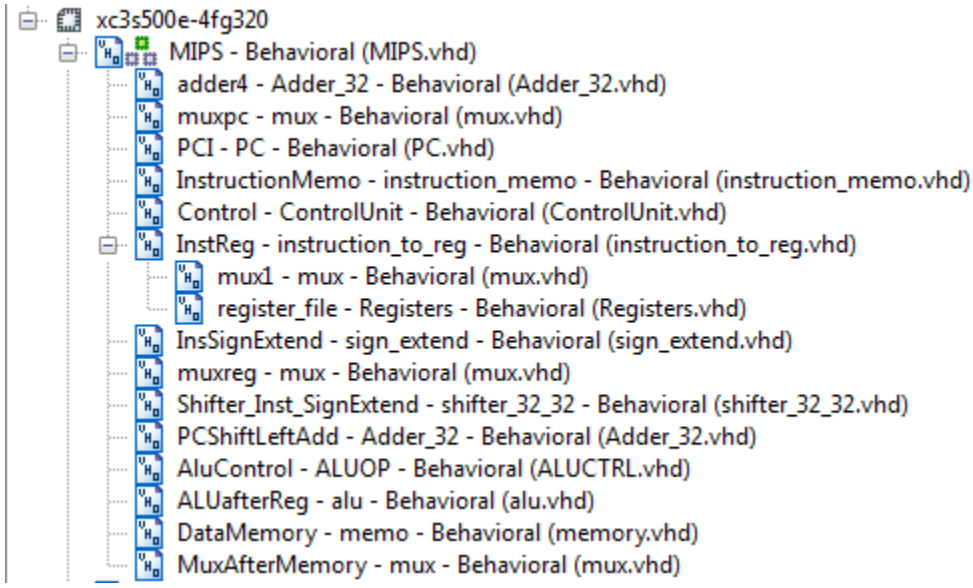
# MIPS Single Cycle Project

## ➢ VHDL Sources / Components:

- Hierarchy:




- Program Counter (PC):

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

entity PC is
    Port ( clk : in  STD_LOGIC;
           inp : in  STD_LOGIC_VECTOR (31 downto 0);
           outp : out  STD_LOGIC_VECTOR (31 downto 0));
end PC;

architecture Behavioral of PC is
signal tmp : std_logic_vector (31 downto 0):= x"00000000";

begin
process(clk,inp,tmp) begin
   if rising_edge(clk) then
      outp <= tmp;
   end if;
   if falling_edge(clk) then
      tmp<=inp;
   end if;
end process;
end Behavioral;
```

PC.vhd

- Adder 32 bits:

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

entity Adder_32 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 Adder_32;

architecture Behavioral of Adder_32 is

begin
sum <= A+B;

end Behavioral;
```

Adder_32.vhd

- Multiplexer (MUX):

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

entity mux is
generic(N: integer :=32);

port(
A:in std_logic_vector(N-1 downto 0);
B:in std_logic_vector(N-1 downto 0);
sel:in std_logic;
Y:out std_logic_vector(N-1 downto 0)
);

end mux;

architecture Behavioral of mux is

begin
process(A,B,sel)
begin
if sel ='0' then
Y<=A;
else
Y<=B;
end if;
end process;
end Behavioral;
```

mux.vhd

- Shifter 32 bits (Shift left by 2)

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

entity shifter_32_32 is
    Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
           B : out  STD_LOGIC_VECTOR (31 downto 0));
end shifter_32_32;

architecture Behavioral of shifter_32_32 is

begin
B(31 downto 2)<=A(29 downto 0);
B(1 downto 0)<="00";

end Behavioral;
```

shifter_32_32.vhd

- Instruction Memory:

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

entity instruction_memo is
    Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
           Instruction : out  STD_LOGIC_VECTOR (31 downto 0));
end instruction_memo;

architecture Behavioral of instruction_memo is

type Memory is array(0 to 23) of STD_LOGIC_VECTOR(7 downto 0);
signal MM:Memory := (
X"00",X"85",X"10",X"20",  -- add $v0, $a0, $a1
X"AC",X"02",X"00",X"08",  -- sw $v0, 8($zero)
X"8C",X"06",X"00",X"08",  -- lw $a2, 8($zero)
X"10",X"46",X"00",X"01",  -- beq $v0, $a2, Good_Processor
X"00",X"46",X"88",X"2A",  -- slt $s1, $v0, $a2
X"00",X"A4",X"88",X"22"); -- Good_Processor: sub $s1, $a1, $a0
begin
   process(Address)
   begin
      Instruction(31 downto 24) <= MM(to_integer(unsigned(Address)));
      Instruction(23 downto 16) <= MM(to_integer(unsigned(Address))+1);
      Instruction(15 downto 8) <= MM(to_integer(unsigned(Address))+2);
      Instruction(7 downto 0) <= MM(to_integer(unsigned(Address))+3);
      end process;
end Behavioral;
```

instruction_memo.vhd

- Instruction to Registers

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

entity instruction_to_reg is
    Port ( Instruction : in  STD_LOGIC_VECTOR (31 downto 0);
           RegDst : in  STD_LOGIC;
           RegWrite : in  STD_LOGIC;
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
           ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
           clk : in  STD_LOGIC);
end instruction_to_reg;

architecture Behavioral of instruction_to_reg is
--componenets
component Registers is
    Port ( ReadReg1 : in  STD_LOGIC_VECTOR (4 downto 0);
           ReadReg2 : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteReg : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
           ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
           RegWrite : in  STD_LOGIC;
           CLK : in  STD_LOGIC);
end component;

component mux is
generic(N: integer :=32);
```

```
component mux is
generic(N: integer :=32);

port( A:in std_logic_vector(N-1 downto 0);
      B:in std_logic_vector(N-1 downto 0);
      sel:in std_logic;
      Y:out std_logic_vector(N-1 downto 0)
);
end component;

--signals
signal out_mux : STD_LOGIC_VECTOR(4 downto 0);

begin

--port mapping
mux1 : mux generic map(N =>5) port map(Instruction(20 downto 16),Instruction(15 downto 11), RegDst , out_mux);

register_file : Registers port map ( Instruction (25 downto 21),Instruction (20 downto 16),out_mux,WriteData,ReadData1,ReadData2,RegWrite,clk);

end Behavioral;
```

- Registers File

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

entity Registers is
    Port ( ReadReg1 : in  STD_LOGIC_VECTOR (4 downto 0);
           ReadReg2 : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteReg : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
           ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
           RegWrite : in  STD_LOGIC;
           CLK : in  STD_LOGIC);
end Registers;

architecture Behavioral of Registers is
type regArray is array (0 to 31) of STD_LOGIC_VECTOR(31 downto 0);
signal regfile: regArray :=(
X"00000000", X"00000000", X"00000000", X"00000000",
X"00000005", X"00000007", X"00000000", X"00000000",
X"00000000", X"00000000", X"00000000", X"00000000",
X"00000000", X"00000000", X"00000000", X"00000000",
X"00000000", X"00000000", X"00000000", X"00000000",
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
ReadData1 <= regfile (TO_INTEGER (UNSIGNED (ReadReg1)));
ReadData2 <= regfile (TO_INTEGER (UNSIGNED (ReadReg2)));
PROCESS (WriteData, RegWrite, CLK)
BEGIN

IF RegWrite = '1' AND RISING_EDGE(CLK) THEN
regfile (TO_INTEGER (UNSIGNED(WriteReg))) <= WriteData;
END IF;
END PROCESS;
end Behavioral;
```

Registers.vhd

- Control Unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ControlUnit is
    Port ( Op : in  STD_LOGIC_VECTOR (5 downto 0);
           RegDst : out  STD_LOGIC;
           AluSrc : out  STD_LOGIC;
           MemtoReg : out  STD_LOGIC;
           RegWrite : out  STD_LOGIC;
           MemRead : out  STD_LOGIC;
           MemWrite : out  STD_LOGIC;
           Branch : out  STD_LOGIC;
           ALUOp1 : out  STD_LOGIC;
           ALUOp0 : out  STD_LOGIC);
end ControlUnit;

architecture Behavioral of ControlUnit is

begin

process(Op)
begin
if Op ="000000" then --R-format
         RegDst<='1';
         AluSrc<='0';
         MemtoReg <='0';
         RegWrite <='1';
         MemRead <='0';
         MemWrite <='0';
         Branch <='0';
         ALUOp1 <='1';
         ALUOp0 <='0';
elsif Op = "100011" then --lw
         RegDst<='0';
         AluSrc <='1';
         MemtoReg<='1';
         RegWrite <='1';
         MemRead <='1';
         MemWrite <='0';
```

```
            Branch <='0';
            ALUOp1 <='0';
            ALUOp0 <='0';
elsif Op = "101011" then --sw
            --RegDst<='0';
            AluSrc <='1';
            --MemtoReg<='1';
            RegWrite <='0';
            MemRead <='0';
            MemWrite <='1';
            Branch <='0';
            ALUOp1 <='0';
            ALUOp0 <='0';
elsif Op = "000100" then --beq
            --RegDst<='0';
            AluSrc <='0';
            --MemtoReg<='1';
            RegWrite <='0';
            MemRead <='0';
            MemWrite <='0';
            Branch <='1';
            ALUOp1 <='0';
            ALUOp0 <='1';
end if;
end process;
end Behavioral;
```

ControlUnit.vhd

- Sign Extend 16 → 32 bits

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

entity sign_extend is
    Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
           B : out  STD_LOGIC_VECTOR (31 downto 0));
end sign_extend;

architecture Behavioral of sign_extend is

begin
B(15 downto 0)<=A;
B(31 downto 16)<= (31 downto 16 => A(15));

end Behavioral;
```

sign_extend.vhd

- ALU Control

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

entity ALUOP is
    Port ( func : in  STD_LOGIC_VECTOR (5 downto 0);
           opcode : in  STD_LOGIC_VECTOR (1 downto 0);
           op : out  STD_LOGIC_VECTOR (3 downto 0));
end ALUOP;

architecture Behavioral of ALUOP is

begin
process(opcode,func)
begin
if opcode ="00" then op<="0010"; -- load ad store
elsif opcode = "01" then op<="0110"; -- branch
elsif opcode = "10" and func = "100000" then op<="0010"; -- add
elsif opcode = "10" and func = "100010" then op<="0110"; -- sub
elsif opcode = "10" and func = "100100" then op<="0000"; -- and
elsif opcode = "10" and func = "100101" then op<="0001"; -- or
elsif opcode = "10" and func = "101010" then op<="0111"; -- set
end if;
end process;

end Behavioral;
```

ALUCTRL.vhd

- ALU

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


entity alu is
    Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
           B : in  STD_LOGIC_VECTOR (31 downto 0);
           ALU_CONTROL : in  STD_LOGIC_VECTOR (3 downto 0);
           ALU_RESULT : out  STD_LOGIC_VECTOR (31 downto 0);
           ZERO : out  STD_LOGIC);
end alu;

architecture Behavioral of alu is
signal resultX: STD_LOGIC_VECTOR (31 downto 0);
begin
process(A,B,ALU_CONTROL)
begin

CASE ALU_CONTROL is
when "0000" => resultX <= A and B;
when "0001" => resultX <= A or B;
when "0010" => resultX <= A + B;
when "0110" => resultX <= A - B;
when "0111" =>
if A< B then resultX<=X"00000001";
else resultX<=X"00000000";
end if;

when "1100" => resultX<= A nor B;
when others => null;
resultX<=X"00000000";
end case;
end process;

zero <= '1' WHEN resultX=X"00000000" else '0';
ALU_RESULT<=resultX;
end Behavioral;
```

alu.vhd

- Data Memory (Main Memory)

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.NUMERIC_STD.ALL;
23
24  entity memo is
25      Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
26             WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
27             MemWrite : in  STD_LOGIC;
28             MemRead : in  STD_LOGIC;
29             CLK : in  STD_LOGIC;
30             ReadData : out  STD_LOGIC_VECTOR (31 downto 0));
31  end memo;
32
33  architecture Behavioral of memo is
34  type Memory is array(0 to 35) of STD_LOGIC_VECTOR(7 downto 0);
35  signal MM:Memory := (
36  X"AB",X"CD",X"EF",X"00",
37  X"75",X"74",X"65",X"72",
38  X"20",X"41",X"72",X"63",
39  X"68",X"69",X"74",X"65",
40  X"12",X"34",X"56",X"78",
41  X"7F",X"7F",X"6D",X"6D",
42  X"00",X"00",X"00",X"00",
43  X"78",X"78",X"6A",X"6A",
44  X"00",X"00",X"00",X"01");
45  begin
46  process(MemRead,MemWrite,Address,WriteData,CLK)
47  begin
48
49     if MemRead = '1' and MemWrite = '0' then
50     ReadData(31 downto 24) <= MM(to_integer(unsigned(Address)));
51     ReadData(23 downto 16) <= MM(to_integer(unsigned(Address))+1);
52     ReadData(15 downto 8) <= MM(to_integer(unsigned(Address))+2);
53     ReadData(7 downto 0) <= MM(to_integer(unsigned(Address))+3);
54
55     elsif MemRead ='0' and MemWrite ='1' and rising_edge(CLK) then
56     MM(to_integer(unsigned(Address))) <= WriteData(31 downto 24);
57     MM(to_integer(unsigned(Address))+1) <= WriteData(23 downto 16);
58     MM(to_integer(unsigned(Address))+2) <= WriteData(15 downto 8);
59     MM(to_integer(unsigned(Address))+3) <= WriteData(7 downto 0);
60
61     end if;
62     end process;
63  end Behavioral;
64
65
```

memory.vhd

- MIPS

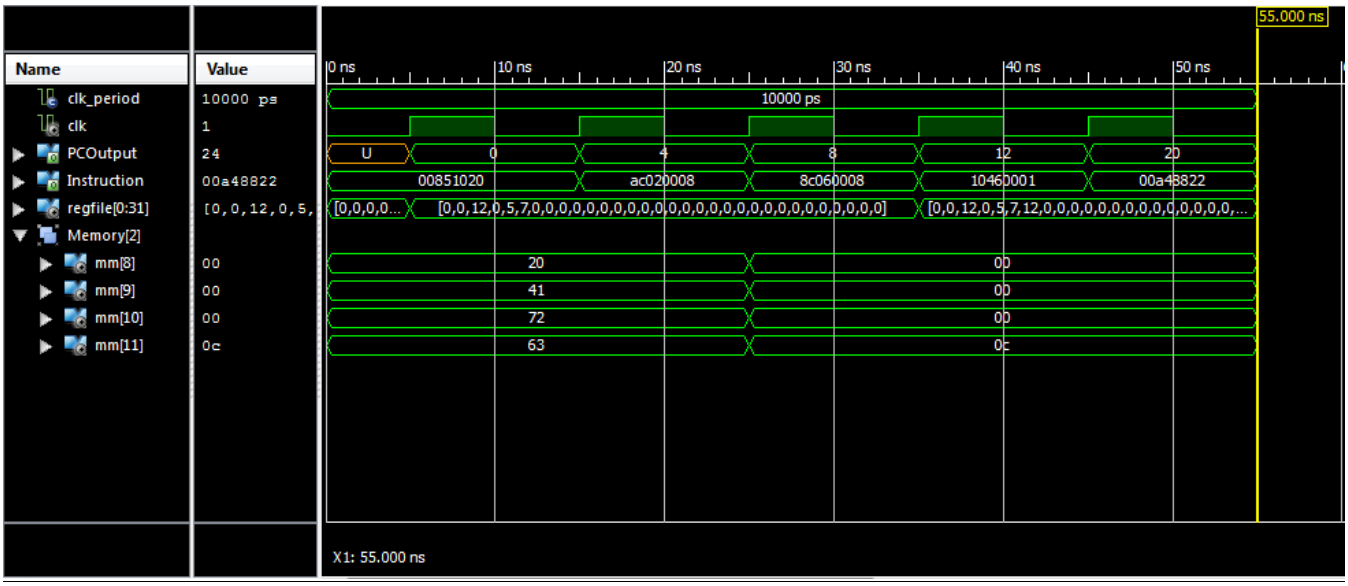
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MIPS is
    Port ( CLK : in  STD_LOGIC);
end MIPS;

architecture Behavioral of MIPS is

--PC
component PC is
    Port ( clk : in  STD_LOGIC;
           inp : in  STD_LOGIC_VECTOR (31 downto 0);
           outp : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

--Instruction_to_Reg
component instruction_to_reg is
    Port ( Instruction : in  STD_LOGIC_VECTOR (31 downto 0);
           RegDst : in  STD_LOGIC;
           RegWrite : in  STD_LOGIC;
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
           ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
           clk : in  STD_LOGIC);
end component;
--Adder32
component  Adder_32 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 component ;

--Control Unit
component  ControlUnit is
    Port ( Op : in  STD_LOGIC_VECTOR (5 downto 0);
           RegDst : out  STD_LOGIC;
           AluSrc : out  STD_LOGIC;
           MemtoReg : out  STD_LOGIC;
           RegWrite : out  STD_LOGIC;
           MemRead : out  STD_LOGIC;
           MemWrite : out  STD_LOGIC;
           Branch : out  STD_LOGIC;
           ALUOp1 : out  STD_LOGIC;
           ALUOp0 : out  STD_LOGIC);
end component ;
```

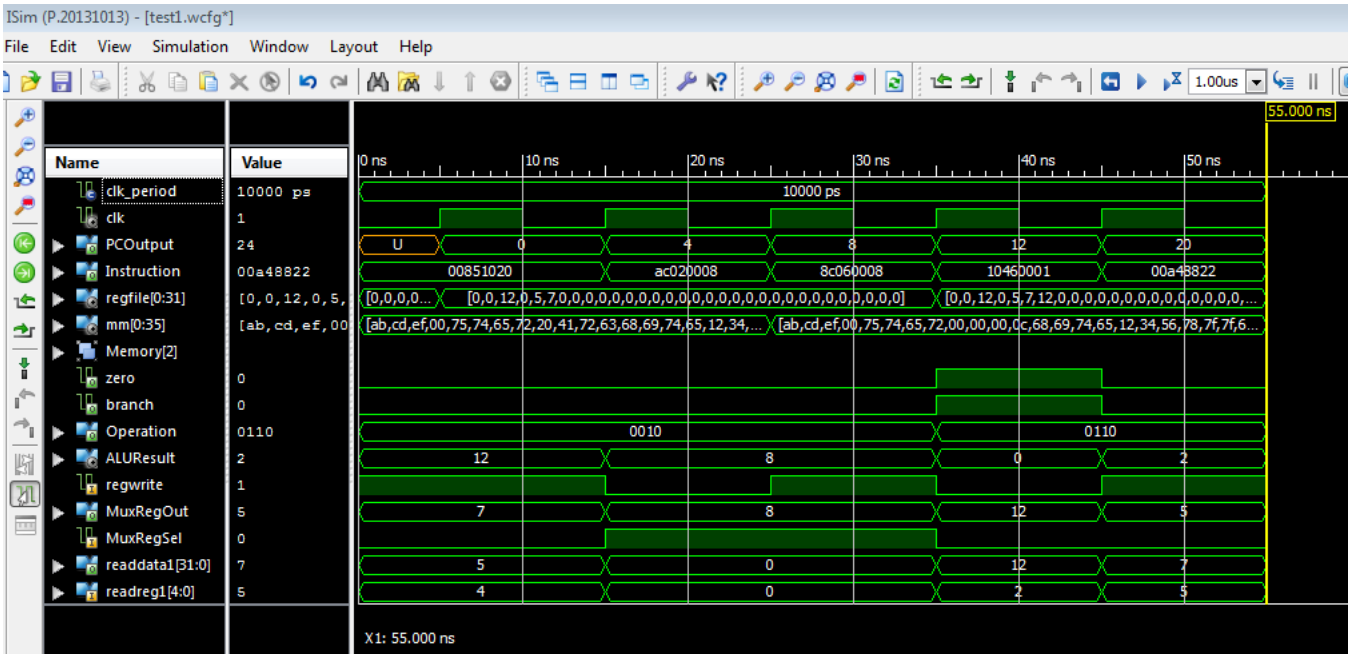
```
--ALU
component alu is
    Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
           B : in  STD_LOGIC_VECTOR (31 downto 0);
           ALU_CONTROL : in  STD_LOGIC_VECTOR (3 downto 0);
           ALU_RESULT : out  STD_LOGIC_VECTOR (31 downto 0);
           ZERO : out  STD_LOGIC);
end component;

-- ALUControl
component  ALUOP is
    Port ( func : in  STD_LOGIC_VECTOR (5 downto 0);
           opcode : in  STD_LOGIC_VECTOR (1 downto 0);
           op : out  STD_LOGIC_VECTOR (3 downto 0));
end component;


-- Instruction memory
component instruction_memo is
    Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
           Instruction : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

--Data Memory
component memo is
    Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           MemWrite : in  STD_LOGIC;
           MemRead : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           ReadData : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

--Shifter 26-28
component  shifter_26_28 is
    Port ( A : in  STD_LOGIC_VECTOR (25 downto 0);
           B : out  STD_LOGIC_VECTOR (27 downto 0));
end component ;

--Shifter 32-32
component shifter_32_32 is
    Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
           B : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

--Sign_extend
component sign_extend is
    Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
           B : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

```

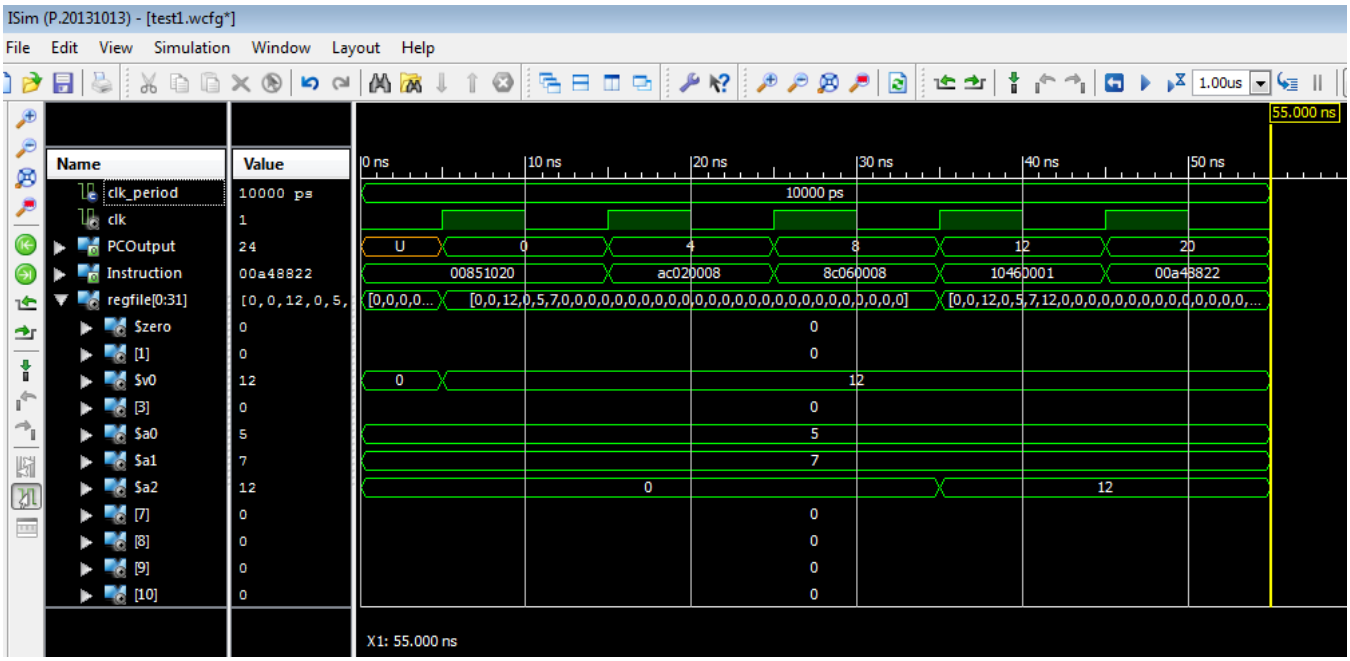
```
--Mux
component mux is
generic(N: integer :=32);

port( A:in std_logic_vector(N-1 downto 0);
      B:in std_logic_vector(N-1 downto 0);
      sel:in std_logic;
      Y:out std_logic_vector(N-1 downto 0)
);
end component;

--Signals Declaration
signal pc4 : STD_LOGIC_VECTOR(31 downto 0) := (others => '0');
signal BranchALUOutput : STD_LOGIC_VECTOR(31 downto 0);
signal pc_address_inp : STD_LOGIC_VECTOR(31 downto 0) := (others => '0');
signal pc_address_outp : STD_LOGIC_VECTOR(31 downto 0) := (others => '0');
signal BranchAndZero : STD_LOGIC ;
signal InstructionOutput : STD_LOGIC_VECTOR(31 downto 0);
signal RegDst,AluSrc,MemtoReg,RegWrite,MemRead,MemWrite,ALUOp1,ALUOp0 : STD_LOGIC;
signal WriteDatatoReg : STD_LOGIC_VECTOR(31 downto 0);
signal ReadData1,ReadData2 : STD_LOGIC_VECTOR (31 downto 0);
signal Ins_SignExtend_Out  : STD_LOGIC_VECTOR (31 downto 0);
signal Mux_Reg_Out  : STD_LOGIC_VECTOR (31 downto 0);
signal Shiftleft2ALU  : STD_LOGIC_VECTOR (31 downto 0);
signal OperationtoALU: STD_LOGIC_VECTOR (3 downto 0);
signal ALUoutput :  STD_LOGIC_VECTOR (1 downto 0);
signal ALUResult: STD_LOGIC_VECTOR (31 downto 0);
signal Zero :STD_LOGIC;
signal Branch :STD_LOGIC;
signal DataMemoryOutput :STD_LOGIC_VECTOR (31 downto 0); --


begin
-- port mapping
adder4 : Adder_32 port map(pc_address_outp,X"00000004",pc4);
muxpc: mux generic map (N=>32) port map (pc4,BranchALUOutput,BranchAndZero,pc_address_inp);
PCI :  PC port map(CLK,pc_address_inp ,pc_address_outp);


InstructionMemo: instruction_memo  port map(pc_address_outp,InstructionOutput);
Control: ControlUnit port map(InstructionOutput(31 downto 26),RegDst,AluSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUoutput(1),ALUoutput(0));
InstReg: instruction_to_reg port map(InstructionOutput,RegDst,RegWrite,WriteDatatoReg,ReadData1,ReadData2,CLK);

InsSignExtend: sign_extend port map(InstructionOutput(15 downto 0) ,Ins_SignExtend_Out);
muxreg: mux generic map (N=>32) port map (ReadData2,Ins_SignExtend_Out,AluSrc,Mux_Reg_Out);
Shifter_Inst_SignExtend : shifter_32_32 port map (Ins_SignExtend_Out,Shiftleft2ALU);
PCShiftLeftAdd : Adder_32 port map(pc4,Shiftleft2ALU,BranchALUOutput);

AluControl : ALUOP port map(InstructionOutput(5 downto 0) ,ALUoutput,OperationtoALU);
ALUafterReg : alu port map(ReadData1,Mux_Reg_Out,OperationtoALU,ALUResult,Zero);
BranchAndZero <= Branch AND Zero;

DataMemory : memo port map(ALUResult,ReadData2,MemWrite,MemRead,CLK,DataMemoryOutput); --
MuxAfterMemory : mux generic map (N=>32) port map (ALUResult,DataMemoryOutput,MemtoReg,WriteDatatoReg); --
end Behavioral;
```

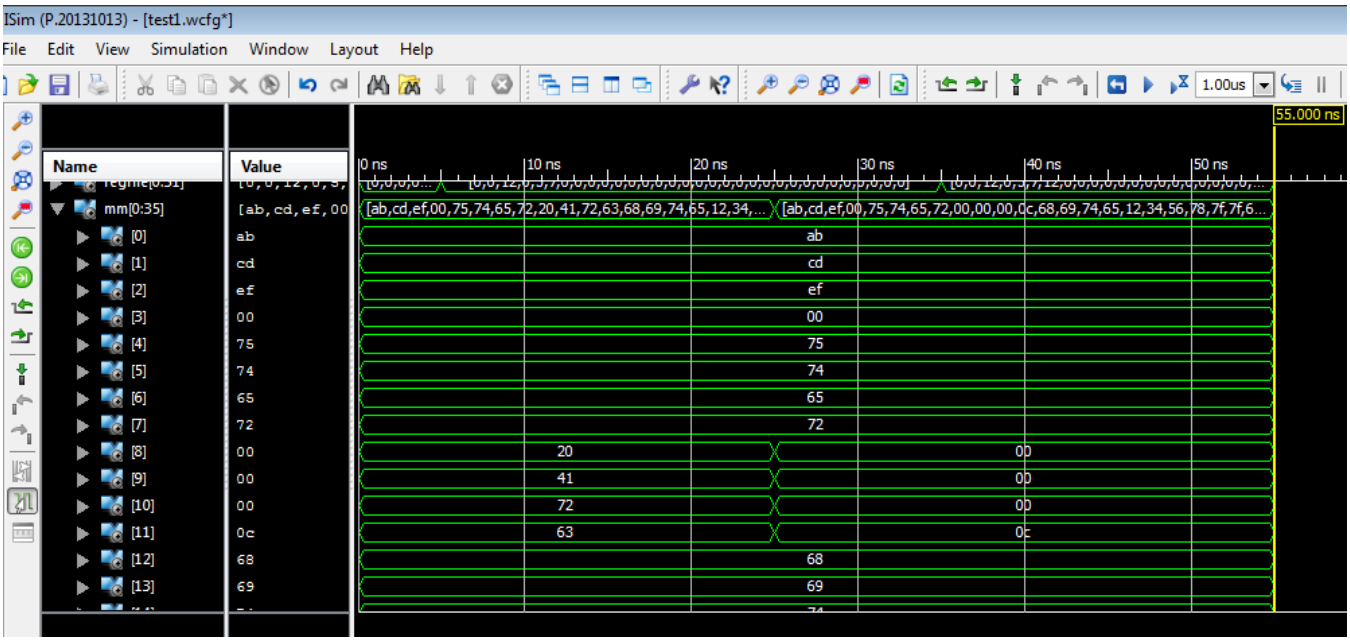
MIPS.vhd

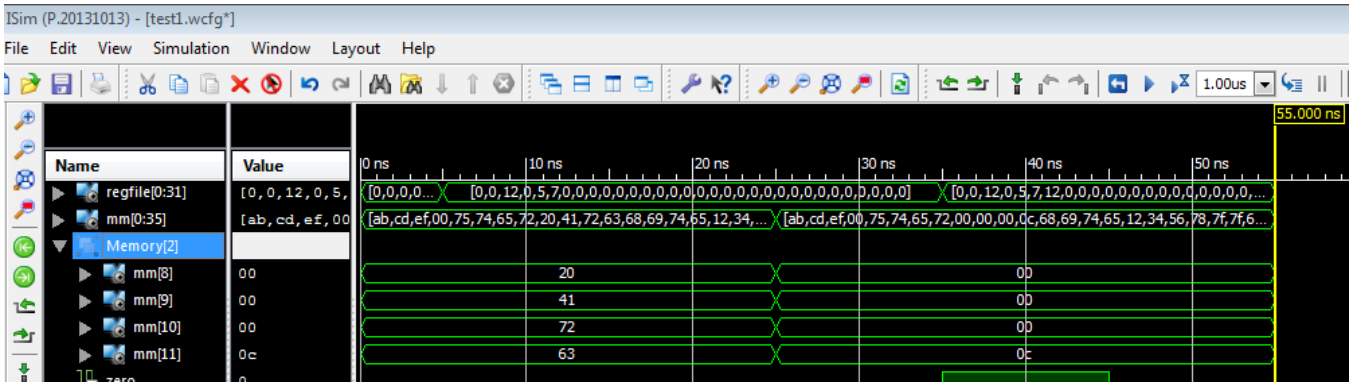
## ➢ Simulation Screenshots




ISim (P.20131013) - [test1.wcfg*]
File Edit View Simulation Window Layout Help
Name
Value
clk_period 10000 ps
clk 1
PCOutput 24
Instruction 00a48822
regfile[0:31] [0,0,12,0,5,
$zero 0
[1] 0
$v0 12
[3] 0
$a0 5
$a1 7
$a2 12
[7] 0
[8] 0
[9] 0
[10] 0
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
55.000 ns
X1: 55.000 ns


ISim (P.20131013) - [test1.wcfg*]
File Edit View Simulation Window Layout Help
Name
Value
mm[0:35] [ab,cd,ef,00
[0] ab
[1] cd
[2] ef
[3] 00
[4] 75
[5] 74
[6] 65
[7] 72
[8] 00
[9] 00
[10] 00
[11] 0c
[12] 68
[13] 69
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
55.000 ns


ISim (P.20131013) - [test1.wcfg*]
File Edit View Simulation Window Layout Help
Name
Value
regfile[0:31] [0,0,12,0,5,
mm[0:35] [ab,cd,ef,00
Memory[2]
mm[8] 00
mm[9] 00
mm[10] 00
mm[11] 0c
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
55.000 ns

- Tested Instructions:

| Assembly Instruction | Object Code (Hex) |
|---|---|
| add $v0, $a0, $a1 | 0x00851020 |
| sw $v0, 8($zero) | 0xAC020008 |
| lw $a2, 8($zero) | 0x8C060008 |
| beq $v0, $a2, Good_Processor | 0x10460001 |
| slt $s1, $v0, $a2 | 0x0046882A |
| Good_Processor: sub $s1, $a1, $a0 | 0x00A48822 |