International University

School of Computer Science & Engineering

FINAL PROJECT

16-Bit Single Cycle Processor on FBGA

Submitted by

Truong Buu Duy – ITITIU21188

Date Submitted:	03/01/2025
Course Instructor:	Dr. Vo Minh Thanh

GRADING GUIDELINE FOR LAB REPORT

Number	Content		Score	Comment
1	**Format (max 9%)**			
	- Font type	Yes No		
	- Font size	Yes No		
	- Lab title	Yes No		
	- Page number	Yes No		
	- Table of contents	Yes No		
	- Header/Footer	Yes No		
	- List of figures (if exists)	Yes No		
	- List of tables (if exists)	Yes No		
	- Lab report structure	Yes No		
2	**English Grammar and Spelling (max 6%)**			
	- Grammar	Yes No		
	- Spelling	Yes No		
3	**Data and Result Analysis (max 85%)**			
Total Score				

Signature:

Date:

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

PC	Program Counter
ALU	Algorithm Logic Uni
Mux	Multiplexer
Clk	Clock

1 THEORETICAL BACKGROUND

1.1 Program Counter

This module's purpose is to determine whether the cycle operates using a new input signal or resets the system input to 0.

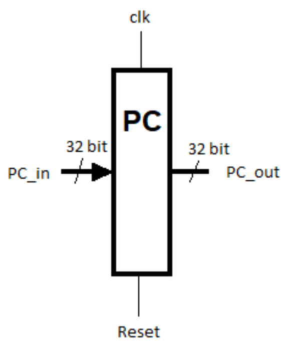


Figure 1.1 Program Counter Module

1.2 ADD

This module is used to sum up 2 input signals which have the same number of bits.

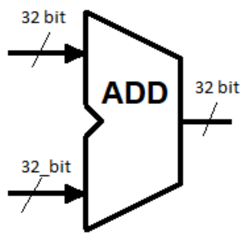


Figure 1.2 ADD Module

1.3 Instruction Memory

This module provides instructions to other components based on the input signal. The system generates a 5-bit signal for the memory register and controller and a 16-bit signal for the adder ALU operator. Instructions indexed 0-5 are sent to the ALU operator controller, while instructions indexed 0-15 pass through a signal extender to the ALU adder. Instructions indexed 16-20 pass through a MUX with indices 11-15 before reaching Read Register 2. Instructions indexed 21-25 are directed to Read Register 1.

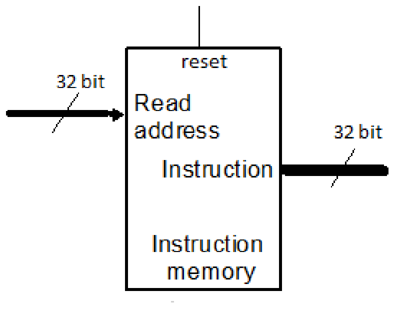


Figure 1.3 Instruction Memory Module

1.4 ALU

This module receives the operating instruction from operator wire and 2 input signals as parameters. Output of this module is the result of the operation.

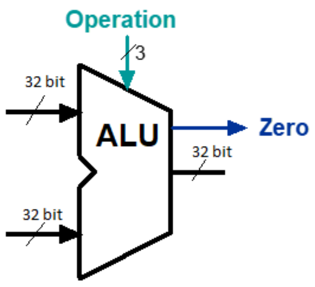


Figure 1.4 ALU Module

1.5 Mux

This module is to choose the output signal, which is one of the input signals.

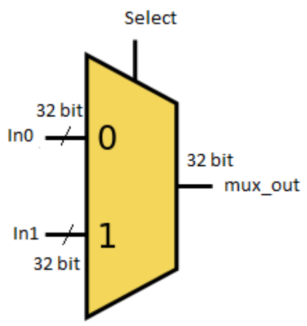


Figure 1.5 Mux Module

1.6 Sign Extend

To covert from 16 bits signal to 32 bits signal, we use signal extender. The extending bits equal the highest bit of the 16 bits signal.

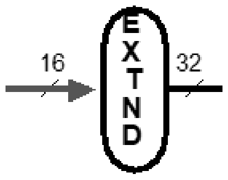


Figure 1.6 Sign Extend Module

1.7 Register File

This module has to role of operating to data memory. The receive instruction would come in from the RN1, RN2 and WN port with 5 bits signals. RegWrite to enable writing data 32 bits signal received 32 bits signal from WD port into a specific address provided by WN.

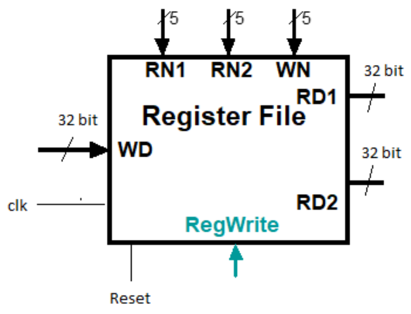


Figure 1.7 Register File Module

1.8 Data Memory

The operator is used to access memory data with the granted address of ADDR port. MemWrite and MemRead would enable/disable reading and writing process of Memory data. At the given address, the data may be export through RD and replace with the input data from WD.

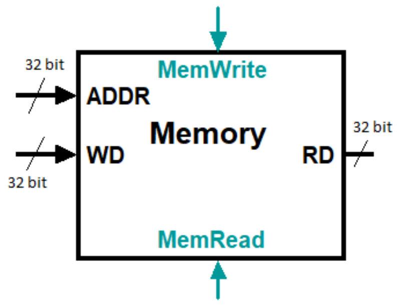


Figure 1.8 Data Memory Module

1.9 Instructions

Table 1.1 R-Type Instructions

Instruction	Operation	Function
add rd,rs,rt	rd = rs + rt	100000
sub rd,rs,rt	rd = rs – rt	100010
and rd,rs,rt	rd = rs & rt	100100

or rd,rs,rt	rd = rs | rt	100101
nor rd,rs,rt	rd = ~(rs|rt)	100111
xor rd,rs,rt	rd = rs ^ rt	100110
nand rd,rs,rt	rd = ~(rs & rt)	101000
xnor rd,rs,rt	rd = ~(rs ^ rt)	101010
sll rd,rt,shamt	rd = rt << shamt	000000
srl rd,rt,shamt	rd = rt >> shamt	000010
ror rd,rt,rs	rd = (rt >> shamt) | (rt << (32 - shamt))	110000
rol rd,rt,rs	rd = (rt << shamt) | (rt >> (32 - shamt))	111000

Table 1.2 I-Type Instructions

Instruction	Operation	Opcode
addi rt,rs,imm	rt = rs * imm	001000
beq rs,rt,imm	if (rs == rt) then branch	000100
bne rs,rt,imm	if !(rs==rt) then branch	000101
lw rt,offset(rs)	rt = mem[offset + rs]	100011
sw rt,offset(rs)	mem[offset + rs] = rt	101011

Table 1.3 J-Type Instructions

Instruction	Operation	Opcode
j address	jump to the instruction with respect address	000010
jal target	jump to the instruction at the specified address and save the return address in the $ra (register 31), which allows returning to the next instruction after the jump.	000011

2 EXPERIMENTAL PROCEDURE AND RESULTS

2.1 Data Memory

Verilog code

```
//Data Memory 16 bit
module Data_Memory_16bit (clk,addr,write_data,read_data,MemRead,MemWrite);
        input clk;
        input [15:0] addr;
        input [15:0] write_data;
        output reg [15:0] read_data;
        input MemRead;
        input MemWrite;
        integer i;
        reg [15:0] DMemory [0:255];

        initial begin
                for (i = 0; i < 256; i = i + 1) begin
                        DMemory[i] = 16'h0000;
                end
        end

        always @(posedge clk) begin
                if (MemWrite) begin
                        DMemory[addr] <= write_data;
                        $display("Memory Write: Address=%h Data=%h", addr, write_data);
                end
        end

        always @(*) begin
                if (MemRead)
                        read_data = DMemory[addr];
                else
                        read_data = 16'h0000;
        end
endmodule
```

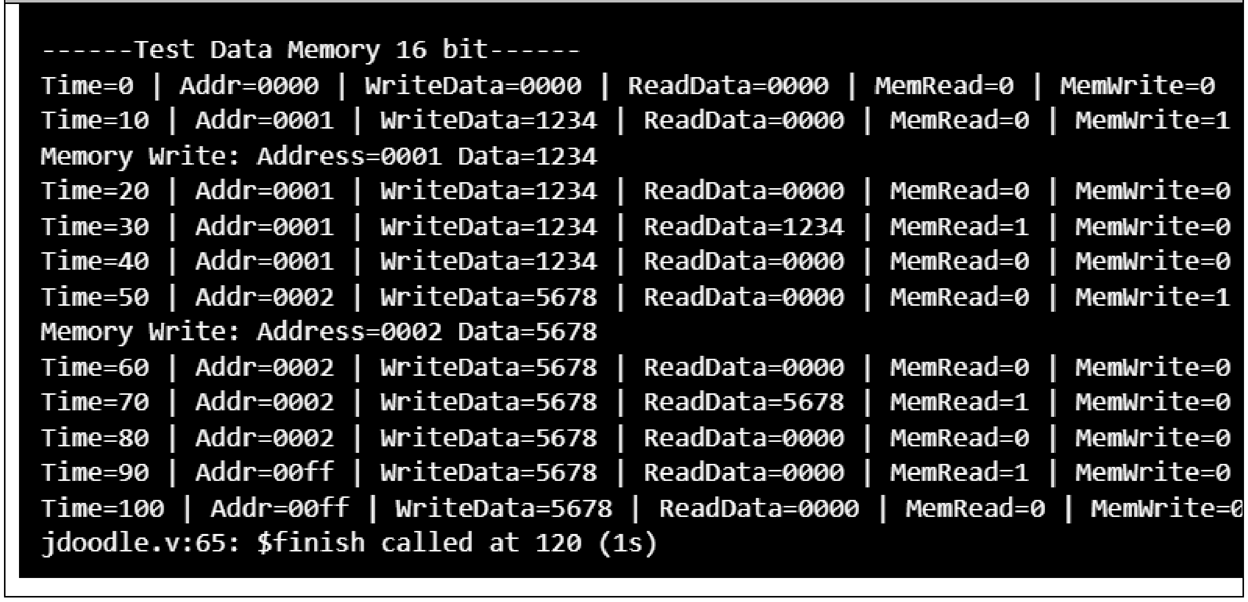
Testbench

```
// Testbench
module tb_Data_Memory_16bit;

   reg clk,MemRead,MemWrite;
   reg [15:0] addr,write_data;

   wire [15:0] read_data;

   Data_Memory_16bit
uut(.clk(clk),.addr(addr),.write_data(write_data),.read_data(read_data),.MemRead(MemRead)
,.MemWrite(MemWrite));

   always #5 clk = ~clk;

   initial begin
      clk = 0;
      addr = 16'h0000;
      write_data = 16'h0000;
      MemRead = 0;
      MemWrite = 0;

      $display("\n------Test Data Memory 16 bit------");

      $monitor("Time=%0t | Addr=%h | WriteData=%h | ReadData=%h | MemRead=%b |
MemWrite=%b",
            $time, addr, write_data, read_data, MemRead, MemWrite);
```

```
// Write data to memory
#10;
addr = 16'h0001;
write_data = 16'h1234;
MemWrite = 1;
MemRead = 0;
#10;
MemWrite = 0; // Stop write

// Read data from memory
#10;
addr = 16'h0001;
MemRead = 1;
#10;
MemRead = 0; // Stop read

// Write
#10;
addr = 16'h0002;
write_data = 16'h5678;
MemWrite = 1;
#10;
MemWrite = 0;

// Read
#10;
addr = 16'h0002;
MemRead = 1;
#10;
MemRead = 0;
```

```
    // Test reading an uninitialized memory location
    #10;
    addr = 16'h00FF;
    MemRead = 1;
    #10;
    MemRead = 0;



    #20;
    $finish;
  end

endmodule
```

Result



```
------Test Data Memory 16 bit------
Time=0 | Addr=0000 | WriteData=0000 | ReadData=0000 | MemRead=0 | MemWrite=0
Time=10 | Addr=0001 | WriteData=1234 | ReadData=0000 | MemRead=0 | MemWrite=1
Memory Write: Address=0001 Data=1234
Time=20 | Addr=0001 | WriteData=1234 | ReadData=0000 | MemRead=0 | MemWrite=0
Time=30 | Addr=0001 | WriteData=1234 | ReadData=1234 | MemRead=1 | MemWrite=0
Time=40 | Addr=0001 | WriteData=1234 | ReadData=0000 | MemRead=0 | MemWrite=0
Time=50 | Addr=0002 | WriteData=5678 | ReadData=0000 | MemRead=0 | MemWrite=1
Memory Write: Address=0002 Data=5678
Time=60 | Addr=0002 | WriteData=5678 | ReadData=0000 | MemRead=0 | MemWrite=0
Time=70 | Addr=0002 | WriteData=5678 | ReadData=5678 | MemRead=1 | MemWrite=0
Time=80 | Addr=0002 | WriteData=5678 | ReadData=0000 | MemRead=0 | MemWrite=0
Time=90 | Addr=00ff | WriteData=5678 | ReadData=0000 | MemRead=1 | MemWrite=0
Time=100 | Addr=00ff | WriteData=5678 | ReadData=0000 | MemRead=0 | MemWrite=0
jdoodle.v:65: $finish called at 120 (1s)
```

2.2 Instruction Memory

Verilog code
// Instruction Memory

```
module Instruction_Memory (read_address, instruction, reset);
        input reset;
        input [31:0] read_address;
        output [31:0] instruction;
        reg [31:0] Imemory [256:0];
        integer i;
        assign instruction = Imemory[read_address];
        always @(posedge reset)
        begin
        for (i=0; i<32; i=i+1)
                begin
                  Imemory[i] = 32'b0;
                end
                /*Test R-type*/
                // add $s6, $s5, $s4          R22 = R21 + R20 = 0x7 (0x5 + 0x2)
                Imemory[0] = 32'b000000_10101_10100_10110_00000_100000;
                // sub $s6, $s5, $s4          R22 = R21 - R20 = 0x3 (0x5 - 0x2)
                Imemory[1] = 32'b000000_10101_10100_10110_00000_100010;
                // mult $s5, $s4              R21 * R20 = 0xA (0x5 * 0x2)
                Imemory[2] = 32'b000000_10101_10100_00000_00000_011000;
                // div $s5, $s4               R21 / R20 = 0x2 (0x5 / 0x2)
                Imemory[3] = 32'b000000_10101_10100_00000_00000_011010;
                // and $s6, $s5, $s4          R22 = AND(R21, R20) = 0x0 (0x5 AND 0x2)
                Imemory[4] = 32'b000000_10101_10100_10110_00000_100100;
                // or $s6, $s5, $s4           R22 = OR(R21, R20) = 0x7 (0x5 OR 0x2)
                Imemory[5] = 32'b000000_10101_10100_10110_00000_100101;
                // xor $s6, $s5, $s4          R22 = XOR(R21, R20) = 0x7 (0x5 XOR 0x2)
                Imemory[6] = 32'b000000_10101_10100_10110_00000_100110;
                // nor $s6, $s5, $s4          R22 = NOR(R21, R20) = 0xFFFFFFF8 (NOR of 0x5
and 0x2)
                Imemory[7] = 32'b000000_10101_10100_10110_00000_100111;
                // nand $s6, $s5, $s4      R22 = NAND(R21, R20) = 0xFFFFFFFD (NAND of
0x5 and 0x2)
                Imemory[8] = 32'b000000_10101_10100_10110_00000_101000;
                // xnor $s6, $s5, $s4      R22 = XNOR(R21, R20) = 0xFFFFFFFE (XNOR of
0x5 and 0x2)
                Imemory[9] = 32'b000000_10101_10100_10110_00000_101010;
                // sll $s6, $s5, $s4       R22 = R21 << R20 = 0x14 (0x5 << 0x2)
                Imemory[10] = 32'b000000_10101_10100_10110_00000_000000;
                // srl $s6, $s5, $s4       R22 = R21 >> R20 = 0x1 (0x5 >> 0x2)
                Imemory[11] = 32'b000000_10101_10100_10110_00000_000010;
                // rol $s6, $s5            R22 = R21 rotated 1 bit left = 0xA (rotate 0x5 left by 1)
                Imemory[12] = 32'b000000_10101_00000_10110_00000_111000;
                // ror $s6, $s5            R22 = R21 rotated 1 bit right = 0x2 (rotate 0x5 right by
1)
                Imemory[13] = 32'b000000_10101_00000_10110_00000_110000;


                /*Test I-type*/
        // addi $s1, $s1, 0x05     R17 = 0x4E => R17 = R17 + 0x5 = 0x53
        Imemory[14] = 32'b001000_10001_10001_00000_00000_000101;
        // addi $s1, $s1, 0x05     R17 = 0x53 => R17 = R17 + 0x5 = 0x58
        Imemory[15] = 32'b001000_10001_10001_00000_00000_000101;
        // addi $s3, $s1, 0x10     R17 = 0x58 => R19 = R17 + 0x10 = 0x68
        Imemory[16] = 32'b001000_10001_10011_00000_00000_010000;
        // sw  $s3, 0x04($s1)      Memory[$s1 + 0x04] = $s3 => Memory[0x58 + 0x04] = 0x68
        Imemory[17] = 32'b101011_10001_10011_00000_00000_000100;
        // lw $s7, 0x04($s1)       $s7 = Memory[$s1 + 0x04] = 0x68
                Imemory[18] = 32'b100011_10001_10111_00000_00000_000100;
        // beq $s7, $s3, 0x08      R23 (0x68) == R19 (0x68) move to Imemory[22 + 8 + 1]
        Imemory[19] = 32'b000100_10111_10011_00000_00000_001000;
        // addi $s1, $zero, 0x11   R17 = 0x11
        Imemory[20] = 32'b001000_00000_10001_00000_00000_010001;
```

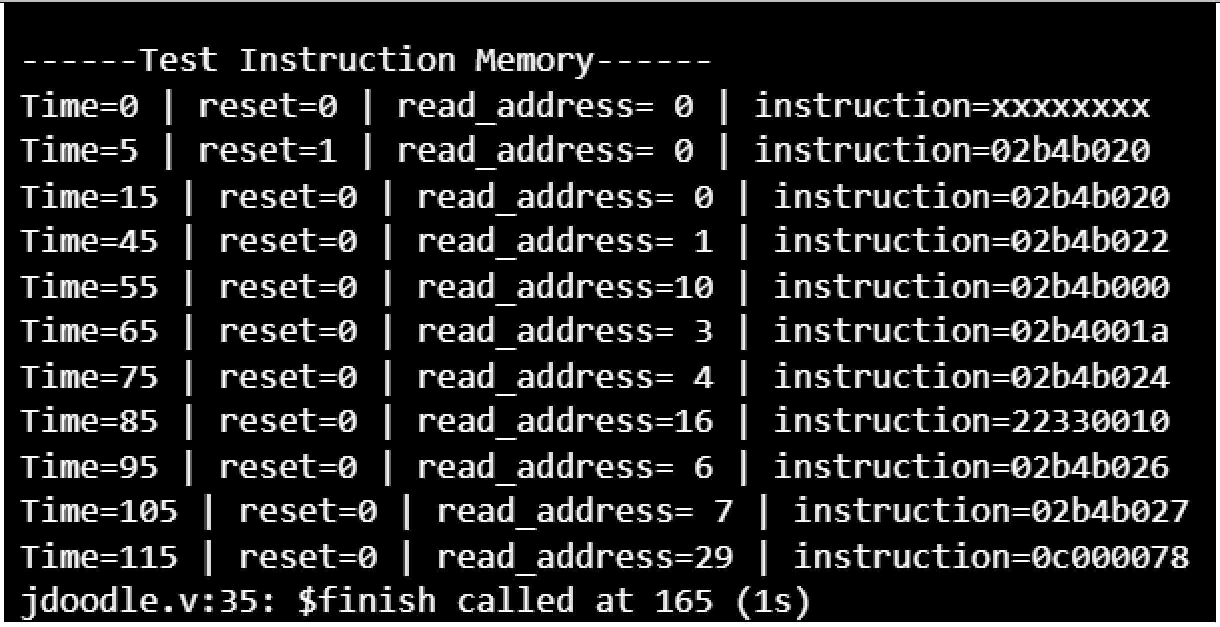
```
// addi $s1, $zero, 0x22      R17 = 0x22
Imemory[21] = 32'b001000_00000_10001_00000_00000_100010;
// addi $s1, $zero, 0x33      R17 = 0x33
Imemory[22] = 32'b001000_00000_10001_00000_00000_110011;
// bne $s1, $s7, 0x04        (R17 (0x33) != R23 (0x68)) move to Imemory[26 + 4 + 1]
Imemory[23] = 32'b000101_10001_10111_00000_00000_000100;
// addi $s2, $zero, 0x66      R18 = 0x66
Imemory[24] = 32'b001000_00000_10010_00000_00001_100110;
// addi $s2, $zero, 0x77      R18 = 0x77
Imemory[25] = 32'b001000_00000_10010_00000_00001_110111;

/* Test J-type  */
        // j 0x74 (JUMP to Imemory[116])
        Imemory[26] = 32'b000010_00000_00000_00000_00001_110100;
        // jal 0x74 (Jump and Link to Imemory[116])
        Imemory[27] = 32'b000011_00000_00000_00000_00001_110100;
        // j 0x78 (JUMP to Imemory[120])
        Imemory[28] = 32'b000010_00000_00000_00000_00001_111000;
        // jal 0x78 (Jump and Link to Imemory[120])
        Imemory[29] = 32'b000011_00000_00000_00000_00001_111000;
        // j 0x64 (JUMP to Imemory[100])
        Imemory[30] = 32'b000010_00000_00000_00000_00001_100100;
        // jal 0x68 (Jump and Link to Imemory[104])
        Imemory[31] = 32'b000011_00000_00000_00000_00001_101000;
        // j 0x6C (JUMP to Imemory[108])
        Imemory[32] = 32'b000010_00000_00000_00000_00001_101100;
        // jal 0x70 (Jump and Link to Imemory[112])
        Imemory[33] = 32'b000011_00000_00000_00000_00001_110000;
        // j 0x74 (JUMP to Imemory[116])
        Imemory[34] = 32'b000010_00000_00000_00000_00001_110100;
        // j 0x7C (JUMP to Imemory[124])
```

```
                        Imemory[35] = 32'b000010_00000_00000_00000_00001_111100;


            end
endmodule
```

Testbench

```
// Testbench
module tb_Instruction_Memory;
    reg reset;
    reg [31:0] read_address;
    wire [31:0] instruction;

    Instruction_Memory uut(.reset(reset),.read_address(read_address),.instruction(instruction));

    initial begin

            $display("\n------Test Instruction Memory------");

        $monitor("Time=%0d | reset=%b | read_address=%2d | instruction=%h", $time, reset,
read_address, instruction);

        reset = 0;
        read_address = 32'd0;

        // Test Case 1
        #5 reset = 1;
        #10 reset = 0;

        // Test Case 2
        #10 read_address = 32'd0;  // Address 0
        #10 read_address = 32'd0;  // Address 0
        #10 read_address = 32'd1;  // Address 1
        #10 read_address = 32'd10;  // Address 10
        #10 read_address = 32'd3;  // Address 3
        #10 read_address = 32'd4;  // Address 4
        #10 read_address = 32'd16;  // Address 16
        #10 read_address = 32'd6;  // Address 6
        #10 read_address = 32'd7;  // Address 7
        #10 read_address = 32'd29;  // Address 29



        #50 $finish;
    end
endmodule
```

Result



```
------Test Instruction Memory------
Time=0 | reset=0 | read_address= 0 | instruction=xxxxxxxx
Time=5 | reset=1 | read_address= 0 | instruction=02b4b020
Time=15 | reset=0 | read_address= 0 | instruction=02b4b020
Time=45 | reset=0 | read_address= 1 | instruction=02b4b022
Time=55 | reset=0 | read_address=10 | instruction=02b4b000
Time=65 | reset=0 | read_address= 3 | instruction=02b4001a
Time=75 | reset=0 | read_address= 4 | instruction=02b4b024
Time=85 | reset=0 | read_address=16 | instruction=22330010
Time=95 | reset=0 | read_address= 6 | instruction=02b4b026
Time=105 | reset=0 | read_address= 7 | instruction=02b4b027
Time=115 | reset=0 | read_address=29 | instruction=0c000078
jdoodle.v:35: $finish called at 165 (1s)
```

2.3 Program Counter

Verilog code

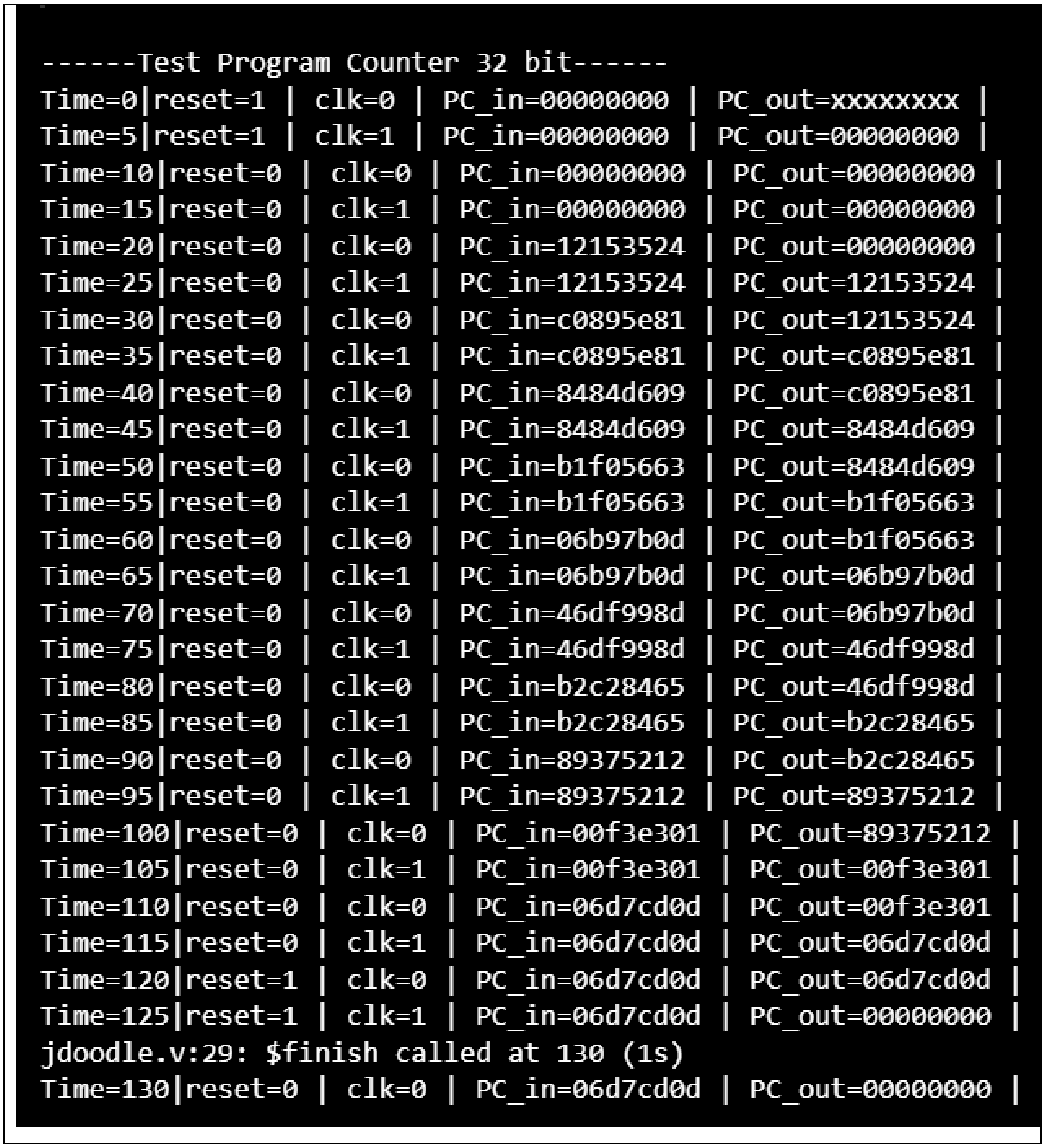
```
// Program Counter
```

```
module Program_Counter(clk, reset, PC_in, PC_out);
    input clk, reset;
    input [31:0] PC_in;
    output reg [31:0] PC_out;

    always @(posedge clk) begin
        if(reset)
            PC_out <= 32'b0;
        else
            PC_out <= PC_in;
    end
endmodule
```

Testbench

```
// Testbench
module tb_Program_Counter;
    reg clk, reset;
    reg [31:0] PC_in;
    wire [31:0] PC_out;

    Program_Counter  uut(.clk(clk),.reset(reset),.PC_in(PC_in),.PC_out(PC_out));

        always #5 clk = ~clk;

        initial begin
        clk = 0;
        reset = 1;
        PC_in = 32'd0;

            $display("\n------Test Program Counter 32 bit------");

        $monitor("Time=%0t|reset=%b | clk=%b | PC_in=%h | PC_out=%h |", $time,reset, clk,
PC_in, PC_out);

        #10 reset = 0;

        repeat(10) begin
            #10 PC_in = $random;
        end

        #10 reset = 1;
        #10 reset = 0;

        $finish;
    end
endmodule
```

Result



```
------Test Program Counter 32 bit------
Time=0|reset=1 | clk=0 | PC_in=00000000 | PC_out=xxxxxxxx |
Time=5|reset=1 | clk=1 | PC_in=00000000 | PC_out=00000000 |
Time=10|reset=0 | clk=0 | PC_in=00000000 | PC_out=00000000 |
Time=15|reset=0 | clk=1 | PC_in=00000000 | PC_out=00000000 |
Time=20|reset=0 | clk=0 | PC_in=12153524 | PC_out=00000000 |
Time=25|reset=0 | clk=1 | PC_in=12153524 | PC_out=12153524 |
Time=30|reset=0 | clk=0 | PC_in=c0895e81 | PC_out=12153524 |
Time=35|reset=0 | clk=1 | PC_in=c0895e81 | PC_out=c0895e81 |
Time=40|reset=0 | clk=0 | PC_in=8484d609 | PC_out=c0895e81 |
Time=45|reset=0 | clk=1 | PC_in=8484d609 | PC_out=8484d609 |
Time=50|reset=0 | clk=0 | PC_in=b1f05663 | PC_out=8484d609 |
Time=55|reset=0 | clk=1 | PC_in=b1f05663 | PC_out=b1f05663 |
Time=60|reset=0 | clk=0 | PC_in=06b97b0d | PC_out=b1f05663 |
Time=65|reset=0 | clk=1 | PC_in=06b97b0d | PC_out=06b97b0d |
Time=70|reset=0 | clk=0 | PC_in=46df998d | PC_out=06b97b0d |
Time=75|reset=0 | clk=1 | PC_in=46df998d | PC_out=46df998d |
Time=80|reset=0 | clk=0 | PC_in=b2c28465 | PC_out=46df998d |
Time=85|reset=0 | clk=1 | PC_in=b2c28465 | PC_out=b2c28465 |
Time=90|reset=0 | clk=0 | PC_in=89375212 | PC_out=b2c28465 |
Time=95|reset=0 | clk=1 | PC_in=89375212 | PC_out=89375212 |
Time=100|reset=0 | clk=0 | PC_in=00f3e301 | PC_out=89375212 |
Time=105|reset=0 | clk=1 | PC_in=00f3e301 | PC_out=00f3e301 |
Time=110|reset=0 | clk=0 | PC_in=06d7cd0d | PC_out=00f3e301 |
Time=115|reset=0 | clk=1 | PC_in=06d7cd0d | PC_out=06d7cd0d |
Time=120|reset=1 | clk=0 | PC_in=06d7cd0d | PC_out=06d7cd0d |
Time=125|reset=1 | clk=1 | PC_in=06d7cd0d | PC_out=00000000 |
jdoodle.v:29: $finish called at 130 (1s)
Time=130|reset=0 | clk=0 | PC_in=06d7cd0d | PC_out=00000000 |
```

2.4 Register File

Verilog code
See code below

```
// Register File 16bit
module Register_File_16bit (clk,read_addr_1, read_addr_2, write_addr, write_data,
RegWrite,read_data_1, read_data_2);
        input clk,RegWrite;
        input [4:0] read_addr_1, read_addr_2, write_addr;
        input [15:0] write_data;
        output [15:0] read_data_1, read_data_2;
        reg [15:0] Regfile [31:0];
        integer i;

        initial begin
                for (i = 0; i < 32; i = i + 1) begin
                        Regfile[i] = 16'b0;
                end
        Regfile[19]=1; // $s3 = 1
        Regfile[17]=3; // $s1 = 3
        Regfile[20]=2; // $s4 = 2
     Regfile[21]=5; // $s5 = 5
        end

        assign read_data_1 = Regfile[read_addr_1];
        assign read_data_2 = Regfile[read_addr_2];

        always @(posedge clk) begin
                if (RegWrite) begin
                        Regfile[write_addr] <= write_data;
                        $display("write_addr=%h write_data=%h",write_addr,write_data);
                end
        end
endmodule
```

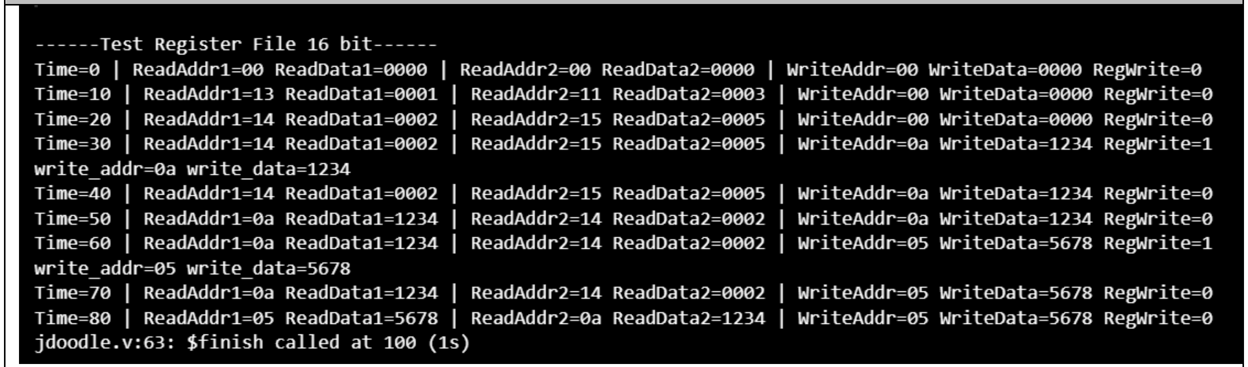
Testbench

```
// Testbench
module tb_Register_File_16bit;
    reg clk,RegWrite;
    reg [4:0] read_addr_1, read_addr_2, write_addr;
    reg [15:0] write_data;
    wire [15:0] read_data_1, read_data_2;



    Register_File_16bit uut
(.clk(clk),.read_addr_1(read_addr_1),.read_addr_2(read_addr_2),.write_addr(write_addr),.write_data(write_data),.RegWrite(RegWrite),.read_data_1(read_data_1),.read_data_2(read_data_2)
    );

       always #5 clk = ~clk;

    initial begin
        clk = 0;
        RegWrite = 0;
        read_addr_1 = 5'b0;
        read_addr_2 = 5'b0;
        write_addr = 5'b0;
        write_data = 16'b0;

            $display("\n------Test Register File 16 bit------");

        $monitor("Time=%0t | ReadAddr1=%h ReadData1=%h | ReadAddr2=%h ReadData2=%h | WriteAddr=%h WriteData=%h RegWrite=%b",
                 $time, read_addr_1, read_data_1, read_addr_2, read_data_2, write_addr, write_data, RegWrite);
```

```
// Read default values from registers
#10;
read_addr_1 = 5'd19; // Reading $s3
read_addr_2 = 5'd17; // Reading $s1

#10;
read_addr_1 = 5'd20; // Reading $s4
read_addr_2 = 5'd21; // Reading $s5

// Write to a register
#10;
write_addr = 5'd10; // Write to $t2
write_data = 16'h1234;
RegWrite = 1;
#10;
RegWrite = 0;

// Read back the written value
#10;
read_addr_1 = 5'd10; // Reading $t2
read_addr_2 = 5'd20; // Reading $s4

// Write to another register
#10;
write_addr = 5'd5; // Write to $a1
write_data = 16'h5678;
RegWrite = 1;
#10;
RegWrite = 0;
```

```
        // Read back the new values
        #10;
        read_addr_1 = 5'd5; // Reading $a1
        read_addr_2 = 5'd10; // Reading $t2

        #20;
        $finish;
    end

endmodule
```

Result



```
------Test Register File 16 bit------
Time=0 | ReadAddr1=00 ReadData1=0000 | ReadAddr2=00 ReadData2=0000 | WriteAddr=00 WriteData=0000 RegWrite=0
Time=10 | ReadAddr1=13 ReadData1=0001 | ReadAddr2=11 ReadData2=0003 | WriteAddr=00 WriteData=0000 RegWrite=0
Time=20 | ReadAddr1=14 ReadData1=0002 | ReadAddr2=15 ReadData2=0005 | WriteAddr=00 WriteData=0000 RegWrite=0
Time=30 | ReadAddr1=14 ReadData1=0002 | ReadAddr2=15 ReadData2=0005 | WriteAddr=0a WriteData=1234 RegWrite=1
write_addr=0a write_data=1234
Time=40 | ReadAddr1=14 ReadData1=0002 | ReadAddr2=15 ReadData2=0005 | WriteAddr=0a WriteData=1234 RegWrite=0
Time=50 | ReadAddr1=0a ReadData1=1234 | ReadAddr2=14 ReadData2=0002 | WriteAddr=0a WriteData=1234 RegWrite=0
Time=60 | ReadAddr1=0a ReadData1=1234 | ReadAddr2=14 ReadData2=0002 | WriteAddr=05 WriteData=5678 RegWrite=1
write_addr=05 write_data=5678
Time=70 | ReadAddr1=0a ReadData1=1234 | ReadAddr2=14 ReadData2=0002 | WriteAddr=05 WriteData=5678 RegWrite=0
Time=80 | ReadAddr1=05 ReadData1=5678 | ReadAddr2=0a ReadData2=1234 | WriteAddr=05 WriteData=5678 RegWrite=0
jdoodle.v:63: $finish called at 100 (1s)
```

2.5 Adder 32 Bit

Verilog code

```
// Adder 32 bit
module Adder32Bit(input1, input2, out);
    input [31:0] input1, input2;
    output [31:0] out;
    reg [31:0]out;
    always@( input1 or input2)
        begin
```

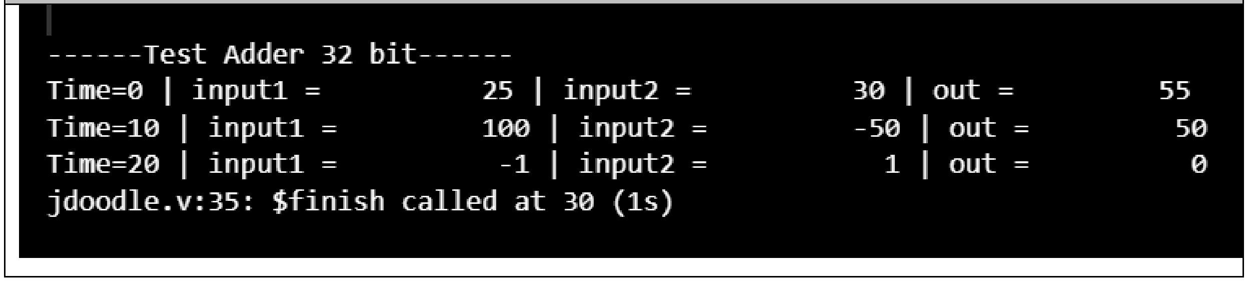
```
          out <= input1 + input2;
      end
endmodule
```

Testbench

```
module tb_Adder32Bit;

   reg signed [31:0] input1, input2;
   wire [31:0] out;

      Adder32Bit uut (.input1(input1),.input2(input2),.out(out));

   initial begin
          $display("------Test Adder 32 bit------");

      $monitor("Time=%0t | input1 = %d | input2 = %d | out = %d", $time, input1, input2, out);

      // Test case 1: Add two positive numbers
      input1 = 32'd25;
      input2 = 32'd30;
      #10;
      // Test case 2: Add a positive and a negative number
      input1 = 32'd100;
      input2 = -32'd50;
      #10;
      // Test case 3: Add two large numbers
      input1 = 32'd4294967295; // Max unsigned 32-bit value
      input2 = 32'd1;
      #10;

      $finish;
   end
```

```
endmodule
```

Result



```
------Test Adder 32 bit------
Time=0 | input1 =          25 | input2 =          30 | out =         55
Time=10 | input1 =         100 | input2 =         -50 | out =         50
Time=20 | input1 =          -1 | input2 =           1 | out =          0
jdoodle.v:35: $finish called at 30 (1s)
```

2.6 Mux N bit

Verilog code

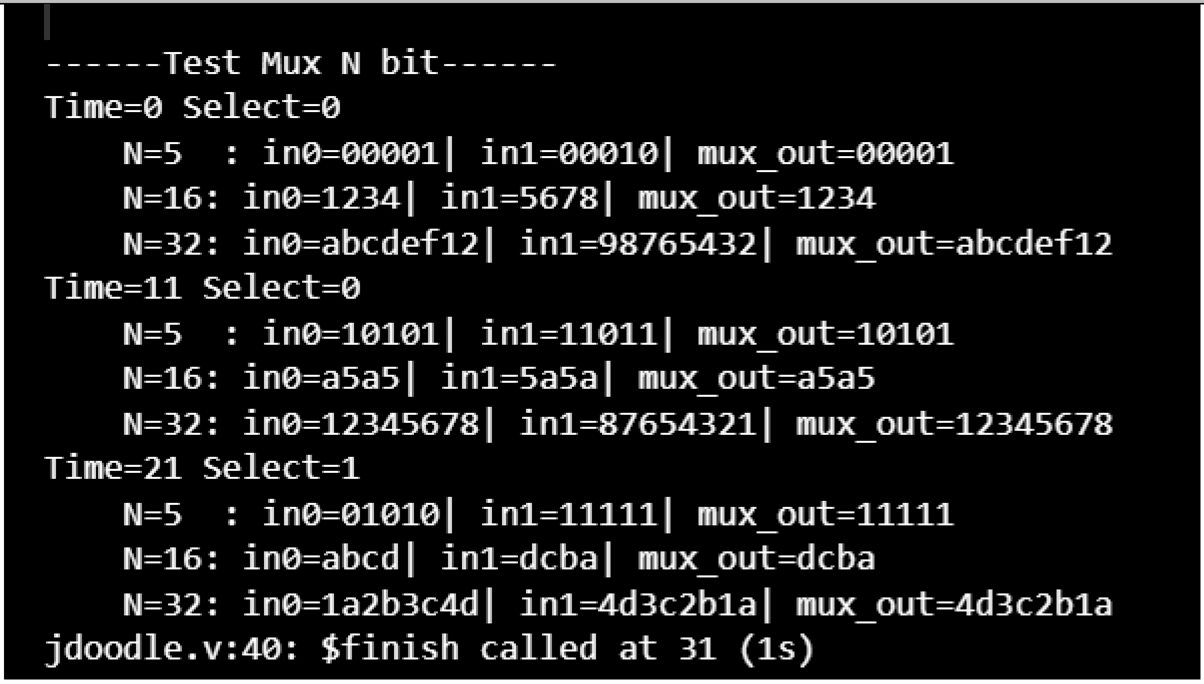
```
// Mux N bit
module Mux_N_bit (in0, in1, mux_out, select);
        parameter N=5;
        input [N-1:0] in0, in1;
        output [N-1:0] mux_out;
        input select;
        assign mux_out = select ? in1 : in0;
endmodule
```

Testbench

```
// Testbench
module tb_Mux_N_bit;
   reg [4:0] in0_5, in1_5;
   reg [15:0] in0_16, in1_16;
   reg [31:0] in0_32, in1_32;
   wire [4:0] mux_out_5;
   wire [15:0] mux_out_16;
   wire [31:0] mux_out_32;
   reg select;
```

```
    Mux_N_bit #(5) uut_5 (.in0(in0_5), .in1(in1_5), .mux_out(mux_out_5), .select(select));
    Mux_N_bit #(16) uut_16 (.in0(in0_16), .in1(in1_16), .mux_out(mux_out_16),
.select(select));
    Mux_N_bit #(32) uut_32 (.in0(in0_32), .in1(in1_32), .mux_out(mux_out_32),
.select(select));

    initial begin
        select = 0;
        in0_5 = 5'b00001;   in1_5 = 5'b00010;
        in0_16 = 16'h1234;  in1_16 = 16'h5678;
        in0_32 = 32'hABCDEF12; in1_32 = 32'h98765432;

        $display("------Test Mux N bit------");
        $monitor("Time=%0t Select=%b \n\tN=5   : in0=%b| in1=%b| mux_out=%b \n\tN=16:
in0=%h| in1=%h| mux_out=%h \n\tN=32: in0=%h| in1=%h| mux_out=%h",
            $time, select, in0_5, in1_5, mux_out_5, in0_16, in1_16, mux_out_16,  in0_32, in1_32,
mux_out_32);

        #1;

        #10;
        select = 0;
        in0_5 = 5'b10101; in1_5 = 5'b11011;
        in0_16 = 16'hA5A5; in1_16 = 16'h5A5A;
        in0_32 = 32'h12345678; in1_32 = 32'h87654321;

        #10;
        select = 1;
        in0_5 = 5'b01010; in1_5 = 5'b11111;
        in0_16 = 16'hABCD; in1_16 = 16'hDCBA;
        in0_32 = 32'h1A2B3C4D; in1_32 = 32'h4D3C2B1A;
        #10;
        $finish;
    end
endmodule
```

Result



```
------Test Mux N bit------
Time=0 Select=0
    N=5  : in0=00001| in1=00010| mux_out=00001
    N=16: in0=1234| in1=5678| mux_out=1234
    N=32: in0=abcdef12| in1=98765432| mux_out=abcdef12
Time=11 Select=0
    N=5  : in0=10101| in1=11011| mux_out=10101
    N=16: in0=a5a5| in1=5a5a| mux_out=a5a5
    N=32: in0=12345678| in1=87654321| mux_out=12345678
Time=21 Select=1
    N=5  : in0=01010| in1=11111| mux_out=11111
    N=16: in0=abcd| in1=dcba| mux_out=dcba
    N=32: in0=1a2b3c4d| in1=4d3c2b1a| mux_out=4d3c2b1a
jdoodle.v:40: $finish called at 31 (1s)
```

2.7 Sign Extend N bit to 32 bit

Verilog code

```
// Sign Extension N-bit
module Sign_Extension_N_bit (sign_in, sign_out);
    parameter N = 16;
    input [N-1:0] sign_in;
    output [31:0] sign_out;

    assign sign_out = {{(32-N){sign_in[N-1]}}, sign_in};
```

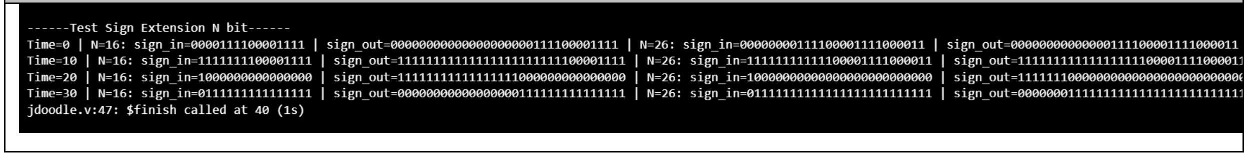
```
endmodule
```

Testbench

```
// Testbench
module tb_Sign_Extension_N_bit;

   reg [15:0] sign_in_16; // For N = 16
   reg [25:0] sign_in_26; // For N = 26
   wire [31:0] sign_out_16;
   wire [31:0] sign_out_26;

   // N = 16
   Sign_Extension_N_bit #(16) uut_16 (.sign_in(sign_in_16),.sign_out(sign_out_16));

   // N = 26
   Sign_Extension_N_bit #(26) uut_26 (.sign_in(sign_in_26),.sign_out(sign_out_26));

   initial begin
         $display("\n------Test Sign Extension N bit------");

      $monitor("Time=%0t | N=16: sign_in=%b | sign_out=%b | N=26: sign_in=%b |
sign_out=%b",
             $time, sign_in_16, sign_out_16, sign_in_26, sign_out_26);

      // Test Case 1: Positive input
      sign_in_16 = 16'b0000_1111_0000_1111;
      sign_in_26 = 26'b0000_0000_1111_0000_1111_0000_11;
      #10;

      // Test Case 2: Negative input
         sign_in_16 = 16'b1111_1111_0000_1111;
      sign_in_26 = 26'b1111_1111_1111_0000_1111_0000_11;
```

```
        #10;

        // Test Case 3: Smallest negative value
        sign_in_16 = 16'b1000_0000_0000_0000;
        sign_in_26 = 26'b1000_0000_0000_0000_0000_0000_00;
        #10;

        // Test Case 4: Largest positive value
            sign_in_16 = 16'b0111_1111_1111_1111;
        sign_in_26 = 26'b0111_1111_1111_1111_1111_1111_11;
        #10;

        $finish;
    end

endmodule
```

Result



```
------Test Sign Extension N bit------
Time=0 | N=16: sign_in=0000111100001111 | sign_out=00000000000000000000111100001111 | N=26: sign_in=00000000111100001111000011 | sign_out=00000000000000111100001111000011
Time=10 | N=16: sign_in=1111111100001111 | sign_out=11111111111111111111111100001111 | N=26: sign_in=11111111111100001111000011 | sign_out=1111111111111111110000111100001
Time=20 | N=16: sign_in=1000000000000000 | sign_out=11111111111111111000000000000000 | N=26: sign_in=10000000000000000000000000 | sign_out=111111100000000000000000000000
Time=30 | N=16: sign_in=0111111111111111 | sign_out=00000000000000000111111111111111 | N=26: sign_in=01111111111111111111111111 | sign_out=000000011111111111111111111111
jdoodle.v:47: $finish called at 40 (1s)
```

2.8 ALU 16 Bits

Verilog code

```
// ALU 16bit
module ALU_16bit(ALU_Sel, A, B, ALU_Out, zero);
            input [3:0] ALU_Sel;
            input [15:0] A;
            input [15:0] B;
            output reg [15:0] ALU_Out;
```

```
        output reg zero;
        parameter     ALU_OP_ADD        = 4'b0000, // Addition
                      ALU_OP_SUB        = 4'b0001, // Subtraction
                      ALU_OP_MUL     = 4'b0010, // Multiplication
                      ALU_OP_DIV     = 4'b0011, // Division
                      ALU_OP_AND        = 4'b0100, // Logical and
                      ALU_OP_OR = 4'b0101, // Logical or
                      ALU_OP_XOR        = 4'b0110, // Logical xor
                      ALU_OP_NOR        = 4'b0111, // Logical nor
                      ALU_OP_NAND       = 4'b1000, // Logical nand
                      ALU_OP_XNOR    = 4'b1001, // Logical xnor
                      ALU_OP_SHL     = 4'b1010, // Logical shift left
                      ALU_OP_SHR     = 4'b1011, // Logical shift right
                      ALU_OP_ROL     = 4'b1100, // Rotate left
                      ALU_OP_ROR     = 4'b1101, // Rotate right
                      ALU_OP_BEQ        = 4'b1110, // Equal comparison
                      ALU_OP_BNE     = 4'b1111; // Not Equal comparison
        always @(*)
              begin
               case(ALU_Sel)
                      ALU_OP_ADD        : begin
                                    zero=1'bx;
                                    ALU_Out = A + B;
                                  end
                      ALU_OP_SUB        : ALU_Out = A - B;
                      ALU_OP_MUL     : ALU_Out = A * B;
                      ALU_OP_DIV     : ALU_Out = A / B;
                      ALU_OP_AND        : ALU_Out = A & B;
                      ALU_OP_OR : ALU_Out = A | B;
                      ALU_OP_XOR        : ALU_Out = A ^ B;
                      ALU_OP_NOR        : ALU_Out = ~(A | B);
```

```
                    ALU_OP_NAND       : ALU_Out = ~(A & B);
                    ALU_OP_XNOR     : ALU_Out = ~(A ^ B);
                    ALU_OP_SHL      : ALU_Out = A<<1;
                    ALU_OP_SHR      : ALU_Out = A>>1;
                    ALU_OP_ROL      : ALU_Out = {A[6:0],A[7]};
                    ALU_OP_ROR      : ALU_Out = {A[0],A[7:1]};
                    ALU_OP_BEQ          : begin
                                        zero = (A==B)?1'b1:1'b0;
                                        ALU_Out = (A==B)?16'd1:16'd0;
                                  end
                    ALU_OP_BNE      : begin
                                        zero = (A!=B)?1'b1:1'b0;
                                        // zero here is different from the above zero
                                        // zero here is equal to 1, when A is different from
B
                                        // zero here is equal to 0, when A is the same as B
                                        ALU_Out = (A!=B)?16'd1:16'd0;
                                  end
              endcase
       end
endmodule
```

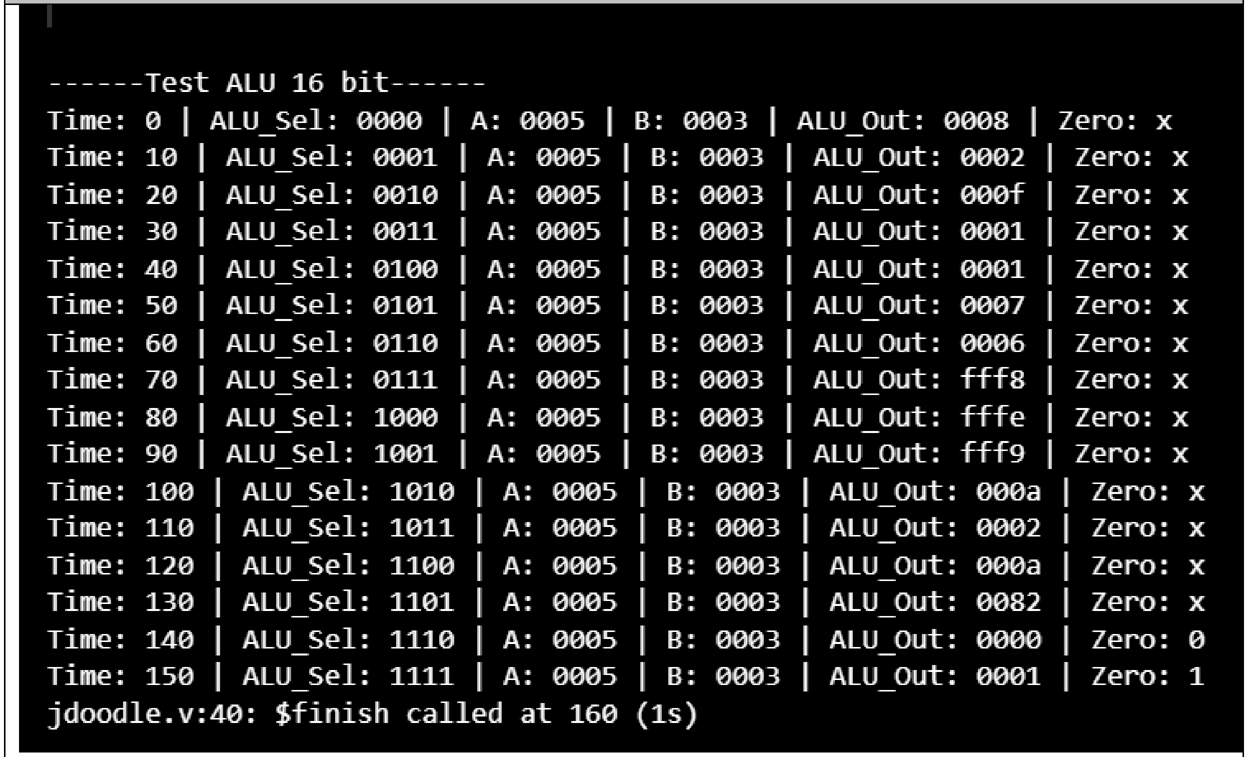
Testbench

```
       // Testbench
module tb_ALU_16bit;

  reg [3:0] ALU_Sel;
  reg [15:0] A,B;
  wire [15:0] ALU_Out;
  wire zero;
```

```
ALU_16bit uut (.ALU_Sel(ALU_Sel),.A(A),.B(B),.ALU_Out(ALU_Out),.zero(zero));

initial begin
    $display("\n------Test ALU 16 bit------");
    $monitor("Time: %0t | ALU_Sel: %b | A: %h | B: %h | ALU_Out: %h | Zero: %b",
             $time, ALU_Sel, A, B, ALU_Out, zero);



    A = 16'h0005;
    B = 16'h0003;



    ALU_Sel = 4'b0000; #10; // Addition
    ALU_Sel = 4'b0001; #10; // Subtraction
    ALU_Sel = 4'b0010; #10; // Multiplication
    ALU_Sel = 4'b0011; #10; // Division
    ALU_Sel = 4'b0100; #10; // Logical AND
    ALU_Sel = 4'b0101; #10; // Logical OR
    ALU_Sel = 4'b0110; #10; // Logical XOR
    ALU_Sel = 4'b0111; #10; // Logical NOR
    ALU_Sel = 4'b1000; #10; // Logical NAND
    ALU_Sel = 4'b1001; #10; // Logical XNOR
    ALU_Sel = 4'b1010; #10; // Logical Shift Left
    ALU_Sel = 4'b1011; #10; // Logical Shift Right
    ALU_Sel = 4'b1100; #10; // Rotate Left
    ALU_Sel = 4'b1101; #10; // Rotate Right
    ALU_Sel = 4'b1110; #10; // Equal Comparison
    ALU_Sel = 4'b1111; #10; // Not Equal Comparison

    $finish;
end
```

```
endmodule
```

Result



```
------Test ALU 16 bit------
Time: 0 | ALU_Sel: 0000 | A: 0005 | B: 0003 | ALU_Out: 0008 | Zero: x
Time: 10 | ALU_Sel: 0001 | A: 0005 | B: 0003 | ALU_Out: 0002 | Zero: x
Time: 20 | ALU_Sel: 0010 | A: 0005 | B: 0003 | ALU_Out: 000f | Zero: x
Time: 30 | ALU_Sel: 0011 | A: 0005 | B: 0003 | ALU_Out: 0001 | Zero: x
Time: 40 | ALU_Sel: 0100 | A: 0005 | B: 0003 | ALU_Out: 0001 | Zero: x
Time: 50 | ALU_Sel: 0101 | A: 0005 | B: 0003 | ALU_Out: 0007 | Zero: x
Time: 60 | ALU_Sel: 0110 | A: 0005 | B: 0003 | ALU_Out: 0006 | Zero: x
Time: 70 | ALU_Sel: 0111 | A: 0005 | B: 0003 | ALU_Out: fff8 | Zero: x
Time: 80 | ALU_Sel: 1000 | A: 0005 | B: 0003 | ALU_Out: fffe | Zero: x
Time: 90 | ALU_Sel: 1001 | A: 0005 | B: 0003 | ALU_Out: fff9 | Zero: x
Time: 100 | ALU_Sel: 1010 | A: 0005 | B: 0003 | ALU_Out: 000a | Zero: x
Time: 110 | ALU_Sel: 1011 | A: 0005 | B: 0003 | ALU_Out: 0002 | Zero: x
Time: 120 | ALU_Sel: 1100 | A: 0005 | B: 0003 | ALU_Out: 000a | Zero: x
Time: 130 | ALU_Sel: 1101 | A: 0005 | B: 0003 | ALU_Out: 0082 | Zero: x
Time: 140 | ALU_Sel: 1110 | A: 0005 | B: 0003 | ALU_Out: 0000 | Zero: 0
Time: 150 | ALU_Sel: 1111 | A: 0005 | B: 0003 | ALU_Out: 0001 | Zero: 1
jdoodle.v:40: $finish called at 160 (1s)
```

2.9 Completed Processor

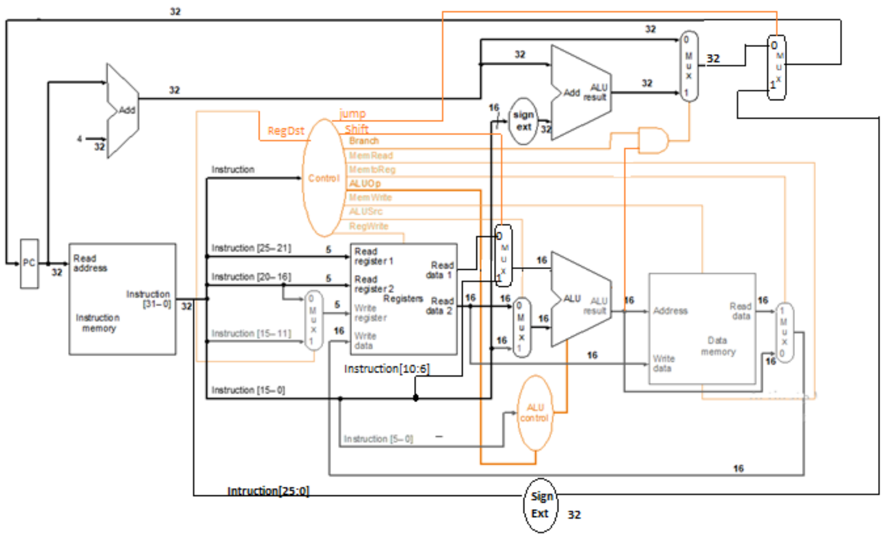


Figure 2.1 Single Cycle Microprocessor Schematic

Top module

```
module test(SW,LEDG,LEDR,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7);
        input [17:0]SW;
        output [17:0]LEDR;
        output [7:0]LEDG;
        output [0:6] HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
        wire [31:0] W_PC_out,W_LED_SEG, W_m2;
        wire [15:0] W_RD1, W_RD2, W_m1, W_ALUout;



        assign LEDR=SW;
        hex_ssd (W_RD1 [3:0], HEX0);
        hex_ssd (W_RD1[7:4], HEX1);
        hex_ssd ( W_RD2[3:0],HEX2);
        hex_ssd ( W_RD2[7:4],HEX3);
        hex_ssd (W_PC_out[3:0],HEX4);
        hex_ssd (W_PC_out[7:4],HEX5);
        hex_ssd (W_ALUout[3:0],HEX6);
        hex_ssd (W_ALUout[7:4],HEX7);



        Single_Cycle_Processor_16bit(.reset(SW[0]),.clk(SW[1]),.W_PC_out(W_PC_out),.in
struction(W_LED_SEG),.W_RD1(W_RD1),.W_RD2(W_RD2),.W_m1(W_m1),.W_m2(W_
m2),.W_ALUout(W_ALUout));
        endmodule
module hex_ssd (BIN, SSD);
        input [3:0] BIN;
        output reg [0:6] SSD;

        always begin
          case(BIN)
```

```
                  0:SSD=7'b0000001;
                  1:SSD=7'b1001111;
                  2:SSD=7'b0010010;
                  3:SSD=7'b0000110;
                  4:SSD=7'b1001100;
                  5:SSD=7'b0100100;
                  6:SSD=7'b0100000;
                  7:SSD=7'b0001111;
                  8:SSD=7'b0000000;
                  9:SSD=7'b0001100;
                  10:SSD=7'b0001000;
                  11:SSD=7'b1100000;
                  12:SSD=7'b0110001;
                  13:SSD=7'b1000010;
                  14:SSD=7'b0110000;
                  15:SSD=7'b0111000;
           endcase
    end
endmodule
```

Verilog code

```
//Single Cycle Processor 16bit
module Single_Cycle_Processor_16bit(reset, clk,W_PC_out, instruction, W_RD1,
W_RD2,W_m1,W_m2,W_ALUout);
        input reset, clk;
        output [31:0] W_PC_out, instruction, W_m2;
        output [15:0] W_RD1, W_RD2, W_m1, W_ALUout;
        wire [31:0] W_PC_in,W_PC_plus_1,W_Branch_add,W_m4,W_jump_PC;
        wire
PCsrc,RegWrite,zero,ALUsrc,MemRead,MemWrite,MemtoReg,RegDst,Branch,jump,shift;
        wire [15:0] W_MemtoReg,W_RD1,W_RD2,W_ALUout,W_m1,W_RDm,W_m5;
        wire [3:0] ALUop;
        wire [4:0] W_m3;

        Program_Counter                                   C1(.clk(clk),.reset(reset),.PC_in(
W_PC_plus_1),.PC_out(W_PC_out));
        Adder32Bit          C2(.input1(W_PC_out),.input2(32'd1),.out(W_PC_plus_1));
        Mux_32_bit
C3(.in0(W_PC_plus_1),.in1(W_m2),.mux_out(W_m4),.select(PCsrc));
        Mux_32_bit
C4(.in0(W_m4),.in1(W_jump_PC),.mux_out(W_PC_in),.select(jump));
        Adder32Bit    C5(.input1(W_PC_plus_1),.input2(W_Branch_add),.out(W_m2));
        Instruction_Memory
C6(.read_address(W_PC_out),.instruction(instruction),.reset(reset));
        Register_File_16bit
C7(.clk(clk),.read_addr_1(instruction[25:21]),.read_addr_2(instruction[20:16]),.write_addr(W
_m3),.write_data(W_MemtoReg),.RegWrite(RegWrite),.read_data_1(W_RD1),.read_data_2(
W_RD2));
        Sign_Extension_16    C8(.sign_in(instruction[15:0]),.sign_out(W_Branch_add));
        Mux_16_bit
C9(.in0(W_RD2),.in1(instruction[15:0]),.mux_out(W_m1),.select(ALUsrc));
        Mux_16_bit
C10(.in0(W_RD1),.in1(instruction[15:0]),.mux_out(W_m5),.select(shift));
        ALU_16bit
        C11(.ALU_Sel(ALUop),.A(W_m5),.B(W_m1),.ALU_Out(W_ALUout),.zero(zero));
        Data_Memory_16bit
C12(.clk(clk),.addr(W_ALUout),.write_data(W_RD2),.read_data(W_RDm),.MemRead(Mem
Read),.MemWrite(MemWrite));
        Mux_16_bit
C13(.in0(W_ALUout),.in1(W_RDm),.mux_out(W_MemtoReg),.select(MemtoReg));
        Sign_Extension_26    C14(.sign_in(instruction[25:0]),.sign_out(W_jump_PC));
        Control
C15(.clk(clk),.Op_intstruct(instruction[31:26]),.ints_function(instruction[5:0]),
.RegDst(RegDst),.Branch(Branch),.MemRead(MemRead),.MemtoReg(MemtoReg),.ALUOp(
ALUop),.MemWrite(MemWrite),.ALUSrc(ALUsrc),.RegWrite(RegWrite),.Zero(zero),.Jump
(jump),.Shift(shift));
        Mux_5_bit
C16(.in0(instruction[20:16]),.in1(instruction[15:11]),.mux_out(W_m3),.select(RegDst));
        and                                     C17(PCsrc,Branch,zero);
endmodule

// Control
module Control(clk, Op_intstruct, ints_function, RegDst, Branch, MemRead, MemtoReg,
ALUOp, MemWrite, ALUSrc, RegWrite, Zero, Jump,Shift);
   input clk,Zero;
   input [5:0] ints_function;
   input [5:0] Op_intstruct;
   output                                                                      reg
RegDst,Branch,MemRead,MemtoReg,Jump,MemWrite,ALUSrc,RegWrite,Shift;
   output reg [3:0] ALUOp;
   always @(*)
   begin
      case (Op_intstruct)
         6'b000000: begin // R-Type Instruction
            RegDst = 1;
            Jump = 0;
            Branch = 0;
            MemRead = 0;
            MemtoReg = 0;
            MemWrite = 0;
            ALUSrc = 0;
            RegWrite = 1;
            Shift =0;
            case (ints_function)
            6'b100000: ALUOp = 4'b0000; // Addition
            6'b100010: ALUOp = 4'b0001; // Subtraction
            6'b011000: ALUOp = 4'b0010; // Multiplication
            6'b011010: ALUOp = 4'b0011; // Division
            6'b100100: ALUOp = 4'b0100; // Logical AND
            6'b100101: ALUOp = 4'b0101; // Logical OR
            6'b100110: ALUOp = 4'b0110; // Logical XOR
            6'b100111: ALUOp = 4'b0111; // Logical NOR
            6'b101000: ALUOp = 4'b1000; // Logical NAND
            6'b101010: ALUOp = 4'b1001; // Logical XNOR
            6'b000000: begin // Logical Shift Left
              ALUOp = 4'b1010;
              Shift = 1;
            end
             6'b000010: begin // Logical Shift Right
              ALUOp = 4'b1011;
              Shift = 1;
            end
            6'b111000: begin // Rotate Left (ROL)
                                        ALUOp = 4'b1100;
                                        Shift = 1;
                                  end
                                  6'b110000: begin // Rotate Right (ROR)
                                        ALUOp = 4'b1101;
                                        Shift = 1;
                                  end
            default:   ALUOp = 4'b0000;
          endcase
        end
        // I-Type Instruction
        6'b000100: begin // BEQ
          RegDst = 0;
          Jump = 0;
          Branch = (Zero == 1) ? 1 : 0;
          MemRead = 0;
          MemtoReg = 0;
          ALUOp = 4'b1110;
          MemWrite = 0;
          ALUSrc = 0;
          RegWrite = 0;
          Shift =0;
        end
        6'b100011: begin // LW
          RegDst = 0;
          Jump = 0;
          Branch = 0;
          MemRead = 1;
          MemtoReg = 1;
          ALUOp = 4'b0000;
          MemWrite = 0;
          ALUSrc = 1;
          RegWrite = 1;
          Shift =0;
        end
        6'b101011: begin // SW
          RegDst = 0;
          Jump = 0;
          Branch = 0;
          MemRead = 0;
          MemtoReg = 0;
          ALUOp = 4'b0000;
          MemWrite = 1;
          ALUSrc = 1;
          RegWrite = 0;
          Shift =0;
        end
        6'b001000: begin // ADDI
          RegDst = 0;
          Jump = 0;
          Branch = 0;
          MemRead = 0;
          MemtoReg = 0;
          ALUOp = 4'b0000;
          MemWrite = 0;
          ALUSrc = 1;
          RegWrite = 1;
                                  Shift =0;
        end
        6'b000101: begin // BNE
          RegDst = 0;
          Jump = 0;
          Branch = (Zero == 0) ? 1 : 0;
          MemRead = 0;
          MemtoReg = 0;
          ALUOp = 4'b1111;
          MemWrite = 0;
          ALUSrc = 0;
          RegWrite = 0;
                                  Shift =0;
        end
                          // J-Type Instruction
        6'b000010: begin // JUMP
          RegDst = 0;
          Jump = 1;
          Branch = 0;
          MemRead = 0;
          MemtoReg = 0;
          ALUOp = 4'b0000;
          MemWrite = 0;
          ALUSrc = 1;
          RegWrite = 0;
                                  Shift =0;
        end
        6'b000011: begin // JUMP and LINK
                                  RegDst= 1;
                                  Jump = 1;
                                  Branch = 0;
                                  MemRead  = 0;
                                  MemtoReg = 1;
                                  ALUOp  = 4'b0000;
                                  MemWrite= 0;
                                  ALUSrc = 0;
                                  RegWrite  = 1;
                                  Shift =0;
                          end
                          default: begin // Default case
          RegDst = 0;
          Branch = 0;
          Jump = 0;
          MemRead = 0;
          MemtoReg = 0;
          ALUOp = 4'b0000;
          MemWrite = 0;
          ALUSrc = 0;
```

```
            RegWrite = 0;
                                    Shift =0;
         end
      endcase
   end
endmodule



// Register File 16bit
module Register_File_16bit (clk,read_addr_1, read_addr_2, write_addr, write_data,
RegWrite,read_data_1, read_data_2);
         input clk,RegWrite;
         input [4:0] read_addr_1, read_addr_2, write_addr;
         input [15:0] write_data;
         output [15:0] read_data_1, read_data_2;
         reg [15:0] Regfile [31:0];
         integer i;

         initial begin
                  for (i = 0; i < 32; i = i + 1) begin
                           Regfile[i] = 16'b0;
                  end
                  Regfile[19]=1; // $s3 = 1
         Regfile[17]=3; // $s1 = 3
         Regfile[20]=2; // $s4 = 2
      Regfile[21]=5; // $s5 = 5
         end

         assign read_data_1 = Regfile[read_addr_1];
         assign read_data_2 = Regfile[read_addr_2];
```

```
        always @(posedge clk) begin
                if (RegWrite) begin
                                Regfile[write_addr] <= write_data;
                                //$display("write_addr=%h
write_data=%h",write_addr,write_data);
                end
        end
endmodule



// ALU 16bit
module ALU_16bit(ALU_Sel, A, B, ALU_Out, zero);
        input [3:0] ALU_Sel;
        input [15:0] A;
        input [15:0] B;
        output reg [15:0] ALU_Out;
        output reg zero;
        parameter       ALU_OP_ADD          = 4'b0000, // Addition
                                ALU_OP_SUB          = 4'b0001, // Subtraction
                                ALU_OP_MUL     = 4'b0010, // Multiplication
                                ALU_OP_DIV     = 4'b0011, // Division
                                ALU_OP_AND          = 4'b0100, // Logical and
                                ALU_OP_OR          = 4'b0101, // Logical or
                                ALU_OP_XOR          = 4'b0110, // Logical xor
                                ALU_OP_NOR               = 4'b0111, // Logical nor
                                ALU_OP_NAND             = 4'b1000, // Logical nand
                                ALU_OP_XNOR     = 4'b1001, // Logical xnor
                                ALU_OP_SHL     = 4'b1010, // Logical shift left
                                ALU_OP_SHR     = 4'b1011, // Logical shift right
                                ALU_OP_ROL     = 4'b1100, // Rotate left
                                ALU_OP_ROR     = 4'b1101, // Rotate right
                                ALU_OP_BEQ          = 4'b1110, // Equal comparison
                                ALU_OP_BNE     = 4'b1111; // Not Equal comparison
        always @(*)
                begin
                        case(ALU_Sel)
                                ALU_OP_ADD          : begin
                                            zero=1'bx;
                                            ALU_Out = A + B;
                                           end
                                ALU_OP_SUB          : ALU_Out = A - B;
                                ALU_OP_MUL     : ALU_Out = A * B;
                                ALU_OP_DIV     : ALU_Out = A / B;
                                ALU_OP_AND          : ALU_Out = A & B;
                                ALU_OP_OR     : ALU_Out = A | B;
                                ALU_OP_XOR          : ALU_Out = A ^ B;
                                ALU_OP_NOR               : ALU_Out = ~(A | B);
                                ALU_OP_NAND              : ALU_Out = ~(A & B);
                                ALU_OP_XNOR    : ALU_Out = ~(A ^ B);
                                ALU_OP_SHL     : ALU_Out = A<<1;
                                ALU_OP_SHR     : ALU_Out = A>>1;
                                ALU_OP_ROL     : ALU_Out = {A[6:0],A[7]};
                                ALU_OP_ROR     : ALU_Out = {A[0],A[7:1]};
                                ALU_OP_BEQ          : begin
                                                          zero = (A==B)?1'b1:1'b0;
                                                          ALU_Out                =
(A==B)?16'd1:16'd0;
                                                    end
                                ALU_OP_BNE     : begin
                                                          zero = (A!=B)?1'b1:1'b0;
                                                    // zero here is different from the
above zero
```

```
                                                                 // zero here is equal to 1, when A
is different from B
                                                                 // zero here is equal to 0, when A
is the same as B
                                                                      ALU_Out                  =
(A!=B)?16'd1:16'd0;
                                                                 end
                              endcase
                        end
endmodule



// Sign Extension 16 bit
module Sign_Extension_16 (sign_in, sign_out);
    parameter N = 16;
    input [N-1:0] sign_in;
    output [31:0] sign_out;

    assign sign_out = {{(32-N){sign_in[N-1]}}, sign_in};
endmodule



// Sign Extension 26 bit
module Sign_Extension_26 (sign_in, sign_out);
    parameter N = 26;
            input [N-1:0] sign_in;
    output [31:0] sign_out;

    assign sign_out = {{(32-N){sign_in[N-1]}}, sign_in};
endmodule
```

```
// Adder 32 bit
module Adder32Bit(input1, input2, out);
    input [31:0] input1, input2;
    output [31:0] out;
    reg [31:0]out;
    always@( input1 or input2)
        begin
            out <= input1 + input2;
        end
endmodule



// Program Counter
module Program_Counter(clk, reset, PC_in, PC_out);
    input clk, reset;
    input [31:0] PC_in;
    output reg [31:0] PC_out;

    always @(posedge clk) begin
        if(reset)
            PC_out <= 32'b0;
        else
            PC_out <= PC_in;
    end
endmodule



// Mux 32_bit
module Mux_32_bit (in0, in1, mux_out, select);
            parameter N=32;
```

```
	input [N-1:0] in0, in1;
	output [N-1:0] mux_out;
	input select;
	assign mux_out = select ? in1 : in0;
endmodule



// Mux 16 bit
module Mux_16_bit (in0, in1, mux_out, select);
	parameter N=16;
	input [N-1:0] in0, in1;
	output [N-1:0] mux_out;
	input select;
	assign mux_out = select ? in1 : in0;
endmodule



// Mux 5 bit
module Mux_5_bit (in0, in1, mux_out, select);
	parameter N=5;
	input [N-1:0] in0, in1;
	output [N-1:0] mux_out;
	input select;
	assign mux_out = select ? in1 : in0;
endmodule



//Data Memory 16 bit
module Data_Memory_16bit (clk,addr,write_data,read_data,MemRead,MemWrite);
	input clk;
	input [15:0] addr;
	input [15:0] write_data;
	output reg [15:0] read_data;
	input MemRead;
	input MemWrite;
	integer i;
	reg [15:0] DMemory [0:255];

	initial begin
		for (i = 0; i < 256; i = i + 1) begin
			DMemory[i] = 16'h0000;
		end
	end

	always @(posedge clk) begin
		if (MemWrite) begin
			DMemory[addr] <= write_data;
			//$display("Memory Write: Address=%h Data=%h", addr, write_data);
		end
	end

	always @(*) begin
		if (MemRead)
			read_data = DMemory[addr];
		else
			read_data = 16'h0000;
	end
endmodule



// Instruction Memory
module Instruction_Memory (read_address, instruction, reset);
```

```
            input reset;
            input [31:0] read_address;
            output [31:0] instruction;
            reg [31:0] Imemory [256:0];
            integer i;
            assign instruction = Imemory[read_address];
            always @(posedge reset)
            begin
            for (i=0; i<32; i=i+1)
                        begin
                          Imemory[i] = 32'b0;
                        end
                        /*Test R-type*/
                        // add $s6, $s5, $s4          R22 = R21 + R20 = 0x7 (0x5 + 0x2)
                        Imemory[0] = 32'b000000_10101_10100_10110_00000_100000;
                        // sub $s6, $s5, $s4          R22 = R21 - R20 = 0x3 (0x5 - 0x2)
                        Imemory[1] = 32'b000000_10101_10100_10110_00000_100010;
                        // mult $s5, $s4              R21 * R20 = 0xA (0x5 * 0x2)
                        Imemory[2] = 32'b000000_10101_10100_00000_00000_011000;
                        // div $s5, $s4               R21 / R20 = 0x2 (0x5 / 0x2)
                        Imemory[3] = 32'b000000_10101_10100_00000_00000_011010;
                        // and $s6, $s5, $s4          R22 = AND(R21, R20) = 0x0 (0x5 AND 0x2)
                        Imemory[4] = 32'b000000_10101_10100_10110_00000_100100;
                        // or $s6, $s5, $s4           R22 = OR(R21, R20) = 0x7 (0x5 OR 0x2)
                        Imemory[5] = 32'b000000_10101_10100_10110_00000_100101;
                        // xor $s6, $s5, $s4          R22 = XOR(R21, R20) = 0x7 (0x5 XOR 0x2)
                        Imemory[6] = 32'b000000_10101_10100_10110_00000_100110;
                        // nor $s6, $s5, $s4          R22 = NOR(R21, R20) = 0xFFFFFFF8 (NOR of 0x5
and 0x2)
                        Imemory[7] = 32'b000000_10101_10100_10110_00000_100111;
                        // nand $s6, $s5, $s4         R22 = NAND(R21, R20) = 0xFFFFFFFD (NAND of
0x5 and 0x2)
                        Imemory[8] = 32'b000000_10101_10100_10110_00000_101000;
                        // xnor $s6, $s5, $s4        R22 = XNOR(R21, R20) = 0xFFFFFFFE (XNOR of
0x5 and 0x2)
                        Imemory[9] = 32'b000000_10101_10100_10110_00000_101010;
                        // sll $s6, $s5, $s4         R22 = R21 << R20 = 0x14 (0x5 << 0x2)
                        Imemory[10] = 32'b00000_10101_10100_10110_00000_000000;
                        // srl $s6, $s5, $s4         R22 = R21 >> R20 = 0x1 (0x5 >> 0x2)
                        Imemory[11] = 32'b000000_10101_10100_10110_00000_000010;
                        // rol $s6, $s5              R22 = R21 rotated 1 bit left = 0xA (rotate 0x5 left by 1)
                        Imemory[12] = 32'b000000_10101_00000_10110_00000_111000;
                        // ror $s6, $s5              R22 = R21 rotated 1 bit right = 0x2 (rotate 0x5 right by
1)
                        Imemory[13] = 32'b000000_10101_00000_10110_00000_110000;


                        /*Test I-type*/
             // addi $s1, $s1, 0x05        R17 = 0x4E => R17 = R17 + 0x5 = 0x53
             Imemory[14] = 32'b001000_10001_10001_00000_00000_000101;
             // addi $s1, $s1, 0x05        R17 = 0x53 => R17 = R17 + 0x5 = 0x58
             Imemory[15] = 32'b001000_10001_10001_00000_00000_000101;
             // addi $s3, $s1, 0x10        R17 = 0x58 => R19 = R17 + 0x10 = 0x68
             Imemory[16] = 32'b001000_10001_10011_00000_00000_010000;
             // sw  $s3, 0x04($s1)        Memory[$s1 + 0x04] = $s3 => Memory[0x58 + 0x04] = 0x68
             Imemory[17] = 32'b101011_10001_10011_00000_00000_000100;
             // lw $s7, 0x04($s1)        $s7 = Memory[$s1 + 0x04] = 0x68
                        Imemory[18] = 32'b100011_10001_10111_00000_00000_000100;
             // beq $s7, $s3, 0x08        R23 (0x68) == R19 (0x68) move to Imemory[22 + 8 + 1]
             Imemory[19] = 32'b000100_10111_10011_00000_00000_001000;
             // addi $s1, $zero, 0x11      R17 = 0x11
             Imemory[20] = 32'b001000_00000_10001_00000_00000_010001;
             // addi $s1, $zero, 0x22      R17 = 0x22
```

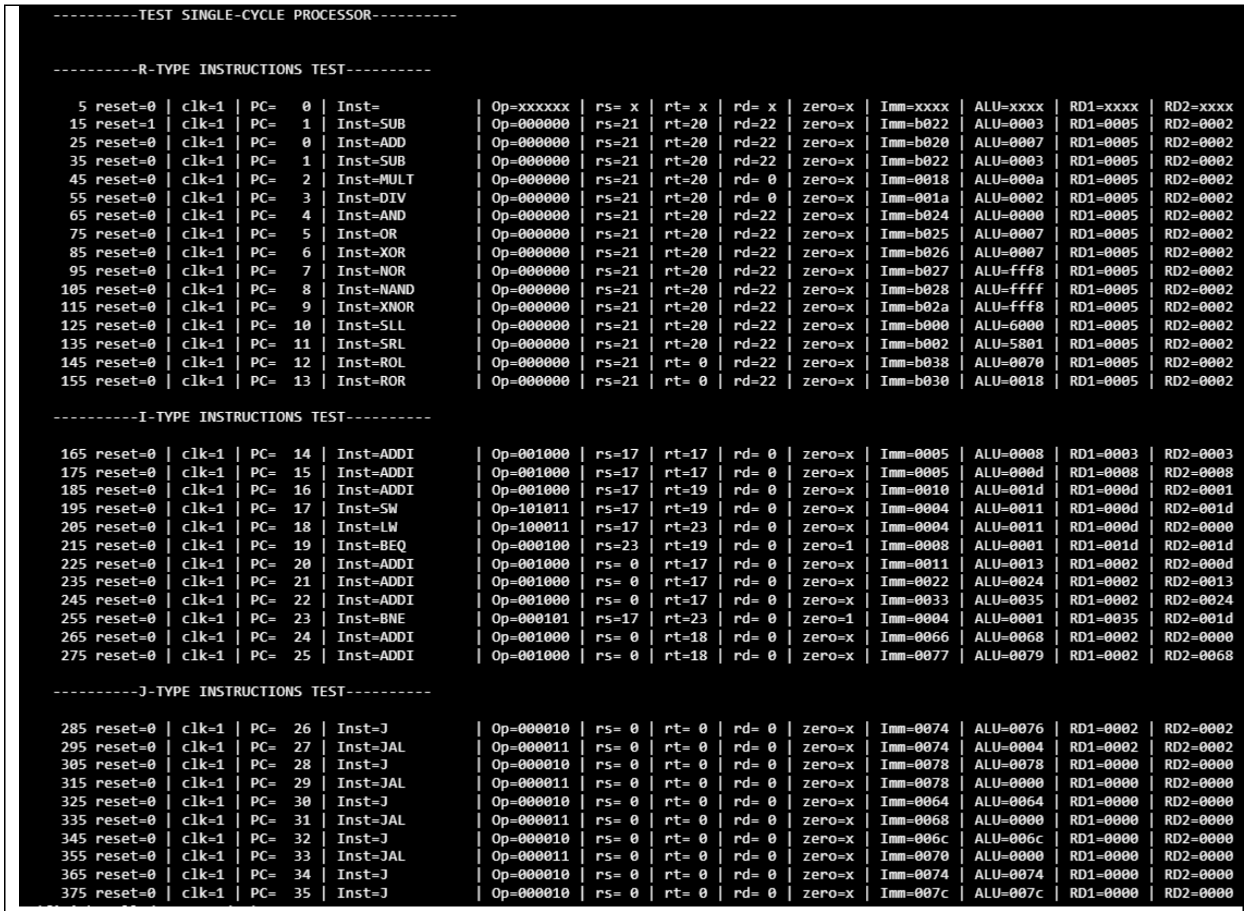
```
        Imemory[21] = 32'b001000_00000_10001_00000_00000_100010;
        // addi $s1, $zero, 0x33      R17 = 0x33
        Imemory[22] = 32'b001000_00000_10001_00000_00000_110011;
        // bne $s1, $s7, 0x04       (R17 (0x33) != R23 (0x68)) move to Imemory[26 + 4 + 1]
        Imemory[23] = 32'b000101_10001_10111_00000_00000_000100;
        // addi $s2, $zero, 0x66      R18 = 0x66
        Imemory[24] = 32'b001000_00000_10010_00000_00001_100110;
        // addi $s2, $zero, 0x77      R18 = 0x77
        Imemory[25] = 32'b001000_00000_10010_00000_00001_110111;

       /* Test J-type  */
                // j 0x74 (JUMP to Imemory[116])
                Imemory[26] = 32'b000010_00000_00000_00000_00001_110100;
                // jal 0x74 (Jump and Link to Imemory[116])
                Imemory[27] = 32'b000011_00000_00000_00000_00001_110100;
                // j 0x78 (JUMP to Imemory[120])
                Imemory[28] = 32'b000010_00000_00000_00000_00001_111000;
                // jal 0x78 (Jump and Link to Imemory[120])
                Imemory[29] = 32'b000011_00000_00000_00000_00001_111000;
                // j 0x64 (JUMP to Imemory[100])
                Imemory[30] = 32'b000010_00000_00000_00000_00001_100100;
                // jal 0x68 (Jump and Link to Imemory[104])
                Imemory[31] = 32'b000011_00000_00000_00000_00001_101000;
                // j 0x6C (JUMP to Imemory[108])
                Imemory[32] = 32'b000010_00000_00000_00000_00001_101100;
                // jal 0x70 (Jump and Link to Imemory[112])
                Imemory[33] = 32'b000011_00000_00000_00000_00001_110000;
                // j 0x74 (JUMP to Imemory[116])
                Imemory[34] = 32'b000010_00000_00000_00000_00001_110100;
                // j 0x7C (JUMP to Imemory[124])
                Imemory[35] = 32'b000010_00000_00000_00000_00001_111100;
```

```
      end
endmodule
```

Testbench

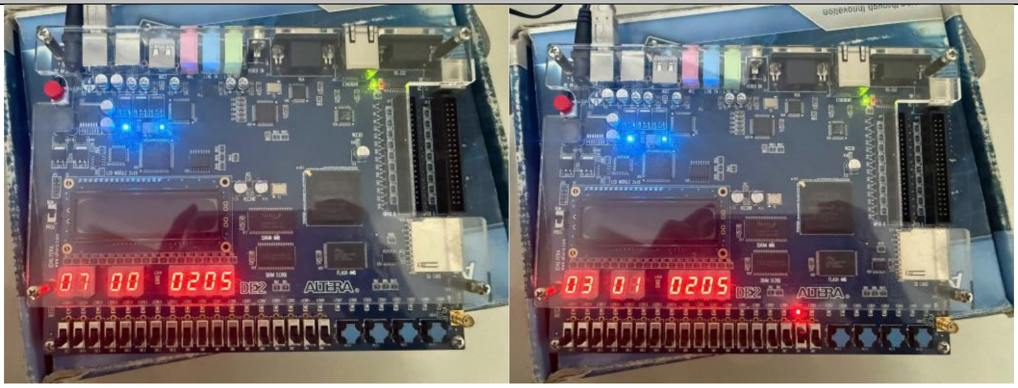
```
// Testbench
module tb_Single_Cycle_Processor;
  reg reset, clk;
  wire [31:0] W_PC_out, instruction, W_m2;
  wire [15:0] W_RD1, W_RD2, W_m1, W_ALUout;
  reg [127:0] instr_name;


  Single_Cycle_Processor_16bit dut(
    .reset(reset),
    .clk(clk),
    .W_PC_out(W_PC_out),
    .instruction(instruction),
    .W_RD1(W_RD1),
    .W_RD2(W_RD2),
    .W_m1(W_m1),
    .W_m2(W_m2),
    .W_ALUout(W_ALUout)
  );


  always #5 clk = ~clk;


  always @(instruction) begin
    case (instruction[31:26])
      6'b000000: begin
```

```
            case (instruction[5:0]) // funct field
                6'b100000: instr_name = "ADD       ";
                6'b100010: instr_name = "SUB       ";
                6'b011000: instr_name = "MULT      ";
                6'b011010: instr_name = "DIV       ";
                6'b100100: instr_name = "AND       ";
                6'b100101: instr_name = "OR        ";
                6'b100110: instr_name = "XOR       ";
                6'b100111: instr_name = "NOR       ";
                6'b101000: instr_name = "NAND      ";
                6'b101010: instr_name = "XNOR      ";
                6'b000000: instr_name = "SLL       ";
                6'b000010: instr_name = "SRL       ";
                6'b111000: instr_name = "ROL       ";
                6'b110000: instr_name = "ROR       ";
                default:   instr_name = "UNKNOWN_R ";
            endcase
        end
        6'b000100: instr_name = "BEQ       ";
        6'b100011: instr_name = "LW        ";
        6'b101011: instr_name = "SW        ";
        6'b001000: instr_name = "ADDI      ";
        6'b000101: instr_name = "BNE       ";
        6'b000010: instr_name = "J         ";
        6'b000011: instr_name = "JAL       ";
        default:   instr_name = "UNKNOWN   ";
    endcase
end

always @(posedge clk) begin
    $display($time,
            " reset=%b | clk=%b | PC=%4d | Inst=%-10s | Op=%6b | rs=%2d | rt=%2d | rd=%2d |
zero=%b | Imm=%4h | ALU=%4h | RD1=%4h | RD2=%4h",
            reset, clk, W_PC_out, instr_name, instruction[31:26],
            instruction[25:21], instruction[20:16], instruction[15:11],
            dut.C11.zero, instruction[15:0], W_ALUout, W_RD1, W_RD2);
    end

    initial begin
        clk = 0;
        reset = 1;
        instr_name = "";



        $display("\n\t\t\t\t----------TEST SINGLE-CYCLE PROCESSOR----------\n");
        $display("\n\t\t\t\t----------R-TYPE INSTRUCTIONS TEST----------\n");

        #10 reset = 0;
        #10 reset = 1;
        #140 $display("\n\t\t\t\t----------I-TYPE INSTRUCTIONS TEST----------\n");



        #120 $display("\n\t\t\t\t----------J-TYPE INSTRUCTIONS TEST----------\n");

        #100 $finish;
    end
endmodule
```

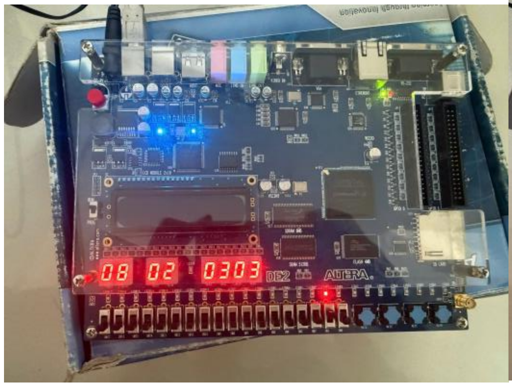
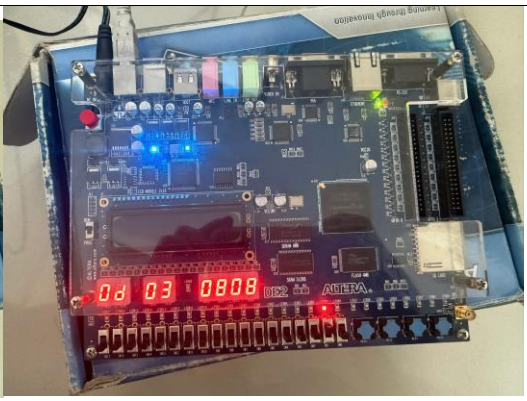
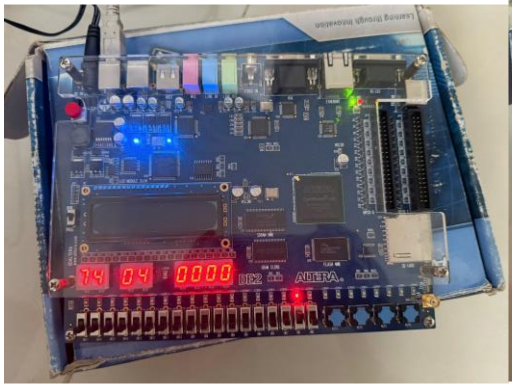
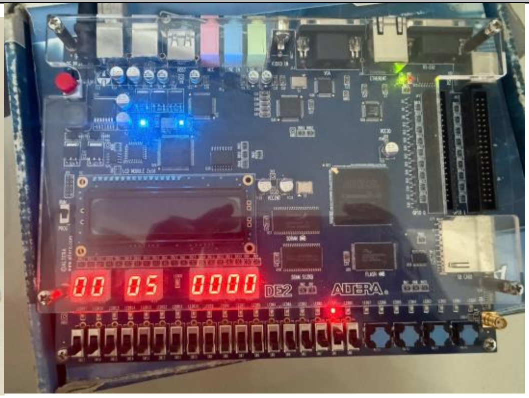


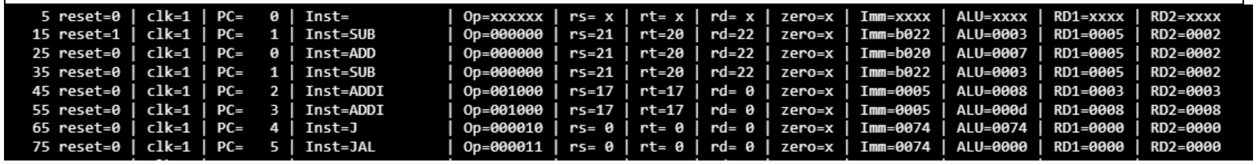
```
 ----------TEST SINGLE-CYCLE PROCESSOR----------


 ----------R-TYPE INSTRUCTIONS TEST----------

   5 reset=0 | clk=1 | PC=   0 | Inst=          | Op=xxxxxx | rs= x | rt= x | rd= x | zero=x | Imm=xxxx | ALU=xxxx | RD1=xxxx | RD2=xxxx
  15 reset=1 | clk=1 | PC=   1 | Inst=SUB       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b022 | ALU=0003 | RD1=0005 | RD2=0002
  25 reset=0 | clk=1 | PC=   0 | Inst=ADD       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b020 | ALU=0007 | RD1=0005 | RD2=0002
  35 reset=0 | clk=1 | PC=   1 | Inst=SUB       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b022 | ALU=0003 | RD1=0005 | RD2=0002
  45 reset=0 | clk=1 | PC=   2 | Inst=MULT      | Op=000000 | rs=21 | rt=20 | rd= 0 | zero=x | Imm=0018 | ALU=000a | RD1=0005 | RD2=0002
  55 reset=0 | clk=1 | PC=   3 | Inst=DIV       | Op=000000 | rs=21 | rt=20 | rd= 0 | zero=x | Imm=001a | ALU=0002 | RD1=0005 | RD2=0002
  65 reset=0 | clk=1 | PC=   4 | Inst=AND       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b024 | ALU=0000 | RD1=0005 | RD2=0002
  75 reset=0 | clk=1 | PC=   5 | Inst=OR        | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b025 | ALU=0007 | RD1=0005 | RD2=0002
  85 reset=0 | clk=1 | PC=   6 | Inst=XOR       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b026 | ALU=0007 | RD1=0005 | RD2=0002
  95 reset=0 | clk=1 | PC=   7 | Inst=NOR       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b027 | ALU=fff8 | RD1=0005 | RD2=0002
 105 reset=0 | clk=1 | PC=   8 | Inst=NAND      | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b028 | ALU=ffff | RD1=0005 | RD2=0002
 115 reset=0 | clk=1 | PC=   9 | Inst=XNOR      | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b02a | ALU=fff8 | RD1=0005 | RD2=0002
 125 reset=0 | clk=1 | PC=  10 | Inst=SLL       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b000 | ALU=6000 | RD1=0005 | RD2=0002
 135 reset=0 | clk=1 | PC=  11 | Inst=SRL       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b002 | ALU=5801 | RD1=0005 | RD2=0002
 145 reset=0 | clk=1 | PC=  12 | Inst=ROL       | Op=000000 | rs=21 | rt= 0 | rd=22 | zero=x | Imm=b038 | ALU=0070 | RD1=0005 | RD2=0002
 155 reset=0 | clk=1 | PC=  13 | Inst=ROR       | Op=000000 | rs=21 | rt= 0 | rd=22 | zero=x | Imm=b030 | ALU=0018 | RD1=0005 | RD2=0002

 ----------I-TYPE INSTRUCTIONS TEST----------

 165 reset=0 | clk=1 | PC=  14 | Inst=ADDI      | Op=001000 | rs=17 | rt=17 | rd= 0 | zero=x | Imm=0005 | ALU=0008 | RD1=0003 | RD2=0003
 175 reset=0 | clk=1 | PC=  15 | Inst=ADDI      | Op=001000 | rs=17 | rt=17 | rd= 0 | zero=x | Imm=0005 | ALU=000d | RD1=0008 | RD2=0008
 185 reset=0 | clk=1 | PC=  16 | Inst=ADDI      | Op=001000 | rs=17 | rt=19 | rd= 0 | zero=x | Imm=0010 | ALU=001d | RD1=000d | RD2=0001
 195 reset=0 | clk=1 | PC=  17 | Inst=SW        | Op=101011 | rs=17 | rt=19 | rd= 0 | zero=x | Imm=0004 | ALU=0011 | RD1=000d | RD2=001d
 205 reset=0 | clk=1 | PC=  18 | Inst=LW        | Op=100011 | rs=17 | rt=23 | rd= 0 | zero=x | Imm=0004 | ALU=0011 | RD1=000d | RD2=0000
 215 reset=0 | clk=1 | PC=  19 | Inst=BEQ       | Op=000100 | rs=23 | rt=19 | rd= 0 | zero=1 | Imm=0008 | ALU=0001 | RD1=001d | RD2=001d
 225 reset=0 | clk=1 | PC=  20 | Inst=ADDI      | Op=001000 | rs= 0 | rt=17 | rd= 0 | zero=x | Imm=0011 | ALU=0013 | RD1=0002 | RD2=000d
 235 reset=0 | clk=1 | PC=  21 | Inst=ADDI      | Op=001000 | rs= 0 | rt=17 | rd= 0 | zero=x | Imm=0022 | ALU=0024 | RD1=0002 | RD2=0013
 245 reset=0 | clk=1 | PC=  22 | Inst=ADDI      | Op=001000 | rs= 0 | rt=17 | rd= 0 | zero=x | Imm=0033 | ALU=0035 | RD1=0002 | RD2=0024
 255 reset=0 | clk=1 | PC=  23 | Inst=BNE       | Op=000101 | rs=17 | rt=23 | rd= 0 | zero=1 | Imm=0004 | ALU=0001 | RD1=0035 | RD2=001d
 265 reset=0 | clk=1 | PC=  24 | Inst=ADDI      | Op=001000 | rs= 0 | rt=18 | rd= 0 | zero=x | Imm=0066 | ALU=0068 | RD1=0002 | RD2=0000
 275 reset=0 | clk=1 | PC=  25 | Inst=ADDI      | Op=001000 | rs= 0 | rt=18 | rd= 0 | zero=x | Imm=0077 | ALU=0079 | RD1=0002 | RD2=0068

 ----------J-TYPE INSTRUCTIONS TEST----------

 285 reset=0 | clk=1 | PC=  26 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0074 | ALU=0076 | RD1=0002 | RD2=0002
 295 reset=0 | clk=1 | PC=  27 | Inst=JAL       | Op=000011 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0074 | ALU=0004 | RD1=0002 | RD2=0002
 305 reset=0 | clk=1 | PC=  28 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0078 | ALU=0078 | RD1=0000 | RD2=0000
 315 reset=0 | clk=1 | PC=  29 | Inst=JAL       | Op=000011 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0078 | ALU=0000 | RD1=0000 | RD2=0000
 325 reset=0 | clk=1 | PC=  30 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0064 | ALU=0064 | RD1=0000 | RD2=0000
 335 reset=0 | clk=1 | PC=  31 | Inst=JAL       | Op=000011 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0068 | ALU=0000 | RD1=0000 | RD2=0000
 345 reset=0 | clk=1 | PC=  32 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=006c | ALU=006c | RD1=0000 | RD2=0000
 355 reset=0 | clk=1 | PC=  33 | Inst=JAL       | Op=000011 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0070 | ALU=0000 | RD1=0000 | RD2=0000
 365 reset=0 | clk=1 | PC=  34 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0074 | ALU=0074 | RD1=0000 | RD2=0000
 375 reset=0 | clk=1 | PC=  35 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=007c | ALU=007c | RD1=0000 | RD2=0000
```

R-Type



I-Type

 
J-Type
 



```
 5 reset=0 | clk=1 | PC=   0 | Inst=          | Op=xxxxxx | rs= x | rt= x | rd= x | zero=x | Imm=xxxx | ALU=xxxx | RD1=xxxx | RD2=xxxx
15 reset=1 | clk=1 | PC=   1 | Inst=SUB       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b022 | ALU=0003 | RD1=0005 | RD2=0002
25 reset=0 | clk=1 | PC=   0 | Inst=ADD       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b020 | ALU=0007 | RD1=0005 | RD2=0002
35 reset=0 | clk=1 | PC=   1 | Inst=SUB       | Op=000000 | rs=21 | rt=20 | rd=22 | zero=x | Imm=b022 | ALU=0003 | RD1=0005 | RD2=0002
45 reset=0 | clk=1 | PC=   2 | Inst=ADDI      | Op=001000 | rs=17 | rt=17 | rd= 0 | zero=x | Imm=0005 | ALU=0008 | RD1=0003 | RD2=0003
55 reset=0 | clk=1 | PC=   3 | Inst=ADDI      | Op=001000 | rs=17 | rt=17 | rd= 0 | zero=x | Imm=0005 | ALU=000d | RD1=0008 | RD2=0008
65 reset=0 | clk=1 | PC=   4 | Inst=J         | Op=000010 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0074 | ALU=0074 | RD1=0000 | RD2=0000
75 reset=0 | clk=1 | PC=   5 | Inst=JAL       | Op=000011 | rs= 0 | rt= 0 | rd= 0 | zero=x | Imm=0074 | ALU=0000 | RD1=0000 | RD2=0000
```

I use the results of this testbench to verify the outcomes on the FBGA because there is no debouncing in the SWing button. As a result, during operation, the register values may change, leading to incorrect results. Therefore, I modified the first 6 commands to test all three types (R, I, J).