Final Verilog Code: All Modules, Top Module, and Testbench

1 All Verilog Modules

```
// Program Counter
module Program_Counter(clk, reset, PC_in, PC_out);
    input clk, reset;
    input [31:0] PC_in;
    output reg [31:0] PC_out;

    always @(posedge clk or posedge reset)
    begin
        if(reset)
            PC_out <= 32'b0;
        else
            PC_out <= PC_in;
    end
endmodule

// PC Plus 4
module PCplus4(fromPC, NexttoPC);
    input [31:0] fromPC;
    output [31:0] NexttoPC;

    assign NexttoPC = 4 + fromPC;
endmodule

// Instruction Memory
module Instruction_Mem(clk, reset, read_address,
   instruction_out);
    input clk, reset;
    input [31:0] read_address;
    output reg [31:0] instruction_out;
```

```
29      integer k;
30      reg [31:0] I_Mem[63:0];
31
32      always @(posedge clk or posedge reset)
33      begin
34          if(reset)
35          begin
36              for(k=0; k<64; k=k+1) begin
37                  I_Mem[k] <= 32'b0;
38              end
39          end
40          else
41              instruction_out <= I_Mem[read_address];
42      end
43  endmodule
44
45  // Register File
46  module Reg_File(clk, reset, RegWrite, Rs1, Rs2, Rd,
        Write_data, read_data1, read_data2);
47      input clk, reset, RegWrite;
48      input [4:0] Rs1, Rs2, Rd;
49      input [31:0] Write_data;
50      output [31:0] read_data1, read_data2;
51      integer k;
52      reg [31:0] Registers[31:0];
53
54      always @(posedge clk or posedge reset)
55      begin
56          if(reset)
57          begin
58              for(k=0; k<32; k=k+1) begin
59                  Registers[k] <= 32'b0;
60              end
61          end
62          else if(RegWrite) begin
63              Registers[Rd] <= Write_data;
64          end
65      end
66
67      assign read_data1 = Registers[Rs1];
68      assign read_data2 = Registers[Rs2];
69  endmodule
70
```

```
71 // Immediate Generator
72 module ImmGen(Opcode, instruction, ImmExt);
73     input [6:0] Opcode;
74     input [31:0] instruction;
75     output [31:0] ImmExt;
76 
77     always @(*)
78     begin
79         case(Opcode)
80             7'b0000011 : ImmExt = {{20{instruction[31]}},
                   instruction[31:20]};
81             7'b0100011 : ImmExt = {{20{instruction[31]}},
                   instruction[31:25], instruction[11:7]};
82             7'b1100011 : ImmExt = {{19{instruction[31]}},
                   instruction[31], instruction[30:25],
                   instruction[11:8], 1'b0};
83             default : ImmExt = 32'b0;
84         endcase
85     end
86 endmodule
87 
88 // Control Unit
89 module Control_Unit(instruction, Branch, MemRead,
    MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite);
90     input [6:0] instruction;
91     output Branch, MemRead, MemtoReg, MemWrite, ALUSrc,
       RegWrite;
92     output [1:0] ALUOp;
93 
94     always @(*)
95     begin
96         case(instruction)
97             7'b0110011 : {ALUSrc, MemtoReg, RegWrite,
                   MemRead, MemWrite, Branch, ALUOp} <=
                   8'b000000_01;
98             7'b0000011 : {ALUSrc, MemtoReg, RegWrite,
                   MemRead, MemWrite, Branch, ALUOp} <=
                   8'b010000_00;
99             7'b0100011 : {ALUSrc, MemtoReg, RegWrite,
                   MemRead, MemWrite, Branch, ALUOp} <=
                   8'b010000_00;
100            7'b1100011 : {ALUSrc, MemtoReg, RegWrite,
                   MemRead, MemWrite, Branch, ALUOp} <=
```

```
                8'b001000_01;
            default : {ALUSrc, MemtoReg, RegWrite,
                MemRead, MemWrite, Branch, ALUOp} <=
                8'b000000_00;
        endcase
    end
endmodule

// ALU
module ALU_unit(A, B, Control_in, ALU_Result, zero);
    input [31:0] A, B;
    input [3:0] Control_in;
    output reg zero;
    output reg [31:0] ALU_Result;

    always @(Control_in, or A or B)
    begin
        case(Control_in)
            4'b0000: begin zero <= 0; ALU_Result <= A & B;
                end
            4'b0001: begin zero <= 0; ALU_Result <= A | B;
                end
            4'b0010: begin zero <= 0; ALU_Result <= A + B;
                end
            4'b0110: begin if(A==B) zero <= 1; else zero
                <= 0; ALU_Result <= A - B; end
            default: begin zero <= 0; ALU_Result <= 0; end
        endcase
    end
endmodule

// ALU Control
module ALU_Control(ALUOp, fun7, fun3, Control_out);
    input fun7;
    input [2:0] fun3;
    input [1:0] ALUOp;
    output reg [3:0] Control_out;

    always @(*)
    begin
        case({ALUOp, fun7, fun3})
            6'b00_0_000: Control_out <= 4'b0010;
            6'b01_0_000: Control_out <= 4'b0110;
```

```
137             6'b01_0_110: Control_out <= 4'b0000;
138             6'b01_0_111: Control_out <= 4'b0001;
139             6'b10_0_000: Control_out <= 4'b0010;
140             6'b10_1_000: Control_out <= 4'b0110;
141             default: Control_out <= 4'b0000;
142         endcase
143     end
144 endmodule
145 
146 // Data Memory
147 module Data_Memory(clk, reset, MemWrite, MemRead,
    read_address, Write_data, MemData_out);
148     input clk, reset, MemWrite, MemRead;
149     input [31:0] read_address, Write_data;
150     output [31:0] MemData_out;
151     integer k;
152     reg [31:0] D_Memory[63:0];
153 
154     always @(posedge clk or posedge reset)
155     begin
156         if(reset)
157         begin
158             for(k=0; k<64; k=k+1) begin
159                 D_Memory[k] <= 32'b0;
160             end
161         end
162         else if(MemWrite) begin
163             D_Memory[read_address] <= Write_data;
164         end
165     end
166 
167     assign MemData_out = (MemRead) ? D_Mem[read_address] :
        32'b0;
168 endmodule
169 
170 // Mux 2
171 module Mux2(sel2, A2, B2, Mux2_out);
172     input sel2;
173     input [31:0] A2, B2;
174     output [31:0] Mux2_out;
175 
176     assign Mux2_out = (sel2==1'b0) ? A2 : B2;
177 endmodule
```

```
178
179 // Mux 3
180 module Mux3(sel3, A3, B3, Mux3_out);
181     input sel3;
182     input [31:0] A3, B3;
183     output [31:0] Mux3_out;
184
185     assign Mux3_out = (sel3==1'b0) ? A3 : B3;
186 endmodule
187
188 // Adder
189 module Adder(in_1, in_2, Sum_out);
190     input [31:0] in_1, in_2;
191     output [31:0] Sum_out;
192
193     assign Sum_out = in_1 + in_2;
194 endmodule
195
196 // AND Gate
197 module AND_logic(branch, zero, and_out);
198     input branch, zero;
199     output and_out;
200
201     assign and_out = branch & zero;
202 endmodule
203
204 \section{Top Module}
205
206 \begin{lstlisting}
207 // Top-Level Module
208 module top(clk, reset);
209     input clk, reset;
210
211     // Wires
212     wire [31:0] PC_top, instruction_top, Rd1_top, Rd2_top,
        ImmExt_top, mux1_top, Sum_out_top, NexttoPC_top,
        PCin_top, address_top, Memdata_top, WriteBack_top;
213     wire RegWrite_top, ALUSrc_top, zero_top, branch_top,
        sel2_top, MemtoReg_top, MemWrite_top, MemRead_top;
214     wire [1:0] ALUOp_top;
215     wire [3:0] control_top;
216
217     // Program Counter
```

```
218     Program_Counter PC (.clk(clk), .reset(reset),
           .PC_in(PCin_top), .PC_out(PC_top));
219
220     // PC Adder
221     PCplus4 PC_Adder (.fromPC(PC_top),
           .NexttoPC(NexttoPC_top));
222
223     // Instruction Memory
224     Instruction_Mem Inst_Memory (.clk(clk), .reset(reset),
           .read_address(PC_top),
           .instruction_out(instruction_top));
225
226     // Register File
227     Reg_File Reg_File (.clk(clk), .reset(reset),
           .RegWrite(RegWrite_top),
           .Rs1(instruction_top[19:15]),
           .Rs2(instruction_top[24:20]),
           .Rd(instruction_top[11:7]),
           .Write_data(WriteBack_top), .read_data1(Rd1_top),
           .read_data2(Rd2_top));
228
229     // Immediate Generator
230     ImmGen ImmGen (.Opcode(instruction_top[6:0]),
           .instruction(instruction_top), .ImmExt(ImmExt_top));
231
232     // Control Unit
233     Control_Unit Control_Unit
           (.instruction(instruction_top[6:0]),
           .Branch(branch_top), .MemRead(MemRead_top),
           .MemtoReg(MemtoReg_top), .ALUOp(ALUOp_top),
           .MemWrite(MemWrite_top), .ALUSrc(ALUSrc_top),
           .RegWrite(RegWrite_top));
234
235     // ALU Control
236     ALU_Control ALU_Control (.ALUOp(ALUOp_top),
           .fun7(instruction_top[30]),
           .fun3(instruction_top[14:12]),
           .Control_out(control_top));
237
238     // ALU
239     ALU_unit ALU (.A(Rd1_top), .B(mux1_top),
           .Control_in(control_top), .ALU_Result(address_top),
           .zero(zero_top));
```

```
240
241     // ALU Mux
242     Mux3 ALU_mux (.sel3(ALUSrc_top), .A3(Rd2_top),
            .B3(ImmExt_top), .Mux3_out(mux1_top));
243
244     // Adder
245     Adder Adder (.in_1(PC_top), .in_2(ImmExt_top),
            .Sum_out(Sum_out_top));
246
247     // AND Gate
248     AND_logic AND (.branch(branch_top), .zero(zero_top),
            .and_out(sel2_top));
249
250     // Mux
251     Mux2 Adder_mux (.sel2(sel2_top), .A2(NexttoPC_top),
            .B2(Sum_out_top), .Mux2_out(PCin_top));
252
253     // Data Memory
254     Data_Memory Data_mem (.clk(clk), .reset(reset),
            .MemWrite(MemWrite_top), .MemRead(MemRead_top),
            .read_address(address_top), .Write_data(Rd2_top),
            .MemData_out(Memdata_top));
255
256     // Mux
257     Mux3 Memory_mux (.sel3(MemtoReg_top),
            .A3(address_top), .B3(Memdata_top),
            .Mux3_out(WriteBack_top));
258
259 endmodule
```

2 Testbench

```
1 `timescale 1ns / 1ps
2
3 module tb_top;
4     // Inputs
5     reg clk;
6     reg reset;
7
8     // Instantiate the top module with name 'uut'
9     top uut (
```

```
        .clk(clk),
        .reset(reset)
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns clock period
    end

    // Test stimulus
    initial begin
        // Initialize
        reset = 1;
        #20 reset = 0; // Release reset after 20ns

        // Test case 1: Basic instruction fetch and PC
           increment
        #50;
        $display("Time=%0t PC_top=%h instruction_top=%h",
           $time, uut.PC_top, uut.instruction_top);

        // Test case 2: Register write and ALU operation
           (e.g., ADD instruction)
        #50;
        $display("Time=%0t Rd1_top=%h Rd2_top=%h
           address_top=%h", $time, uut.Rd1_top,
           uut.Rd2_top, uut.address_top);

        // Test case 3: Memory read/write
        #50;
        $display("Time=%0t Memdata_top=%h
           WriteBack_top=%h", $time, uut.Memdata_top,
           uut.WriteBack_top);

        // Test case 4: Branch condition
        #50;
        $display("Time=%0t PCin_top=%h Sum_out_top=%h",
           $time, uut.PCin_top, uut.Sum_out_top);

        // End simulation
        #100 $finish;
    end
```

```
45
46      // Monitor signals
47      initial begin
48          $monitor("Time=%0t reset=%b PC_top=%h
                instruction_top=%h RegWrite_top=%b", $time,
                reset, uut.PC_top, uut.instruction_top,
                uut.RegWrite_top);
49      end
50
51  endmodule
```