**following unit of 16 bit MIPS processor in verilog code with the following module with their Test benches a) Instruction memory b) Register file c)Data Memory d)ALU unit e)ALU Control unit f)control Unit**

**1. Instruction Memory**

module InstructionMemory (

input [15:0] address,

output [15:0] instruction

);

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

// Initialize memory with some instructions

initial begin

mem[0] = 16'b0001000100100011; // Example instruction

mem[1] = 16'b0010001101000101; // Example instruction

end

assign instruction = mem[address];

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: %b", instruction);

address = 1;

#10;

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

$stop;

end

endmodule

**2. Register File**

module RegisterFile (

input [2:0] 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 [0:7]; // 8 x 16-bit registers

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

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

always @(\*) begin

if (reg\_write)

registers[write\_reg] = write\_data;

end

endmodule

// Testbench for Register File

module RegisterFile\_tb;

reg [2:0] read\_reg1, read\_reg2, write\_reg;

reg [15:0] write\_data;

reg reg\_write;

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

RegisterFile uut (

.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

// Write to register 1

write\_reg = 1;

write\_data = 16'h1234;

reg\_write = 1;

#10;

// Read from register 1

read\_reg1 = 1;

reg\_write = 0;

#10;

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

$stop;

end

endmodule

**3. Data Memory**

module DataMemory (

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

assign read\_data = mem\_read ? mem[address] : 16'b0;

always @(\*) begin

if (mem\_write)

mem[address] = write\_data;

end

endmodule

// Testbench for Data Memory

module DataMemory\_tb;

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

reg mem\_write, mem\_read;

wire [15:0] read\_data;

DataMemory uut (

.address(address),

.write\_data(write\_data),

.mem\_write(mem\_write),

.mem\_read(mem\_read),

.read\_data(read\_data)

);

initial begin

// Write to memory

address = 16'h0001;

write\_data = 16'hABCD;

mem\_write = 1;

mem\_read = 0;

#10;

// Read from memory

mem\_write = 0;

mem\_read = 1;

#10;

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

$stop;

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'b0;

endcase

end

assign zero = (result == 16'b0);

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'h0005;

src2 = 16'h0003;

alu\_control = 3'b000; // ADD

#10;

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

$stop;

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 Unit

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);

$stop;

end

endmodule

**6. Control Unit**

module ControlUnit (

input [2:0] opcode,

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

output reg [1:0] alu\_op

);

always @(\*) begin

case (opcode)

3'b000: begin // R-type

reg\_write = 1;

mem\_write = 0;

mem\_read = 0;

alu\_src = 0;

alu\_op = 2'b10;

end

3'b001: begin // LW

reg\_write = 1;

mem\_write = 0;

mem\_read = 1;

alu\_src = 1;

alu\_op = 2'b00;

end

3'b010: begin // SW

reg\_write = 0;

mem\_write = 1;

mem\_read = 0;

alu\_src = 1;

alu\_op = 2'b00;

end

default: begin

reg\_write = 0;

mem\_write = 0;

mem\_read = 0;

alu\_src = 0;

alu\_op = 2'b00;

end

endcase

end

endmodule

// Testbench for Control Unit

module ControlUnit\_tb;

reg [2:0] opcode;

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

wire [1:0] alu\_op;

ControlUnit uut (

.opcode(opcode),

.reg\_write(reg\_write),

.mem\_write(mem\_write),

.mem\_read(mem\_read),

.alu\_src(alu\_src),

.alu\_op(alu\_op)

);

initial begin

opcode = 3'b000; // R-type

#10;

$display("Control Signals: RegWrite=%b, MemWrite=%b, MemRead=%b, ALUSrc=%b, ALUOp=%b", reg\_write, mem\_write, mem\_read, alu\_src, alu\_op);

$stop;

end

endmodule