MIPS

1. Υλοποίηση και έλεγχος μονάδων

1.1.1 ALU

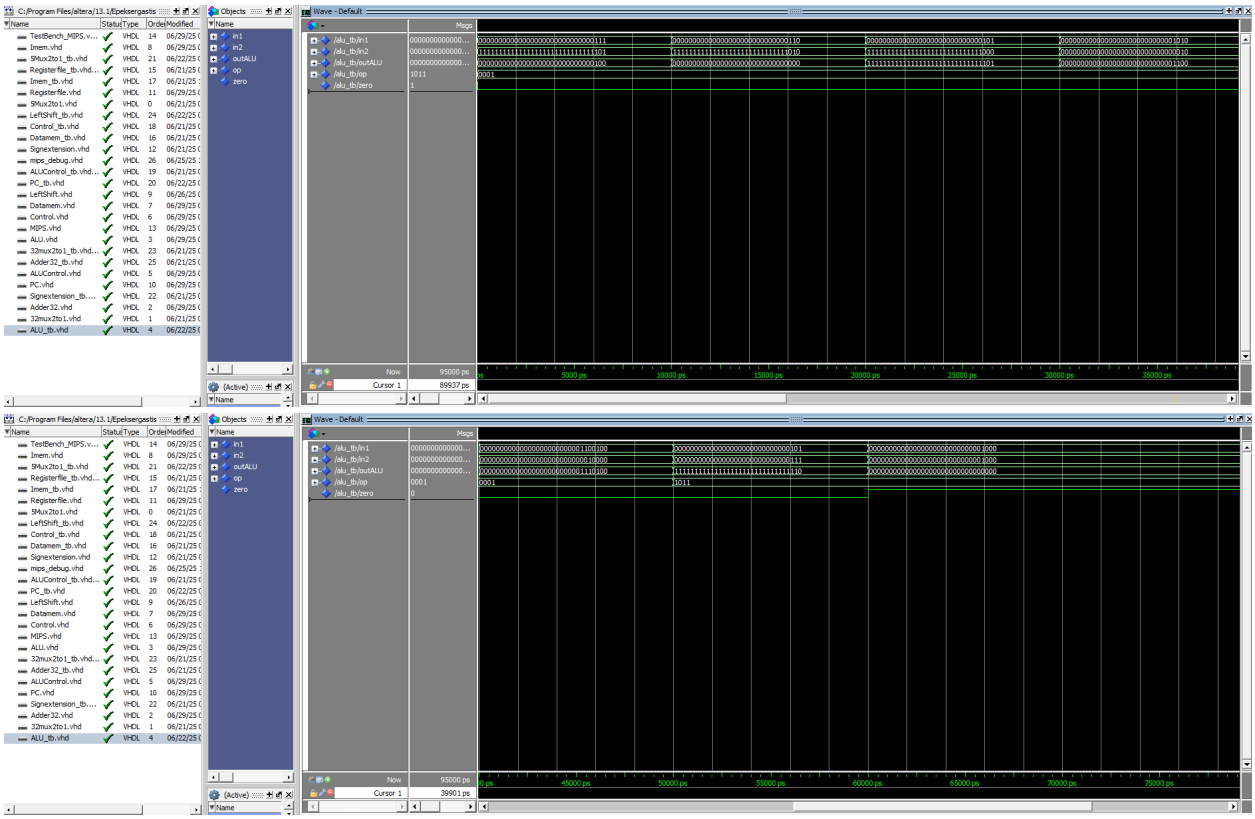
Main :

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

ENTITY alu IS
  port (
    in1    : in  std_logic_vector(31 downto 0);
    in2    : in  std_logic_vector(31 downto 0);
    op     : in  std_logic_vector(3 downto 0);
    outALU : out std_logic_vector(31 downto 0);
    Zero   : out std_logic);
END alu;

ARCHITECTURE ALU_1 of ALU is
  component FullAdder
    port (
      in1, in2 : in  std_logic_vector(31 downto 0);
      carryin  : in  std_logic_vector(0 downto 0);
      sum      : out std_logic_vector(31 downto 0);
      carryout : out std_ulogic);
  end component;
  component Shifter
    port (
      Sin  : in  std_logic_vector(31 downto 0);
      Sout : out std_logic_vector(31 downto 0);
      opS  : in  std_ulogic;
      num  : in  std_logic_vector(4 downto 0));
  end component;
  signal op_shifter: std_logic;
  signal carryout:   std_logic;
  signal out_FA:     std_logic_vector(31 downto 0);
  signal out_SH:     std_logic_vector(31 downto 0);
  signal x:          std_logic_vector(31 downto 0) := (others=>'X');
  signal slt_result: std_logic_vector(31 downto 0);
BEGIN

  op_shifter <=
    '0' when op="0111" else
    '1' when op="1000" else
    'X';

  FA_ALU: FullAdder port map(in1 => in1, in2 => in2, carryin => "0", sum => out_FA, carryout=>carryout);
  SH_ALU: Shifter   port map(Sin => in1, Sout => out_SH, opS => op_shifter, num => in2(4 downto 0));

  slt_result <=
    std_logic_vector(to_unsigned(1,32)) when (to_integer(signed(in1)) < to_integer(signed(in2))) else
    std_logic_vector(to_unsigned(0,32));

  with op select
    outALU <=
      -- sum:
      out_FA when "0001",
```

```
53          -- and:
54          in1 and in2 when "0010",
55          -- or:
56          in1 or in2 when "0011",
57          -- nor:
58          in1 nor in2 when "0100",
59          -- and immediate:
60          in1 and in2 when "0101",
61          -- or immediate:
62          in1 or in2 when "0110",
63          -- shift left:
64          out_SH  when "0111",
65          -- shift right
66          out_SH  when "1000",
67          -- slt
68          slt_result when "1100",
69          -- addi
70          out_FA when "1101",
71
72          std_logic_vector(signed(in1) - signed(in2)) when "1010", -- beq comparison
73          std_logic_vector(signed(in1) - signed(in2)) when "1011", -- bne comparison
74          x when others;
75
76      -- Zero = '1' when ALU result is zero (registers are equal)
77      -- Zero = '0' when ALU result is non-zero (registers are not equal)
78      Zero <=
79        '1' when (in1 = in2) else
80        '0';
81
82  END ALU_1;
```

TestBench

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

entity ALU_tb is
end ALU_tb;

architecture Behavioral of ALU_tb is

    component ALU
        port (
            in1    : in  std_logic_vector(31 downto 0);
            in2    : in  std_logic_vector(31 downto 0);
            op     : in  std_logic_vector(3 downto 0);
            outALU : out std_logic_vector(31 downto 0);
            zero   : out std_logic
        );
    end component;

    signal in1, in2, outALU : std_logic_vector(31 downto 0);
    signal op               : std_logic_vector(3 downto 0);
    signal zero             : std_logic;

begin

    uut: ALU
        port map (
            in1    => in1,
            in2    => in2,
            op     => op,
            outALU => outALU,
            zero   => zero
        );

    process
    begin
        -- 1. add: 7 + (-3) = 4
        in1 <= std_logic_vector(to_signed(7, 32));
        in2 <= std_logic_vector(to_signed(-3, 32));
        op  <= "0001";  -- add
        wait for 10 ns;

        -- 2. add: 6 + (-6) = 0 (zero flag should be '1' if op=1010)
        in1 <= std_logic_vector(to_signed(6, 32));
        in2 <= std_logic_vector(to_signed(-6, 32));
        op  <= "0001";  -- add
        wait for 10 ns;

        -- 3. sub: 5 - 8 = -3 ? simulate as 5 + (-8)
        in1 <= std_logic_vector(to_signed(5, 32));
        in2 <= std_logic_vector(to_signed(-8, 32));
        op  <= "0001";  -- add/sub using negative in2
        wait for 10 ns;

        -- 4. addi: same as add
        in1 <= std_logic_vector(to_signed(10, 32));
        in2 <= std_logic_vector(to_signed(2, 32));  -- immediate
        op  <= "0001";  -- add
        wait for 10 ns;

        -- 5. lw/sw: base + offset
        in1 <= std_logic_vector(to_signed(100, 32));  -- base
        in2 <= std_logic_vector(to_signed(16, 32));   -- offset
        op  <= "0001";  -- add
        wait for 10 ns;

        -- 6. bne: test when in1 /= in2
        in1 <= std_logic_vector(to_signed(5, 32));
        in2 <= std_logic_vector(to_signed(7, 32));
        op  <= "1011";  -- bne
        wait for 10 ns;

        -- 7. bne: test when in1 = in2 (zero = '0')
        in1 <= std_logic_vector(to_signed(8, 32));
        in2 <= std_logic_vector(to_signed(8, 32));
        op  <= "1011";  -- bne
        wait for 10 ns;

        wait;
    end process;

end Behavioral;
```



Σχολια-Παρατηρησεις :

Η alu ειναι υπευθηνη για την εκτελεση των αριθμητικων και λογικων πραξεων οπως add,sub,and,or,slt,shift,not,addi,επιστρεφει ενα outalu και εχουμε ενα flag για zero που χρησιμοποιηται στην beq και bne

Η alu ενσωματωνει την fulladder ,shifter ,slt και επιλεγει μια απο αυτες με op.

Βλεπουμε οτι πραγματι για το 1.1.1 της ασκησης , το outALU ειναι η σωστη για

7 + (-3) , outALU = 000...100 = 4

6 + (-6) , outALU = 000...000 = 0

5 – 8 , outALU = 111..101 = -3

Στο testbench χρησιμοποιω και αλλες λειτουργικοτητες του mips οπως την add,sub,addi,lw,sw,bne

1.1.2 Register file

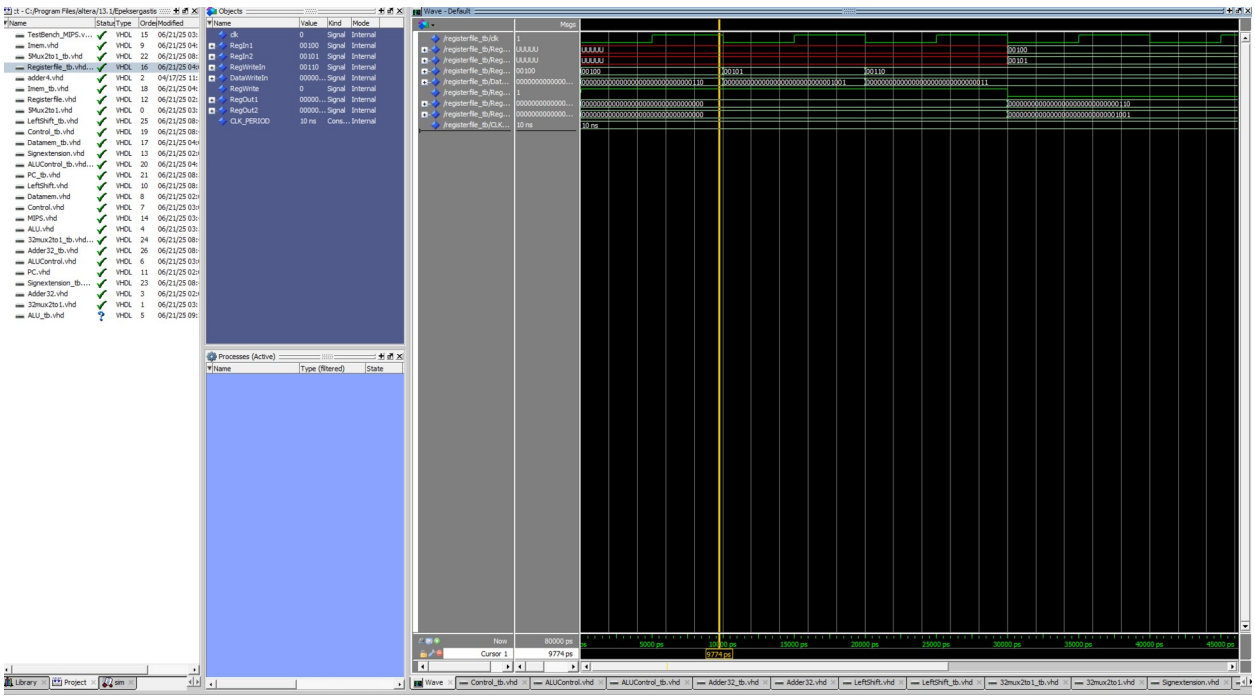
Main:

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

ENTITY Registers is
  port (
    CLK         : in  std_logic;
    RegIn1      : in  std_logic_vector(4 downto 0);
    RegIn2      : in  std_logic_vector(4 downto 0);
    RegWriteIn  : in  std_logic_vector(4 downto 0);
    DataWriteIn : in  std_logic_vector(31 downto 0);
    RegWrite    : in  std_logic;
    RegOut1     : out std_logic_vector(31 downto 0);
    RegOut2     : out std_logic_vector(31 downto 0));
END Registers;

ARCHITECTURE Registers_1 of Registers is
  type registers is array (0 to 31) of std_logic_vector(31 downto 0);
  signal regs: registers := (others=> (others => '0'));
BEGIN

  RegOut1 <= regs(to_integer(unsigned(RegIn1))) when to_integer(unsigned(RegIn1)) /= 0 else (others => '0');
  RegOut2 <= regs(to_integer(unsigned(RegIn2))) when to_integer(unsigned(RegIn2)) /= 0 else (others => '0');


  write_process: process(CLK)
  begin
    if rising_edge(CLK) then
      -- Write to register if RegWrite is enabled and not writing to register 0
      if RegWrite = '1' and to_integer(unsigned(RegWriteIn)) /= 0 then
        regs(to_integer(unsigned(RegWriteIn))) <= DataWriteIn;
      end if;
    end if;
  end process;

END Registers_1;
```

TestBench :

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

entity Registerfile_tb is
end Registerfile_tb;

architecture Behavioral of Registerfile_tb is

    component Registers
        port (
            clk         : in  std_logic;
            RegIn1      : in  std_logic_vector(4 downto 0);
            RegIn2      : in  std_logic_vector(4 downto 0);
            RegWriteIn  : in  std_logic_vector(4 downto 0);
            DataWriteIn : in  std_logic_vector(31 downto 0);
            RegWrite    : in  std_logic;
            RegOut1     : out std_logic_vector(31 downto 0);
            RegOut2     : out std_logic_vector(31 downto 0)
        );
    end component;

    -- Signals
    signal clk         : std_logic := '0';
    signal RegIn1      : std_logic_vector(4 downto 0);
    signal RegIn2      : std_logic_vector(4 downto 0);
    signal RegWriteIn  : std_logic_vector(4 downto 0);
    signal DataWriteIn : std_logic_vector(31 downto 0);
    signal RegWrite    : std_logic;
    signal RegOut1     : std_logic_vector(31 downto 0);
    signal RegOut2     : std_logic_vector(31 downto 0);

    constant CLK_PERIOD : time := 10 ns;

begin

    uut: Registers
        port map (
            clk         => clk,
            RegIn1      => RegIn1,
            RegIn2      => RegIn2,
            RegWriteIn  => RegWriteIn,
            DataWriteIn => DataWriteIn,
            RegWrite    => RegWrite,
            RegOut1     => RegOut1,
            RegOut2     => RegOut2
        );

    -- Clock process
    clk_process : process
    begin
        while now < 200 ns loop
```

```
53              clk <= '0';
54              wait for CLK_PERIOD / 2;
55              clk <= '1';
56              wait for CLK_PERIOD / 2;
57          end loop;
58          wait;
59      end process;
60
61      -- Test process
62      stim_proc: process
63      begin
64          -- Εγγραφή του 6 στον καταχωρητή $4
65          RegWrite <= '1';
66          RegWriteIn <= "00100"; -- $4
67          DataWriteIn <= std_logic_vector(to_signed(6, 32));
68          wait for CLK_PERIOD;
69
70          -- Εγγραφή του 9 στον καταχωρητή $5
71          RegWriteIn <= "00101"; -- $5
72          DataWriteIn <= std_logic_vector(to_signed(9, 32));
73          wait for CLK_PERIOD;
74
75          -- Εγγραφή του 3 στον καταχωρητή $6
76          RegWriteIn <= "00110"; -- $6
77          DataWriteIn <= std_logic_vector(to_signed(3, 32));
78          wait for CLK_PERIOD;
79
80          -- Ανάγνωση των καταχωρητών $4 και $5
81          RegWrite <= '0';
82          RegIn1 <= "00100"; -- $4
83          RegIn2 <= "00101"; -- $5
84          wait for CLK_PERIOD;
85
86
87          wait for 10 ns;
88
89
90          wait;
91      end process;
92
93  end Behavioral;
94
95
```



Σχολια-Παρατηρησεις :

Σκοπος του register file ειναι να αποτελει ενα πινακα καταχωρητων του mips και περιλαμβανει 32 bit καταχωρητες. Κανω εγγραφη μονο αν το regwrite = 1 και στην ακμη ρολογιου. Επισης απαγορευω την εγγραφη στον $0 γιατι εκει η τιμη ειναι παντα 0,αν δωθει τιμη για τον καταχωρητη αγνοείται.

Test :

Εγγραφή του 6 στον καταχωρητή $4

Εγγραφή του 9 στον καταχωρητή $5

Εγγραφή του 3 στον καταχωρητή $6

Ανάγνωση των καταχωρητών $4 και $5

Στην εικονα βλεπω οτι

RegWritein εχει το $4 = 0100, $5 =00101, $6 = 00110

DataWritten in εχει 00110 = 6, 1001 = 9 ,0011 = 3 και ολα αυτα με regwrite = 1 δηλαδη enabled

Οταν γινει το regwrite 0 τοτε θελουμε να διαβασουμε απο regin1 = $4 και regin2= $5, και μας δινει το νουμερο 6 και 9 στο RegOut 1 και RegOut2 αντιστοιχα.

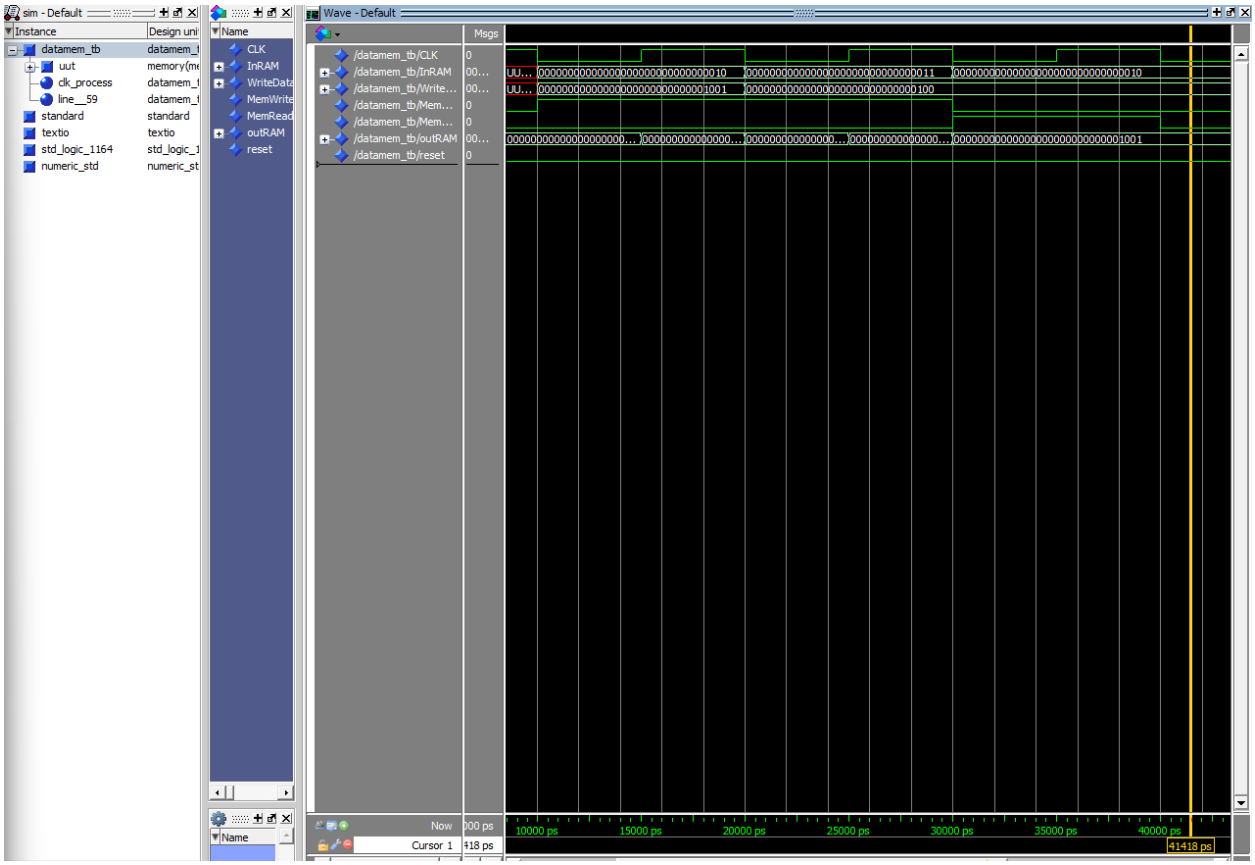
1.1.3. data memory

Main:

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

entity Memory is
    port (
        CLK       : in  std_logic;
        inRAM     : in  std_logic_vector(31 downto 0);
        WriteData : in  std_logic_vector(31 downto 0);
        MemWrite  : in  std_logic;
        MemRead   : in  std_logic;
        outRAM    : out std_logic_vector(31 downto 0);
        reset     : in  std_logic
    );
end Memory;

architecture Memory_1 of Memory is
    -- RAM: 1024 words of 32-bit memory
    type ram_type is array (0 to 1023) of std_logic_vector(31 downto 0);
    signal ram : ram_type := (others => (others => '0'));
    signal Address : integer := 0;

    --  Function to check if a vector has only defined bits
    function is_defined(vec : std_logic_vector) return boolean is
    begin
        for i in vec'range loop
            if vec(i) = 'U' or vec(i) = 'X' then
                return false;
            end if;
        end loop;
        return true;
    end function;

begin
    -- Address decoding with proper bounds checking
    process(inRAM)
        variable temp_addr : integer;
    begin
        if is_defined(inRAM) then

            temp_addr := to_integer(signed(inRAM));

            -- Bounds checking: ensure address is within valid range
            if temp_addr >= 0 and temp_addr < 1024 then
                Address <= temp_addr;
            else
                -- For negative or out-of-bounds addresses, use address 0
                Address <= 0;
            end if;
        else
            Address <= 0;
        end if;
    end process;

    --  Clocked write to memory
    process(CLK)
    begin
        if rising_edge(CLK) then
            if (MemWrite = '1' and reset = '0') then
                ram(Address) <= WriteData;
            end if;
        end if;
    end process;

    --  Read from memory
    with reset select
        outRAM <= ram(Address) when '0',
                  (others => '0') when others;

end Memory_1;
```

TestBench

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

entity Datamem_tb is
end Datamem_tb;

architecture Behavioral of Datamem_tb is

    component Memory
        port (
            CLK        : in  std_logic;
            InRAM          : in std_logic_vector(31 downto 0);
            WriteData      : in std_logic_vector(31 downto 0);
            MemWrite       : in std_logic;
            MemRead        : in std_logic;
            outRAM         : out std_logic_vector(31 downto 0);
            reset          : in std_logic
        );
    end component;

    -- Signals
    signal CLK          : std_logic := '0';
    signal InRAM        : std_logic_vector(31 downto 0);
    signal WriteData    : std_logic_vector(31 downto 0);
    signal MemWrite     : std_logic := '0';
    signal MemRead      : std_logic := '0';
    signal outRAM       : std_logic_vector(31 downto 0);
    signal reset        : std_logic := '0';

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Memory
        port map (
            CLK            => CLK,
            InRAM          => InRAM,
            WriteData      => WriteData,
            MemWrite       => MemWrite,
            MemRead        => MemRead,
            outRAM         => outRAM,
            reset          => reset
        );

    -- Clock process
    clk_process : process
    begin
        while now < 200 ns loop
            CLK <= '0';
            wait for 5 ns;
            CLK <= '1';
            wait for 5 ns;
        end loop;
        wait;
    end process;

    -- Stimulus process
    process
    begin
        -- Reset low
        reset <= '0';
        wait for 10 ns;

        -- Write 9 to memory address 2
        InRAM <= std_logic_vector(to_unsigned(2, 32));
        WriteData <= std_logic_vector(to_signed(9, 32));
        MemWrite <= '1';
        wait for 10 ns;
        MemWrite <= '0';

        -- Write 4 to memory address 3
        InRAM <= std_logic_vector(to_unsigned(3, 32));
        WriteData <= std_logic_vector(to_signed(4, 32));
        MemWrite <= '1';
        wait for 10 ns;
        MemWrite <= '0';

        -- Read memory address 2
        InRAM <= std_logic_vector(to_unsigned(2, 32));
        MemRead <= '1';
        wait for 10 ns;
        MemRead <= '0';

        wait;
    end process;

end Behavioral;
```



Σχολια- Παρατηρησεις :

Σκοπος της Data memory ειναι να αποθηκευσει δεδομενα του προγραμματος για προσωρινες τιμες . Γραφει μονο οταν το MemWrite ειναι Enabled δηλαδη = 1

Test :

Εγγραφή του 9 στη θέση μνήμης 2

Εγγραφή του 4 στη θέση μνήμης 3

Ανάγνωση της θέσης μνήμης 2

Στην εικονα βλεπουμε οτι

InRam = 2 (00..10) γραφει το WriteData = 00...1001 (9)

InRam = 4 (00..100) γραφει το WriteData = 00...0011 (3)

Και επειτα το Memread = 1 και διαβαζουμε το InRam = 2 και μας δινει το

OutRam = 00..1001 (9)

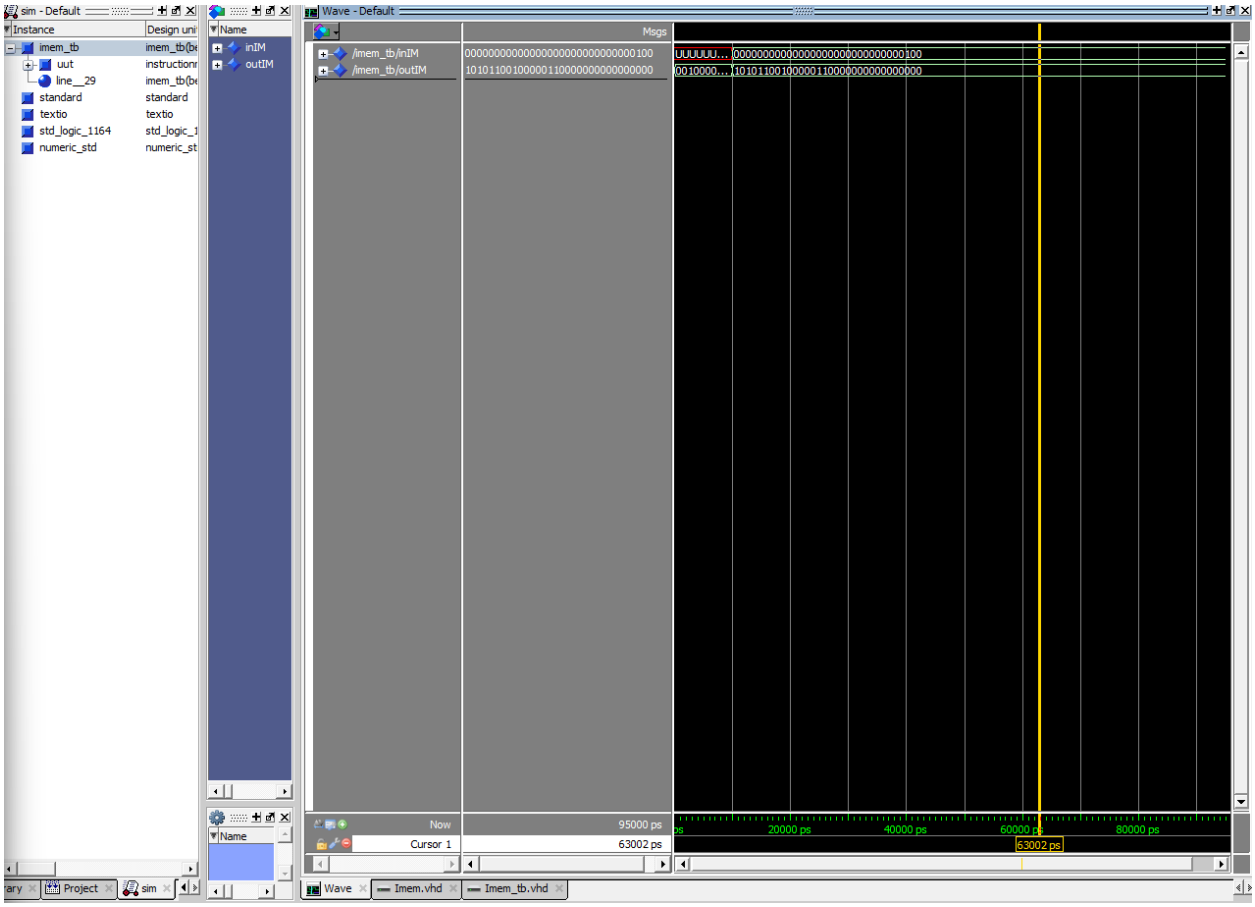
1.1.4. Instruction memory

Main :

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

ENTITY InstructionMemory IS
  port (
    inIM  : in  std_logic_vector(31 downto 0);
    outIM : out std_logic_vector(31 downto 0)
  );
END InstructionMemory;

ARCHITECTURE InstructionMemory_1 of InstructionMemory IS

  type mem_type is array (natural range <>) of std_logic_vector(31 downto 0);
  signal mem : mem_type(0 to 255) := (
    0 => x"20000000",  -- addi $0, $0, 0
    1 => x"20040000",  -- addi $4, $0, 0
    2 => x"20030001",  -- addi $3, $0, 1
    3 => x"20050003",  -- addi $5, $0, 3
    4 => x"AC830000",  -- sw $3, 0($4)
    5 => x"20630001",  -- addi $3, $3, 1
    6 => x"20840001",  -- addi $4, $4, 1
    7 => x"20A5FFFF",  -- addi $5, $5, -1
    8 => x"14A0FFFB",  -- bne $5, $0, L1 (-5)
    others => x"00000000"
  );

  signal FullInstruction : std_logic_vector(31 downto 0);

BEGIN

  process(inIM)
    variable addr : integer;
  begin
    addr := to_integer(unsigned(inIM));

    if (addr >= 0 and addr <= 255) then
      FullInstruction <= mem(addr);
    else
      FullInstruction <= x"00000000";
      assert false
        report "Instruction fetch out of bounds: PC = " & integer'image(addr)
        severity note;
    end if;
  end process;

  outIM <= FullInstruction;

END InstructionMemory_1;

```

TestBench :

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

entity Imem_tb is
end Imem_tb;

architecture Behavioral of Imem_tb is

    component InstructionMemory
        port (
            inIM  : in  std_logic_vector(31 downto 0);
            outIM : out std_logic_vector(31 downto 0)
        );
    end component;

    signal inIM  : std_logic_vector(31 downto 0);
    signal outIM : std_logic_vector(31 downto 0);

begin

    uut: InstructionMemory
        port map (
            inIM  => inIM,
            outIM => outIM
        );

    process
    begin

        wait for 10 ns;

        inIM <= std_logic_vector(to_unsigned(4, 32));
        wait for 10 ns;


        wait;
    end process;

end Behavioral;

```



Σχολια - Παρατηρησεις :

Το instruction memory αποθηκευει τις εντολες που θα εκτελεσει το προγραμμα

Test :

Εγγραφή του Κώδικα 1 στη μνήμη εντολών

(ξεκινώντας από τη θέση 0)

Ανάγνωση της θέσης μνήμης 4.

Στην εικονα βλεπουμε :

Οτι το InIM παει στην διευθυνση 4 ,αυτη που θελουμε να διαβασουμε και επειτα το outim 10101100100000110000000000000000 που ειναι το AC830000 σε δεκαεξαδικο

Εβαλα το report για debugging γιατι στην αρχη ειχα προβληματα με το PC

1.1.5. Control

Main:

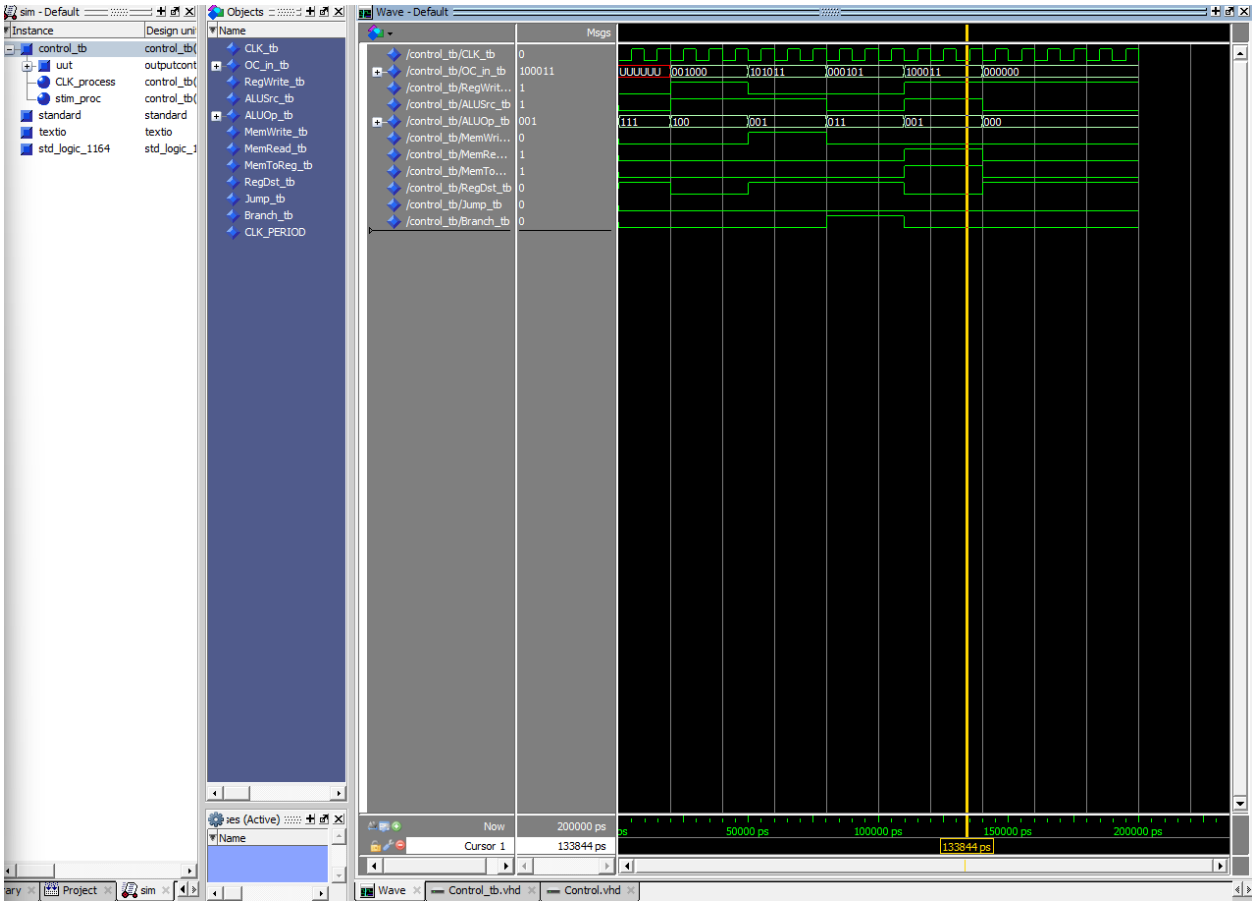
```
library ieee;
use ieee.std_logic_1164.all;

ENTITY OutputControl is
  port (
    CLK      : in std_logic;
    OC_in    : in  std_logic_vector(5 downto 0);
    RegWrite : out std_logic := '0';
    ALUSrc   : out std_logic;
    ALUOp    : out std_logic_vector(2 downto 0);
    MemWrite : out std_logic;
    MemRead  : out std_logic;
    RegDst   : out std_logic;
    MemToReg : out std_logic;
    Jump     : out std_logic;
    Branch   : out std_logic);
END OutputControl;

ARCHITECTURE OutputControl_1 of OutputControl is
BEGIN

  -- Removed the CLK dependency in RegWrite to prevent iteration error loop i was getting
  with OC_in select
    RegWrite <=
      '1' when "100011",--0x23 load by memory
      '1' when "000000",--0x00 all arithmetic functions
      '1' when "001000",--0x08 addi
      '0' when others;

  with OC_in select
    ALUSrc <=
      '1' when "100011", --0x23 load word
      '1' when "101011", --0x2B store word
      '1' when "001000", --0x08 addi
      '0' when others;

  with OC_in select
    ALUOp <=
      "000" when "000000", --Arith operation
      "001" when "100011", --RAM operations (load)
      "001" when "101011", --RAM operations (store)
      "010" when "000100", --0x04 beq
      "011" when "000101", --0x05 bne
      "100" when "001000", --0x08 addi
      "111" when others;

  with OC_in select
    MemWrite <=
      '1' when "101011", --only store
      '0' when others;

  with OC_in select
```

```
    MemRead <=
        '1' when "100011", --only read
        '0' when others;

  with OC_in select
    MemToReg <=
      '1' when "100011", --only read
      '0' when others;

  with OC_in select
    RegDst <=
      '0' when "100011", --0x23 load word
      '0' when "001000", --0x08 addi
      '1' when others;

  with OC_in select
    Jump <=
      '1' when "000010", --0x02 jump to address
      '0' when others;

  with OC_in select
    Branch <=
      '1' when "000100", --0x04 beq
      '1' when "000101", --0x05 bne
      '0' when others;

END OutputControl_1;
```

TestBench :

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

entity Control_tb is
end Control_tb;

architecture Control_tb_arch of Control_tb is

    component OutputControl
        port (
            CLK       : in  std_logic;
            OC_in     : in  std_logic_vector(5 downto 0);
            RegWrite  : out std_logic;
            ALUSrc    : out std_logic;
            ALUOp     : out std_logic_vector(2 downto 0);
            MemWrite  : out std_logic;
            MemRead   : out std_logic;
            MemToReg  : out std_logic;
            RegDst    : out std_logic;
            Jump      : out std_logic;
            Branch    : out std_logic
        );
    end component;

    -- Signal
    signal CLK_tb       : std_logic := '0';
    signal OC_in_tb     : std_logic_vector(5 downto 0);
    signal RegWrite_tb  : std_logic;
    signal ALUSrc_tb    : std_logic;
    signal ALUOp_tb     : std_logic_vector(2 downto 0);
    signal MemWrite_tb  : std_logic;
    signal MemRead_tb   : std_logic;
    signal MemToReg_tb  : std_logic;
    signal RegDst_tb    : std_logic;
    signal Jump_tb      : std_logic;
    signal Branch_tb    : std_logic;

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

begin

    uut: OutputControl
        port map (
            CLK       => CLK_tb,
            OC_in     => OC_in_tb,
            RegWrite  => RegWrite_tb,
            ALUSrc    => ALUSrc_tb,
            ALUOp     => ALUOp_tb,
            MemWrite  => MemWrite_tb,
            MemRead   => MemRead_tb,
            MemToReg  => MemToReg tb,
```

```
            MemtoReg  => MemtoReg_tb,
            RegDst    => RegDst_tb,
            Jump      => Jump_tb,
            Branch    => Branch_tb
        );

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


    stim_proc: process
    begin
        -- Wait for initial setup
        wait for 20 ns;

        -- Test 1: addi $1, $0, 4 (opcode = 001000)
        OC_in_tb <= "001000";
        wait for 30 ns;

        -- Test 2: sw $6, 0($4) (opcode = 101011)
        OC_in_tb <= "101011";
        wait for 30 ns;

        -- Test 3: bne $5,$0,L1 (opcode = 000101)
        OC_in_tb <= "000101";
        wait for 30 ns;

        -- Additional tests for completeness
        -- Test lw instruction (opcode = 100011)
        OC_in_tb <= "100011";
        wait for 30 ns;

        -- Test R-type instruction (opcode = 000000)
        OC_in_tb <= "000000";
        wait for 30 ns;

        wait;
    end process;

end Control_tb_arch;
```



Σχολια-Παρατηρσεις :

Το control Unit παιρνει τις εντολες και τις αποκωδικοποιει και δινει τα σωστα control signals σε διαφορα μερη του επεξεργαστη mips

Test :

Είσοδος των ακόλουθων εντολών και έλεγχος των εξόδων

addi $1, $0, 4

sw $6, 0($4)

bne $5,$0,L1

Εχουμε ενα clock ανα 10 ns

1.addi $1, $0, 4

Εχουμε

OC_in = 001000

regwrite = 1 για να γραφει στον register

Alusrc = 1 χρησιμοποιει immediate value , aluop = 111 και regDest = 0

2. sw $6, 0($4)

Oc_in = 101011

MemWrite = 1 δηλαδη γραφει στην μνημη

Alusrc = 1

AluOP = 001 (ram operations)

3. bne $5,$0,L1

Oc_in = 000101

Branch = 1(γιατι εχουμε branch εντολη)

ALUOP = 011

Επισης εβαλα και

Lw για να δω την λειτουργια και r instruction

Οποτε δινει : oc_in = 100011

Regwrite = 1 θα γραψει απο μνημη σε register

Memread = 1 αρα θελουμε να διαβασει απο μνημη

Memtoreg = 1 και απο μνημη σε reg

Alusrc = 1

AluOP=001

Επιπλεον : R-Type instruction

Oc_in = 000000

Regwrite = 1(γραφει αποτελεσμα σε register)

Aluop=000 (arithmetic operation)

Regdst = 1

Το simulation δειχνει οτι το control λειτουργει σωστα

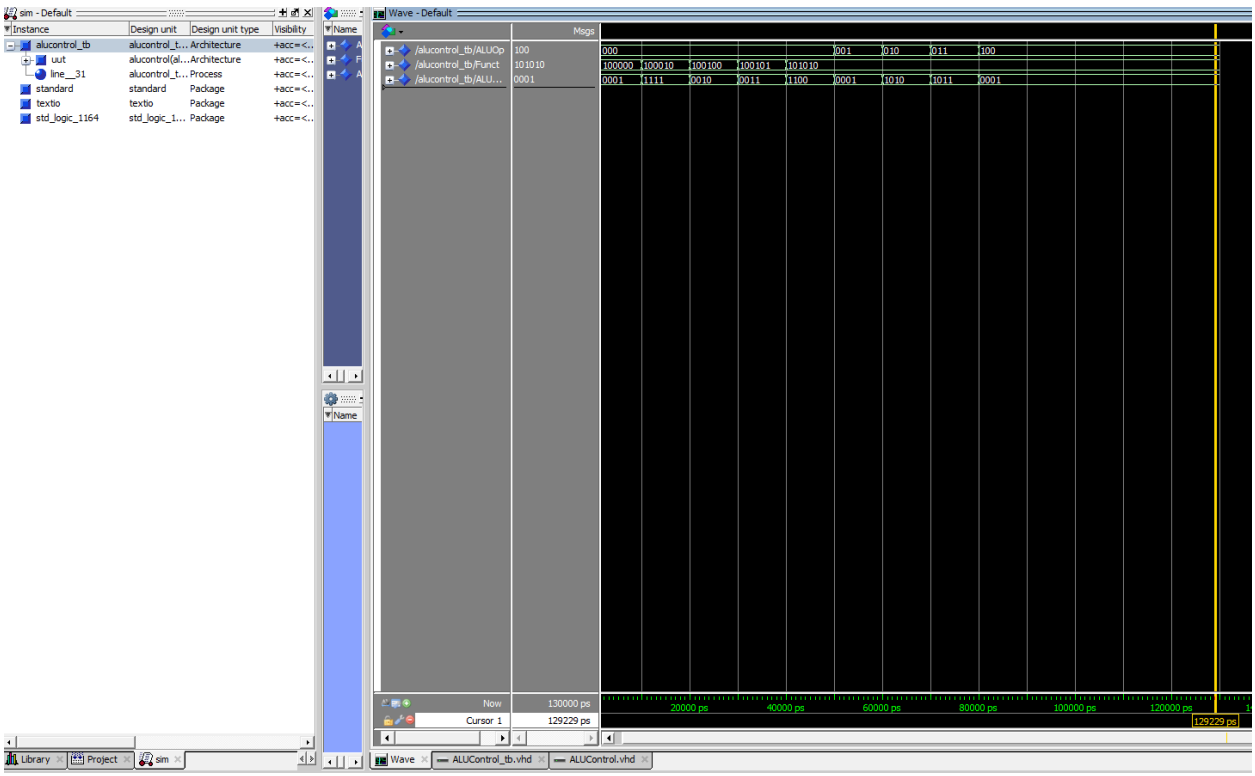
1.1.6. Control ALU

Main

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

entity ALUControl is port (
    ALUOp        : in  std_logic_vector(2 downto 0);
    Funct        : in  std_logic_vector(5 downto 0);
    ALUCont_out  : out std_logic_vector(3 downto 0));
end ALUControl;

architecture ALUControl_1 of ALUControl is
    signal regALUControl_Funct: std_logic_vector(3 downto 0) :=
        (others=>'0');
    signal regALUControl_op: std_logic_vector(3 downto 0) :=
        (others=>'0');

begin
    -- Funct is defined only for R type
    with Funct select
        regALUControl_Funct <=
            "0001" when "100000", --0x20
            "0010" when "100100", --0x24
            "0011" when "100101", --0x25
            "0100" when "100111", --0x27
            "0101" when "001100", --0x0C
            "0110" when "001101", --0x0D
            "0111" when "000000", --0x00
            "1000" when "000010", --0x02
            "1100" when "101010", --0x2A slt
            "1111" when others;

    with ALUOp select
        regALUControl_op <=
            "0001" when "001",    -- load: add base and relative address
            "1010" when "010",    -- beq
            "1011" when "011",    -- bne
            "0001" when "100",    -- addi
            "1111" when others;

    with ALUOp select
        ALUCont_out <=
            regALUControl_Funct when "000",
            regALUControl_op when others;

end ALUControl_1;

```

TestBench

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

entity ALUControl_tb is
end ALUControl_tb;

architecture Behavioral of ALUControl_tb is

    component ALUControl
        port (
            ALUOp        : in  std_logic_vector(2 downto 0);
            Funct        : in  std_logic_vector(5 downto 0);
            ALUCont_out  : out std_logic_vector(3 downto 0)
        );
    end component;

    signal ALUOp       : std_logic_vector(2 downto 0);
    signal Funct       : std_logic_vector(5 downto 0);
    signal ALUCont_out : std_logic_vector(3 downto 0);

begin

    uut: ALUControl
        port map (
            ALUOp        => ALUOp,
            Funct        => Funct,
            ALUCont_out  => ALUCont_out
        );

    process
    begin
        -- R-type: add => ALUOp = "000", Funct = "100000" (0x20) => ALUCont = "0001"
        ALUOp <= "000";
        Funct <= "100000";  -- add
        wait for 10 ns;

        -- R-type: sub => Funct = "100010"
        Funct <= "100010";
        wait for 10 ns;

        -- R-type: and => Funct = "100100" => "0010"
        Funct <= "100100";
        wait for 10 ns;

        -- R-type: or => Funct = "100101" => "0011"
        Funct <= "100101";
        wait for 10 ns;

        -- R-type: slt => Funct = "101010" => "1100"
        Funct <= "101010";
        wait for 10 ns;

        -- I-type: lw/sw => ALUOp = "001" => should output "0001" (add)
        ALUOp <= "001";
        wait for 10 ns;

        -- I-type: beq => ALUOp = "010" => "1010"
        ALUOp <= "010";
        wait for 10 ns;

        -- I-type: bne => ALUOp = "011" => "1011"
        ALUOp <= "011";
        wait for 10 ns;

        -- I-type: addi => ALUOp = "100" => "0001"
        ALUOp <= "100";
        wait for 10 ns;

        wait;
    end process;

end Behavioral;
```



Σχολια-Παρατηρησεις:

Το ALU control unit λειτουργει με το control unit , παιρνει τα signal του και τα μετατρεπει σε συγκεκριμενους κωδικους του για την alu και ετσι η alu κανει την ακριβη πραξην που πρεπει να εκτελεσει .

Τεστ:

Είσοδος των ακόλουθων συνδυασμών και έλεγχος των εξόδων

Funct – ALUop

100000 – 10

100010 – 10

111111 – 00

111111 – 01

Η funct ειναι μονο για τα r-type instructions

Και για τα i type χρησιμοποιει την aluop

Τι βλεπουμε στο simulation :

1. ALUOp = "000", Funct = "100000" -> ALUCont_out = "0001" (ADD)

2. ALUOp = "000", Funct = "100010" -> ALUCont_out = "1111" (SUB)

3. ALUOp = "000", Funct = "100100" -> ALUCont_out = "0010"(AND)

4. ALUOp = "000", Funct = "100101" -> ALUCont_out = "0011"(OR)

5. ALUOp = "000", Funct = "101010" -> ALUCont_out = "1100"(SLT)

6. ALUOp = "001" -> ALUCont_out = "0001" (LW)

7. ALUOp = "010" -> ALUCont_out = "1010" (BRANCH EQUAL)

8. ALUOp = "011" -> ALUCont_out = "1011" (BRANCH NOT EQUAL)

9. ALUOp = "100" -> ALUCont_out = "0001" (ADDI)

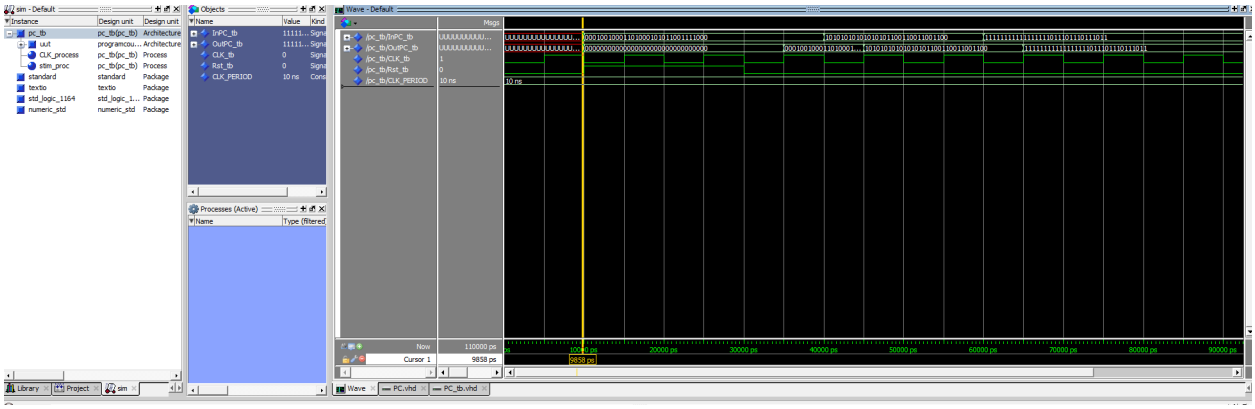
1.1.7. Program counter

Main

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY ProgramCounter IS
6       port (
7           inPC  : in  std_logic_vector(31 downto 0);
8           outPC : out std_logic_vector(31 downto 0);
9           CLK   : in  std_logic;
10          Rst   : in  std_logic
11      );
12  END ProgramCounter;
13
14  ARCHITECTURE ProgramCounter_1 of ProgramCounter IS
15  BEGIN
16      reg: process(CLK,Rst)
17      begin
18          if Rst = '1' then
19              outPC <= (others => '0');  -- clear PC on reset
20          elsif rising_edge(CLK) then
21              outPC <= inPC;             -- update PC on clock edge
22          end if;
23      end process;
24  END ProgramCounter_1;
25
```

TestBench

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

entity PC_tb is
end PC_tb;

architecture PC_tb of PC_tb is
    -- Component declaration
    component ProgramCounter
        port (
            InPC  : in  std_logic_vector(31 downto 0);
            OutPC : out std_logic_vector(31 downto 0);
            CLK   : in  std_logic;
            Rst   : in  std_logic
        );
    end component;

    -- Signal declarations
    signal InPC_tb  : std_logic_vector(31 downto 0);
    signal OutPC_tb : std_logic_vector(31 downto 0);
    signal CLK_tb   : std_logic := '0';
    signal Rst_tb   : std_logic := '0';

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

begin

    uut: ProgramCounter
        port map (
            InPC  => InPC_tb,
            OutPC => OutPC_tb,
            CLK   => CLK_tb,
            Rst   => Rst_tb
        );

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


    stim_proc: process
    begin

        wait for 10 ns;

        -- Test reset functionality
        Rst_tb <= '1';
        InPC_tb <= x"12345678";  -- Should be ignored during reset
        wait for 20 ns;
        Rst_tb <= '0';
        wait for 10 ns;

        -- Test 1: Write and read 0xAAAACCCC
        InPC_tb <= x"AAAACCCC";
        wait for 20 ns;  -- Wait for clock edge

        -- Test 2: Write and read 0xFFFFBBBB
        InPC_tb <= x"FFFFBBBB";
        wait for 20 ns;  -- Wait for clock edge


        wait;
    end process;

end PC_tb;
```



Σχολια - Παρατηρησεις :

Ο PC κραταει τη διευθυνση της τρεχουσας εντολής

Τεστ:

Εγγραφή και ανάγνωση των ακόλουθων δεδομένων:

0xAAAA CCCC

0xFFFF BBBB

Αρχηκα κανουμε ενα reset , Rst= 1 και το outpc δινει 0 ανεξαρτητως το pcin

Εχω βαλει για τεστ pc in 0x"12345678"

Μετα θετω rst=0 και βαζω το pcin = 0xAAAA CCCC

Το outpc δινει το περιεχομενο αυτο σε 32 bit

Επειτα βαζω inpc = 0xFFFF BBBB

Outpc = 0xFFFF BBBB.

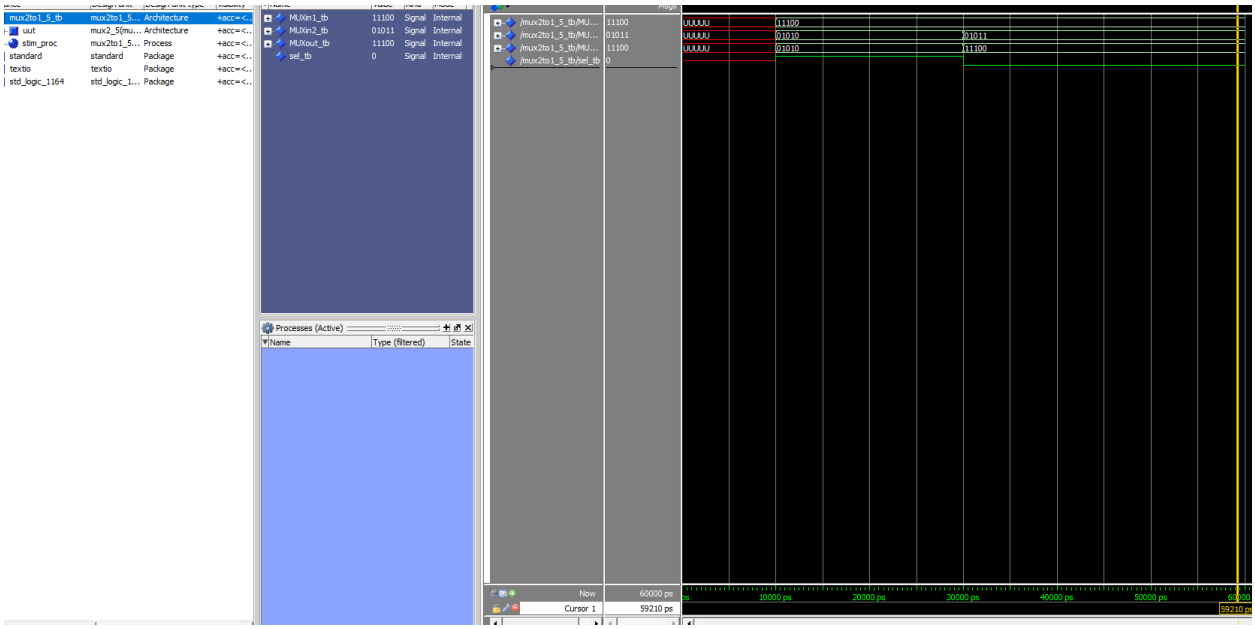
1.1.8. 5mux 2-1

Main

```vhdl
Library IEEE;
use IEEE.std_logic_1164.all;
ENTITY MUX2_5 IS PORT (
    MUXin1 : in std_logic_vector(4 downto 0);
    MUXin2 : in std_logic_vector(4 downto 0);
    MUXout : out std_logic_vector(4 downto 0);
    sel : in std_logic);
END MUX2_5;
ARCHITECTURE MUX2_5_1 of MUX2_5 is
BEGIN
    MUXout <= MUXin1 when sel='0' else
              MUXin2 when sel='1';
END;

```

TestBench

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

entity Mux2to1_5_tb is
end Mux2to1_5_tb;

architecture Mux2to1_5_tb of Mux2to1_5_tb is
    -- Component declaration
    component MUX2_5
        port (
            MUXin1 : in  std_logic_vector(4 downto 0);
            MUXin2 : in  std_logic_vector(4 downto 0);
            MUXout : out std_logic_vector(4 downto 0);
            sel    : in  std_logic
        );
    end component;

    -- Signal declarations
    signal MUXin1_tb : std_logic_vector(4 downto 0);
    signal MUXin2_tb : std_logic_vector(4 downto 0);
    signal MUXout_tb : std_logic_vector(4 downto 0);
    signal sel_tb    : std_logic;

begin

    uut: MUX2_5
        port map (
            MUXin1 => MUXin1_tb,
            MUXin2 => MUXin2_tb,
            MUXout => MUXout_tb,
            sel    => sel_tb
        );

    -- Stimulus process
    stim_proc: process
    begin
        -- Wait for initial setup
        wait for 10 ns;


        -- ΕΙΣΟΔΟΣ0 - ΕΙΣΟΔΟΣ1 - ΕΠΙΛΟΓΗ
        -- 0x1C - 0x0A - 1
        -- 0x1C - 0x0B - 0

        -- Test 1: Input0=0x1C, Input1=0x0A, Select=1 (should output 0x0A)
        MUXin1_tb <= "11100";  -- 0x1C
        MUXin2_tb <= "01010";  -- 0x0A
        sel_tb <= '1';
        wait for 20 ns;

        -- Test 2: Input0=0x1C, Input1=0x0B, Select=0 (should output 0x1C)
        MUXin1_tb <= "11100";  -- 0x1C
        MUXin2_tb <= "01011";  -- 0x0B
        sel_tb <= '0';
        wait for 20 ns;



        wait;
    end process;

end Mux2to1_5_tb;

```



Σχολια - Παρατηρησεις :

Αυτό είναι ένας 2 to 1 multiplexe **r** που λειτουργεί με 5-bit vectors - χρησιμοποιείται για την επιλογή register destinations στον mips επεξεργαστή.

Τεστ:

ΕΙΣΟΔΟΣ0 – ΕΙΣΟΔΟΣ1 – ΕΠΙΛΟΓΗ

0x1C – 0x0A – 1

0x1C – 0x0B – 0

Οταν sel = 0 τοτε muxout = muxin1

Οταν sel = 1 τοτε muxout = muxin2

1. 0x1C – 0x0A – 1
   Βλεπουμε muxin1 = 11100 (0x1C)
   Muxin2 = 01010(0x0A)
   Sel = 1
   Muxout = 01010(muxin2)
2. 0x1C – 0x0B – 0
   muxin1 = 11100 (0x1C)
   Muxin2 = 01011(0x0B)

Sel = 0
Muxout = 11100(muxin1)

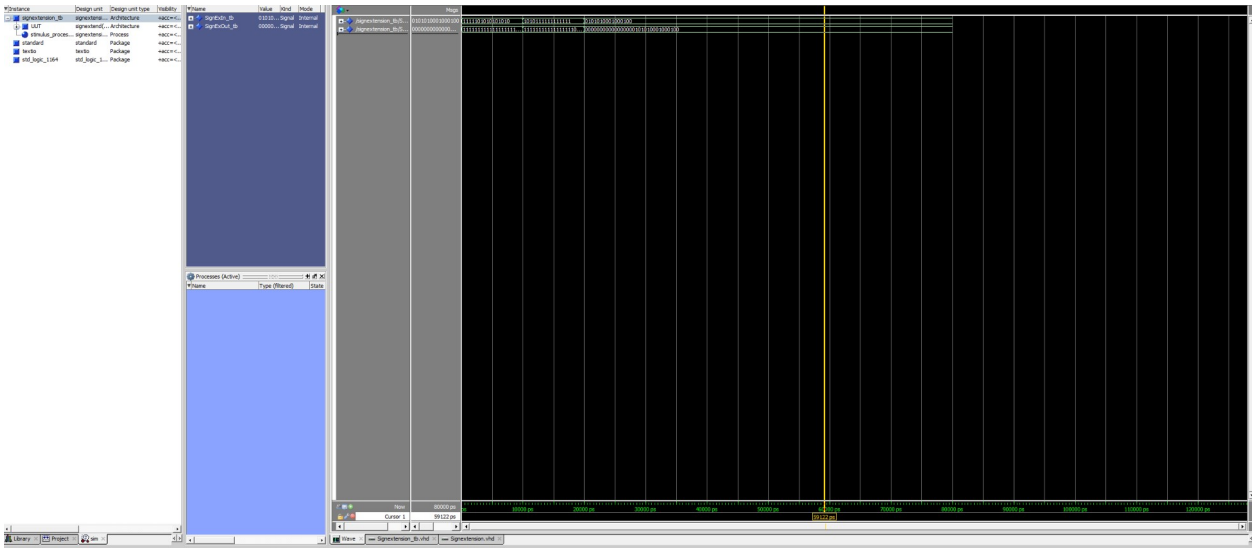
1.1.9. Sign Extenstion

Main

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4   ENTITY SignExtend IS port (
5       SignExIn  : in  std_logic_vector(15 downto 0);
6       SignExOut : out std_logic_vector(31 downto 0)
7   );
8   END SignExtend;
9
10  ARCHITECTURE SignExtend_1 of SignExtend IS
11      signal ones  : std_logic_vector(15 downto 0) := (others=>'1');
12      signal zeros : std_logic_vector(15 downto 0) := (others=>'0');
13  BEGIN
14
15
16
17  SignExOut <= ones & SignExIn  when SignExIn(15) = '1' else
18               zeros & SignExIn when SignExIn(15) = '0';
19
20  END SignExtend_1;
```

TestBench

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Signextension_tb is
end Signextension_tb;

architecture Signextension_tb of Signextension_tb is
    -- Component declaration
    component SignExtend
        port (
            SignExIn  : in  std_logic_vector(15 downto 0);
            SignExOut : out std_logic_vector(31 downto 0)
        );
    end component;

    -- Test signals
    signal SignExIn_tb  : std_logic_vector(15 downto 0);
    signal SignExOut_tb : std_logic_vector(31 downto 0);

begin
    -- Component instantiation
    UUT: SignExtend
        port map (
            SignExIn  => SignExIn_tb,
            SignExOut => SignExOut_tb
        );

    -- Test process
    stimulus_process: process
    begin
        -- Test case 1: 0xFAAA (negative number)
        SignExIn_tb <= x"FAAA";
        wait for 10 ns;

        -- Test case 2: 0xAFFF (negative number)
        SignExIn_tb <= x"AFFF";
        wait for 10 ns;

        -- Test case 3: 0x5444 (positive number)
        SignExIn_tb <= x"5444";
        wait for 10 ns;

        -- End simulation
        wait;
    end process;

end Signextension_tb;
```



Σχολια - Παρατηρησεις :

Αυτο το unit παιρνει ενα 16 bit signed και το κανει 32 bit signed number

Signexin = 16bit

Signexout = 32 bit

Επισης :

Αν msb (most significant bit)= 1 τοτε negative number

Αλλιως positive

Τεστ :

Είσοδος των ακόλουθων συνδυασμών και έλεγχος εξόδων

0xFAAA

0xAFFF

0x5444

Βλεπουμε στο simulation οτι κανανε extend σε

SignOut = 0xFFFFAAAA (negative number extended)

SignOut = 0xFFFFAFFF (negative number extended)

SignOut = 0x00005444 (positive number extended)

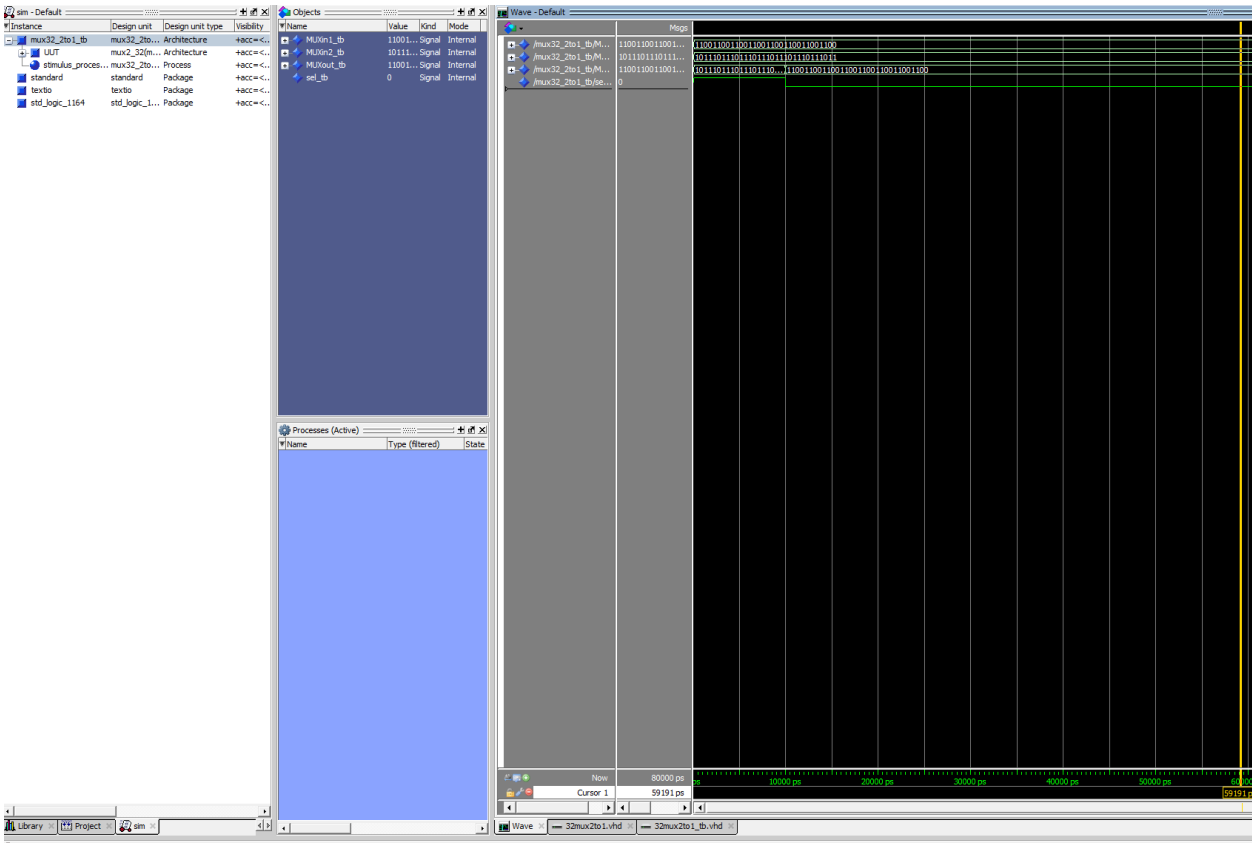
1.1.10. 32mux 2-1

Main

```
1
2   Library IEEE;
3   use IEEE.std_logic_1164.all;
4
5   ENTITY MUX2_32 is PORT (
6     MUXin1 : in std_logic_vector(31 downto 0);
7     MUXin2 : in std_logic_vector(31 downto 0);
8     MUXout : out std_logic_vector(31 downto 0);
9     sel : in std_logic );
10
11  END MUX2_32;
12
13  ARCHITECTURE MUX2_32_1 of MUX2_32 is
14  BEGIN
15    MUXout <= MUXin1 when sel='0' else
16              MUXin2 when sel='1';
17  END MUX2_32_1;
```

TestBench

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux32_2to1_tb is
end Mux32_2to1_tb;

architecture testbench of Mux32_2to1_tb is
    -- Component declaration
    component MUX2_32
        port (
            MUXin1 : in std_logic_vector(31 downto 0);
            MUXin2 : in std_logic_vector(31 downto 0);
            MUXout : out std_logic_vector(31 downto 0);
            sel : in std_logic
        );
    end component;

    -- Test signals
    signal MUXin1_tb : std_logic_vector(31 downto 0);
    signal MUXin2_tb : std_logic_vector(31 downto 0);
    signal MUXout_tb : std_logic_vector(31 downto 0);
    signal sel_tb : std_logic;

begin
    -- Component instantiation
    UUT: MUX2_32
        port map (
            MUXin1 => MUXin1_tb,
            MUXin2 => MUXin2_tb,
            MUXout => MUXout_tb,
            sel => sel_tb
        );

    -- Test process
    stimulus_process: process
    begin
        -- Setup test inputs
        MUXin1_tb <= x"CCCCCCCC";  -- ΕΙΣΟΔΟΣ0
        MUXin2_tb <= x"BBBBBBBB";  -- ΕΙΣΟΔΟΣ1

        -- Test case 1: sel = 1 (should select input2)
        sel_tb <= '1';
        wait for 10 ns;

        -- Test case 2: sel = 0 (should select input1)
        sel_tb <= '0';
        wait for 10 ns;

        -- End simulation
        wait;
    end process;

end testbench;

```



Σχολια - Παρατηρησεις :

Δουλευει σαν τον 5mux 2 to 1 απλα χρησιμοποιει 32 bit inputs

Muxin1 οταν sel = 0

Muxin2 οταν sel =1

Test :

Είσοδος των ακόλουθων συνδυασμών και έλεγχος εξόδων

ΕΙΣΟΔΟΣ0 – ΕΙΣΟΔΟΣ1 – ΕΠΙΛΟΓΗ

0xCCCCCCCC – 0xBBBBBBBB – 1

0xCCCCCCCC – 0xBBBBBBBB – 0

Βλεπουμε :

Για sel = 1

Mux out = muxin2 = 100110... = 0xBBBBBBBB

Για sel = 0

Muxout = muxin1 = 11001.. = 0xCCCCCCCC

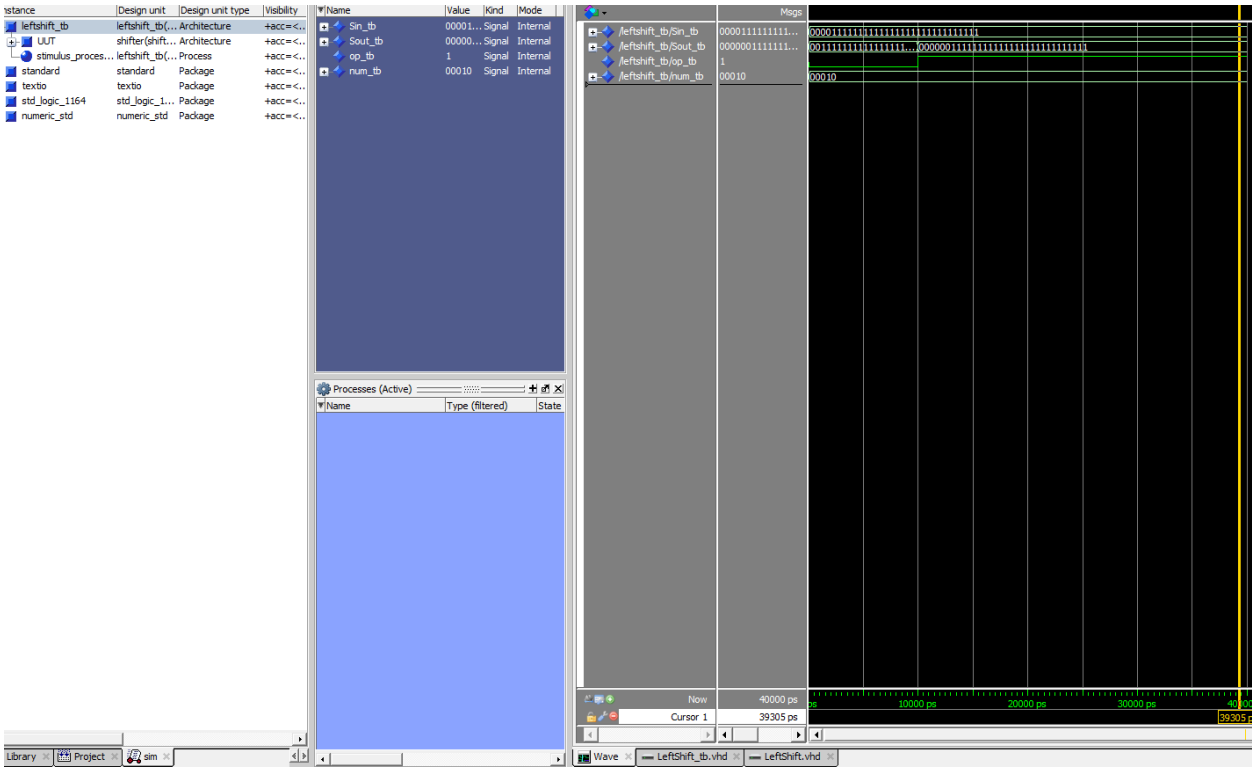
1.1.11. Left Shift

Main

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY Shifter is
6     port (
7       Sin  : in  std_logic_vector(31 downto 0);
8       Sout : out std_logic_vector(31 downto 0);
9       opS  : in  std_ulogic;
10      num  : in  std_logic_vector(4 downto 0));
11  END Shifter;
12
13  ARCHITECTURE Shifter_1 of Shifter is
14
15    signal tmp: unsigned(31 downto 0);
16
17  BEGIN
18
19     tmp <=  to_unsigned(to_integer(signed(Sin)),tmp'length) sll to_integer(signed(num)) when opS='0' else
20        to_unsigned(to_integer(signed(Sin)),tmp'length) srl to_integer(signed(num)) when opS='1';
21
22    Sout <= std_logic_vector(to_signed(to_integer(tmp),Sout'length));
23
24  END Shifter_1;
```

TestBench

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

entity Leftshift_tb is
end Leftshift_tb;

architecture testbench of Leftshift_tb is
    -- Component declaration
    component Shifter
        port (
            Sin  : in  std_logic_vector(31 downto 0);
            Sout : out std_logic_vector(31 downto 0);
            opS   : in  std_ulogic;
            num  : in  std_logic_vector(4 downto 0)
        );
    end component;

    -- Test signals
    signal Sin_tb  : std_logic_vector(31 downto 0);
    signal Sout_tb : std_logic_vector(31 downto 0);
    signal op_tb   : std_ulogic;
    signal num_tb  : std_logic_vector(4 downto 0);

begin
    -- Component instantiation
    UUT: Shifter
        port map (
            Sin  => Sin_tb,
            Sout => Sout_tb,
            opS   => op_tb,
            num  => num_tb
        );

    -- Test process
    stimulus_process: process
    begin
        -- Test case: 0x0FFFFFFF left shift by 2 positions
        Sin_tb <= x"0FFFFFFF";
        op_tb  <= '0';  -- left shift
        num_tb <= "00010";  -- shift by 2
        wait for 10 ns;


        -- Test: shift 0x80000000 right by 1
        Sin_tb <= x"0FFFFFFF";
        op_tb  <= '1';  -- right shift
        num_tb <= "00010";
        wait for 10 ns;

        wait;
    end process;

end testbench;

```



Σχολια-Παρατηρησεις:

Το unit αυτο κανει ολισθηση αριστερα κατα 2 σε 32 bit number

Τεστ :

Είσοδος των ακόλουθων συνδυασμών και έλεγχος εξόδων

0x0FFF FFFF

Βλεπουμε οτι

Sin = 0x0FFFFFFF

Op = 0 (left)

Num = 2 (ολισθηση κατα 2)

Αποτελεσμα : 0x3FFFFFFC

Επισης δοκιμασα και right shift

Sin = 0x0FFFFFFF

Op = 1 (right)

Num = 2 (ολισθηση κατα 2)

Αποτελεσμα : 0x03FFFFFF

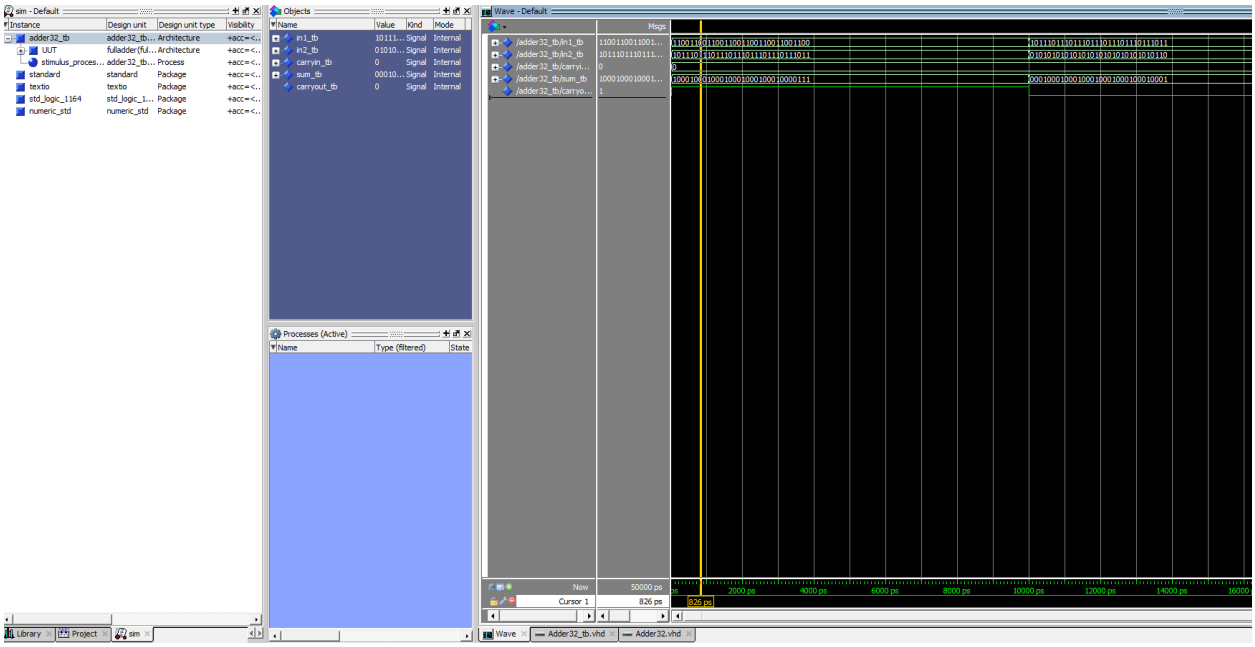
1.1.12. Adder 32bit

Main

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

ENTITY FullAdder IS port (
    in1, in2    : in  std_logic_vector(31 downto 0);
    carryin     : in  std_logic_vector(0 downto 0);
    sum         : out std_logic_vector(31 downto 0);
    carryout    : out std_logic);
END FullAdder;

-- Architecture for FullAdder
ARCHITECTURE FullAdder_1 of FullAdder is
    -- Temp signal for sum with 32th for carry
    signal tmp: std_logic_vector(32 downto 0);
BEGIN
    tmp      <= std_logic_vector(to_signed( to_integer(signed(in1)) + to_integer(signed(in2)) + to_integer(signed(carryin)),33));
    carryout <= tmp(32);
    sum      <= tmp(31 downto 0);
END FullAdder_1;
```

TestBench

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

entity Adder32_tb is
end Adder32_tb;

architecture testbench of Adder32_tb is
    -- Component declaration
    component FullAdder
        port (
            in1, in2    : in  std_logic_vector(31 downto 0);
            carryin     : in  std_logic_vector(0 downto 0);
            sum         : out std_logic_vector(31 downto 0);
            carryout    : out std_logic
        );
    end component;

    -- Test signals
    signal in1_tb      : std_logic_vector(31 downto 0);
    signal in2_tb      : std_logic_vector(31 downto 0);
    signal carryin_tb  : std_logic_vector(0 downto 0);
    signal sum_tb      : std_logic_vector(31 downto 0);
    signal carryout_tb : std_logic;

begin
    -- Component instantiation
    UUT: FullAdder
        port map (
            in1 => in1_tb,
            in2 => in2_tb,
            carryin => carryin_tb,
            sum => sum_tb,
            carryout => carryout_tb
        );

    -- Test process
    stimulus_process: process
    begin
        -- No carry in for both tests
        carryin_tb <= "0";

        -- Test case 1: 0xCCCCCCCC + 0xBBBBBBBB
        in1_tb <= x"CCCCCCCC";
        in2_tb <= x"BBBBBBBB";
        wait for 10 ns;

        -- Test case 2: 0xBBBBBBBB + 0x55555556
        in1_tb <= x"BBBBBBBB";
        in2_tb <= x"55555556";
        wait for 10 ns;

        -- End simulation
        wait;
    end process;

end testbench;
```



Σχολια - Παρατηρησεις :

O adder ειναι ενα unit που κανει προσθεσεις με δυο 32 bit number

Εχουμε in1 , in2 που ειναι 32 bit αριθμοι και carryin που ειναι 1 carry input

Εχουμε ενα sum για τα αποτελεσματα και ενα carry out αν υπαρξει overflow

Μετατρεπει τις εισοδους σε προσημασμενους ακεραιους αριθμους ,εκτελει προσθεση με προσωρινο αποτελεσμα tmp 32 bit και εξαγει το κρατουμενο απο το 32bit ,τελος επιστρεφει το αθροισμα sum

Test :

Είσοδος των ακόλουθων συνδυασμών και έλεγχος εξόδων

0xCCCCCCCC - 0xBBBBBBBB

0xBBBBBBBB – 0x555555556

1. 0xCCCCCCCC – 0xBBBBBBBB
   Εχουμε
   Αποτελεσμα : 0x88888887
   Carry out = 1
2. 0xBBBBBBBB – 0x555555556

Αποτελεσμα : 0x11111111
Carryout = 0

2 .Υλοποίηση και έλεγχος του επεξεργαστή

2.1 Entity MIPS

```
ENTITY MIPS IS port (

CLK : in std_logic;

 Rst : in std_logic;

outMIPS : out std_logic_vector(31 downto 0);

  );

END MIPS;
```

2.2 Εντολές

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY InstructionMemory IS
6     port (
7       inIM  : in  std_logic_vector(31 downto 0);
8       outIM : out std_logic_vector(31 downto 0)
9     );
10  END InstructionMemory;
11
12  ARCHITECTURE InstructionMemory_1 of InstructionMemory IS
13
14    type mem_type is array (natural range <>) of std_logic_vector(31 downto 0);
15    signal mem : mem_type(0 to 255) := (
16      0 => x"20000000",  -- addi $0, $0, 0
17      1 => x"20040000",  -- addi $4, $0, 0
18      2 => x"20030001",  -- addi $3, $0, 1
19      3 => x"20050003",  -- addi $5, $0, 3
20      4 => x"AC830000",  -- sw $3, 0($4)
21      5 => x"20630001",  -- addi $3, $3, 1
22      6 => x"20840001",  -- addi $4, $4, 1
23      7 => x"20A5FFFF",  -- addi $5, $5, -1
24      8 => x"14A0FFFB",  -- bne $5, $0, L1 (-5)
25      others => x"00000000"
26    );
27
28    signal FullInstruction : std_logic_vector(31 downto 0);
29
30  BEGIN
31
32    process(inIM)
33      variable addr : integer;
34    begin
35      addr := to_integer(unsigned(inIM));
36
37      if (addr >= 0 and addr <= 255) then
38        FullInstruction <= mem(addr);
39      else
40        FullInstruction <= x"00000000";
41        assert false
42          report "Instruction fetch out of bounds: PC = " & integer'image(addr)
43          severity note;
44      end if;
45    end process;
46
47    outIM <= FullInstruction;
48
49  END InstructionMemory_1;
50
```

2.3 Ανάπτυξη architecture του MIPS

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

ENTITY MIPS IS
  port (
    CLK     : in  std_logic;
    Rst     : in  std_logic;
    outMIPS : out std_logic_vector(31 downto 0);

    -- Debug outputs for waveforms
    debug_PC : out std_logic_vector(31 downto 0);
    debug_instruction : out std_logic_vector(31 downto 0);
    debug_reg_data1 : out std_logic_vector(31 downto 0);
    debug_reg_data2 : out std_logic_vector(31 downto 0);
    debug_alu_result : out std_logic_vector(31 downto 0);
    debug_mem_data : out std_logic_vector(31 downto 0);
    debug_write_reg : out std_logic_vector(4 downto 0);
    debug_write_data : out std_logic_vector(31 downto 0);
    debug_mem_addr : out std_logic_vector(31 downto 0);
    debug_controls : out std_logic_vector(7 downto 0) -- RegWrite, ALUSrc, MemWrite, MemRead, RegDst, MemToReg, Jump, Branch
  );
END MIPS;

ARCHITECTURE MIPS_1 of MIPS is

  component FullAdder
    port (
      in1, in2 : in  std_logic_vector(31 downto 0);
      carryin  : in  std_logic_vector(0 downto 0);
      sum      : out std_logic_vector(31 downto 0);
      carryout : out std_logic);
  end component;

  component ALU
    port (
      in1 : in std_logic_vector(31 downto 0);
      in2 : in std_logic_vector(31 downto 0);
      op  : in std_logic_vector(3 downto 0);
      outALU  : out std_logic_vector(31 downto 0);
      Zero    : out std_logic);
  end component;

  component ProgramCounter
    port (
      inPC  : in  std_logic_vector(31 downto 0);
      outPC : out std_logic_vector(31 downto 0);
      CLK   : in  std_logic;
      Rst   : in  std_logic);
  end component;

  component InstructionMemory
    port (
      inIM  : in  std_logic_vector(31 downto 0);
      outIM : out std_logic_vector(31 downto 0));
  end component;

  component Registers
    port (
      CLK         : in  std_logic;
      RegIn1      : in  std_logic_vector(4 downto 0);
      RegIn2      : in  std_logic_vector(4 downto 0);
      RegWriteIn  : in  std_logic_vector(4 downto 0);
      DataWriteIn : in  std_logic_vector(31 downto 0);
      RegWrite    : in  std_logic;
      RegOut1     : out std_logic_vector(31 downto 0);
      RegOut2     : out std_logic_vector(31 downto 0));
```

```
68      end component;
69
70      component ALUControl
71        port (
72          ALUOp       : in  std_logic_vector(2 downto 0);
73          Funct       : in  std_logic_vector(5 downto 0);
74          ALUCont_out : out std_logic_vector(3 downto 0));
75      end component;
76
77      component OutputControl
78        port (
79          CLK         : in std_logic;
80          OC_in       : in  std_logic_vector(5 downto 0);
81          RegWrite    : out std_logic;
82          ALUSrc      : out std_logic;
83          ALUOp       : out std_logic_vector(2 downto 0);
84          MemWrite    : out std_logic;
85          MemRead     : out std_logic;
86          RegDst      : out std_logic;
87          MemToReg    : out std_logic;
88          Jump        : out std_logic;
89          Branch      : out std_logic);
90      end component;
91
92      component Memory
93        port (
94          CLK       : in  std_logic;
95          inRAM     : in  std_logic_vector(31 downto 0);
96          WriteData : in  std_logic_vector(31 downto 0);
97          MemWrite  : in  std_logic;
98          MemRead   : in  std_logic;
99          outRAM    : out std_logic_vector(31 downto 0);
100         reset     : in  std_logic);
101     end component;
102
103     component MUX2_5
104       port (
105         MUXin1  :  in  std_logic_vector(4 downto 0);
106         MUXin2  :  in  std_logic_vector(4 downto 0);
107         MUXout  :  out std_logic_vector(4 downto 0);
108         sel     :  in  std_logic);
109     end component;
110
111     component MUX2_32
112       port (
113         MUXin1  :  in  std_logic_vector(31 downto 0);
114         MUXin2  :  in  std_logic_vector(31 downto 0);
115         MUXout  :  out std_logic_vector(31 downto 0);
116         sel     :  in  std_logic);
117     end component;
118
119     component SignExtend
120       port (
121         SignExIn  :  in  std_logic_vector(15 downto 0);
122         SignExOut :  out std_logic_vector(31 downto 0));
123     end component;
124
125     -- Signals
126     signal RegWrite, ALUSrc, MemWrite, MemRead, RegDst, MemToReg, Jump, Zero, Branch, BranchTaken : std_logic;
127     signal PC_FA_IM, FA_PC_OUT, OUT_IM, ONE, outALU : std_logic_vector(31 downto 0);
128     signal ALUOp : std_logic_vector(2 downto 0);
129     signal RegOut1, RegOut2, DataWriteIn, MUXaluOut, SignExOut, outRAM, ShiftJump2MuxJump, MuxJump2PC, MuxBranch2MuxJump, ALUbranchOut,
130     signal ALUControl_out : std_logic_vector(3 downto 0);
131     signal MUXregOut : std_logic_vector(4 downto 0);
```

```
132
133  BEGIN
134
135    -- PC increment and branch calc
136    ONE <= std_logic_vector(to_unsigned(1,32));
137    PC_plus_1 <= std_logic_vector(unsigned(PC_FA_IM) + unsigned(ONE));
138
139
140    PC_branch_target <= std_logic_vector(signed(PC_plus_1) + signed(SignExOut));
141
142    -- Program Counter
143    PC1: ProgramCounter
144      port map(
145        inPC  => MuxJump2PC,
146        outPC => PC_FA_IM,
147        CLK   => CLK,
148        Rst   => Rst);
149
150    -- Instruction Memory
151    IM1: InstructionMemory
152      port map(
153        inIM  => PC_FA_IM,
154        outIM => OUT_IM);
155
156    -- Register File
157    REG1: Registers
158      port map(
159        CLK          => CLK,
160        RegIn1       => OUT_IM(25 downto 21),
161        RegIn2       => OUT_IM(20 downto 16),
162        RegWriteIn   => MUXregOut,
163        DataWriteIn  => DataWriteIn,
164        RegWrite     => RegWrite,
165        RegOut1      => RegOut1,
166        RegOut2      => RegOut2);
167
168    -- Main Control Unit
169    OC1: OutputControl
170      port map(
171        CLK      => CLK,
172        OC_in    => OUT_IM(31 downto 26),
173        RegWrite => RegWrite,
174        ALUSrc   => ALUSrc,
175        ALUOp    => ALUOp,
176        MemWrite => MemWrite,
177        MemRead  => MemRead,
178        RegDst   => RegDst,
179        MemToReg => MemToReg,
180        Jump     => Jump,
181        Branch   => Branch);
182
183    -- ALU Control Unit
184    ALUC_1: ALUControl
185      port map(
186        ALUOp       => ALUOp,
187        Funct       => OUT_IM(5 downto 0),
188        ALUCont_out => ALUControl_out);
189
190    -- ALU
191    ALU1: ALU
192      port map(
193        in1    => RegOut1,
194        in2    => MUXaluOut,
195        op     => ALUControl_out,
```

```
196         outALU => outALU,
197         Zero   => Zero);
198 
199     -- Data Memory
200     RAM1: Memory
201       port map(
202         CLK       => CLK,
203         inRAM     => outALU,
204         WriteData => RegOut2,
205         MemWrite  => MemWrite,
206         MemRead   => MemRead,
207         outRAM    => outRAM,
208         reset     => Rst);
209 
210     -- MUXes
211     MUXreg: MUX2_5
212       port map(
213         MUXin1 => OUT_IM(20 downto 16),
214         MUXin2 => OUT_IM(15 downto 11),
215         MUXout => MUXregOut,
216         sel    => RegDst);
217 
218     MUXaluIn: MUX2_32
219       port map(
220         MUXin1 => RegOut2,
221         MUXin2 => SignExOut,
222         MUXout => MUXaluOut,
223         sel    => ALUSrc);
224 
225     MUXram: MUX2_32
226       port map(
227         MUXin1 => outALU,
228         MUXin2 => outRAM,
229         MUXout => DataWriteIn,
230         sel    => MemToReg);
231 
232     ShiftJump2MuxJump <= PC_plus_1(31 downto 28) & OUT_IM(25 downto 0) & "00";
233 
234     MUXjump: MUX2_32
235       port map(
236         MUXin1 => MuxBranch2MuxJump,
237         MUXin2 => ShiftJump2MuxJump,
238         MUXout => MuxJump2PC,
239         sel    => Jump);
240 
241     --  Complete branch logic for both beq and bne
242     -- beq (000100): branch if equal (Zero = '1')
243     -- bne (000101): branch if not equal (Zero = '0')
244     BranchTaken <= Branch and Zero when OUT_IM(31 downto 26) = "000100" else      -- beq
245                    Branch and (not Zero) when OUT_IM(31 downto 26) = "000101" else -- bne
246                    '0';
247 
248     MUXbranch: MUX2_32
249       port map(
250         MUXin1 => PC_plus_1,
251         MUXin2 => PC_branch_target,
252         MUXout => MuxBranch2MuxJump,
253         sel    => BranchTaken);
254 
255     SignEx1: SignExtend
256       port map(
257         SignExIn  => OUT_IM(15 downto 0),
258         SignExOut => SignExOut);
```

```
259 
260     -- Debug output connections
261     debug_PC <= PC_FA_IM;
262     debug_instruction <= OUT_IM;
263     debug_reg_data1 <= RegOut1;
264     debug_reg_data2 <= RegOut2;
265     debug_alu_result <= outALU;
266     debug_mem_data <= outRAM;
267     debug_write_reg <= MUXregOut;
268     debug_write_data <= DataWriteIn;
269     debug_mem_addr <= outALU when (MemRead = '1' or MemWrite = '1') else (others => '0');
270     debug_controls <= RegWrite & ALUSrc & MemWrite & MemRead & RegDst & MemToReg & Jump & Branch;
271 
272     outMIPS <= outALU;
273 
274 END MIPS_1;
```

Σχολια - Παρατηρήσεις:

Συνδέσαμε ολες τις επιμέρους μονάδες που ειχαμε φτιάξει προηγουμένως για τον επεξεργαστή mips .Εχουμε : program counter που κρατα την διευθυνση μνήμης . Να τονίσω οτι αυξάνεται στον ίδιο τον mips με pc+1 . Εχουμε, ενα instruction memory που εχουμε βάλει τις εντολές που

θελουμε να τρεξει το mips . Επιπλεον χωριζει το opcode , rs,rt,rd ,immediate και το output control παραγει τα καταλληλα σήηματα για regwrite ,alusrc,aluop etc. Οι registers διαβαζουν τα περιεχόμενα του rs ,rt . Αν alusrc = 0 τοτε χρηησιμοποιει τιμή απο δευτερο καταχωρητη , αν alusrc =1 τοτε χρησιμοποιει signextended immediate , η alu εκτελει τη πραξη και δινει αποτελεσμα ,δηλαδη outalu και σημα zero . Επίσης εχουμε memory access με χρηση εντολών οπως lw ,sw και τα σήματα memwrite οπου διαβαζει απο outalu ,memread γραφει στον regout2 . Εχουμε και memtoreg που γραφει απο μνημη σε καταχωρητη και υπολογισμο pc .Αν jumb=1 τοτε πηγαινει στο jumb address , αν branch =1 και zero =1 τοτε κανει branch ,αλλιως κανει pc+1.

Εχω βαλει στο entity debugs διοτι επερνα καποια λαθη οπως iteration erros , αρνητικο pc , branch faults και ηθελα να δω ακριβως τι γινεται στον επεξεργαστη οποτε θα δουμε μια πληρης εικονα του επεξεργαστη αργοτερα

2.4 Προσομοίωση - Έλεγχος λειτουργιών

2.4.1 Ρυθμιση ρολογιού

Θελουμε συχνότητα 200Mhz, Αρα T=1/f => T = 1/200 * 10^6 = 5 ns

Ολοκληρη η περιοδος πρεπει να ειναι 5 ns ,

Αρα οταν ειναι high ο παλμος πρεπει να διαρκεσει 2.5ns και οταν ειναι low πάλι 2.5ns ,ετσι ενας πλήρης κύκλος ειναι 5 ns.

```
clk_process: process
begin
  while true loop
    CLK <= '1';
    wait for 2.5 ns;
    CLK <= '0';
    wait for 2.5 ns;
  end loop;
end process;
```

2.4.2 Reset

```
rst_process: process
begin
  Rst <= '1';
  wait for 12.5 ns;
  Rst <= '0';
  wait;
end process;
```

2.4.3 Προσομοίωση

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY TestBench is
6   END TestBench;
7
8   ARCHITECTURE TestBench_MIPS of TestBench is
9
10  component MIPS
11      port (
12          CLK : in std_logic;
13          Rst : in std_logic;
14          outMIPS : out std_logic_vector(31 downto 0);
15          -- Debug outputs
16          debug_PC : out std_logic_vector(31 downto 0);
17          debug_instruction : out std_logic_vector(31 downto 0);
18          debug_reg_data1 : out std_logic_vector(31 downto 0);
19          debug_reg_data2 : out std_logic_vector(31 downto 0);
20          debug_alu_result : out std_logic_vector(31 downto 0);
21          debug_mem_data : out std_logic_vector(31 downto 0);
22          debug_write_reg : out std_logic_vector(4 downto 0);
23          debug_write_data : out std_logic_vector(31 downto 0);
24          debug_mem_addr : out std_logic_vector(31 downto 0);
25          debug_controls : out std_logic_vector(7 downto 0)
26      );
27  end component;
28
29  -- signals
30  signal CLK : std_logic := '0';
31  signal Rst : std_logic := '1';
32  signal outMIPS : std_logic_vector(31 downto 0);
33
34  -- Debug signals waveform viewing
35  signal debug_PC : std_logic_vector(31 downto 0);
36  signal debug_instruction : std_logic_vector(31 downto 0);
37  signal debug_reg_data1 : std_logic_vector(31 downto 0);
38  signal debug_reg_data2 : std_logic_vector(31 downto 0);
39  signal debug_alu_result : std_logic_vector(31 downto 0);
40  signal debug_mem_data : std_logic_vector(31 downto 0);
41  signal debug_write_reg : std_logic_vector(4 downto 0);
42  signal debug_write_data : std_logic_vector(31 downto 0);
43  signal debug_mem_addr : std_logic_vector(31 downto 0);
44  signal debug_controls : std_logic_vector(7 downto 0);
45
46  --  control signals
47  signal RegWrite : std_logic;
48  signal ALUSrc : std_logic;
49  signal MemWrite : std_logic;
50  signal MemRead : std_logic;
51  signal RegDst : std_logic;
52  signal MemToReg : std_logic;
53  signal Jump : std_logic;
54  signal Branch : std_logic;
55
56  -- Instruction decoding signals
57  signal opcode : std_logic_vector(5 downto 0);
58  signal rs : std_logic_vector(4 downto 0);
59  signal rt : std_logic_vector(4 downto 0);
60  signal rd : std_logic_vector(4 downto 0);
61  signal immediate : std_logic_vector(15 downto 0);
62  signal jump_addr : std_logic_vector(25 downto 0);
63
64
65  signal reg_write_occurred : std_logic := '0';
66  signal prev_write_reg : std_logic_vector(4 downto 0) := (others => '0');
67  signal prev_write_data : std_logic_vector(31 downto 0) := (others => '0');
68
69
70  signal mem_write_occurred : std_logic := '0';
71  signal mem_read_occurred : std_logic := '0';
```

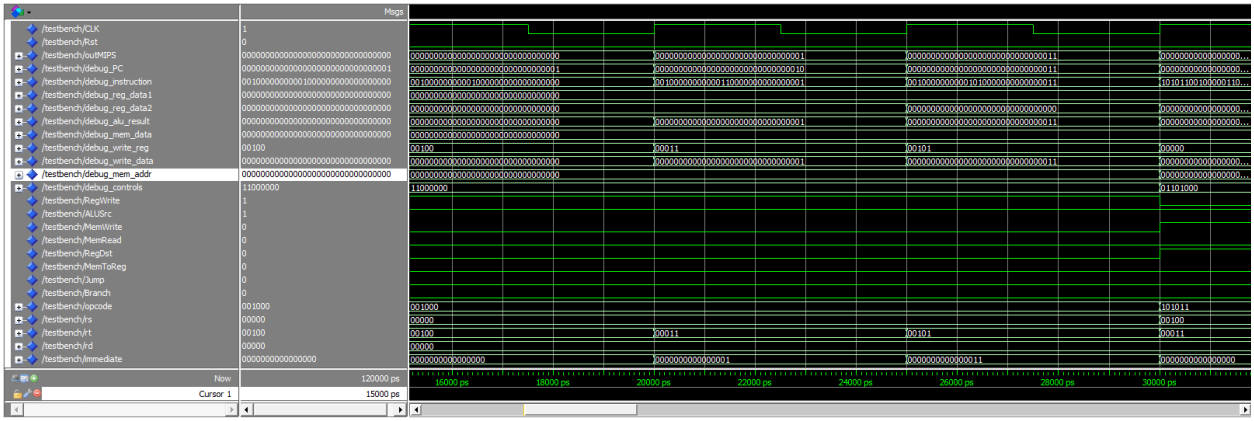

Ctrl+A Select All
Ctrl+G Goto
Ctrl+H Replace
Ctrl+U Delete Line
Ctrl+W Add Selection T
View All Short

```
    signal prev_mem_addr : std_logic_vector(31 downto 0) := (others => '0');
    signal prev_mem_data : std_logic_vector(31 downto 0) := (others => '0');

    BEGIN

    UUT: MIPS
        port map (
            CLK => CLK,
            Rst => Rst,
            outMIPS => outMIPS,
            debug_PC => debug_PC,
            debug_instruction => debug_instruction,
            debug_reg_data1 => debug_reg_data1,
            debug_reg_data2 => debug_reg_data2,
            debug_alu_result => debug_alu_result,
            debug_mem_data => debug_mem_data,
            debug_write_reg => debug_write_reg,
            debug_write_data => debug_write_data,
            debug_mem_addr => debug_mem_addr,
            debug_controls => debug_controls
        );


    RegWrite <= debug_controls(7);
    ALUSrc <= debug_controls(6);
    MemWrite <= debug_controls(5);
    MemRead <= debug_controls(4);
    RegDst <= debug_controls(3);
    MemToReg <= debug_controls(2);
    Jump <= debug_controls(1);
    Branch <= debug_controls(0);


    opcode <= debug_instruction(31 downto 26);
    rs <= debug_instruction(25 downto 21);
    rt <= debug_instruction(20 downto 16);
    rd <= debug_instruction(15 downto 11);
    immediate <= debug_instruction(15 downto 0);
    jump_addr <= debug_instruction(25 downto 0);

    -- Clock generation: 200 MHz, period = 5 ns
    clk_process: process
    begin
        while true loop
            CLK <= '1';
            wait for 2.5 ns;
            CLK <= '0';
            wait for 2.5 ns;
        end loop;
    end process;

    -- Reset
    rst_process: process
    begin
        Rst <= '1';
        wait for 12.5 ns;
        Rst <= '0';
        wait;
    end process;

    -- Process
    reg_monitor: process(CLK)
    begin
        if rising_edge(CLK) then
            if RegWrite = '1' and Rst = '0' then
                reg_write_occurred <= '1';
                prev_write_reg <= debug_write_reg;
                prev_write_data <= debug_write_data;
            else
                reg_write_occurred <= '0';
            end if;
        end if;
    end process;

    -- Process
    mem_monitor: process(CLK)
    begin
        if rising_edge(CLK) then
            if Rst = '0' then
                if MemWrite = '1' then
                    mem_write_occurred <= '1';
                    prev_mem_addr <= debug_mem_addr;
                    prev_mem_data <= debug_reg_data2;
                else
                    mem_write_occurred <= '0';
                end if;

                if MemRead = '1' then
                    mem_read_occurred <= '1';
                    prev_mem_addr <= debug_mem_addr;
                    prev_mem_data <= debug_mem_data;
                else
                    mem_read_occurred <= '0';
                end if;
            else
                mem_write_occurred <= '0';
                mem_read_occurred <= '0';
            end if;
        end if;
    end process;

    END TestBench_MIPS;
```

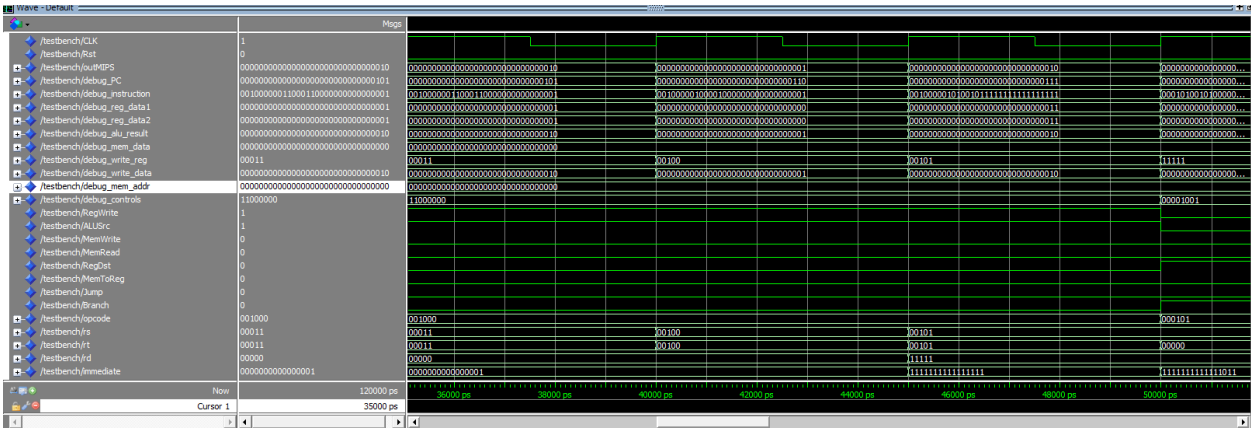
Σχολια - Παρατηρήσεις

Εχουμε 200Mhz συχνοτητα ,κανω monitor τους register να βλεπω ακριβως τι γινεται οπως και memory monitor ,κανω signal assignments RegWrite <= debug_controls(7) και αναλυση εντολω , opcode <= debug_instruction(31 downto 26); κτλπ

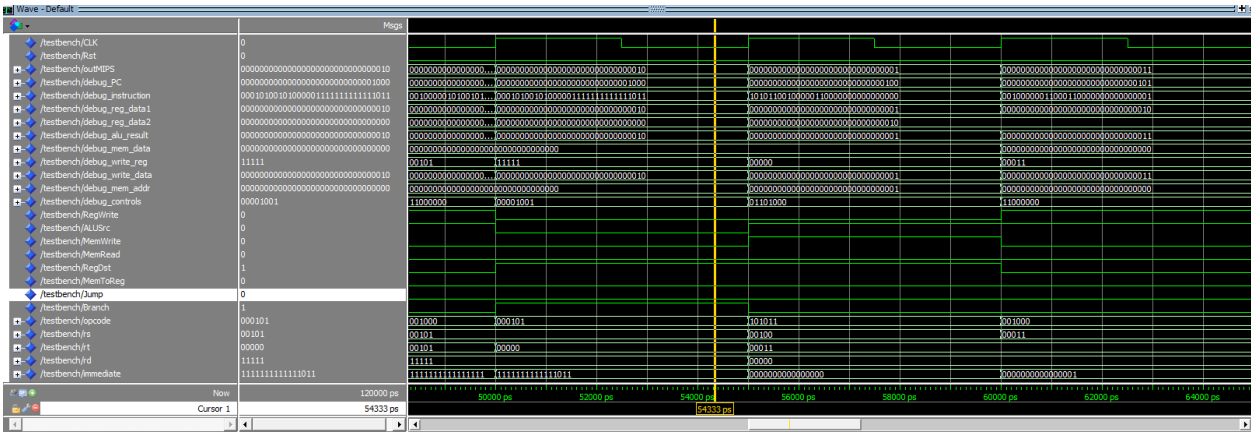
2.3.4 ΑΠΟΤΕΛΕΣΜΑΤΑ :



15ns – 30ns



35-50ns



55- 64ns branch γινεται 1 . Pc απο 8 γινεται 4

Ας δουμε τι γινεται αναλυτικα:

15 ns ,εχουμε pc = 0 , και η πρωτη μας εντολη ειναι η addi $0, $0, 0 . Και απλα κανει τον $0 , μηδεν , εχουμε reg used = 0 ,immediate = 0, alu result = 0 και control signals :

Regwrite = 1

Alusrc = 1

Regdst=0

Memtoreg=0

Memwrite = 0

Memread = 0

Jumb = 0

Branch = 0

20ns 15 ns ,εχουμε pc = 1 , και η δευτερη μας εντολη ειναι η addi $4, $0, 0 . Και απλα κανει τον $4 , μηδεν regwrite $0,0 ignored.

Επειτα 25ns addi $3, $0, 1

Βαζει στον καταχωρητη $3 το 1 και εχουμε aluresult =1

30ns pc =3 , intruction = addi $5, $0, 3

Alu result = 3

Στα 35 ns εχουμε sw $3, 0($4), κανει store $3 δηλαδη το 1 στο mem $4, alu = 0 και signals RegWrite=0, MemWrite=1, ALUSrc=1

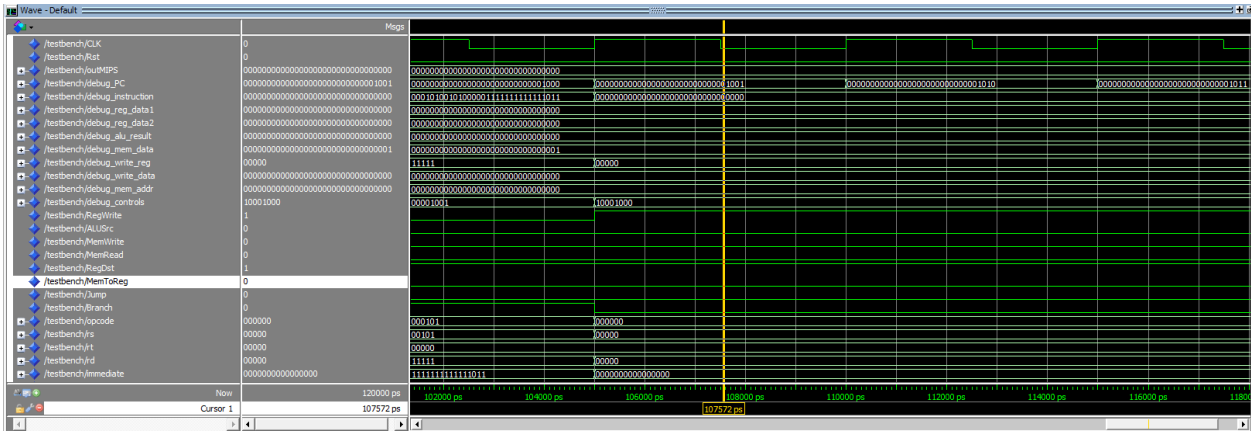
Στα 40ns pc = 5 , intructions = addi $3, $3, 1 , αρα γινεται 2 ο καταχωρητης 3, aluresult = 2

Στα 45 ns pc = 6 , instructions = addi $4, $4, 1, κανει το $4 ,1

Στα 50 ns pc = 7 , instructions = addi $5, $5, -1, κανει αφαιρεση τον$5 απο 3 σε 2, Alu result = 2

Στα 55ns pc = 8 ,instructions = bne $5, $0, -5, το $5 δεν ειναι ισο με $0 αρα παει πισω στο branch, pc = 4,control signals : Branch=1, RegWrite=0, MemWrite=0

Αρα επαναλαμβανει αυτο αλλες 2 φορες μεχρι οτου το $5 να ειναι = με $0 δηλαδη με 0 .



Εδω βγαινει απο Loop. Η προσωμίωση ετρεξε περιπου για 20-22 εφοσον βγει απο τη λουπα