Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργασία Σχεδίασης RISC-V σε γλώσσα Verilog στο πλαίσιο του μαθήματος Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής Ι 2023-2024

Γαλάνη Δήμητρα

Άσκηση 1

Αριθμητική/Λογική μονάδα ALU (Arithmetic Logic Unit)

Για την υλοποίηση της ALU αρχικά καθορίσαμε τις σταθερές στο αρχείο parameters.v χρησιμοποιώντας τον τύπο parameter σύμφωνα με το πρότυπο της εκφώνησης:

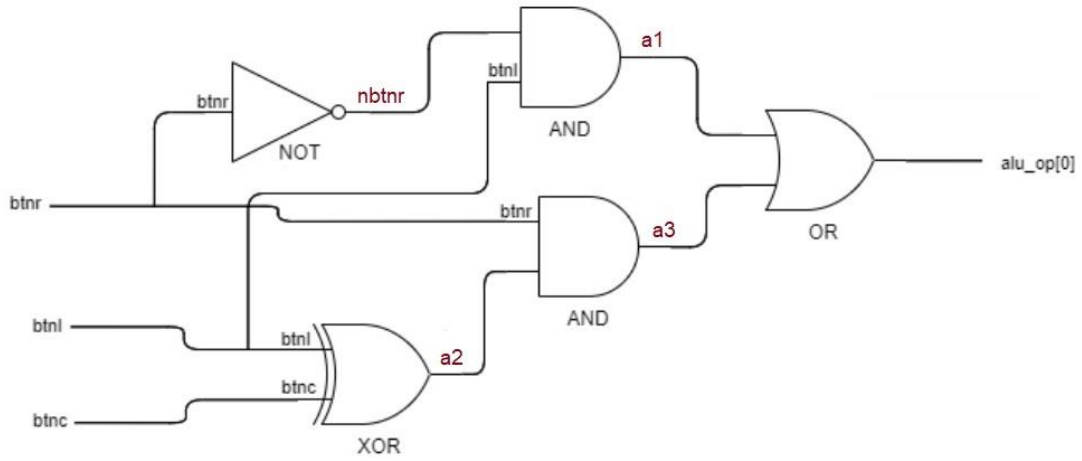
parameter[3:0] ALUOP_SUB = 4'b0110;

Στο αρχείο αυτό έχουμε ορίσει όλες τις παραμέτρους που θα χρησιμοποιήσουμε σε αυτήν την εργασία. Στην συνέχεια υλοποιήσαμε έναν multiplexer κατά τον οποίο ανάλογα με την τιμή του σήματος alu_op διαλέγουμε την κατάλληλη πράξη που πρέπει να εκτελέσουμε. Το σήμα zero γίνεται υψηλό μόνο όταν το αποτέλεσμα της ALU είναι μηδενικό. Αξίζει να επισημάνουμε ότι χρησιμοποιώντας τον τύπο signed wire εξασφαλίζουμε ότι οι τελεστές που εισάγουμε στην ALU θα αντιμετωπιστούν ως προσημασμένοι.

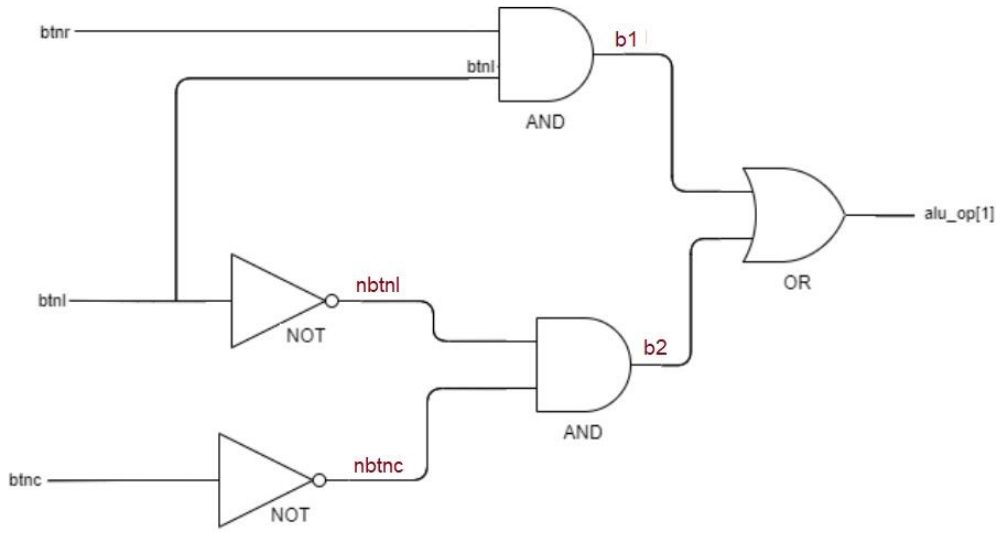
Άσκηση 2

Calculator

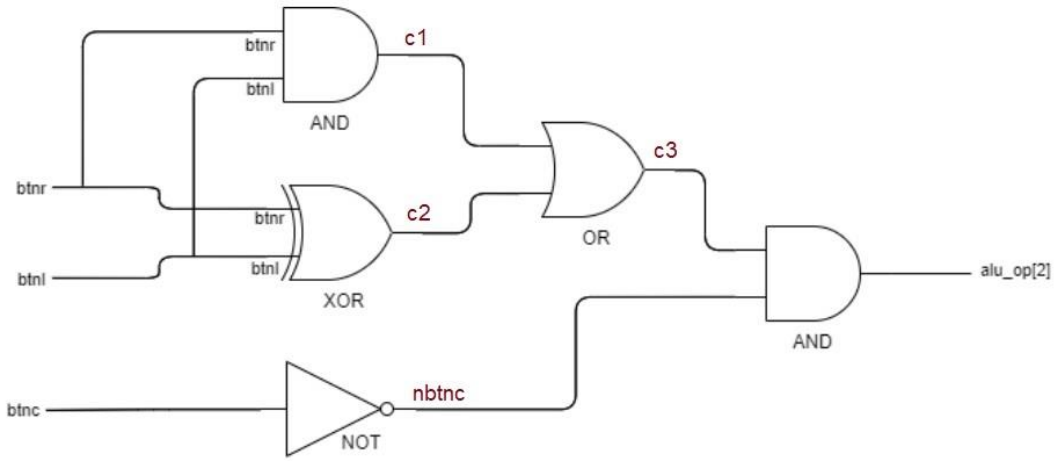
Για την παραγωγή του αρχείου decoder.v αρχικά ονομάτισα της εξόδους κάθε λογικής πύλης σύμφωνα με τα σχήματα και στην συνέχεια χρησιμοποιώντας structural verilog υλοποίησα τα κυκλώματα, γράφοντας στο κατάλληλο πρότυπο κύκλωμα επιπέδου πύλης την έξοδο ακολουθούμενη από τις αντίστοιχες εισόδους.



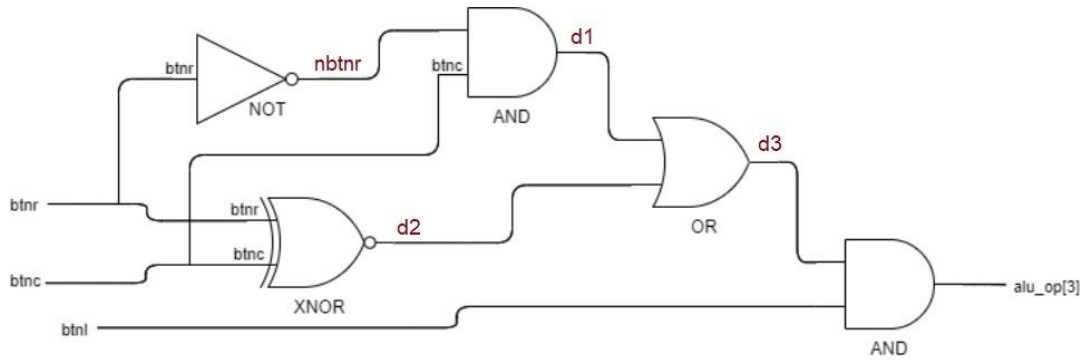
Εικόνα 1: Κύκλωμα για τον καθορισμό του alu_op[0].



Εικόνα 2: Κύκλωμα για τον καθορισμό του alu_op[1].



Εικόνα 3: Κύκλωμα για τον καθορισμό του alu_op[2].



Εικόνα 4: Κύκλωμα για τον καθορισμό του alu_op[3].

Για την υλοποίηση του calc.v αρχείου έγιναν αρχικά οι κατάλληλες συνδέσεις και οι κατάλληλες επεκτάσεις προσήμου για να έχουν οι είσοδοι της ALU μέγεθος 32bit. Στην συνέχεια εισάχθηκαν τα δύο υποκυκλώματα που περιγράψαμε παραπάνω και αντιστοιχήθηκαν οι θύρες έτσι ώστε να συμβαδίζουν με την εκφώνηση.

Αν το btnu είναι ενεργοποιημένο ο accumulator, ο οποίος κρατάει την τρέχουσα τιμή της αριθμομηχανής, μηδενίζεται σύγχρονα με το ρολόι. Δηλαδή το btnu δρα σαν το clear κουμπί του calculator. Ο σύγχρονος αυτός μηδενισμός επιτυγχάνεται με ένα always block του οποίου η λίστα ευαισθησίας περιέχει την θετική ακμή του ρολογιού, σε συνδυασμό με μια εντολή ελέγχου που ελέγχει αν το σήμα btnu είναι υψηλό.

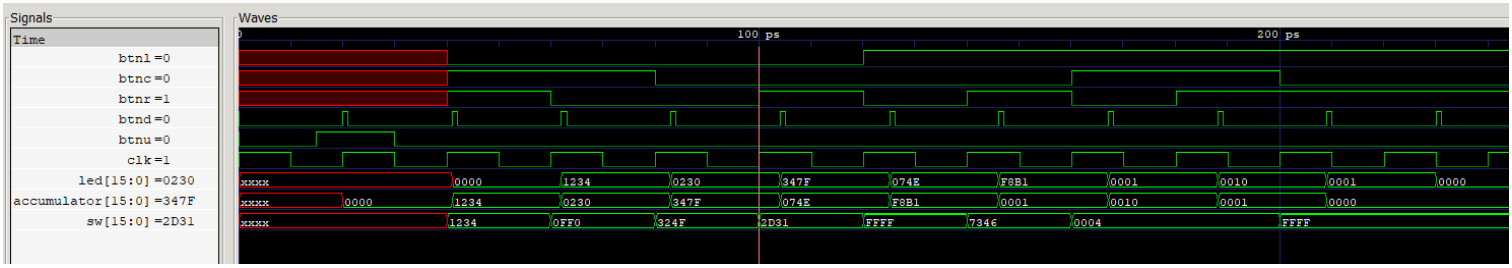
Η υλοποίηση του btnd έχει γίνει με ασύγχρονο τρόπο. Με την θετική ακμή του btnd το αποτέλεσμα της alu μεταφέρεται με non-blocking τρόπο στον accumulator και όπως και η τιμή του accumulator μεταφέρεται με non-blocking τρόπο στα leds. Επειδή χρησιμοποιείται non-blocking ανάθεση το led ενημερώνεται στον επόμενο κύκλο ρολογιού.

Για το test bench αρχείο του calc.v ενεργοποιώ το ρολόι με ένα initial block το οποίο εκτελείται μόνο μια φορά στην αρχή του χρόνου. Σε ένα always block ρυθμίζω το ρολόι έτσι ώστε να αλλάζει κατάσταση κάθε 10 μονάδες χρόνου. Σε ένα άλλο always block θέτω το btnd να ενεργοποιείται κάθε 20 μονάδες χρόνου για μια μονάδα χρόνου όταν όλα τα σήματα (δηλαδή τα btnl, btnc, btnr, sw) έχουν σταθεροποιηθεί στις επιθυμητές τιμές έτσι ώστε να αποθηκεύεται το επιθυμητό σήμα και να υπολογίζεται σωστά η πράξη που θέλω να πραγματοποιήσω.

Οι διαφορετικές τιμές σημάτων, δηλαδή η προσθήκη νέας πράξης και αριθμού για να γίνει η πράξη αυτή μαζί με το προηγούμενο αποτέλεσμα φορτώνονται κάθε 20 μονάδες χρόνου.

Στις κυματομορφές βλέπουμε ότι τα σήματα εισόδου που καθορίζουν την λειτουργία του calculator αλλάζουν τιμή σύγχρονα με το ρολόι.

Με το που ενεργοποιηθεί το σήμα, η τιμή του sw περνάει στον accumulator και η προηγούμενη τιμή του accumulator περνάει στα leds.



Εικόνα 5: Κυματομορφές προσομοίωσης Άσκησης 2

Άσκηση 3

Αρχείο καταχωρητών - Regfile

Για τον σχεδιασμό του regfile.v ακολουθήθηκαν οι οδηγίες της εκφώνησης και αρχικά γίνεται η αρχικοποίηση όλων των καταχωρητών σε μηδέν. Συνδέουμε τα readData με τον καταχωρητή που θα έχει κάθε φορά το νούμερο readReg. Στην θετική ακμή του ρολογιού εάν το σήμα write είναι υψηλό γίνεται η καταγραφή των writeData στον αντίστοιχο καταχωρητή. Υλοποιήσαμε το κύκλωμα με τέτοιον τρόπο έτσι ώστε στην ακμή του ρολογιού να διαβάζει τις τιμές και μετά να εγγράφει τα δεδομένα.

Στην πράξη δεν θα προκύπτει ταυτόχρονη ανάγνωση και εγγραφή εφόσον η ανάγνωση γίνεται αποκλειστικά στο ID στάδιο του FSM ενώ η εγγραφή γίνεται στο WB στάδιο του FSM.

Άσκηση 4

Διαδρομή δεδομένων – Datapath

Έχω δημιουργήσει ένα αρχείο που να περιέχει τις παραμέτρους το οποίο θα χρησιμοποιήσω τόσο στο αρχείο datapath.v οσο και στο αρχείο multicycle.v γιατί σε διάφορα σημεία του κώδικα χρησιμοποιώ αυτές αντί για δυαδικούς αριθμούς.

Για την υλοποίηση του datapath έχω χρησιμοποιήσει συνδυαστική λογική και το κύκλωμα που έχω υλοποιήσει δεν έχει μνήμη αλλά ανατίθενται οι τιμές μέσω καλωδίων. Τα σήματα ελέγχου είναι αυτά που θα αλλάζουν σε κάθε κύκλο ρολογιού ανάλογα με το στάδιο FSM που θα βρίσκεται η μηχανή με συνέπεια να αλλάζουν κατευθείαν και οι έξοδοι αφού θα έχουν αλλάξει οι είσοδοι.

Χρησιμοποιώ έναν multiplexer που ελέγχεται από το σήμα ALUSrc για να επιλεγεί κατάλληλα ο δεύτερος τελεστής ανάμεσα από την δεύτερη θύρα του αρχείου καταχωρητών και από τα δεδομένα του Immediate Generator.

Παρομοίως για να επιλεγούν τα δεδομένα που θα καταγραφούν πίσω στους καταχωρητές χρησιμοποιείται ένας multiplexer ο οποίος ελέγχεται από το σήμα ελέγχου MemtoReg.

Η επιλογή για το τρόπο ανανέωσης του PC δηλαδή αν θα προχωρήσει στην επόμενη εντολή ή αν θα κάνει διακλάδωση επιλέγεται στην θετική ακμή του ρολογιού ανάλογα με τα σήματα PCSrc και Zero.

Όσον αφορά την αποκωδικοποίηση του instruction:

Στα 32bit του instruction περιέχονται οι εξής πληροφορίες:

- ποια εντολή εκτελείται
- νούμερα των καταχωρητών ανάγνωσης

- νούμερο καταχωρητή εγγραφής
- immediate δεδομένα που χρησιμοποιούνται στην εντολές τύπου I στις οποίες η πηγή δεδομένων υπάρχει μέσα στο instruction, δηλαδή τα δεδομένα που χρησιμοποιούνται δεν είναι αποθηκευμένα στους καταχωρητές αλλά εισάγονται κατά την κλήση της εντολής.

Από την εκφώνηση οι εντολές χωρίζονται στις εξής κατηγορίες:

- R-TYPE: ADD, SUB, AND, OR, XOR, SLT, SLL, SRL, SRA ALU
- I-TYPE: ADDI, ANDI, ORI, XORI, SLTI, SLLI, SRLI, SRAI
- S-TYPE: LW, SW (στον κώδικα τις χωρίζουμε σε 2 διαφορετικές κατηγορίες γιατί τα χωρία τους διαφέρουν)
- B-TYPE: BEQ

Από το εγχειρίδιο των εντολών RISC-V που μας δόθηκε γίνεται κατανοητό ότι οι εντολές που ανήκουν στις ίδιες κατηγορίες έχουν ίδιο opcode δηλαδή τα bits instr[6:0] είναι ίδια. Με εξαίρεση βέβαια τις εντολές SW και LW οι οποίες έχουν διαφορετικό opcode.

- Ο πρώτος καταχωρητής ανάγνωσης βρίσκεται στα bits instr[19:15].
- Ο δεύτερος καταχωρητής ανάγνωσης εάν υπάρχει θα βρίσκεται στα bits instr[24:20].
- Ο καταχωρητής εγγραφής εάν υπάρχει θα βρίσκεται στα bits instr[11:7].

Οπότε συνδέω τα bits αυτά με τις κατάλληλες εισόδους του regfile, εάν η εντολή που εκτελείται δεν έχει καταχωρητή εγγραφής ή καταχωρητή ανάγνωσης δεν υπάρχει πρόβλημα γιατί τα σήματα ελέγχου καθορίζουν την πληροφορία που θα περάσει στα επόμενα στάδια.

Προκειμένου να καθορίσουμε τα δεδομένα για τις άμεσες εντολές χρησιμοποιούμε την υλοποίηση ενός multiplexer ο οποίος ανάλογα με τον τύπο της εντολής επιλέγει τα κατάλληλα bits σύμφωνα με το εγχειρίδιο για να δημιουργηθούν τα IG_data, τα οποία κατά την ανάθεση έχουν επεκταθεί ως προς το πρόσημό τους με την βοήθεια του τελεστή concatenation.

Άσκηση 5

Ελεγκτής πολλαπλών κύκλων -Multicycle

Εισάγοντας το instance του υποκυκλώματος datapath περνάω την παράμετρο INITIAL_PC έτσι ώστε να κληρονομηθεί η τιμή της παραμέτρου στο κάτω επίπεδο από το ανώτερο ιεραρχικά επίπεδο multicycle και επομένως όταν πραγματοποιηθεί μια αλλαγή της παραμέτρου στο πάνω επίπεδο να αλλάξει και στο κάτω επίπεδο.

Υλοποιώ το σήμα ALUCtrl που επιλέγει την πράξη που θα κάνει η ALU ανάλογα με την εντολή ελέγχοντας το opcode, το funct3 και το funct7 για κάθε εντολή με έναν multiplexer.

Με όμοιο τρόπο υλοποιώ το ALUSrc το οποίο γίνεται υψηλό εάν εκτελώ I-type εντολή ή B-type εντολή με σκοπό ο δεύτερος τελεστής της ALU να παρθεί από τα άμεσα δεδομένα.

Το σήμα PCSrc το υλοποιώ με έναν πολυπλέκτη και φροντίζω να γίνεται υψηλό σύμφωνα με τις συνθήκες της εκφώνησης.

Η κατάσταση του σήματος MemtoReg μας ενδιαφέρει μόνο κατά την διάρκεια του σταδίου WB και μας είναι αδιάφορο για την υπόλοιπη διάρκεια οπότε μπορούμε να το υλοποιήσουμε εκτός procedural block.

Τα σήματα ελέγχου MemRead και MemWrite γίνονται υψηλά μόνο κατά την διάρκεια του σταδίου MEM - Memory Access οπότε τα υλοποιώ μέσα στο procedural block του FSM για τις εξόδους καθώς με την μεταβολή τους αλλάζουν οι έξοδοι αυτού του σταδίου. Στα υπόλοιπα στάδιο του FSM τα σήματα αυτά είναι χαμηλά για να μην προκληθούν λάθη. Το σήμα MemRead μας δείχνει πότε διαβάζουμε από την μνήμη, δηλαδή πότε χρησιμοποιούμε εντολή LW. Ενώ το σήμα MemWrite μας δείχνει πότε γράφουμε στην μνήμη, δηλαδή πότε χρησιμοποιούμε εντολή SW.

Κατά αναλογία το RegWrite πρέπει να γίνεται υψηλό μόνο κατά την διάρκεια του σταδίου WB - Write Back οπότε το υλοποιώ μέσα στο procedural block του FSM για τις εξόδους στο κατάλληλο case που προσδιορίζει το στάδιο WB. Το σήμα αυτό μας επιτρέπει ή μας απαγορεύει να γράψουμε στους καταχωρητές.

Το σήμα loadPC γίνεται υψηλό μόνο στο στάδιο WB για να φορτωθεί η νέα τιμή του Program Counter πριν επιστρέψω στο αρχικό στάδιο της μηχανής FSM, για να φορτωθεί η επόμενη εντολή. Οπότε το θέτω μονάδα μέσα στο procedural block του FSM για τις εξόδους στο στάδιο WB και στα υπόλοιπα στάδια το βάζω ίσο με 0.

Αφότου ορίσω τα σήματα ελέγχου υλοποιώ την μηχανή FSM.

Προκειμένου να περιγράψω μια μηχανή FSM χρειάζομαι αρχικά δύο σήματα τύπου reg τα οποία μοντελοποιούν την παρούσα (current_state) και την επόμενη κατάσταση (next_state). Εφόσον έχω 5 καταστάσεις χρειάζομαι 3bits για να απαριθμηθούν.

Για να περιγραφεί η μηχανή FSM χρησιμοποιώ 3 procedural blocks.

- Ένα always block που ενεργοποιείται με την θετική ακμή του ρολογιού για την αποθήκευση της κατάστασης (State Memory). Εάν είναι υψηλό το σήμα rst επαναφέρεται η FSM στην αρχική κατάσταση, ενώ αλλιώς πηγαίνει στην επόμενη κατάσταση.
- Ένα always block που ενεργοποιείται με την αλλαγή της κατάστασης και προσδιορίζει την επόμενη κατάσταση (Next State Logic). Σε κάθε κατάσταση που αλλάζει με την ακμή του ρολογιού από το always block που περιγράψαμε παραπάνω, ενημερώνουμε την τιμή της επόμενης κατάστασης που θα είναι

το επόμενο στάδιο, έτσι ώστε με το επόμενο ρολόι να προχωρήσει σε αυτό το στάδιο μέσω του State Memory κ.ο.κ.

- Ένα always block που προσδιορίζει τις τιμές των εξόδων (Outpout Logic) το οποίο ενεργοποιείται και αυτό με την αλλαγή της επόμενης κατάστασης. Με το που αλλάξει η επόμενη κατάσταση προσδιορίζουμε τις μεταβλητές που έχουν σημασία για το εκάστοτε στάδιο όπως έχουμε περιγράψει παραπάνω. Δηλαδή τις μεταβλητές που θέλουμε να είναι ενεργοποιημένες μόνο σε συγκεκριμένα στάδια.

Η λογική του FSM που δημιουργήσαμε περιγράφεται και με το παρακάτω σχηματικό διάγραμμα. Η μετάβαση σε κάθε κατάσταση γίνεται με το ρολόι. Η επόμενη κατάσταση που μπορεί να πάρει η μηχανή σε κάθε στάδιο φαίνεται με τα κατευθυνόμενα βέλη. Οι μεταβλητές που προσδιορίζουμε για την σωστή λειτουργία του κυκλώματος φαίνονται μέσα στα πλαίσια.

Εικόνα 6: Σχηματικό διάγραμμα FSM

Για να επαληθεύσουμε την σωστή λειτουργία του κυκλώματος δημιουργούμε ένα test bench. Στο αρχείο αυτό συνδέουμε τα αρχεία της μνήμης εντολών και της μνήμης δεδομένων που μας δόθηκαν με τις κατάλληλες θύρες, όπως επίσης δημιουργούμε ένα αντίτυπο για να ελέγξουμε το σύστημα μας. Αρχικά στέλνουμε το σήμα rst για να αρχικοποιηθεί η μηχανή μας και ορίζουμε το ρολόι να αλλάζει κατάσταση κάθε 10 μονάδες χρόνου. Έπειτα απενεργοποιούμε το rst και ορίζουμε το πότε να τελειώσει το κύκλωμα.

Κάνοντας αποκωδικοποίηση του αρχείου rom_bytes.data βρήκαμε ότι οι εντολές που θα τρέξουν στον επεξεργαστή μας είναι οι εξής:

PC(0x0000) ADDI $1, $0, 4

PC(0x0004) ADDI $2, $0, 1

PC(0x0008) ADDI $3, $0, 3

PC(0x000C) ADDI $4, $0, 7

PC(0x0010) ADDI $5, $0, -2

PC(0x0014) ADD $6, $3, $3

PC(0x0018) SUB $7, $6, $5

PC(0x001C) SLL $8, $3, $2

PC(0x0020) SLT $9, $8, $4

PC(0x0024) XOR $10, $1, $7

PC(0x0028) SRL $11, $10, $9

PC(0x002C) AND $12, $3, $6

PC(0x0030) OR $13, $10, $12

PC(0x0034) SRA $14, $5, $2

PC(0x0038) SW $10, 0($2)

PC(0x003C) LW $15, 0($2)

PC(0x0040) ANDI $16, $13, 7

PC(0x0044) ORI $17, $13, 4

PC(0x0048) SRLI $18, $4, 3

PC(0x004C) BEQ $6, $8, 6

PC(0x0050)

PC(0x0054)

PC(0x0058)

PC(0x005C)

PC(0x0060)

PC(0x0064)

PC(0x0068)

PC(0x006C)

PC(0x0070)

PC(0x0074)

PC(0x0078)

PC(0x007C) XORI $19, $11, 12

PC(0x0080) SLLI $20, $19, 1

PC(0x0084) SRAI $21, $5, 2

PC(0x0088) SLTI $22, $20, 28

Και παρατηρήθηκε ότι στην εντολή που εκτελείται όταν το PC είναι PC(0x004C) η διακλάδωση δεν περνάει όλες τις κενές γραμμές. Οπότε για να μην εμφανίζεται στην τελική κυματομορφή ένα κενό για ένα διάστημα αλλάχθηκαν οι γραμμές 77-80 του αρχείου rom_bytes.data με τις αντίστοιχες έτσι ώστε αντί να πηγαίνει 6 γραμμές παρακάτω να πηγαίνει 48 γραμμές παρακάτω:

Παλιές γραμμές κώδικα 77-80

00000000

10000011

00000011

01100011

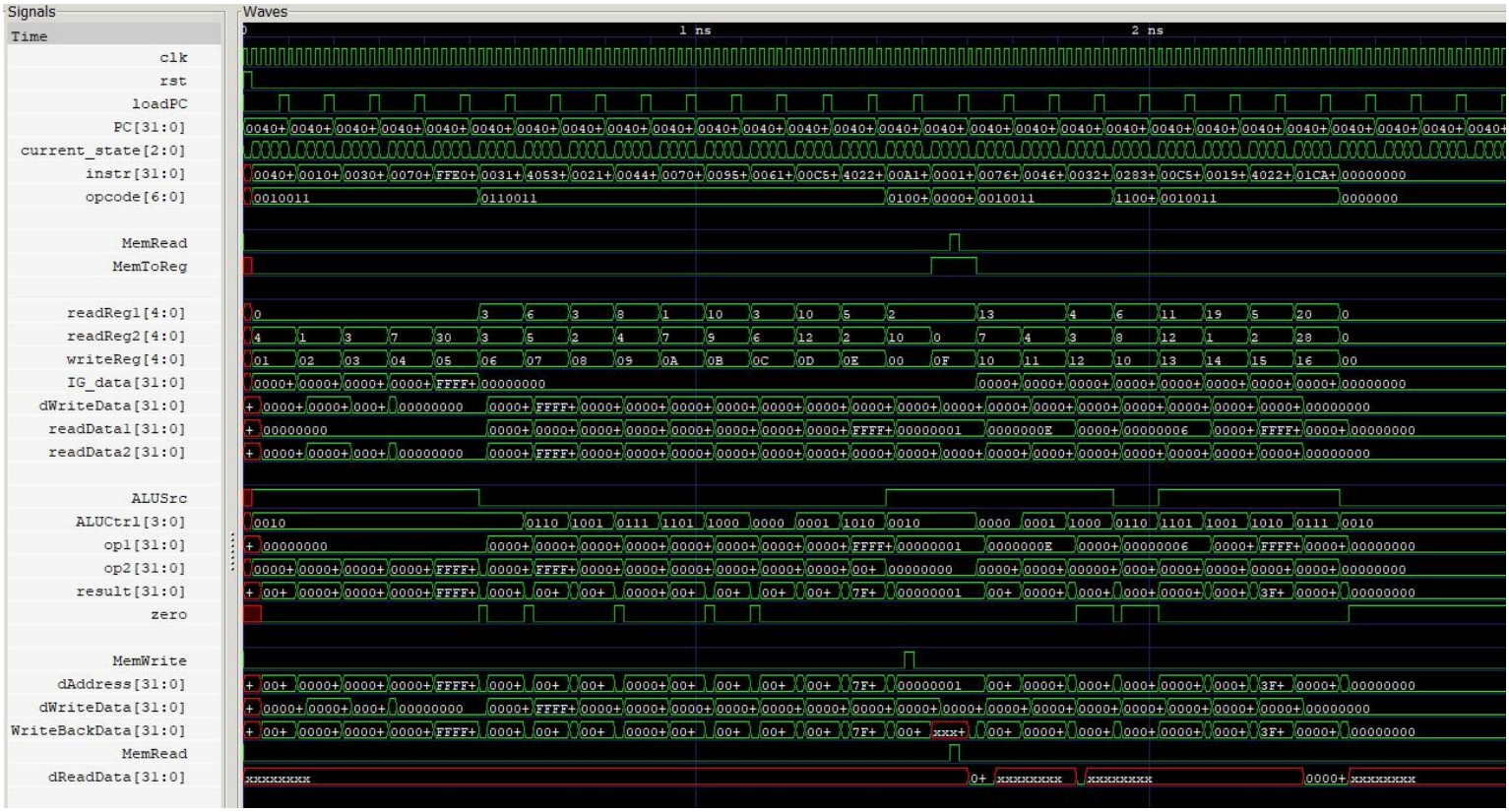
Καινούργιες γραμμές κώδικα 77-80

00000010

10000011

00001000

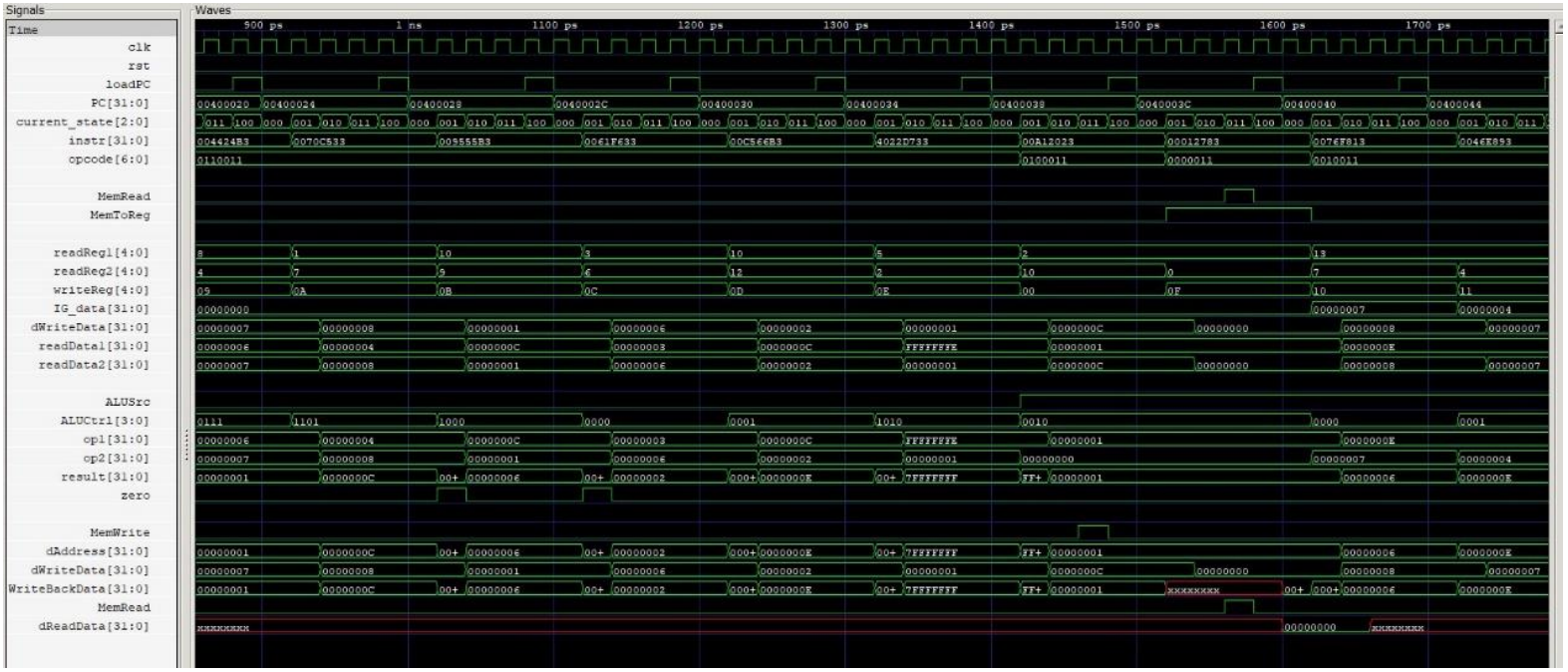
01100011



Εικόνα 7: Κυματομορφές προσομοίωσης Άσκησης 5

Χώρισα την κυματομορφές σε τρία στάδια για λόγους ευκρίνειας:

Εικόνα 8: Κυματομορφές 0-900ps



Εικόνα 9: Κυματομορφές 900-1700ps

Εικόνα 10: Κυματομορφές 1700-2500ps