Team Members:

1. Aditya Prakash Borate (23110065)
2. Haarit Ravindrakumar Chavda (23110077)

Complete Tiny Processor

• Synthesizable Design Code

- ProgramMemory.v

```
 1. Arithmetic Demo: ADD, SUB, INC, DEC, MOV, NOP, HLT
`timescale 1ns / 1ps
module ProgramMemory(
    input  [3:0] PC,
    output reg [7:0] Instruction
);
    reg [7:0] MEMORY [15:0];
    initial begin
        MEMORY[0] = 8'b0000_0000; // NOP
        MEMORY[1] = 8'b1001_0001; // MOV ACC,R1
        MEMORY[2] = 8'b0001_0010; // ADD R2
        MEMORY[3] = 8'b0010_0011; // SUB R3
        MEMORY[4] = 8'b0000_0110; // INC ACC
        MEMORY[5] = 8'b0000_0111; // DEC ACC
        MEMORY[6] = 8'b1010_0100; // MOV R4,ACC
        MEMORY[7] = 8'b1111_1111; // HLT
        MEMORY[8] = 8'b0000_0000;
        MEMORY[9] = 8'b0000_0000;
        MEMORY[10]= 8'b0000_0000;
        MEMORY[11]= 8'b0000_0000;
        MEMORY[12]= 8'b0000_0000;
        MEMORY[13]= 8'b0000_0000;
        MEMORY[14]= 8'b0000_0000;
        MEMORY[15]= 8'b1111_1111;
    end
    always @(*) Instruction = MEMORY[PC];
endmodule




//  2. MUL & SUB Demo: MUL, SUB, HLT
//`timescale 1ns / 1ps
//module ProgramMemory(
//    input  [3:0] PC,
//    output reg [7:0] Instruction
//);
//    reg [7:0] MEMORY [15:0];
//    initial begin
```

```
//        MEMORY[0] = 8'b1001_0001; // MOV ACC,R1
//        MEMORY[1] = 8'b0011_0010; // MUL R2
//        MEMORY[2] = 8'b0010_0011; // SUB R3
//        MEMORY[3] = 8'b1111_1111; // HLT
//        MEMORY[4] = 8'b0000_0000;
//        MEMORY[5] = 8'b0000_0000;
//        MEMORY[6] = 8'b0000_0000;
//        MEMORY[7] = 8'b0000_0000;
//        MEMORY[8] = 8'b0000_0000;
//        MEMORY[9] = 8'b0000_0000;
//        MEMORY[10]= 8'b0000_0000;
//        MEMORY[11]= 8'b0000_0000;
//        MEMORY[12]= 8'b0000_0000;
//        MEMORY[13]= 8'b0000_0000;
//        MEMORY[14]= 8'b0000_0000;
//        MEMORY[15]= 8'b1111_1111;
//    end
//    always @(*) Instruction = MEMORY[PC];
//endmodule


// 3. All shifts + AND, XRA, NOP, HLT
//`timescale 1ns / 1ps
//module ProgramMemory(
//    input  [3:0] PC,
//    output reg [7:0] Instruction
//);
//    reg [7:0] MEMORY [15:0];
//    initial begin
//        MEMORY[0] = 8'b0000_0000; // NOP
//        MEMORY[1] = 8'b1001_0001; // MOV ACC,R1
//        MEMORY[2] = 8'b0000_0001; // LSL ACC
//        MEMORY[3] = 8'b0000_0010; // LSR ACC
//        MEMORY[4] = 8'b0000_0011; // CIR ACC
//        MEMORY[5] = 8'b0000_0100; // CIL ACC
//        MEMORY[6] = 8'b0000_0101; // ASR ACC
//        MEMORY[7] = 8'b0101_0010; // AND R2
//        MEMORY[8] = 8'b0110_0011; // XRA R3
//        MEMORY[9]= 8'b1111_1111; // HLT
//        // fill with NOP
//        MEMORY[10]= 8'b0000_0000;
//        MEMORY[11]= 8'b0000_0000;
//        MEMORY[12]= 8'b0000_0000;
//        MEMORY[13]= 8'b0000_0000;
//        MEMORY[14]= 8'b0000_0000;
//        MEMORY[15]= 8'b1111_1111;
//    end
//    always @(*) Instruction = MEMORY[PC];
//endmodule



// 4. Branch & Return Demo: CMP, Br, Ret, MOV, HLT
//`timescale 1ns / 1ps
//module ProgramMemory(
```

```
//    input  [3:0] PC,
//    output reg [7:0] Instruction
//);
//    reg [7:0] MEMORY [15:0];
//    initial begin
//        // If R1 < R2 then load ACC?R4 else ACC?R3; then save to R5, return to HLT
//        MEMORY[0] = 8'b1001_0001; // MOV ACC,R1
//        MEMORY[1] = 8'b0111_0010; // CMP R2
//        MEMORY[2] = 8'b1000_0101; // Br 5
//        MEMORY[3] = 8'b1001_0011; // MOV ACC,R3
//        MEMORY[4] = 8'b1000_0110; // Br 6
//        MEMORY[5] = 8'b1001_0100; // MOV ACC,R4
//        MEMORY[6] = 8'b1010_0101; // MOV R5,ACC
//        MEMORY[7] = 8'b1011_1001; // RET 9
//        MEMORY[8] = 8'b0000_0000; // NOP (dead)
//        MEMORY[9] = 8'b1111_1111; // HLT
//        // unused
//        MEMORY[10]= 8'b0000_0000;
//        MEMORY[11]= 8'b0000_0000;
//        MEMORY[12]= 8'b0000_0000;
//        MEMORY[13]= 8'b0000_0000;
//        MEMORY[14]= 8'b0000_0000;
//        MEMORY[15]= 8'b1111_1111;
//    end
//    always @(*) Instruction = MEMORY[PC];
//endmodule


//5. Upto counter upto 15
//`timescale 1ns / 1ps

//module ProgramMemory(
//    input [3:0] PC,
//    output reg [7:0] Instruction
//);
//    reg [7:0] MEMORY [15:0];
//    initial begin
//        //Increment ACC (starting from R1) until it equals R15

//        MEMORY[0] = 8'b1001_0000; // MOV ACC, R1
//        MEMORY[1] = 8'b0000_0110; // INC ACC
//        MEMORY[2] = 8'b0111_1111; // CMP R15
//        MEMORY[3] = 8'b1000_0001; // Br 2
//        MEMORY[4] = 8'b1111_1111; // HLT
//        MEMORY[5] = 8'b1111_1111; // HLT

//        MEMORY[6] = 8'b0000_0000;
//        MEMORY[7] = 8'b0000_0000;
//        MEMORY[8] = 8'b0000_0000;
//        MEMORY[9] = 8'b0000_0000;
//        MEMORY[10] = 8'b0000_0000;
//        MEMORY[11] = 8'b0000_0000;
//        MEMORY[12] = 8'b0000_0000;
//        MEMORY[13] = 8'b0000_0000;
//        MEMORY[14] = 8'b0000_0000;
```

```
//        MEMORY[15] = 8'b1111_1111;
//    end

//    always @(*) begin
//        Instruction = MEMORY[PC];
//    end
//endmodule
```

- Registerfile.v

```
`timescale 1ns / 1ps
module Register #(
    parameter INIT_VAL = 8'd0
)(
    input clk,
    input we,
    input [7:0] d,
    output reg [7:0] q
);
    initial begin
        q = INIT_VAL;
    end

    always @(posedge clk) begin
        if (we)
            q <= d;
    end
endmodule

module RegisterFile (
    input clk,
    input [3:0] Addr,         // Address to read/write
    input [7:0] WriteData,    // Data to write
    input WriteEnable,        // Write enable signal
    output [7:0] ReadData     // Output data
);
    wire [7:0] reg_out [15:0];
    reg [15:0] we_vector;

    integer i;
    always @(*) begin
        for (i = 0; i < 16; i = i + 1)
            we_vector[i] = (WriteEnable && (Addr == i));
    end

    // Generate 16 Register Instances
    genvar idx;
    generate
        for (idx = 0; idx < 16; idx = idx + 1) begin : reg_block
            Register #(.INIT_VAL(idx)) reg_inst (
                .clk(clk),
                .we(we_vector[idx]),
                .d(WriteData),
                .q(reg_out[idx])
```

```
        );
    end
  endgenerate

  assign ReadData = reg_out[Addr];

endmodule
```

- ALU.v

```
`timescale 1ns / 1ps
// ALU: Performs Arithmetic and Logical Operations
module ALU (
  input [7:0] ACC,          // Accumulator input
  input [7:0] Ri,           // Register input
  input [7:0] Opcode,       // Instruction Opcode
  output reg [7:0] Result,  // ALU output
  output reg CB,            // Carry/Borrow
  output reg [7:0] EXT      // Extended register for MUL result
);
  reg [8:0] temp;
  reg [15:0] temp_mul;


  initial begin
    // Default output assignments
    Result = ACC;
    CB = 0;
    EXT = 0;

  end

  always @(*) begin

    case (Opcode[7:4])
      4'b0001: begin // ADD
        temp = {1'b0, ACC} + {1'b0, Ri};
        Result = temp[7:0];
        CB = temp[8];
      end
      4'b0010: begin // SUB
        temp = {1'b0, ACC} - {1'b0, Ri};
        Result = temp[7:0];
        CB = temp[8];
      end
      4'b0011: begin // MUL
        temp_mul = ACC * Ri;
        Result = temp_mul[7:0];
        EXT = temp_mul[15:8];
      end
      4'b0101: begin // AND
        Result = ACC & Ri;
      end
      4'b0110: begin // XRA Ri (bitwise XOR)
```

```
            Result = ACC ^ Ri;
        end
        4'b0111: begin // CMP Ri (Compare ACC with Ri)
            // If ACC > Ri then CB is set to 1, otherwise 0.
            CB = (ACC < Ri) ? 1'b1 : 1'b0;
        end
        4'b0000: begin // Shift, Increment, Decrement operations
            case (Opcode[3:0])
                4'b0001: Result = ACC << 1;                 //  Logical Shift Left
                4'b0010: Result = ACC >> 1;                 //  Logical Shift Right
                4'b0011: Result = {ACC[0], ACC[7:1]};       //  Circuit (rotate) right
                4'b0100: Result = {ACC[6:0], ACC[7]};       //  Circuit (rotate) left
                4'b0101: Result = {ACC[7], ACC[7:1]};       //  Arithmetic Shift Right
                4'b0110: begin                          //  Increment ACC
                    temp = {1'b0, ACC} + 9'd1;
                    Result = temp[7:0];
                    CB = temp[8];
                end
                4'b0111: begin                          //  Decrement ACC
                    temp = {1'b0, ACC} - 9'd1;
                    Result = temp[7:0];
                    CB = temp[8];
                end
                default: Result = ACC; //  default case for undefined sub-opcodes
            endcase
        end
        default: Result = ACC; //  default case for undefined opcodes
    endcase
  end
endmodule
```

- Processor.v

```
`timescale 1ns / 1ps

// Top module for processor
module Processor (
    input clk,
    input rstn,
    output reg  [7:0] ACC
);
    // Program counter
    reg  [3:0] PC;

    // Extra space to store result of mulitplication
    wire [7:0] EXT;

    // Carry bit for addition , subtractor and for branch instruction
    wire       CB;

    // Wires for connecting instructions from program memory to input of alu
    wire [7:0] Instruction;

    // Wires to connect the active register to input of alu
    wire [7:0] RegData;
```

```
// Wires to connect the result of alu to accumulator
wire [7:0] ALUResult;

// Storage to extract adxdress from operation code
reg  [3:0] Addr;

// Flags for  write,branch,return and halt
reg        WriteEnable;
reg        IsBranch;
reg        IsRet;
reg        IsHalt;


// Instantiating all the modules
ProgramMemory U1(PC, Instruction);
RegisterFile  U2(clk, Addr, ACC, WriteEnable, RegData );
ALU           U3(ACC, RegData, Instruction, ALUResult, CB, EXT);


// Decoding the operation code
always @(*) begin
   Addr        = Instruction[3:0];
   WriteEnable = 0;
   IsBranch    = 0;
   IsRet       = 0;
   IsHalt      = 0;

   case (Instruction[7:4])
      4'b1010: WriteEnable = 1;
      4'b1000: IsBranch    = 1;
      4'b1011: IsRet       = 1;
      4'b1111: IsHalt      = 1;
   endcase
end


always @(posedge clk or negedge rstn) begin

   // reset
   if (!rstn) begin
      PC     <= 4'd0;
      ACC    <= 8'd0;
   end

   else begin
      // Either move data from register or update the acc by alu result
      case (Instruction[7:4])
         4'b1001: ACC <= RegData;

         default: ACC <= ALUResult;
      endcase

      // Branching Condition
      if (IsBranch && CB)
```

```
            PC <= Instruction[3:0];
        // Return condition
        else if (IsRet)
            PC <= Instruction[3:0];
        // Halting condition
        else if (!IsHalt)
            PC <= PC + 1;
        else
            PC <= PC;
    end
  end
endmodule
```

• Testbench Code

- ProgramMemory_tb.v

```
`timescale 1ns / 1ps

module ProgramMemory_tb;

  reg [3:0] PC;
  wire [7:0] Instruction;

  // Instantiate the ProgramMemory module
  ProgramMemory uut (
      .PC(PC),
      .Instruction(Instruction)
  );

  integer i;

  initial begin
      // Loop through all possible PC values (0 to 15)
      for (i = 0; i < 16; i = i + 1) begin
          PC = i[3:0];
          #10; // Wait for 10 ns to allow Instruction to update
      end
      #10; // Final delay before simulation ends
      $finish;
  end

endmodule
```

- Registerfile_tb.v

```
`timescale 1ns / 1ps

module RegisterFile_tb;
  reg clk;
  reg [3:0] Addr;
  reg [7:0] WriteData;
```

```
    reg WriteEnable;
    wire [7:0] ReadData;

    RegisterFile uut (
        .clk(clk),
        .Addr(Addr),
        .WriteData(WriteData),
        .WriteEnable(WriteEnable),
        .ReadData(ReadData)
    );

    initial begin
        $display("Time\tWriteEnable\tAddr\tWriteData\tReadData");
        $monitor("%g\t%b\t\t%h\t%h\t\t%h", $time, WriteEnable, Addr, WriteData,
ReadData);
    end

    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns clock period
    end

    initial begin
        // Read default value
        WriteEnable = 0;
        Addr = 4'd2; #10;

        // Write 0xAA to register 2
        WriteEnable = 1;
        Addr = 4'd2;
        WriteData = 8'hAA; #10;

        // Read back from register 2
        WriteEnable = 0;
        Addr = 4'd2; #10;

        // Write 0xFF to register 5
        WriteEnable = 1;
        Addr = 4'd5;
        WriteData = 8'hFF; #10;

        // Read back from register 5
        WriteEnable = 0;
        Addr = 4'd5; #10;

        $finish;
    end
endmodule
```

- ALU_tb.v

```
`timescale 1ns / 1ps

module ALU_tb;
    reg [7:0] ACC;
```

```
    reg [7:0] Ri;
    reg [7:0] Opcode;
    wire [7:0] Result;
    wire CB;
    wire [7:0] EXT;

    ALU uut (
        .ACC(ACC),
        .Ri(Ri),
        .Opcode(Opcode),
        .Result(Result),
        .CB(CB),
        .EXT(EXT)
    );

    initial begin
        // ADD R1
        ACC = 8'd10; Ri = 8'd20; Opcode = 8'b0001_0001; #10;

        // SUB R2
        ACC = 8'd30; Ri = 8'd25; Opcode = 8'b0010_0010; #10;

        // MUL R3
        ACC = 8'd5; Ri = 8'd4; Opcode = 8'b0011_0011; #10;

        // AND R4
        ACC = 8'b10101010; Ri = 8'b11110000; Opcode = 8'b0101_0100; #10;

        // XRA R5
        ACC = 8'b10101010; Ri = 8'b11110000; Opcode = 8'b0110_0101; #10;

        // CMP R6
        ACC = 8'd10; Ri = 8'd15; Opcode = 8'b0111_0110; #10;

        // LSL ACC
        ACC = 8'b00001111; Ri = 8'd0; Opcode = 8'b0000_0001; #10;

        // LSR ACC
        ACC = 8'b11110000; Ri = 8'd0; Opcode = 8'b0000_0010; #10;

        // INC ACC
        ACC = 8'd255; Ri = 8'd0; Opcode = 8'b0000_0110; #10;

        // DEC ACC
        ACC = 8'd0; Ri = 8'd0; Opcode = 8'b0000_0111; #10;

        $finish;
    end
endmodule
```

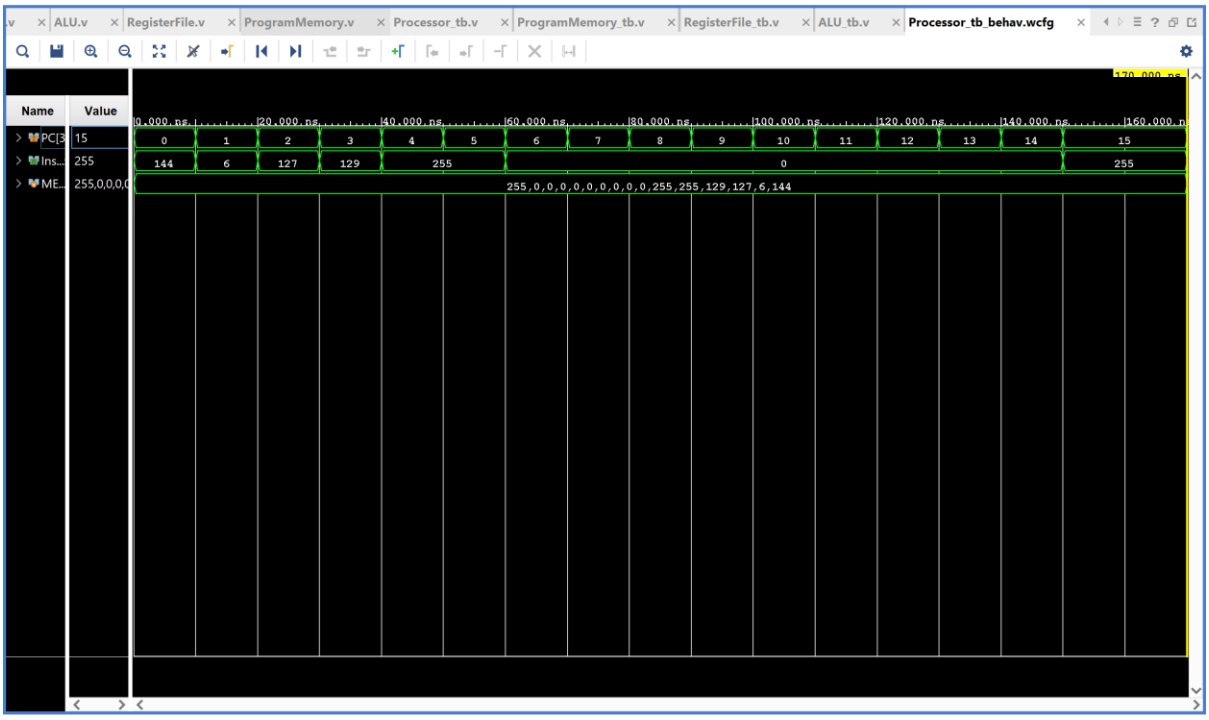
- Processor_tb.v

```
`timescale 1ns / 1ps

module Processor_tb;
```

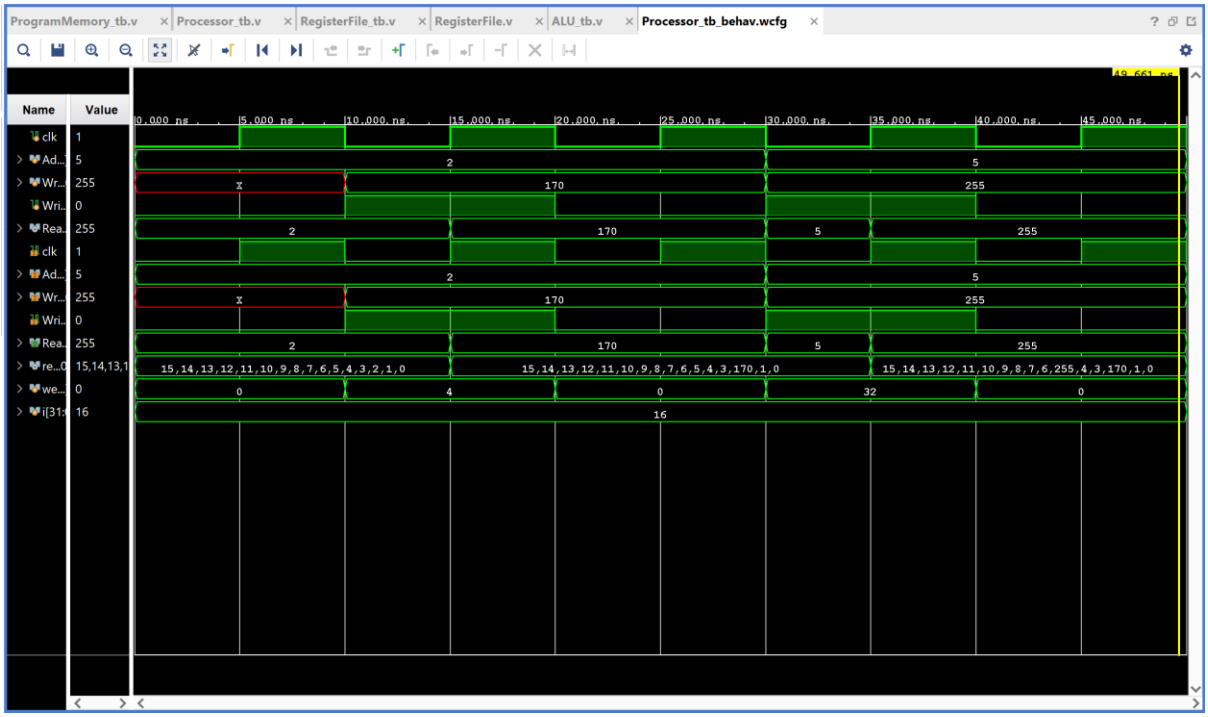
```
    reg clk;
    reg rstn;
    wire [7:0] ACC;

    Processor uut (
        .clk(clk),
        .rstn(rstn),
        .ACC(ACC)
    );

    initial begin
        $display("Time\tPC\tInstruction\tACC\tOutput");
    end

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

    // Reset and run
    initial begin
        rstn = 0; #12;
        rstn = 1;
    end

    // Stop simulation after instruction 18 (HLT)
    initial begin
        #300;
        $finish;
    end
endmodule
```

- # Simulation Results of Individual Components

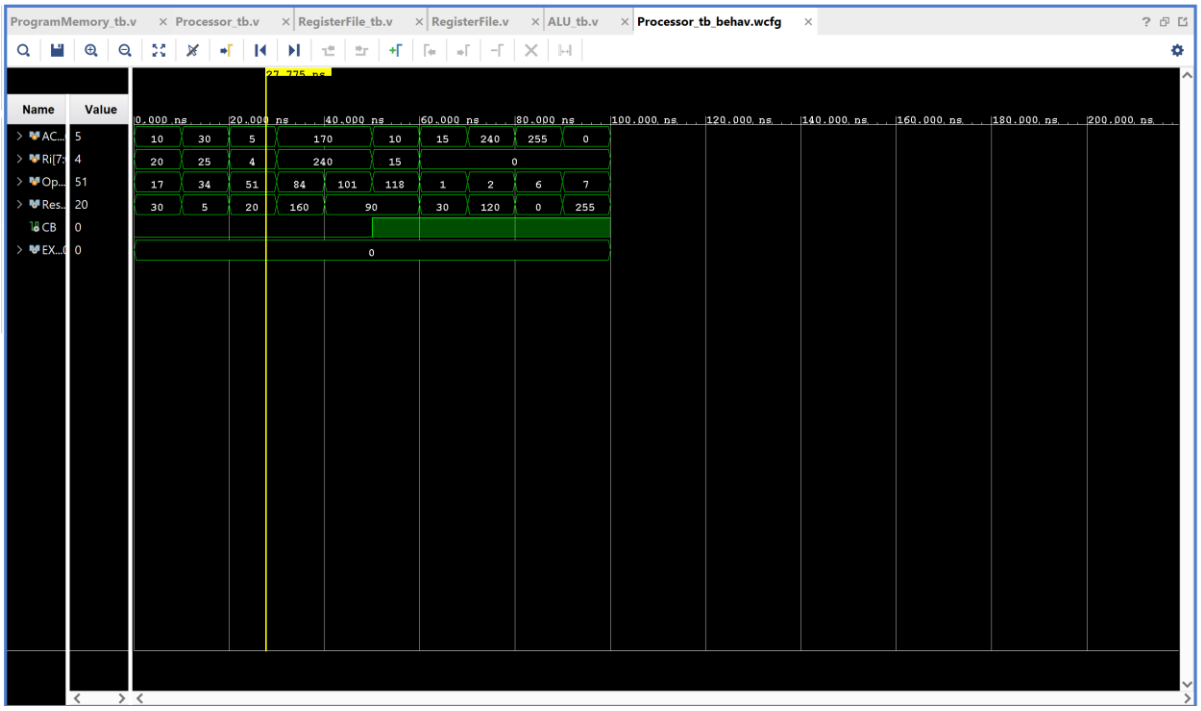
1. ProgramMemory_tb



2. RegisterFile_tb

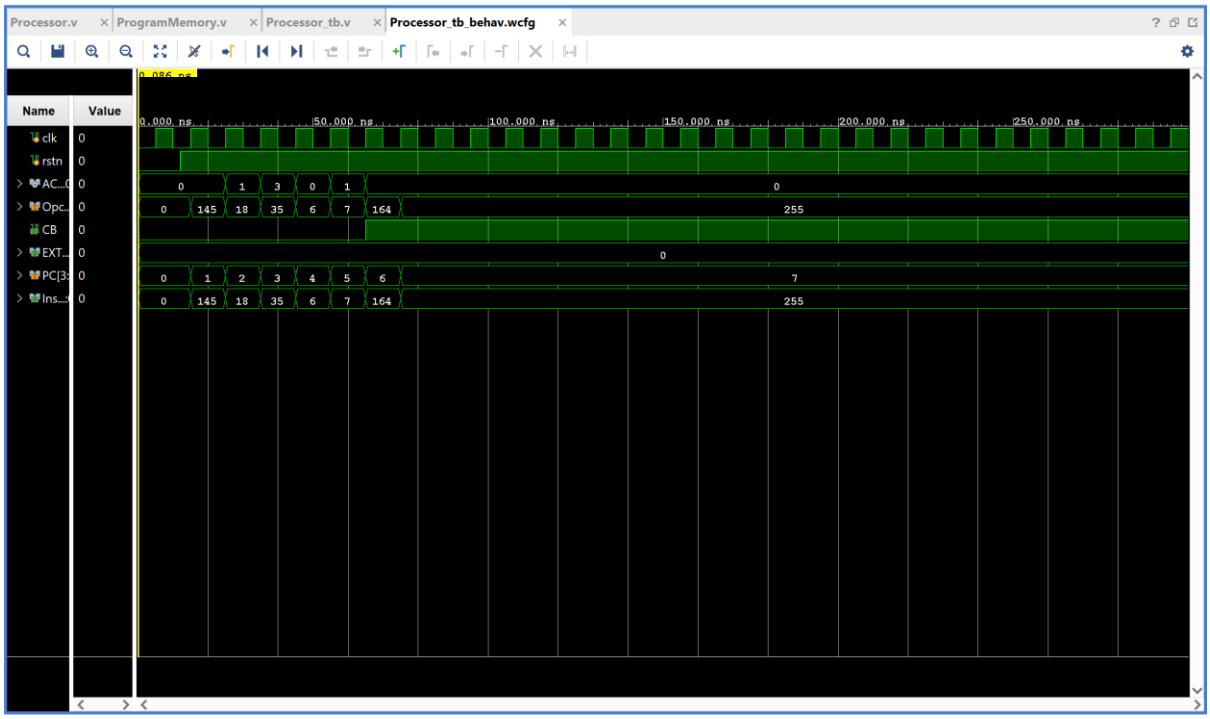


3. ALU_tb

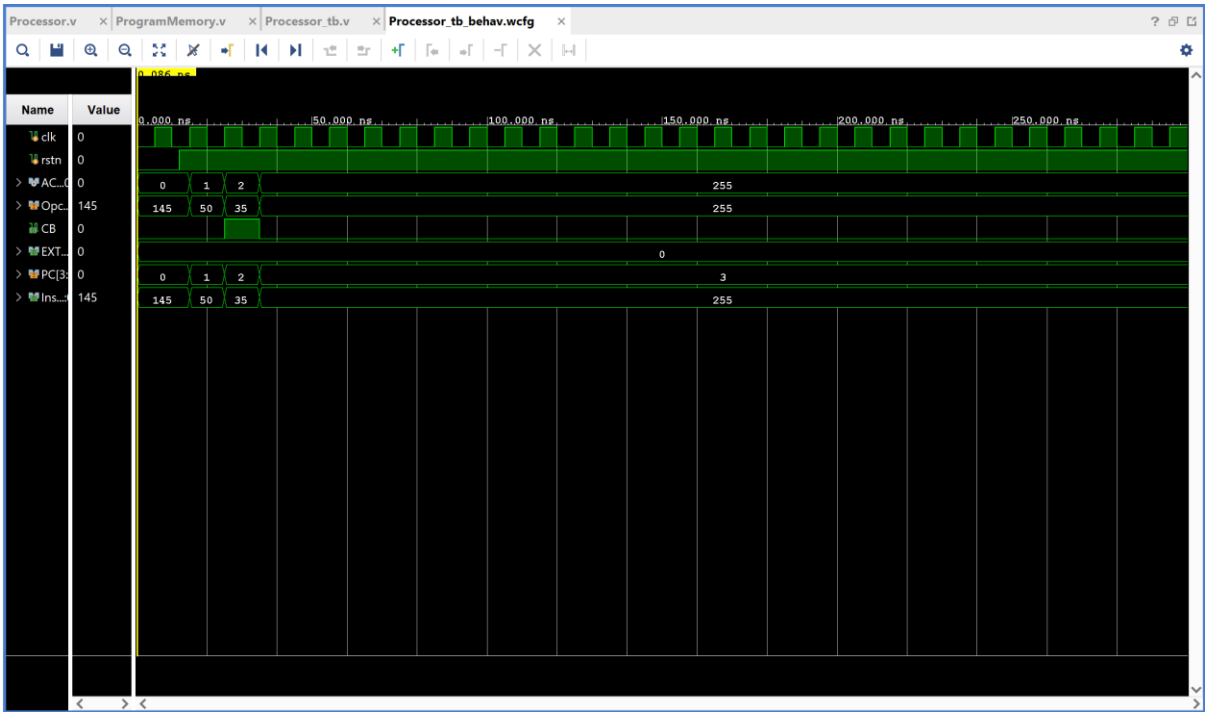


- # Simulation Results of Processor

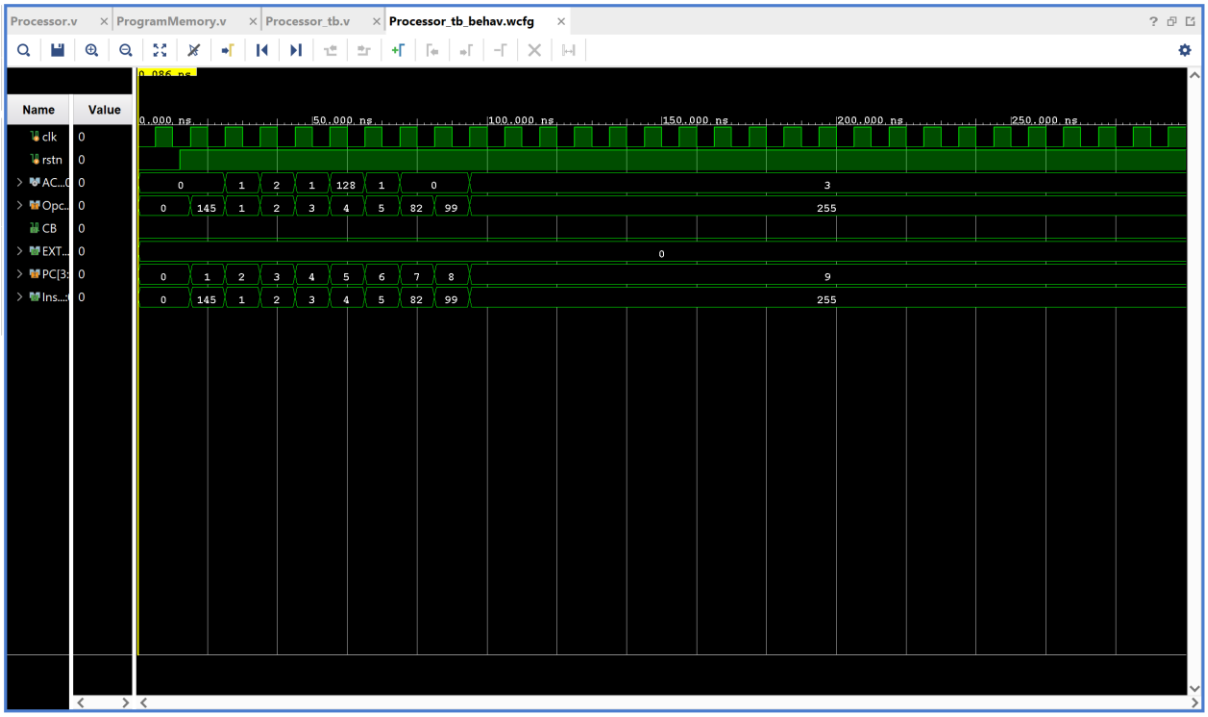
1. ADD, SUB, INC, DEC, MOV, NOP, HLT



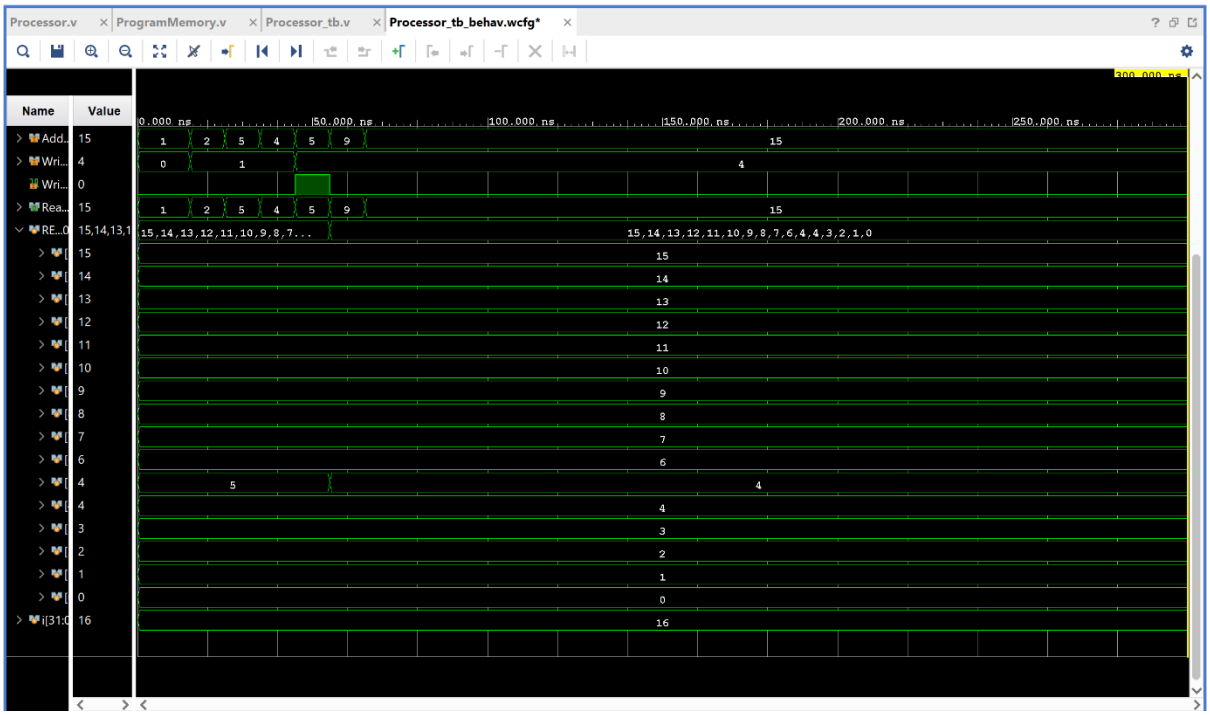
2. MUL, SUB, HLT



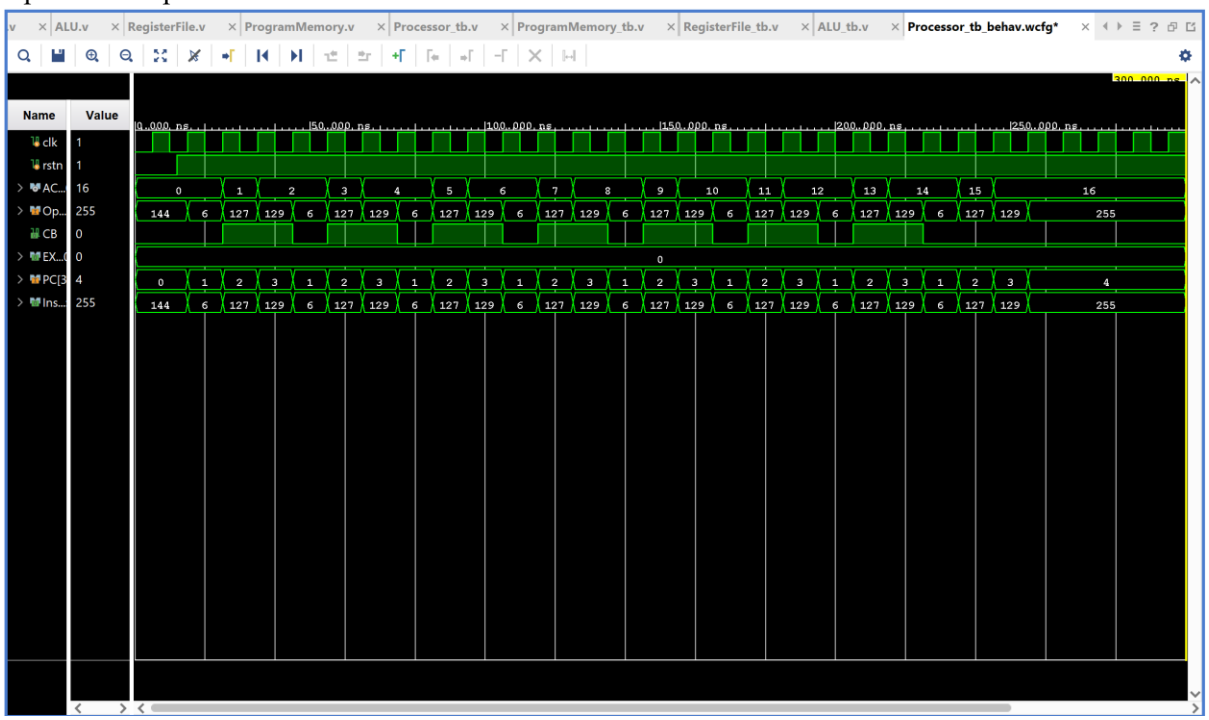
3. All shifts + AND, XRA, NOP, HLT



4. CMP, Br, Ret, MOV, HLT



5. Up Counter upto 15



- Block Diagram