Ψηφιακά Συστήματα HW σε χαμηλά επίπεδα λογικής Ι

Φοιτήτρια : Κωνσταντίνου Αμαλία
ΑΕΜ : 10745
Επιβλέπων καθηγητής : Παυλίδης
Ημερομηνία :

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή

Άσκηση 1

Στο πρώτο κομμάτι της εργασίας υλοποιήθηκε μια Αριθμητική-Λογική Μονάδα (ALU) χρησιμοποιώντας την HDL Verilog. Αυτή αξιοποιήθηκε αργότερα στον επεξεργαστή RISC-V.

1.1 Δομή της ALU

Η ALU αποτελείται από:

- Είσοδοι : Τελεστές των 32-bit op1 και op2, Είσοδος των 4-bit alu_op (δείχνει ποια λειτουργία πραγματοποιείται).
- Έξοδοι : 32-bit result, έξοδος zero (δείχνει ότι το αποτέλεσμα είναι μηδέν).

1.2 Λειτουργίες της ALU

Οι λειτουργίες που εκτελούνται είναι:

- Λογικές πράξεις : AND, OR, Logic Shift Right, Logic Shift Left, XOR.
- Αριθμητικές πράξεις : Πρόσθεση, Αφαίρεση, Μικρότερο από, Arithmetic Shift Right.

Η ALU δέχεται δύο εισόδους (op1, op2) και εκτελεί την επιλεγμένη λειτουργία σύμφωνα με το σήμα εισόδου alu_op. Όρισα σταθερές με τη χρήση παραμέτρων για τον καθορισμό της εντολής που θα εκτελεστεί κάθε φορά.

Ιδιαίτερη προσοχή δόθηκε στην πράξη μικρότερο από όπου οι είσοδοι μετατράπηκαν πρώτα σε προσημασμένους αριθμούς, έτσι ώστε το αποτέλεσμα να είναι έγκυρο. Όταν η είσοδος op1 είναι μικρότερη από την op2, το πρόγραμμα επιστρέφει 1. Στην αντίθετη περίπτωση, επιστρέφει 0.

Επιπλέον, για την αριθμητική ολίσθηση, το op1 μετατράπηκε σε προσημασμένο και το αποτέλεσμα πίσω σε μη-προσημασμένο αριθμό.

Άσκηση 2

Στο δεύτερο κομμάτι της εργασίας σχεδίασα ένα κύκλωμα αριθμομηχανής το οποίο χρησιμοποιεί την ALU της πρώτης άσκησης. Ανάλογα με τα κουμπιά που πατάει ο χρήστης, η αριθμομηχανή εκτελεί διαφορετικές πράξεις. Το αποτέλεσμα απεικονίζεται στα LEDs που ενημερώνονται ταυτόχρονα με τον συσσωρευτή . Για την υλοποίηση του κυκλώματος δημιούργησα ένα αρχείο calc.v στο οποίο έκανα include το alu.v.

2.1 Δομή της calc

Η calc περιέχει:

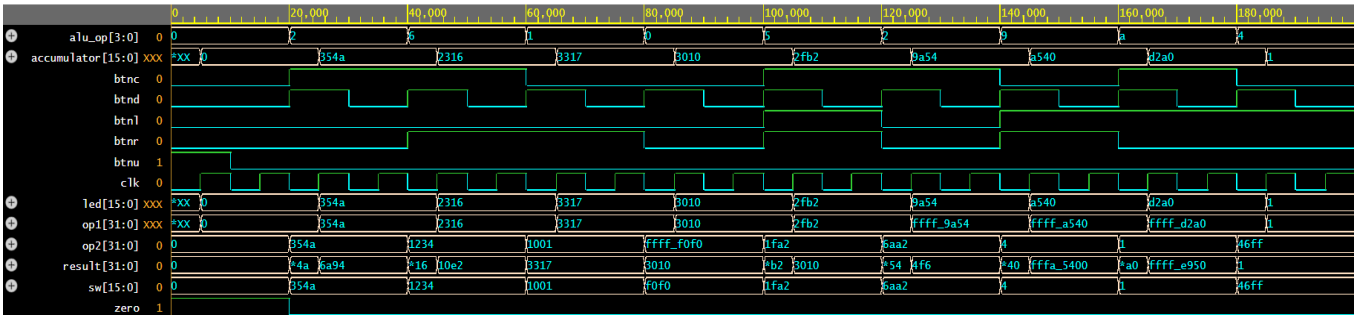
- ALU για την πραγματοποίηση των αριθμητικών και λογικών πράξεων
- Συσσωρευτή για την αποθήκευση των αποτελεσμάτων των πράξεων

- Εισόδους και εξόδους του κυκλώματος (κουμπιά, LEDs)

Το νέο αρχείο είχε ως εισόδους τα btnc, btnl,btnu, btnr btnd και ως έξοδο τα LEDs.
Η κυματομορφή που προκύπτει φαίνεται παρακάτω. Σε αυτό φαίνεται πρώτα η πράξη που εκτελείται κάθε φορά (alu_op[3:0]), η οποία αλλάζει με βάση τις καταστάσεις των πλήκτρων. Έπειτα ο συσσωρευτής 16-bit (accumulator), ο οποίος αποθηκεύει την τρέχουσα τιμή της αριθμομηχανής. Συγκεκριμένα, όταν πατιέται το btnu μηδενίζεται και όταν πατιέται το btnd ενημερώνεται με τα χαμηλότερα 16-bit του αποτελέσματος. Παρακάτω, οι είσοδοι από τα πλήκτρα και οι δύο είσοδοι της ALU, op1 και op2. Συγκεκριμένα, η op1 αντιστοιχεί στην τιμή του συσσωρευτή που έχει επεκταθεί στα 32 bits. Η op2 είναι η τιμή των διακοπτών που επίσης επεκτείνεται στα 32 bits. Το αποτέλεσμα της ALU, που φαίνεται στο διάγραμμα παρακάτω, αλλάζει σύμφωνα με τις εισόδους της και την alu_op.

Επιπλέον, δημιουργήθηκε το module calc_enc, το οποίο κάνουμε instantiate στον κύριο κώδικα, προσέχοντας να συνδέσουμε τις αντίστοιχες πύλες. Αυτό αναθέτει τιμές στο alu_op, στο opcode της alu, με βάση τις τιμές των btnr, btnc, btnl. Από τα διαγράμματα που μας δόθηκαν, προέκυψαν οι σχέσεις για τα σήματα alu_op[], δηλαδή ποιά πράξη θα εκτελεστεί κάθε φορά.

Σε αυτό το τμήμα της εργασίας υλοποιήθηκε και ένα testbench calc_tb το οποίο έχει ως στόχο την επαλήθευση της συμπεριφοράς του κώδικα μου. Συνέδεσα το υπό έλεγχο μοντέλο uut κάνοντας port mapping των εισόδων και των εξόδων. Παράχθηκε το σήμα ρολογιού με περίοδο 10ns. Συνέχισα με την δημιουργία ενός initial block, στο οποίο αρχικοποίησα τις τιμές των κουμπιών και του 16-μπιτ καταχωρητή sw στο 0. Μέσα σε αυτό έτρεξα ελέγχους για διάφορα είδη πράξεων, όπως των ADD, SUB, OR, AND κλπ. Πήρα τα αναμενόμενα αποτελέσματα από το πρόγραμμά μου για τους ελέγχους αυτούς. Τέλος, σημιουργήθηκε το αρχείο calc_tb.vcd για την αποθήκευση των κυματομορφών.



Άσκηση 3

Στο τμήμα αυτό υλοποίησα ένα αρχείο καταχωρητών, το οποίο αποθηκεύει τις τιμές των καταχωρητών που χρησιμοποιούνται από τον επεξεργαστή. Παρέχει παράλληλα και την δυνατότητα τα δεδομένα που μόλις εγγράφηκαν να χρησιμοποιηθούν αμέσως. Στην

υλοποίηση αυτή είχα ως θύρες το ρολόι, ένα σήμα εγγραφής, δύο θύρες ανάγνωσης δεδομένων, μία διεύθυνση θύρας εγγραφής , τα δεδομένα προς εγγραφή και τα δεδομένα ανάγνωσης από τις δύο θύρες.

Δημιούργησα έναν πίνακα 32xDatawidth, που είναι το αρχείο των καταχωρητών. Αρχικοποίησα τον πίνακα reg_file σε τιμή 32'b0. Σε ένα always block όρισα ότι η εγγραφή δεδομένων στο reg_file και συγκεκριμένα στην διεύθυνση writeReg γίνεται στις θετικές ακμές του ρολογιού και μόνο όταν το σήμα write, εγγραφής, είναι ενεργό. Σε ένα άλλο always block όρισα την λογική που ακολουθεί ο κώδικας για την ανάγνωση. Όπως μας ζητήθηκε αν η διεύθυνση ανάγνωσης είναι ίση με την διεύθυνση εγγραφής και το σήμα write είναι ενεργό, τότε τα δεδομένα λαμβάνονται απευθείας από το writeData, αλλιώς από τον πίνακα reg_file.

Για επιπλέον έλεγχο δημιουργήθηκε ένα testbench το οποίο επαληθεύει τη σωστή λειτουργία του αρχείου καταχωρητή. Συνέδεσα το υπό έλεγχο μοντέλο uut κάνοντας port mapping των εισόδων και των εξόδων. Παράγεται σήμα ρολογιού με περίοδο 10ns.

Έτρεξα δοκιμές :

- Πρώτη της εγγραφής στον καταχωρητή, όπου ενεργοποιείται το σήμα εγγραφής.
  ```
  write = 1
  writeReg = 5'd1; // Διεύθυνση εγγραφής 1
  writeData = 32'hDEADBEEF;
  ```

- Δεύτερη της ανάγνωσης από τον καταχωρητή, όπου τα δεδομένα που είναι αποθηκευμένα στον καταχωρητή διαβάζονται.
  ```
  write = 0;
  readReg1 = 5'd1;
  ```

- Τρίτη την ταυτόχρονη εγγραφή στον καταχωρητή και ανάγνωση από αυτόν.

- Τέταρτη την προτεραιότητα εγγραφής, με την οποία επαληθεύεται ότι τα δεδομένα διαβάζονται και πριν την ολοκλήρωση της εγγραφής.

Άσκηση 4

Συνεχίσαμε με την σχεδίαση της μονάδας διαδρομής δεδομένων του επεξεργαστή. Αυτή αποτελεί καθοριστικό κομμάτι του επεξεργαστή μας καθώς συνδέει τις μονάδες λειτουργίας ALU και Registers που δημιουργήσαμε στις προηγούμενες ασκήσεις. Καθορίζει, δηλαδή, τη ροή των δεδομένων μεταξύ των μονάδων alu, regfile και της μνήμης. Απαραίτητο για την υλοποίησή της ήταν να συμπεριλάβουμε την alu και το regfile που δημιουργήσαμε στην πρώτη και στην τρίτη άσκηση αντίστοιχα. Επιπλέον ορίσαμε τις θύρες του module όπως μας δόθηκαν στην εκφώνηση της άσκησης.

Πρώτα, έγινε instantiate του module regfile. Το σχήμα 7 της άσκησης μας βοήθησε να καταλάβουμε τις σχέσεις που συνδέουν τα δεδομένα. Συγκεκριμένα, διαπιστώσαμε ότι το instr[19:15] είναι είσοδος στο ReadReg1, το instr[24:20] στο ReadReg2, το instr[11:7] στο

WriteReg. Έχουμε ανάγνωση δύο καταχωρητών rs1, rs2 και εγγραφή στον καταχωρητή rd, μόνο στην περίπτωση που το σήμα RegWrite είναι ενεργό (αντιστοιχεί στο σήμα Write του regfile που αναφέρθηκε προηγουμένως).

Επιπλέον, ορίσαμε τα είδη των εντολών I, S, B σύμφωνα με το pdf που μας δόθηκε. Καθεμία δέχεται ως είσοδο ένα τμήμα της εντολής instr και κάνει επέκταση προσήμου έτσι ώστε να έχει μήκος 32 bit.

Έπειτα, ασχοληθήκαμε με τον τρόπο που ενημερώνεται ο καταχωρητής PC. Όταν το σήμα ελέγχου loadPC ενεργοποιηθεί το PC θα πρέπει να ενημερωθεί στην επόμενη ακμή ρολογιού. Αν το σήμα Reset ενεργοποιηθεί, τότε ο Program Counter παίρνει την αρχική του τιμή. Διαφορετικά, παίρνει την επόμενη τιμή του. Η τελευταία καθορίζεται από το PCSrc. Το PCSrc είναι σήμα ελέγχου που ενεργοποιείται όταν εκτελείται μια εντολή τύπου Branch (B). Όταν αυτό είναι ενεργό θα γίνει (PC + branch offset), δηλαδή θα προκύψει διακλάδωση. Αλλιώς, θα πάρει τιμή (PC+4), δηλαδή προχωρά στην επόμενη εντολή.

Έγινε instantiate του module ALU. Η δεύτερη είσοδος της ALU, για ενεργό σήμα ελέγχου ALUSrc είναι το πεδίο immediate της εντολής, αλλιώς είναι το readData2. Χρησιμοποιώντας τις δύο εισόδους της η ALU εκτελεί αριθμητικές ή λογικές πράξεις, έτσι προκύπτει το αποτέλεσμα aluResult.

Τέλος, έγινε ανάθεση τιμών στις εξόδους του module. Συγκεκριμένα, η έξοδος dAddress πήρε την τιμή της εξόδου της ALU. Ένω η dWriteData-που συμβολίζει τα δεδομένα που θα εγγραφούν στη μνήμη-πήρε την τιμή των δεδομένων ανάγνωσης της δεύτερης θύρας του καταχωρητή. Στο τμήμα Write Back επιλέγεται το αποτέλεσμα που θα γραφτεί στους καταχωρητές, τα δεδομένα που διαβάστηκαν από τη μνήμη (MemToReg ενεργό) ή η έξοδος της ALU.

Άσκηση 5

Στο τμήμα αυτό υλοποιήσαμε έναν επεξεργαστή RISC, το top_proc.v, το οποίο συνδυάζει όλες τις προηγούμενες ενότητες. (alu, datapath, registers, ram, rom). Εμπεριέχει μια μηχανή πεπερασμένων καταστάσεων (FSM) για τον έλεγχο των σταδίων :

1. IF (Instruction Fetch): Ανάγνωση εντολής από τη μνήμη και ο program counter αυξάνεται κατά 4.
2. ID (Instruction Decode): Αποκωδικοποίηση της εντολής και αποφασίζεται αν τα δεδομένα προέρχονται από καταχωρητή ή από άμεση τιμή.
3. EX (Execute):Η ALU εκτελεί τη λειτουργία που καθορίζεται από το πεδίο funct3 και funct7 της εντολής.
4. MEM (Memory Access): Ανάγνωση ή εγγραφή από/στη μνήμη (lw, sw αντίστοιχα).
5. WB (Write Back): Τα δεδομένα που προκύπτουν από την ALU ή τη μνήμη εγγράφονται στους καταχωρητές.

Για την υλοποίηση της άσκησης ξεκινήσαμε με τη δημιουργία του FSM, της μηχανής καταστάσεων, η οποία καθορίζει την πορεία που ακολουθούν οι καταστάσεις (IF->ID->EX->MEM->WB). Σε κάθε κύκλο ρολογιού μεταβαίνουμε στην επόμενη κατάσταση.

Η αρχική κατάσταση είναι η IF, αφού διαβαστεί η εντολή μεταβαίνουμε στην ID. Μετά την αποκωδικοποίηση πηγαίνουμε στην EX, όπου εκτελούνται οι εντολές. Έπειτα μεταβαίνουμε στη WB για την εγγραφή των αποτελεσμάτων στους καταχωρητές. Η default κατάσταση είναι η IF.

Στην κατάσταση ID με βάση την τιμή του opcode διαπιστώνουμε τι είδους εντολή έχουμε και δίνουμε τιμή στο σήμα ALUSrc, το οποίο καθορίζει την τιμή της δεύτερης εισόδου της alu. Δηλαδή για ALUSrc=1 η είσοδος της είναι τα άμεσα δεδομένα, ενώ για ALUSrc=0 είναι η τιμή readData2. Η ALUSrc θα είναι 1 για τις εντολές τύπου I, S (όπου χρειαζόμαστε τα άμεσα δεδομένα της alu). Αντίθετα, θα είναι 0 για τις B, όπου χρησιμοποιούμε καταχωρητές. Στην κατάσταση EX το σήμα ελέγχου ALUCtrl καθορίζει την λειτουργία που θα εκτελέσει κάθε φορά η ALU. Στην περίπτωση των εντολών διακλάδωσης BEQ, βάζουμε PCSrc = Zero. Οπότε όπως μας ζητείται το PCSrc είναι 1 για εντολές διακλάδωσης με τους δύο καταχωρητές ίσους.

Στην κατάσταση MEM χρησιμοποιούμε το σήμα MemRead_reg (δίνει τιμή στο MemRead), που γίνεται 1 για load εντολές και το MemWrite_reg (δίνει τιμή στο MemWrite), που γίνεται 1 για store εντολές.

Στην κατάσταση WB το σήμα RegWrite τίθεται ίσο με 1 για τις εντολές που απαιτούν εγγραφή, ώστε τα δεδομένα να εγγραφούν στο αρχείο. Ενώ, το σήμα MemToReg γίνεται 1 μόνο για τις load εντολές. Επίσης, το LoadPC θα γίνει 1 που δηλώνει ότι πρέπει να περαστεί νέα τιμή στον program counter για να περάσουμε στην κατάσταση IF.

Επιπλέον, κάναμε instantiate το module datapath, τη διαδρομή δεδομένων, που αποτελεί βασικό μέρος του επεξεργαστή. Βλέπουμε την σύνδεση που υπάρχει ανάμεσα στην τελευταία και το FSM.

Για την αποκωδικοποίηση των εντολών λαμβάνουμε υπόψη ότι μία εντολή αντιστοιχεί σε 4 γραμμές από το αρχείο rom_bytes.data, ώστε να σχηματιστούν οι 32 bit εντολές που δέχεται το RISC-V.
Επομένως, η πρώτη εντολή θα είναι η 00000000 01110000 00000000 10010011, που αντιστοιχεί σε ADDI λειτουργία (opcode 0010011). Ενώ η immediate θα είναι 7. Άρα, παίρνουμε **addi x1, x0, 7** (x1=7)**.**
Παίρνουμε για την δεύτερη εντολή 00000001010100000000000100010011, σειρές 5 έως 9, όπου προκύπτει **addi x2, x0, 21** (x2=21). Παρόμοια δουλεύουμε και για τις υπόλοιπες και παίρνουμε:

9 : add x3, x1, x2 Expect: x3 = 28
13 : addi x4, x0, -9 Expect: x4 = -9
17 : addi x5, x2, -17 Expect: x5 = 4

21 : add x6, x5, x4 Expect: x6 = -5
25 : sub x7, x3, x2 Expect: x7 = 7
29 : sll x8, x7, x5 Expect: x8 = 112
33 : slt x9, x4, x8 Expect: x9 = 1
37 : xor x10, x8, x2 Expect: x10 = 101
41 : and x11, x10, x8 Expect: x11 = 96
45 : srl x12, x11, x9 Expect: x12 = 48
49 : or x13, x12, x3 Expect: x13 = 60
53 : sra x14, x4, x9 Expect: x14 = -5
57 : sw x11, 0(x10) Expect: RAM[4] = 96
61 : lw x15, 0(x10) Expect: x15 = 96
65 : andi x16, x8, -45 Expect: x16 = 80
69 : ori x17, x16, 22 Expect: x17 = 86
73 : srli x18, x13, 1 Expect: x18 = 30
77 : beq x15, x11, 16 Expect: PC = PC + 16

97: slti x9, x18, 15 Expect: x9 = 0
101: xori x19, x8, 58 Expect: x19 = 74
105: slli x20, x17, 1 Expect: x20 = 172
109: srai x5, x15, 2 Expect: x5 = 24

Στην συνέχεια, υλοποιήθηκε testbench για τη μονάδα αυτή που έκανε instantiate τα rom.v (μνήμη εντολών), ram.v (μνήμη δεδομένων) και έτρεχε προσομοίωση για το κύκλωμα. Ορίστηκαν σήματα που αντιστοιχούν στις θύρες του επεξεργαστή και χρησιμοποιήθηκαν κατά την σύνδεση του uut με το testbench. Παράχθηκε το σήμα ρολογιού με περίοδο 10ns. Σε ένα initial block αρχικοποιήθηκαν τα σήματα ckl και reset (ίσο με 1 για να γίνει επαναφορά του επεξεργαστή στην αρχική κατάσταση) και μετά την πάροδο 10ns μηδενισμός του reset. Το testbench έτρεχε τις εντολές που εισήχθησαν στο rom.v από το rom_bytes.data. Το τελευταίο περιέχει 512 σειρές από 8-μπιτ αριθμούς, οι οποίες συνδυάζονται ανά τέσσερις και σχηματίζουν τις 32-μπιτ εντολές που δέχεται το κύκλωμα μας. Τέλος, δημιουργήθηκε το αρχείο top_proc_tb.vcd στο οποίο αποθηκεύεται η κυματομορφή που παρατίθεται παρακάτω.

Ας συνεχίσουμε με την ανάλυση των δεδομένων που παίρνουμε.

1η εντολή : `5 PC=0, instr=00000000011100000000000010010011, dAddress=00000000, dWriteData=00000000, MemRead=0, MemWrite=0, WriteBackData=00000000`
Εκτελείται η ADDI x1, x0, 7. Τα MemRead και MemWrite παραμένουν 0 αφού δεν απαιτείται πρόσβαση στη μνήμη. Την χρονική στιγμή t=15 ο PC ενημερώνεται και παίρνει την τιμή 4.

2η εντολή : `25 PC=4, instr=00000001010100000000000100010011, dAddress=00000015, dWriteData=00000000, MemRead=0, MemWrite=0, WriteBackData=00000015`

Εκτελείται ADDI x2, x0, 21. Τα MemRead και MemWrite παραμένουν 0 αφού δεν απαιτείται πρόσβαση στη μνήμη. Το WriteBackData δείχνει 21 όπως αναμέναμε.

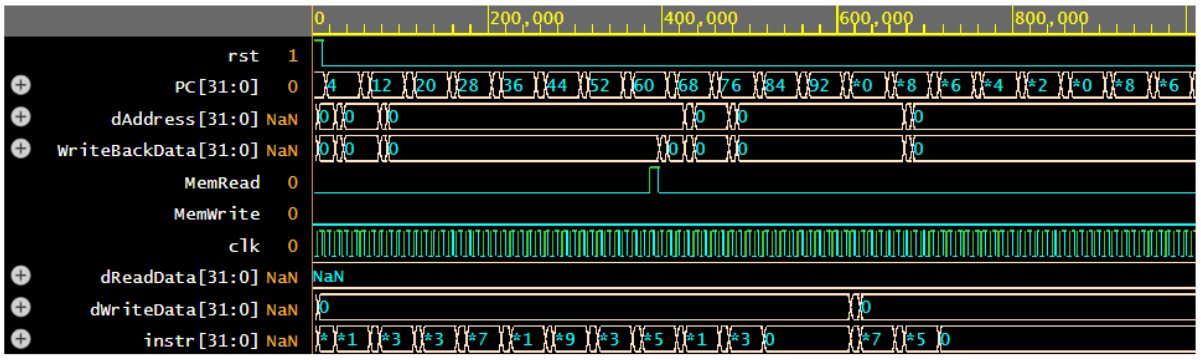
4η εντολή : `75 PC= 12, instr=11111111011100000000001000010011, dAddress=fffffff7, dWriteData=00000000, MemRead=0, MemWrite=0, WriteBackData=fffffff7`
Εκτελείται ADDI x4, x0, -9. Ο PC έχει την αναμενόμενη τιμή αφού δεν είχαμε διακλαδώσεις. Το WriteBackData ενημερώνεται σωστά με την τιμή `fffffff7` που αντιστοιχεί σε -9 στο 32-bit σύστημα.

…

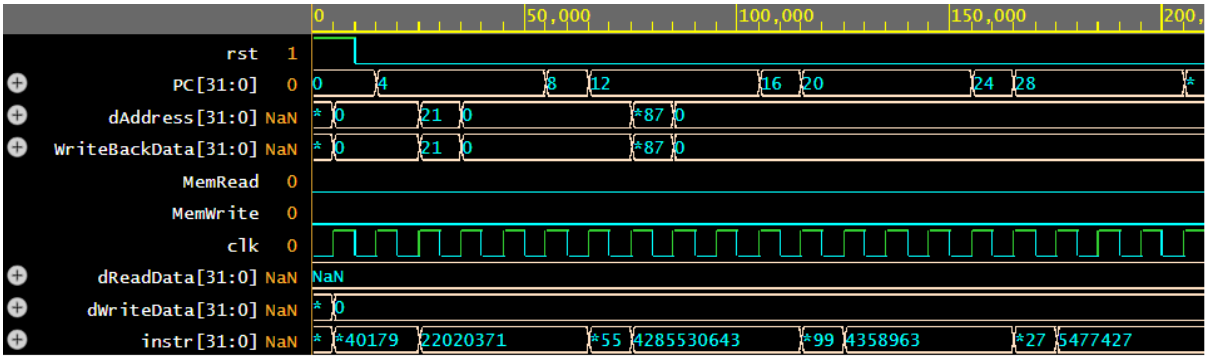
13η εντολή : `365 PC= 60, instr=00000001011001010100000000100011, dAddress=00000000, dWriteData=00000000, MemRead=0, MemWrite=0, WriteBackData=00000000`
Προσοχή αξίζει να δοθεί στην εντολή sw x11, 0(x10), η οποία είναι τύπου store λόγω του opcode 0100011. Αποθηκεύει τη λέξη δεδομένων από τον καταχωρητή x11 στη μνήμη. Παρατηρούμε ωστόσο ότι δεν πραγματοποιείται η εγγραφή όπως θα περιμέναμε (αφού το MemWrite παραμένει απενεργοποιημένο).

14η εντολή : `385 PC= 60, instr=00000000000001010100111110000011, dAddress=00000000, dWriteData=00000000, MemRead=1, MemWrite=0, WriteBackData=00000000`
Αξιοσημείωτο σε αυτή την εντολή lw x15, 0(x10) είναι ότι φορτώνουμε τη λέξη δεδομένων από την μνήμη και την αποθηκεύουμε στον καταχωρητή 15. Για αυτόν το λόγο παίρνουμε MemRead=1.

Τα διαγράμματα που λαμβάνουμε είναι :



Αναλυτικότερα, σε μικρότερο χρονικό διάστημα:



Στο διάγραμμα βλέπουμε ότι το σήμα rst στην αρχή είναι ίσο με 1 και μετά μηδενίζεται. Επιπλέον, φαίνονται καθαρά οι σχέσεις ανάμεσα στις εντολές και στις θέσεις που παίρνει ο PC. Ιδιαίτερα βλέπουμε πως για εντολές τύπου ADD, ADDI, ANDI, SUB όπου δεν απαιτείται διακλάδωση ο PC αυξάνεται κατά 4 κάθε φορά. Επιπλέον, όπως αναμέναμε από την σχεδίαση του κυκλώματός μας οι εντολές λαμβάνονται στις θετικές ακμές του ρολογιού αφού αυτό είναι θετικά ακμοπυροδότητο.