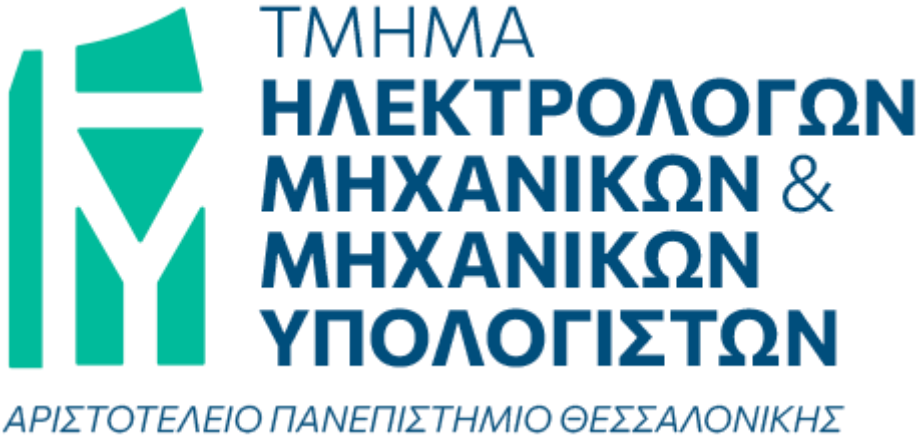


Ψηφιακά Συστήματα HW-1

Εργαστηριακές Ασκήσεις

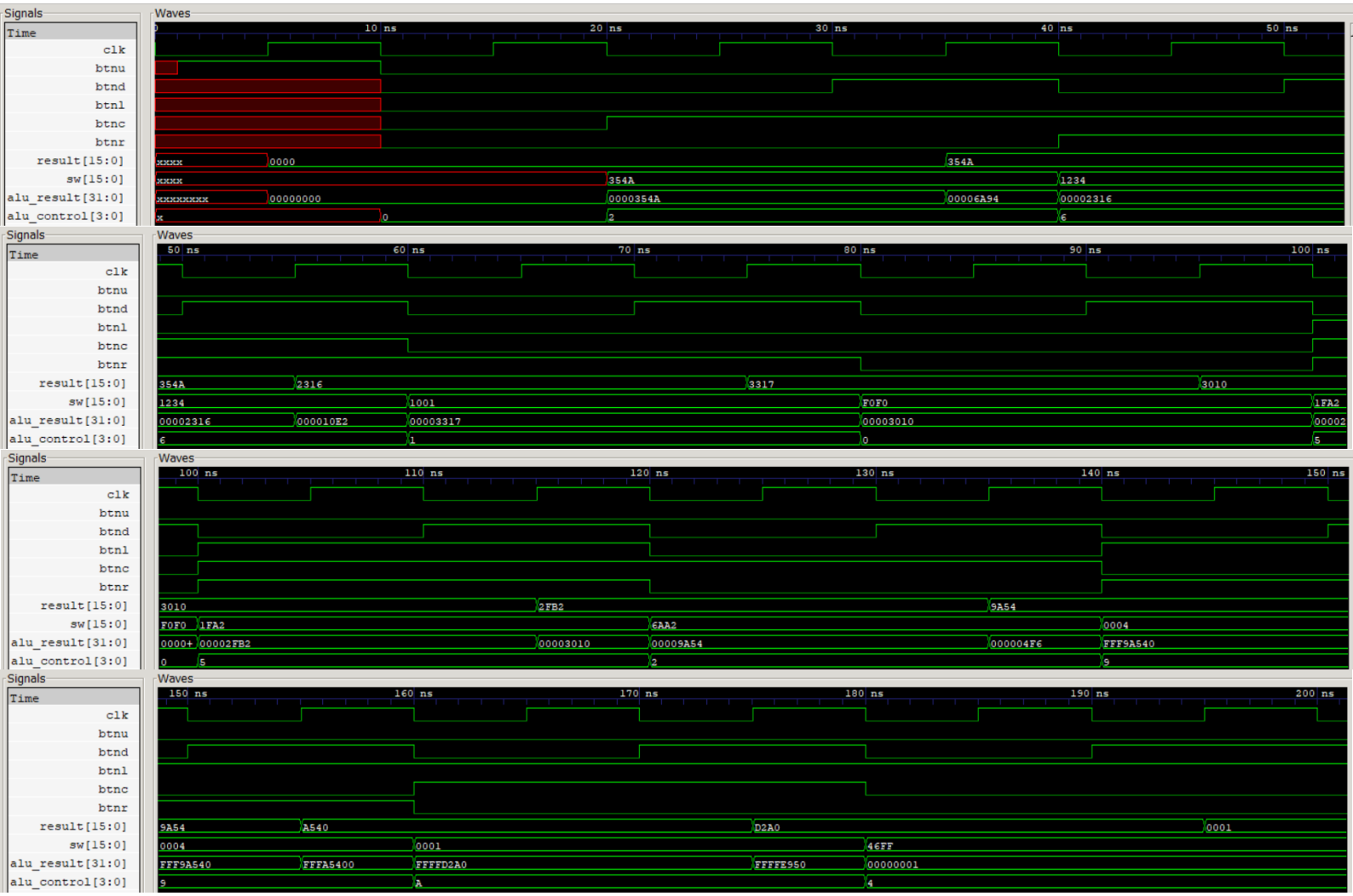
Ορθοδόξου Ιωάννης
ΑΕΜ: 10822
iorthodo@ece.auth.gr

Άσκηση 1

Σε αυτή την άσκηση θα δημιουργήσουμε μια “32-bit” ALU (Arithmetic Logic Unit) όπου θα την χρησιμοποιήσουμε σε επόμενες ασκήσεις για την υλοποίηση μιας απλής «αριθμομηχανής» αλλά και για τον επεξεργαστή RISC-V. Αρχικά ορίζουμε το module alu στο αρχείο alu.v με τις κατάλληλες θύρες και ορίζουμε 9 παραμέτρους των 4 bit, όπου κάθε παράμετρος αντιστοιχεί σε μια εντολή λειτουργίας της ALU. Έπειτα υλοποιούμε τον πολυπλέκτη της ALU, για την επιλογή της εντολής λειτουργίας, με χρήση της εντολής case όπου ανατίθεται στο σήμα result το αποτέλεσμα της κάθε εντολής. Για τις εντολές «μικρότερο από» και στη «αριθμητική ολίσθηση δεξιά» χρησιμοποιούμε την συνάρτηση $signed έτσι ώστε να μετατρέπονται σε προσημασμένους αριθμούς τα op1,op2 στην «μικρότερο από» και το op1 στην άλλη εντολή. Τέλος υπολογίζουμε το σήμα zero με χρήση του τελεστή ‘?’.

Άσκηση 2

Σε αυτή την άσκηση θα σχεδιάσουμε ένα κύκλωμα αριθμομηχανής που θα χρησιμοποιεί την ALU της άσκησης 1. Στο αρχείο calc.v ορίζουμε ένα module με όνομα calc και προσθέτουμε τις κατάλληλες θύρες. Ακόμη ορίζουμε τα εσωτερικά σήματα extend_sw, alu_result, extend_acc, alu_op1, alu_op2, alu_control, όπου είναι η επέκταση προσήμου για το σήμα sw, το αποτέλεσμα της alu, η επέκταση προσήμου του accumulator, η είσοδος της alu για το σήμα op1, η είσοδος της alu για το σήμα op2 αντίστοιχα. Για τις επεκτάσεις προσήμων χρησιμοποιούμε τον τελεστή concatenation (πχ. assign extend_sw={{16{sw[15]}},sw}). Για την σχεδίαση του καταχωρητή 16 bit χρησιμοποιούμε ένα always block οπού στην λίστα ευαισθησίας έχουμε την θετική ακμή του ρολογιού (posedge clk), και χρησιμοποιούμε το σήμα btnu για να τον μηδενίσουμε και το σήμα btnd για να τον ενημερώσουμε δηλαδή να πάρει την τιμή που έχουν τα 16 χαμηλότερα bit του αποτελέσματος της ALU. Για να παράγουμε το σήμα alu_op υλοποιούμε τα συνδυαστικά κυκλώματα της εκφώνησης σε structural Verilog, δηλαδή με χρήση των not(), and(), or(), σε ένα καινούργιο αρχείο με όνομα calc_enc.v. Για το testbench ορίζουμε το στιγμιότυπο του module calc και μέσω ενός initial block αρχικοποιούμε το clk σε 0, και έπειτα χρησιμοποιούμε το forever με καθυστέρηση 5 μονάδες χρόνου για να του αντιστρέψουμε την τιμή του. Με αυτό το τρόπο δημιουργούμε περίοδο ρολογιού 10 μονάδες χρόνου. Ακολούθως μέσω ενός άλλου initial block δημιουργούμε την τα σήματα που δίνονται στο πίνακας της εκφώνησης για να ελέγξουμε την σωστή λειτουργία της αριθμομηχανής αλλά και της ALU. Παρακάτω φαίνονται οι κυματομορφές της προσομοίωσης.



Από την προσομοίωση παρατηρούμε πως τα αποτελέσματα που έχουμε είναι τα αναμενόμενα. Το σήμα result παίρνει την τιμή του σήματος alu_result όταν το σήμα btnd είναι 1 και έρθει θετική ακμή του ρολογιού. Αυτό συμβαίνει επειδή το σήμα result είναι συνδεδεμένο με το σήμα led όπου είναι η έξοδος του accumulator. Ακόμη παρατηρούμε πως το αποτέλεσμα της alu (alu_result) αλλάζει όταν αλλάξει κάποια από της εισόδους της, αφού είναι συνδυαστικό κύκλωμα και όχι ακολουθιακό.

Άσκηση 3

Σε αυτή την άσκηση θα σχεδιάσουμε ένα αρχείο καταχωρητών όπου θα χρειαστούμε στην συνέχεια για τον επεξεργαστή RISC-V. Δημιουργούμε ένα αρχείο με όνομα regfile.v και ορίζουμε το module regfile. Ακολούθως ορίζουμε τις κατάλληλες θύρες και την παράμετρο DATAWIDTH. Για τους καταχωρητές ορίζουμε ένα πίνακα 32x32 bit. Έπειτα μέσω ενός initial block με μιας for αρχικοποιούμε τους καταχωρητές σε μηδενικά και μέσω ενός always block, με την θετική ακμή του ρολογιού στην λίστα ευαισθησία, διαβάζουμε ή/και γράφουμε στους καταχωρητές. Η προτεραιότητα στην

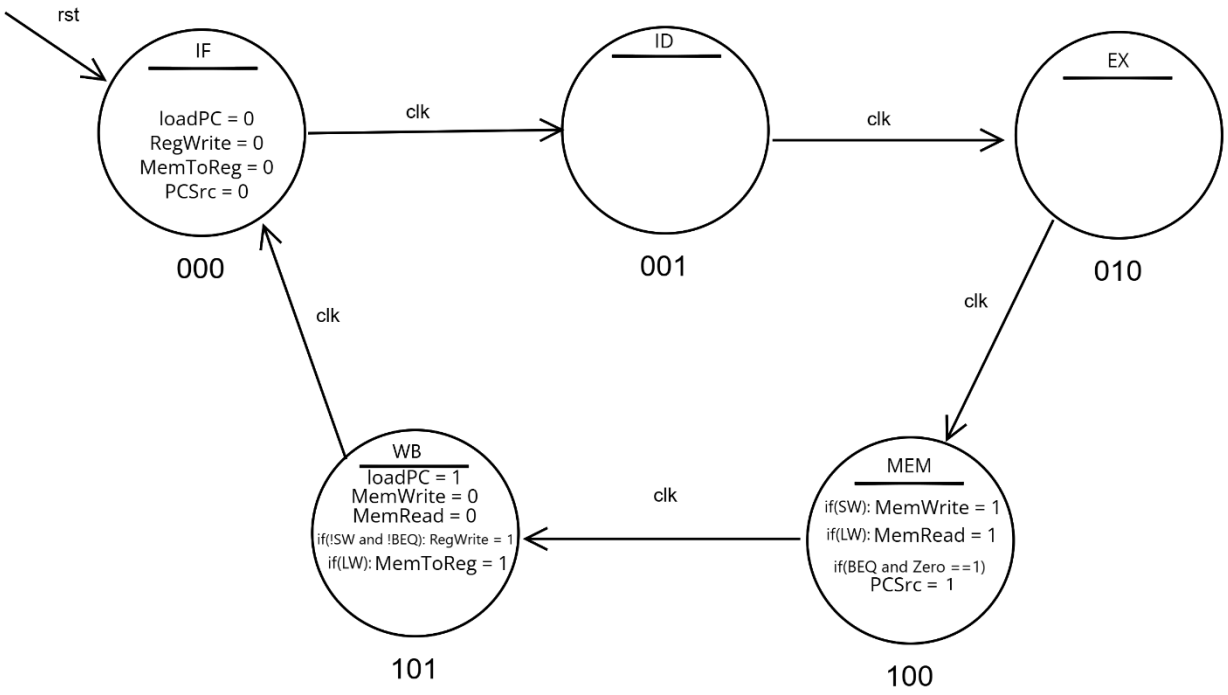
εγγραφή του “writeData” γίνεται μέσω της εντολής if όπου αν χρειάζεται να διαβάσει και να γράψει στο ίδιο καταχωρητή, θα γράψει τα δεδομένα στο καταχωρητή και θα δώσει τα δεδομένα στην αντίστοιχη έξοδο ανάγνωσης (readData1, readData2).

Άσκηση 4

Σε αυτή την άσκηση θα σχεδιάσουμε την μονάδα διαδρομής δεδομένων του επεξεργαστή. Αρχίζουμε την άσκηση με ένα αρχείο datapath.v όπου ορίζουμε το module datapath με τις κατάλληλες θύρες και την παράμετρο INITIAL_PC. Ακόμη ορίζουμε 4 παραμέτρους όπου η κάθε μια αντιστοιχεί στο opcode των κύριων εντολών (I,R,S,B types), και ορίζουμε εσωτερικά σήματα (opcode, rd, rs1, rs2, ext_imm, imm, branch_offset, read_data1, read_data2, alu_result, op2_alu). Έπειτα μέσω ενός always block καθορίζουμε την τιμή του PC, και στην συνέχεια θέτουμε τα εσωτερικά σήματα (opcode, rd, rs1, rs2) να παίρνουν τα κατάλληλα bits του σήματος instr. Για να θέσουμε το σήμα imm χρησιμοποιούμε τον τελεστή ‘?’ και ανάλογα ποιος τύπος εντολής είναι, παίρνει τα κατάλληλα bits, πάλι από το σήμα instr, ενώ το σήμα ext_imm είναι η επέκταση προσήμου του σήματος imm. Ακολούθως αρχικοποιούμε τα regfile, και alu, δίνοντας τα κατάλληλα σήματα και υπολογίζουμε το branch_offset. Τέλος θέτουμε τις εξόδους, dAddress να είναι ίση με το σήμα alu_result ,την dWriteData να είναι ίση με το σήμα read_data2, και την WriteBackData ανάλογα από την εντολή θα είναι είτε το σήμα alu_result είτε το dReadData.

Άσκηση 5

Σε αυτή την άσκηση θα σχεδιάσουμε ένα ελεγκτή πολλαπλών κύκλων όπου θα εκτελεί κάθε εντολή σε πέντε κύκλους ρολογιού. Αρχίζουμε την άσκηση με ένα αρχείο top_proc.v όπου ορίζουμε το module top_proc με τις κατάλληλες θύρες και την παράμετρο INITIAL_PC. Έπειτα ορίζουμε πέντε παραμέτρους όπου αντιστοιχούν στις πέντε καταστάσεις του FSM και δύο εσωτερικά σήματα όπου θα έχουν την τιμή της τρέχουσας και της επόμενης κατάστασης (current_state, next_state). Ακόμη ορίζουμε τα εσωτερικά σήματα ελέγχου όπου θα είναι οι έξοδοι του FSM. Στην συνέχεια μέσω τριών always block σχεδιάζουμε το FSM. Στο πρώτο block ορίζουμε ένα σύγχρονο reset και θέτουμε την τιμή της επομένης κατάστασης στο σήμα της τρέχουσας. Στο δεύτερο δημιουργούμε την λογική για την τιμή της επόμενης κατάστασης. Ακολούθως στο τελευταίο block ορίζουμε της τιμές των σημάτων ελέγχουν που πρέπει να παίρνουν ανάλογα με την κατάσταση που βρίσκονται. Τέλος ορίζουμε τις τιμές των σημάτων ελέγχου της ALU (ALUSrc, ALUCtrl), όπου δεν εξαρτώνται από την κατάσταση που βρίσκεται το FSM, και αρχικοποιούμε το κύκλωμα της διαδρομής δεδομένων με τα κατάλληλα σήματα. Το σχετικό διάγραμμα FSM δίνετε πιο κάτω.

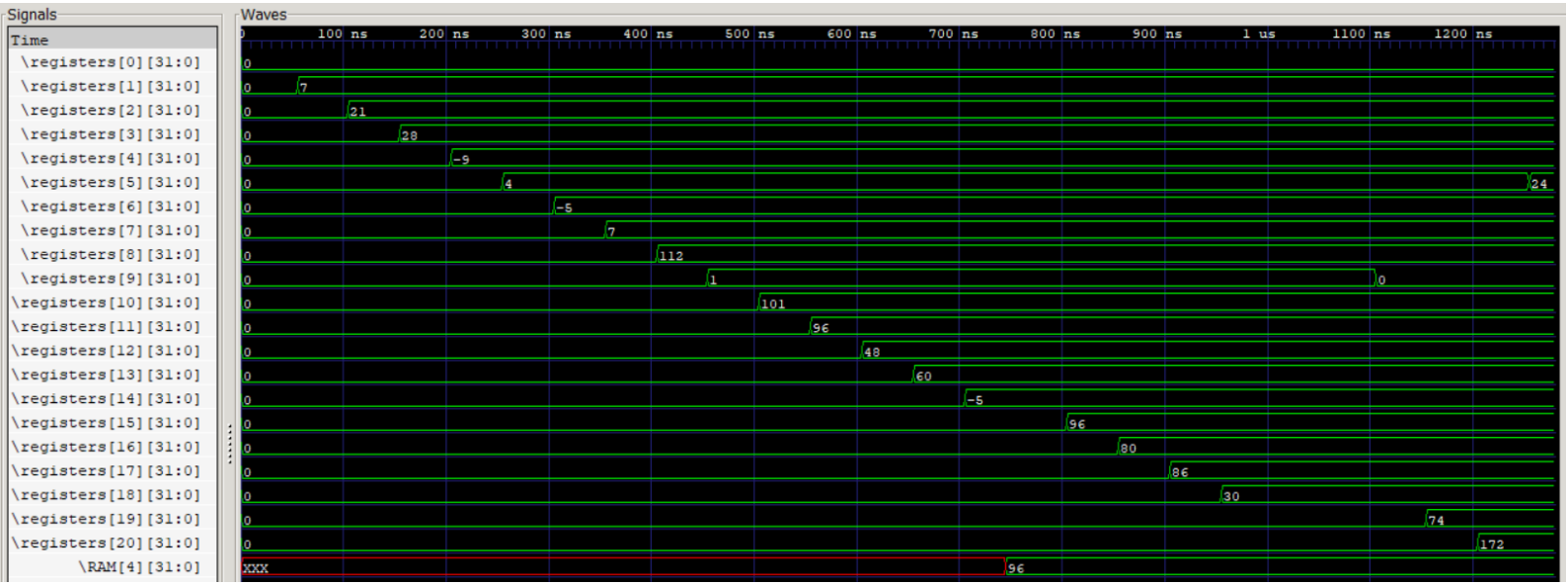


Ακόμη οι εντολές που βρίσκονται στο αρχείο rom_bytes.data είναι οι ακόλουθες σε assembly:

1. I-type: addi x1, x0, 7, Αναμενόμενη τιμή: x1 = 7
2. I-type: addi x2, x0, 21, Αναμενόμενη τιμή: x2 = 21
3. R-type: add x3, x1, x2, Αναμενόμενη τιμή: x3 = 28
4. I-type: addi x4, x0, -9, Αναμενόμενη τιμή: x4 = -9
5. I-type: addi x5, x2, -17, Αναμενόμενη τιμή: x5 = 4
6. R-type: add x6, x5, x4, Αναμενόμενη τιμή: x6 = -5
7. R-type: sub x7, x3, x2, Αναμενόμενη τιμή: x7 = 7
8. R-type: sll x8, x7, x5, Αναμενόμενη τιμή: x8 = 112
9. R-type: slt x9, x4, x8, Αναμενόμενη τιμή: x9 = 1
10. R-type: xor x10, x8, x2, Αναμενόμενη τιμή: x10 = 101
11. R-type: and x11, x10, x8, Αναμενόμενη τιμή: x11 = 96
12. R-type: srl x12, x11, x9, Αναμενόμενη τιμή: x12 = 48
13. R-type: or x13, x12, x3, Αναμενόμενη τιμή: x13 = 60
14. R-type: sra x14, x4, x9, Αναμενόμενη τιμή: x14 = -5
15. S-type: sw x11, 0(x5), Αναμενόμενη τιμή: RAM [4] = 96
16. I-type: lw x15, 0(x5), Αναμενόμενη τιμή: x15 = 96
17. I-type: andi x16, x8, -45, Αναμενόμενη τιμή: x16 = 80
18. I-type: ori x17, x16, 22, Αναμενόμενη τιμή: x17 = 86
19. I-type: srli x18, x13, 1, Αναμενόμενη τιμή: x18 = 30
20. B-type: beq x15, x11, 16, Αναμενόμενη τιμή: PC = PC +16

21. 0
22. 0
23. 0
24. 0
25. I-type: slti x9, x18, 15, Αναμενόμενη τιμή: x9 = 0
26. I-type: xori x19, x8, 58, Αναμενόμενη τιμή: x19 = 74
27. I-type: slli x20, x17, 1, Αναμενόμενη τιμή: x20 = 172
28. I-type: srai x5, x15, 2, Αναμενόμενη τιμή: x5 = 24
29. 0
30. .
31. .
32. .
33. 0

Τέλος για το αρχείο testbench, αρχικοποιούμε τα κυκλώματα της μνήμης RAM και ROM και το κύκλωμα του ελεγκτή. Μέσω ενός always block θέτουμε την περίοδο του ρολογιού να είναι 10 μονάδες χρόνου και έπειτα κάνουμε reset τα κυκλώματα και αφήνουμε να τρέξει η προσομοίωση για 1270 μονάδες χρόνου (24 εντολές * 5 κύκλους ρολογιού * 10 μονάδες χρόνου μια περίοδος + 70). Παρακάτω φαίνονται οι κυματομορφές των 21 πρώτων καταχωρητών και η πέμπτη θέση μνήμης στην RAM (η μοναδική που αλλάζει).



Από τις κυματομορφές παρατηρούμε πως οι καταχωρητές και η θέση μνήμης παίρνουν τις αναμενόμενες τιμές όπως έχουμε υπολογίσει πιο πάνω.

Σημείωση: Όταν εκτελείτε η εντολή BEQ στο PC προστίθεται +16 όπου κάνει jump 4 εντολές συμπεριλαμβανομένης και της BEQ εντολής (αγνοεί μόνο 3 εντολές) με αποτέλεσμα να καταλήγει σε μια εντολή με μηδενικά bits. Αυτό φαίνεται από την χρονική απόσταση που γίνεται η εγγραφή στους καταχωρητές 19 (x18) και 10 (x9) όπου είναι πολύ μεγαλύτερη από ότι συνήθως.

Σημείωση 2: Η εργασία έχει γίνει με χρήση Icarus Verilog και με gtkWave για την εμφάνιση των κυματομορφών.