ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ
ΗΡΥ 312: ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2019-2020

Ομάδα εργασίας LAB31243764
Γκογκολάκη Ελένη 2012030071
Ραφαήλ Τσιριβάκος 2013030199

Εργασία #1 : Δημιουργία επεξεργαστή ενός κύκλου

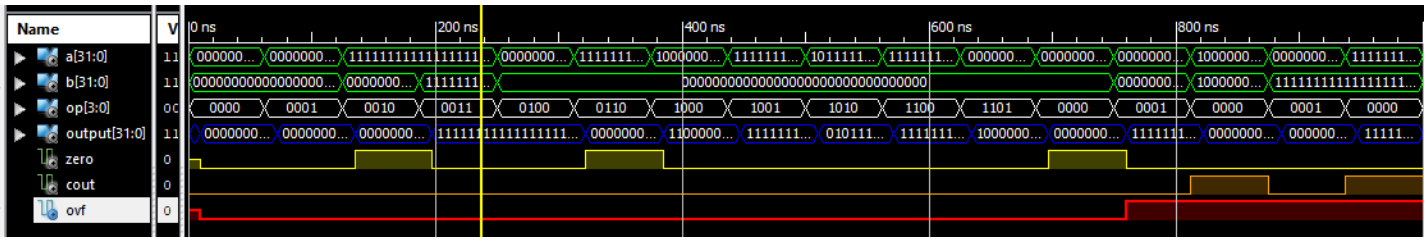
1η φάση

«Σχεδίαση μονάδας αριθμητικών και λογικών πραξεων (ALU)
και αρχείου καταχωρητών (Register File)»

A) Μονάδα αριθμητικών και λογικών πράξεων (Arithmetic Logic Unit ή ALU)

Με βάση τα δεδομένα της εκφώνησης υλοποιήσαμε την ALU σε VHDL. Χρησιμοποιήσαμε Behavioral κώδικα. Για παράδειγμα, για να φτιάξουμε τον αθροιστή χρησιμοποιήσαμε τον τελεστή «+» και για την αφαίρεση «-». Προσθέσαμε καθυστέρηση στην τελική έξοδο της ALU χρησιμοποιώντας το **after** της VHDL ώστε το αποτέλεσμα να παράγεται 10 nanoseconds μετά την είσοδο.

Προσομοιώσαμε την ALU στο περιβάλλον της Xilinx και παρακάτω φαίνεται η κυματομορφή που προκύπτει με εισόδους που βεβαιωνόμαστε ότι τα σήματα **Zero**, **Cout** και **Ovf** λειτουργούν σωστά, αλλά και corner cases. Σε κάθε περίπτωση έχουμε το επιθυμητό αποτέλεσμα.



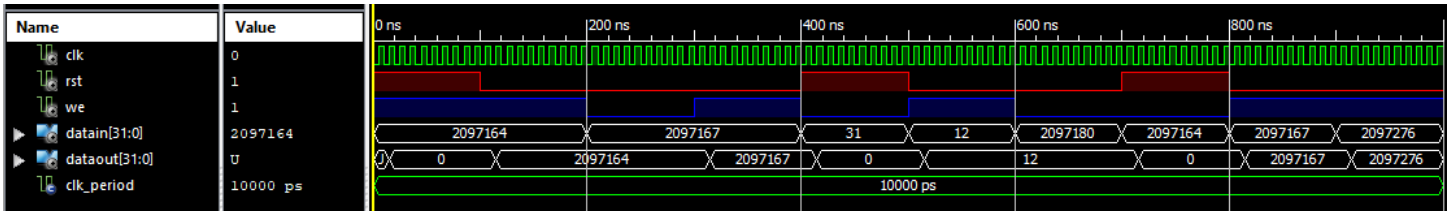
B) Αρχείο καταχωρητών (Register File ή RF)

B1. Δημιουργία καταχωρητή

Υλοποιήσαμε σε VHDL έναν καταχωρητή 32 bits.

Προσομοιώσαμε τον Register στο περιβάλλον της Xilinx και παρακάτω φαίνεται η κυματομορφή που προκύπτει.

Η υλοποίηση είναι σωστή, καθώς γίνεται εγγραφή, δηλαδή το datain παίρνει την τιμή του dataout όταν το σήμα we είναι '1' και το rst είναι '0', διαφορετικά παίρνει την τιμή '0'.



B2. Δημιουργία αρχείου καταχωρητών (RF)

Δημιουργήσαμε 32 καταχωρητές, όπως αυτόν που υλοποιήσαμε στο βήμα B1, χρησιμοποιώντας το **for-generate** της VHDL. Στη συνέχεια υλοποιήσαμε το RF με 32 καταχωρητές με τρεις θύρες, δύο ανάγνωσης και μία εγγραφής.

Οι πολυπλέκτες και ο αποκωδικοποιητής έχουν στην έξοδο τους καθυστέρηση 10 nsec σε σχέση με τις εισόδους τους. Οι πύλες AND έχουν καθυστέρηση 2 nsec σε σχέση με τις εισόδους τους.

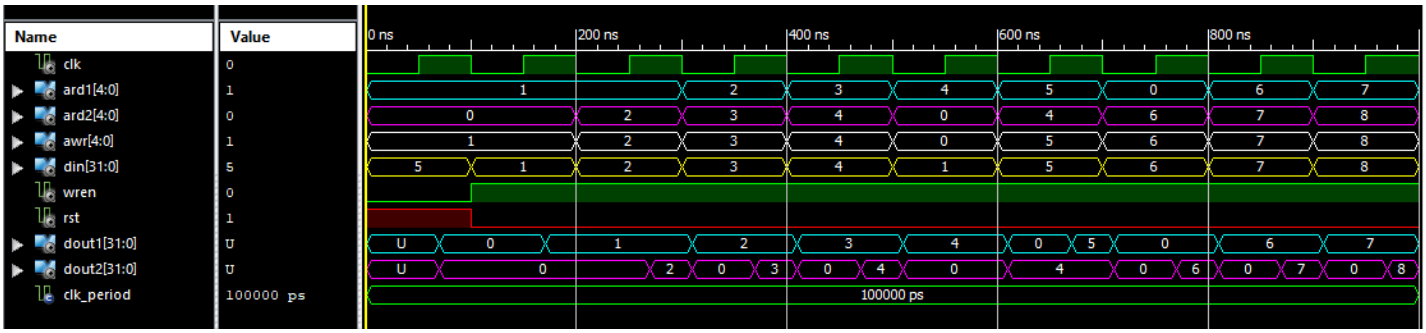
Για τον καταχωρητή R0 στον MIPS η τιμή του είναι πάντα σταθερή = ‘0’, οπότε θέσαμε για τον R0 το DataIn <= “00000000000000000000000000000000”.

Προσομοιώσαμε την RF στο περιβάλλον της Xilinx και παρακάτω φαινεται το waveform που προκύπτει.

Δώσαμε πολλές διαφορετικές τιμές στις εισόδους ώστε να επαληθεύσουμε την ορθή λειτουργία της σε όλες τις περιπτώσεις.

Στο testbench, θέσαμε την περίοδο του ρολογιού (CLK) με 100ns.

Όταν έχουμε rst=’0’ και wren=’1’ γίνεται η εγγραφή. Στο R0 έχουμε τιμή πάντα ‘0’ ανεξάρτητα τι ζητάμε να μπει. Για τους υπόλοιπους καταχωρητές βάλαμε din ίσο με τις διευθύνσεις τους, για να είναι πιο εύκολος ο έλεγχος. Η τιμή του κάθε καταχωρητή γίνεται ίση με το din στην ακμή του ρολογιού, ενώ πριν είναι ‘0’.



2η φάση

«Σχεδίαση των βασικών βαθμίδων του datapath»

Αρχιτεκτονική Συνόλου Εντολών

Θα υλοποιήσετε τμήματα ενός non-pipelined επεξεργαστή βασισμένου σε υποσύνολο της αρχιτεκτονικής συνόλου εντολών CHARIS (CHAnia Risc Instruction Set).

Α. Μελετήστε την κωδικοποίηση των εντολών του CHARIS

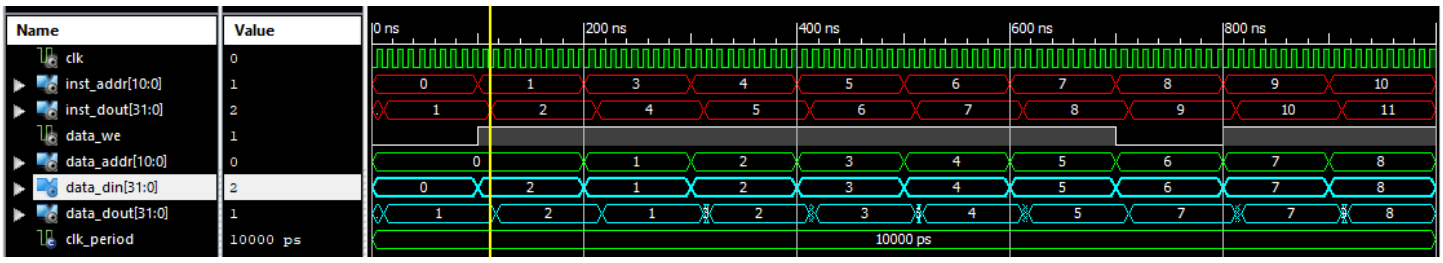
Παρατηρήσαμε την ομαδοποίηση τους έχοντας στο μυαλό μας τον σκοπό της αποκωδικοποίησης: να παραχθούν τα απαραίτητα σήματα ελέγχου για την εκτέλεση της κάθε εντολής. Βρείκαμε τις υπάρχουσες συμμετρίες ώστε να απλοποιήσετε τη λογική αποκωδικοποίησης.

Β. Υλοποίηση κύριας μνήμης 2048x32

Φτιάξετε μια μνήμη RAM 2048 θέσεων, των 32 bits η κάθε θέση. Έχει μια θύρα ανάγνωσης για τις εντολές, και μια θύρα ανάγνωσης/εγγραφής για τα δεδομένα.

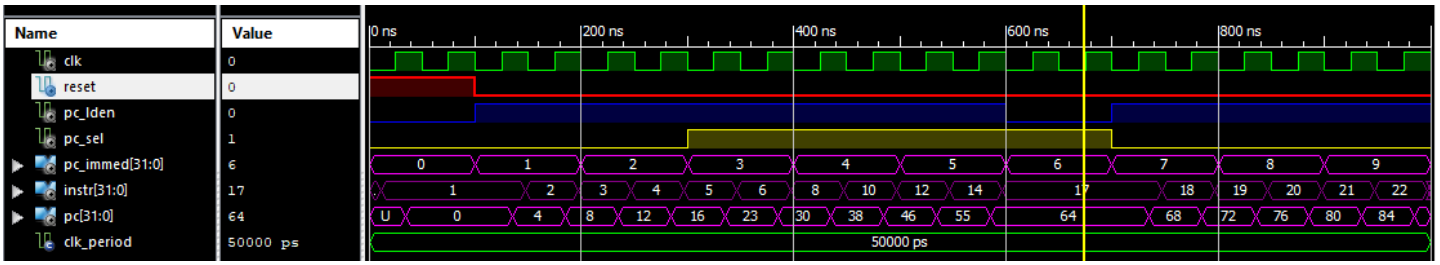
Πριν συνδέουμε τον κώδικα της μνήμης στις βαθμίδες IF και MEM, επαληθεύσαμε πρώτα τη λειτουργία μόνη της, σ’ ένα project. Φτιάξαμε testbench και καλέσαμε ένα instance της μνήμης.

Η μνήμη είναι αρχικοποιημένη με το αρχείο rom.data.



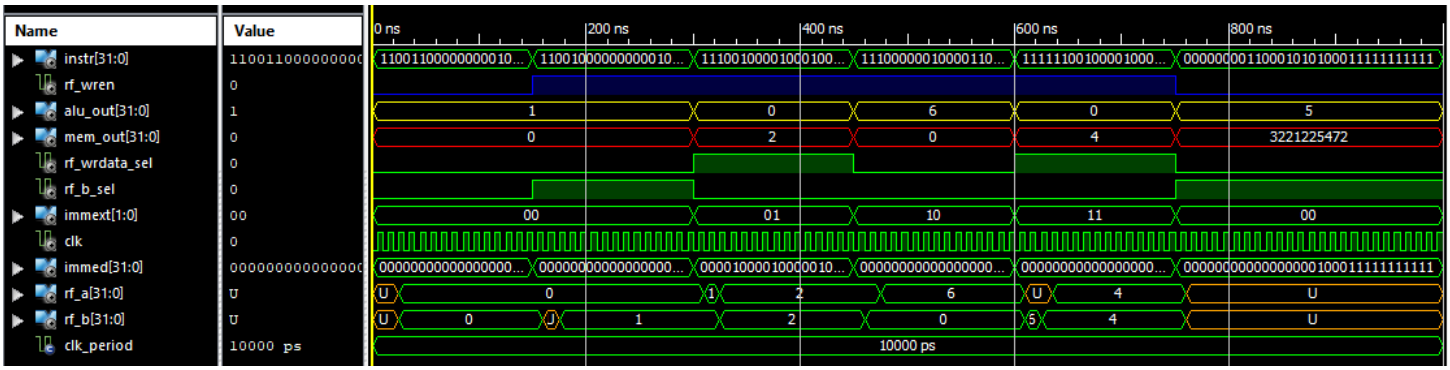
Γ. Βαθμίδα ανάκλησης εντολών (IF)

Γράψαμε κώδικα VHDL που να υλοποιεί τα επιμέρους τμήματα της βαθμίδας IF και κάναμε τις κατάλληλες εσωτερικές συνδέσεις χρησιμοποιώντας την μνήμη, πολυπλέκτες, καταχωρητές και αθροιστές.
Γράψαμε testbench και επαληθεύσαμε τη λειτουργία της βαθμίδας IF κανοντας instance τη μνήμη. Χρησιμοποιήσαμε το αρχείο rom.data για αρχικοποίηση της μνήμης.
Το PC με PC_sel='0' αυξάνει κατά 4, ενώ με PC_sel='1', προστίθεται σε αυτό και η τιμή του PC_Immed και στη μνημη περνάνε τα στοιχεία του 12-2 ως διεύθυνση στη μνήμη. Έτσι, για το αρχείο rom.data, με PC='0', έχουμε και διεύθυνση '0', όπου υπάρχει η τιμή 1 (Instr='1'). Αντίστοιχα, για PC='12', η διεύθυνση είναι '3', όπου βρίσκεται η τιμή 4 (Instr='4'), κλπ.



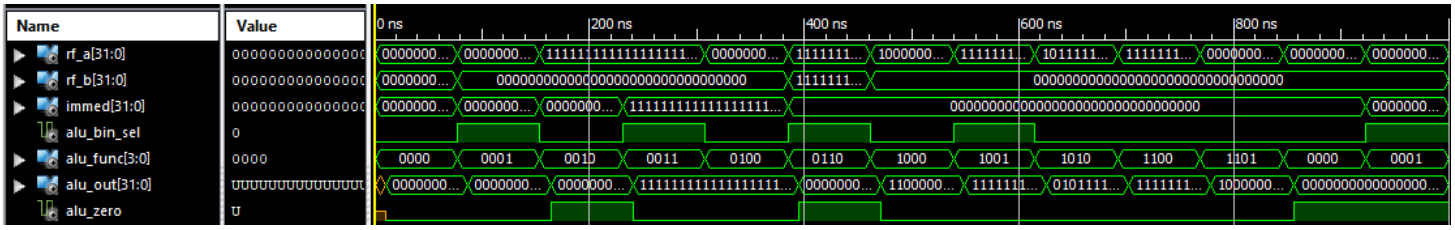
Δ. Βαθμίδα αποκωδικοποίησης εντολών (DECODE)

Γράψαμε κώδικα VHDL που να υλοποιεί τα επιμέρους τμήματα της βαθμίδας DEC και κάνετε τις κατάλληλες εσωτερικές συνδέσεις χρησιμοποιώντας το Register File που παράγαμε στην 1η φάση, πολυπλέκτες και καταχωρητές. Το συννεφάκι στην Εικόνα 5 της εκφώνησης είναι ένας πολυπλέκτης που δέχεται σαν είσοδο τα 16 bits του immediate μιας εντολής και τη μετατρέπει σε ένα σήμα 32 bits, επιλέγοντας αν θα γίνει zero-filling του immediate για ImmExt="00", ολίσθηση και zero-filling για ImmExt="01", sign-extension για ImmExt="10" ή ολίσθηση και sign-extention.
Γράψαμε testbench και επαληθεύσαμε τη λειτουργία της βαθμίδας DEC με κατάλληλες τιμές του Instr. Έχουμε τις επιθημητά RF_A, RF_B και Immed ανάλογα τις εισόδους.



Ε. Βαθμίδα εκτέλεσης εντολών (EX)

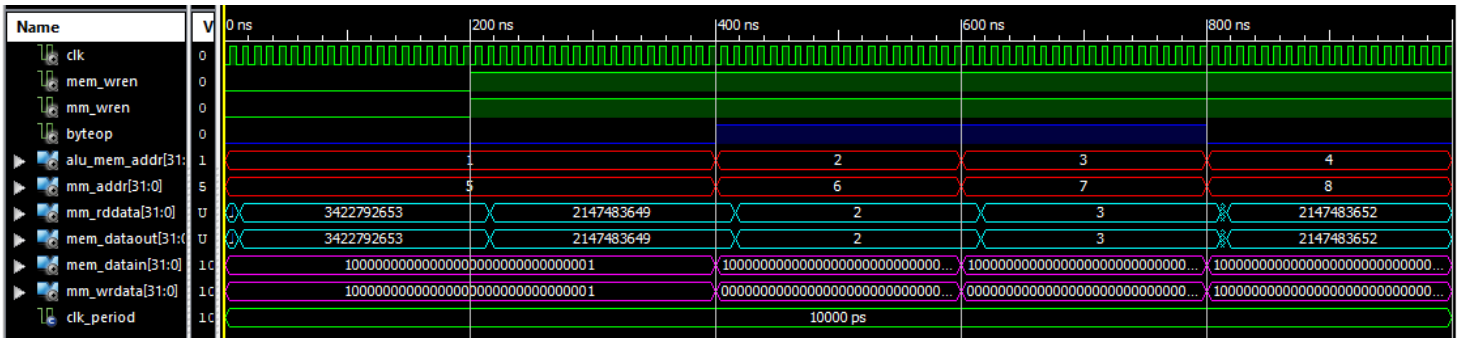
Γράψαμε κώδικα VHDL που να υλοποιεί τα επιμέρους τμήματα της βαθμίδας EX και κάναμε τις κατάλληλες εσωτερικές συνδέσεις χρησιμοποιώντας πολυπλέκτες.
Γράψαμε testbench και επαληθεύσαμε τη λειτουργία της βαθμίδας EX. Σε κάθε είσοδο έχουμε το αναμενόμενο αποτέλεσμα.



ΣΤ. Βαθμίδα πρόσβασης μνήμης (MEM)

Γράψαμε κώδικα VHDL που να υλοποιεί τα επιμέρους τμήματα της βαθμίδας MEM. Συνδέσαμε τη βαθμίδα MEM με την εξωτερική μνήμη στο testbench και επαληθεύσαμε τη λειτουργία της.

Στο waveform φαίνεται η σωστή διευθυνσιοδότηση της μνήμης και η διαφορά των MEM_DataIn και MEM_DataOut ανάλογα το ByteOp, το οποίο όταν ισούται με '0' έχουμε zero-filling στα στοιχεία 31-8 του MEM_DataIn.



3η φάση

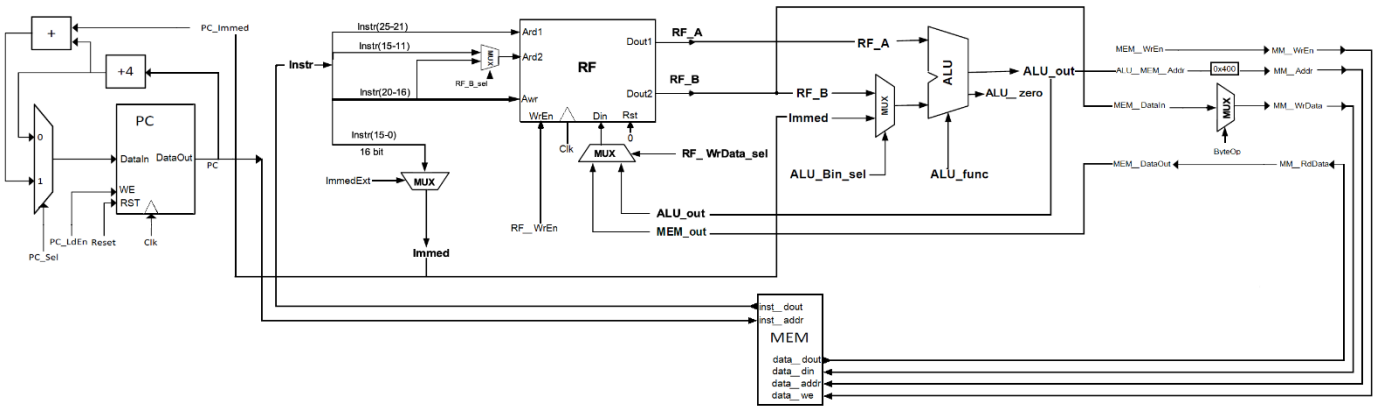
«Ολοκλήρωση του datapath, δημιουργία του control και τελικό integration»

Εκτέλεση

Γράψαμε τον κώδικα VHDL που υλοποιεί το **DATAPATH** του επεξεργαστή στο οποίο κάνεμε τη σύνδεση των βαθμιδων IF, DECODE, EX και MEM που δημιουργήσαμε στην προηγούμενη φάση. Συλλέξαμε όλα τα σήματα ελέγχου. Η διεπαφή του DATAPATH.vhd περιλαμβάνει όλα τα σήματα ελέγχου όλων των επιμέρους βαθμίδων που datapath.

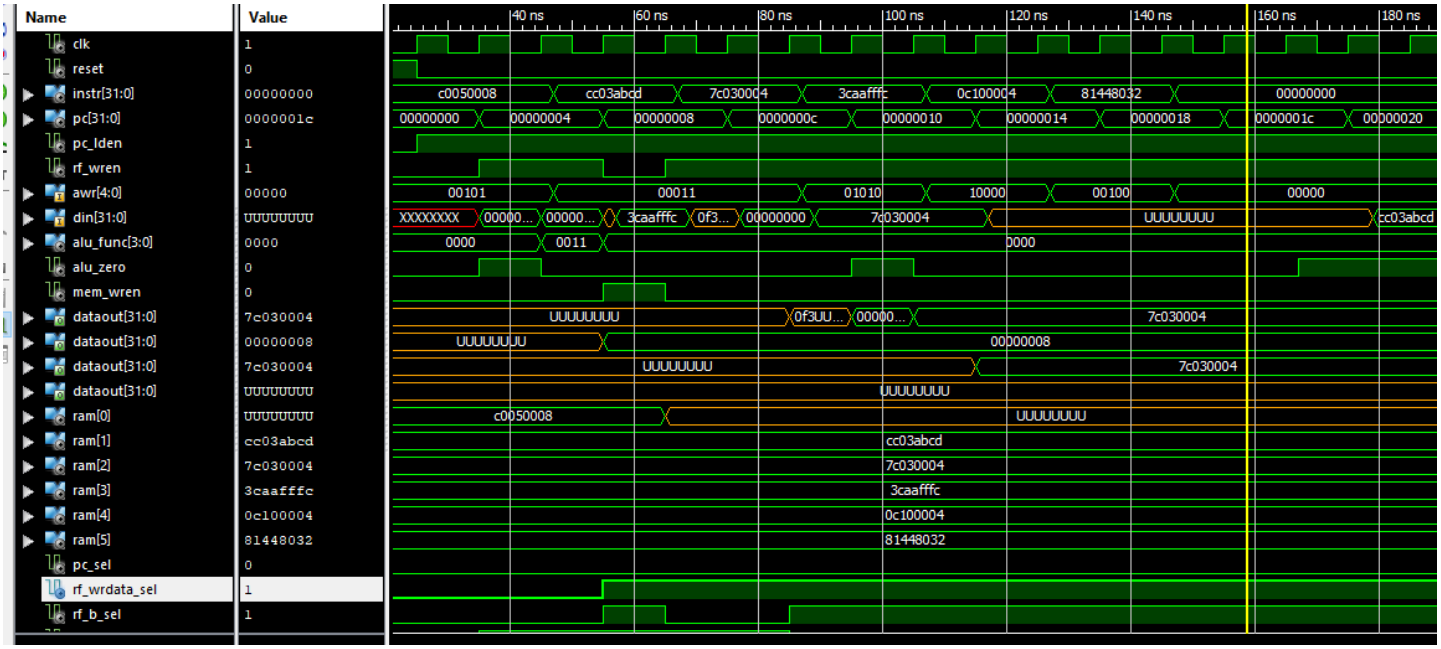
Υπάρχει μια κοινή μνήμη RAM η οποία θα είναι συνδεδεμένη ταυτόχρονα με τις βαθμίδες IF και MEM στα testbenches που δημιουργήσαμε.

Δημιουργήσαμε επίσης προγράμματα αναφοράς για τα testbenches που δημιουργούν τα σήματα ελέγχου για να επιβεβαιώσουμε την ορθή λειτουργία του datapath. Δεν έχουμε απολύτως σωστά αποτελέσματα όμως, ίσως λόγω κακού χρονισμού. Αυτό περνάει φαίνεται και στα υπολοιπα testbenches.



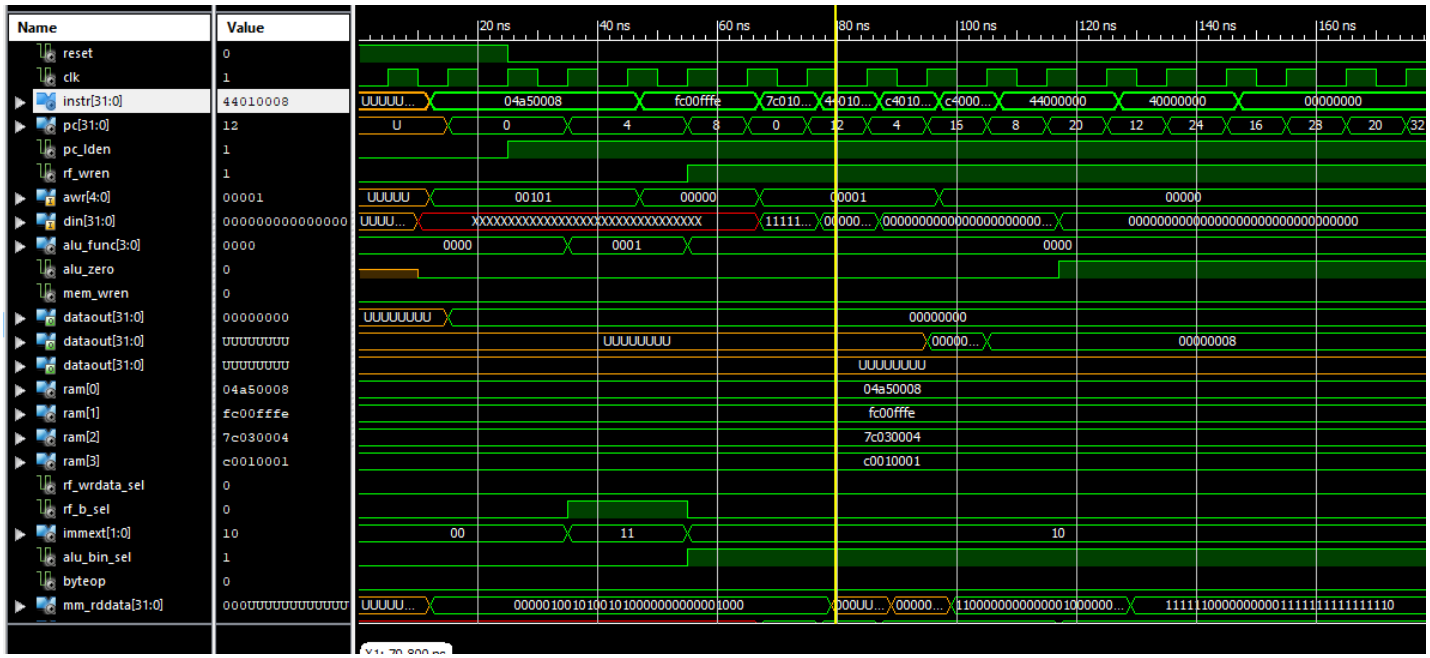
Πρόγραμμα αναφοράς #1

00: addi r5, r0, 8
04: ori r3, r0, 0xABCB
08: sw r3, 4(r0) // γράφει στην διεύθυνση 0x4 => 0x404 την τιμή 0x0000ABCD
0C: lw r10, -4(r5) // διαβάζει από την διεύθυνση 0x4 => 0x404 την τιμή 0x0000ABCD
10: lb r16, 4(r0) // διαβάζει **byte** από την διεύθυνση 0x4 => 0x404 την τιμή 0xFFFFFFCD
14: nand r4, r10, r16

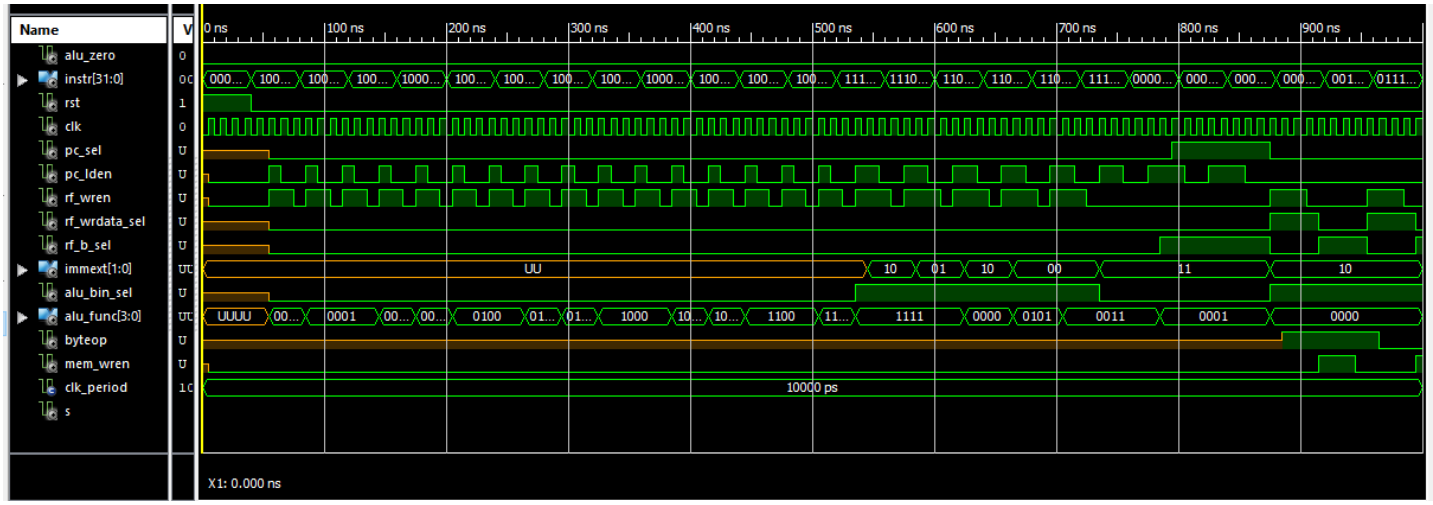


Πρόγραμμα αναφοράς #2

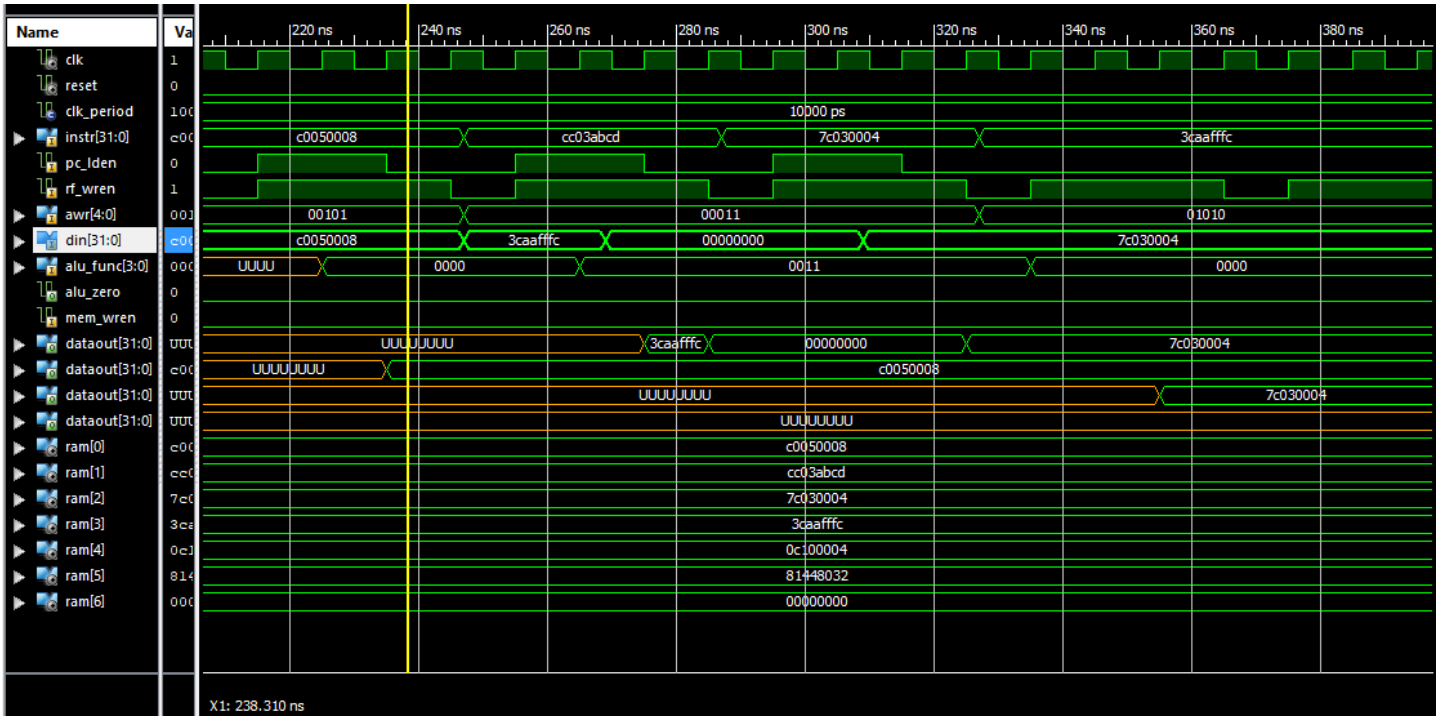
00: bne r5, r5, 8 // αποτυχημένη διακλάδωση
04: b -2 // branch (PC=04 + 4 -2*4 = 00) infinite loop!
08: addi r1, r0, 1 // δεν θα εκτελεστεί

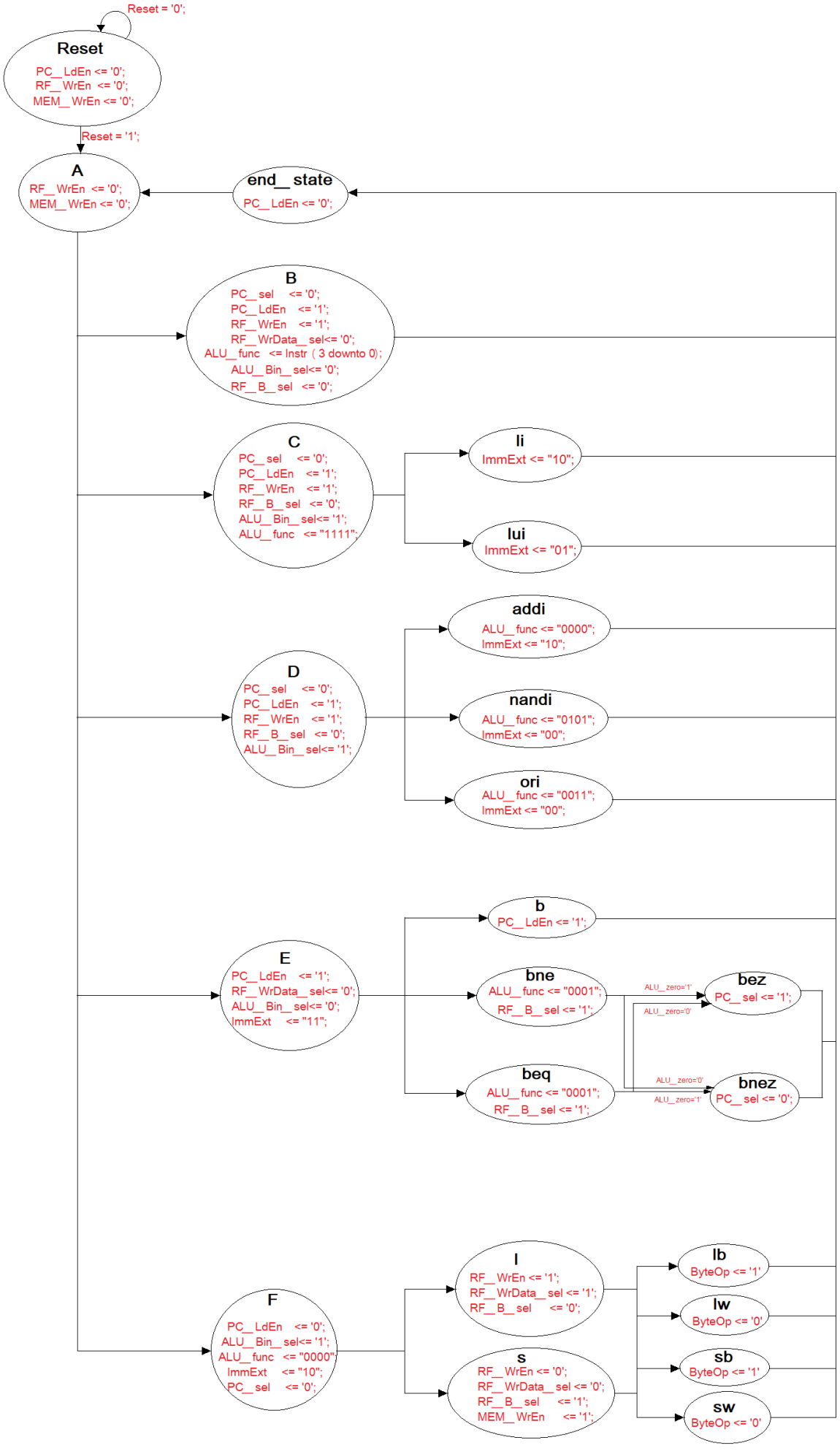


Στη συνέχεια γράψαμε τον κώδικα VHDL που υλοποιεί το **CONTROL** του επεξεργαστή το οποίο παράγει τα σωστά σήματα ελέγχου για την κάθε εντολή.
Δημιουργήσαμε ένα testbench για το control για να επιβεβαιώσουμε την ορθή λειτουργία του. Στο συγκεκριμένο testbench δώσαμε όλες τις εντολές που υποστηρίζει ο επεξεργαστής με τη σειρά και παρατηρούμε ότι παράγονται σωστά τα σήματα που θέλουμε.
Παρακάτω φαίνεται ο τρόπος που υλοποιήσαμε το CONTROL.



Τέλος, γράψαμε τον κώδικα VHDL που υλοποιεί το toplevel του επεξεργαστή ενός κύκλου.Το συγκεκριμένο αρχείο περιλαμβάνει τρία components (port maps): datapath, control και το κοινό module μνήμης.
Για να ελέγξουμε το κύκλωμας δώσαμε στο test bench Reset=’1’ για μερικούς κύκλους και μετά Reset=’0’ έτσι ώστε να ξεκινήσει η εκτέλεση από την θέση 0, χρησιμοποιώντας το πρόγραμμα αναφοράς 1.
Τα σήματα είναι με τη σειρα που μας ζητείται.




Reset = '0';
Reset
PC__LdEn <= '0';
RF__WrEn <= '0';
MEM__WrEn <= '0';
Reset = '1';
A
RF__WrEn <= '0';
MEM__WrEn <= '0';
end__state
PC__LdEn <= '0';
B
PC__sel <= '0';
PC__LdEn <= '1';
RF__WrEn <= '1';
RF__WrData__sel<= '0';
ALU__func <= Instr (3 downto 0);
ALU__Bin__sel<= '0';
RF__B__sel <= '0';
C
PC__sel <= '0';
PC__LdEn <= '1';
RF__WrEn <= '1';
RF__B__sel <= '0';
ALU__Bin__sel<= '1';
ALU__func <= "1111";
li
ImmExt <= "10";
lui
ImmExt <= "01";
D
PC__sel <= '0';
PC__LdEn <= '1';
RF__WrEn <= '1';
RF__B__sel <= '0';
ALU__Bin__sel<= '1';
addi
ALU__func <= "0000";
ImmExt <= "10";
nandi
ALU__func <= "0101";
ImmExt <= "00";
ori
ALU__func <= "0011";
ImmExt <= "00";
E
PC__LdEn <= '1';
RF__WrData__sel<= '0';
ALU__Bin__sel<= '0';
ImmExt <= "11";
b
PC__LdEn <= '1';
bne
ALU__func <= "0001";
RF__B__sel <= '1';
ALU__zero='1'
ALU__zero='0'
bez
PC__sel <= '1';
beq
ALU__func <= "0001";
RF__B__sel <= '1';
ALU__zero='0'
ALU__zero='1'
bnez
PC__sel <= '0';
F
PC__LdEn <= '0';
ALU__Bin__sel<= '1';
ALU__func <= "0000";
ImmExt <= "10";
PC__sel <= '0';
l
RF__WrEn <= '1';
RF__WrData__sel <= '1';
RF__B__sel <= '0';
s
RF__WrEn <= '0';
RF__WrData__sel <= '0';
RF__B__sel <= '1';
MEM__WrEn <= '1';
lb
ByteOp <= '1'
lw
ByteOp <= '0'
sb
ByteOp <= '1'
sw
ByteOp <= '0'