Εργασία στα Ψηφιακά Συστήματα Hardware σε Χαμηλά Επίπεδα Λογικής Ι

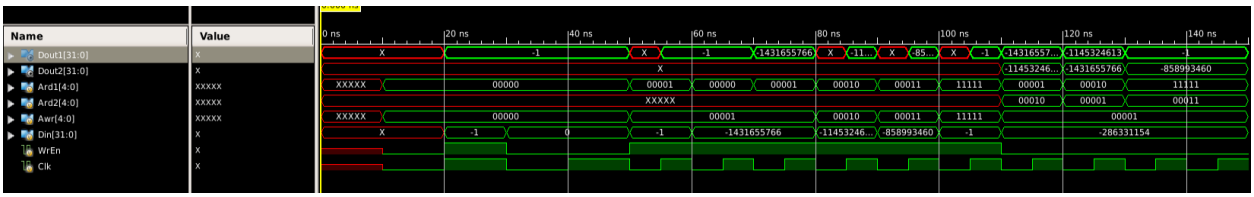
Γιαννόπουλος Νικόλαος
Α.Ε.Μ.: 9629
EMAIL: ngiannop@ece.auth.gr

Χρυσοστόμου Ιωσήφ
Α.Ε.Μ.: 9130
EMAIL: chrysosie@ece.auth.gr

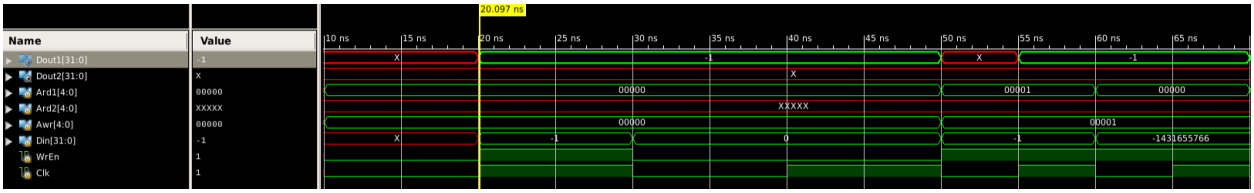
Περιεχόμενα

Σχεδιασμός και υλοποίηση του RegisterFile

Τώρα θα δούμε τα testbench του RF όπου φαίνονται στην *Εικόνα 1.*

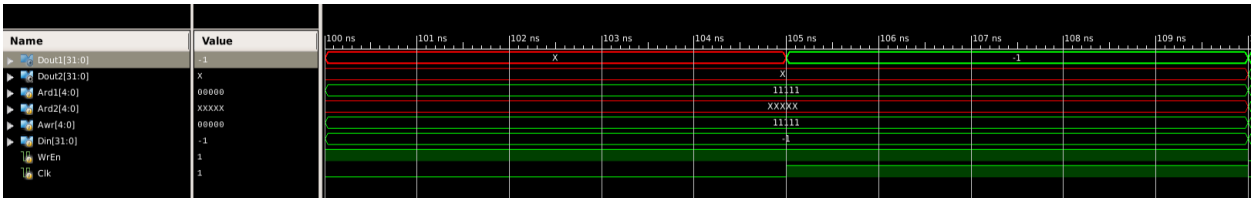


Εικόνα 1



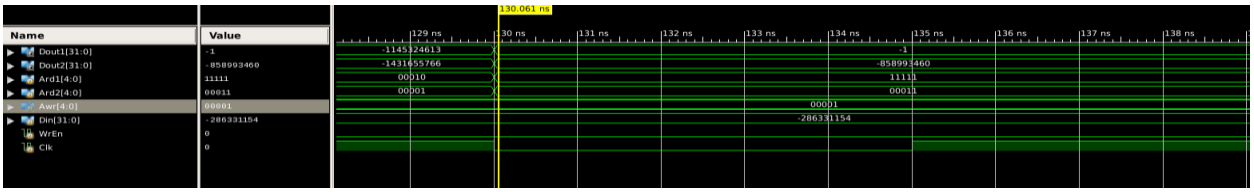
Εικόνα 2

Στην *Εικόνα 2* στα 20ns του testbench και βλέπουμε ότι λειτουργεί ορθά καθώς έχουμε ορίσει το Din = -1_{10} και Ard1 = 00000_2 , Awr = 00000_2 με WrEn = 1_2 προκύπτει η έξοδος Dout1 = -1_{10} .



Εικόνα 3

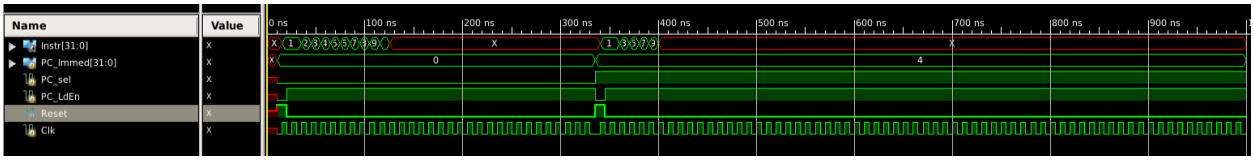
Στην εικόνα 3 στα 105ns για Din = -1_{10} και Ard1 = 11111_2 , Awr = 11111_2 με WrEn = 1_2 προκύπτει η έξοδος Dout1 = -1_{10} . Βλέποντας την εικόνα 4 την χρονική στιγμή 130ns καθώς έχουμε WrEn = 0_2 και Adr1 = 11111_2 όπου η έξοδος Dout1 = -1_{10} το οποίο λειτουργεί σωστά.



Εικόνα 4

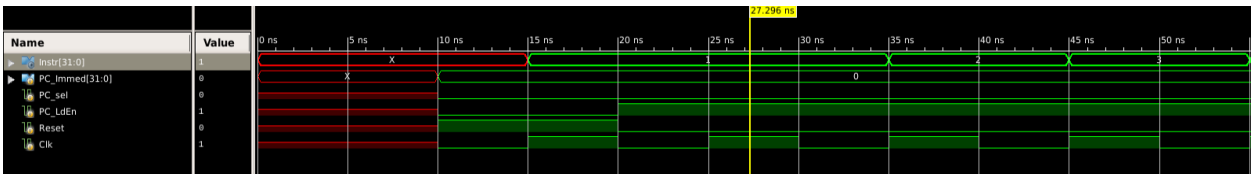
Σχεδιασμός και υλοποίηση βαθμίδας ανάκλησης εντολών (IF)

Κάνοντας προσομοίωση βλέπουμε στην *Εικόνα 5* όλα τα στάδια του testbench.



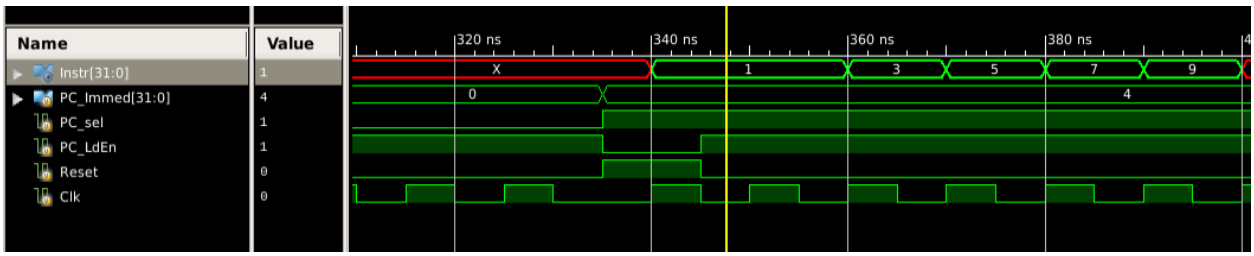
Εικόνα 5

Βλέποντας την *Εικόνα 6* ότι λειτουργεί ορθά καθώς PC_sel = 0_2 PC_LdEn = 1_2 και το Instr = 1_{10} που είναι η πρώτη εντολή του rom.data.



Εικόνα 6

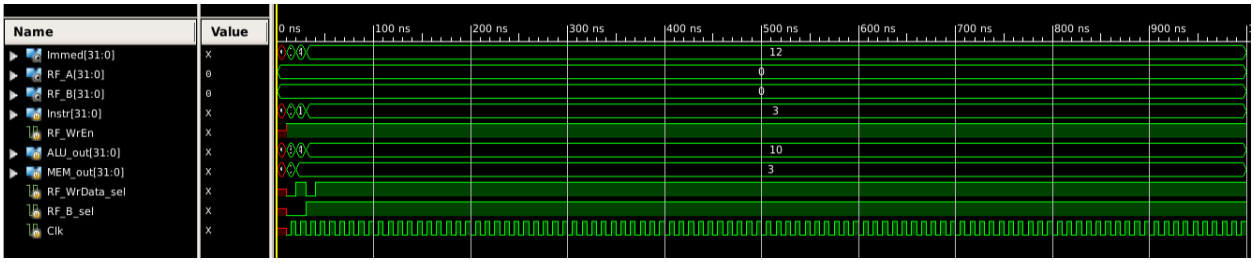
Στην *Εικόνα 7* την χρονική στιγμή στα 340ns βλέπουμε ότι έχοντας ενεργοποιήσει PC_sel = 1_2 έχοντας σαν αρχικό Instr = 1_{10} και PC_Immed = 4_{10} στην χρονική στιγμή τον 360ns βλέπουμε ότι το Instr = 3_{10} το οποίο λειτουργεί ορθά.



Εικόνα 7

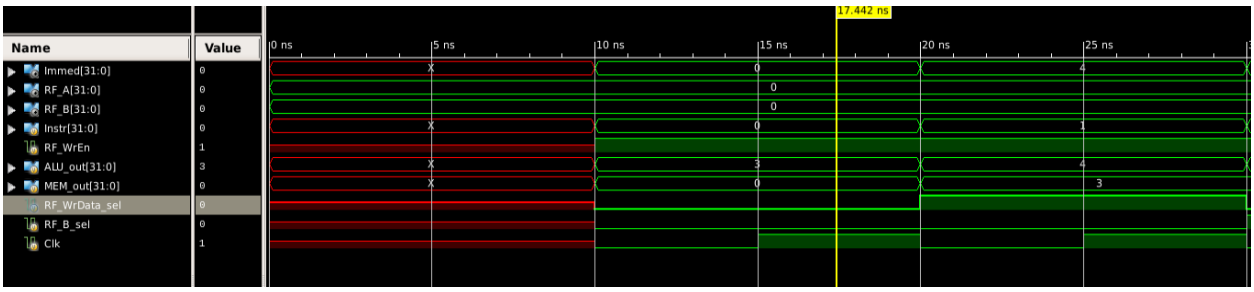
Σχεδιασμός και υλοποίηση βαθμίδας αποκωδικοποίησης εντολών (DECODE)

Κάνοντας προσομοίωση βλέπουμε στην *Εικόνα 8* ολόκληρο το testbench.



Εικόνα 8

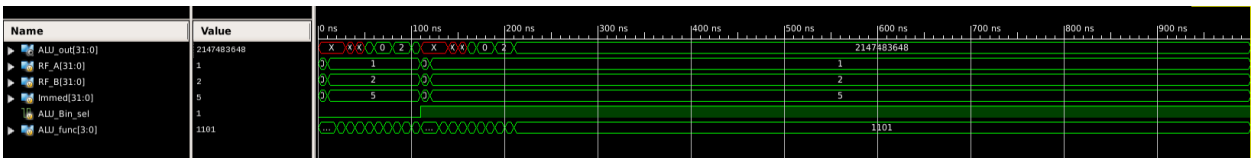
Στην *Εικόνα 9* στο testbench βλέπουμε ότι την χρονική διάρκεια του 10ns έχουμε Instr = 0_{10} όμως το ALU_out = 3_{10} το RF_WrDataSel = 0_2 όλα μαζί θα μας οδηγήσουν σε RF_A = RF_B = Immed = 0_{10} το οποίο λειτουργεί ορθά. Την χρονική στιγμή τον 20ns Immed = 4_{10} έχουμε Instr = 1_{10} αρά το Opcode = 000000_2 και RF_WrData_sel = 1_2 RF_WrEn = 1_2 που μας οδηγεί σε shit 2 θέσεων δεξιά και προκύπτει ALU_out = 4_{10} MEM_out = 3_{10} το οποίο λειτουργεί σωστά.



Εικόνα 9

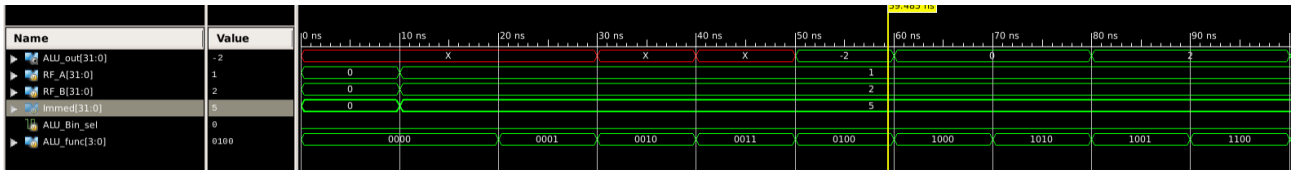
Σχεδιασμός και υλοποίηση βαθμίδας Εκτέλεσης Εντολών (ALU)

Τώρα θα δούμε τα testbench του ALU MODULE στην παρακάτω εικόνα βλέπουμε ολόκληρο το testbench.



Εικόνα 10

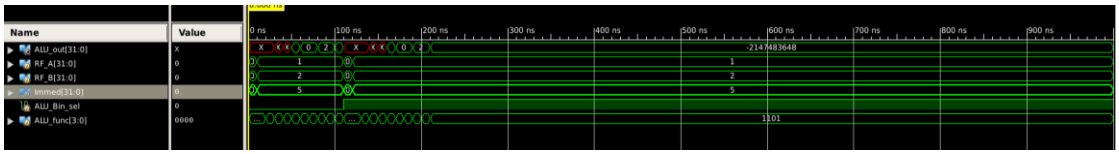
Στην χρονική στιγμή το 50ns έχοντας ALU_func = 0100_2 το οποίο κάνει αντιστροφή του πρώτου καταχωρητή του RF βλέπουμε ότι το RF_B = 2_{10} στο ALU_out = -2_{10} το οποίο λειτουργεί σύμφωνα με της προδιαγραφές. Επιπλέον την χρονική στιγμή το 60ns βλέπουμε ότι καθώς έχουμε ALU_func = 1000_2 το οποίο κάνει αριθμητική ολίσθηση δεξιά κατά 1 θέση καθώς ο καταχωρητής είναι RF_A = 1_{10} η έξοδος μας προκύπτει ALU_out = 0_{10} .



Εικόνα 11

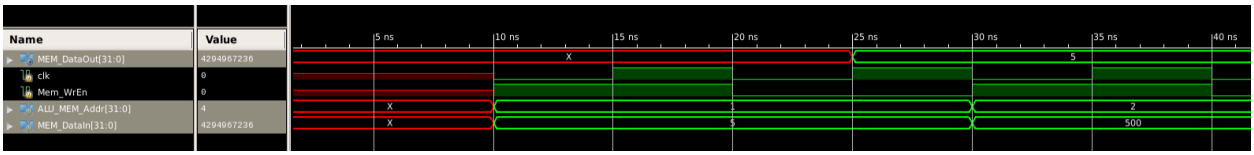
Σχεδιασμός και υλοποίηση βαθμίδας Πρόσβασης Μνήμης (MEM)

Τώρα θα δούμε τα testbench του MEMSTAGE στην *Εικόνα 12* βλέπουμε ολόκληρο το testbench.



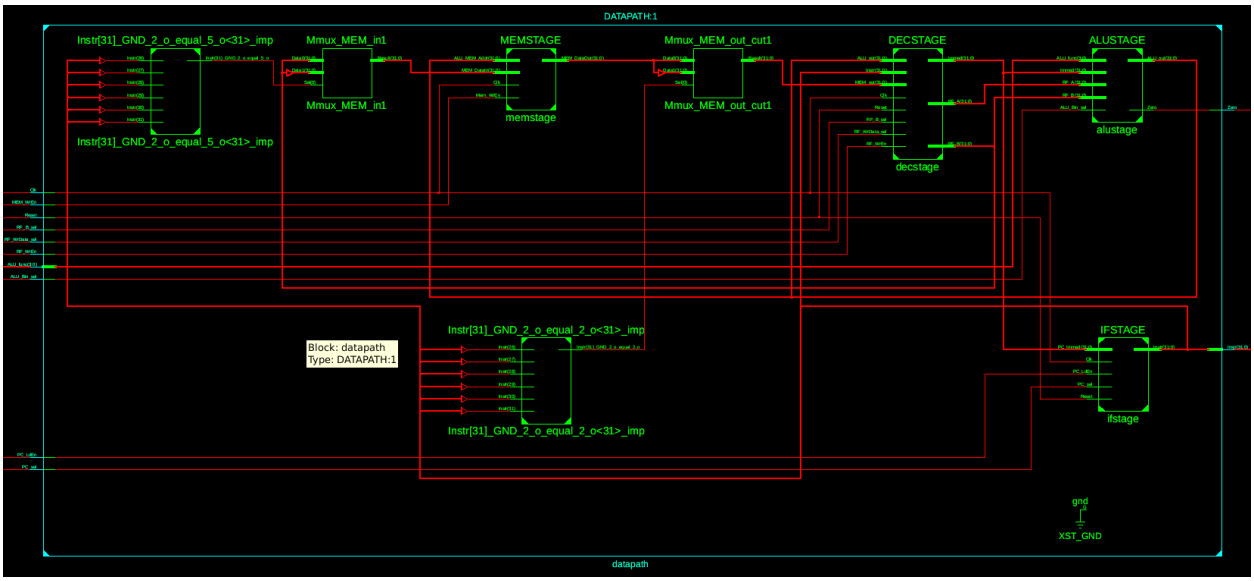
Εικόνα 12

Στην *Εικόνα 13* βλέπουμε ότι την χρονική στιγμή τον 10ns έχουμε το Mem_WrEn = 1 ώστε να μπορέσει να γίνει η εγγραφή στην μνήμη RAM όπου το ALU_MEM_Addr = 1_{10} και τα δεδομένα που θα γράψει στην συγκεκριμένη διεύθυνση θα είναι MEM_DataIn = 5_{10}, τώρα πηγαίνοντας στην χρονική στιγμή τον 25ns βλέπουμε ότι έξοδος MEM_DataOut = 5_{10} και έχοντας απενεργοποιήσει το Mem_WrEn = 0 το οποίο λειτουργεί ορθά.



Εικόνα 13

Datapath

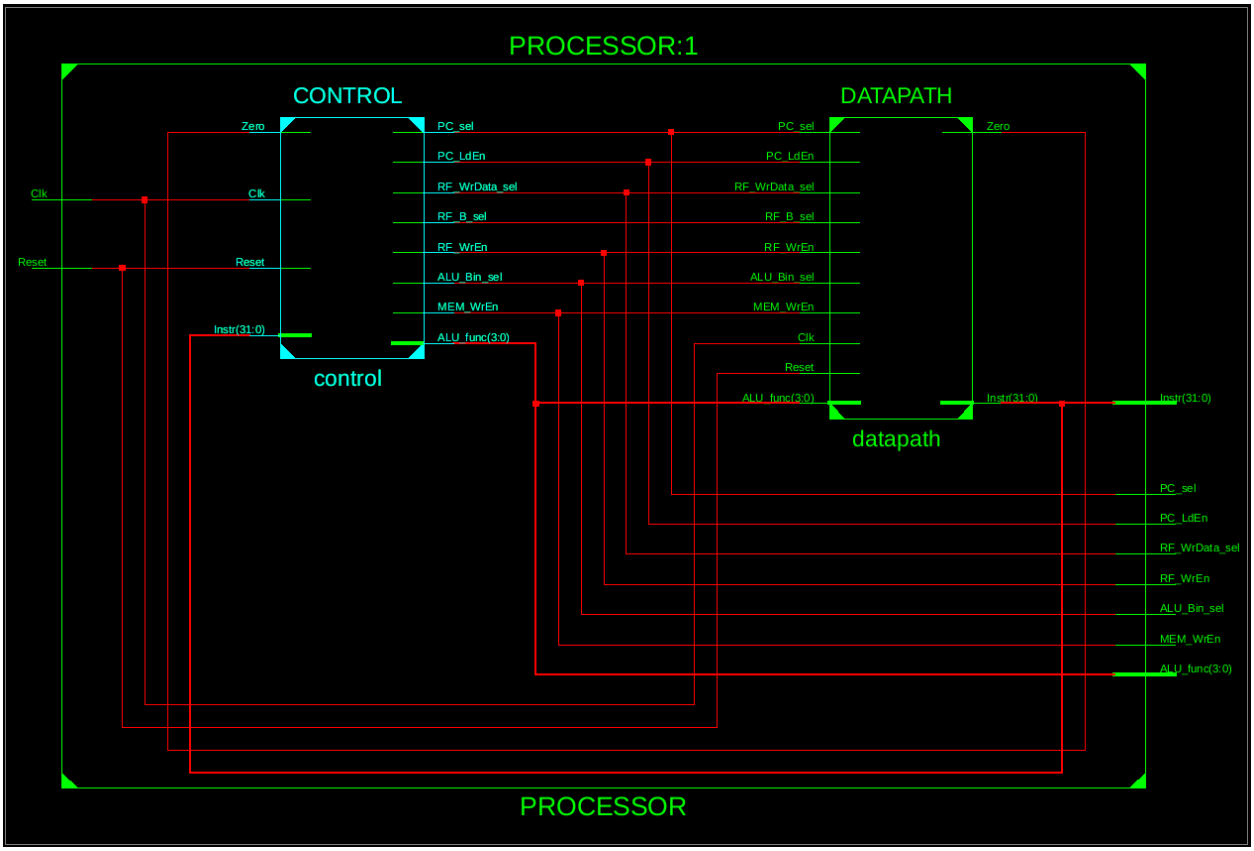


Εικόνα 14

Στην *Εικόνα 14* βλέπουμε την υλοποίηση του datapath. Η επιπλέον λογική που προστέθηκε είναι τα *Mmux_MEM_in1* και *Mmux_MEM_out_cut1* τα οποία χρησιμοποιούνται για την εκτέλεση των εντολών *lw*, *lb.*

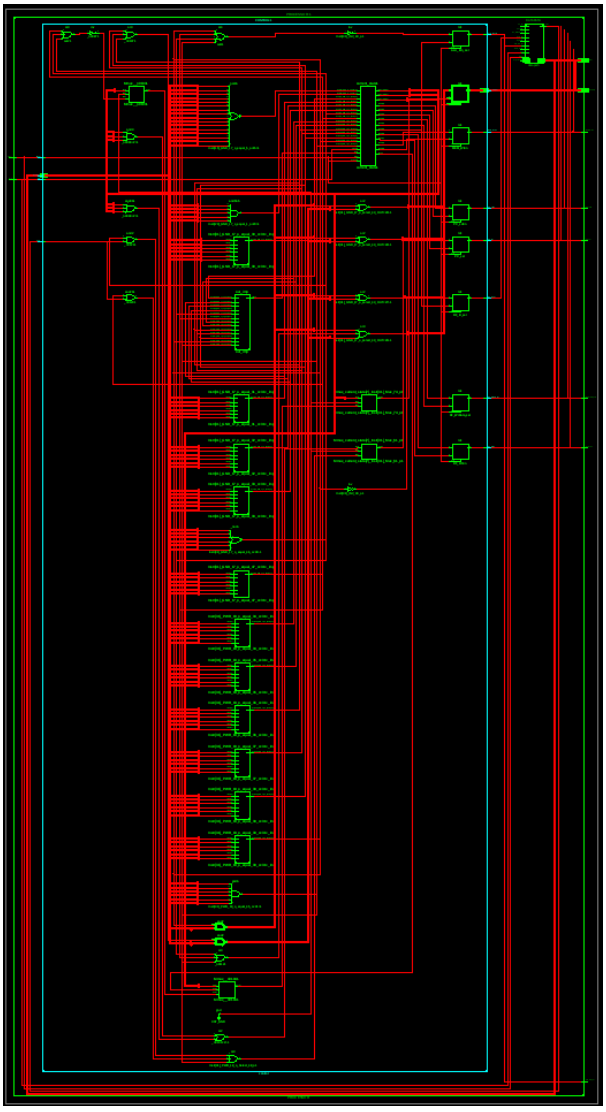
Βασική FSM Ελέγχου του Επεξεργαστή και ολοκλήρωση ενός επεξεργαστή πολλαπλών κύκλων

Η FSM αποτελεί μια υλοποίηση μηχανής πεπερασμένων καταστάσεων τύπου Meely. Για υλοποίηση τύπου Moore η απαίτηση σε αριθμό καταστάσεων θα την καθιστούσε ανέφικτη. Οι καταστάσεις της FSM είναι οι εξής: *i_fetch*, *i_load*, *comb_exec*, *branch*, *mem_wr*, *mem_rd*, *ld_wb*, και *alu_wb*. Δεν ακολουθήθηκε το πρότυπο καταστάσεων που δόθηκε εφόσον καθορίσαμε πως 2 (πιθανόν και 3) καταστάσεις μπορούν να παραληφθούν ωστόσο κρίθηκε απαραίτητη η χρήση μιας κατάστασης επιπλέον ώστε να λειτουργήσει σωστά σύμφωνα με τη λογική που αναπτύχθηκε. Έτσι ο τελικός αριθμός καταστάσεων είναι 8 (από 9).



Εικόνα 15

Στην εικόνα 15 φαίνεται η σύνδεση μεταξύ του *CONTROL* και του *DATAPATH.*



Εικόνα 16

i_fetch

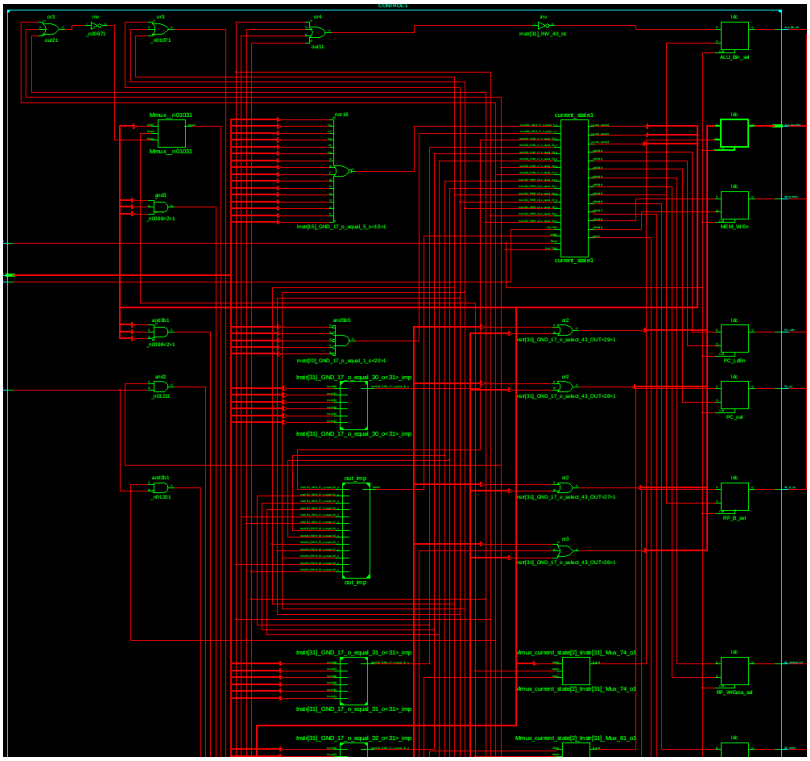
Σε αυτή τη κατάσταση ενεργοποιείται το σήμα εγγραφής *PC_LdEn* ενώ απενεργοποιούνται τα υπόλοιπα σήμα εγγραφής, *RF_WrEn* και *MEM_WrEn*. Η μηχανή μεταβαίνει στην *i_load* ως επόμενη κατάσταση.

i_load

Σκοπός αυτής της κατάστασης είναι να λειτουργεί ως buffer στη λογική του *datapath*, δηλαδή στο πέρας της κατάστασης αυτής διαβάζεται η νέα εντολή από τη μνήμη. Απενεργοποιεί το σήμα *PC_LdEn*. Η μηχανή μεταβαίνει στην *comb_exec* ως επόμενη κατάσταση.

comb_exec

Στη κατάσταση αυτή έχουμε τη νέα εντολή ως είσοδο στη μηχανή. Θέτονται οι έλεγχοι που αφορούν συνδυαστική λογική, δηλαδή ανάγνωση από το RF και εκτέλεση πράξεων στην ALU. Το Opcode καθορίζει την επόμενη κατάσταση η οποία μπορεί να είναι: *branch*, *mem_wr*, *mem_rd* ή *alu_wb*.



Εικόνα 17

branch

Ελέγχει το σήμα *PC_sel*. Η σύγκριση γίνεται στην ALU και η απόφαση του branching λαμβάνεται σύμφωνα με την εντολή (*beq/bne*) και την έξοδο *Zero* της *ALU*. Η μηχανή μεταβαίνει στην *i_fetch* ως επόμενη κατάσταση.

mem_wr

Επιτρέπει την εγγραφή στη μνήμη ενεργοποιώντας το σήμα *MEM_WrEn* και θέτει το *PC_sel* ώστε να φορτωθεί η αμέσως επόμενη εντολή. Η μηχανή μεταβαίνει στην *i_fetch* ως επόμενη κατάσταση.

alu_wb

Επιτρέπει την εγγραφή του αποτελέσματος της *ALU* στο *RF* ενεργοποιώντας το σήμα *RF_WrEn* και θέτει το *PC_sel* ώστε να φορτωθεί η αμέσως επόμενη εντολή. Η μηχανή μεταβαίνει στην *i_fetch* ως επόμενη κατάσταση.

mem_wr

Επιτρέπει την ανάγνωση από τη μνήμη απενεργοποιώντας το σήμα *MEM_WrEn*. Η μηχανή μεταβαίνει στην *ld_wb* ως επόμενη κατάσταση.

ld_wb

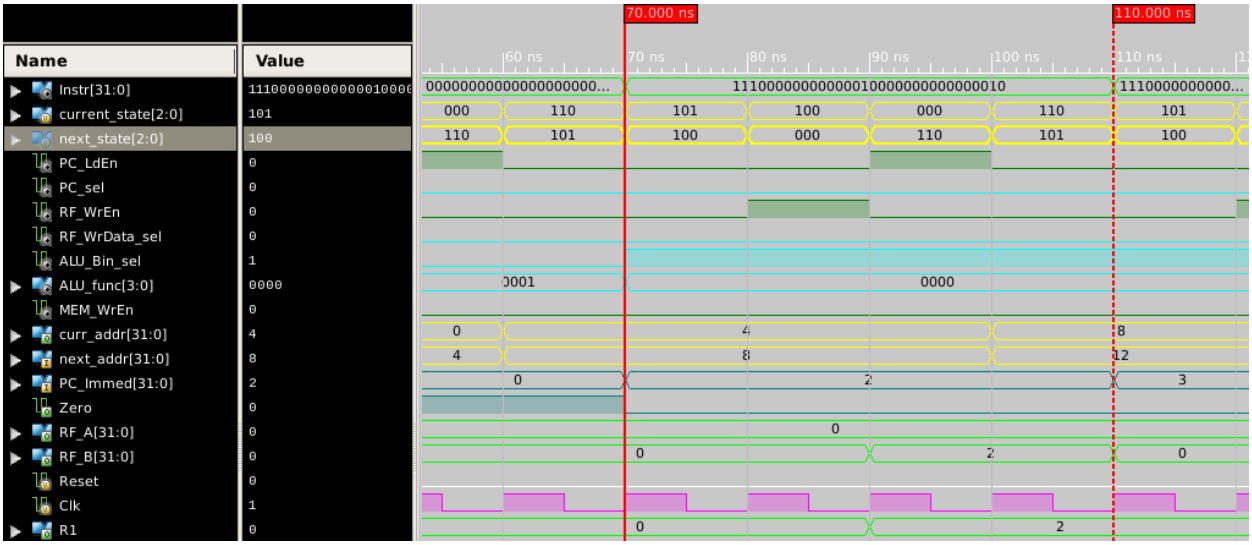
Επιτρέπει την εγγραφή της τιμής που διαβάστηκε από τη μνήμη στο *RF* ενεργοποιώντας το σήμα *RF_WrEn* και θέτει το *PC_sel* ώστε να φορτωθεί η αμέσως επόμενη εντολή. Η μηχανή μεταβαίνει στην *i_fetch* ως επόμενη κατάσταση.

Τώρα θα δούμε το testbench της FSM

Στην *Εικόνα 18* φαίνεται η πρώτη εντολή (*NOP*) της οποίας η εκτέλεση ξεκινά όταν απενεργοποιηθεί το σήμα *Reset*. Κατά την εκτέλεση της δεν ενεργοποιείται κανένα σήμα εγγραφής, ούτε αλλάζει το *PC_sel* έστω και αν μοιράζεται το ίδιο *OPCODE* με την εντολή *beq*.

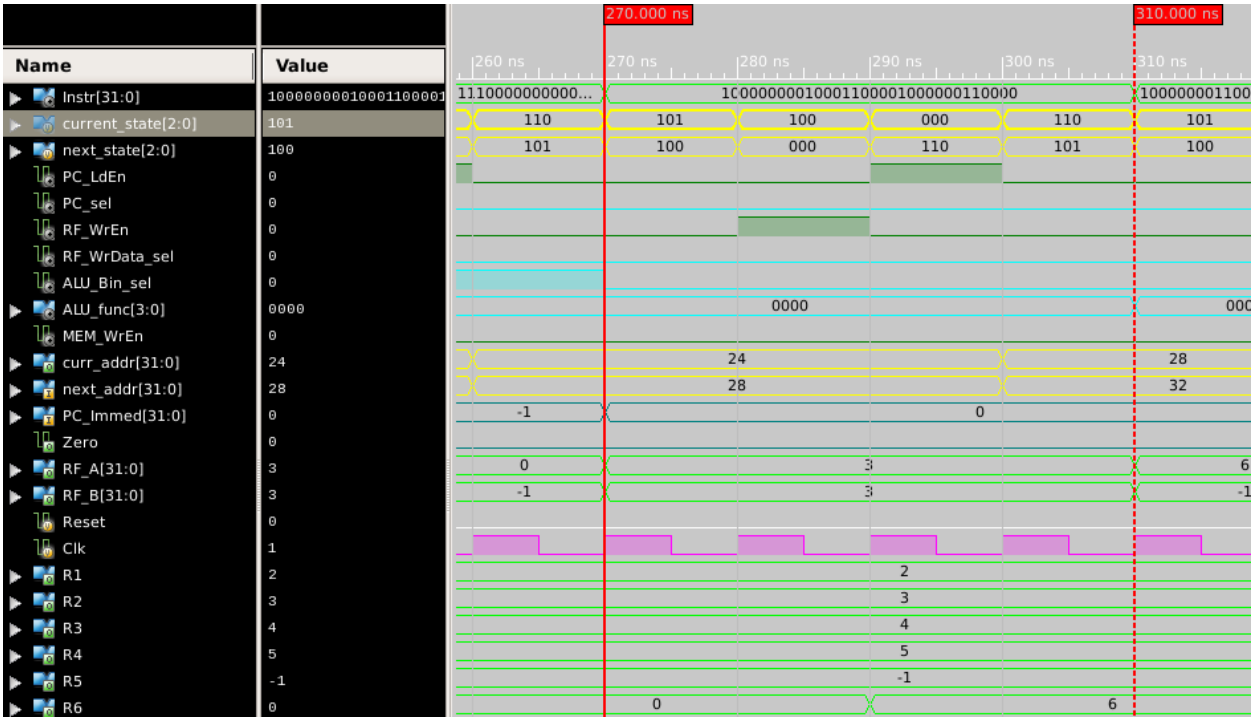
Εικόνα 18

Στη *Εικόνα 19* βλέπουμε την εκτέλεση μιας *li*, στη δεύτερη κατάσταση ενεργοποιείται το *RF_WrEn* ενώ στην τρίτη το *PC_LdEn*, ώστε να φορτωθεί η επόμενη εντολή.



Εικόνα 19

Στην *Εικόνα 20* φαίνεται η εκτέλεση μιας εντολής *add* η οποία ενεργοποιεί παρόμοια σήματα εγγραφής με την παραπάνω εντολή αλλά με διαφορετικός έλεγχο πολυπλεκτών. Η μετάβαση της τιμής του καταχωρητή *R6* φαίνεται στη κυματομορφή.



Εικόνα 20

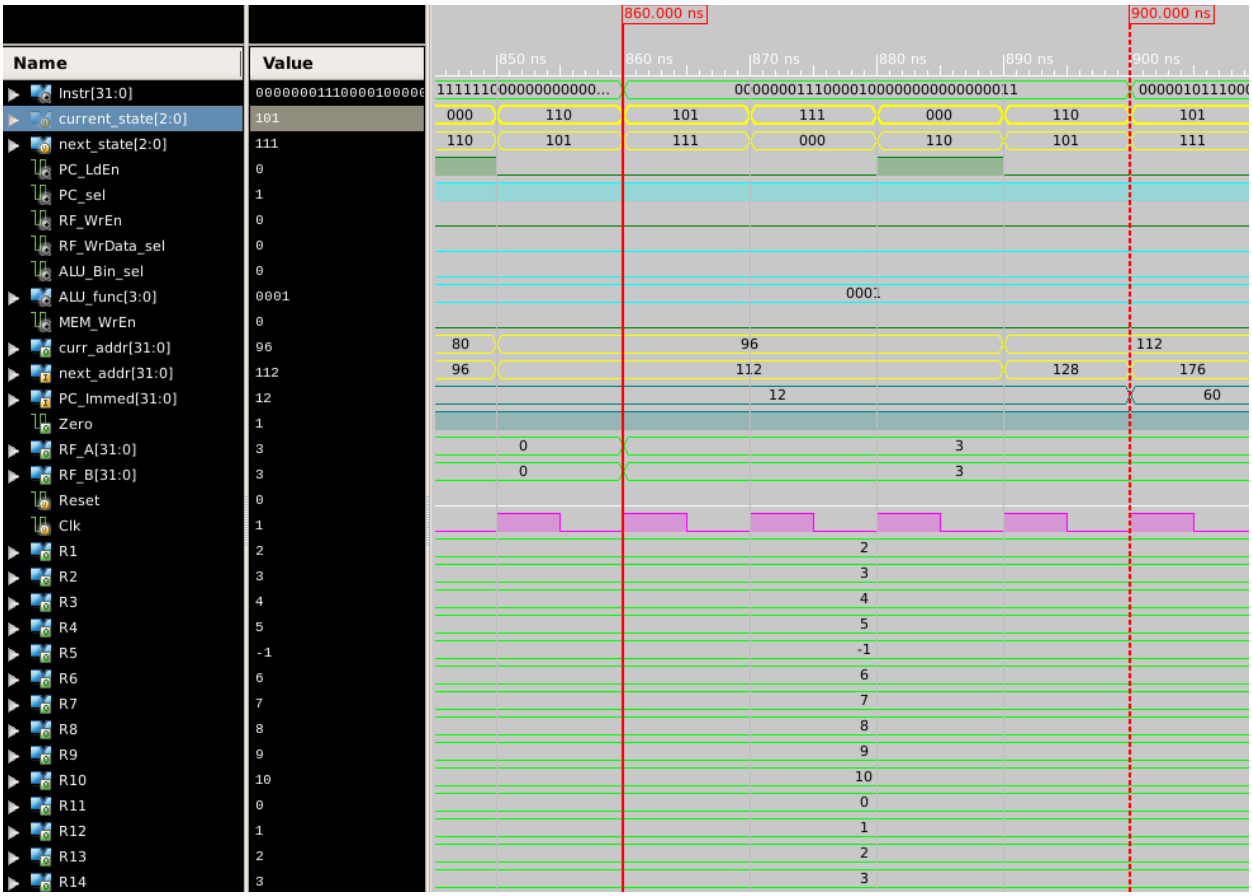
Στην *Εικόνα 21* φαίνεται η εκτέλεση μιας *addi*, παρόμοια διαδικασία με την παραπάνω εντολή αλλά διαφορετικά σήματα ελεγχου πολυπλεκτών, συγκεκριμένα του *ALU_Bin_sel*. Σημειώνεται πως η τιμή που δίνεται στο *alu_func* εξάγεται από το *OPCODE* σε πράξεις με *immediate* αριθμούς.

Εικόνα 21

Στην περίπτωση αυτή γίνεται απόπειρα εγγραφής στον καταχωρητή *R0*, το *Control* αντιλαμβάνεται ότι δεν είναι επιτρεπτή εντολή και φορτώνει την επόμενη.

Εικόνα 22

Στην *Εικόνα 23* φαίνεται η εκτέλεση μιας *beq*. Κύρια σημεία ενδιαφέροντος είναι τα σήματα *Zero* και *PC_sel* όπου ουσιαστικά το *branch* γίνεται δεκτό (*Zero = 1*) και το *PC_sel = 1*(στο συγκεκριμένο παράδειγμα το *PC_sel* διατηρείται στο 1 εφόσον η προηγούμενη εντολή ήταν ένα πετυχημένο branch).

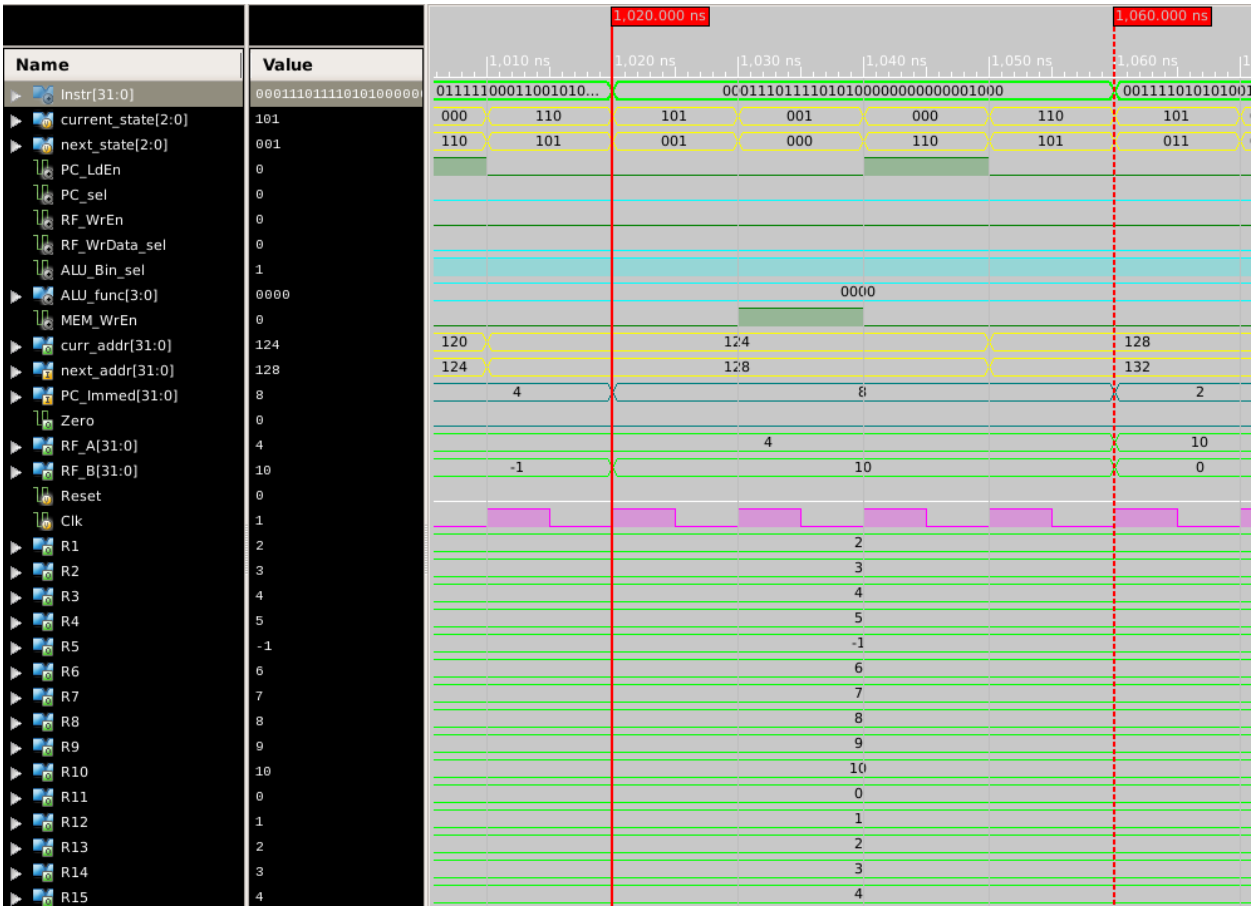


Εικόνα 23

Στην *Εικόνα 24* φαίνεται η εκτέλεση μιας εντολής *lb*. Αυτό που αξίζει να παρατηρηθεί και διαφέρει από άλλες εντολές που γράφουν στο *RF* είναι ότι η εκτέλεση της διαρκεί έναν κύκλο ρολογιού παραπάνω και αυτό διότι απαιτείται ένας επιπλέον κύκλος για την ανάγνωση από τη μνήμη.

Εικόνα 24

Στην *Εικόνα 25* φαίνεται η εκτέλεση μιας εντολής *sb* στην οποία, όπως και την *branch*, δεν γράφει στο *RF* αλλά στη μνήμη, *MEM_WrEn = 1.*



Εικόνα 25

Προβλήματα/Δυσκολίες

Τα αρχικά προβλήματα ήταν η εξοικείωση με την Verilog το οποίο διαφέρει από κάποια γλώσσα προγραμματισμού μιας και πρέπει να σκεφτόμαστε ότι γράφουμε υλικό και όχι λογισμικό. Κάτι το οποίο δεν έχει διδαχθεί στα προηγούμενα έτη. Καλό θα ήταν να αλλάξει αυτό και να προστεθή ως στόχος στο μάθημα Λογικής Σχεδίασης για να μπορέσει ο κάθε ένας που ενδιαφέρεται αργότερα να προχωρήσει σε ποιο προχωρημένο στάδιο ως ένας pipelined επεξεργαστής , διαχείριση του Κοινού Δίαυλου Δεδομένων (Common Data Bus) και πολλά αλλά. Η επόμενη δυσκολία ήταν στο RegisterFile που έπρεπε να γράψεις 32 διαφορετικούς καταχωρητές και να τα συνδέσεις λογικά. Γενικότερα τα προβλήματα ήταν η εξοικείωση με την λογική της γραφής κώδικα για υλικό, το οποίο στις αρχικές απόπειρες δυσκόλευε κάθε έναν από εμάς που ήταν η πρώτη του εμπειρία με την Verilog.

Ιδιαίτερη δυσκολία παρουσιάστηκε στην υλοποίηση της FSM, ενώ όλα τα υπόλοιπα modules ήταν ουσιαστικά λειτουργικά από τη πρώτη εκτέλεση των testbench τους. Οι προσπάθειες υλοποίησης του Control module ήταν αμέτρητες με ριζικές αλλαγές από τη μια υλοποίηση στην άλλη. Στο debugging του design παρατηρώντας τις κυματομορφές των σημάτων δημιουργήθηκε αρχικά σύγχυση μεταξύ αιτίου και αποτελέσματος, δηλαδή ποιο σήμα επηρέασε τα υπόλοιπα εφόσον όλες οι κυματομορφές μεταβάλλονται ταυτόχρονα με τον χτύπο του ρολογιού.

Καλή βοήθεια θα ήταν το Xilinx ISE να υποστήριζε custom radix definitions ώστε να μπορούσαμε να αναπαραστήσουμε τα states της FSM με την ονομασία τους και όχι με την τιμή τους, κάτι που θα διευκόλυνε τη διαδικασία αποσφαλμάτωσης.