```
1   // Eugene Ngo
2   // 5/3/23
3   // EE 469
4   // Lab 3
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] InstrF,
21      input  logic [31:0] ReadData,
22      output logic [31:0] WriteDataM,
23      output logic [31:0] PC, ALUResultM,
24      output logic        MemWrite
25  );
26
27      // datapath buses and signals
28      logic [31:0] PCPrime, PCPlus4, PCPlus8, PCF; // pc signals
29      logic [ 3:0] RA1, RA2;                   // regfile input addresses
30      logic [31:0] RD1E, RD1D, RD2E, RD2D;                    // raw regfile outputs
31      logic [ 3:0] ALUFlags;                   // alu combinational flag outputs
32      logic [ 3:0] FlagsReg;                   // Flag output from recent CMP
33      logic [31:0] ExtImm, SrcA, SrcB, ExtImmE;         // immediate and alu inputs
34      logic [31:0] ResultW;                     // computed or fetched value to be written
    into regfile or pc
35      logic [31:0] RA1E, RA2E, SrcAE, SrcBE, InstrD, ALUResultE, ALUOutW, WriteDataE,
    ReadDataW;
36
37      // control signals
38      logic PCSrc, MemToReg, ALUSrc, RegWrite, FlagWrite;
39      logic [1:0] RegSrc, ImmSrc, ALUControl;
40      logic PCSrcD, PCSrcE, PCSrcM, PCSrcW;
41      logic RegWriteD, RegWriteE, RegWriteM, RegWriteW;
42      logic MemToRegD, MemToRegE, MemToRegM, MemToRegW;
43      logic MemWriteD, MemWriteE, MemWriteM;
44      logic CondExE;
45      logic [1:0] ALUControlD, ALUControlE;
46      logic BranchD, BranchE;
47      logic ALUSrcD, ALUSrcE;
48      logic StallF, StallD;
49      logic FlushD, FlushE;
50      logic ldrstall;
51      logic PCWrPendingF;
52      logic FlagWriteD, FlagWriteE;
53      logic BranchTakenE;
54      logic [3:0] FlagsE, Flags;
55      logic [3:0] CondE;
56      logic [1:0] ForwardAE, ForwardBE;
57      logic [3:0] WA3E, WA3M, WA3W;
58
59
60      /* 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
61      ** 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
62      ** ports of the register file and alu take some time to study and understand the logic
    and flow of the datapath.
63      */
64      //-------------------------------------------------------------------------------
65      //                                      DATAPATH
66      //-------------------------------------------------------------------------------
```

```
67
68      // Checks if we are branching and change PC accordingly
69       always_comb begin
70         if(BranchTakenE) begin
71            PCPrime = ALUResultE - 8;
72         end else if(PCSrcW) begin
73            PCPrime = ResultW;
74         end else begin
75            PCPrime = PCPlus4;
76         end
77       end
78
79       assign PCPlus4 = PCF + 'd4;                  // default value to access next instruction
80       assign PCPlus8 = PCPlus4 + 'd4;             // value read when reading from reg[15]
81
82       // update the PC, at rst initialize to 0
83       always_ff @(posedge clk) begin
84           if (rst) PC <= '0;
85           else     PC <= PCPrime;
86       end
87
88       // determine the register addresses based on control signals
89       // RegSrc[0] is set if doing a branch instruction
90       // RefSrc[1] is set when doing memory instructions
91       assign RA1 = RegSrc[0] ? 4'd15        : InstrD[19:16];
92       assign RA2 = RegSrc[1] ? InstrD[15:12] : InstrD[ 3: 0];
93
94       // register memory
95       // when instructed, we write into the registers the value we want
96       // also read out any data that we request via our RA1 and RA2
97       reg_file u_reg_file (
98           .clk       (!clk),
99           .wr_en     (RegWriteW),
100          .write_data(ResultW),
101          .write_addr(WA3W),
102          .read_addr1(RA1),
103          .read_addr2(RA2),
104          .read_data1(RD1D),
105          .read_data2(RD2D)
106      );
107
108      // two muxes, put together into an always_comb for clarity
109      // determines which set of instruction bits are used for the immediate
110      always_comb begin
111          if      (ImmSrc == 'b00) ExtImm = {{24{InstrD[7]}},InstrD[7:0]};          // 8 bit
     immediate - reg operations
112          else if (ImmSrc == 'b01) ExtImm = {20'b0, InstrD[11:0]};                 // 12 bit
     immediate - mem operations
113          else                     ExtImm = {{6{InstrD[23]}}, InstrD[23:0], 2'b00}; // 24 bit
     immediate - branch operation
114      end
115
116      assign SrcBE = (ALUSrcE) ? ExtImmE : WriteDataE;
117
118
119
120      //data forwarding
121      // Changes ForwardAE output depending on whether or not
122      // execute stage register matches memory or writeback registers
123      logic MATCH_1E_M, MATCH_2E_M, MATCH_1E_W, MATCH_2E_W;
124      always_comb begin
125        MATCH_1E_M = RA1E == WA3M;
126        MATCH_2E_M = RA2E == WA3M;
127        MATCH_1E_W = RA1E == WA3W;
128        MATCH_2E_W = RA2E == WA3W;
129        if(MATCH_1E_M & RegWriteM) begin
130           ForwardAE = 2'b10;
131        end else if (MATCH_1E_W & RegWriteW) begin
132           ForwardAE = 2'b01;
133        end else begin
134           ForwardAE = 2'b00;
135        end
136        if(MATCH_2E_M & RegWriteM) begin
```

```
137             ForwardBE = 2'b10;
138           end else if (MATCH_2E_W & RegWriteW) begin
139             ForwardBE = 2'b01;
140           end else begin
141             ForwardBE = 2'b00;
142           end
143         end
144
145       //stalling and flushing
146
147         assign PCSrcD = (RegWriteD & (InstrD[15:12] == 4'b1111));
148         assign ldrstall = (MemToRegE) & ((RA1 == WA3E) | (RA2 == WA3E));
149         assign PCWrPendingF = PCSrcD | PCSrcE | PCSrcM;
150         assign StallF = ldrstall | PCWrPendingF;
151         assign FlushD = PCWrPendingF | PCSrcW | BranchTakenE;
152         assign FlushE = ldrstall | BranchTakenE;
153         assign StallD = ldrstall;
154
155         // Forward multiplexer
156         always_comb begin
157           case(ForwardAE)
158           2'b00 : begin
159             SrcAE = (RA1E == 'd15) ? PCPlus8 : RD1E;
160           end
161           2'b01 : begin
162             SrcAE = ResultW; //data forward to end
163           end
164           2'b10 : begin
165             SrcAE = ALUResultM; //data forward to previous alu instruction
166           end
167           default : begin
168             SrcAE = 0;
169           end
170           endcase
171           case(ForwardBE)
172           2'b00 : begin
173             WriteDataE = (RA2E == 'd15) ? PCPlus8 : RD2E;
174           end
175           2'b01 : begin
176             WriteDataE = ResultW; //data forward to end
177           end
178           2'b10 : begin
179             WriteDataE = ALUResultM; //data forward to previous alu instruction
180           end
181           default : begin
182             WriteDataE = 0;
183           end
184           endcase
185         end
186
187         // instruction memory
188         // contained machine code instructions which instruct processor on which operations to
    make
189         // effectively a rom because our processor cannot write to it
190         alu u_alu (
191             .a          (SrcAE),
192             .b          (SrcBE),
193             .ALUControl (ALUControlE),
194             .Result     (ALUResultE),
195             .ALUFlags   (ALUFlags)
196         );
197
198         assign MemWrite = MemWriteM;
199
200         // register 1
201         always_ff@(posedge clk) begin
202             if(rst | FlushD) begin
203                 InstrD <= 0;
204             end else if (StallD) begin
205                 InstrD <= InstrD;
206             end else begin
207                 InstrD <= InstrF;
208             end
```

```
209
210             if(rst) begin
211                PCF <= 0;
212             end else if (StallF) begin
213                PCF <= PCF;
214             end else begin
215                PCF <= PCPrime;
216             end
217         end
218
219         // register 2
220         always_ff@(posedge clk) begin
221             if(rst | FlushE) begin
222                PCSrcE <= 0;
223                RegWriteE <= 0;
224                ALUControlE <= 0;
225                MemToRegE <= 0;
226                MemWriteE <= 0;
227                RD1E <= 0;
228                RD2E <= 0;
229                RA1E <= 0;
230                RA2E <= 0;
231                FlagWriteE <= 0;
232                CondE <= 0;
233                BranchE <= 0;
234                ALUSrcE <= 0;
235                FlagsE <= 0;
236                WA3E <= 0;
237                ExtImmE <= 0;
238             end else begin
239                PCSrcE <= PCSrcD;
240                RegWriteE <= RegWriteD;
241                ALUControlE <= ALUControlD;
242                MemToRegE <= MemToRegD;
243                MemWriteE <= MemWriteD;
244                RD1E <= RD1D;
245                RD2E <= RD2D;
246                RA1E <= RA1;
247                RA2E <= RA2;
248                FlagWriteE <= FlagWriteD;
249                ExtImmE <= ExtImm;
250                BranchE <= BranchD;
251                ALUSrcE <= ALUSrcD;
252                WA3E <=InstrD[15:12];
253                CondE <= InstrD[31:28];
254                if(FlagWriteE) begin
255                   FlagsE <= ALUFlags;
256                end
257             end
258         end
259
260         // register 3
261         always_ff@(posedge clk) begin
262             if(rst) begin
263                PCSrcM <= 0;
264                WA3M <= 0;
265                ALUResultM <= 0;
266                WriteDataM <= 0;
267                RegWriteM <= 0;
268                MemToRegM <= 0;
269                MemWriteM <= 0;
270             end else begin
271                PCSrcM <= PCSrcE & CondExE;
272                WA3M <= WA3E;
273                ALUResultM <= ALUResultE;
274                WriteDataM <= WriteDataE;
275                RegWriteM <= RegWriteE & CondExE;
276                MemToRegM <= MemToRegE;
277                MemWriteM <= MemWriteE & CondExE;
278             end
279         end
280
281         assign BranchTakenE = (BranchE & CondExE);
```

```
282
283         // register 4
284         assign ResultW = (MemToRegW) ? ReadDataW : ALUOutW;
285         always_ff@(posedge clk) begin
286           if(rst) begin
287              PCSrcW <= 0;
288              RegWriteW <= 0;
289              MemToRegW <= 0;
290              ALUOutW <= 0;
291              ReadDataW <= 0;
292              WA3W <= 0;
293           end else begin
294              PCSrcW <= PCSrcM;
295              RegWriteW <= RegWriteM;
296              MemToRegW <= MemToRegM;
297              ALUOutW <= ALUResultM;
298              ReadDataW <= ReadData;
299              WA3W <= WA3M;
300           end
301         end
302
303         logic V = FlagsE[0];
304         logic C = FlagsE[1];
305         logic Z = FlagsE[2];
306         logic N = FlagsE[3];
307
308         always_comb begin
309         case(CondE)
310            4'b0000: begin
311              CondExE = Z;
312            end
313            4'b0001: begin
314              CondExE = !Z;
315            end
316            4'b0010: begin
317              CondExE = C;
318            end
319            4'b0011: begin
320              CondExE = !C;
321            end
322            4'b0100: begin
323              CondExE = N;
324            end
325            4'b0101: begin
326              CondExE = !N;
327            end
328            4'b0110: begin
329              CondExE = V;
330            end
331            4'b0111: begin
332              CondExE = !V;
333            end
334            4'b1000: begin
335              CondExE = !Z&C;
336            end
337            4'b1001: begin
338              CondExE = Z | !C;
339            end
340            4'b1010: begin
341              CondExE = !(N ^ V);
342            end
343            4'b1011: begin
344              CondExE = N ^ V;
345            end
346            4'b1100: begin
347              CondExE = !Z & !(N ^ V);
348            end
349            4'b1101: begin
350              CondExE = Z | (N ^ V);
351            end
352            4'b1110: begin
353              CondExE = 1;
354            end
```

```
355         4'b1111: begin
356           CondExE = 1;
357         end
358         default : begin
359           CondExE = 1;
360         end
361         endcase
362       end
363
364       /* The control conists of a large decoder, which evaluates the top bits of the instruction and produces the control bits
365       ** which become the select bits and write enables of the system. The write enables (RegWrite, MemWrite and PCSrc) are
366       ** especially important because they are representative of your processors current state.
367       */
368       //------------------------------------------------------------------------------
369       //                                      CONTROL
370       //------------------------------------------------------------------------------
371
372
373       always_comb begin
374           casez (InstrD[31:20])
375
376               // ADD (Imm or Reg)
377               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
378                   //PCSrcD    = 0;
379                   MemToRegD = 0;
380                   MemWriteD = 0;
381                   ALUSrcD   = InstrD[25]; // may use immediate
382                   RegWriteD = 1;
383                   RegSrc    = 'b00;
384                   ImmSrc    = 'b00;
385                   ALUControlD = 'b00;
386                   FlagWriteD = 0;
387                   BranchD = 0;
388               end
389
390               // SUB (Imm or Reg)
391               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
392                   //PCSrcD    = 0;
393                   MemToRegD = 0;
394                   MemWriteD = 0;
395                   ALUSrcD   = InstrD[25]; // may use immediate
396                   RegWriteD = 1;
397                   RegSrc    = 'b00;
398                   ImmSrc    = 'b00;
399                   ALUControlD = 'b01;
400                   FlagWriteD = 0;
401                   BranchD = 0;
402               end
403
404               // CMP (Imm or Reg)
405               12'B111000?00101 : begin   // note that we use wildcard "?" in bit 25. That bit decides whether we use immediate or reg, but regardless we sub
406                   //PCSrcD    = 0;
407                   MemToRegD = 0;
408                   MemWriteD = 0;
409                   ALUSrcD   = InstrD[25]; // may use immediate
410                   RegWriteD = 1;
411                   RegSrc    = 'b00;
412                   ImmSrc    = 'b00;
413                   ALUControlD = 'b01;
414                   FlagWriteD = 1;
415                   BranchD = 0;
416               end
417
418               // AND
419               12'b111000000000 : begin
420                   //PCSrcD    = 0;
421                   MemToRegD = 0;
```

```
422                     MemWriteD = 0;
423                     ALUSrcD   = 0;
424                     RegWriteD = 1;
425                     RegSrc    = 'b00;
426                     ImmSrc    = 'b00;    // doesn't matter
427                     ALUControlD = 'b10;
428                     FlagWriteD = 0;
429                     BranchD = 0;
430                 end
431
432                 // ORR
433                 12'b111000011000 : begin
434                     //PCSrcD    = 0;
435                     MemToRegD = 0;
436                     MemWriteD = 0;
437                     ALUSrcD   = 0;
438                     RegWriteD = 1;
439                     RegSrc    = 'b00;
440                     ImmSrc    = 'b00;    // doesn't matter
441                     ALUControlD = 'b11;
442                     FlagWriteD = 0;
443                     BranchD = 0;
444                 end
445
446                 // LDR
447                 12'b111001011001 : begin
448                     //PCSrcD    = 0;
449                     MemToRegD = 1;
450                     MemWriteD = 0;
451                     ALUSrcD   = 1;
452                     RegWriteD = 1;
453                     RegSrc    = 'b10;    // msb doesn't matter
454                     ImmSrc    = 'b01;
455                     ALUControlD = 'b00; // do an add
456                     FlagWriteD = 0;
457                     BranchD = 0;
458                 end
459
460                 // STR
461                 12'b111001011000 : begin
462                     //PCSrcD    = 0;
463                     MemToRegD = 0; // doesn't matter
464                     MemWriteD = 1;
465                     ALUSrcD   = 1;
466                     RegWriteD = 0;
467                     RegSrc    = 'b10;    // msb doesn't matter
468                     ImmSrc    = 'b01;
469                     ALUControlD = 'b00;  // do an add
470                     FlagWriteD = 0;
471                     BranchD = 0;
472                 end
473
474                 // B
475                 12'b????1010???? : begin
476                         case (InstrD[31:28])
477                             4'b1110 : begin
478                                 //PCSrcD    = 1;
479                                 MemToRegD = 0;
480                                 MemWriteD = 0;
481                                 ALUSrcD   = 1;
482                                 RegWriteD = 0;
483                                 RegSrc    = 'b01;
484                                 ImmSrc    = 'b10;
485                                 ALUControlD = 'b00;  // do an add
486                                 FlagWriteD = 0;
487                                 BranchD = 1;
488                             end
489
490                             // equal
491                             4'b0000 : begin
492                                //PCSrcD    = 1;
493                                MemToRegD = 0;
494                                MemWriteD = 0;
```

```
495                             ALUSrcD   = 1;
496                             RegWriteD = 0;
497                             RegSrc    = 'b01;
498                             ImmSrc    = 'b10;
499                             ALUControlD = 'b00;
500                             FlagWriteD = 0;
501                             BranchD = 1;
502                         end
503
504                         // not equal
505                         4'b0001 : begin
506                             //PCSrcD    = 1;
507                             MemToRegD = 0;
508                             MemWriteD = 0;
509                             ALUSrcD   = 1;
510                             RegWriteD = 0;
511                             RegSrc    = 'b01;
512                             ImmSrc    = 'b10;
513                             ALUControlD = 'b00;  // do an add
514                             FlagWriteD = 0;
515                             BranchD = 1;
516                         end
517
518                         // Greater or Equal
519                         4'b1010 : begin
520                             //PCSrcD    = 1;
521                             MemToRegD = 0;
522                             MemWriteD = 0;
523                             ALUSrcD   = 1;
524                             RegWriteD = 0;
525                             RegSrc    = 'b01;
526                             ImmSrc    = 'b10;
527                             ALUControlD = 'b00;  // do an add
528                             FlagWriteD = 0;
529                             BranchD = 1;
530                         end
531
532                         // Greater
533                         4'b1100 : begin
534                             //PCSrcD    = 1;
535                             MemToRegD = 0;
536                             MemWriteD = 0;
537                             ALUSrcD   = 1;
538                             RegWriteD = 0;
539                             RegSrc    = 'b01;
540                             ImmSrc    = 'b10;
541                             ALUControlD = 'b00;  // do an add
542                             FlagWriteD = 0;
543                             BranchD = 1;
544                         end
545
546                         // Less or Equal
547                         4'b1101 : begin
548                             //PCSrcD    = 1;
549                             MemToRegD = 0;
550                             MemWriteD = 0;
551                             ALUSrcD   = 1;
552                             RegWriteD = 0;
553                             RegSrc    = 'b01;
554                             ImmSrc    = 'b10;
555                             ALUControlD = 'b00;  // do an add
556                             FlagWriteD = 0;
557                             BranchD = 1;
558                         end
559
560                         // Less
561                         4'b1011 : begin
562                             //PCSrcD    = 1;
563                             MemToRegD = 0;
564                             MemWriteD = 0;
565                             ALUSrcD   = 1;
566                             RegWriteD = 0;
567                             RegSrc    = 'b01;
```

```
568                             ImmSrc   = 'b10;
569                             ALUControlD = 'b00;  // do an add
570                             FlagWriteD = 0;
571                             BranchD = 1;
572                          end
573
574                          default: begin
575                             //PCSrcD    = 0;
576                             MemToRegD = 0; // doesn't matter
577                             MemWriteD = 0;
578                             ALUSrcD   = 0;
579                             RegWriteD = 0;
580                             RegSrc   = 'b00;
581                             ImmSrc   = 'b00;
582                             ALUControlD = 'b00;  // do an add
583                             FlagWriteD = 0;
584                             BranchD = 0;
585                          end
586                       endcase
587
588
589                end
590
591             default: begin
592                   //PCSrcD    = 0;
593                      MemToRegD = 0;
594                      MemWriteD = 0;
595                      ALUSrcD   = 0;
596                      RegWriteD = 0;
597                      RegSrc   = 'b00;
598                      ImmSrc   = 'b00;
599                      ALUControlD = 'b00;
600                      FlagWriteD = 0;
601                      BranchD = 0;
602             end
603            endcase
604       end
605
606
607   endmodule
```