Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Σχεδίαση και υλοποίηση πυρήνα επεξεργαστή με διοχέτευση διπλής εντολής χωρίς ανάγκη πρόβλεψης διακλαδώσεων

Διπλωματική Εργασία

ΚΩΝΣΤΑΝΤΙΝΟΣ Α. ΜΟΣΧΟΣ

Επιβλέπων
Μπέλλας Νικόλαος
Καθηγητής

Βόλος, Σεπτέμβριος 2020

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Σχεδίαση και υλοποίηση πυρήνα επεξεργαστή με διοχέτευση διπλής εντολής χωρίς ανάγκη πρόβλεψης διακλαδώσεων

Διπλωματική Εργασία

ΚΩΝΣΤΑΝΤΙΝΟΣ Α. ΜΟΣΧΟΣ

Επιτροπή επίβλεψης

Επιβλέπων	Συνεπιβλέπων	Συνεπιβλέπων
Μπέλλας Νικόλαος	Αντωνόπουλος Χρήστος	Δημητρίου Γεώργιος
Καθηγητής	Αναπληρωτής Καθηγητής	Επίκουρος Καθηγητής

Βόλος, Σεπτέμβριος 2020

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Κων/νος Μόσχος

Βόλος,Σεπτέμβριος 2020

University of Thessaly
Faculty of Engineering
Department of Electrical & Computer Engineering

Design and implementation of a processor core with a double-instruction pipeline and no need of branch prediction

Diploma Thesis

KONSTANTINOS A. MOSCHOS

Supervisor
Bellas Nikolaos
Professor

Volos, September 2020

Περίληψη

Η εργασία αυτή ανήκει στην επιστημονική περιοχή της Αρχιτεκτονικής Υπολογιστών.Έγινε σχεδίαση και υλοποίηση ενός επεξεργαστή σε γλώσσα Verilog που ανταποκρίνεται στο βασικό RISC-V ISA. Σκοπός της εργασίας είναι η αντιμετώπιση του προβλήματος της ποινής που προκύπτει απο το πάγωμα και το άδειασμα των σταδίων μιας αρχιτεκτονικής παροχέτευσης μετά από εντολές διακλάδωσης .Η βασική ιδέα της αρχιτεκτονικής έγκειται στην έκδοση μια 64 Bit λέξης εντολής που έχει κατασκευαστεί κατάλληλα σε επίπεδο μεταγλώττισης ώστε να περιλαμβάνει¨:
i)Στην γενική περίπτωση εντολές απο διαδοχικές θέσεις μνήμης, οι οποίες θα αποδικωποιηθούν παράλληλα απο την διπλή μονάδα αποκωδικοποιήσης και ύστερα θα προωθηθούν σειρικά για εκτέλεση.
ii)Στην ειδική περίπτωση μετά απο εντολή διακλάδωσης οι δύο επόμενες λέξεις εντολών 64Bit περιλαμβάνουν εντολές και απο τις δύο κατευθύνσεις τις διακλάδωσης οι οποίες έχουν προπαρασκευαστεί κατάλληλα απο τον μεταγλωττιστή και τοποθετηθεί διαδοχικά στην μνήμη .
Αυτό επιτρέπει στον επέξεργαστη να αποτιμήσει την εντολή διακλάδωσης και με κατάλληλα σήματα να επιλέξει ποιες εντολές θα πρέπει να εκτελεστούν απο τα επόμενα μπλοκ ,ενω η επόμενη εκδοσή εντολών θα γίνει από έγκυρη διευθυνση μνήμης.Τέλος έγινε μια θεωρητική ανάλυση για το πως θα συνέβαλε η υλοποίηση μας σε αλγορίθμους που αντιμετωπίζουνε πρόβλημα στην πρόβλεψη των διακλάδωσεων με τις υπάρχουσες μεθόδους που συναντόνται στην βιβλιογραφία αλλά και στις εμπορικές υλοποιήσεις ,οι οποίες και αναλύονται .

Λέξεις Κλειδιά

αρχιτεκτονική υπολογιστών,αρχιτεκτονική παροχεύτευσης,risc-v,πρόβλεψη διακλαδώσεων , έκδοση διπλής εντολής,μεταγλωττιστής , δυναμικές τεχνικές προβλεψης , στατικές τεχνικές πρόβλεψης ,quicksort, k-nearest neighbor . . .

Abstract

This thesis belongs on the Computer Architecture scientific area. A RISC-V processor was designed and implemented in Verilog Language. This architecture solves the problem of penalty cycles which arises from the process of stalling and flushing the stages of a Pipelined architecture after branch instructions. The main idea of the architecture is found on a 64 bit version instruction-word that has properly manufactured in a compiler level in order to include:
I. General, instruction from consecutive memory positions, which will be parallel decoded from the double decode unit and then will be serial forwarded for execution.
II. While in the special case of a branch instruction the following two 64bit instruction-words include instructions, from both directions of the branch, which have been properly constructed from the compiler and have sequentially been placed in the memory.
This allows the processor to evaluate the branch instructions and choose with the right signals which instructions have to be executed from the following blocks, while the next instruction will be fetched from valid memory address. Then I worked on a theoretical analysis about how the processor that designed and implemented could reduce the execution time of the algorithms which are confronted with the branch prediction according to the existing branch prediction schemes and the known commercial implementations.

Keywords

computer architecture, pipelined architecture,branch prediction ,risc-v,penalty cycles,64-bit double instruction word,static branch predictions schemes ,dynamic branch prediction schemes,quicksort,k-nearest neighbor . . .

Εδώ αναφέρονται οι αφιερώσεις (προαιρετικό τμήμα).

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ.Δημητρίου για την ανάθεση της παρούσας διπλωματικής καθώς και για την πολύτιμη βοήθεια του κατά την διάρκεια πραγματοποίησης της.

Περιεχόμενα

Περίληψη i

Abstract iii

Ευχαριστίες vii

Περιεχόμενα ix

Κατάλογος σχημάτων xiii

Κατάλογος πινάκων xvii

1 Εισαγωγή 1

1.1 Στόχος της εργασίας 1

1.2 Οργάνωση του τόμου 2

2 Αρχιτεκτονική Υπολογιστών 3

2.1 Γενικά 3

2.2 Επίπεδα Οργάνωσης 3

2.2.1 Αρχιτεκτονική Συνόλου Εντολών 3

2.2.2 Μικροαρχιτεκτονική 3

2.2.3 Σχεδίαση Συστήματος 4

2.3 Αρχιτεκτονική CISC 4

2.4 Αρχιτεκτονική RISC 4

2.5 Παράδειγμα σύγκρισης 5

2.5.1 Η προσέγγιση CISC 5

2.5.2 Η προσέγγιση RISC 6

2.6 Η εξίσωση απόδοσης 7

2.7 Νόμος του Amdahl 7

3 RISC-V ISA 9

3.1 Γενικά 9

3.2 Κωδικοποίηση Μήκος Εντολής 10

3.3 Φάκελος καταχωρητων . . . 11
3.4 Βασικές κωδικοποιήσεις εντολών . . . 11
3.5 Εντολές ακεραίων . . . 12
3.5.1 Πράξεις ακεραίων μεταξύ καταχωρητή και στάθερας . . . 13
3.5.2 Πράξεις ακεραίων μεταξύ καταχωρητών . . . 14
3.5.3 Εντολές φόρτωσης και αποθήκευσης . . . 15
3.5.4 Εντολές ελέγχου ροής προγράμματος . . . 16

4 Classic RISC pipeline **19**
4.1 Single Cycle Επεξεργαστής . . . 19
4.2 Pipelined Επεξεργαστής . . . 20
4.3 Κίνδυνοι (Hazards) . . . 22
4.4 Branch prediction . . . 25
4.5 Τεχνικές πρόβλεψης διακλαδώσεων . . . 26
4.5.1 Στατικές . . . 26
4.5.2 Δυναμικές . . . 27
4.5.3 Branch Target Buffer . . . 36
4.5.4 Predication . . . 38

5 Περιγραφή υλοποίησης **41**
5.1 Γενική περιγραφή . . . 41
5.2 Στάδια υλοποιησής . . . 44
5.2.1 Instruction fetch unit . . . 44
5.2.2 Instruction Decode unit . . . 45
5.2.3 Flow control unit . . . 47
5.2.4 Forward unit . . . 49
5.2.5 Execution stage . . . 50
5.2.6 Memory stage . . . 52
5.2.7 Write back stage . . . 53
5.3 Πρωτοτυπία υλοποίησης . . . 54

6 Επαλήθευση λειτουργίας **57**
6.1 Προσομοίωση . . . 57
6.2 Έλεγχος διπλής έκδοσης . . . 57
6.3 Παραδείγματα ελέγχου . . . 60
6.3.1 Risc-v assembly χωρίς εντολές διακλάδωσης I . . . 60
6.3.2 Risc-v assembly χωρίς εντολές διακλάδωσης II . . . 63
6.3.3 Risc-v assembly με εντολές διακλάδωσης . . . 66

7 Αλγόριθμοι **71**
7.1 Κριτήριο επιλογής . . . 71
7.2 Quicksort . . . 71

7.3 Ανάλυση πολυπλοκότητας με βάση το στοιχείο διαχωρισμού 74
7.4 Quicksort και Branch Misprediction 76
7.5 Υλοποίηση του Quicksort για αρχιτεκτονική διπλής έκδοσης 80
7.6 K-nearest neighbor 84
7.7 K-nearest neighbor και branch misprediction 85

8 Επίλογος **91**
8.1 Σύνοψη και συμπεράσματα 91
8.2 Μελλοντικές επεκτάσεις 91

I **93**

Βιβλιογραφία **95**

Κατάλογος σχημάτων

2.1 Μοντέλο Μνήμης-Καταχωρητών και μονάδας εκτέλεσης 6

3.1 Κωδικοποίηση Εντολής 11
3.2 Βασικές κωδικοποιήσεις εντολών 12
3.3 B-type J-type 12
3.4 Πράξεις Καταχωρητή-στεθεράς 13
3.5 Πράξεις Καταχωρητή-στεθεράς 14
3.6 Πράξεις Καταχωρητή-σταθεράς 14
3.7 Πράξεις μεταξύ καταχωρητών 15
3.8 Πράξεις μεταξύ καταχωρητών 15
3.9 Εντολές φόρτωσης και αποθήκευσης 16
3.10 Η εντολή JAL 16
3.11 Η εντολή JALR 17
3.12 Οι εντολές διακλάδωσεις με συνθήκη 17
3.13 RV32I Σύνολο Εντολών 18

4.1 Risc-v RV32I simple datapath 20
4.2 Παροχεύτευση εντολών 21
4.3 Pipelined RV32I proseccor 22
4.4 Δομική εξάρτηση μνήμης 23
4.5 Data hazards μεταξύ σταδίων 24
4.6 Block Diagram επεξεργαστή με μονάδα κινητής υποδιαστολής πολλαπλασιασμού και διαίρεσης και υποστήριξη εντολών εκτός σειράς(Pentium 4). 24
4.7 Αποτίμηση εντολής διακλάδωσης 3cc αργότερα 25
4.8 Εντολές που αλλάζουν την ροή εκτέλεσης 25
4.9 Αποτίμηση εντολών 26
4.10 Παραδείγματα πλήρωσης delay slot 27
4.11 Διάγραμμα καταστάσεων 1 bit predictor 28
4.12 Πίνακας αλήθειας 1 bit predictor 28
4.13 Διάγραμμα καταστάσεων 2 Bit predictor 29
4.14 Απεικόνιση πρόβλεψης local branch predictor για κάθε πιθανή ιστορία μιας εντολής διακλάδωσης. 30

4.15 Συνολό-συσχετιστική απεικόνιση local branch predictor 30
4.16 Global Branch Prediction 31
4.17 Predictor with shared counters 32
4.18 Tournament predictor of alpha 21264 proseccor 33
4.19 Hierachical predictor 34
4.20 Άθροιση των γινονομένου εισόδων –βαρών για την κατασκευη του αλγορίθμου perceptron 35
4.21 Perceptron branch predictor 35
4.22 Return address stack 37
4.23 Branch Target Buffer 37
4.24 Παράδειγμα κώδικα 38
4.25 Παραδοσιακή Αρχιτεκτονική , Αρχιτεκτονική IA-64 39

5.1 64bit λέξη εντολής 41
5.2 Αρχιτεκτονική 5 σταδίων 42
5.3 Παροχέτευση εντολών στα στάδια της υλοποίησης χωρίς εντολή διακλάδωσης . 42
5.4 Παροχέτευση εντολών στα στάδια της υλοποίησης με εντολή διακλάδωσης που πραγματοποιεί άλμα 43
5.5 Παροχέτευση εντολών στα στάδια της υλοποίησης με εντολή διακλάδωσης που δεν πραγματοποιεί αλμα 43
5.6 Instruction fetch unit 45
5.7 Βασικά σήματα διάκρισης αποκωδικοποίσης RV32I 46
5.8 Τα πέδια που λαμβάνουν τιμή με βάση το Op_{type} 46
5.9 Μονάδες αποδικωποίησης κατα RV32I και αναγνωση τελούμενων εισόδου 47
5.10 Μορφή σταδίων υλοποίησης με την τοποθέτηση του flow_control 48
5.11 Forward logic unit 50
5.12 Διάγραμμα Execution unit 51
5.13 Μονάδα ALU.Αποτίμηση ALU_RESULT και BR_TAKEN με βάση την τιμή του πεδίου ALUCODE 52
5.14 Μνήμη δεδομένων και κατανομή δεδομένων για εγγραφή ή ανάγνωση σε αυτή. . 53
5.15 Write back stage 54

6.1 Διπλή έκδοση χωρίς εντολή διακλάδωσης 58
6.2 Έκδοση και εκτέλεση εντολών μετα απο εντολή διακλάδωσης με ψευδή αποτίμηση 58
6.3 Έκδοση και εκτέλεση εντολών μετά απο εντολή διακλάδωσης με αληθή αποτίμηση 59
6.4 Παράδειγμα Risc-v assembly 60
6.5 Μετατροπή σε Hex 60
6.6 Αποτελέσματα προσομοίωσης 61
6.7 Αποτελέσματα προσομοίωσης 61
6.8 Αποτελέσματα προσομοίωσης 62
6.9 Αποτελέσματα προσομοίωσης 62
6.10 Risc-v 63

6.11 Risc-v hex 63
6.12 Αποτελέσματα προσομοίωσης 64
6.13 Αποτελέσματα προσομοίωσης 64
6.14 Αποτελέσματα προσομοίωσης 65
6.15 Αποτελέσματα προσομοίωσης 66
6.16 Risc-v assembly κλασσικής αρχιτεκτονικής 66
6.17 Risc-v assembly μετά τον μετασχηματισμό 67
6.18 Αποτελέσματα προσομοίωσης 67
6.19 Αποτελέσματα προσομοίωσης 68
6.20 Αποτελέσματα προσομοίωσης 69
6.21 Risc-v assembly κλασσικής αρχιτεκτονικής και Risc-v assembly μετά τον μετασχηματισμό 69

7.1 Ψευδοκώδικας Quicksort 72
7.2 Πλήρες στιγμιότυπο εκτέλεσης αλγορίθμου 73
7.3 Χειρότερη περίπτωση εκτέλεσης 74
7.4 Καλύτερη περίπτωση εκτέλεσης 75
7.5 Μέση περίπτωση εκτέλεσης 75
7.6 Branch Misses με βάση το στοιχείο διαχωρισμού 76
7.7 Καμπύλή Branch Misses με βάση το στοιχείο διαχωρισμού 77
7.8 Χρόνος εκτέλεσης σε σχέση με το στοιχείο διαχωρισμού για διάφορες τιμές του n 78
7.9 Branch Misses σε σχέση με το στοιχείο διαχωρισμού για διάφορες τιμές του n . . 78
7.10 Εντολές που εκτελούνται σε σχέση με το στοιχείο διαχωρισμού για διάφορες τιμές του n 79
7.11 Υλοποίηση του Quicksort σε γλώσσα C 80
7.12 Υλοποίηση του Quicksort σε Assembly Riscv I 81
7.13 Υλοποίηση του Quicksort σε Assembly Riscv I 82
7.14 Υλοποίηση του Quicksort για αρχιτεκτονική διπλής έκδοσης 83
7.15 Ευκλείδια απόσταση 85

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

Το πρόβλημα της πρόβλεψης διακλαδώσεων εμφανίστηκε στην αρχιτεκτονική υπολογιστών παράλληλα με την εισάγωση αρχιτεκτονικών παροχέτευσης .Ακόμη και στις πρώιμες απλές αρχιτεκτονικές η ποινή που προέκυπτε μετά απο εντολές διακλάδωσης ήταν αρκετή ώστε να πραγματοποιηθεί σχετική ερευνητική δουλειά προκειμένου να αντιμετωπιστεί η ζημιά στην απόδοση του επεξεργαστή. Οι πρώτες προσπάθειες περιλάμβαναν απλές στατικές μεθόδους πρόβλεψης διακλαδώσεων χωρίς να εκτεμεταλεύονται καθόλου δεδομένα απο την εκτέλεση του προγράμματος. Στην συνέχεια η ένταξη των δυνάμικων μεθόδων προσέφερε μια ουσιαστικότερη αντιμετώπιση του προβλήματος . Η επανάσταση στον τομέα της πρόβλεψης διακλαδώσεων όμως έγινε με την δημιουργία του 2-επίπεδου μοντέλου πρόβλεψης διακλαδώσεων που συσχετίζει την κάθε πιθανή ιστορία μιας ή πολλών εντολών διακλάδωσης με μια πρόβλεψη.Σήμερα η ανάγκη για αποδοτικούς τρόπους επίλυσης των εντολών διακλάδωσης παραμένει επιτακτική καθώς οι σύγχρονες αρχιτεκτονικές διαχειρίζονται πόλλες εντολές ανά κύκλο σε κάθε στάδιο εκτέλεσης και τα στάδια εκτέλεσης είναι αρκετά περισσότερα στις εμπορικές υλοποιήσεις απο οτι της κλασσικής υλοποίησης των πέντε σταδίων.Από τα παράπανω προκύπτουν περισσότεροι κύκλοι καθυστέρησης μετά απο εντολές διακλάδωσης . Επίσης πολλές εντολές διακλάδωσης εξαρτόνται απο τα δεδομένα εισόδου ενός προγράμματος συνεπώς δεν είναι δυνατόν να προκύψει κάποιο μοτίβο ικανό να εκπαιδεύσει μια μηχανή πρόβλεψης.

1.1 Στόχος της εργασίας

Η εργασία αυτή έχει ως σκοπό να προτείνει μια μέθοδο που θα αντιμετωπίζει τις εντολές διακλάδωσης με ένα διαφορετικό τρόπο που απαιτεί την χρήση του μεταγλωττιστή άλλα και κατάλληλες διαφοροποιήσεις σε επίπεδο υλικού στην αρχιτεκτονική πέντε σταδίων. Η υλοποίηση που σχεδιάστηκε και θα αναλύθεί στην ιδανική περίπτωση εξαλείφει τους κύκλους καθυστέρησης μετά των εντολών διακλάδωσης χωρίς την χρήση κάποιας μεθόδου πρόβλεψης ,με τα πλεονεκτήματα που αυτό επιφέρει στην πολυπλοκότητα της υλοποίησης και συνεπώς στην κατανάλωση ενέργειας. Επιπρόσθετα η μέθοδος αυτή δεν αντιμετωπίζει πρόβλημα μετά απο εντολές με τυχαία αποτίμηση όπως όλα τα γνωστά σχήματα πρόβλεψης.

1.2 Οργάνωση του τόμου

Το 2ο Κεφάλαιο ασχολείται με τα βασικά θέματα της Αρχιτεκτονικής Υπολογιστών εστιάζοντας στις διαφορές αρχιτεκτονικών RISC και CISC και την σύνδεση τους με την απόδοση της υπολογιστικής μηχανής .
Στο 3ο Κεφάλαιο παρουσιάζεται το RISC-V ISA του πανεπιστημίου του Berkeley.
Στο 4ο Κεφάλαιο περιγράφεται η αρχιτεκτονική RISC πέντε στάδιων, το πρόβλημα μέτα απο εντολές που αλλάζουν την ροή εκτέλεσης και οι τεχνικές πρόβλεψης διακλαδώσεων που συναντόνται στην βιβλιογραφία και στις εμπορικές υλοποιήσεις.
Στο 5ο Κεφάλαιο περιγράφεται αναλυτικά η υλοποίηση που έγινε σε γλώσσα Verilog.
Στο 6ο Κεφάλαιο επαληθεύεται η ορθότητα της υλοποίσης με την παρατήσηση των αποτελεσμάτων της προσομοιωσης .
Στο 7ο Κεφάλαιο προτείνονται δύο αλγόριθμοι που θεωρήθηκαν ιδανικά παραδείγματα για την εκτέλεση τους απο τον επεξεργαστή που σχεδιάστηκε.

Κεφάλαιο 2

Αρχιτεκτονική Υπολογιστών

2.1 Γενικά

Η αρχιτεκτονική υπολογιστών, είναι το γνωστικό πεδίο της μηχανικής υπολογιστών το οποίο πραγματεύεται τον λογικό σχεδιασμό, τη δομή και τη λειτουργία του υλικού ενός υπολογιστικού συστήματος. Ως επιστημονικός τομέας εστιάζει στην έρευνα και σχεδίαση υλικού που επιτρέπει την αποδοτική εκτέλεση αλγορίθμων και υπολογισμών, με βάση τις διαθέσιμες τεχνολογίες κατασκευής ολοκληρωμένων κυκλωμάτων. Συνήθως η αρχιτεκτονική υπολογιστών δίνει έμφαση στη δομή και λειτουργία του επεξεργαστή και στους τρόπους προσπέλασής του στη μνήμη.[23],[12] Ένας υπολογιστής δομείται σε μία ιεραρχία αφηρημένων επιπέδων οργάνωσης τα οποία οικοδομούνται το ένα πάνω στο άλλο: κάθε υπερκείμενο επίπεδο αξιοποιεί το υποκείμενό του. Η τακτική αυτή ονομάζεται «δομημένη οργάνωση υπολογιστών» και επιτρέπει τη συστηματική και εύκολη ανάλυση, σχεδίαση και κατανόηση των υπολογιστικών συστημάτων. [2]

2.2 Επίπεδα Οργάνωσης

2.2.1 Αρχιτεκτονική Συνόλου Εντολών

Αρχιτεκτονική Συνόλου Εντολών (Instruction Set Architecture, ISA), είναι η λογική αφαίρεση ενός υπολογιστικού συστήματος στο επίπεδο της Γλώσσας Μηχανής,το προγραμματιστικό μοντέλο που αντιλαμβάνεται ο προγραμματιστής που προγραμματίζει σε αυτό το χαμηλότερο δυνατο επίπεδο. Περιλαμβάνει το σύνολο εντολών, τις μεθόδους διευθυνσιοδότησης (προσπέλασης μνήμης), τη διαχείριση καταχωρητών, τη κωδικοποίηση διευθύνσεων και δεδομένων, το μηχανισμό κλήσης ρουτινών, τη διαχείριση εισόδου/εξόδου, τη διαχείριση των καταστάσεων και σημάτων διακοπής του επεξεργαστή. Το ISA αποτελεί την διαχωριστική γραμμή ανάμεσα στο λειτουργικό και στο υλικό .[2]

2.2.2 Μικροαρχιτεκτονική

Μικροαρχιτεκτονική (Microarchitecture), είναι το αμέσως χαμηλότερο επίπεδο, πιο συγκεκριμένο και λεπτομερές από το επίπεδο Αρχιτεκτονικής Συνόλου Εντολών. Περιλαμβάνει τη λεπτο-

μερή περιγραφή του τρόπου σύνδεσης, λειτουργίας και χρονισμού των συστατικών μερών του υλικού, έτσι ώστε αυτά να υλοποιούν το σύνολο των εντολών. Δηλαδή τη πλήρη περιγραφή του κύκλου Ανάκλησης – Εκτέλεσης όλων των εντολών που υποστηρίζει ο υπολογιστής. Επίσης περιλαμβάνονται και θέματα Παραλληλισμού Επιπέδου Εντολής (Instruction Level Parallelism, ILP), δηλαδή αρχιτεκτονικές βελτιώσεις με στόχο την αύξηση της απόδοσης του επεξεργαστή. Ένα δεδομένο ISA μπορεί να υλοποιείται με διαφορετικές μικροαρχιτεκτονικές. Οι υλοποιήσεις αυτές μπορεί να διαφέρουν ανάλογα με τους στόχους ενός συγκεκριμένου σχεδιασμού.[2]

2.2.3 Σχεδίαση Συστήματος

Σχεδίαση Συστήματος (System Design) που περιλαμβάνει τη διασύνδεση και λειτουργία των βασικών συστατικών στοιχείων του υλικού του υπολογιστή, κυρίως εκτός του επεξεργαστή, καθώς αυτά επηρεάζουν την απόδοση του επεξεργαστή, όπως:[2]

- Ιεραρχίες μνήμης (κρυφή μνήμη, εικονική μνήμη)
- Δίαυλοι, Ρολόγια, Διακόπτες, Ελεγκτές .
- Συστήματα συν-επεξεργασίας (GPUs, DMAs, NICs)
- Παραλληλισμός σε επίπεδο Επεξεργαστών.

2.3 Αρχιτεκτονική CISC

Τα συστήματα αρχιτεκτονικής συνόλου εντολών CISC (complex instruction set computer) έχουν αρκετά μεγάλο σύνολο σύνθετων εντολών , εντολών που ολοκληρώνονται σε πολλούς κύκλους μηχανής .Κάθε εντολή cisc εκτελεί ποίκιλες λειτουργίες χαμηλού επιπέδου (φόρτωση απο την μνήμη , αριθμητική λειτουργία, αποθηκεύση) . Το αρχικό υψηλό κόστος των μνημών Ram και των αποθηκευτικών μέσων και η αργή προσπέλαση τους έκανε επιτακτική την ανάγκη για κώδικα μικρότερου μεγέθους . Απο την άλλη το σύνολο σύνθετων εντολών που μοιάζει με εντολές υψηλότερου επιπέδου κάνει πολύ εύκολη για τους προγραμματιστές την συγγραφή προγραμμάτων σε επίπεδο μηχανής και πολύ πιο απλή την υλοποίηση μεταγλωτιστών. Με την παρόδου του χρόνου ,καθώς , το κόστος των αποθηκευτικών μέσων μειώνεται οι αρχιτεκτονικές που βασιζονται σε CISC ISA τείνουν να αντικατασταθούν από απλούστερες RISC αρχιτεκτονικές , με μεγάλο πλεονέκτημα την χρήση διοχέυσης που επιτρέπουν οι τελευταίες. Η διοχετεύση επιτρέπει πολύ υψηλες ταχύτητες ρολογιού και έτσι ακόμα και οι πιο εμπορικοί x86 τρέχουν microcode, τα πολύπλοκα δηλαδή assembly instructions του x86/x86_64 ISA μεταφράζονται απ’ τον επεξεργαστή σε μικρές απλές εντολές on-the-fly και τρέχουν έτσι εσωτερικά, είναι δηλαδή στο εσωτερικό τους RISC.[7],[9]

2.4 Αρχιτεκτονική RISC

Η αρχιτεκτονική Risc χρησιμοποιεί ενα σύνολο απλών εντολών που ολοκληρώνονται σε ενα κύκλο μηχανής . Αναπτύχθηκε απο την IBM στα μεσα της δεκαετίας του 1970.Το απότελεσμα των

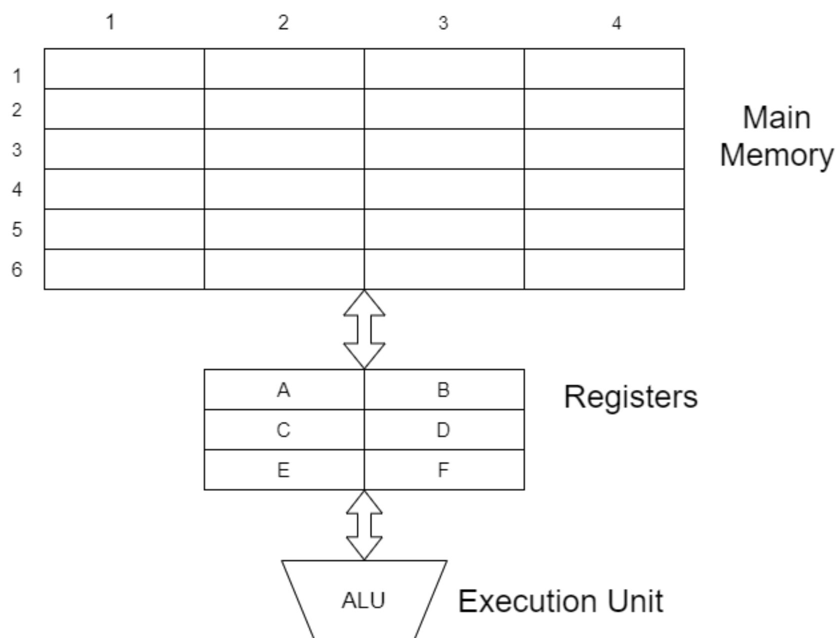
απλών εντολών και οι ξεχωριστές εντολές για πρόσβαση στην μνήμη επέτρεψαν την δημιουργία ενός απλούστερου επεξεργαστή με λιγότερα τρανζίστορ και μικρότερο κόστος κατασκευής ενώ παράλληλα μπορούσε να εκτελέσει περισσότερες εντολές στην ίδια μονάδα χρονου απο τα cisc συστήματα. Βέβαια αυτό εκάνε επιτακτικότερη την ανάγκη για πολυπλότερους compilers που θα παράγουν αποδοτικά τον κώδικα χαμηλού επιπέδου και επιπλέον για γρήγορους και μεγαλύτερης χωρητικότητας αποθηκευτικούς χώρους κοντά στην κεντρική μονάδα επεξεργασίας.(caches ,μνημες ram). Μετά τον πρώτο RISC που σχεδιάστηκε σε χρόνο ρεκόρ, ήρθε ο RISC-II (1981) βάσει του οποίου υλοποιήθηκαν οι SPARC επεξεργαστές της Sun. Μετέπειτα ερευνητικές προσπάθειες στο Berkeley, γνωστές ως SOAR (1984) και SPUR (1988), θεωρούνται η συνέχεια του RISC και ονομάστηκαν αργότερα RISC-III και RISC-IV αντίστοιχα. Το 2011 μετά από αρκετές αρχιτεκτονικές που ακολούθησαν το μοντέλο του RISC, το Berkeley και η ομάδα του επανήλθαν με τον RISC-V. [7],[9]

2.5 Παράδειγμα σύγκρισης

Ο πιο αντιπροσωπευτικός τρόπος να εξεταστούν τα πλεονεκτήματα και τα μειονεκτήματα της αρχιτεκτονικής RISC και της αρχιτεκτονικής CISC (Complex Instruction Set Computers) είναι μεσα απο ένα πραγματικό παράδειγμα, του πολλαπλασιασμού δύο αριθμών στη μνήμη. Στα δεξιά υπάρχει ένα διάγραμμα που αντιπροσωπεύει το μοντέλο μνήμης για έναν υπολογιστή. Η κύρια μνήμη διαιρείται σε θέσεις που αριθμούνται από (σειρά) 1: (στήλη) 1 έως (σειρά) 6: (στήλη) 4. Η μονάδα εκτέλεσης είναι υπεύθυνη για τη διεξαγωγή όλων των υπολογισμών. Ωστόσο, η μονάδα εκτέλεσης μπορεί να λειτουργεί μόνο σε δεδομένα που έχουν φορτωθεί σε έναν από τους έξι καταχωρητές (A, B, C, D, E ή F). Ας υποθέσουμε ότι θέλουμε να βρούμε το γινόμενο δύο αριθμών , ο ένας ειναι αποθηκευμένος στη θέση 2: 3 και ο άλλος στη θέση 5: 2 και να αποθηκεύσουμε το γινόμενο στη θέση 2:3.[6]

2.5.1 Η προσέγγιση CISC

Ο πρωταρχικός στόχος της αρχιτεκτονικής CISC είναι να ολοκληρώσει την λειτουργία σε όσο το δυνατόν λιγότερες γραμμές κώδικα. Αυτό επιτυγχάνεται με την κατασκευή του υλικού επεξεργαστή που είναι ικανό να κατανοήσει και να εκτελέσει σύνθετες εντολές . Για αυτή τη συγκεκριμένη λειτουργία , ένας επεξεργαστής CISC θα διαθέτει συγκεκριμένη εντολή που θα εκτέλει αύτη την διαδικάσια (θα την ονομάσουμε ”MULT”). Όταν εκτελείται, αυτή η εντολή φορτώνει τις δύο τιμές σε ξεχωριστούς καταχωρητές, πολλαπλασιάζει τους τελεστές στην μονάδα εκτέλεσης και στη συνέχεια αποθηκεύει το γινόμενο στον κατάλληλο καταχωρητή και μετέπειτα στην μνήμη του υπολογιστή. Έτσι, ολόκληρο το έργο πολλαπλασιασμού δύο αριθμών μπορεί να ολοκληρωθεί με μία εντολή:
MULT 2: 3, 5: 2
Η εντολή αυτή λειτουργεί απευθείας στην μνήμη του υπολογιστή,ανάγνωση και εγγραφή και δεν απαιτεί από τον προγραμματιστή να καλέσει με ξεχωριστές εντολές . Μοιάζει πολύ με μια εντολή σε μια γλώσσα υψηλότερου επιπέδου. Για παράδειγμα, αν αφήσουμε το ”a” να αντιπροσωπεύει την



Σχήμα 2.1: Μοντέλο Μνήμης-Καταχωρητών και μονάδας εκτέλεσης

τιμή 2: 3 και το ”b” να αντιπροσωπεύει την τιμή 5: 2, τότε αυτή η εντολή είναι ίδια με τη δήλωση ”a = a * b”.Ένα από τα κύρια πλεονεκτήματα αυτού του συστήματος είναι ότι ο μεταγλωττιστής πρέπει να κάνει πολύ λίγη δουλειά για να μεταφράσει εναν κώδικα υψηλού επιπέδου στην γλώσσα μηχανής. Επειδή το μήκος του κώδικα είναι σχετικά μικρό, απαιτείται ελάχιστη μνήμη RAM για την αποθήκευση των εντολών. Απαιτείται ομώς πολύπλοκο και ακριβό υλικό που να μπορεί να αναγνωρίσει όλες τις σύνθετες εντολές.

2.5.2 Η προσέγγιση RISC

Οι επεξεργαστές RISC χρησιμοποιούν μόνο απλές εντολές που μπορούν να εκτελεστούν μέσα σε ένα κύκλο ρολογιού. Έτσι, η εντολή ”MULT” που περιγράφηκε παραπάνω μπορεί να χωριστεί σε τρεις χωριστές εντολές: ”LOAD”, η οποία μετακινεί δεδομένα από την μνήμη σε ένα καταχωρητή, το ”PROD”, το οποίο βρίσκει γινόμενο δύο τελεστών που βρίσκονται μέσα στους καταχωρητές, και την STORE ”, η οποία μετακινεί δεδομένα από ένα καταχωρητή στην μνήμη. Προκειμένου να εκτελεστεί η ακριβής σειρά βημάτων που περιγράφονται στην προσέγγιση CISC, ένας προγραμματιστής θα χρειαζόταν να κωδικοποιήσει τέσσερις γραμμές κώδικα:
LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A
Αρχικά, αυτό μπορεί να φαίνεται σαν πολύ λιγότερο αποτελεσματικός τρόπος για την ολοκλήρωση της λειτουργίας. Επειδή υπάρχουν περισσότερες γραμμές κώδικα, απαιτείται περισσότερη μνήμη RAM για την αποθήκευση των εντολών επιπέδου μηχανής. Ο μεταγλωττιστής πρέπει επίσης να εκτελέσει περισσότερες εργασίες για να μετατρέψει μια γλώσσα υψηλού επιπέδου σε κώδικα αυτής

της μορφης.Ωστόσο, η στρατηγική RISC φέρνει επίσης μερικά πολύ σημαντικά πλεονεκτήματα. Επειδή κάθε εντολή απαιτεί μόνο έναν κύκλο ρολογιού για εκτέλεση, ολόκληρο το πρόγραμμα θα εκτελεστεί περίπου στο ίδιο χρονικό διάστημα με την εντολή πολλαπλών κύκλων ”MULT”. Αυτές οι απλές εντολές RISC απαιτούν λιγότερα τρανζίστορ σε επίπεδο υλικού από τις περίπλοκες εντολές, αφήνοντας περισσότερο πόρους για καταχωρητές γενικής χρήσης. Επειδή όλες οι οδηγίες εκτελούνται σε ένα ομοιόμορφο χρονικό διάστημα ένος κύκλου, είναι δυνατή η επικάλυψη μεταξυ τους . Ο διαχωρισμός των οδηγιών ”LOAD” και ”STORE” μειώνει πραγματικά την ποσότητα εργασίας που πρέπει να εκτελέσει ο υπολογιστής. Αφού εκτελεστεί μια εντολή ”MULT” τύπου CISC, ο επεξεργαστής σβήνει αυτόματα τους καταχωρητές. Εάν ένας από τους τελεστές πρέπει να χρησιμοποιηθεί για έναν άλλο υπολογισμό, ο επεξεργαστής πρέπει να φορτώσει εκ νέου τα δεδομένα από την μνήμη μνήμης σε ένα καταχωρητή. Στο RISC, ο τελεστής θα παραμείνει στον καταχωρητή και μπορεί να επαναχρησιμοποιηθεί μέχρι να φορτωθεί μια άλλη τιμή στη θέση του.

2.6 Η εξίσωση απόδοσης

Οι υπολογιστές κατασκευάζονται χρησιμοποιώντας ενα ρολόι το οποίο λειτουργεί με σταθερό ρυθμό,αυτά τα διακριτά χρονικά διαστήματα ονομάζονται κύκλοι .Συνεπώς ο χρόνος CPU των προγραμμάτων μπορει να εκφραστεί :

Χρόνος CPU = Κύκλοι ρολογιού CPU για ενα πρόγραμμα / Ρυθμός Ρολογιού

Πέρα απο το πλήθος των κύκλων ρολογιού ενός προγράμματος μπορούμε να υπολογίσουμε και το πλήθος των εντολών ,με αυτα τα δύο μεγέθη υπολογίζουμε εναν πολύ σημαντικό δείκτη απόδοσης ενος επεξεργαστή τον CPI (αριθμός κύκλων ανα εντολή)

[12] CPI = Κύκλοι ρολογιού CPU για ένα πρόγραμμα / Αριθμός εντολών

Η ακόλουθη εξίσωση χρησιμοποιείται συνήθως για την έκφραση της απόδοσης ενός υπολογιστή:

Χρόνος /Πρόγραμμα= (Χρόνος/ Κύκλος)*(Κύκλοι/Εντολή)*(Εντολές/Πρόγραμμα)

2.7 Νόμος του Amdahl

Η αύξηση της απόδοσης που μπορεί να προκύψει σε ένα υπολογιστικό σύστημα με την βελτίωση κάποιου τμηματός του μπορεί να υπολογιστεί χρησιμοποιώντας τον νόμο του Amdahl. Σύμφωνα με τον νόμο του Amdahl η βελτίωση της απόδοσης που προκύπτει απο την χρήση κάποιας ταχύτερης μεθόδου εκτέλεσης περιορίζεται απο κλάσμα του χρόνου κατά το οποίο μπορεί να χρησιμοποιηθεί αυτή μέθοδος .Ο νόμος του Amdahl ορίζει την επιτάχυνση που μπορεί να επιτευχθεί με κάποια βελτίωση .
[12]

Επιτάχυνση = Χρόνος εκτέλεσης ολόκληρης της εργασίας χωρίς βελτίωση / Χρόνος εκτέλεσης

ολόκληρης της εργασίας με βελτίωση όπου είναι εφικτό

Η επιτάχυνση εξαρτάται απο δύο βασικούς παράγοντες.

- Απο το κλάσμα του χρόνου εφαρμογής της βελτίωσης ως προς τον συνολικό χρόνο του προγράμματος . Αν για παράδειγμα μια βελτίωση εφαρμόζεται για 20 δευτερόλεπτα σε ένα πρόγραμμα 60 δευτερολέπτων τότε το κλάσμα ισούται με 2/6 . Είναι δεδομένο οτι η τιμή του κλάσματος θα είναι μικρότερη της μονάδας.
- Απο την βελτίωση που θα προκύψει μέσω της βελτιωμένης μεθόδου αν χρησιμοποιούνταν για όλο το πρόγραμμα . Αν η βελτιωμένη μέθοδος χρειάζεται 2 δευτερόλεπτα για την εκτέλεση ενός ενός τμήματος του προγράμματος στο οποίο η αρχική μέθοδος θα χρειαζότανε 5 δευτερόλεπτα τότε η βελτίωση ισούται με 5/2 . Η τιμή του κλάσματος αυτού είναι πάντα μεγαλύτερη της μονάδας .

Συνεπώς ο χρόνος εκτέλεσης του προγράμματος στον αρχικό επεξεργαστή μετα την χρήση της βελτιωμένης μεθόδου θα ισούται με το άθροισμα του χρόνου που δαπανάται για το τμήμα της εργασίας που δεν χρησιμοποιεί βελτίωση με τον χρόνο που δαπανάται για το τμήμα της εργασίας με την βελτιωμένη μέθοδο .[12]

Χρόνος εκτέλεσης νέος = Χρόνος εκτέλεσης παλαιός * ((1-κλάσμα βελτιωμένο) + (κλάσμα βελτιωμένο /επιτάχυνση βελτιωμένη))

Επιτάχυνση συνολίκη = Χρόνος εκτέλεσης νέος / Χρόνος εκτέλεσης παλαίος

Κεφάλαιο 3

RISC-V ISA

3.1 Γενικά

Το RISC-V (ISA) είναι μια αρχιτεκτονική συνόλου εντολών RISC που σχεδιάστηκε για να υποστηρίξει την έρευνα και την εκπαίδευση στον τομέα της και τείνει να γίνει μια ανοικτή αρχιτεκτονική για εφαρμογές της βιομηχανίας.

- Αποτελεί ενα πλήρως ανοικτό ISA που είναι ελεύθερα διαθέσιμο στον ακαδημαϊκό κόσμο και στη βιομηχανία.
- Είναι κατάλληλο για άμεση υλοποίηση υλικού, όχι μόνο προσομοίωση .
- Είναι καταληλο για ένα διαφορετικές μικροαρχιτεκτονικές (π.χ. in order,out of order) και μπορεί να υλοποιηθεί σε FPGA και ASIC.
- Περιλαμβάνει σε ένα βασικο σύνολο εντολών ακεραιών, και προαιρετικές τυποποιημένες επεκτάσεις, για την υποστήριξη της ανάπτυξης λογισμικού γενικής χρήσης.
- Υποστήριξη για το αναθεωρημένο πρότυπο κινητής υποδιαστολής IEEE-754 του 2008 .
- Τόσο παραλλαγές διαστήματος διευθύνσεων 32-bit όσο και 64-bit για εφαρμογές, πυρήνες λειτουργικού συστήματος και υλοποιήσεις υλικού.
- Υποστήριξη για παράλληλες αρχιτεκτονικές, συμπεριλαμβανομένων των ετερογενών πολυεπεξεργαστών.
- Προαιρετικές εντολές μεταβλητού μήκους τόσο για την επέκταση του διαθέσιμου χώρου κωδικοποίησης εντολών όσο και για την κωδικοποίηση πυκνών οδηγιών για βελτιωμένη απόδοση.
- Είναι πλήρως εικονικοποιημένο ISA για να διευκολύνει την ανάπτυξη του hypervisor.

Οι standard επεκτάσεις που έχουμε μέχρι στιγμής είναι:

- I(ntegers): Βασικές πράξεις με ακέραιους (and,or,xor,add,sub,load,store κλπ) και control flow (jumps κλπ).

- M(ultiplication): Πολλαπλασιασμός και διαίρεση για ακέραιους.

- A(tomics): Διαδικασίες που αφορούν εντολές load/store στη μνήμη κατά τρόπο που να είναι "σύγχρονες" μεταξύ των διαφορετικών πυρήνων/threads.

- F(loats): Πράξεις με δεκαδικούς 32bit - single precision (add/sub/multiply/divide/compare κλπ).

- D(ouble): Πράξεις με δεκαδικούς 64bit - double precision.

- Q(uad): Πράξεις με δεκαδικούς 128bit - quad precision.

- C(ompressed): Υποστήριξη για συμπιεσμένες εντολές με σκοπό την ελάττωση του μεγέθους παραγόμενου κώδικα (-16bit- εκδοχή της -με κάποιες συμβάσεις όπως το ποιοι καταχωρητές χρησιμοποιούνται-).

Οι standard επεκτάσεις που είναι υπό ανάπτυξη αφορούν λειτουργίες που θα βρίσκονται στους περισσότερους επεξεργαστές όπως bit manipulation (B) και vectors (V) και πιο ενδιαφέρουσες ιδέες όπως acceleration των JIT γλωσσών (J) . Για να τρέξει ένας επεξεργαστής ένα πλήρες λειτουργικό χρειάζεται να υλοποιεί κατ' ελάχιστον τις επεκτάσεις IMAFD, καθώς και το privilege spec, όπου ορίζει τον διαχωρισμό των διαφόρων επιπέδων προστασίας στον επεξεργαστή, τη διαχείριση της εικονικής μνήμης κλπ. Το σετ αυτό το λέμε πιο σύντομα G(eneric), οπότε αν δείτε έναν επεξεργαστή να λέει πως είναι rv64gc π.χ. σημαίνει πως υλοποιεί τις επεκτάσεις IMAFDC.[27],[28]

3.2 Κωδικοποίηση Μήκος Εντολής

Το βασικό RISC-V ISA έχει 32-bit εντολες σταθερού μήκους. Ωστόσο, το πρότυπο σύστημα κωδικοποίησης RISC-V έχει σχεδιαστεί για να υποστηρίζει εντολές μεταβλητού μήκους. Όλες οι 32-bit εντολές έχουν τα χαμηλότερα δύο δυαδικά ψηφία τους ρυθμισμένα στο 11. Οι συμπιεσμένες 16-bit εντολές έχουν τα χαμηλότερα δύο δυαδικά ψηφία τους ίσα με 00, 01 ή 10. Οι τυποποιημένες επεκτάσεις εντολών που κωδικοποιούνται με περισσότερα από 32 bits έχουν επιπλέον ψηφία χαμηλής τάξης που έχουν οριστεί σε 1, για μήκη 48 και 64 bit . Εντολές μεταξύ 80 bit και 176 bit κωδικοποιούνται χρησιμοποιώντας ένα πεδίο των 3-bit στις θέσεις [14:12] δίνοντας το πλήθος των επιπλεον 16-bit λέξεων εκτός από τις πρώτες λέξεις 5 × 16-bit. Η κωδικοποίηση με bits [14:12] στο 111 είναι αποκλειστικά για μελλοντική χρήση. Το σχήμα 1.1 απεικονίζει την τυπική σύμβαση κωδικοποίησης μήκους εντολής RISC-V.[27],[28]

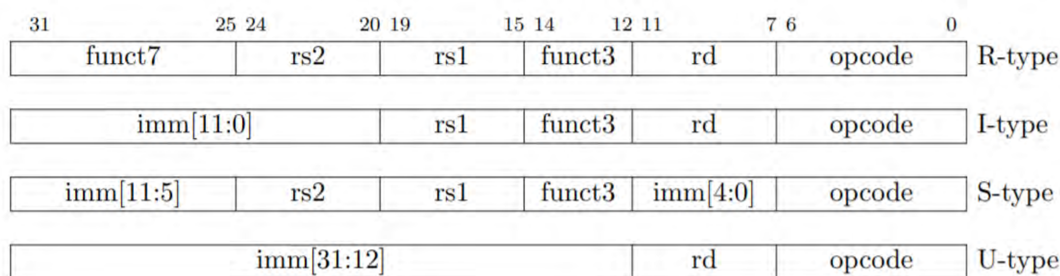
Σχήμα 3.1: Κωδικοποίηση Εντολής

3.3 Φάκελος καταχωρητων

Υπάρχουν 31 γενικού σκοπού καταχωρητές οι x1-x31, οι οποίοι κατέχουν ακέραιες τιμές,με τον καταχωρητή x0 είναι σταθερός 0.Η τυπική σύμβαση κλήσης λογισμικού χρησιμοποιεί τον καταχωρητή x1 για να κρατήσει τη διεύθυνση επιστροφής σε μια κλήση. Για το RV32, οι καταχωρητές x έχουν πλάτος 32 bits,και για το RV64, έχουν πλάτος 64 bits. Υπάρχει ένας προσθετός καταχωρητής ο μετρητής προγράμματος pc κρατά τη διεύθυνση της υπο εκτέλεση εντολής. Ο αριθμός των διαθέσιμων καταχωρητών μπορεί να έχει μεγάλες επιπτώσεις στο μέγεθος, την απόδοση,και την κατανάλωση ενέργειας. Ένας μεγαλύτερος αριθμός καταχωρητών ακέραιων αριθμών θα βοηθούσε την απόδοση σε κώδικα υψηλής απόδοσης, όπου μπορεί να υπάρχει εκτεταμένη χρήση ξετυλίγματος βρόχου, software pipelining , και loop nest optimization. Η υλοποιήση του φακέλου καταχωρητών στο υλικό μπορεί να βελτιστοποιηθεί ώστε να μειωθεί η ενέργεια πρόσβασης για τους καταχωρητές με μεγαλυτερη συχνότητα πρόσβασης.Για ενσωματωμένες εφαρμογές με περιορισμένη χρήση πόρων, εχει οριστεί το υποσύνολο RV32E, το οποίο έχει μόνο 16 καταχωρητές. [27],[28]

3.4 Βασικές κωδικοποιήσεις εντολών

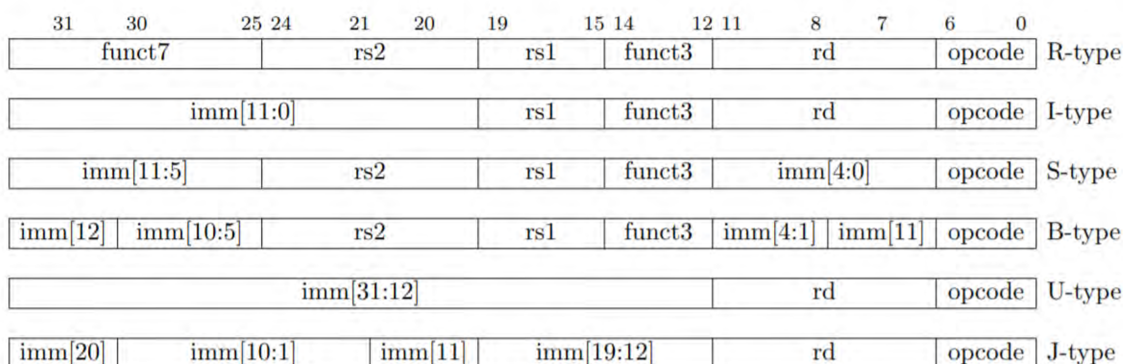
Στη βάση ISA, υπάρχουν τέσσερις βασικές μορφές εντολών (R-type , I-type, S-type , U-type). Όλες είναι σταθερόυ μήκος 32 bit και πρέπει να ευθυγραμμίζονται σε ένα όριο τεσσάρων bytes στη μνήμη. Μια εξαίρεση μη έγκαιρης διευθυνσιοδότητησης δημιουργείται κάθε φορά που μια εντολή διακλάδωσης η μια εντολή άλματος επιθυμεί πρόσβαση σε μη έγκυρη διεύθυνση.



Σχήμα 3.2: Βασικές κωδικοποιήσεις εντολών

Το RISC-V ISA διατηρεί τους καταχωρητές πηγής (rs1 και rs2) και προορισμού (rd) στην ίδια θέση σε όλες τις μορφές για την απλούστευση της αποκωδικοποίησης επειδή η αποκωδικοποίηση της διεύθυνσης των καταχωρητών και η ανάκτηση τιμής απο το φάκελο καταχωτητών παίζει καίριο ρόλο στην διαμόρφωση του κρίσιμου μονοπατιού.Εκτός από τις 5-bit σταθερές που χρησιμοποιούνται στις εντολές CSR, οι σταθερές τιμές είναι πάντα sign-extended και καταλαμβάνουν τα αριστερότερα διαθέσιμα bit της εντολής για να μειώσουν την πολυπλοκότητα του υλικού, το bit προσήμου είναι πάντα στην θέση 31 για αποδοτικότερη υλοποιήση.

Υπάρχουν δύο επιπλέον παραλλαγές των μορφών εντολών με βάση το χειρισμό των σταθερών η B-type και J-type. Η μόνη διαφορά μεταξύ των μορφών S και B είναι ότι χρησιμοποιείται η 12-bit τιμή της σταθεράς για να κωδικοποιούνται οι διευθύνσεις των εντολών διακλάδωσης σε πολλαπλάσια του 2,με το imm[0] παντα 0 .Ομοίως, η μόνη διαφορά μεταξύ των μορφών U και J είναι ότι η σταθερα 20-bit μετατοπίζεται αριστερά κατά 12 bits στην U μορφη και με 1 bit στην J μορφη.[27],[28]



Σχήμα 3.3: B-type J-type

3.5 Εντολές ακεραίων

Οι υπολογιστικές εντολές μεταξύ ακεραίων λαμβάνουν ορίσματα των 32 bits εκτελούν την αριθμητική πράξη και αποθηκεύουν το αποτέλεσμα στον καταχωρητή προορισμού rd, επίσης 32

bit . Οι εντολές αυτές ανάλογα με τα ορίσματα που λαμβάνουν είτε κωδικοποιούνται ως καταχωρήτη- καταχωρήτη (rs1-rs2) με μορφή τύπου R είτε ως καταχωρητή-σταθεράς(rs1-imm) με μορφή τύπου I. Ο προορισμός είναι κοινός, ο καταχώρητης rd . Δεν υπάρχουν ειδικές εντολές που να αντιμετωπίζουνε πιθανή υπερχείλιση σε πράξεις μεταξύ ακεραίων αλλά αυτό μπορεί να επιλυθεί με την εφαρμογή εντολών διακλάδωσης.Πιο συγκεκριμένα ο έλεγχος υπερχείλισης για την πρόσθεση μη προσημασμένων αριθμών απαιτεί μόνο μια επιπλέων εντολή διακλαδωσης
add t0, t1, t2 ; bltu t0, t1, overflow
Για την προσημασμένη προσθήκη, εάν είναι γνωστό το πρόσημο του ενός τελεστή, ο έλεγχος υπερχείσης καλύπτεται πάλι με μια εντολή διακλάδωσης .
addi t0, t1, +imm ; blt t0, t1, overflow
Για την γενική περίπτωση πρόσθεσης δύο προσημασμένων αριθμών χρειαζόμαστε τρεις πρόσθετες εντολές.
add t0, t1, t2 ;
slti t3, t2, 0 ;
slt t4, t0, t1 ;
bne t3, t4, overflow
Σε αυτή την περίπτωση αξιοποιούμε την παρατήρηση ότι το άθροισμα θα πρέπει να είναι μικρότερο από έναν από τους τελεστές αν και μόνο εάν ο άλλος τελεστής είναι αρνητικός.[27],[28]

3.5.1 Πράξεις ακεραίων μεταξύ καταχωρητή και στάθερας

31–20	19–15	14–12	11–7	6–0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

Σχήμα 3.4: Πράξεις Καταχωρητή-στεθεράς

Η ADDI προσθέτει άμεσα την προσημασμένη επέκταση της 12-bit σταθεράς στον καταχώρητη του rs1. Η ADDI rd, rs1, 0 χρησιμοποιείται για την υλοποίηση της MV rd,rs1 , ψευδοεντολής assembly. Η SLTI τοποθετεί την τιμή 1 στο καταχωτητή rd αν η προσημασμένη τιμή του rs1 είναι μικρότερη από την sign-extended σταθερά , αλλιώς 0 γράφεται στο rd. Η SLTIU είναι ακριβως οπως το SLTI αλλα με μη προσημασμένους αριθμούς.Οι ANDI, ORI, XORI είναι λογικές πράξεις που εκτελούν το λογικό AND, OR, και XOR στο καταχωρητή rs1 με την προσημασμένη επέκταση της 12bit σταθερας.[27],[28]

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

Σχήμα 3.5: Πράξεις Καταχωρητή-στεθεράς

Οι μετατοπίσεις με μια σταθερά κωδικοποιούνται σαν εντολές της μορφής τύπου I. Ο τελεστής που πρέπει να μετατοπιστεί είναι ο rs1 και η ποσότητα μετατόπισης κωδικοποιείται στα κατώτερα 5 bits της σταθερας .
Η SLLI είναι μια λογική αριστερή μετατόπιση.
Η SRLI είναι μια λογική δεξιά μετατόπιση .
Η SRAI είναι μια αριθμητική δεξια μετατόπιση.

31 12	11 7	6 0
imm[31:12]	rd	opcode
20	5	7
U-immediate[31:12]	dest	LUI
U-immediate[31:12]	dest	AUIPC

Σχήμα 3.6: Πράξεις Καταχωρητή-σταθεράς

Η LUI χρησιμοποιείται για τη δημιουργία σταθερών 32 bit και χρησιμοποιεί τη μορφή τύπου U.Η LUI τοποθετεί την σταθερή τιμή στα κορυφαία 20 bits του καταχωτηρητή rd, συμπληρώνοντας τα χαμηλότερα 12 bits με μηδενικά. Η AUIPC χρησιμοποιείται για τη δημιουργία διευθύνσεων σχετικών με τον μετρητή προγράμματος pc και χρησιμοποιεί την U μορφή. Λαμβανει 20-bit σταθεράς , γεμίζοντας με τα χαμηλότερα 12 bits με το μηδέν, προσθέτει αυτή την τιμή στον μετρητή προγράμματος και αποθηκευεί την τιμή στο Rd .[27],[28]

3.5.2 Πράξεις ακεραίων μεταξύ καταχωρητών

Το RV32I ορίζει διάφορες αριθμητικές πράξεις τύπου R. Όλες οι εντολές διαβάζουν τους καταχωρητές rs1 και rs2 ως καταχωρητές εισόδου και γράφουν το αποτέλεσμα στο καταχωρητή rd. Τα πεδία funct7 και funct3 επιλέγουν την ακριβή πραξη.

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

Σχήμα 3.7: Πράξεις μεταξύ καταχωρητών

Η ADD και η SUB για πρόσθεση και αφαίρεση αντίστοιχα. Οι υπερχειλίσεις αγνοούνται και τα λιγοτέρο σημαντικά bits XLEN των αποτελεσμάτων γράφονται στον καταχωρητή προορισμού. Η SLT και η SLTU εκτελούν προσημασμένη και μη προσημασμένη σύγκριση αντίστοιχα, γράφοντας 1 στον rd εάν rs1 <rs2, 0 αλλιώς. Οι AND, OR,και XOR εκτελούν τις αντίστοιχες δυαδικές λογικές πράξεις. Οι SLL, SRL και SRA εκτελούν λογική αριστερά, λογική δεξιά και αριθμητική δεξιά μετατοπίση στην τιμή καταχωρητη rs1 και το ποσό μετατόπισης ορίζεται απο 5 κατώτερα bits του καταχωρητή rs2.[27],[28]

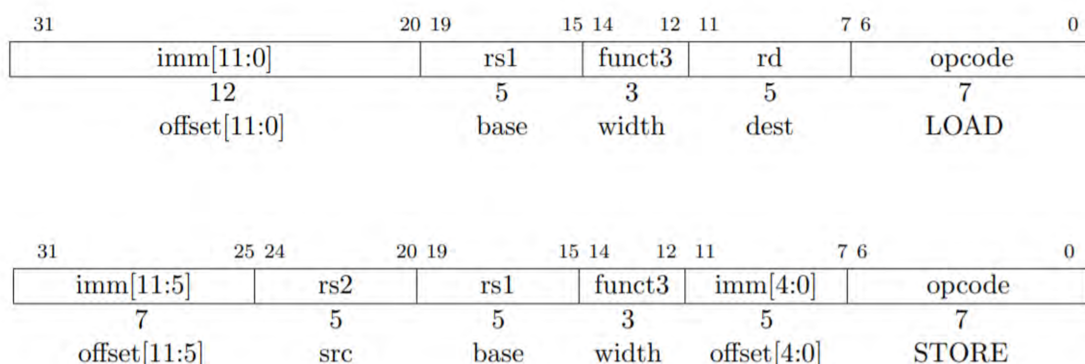
NOP Instruction

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
0	0	ADDI	0	OP-IMM

Σχήμα 3.8: Πράξεις μεταξύ καταχωρητών

3.5.3 Εντολές φόρτωσης και αποθήκευσης

Το RV32I είναι μια αρχιτεκτονική συνόλου εντολών όπου μόνο οι εντολές φόρτωσης και αποθήκευσης έχουν πρόσβαση στη μνήμη.



Σχήμα 3.9: Εντολές φόρτωσης και αποθήκευσης

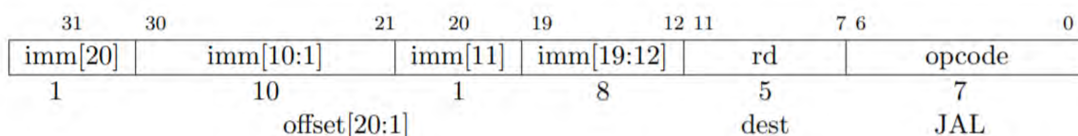
Οι εντολές φόρτωσης και αποθήκευσης μεταφέρουν μια τιμή μεταξύ των καταχωρητών και της μνήμης. Οι εντολές φόρτωσης κωδικοποιούνται στη μορφή τύπου I και οι εντολές αποθήκευσης στην μορφή τύπου S. Η διεύθυνση προσπέλασης αποκτάται με την προσθήκη στον rs1 του sign-extended 12bit offset . Οι εντολές φόρτωσης αντιγράφουν μια τιμή από τη μνήμη στον rd. Οι εντολές αποθήκευσης αντίγραφουν την τιμή του rs2 στη μνήμη. Η εντολή LW φορτώνει μια τιμή 32-bit από τη μνήμη σε rd. Το LH φορτώνει μια τιμή 16 bit από τη μνήμη, και εκτείνεται προσημασμενα σε 32-bit πριν από την αποθήκευση σε rd. Το LHU φορτώνει μια τιμή 16-bit από τη μνήμη, αλλά στη συνέχεια εκτείνεται με 0 σε 32-bit πριν την αποθήκευτει στον rd. Τα LB και LBU ορίζονται ανάλογα με τιμές 8-bit. Οι εντολές SW, SH και SB αποθηκεύουν τιμές 32 bit, 16 bit και 8 bit από τα λιγότερο σημαντικά bit του καταχωρητή rs2 στη μνήμη.

3.5.4 Εντολές ελέγχου ροής προγράμματος

Το RV32I παρέχει δύο τύπους εντολών ελέγχου ροής, εντολές δικλάδωσης χωρίς συνθήκη και εντολές διακλάδωσης υπό συνθήκη.[27],[28]

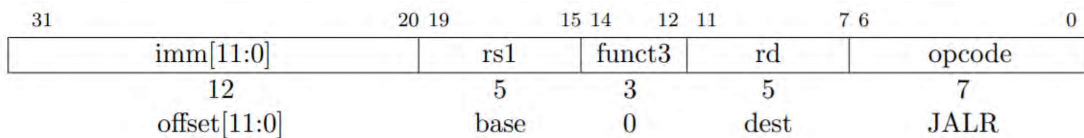
- Εντολές δικλάδωσης χωρίς συνθήκη .

  Η εντολή άλματος και σύνδεσης (JAL) χρησιμοποιεί τη μορφή τύπου J.Η τιμή της σταθεράς επεκτείνεται προσημασμένα σε τιμή 32 Bit και προστίθεται στην τιμή του pc ,ετσι δημιουργείται η διεύθυνση του άλματος. Το JAL αποθηκεύει τη διεύθυνση της εντολής μετά από το άλμα (pc + 4) στο καταχωρητή rd.



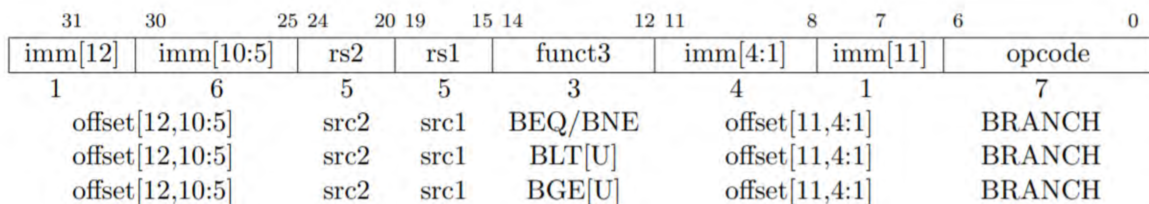
Σχήμα 3.10: Η εντολή JAL

Η έμμεση εντολή άλματος JALR χρησιμοποιεί την κωδικοποίηση τύπου I. Η διεύθυνση προορισμού επιτυγχάνεται με την προσθήκη της προσημασμένης σταθεράς 12-bit στον καταχωρητή rs1, και στη συνέχεια τον μηδενισμό των λιγότερων σημαντικών ψηφίων του αποτέλεσματος . Το JALR αποθηκεύει τη διεύθυνση της εντολής μετά από το άλμα (pc + 4) στο καταχωρητή rd.



Σχήμα 3.11: Η εντολή JALR

- Εντολές διακλάδωσης με συνθήκη
  Όλες οι εντολές διακλάδωσης με συνθήκη χρησιμοποιούν τη μορφή εντολών τύπου B,και συγκρίνουν δύο καταχωρητές.



Σχήμα 3.12: Οι εντολές διακλάδωσεις με συνθήκη

Η BEQ εκτελεί αλμά οταν οι τιμες των rs1 και rs2 είναι ίσες ενώ η BNE όταν είναι άνισες .Οι BLT και BLTU εκτελούν το αλμα αν ο rs1 είναι μικρότερος από rs2, χρησιμοποιώντας προσημασμένη και προσημασμένη σύγκριση αντίστοιχα. Η BGE και το BGEU εκτελούν το άλμα αν ο rs1 είναι μεγαλύτερος από rs2, χρησιμοποιώντας προσημασμένη και προσημασμένη σύγκριση αντίστοιχα.. Οι εντολές BGT, BGTU,BLE και BLEU μπορούν να συντεθούν αντιστρέφοντας τους τελεστές απο BLT, BLTU, BGE και BGEU αντίστοιχα.[27],[28]

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000 pred	succ	00000	000	00000	0001111	FENCE
0000 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

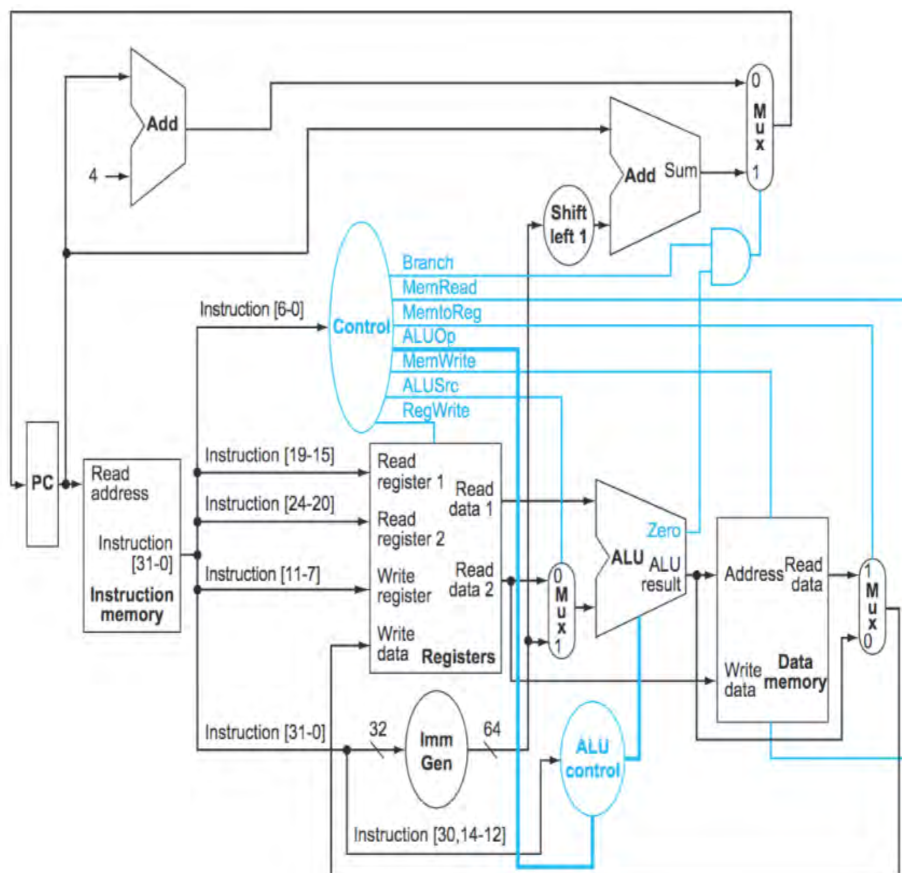
Σχήμα 3.13: RV32I Σύνολο Εντολών

Κεφάλαιο 4

Classic RISC pipeline

4.1 Single Cycle Επεξεργαστής

Ένας single cycle επεξεργαστής ολοκληρώνει την εκτέλεση κάθε εντολής σε εναν κύκλο μηχανής ,μέσω μιας ακολουθίας διαδικασιών. Περιληπτικά , την προσκόμιση απο την κύρια μνήμη εντολών (RAMI) , την αποκωδικοποίηση των τελούμενων εισόδου εξόδου και της βασίκης λειτουργίας , την εκτέλεση σε μια μονάδα επεξεργασίας ,την προσπέλαση της μνήμης δεδομένων (RAMD) αν απαιτείται και την εγγραφή των αποτέλεσμάτων στην μνήμη ή στο φακελό καταχωρητών.Η τεχνική αυτη όμως δημιουργεί μεγαλο κρίσιμο μονοπάτι ,καθως αυτό καθορίζεται από την εντόλη που χρησιμοποιεί τις περισσότερες απο τις παραπάνω διαδικασίες, στην περίπτωση του RiscV συνόλου εντολών αυτές οι εντολές είναι οι Load.Συνεπώς σε έναν sigle cycle επεξεργαστή παρόλο που έχουμε θεωρητικά βέλτιστο CPI ισο με 1, η ταχύτητα ρολογιού ειναι περιορισμένη πολύ χαμηλά .[12],[11]



Σχήμα 4.1: Risc-v RV32I simple datapath

4.2 Pipelined Επεξεργαστής

Η παροχεύτευση είναι μια τεχνική υλοποίησης σε επίπεδο αρχιτεκτονικής ,σύμφωνα με την οποία η εκτέλεση διαδοχικών εντολών γίνεται με επικάλυψη .Σκοπός της τεχνικής είναι η εκτέλεση διαδικασίων σε κάθε στάδιο που μπορούν να ολοκληρωθούν σε μικρό κύκλο ρολογιού . Στην βάσικη αρχιτεκτονική παροχέτευσης η εκτέλεση της κάθε εντολής ολοκληρώνεται σε πέντε στάδια τα οποία επικοινωνούν μεταξύ τους μέσω των καταχωρητών παροχέτευσης , αυτό επιτρέπει την πλήρωση των πέντε σταδίων με διαφορετικές εντολές οι οποίες προχωρούν σειριακά μεταφέροντας τα δεδομένα κάθε σταδίου στο επόμενο και φτάνουν στο τελικό στάδιο με ρυθμό κοντά στο 1 στην ιδανική περίπτωση. Επομένως η ικανότητα διαικπεραίωσης (throuput) του επεξεργαστή πλησιάζει την μονάδα πετυχαίνοντας παράλληλα χαμηλό κύκλο ρολογιού.[12],[**?**],[19]

	1 cc	2 cc	3 cc	4 cc	5 cc	6 cc	7 cc	8 cc	9 cc
LW $8, 0 ($9)	IF	ID	EX	MEM	WB				
ORI $16, $17, 28		IF	ID	EX	MEM	WB			
ADD $14, $8, $9			IF	ID	EX	MEM	WB		
SW $21, 88 ($3)				IF	ID	EX	MEM	WB	
SUB $21, $21, $14					IF	ID	EX	MEM	WB

Σχήμα 4.2: Παροχεύτευση εντολών

Τα σταδια της παροχεύτευσης:

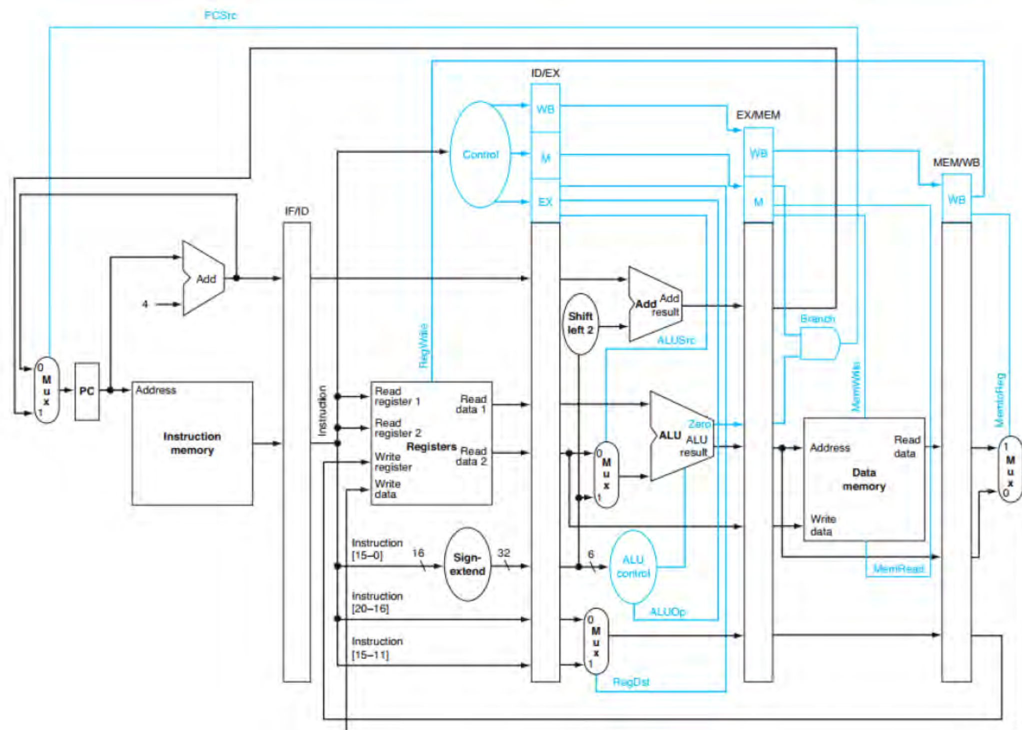
1. Instruction Fetch: Σε αυτό το στάδιο η εντολή διαβάζεται απο την μνήμη εντολών και το pc αυξάνεται κατά 4 ώστε να δείχνει στην επόμενη θέση μνήμης.Είναι το στάδιο πριν την αποκωδικοποιήση και είναι κοινό για όλες τις εντολές .

2. Instruction Decode και Register Read : Σε αυτό το στάδιο η εντολή αποκωδικοποιείται σύμφωνα με το πρότυπο του συνόλου εντολών , τα σημάτα ελέγχου λαμβάνουν τίμη καθορίζονται οι καταχωρητές πήγης και προορισμού και διαβάζονται απο το φάκελο καταχωρητών οι τιμές των καταχωρητών πηγής. Επίσης κοινό σταδιο για όλες τις εντολές.

3. Execution : Εκτελούνται οι λογικές και αριθμητικές πραξεις στην ALU μεταξύ των τελούμενων εισόδου , υπολογίζονται οι διευθύνσεις άλματος και αποτιμάται η συνθήκη διακλάδωσης και υπολογίζεται η διευθυνση προσπέλασης μνήμης ανάλογα τον τύπο εντολής .

4. Data Memory access : Σε περίπτωση εντολής προσπέλασης μνήμης Load/Store διαβάζονται τα δεδομένα απο την μνήμη η αποθηκεύονται σε αυτήν .Οι υπόλοιπες εντολές περνούν το στάδιο αυτό χωρίς να εκτελείται κάποια λειτουργία .

5. Write Back Register write: Οι εντολές που γράφουνε σε καταχωρητή χρησιμοποιούν το στάδιο αυτό για να προσπελάσουν το φάκελο καταχωρητών και να αποθηκέυσουν το αποτέλεσμα.[12],[**?**]

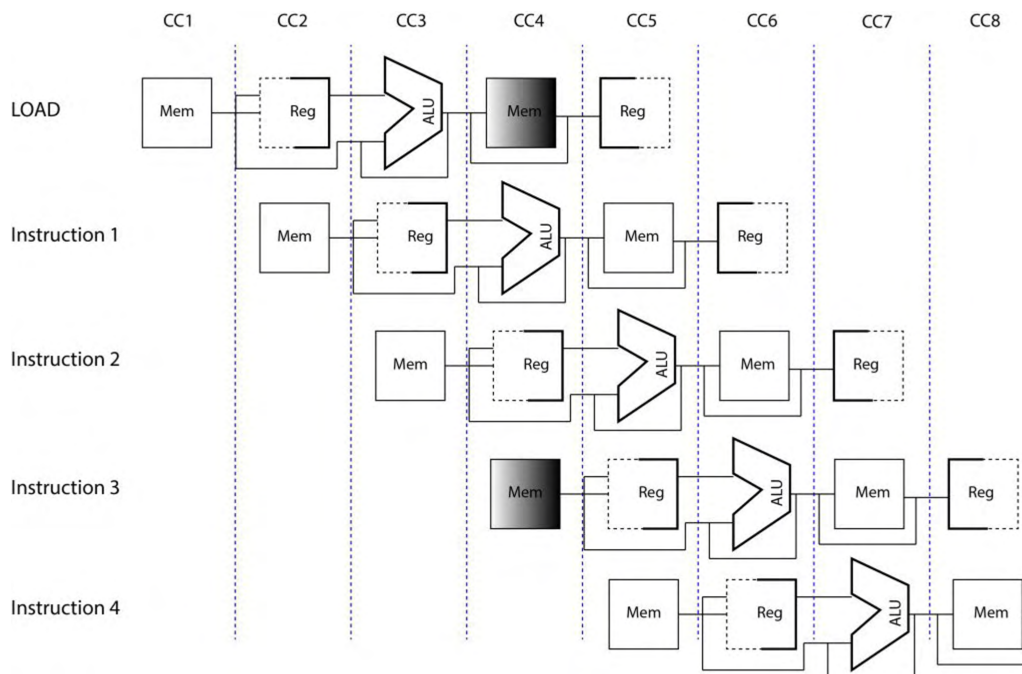


Σχήμα 4.3: Pipelined RV32I proseccor

4.3 Κίνδυνοι (Hazards)

Παρά το γεγονός οτι οι εντόλες εκτελούνται σε σειρά περνώντας διαδοχικά από τα στάδια του pipeline υπάρχουν περιπτώσεις που καποία εντολή δεν μπορεί να προχωρήσει στην εκτέλεση καθώς χρειάζεται δεδομένα που παράγουν προηγούμενες εντολές και τα οποία δεν ειναι ακόμα αποθηκευμένα στην κατάλληλη θέση μνήμης ή στο φάκελο καταχωρητών.Ακόμα μια εντολή χρειάζεται να χρησιμόποιησει κάποια δομική μονάδα που βρίσκεται σε χρήση απο άλλη εντολή στον ίδιο κύκλο. Τέλος ύπαρχουν περιπτώσεις που μια εντολή άλματος ή διακλάδωσης θα αλλάξει την ροή εκτέλεσης όταν θα φτάσει στο στάδιο εκτέλεσης και θα οδηγήσει σε λανθασμένη προσκόμιση εντολής σε πρώτο χρόνο . Κάθε ενάς απο τους παραπάνω κινδύνους που περιγράφηκαν περιλήπτικα απότελει και μια διαφορετική κατηγορία και επιδέχεται τρόπους αντιμετώπισης .

1. Structural Hazards : Όταν δύο εντολές απαιτούν την ίδια δομική μονάδα στον ίδιο κύκλο ρολογιού , απότελει συχνό κίνδυνο των υπερβαθμωτών υλοποιήσεων και επιλύεται είτε με τεχνικές δρομολόγησης σε επιπέδο λογισμικού ή υλικόυ είτε με προσθήκη επιπλέον υλικού. Συναντάται επίσης σε υλοποιήσεις με κοινή μνήμη δεδομένων και εντολών.



Σχήμα 4.4: Δομική εξάρτηση μνήμης

2. Data Hazards : Διακρίνονται σε τρείς επιμέρους κατηγορίες

    - Ανάγνωση μετά απο εγγραφή (RAW) όταν μια εντολή χρειάζεται δεδομένα που παράγει μια εντολή που βρίσκεται ακόμα υπο εκτέλεση.Επιλύεται με προσθήκη καταχωρητών που προωθούν τα τελούμενα μεταξύ των σταδίων απο την στιγμή της δημιουργίας ,η μοναδική περιπτωση που ο κινδυνός δεν μπορει να αντιμετωπιστεί και ειναι απαραίτητο το pipeline να παγώσει για ένα κύκλο ειναι μια εντολή ζήτα την τιμή του καταχωρητή που γράφεται απο την ακριβως προηγούμενη εντολή Load.Αυτό ισχύει με την παραδοχή οτι κάθε εντολή χρησιμοποιεί το κάθε στάδιο για εναν κύκλο ρολογιού και ολοκληρώνεται μετά απο πέντε κύκλους επίσης οι εντόλες εκτελούνται με την σειρά που διαβάζονται απο την μνήμη .

      Στις εμπορικές υλοποιήσεις που προστιθόνται μονάδες πολλαπλασιασμού, διαίρεσης,κινητής υποδιαστολής κάθε εντολή απαιτεί διαφορετικούς κύκλους εκτέλεσης και προσφέρεται η δυνατότητα για εκτέλεση εκτός σειρας . Τότε περιπλεκεται η επίλυση των RAW εξαρτήσεων και δημιουργούνται δύο νέοι τύποι εξαρτήσεων.[4],[5]

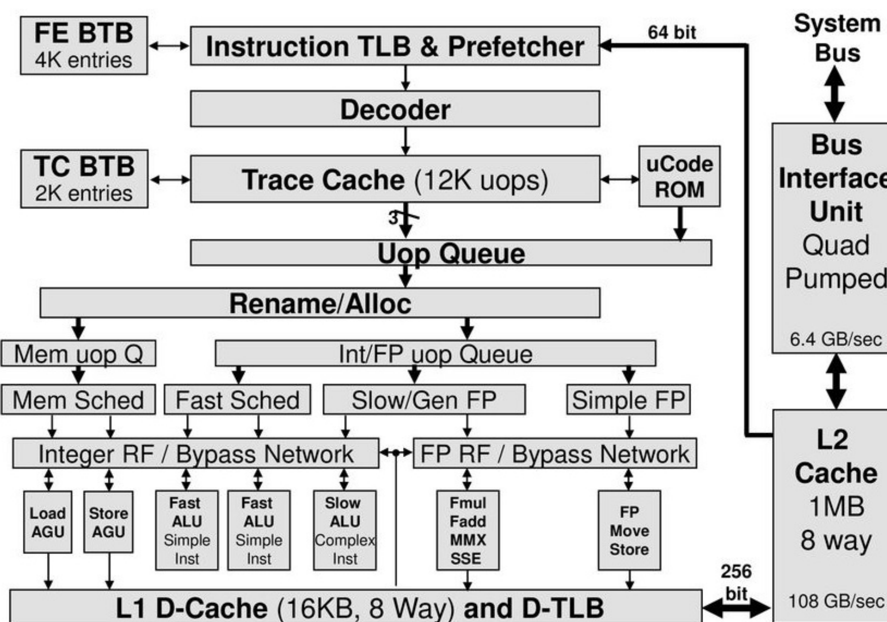
    - Γράψιμο μετά απο ανάγνωση (WAR). Μια εντόλη μεταγενέστερη που ομως εχει δρομολόγηθει νωρίτερα γράφει σε μια θέση μνήμης ή σε έναν καταχωρητή που θα διαβάσει αργότερα μια εντολή που προηγείται σε σειρά εκτελέσης .

    - Γράψιμο μετά απο Γράψιμο (WAW) Μια εντόλη μεταγενέστερη που ομως εχει δρομολογηθεί νωρίτερα γράφει σε μια θέση μνήμης ή σε έναν καταχωρητή που θα γράψει αργότερα μια εντολή που προηγείται σε σειρά εκτελέσης .Δεν θα αποτελεί σκοπό της

εργασίας η ανάλυση των εξαρτήσεων δεδομένων ούτε η μελέτη πολύπλοκων υλοποιήσεων, αλλά κρίθηκε σκόπιμο να αναφερθούν όλα τα είδη.

	1 cc	2 cc	3 cc	4 cc	5 cc	6 cc	7 cc
ADD \$8, \$7, \$3	IF	ID	EX	MEM	WB		
MUL \$16, \$8, \$9		IF	ID	EX	MEM	WB	
SLL \$14, \$8, \$9	IF	ID	EX	MEM	WB		
LW \$20, 8 (\$14)		IF	ID	EX	MEM	WB	
LW \$20, 21388 (\$9)	IF	ID	EX	MEM	WB		
NOP							
ADD \$21, \$13, \$20			IF	ID	EX	MEM	WB
LW \$20, 21388 (\$9)	IF	ID	EX	MEM	WB		
SW \$20, 1048(\$7)		IF	ID	EX	MEM	WB	

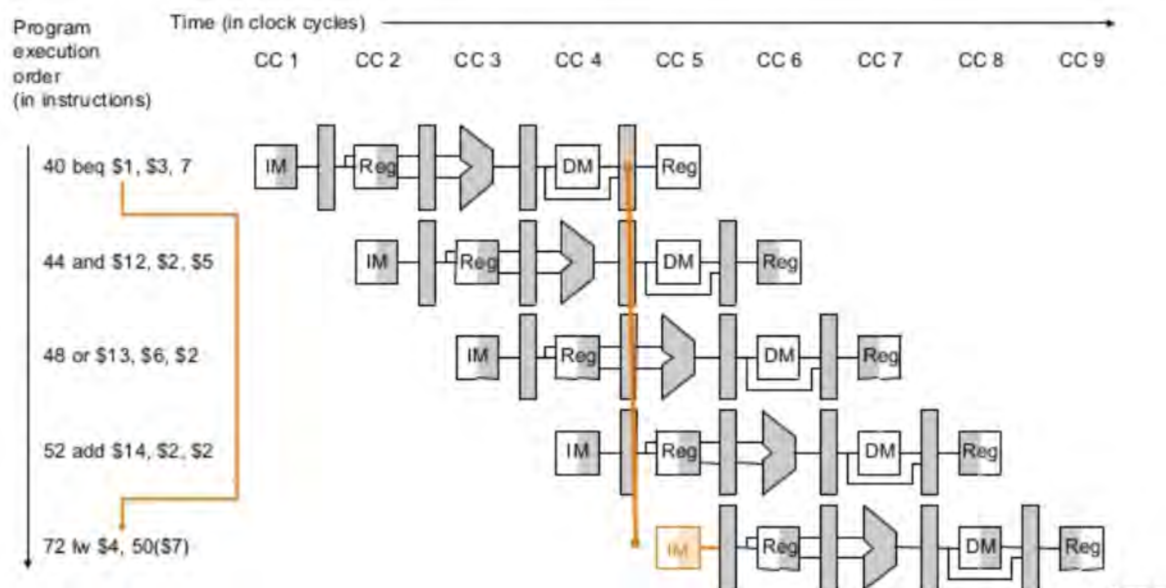
Σχήμα 4.5: Data hazards μεταξύ σταδίων



Σχήμα 4.6: Block Diagram επεξεργαστή με μονάδα κινητής υποδιαστολής πολλαπλασιασμού και διαίρεσης και υποστήριξη εντολών εκτός σειράς(Pentium 4).

3. Control Hazards : Προκύπτουν μετά απο εντολές που αλλάζουν την ροή εκτέλεσης είτε προκειται για εντολές άλματος είτε διακλάδωσης υπό συνθήκη .Αντιμετωπίζονται συμβατικά με

πάγωμα μέχρι την αποτίμηση της συνθήκης και αν χρειάζεται εκκαθάριση των λανθασμένων εντολών. Έχουν προταθεί διαφορές τεχνικές για την ελαχιστοποίηση του χρόνου παγώματος και την αποδοτικότερη υλοποίηση των εντολών ελέγχου ,στα πλαίσια αυτής της εργασίας θα γίνει μια ανασκόπηση των βασικών μεθόδων που χρησιμοποιούνται και θα προταθεί και θα υλοποιηθεί ένας διαφορετικός τρόπος αντιμετώπισης .



Σχήμα 4.7: Αποτίμηση εντολής διακλάδωσης 3cc αργότερα .

4.4 Branch prediction

Για κάθε επιτυχημένο fetch στο pipeline του επεξεργαστή πρέπει να γνωρίζουμε αν η προηγούμενη εντολή ήτανε εντολή διακλάδωσης ή άλματος ,αν πραγματοποιεί το άλμα μετά την αποτίμηση της συνθήκης και ποια είναι η διεύθυνση του άλματος . Στο σύνολο εντολών του riscv οι εντολές που αλλάζουν την ροή εκτελεσης φαινονται στο Σχημα 4.8.

Jal: αλμα στην θέση PC+ immediate
Jalr: αλμα στην θέση rs1 + immediate
Branch : αν rs1 συνθήκη rs2,αλμα στη θέση PC +immediate

Σχήμα 4.8: Εντολές που αλλάζουν την ροή εκτέλεσης

Η αποτίμηση των παραπάνω ερωτήσεων για κάθε εντολή γίνεται σύμφωνα με το Σχήμα 4.9 σε μια κλασσική υλοποίηση πέντε σταδίων.

εντολή	άλμα /όχι άλμα	στόχος αλματος
Jal	μετά το decode stage	μετά το decode stage
Jalr	μετά το decode stage	μετά το reg fetch
B<συνθηκη>	μετά το execute stage	μετά το decode stage

Σχήμα 4.9: Αποτίμηση εντολών

Αυτό σημαίνει οτι μετά απο κάθε εντολή άλματος χωρίς συνθήκη ο επεξεργαστής οφείλει να αποσύρει απο το pipeline την εντολή που ακολούθησε ώστε στον επόμενο κυκλο να προσπελάσει την σωστή, ενώ η εντολή άλματος έχει αναγνωριστεί απο το decode unit.Σε κάθε εντολή διακλάδωσης υπό συνθήκη οφείλει να αποσύρει απο το pipeline την επόμενη εντολή καθώς θα έχει αναγνωριστεί απο το decode unit η εντολή διακλάδωσης και επίσης να παγώσει για ένα κύκλο τον pc ώστε η εντολή διακλάδωσης να περάσει το execution stage και να αποτιμηθεί η συνθήκη ώστε η τίμη του pc να λάβει την σωστη τιμή .Άυτο σημαίνει οτι για την απλή υλοποιήση των πέντε σταδίων έχουμε ένα κύκλο καθυστέρηση για κάθε εντολή άλματος και δύο κύκλους καθυστέρησης για κάθε εντολή διακλάδωσης υπό συνθήκη .Στους σύγχρονους μικροεπεξεργαστές με εξαιρετικά μεγάλου μήκους Pipeline και δυνατότητα έκδοσης πολλών εντολών ανα κύκλο η καθυστέρηση που προκύπτει μετά απο εντολές διακλάδωσης ή άλματος κυμαίνονται μεταξύ 10 -20 κυκλών .Κάθως αποτέλει σημάντικο παράγοντα της απόδοσης ενος μικροεπεξεργάστη η καθυστέρηση μετά απο εντολές που αλλάζουν την ροή του προγράμματος, έχει γίνει σχετική ερευνητική δουλειά για την μείωση της καθυστέρησης στο Pipeline του επεξεργάστη .

4.5 Τεχνικές πρόβλεψης διακλαδώσεων

4.5.1 Στατικές

Ονομάζονται έτσι γιατί η απόφαση για την εκτέλεση ή όχι του άλματος λαμβάνεται απο τον μεταγλωττιστή κατά την μεταγλώττιση του προγράμματος και δεν επηρεάζεται από την εκτέλεση. Οι πιο διαδεδομένες τεχνικές στατικής πρόβλεψης ειναι :

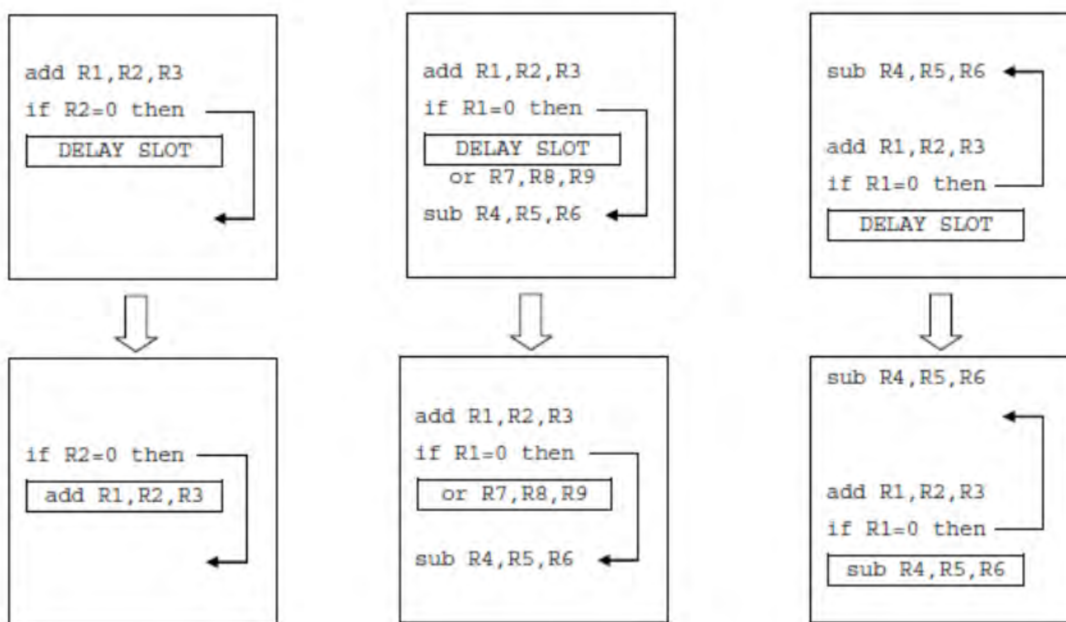
- Ολες οι εντολές διακλάδωσης να εκτελούν άλμα : Επείδη σε ενα τυπικό πρόγραμμα ο αριθμός των διακλαδώσεων που εκτελούν άλμα συνήθως είναι μεγαλύτερος απο αυτές που δεν εκτελούν, μια στατική πρόβλεψη όλων των διακλαδώσεων ως αληθείς επιφέρει ποσοστά επιτυχίας ανω του 50 .Ενδεικτικά σε ενα Loop θα έχουμε λαθος πρόβλεψη μόνο στην τελευταία εκτέλεση.

- Οι εντολές που πραγματοποιούν άλμα προς τα πίσω προβλέπεται οτι θα εκτελεστούν (αρνητικό offset ως προς τον Pc).Οι εντολές που πραγματοποιούν άλμα προς τα εμπρός προβλέπεται ότι δεν θα εκτελεστούν (θετικό Offset ως προς τον Pc).Η τεχνική αυτή χρησιμοποιείται στον Intel Pentium 4 σε περίπτωση που αποτύχει ο μηχανισμός δυναμικής πρόβλεψης.

- Εκτέλεση του προγράμματος και καταγραφή των στατιστικών για κάθε εντολή διακλάδωσης,συνεπώς προσθήκη bit ελέγχου στην κωδικοποίηση επανεκτέλεση συμφώνα με το bit ελέγχου.

- Εκτέλεση άλματος για συγκεκριμένες εντολές διακλάδωσης και μόνο. Όλες οι άλλες παραλείπονται .

- Delay slot: Αναζήτηση απο τον compiler για εντολές που εκτέλουνται σε κάθε περίπτωση και τοποθέτηση τους μετά την εντολή διακλάδωσης εως την αποτίμηση της συνθήκης.



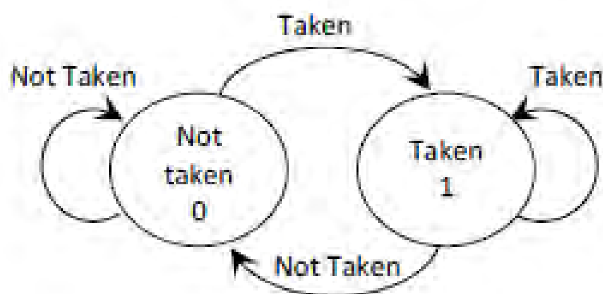
Σχήμα 4.10: Παραδείγματα πλήρωσης delay slot

Σε καθε περίπτωση οι τεχνικές στατικής πρόβλεψης διακλαδώσεων αποτελούν εύκολη και φθήνη υλοποίηση σε επίπεδο υλικού καθώς δεν χρησιμοποιούνται δεδομένα που προκύπτουν κατά την εκτέλεση , ωστόσο αυτό περιορίζει το ποσοστό επιτυχίας .[26]

4.5.2 Δυναμικές

Τεχνικές δυναμικής πρόβλεψης διακλαδώσεων περιλαμβανούν τις τεχνικές προβλεψης που χρησιμοποιούν δεδομένα που προκύπτουν κατα την εκτέλεση του προγράμματος σε πραγματικό χρόνο.

- 1 Level Branch Predictor
  i) 1 bit predictor : Χρήση του Branch History Table και αποθήκευση ενός bit για κάθε εντολή διακλάδωσης . Συνεπώς εκτέλεση κάθε εντολής διακλάδωσης σύμφωνα με την προηγούμενη εκτέλεση .[26]



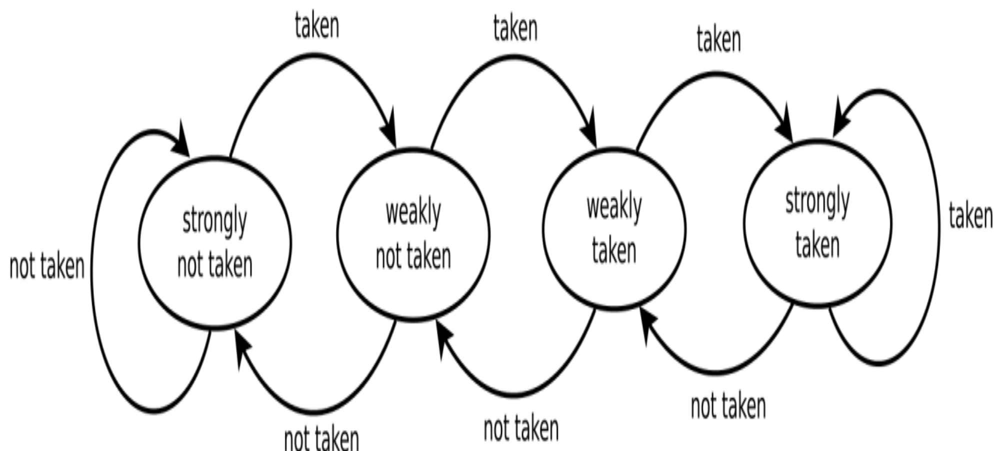
Σχήμα 4.11: Διάγραμμα καταστάσεων 1 bit predictor

current state	input (branch execution)	output (prediction)	next state
0	NT	NT	0
0	T	NT	1
1	NT	T	0
1	T	T	1

Σχήμα 4.12: Πίνακας αλήθειας 1 bit predictor

ii) 2bit predictor: Ο 1 bit predictor αλλάζει την πρόβλεψη του πολύ γρήγορα ακόμα και αν μια εντολή διακλάδωσης αλλάζει την συμπεριφορά της ανά πολλές επαναλήψεις.Για το λόγω αυτό ο 2 Bit predictor χρησιμοποιεί ενα επιπλέον Bit για την περιγραφή των πιθανών καταστάσεων και με αυτο τον τρόπο είσαγει μια καθυστέρηση στην αλλαγή της πρόβλεψης.Οι πιθάνες καταστάσεις σε αυτή την περίπτωση είναι δύο για κάθε πρόβλεψη Strong Taken ,Taken,Not Taken,Strongly Not Taken και η μετάβαση μεταξύ των καταστάσεων γίνεται σύμφωνα με το παρακάτω σχήμα .

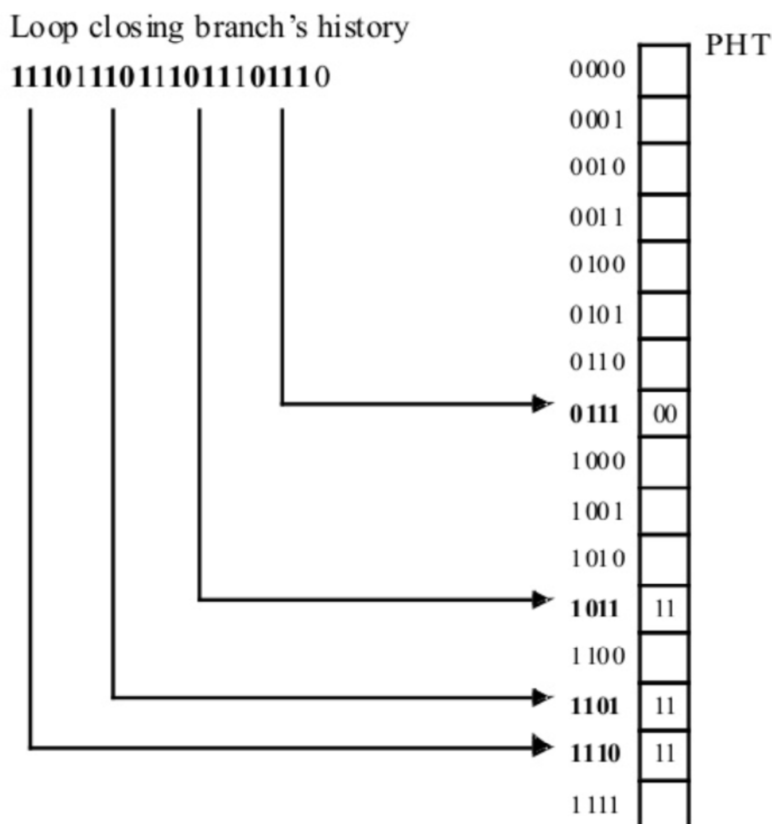
Για ένα μεγάλο πίνακα απο μετρητές κάθε εντολή διακλάδωσης μπορεί να αντιστοιχηθεί με ένα μετρητή ,ωστόσο συχνά απαιτείται ή χρήση ενός μετρητή από περισσότερες εντολές διακλάδωσης κάτι που προκαλεί πτώση της ακρίβειας πρόβλεψης.Για ενα δωσμένο πλήθος απο μετρητές μια σύνολο-συσχετιστική (set-associative) υλοποίηση επιφέρει συνήθως καλύτερα αποτελέσματα.[26]



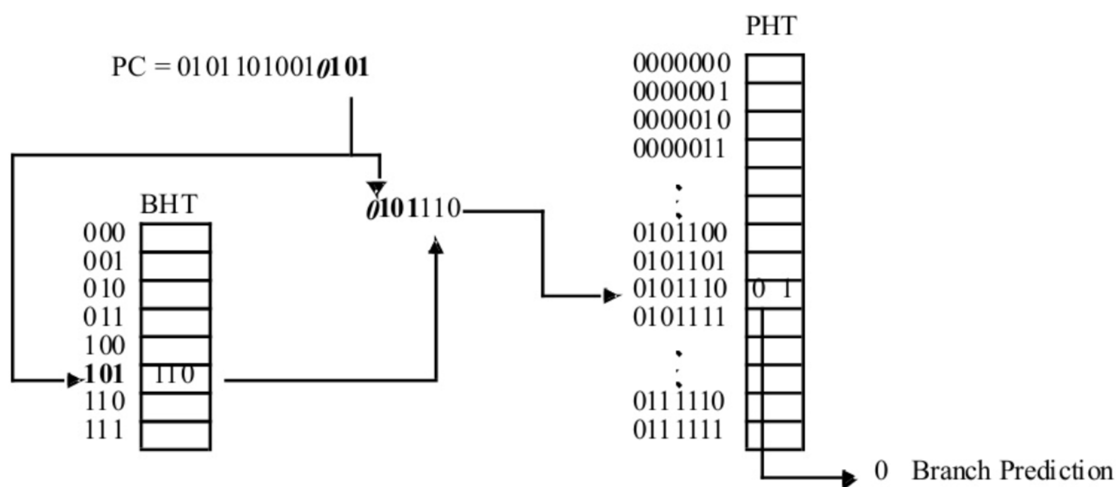
Σχήμα 4.13: Διάγραμμα καταστάσεων 2 Bit predictor

- 2-Level Branch Predictor

  i) Local branch prediction : Ένας τρόπος να αυξηθεί η αποτελεσματικότητα των μηχανισμών πρόβλεψης διακλαδώσεων είναι η παρατήρηση πως πολλές εντολές διακλάδωσης εκτελούν επαναλαμβανόμενα μοτίβα με σύνηθες παράδειγμα η εντολή διακλάδωσης ενος for-loop . Για for (i=1; i<=4; i++) η εντολή διακλάδωσης εκτελεί το μοτίβο $(1110)^n$ οπου το 1 αντιπροσωπέυει την πραγματοποίηση του άλματος και το 0 όχι και n είναι ο αριθμός που το for loop θα εκτελεστεί .Για την συγκεκριμένη περίπτωση αν ξέρουμε το αποτέλεσμα της εντολής διακλάδωσης τις 3 τελευταίες εκτελέσεις της ,μπορούμε με βεβαιότητα να προβλέψουμε την επόμενη εκτέλεση.Αυτή η μέθοδος χρησιμοποιεί δύο πίνακες , ο πρώτος πίνακας ,αποκαλούμενος και πίνακας ιστορίας,κρατά το αποτέλεσμα των προηγούμενων εκτελέσεων των πρόσφατων εντολών διακλάδωσης και επειδή ο πίνακας αυτός θα είναι καθορισμένου μεγέθους μπορεί να υλοποιηθεί πάλι η συνολοσυσχετιστική απεικόνιση των εντολών διακλάδωσης στον πίνακα καταγραφής της ιστορίας.Κάθε πεδίο του πίνακα απεικονίζει την αποτίμηση των n πιο πρόσφατων εκτελέσεων της εντολής διακλάδωσης που συσχετίζοται με το πεδίο,με n να είναι το μέγεθος σε bits του πεδίου.Ο δεύτερος πίνακας είναι ένας πινακας 2 bit μετρητών με το πλήθος των οποίων για καθε εγγραφή να ειναι 2^n δηλαδή για κάθε πιθανό συνδιασμό ιστορίας αντιστοιχεί και ένας μετρητής 2Bit η τιμή του οποίου καθορίζει την πρόβλεψη της επόμενης εκτέλεσης της εντολής διακλάδωσης.Με την κατάλληλη αρχικοποίηση η μεθοδος αυτή μπορεί να προβλέψει με απόλυτη επιτυχία κάθε επαναλαμβανόμενο μοτίβο μιας εντολής διακλάδωσης ,ωστόσο το μέγεθος του μοτίβου μπορεί ναι είναι εως ένα ψηφίο μεγαλύτερο των ψηφίων ιστορίας . Συνεπώς για την επιτυχή πρόβλεψη της εντολής διακλάδωσης μιας επανάληψης 4 φορών απαιτούνται 3 ψηφία ιστορίας και κατ' επέκταση 2^3 μετρητές των 2 bits ,αυτός ο μεγάλος αριθμός μετρητών είναι ενα θέμα που θα προσπαθήσει να επιλύσει μια τροποποίηση της μεθόδου αυτης. [26],[22],[29],[30]



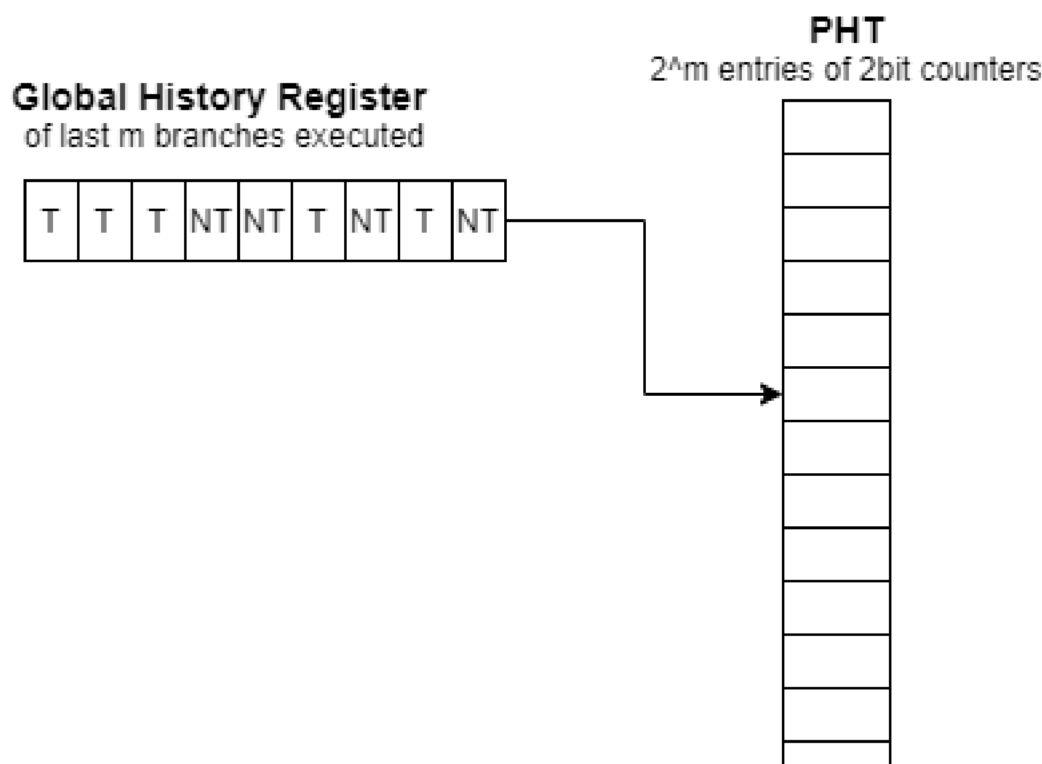
Σχήμα 4.14: Απεικόνιση πρόβλεψης local branch predictor για κάθε πιθανή ιστορία μιας εντολής διακλάδωσης.



Σχήμα 4.15: Συνολό-συσχετιστική απεικόνιση local branch predictor

ii)Global brach prediction: Σύμφωνα με την μέθοδο local branch prediction ο μόνος παράγοντας που επηρεάζει την πρόβλεψη είναι οι προηγούμενες εκτελέσεις της ίδιας εντολής

διακλάδωσης. Σε πολλές περιπτώσεις όμως η αποτίμηση μιας διακλάδωσης εξαρτάται απο το αποτέλεσμα προηγούμενων διακλάδωσεων. Σε ενα παράδειγμα της μορφής if (x1>0) : ; if(x1<0) Η αποτίμηση της δεύτερης εντολής If εξαρτάται απο την πρώτη εντολή, ο συσχετισμός αυτός μπορεί να αξιοποιηθεί χρησιμοποιώντας την μέθοδο global branch prediction .Στην περίπτωση αυτή χρησιμοποείται μια κοινή ιστορία για όλες τις εντολές διακλάδωσης των n bits και αντίστοιχα ένας πίνακας 2^n μετρητών των δύο bits ώστε να υπάρχει ένας μετρητής για κάθε πιθανή ιστορία. [29]



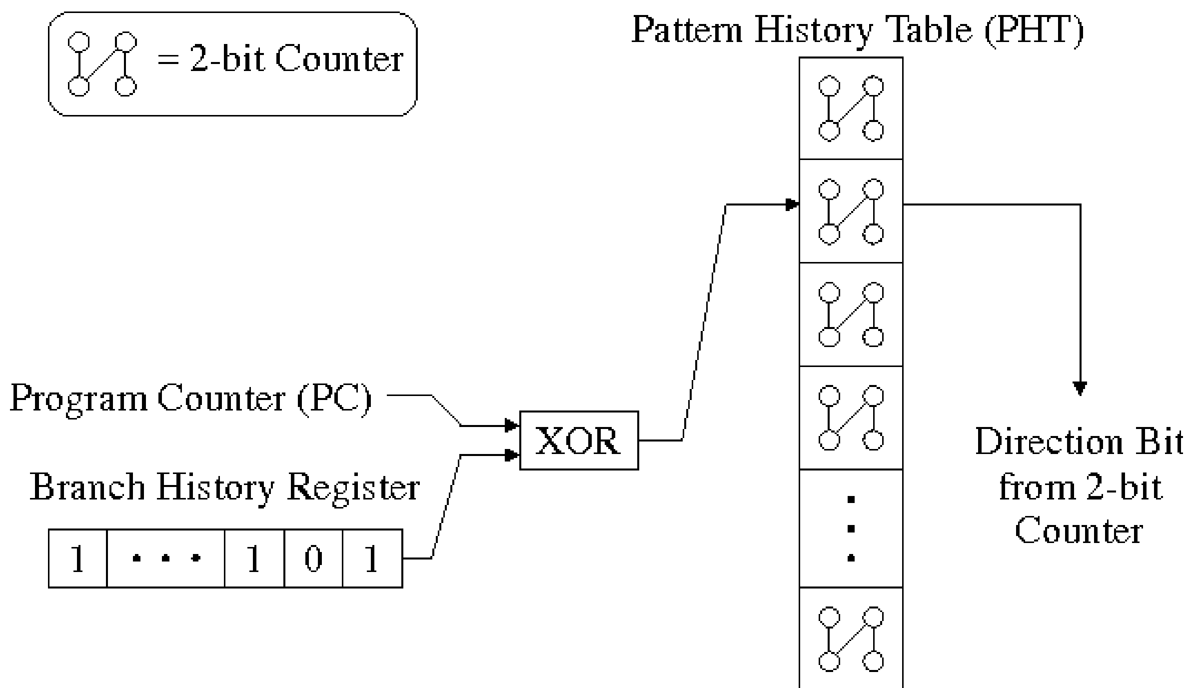
Σχήμα 4.16: Global Branch Prediction

Η μεθοδός αυτή μπορεί να φανεί χρήσιμη και να εκτελέσει και την λείτουργία ενος local branch predictor υπό προυποθέσεις.

```
for (i=0; i<100; i++)
for (j=0; j<3; j++)
```

Στο παραπάνω στιγμιότυπο κώδικα το οποίο αποτελείται απο δύο εντολές διακλάδωσης η εσωτερική εντολή αρχικά για δύο επαναλήψεις εκτελεί άλμα και κατόπιν στην τρίτη επανάληψη δεν εκτελεί το άλμα ενώ η εξωτερική εντολή εκτελεί άλμα για τις εκατό πρώτες επαναλήψεις της και κατόπιν για μια επανάληψη δεν εκτελεί . Ενας Global branch predictor με μεγεθος ιστορίας 4 θα είχε μετά την αρχικοποίηση πολύ καλή συμπεριφορά πρόβλεψης . [26],[22],[29],[30]

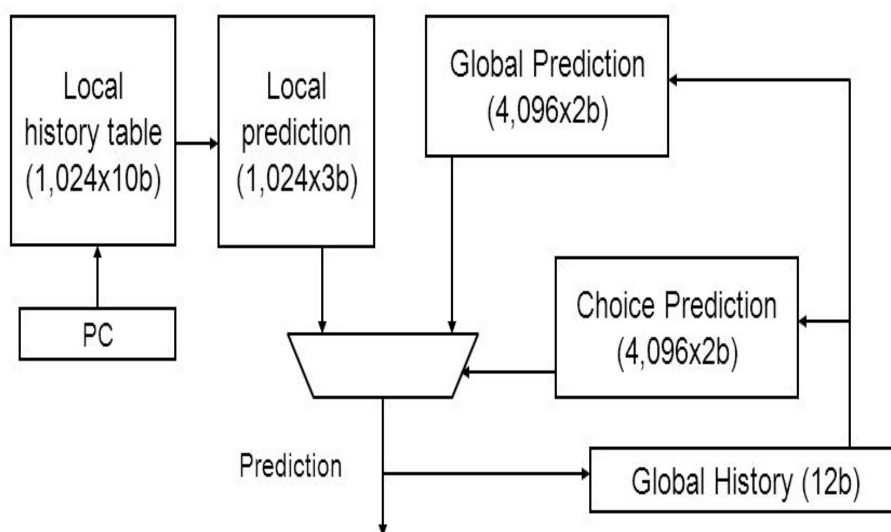
iii) Για κάθε N bit ιστορία χρησιμοποιούμε 2^N μετρητές των 2 bit ωστόσο κατι τέτοιο αποτελεί σπατάλη καθώς το πλήθος των μετρητών που χρειαζόμαστε είναι N αλλά χωρίς να γνωρίζουμε ποιές απο τις πίθανες ιστορίες είναι σημαντικές αναγκαζόμαστε να δεσμέυσουμε χώρο για μετρητές που δεν χρησιμοποιούνται. Ένας διαφορετικός τρόπος προσέγγισης είναι να μην δεσμεύσουμε χώρο για κάθε πιθανή ιστορία μιας εντολής διακλάδωσης αλλά να δημιουργήσουμε εναν πίνακα 2bit μετρητών κοινούς για ολές τις εντολές διακλάδωσης και για όλες τις πιθανές ιστορίες. Η αντιστοίχιση μιας εντολής διακλάδωσης με βάση την ιστορία της στον κατάλληλο μετρητή μπορεί να γίνει συνδιάζοντας με xor την διευθυνση της εντολής με την ιστορία της. Συνεπώς καθε διαφορετική ιστορία θα αντιστοιχίζεται με διαφορετικό μετρητή , οπως και κάθε διαφορετική εντολή διακλάδωσης. Ο παραπάνω τρόπος εγκυμονεί τον κίνδυνο να υπάρξει επικάλυψη μεταξύ διαφορετικών εντολών ή διαφορετικών ιστοριών στον ίδιο μετρητή ,ωστόσο με μεγάλο πίνακα μετρητών η περίπτωση αυτή είναι σπάνια .Με αυτό τον τρόπο οι εντολές που εκτελούν μικρά επαναλαμβανόμενα μοτίβα λαμβάνουν τους ανάλογους μετρητές ενώ εντολές που εκτελούν μεγάλα μοτίβα λαμβάνουν ανάλογα περισσότερους μετρητες.Η παραπάνω μέθοδος συναντάνται τόσο σε local branch predictor όσο και σε global predictor με τις βιβλιογραφικές ονομασίες Pshare και Gshare αντίστοιχα. [26],[22],[29],[30],[31]



Σχήμα 4.17: Predictor with shared counters

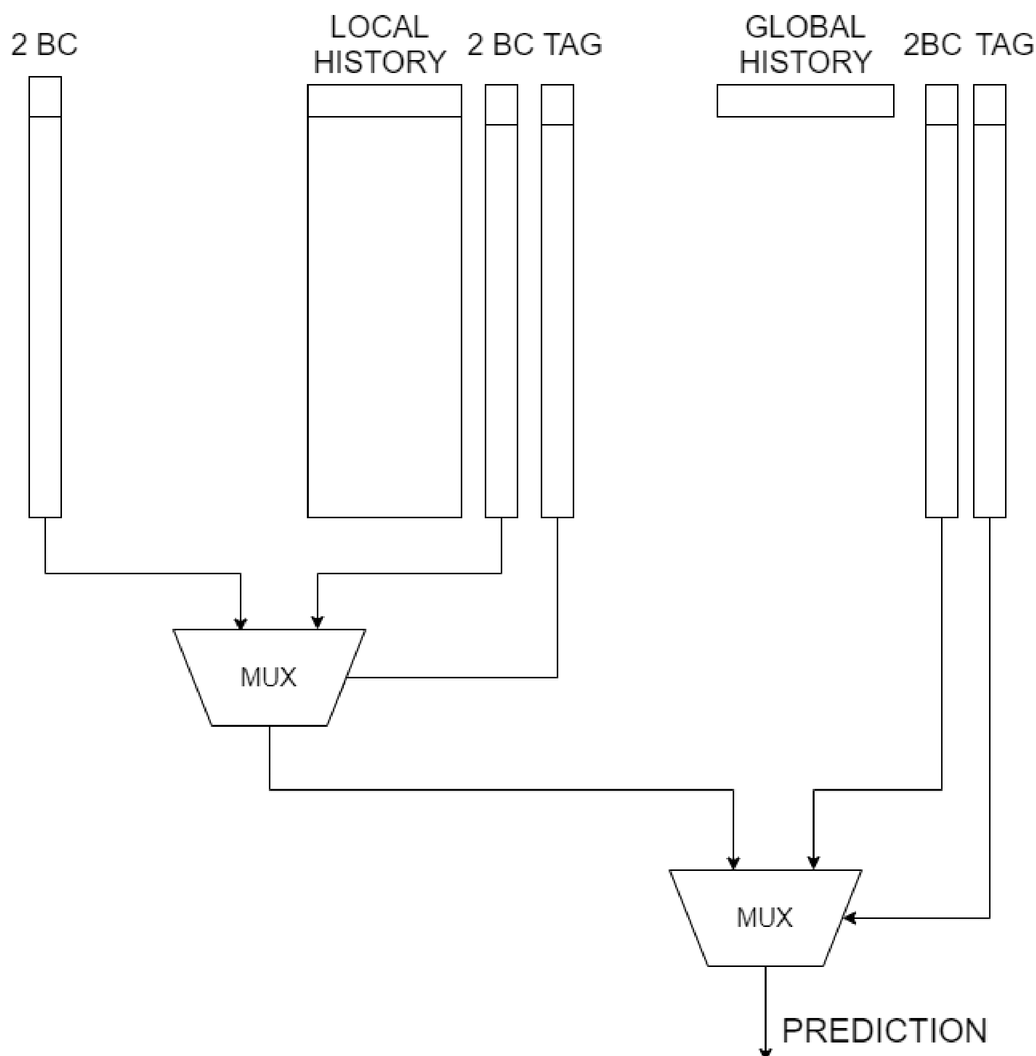
iv)Tournament predictor : Ο Pshare και ο Gshare αποτελούν δύο αξιόπιστους τρόπους πρόβλεψης διακλαδώσεων αλλα δεν συμπεριφέρονται εξίσου για όλους τους τύπους των εντολών διακλάδωσης .Για διακλαδωσεις που συσχετίζονται ο gshare είναι σαφώς καλύτερος ενώ για διακλαδώσεις χωρίς συσχέτιση ο pshare είναι καλυτερος. Για αυτο το σκοπό ο Tournament predictor συνδιάζει αυτούς τους δύο χρησιμοποιώντας ενάν επιπλέον πίνακα 2 Bit

μετρητών με βαση τον οποίο γίνεται η επιλογή της καταλληλότερης μεθόδου. Ο επιπλεον πίνακας αποκαλούμενος και ως meta-predictor ανανεώνεται αναλογα με το ποια μέθοδος έδωσε την σωστή προβλεψη για την εντολή διακλάδωσης. .[18]



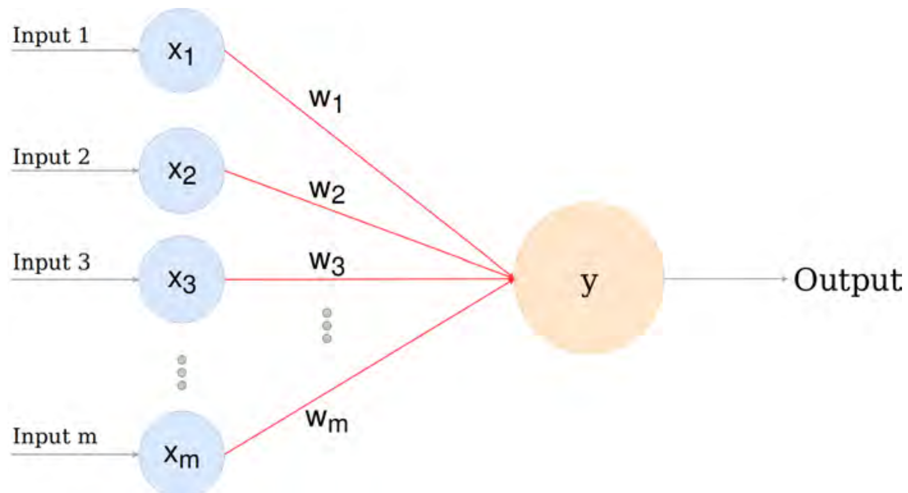
Σχήμα 4.18: Tournament predictor of alpha 21264 proseccor

v) Hierachical predictor : Η ίδεα της ιεραρχικής μεθόδου πρόβλεψης διακλαδώσεων είναι ο συνδιασμός μιας απλής μεθόδου η οποία είναι ικανή να προβλέψει με επιτυχία την πλειόνοτητα των εντολών διακλάδωσης οπως ενάς 2bit predictor και μιας ακριβότερης μεθόδου που απαιτείται για καποίες συγκεκριμένες εντολές. Ο Pentium M χρησιμοποιεί ενα hierarchical predictor που αποτελείται από έναν 2 bit predictor , από ένα pshare και απο ενα gshare predictor . Η διαδικασία προσπέλασης τους και ανανέωσης των τιμών είναι σχετικά απλή. Σε περίπτωση που υπάρχει η συγκεκριμένη εντολή στο gshare χρησιμοποιούμε την πρόβλεψη αυτής της μεθόδου αλλιώς ελέγχουμε στο pshare και αν υπάρχει ,χρησιμοποιούμε την πρόβλεψη αυτής της μεθόδου ,διαφορετικά χρησιμοποιούμε την πρόβλεψη απο τον πίνακα των 2Bits μετρητών. Ανάλογα σε ποια μέθοδο χρησιμοποιήσαμε για να πάρουμε την πρόβλεψη ανανεώνουμε την εκάστοτε. Σε περίπτωση που η μέθοδος που χρησιμοποιήσαμε δίνει λάθος πρόβλεψη εισάγουμε την εντολή στην αμέσως επόμενη μέθοδο . Συνεπώς μόνο αν ο 2 Bit predictor δίνει λάθος πρόβλεψη εισάγουμε την εντολή στο Pshare και αντίστοιχα αν η Pshare δίνει λάθος πρόβλεψη την εισάγουμε στο gshare. [26],[29]

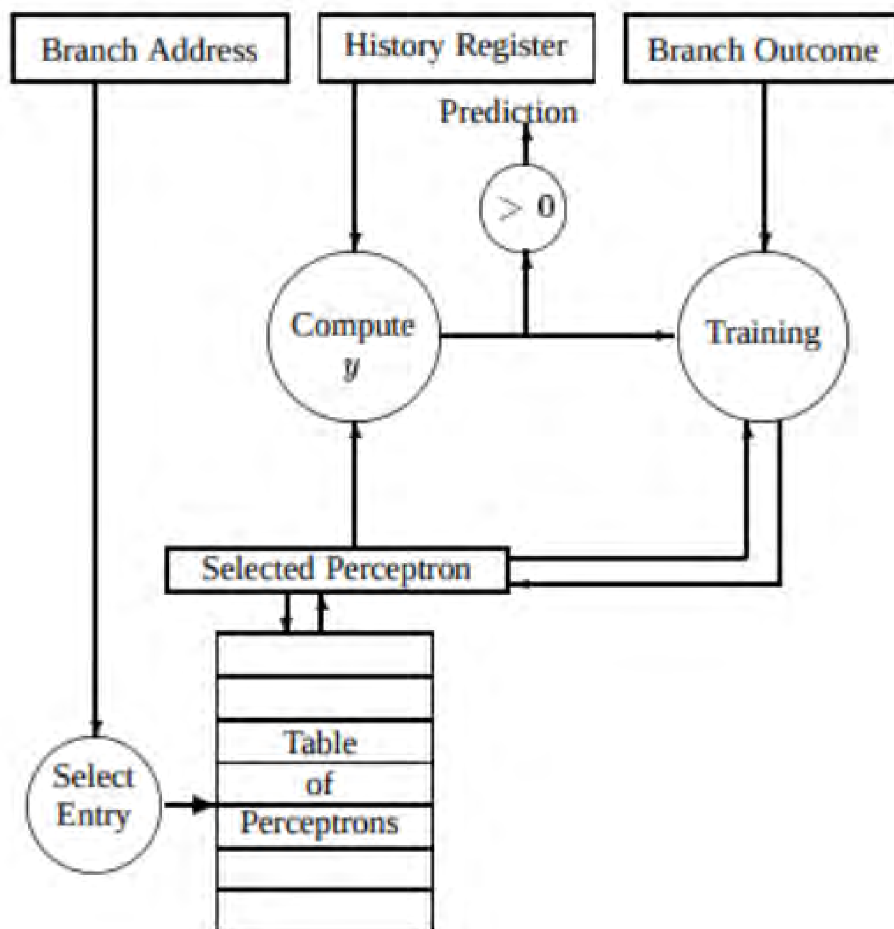


Σχήμα 4.19: Hierachical predictor

vi) Perceptron predictor : Το σημαντικότερο πρόβλημα των 2-level predictors με μετρητές είναι η δυσκολία πρόβλεψης εντολών που εμφανίζουνε συσχέτιση με μακρινές εντολές διακλάδωσης .Για την ορθή πρόβλεψη τετοιών εντολών απαιτούνται πολλά Bits ιστορίας και καθώς για καθε πίθανη ιστορία απαιτείται διαφορετικός μετρητής πρόβλεψης ,αυτό αυξάνει εκθετικά το πλήθος των μετρητών με αποτέλεσμα να δημιουργείται ένα μια μη αποδοτική μηχανή πρόβλεψης οσον αφορά την σχεδίαση και την κατανάλωση ενέργειας. Η εισάγωγη του αλγορίθμου percepton του κλάδου της μηχανικής μάθησης προσφέρει έναν εναλλακτικό τρόπο για την αντιμετώπιση του παραπάνω. Ο αλγόριθμος perceptron αποτελεί την απλούστερη μορφή νευρωνικού δικτύου και μπορεί να ταξινομίσει με επιτυχία γραμμικά διαχωρίσιμα σύνολα .Χρησιμοποιεί εναν πίνακα βαρών αντίστοιχου μεγεθούς με το πλήθος των εισόδων. Πολλαπλασιάζει κάθε στοιχείο της εισόδου με το αντίστοιχο βάρος και προσθέτει τα γινόμενα μεταξύ τους .



Σχήμα 4.20: Άθροιση των γινονομένου εισόδων –βαρών για την κατασκευη του αλγορίθμου perceptron



Σχήμα 4.21: Perceptron branch predictor

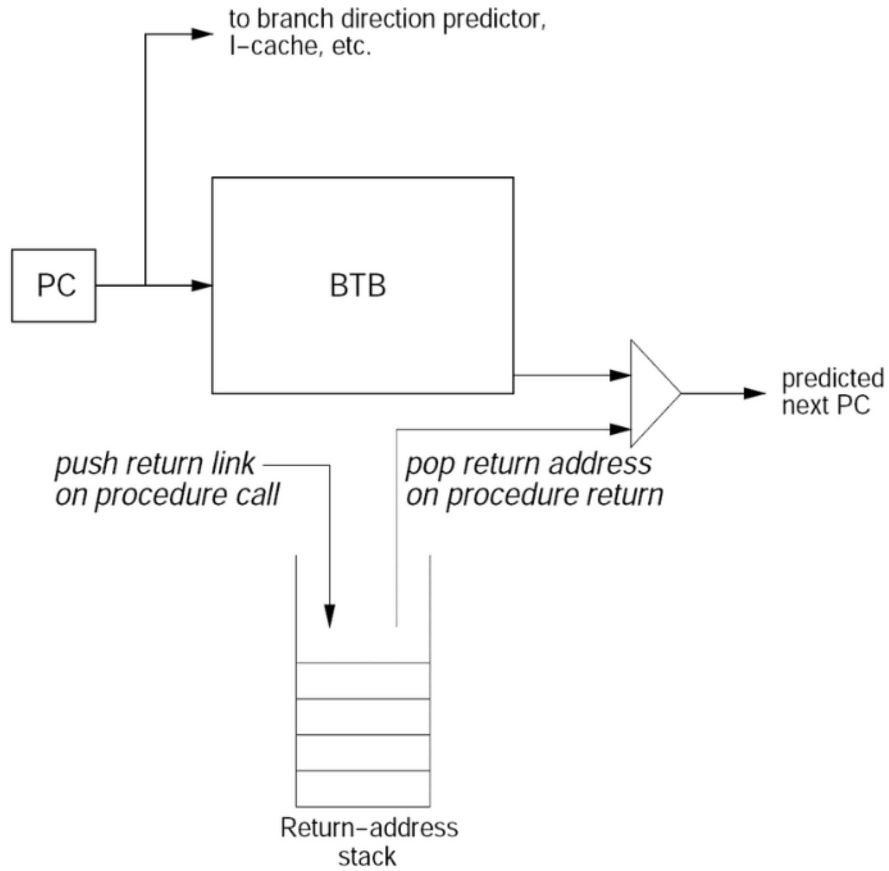
Υστερα απο εκπαίδευση του αλγορίθμου δηλαδή την επίλογη κατάλληλων βαρών και της

τιμής διαχωρισμού μπορει να υλοποιήσει τις γραμμικά διαχωρίσιμες συναρτήσεις . Ετσι μπορεί να υλοποιήσει τα λογικά AND,OR ,NOT αλλα όχι το λογικό XOR . Κατι τετοιο θα μπορούσε να υλοποιηθει με Perceptron δύο επιπέδων το οποίο ομώς δεν μας απασχολέι καθώς αποτελεί πολύπλοκη υλοποίηση για σχεδιασμό και χρήση σε εναν predictor.Συνεπώς με την κατάλληλη επιλογή βαρών μέσω εκπαίδευσης ο αλγοριθμός είναι ικανός να μάθει ποια στοιχεία ιστορίας μιας διακαλάδώσης συνεισφέρουν ποσοτικά και ποιοτικά στην νέα πρόβλεψη.[14],[15]

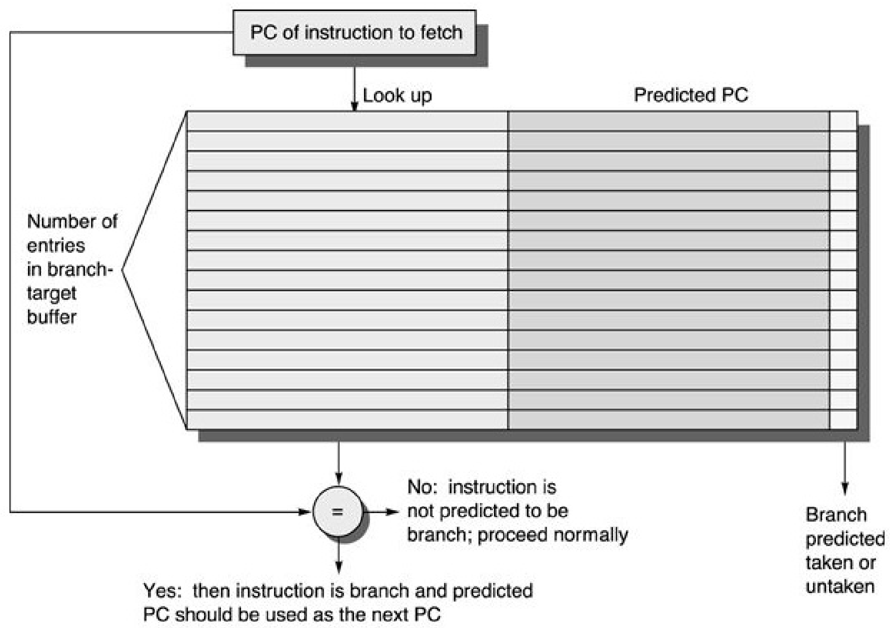
vii)Return Address Stack: Με τα εργαλεία που εχουμε αναλύσει παραπάνω μπορούμε με επιτυχία να επιλύσουμε το πρόβλημα της πρόβλεψης διακλαδώσεων τόσο για τις εντολές διακλάδωσης υπο σύνθηκη τοσο για τις εντολές άλματος . Το κοινό στοιχείο αυτών των εντολών είναι οτι η διεύθυνση του άλματος είναι οτι σε κάθε εκτέλεση σταθερή οπότε αρκεί για την πρόβλεψη της διεύθυνσης μια καταχώρηση στο BTB . Στις εντολές return ομως δεν ισχύει αυτό,μια συνάρτητη μπορεί να έχει πολλαπλά σημεία κλήσης συνεπώς είναι δυνατόν σε κάθε εκτέλεση της να επιστρέφει σε διαφορετικά σημεία του προγραμματος. Για αυτο το σκοπό υλοποιούμε μια μικρή μνήμη σύμφωνα με τα πρότυπα της stack η οποία ομως βρίσκεται πάνω στο επεξεργαστή και έχει άμεση προσπέλαση.Κάθε φορά που καλείται μια συνάρτηση τοποθετείται μια εγγραφή στην μνήμη και ο δείκτης προχωρά μια θέση ,σε κάθε εκτέλεση μιας εντολής return η διεύθυνση επιστροφής ανακτάται απο την stack του επεξεργαστή και ο δείκτης πλέον δείχνει την προηγούμενη κλήση μιας συναρτησης. Επειδή η μνήμη αυτη είναι πολύ μίκρη πρέπει να λάβουμε υπόψη την πολιτική που θα ακολουθήσουμε σε περίπτωση που γεμίσει . Σε αυτή την περίπτωση είναι προτιμότερο να συνεχίσουμε να τοποθετούμε καινούριες εγγραφες διαγράφοντας παράλληλα τις παλαιότερες . Αυτο συμβαίνει γιατί στα περισσότερα προγράμματα οι εξωτερικές κλήσεις συναρτήσεων επιστρέφουν συνήθως τελευταίες ενώ εσωτερικά μια συνάρτηση μπορεί να εκτελεστεί και επιστρέψει πολλές φορές. [25]

4.5.3 Branch Target Buffer

Μετά απο μία εντολή διακλάδωσης δεν αρκεί μόνο γνώση αν θα πραγματοποιήσει ή όχι το άλμα αλλά σε περίπτωση που το πραγματοποίησει θα πρέπει να γνωρίζουμε και την διεύθυνση του άλματος . Συνήθως αυτό γίνεται γνωστό μετά την φάση αποκωδικοποίησης ή ακόμα και μετά την φάση εκτέλεσης . Για τον σκοπό αυτό στην φάση έκδοσης μιας εντολής τοποθετείται ένας πίνακας ο BTB που αποθηκεύει τις διευθύνσεις των εντολών διακλάδωσης στο ένα του πεδίο και στο άλλο τις διευθύνσεις προορισμού των εντολών διακλάδωσης. Ο BTB συνήθως συνδιάζεται με μία απο τις προαναφερθήσες μεθόδους πρόβλεψης για την αποτίμηση τις εντολής ,έτσι συνήθως ο BTB περιλαμβάνει τον προορισμό μιας εντολής διακλάδωσης άλλα και καποιά bit ιστορίας. [20]



Σχήμα 4.22: Return address stack



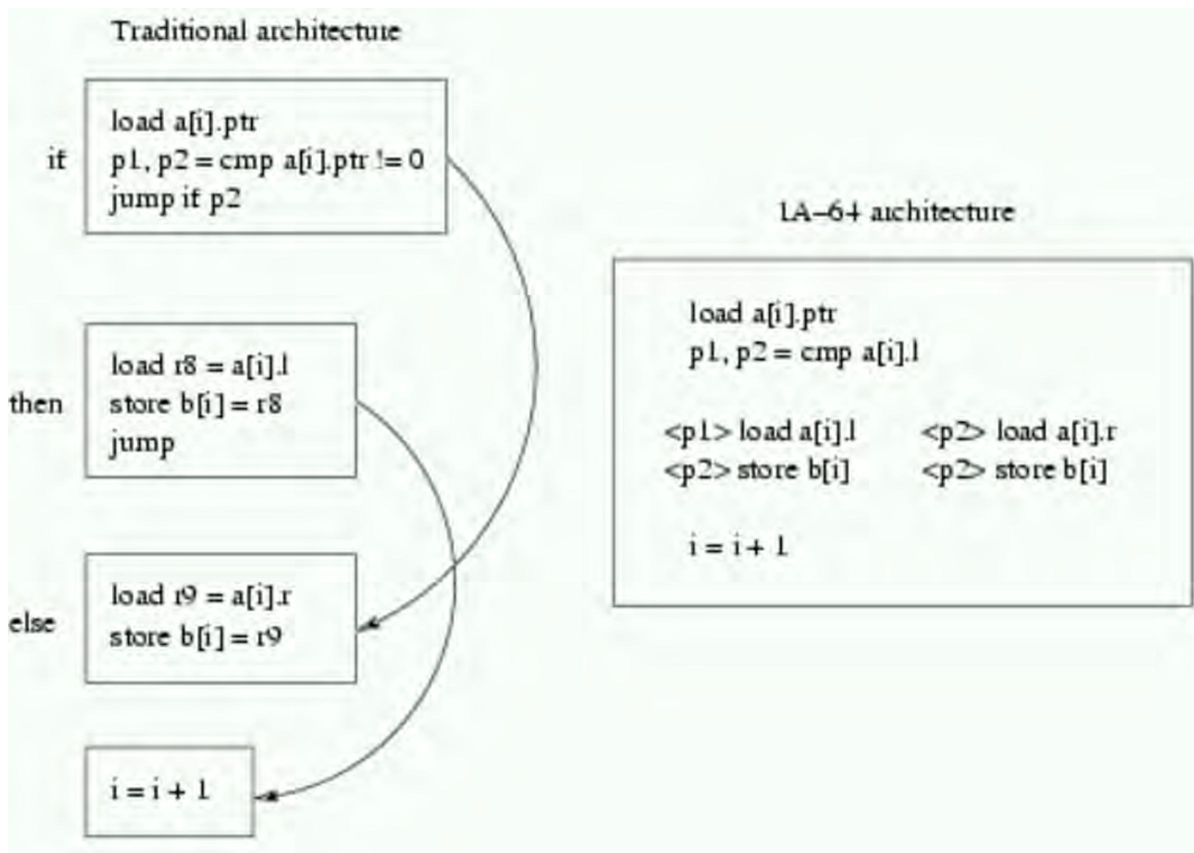
Σχήμα 4.23: Branch Target Buffer

4.5.4 Predication

Η εκτέλεση υπο συνθήκη ή αλλιώς predication είναι μια τεχνική που δημιουργήθηκε για να μειώσει του κύκλους καθυστέρησης μετά απο εντολές διακλάδωσης.Παρόλο τις πολύπλοκές μεθόδους πρόβλεψης διακλαδώσεων που αναφέρθηκαν παραπάνω και χρησιμοποιούνται στις εμπορικές υλοποιήσεις έχουν καταφέρει να μειώσουνε σε μεγάλο βαθμό το ποσοστό των μη προβλεψιμων διακλαδώσεων ,μια λάθος πρόβλεψη επιφέρει σημαντική μείωση της απόδοσης του επεξεργαστή. Τυπικά ενα ποσοστο 5-10 της εκατό λάθος προβλέψεων μπορεί να ρίξει την απόδοση του επεξεργαστή έως 30-40 της εκατό. Η εκτέλεση υπό συνθήκη αφαιρεί τις εντολές διακλάδωσης απο τον κώδικα και επιτρέπει τις δύο κατευθύνσεις να εκτελεστούν παράλληλα μετατρέποντας το πρόβλημα των control hazards σε πρόβλημα εξαρτήσεων δεδομένων, καθώς η εντολή επιτρέπεται να αλλάξει την κατάσταση του επεξεργαστή μόνο σε περίπτωση που το ειδικό πεδίο που βρίσκεται στην λέξη της εντολής λάβει αλήθη τιμή από την εκτέλεση της εντολής διακλαδωσης. O Intel 64-IA είναι ένα παράδειγμα εμπορικής υλοποίησης της τεχνικής αυτής.Οι περισσότερες εντολές που ανήκουν στο Σύνολο Εντολών του συγκεκριμένου επεξεργαστή διαθέτουν το συγκεκριμένο πεδίο που καθορίζει την εκτέλεση ή όχι της εντολής . Στα Σχηματα 4.24 και 4.25 δίνεται ένα αντιπροσωπευτικό παράδειγμα των αλλαγών σε επίπεδο μεταγλώττισης γαι την υποστήριξη αυτής της αρχιτεκτονικής.

```
if a[i].ptr != 0 {
    b[i] = a[i].l;
} else {
    b[i] = a[i].r;
}
i = i + 1;
```

Σχήμα 4.24: Παράδειγμα κώδικα



Σχήμα 4.25: Παραδοσιακή Αρχιτεκτονική , Αρχιτεκτονική IA-64

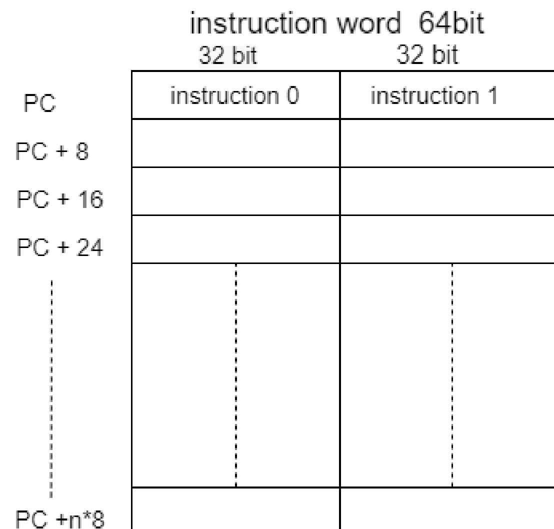
Η τιμή του a[i].ptr φορτώνεται απο την μνήμη και χρησιμοποιείται σαν συνθήκη για την εντολή διακλάδωσης . Σε μια παραδοσιακή αρχιτεκτονική ο κώδικας του Σχήματος 4.24 χωρίζεται σε 4 blocks όπως φαίνεται στο Σχήμα 4.25. Το πρώτο block περιλαμβάνει την σύγκριση το δέυτερο το κομμάτι του If το τρίτο το else και τέλος η εντολή πρόσθεσης που εκτελείται σε κάθε περίπτωση.Στην αρχιτεκτονική του IA-64 μετά την εντολή φόρτωσης τοποθετείται μια εντολή σύγκρισης που δίνει τιμή στις μεταβλητές p1,p2 .Στην συνέχεια οι επόμενες εντολές αποστέλονται παράλληλα για εκτέλεση αλλά μόνο οι εντολές με αληθή αποτίμηση του p1 ή πεδίο επιτρέπεται να αλλάξουν την κατάσταση του επεξεργαστή.Συνεπώς δεν υπάρχει καθυστέρηση και καμία περίπτωση λάθος πρόβλεψης με μοναδική παραχώρηση την υποστήριξη παράλληλης εκτέλεσης απο τον επεξεργαστή και την τροποποίηση σε επίπεδο μεταγλωττιστή για την υποστήριξη των συγκεκριμένων εντολών.[8],[3]

Κεφάλαιο 5

Περιγραφή υλοποίησης

5.1 Γενική περιγραφή

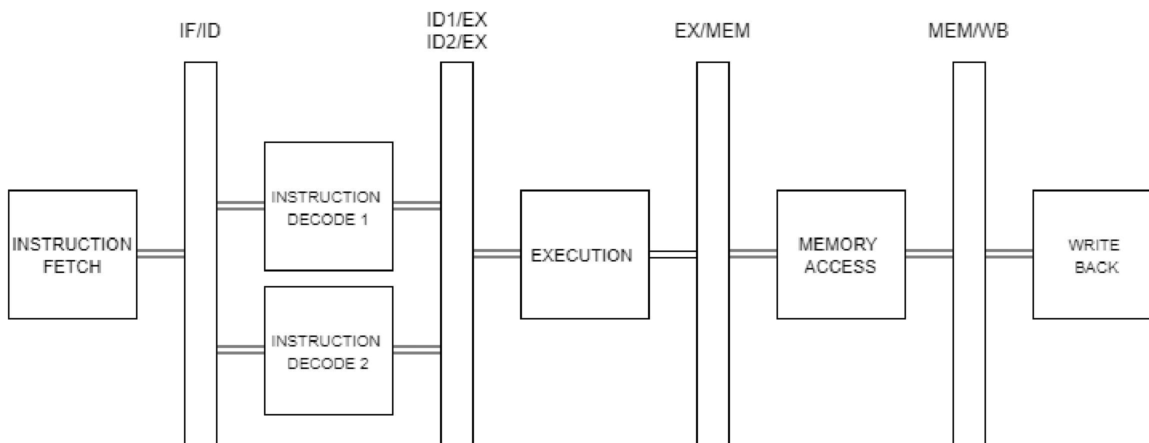
Η υλοποίηση που προτείνεται και υλοποιείται σε αυτή την εργασία αποτελεί έναν εναλλακτικό τρόπο αντιμετώπισης των εντολών διακλάδωσης χωρίς την χρήση μονάδας πρόβλεψης διακλαδώσεων .Η τροποποίηση λαμβάνει χώρα στην κλασσική αρχιτεκτονική RISC των πέντε σταδίων που περιγράφεται παραπάνω και η αποτίμηση των εντολών δικλαδωσης είτε με συνθήκη είτε χωρίς γίνεται στο τέλος του σταδίου EX. Η βασική ιδέα είναι η έκδοση στον επεξεργαστή μιας λέξης εντολής των 64 bits η οποία λέξη είναι διαμορφωμένη κατάλληλα σε επίπεδο μεταγλώτισσης ώστε να περιλαμβάνει δύο εντολες των RISC RV32Ibit.Οι εντολές που αποτελούν την ενιαία εντολή μπορεί να ειναι δύο εντολές απο διαδοχικές θέσεις μνήμης ή δυο εντολές απο διαφορετικές θέσεις μνήμης .



Σχήμα 5.1: 64bit λέξη εντολής

Η λέξη εντολής 64 Bit προωθείται και αποδικωποιείται απο το decode unit στις δύο επιμέ-

ρους εντολές ταυτόχρονα και στην συνέχεια επιλέγεται ποια απο τις δύο θα προωθηθεί στο στάδιο εκτέλεσης το οποίο έχει την δυνατοτήτα αντιμετώπισης μιας εντολής ανά κύκλο .Απο πλευράς υλικού η τροποποίηση απο μια κλασσική μικροαρχιτεκτονική risc ειναι η έκδοση 64bit εντολής και η διπλή μονάδα αποδικωποίησης καθώς και κάποια επιπλέον σήματα ελέγχου, τα υπόλοιπα στάδια του pipeline ειναι της βασικής αρχιτεκτονικής risc.



Σχήμα 5.2: Αρχιτεκτονική 5 σταδίων

Πιο συγκεκριμένα:
1η περίπτωση : 'Εκδοση ανα 2 cc . Όταν στο στάδιο ID και EX δεν υπάρχει εντολή διακλάδωσης η έκδοση γίνεται ανα δύο κύκλους ,οι εντολές που αποτελούν την 64Bit λέξη ειναι εντολές απο διαδοχικές θέσεις μνήμης που σύμφωνα με την ροή του προγράμματος πρέπει να εκτελεστούν σειριακά .Οι εντολές αποδικωποιούνται ταυτόχρονα απο το decode unit και διαβάζουνε τα τελούμενα εισόδου απο το φάκελο καταχωρητών . Ανα κύκλο προωθείται κάθε μια απο τις δύο εντολές στο ex stage .

<table>
<tr><th></th><th>INSTRUCTIONS</th><th>1cc</th><th>2cc</th><th>3cc</th><th>4cc</th><th>5cc</th><th>6cc</th><th>7cc</th><th>8cc</th><th>9cc</th><th>10cc</th><th>11cc</th><th>12cc</th></tr>
<tr><td>Pc</td><td>Add x1,x1,x2
or x2 x2 x3</td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Pc+8</td><td>Addi x2 x1 2
ori x3 x2 2</td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
MEM</td><td>|
WB</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Pc+16</td><td>Sw x2 4(x1)
lw x3 4(x1)</td><td></td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td><td></td><td></td></tr>
<tr><td>Pc+24</td><td>add x1 x1 x2
or x2 x2 x3</td><td></td><td></td><td></td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td></tr>
</table>

Σχήμα 5.3: Παροχέτευση εντολών στα στάδια της υλοποίησης χωρίς εντολή διακλάδωσης

2η περιπτωση : Έκδοση ανα 1cc .Όταν στο στάδιο ID ή EX υπάρχει εντολή διακλάδωσης η έκδοση γίνεται αν 1cc και η εντολές που αποτελούν την 64Bit λεξη είναι εντολές απο διαφορετικές θέσεις μνήμης.Πιο συγκεκριμένα η πρώτη εντολή είναι η επόμενη εντολή της εντολής διακλάδω-

σης δηλαδή της εντολής που θα πρέπει να εκτελεστεί σε περίπτωση που δεν εκτελεστεί το άλμα και η δεύτερη είναι η εντολή που βρίσκεται στην διεύθυνση του άλματος.Έχουμε δηλαδή ταυτόχρονη έκδοση και των δύο κατευθύνσεων . Στον επόμενο κύκλο γίνεται εκδοση της μεθεπόμενης της εντολής διακλάδωσης μαζι με την επόμενη εντολή απο την διεύθυνση άλματος . Απο τα τις δυο αυτές λέξεις των 64 Bit θα εκτελεστεί η μια εντολή απο κάθε λέξη ανάλογα το αποτελέσμα που θα προκύψει απο την εντολή διακλαδωσής . Στον επόμενο κύκλο ο μετρητής προγράμματος εχει πλέον την σωστή τίμη αφού η εντολή διακλάδωσης εχει περάσει το EX στάδιο οπότε εχεί ανανεωθεί η τιμή του pc ,και έχουμε έκδοση ανα 2cc και χειρισμό συμφωνα με την πρώτη περίπτωση.Επειδή στην περίπτωση εκτέλεσης του άλματος έχουνε εκτελεστεί ήδη δύο εντολές ο compiler αναλαμβάνει να μεριμνήσει ώστε μετα την εκτέλεση άλματος η εντολή διακλάδωσης να δείχνει στην σωστή διεύθυνση και να λαμβάνει υπόψη τις δυο εντολές που εκτελέστηκαν ως ενσωματομένες στις λέξεις πριν την ανανέωση της τίμης του μετρητή προγράμματος.

<table>
<tr><th></th><th>INSTRUCTIONS</th><th>1cc</th><th>2cc</th><th>3cc</th><th>4cc</th><th>5cc</th><th>6cc</th><th>7cc</th><th>8cc</th><th>9cc</th><th>10cc</th><th>11cc</th><th>12cc</th></tr>
<tr><td>PC</td><td>Add x1 x0 x0
Beq L1 x1 x0</td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>PC+8</td><td>**L1**: ori x1 x2 1
Addi x2 x1 2</td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|</td><td>ME|</td><td>WB|</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>PC+16</td><td>**L1+4**:lwx24(x1)
Sw x2 4(x1)</td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|</td><td>ME|</td><td>WB|</td><td></td><td></td><td></td><td></td></tr>
<tr><td>**NEW PC**</td><td>Add x1 x1 x2
Or x2 x2 x3</td><td></td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td><td></td><td></td></tr>
<tr><td>**NEW PC +8**</td><td>Add x1 x1 x2
Or x2 x2 x3</td><td></td><td></td><td></td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td></tr>
</table>

Σχήμα 5.4: Παροχέτευση εντολών στα στάδια της υλοποίησης με εντολή διακλάδωσης που πραγματοποιεί άλμα .

<table>
<tr><th></th><th>INSRUCTIONS</th><th>1cc</th><th>2cc</th><th>3cc</th><th>4cc</th><th>5cc</th><th>6cc</th><th>7cc</th><th>8cc</th><th>9cc</th><th>10cc</th><th>11cc</th><th>12cc</th></tr>
<tr><td>PC</td><td>Add x1 x0 x0
Bne L1 x1 x0</td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>PC+8</td><td>**L1**:ori x1 x2 1
Addi x2 x1 2</td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>|
EX</td><td>|
ME</td><td>|
WB</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>PC+16</td><td>**L1+4**: lw x2 x1 0
Sw x2 x1 2</td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>|
EX</td><td>|
ME</td><td>|
WB</td><td></td><td></td><td></td><td></td></tr>
<tr><td>PC+24</td><td>Add x1 x1 x2
Or x2 x2 x3</td><td></td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td><td></td><td></td></tr>
<tr><td>PC+32</td><td>Add x1 x1 x2
Or x2 x2 x3</td><td></td><td></td><td></td><td></td><td></td><td></td><td>IF|
IF</td><td>ID1|
ID2</td><td>EX|
ID2</td><td>ME|
EX</td><td>WB|
ME</td><td>|
WB</td></tr>
</table>

Σχήμα 5.5: Παροχέτευση εντολών στα στάδια της υλοποίησης με εντολή διακλάδωσης που δεν πραγματοποιεί αλμα

Ο μεθοδός υλοποιήσης πετυχαίνει την διαικπεραίωση μιας εντολής ανά κύκλο στην γενική περίπτωση,καθώς υπο κανονικές συνθήκες εκτέλεσης δεν υπάρχει καθυστέρηση κύκλων για την εκτέλεση εντολών διακλάδωσης . Κάτι το οποίο φαινεται και απο τον ρυθμό αλλάγης του μετρητή

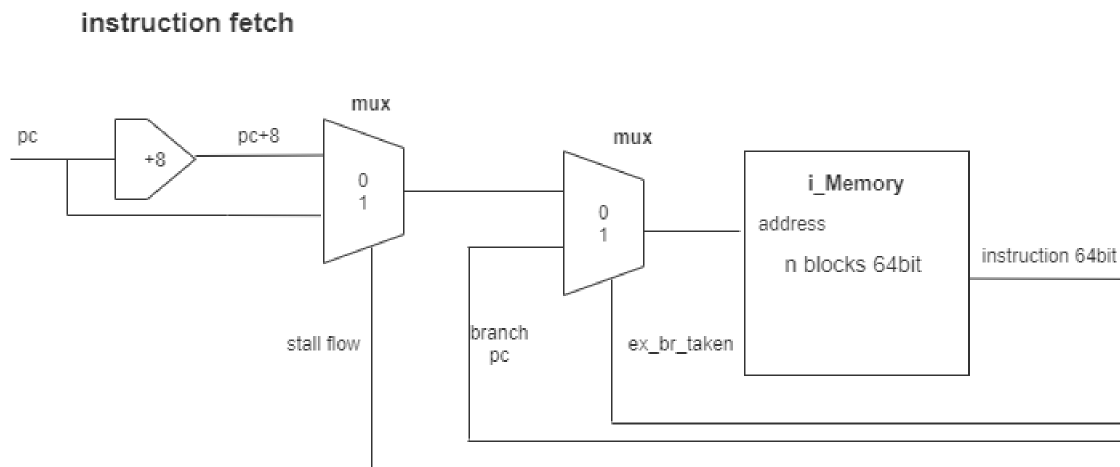
προγράμματος . Οταν έχουμε ανανέωση της τίμης του μετρητή ανα 2cc γίνεται εκδοση δύο εντολών από διαδοχικές θέσεις μνήμης οι οποίες θα εκτελεστουν και οι δύο σειριακά ,οταν εχουμε εκδοση ανα 1cc γίνεται έκδοση δύο εντολών απο τις δύο πιθανές κατευθύνσεις της εντολής διακλάδωσης από τις οποίες θα εκτελεστεί μόνο η μία ανάλογα την αποτίμηση.Η μόνη πιθανή περίπτωση να απαιτείται καθυστέρηση για την διακπεραίωση των εντολών διακλάδωσης είναι η πυκνότητα των εντολών αυτών να μην επιτρέπει στον μεταγλωττιστή την δημιουργία των δύο ζευγών εντολών απο τις δύο κατευθύνσεις, οι οποίες πρέπει να μην αποτελούν εντολές άλματος.Με την περίπτωση αυτη θα ασχοληθούμε στο τελευταίο κεφάλαιο με βάση πραγματικά προγράμματα εκτέλεσης.

5.2 Στάδια υλοποιησής

Ακολούθει αναλυτική περιγραφη της μικροαρχιτεκτόνικης υλοποίησης η οποία έγινε σε γλώσσα Verilog .Για την υλοποίηση ενός pipelined επεξεργαστή που μπορεί να χειρίζεται τις βασικές εντολές του RV32I θα πρέπει να υλοποιηθούν τα βασικά στάδια του Pipeline , οι καταχωρητές που συνδέουν τα στάδια μεταξύ τους και οι απαραίτητες λειτουργικές μονάδες μεταξύ των οποίων η μνήμη εντολών ,η μνήμη δεδομένων ο φάκελος καταχωρητών και η αριθμητική λογική μοναδα .

5.2.1 Instruction fetch unit

Στην παρουσα υλοποίηση το instruction fetch unit αναλαμβάνει την ανανεώση της τιμής του μετρητή προγράμματος και την πρόσβαση στην μνήμη εντολών. Η τιμή του μετρητή ανανεώνεται στην γενική περίπτωση ανά 2 cc μηχανής στην τιμή PC+8.Μπορεί να ανανεωθεί και ανα 1cc στην τιμή PC+8 στην περίπτωση που υπάρχει στα στάδια ID ή EX εντολή διακλάδωσης .Το σήμα που καθορίζει τον κύκλο που ανανεώται η τιμή του μετρητή ειναι το stall_flow που παράγεται απο επόμενο στάδιο οπως θα δούμε παρακάτω. Το σήμα που καθορίζει αν η αποτίμηση της εντολής διακλάδωσης που προηγείται είναι αληθής ή οχι ειναι το ex_br_taken και παράγεται μαζί με την διεύνθυνση του αλματός branch_pc δύο σταδία παρακάτω στο execution unit, στην περίπτωση που ex_br_taken =1 και stall_flow =0 η τιμή του Pc ανανεώνεται στην τιμή branch_pc.Η τίμη του pc δίνει σε κάθε περίπτωση την διεύθυνση προσπέλασης της μνήμης εντολών. Η μνήμη εντολών για τις ανάγκες της εργασίας ειναι υλοποιημένη σαν πίνακας με blocks των 64 bits που φιλοξενούν δύο εντολές ανά block.



Σχήμα 5.6: Instruction fetch unit

5.2.2 Instruction Decode unit

Σκοπός του σταδίου αυτού είναι ο χειρισμός της 64bit λέξης που προωθείται απο το fetch unit, η αποκωδικοποιήση των δύο εντολών που περιέχει και η δημιουργία των κατάλληλων σημάτων απαραίτητα για την εκτέλεση της κάθε εντολής αλλά επισής και του σήματος stall flow που καθορίζει τον χρόνο της επόμενης εκδοσής απο το fetch unit οπως είδαμε προηγουμένως. Επίσης σε αυτό το στάδιο γίνεται η ανίχνευση για πιθανούς κινδύνους εξαρτήσεων δεδομένων μεταξυ των εντολών του σταδίου και των σταδίων που προηγούνται . Οι κίνδυνοι εξαρτήσεων τύπου RAW επιλύονται με κατάλληλη μονάδα προώθησης .Τέλος στο στάδιο αυτο γίνεται η προσπέλαση του Φακέλου Καταχωρητών και η αναγνωσή των τελούμενων εισόδου αν ειναι απαραίτητο. Το decode module ειναι κοινό για τις δυο εντολές που αποτελούν την λεξη 64Bit που προωθείται απο το στάδιο της έκδοσης (fetch) καλείται ταυτοχρονα δύο φορές μια για τα στοιχεία if_instr[31:0] και μια για τα στοιχεία if_instr[63:32],πρώτη και δεύτερη εντολή της λέξης αντίστοιχα. Σύμφωνα με όσα αναφερθήκαν στο κεφάλαιο περιγραφης του ISA το decode module αποδικωποιεί με βάση το πρότυπο του RISCV32I την 32 Bit λεξη εντολής . Τα βασικά πεδία της εντόλης if_instr που ειναι το Opcode ,Funct3,Funct5 αποτελούν τα κριτήρια με τα οποία οι πολυπλέκτες δίνουν τιμή στα σήματα διάκρισης για τις επιμέρους εντολές του ISA ,που περιγράφονται στο Σχ 4.5. και 4.6.

Alucode	Είναι ενα πεδίο μεγέθους 6 bit και αποτελεί το διακριτικό που θα χρησιμοποιησει η alu στο execution stage για να εκτελέσει την κατάλληλη πράξη ανάλογα την εντολή .
Reg_we	Ειναι ενα σήμα του 1Bit που υποδηλώνει αν η εντολή γράφει ή οχι σε κάποιο καταχωρητή.
Is_store	Είναι ενα σήμα του 1Bit που υποδηλώνει αν η εντολή είναι εντολή αποθήκευσης στην μνήμη
Is_load	Ειναι ενα σήμα του 1Bit που υποδηλώνει αν η εντολή είναι εντολή φορτωσης από την μνήμη .
Aluop1_type	Είναι το πεδίο μεγέθους 2 Bit που υποδήλωνει τον τύπο του πρώτου τελούμενου εισόδου ,οι τιμές που παίρνει ειναι REG αν προκειταί για καταχωρητη, IMM αν πρόκειται για σταθέρα ,PC αν λαμβάνει την τιμή του μετρητή προγράμματος και NONE αν δεν υπάρχει πρώτο τελούμενο στην παρούσα εντολή .
Aluop2_type	Είναι το πεδίο μεγέθους 2 Bit που υποδήλωνει τον τύπο του δεύτερου τελούμενου εισόδου ,οι τιμές που παίρνει ειναι REG αν προκειταί για καταχωρητη, IMM αν προκειται για σταθέρα ,PC αν λαμβάνει την τιμή του μετρητή προγράμματος και NONE αν δεν υπάρχει δεύτερο τελούμενο στην παρούσα εντολή .
Op_type	Είναι το πεδίο μεγέθους 3Bit που κατατάσει τις εντολές στους βασικούς τύπους εντολών συμφωνα με το πρότυπο του ISA , οι τιμές που παίρνει ειναι TYPE_U, TYPE_J, TYPE_I, TYPE_B ,TYPE_S και TYPE_R.Αποτελούν σημαντικά πεδία γιατί αποτελουν κριτήριο για το αν υπάρχει εγκαίρη διεύθυνση προσπέλασης του Φακέλου καταχωρητητών στην παρούσα εντολή η αν πρέπει να μηδενιστεί και επίσης γίνεται στο χτίσιμο της σταθέρας imm που βρίσκεται σε διαφορετικές θέσεις αναλογα τον τύπο της εντολής.

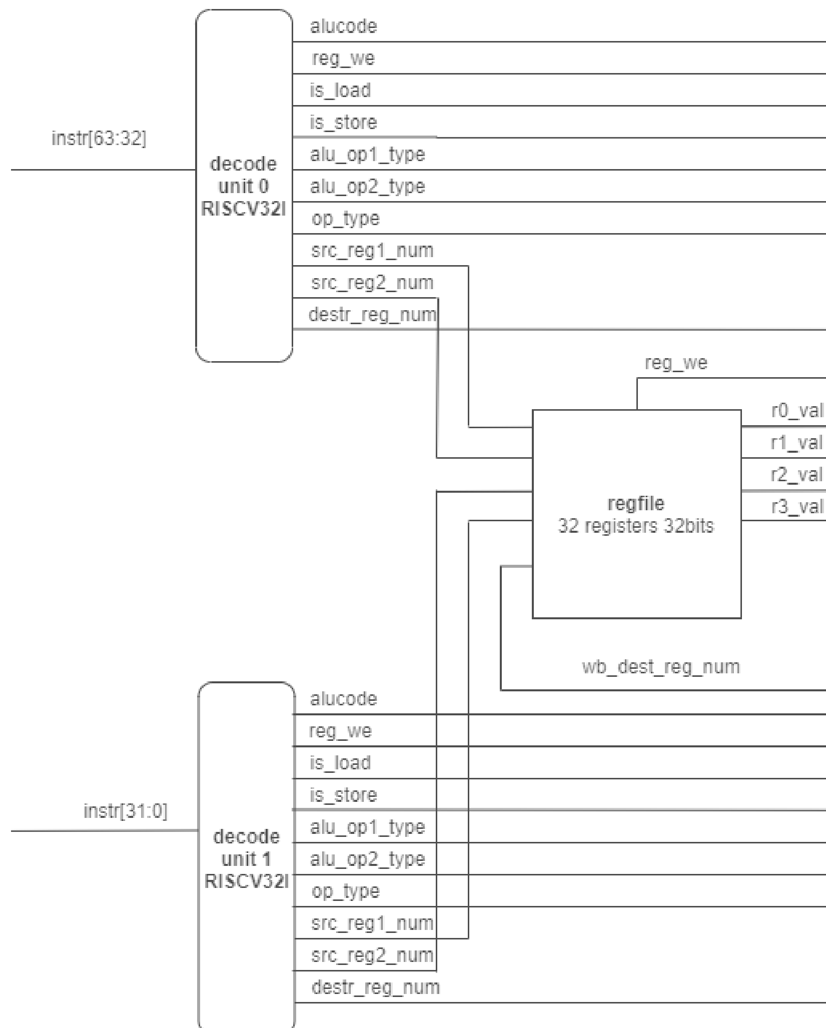
Σχήμα 5.7: Βασικά σήματα διάκρισης αποκωδικοποίσης RV32I

srcreg1_num	Είναι η διευθυνση προσπέλασης του φακέλου καταχωρητων για το πρώτο τελούμενο .
srcreg2_num	Είναι η διεύθυνση προσπέλασης του φακέλου καταχωρητών για το δεύτερο τελούμενο .
dstreg_num	Είναι η διεύθυνση προορισμού του φακέλου καταχωρητών σε περίπτωση εγγραφή

Σχήμα 5.8: Τα πέδια που λαμβάνουν τιμή με βάση το $Op_{t}ype$

Ο φακέλος καταχωρητών αποτελεί εναν πίνακα 32 θεσεων με block των 32 Bits.Διαθέτει τέσ-

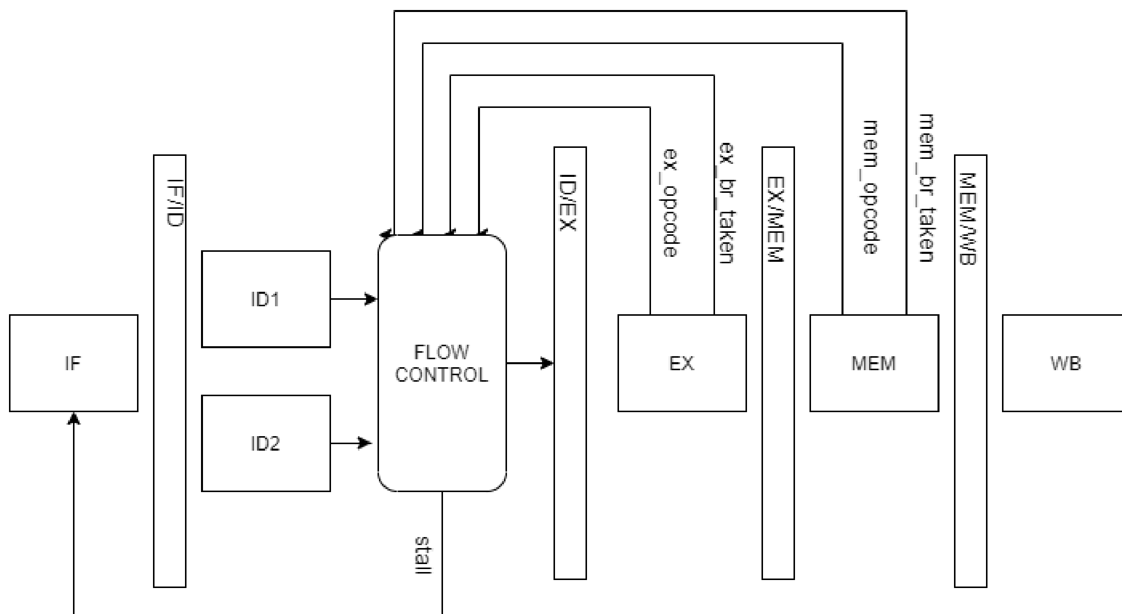
σερεις θέσεις ανάγνωσης ώστε να μπορεί να ικανοποιήσει την ανάγνωση των τελούμενων και των δύο εντολών ταυτόχρονα ,λαμβάνει σαν εισόδους τις διευθύνσεις και επιστρέφει τις τιμές για κάθε καταχωρήτη.Η θέση εγγραφής είναι μια οπώς και στην κλασσική υλοποιήση καθώς δεν υπάρχει αναγκή για ταυτόχρονη εγγραφή περισσοτέρων τελούμενων. Η εγγραφή είναι σύγχρονη και λαμβάνει χώρα στην περίπτωση αληθούς σήματος ενεργοποίησης. Με την διπλή κλήση του decode module για τις δύο εντολές έχουμε ταυτόχρονη αποκωδικοποίηση των εντολών που θα προωθηθούν στο στάδιο εκτέλεσης .



Σχήμα 5.9: Μονάδες αποδικωποίησης κατα RV32I και αναγνωση τελούμενων εισόδου .

5.2.3 Flow control unit

Για την επίτευξη αυτού του σκόπου δημιουργηθήκε το flow_control_module το οποίο αναλαμβάνει τον ρόλο της παροχετεύεσης μεταξύ του decode stage και του execution stage.Λαμβάνει σαν εισόδους τα αποτελέσματα της αποκωδικοποίσης και των δύο εντολών καθώς και τις τιμές των τελούμενων εισόδου που προέρχονται απο το φάκελο καταχωρητών ή απο τις μονάδες προώθησης σε περίπτωση εξαρτήσεων τυπου RAW.Οι μονάδες προώθησης θα αναλυθούν στην συνέχεια.



Σχήμα 5.10: Μορφή σταδίων υλοποίησης με την τοποθέτηση του flow_control

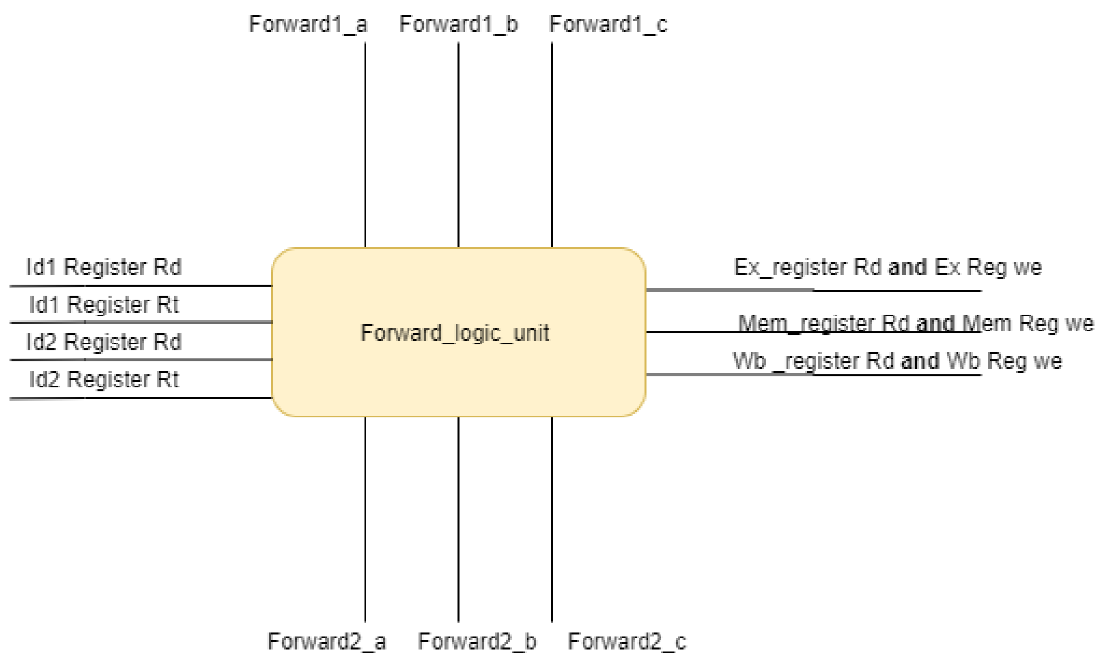
Ο καίριος ρόλος του flow control unit έγκειται στην παροχετεύση στο execution stage της σωστής εντολής μεταξυ τών δύο που προκύπτουν απο τις μονάδες αποκωδικοποιήσης,καθώς και την παραγωγή του σήματος stall_flow που καθορίζει σε ποιο κύκλο θα γίνει η επόμενη έκδοση εντολών. Επιπλέον σήματα που λαμβάνει απο τα επόμενα σταδια του pipeline ειναι τα ex_opcode και mem_opcode για τον έλεγχο της εντολής που βρίσκεται στο execution stage και στο Memory stage αντίστοιχα και το ex_br_taken και mem_br_taken που η γίνονται αληθή μόνο στην περίπτωση που η εντολή που βρίσκεται στο στάδιο execution και memory αντίστοιχα ειναι εντολή άλματος και έχει αποτιμηθεί ως αληθής.

Ακολουθεί αναλυτική περιγραφή της λειτουργικότητας της μονάδας η οποία αποτελείτα απο τις εξης περιπτώσεις.
i)Σε περιπτωση που καμία απο τις εντολές που βρισκονται στο στάδιο ID EX και MEM δεν είναι εντολή διακλάδωσης προωθείται στο execution stage αρχικά η πρώτη εντολή ID1 με ταυτόχρονο πάγωμα της έκδοσης καινούριου μπλοκ εντολών stall_flow =1 και στην συνέχεια η δεύτερη ID2 με ξεπάγωμα της έκδοσης καινούριου μπλοκ εντολών stall_flow =0.
ii)Στην περιπτωση που υπάρχει εντολή διακλάδωσης στο σταδιο ID και συγκεκριμένα στην θέση ID2, ισχύει η παραδοχή οτι ο compiler θα τοποθετεί πάντα την εντολή διακλάδωσης στην δεύτερη εντολή της λέξης .Τότε αποστέλεται η πρώτη εντολή στο execution stage αλλα έχουμε ξεπάγωμα έκδοσης καινούριου μπλοκ εντολών στον επόμενο κύκλο στην συνέχεια αποστέλεται η δεύτερη εντολή διακλάδωσης , και το ξεπάγωμα διατηρείται .Στον επόμενο κύκλο που η εντολή διακλαδωσης θα περάσει στο execution stage και θα αποτιμηθεί αν πραγματοποιεί ή όχι το άλμα με την τιμή του σήματος ex_br_taken.Ανάλογα την τιμή του σήματος θεωρούμε οτι η πρωτη εντολή εκτελείται για εκτέλεση αλματος και η δεύτερη για μή εκτέλεση ,οποτε για ex_br_taken =1 στέλνουμε στο execution stage την εντολή id_instr1 και για ex_br_taken=0 στέλνουμε την εντολή id_instr2.Σε

αυτό το σημείο χρειάζεται να παγώσουμε ξανά την έκδοση εντολών καθώς εχούν εκδοθεί ανα 1cc οι δύο λεξεις που περιλαμβάνουν εντολές απο δύο κατευθύνσεις απο τις οποίες θα εκτέλεστει η μια εντολή απο κάθε λέξη οπώς είπαμε παραπάνω.Στον επόμενο κύκλο η εντολή διακλάδωσης έχει περάσει στο mem_stage οποτε θα αποστείλουμε στο ex_stage την μία απο τις δύο εντολές του id_stage αναλογα με την τίμη του σηματος mem_br_taken. Επίσης θα ξεπαγώσουμε την εκδόση εντολών για τον επόμενο κύκλο.Ισχύει η παραδοχή οτι τα δύο block εντολών που εκδίδονται ανά κύκλο μετα την ανίχνευση της εντολής διακλάδωσης δεν περιέχουνε αυτά εντολές διακλάδωσης .

5.2.4 Forward unit

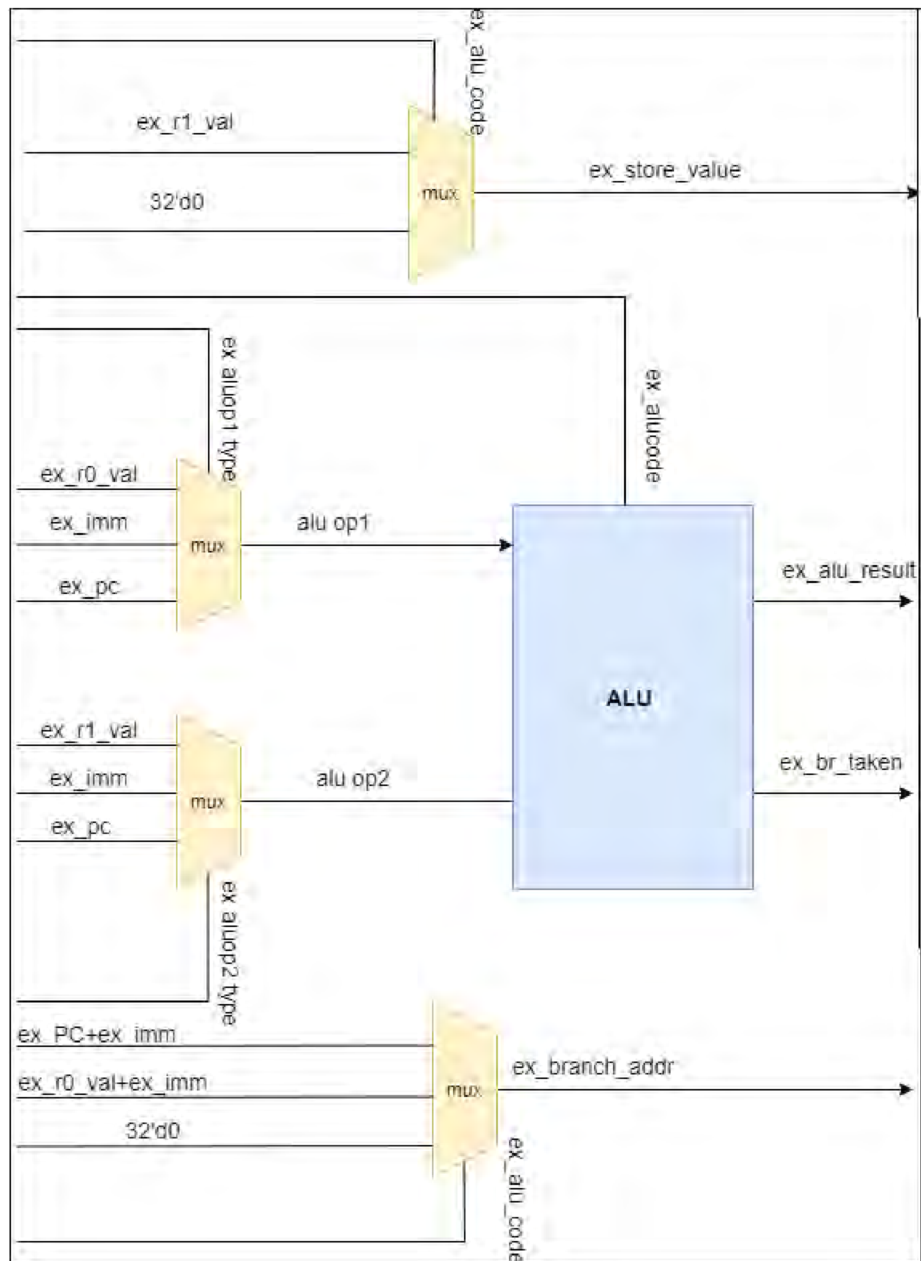
Στις υλοποίησεις που χρησιμοποιούν την τεχνική της παροχετεύτεσης προκύπτουν εξαρτήσεις δεδομένων.Λόγω της απλότητας υλοποίησης βασισμένη στην κλασσίκη RISC Pipeline ο τυπός των εξαρτήσεων δεδομένων που συναντόνται στην παρούσα υλοποίηση είναι μονο εξαρτήσεις τύπου ΑΜΕ(RAW),ανάγνωση μετά απο εγγραφή.Δεν παρατήρούνται εξαρτήσεις τύπου ΕΜΕ ,εγγραφή μέτα απο εγγραφή , και τύπου ΕΜΑ,εγγραφή μετα απο ανάγνωση καθώς η έκδοση και η διεκπεραίωση κάθε εντολής πραγματοποείται εντός σειράς (in order) . Οι εξαρτήσεις που πρέπει να αντιμετωπιστούν περιορίζονται στην περίπτωση μια εντολή να χρειάζεται κάποιο καταχωρητή η μια θέση μνήμης που γράφεται απο μια εντολή που βρίσκεται ακόμα εντός του pipeline και δεν εχει περάσει το στάδιο εγγραφής.Επειδή η ανίχνευση για εξαρτήσεις κάθε εντολής γίνεται στο στάδιο της αποκωδικοποίησης και σε αυτό το στάδιο η υλοποίηση μας χειρίζεται δύο εντολές θα πρέπει η μονάδα προώθησης που σχεδιαστηκέ να ανιχνεύει και να επιλύει σε κάθε κύκλο τις εξαρτήσεις που προκύπτουν και για τις δύο εντόλες ανεξάρτητα απο ποια θα αποσταλεί για εκτέλεση. . Σκοπός της μοναδας forward unit είναι να ελέγξει αν οι καταχωρητές εισόδου των εντολών που βρίσκονται στο decode stage ταυτίζονται με καποιόν απο τους καταχωρητές προορισμού των εντολών που βρίσκονται στα επόμενα στάδια.Σε περίπτωση που ανιχνευτεί εξάρτηση απο εντολή που βρίσκεται στο execution stage χρησιμοποείται μια μεταβλητη των 2bit για να καθοριστεί αναλυτικά η εξάρτηση,δηλαδή αν πρόκειται για κοινούς και τους δύο καταχωρητές εισόδου με τον καταχωρητή προορισμού ή έναν απο τους δύο.Η ίδια διαδικασία επαναλαμβάνεται για την ανίχνευση εξαρτήσεων μεταξύ του decode stage και του memory stage και μεταξυ του decode stage και του writeback stage.



Σχήμα 5.11: Forward logic unit

5.2.5 Execution stage

Η βασική λειτούργια του σταδίου αυτού ειναι η εκτέλεση της αριθμητικής ή λογικής πράξης μεταξυ των δύο τελούμενων εισόδου .Η πράξη αυτη πραγματοποιείται στην alu.Ο τύπος της πράξης καθορίζεται απο το σήμα alucode που έχει προκύψει απο την αποκωδικοποίση .Τα τελούμενα είσοδου ανάλογα με τα σήματα aluop1 και aluop2 μπορεί να λαμβάνουν τιμή απο το φάκελο καταχωρητών ή τις μονάδες προώθησης αλλά μπόρει να αποδίδεται η τιμή της σταθέρας Imm ή τιμή του μετρητή προγράμματος.Η ALU εκτός απο το aluresult το αποτέσμα της πράξης των δύο τελούμενων παράγει και το σήμα ex_br_taken το οποίο δηλώνει πραγματοποιήση άλματος είτε αν έχουμε εντολή διακλάδωσης χωρίς συνθήκη είτε αν έχουμε εντολή διακλάδωσης με συνθήκη και αυτή έχει αποτιμηθεί ως αληθής.Η διεύθυνση του άλματος υπολογίζεται επίσης απο το execution stage χρησιμοποιώντας το σήμα alucode για να διαχωριστεί αν ο στόχος είναι εξαρτώμενος απο καταχωρήτη και την προσθήκη σταθερας (JALR) η αν είναι εξαρτώμενος απο τον μετρητή προγράμματος και την προσθήκη σταθεράς σε ολές τις άλλες εντολές άλματος . Τέλος στο execution stage λαμβάνει τιμή η μεταβλητή αποθήκευσης στην μνήμη.



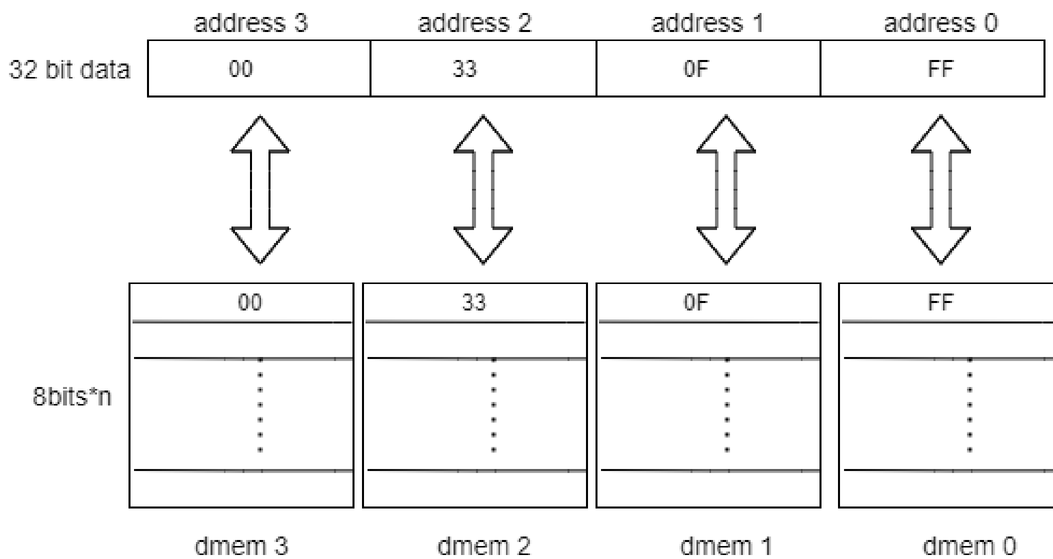
Σχήμα 5.12: Διάγραμμα Execution unit

<table>
<tr><th>ALUCODE</th><th>ALU_RESULT</th><th>BR_TAKEN</th></tr>
<tr><td>ALU_LUI</td><td>Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_JAL,ALU_JALR</td><td>Op2+32’d4</td><td>ENABLE</td></tr>
<tr><td>ALU_BEQ</td><td>32’d0</td><td>Op1==Op2</td></tr>
<tr><td>ALU_BNE</td><td>32’d0</td><td>Op1!=Op2</td></tr>
<tr><td>ALU_BLT</td><td>32’d0</td><td>Signed_Op1<Signed_Op2</td></tr>
<tr><td>ALU_BGE</td><td>32’d0</td><td>Signed_Op1>=Signed_Op2</td></tr>
<tr><td>ALU_BLTU</td><td>32’d0</td><td>Op1<Op2</td></tr>
<tr><td>ALU_BGEU</td><td>32’d0</td><td>Op1>=Op2</td></tr>
<tr><td>ALU_LB, ALU_LH,ALU_LW,
ALU_LBU,ALU_LHU,ALU_SB,
ALU_SH,ALU_SW</td><td>Op1+Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_ADD</td><td>Op1+Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_SUB</td><td>Op1-Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_SLT</td><td>Signed_Op1<Signed_Op2 ?
32’d1: 32’d0</td><td>DISABLE</td></tr>
<tr><td>ALU_SLTU</td><td>Op1<Op2 ? 32’d1:32’d0</td><td>DISABLE</td></tr>
<tr><td>ALU_XOR</td><td>Op1^Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_OR</td><td>Op1|Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_AND</td><td>Op1 & Op2</td><td>DISABLE</td></tr>
<tr><td>ALU_SLL</td><td>Op1<<Op2[4:0]</td><td>DISABLE</td></tr>
<tr><td>ALU_SLR</td><td>Op1>>Op2[4:0]</td><td>DISABLE</td></tr>
<tr><td>ALU_SRA</td><td>Signed_Op1>>>Signed_Op2[4:0]</td><td>DISABLE</td></tr>
</table>

Σχήμα 5.13: Μονάδα ALU.Αποτίμηση ALU_RESULT και BR_TAKEN με βάση την τιμή του πεδίου ALUCODE

5.2.6 Memory stage

Η λειτουργία του Memory stage είναι η πρόσβαση στην μνήμη δεδομένων είτε για εγγραφή στην περίπτωση εντολών Save είτε για ανάγνωση στην περίπτωση εντολών Load .Η ανάγνωση και η εγγραφή ενεργοποιείται απο αντίστοιχα σήματα που προκύπτουν απο το στάδιο της αποκωδικοποίησης is load και is store. Η ανάγνωση και η εγγραφή θα πρέπει να πραγματοποιείται για ολόκληρη λέξη 32 bits ,για μισή 16 Bits και για 1 byte(8 bits) για το σκοπό η μνήμη δεδομένων αποτελέιται απο 4 στιγμιότυπα μιας μνήμης με μέγεθος κελιού 1 byte ,συνδιάζοντας και τα 4 στιγμιότυπα προκύπτει μια πλήρη λέξη των 32 Bits ή 4 bytes .

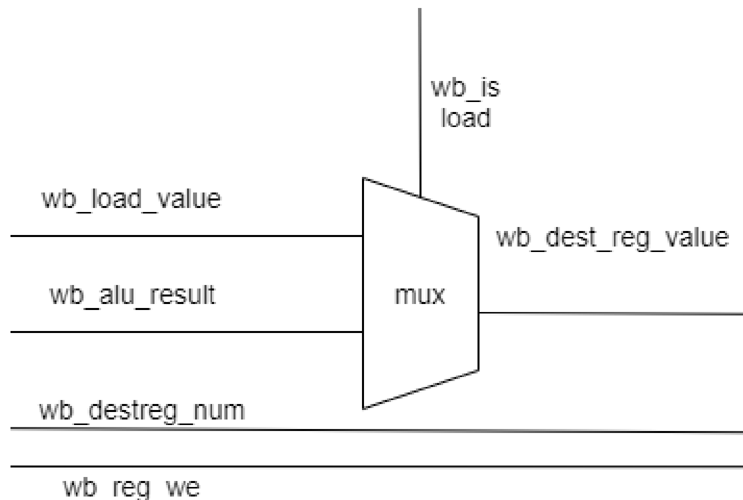


Σχήμα 5.14: Μνήμη δεδομένων και κατανομή δεδομένων για εγγραφή ή ανάγνωση σε αυτή.

Ας εξετάσουμε αναλυτικά την μεθόδο υλοποίησης για εντολές sw ,sb,sh και lw,lb,lh Για την υλοποίηση των εντολών αποθήκευσης απαιτούνται τα δεδομενα προς αποθήκευση τα οποία πρωθούνται απο το execution stage στην μεταβλητή store_value και η διεύθυνση αποθήκευσης η οποία προκύπτει ως αποτέλεσμα της alu.Ανάλογα αν έχουμε εντολή πλήρης αποθήκευσης μισής ή ενος byte ενεργοποιούνται τα αντίστοιχα στιγμιότυπα της μνήμης δεδομένων .Για sw γράφουμε σε όλα ,για sh θα γράψουμε σε δύο και για sb σε ένα .Η επιλογή του σωστού στιγμιότυπου για εγγραφή γίνεται ανάλογα τα δύο πρώτα Bits της διεύθυνσης εγγραφής.Τα χρήσιμα δεδομένα προς εγγραφή σε περίπτωση sw αποτελούνται και απο τα 32 bits της μεταβλητής store_value ενω σε περίπτωση sh και sb αποτελούνται απο τα 16 και τα 8 αντίστοιχα λιγότερο σημαντικά ψηφια της μεταβλητής store_value που συνθέτουν τα bytes εγγραφής.
Στην περίπτωση εντολών load η διεύθυνση ανάγνωσης είναι πάλι το αποτέλεσμα της Alu απο το Execution Unit . Με βάση αυτην την διεύθυνση γίνεται η ανάγνωση ενός byte απο κάθε στιγμιότυπο της μνήμης δεδομένων . Σε εντολή Lw χρησιμοποιούνται και τα 4 Bytes τα οποία συνθέτουν αποτέλεσμα ανάγνωσης .Σε εντολή lh χρησιμοποιούνται τα δύο bytes ανάλογα με αρχίκα 2 ψηφία της διεύθυνσης ανάγνωσης και σε εντολή Lb χρησιμοποιείται το ένα πάλι ανάλογα τα 2 αρχικά ψηφία της διεύθυνσης ανάνγνωσης.Σε κάθε περίπτωση το τελικό απότελεσμα πρέπει να είναι μεγέθους 32Bits οπότε πραγματοποιείται προσημασμένη επέκταση για τις εντολές Lb και lh και μη προσημασμένη επέκταση για τις εντολές Lbu και Lhu.

5.2.7 Write back stage

Το πιο απλό απο τα σταδία υποποίησης η μόνη λειτουργία είναι η εγγραφή αν απαιτείται στον φάκελο καταχωρητών.Επιλέγει μέσω ενος πολυπλέκτη αν το αποτέλεσμα εγγραφης προέρχεται απο το αποτέλεσμα της alu η αν προέρχεται απο την ανάγνωση της μνήμης και καλεί τον φάκελο καταχωρητών.



Σχήμα 5.15: Write back stage

5.3 Πρωτοτυπία υλοποίησης

Η πρωτοτυπία της τρέχουσας υλοποίησης σε σχέση με άλλες τεχνίκες που έχουν προταθεί μπορεί να συνοψιστεί.

- Στην δημιουργία ενός εξυπνού μεταγλωτιστή που θα δημιουργεί τα κατάλληλα ζεύγη εντολών και θα τα τοποθετεί στην μνήμη.Σκόπος του είναι στην φάση της μεταγλώτισσης να αναγνωρίζει τις εντολές διακλάδωσης υπο συνθήκη ,να εντοπίζει την διεύθυνση του αλματός και εφόσον το επιτρέπουν οι περιορισμοί να δημιουργεί τα δύο επομενα μπλοκ εντολών με τις εντολές των δύο κατευθύνσεων τα μπλοκ αυτα τα τοποθετεί στην μνήμη ακριβώς μετά την εντολή διακλάδωσης .Συνεπώς στην κανονικη ροή ροή εκτέλεσης γίνεται έκδοση στον επεξεργαστή ταυτοχρόνα εντολές και για αληθή και για ψευδή αποτίμηση. Στην γενική περίπτωση τα μπλοκ εντολών αποτελούνται απο εντολές που θα ανήκαν σε διαδοχικες θέσεις μνήμης.Η ιδέα δημιουργίας μπλόκ εντολών δύο κατευθύνσεων και η συγχώνευση των διαφορετικών θέσεων μνήμης μιας κλάσικης υλοποίησης σε μία στο επιπέδο του μεταγλωτιστή κάνει πολύ απλούστερη την σχέδιαση του υλικού του επεξεργαστη και εξοικονομεί κύκλους μηχανής κάθως δεν χρειάζεται να αποδικωποιεί πρώτα την εντολή διακλάδωσης και επείτα να κάνει ταυτόχρονη έκδοση απο δύο θέσεις μνήμης.Επίσης υλοποιήσεις που ακολουθούν αύτη την τεχνική αντιμετωπίζουν πρόβλημα στην αποδοτική υλοποίηση μια μνήμης cache που προσφέρει την δυνατότητα πρόσβασης σε δύο απόμακρισμένες μεταξύ τους διευθύνσεις.
- Η δευτερη ιδιότητα της υλοποιήσης μας είναι στην διαχείριση διπλών εντολών και στην γενική περίπτωση και όχι διπλή έκδοση μόνο μετά απο εντολή διακλάδωσης . Επειδή ο επεξεργαστής διαθέτει διπλή μονάδα αποκωδικοποίησης και λόγω της παραδοχής οτι ο μεταγλωττιστής θα τοποθετεί πάντα τυχόν εντολή διακλάδωσης στην δεύτερη θέση του μπλοκ επιτυγχάνεται η έγκαιρη αποτίμηση των κατάλληλων σημάτων που καθορίζουν τον χρονισμό του επεξεργαστή,δηλαδή σε ποιο κύκλο θα κάνει έκδοση καινούριου μπλοκ χωρίς να

χρείαζεται να παγώσει το στάδιο εκτέλεσης κάθως πάντα υπάρχει διαθέσιμη εντολή να προωθηθεί . Σύνεπως επιλύει με επιτύχια το πρόβλημα των εντολών διακλάδωσης μην απαιτώντας κάνενα κύκλο καθυστέρησης, χώρις να απαιτείται σε καμία περίπτωση να γίνει άδειασμα άχρηστων εντολών και χωρίς μονάδα πρόβλεψης διακλαδώσεων .Άκομη δεν εκτέλει εντολές που στην συνέχεια δεν θα καταχωρίσει κάθως οι επιπλέον εντολές φτάνουν εως το στάδιο αποκωδικοποίησης .

Δεν θα γινότανε όμως να μην υπαρχουνε περιορισμοί σε μια πρωτότυπη υλοποίηση. Ο βασικός περιορισμός εγκειται στο γεγονός οτί μετα απο εντολή διακλάδωσης ο μεταγλωττισής πρέπει να εντοπίσει και στις δύο κατευθύνσεις απο δύο εντολές που δεν είναι εντολές διακλάδωσης και μπορούν να προωθηθούν με ασφάλεια στον επεξεργαστή όσο αποτιμάται η αρχική εντολή διακλάδωσης. Αυτός ο περιορισμός αποτελεί πρόβλημα μόνο στην περίπτωση ενός προγράμματος με πολύ μεγάλη περιεκτικότητα σε εντολές διακλάδωσης η σε διαδοχικές εντολές διακάδωσης .Σε αυτή την περίπτωση απαιτείται πάγωμα της παρόχετευσης και σίγουρα κάποιοι κύκλοι καθυστέρησης .Θα μπορούσε να αποτελέσει αντικείμενο μελλοντικής εργασίας η διαχείριση τέτοιων περιπτώσεων.

Κεφάλαιο 6

Επαλήθευση λειτουργίας

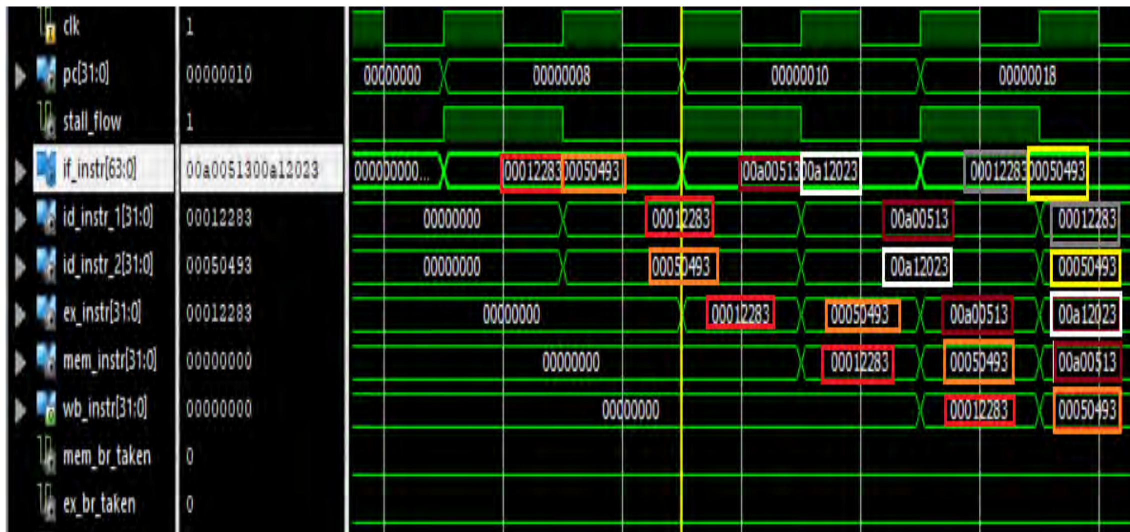
6.1 Προσομοίωση

Για τον έλεγχο της λειτουργίας της παράπανω υλοποίησης χρησιμοποιήθηκε ο προσομοιωτής που παρέχει το εργαλείο της Xilinx Ise design suite.Κάθως το εργαλείο παρέχει και την δυνατότητα επαλήθευσης σε πραγματικό περιβάλλον με την χρήση Fpga (πλατφορμα επαναπρογραμματιζόμενης λογικης) θα μπορούσε μελλοντικά νεα γίνει πειραματικά προσπάθεια τοποθέτησης του επεξεργαστή σε μια σε μια τέτοια πλατφόρμα με τους που διαθέτει τους κατάλληλους πόρους δηλαδή ικανό αριθμό Luts .Στα πλαίσια αυτής της εργασίας θα ελέγχουμε μέσω του προσομοιωτή για την λειτουργικότητα σε επίπεδο διοχέτευσης των διπλών εντολών που αποτελεί και την πρωτοτυπία της υλοποίησης αλλά και την σωστή αποτίμηση όλων των εντολών που περιλαμβάνονται στο RISC V I ISA καθώς και τα θέματα που αφορούν τις μονάδες προώθησης.

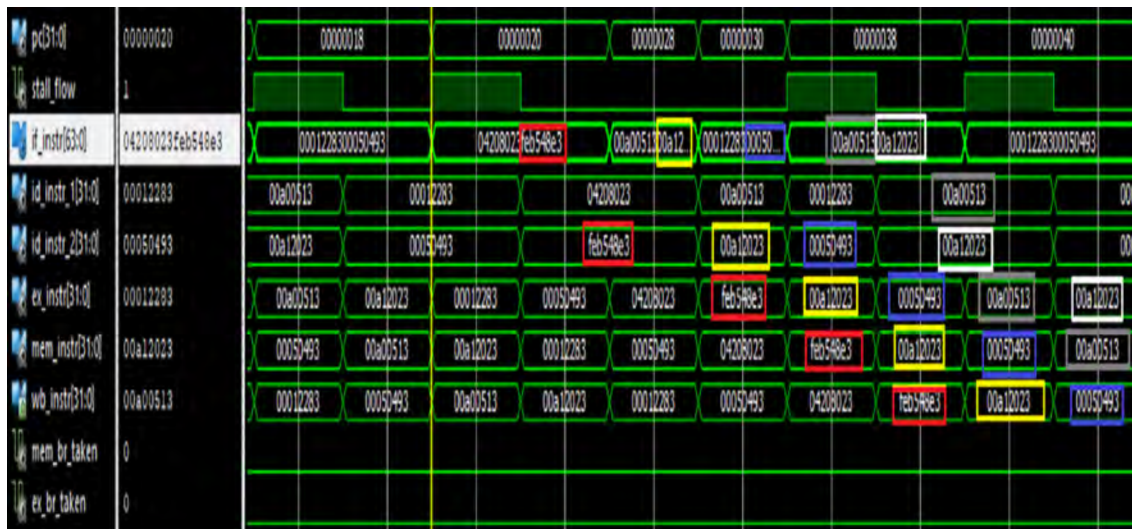
6.2 Έλεγχος διπλής έκδοσης

Στο Σχημα 6.1 μπορούμε να διακρίνουμε την γενική περίπτωση λειτουργίας του επεξεργαστή. Η έκδοση καινούριας λέξης εντολής στο if_stage πραγματοποιείται ανα 2cc,κάτι το οποίο ελέγχεται απο την άνοδο του σήματος stall_flow. Η λέξη εντολής αποτελείται απο δύο εντολές των 32bits οι οποίες στον επόμενο κύκλο διοχετεύονται στις δύο μονάδες αποκωδικόποιησης.Στην συγκεκριμένη περίπτωση που δεν υπάρχει εντολή διακλάδωσης σε κανένα απο τα στάδια id ,ex και mem , οι δύο εντολές προωθούνται στα επόμενα στάδια μια σε κάθε κύκλο .

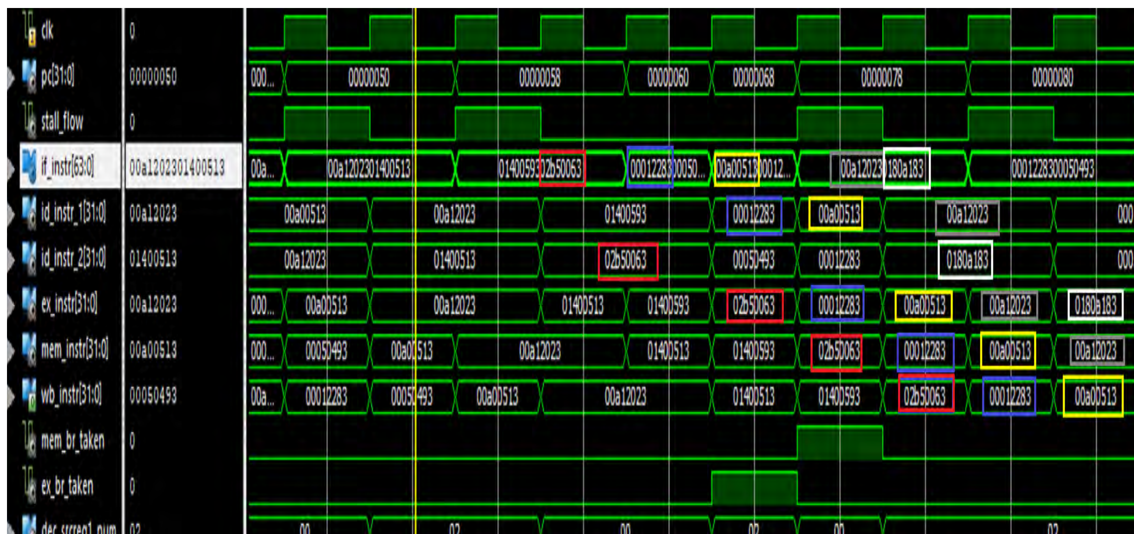
Στο Σχήμα 6.2 η εντολή που είναι μαρκαρισμένη με κόκκινο χρώμα αποτελεί εντολή διακλάδωσης ,αυτό σημαίνει οτι τα δύο επόμενα block εντολών η εκδοσή θα γίνει ανά 1cc ,καθώς περιλαμβάνουν εντολές για αληθή και ψευδή αποτίμηση της εντολής διακλάδωσης ,για να επιτευχθεί συνέχεια εκτέλεσης χωρίς χαμένο κύκλο και CPI =1. Οι εντολές διοχετεύονται στο id stage και κατόπιν μόνο οι χρωματισμένες εντολές με κίκτρινο και μπλε χρώμα θα διοχετευτούν στα επόμενα στάδια ,δηλαδή οι εντολές που βρίσκονται στην μοναδα id_2 . Η επιλογή έγινε καθώς η εντολή διακλάδωσης αποτιμήθηκε ψευδή με το σήμα ex_br_taken να παραμένει 0.Στην συνέχεια έχουμε έκδοση εντολών και διοχεύτεση σύμφωνα με την αρχική περίπτωση , είναι οι εντολές με γκρι και λευκό χρώμα . Στο Σχήμα 6.3 η εντολή δικλάδωσης είναι μαρκαρισμένη με κόκκινο



Σχήμα 6.1: Διπλή έκδοση χωρίς εντολή διακλάδωσης



Σχήμα 6.2: Έκδοση και εκτέλεση εντολών μετα απο εντολή διακλάδωσης με ψευδή αποτίμηση



Σχήμα 6.3: Έκδοση και εκτέλεση εντολών μετά απο εντολή διακλάδωσης με αληθή αποτίμηση

χρώμα .Στην συνέχεια παρατηρούμε έκδοση ανα 1cc για τα δύο επόμενα block και εκτέλεση των εντολής που βρίσκενται στην θέση Id1 και είναι μαρκαρισμένες με μπλε και κίτρινο χρώμα , αυτό συμβαίνει επείδη η εντολή διακλάδωσης αποτιμήθηκε ως αληθής κάτι που φαινεται με την άνοδο του σήματος ex_br_taken . Κατόπιν η τιμή του μετρητή προγράμματος αλλάζει απο x68 σε x78 κάτι που φανερώνει την πραγματοποίηση του άλματος.Ο μεταγλωττιστής έχει λάβει υπόψην της δύο εντολές που εκτελέστηκαν στην διέυθυνση του άλματος πρωτού αλλαξει ο μετρητής προγράμματος.Στην συνέχεια οι επόμενες εντολές εκτέλούνται σύμφωνα με την γενική περίπτωση . Σε κάθε κύκλο ολοκληρώνεται μια εντολή σε ολές τις παραπάνω περιπτώσεις είτε υπάρχει εντολή διακλάδωσης είτε οχι . Στην συνέχεια θα ελέγξουμε για την ορθή αποτίμηση των εντολών και την λειτουργικότητα των μονάδων προώθησης σε συγκεκριμένους Risc-v assembly κώδικες.

6.3 Παραδείγματα ελέγχου

6.3.1 Risc-v assembly χωρίς εντολές διακλάδωσης I

```
nop
nop
addi x8 x2 48
sw x10 -24(x8)
addi x15 x15 1
sub x2 x2 x15
addi x11 x0 6
addi x10 x0 5
add x12 x10 x11
addi x10 x0 10
addi x10 x0 10
addi x8,x8,-4
lw x2 -20(x8)
addi x8,x8,-4
addi x8 x2 48
addi x9 x10 0
```

Σχήμα 6.4: Παράδειγμα Risc-v assembly

```
0000000000000000
03010413FEA42423
0017879340F10133
0060059300500513
00B5063300A00513
00A00513FFC40413
FEC42103FFC40413
0301041300050493
```

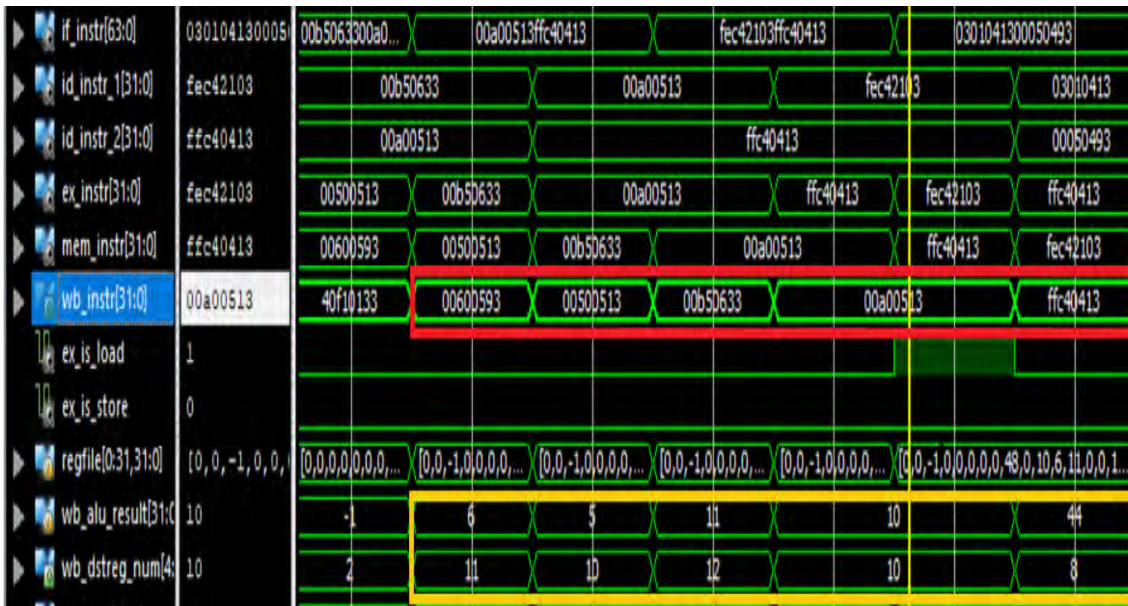
Σχήμα 6.5: Μετατροπή σε Hex

Σχήμα 6.6: Αποτελέσματα προσομοίωσης

Σύμφωνα με το Σχήμα 6.6 αρχικά όλοι οι καταχωρητές είναι αρχικοποιημένοι σε 0. Παρατηρούμε η εντολη addi x8 x2 48 γραφει στην θέση 8 του φακέλου καταχωρητών την τιμή 48.Η εντολή sw x10 -24(x8) θα πρέπει να γραψει την τιμή 0 στο 24 byte της μνήμης και καθώς κάθε θέση της μνήμης αντιστοιχεί σε 4 bytes συνεπώς θα γράψει στη θέση 6 (Σχήμα 6.7) , με την εντολή αυτή ελέγχεται και η μονάδα προώθησης μεταξυ execution και decode stage .

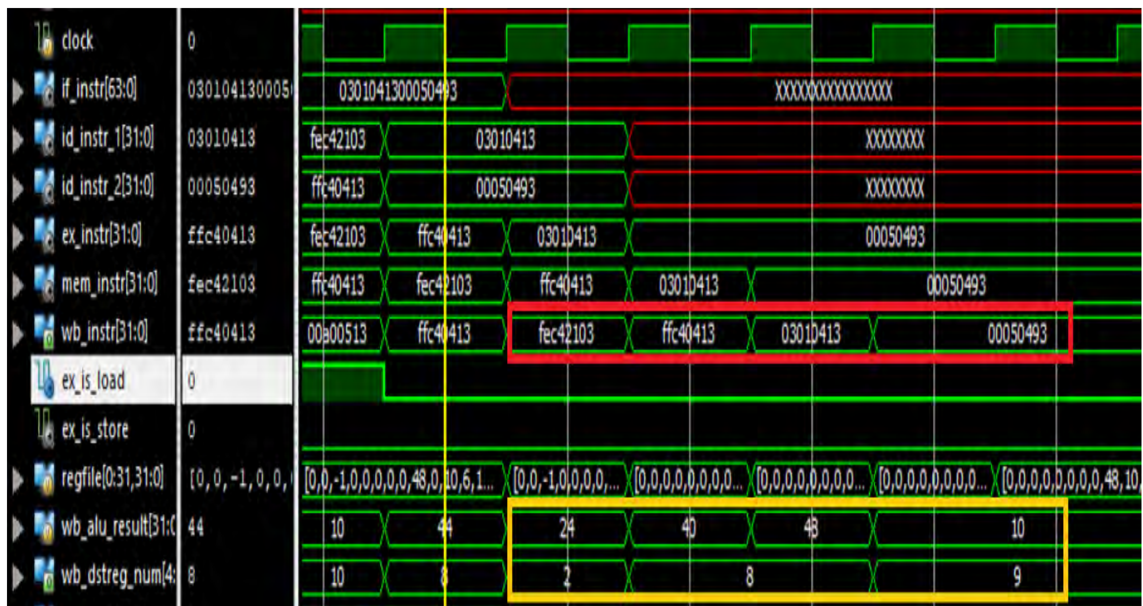
Σχήμα 6.7: Αποτελέσματα προσομοίωσης

Η εντολή addi x15 x15 1 θα πρέπει να γράψει την τιμή 1 στο καταχωρητή 15 και η εντολή sub x2 x2 x15 γράφει την τιμή -1 στον καταχωρητή 2 .



Σχήμα 6.8: Αποτελέσματα προσομοίωσης

Η addi x11 x0 6 και η addi x10 x0 5 γράφουν τις τιμές 6 και 5 στους καταχωρητές 11 και 12 αντίστοιχα και κατόπιν η εντολή add x12 x10 x11 γράφει την τιμή 11 στον καταχωρητή 12 , η σωστή εκτέλεση αυτής της εντολής επιβεβαιώνει την ορθή λειτουργία των μονάδων προώθησης μεταξύ decode και execution και decode και memory stage .Κατόπιν εκτελείται δυο φορές η εντολή addi x10 x0 10 και στην συνέχεια η εντολή addi x8 x8 -4 που κάνει την τιμή του καταχωρητή 8 να ισούται με 44 .



Σχήμα 6.9: Αποτελέσματα προσομοίωσης

Στην συνέχεια με την εκτέλεση της εντολής lw x2 -20x8 θα γραφτεί στον καταχωρητή 2 η τιμή

0 που είχε αποθηκευτεί στην μνήμη με την εντολή sw . Τέλος οι επόμενες εντολές addi εκτελούνται σύμφωνα με τα αναμενόμενα.

6.3.2 Risc-v assembly χωρίς εντολές διακλάδωσης II

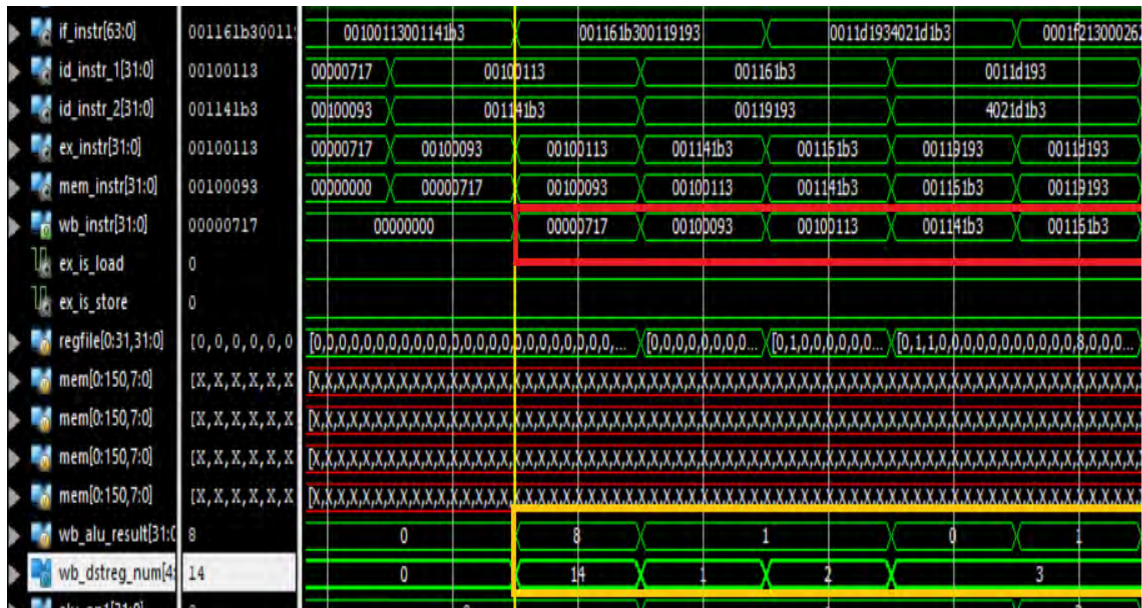
```
auipc x14 0
addi x1 x0 1
addi x2 x0 1
xor   x3 x2 x1
or x3 x2 x1
slli x3 x3 1
srli x3 x3 1
sra x3 x3 x2
andi x4 x3 0
ori x4 x4 0
sb x1 8(x1)
lb x5 8(x1)
sh x1 16(x1)
lh x6 16(x1)
```

Σχήμα 6.10: Risc-v

```
0000000000000000
0000071700100093
0010011300114 1B3
001161B300119193
0011D1934021D1B3
0001F21300026213
0010842300808283
0010982301008303
```

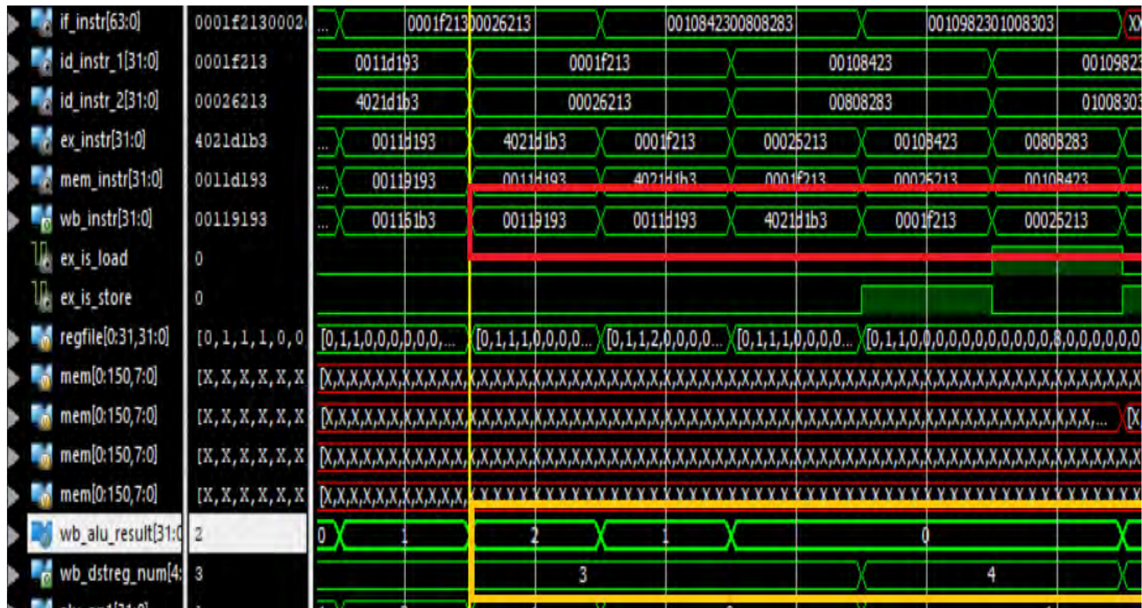
Σχήμα 6.11: Risc-v hex

Η εντολή auipc x14 0 γράφει στον καταχωτηρη 14 την τιμή του μετρητή προγράμματος και καθώς η εντολή είναι η πρώτη μετά τις δύο εντολές αρχικοποίησης την τιμή 8. Οι δύο επόμενες addi γράφουν την τιμή ένα στους καταχωρητες 1 και 2 αντίστοιχα . Η εντολή xor x3 x1 x2 θα πρέπει να γράψει 0 στον καταχωρητη 3 , και η εντολή or x3 x1 x2 θα πρέπει να γράψει 1 στον καταχωρητή 3. Με την σωστή αποτίμηση αυτών των εντολών φαίνεται ότι και η μονάδα προώθησης του write back stage λειτουργεί σωστά αλλά και οι υπόλοιπες που εχουν ελεγχεί και προηγούμενα .



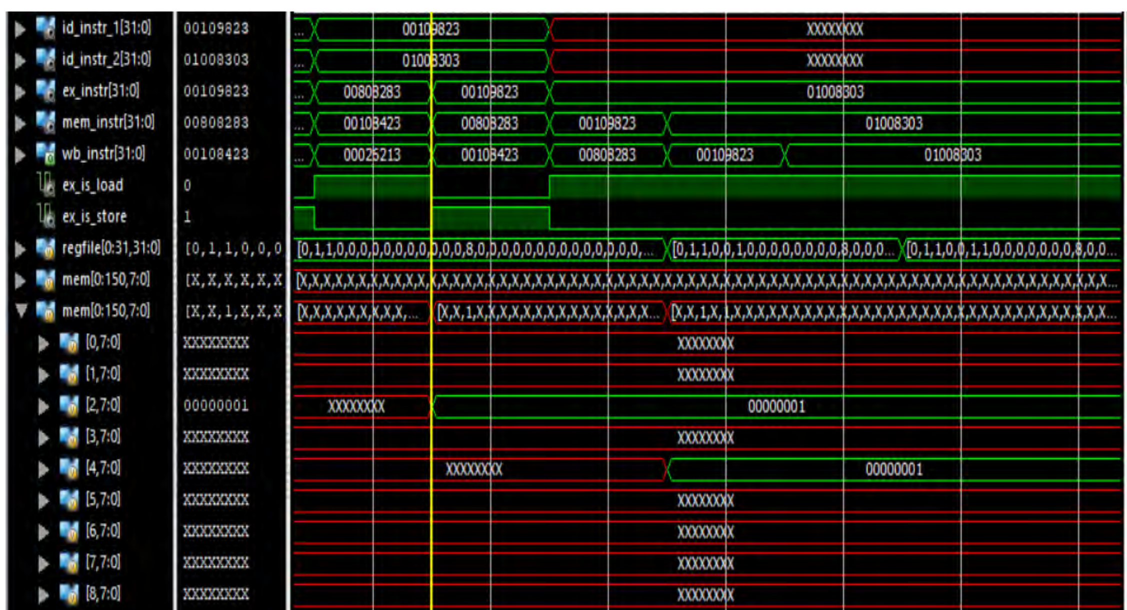
Σχήμα 6.12: Αποτελέσματα προσομοίωσης

Η εντολή slli x3 x3 1 πραγματοποιεί λογική αριστερή μετατόπιση κατά μια θέση συνεπώς στον καταχωρητή 3 η τιμή θα μεταβληθεί σε 2 . Η εντολή srli x3 x3 1 πραγματοποιεί δεξία μετατόπιση κατά 1 συνεπώς θα αναιρεθεί η πράξη της προηγούμενης εντολής . Η sra x3 x3 x2 πραγματοποιεί αριστερή μετατόπιση στον καταχωρητή 3 σύμφωνα με την τίμη του καταχωρητή 2 συνεπώς στην θέση 1 θα γραφτεί η τιμή 0.Η εντολή andi x4 x3 0 πραγματοποιεί λογικό and μεταξύ του x3 και του 0 οπότε στην θέση 4 μένει η τιμή 0. Η εντολή ori x4 x3 0 πάλι αφήνει τον καταχωρητή 4 στην τιμή 0.



Σχήμα 6.13: Αποτελέσματα προσομοίωσης

Η εντολή sb x1 8(x1) θα πρέπει να γράψει την τιμή 1 στο 10ο byte της μνήμης και η τιμή αυτη να γράφτει σαν 1 Byte , συνεπώς η τιμή 1 γράφεται μόνο στο δεύτερο στιγμιότυπο μνήμης στην θέση 2 . Η εντολή lb x5 8(x1) διαβάζει απο την ίδια θέση μνήμης και τοποθετεί το Byte που διάβασε στον καταχωρητη 5 μετά απο προσημασμένη επέκταση σε 32Bit.



Σχήμα 6.14: Αποτελέσματα προσομοίωσης

Η εντολή sh x1 16(x1) θα πρέπει να γράψει στο 18 byte την τιμή 1 και η τιμή αυτη να έχει μέγεθος 2 bytes συνεπώς θα γράψει στην θέση 4 απο το δεύτερο και τρίτο στιγμιότυπο μνήμης . Τέλος η εντολή lh x6 16(x1) τοποθετεί την τιμή αυτή στον καταχωρητή 6. Με αυτά τα παραδείγματα γίνεται αντιληπτή η οργάνωση της μνήμης για την υποστήριξη εντολών που γράφουν και διαβάζουνε ανα Byte ή ανα 2 Bytes .

Σχήμα 6.15: Αποτελέσματα προσομοίωσης

6.3.3 Risc-v assembly με εντολές διακλάδωσης

```
addi x1 x2 16
sub x2 x2 x1
sw x2 0(x1)
beq x1 x2 l1
ori x3 x1 8
slti x3 x3 2
lw x4 0(x1)
mv x3 x4
l1:
addi x5 x0 15
addi x6 x5 10
sll x6 x6 x5
bne x5 x1 l2
addi x10 x0 10
add x11 x10 x0
mv x12 x5
sb x12 8(x0)
l2:
xor x1 x2 x1
slli x2 x1 1
addi x1 x2 1
addi x3 x1 1
```

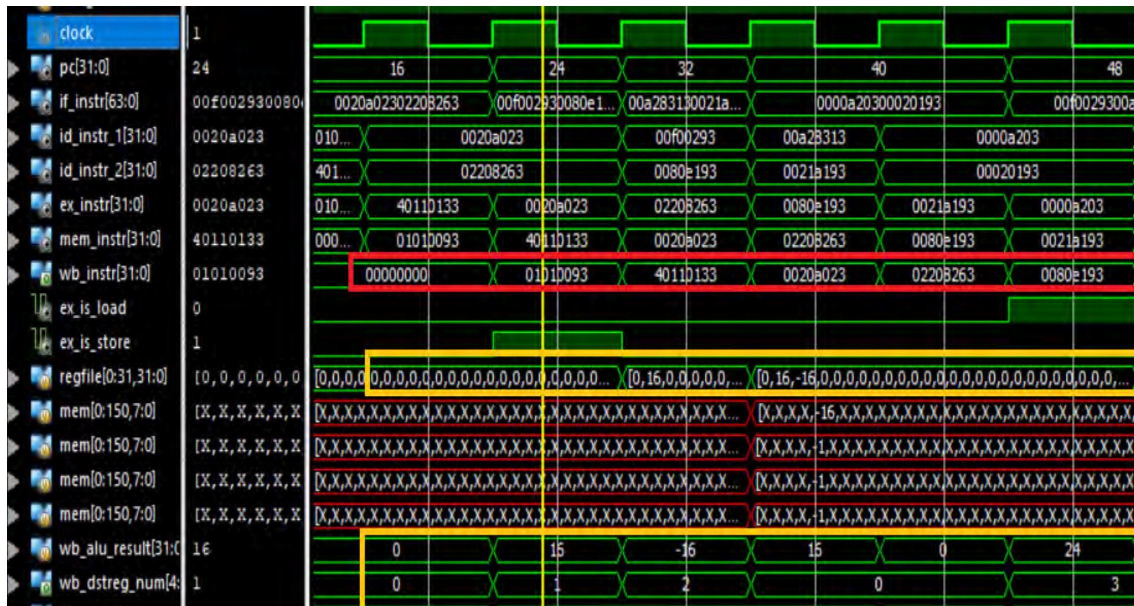
Σχήμα 6.16: Risc-v assembly κλασσικής αρχιτεκτονικής

```
addi x1 x2 16       sub x2 x2 x1
sw x2 0(x1)         beq x1 x2 l1
addi x5 x0 15       ori x3 x1 8
addi x6 x5 10       slti x3 x3 2
lw x4 0(x1)         mv x3 x4
addi x5 x0 15       addi x6 x5 10
l1:
sll x6 x6 x5        bne x5 x6 l2
xor x1 x2 x1        addi x10 x0 10
slli x2 x1 1        add x11 x10 x0
mv x12 x5 sb        x12 8(x0)
xor x1 x2 x1        slli x2 x1 1
l2:
addi x1 x2 1        addi x3 x1 1
```

Σχήμα 6.17: Risc-v assembly μετά τον μετασχηματισμό

Στο σχήμα 6.16 φαίνεται ένα τυπικό παρέδειγμα riscv assembly με δύο εντολές διακλάδωσης και στο σχήμα 6.17 απεικονίζεται ο ίδιος κώδικας μετά τις απαραίτητες αλλαγές που γίνονται σε επίπεδο μεταγλώττισης σύμφωνα με τα όσα έχουν περιγραφεί θεωρητικά παραπάνω,προκειμένου να τροφοδοτήσει την υλοποίηση .

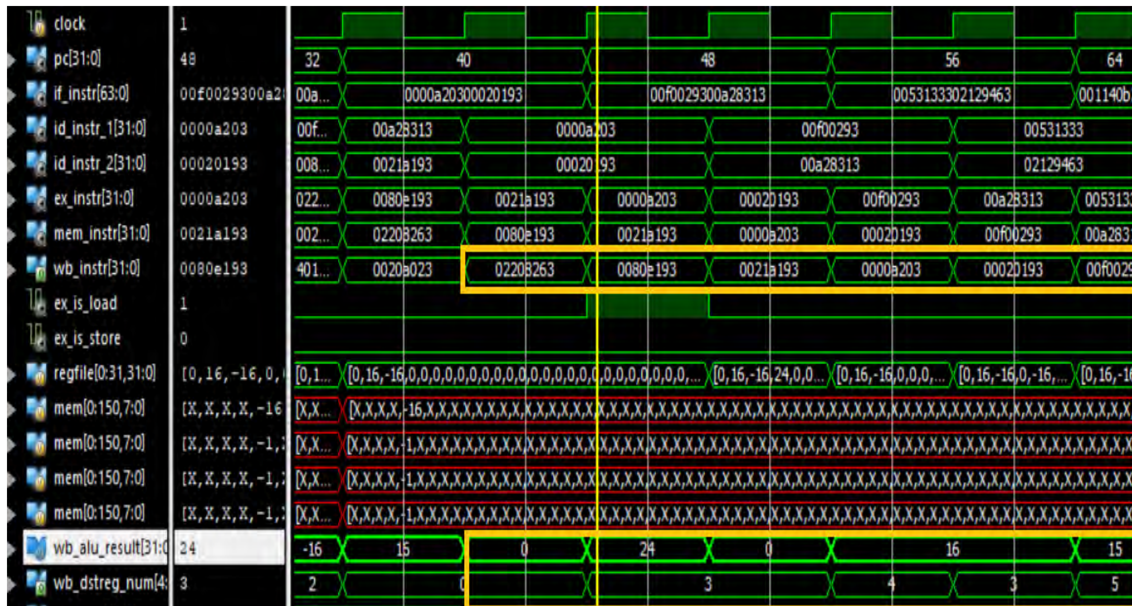
Οι εντολές addi x1 x2 16 και sub x2 x2 x1 γράφουνε στις θέσεις 1 και 2 του φακέλου καταχωρητών τις τιμές 16 και -16 αντίστοιχα . Η εντολή sw x2 0(x1) γράφει στο 16ο byte της μνήμης δηλαδή στην 5η θέση την τιμή -16 .Στην συνέχεια η εντολή διακλάδωσης αποτιμάται ψευδή συνεπώς δεν πραγματοποιείται άλμα και προωθούνται για εκτέλεση οι δεύτερες εντολές των δύο επόμενων block .



Σχήμα 6.18: Αποτελέσματα προσομοίωσης

Επομένως η ori x3 x1 8 γράφει 24 στον καταχωρητή 3 και slti x3 x3 2 γράφει 0 στον ίδιο καταχωρητή μετά την σύγκριση που κάνει. Ακολουθεί η εντολή lw x4 0(x1) που διαβάζει απο την μνήμη την τιμή -16 που τοποθετήθηκε προηγούμενα και η εντολή mv x3 x4 που τοποθετεί την

απόλυτη τιμή του καταχωρητή χ4 στο χ3.



Σχήμα 6.19: Αποτελέσματα προσομοίωσης

Κατόπιν εκτελούνται οι δύο εντολές addi x5 x0 15 και addi x6 x5 10 οι οποίες με την τροποποίηση του μεταγλωττιστή έχουνε τοποθετηθεί πριν την διεύθυνση του άλματος σε περίπτωση που γινότανε ώστε να μν εκτελεστούν εις διπλούν σε αληθή αποτίμηση της εντολής διακλάδωσης. Το επόμενο block εντολών περιλαμβάνει μια εντολή ολίσθησης την sll x6 x6 x5 και την εντολή διακλάδωσης bne x5 x6 l2 η οποία αποτιμάται αληθή και πραγματοποιεί το άλμα ,συνεπώς θα εκτελεστούν οι πρώτες εντολές απο τα δύο επόμενα block και κατόπιν θα αλλάξει η τιμή του μετρητή προγράμματος. Να σημειωθεί οτι απαραίτητη ενέργεια του μεταγλωττιστή είναι η κατάλληλη διαμόρφωση offset της εντολής διακλάδωσης ώστε να δείχνει στην σωστή διεύθυνση μετά την δημιουργία των διπλών εντολών .Η εντολή xor x1 x1 x2 γράφει την τιμή -32 στον καταχωρητή 1 Και η εντολή slli x2 x1 1 ολισθαίνει προς τα αριστερά την τιμή του x1 και την γράφει στον καταχωρητή x2 . Στην συνέχεια η εντολή διακλάδωσης πραγματοποιεί το άλμα και η τιμή του μετρητή προγράμματος δείχνει στην νέα διεύθυνση απο όπου εκτελούνται οι δύο τελευταίες εντολές του προγράμματος κάνοντας τις ανάλογες προσθέσεις.

Σχήμα 6.20: Αποτελέσματα προσομοίωσης

Τα σχήματα 6.21 και 6.22 δείχνουν τον σχηματισμό των block εντολών για κώδικα που περιλαμβάνει μια εντολή διακλάδωσης με αρνητικό Offset η οποία κάνει άλμα σε προγενέστερη διεύθυνση μνήμης . Όλοι οι κώδικες που εξετάστηκαν και επιβεβαιώθηκαν δεν αποτελούν υλοποίησεις κάποιων αλγορίθμων άλλα έχουν καθαρά σκοπό τον έλεγχο της λειτουργικότητας του επεξεργαστή και την επίδειξη των αλλαγών που απαιτούνται σε επίπεδο μεταγλώττισης . Συνεπώς σε πραγματικές εφαρμογές είναι δυνατόν τα ζεύγη εντολών να μην μπορούν να προκύψουν σε ολές τις περιπτώσεις, σε αυτές τις καταστάσεις απαιτείται η εισοδός εντολών nop , αυτό θα πραγματευτούμε στο επόμενο κεφάλαιο μεταξύ αλλών.

```
add x10 x0 x0
add x11 x8 x8
addi x12 x11 80
sub x6 x10 x11
loop:
lw x13 0(x11)
add x10 x10 x13
addi x11 x11 4
addi x13 x11 4
addi x15 x13 4
blt x11 x12 loop
add x14 x0 x16
sw x10 0(x0)
sw x11 8(x0)
sw x14 16(x0)
```

```
add x10 x0 x0        add x11 x8 x8
addi x12 x11 80      sub x6 x10 x11
lw x13 0(x11)        add x10 x10 x13
loop:
addi x11 x11 4       addi x13 x11 4
addi x15 x13 4       blt x11 x12 loop
lw x13 0(x11)        add x14 x0 x16
add x10 x10 x13      sw x10 0(x0)
sw x11 8(x0)         sw x14 16(x0)
```

Σχήμα 6.21: Risc-v assembly κλασσικής αρχιτεκτονικής και Risc-v assembly μετά τον μετασχηματισμό

Κεφάλαιο 7

Αλγόριθμοι

7.1 Κριτήριο επιλογής

Οι τεχνικές πρόβλεψης διακλαδώσεων που αναλύθηκαν παραπάνω και χρησιμοποιούνται ευρέως στις εμπορικές υλοποιήσεις έχουν ενα κοίνο χαρακτηριστικό εκμεταλεύονται πόρους του επεξεργαστή αποθηκεύοντας μεγάλους πίνακες ιστορίας και συσχετίζουν την κάθε πιθάνη ιστορία είτε τοπική είτε ολόκληρου του προγράμματος με μία πρόβλεψη ύστερα απο κατάλληλη εκπαίδευση.Συνεπώς οι τεχνικές αυτες είναι αποδοτικές εφόσον οι διακλαδώσεις ενός προγράμματος ακολουθούν καποιο μοτίβο οπως συμβαίνει σε μια διακλάδωση for loop ή σε ένα if else για την διακλάδωση του else . Όμως υπάρχουν διακλαδώσεις που δεν ακολουθούν κανέναν μοτίβο και η αποτίμηση τους δεν εξαρταται απο κανένα παράγοντα που μπορεί να προκαθοριστεί και να έχει νόημα να αποθηκευτεί κάπου. Συνεπώς σε διακλαδώσεις που ακολουθούν τυχαία αποτίμηση χωρίς να εξαρτόνται απο προηγούμενη εκτέλεση της ίδιας ή κάποιας άλλης διακλάδωσης οι τεχνικές προβλεψης διακλάδωσεων κάνουν σπατάλη πόρων του συστήματος μην προσφέροντας ουσιαστική μείωση της ποινής.Η υλοποιήση που προτείνεται στα πλαίσια αυτης της εργασίας δεν αντιμετωπίζει αυτο το πρόβλημα . Μια τετοια υλοποιήση θα αποτελούσε ενδιαφέρουσα ιδέα για έναν επεξεργαστή ειδικού σκοπού που θα εκτελεί αλγορίθμους που περιλαμβάνουν την πολλαπλή εκτέλεση εντολών διακλάδωσης τυχαίας αποτίμησης .Οι αλγοριθμοί που κάνουν ταξινόμηση είναι ενα παράδειγμα με σημαντικότερο τον Quicksort αλλα και οι αλγόριθμοι ομαδοποίησης που χρησιμοποιούνται στις machine learning εφαρμογές. .

7.2 Quicksort

Ο quicksort είναι ένας πολύ αποδοτικός αλγόριθμος ταξινόμισης ,αναπτύχθηκε απο τον Βρετανό επιστήμονα υπολογιστών Tony Hoare το 1957 .Είναι ακόμη πολύ διαδεδομένος και αποτελεί την βασική υλοποίηση της συνάρτησης sort σε βιβλιοθήκες απο πολλές γλώσσες προγραμματισμού.Όταν υλοποείται σωστά είναι περίπου δύο με τρείς φορές ταχύτερος απο τους ανταγωνιστές του merge sort και heap sort.O quicksort είναι ένας αλγόριθμος της κατηγορίας « διαίρει και βασίλευε .Η βασική ίδεα είναι η επιλογή ενός στοιχείου διαχωρισμού και η ταξινόμηση των άλλων στοιχείων δεξία ή αριστερά του στοιχείου διαχωρισμού ανάλογα ανα είναι μεγαλύτερα ή μικρό-

τερα του στοιχείου διαχωρισμού.Οι υποπίνακες που δημιουργούνται ταξινομούνται με τον ίδιο τρόπο αναδρομικά. Ο quicksort ανήκει στην κατηγορία των in-place αλγορίθμων μην απαιτώντας μεγάλη πρόσθετη μνήμη για να ολοκληρωθεί.Η μαθηματική ανάλυση του quicksort δείχνει οτι στην μέση περίπτωση ο Quicksort χρησιμοποιεί O(nlogn) συγκρίσεις για να ταξινομήσει n αντικείμενα και στην χειρότερη περίπτωση O(N^2).[13]

Algorithm 1. Sort array part $a[\ell..r]$

```
Procedure quicksort(ℓ, r : integer);
  if r > ℓ then
    i = ℓ; j = r; x = pivot();
    repeat
      while a[i] < x do i++ ; endwhile {Loop I}
      while a[j] > x do j-- ; endwhile {Loop J}
      if i ≤ j then swap(a[i], a[j]);
    until j ≤ i
    quicksort(ℓ, i − 1);
    quicksort(i + 1, r);
  end if
```

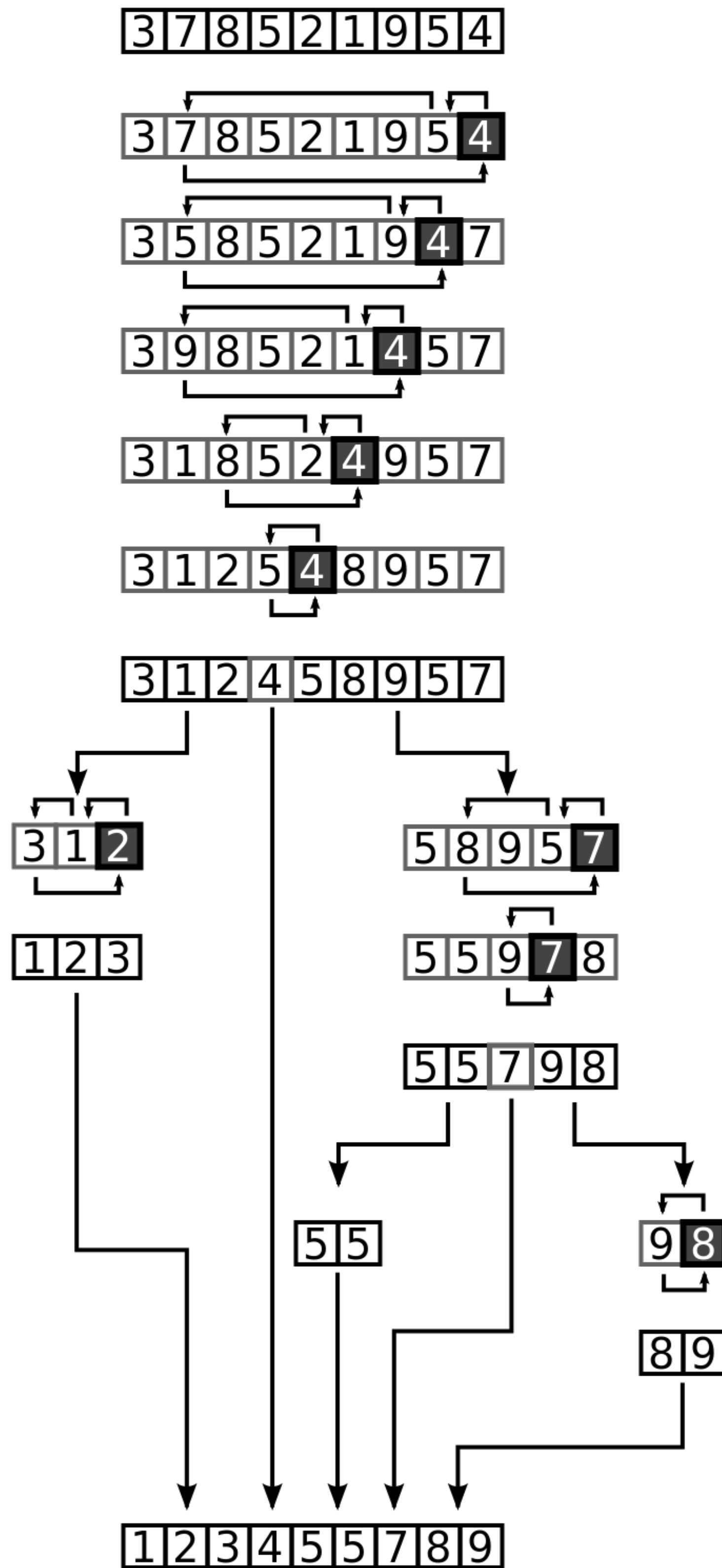
Σχήμα 7.1: Ψευδοκώδικας Quicksort

Αναλυτικά τα αλγοριθμικά βήματα περιγραφής του αλγορίθμου :

- Επιλογή ενός στοίχειου διαχωρισμού του πίνακα

- Διαμέριση . Ανταλλαγή στοιχείων του πίνακα ώστε τα στοιχεία μεγαλύτερα του στοιχείου διαχωρισμού να τοποθετηθούν δεξιά του και τα μικρότερα αριστερά του.Μετά τον διαχωρισμό το στοιχείο διαχωρισμού βρίσκεται στην τελική του θέση.

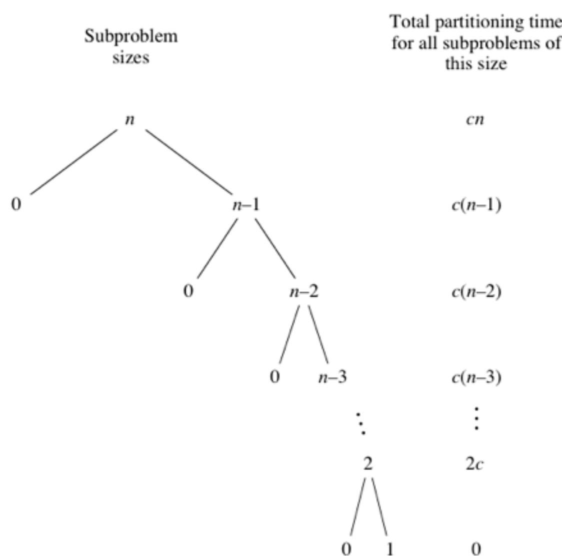
- Αναδρομική εφαρμογή τον παραπάνω βημάτων στον υποπίνακα με στοιχεία μικρότερα του στοιχείου διαχωρισμού και στον υποπίνακα με στοιχεία μεγαλύτερα του στοιχείου διαχωρισμού.



Σχήμα 7.2: Πλήρες στιγμιότυπο εκτέλεσης αλγορίθμου

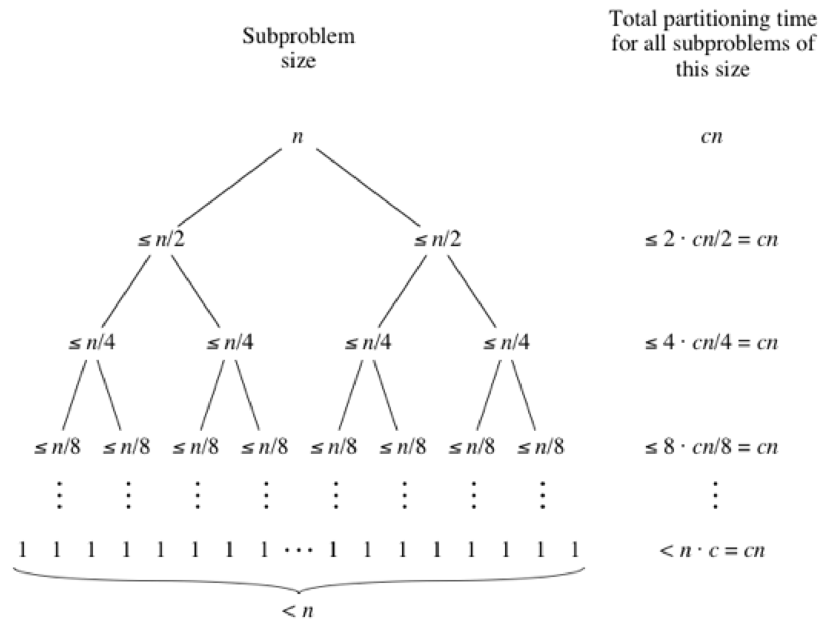
7.3 Ανάλυση πολυπλοκότητας με βάση το στοιχείο διαχωρισμού

i Αν σαν στοιχείο διαχωρισμού επιλεγεί το μικρότερο στοιχείο του πίνακα και ο πίνακας είναι ήδη ταξινομημένος στην πρώτη κλήση η Partition συγκρίνει μεταθέτει n στοιχεία και διασπά τον πίνακα σε κενό αριστερό πίνακα και έναν δεξί πίνακα μεγέθους n-1 .Όμοια στην δεύτερη κλήση θα απομείνει στα δεξιά πίνακας μεγέθους n-2 ενώ θα έχουν εκτελεστεί n-1 συγκρίσεις μεταθέσεις. Το Σχήμα 7.3 φαίνεται το δέντρο των υποπροβλημάτων απο τήν εκτέλεση στην καλύτερη περίπτωση απ' οπου προκύπτει και η πολυπλοκότητα $\Theta(n^2)$



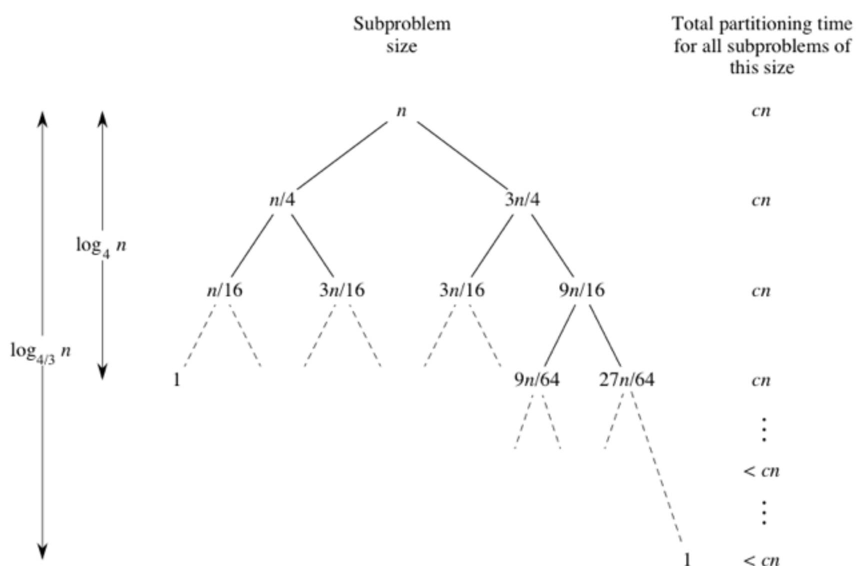
Σχήμα 7.3: Χειρότερη περίπτωση εκτέλεσης

ii Η καλύτερη περίπτωση εκτέλεσης του αλγορίθμου συμβαίνει οταν μετά την τοποθέτηση των στοιχείων διαχωρισμού οι αριστεροί και οι δεξιοί υποπίνακες που προκύπτουν σε όλες τις περιπτώσεις έχουν ίδιο μέγεθος ή διαφέρουν κατά ένα .Σε περίπτωση που ο αρχικός πίνακας εχεί μέγεθος περιττό και το στοιχείο διαχωρισμού τοποθετηθεί ακριβώς στην μέση κάθε υποπίνακας έχει μέγεθος n-1/2 , ενώ στην περίπτωση που ο αρχικός πίνακας έχει μέγεθος άρτιο τότε οι υποπίνακες που θα δημιουργηθουν θα έχουν μέγεθος n/2 και n/2 -1 . Το Σχήμα 7.4 φαίνεται το δέντρο των υποπροβλημάτων απο τήν εκτέλεση στην καλύτερη περίπτωση απ' οπου προκύπτει και η πολυπλοκότητα $\Theta(nlogn)$.



Σχήμα 7.4: Καλύτερη περίπτωση εκτέλεσης

iii Στην μέση περίπτωση το στοιχείο διαχωρισμού μπορεί να βρεθεί σε οποιαδήποτε θέση του πίνακα με την ίδια πιθανότητα ,υπάρχουν n δυνατές θέσεις κάθε μία με πιθανότητα 1/n. Συνεπώς οι δύο υποπίνακες που προκύπτουν δεν είναι ίδιου μεγέθους ,το Σχήμα 7.5 δείχνει το δέντρο των υποπροβλημάτων αν υποθέσουμε οτι το στοιχείο διαχωρισμού κάθε φορά δεξία του τοποθετεί τα 3/4 των στοιχείων και αριστερά τα του το 1/4. Η απόδειξη της πολυπλοκότητας είναι αρκετά δύσκολη και προκύπτει η πολυπλοκότητα O(*nlogn*) για την μέση περίπτωση .[1]



Σχήμα 7.5: Μέση περίπτωση εκτέλεσης

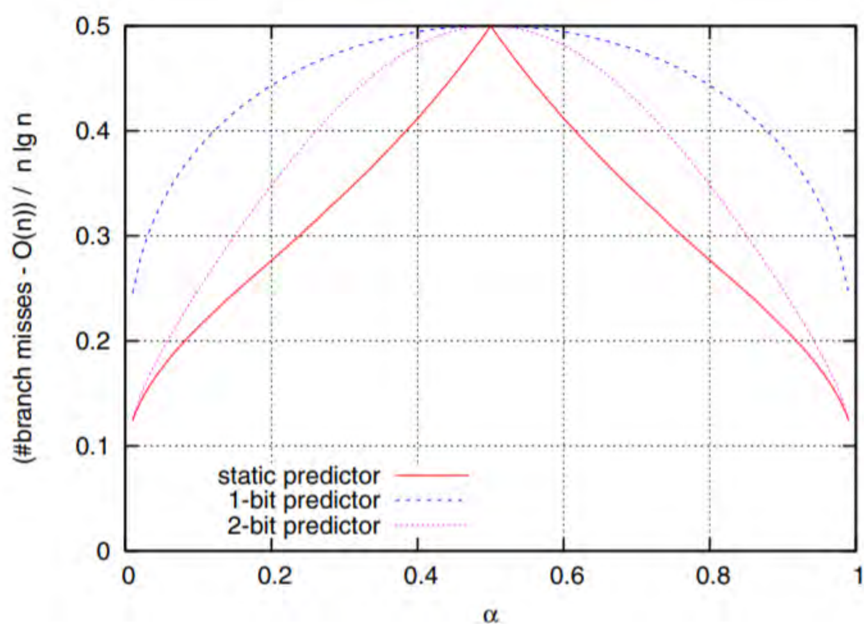
7.4 Quicksort και Branch Misprediction

Σύμφωνα με την ανάλυση πολυπλοκότητας θα περίμενε κανείς με την επιλογή ενός στοιχείου διαχωρισμού κοντά στο μέσο των στοιχείων του πίνακα να επιφέρει βελτιωμένους χρόνους εκτέλεσης καθώς θα πρέπει να εκτελεστούν λιγότερες συγκρίσεις και θα προκύψουν λιγότερες αστοχίες της μνήμης cache ωστόσο αυτό δεν συμβαίνει πάντα .Ο λόγος είναι ότι οι συγκρίσεις που εκτελούνται στον Quicksort επηρεάζουν την ταχύτητα εκτέλεσης με ένα πιο έμμεσο τρόπο . Οι εντολές διακλάδωσης της Assembly που εξαρτόνται απο της εντολές σύγκρισης της C είναι εξαιρετικά δύσκολο να προβλεπτούν.Συνεπώς στους συγχρονούς επεξεργαστές με μεγάλο μήκος παροχεύτευσης και πολλές εντολές να εκτελούνται παράλληλα μια αποτυχημένη πρόβλεψη εντολής διακλαδωσης επιφέρει αρκετούς κύκλους καθυστέρησης μετά την απομάκρυνση των λανθασμένων εντολών.Στην καλύτερη περίπτωση πολυπλοκότητας η πιθανότητα η εντολή διακλαδωσης να αποτιμηθεί αληθώς ή ψευδώς είναι 50% χωρίς να ακολουθεί κανένα επαναλαμβανόμενο μοτίβο συνεπώς σε αυτή την περίπτωση ακόμη και ο πιο έξυπνος branch predictor είναι αχρείαστος και κάνει σπατάλη πόρων, αυτός είναι και ο λόγος αύξησης του χρόνου εκτέλεσης . Οι K. Kaligosi και P.Sanders απέδείξαν τον αριθμό των Branch Misses του Quicksort για κάθε στοιχείου διαχωρισμού για τις τρεις πιο διαδεδομένες μεθόδους πρόβλεψης διακλαδώσεων static branch predictor , 1 bit predictor και 2 bit predictor.[16],[21],[10]

	random pivot	α-skewed pivot
static predictor	$\frac{\ln 2}{2} n \lg n + \mathcal{O}(n)$, $\frac{\ln 2}{2} \approx 0.3466$	$\frac{\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, $\alpha < 1/2$ $\frac{1-\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, $\alpha \geq 1/2$
1-bit predictor	$\frac{2\ln 2}{3} n \lg n + \mathcal{O}(n)$, $\frac{2\ln 2}{3} \approx 0.4621$	$\frac{2\alpha(1-\alpha)}{H(\alpha)} n \lg n + \mathcal{O}(n)$
2-bit predictor	$\frac{28\ln 2}{45} n \lg n + \mathcal{O}(n)$, $\frac{28\ln 2}{45} \approx 0.4313$	$\frac{2\alpha^4-4\alpha^3+\alpha^2+\alpha}{(1-\alpha(1-\alpha))H(\alpha)} n \lg n + \mathcal{O}(n)$

Σχήμα 7.6: Branch Misses με βάση το στοιχείο διαχωρισμού

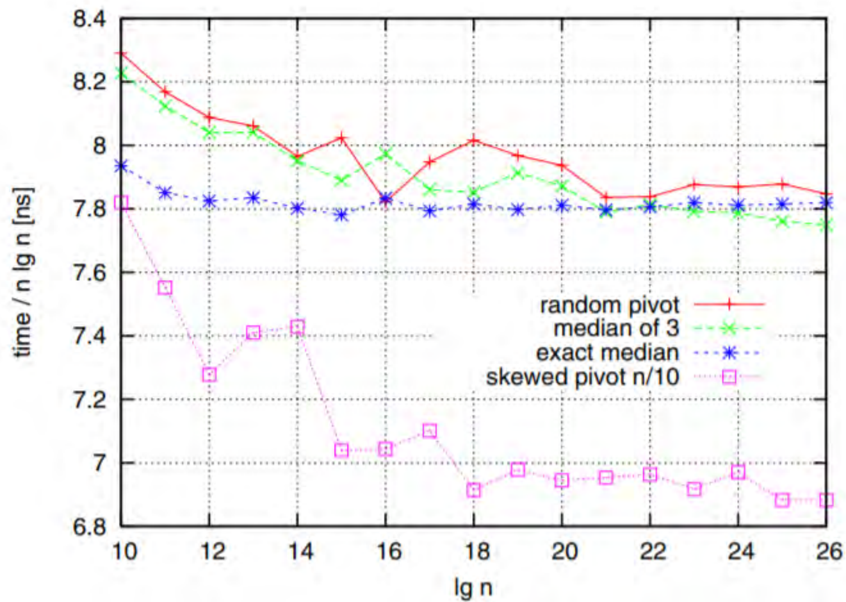
Ο Static branch predictor φαίνεται πιο αποδοτικός με λιγότερα misses και στην περίπτωση της τυχαίας επιλογής στοιχείου διαχωρισμού και στην περίπτωση επιλογής α-στοιχείου διαχωρισμού.



Σχήμα 7.7: Καμπύλή Branch Misses με βάση το στοιχείο διαχωρισμού

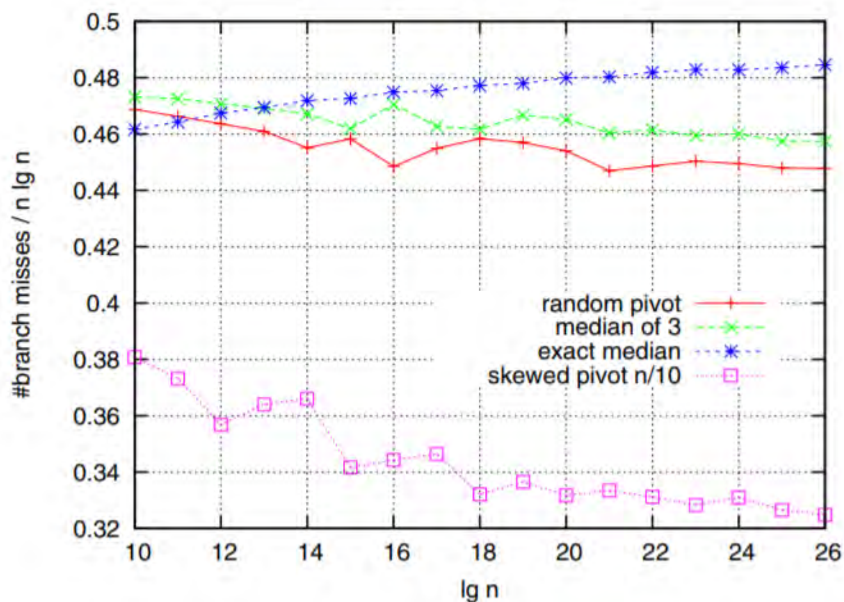
Όπως ήταν αναμενόμενο στο Σχήμα 7.7 φαίνεται ότι για στοιχείο διαχωρισμού το μέσον του πίνακα έχουμε τις περισσότερες λάθος προβλέψεις.

Στα Σχήματα 7.8,7.9,7.10 που προέκυψαν πειραματικά φαίνεται η σύγκριση για επιλογή τυχαίου στοιχείου διαχωρισμού, για το μέσο τρίων τυχαίων στοιχείων ,για το ακριβες μέσο δηλαδή το 1/2-skewed και για το 1/10-skewed σε σχέση με τον χρόνο εκτέλεσης , τον αριθμό των Branch mispredictions και των αριθμό των εκτελεσθέντων εντολών, για διαφορετικες τιμές του n . To n καθορίζει το άνω όριο των στοιχείων του πίνακα για ταξινόμηση απο την σχέση max $[100,10^7/n]$.



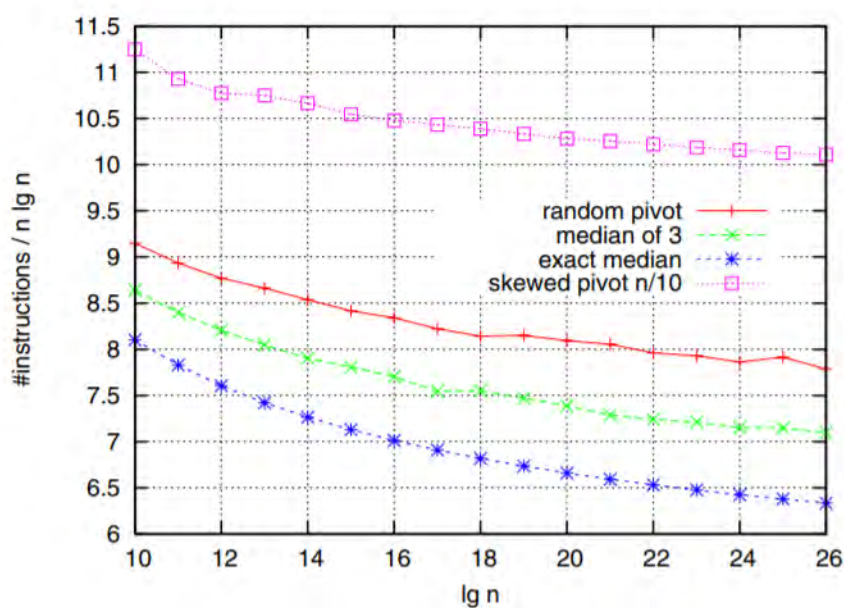
Σχήμα 7.8: Χρόνος εκτέλεσης σε σχέση με το στοιχείο διαχωρισμού για διάφορες τιμές του n

Στο Σχήμα 7.8 φαίνεται ότι η επιλογή τυχαίου στοιχείο διαχωρισμού σε σχέση με το ακριβές μέσον και με το μέσον των τριών οδηγεί σε ελαφρώς χειρότερα αποτελέσματα με τις καμπύλες όμως να βρίσκονται αρκέτα κοντά και να μην διακαιλογείται σε καμία περίπτωση ότι η επιλογή στοιχείου διαχωρισμού κοντά στο μέσον επιφέρει αύξηση της ταχύτητας σε σχέση με την τυχαία επιλογή. Αντίθετα η επιλογή για στοιχείο διαχωρισμού το 1/10-skewed επιφέρει σημαντική μείωση του χρόνου εκτέλεσης .



Σχήμα 7.9: Branch Misses σε σχέση με το στοιχείο διαχωρισμού για διάφορες τιμές του n

Στο Σχημα 7.9 βλέπουμε την τυχαία επιλογή σημείου διαχωρισμού να έχει λιγότερα branch misses, για τις περισσότερες τιμές του n ,σε σχέση με το ακριβές μέσον και με το μέσον των τριών. Η επιλογή του 1/10-skewed έχει σαφώς καλύτερα αποτελέσματα.



Σχήμα 7.10: Εντολές που εκτελούνται σε σχέση με το στοιχείο διαχωρισμού για διάφορες τιμές του n

Στο σχήμα 7.10 φαίνεται ο αριθμός των εντολών που εκτελούνται σε σχέση με την επιλογή του σημείου διαχωρισμού. Ο αριθμός των εντολών είναι ανάλογος του αριθμού των συγκρίσεων συνεπώς ο η επιλογή του μέσου για στοιχείο διαχωρισμού πετυχαίνει τις λιγότερες εντολές.

7.5 Υλοποίηση του Quicksort για αρχιτεκτονική διπλής έκδοσης

```c
/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
    array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];    // pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
 arr[] --> Array to be sorted,
  low  --> Starting index,
  high  --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Σχήμα 7.11: Υλοποίηση του Quicksort σε γλώσσα C

```
# quicksort(*arr, lo, hi)
# arr -> a1
# lo -> a2
# hi -> a3
quicksort:
blt a3, a2, quicksort_exit        # if (lo >= hi) we just return
# save stuff in the stack
addi sp, sp, -32
sw ra, 0(sp)
sw s10, 8(sp)                     # s10 is going to hold lo
sw s11, 16(sp)                              # s11 is going to hold hi
sw s9, 24(sp)                     # s9 is going to hold the pivot
# hold lo and hi
mv s10, a2 # s10 <- lo
mv s11, a3 # s11 <- hi
# call partition
jal ra, partition
# save the pivot on s9
mv s9, a0
# s9 = pivot
# s10 = lo
# s11 = hi
# recursively call quicksort on both subarrays
addi a3, s9, -1          # hi = pivot (-1)
mv a2, s10               # lo = lo
jal ra, quicksort        # quicksort(a1, lo, pivot-1)
addi a2, s9, 1           # lo = pivot (+1)
mv a3, s11               # hi = hi
jal ra, quicksort
# load stuff back from the stack
lw ra, 0(sp)
lw s10, 8(sp)
lw s11, 16(sp)
lw s9, 24(sp)
addi sp, sp, 32
quicksort_exit:
ret
# partition(*arr, lo, hi) -> pivot
# arr -> a1
# lo -> a2
# hi -> a3
# pivot <- a0 (return value)
```

Σχήμα 7.12: Υλοποίηση του Quicksort σε Assembly Riscv I

```
partition:
# save stuff in the stack
addi sp, sp, -24
sw ra, 0(sp)
sw s10, 8(sp)
sw s11, 16(sp)
# init pivot to high (a3)
add t0, a1, a3
lbu t0, 0(t0)
addi t2, a2, -1                  # (i) index of the smaller element => t2 = low - 1
mv t6, a2                          # t6 = j = low
addi t5, a3, -1                   # t5 = high-1
partition_forloop:
bgt t6, t5, partition_forloop_end       # if t6 > t5 then partition_forloop_end
add s11, a1, t6                   # s11 = *arr[j]
lbu t1, 0(s11)                    # t1 = *(arr[j])
bgtu t1, t0, partition_forloop_inner_skip # if t1>t0 skip (if arr[j]>pivot)
addi t2, t2, 1                    # i++
add s10, a1, t2                   # s10 = *arr[t2] = *arr[i]
lbu t3, 0(s10)                    # t3 = *(arr[i])
sw t3, 0(s11)                     # arr[j] = t3
sw t1, 0(s10)                     # arr[i] = t1
partition_forloop_inner_skip:
addi t6, t6, 1                    # j++
j partition_forloop
partition_forloop_end:
addi a0, t2, 1                    # write return value as i+1
# swap(&arr[i+1], &arr[high])
add s10, a1, a0                   # s10 = *arr[i+1]
add s11, a1, a3                   # s11 = *arr[high]
lbu t2, 0(s10)                    # t2 = *s10
lbu t3, 0(s11)                    # t3 = *s11
sw t2, 0(s11)
sw t3, 0(s10)
# load stuff back from the stack
lw ra, 0(sp)
lw s10, 8(sp)
lw s11, 16(sp)
addi sp, sp, 24
partition_bail:
ret
```

Σχήμα 7.13: Υλοποίηση του Quicksort σε Assembly Riscv I

```
quicksort:
nop                     blt a3, a2, quicksort_exit
ret                     addi sp, sp, -32
nop                     sw ra, 0(sp)
sw s10, 8(sp)           sw s11, 16(sp)
sw s9, 24(sp)           mv s10, a2
mv s11, a3              jal ra, partition
mv s9, a0               addi a3, s9, -1
mv a2, s10              jal ra, quicksort
addi a2, s9, 1          mv a3, s11
nop                     jal ra, quicksort
lw ra, 0(sp)            lw s10, 8(sp)
lw s11, 16(sp)          lw s9, 24(sp)
addi sp, sp, 32         ret
quicksort_exit:
partition:
addi sp, sp, -24        sw ra, 0(sp)
sw s10, 8(sp)           sw s11, 16(sp)
add t0, a1, a3          lbu t0, 0(t0)
addi t2, a2, -1         mv t6, a2
addi t5, a3, -1         nop
partition_forloop:
nop                     bgt t6, t5, partition_forloop_end
addi a0, t2, 1          add s11, a1, t6
add s10, a1, a0         lbu t1, 0(s11)
nop                     bgtu t1, t0, partition_forloop_inner_skip
addi t6, t6, 1          addi t2, t2, 1
j partition_forloop     add s10, a1, t2
lbu t3, 0(s10)          sw t3, 0(s11)
sw t1, 0(s10)           addi t6, t6, 1
j partition_forloop     nop
partition_forloop_inner_skip:
addi a0, t2, 1          add s10, a1, a0
partition_forloop_end:
add s11, a1, a3         lbu t2, 0(s10)
lbu t3, 0(s11)          sw t2, 0(s11)
sw t3, 0(s10)           lw ra, 0(sp)
lw s10, 8(sp)           lw s11, 16(sp)
addi sp, sp, 24         nop
partition_bail:
ret                     nop
```

Σχήμα 7.14: Υλοποίηση του Quicksort για αρχιτεκτονική διπλής έκδοσης

Στα Σχήματα 7.11 παρουσιάζεται μια βασική υλοποίηση του Quicksort σε γλώσσα C ,στο Σχήμα 7.12 και 7.13 η Assembly Riscv που προκύπτει και στο Σχήμα 7.14 η Assembly που θα

δημιουργούσε ο ειδικός μεταγλωττιστής για να τροφοδοτήσει την υλοποίηση μας. Μετά από κάθε εντολή διακλάδωσης υπό συνθήκη δημιουργεί τις δύο επόμενες λέξεις εντολών σύμφωνα με τα όσα έχουνε αναφερθεί ,επίσης αλλάζει τους στόχους των εντολών διακλάδωσης ώστε να μην υπάρχουνε διπλές εκτέλεσεις.
Παρατηρήσεις :

- Η πρώτη εντολή διακλάδωσεις blt a3,a2,quicksort_exit παρότι δεν μας ενδιαφέρει αρκετά καθώς θα εκτελεστεί μόνο μία φορά δεν έγινε πλήρης πλήρωση των δύο λέξεων που την ακολούθουν καθώς στην δεύτερη λέξη έπρεπε να τοποθετηθεί εντολή Nop .

- Η δεύτερη εντολή διακλάδωσης bgt t6 t5 partition for loop end θα εκτελεστεί όσες φόρες κληθεί αναδρομικά η Partition δηλαδή όσες διαμερίσεις γίνουνται στον αρχικό πίνακα όπως είναι λογικό η εντολή αύτη αποτιμάται αληθώς μόνο μια φορά στην τελευταία κλήση ,συνεπώς είναι μια διακλάδωση εύκολα προβλέψιμη από μια στατική μέθοδο πρόβλεψης και δεν μας ενδιαφέρει ιδιαίτερα παρόλο αυτά πετυχαίνουμε πλήρη πλήρωση των δύο λέξεων που την ακολουθούν .

- Η διακλάδωση bgtu t1 t0 partition for loop innet skip είναι η διακλάδωση σύγκρισης των στοιχείων του πίνανα με το στοιχείο διαχωρισμού που εκτελείται επανειλημμένα και είναι απίθανο να προβλεπτεί , αποτελεί την διακλάδωση που καθορίζει το χρόνο εκτέλεσης του αλγορίθμου . Σε αυτή την περίπτωση έγινε πληρώση και των δύο λέξεων συνεπώς η ποινή που προέκυπτε από κάθε λάθος πρόβλεψη εξαλείφθηκε .

Η μόνη περίπτωση αξιόλογης ποινής που παραμένει πλέον στην εκτέλεση του αλγορίθμου είναι η εντολή J partition for loop η οποία για να επιλυθεί με επιτυχία χρειάζεται η τοποθέτηση ενός BTB στην υλοποίήση μας που θα λειτουργεί συνδιαστικά με την διπλή έκδοση . Θα μπορούσε να αποτελέσει αντικείμενο μελλοντικής επέκτασης .

.

7.6 K-nearest neighbor

O k-nearest neighbor είναι ένας αλγόριθμος ομαδοποίησης (classification) που ανήκει στις supervised μεθόδους μηχανικής μάθησης. Στοχέυει στην ομαδοποίηση των δεδομένων που δίδονται σαν είσοδοι σε γνωστά σύνολα που επίσης δίνονται και χρησιμοποίουνται σαν δεδομένα εκπαίδευσης. Ο αλγόριθμος τοποθετεί ένα στοιχείο εισόδου σε μία απο τις προκαθορισμένες κλάσεις με βάση την απόσταση του απο k- στοιχεία των κλάσεων. Ο 1-κοντινότερος γείτονας τοποθετεί το στοιχείο εισόδου στην κλάση που περιλαμβάνει το στοιχείο με την κοντινότερη απόσταση συνεπώς ο k-κοντινότερος γείτονας τοποθετεί το στοιχείο εισόδου στην κλάση που έχει τους περισσότερος κοντινούς γειτόνους . Τα δεδομένα εκπαίδευσης είναι σημεία στο πολυδιάστατο χώρο που περιλαμβάνουν και μία ετικέτα που υποδηλώνει την κλάση στην οποία ανήκουνε.Η φάση εκπαίδευσης του αλγορίθμου αποτελείται μόνο απο την αποθήκευση αυτών των δεδομένων με την

αντίστοιχη ετικέτα.Η πιο συχνή μέθοδος μέτρησης της απόστασης για συνεχόμενες μεταβλητές είναι η ευκλείδια απόσταση ενώ για διακριτές μεταβλητές μπορεί να χρησιμοποιηθεί και η μέθοδος Hamming. [17],[24]

- Η ευκλείδια απόσταση δύο στοιχείων του ν-διαστατου χώρου p = (p1, p2,..., pn) και q = (q1, q2,..., qn) δίνεται απο των τύπο του Σχήματος 7.15.

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Σχήμα 7.15: Ευκλείδια απόσταση

- Η απόσταση Hamming μεταξύ δύο διακριτών στοιχείων είναι το πλήθος των αλλαγών που απαιτούνται προκειμένου το ένα στοιχείο να ταυτιστεί με το άλλο.

.

7.7 K-nearest neighbor και branch misprediction

Για να συσχετίσουμε την εκτέλεση του αλγορίθμου με πιθανό πρόβλημα που αντιμετωπίζει με τις εντολές διακλάδωσεις αρκεί να δουμε την υλοποίηση του σε μια γλώσσα υψηλού επιπέδου.

```
// C++ program to find groups of unknown
// Points using K nearest neighbour algorithm.
#include <bits/stdc++.h>
using namespace std;

struct Point
{
        int val;          // Group of point
        double x, y;      // Co-ordinate of point
        double distance; // Distance from test point
};

// Used to sort an array of points by increasing
// order of distance
bool comparison(Point a, Point b)
{
        return (a.distance < b.distance);
}

// This function finds classification of point p using
```

```
// k nearest neighbour algorithm. It assumes only two
// groups and returns 0 if p belongs to group 0, else
// 1 (belongs to group 1).
int classifyAPoint(Point arr[], int n, int k, Point p)
{
        // Fill distances of all points from p
        for (int i = 0; i < n; i++)
                arr[i].distance =
                        sqrt((arr[i].x - p.x) * (arr[i].x - p.x) +
                                (arr[i].y - p.y) * (arr[i].y - p.y));

        // Sort the Points by distance from p
        sort(arr, arr+n, comparison);

        // Now consider the first k elements and only
        // two groups
        int freq1 = 0;    // Frequency of group 0
        int freq2 = 0;    // Frequency of group 1
        for (int i = 0; i < k; i++)
        {
                if (arr[i].val == 0)
                        freq1++;
                else if (arr[i].val == 1)
                        freq2++;
        }

        return (freq1 > freq2 ? 0 : 1);
}

// Driver code
int main()
{
        int n = 17; // Number of data points
        Point arr[n];

        arr[0].x = 1;
        arr[0].y = 12;
        arr[0].val = 0;

        arr[1].x = 2;
        arr[1].y = 5;
```

```
arr[1].val = 0;

arr[2].x = 5;
arr[2].y = 3;
arr[2].val = 1;

arr[3].x = 3;
arr[3].y = 2;
arr[3].val = 1;

arr[4].x = 3;
arr[4].y = 6;
arr[4].val = 0;

arr[5].x = 1.5;
arr[5].y = 9;
arr[5].val = 1;

arr[6].x = 7;
arr[6].y = 2;
arr[6].val = 1;

arr[7].x = 6;
arr[7].y = 1;
arr[7].val = 1;

arr[8].x = 3.8;
arr[8].y = 3;
arr[8].val = 1;

arr[9].x = 3;
arr[9].y = 10;
arr[9].val = 0;

arr[10].x = 5.6;
arr[10].y = 4;
arr[10].val = 1;

arr[11].x = 4;
arr[11].y = 2;
arr[11].val = 1;
```

```
    arr[12].x = 3.5;
    arr[12].y = 8;
    arr[12].val = 0;

    arr[13].x = 2;
    arr[13].y = 11;
    arr[13].val = 0;

    arr[14].x = 2;
    arr[14].y = 5;
    arr[14].val = 1;

    arr[15].x = 2;
    arr[15].y = 9;
    arr[15].val = 0;

    arr[16].x = 1;
    arr[16].y = 7;
    arr[16].val = 0;

    /*Testing Point*/
    Point p;
    p.x = 2.5;
    p.y = 7;

    // Parameter to decide group of the testing point
    int k = 3;
    printf ("The value classified to unknown point"
                " is %d.\n", classifyAPoint(arr, n, k, p));
    return 0;
}
```

Η υλοποίηση σε C++ του αλγορίθμου περιλαμβάνει αρχικά τα 17 στοιχεία εκπαίδευσης του 2-διάστατου χώρου μεταβλητές τυπού Point με πεδία χ,y,val προκαθορισμένες και distance. Το στοιχείο εισοδου σαν μεταβλητή τύπου Point με προκαθορισμένες τιμές χ,y.

- Αρχικά υπολογίζεται η ευκλέιδια απόσταση του στοιχείου εισόδου απο όλα τα στοιχεία εκπαίδευσης.

- Κατόπιν ταξινομουνται τα στοιχεία εκπαίδευσης σε αύξουσα σειρά απόστασης από το στοιχείο εισόδου .

- Τέλος το στοιχείο εισόδου λαμβάνει την σωστή ετικέτα κλάσης ανάλογα με την κλάσ η που που ανήκουν οι περισσότεροι κόντινοι γείτονες .

Το γεγόνος οτι ο αλγόριθμος χρησιμοποιεί συνάρτηση για ταξινόμηση υποδεικνύει ότι σίγουρα αντιμετωπίζει το πρόβλημα που περιγραφηκε παραπάνω με τις εντολές διακλάδωσης του Quick-sort καθώς η C++ τον χρησιμοποιεί σαν προκαθορισμένη μέθοδο ταξινόμησης,χωρίς αυτό να σημαίνει ότι οι υπόλοιπες μέθοδοι ταξινόμησης δεν αντιμέτωπίζουν ανάλογα προβλήματα . Επιπλέον αλλές εντολές διακλάδωσης που δεν μπορόυν να προβλεπτούν είναι οι διακλαδώσεις που υπολογίζουνε πόσοι κοντινοί γείτονες υπάρχουν από κάθε κλάση. Αυτές οι διακλαδώσεις εκτελούνται κ-φορες για κ κοντινότερους γείτονες. Τέλος η διακλάδωση που επιστρέφει την κλάση που ανήκει η μεταβλητή εισόδου που εκτελείται μια φορά.

Κεφάλαιο 8

Επίλογος

8.1 Σύνοψη και συμπεράσματα

Η υλοποίηση που προτάθηκε και υλοποίηθηκε αντιμετωπίζει στην ιδάνικη περίπτωση της εντολές διακάδωσης με μηδέν κύκλους καθυστέρησης, δημιουργώντας έναν πολυπλοκότερο μεταγλωττιστή αλλά ένα σχετικά απλό υλικό που απαιτεί σιγουρα λιγότερο κόστος σε επίπεδο κατασκευής άλλα και κατανάλωσης ενέργειας. Αντιμέτωπίζει με την ίδια επιτυχία εντολές που ακολουθούν συγκεκριμένα μοτίβα αλλά και εντολές τυχαίας αποτίμησης . Απαιτείται η εισάγωγή εντολών Nop μόνο στις περιπτώσεις διαδοχικών εντολώ διακλάδωσης ή κώδικα με πολύ μεγάλη πυκνότητα απο τέτοιες εντολές .

8.2 Μελλοντικές επεκτάσεις

Σαν μελλοντικές επεκτάσεις η υλοποίηση της αρχιτεκτονικής σε μια πλατφόρμα επαναπρογραμματιζόμενης λογικής(FPGA) ώστε να προκύψουν αληθινά συμπεράσματα πολυπλοκότητας και κατανάλωσης ενέργειας . Επίσης η περαιτέρω αρχιτεκτονική υποστήριξη για διαδοχικές εντολές διακλάδωσης. Επίσης σε μια πολυπλόκοτερη αρχιτεκτονική με περισσότερες εξαρτήσεις δεδομένων θα μπορούσε να μελετηθεί η και η εκτέλεση εκτός σειράς απο τις υπάρχουσες εντολές που βρίσκονται στο στάδιο της αποκωδικοποιήσης .

Παράρτημα I

Ο κώδικας της εργασίας βρίσκεται: `https://github.com/kmoschos95/Riscv-core-.git`

Βιβλιογραφία

[1] Complexity Analysis of Quicksort. https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort.

[2] Computer architecture. https://en.wikipedia.org/wiki/Computer_architecture.

[3] HASE Predication Project. http://www.icsa.inf.ed.ac.uk/cgi-bin/hase/dlx-pred.pl?menu.html,pred.html.

[4] Out of order execution. https://en.wikipedia.org/wiki/Out-of-order_execution#:~:text=In%20computer%20engineering%2C%20out%2Dof,that%20would%20otherwise%20be%20wasted.

[5] Out of order processor. https://www.sciencedirect.com/topics/computer-science/out-of-order-processor.

[6] Risc vs Cisc. https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/,.

[7] What is RISC and CISC Architecture with Advantages and Disadvantages. https://www.watelectronics.com/what-is-risc-and-cisc-architecture/.

[8] David I August, Daniel A Connors, Scott A Mahlke, John W Sias, Kevin M Crozier, Ben Chung Cheng, Patrick R Eaton, Qudus B Olaniran και W MW Hwu. Integrated predicated and speculative execution in the IMPACT EPIC architecture. Στο *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No. 98CB36235)*, σελίδες 227–237. IEEE, 1998.

[9] Dileep Bhandarkar και Douglas Clark. Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization. *ACM SIGPLAN Notices*, 26:310–319, 1991.

[10] Paul Biggar, Nicholas Nash, Kevin Williams και David Gregg. An experimental study of sorting and branch prediction. *Journal of Experimental Algorithmics (JEA)*, 12:1–39, 2008.

[11] D. K. Dennis, A. Priyam, S. S. Virk, S. Agrawal, T. Sharma, A. Mondal και K. C. Ray. Single cycle RISC-V micro architecture processor and its FPGA prototype. Στο *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, σελίδες 1–5, 2017.

[12] John L. Hennessy και David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5thή έκδοση, 2011.

[13] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[14] C. Y. Ho και A. Fong. Combining Local and Global History Hashing in Perceptron Branch Prediction. *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*, σελίδες 54–59, 2007.

[15] Daniel A Jiménez, Stephen W Keckler και Calvin Lin. The impact of delay on the design of branch predictors. Στο *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, σελίδες 67–76, 2000.

[16] Kanela Kaligosi και Peter Sanders. How branch mispredictions affect quicksort. Στο *European Symposium on Algorithms*, σελίδες 780–791. Springer, 2006.

[17] James M Keller, Michael R Gray και James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[18] Richard E Kessler. The alpha 21264 microprocessor. *IEEE micro*, 19(2):24–36, 1999.

[19] Dilip Kumar και Kirat Singh. Design of High performance MIPS-32 Pipeline Processor. 2012.

[20] Johnny KF Lee και Alan Jay Smith. Branch prediction strategies and branch target buffer design. *Computer*, (1):6–22, 1984.

[21] Conrado Martínez, Markus E Nebel και Sebastian Wild. Analysis of branch misses in quicksort. Στο *2015 Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, σελίδες 114–128. SIAM, 2014.

[22] Sparsh Mittal. A survey of value prediction techniques for leveraging value locality. *Concurr. Comput. Pract. Exp.*, 29, 2017.

[23] David A. Patterson και John L. Hennessy. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5thή έκδοση, 2013.

[24] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[25] Kevin Skadron, Pritpal S Ahuja, Margaret Martonosi και Douglas W Clark. Improving prediction for procedure returns with return-address-stack repair mechanisms. Στο *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*, σελίδες 259–271. IEEE, 1998.

[26] James E Smith. A study of branch prediction strategies. Στο *25 years of the international symposia on Computer architecture (selected papers)*, σελίδες 202–215, 1998.

[27] Andrew Waterman, Yunsup Lee, David A Patterson και Krste Asanović. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1. 2016.

[28] Andrew Shell Waterman. *Design of the RISC-V instruction set architecture*. Διδακτορική Διατριβή, UC Berkeley, 2016.

[29] Tse Yu Yeh και Yale N Patt. Two-level adaptive training branch prediction. Στο *Proceedings of the 24th annual international symposium on Microarchitecture*, σελίδες 51–61, 1991.

[30] Tse Yu Yeh και Yale N Patt. Alternative implementations of two-level adaptive branch prediction. *ACM SIGARCH Computer Architecture News*, 20(2):124–134, 1992.

[31] Tse Yu Yeh και Yale N Patt. A comparison of dynamic branch predictors that use two levels of branch history. Στο *Proceedings of the 20th annual international symposium on computer architecture*, σελίδες 257–266, 1993.