ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Ψηφιακά Συστήματα **HW** σε Χαμηλά Επίπεδα Λογικής **I**

Εργασία 2024-25

Αθανάσιος Σούρλας
ΑΕΜ 10709
sourlasa@ece.auth.gr

Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

17 Δεκεμβρίου 2024

Περιεχόμενα

1 Εισαγωγή

Το παρόν report αφορά την εργασία του μαθήματος **Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής I**, η οποία επικεντρώνεται στον σχεδιασμό ενός **32-bit επεξεργαστή RISC-V**, χρησιμοποιώντας **verilog**. Το project χωρίζεται σε πέντε επιμέρους εργασίες, οι οποίες περιγράφονται συνοπτικά παρακάτω:

1. **Άσκηση 1: Υλοποίηση ALU (Arithmetic Logic Unit)** Η Άσκηση 1 αφορά τον σχεδιασμό και την υλοποίηση μιας αριθμητικής/λογικής μονάδας (ALU), η οποία είναι υπεύθυνη για την εκτέλεση αριθμητικών και λογικών πράξεων.
2. **Άσκηση 2: Κατασκευή αριθμομηχανής** Η Άσκηση 2 αφορά την κατασκευή ενός κυκλώματος αριθμομηχανής που χρησιμοποιεί την ALU από την προηγούμενη άσκηση.
3. **Άσκηση 3: Υλοποίηση αρχείου καταχωρητών** Η Άσκηση 3 αφορά τη δημιουργία ενός αρχείου καταχωρητών (Register File), το οποίο θα ενσωματωθεί στον επεξεργαστή.
4. **Άσκηση 4: Μονάδα διαδρομής δεδομένων (Datapath)** Η Άσκηση 4 αφορά το σχεδιασμό της διαδρομής δεδομένων (datapath) του επεξεργαστή, η οποία διασφαλίζει τη σωστή ροή των δεδομένων και εντολών μεταξύ των επιμέρους μονάδων του συστήματος.
5. **Άσκηση 5: Μονάδα ελέγχου** Η Άσκηση 5 αφορά το σχεδιασμό ενός ελεγκτή πολλαπλών κύκλων, που εκτελεί κάθε εντολή σε πέντε κύκλους ρολογιού. Ο ελεγκτής απαιτείται για τη σωστή αλληλεπίδραση μεταξύ των μονάδων του επεξεργαστή.

Η εργασία αναπτύχθηκε στο Microsoft Visual Studio Code, χρησιμοποιώντας τον προσομοιωτή Icarus Verilog για την περιγραφή και προσομοίωση του κώδικα, ενώ για την απεικόνιση των κυματομορφών χρησιμοποιήθηκε το εργαλείο GTKWave.

2 Εργασία 1

Η υλοποίηση της ALU ήταν σχετικά απλή, καθώς περιλάμβανε τη δημιουργία των παραμέτρων για τις διάφορες λειτουργίες της ALU και την εφαρμογή ενός πολυπλέκτη για την επιλογή της κατάλληλης λειτουργίας, βάσει του ελέγχου εισόδου. Το μόνο σημείο προσοχής είναι οι πράξεις SLT και ASL, όπως αναφέρεται και στην εκφώνηση της εργασίας. Ο κώδικας της Άσκησης 1, περιλαμβάνεται στο αρχείο alu.v.

3 Εργασία 2

Στην άσκηση αυτή, δημιουργήσαμε τον accumulator, όπως φαίνεται στο Σχήμα 1, ο οποίος αποθηκεύει την τρέχουσα τιμή της αριθμομηχανής. Επίσης, δημιουργήσαμε το σήμα `alu_op` χρησιμοποιώντας structural Verilog, το οποίο καθορίζει ποια λειτουργία της ALU θα εκτελείται. Αυτές οι υλοποιήσεις βρίσκονται στα αρχεία calc.v και `calc_enc.v`, αντίστοιχα.

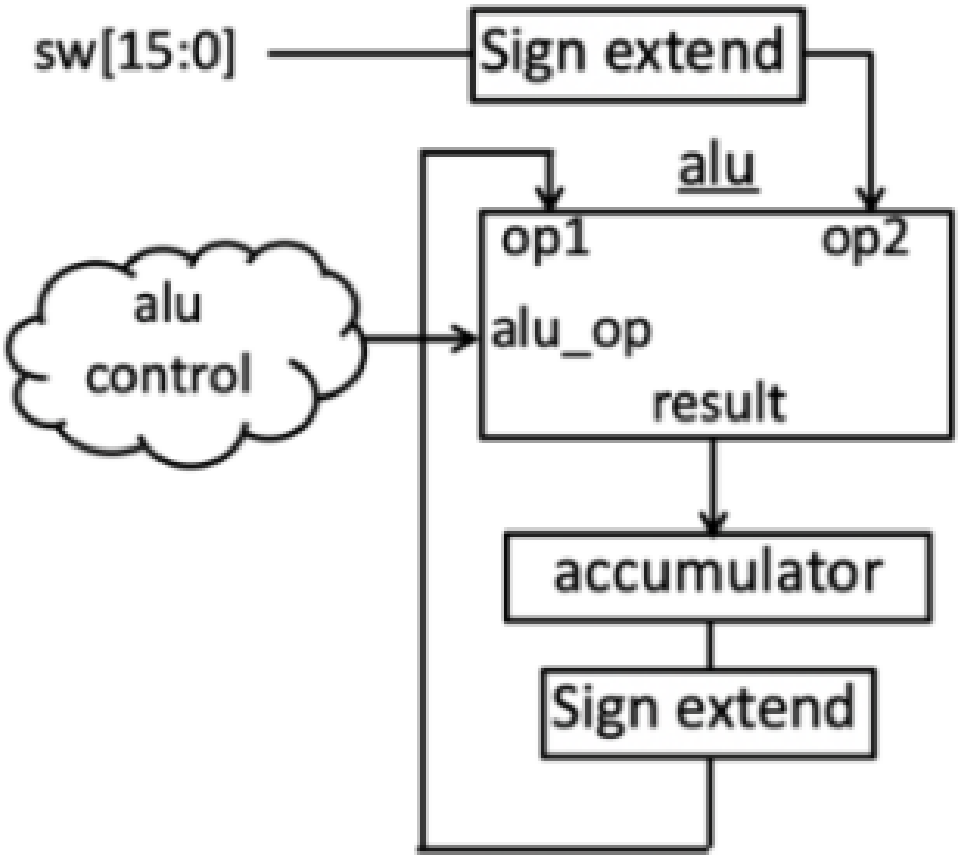

Figure 1: Διάγραμμα ροής της αριθμομηχανής.

Στην άσκηση, δημιουργήσαμε το testbench (`calc_tb.v`) για να ελέγξουμε τη σωστή λειτουργία της αριθμομηχανής και της ALU. Στο testbench, δοκιμάζουμε διάφορες τιμές για τα πλήκτρα btnl, btnc και btnr, παρακολουθώντας αν τα αποτελέσματα συμφωνούν με τα αναμενόμενα αποτελέσματα που αναφέρονται στον πίνακα της εκφώνησης.

3.1 Αποτελέσματα testbench.

Τα εξής αποτελέσματα εκτυπώθηκαν στο CMD και επιβεβαιώθηκε ότι τα αποτελέσματα συμφωνούν με τα αναμενόμενα:

- AND: Expected: 0x3010, Got: 3010
- XOR: Expected: 0x2fb2, Got: 2fb2
- ADD: Expected: 0x9a54, Got: 9a54
- Logical Shift Left: Expected: 0xa540, Got: a540
- Shift Right Arithmetic: Expected: 0xd2a0, Got: d2a0
- Less Than: Expected: 0x0001, Got: 0001
- XOR: Expected: 0x2fb2, Got: 2fb2
- ADD: Expected: 0x9a54, Got: 9a54
- Logical Shift Left: Expected: 0xa540, Got: a540
- Shift Right Arithmetic: Expected: 0xd2a0, Got: d2a0
- Less Than: Expected: 0x0001, Got: 0001

Παρατηρήθηκε στις κυματομορφές, ότι το αποτέλεσμα της ALU αλλάζει λίγο μετά την εκτέλεση κάθε εντολής. Αυτή η συμπεριφορά είναι φυσιολογική, καθώς συνδέεται με τον συγχρονισμό του σήματος btnu στο testbench. Συγκεκριμένα, το op1 ενημερώνεται με την τιμή του accumulator, ενώ το op2, που επηρεάζεται από την είσοδο των διακοπτών, δεν έχει ανανεωθεί ακόμη για την επόμενη εντολή. Ωστόσο, αυτή η αλλαγή δεν επηρεάζει την ορθή λειτουργία του συστήματος, καθώς οι κυματομορφές που καταγράφηκαν επιβεβαιώνουν τη σωστή εκτέλεση των πράξεων από την αριθμητική/λογική μονάδα (ALU), την αριθμομηχανή και τον encoder. Συνολικά, η παρατηρούμενη συμπεριφορά δεν υποδηλώνει κάποιο σφάλμα, αλλά αποτελεί μέρος της αναμενόμενης λειτουργίας του κυκλώματος.

Στα Figures 2-11 φαίνονται αναλυτικά όλες τις ζητούμενες κυματομορφές. Συγκεκριμένα:

- **Figure 2**: Όλες οι κυματομορφές
- **Figure 3**: 1η πράξη
- **Figure 4**: 2η πράξη
- **Figure 5**: 3η πράξη
- **Figure 6**: 4η πράξη
- **Figure 7**: 5η πράξη
- **Figure 8**: 6η πράξη
- **Figure 9**: 7η πράξη
- **Figure 10**: 8η πράξη
- **Figure 11**: 9η πράξη

4 Εργασία 3

Στην Άσκηση 3 υλοποιήθηκε το αρχείο καταχωρητών του επεξεργαστή (regfile.v), το οποίο αποθηκεύει και διαχειρίζεται τις τιμές των καταχωρητών. Το μόνο σημείο προσοχής ήταν η διαχείριση της περίπτωσης όπου η διεύθυνση εγγραφής (writeReg) είναι ίδια με κάποια από τις διευθύνσεις ανάγνωσης (readReg1 ή readReg2). Σε αυτή την περίπτωση δόθηκε προτεραιότητα στην εγγραφή.

5 Εργασία 4

Στην άσκηση αυτή, υλοποιήθηκε η διαδρομή δεδομένων (datapath.v) του επεξεργαστή, η οποία αποτελεί το κεντρικό στοιχείο για την επεξεργασία εντολών. Το datapath σχεδιάστηκε ώστε να υποστηρίζει ένα σύνολο εντολών RISC-V. Συγκεκριμένα, υποστηρίζονται οι εξής 20 εντολές: ADD, SUB, AND, OR, XOR, SLT, SLL, SRL, SRA, ADDI, ANDI, ORI, XORI, SLTI, SLLI, SRLI, SRAI, LW, SW, BEQ.

Η υλοποίηση περιλαμβάνει τον Program Counter (PC) με επιλογή μεταξύ PC + 4 ή διεύθυνσης διακλάδωσης, το αρχείο καταχωρητών από την Άσκηση 3 με αποκωδικοποίηση των readReg1, readReg2 και writeReg, τη δημιουργία άμεσων δεδομένων (Immediate) για τύπους I, S και B με σωστή επέκταση προσήμου, την ALU από την Άσκηση 1 με πολυπλέκτη εισόδου (ALUSrc), τη λογική υπολογισμού διεύθυνσης στόχου διακλάδωσης και τον πολυπλέκτη Write Back για επιλογή αποτελέσματος ALU ή dReadData (MemtoReg), με σύνδεση στο WriteBackData.

5.1 RISC-V Εντολές

Όπως αναφέρεται και στο εγχειρίδιο εντολών RISC-V, η αρχιτεκτονική εντολών RISC-V (ISA) τοποθετεί τους καταχωρητές πηγής (rs1 και rs2) και προορισμού (rd) στις ίδιες θέσεις σε όλες τις μορφές εντολών, διευκολύνοντας έτσι την αποκωδικοποίηση. Εξαιρώντας τις 5-bit άμεσες τιμές που χρησιμοποιούνται στις εντολές CSR, οι άμεσες τιμές επεκτείνονται πάντα με πρόσημο και συνήθως τοποθετούνται προς τα αριστερά της εντολής, μειώνοντας την πολυπλοκότητα του υλικού. Ιδιαίτερα, το bit προσήμου για όλες τις άμεσες τιμές βρίσκεται πάντα στη θέση 31 της εντολής, επιταχύνοντας την επεξεργασία της επέκτασης προσήμου.

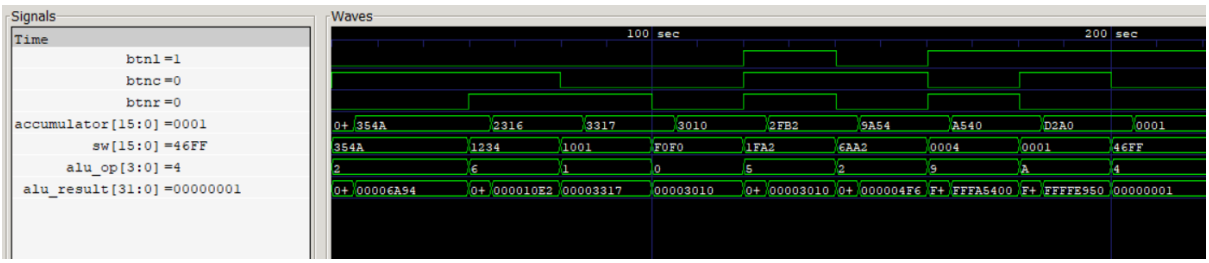

Figure 2: Κυματομορφές του testbench

Λόγω αυτής της σταθερής δομής, είναι δυνατό να αναθέσουμε τιμές στα πεδία rs1, rs2, rd και στις άμεσες τιμές (immediates) για όλους τους τύπους εντολών, ακόμα κι αν ορισμένα από αυτά δεν έχουν νόημα για μια συγκεκριμένη εντολή. Ωστόσο, μέσω του πεδίου opcode, καθορίζεται ποια από αυτά τα πεδία είναι σχετικά και χρησιμοποιούνται για κάθε εντολή, διασφαλίζοντας την ορθή αποκωδικοποίηση και επεξεργασία της.

Αξίζει να σημειωθεί ότι η εντολή LW είναι η μοναδική της οποίας το opcode δεν είναι κάποιο από τους τύπους R, S, I ή B.

6 Εργασία 5

Στην Άσκηση 5 υλοποιήθηκε ένας ελεγκτής πολλαπλών κύκλων που εκτελεί κάθε εντολή σε πέντε κύκλους ρολογιού, συνδέοντας τις μονάδες ελέγχου και διαδρομής δεδομένων. Ο ελεγκτής περιλαμβάνει ένα FSM πέντε καταστάσεων (IF, ID, EX, MEM, WB) για τη διαχείριση της εκτέλεσης εντολών. Επιπλέον, παράχθηκαν τα απαραίτητα σήματα ελέγχου (flags), όπως τα ALUCtrl, ALUSrc, MemRead, MemWrite, MemtoReg, RegWrite, loadPC και PCSrc, εξασφαλίζοντας την ορθή αλληλεπίδραση των μονάδων του επεξεργαστή.

6.1 Δεδομένα ROM

Στα πλαίσια της εργασίας, μας δόθηκαν μνήμη δεδομένων και μνήμη εντολών, τις οποίες έπρεπε να συνδέσουμε στον επεξεργαστή και να χρησιμοποιήσουμε στο testbench. Ας δούμε τα δεδομένα της μνήμης εντολών (ROM) για να αποκτήσουμε μια ιδέα σχετικά με τα αναμενόμενα αποτελέσματα της προσομοίωσης στο testbench.

Ανοίγοντας το αρχείο `rom_bytes.data` στο Notepad, παρατηρούμε ότι κάθε γραμμή περιέχει 8 bits, ενώ συνολικά υπάρχουν 512 γραμμές. Ωστόσο, ένα μεγάλο μέρος του αρχείου αποτελείται αποκλειστικά από μηδενικά, συγκεκριμένα από τη γραμμή 113 και κάτω, τα οποία αγνοούμε προς το παρόν. Εστιάζουμε, λοιπόν, στο αρχείο μέχρι τη γραμμή 112.

Οι εντολές είναι 32-bit, δηλαδή καταλαμβάνουν 4 γραμμές η καθεμία. Έτσι, για 112 γραμμές περιμένουμε συνολικά $112/4 = 28$ εντολές. Ωστόσο, παρατηρούμε ότι από τις γραμμές 81 έως 96 υπάρχουν 4 εντολές που περιέχουν μόνο μηδενικά. Αυτές οι γραμμές θα προσπεραστούν λόγω μιας εντολής BEQ που προηγείται. Τελικά, περιμένουμε την εκτέλεση 20 συνολικών εντολών (με την 20ή να είναι η BEQ), ενώ οι υπόλοιπες 4 μετά τη BEQ οδηγούν σε συνολικά 24 εντολές.

Για τις 24 εντολές, επιβεβαιώσαμε ότι διαβάζονται σωστά μέσα από τα αποτελέσματα του testbench, παρακολουθώντας την τιμή `instr` (σε δεκαεξαδική μορφή). Στη συνέχεια, η τιμή αυτή εισήχθη σε RISC-V decoder, μέσω του οποίου διαπιστώσαμε τόσο τον τύπο της εντολής όσο και την αντιστοιχία της με την 32-bit δυαδική αναπαράσταση που περιέχεται στο αρχείο της μνήμης εντολών. Συγκεκριμένα, οι 24 εντολές που εκτελούνται ειναι οι εξής και φαίνονται αναλυτικότερα στα αντίστοιχα figures:

1. `ADDI`
2. `ADDI`

3. ADD
4. ADDI
5. ADDI
6. ADD
7. SUB
8. SLL
9. SLT
10. XOR
11. AND
12. SRL
13. OR
14. SRA
15. SW
16. LW
17. ANDI
18. ORI
19. SRLI
20. BEQ
21. SLTI
22. XORI
23. SLLI
24. SRAI

6.2 Σχεδιασμος FSM

Το FSM (Finite State Machine) που σχεδιάστηκε για τη μονάδα ελέγχου του επεξεργαστή αποτελείται από τρία βασικά always blocks:

- **Αποθήκευση Κατάστασης (Ακολουθιακή Λογική):** Διατηρεί την τρέχουσα κατάσταση (`current_state`) και ενημερώνεται συγχρονισμένα με το ρολόι (clk) ή το σήμα επαναφοράς (rst). Σε περίπτωση επαναφοράς, η κατάσταση επανέρχεται στην αρχική (Instruction Fetch (IF)).
- **Λογική Επόμενης Κατάστασης (Συνδυαστική Λογική):** Καθορίζει τη μετάβαση από την τρέχουσα κατάσταση στην επόμενη (`next_state`), με βάση τη φάση εκτέλεσης της εντολής και τον τύπο της. Η λογική αυτή εξασφαλίζει τη σωστή ροή μεταξύ των σταδίων IF, ID, EX, MEM, WB.
- **Λογική Εξόδων (Συνδυαστική Λογική):** Παράγει τα σήματα ελέγχου που αντιστοιχούν σε κάθε στάδιο της εκτέλεσης, σύμφωνα με τις απαιτήσεις της εκφώνησης.

Στην κατανόηση της λειτουργίας του FSM μπορεί να βοηθήσει και το σχετικό διάγραμμα.

6.3 Αποτελέσματα testbench

Όπως παρατηρούμε και στην εικόνα, οι περισσότερες εντολές εκτελούνται σε τέσσερις κύκλους ρολογιού. Αυτό οφείλεται στο γεγονός ότι οι περισσότερες από αυτές δεν απαιτούν

την είσοδο στο στάδιο MEM του FSM. Αντίθετα, οι εντολές LW και SW εισέρχονται στο στάδιο MEM, όπως ακριβώς προβλέπεται από τη λειτουργία του FSM, προκειμένου να πραγματοποιηθούν οι απαιτούμενες λειτουργίες μνήμης.

Ενδεικτικά, παρουσιάζονται δύο εντολές: η `ADDI` και η `ADD`. Τα αποτελέσματα των κυματομορφών, οι καταχωρητές, τα flags και τα αποτελέσματα των πράξεων (dAddress) εμφανίζονται στις εικόνες `ADDI Result` και `ADD Result`.

Επίσης, παρατηρούνται και οι εντολές `SW` και `LW`, στις οποίες τα flags `MemWrite` και `MemRead` ενεργοποιούνται κατάλληλα, όπως αναμένεται.

Τέλος, παρατηρείται και η εντολή `BEQ`, όπου τα flags zero της ALU, loadPC, και PCSrc ενεργοποιούνται σωστά, με τον Program Counter να ανανεώνεται κατάλληλα στην τιμή του immediate (PC + 16).

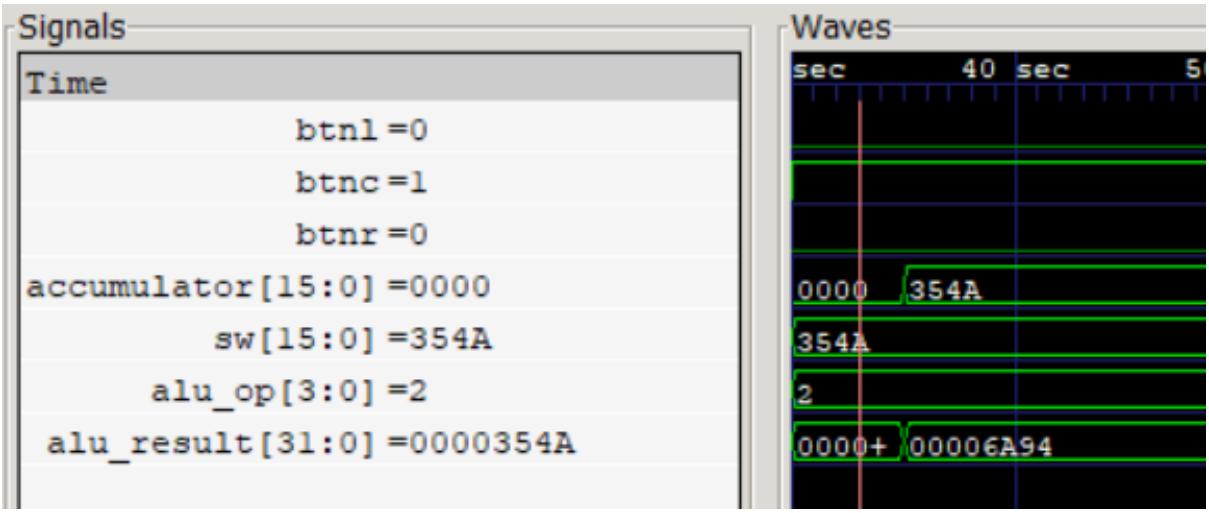

Figure 3: Κυματομορφές της 1ης πράξης

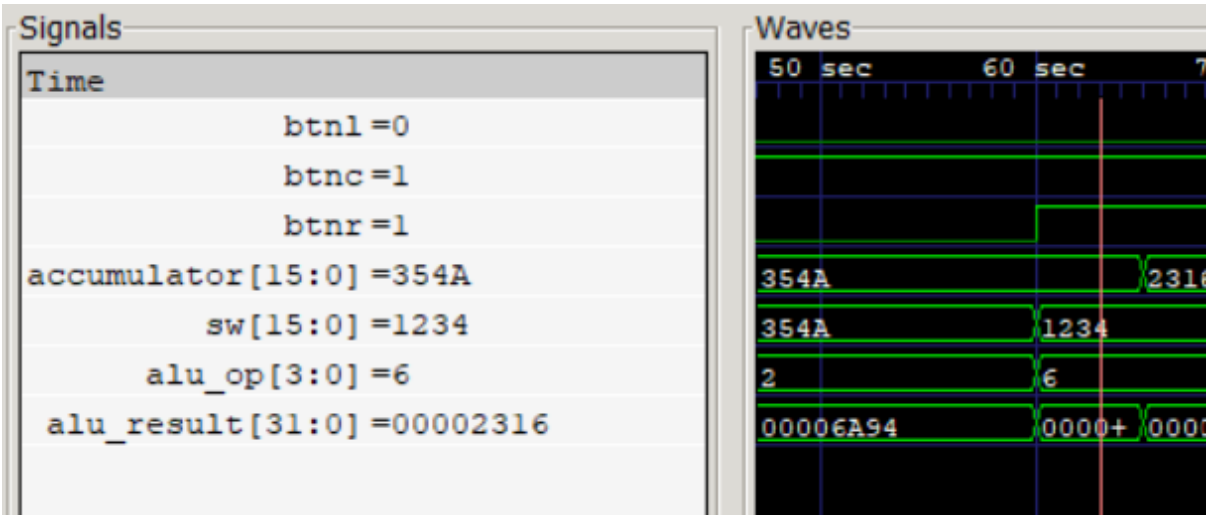

Figure 4: Κυματομορφές της 2ης πράξης

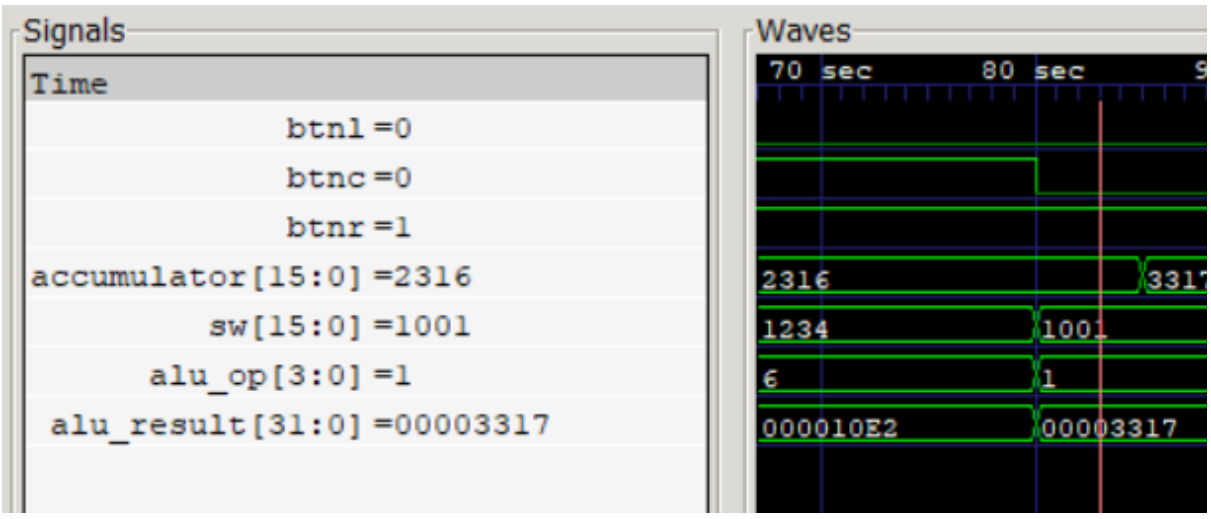

Figure 5: Κυματομορφές της 3ης πράξης

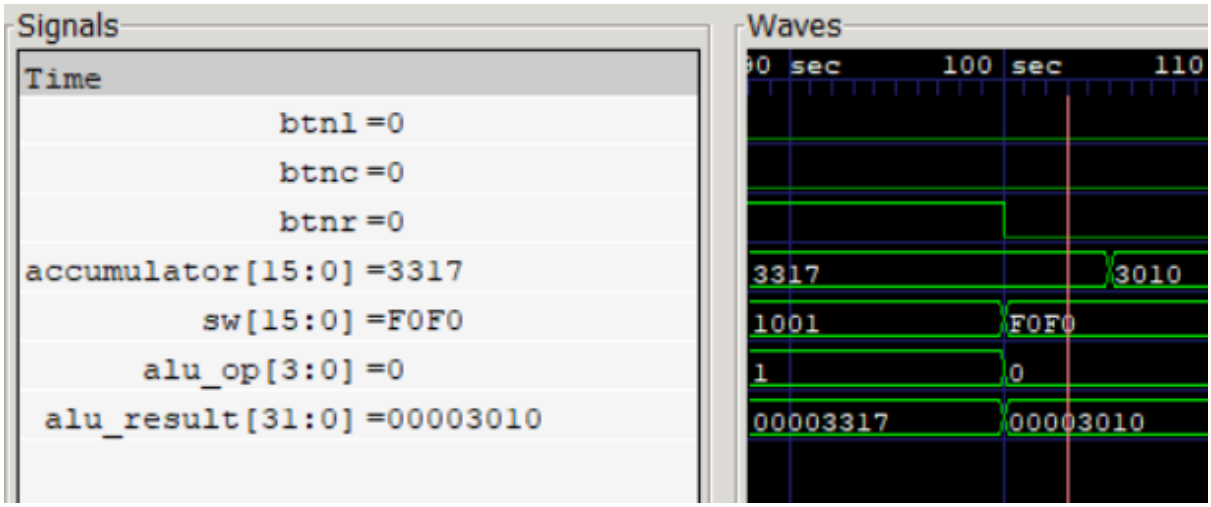

Figure 6: Κυματομορφές της 4ης πράξης

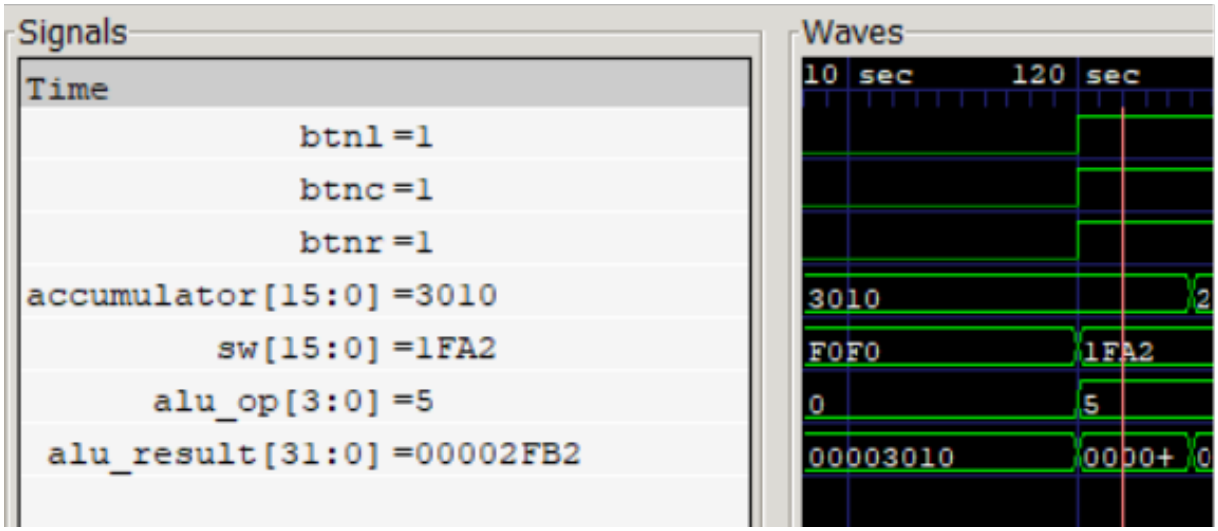

Figure 7: Κυματομορφές της 5ης πράξης

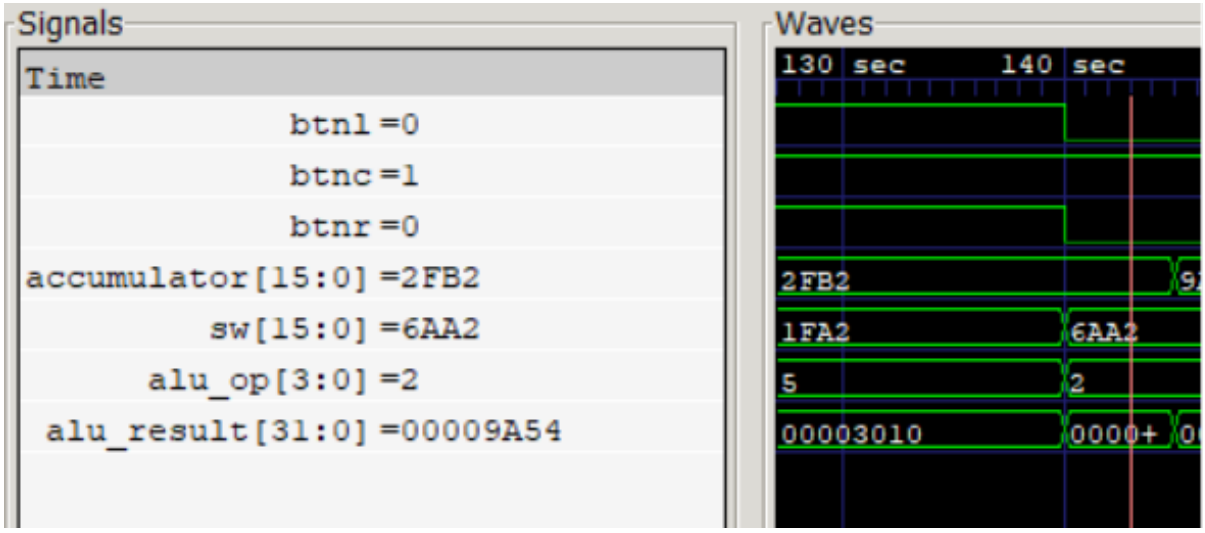

Figure 8: Κυματομορφές της 6ης πράξης

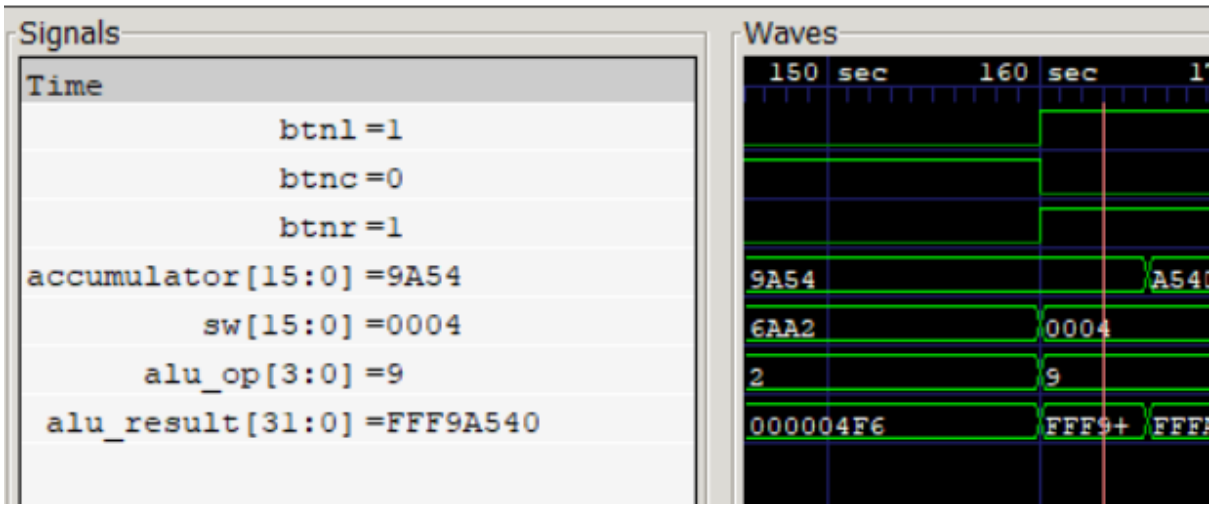

Figure 9: Κυματομορφές της 7ης πράξης

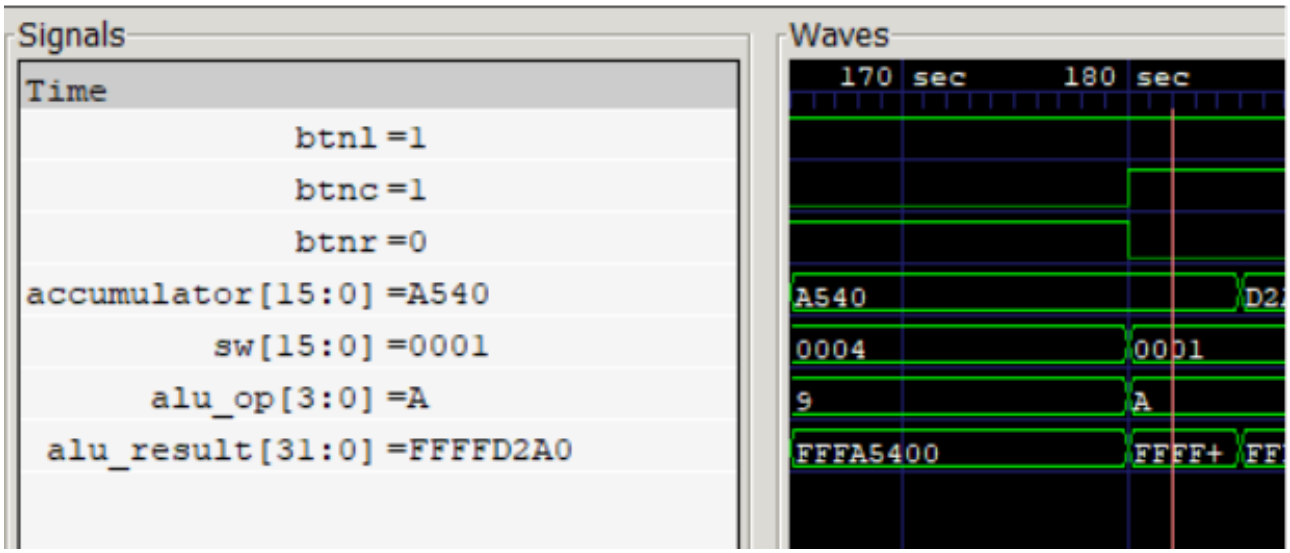

Figure 10: Κυματομορφές της 8ης πράξης

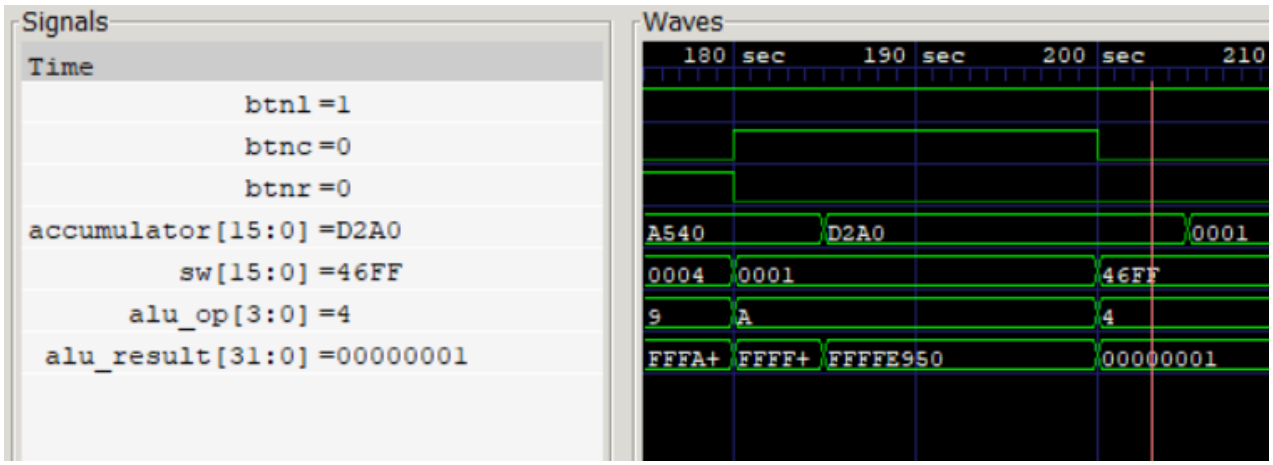

Figure 11: Κυματομορφές της 9ης πράξης

Figure 12: Instruction 1

Figure 13: Instruction 2

Figure 14: Instruction 3

Figure 15: Instruction 4

Figure 16: Instruction 5

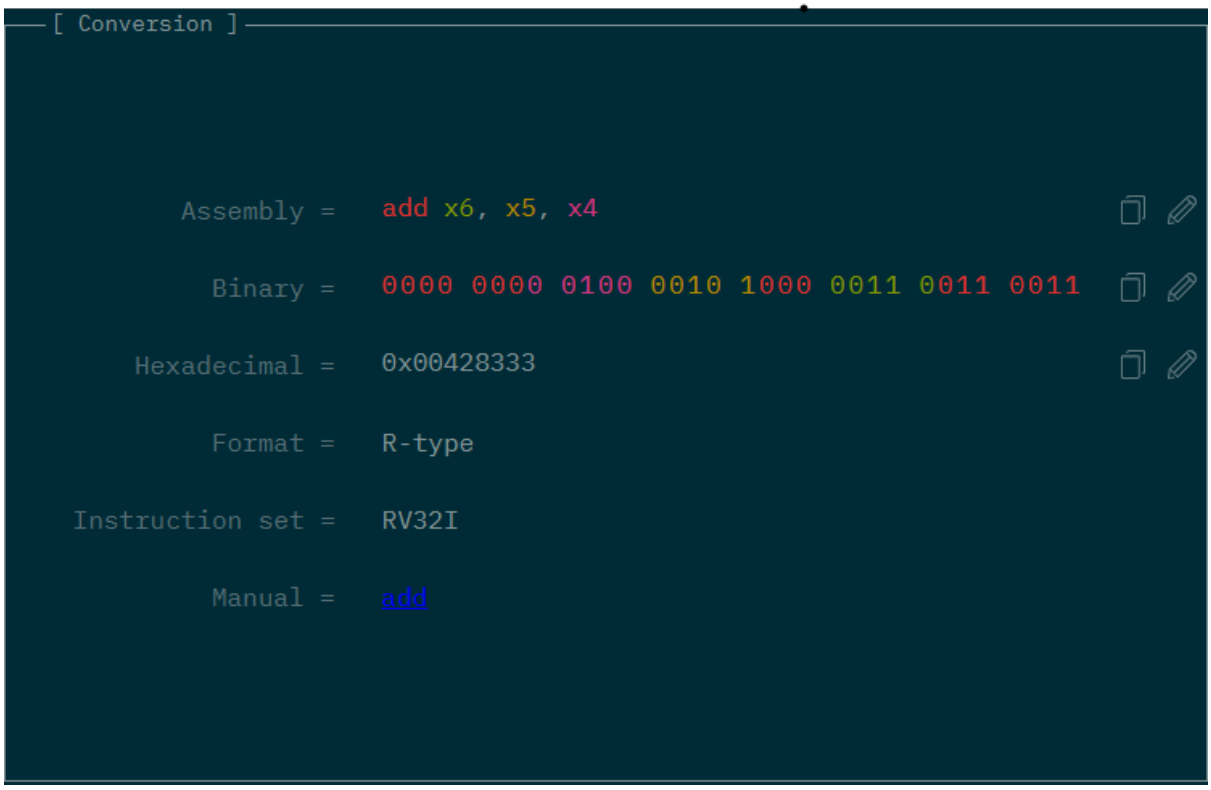

Figure 17: Instruction 6

Figure 18: Instruction 7

Figure 19: Instruction 8

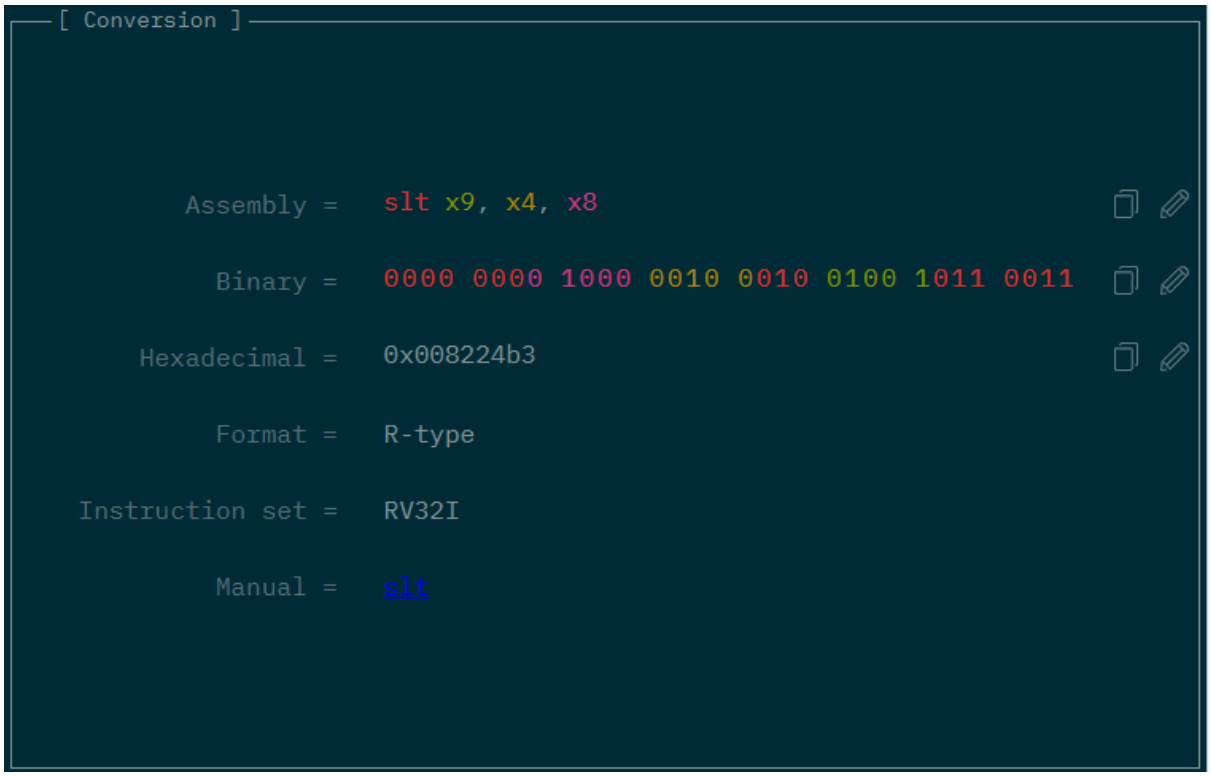

Figure 20: Instruction 9

Figure 21: Instruction 10

Figure 22: Instruction 11

Figure 23: Instruction 12

Figure 24: Instruction 13

Figure 25: Instruction 14

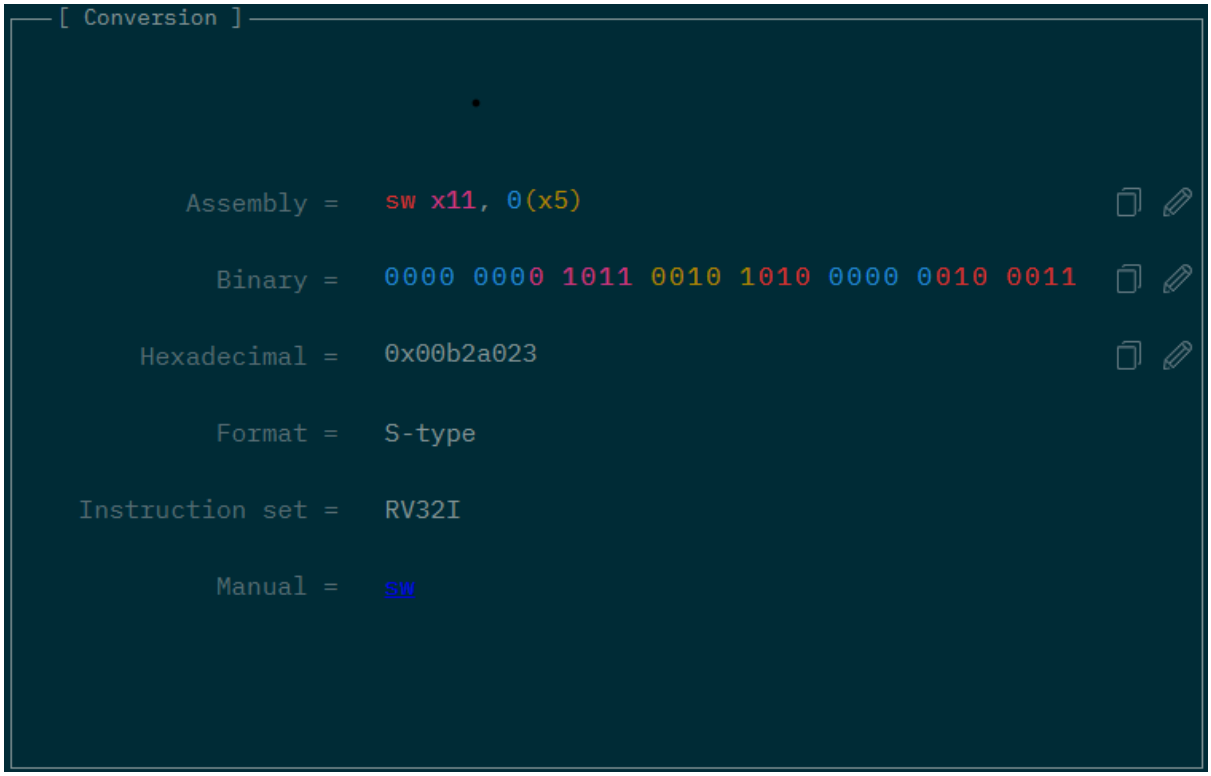

Figure 26: Instruction 15

Figure 27: Instruction 16

Figure 28: Instruction 17

Figure 29: Instruction 18

Figure 30: Instruction 19

Figure 31: Instruction 20

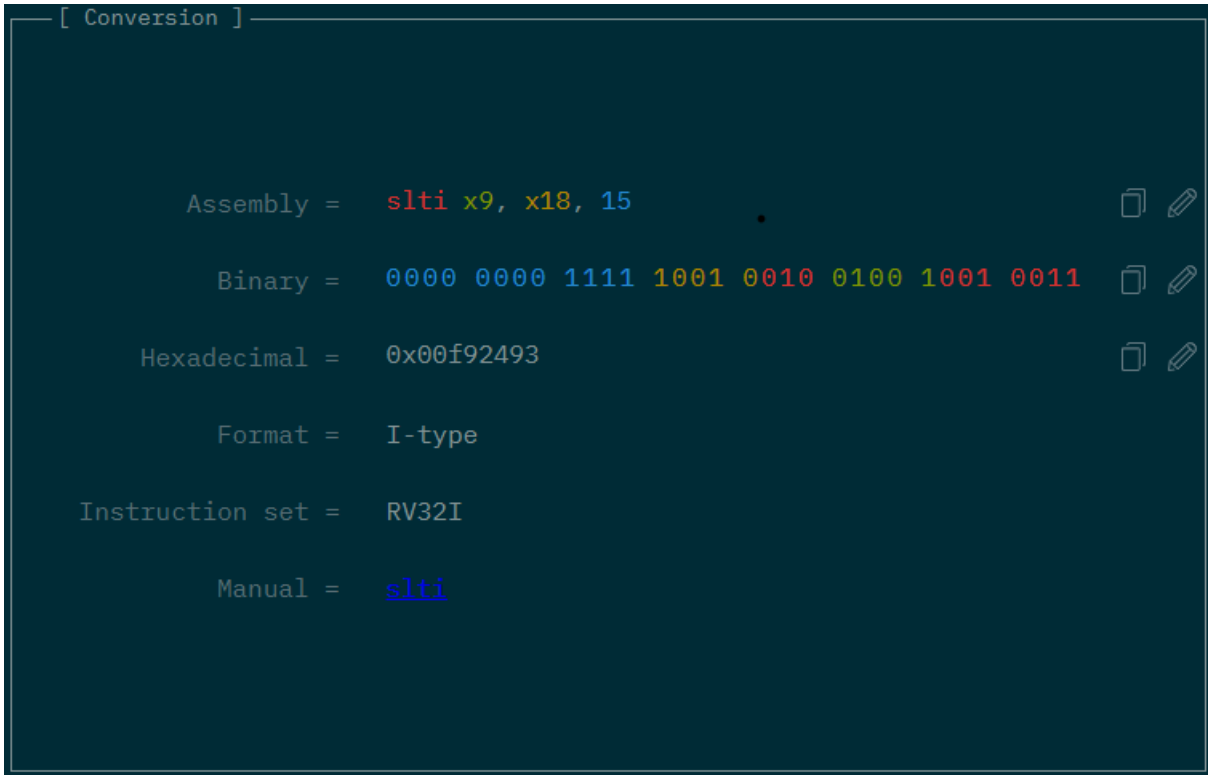

Figure 32: Instruction 21

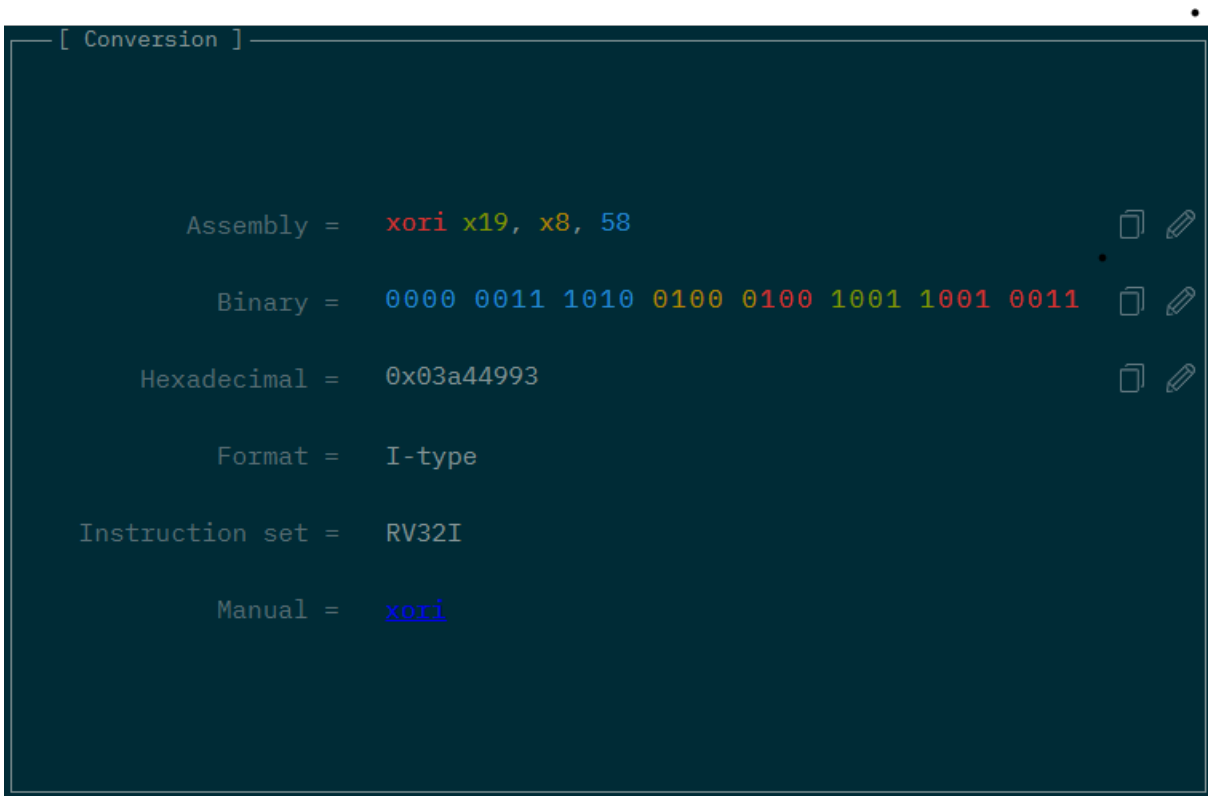

Figure 33: Instruction 22

Figure 34: Instruction 23

Figure 35: Instruction 24

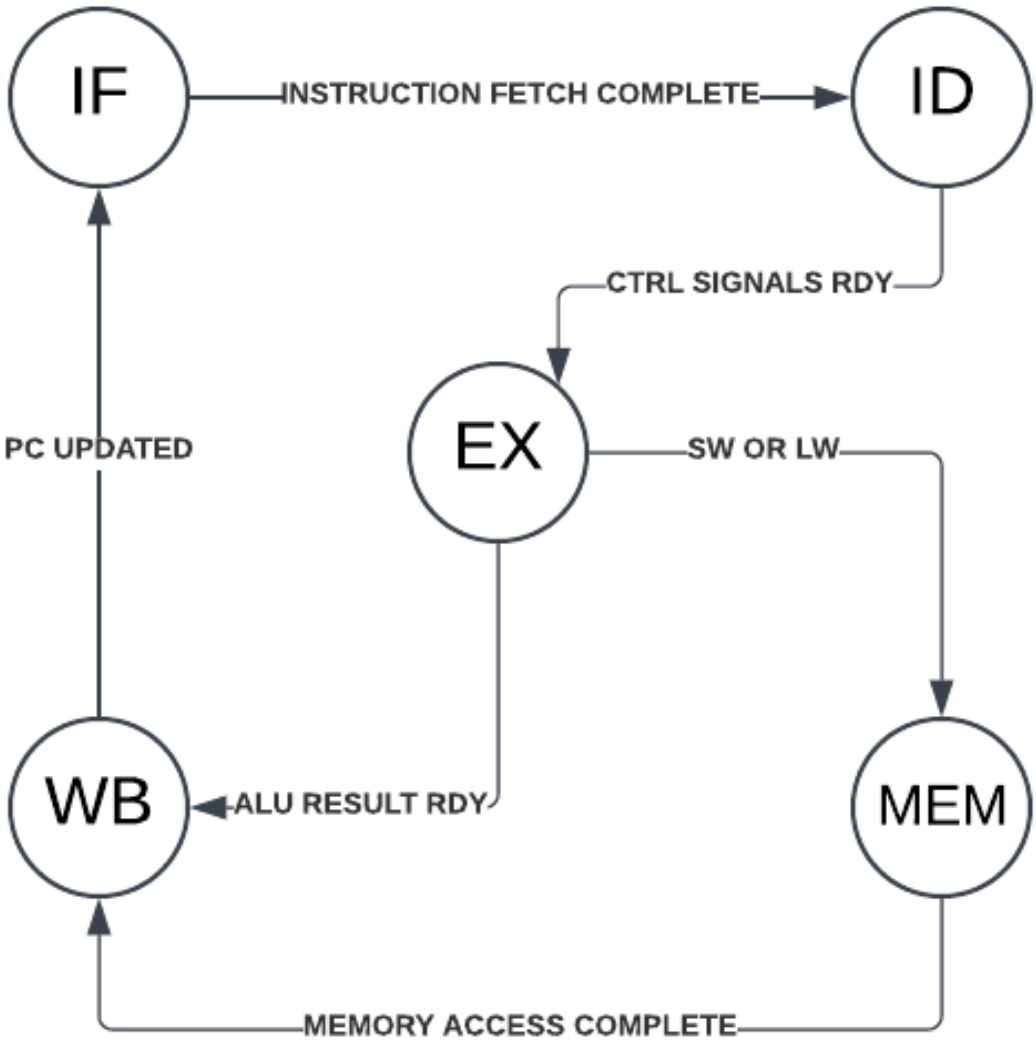

Figure 36: Διάγραμμα του FSM

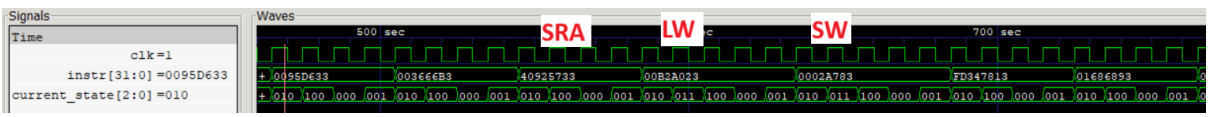

Figure 37: Instructions and Current State

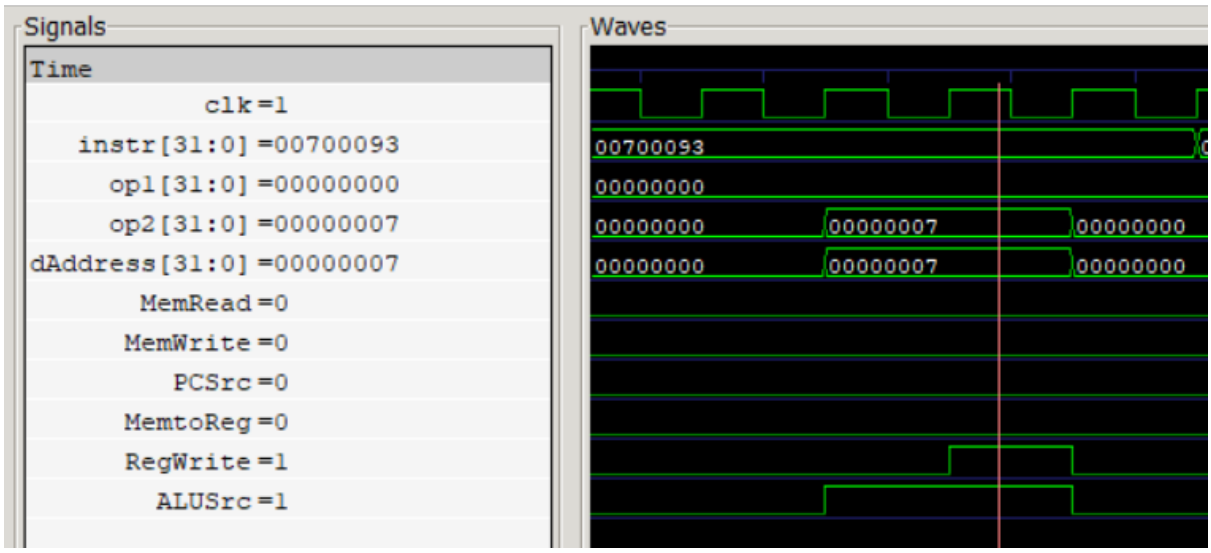

Figure 38: ADDI Result

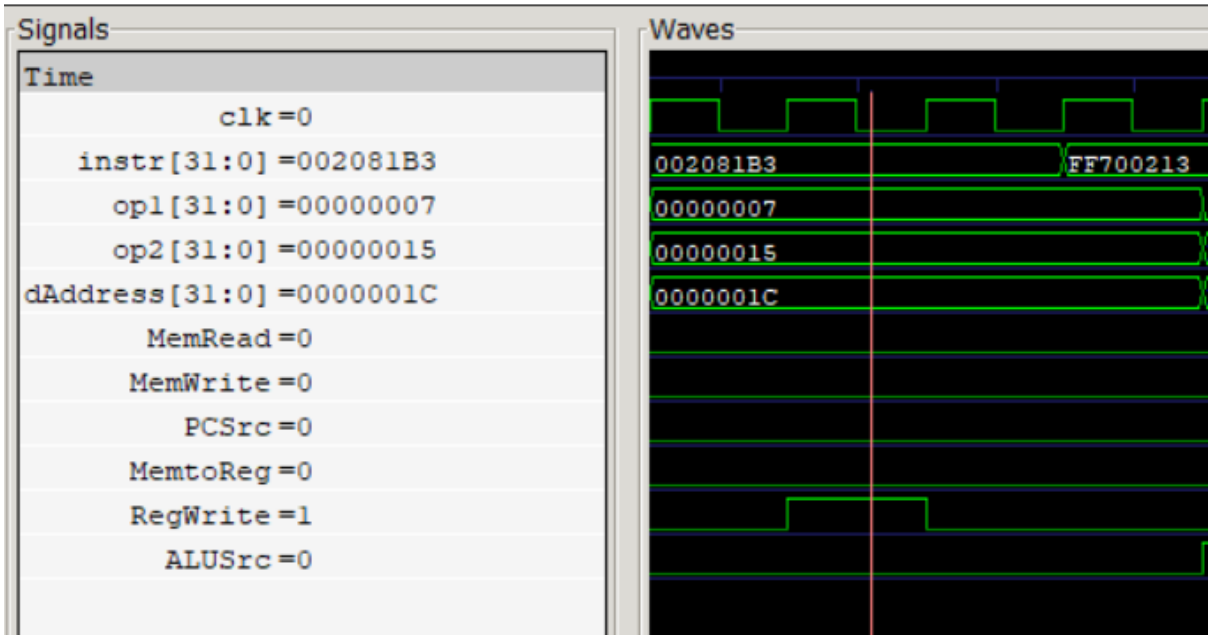

Figure 39: ADD Result

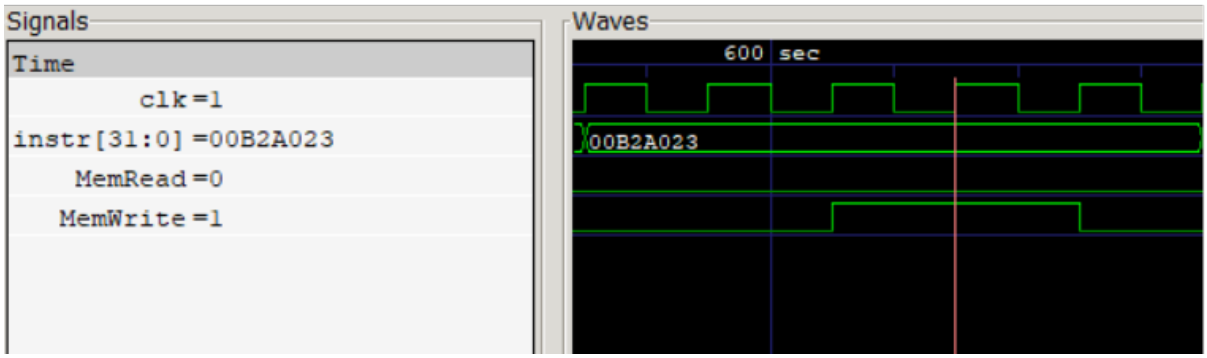

Figure 40: SW Instruction

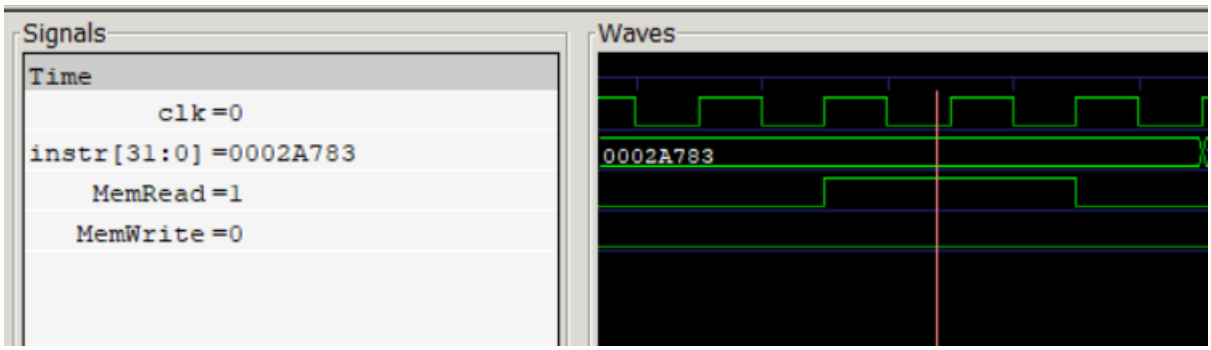

Figure 41: LW Instruction

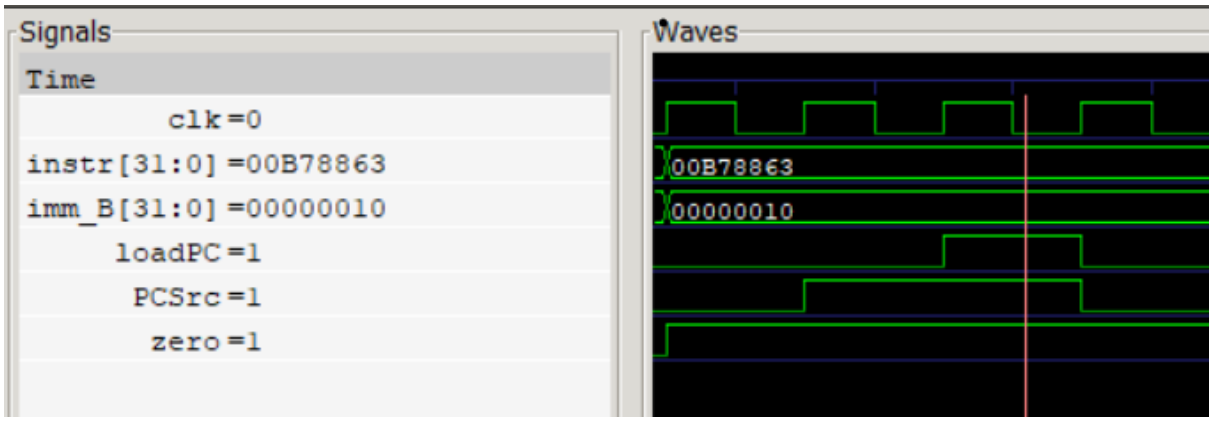

Figure 42: BEQ Instruction

Αναφορές

Για τη βελτίωση της γραμματικής και συντακτικής δομής του κειμένου, χρησιμοποιήθηκε το ChatGPT, το οποίο χρησιμοποιήθηκε αποκλειστικά για τη διόρθωση λαθών και την επεξεργασία του κειμένου σε επίπεδο γλώσσας. Το εργαλείο αυτό αξιοποιήθηκε μόνο για γραμματικές και συντακτικές διορθώσεις και δεν συνέβαλε στην ανάπτυξη ή συγγραφή του περιεχομένου της εργασίας.