**following unit of 8 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**

**a) Instruction Memory**

module InstructionMemory (

input [7:0] address,

output reg [7:0] instruction

);

reg [7:0] mem [0:255]; // 256 bytes of memory

// Initialize memory with some instructions

initial begin

mem[0] = 8'b00000001; // Example instruction

mem[1] = 8'b00000010;

mem[2] = 8'b00000011;

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 = 8'd0;

#10;

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

address = 8'd1;

#10;

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

$finish;

end

endmodule

**b) 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 registers of 8 bits each

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'd1;

write\_data = 8'hAA;

#10;

reg\_write = 0;

read\_reg1 = 3'd1;

#10;

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

$finish;

end

endmodule

**c) 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 bytes of memory

always @(posedge clk) begin

if (mem\_write) begin

mem[address] <= write\_data;

end

end

assign read\_data = (mem\_read) ? mem[address] : 8'b0;

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'd10;

write\_data = 8'hFF;

#10;

mem\_write = 0;

mem\_read = 1;

#10;

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

$finish;

end

endmodule

**d) ALU Unit**

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'b0;

endcase

end

assign zero = (result == 8'b0);

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'd5;

src2 = 8'd3;

alu\_control = 3'b000; // ADD

#10;

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

alu\_control = 3'b001; // SUB

#10;

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

$finish;

end

endmodule

**e) ALU Control Unit**

module ALUControl (

input [1:0] alu\_op,

input [5:0] funct,

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)

6'b100000: alu\_control = 3'b000; // ADD

6'b100010: alu\_control = 3'b001; // SUB

6'b100100: alu\_control = 3'b010; // AND

6'b100101: alu\_control = 3'b011; // OR

6'b100111: alu\_control = 3'b100; // NOR

default: alu\_control = 3'b000;

endcase

default: alu\_control = 3'b000;

endcase

end

endmodule

// Testbench for ALU Control Unit

module ALUControl\_tb;

reg [1:0] alu\_op;

reg [5:0] funct;

wire [2:0] alu\_control;

ALUControl uut (

.alu\_op(alu\_op),

.funct(funct),

.alu\_control(alu\_control)

);

initial begin

alu\_op = 2'b10;

funct = 6'b100000; // ADD

#10;

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

funct = 6'b100010; // SUB

#10;

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

$finish;

end

endmodule

**f) Control Unit**

module ControlUnit (

input [5:0] opcode,

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

output reg [1:0] alu\_op

);

always @(\*) begin

case (opcode)

6'b000000: begin // R-type

reg\_dst = 1;

reg\_write = 1;

alu\_src = 0;

mem\_write = 0;

mem\_read = 0;

mem\_to\_reg = 0;

alu\_op = 2'b10;

end

6'b100011: begin // LW

reg\_dst = 0;

reg\_write = 1;

alu\_src = 1;

mem\_write = 0;

mem\_read = 1;

mem\_to\_reg = 1;

alu\_op = 2'b00;

end

6'b101011: begin // SW

reg\_dst = 0;

reg\_write = 0;

alu\_src = 1;

mem\_write = 1;

mem\_read = 0;

mem\_to\_reg = 0;

alu\_op = 2'b00;

end

default: begin

reg\_dst = 0;

reg\_write = 0;

alu\_src = 0;

mem\_write = 0;

mem\_read = 0;

mem\_to\_reg = 0;

alu\_op = 2'b00;

end

endcase

end

endmodule

// Testbench for Control Unit

module ControlUnit\_tb;

reg [5:0] opcode;

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

wire [1:0] alu\_op;

ControlUnit uut (

.opcode(opcode),

.reg\_dst(reg\_dst),

.reg\_write(reg\_write),

.alu\_src(alu\_src),

.mem\_write(mem\_write),

.mem\_read(mem\_read),

.mem\_to\_reg(mem\_to\_reg),

.alu\_op(alu\_op)

);

initial begin

opcode = 6'b000000; // R-type

#10;

$display("Control Signals: R-type");

opcode = 6'b100011; // LW

#10;

$display("Control Signals: LW");

$finish;

end

endmodule