

```

1 // TimeScale
2 `timescale 1ns/1ps
3
4 //Module Definition
5 module alu(
6     // Inputs
7     input I_clk,
8     input I_en,
9     input [4:0] I_aluop,
10    input [15:0] I_dataA,
11    input [15:0] I_dataB,
12    input [7:0] I_imm,
13    // Outputs
14    output [15:0] O_dataResult,
15    output reg O_shldBranch
16 );
17
18 //Reg/Wire Declaration
19 reg [17:0] int_result;
20 wire op_lsb;
21 wire [3:0] opcode;
22
23 // Parameter Declaration
24 localparam Add = 0,
25             Sub = 1,
26             OR = 2,
27             AND = 3,
28             XOR = 4,
29             NOT = 5,
30             Load = 8,
31             Cmp = 9,
32             SHL = 10,
33             SHR = 11,
34             JMPA = 12,
35             JMPR = 13;
36
37
38 // Initial Block
39 initial begin
40     int_result <= 0;
41 end
42
43 //Assigning Values
44 assign op_lsb = I_aluop[0];
45 assign opcode = I_aluop[4:1];
46 assign O_dataResult = int_result[15:0];
47
48 //ALU Operations
49 always@(negedge I_clk) begin
50
51     if(I_en) begin
52         case(opcode)
53
54             Add : begin
55                 int_result <= (op_lsb ? ($signed(I_dataA) + $signed(I_dataB)) : (
56                     I_dataA + I_dataB));
57                 O_shldBranch <= 0;
58             end
59
60             Sub : begin
61                 int_result <= (op_lsb ? ($signed(I_dataA) + $signed(I_dataB)) : (
62                     I_dataA + I_dataB));
63                 O_shldBranch <= 0;
64             end
65
66             OR :begin
67                 int_result <= I_dataA | I_dataB;
68                 O_shldBranch <= 0;
69             end
70
71             AND :begin
72                 int_result <= I_dataA & I_dataB;
73                 O_shldBranch <= 0;
74             end
75
76             XOR :begin
77                 int_result <= I_dataA ^ I_dataB;
78                 O_shldBranch <= 0;
79             end
80
81             NOT :begin
82                 int_result <= ~I_dataA;
83                 O_shldBranch <= 0;
84             end
85
86             Load :begin
87                 int_result <= I_imm;
88                 O_shldBranch <= 0;
89             end
90
91             Cmp :begin
92                 int_result <= I_dataA - I_dataB;
93                 O_shldBranch <= 0;
94             end
95
96             SHL :begin
97                 int_result <= I_dataA << I_imm;
98                 O_shldBranch <= 0;
99             end
100            SHR :begin
101                int_result <= I_dataA >> I_imm;
102                O_shldBranch <= 0;
103            end
104            JMPA :begin
105                int_result <= I_dataA;
106                O_shldBranch <= 0;
107            end
108            JMPR :begin
109                int_result <= I_dataB;
110                O_shldBranch <= 0;
111            end
112
113            default :begin
114                int_result <= 0;
115                O_shldBranch <= 0;
116            end
117        endcase
118    end
119 end

```

```

68
69     AND :begin
70         int_result <= I_dataA & I_dataB;
71         O_shldBranch <= 0;
72     end
73
74     XOR :begin
75         int_result <= I_dataA ^ I_dataB;
76         O_shldBranch <= 0;
77     end
78
79     NOT :begin
80         int_result <= ~I_dataA;
81         O_shldBranch <= 0;
82     end
83
84     Load : begin
85         int_result <= (op_lsb ? ({I_imm, 8'h00}) : ({8'h00, I_imm}));
86         O_shldBranch <= 0;
87     end
88
89     Cmp : begin
90         if(op_lsb) begin
91             int_result[0] <= ($signed(I_dataA) == $signed(I_dataB)) ? 1 :
92                 0;
93             int_result[1] <= ($signed(I_dataA) == 0) ? 1 : 0;
94             int_result[2] <= ($signed(I_dataB) == 0) ? 1 : 0;
95             int_result[3] <= ($signed(I_dataA) > $signed(I_dataB)) ? 1 :
96                 0;
97             int_result[4] <= ($signed(I_dataA) < $signed(I_dataB)) ? 1 :
98                 0;
99             end
100         else begin
101             int_result[0] <= (I_dataA == I_dataB) ? 1 : 0;
102             int_result[1] <= (I_dataA == 0) ? 1 : 0;
103             int_result[2] <= (I_dataB == 0) ? 1 : 0;
104             int_result[3] <= (I_dataA > I_dataB) ? 1 : 0;
105             int_result[4] <= (I_dataA < I_dataB) ? 1 : 0;
106         end
107         O_shldBranch <= 0;
108     end
109
110     SHL : begin
111         int_result <= I_dataA << (I_dataB[3:0]);
112         O_shldBranch <= 0;
113     end
114
115     SHR : begin
116         int_result <= I_dataA >> (I_dataB[3:0]);
117         O_shldBranch <= 0;
118     end
119
120     JMPA : begin
121         int_result <= (op_lsb ? I_dataA : I_imm);
122         O_shldBranch <= 1;
123     end
124
125     JMPR : begin
126         int_result <= I_dataA;
127         O_shldBranch <= I_dataB[{op_lsb , I_imm[1:0]}];
128     end
129 endcase
end
endmodule

```

```

1  //TimeScale
2  `timescale 1ns / 1ps
3
4  //Module Definition
5  module ctrl_unit(
6      // Inputs
7      input I_clk,
8      input I_reset,
9      // Outputs
10     output O_enfetch,
11     output O_endec,
12     output O_enrgrd,
13     output O_enalu,
14     output O_enrgwr,
15     output O_enmem
16 );
17     // Reg Declaration
18     reg [5:0] state;
19
20     // Initial Block
21     initial begin
22         state <= 6'b000001;
23     end
24
25     //State Select Block
26     always@(posedge I_clk) begin
27         if(I_reset)
28             state <= 6'b000001;
29         else begin
30             case(state)
31                 6'b000001 : state <= 6'b000010;
32                 6'b000010 : state <= 6'b000100;
33                 6'b000100 : state <= 6'b001000;
34                 6'b001000 : state <= 6'b010000;
35                 6'b010000 : state <= 6'b100000;
36                 default : state <= 6'b000001;
37             endcase
38         end
39     end
40
41     //Assignment Enable Signals
42     assign O_enfetch = state[0];
43     assign O_endec = state[1];
44     assign O_enrgrd = state[2] | state[4];
45     assign O_enalu = state[3];
46     assign O_enrgwr = state[4];
47     assign O_enmem = state[5];
48
49 endmodule
50

```

```

1  //Timescale
2  `timescale 1ns / 1ps
3  //Module Definition
4  module fake_ram(
5      //Inputs
6      input I_clk,
7      input I_we,
8      input [15:0] I_addr,
9      input [15:0] I_data,
10     //outputs
11     output reg [15:0] O_data
12 );
13     //Memory declaration
14     reg [15:0] mem [8:0];
15
16     //Initialize registers
17     initial begin
18
19         mem[0] = 16'b1000000011111110;
20         mem[1] = 16'b1000100111101101;
21         mem[2] = 16'b0010001000100000;
22         mem[3] = 16'b1000001100000001;
23         mem[4] = 16'b1000010000000001;
24         mem[5] = 16'b0000001101110000;
25         mem[6] = 16'b1100000000000101;
26         mem[7] = 0;
27         mem[8] = 0;
28
29         O_data = 16'b0000000000000000;
30     end
31
32     //Ram operation
33     always@(negedge I_clk)begin
34         if(I_we) begin
35             mem[I_addr[15:0]] <= I_data;
36         end
37         O_data <= mem[I_addr[15:0]];
38     end
39 endmodule

```

```

1  //TimeScale
2  `timescale 1ns / 1ps
3
4  //Module Definition
5  module inst_dec(
6      // Inputs
7      input I_clk,
8      input I_en,
9      input [15:0] I_inst,
10     // Outputs
11     output reg [4:0] O_aluop,
12     output reg [2:0] O_selA,
13     output reg [2:0] O_selB,
14     output reg [2:0] O_selD,
15     output reg [15:0] O_imm,
16     output reg O_regwe
17 );
18     // Initial Block
19     initial begin
20         O_aluop <= 0;
21         O_selA <= 0;
22         O_selB <= 0;
23         O_selD <= 0;
24         O_imm <= 0;
25         O_regwe <= 0;
26     end
27     always@(negedge I_clk) begin
28
29         if(I_en) begin
30             O_aluop <= I_inst[15:11];
31             O_selA <= I_inst[10:8]; //REG A
32             O_selB <= I_inst[7:5]; //REG B
33             O_selD <= I_inst[4:2]; //REG D
34             O_imm <= I_inst[7:0]; //Imm Data
35
36             //REG Write Enable
37             case(I_inst[15:12])
38                 4'b0111: O_regwe <= 0;
39                 4'b1100: O_regwe <= 0;
40                 4'b1101: O_regwe <= 0;
41                 default: O_regwe <= 1;
42             endcase
43         end
44     end
45
46 endmodule

```

```

1  // TimeScale
2  `timescale 1ns / 1ps
3
4  //Module Definition
5  module pc_unit(
6      // Inputs
7      input I_clk,
8      input [1:0] I_opcode,
9      input [15:0] I_pc,
10     // Outputs
11     output reg [15:0] O_pc
12 );
13
14     // Initial Block
15     initial begin
16         O_pc <= 0;
17     end
18
19     // Program Counter State
20     always@(negedge I_clk) begin
21         case(I_opcode)
22             2'b00 : O_pc <= O_pc;
23             2'b01 : O_pc <= O_pc + 1;
24             2'b10 : O_pc <= I_pc;
25             2'b11 : O_pc <= 0;
26         endcase
27     end
28
29 endmodule

```

```

1  // TimeScale
2  `timescale 1ns / 1ps
3
4  //Module Definition
5  module reg_file(
6      // Inputs
7      input I_clk,
8      input I_en,
9      input I_we,
10     input [2:0] I_selA,
11     input [2:0] I_selB,
12     input [2:0] I_selD,
13     input [15:0] I_dataD,
14     // Outputs
15     output reg [15:0] O_dataA,
16     output reg [15:0] O_dataB
17 );
18
19     // Internal register declaration
20     reg [15:0] regs [7:0];
21
22     // Loop Variable
23     integer count;
24
25     // Initialize register
26     initial begin
27         O_dataA = 0;
28         O_dataB = 0;
29
30         for(count = 0; count < 8; count = count + 1) begin
31             regs[count] = 0;
32         end
33     end
34
35     // Assigning correct values to Op regs
36     always@(negedge I_clk) begin
37         if(I_en) begin
38             if(I_we)
39                 regs[I_selD] <= I_dataD;
40
41             O_dataA <= regs[I_selA];
42             O_dataB <= regs[I_selB];
43         end
44     end
45
46 endmodule

```

```

1  //Timescale
2  `timescale 1ns / 1ps
3
4  //Module Definition
5  module decoder_unittests();
6      //Variable Declaration
7      //Regs
8      reg I_Clk;
9      reg I_En;
10     reg [15:0] I_Inst;
11     //Wires
12     wire [4:0] O_Aluop;
13     wire [2:0] O_SelA;
14     wire [2:0] O_SelB;
15     wire [2:0] O_SelD;
16     wire [15:0] O_Imm;
17     wire O_Regwe;
18
19     inst_dec inst_unit(
20         // Inputs
21         I_Clk,
22         I_En,
23         I_Inst,
24         // Outputs
25         O_Aluop,
26         O_SelA,
27         O_SelB,
28         O_SelD,
29         O_Imm,
30         O_Regwe
31     );
32
33     initial begin
34         //Time = 0
35         I_Clk <= 0;
36         I_En <= 0;
37         I_Inst <= 0;
38         //Time = 10
39         #10;
40         I_Inst = 16'b0001011100000100;
41         //Time = 20
42         #10;
43         I_En = 1;
44     end
45
46     always begin
47         #5;
48         I_Clk = ~I_Clk;
49     end
50 endmodule

```



```

1 // Timescale
2 `timescale 1ns / 1ps
3
4 // Module Definition
5 module main_test();
6 // variable declaration
7 //Regs
8 reg clk;
9 reg reset;
10 reg reg_we = 0;
11 reg [15:0] dataI = 0;
12 //wire
13 wire [2:0] selA;
14 wire [2:0] selB;
15 wire [2:0] selD;
16 wire [15:0] dataA;
17 wire [15:0] dataB;
18 wire [15:0] dataD;
19 wire [4:0] aluop;
20 wire [7:0] imm;
21 wire [15:0] dataO;
22 wire [1:0] opcode;
23 wire [15:0] pcO;
24
25 wire shldBranch;
26 wire enfetch;
27 wire enalu;
28 wire endec;
29 wire enmem;
30 wire enrgrd;
31 wire enrgwr;
32 wire regwe;
33 wire update;
34
35 //Assignments;
36 assign enrgwr = regwe & update;
37 assign opcode = (reset) ? 2'b11 : ((shldBranch) ? 2'b10 : ((enmem) ? 2'b01 : 2'b00));
38
39 //instatiation
40 reg_file main_reg(
41 // Inputs
42 clk,
43 enrgrd,
44 enrgwr,
45 selA,
46 selB,
47 selD,
48 dataD,
49 // Outputs
50 dataA,
51 dataB
52 );
53
54 inst_dec main_inst(
55 // Inputs
56 clk,
57 endec,
58 dataO,
59 // outputs
60 aluop,
61 selA,
62 selB,
63 selD,
64 imm,
65 regwe
66 );
67
68 alu main_alu(
69 // Inputs

```

```

70     clk,
71     enalu,
72     aluop,
73     dataA,
74     dataB,
75     imm,
76     // Outputs
77     dataD,
78     shldBranch
79 );
80
81 ctrl_unit main_ctrl(
82     // Inputs
83     clk,
84     reset,
85     // Outputs
86     enfetch,
87     endec,
88     enrgrd,
89     enalu,
90     update,
91     enmem
92 );
93
94 pc_unit pc_main(
95     // Inputs
96     clk,
97     opcode,
98     dataD,
99     // Outputs
100    pcO
101 );
102
103 fake_ram main_ram (
104     // Inputs
105     clk,
106     reg_we,
107     pcO,
108     dataI,
109     // Outputs
110     dataO
111 );
112
113 initial begin
114
115     clk = 0;
116     reset = 1;
117     #20
118     reset = 0;
119
120 end
121
122 //Clock generation
123 always begin
124     #5;
125     clk = ~clk;
126 end
127
128
129 endmodule
130
131
132
133
134

```

```

1 //Timescale
2 `timescale 1ns / 1ps
3
4 //Module Definition
5 module regfile_unittest();
6     //Variable Declaration
7     //Regs
8     reg I_clk;
9     reg [15:0] I_dataD;
10    reg I_en;
11    reg [2:0] I_selA;
12    reg [2:0] I_selB;
13    reg [2:0] I_selD;
14    reg I_we;
15    //Wires
16    wire O_dataA;
17    wire O_dataB;
18
19    reg_file reg_test(
20        //Inputs
21        I_clk,
22        I_en,
23        I_we,
24        I_selA,
25        I_selB,
26        I_selD,
27        I_dataD,
28        //Outputs
29        O_dataA,
30        O_dataB
31    );
32
33    initial begin
34        //Reset all Inputs
35        I_clk = 1'b0;
36        I_dataD = 0;
37        I_en = 0;
38        I_selA = 0;
39        I_selB = 0;
40        I_selD = 0;
41        I_we = 0;
42
43        //Start Test
44        //Time = 7
45        #7
46        I_en = 1'b1;
47
48        I_selA = 3'b000;
49        I_selB = 3'b001;
50        I_selD = 3'b000;
51
52        I_dataD = 16'hFFFF;
53        I_we = 1'b1;
54
55        //Time = 17
56        I_we = 1'b0;
57        I_selD = 3'b010;
58        I_dataD = 16'h2222;
59
60        //Time = 27
61        #10;
62        I_we = 1;
63
64        //Time = 37
65        #10;
66        I_dataD = 16'h3333;
67
68        //Time = 47
69        #10;

```

```
70     I_we = 0;
71     I_selD = 3'b000;
72     I_dataD = 16'hFEED;
73
74     //Time = 57
75     #10;
76     I_selD = 3'b100;
77     I_dataD = 16'h4444;
78
79     //Time = 67
80     #10;
81     I_we = 1;
82
83     //Time = 117
84     #50;
85     I_selA = 3'b100;
86     I_selB = 3'b100;
87     #20;
88     $finish;
89
90 end
91
92 //Clock generation
93 always begin
94     #5;
95     I_clk = ~I_clk;
96 end
97
98 endmodule
```

