ILLINOIS INSTITUTE OF TECHNOLOGY

Salvador Jimenez

ECE 485 Project 2

Post Project Report: 32-bit RISC MIPS Processor

Due Date: November 20th, 2023

Mustafa Tahir, A20458481

Salvador Jimez Gomez, A20495915

Table of Contents:

Abstract

In this project, students will commence a project aimed at enabling them to become well-versed in FPGA design and to acquire practical experience in working with VHDL, one of the most commonly used hardware description languages. As the project progresses beyond the multiplexer implementation, students will find themselves delving deeper into the intricacies of processor architecture, instruction set design, and the myriad components that constitute a fully functional 32-bit RISC MIPS processor. This hands-on experience with industry-standard tools and the latest version of VHDL positions students for a comprehensive understanding of digital design and processor development.

Introduction

Our project kicks off with crafting a 32-bit processor capable of handling R-type, I-type, and J-type instructions. Our initial task involves building a datapath for a 5-stage MIPS pipeline—IF, ID, EX, MEM, and WB—each stage operating within a clock cycle.

The focus shifts to integrating the datapath and control logic for specific instructions, ensuring seamless execution. Our Arithmetic Logic Unit (ALU) follows a similar strategy. The aim is to establish a solid foundation, laying the groundwork for a versatile processor that accommodates diverse instruction types within the MIPS pipeline framework.

Implementation

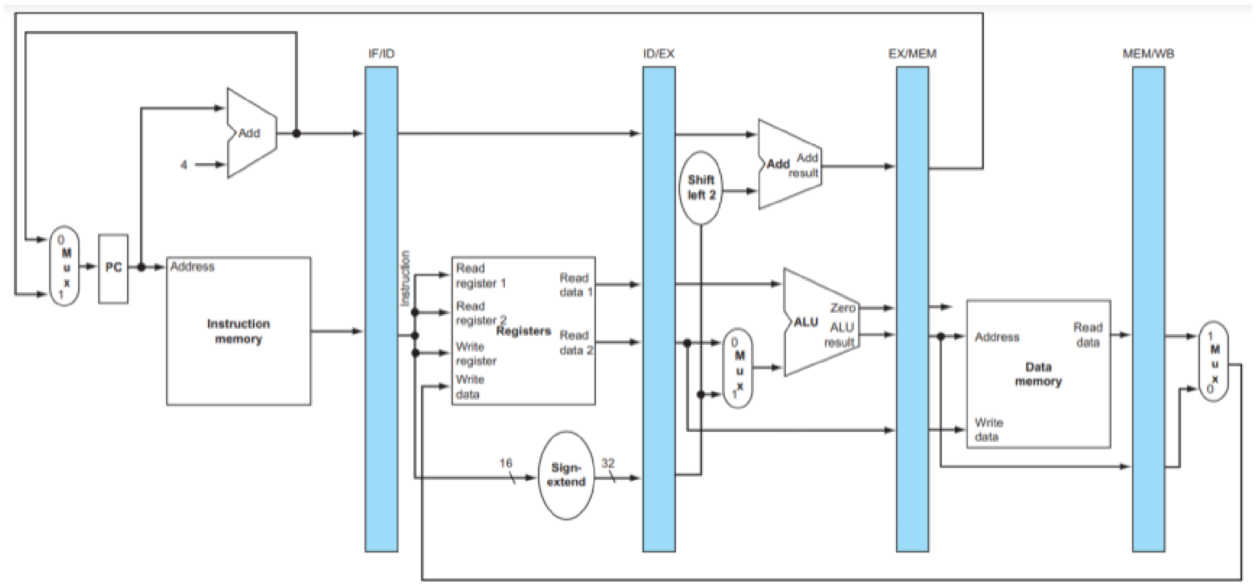


Figure 1.0 - High Level overview of the datapath implementation

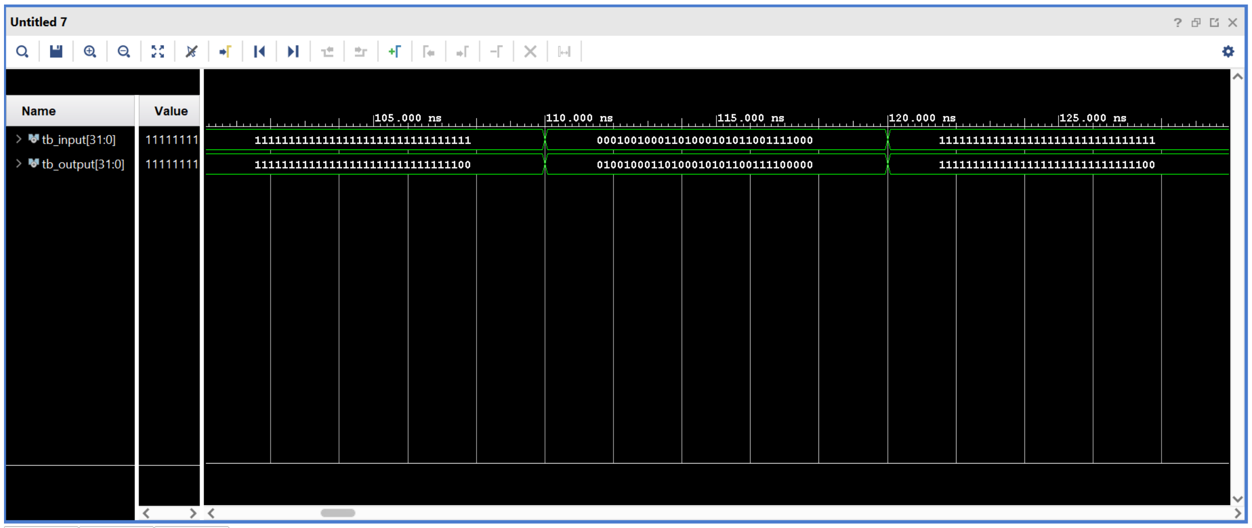
The pipeline architecture, comprising the five essential stages (IF, ID, EX, MEM, WB), underwent a systematic breakdown into structural models. These models were intricately interlinked under an encompassing CPU structural model, facilitating a comprehensive representation of the entire processor. Initially, the behavioral design phase involved the creation of individual components, each corresponding to a specific pipeline stage. For instance, components in the IF stage (PCMUX, PC, Instruction Memory, Adder) were independently implemented using behavioral models. Subsequently, these components were interconnected within a structural model encapsulated in a designated design file named IF_Stage.

Following the creation of stage-specific files (IF_Stage, ID_Stage, EX_Stage, MEM_Stage, WB_Stage), an interconnection scheme was established under a consolidated structural model named CPU. The interconnections within this file were structured as follows:

```
IF_Stage -> IF_ID_Reg -> ID_Stage -> ID_EX_Reg -> EX_Stage -> EX_MEM_Reg -> MEM_Stage
-> MEM_WB_Reg -> WB_Stage
```

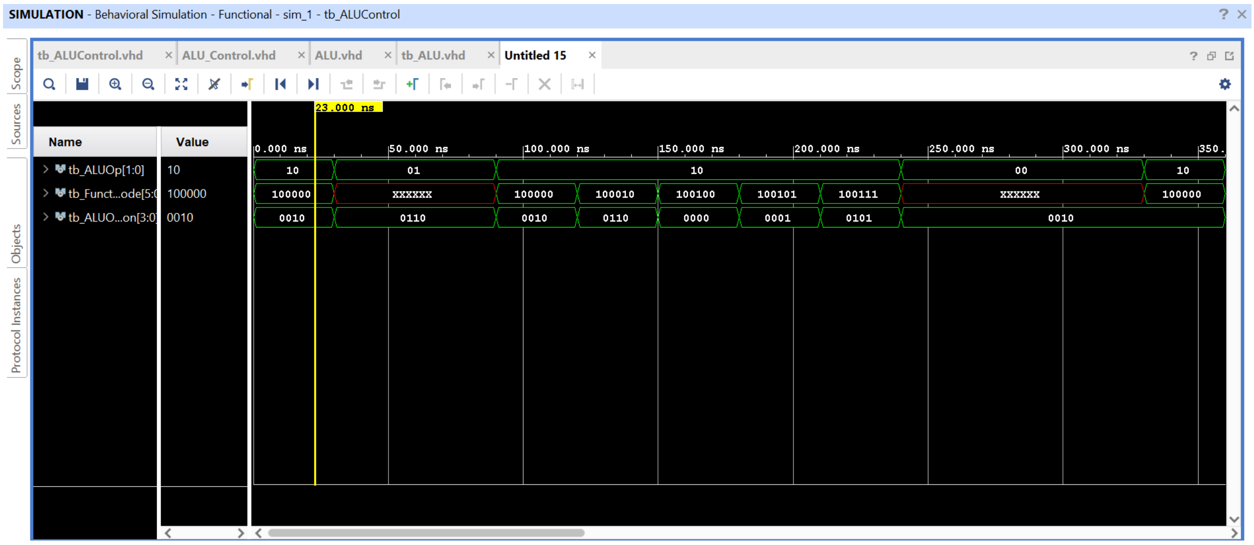
This hierarchical breakdown of our design serves a crucial purpose, enabling a systematic approach to implementation, testing, and debugging for each stage of the pipeline. Consequently, each individual component possesses its dedicated test bench, and likewise, each stage is equipped with its own test bench, facilitating a modular and efficient development and validation process.

TestBench Results



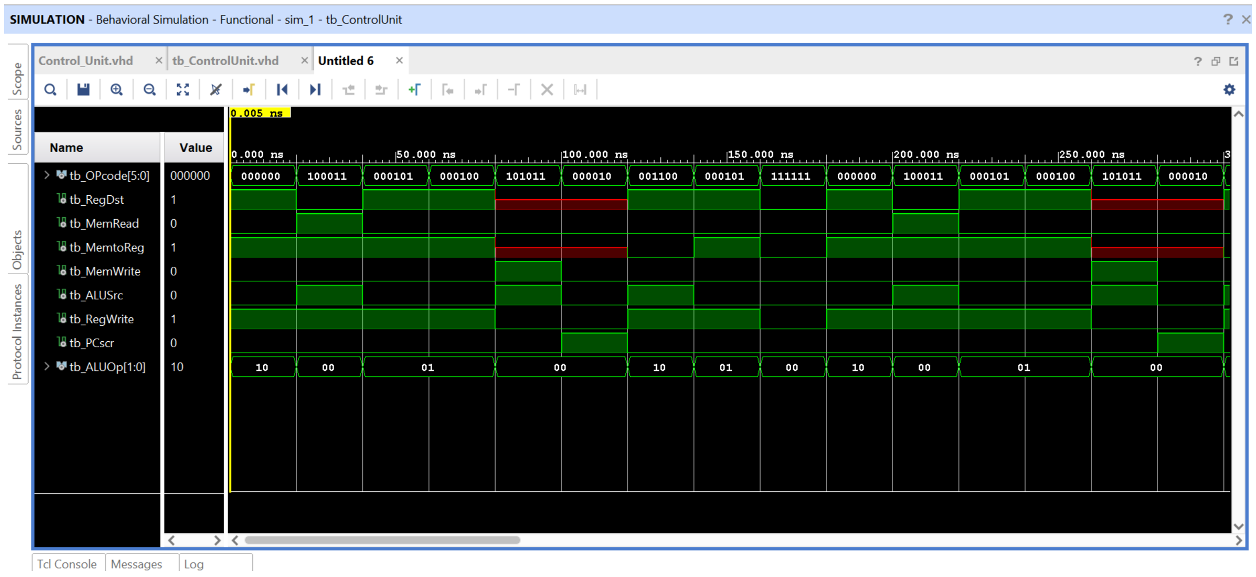
```
tb_input[31:0]   11111111   11111111111111111111111111111111   00010010001101000101011001111000   11111111111111111111111111111111
tb_output[31:0]  11111111   11111111111111111111111111111100   01001000110100010101100111100000   11111111111111111111111111111100
```

Shift Left by 2 Simulation



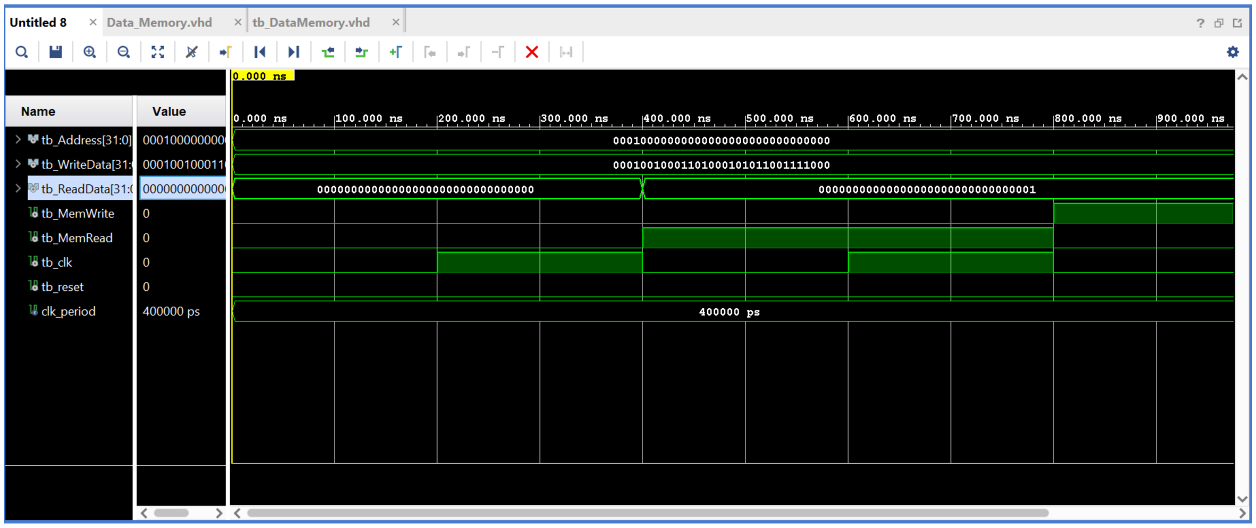
```
tb_ALUOp[1:0]        10      10       01       10                                          00       10
tb_Funct...ode[5:0   100000  100000   XXXXXX   100000  100010  100100  100101  100111      XXXXXX   100000
tb_ALUO...on[3:0]    0010    0010     0110     0010    0110    0000    0001    0101        0010
```

ALU Control Simulation



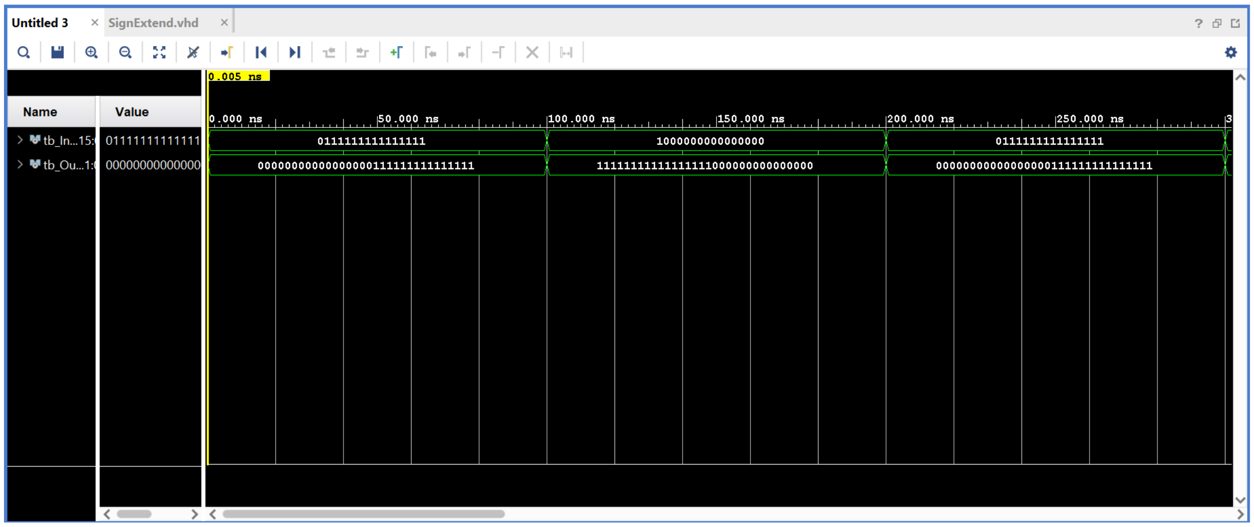
Name	Value
tb_OPcode[5:0]	000000
tb_RegDst	1
tb_MemRead	0
tb_MemtoReg	1
tb_MemWrite	0
tb_ALUSrc	0
tb_RegWrite	1
tb_PCscr	0
tb_ALUOp[1:0]	10

Control Unit Simulation

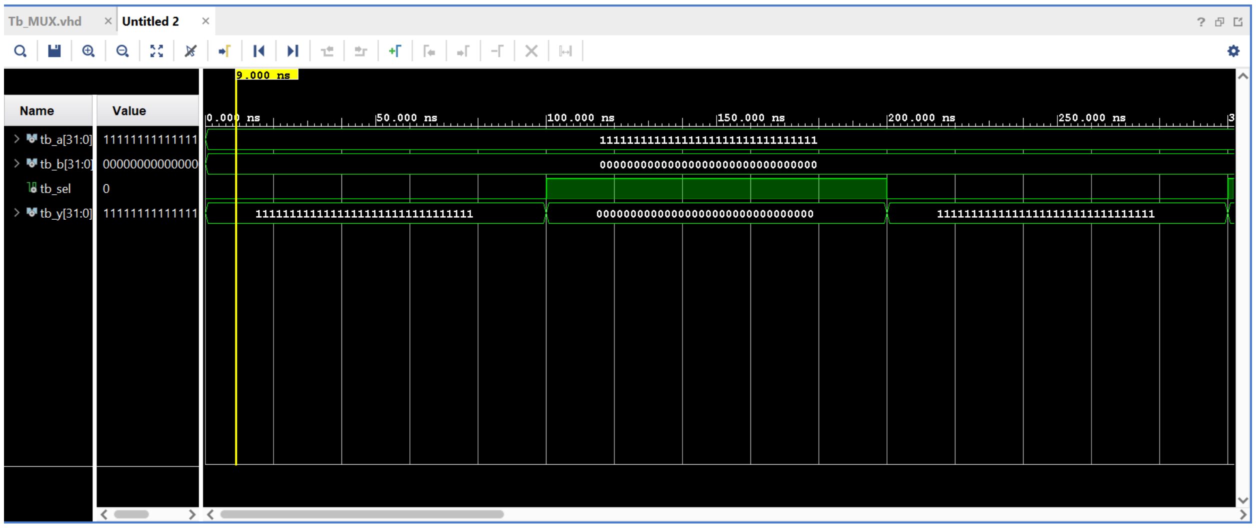


Name	Value
tb_Address[31:0]	0001000000000
tb_WriteData[31:	0001001000110
tb_ReadData[31:(	0000000000000
tb_MemWrite	0
tb_MemRead	0
tb_clk	0
tb_reset	0
clk_period	400000 ps

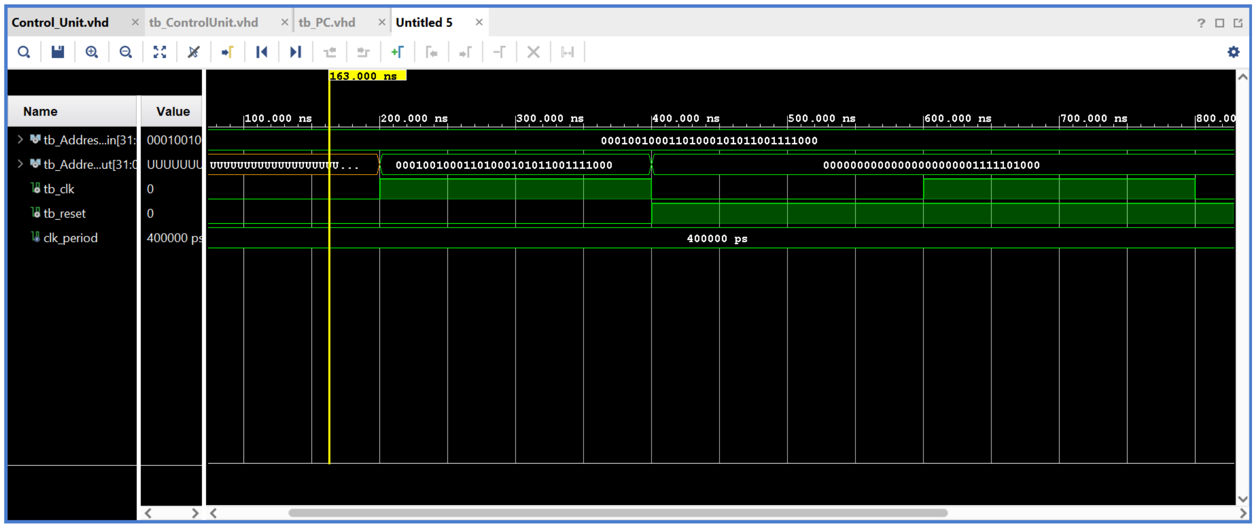
Data Memory



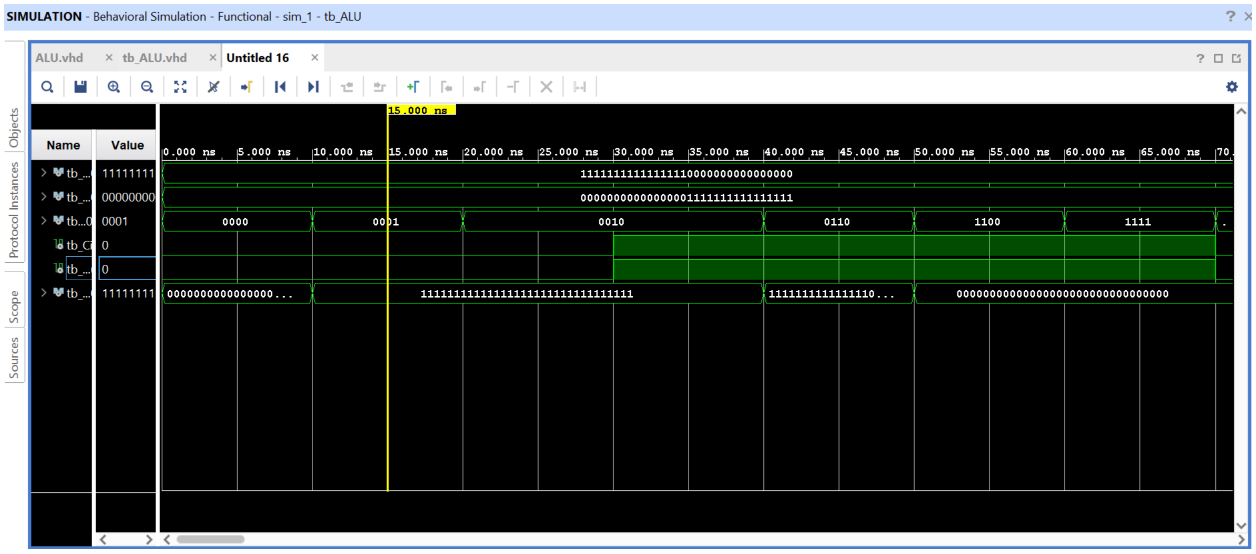
SignExtend Simulation



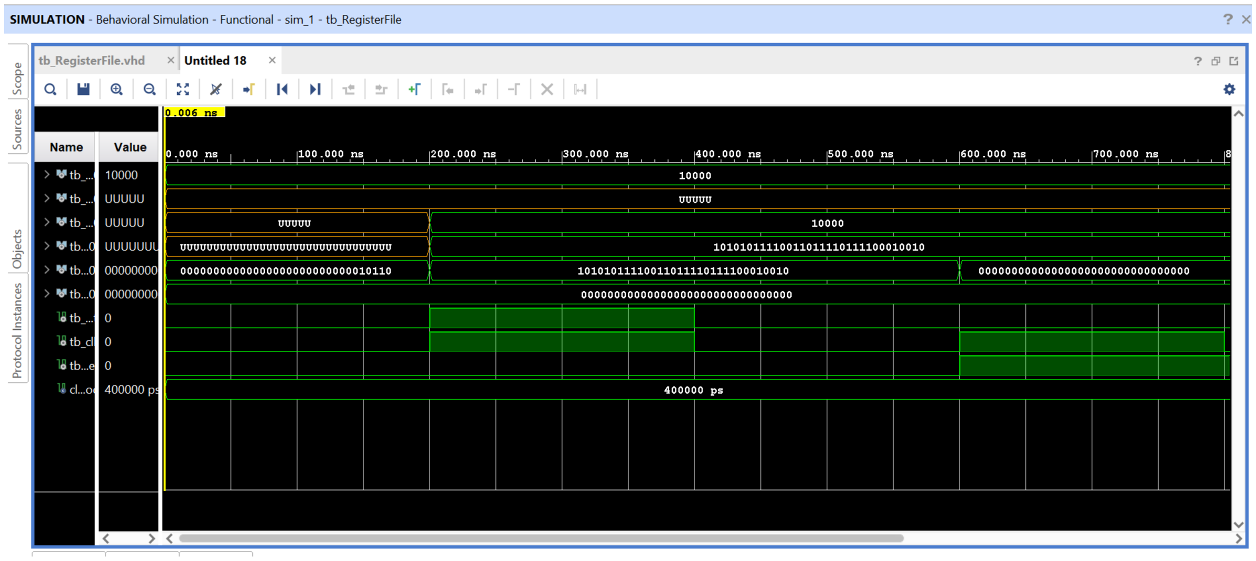
Mux Simulation



PC Simulation



ALU Simulation



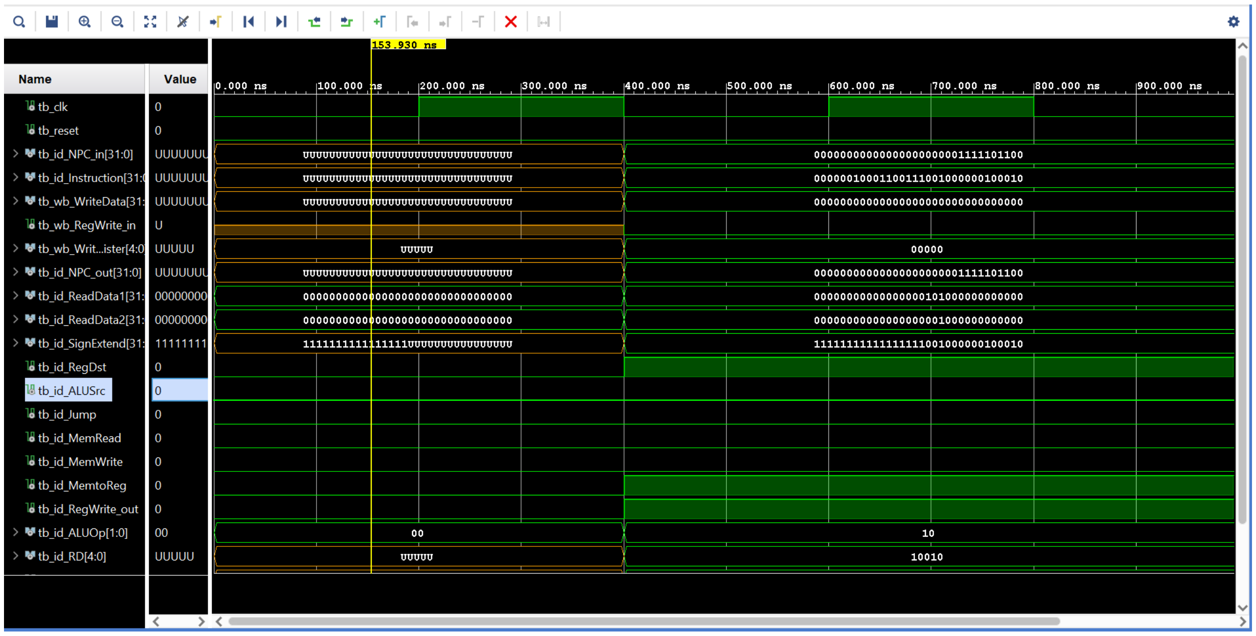
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_RegisterFile

tb_RegisterFile.vhd | Untitled 18

Name	Value
tb_...	10000
tb_...	UUUUU
tb_...	UUUUU
tb...0	UUUUUUU
tb...0	00000000
tb...0	00000000
tb_...	0
tb_cl	0
tb...e	0
cl...o	400000 ps

```
0.006 ns
0.000 ns   100.000 ns   200.000 ns   300.000 ns   400.000 ns   500.000 ns   600.000 ns   700.000 ns
10000
UUUUU
UUUUU                              10000
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   10101011110011011110111100010010
00000000000000000000000000010110   10101011110011011110111100010010   00000000000000000000000000000000
00000000000000000000000000000000
400000 ps
```

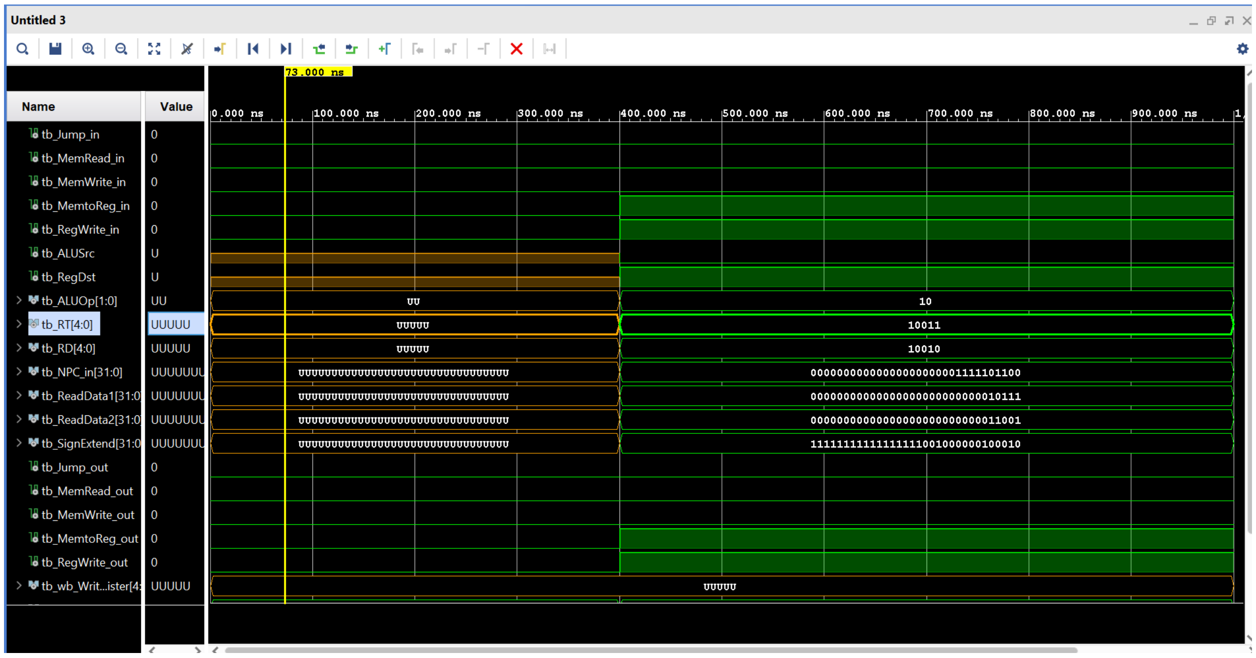
RegFile Simulation



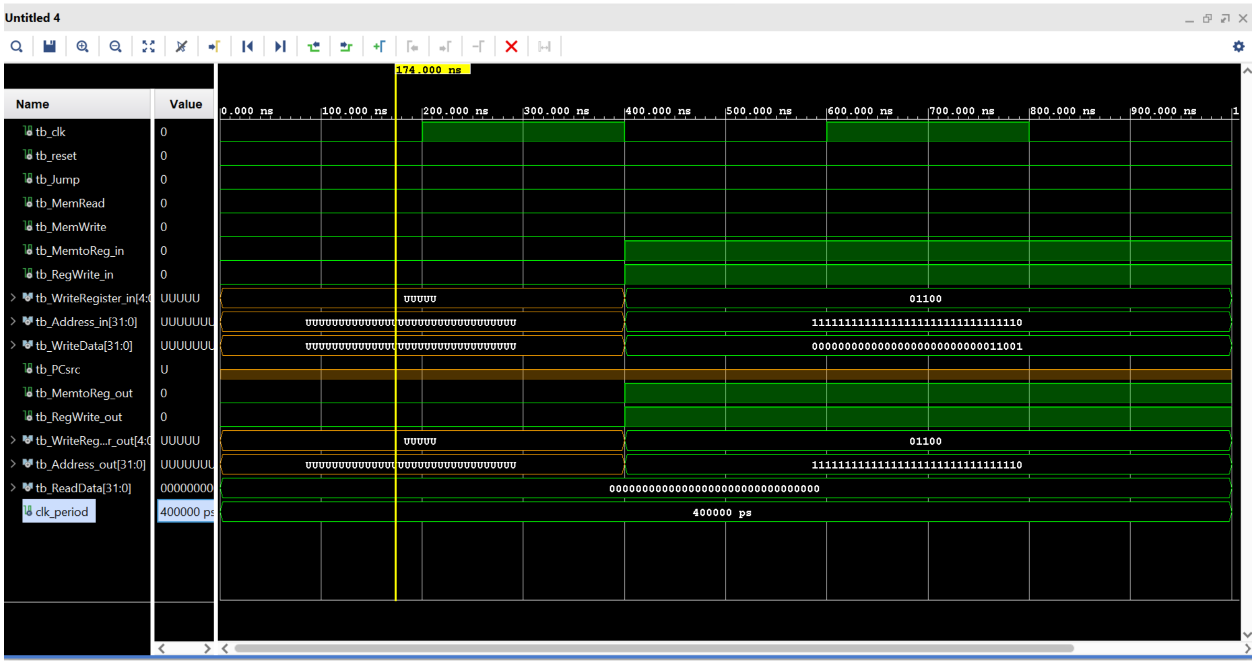
Name	Value
tb_clk	0
tb_reset	0
tb_id_NPC_in[31:0]	UUUUUUU
tb_id_Instruction[31:0]	UUUUUUU
tb_wb_WriteData[31:	UUUUUUU
tb_wb_RegWrite_in	U
tb_wb_Writ...ister[4:0]	UUUUU
tb_id_NPC_out[31:0]	UUUUUUU
tb_id_ReadData1[31:	00000000
tb_id_ReadData2[31:	00000000
tb_id_SignExtend[31:	11111111
tb_id_RegDst	0
tb_id_ALUSrc	0
tb_id_Jump	0
tb_id_MemRead	0
tb_id_MemWrite	0
tb_id_MemtoReg	0
tb_id_RegWrite_out	0
tb_id_ALUOp[1:0]	00
tb_id_RD[4:0]	UUUUU

```
153.930 ns
0.000 ns   100.000 ns   200.000 ns   300.000 ns   400.000 ns   500.000 ns   600.000 ns   700.000 ns   800.000 ns   900.000 ns
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000000000001111101100
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000010001100111001000000100010
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000000000000000000000
UUUUU                              00000
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000000000001111101100
00000000000000000000000000000000   00000000000000000101000000000000
00000000000000000000000000000000   00000000000000000001000000000000
1111111111111111UUUUUUUUUUUUUUUU   11111111111111111001000000100010
00                                 10
UUUUU                              10010
```

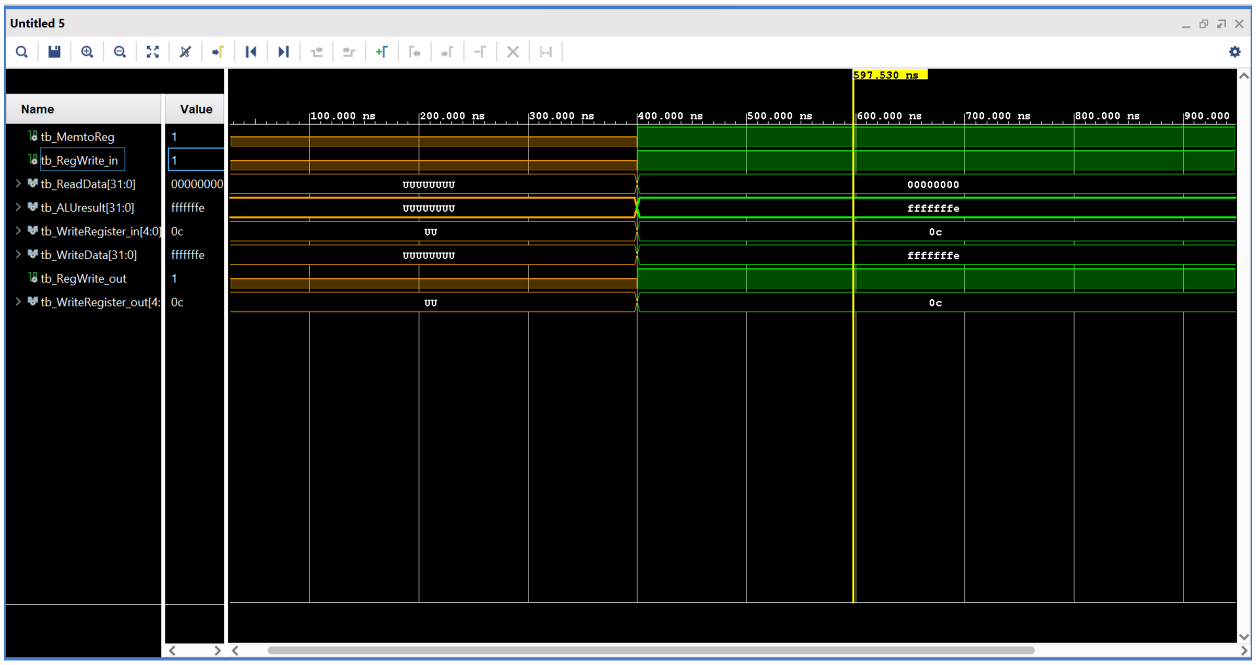
ID Stage Simulation



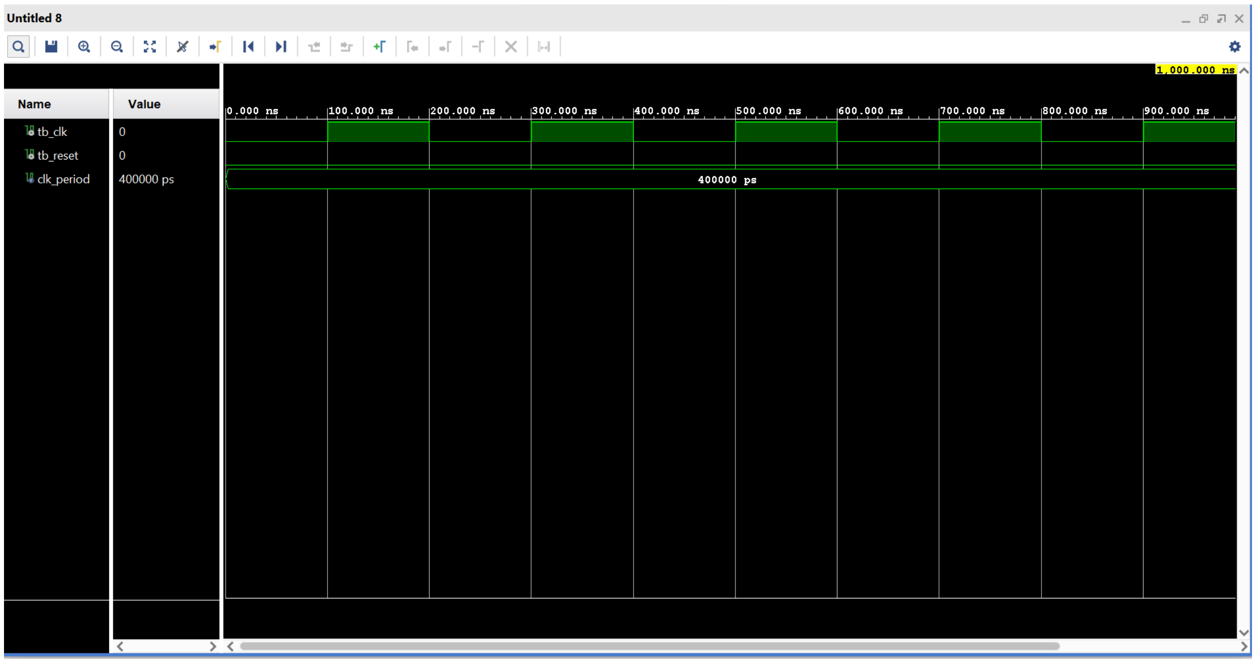
EX Stage Simulation



MEM Stage Simulation



WB Stage Simulation



Processor Testbench

Concluding Remarks

The significance of the 5-stage MIPS pipeline cannot be overstated, as it forms the cornerstone of our processor architecture, providing a structured framework for the seamless execution of instructions. Our deliberate decision to initially focus on the pipeline sans memory management and control interface complexities speaks to our strategic approach in tackling one layer of intricacies at a time.

In essence, this concerted effort is directed towards establishing a robust foundation for our processor, navigating the intricate roadmap that accommodates a spectrum of instruction types while upholding the stringent demands inherent in the MIPS pipeline architecture. As we progress, we remain steadfast in our commitment to achieving a processor that not only meets the technical specifications but also reflects the culmination of strategic design thinking and meticulous execution.

Appendices

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

use IEEE.NUMERIC_STD.ALL;

entity tb_ALU is
end tb_ALU;
```

```vhdl
architecture Behavioral of tb_ALU is
    signal tb_a1, tb_a2: std_logic_vector(31 downto 0);
    signal tb_alu_ctrl: std_logic_vector(3 downto 0);
    signal tb_Cin, tb_Cout: std_logic;
    signal tb_alu_rslt: std_logic_vector (31 downto 0);

begin

uut: entity work.ALU(Behavioral)
    port map(
        in1 => tb_a1,
        in2 => tb_a2,
        alu_ctrl => tb_alu_ctrl,
        Cin => tb_Cin,
        Cout => tb_Cout,
        alu_rslt => tb_alu_rslt
    );

 stim_proc: process
 begin

    tb_a1 <= x"FFFF0000";
    tb_a2 <= x"0000FFFF";
    tb_Cin <= '0';

    tb_alu_ctrl <= "0000"; --AND
    wait for 10ns;

    tb_alu_ctrl <= "0001"; --OR
    wait for 10ns;

    tb_alu_ctrl <= "0010"; --add, cout=0
    wait for 10ns;

    tb_Cin <= '1';
    tb_alu_ctrl <= "0010"; --add, cout=1
    wait for 10ns;

    tb_alu_ctrl <= "0110"; --sub
    wait for 10ns;

    tb_alu_ctrl <= "1100"; --NOR
    wait for 10ns;

    tb_alu_ctrl <= "1111"; --random error
    wait for 10ns;


 end process;

end Behavioral;
```

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

entity tb_ALUControl is
end tb_ALUControl;

architecture Behavioral of tb_ALUControl is

signal tb_ALUOp: std_logic_vector(1 downto 0);
signal tb_FunctionCode: std_logic_vector(5 downto 0);
signal tb_ALUOperation: std_logic_vector(3 downto 0);

begin

uut: entity work.ALUControl(Behavioral)
        port map(
                    ALUOp => tb_ALUOp,
                    FuncCode => tb_FunctionCode,
                    ALUOperation => tb_ALUOperation
                );

stim_Proc: Process
begin

tb_ALUOp <= "10";
tb_FunctionCode <= "100000";  --add
wait for 30ns;


tb_ALUOp <= "01";
tb_FunctionCode <= "XXXXXX";  --beq
wait for 30ns;


tb_ALUOp <= "01";
tb_FunctionCode <= "XXXXXX";  --bne
wait for 30ns;


tb_ALUOp <= "10";
tb_FunctionCode <= "100000";  --add
wait for 30ns;


tb_ALUOp <= "10";
tb_FunctionCode <= "100010";  --sub
wait for 30ns;


tb_ALUOp <= "10";
tb_FunctionCode <= "100100";  -- and
```

```
wait for 30ns;


tb_ALUOp <= "10";
tb_FunctionCode <= "100101"; -- or
wait for 30ns;


tb_ALUOp <= "10";
tb_FunctionCode <= "100111"; --nor
wait for 30ns;


tb_ALUOp <= "00";
tb_FunctionCode <= "XXXXXX";  --addi
wait for 30ns;


tb_ALUOp <= "00";
tb_FunctionCode <= "XXXXXX";  --lw
wait for 30ns;


tb_ALUOp <= "00";
tb_FunctionCode <= "XXXXXX";  --sw
wait for 30ns;


end process;


end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_WB_Stage is
end tb_WB_Stage;

architecture Structural of tb_WB_Stage is

signal tb_MemtoReg, tb_RegWrite_in: std_logic;
signal tb_ReadData, tb_ALUresult: std_logic_vector(31 downto 0);
signal tb_WriteRegister_in: std_logic_vector(4 downto 0);

signal tb_WriteData: std_logic_vector(31 downto 0);
signal tb_RegWrite_out: std_logic;
signal tb_WriteRegister_out: std_logic_vector(4 downto 0);

begin

uut: entity work.WB_Stage(Structural)
```

```vhdl
    port map(
            MemtoReg=>tb_MemtoReg,
            RegWrite_in=>tb_RegWrite_in,
            ReadData=>tb_ReadData,
            ALUresult=>tb_ALUresult,
            WriteRegister_in=>tb_WriteRegister_in,
            WriteData=>tb_WriteData,
            RegWrite_out=>tb_RegWrite_out,
            WriteRegister_out=>tb_WriteRegister_out
            );

stim_proc: process
begin
    wait for 400ns;
    tb_MemtoReg<='1';
    tb_RegWrite_in<='1';
    tb_ReadData<=x"00000000";
    tb_ALUresult<=x"FFFFFFFE";
    tb_WriteRegister_in<="01100";

end process;

end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_ControlUnit is
end tb_ControlUnit;

architecture Behavioral of tb_ControlUnit is

signal tb_OPcode: std_logic_vector(5 downto 0);
signal tb_RegDst, tb_MemRead, tb_MemtoReg, tb_MemWrite, tb_ALUSrc, tb_RegWrite, tb_PCscr:
std_logic;
signal tb_ALUOp: std_logic_vector(1 downto 0);

begin

uut: entity work.ControlUnit(Behavioral)
        port map(
                    OPcode => tb_OPcode,
                    RegDst=> tb_RegDst,
                    MemRead=>tb_MemRead,
                    MemtoReg=>tb_MemtoReg,
                    MemWrite=> tb_MemWrite,
                    ALUSrc=> tb_ALUSrc,
                    ALUOp => tb_ALUOp,
                    RegWrite=> tb_RegWrite,
                    PCscr=> tb_PCscr
                );
```

```vhdl
stim_proc: process
begin

    tb_OPcode <= "000000"; --R-type
    wait for 20ns;

    tb_OPcode<= "100011"; --lw
    wait for 20ns;

    tb_OPcode<= "000101"; --bne
    wait for 20ns;

    tb_OPcode<= "000100"; --beq
    wait for 20ns;

    tb_OPcode<= "101011"; --sw
    wait for 20ns;

    tb_OPcode<= "000010"; --jump
    wait for 20ns;

    tb_OPcode<= "001100"; --andi
    wait for 20ns;

    tb_OPcode<= "000101"; --bne
    wait for 20ns;

    tb_OPcode<= "111111"; --random error
    wait for 20ns;

    end process;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_WB_Stage is
end tb_WB_Stage;

architecture Structural of tb_WB_Stage is

signal tb_MemtoReg, tb_RegWrite_in: std_logic;
signal tb_ReadData, tb_ALUresult: std_logic_vector(31 downto 0);
signal tb_WriteRegister_in: std_logic_vector(4 downto 0);

signal tb_WriteData: std_logic_vector(31 downto 0);
signal tb_RegWrite_out: std_logic;
signal tb_WriteRegister_out: std_logic_vector(4 downto 0);

begin
```

```vhdl
uut: entity work.WB_Stage(Structural)
    port map(
            MemtoReg=>tb_MemtoReg,
            RegWrite_in=>tb_RegWrite_in,
            ReadData=>tb_ReadData,
            ALUresult=>tb_ALUresult,
            WriteRegister_in=>tb_WriteRegister_in,
            WriteData=>tb_WriteData,
            RegWrite_out=>tb_RegWrite_out,
            WriteRegister_out=>tb_WriteRegister_out
            );

stim_proc: process
begin
    wait for 400ns;
    tb_MemtoReg<='1';
    tb_RegWrite_in<='1';
    tb_ReadData<=x"00000000";
    tb_ALUresult<=x"FFFFFFFE";
    tb_WriteRegister_in<="01100";

end process;

end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_shiftleft2 is
end tb_shiftleft2;

architecture Behavioral of tb_shiftleft2 is
signal tb_input, tb_output: std_logic_vector(31 downto 0);
begin

uut: entity work.ShiftLeft2(Behavioral)
       port map(
        input => tb_input,
        output => tb_output
       );

stim_proc: process

    begin

        tb_input <= x"FFFFFFFF";
        wait for 10ns;
        tb_input <= x"12345678";
        wait for 10ns;
        end process;

end Behavioral;
```

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

entity tb_WB_Stage is
end tb_WB_Stage;

architecture Structural of tb_WB_Stage is

signal tb_MemtoReg, tb_RegWrite_in: std_logic;
signal tb_ReadData, tb_ALUresult: std_logic_vector(31 downto 0);
signal tb_WriteRegister_in: std_logic_vector(4 downto 0);

signal tb_WriteData: std_logic_vector(31 downto 0);
signal tb_RegWrite_out: std_logic;
signal tb_WriteRegister_out: std_logic_vector(4 downto 0);

begin

uut: entity work.WB_Stage(Structural)
    port map(
            MemtoReg=>tb_MemtoReg,
            RegWrite_in=>tb_RegWrite_in,
            ReadData=>tb_ReadData,
            ALUresult=>tb_ALUresult,
            WriteRegister_in=>tb_WriteRegister_in,
            WriteData=>tb_WriteData,
            RegWrite_out=>tb_RegWrite_out,
            WriteRegister_out=>tb_WriteRegister_out
            );

stim_proc: process
begin
    wait for 400ns;
    tb_MemtoReg<='1';
    tb_RegWrite_in<='1';
    tb_ReadData<=x"00000000";
    tb_ALUresult<=x"FFFFFFFE";
    tb_WriteRegister_in<="01100";

end process;

end Structural;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_WB_Stage is
end tb_WB_Stage;

architecture Structural of tb_WB_Stage is
```

```vhdl
signal tb_MemtoReg, tb_RegWrite_in: std_logic;
signal tb_ReadData, tb_ALUresult: std_logic_vector(31 downto 0);
signal tb_WriteRegister_in: std_logic_vector(4 downto 0);

signal tb_WriteData: std_logic_vector(31 downto 0);
signal tb_RegWrite_out: std_logic;
signal tb_WriteRegister_out: std_logic_vector(4 downto 0);

begin

uut: entity work.WB_Stage(Structural)
    port map(
            MemtoReg=>tb_MemtoReg,
            RegWrite_in=>tb_RegWrite_in,
            ReadData=>tb_ReadData,
            ALUresult=>tb_ALUresult,
            WriteRegister_in=>tb_WriteRegister_in,
            WriteData=>tb_WriteData,
            RegWrite_out=>tb_RegWrite_out,
            WriteRegister_out=>tb_WriteRegister_out
            );

stim_proc: process
begin
    wait for 400ns;
    tb_MemtoReg<='1';
    tb_RegWrite_in<='1';
    tb_ReadData<=x"00000000";
    tb_ALUresult<=x"FFFFFFFE";
    tb_WriteRegister_in<="01100";

end process;

end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_PC is
end tb_PC;

architecture Behavioral of tb_PC is
signal tb_Address_in, tb_Address_out: std_logic_vector(31 downto 0);

signal tb_clk: std_logic:= '0';
signal tb_reset: std_logic:= '0';

constant clk_period: time:= 400ns;

begin
```

```
uut: entity work.PC(Behavioral)
        port map(
                    clk=>tb_clk,
                    rst => tb_reset,
                    Address_in =>tb_Address_in,
                    Address_out=>tb_Address_out
                );

clk_process: process
begin

    tb_clk <= '0';
    wait for clk_period/2;

    tb_clk<= '1';
    wait for clk_period/2;

end process;

stim_proc: process
    begin

    tb_Address_in<=x"12345678";
    wait for 400ns;

    tb_reset<= '1';

end process;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_DataMemory is
end tb_DataMemory;


architecture Behavioral of tb_DataMemory is

signal tb_Address, tb_WriteData, tb_ReadData : std_logic_vector(31 downto 0);

signal tb_MemWrite, tb_MemRead: std_logic;

signal tb_clk: std_logic:= '0';
signal tb_reset: std_logic:= '0';

constant clk_period: time:= 400ns;


begin
```

```vhdl
uut: entity work.DataMemory(Behavioral)
        port map (
                    Address => tb_Address,
                    WriteData => tb_WriteData,
                    ReadData => tb_ReadData,
                    MemWrite => tb_MemWrite,
                    MemRead => tb_MemRead,
                    clk => tb_clk,
                    rst => tb_reset
                 );


clk_process: process
begin

tb_clk <= '0';
wait for clk_period/2;

tb_clk<= '1';
wait for clk_period/2;

end process;

stim_proc: process
begin

tb_Address <= x"10000000"; -- modifiy address 0x1000 0000
tb_WriteData <= x"12345678";

tb_MemWrite <= '0';
tb_MemRead <= '0';
wait for 400ns;

tb_MemWrite <= '0';
tb_MemRead <= '1';
wait for 400ns;

tb_MemWrite <= '1';
tb_MemRead <= '0';
wait for 400ns;

tb_MemWrite <= '0';
tb_MemRead <= '1';
wait for 400ns;

tb_Address <= x"10000004"; -- modify address 0x1000 0004
tb_WriteData <= x"12345678";

tb_MemWrite <= '0';
tb_MemRead <= '0';
wait for 400ns;
```

```vhdl
tb_MemWrite <= '0';
tb_MemRead <= '1';
wait for 400ns;

tb_MemWrite <= '1';
tb_MemRead <= '0';
wait for 400ns;

tb_MemWrite <= '0';
tb_MemRead <= '1';
wait for 400ns;

tb_reset<= '1';
wait for 400ns;

tb_MemRead <= '1';
tb_Address<=x"10000000";
wait for 400ns;
end process;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_PC is
end tb_PC;

architecture Behavioral of tb_PC is
signal tb_Address_in, tb_Address_out: std_logic_vector(31 downto 0);

signal tb_clk: std_logic:= '0';
signal tb_reset: std_logic:= '0';

constant clk_period: time:= 400ns;

begin

uut: entity work.PC(Behavioral)
        port map(
                    clk=>tb_clk,
                    rst => tb_reset,
                    Address_in =>tb_Address_in,
                    Address_out=>tb_Address_out
                );

clk_process: process
begin

    tb_clk <= '0';
    wait for clk_period/2;
    tb_clk<= '1';
    wait for clk_period/2;

end process;

stim_proc: process
    begin

    tb_Address_in<=x"12345678";
    wait for 400ns;

    tb_reset<= '1';

end process;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_EX_Stage is
end tb_EX_Stage;

architecture Structural of tb_EX_Stage is

signal tb_Jump_in, tb_MemRead_in, tb_MemWrite_in, tb_MemtoReg_in, tb_RegWrite_in:
std_logic:= '0';
signal tb_ALUSrc, tb_RegDst: std_logic;
signal tb_ALUOp: std_logic_vector(1 downto 0);

signal tb_RT, tb_RD: std_logic_vector(4 downto 0);

signal tb_NPC_in, tb_ReadData1, tb_ReadData2, tb_SignExtend: std_logic_vector(31 downto 0);

signal tb_Jump_out, tb_MemRead_out, tb_MemWrite_out, tb_MemtoReg_out, tb_RegWrite_out:
std_logic:= '0';
signal tb_wb_WriteRegister: std_logic_vector(4 downto 0);
signal tb_ALUresult, tb_JumpAddr, tb_WriteData: std_logic_vector( 31 downto 0);

begin

uut: entity work.EX_Stage(Structural)
    port map(
                RegWrite_in=>tb_RegWrite_in,
                RegWrite_out=>tb_RegWrite_out,

                MemtoReg_in=>tb_MemtoReg_in,
                MemtoReg_out=>tb_MemtoReg_out
                ,
                MemWrite_in=>tb_MemWrite_in,
                MemWrite_out=>tb_MemWrite_out,
                MemRead_in=>tb_MemRead_in,
                MemRead_out=>tb_MemRead_out,

                Jump_in=>tb_Jump_in,
                Jump_out=>tb_Jump_out,

                RegDst=>tb_RegDst,
                ALUSrc=>tb_ALUSrc,

                ALUOp=>tb_ALUOp,

                RT=>tb_RT,
                RD=>tb_RD,

                NPC_in=>tb_NPC_in,
                ReadData1=>tb_ReadData1,
                ReadData2=>tb_ReadData2,
                SignExtend=>tb_SignExtend,

                wb_WriteRegister=>tb_wb_WriteRegister,

                ALUresult=>tb_ALUresult,
                JumpAddr=>tb_JumpAddr,
                WriteData=>tb_WriteData
            );

stim_proc: process
begin
    wait for 400ns;

    tb_MemtoReg_in<='1';
    tb_RegDst<='1';
    tb_ALUSrc<= '0';
    tb_RegWrite_in<='1';
    tb_ReadData1<= x"00000017";
    tb_ReadData2<= x"00000019";
    tb_ALUOp<= "10";
    tb_SignExtend<=x"FFFF9022";
    tb_NPC_in<=x"000003EC";
    tb_RD<="10010";
    tb_RT<="10011";

    wait for 400ns;

end process;

end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_ID_Stage is
```

```vhdl
end tb_ID_Stage;

architecture Structural of tb_ID_Stage is

signal tb_clk: std_logic:= '0';
signal tb_reset: std_logic:= '0';

constant clk_period: time:= 400ns;

signal tb_id_NPC_in, tb_id_Instruction, tb_wb_WriteData: std_logic_vector(31 downto 0);
signal tb_wb_RegWrite_in: std_logic;
signal tb_wb_WriteRegister: std_logic_vector(4 downto 0);

signal tb_id_NPC_out, tb_id_ReadData1, tb_id_ReadData2, tb_id_SignExtend:
std_logic_vector(31 downto 0);
signal tb_id_RegDst, tb_id_ALUSrc, tb_id_Jump, tb_id_MemRead, tb_id_MemWrite,
tb_id_MemtoReg, tb_id_RegWrite_out: std_logic;
signal tb_id_ALUOp: std_logic_vector(1 downto 0);
signal tb_id_RD, tb_id_RT: std_logic_vector(4 downto 0);


begin

uut: entity work.ID_Stage(Structural)
    port map(
                clk => tb_clk,
                rst => tb_reset,
                NPC_in=> tb_id_NPC_in,
                Inst=> tb_id_Instruction,
                wb_WriteData=> tb_wb_WriteData,
                wb_RegWrite_in=> tb_wb_RegWrite_in,
                wb_WriteRegister=> tb_wb_WriteRegister,
                NPC_out=> tb_id_NPC_out,
                ReadData1=> tb_id_ReadData1,
                ReadData2=> tb_id_ReadData2,
                SignExtend1=> tb_id_SignExtend,
                ALUSrc=> tb_id_ALUSrc,
                PCscr=> tb_id_Jump,
                MemRead=> tb_id_MemRead,
                MemWrite=> tb_id_MemWrite,
                MemtoReg=> tb_id_MemtoReg,
                RegWrite_out=>tb_id_RegWrite_out,
                RegDst=>tb_id_RegDst,
                ALUOp=> tb_id_ALUOp,
                RD=> tb_id_RD,
                RT=> tb_id_RT
            );


clk_process: process
    begin
```

```
        tb_clk <= '0';
        wait for clk_period/2;

        tb_clk<= '1';
        wait for clk_period/2;

        end process;

stim_proc: process
begin
    wait for 400ns;

    tb_id_NPC_in <= x"000003EC";
    tb_id_Instruction<= x"02339022";
    tb_wb_WriteData<= x"00000000";
    tb_wb_RegWrite_in<= '0';
    tb_wb_WriteRegister<= "00000";

    wait for 400ns;

    end process;


end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Tb_MUX is
end Tb_MUX;

architecture Behavioral of Tb_MUX is

signal tb_a : STD_LOGIC_VECTOR(31 downto 0);
signal tb_b : STD_LOGIC_VECTOR(31 downto 0);
signal tb_sel : STD_LOGIC;
signal tb_y : STD_LOGIC_VECTOR(31 downto 0);


begin

uut: entity work.MUX(Behavioral)
    port map(
        a => tb_a,
        b => tb_b,
        sel => tb_sel,
        y => tb_y
    );

stim_proc : process
begin
   tb_a <= x"ffffffff";
```

```vhdl
    tb_b <= x"00000000";

     tb_sel <= '0';
     wait for 100ns;

     tb_sel <= '1';
     wait for 100ns;

    --tb_a <= "11111";
    --tb_b <= "10101";

    --tb_sel <= '0';
    --wait for 100ns;

    --tb_sel <= '1';
    --wait for 100ns;

end process;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_IF_stage is
end tb_IF_stage;

architecture Structural of tb_IF_stage is

signal tb_clk: std_logic:= '0';
signal tb_reset: std_logic:= '0';

constant clk_period: time:= 400ns;

signal tb_JumpAddr, tb_NPC, tb_Instruction_out: std_logic_vector(31 downto 0);
signal tb_PCsrc: std_logic;

begin

uut: entity work.IF_Stage(Structural)
    port map(
                clk => tb_clk,
                rst => tb_reset,
                JumpAddr => tb_JumpAddr,
                NPC=> tb_NPC,
                Instruction_out=>tb_Instruction_out,
                PCsrc=>tb_PCsrc
            );

clk_process: process
begin
```

```
    tb_clk <= '0';
    wait for clk_period/2;

    tb_clk<= '1';
    wait for clk_period/2;

    end process;

stim_proc: process
begin
    wait for 400ns;
    tb_JumpAddr<=x"00000000";
    tb_reset<='0';
    tb_PCsrc<='0';
    wait for 400ns;

    tb_reset<='1';
    wait for 200ns;

    end process;

end Structural;

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

entity tb_InstructionMemory is
end tb_InstructionMemory;

architecture Behavioral of tb_InstructionMemory is

signal tb_Address : std_logic_vector(31 downto 0) := x"000003E4";
signal tb_Instruction: std_logic_vector(31 downto 0);

begin

uut: entity work.InstructionMemory
    port map(
                Address => tb_Address,
                Inst => tb_Instruction
            );

stim_proc: process

begin

    for i in 0 to 15 loop
        tb_Address <= x"000003E8" or std_logic_vector(to_unsigned(i*4,32));
        wait for 30ns;
    end loop;
    wait;
```

```
end process;


end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_MEM_Stage is
end tb_MEM_Stage;

architecture Structural of tb_MEM_Stage is

signal tb_clk: std_logic:= '0';
signal tb_reset: std_logic:= '0';

constant clk_period: time:= 400ns;

signal tb_Jump, tb_MemRead, tb_MemWrite, tb_MemtoReg_in, tb_RegWrite_in: std_logic:='0';
signal tb_WriteRegister_in: std_logic_vector(4 downto 0);
signal tb_Address_in, tb_WriteData: std_logic_vector(31 downto 0);

signal tb_PCsrc,tb_MemtoReg_out, tb_RegWrite_out: std_logic;
signal tb_WriteRegister_out: std_logic_vector(4 downto 0);
signal tb_Address_out, tb_ReadData: std_logic_vector(31 downto 0);

begin

uut: entity work.MEM_Stage(Structural)
        port map(
                    clk => tb_clk,
                    rst => tb_reset,
                    Jump_in => tb_Jump,
                    MemRead=>tb_MemRead,
                    MemWrite=>tb_MemWrite,
                    WriteRegister_in=>tb_WriteRegister_in,
                    Address_in=>tb_Address_in,
                    WriteData=>tb_WriteData,
                    jump_out=>tb_PCsrc,
                    MemtoReg_in=>tb_MemtoReg_in,
                    RegWrite_in=>tb_RegWrite_in,
                    MemtoReg_out=>tb_MemtoReg_out,
                    RegWrite_out=>tb_RegWrite_out,
                    WriteRegister_out=>tb_WriteRegister_out,
                    Address_out=>tb_Address_out,
                    ReadData=>tb_ReadData
                );

clk_process: process
begin
```

```
tb_clk <= '0';
wait for clk_period/2;

tb_clk<= '1';
wait for clk_period/2;

end process;

stim_proc: process
begin
    wait for 400ns;
    tb_MemtoReg_in<='1';
    tb_RegWrite_in<='1';
    tb_WriteRegister_in<="01100";
    tb_Address_in<=x"FFFFFFFE";
    tb_WriteData<=x"00000019";

end process;


end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity One_bit_RCA is
    Port ( A, B, CIN : in STD_LOGIC;
           Sum, COUT : out STD_LOGIC);
end One_bit_RCA;

architecture Behavioral of One_bit_RCA is
begin

Sum <= A xor B xor Cin;
Cout <= (A and B) or (A and Cin) or (B and Cin);

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Four_bit_RCA is
    Port ( A,B : in STD_LOGIC_VECTOR (3 downto 0);
           Cin : in STD_LOGIC;
           Sum : out STD_LOGIC_VECTOR (3 downto 0);
           Cout : out STD_LOGIC);
end Four_bit_RCA;

architecture Structural of Four_bit_RCA is

Component One_bit_RCA
Port (
    A,B,CIN: in STD_LOGIC;
    Sum,Cout: out STD_LOGIC
);
End Component;

signal s1,s2,s3: STD_LOGIC;
begin

    A1: One_bit_RCA port map ( A(0), B(0), Cin, Sum(0), s1);
    A2: One_bit_RCA port map ( A(1), B(1), s1, Sum(1), s2);
    A3: One_bit_RCA port map ( A(2), B(2), s2, Sum(2), s3);
    A4: One_bit_RCA port map ( A(3), B(3), s3, Sum(3), Cout);

end Structural;

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

entity ALU is
    Port ( in1, in2: in STD_LOGIC_VECTOR (31 downto 0);
           Cin: in std_logic;
           alu_ctrl : in STD_LOGIC_VECTOR (3 downto 0);
           Cout: out std_logic;
           alu_rslt : out STD_LOGIC_VECTOR (31 downto 0));
end ALU;

architecture Behavioral of ALU is

signal temp_rslt: std_logic_vector(31 downto 0);
signal temp_add_rslt: std_logic_vector(31 downto 0);
signal temp_cout: std_logic;

Component Thirtytwo_bit_RCA
 Port (
       A, B : in STD_LOGIC_VECTOR (31 downto 0);
       Cin : in STD_LOGIC;
       Sum : out STD_LOGIC_VECTOR (31 downto 0);
       Cout : out STD_LOGIC
       );
End Component;

begin
RCA: Thirtytwo_bit_RCA port map ( in1, in2, Cin, temp_add_rslt, temp_cout);
    process(in1, in2, alu_ctrl)
    begin
        case alu_ctrl is
            when "0000" => -- AND
                temp_rslt <= in1 and in2;
            when "0110" => -- SUB
                temp_rslt <= std_logic_vector(unsigned(in1) - unsigned(in2));
            when "0001" => -- OR
```

```vhdl
                temp_rslt <= in1 or in2;
            when "0010" => -- ADD
                temp_rslt <= temp_add_rslt;
            when "1100" => -- NOR
                temp_rslt <= in1 nor in2;
            when others => null;
                temp_rslt <= x"00000000";
           end case;
    end process;

alu_rslt <= temp_rslt;
Cout <= temp_cout;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ALUControl is
    Port ( ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           FuncCode : in STD_LOGIC_VECTOR (5 downto 0);
           ALUOperation : out STD_LOGIC_VECTOR (3 downto 0));
end ALUControl;

architecture Behavioral of ALUControl is
begin

    ALUOperation(0) <= (FuncCode(0) or FuncCode(3)) and ALUOp(1) ;
    ALUOperation(1) <= (not ALUOp(1)) or (not FuncCode(2));
    ALUOperation(2) <= (ALUOp(1) and FuncCode(1)) or ALUOp(0);
    ALUOperation(3) <= '0';

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ControlUnit is
    Port ( OPcode : in STD_LOGIC_VECTOR (5 downto 0);
           ALUOp : out STD_LOGIC_VECTOR(1 downto 0);
           RegDst, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite, PCscr : out STD_LOGIC
           );
end ControlUnit;

architecture Behavioral of ControlUnit is
begin
    process(OPcode)
    begin
        case OPcode is
               when "000000" =>  -- r-type (or)(add)(nor)(and)
                    RegDst <= '1';
                    MemRead <= '0';
                    MemtoReg <= '1';
                    ALUOp <= "10";
                    MemWrite <= '0';
                    ALUSrc <= '0';
                    RegWrite <='1';
                    PCscr <= '0';

                when "000101" =>   --bne
                    RegDst <= '1';
                    MemRead <= '0';
                    MemtoReg <= '1';
                    ALUOp <= "01";
                    MemWrite <= '0';
                    ALUSrc <= '0';
                    RegWrite <='1';
                    PCscr <= '0';

                when "100011" =>  --lw
                    RegDst <= '0';
                    MemRead <= '1';
                    MemtoReg <= '1';
                    ALUOp <= "00";
                    MemWrite <= '0';
                    ALUSrc <= '1';
                    RegWrite <='1';
                    PCscr <= '0';

                when "101011" =>  --sw
                    RegDst <= 'X';
                    MemRead <= '0';
                    MemtoReg <= 'X';
                    ALUOp <= "00";
                    MemWrite <= '1';
                    ALUSrc <= '1';
                    RegWrite <= '0';
                    PCscr <= '0';

                when "000010" =>  --jump
                    RegDst <= 'X';
                    MemRead <= '0';
                    MemtoReg <= 'X';
                    ALUOp <= "00";
                    MemWrite <= '0';
                    ALUSrc <= '0';
                    RegWrite <='0';
                    PCscr <= '1';

                 when "000100" =>  --beq
                    RegDst <= '1';
                    MemRead<= '0';
                    MemtoReg<= '1';
                    ALUOp<= "01";
```

```
                     MemWrite<= '0';
                     ALUSrc<= '0';
                     RegWrite<='1';
                     PCscr <= '0';

                  when "001101" =>  --ori
                     RegDst <= '1';
                     MemRead<= '0';
                     MemtoReg<= '0';
                     ALUOp<= "10";
                     MemWrite<= '0';
                     ALUSrc<= '1';
                     RegWrite<='1';
                     PCscr <= '0';

                  when "001100" =>  --andi
                     RegDst <= '1';
                     MemRead<= '0';
                     MemtoReg<= '0';
                     ALUOp<= "10";
                     MemWrite<= '0';
                     ALUSrc<= '1';
                     RegWrite<='1';
                     PCscr <= '0';

                when others=> null;
                     RegDst <= '0';
                     MemRead<= '0';
                     MemtoReg<= '0';
                     ALUOp<= "00";
                     MemWrite<= '0';
                     ALUSrc<= '0';
                     RegWrite<='0';
                     PCscr <= '0';
        end case;
    end process;
end Behavioral;

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

entity DataMemory is
    Port (
           clk, rst: in  STD_LOGIC;
           Address, WriteData  : in STD_LOGIC_VECTOR (31 downto 0);
           MemRead, MemWrite : in std_logic;
           ReadData : out STD_LOGIC_VECTOR (31 downto 0));
end DataMemory;

architecture Behavioral of DataMemory is
type mem is array (0 to 20) of std_logic_vector(31 downto 0);
```

```
signal DataMem: mem := ( x"00000001",
                         x"00000002",
                         x"00000003",
                         x"00000004",
                         x"00000005",
                         x"00000006",
                         x"00000007",
                         x"00000008",
                         x"00000009",
                         x"00000010",
                         x"00000011",
                         x"00000012",
                         x"00000013",
                         x"00000014",
                         x"00000015",
                         x"00000016",
                         x"00000017",
                         x"00000018",
                         x"00000019",
                         x"00000020",
                         x"00000021"
                       );
begin
    process(clk)
    begin
      if (rst = '1') then
        DataMem<= (others=>(others=>'0'));
        -- 0x1000 0000 is 268435456 in decimal
        elsif (rising_edge(clk) and MemWrite = '1' ) then
            if Address < x"10000000" then
                DataMem(0) <= WriteData;
            else
                DataMem(to_integer(unsigned(Address))- 268435456 / 4) <= WriteData;
            end if;
      end if;

      if (Address = "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU" or Address < x"10000000") then
        ReadData<=x"00000000";
        elsif (MemRead='1') then
            ReadData <= DataMem((to_integer(unsigned(Address)) - 268435456) / 4);
      end if;
    end process;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity EX_MEM_Register is
    Port (
           clk, rst, RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in : in
STD_LOGIC ;
           RegWrite_out,MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;
           ALUresult_in, WriteData_in, JumpAddr_in : in STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           ALUresult_out, WriteData_out, JumpAddr_out : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0)
           );
end EX_MEM_Register;

architecture Behavioral of EX_MEM_Register is
begin
    process(clk)
    begin

        if rst = '1' then

            RegWrite_out <= '0';
            MemWrite_out<='0';
            MemRead_out<='0';
            MemtoReg_out <= '0';
            Jump_out<='0';

            WriteData_out<= x"00000000";
            WriteRegister_out<= "00000";
            JumpAddr_out<=x"00000000";
            ALUresult_out<= x"00000000";

        elsif rising_edge(clk) then
                RegWrite_out <= RegWrite_in;
                MemtoReg_out <= MemtoReg_in;
                MemWrite_out<=MemWrite_in;
                MemRead_out<=MemRead_in;
                Jump_out<=Jump_in;
                ALUresult_out<= ALUresult_in;
                WriteData_out<= WriteData_in;
                WriteRegister_out<= WriteRegister_in;
                JumpAddr_out<=JumpAddr_in;
        end if;
    end process;
end Behavioral;



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity EX_Stage is
    Port (
           RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in, RegDst, ALUSrc : in
STD_LOGIC;
           RegWrite_out, MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;
           ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           NPC_in, ReadData1, ReadData2, SignExtend : in STD_LOGIC_VECTOR (31 downto 0);
           RT, RD : in STD_LOGIC_VECTOR (4 downto 0);

           JumpAddr, ALUresult, WriteData : out STD_LOGIC_VECTOR (31 downto 0);
           wb_WriteRegister: out STD_LOGIC_VECTOR (4 downto 0));
end EX_Stage;

architecture Structural of EX_Stage is

component ShiftLeft2 is
    Port ( input : in STD_LOGIC_VECTOR (31 downto 0);
           output : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component Thirtytwo_bit_RCA is
    Port (     A,B : in STD_LOGIC_VECTOR (31 downto 0);
               Cin : in STD_LOGIC;
               Sum : out STD_LOGIC_VECTOR (31 downto 0);
               Cout : out STD_LOGIC);
end component;

component MUX is
    port(
            a, b: in std_logic_vector(31 downto 0);
            sel: in std_logic;
            y: out std_logic_vector(31 downto 0)
        );
end component;

component ALU is
    Port ( in1, in2 : in STD_LOGIC_VECTOR (31 downto 0);
           Cin: in std_logic;
           alu_ctrl : in STD_LOGIC_VECTOR (3 downto 0);
           Cout: out std_logic;
           alu_rslt : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component ALUControl is
    Port ( ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           FuncCode : in STD_LOGIC_VECTOR (5 downto 0);
           ALUOperation : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component MUX_5bit is
    port(
            input1,input2: in std_logic_vector(4 downto 0);
            ctrl: in std_logic;
            output: out std_logic_vector(4 downto 0)
        );
end component;

signal signextend_out, MUX32bit_out: std_logic_vector(31 downto 0);
signal MUX5bit_out: std_logic_vector(4 downto 0);
signal ALUControl_out: std_logic_vector(3 downto 0);
signal add_c_out, alu_c_out: std_logic;

begin

Shifter: Shiftleft2 port map(SignExtend, signextend_out);

RCA: Thirtytwo_bit_RCA port map(NPC_in, signextend_out, '0', JumpAddr, add_c_out);

MUX32bit: MUX port map(ReadData2, SignExtend, ALUSrc, MUX32bit_out);

A_L_U: ALU port map(ReadData1, MUX32bit_out, '0', ALUControl_out, alu_c_out, ALUresult);

A_L_U_ctrl: ALUControl port map (ALUOp, SignExtend(5 downto 0), ALUControl_out);


wb_WriteRegister <= MUX5bit_out;
RegWrite_out <= RegWrite_in;
MemtoReg_out <= MemtoReg_in;
MemWrite_out <= MemWrite_in;
MemRead_out <= MemRead_in;
Jump_out <= Jump_in;
WriteData <= ReadData2;

end Structural;



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ID_EX_Register is
    Port (
           clk, rst : in STD_LOGIC;

           RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in : in STD_LOGIC;
           RegWrite_out, MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;

           RegDst_in, ALUSrc_in : in STD_LOGIC;
           ALUOp_in : in STD_LOGIC_VECTOR (1 downto 0);
           RegDst_out, ALUSrc_out : out STD_LOGIC;
           ALUOp_out : out STD_LOGIC_VECTOR (1 downto 0);

           NPC_in, A1_in, A2_in, signext_in: in std_logic_vector(31 downto 0);
           rt_in, rd_in: in std_logic_vector(4 downto 0);

           NPC_out, A1_out, A2_out, signext_out: out std_logic_vector(31 downto 0);
           rt_out, rd_out: out std_logic_vector(4 downto 0)
           );

end ID_EX_Register;
```

```
architecture Behavioral of ID_EX_Register is
begin
    process(clk)
    begin

        if rst = '1' then

            MemWrite_out<='0';
            RegWrite_out <= '0';
            MemtoReg_out <= '0';
            MemRead_out<='0';
            Jump_out<='0';
            RegDst_out<='0';
            ALUOp_out<= "00";
            ALUSrc_out<= '0';
            NPC_out<= x"00000000";
            A1_out<= x"00000000";
            A2_out<= x"00000000";
            signext_out<= x"00000000";
            rt_out<= "00000";
            rd_out<="00000";

            elsif rising_edge(clk) then
                RegWrite_out <= RegWrite_in;
                MemtoReg_out <= MemtoReg_in;
                MemWrite_out<=MemWrite_in;
                MemRead_out<=MemRead_in;
                Jump_out<=Jump_in;
                RegDst_out<=RegDst_in;
                ALUOp_out<= ALUOp_in;
                ALUSrc_out<= ALUSrc_in;
                NPC_out<= NPC_in;
                A1_out<= A1_in;
                A2_out<= A2_in;
                signext_out<= signext_in;
                rt_out<= rt_in;
                rd_out<=rd_in;
    end if;
end process;



end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ID_Stage is
    Port ( clk, rst: in STD_LOGIC;
           NPC_in, Inst : in STD_LOGIC_VECTOR (31 downto 0);
           NPC_out : out STD_LOGIC_VECTOR (31 downto 0);
           wb_WriteRegister : in STD_LOGIC_VECTOR (4 downto 0);
```

```vhdl
           wb_WriteData : in STD_LOGIC_VECTOR (31 downto 0);
           ReadData1, ReadData2, SignExtend1 : out STD_LOGIC_VECTOR (31 downto 0);
           RT, RD : out STD_LOGIC_VECTOR (4 downto 0);
           wb_RegWrite_in : in STD_LOGIC;
           RegWrite_out, MemtoReg, MemWrite, MemRead, PCscr, RegDst, ALUSrc : out STD_LOGIC;
           ALUOp : out STD_LOGIC_VECTOR (1 downto 0)
           );
end ID_Stage;

architecture Structural of ID_Stage is

component SignExtend is
    port (
           Input : in STD_LOGIC_VECTOR(15 downto 0);
           Output : out STD_LOGIC_VECTOR(31 downto 0)
         );
end component;

component ControlUnit is
    port(
           OPcode : in STD_LOGIC_VECTOR (5 downto 0);
           RegDst,MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite, PCscr : out STD_LOGIC;
           ALUOp : out STD_LOGIC_VECTOR(1 downto 0)
        );
end component;

component RegisterFile is
    port(
           RegWrite,clk, rst : in STD_LOGIC;
           ReadRegister1,ReadRegister2,WriteRegister : in STD_LOGIC_VECTOR (4 downto 0);
           WriteData : in STD_LOGIC_VECTOR (31 downto 0);
           ReadData1, ReadData2 : out STD_LOGIC_VECTOR (31 downto 0)
        );
end component;

begin

Control: ControlUnit port map(Inst(31 downto 26), RegDst, MemRead, MemtoReg, MemWrite,
ALUSrc, RegWrite_out, PCscr, ALUOp);
RegFile: RegisterFile port map(clk, rst, wb_RegWrite_in, Inst(25 downto 21), Inst(20 downto
16),
 wb_WriteRegister, wb_WriteData,ReadData1, ReadData2);
EXT: SignExtend port map(Inst(15 downto 0), SignExtend1);

RT <= Inst(20 downto 16);
RD <= Inst(15 downto 11);
NPC_out <= NPC_in;
end Structural;

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

```vhdl
entity IF_ID_Register is
    Port (
           clk, rst: in std_logic;
           NPC_in,Inst_in : in STD_LOGIC_VECTOR (31 downto 0);
           NPC_out, Inst_out : out STD_LOGIC_VECTOR (31 downto 0));
end IF_ID_Register;

architecture Behavioral of IF_ID_Register is
begin
    process(clk)
    begin

        if rst ='1' then
            NPC_out <= x"00000000";
            Inst_out<= x"00000000";

            elsif rising_edge(clk) then
                NPC_out <= NPC_in;
                Inst_out<= Inst_in;
        end if;
    end process;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;



entity InstructionMemory is
    Port ( Address : in STD_LOGIC_VECTOR (31 downto 0);
           Inst : out STD_LOGIC_VECTOR (31 downto 0));
end InstructionMemory;

architecture Behavioral of InstructionMemory is
type mem16by32 is array (0 to 20) of std_logic_vector(31 downto 0);

signal InstMem: mem16by32 := (  x"8E4B009C",--10001110010010110000000010011100  -- 0x0000
03E8  --   lw $t3, 300($s2)
                                x"AEEE0120",--10101110111011100000000100100000  -- 0x0000
03EC  --   sw $t6, 400($s7)
                                x"16CD00C8",--00010110110011010000000011001000  -- 0x0000
03F0  --   bne $s6, $t5, 200
                                x"01696820",--00000001011010010110100000100000  -- 0x0000
03F4  --   add $t5, $t3, $s1
                                x"33160064",--00110011000101100000000001100100  -- 0x0000
03F8  --   andi $s6,$t8,100             --Salvador
                                x"11F80064",--00010001111110000000000001100100  -- 0x0000
03FC  --   beq $t7,$t8,100              --Mustafa
                                x"012A9825",--00000001001010101001100000100101  -- 0x0000
0400  --   or $s3,$t1,$t2               --Salvador
                                x"01F88824",--00000001111110001000100000100100  -- 0x0000
0404  --   and $s1,$t7,$t8    --Nand part 1 --Mustafa
                                x"00114827",--00000000000100010100100000100111   -- 0x0000
0408  --   nor $t1,$zero,$s1  --Nand part 2 --Mustafa
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000",
                                x"00000000"
                                 );

begin
    --0x0000 0358 => 10000 (decimal);
    Inst <= x"00000000" when Address = "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU" else
                   InstMem((to_integer(unsigned(Address)) - 1000) / 4);

end Behavioral;




library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Processor is
  Port ( clk,rst: in std_logic);
end Processor;

architecture Structural of Processor is

component IF_Stage is
    port(
           clk, rst, PCsrc : in STD_LOGIC;
           JumpAddr : in STD_LOGIC_VECTOR (31 downto 0);
           NPC, Instruction_out : out STD_LOGIC_VECTOR (31 downto 0)
        );
end component;

component IF_ID_Register is
    port(
           clk, rst: in std_logic;
           NPC_in, Inst_in : in STD_LOGIC_VECTOR (31 downto 0);
           NPC_out,Inst_out : out STD_LOGIC_VECTOR (31 downto 0)
        );
end component;
```

```
component ID_Stage is
    port(
           clk,rst : in STD_LOGIC;
           NPC_in, Inst : in STD_LOGIC_VECTOR (31 downto 0);
           NPC_out : out STD_LOGIC_VECTOR (31 downto 0);

           wb_WriteRegister : in STD_LOGIC_VECTOR (4 downto 0);
           wb_WriteData : in STD_LOGIC_VECTOR (31 downto 0);

           ReadData1, ReadData2, SignExtend1 : out STD_LOGIC_VECTOR (31 downto 0);
           RT, RD : out STD_LOGIC_VECTOR (4 downto 0);

           wb_RegWrite_in : in STD_LOGIC;

           RegWrite_out, MemtoReg, MemWrite, MemRead, PCscr, RegDst, ALUSrc : out STD_LOGIC;
           ALUOp : out STD_LOGIC_VECTOR (1 downto 0)
        );
end component;

component ID_EX_Register is
    port(
           clk,rst,RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in, RegDst_in,
ALUSrc_in  : in STD_LOGIC;
           RegWrite_out, MemtoReg_out, MemWrite_out, MemRead_out, Jump_out,
RegDst_out,ALUSrc_out : out STD_LOGIC;

           ALUOp_in : in STD_LOGIC_VECTOR (1 downto 0);
           ALUOp_out : out STD_LOGIC_VECTOR (1 downto 0);

           NPC_in, A1_in, A2_in, signext_in: in std_logic_vector(31 downto 0);
           rt_in,rd_in: in std_logic_vector(4 downto 0);

           NPC_out, A1_out, A2_out, signext_out: out std_logic_vector(31 downto 0);
           rt_out, rd_out: out std_logic_vector(4 downto 0)
        );
end component;

component EX_Stage is
    port(
           RegWrite_in,MemtoReg_in, MemWrite_in, MemRead_in, Jump_in,  RegDst,ALUSrc : in
STD_LOGIC;
           RegWrite_out, MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;

           ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           NPC_in, ReadData1, ReadData2, SignExtend : in STD_LOGIC_VECTOR (31 downto 0);
           RT, RD : in STD_LOGIC_VECTOR (4 downto 0);

           JumpAddr, ALUresult, WriteData : out STD_LOGIC_VECTOR (31 downto 0);
           wb_WriteRegister: out STD_LOGIC_VECTOR (4 downto 0)
        );
end component;
```

```vhdl
component EX_MEM_Register is
    port(
           clk,rst : in STD_LOGIC;

           RegWrite_in,MemtoReg_in : in STD_LOGIC;
           RegWrite_out, MemtoReg_out : out STD_LOGIC;

           MemWrite_in, MemRead_in, Jump_in : in STD_LOGIC;
           MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;

           ALUresult_in, WriteData_in, JumpAddr_in : in STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           ALUresult_out, WriteData_out, JumpAddr_out : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0)
        );
end component;

component MEM_Stage is
    port(
           clk, rst,Jump_in, MemRead, MemWrite, RegWrite_in, MemtoReg_in: in std_logic;
           Jump_out,RegWrite_out, MemtoReg_out : out STD_LOGIC;
           Address_in, WriteData, JumpAddr_in : in STD_LOGIC_VECTOR (31 downto 0);
           Address_out, ReadData, JumpAddr_out : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in: in std_logic_vector(4 downto 0);
           WriteRegister_out: out std_logic_vector(4 downto 0)
        );
end component;

component MEM_WB_Register is
    port(
           clk, rst, RegWrite_in, MemtoReg_in : in STD_LOGIC;
           RegWrite_out, MemtoReg_out : out STD_LOGIC;

           ReadData_in, ALUresult_in : in STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           ReadData_out, ALUresult_out : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0)
        );
end component;

component WB_Stage is
    port(
           RegWrite_in, MemtoReg : in STD_LOGIC;
           ReadData, ALUresult : in STD_LOGIC_VECTOR (31 downto 0);
           WriteData : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0);
           RegWrite_out : out STD_LOGIC
        );
end component;

signal MEM_Jump_out, mem_RegWrite_out, mem_MemtoReg_out: std_logic;
signal MEM_JumpAddr_out, mem_Address_out, mem_ReadData_out: std_logic_vector(31 downto 0);
signal mem_WriteRegister: std_logic_vector(4 downto 0);

signal IF_NPC, IF_Instr: std_logic_vector(31 downto 0);

signal wb_WriteRegister_out: std_logic_vector(4 downto 0);
signal wb_WriteData_out: std_logic_vector(31 downto 0);
signal wb_RegWrite_out: std_logic;

signal id_ReadData1_out, id_ReadData2_out, id_SignExtend_out: std_logic_vector(31 downto 0);
signal id_RT_out, id_RD_out: std_logic_vector(4 downto 0);
signal
id_RegWrite_out,id_MemtoReg_out,id_MemWrite_out,id_MemRead_out,id_Jump_out,id_RegDst_out,id_
ALUSrc_out: std_logic;
signal id_ALUOp_out: std_logic_vector(1 downto 0);

signal ex_RegWrite_out, ex_MemtoReg_out, ex_MemWrite_out, ex_MemRead_out, ex_Jump_out:
std_logic;
signal ex_JumpAddr_out, ex_ALUresult_out, ex_WriteData_out: std_logic_vector(31 downto 0);
signal ex_WriteRegister: std_logic_vector(4 downto 0);

signal IF_ID_NPC, IF_ID_Instr: std_logic_vector(31 downto 0);
signal ID_NPC: std_logic_vector(31 downto 0);

signal ID_EX_RegWrite, ID_EX_MemtoReg, ID_EX_MemWrite, ID_EX_MemRead, ID_EX_Jump,
ID_EX_RegDst, ID_EX_ALUSrc: std_logic;
signal ID_EX_ALUOp: std_logic_vector(1 downto 0);
signal ID_EX_NPC, ID_EX_A1, ID_EX_A2, ID_EX_Signext: std_logic_vector(31 downto 0);
signal ID_EX_RT, ID_EX_RD: STD_LOGIC_VECTOR(4 downto 0);

signal EX_MEM_RegWrite, EX_MEM_MemtoReg, EX_MEM_MemWrite, EX_MEM_MemRead, EX_MEM_Jump:
std_logic;
signal EX_MEM_ALUresult, EX_MEM_WriteData, EX_MEM_JumpAddr: std_logic_vector(31 downto 0);
signal EX_MEM_WriteRegister: std_logic_vector(4 downto 0);

signal MEM_WB_RegWrite, MEM_WB_MemtoReg: std_logic;
signal MEM_WB_ReadData, MEM_WB_ALUresult: std_logic_vector(31 downto 0);
signal MEM_WB_WriteRegister: std_logic_vector(4 downto 0);

begin

Stage_IF: IF_Stage port map(clk, rst, MEM_Jump_out,MEM_JumpAddr_out, IF_NPC, IF_Instr);

Reg1: IF_ID_Register port map(clk, rst, IF_NPC, IF_Instr, IF_ID_NPC, IF_ID_Instr);

Stage_ID: ID_Stage port map(clk, rst, IF_ID_NPC, IF_ID_Instr, ID_NPC,
                            wb_WriteRegister_out, wb_WriteData_out,
                            id_ReadData1_out,id_ReadData2_out,
                            id_SignExtend_out,id_RT_out,id_RD_out,
                            wb_RegWrite_out,id_RegWrite_out,
                            id_MemtoReg_out,id_MemWrite_out,
                            id_MemRead_out,id_Jump_out,
                            id_RegDst_out,id_ALUSrc_out,id_ALUOp_out);


Reg2: ID_EX_Register port map(clk, rst, id_RegWrite_out, id_MemtoReg_out,
id_MemWrite_out,id_MemRead_out,id_Jump_out, id_RegDst_out,id_ALUSrc_out,

ID_EX_RegWrite,ID_EX_MemtoReg,ID_EX_MemWrite,ID_EX_MemRead,ID_EX_Jump,ID_EX_RegDst,
ID_EX_ALUSrc,
                              ID_EX_ALUOp,id_ALUOp_out,

ID_NPC,id_ReadData1_out,id_ReadData2_out,id_SignExtend_out,id_RT_out,id_RD_out,
                              ID_EX_NPC,ID_EX_A1,ID_EX_A2,ID_EX_Signext, ID_EX_RT,ID_EX_RD);

Stage_EX: EX_Stage port map(ID_EX_RegWrite,
ID_EX_MemtoReg,ID_EX_MemWrite,ID_EX_MemRead,ID_EX_Jump,ID_EX_RegDst,ID_EX_ALUSrc,
                            ex_RegWrite_out, ex_MemtoReg_out, ex_MemWrite_out,
ex_MemRead_out, ex_Jump_out, ID_EX_ALUOp,
                            ID_EX_NPC, ID_EX_A1, ID_EX_A2, ID_EX_Signext, ID_EX_RT,
ID_EX_RD,
                            ex_JumpAddr_out, ex_ALUresult_out, ex_WriteData_out,
ex_WriteRegister);

Reg3: EX_MEM_Register port map(clk, rst, ex_RegWrite_out, ex_MemtoReg_out, EX_MEM_RegWrite,
EX_MEM_MemtoReg,
                              ex_MemWrite_out, ex_MemRead_out, ex_Jump_out, EX_MEM_MemWrite,
EX_MEM_MemRead, EX_MEM_Jump,
                              ex_ALUresult_out, ex_WriteData_out,ex_JumpAddr_out,
ex_WriteRegister, EX_MEM_ALUresult,
                              EX_MEM_WriteData,EX_MEM_JumpAddr,EX_MEM_WriteRegister);

Stage_MEM: MEM_Stage port map(clk, rst, EX_MEM_Jump, EX_MEM_MemRead, EX_MEM_MemWrite,
EX_MEM_RegWrite,
                              EX_MEM_MemtoReg, MEM_Jump_out, mem_RegWrite_out,
mem_MemtoReg_out,
                              EX_MEM_ALUresult, EX_MEM_WriteData, EX_MEM_JumpAddr,
mem_Address_out,
                              mem_ReadData_out, MEM_JumpAddr_out, EX_MEM_WriteRegister,
mem_WriteRegister);

Reg4: MEM_WB_Register port map(clk, rst, mem_RegWrite_out, mem_MemtoReg_out,
MEM_WB_RegWrite, MEM_WB_MemtoReg,
                              mem_ReadData_out, mem_Address_out, mem_WriteRegister,
                              MEM_WB_ReadData, MEM_WB_ALUresult, MEM_WB_WriteRegister);

    Stage_WB: WB_Stage port map(MEM_WB_RegWrite, MEM_WB_MemtoReg, MEM_WB_ReadData,
MEM_WB_ALUresult,
                            wb_WriteData_out, MEM_WB_WriteRegister, wb_WriteRegister_out,
wb_RegWrite_out);

end Structural;
```

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




entity EX_Stage is
    Port (
           RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in, RegDst, ALUSrc : in
STD_LOGIC;
           RegWrite_out, MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;
           ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           NPC_in, ReadData1, ReadData2, SignExtend : in STD_LOGIC_VECTOR (31 downto 0);
           RT, RD : in STD_LOGIC_VECTOR (4 downto 0);

           JumpAddr, ALUresult, WriteData : out STD_LOGIC_VECTOR (31 downto 0);
           wb_WriteRegister: out STD_LOGIC_VECTOR (4 downto 0));
end EX_Stage;

architecture Structural of EX_Stage is

component ShiftLeft2 is
    Port ( input : in STD_LOGIC_VECTOR (31 downto 0);
           output : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component Thirtytwo_bit_RCA is
    Port (     A,B : in STD_LOGIC_VECTOR (31 downto 0);
               Cin : in STD_LOGIC;
               Sum : out STD_LOGIC_VECTOR (31 downto 0);
               Cout : out STD_LOGIC);
end component;

component MUX is
    port(
            a, b: in std_logic_vector(31 downto 0);
            sel: in std_logic;
            y: out std_logic_vector(31 downto 0)
        );
end component;

component ALU is
    Port ( in1, in2 : in STD_LOGIC_VECTOR (31 downto 0);
           Cin: in std_logic;
           alu_ctrl : in STD_LOGIC_VECTOR (3 downto 0);
           Cout: out std_logic;
           alu_rslt : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component ALUControl is
    Port ( ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           FuncCode : in STD_LOGIC_VECTOR (5 downto 0);
           ALUOperation : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component MUX_5bit is
    port(
            input1,input2: in std_logic_vector(4 downto 0);
            ctrl: in std_logic;
            output: out std_logic_vector(4 downto 0)
        );
end component;

signal signextend_out, MUX32bit_out: std_logic_vector(31 downto 0);
signal MUX5bit_out: std_logic_vector(4 downto 0);
signal ALUControl_out: std_logic_vector(3 downto 0);
signal add_c_out, alu_c_out: std_logic;

begin

Shifter: Shiftleft2 port map(SignExtend, signextend_out);

RCA: Thirtytwo_bit_RCA port map(NPC_in, signextend_out, '0', JumpAddr, add_c_out);

MUX32bit: MUX port map(ReadData2, SignExtend, ALUSrc, MUX32bit_out);

A_L_U: ALU port map(ReadData1, MUX32bit_out, '0', ALUControl_out, alu_c_out, ALUresult);

A_L_U_ctrl: ALUControl port map (ALUOp, SignExtend(5 downto 0), ALUControl_out);


wb_WriteRegister <= MUX5bit_out;
RegWrite_out <= RegWrite_in;
MemtoReg_out <= MemtoReg_in;
MemWrite_out <= MemWrite_in;
MemRead_out <= MemRead_in;
Jump_out <= Jump_in;
WriteData <= ReadData2;

end Structural;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MEM_WB_Register is
    Port ( clk, rst, RegWrite_in, MemtoReg_in  : in STD_LOGIC;
           RegWrite_out, MemtoReg_out : out STD_LOGIC;
           ReadData_in, ALUresult_in : in STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           ReadData_out, ALUresult_out : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0)
           );
end MEM_WB_Register;
```

```
architecture Behavioral of MEM_WB_Register is
begin
    process(clk)
    begin

        if rst = '1' then
            RegWrite_out <= '0';
            MemtoReg_out <= '0';

            elsif rising_edge(clk) then
                ReadData_out<= ReadData_in;
                ALUresult_out<=ALUresult_in;
                WriteRegister_out<=WriteRegister_in;
        end if;
end process;

end Behavioral;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX 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 MUX;

architecture Behavioral of MUX is
begin

    y <= a when sel = '0' else b;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PC is
    Port (
           clk, rst: in std_logic;
           Address_in : in STD_LOGIC_VECTOR (31 downto 0);
           Address_out : out STD_LOGIC_VECTOR (31 downto 0));
end PC;

architecture Behavioral of PC is

type mem is array (0 to 0) of std_logic_vector(31 downto 0);

signal reg: mem;
signal temp:std_logic_vector(31 downto 0);
```

```vhdl
begin
    process(clk)
    begin

        if (rst = '1') then
            reg(0)<= x"000003E8";
        elsif rising_edge(clk) then
            if Address_in = "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU" then
                reg(0) <= x"000003E8";
            else
                reg(0) <= Address_in;
            end if;
        end if;

end process;

Address_out<= reg(0);

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Thirtytwo_bit_RCA is
    Port ( A,B : in STD_LOGIC_VECTOR (31 downto 0);
           Cin : in STD_LOGIC;
           Sum : out STD_LOGIC_VECTOR (31 downto 0);
           Cout : out STD_LOGIC
           );
end Thirtytwo_bit_RCA;

architecture Structural of Thirtytwo_bit_RCA is
Component Four_bit_RCA
    Port (
    A,B: in STD_LOGIC_VECTOR (3 downto 0);
    Cin: in STD_LOGIC;
    Sum: out STD_LOGIC_VECTOR (3 downto 0);
    Cout: out STD_LOGIC
    );
End Component;

signal s1,s2,s3,s4,s5,s6,s7: STD_LOGIC;
begin
    a1:Four_bit_RCA port map ( A(3 downto 0), B(3 downto 0), Cin, Sum(3 downto 0), s1);
    a2:Four_bit_RCA port map ( A(7 downto 4), B(7 downto 4), s1, Sum(7 downto 4), s2);
    a3:Four_bit_RCA port map ( A(11 downto 8), B(11 downto 8), s2, Sum(11 downto 8), s3);
    a4:Four_bit_RCA port map ( A(15 downto 12), B(15 downto 12), s3, Sum(15 downto 12), s4);
    a5:Four_bit_RCA port map ( A(19 downto 16), B(19 downto 16), s4, Sum(19 downto 16), s5);
    a6:Four_bit_RCA port map ( A(23 downto 20), B(23 downto 20), s5, Sum(23 downto 20), s6);
    a7:Four_bit_RCA port map ( A(27 downto 24), B(27 downto 24), s6, Sum(27 downto 24), s7);
    a8:Four_bit_RCA port map ( A(31 downto 28), B(31 downto 28), s7, Sum(31 downto 28),
Cout);

end Structural;

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

entity RegisterFile is
    Port ( clk, rst, RegWrite : in STD_LOGIC;
           ReadRegister1,ReadRegister2, WriteRegister : in STD_LOGIC_VECTOR (0 to 4);
           WriteData : in STD_LOGIC_VECTOR (0 to 31);
           ReadData1, ReadData2 : out STD_LOGIC_VECTOR (0 to 31)
           );
end RegisterFile;

architecture Behavioral of RegisterFile is

type reg is array (0 to 31) of std_logic_vector (31 downto 0);

signal reg_array: reg := ( x"00000000",--$zero
                           x"00000001",--$at
                           x"00000002",--$v0
                           x"00000003",--$v1
                           x"00000004",--$a0
                           x"00000005",--$a1
                           x"00000006",--$a2
                           x"00000007",--$a3
                           x"00000012",--$t0
                           x"00000049",--$t1
                           x"11111111",--$t2
                           x"00000011",--$t3
                           x"00000012",--$t4
                           x"00000000",--$t5
                           x"00000014",--$t6
                           x"00000015",--$t7
                           x"00000016",--$s0
                           x"00005000",--$s1
                           x"00000018",--$s2
                           x"00001000",--$s3
                           x"00000020",--$s4
                           x"00000021",--$s5
                           x"00000022",--$s6
                           x"00000023",--$s7
                           x"00000024",--$t8
                           x"00000025",--$t9
                           x"00000026",--$k0
                           x"00000027",--$k1
                           x"00000028",--$gp
                           x"00000029",--$sp
                           x"00000030",--$fp
                           x"00000031"--$ra
```

```
                         );
begin

process(CLK)
begin
    if (rst ='1') then
        reg_array(to_integer(unsigned(WriteRegister))) <= (others => '0');
        elsif(rising_edge(CLK) and RegWrite = '1') then
            reg_array(to_integer(unsigned(WriteRegister))) <= WriteData;
    end if;

end process;

    ReadData1 <= reg_array(to_integer(unsigned(ReadRegister1)));
    ReadData2 <= reg_array(to_integer(unsigned(ReadRegister2)));
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity EX_MEM_Register is
    Port (
           clk, rst, RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in : in
STD_LOGIC ;
           RegWrite_out,MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;
           ALUresult_in, WriteData_in, JumpAddr_in : in STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           ALUresult_out, WriteData_out, JumpAddr_out : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0)
           );
end EX_MEM_Register;

architecture Behavioral of EX_MEM_Register is
begin
    process(clk)
    begin

        if rst = '1' then

            RegWrite_out <= '0';
            MemWrite_out<='0';
            MemRead_out<='0';
            MemtoReg_out <= '0';
            Jump_out<='0';

            WriteData_out<= x"00000000";
            WriteRegister_out<= "00000";
            JumpAddr_out<=x"00000000";
            ALUresult_out<= x"00000000";

        elsif rising_edge(clk) then
                RegWrite_out <= RegWrite_in;
                MemtoReg_out <= MemtoReg_in;
                MemWrite_out<=MemWrite_in;
                MemRead_out<=MemRead_in;
                Jump_out<=Jump_in;
                ALUresult_out<= ALUresult_in;
                WriteData_out<= WriteData_in;
                WriteRegister_out<= WriteRegister_in;
                JumpAddr_out<=JumpAddr_in;
        end if;
    end process;
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity EX_Stage is
    Port (
           RegWrite_in, MemtoReg_in, MemWrite_in, MemRead_in, Jump_in, RegDst, ALUSrc : in
STD_LOGIC;
           RegWrite_out, MemtoReg_out, MemWrite_out, MemRead_out, Jump_out : out STD_LOGIC;
           ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           NPC_in, ReadData1, ReadData2, SignExtend : in STD_LOGIC_VECTOR (31 downto 0);
           RT, RD : in STD_LOGIC_VECTOR (4 downto 0);

           JumpAddr, ALUresult, WriteData : out STD_LOGIC_VECTOR (31 downto 0);
           wb_WriteRegister: out STD_LOGIC_VECTOR (4 downto 0));
end EX_Stage;

architecture Structural of EX_Stage is

component ShiftLeft2 is
    Port ( input : in STD_LOGIC_VECTOR (31 downto 0);
           output : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component Thirtytwo_bit_RCA is
    Port (     A,B : in STD_LOGIC_VECTOR (31 downto 0);
               Cin : in STD_LOGIC;
               Sum : out STD_LOGIC_VECTOR (31 downto 0);
               Cout : out STD_LOGIC);
end component;

component MUX is
    port(
            a, b: in std_logic_vector(31 downto 0);
            sel: in std_logic;
            y: out std_logic_vector(31 downto 0)
        );
end component;

component ALU is
    Port ( in1, in2 : in STD_LOGIC_VECTOR (31 downto 0);
           Cin: in std_logic;
           alu_ctrl : in STD_LOGIC_VECTOR (3 downto 0);
           Cout: out std_logic;
           alu_rslt : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component ALUControl is
    Port ( ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
           FuncCode : in STD_LOGIC_VECTOR (5 downto 0);
           ALUOperation : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component MUX_5bit is
    port(
            input1,input2: in std_logic_vector(4 downto 0);
            ctrl: in std_logic;
            output: out std_logic_vector(4 downto 0)
        );
end component;

signal signextend_out, MUX32bit_out: std_logic_vector(31 downto 0);
signal MUX5bit_out: std_logic_vector(4 downto 0);
signal ALUControl_out: std_logic_vector(3 downto 0);
signal add_c_out, alu_c_out: std_logic;

begin

Shifter: Shiftleft2 port map(SignExtend, signextend_out);

RCA: Thirtytwo_bit_RCA port map(NPC_in, signextend_out, '0', JumpAddr, add_c_out);

MUX32bit: MUX port map(ReadData2, SignExtend, ALUSrc, MUX32bit_out);

A_L_U: ALU port map(ReadData1, MUX32bit_out, '0', ALUControl_out, alu_c_out, ALUresult);

A_L_U_ctrl: ALUControl port map (ALUOp, SignExtend(5 downto 0), ALUControl_out);


wb_WriteRegister <= MUX5bit_out;
RegWrite_out <= RegWrite_in;
MemtoReg_out <= MemtoReg_in;
MemWrite_out <= MemWrite_in;
MemRead_out <= MemRead_in;
Jump_out <= Jump_in;
WriteData <= ReadData2;

end Structural;


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

entity ShiftLeft2 is
    Port ( input : in STD_LOGIC_VECTOR (31 downto 0);
           output : out STD_LOGIC_VECTOR (31 downto 0));
end ShiftLeft2;

architecture Behavioral of ShiftLeft2 is
begin

output <= input(29 downto 0) & "00";

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

Output <= x"0000" & Input when Input(15) = '0' else x"FFFF" & Input;


end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity WB_Stage is
    Port ( RegWrite_in, MemtoReg : in STD_LOGIC;
           ReadData, ALUresult : in STD_LOGIC_VECTOR (31 downto 0);
           WriteData : out STD_LOGIC_VECTOR (31 downto 0);
           WriteRegister_in : in STD_LOGIC_VECTOR (4 downto 0);
           WriteRegister_out : out STD_LOGIC_VECTOR (4 downto 0);
           RegWrite_out : out STD_LOGIC
           );
end WB_Stage;

architecture Structural of WB_Stage is

component MUX is

    Port ( a, b : in STD_LOGIC_VECTOR(31 downto 0);
           sel : in STD_LOGIC;
           y : out STD_LOGIC_VECTOR(31 downto 0)
         );
end component;
```

```vhdl
begin
    MUX32bit: MUX port map(ReadData,ALUresult,MemtoReg,WriteData);
    RegWrite_out<=RegWrite_in;
    WriteRegister_out<= WriteRegister_in;


end Structural;
```