```
1   /*
2      DataPath.v stands as the top-level module, instantiating all registers and the ALU.
3      For Phase 1 purposes, all control signals are *simulated* to validate the correctness of
    datapath operations.
4   */
5
6   `timescale 1ns / 1ps
7
8   /*
9      DESIGN DECISION:
10     Our group decided to incorporate as much vectorization as possible, given the sparsity
    of control signals such as:
11     register enables, register 'reads', and ALU operation IDs. This allows for far cleaner
    and legible description,
12     without loss of convenience associated to referencing registers by name.
13
14     All register and operation IDs are associated their own indices. This is achieved with
    the following const. definitions.
15
16     Enable signals for general purpose registers (like 'rXin') are grouped under GRin
    (One-Hot-Encoded).
17     Enable signals for datapath registers (like 'PCin', 'IRin', 'MARin', 'MDRin', etc.) are
    grouped under DPin (One-Hot-Encoded).
18     Read signals for general purpose registers (like 'rXout') are grouped under GRout
    (One-Hot-Encoded).
19     Read signals for datapath registers (like 'PCout', 'IRout', 'MARout', 'MDRout', etc.)
    are grouped under DPout (One-Hot-Encoded).
20     ALU control signals (like 'add', 'sub', etc.) are grouped under ALUopp (One-Hot-Encoded).
21  */
22
23
24  `define PC 0
25  `define IR 1
26  `define Y 2
27  `define MAR 3
28  `define MDR 4
29  `define INPORT 5
30  `define OUTPORT 6
31  `define Z   7 //Z used for Enable
32  `define ZHI 8 //ZHI / ZLO used for outputs
33  `define ZLO 9
34  `define HI 10
35  `define LO 11
36  `define READ 12
37  `define C   13
38
39  `define ADD 0
40  `define SUB 1
41  `define NEG 2
42  `define MUL 3
43  `define DIV 4
44  `define AND 5
45  `define OR  6
46  `define ROR 7
47  `define ROL 8
48  `define SLL 9
49  `define SRA 10
50  `define SRL 11
51  `define NOT 12
52  `define INC 13
53
54  module DataPath (
55     /*******Control Signals******/
56
57     input clk, clr, CONin,
58     input Gra, Grb, Grc, Rin, Rout, BAout, RAM_wr,
59
60
61     // Register Write Control
62     input [15:0] DPin,
63
64     // Register Read Control
65     input [15:0] DPout,
66
67     // ALU Control
```

```
68      input [15:0] ALUopp,
69
70      // Input for Disconnected register ends (INPortIn)
71      input  [31:0] INPORTin,
72      // Output Disconnected Register ends (IRout, MARout, OUTPORTout)
73      output [31:0] OUTPORTout,
74      output CON
75  );
76
77      // Output from Bus
78      wire [31:0] BusMuxOut;
79
80      // Inputs to Bus
81      wire [31:0] BusMuxInGR;
82      wire [31:0] BusMuxInPC;
83      wire [31:0] BusMuxInINPORT;
84      wire [31:0] BusMuxInMDR;
85      wire [31:0] BusMuxInHI;
86      wire [31:0] BusMuxInLO;
87
88      wire [31:0] YtoA;
89      wire [63:0] CtoZ;
90
91      wire [63:0] ZtoBusMux;
92
93      wire [31:0] IRout;
94      wire [31:0] MARout;
95      wire [31:0] C;
96      wire [15:0] GRin;
97      wire [15:0] GRout;
98
99      wire [31:0] Mdatain;
100
101     wire GR_Read;
102     assign GR_Read = |GRout;
103
104     wire [31:0] MDRin;
105     assign MDRin = DPin[`READ] ?  Mdatain : BusMuxOut ;
106
107     // General Purpose Register instantiation
108     R0_R15_GenPurposeRegs GR(clk, clr, BAout, BusMuxOut, GRin, GRout, BusMuxInGR);
109
110     // All Datapath Register instantiations.
111     register PC       (clr, clk, DPin[`PC], BusMuxOut, BusMuxInPC);
112     register IR       (clr, clk, DPin[`IR], BusMuxOut, IRout);
113     register Y        (clr, clk, DPin[`Y], BusMuxOut, YtoA);
114     register MAR      (clr, clk, DPin[`MAR], BusMuxOut, MARout);
115     register MDR      (clr, clk, DPin[`MDR], MDRin, BusMuxInMDR);
116     register INPORT   (clr, clk, DPin[`INPORT], INPORTin, BusMuxInINPORT);
117     register OUTPORT  (clr, clk, DPin[`OUTPORT], BusMuxOut, OUTPORTout);
118     register HI       (clr, clk, DPin[`HI], BusMuxOut, BusMuxInHI);
119     register LO       (clr, clk, DPin[`LO], BusMuxOut, BusMuxInLO);
120     register Z        (clr, clk, DPin[`Z] , CtoZ, ZtoBusMux);
121        defparam Z.DATA_WIDTH_IN = 64,
122                 Z.DATA_WIDTH_OUT = 64;
123
124     conditional_ff_logic CON_FF (IRout[20:19], BusMuxOut, CONin, clk, CON);
125
126     // Bus
127     Bus DataPathBus   (BusMuxInGR, BusMuxInHI, BusMuxInLO, ZtoBusMux[63:32], ZtoBusMux[31:0],
     BusMuxInPC, BusMuxInMDR, BusMuxInINPORT, C,
128                        GR_Read, DPout[`HI], DPout[`LO], DPout[`ZHI], DPout[`ZLO], DPout[`PC],
     DPout[`MDR], DPout[`INPORT], DPout[`C], BusMuxOut );
129
130     // ALU
131     ALU DP_ALU        (YtoA, BusMuxOut, ALUopp, clk, CtoZ);
132
133     // Select And Encode Module
134
135     SelectAndEncodeLogic DP_SnEL(IRout, Gra, Grb, Grc, Rin, Rout, BAout, C, GRin, GRout);
136
137     // RAM
138     ram DP_ram (clk, RAM_wr, MARout[8:0], MARout[8:0], BusMuxInMDR, Mdatain);
139
140 endmodule
```

```
141
142
143
144   module datapath_tb();
145
146      // Control Signals
147      reg clk, clr, CONin;
148      reg Gra, Grb, Grc, Rin, Rout, BAout, RAM_wr;
149
150      // Register Write Control
151      reg [15:0] DPin;
152
153      // Register Read Control
154      reg [15:0] DPout;
155
156      // ALU Control
157      reg [15:0] ALUopp;
158
159      // Input for Disconnected register ends (INPortIn)
160      reg [31:0] INPORTin;
161      // Output Disconnected Register ends (OUTPORTout)
162      wire [31:0] OUTPORTout;
163      //Output for CON
164      wire CON;
165
166      // Unit Under Test
167     DataPath UUT (clk, clr, CONin, Gra, Grb, Grc, Rin, Rout, BAout, RAM_wr, DPin, DPout,
    ALUopp, INPORTin, OUTPORTout, CON);
168
169      // Establishing Clock Behaviour
170      parameter clock_period = 20;
171      initial begin
172         clk <= 0;
173         forever #(clock_period/2) clk <= ~clk;
174      end
175
176      reg [4:0] op_code;
177      reg [15:0] alu_code;
178
179      parameter LD = 5'b00000, LDI = 5'b00001, ST = 5'b00010,
180                ADD = 5'b00011, SUB = 5'b00100, AND = 5'b00101, OR = 5'b00110, ROR = 5'b00111,
    ROL = 5'b01000, SHR = 5'b01001, SHRA = 5'b01010, SHL = 5'b01011,
181                ADDI = 5'b01100, ANDI = 5'b01101, ORI = 5'b01110, DIV = 5'b01111, MUL =
    5'b10000, NEG = 5'b10001, NOT = 5'b10010,
182                BR = 5'b10011, JAL = 5'b10100, JR = 5'b10101, IN = 5'b10110, OUT = 5'b10111,
    MFLO = 5'b11000, MFHI = 5'b11011,
183                NOP = 5'b11010, HALT = 5'b11011;
184
185      initial begin
186      //---------Default Values----------//
187         init_zeros();
188
189         @(posedge clk)
190
191      //---------Pre-Load Values----------//
192
193         load_reg(32'h0180_0000, 32'hB6);    // Load R3
194
195         //load_reg(32'h0280_0000, 32'h0);      // Load R5
196
197      //---------Specify Instr-----------//
198
199         op_code = ST;
200         alu_code = op_to_alu (op_code);
201
202      //---------Fetch Instruction---------//
203         fetch_instr ();
204
205      //---------Preform Instruction---------//
206         case (op_code)
207
208            ADD, SUB, AND, OR, ROR, ROL, SHR, SHRA, SHL: ALU (alu_code);
209
210            ADDI, ANDI, ORI: ALU_imm (alu_code);
211
```

```
212             MUL: ALU_mul (alu_code);
213
214             DIV: ALU_div (alu_code);
215
216             NEG, NOT: ALU_neg_not (alu_code);
217
218             BR: BRANCH();
219
220             JAL: JAL_T();
221
222             JR: JR_T();
223
224             LD: LOAD();
225
226             LDI: LOAD_imm();
227
228             ST: STORE();
229
230             MFHI: Move_HI();
231
232             MFLO: Move_LO();
233
234             OUT: out();
235
236             IN: in();
237
238             default: $stop;
239
240         endcase
241     //-------End of Simulation--------//
242         @(posedge clk)
243         $stop;
244     end
245
246     //------Functions-------//
247
248     //Convert from the 5-bit machine op_code to the 16-bit bus used to control the ALU
249     function [15:0] op_to_alu (input [4:0] op_code);
250         begin
251             case (op_code)
252                 ADD, ADDI:  op_to_alu = 1 << `ADD;
253                 SUB:        op_to_alu = 1 << `SUB;
254                 AND, ANDI:  op_to_alu = 1 << `AND;
255                 OR , ORI :  op_to_alu = 1 << `OR ;
256                 ROR:        op_to_alu = 1 << `ROR;
257                 ROL:        op_to_alu = 1 << `ROL;
258                 SHR:        op_to_alu = 1 << `SRL;
259                 SHRA:       op_to_alu = 1 << `SRA;
260                 SHL:        op_to_alu = 1 << `SLL;
261                 DIV:        op_to_alu = 1 << `DIV;
262                 MUL:        op_to_alu = 1 << `MUL;
263                 NEG:        op_to_alu = 1 << `NEG;
264                 NOT:        op_to_alu = 1 << `NOT;
265                 default:    op_to_alu = 16'b0;
266             endcase
267         end
268     endfunction
269
270     //--------General Tasks-------------//
271
272     //Set inital state of the CPU to zeros
273     task init_zeros ();
274         begin
275             clr <= 0;
276             Gra<= 0; Grb<= 0; Grc<= 0; Rin<= 0; Rout<= 0; BAout<= 0; RAM_wr <= 0; CONin <= 0;
277             DPin <= 16'b0; DPout <= 16'b0;
278             ALUopp <= 16'b0;
279             op_code <= 5'b0; alu_code <= 16'b0;
280         end
281     endtask
282
283     //Pre-Load a value into a register, machine_code must specify the register in the Ra slot
284     task load_reg (input [31:0] machine_code, input [31:0] value);
285         begin
286             INPORTin <= machine_code; DPin[`INPORT] <= 1;
```

```
287
288             @(posedge clk)
289
290             INPORTin <= value; DPin[`INPORT] <= 1;
291             DPout[`INPORT] <= 1; DPin[`IR] <= 1;
292
293             @(posedge clk)
294             INPORTin <= 32'b0; DPin[`INPORT] <= 0; DPin[`IR] <= 0;
295             DPout[`INPORT] <= 1; Gra <= 1; Rin <= 1;
296
297             @(posedge clk)
298             DPout[`INPORT] <= 0; Gra <= 0; Rin <= 0;
299
300          end
301       endtask
302
303       // Pre-load a value into PC
304       task load_pc (input [31:0] value);
305          begin
306             INPORTin <= value; DPin[`INPORT] <= 1;
307
308             @(posedge clk)
309
310             INPORTin <= 32'b0; DPin[`INPORT] <= 0;
311             DPout[`INPORT] <= 1; DPin[`PC] <= 1;
312
313             @(posedge clk)
314             DPout[`INPORT] <= 0; DPin[`PC] <= 0;
315          end
316       endtask
317
318
319       //T0 - T2 Cycles are used to fetch instructions from memory into the IR
320       task fetch_instr ();
321          begin
322             //T0
323             DPout[`PC] <= 1; DPin[`MAR] <= 1; ALUopp[`INC] <= 1; DPin[`Z] <= 1;
324             @(posedge clk)
325             DPout[`PC] <= 0; DPin[`MAR] <= 0; ALUopp[`INC] <= 0; DPin[`Z] <= 0;
326             //T1
327             DPout[`ZLO] <= 1; DPin[`PC] <= 1;
328             DPin[`MDR] <= 1; DPin[`READ] <= 1;
329             @(posedge clk)
330             DPout[`ZLO] <= 0; DPin[`PC] <= 0;
331             DPin[`MDR] <= 0; DPin[`READ] <= 0;
332             //T2
333             DPout[`MDR] <= 1; DPin[`IR] <= 1;
334             @(posedge clk)
335             DPout[`MDR] <= 0; DPin[`IR] <= 0;
336          end
337       endtask
338
339       //-------ALU tasks-------//
340
341       //General ALU used for most ALU opperations
342       task ALU (input [15:0] alu_code);
343          begin
344             //T3
345             Rout <= 1; Grb <= 1; DPin[`Y] <= 1;
346             @(posedge clk)
347             Rout <= 0; Grb <= 0; DPin[`Y] <= 0;
348             //T4
349             Rout <= 1; Grc <= 1; ALUopp <= alu_code; DPin[`Z] <= 1;
350             @(posedge clk)
351             Rout <= 0; Grc <= 0; ALUopp <= 16'b0; DPin[`Z] <= 0;
352             //T5
353             DPout[`ZLO] <= 1; Rin <= 1; Gra <= 1;
354             @(posedge clk)
355             DPout[`ZLO] <= 0; Rin <= 0; Gra <= 0;
356          end
357       endtask
358
359       //Used for ALU opperations that involve a Immediate value from IR
360       task ALU_imm (input [15:0] alu_code);
361          begin
```

```
            //T3
            Rout <= 1; Grb <= 1; DPin[`Y] <= 1;
            @(posedge clk)
            Rout <= 0; Grb <= 0; DPin[`Y] <= 0;
            //T4
            DPout[`C] <= 1; ALUopp <= alu_code; DPin[`Z] <= 1;
            @(posedge clk)
            DPout[`C] <= 0; ALUopp <= 16'b0; DPin[`Z] <= 0;
            //T5
            DPout[`ZLO] <= 1; Rin <= 1; Gra <= 1;
            @(posedge clk)
            DPout[`ZLO] <= 0; Rin <= 0; Gra <= 0;
        end
    endtask

    //Used for the mulitplicaiton opperation
    task ALU_mul (input [15:0] alu_code);
        begin
            //T3
            Rout <= 1; Gra <= 1; DPin[`Y] <= 1;
            @(posedge clk)
            Rout <= 0; Gra <= 0; DPin[`Y] <= 0;
            //T4
            Rout <= 1; Grb <= 1; ALUopp <= alu_code; DPin[`Z] <= 1;
            @(posedge clk)
            Rout <= 0; Grb <= 0; ALUopp <= 16'b0; DPin[`Z] <= 0;
            //T5
            DPout[`ZLO] <= 1; DPin[`LO] <= 1;
            @(posedge clk)
            DPout[`ZLO] <= 0; DPin[`LO] <= 0;
            //T6
            DPout[`ZHI] <= 1; DPin[`HI] <= 1;
            @(posedge clk)
            DPout[`ZHI] <= 0; DPin[`HI] <= 0;
        end
    endtask

    //Used for the division opperation
    task ALU_div (input [15:0] alu_code);
        begin
            //T3
            Rout <= 1; Gra <= 1; DPin[`Y] <= 1;
            @(posedge clk)
            Rout <= 0; Gra <= 0; DPin[`Y] <= 0;
            //T4
            Rout <= 1; Grb <= 1; ALUopp <= alu_code; DPin[`Z] <= 1;
            repeat (34) @(posedge clk)
            Rout <= 0; Grb <= 0; ALUopp <= 16'b0; DPin[`Z] <= 0;
            //T5
            DPout[`ZLO] <= 1; DPin[`LO] <= 1;
            @(posedge clk)
            DPout[`ZLO] <= 0; DPin[`LO] <= 0;
            //T6
            DPout[`ZHI] <= 1; DPin[`HI] <= 1;
            @(posedge clk)
            DPout[`ZHI] <= 0; DPin[`HI] <= 0;
        end
    endtask

    //Used for single argument ALU operation such as neg/not
    task ALU_neg_not (input [15:0] alu_code);
        begin
            //T3
            Rout <= 1; Grb <= 1; ALUopp <= alu_code; DPin[`Z] <= 1;
            @(posedge clk)
            Rout <= 0; Grb <= 0; ALUopp <= 16'b0; DPin[`Z] <= 0;
            //T4
            DPout[`ZLO] <= 1; Rin <= 1; Gra <= 1;
            @(posedge clk)
            DPout[`ZLO] <= 0; Rin <= 0; Gra <= 0;
        end
    endtask

    //-------BRANCH AND JUMP INSTRUCTIONS-------//
    task BRANCH();
```

```
437         begin
438            // T3
439            Gra <= 1; Rout <= 1; CONin <= 1;
440            @(posedge clk)
441            Gra <= 0; Rout <= 0; CONin <= 0;
442
443            #1;
444            if (CON) begin
445               // T4
446               DPout[`PC] <= 1; DPin[`Y] <= 1;
447               @(posedge clk)
448               DPout[`PC] <= 0; DPin[`Y] <= 0;
449
450               // T5
451               DPout[`C] <= 1; ALUopp[`ADD] <= 1; DPin[`Z] <= 1;
452               @(posedge clk)
453               DPout[`C] <= 0; ALUopp[`ADD] <= 0; DPin[`Z] <= 0;
454
455               // T6
456               DPout[`ZLO] <= 1; DPin[`PC] <= 1;
457               @(posedge clk)
458               DPout[`Z] <= 0; DPin[`PC] <= 0;
459            end
460         end
461      endtask
462
463      task JAL_T();
464         begin
465            // T3
466            DPout[`PC] <= 1; Rin <= 1; Grb <= 1; // NEED TO MAKE SURE 4'b1000 IS IN GRB SLOT??
467            @(posedge clk)
468            DPout[`PC] <= 0; Rin <= 0; Grb <= 0;
469
470            // T4
471            Rout <= 1; Gra <= 1; DPin[`PC] <= 1;
472            @(posedge clk)
473            Rout <= 0; Gra <= 0; DPin[`PC] <= 0;
474         end
475      endtask
476
477      task JR_T();
478         begin
479            // T3
480            DPin[`PC] <= 1; Rout <= 1; Gra <= 1;
481            @(posedge clk)
482            DPin[`PC] <= 0; Rout <= 0; Gra <= 0;
483         end
484      endtask
485
486      //-------Load and Store Tasks-------//
487      task LOAD();
488         begin
489            //T3
490            Grb <= 1; BAout <= 1; DPin[`Y] <= 1;
491            @(posedge clk)
492            Grb <= 0; BAout <= 0; DPin[`Y] <= 0;
493
494            //T4
495            DPout[`C] <= 1; ALUopp[`ADD] <= 1; DPin[`Z] <= 1;
496            @(posedge clk)
497            DPout[`C] <= 0; ALUopp[`ADD] <= 0; DPin[`Z] <= 0;
498
499            //T5
500            DPout[`ZLO] <= 1; DPin[`MAR] <= 1;
501            @(posedge clk)
502            DPout[`ZLO] <= 0; DPin[`MAR] <= 0;
503
504            //T6
505            DPin[`MDR] <= 1; DPin[`READ] <= 1;
506            @(posedge clk)
507            DPin[`MDR] <= 0; DPin[`READ] <= 0;
508
509            //T7
510            DPout[`MDR] <= 1; Gra <= 1; Rin <= 1;
511            @(posedge clk)
```

```
512             DPout[`MDR] <= 0; Gra <= 0; Rin <= 0;
513
514          end
515       endtask
516
517       task LOAD_imm();
518          begin
519             // T3
520             Grb <= 1; BAout <= 1; DPin[`Y] <= 1;
521             @(posedge clk)
522             Grb <= 0; BAout <= 0; DPin[`Y] <= 0;
523
524             //T4
525             DPout[`C] <= 1; ALUopp[`ADD] <= 1; DPin[`Z] <= 1;
526             @(posedge clk)
527             DPout[`C] <= 0; ALUopp[`ADD] <= 0; DPin[`Z] <= 0;
528
529             //T5
530             DPout[`ZLO] <= 1; Gra <= 1; Rin <= 1;
531             @(posedge clk)
532             DPout[`ZLO] <= 0; Gra <= 0; Rin <= 0;
533
534             end
535       endtask
536
537       task STORE();
538          begin
539             //T3
540             Grb <= 1; BAout <= 1; DPin[`Y] <= 1;
541             @(posedge clk)
542             Grb <= 0; BAout <= 0; DPin[`Y] <= 0;
543
544             //T4
545             DPout[`C] <= 1; ALUopp[`ADD] <= 1; DPin[`Z] <= 1;
546             @(posedge clk)
547             DPout[`C] <= 0; ALUopp[`ADD] <= 0; DPin[`Z] <= 0;
548
549             //T5
550             DPout[`ZLO] <= 1; DPin[`MAR] <= 1;
551             @(posedge clk)
552             DPout[`ZLO] <= 0; DPin[`MAR] <= 0;
553
554             //T6
555             DPin[`MDR] <= 1; Gra <= 1; Rout <= 1;
556             @(posedge clk)
557             DPin[`MDR] <= 0; Gra <= 0; Rout <= 0;
558
559             //T7
560             RAM_wr <= 1;
561             @(posedge clk)
562             RAM_wr <= 0;
563
564          end
565       endtask
566
567          //-------Move and Port Tasks-------//
568
569       task Move_HI();
570          begin
571
572          Rin <= 1; Gra <= 1; DPout[`HI] <= 1;
573          @(posedge clk)
574          Rin <= 0; Gra <= 0; DPout[`HI] <= 0;
575
576          end
577       endtask
578
579       task Move_LO();
580          begin
581
582          Rin <= 1; Gra <= 1; DPout[`LO] <= 1;
583          @(posedge clk)
584          Rin <= 0; Gra <= 0; DPout[`LO] <= 0;
585
586          end
```

```
587       endtask
588
589       task out ();
590          begin
591
592          Rout <= 1; Gra <= 1; DPin[`OUTPORT] <= 1;
593          @(posedge clk)
594          Rout <= 0; Gra <= 0; DPin[`OUTPORT] <= 0;
595
596          end
597       endtask
598
599       task in ();
600          begin
601
602          Rin <= 1; Gra <= 1; DPout[`INPORT] <= 1;
603          @(posedge clk)
604          Rin <= 0; Gra <= 0; DPout[`INPORT] <= 0;
605
606          end
607       endtask
608
609    endmodule
610
```