MULTI-CYCLE CPU PROJECT – HW1

Δανάη Καραβίτη 9918

Χριστίνα Σακελλάρη 9785

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών ΑΠΘ

2021-2022

ΕΙΣΑΓΩΓΗ

Αυτή είναι η τελική αναφορά για την εργασία ενός επεξεργαστή πολλαπλών κύκλων. Περιλαμβάνει μια περιγραφή του project, μια λίστα εντολών που έχουν υλοποιηθεί, το διάγραμμα της Finite State Machine (FSM), το σχήμα του datapath και της μονάδας controller, τις κυματομορφές, την λίστα των αρχείων που παραδόθηκαν και σχόλια για τις επιλογές σχεδίασης.

Σκοπός του project ήταν η σχεδίαση ενός επεξεργαστή πολλαπλών κύκλων σε Verilog HDL. Τα δύο κύρια στοιχεία της εργασίας είναι ο controller και το datapath. Επειδή πρόκειται για CPU πολλαπλών κύκλων, ο αριθμός των κύκλων ρολογιού δεν είναι σταθερός και εξαρτάται από το ποια εντολή λαμβάνεται από τη μνήμη. Δεν έχει δρομολογηθεί, επομένως θα επεξεργάζεται μία εντολή κάθε φορά. Η συντομότερη εντολή είναι το NOP και μια από τις μεγαλύτερες εντολές είναι το LW. Επίσης χρησιμοποιήθηκαν καταχωρητές για να κρατούν τις ενδιάμεσες τιμές μεταξύ των σταδίων.

Παρακάτω παρουσιάζεται λίστα με τις εντολές που έχουν υλοποιηθεί και τα αντίστοιχα opcodes και func τους. Το Opcode έχει μήκος 6 bit. Σύμφωνα με το opcode κάθε εντολής ορίζονται και τα 4bits που μπαίνουν σαν input στον έλεγχο της ALU. Ο τελικός στόχος αυτού του project είναι να λάβει ένα σύνολο εντολών από το αρχείο rom.data, να υλοποιήσει την κάθε εντολή και να χρησιμοποιήσει όπου χρειάζεται την μνήμη.

Οι εντολές που μας ζητήθηκε να υλοποιήσουμε έχουν τους παρακάτω τύπους-format:

6bits	**5bits**	**5bits**	**5bits**	**5bits**	**6bits**
Opcode	Rs	Rd	Rt	Not-used	Func

6bits	**5bits**	**5bits**	**16bits**
Opcode	Rs	Rd	Immediate

Ταξινόμηση εντολών ανά type:

- R-Type (opcode = 100000)
    - ADD (func = 110000)
    - SUB (func = 110001)
    - AND (func = 110010)
    - NOT (func = 110100)
    - OR (func = 110011)
    - SRA (func = 111000)
    - SLL (func = 111001)
    - SRL (func = 111010)
    - ROL (func = 111100)
    - ROR (func = 111101)

- I-Type
    - LI (opcode = 111000)
    - LUI (opcode = 111001)
    - ADDI (opcode = 110000)
    - ANDI (opcode = 110010)
    - ORI (opcode = 110011)
- B-Type
    - B (opcode = 111111)
    - BEQ (opcode = 000000)
    - BNE (opcode = 000001)
    - LB (opcode = 000011)
    - SB (opcode = 000111)
    - LW (opcode = 001111)
    - SW (opcode = 011111)
- NOP (Instr = 32'b0)

Part 1

Σε αυτό το μέρος ζητήθηκε η υλοποίηση μιας μονάδας αριθμητικών και λογικών πράξεων και ενός αρχείου καταχωρητών. Πρώτα, ορίσαμε τις εισόδους, τις εξόδους και κάποιες επιπλέον μεταβλητές τύπου reg για να αποθηκεύονται οι τιμές τους. Έπειτα υλοποιήσαμε τον κώδικα για την ALU, χρησιμοποιώντας μια συνάρτηση always και μια case.

Στην συνέχεια δημιουργήσαμε ένα module για έναν καταχωρητή 32bits, έτσι ώστε όταν το input WE είναι 1, να πραγματοποιείται εγγραφή στον καταχωρητή και εάν το input RESET είναι 1 (ανεξαρτήτως της τιμής του clk) να μηδενίζει ο καταχωρητής.

Για τον σχεδιασμό του RF, καλέσαμε 32 φορές το module του 32bit register, δημιουργήσαμε έναν decoder και 2 mux. Σε αυτό το στάδιο, ανάλογα με την είσοδο Awr ενεργοποιείται η εγγραφή του εκάστοτε καταχωρητή. Έπειτα μέσω των εισόδων Ard1 και Ard2, ελέγχονται οι έξοδοι του RF, δηλαδή από ποιόν καταχωρητή θέλουμε να διαβάσουμε την τιμή. Για την υλοποιήση του RF έπρεπε να προσέξουμε, όλοι οι καταχωρητές να έχουν κοινή είσοδο δεδομένων και κοινό clk, αλλά επίσης ο καταχωρητής R0 έχει πάντα την τιμή 0, αφού σκόπιμα έχουμε θέσει την είσοδο WE του σε 0.

Part 2

Σκοπός αυτού του μέρους ήταν η υλοποιήση των παρακάτω βαθμίδων:

- Βαθμίδα ανάκλασης εντολών (IFSTAGE): Ακολουθήσαμε την εικόνα του datapath που μας δόθηκε και έτσι δημιουργήσαμε έναν fulladder όπου κάνει PC+4+Immed, έναν pcadder όπου κάνει PC+4, έναν καταχωρητή PC_Reg, μια μνήμη IMEM και ένα mux3to1. Στο mux του σχήματος επιλέξαμε να προσθέσουμε ακόμα μία είσοδο, ώστε να παίρνει την αρχική τιμή 0 και να διαβάζει την πρώτη εντολή από την μνήμη και το αρχείο rom.data. Ακόμα, χρησιμοποιήσαμε τα [11:2] bits του PC_out για είσοδο του IMEM γιατί έπρεπε να δίνει 1 εντολή ανά 1 αλλαγή στην διεύθυνση της μνήμης.
- Βαθμίδα αποκωδικοποίησης (DECSTAGE): Σε αυτό το βήμα υλοποιήσαμε ένα αρχείο καταχωρητών RF, έναν mux2to1, έναν mux3to1 και ένα module cloud που απεικονίζεται στο διάγραμμα ως σύννεφο. Αρχικά, επιλέξαμε να χρησιμοποιήσουμε ένα mux3to1 παρόλο που στην εικόνα μας δόθηκε mux2to1, διότι θέλαμε να παίρνει σαν είσοδο το ALU_out, το MEM_out και το MEM_out_ZeroFill, ώστε να μπορούμε να δώσουμε το κατάλληλο

RF_Data_Sel ανάλογα με το opcode της εντολής. Επίσης, η μονάδα που απεικονίζεται ως σύννεφο δέχεται σαν είσοδο 16bits (Immediate) και τα μετατρέπει σε σήμα 32bits, ελέγχει το opcode της εντολής και σύμφωνα με αυτό κάνει shiftLeft κατά 2 ή όχι και επίσης κάνει zeroFill ή signExtension.

- Βαθμίδα εκτέλεσης πράξεων (ALUSTAGE): Σε αυτή την βαθμίδα υλοποιήσαμε ένα mux2to1 και καλέσαμε το ALU module που είχε δημιουργηθεί στο part1, όπως φαίνεται στο σχήμα. Επίσης, κάναμε έναν έλεγχο με την χρήση always και case για το opcode κάθε εντολής ώστε να αντιστοιχίσουμε το ALU_func ανάλογα με την πράξη που θέλουμε να γίνει.
- Βαθμίδα μνήμης RAM (MEMSTAGE): Τέλος για αυτή την βαθμίδα χρησιμοποιήθηκε ο κώδικας για την δημιουργία της μνήμης MEM. Προσθέσαμε, επίσης, ένα mux2to1 που παίρνει σαν είσοδο το RF_B και το RF_B_ZeroFill, ώστε να μπορούμε μέσω του controller να δίνουμε την κατάλληλη τιμή στο MUX_MEM_SEL ανάλογα με την εντολή που έχουμε.

Part 3

Σκοπός του 3ου μέρους ήταν η δημιουργία ενός ενιαίου datapath, η υλοποίηση της μονάδας ελέγχου του datapath, το οποίο ονομάσαμε controller, και την ένωση αυτών των δύο σε ένα αρχείο PROCESSOR.v.

Για τον σχεδιασμό του datapath, εισάχθηκαν σαν είσοδο όλα τα σήματα τα οποία θέλαμε να ελέγξουμε από τον controller (πχ. PC_Sel, MUX_MEM_SEL κτλ.), δηλώσαμε τις απαραίτητες μεταβλητές τύπου wire και reg, και καλέσαμε τα modules IFSTAGE, DECSTAGE, ALUSTAGE και MEMSTAGE.

Όσον αφορά την υλοποιήσει του controller, ο οποίος παίρνει σαν είσοδο το opcode, το Instr, το clk, το reset, το zero και έχει σαν έξοδο όλες τις μεταβλητές που μπαίνουν σαν είσοδο στο datapath, οι οποίες είναι:

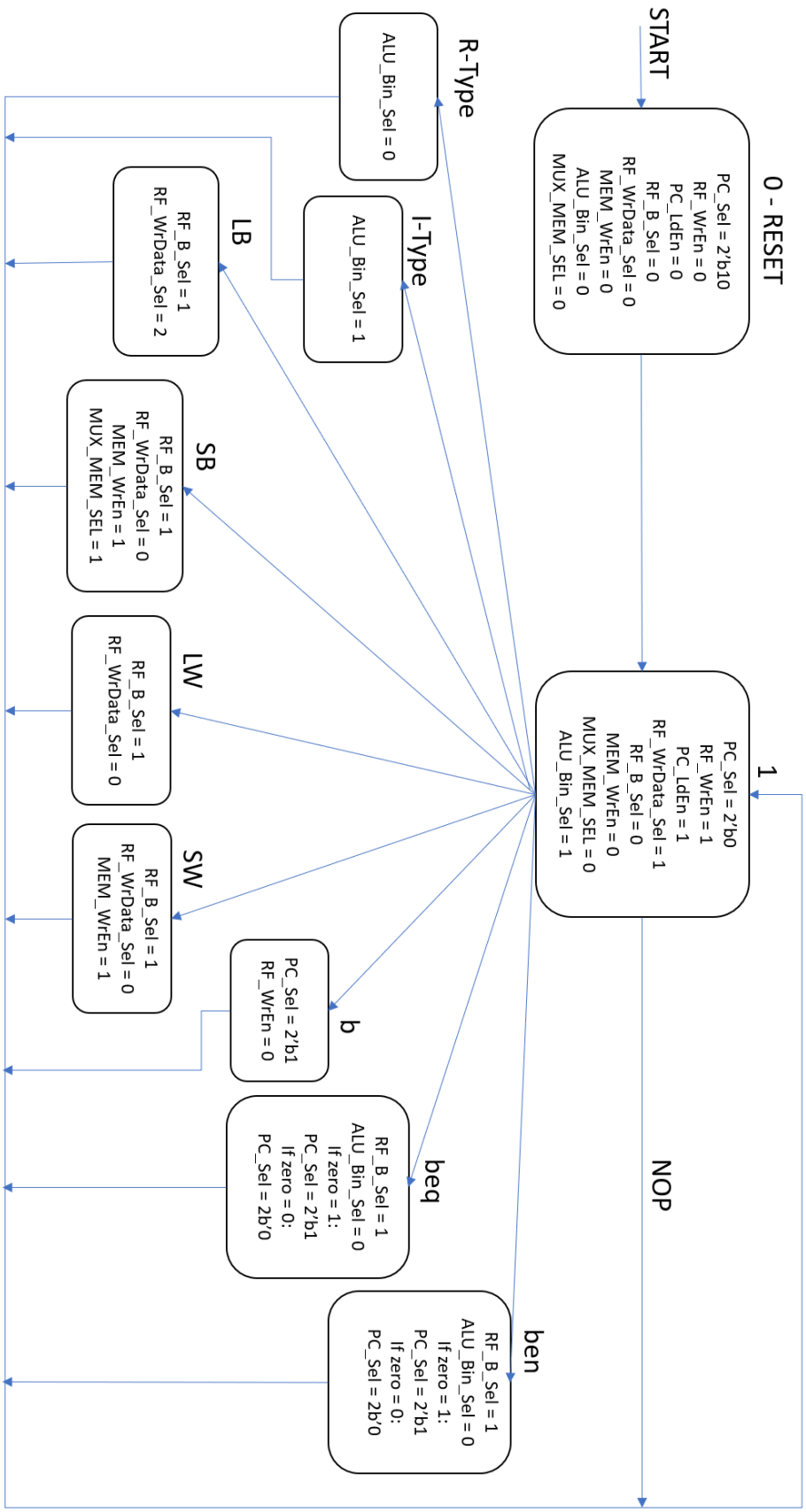
PC_Sel, PC_LdEn, RF_WrEn, RF_B_Sel, RF_WrData_Sel, ALU_Bin_Sel, MEM_WrEn, MUX_MEM_SEL

Αρχικά υλοποιήσαμε το reset, έτσι ώστε άμα το reset πάρει την τιμή 1, όλες οι μεταβλητές-έξοδοι να αρχικοποιούνται σε μηδέν. Έπειτα, ελέγχοντας το opcode διαχωρίζουμε τον τύπο της κάθε εντολής και δίνουμε τις επιθυμητές τιμές στις εξόδους, ανάλογα με την πράξη που πρέπει να γίνει στην κάθε εντολή. Για την περίπτωση των r-type εντολών, χρησιμοποιείτε και το func για να ελεγχθεί ακριβώς η εντολή, πράγμα που δεν χρειαζόταν καθώς όλες οι r-type εντολές χρειάζονται τις ίδιες τιμές εξόδων, αλλά το κάναμε αναλυτικά για να είναι πιο ξεκάθαρος και ευανάγνωστος ο κώδικας μας. Ακόμα, για την υλοποίηση του nop, ελέγχουμε την εντολή Instr, η οποία άμα είναι ίση με μηδέν, δεν καλείται κανένα στάδιο και απλά διαβάζουμε την επόμενη εντολή. Για τις εντολές beq και bne, ο έλεγχος του zero γίνεται μέσα στον controller και ανάλογα με την τιμή του, επιλέγεται εάν θα γίνει PC+4 ή PC+4+Immed.

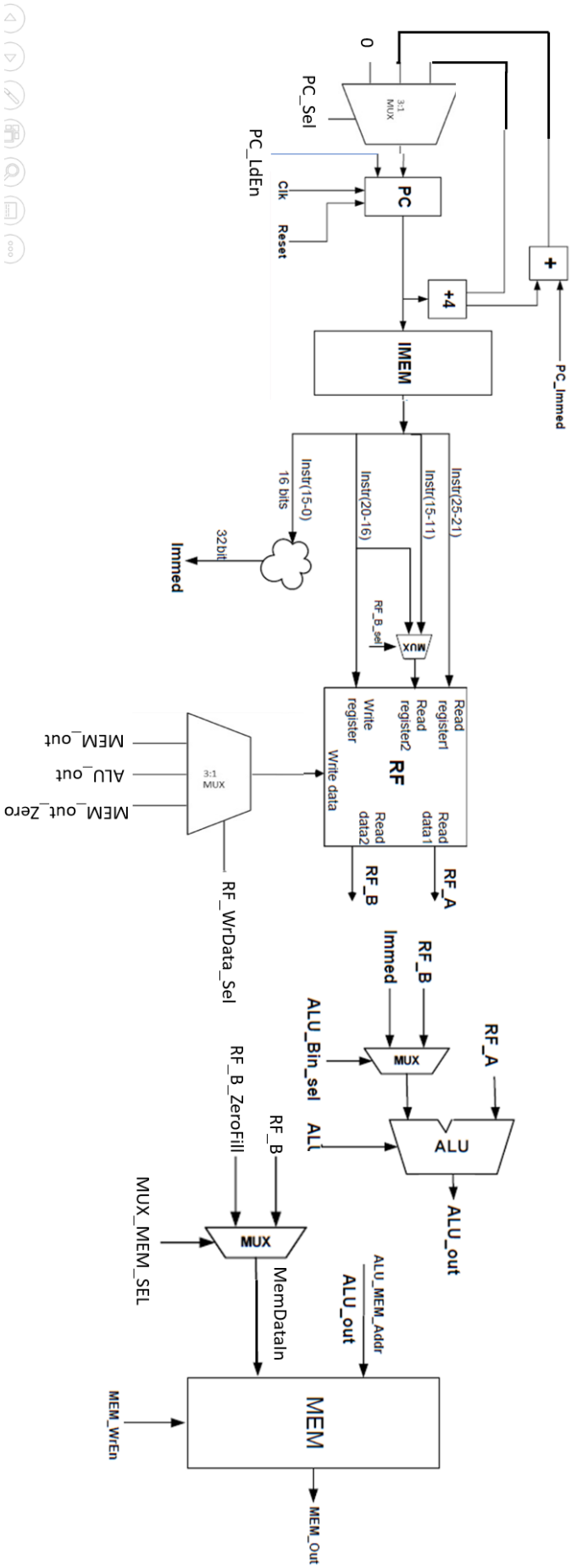
Τέλος, δημιουργήσαμε ένα πολύ απλό testbench, το οποίο λέγεται prosTB.v, όπου δίνουμε την συχνότητα του clk και το ορίζουμε να αλλάζει ανά μια περίοδο. Καθώς στην αρχή κάνουμε και ένα reset ώστε όλες οι μεταβλητές να αρχικοποιηθούν στο μηδέν.

Παρόλο που η προσομοίωση έτρεξε κανονικά και βγάζει τις περισσότερες φορές τις σωστές τιμές των καταχωρητών και των πράξεων, μετά από ένα πλήθος εντολών παρουσιάζεται ένα error το οποίο δεν καταφέραμε να διορθώσουμε και τερματίζει τον κώδικα. Ωστόσο, εάν αλλάξουμε την σειρά των εντολών στο αρχείο rom.data, υλοποιούνται κανονικά όλες οι εντολές.

FSM Diagram

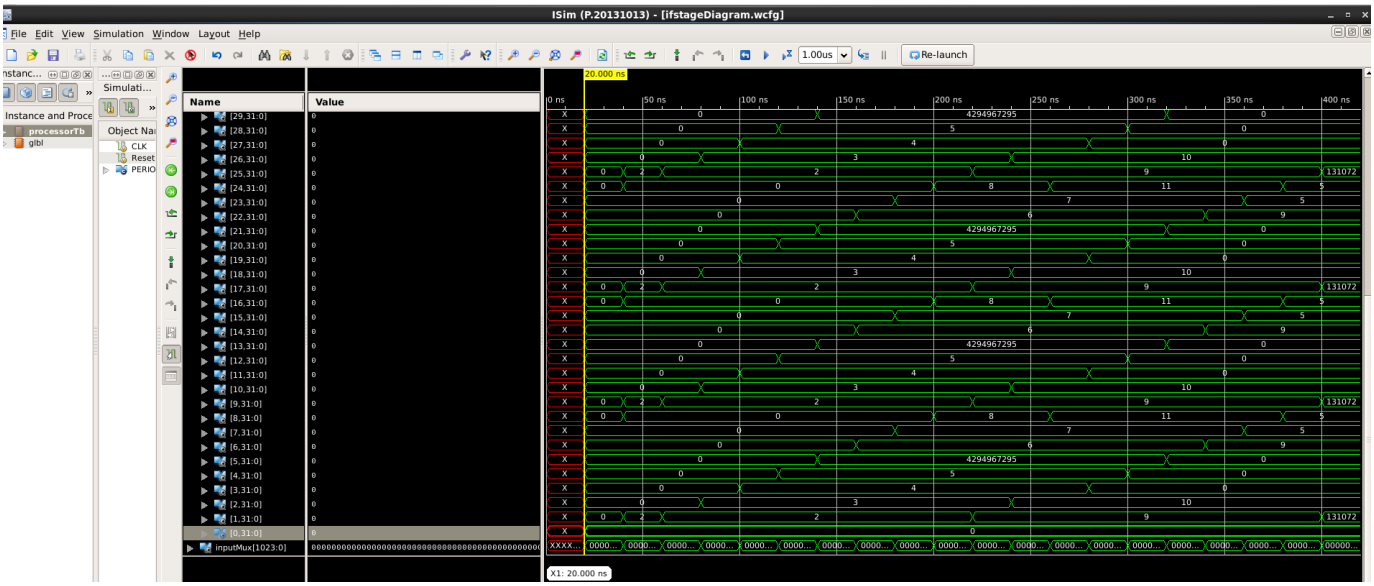

START
0 - RESET
PC_Sel = 2'b10
RF_WrEn = 0
PC_LdEn = 0
RF_B_Sel = 0
RF_WrData_Sel = 0
MEM_WrEn = 0
ALU_Bin_Sel = 0
MUX_MEM_SEL = 0
1
PC_Sel = 2'b0
RF_WrEn = 1
PC_LdEn = 1
RF_WrData_Sel = 1
RF_B_Sel = 0
MEM_WrEn = 0
MUX_MEM_SEL = 0
ALU_Bin_Sel = 1
NOP
R-Type
ALU_Bin_Sel = 0
I-Type
ALU_Bin_Sel = 1
LB
RF_B_Sel = 1
RF_WrData_Sel = 2
SB
RF_B_Sel = 1
RF_WrData_Sel = 0
MEM_WrEn = 1
MUX_MEM_SEL = 1
LW
RF_B_Sel = 1
RF_WrData_Sel = 0
SW
RF_B_Sel = 1
RF_WrData_Sel = 0
MEM_WrEn = 1
b
PC_Sel = 2'b1
RF_WrEn = 0
beq
RF_B_Sel = 1
ALU_Bin_Sel = 0
If zero = 1:
PC_Sel = 2'b1
If zero = 0:
PC_Sel = 2b'0
ben
RF_B_Sel = 1
ALU_Bin_Sel = 0
If zero = 1:
PC_Sel = 2'b1
If zero = 0:
PC_Sel = 2b'0

DATAPATH SCHEMATIC

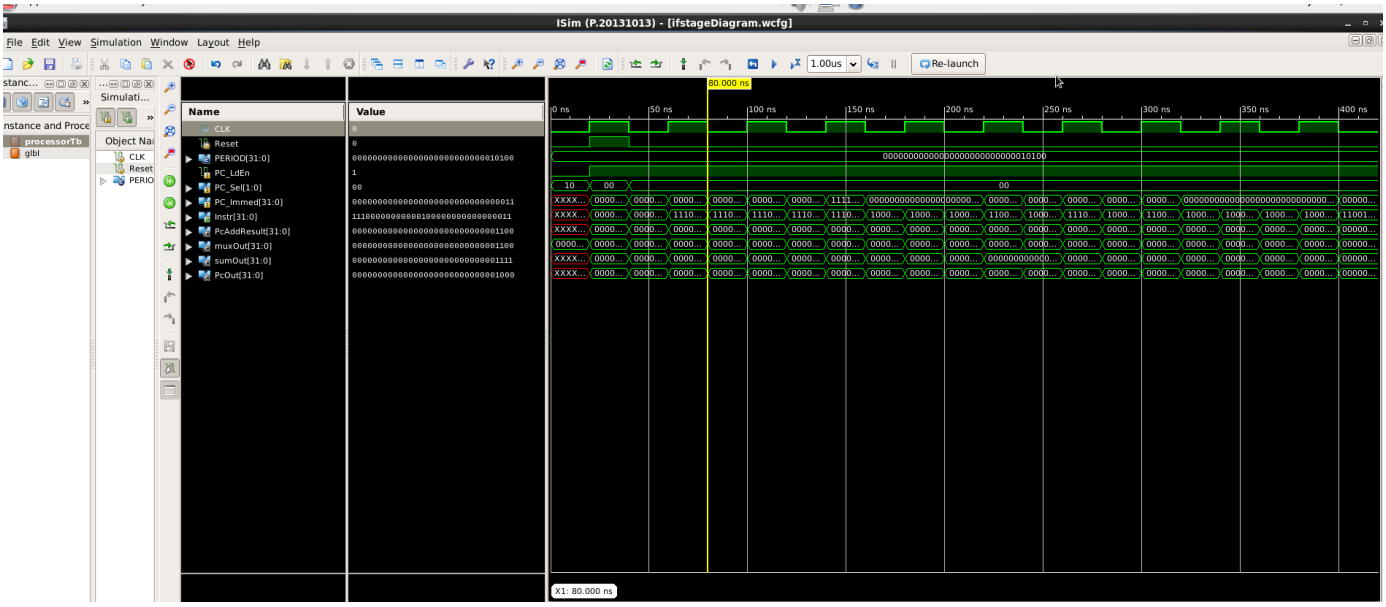


ΚΥΜΜΑΤΟΜΟΡΦΕΣ

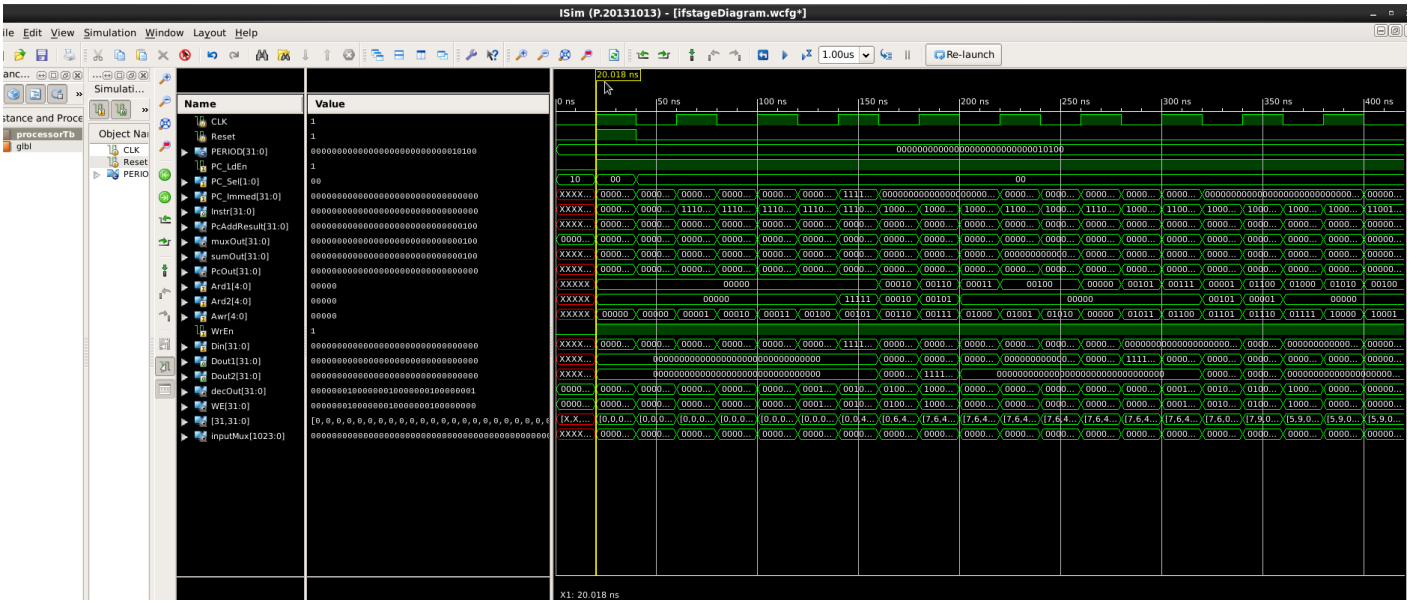
Για την προσομοίωση χρησιμοποιήθηκε το αρχείο rom.data.



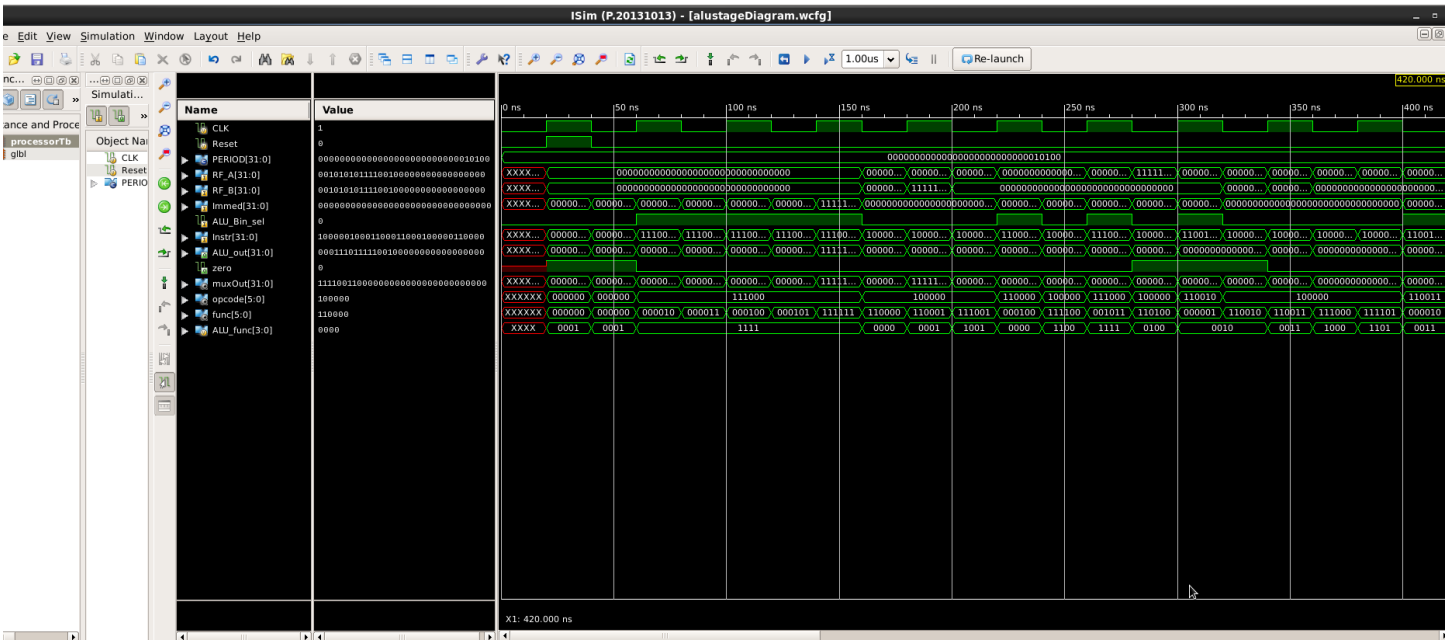
Εικόνα 1 IFSTAGE (Registers)



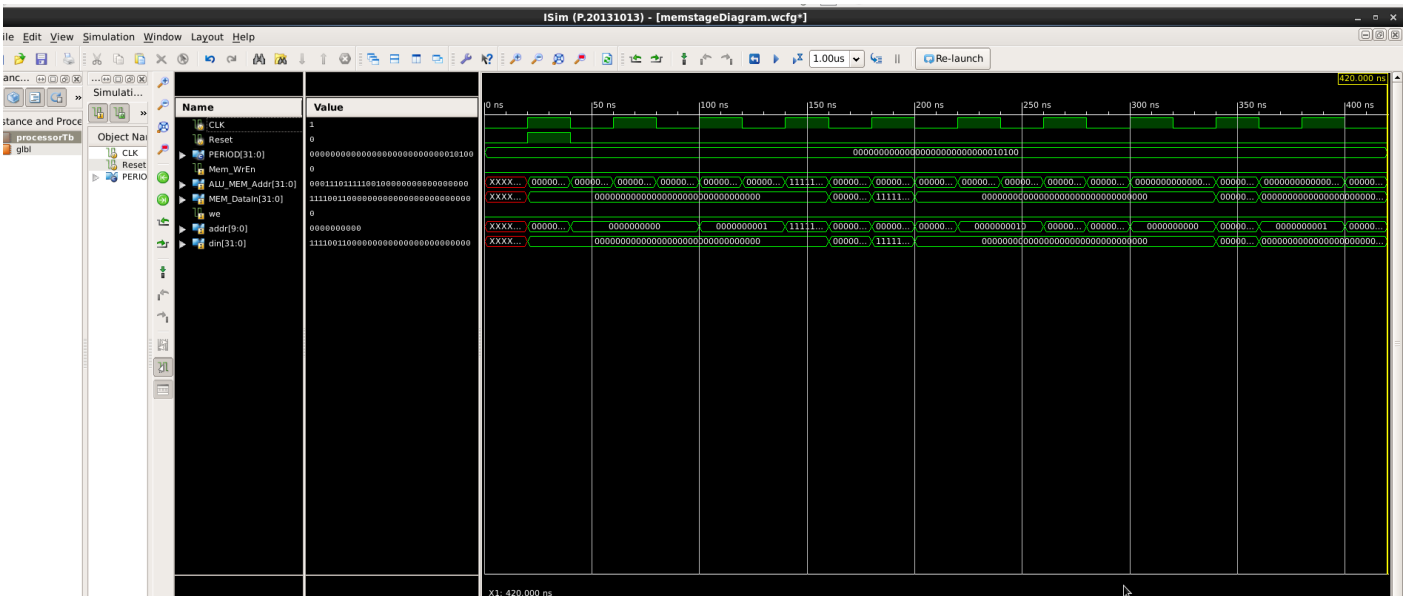
Εικόνα 2 IFSTAGE



Εικόνα 3 DECSTAGE



Εικόνα 4 ALUSTAGE



Εικόνα 5 MEMSTAGE

Όπως φαίνεται από τις εικόνες, οι τιμές των καταχωρητών τις περισσότερες φορές είναι οι επιθυμητές.

Παραδοτέα Αρχεία (22 project files)

IFSTAGE.v DECSTAGE.v MEMSTAGE.v ALUSTAGE.v datapath.v c2.v ALU.v RF2.v processorTb.v processor.v