**ACCUMULATOR BASED PROCESSOR CODE (SIMULATION)**

**REGISTER MODULE:**

module RegisterFile (

input clk,

input we, // Write enable

input [7:0] addr, // 8-bit address (256 locations)

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

output reg [7:0] data\_out // Data output

);

reg [7:0] registers [0:255]; // 256x8-bit memory

always @(posedge clk) begin

if (we)

registers[addr] <= data\_in; // Write operation (non-blocking)

data\_out <= registers[addr]; // Read operation (non-blocking)

end

endmodule

**ACCUMULATOR MODULE:**

module Accumulator\_CPU (

input clk, reset,

input we, // Write Enable for instruction loading

input [3:0] instr\_addr, // Address to write instruction

input [11:0] instr\_in, // Instruction input (4-bit opcode + 8-bit operand)

output reg [7:0] AC, // Accumulator

output reg [3:0] PC // Program Counter

);

reg [11:0] instruction\_mem [0:9]; // 10 instructions (modifiable via we)

reg [3:0] opcode;

reg [7:0] operand;

reg [1:0] state; // FSM State

// FSM States

parameter FETCH = 2'b00, DECODE = 2'b01, EXECUTE = 2'b10;

always @(posedge clk or posedge reset) begin

if (reset) begin

PC <= 0;

AC <= 0;

state <= FETCH;

end else if (we) begin

instruction\_mem[instr\_addr] <= instr\_in; // Manual Instruction Loading

end else begin

case (state)

FETCH: begin

{opcode, operand} <= instruction\_mem[PC]; // Fetch instruction

PC <= PC + 1; // Increment PC

state <= DECODE;

end

DECODE: begin

state <= EXECUTE;

end

EXECUTE: begin

case (opcode)

4'b0001: AC <= operand; // LOAD

4'b0010: AC <= AC + operand; // ADD

4'b0011: AC <= AC - operand; // SUB

4'b0100: AC <= AC & operand; // AND

4'b0101: AC <= AC | operand; // OR

4'b0110: AC <= AC ^ operand; // XOR

4'b0111: AC <= ~AC; // NOT

4'b1000: AC <= AC << 1; // SHL (Shift Left)

4'b1001: AC <= AC >> 1; // SHR (Shift Right)

4'b1010: PC <= PC; // HALT

default: ;

endcase

state <= (opcode == 4'b1010) ? FETCH : FETCH;

end

endcase

end

end

endmodule

**TEST BENCH:**

`timescale 1ns / 1ps

module tb;

reg clk, reset, we;

reg [3:0] instr\_addr;

reg [11:0] instr\_in;

wire [7:0] AC;

wire [3:0] PC;

// Instantiate the CPU

Accumulator\_CPU uut (

.clk(clk),

.reset(reset),

.we(we),

.instr\_addr(instr\_addr),

.instr\_in(instr\_in),

.AC(AC),

.PC(PC)

);

// Clock Generation (10ns period)

always #5 clk = ~clk;

initial begin

// Dump file for waveform debugging

$dumpfile("cpu\_test.vcd");

$dumpvars(0, tb);

// Initialize signals

clk = 0;

reset = 1;

we = 1;

instr\_addr = 0;

instr\_in = 0;

#10 reset = 0; // Release reset

// Load Instructions into Memory

#10 we = 1; instr\_addr = 4'd0; instr\_in = 12'b000100000011; // LOAD 3

#10 we = 1; instr\_addr = 4'd1; instr\_in = 12'b001000000101; // ADD 5

#10 we = 1; instr\_addr = 4'd2; instr\_in = 12'b001100000010; // SUB 2

#10 we = 1; instr\_addr = 4'd3; instr\_in = 12'b101000000000; // HALT

#10 we = 0; // Stop writing instructions

// Run Simulation

#100;

$finish; // End Simulation

end

endmodule

**OUTPUT WAVEFORM:**

