Take Home 5 Assignment

Verilog code for Tiny Processor including testbench and simulation results

Processor Code:

```
`timescale 1ns/1ps

module register_file(
    input wire clock,
    input wire write_enable,
    input wire [3:0] write_address,
    input wire [7:0] write_data,
    input wire [3:0] read_address,
    output wire [7:0] read_data
);
    reg [7:0] registers [0:15];
    assign read_data = registers[read_address];

    always @(posedge clock) begin
        if (write_enable)
            registers[write_address] <= write_data;
    end
endmodule

module alu(
    input wire [7:0] accumulator,
    input wire [7:0] register_data,
    input wire [3:0] opcode,
    input wire [3:0] function_code,
    output reg [7:0] result_accumulator,
    output reg [7:0] result_ext,
    output reg result_carry_borrow
);
    reg [15:0] multiplication_result;

    always @(*) begin
        result_accumulator = accumulator;
        result_ext = 8'b0;
        result_carry_borrow = 1'b0;
        multiplication_result = 16'b0;

        case (opcode)
            4'b0000: begin
                case (function_code)
                    4'b0001: result_accumulator = {accumulator[6:0], 1'b0};  // LSL
                    4'b0010: result_accumulator = {1'b0, accumulator[7:1]};   // LSR
```

```
                4'b0011: result_accumulator = {accumulator[0], accumulator[7:1]};  // CIR
                4'b0100: result_accumulator = {accumulator[6:0], accumulator[7]};  // CIL
                4'b0101: result_accumulator = {accumulator[7], accumulator[7:1]};  // ASR
                4'b0110: {result_carry_borrow, result_accumulator} = accumulator + 8'b1;  // INC
                4'b0111: {result_carry_borrow, result_accumulator} = accumulator - 8'b1;  // DEC
            endcase
        end
        4'b0001: {result_carry_borrow, result_accumulator} = accumulator + register_data;  //
ADD
        4'b0010: {result_carry_borrow, result_accumulator} = accumulator - register_data;  //
SUB
        4'b0011: begin  // MUL
            multiplication_result = accumulator * register_data;
            {result_ext, result_accumulator} = multiplication_result;
        end
        4'b0101: result_accumulator = accumulator & register_data;  // AND
        4'b0110: result_accumulator = accumulator ^ register_data;  // XRA
        4'b0111: result_carry_borrow = (accumulator < register_data) ? 1'b1 : 1'b0;  // CMP
        4'b1001: result_accumulator = register_data;  // MOV ACC, Ri
        default: result_accumulator = accumulator;
      endcase
   end
endmodule

module control_unit(
   input wire [7:0] instruction,
   input wire carry_borrow_flag,
   output reg [3:0] alu_opcode,
   output reg [3:0] alu_function,
   output reg program_counter_update,
   output reg [3:0] program_counter_next,
   output reg accumulator_update,
   output reg ext_register_update,
   output reg carry_borrow_update,
   output reg register_write_enable,
   output reg halt_processor
);
   wire [3:0] opcode = instruction[7:4];
   wire [3:0] operand = instruction[3:0];

   always @(*) begin
      // Default values
      alu_opcode = opcode;
      alu_function = operand;
```

```
        program_counter_update = 1'b0;
        program_counter_next = 4'b0;
        accumulator_update = 1'b0;
        ext_register_update = 1'b0;
        carry_borrow_update = 1'b0;
        register_write_enable = 1'b0;
        halt_processor = 1'b0;

        case (opcode)
            4'b0000: begin
                case (operand)
                    4'b0000: ; // NOP
                    4'b0001,4'b0010,4'b0011,4'b0100,4'b0101: accumulator_update = 1'b1;
                    4'b0110,4'b0111: {accumulator_update, carry_borrow_update} = 2'b11;
                endcase
            end
            4'b0001,4'b0010: {accumulator_update, carry_borrow_update} = 2'b11; // ADD/SUB
            4'b0011: {accumulator_update, ext_register_update} = 2'b11; // MUL
            4'b0101,4'b0110: accumulator_update = 1'b1; // AND/XRA
            4'b0111: carry_borrow_update = 1'b1; // CMP
            4'b1000: begin // Branch
                if (carry_borrow_flag) begin
                    program_counter_update = 1'b1;
                    program_counter_next = operand;
                end
            end
            4'b1001: accumulator_update = 1'b1; // MOV ACC,Ri
            4'b1010: register_write_enable = 1'b1; // MOV Ri,ACC
            4'b1011: begin // Return
                program_counter_update = 1'b1;
                program_counter_next = operand;
            end
            4'b1111: halt_processor = (operand == 4'b1111); // HLT
        endcase
    end
endmodule

module Processor(
    input wire clock,
    input wire reset,
    output wire [7:0] acc,
    output wire [7:0] ext,
    output wire c_b_f,
    output wire [3:0] p_c
```

```
);
    reg [7:0] instruction_memory [0:15];
    reg [7:0] accumulator;
    reg [7:0] ext_register;
    reg carry_borrow_flag;
    reg [3:0] program_counter;
    reg halt;

    wire [7:0] current_instruction = instruction_memory[program_counter];
    wire [3:0] alu_opcode, alu_function;
    wire program_counter_update;
    wire [3:0] program_counter_next;
    wire accumulator_update, ext_register_update, carry_borrow_update;
    wire register_write_enable, halt_processor;
    wire [7:0] result_accumulator, result_ext;
    wire result_carry_borrow;
    wire [7:0] register_data;

    assign acc = accumulator;
    assign ext = ext_register;
    assign c_b_f = carry_borrow_flag;
    assign p_c = program_counter;

    register_file reg_file(
        .clock(clock),
        .write_enable(register_write_enable),
        .write_address(current_instruction[3:0]),
        .write_data(accumulator),
        .read_address(current_instruction[3:0]),
        .read_data(register_data)
    );

    alu alu_unit(
        .accumulator(accumulator),
        .register_data(register_data),
        .opcode(alu_opcode),
        .function_code(alu_function),
        .result_accumulator(result_accumulator),
        .result_ext(result_ext),
        .result_carry_borrow(result_carry_borrow)
    );

    control_unit ctrl_unit(
        .instruction(current_instruction),
```

```
        .carry_borrow_flag(carry_borrow_flag),
        .alu_opcode(alu_opcode),
        .alu_function(alu_function),
        .program_counter_update(program_counter_update),
        .program_counter_next(program_counter_next),
        .accumulator_update(accumulator_update),
        .ext_register_update(ext_register_update),
        .carry_borrow_update(carry_borrow_update),
        .register_write_enable(register_write_enable),
        .halt_processor(halt_processor)
    );

    always @(posedge clock or posedge reset) begin
        if (reset) begin
            accumulator <= 8'b0;
            ext_register <= 8'b0;
            carry_borrow_flag <= 1'b0;
            program_counter <= 4'b0;
            halt <= 1'b0;
        end else if (!halt) begin
            if (accumulator_update)
                accumulator <= result_accumulator;
            if (ext_register_update)
                ext_register <= result_ext;
            if (carry_borrow_update)
                carry_borrow_flag <= result_carry_borrow;

            program_counter <= program_counter_update ?
                        program_counter_next :
                        program_counter + 1;
            halt <= halt_processor;
        end
    end
endmodule
```

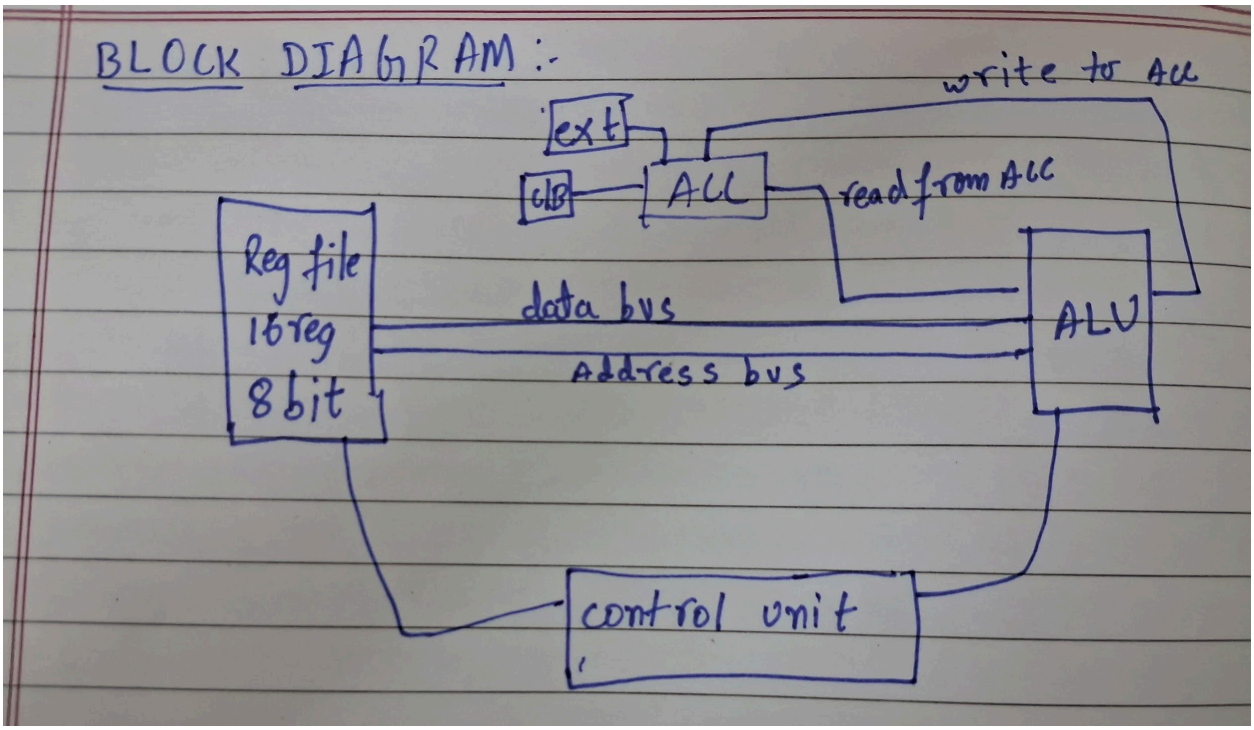
Testbench:
This program uses most of the instructions, so we have written only one testbench

```
`timescale 1ns/1ps

module Processor_tb;

    reg clock;
    reg reset;
```

```
wire [7:0] acc;
wire [7:0] ext;
wire c_b_f;
wire [3:0] pc;

Processor uut (
    .clock(clock),
    .reset(reset),
    .acc(acc),
    .ext(ext),
    .c_b_f(c_b_f),
    .p_c(pc)
);

always #5 clock = ~clock;

initial begin
    // Initialize instruction memory with test program
    uut.instruction_memory[0]  = 8'b10010001; // MOV ACC, R1
    uut.instruction_memory[1]  = 8'b00010001; // ADD R1
    uut.instruction_memory[2]  = 8'b00100010; // SUB R2
    uut.instruction_memory[3]  = 8'b00110011; // MUL R3
    uut.instruction_memory[4]  = 8'b00000001; // LSL ACC
    uut.instruction_memory[5]  = 8'b00000010; // LSR ACC
    uut.instruction_memory[6]  = 8'b00000011; // CIR ACC
    uut.instruction_memory[7]  = 8'b00000100; // CIL ACC
    uut.instruction_memory[8]  = 8'b00000101; // ASR ACC
    uut.instruction_memory[9]  = 8'b01010100; // AND R4
    uut.instruction_memory[10] = 8'b01100101; // XRA R5
    uut.instruction_memory[11] = 8'b01110110; // CMP R6
    uut.instruction_memory[12] = 8'b00000110; // INC ACC
    uut.instruction_memory[13] = 8'b00000111; // DEC ACC
    uut.instruction_memory[14] = 8'b10100111; // MOV R7, ACC
    uut.instruction_memory[15] = 8'b11111111; // HLT

    // Initialize registers
    uut.reg_file.registers[1] = 8'h02;
    uut.reg_file.registers[2] = 8'h01;
    uut.reg_file.registers[3] = 8'h03;
    uut.reg_file.registers[4] = 8'h0F;
    uut.reg_file.registers[5] = 8'hAA;
    uut.reg_file.registers[6] = 8'h10;

    // Initialize
```

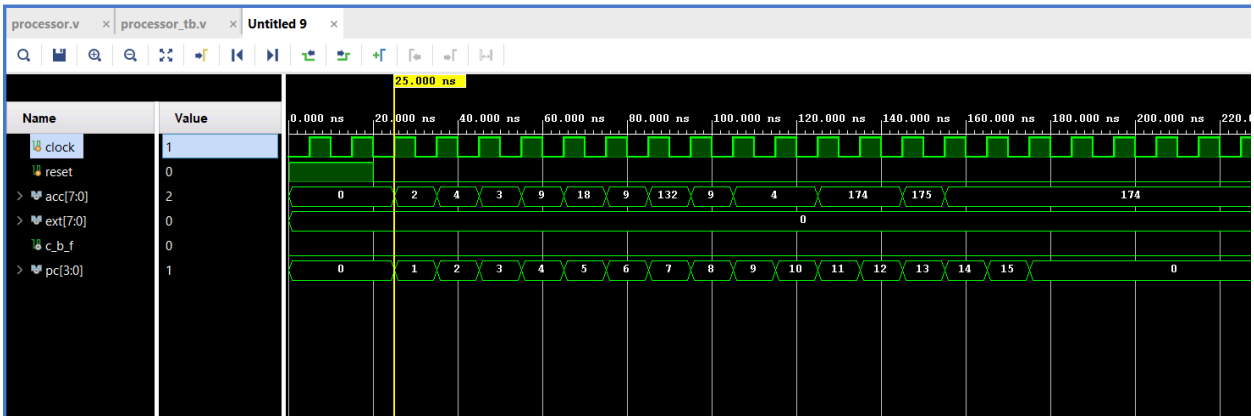
```
        clock = 0;
        reset = 1;
        #20;
        reset = 0;

        // Run simulation
        #500;
        $stop;
    end
endmodule
```

Block Diagram:



Simulation:

Take Home 5 Assignment



Schematic: