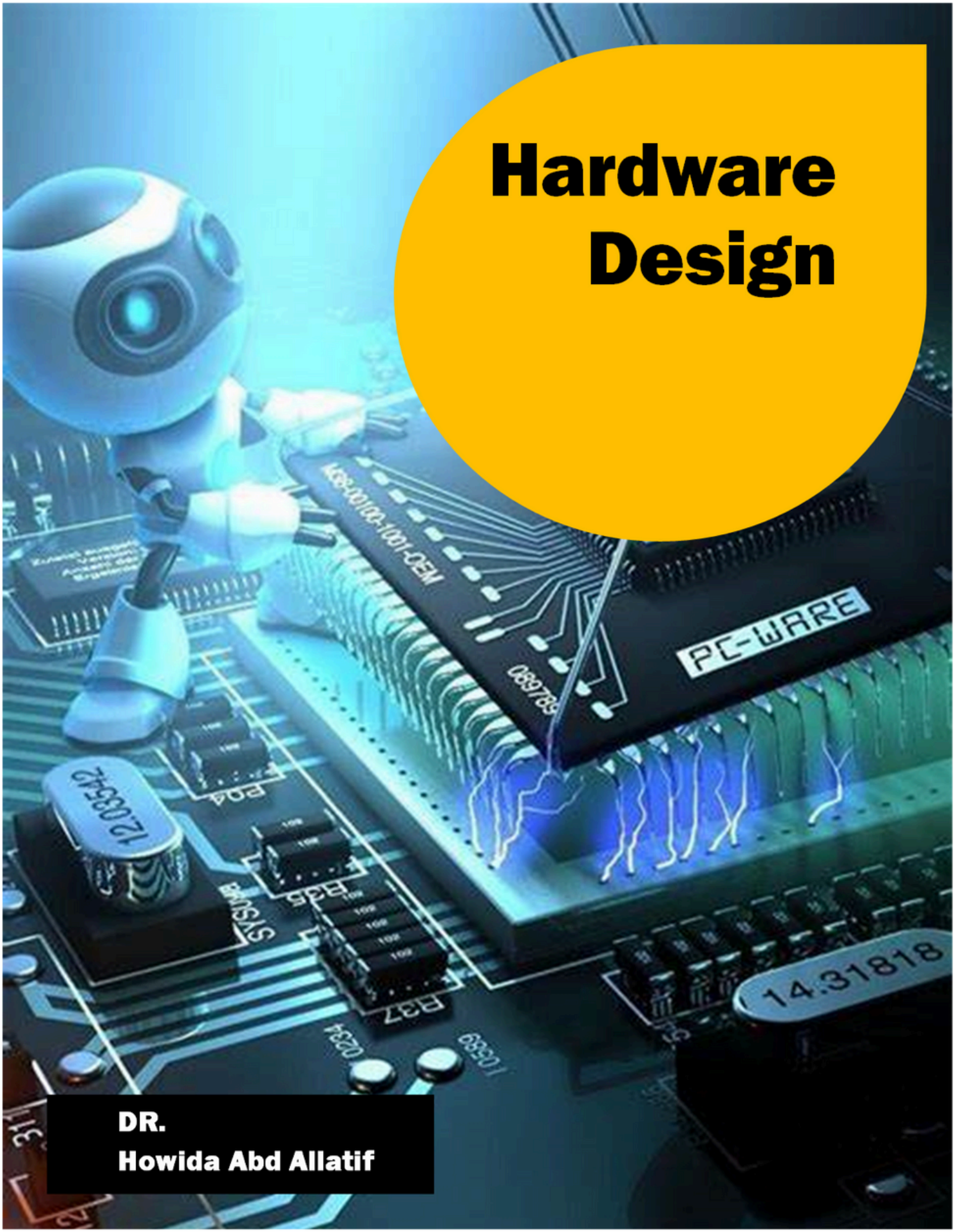


Hardware Design

DR.
Howida Abd Allatif

TEAM MEMBERS

No	Name	Academic Number	Tasks completed
1	كريم وائل طلبه محمد	20812021101083	Register File + TB & Report
2	محمد محمود محمد غانم	20812021101295	Alu + TB & Adder + TB
3	محمد أحمد محمود محمد حسين	20812021101167	Instruction Memory & Muxs +TBs
4	أحمد محمد صلاح الدين عبد الخالق	20812021100722	Control Unit + TB & Sign Extend
5	أحمد جمال عزقلانى عطيه	20812021100234	Data Memory +TB
6	عبدالرحمن علاء أحمد مصيلحى	20812021100238	PC + TB Shift Left + Tb Alu Control TB

THE ASSIGNMENT AIM

Creating a 32-bit processor system involves several steps and components.

Overall Architecture Design:

- Define the architecture of the processor, including its components such the ALU (Arithmetic Logic Unit), control unit, register file.

Testing and Verification:

- Create comprehensive testbenches to verify the functionality of each component and the overall processor system.
- Utilize simulation tools to conduct thorough testing under various instruction sequences and data inputs.

Data Path:

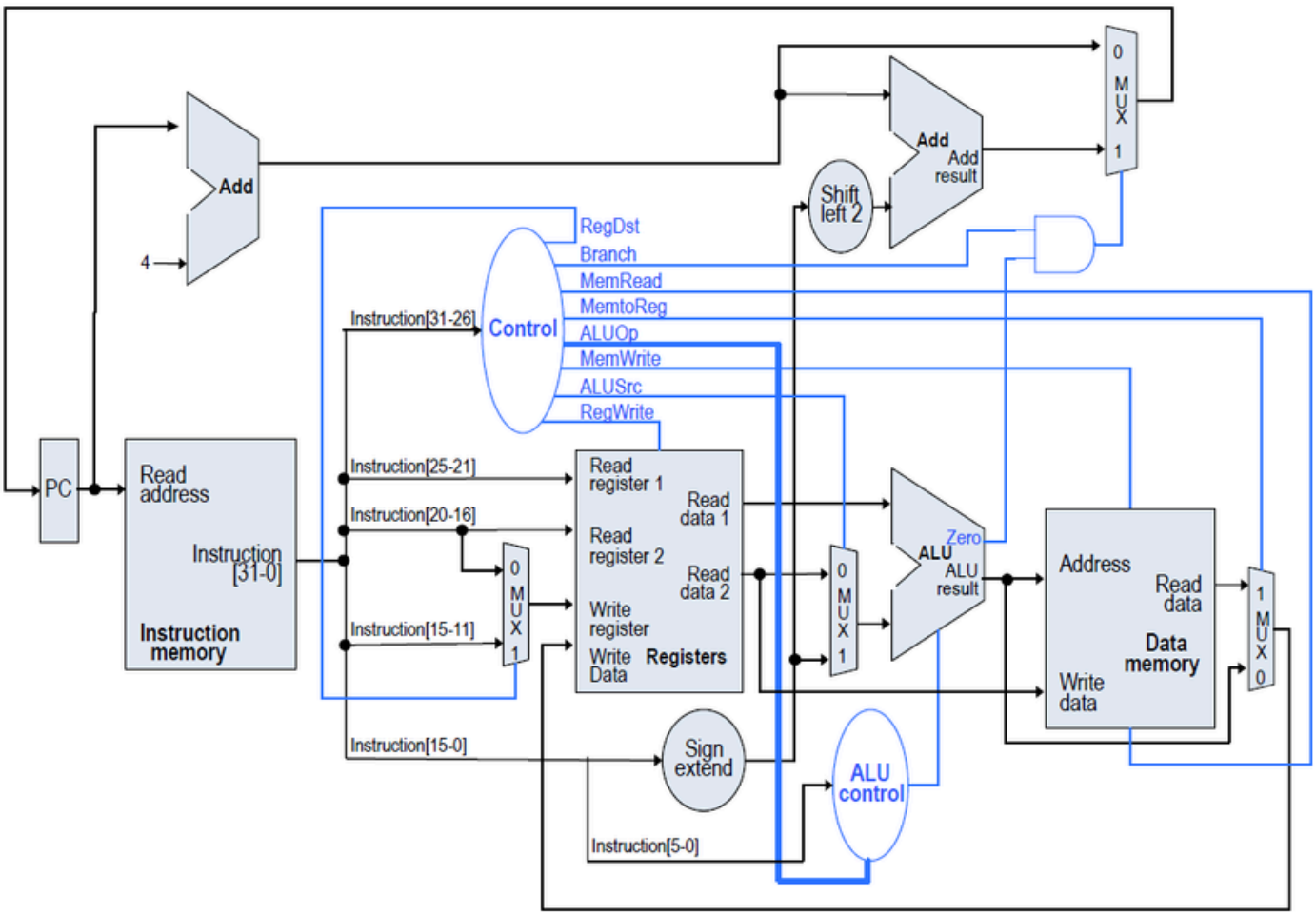
- Ensure that the data path supports 32-bit data operations and can execute instructions efficiently.

Documentation:

- Document the design process, architecture, components, interfaces, and operation of the 32-bit processor system.
- Disseminate the project findings through reports and presentations.

The successful completion of this project will contribute to the advancement of hardware design and computer architecture, providing valuable insights into the design and implementation of high-performance processor systems using VHDL.

THE PROBLEM SOLUTION



Required Components:

1. Program Counter (PC)
2. Register File
3. Arithmetic Logic Unit (ALU)
4. Data Memory Unit (DMU)
5. Control Unit
6. Multiplexer (MUX) Units
7. Sign Extend Unit
8. Shift Register
9. Clock and Reset Logic

Program Counter (PC):

-Stores the address of the next instruction to be fetched.

-Increments by the instruction size after each fetch.

Register File:

-Stores general-purpose registers.

-Provides read and write access to registers based on instruction requirements.

Arithmetic Logic Unit (ALU):

-Performs arithmetic and logical operations on data.

-Supports operations like addition, subtraction, AND, OR, shift, and comparison.

Data Memory Unit (DMU):

-Stores data accessed by load and store instructions.

-Interfaces with data memory.

Control Unit:

-Generates control signals based on the instruction opcode and other relevant bits.

-Controls the data path components based on the current instruction.

Multiplexer (MUX) Units:
Selects between different sources of data or control signals based on control signals.

Sign Extend Unit:
Extends immediate values to the full width of the data path when necessary.

Shift Register:
Performs logical and arithmetic shifts on data.

Clock and Reset Logic:
Provides clock signals to synchronize operations and reset signals to initialize the processor.

IMPLEMENTATION

- ***Program Counter (PC)***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY PC IS
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        din : IN STD_LOGIC_VECTOR (31 DOWNTO 0) := (OTHERS => '0');
        dout : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
END PC;

ARCHITECTURE Behavioral OF PC IS
    SIGNAL temp : STD_LOGIC_VECTOR (31 DOWNTO 0);
BEGIN
    PROCESS (clk, reset)
    BEGIN
        IF (reset = '1') THEN
            temp <= (OTHERS => '0');
        ELSIF (clk'event AND clk = '1') THEN
            temp <= din;
        END IF;
    END PROCESS;
    dout <= temp;

END Behavioral;
```

- ***Register File***

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY Registers IS PORT (
    clk, rst, rw : IN STD_LOGIC;
    readRg1, readRg2, writeRg : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
    readData1, readData2 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    writeData : IN STD_LOGIC_VECTOR(31 DOWNTO 0)

);
END Registers;

ARCHITECTURE RegistersArch OF Registers IS
    TYPE registerFile IS ARRAY(31 DOWNTO 0) OF STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL regFile : registerFile;
    SIGNAL t0 : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL address : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN

    PROCESS (clk, rst)
        VARIABLE zVector : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
    BEGIN
        IF rst = '1' THEN
            -- Initialize all registers to Zeros on reset
            FOR i IN regFile'RANGE LOOP
                regFile(i) <= zVector;
            END LOOP;
        ELSIF rising_edge(clk) THEN
            -- Writing process
            IF rw = '1' THEN
                regFile(to_integer(unsigned(writeRg))) <= writeData;
            END IF;
        END IF;
    END PROCESS;

    -- Get registers data
    readData1 <= regFile(to_integer(unsigned(readRg1)));
    readData2 <= regFile(to_integer(unsigned(readRg2)));
    t0 <= regFile(to_integer(unsigned(address)));
END RegistersArch;
```

- ***Control Unit***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY ControlUnit IS
    PORT (
        OpCode : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
        MemtoReg : OUT STD_LOGIC;
        MemWrite : OUT STD_LOGIC;
        MemRead : OUT STD_LOGIC;
        Branch : OUT STD_LOGIC;
        AluSrc : OUT STD_LOGIC;
        RegDst : OUT STD_LOGIC;
        RegWrite : OUT STD_LOGIC;
        AluOp : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END ControlUnit;
ARCHITECTURE Behavioral OF ControlUnit IS
    SIGNAL aux : STD_LOGIC_VECTOR (9 DOWNTO 0) := (OTHERS => '0');
BEGIN
    WITH OpCode SELECT
        aux <= "1000001001" WHEN "000000", -- R format
        "0100000010" WHEN "000100", -- beq
        "0000011000" WHEN "001000", -- addi
        "0000011011" WHEN "001101", -- ori
        "0000011100" WHEN "001111", -- lui
        "0011011000" WHEN "100011", -- lw
        "0000110000" WHEN "101011", -- sw
        "0000000000" WHEN OTHERS;

    RegDst <= aux(9);
    Branch <= aux(8);
    MemRead <= aux(7);
    MemtoReg <= aux(6);
    MemWrite <= aux(5);
    AluSrc <= aux(4);
    RegWrite <= aux(3);
    AluOp <= aux(2 DOWNTO 0);
END Behavioral;
```

- ***Arithmetic Logic Unit (ALU)***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY alu IS
    PORT (
        a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        cntrl : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        zero : OUT STD_LOGIC;
        result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END alu;

ARCHITECTURE Behavioral OF alu IS
    SIGNAL aux_result : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
    PROCESS (a, b, cntrl)
    BEGIN
        CASE cntrl IS
            WHEN "000" => aux_result <= (a AND b); --And operation
            WHEN "001" => aux_result <= (a OR b); --Or operation
            WHEN "010" => aux_result <= (a + b); --Add operator
            WHEN "011" => aux_result <= (a); --equal
            WHEN "100" => aux_result <= (b(15 DOWNTO 0) & "0000000000000000");
            WHEN "101" => NULL; --null
            WHEN "110" => aux_result <= (a - b); --subtraction operation
            WHEN "111" =>
                IF a < b THEN --If (a<b), then equal to 1
                    aux_result <= ("00000000000000000000000000000001");
                ELSE --else, equal to 0
                    aux_result <= ("00000000000000000000000000000000");
                END IF;
            WHEN OTHERS => NULL;
        END CASE;
    END PROCESS;

    PROCESS (aux_result)
    BEGIN
        IF (aux_result = "00000000000000000000000000000000") THEN
            zero <= '1';
        ELSE
            zero <= '0';
        END IF;
        result <= aux_result;
    END PROCESS;
END Behavioral;
```

- ***ALU Control***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Alu_Control IS
    PORT (
        funct : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        alu_op : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        alu_ctrl : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END Alu_Control;

ARCHITECTURE Behavioral OF Alu_Control IS
BEGIN
    alu_ctrl <= "010" WHEN (alu_op = "000") OR (alu_op = "001" AND funct = "100000")
        ELSE
        "101" WHEN (alu_op = "001" AND funct = "000000")
        ELSE
        "110" WHEN (alu_op = "001" AND funct = "100010") OR (alu_op = "010")
        ELSE
        "000" WHEN alu_op = "001" AND funct = "100100"
        ELSE
        "001" WHEN (alu_op = "001" AND funct = "100101") OR (alu_op = "011")
        ELSE
        "111" WHEN alu_op = "001" AND funct = "101010"
        ELSE
        "000" WHEN alu_op = "001" AND funct = "001000"
        ELSE
        "100" WHEN alu_op = "100";
END Behavioral;
```

- ***Data Memory Unit (DMU)***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Data_Memory IS
    PORT (
        clk : IN STD_LOGIC;
        MemRead, MemWrite : IN STD_LOGIC;
        Address : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        WriteData : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        ReadData : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END Data_Memory;

ARCHITECTURE Data_Memory OF Data_Memory IS
BEGIN
    PROCESS (clk, MemRead, Address)

        SUBTYPE Register0 IS STD_LOGIC_VECTOR(31 DOWNTO 0);
        TYPE REG_Bank IS ARRAY(31 DOWNTO 0) OF Register0;
        VARIABLE DataMemory : REG_Bank := (
            x"0000_0003",
            x"0000_0004",
            x"0000_0005",
            x"0000_0006",
            OTHERS => (OTHERS => '0')
        );
    BEGIN
        IF falling_edge(clk) THEN
            IF (MemWrite = '1') THEN
                ReadData <= (OTHERS => '0');
                DataMemory(To_Integer(unsigned(Address(6 DOWNTO 2)))) := WriteData;
            END IF;
        END IF;
        IF (MemRead = '1') THEN
            ReadData <= DataMemory(To_Integer(unsigned(Address(6 DOWNTO 2))));
        END IF;
    END PROCESS;
END Data_Memory;
```

- ***Instruction Memory***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY instruction_memory IS
    PORT (
        READ_ADDRESS : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        INSTRUCTION : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END instruction_memory;

ARCHITECTURE Behavioral OF instruction_memory IS
BEGIN

    PROCESS (READ_ADDRESS)
        TYPE ROM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(31 DOWNTO 0);
        VARIABLE ROM_DATA : ROM := (
            "00000000000000001000000100000", -- add r8, r0, r0
            "10001100000000010000000000000000", -- lw r1, 0x0(r0)
            "10001100000001000000000000000100", -- lw r2, 0x4(r0)
            "10001100000001100000000000001000", -- lw r3, 0x8(r0)
            "00000000100000100011000000100000", -- add r3, r2, r1
            "00000000100000100100000000100101", -- or r4, r2, r1
            "00000000100000100101000000100010", -- sub r5, r2, r1
            "00000000100000100110000000100100", -- and r6, r2, r1
            "00100001000010000000000000000001", -- L2: addi r8, r8, 0x01
            "00000000000010000011100000101010", -- L1: slt r7, r0, r8
            "00010000111010001111111111111101", -- beq r7, r8, L2
            "10101101001010000000000000000000", -- sw r8, 0x0(r9)
            "10001101001010100000000000000000", -- lw r10, 0x0(r9)
            OTHERS => (OTHERS => '0'));
    BEGIN
        INSTRUCTION <= ROM_DATA(to_integer(unsigned(READ_ADDRESS(4 DOWNTO 0))));
    END PROCESS;

END Behavioral;
```

- ***Multiplexer (MUX) Units***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux_5bits IS
    PORT (
        selector : IN STD_LOGIC; -- selector line
        input_1 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- input 1
        input_2 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- input 2
        output : OUT STD_LOGIC_VECTOR (4 DOWNTO 0) -- output
    );
END mux_5bits;

ARCHITECTURE Behavioral OF mux_5bits IS
BEGIN
    output <= input_1 WHEN (selector = '0') ELSE
        input_2;
END Behavioral;
```

.....

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux_32bits IS
    PORT (
        selector : IN STD_LOGIC; -- selector line
        input_1 : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- input 1
        input_2 : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- input 2
        output : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) -- output
    );
END mux_32bits;

ARCHITECTURE Behavioral OF mux_32bits IS
BEGIN
    output <= input_1 WHEN (selector = '0') ELSE
        input_2;
END Behavioral;
```

- ***Shift Register***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Shift_left_2 IS
    PORT (
        input : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        output : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END Shift_left_2;

ARCHITECTURE Behavioral OF Shift_left_2 IS

BEGIN
    output <= input(29 DOWNTO 0) & "00";
END Behavioral;
```

- ***Sign Extend Unit***

```
ENTITY SignExtend IS
    PORT (
        INPUT : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
        OUTPUT : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
END SignExtend;

ARCHITECTURE Behavioral OF SignExtend IS
BEGIN
    -- Sign extension logic
    OUTPUT <= "0000000000000000" & INPUT(15 DOWNTO 0);
END Behavioral;
```

- ***Full MIPS Processor System_p1***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY Mips_Processor IS PORT (
    clk : IN STD_LOGIC;
    reset : IN STD_LOGIC);
END Mips_Processor;

ARCHITECTURE behave OF Mips_Processor IS
    SIGNAL pc_Mux, pc_instruction : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL instruction_output : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL opcode : STD_LOGIC_VECTOR(5 DOWNTO 0);
    SIGNAL rs : STD_LOGIC_VECTOR(4 DOWNTO 0);
    SIGNAL rt : STD_LOGIC_VECTOR(4 DOWNTO 0);
    SIGNAL rd : STD_LOGIC_VECTOR(4 DOWNTO 0);
    SIGNAL fun_extend : STD_LOGIC_VECTOR(5 DOWNTO 0);
    SIGNAL immediate : STD_LOGIC_VECTOR(15 DOWNTO 0);
    SIGNAL sign_extend_output : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL write_Data : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL readData1, readData2 : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL mux_Reg : STD_LOGIC_VECTOR(4 DOWNTO 0);
    SIGNAL mux_Alu : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL alu_alu_control : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL zero : STD_LOGIC;
    SIGNAL alu_result : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL shift_adder : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL adder_mux : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL and_out : STD_LOGIC;
    SIGNAL memory_mux : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL MemtoReg, MemWrite, Branch, AluSrc, RegDst, RegWrite, MemRead : STD_LOGIC;
    SIGNAL AluOp : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL next_pc : STD_LOGIC_VECTOR(31 DOWNTO 0);
    COMPONENT PC IS

        PORT (
            clk :
            IN STD_LOGIC;
            reset : IN STD_LOGIC;
            din : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            dout : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
    END COMPONENT;

    COMPONENT instruction_memory IS
        PORT (

            READ_ADDRESS : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            INSTRUCTION : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
    END COMPONENT;
    --Mux 32 bit
    COMPONENT mux_32bits IS
        PORT (
            selector : IN STD_LOGIC; -- selector line
            input_1 : IN STD_LOGIC_VECTOR (31 DOWNTO 0); -- input 1
```

- ***Full MIPS Processor System_p2***

```
    PORT (
        selector : IN STD_LOGIC; -- selector line
        input_1 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- input 1
        input_2 : IN STD_LOGIC_VECTOR (4 DOWNTO 0); -- input 2
        output : OUT STD_LOGIC_VECTOR (4 DOWNTO 0) -- output
    );
END COMPONENT;

--Control unit
COMPONENT ControlUnit IS
    PORT (
        OpCode : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
        --  Funct  : in  STD_LOGIC_VECTOR (5 downto 0);
        MemtoReg : OUT STD_LOGIC;

        MemWrite : OUT STD_LOGIC;
        MemRead : OUT STD_LOGIC;
        Branch : OUT STD_LOGIC;
        AluSrc : OUT STD_LOGIC;
        RegDst : OUT STD_LOGIC;
        RegWrite : OUT STD_LOGIC;
        AluOp : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END COMPONENT;
-- ALU control
COMPONENT Alu_Control IS
    PORT (
        funct : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        alu_op : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        alu_ctrl : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END COMPONENT;

--Data Memory
COMPONENT Data_Memory IS
    PORT (
        clk : IN STD_LOGIC;

        MemRead, MemWrite : IN STD_LOGIC;
        Address : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        WriteData : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        ReadData : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END COMPONENT;
--shift left 2 bit
COMPONENT Shift_left_2 IS
    PORT (

        input : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        output : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END COMPONENT;
-- sign Extend
COMPONENT SignExtend IS
    PORT (
```

- ***Full MIPS Processor System_p3***

```
        PORT (
            a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            cntrl : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
            zero : OUT STD_LOGIC;
            result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
    END COMPONENT;
    COMPONENT adder IS
        PORT (
            a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            c : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
    END COMPONENT;
    COMPONENT Registers IS PORT (
        clk, rst, rw : IN STD_LOGIC;
        readRg1, readRg2, writeRg : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        readData1, readData2 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        writeData : IN STD_LOGIC_VECTOR(31 DOWNTO 0)

        );
    END COMPONENT;

BEGIN

    opcode <= instruction_output(31 DOWNTO 26);
    rs <= instruction_output(25 DOWNTO 21);
    rt <= instruction_output(20 DOWNTO 16);
    rd <= instruction_output(15 DOWNTO 11);
    fun_extend <= instruction_output(5 DOWNTO 0);
    immediate <= instruction_output(15 DOWNTO 0);
    and_out <= zero AND Branch;

    pc_map : PC PORT MAP(clk, reset, next_pc, pc_instruction);
    instruction_map : instruction_memory PORT MAP(pc_instruction, instruction_output);
    control_map : ControlUnit PORT MAP(Opcode, MemtoReg, MemWrite, MemRead, Branch, AluSrc, RegDst, RegWrite, AluOp);
    pc_adder : adder PORT MAP(pc_instruction, "00000000000000000000000000000001", next_pc);
    mux_reg5 : mux_5bits PORT MAP(RegDst, rt, rd, mux_Reg);
    Reg_file : Registers PORT MAP(clk, reset, RegWrite, rs, rt, mux_Reg, readData1, readData2, write_Data);
    sign_extend : SignExtend PORT MAP(immediate, sign_extend_output);
    mux_alu_map : mux_32bits PORT MAP(AluSrc, readData2, sign_extend_output, mux_Alu);
    Alu_contol_map : Alu_Control PORT MAP(fun_extend, AluOp, alu_alu_control);
    Alu_map : alu PORT MAP(readData1, mux_Alu, alu_alu_control, zero, alu_result);
    shift_l : Shift_left_2 PORT MAP(sign_extend_output, shift_adder);
    adder_mux_map : adder PORT MAP(next_pc, shift_adder, adder_mux);

    last_mux_map : mux_32bits PORT MAP(and_out, next_pc, adder_mux, pc_Mux);

    data_memory_map : Data_Memory PORT MAP(clk, MemRead, MemWrite, alu_result, readData2, memory_mux);
    data_memory_mux : mux_32bits PORT MAP(MemtoReg, alu_result, memory_mux, write_Data);

END behave;
```

TESTBENCHES

- ***Program Counter (PC)***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PC_Testbench IS
END PC_Testbench;

ARCHITECTURE Behavioral OF PC_Testbench IS
    COMPONENT PC
        PORT (
            clk : IN STD_LOGIC;
            reset : IN STD_LOGIC;
            din : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            dout : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
    END COMPONENT;
    SIGNAL clk : STD_LOGIC;
    SIGNAL reset : STD_LOGIC;
    SIGNAL din : STD_LOGIC_VECTOR (31 DOWNTO 0);
    SIGNAL dout : STD_LOGIC_VECTOR (31 DOWNTO 0);

BEGIN
    pcc : PC PORT MAP(clk => clk, reset => reset, din => din, dout => dout);
    PROCESS
    BEGIN
        clk <= '1';
        WAIT FOR 100ns;
        Clk <= '0';
        WAIT FOR 100ns;
    END PROCESS;
    PROCESS
    BEGIN
        reset <= '0';
        din <= x"ffffffff";
        WAIT FOR 100ns;
        reset <= '1';
        din <= x"ffffcccc";
        WAIT FOR 100ns;
        reset <= '1';
        din <= x"ffffccca";
        WAIT FOR 100ns;
        reset <= '1';
        din <= x"ffffcccf";
        WAIT FOR 100ns;
    END PROCESS;
END Behavioral;
```

- ***Register File***

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY RegisterFile_tb IS
END RegisterFile_tb;

ARCHITECTURE tb OF RegisterFile_tb IS
    COMPONENT Registers PORT (
        clk, rst, rw : IN STD_LOGIC;
        readRg1, readRg2, writeRg : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        readData1, readData2 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        writeData : IN STD_LOGIC_VECTOR(31 DOWNTO 0));
    END COMPONENT;

    SIGNAL clk, rst, rw : STD_LOGIC;
    SIGNAL readRg1, readRg2, writeRg : STD_LOGIC_VECTOR(4 DOWNTO 0);
    SIGNAL readData1, readData2, writeData : STD_LOGIC_VECTOR(31 DOWNTO 0);
    CONSTANT clk_period : TIME := 100ns;

BEGIN
    -- Clock process
    PROCESS
    BEGIN
        clk <= '0';
        WAIT FOR clk_period / 2;
        clk <= '1';
        WAIT FOR clk_period / 2;
    END PROCESS;

    -- Port the signals to the Register File
    rg : Registers PORT MAP(
        clk => clk,
        rst => rst,
        rw => rw,
        readRg1 => readRg1,
        readRg2 => readRg2,
        writeRg => writeRg,
        readData1 => readData1,
        readData2 => readData2,
        writeData => writeData
    );

    PROCESS
    BEGIN
        rst <= '0';
        rw <= '1';
        writeRg <= "00000"; -- Write at 0
        writeData <= (2 => '1', 0 => '1', OTHERS => '0'); -- "00,,,101" = 5
        readRg1 <= "00000"; -- 0
        readRg2 <= "00001"; -- 1
        WAIT FOR clk_period;
        rw <= '1';
        writeRg <= "00001"; -- Write at 1
        writeData <= (2 => '1', 1 => '1', 0 => '1', OTHERS => '0'); -- "00,,,111" = 7
```

```
        readRg1 <= "00000"; -- 0
        readRg2 <= "00001"; -- 1
        WAIT FOR clk_period;
        rw <= '1';
        writeRg <= "00011"; -- Write at 3
        writeData <= (2 => '1', 1 => '1', 0 => '1', OTHERS => '0'); -- "00,,,111" = 7
        readRg1 <= "00000"; -- 0
        readRg2 <= "00001"; -- 1
        WAIT FOR clk_period;
        rw <= '0';
        writeRg <= "00100"; -- Write at 4 & rw = 0
        writeData <= (2 => '1', 1 => '1', 0 => '1', OTHERS => '0'); -- "00,,,111" = 7
        readRg1 <= "00011"; -- 3
        readRg2 <= "00100"; -- 4
        WAIT FOR clk_period;
        rst <= '1';
        WAIT FOR clk_period;
    END PROCESS;
END tb;
```

- ***Control Unit TruthTable***

Control	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

- ***Control Unit***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY ControlUnit_tb IS
    -- Testbench has no ports
END ControlUnit_tb;

ARCHITECTURE Behavioral OF ControlUnit_tb IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT ControlUnit IS
        PORT (
            OpCode : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
            MemtoReg : OUT STD_LOGIC;
            MemWrite : OUT STD_LOGIC;
            MemRead : OUT STD_LOGIC;
            Branch : OUT STD_LOGIC;
            AluSrc : OUT STD_LOGIC;
            RegDst : OUT STD_LOGIC;
            RegWrite : OUT STD_LOGIC;
            AluOp : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
        );
    END COMPONENT;
    -- Signals to connect to UUT
    SIGNAL OpCode : STD_LOGIC_VECTOR (5 DOWNTO 0) := (OTHERS => '0');
    SIGNAL MemtoReg : STD_LOGIC;
    SIGNAL MemWrite : STD_LOGIC;
    SIGNAL MemRead : STD_LOGIC;
    SIGNAL Branch : STD_LOGIC;
    SIGNAL AluSrc : STD_LOGIC;
    SIGNAL RegDst : STD_LOGIC;
    SIGNAL RegWrite : STD_LOGIC;
    SIGNAL AluOp : STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut : ControlUnit PORT MAP(
        OpCode => OpCode,
        MemtoReg => MemtoReg,
        MemWrite => MemWrite,
        MemRead => MemRead,
        Branch => Branch,
        AluSrc => AluSrc,
        RegDst => RegDst,
        RegWrite => RegWrite,
        AluOp => AluOp
    );

    -- Stimulus process
    stim_proc : PROCESS
    BEGIN
        -- Test case 1: R-type instruction (e.g., jr)
        OpCode <= "000000";
        WAIT FOR 100 ns;
        -- Check outputs expected for R-type instruction
        ASSERT (RegDst = '1' AND Branch = '0' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '0' AND AluSrc = '0' AND RegWrite = '1' AND AluOp = "001") REPORT "Test case 1
failed" SEVERITY error;

        -- Test case 2: Branch on equal (beq)
```

- ***Control Unit***

```
        OpCode <= "000100";
        WAIT FOR 100 ns;
        -- Check outputs expected for beq
        ASSERT (RegDst = '0' AND Branch = '1' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '0' AND AluSrc = '0' AND RegWrite = '0' AND AluOp = "010") REPORT "Test case 2
failed" SEVERITY error;

        -- Test case 3: Add immediate (addi)
        OpCode <= "001000";
        WAIT FOR 100 ns;
        -- Check outputs expected for addi
        ASSERT (RegDst = '0' AND Branch = '0' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '0' AND AluSrc = '1' AND RegWrite = '1' AND AluOp = "000") REPORT "Test case 3
failed" SEVERITY error;

        -- Test case 4: OR immediate (ori)
        OpCode <= "001101";
        WAIT FOR 100 ns;
        -- Check outputs expected for ori
        ASSERT (RegDst = '0' AND Branch = '0' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '0' AND AluSrc = '1' AND RegWrite = '1' AND AluOp = "011") REPORT "Test case 4
failed" SEVERITY error;

        -- Test case 5: Load upper immediate (lui)
        OpCode <= "001111";
        WAIT FOR 100 ns;
        -- Check outputs expected for lui
        ASSERT (RegDst = '0' AND Branch = '0' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '0' AND AluSrc = '1' AND RegWrite = '1' AND AluOp = "100") REPORT "Test case 5
failed" SEVERITY error;

        -- Test case 6: Load word (lw)
        OpCode <= "100011";
        WAIT FOR 100 ns;
        -- Check outputs expected for lw
        ASSERT (RegDst = '0' AND Branch = '0' AND MemRead = '1' AND MemtoReg = '1' AND
MemWrite = '0' AND AluSrc = '1' AND RegWrite = '1' AND AluOp = "000") REPORT "Test case 6
failed" SEVERITY error;

        -- Test case 7: Store word (sw)
        OpCode <= "101011";
        WAIT FOR 100 ns;
        -- Check outputs expected for sw
        ASSERT (RegDst = '0' AND Branch = '0' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '1' AND AluSrc = '1' AND RegWrite = '0' AND AluOp = "000") REPORT "Test case 7
failed" SEVERITY error;
        -- Test case 8: Default case (invalid OpCode)
        OpCode <= "111111";
        WAIT FOR 100 ns;
        -- Check outputs expected for default case
        ASSERT (RegDst = '0' AND Branch = '0' AND MemRead = '0' AND MemtoReg = '0' AND
MemWrite = '0' AND AluSrc = '0' AND RegWrite = '0' AND AluOp = "000") REPORT "Test case 8
failed" SEVERITY error;
```

- ***Arithmetic Logic Unit (ALU)***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_testbench IS
END alu_testbench;

ARCHITECTURE behavior OF alu_testbench IS
   COMPONENT alu
      PORT (
         a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         cntrl : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
         zero : OUT STD_LOGIC;
         result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
      );
   END COMPONENT;
   --inputs
   SIGNAL a : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
   SIGNAL b : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
   SIGNAL cntrl : STD_LOGIC_VECTOR(2 DOWNTO 0) := (OTHERS => '0');
   --outputs
   SIGNAL zero : STD_LOGIC;
   SIGNAL result : STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN
   -- instantiate the Unit Under Test (UUT)
   uut : alu PORT MAP(
      a => a,
      b => b,
      cntrl => cntrl,
      zero => zero,
      result => result
   );
   stim_proc : PROCESS -- Stimulus process
   BEGIN
      -- hold reset state for 100 ns.
      WAIT FOR 100 ns;
      -- insert stimulus here
      cntrl <= "000";
      a <= x"0000_0003";
      b <= x"0000_0002";
      WAIT FOR 40 ns;
      cntrl <= "001";
      WAIT FOR 40 ns;
      cntrl <= "010";
      WAIT FOR 40 ns;
      cntrl <= "011";
      WAIT FOR 40 ns;
      cntrl <= "100";
      WAIT FOR 40 ns;
      cntrl <= "110";
      WAIT FOR 40 ns;
      cntrl <= "111";
      WAIT FOR 40 ns;
      WAIT;
   END PROCESS;
END behavior;
```

- ***ALU Control***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY Tb_Alu_Control IS
END Tb_Alu_Control;

ARCHITECTURE Behavioral OF Tb_Alu_Control IS
    SIGNAL Tb_funct : STD_LOGIC_VECTOR(5 DOWNTO 0) := (OTHERS => '0');
    SIGNAL Tb_alu_op : STD_LOGIC_VECTOR(2 DOWNTO 0) := (OTHERS => '0');
    SIGNAL Tb_alu_ctrl : STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
    utt : ENTITY work.ALU_Control(Behavioral) PORT MAP (
        funct => Tb_funct,
        alu_op => Tb_alu_op,
        alu_ctrl => Tb_alu_ctrl
        );
    PROCESS
    BEGIN
        Tb_alu_op <= "001"; --010
        Tb_funct <= "100000";
        WAIT FOR 50ns;

        Tb_alu_op <= "001"; --101
        Tb_funct <= "000000";
        WAIT FOR 50ns;

        Tb_alu_op <= "001"; --110
        Tb_funct <= "100010";
        WAIT FOR 50ns;

        Tb_alu_op <= "001"; --001
        Tb_funct <= "100101";
        WAIT FOR 50ns;

        Tb_alu_op <= "001"; --111
        Tb_funct <= "101010";
        WAIT FOR 50ns;

    END PROCESS;
END Behavioral;
```

- ***Data Memory Unit (DMU)***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Data_Memory_Testbench IS
END Data_Memory_Testbench;

ARCHITECTURE Data_Memory_Testbench OF Data_Memory_Testbench IS
    COMPONENT Data_Memory
        PORT (
            clk : IN STD_LOGIC;
            MemRead, MemWrite : IN STD_LOGIC;
            Address : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            WriteData : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            ReadData : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
    END COMPONENT;
    SIGNAL clk : STD_LOGIC := '0';
    SIGNAL MemRead, MemWrite : STD_LOGIC := '0';
    SIGNAL Address : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
    SIGNAL WriteData : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
    SIGNAL ReadData : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
    CONSTANT clk_period : TIME := 10 ns;
BEGIN
    tree : Data_Memory PORT MAP(
        clk => clk,
        MemRead => MemRead,
        MemWrite => MemWrite,
        Address => Address,
        WriteData => WriteData,
        ReadData => ReadData
    );
    PROCESS  -- clk process
    BEGIN
        clk <= '0';
        WAIT FOR clk_period/2;
        clk <= '1';
        WAIT FOR clk_period/2;
    END PROCESS;
    PROCESS
    BEGIN
        WAIT FOR clk_period * 10;
        MemRead <= '1';
        MemWrite <= '0';
        Address <= x"0000_0000";
        WriteData <= x"0000_0000";
        WAIT FOR clk_period * 10;
        Address <= x"0000_0004";
        WAIT FOR clk_period * 10;
        MemRead <= '0';
        MemWrite <= '1';
        writeData <= x"0000_1111";
        WAIT FOR clk_period * 10;
        MemWrite <= '0';
        MemRead <= '1';
        writeData <= x"0000_0000";
        WAIT;
    END PROCESS;

END Data_Memory_Testbench;
```

- ***Instruction Memory***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY instruction_memory_testbench IS
END instruction_memory_testbench;

ARCHITECTURE behavior OF instruction_memory_testbench IS

   COMPONENT instruction_memory
      PORT (
         READ_ADDRESS : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         INSTRUCTION : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
      );
   END COMPONENT;
   --inputs
   SIGNAL READ_ADDRESS : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
   --outputs
   SIGNAL INSTRUCTION : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
   -- instantiate the Unit Under Test (UUT)
   uut : instruction_memory PORT MAP(
      READ_ADDRESS => READ_ADDRESS,
      INSTRUCTION => INSTRUCTION
   );
   -- Stimulus process
   stim_proc : PROCESS
   BEGIN
      -- hold reset state for 100 ns.
      WAIT FOR 100 ns;

      READ_ADDRESS <= x"0000_0000";
      WAIT FOR 30 ns;

      READ_ADDRESS <= x"0000_0004";
      WAIT FOR 30 ns;

      READ_ADDRESS <= x"0000_0008";
      WAIT FOR 30 ns;

      READ_ADDRESS <= x"0000_000C";
      WAIT FOR 30 ns;

      READ_ADDRESS <= x"0000_0010";
      WAIT FOR 30 ns;

      READ_ADDRESS <= x"0000_0014";
      WAIT FOR 30 ns;
      WAIT;
   END PROCESS;
END;
```

- ***Multiplexer (MUX) Units***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux_5bits_testBench IS
END mux_5bits_testBench;

ARCHITECTURE behavior OF mux_5bits_testBench IS
   COMPONENT mux_5bits
      PORT (
         selector : IN STD_LOGIC;
         input_1 : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         input_2 : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         output : OUT STD_LOGIC_VECTOR(4 DOWNTO 0)
      );
   END COMPONENT;
   SIGNAL selector : STD_LOGIC := '0';
   SIGNAL input_1 : STD_LOGIC_VECTOR(4 DOWNTO 0) := (OTHERS => '0');
   SIGNAL input_2 : STD_LOGIC_VECTOR(4 DOWNTO 0) := (OTHERS => '0');
   SIGNAL output : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
   uut : mux_5bits PORT MAP(
      selector => selector,
      input_1 => input_1,
      input_2 => input_2,
      output => output
   );
   stim_proc : PROCESS
   BEGIN
      WAIT FOR 20 ns;
      input_1 <= "00001";
      input_2 <= "00010";
      selector <= '0';
      WAIT FOR 20 ns;
      selector <= '1';
      WAIT FOR 20 ns;
      input_2 <= "00011";
      WAIT FOR 20 ns;
      selector <= '0';
      WAIT FOR 20 ns;
      input_1 <= "00100";
      WAIT;
   END PROCESS;
END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY mux_32bits_testBench IS
END mux_32bits_testBench;

ARCHITECTURE behavior OF mux_32bits_testBench IS
   COMPONENT mux_32bits
      PORT (
         selector : IN STD_LOGIC;
         input_1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         input_2 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         output : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
      );
   END COMPONENT;
   SIGNAL selector : STD_LOGIC := '0';
   SIGNAL input_1 : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
   SIGNAL input_2 : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '0');
   SIGNAL output : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
   uut : mux_32bits PORT MAP(
      selector => selector,
      input_1 => input_1,
      input_2 => input_2,
      output => output);
   stim_proc : PROCESS
   BEGIN
      WAIT FOR 20 ns;
      input_1 <= X"0000_0005";
      input_2 <= X"0000_0003";
      selector <= '0';
      WAIT FOR 20 ns;
      selector <= '1';
      WAIT FOR 20 ns;
      input_2 <= X"0000_0007";
      WAIT FOR 20 ns;
      selector <= '0';
      WAIT FOR 20 ns;
      input_1 <= X"0000_0010";
      WAIT;
   END PROCESS;
END;
```

- ***Shift Register***

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Shift_Left_testbench IS
END Shift_Left_testbench;

ARCHITECTURE Behavioral OF Shift_Left_testbench IS
    COMPONENT Shift_left_2
        PORT (
            input : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            output : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
    END COMPONENT;
    SIGNAL input : STD_LOGIC_VECTOR (31 DOWNTO 0);
    SIGNAL output : STD_LOGIC_VECTOR (31 DOWNTO 0);
BEGIN
    ss : Shift_left_2 PORT MAP(input => input, output => output);
    PROCESS
    BEGIN
        input <= X"ffffffff";
        WAIT FOR 100ns;
        input <= X"fffffffC";
        WAIT FOR 100ns;
    END PROCESS;
END Behavioral;
```

- ***Full MIPS Processor System***

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY Mips_testBench IS
END Mips_testBench;

ARCHITECTURE behavior OF Mips_testBench IS
   COMPONENT Mips_Processor
      PORT (
         clk : IN STD_LOGIC;
         reset : IN STD_LOGIC
      );
   END COMPONENT;
   SIGNAL clk : STD_LOGIC := '0';
   SIGNAL reset : STD_LOGIC := '1';
   CONSTANT clk_period : TIME := 100 ns;
BEGIN
   uut : MIPS_processor PORT MAP(
      clk,
      reset
   );
   clk_input : PROCESS
   BEGIN
      clk <= '0';
      WAIT FOR clk_period/2;
      clk <= '1';
      WAIT FOR clk_period/2;
   END PROCESS;
   stimulus_process : PROCESS
   BEGIN
      WAIT FOR clk_period;
      reset <= '0';
      WAIT FOR clk_period * 10;
      reset <= '1';
      WAIT;
   END PROCESS;
END
```

Thank You!