// 8-bit RISC Processor Modules and Testbenches

**1. Instruction Memory**

module InstructionMemory (

input [7:0] address,

output reg [7:0] instruction

);

reg [7:0] mem [0:255]; // 256 x 8-bit memory

initial begin

// Load some sample instructions

mem[0] = 8'h01; // Example instruction

mem[1] = 8'h02;

mem[2] = 8'h03;

end

always @(\*) begin

instruction = mem[address];

end

endmodule

// Testbench for Instruction Memory

module InstructionMemory\_tb;

reg [7:0] address;

wire [7:0] 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 [2:0] read\_reg1, read\_reg2, write\_reg,

input [7:0] write\_data,

input reg\_write,

output [7:0] read\_data1, read\_data2

);

reg [7:0] registers [0:7]; // 8 x 8-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 [2:0] read\_reg1, read\_reg2, write\_reg;

reg [7:0] write\_data;

reg reg\_write;

wire [7:0] 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 = 3'b001;

write\_data = 8'h12;

#10;

reg\_write = 0;

read\_reg1 = 3'b001;

#10;

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

$finish;

end

endmodule

**3. Data Memory**

module DataMemory (

input clk,

input [7:0] address,

input [7:0] write\_data,

input mem\_write, mem\_read,

output [7:0] read\_data

);

reg [7:0] mem [0:255]; // 256 x 8-bit memory

always @(posedge clk) begin

if (mem\_write) begin

mem[address] <= write\_data;

end

end

assign read\_data = (mem\_read) ? mem[address] : 8'h00;

endmodule

// Testbench for Data Memory

module DataMemory\_tb;

reg clk;

reg [7:0] address, write\_data;

reg mem\_write, mem\_read;

wire [7:0] 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 = 8'h01;

write\_data = 8'h34;

#10;

mem\_write = 0;

mem\_read = 1;

#10;

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

$finish;

end

endmodule

**4. ALU**

module ALU (

input [7:0] src1, src2,

input [2:0] alu\_control,

output reg [7:0] 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 = 8'h00;

endcase

end

assign zero = (result == 8'h00);

endmodule

// Testbench for ALU

module ALU\_tb;

reg [7:0] src1, src2;

reg [2:0] alu\_control;

wire [7:0] result;

wire zero;

ALU uut (

.src1(src1),

.src2(src2),

.alu\_control(alu\_control),

.result(result),

.zero(zero)

);

initial begin

src1 = 8'h01;

src2 = 8'h02;

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 [7:0] data\_in, // Data input

input [2:0] opcode, // Operation code

input [7:0] operand1, // Operand 1

input [7:0] operand2, // Operand 2

output reg [7:0] result, // Result of operation

output reg zero\_flag // Zero flag

);

reg [7:0] reg1, reg2; // Registers to store operands

// ALU Operations

always @(posedge clk or posedge reset) begin

if (reset) begin

reg1 <= 8'b0;

reg2 <= 8'b0;

result <= 8'b0;

zero\_flag <= 0;

end else begin

reg1 <= operand1;

reg2 <= operand2;

case (opcode)

3'b000: result <= reg1 + reg2; // Addition

3'b001: result <= reg1 - reg2; // Subtraction

3'b010: result <= reg1 & reg2; // AND

3'b011: result <= reg1 | reg2; // OR

3'b100: result <= reg1 ^ reg2; // XOR

default: result <= 8'b0; // Default case

endcase

// Zero flag logic

if (result == 8'b0)

zero\_flag <= 1;

else

zero\_flag <= 0;

end

end

endmodule

module datapath\_tb;

reg clk;

reg reset;

reg [7:0] data\_in;

reg [2:0] opcode;

reg [7:0] operand1;

reg [7:0] operand2;

wire [7:0] result;

wire zero\_flag;

// Instantiate the datapath module

datapath uut (

.clk(clk),

.reset(reset),

.data\_in(data\_in),

.opcode(opcode),

.operand1(operand1),

.operand2(operand2),

.result(result),

.zero\_flag(zero\_flag)

);

// Clock generation

always begin

#5 clk = ~clk;

end

// Stimulus

initial begin

// Initialize signals

clk = 0;

reset = 0;

data\_in = 8'b0;

opcode = 3'b000;

operand1 = 8'b00001010; // 10

operand2 = 8'b00000101; // 5

// Apply reset

reset = 1;

#10 reset = 0;

// Test: Addition (10 + 5)

opcode = 3'b000; // Addition

#10;

// Test: Subtraction (10 - 5)

opcode = 3'b001; // Subtraction

#10;

// Test: AND (10 & 5)

opcode = 3'b010; // AND

#10;

// Test: OR (10 | 5)

opcode = 3'b011; // OR

#10;

// Test: XOR (10 ^ 5)

opcode = 3'b100; // XOR

#10;

// End simulation

$finish;

end

endmodule

**7. Control Unit**

module ControlUnit (

input [7:0] instruction, // Changed from 15:0 to 7:0 for 8-bit

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

);

always @(\*) begin

case (instruction[7:4]) // Adjusting to the 8-bit instruction format

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 [7:0] instruction; // Changed from 15:0 to 7:0 for 8-bit

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 = 8'b0000\_XXXX)

instruction = 8'h00; // 0000 0000

#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);

// Test SUB instruction (instruction = 8'b0001\_XXXX)

instruction = 8'h01; // 0000 0001

#10;

$display("SUB: 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);

// Test LW instruction (instruction = 8'b0010\_XXXX)

instruction = 8'h02; // 0000 0010

#10;

$display("LW: 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);

// Test SW instruction (instruction = 8'b0011\_XXXX)

instruction = 8'h03; // 0000 0011

#10;

$display("SW: 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