ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ HW I

Εργασία στη Verilog

Μαυρογεωργιάδη Χριστίνα 10439

Δεκέμβριος 2023

Περιεχόμενα

1 Άσκηση 1 **1**

2 Άσκηση 2 **2**

3 Άσκηση 3 **4**

4 Άσκηση 4 **5**

5 Άσκηση 5 **10**

1 Άσκηση 1

Στην άσκηση αυτή, σχεδιάζεται μια Αριθμητική Λογική Μονάδα (Arithmetic Logic Unit - ALU), η οποία υλοποιεί τις ακόλουθες πράξεις: προσημασμένη πρόσθεση, προσημασμένη αφαίρεση, λογικό AND, λογικό OR, λογικό XOR, σύγκριση Μικρότερο από΄ και τρεις διαφορετικές πράξεις ολίσθησης. Για το σκοπό αυτό, δημιουργήθηκε ένα αρχείο Verilog με όνομα alu.v. Παρακάτω, εξηγείται η διαδικασία που ακολουθήθηκε για την σχεδίαση του δεδομένου module.

Ορίζουμε αρχικά τις εισόδους και τις εξόδους της ALU.

Οι έξοδοι είναι:
zero : μεταβλητή τύπου reg πλάτους 1-bit, η οποία τίθεται ίση με 1 αν το αποτέλεσμα της πράξης της ALU ισούται με 0, ενώ τίθεται ίση με 0 σε οποιαδήποτε άλλη περίπτωση
result : μεταβλητή τύπου reg πλάτους 32-bit. Στη μεταβλητή αυτή αποθηκεύεται το αποτέλεσμα της πράξης που εκτελεί η ALU.

Οι είσοδοι είναι:
op1 : σήμα τύπου wire πλάτους 32-bit. Το σήμα αυτό αποτελεί τον πρώτο τελεστή εισόδου της ALU. op2 : σήμα τύπου wire πλάτους 32-bit. Το σήμα αυτό αποτελεί τον δεύτερο τελεστή εισόδου της ALU. alu_op : σήμα τύπου wire πλάτους 4-bit. Το σήμα αυτό αποτελεί την είσοδο ελέγχου της ALU, η οποία καθορίζει ποια λειτουργία θα εκτελεστεί.

Σχόλια :

- Τα σήματα εξόδου δηλώνονται ως τύπου reg, καθώς βρίσκονται στο αριστερό μέρος αναθέσεων μέσα σε procedural block. Τα σήματα εισόδου μπορούν να δηλωθούν ως τύπου wire.
- Δηλώνουμε 9 παραμέτρους, μία για κάθε λειτουργία της ALU. Οι παράμετροι αναπαριστώνται από μια δυαδική λέξη πλάτους 4-bit. Με αυτόν τον τρόπο μπορούμε να μεταβάλλουμε τις σταθερές που αντιστοιχούν σε κάθε λειτουργία κατ΄ επιλογή, για παράδειγμα κατά το instantiation του module, χωρίς να χρειαστεί να παραλλάξουμε τον κώδικα.
- Το module υλοποιεί ένα κύκλωμα συνδυαστικής λογικής. Έχει επιλεχθεί να χρησιμοποιηθεί procedural Verilog. Για το σκοπό αυτό, δηλώνεται ένα always block και στη λίστα ευαισθησίας του συμπεριλαμβάνουμε όλα τα σήματα εισόδου του κυκλώματος, μέσω του συμβόλου *. Για την ανάθεση των σημάτων εξόδου χρησιμοποιούνται blocking εντολές.

- Η επιλογή της λειτουργίας που θα εκτελεστεί από το κύκλωμα της ALU γίνεται μέσω ενός πολυπλέκτη με σήμα ελέγχου το σήμα εισόδου alu_op. Ο πολυπλέκτης υλοποιείται από μία εντολή case. Ανάλογα με την τιμή του σήματος ελέγχου, αν αυτό αντιστοιχεί στην τιμή κάποιας από τις γνωστές παραμέτρους που έχουν δηλωθεί, εκτελείται η επιθυμητή λειτουργία. Διαφορετικά, στο default case επιλέγεται η έξοδος να οδηγηθεί στη τιμή 0.

- Οι λειτουργίες της πρόσθεσης και της αφαίρεσης πραγματοποιούνται πάνω σε προσημασμένους αριθμούς. Αυτό σημαίνει ότι τα σήματα εισόδου op1 και op2 πρέπει να δωθούν ως προσημασμένοι αριθμοί σε μορφή συμπληρώματος ως προς 2.

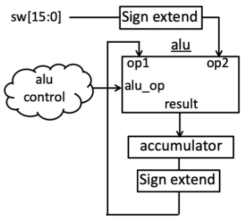
- Για να εκτελεστεί σωστά η λειτουργία ¨μικρότερο από' τα σήματα op1 και op2 πρέπει να μετατραπούν σε σήματα τύπου signed μέσω του system function $signed(). Το αποτέλεσμα της πράξης είναι η λογική τιμή 0 ή 1 ανάλογα με το αποτέλεσμα της σύγκρισης.

- Αντίστοιχα, για να εκτελεστεί σωστά η λειτουργία της αριθμητικής ολίσθησης δεξιά, το σήμα op1 πρέπει να μετατραπεί σε σήμα τύπου signed. Το αποτέλεσμα της πράξης αποθηκεύεται στη μεταβλητή result, η οποία είναι τύπου unsigned.

- Τέλος, χρησιμοποιείται μια εντολή if, η οποία είναι υπεύθυνη για την ανάθεση του σήματος εξόδου zero. Η ανάθεση που θα εκτελεστεί ελέγχεται από την τιμή του σήματος result.

2 Άσκηση 2

Στην άσκηση αυτή, σχεδιάζουμε ένα κύκλωμα αριθμομηχανής, το οποίο φαίνεται στο παρακάτω σχήμα. Συγκεκριμενα, το κύκλωμα αποτελείται από δύο βασικά στοιχεία: το κύκλωμα της ALU που υλοποιήθηκε παραπάνω και έναν συσσωρευτη (accumulator) μεγέθους 16-bit, ο οποίος αποθηκεύει την τρέχουσα τιμή της εξόδου της ALU.



Σχήμα 1: Διάγραμμα ροής της αριθμομηχανής

Το κύκλωμα υλοποιείται σε ένα αρχείο Verilog με όνομα calc.v. Στη συνέχεια, ακολουθεί περιγραφή της διαδικασίας σχεδιασμού του module.

Ορίζουμε αρχικά τις εισόδους και τις εξόδους της αριθμομηχανής.

Η έξοδος είναι:
led : μεταβλητή τύπου reg πλάτους 16-bit, η οποία δείχνει το περιεχόμενο του συσσωρευτή.

Οι είσοδοι είναι:
clk : σήμα τύπου wire πλάτους 1-bit. Η είσοδος αυτή είναι το ρολόι του συστήματος.
sw : σήμα τύπου wire πλάτους 16-bit. Το σήμα αυτό αναπαριστά τους διακόπτες της αριθμομηχανής για την εισαγωγή των δεδομένων εισόδου που δίνονται στον δεύτερο τελεστή της ALU.
btnu, btnd, btnc, btnl, btnr : σήμα τύπου wire πλάτους 1-bit. Αναπαριστούν πλήκτρα της αριθμομηχανής, τα οποία είναι υπεύθυνα για τη επιλογή της λειτουργίας της ALU και για τις λειτουργίες του συσσωρευτή.

Δηλώνουμε επίσης τα εξής εσωτερικά σήματα:
accumulator : μεταβλητή τύπου reg πλάτους 16-bit. Η μεταβλητή χρησιμοποιείται για την υλοποίηση του καταχωρητή αποθήκευσης της εξόδου της τεξτλατινΑΛΥ.
result, alu_op_control : σήματα τύπου wire πλάτους 32-bit και 4-bit αντίστοιχα. Χρησιμοποιούνται για την διασύνδεση των θυρών των modules.

Η μεταβλητή εξόδου led καθώς και η μεταβλητή accumulator δηλώνονται ως τύπου reg, καθώς βρίσκονται στο αριστερό τμήμα αναθέσεων μέσα σε procedural block. Τα σήματα εισόδου μπορούν να δηλωθούν ως τύπου wire.

Module instantiation και port mapping :

- Κάνουμε instantiate δύο modules: alu και decoder και συνδέουμε κατάλληλα τις θύρες τους. Το decoder module είναι υπεύθυνο για την παραγωγή του κατάλληλου σήματος ελέγχου της ALU, λαμβάνοντας υπ' όψη το πάτημα των πλήκτρων btnc, btnl, btnr και η λειτουργία του θα περιγραφεί αργότερα σε αυτή την ενότητα.
- Στις θύρες εισόδου op1 και op2 της ALU συνδέουμε την έξοδο του accumulator και το σήμα εισόδου sw αφού εκτελέσουμε επέκταση προσήμου. Η επέκταση προσήμου γίνεται επαναλαμβάνοντας το bit 15 των δυαδικών λέξεων 16 φορές μπροστά από τη λέξη μέσω concatenation. Ενδεικτικά:
{{16 {accumulator[15]}}, accumulator}.
- Η θύρα εξόδου zero του module της ALU δεν χρησιμοποιείται από την αριθμομηχανή, για αυτό παραλείπουμε την σύνδεση αυτής της θύρας του module με κάποιο σήμα.
- Η θύρα εξόδου result συνδέεται μέσω της εσωτερικής διασύνδεσης result στην είσοδο του καταχωρητή accumulator.
- Η θύρα εισόδου alu_op της ALU συνδέεται μέσω της εσωτερικής διασύνδεσης alu_op_control στην θύρα εξόδου alu_op του decoder module.

Λειτουργία συσσωρευτή (accumulator) :

Θα επεξηγήσουμε τώρα την λειτουργία του συσσωρευτή. Ο συσσωρευτής υλοποιείται από ένα always block και αποτελεί ακολουθιακό στοιχείο του κυκλώματος. Για το λόγο αυτό, οι αναθέσεις μέσα στο block είναι non-blocking. Στη λίστα ευαισθησίας δηλώνονται τα σήματα clk και btnd συνοδευόμενα από τη λέξη-κλειδί posedge, η οποία περιορίζει την εκτέλεση του block μόνο σε μεταβάσεις ανόδου των σημάτων της λίστας. Το σήμα btnd, εφόσον συμπεριλαμβάνεται στη λίστα, είναι ασύχρονο με το ρολόι, γεγονός το οποίο αποτέλεσε ελεύθερη σχεδιαστική επιλογή. Αντιθέτως, το σήμα btnu είναι σύγχρονο με το ρολόι καθώς δεν περιλαμβάνεται στη λίστα ευαισθήσίας και έχει προτεραιότητα σε σχέση με το σήμα btnd. Αναλυτικότερα, σε κάθε θετική μετάβαση του ρολογιού, αν το πλήκτρο btnu είναι ενεργοποιημένο, εκτελείται reset, δηλαδή το περιέχομενο του καταχωρητή ανανεώνεται σε 0. Διαφορετικά, αν το πλήκτρο btnd είναι ενεργοποιημένο, το περιεχόμενο του καταχωρητή ανανεώνεται στην τιμή των 16 least significant bits του σήματος result. Στη σχεδίαση αυτή δεν έχει ληφθεί υπ' όψη η περίπτωση τα πλήκτρα btnd και btnu να πατηθούν ταυτόχρονα. Τότε το reset του καταχωρητή θα γίνει σύγχρονα με την μετάβαση ανόδου του btnd και όχι του ρολογιού. Ωστόσο, θεωρήθηκε πως μια τέτοια ενέργεια θα ήταν άσκοπη, για αυτό και δεν προχώρησα σε πολυπλοκότερη σχεδίαση του καταχωρητή για την αντιμετώπιση του προβλήματος. Τέλος, μέσω ενός διαφορετικού always block, σε κάθε αλλαγή του περιεχομένου του καταχωρητή, η τιμή του ανατίθεται στην μεταβλητή εξόδου led μέσω μιας blocking ανάθεσης.

Υλοποίηση decoder module :

Καταληκτικά, θα επεξηγήσουμε τη λειτουργία του decoder module, το οποίο υλοποιείται σε ένα νεό αρχείο Verilog με όνομα decoder.v. Όπως έχει προαναφερθεί, σκοπός του module είναι να αποκωδικοποιήσει το πάτημα των πλήκτρων btnu, btnl, btnr της αριθμομηχανής, τα οποία συνδέονται στις θύρες εισόδου του, και να παράξει το κατάλληλο σήμα alu_op που θα οδηγηθεί στην είσοδο της ALU. Καθώς τα 3 πλήκτρα μπορούν να παράξουν μόνο 8 διαφορετικούς συνδυασμούς, σε αντίθεση με την ALU η οποία είναι ικανή να εκτελέσει 9 διαφορετικές λειτουργίες, η λειτουργία της αριθμητικής ολίσθησης δεξιά παραλείπεται από την αριθμομηχανή. Παρακάτω, παρουσιάζεται ο τρόπος σχεδίασης του module.

Ορίζουμε αρχικά τις εισόδους και τις εξόδους του module.

Η έξοδος είναι:
alu_op : σήμα τύπου wire πλάτους 1-bit.

Οι είσοδοι είναι:
btnc, btnl, btnr : σήματα τύπου wire πλάτους 1-bit.

Δεδομένου ότι η υλοποίηση του module ζητείται να γίνει σε structural Verilog, τα σήματα εισόδου καθώς και το σήμα εξόδου, τα οποία δίνονται στις θύρες εισόδου και εξόδου αντίστοιχα των primitives, πρέπει να είναι τύπου wire. Κάθε bit της εξόδου παράγεται από ένα συνδυαστικό κύκλωμα με είσοδους τα προαναφερθέντα πλήκτρα. Η

δήλωση των εσωτερικών διασυνδέσεων του κάθε υποκυκλώματος γίνεται έμμεσα. Για την υλοποίηση των συνδυαστικών κυκλωμάτων, χρησιμοποιούνται λογικές πύλες που ενυπάρχουν ως gate-level primitives στη Verilog.

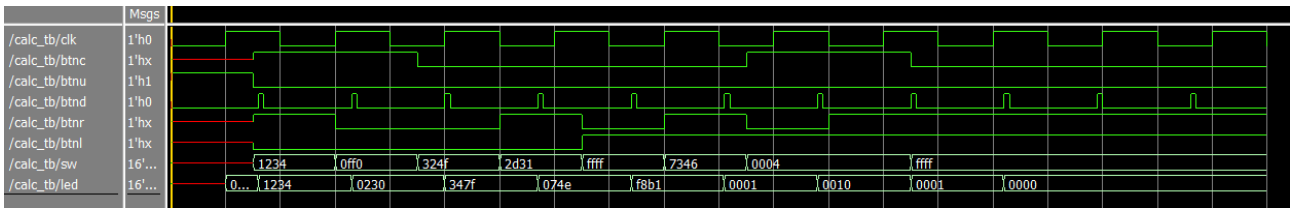
Testbench :

Για την επαλήθευση της ορθής λειτουργίας της αριθμομηχανής καθώς και της ALU, δημιουργούμε ένα νέο αρχείο Verilog με όνομα calc_tb.v. Παρακάτω, θα εξηγήσουμε την υλοποίηση του testbench αρχείου.

Δηλώνουμε πρώτα τα σήματα εισόδου και εξόδου του calc module ως εσωτερικά σήματα. Τα σήματα εισόδου δηλώνονται ως τύπου reg, ενώ το σήμα εξόδου δηλώνεται ως τύπου wire. Στη συνέχεια, πραγματοποιείται το instantiation του calc module και γίνεται η αντιστοίχιση των θυρών του με τα εσωτερικά σήματα που δηλώθηκαν προηγουμένως. Τέλος, πρέπει να φροντίσουμε για την ανάθεση τιμών στα σήματα εισόδου κατά την διάρκεια της προσομοίωσης.

- Στην αρχή του testbench αρχείου, θα πρέπει να δηλωθεί το timescale της προσομοίωσης.
- Δηλώνεται ένα initial block, μέσα στο οποίο ανατίθεται αρχικά στο σήμα btnu η τιμή 1. Το σήμα αυτό είναι υπεύθυνο για το reset και είναι σύγχρονο με το ρολόι. Παραμένει ενεργό για 15 μονάδες χρόνου μέχρι να συμπέσει με μια θετική ακμή του ρολογιού και έπειτα λαμβάνει την τιμή 0 για το υπόλοιπο της προσομοίωσης.
- Για το σήμα ρολογιού δηλώνονται ένα initial block, μέσα στο οποίο του ανατίθεται μια αρχική τιμή ίση με 0, και ένα always block το οποίο εκτελείται καθόλη την προσομοίωση. Η τιμή του ρολογιού εναλλάσεται από 0 σε 1 και αντίστροφα κάθε 10 μονάδες χρόνου, οπότε δημιουργούνται παλμοί με χρονική περίοδο 10.
- Για το σήμα btnd δηλώνονται ένα initial block, μέσα στο οποίο του ανατίθεται μια αρχική τιμή ίση με 0, και ένα always block το οποίο εκτελείται καθόλη την προσομοίωση. Το σήμα αυτό είναι υπεύθυνο για την ενημέρωση του περιεχομένου του συσσωρευτή. Η ενημέρωση αυτή γίνεται ασύγχρονα με το ρολόι και πρέπει να πραγματοποιείται μετά από κάθε μεταβολή της εισόδου sw, η οποία προκαλεί αλλαγή στο αποτέλεσμα της πράξης της ALU. Για το σκοπό αυτό, κάθε 16 μονάδες χρόνου παράγεται ένας θετικός παλμός διάρκειας μίας μονάδας χρόνου, ώστε να επιτρέψει στο αποτέλεσμα της πράξης να αποθηκευτεί στον καταχωρητή. Η εναλλαγή των τιμών εισόδου γίνεται κάθε 15 μονάδες χρόνου.
- Οι τιμές των πλήκτρων btnc, btnl, btnr καθώς και η τιμή των δεδομένων εισόδου sw ανανεώνονται μέσα σε ένα initial block κάθε 15 μονάδες χρόνου στις τιμές που δίνονται στην εκφώνηση της εργασίας. Οι αρχικές τιμές τους είναι απροσδιόριστες.

Μετά την εκτέλεση της προσομοίωσης, παράχθηκαν οι ακόλουθες κυματομορφές.



Σχήμα 2: Αποτέλεσμα προσομοίωσης

Τα περιεχόμενα του συσσωρευτή, όπως φαίνονται στο παράθυρο της προσομοίωσης στο Σχήμα 5, συνάδουν με τα αναμενόμενα που αναφέρονται στην εκφώνηση της εργασίας, επαληθεύοντας τη σωστή λειτουργία όλων των επιμέρους στοιχείων του κυκλώματος.

3 Άσκηση 3

Στην άσκηση αυτή, σχεδιάζουμε ένα αρχείο καταχωρητών το οποίο θα χρησιμοποιηθεί αργότερα ως τμήμα ενός επεξεργαστή. Για το σκοπό αυτό, δημιουργήθηκε ένα νέο αρχείο Verilog με όνομα regfile.v. Παρακάτω, θα σχολιαστεί αναλυτικότερα η διαδικασία σχεδίασης.

Ορίζουμε αρχικά τις εισόδους και τις εξόδους του module.

Οι έξοδοι είναι:
readData1, readData2 : μεταβλητές τύπου reg πλάτους 32-bit. Αποτελούν τις θύρες εξόδου των δεδομένων ανάγνωσης του αρχείου καταχωρητών.

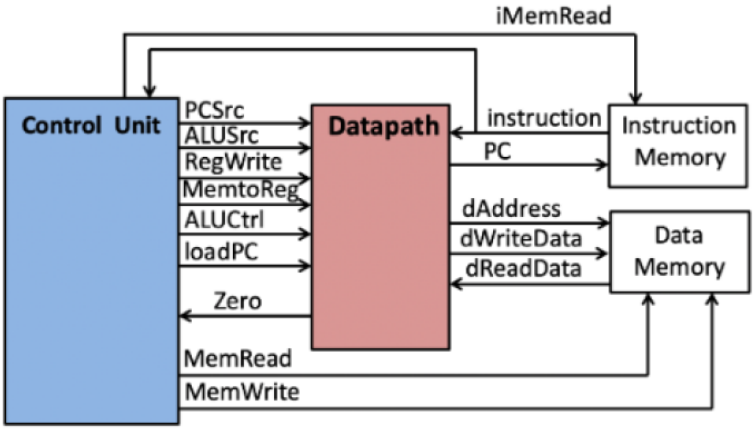
Οι είσοδοι είναι:
writeData : σήμα τύπου wire πλάτους 32-bit. Αποτελεί τη θύρα εισόδου των δεδομένων εγγραφής του αρχείου καταχωρητών.
writeReg, readReg1, readReg2 : σήματα τύπου wire πλάτους 5-bit. Αποτελούν τις θύρες εισόδου των διευθύνσεων για τη θύρα εγγραφής και τις θύρες ανάγνωσης αντίστοιχα του αρχείου καταχωρητών.
clk : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα ρολογιού του αρχείου καταχωρητών.
write : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα ελέγχου που υποδεικνύει εγγραφή δεδομένων στο αρχείο καταχωρητών.

Σχόλια :

- Οι μεταβλητές εξόδου readData1, readData2 δηλώνονται ως τύπου reg, καθώς βρίσκονται στο αριστερό τμήμα αναθέσεων μέσα σε procedural block. Τα σήματα εισόδου μπορούν να δηλωθούν ως τύπου wire.
- Υλοποιούμε το αρχείο καταχωρητών 32 επί 32 bit δηλώνοντας έναν πίνακα (array) 32 στοιχείων, ονόματος Mem, όπου ο τύπος του κάθε στοιχείου είναι ένα διάνυσμα (vector) τύπου reg και πλάτους 32 bits. Με άλλα λόγια, το αρχείο καταχωρητών μας συνίσταται από 32 διευθύνσεις δυαδικών λέξεων πλάτους 32-bit.
- Οι θύρες εισόδου των διευθύνσεων ανάγνωσης και εγγραφής έχουν πλάτος 5 bits, οπότε μπορούν να παράξουν $2^5 = 32$ δυαδικούς αριθμούς που αναπαριστούν τις 32 δυνατές διευθύνσεις του αρχείου.
- Αρχικοποιούμε τους 32 καταχωρητές του αρχείου στην τιμή 0 μέσω ενός initial block και μιας εντολής for. Η εντολή for επαναλαμβάνεται 32 φορές, όσοι είναι και οι καταχωρητές, και αναθέτει στο κάθε διάνυσμα την τιμή 0.
- Η ανάγνωση και εγγραφή των δεδομένων γίνεται μέσω ενός always block. Στη λίστα ευαισθησίας του συμπεριλαμβάνεται το σήμα ρολογιού, συνοδευόμενο από τη λέξη-κλειδί posedge, η οποία περιορίζει την εκτέλεση του block μόνο στις μεταβάσεις ανόδου του σήματος.
- Σε κάθε θετική ακμή του ρολογιού, τα δεδομένα των καταχωρητών που είναι αποθηκευμένα στις διευθύνσεις readReg1, readReg2 ανατίθενται στις εξόδους readData1, readData2 αντίστοιχα.
- Αν το σήμα write είναι ενεργό, τότε τα δεδομένα εγγραφής writeData εγγράφονται στον καταχωρητή με διεύθυνση writeReg. Το σήμα write δεν περιλαμβάνεται στη λίστα ευαισθησίας του always block και είναι σύγχρονο με το σήμα ρολογιού.
- Παρόλο που το κύκλωμα αποτελεί ένα ακουθιακό στοιχείο ευαίσθητο στη θετική ακμή του ρολογιού, έχει επιλεχθεί να χρησιμοποιηθούν blocking αναθέσεις, με σκοπό την αποφυγή read/write conflicts. Αυτό σημαίνει ότι οι εντολές εκτελούνται η μία μετά την άλλη και όχι ταυτόχρονα μεταξύ τους στην ακμή του ρολογιού. Με αυτόν τον τρόπο, στην περίπτωση που το σήμα write είναι ενεργό και η διεύθυνση εγγραφής ταυτίζεται με κάποια από τις διευθύνσεις ανάγνωσης, τα δεδομένα ανάγνωσης θα είναι τα ήδη αποθηκευμένα δεδομένα στο αρχείο καταχωρητών και όχι τα δεδομένα που εγγράφονται στον τρέχοντα κύκλο ρολογιού. Δίνουμε δηλαδή προτεραιότητα στην ανάγνωση των δεδομένων και όχι στην εγγραφή τους, με σκοπό την αποφυγή απώλειας των δεδομένων που είναι αποθηκευμένα στους καταχωρητές.

4 Άσκηση 4

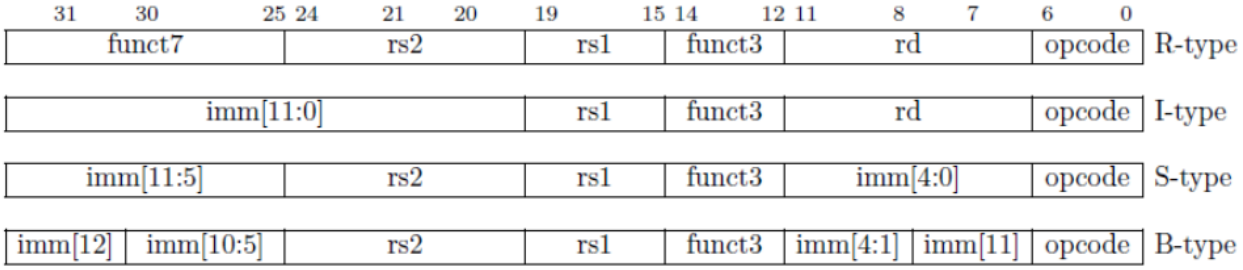
Στην άσκηση αυτή, σχεδιάζουμε τη διαδρομή δεδομένων (datapath) ενός επεξεργαστή. Ένα διάγραμμα υψηλού επιπέδου της δομής του επεξεργαστή φαίνεται παρακάτω.



Σχήμα 3: Διάγραμμα υψηλού επιπέδου του επεξεργαστή

Η διαδρομή δεδομένων θα υποστηρίζει τις ακόλουθες λειτουργίες:
Register-to-Register : ADD, SUB, AND, OR, XOR, SLT, SLL, SRL, SRA
ALU Immmediate : ADDI, ANDI, ORI, XORI, SLTI, SLLI, SRLI, SRAI
Μνήμη : LW, SW
Διακλάδωση : BEQ

Το κύκλωμα που σχεδιάζουμε, θα αποκωδικοποιεί εντολές που περιλαμβάνονται στο instruction set του επεξεργαστή RISC-V. Σε αυτό το σημείο, είναι σημαντικό να κατανοήσουμε τον τρόπο με τον οποίο είναι σχεδιασμένο το instruction set του συγκεκριμένου επεξεργαστή καθώς και το πώς να ερμηνεύουμε τα ιδιαίτερα πεδία (fields) της δυαδικής λέξης πλάτους 32-bit που αναπαριστά την κάθε εντολή.
Το instruction set του RISC-V περιέχει τους εξής τύπους εντολών:
R-type, I-type, S-type, B-type, U-type, J-type
Εμείς θα επικεντρωθούμε στους πρώτους 4 τύπους εντολών.
Όλες οι εντολές έχουν ένα ***opcode field***, το οποίο καταλαμβάνει τα bits [6:0] σε κάθε τύπο εντολής. Με βάση την τιμή του ***opcode***, είμαστε σε θέση να ερμηνεύσουμε σε ποιον τύπο ανήκει μια δεδομένη εντολή και στη συνέχεια να προσδιορίσουμε πώς είναι διαμορφωμένα τα υπόλοιπα πεδία (fields) της εντολής.



Σχήμα 4: RISC-V Instruction Encoding

■ R-type Instructions

Οι εντολές αυτού του τύπου (Register-to-Register) χρησιμοποιούνται για λειτουργίες που αναθέτουν στον **destination register *rd*** την τιμή του αποτελέσματος μιας αριθμητικής, λογικής, ή πράξης ολίσθησης, η οποία εφαρμόζεται στο περιεχόμενο των **source registers *rs1, rs2***. Το πεδίο ***funct3*** χρησιμοποιείται για να αναγνωρίσουμε την εκτελούμενη πράξη, ενώ το bit 30 του πεδίου ***funct7*** χρησιμοποιείται για να διαχωρίσουμε την πρόσθεση από την αφαίρεση και την αριθμητική από την λογική ολίσθηση. Όλες οι εντολές αυτού του τύπου που θα υλοποιηθούν στα πλαίσια της δεδομένης άσκησης έχουν τιμή του ***opcode* field** · ′0110011′.

■ I-type Instructions

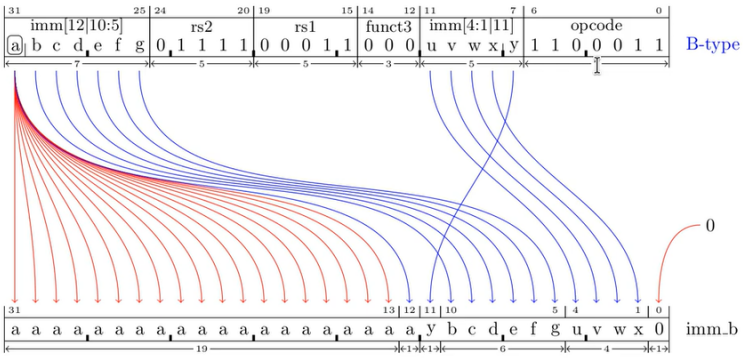
Στις εντολές αυτού του τύπου ανήκουν οι εντολές ALU Immediate καθώς και η εντολή μνήμης LW. Χρησιμοποιούνται για να κωδικοποιήσουν εντολές στις οποίες εμπλέκονται ένας **source register *rs1*** και ένας τελεσταίος τύπου signed immediate μεγέθους 12-bit. Το αποτέλεσμα της λειτουργίας που εκτελείται ανάμεσα σε αυτά τα δύο στοιχεία αποθηκεύεται στον **destination register *rd***. Οι εντολές τύπου ALU Immediate έχουν τιμή του ***opcode* field** : ′0010011′, ενώ η εντολή LW έχει τιμή : ′0000011′. Η συγκεκριμένη λειτουργία καθορίζεται από το πεδίο ***funct3***. Πριν εφαρμοστεί η πράξη, ο τελεσταίος immediate εξάγεται από τα τελευταία 12 bit της λέξης και μετατρέπεται σε έναν δυαδικό αριθμό μεγέθους 32-bit με επέκταση προσήμου.

- **S-type Instructions**

Οι εντολές αυτού του τύπου χρησιμοποιούνται για την αποθήκευση δεδομένων στη μνήμη δεδομένων. Στα πλαίσια της άσκησης, ασχολούμαστε μόνο με την εντολή SW. Εμπλέκουν δύο **source registers *rs1, rs2*** και έναν τελεσταίο τύπου signed immediate μεγέθους 12-bit. Ο τελεσταίος immediate εξάγεται από bits ως εξής: τα 5 least significant bits εξάγονται από τα bits [11:7] της λέξης, ενώ τα υπόλοιπα 7 bits εξάγονται από τα bits [31:25] της λέξης. Στη συνέχεια, μετατρέπεται σε έναν δυαδικό αριθμό μεγέθους 32-bit με επέκταση προσήμου. Όλες οι εντολές S-type έχουν τιμή του ***opcode* field** : ′0100011′. Η συγκεκριμένη λειτουργία δηλώνεται από την τιμή του πεδίου ***funct3***.

- **B-type Instructions**

Οι εντολές αυτού του τύπου χρησιμοποιούνται για την διακλάδωση του προγράμματος σε μια νέα θέση της μνήμης εντολών ανάλογα με την ικανοποίηση μιας συνθήκης. Στα πλαίσια αυτής της άσκησης, ασχολούμαστε μόνο με την εντολή BEQ, η οποία προχωρά σε διακλάδωση αν οι δύο τελεστές της είναι ίσοι. Οι εντολές B-type έχουν την ίδια μορφή με τις εντολές S-type, δηλαδή εμπλέκουν δύο **source registers *rs1, rs2*** και έναν τελεσταίο τύπου signed immediate μεγέθους 12-bit. Ωστόσο, διαφέρουν ως προς τον τρόπο με τον οποίο εξάγεται ο τελεσταίος immediate, ο οποίος είναι περισσότερο πολύπλοκος και δε θα εξηγηθεί αναλυτικά εδώ, αλλά φαίνεται σε ένα παράδειγμα στο παρακάτω σχήμα. Όλες οι εντολές B-type έχουν τιμή του ***opcode* field** : ′1100011′. Η συγκεκριμένη λειτουργία δηλώνεται από την τιμή του πεδίου ***funct3***.



Σχήμα 5: Immediate operand extraction from a B-type Instruction

Οι λεπτομέρειες της κωδικοποίησης των παραπάνω εντολών (***funct3* , *funct7* fields**) είναι διαθέσιμες στο εγχειρίδιο που δίνεται.

Προχωρούμε τώρα στην επεξήγηση του σχεδιασμού της διαδρομής δεδομένων (datapath). Για την υλοποίηση, δημιουργήθηκε ένα νέο αρχείο Verilog με όνομα datapath.v. Ακολούθως σχολιάζονται οι λεπτομέρειες σχεδιασμού.

Ορίζουμε αρχικά μια παράμετρο INITIAL_PC, η οποία καθορίζει την αρχική τιμή του **program counter**, δηλαδή την διεύθυνση μνήμης από την οποία θα ξεκινήσει η ανάγνωση των λέξεων-εντολών από τη μνήμη εντολών. Στη συνέχεια. ορίζουμε τις εισόδους και τις εξόδους του κυκλώματος.

Οι έξοδοι είναι:
dAddress : μεταβλητή τύπου reg πλάτους 32-bit. Αποτελεί την διεύθυνση της μνήμης δεδομένων, στην οποία θα γίνει εγγραφή ή ανάγνωση δεδομένων από το **datapath**.
PC : μεταβλητή τύπου reg πλάτους 32-bit. Αποτελεί την διεύθυνση της μνήμης εντολών, από την οποία γίνεται η ανάγνωση της επόμενης εντολής που θα εκτελεστεί από τον επεξεργαστή.

dWriteData : μεταβλητή τύπου reg πλάτους 32-bit. Αποτελεί τα δεδομένα προς εγγραφή στη μνήμη δεδομένων.
dWriteBackData : μεταβλητή τύπου reg πλάτους 32-bit. Αποτελεί τα δεδομένα προς εγγραφή στο αρχείο καταχωρητών.
Zero : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που ενδεικνύει αν το αποτέλεσμα της ALU είναι μηδέν. Το σήμα αυτό χρησιμοποιείται κατά την εκτέλεση της εντολής διακλάδωσης BEQ.

Οι είσοδοι είναι:
instr : σήμα τύπου wire πλάτους 32-bit. Αποτελεί τη δυαδική λέξη που αναπαριστά την εντολή προς εκτέλεση, η οποία διαβάζεται από τη μνήμη εντολών.
dReadData : σήμα τύπου wire πλάτους 32-bit. Αποτελεί τα δεδομένα ανάγνωσης από τη μνήμη δεδομένων.
ALUCtrl : σήμα τύπου wire πλάτους 4-bit. Αποτελεί το σήμα ελέγχου που καθορίζει ποια λειτουργία θα εκτελεστεί από την ALU.
clk : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα ρολογιού του datapath.
rst : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα **Reset** του **datapath** και είναι σύγχρονο με το ρολόι.
PCSrc : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που ελέγχει την ανανέωση του **program counter** στη διεύθυνση μνήμης της επόμενης εντολής.
ALUSrc : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που ελέγχει την πηγή του δεύτερου τελεσταίου της μονάδας ALU.
RegWrite : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που επιτρέπει την εγγραφή δεδομένων στο αρχείο καταχωρητών.
MemToReg : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που επιλέγει την πηγή των δεδομένων εγγραφής στο αρχείο καταχωρητών.
loadPC : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που επιτρέπει την ανανέωση του **program counter** σε μια νέα τιμή.

Οι μεταβλητές εξόδου, εκτός του σήματος Zero δηλώνονται ως τύπου reg, καθώς βρίσκονται στο αριστερό τμήμα αναθέσεων μέσα σε procedural block. Τα σήματα εισόδου μπορούν να δηλωθούν ως τύπου wire.

Κατόπιν, θα επεξηγήσουμε τη λειτουργία των εξής επιμέρους στοιχείων του **datapath**, με τη σειρά που υλοποιούνται στον κώδικα Verilog :

Immediate Generation Unit

Οι εντολές **I-type, S-type, B-type** έχουν πεδία που αντιστοιχούν σε έναν τελεσταίο **immediate** μεγέθους 12-bit.

- Ορίζουμε μια εσωτερική μεταβλητή ονόματος imm τύπου reg μεγέθους 32-bit για την αναπαράσταση του **immediate operand**.
- Δηλώνουμε ένα always block. Στη λίστα ευαισθησίας του δηλώνεται το σύμβολο *, το οποίο σημαίνει η εκτέλεση του block είναι ευαίσθητη σε οποιαδήποτε μετάβαση σήματος που βρίσκεται στο δεξί τμήμα οποιασδήποτε εντολής ανάθεσης, δηλαδή στα πεδία της μεταβλητής instr.
- Δηλώνουμε μια εντολή casez, με είσοδο το ***opcode*** **field** της λέξης εντολής. Ανάλογα με τον τύπο της εντολής, το **immediate operand** εξάγεται από διαφορετικά πεδία της λέξης εντολής. Επιπλέον, στην περίπτωση εντολής **I-type**, πρέπει να διαχειριστούμε ειδικά τις εντολές ολίσθησης, στις οποίες το **immediate operand** έχει μέγεθος μόνο 5-bit. Οι εντολές αυτές διαχωρίζονται από τις υπόλοιπες μέσω μιας εντολής casez, με είσοδο το ***funct3*** **field** της λέξης εντολής.
- Αν η εντολή δεν ανήκει σε κάποιον από τους τρεις προαναφερθόντες τύπους (**default case**), τότε η τιμή του **immediate operand** είναι αδιάφορη.
- Σε όλες τις περιπτώσεις, το **immediate operand** υπόκειται σε επέκταση προσήμου με χρήση του τελεστή concatenation.

Register File Unit

Όλες οι εντολές χρησιμοποιούν το αρχείο καταχωρητών, είτε για διαβάσουν ή/και για να γράψουν δεδομένα στους 32 καταχωρητές του επεξεργαστή. Για το λόγο αυτό, συμπεριλαμβάνουμε στο **datapath** ένα instance του **regfile module** που υλοποιήθηκε στη Άσκηση 3.

- Δηλώνουμε τα εσωτερικά σήματα διασύνδεσης readData1, readData2, τύπου wire, τα οποία συνδέονται στις αντίστοιχες εξόδους του **regfile module** και writeData, τύπου reg, το οποίο συνδέεται στην αντίστοιχη είσοδο του **regfile module**.
- Κάνουμε την αντιστοίχιση θυρών (port mapping) του **regfile module**. Συνδέουμε τα εσωτερικά σήματα διασύνδεσης που δηλώθηκαν παραπάνω στις κατάλληλες θύρες εισόδου και εξόδου. Συνδέουμε τα σήματα εισόδου clk, RegWrite του **datapath module** στις θύρες του σήματος ρολογιού και του σήματος write του **regfile module**.
- Τέλος, πρέπει να συνδέσουμε στις θύρες εισόδου readReg1, readReg2, writeReg του **regfile module** τις διευθύνσεις των καταχωρητών ανάγνωσης και εγγραφής αντίστοιχα. Αυτές προέρχονται από την αποκωδικοποίηση της λέξης εντολής instr. Συγκεκριμένα, ανακτώνται από τα πεδία ***rs1, rs2, rd*** που βρίσκονται στα bits [19:15], [24:20], [11:7] αντίστοιχα της λέξης εντολής. Τα πεδία αυτά βρίσκονται στην ίδια θέση για όλους τους τύπους εντολών.

ALU Unit

Σημαντικό στοιχείο του **datapath** είναι η μονάδα της **ALU**, η οποία χρησιμοποιείται από όλες τις εντολές και πραγματοποιεί τις απαιραίτητες αριθμητικές και λογικές πράξεις για την εκτέλεσή τους. Για το λόγο αυτό. συμπεριλαμβάνουμε στο **datapath** ένα instance του **alu module** που υλοποιήθηκε στη Άσκηση 1.

- Δηλώνουμε τα εσωτερικά σήματα διασύνδεσης op2, τύπου reg, το οποίο συνδέεται στην αντίστοιχη θύρα εισόδου της **ALU**, και result, τύπου wire, το οποίο συνδέεται στην αντίστοιχη θύρα εξόδου της **ALU**.
- Κάνουμε την αντιστοίχιση θυρών (port mapping) του **alu module**. Συνδέουμε τα εσωτερικά σήματα διασύνδεσης που δηλώθηκαν παραπάνω στις κατάλληλες θύρες εισόδου και εξόδου. Συνδέουμε το σήμα εισόδου ALUCtrl του **datapath module** στη θύρα alu_op του **alu module**, καθώς και το σήμα εξόδου zero του **alu module** στην αντίστοιχη θύρα εξόδου Zero του **datapath module**. Τέλος, συνδέουμε το εσωτερικό σήμα readData1 που έχει δηλωθεί παραπάνω στη θύρα εισόδου op1 του **alu module**.
- Το εσωτερικό σήμα op2 συνδέεται στην έξοδο ενός πολυπλέκτη. Ο πολυπλέκτης έχει δύο πιθανές εισόδους : readData2 και imm, δηλαδή είτε την δεύτερη έξοδο του αρχείου καταχωρητών είτε το **immediate operand** που παράγεται από την αποκωδικοποίηση της εντολής. Η επιλογή του σήματος εισόδου που θα ανατεθεί στην έξοδο του πολυπλέκτη γίνεται από το σήμα ελέγχου ALUSrc, το οποίο αποτελεί είσοδο του **datapath module**. Ο πολυπλέκτης υλοποιείται από τον τελεστή ? : μέσα σε ένα always block, με λίστα ευαισθησίας *.

Write Back Unit

Οι εντολές **R-type, I-type** εμπλέκουν έναν **destination register *rd***. Ανάλογα με τον τύπο της εντολής, αποθηκεύεται στο αρχείο καταχωρητών είτε το αποτέλεσμα της πράξης της **ALU** που εκτελέστηκε στην περίπτωση των εντολών Register-to-Register και ALU Immediate, είτε τα δεδομένα ανάγνωσης από τη μνήμη δεδομένων στην περίπτωση της εντολής LW. Υλοποιούμε για αυτόν τον σκοπό έναν πολυπλέκτη με έξοδο το εσωτερικό σήμα writeData που έχει δηλωθεί νωρίτερα και συνδέεται στη θύρα εισόδου του **regfile module**. Τα σήματα εισόδου του πολυπλέκτη είναι τα σήματα dReadData (είσοδος του **datapath module**) και result (εσωτερικό σήμα που συνδέεται στη θύρα εξόδου του **alu module**). Το σήμα επιλογής της εισόδου που θα ανατεθεί στην έξοδο είναι το σήμα MemToReg, το οποίο αποτελεί σήμα εισόδου του **datapath module**. Ο πολυπλέκτης υλοποιείται όπως και προηγουμένως από τον τελεστή ? : μέσα σε ένα always block, με λίστα ευαισθησίας *. Τα δεδομένα εγγραφής writeData συνδέονται και στη θύρα εξόδου WriteBackData του **datapath module** με μία blocking ανάθεση.

Program Counter Unit

Καταληκτικά, θεμελιώδες στοιχείο της διαδρομής δεδομένων είναι ο **program counter**. Συγκεκριμένα, είναι ένας καταχωρητής στον οποίο περιέχεται η τιμή της διεύθυνσης μνήμης της τρέχουσας εντολής που εκτελείται από τον επεξεργαστή. Ο καταχωρητής υλοποιείται από ένα always block, με λίστα ευαισθησίας το σήμα ρολογιού clk, συνοδευόμενο από τη λεξη-κλειδί posedge, η οποία περιορίζει την εκτέλεση του block μόνο σε μεταβάσεις ανόδου του σήματος ρολογιού. Αν το σύγχρονο (δεν περιλαμβάνεται στη λίστα ευαισθησίας) σήμα rst είναι ενεργό, τότε η τιμή του καταχωρητή αρχικοποιείται στην τιμή της παραμέτρου INITIAL_PC. Αν το σύγχρονο σήμα loadPC είναι ενεργό, τότε ο **program counter** ανανεώνεται στην επόμενη θετική ακμή του ρολογιού. Μπορεί να λάβει μία από τις δύο ακόλουθες τιμές : PC + 4, όταν το πρόγραμμα προχωρά στην ακόλουθη εντολή στη μνήμη, ή PC +

branch_offset, όταν το πρόγραμμα πραγματοποιεί διάκλαδωση. Η επόμενη τιμή του **PC** ελέγχεται από έναν πολυπλέκτη, ο οποίος υλοποιείται με μία εντολή if μέσα στο always block. Η επιλογή της τιμής που θα φορτωθεί στην έξοδο του πολυπλέκτη γίνεται από το σήμα ελέγχου PCSrc. Τέλος, αναφέρουμε ότι η τιμή του branch_offset είναι η τιμή της μεταβλητής imm που υπολογίζεται κατάλληλα στο Immediate Generation Unit, αποκωδικοποιώντας τα πεδία της λέξης εντολής BEQ.

Data Memory Input

Στην περίπτωση των εντολών SW/LW είναι απαραίτητο να καθορίσουμε τη διεύθυνση της μνήμης δεδομένων στην οποία θα αποθηκευθούν ή θα εγγραφούν δεδομένα. Η διεύθυνση αυτή αναπαρίσταται από το σήμα dAddress. Στην τιμή του σήματος αυτού, αναθέτουμε την τιμή του αποτελέσματος της πράξης της ALU. Επιπλέον, στην περίπτωση της εντολής SW, είναι απαραίτητο να καθορίσουμε και την τιμή των δεδομένων που θα αποθηκευθούν στη μνήμη. Η τιμή αυτή αναπαρίσταται από το σήμα dWriteData. Στο σήμα αυτό ανατίθεται η δεύτερη έξοδος δεδομένων του αρχείου καταχωρητών.

5 Άσκηση 5

Στην άσκηση αυτή, σχεδιάζουμε έναν ελεγκτή πολλαπλών κύκλων που εκτελεί κάθε εντολή σε 5 κύκλους ρολογιού. Για το σκοπό αυτό, δημιουργούμε ένα νέο αρχείο Verilog με όνομα multicycle.v. Ακολούθως σχολιάζονται οι λεπτομέρειες σχεδιασμού.

Ορίζουμε αρχικά μια παράμετρο INITIAL_PC, η οποία καθορίζει την αρχική τιμή του **program counter**, δηλαδή την διεύθυνση μνήμης από την οποία θα ξεκινήσει η ανάγνωση των λέξεων-εντολών από τη μνήμη εντολών. Στη συνέχεια. ορίζουμε τις εισόδους και τις εξόδους του κυκλώματος.

Οι έξοδοι είναι:
dAddress : μεταβλητή τύπου wire πλάτους 32-bit. Αποτελεί την διεύθυνση της μνήμης δεδομένων, στην οποία θα γίνει εγγραφή ή ανάγνωση δεδομένων από το **datapath**.
dWriteData : μεταβλητή τύπου wire πλάτους 32-bit. Αποτελεί τα δεδομένα προς εγγραφή στη μνήμη δεδομένων.
PC : μεταβλητή τύπου wire πλάτους 32-bit. Αποτελεί την διεύθυνση της μνήμης εντολών, από την οποία γίνεται η ανάγνωση της επόμενης εντολής που θα εκτελεστεί από τον επεξεργαστή.
WriteBackData : μεταβλητή τύπου wire πλάτους 32-bit. Αποτελεί τα δεδομένα προς εγγραφή στο αρχείο καταχωρητών.
MemRead : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που ενδεικνύει αν το αποτέλεσμα της ALU είναι μηδέν. Το σήμα αυτό χρησιμοποιείται κατά την εκτέλεση της εντολής διακλάδωσης BEQ.
MemWrite : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα που ενδεικνύει αν το αποτέλεσμα της ALU είναι μηδέν. Το σήμα αυτό χρησιμοποιείται κατά την εκτέλεση της εντολής διακλάδωσης BEQ.

Οι είσοδοι είναι:
instr : σήμα τύπου wire πλάτους 32-bit. Αποτελεί τη δυαδική λέξη που αναπαριστά την εντολή προς εκτέλεση, η οποία διαβάζεται από τη μνήμη εντολών.
dReadData : σήμα τύπου wire πλάτους 32-bit. Αποτελεί τα δεδομένα ανάγνωσης από τη μνήμη δεδομένων.
rst : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα **Reset** του ελεγκτή και είναι σύγχρονο με το ρολόι.
clk : σήμα τύπου wire πλάτους 1-bit. Αποτελεί το σήμα ρολογιού του ελεγκτή.

Τα σήματα εξόδου dAddress, dWriteData, PC, WriteBackData δηλώνονται ως τύπου wire, διότι συνδέονται στις αντίστοιχες θύρες εξόδου του **datapath module**. Οι μεταβλητές εξόδου MemRead, MemWrite δηλώνονται ως τύπου reg, καθώς βρίσκονται στο αριστερό τμήμα αναθέσεων μέσα σε procedural block. Τα σήματα εισόδου μπορούν να δηλωθούν ως τύπου wire.

Κατόπιν, θα επεξηγήσουμε τη λειτουργία των εξής επιμέρους στοιχείων του ελεγκτή, με τη σειρά που εμφανίζονται στον κώδικα Verilog:

ALU Control Logic

Στη μονάδα αυτή, συμπεριλαμβάνεται η παραγωγή των σημάτων ALUCtrl, ALUSrc, τα οποία συνδέονται στις αντίστοιχες θύρες εισόδου του **datapath module** και ελέγχουν τη λειτουργία της μονάδας ALU.

- ALUCtrl

  Το σήμα ALUCtrl είναι ένα σήμα ελέγχου μεγέθους 4-bit, το οποίο υπαγορεύει τη λειτουργία που θα εκτελέσει η ALU. Η παραγωγή του σήματος αυτού απαιτεί αποκωδικοποίηση των πεδίων ***opcode, funct3, funct7*** της λέξης εντολής. Συγκεκριμένα, οι εντολές LW/SW απαιτούν πρόσθεση, η εντολή BEQ αφαίρεση, ενώ οι εντολές τύπου Register-to-Register, ALU Immediate απαιτούν λειτουργία που υπαγορεύεται από τα πεδία ***funct3, funct7***.

  Αρχικά, ορίζουμε 9 νέες παραμέτρους μεγέθους 4-bit, οι οποίες αναπαριστούν τις 9 λειτουργίες της ALU. Οι τιμές των παραμέτρων αυτών ορίζονται με βάση την κωδικοποίηση των εντολών R-type του RISC-V instruction set. Αναλυτικότερα, το MSB της κάθε παραμέτρου είναι το bit instr[30] του πεδίου ***funct7***, ενώ τα 3 LSB είναι τα bits instr[14:12] του πεδίου ***funct3*** των εντολών R-type. Χρησιμοποιώντας την εντολή defparam και το κατάλληλο ιεραρχικό μονοπάτι DATAPATH.ALU.<parameter_name> μπορούμε να κάνουμε override τις τιμές των παραμέτρων του **alu module**, το οποίο βρίσκεται μέσα στο **datapath module**, που δηλώνεται αργότερα μέσα στο παρόν module. Με αυτόν τον τρόπο, μπορούμε να υπαγορεύσουμε άμεσα στην ALU την επιθυμητή λειτουργία με βάση τα κατάλληλα bits της λέξης-εντολής.

  Δηλώνουμε, λοιπόν, μια μεταβλητή ALUCtrl μεγέθους 4-bit τύπου reg (βρίσκεται στο αριστερό μέρος ανάθεσης μέσα σε procedural block). Δηλώνουμε ένα συνδυαστικό always block με λίστα ευαισθησίας *. Μέσα στο block χρησιμοποιούμε την εντολή casez με είσοδο το πεδίο ***opcode*** της λέξης εντολής, έτσι ώστε να διαχωρίσουμε τις εντολές με βάση τον τύπο τους.

  - Στην περίπτωση των εντολών R-type, η λειτουργία της ALU υπαγορεύεται όπως εξηγήθηκε παραπάνω από το bit instr[30] του πεδίου ***funct7*** και τα bits instr[14:12] του πεδίου ***funct3***.
  - Στην περίπτωση των εντολών ALU Immediate δεν υπάρχει πεδίο ***funct7***, παρά μόνο για τις εντολές ολίσθησης. Για το λόγο αυτό, για τις εντολές ολίσθησης η λειτουργία της ALU καθορίζεται όπως και για τις εντολές R-type, ενώ για τις υπόλοιπες εντολές η λειτουργία καθορίζεται μόνο από το πεδίο ***funct3*** προσθέτοντας ως LSB την τιμή 0, το οποίο συνάδει με την κωδικοποίηση των παραμέτρων.
  - Στην περίπτωση των εντολών LW/SW, η λειτουργία ανατίθεται ως πρόσθεση.
  - Στην περίπτωση της εντολής BEQ, η λειτουργία ανατίθεται ως αφαίρεση.
  - Σε κάθε άλλη περίπτωση (default case), η τιμή της λειτουργίας είναι απροσδιόριστη.

- ALUSrc

  Το σήμα ALUSrc υπαγορεύει την πηγή του δεύτερου τελεσταίου της ALU, με βάση τον τύπο της εντολής, είναι δηλαδή το σήμα επιλογής ενός πολυπλέκτη. Οι εντολές LW/SW και οι εντολές τύπου ALU Immediate απαιτούν άμεσα δεδομένα (immediate data) στη δεύτερη θύρα εισόδου της μονάδας ALU, ενώ οι υπόλοιπες εντολές όχι.

  Δηλώνουμε, λοιπόν, μια μεταβλητή ALUSrc μεγέθους 1-bit τύπου reg (βρίσκεται στο αριστερό μέρος ανάθεσης μέσα σε procedural block). Δηλώνουμε ένα συνδυαστικό always block με λίστα ευαισθησίας *. Μέσα στο block χρησιμοποιούμε την εντολή casez με είσοδο το πεδίο ***opcode*** της λέξης εντολής, έτσι ώστε να διαχωρίσουμε τις εντολές με βάση τον τύπο τους.

  - Στην περίπτωση των εντολών LW/SW και ALU Immediate, το σήμα ALUSrc λαμβάνει την τιμή 1.
  - Σε κάθε άλλη περίπτωση (default case), λαμβάνει την τιμή 0.

Multiplexer Control Signals

Τα σήματα MemToReg, PCSrc αποτελούν σήματα ελέγχου των πολυπλεκτών εισόδου του καταχωρητή του program counter και των δεδομένων εγγραφής στο αρχείου καταχωρητών. Η τιμή τους εξαρτάται από την τρέχουσα εντολή, ενώ η τιμή του PCSrc εξαρτάται επιπλέον από την τιμή του σήματος εξόδου της ALU Zero. Παρακάτω σχολιάζουμε την τρόπο παραγωγής των δύο σημάτων:

- MemToReg

  Το σήμα MemToReg υπαγορεύει την πηγή των δεδομένων εγγραφής στο αρχείο καταχωρητών και εξαρτάται μόνο από την τρέχουσα εντολή. Δηλώνουμε ένα εσωτερικό σήμα τύπου reg μεγέθους 1-bit. Δηλώνουμε ένα always block, του οποίου η λίστα ευαισθησίας περιλαμβάνει το σήμα instr, δηλαδή την τιμή της λέξης εντολής. Μέσα στο block χρησιμοποιούμε την εντολή if η οποία ελέγχει την τιμή του πεδίου ***opcode*** της λέξης εντολής. Στην περίπτωση που η τρέχουσα εντολή πρόκειται για μια εντολή LW, το σήμα MemToReg ενεργοποιείται, οπότε στην έξοδο του πολυπλέκτη εμφανίζονται τα δεδομένα ανάγνωσης της μνήμης δεδομένων. Σε αντίθετη περίπτωση, το σήμα παραμένει ανενεργό και στην έξοδο του πολυπλέκτη εμφανίζεται το αποτέλεσμα της ALU.

- PCSrc

  Το σήμα PCSrc υπαγορεύει την πηγή της εισόδου του καταχωρητή του program counter και εξαρτάται τόσο από την τρέχουσα εντολή όσο και από την τιμή του σήματος Zero. Δηλώνουμε ένα εσωτερικό σήμα PCSrc τύπου reg μεγέθους 1-bit καθώς και ένα εσωτερικό σήμα Zero τύπου wire μεγέθους 1-bit, το οποίο συνδέεται στη θύρα εξόδου του **datapath module**. Δηλώνουμε ένα always block, του οποίου η λίστα ευαισθησίας περιλαμβάνει το σήμα instr, δηλαδή την τιμή της λέξης εντολής και το σήμα Zero, το οποίο δηλώνει αν το αποτέλεσμα της ALU είναι ίσο με 0. Μέσα στο block χρησιμοποιούμε την εντολή if η οποία ελέγχει την τιμή του πεδίου ***opcode*** της λέξης εντολής καθώς και την τιμή του σήματος Zero. Στην περίπτωση που η τρέχουσα εντολή πρόκειται για μια εντολή BEQ και το σήμα Zero είναι ενεργοποιημένο, το σήμα PCSrc ενεργοποιείται, οπότε στην έξοδο του πολυπλέκτη εμφανίζεται το άθροισμα PC + branch_offset. Σε αντίθετη περίπτωση, το σήμα παραμένει ανενεργό και στην έξοδο του πολυπλέκτη εμφανίζεται το άθροισμα PC + 4.

Datapath Unit

Δηλώνουμε πρώτα τα εσωτερικά σήματα διασύνδεσης RegWrite, loadPC ως τύπου reg. Στη συνέχεια, κάνουμε instantiate το **datapath module**, κάνοντας override την παράμετρο INITIAL_PC του module από την αντίστοιχη παράμετρο του **multicycle module**. Κατά την αντιστοίχιση θυρών (port mapping) συνδέουμε κατάλληλα τα εσωτερικά σήματα που δηλώθηκαν παραπάνω, τα εσωτερικά σήματα ALUCtrl, ALUSrc που παρήχθηκαν στη μονάδα ALU Control Logic και τα εσωτερικά σήματα MemToReg, PCSrc που παρήχθηκαν για τον έλεγχο των πολυπλεκτών. Τέλος, συνδέουμε τα σήματα εξόδου dAddress, dWriteData, WriteBackData, PC του **multicycle module** στις αντίστοιχες θύρες εξόδου του **datapath module**, καθώς και τα σήματα εισόδου dReadData, instr, rst, clk του **multicycle module** στις αντίστοιχες θύρες εισόδου του **datapath module**.

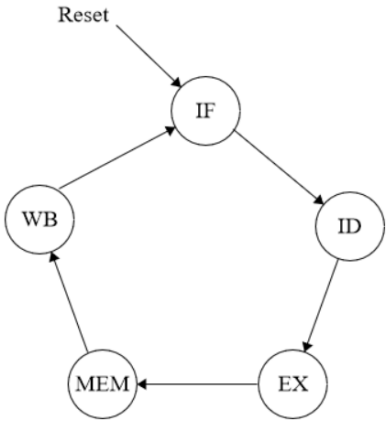
Finite State Machine

Το επόμενο και τελικό βήμα σχεδιασμού της μονάδας ελέγχου είναι ο σχεδισμός μιας μηχανής πεπερασμένων καταστάσεων (Finite State Machine), η οποία διατρέχει με τη σειρά τα ακόλουθα 5 βήματα εκτέλεσης μιας εντολής:

Κατάσταση	**Σκοπός**
IF	Instruction Fetch: Παροχή του PC στη μνήμη εντολών
ID	Instruction Decode: Αποκωδικοποίηση της ληφθείσας εντολής και έναρξη πρόσβασης στους καταχωρητές
EX	Execute: Εκτέλεση της λειτουργίας στην ALU
MEM	Memory: Εκτέλεση πρόσβασης στη μνήμη (για lw/sw)
WB	Write Back: Εγγραφή νέων δεδομένων στους καταχωρητές

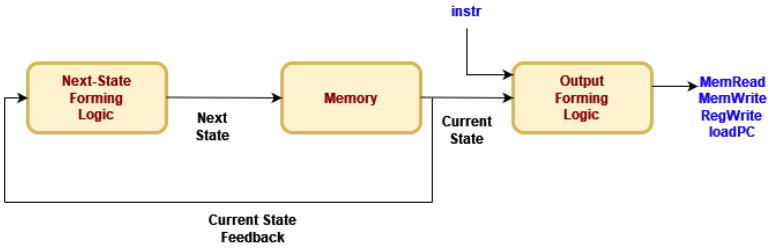
Σχήμα 6: Οι 5 καταστάσεις της μηχανής

Το διάγραμμα καταστάσεων (State Diagram) της μηχανής φαίνεται στο παρακάτω σχήμα:



Σχήμα 7: Finite State Machine State Diagram

Η μηχανή που θα σχεδιάσουμε έχει σήμα εισόδου την λέξη εντολής instr, ενώ τα σήματα εξόδου είναι τα σήματα MemRead, MemWrite, RegWrite, loadPC. Παρατηρούμε ότι τα σήματα εξόδου καθορίζονται τόσο από την τρέχουσα κατάσταση όσο και από το σήμα εισόδου. Συνεπώς, η μηχανή είναι μια μηχανή **Mealy**.



Σχήμα 8: Finite State Machine Diagram

Στα παρακάτω σχήματα απεικονίζονται ο πίνακας μεταβάσεων (state transition table) και ο πίνακας εξόδων (output table). Οι μεταβάσεις των καταστάσεων δεν εξαρτώνται από την είσοδο και προκύπτουν μόνο από την τρέχουσα κατάσταση όπως φαίνεται στον πίνακα. Αντίθετα, οι έξοδοι εξαρτώνται τόσο από την είσοδο όσο και από την τρέχουσα κατάσταση.

- Το σήμα MemRead ενεργοποιείται μόνο κατά το στάδιο **Memory**, όταν η εντολή που εκτελείται είναι η LW, έτσι ώστε να επιτραπεί η ανάγνωση από τη μνήμη δεδομένων.
- Το σήμα MemWrite ενεργοποιείται μόνο κατά το στάδιο **Memory**, όταν η εντολή που εκτελείται είναι η SW, έτσι ώστε να επιτραπεί η εγγραφή στη μνήμη δεδομένων.
- Το σήμα loadPC ενεργοποιείται μόνο κατά το στάδιο **Write Back**, ανεξαρτήτως της εντολής που εκτελείται. Με αυτόν τον τρόπο, επιτρέπει την ενημέρωση της τιμής του program counter σε μια νέα τιμή πριν από το επόμενο στάδιο του **Instruction Fetch**, στο οποίο θα ξεκινήσει η εκτέλεση της νέας εντολής.
- Το σήμα RegWrite χρησιμοποιείται για τον έλεγχο της εγγραφής τιμών στο αρχείο καταχωρητών. Ενεργοποιείται μόνο κατά το στάδιο **Write Back**. Οι εντολές SW, BEQ δεν απαιτούν εγγραφή πίσω στο αρχείο καταχωρητών, επομένως στην περίπτωση αυτών το σήμα παραμένει ανενεργό.

Το σύμβολο **X** σημαίνει don't care/ αδιάφορη τιμή.
Το σύμβολο ∼ σημαίνει otherwise/ διαφορετικά, δηλαδή σε όλες τις υπόλοιπες περιπτώσεις εισόδου.

Current State	Next State
Instruction Fetch	Instruction Decode
Instruction Decode	Execute
Execute	Memory
Memory	Write Back
Write Back	Instruction Fetch

Σχήμα 9: State Transition Table

Current State	Input	Outputs			
	instr	MemRead	MemWrite	RegWrite	loadPC
Instruction Fetch	X	0	0	0	0
Instruction Decode	X	0	0	0	0
Execute	X	0	0	0	0
Memory	LW	1	0	0	0
Memory	SW	0	1	0	0
Memory	~	0	0	0	0
Write Back	SW, BEQ	0	0	0	1
Write Back	~	0	0	1	1

Σχήμα 10: Output Table

Το τελικό βήμα στη σχεδίαση του FSM είναι η επιλογή μιας κωδικοποίησης των καταστάσεων (state assignment). Σε αυτή την περίπτωση επιλέγουμε να χρησιμοποιήσουμε τον δυαδικό κώδικα. Καθώς το πλήθος των καταστάσεων είναι ίσο με 5, η κάθε δυαδική λέξη είναι απαραίτητο να αποτελείται από τουλάχιστον 3 bits, έτσι ώστε να μπορεί να κωδικοποιηθεί όλο το πληθος των καταστάσεων.

State	Encoding
Instruction Fetch	000
Instruction Decode	001
Execute	010
Memory	011
Write Back	100

Σχήμα 11: State Assignment

Για την περιγραφή ενός FSM σε Verilog απαιτούνται τα ακόλουθα βήματα:

- Ορίζουμε δύο σήματα τύπου reg τα οποία μοντελοποιούν την παρούσα και επόμενη κατάσταση. Τα σήματα αυτά ονομάστηκαν current_state, next_state αντίστοιχα και έχουν μέγεθος 3-bit, δεδομένης της ανάθεσης καταστάσεων που περιγράφηκε παραπάνω.
- Ορίζουμε παραμέτρους για κάθε κατάσταση της μηχανής. Οι παράμετροι αυτές έχουν μέγεθος 3-bit και λαμβάνουν τις τιμές της επιλεγμένης δυαδικής κωδικοποίησης των καταστάσεων.
- Ορίζουμε 3 procedural blocks για να περιγράψουμε την αποθήκευση της επόμενης κατάστασης (ακολουθιακή λογική), τη λογική που προσδιορίζει την επόμενη κατάσταση (συνδυαστική λογική) και τη λογική που προσδιορίζει τις τιμές των εξόδων (συνδυαστική λογική).

Procedural Block Αποθήκευσης Κατάστασης

Δηλώνουμε ένα always block, του οποίου η λίστα ευαισθησίας περιλαμβάνει το σήμα ρολογιού clk, συνοδευόμενο από τη λέξη-κλειδί posedge. Σε κάθε θετική ακμή του ρολογιού, αν το σύγχρονο σήμα rst είναι ενεργό, τότε η τρέχουσα κατάσταση ορίζεται ως η κατάσταση Instruction Fetch. Διαφορετικά, η τρέχουσα κατάσταση ορίζεται ως η επόμενη κατάσταση.

Procedural Block για την Επόμενη Κατάσταση

Δηλώνουμε ένα always block του οποίου η λίστα ευαισθησίας περιλαμβάνει το σήμα current_state. Χρησιμοποιούμε μια εντολή case η οποία προσδιορίζει την μετάβαση από την τρέχουσα κατάσταση στην επόμενη, όπως ορίστηκε από τον πίνακα μετάβασης καταστάσεων. Στην περίπτωση default, ως επόμενη κατάσταση ανατίθεται η κατάσταση Instruction Fetch.

Procedural Block για την Εξόδους

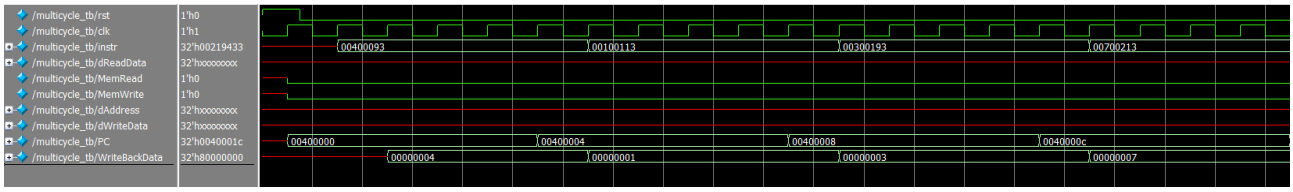
Δηλώνουμε ένα always block του οποίου η λίστα ευαισθησίας περιλαμβάνει το σήμα current_state και το σήμα εισόδου instr (μηχανή **Mealy**). Χρησιμοποιούμε μια εντολή case με είσοδο την τρέχουσα κατάσταση current_state. Οι τιμές των εξόδων στα στάδια **Memory, Write Back** ανατίθενται όπως εξηγήθηκε στον πίνακα εξόδων, ενώ στην περίπτωση default, όλες οι τιμές των εξόδων ανατίθενται ίσες με 0.

Testbench

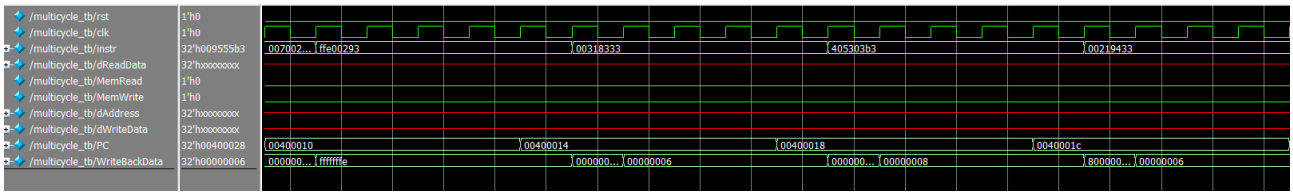
Για να ελέγξουμε την ορθή λειτουργία του επεξεργαστή που σχεδιάσαμε, δημιουργούμε ένα νέο αρχείο Verilog με όνομα multicycle_tb.v. Το αρχείο αυτό αποτελεί ένα testbench αρχείο το οποίο παράγει ένα σύνολο πιθανών συνδυασμών σημάτων εισόδου και ελέγχει αν οι τιμές των σημάτων εξόδου του **multicycle module** είναι οι αναμενόμενες.
Πρώτο μας βήμα είναι να ορίσουμε το **timescale** της προσομοίωσης. Στη συνέχεια, δηλώνουμε ένα module με όνομα multicycle_tb. Επισημαίνουμε πως το testbench module δεν απαιτεί σήματα εισόδου και εξόδου. Στη σχεδίαση του testbench module ακολουθούμε τα εξής βήματα:

- Δηλώνουμε τα σήματα εισόδου και εξόδου του **multicycle module**. Τα σήματα εισόδου ορίζονται ως τύπου reg, ενώ τα σήματα εξόδου ως τύπου wire.
- Κάνουμε instantiate τα modules: **multicycle, rom, ram**, τα οποία αποτελούν τον επεξεργαστή, τη μνήμη εντολών και τη μνήμη δεδομένων αντίστοιχα.
- Κάνουμε την αντιστοίχιση θυρών των modules. Συνδέουμε τα σήματα εισόδου και εξόδου που δηλώσαμε παραπάνω στις κατάλληλες θύρες του **multicycle module** (η αντιστοίχιση έγινε χωροταξικά). Στη θύρα εισόδου του **rom module** που αναπαριστά τη διεύθυνση ανάγνωσης συνδέουμε τα 9 LSB του program counter, ενώ στη θύρα εξόδου που αναπαριστά τα δεδομένα ανάγνωσης συνδέουμε το σήμα που αναπαριστά την λέξη εντολής. Τέλος, συνδέουμε στην είσοδο του **rom module** το σήμα ρολογιού. Συνδέουμε στην είσοδο του **ram module** το σήμα ρολογιού και τα σήματα ελέγχου MemRead, MemWrite. Αντιστοιχίζουμε, επίσης, τις θύρες δεδομένων ανάγνωσης/εγγραφής και τη θύρα διεύθυνσης ανάγνωσης/εγγραφής.
- Παράγουμε το σήμα rst και το σήμα ρολογιού clk. Για την παραγωγή του σήματος rst χρησιμοποιούμε ένα initial block, το οποίο εκτελείται μόνο μια φορά κατά την προσομοίωση. Αναθέτουμε στο σήμα την τιμή 1 (active-high signal) για 15 μονάδες χρόνου και στη συνέχεια για το υπόλοιπο της προσομοίωσης αναθέτουμε την τιμή 0. Φροντίζουμε έτσι ώστε το σήμα rst να έχει την τιμή 1 στην πρώτη θετική ακμή του ρολογιού, καθώς το σήμα είναι σύγχρονο με το ρολόι. Για την παραγωγή του σήματος ρολογιού, χρησιμοποιούμε ένα initial block και ένα always block. Στο initial block, το οποίο εκτελείται μόνο μια φορά, αναθέτουμε στο σήμα την αρχική τιμή 0. Έπειτα, στο always block, αναθέτουμε στο σήμα κάθε 10 μονάδες χρόνου την αντίθετη τιμή από την τρέχουσα, με αποτέλεσμα να παράγονται παλμοί περιόδου 20 μονάδων χρόνου.
- Για να επιβεβαιώσουμε ότι τα δεδομένα που εγγράφονται στο αρχείο καταχωρητών είναι τα αναμενόμενα, αποθηκευούμε σε ένα αρχείο registers.txt την τιμή του σήματος Write Back Data κατά το στάδιο **Write Back** της μηχανής, καθώς και την αντίστοιχη τιμή του program counter που δείχνει την διεύθυνση της τρέχουσας εντολής στη μνήμη εντολών. Η μηχανή βρίσκεται για πρώτη φορά στο στάδιο **Write Back** μετά από 100 μονάδες χρόνου, οπότε τότε γίνεται η πρώτη εγγραφή στο αρχείο. Οι επόμενες εγγραφές γίνονται κάθε 5 κύκλους ρολογιού (δηλαδή 5*20 = 100 μονάδες χρόνου).

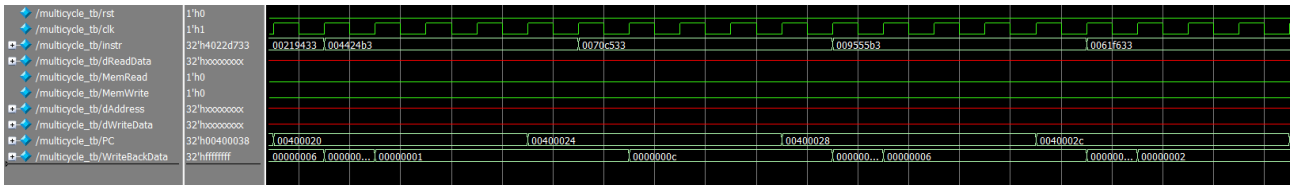
Παρακάτω παρουσιάζονται οι κυματομορφές που παρήχθηκαν κατά την προσομοίωση, καθώς και τα περιεχόμενα του αρχείου registers.txt.



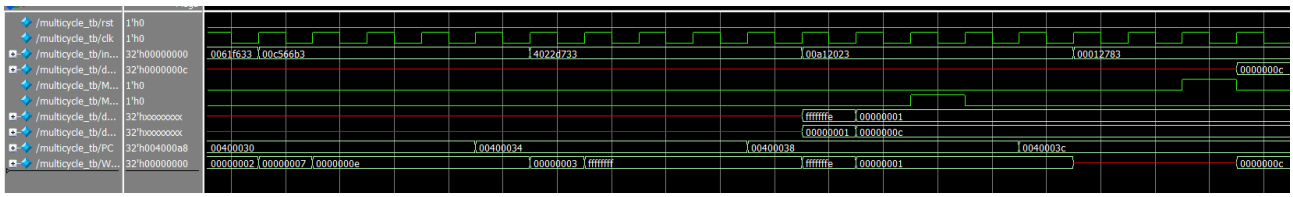
Σχήμα 12: Simulation (Instructions 1-4)



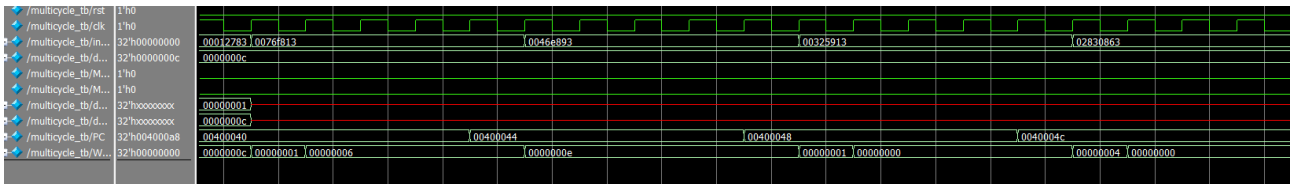
Σχήμα 13: Simulation (Instructions 5-8)



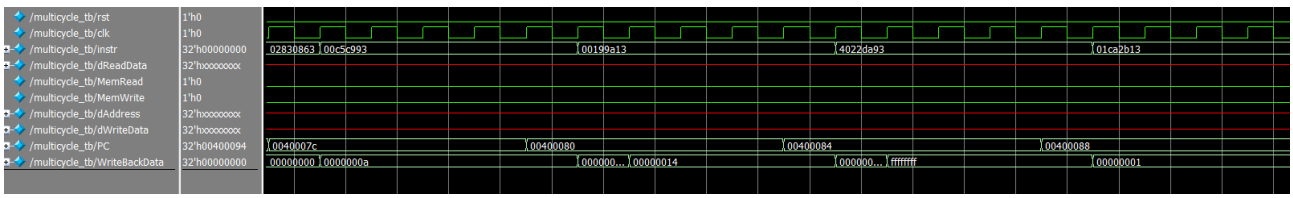
Σχήμα 14: Simulation (Instructions 9-12)



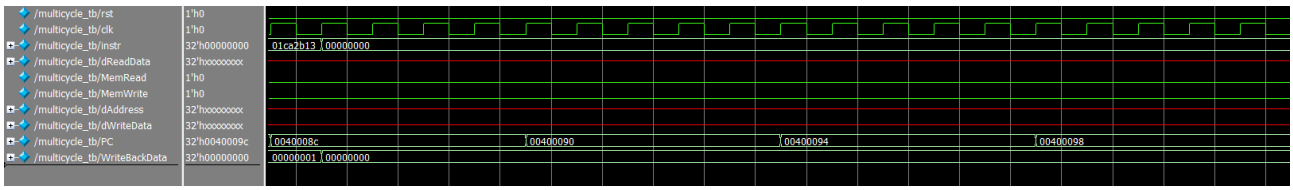
Σχήμα 15: Simulation (Instructions 13-16)



Σχήμα 16: Simulation (Instructions 17-20)



Σχήμα 17: Simulation (Instructions 21-24)



Σχήμα 18: End of Simulation

Στις κυματομορφές φαίνονται το σήμα rst, το σήμα ρολογιού clk, η τιμή της λέξης εντολής instr, οι θύρες dReadData, dWriteData, dAddress της μνήμης δεδομένων, τα σήματα ελέγχου MemRead, MemWrite της μνήμης δεδομένων, ο program counter PC και τέλος τα δεδομένα Write Back Data.

```
PC: 00400000    | Write Back Data: 00000004
PC: 00400004    | Write Back Data: 00000001
PC: 00400008    | Write Back Data: 00000003
PC: 0040000c    | Write Back Data: 00000007
PC: 00400010    | Write Back Data: fffffffe
PC: 00400014    | Write Back Data: 00000006
PC: 00400018    | Write Back Data: 00000008
PC: 0040001c    | Write Back Data: 00000006
PC: 00400020    | Write Back Data: 00000001
PC: 00400024    | Write Back Data: 0000000c
PC: 00400028    | Write Back Data: 00000006
PC: 0040002c    | Write Back Data: 00000002
PC: 00400030    | Write Back Data: 0000000e
PC: 00400034    | Write Back Data: ffffffff
PC: 00400038    | Write Back Data: 00000001
PC: 0040003c    | Write Back Data: 0000000c
PC: 00400040    | Write Back Data: 00000006
PC: 00400044    | Write Back Data: 0000000e
PC: 00400048    | Write Back Data: 00000000
PC: 0040004c    | Write Back Data: 00000000
PC: 0040007c    | Write Back Data: 0000000a
PC: 00400080    | Write Back Data: 00000014
PC: 00400084    | Write Back Data: ffffffff
PC: 00400088    | Write Back Data: 00000001
```

Σχήμα 19: Write Back Data

Χρησιμοποιώντας έναν **disassembler**, προσδιορίστηκαν οι εντολές που εκτελούνται από τον επεξεργαστή.

```
PC: 00400000    | addi x1, x0, 4
PC: 00400004    | addi x2, x0, 1
PC: 00400008    | addi x3, x0, 3
PC: 0040000c    | addi x4, x0, 7
PC: 00400010    | addi x5, x0, -2
PC: 00400014    | add x6, x3, x3
PC: 00400018    | sub x7, x6, x5
PC: 0040001c    | sll x8, x3, x2
PC: 00400020    | slt x9, x8, x4
PC: 00400024    | xor x10, x1, x7
PC: 00400028    | srl x11, x10, x9
PC: 0040002c    | and x12, x3, x6
PC: 00400030    | or x13, x10, x12
PC: 00400034    | sra x14, x5, x2
PC: 00400038    | sw x10, 0(x2)
PC: 0040003c    | lw x15, 0(x2)
PC: 00400040    | andi x16, x13, 7
PC: 00400044    | ori x17, x13, 4
PC: 00400048    | srli x18, x3, 3
PC: 0040004c    | beq x6, x8, 48
PC: 0040007c    | xori x19, x11, 12
PC: 00400080    | slli x20, x19, 1
PC: 00400084    | srai x21, x5, 2
PC: 00400088    | slti x22, x20, 28
```

Σχήμα 20: Instructions

Συγκρίνοντας τα αποτελέσματα των παρακάτω εντολών με τις τιμές των δεδομένων Write Back Data, επαληθεύεται η σωστή λειτουργία του επεξεργαστή.

Προσοχή: Η εντολή beq στη διεύθυνση 0040004c πραγματοποιεί διακλάδωση, η οποία στέλνει το πρόγραμμα σε διευθύνσεις της μνήμης εντολών όπου δεν υπάρχουν αποθηκευμένες εντολές, παρά μόνο μηδενικά. Για το λόγο αυτό, τροποποιήθηκε το αρχείο rom_bytes.data έτσι ώστε μετά την διακλάδωση το πρόγραμμα να οδηγείται στην πρώτη μη μηδενική εντολή στη διεύθυνση 0040007c. Αυτό έγινε αλλάζοντας το branch_offset από την αρχική του τιμή 6 στην τιμή 48.