Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής Ι

Παπουτσή Νικολέτα
ΑΕΜ : 10858

Περιεχόμενα

Η παρούσα εργασία έχει ως στόχο τη σχεδίαση των μονάδων ενός επεξεργαστή RISC-V. Για την υλοποίηση και επαλήθευση όλων των παρακάτω χρησιμοποιήθηκε η πλατφόρμα **EDA playground** και το **EPWave** για τις κυματομορφές προσομοίωσης.

Άσκηση 1 : Σχεδιασμός και υλοποίηση ALU

Στην Άσκηση 1, υλοποιήθηκε μια αριθμητική/ λογική μονάδα (ALU) για τον επεξεργαστή RISC-V.

Το αρχείο υλοποίησης είναι το **alu.v**.

- Οι λειτουργίες της ALU καθορίζονται από έναν 4-bit κωδικό (alu_op) , ο οποίος αντιστοιχεί σε συγκεκριμένες αριθμητικές και λογικές πράξεις όπως λογική AND, OR και XOR, προσημασμένη πρόσθεση και αφαίρεση, «μικρότερο από» σύγκριση (SLT) και τρεις διαφορετικές ολισθήσεις (shift).

- Η ALU λαμβάνει δύο εισόδους 32-bit (op1,op2). Ένα always μπλοκ με χρήση της εντολής case καθορίζει ποια πράξη θα εκτελεστεί με βάση το alu_op.

Άσκηση 2 : Σχεδίαση κυκλώματος αριθμομηχανής

Η άσκηση έχει στόχο τον σχεδιασμό και την υλοποίηση ενός κυκλώματος αριθμομηχανής που περιλαμβάνει τρία κύρια μέρη :

1. **ALU**
    - Η ALU που εκτελεί τις αριθμητικές και λογικές πράξεις που καθορίζονται από το σήμα alu_op.
2. **Module calc_enc**
    - Το module calc_enc υλοποιείται σε Structural Verilog και καθορίζει το alu_op με βάση τις τιμές των πλήκτρων εισόδου (btnl,btnc,btnr).
3. **Accumulator**
    - Ο συσσωρευτής 16-bit καταχωρητής (accumulator) αποθηκεύει το αποτέλεσμα της ALU και ενημερώνεται μόνο όταν πατηθεί το btnd. Μηδενίζεται όταν πατηθεί το btnu.

Η αριθμομηχανή εμφανίζει το αποτέλεσμα της πράξης στα LEDs.

Υλοποίηση calc_enc.v

Υλοποιείται με **Structural Verilog** σύμφωνα με τα σχήματα 2-5 και κάθε bit του alu_op παράγεται ξεχωριστά.

Υλοποίηση calc.v

Στο αρχείο **calc.v** υλοποιείται ο συσσωρευτής και συνδέονται η ALU και το module calc_enc.

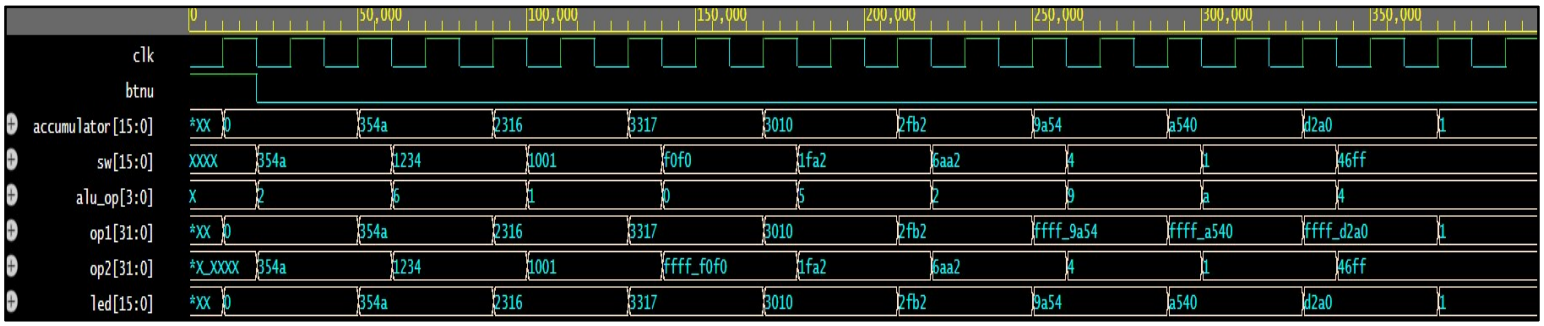
- Η σύνδεση των modules **alu** και **calc_enc** με το κύριο module **calc** γίνεται μέσω **ρητής αντιστοίχισης** των θυρών εισόδου και εξόδου. Η χρήση της εντολής include χρησιμοποιείται για την ενσωμάτωση των αρχείων alu.v και calc_enc.v.
- Καθώς τα σήματα **sw** (δεδομένα εισόδου) και η έξοδος του accumulator είναι 16-bit απαιτείται επέκταση προσήμου ώστε να τα δεχτεί η ALU ως δεδομένα εισόδου 32-bit. Ο τελεστής **concatenation** της Verilog χρησιμοποιείται για την επέκταση προσήμου.
- Ο accumulator υλοποιείται με ένα **always @(posedge clk) block**, όπου ελέγχονται δύο περιπτώσεις :
    - Με το πάτημα του πλήκτρου **btnu**, ο συσσωρευτής μηδενίζεται.
    - Με το πάτημα του πλήκτρου **btnd**, ο συσσωρευτής ενημερώνεται με τα 16 λιγότερο σημαντικά bits του αποτελέσματος της ALU.
- Το σήμα εξόδου led συνδέεται με τον συσσωρευτή.

- Στην υλοποίηση του accumulator χρησιμοποιούνται **non-blocking** εντολές (<=) για την εκχώρηση τιμών στο always block , καθώς βασίζεται στο ρολόι (clk) για τον συγχρονισμό του .

Testbench: calc_tb.v

- Με την εντολή include ενσωματώνεται το module calc.v που πρόκειται να γίνει ο έλεγχος.
- Με την εντολή timescale ορίζεται η κλίμακα του χρόνου(1ns/1ns).
- Ένα initial block για τη δημιουργία του ρολογιού (clk).
- Ένα κύριο initial block που περιέχει τα inputs {btnl,btnc,btnr} που χρησιμοποιούνται για τον έλεγχο της ορθής λειτουργίας της αριθμομηχανής.
- Για την εμφάνιση των κυματομορφών χρησιμοποιούνται οι εντολές $dumpfile("calc_tb.vcd"), $dumpvars(0,calc_tb), $finish.

Κυματομορφές προσομοίωσης



Άσκηση 3 : Αρχείο καταχωρητών (Register File)

Το αρχείο καταχωρητών αποθηκεύει τις τιμές των καταχωρητών που χρησιμοποιούνται από τον επεξεργαστή RISC-V.

Υλοποίηση regfile.v

Σχεδιάζετε ένα αρχείο καταχωρητών που περιλαμβάνει :

- 32 καταχωρητές δεδομένων, καθένας με πλάτος που εξαρτάται από τη παράμετρο DATAWIDTH.

Σε κάθε κύκλο ρολογιού γίνεται εγγραφή δεδομένων στον καταχωρητή που καθορίζεται από τη διεύθυνση writeReg και δύο ταυτόχρονες αναγνώσεις δεδομένων από τους καταχωρητές που καθορίζονται από τις διεύθυνσεις readReg1 και readReg2.

Για την υλοποίηση των λειτουργιών ανάγνωσης και εγγραφής γίνεται χρήση non-blocking εντολών (<=), ώστε όλες οι τιμές στο always block να ενημερώνονται παράλληλα στο τέλος του κύκλου ρολογιού .

Αν η διεύθυνση εγγραφής (writeReg) είναι ίδια με κάποια από τις διευθύνσεις ανάγνωσης (readReg1 ή readReg2), η έξοδος (readData1 ή readData2) ενημερώνεται άμεσα με τη τιμή του writeData.

Άσκηση 4 :Σχεδιασμός μονάδας διαδρομής δεδομένων (Datapath)

Η άσκηση έχει στόχο τον σχεδιασμό της διαδρομής δεδομένων (datapath) ενός επεξεργαστή RISC-V , που υποστηρίζει εντολές I-type, S-type και B-type.

Υλοποίηση datapath.v

Με την εντολή include ενσωματώνονται τα modules από τις προηγούμενες ασκήσεις (alu.v και regfile.v). Από το εγχειρίδιο των εντολών RISC-V χρησιμοποιήθηκαν τα παρακάτω opcode :

- Immediate : 0010011
- LW : 0000011
- SW : 0100011
- BEQ : 1100011

Immediate Generation

Υπολογίζονται οι τιμές immediate με βάση το opcode της κάθε εντολής αφού έχει πραγματοποιηθεί sign extension. Για τις B-type εντολές πραγματοποιήθηκε μετατοπιση του immediate κατά 1 bit προς τα αριστερα.

Program Computer

Με ένα always block γίνεται η ενημέρωση του PC. Αν το σήμα rst είναι ενεργό, το PC παίρνει την αρχική του τιμή (INITIAL_PC). Διαφορετικά, αν το σήμα loadPC είναι ενεργό, ελέγχεται το PCSrc :

- PCSrc = 1 : PC + branch_offset(immediate)
- PCSrc = 0: PC + 4

ALU

Με ένα always block καθορίζεται το op2 της ALU. Αν το ALUSrc είναι ενεργό (1), επιλέγεται το immediate. Αλλιώς, επιλέγεται το readData2.

Write Back

Αυτό το always block καθορίζει ποια δεδομένα θα εγγραφούν στο Register File. Αν το MemToReg είναι ενεργό (1) εγγράφονται τα δεδομένα από τη μνήμη dReadData. Αλλιώς, εγγράφεται το αποτέλεσμα της ALU.

Άσκηση 5 : FSM και Ελεγκτής

Η άσκηση αυτή υλοποιεί έναν επεξεργαστή πολλαπλών κυκλών RISC-V και περιλαμβάνει μια μηχανή FSM (Finite State Machine) για τον συντονισμό των σταδίων εκτέλεσης των εντολών.

Υλοποίηση top_proc.v

Μονάδες υλοποίησης:

- FSM : Αποκωδικοποίηση 5 καταστάσεων¨
    - IF(Instruction Fetch) : Ανάκτηση Εντολών από τη ROM
    - ID(Instruction Decode) : Αποκωδικοποίηση εντολών
    - EX(Execute) : Εκτέλεση πράξεων που καθορίζεται από την ALU
    - MEM(Memory) : Πρόσβαση στη μνήμη
    - WB(Write Back) : Εγγραφή αποτελεσμάτων
- ALU
- Datapath
- ROM (Μονάδα Μνήμης Εντολών)
- RAM (Μονάδα Μνήμης Δεδομένων)

Ένα always@(posedge clk) block ελέγχει αν έχει ενεργοποιηθεί το σήμα reset. Εάν ναι, η FSM επιστρέφει στην κατάσταση IF. Διαφορετικά, μεταβαίνει στην επόμενη κατάσταση (next_state).

Ένα always block ευαίσθητο σε κάθε σήμα καθορίζει με το σήμα ALUSrc εάν η ALU θα χρησιμοποιήσει immediate δεδομένα ή δεδομένα από τους καταχωρητές.

Με ένα always block και if καθορίζεται η επιλογή της πράξης ALUCtrl που βασίζεται στον τύπο της εντολής (instr[6:0]) και των funct3,funct7.

Ένα always block διαχειρίζεται τις μεταβάσεις της FSM. Σε κάθε στάδιο ενεργοποιούνται τα αντίστοιχα σήματα ελέγχου. Το next_state καθορίζεται από το opcode της εντολής και τη τρέχουσα κατάσταση.

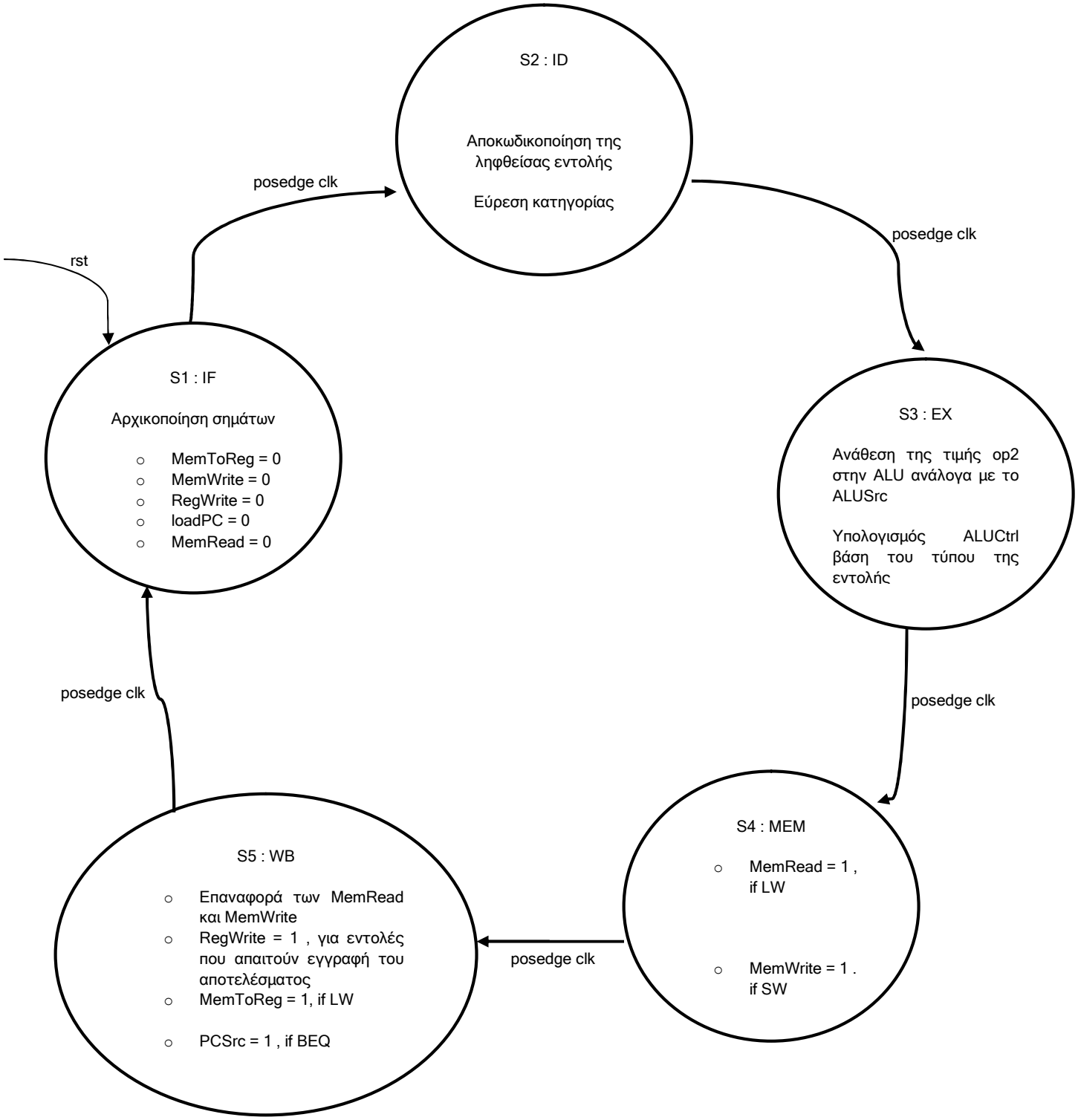
Υλοποίηση top_proc_tb.v

Το αρχείο αποτελεί το testbench για την επαλήθευση ορθής λειτουργίας ολόκληρου του επεξεργαστή. Γίνεται διασύνδεση των μνημών ROM και RAM με τον επεξεργαστή. Το αρχείο rom_bytes.data περιέχει 32-bit εντολές που καλείται να εκτελέσει ο επεξεργαστής. Για να επαληθευτεί η σωστή ή όχι λειτουργία του επεξεργαστή, τα δεδομένα μετατράπηκαν σε κώδικα Assemply

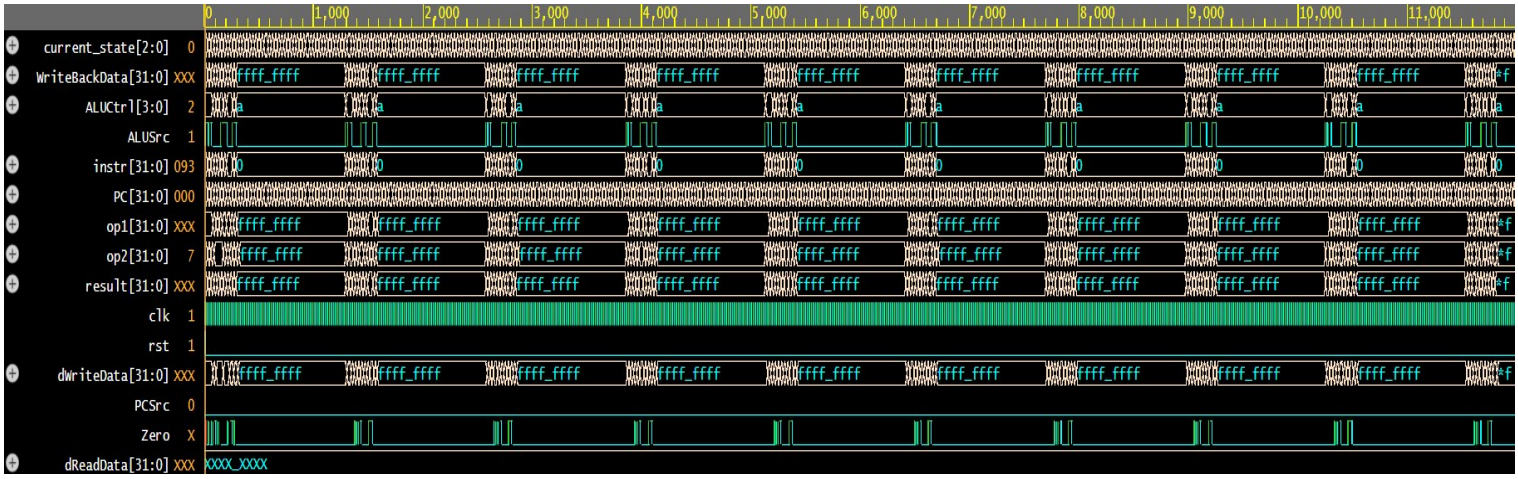
Το FSM που υλοποιείται είναι μια Moore μηχανή γιατί:

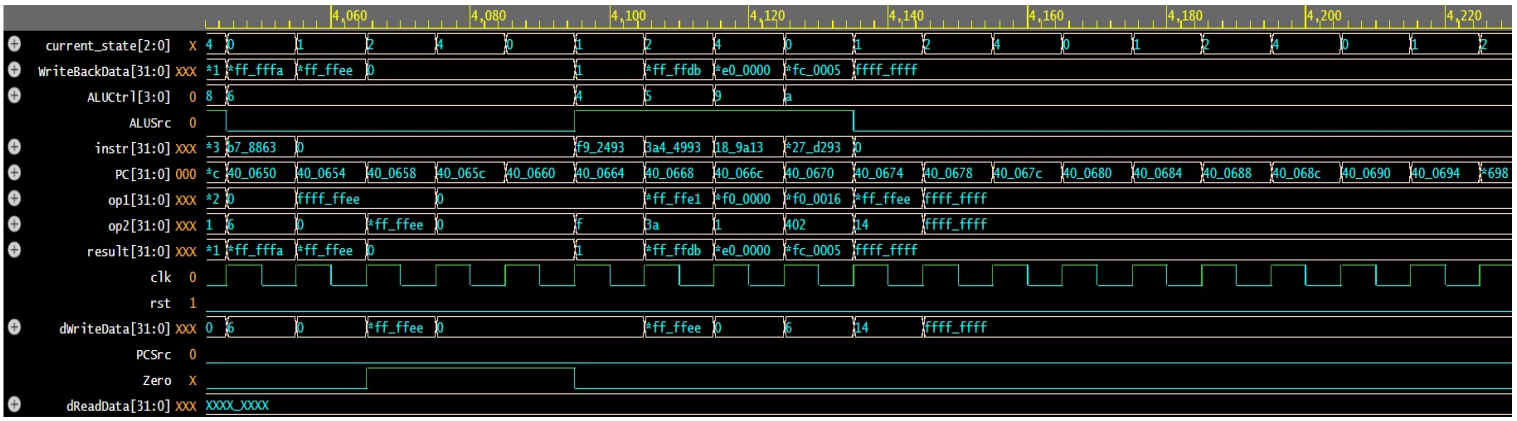
- Τα σήματα εξόδου καθορίζονται αποκλειστικά από τη τρέχουσα κατάσταση (current_state) και όχι από τις εισόδους απευθείας μέσα στο FSM. Οι έξοδοι βρίσκονται στο always block που είναι ευαίσθητο στη τρέχουσα κατάσταση (current_state).

ΣΧΗΜΑΤΙΚΟ ΔΙΑΓΡΑΜΜΑ FSM



Κυματομορφές προσομοίωσης





Δεδομένα σε κώδικα Assemply

Η αριστερή στήλη αναφέρεται στη διεύθυνση της ROM όπου είναι αποθηκευμένη η εντολή.

1: addi x1, x0, 7	Expected result: x1 = 7
5: addi x2, x0, 21	Expected result: x2 = 21
9: add x3, x1, x2	Expected result: x3 = 28
13: addi x4, x0, -9	Expected result: x4 = -9
17: addi x5, x2, -17	Expected result: x5 = 4
21: add x6, x5, x4	Expected result: x6 = -5
25: sub x7, x3, x2	Expected result: x7 = 7
29: sll x8, x7, x5	Expected result: x8 = 112
33: slt x9, x4, x8	Expected result: x9 = 1
37: xor x10, x8, x2	Expected result: x10 = 101
41: and x11, x10, x8	Expected result: x11 = 96
45: srl x12, x11, x9	Expected result: x12 = 48
49: or x13, x12, x3	Expected result: x13 = 60
53: sra x14, x4, x9	Expected result: x14 = -5
57: sw x11, 0(x5)	Expected result: RAM[4] = 96
61: lw x15, 0(x5)	Expected result: x15 = 96
65: andi x16, x8, -45	Expected result: x16 = 80
69: ori x17, x16, 22	Expected result: x17 = 86
73: srli x18, x13, 1	Expected result: x18 = 30
77: beq x15, x11, 16	Expected result: PC = PC + 16

97: slti x9, x18, 15 Expected result: x9 = 0

101: xori x19, x8, 58 Expected result: x19 = 74

105: slli x20, x17, 1 Expected result: x20 = 172

109: srai x5, x15, 2 Expected result: x5 = 24