

```

1  `timescale 1ns / 1ps
2
3  `define PC 0
4  `define IR 1
5  `define Y 2
6  `define MAR 3
7  `define MDR 4
8  `define INPORT 5
9  `define OUTPORT 6
10 `define Z 7 //Z used for Enable
11 `define ZHI 8 //ZHI / ZLO used for outputs
12 `define ZLO 9
13 `define HI 10
14 `define LO 11
15 `define READ 12
16 `define C 13
17
18 `define ADD 0
19 `define SUB 1
20 `define NEG 2
21 `define MUL 3
22 `define DIV 4
23 `define AND 5
24 `define OR 6
25 `define ROR 7
26 `define ROL 8
27 `define SLL 9
28 `define SRA 10
29 `define SRL 11
30 `define NOT 12
31 `define INC 13
32
33 module Control (
34     // External Inputs
35     input reset, stop, clk,
36
37     // Datapath Inputs
38     input CON,
39     input [31:27] IRop,
40
41     // Execution Control Signals
42     output reg clr, CONin, RAM_wr,
43
44     // General Purpose Register Control
45     output reg Gra, Grb, Grc, Rin, Rout, BAout,
46
47     // Datapath Register Control
48     output reg [15:0] DPin, DPout,
49
50     // ALU Control
51     output [15:0] ALUopp,
52
53     // Status indicator
54     output run
55 );
56
57
58
59
60
61 //----- INTERNAL REGISTERS -----//
62
63 reg [4:0] ps, ns; // State Registers
64 reg [3:0] op; // ALU operation state.
65
66 //-----//
67
68 //----- PARAMETER DEFINITIONS -----//
69
70 // IR op code labels
71 parameter LD = 5'b00000, LDI = 5'b00001, ST = 5'b00010,
72 ADD = 5'b00011, SUB = 5'b00100, AND = 5'b00101, OR = 5'b00110, ROR = 5'b00111,
73 ROL = 5'b01000, SHR = 5'b01001, SHRA = 5'b01010, SHL = 5'b01011,
74 ADDI = 5'b01100, ANDI = 5'b01101, ORI = 5'b01110, DIV = 5'b01111, MUL =
5'b10000, NEG = 5'b10001, NOT = 5'b10010,

```

```

75         BR = 5'b10011, JAL = 5'b10100, JR = 5'b10101, IN = 5'b10110, OUT = 5'b10111,
MFLO = 5'b11000, MFHI = 5'b11001,
76         NOP = 5'b11010, HALT = 5'b11011;
77
78     // State variable labels
79     parameter Treset = 5'b11111, Thalt = 5'b11110, T0 = 5'b00000, T1 = 5'b00001, T2 =
5'b00010, T3 = 5'b00011, // States common to all instructions
80     T4ALU = 5'b00100, T4dm = 5'b00101, T4ALUimm = 5'b00110, T4nn = 5'b00111, T4br =
5'b01000, T4ldst = 5'b01001, T4jal = 5'b01010, // T4 states
81     T5ALU = 5'b01011, T5br = 5'b01100, T5ldi = 5'b01101, T5ld = 5'b01110, T5st =
5'b01111, // T5 states
82     T6dm = 5'b10000, T6br = 5'b10001, T6ld = 5'b10010, T6st = 5'b10011, // T6 states
83     T7ld = 5'b10100, T7st = 5'b10101; // T7 states
84
85
86     //-----//
87
88     //----- NEXT-STATE LOGIC AND CONTROL SIGNAL ASSERTION -----//
89
90
91
92     always @(*) begin
93         // Default all control signals to zero.
94         zeroes ();
95
96         // Assert next state behaviour and control signals according to present state.
97         case (ps)
98             // INSTRUCTION FETCH STAGE
99             T0:
100                 begin
101                     ns = T1;
102                     DPout[`PC] = 1'b1;
103                     DPin[`MAR] = 1'b1;
104                     DPin[`Z] = 1'b1;
105                 end
106
107             T1:
108                 begin
109                     ns = T2;
110                     DPout[`ZLO] = 1'b1;
111                     DPin[`PC] = 1'b1;
112                     DPin[`MDR] = 1'b1;
113                     DPin[`READ] = 1'b1;
114                 end
115
116             T2:
117                 begin
118                     ns = T3;
119                     DPout[`MDR] = 1'b1;
120                     DPin[`IR] = 1'b1;
121                 end
122
123             // DECODE STAGE
124             T3: case(IROP[31:27])
125                 MUL, DIV:
126                     begin
127                         ns = T4dm;
128                         Rout = 1'b1;
129                         Gra = 1'b1;
130                         DPin[`Y] = 1'b1;
131                     end
132
133                 ADD, SUB, AND, OR, ROR, ROL, SHR, SHRA, SHL:
134                     begin
135                         ns = T4ALU;
136                         Rout = 1'b1;
137                         Grb = 1'b1;
138                         DPin[`Y] = 1'b1;
139                     end
140
141                 ADDI, ANDI, ORI:
142                     begin
143                         ns = T4ALUimm;
144                         Rout = 1'b1;
145                         Grb = 1'b1;
146                         DPin[`Y] = 1'b1;

```

```

147     end
148
149     NEG, NOT:
150     begin
151         ns = T4nn;
152         Rout = 1'b1;
153         Grb = 1'b1;
154         DPin[`Y] = 1'b1;
155     end
156
157     BR:
158     begin
159         ns = T4br;
160         Rout = 1'b1;
161         Gra = 1'b1;
162         CONin = 1'b1;
163     end
164
165     LD, LDI, ST:
166     begin
167         ns = T4ldst;
168         BAout = 1'b1;
169         Grb = 1'b1;
170         DPin[`Y] = 1'b1;
171     end
172
173     JAL:
174     begin
175         ns = T4jal;
176         Rin = 1'b1;
177         Grb = 1'b1;
178         DPout[`PC] = 1'b1;
179     end
180
181     JR:
182     begin
183         ns = T0;
184         Rout = 1'b1;
185         Gra = 1'b1;
186         DPin[`PC] = 1'b1;
187     end
188
189     HALT:
190         ns = Thalt;
191
192     IN:
193     begin
194         ns = T0;
195         Rin = 1'b1;
196         Gra = 1'b1;
197         DPout[`INPORT] = 1'b1;
198     end
199
200     OUT:
201     begin
202         ns = T0;
203         Rout = 1'b1;
204         Gra = 1'b1;
205         DPin[`OUTPORT] = 1'b1;
206     end
207
208     MFLO:
209     begin
210         ns = T0;
211         Rin = 1'b1;
212         Gra = 1'b1;
213         DPout[`LO] = 1'b1;
214     end
215
216     MFHI:
217     begin
218         ns = T0;
219         Rin = 1'b1;
220         Gra = 1'b1;
221         DPout[`HI] = 1'b1;
222     end

```

```

223
224
225         default: ns = T0;
226     endcase
227
228 // T4-T7 ACCORDING TO DECODE STAGE.
229 // T4 Steps.
230 T4dm:
231 begin
232     ns = T5ALU;
233     Rout = 1'b1;
234     Grb = 1'b1;
235     DPin[`Z] = 1'b1;
236 end
237
238 T4ALU:
239 begin
240     ns = T5ALU;
241     Rout = 1'b1;
242     Grc = 1'b1;
243     DPin[`Z] = 1'b1;
244 end
245
246 T4ALUimm:
247 begin
248     ns = T5ALU;
249     DPout[`C] = 1'b1;
250     DPin[`Z] = 1'b1;
251 end
252
253 T4nn:
254 begin
255     ns = T5ALU;
256     Rout = 1'b1;
257     Grb = 1'b1;
258     DPin[`Z] = 1'b1;
259 end
260
261 T4br: if (CON) begin
262     ns = T5br;
263     DPout[`PC] = 1'b1;
264     DPin[`Y] = 1'b1;
265 end
266     else ns = T0;
267
268 T4ldst:
269 begin
270     DPout[`C] = 1'b1;
271     DPin[`Z] = 1'b1;
272
273     if (IROP[31:27] == ST) ns = T5st;
274     else if (IROP[31:27] == LD) ns = T5ld;
275     else ns = T5ldi;
276 end
277
278 T4jal:
279 begin
280     ns = T0;
281     Rout = 1'b1;
282     Gra = 1'b1;
283     DPin[`PC] = 1'b1;
284 end
285
286 // T5 Steps.
287 T5ALU:
288 begin
289     DPout[`ZLO] = 1'b1;
290     if (IROP[31:27] == MUL || IROP[31:27] == DIV) begin
291         ns = T6dm;
292         DPin[`LO] = 1'b1;
293     end
294     else begin
295         ns = T0;
296         Rin = 1'b1;
297         Gra = 1'b1;
298     end

```

```

299         end
300
301     T5br:
302     begin
303         ns = T6br;
304         DPout[`C] = 1'b1;
305         DPin[`Z] = 1'b1;
306     end
307
308     T5st:
309     begin
310         ns = T6st;
311         DPout[`ZLO] = 1'b1;
312         DPin[`MAR] = 1'b1;
313     end
314
315     T5ld:
316     begin
317         ns = T6ld;
318         DPout[`ZLO] = 1'b1;
319         DPin[`MAR] = 1'b1;
320     end
321
322     T5ldi:
323     begin
324         ns = T0;
325         DPout[`ZLO] = 1'b1;
326         Gra = 1'b1;
327         Rin = 1'b1;
328     end
329
330     T6st:
331     begin
332         ns = T7st;
333         DPin[`MDR] = 1'b1;
334         Gra = 1'b1;
335         Rout = 1'b1;
336     end
337
338     T6ld:
339     begin
340         ns = T7ld;
341         DPin[`MDR] = 1'b1;
342         DPin[`READ] = 1'b1;
343     end
344
345     T6br:
346     begin
347         ns = T0;
348         DPout[`ZLO] = 1'b1;
349         DPin[`PC] = 1'b1;
350     end
351
352     T6dm:
353     begin
354         ns = T0;
355         DPout[`ZHI] = 1'b1;
356         DPin[`HI] = 1'b1;
357     end
358
359     T7ld:
360     begin
361         ns = T0;
362         DPout[`MDR] = 1'b1;
363         Gra = 1'b1;
364         Rin = 1'b1;
365     end
366
367     T7st:
368     begin
369         ns = T0;
370         RAM_wr = 1'b1;
371     end
372
373     Treset:
374     begin

```

```

375         clr = 1'b1;
376         ns = T0;
377     end
378     Thalt: ns = Thalt;
379
380     default: ns = Thalt;
381 endcase
382 end
383
384 //-----//
385
386 //----- STATE TRANSITIONS -----//
387
388 always @(posedge clk) begin
389     if (reset) begin
390         ps <= Treset;
391         op <= 4'b0;
392     end
393     else if (stop)
394         ps <= Thalt;
395     else
396         ps <= ns;
397         if (ps == T3) // Assign op only when exiting T3
398             op <= op_to_alu(IRop[31:27]);
399     end
400
401 //-----//
402
403 //----- MODULE FUNCTIONS -----//
404
405 function [3:0] op_to_alu (input [4:0] op_code);
406 begin
407     case (op_code)
408         ADD, ADDI, LD, ST, LDI, BR: op_to_alu = `ADD;
409         SUB: op_to_alu = `SUB;
410         AND, ANDI: op_to_alu = `AND;
411         OR , ORI : op_to_alu = `OR ;
412         ROR: op_to_alu = `ROR;
413         ROL: op_to_alu = `ROL;
414         SHR: op_to_alu = `SRL;
415         SHRA: op_to_alu = `SRA;
416         SHL: op_to_alu = `SLL;
417         DIV: op_to_alu = `DIV;
418         MUL: op_to_alu = `MUL;
419         NEG: op_to_alu = `NEG;
420         NOT: op_to_alu = `NOT;
421         default: op_to_alu = 4'b0;
422     endcase
423 end
424 endfunction
425
426 assign ALUopp = (ps == T0) ? 1'b1 << `INC : 1'b1 << op;
427 assign run = ~(ps == Thalt);
428
429 task zeroes();
430 begin
431     clr = 0; CONin = 0;
432     Gra = 0; Grb = 0; Grc = 0; Rin = 0; Rout = 0; BAout = 0; RAM_wr = 0;
433     DPin = 16'b0; DPout = 16'b0;
434 end
435 endtask
436
437 endmodule
438
439
440
441
442
443
444
445 `timescale 1ns/1ps
446
447 module Control_tb;
448
449     // Inputs to DUT
450     reg reset, stop, clk, CON;

```

```

451     reg [31:27] IR; // Only bits 31:27 are used in decoding
452
453     // Outputs from DUT
454     wire      clr, CONin, RAM_wr;
455     wire      Gra, Grb, Grc, Rin, Rout, BAout;
456     wire [15:0] DPin, DPout, ALUopp;
457     wire      run;
458
459     // Instantiate the Device Under Test (DUT)
460     Control uut (
461         .reset(reset),
462         .stop(stop),
463         .clk(clk),
464         .CON(CON),
465         .IR(IR),
466         .clr(clr),
467         .CONin(CONin),
468         .RAM_wr(RAM_wr),
469         .Gra(Gra),
470         .Grb(Grb),
471         .Grc(Grc),
472         .Rin(Rin),
473         .Rout(Rout),
474         .BAout(BAout),
475         .DPin(DPin),
476         .DPout(DPout),
477         .ALUopp(ALUopp),
478         .run(run)
479     );
480
481     // Clock generation: 10 ns period
482     initial begin
483         clk = 0;
484         forever #5 clk = ~clk;
485     end
486
487     // Task to run simulation for a given number of clock cycles
488     task run_cycles(input integer n);
489         integer i;
490         begin
491             for(i = 0; i < n; i = i + 1)
492                 @(posedge clk);
493         end
494     endtask
495
496     // Task to apply an instruction and wait for the expected number of cycles.
497     // We use the expected cycle count to wait until the FSM should have returned to T0.
498     // Optionally, the branch instruction takes a CON input.
499     task apply_instruction(input [4:0] instr, input integer expected_cycles, input con_val);
500         begin
501             $display("\n[%0t] Applying instruction: %b, CON = %b", $time, instr, con_val);
502             IR = instr;
503             CON = con_val;
504
505             run_cycles(expected_cycles);
506
507             // Check if the FSM has returned to T0 (the instruction fetch state)
508             // (Note: In this design T0 is used as the starting/fetch state.)
509             if(uut.ps != 5'b00000)
510                 $display("WARNING: FSM did not return to T0 after %0d cycles. Current state
= %b", expected_cycles, uut.ps);
511             else
512                 $display("PASSED: FSM returned to T0 as expected after %0d cycles",
expected_cycles);
513         end
514     endtask
515
516     // Main stimulus sequence
517     initial begin
518         $display("Starting Control Module Testbench");
519         // Initialize signals
520         reset = 1; stop = 0; CON = 0; IR = 5'b0;
521         run_cycles(2); // Hold reset for a couple of cycles
522         reset = 0;
523         run_cycles(2); // Allow FSM to complete reset and move into T0
524

```

```

525 //-----
526 // Test 1: ALU Operation (e.g. ADD)
527 // FSM path: T0 -> T1 -> T2 -> T3 -> T4ALU -> T5ALU -> T0 (~6 cycles)
528 // Instruction code for ADD is 5'b00011.
529 apply_instruction(5'b00011, 6, 0);
530
531 //-----
532 // Test 2: ALU Immediate Operation (e.g. ADDI)
533 // FSM path: T0 -> T1 -> T2 -> T3 -> T4ALUimm -> T5ALU -> T0 (~6 cycles)
534 // Instruction code for ADDI is 5'b01100.
535 apply_instruction(5'b01100, 6, 0);
536
537 //-----
538 // Test 3: Negative Operation (NEG)
539 // FSM path: T0 -> T1 -> T2 -> T3 -> T4nn -> T5ALU -> T0 (~6 cycles)
540 // Instruction code for NEG is 5'b10001.
541 apply_instruction(5'b10001, 6, 0);
542
543 //-----
544 // Test 4: Multiply (MUL) Operation
545 // FSM path: T0 -> T1 -> T2 -> T3 -> T4dm -> T5ALU -> T6dm -> T0 (~7 cycles)
546 // Instruction code for MUL is 5'b10000.
547 apply_instruction(5'b10000, 7, 0);
548
549 //-----
550 // Test 5: Division (DIV) Operation
551 // FSM path: Similar to MUL: T0 -> T1 -> T2 -> T3 -> T4dm -> T5ALU -> T6dm -> T0
(~7 cycles)
552 // Instruction code for DIV is 5'b01111.
553 apply_instruction(5'b01111, 7, 0);
554
555 //-----
556 // Test 6: Branch (BR) with false condition
557 // FSM path: When CON is false, T4br goes directly to T0:
558 // T0 -> T1 -> T2 -> T3 -> T4br -> T0 (~5 cycles)
559 // Instruction code for BR is 5'b10011.
560 apply_instruction(5'b10011, 5, 0);
561
562 //-----
563 // Test 7: Branch (BR) with true condition
564 // FSM path: When CON is true, branch takes the full path:
565 // T0 -> T1 -> T2 -> T3 -> T4br -> T5br -> T6br -> T0 (~7 cycles)
566 apply_instruction(5'b10011, 7, 1);
567
568 //-----
569 // Test 8: Memory Load (LD)
570 // FSM path: T0 -> T1 -> T2 -> T3 -> T4ldst -> T5ld -> T6ld -> T7ld -> T0 (~8 cycles)
571 // Instruction code for LD is 5'b00000.
572 apply_instruction(5'b00000, 8, 0);
573
574 //-----
575 // Test 9: Memory Store (ST)
576 // FSM path: T0 -> T1 -> T2 -> T3 -> T4ldst -> T5st -> T6st -> T7st -> T0 (~8 cycles)
577 // Instruction code for ST is 5'b00010.
578 apply_instruction(5'b00010, 8, 0);
579
580 //-----
581 // Test 10: Jump and Link (JAL)
582 // FSM path: T0 -> T1 -> T2 -> T3 -> T4jal -> T0 (~5cycles)
583 // Instruction code for JAL is 5'b10100.
584 apply_instruction(5'b10100, 5, 0);
585
586 //-----
587 // Test 11: Jump Register (JR)
588 // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~4 cycles) because the control goes
directly to T0 in JR.
589 // Instruction code for JR is 5'b10101.
590 apply_instruction(5'b10101, 4, 0);
591
592 //-----
593 // Test 12: IN Instruction
594 // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
595 // Instruction code for IN is 5'b10110.
596 apply_instruction(5'b10110, 4, 0);
597
598 //-----

```



```

599      // Test 13: OUT Instruction
600      // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
601      // Instruction code for OUT is 5'b10111.
602      apply_instruction(5'b10111, 4, 0);
603
604      //-----
605      // Test 14: MFLO Instruction
606      // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
607      // Instruction code for MFLO is 5'b11000.
608      apply_instruction(5'b11000, 4, 0);
609
610      //-----
611      // Test 15: MFHI Instruction
612      // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
613      // Instruction code for MFHI is 5'b11001.
614      apply_instruction(5'b11001, 4, 0);
615
616      //-----
617      // Test 16: HALT Instruction
618      // FSM path: T0 -> T1 -> T2 -> T3 -> Thalt and then remain in Thalt (~5 cycles)
619      // Instruction code for HALT is 5'b11011.
620      apply_instruction(5'b11011, 4, 0);
621
622      // End simulation after a short delay
623      #20;
624      $display("Simulation complete.");
625      $stop;
626  end
627
628      // Optional: Monitor important signals and the FSM state
629      // (Accessing the internal state variable (ps) hierarchically from the DUT)
630  initial begin
631      $monitor("Time=%0t | FSM State=%b | IR=%b | clr=%b | CONin=%b | RAM_wr=%b | Gra=%b
| Grb=%b | Grc=%b | Rin=%b | Rout=%b | BAout=%b",
632              $time, uut.ps, IR, clr, CONin, RAM_wr, Gra, Grb, Grc, Rin, Rout, BAout);
633  end
634 endmodule
635
636

```