Simulation Report for 4-Stage Pipelined Processor Design in Verilog

Name: Md. Sameer Ahmed

Internship Domain: VLSI

Task: Design a 4-stage pipelined processor with basic instructions: ADD, SUB, AND, LOAD

**Objective**

To design and simulate a simple 4-stage pipelined processor using Verilog HDL, supporting basic instructions such as ADD, SUB, AND, and LOAD. The processor is implemented with instruction fetch, decode, execute, and memory/write-back stages.

**Tools Used**

- Language: Verilog HDL

- Simulator: Icarus Verilog

- Waveform Viewer: GTKWave

**Processor Architecture**

**The processor design follows a 4-stage pipeline:**

1. Instruction Fetch (IF): Fetch instruction from instruction memory.

2. Instruction Decode (ID): Decode instruction and read operands from registers.

3. Execute (EX): Perform ALU operations or memory address calculation.

4. Memory/Write-Back (MEM/WB): Read data from memory or write result back to register.

**Pipelined Processor Verilog Code (with Comments)**

module PipelinedProcessor (

input clk, // Clock signal

input reset // Reset signal

);

reg [15:0] instr\_mem [0:15]; // Instruction memory

reg [7:0] data\_mem [0:15]; // Data memory

reg [7:0] regfile [0:7]; // 8 registers (R0-R7)

// Pipeline registers

reg [15:0] IF\_ID;

reg [15:0] ID\_EX;

reg [7:0] EX\_MEM\_result;

reg [2:0] EX\_MEM\_rd;

reg EX\_MEM\_mem\_read;

reg EX\_MEM\_reg\_write;

reg [7:0] MEM\_WB\_result;

reg [2:0] MEM\_WB\_rd;

reg MEM\_WB\_reg\_write;

reg [3:0] PC;

// Instruction fields

wire [3:0] opcode = IF\_ID[15:12];

wire [2:0] rd = IF\_ID[11:9];

wire [2:0] rs1 = IF\_ID[8:6];

wire [2:0] rs2 = IF\_ID[5:3];

// IF Stage

always @(posedge clk or posedge reset) begin

if (reset) begin

PC <= 0;

IF\_ID <= 0;

end else begin

IF\_ID <= instr\_mem[PC];

PC <= PC + 1;

end

end

// ID Stage

reg [7:0] reg\_rs1, reg\_rs2;

always @(posedge clk) begin

reg\_rs1 <= regfile[rs1];

reg\_rs2 <= regfile[rs2];

ID\_EX <= IF\_ID;

end

// EX Stage

reg [7:0] alu\_result;

always @(posedge clk) begin

case (ID\_EX[15:12])

4'b0001: alu\_result <= reg\_rs1 + reg\_rs2; // ADD

4'b0010: alu\_result <= reg\_rs1 - reg\_rs2; // SUB

4'b0011: alu\_result <= reg\_rs1 & reg\_rs2; // AND

4'b0100: alu\_result <= data\_mem[reg\_rs1]; // LOAD

default: alu\_result <= 8'b00000000;

endcase

EX\_MEM\_result <= alu\_result;

EX\_MEM\_rd <= ID\_EX[11:9];

EX\_MEM\_mem\_read <= (ID\_EX[15:12] == 4'b0100);

EX\_MEM\_reg\_write <= 1;

end

// MEM/WB Stage

always @(posedge clk) begin

if (EX\_MEM\_mem\_read)

MEM\_WB\_result <= EX\_MEM\_result;

else

MEM\_WB\_result <= EX\_MEM\_result;

MEM\_WB\_rd <= EX\_MEM\_rd;

MEM\_WB\_reg\_write <= EX\_MEM\_reg\_write;

end

// Write Back Stage

always @(posedge clk) begin

if (MEM\_WB\_reg\_write)

regfile[MEM\_WB\_rd] <= MEM\_WB\_result;

end

endmodule

**Testbench Verilog Code (with Comments)**

module tb\_PipelinedProcessor;

reg clk;

reg reset;

// Instantiate processor

PipelinedProcessor uut (

.clk(clk),

.reset(reset)

);

// Clock generation

always #5 clk = ~clk;

// Load instructions and data

initial begin

clk = 0; reset = 1;

#10 reset = 0;

// Instruction Memory Initialization

uut.instr\_mem[0] = 16'b0001\_001\_010\_011\_000; // ADD R1 = R2 + R3

uut.instr\_mem[1] = 16'b0010\_100\_101\_110\_000; // SUB R4 = R5 - R6

uut.instr\_mem[2] = 16'b0011\_000\_001\_010\_000; // AND R0 = R1 & R2

uut.instr\_mem[3] = 16'b0100\_011\_100\_000\_000; // LOAD R3 = MEM[R4]

// Register Initialization

uut.regfile[2] = 8'd10;

uut.regfile[3] = 8'd5;

uut.regfile[5] = 8'd20;

uut.regfile[6] = 8'd7;

uut.regfile[4] = 4; // For LOAD base address

// Data Memory Initialization

uut.data\_mem[4] = 8'd99;

#100 $finish;

end

endmodule

**Simulation Output (Expected)**

Cycle by cycle, the instructions move through the pipeline stages. Expected outputs:

- ADD: R1 = 10 + 5 = 15

- SUB: R4 = 20 - 7 = 13

- AND: R0 = R1 & R2 = 15 & 10 = 10

- LOAD: R3 = MEM[R4] = 99

**Conclusion**

The 4-stage pipelined processor was successfully implemented using Verilog. It supports ADD, SUB, AND, and LOAD instructions and simulates correct data flow through IF, ID, EX, and MEM/WB stages.