```
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
38  `define ADD 0
39  `define SUB 1
40  `define NEG 2
41  `define MUL 3
42  `define DIV 4
43  `define AND 5
44  `define OR  6
45  `define ROR 7
46  `define ROL 8
47  `define SLL 9
48  `define SRA 10
49  `define SRL 11
50  `define NOT 12
51  `define INC 13
52
53  module DataPath (
54     /*******Control Signals******/
55     input clk, clr,
56
57     //Register Write Control
58     input [15:0] GRin,
59     input [15:0] DPin,
60
61     //Register Read Control
62     input [15:0] GRout,
63     input [15:0] DPout,
64
65     //ALU Control
66     input [15:0] ALUopp,
```

```
67
68      //Input for Disconnected register ends (INPortIn)
69      input  [31:0]INPORTin,
70      input  [31:0]Mdatain,
71      //Output Disconnected Register ends (IRout, MARout, OUTPORTout)
72      output [31:0] IRout,
73      output [31:0] MARout,
74      output [31:0] OUTPORTout,
75      output [31:0] BusMuxInMDR
76  );
77
78
79      wire [31:0] BusMuxOut;
80
81
82      wire [31:0] BusMuxInGR;
83      wire [31:0] BusMuxInGR2;
84      wire [31:0] BusMuxInPC;
85      wire [31:0] BusMuxInINPORT;
86      wire [31:0] BusMuxInHI;
87      wire [31:0] BusMuxInLO;
88
89      wire [31:0] YtoA;
90      wire [63:0] CtoZ;
91
92      wire [63:0] ZtoBusMux;
93
94      wire GR_Read;
95      assign GR_Read = |GRout;
96
97      wire [31:0] MDRin;
98      assign MDRin = DPin[`READ] ?  Mdatain : BusMuxOut ;
99
100     // General Purpose Register instantiation
101     R0_R15_GenPurposeRegs GR(clk, clr, BusMuxOut, GRin, GRout, BusMuxInGR, BusMuxInGR2);
102
103     // All Datapath Register instantiations.
104     register PC       (clr, clk, DPin[`PC], BusMuxOut, BusMuxInPC);
105     register IR       (clr, clk, DPin[`IR], BusMuxOut, IRout);
106     register Y        (clr, clk, DPin[`Y], BusMuxOut, YtoA);
107     register MAR      (clr, clk, DPin[`MAR], BusMuxOut, MARout);
108     register MDR      (clr, clk, DPin[`MDR], MDRin, BusMuxInMDR);
109     register INPORT   (clr, clk, DPin[`INPORT], INPORTin, BusMuxInINPORT);
110     register OUTPORT  (clr, clk, DPin[`OUTPORT], BusMuxOut, OUTPORTout);
111     register HI       (clr, clk, DPin[`HI], BusMuxOut, BusMuxInHI);
112     register LO       (clr, clk, DPin[`LO], BusMuxOut, BusMuxInLO);
113     register Z        (clr, clk, DPin[`Z] , CtoZ, ZtoBusMux);
114        defparam Z.DATA_WIDTH_IN = 64,
115                 Z.DATA_WIDTH_OUT = 64;
116
117     // Bus
118     Bus DataPathBus   (BusMuxInGR, BusMuxInHI, BusMuxInLO, ZtoBusMux[63:32], ZtoBusMux[31:0],
     BusMuxInPC, BusMuxInMDR, BusMuxInINPORT,
119                        GR_Read, DPout[`HI], DPout[`LO], DPout[`ZHI], DPout[`ZLO], DPout[`PC],
     DPout[`MDR], DPout[`INPORT], BusMuxOut );
120
121     // ALU
122     ALU DP_ALU        (YtoA, BusMuxOut, ALUopp, clk, CtoZ);
123
124 endmodule
125
126
127
128 module datapath_tb();
129
130     // Control Signals
131     reg clk, clr;
132
133     // Register Write Control
134     reg [15:0] GRin;
135     reg [15:0] DPin;
136
137     // Register Read Control
138     reg [15:0] GRout;
```

```
139      reg [15:0] DPout;
140
141      // ALU Control
142      reg [15:0] ALUopp;
143
144      // Input for Disconnected register ends (INPortIn)
145      reg [31:0] INPORTin;
146      reg [31:0] Mdatain;
147      // Output Disconnected Register ends (IRout, MARout, OUTPORTout)
148      wire [31:0] IRout;
149      wire [31:0] MARout;
150      wire [31:0] OUTPORTout;
151      wire [31:0] BusMuxInMDR;
152
153      // Unit Under Test
154      DataPath UUT (clk, clr, GRin, DPin, GRout, DPout, ALUopp, INPORTin, Mdatain, IRout,
     MARout, OUTPORTout, BusMuxInMDR);
155
156      // Establishing Clock Behaviour
157      parameter clock_period = 20;
158      initial begin
159         clk <= 0;
160         forever #(clock_period/2) clk <= ~clk;
161      end
162
163      initial begin
164         //---------Default Values----------//
165
166         // Clear signal
167         clr <= 0;
168         // Register Identifiers: GR = General Register, DP = Datapath Register
169         GRin <= 16'b0; DPin <= 16'b0; GRout <= 16'b0; DPout <= 16'b0;
170         //ALU Control.
171         ALUopp <= 16'b0;
172         // Memory Data In.
173         Mdatain <= 32'b0;
174
175
176         @(posedge clk)
177         //---------Preset R3----------//
178
179         load_reg(4'd2, -{32'd374});
180
181         //---------Preset R7----------//
182
183         load_reg(4'd6, {32'd10});
184
185
186         //---------AND R4, R3, R7---------//
187
188         T0 ();
189         T1 (32'h81300000);
190         T2 ();
191         T3 (4'd2);
192         T4 (4'd6, `MUL);
193         T5 (4'd0, 1'b1); //HILO
194         T6 ();
195
196         @(posedge clk)
197         $stop;
198      end
199
200      task load_reg (input [3:0] reg_id, input [31:0] value);
201         begin
202            Mdatain <= value; DPin[`READ] <= 1; DPin[`MDR] <= 1;
203
204            @(posedge clk)
205            Mdatain <= 32'b0; DPin[`READ] <= 0; DPin[`MDR] <= 0; // Reset from previous cycle
206            DPout[`MDR] <= 1; GRin[reg_id] <= 1;
207
208            @(posedge clk)
209            DPout[`MDR] <= 0; GRin[reg_id] <= 0;
210         end
211      endtask
```

```
212
213      task T0 ();
214         begin
215            DPout[`PC] <= 1; DPin[`MAR] <= 1; ALUopp[`INC] <= 1; DPin[`Z] <= 1; // MAR <-
     [PC], PC <- [PC] + 1
216            @(posedge clk)
217            DPout[`PC] <= 0; DPin[`MAR] <= 0; ALUopp[`INC] <= 0; DPin[`Z] <= 0;
218         end
219      endtask
220
221      task T1 (input [31:0] op_code);
222         begin
223            DPout[`ZLO] <= 1; DPin[`PC] <= 1;   // Accept incremented value.
224            DPin[`MDR] <= 1; DPin[`READ] <= 1; Mdatain <= op_code; // MDR <- op_code
225
226            @(posedge clk)
227            DPout[`ZLO] <= 0; DPin[`PC] <= 0;
228            DPin[`MDR] <= 0; DPin[`READ] <= 0; Mdatain <= 32'b0;
229         end
230      endtask
231
232      task T2 ();
233         begin
234            DPout[`MDR] <= 1; DPin[`IR] <= 1; // IR <- [MDR] (op_code)
235
236            @(posedge clk)
237            DPout[`MDR] <= 0; DPin[`IR] <= 0;
238         end
239      endtask
240
241      task T3 (input [3:0] reg_id);
242         begin
243            GRout[reg_id] <= 1; DPin[`Y] <= 1; // Y <- [GR[reg_id]]
244
245            @(posedge clk)
246            GRout[reg_id] <= 0; DPin[`Y] <= 0;
247         end
248      endtask
249
250      task T4 (input [3:0] reg_id, input [3:0] opp);
251         begin
252            GRout[reg_id] <= 1; ALUopp[opp] <= 1; DPin[`Z] <= 1; // Z <- [Y] opp [GR[reg_id]]
253            if (opp == `DIV) begin
254               repeat (34) begin
255                  @(posedge clk);
256               end
257            end
258            @(posedge clk)
259            GRout[reg_id] <= 0; ALUopp[opp] <= 0; DPin[`Z] <= 0;
260         end
261      endtask
262
263      task T5 (input [3:0] reg_id, input HILO);
264         begin
265            DPout[`ZLO] <= 1;
266            if (HILO)
267               DPin[`LO] <= 1;
268            else
269               GRin[reg_id] <= 1; // GR[reg_id] <- [ZLO]
270
271            @(posedge clk)
272            DPout[`ZLO] <= 0; GRin[reg_id] <= 0; DPin[`LO] <= 0;
273         end
274      endtask
275
276      task T6 ();
277         begin
278            DPout[`ZHI] <= 1; DPin[`HI] <= 1;
279
280            @(posedge clk)
281            DPout[`ZHI] <= 0; DPin[`HI] <= 0;
282         end
283      endtask
284
```

```
285
286     endmodule
287
```