ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΤΕΛΙΚΗ ΕΡΓΑΣΙΑ

ΟΝΟΜΑΤΕΠΏΝΥΜΟ : Μαρίνο Τσελάνι
ΕΞΑΜΗΝΟ ΦΟΙΤΗΤΗ: 4ο
ΑΜ : 20390241
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2022

Αυτή η εργασία έχει ως σκοπό την σχεδίαση και υλοποίηση του επεξεργαστή MIPS. Το πρόγραμμα αποτελείται από τα αρχεία :

Shifter.vhd,
cntrlAlu.vhd,
mn_dedomenon.vhd,
mn_entolon.vhd,
5mux.vhd,
alu.vhd,
and_gate.vhd,
cntrl.vhd,
pc.vhd,
regfile.vhd,
32mux.vhd,
adder.vhd,
epektash16_32.vhd που χρησιμοποιούνται από το κεντρικο πρόγραμμα(telikh_askhsh.vhd), ενώ το test bench κεντρικού προγράμματος είναι το tb_telikh_askish.vhd .

1. ALU

Ο κώδικας βρίσκεται στο alu.vhd :

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

ENTITY alu IS                          -- 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 alu;

-- OP mapping:
--
-- 0000: do nothing
-- 0001: sum/subtract
-- 0010: and
-- 0011: or
-- 0100: nor
-- 0101: and immediate (with constant)
-- 0110: or immediate (with constant)
-- 0111: shift left
-- 1000: shift right
-- 1001: add immediate
-- 1010: check if equal
-- 1011: check if not equal
-- 1100: slt operation
-- 1101: addi

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

	with op select
		outALU <=
			-- sum:
			out_FA when "0001",
			-- and:
			in1 and in2 when "0010",
			-- or:
			in1 or in2 when "0011",
			-- nor:
			in1 nor in2 when "0100",
			-- and immediate:
			in1 and in2 when "0101",
			-- or immediate:
			in1 or in2 when "0110",
			-- shift left:
			out_SH  when "0111",
			-- shift right
			out_SH  when "1000",
			-- slt
			slt_result when "1100",
			-- addi
			out_FA when "1101",
```

```
                                x when others;

        Zero <=
                -- beq
                '1' when ((in1 = in2) and (op = "1010")) else
                -- bne
                '1' when ((in1 /= in2) and (op = "1011")) else
                '0';
END ALU_1;
```

2. Register file

Ο κώδικας βρίσκεται στο regflie.vhd :

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

ENTITY Registers is
        port (
                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'));
        signal RegWriteDelayed : std_logic;
        signal x: std_logic_vector(31 downto 0) := (others=>'X');
--
BEGIN
--
        RegOut1 <= regs(to_integer(unsigned(RegIn1)));
        RegOut2 <= regs(to_integer(unsigned(RegIn2)));
        --
        RegWriteDelayed <= transport RegWrite after 1 ns;
        --
        regs(to_integer(unsigned(RegWriteIn))) <= DataWriteIn when (RegWriteDelayed='1' and DataWriteIn/=x);
--
END Registers_1;
```

3. Μνήμη δεδομένων

Ο κώδικας βρίσκεται στο mn_dedomenon.vhd :

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

```
ENTITY Memory IS
	port (
		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

	type ram_type is array (natural range <>) of std_logic_vector(31 downto 0);
	signal ram:    ram_type(0 to 1023) := (others=> (others => '0'));
	signal Address: integer := 0;

BEGIN

	Address <= to_integer(unsigned(inRAM))
	    when (to_integer(unsigned(inRAM)) <= 1023)
	    else 0;


	ram(Address)    <= WriteData when (MemWrite='1' and reset='0');

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

END Memory_1;
```

4. Μνήμη εντολών

Ο κώδικας βρίσκεται στο mn_entolon.vhd :

```
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(7 downto 0);
	signal mem: mem_type(0 to 1023) := (
	0 => "00100001",
1 => "01000001",
2 => "00000000",
3 => "00010100",
```

```
4 => "00100001",
5 => "01000010",
6 => "00000000",
7 => "00001010",
8 => "00100001",
9 => "01000011",
10 => "00000000",
11 => "00110010",
12 => "00000000",
13 => "00100010",
14 => "00101000",
15 => "00100000",
16 => "10101100",
17 => "01100101",
18 => "00000000",
19 => "00000101",
20 => "10001100",
21 => "01101010",
22 => "00000000",
23 => "00000101",
                others=> (others => '0'));

        signal FullInstruction: std_logic_vector(31 downto 0);
        signal IM_address:      integer;

BEGIN

        IM_address <= to_integer(unsigned(inIM));
        FullInstruction <= mem(IM_address) & mem(IM_address+1) & mem(IM_address+2) & mem(IM_address+3)
                        when (IM_address >= 0) else std_logic_vector(to_signed(-1,32));
        outIM <= FullInstruction;

END InstructionMemory_1;
```

5. Μονάδα ελέγχου

Ο κώδικας βρίσκεται στο cntrl.vhd :

```
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
--
	with OC_in select
		RegWrite <=
			('1' and CLK) when "100011",
			('1' and CLK) when "000000",
			('1' and CLK) when "001000",
			'0' when others;

	with OC_in select
		ALUSrc <=
			'1' after 2 ns when "100011",
			'1' after 2 ns when "101011",
			'1' after 2 ns when "001000",
			'0' when others;

	with OC_in select
		ALUOp <=
			"000" after 2 ns when "000000",
			"001" after 2 ns when "100011",
			"001" after 2 ns when "101011",
			"010" after 2 ns when "000100",
			"011" after 2 ns when "000101",
			"100" after 2 ns when "001000",
			"111" when others;

	with OC_in select
		MemWriteen "100011",
			'0' when "001000",
			'1' when others;

	with OC_in select
		Jump <=
			'1' when "000010",
			'0' when others;

	with OC_in select
		Branch <=
			'1' when "000100",
			'1' when "000101",
			'0' when others;
END OutputControl_1; <=
			'1' after 10 ns when "101011",
			'0' when others;

	with OC_in select
	MemRead <=
	'1' after 2 ns when "100011",
	'0' when others;

	with OC_in select
		MemToReg <=
			'1' after 2 ns when "100011",
			'0' when others;

	with OC_in select
		RegDst <=
```

```
                                '0' when "100011",
                                '0' when "001000",
                                '1' when others;

        with OC_in select
                Jump <=
                                '1' when "000010",
                                '0' when others;

        with OC_in select
                Branch <=
                                '1' when "000100",
                                '1' when "000101",
                                '0' when others;
END OutputControl_1;
```

6. Μονάδα ελέγχου ALU

Ο κώδικας βρίσκεται στο cntrlAlu.vhd :

```
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 tmpALUControl_func: std_logic_vector(3 downto 0) := (others=>'0');
        signal tmpALUControl_op:   std_logic_vector(3 downto 0) := (others=>'0');
--
-- OP mapping:
--
-- 0000: do nothing
-- 0001: sum/subtract
-- 0010: and
-- 0011: or
-- 0100: nor
-- 0101: and immediate (with constant)
-- 0110: or immediate (with constant)
-- 0111: shift left
-- 1000: shift right
-- 1001: add immediate
-- 1010: check if equal
-- 1011: check if not equal
-- 1100: slt operation
-- 1101: addi
--
-- ALUOp mapping:
-- 00: R type -> i look the funct operation
-- 11: I type -> i have to set the sum in ALU
```

```
-- 10: I type branch instructions
--
BEGIN


	with Funct select
		tmpALUControl_func <=
			"0001" when "100000",
			"0010" when "100100",
			"0011" when "100101",
			"0100" when "100111",
			"0101" when "001100",
			"0110" when "001101",
			"0111" when "000000",
			"1000" when "000010",
			"1100" when "101010",
			"1111" when others;
	with ALUOp select
		tmpALUControl_op <=
			"0001" when "001",
			"1010" when "010",
			"1011" when "011",
			"1101" when "100",
			"1111" when others;
	with ALUOp select
		ALUCont_out <=
			tmpALUControl_func when "000",
			tmpALUControl_op   when others;

END ALUControl_1;
```

7. PC

Ο κώδικας βρίσκεται στο pc.vhd :

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

ENTITY ProgramCounter IS
	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 ProgramCounter;

ARCHITECTURE ProgramCounter_1 of ProgramCounter is

BEGIN
	reg: process(CLK)
	begin
		if (Rst='1') then
			outPC <= std_logic_vector(to_signed(-4,32));
		end if;
		if rising_edge(CLK) then
			outPC <= inPC;
```

```
                end if;
        end process;
END ProgramCounter_1;
```

8. 5-πλό πολυπλέκτη 2-σε-1

Ο κώδικας βρίσκεται στο 5mux.vhd :

```
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 MUX2_5_1;
```

9. Μονάδα επέκτασης προσήμου 16-σε-32

Ο κώδικας βρίσκεται στο epektash16_32.vhd :

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

ENTITY SignExtend IS
        port (
                SignExIn  :  in  std_logic_vector(15 downto 0);
                SignExOut :  out std_logic_vector(31 downto 0));
END SignExtend;

ARCHITECTURE SignExtend_1 of SignExtend is
--
        signal ones   : std_logic_vector(15 downto 0) := (others=>'1');
        signal zeros  : std_logic_vector(15 downto 0) := (others=>'0');
--
BEGIN
        SignExOut <= ones  & SignExIn  when SignExIn(15)='1' else
                zeros & SignExIn  when SignExIn(15)='0';
END SignExtend_1;
```

10. 32-πλό πολυπλέκτη 2-σε-1

Ο κώδικας βρίσκεται στο 32mux.vhd :

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

ENTITY MUX2_32 IS
	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 MUX2_32;

ARCHITECTURE MUX2_32_1 of MUX2_32 is

BEGIN
	MUXout <= MUXin1 when sel='0' else
		MUXin2 when sel='1';
END MUX2_32_1;
```

11. Μονάδα ολίσθησης αριστερά κατά 2 (32-bit)

Ο κώδικας βρίσκεται στο shifter.vhd :

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

ENTITY MUX2_32 IS
	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 MUX2_32;

ARCHITECTURE MUX2_32_1 of MUX2_32 is

BEGIN
	MUXout <= MUXin1 when sel='0' else
		MUXin2 when sel='1';
END MUX2_32_1;
```

12. Αθροιστή 32 bits

Ο κώδικας βρίσκεται στο adder.vhd :

```
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 FullAdder_1 of FullAdder is


	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;
```

13. AND gate

Ο κώδικας βρίσκεται στο and_gate.vhd :

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;


entity and_gate is

	port(

		a : in  std_ulogic;

		b : in  std_ulogic;

		c : out std_ulogic

	);

end entity and_gate;


architecture ANDGATE of and_gate is
```

```
begin

	c <= a and b;

end architecture ANDGATE;
```

Κεντρικό προγραμμα

Ο κώδικας βρίσκεται στο telikh_askhsh.vhd:

```
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));
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_ulogic;
			Rst   : in  std_ulogic);
	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 (
		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;

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;

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;
		RegDst     : out std_logic;
		MemToReg    : out std_logic;
		Jump       : out std_logic;
		Branch     : out std_logic);
end component;

component Memory
	port (
		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;

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;

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;
```

```
component SignExtend
        port (
                SignExIn  :  in  std_logic_vector(15 downto 0);
                SignExOut :  out std_logic_vector(31 downto 0));
end component;

-- Signals between blocks
signal RegWrite:    std_logic;
signal ALUSrc:      std_logic;
signal MemWrite:    std_logic;
signal MemRead:     std_logic;
signal RegDst:      std_logic;
signal MemToReg:    std_logic;
signal Jump:        std_logic;
signal Zero:        std_logic;
signal Branch:      std_logic;
signal BranchTaken: std_logic;
signal PC_FA_IM:          std_logic_vector(31 downto 0);
signal FA_PC_OUT:         std_logic_vector(31 downto 0);
signal OUT_IM:            std_logic_vector(31 downto 0);
signal FOUR:              std_logic_vector(31 downto 0);
signal outALU:            std_logic_vector(31 downto 0);
signal ALUOp:             std_logic_vector(2 downto 0);
signal RegOut1:           std_logic_vector(31 downto 0);
signal RegOut2:           std_logic_vector(31 downto 0);
signal ALUControl_out:    std_logic_vector(3 downto 0);
signal DataWriteIn:       std_logic_vector(31 downto 0);
signal MUXregOut:         std_logic_vector(4 downto 0);
signal MUXaluOut:         std_logic_vector(31 downto 0);
signal SignExOut:         std_logic_vector(31 downto 0);
signal outRAM:            std_logic_vector(31 downto 0);
signal ShiftJump2MuxJump: std_logic_vector(31 downto 0);
signal MuxJump2PC:        std_logic_vector(31 downto 0);
signal MuxBranch2MuxJump: std_logic_vector(31 downto 0);
signal ALUbranchOut:      std_logic_vector(31 downto 0);
signal SignExOutAligned:  std_logic_vector(31 downto 0);

BEGIN
        --
        FOUR <= std_logic_vector(to_unsigned(4,32));
        FA_PC1: FullAdder
                port map(
                        in1    => PC_FA_IM,
                        in2    => FOUR,
                        carryin => "0",
                        sum    => FA_PC_OUT);

        SignExOutAligned <= SignExOut(29 downto 0) & "00";
        FA_BRANCH: FullAdder
                port map(
                        in1    => FA_PC_OUT,
                        in2    => SignExOutAligned,
                        carryin => "0",
                        sum    => ALUbranchOut);

        PC1: ProgramCounter
                port map(
```

```
                inPC  => MuxJump2PC,
                outPC => PC_FA_IM,
                CLK   => CLK,
                Rst   => Rst);

IM1: InstructionMemory
        port map(
                inIM  => PC_FA_IM,
                outIM => OUT_IM);

REG1: Registers
        port map(
                RegIn1     => OUT_IM(25 downto 21),
                RegIn2     => OUT_IM(20 downto 16),
                RegWriteIn  => MUXregOut,
                DataWriteIn => DataWriteIn,
                RegWrite    => RegWrite,
                RegOut1     => RegOut1,
                RegOut2     => RegOut2);

OC1: OutputControl
        port map(
                CLK     => CLK,
                OC_in    => OUT_IM(31 downto 26),
                RegWrite => RegWrite,
                ALUSrc   => ALUSrc,
                ALUOp    => ALUOp,
                MemWrite => MemWrite,
                MemRead  => MemRead,
                RegDst   => RegDst,
                MemToReg => MemToReg,
                Jump     => Jump,
                Branch   => Branch);

ALUC_1: ALUControl
        port map(
                ALUOp      => ALUOp,
                Funct      => OUT_IM(5 downto 0),
                ALUCont_out => ALUControl_out);
ALU1: ALU
        port map(
                in1    => RegOut1,
                in2    => MUXaluOut,
                op     => ALUControl_out,
                outALU => outALU,
                Zero   => Zero);
RAM1: Memory
        port map(
                inRAM     => outALU,
                WriteData => RegOut2,
                MemWrite  => MemWrite,
                MemRead   => MemRead,
                outRAM    => outRAM,
                reset     => Rst);
MUXreg: MUX2_5
        port map(
                MUXin1 => OUT_IM(20 downto 16),
                MUXin2 => OUT_IM(15 downto 11),
```

```
                MUXout => MUXregOut,
                sel    => RegDst);

        MUXaluIn: MUX2_32
                port map(
                        MUXin1 => RegOut2,
                        MUXin2 => SignExOut,
                        MUXout => MUXaluOut,
                        sel    => ALUsrc);

        MUXram: MUX2_32
                port map(
                        MUXin1 => outALU,
                        MUXin2 => outRAM,
                        MUXout => DataWriteIn,
                        sel    => MemToReg);



        ShiftJump2MuxJump <= FA_PC_OUT(31 downto 28) & OUT_IM(25 downto 0) & "00";
        MUXjump: MUX2_32
                port map(
                        MUXin1 => MuxBranch2MuxJump,
                        MUXin2 => ShiftJump2MuxJump,
                        MUXout => MuxJump2PC,
                        sel    => Jump);

        BranchTaken <= Branch and Zero;
        MUXbranch: MUX2_32
                port map(
                        MUXin1 => FA_PC_OUT,
                        MUXin2 => ALUbranchOut,
                        MUXout => MuxBranch2MuxJump,
                        sel    => BranchTaken);

        SignEx1: SignExtend
                port map(
                        SignExIn  =>  OUT_IM(15 downto 0),
                        SignExOut => SignExOut);


        outMIPS  <= outALU;
-
END MIPS_1;
```

Στο πρόγραμμα αυτό χρησιμοποιούμε τα προηγούμενα .vhd αρχεία ως components και με την καθοδήγηση της εικόνας με τον χάρτη του MIPS που δίνεται στην εκφώνηση θα πράξουμε ανάλογα στο πρόγραμμα .

Test Bench Κεντρικού προγράμματος

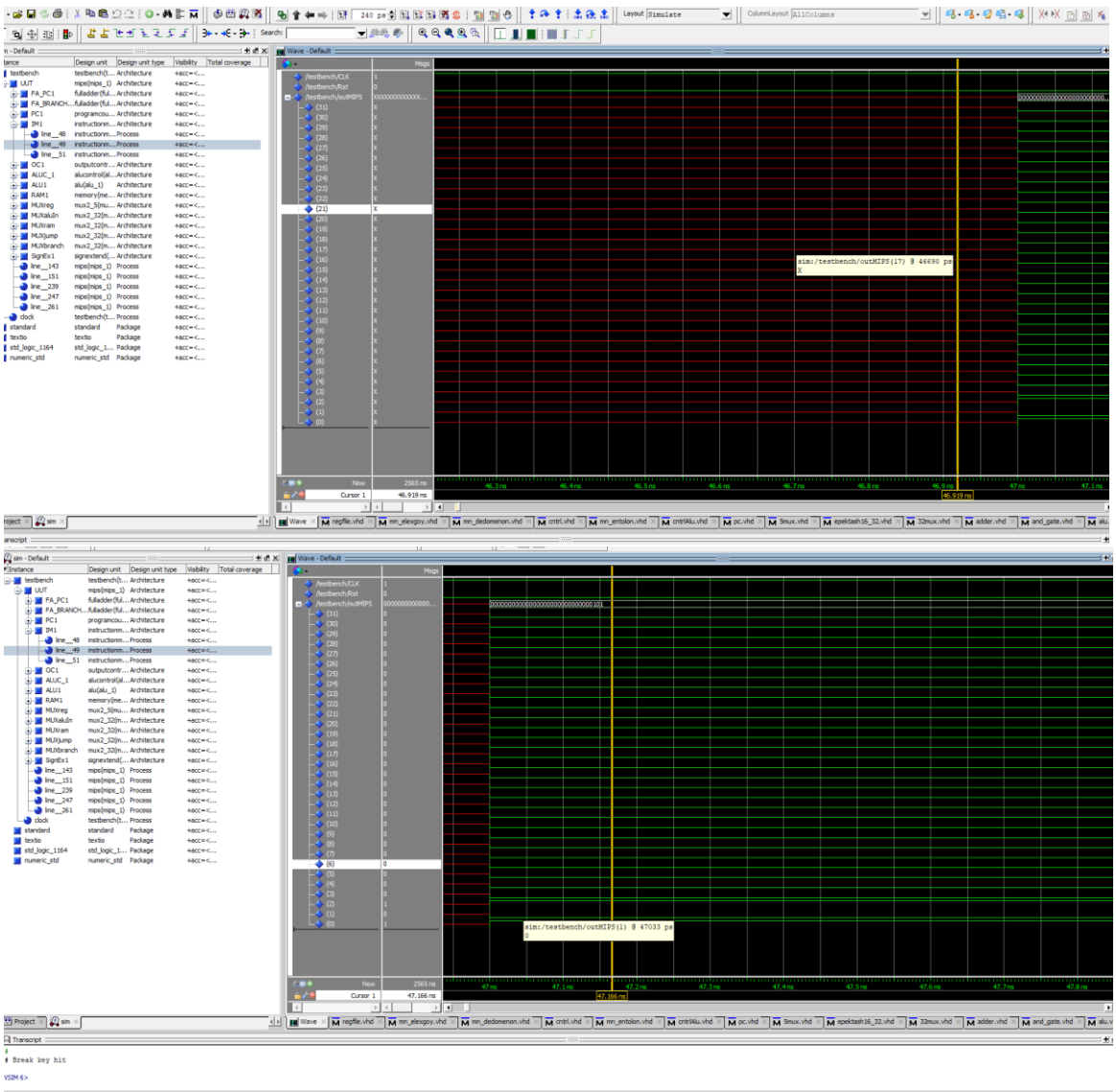
Ο κώδικας βρίσκεται στο tb_telikh_askhsh.vhd:

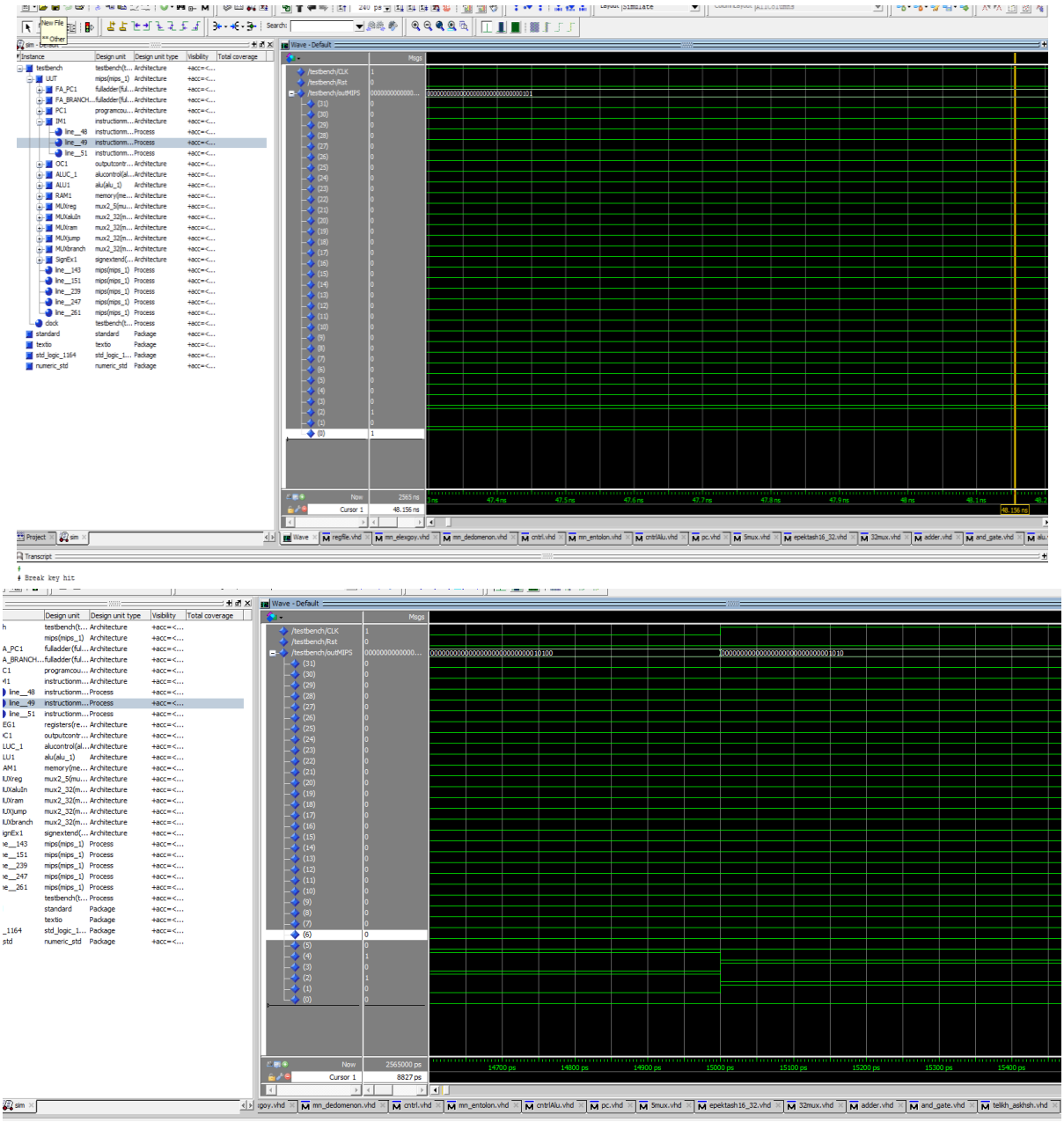
```
library ieee;
use ieee.std_logic_1164.all;

ENTITY TestBench is
END TestBench;
ARCHITECTURE TestBench_MIPS of TestBench is
	component MIPS
		port (
			CLK     : in  std_logic;
			Rst     : in  std_logic;
			outMIPS : out std_logic_vector(31 downto 0));
	end component;
	signal CLK: std_logic;
	signal Rst: std_logic;
	signal outMIPS: std_logic_vector(31 downto 0);
BEGIN
	UUT:  MIPS
		port map (CLK, Rst, outMIPS);

	clock: process
		variable clktmp : std_logic:='1';
		variable reset  : std_logic:='1';
	begin
		clktmp:= NOT clktmp;
		CLK <= clktmp;
		if (reset = '1') then
			Rst <= reset;
			reset := '0';
		else
```

```
                Rst <= '0';
            end if;
            wait for 5 ns;
    end process;
END TestBench_MIPS;
```

Στο test bench ουσιαστικά έχουμε το κύριο πρόγραμμα ως component και έπειτα ρυθμίζεται τα clock και reset ώστε να λειτουργεί το πρόγραμμα ορθά.

Οι παρακάτω φωτογραφίες απεικονίζουν την λειτουργία του MIPS με το compile και run του tb_telikh_askhsh.vhd .




Wave - Default
/testbench/CLK
/testbench/Rst
/testbench/outMIPS
Break key hit