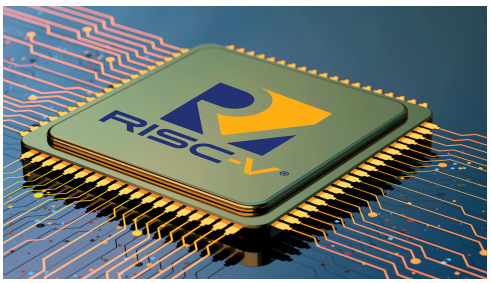
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ψηφιακά συστήματα HW-1

Εργαστηριακές Ασκήσεις



Σκλαβενίτης Γεώργιος
ΑΕΜ: 10708

Χειμερινό Εξάμηνο 2024-2025

Περιεχόμενα

1 Άσκηση 1

Στην άσκηση 1 ζητείται να υλοποιήσουμε μια αριθμητική/λογική μονάδα (ALU) για τον επεξεργαστή RISC-V. Το κύκλωμα είναι συνδιαστικό καθώς η έξοδος εξαρτάται μόνο από τις εισόδους και δεν υπάρχει μνήμη για αποθήκευση καταστάσεων. Ξεκινάμε δημιουργώντας το module `alu`, και ορίζοντας τις εισόδους και τις εξόδους του.

- Ορίζουμε τις παραμέτρους των πράξεων ως `ALUOP_*` με τιμές όπως δίνονται από τον πίνακα της εκφώνησης.
- Με ένα always block, με ευαισθησία σε οποιαδήποτε αλλαγή των εισόδων, και την εντολή case, η τιμή του σήματος εισόδου `alu_op` είναι αυτή που καθορίζει ποια πράξη θα εκτελέσουν οι τελεστές εισόδου `op1` και `op2`, σύμφωνα με τις τιμές των παραμέτρων που ορίσαμε.
- Η πράξη "μικρότερο από" (`LT`) χρησιμοποιεί προσημασμένη σύγκριση των δύο τελεστών.
- Η αριθμητική ολίσθηση (`SRA`) διατηρεί το πρόσημο του `op1` κατά τη μετατόπιση και επιστρέφει το αποτέλεσμα σε μη προσημασμένη μορφή, με τη χρήση των συναρτήσεων `$signed` και `$unsigned`.
- Η προεπιλεγμένη τιμή του αποτελέσματος είναι μηδέν.
- Η έξοδος `zero` ενεργοποιείται (γίνεται `1`) όταν το αποτέλεσμα της επιλεγμένης πράξης είναι 0.

2 Άσκηση 2

Στην Άσκηση 2 ζητείται η υλοποίηση ενός κυκλώματος αριθμομηχανής, το οποίο χρησιμοποιεί την ALU που κατασκευάστηκε στην Άσκηση 1. Το κύκλωμα περιλαμβάνει εισόδους για την αλληλεπίδραση του χρήστη μέσω κουμπιών και διακοπτών, ενώ αποθηκεύει και επεξεργάζεται τα δεδομένα σε έναν καταχωρητή.

- Στο module '`calc.v`':
  1. Ξεκινάμε δημιουργώντας το `module` της αριθμομηχανής, ορίζοντας τις εισόδους και τις εξόδους του.
  2. Δημιουργούμε έναν καταχωρητή `accumulator` για την αποθήκευση της τρέχουσας τιμής της αριθμομηχανής.
  3. Ενσωματώνουμε τη μονάδα ALU που κατασκευάστηκε στην προηγούμενη άσκηση και τον κωδικοποιητή του module '`calc_enc`'. Οι είσοδοι της ALU παράγονται με επέκταση προσήμου του καταχωρητή και των διακοπτών του module '`calc_enc`'.
  4. Ο καταχωρητής `accumulator` ανανεώνεται σε κάθε θετική ακμή του ρολογιού (ακολουθιακή λογική) σε ένα `always` block και με non-blocking εντολές, ανάλογα με τα σήματα ελέγχου από τα κουμπιά. Πιο συγκεκριμένα, μηδενίζεται όταν πατηθεί το κουμπί `btnu` και γράφει το αποτέλεσμα της ALU όταν πατηθεί το `btnd` (κατώτερα 16 bit).
  5. Η έξοδος του κυκλώματος (`led`) λαμβάνει το περιεχόμενο του καταχωρητή οποιαδήποτε στιγμή αλλάζει.
- Το module '`calc_enc`' κωδικοποιεί τις εντολές από τα κουμπιά 'btnl', 'btnc', 'btnr' σε ένα σήμα λειτουργίας ALU ('`alu_op`') σε Structural Verilog. Χρησιμοποιεί λογικές πύλες NOT, AND και OR για να παράγει τις διάφορες λειτουργίες της ALU, ακριβώς όπως ζητήθηκε στην εκφώνηση.
- Το `testbench` εξομοιώνει τη λειτουργία του κυκλώματος, παρέχοντας εισόδους και καταγράφοντας τις εξόδους, ώστε να διασφαλιστεί ότι οι πράξεις εκτελούνται σωστά.

1. Ορίζουμε το `testbench` module και συνδέουμε τις εισόδους και εξόδους του με το κύκλωμα της αριθμομηχανής (`calc`), που αποτελεί τη μονάδα υπό έλεγχο (*Device Under Test, DUT*).
2. Εξετάζουμε όλες τις πράξεις της αριθμομηχανής. Για κάθε δοκιμή:
   - Ρυθμίζουμε τα κουμπιά εισόδου ώστε να καθορίζεται η επιθυμητή πράξη (`alu_op`).
   - Ορίζουμε με την τιμή του `sw`.
   - Αρχικοποιούμε το κύκλωμα.
   - Εμφανίζεται το αποτέλεσμα της ALU και επαληθεύεται, συγκρίνοντάς το με την αναμενόμενη τιμή (όπως δόθηκε στον πίνακα της εκφώνησης) και γίνεται `$display` αν έγινε σωστά η πράξη ή όχι.

Κυματομορφές άσκησης 2

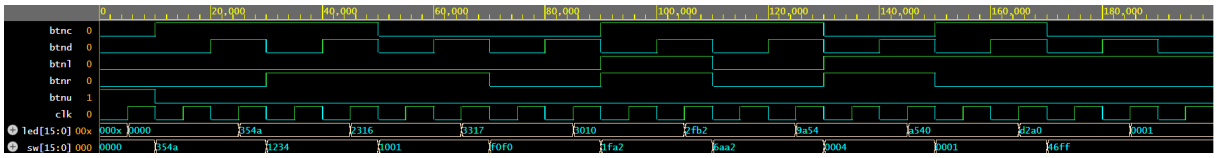


Figure 1: Κυματομορφή προσομοίωσης άσκησης 2

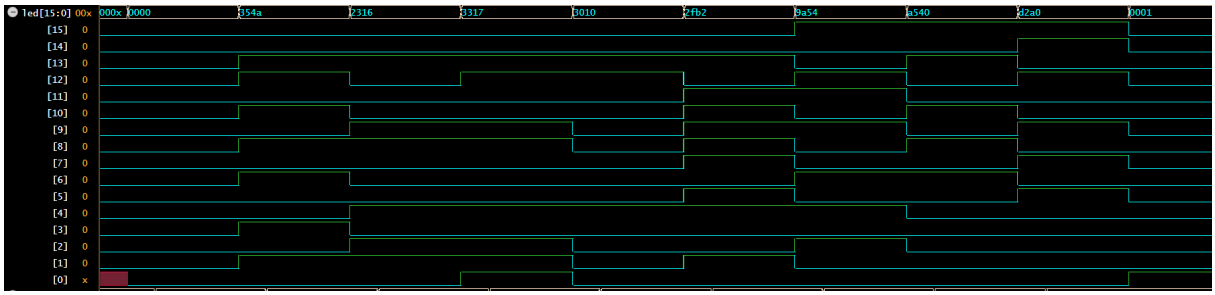


Figure 2: Κυματομορφή εξόδου led

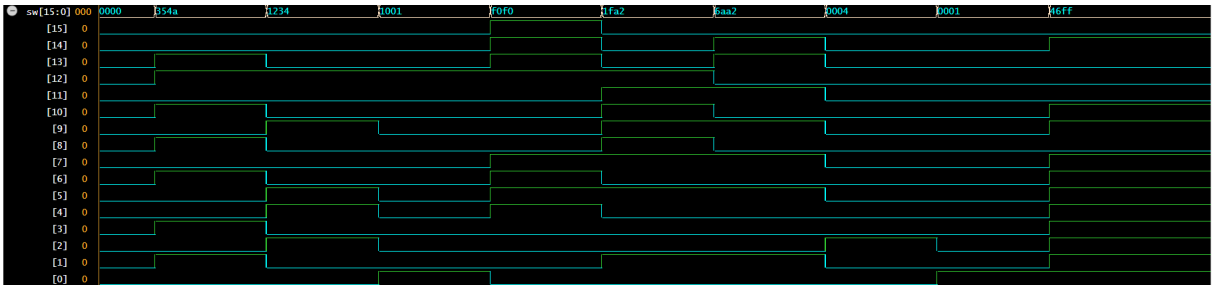


Figure 3: Κυματομορφή διακόπτη sw

3 Άσκηση 3

Στην άσκηση αυτή υλοποιούμε το αρχείο καταχωρητών του επεξεργαστή. Η βασική διαφορά του συγκεκριμένου module ως προς την υλοποίηση είναι η αρχικοποίηση της παραμέτρου *DATAWIDTH*

και το όρισμα του πλάτους των θυρών του module βάση αυτής. Μιας και το *DATAWIDTH* είναι 32 bit, και εμείς θέλουμε για τις θύρες `writeData`, `readData1` και `readData2` επίσης 32 bit, πρέπει να οριστούν ως *DATAWIDTH* - 1.
Κατά τη θετική ακμή του ρολογιού:

- Ελέγχεται αν το σήμα εγγραφής (`write`) είναι ενεργό. Εάν είναι ενεργό τα δεδομένα από την είσοδο `writeData` αποθηκεύονται στον καταχωρητή `writeReg`.
- Ανάγνωση από τους καταχωρητές:
  - Η ανάγνωση είναι σύγχρονη.
  - Οι τιμές από τις διευθύνσεις `readReg1` και `readReg2` εμφανίζονται απευθείας στις εξόδους `readData1` και `readData2`.
- Λογική για σύμπτωση διευθύνσεων εγγραφής και ανάγνωσης:
  - Εάν η διεύθυνση `writeReg` είναι ίδια με τη `readReg1` ή τη `readReg2` και το σήμα `write` είναι ενεργό οι έξοδοι `readData1` και `readData2` ενημερώνονται με την τιμή του `writeData`.
- Χρησιμοποιήθηκαν non-blocking εντολές, αφού η λογική είναι ακολουθιακή.

4 Άσκηση 4

Στην άσκηση υλοποιούμε το κύκλωμα του *datapath* για τον επεξεργαστή.

- Αρχικά, ορίζουμε το module με τις θύρες εισόδου και εξόδου του.
- Ορίζουμε ως τοπικές παραμέτρους τις τιμές των opcode, σύμφωνα με τα specifications του RISC-V.

```
localparam [6:0] IMMEDIATE  = 7'b0010011;
localparam [6:0] NON_IMMEDIATE = 7'b0110011;
localparam [6:0] LW         = 7'b0000011;
localparam [6:0] SW         = 7'b0100011;
localparam [6:0] BEQ        = 7'b1100011
```

- Δημιουργήθηκε η μεταβλητή `PC_inter`, η οποία χρησιμεύει για την αποθήκευση της τρέχουσας τιμής του καταχωρητή προγράμματος (PC). Κατά την αρχικοποίηση, η τιμή του PC τίθεται στην τιμή της παραμέτρου `INITIAL_PC`. Δημιουργήθηκαν και άλλα wires για να γίνει εύκολη η διασύνδεση των `regfile` module, της προηγούμενης άσκησης, και του `alu` module της πρώτης. Οι `readReg1 & 2` καθώς και η `writeReg` του `regfile` παίρνουν υποπεδία από την εντολή instruction (32 bit) για να καθορίσουν ποιοι καταχωρητές χρησιμοποιούνται. Αυτές οι τρεις τιμές αντιστοιχούν σε 5-bit διευθύνσεις ($2^5 = 32$) και χρησιμοποιούνται για να επιλέξουν ποιον από τους 32 καταχωρητές θα διαβάσουμε ή θα γράψουμε.
- Υλοποιείται επέκταση προσήμου (sign extension) για τις διαφορετικές μορφές των άμεσων (immediate) τιμών που συναντάμε σε εντολές RISC-V. Το `immediate_I` χρησιμοποιείται για τις εντολές τύπου `I`, το `immediate_S` για τις εντολές αποθήκευσης και το `immediate_B` για τις εντολές διακλάδωσης.
  - Στις εντολές τύπου **I**, η άμεση τιμή προέρχεται από τα 12 πιο σημαντικά bits της εντολής (`instr[31:20]`) και επεκτείνεται με πρόσημο σε 32 bits.
  - Στις εντολές τύπου **S**, η άμεση τιμή σχηματίζεται από τη συνένωση των bits `instr[31:25]` (πιο σημαντικά) και `instr[11:7]` (λιγότερο σημαντικά).

– Στις εντολές τύπου **B**, η άμεση τιμή δημιουργείται από τα bits `instr[31]`, `instr[7]`, `instr[30:25]`, και `instr[11:8]`, με την προσθήκη ενός επιπλέον 0 στο τέλος για ευθυγράμμιση.

Σε όλες τις περιπτώσεις, τα bits επεκτείνονται με πρόσημο για τη χρήση τους από την ALU.

- Έπειτα, μέσω ενός `always` block επιρρεπές σε οποιαδήποτε αλλαγή των `immediate` τιμών και μιας `if` καθορίζουμε ποιον τύπο άμεσης τιμής (`immediate`) θα χρησιμοποιήσει, με βάση το πεδίο `opcode` της εντολής (`instr[6:0]`). Αν η εντολή είναι τύπου `SW`, επιλέγεται η άμεση τιμή `immediate_S`. Αν είναι τύπου `BEQ`, επιλέγεται η `immediate_B`, με επιπλέον ολίσθηση (shift) αριστερά κατά 1 θέση, αφού οι άμεσες τιμές στις διακλαδώσεις είναι ευθυγραμμισμένες με λέξεις μνήμης. Τέλος, αν πρόκειται για εντολές τύπου `I-TYPE` ή `LW`, επιλέγεται η `immediate_I`, που αφορά την υπογεγραμμένη άμεση τιμή της εντολής.

- Ο `PC` (Program Counter) ενημερώνεται κατά την ακμή του ρολογιού. Αν το `rst` είναι ενεργό, ο `PC` αρχικοποιείται στην τιμή `INITIAL_PC`. Αν το `PCSrc` είναι ενεργό, τότε η τιμή του `PC` αλλάζει κατά την τιμή του immediate. Διαφορετικά, αυξάνεται κατά 4 (επόμενη εντολή).

- Η τιμή του `op2` υπολογίζεται δυναμικά, ανάλογα με το αν θα χρησιμοποιηθεί η τιμή ενός καταχωρητή ή μια άμεση τιμή. Αυτό καθορίζεται από το σήμα `ALUSrc`. Αν το `ALUSrc` είναι 1, τότε το `op2` παίρνει την τιμή `immediate`, ενώ αν είναι 0, τότε παίρνει την τιμή από το `readData2`.

- Τέλος, τα δεδομένα εξόδου για την εγγραφή στη μνήμη (`dWriteData`) και την εγγραφή πίσω στον καταχωρητή (`WriteBackData`) καθορίζονται ως εξής. Αν το `MemToReg` είναι 1, τότε η τιμή που θα γραφεί είναι το δεδομένο από τη μνήμη (`dReadData`). Αν είναι 0, τότε η τιμή που θα γραφεί προέρχεται από την έξοδο της ALU.

5 Άσκηση 5

Στην άσκηση αυτή υλοποιούμε το module `procedures`, το οποίο διαχειρίζεται τη ροή δεδομένων του επεξεργαστή RISC-V και το αντίστοιχο testbench.

1. Η κύρια λειτουργία του module είναι να ελέγχει τη ροή των εντολών και να διαχειρίζεται τη μνήμη, καθώς και την εκτέλεση διαφορετικών τύπων εντολών, όπως αριθμητικές, εντολές φόρτωσης/εγγραφής και συγκρίσεις.

   - Ορίζουμε τις θύρες εισόδου και εξόδου καθώς και την παράμετρο `INITIAL_PC`, η οποία χρησιμοποιείται για την αρχικοποίηση του μετρητή προγράμματος (PC) στην τιμή `0x00400000`.
   - Έπειτα συνδέουμε το module με το datapath που δημιουργήσαμε στην προηγούμενη άσκηση.
     Η τιμή του `INITIAL_PC` διασυνδέεται επίσης πιο ειδικά: `#(.INITIAL_PC(INITIAL_PC))`.
   - Στη συνέχεια, ο κώδικας ορίζει τις καταστάσεις του pipeline για τον έλεγχο της ροής των δεδομένων και των εντολών. Η διαδικασία αυτή χρησιμοποιεί ένα finite state machine (FSM), για την διαχείριση των φάσεων του pipeline: `IF`, `ID`, `EX`, `MEM`, και `WB`.

     ```
     parameter [2:0] IF = 3'b000;
     parameter [2:0] ID = 3'b001;
     parameter [2:0] EX = 3'b010;
     parameter [2:0] MEM = 3'b011;
     parameter [2:0] WB = 3'b100;
     ```

   - Συνεχίζοντας, ορίζουμε τις παραμέτρους για τους διάφορους τύπους εντολών, χρησιμοποιώντας τα `opcode` και `funct3` πεδία για να διακρίνουμε τις εντολές, αντιστοιχίζοντας

τα πρώτα 6 bit και τα bit 12-14 αντίστοιχα. Χώρισα τις παραμέτρους στις βασικές της μνήμης: LW, SW, την διακλάδωση BEQ, ως `IMMEDIATE` τις ALU Immediate και ως `NON_IMMEDIATE` τις Register-Register. Αυτό μας επιτρέπει να αποκωδικοποιήσουμε το σήμα `ALUCtrl`.

- Με ένα always block, που εκτελείται σε κάθε ανερχόμενο μέτωπο του ρολογιού (`clk`) ή σε κάθε ανερχόμενο μέτωπο του σήματος επαναφοράς (`rst`), όταν το `rst` είναι ενεργό, όλες οι παράμετροι και σήματα αρχικοποιούνται στις αρχικές τους τιμές και η κατάσταση του FSM τίθεται στη φάση `IF`. Όταν το `rst` δεν είναι ενεργό, η κατάσταση του FSM αλλάζει διαδοχικά στις επόμενες φάσεις: `IF -> ID -> EX -> MEM -> WB -> IF`. Συνεπώς, η υλοποίηση έγινε με non-blocking εντολές ακολουθώντας την ακολουθιακή λογική.
- Σε επόμενο always block, το οποίο εκτελείται σε κάθε ανερχόμενο μέτωπο του ρολογιού (`clk`), καθορίζονται οι καταστάσεις των φάσεων του pipeline. Ανάλογα με την τρέχουσα κατάσταση του FSM (`IF`, `ID`, `EX`, `MEM`, `WB`), καθορίζονται οι φάσεις του pipeline. Στη φάση `MEM`, ενεργοποιείται το `MemRead` για την εντολή `LW` και το `MemWrite` για την εντολή `SW`. Στη φάση `WB`, ενεργοποιείται το `loadPC`, το `MemToReg` για την εντολή `LW`, και το `RegWrite` για τις εντολές που δεν είναι `SW` ή `BEQ`.
- Με ένα ακόμη always block, που εκτελείται συνεχώς, καθορίζεται ο έλεγχος της ALU. Αν το `opcode` είναι `NON_IMMEDIATE`, τότε ανάλογα με την τιμή του `funct3` και το bit 30 της εντολής (`instr[30]`), καθορίζεται η τιμή του `ALUCtrl`. Αν το `opcode` είναι `IMMEDIATE`, τότε καθορίζεται η τιμή του `ALUCtrl` ανάλογα με το `funct3`. Για τις εντολές `LW` και `SW`, το `ALUCtrl` τίθεται σε `ALUOP_ADD`, ενώ για την εντολή `BEQ`, το `ALUCtrl` τίθεται σε `ALUOP_SUB`. Δεν ορίστηκε ξεχωριστή παράμετρος function7, καθώς χρησιμεύει στην διαφοροποίηση μόνο μεταξύ των `ARITHMETIC` και `SRL` εντολών, οπότε έγινε με ένα απλό if statement στο μπλοκ.
- Τέλος, με ένα always block ενημερώνονται τα σήματα `ALUSrc` και `PCSrc`. Το `ALUSrc` τίθεται σε 1 αν το `opcode` είναι `LW`, `SW` ή `IMMEDIATE`, διαφορετικά τίθεται σε 0. Το `PCSrc` καθορίζεται από την τιμή του `Zero` αν η εντολή είναι `BEQ`, διαφορετικά τίθεται σε 0.

2. Το `testbench` εξομοιώνει τη λειτουργία του RISC-V επεξεργαστή, παρέχοντας σήματα ρολογιού και επαναφοράς, καθώς και διασυνδέοντας τις μνήμες εντολών και δεδομένων, για να ελεγχθεί η ορθή λειτουργία του πυρήνα.

   (a) Ορίζεται το `testbench` module, και οι είσοδοι/έξοδοι του RISC-V πυρήνα (`procedures`) συνδέονται με τις κατάλληλες μονάδες μνήμης ((`INSTRUCTION_MEMORY`),(`DATA_MEMORY`)).

   (b) Το `PC` από το module (`procedures`) συνδέεται με τη θύρα διεύθυνσης (`addr`) της μνήμης εντολών (`rom`), ενώ τα πρώτα 9 bits του χρησιμοποιούνται για να προσαρμοστούν στο μέγεθος της διεύθυνσης της μνήμης. Παρόμοια, η θύρα `dAddress` συνδέεται με τη μνήμη δεδομένων (`ram`), επίσης με τα πρώτα 9 bits.

   (c) Το σήμα ρολογιού (`clk`) παράγεται με τη χρήση ενός `always` μπλοκ, το οποίο αλλάζει περιοδικά την τιμή του κάθε 5 ns.

   (d) Η επαναφορά (`rst`) ενεργοποιείται αρχικά για τα πρώτα 15 ns, ώστε να διασφαλιστεί η αρχικοποίηση του πυρήνα, και στη συνέχεια απενεργοποιείται.

   (e) Με τη χρήση των `$dumpfile` και `$dumpvars`, καταγράφονται όλες οι αλλαγές στα σήματα σε ένα αρχείο εξόδου μορφής `VCD`.

Κυματομορφές άσκησης 5

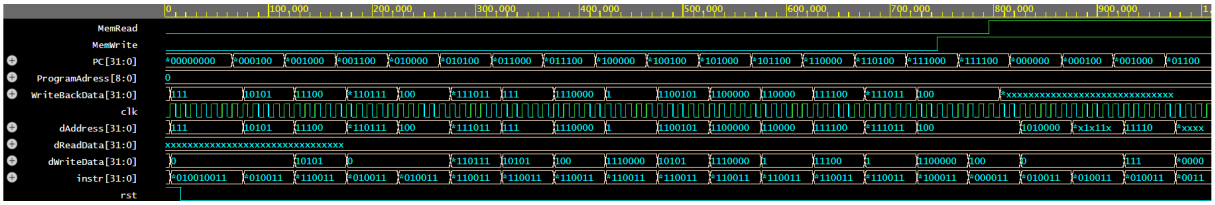


Figure 4: Κυματομορφή προσομοίωσης άσκησης 5 - σήματα του testbench

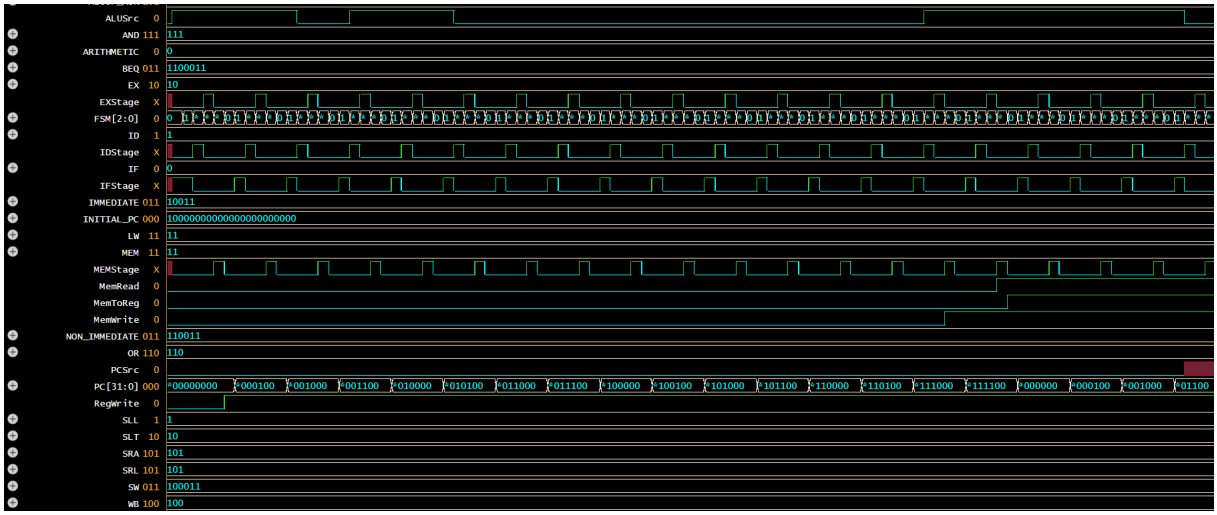


Figure 5: Κυματομορφή προσομοίωσης άσκησης 5 - σήματα του core

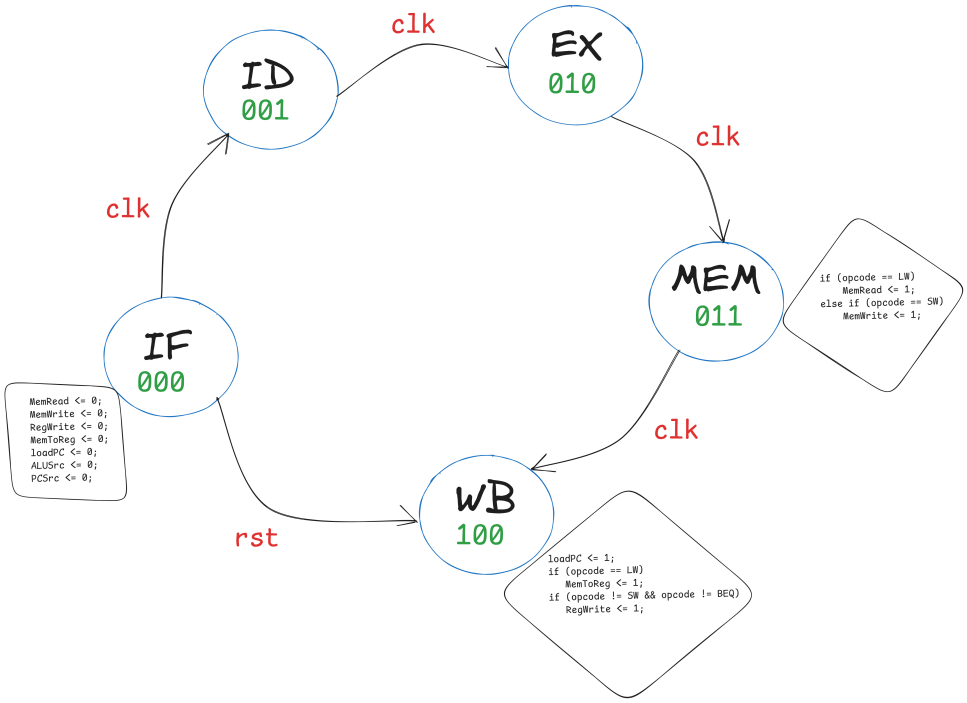


Figure 6: Το διάγραμμα του FSM

Οι εντολές που τρέχει ο επεξεργαστής από την ROM

```
00000000011100000000000010010011  addi x1, x0, 7
00000001010100000000000100010011  addi x2, x0, 21
00000000001000001000000110110011  add x3, x1, x2
11111111011100000000001000010011  addi x4, x0, -9
11111110111100010000001010010011  addi x5, x2, -17
00000000010000101000001100110011  add x6, x5, x4
01000000001000011000001110110011  sub x7, x3, x2
00000000010100111001010000110011  sll x8, x7, x5
00000000100000100010010010110011  slt x9, x4, x8
00000000001001000100010100110011  xor x10, x8, x2
00000000100001010111010110110011  and x11, x10, x8
00000000100101011101011000110011  srl x12, x11, x9
00000000001101100110011010110011  or x13, x12, x3
01000000100100100101011100110011  sra x14, x4, x9
00000000101100101010000000100011  sw x11, 0(x5)
00000000000000010101001111000011  lw x15, 0(x5)
11111101001101000111100000010011  andi x16, x8, -45
00000001011010000110100010010011  ori x17, x16, 22
00000000000101101101100100010011  srli x18, x13, 1
00000000101101111000100001100011  beq x15, x11, 16
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000111110010010010010010011  slti x9, x18, 15
00000011101001000100100110010011  xori x19, x8, 58
00000000000110001001101000010011  slli x20, x17, 1
01000000010011111010010010010011  srai x5, x15, 2
```