// 16-bit RISC Processor Modules and Testbenches

**1. Instruction Memory**

module InstructionMemory (

input [15:0] address,

output reg [15:0] instruction

);

reg [15:0] mem [0:255]; // 256 x 16-bit memory

initial begin

// Load some sample instructions

mem[0] = 16'h0001; // Example instruction

mem[1] = 16'h0002;

mem[2] = 16'h0003;

end

always @(\*) begin

instruction = mem[address];

end

endmodule

// Testbench for Instruction Memory

module InstructionMemory\_tb;

reg [15:0] address;

wire [15:0] instruction;

InstructionMemory uut (

.address(address),

.instruction(instruction)

);

initial begin

address = 0;

#10;

$display("Instruction at address 0: %h", instruction);

address = 1;

#10;

$display("Instruction at address 1: %h", instruction);

$finish;

end

endmodule

**2. Register File**

module RegisterFile (

input clk,

input [3:0] read\_reg1, read\_reg2, write\_reg,

input [15:0] write\_data,

input reg\_write,

output [15:0] read\_data1, read\_data2

);

reg [15:0] registers [0:15]; // 16 x 16-bit registers

always @(posedge clk) begin

if (reg\_write) begin

registers[write\_reg] <= write\_data;

end

end

assign read\_data1 = registers[read\_reg1];

assign read\_data2 = registers[read\_reg2];

endmodule

// Testbench for Register File

module RegisterFile\_tb;

reg clk;

reg [3:0] read\_reg1, read\_reg2, write\_reg;

reg [15:0] write\_data;

reg reg\_write;

wire [15:0] read\_data1, read\_data2;

RegisterFile uut (

.clk(clk),

.read\_reg1(read\_reg1),

.read\_reg2(read\_reg2),

.write\_reg(write\_reg),

.write\_data(write\_data),

.reg\_write(reg\_write),

.read\_data1(read\_data1),

.read\_data2(read\_data2)

);

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

reg\_write = 1;

write\_reg = 4'b0001;

write\_data = 16'h1234;

#10;

reg\_write = 0;

read\_reg1 = 4'b0001;

#10;

$display("Read Data 1: %h", read\_data1);

$finish;

end

endmodule

**3. Data Memory**

module DataMemory (

input clk,

input [15:0] address,

input [15:0] write\_data,

input mem\_write, mem\_read,

output [15:0] read\_data

);

reg [15:0] mem [0:255]; // 256 x 16-bit memory

always @(posedge clk) begin

if (mem\_write) begin

mem[address] <= write\_data;

end

end

assign read\_data = (mem\_read) ? mem[address] : 16'h0000;

endmodule

// Testbench for Data Memory

module DataMemory\_tb;

reg clk;

reg [15:0] address, write\_data;

reg mem\_write, mem\_read;

wire [15:0] read\_data;

DataMemory uut (

.clk(clk),

.address(address),

.write\_data(write\_data),

.mem\_write(mem\_write),

.mem\_read(mem\_read),

.read\_data(read\_data)

);

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

mem\_write = 1;

address = 16'h0001;

write\_data = 16'h5678;

#10;

mem\_write = 0;

mem\_read = 1;

#10;

$display("Read Data: %h", read\_data);

$finish;

end

endmodule

**4. ALU**

module ALU (

input [15:0] src1, src2,

input [2:0] alu\_control,

output reg [15:0] result,

output zero

);

always @(\*) begin

case (alu\_control)

3'b000: result = src1 + src2; // ADD

3'b001: result = src1 - src2; // SUB

3'b010: result = src1 & src2; // AND

3'b011: result = src1 | src2; // OR

3'b100: result = ~(src1 | src2); // NOR

default: result = 16'h0000;

endcase

end

assign zero = (result == 16'h0000);

endmodule

// Testbench for ALU

module ALU\_tb;

reg [15:0] src1, src2;

reg [2:0] alu\_control;

wire [15:0] result;

wire zero;

ALU uut (

.src1(src1),

.src2(src2),

.alu\_control(alu\_control),

.result(result),

.zero(zero)

);

initial begin

src1 = 16'h0001;

src2 = 16'h0002;

alu\_control = 3'b000; // ADD

#10;

$display("Result: %h, Zero: %b", result, zero);

$finish;

end

endmodule

**5. ALU Control Unit**

module ALUControl (

input [2:0] funct,

input [1:0] alu\_op,

output reg [2:0] alu\_control

);

always @(\*) begin

case (alu\_op)

2'b00: alu\_control = 3'b000; // ADD

2'b01: alu\_control = 3'b001; // SUB

2'b10: case (funct)

3'b000: alu\_control = 3'b000; // ADD

3'b001: alu\_control = 3'b001; // SUB

3'b010: alu\_control = 3'b010; // AND

3'b011: alu\_control = 3'b011; // OR

default: alu\_control = 3'b000;

endcase

default: alu\_control = 3'b000;

endcase

end

endmodule

// Testbench for ALU Control

module ALUControl\_tb;

reg [2:0] funct;

reg [1:0] alu\_op;

wire [2:0] alu\_control;

ALUControl uut (

.funct(funct),

.alu\_op(alu\_op),

.alu\_control(alu\_control)

);

initial begin

funct = 3'b000;

alu\_op = 2'b10;

#10;

$display("ALU Control: %b", alu\_control);

$finish;

end

endmodule

**6. Datapath**

module datapath (

input clk,

input reset,

input [15:0] in\_data, // Data input (e.g., from memory)

input [15:0] ALU\_result, // Result from ALU

input [15:0] reg\_data, // Data to be written into registers

input [3:0] reg\_addr, // Register address for read/write

input reg\_write, // Control signal for writing to register

output reg [15:0] reg\_out // Output from the register

);

// 16-bit Register File (for simplicity, only one register here)

reg [15:0] reg\_file [15:0]; // 16 registers of 16-bit width

// On reset, initialize all registers to 0

always @(posedge clk or posedge reset) begin

if (reset) begin

reg\_file[reg\_addr] <= 16'b0; // Reset register value

reg\_out <= 16'b0; // Reset output

end

else if (reg\_write) begin

reg\_file[reg\_addr] <= reg\_data; // Write data to register

end

else begin

reg\_out <= reg\_file[reg\_addr]; // Output value from register

end

end

endmodule

module datapath\_tb;

reg clk;

reg reset;

reg [15:0] in\_data;

reg [15:0] ALU\_result;

reg [15:0] reg\_data;

reg [3:0] reg\_addr;

reg reg\_write;

wire [15:0] reg\_out;

// Instantiate the datapath module

datapath uut (

.clk(clk),

.reset(reset),

.in\_data(in\_data),

.ALU\_result(ALU\_result),

.reg\_data(reg\_data),

.reg\_addr(reg\_addr),

.reg\_write(reg\_write),

.reg\_out(reg\_out)

);

// Clock generation

always begin

#5 clk = ~clk; // Toggle clock every 5ns

end

// Test procedure

initial begin

// Initial setup

clk = 0;

reset = 0;

reg\_data = 16'hABCD; // Data to be written to the register

reg\_addr = 4'b0001; // Select register 1

reg\_write = 0;

// Apply reset

reset = 1;

#10 reset = 0;

// Write data to register (after reset)

reg\_write = 1;

#10 reg\_write = 0;

// Check the output from register

#10 if (reg\_out == 16'hABCD) begin

$display("Test Passed: Register output is %h", reg\_out);

end else begin

$display("Test Failed: Register output is %h", reg\_out);

end

// Apply new data and write to another register

reg\_data = 16'h1234;

reg\_addr = 4'b0010; // Select register 2

reg\_write = 1;

#10 reg\_write = 0;

// Check new register output

#10 if (reg\_out == 16'h1234) begin

$display("Test Passed: Register output is %h", reg\_out);

end else begin

$display("Test Failed: Register output is %h", reg\_out);

end

$finish;

end

endmodule

**7. Control Unit Module**

module ControlUnit (

input [15:0] instruction,

output reg reg\_write, mem\_write, mem\_read, alu\_src, reg\_dst

);

always @(\*) begin

case (instruction[15:12])

4'b0000: begin // ADD

reg\_write = 1;

mem\_write = 0;

mem\_read = 0;

alu\_src = 0;

reg\_dst = 1;

end

4'b0001: begin // SUB

reg\_write = 1;

mem\_write = 0;

mem\_read = 0;

alu\_src = 0;

reg\_dst = 1;

end

4'b0010: begin // LW (Load Word)

reg\_write = 1;

mem\_write = 0;

mem\_read = 1;

alu\_src = 1;

reg\_dst = 0;

end

4'b0011: begin // SW (Store Word)

reg\_write = 0;

mem\_write = 1;

mem\_read = 0;

alu\_src = 1;

reg\_dst = 0;

end

default: begin // Default case

reg\_write = 0;

mem\_write = 0;

mem\_read = 0;

alu\_src = 0;

reg\_dst = 0;

end

endcase

end

endmodule

module ControlUnit\_tb;

reg [15:0] instruction;

wire reg\_write, mem\_write, mem\_read, alu\_src, reg\_dst;

ControlUnit uut (

.instruction(instruction),

.reg\_write(reg\_write),

.mem\_write(mem\_write),

.mem\_read(mem\_read),

.alu\_src(alu\_src),

.reg\_dst(reg\_dst)

);

initial begin

// Test ADD instruction

instruction = 16'h0000;

#10;

$display("ADD: reg\_write=%b, mem\_write=%b, mem\_read=%b, alu\_src=%b, reg\_dst=%b", reg\_write, mem\_write, mem\_read, alu\_src, reg\_dst);

$finish;

end

endmodule