Tiny processor

Design code :

```
`timescale 1ns / 1ps

// Arithmetic Module (Combines ADD and SUB)
module addANDsub (
    input wire op_select,           // 0 = ADD, 1 = SUB
    input wire [7:0] operand,       // Register value
    input wire [7:0] accumulator_in, // Accumulator input
    output reg [7:0] accumulator_out,// Accumulator output
    output reg carry_borrow         // Carry/Borrow flag
);

always @(*) begin
    if (op_select == 1'b0) begin
        // ADD operation
        {carry_borrow, accumulator_out} = accumulator_in + operand;
    end else begin
        // SUB operation
        {carry_borrow, accumulator_out} = accumulator_in - operand;
    end
end
endmodule

// Shift/Rotate Module
module shifter (
    input wire [7:0] accumulator_in,
    input wire [3:0] shift_op,
    output reg [7:0] accumulator_out
);

always @(*) begin
    case (shift_op)
        4'b0001: accumulator_out = accumulator_in << 1;             // LSL
        4'b0010: accumulator_out = accumulator_in >> 1;             // LSR
        4'b0011: accumulator_out = {accumulator_in[0], accumulator_in[7:1]};  // CIR
        4'b0100: accumulator_out = {accumulator_in[6:0], accumulator_in[7]};  // CIL
        4'b0101: accumulator_out = {accumulator_in[7], accumulator_in[7:1]};  // ASR
        default: accumulator_out = accumulator_in;                  // No operation
    endcase
end
```

```
endmodule

// Logical Operations Module
module logicalop (
    input wire [7:0] acc_in,
    input wire [7:0] reg_val,
    input wire [1:0] logic_op,
    output reg [7:0] acc_out,
    output reg cmp_flag
);

always @(*) begin
    acc_out = acc_in;   // Default
    cmp_flag = 0;       // Default
    case (logic_op)
        2'b00: acc_out = acc_in & reg_val;                  // AND
        2'b01: acc_out = acc_in ^ reg_val;                  // XOR
        2'b10: cmp_flag = (acc_in < reg_val) ? 1'b1 : 1'b0;  // CMP
        default: ; // No operation
    endcase
end
endmodule

module processor(
    input wire clk,
    input wire reset,
    input wire [7:0] opcode,
    output reg [3:0] program_counter,
    output reg [7:0] ACC,
    output reg [7:0] EXT,
    output reg CorB
);

    // Register file with 16 registers, each 8-bit
    reg [7:0] register [0:15];

    // Initialize registers with random values
    initial begin
        register[0] = 8'h12;
        register[1] = 8'h34;
        register[2] = 8'h56;
        register[3] = 8'h78;
        register[4] = 8'h9A;
        register[5] = 8'hBC;
```

```
    register[6] = 8'hDE;
    register[7] = 8'hF0;
    register[8] = 8'h01;
    register[9] = 8'h23;
    register[10] = 8'h45;
    register[11] = 8'h67;
    register[12] = 8'h89;
    register[13] = 8'hAB;
    register[14] = 8'hCD;
    register[15] = 8'hEF;
end

// Generate control signals for modules
wire add_sub_op;
assign add_sub_op = (opcode[7:4] == 4'b0010); // 1 for SUB, 0 for ADD

wire [1:0] logic_op_sel;
assign logic_op_sel = (opcode[7:4] == 4'b0101) ? 2'b00 :
                      (opcode[7:4] == 4'b0110) ? 2'b01 :
                      (opcode[7:4] == 4'b0111) ? 2'b10 : 2'b00;

// Instantiate modules
wire [7:0] addsub_out;
wire addsub_cb;
addANDsub addsub_module(
    .op_select(add_sub_op),
    .operand(register[opcode[3:0]]),
    .accumulator_in(ACC),
    .accumulator_out(addsub_out),
    .carry_borrow(addsub_cb)
);

wire [7:0] shift_out;
shifter shift_module(
    .accumulator_in(ACC),
    .shift_op(opcode[3:0]),
    .accumulator_out(shift_out)
);

wire [7:0] logic_out;
wire logic_cmp;
logicalop logic_module(
    .acc_in(ACC),
    .reg_val(register[opcode[3:0]]),
```

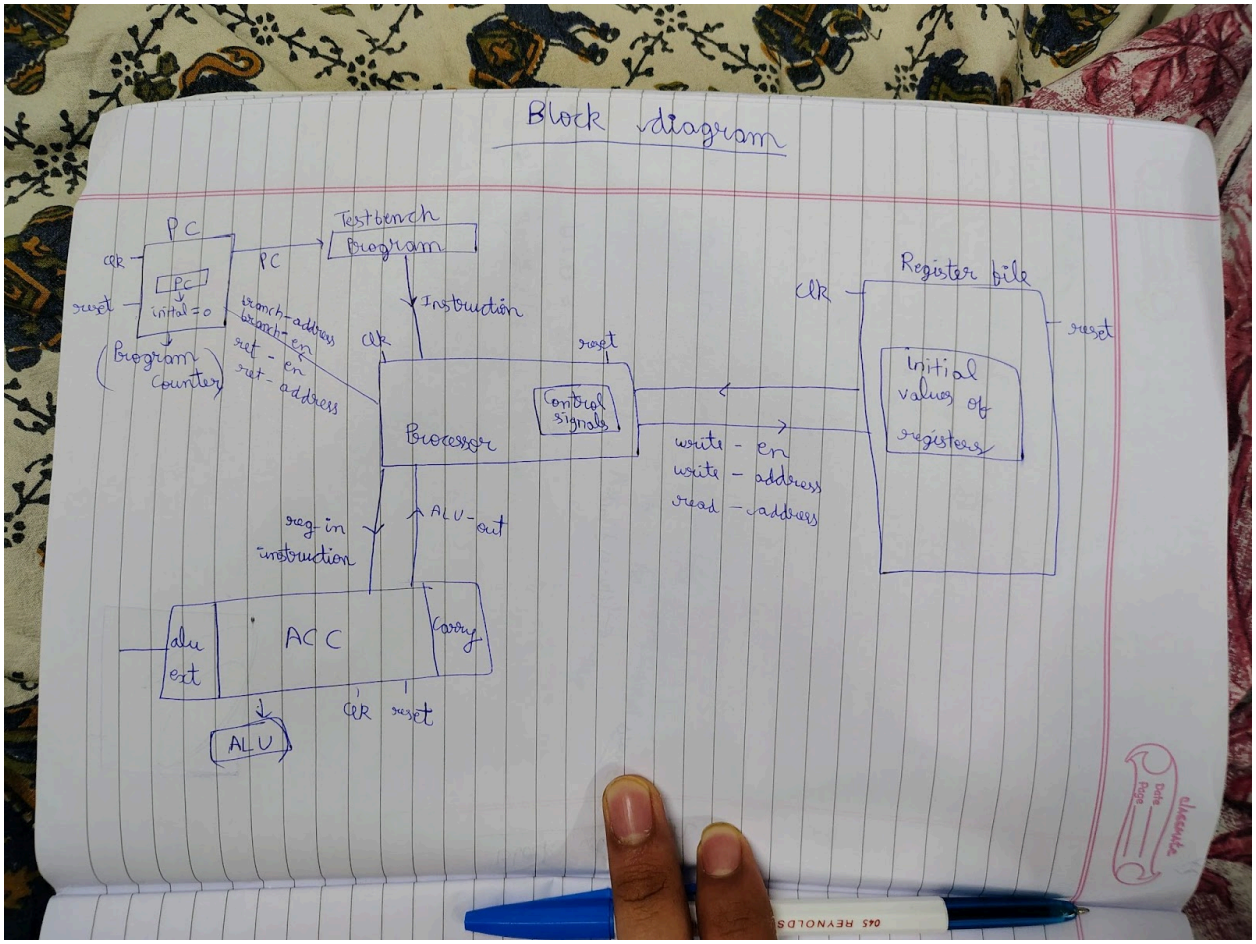
```
    .logic_op(logic_op_sel),
    .acc_out(logic_out),
    .cmp_flag(logic_cmp)
);

// Main processor operation
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // Reset all registers
        program_counter <= 4'b0000;
        ACC <= 8'b00000000;
        EXT <= 8'b00000000;
        CorB <= 1'b0;
    end else begin
        // Default operation: increment PC
        program_counter <= program_counter + 1;

        // Process instruction based on opcode
        case (opcode[7:4])
            4'b0000: begin // NOP and shift operations
                case (opcode[3:0])
                    4'b0000: ; // NOP - do nothing
                    4'b0001: ACC <= shift_out; // LSL
                    4'b0010: ACC <= shift_out; // LSR
                    4'b0011: ACC <= shift_out; // CIR
                    4'b0100: ACC <= shift_out; // CIL
                    4'b0101: ACC <= shift_out; // ASR
                    4'b0110: begin // INC
                        {CorB, ACC} <= ACC + 1;
                    end
                    4'b0111: begin // DEC
                        {CorB, ACC} <= ACC - 1;
                    end
                    default: ; // Invalid opcode - do nothing
                endcase
            end
            4'b0001: begin // ADD
                ACC <= addsub_out;
                CorB <= addsub_cb;
            end
            4'b0010: begin // SUB
                ACC <= addsub_out;
                CorB <= addsub_cb;
            end
```

```
            4'b0011: begin // MUL
               {EXT, ACC} <= ACC * register[opcode[3:0]];
            end
            4'b0101: begin // AND
               ACC <= logic_out;
            end
            4'b0110: begin // XRA (XOR)
               ACC <= logic_out;
            end
            4'b0111: begin // CMP
               CorB <= logic_cmp;
            end
            4'b1000: begin // Branch if C/B=1
               if (CorB) begin
                  program_counter <= opcode[3:0];
               end
            end
            4'b1001: begin // MOV ACC, Ri
               ACC <= register[opcode[3:0]];
            end
            4'b1010: begin // MOV Ri, ACC
               register[opcode[3:0]] <= ACC;
            end
            4'b1011: begin // Return
               program_counter <= opcode[3:0];
            end
            4'b1111: begin // HLT
               if (opcode[3:0] == 4'b1111) begin
                  program_counter <= program_counter; // Stop incrementing PC
               end
            end
            default: ; // Invalid opcode - do nothing
         endcase
      end
   end
endmodule
```

Block diagram



Testbentch 1:

```
`timescale 1ns / 1ps

module processor_tb;

    // Inputs
    reg clk;
    reg reset;
    reg [7:0] opcode;

    // Outputs
    wire [3:0] program_counter;
    wire [7:0] ACC;
    wire [7:0] EXT;
    wire CorB;
```

```
// Instantiate the Unit Under Test (UUT)
processor uut (
    .clk(clk),
    .reset(reset),
    .opcode(opcode),
    .program_counter(program_counter),
    .ACC(ACC),
    .EXT(EXT),
    .CorB(CorB)
);

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

// Test procedure
initial begin
    // Initialize Inputs
    reset = 1;
    opcode = 8'b00000000; // NOP

    // Wait for global reset
    #20;
    reset = 0;

    // Test 1: Basic Arithmetic Operations
    $display("\nTest 1: Basic Arithmetic Operations");
    // Load R5 into ACC
    opcode = 8'b10010001; // MOV ACC, R1 (R1=8'h34)
    #10;
    $display("MOV ACC, R1: ACC=%h, PC=%d", ACC, program_counter);

    // Add R5 (R5=8'hBC)
    opcode = 8'b00010101; // ADD R5
    #10;
    $display("ADD R5: ACC=%h, C/B=%b, PC=%d", ACC, CorB, program_counter);

    // Subtract R2 (R2=8'h56)
    opcode = 8'b00100010; // SUB R2
    #10;
    $display("SUB R2: ACC=%h, C/B=%b, PC=%d", ACC, CorB, program_counter);
```

```
        // Test 2: Multiplication
        $display("\nTest 2: Multiplication");
        // Load R3 into ACC (R3=8'h78)
        opcode = 8'b10010011; // MOV ACC, R3
        #10;
        $display("MOV ACC, R3: ACC=%h, PC=%d", ACC, program_counter);

        // Multiply with R4 (R4=8'h9A)
        opcode = 8'b00110100; // MUL R4
        #10;
        $display("MUL R4: ACC=%h (low), EXT=%h (high), PC=%d", ACC, EXT,
program_counter);

        // Test 3: Shift Operations
        $display("\nTest 3: Shift Operations");
        // Load R6 into ACC (R6=8'hDE)
        opcode = 8'b10010110; // MOV ACC, R6
        #10;
        $display("MOV ACC, R6: ACC=%h, PC=%d", ACC, program_counter);

        // Logical Shift Left
        opcode = 8'b00000001; // LSL
        #10;
        $display("LSL: ACC=%h, PC=%d", ACC, program_counter);

        // Logical Shift Right
        opcode = 8'b00000010; // LSR
        #10;
        $display("LSR: ACC=%h, PC=%d", ACC, program_counter);

        // Arithmetic Shift Right
        opcode = 8'b00000101; // ASR
        #10;
        $display("ASR: ACC=%h, PC=%d", ACC, program_counter);

        // Test 4: Logical Operations
        $display("\nTest 4: Logical Operations");
        // Load R7 into ACC (R7=8'hF0)
        opcode = 8'b10010111; // MOV ACC, R7
        #10;
        $display("MOV ACC, R7: ACC=%h, PC=%d", ACC, program_counter);

        // AND with R8 (R8=8'h01)
        opcode = 8'b01011000; // AND R8
```

```
#10;
$display("AND R8: ACC=%h, PC=%d", ACC, program_counter);

// XOR with R9 (R9=8'h23)
opcode = 8'b01101001; // XRA R9
#10;
$display("XOR R9: ACC=%h, PC=%d", ACC, program_counter);

// Test 5: Comparison and Branching
$display("\nTest 5: Comparison and Branching");
// Load R10 into ACC (R10=8'h45)
opcode = 8'b10011010; // MOV ACC, R10
#10;
$display("MOV ACC, R10: ACC=%h, PC=%d", ACC, program_counter);

// Compare with R11 (R11=8'h67)
opcode = 8'b01111011; // CMP R11
#10;
$display("CMP R11: C/B=%b (ACC < R11), PC=%d", CorB, program_counter);

// Branch if C/B=1 (should branch)
opcode = 8'b10000010; // Br to address 2
#10;
$display("Branch taken: PC=%d", program_counter);

// Test 6: Move Operations
$display("\nTest 6: Move Operations");
// Move R12 to ACC (R12=8'h89)
opcode = 8'b10011100; // MOV ACC, R12
#10;
$display("MOV ACC, R12: ACC=%h, PC=%d", ACC, program_counter);

// Move ACC to R13
opcode = 8'b10101101; // MOV R13, ACC
#10;
$display("MOV R13, ACC: R13=%h, PC=%d", uut.register[13], program_counter);

// Test 7: Increment/Decrement
$display("\nTest 7: Increment/Decrement");
// Increment ACC
opcode = 8'b00000110; // INC
#10;
$display("INC: ACC=%h, C/B=%b, PC=%d", ACC, CorB, program_counter);
```

```
        // Decrement ACC
        opcode = 8'b00000111; // DEC
        #10;
        $display("DEC: ACC=%h, C/B=%b, PC=%d", ACC, CorB, program_counter);

        // Test 8: Halt Instruction
        $display("\nTest 8: Halt Instruction");
        opcode = 8'b11111111; // HLT
        #10;
        $display("HLT: PC should stop incrementing. Current PC=%d", program_counter);
        #10;
        $display("After another cycle, PC=%d (should be same)", program_counter);

        // End simulation
        #20;
        $display("\nAll tests completed");
        $finish;
    end

    // Monitor changes
    initial begin
        $monitor("Time=%0t: PC=%d, Opcode=%b, ACC=%h, EXT=%h, C/B=%b",
                 $time, program_counter, opcode, ACC, EXT, CorB);
    end

endmodule
```

TCL output :

```
Time=0: PC= 0, Opcode=00000000, ACC=00, EXT=00, C/B=0

Test 1: Basic Arithmetic Operations
Time=20000: PC= 0, Opcode=10010001, ACC=00, EXT=00, C/B=0
Time=25000: PC= 1, Opcode=10010001, ACC=34, EXT=00, C/B=0
MOV ACC, R1: ACC=34, PC= 1
```

Time=30000: PC= 1, Opcode=00010101, ACC=34, EXT=00, C/B=0
Time=35000: PC= 2, Opcode=00010101, ACC=f0, EXT=00, C/B=0
ADD R5: ACC=f0, C/B=0, PC= 2
Time=40000: PC= 2, Opcode=00100010, ACC=f0, EXT=00, C/B=0
Time=45000: PC= 3, Opcode=00100010, ACC=9a, EXT=00, C/B=0
SUB R2: ACC=9a, C/B=0, PC= 3

Test 2: Multiplication
Time=50000: PC= 3, Opcode=10010011, ACC=9a, EXT=00, C/B=0
Time=55000: PC= 4, Opcode=10010011, ACC=78, EXT=00, C/B=0
MOV ACC, R3: ACC=78, PC= 4
Time=60000: PC= 4, Opcode=00110100, ACC=78, EXT=00, C/B=0
Time=65000: PC= 5, Opcode=00110100, ACC=30, EXT=48, C/B=0
MUL R4: ACC=30 (low), EXT=48 (high), PC= 5

Test 3: Shift Operations
Time=70000: PC= 5, Opcode=10010110, ACC=30, EXT=48, C/B=0
Time=75000: PC= 6, Opcode=10010110, ACC=de, EXT=48, C/B=0
MOV ACC, R6: ACC=de, PC= 6
Time=80000: PC= 6, Opcode=00000001, ACC=de, EXT=48, C/B=0
Time=85000: PC= 7, Opcode=00000001, ACC=bc, EXT=48, C/B=0
LSL: ACC=bc, PC= 7
Time=90000: PC= 7, Opcode=00000010, ACC=bc, EXT=48, C/B=0
Time=95000: PC= 8, Opcode=00000010, ACC=5e, EXT=48, C/B=0
LSR: ACC=5e, PC= 8
Time=100000: PC= 8, Opcode=00000101, ACC=5e, EXT=48, C/B=0
Time=105000: PC= 9, Opcode=00000101, ACC=2f, EXT=48, C/B=0
ASR: ACC=2f, PC= 9

Test 4: Logical Operations
Time=110000: PC= 9, Opcode=10010111, ACC=2f, EXT=48, C/B=0
Time=115000: PC=10, Opcode=10010111, ACC=f0, EXT=48, C/B=0
MOV ACC, R7: ACC=f0, PC=10
Time=120000: PC=10, Opcode=01011000, ACC=f0, EXT=48, C/B=0
Time=125000: PC=11, Opcode=01011000, ACC=00, EXT=48, C/B=0
AND R8: ACC=00, PC=11
Time=130000: PC=11, Opcode=01101001, ACC=00, EXT=48, C/B=0
Time=135000: PC=12, Opcode=01101001, ACC=23, EXT=48, C/B=0
XOR R9: ACC=23, PC=12

Test 5: Comparison and Branching
Time=140000: PC=12, Opcode=10011010, ACC=23, EXT=48, C/B=0
Time=145000: PC=13, Opcode=10011010, ACC=45, EXT=48, C/B=0
MOV ACC, R10: ACC=45, PC=13

Time=150000: PC=13, Opcode=01111011, ACC=45, EXT=48, C/B=0
Time=155000: PC=14, Opcode=01111011, ACC=45, EXT=48, C/B=1
CMP R11: C/B=1 (ACC < R11), PC=14
Time=160000: PC=14, Opcode=10000010, ACC=45, EXT=48, C/B=1
Time=165000: PC= 2, Opcode=10000010, ACC=45, EXT=48, C/B=1
Branch taken: PC= 2

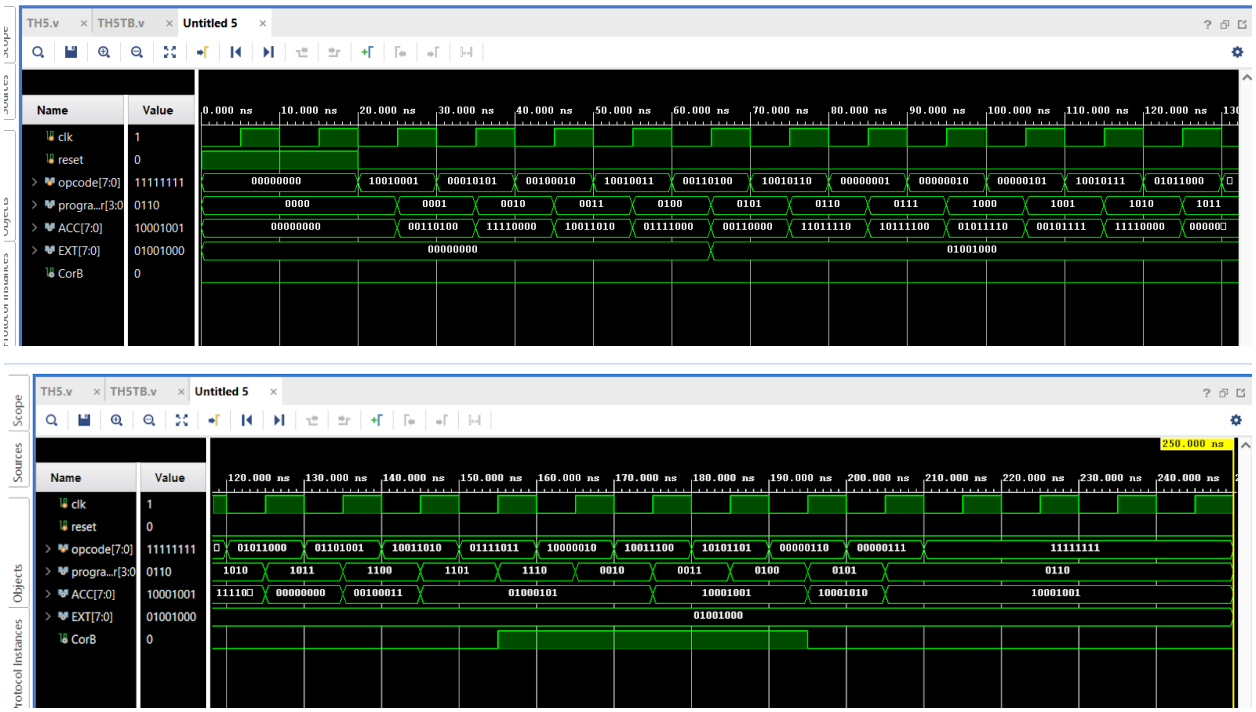
Test 6: Move Operations
Time=170000: PC= 2, Opcode=10011100, ACC=45, EXT=48, C/B=1
Time=175000: PC= 3, Opcode=10011100, ACC=89, EXT=48, C/B=1
MOV ACC, R12: ACC=89, PC= 3
Time=180000: PC= 3, Opcode=10101101, ACC=89, EXT=48, C/B=1
Time=185000: PC= 4, Opcode=10101101, ACC=89, EXT=48, C/B=1
MOV R13, ACC: R13=89, PC= 4

Test 7: Increment/Decrement
Time=190000: PC= 4, Opcode=00000110, ACC=89, EXT=48, C/B=1
Time=195000: PC= 5, Opcode=00000110, ACC=8a, EXT=48, C/B=0
INC: ACC=8a, C/B=0, PC= 5
Time=200000: PC= 5, Opcode=00000111, ACC=8a, EXT=48, C/B=0
Time=205000: PC= 6, Opcode=00000111, ACC=89, EXT=48, C/B=0
DEC: ACC=89, C/B=0, PC= 6

Test 8: Halt Instruction
Time=210000: PC= 6, Opcode=11111111, ACC=89, EXT=48, C/B=0
HLT: PC should stop incrementing. Current PC= 6
After another cycle, PC= 6 (should be same)

All tests completed

Simulation 1:



Testbench 2:

```
`timescale 1ns / 1ps

module processor_simple_tb;

    // Inputs
    reg clk;
    reg reset;
    reg [7:0] opcode;
```

```
    // Outputs
    wire [3:0] program_counter;
    wire [7:0] ACC;
    wire [7:0] EXT;
    wire CorB;

    // Instantiate the processor
    processor uut (
        .clk(clk),
        .reset(reset),
        .opcode(opcode),
        .program_counter(program_counter),
        .ACC(ACC),
        .EXT(EXT),
        .CorB(CorB)
    );

    // Clock generation (100MHz)
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Test procedure
    initial begin
        // Initialize
        reset = 1;
        opcode = 8'b00000000; // NOP
        #20;
        reset = 0;

        $display("\n--- Starting Simple Testbench ---");
        $display("Time\tPC\tOpcode\t\tACC\tEXT\tC/B\tOperation");

        // Test 1: Circular Shifts
        opcode = 8'b10010010; // MOV ACC, R2 (R2=56h)
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tMOV ACC, R2", $time, program_counter, opcode,
ACC, EXT, CorB);

        opcode = 8'b00000011; // CIR (Circular Right)
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tCIR", $time, program_counter, opcode, ACC, EXT,
CorB);
```

```
        opcode = 8'b00000100; // CIL (Circular Left)
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tCIL", $time, program_counter, opcode, ACC, EXT,
CorB);

        // Test 2: Division Simulation (using MUL/EXT)
        opcode = 8'b10010011; // MOV ACC, R3 (R3=78h)
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tMOV ACC, R3", $time, program_counter, opcode,
ACC, EXT, CorB);

        opcode = 8'b00110100; // MUL R4 (R4=9Ah)
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tMUL R4", $time, program_counter, opcode, ACC,
EXT, CorB);

        // Test 3: Flag Operations
        opcode = 8'b00000110; // INC
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tINC", $time, program_counter, opcode, ACC, EXT,
CorB);

        opcode = 8'b00000111; // DEC
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tDEC", $time, program_counter, opcode, ACC, EXT,
CorB);

        // Test 4: Conditional Branch
        opcode = 8'b01110100; // CMP R4 (R4=9Ah)
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tCMP R4", $time, program_counter, opcode, ACC,
EXT, CorB);

        opcode = 8'b10000101; // Br to 5 if C/B=1
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tBranch", $time, program_counter, opcode, ACC,
EXT, CorB);

        // Test 5: Halt
        opcode = 8'b11111111; // HLT
        #10;
        $display("%0t\t%h\t%b\t%h\t%h\t%b\tHLT", $time, program_counter, opcode, ACC, EXT,
CorB);
```

```
        #10;

        $display("\n--- Test Complete ---");
        $finish;
    end

endmodule
```

TCL output 2:

```
--- Starting Simple Testbench ---
Time    PC      Opcode          ACC     EXT     C/B     Operation
30000   1       10010010  56    00      0       MOV ACC, R2
40000   2       00000011  2b    00      0       CIR
50000   3       00000100  56    00      0       CIL
60000   4       10010011  78    00      0       MOV ACC, R3
70000   5       00110100  30    48      0       MUL R4
80000   6       00000110  31    48      0       INC
90000   7       00000111  30    48      0       DEC
100000          8       01110100  30    48      1       CMP R4
1100005         10000101  30    48      1       Branch
120000          5       11111111  30    48      1       HLT

--- Test Complete ---
```

Simulation 2:


TH5.v
TH5TB.v
Untitled 1
Scope
Sources
Objects
Protocol Instances
Name
Value
clk
0
reset
0
opcode[7:0]
00110100
progra...r[3:0]
0100
ACC[7:0]
01111000
EXT[7:0]
00000000
CorB
0
60.000 ns
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns
120.000 ns
00000000
10010010
00000011
00000100
10010011
00110100
00000110
00000111
01110100
10000101
11111111
0000
0001
0010
0011
0100
0101
0110
0111
1000
0101
00000000
01010110
00101011
01010110
01111000
00110000
00110001
00110000
00000000
01001000