

**Πανεπιστήμιο Δυτικής Αττικής**

**Σχολή Μηχανικών**

**Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών**

**Σχεδίαση Ψηφιακών**

**Συστημάτων**

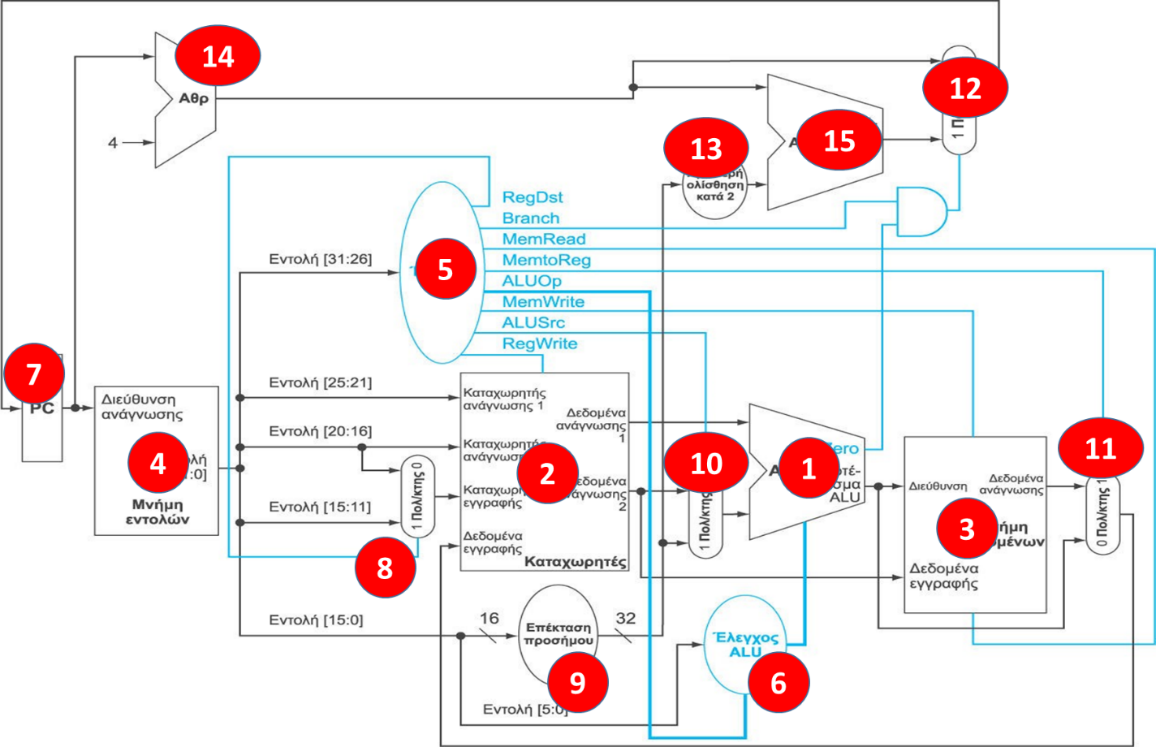
**Εξαμηνιαία Εργασία**

**ΝΙΚΟΛΑΟΣ ΘΩΜΑΣ**

**ΑΜ: 21390068**

**ΑΘΗΝΑ**

**Πέμπτη, 15 Ιουνίου 2023**

****

***Εικόνα 1: Υλοποίηση Επεξεργαστή.***

**Κώδικας Κυκλώματος:**

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 structure of MIPS is

component ALU port(

in1,in2: in std\_logic\_vector(31 downto 0);

opcode: in std\_logic\_vector(3 downto 0);

ALUout: out std\_logic\_vector(31 downto 0);

Zero: out std\_logic);

end component;

-- Η ALU όπως φαίνεται και από την εικόνα 1, έχει 3 εισόδους και 2 εξόδους.

component RegFile 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;

-- Το Register File όπως φαίνεται και από την εικόνα 1, έχει 5 εισόδους και 2 εξόδους.

component DataMemory port(

RAMin: in std\_logic\_vector(31 downto 0);

WriteData: in std\_logic\_vector(31 downto 0);

MemWrite: in std\_logic;

MemRead: in std\_logic;

RAMout: out std\_logic\_vector(31 downto 0);

reset: in std\_logic);

end component;

-- Η Μνήμη Δεδομένμων όπως φαίνεται και από την εικόνα 1, έχει 5 εισόδους, 1 έξοδο και το reset.

component InstructionMemory port(

IMin: in std\_logic\_vector(31 downto 0);

IMout: out std\_logic\_vector(31 downto 0));

end component;

-- Η Μνήμη Εντολών όπως φαίνεται και από την εικόνα 1, έχει 1 είσοδο και 1 έξοδο.

component ControlUnit 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,MemRead: out std\_logic;

RegDest: out std\_logic;

MemToReg: out std\_logic;

Jump: out std\_logic;

Branch: out std\_logic);

end component;

-- Η Μονάδα Ελέγχου όπως φαίνεται και από την εικόνα 1, έχει 1 είσοδο, 7 εξόδους και το clock.

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;

-- Η Μονάδα Ελέγχου ALU όπως φαίνεται και από την εικόνα 1, έχει 2 εισόδους και 1 έξοδο.

component PC 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;

-- Το PC όπως φαίνεται και από την εικόνα 1, έχει 1 είσοδο, 1 έξοδο, το clock και το reset.

component mux2to1\_5 port(

min1,min2: in std\_logic\_vector(4 downto 0);

mout: out std\_logic\_vector(4 downto 0);

sel: in std\_logic);

end component;

-- Ο 5-πλός πολυπλέκτης 2-σε-1 όπως φαίνεται και από την εικόνα 1, έχει 2 εισόδους, 1 έξοδο και 1 ακόμη είσοδο.

component SignExtend port(

S\_EXin: in std\_logic\_vector(15 downto 0);

S\_EXout: out std\_logic\_vector(31 downto 0));

end component;

-- Η Μονάδα επέκτασης προσήμου 16-σε-32 όπως φαίνεται και από την εικόνα 1, έχει 1 είσοδο και 1 έξοδο.

component mux2to1\_32 port(

min1,min2: in std\_logic\_vector(31 downto 0);

mout: out std\_logic\_vector(31 downto 0);

sel: in std\_logic);

end component;

-- Ο 32-πλός πολυπλέκτης 2-σε-1 όπως φαίνεται και από την εικόνα 1, έχει 2 εισόδους, 1 έξοδο και 1 ακόμη είσοδο.

component Adder32 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;

-- Ο Αθροιστής 32 bits όπως φαίνεται και από την εικόνα 1, έχει 2 εισόδους, έξοδο, το carryin και το carryout bit.

signal RegWrite,ALUSrc,MemWrite,MemRead,RegDest,MemToReg,Jump,Zero,Branch,BranchTaken: std\_logic;

signal PC\_FA\_IM,FA\_PC\_OUT,OUT\_IM,FOUR,ALUout,RegOut1,RegOut2,DataWriteIn,MUXaluOut: std\_logic\_vector(31 downto 0);

signal ALUControl\_out: std\_logic\_vector(3 downto 0);

signal MUXregOut: std\_logic\_vector(4 downto 0);

signal ALUOp: std\_logic\_vector(2 downto 0);

signal SignExOut,RAMout,ShiftJump2MuxJump,MuxJump2PC,MuxBranch2MuxJump,ALUbranchout,SignExOutAligned: std\_logic\_vector(31 downto 0);

-- Όλα τα σήματα που χρησιμοποιούνται και παρακάτω γίνεται η αντιστοίχιση τους.

begin

FOUR <= std\_logic\_vector(to\_unsigned(4,32));

FA\_PC1: Adder32 port map(

in1 => PC\_FA\_IM,

in2 => FOUR,

carryin => "0",

sum => FA\_PC\_OUT);

SignExOutAligned <= SignExOut(29 downto 0) & "00";

Fa\_Branch: Adder32 port map(

in1 => FA\_PC\_OUT,

in2 => SignExOutAligned,

carryin => "0",

sum => ALUbranchout);

PC1: PC port map(

inPC => MuxJump2PC,

outPC => PC\_FA\_IM,

clk => clk,

rst => rst);

IM1: InstructionMemory port map(

IMin => PC\_FA\_IM,

IMout => OUT\_IM);

Reg1: RegFile 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: ControlUnit port map(

clk => clk,

OC\_in => OUT\_IM(31 downto 26),

RegWrite => RegWrite,

ALUSrc => ALUSrc,

ALUOp => ALUOp,

MemWrite => MemWrite,

MemRead => MemRead,

RegDest => RegDest,

MemToReg => MemToReg,

Jump => Jump,

Branch => Branch);

ALUC: ALUControl port map(

ALUOp => ALUOp,

Funct => OUT\_IM(5 downto 0),

ALUCont\_out => ALUControl\_out);

ALU1: ALU port map(

in1 => RegOut1,

in2 => MUXaluOut,

opcode => ALUControl\_out,

ALUout => ALUout,

Zero => Zero);

RAM: DataMemory port map(

RAMin => ALUout,

WriteData => RegOut2,

MemWrite => MemWrite,

MemRead => MemRead,

RAMout => RAMout,

reset => rst);

Mreg: mux2to1\_5 port map(

Min1 => OUT\_IM(20 downto 16),

Min2 => OUT\_IM(15 downto 11),

Mout => MUXregOut,

sel => RegDest);

Malu: mux2to1\_32 port map(

Min1 => RegOut2,

Min2 => SignExOut,

Mout => MUXaluOut,

sel => ALUSrc);

Mram: mux2to1\_32 port map(

Min1 => ALUout,

Min2 => RAMout,

Mout => DataWriteIn,

sel => MemToReg);

ShiftJump2MuxJump <= FA\_PC\_OUT(31 downto 28) & OUT\_IM(25 downto 0) & "00";

Mjump: mux2to1\_32 port map(

Min1 => MuxBranch2Muxjump,

Min2 => ShiftJump2MuxJump,

Mout => MuxJump2Pc,

sel => Jump);

BranchTaken <= Branch and Zero;

Mbranch: mux2to1\_32 port map(

Min1 => FA\_PC\_OUT,

Min2 => ALUbranchOut,

Mout => MuxBranch2Muxjump,

sel => BranchTaken);

S\_EX: SignExtend port map(

S\_EXin => OUT\_Im(15 downto 0),

S\_EXout => SignExOut);

outMIPS <= ALUout;

end structure;

--ALU

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity ALU is port(

in1,in2: std\_logic\_vector(31 downto 0);

opcode: in std\_logic\_vector(3 downto 0);

ALUout: out std\_logic\_vector(31 downto 0);

Zero: out std\_logic);

end ALU;

architecture Behavioral of ALU is

component Adder32 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 Shifter port(

Sin: in std\_logic\_vector(31 downto 0);

Sout: out std\_logic\_vector(31 downto 0);

Sopcode: in std\_logic;

num: in std\_logic\_vector(4 downto 0));

end component;

-- Η Μονάδα ολίσθησης αριστερά κατά 2 (32-bit) έχει 3 εισόδους και 1 έξοδο.

signal op\_shifter,carryout: std\_logic;

signal out\_FA,out\_SH,slt\_result: std\_logic\_vector(31 downto 0);

signal x: std\_logic\_vector(31 downto 0) := (others=>'X');

begin

op\_shifter <= '0' when opcode="0111" else '1' when opcode="1000" else 'X';

FA\_ALU: Adder32 port map(

in1 => in1,

in2 => in2,

carryin => "0",

sum => out\_FA,

carryout => carryout);

SH\_ALU: Shifter port map(

Sin => in1,

Sout => out\_SH,

Sopcode => 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 opcode select ALUout <=

out\_FA when "0001",

in1 and in2 when "0010",

in1 or in2 when "0011",

in1 nor in2 when "0100",

in1 and in2 when "0101",

in1 or in2 when "0110",

out\_SH when "0111",

out\_SH when "1000",

slt\_result when "1100",

out\_Fa when "1101",

x when others;

Zero <= '1' when ((in1 = in2) and (opcode = "1010")) else '1' when ((in1 /= in2) and (opcode = "1011")) else '0';

end Behavioral;

-- Η ALU είναι υπεύθυνη για την εκτέλεση αριθμητικών και λογικών πράξεων σε δεδομένα. Οπότε ανάλογα με το opcode που δέχεται σε είσοδο κάνει και την ανάλογη πράξη.

--ALUControl

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 behavioral of ALUControl is

signal tmpALUControl\_func,tmpALUControl\_op: std\_logic\_vector(3 downto 0) :=(others=>'0');

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 behavioral;

-- Η Μονάδα ελέγχου ALU ελέγχει τις λογικές και αριθμητικές πράξεις που εκτελεί η ALU με βάση τα opcode. Συνολικά, η Μονάδα ελέγχου ALU διασφαλίζει ότι η ALU λειτουργεί σωστά και εκτελεί τις απαιτούμενες πράξεις για τη σωστή εκτέλεση των εντολών MIPS.

--RegFile

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity RegFile is port(

RegIn1,RegIn2,RegWriteIn: in std\_logic\_vector(4 downto 0);

DatawriteIn: in std\_logic\_vector(31 downto 0);

RegWrite: in std\_logic;

RegOut1,RegOut2: out std\_logic\_vector(31 downto 0));

end RegFile;

architecture behavioral of RegFile 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 behavioral;

-- Το Register File αναλαμβάνει την αποθήκευση και την ανάκτηση δεδομένων από τα καταχωρητές του επεξεργαστή.

--Adder32

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity Adder32 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 Adder32;

architecture behavioral of Adder32 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 behavioral;

-- Το Adder32 είναι υπεύθυνο για την εκτέλεση των αριθμητικών πράξεων πρόσθεσης μέσα στον MIPS.

--InstructionMemory

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity InstructionMemory is port(

IMin: in std\_logic\_vector(31 downto 0);

IMout: out std\_Logic\_vector(31 downto 0));

end InstructionMemory;

architecture behavioral of INstructionMemory is

type mem\_type is array (0 to 31) of std\_logic\_vector(31 downto 0);

signal mem: mem\_type := (

"00100000000000000000000000000000", -- addi $0, $0, 0

"00100000010000100000000000000000", -- addi $2, $2, 0

"00100000010001000000000000000000", -- addi $2, $4, 0

"00100000011000000000000000000001", -- addi $3, $0, 1

"00100000101000000000000000000011", -- addi $5, $0, 3

"00000000011000110011000000100000", -- L1: add $6, $3, $0

"10101100100001100000000000000000", -- sw $6, 0($4)

"00100000011000110000000000000001", -- addi $3, $3, 1

"00100000100001000000000000000001", -- addi $4, $4, 1

"00100000101001011111111111111110", -- addi $5, $5, -1

"00010100101000001111111111111010", -- bne $5, $0, L1

"00000000000000000000000000000000", -- No Operation

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000", -- NOP

"00000000000000000000000000000000" -- NOP

);

signal FullInstruction: std\_logic\_vector(31 downto 0);

signal IM\_address: integer;

begin

IM\_address <= to\_integer(unsigned(IMin)) when (to\_integer(unsigned(IMin)) >=0) else 0;

FullInstruction <= mem(IM\_address) when (IM\_address >= 0) else std\_logic\_vector(to\_signed(-1,32));

IMout <= FullInstruction;

end behavioral;

-- Η Instruction Memory είναι μια μνήμη που αποθηκεύει τις εντολές που εκτελεί ο επεξεργαστής, στην περίπτωση μας αυτές που δίνονται από την εκφώνηση.

--mux2to1\_5

library ieee;

use ieee.std\_logic\_1164.all;

entity mux2to1\_5 is port(

Min1,Min2: in std\_logic\_vector(4 downto 0);

Mout: out std\_logic\_vector(4 downto 0);

sel: in std\_logic);

end mux2to1\_5;

architecture behavioral of mux2to1\_5 is

begin

Mout <= Min1 when sel='0' else Min2 when sel='1';

end behavioral;

-- Το Mux2to1\_5 (5bit) επιλέγει μεταξύ δύο εισόδων και παράγει μια μοναδική έξοδο.

--mux2to1\_32

library ieee;

use ieee.std\_logic\_1164.all;

entity mux2to1\_32 is port(

Min1,Min2: in std\_logic\_vector(31 downto 0);

Mout: out std\_logic\_vector(31 downto 0);

sel: in std\_logic);

end mux2to1\_32;

architecture behavioral of mux2to1\_32 is

begin

Mout <= Min1 when sel='0' else Min2 when sel='1';

end behavioral;

-- Το Mux2to1\_5 (32bit) επιλέγει μεταξύ δύο εισόδων και παράγει μια μοναδική έξοδο.

--ControlUnit

library ieee;

use ieee.std\_logic\_1164.all;

entity ControlUnit is 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,MemRead: out std\_logic;

RegDest: out std\_logic;

MemToReg: out std\_logic;

Jump: out std\_logic;

Branch: out std\_logic);

end ControlUnit;

architecture behavioral of ControlUnit 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

MemWrite <=

'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

RegDest <=

'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 behavioral;

-- Η μονάδα ελέγχου αναγνωρίζει τον τύπο της κάθε εντολής και παρέχει τα αντίστοιχα σήματα ελέγχου στα υπόλοιπα κυκλώματα του mips, ρυθμίζοντας τη ροή των δεδομένων και των ελέγχων.

--PC

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity PC 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 PC;

architecture behavioral of PC is

begin

reg: process(CLK)

begin

if (rst='1') then

outPC <= std\_logic\_vector(to\_signed(-1,32));

end if;

if rising\_edge(CLK) then

outPC <= inPC;

end if;

end process;

end behavioral;

-- Ο PC χρησιμοποιείται για να παρακολουθεί την εκτέλεση του προγράμματος και να δείχνει την διεύθυνση της επόμενης εντολής που πρέπει να εκτελεστεί.

--DataMemory

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity DataMemory is port(

RAMin: in std\_logic\_vector(31 downto 0);

WriteData: in std\_logic\_vector(31 downto 0);

MemWrite: in std\_logic;

MemRead: in std\_logic;

RAMout: out std\_logic\_vector(31 downto 0);

reset: in std\_logic);

end DataMemory;

architecture behavioral of DataMemory is

type ram\_type is array (natural range <>) of std\_logic\_vector(31 downto 0);

signal ram: ram\_type(0 to 15) := (others=> (others=>'0'));

signal Address: integer := 0;

begin

Address <= to\_integer(unsigned(RAMin)) when (to\_integer(unsigned(RAMin)) <= 15) else 0;

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

with reset select

RAMout <=

ram(Address) when '0',

(others=>'0') when others;

end behavioral;

-- Χρησιμοποιείται για την αποθήκευση δεδομένων που χρησιμοποιούνται από το πρόγραμμα που εκτελείται.

--Shifter

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity Shifter is port(

Sin: in std\_logic\_vector(31 downto 0);

Sout: out std\_logic\_vector(31 downto 0);

Sopcode: in std\_logic;

num: in std\_logic\_vector(4 downto 0));

end Shifter;

architecture behavioral of Shifter is

signal tmp: unsigned(31 downto 0);

begin

tmp <= to\_unsigned(to\_integer(signed(Sin)),tmp'length) sll

to\_integer(signed(num)) when Sopcode='0' else

to\_unsigned(to\_integer(signed(Sin)),tmp'length) srl

to\_integer(signed(num)) when Sopcode='1';

Sout <= std\_logic\_vector(to\_signed(to\_integer(tmp),Sout'length));

end behavioral;

-- Ο shifter εκτελεί τις λογικές πράξεις της μετατόπισης, προκειμένου να αλλάξει τη θέση των bits στον αριθμό προς τα αριστερά.

--SignExtend

library ieee;

use ieee.std\_logic\_1164.all;

entity SignExtend is port(

S\_EXin: in std\_logic\_vector(15 downto 0);

S\_EXout: out std\_logic\_vector(31 downto 0));

end SignExtend;

architecture behavioral of SignExtend is

signal ones: std\_logic\_vector(15 downto 0) := (others=>'1');

signal zeros: std\_logic\_vector(15 downto 0) := (others=>'0');

begin

S\_EXout <= ones & S\_EXin when S\_EXin(15)='1' else

zeros & S\_EXin when S\_EXin(15)='0';

end behavioral;

-- Το SignExtend μετατρέπει έναν δυαδικό αριθμό μικρότερου μήκους σε έναν δυαδικό αριθμό μεγαλύτερου μήκους, διατηρώντας το πρόσημο του αρχικού αριθμού.

**Κώδικας TestBench:**

library ieee;

use ieee.std\_logic\_1164.all;

entity mips\_tb is

end mips\_tb;

architecture mipstb of mips\_tb is

component mips port(

clk: in std\_logic;

rst: in std\_logic;

outMips: out std\_logic\_vector(31 downto 0));

end component;

signal clk,rst: std\_logic;

signal outMips: std\_logic\_vector(31 downto 0);

begin

M: MIPS port map(clk,rst,outMips);

cl: process

variable clktmp,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 33 ns;

end process;

end mipstb;

library ieee;

use ieee.std\_logic\_1164.all;

entity ALU\_TB is

end ALU\_TB;

architecture TestALU of ALU\_TB is

component ALU port(

in1,in2: in std\_logic\_vector(31 downto 0);

opcode: in std\_logic\_vector(3 downto 0);

outALU: out std\_logic\_vector(31 downto 0);

zero: out std\_logic);

end component;

signal clk,rst: std\_ulogic;

signal ind1,ind2,output: std\_logic\_vector(31 downto 0);

signal oper: std\_logic\_vector(3 downto 0);

begin

A: ALU port map(ind1,ind2,oper,output);

cl: process

variable clktmp: std\_ulogic:='0';

variable reset: std\_ulogic:='1';

begin

clktmp:= NOT clktmp;

clk <= clktmp;

if (reset='1') then

rst <= reset;

reset := '0';

else

rst <= '0';

end if;

wait for 50 ns;

end process;

s: process(clk)

begin

if (clk='1' and clk'event) then

ind1 <= "00000000000000000000000010010110";

ind2 <= "00000000000000000000001000000010";

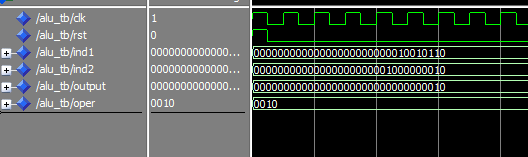
oper <= "0010";

end if;

end process;

end TestALU;

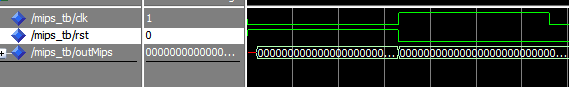
**Alu TestBench.**



***Εικόνα 2: Wave από το testbench***

***της ALU.***

**Mips TestBench.**



***Εικόνα 3: Wave από το testbench***

***του MIPS.***