begin
203             PCSrc = 1;
204             MemtoReg = 0;
205             MemWrite = 0;
206             ALUSrc = 1;
207             RegWrite = 0;

```

```

208         RegSrc    = 'b01;
209         ImmSrc    = 'b10;
210         ALUControl = 'b00; // do an add
211         FlagWrite = 0;
212     end else begin
213         PCSrc      = 0;
214         MemtoReg    = 0;
215         MemWrite    = 0;
216         ALUSrc      = 0;
217         RegWrite    = 0;
218         RegSrc      = 'b00;
219         ImmSrc      = 'b00; // doesn't matter
220         ALUControl  = 'b00;
221         FlagWrite   = 0;
222     end
223     // PCSrc      = 1;
224     // MemtoReg    = 0;
225     // MemWrite    = 0;
226     // ALUSrc      = 1;
227     // RegWrite    = 0;
228     // RegSrc      = 'b01;
229     // ImmSrc      = 'b10;
230     // ALUControl  = 'b00;
231     // FlagWrite   = 0;
232 end
233
234 // CMP
235 8'b00?00101 : begin
236     PCSrc      = 0;
237     MemtoReg    = 0;
238     MemWrite    = 0;
239     ALUSrc      = Instr[25];
240     RegWrite    = 1;
241     RegSrc      = 'b00;
242     ImmSrc      = 'b00;
243     ALUControl  = 'b01; // subtract
244     FlagWrite   = 1;
245 end
246
247 default: begin
248     PCSrc      = 0;
249     MemtoReg    = 0; // doesn't matter
250     MemWrite    = 0;
251     ALUSrc      = 0;
252     RegWrite    = 0;
253     RegSrc      = 'b00;
254     ImmSrc      = 'b00;
255     ALUControl  = 'b00; // do an add
256     FlagWrite   = 0;
257 end
258 endcase
259 end
260
261 endmodule

```