

**Πανεπιστήμιο Δυτικής Αττικής**

**Σχολή Μηχανικών**

**Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών**

**Σχεδίαση Ψηφιακών Συστημάτων**

**Εργασία Εξαμήνου**

***Στοιχεία Ιδρύματος***

Ίδρυμα: Πανεπιστήμιο Δυτικής Αττικής

Τμήμα: Σχολή Μηχανικών Πληροφορικής και Υπολογιστών

Μάθημα: Σχεδίαση Ψηφιακών Συστημάτων (ICE-4005)(Θεωρία)

Αίθουσα Θεωρίας: Πανεπιστημιούπολη 1 Κ16.113

Εξάμηνο: Εαρινό

***Στοιχεία Φοιτητή***

Ονοματεπώνυμο: Πανταζόπουλος Παναγιώτης

Εξάμηνο: 4ο

ΑΜ: 22390174

Τμήμα: 2

Προθεσμία Παράδοσης: 17/06

**ALU.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ALU is

Port (

A : in STD\_LOGIC\_VECTOR (31 downto 0);

B : in STD\_LOGIC\_VECTOR (31 downto 0);

ALUOp : in STD\_LOGIC\_VECTOR (2 downto 0);

Result : out STD\_LOGIC\_VECTOR (31 downto 0);

Zero : out STD\_LOGIC

);

end ALU;

architecture Behavioral of ALU is

begin

process(A, B, ALUOp)

variable signed\_A : signed(31 downto 0);

variable signed\_B : signed(31 downto 0);

variable temp\_result : signed(31 downto 0);

begin

signed\_A := signed(A);

signed\_B := signed(B);

case ALUOp is

when "000" =>

temp\_result := signed\_A + signed\_B; -- ADD

when "001" =>

temp\_result := signed\_A - signed\_B; -- SUB

when others =>

temp\_result := (others => '0'); -- Default case

end case;

Result <= std\_logic\_vector(temp\_result); -- Assign internal result to output port

-- Set Zero flag

if temp\_result = 0 then

Zero <= '1';

else

Zero <= '0';

end if;

end process;

end Behavioral;

**ALU\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ALU\_tb is

end ALU\_tb;

architecture Behavioral of ALU\_tb is

signal A, B : STD\_LOGIC\_VECTOR(31 downto 0);

signal ALUOp : STD\_LOGIC\_VECTOR(2 downto 0);

signal Result : STD\_LOGIC\_VECTOR(31 downto 0);

signal Zero : STD\_LOGIC;

component ALU

Port ( A : in STD\_LOGIC\_VECTOR(31 downto 0);

B : in STD\_LOGIC\_VECTOR(31 downto 0);

ALUOp : in STD\_LOGIC\_VECTOR(2 downto 0);

Result : out STD\_LOGIC\_VECTOR(31 downto 0);

Zero : out STD\_LOGIC);

end component;

begin

UUT: ALU

Port map(

A => A,

B => B,

ALUOp => ALUOp,

Result => Result,

Zero => Zero

);

stimulus\_process: process

begin

-- Test 5 + (-4)

A <= std\_logic\_vector(to\_signed(5, 32));

B <= std\_logic\_vector(to\_signed(-4, 32));

ALUOp <= "000";

wait for 10 ps;

-- Test 5 + (-5)

A <= std\_logic\_vector(to\_signed(5, 32));

B <= std\_logic\_vector(to\_signed(-5, 32));

ALUOp <= "000";

wait for 10 ps;

-- Test 7 - 8

A <= std\_logic\_vector(to\_signed(7, 32));

B <= std\_logic\_vector(to\_signed(8, 32));

ALUOp <= "001";

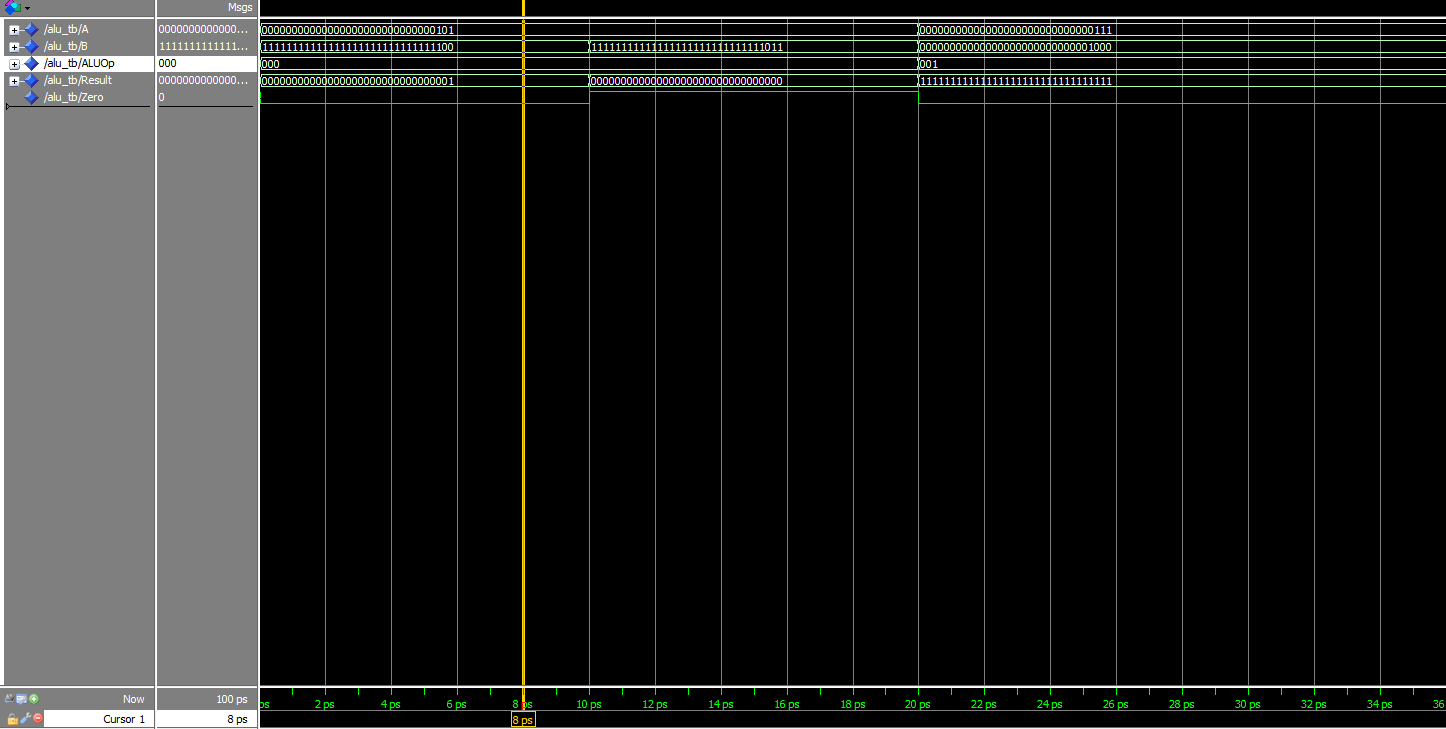
wait for 10 ps;

wait;

end process;

end Behavioral;

**Screenshot Testbench (ALU\_tb.vhd)**



**RegisterFile.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity RegisterFile is

Port (

clk : in STD\_LOGIC;

ReadReg1, ReadReg2, WriteReg : in STD\_LOGIC\_VECTOR(3 downto 0);

WriteData : in STD\_LOGIC\_VECTOR(31 downto 0);

RegWrite : in STD\_LOGIC;

ReadData1, ReadData2 : out STD\_LOGIC\_VECTOR(31 downto 0)

);

end RegisterFile;

architecture Behavioral of RegisterFile is

type regfile is array (15 downto 0) of STD\_LOGIC\_VECTOR(31 downto 0);

signal Registers : regfile := (others => (others => '0'));

begin

process(clk)

begin

if rising\_edge(clk) then

if RegWrite = '1' then

Registers(to\_integer(unsigned(WriteReg))) <= WriteData;

end if;

end if;

end process;

ReadData1 <= Registers(to\_integer(unsigned(ReadReg1)));

ReadData2 <= Registers(to\_integer(unsigned(ReadReg2)));

end Behavioral;

**RegisterFile\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity RegisterFile\_tb is

end RegisterFile\_tb;

architecture Behavioral of RegisterFile\_tb is

signal clk, RegWrite : STD\_LOGIC;

signal ReadReg1, ReadReg2, WriteReg : STD\_LOGIC\_VECTOR(3 downto 0);

signal WriteData : STD\_LOGIC\_VECTOR(31 downto 0);

signal ReadData1, ReadData2 : STD\_LOGIC\_VECTOR(31 downto 0);

component RegisterFile

Port ( clk : in STD\_LOGIC;

ReadReg1, ReadReg2, WriteReg : in STD\_LOGIC\_VECTOR(3 downto 0);

WriteData : in STD\_LOGIC\_VECTOR(31 downto 0);

RegWrite : in STD\_LOGIC;

ReadData1, ReadData2 : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

constant CLK\_PERIOD : time := 10 ps;

begin

UUT: RegisterFile Port map(clk => clk, ReadReg1 => ReadReg1, ReadReg2 => ReadReg2, WriteReg => WriteReg, WriteData => WriteData, RegWrite => RegWrite, ReadData1 => ReadData1, ReadData2 => ReadData2);

clk\_process : process

begin

clk <= '0';

wait for CLK\_PERIOD/2;

clk <= '1';

wait for CLK\_PERIOD/2;

end process;

process

begin

-- Write 5 to register $3

RegWrite <= '1';

WriteReg <= "0011";

WriteData <= std\_logic\_vector(to\_signed(5, 32));

wait for CLK\_PERIOD;

-- Write 7 to register $4

WriteReg <= "0100";

WriteData <= std\_logic\_vector(to\_signed(7, 32));

wait for CLK\_PERIOD;

-- Write 9 to register $5

WriteReg <= "0101";

WriteData <= std\_logic\_vector(to\_signed(9, 32));

wait for CLK\_PERIOD;

-- Read registers $3 and $4

RegWrite <= '0';

ReadReg1 <= "0011";

ReadReg2 <= "0100";

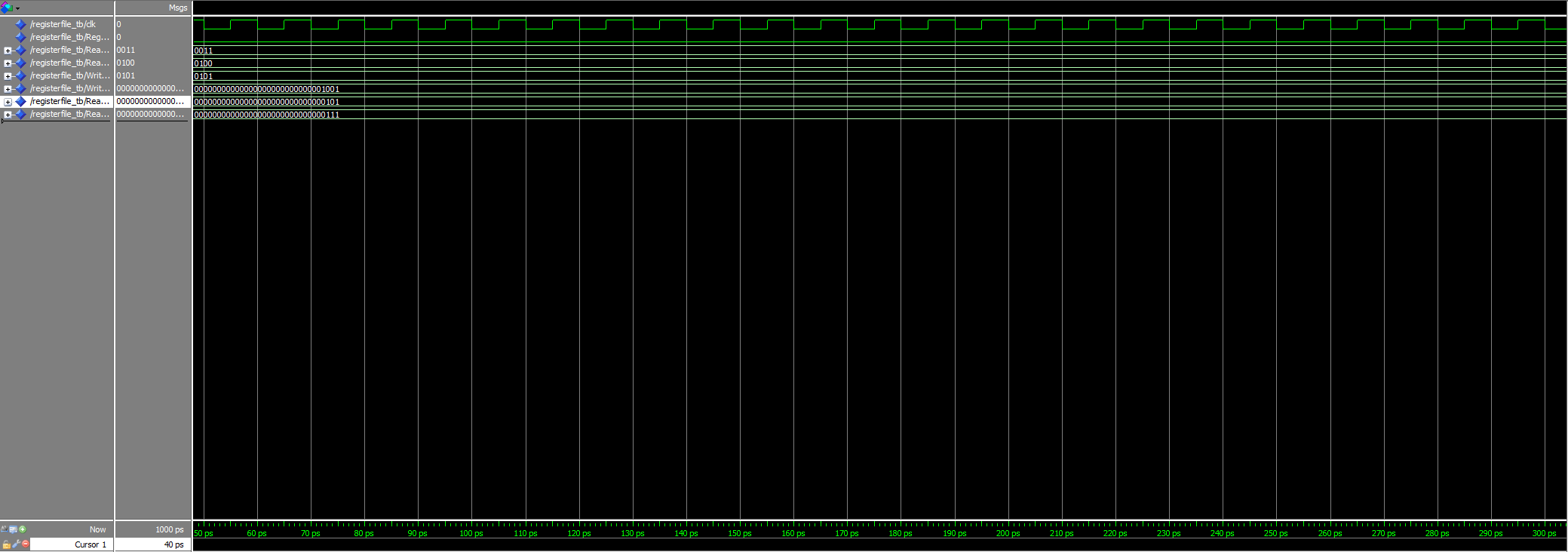
wait for CLK\_PERIOD;

wait;

end process;

end Behavioral;

**Screenshot Testbench(RegisterFile\_tb.vhd)**



**Datamem.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity DataMem is

Port ( Address : in STD\_LOGIC\_VECTOR(3 downto 0);

WriteData : in STD\_LOGIC\_VECTOR(31 downto 0);

MemWrite, MemRead : in STD\_LOGIC;

ReadData : out STD\_LOGIC\_VECTOR(31 downto 0));

end DataMem;

architecture Behavioral of DataMem is

type memtype is array (15 downto 0) of STD\_LOGIC\_VECTOR(31 downto 0);

signal Memory : memtype := (others => (others => '0'));

begin

process(Address, WriteData, MemWrite, MemRead)

begin

if MemWrite = '1' then

Memory(to\_integer(unsigned(Address))) <= WriteData;

end if;

if MemRead = '1' then

ReadData <= Memory(to\_integer(unsigned(Address)));

else

ReadData <= (others => 'Z'); -- High impedance state when not reading

end if;

end process;

end Behavioral;

**Datamem\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity DataMem\_tb is

end DataMem\_tb;

architecture Behavioral of DataMem\_tb is

signal Address : STD\_LOGIC\_VECTOR(3 downto 0);

signal WriteData : STD\_LOGIC\_VECTOR(31 downto 0);

signal MemWrite, MemRead : STD\_LOGIC;

signal ReadData : STD\_LOGIC\_VECTOR(31 downto 0);

component DataMem

Port ( Address : in STD\_LOGIC\_VECTOR(3 downto 0);

WriteData : in STD\_LOGIC\_VECTOR(31 downto 0);

MemWrite, MemRead : in STD\_LOGIC;

ReadData : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

begin

UUT: DataMem Port map(Address => Address, WriteData => WriteData, MemWrite => MemWrite, MemRead => MemRead, ReadData => ReadData);

process

begin

-- Write 5 to memory location 0

Address <= std\_logic\_vector(to\_unsigned(0, 4)); -- 4 bits wide

WriteData <= std\_logic\_vector(to\_signed(5, 32));

MemWrite <= '1';

MemRead <= '0';

wait for 10 ps;

-- Write 7 to memory location 1

Address <= std\_logic\_vector(to\_unsigned(1, 4)); -- 4 bits wide

WriteData <= std\_logic\_vector(to\_signed(7, 32));

wait for 10 ps;

-- Read memory location 0

Address <= std\_logic\_vector(to\_unsigned(0, 4)); -- 4 bits wide

MemWrite <= '0';

MemRead <= '1';

wait for 10 ps;

wait;

end process;

end Behavioral;

**Screenshot Testbench(Datamem\_tb.vhd)**



**Imem.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity IMem is

Port (

Address : in STD\_LOGIC\_VECTOR(3 downto 0);

ReadData : out STD\_LOGIC\_VECTOR(31 downto 0)

);

end IMem;

architecture Behavioral of IMem is

type memtype is array (15 downto 0) of STD\_LOGIC\_VECTOR(31 downto 0);

signal Memory : memtype := (

0 => x"20000000", -- addi $0, $0, 0

1 => x"20420000", -- addi $2, $2, 0

2 => x"20840000", -- addi $2, $4, 0

3 => x"20030001", -- addi $3, $0, 1

4 => x"20050003", -- addi $5, $0, 3

5 => x"00603020", -- add $6, $3, $0

6 => x"AC860000", -- sw $6, 0($4)

7 => x"20630001", -- addi $3, $3, 1

8 => x"20840001", -- addi $4, $4, 1

9 => x"20A5FFFF", -- addi $5, $5, -1

10 => x"14A0FFF9", -- bne $5, $0, L1 (Offset -7)

others => (others => '0')

);

begin

process(Address)

begin

ReadData <= Memory(to\_integer(unsigned(Address)));

end process;

end Behavioral;

**Imem\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity IMem\_tb is

end IMem\_tb;

architecture Behavioral of IMem\_tb is

signal Address : STD\_LOGIC\_VECTOR(3 downto 0);

signal ReadData : STD\_LOGIC\_VECTOR(31 downto 0);

component IMem

Port (

Address : in STD\_LOGIC\_VECTOR(3 downto 0);

ReadData : out STD\_LOGIC\_VECTOR(31 downto 0)

);

end component;

begin

UUT: IMem Port map(Address => Address, ReadData => ReadData);

process

begin

-- Read the memory at position 4

Address <= std\_logic\_vector(to\_unsigned(4, 4));

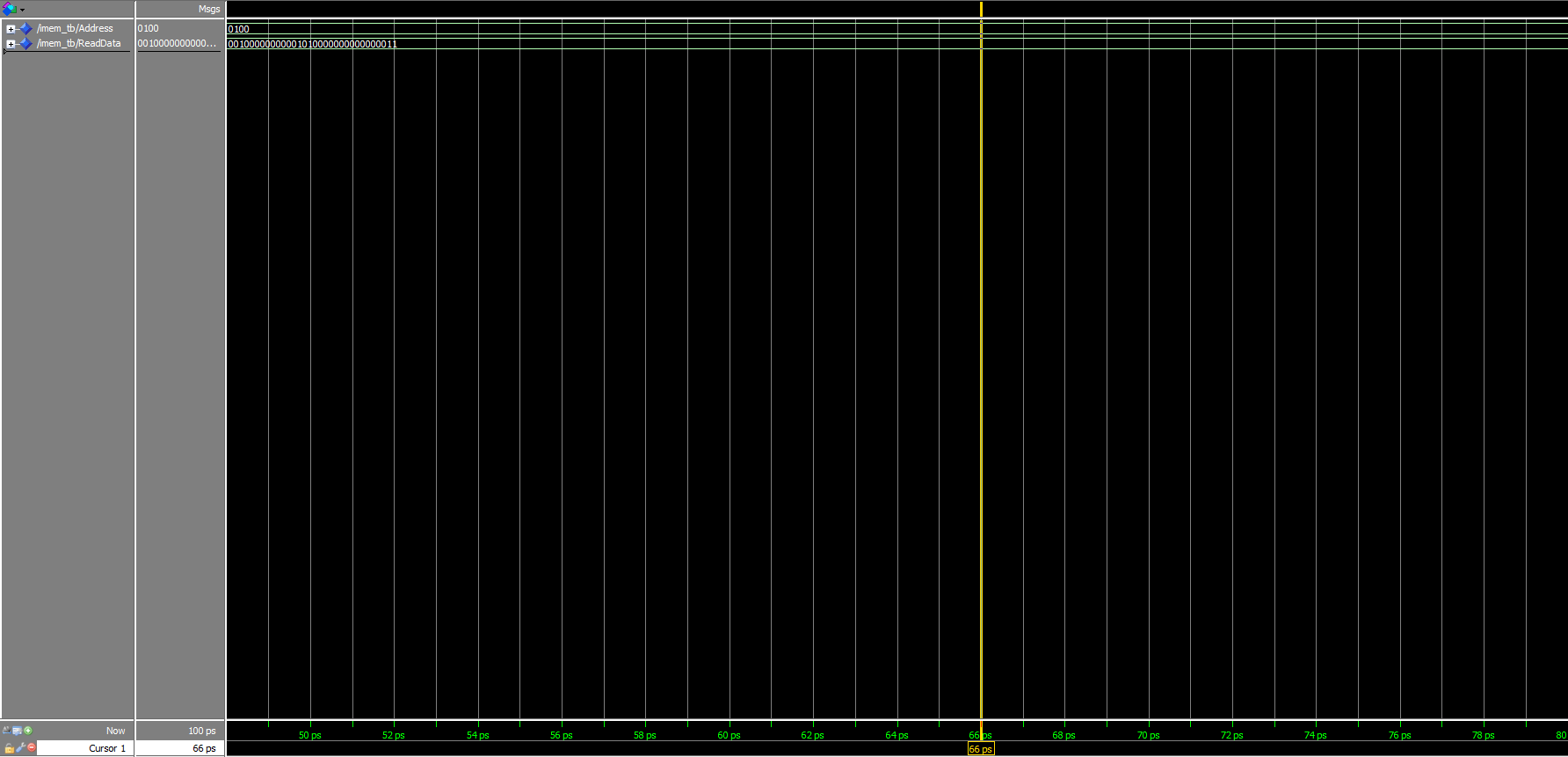
wait for 10 ns;

wait;

end process;

end Behavioral;

**Screenshot Testbench(Imem\_tb.vhd)**



**Control.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Control is

Port (

opcode : in std\_logic\_vector(5 downto 0);

RegDst : out std\_logic;

Branch : out std\_logic;

MemRead : out std\_logic;

MemtoReg : out std\_logic;

ALUOp : out std\_logic\_vector(1 downto 0);

MemWrite : out std\_logic;

ALUSrc : out std\_logic;

RegWrite : out std\_logic

);

end Control;

architecture Behavioral of Control\_Unit is

begin

process(opcode)

begin

-- Default values for control signals

RegDst <= '0';

Branch <= '0';

MemRead <= '0';

MemtoReg <= '0';

ALUOp <= "00";

MemWrite <= '0';

ALUSrc <= '0';

RegWrite <= '0';

case opcode is

when "000000" => -- R-type instructions (add, sub)

RegDst <= '1';

ALUOp <= "10"; -- ALU Control will further decode the funct field

RegWrite <= '1';

when "100011" => -- lw (load word)

ALUSrc <= '1';

MemtoReg <= '1';

RegWrite <= '1';

MemRead <= '1';

when "101011" => -- sw (store word)

ALUSrc <= '1';

MemWrite <= '1';

when "000100" => -- beq (branch if equal)

Branch <= '1';

ALUOp <= "01"; -- ALU performs subtraction

when "001000" => -- addi (add immediate)

ALUSrc <= '1';

RegWrite <= '1';

when "000101" => -- bne (branch if not equal)

Branch <= '1';

ALUOp <= "01"; -- ALU performs subtraction

when others =>

-- Default case for unrecognized opcodes

RegDst <= '0';

Branch <= '0';

MemRead <= '0';

MemtoReg <= '0';

ALUOp <= "00";

MemWrite <= '0';

ALUSrc <= '0';

RegWrite <= '0';

end case;

end process;

end Behavioral;

**Control\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Control\_tb is

end Control\_tb;

architecture Behavioral of Control\_tb is

-- Component declaration for the Control Unit

component Control

Port (

opcode : in std\_logic\_vector(5 downto 0);

RegDst : out std\_logic;

Branch : out std\_logic;

MemRead : out std\_logic;

MemtoReg : out std\_logic;

ALUOp : out std\_logic\_vector(1 downto 0);

MemWrite : out std\_logic;

ALUSrc : out std\_logic;

RegWrite : out std\_logic

);

end component;

-- Testbench signals

signal opcode : std\_logic\_vector(5 downto 0) := (others => '0');

signal RegDst : std\_logic;

signal Branch : std\_logic;

signal MemRead : std\_logic;

signal MemtoReg : std\_logic;

signal ALUOp : std\_logic\_vector(1 downto 0);

signal MemWrite : std\_logic;

signal ALUSrc : std\_logic;

signal RegWrite : std\_logic;

begin

uut: Control

Port map (

opcode => opcode,

RegDst => RegDst,

Branch => Branch,

MemRead => MemRead,

MemtoReg => MemtoReg,

ALUOp => ALUOp,

MemWrite => MemWrite,

ALUSrc => ALUSrc,

RegWrite => RegWrite

);

-- Test process

process

begin

-- Test case for addi $0, $0, 0

opcode <= "001000"; -- opcode for addi

wait for 10 ps;

assert (ALUSrc = '1' and RegWrite = '1' and RegDst = '0' and Branch = '0' and MemRead = '0' and MemtoReg = '0' and ALUOp = "00" and MemWrite = '0')

report "Test case for addi failed" severity error;

report "addi $0, $0, 0 test passed" severity note;

-- Test case for sw $6, 0($4)

opcode <= "101011"; -- opcode for sw

wait for 10 ps;

assert (ALUSrc = '1' and MemWrite = '1' and RegDst = '0' and Branch = '0' and MemRead = '0' and MemtoReg = '0' and ALUOp = "00" and RegWrite = '0')

report "Test case for sw failed" severity error;

report "sw $6, 0($4) test passed" severity note;

-- Test case for bne $5, $0, L1

opcode <= "000101"; -- opcode for bne

wait for 10 ps;

assert (Branch = '1' and ALUOp = "01" and ALUSrc = '0' and RegWrite = '0' and RegDst = '0' and MemRead = '0' and MemtoReg = '0' and MemWrite = '0')

report "Test case for bne failed" severity error;

report "bne $5, $0, L1 test passed" severity note;

wait;

end process;

end behavioral;

**Screenshot Testbench(Control\_tb.vhd)**

**ALUcontrol.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ALUControl is

Port (

funct : in STD\_LOGIC\_VECTOR(5 downto 0);

ALUOp : in STD\_LOGIC\_VECTOR(1 downto 0);

ALUControlOut : out STD\_LOGIC\_VECTOR(2 downto 0)

);

end ALUControl;

architecture Behavioral of ALUControl is

begin

process(funct, ALUOp)

begin

case ALUOp is

when "00" => ALUControlOut <= "010"; -- add (lw/sw)

when "01" => ALUControlOut <= "110"; -- sub (beq)

when "10" =>

case funct is

when "100000" => ALUControlOut <= "010"; -- add

when "100010" => ALUControlOut <= "110"; -- sub

when "100100" => ALUControlOut <= "000"; -- and

when "100101" => ALUControlOut <= "001"; -- or

when "101010" => ALUControlOut <= "111"; -- slt

when others => ALUControlOut <= "000"; -- or any default valid value

end case;

when others => ALUControlOut <= "000"; -- or any default valid value

end case;

end process;

end Behavioral;

**ALUcontrol\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ALUControl\_tb is

end ALUControl\_tb;

architecture Behavioral of ALUControl\_tb is

signal funct : STD\_LOGIC\_VECTOR(5 downto 0);

signal ALUOp : STD\_LOGIC\_VECTOR(1 downto 0);

signal ALUControlOut : STD\_LOGIC\_VECTOR(2 downto 0);

component ALUControl

Port (

funct : in STD\_LOGIC\_VECTOR(5 downto 0);

ALUOp : in STD\_LOGIC\_VECTOR(1 downto 0);

ALUControlOut : out STD\_LOGIC\_VECTOR(2 downto 0)

);

end component;

begin

-- Instantiate the ALUControl component

UUT: ALUControl

Port map(

funct => funct,

ALUOp => ALUOp,

ALUControlOut => ALUControlOut

);

-- Stimulus process

stimulus\_process : process

begin

-- Test case 1: Funct = 100000, ALUOp = 10

funct <= "100000";

ALUOp <= "10";

wait for 10 ps;

-- Test case 2: Funct = 100010, ALUOp = 10

funct <= "100010";

ALUOp <= "10";

wait for 10 ps;

-- Test case 3: Funct = 111111, ALUOp = 00

funct <= "111111";

ALUOp <= "00";

wait for 10 ps;

-- Test case 4: Funct = 111111, ALUOp = 01

funct <= "111111";

ALUOp <= "01";

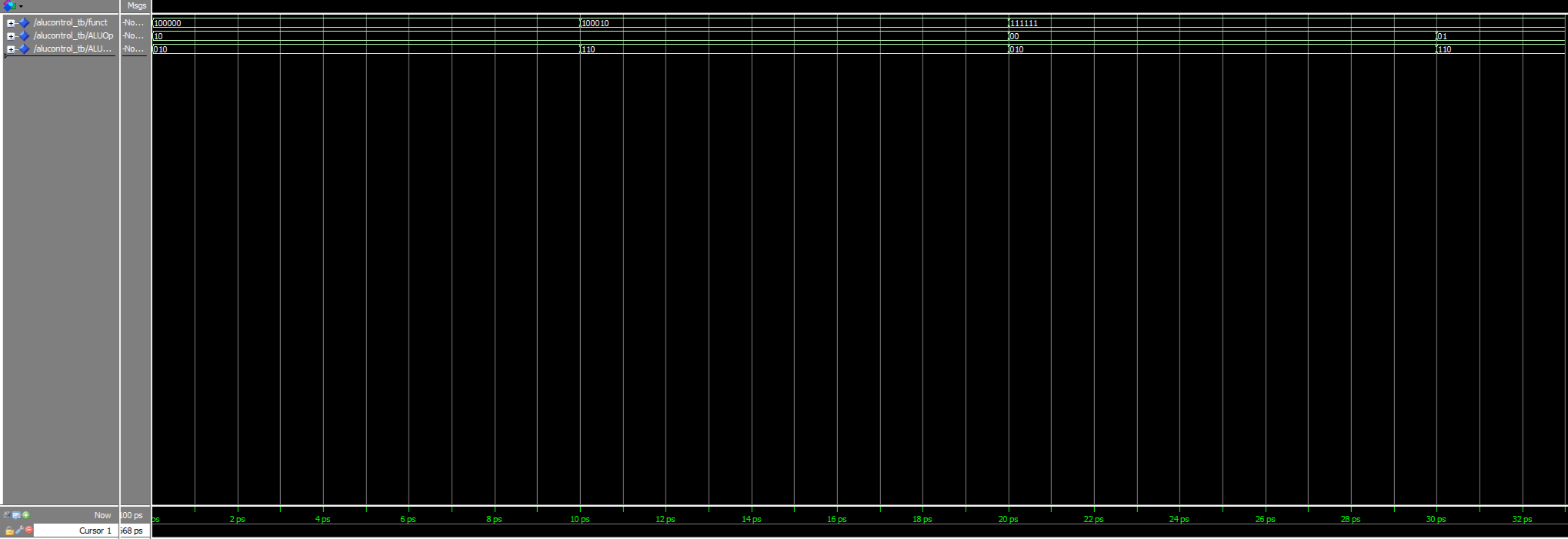
wait for 10 ps;

wait; -- End of simulation

end process stimulus\_process;

end Behavioral;

**Screenshot Testbench(ALUcontrol\_tb.vhd)**



**PC.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity PC is

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

PC\_in : in STD\_LOGIC\_VECTOR(31 downto 0);

PC\_out : out STD\_LOGIC\_VECTOR(31 downto 0));

end PC;

architecture Behavioral of PC is

signal PC\_reg : STD\_LOGIC\_VECTOR(31 downto 0) := (others => '0');

begin

process(clk, reset)

begin

if reset = '1' then

PC\_reg <= (others => '0');

elsif rising\_edge(clk) then

PC\_reg <= PC\_in;

end if;

end process;

PC\_out <= PC\_reg;

end Behavioral;

**PC\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity PC\_tb is

end PC\_tb;

architecture Behavioral of PC\_tb is

signal clk, reset : STD\_LOGIC;

signal PC\_in : STD\_LOGIC\_VECTOR(31 downto 0);

signal PC\_out : STD\_LOGIC\_VECTOR(31 downto 0);

component PC

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

PC\_in : in STD\_LOGIC\_VECTOR(31 downto 0);

PC\_out : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

constant CLK\_PERIOD : time := 10 ps;

begin

UUT: PC Port map(clk => clk, reset => reset, PC\_in => PC\_in, PC\_out => PC\_out);

clk\_process : process

begin

clk <= '0';

wait for CLK\_PERIOD/2;

clk <= '1';

wait for CLK\_PERIOD/2;

end process;

process

begin

reset <= '1';

wait for 10 ps;

reset <= '0';

-- Write 0xAAAA BBBB

PC\_in <= x"AAAA\_BBBB";

wait for 10 ps;

-- Write 0xFFFF CCCC

PC\_in <= x"FFFF\_CCCC";

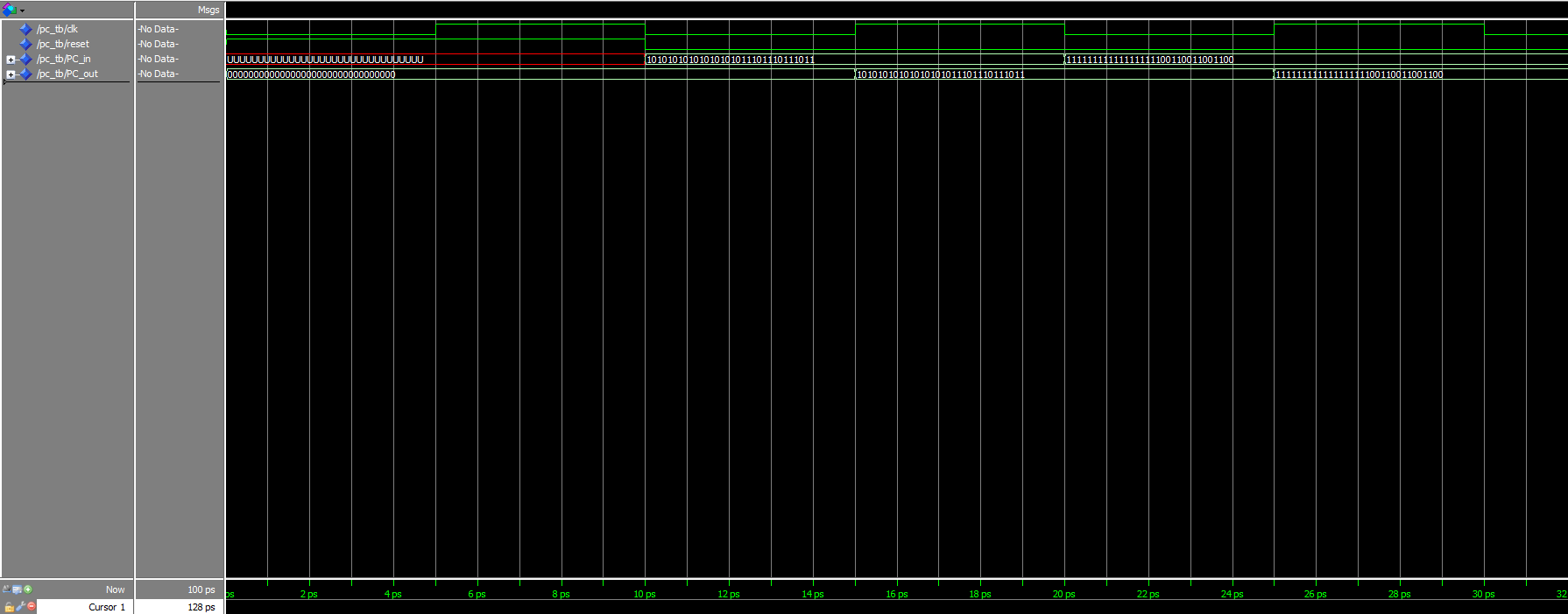
wait for 10 ps;

wait;

end process;

end Behavioral;

**Screenshot Testbench(PC\_tb.vhd)**



**5Mux2to1.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity A5Mux2to1 is

Port (

A : in STD\_LOGIC\_VECTOR (4 downto 0);

B : in STD\_LOGIC\_VECTOR (4 downto 0);

sel : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (4 downto 0)

);

end A5Mux2to1;

architecture Behavioral of A5Mux2to1 is

begin

process(A, B, sel)

begin

if sel = '0' then

Y <= A;

else

Y <= B;

end if;

end process;

end Behavioral;

**5Mux2to1\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity A5Mux2to1\_tb is

end A5Mux2to1\_tb;

architecture Behavioral of A5Mux2to1\_tb is

signal A, B : STD\_LOGIC\_VECTOR(4 downto 0);

signal sel : STD\_LOGIC;

signal Y : STD\_LOGIC\_VECTOR(4 downto 0);

component a5Mux2to1

Port (

A : in STD\_LOGIC\_VECTOR(4 downto 0);

B : in STD\_LOGIC\_VECTOR(4 downto 0);

sel : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR(4 downto 0)

);

end component;

begin

-- Instantiate the a5Mux2to1 component

UUT: a5Mux2to1

Port map(

A => A,

B => B,

sel => sel,

Y => Y

);

-- Stimulus process

stimulus\_process : process

begin

-- Test case 1: A = 0x1A, B = 0x0B, sel = 1

A <= "11010"; -- 0x1A

B <= "01011"; -- 0x0B

sel <= '1';

wait for 10 ps;

-- Test case 2: A = 0x1A, B = 0x0B, sel = 0

sel <= '0';

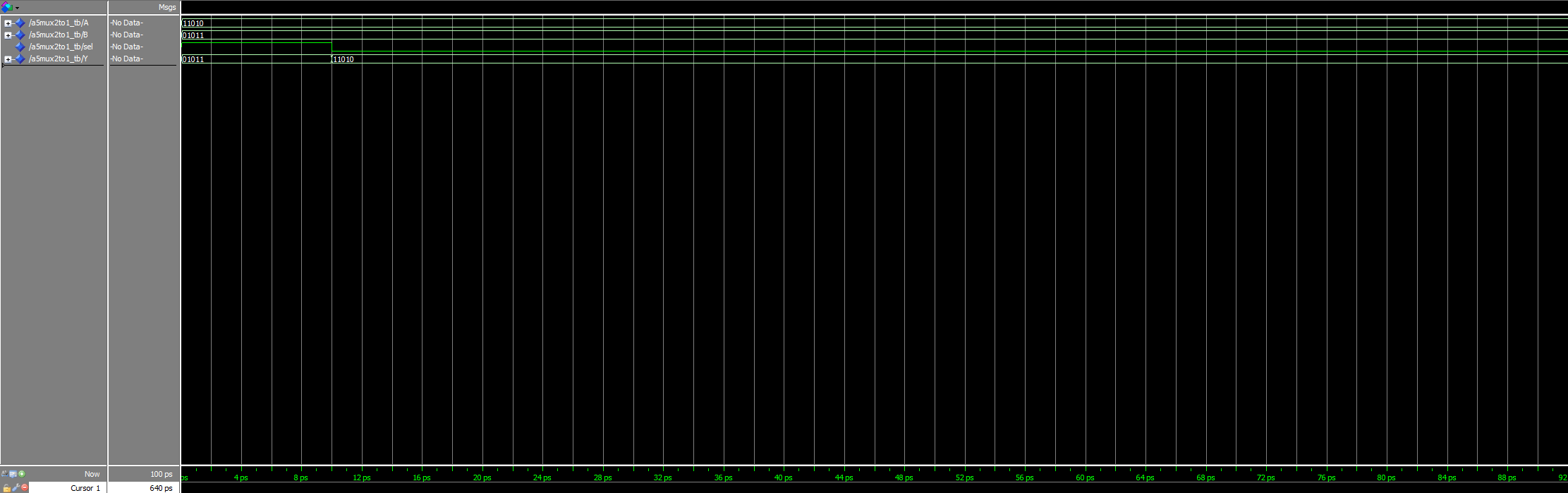
wait for 10 ps;

wait; -- End of simulation

end process stimulus\_process;

end Behavioral;

**Screenshot Testbench(5Mux2to1\_tb.vhd)**



**Signextension.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity SignExtension is

Port ( inst : in STD\_LOGIC\_VECTOR (15 downto 0);

data : out STD\_LOGIC\_VECTOR (31 downto 0));

end SignExtension;

architecture Behavioral of SignExtension is

begin

process(inst)

begin

data <= std\_logic\_vector(resize(signed(inst), 32));

end process;

end Behavioral;

**Signextension\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity SignExtension\_tb is

end SignExtension\_tb;

architecture Behavioral of SignExtension\_tb is

signal inst : STD\_LOGIC\_VECTOR(15 downto 0);

signal data : STD\_LOGIC\_VECTOR(31 downto 0);

component SignExtension

Port ( inst : in STD\_LOGIC\_VECTOR(15 downto 0);

data : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

begin

UUT: SignExtension Port map(inst => inst, data => data);

process

begin

-- 0xFFFF

inst <= "1111111111111111";

wait for 10 ps;

-- 0xAAAA

inst <= "1010101010101010";

wait for 10 ps;

-- 0x5555

inst <= "0101010101010101";

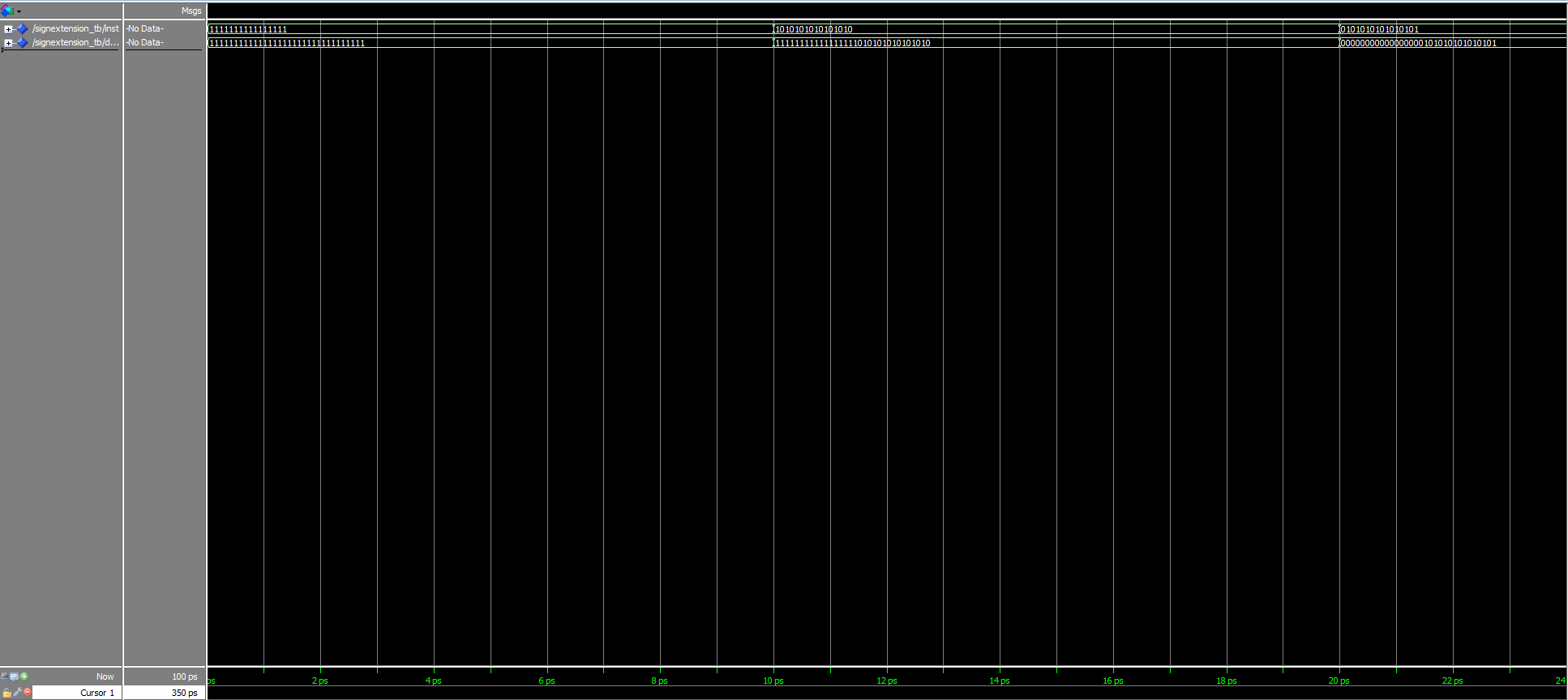
wait for 10 ps;

wait;

end process;

end Behavioral;

**Screenshot Testbench(Signextension\_tb.vhd )**



**32Mux2to1.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity A32Mux2to1 is

Port (

A : in STD\_LOGIC\_VECTOR (31 downto 0);

B : in STD\_LOGIC\_VECTOR (31 downto 0);

sel : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (31 downto 0)

);

end A32Mux2to1;

architecture Behavioral of A32Mux2to1 is

begin

process(A, B, sel)

begin

if sel = '0' then

Y <= A;

else

Y <= B;

end if;

end process;

end Behavioral;

**32Mux2to1\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity A32Mux2to1\_tb is

end A32Mux2to1\_tb;

architecture Behavioral of A32Mux2to1\_tb is

signal A, B : STD\_LOGIC\_VECTOR(31 downto 0);

signal sel : STD\_LOGIC;

signal Y : STD\_LOGIC\_VECTOR(31 downto 0);

component A32Mux2to1

Port (

A : in STD\_LOGIC\_VECTOR(31 downto 0);

B : in STD\_LOGIC\_VECTOR(31 downto 0);

sel : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR(31 downto 0)

);

end component;

begin

-- Instantiate the A32Mux2to1 component

UUT: A32Mux2to1

Port map(

A => A,

B => B,

sel => sel,

Y => Y

);

-- Stimulus process

stimulus\_process : process

begin

-- Test case 1: A = 0xAAAAAAAA, B = 0xBBBBBBBB, sel = 1

A <= x"AAAAAAAA";

B <= x"BBBBBBBB";

sel <= '1';

wait for 10 ps;

-- Test case 2: A = 0xAAAAAAAA, B = 0xBBBBBBBB, sel = 0

sel <= '0';

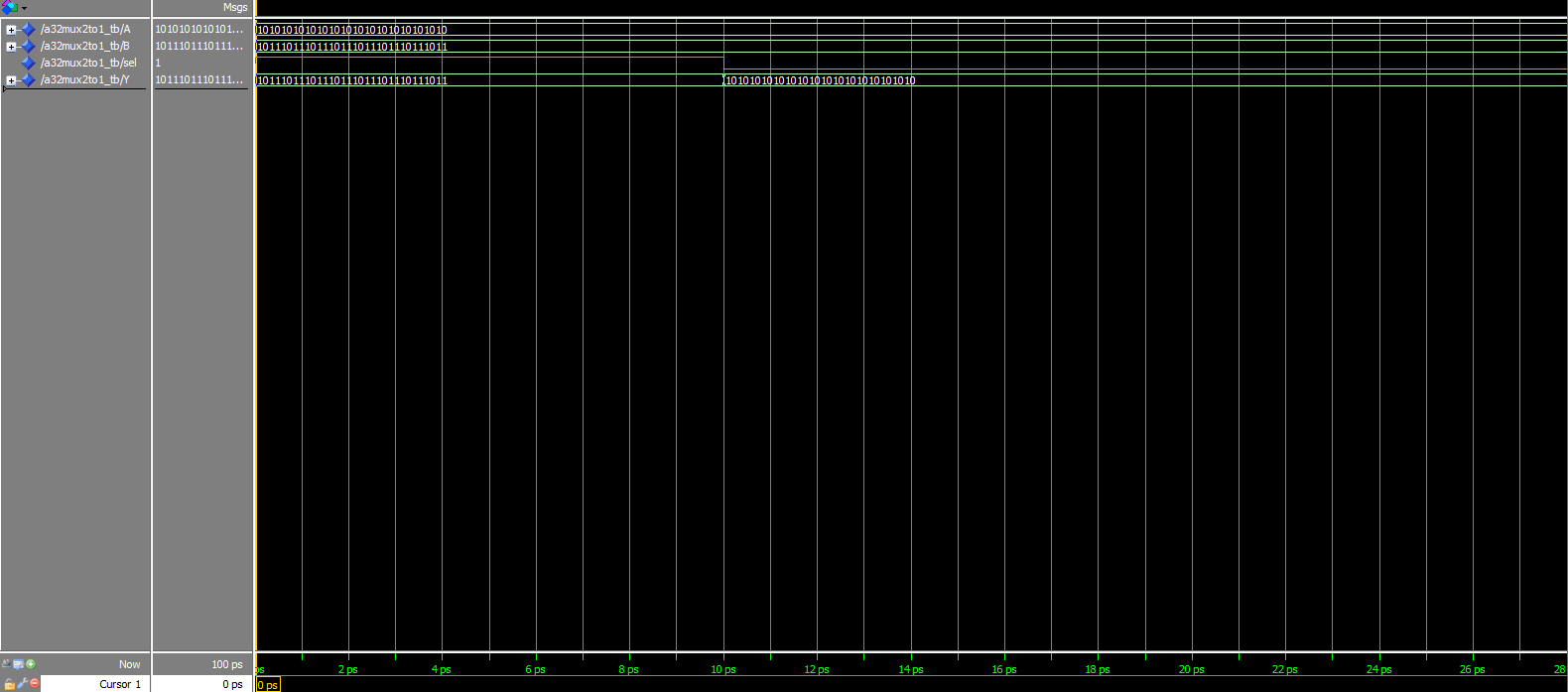
wait for 10 ps;

wait; -- End of simulation

end process stimulus\_process;

end Behavioral;

**Screenshot Testbench(32Mux2to1\_tb.vhd)**



**Leftshift.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity LeftShift is

Port ( data\_in : in STD\_LOGIC\_VECTOR (31 downto 0);

data\_out : out STD\_LOGIC\_VECTOR (31 downto 0));

end LeftShift;

architecture Behavioral of LeftShift is

begin

process(data\_in)

begin

data\_out <= std\_logic\_vector(shift\_left(unsigned(data\_in), 2));

end process;

end Behavioral;

**LeftShift\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity LeftShift\_tb is

end LeftShift\_tb;

architecture Behavioral of LeftShift\_tb is

signal data\_in : STD\_LOGIC\_VECTOR(31 downto 0);

signal data\_out : STD\_LOGIC\_VECTOR(31 downto 0);

component LeftShift

Port ( data\_in : in STD\_LOGIC\_VECTOR(31 downto 0);

data\_out : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

begin

UUT: LeftShift Port map(data\_in => data\_in, data\_out => data\_out);

process

begin

-- 0x0FFFFFFF

data\_in <= x"0FFFFFFF";

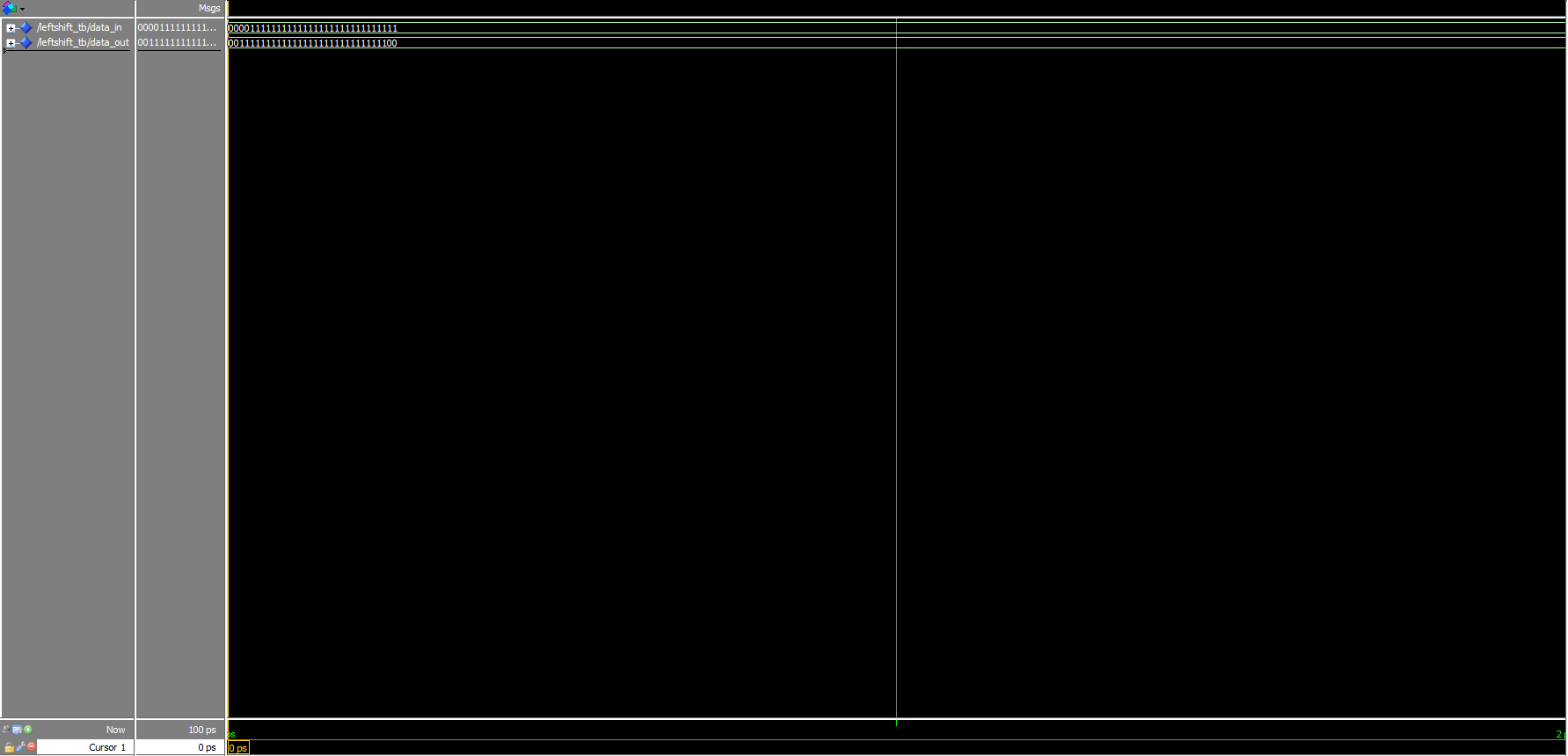
wait for 10 ps;

wait;

end process;

end Behavioral;

**Screenshot Testbench(LeftShift\_tb.vhd)**



**Adder32.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity Adder32 is

Port ( A : in STD\_LOGIC\_VECTOR (31 downto 0);

B : in STD\_LOGIC\_VECTOR (31 downto 0);

Sum : out STD\_LOGIC\_VECTOR (31 downto 0));

end Adder32;

architecture Behavioral of Adder32 is

begin

process(A, B)

begin

Sum <= std\_logic\_vector(unsigned(A) + unsigned(B)); -- Add the two 32-bit inputs

end process;

end Behavioral;

**Adder32\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Adder32\_tb is

end Adder32\_tb;

architecture Behavioral of Adder32\_tb is

signal A, B : STD\_LOGIC\_VECTOR(31 downto 0);

signal Sum : STD\_LOGIC\_VECTOR(31 downto 0);

component Adder32

Port (

A : in STD\_LOGIC\_VECTOR(31 downto 0);

B : in STD\_LOGIC\_VECTOR(31 downto 0);

Sum : out STD\_LOGIC\_VECTOR(31 downto 0)

);

end component;

begin

-- Instantiate the Adder32 component

UUT: Adder32

Port map(

A => A,

B => B,

Sum => Sum

);

-- Stimulus process

stimulus\_process : process

begin

-- Test case 1: A = 0xAAAAAAAA, B = 0xBBBBBBBB

A <= x"AAAAAAAA";

B <= x"BBBBBBBB";

wait for 10 ps;

-- Test case 2: A = 0xAAAAAAAA, B = 0x55555556

A <= x"AAAAAAAA";

B <= x"55555556";

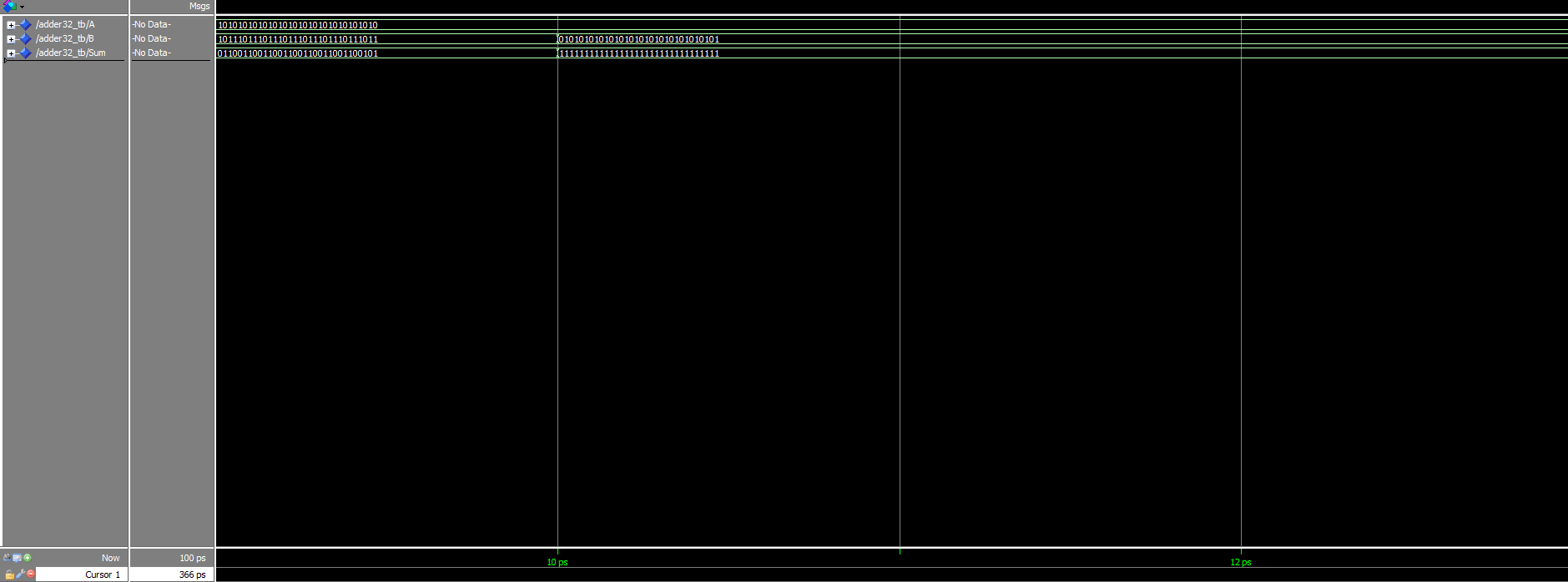
wait for 10 ps;

wait; -- End of simulation

end process stimulus\_process;

end Behavioral;

**Screenshot Testbench( Adder32\_tb.vhd)**



**MIPS.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity MIPS is

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC);

end MIPS;

architecture Behavioral of MIPS is

-- Declare internal signals

signal PC\_reg, nextPC, PC\_plus1, branchAddr, jumpAddr : STD\_LOGIC\_VECTOR(31 downto 0);

signal instruction : STD\_LOGIC\_VECTOR(31 downto 0);

signal readData1, readData2, writeData, ALUResult, signExtended, shiftedSignExtended : STD\_LOGIC\_VECTOR(31 downto 0);

signal zero : STD\_LOGIC;

signal RegDst, ALUSrc, MemToReg, RegWrite, MemRead, MemWrite, Branch, Jump : STD\_LOGIC;

signal ALUOp : STD\_LOGIC\_VECTOR(1 downto 0);

signal ALUControlOut : STD\_LOGIC\_VECTOR(2 downto 0);

signal muxOut : STD\_LOGIC\_VECTOR(4 downto 0); -- 5-bit wide for RegisterFile

signal muxOut\_32bit : STD\_LOGIC\_VECTOR(31 downto 0);

signal PCSrc : STD\_LOGIC;

-- Instantiate components

component PC

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

PC\_in : in STD\_LOGIC\_VECTOR(31 downto 0);

PC\_out : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

component IMem

Port ( Address : in STD\_LOGIC\_VECTOR(3 downto 0);

ReadData : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

component Control\_Unit

Port ( opcode : in std\_logic\_vector(5 downto 0);

RegDst : out std\_logic;

Branch : out std\_logic;

MemRead : out std\_logic;

MemtoReg : out std\_logic;

ALUOp : out std\_logic\_vector(1 downto 0);

MemWrite : out std\_logic;

ALUSrc : out std\_logic;

RegWrite : out std\_logic);

end component;

component RegisterFile

Port ( clk : in STD\_LOGIC;

ReadReg1, ReadReg2, WriteReg : in STD\_LOGIC\_VECTOR(3 downto 0);

WriteData : in STD\_LOGIC\_VECTOR(31 downto 0);

RegWrite : in STD\_LOGIC;

ReadData1, ReadData2 : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

component SignExtension

Port ( inst : in STD\_LOGIC\_VECTOR(15 downto 0);

data : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

component ALUControl

Port ( funct : in STD\_LOGIC\_VECTOR(5 downto 0);

ALUOp : in STD\_LOGIC\_VECTOR(1 downto 0);

ALUControlOut : out STD\_LOGIC\_VECTOR(2 downto 0));

end component;

component ALU

Port ( A : in STD\_LOGIC\_VECTOR(31 downto 0);

B : in STD\_LOGIC\_VECTOR(31 downto 0);

ALUOp : in STD\_LOGIC\_VECTOR(2 downto 0);

Result : out STD\_LOGIC\_VECTOR(31 downto 0);

Zero : out STD\_LOGIC);

end component;

component DataMem

Port ( Address : in STD\_LOGIC\_VECTOR(3 downto 0);

WriteData : in STD\_LOGIC\_VECTOR(31 downto 0);

MemWrite, MemRead : in STD\_LOGIC;

ReadData : out STD\_LOGIC\_VECTOR(31 downto 0));

end component;

component B5Mux2to1

Port ( A : in STD\_LOGIC\_VECTOR (4 downto 0);

B : in STD\_LOGIC\_VECTOR (4 downto 0);

sel : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (4 downto 0));

end component;

component B32Mux2to1

Port ( A : in STD\_LOGIC\_VECTOR (31 downto 0);

B : in STD\_LOGIC\_VECTOR (31 downto 0);

sel : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (31 downto 0));

end component;

component LeftShift

Port ( data\_in : in STD\_LOGIC\_VECTOR (31 downto 0);

data\_out : out STD\_LOGIC\_VECTOR (31 downto 0));

end component;

component Adder32

Port ( A : in STD\_LOGIC\_VECTOR (31 downto 0);

B : in STD\_LOGIC\_VECTOR (31 downto 0);

Sum : out STD\_LOGIC\_VECTOR (31 downto 0));

end component;

begin

-- Instantiate Program Counter (PC)

u1: PC port map(clk => clk, reset => reset, PC\_in => nextPC, PC\_out => PC\_reg);

-- Instantiate Instruction Memory

u2: IMem port map(Address => PC\_reg(3 downto 0), ReadData => instruction);

-- Instantiate Control Unit

u3: Control\_Unit port map(opcode => instruction(31 downto 26), RegDst => RegDst, ALUSrc => ALUSrc, MemtoReg => MemtoReg, RegWrite => RegWrite, MemRead => MemRead, MemWrite => MemWrite, Branch => Branch, ALUOp => ALUOp);

-- Instantiate Register File

u4: RegisterFile port map(

clk => clk,

ReadReg1 => instruction(25 downto 22), -- Use lower 4 bits for register addresses

ReadReg2 => instruction(21 downto 18), -- Use lower 4 bits for register addresses

WriteReg => muxOut(3 downto 0), -- Ensure correct bit-width

WriteData => writeData,

RegWrite => RegWrite,

ReadData1 => readData1,

ReadData2 => readData2

);

-- Instantiate Sign Extension

u5: SignExtension port map(inst => instruction(15 downto 0), data => signExtended);

-- Instantiate ALU Control Unit

u6: ALUControl port map(funct => instruction(5 downto 0), ALUOp => ALUOp, ALUControlOut => ALUControlOut);

-- Instantiate ALU

u7: ALU port map(A => readData1, B => muxOut\_32bit, ALUOp => ALUControlOut, Result => ALUResult, Zero => zero);

-- Instantiate Data Memory

u8: DataMem port map(Address => ALUResult(3 downto 0), WriteData => readData2, MemWrite => MemWrite, MemRead => MemRead, ReadData => writeData);

-- Instantiate 5-bit 2-to-1 Multiplexer

u9: B5Mux2to1 port map(A => instruction(20 downto 16), B => instruction(15 downto 11), sel => RegDst, Y => muxOut);

-- Instantiate 32-bit 2-to-1 Multiplexer (ALU Source)

u10: B32Mux2to1 port map(A => readData2, B => signExtended, sel => ALUSrc, Y => muxOut\_32bit);

-- Instantiate Left Shift 2 Unit

u11: LeftShift port map(data\_in => signExtended, data\_out => shiftedSignExtended);

-- Instantiate Adder32 for PC+1

u12: Adder32 port map(A => PC\_reg, B => x"00000001", Sum => PC\_plus1);

-- Instantiate Adder32 for Branch Address

u13: Adder32 port map(A => PC\_plus1, B => shiftedSignExtended, Sum => branchAddr);

-- Compute next PC address based on branch and jump logic

PCSrc <= Branch and zero;

-- Select next PC value

process (PC\_plus1, branchAddr, PCSrc)

begin

if PCSrc = '1' then

nextPC <= branchAddr;

else

nextPC <= PC\_plus1;

end if;

end process;

end Behavioral;

**Εξήγηση του MIPS.vhd**

Το συγκεκριμένο πρόγραμμα περιγράφει τη λειτουργία ενός βασικού επεξεργαστή MIPS χρησιμοποιώντας την γλώσσα VHDL. Περιλαμβάνει την οντότητα "MIPS" με δύο εισόδους, το ρολόι (clk) και την επαναφορά (reset). Η αρχιτεκτονική "Behavioral" δηλώνει διάφορα εσωτερικά σήματα για την αποθήκευση διευθύνσεων, δεδομένων και άλλων απαραίτητων στοιχείων για τη λειτουργία του επεξεργαστή. Το πρόγραμμα περιλαμβάνει επίσης τα υποσυστήματα του επεξεργαστή, όπως ο μετρητής προγράμματος (PC), η μνήμη εντολών (IMem), η μονάδα ελέγχου (Control Unit), το αρχείο καταχωρητών (Register File), η μονάδα αριθμητικής λογικής (ALU), η μονάδα επέκτασης συμβόλου (Sign Extension), η μονάδα ελέγχου της ALU (ALU Control), η μνήμη δεδομένων (DataMem) και διάφοροι πολυπλέκτες (multiplexers). Το πρόγραμμα συνδυάζει αυτά τα υποσυστήματα για να εκτελεί βασικές εντολές του σετ εντολών MIPS, συμπεριλαμβανομένων της ανάγνωσης και εγγραφής δεδομένων από/προς τους καταχωρητές και τη μνήμη, την εκτέλεση αριθμητικών και λογικών πράξεων, και την ενημέρωση της διεύθυνσης του μετρητή προγράμματος για διακλαδώσεις και άλματα. Οι πολυπλέκτες χρησιμοποιούνται για την επιλογή των κατάλληλων τιμών στις εισόδους των διαφόρων υποσυστημάτων, ενώ οι αθροιστές χρησιμοποιούνται για τον υπολογισμό της επόμενης διεύθυνσης εντολής και της διεύθυνσης διακλάδωσης.

**MIPS\_tb.vhd**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity MIPS\_tb is

end MIPS\_tb;

architecture Behavioral of MIPS\_tb is

signal clk, reset : STD\_LOGIC;

component MIPS

Port (

clk : in STD\_LOGIC;

reset : in STD\_LOGIC

);

end component;

begin

-- Instantiate the MIPS processor

uut: MIPS port map(

clk => clk,

reset => reset

);

-- Clock process

clk\_process: process

begin

clk <= '0';

wait for 2.5 ns;

clk <= '1';

wait for 2.5 ns;

end process;

-- Test process

process

begin

reset <= '1'; -- Assert reset

wait for 4.9 ns; -- Hold reset for 5 ns

reset <= '0'; -- Deassert reset

wait for 100 ns; -- Run simulation for 1000 ns

wait;

end process;

end Behavioral;

**Εξήγηση του MIPS\_tb.vhd**

Αυτό το πρόγραμμα περιγράφει ένα testbench για τη δοκιμή του επεξεργαστή MIPS. Το testbench, με την οντότητα "MIPS\_tb" και την αρχιτεκτονική "Behavioral", δημιουργεί σήματα ρολογιού (clk) και επαναφοράς (reset) για τη δοκιμή του επεξεργαστή. Στο εσωτερικό του testbench, γίνεται δήλωση και ενσωμάτωση του επεξεργαστή MIPS ως ένα υποσύστημα με την ονομασία "uut" (unit under test), συνδέοντας τα σήματα clk και reset στις αντίστοιχες εισόδους του MIPS. Το testbench περιλαμβάνει δύο διεργασίες: η πρώτη δημιουργεί το σήμα ρολογιού με περίοδο 5 ns, εναλλάσσοντας μεταξύ '0' και '1' κάθε 2.5 ns. Η δεύτερη διεργασία διαχειρίζεται τη δοκιμή του επεξεργαστή MIPS, ξεκινώντας με την ενεργοποίηση της επαναφοράς (reset) για 5 ns και στη συνέχεια την απενεργοποίηση της, προκειμένου να αρχίσει η κανονική λειτουργία του επεξεργαστή. Η προσομοίωση διαρκεί για 100 ns, επιτρέποντας την παρατήρηση της λειτουργίας του επεξεργαστή κατά την εκτέλεση εντολών. Αυτή η διάταξη βοηθά στον έλεγχο της σωστής λειτουργίας του MIPS επεξεργαστή σε συνθήκες πραγματικού χρόνου, επιβεβαιώνοντας την ορθή λειτουργία του πριν από την υλοποίησή του σε υλικό. Τα αποτελέσματα προβάλλονται στο 16αδικό σύστημα για να διαβάζεται πιο εύκολα.

**Screenshot Testbench(MIPS\_tb.vhd)**

