Department of Information Technology

National Institute of Technology Srinagar

Hazratbal, Srinagar, Jammu and Kashmir - 190006, India.

ASSIGNMENT

Microprocessor Lab

Submitted by

KHUSHBOO

2022BITE002

Tavaheed Tariq

2022BITE008

B.TECH. IT(5th semester Section A)

Submit to

Dr. Janibul Bashir

Department of Information Technology

National Institute of Technology, Srinagar

AUTUMN 2024

ASSIGNMENT

Part 1:

1. Design a 4-bit equality comparator circuit that has two 4-bit binary inputs (A and B) and outputs a logic-1 if both inputs are equal.
2. Design a 4-bit adder.
3. Design the circuits for multiplication.
4. Design the circuits for division.

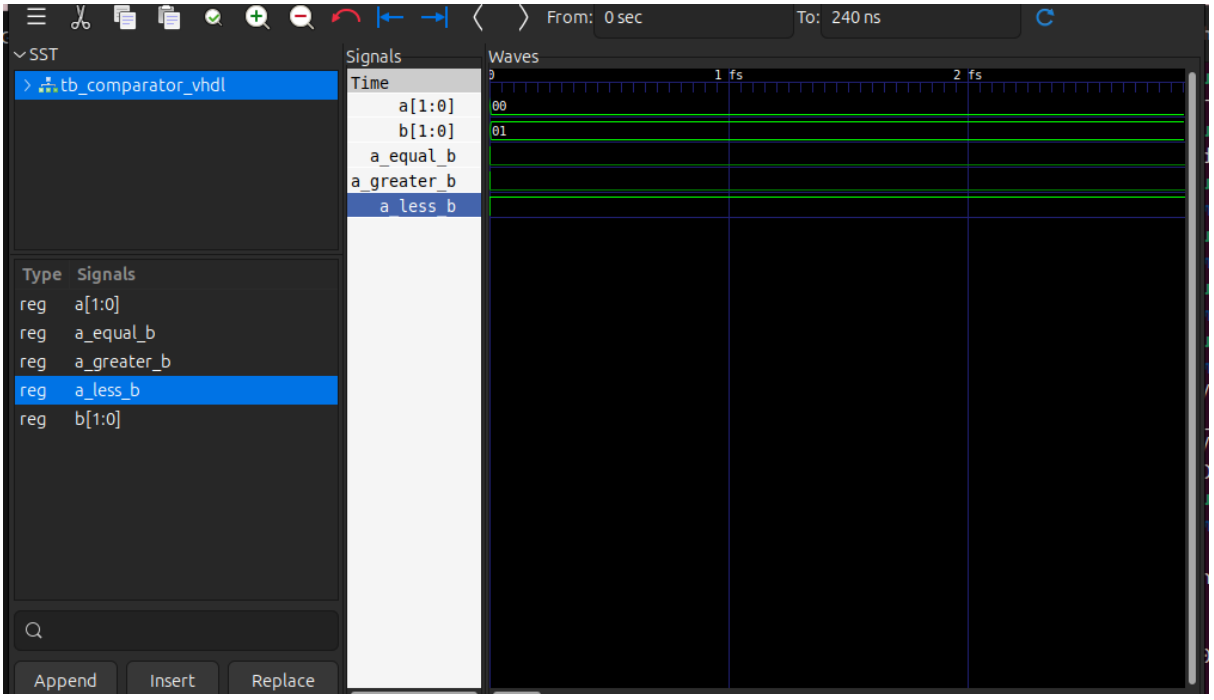
PART 1-1: Design a 4-bit equality comparator circuit that has two 4-bit binary inputs (A and B) and outputs a logic-1 if both inputs are equal.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity comparator_VHDL is
port (
 A,B: in std_logic_vector(1 downto 0); -- two inputs for comparison
 A_less_B: out std_logic; -- '1' if A < B else '0'
 A_equal_B: out std_logic;-- '1' if A = B else '0'
 A_greater_B: out std_logic-- '1' if A > B else '0'
 );
end comparator_VHDL;
architecture comparator_structural of comparator_VHDL is
signal tmp1,tmp2,tmp3,tmp4,tmp5, tmp6, tmp7, tmp8: std_logic;
-- temporary signals
begin
 -- A_equal_B combinational logic circuit
 tmp1 <= A(1) xnor B(1);
 tmp2 <= A(0) xnor B(0);
 A_equal_B <= tmp1 and tmp2;
 -- A_less_B combinational logic circuit
 tmp3 <= (not A(0)) and (not A(1)) and B(0);
 tmp4 <= (not A(1)) and B(1);
 tmp5 <= (not A(0)) and B(1) and B(0);
 A_less_B <= tmp3 or tmp4 or tmp5;
 -- A_greater_B combinational logic circuit
 tmp6 <= (not B(0)) and (not B(1)) and A(0);
 tmp7 <= (not B(1)) and A(1);
 tmp8 <= (not B(0)) and A(1) and A(0);
```

```
 A_greater_B <= tmp6 or tmp7 or tmp8;
end comparator_structural;
```

```
--testbench
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
ENTITY tb_comparator_VHDL IS
END tb_comparator_VHDL;

ARCHITECTURE behavior OF tb_comparator_VHDL IS

    -- Component Declaration for the comparator in VHDL

    COMPONENT comparator_VHDL
    PORT(
         A : IN  std_logic_vector(1 downto 0);
         B : IN  std_logic_vector(1 downto 0);
         A_less_B : OUT  std_logic;
         A_equal_B : OUT  std_logic;
         A_greater_B : OUT  std_logic
        );
    END COMPONENT;
   --Inputs
   signal A : std_logic_vector(1 downto 0) := (others => '0');
   signal B : std_logic_vector(1 downto 0) := (others => '0');
  --Outputs
   signal A_less_B : std_logic;
   signal A_equal_B : std_logic;
   signal A_greater_B : std_logic;
BEGIN

 -- Instantiate the comparator in VHDL
   uut: comparator_VHDL PORT MAP (
          A => A,
          B => B,
          A_less_B => A_less_B,
          A_equal_B => A_equal_B,
          A_greater_B => A_greater_B
        );

   -- Stimulus process
   stim_proc: process
   begin
  -- create test cases for A_less_B
```

```
   for i in 0 to 3 loop
            A <= std_logic_vector(to_unsigned(i,2));
            B <= std_logic_vector(to_unsigned(i+1,2));
            wait for 20 ns;
        end loop;
   -- create test cases for A_greater_B
    for i in 0 to 3 loop
            A <= std_logic_vector(to_unsigned(i+1,2));
            B <= std_logic_vector(to_unsigned(i,2));
            wait for 20 ns;
        end loop;
   -- create test cases for A_equal_B
   for i in 0 to 3 loop
            A <= std_logic_vector(to_unsigned(i,2));
            B <= std_logic_vector(to_unsigned(i,2));
            wait for 20 ns;
        end loop;
        wait;
    end process;

END;
```



Steps to Simulate VHDL Code for Comparator Circuit

1. Compile the VHDL file for the comparator circuit:

```
ghdl -a comparator_VHDL.vhd
```

2. Compile the VHDL file for the comparator testbench:

```
ghdl -a tb_comparator_VHDL
```

3. **Elaborate the Testbench**

```
ghdl -e tb_comparator_VHDL
```

4. **Run the Simulation**

```
ghdl -r tb_comparator_VHDL --vcd=tb_comparator_VHDL.vcd
```

5. **View the Waveform**

```
gtkwave tb_comparator_VHDL.vcd
```

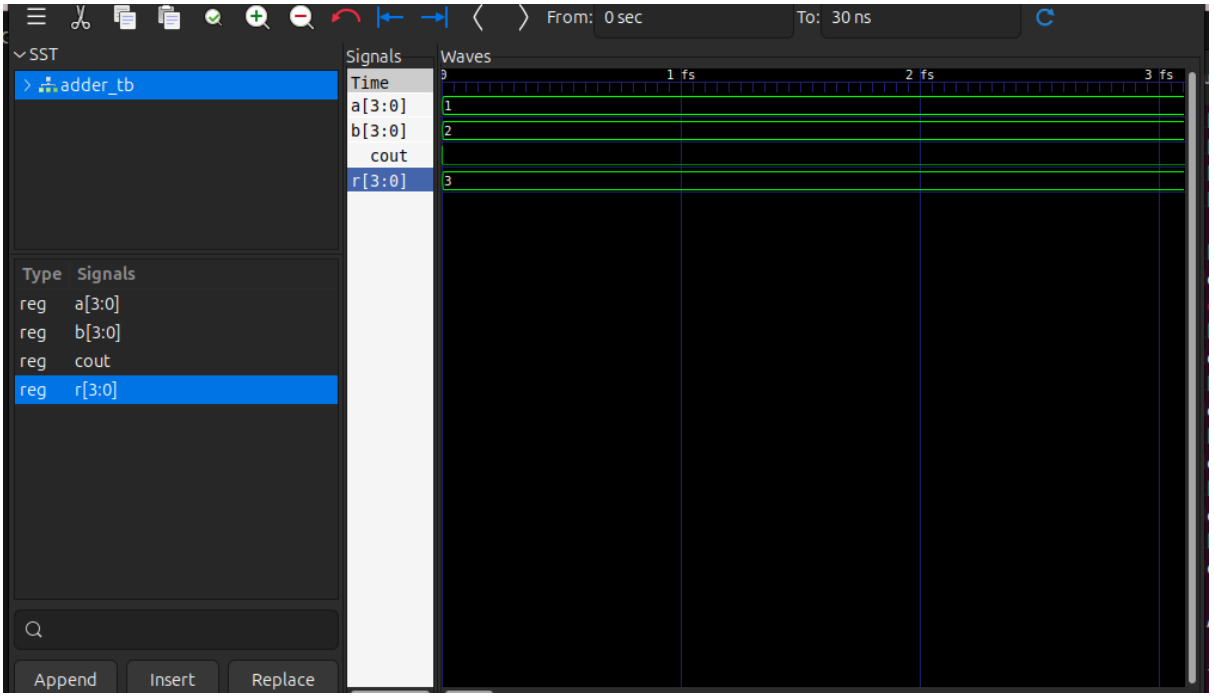
PART 1-2: Design a 4-bit adder.

```
-- FULL ADDER
library ieee;
use ieee.std_logic_1164.all;

entity Full_Adder is
   port(
      X, Y, Cin : in std_logic;
      Sum, Cout : out std_logic
   );
end Full_Adder;

architecture bhv of Full_Adder is
begin
   Sum <= (X xor Y) xor Cin;
   Cout <= (X and Y) or (Cin and (X xor Y));
end bhv;

-- 4-BIT ADDER
```

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
   port(
      A, B : in std_logic_vector(3 downto 0);
      R    : out std_logic_vector(3 downto 0);
      Cout : out std_logic
   );
end adder;

architecture struct of adder is
   component Full_Adder is
      port(
         X, Y, Cin : in std_logic;
         Sum, Cout : out std_logic
      );
   end component;

   signal C1, C2, C3, C4: std_logic;

begin
   FA0: Full_Adder port map(A(0), B(0), '0', R(0), C1); -- Least signific
   FA1: Full_Adder port map(A(1), B(1), C1, R(1), C2);
   FA2: Full_Adder port map(A(2), B(2), C2, R(2), C3);
   FA3: Full_Adder port map(A(3), B(3), C3, R(3), C4); -- Most significar
   Cout <= C4; -- Final carry-out
end struct;
```

```
--tb
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder_tb is
end adder_tb;

architecture behavior of adder_tb is
    -- Signal declarations for inputs and outputs
    signal a, b : std_logic_vector(3 downto 0);
    signal r    : std_logic_vector(3 downto 0);
    signal cout : std_logic;

    -- Component declaration
    component adder is
```

```
        port(
            A, B : in std_logic_vector(3 downto 0);
            R    : out std_logic_vector(3 downto 0);
            Cout : out std_logic
        );
    end component;
begin
    -- Instantiate the adder
    uut: adder
        port map(
            A => a,      -- Match input A
            B => b,      -- Match input B
            R => r,      -- Match output R
            Cout => cout -- Match output Cout
        );

    -- Test process
    stim_proc: process
    begin
        -- Test case 1
        a <= "0001"; b <= "0010";
        wait for 10 ns;

        -- Test case 2
        a <= "0101"; b <= "1011";
        wait for 10 ns;

        -- Test case 3
        a <= "1111"; b <= "0001";
        wait for 10 ns;

        -- End of simulation
        wait;
    end process;
end behavior;
```



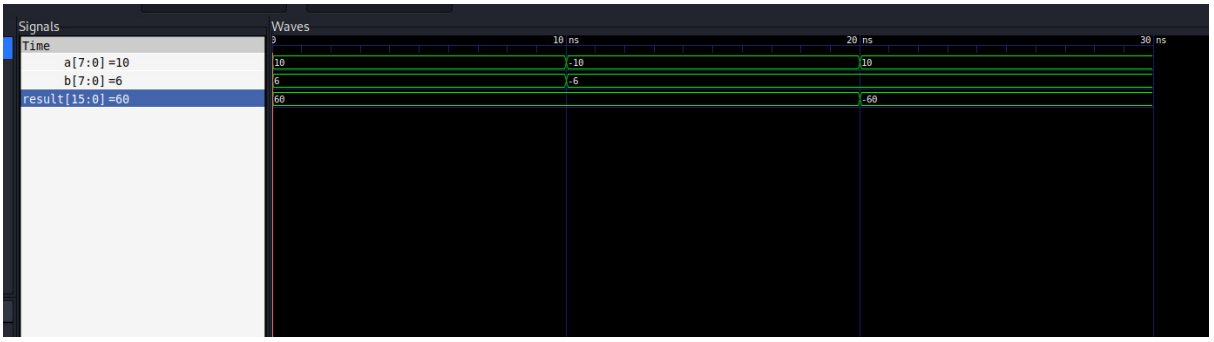
PART 1-3: Design the circuits for multiplication.

SIGNED MULTIPLIER

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_multiplier is
  port (
    a, b : in signed(7 downto 0);
    result : out signed(15 downto 0)
  );
end entity signed_multiplier;

architecture rtl of signed_multiplier is
begin
  process(a, b)
  begin
    result <= resize(a * b, 16);
  end process;
end architecture rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
entity tb_signed_multiplier is
end entity tb_signed_multiplier;

architecture tb of tb_signed_multiplier is
    component signed_multiplier
        port (
            a, b : in signed(7 downto 0);
            result : out signed(15 downto 0)
        );
    end component;

    signal a, b : signed(7 downto 0);
    signal result : signed(15 downto 0);
begin
    dut : signed_multiplier
        port map (
            a => a,
            b => b,
            result => result
        );

    process
    begin
        -- Positive numbers
        a <= "00001010"; -- 10
        b <= "00000110"; -- 6
        wait for 10 ns;
        assert result = "0000000000111100" report "Test failed for 10 * (

        -- Negative numbers
        a <= "11110110"; -- -10
        b <= "11111010"; -- -6
        wait for 10 ns;
        assert result = "0000000000111100" report "Test failed for -10 *

        -- Mixed signs
        a <= "00001010"; -- 10
        b <= "11111010"; -- -6
        wait for 10 ns;
        assert result = "1111111111000100" report "Test failed for 10 *

        wait;
    end process;
end architecture tb;
```

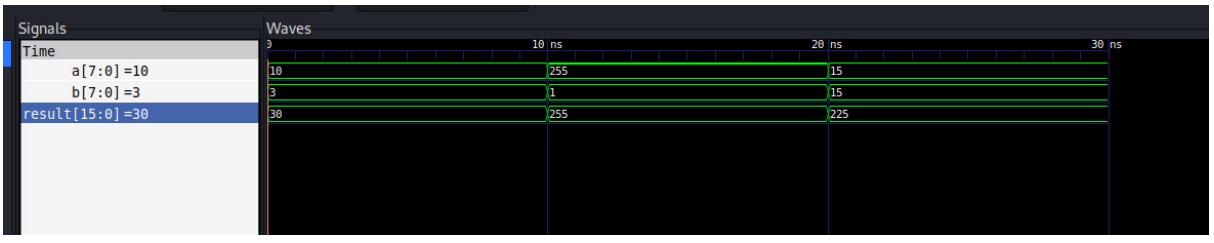


UNSIGNED MULTIPLIER

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity unsigned_multiplier is
    port (
        a, b : in unsigned(7 downto 0);
        result : out unsigned(15 downto 0)
    );
end entity unsigned_multiplier;

architecture rtl of unsigned_multiplier is
begin
    process(a, b)
    begin
        result <= a * b;
    end process;
end architecture rtl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_unsigned_multiplier is
end entity tb_unsigned_multiplier;

architecture tb of tb_unsigned_multiplier is
    component unsigned_multiplier
        port (
            a, b : in unsigned(7 downto 0);
            result : out unsigned(15 downto 0)
```

```
        );
    end component;

    signal a, b : unsigned(7 downto 0);
    signal result : unsigned(15 downto 0);
begin
    DUT : unsigned_multiplier
        port map (
            a => a,
            b => b,
            result => result
        );

    process
    begin
        a <= "00001010"; -- 10
        b <= "00000011"; -- 3
        wait for 10 ns;
        assert result = "0000000000011110" -- 30
            report "Test failed for 10 * 3" severity error;

        a <= "11111111"; -- 255
        b <= "00000001"; -- 1
        wait for 10 ns;
        assert result = "0000000011111111" -- 255
            report "Test failed for 255 * 1" severity error;

        a <= "00001111"; -- 15
        b <= "00001111"; -- 15
        wait for 10 ns;
        assert result = "0000000011100001" -- 225
            report "Test failed for 15 * 15" severity error;

        wait;
    end process;
end architecture tb;
```



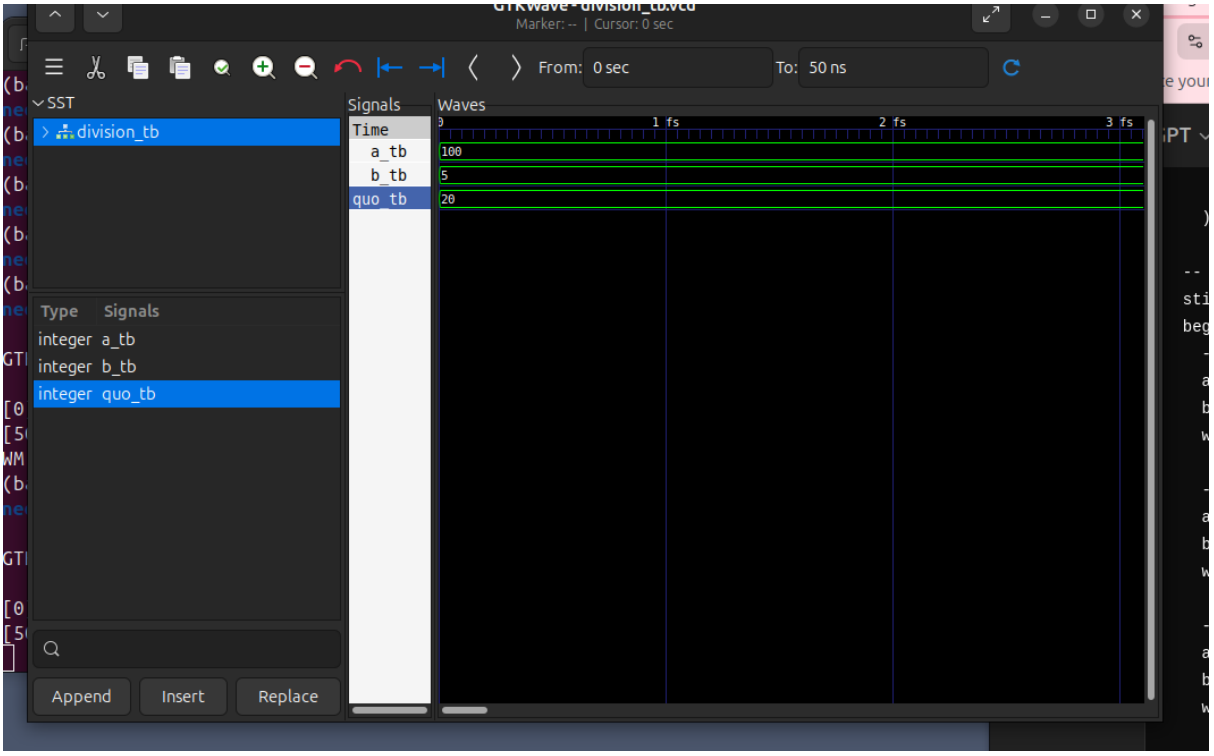
PART 1-4: Design the circuits for division.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity division is
  port (
    a   : in integer range 0 to 255;  -- Dividend
    b   : in integer range 1 to 17;   -- Divisor (range starts from 1)
    quo : out integer range 0 to 255  -- Quotient
  );
end entity division;

architecture structure of division is
begin
  process(a, b)
  begin
    -- Perform division and handle edge cases
    if b /= 0 then
      quo <= a / b; -- Calculate quotient
    else
      quo <= 0;     -- Default output when divisor is zero
    end if;
  end process;
end architecture structure;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity division_tb is
end entity division_tb;

architecture behavior of division_tb is

  -- Component declaration for the unit under test (UUT)
  component division is
    port (
      a   : in integer range 0 to 255;  -- Dividend
      b   : in integer range 1 to 17;   -- Divisor
      quo : out integer range 0 to 255  -- Quotient
```

```
    );
  end component;

  -- Signals to connect to the UUT
  signal a_tb   : integer range 0 to 255 := 0;
  signal b_tb   : integer range 1 to 17 := 1;
  signal quo_tb : integer range 0 to 255;

begin

  -- Instantiate the Unit Under Test (UUT)
  uut: division
    port map (
      a   => a_tb,
      b   => b_tb,
      quo => quo_tb
    );

  -- Stimulus process
  stimulus: process
  begin
    -- Test 1: A = 100, B = 5
    a_tb <= 100;
    b_tb <= 5;
    wait for 10 ns;

    -- Test 2: A = 255, B = 17
    a_tb <= 255;
    b_tb <= 17;
    wait for 10 ns;

    -- Test 3: A = 40, B = 8
    a_tb <= 40;
    b_tb <= 8;
    wait for 10 ns;

    -- Test 4: A = 7, B = 1
    a_tb <= 7;
    b_tb <= 1;
    wait for 10 ns;

    -- Test 5: A = 50, B = 3
    a_tb <= 50;
    b_tb <= 3;
    wait for 10 ns;
```

```
    -- End simulation
    wait;
  end process;

end architecture behavior;
```



Part 2:

1. Implement an Arithmetic Logic Unit (ALU).
   Required: Develop an ALU that takes two 8-bit inputs A and B, and executes the following six instructions: add, sub, and, or, xor, nor.
   The ALU generates an 8-bit output that we call ‘Result’ and an additional 1-bit flag ‘Zero’ that will be set to ‘logic-1’ if all the bits of ‘Result’ are 0.
   Note: Draw a block diagram that will implement the ALU operations listed above.

PART 2: IMPLEMENT AN ALU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.CustomALU_pkg.ALL;
```

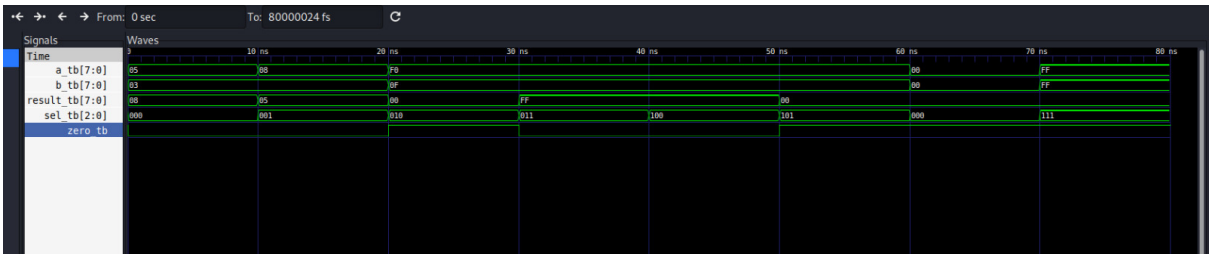
```
entity ALU is
    Port (
        A : in STD_LOGIC_VECTOR (7 downto 0);  -- 8-bit input A
        B : in STD_LOGIC_VECTOR (7 downto 0);  -- 8-bit input B
        sel : in STD_LOGIC_VECTOR (2 downto 0); -- 3-bit select input
        Result : out STD_LOGIC_VECTOR (7 downto 0); -- 8-bit output Resu
        Zero : out STD_LOGIC                 -- Zero flag
    );
end ALU;

architecture Behavioral of ALU is
begin
    process(A, B, sel)
    variable temp_result : STD_LOGIC_VECTOR (7 downto 0);
    begin
        case sel is
            when "000" => -- ADD
                temp_result := add_8bit(A, B);
            when "001" => -- SUBTRACT
                temp_result := sub_8bit(A, B);
            when "010" => -- AND
                temp_result := and_8bit(A, B);
            when "011" => -- OR
                temp_result := or_8bit(A, B);
            when "100" => -- XOR
                temp_result := xor_8bit(A, B);
            when "101" => -- NOR
                temp_result := nor_8bit(A, B);
            when others =>
                temp_result := (others => '0');  -- Default case
        end case;

        Result <= temp_result; -- Assign result to output

        -- Set the Zero flag
        if temp_result = "00000000" then
            Zero <= '1';
        else
            Zero <= '0';
        end if;
    end process;
end Behavioral;
```

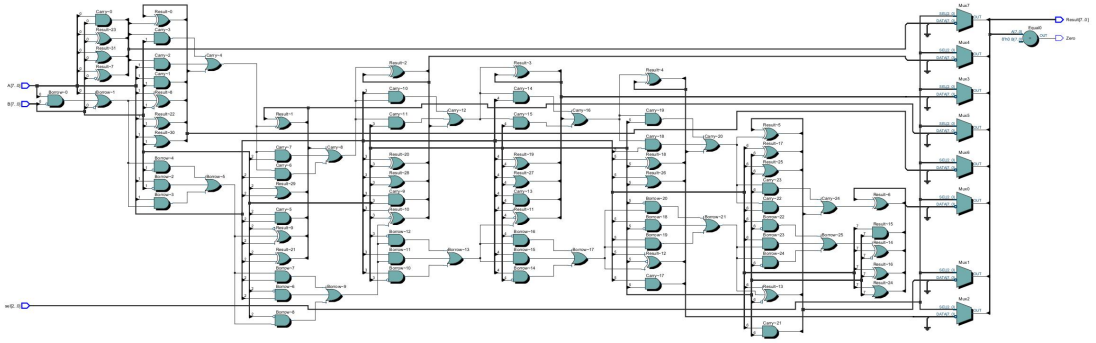
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.NUMERIC_STD.ALL;
use work.CustomALU_pkg.ALL;

entity ALU_tb is
end ALU_tb;

architecture Behavioral of ALU_tb is
    -- Component Declaration
    component ALU
        Port (
            A : in STD_LOGIC_VECTOR (7 downto 0);
            B : in STD_LOGIC_VECTOR (7 downto 0);
            sel : in STD_LOGIC_VECTOR (2 downto 0);
            Result : out STD_LOGIC_VECTOR (7 downto 0);
            Zero : out STD_LOGIC
        );
    end component;

    -- Test Signals
    signal A_tb      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
    signal B_tb      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
    signal sel_tb    : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
    signal Result_tb : STD_LOGIC_VECTOR (7 downto 0);
    signal Zero_tb   : STD_LOGIC;

    -- Helper procedure for printing test results
    procedure print_test_case(
        test_name: string;
        A, B: STD_LOGIC_VECTOR(7 downto 0);
        sel: STD_LOGIC_VECTOR(2 downto 0);
        result: STD_LOGIC_VECTOR(7 downto 0)) is
    begin
        report "Test: " & test_name &
               " | A=" & integer'image(to_integer(unsigned(A))) &
               " | B=" & integer'image(to_integer(unsigned(B))) &
               " | Result=" & integer'image(to_integer(unsigned(result))
    end procedure;

begin
    -- Unit Under Test (UUT)
    UUT: ALU port map (
        A => A_tb,
        B => B_tb,
        sel => sel_tb,
        Result => Result_tb,
```

```
        Zero => Zero_tb
    );

    -- Stimulus Process
    stim_proc: process
    begin
        -- Wait for 10ns for global reset
        wait for 10 ns;

        -- Test Case 1: Addition (000)
        A_tb <= "00000101";  -- 5
        B_tb <= "00000011";  -- 3
        sel_tb <= "000";     -- ADD
        wait for 10 ns;
        print_test_case("Addition", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 2: Subtraction (001)
        A_tb <= "00001000";  -- 8
        B_tb <= "00000011";  -- 3
        sel_tb <= "001";     -- SUB
        wait for 10 ns;
        print_test_case("Subtraction", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 3: AND (010)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "010";     -- AND
        wait for 10 ns;
        print_test_case("AND", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 4: OR (011)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "011";     -- OR
        wait for 10 ns;
        print_test_case("OR", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 5: XOR (100)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "100";     -- XOR
```

```
        wait for 10 ns;
        print_test_case("XOR", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 6: NOR (101)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "101";     -- NOR
        wait for 10 ns;
        print_test_case("NOR", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 7: Zero flag test
        A_tb <= "00000000";
        B_tb <= "00000000";
        sel_tb <= "000";     -- ADD
        wait for 10 ns;
        print_test_case("Zero Flag", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 8: Carry operation
        A_tb <= "11111111";
        B_tb <= "11111111";
        sel_tb <= "111";     -- Carry operation
        wait for 10 ns;
        print_test_case("Carry Op", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- End simulation
        report "Simulation completed successfully";
        wait;
    end process;

end Behavioral;
```





Part 3:

1. Design an in-order processor, in which the main execution unit , that is , ALU is at least able to perform functions listed in Part 2 of the assignment

PART 3: INORDER PROCESSOR

Directory structure

```
inorder_processor/
├── src/
│   ├── ALU.vhd
│   ├── CustomALU_pkg.vhd
│   ├── DataMem.vhd
│   ├── InstrMem.vhd
│   ├── Processor.vhd
│   └── Top.vhd
├── tb/
│   ├── ALU_tb.vhd
│   ├── Processor_tb.vhd
│   └── Top_tb.vhd
├── work/
│   ├── processor_tb.vcd
│   ├── top_tb.vcd
│   ├── processor_wave.gtkw
│   └── top_wave.gtkw
├── simulate.sh
└── view_waves.sh
```

ALU.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.CustomALU_pkg.ALL;

entity ALU is
    Port (
        A : in STD_LOGIC_VECTOR (7 downto 0);  -- 8-bit input A
        B : in STD_LOGIC_VECTOR (7 downto 0);  -- 8-bit input B
        sel : in STD_LOGIC_VECTOR (2 downto 0); -- 3-bit select input
        Result : out STD_LOGIC_VECTOR (7 downto 0); -- 8-bit output Resu
        Zero : out STD_LOGIC                -- Zero flag
    );
end ALU;
```

```
architecture Behavioral of ALU is
begin
    process(A, B, sel)
    variable temp_result : STD_LOGIC_VECTOR (7 downto 0);
    begin
        case sel is
            when "000" => -- ADD
                temp_result := add_8bit(A, B);
            when "001" => -- SUBTRACT
                temp_result := sub_8bit(A, B);
            when "010" => -- AND
                temp_result := and_8bit(A, B);
            when "011" => -- OR
                temp_result := or_8bit(A, B);
            when "100" => -- XOR
                temp_result := xor_8bit(A, B);
            when "101" => -- NOR
                temp_result := nor_8bit(A, B);
            when others =>
                temp_result := (others => '0');  -- Default case
        end case;

        Result <= temp_result; -- Assign result to output

        -- Set the Zero flag
        if temp_result = "00000000" then
            Zero <= '1';
        else
            Zero <= '0';
        end if;
    end process;
end Behavioral;
```

CustomALU_Pkg.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package CustomALU_Pkg is
    function add_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOGI
    function sub_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOGI
    function and_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOGI
    function or_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOGIC
    function xor_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOGI
```

```
    function nor_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOG
end CustomALU_Pkg;

package body CustomALU_Pkg is
    -- Addition Function
    function add_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOG
        variable Result: STD_LOGIC_VECTOR(7 downto 0);
        variable Carry: STD_LOGIC_VECTOR(8 downto 0);  -- One extra bit
    begin
        Carry(0) := '0';  -- Initial carry is 0

        for i in 0 to 7 loop
            Result(i) := A(i) xor B(i) xor Carry(i);
            Carry(i+1) := (A(i) and B(i)) or (A(i) and Carry(i)) or (B(i
        end loop;

        return Result;
    end function;

    -- Subtraction Function
    function sub_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOG
        variable Result: STD_LOGIC_VECTOR(7 downto 0);
        variable Borrow: STD_LOGIC_VECTOR(8 downto 0);  -- One extra bit
        variable B_comp: STD_LOGIC_VECTOR(7 downto 0);  -- Two's compleme
    begin
        -- Create two's complement of B
        B_comp := not B;
        Borrow(0) := '1';  -- Add 1 for two's complement

        for i in 0 to 7 loop
            Result(i) := A(i) xor B_comp(i) xor Borrow(i);
            Borrow(i+1) := (A(i) and B_comp(i)) or (A(i) and Borrow(i)) 
        end loop;

        return Result;
    end function;

    -- Bitwise AND (no changes needed)
    function and_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOG
        variable Result: STD_LOGIC_VECTOR(7 downto 0);
    begin
        for i in 7 downto 0 loop
            Result(i) := A(i) and B(i);
        end loop;
        return Result;
```

```
    end function;

    -- Bitwise OR (no changes needed)
    function or_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOGI
        variable Result: STD_LOGIC_VECTOR(7 downto 0);
    begin
        for i in 7 downto 0 loop
            Result(i) := A(i) or B(i);
        end loop;
        return Result;
    end function;

    -- Bitwise XOR (no changes needed)
    function xor_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOG
        variable Result: STD_LOGIC_VECTOR(7 downto 0);
    begin
        for i in 7 downto 0 loop
            Result(i) := A(i) xor B(i);
        end loop;
        return Result;
    end function;

    -- Bitwise NOR (no changes needed)
    function nor_8bit(A, B: STD_LOGIC_VECTOR(7 downto 0)) return STD_LOG
        variable Result: STD_LOGIC_VECTOR(7 downto 0);
    begin
        for i in 7 downto 0 loop
            Result(i) := not (A(i) or B(i));
        end loop;
        return Result;
    end function;
end CustomALU_Pkg;
```

DataMem.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DataMem is
    Port (
        clk : in STD_LOGIC;
        addr : in STD_LOGIC_VECTOR(7 downto 0);
        data_in : in STD_LOGIC_VECTOR(7 downto 0);
        data_out : out STD_LOGIC_VECTOR(7 downto 0);
```

```
        wr_en : in STD_LOGIC
    );
end DataMem;

architecture Behavioral of DataMem is
    type mem_type is array (0 to 255) of STD_LOGIC_VECTOR(7 downto 0);
    signal memory : mem_type := (others => (others => '0'));
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if wr_en = '1' then
                memory(to_integer(unsigned(addr))) <= data_in;
            end if;
            data_out <= memory(to_integer(unsigned(addr)));
        end if;
    end process;
end Behavioral;
```

InstrMem.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity InstrMem is
    Port (
        clk : in STD_LOGIC;
        addr : in STD_LOGIC_VECTOR(7 downto 0);
        instr : out STD_LOGIC_VECTOR(31 downto 0)
    );
end InstrMem;

architecture Behavioral of InstrMem is
    type mem_type is array (0 to 255) of STD_LOGIC_VECTOR(31 downto 0);
    signal memory : mem_type := (
        -- Example program
        0 => "00000000000000110000001000000000",  -- ADD R3, R2
        1 => "00100000000001000000001100000000",  -- SUB R4, R3
        2 => "01000000000001010000010000000000",  -- AND R5, R4
        3 => "01100000000001100000010100000000",  -- OR  R6, R5
        4 => "10000000000001110000011000000000",  -- XOR R7, R6
        5 => "10100000000010000000011100000000",  -- NOR R8, R7
        others => (others => '0')
    );
```

```
begin
    process(clk)
    begin
        if rising_edge(clk) then
            instr <= memory(to_integer(unsigned(addr)));
        end if;
    end process;
end Behavioral;
```

Processor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity Processor is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;

        -- Instruction Memory Interface
        instr_addr : out STD_LOGIC_VECTOR (7 downto 0);
        instr : in STD_LOGIC_VECTOR (31 downto 0);

        -- Data Memory Interface
        data_addr : out STD_LOGIC_VECTOR (7 downto 0);
        data_in : out STD_LOGIC_VECTOR (7 downto 0);
        data_out : in STD_LOGIC_VECTOR (7 downto 0);
        data_wr : out STD_LOGIC
    );
end Processor;

architecture Behavioral of Processor is
    -- Components
    component ALU
        Port (
            A : in STD_LOGIC_VECTOR (7 downto 0);
            B : in STD_LOGIC_VECTOR (7 downto 0);
            sel : in STD_LOGIC_VECTOR (2 downto 0);
            Result : out STD_LOGIC_VECTOR (7 downto 0);
            Zero : out STD_LOGIC
        );
    end component;

    -- Processor Signals
```

```
    signal PC : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
    signal Inst_Reg : STD_LOGIC_VECTOR (31 downto 0);
    signal Reg_A, Reg_B : STD_LOGIC_VECTOR (7 downto 0);
    signal ALU_Res : STD_LOGIC_VECTOR (7 downto 0);
    signal ALU_Ctrl : STD_LOGIC_VECTOR (2 downto 0);
    signal Zero_Flag : STD_LOGIC;

begin
    -- Instruction Fetch
    instr_addr <= PC;
    Inst_Reg <= instr;

    -- Instruction Decode
    process(Inst_Reg)
    begin
        case Inst_Reg(31 downto 29) is
            when "000" => ALU_Ctrl <= "000"; -- ADD
            when "001" => ALU_Ctrl <= "001"; -- SUB
            when "010" => ALU_Ctrl <= "010"; -- AND
            when "011" => ALU_Ctrl <= "011"; -- OR
            when "100" => ALU_Ctrl <= "100"; -- XOR
            when "101" => ALU_Ctrl <= "101"; -- NOR
            when others => ALU_Ctrl <= "111"; -- Invalid
        end case;

        Reg_A <= Inst_Reg(23 downto 16);
        Reg_B <= Inst_Reg(15 downto 8);
    end process;

    -- ALU Execution
    ALU_Unit: ALU
        port map (
            A => Reg_A,
            B => Reg_B,
            sel => ALU_Ctrl,
            Result => ALU_Res,
            Zero => Zero_Flag
        );

    -- Memory Access
    data_addr <= ALU_Res(7 downto 0);
    data_in <= Reg_B;
    data_wr <= '1' when ALU_Ctrl = "000" else '0'; -- Write for ADD only

    -- Program Counter Update
```

```
    process(clk, reset)
    begin
        if reset = '1' then
            PC <= (others => '0');
        elsif rising_edge(clk) then
            PC <= std_logic_vector(unsigned(PC) + 1);
        end if;
    end process;

end Behavioral;
```

Top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Top is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC
    );
end Top;

architecture Behavioral of Top is
    -- Component declarations
    component Processor is
        Port (
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            instr_addr : out STD_LOGIC_VECTOR(7 downto 0);
            instr : in STD_LOGIC_VECTOR(31 downto 0);
            data_addr : out STD_LOGIC_VECTOR(7 downto 0);
            data_in : out STD_LOGIC_VECTOR(7 downto 0);
            data_out : in STD_LOGIC_VECTOR(7 downto 0);
            data_wr : out STD_LOGIC
        );
    end component;

    component InstrMem is
        Port (
            clk : in STD_LOGIC;
            addr : in STD_LOGIC_VECTOR(7 downto 0);
            instr : out STD_LOGIC_VECTOR(31 downto 0)
        );
    end component;
```

```
    component DataMem is
        Port (
            clk : in STD_LOGIC;
            addr : in STD_LOGIC_VECTOR(7 downto 0);
            data_in : in STD_LOGIC_VECTOR(7 downto 0);
            data_out : out STD_LOGIC_VECTOR(7 downto 0);
            wr_en : in STD_LOGIC
        );
    end component;

    -- Internal signals
    signal instr_addr : STD_LOGIC_VECTOR(7 downto 0);
    signal instruction : STD_LOGIC_VECTOR(31 downto 0);
    signal data_addr : STD_LOGIC_VECTOR(7 downto 0);
    signal data_to_mem : STD_LOGIC_VECTOR(7 downto 0);
    signal data_from_mem : STD_LOGIC_VECTOR(7 downto 0);
    signal data_wr : STD_LOGIC;

begin
    -- Processor instantiation
    proc: Processor
        port map (
            clk => clk,
            reset => reset,
            instr_addr => instr_addr,
            instr => instruction,
            data_addr => data_addr,
            data_in => data_to_mem,
            data_out => data_from_mem,
            data_wr => data_wr
        );

    -- Instruction Memory instantiation
    imem: InstrMem
        port map (
            clk => clk,
            addr => instr_addr,
            instr => instruction
        );

    -- Data Memory instantiation
    dmem: DataMem
        port map (
            clk => clk,
```

```
            addr => data_addr,
            data_in => data_to_mem,
            data_out => data_from_mem,
            wr_en => data_wr
        );

end Behavioral;
```

ALU_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.CustomALU_pkg.ALL;

entity ALU_tb is
end ALU_tb;

architecture Behavioral of ALU_tb is
    -- Component Declaration
    component ALU
        Port (
            A : in STD_LOGIC_VECTOR (7 downto 0);
            B : in STD_LOGIC_VECTOR (7 downto 0);
            sel : in STD_LOGIC_VECTOR (2 downto 0);
            Result : out STD_LOGIC_VECTOR (7 downto 0);
            Zero : out STD_LOGIC
        );
    end component;

    -- Test Signals
    signal A_tb      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
    signal B_tb      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
    signal sel_tb    : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
    signal Result_tb : STD_LOGIC_VECTOR (7 downto 0);
    signal Zero_tb   : STD_LOGIC;

    -- Helper procedure for printing test results
    procedure print_test_case(
        test_name: string;
        A, B: STD_LOGIC_VECTOR(7 downto 0);
        sel: STD_LOGIC_VECTOR(2 downto 0);
        result: STD_LOGIC_VECTOR(7 downto 0)) is
    begin
        report "Test: " & test_name &
```

```
                " | A=" & integer'image(to_integer(unsigned(A))) &
                " | B=" & integer'image(to_integer(unsigned(B))) &
                " | Result=" & integer'image(to_integer(unsigned(result))
    end procedure;

begin
    -- Unit Under Test (UUT)
    UUT: ALU port map (
        A => A_tb,
        B => B_tb,
        sel => sel_tb,
        Result => Result_tb,
        Zero => Zero_tb
    );

    -- Stimulus Process
    stim_proc: process
    begin
        -- Wait for 10ns for global reset
        wait for 10 ns;

        -- Test Case 1: Addition (000)
        A_tb <= "00000101";  -- 5
        B_tb <= "00000011";  -- 3
        sel_tb <= "000";     -- ADD
        wait for 10 ns;
        print_test_case("Addition", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 2: Subtraction (001)
        A_tb <= "00001000";  -- 8
        B_tb <= "00000011";  -- 3
        sel_tb <= "001";     -- SUB
        wait for 10 ns;
        print_test_case("Subtraction", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 3: AND (010)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "010";     -- AND
        wait for 10 ns;
        print_test_case("AND", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;
```

```
        -- Test Case 4: OR (011)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "011";     -- OR
        wait for 10 ns;
        print_test_case("OR", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 5: XOR (100)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "100";     -- XOR
        wait for 10 ns;
        print_test_case("XOR", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 6: NOR (101)
        A_tb <= "11110000";
        B_tb <= "00001111";
        sel_tb <= "101";     -- NOR
        wait for 10 ns;
        print_test_case("NOR", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 7: Zero flag test
        A_tb <= "00000000";
        B_tb <= "00000000";
        sel_tb <= "000";     -- ADD
        wait for 10 ns;
        print_test_case("Zero Flag", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- Test Case 8: Carry operation
        A_tb <= "11111111";
        B_tb <= "11111111";
        sel_tb <= "111";     -- Carry operation
        wait for 10 ns;
        print_test_case("Carry Op", A_tb, B_tb, sel_tb, Result_tb);
        wait for 3 fs;

        -- End simulation
        report "Simulation completed successfully";
        wait;
    end process;
```

```
end Behavioral;
```

Processor_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Processor_tb is
end Processor_tb;

architecture Behavioral of Processor_tb is
    -- Component Declaration
    component Processor
        Port (
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            instr_addr : out STD_LOGIC_VECTOR (7 downto 0);
            instr : in STD_LOGIC_VECTOR (31 downto 0);
            data_addr : out STD_LOGIC_VECTOR (7 downto 0);
            data_in : out STD_LOGIC_VECTOR (7 downto 0);
            data_out : in STD_LOGIC_VECTOR (7 downto 0);
            data_wr : out STD_LOGIC
        );
    end component;

    -- Test Signals
    signal clk : STD_LOGIC := '0';
    signal reset : STD_LOGIC := '0';
    signal instr_addr : STD_LOGIC_VECTOR (7 downto 0);
    signal instr : STD_LOGIC_VECTOR (31 downto 0);
    signal data_addr : STD_LOGIC_VECTOR (7 downto 0);
    signal data_in : STD_LOGIC_VECTOR (7 downto 0);
    signal data_out : STD_LOGIC_VECTOR (7 downto 0);
    signal data_wr : STD_LOGIC;

    -- Clock period definition
    constant CLK_PERIOD : time := 10 ns;

begin
    -- Instantiate the Unit Under Test (UUT)
    UUT: Processor port map (
        clk => clk,
        reset => reset,
```

```
        instr_addr => instr_addr,
        instr => instr,
        data_addr => data_addr,
        data_in => data_in,
        data_out => data_out,
        data_wr => data_wr
    );

    -- Clock process
    clk_process: process
    begin
        clk <= '0';
        wait for CLK_PERIOD/2;
        clk <= '1';
        wait for CLK_PERIOD/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- Initialize
        reset <= '1';
        data_out <= x"00";
        wait for CLK_PERIOD*2;

        -- Release reset
        reset <= '0';
        wait for CLK_PERIOD;

        -- Test ADD instruction
        instr <= "00000000" & x"FF" & x"01" & x"00"; -- ADD FF + 01
        wait for CLK_PERIOD;

        -- Test SUB instruction
        instr <= "00100000" & x"FF" & x"01" & x"00"; -- SUB FF - 01
        wait for CLK_PERIOD;

        -- Test AND instruction
        instr <= "01000000" & x"FF" & x"0F" & x"00"; -- AND FF & 0F
        wait for CLK_PERIOD;

        -- Test OR instruction
        instr <= "01100000" & x"F0" & x"0F" & x"00"; -- OR F0 | 0F
        wait for CLK_PERIOD;
```

```
        -- Test XOR instruction
        instr <= "10000000" & x"FF" & x"55" & x"00"; -- XOR FF ^ 55
        wait for CLK_PERIOD;

        -- Test NOR instruction
        instr <= "10100000" & x"FF" & x"00" & x"00"; -- NOR FF NOR 00
        wait for CLK_PERIOD;

        -- Wait for a few more cycles
        wait for CLK_PERIOD*4;

        -- Test reset again
        reset <= '1';
        wait for CLK_PERIOD*2;
        reset <= '0';

        wait;
    end process;
end Behavioral;
```

Top_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Top_tb is
end Top_tb;

architecture Behavioral of Top_tb is
    -- Component Declaration
    component Top
        Port (
            clk : in STD_LOGIC;
            reset : in STD_LOGIC
        );
    end component;

    -- Test Signals
    signal clk : STD_LOGIC := '0';
    signal reset : STD_LOGIC := '0';

    -- Clock period definition
    constant CLK_PERIOD : time := 10 ns;
```

```
begin
    -- Instantiate the Unit Under Test (UUT)
    UUT: Top port map (
        clk => clk,
        reset => reset
    );

    -- Clock process
    clk_process: process
    begin
        clk <= '0';
        wait for CLK_PERIOD/2;
        clk <= '1';
        wait for CLK_PERIOD/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- Initialize with reset
        reset <= '1';
        wait for CLK_PERIOD*4;

        -- Release reset and let it run
        reset <= '0';
        wait for CLK_PERIOD*20;  -- Run for 20 clock cycles

        -- Apply reset again
        reset <= '1';
        wait for CLK_PERIOD*2;
        reset <= '0';

        -- Let it run some more
        wait for CLK_PERIOD*20;

        -- End simulation
        wait;
    end process;
end Behavioral;
```

simulate.sh

```
#!/bin/bash
```

```
# Create work directory if it doesn't exist
mkdir -p work
cd work

# Clean up any existing files
rm -f *.cf *.vcd *.o

# Analysis (compilation) phase
echo "Compiling VHDL files..."
ghdl -a --workdir=. ../src/CustomALU_Pkg.vhd
ghdl -a --workdir=. ../src/ALU.vhd
ghdl -a --workdir=. ../src/InstrMem.vhd
ghdl -a --workdir=. ../src/DataMem.vhd
ghdl -a --workdir=. ../src/Processor.vhd
ghdl -a --workdir=. ../src/Top.vhd
ghdl -a --workdir=. ../tb/Processor_tb.vhd
ghdl -a --workdir=. ../tb/Top_tb.vhd

# Elaborate phase
echo "Elaborating testbenches..."
ghdl -e --workdir=. Processor_tb
ghdl -e --workdir=. Top_tb

# Run simulation and generate VCD file for Processor testbench
echo "Running Processor simulation..."
ghdl -r --workdir=. Processor_tb --vcd=processor_tb.vcd --stop-time=200ns

# Run simulation and generate VCD file for Top testbench
echo "Running Top simulation..."
ghdl -r --workdir=. Top_tb --vcd=top_tb.vcd --stop-time=500ns

echo "Simulation complete. VCD files generated."
echo "To view waveforms, run:"
echo "gtkwave work/processor_tb.vcd"
echo "gtkwave work/top_tb.vcd"

cd ..
```

view_waves.sh

```
#!/bin/bash

# View processor testbench waveform
echo "Opening Processor testbench waveform..."
gtkwave -A work/processor_tb.vcd processor_wave.gtkw &
```

```
# Wait a bit before opening the second window
sleep 1

# View top testbench waveform
echo "Opening Top testbench waveform..."
gtkwave -A work/top_tb.vcd top_wave.gtkw &
```

To compile the processor run :

```
./simulate.sh
```

and to view waves run:

```
 ./view_waves.sh
```