

**Σχεδίαση Ψηφιακών Συστημάτων**

**Ρόμπερτ Πολοβίνα – 23390338**

Εργασία Εξαμήνου – Σχεδίαση MIPS

2024-2025

**Μέρος Πρώτο: Εισαγωγή**

Σκοπός της εργασίας είναι η σχεδίαση και υλοποίηση του επεξεργαστή MIPS απλού κύκλου. Ο επεξεργαστής έχει ως είσοδο τα σήματα reset και clock, ενώ δεν έχει έξοδο. Το σήμα reset οδηγεί τη μονάδα PC στην τιμή 0. Επίσης οδηγεί το Register file και μηδενίζει όλους τους καταχωρητές. Ο επεξεργαστής θα εκτελεί τις εντολές: add, sub, addi, lw, sw, bne.

Ο επεξεργαστής περιλαμβάνει τις ακόλουθες μονάδες:

A table with numbers and letters

AI-generated content may be incorrect.

A diagram of a machine

AI-generated content may be incorrect.

Η παρούσα εργασία αποτελείται από τρία μέρη, με το πρώτο να είναι αυτή η εισαγωγή. Στο δεύτερο μέρος θα δούμε τους κώδικες και τους ελέγχους των διαφόρων components τα οποία αποτελούν τον MIPS, και στο τρίτο παρουσιάζεται η διασύνδεση όλων αυτών των εξαρτημάτων ώστε να δημιουργηθεί και να λειτουργήσει ο επεξεργαστής μας.

Πριν προχωρήσουμε θα ήθελα να επισημάνω τρία πράγματα. Πρώτον, οι εικόνες που συνοδεύουν το κάθε εξάρτημα έχουν επαρκή ανάλυση. Λόγω όμως της εγγενούς δυσκολίας της απεικόνισης τους σε σελίδες με διάταξη πορτραίτου, η χρήση της λειτουργίας της μεγέθυνσης κρίνεται απαραίτητη.

Δεύτερον, στα Testbenches μου δεν υλοποιώ το ρολόι μέσα σε process, όπως συνηθίζεται, αλλά σε μία μόνο γραμμή κώδικα:

*clk* ***<=******not*** *clk* ***after*** *clk\_period****/****2****;***

Τον τρόπο αυτόν υλοποίησης τον διδάχθηκα στο εργαστήριο του μαθήματος ως μία πιο γρήγορη και απλή εναλλακτική.

Τέλος, επειδή παρατηρείται σε πάνω από ένα Testbench θα ήθελα να εξηγήσω εδώ την ύπαρξη της επόμενης γραμμής κώδικα:

*std****.****env****.****stop****;***

Σε οποιαδήποτε Testbenches περιλαμβάνεται η παραπάνω εντολή, η προσομοίωση έτρεχε απ’ αόριστον μετά την επιλογή του “Run”. Η συμπεριφορά αυτή παρατηρείται και σε testbenches στα οποία χρησιμοποιούμε ρολόι αλλά και σε άλλα στα οποία δεν γίνεται χρήση ρολογιού. Επίσης, οι κώδικες είναι σχετικά απλοί και δεν διαφέρουν θεμελιωδώς μεταξύ τους. Συνεπώς δεν μπόρεσα να καταλάβω γιατί συμβαίνει αυτό το πρόβλημα σε μερικά testbenches και όχι σε άλλα.

Εν τέλει η εντολή αυτή ήταν η γρηγορότερη και ευκολότερη λύση χωρίς να επηρεάζει τα αποτελέσματα των δοκιμών και γενικότερα για τους σκοπούς της εργασίας ήταν απλά αρκετή για να σταματάω την εκτέλεση του testbench εκεί που πρέπει.

**Μέρος Δεύτερο: Components**

***1.1.1. ALU***

Κώδικας ALU:

**library** IEEE**;**

**use** IEEE**.**std\_logic\_1164**.ALL;**

**use** IEEE**.**numeric\_std**.ALL;**

**entity** ALU **is** **port(**

ALUin1**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

ALUin2**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

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

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

zero**:** **out** std\_logic**);**

**end** ALU**;**

**architecture** behavioral **of** ALU **is**

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

**begin**

**process(**ALUin1**,** ALUin2**,** ALUctrl**)**

**begin**

**case(**ALUctrl**)** **is**

**when** "0000" **=>** result **<=** ALUin1 **AND** ALUin2**;**

**when** "0001" **=>** result **<=** ALUin1 **OR** ALUin2**;**

**when** "0010" **=>** result **<=** std\_logic\_vector**(**signed**(**ALUin1**)** **+** signed**(**ALUin2**));**

**when** "0110" **=>** result **<=** std\_logic\_vector**(**signed**(**ALUin1**)** **-** signed**(**ALUin2**));**

**when** **others** **=>** result **<=** **(others** **=>** '0'**);**

**end** **case;**

**end** **process;**

zero **<=** '1' **when** result **=** x"00000000" **else** '0'**;**

ALUresult **<=** result**;**

**end** behavioral**;**

H ALU είναι υπέυθυνη να εκτελεί πράξεις αναλόγως το σήμα ελέγχου που δέχεται από την μονάδα ελέγχου της. Στο testbench ελέγχουμε την ALU κάνοντας τις πράξεις που ορίζει η εκφώνηση.

Κώδικας ALU tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** ALU\_tb **is**

**end** ALU\_tb**;**

**architecture** behavior **of** ALU\_tb **is**

**signal** ALUin1**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** ALUin2**:** std\_logic\_vector**(**31 **downto** 0**);**

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

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

**begin**

uut**:** **entity** work**.**ALU

**port** **map(**

ALUin1 **=>** ALUin1**,**

ALUin2 **=>** ALUin2**,**

ALUctrl **=>** ALUctrl**,**

ALUresult **=>** ALUresult**);**

**process**

**begin**

-- 7+(-3)

ALUin1 **<=** std\_logic\_vector**(to\_signed(**7**,**32**));**

ALUin2 **<=** std\_logic\_vector**(to\_signed(-**3**,**32**));**

ALUctrl **<=** "0010"**;**

**wait** **for** 10 ns**;**

-- 6+(-6)

ALUin1 **<=** std\_logic\_vector**(to\_signed(**6**,**32**));**

ALUin2 **<=** std\_logic\_vector**(to\_signed(-**6**,**32**));**

ALUctrl **<=** "0010"**;**

**wait** **for** 10 ns**;**

-- 5-8

ALUin1 **<=** std\_logic\_vector**(to\_signed(**5**,**32**));**

ALUin2 **<=** std\_logic\_vector**(to\_signed(**8**,**32**));**

ALUctrl **<=** "0110"**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

A screenshot of a computer

AI-generated content may be incorrect.

***1.1.2. Register File***

Κώδικας Register File:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Registerfile **is** **port(**

clk**:** **in** std\_logic**;**

reset**:** **in** std\_logic**;**

read\_reg1**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

read\_reg2**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

write\_reg**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

write\_data**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

write\_enable**:** **in** std\_logic**;**

read\_data1**:** **out** std\_logic\_vector**(**31 **downto** 0**);**

read\_data2**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** Registerfile**;**

**architecture** behavioral **of** Registerfile **is**

**type** reg\_array **is** **array(**15 **downto** 0**)** **of** std\_logic\_vector**(**31 **downto** 0**);**

**signal** registers**:** reg\_array **:=** **(others** **=>** **(others** **=>** '0'**));**

**begin**

read\_data1 **<=** registers**(to\_integer(**unsigned**(**read\_reg1**)));**

read\_data2 **<=** registers**(to\_integer(**unsigned**(**read\_reg2**)));**

**process(**clk**,** reset**)**

**begin**

**if** reset **=** '1' **then**

registers **<=** **(others** **=>** **(others** **=>** '0'**));**

**elsif** **rising\_edge(**clk**)** **then**

**if** write\_enable**=** '1' **then**

registers**(to\_integer(**unsigned**(**write\_reg**)))** **<=** write\_data**;**

**end** **if;**

**end** **if;**

**end** **process;**

**end** behavioral**;**

Το αρχείο καταχωρητών έχει υλοποιηθεί ως 16 θέσεις των 32 bits. Η ανάγνωση δεδομένων συμβαίνει εκτός του process. Πριν προχωρήσουμε στην εγγραφή των δεδομένων, ελέγχουμε για το σήμα reset. Αν υπάρχει, οι καταχωρητές μηδενίζονται.

Έπειτα ελέγχουμε το σήμα του ρολογιού και αν έχουμε σήμα εγγραφής. Αν ναι, προχωράμε στην εγγραφή των δεδομένων. Στο testbench ελέγχουμε την λειτουργία reset καθώς και τις παρακάτω:

* Εγγραφή του 6 στον καταχωρητή $4
* Εγγραφή του 9 στον καταχωρητή $5
* Εγγραφή του 3 στον καταχωρητή $6
* Ανάγνωση των καταχωρητών $4 και $5

Κώδικας File Register tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Registerfile\_tb **is**

**end** Registerfile\_tb**;**

**architecture** behavior **of** Registerfile\_tb **is**

**signal** clk**:** std\_logic **:=** '0'**;**

**signal** reset**:** std\_logic **:=** '0'**;**

**signal** read\_reg1**:** std\_logic\_vector**(**4 **downto** 0**);**

**signal** read\_reg2**:** std\_logic\_vector**(**4 **downto** 0**);**

**signal** write\_reg**:** std\_logic\_vector**(**4 **downto** 0**);**

**signal** write\_data**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** write\_enable**:** std\_logic**;**

**signal** read\_data1**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** read\_data2**:** std\_logic\_vector**(**31 **downto** 0**);**

**constant** clk\_period**:** time **:=** 10 ns**;**

**begin**

uut**:** **entity** work**.**Registerfile **port** **map(**

clk **=>** clk**,**

reset **=>** reset**,**

read\_reg1 **=>** read\_reg1**,**

read\_reg2 **=>** read\_reg2**,**

write\_reg **=>** write\_reg**,**

write\_data **=>** write\_data**,**

write\_enable **=>** write\_enable**,**

read\_data1 **=>** read\_data1**,**

read\_data2 **=>** read\_data2**);**

clk **<=** **not** clk **after** clk\_period**/**2**;**

**process**

**begin**

--Reset

reset **<=** '1'**;**

**wait** **for** clk\_period**;**

reset **<=** '0'**;**

-- 6 -> $4

write\_enable **<=** '1'**;**

write\_reg **<=** "00100"**;**

write\_data **<=** std\_logic\_vector**(to\_signed(**6**,**32**));**

**wait** **for** clk\_period**;**

-- 9 -> $5

write\_reg **<=** "00101"**;**

write\_data **<=** std\_logic\_vector**(to\_signed(**9**,**32**));**

**wait** **for** clk\_period**;**

-- 3 -> $6

write\_reg **<=** "00110"**;**

write\_data **<=** std\_logic\_vector**(to\_signed(**3**,**32**));**

**wait** **for** clk\_period**;**

write\_enable **<=** '0'**;**

read\_reg1 **<=** "00100"**;** --$4

read\_reg2 **<=** "00101"**;** --$5

**wait** **for** clk\_period**;**

std**.**env**.**stop**;**

**wait;**

**end** **process;**

**end** behavior**;**

**A screenshot of a computer

AI-generated content may be incorrect.**

***1.1.3. Data Memory***

Κώδικας Datamem:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Datamem **is** **port(**

clk**:** **in** std\_logic**;**

mem\_write**:** **in** std\_logic**;**

mem\_read**:** **in** std\_logic**;**

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

write\_data**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

read\_data**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** Datamem**;**

**architecture** behavioral **of** Datamem **is**

**type** memory\_array **is** **array(**15 **downto** 0**)** **of** std\_logic\_vector**(**31 **downto** 0**);**

**signal** memory**:** memory\_array **:=** **(others** **=>** **(others** **=>** '0'**));**

**begin**

**process(**clk**)**

**begin**

**if** **rising\_edge(**clk**)** **then**

**if** mem\_write **=** '1' **then**

memory**(to\_integer(**unsigned**(**address**)))** **<=** write\_data**;**

**end** **if;**

**end** **if;**

**end** **process;**

**process(**mem\_read**,** address**,** memory**)**

**begin**

**if** mem\_read **=** '1' **then**

read\_data **<=** memory**(to\_integer(**unsigned**(**address**)));**

**end** **if;**

**end** **process;**

**end** behavioral**;**

Η μνήμη δεδομένων έχει επίσης υλοποιηθεί ως 16 θέσεις των 32 bits. Εδώ η ανάγνωση και η εγγραφή γίνονται σε 2 διαφορετικά processes. Για την εγγραφή ελέγχουμε το σήμα εγγραφής δεδομένων και αν είναι 1, προχωράμε στην εγγραφή στην επιθυμητή θέση μνήμης.

Για την ανάγνωση ελέγχουμε εάν το σήμα ανάγνωσης δεδομένων είναι 1 και αν ναι, πραγματοποιείται η ανάγνωση των δεδομένων της θέσης μνήμης που θέλουμε.

Στο testbench ελέγχουμε τις παρακάτω λειτουργίες:

* Εγγραφή του 9 στη θέση μνήμης 2.
* Εγγραφή του 4 στη θέση μνήμης 3.
* Ανάγνωση της θέσης μνήμης 2.

Κώδικας Datamem tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Datamem\_tb **is**

**end** Datamem\_tb**;**

**architecture** behavior **of** Datamem\_tb **is**

**signal** clk**:** std\_logic **:=** '0'**;**

**signal** mem\_write**:** std\_logic**;**

**signal** mem\_read**:** std\_logic**;**

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

**signal** write\_data**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** read\_data**:** std\_logic\_vector**(**31 **downto** 0**);**

**constant** clk\_period**:** time **:=** 10 ns**;**

**begin**

uut**:entity** work**.**Datamem **port** **map(**

clk **=>** clk**,**

mem\_write **=>** mem\_write**,**

mem\_read **=>** mem\_read**,**

address **=>** address**,**

write\_data **=>** write\_data**,**

read\_data **=>** read\_data**);**

clk **<=** **not** clk **after** clk\_period**/**2**;**

**process**

**begin**

mem\_write **<=** '1'**;**

mem\_read **<=** '0'**;**

-- write 9 -> 2

address **<=** x"00000002"**;**

write\_data **<=** std\_logic\_vector**(to\_signed(**9**,**32**));**

**wait** **for** clk\_period**;**

-- write 4 -> 3

address **<=** x"00000003"**;**

write\_data **<=** std\_logic\_vector**(to\_signed(**4**,**32**));**

**wait** **for** clk\_period**;**

-- read <- 2

mem\_write **<=** '0'**;**

mem\_read **<=** '1'**;**

address **<=** x"00000002"**;**

**wait** **for** clk\_period**;**

std**.**env**.**stop**;**

**wait;**

**end** **process;**

**end** behavior**;**

A computer screen shot of a black and green screen

AI-generated content may be incorrect.

***1.1.4. Instruction Memory***

Κώδικας Imem:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Imem **is** **port(**

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

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

**end** Imem**;**

**architecture** behavioral **of** Imem **is**

**type** mem\_array **is** **array(**15 **downto** 0**)** **of** std\_logic\_vector**(**31 **downto** 0**);**

**signal** memory**:** mem\_array **:=** **(**

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

**others** **=>** **(others** **=>** '0'**));**

**begin**

instruction **<=** memory**(to\_integer(**unsigned**(**address**(**3 **downto** 0**))));**

**end** behavioral**;**

Όπως και οι προηγούμενες μνήμες, έτσι και η μνήμη εντολών έχει υλοποιηθεί ως 16 θέσεις των 32 bits. Προς διευκόλυνση, οι ζητούμενες εντολές είναι hardcoded στην μνήμη.

Στο τέλος χρησιμοποιούμε μόνο τα τελευταία 4 bits των διευθύνσεων διότι αλλιώς το Modelsim εμφάνιζε το επόμενο σφάλμα κατά την εκκίνηση της προσομοίωσης:

*“Fatal error in Architecture behavioral at C:/altera/13.1/modelsim\_ase/win32aloem/3.2.1.4 Imem.vhd line 27”*.

Αυτό συνέβαινε διότι μία εντολή αποτελείται από 32 bits ενώ ο πίνακας μας έχει 16 θέσεις, συνεπώς υπήρχε πιθανότητα να βγούμε εκτός ορίων.

Η πιο συνηθισμένη λύση σε απλουστευμένες προσομοιώσεις όπως αυτή, είναι να κρατάμε μόνο τα τελευταία 4 bits διότι μόνο τόσα χρειάζονται για να προσπελάσουμε θέση σε μνήμη έως 16 θέσεων (2^4= 16).

Στο testbench πραγματοποιούμε ανάγνωση της εντολής που περιέχεται στην θέση μνήμης 4.

Κώδικας Imem tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Imem\_tb **is**

**end** Imem\_tb**;**

**architecture** behavior **of** Imem\_tb **is**

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

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

**begin**

uut**:** **entity** work**.**Imem

**port** **map(**

address **=>** address**,**

instruction **=>** instruction**);**

**process**

**begin**

address **<=** x"00000004"**;**

**wait** **for** 10 ns**;**

std**.**env**.**stop**;**

**end** **process;**

**end** behavior**;**

A screenshot of a computer

AI-generated content may be incorrect.

***1.1.5. Control Unit***

Κώδικας Control:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Control **is** **port(**

opcode**:** **in** std\_logic\_vector**(**5 **downto** 0**);**

regdst**:** **out** std\_logic**;**

ALUsrc**:** **out** std\_logic**;**

memtoreg**:** **out** std\_logic**;**

regwrite**:** **out** std\_logic**;**

memread**:** **out** std\_logic**;**

memwrite**:** **out** std\_logic**;**

branch**:** **out** std\_logic**;**

ALUop**:** **out** std\_logic\_vector**(**1 **downto** 0**));**

**end** Control**;**

**architecture** behavioral **of** Control **is**

**begin**

**process(**opcode**)**

**begin**

--Initialization

regdst **<=** '0'**;**

ALUsrc **<=** '0'**;**

memtoreg **<=** '0'**;**

regwrite **<=** '0'**;**

memread **<=** '0'**;**

memwrite **<=** '0'**;**

branch **<=** '0'**;**

ALUop **<=** "00"**;**

**case** opcode **is**

--Type R instr.

**when** "000000" **=>**

regdst **<=** '1'**;**

ALUsrc **<=** '0'**;**

memtoreg **<=** '0'**;**

regwrite **<=** '1'**;**

ALUop **<=** "10"**;**

--addi

**when** "001000" **=>**

regdst **<=** '0'**;**

ALUsrc **<=** '1'**;**

memtoreg **<=** '0'**;**

regwrite **<=** '1'**;**

ALUop **<=** "00"**;**

--lw

**when** "100011" **=>**

regdst **<=** '0'**;**

ALUsrc **<=** '1'**;**

memtoreg **<=** '1'**;**

regwrite **<=** '1'**;**

memread **<=** '1'**;**

ALUop **<=** "00"**;**

--sw

**when** "101011" **=>**

ALUsrc **<=** '1'**;**

memwrite **<=** '1'**;**

ALUop **<=** "00"**;**

--bne

**when** "000101" **=>**

branch **<=** '1'**;**

ALUop **<=** "01"**;**

**when** **others** **=>**

**null;**

**end** **case;**

**end** **process;**

**end** behavioral**;**

Η μονάδα ελέγχου είναι υπεύθυνη να διερμηνεύει τις εντολές, αναλύει τον *opcode* τους και να καθορίζει ποια σήματα θα ενεργοποιηθούν στον επεξεργαστή για την εκτέλεση της κάθε εντολής. Εδώ ξεκινάμε με μία αρχικοποίηση και συνεχίζουμε με τον ρητό καθορισμό των σημάτων που θα ενεργοποιηθούν για τις εντολές **add,** **sub,** addi**,** lw**,** swκαι bne.

Στο testbench εισάγουμε τα opcodes τριών εντολών και θέλουμε να δούμε εάν η μονάδα ελέγχου θα θέσει στα σήματα τις σωστές τιμές.

Κώδικας Control tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Control\_tb **is**

**end** Control\_tb**;**

**architecture** behavior **of** Control\_tb **is**

**signal** opcode**:** std\_logic\_vector**(**5 **downto** 0**);**

**signal** regdst**:** std\_logic**;**

**signal** ALUsrc**:** std\_logic**;**

**signal** memtoreg**:** std\_logic**;**

**signal** regwrite**:** std\_logic**;**

**signal** memread**:** std\_logic**;**

**signal** memwrite**:** std\_logic**;**

**signal** branch**:** std\_logic**;**

**signal** ALUop**:** std\_logic\_vector**(**1 **downto** 0**);**

**begin**

uut**:** **entity** work**.**Control

**port** **map(**

opcode **=>** opcode**,**

regdst **=>** regdst**,**

ALUsrc **=>** ALUsrc**,**

memtoreg **=>** memtoreg**,**

regwrite **=>** regwrite**,**

memread **=>** memread**,**

memwrite **=>** memwrite**,**

branch **=>** branch**,**

ALUop **=>** ALUop**);**

**process**

**begin**

--addi $1, $0, 4

opcode **<=** "001000"**;**

**wait** **for** 10 ns**;**

--sw $6, $4

opcode **<=** "101011"**;**

**wait** **for** 10 ns**;**

--bne $5, $0, L1

opcode **<=** "000101"**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

* Βλέπουμε ότι για την εντολή addi $1**,** $0**,** 4 (opcode 00100), έχουμε τα επιθμητά σήματα που θέσαμε στον κώδικα.

A screen shot of a computer

AI-generated content may be incorrect.

* Το ίδιο ισχύει και για την εντολή sw $6**,** $4 (opcode 101011).

A screenshot of a computer program

AI-generated content may be incorrect.

* Παρομοίως και για την bne $5**,** $0**,** L1 (opcode 000101).

A screenshot of a computer program

AI-generated content may be incorrect.

***1.1.6. ALU Control***

Κώδικας ALUcontrol:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** ALUcontrol **is** **port(**

funct**:** **in** std\_logic\_vector**(**5 **downto** 0**);**

ALUop**:** **in** std\_logic\_vector**(**1 **downto** 0**);**

ALUcontrol**:** **out** std\_logic\_vector**(**3 **downto** 0**));**

**end** ALUcontrol**;**

**architecture** behavioral **of** ALUcontrol **is**

**begin**

**process(**funct**,** ALUop**)**

**begin**

**case** ALUop **is**

--lw, sw, addi

**when** "00" **=>** ALUcontrol **<=** "0010"**;**

--bne

**when** "01" **=>** ALUcontrol **<=** "0110"**;**

--R type

**when** "10" **=>**

**case** funct **is**

**when** "100000" **=>** ALUcontrol **<=** "0010"**;**

**when** "100010" **=>** ALUcontrol **<=** "0110"**;**

**when** **others** **=>** ALUcontrol **<=** "1111"**;**

**end** **case;**

**when** **others** **=>** ALUcontrol **<=** "1111"**;**

**end** **case;**

**end** **process;**

**end** behavioral**;**

Η μονάδα ελέγχου της ALU λαμβάνει το σήμα ALUOp από την κεντρική μονάδα ελέγχου και με βάση το σήμα αυτό, «λέει» στην ALU τι πράξη πρέπει να εκτελέσει.

Ο παραπάνω κώδικας υλοποιεί ακριβώς αυτό. Για τις εντολές R-Type όμως (**add,** **sub**), χρειάζεται να ληφθεί υπόψιν και άλλη μία τιμή, η *funct*, η οποία θα καθορίσει το εξερχόμενο σήμα της μονάδας ALU control.

Στον έλεγχο του testbench δοκιμάζουμε την έξοδο για διάφορες τιμές του *funct*, με βάση την εκφώνηση.

Κώδικας ALUcontrol tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** ALUcontrol\_tb **is**

**end** ALUcontrol\_tb**;**

**architecture** behavior **of** ALUcontrol\_tb **is**

**signal** funct**:** std\_logic\_vector**(**5 **downto** 0**);**

**signal** ALUop**:** std\_logic\_vector**(**1 **downto** 0**);**

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

**begin**

uut**:** **entity** work**.**ALUcontrol **port** **map(**

funct **=>** funct**,**

ALUop **=>** ALUop**,**

ALUcontrol **=>** ALUcontrol**);**

**process**

**begin**

funct **<=** "100000"**;**

ALUop **<=** "10"**;**

**wait** **for** 10 ns**;**

funct **<=** "100010"**;**

ALUop **<=** "10"**;**

**wait** **for** 10 ns**;**

funct **<=** "111111"**;**

ALUop **<=** "00"**;**

**wait** **for** 10 ns**;**

funct **<=** "111111"**;**

ALUop **<=** "01"**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

Βλέπουμε ότι:

* για *funct*= 100000 & *ALUop*= 10, έχουμε *ALUcontrol*= 0010,

A screen shot of a computer

AI-generated content may be incorrect.

* για *funct*= 100010 & *ALUop*= 10, έχουμε *ALUcontrol*= 0110,

A screenshot of a computer program

AI-generated content may be incorrect.

* για *funct*= 111111 & *ALUop*= 00, έχουμε *ALUcontrol*= 0010 και

A grey screen with white text

AI-generated content may be incorrect.

* και για *funct*= 111111 & *ALUop*= 01, έχουμε *ALUcontrol=* 0110.

A grey background with white text

AI-generated content may be incorrect.

Τα αποτελέσματα συνάδουν πλήρως με τον κώδικα.

***1.1.7. Program Counter***

Κώδικας PC:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**STD\_LOGIC\_ARITH**.ALL;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

**entity** PC **is** **port(**

clk**:** **in** std\_logic**;**

reset**:** **in** std\_logic**;**

pc\_in**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

pc\_out**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** PC**;**

**architecture** behavioral **of** PC **is**

**signal** pc\_reg**:** std\_logic\_vector**(**31 **downto** 0**);**

**begin**

**process(**clk**,** reset**)**

**begin**

**if** reset **=** '1' **then**

pc\_reg **<=** **(others** **=>** '0'**);**

**elsif** **rising\_edge(**clk**)** **then**

pc\_reg **<=** pc\_in**;**

**end** **if;**

**end** **process;**

pc\_out **<=** pc\_reg**;**

**end** behavioral**;**

O Program Counter είναι υπεύθυνος να δείχνει στην επόμενη εντολή του προγράμματος που η CPU πρέπει να εκτελέσει. Δέχεται ως είσοδο μια διεύθυνση μνήμης η οποία είναι και η έξοδος του. Στο testbench δοκιμάζουμε ως είσοδο δύο τιμές και βλέπουμε αν η έξοδος του PC θα είναι σωστή.

Κώδικας PC tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** PC\_tb **is**

**end** PC\_tb**;**

**architecture** behavior **of** PC\_tb **is**

**signal** clk**:** std\_logic **:=** '0'**;**

**signal** reset**:** std\_logic **:=** '0'**;**

**signal** pc\_in**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** pc\_out**:** std\_logic\_vector**(**31 **downto** 0**);**

**constant** clk\_period**:** time **:=** 10 ns**;**

**begin**

uut**:** **entity** work**.**PC

**port** **map(**

clk **=>** clk**,**

reset **=>** reset**,**

pc\_in **=>** pc\_in**,**

pc\_out **=>** pc\_out**);**

clk **<=** **not** clk **after** clk\_period**/**2**;**

**process**

**begin**

reset **<=** '1'**;**

**wait** **for** clk\_period**;**

reset **<=** '0'**;**

pc\_in **<=** x"AAAACCCC"**;**

**wait** **for** clk\_period**;**

pc\_in **<=** x"FFFFBBBB"**;**

**wait** **for** clk\_period**;**

**wait;**

**end** **process;**

**end** behavior**;**

A screenshot of a computer

AI-generated content may be incorrect.

***1.1.8. 5MUX2to1***

Κώδικας 5Μux2to1:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Mux5\_2to1 **is** **port(**

in0**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

in1**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

set**:** **in** std\_logic**;**

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

**end** Mux5\_2to1**;**

**architecture** behavioral **of** Mux5\_2to1 **is**

**begin**

**process(**in0**,** in1**,** set**)**

**begin**

**if** set **=** '0' **then**

output **<=** in0**;**

**else** output **<=** in1**;**

**end** **if;**

**end** **process;**

**end** behavioral**;**

Ο 5-bit Πολυπλέκτης 2 σε 1 χρησιμοποιείται σε πολλαπλά σημεία του MIPS για να επιτυγχάνεται η σωστή διαδρομή για κάθε εντολή. Ως εισόδους έχει 2 αριθμούς των 5 bit και καλείται να διαλέξει έναν απ’ τους δύο. Η επιλογή γίνεται με βάση κάποιο από τα σήματα της κεντρικής μονάδας ελέγχου *Control*.

Στο testbench δοκιμάζουμε 2 ζευγάρια αριθμών με διαφορετικές τιμές σήματος επιλογής για να επιβεβαιώσουμε ότι θα κάνει την σωστή επιλογή.

Κώδικας 5Mux2to1 tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Mux5\_2to1\_tb **is**

**end** Mux5\_2to1\_tb**;**

**architecture** behavior **of** Mux5\_2to1\_tb **is**

**signal** in0**:** std\_logic\_vector**(**4 **downto** 0**);**

**signal** in1**:** std\_logic\_vector**(**4 **downto** 0**);**

**signal** set**:** std\_logic**;**

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

**begin**

uut**:** **entity** work**.**Mux5\_2to1 **port** **map(**

in0 **=>** in0**,**

in1 **=>** in1**,**

set **=>** set**,**

output **=>** output**);**

**process**

**begin**

in0 **<=** "11100"**;** --0x1C

in1 **<=** "01010"**;** --0x0A

set **<=** '1'**;**

**wait** **for** 10 ns**;**

in0 **<=** "11100"**;**

in1 **<=** "01011"**;**

set **<=** '0'**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

A screenshot of a computer

AI-generated content may be incorrect.

***1.1.9. Sign Extension Unit***

Κώδικας Signextension:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Signextension **is** **port(**

in16**:** **in** std\_logic\_vector**(**15 **downto** 0**);**

out32**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** Signextension**;**

**architecture** behavioral **of** Signextension **is**

**begin**

**process(**in16**)**

**variable** tmp**:** signed**(**31 **downto** 0**);**

**begin**

tmp**:=** **resize(**signed**(**in16**),**32**);**

out32 **<=** std\_logic\_vector**(**tmp**);**

**end** **process;**

**end** behavioral**;**

Κώδικας Signextension tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Signextension\_tb **is**

**end** Signextension\_tb**;**

**architecture** behavior **of** Signextension\_tb **is**

**signal** in16**:** std\_logic\_vector**(**15 **downto** 0**);**

**signal** out32**:** std\_logic\_vector**(**31 **downto** 0**);**

**begin**

uut**:** **entity** work**.**Signextension **port** **map(**

in16 **=>** in16**,**

out32 **=>** out32**);**

**process**

**begin**

in16 **<=** x"FAAA"**;**

**wait** **for** 10 ns**;**

in16 **<=** x"AFFF"**;**

**wait** **for** 10 ns**;**

in16**<=** x"5444"**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

Η λειτουργία της μονάδας επέκτασης προσήμου είναι αρκετά απλή. Έχει ως είσοδο μια τιμή 16 bits και καλείται να την επεκτείνει στα 32 bits.

A screenshot of a computer

AI-generated content may be incorrect.

***1.1.10. 32MUX2to1***

Κώδικας 32Mux2to1:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Mux32\_2to1 **is** **port(**

in0**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

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

set**:** **in** std\_logic**;**

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

**end** Mux32\_2to1**;**

**architecture** behavioral **of** Mux32\_2to1 **is**

**begin**

**process(**in0**,** in1**,** set**)**

**begin**

**if** set **=** '0' **then**

output **<=** in0**;**

**else** output **<=** in1**;**

**end** **if;**

**end** **process;**

**end** behavioral**;**

Ο 32bit Πολυπλέκτης 2 σε 1 έχει ακριβώς την ίδια λειτουργία με τον Πολυπλέκτη 5 bit παραπάνω, με την μόνη διαφορά ότι ως εισόδους έχει δύο αριθμούς των 32 bits αντί των 5 bits.

Κώδικας 32Mux2to1 tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Mux32\_2to1\_tb **is**

**end** Mux32\_2to1\_tb**;**

**architecture** behavior **of** Mux32\_2to1\_tb **is**

**signal** in0**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** in1**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** set**:** std\_logic**;**

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

**begin**

uut**:** **entity** work**.**Mux32\_2to1 **port** **map(**

in0 **=>** in0**,**

in1 **=>** in1**,**

set **=>** set**,**

output **=>** output**);**

**process**

**begin**

in0 **<=** x"CCCCCCCC"**;**

in1 **<=** x"BBBBBBBB"**;**

set **<=** '1'**;**

**wait** **for** 10 ns**;**

in0 **<=** x"CCCCCCCC"**;**

in1 **<=** x"BBBBBBBB"**;**

set **<=** '0'**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

A screenshot of a computer

AI-generated content may be incorrect.

***1.1.11. Shift Left 2***

Κώδικας Leftshift:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Leftshift **is** **port(**

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

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

**end** Leftshift**;**

**architecture** behavioral **of** Leftshift **is**

**begin**

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

**end** behavioral**;**

Κώδικας Leftshift tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Leftshift\_tb **is**

**end** Leftshift\_tb**;**

**architecture** behavior **of** Leftshift\_tb **is**

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

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

**begin**

uut**:** **entity** work**.**Leftshift **port** **map(**

input **=>** input**,**

output **=>** output**);**

**process**

**begin**

input **<=** x"0FFFFFFF"**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

H μονάδα αριστερής ολίσθησης κατά 2 είναι επίσης μία αρκετά απλή μονάδα. Είναι υπεύθυνη, όπως δηλώνει το όνομά της, για την ολίσθηση προς τα αριστερά ενός αριθμού 32 bit. Στο testbench δοκιμάζουμε αυτή τη λειτουργία με μία τιμή.

A computer screen shot of a black screen

AI-generated content may be incorrect.

***1.1.12. 32-bit Adder***

Κώδικας Adder32:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** Adder32 **is** **port(**

in0**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

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

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

**end** Adder32**;**

**architecture** behavioral **of** Adder32 **is**

**begin**

result **<=** std\_logic\_vector**(**unsigned**(**in0**)+**unsigned**(**in1**));**

**end** behavioral**;**

Κώδικας Adder32 tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** Adder32\_tb **is**

**end** Adder32\_tb**;**

**architecture** behavior **of** Adder32\_tb **is**

**signal** in0**,** in1**,** result**:** std\_logic\_vector**(**31 **downto** 0**);**

**begin**

uut**:** **entity** work**.**Adder32 **port** **map(**

in0 **=>** in0**,**

in1 **=>** in0**,**

result **=>** result**);**

**process**

**begin**

in0 **<=** x"CCCCCCCC"**;**

in1 **<=** x"BBBBBBBB"**;**

**wait** **for** 10 ns**;**

in0 **<=** x"BBBBBBBB"**;**

in1 **<=** x"55555556"**;**

**wait** **for** 10 ns**;**

**wait;**

**end** **process;**

**end** behavior**;**

Ο αθροιστής αναλαμβάνει την πρόσθεση δύο αριθμών, που αποτελούν και τις εισόδους του, και το άθροισμα τους αποτελεί την μοναδική του έξοδο. Για τους σκοπούς της εργασίας δεν λαμβάνεται υπόψιν η υπερχείλιση.

A screenshot of a computer

AI-generated content may be incorrect.

**Μέρος Τρίτο: MIPS**

Κώδικας MIPS:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** MIPS **is** **port(**

clock**:** **in** std\_logic**;**

reset**:** **in** std\_logic**);**

**end** MIPS**;**

**architecture** behavioral **of** MIPS **is**

**component** ALU **port(**

ALUin1**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

ALUin2**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

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

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

zero**:** **out** std\_logic**);**

**end** **component;**

**component** Registerfile **port(**

clk**:** **in** std\_logic**;**

reset**:** **in** std\_logic**;**

read\_reg1**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

read\_reg2**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

write\_reg**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

write\_data**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

write\_enable**:** **in** std\_logic**;**

read\_data1**:** **out** std\_logic\_vector**(**31 **downto** 0**);**

read\_data2**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** **component;**

**component** Datamem **port(**

clk**:** **in** std\_logic**;**

mem\_write**:** **in** std\_logic**;**

mem\_read**:** **in** std\_logic**;**

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

write\_data**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

read\_data**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** **component;**

**component** Imem **port(**

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

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

**end** **component;**

**component** Control **port(**

opcode**:** **in** std\_logic\_vector**(**5 **downto** 0**);**

regdst**:** **out** std\_logic**;**

ALUsrc**:** **out** std\_logic**;**

memtoreg**:** **out** std\_logic**;**

regwrite**:** **out** std\_logic**;**

memread**:** **out** std\_logic**;**

memwrite**:** **out** std\_logic**;**

branch**:** **out** std\_logic**;**

ALUop**:** **out** std\_logic\_vector**(**1 **downto** 0**));**

**end** **component;**

**component** ALUcontrol **port(**

funct**:** **in** std\_logic\_vector**(**5 **downto** 0**);**

ALUop**:** **in** std\_logic\_vector**(**1 **downto** 0**);**

ALUcontrol**:** **out** std\_logic\_vector**(**3 **downto** 0**));**

**end** **component;**

**component** PC **port(**

clk**:** **in** std\_logic**;**

reset**:** **in** std\_logic**;**

pc\_in**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

pc\_out**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** **component;**

**component** Mux5\_2to1 **port(**

in0**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

in1**:** **in** std\_logic\_vector**(**4 **downto** 0**);**

set**:** **in** std\_logic**;**

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

**end** **component;**

**component** Signextension **port(**

in16**:** **in** std\_logic\_vector**(**15 **downto** 0**);**

out32**:** **out** std\_logic\_vector**(**31 **downto** 0**));**

**end** **component;**

**component** Mux32\_2to1 **port(**

in0**:** **in** std\_logic\_vector**(**31 **downto** 0**);**

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

set**:** **in** std\_logic**;**

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

**end** **component;**

--Η μονάδα Leftshift παραλείπεται

--Η μονάδα Adder32 δεν προστίθεται ως component

--Control

**signal** RegDst**:** std\_logic**;**

**signal** Branch**:** std\_logic**;**

**signal** MemRead**:** std\_logic**;**

**signal** MemToReg**:** std\_logic**;**

**signal** ALUOp**:** std\_logic\_vector**(**1 **downto** 0**);**

**signal** MemWrite**:** std\_logic**;**

**signal** ALUSrc**:** std\_logic**;**

**signal** RegWrite**:** std\_logic**;**

--PC

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

--ALU

**signal** Zero**:** std\_logic**;**

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

--Register File

**signal** RegOut1**:** std\_logic\_vector**(**31 **downto** 0**);**

**signal** RegOut2**:** std\_logic\_vector**(**31 **downto** 0**);**

--MUX

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

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

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

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

**signal** Branch\_isTrue**:** std\_logic**;**

--ALU Control

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

--Sign Extend

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

--Data Mem

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

--Instruction Mem

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

--Adder32

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

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

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

**begin**

ONE **<=** std\_logic\_vector**(to\_unsigned(**1**,**32**));**

FA\_PC**:** **entity** work**.**Adder32 **port** **map(**

in0 **=>** PCOut**,**

in1 **=>** ONE**,**

result **=>** AddressNoBranch**);**

FA\_Branch**:** **entity** work**.**Adder32 **port** **map(**

in0 **=>** AddressNoBranch**,**

in1 **=>** SignExOut**,**

result **=>** AddressBranch**);**

Prog\_Counter**:** PC **port** **map(**

pc\_in **=>** MUXtoPC**,**

pc\_out **=>** PCOut**,**

clk **=>** clock**,**

reset **=>** reset**);**

Instr\_Mem**:** Imem **port** **map(**

address **=>** PCout**,**

instruction **=>** IMOut**);**

RegFile**:** Registerfile **port** **map(**

read\_reg1 **=>** IMOut**(**25 **downto** 21**),**

read\_reg2 **=>** IMOut**(**20 **downto** 16**),**

write\_reg **=>** MUXtoReg**,**

write\_data **=>** MUXtoRegWrite**,**

write\_enable **=>** RegWrite**,**

read\_data1 **=>** RegOut1**,**

read\_data2 **=>** RegOut2**,**

clk **=>** clock**,**

reset **=>** reset**);**

Ctrl**:** Control **port** **map(**

opcode **=>** IMOut**(**31 **downto** 26**),**

regdst **=>** RegDst**,**

ALUsrc **=>** ALUSrc**,**

memtoreg **=>** MemToReg**,**

regwrite **=>** RegWrite**,**

memread **=>** MemRead**,**

memwrite **=>** MemWrite**,**

branch **=>** Branch**,**

ALUop **=>** ALUOp**);**

ALUctrl**:** ALUcontrol **port** **map(**

funct **=>** IMOut**(**5 **downto** 0**),**

ALUop **=>** ALUOp**,**

ALUcontrol **=>** ALUctrl\_out**);**

ALU1**:** ALU **port** **map(**

ALUin1 **=>** RegOut1**,**

ALUin2 **=>** MUXtoALU**,**

ALUctrl **=>** ALUctrl\_out**,**

ALUresult **=>** ALUout**,**

zero **=>** Zero**);**

Data\_Mem**:** Datamem **port** **map(**

mem\_write **=>** MemWrite**,**

mem\_read **=>** MemRead**,**

address **=>** ALUout**,**

write\_data **=>** RegOut2**,**

read\_data **=>** DataMemOut**,**

clk **=>** clock**);**

MUXreg**:** Mux5\_2to1 **port** **map(**

in0 **=>** IMOut**(**20 **downto** 16**),**

in1 **=>** IMout**(**15 **downto** 11**),**

set **=>** RegDst**,**

output **=>** MUXtoReg**);**

MUXalu\_in**:** Mux32\_2to1 **port** **map(**

in0 **=>** RegOut2**,**

in1 **=>** SignExOut**,**

set **=>** ALUSrc**,**

output **=>** MUXtoALU**);**

MUXdatamem**:** Mux32\_2to1 **port** **map(**

in1 **=>** DataMemOut**,**

in0 **=>** ALUout**,**

set **=>** MemToReg**,**

output **=>** MUXtoRegWrite**);**

Branch\_isTrue **<=** Branch **AND** (**not** Zero)**;**

MUXaddress**:** Mux32\_2to1 **port** **map(**

in0 **=>** AddressNoBranch**,**

in1 **=>** AddressBranch**,**

set **=>** Branch\_isTrue**,**

output **=>** MUXtoPC**);**

Sign\_Ext**:** Signextension **port** **map(**

in16 **=>** IMOut**(**15 **downto** 0**),**

out32 **=>** SignExOut**);**

**end** behavioral**;**

Κώδικας MIPS tb:

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** MIPS\_tb **is**

**end** MIPS\_tb**;**

**architecture** behavior **of** MIPS\_tb **is**

**signal** clock**:** std\_logic **:=** '0'**;**

**signal** reset**:** std\_logic **:=** '0'**;**

--200MHz => 5 ns

**constant** clk\_period**:** time **:=** 5 ns**;**

**begin**

uut**:** **entity** work**.**MIPS **port** **map(**

clock **=>** clock**,**

reset **=>** reset**);**

clock **<=** **not** clock **after** clk\_period**/**2**;**

**process**

**begin**

reset **<=** '1'**;**

**wait** **for** 3 ns**;**

reset **<=** '0'**;**

**wait** **for** 100 ns**;**

std**.**env**.**stop**;**

**wait;**

**end** **process;**

**end** behavior**;**

O κώδικας του MIPS αποτελείται από τα 2 σήματα που περιλαμβάνει (clock & reset) και την διασύνδεση με όλα τα εξαρτήματα που τον αποτελούν. Περιλαμβάνονται και πολλά signals τα οποία δρουν ως τα καλώδια που συνδέουν τις διάφορες εξόδους των εξαρτημάτων με τις εισόδους άλλων εξαρτημάτων.

Η μονάδα αριστερής ολίσθησης έχει παραλειφθεί για ευκολία, σύμφωνα με τις οδηγίες της άσκησης. Οι αθροιστές δεν έχουν δηλωθεί στα components αλλά εισάγονται απευθείας ως entity γιατί αλλιώς είχαμε *“No default binding for component instance”* error.

Έχοντας τις εντολές hardcoded στην μνήμη εντολών, το testbench είναι αρκετά απλό και μικρό. Το ρολόι του συστήματος υπολογίστηκε στα 5 nanoseconds, με δεδομένο από την άσκηση 200 MHz συχνότητα:

***Περίοδος = 1/ 200\* 10^6 = 5\* 10^(-9) = 5 ns***

Αξίζει να σχολιαστεί ότι το σήμα επιλογής Branch\_isTrueτου MUX που επιλέγει ανάμεσα στην διεύθυνση της αμέσως επόμενης εντολής ή στην διεύθυνση της εντολής για jump, είναι αποτέλεσμα πράξης **AND** με το **αντεστραμμένο** σήμα Zero της ALU διότι εκτελούμε μόνο εντολή bne. Αυτό σημαίνει πως αν δεν αντιστρέφαμε το σήμα, το jump δεν θα συνέβαινε ποτέ. Αυτό γιατί η ALU δεν θα άναβε το σήμα Zero διότι όταν θα ελέγχονταν τα περιεχόμενα των καταχωρητών, δεν θα ήταν ίσα και άρα η διαφορά τους δεν θα ήταν μηδέν.

Τα αποτελέσματα παρουσιάζονται στις εικόνες που ακολουθούν. Όπως βλέπουμε από τον παλμό του ρολογιού, η προσομοίωση έτρεξε για 20 κύκλους.

A computer screen shot of a black and white screen

AI-generated content may be incorrect.

Eικόνα 3.1. Τα βασικά σήματα του MIPS

A screenshot of a computer

AI-generated content may be incorrect.

Εικόνα 3.2. Τα περιεχόμενα των καταχωρητών. Από τις τιμές του καταχωρητή $5, ο οποίος αποτελεί τον μετρητή της επανάληψης, καταλαβαίνουμε ότι ορθά μειώνεται από το 3 στο 0 και συνεπώς η επανάληψη τρέχει 3 φορές.

Μπορούμε επίσης να επιβεβαιώσουμε την ορθότητα της εκτέλεσης των εντολών και από τις δεκαεξαδικές τιμές που βλέπουμε στην γραμμή της Μνήμης Εντολών.

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

A computer screen shot of a computer

AI-generated content may be incorrect.8 **=>** x"14A0FFFB"**,** --bne $5, $0, L1

A screenshot of a computer

AI-generated content may be incorrect.  
Εικόνα 3.3. Τα περιεχόμενα της μνήμης δεδομένων. Εδώ βλέπουμε πως η εντολή

sw $3**,** 0**(**$4**)** έχει εκτελεστεί με επιτυχία για τις τιμές των $3 και $4 που αλλάζουν σε κάθε επανάληψη.