ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ Η/Υ

"Προσομοίωση κατανάλωσης ενέργειας ενός επεξεργαστή πολλαπλών μονάδων εκτέλεσης με βάση τα εργαλεία Simplescalar και Wattch"

"Power consumption simulation of a multiple functional unit processor based on the tools Simplescalar and Wattch"

Διπλωματική Εργασία

Καλαϊτζίδης Π. Κλεόβουλος

Επιβλέποντες:

Γεώργιος Δημητρίου
Επιστημονικός Υπότροφος,
Πανεπιστήμιο Θεσσαλίας

Γεώργιος Σταμούλης
Καθηγητής,
Πανεπιστήμιο Θεσσαλίας

Βόλος, Οκτώβριος 2014

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Μηχανικού Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ του Πανεπιστημίου Θεσσαλίας.

……..
Καλαϊτζίδης Κλεόβουλος
Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Copyright © Kalaitzidis Kleovoulos, 2014
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Στην Οικογένεια μου και στους φίλους μου

Ευχαριστίες

Με την περάτωση της παρούσας διπλωματικής εργασίας και φτάνοντας στο τέλος των προπτυχιακών σπουδών μου, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που με βοήθησαν να φτάσω ως εδώ. Την οικογένεια μου για την αμέριστη συμπαράσταση και την ανεκτίμητη βοήθεια καθ' όλη την διάρκεια των σπουδών μου. Τους επιβλέποντες καθηγητές μου, κ. Γεώργιο Δημητρίου και κ. Γεώργιο Σταμούλη για την άριστη συνεργασία, την καθοδήγηση και τις υποδείξεις κατά την εκπόνηση της διπλωματικής εργασίας. Τέλος, όλους τους φίλους μου που ήταν δίπλα μου όλα αυτά τα χρόνια.

Καλαϊτζίδης Κλεόβουλος

Βόλος, 2014

Εισαγωγή

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη της προσομοίωσης μιας νέας αρχιτεκτονικής ενός υπερβαθμωτού επεξεργαστή, με δυνατότητα εκτίμησης της καταναλισκόμενης ισχύος κατά την εκτέλεση των εφαρμογών. Η αρχιτεκτονική που προσομοιώνουμε περιλαμβάνει εκτέλεση εκτός σειράς, επιτάχυνση βρόγχων και πλάνο για μείωση της απαιτούμενης ενέργειας. Ο υπολογισμός της ενέργειας γίνεται σε επίπεδο αρχιτεκτονικής μέσω του εργαλείου Wattch.

Αρχικά, η αρχιτεκτονική προσομοιώνεται με τον προσομοιωτή Simplescalar. Ο προσομοιωτής αυτός προσφέρει ένα πλήρες περιβάλλον για την προσομοίωση αρχιτεκτονικών διαθέτοντας τον δικό του assembler και cross-compiler. Η προσομοίωση αναπτύχθηκε σε γλώσσα προγραμματισμού C και εντάχθηκε στο περιβάλλον του Simplescalar ώστε να μπορεί κάθε εφαρμογή να εκτελείται μέσω της προσομοίωσης αυτής.

Μετέπειτα χρησιμοποιήθηκαν οι δυνατότητες που δίνει το framework του Wattch ώστε να εκτιμηθεί η ενέργεια που καταναλώνει το σύστημα. Οι δυνατότητες αυτές περιλαμβάνουν την κατηγοριοποίηση των διαθέσιμων πόρων και την ενσωμάτωση παραμετροποιημένων μοντέλων για κάθε κατηγορία πόρου, μέσω των οποίων υπολογίζεται η κατανάλωση ενέργειας για κάθε κύκλο μηχανής και αθροίζεται παρέχοντας ένα συνολικό αποτέλεσμα.

Συνολικά, η προσομοίωση αυτή βοηθάει στην αξιολόγηση της απόδοσης της νέας αυτής αρχιτεκτονικής και παρέχει μια πρώτη εικόνα των απαιτήσεων της σε ενέργεια.

Abstract

The purpose of this diploma thesis was the simulation of a new processor architecture of a superscalar processor, coupled with a power estimation model. This computer architecture includes out-of-order execution, loop acceleration and a method for keeping power consumption on a low level. The power consumption is being estimated by using the tool Wattch.

At first, the basic tool for simulating the aforementioned processor architecture was the simulator named Simplescalar. This simulator offers a complete environment, which includes an assembler and a cross-compiler, both of them tailor-made for the simulator. We developed the simulation using the C language and afterwards, we included it in the Simplescalar's environment, giving us the opportunity to execute any application over this simulation.

Subsequently, we manipulated the methods that the tool Wattch offers in order to calculate the power dissipation. These methods categorize the hardware of the system in different groups and uses specific power models to estimate the power consumption for every cycle, and also for the overall execution of an application.

All in all, this simulation helps us to assess the performance of this new processor architecture by giving us a first sight of its power demands.

Περιεχόμενα

1. Εισαγωγή

1.1 Περιγραφή του προβλήματος και συμβολή της εργασίας

Η απαίτηση για συνεχή αύξηση της απόδοσης των επεξεργαστών, έφερε στην επιφάνεια τους επεξεργαστές πολλαπλών πυρήνων (multi-core) αλλά και τους πολυνηματικούς επεξεργαστές (multi-threaded). Όμως η επίτευξη υψηλής απόδοσης σε αυτά τα μοντέρνα συστήματα αποτελεί ένα περίπλοκο έργο καθώς η ολοένα αύξηση των πυρήνων ή και των νημάτων ανά επεξεργαστή έχουν μεγαλύτερες απαιτήσεις σε ό,τι αφορά την κατανάλωση ισχύος. Άμεση συνέπεια της επαυξημένης κατανάλωσης ισχύος είναι και η αύξηση της εκπεμπόμενης θερμότητας του επεξεργαστή, κάτι το οποίο είναι αρκετά ζημιογόνο ειδικά για τους επεξεργαστές που προορίζονται για ενσωματωμένα συστήματα.

Έτσι λοιπόν, με την ταχεία εξέλιξη και την επιστημονική έρευνα των υπολογιστικών συστημάτων, η κατανάλωση ενέργειας αποτελεί τροχοπέδη για την εισαγωγή νέων αρχιτεκτονικών υπολογιστών. Πλέον, η αξιολόγηση μιας καινούριας αρχιτεκτονικής δεν περιορίζεται μόνο στην ταχύτητα εκτέλεσης που προσφέρει αλλά και στις απαιτήσεις σε κατανάλωση ενέργειας που έχει. Επίσης, ο ορισμός των πολυνηματικών αρχιτεκτονικών είναι η παράλληλη εκτέλεση του κώδικά που δίνουμε στην μηχανή. Έτσι λοιπόν για να εκμεταλλευτούμε μια τέτοια αρχιτεκτονική θα πρέπει ο κώδικας να έχει δυνατότητες να γίνει παράλληλος και να μπορεί να εκτελεστεί από διαφορετικά νήματα ταυτόχρονα. Στην περίπτωση όμως που οι ενδογενείς εξαρτήσεις του κώδικα δεν το επιτρέπουν αυτό τότε ο παραλληλισμός δεν επιτυγχάνεται και η απόδοση μειώνεται κατακόρυφα φτάνοντας τα όρια των απλών αρχιτεκτονικών.

Το συμπέρασμα είναι ότι οι τεχνικές παράλληλων νημάτων μπορούν να προσφέρουν μεγάλη απόδοση με την προϋπόθεση εξίσου μεγάλης απαίτησης σε ισχύ. Αλλά παρόλα αυτά η απόδοση δεν μπορεί να είναι σταθερά μέγιστη διότι ο παραλληλισμός μπορεί να αποτυγχάνει. Έτσι λαμβάνοντας υπόψιν και την κατανάλωση ενέργειας ως ένα μέτρο

απόδοσης, καταλήγουμε ότι υπάρχουν ακόμα μεγάλες απαιτήσεις για την εμφάνιση νέων αρχιτεκτονικών που θα έχουν κεντρικό άξονα την φύση της εφαρμογής για να πετύχουν γρήγορη εκτέλεση αλλά και να έχουν χαμηλές απαιτήσεις ισχύος.

Σκοπός της εργασίας αυτής είναι η προσομοίωση της κατανάλωσης ενέργειας μιας νέας αρχιτεκτονικής, η οποία εκτός από την γρήγορη εκτέλεση, στόχο έχει και την χαμηλή κατανάλωση ενέργειας. Κεντρικά χαρακτηριστικά της η ταχεία εκτέλεση βρόχων και η ελαχιστοποίηση της απαιτούμενης ενέργειας κατά την εκτέλεση, τα οποία θα αναλυθούν και θα παρουσιαστούν αναλυτικά στην συνέχεια της εργασίας.

1.2 Διάρθρωση της εργασίας

Ο τρόπος με τον οποίο αναπτύσσεται η παρούσα εργασία προσπαθεί να ακολουθήσει μια λογική ροή για την ομαλή ένταξη του αναγνώστη στο θέμα και μετέπειτα την ανάπτυξη του θέματος.

Αρχικά λοιπόν, πέραν της εισαγωγής, το δεύτερο κεφάλαιο ασχολείται με την παρουσίαση κάποιων βασικών αρχιτεκτονικών που χρησιμοποιήθηκαν ή και που χρησιμοποιούνται, έτσι ώστε το τρίτο κεφάλαιο να έρθει με την σειρά του και να παρουσιάσει την νέα αρχιτεκτονική. Στο τέταρτο και πέμπτο κεφάλαιο περιγράφονται αναλυτικά τα εργαλεία που χρησιμοποιήθηκαν για την προσομοίωση της νέας αρχιτεκτονικής, ενώ στο έκτο κεφάλαιο υπάρχει λεπτομερής παρουσίαση της προσομοίωσης αυτής. Έπειτα, στο έβδομο κεφάλαιο παρουσιάζονται κάποια πειραματικά αποτελέσματα της προσομοίωσης και στο όγδοο αναφέρονται οι προοπτικές για μελλοντική ανάπτυξη της. Τέλος, το παράρτημα περικλείει αποκλειστικές οδηγίες για την εγκατάσταση του εργαλείου Simplescalar.

2. Αρχιτεκτονικές Υπολογιστών

Η εξέλιξη και η ανάπτυξη διάφορων αρχιτεκτονικών κατά την ιστορία δείχνει ότι κάθε καινούρια τεχνολογία έχει επηρεαστεί άμεσα από κάποιες προηγούμενες ή ακόμα έχει υιοθετήσει και κάποια τεχνικά χαρακτηριστικά τους. Για τον λόγο αυτό, στο κεφάλαιο αυτό θα παρουσιάσουμε κάποιες βασικές αρχιτεκτονικές πάνω στις οποίες μπορούμε να πούμε ότι στηρίζεται η νέα αρχιτεκτονική που παρουσιάζεται στο κεφάλαιο 3 και που υλοποιείται από τον επεξεργαστή που προσομοιώσαμε στα πλαίσια αυτής της διπλωματικής εργασίας.

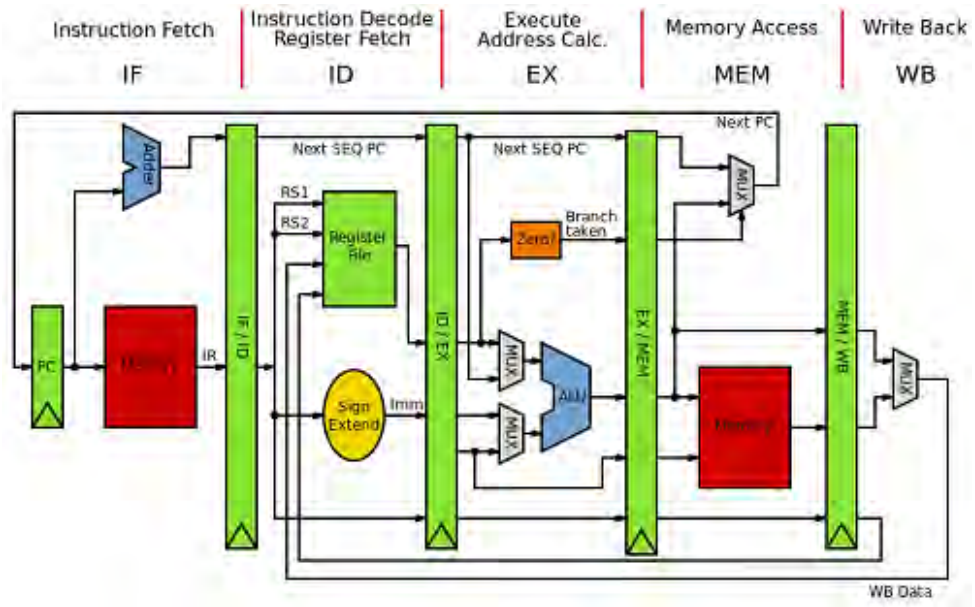
2.1 Η Αρχιτεκτονική CISC

Η αρχιτεκτονική τύπου CISC (Complex instruction set computing) χρησιμοποιούνταν ευρέως στους ηλεκτρονικούς υπολογιστές μέχρι τις αρχές του 1980. Ο κύριος στόχος της αρχιτεκτονικής αυτής είναι η εκτέλεση εντολών με όσο το δυνατόν λιγότερες εντολές χαμηλού επιπέδου (assembly). Αυτό επιτυγχάνεται αναπτύσσοντας επεξεργαστές με υλικό που τους επιτρέπει να εκτελούν μια σειρά από λειτουργίες χαμηλού επιπέδου (φόρτωση από την μνήμη, αριθμητικές λειτουργίες κτλ.) για κάθε ξεχωριστή σύνθετη εντολή assembly, όπως υποδηλώνει και το όνομα της. Έτσι για την εκτέλεση κάθε πολύπλοκης εντολής εκτελούνται πολλές ξεχωριστές διαφορετικές λειτουργίες. Για παράδειγμα, μια σύνθετη εντολή πολλαπλασιασμού σε επίπεδο assembly θα εκτελούσε τρείς διαφορετικές λειτουργίες. Πρώτον, την φόρτωση των τιμών από την μνήμη σε δυο διαφορετικούς καταχωρητές, δεύτερον την εκτέλεση του πολλαπλασιασμού μεταξύ των δυο αυτών τιμών και τρίτον την καταχώρηση του αποτελέσματος στον κατάλληλο καταχωρητή. Η εκτέλεση τέτοιου τύπου εντολών όμως απαιτεί μεγαλύτερο κύκλο μηχανής και έτσι η συχνότητα του ρολογιού οφείλει να είναι αρκετά χαμηλή σε σχέση με τις σημερινές αρχιτεκτονικές, δυσκολεύοντας σε μεγάλο βαθμό τους προγραμματιστές.

Η έμφαση λοιπόν της συγκεκριμένης τεχνολογίας δίνεται στο υλικό και στην ανάπτυξη ενός μειωμένου σετ εντολών. Κύριο μειονέκτημα της είναι η ύπαρξη σύνθετων εντολών που απαιτούν μεγάλο κύκλο μηχανής για να πραγματοποιήσουν τις λειτουργίες τους.

2.2 Η Αρχιτεκτονική RISC σε αντιδιαστολή με την CISC

Η αρχιτεκτονική τύπου RISC (Reduced instruction set computing) προτάθηκε για πρώτη φορά στις αρχές του 1970 και στις αρχές του 1980 δημιουργήθηκαν οι πρώτοι επεξεργαστές βασιζόμενοι σε αυτήν την τεχνολογία. Οι επεξεργαστές τύπου RISC δεν χρησιμοποιούν την λογική των σύνθετων εντολών της τεχνολογίας CISC. Κάθε εντολή εκτελεί μια συγκεκριμένη λειτουργία η οποία ολοκληρώνεται στα πλαίσια ενός κύκλου μηχανής, προκαλώντας έτσι την μείωση του εύρους του. Έτσι λοιπόν παίρνοντας και πάλι ως παράδειγμα έναν πολλαπλασιασμό σε επίπεδο assembly, η πράξη αυτή απαιτεί τρεις διαφορετικές λειτουργίες που επιτυγχάνονται με την εκτέλεση τριών διαφορετικών εντολών assembly και όχι μιας, όπως συμβαίνει με τις σύνθετες εντολές της αρχιτεκτονικής CISC.



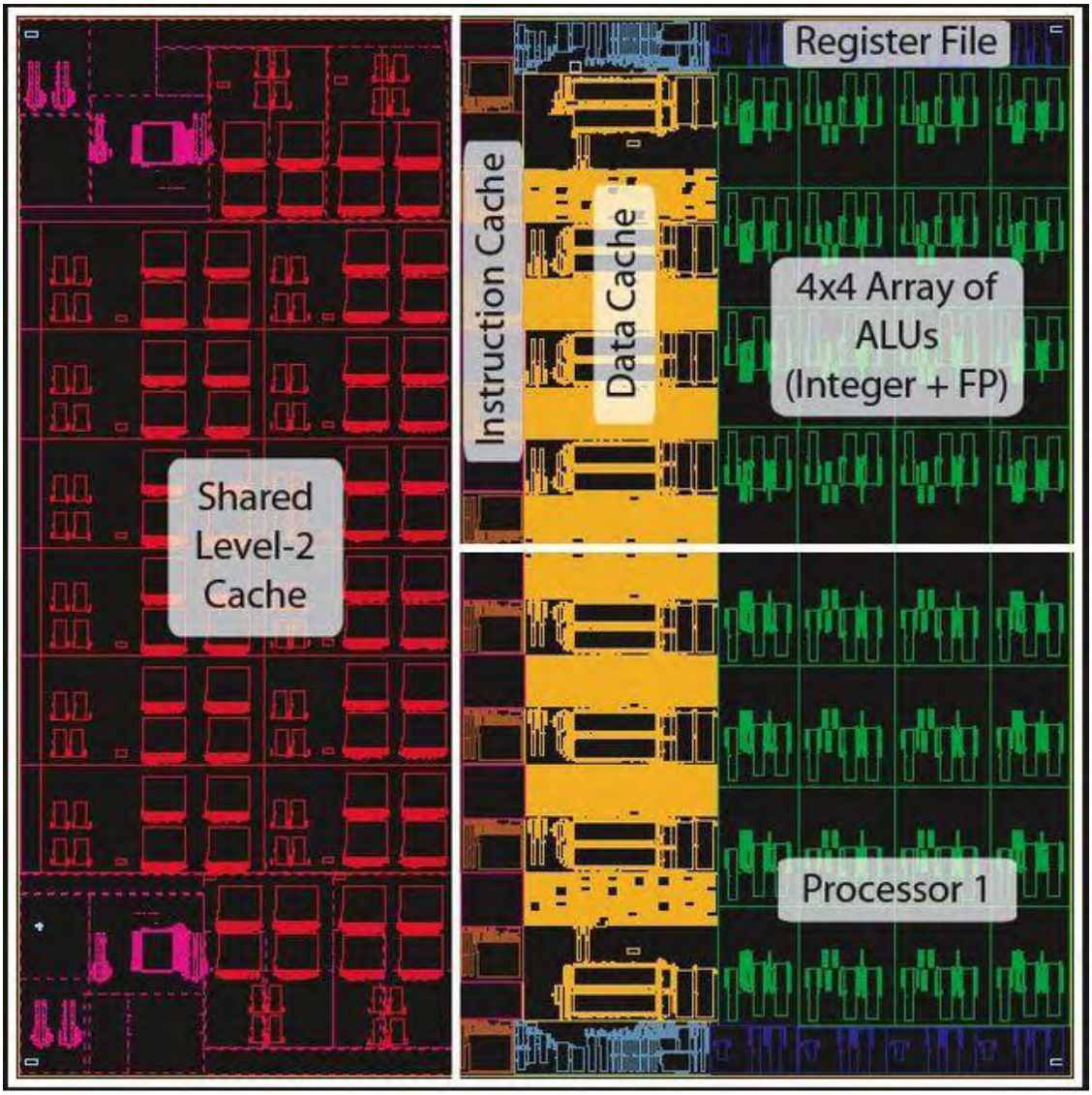
Εικόνα 1. Η μορφή του pipeline στον επεξεργαστή MIPS. Συνολικά 5 στάδια : instruction fetch, instruction decode, execute, memory access and write back .

Η πρώτη εντύπωση είναι ότι η τεχνολογία αυτή έχει χαμηλότερη απόδοση επειδή πρώτον, απαιτεί μεγαλύτερη RAM για την αποθήκευση του κώδικα assembly και δεύτερον, η δουλειά του μεταγλωττιστή για την μετάφραση κώδικα υψηλού επιπέδου σε εντολές χαμηλού επιπέδου αυτής της μορφής, είναι περισσότερη. Παρόλα αυτά, η φιλοσοφία της RISC έχει εξίσου σημαντικά πλεονεκτήματα που την κάνουν να έχει καλύτερη απόδοση. Η ανάγκη ενός κύκλου μηχανής για κάθε εντολή κάνουν τον συνολικό χρόνο εκτέλεσης ενός προγράμματος να είναι προσεγγιστικά ίδιος με αυτόν που θα είχε το πρόγραμμα αν εκτελούνταν με τις σύνθετες εντολές τύπου CISC. Επίσης οι εντολές RISC απαιτούν λιγότερα transistors και κατ' επέκταση λιγότερο χώρο στο υλικό, επιτρέποντας έτσι την ύπαρξη περισσότερου χώρου για την ενσωμάτωση μεγαλύτερου αριθμού καταχωρητών γενικού σκοπού (οι επεξεργαστές CISC έχουν περίπου οχτώ με δεκαέξι καταχωρητές γενικού σκοπού ενώ οι RISC μπορούν να έχουν διπλάσιο ή και τριπλάσιο αριθμό).

Τέλος, λόγω του σταθερού αυτού χρονικού διαστήματος εκτέλεσης κάθε εντολής,(σε έναν κύκλο μηχανής) υπάρχει η δυνατότητα για εκτέλεση εντολών με διασωλήνωση (pipelining), προσφέροντας παραλληλοποίηση στην εκτέλεση του κώδικα και κάνοντας την αρχιτεκτονική αυτή να ξεχωρίζει κατά πολύ από αυτή της CISC. Ο επεξεργαστής MIPS αποτελεί μια εφαρμογή της τεχνολογίας RISC, δημοσιεύτηκε το 1984 και η φιλοσοφία που ακολουθεί κατά την εκτέλεση των εντολών φαίνεται στην Εικόνα 1. Το pipeline της εκτέλεσης χωρίζεται σε πέντε στάδια και έτσι υπάρχει η δυνατότητα εκμετάλλευσης του παραλληλισμού σε επίπεδο εντολής. Κάθε εντολή περνάει με την σειρά από κάθε ξεχωριστό στάδιο του pipeline και έτσι ο κάθε επόμενος κύκλος μηχανής βρίσκει διαφορετικές εντολές σε διαφορετικά στάδια, εφαρμόζοντας έτσι παράλληλη εκτέλεση του κώδικα, ακολουθώντας πάντα τις εκάστοτε εξαρτήσεις των εντολών.

2.3 Η Αρχιτεκτονική EDGE και ο επεξεργαστής TRIPS

Η εμφάνιση του πρώτου επεξεργαστή τεχνολογίας EDGE (Explicit Data Graph Execution) έγινε το 2006 με την κατασκευή του επεξεργαστή TRIPS από μια ομάδα του πανεπιστημίου του Τέξας σε συνεργασία με τις εταιρείες IBM, Intel και Sun Microsystems.



Εικόνα 2. Η σχεδιαστική κάτοψη ενός επεξεργαστή TRIPS. Στην εικόνα φαίνονται ο επεξεργαστής, η cache, η διάταξη των οχτώ αριθμητικών λογικών μονάδων (ALU) , καθώς και οι caches εντολών και δεδομένων (instruction cache-data cache).

Βασικό χαρακτηριστικό της αρχιτεκτονικής EDGE είναι η άμεση επικοινωνία των εντολών μεταξύ τους. Αυτό σημαίνει πως το υλικό έχει την δυνατότητα να μεταφέρει το αποτέλεσμα της εκτέλεσης μιας εντολής στις εκάστοτε εντολές οι οποίες το χρειάζονται σαν είσοδο για την έναρξη της εκτέλεσης τους. Έτσι υιοθετείτε ένα σχήμα παραγωγού-καταναλωτή με ενεργή αναμονή του καταναλωτή μέχρι ο παραγωγός να διαθέσει το προϊόν, κάνοντας τις εντολές να εκτελούνται με μια διάταξη εξαρτώμενη από την ροή δεδομένων. Κάθε παραγωγός με πολλαπλούς καταναλωτές διαχωρίζει τον κάθε ένα ξεχωριστά και κατά την εκτέλεση προωθεί στην είσοδο του το αποτέλεσμα του, αποφεύγοντας να το γράψει σε κάποιο καταχωρητή όπου και έχουν πρόσβαση όλες οι εντολές.

Η τεχνολογία EDGE είναι η πρώτη αρχιτεκτονική που εκμεταλλεύεται τον παραλληλισμό σε επίπεδο ροής δεδομένων και που προσφέρει υψηλές δυνατότητες για παράλληλη εκτέλεση εντολών, κάτι που την καθιστά πρωτοπόρα. Εν γένει, ο ορισμός του ISA (Instruction Set Architecture) της EDGE προσφέρει ένα πλουσιότερο interface μεταξύ του μεταγλωττιστή και της μικρό-αρχιτεκτονικής. Αυτό συμβαίνει διότι το ίδιο το ISA ορίζει ξεκάθαρα τον γράφο μετάβασης των δεδομένων τον οποίο ο μεταγλωττιστής δημιουργεί εσωτερικά. Με αυτόν τον τρόπο δεν χρειάζεται το υλικό να επανεξετάζει και να ανακαλύπτει δυναμικά κατά την διάρκεια της εκτέλεσης, τις εξαρτήσεις από δεδομένα μεταξύ των εντολών.

Η αρχιτεκτονική EDGE έχει μεγαλύτερη απόδοση σε σχέση με τις αρχιτεκτονικές RISC και CISC λαμβάνοντας υπόψιν και την κατανάλωση ισχύος. Λόγω της άμεσης επικοινωνίας των εντολών και της προώθησης δεδομένων, το σύστημα δεν επιβαρύνεται με λειτουργίες όπως εγγραφή σε καταχωρητές, μετονομασία καταχωρητών, έλεγχο διαθέσιμων εισόδων κάθε εντολής στην διαδικασία αποστολής στις λειτουργικές μονάδες, πρόβλεψη διακλαδώσεων κτλ. Λειτουργίες βασικές στην εκτέλεση εκτός σειράς RISC/CISC. Το σύστημα λοιπόν αποφεύγει πλήθος λειτουργιών που το επιβαρύνουν στο ποσό της καταναλισκόμενης ενέργειας και έτσι καταφέρνει και παράλληλη εκτέλεση στηριζόμενο στην ροή δεδομένων, αλλά και χαμηλή κατανάλωση ενέργειας.

3. Παρουσίαση της νέας Αρχιτεκτονικής

3.1 Οι στόχοι

Όπως αναφέραμε και στην εισαγωγή της εργασίας, η εξέλιξη των υπολογιστικών συστημάτων και η εμφάνιση τεχνολογιών που προσφέρουν μείωση του χρόνου εκτέλεσης των προγραμμάτων, καθιστά αναγκαίο το γεγονός της εξίσου μελέτης και εμφάνισης νέων αρχιτεκτονικών που θα προσφέρουν και χαμηλή κατανάλωση ισχύος. Προτέρημα ενός συστήματος δεν αποτελεί μόνο η εκμετάλλευση του πιθανού παραλληλισμού για την βέλτιστη εκτέλεση, αλλά και η ανάπτυξη του έτσι ώστε η εκτέλεση να συνδυάζεται με χαμηλές απαιτήσεις ενέργειας.

Η νέα αρχιτεκτονική που θα παρουσιάσουμε έχει όμοιους στόχους με αυτούς της τεχνολογίας EDGE και του επεξεργαστή TRIPS. Το σετ εντολών της είναι βασισμένο σε αυτό του MIPS εμπεριέχοντας εντολές χαμηλού επιπέδου της μορφής RISC αλλά και καινούριες εντολές που ενσωματώνουν νέες λειτουργίες. Η λειτουργικότητα των νέων εντολών έχει να κάνει με την εισαγωγή ενός νέου τρόπου εκτέλεσης βρόχων με σκοπό την επιτάχυνση τους. Εκτός όμως αυτού, βασικός στόχος αυτής της αρχιτεκτονικής είναι η ανάπτυξη επεξεργαστών που θα εκμεταλλεύονται τον παραλληλισμό σε επίπεδο εντολής αλλά και σε επίπεδο ροής δεδομένων προϋποθέτοντας όμως και χαμηλή καταναλισκόμενη ενέργεια.

Πιο αναλυτικά, η κύρια ιδέα είναι η δημιουργία συστημάτων τα οποία θα μπορούν να κλιμακώνουν την απόδοση ανεξάρτητα από την δυνατότητα παραλληλισμού των εφαρμογών και τον αριθμό των επεξεργαστών που διαθέτουν. Οι σημερινοί multicore και multithreaded επεξεργαστές καταφέρνουν να ενισχύσουν την απόδοση χωρίς όμως δυνατότητες υψηλής κλιμάκωσης, λόγω της άμεσης εξάρτησης τους από τους λόγους που αναφέραμε προηγουμένως. Έτσι λοιπόν η νέα αρχιτεκτονική ενσωματώνει έναν σχεδιασμό που προσφέρει δυνατότητες για υψηλή κλιμάκωση της απόδοσης.

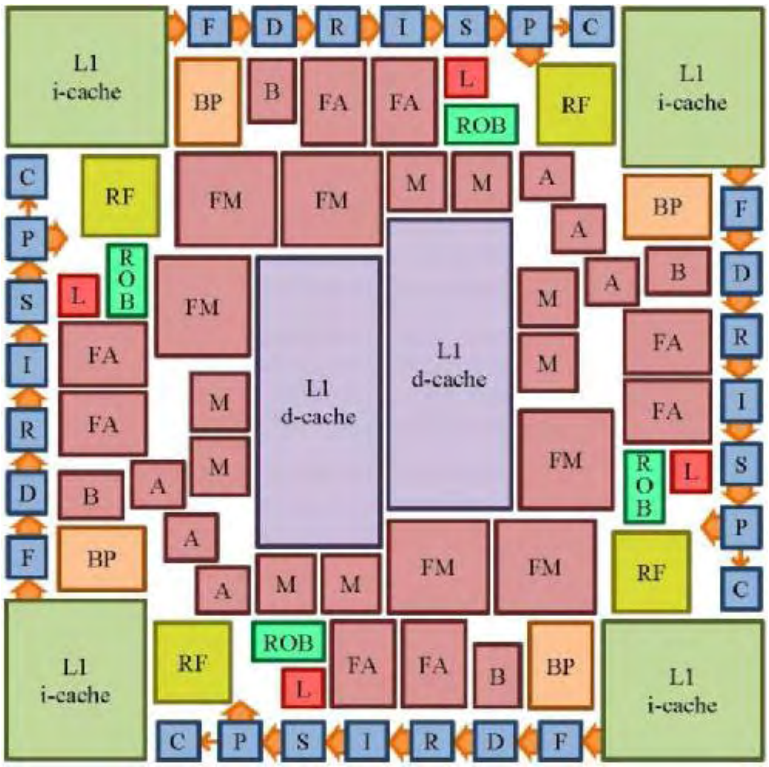
Εν κατακλείδι, οι δύο βασικοί στόχοι και λόγοι δημιουργίας της συγκεκριμένης αρχιτεκτονικής είναι η ελαχιστοποίηση του χρόνου εκτέλεσης των βρόγχων αλλά και η μείωση της καταναλισκόμενης ενέργειας του συστήματος. Έτσι θεωρώντας ότι σε κάθε εφαρμογή η μεγαλύτερη καθυστέρηση αποδίδεται στην εκτέλεση των βρόγχων, με αυτήν την νέα μέθοδο η αρχιτεκτονική καταφέρνει να μειώσει τον χρόνο εκτέλεσης των εφαρμογών συνολικά. Ο τρόπος με τον οποίο τα καταφέρνει αυτά παρουσιάζεται παρακάτω και τεκμηριώνεται σε επόμενα κεφάλαια μέσω της προσομοίωσης που αναπτύξαμε.

3.2 Ανάλυση της μικροαρχιτεκτονικής

3.2.1 Το μοντέλο του επεξεργαστή

Οι τυπικοί multicore επεξεργαστές αποτελούνται από πολλούς ανεξάρτητους πυρήνες οι οποίοι επικοινωνούν μεταξύ τους μέσω της μνήμης. Στην ουσία η συνολική έρευνα που έχει γίνει σε ό,τι αφορά την παράλληλη εκτέλεση, έχει εφαρμοστεί σε τέτοιου είδους συστήματα, κάνοντας έτσι εύκολη την μεταφορά από τεχνολογίες single-core σε multi-core. Όμως, ο σχεδιασμός απλών multicore επεξεργαστών βάση των σύνθετων single-core ωθεί στον παραμερισμό της ανάπτυξης πιθανών τεχνολογιών υψηλών επιδόσεων τύπου single-core. Η τάση αυτής της εξέλιξης είναι να αγνοούνται κάποιες ιδιαίτερα σημαντικές λειτουργίες όπως η εκτέλεση εκτός σειράς εντολών δυναμικά δρομολογημένων, ο υποθετικός πολυνηματισμός, η εκτέλεση βάση ροής δεδομένων. Σε αντίθεση, υιοθετούνται τεχνολογίες που χρησιμοποιούν συν-επεξεργαστές, τους γνωστούς co-processors. Οι co-processors είναι επεξεργαστές στοχευμένης λειτουργικότητας και δουλεύουν σε περίπτωση που ο κύριος επεξεργαστής τους αναθέσει κάποια συγκεκριμένη εκτέλεση. Με αυτόν τον τρόπο επιτυγχάνεται η επιτάχυνση της εκτέλεσης, εφόσον τα τμήματα εφαρμογών που απαιτούν πολύπλοκο τρόπο επεξεργασίας ανατίθενται στους ειδικά σχεδιασμένους για αυτόν τον σκοπό co-processors. Ένα βασικό παράδειγμα τέτοιου είδους επεξεργαστών

είναι οι μονάδες GPUs που διαθέτουν επεξεργαστές τύπου vector, οι οποίοι όμως έχουν υψηλό κόστος και υψηλή κατανάλωση ενέργειας. Για αυτόν τον λόγο δεν χρησιμοποιούνται σαν επεξεργαστές γενικού σκοπού, παρά μόνο σαν co-processors για την εκτέλεση εφαρμογών που περιλαμβάνουν επεξεργασία γραφικών υπολογισμών(ετερογενείς αρχιτεκτονικές) ή σε επιστημονικές εφαρμογές μεγάλων απαιτήσεων.



Εικόνα 3. Μοντέλο ενός επεξεργαστή που ενσωματώνει 4 pipelines. Στην εικόνα φαίνονται τα 4 pipelines με τις λειτουργικές μονάδες.

Έχοντας λοιπόν υπόψιν όλα τα παραπάνω, μπορεί να γίνει κατανοητή η σημαντικότητα της νέας αρχιτεκτονικής που παρουσιάζουμε. Βασικό της χαρακτηριστικό είναι ότι υλοποιεί πολλές λειτουργικότητες των multicore επεξεργαστών παρότι δεν είναι αρχιτεκτονική τύπου multicore. Ουσιαστικά ο σχεδιασμός της είναι εκ διαμέτρου αντίθετος με αυτόν των multicore επεξεργαστών επειδή προϋποθέτει την ύπαρξη πολλαπλών ροών εντολών(instruction pipelines) που μοιράζονται ένα κοινό δίκτυο πόρων και όχι ξεχωριστά δίκτυα. Το κοινό αυτό δίκτυο(κοινώς το back-end) αποτελείται από πλήθος λειτουργικών μονάδων οι οποίες είναι απαραίτητες κατά την εκτέλεση κάθε ξεχωριστού σταδίου pipeline. Με αυτόν τον τρόπο ο επεξεργαστής υλοποιεί το πιο βασικό χαρακτηριστικό του multithreading, δηλαδή την από κοινού χρησιμοποίηση των πόρων του συστήματος(resource sharing). Έτσι με τον ίδιο τρόπο που χαρακτηρίζουμε τους multicore επεξεργαστές ανάλογα με τον αριθμό των πυρήνων που διαθέτουν(π.χ. dual-core,

octa-core), με τον ίδιο τρόπο θα ονομάζαμε και τον επεξεργαστή της εικόνας 3 ως έναν quad-pipeline επεξεργαστή.

Αναλύοντας την εικόνα 3 μπορούμε να δούμε τα τέσσερα ξεχωριστά pipelines, με το καθένα να έχει σαν σημείο έναρξης την αντίστοιχη μνήμη εντολών(i-cache). Τα στάδια του pipeline είναι εφτά, η προσκόμιση(fetch F) και η αποκωδικοποίηση(decode D) των εντολών, η μετονομασία των καταχωρητών(rename R), η έκδοση(issue I) των εντολών στις λειτουργικές μονάδες, η δρομολόγηση τους για εκτέλεση(schedule S), η εκτέλεση(dispatch P) και τέλος η απόσυρση (commit C) των εντολών. Στις επόμενες ενότητες ακολουθεί αναλυτική περιγραφή του δικτύου των πόρων και της διαδικασίας εκτέλεσης των εντολών.

3.2.2 Pipelines

Όπως αναφέραμε και παραπάνω, η ενσωμάτωση τεσσάρων pipelines που μοιράζονται τους πόρους του συστήματος μέσω ενός δικτύου, κάνει την νέα αυτή αρχιτεκτονική να είναι άκρως αντίθετη με τις αρχιτεκτονικές των multicore επεξεργαστών. Βλέποντας αναλυτικά το pipeline θα περιγράψουμε κάθε στάδιο από το οποίο περνούν οι εντολές.

Στάδιο FETCH: Κατά το στάδιο αυτό έχουμε την πρόσβαση στην i-cache και την προσκόμιση των εντολών για την περαιτέρω επεξεργασία. Το εύρος του κάθε pipeline ξεχωριστά μπορεί να οριστεί και να είναι από ένα έως τέσσερα.

Στάδιο DECODE: Μετά την προσκόμιση των εντολών σειρά έχει η αποκωδικοποίηση τους, πράγμα το οποίο γίνεται στο συγκεκριμένο στάδιο και έπειτα ακολουθεί η μετονομασία των καταχωρητών.

Στάδιο RENAME: Κατά την μετονομασία των καταχωρητών γίνεται αντιστοίχιση των αποτελεσμάτων της εντολής στους φυσικούς καταχωρητές για λόγους εξαρτήσεων από δεδομένα αλλά και για υποθετική εκτέλεση σε περίπτωση άλματος. Το σύστημα περιλαμβάνει 128 φυσικούς καταχωρητές και όμοια ο πίνακας αναδιάταξης(reorder buffer-ROB)έχει 128 θέσεις. Στο τέλος της διαδικασίας αυτής καταλαμβάνεται μια θέση στον πίνακα αναδιάταξης για κάθε εντολή.

Στάδιο ISSUE: Εφόσον οι εντολές έχουν αποκωδικοποιηθεί επόμενο βήμα είναι οι εντολές να καταλάβουν την συγκεκριμένη λειτουργική μονάδα που χρειάζονται για την εκτέλεση τους. Εφόσον λοιπόν ελεγχθεί η διαθεσιμότητα της εκάστοτε μονάδας, η εντολή εκδίδεται και με πιθανή διάταξη εκτός σειράς αφού η έκδοση εξαρτάται μόνο από την διαθεσιμότητα των πόρων. Μετά την έκδοση, η εκάστοτε εντολή βρίσκεται στο σταθμό(reservation station)της αντίστοιχης λειτουργικής μονάδας. Οι εξαρτημένες εντολές τοποθετούνται και αποστέλλονται σε συγκεκριμένες μονάδες οι οποίες είναι άμεσα συνδεδεμένες με την μονάδα που θα διαθέσει το ζητούμενο δεδομένο. Αν αυτό δεν είναι δυνατό, τότε η εντολή εκδίδεται σε κάποια άλλη μονάδα σε μεγαλύτερη απόσταση.

Στάδιο SCHEDULE: Σειρά έχει η δρομολόγηση των εντολών προς εκτέλεση. Οι εντολές οι οποίες έχουν διαθέσιμα δεδομένα προς εκτέλεση δρομολογούνται προς εκτέλεση.

Στάδιο DISPATCH: Η εκτέλεση των εντολών πραγματοποιείται παρέχοντας στην μονάδα τα τελούμενα της εντολής η οποία τελικώς ξεκινάει την εκτέλεση της εκάστοτε λειτουργίας για όσους κύκλους μηχανής απαιτούνται. Στο τέλος της εκτέλεσης, η έξοδος της μονάδας αποστέλλεται στις μονάδες που έχουν άμεση εξάρτηση και περιμένουν τα δεδομένα για να αρχίσουν την εκτέλεση τους. Επίσης το αποτέλεσμα αποστέλλεται και στον πίνακα αναδιάταξης και μετέπειτα ο σταθμός που καταλάμβανε η εντολή απελευθερώνεται.

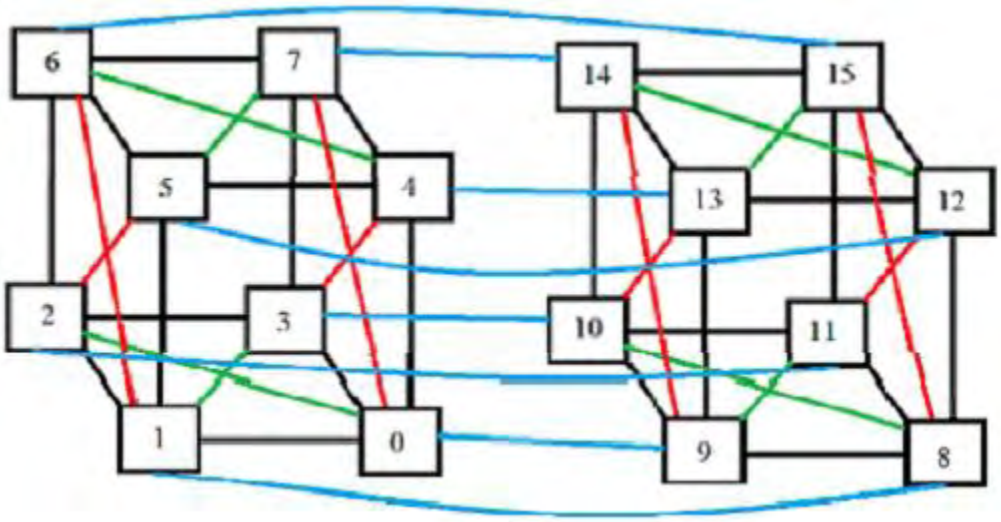
Στάδιο COMMIT: Εφόσον η εντολή έχει εκτελεστεί και έχει αποστείλει της έξοδο της μπορεί να αποσυρθεί, μόνο όμως σε διάταξη εντός σειράς. Μπορεί η έκδοση και η εκτέλεση των εντολών να γίνεται εκτός σειράς αλλά η απόσυρση που συνεπάγεται την αποστολή του αποτελέσματος είτε στον φάκελο καταχωρητών(RF)είτε στην μνήμη(d-cache) πρέπει να γίνεται πάντα εντός σειράς. Κατά την απόσυρση αποδεσμεύεται και η αντίστοιχη θέση στον πίνακα αναδιάταξης.

3.2.3 Το κοινό δίκτυο λειτουργικών μονάδων

Αρχικά να αναφέρουμε ότι με τον όρο λειτουργική μονάδα αναφερόμαστε σε κάθε πόρο-στοιχείο του υλικού που δέχεται στοιχεία από τα στάδια issue και dispatch του pipeline των εντολών, για να ολοκληρώσει μια συγκεκριμένη λειτουργία επί της εντολής.

Η κύρια ιδέα της τεχνολογίας που αναλύουμε είναι η άμεση διασύνδεση των λειτουργικών μονάδων με τέτοιο τρόπο ώστε τα δεδομένα να προωθούνται σε περίπτωση εξαρτήσεων μεταξύ των εντολών, χωρίς όμως μεγάλη καθυστέρηση. Για να γίνει αυτό σημαντικό ρόλο παίζει ο ιδιαίτερος σχεδιασμός της τοπολογίας του δικτύου των λειτουργικών μονάδων. Η αρχιτεκτονική ενσωματώνει μια ιδιαίτερη τοπολογία που στόχο έχει να ελαχιστοποιήσει τον αριθμό των hops(μεταπηδήσεων) που χρειάζονται για την εκάστοτε μεταφορά της πληροφορίας. Το τελευταίο είναι το κύριο μειονέκτημα του TRIPS, ο οποίος χρησιμοποιεί την τοπολογία ¨ένα προς ένα¨ (mesh network) για την επικοινωνία των λειτουργικών μονάδων. Αυτό προκαλεί αρκετή καθυστέρηση λόγω της απαίτησης ενός μεγάλου αριθμού hops για την μεταφορά δεδομένων, καθώς και ενός εκλεπτυσμένου κυκλώματος ελέγχου για τον εντοπισμό του μονοπατιού.

Η τοπολογία και οι διασυνδέσεις των κόμβων-πόρων του συστήματος φαίνεται στην παρακάτω εικόνα.



Εικόνα 4. Το δίκτυο ροής δεδομένων του συστήματος(the hypercube-based dataflow network).

Ο κάθε κόμβος του δικτύου αυτού φιλοξενεί μια συγκεκριμένη μονάδα εκτέλεσης, περιορίζοντας το σύστημα στην ενσωμάτωση δεκαέξι μονάδων οποιασδήποτε κατηγορίας. Ο συγκεκριμένος σχεδιασμός βασίζεται στον κλασικό σχεδιασμό δικτύων της μορφής hypercube αλλά διαθέτει περισσότερες συνδέσεις μεταξύ των κόμβων. Έτσι σε

αντιδιαστολή με το mesh network του TRIPS, η συγκεκριμένη μορφή του δικτύου επιτρέπει στην αρχιτεκτονική να μειώσει την καθυστέρηση της προώθησης των δεδομένων επιλέγοντας πρώτιστα να εκδώσει τις εξαρτημένες εντολές σε κόμβους αποστάσεως ενός hop από τον κόμβο που παράγει το ζητούμενο δεδομένο.

Εφόσον κάθε κόμβος αντιπροσωπεύει και μια μονάδα επεξεργασίας χρειάστηκε μελέτη για τον ορισμό του είδους, της θέσης και του πλήθους των μονάδων κάθε κατηγορίας που θα ενσωματωθούν. Τελικώς η αρχιτεκτονική υποστηρίζει 5 διαφορετικά είδη μονάδων, τα οποία παραθέτουμε παρακάτω.

Μονάδα πρόσθεσης αριθμών κινητής υποδιαστολής(FA):

Η μονάδα αυτή εκτελεί πρόσθεση και αφαίρεση ακεραίων αριθμών αλλά και αριθμών κινητής υποδιαστολής, αριστερή και δεξιά λογική και αριθμητική ολίσθηση, μετατροπή ακεραίων σε αριθμούς κινητής υποδιαστολής και λογικές πράξεις. Ο συνδυασμός και των δύο ειδών προσθέσεων βασίζεται στο γεγονός ότι κατά τις προσθέσεις αριθμών κινητής υποδιαστολής εκτελούνται εξίσου προσθέσεις ακεραίων αριθμών για τον υπολογισμό του συντελεστή και ολισθήσεις για τον υπολογισμό του εκθέτη. Αντίστοιχα, κατά την μετατροπή ακεραίων σε αριθμούς κινητής υποδιαστολής, γίνεται ολίσθηση για τον υπολογισμό του συντελεστή και ακέραια πρόσθεση για τον υπολογισμό του εκθέτη. Το σύστημα περιλαμβάνει τέσσερις τέτοιες μονάδες που βρίσκονται στους κόμβους 1 ,3, 12 και 14.

Μονάδα πολλαπλασιασμού αριθμών κινητής υποδιαστολής(FM):

Η μονάδα αυτή εκτελεί πολλαπλασιασμό, πρόσθεση, και αφαίρεση ακεραίων αλλά και αριθμών κινητής υποδιαστολής και λογικές πράξεις. Η μονάδα υποστηρίζει και πάλι τις αντίστοιχες πράξεις σε ακέραιους που περιλαμβάνει ο πολλαπλασιασμός αριθμών κινητής υποδιαστολής. Το σύστημα περιλαμβάνει τρείς τέτοιες μονάδες που βρίσκονται στους κόμβους 2, 5 και 8.

Αριθμητική λογική μονάδα(ALU):

Η μονάδα αυτή εκτελεί πρόσθεση και αφαίρεση ακεραίων, αριστερή και δεξιά λογική και αριθμητική ολίσθηση, μετατροπή αριθμών κινητής υποδιαστολής σε ακέραιους και λογικές πράξεις. Το σύστημα περιλαμβάνει τρείς τέτοιες μονάδες που βρίσκονται στους κόμβους 0,13 και 15.

Μονάδα μνήμης(M):

Η μονάδα αυτή έχει πρόσβαση στην μνήμη δεδομένων(d-cache) και μπορεί να εκτελέσει πρόσθεση και αφαίρεση ακεραίων. Τον λόγο για την ιδιαίτερη εμφάνιση αριθμητικών πράξεων σε μια μονάδα μνήμης θα τον εξηγήσουμε παρακάτω. Το σύστημα περιλαμβάνει τέσσερις τέτοιες μονάδες που βρίσκονται στους κόμβους 4, 7, 10 και 9.

Μονάδα εκτέλεσης διακλαδώσεων(B):

Η μονάδα αυτή μπορεί να εκτελέσει συγκεκριμένες εντολές διακλάδωσης καθώς και πρόσθεση και αφαίρεση ακεραίων για λόγους που όμοια θα εξηγήσουμε παρακάτω. Το σύστημα περιλαμβάνει δυο τέτοιες μονάδες που βρίσκονται στους κόμβους 6 και 11.

3.2.4 Το σύνολο εντολών της νέας Αρχιτεκτονικής

Όπως έχουμε αναφέρει, η αρχιτεκτονική βασίζεται στον επεξεργαστή MIPS της τεχνολογίας RISC. Έτσι λοιπόν περιλαμβάνει όλες τις εντολές χαμηλού επιπέδου-assembly τύπου RISC αλλά με κάποιες ακόμα συγκεκριμένες προσθήκες που της προσδίδουν καινούριες λειτουργικότητες. Το κλασικό σετ εντολών έχει επεκταθεί και περιλαμβάνει τέσσερις νέες εντολές φόρτωσης από την μνήμη, τέσσερις νέες εντολές αποθήκευσης στην μνήμη και 4 νέες εντολές διακλάδωσης. Στον πίνακα της επόμενης σελίδας παρουσιάζουμε τις νέες εντολές που έχουν ενσωματωθεί.

Πίνακας 1. Οι νέες εντολές και η λειτουργικότητα της κάθε εντολής μέσω απλών εντολών MIPS.

Εντολή	**Λειτουργικότητα**
lwi1 $t0,0($s1)	lw $t0,0($s1) , addi $s1,$s1,1
lwi4 $t0,0($s1)	lw $t0,0($s1) , addi $s1,$s1,4
lwd1 $t0,0($s1)	lw $t0,0($s1) , addi $s1,$s1,-1
lwd4 $t0,0($s1)	lw $t0,0($s1) , addi $s1,$s1,-4
swi1 $t0,0($s1)	sw $t0,0($s1) , addi $s1,$s1,1
swi4 $t0,0($s1)	sw $t0,0($s1) , addi $s1,$s1,4
swd1 $t0,0($s1)	sw $t0,0($s1) , addi $s1,$s1,-1
swd4 $t0,0($s1)	sw $t0,0($s1) , addi $s1,$s1,-4
lbeqi1 $s0,$s1,L	addi $s0, $s0,1 , beq $s0,$s1,L
lbeqi4 $s0,$s1,L	addi $s0, $s0,4 , beq $s0,$s1,L
lbeqd1 $s0,$s1,L	addi $s0, $s0,-1 , beq $s0,$s1,L
lbeqd4 $s0,$s1,L	addi $s0, $s0,-4 , beq $s0,$s1,L

Όπως φαίνεται λοιπόν και από την επεξήγηση της λειτουργικότητας κάθε εντολής στον πίνακα, οι νέες αυτές εντολές εκτελούν 2 μορφές πράξεων, είτε μια πρόσβαση στην μνήμη και μια πρόσθεση/αφαίρεση, είτε μια πρόσθεση/αφαίρεση και μια απλή εντολή διακλάδωσης. Οι νέες εντολές πρόσβασης στην μνήμη, εφόσον ολοκληρώσουν την φόρτωση/αποθήκευση του δεδομένου από την μνήμη, αυξάνουν/μειώνουν κατά ένα ή κατά τέσσερα τον δείκτη(index) στην μνήμη. Αντίστοιχα οι νέες εντολές διακλάδωσης, αφού πρώτα εκτελέσουν την αύξηση/μείωση κατά ένα ή τέσσερα του καταχωρητή ελέγχου(loop index), εκτελούν την απλή εντολή διακλάδωσης MIPS χρησιμοποιώντας την καινούρια τιμή του καταχωρητή. Στην συνέχεια παρουσιάζουμε ακριβώς τον λόγο για τον οποίο οι εντολές αυτές είναι χρήσιμες στην λειτουργία της εκτέλεσης.

3.2.5 Η επιτάχυνση βρόγχων και η διαχείριση ενέργειας

Η ύπαρξη των νέων εντολών που αναφέραμε στην ενότητα 3.2.4 εξυπηρετεί την ύπαρξη της ιδιαίτερης δομής rapid loop. Μέσω αυτής, επιτυγχάνεται η επιτάχυνση της εκτέλεσης των βρόγχων και κατ' επέκταση της εφαρμογής γενικότερα. Ακόμη, η σχεδίαση συμπεριλαμβάνει και πλάνο για την εξοικονόμηση της καταναλισκόμενης ισχύος.

Αναλυτικότερα, η δομή rapid loop ξεκινάει με την άφιξη μιας εντολής διακλάδωσης από τις τέσσερις νέες εντολές που περιγράψαμε στην προηγούμενη ενότητα. Η ιδέα είναι ότι οι εντολές που ανήκουν στον ιδιαίτερο βρόγχο rapid loop αποκωδικοποιούνται και εκδίδονται στην αντίστοιχη μονάδα μια φορά και μετέπειτα εκτελούνται για όσες επαναλήψεις είναι απαραίτητες χωρίς να χρειαστεί η εκ νέου επεξεργασία τους για να φτάσουν στο σημείο εκτέλεσης. Έτσι λοιπόν, κατά την άφιξη των ειδικών εντολών διακλαδώσεων, ο μεταγλωττιστής αναγνωρίζει την έναρξη ενός rapid loop. Κάθε επόμενη εντολή η οποία βρίσκεται εντός των ορίων της διεύθυνσης διακλάδωσης κατηγοριοποιείται ως μια εντολή rapid loop. Οι εντολές αυτές αποκωδικοποιούνται και εκδίδονται μαζικά στις λειτουργικές μονάδες και μένουν εκεί για όσες επαναλήψεις πρέπει να εκτελεστούν. Ο δρομολογητής των εντολών εγγράφει κάθε εντολή στον πίνακα loop buffer. Σε κάθε κύκλο μηχανής κάθε εντολή η οποία βρίσκεται σε κάποια μονάδα εκτελείται εφόσον έχει διαθέσιμα τα δεδομένα της. Για να εξασφαλιστεί ότι οι εντολές θα εκτελεστούν τόσες φορές όσες και οι επαναλήψεις υπάρχει έλεγχος των εξαρτήσεων των εντολών. Κάθε εντολή πρέπει να έχει μια εξάρτηση από κάποια προηγούμενη της κι αν αυτό δεν συμβαίνει τότε η εντολή τίθεται σε εξάρτηση με την εντολή διακλάδωσης, η οποία σε κάθε εκτέλεση της στέλνει στις εξαρτημένες εντολές την τιμή του loop index με την μορφή ενός token. Έτσι οι εντολές αλυσιδωτά εκτελούνται τόσες φορές όσα και τα tokens που αποστέλλει η ειδική εντολή διακλάδωσης. Η μορφή της εκτέλεσης λοιπόν μοιάζει με μια συνεχή ροή δεδομένων και κάθε εκτέλεση της εντολής διακλάδωσης σηματοδοτεί και την εκτέλεση μιας

επανάληψης του βρόγχου. Όταν η συνθήκη διακλάδωσης γίνει αληθής τότε δεν αποστέλλονται άλλα tokens στις εντολές και εφόσον καταναλώσουν όλα τα εισερχόμενα δεδομένα τους, οι εντολές αποσύρονται μαζικά εγγράφοντας στον φάκελο καταχωρητών το αποτέλεσμα της τελευταίας εκτέλεσης τους.

Κομβικό σημείο αποτελεί το γεγονός ότι με την χρήση των ειδικών εντολών διακλάδωσης δεν είναι αναγκαία η χρήση εντολών για την αυξομείωση του loop index. Κάθε μια από αυτές τις εντολές φροντίζει με τρόπο που περιγράψαμε στην λειτουργικότητα της για την διαμόρφωση του loop index τον οποίο ελέγχει και κρίνει το πιθανό άλμα ή όχι. Το ίδιο συμβαίνει και με τις ειδικές εντολές μνήμης. Η χρήση τους έχει νόημα στα πλαίσια του rapid loop γιατί και πάλι μετά την αλληλεπίδραση με την μνήμη διαμορφώνουν τον δείκτη ώστε να δείχνει στην επόμενη επιθυμητή θέση μνήμης.

Αποτέλεσμα των παραπάνω είναι ότι κατά την διάρκεια ενός rapid loop δεν είναι αναγκαία η προσφορά ισχύος σε όλα τα στάδια του pipeline. Με αυτόν τον τρόπο ενεργά παραμένουν τα στοιχεία που εμπλέκονται στην επαναληπτική εκτέλεση των εντολών, δηλαδή οι αντίστοιχες μονάδες και τα υπόλοιπα στοιχεία του συστήματος δέχονται μια ελάχιστη απαιτούμενη ισχύ. Ακόμη λόγω των ιδιαίτερων εντολών διακλάδωσης δεν υπάρχει ανάγκη της πρόβλεψης διακλάδωσης εφόσον η εντολή από την φύση της γνωρίζει εκ των προτέρων εάν θα εκτελέσει άλμα ή όχι. Αυτό είναι απόρροια της λειτουργικότητας της η οποία ορίζει αρχικά την αυξομείωση του loop index και μετέπειτα τον έλεγχο διακλάδωσης, αλλά και της μορφής του rapid loop. Ο περιορισμός τίθεται στο γεγονός ότι η εντολή διακλάδωσης πρέπει να προηγείται των υπόλοιπων εντολών του βρόγχου ώστε να υπάρχουν οι απαραίτητες εξαρτήσεις που περιγράψαμε προηγουμένως. Κατά την διάρκεια λοιπόν της εκτέλεσης ενός rapid loop ισχύς παρέχεται μόνο στους αναγκαίους πόρους και ακόμη δεν είναι απαραίτητη η πρόβλεψη διακλάδωσης. Έτσι μειώνεται η κατανάλωση ενέργειας κατά την διάρκεια ενός βρόγχου και παράλληλα επιταχύνεται η εκτέλεση.

3.2.6 Οι περιορισμοί του συστήματος

Η ύπαρξη της δομής rapid loop θέτει συγκεκριμένους περιορισμούς στο σύστημα. Από την περιγραφή του τρόπου εκτέλεσης του rapid loop προκύπτει ότι ο αριθμός των εντολών θα πρέπει να είναι ίσος ή μικρότερος με τον αριθμό των διαθέσιμων μονάδων. Σε περίπτωση εκτέλεσης κώδικα υψηλού επιπέδου, ο μεταγλωττιστής θα πρέπει να καθορίσει αν κάποιος βρόγχος μπορεί να προσαρμοστεί στα πλαίσια ενός rapid loop κι αν όχι να τον τροποποιήσει. Αν ο ίδιος ο χρήστης εκτελεί κώδικα μηχανής, τότε όταν εισάγει ένα rapid loop θα πρέπει εξίσου να ακολουθήσει και τους περιορισμούς του. Ακόμη ο περιορισμός τίθεται και στο είδος των εντολών. Δηλαδή, στην συγκεκριμένη μορφή του επεξεργαστή που περιγράψαμε προηγουμένως, ένα rapid loop μπορεί να έχει μέχρι και 16 εντολές, αρκεί αυτές να μπορούν να κατανεμηθούν ανάλογα με τις απαιτήσεις τους στα 5 διαφορετικά είδη λειτουργικών μονάδων που ενσωματώνονται.

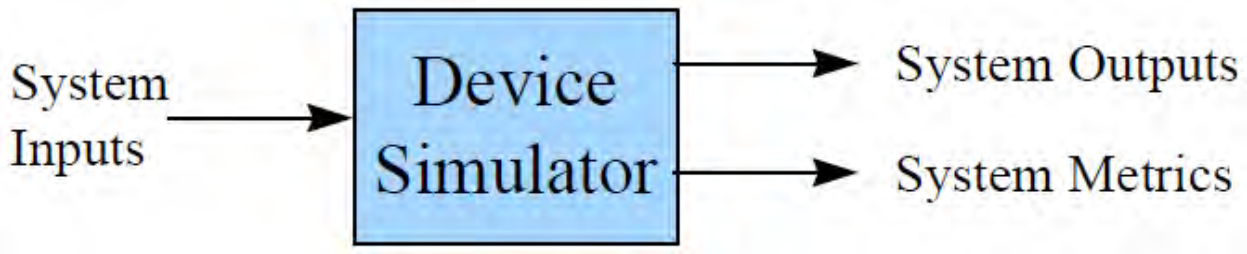
Ανεξάρτητα όμως αυτών των περιορισμών, τα οφέλη της ενσωμάτωσης της συγκεκριμένης αρχιτεκτονικής είναι ικανοποιητικά γιατί καταφέρνει να μειώσει την κατανάλωση ενέργειας αλλά και την καθυστέρηση της εκτέλεσης των εφαρμογών συνολικά. Αυτό τεκμηριώνεται και παρουσιάζεται στην συνέχεια μέσω της προσομοίωσης που κάναμε χρησιμοποιώντας τα εργαλεία Simplescalar και Wattch.

4. Ο προσομοιωτής Simplescalar

Στο παρακάτω κεφάλαιο θα κάνουμε μια περιγραφή του εργαλείου Simplescalar, μέσω του οποίου έγινε η προσομοίωση της συγκεκριμένης αρχιτεκτονικής που περιγράψαμε παραπάνω.

4.1 Ορισμοί και είδη προσομοίωσης

Με τον όρο "προσομοιωτής αρχιτεκτονικών" ορίζεται κάθε εργαλείο το οποίο αναπαράγει την συμπεριφορά ενός υπολογιστικού συστήματος. Αφού πάρει τις απαιτούμενες εισόδους, δημιουργεί τόσο τις προβλεπόμενες εξόδους αλλά και μετρήσεις σε ότι αφορά την εκτέλεση.



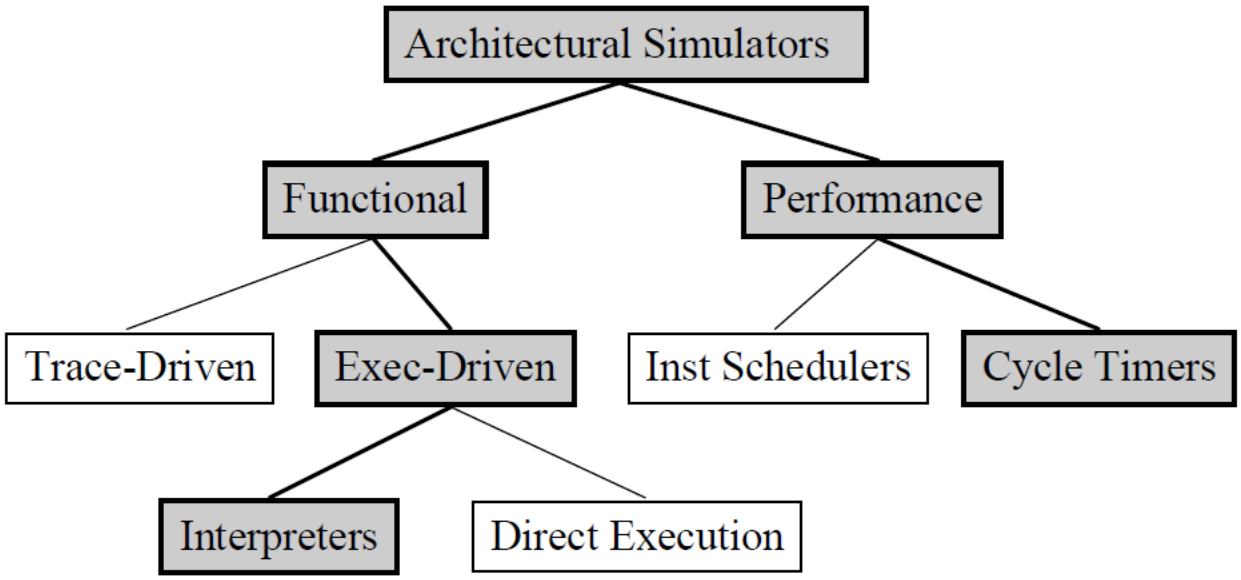
Εικόνα 5. Διαγραμματική αναπαράσταση της λειτουργίας των προσομοιωτών αρχιτεκτονικών.

Ο διαχωρισμός των προσομοιωτών αυτών τους κατατάσσει σε δυο κατηγορίες, τους λειτουργικούς(functional simulators) και τους παραστατικούς(performance simulators). Η πρώτη κατηγορία εκτελεί και αναπαριστά την αρχιτεκτονική η οποία είναι ορατή στους προγραμματιστές. Η δεύτερη κατηγορία αποτυπώνει την μικρό-αρχιτεκτονική, η οποία αποτελεί το ενδότερο μοντέλο του συστήματος, και συνήθως αυτού του είδους η προσομοίωση προσφέρει και μετρήσεις σε ό,τι αφορά τον χρόνο εκτέλεσης των εντολών.

Οι λειτουργικοί προσομοιωτές μπορούν επίσης να χωριστούν σε άλλα δυο είδη, τους "trace-driven" και τους "execution-driven". Οι "trace-driven" διαβάζουν και ακολουθούν τα ίχνη(trace) που συλλέγονται από προηγούμενη εκτέλεση των εντολών, ενώ οι

"execution-driven" εκτελούν κανονικά το πρόγραμμα, δημιουργώντας το "trace" δυναμικά. Παρότι η εφαρμογή της δεύτερης κατηγορίας των λειτουργικών προσομοιωτών είναι δυσκολότερη από την πρώτη, οι "execution-driven" προσομοιωτές μπορούν να προσφέρουν περισσότερη πληροφορία.

Οι παραστατικοί προσομοιωτές χωρίζονται και αυτοί σε δυο κατηγορίες, σε αυτούς που χρησιμοποιούν τις μονάδες "instruction-schedulers" και σε αυτούς που χρησιμοποιούν τις μονάδες "Cycle-timers". Οι πρώτοι ακολουθώντας διάταξη εντός σειράς, δρομολογούν κάθε φορά μία εντολή στον γράφο εκτέλεσης βασιζόμενοι στην διαθεσιμότητα των πόρων της μικρό-αρχιτεκτονικής. Οι δεύτεροι, παρακολουθούν την κατάσταση των πόρων της μικρό-αρχιτεκτονικής σε κάθε κύκλο μηχανής.



Εικόνα 6. Ταξινόμηση των εργαλείων προσομοίωσης.

Στην παραπάνω εικόνα βλέπουμε σχηματικά την ταξινόμηση των προσομοιωτών που παρουσιάσαμε σε μορφή δέντρου. Αναφερόμενοι πλέον στο συγκεκριμένο εργαλείο Simplescalar, τα σκιασμένα μονοπάτια του δέντρου αυτού αποτελούν την κατηγοριοποίηση του Simplescalar συνολικά.

4.2 Γενική περιγραφή του Simplescalar

Ο προσομοιωτής Simplescalar αποτελεί μια πλήρη, γρήγορη, ευέλικτη και ακριβή προσομοίωση διάφορων μικρό-αρχιτεκτονικών συστημάτων και αναπτύχθηκε στο πανεπιστήμιο του Michigan, κυρίως από τον καθηγητή Todd Austin μεταξύ του 1994 και 1996. Βρίσκει ευρεία εφαρμογή στην ερευνητική διαδικασία για την αξιολόγηση καινούριων συστημάτων που δεν έχουν ακόμα ενσωματωθεί στην παραγωγή. Ένας από τους λόγους για τους οποίους έχει γίνει γνωστός και κυρίαρχος στην ακαδημαϊκή κοινότητα είναι τα δικαιώματα χρήσης που προσφέρει. Επί της ουσίας, προωθεί και υποστηρίζει την έρευνα επιτρέποντας την ελεύθερη διανομή, αναπροσαρμογή και επέκταση του κώδικα στα όρια της μη εμπορικής χρήσης, αλλά και στα πλαίσια της αντίστοιχης εμπορικής χρήσης διανέμεται με αγορά των δικαιωμάτων μεταχείρισης.

Τα πλεονεκτήματα του Simplescalar είναι η υψηλή ευελιξία, η φορητότητα, η δυνατότητα επέκτασης καθώς και η απόδοση. Το σύνολο των εργαλείων που προσφέρει περιλαμβάνει πέντε προσομοιωτές, κάθε ένας εκ των οποίων διαφέρει με τους υπόλοιπους στην λεπτομέρεια της προσομοίωσης. Το εύρος τους κυμαίνεται μεταξύ ενός πολύ γρήγορου λειτουργικού προσομοιωτή και ενός λεπτομερή που ενσωματώνει έκδοση εντολών εκτός σειράς καθώς και υποθετική εκτέλεση. Το σετ εργαλείων έχει δυνατότητες φορητότητας, αρκεί στο σύστημα όπου θέλουμε να το χρησιμοποιήσουμε να υπάρχουν εγκατεστημένα τα εργαλεία GNU(gcc compiler, Make). Έχει ελεγχθεί σε πολλές πλατφόρμες και ακόμη μπορεί να αναδιαμορφωθεί και να επεκταθεί χρησιμοποιώντας τα βασικά στοιχεία ανάλογα με τις απαιτήσεις του κάθε χρήστη.

4.3 Το περιβάλλον του Simplescalar

Η αρχιτεκτονική του Simplescalar προέρχεται από αυτήν του MIPS-IV. Ο σχεδιασμός και η υλοποίηση του ορίζει την αρχιτεκτονική τόσο σε μορφή little-endian όσο και σε μορφή big-endian παρέχοντας

υψηλή φορητότητα(ο τύπος που επιλέγεται είναι κάθε φορά ίδιος με τον τύπο του υπολογιστή όπου φιλοξενείται ο Simplescalar). Το ISA του Simplescalar, που αναφέρεται και ως PISA(Portable ISA), είναι ένα υπερσύνολο του ISA του MIPS με κάποιες διαφορές και προσθήκες, με κυριότερη την κωδικοποίηση των εντολών με 64 bits. Να σημειώσουμε εδώ ότι ο Simplescalar προσφέρει και υλοποίηση του ISA της αρχιτεκτονικής ALPHA αλλά στα πλαίσια της παρούσας διπλωματικής ασχοληθήκαμε αποκλειστικά με την προσομοίωση χρησιμοποιώντας το Simplescalar PISA. Παρακάτω, στους πίνακες 2 και 3 παραθέτουμε αναλυτικά το σύνολο των εντολών που ενσωματώνονται με την αρχιτεκτονική PISA. Βλέπουμε πως κατά κύριο λόγο οι εντολές είναι όμοιες με αυτές του MIPS, με κάποιες αλλαγές και προσθήκες.

Πίνακας 2. Το σύνολο εντολών του Simplescalar (εντολές ελέγχου και μνήμης)

Control	**Load/Store**
j - jump	lb - load byte
jal - jump and link	lbu - load byte unsigned
jr - jump register	lh - load half (short)
jalr - jump and link register	lhu - load half (short) unsigned
beq - branch == 0	lw - load word
bne - branch != 0	dlw - load double word
blez - branch <= 0	l.s - load single-precision FP
bgtz - branch > 0	l.d - load double-precision FP
bltz - branch < 0	sb - store byte
bgez - branch >= 0	sbu - store byte unsigned
bct - branch FCC TRUE	sh - store half (short)
bcf - branch FCC FALSE	shu - store half (short) unsigned
	sw - store word
	dsw - store double word
	s.s - store single-precision FP
	s.d - store double-precision FP

Πίνακας 3. (συνέχεια) Το σύνολο εντολών του Simplescalar (εντολές αριθμών κινητής υποδιαστολής και ακεραίων, διάφορες εντολές)

Integer Arithmetic	**Floating Point Arithmetic**
add - integer add addu - integer add unsigned sub - integer subtract subu - integer subtract unsigned mult - integer multiply multu - integer multiply unsigned div - integer divide divu - integer divide unsigned and - logical AND or - logical OR xor - logical XOR nor - logical NOR sll - shift left logical srl - shift right logical sra - shift right arithmetic slt - set less than sltu - set less than unsigned	add.s - single-precision (SP) add add.d - double-precision (DP) add sub.s - SP subtract sub.d - DP subtract mult.s - SP multiply mult.d - DP multiply div.s - SP divide div.d - DP divide abs.s - SP absolute value abs.d - DP absolute value neg.s - SP negation neg.d - DP negation sqrt.s - SP square root sqrt.d - DP square root cvt - int., single, double conversion c.s - SP compare c.d - DP compareFP
Miscellaneous	
nop – no operation syscall – system call break - declare program error	

Ακόμη, στον πίνακα 4 παραθέτουμε το σύνολο των φυσικών καταχωρητών της αρχιτεκτονικής PISA του Simplescalar. Για κάθε έναν δίνουμε το όνομα του στο υλικό και στο λογισμικό(τα οποία και αναγνωρίζει ο assembler), καθώς και μια μικρή περιγραφή. Παρατηρούμε ότι το σύνολο των καταχωρητών είναι όμοιο με αυτό του MIPS-IV τόσο στο πλήθος όσο και στην σημασιολογία.

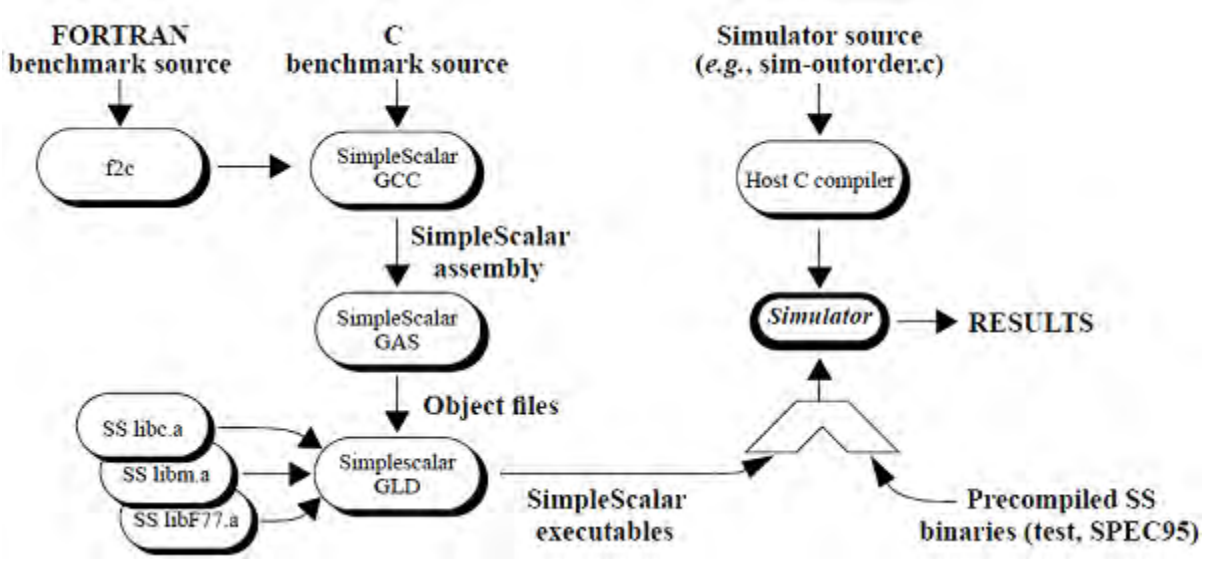
Πίνακας 4. Ορισμός του συνόλου των καταχωρητών της αρχιτεκτονικής του Simplescalar.

Hardware Name	**Software Name**	**Description**
$0	$zero	zero-valued source/sink
$1	$at	reserved by assembler
$2-$3	$v0-$v1	fn return result regs
$4-$7	$a0-$a3	fn argument value regs
$8-$15	$t0-$t7	temp regs, caller saved
$16-$23	$s0-$s7	saved regs, callee saved
$25-$25	$t8-$t9	temp regs, caller saved
$26-$27	$k0-$k1	reserved by OS
$28	$gp	global pointer
$29	$sp	stack pointer
$30	$s8	saved regs, callee saved
$31	$ra	return address reg
$hi	$hi	high result register
$lo	$lo	low result register
$f0-$f31	$f0-$f31	floating point registers
$fcc	$fcc	floating point condition code

4.3 Τρόπος λειτουργίας

Ο Simplescalar διαθέτει τα δικά του αποκλειστικά φτιαγμένα εργαλεία για μετάφραση, μεταγλώττιση και εκτέλεση του κώδικα που του δίνει ο χρήστης. Ενσωματώνει έναν μεταγλωττιστή, ο οποίος αποτελεί μια τροποποιημένη αποκλειστική έκδοση του κοινού GCC προσαρμοσμένο στα χαρακτηριστικά του προσομοιωτή(Simplescalar

GCC),καθώς επίσης και έναν assembler(Simplescalar GAS) και έναν loader(Simplescalar GLD). Ο χρήστης μπορεί να δώσει σαν είσοδο κώδικα γραμμένο είτε σε γλώσσα FORTRAN είτε σε γλώσσα C(όπως επίσης φυσικά και κώδικα assembly ακολουθώντας το ακριβές μοτίβο της αρχιτεκτονικής του Simplescalar που περιγράψαμε σε αυτό το κεφάλαιο). Σε περίπτωση που δοθεί σαν είσοδος κάποια εφαρμογή γραμμένη σε γλώσσα FORTRAN, υπάρχει ειδική δομή η οποία κάνει την μετάφραση της σε C και δίνει αυτήν την είσοδο στον gcc-compiler. Ο μεταγλωττιστής παράγει τον κώδικα assembly και με την σειρά τους ο assembler και ο loader στηριζόμενοι και στις διαθέσιμες βιβλιοθήκες του προσομοιωτή, παράγουν το εκτελέσιμο αρχείο το οποίο δίδεται στον εκάστοτε προσομοιωτή προς εκτέλεση. Οι προσομοιωτές είναι ήδη μεταγλωττισμένοι μέσω του ενσωματωμένου compiler τύπου ANSI C του υπολογιστή όπου εφαρμόζεται ο Simplescalar.



Εικόνα 7. Διάγραμμα ροής της συνολικής εκτέλεσης στον Simplescalar.

Πιο αναλυτικά, η αρχιτεκτονική του Simplescalar βρίσκεται ορισμένη στο αρχείο *machine.def*. Στο αρχείο αυτό βρίσκεται ορισμένη κάθε εντολή assembly που αναγνωρίζει ο Simplescalar καθώς και η λειτουργικότητα της στην μορφή μιας μακροεντολής. Κάθε μακροεντολή εντός του αρχείου γίνεται μια φορά *define* και μια φορά *undef* για να υπάρχει η δυνατότητα να γίνει πολλαπλές φορές *include* από τα

αρχεία των προσομοιωτών. Κατά την εκτέλεση του προσομοιωτή, η συνάρτηση *main()*(ορισμένη στο αρχείο *main.c*) κάνει όλες τις αρχικοποιήσεις και φορτώνει τα δεδομένα στην μνήμη. Έπειτα καλεί την συνάρτηση *sim_main()* η οποία είναι υλοποιημένη ξεχωριστά από κάθε προσομοιωτή ανάλογα με την λεπτομέρεια της προσομοίωσης που προφέρει. Σε αυτήν την συνάρτηση είναι ορισμένη η ραχοκοκαλιά της προσομοίωσης επιτρέποντας κάθε προσομοιωτή να δρα διαφορετικά και ανεξάρτητα με τις ορισμένες συναρτήσεις των υπολοίπων. Στην ουσία, στο αρχείο κάθε προσομοιωτή ορίζονται όλες οι απαραίτητες συναρτήσεις της εκτέλεσης και καλούνται με συγκεκριμένο τρόπο από την συνάρτηση *sim_main()*. Οι υπόλοιπες λειτουργικότητες των προσομοιωτών(υλοποίηση της μνήμης, των καταχωρητών, των εντολών assembly κτλ) είναι κοινές και εμπεριέχονται σε διάφορα αρχεία τα οποία και γίνονται include σε κάθε προσομοιωτή που είναι απαραίτητα και χρησιμοποιούνται αυτούσια.

Οι δυνατότητες που προσφέρει ο προσομοιωτής συνολικά είναι μεγάλες. Εκτός της βασικής παροχής των διαθέσιμων προσομοιωτών για εκτέλεση εφαρμογών, το περιβάλλον και ο τρόπος λειτουργίας επιτρέπει στον κάθε χρήστη να αναπτύξει τον δικό του προσομοιωτή. Ο χρήστης μπορεί να αναπτύξει την δικιά του προσομοίωση έχοντας στην διάθεση του όλες τις απαραίτητες βιβλιοθήκες και το απαιτούμενο περιβάλλον ώστε να μπορεί εύκολα να την ενσωματώσει. Με παρόμοιο τρόπο δουλέψαμε και στην δικιά μας προσομοίωση για την ανάπτυξη της νέας αρχιτεκτονικής και του επεξεργαστή που παρουσιάσαμε στο κεφάλαιο 3. Πέραν όμως της προσομοίωσης της εκτέλεσης απώτερος στόχος της εργασίας αποτελεί και η καταγραφή της καταναλισκόμενης ενέργειας κατά την εκτέλεση εφαρμογών με τον νέο επεξεργαστή. Γι’ αυτόν τον λόγο στο επόμενο κεφάλαιο θα παρουσιάσουμε και το αντίστοιχο εργαλείο που χρησιμοποιήσαμε για την καταμέτρηση της ενέργειας.

5. Το εργαλείο Wattch

5.1 Παρουσίαση του Wattch

Ένα άλλο εργαλείο πάνω στο οποίο βασίζεται η προσομοίωση που αναπτύξαμε είναι το Wattch. Το Wattch αποτελεί ένα framework για τον υπολογισμό και την ανάλυση της καταναλισκόμενης ενέργειας των επεξεργαστών σε επίπεδο αρχιτεκτονικής. Είναι μια ακόμα προσομοίωση, η οποία είναι άρρηκτα συνδεδεμένη με τον Simplescalar, εφόσον όλη η μελέτη της ενέργειας του Wattch βασίζεται στους ήδη υπάρχοντες προσομοιωτές που υλοποιούνται στον Simplescalar.

Το Wattch δεν είναι το μόνο εργαλείο που σκοπό έχει την καταμέτρηση της απαιτούμενης ισχύος για την εκτέλεση μιας εφαρμογής. Αντιθέτως, η ανάπτυξη μεθόδων για την βέλτιστη προσαρμογή των αρχιτεκτονικών με σκοπό την απαίτηση χαμηλών ποσών ενέργειας αποτέλεσε και αποτελεί πηγή έρευνας και ειδικότερα στους σχεδιαστές επεξεργαστών για ενσωματωμένα συστήματα. Όπως έχουμε αναφέρει στόχος των σχεδιαστών είναι να σχεδιάσουν αρχιτεκτονικές αποδοτικές τόσο στην εκτέλεση όσο και στον ενεργειακό τομέα, ειδικά όταν πρόκειται για συστήματα που απαιτούν χαμηλές θερμοκρασίες στο επίπεδο του επεξεργαστή λόγω ενσωμάτωσης τους σε μικρό-συσκευές. Έτσι λοιπόν εργαλεία όπως το Wattch βοηθούν στην ποσοτικοποίηση του ύψους της απαιτούμενης ισχύος για την εκτέλεση μιας εφαρμογής(benchmark) πάνω σε μια πειραματική αρχιτεκτονική. Για τον σκοπό αυτό έχουν αναπτυχθεί αρκετά εργαλεία ανάλυσης της ενέργειας, όπως για παράδειγμα το PowerMill,το οποίο βασίζεται στην κυκλωματική διάταξη της αρχιτεκτονικής ή στο επίπεδο της ανάπτυξης της με την χρήση της γλώσσας Verilog. Αυτού του είδους οι αναλύσεις παρέχουν καλή ακρίβεια προσομοίωσης αλλά μπορούν να εφαρμοστούν μόνο στο τελευταίο στάδιο του σχεδιασμού μιας αρχιτεκτονικής, όταν δηλαδή οι σχεδιαστικές λεπτομέρειες του κυκλώματος είναι πλέον γνωστές. Κάτι τέτοιο όμως είναι ασύμφορο και μη επιθυμητό. Η ύπαρξη εκτίμησης της ενεργειακής απόδοσης μιας

αρχιτεκτονικής είναι εμφανώς πιο αποδοτική προτού αυτή φτάσει στο επίπεδο σχεδιασμού του κυκλώματος. Έτσι υπάρχει η δυνατότητα της αξιολόγησης της ώστε να τροποποιηθεί και να βελτιωθεί χωρίς την σπατάλη χρόνου και δουλειάς πάνω σε μη-αποδοτικούς σχεδιασμούς. Ακριβώς αυτή είναι και η διαδικασία της έρευνας σε οποιοδήποτε κομμάτι της επιστήμης των υπολογιστών, ότι δηλαδή κάθε τεχνολογία προσομοιώνεται, αξιολογείται και βελτιώνεται σε βέλτιστο βαθμό προτού φτάσει σε πιο απτό και χαμηλό επίπεδο ανάλυσης.

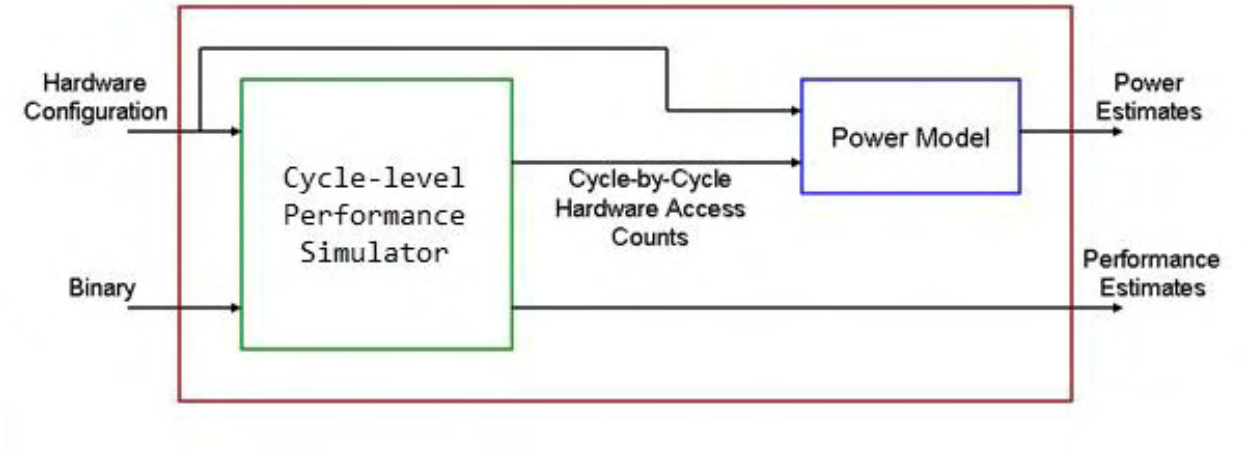
Το Wattch λοιπόν έρχεται να ικανοποιήσει τις απαιτήσεις αυτές των ερευνητικών ομάδων, προσφέροντας μια προσομοίωση βασισμένη αποκλειστικά στο επίπεδο της αρχιτεκτονικής, ακρίβεια με απόκλιση στο 10% και πολύ μεγάλη ταχύτητα(έως και 1000 φορές γρηγορότερη προσομοίωση σε σχέση με άλλα εργαλεία που δεν αναπτύσσονται στα πλαίσια της αρχιτεκτονικής του επεξεργαστή). Αναπτύχθηκε περίπου το 2000 στο πανεπιστήμιο του Princeton και βασίζεται εξ 'ολοκλήρου στον Simplescalar και συγκεκριμένα στον προσομοιωτή *sim-outorder* που υλοποιεί έκδοση εντολών με διάταξη εκτός σειράς.

5.2 Η μεθοδολογία ανάλυσης της ενέργειας

5.2.1 Γενικός τρόπος υλοποίησης

Η ανάλυση και εκτίμηση της ενέργειας βασίζεται σε παραμετροποιημένα μοντέλα υπολογισμού της ενέργειας κάποιων κοινών και ευρέως χρησιμοποιούμενων δομών υλικού στους μοντέρνους υπερβαθμωτούς επεξεργαστές. Αυτά τα μοντέλα σε συνδυασμό με την καταγραφή της χρησιμοποίησης των πόρων κατά την διάρκεια της εκτέλεσης, ενσωματώνονται στον προσομοιωτή *sim-outorder* του Simplescalar και παρέχουν εκτίμηση της καταναλισκόμενης ενέργειας. Αναφέρουμε μόνο τον προσομοιωτή *sim-outorder* γιατί όπως είπαμε το Wattch αναπτύχθηκε για αποκλειστική χρήση μέσω αυτού, αφού αποτελεί τον πλέον πολύπλοκο και ακριβή προσομοιωτή στα πλαίσια του Simplescalar. Για να μπορέσει να χρησιμοποιηθεί και από τους

λοιπούς προσομοιωτές θα πρέπει αυτοί να τροποποιηθούν ώστε να συμπεριλάβουν τις απαραίτητες μεταβλητές και συναρτήσεις στις



Εικόνα 8. Η συνολική δομή της προσομοίωσης της ενέργειας.

οποίες βασίζεται το Wattch. Κύριο μέλημα λοιπόν της προσομοίωσης είναι να καταχωρεί σε συγκεκριμένους μετρητές το πλήθος των προσβάσεων που γίνεται σε κάθε μονάδα ξεχωριστά κατά την εκτέλεση. Ο χρήστης έχοντας ενσωματώσει τα στοιχεία του Wattch, εκτελεί μια εφαρμογή χρησιμοποιώντας τον προσομοιωτή *sim-outorder* και παράλληλα με την εκτέλεση, λαμβάνει και την εκτίμηση της ισχύος που καταναλώθηκε σε επίπεδο αρχιτεκτονικής. Αυτό το συνολικό μοντέλο της προσομοίωσης της ενέργειας φαίνεται στην εικόνα 8.

Η εκτίμηση της ισχύος προκύπτει από τον υπολογισμό της δυναμικής ισχύος των τρανζίστορ του επεξεργαστή, η οποία αποτελεί το κύριο μέρος της κατανάλωσης ενέργειας στους μικρό-επεξεργαστές τύπου CMOS. Η δυναμική ισχύς δίνεται από τον τύπο $P_d=C*V_{dd}^2*a*f$, όπου το "C" αποτελεί την χωρητικότητα, το "V_{dd}" την τάση τροφοδοσίας και το "f" την συχνότητα του ρολογιού. Το "a" ονομάζεται activity factor, σχετίζεται με τα προγράμματα ελέγχου που εκτελούνται και παίρνει τιμές μεταξύ του μηδενός(0) και της μονάδας(1). Οι υπολογισμοί του Wattch βασίζονται στην εκτίμηση της χωρητικότητας "C" των πόρων του υλικού βάσει του κυκλώματος και των διαστάσεων των τρανζίστορ. Η τάση τροφοδοσίας και η συχνότητα του ρολογιού

βασίζονται στην υποθετική τεχνολογία του επεξεργαστή που υλοποιεί η προσομοίωση. Το Wattch συγκεκριμένα χρησιμοποιεί τις παραμέτρους της τεχνολογίας διαστάσεων 0.35μm.

5.2.2 Τα παραμετροποιημένα μοντέλα ενέργειας

Οι βασικές μονάδες του επεξεργαστή χωρίζονται σε τέσσερις βασικές κατηγορίες:

- Δομές πίνακα(array structures) που περιλαμβάνουν μνήμες εντολών και δεδομένων, καταχωρητές, πρόβλεψη διακλαδώσεων, μεγάλο τμήμα του παραθύρου εντολών και την ουρά των εντολών τύπου load/store.
- Μνήμες πλήρους συσχέτισης διευθυνσιοδοτημένες βάσει του περιεχομένου(Fully Associative Content-Addressable Memories-CAM) που περιλαμβάνουν το wakeup logic παραθύρου εντολών(Instruction window wakeup logic) καθώς και τον πίνακα σελίδων TLB.
- Καλώδια και συνδυαστική λογική(Combinational logic and Wires) που περιλαμβάνουν τις λειτουργικές μονάδες του συστήματος, το selection logic του παραθύρου εντολών(Instruction window selection logic), τον έλεγχο εξαρτήσεων των εντολών (dependency check logic) και τα result buses.
- Μονάδες Clocking που περιλαμβάνουν ό,τι έχει σχέση με το ρολόι του επεξεργαστή και την ενέργεια που καταναλώνει.

Με αυτήν την κατηγοριοποίηση, το Wattch ορίζει την χωρητικότητα του κάθε μοντέλου και βασιζόμενο στον αριθμό των προσβάσεων σε κάθε στοιχείο του υλικού κατά την εκτέλεση εφαρμογών, υπολογίζει την προβλεπόμενη ισχύ. Είναι σημαντικό να αναφέρουμε ότι η μοντελοποίηση της ισχύος των λειτουργικών μονάδων βασίζεται σε μετρήσεις που έχουν γίνει σε προηγούμενες έρευνες στο παρελθόν. Αυτές οι τιμές χρησιμοποιούνται σαν την βασική τιμή της ισχύος κάθε μονάδας και κλιμακώνονται ανάλογα με την χρήση της εκάστοτε μονάδας. Ακόμη, το Wattch ενσωματώνει τρείς διαφορετικούς τρόπους

υπολογισμού τις ενέργειας βασιζόμενο σε τρία διαφορετικά είδη του conditional clocking. Το conditional clocking(cc) αποτελεί τον τρόπο με τον οποίο τα μέρη του υλικού καταναλώνουν ενέργεια. Αυτά τα είδη είναι:

- Το απλό (cc1): η ισχύς που καταναλώνεται ισοδυναμεί με την μοντελοποιημένη ισχύ του υλικού στοιχείου μόνο όταν αυτό χρησιμοποιείται, ειδάλλως είναι μηδενική.
- Το ιδανικό (cc2): λαμβάνει υπόψιν τις θύρες εισόδου της εκάστοτε μονάδας και κλιμακώνει την καταναλισκόμενη ισχύ ανάλογα με τις θύρες που χρησιμοποιούνται και όταν η μονάδα δεν χρησιμοποιείται η ισχύς που τις παρέχεται είναι μηδενική.
- Το μη-ιδανικό (cc3): υπολογίζει την ισχύ όμοια με το δεύτερο είδος, παρέχοντας όμως 10% της μέγιστης ισχύος σε κάθε μονάδα σε περίπτωση που αυτή δεν χρησιμοποιείται.

Όλα τα παραπάνω υπολογίζονται σε κάθε κύκλο μηχανής και αθροίζονται για την συνολική εκτέλεση των εφαρμογών. Όπως έχουμε αναφέρει, το Wattch προσφέρει μια γρήγορη προσομοίωση κατανάλωσης ισχύος με απόκλιση 10%. Η ομάδα που ανέπτυξε τον προσομοιωτή επαλήθευσε τα αποτελέσματα της συγκρίνοντας τα με μετρήσεις που έγιναν πάνω σε κανονικούς εργοστασιακούς επεξεργαστές, σε συνεργασία με την Intel και κατέληξε στο συγκεκριμένο ποσοστό λάθους.

Γνωρίζοντας πλέον αναλυτικά το περιβάλλον της προσομοίωσης, στο επόμενο κεφάλαιο θα περιγράψουμε αναλυτικά την προσομοίωση που κάναμε για την αξιολόγηση της νέας αρχιτεκτονικής που περιγράψαμε στο κεφάλαιο 3.

6. Προσομοίωση της νέας αρχιτεκτονικής

6.1 Ανάλυση της προσομοίωσης στον Simplescalar

6.1.1 Περιγραφή και προσαρμογή του περιβάλλοντος

Μετά την εγκατάσταση του Simplescalar που περιγράφεται στο Παράρτημα, στον φάκελο *simplesim-3.0* βρίσκονται τα αρχεία των πέντε διαφορετικών προσομοιωτών(*sim-fast*, *sim-safe*, *sim-cache*, *sim-profile* και *sim-outorder*) καθώς και όλα τα απαραίτητα αρχεία που περιλαμβάνουν μεταβλητές και συναρτήσεις αναγκαίες για την εκάστοτε υλοποίηση. Εντός του φακέλου αυτού υπάρχει και το αρχείο Makefile που κατευθύνει τον compiler στην μεταγλώττιση των προσομοιωτών αλλά και στην διασύνδεση των παραγόμενων αρχείων object μεταξύ τους με τρόπο που εμείς ορίζουμε ξεχωριστά για κάθε προσομοιωτή. Για να ενσωματώσουμε την δικιά μας καινούρια προσομοίωση δημιουργήσαμε ένα καινούριο αρχείο *sim-rapidloop.c* και τροποποιήσαμε το αρχείο Makefile ώστε να αναγνωρίζει και να δημιουργεί το εκτελέσιμο του καινούριου προσομοιωτή. Μέσα λοιπόν στο αρχείο *sim-rapidloop.c* αναπτύξαμε τον βασικό κώδικα για την προσομοίωση της νέας αρχιτεκτονικής χρησιμοποιώντας και τροποποιώντας κατάλληλα τα υπόλοιπα απαραίτητα αρχεία, στα οποία θα αναφερθούμε μετέπειτα.

6.1.2 Βασικά στοιχεία της προσομοίωσης

Ο πιο πολύπλοκος και λεπτομερής προσομοιωτής που περιλαμβάνει ο Simplescalar είναι ο sim-outorder. Ανήκει στην κατηγορία των παραστατικών προσομοιώσεων(performance simulator), αποτυπώνοντας πλήρως μια αρχιτεκτονική που ακολουθεί εκτέλεση εντολών με έκδοση εκτός σειράς μέσω ενός pipeline εντολών και παρουσιάζει μετρήσεις σχετικά με τον χρόνο εκτέλεσης των εφαρμογών. Το pipeline του ακολουθεί τα πέντε βασικά στάδια μιας αρχιτεκτονικής με έκδοση εκτός σειράς, με το καθένα από αυτά να υλοποιείται σε μια

διαφορετική συνάρτηση εντός του αρχείου *sim-outorder.c*. Η προσομοίωση αυτή όμως μπορεί να κρατάει μετρήσεις για έκδοση εντολών εκτός σειράς, αλλά η εκτέλεση τους γίνεται εντός σειράς. Κάθε εντολή που φτάνει στο σύστημα εκτελείται και απλά κρατούνται συγκεκριμένες μετρήσεις για την υποθετική εκτέλεση της εκτός σειράς.

Βασιζόμενοι λοιπόν σε αυτόν τον προσομοιωτή που φαίνεται να έχει κάποιες ομοιότητες στον τρόπο εκτέλεσης, αναπτύξαμε την δικιά μας προσομοίωση στο αρχείο *sim-rapidloop.c*. Βασική διαφορά του δικού μας προσομοιωτή με τον sim-outorder, είναι ότι η υλοποίηση μας εκτός των ιδιαίτερων χαρακτηριστικών που περιλαμβάνει(rapid loop), υλοποιεί πραγματική εκτέλεση εκτός σειράς και προώθηση δεδομένων μεταξύ των μονάδων, ενσωματώνοντας έξι στάδια στο pipeline των εντολών.

Για να μπορέσει να υλοποιηθεί η προώθηση δεδομένων μεταξύ των εντολών έπρεπε να αλλάξουμε τον τρόπο με τον οποίο ορίζονται και υλοποιούνται οι εντολές του Simplescalar. Όπως αναφέραμε κατά την περιγραφή του Simplescalar, ο ορισμός και ο τρόπος εκτέλεσης κάθε εντολής του PISA βρίσκεται στο αρχείο machine.def. Η μέχρι τώρα λειτουργικότητα κάθε εντολής περιλάμβανε την ανάγνωση του φακέλου καταχωρητών για την απόκτηση των τελούμενων. Στην δική μας προσομοίωση, στην λειτουργικότητα της κάθε εντολής υπάρχει δυνατότητα για απόκτηση των τελούμενων από άλλες εντολές από τις οποίες η εντολή είναι εξαρτημένη και οι οποίες έχουν εκτελεστεί αλλά δεν έχουν γράψει ακόμα την έξοδο τους στον φάκελο καταχωρητών. Ακόμα, για κάθε εντολή έχει αντικατασταθεί το σκέλος του κώδικα που αφορά την καταγραφή της εξόδου της στον ΦΚ. Πλέον οι εντολές διατηρούν το αποτέλεσμα τους για να μπορούν να το προωθήσουν και να το γράψουν στον ΦΚ, όταν αυτό είναι επιθυμητό.

Όσον αφορά την εκτέλεση εκτός σειράς, η προσομοίωση περιλαμβάνει μετονομασία των καταχωρητών, κάτι το οποίο είναι βασικό για να μπορέσει να γίνει αυτού του είδους η εκτέλεση. Παρόλα αυτά η λειτουργικότητα των εντολών περιλάμβανε ανάγνωση του φακέλου των

καταχωρητών χωρίς να ληφθεί υπόψιν η μετονομασία τους. Στην ουσία η μετονομασία γίνεται για λόγους καταγραφής μετρήσεων. Έτσι για να προσαρμόσουμε την λογική αυτή και να καταφέρουμε την υλοποίηση εκτέλεσης εκτός σειράς αναπτύξαμε για κάθε εντολή τον τρόπο με τον οποίο αυτή αποσύρεται και γράφει στον φάκελο καταχωρητών. Έτσι λοιπόν η προσομοίωση συνοδεύεται και από ένα ακόμα πρόσθετο αρχείο(*commit.def*) που είναι της ίδιας λογικής με το machine.def υλοποιώντας μια μακρό-εντολή για κάθε εντολή του PISA, την οποία ο προσομοιωτής θα καλεί για να γράψει την έξοδο της εντολής στον ΦΚ. Συνολικά, η διαδικασία που ακολουθείται είναι ότι όταν η εντολή εκτελείται λαμβάνει τα τελούμενα της είτε από τον ΦΚ(χωρίς να λάβει υπόψιν την μετονομασία) είτε από προώθηση, και διατηρεί την έξοδο της για να την προωθήσει και να την γράψει στον ΦΚ όταν είναι η σειρά της(η εγγραφή στον ΦΚ πρέπει γίνεται μόνο με διάταξη εντός σειράς).

Αναφερόμενοι στο machine.def και στον ορισμό του τρόπου εκτέλεσης των εντολών, να επισημάνουμε σε αυτό το σημείο ότι για την ενσωμάτωση των νέων εντολών της αρχιτεκτονικής προσθέσαμε επιπλέον μακροεντολές στο αρχείο αυτό. Μάλιστα, πλέον με το αλλαγμένο αρχείο machine.def οι υπόλοιποι προσομοιωτές δεν μπορούν να μεταγλωττιστούν γιατί δεν περιλαμβάνουν τον ορισμό των μεταβλητών που έχει η δικιά μας προσομοίωση και χρησιμοποιούνται σε όλες τις μακροεντολές του machine.def. Για να ολοκληρωθεί όμως η ενσωμάτωση εντολών είναι απαραίτητος ο καθορισμός του συγκεκριμένου opcode που θα έχουν. Έτσι λοιπόν, επεκτείνοντας το αρχείο *ss-opc.c* που περιλαμβάνει τους ορισμούς των opcodes όλων των εντολών και βρίσκεται στον φάκελο *binutils-2.5.2/opcodes*, προσθέσαμε και τα opcodes των καινούριων εντολών για να είναι ικανός ο assembler να τις αναγνωρίσει.

Είναι πολύ σημαντικό να αναφέρουμε επίσης, ότι η αρχιτεκτονική που προσομοιώσαμε και που περιγράψαμε στο κεφάλαιο 3, έχει σαν κύριο άξονα τις λειτουργικές μονάδες και την συγκεκριμένη τοπολογία τους. Η λογική όμως του Simplescalar έχει σαν κεντρικό άξονα τα στάδια του pipeline που διανύουν οι εντολές. Οι λειτουργικές

μονάδες ορίζονται σαν ένας μονοδιάστατος πίνακας, με κάθε στοιχείο του πίνακα αυτού να αποτελεί και μια ξεχωριστή μονάδα η οποία εξυπηρετεί διαφορετικές λειτουργικότητες. Εμείς, ακολουθώντας την περιγραφή που παραθέσαμε στο κεφάλαιο 3, ορίσαμε τις συγκεκριμένες μονάδες και τις λειτουργικότητες τους με τον τρόπο όμως που ήδη γινόταν στον Simplescalar. Στα πλαίσια της προσομοίωσης η τοπολογία τους δεν είναι "ορατή" και η προώθηση των δεδομένων γίνεται από εντολή σε εντολή, θεωρώντας πως έτσι αντιπροσωπεύεται και η εκάστοτε λειτουργική μονάδα.

Πίνακας 5. Οι λειτουργικές μονάδες της προσομοίωσης

Λειτουργική Μονάδα	***Πλήθος***	***Είδος πράξης/Καθυστέρηση***
Integer-ALU(A)	3	IntALU / 1 FloatCVT / 2
FP-MULT/DIV(FM)	3	IntMULT / 3 IntDIV / 20 FloatMULT / 4 FloatDIV / 12 FloatSQRT / 24
FP-ADDER(FA)	4	FloatADD / 2 FloatCMP / 2 FloatCVT / 2 IntALU / 1
MEMORY PORT(M)	4	RdPort / 1 WrPort / 1 MemRapid / 2
RAPID BRANCH(B)	2	Branch / 1

Πίνακας 6. Επεξήγηση των πράξεων που ενσωματώνει η προσομοίωση

Είδος Πράξης	***Επεξήγηση***
IntALU	πρόσθεση/αφαίρεση ακεραίων και λογικές πράξεις
IntMULT	πολλαπλασιασμός ακεραίων
IntDIV	διαίρεση ακεραίων
FloatADD	πρόσθεση αριθμών κινητής υποδιαστολής
FloatCMP	σύγκριση αριθμών κινητής υποδιαστολής
FloatMULT	μετατροπή αριθμών κινητής υποδιαστολής σε ακεραίους και αντίστροφα
FloatDIV	διαίρεση αριθμών κινητής υποδιαστολής
FloatSQRT	τετραγωνική ρίζα αριθμών κινητής υποδιαστολής
RdPort	ανάγνωση μνήμης δεδομένων
WrPort	εγγραφή στην μνήμη δεδομένων
MemRapid	ειδικές εντολές προσπέλασης μνήμης
Branch	ειδικές εντολές ελέγχου

Στον πίνακα 5 της προηγούμενης σελίδας παρουσιάζονται αναλυτικά οι λειτουργικές μονάδες, το πλήθος κάθε μονάδας καθώς και οι λειτουργίες που μπορούν να εκτελέσουν συνοδευόμενες από την καθυστέρηση τους σε κύκλους μηχανής. Ακόμη, στον πίνακα 6 εξηγούμε τα είδη των πράξεων τα οποία ενσωματώθηκαν στην προσομοίωση και χρησιμοποιήθηκαν στον πίνακα 5.

Ένα άλλο εξίσου βασικό χαρακτηριστικό της προσομοίωσης μας είναι ότι κάθε εντολή διαθέτει η ίδια μια ουρά για κάθε καταχωρητή εισόδου. Με αυτόν τον τρόπο μπορεί να υλοποιηθεί η λογική της επιτάχυνσης βρόγχων(rapid loop) που περιγράψαμε αναλυτικά στο κεφάλαιο 3 και θα περιγράψουμε την προσομοίωση της παρακάτω. Θα ήταν σημαντικό εδώ να αναφέρουμε ότι η υλοποίηση ακολουθεί πιστά τους περιορισμούς που αναφέραμε ότι θέτει η νέα αρχιτεκτονική σε ό,τι αφορά την υλοποίηση των rapid loops.

Προσαρμόσαμε λοιπόν την λογική της νέας αρχιτεκτονικής σε αυτή του Simplescalar και αποτυπώσαμε όσο το δυνατόν καλύτερα τα ιδιαίτερα χαρακτηριστικά της. Αλλά αυτό φαίνεται καλύτερα στην επόμενη ενότητα, όπου και ακολουθεί ακόμα πιο λεπτομερής περιγραφή της προσομοίωσης που φτιάξαμε.

6.1.3 Η εκτέλεση των εντολών στο pipeline

Κατά την περιγραφή της καινούριας αρχιτεκτονικής στο κεφάλαιο 3 αναφέραμε πως η ιδέα περιλαμβάνει την υλοποίηση τεσσάρων pipelines που μοιράζονται ένα κοινό δίκτυο πόρων και ότι το εύρος του κάθε pipeline μπορεί να οριστεί από ένα έως τέσσερα. Ο στόχος της παρούσας διπλωματικής ήταν η προσομοίωση να είναι αντιπροσωπευτική της αρχιτεκτονικής αυτής αλλά και παράλληλα ικανή να εκμεταλλευτεί τις λειτουργικότητες του Wattch έτσι ώστε να έχουμε κάποια πρώτα αποτελέσματα σε ότι αφορά την κατανάλωση ενέργειας. Λόγω λοιπόν των ορίων επέκτασης των προσομοιώσεων που θέτουν ο Simplescalar και το Wattch, αναπτύξαμε την προσομοίωση της αρχιτεκτονικής περιλαμβάνοντας ένα pipeline για εκτέλεση εντολών. Το pipeline αυτό έχει εύρος που ισούται με 4 και εκμεταλλεύεται τους συνολικούς πόρους του συστήματος. Παρακάτω περιγράφουμε αναλυτικά κάθε στάδιο του pipeline αυτού.

Στάδιο FETCH: Αποτελεί την αρχή της εκτέλεσης κάθε εντολής. Κατά το στάδιο αυτό γίνεται η προσκόμιση των εντολών και η εισαγωγή τους σε μια ουρά(fetch queue) η οποία τροφοδοτεί το επόμενο στάδιο, αυτό της αποκωδικοποίησης. Στα πλαίσια της διπλωματικής εργασίας δεν υλοποιήσαμε πρόβλεψη διακλάδωσης αλλά έναν διαφορετικό τρόπο συγχρονισμού που θα περιγράψουμε στο επόμενο στάδιο. Τέλος, εφόσον το pipeline έχει εύρος τέσσερα, σε κάθε στάδιο FETCH προσκομίζονται τέσσερις νέες εντολές, εάν αυτό είναι δυνατό(λόγω πληρότητας των ουρών προσκόμισης αποκωδικοποίησης το πλήθος των εντολών που

προσκομίζονται σε διαφορετικούς κύκλους μηχανής μπορεί να είναι μικρότερο του τέσσερα).
Το στάδιο αυτό υλοποιείται στην συνάρτηση **ruu_fetch.**

Στάδιο DECODE: Το στάδιο αυτό τροφοδοτείται από το προηγούμενο στάδιο FETCH και αποκωδικοποιεί κάθε εντολή η οποία βρίσκεται στην ουρά προσκόμισης. Κάθε εντολή που αποκωδικοποιείται καταλαμβάνει μια θέση στην λίστα εντολών του συστήματος, που υλοποιείται από μια κυκλική λίστα. Ο συγχρονισμός που ενσωματώνουμε λόγω της μη υλοποίησης πρόβλεψης διακλάδωσης και της εκτέλεσης εκτός σειράς, υποχρεώνει κάθε εντολή ελέγχου και SYSCALL να μην αποκωδικοποιηθεί και να περιμένει στην ουρά προσκόμισης, μέχρις ότου όλες οι προηγούμενες εντολές της έχουν εκτελεστεί. Έτσι εξασφαλίζουμε ότι κάθε φορά που κάποια εντολή ελέγχου εκτελείται δεν υπάρχει προηγούμενη εντολή που δεν έχει εκτελεστεί ακόμα επειδή προσπεράστηκε και πιθανόν να ακυρωθεί αν η εντολή ελέγχου εκτελέσει άλμα. Ο ίδιος συγχρονισμός εφαρμόζεται και στις νέες ειδικές εντολές ελέγχου για την υλοποίηση του rapid loop. Σε αυτές τις εντολές, ο συγχρονισμός αυτός είναι απαραίτητος και για την οργάνωση του rapid loop ώστε να είναι γνωστό ποιες είναι οι εντολές που περιλαμβάνονται σε αυτό.
Όταν τελικώς μια ειδική εντολή ελέγχου αποκωδικοποιείται, ενεργοποιείται ένα κοινό flag το οποίο προσδιορίζει την πιθανή έναρξη ενός rapid loop. Σε αυτήν την περίπτωση δεν αποκωδικοποιούνται επόμενες εντολές μέχρις ότου οριστικοποιηθεί αν υπάρξει rapid loop ή όχι για να είναι γνωστός ο τρόπος χειρισμού των εντολών.
Αν η εντολή είναι εντολή μνήμης, τότε μεταφράζεται σε δύο εντολές, σε μια εντολή υπολογισμού της διεύθυνσης προσπέλασης και στην καθ' εαυτού εντολή μνήμης. Η δεύτερη εισέρχεται στην ουρά LSQ ενώ η πρώτη στην ουρά RUU. Και οι δύο είναι κυκλικές λίστες με όμοια χαρακτηριστικά και δυνατότητες. Να σημειώσουμε εδώ ότι οι εντολές μνήμης εισέρχονται σε διαφορετική λίστα αναμένοντας για την διεύθυνση προσπέλασης για να εκτελεστούν. Σε περίπτωση rapid loop θεωρούμε ότι υπάρχουν μόνο οι ειδικές εντολές μνήμης για να υπάρχει επαναληπτικά και η προσπέλαση διαφορετικών στοιχείων κάθε φορά.

Αυτές οι ειδικές εντολές μνήμης δεν απαιτούν την εκτέλεση διαφορετικής εντολής για τον υπολογισμό διεύθυνσης προσπέλασης αφού το κάνουν μόνες τους κι έτσι δεν εισέρχονται στην ουρά LSQ αλλά στην ουρά RUU. Έτσι σε κάθε rapid loop υπάρχουν εντολές μόνο στην ουρά RUU.

Μετά την αποκωδικοποίηση, η θέση που καταλαμβάνει η εντολή είτε στην λίστα RUU είτε στην λίστα LSQ περιλαμβάνει πεδία για τις εξής πληροφορίες:

- το opcode της εντολής
- την τιμή του μετρητή προγράμματος(PC) της επόμενης εντολής
- flag που δείχνει αν η εντολή είναι στην λίστα LSQ
- την διεύθυνση προσπέρασης μνήμης για τις αντίστοιχες εντολές μνήμης
- την τιμή του καταχωρητή βάσης(base register) για τις εντολές μνήμης
- την έξοδο της εντολής που θα γραφτεί στον φάκελο καταχωρητών
- flag που δείχνει αν η εντολή είναι εξαρτημένη από κάποια άλλη
- τα προωθούμενα δεδομένα προς την εντολή σε περίπτωση που η εντολή είναι εξαρτημένη
- τους καταχωρητές εισόδου και εξόδου
- flag που δείχνει αν η εντολή είναι εντός κάποιου rapid loop
- τον αριθμό των επαναλήψεων που θα εκτελέσει η εντολή εάν είναι εντός κάποιου rapid loop
- λίστα εξαρτημένων εντολών από την εντολή
- λίστα εξαρτημένων εντολών από προηγούμενη επανάληψη για προς τα πίσω προώθηση της εξόδου(σε περίπτωση rapid loop)
- flags που δείχνουν αν η εντολή έχει εκδοθεί, δρομολογηθεί ή βρίσκεται σε διαδικασία εκτέλεσης σε κάποια μονάδα
- flags για διαθέσιμες εισόδους από τον φάκελο καταχωρητών

Τα πεδία αυτά είτε παίρνουν την κατάλληλη τιμή ήδη στο στάδιο αυτό είτε αρχικοποιούνται με ουδέτερες τιμές και τροποποιούνται κατά την διαπέραση από τα υπόλοιπα στάδια του pipeline.
Το στάδιο αυτό υλοποιείται στην συνάρτηση **ruu_decode**.

Στάδιο ISSUE: Κατά το στάδιο αυτό, επιχειρείται η έκδοση κάθε εντολής που βρίσκεται σε μια από τις δυο λίστες εντολών(LSQ, RUU) και δεν είναι μαρκαρισμένη ως issued. Ο κανόνας που ακολουθείται είναι ότι η έκδοση των εντολών γίνεται εφόσον υπάρχει διαθέσιμη λειτουργική μονάδα που μπορεί να εξυπηρετήσει την εκάστοτε εντολή. Τότε η εντολή μαρκάρεται ως issued και η λειτουργική μονάδα ως κατειλημμένη.
Το στάδιο αυτό υλοποιείται στην συνάρτηση **ruu_issue.**

Στάδιο SCHEDULE: Στο στάδιο αυτό γίνεται η τελική δρομολόγηση των εντολών προς εκτέλεση. Δρομολογούνται μόνο εντολές που έχουν εκδοθεί σε κάποια λειτουργική μονάδα και που έχουν διαθέσιμα όλα τους τα δεδομένα για εκτέλεση. Να αναφέρουμε ότι η αρχιτεκτονική υποστηρίζει και προώθηση δεδομένων και έτσι οι εντολές που έχουν εξάρτηση από κάποια άλλη μπορεί να πάρουν το απαραίτητο στοιχείο μέσω προώθησης και όχι μέσω του φακέλου καταχωρητών.
Για τις εντολές που βρίσκονται στην λίστα RUU, η δρομολόγηση προς εκτέλεση γίνεται σύμφωνα με το μοντέλο που αναφέραμε μόλις προηγουμένως και τότε οι εντολές μαρκάρονται ως scheduled. Οι εντολές που είναι τύπου rapid loop δεν χρειάζεται να έχουν όλες τις εισόδους διαθέσιμες για να δρομολογηθούν και σε τέτοια περίπτωση κάθε μια από αυτές δρομολογείται κατ' ευθείαν και αποστέλλεται για εκτέλεση μαζί με τα διαθέσιμα τελούμενα της και τις εξαρτήσεις τις.
Για τις εντολές που βρίσκονται στην ουρά LSQ εφαρμόζουμε ένα είδος συγχρονισμού για να αποτρέψουμε πιθανές συγκρούσεις μνήμης(memory confliction) μεταξύ των εντολών μνήμης. Μέσω αυτού, εντολές τύπου load που έπονται κάποιας εντολής τύπου store που δεν έχει εκτελεστεί ακόμα και έχουν κοινή διεύθυνση προσπέλασης, δεν δρομολογούνται προς εκτέλεση ακόμα κι αν πληρούν τις προϋποθέσεις που αναφέραμε παραπάνω. Έτσι αποτρέπουμε το πιθανό λογικό λάθος φόρτωσης εσφαλμένου δεδομένου μέσω των εντολών load.
Οι εντολές που δρομολογούνται προς εκτέλεση εισέρχονται σε μια λίστα(ready queue) η οποία τροφοδοτεί το επόμενο στάδιο της εκτέλεσης. Εδώ να σημειώσουμε ότι έχουμε και την υλοποίηση του loop buffer που αναφέραμε στην περιγραφή της αρχιτεκτονικής. Έτσι

κατά την δρομολόγηση εντολών που βρίσκονται εντός κάποιου rapid loop τοποθετούμε τις εντολές σε διαφορετική λίστα, δηλαδή στον loop buffer. Ακόμη, κατά την δρομολόγηση της τελευταίας εντολής του rapid loop, διαβάζονται όλα τα διαθέσιμα δεδομένα των εντολών που βρίσκονται στον loop buffer, σηματοδοτείται η έναρξη ενός rapid loop ενεργοποιώντας ένα κοινό flag και απενεργοποιείται κάθε στάδιο του pipeline εκτός του dispatch(που θα περιγράψουμε αμέσως τώρα).
Το στάδιο αυτό υλοποιείται στην συνάρτηση **ruu_schedule.**

Στάδιο DISPATCH: Το στάδιο αυτό προσομοιώνει την εκτέλεση των εντολών στις λειτουργικές μονάδες. Η εκάστοτε εντολή εκτελείται από την μονάδα για όσους κύκλους απαιτείται από τον ορισμό των διαφορετικών πράξεων που περιγράψαμε στην προηγούμενη ενότητα. Καθ' όλη αυτήν την διάρκεια η λειτουργική μονάδα θεωρείται κατειλημμένη. Με το πέρας της εκτέλεσης η εντολή κρατά την έξοδο της και την προωθεί σε όλες τις εντολές που περιμένουν το δεδομένο για να δρομολογηθούν προς εκτέλεση. Εδώ να σημειώσουμε πως θεωρούμε ότι υλοποιείται η ιδιαίτερη τοπολογία του δικτύου των λειτουργικών μονάδων που περιγράψαμε στο κεφάλαιο 3 και τα δεδομένα προωθούνται άμεσα μόνο προς τις εξαρτημένες εντολές.
Στην περίπτωση της άφιξης μιας ειδικής εντολής ελέγχου για πρώτη φορά, έχουμε την εκτέλεση της και την σηματοδότηση της πιθανής έναρξης ενός rapid loop. Έπειτα, και σε κάθε εκτέλεση της η εντολή εκτός από την προώθηση δεδομένων στις εξαρτημένες εντολές, προωθεί τον επαυξημένο loop index και στον εαυτό της.
Κατά την εκτέλεση του rapid loop οι εντολές εκτελούνται κι έπειτα προωθούν την έξοδο τους στις εξαρτημένες εντολές αλλά και σε εντολές που έχουν έμμεση εξάρτηση σε επόμενη επανάληψη(οι εντολές λέγεται ότι έχουν εξάρτηση από δεδομένα προηγούμενης επανάληψης). Αυτή η προς τα πίσω προώθηση φυσικά δεν συμβαίνει εκτός του rapid loop γιατί τότε για κάθε ξεχωριστή επανάληψη οι εντολές περνούν από όλα τα στάδια του pipeline. Εντός του rapid loop χρησιμοποιούνται άμεσα και οι ουρές στους καταχωρητές εισόδου. Η λογική είναι ότι κάθε φορά που κάποια εντολή προωθεί την έξοδο της σε κάποια άλλη, αυτή η έξοδος μπαίνει στο τέλος της ουράς. Όποτε

η εντολή έχει τα απαιτούμενα τελούμενα της διαθέσιμα, εκτελείται και καταναλώνει ένα στοιχείο από κάθε ουρά.
Έτσι, βασισμένοι και στις ψευδο-εξαρτήσεις που αναφέραμε στην περιγραφή της αρχιτεκτονικής, κάθε εντολή εκτελείται τόσες φορές όσες και η ειδική εντολή ελέγχου στην αρχή του rapid loop.
Το στάδιο αυτό υλοποιείται στην συνάρτηση **ruu_dispatch**.

Στάδιο COMMIT: Κατά το στάδιο αυτό, γίνεται η απόσυρση των εντολών και η εγγραφή του φακέλου καταχωρητών, αν αυτό είναι απαραίτητο. Κάθε εντολή που φτάνει σε αυτό το σημείο έχει περάσει από το στάδιο εκτέλεσης, πιθανόν με διάταξη εκτός σειράς. Όμως στο στάδιο αυτό υπάρχει συγχρονισμός που επιβάλλει την απόσυρση των εντολών σε διάταξη εντός σειράς μόνο. Αφού η εντολή γράψει την έξοδο της στον ΦΚ, απελευθερώνει την θέση που δεσμεύει στην λίστα εντολών(RUU-LSQ), πράγμα που σηματοδοτεί την απόσυρση της.
Το στάδιο αυτό υλοποιείται στην συνάρτηση **ruu_commit**.

Η κυριότερη συνάρτηση της υλοποίησης είναι η sim_main η οποία καλείται με τον τρόπο που περιγράψαμε στην ενότητα 4.3 και διαμορφώνει και ορίζει την σειρά με την οποία καλούνται οι παραπάνω συναρτήσεις. Ο κύριος βρόγχος της προσομοίωσης υπάρχει εντός αυτής της συνάρτησης και δομείται όπως φαίνεται στο πλαίσιο αριστερά. Ο βρόγχος αυτός εκτελείται μια φορά για κάθε κύκλο μηχανής που προσομοιώνουμε, γιατί κάθε επανάληψη του ισοδυναμεί με μια διαπέραση από το pipeline. Ο τρόπος που καλούμε τις αντίστοιχες συναρτήσεις-στάδια είναι αντίστροφος διότι έτσι επιτυγχάνουμε τον κατάλληλο συγχρονισμό, με κάθε εντολή να μπορεί να διαπερνά από ένα μόνο στάδιο σε κάθε κύκλο μηχανής.

```
ruu_init ();
for (;;) {
        ruu_commit();
        ruu_dispatch();
        ruu_schedule();
        ruu_issue();
        ruu_decode();
        ruu_fetch();
}
```

6.2 Ενσωμάτωση του Wattch

Όπως αναφέραμε και στο κεφάλαιο 5 που παρουσιάσαμε και αναλύσαμε το εργαλείο Wattch, όλοι οι υπολογισμοί του και γενικότερα η υλοποίηση του στηρίζεται αποκλειστικά στην υλοποίηση του προσομοιωτή sim-outorder. Με άλλα λόγια, αν δεν υφίσταται αυτή η προσομοίωση τότε δεν υφίσταται και το εργαλείο. Δεν είναι λοιπόν κάποιο εργαλείο το οποίο δέχεται εισόδους ανεξάρτητα και μπορεί να σταθεί μόνο του σαν μια ξεχωριστή υλοποίηση. Κλειδί στην όλη διαδικασία υπολογισμού της ενέργειας αποτελεί ο τρόπος εκτίμησης της ο οποίος πέραν των παραμετροποιημένων μοντέλων που αναλύσαμε στο κεφάλαιο 5, εμπλέκει το συνολικό αριθμό προσβάσεων στο υλικό του οποίου την ενέργεια εκτιμούμε.

Αναφερόμενοι λοιπόν στο ότι το Wattch είναι άρρηκτα συνδεδεμένο με τον προσομοιωτή sim-outorder εννοούμε ότι μετά την ανάπτυξη του προσομοιωτή, η ομάδα που ανέπτυξε το Wattch πρόσθεσε εντός του αρχείου sim-outorder.c πρόσθετο κώδικα ο οποίος ελέγχει τον τρόπο με τον οποίο αυξάνονται οι μετρητές των προσβάσεων σε κάθε μονάδα υλικού. Για να εκτιμήσουμε λοιπόν κι εμείς την καταναλισκόμενη ισχύ κατά την εκτέλεση εφαρμογών με τον προσομοιωτή μας, αφότου τελειώσαμε την προσομοίωση της εκτέλεσης, προσθέσαμε κώδικα που αφορά τους μετρητές αυτούς που κρίνουν το ποσό της ενέργειας που καταναλώνεται.

Προκειμένου να επεκτείνουμε τον κώδικα όμως με τον τρόπο που περιγράψαμε παραπάνω, αρχικά ενσωματώσαμε το Wattch. Κατ' αρχήν προμηθευτήκαμε από την σελίδα που φιλοξενεί το Wattch τα απαραίτητα αρχεία. Τα αρχεία αυτά ήταν τα *power.[c/h]* που αποτελούν τον κορμό της προσομοίωσης της ενέργειας γιατί περιλαμβάνουν όλα τα παραμετροποιημένα μοντέλα αλλά και τις μεταβλητές που καθορίζονται στην προσομοίωση της αρχιτεκτονικής και επηρεάζουν την εκτίμηση της ενέργειας. Ακόμη, εκτός αυτών των αρχείων σημαντική είναι και η ενσωμάτωση του φακέλου *Cacti* που περιέχει άλλες επίσης απαραίτητες συναρτήσεις που χρησιμοποιούνται από τις ορισμένες

συναρτήσεις των δυο προαναφερθέντων αρχείων. Ο φάκελος τοποθετήθηκε στον φάκελο *$IDIR/simplesim-3.0* όπου βρίσκονται όλα τα αρχεία του Simplescalar.
Για να κάνουμε λοιπόν τον προσομοιωτή μας να "βλέπει" τα καινούρια αυτά αρχεία και να συνδέσει τα object αρχεία τους στο εκτελέσιμο του, τροποποιήσαμε για άλλη μια φορά το αρχείο Makefile. Μετά από αυτήν την τροποποίηση στην ουσία ο sim-rapidloop υλοποιεί και την προσομοίωση της ενέργειας, αρκεί να γίνει σωστός χειρισμός στις μεταβλητές και συναρτήσεις που καθορίζουν τις εκτιμήσεις. Παρακάτω θα εξηγήσουμε επακριβώς τον τρόπο με τον οποίο χειριστήκαμε εμείς τον υπολογισμό της ενέργειας.

6.3 Επέκταση της προσομοίωσης με το Wattch

Ξεκινώντας την ανάλυση της υλοποίησης να πούμε ότι τίποτα δεν αλλάζει στην μέχρι τώρα υλοποίηση του προσομοιωτή μας. Η εκτέλεση των εντολών και ό,τι αυτό συνεπάγεται γίνεται με τον ίδιο ακριβώς τρόπο που γινόταν και προηγουμένως. Ο στόχος τώρα είναι διαχειριστούμε τις λειτουργίες που παρέχει το Wattch ώστε να μπορέσουμε να εκτιμήσουμε την κατανάλωση ισχύος του προσομοιωτή μας. Σε αυτό το σημείο να αναφέρουμε ότι η υλοποίηση μας στηρίζεται εξ' ολοκλήρου στην χρησιμοποίηση των ήδη υπαρχόντων συναρτήσεων και μοντέλων του Wattch με τρόπο που θα δείξουμε παρακάτω αναφέροντας την χρήση των καίριων μεταβλητών και συναρτήσεων.

Η προσομοίωση ενέργειας μέσω του Wattch χρησιμοποιεί κάποιες μεταβλητές που είναι ορισμένες εντός του προσομοιωτή και αφορούν τον τρόπο επεξεργασίας των εντολών και το υλικό του επεξεργαστή. Στην ουσία οι μεταβλητές αυτές γίνονται define μέσω του προσομοιωτή και declare εντός του αρχείου power.c. Οι συγκεκριμένες μεταβλητές όμως είναι μεταβλητές οι οποίες υπήρχαν και προηγουμένως και απλά τώρα τις χρησιμοποιεί και το Wattch. Μια μεταβλητή η οποία δεν υπήρχε και γίνεται τώρα define και αρχικοποιείται(με 32) είναι η "data_width" που αποτελεί το εύρος σε bits του datapath. Οι

υπόλοιπες μεταβλητές-μετρητές που κρατούν το πλήθος των προσβάσεων σε κάθε πόρο ανά κύκλο μηχανής παρουσιάζονται στον παρακάτω πίνακα.

Πίνακας 7. Τρόπος χειρισμού των μεταβλητών που καθορίζουν την εκτίμηση της ενέργειας

Μεταβλητή-μετρητής	***Επεξήγηση***
rename_access	αφορά το τμήμα της μετονομασίας και αυξάνεται στάδιο αποκωδικοποίησης των εντολών
bpred_access	η πρόβλεψη διακλάδωσης είναι μηδενική αφού δεν ενσωματώνεται στην προσομοίωση
window_access	αυξάνεται σε κάθε πρόσβαση στο παράθυρο εντολών-λίστα RUU σε οποιοδήποτε στάδιο
lsq_access	αυξάνεται σε κάθε πρόσβαση στο παράθυρο εντολών-λίστα LSQ σε οποιοδήποτε στάδιο
regfile_access	αυξάνεται κατά την εκτέλεση και απόσυρση των εντολών σε κάθε πρόσβαση στον ΦΚ
icache_access	αυξάνεται σε κάθε πρόσβαση στην μνήμη εντολών, δηλαδή στο στάδιο FETCH
dcache_access	η μνήμη εντολών προσπελαύνεται σε πιθανή εκτέλεση εντολών μνήμης και ο μετρητής αυξάνεται στο στάδιο DISPATCH
alu_access	αυξάνεται σε κάθε χρήση της μονάδας ALU στο στάδιο DISPATCH.
ialu_access	αυξάνεται σε κάθε πράξη ακεραίων στο στάδιο DISPATCH.
falu_access	αυξάνεται σε κάθε πράξη αριθμών κινητής υποδιαστολής στο στάδιο DISPATCH
resultbus_access	αυξάνεται σε μεταφορά της εξόδου στον φάκελο καταχωρητών ή στην διαδικασία της προώθησης της εξόδου, αν υπάρχουν εξαρτημένες εντολές από το συγκεκριμένο δεδομένο.
Window_preg_access	αυξάνεται σε κάθε πρόσβαση του παραθύρου εντολών-λίστα RUU είτε για εγγραφή είτε για

	ανάγνωση των καταχωρητών, δηλαδή στο στάδιο DISPATCH και COMMIT
window_selection_access	αυξάνεται στο στάδιο SCHEDULE
window_wakeup_access	αυξάνεται σε κάθε προώθηση εξόδου σε άλλη εντολή από την λίστα RUU, δηλαδή στο DISPATCH
lsq_store_data_access	αυξάνεται σε κάθε εκτέλεση εντολής store
lsq_load_data_access	αυξάνεται σε κάθε εκτέλεση εντολής load
lsq_preg_access	αυξάνεται σε κάθε πρόσβαση του παραθύρου εντολών-λίστα LSQ είτε για εγγραφή είτε για ανάγνωση των καταχωρητών, δηλαδή στο στάδιο DISPATCH και COMMIT
lsq_wakeup_access	αυξάνεται σε κάθε προώθηση εξόδου σε άλλη εντολή από την λίστα LSQ, δηλαδή στο DISPATCH

Στην αρχή κάθε επανάληψης του βρόγχου που αναφέραμε στην ενότητα 6.2.3, δηλαδή σε κάθε διαπέραση του pipeline, καλούμε την συνάρτηση clear_access_stats η οποία μηδενίζει όλες τις παραπάνω μεταβλητές. Στην συνέχεια οι μεταβλητές διαμορφώνονται με τον τρόπο που περιγράφουμε στον πίνακα 7 και στο τέλος του βρόγχου(pipeline) καλούμε την συνάρτηση update_power_stats η οποία υπολογίζει την ενέργεια του κάθε πόρου βάσει του αριθμού των προσβάσεων σε αυτόν και την προσθέτει στην συνολική ενέργεια που έχει καταναλώσει ο πόρος μέχρι εκείνη την στιγμή. Έτσι στο τέλος της εκτέλεσης έχουμε την καταναλισκόμενη ισχύ για κάθε πόρο που μας οδηγεί στο να βρούμε την συνολική ισχύ ολόκληρου του συστήματος.

Όπως αναφέραμε, η προσομοίωση μας στηρίχτηκε αποκλειστικά στις ήδη υπάρχουσες μεθόδους του Wattch για τους ήδη υπάρχοντες πόρους. Για την καινούρια μονάδα Branch Unit που διαχειρίζεται τις ειδικές εντολές ελέγχου σαφώς και δεν υπάρχει υλοποίηση στο Wattch. Θεωρούμε λοιπόν ότι καταναλώνει ισχύ που ισούται με την εκτέλεση 2 εντολών ακεραίων(λόγω του ελέγχου της συνθήκης και του υπολογισμού του επόμενου loop index) και έτσι χρεώνουμε αντίστοιχα αυτή την μονάδα. Το ίδιο συμβαίνει και κατά την εκτέλεση των ειδικών εντολών μνήμης, για τις οποίες εκτός της πρόσβασης στην

μνήμη δεδομένων, καταμετρούμε και ισχύ όπως για την εκτέλεση μιας ακέραιας εντολής. Ακόμη, θεωρούμε ότι η προώθηση δεδομένων θα έχει αντίστοιχη κατανάλωση με την μεταφορά δεδομένων στον ΦΚ(result_bus) και εκεί καταμετρούμε την αντίστοιχη ενέργεια σε περίπτωση προώθησης. Τέλος, η προώθηση δεδομένων μιας εντολής προς τον εαυτό της δεν καταμετράται στην καταναλισκόμενη ισχύ επειδή θεωρούμε ότι μια τέτοια πράξη απαιτεί ελάχιστη έως καθόλου ισχύ, εφόσον το δεδομένο παραμένει στην μονάδα-εντολή και δεν διανύει κάποιο μονοπάτι.

7. Πειραματικά Αποτελέσματα

7.1 Τρόπος αξιολόγησης

Ο σκοπός της προσομοίωσης που αναπτύξαμε στα πλαίσια αυτής της διπλωματικής εργασίας ήταν να αξιολογήσει κατά πόσο η νέα αυτή αρχιτεκτονική εκπληρώνει τους στόχους της, οι οποίοι είναι η επιτάχυνση της εκτέλεσης και η μείωση της καταναλισκόμενης ενέργειας αξιοποιώντας την ιδιαίτερη μορφή εκτέλεσης των rapid loops που περιγράψαμε. Πλέον έχουμε έναν χειροπιαστό τρόπο να δούμε τα οφέλη που προσφέρει η νέα αρχιτεκτονική εκτελώντας διάφορες εφαρμογές μέσω της προσομοίωσης.

Βασικό χαρακτηριστικό της προσομοίωσης είναι η υλοποίηση της επιτάχυνσης βρόγχων αλλά και της εξοικονόμησης ενέργειας με την απενεργοποίηση του pipeline κατά την εκτέλεση των rapid loops. Με αυτόν τον τρόπο οι εντολές των βρόγχων θα διέρχονται μια φορά από το pipeline και θα παραμένουν στις λειτουργικές μονάδες για να εκτελεστούν όσες φορές απαιτεί η συνθήκη ελέγχου του βρόγχου. Κατ' επέκταση, αυτό φαίνεται πως θα επιταχύνει κατά πολύ την εκτέλεση των εφαρμογών και θα μειώσει τις απαιτήσεις σε ενέργεια σε συνολικό επίπεδο, μιας και οι βρόγχοι αποτελούν τον βασικότερο λόγο καθυστέρησης και κατανάλωσης ισχύος.

Τα παραπάνω όμως δεν συνιστούν τεκμηρίωση αλλά υπόθεση της απόδοσης της νέας αρχιτεκτονικής. Για να δείξουμε τα οφέλη της εκτέλεσης με την νέα αρχιτεκτονική θα παρουσιάσουμε τα αποτελέσματα από την εκτέλεση κώδικα σε δυο μορφές. Η πρώτη μορφή κώδικα θα εκτελείται μέσω της νέας αρχιτεκτονικής, δηλαδή θα υλοποιεί εκτέλεση εκτός σειράς χρησιμοποιώντας τις 16 λειτουργικές μονάδες που περιγράψαμε, αλλά δεν θα ενσωματώνει την επιτάχυνση βρόγχων. Έτσι κάθε βρόγχος του κώδικα αυτού θα εκτελείται με τον συνηθισμένο τρόπο, με τις εντολές να διέρχονται από όλα τα στάδια του pipeline πολλαπλές φορές. Η δεύτερη μορφή κώδικα εν αντιθέσει, θα ενσωματώνει την επιτάχυνση βρόγχων, άρα και την εξοικονόμηση

ενέργειας. Τον κώδικα της πρώτης μορφής τον δίνουμε στο σύστημα σε γλώσσα προγραμματισμού C και μεταγλωττίζεται από τον cross-compiler του Simplescalar ώστε να εκτελεστεί μέσω της προσομοίωσης. Ο κώδικας της δεύτερης μορφής που θέλουμε να εκτελέσουμε δεν μπορεί να προκύψει από την μεταγλώττιση του κώδικα γλώσσας C, μιας και οι νέες εντολές χαμηλού επιπέδου που αναπτύξαμε δεν ενσωματώνονται στον cross-compiler. Έτσι λοιπόν προσπερνώντας τον compiler, κάνουμε εμείς την μετάφραση του κώδικα σε assembly και αυτόν τον κώδικα δίνουμε στον assembler για την προσομοίωση της εκτέλεσης. Οι κώδικες λοιπόν που χρησιμοποιήσαμε φαίνονται στον παρακάτω πίνακα.

Πίνακας 8. Κώδικες ελέγχου της προσομοίωσης(benchmarks)

Κώδικας 1

```
for ( k=1 ; k<=n ; k++ ) {
   x[k] = q + y[k]*( r*z[k+10] + t*z[k+11] );
}
```

Κώδικας 2

```
 for ( k=1 ; k<=n ; k++ ) {
     q += z[k]*x[k];
  }
```

Μεταφρασμένος Κώδικας 1

```
$L1:    lbeqi1  $13,$14,$L2
        lwif    $17,0($15)
        mult    $17,$18
        mflo    $17
        lwif    $19,0($16)
        mult    $19,$20
        mflo    $19
        add     $19,$19,$17
        lwif    $17,0($22)
        mult    $17,$19
        mflo    $17
        add     $21,$21,$17
        swdf    $21,($22)
$L2:
```

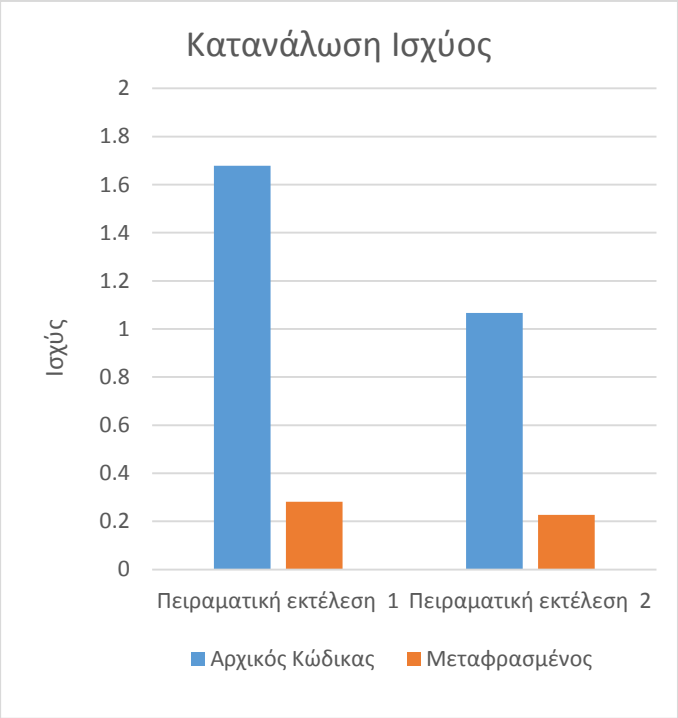
Μεταφρασμένος Κώδικας 2

```
$L1:    lbeqi1 $13,$14,$L2
        lwif $17,0($11)
        lwif $18,0($12)
        mult $17,$18
        mflo $16
        add $15,$15,$16
$L2:
```

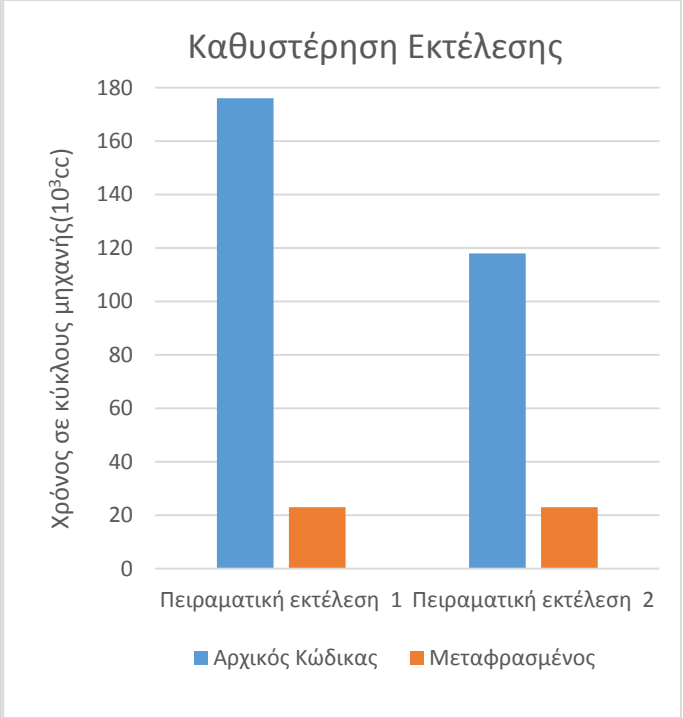
7.2 Σχολιασμός Πειραμάτων

Οι μετρήσεις που παίρνουμε μέσω της προσομοίωσης είναι πολλαπλές και αφορούν είτε το κομμάτι του χρόνου, είτε το κομμάτι της ενέργειας. Σε ό,τι αφορά την ενέργεια, το Wattch προσφέρει όπως αναφέραμε στο κεφάλαιο 5, τρείς διαφορετικές πολιτικές για την καταμέτρηση της ενέργειας(conditional clocking). Εμείς θεωρώντας την πολιτική 3(cc3) αντιπροσωπευτική της προσομοίωσης, λαμβάνουμε αυτήν υπόψιν κατά την ανάγνωση των μετρήσεων και μέσω αυτής θα παρουσιάσουμε και τα πειράματα μας. Τα διαγράμματα 1 και 2 λοιπόν που παρουσιάζονται παρακάτω, μας δίνουν μια πρώτη ιδέα για την κατανάλωση της ισχύος και την καθυστέρηση της εκτέλεσης των παραπάνω κωδίκων για 2000 επαναλήψεις.

Διάγραμμα 1.



Διάγραμμα 2.



Είναι φανερό ότι και στις δύο περιπτώσεις πειραμάτων, η ενσωμάτωση της επιτάχυνσης βρόγχων δημιουργεί ένα μεγάλο χάσμα στην απόδοση. Ο μεταφρασμένος κώδικας του πειράματος 1 καταναλώνει σχεδόν πέντε φορές λιγότερη ενέργεια και εκτελείται αντίστοιχα πέντε φορές πιο γρήγορα από τον αρχικό κώδικα. Αντίστοιχα, στο δεύτερο πείραμα, ο

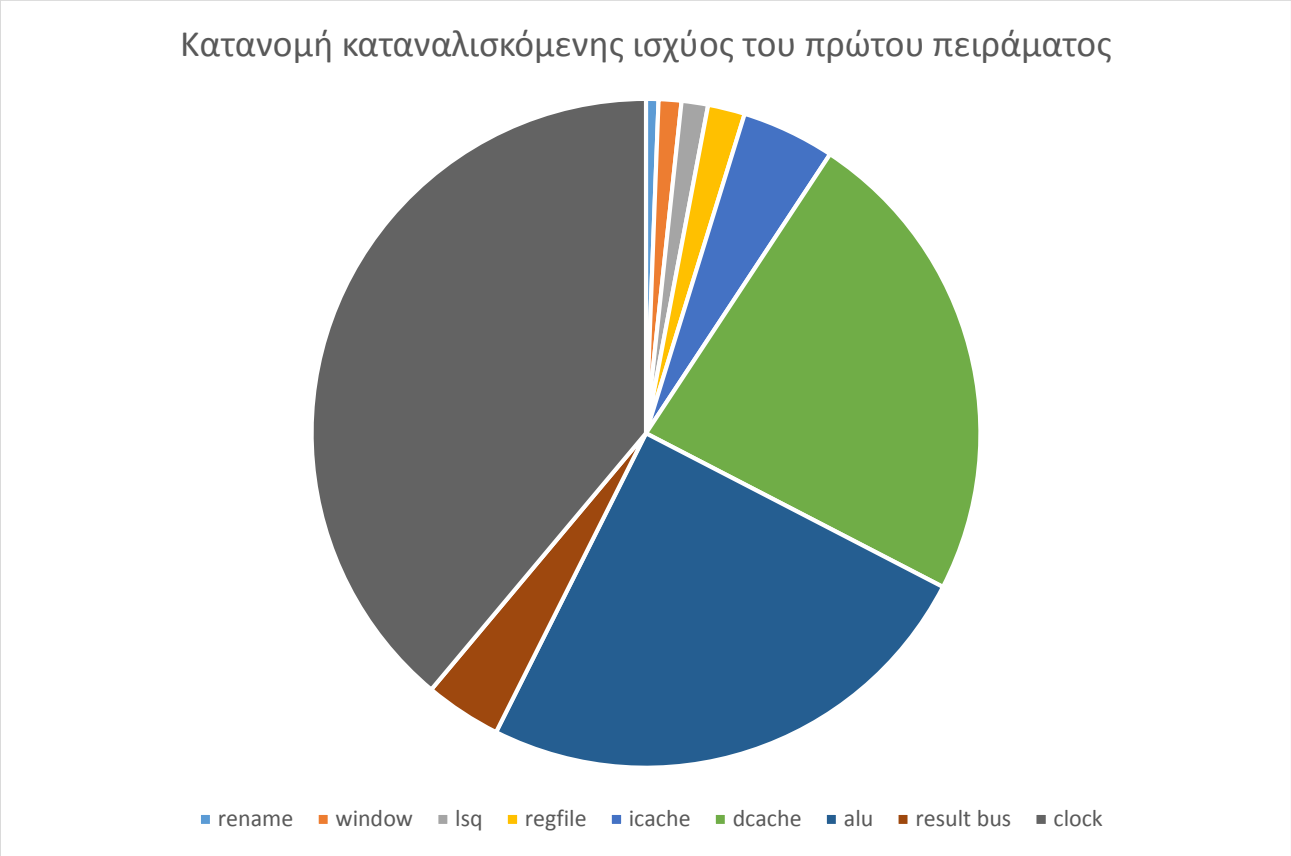
μεταφρασμένος κώδικας υπερτερεί του αρχικού εξαιτίας της μειωμένης καταναλισκόμενης ενέργειας του κατά έξι φορές αλλά και της γρηγορότερης εκτέλεσης του κατά 7.5 φορές από τον αρχικό κώδικα. Στο σημείο αυτό να επισημάνουμε ότι κατά την εκτέλεση εφαρμογών χρησιμοποιώντας τις προσομοιώσεις του Simplescalar, εκτελούνται αθροιστικά πάνω από 6000 εντολές χαμηλού επιπέδου. Στην περίπτωση μας, οι βρόγχοι που εκτελούμε κάνουν 2000 επαναλήψεις και το ποσό σε κύκλους μηχανής αλλά και σε κατανάλωση ισχύος δεν φαντάζει υπερβολικό αλλά σε διαφορετικές περιπτώσεις και πάλι θα πρέπει να λάβουμε υπόψιν μας ότι εντός των μετρήσεων συμπεριλαμβάνεται και ένα μεγάλο κομμάτι εντολών το οποίο δεν ανήκει στην εφαρμογή.

Αυτό που παρατηρούμε από τα παραπάνω διαγράμματα είναι ότι με την επιτάχυνση βρόγχων η απόδοση του συστήματος μεταξύ των δύο πειραμάτων δεν διαφέρει κατά μεγάλο βαθμό. Βλέπουμε, λοιπόν ότι τα δύο πειράματα στο επίπεδο του αρχικού κώδικα έχουν δικαιολογημένα μεγάλη διαφορά τόσο στην καθυστέρηση όσο και στην κατανάλωση ενέργειας λόγω της μεγαλύτερης έκτασης του πρώτου πειράματος. Αντιθέτως, όταν εκτελούμε τους μεταφρασμένους κώδικες, η διαφορά μεταξύ των δυο πειραμάτων είναι ελάχιστη παρά το εμφανές μεγαλύτερο πλήθος εντολών του πειράματος 1. Ειδικότερα οι κύκλοι μηχανής της εκτέλεσης τους διαφέρουν ελάχιστα έως αμελητέα για το εύρος της εκτέλεσης αυτής, αφού το πρώτο πείραμα εκτελείται για 23014 κύκλους και το δεύτερο για 23005. Χαρακτηριστικά, για να πετύχουμε σημαντική διαφορά μεταξύ των δύο πειραμάτων στο επίπεδο του μεταφρασμένου κώδικα, μειώσαμε τις επαναλήψεις του δεύτερου πειράματος στο μισό, δηλαδή 1000. Τότε η κατανάλωση ενέργειας του έγινε ίση με 0.181 και η καθυστέρηση του ίση με 18505 κύκλους μηχανής.

Σημαντικό επίσης είναι να διακρίνουμε τον τρόπο με τον οποίο κατανέμεται η συνολική ισχύς στους διάφορους πόρους του συστήματος. Η συνολική καταναλισκόμενη ισχύς προκύπτει αθροιστικά από την ισχύ που καταναλώνεται κατά την μετονομασία των καταχωρητών, την πρόσβαση στο παράθυρο εντολών(λίστες RUU-LSQ), στον φάκελο καταχωρητών και την μνήμη δεδομένων και εντολών. Ακόμη,

συμπεριλαμβάνεται η καταναλισκόμενη ισχύς κατά την χρήση της αριθμητικής λογικής μονάδας, του result bus, καθώς και η ισχύς του ρολογιού. Για τον λόγο αυτόν παρουσιάζουμε ένα ακόμα διάγραμμα που δείχνει ποσοστιαία την κατανάλωση ισχύος σε κάθε πόρο που το Wattch προσομοιώνει, χρησιμοποιώντας τον κώδικα του πρώτου πειράματος που ενσωματώνει την επιτάχυνση βρόγχων και πάλι για 2000 επαναλήψεις.

Διάγραμμα 3.



Το διάγραμμα 3 φανερώνει ότι η ισχύς που καταναλώνεται στο ρολόι του επεξεργαστή αποτελεί το μεγαλύτερο κομμάτι επί του συνόλου. Αυτό ήταν κάτι αναμενόμενο μιας και είναι γνωστό ότι το κύκλωμα του ρολογιού απαιτεί την περισσότερη ισχύ στους υπερβαθμωτούς επεξεργαστές υψηλής απόδοσης. Το επόμενο μεγαλύτερο κομμάτι ενέργειας καταναλώνεται στην μνήμη δεδομένων και έπειτα στην αριθμητική λογική μονάδα. Η ενέργεια της κεντρικής λογικής μονάδας είναι το άθροισμα της χρήσης της μονάδας τόσο για πράξεις ακεραίων όσο και για πράξεις αριθμών κινητής υποδιαστολής και σύμφωνα με

την προσομοίωση μας, σε αυτήν εμπεριέχεται και η ενέργεια κατά την εκτέλεση των ειδικών εντολών μνήμης και ελέγχου με τρόπο που περιγράψαμε στην ενότητα 6.4. Ομοίως, η ενέργεια της μνήμης δεδομένων περιλαμβάνει και κομμάτι της ενέργειας κατά την εκτέλεση των ειδικών εντολών μνήμης.

Από το διάγραμμα φαίνεται πως το ποσοστό της ενέργειας που καταναλώνεται στην μνήμη εντολών καθώς και στον φάκελο καταχωρητών είναι πολύ μικρό σε σχέση με το συνολικό ποσοστό. Είναι φανερό ότι αυτό συμβαίνει λόγω της αξιοποίησης της επιτάχυνσης βρόγχων που προσφέρει η αρχιτεκτονική, η οποία μειώνει τις προσκομίσεις εντολών και τις προσβάσεις στο φάκελο καταχωρητών. Οι εντολές που περιλαμβάνονται στο rapid loop προσκομίζονται μια φορά στο σύστημα, διαβάζουν μια φορά τους καταχωρητές εισόδου και γράφουν μια φορά τω καταχωρητή εξόδου. Έτσι λοιπόν η περισσότερη ισχύς κατανέμεται είτε στην μνήμη δεδομένων είτε στην αριθμητική λογική μονάδα, ανάλογα με το είδος των εντολών assembly που εκτελούνται.

Πίνακας 9. Αναλυτικές μετρήσεις της καταναλισκόμενης ισχύος του πρώτου πειράματος

Προέλευση κατανάλωσης	**Ισχύς**	**Ποσοστό επί του συνόλου**
rename	1693.9932	0.60
window	3102.5595	1.10
lsq	3594.5282	1.28
register file	4999.3327	1.79
icache	12698.0418	4.51
dcache	65597.7236	23.32
arithmetic logical unit	69761.9575	24.80
result bus	10408.6415	3.70
clock	109405.9121	38.90

8. Μελλοντική Ανάπτυξη

Η εργασία αυτή ήταν μια πολύ φιλόδοξη προσπάθεια να προσομοιώσουμε την κατανάλωση ενέργειας κατά την εκτέλεση εφαρμογών μέσω της αρχιτεκτονικής που παρουσιάσαμε στο κεφάλαιο 3. Προσπαθήσαμε να αποτυπώσουμε με τον καλύτερο δυνατό τρόπο όλα τα κομμάτια που ενσωματώνει το συγκεκριμένο μοντέλο του επεξεργαστή χρησιμοποιώντας τα εργαλεία Simplescalar και Wattch. Το σίγουρο όμως είναι ότι η προσομοίωση αυτή έχει την δυνατότητα να βελτιωθεί και να επεκταθεί περισσότερο.

Κάτι βασικό το οποίο χρειάζεται επέκταση είναι τεχνική των pipelines εκτέλεσης των εντολών. Στα πλαίσια αυτής της διπλωματικής εργασίας, η προσομοίωση που αναπτύξαμε ενσωματώνει ένα pipeline που μοιράζεται δεκαέξι μονάδες εκτέλεσης, παρότι ο ορισμός της αρχιτεκτονικής περιλαμβάνει τέσσερα pipelines. Βλέποντας όμως τα αποτελέσματα της προσομοίωσης, τα οποία αναλύσαμε στο κεφάλαιο 7, θεωρούμε ότι αξίζει η οποιαδήποτε προσπάθεια για να επεκταθεί η προσομοίωση και να ενσωματώσει 4 pipelines. Πιστεύουμε πως έτσι η απόδοση θα είναι ακόμα μεγαλύτερη και ταυτόσημη με αυτή την οποία πρεσβεύει η αρχιτεκτονική.

Ένα άλλο σημείο στο οποίο μπορεί να βελτιωθεί η προσομοίωση είναι η ανάπτυξη πρόβλεψης διακλαδώσεων και η περαιτέρω αναλυτική αποτύπωση των μονάδων εκτέλεσης με τον τρόπο που ορίζονται από την αρχιτεκτονική. Η πρόβλεψη διακλάδωσης θα ήταν χρήσιμη στις εντολές άμεσου άλματος χωρίς συνθήκη αλλά και στις απλές εντολές ελέγχου. Ο συγχρονισμός που αναπτύξαμε αντί της υποθετικής εκτέλεσης, είναι απαραίτητος για την ενσωμάτωση της επιτάχυνσης των βρόγχων, όμως σε ότι αφορά τα άμεσα άλματα χωρίς συνθήκη και τις εντολές ελέγχου, προσθέτει περισσότερη καθυστέρηση στο σύστημα. Σε πιθανή σύγκριση λοιπόν του μοντέλου του επεξεργαστή με κάποιον άλλον θεωρούμε ότι θα ήταν καλό να υπάρχει η υποθετική εκτέλεση ώστε να φανεί εξ' ολοκλήρου το εύρος της απόδοσης που μπορεί να επιτευχθεί. Σε ό,τι αφορά τις μονάδες εκτέλεσης, η καλύτερη προσομοίωση τους είναι

επίσης επιθυμητή ώστε να μην υποθέτουμε τον τρόπο σύνδεσης μεταξύ τους, αλλά να ακολουθούν την τοπολογία που παρουσιάσαμε στην ενότητα 3.2.3

Σχετικά με το Wattch, όπως αναφέραμε και στο κεφάλαιο 6, οι υπολογισμοί της ισχύος βασίζεται αποκλειστικά στις υπάρχουσες μεθόδους του Wattch. Στόχος λοιπόν είναι να επεκτείνουμε την υλοποίηση του Wattch ώστε να περιλαμβάνει μοντελοποιημένες παραμέτρους υπολογισμού ενέργειας που αφορούν τα νέα στοιχεία που ενσωματώνει η αρχιτεκτονική, όπως η νέα μονάδα εκτέλεσης ειδικών εντολών βρόγχων. Στην προσομοίωση μας χρησιμοποιούμε έναν τρόπο που συμπεριλαμβάνει και αυτά τα κομμάτια στην συνολική ενέργεια, αλλά μια αναλυτική προσομοίωση θα είναι πιο παραστατική και δεν θα αφήνει περιθώρια αμφισβήτησης. Με τα μέχρι τώρα δεδομένα, το σφάλμα εκτίμησης της ενέργειας δεν μπορεί να προσδιοριστεί επακριβώς. Εφόσον όμως το Wattch έχει σφάλμα που δεν ξεπερνά το 10% υποθέτουμε πως και το δικό μας σφάλμα περιορίζεται σε κάποιο όχι πολύ μεγαλύτερο ποσοστό. Βέβαια αναφέροντας το προηγούμενο να ξεκαθαρίσουμε ότι τα αποτελέσματα των πειραμάτων που παρουσιάσαμε στο κεφάλαιο 7 δεν μπορούν να αμφισβητηθούν διότι αφορούν την εκτέλεση εφαρμογών στην ίδια αρχιτεκτονική και έτσι το σφάλμα είναι κοινό.

ΠΑΡΑΡΤΗΜΑ

ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ SIMPLESCALAR

Παρακάτω δίνονται οδηγίες για την εγκατάσταση του Simplescalar στο προσωπικό μας υπολογιστή και συγκεκριμένα στο περιβάλλον Ubuntu. Προμηθευτήκαμε τον προσομοιωτή από την επίσημη σελίδα του Simplescalar στο διαδίκτυο επιλέγοντας την τελευταία διαθέσιμη έκδοση τόσο των προσομοιωτών όσο και των λοιπών εργαλείων(PISA GNU GCC compiler, PISA assembler, linker).Τα αρχεία που επιλέξαμε ήταν τα *simplesim-3v0e.tgz, simpletools-2v0.tgz* και *simpleutils-2v0.tgz.* Παρακάτω περιγράφουμε τον τρόπο εγκατάστασης του Simplescalar και στο Παράρτημα παραθέτουμε τις σημαντικές αλλαγές που έπρεπε να γίνουν ώστε να μπορέσει να γίνει σωστά η εγκατάσταση. Η διαδικασία λοιπόν που ακολουθήσαμε είναι η εξής:

1. Αρχικοποίηση του περιβάλλοντος εγκατάστασης του Simplescalar. Θέτουμε το μονοπάτι προς τον φάκελο εγκατάστασης σε μια μεταβλητή $IDIR (*export IDIR=/home/path*) και δυο ακόμα παραμέτρους, τις $HOST και $TARGET. Το $HOST είναι ένα string που αντιπροσωπεύει την αρχιτεκτονική και το σύστημα του υπολογιστή όπου πρόκειται να εγκαταστήσουμε τον προσομοιωτή και το $TARGET είναι ο τύπος endian του συστήματος. Στην συγκεκριμένη περίπτωση λοιπόν εκτελέσαμε τις εντολές *export HOST=i386-unknown-linux* και *export TARGET=sslittle-na-sstrix* διότι ο υπολογιστής μας είναι τύπου little endian, σε διαφορετική περίπτωση αρχικοποιούμε με *ssbig-na-sstrix.*
2. Αφού δημιουργήσαμε τον φάκελο $IDIR(mkdir $IDIR) μεταφέραμε και αποσυμπιέσαμε εκεί όλα τα αρχεία. Μετά την αποσυμπίεση εμφανίστηκαν οι φάκελοι binutils-2.5.2, f2c-1994.09.27, gcc-2.6.3, glibc-1.09, simplesim-3.0, ssbig-na-sstrix και sslittle-na-sstrix.

3. Έπειτα σειρά είχε η εγκατάσταση των binutils.

```
cd $IDIR/binutils-2.5.2
configure --host=$HOST --target=$TARGET --with-gnu-as--with-
gnu-ld --prefix=$IDIR
make
make install
```

Έτσι δημιουργήθηκε ο φάκελος *bin* εντός του φακέλου $IDIR.

4. Εγκατάσταση των προσομοιωτών. Αυτό γίνεται απαραίτητα πριν την εγκατάσταση του cross-compiler, οποίος θα χρειαστεί τα αρχεία των προσομοιωτών για την εγκατάσταση του.

```
cd $IDIR/simplesim-3.0
make config-pisa
make
```

5. Τέλος, εγκατάσταση του cross-compiler.

```
cd $IDIR/ gcc-2.6.3
./configure --host=$HOST --target=$TARGET --with-gnu-as --
with-gnu-ld --prefix=$IDIR
chmod -R +w .
make LANGUAGES=c
../simplesim-3.0/sim-safe ./enquire -f >! float.h-cross
make install
```

Η παραπάνω διαδικασία παρουσιάζει πολλαπλά σφάλματα λόγω των διάφορων εκδόσεων του Simplescalar και για αυτό θεωρήσαμε απαραίτητο να παρουσιάσουμε ποιες αλλαγές κάναμε και σε ποια αρχεία.

1. Πριν την εγκατάσταση των binutils εκτελούμε τα εξής:

   ```
   $ cd $IDIR/binutils-2.5.2
   $ sed -i -e "s/va_list ap = args;/va_list ap; va_copy(ap,
   args);/g" libiberty/vasprintf.c
   $ sed -i -e "s/char \*malloc ();/\/\/char \*malloc ();/g"
   libiberty/vasprintf.c
   ```

2. Έπειτα εφαρμόζουμε τις εξής αλλαγές:

   ```
   binutils-2.5.2/libiberty/functions.def
   Line: 36
   Action: comment out
   Line: 56
   Action: comment out

   binutils-2.5.2/ld/ldlex.l
   Line: 476
   Action: change yy_current_buffer to YY_CURRENT_BUFFER
   binutils-2.5.2/ld/ldmisc.c
   Line: 24
   Action: change varargs.h to stdarg.h




   Line: 343 356 388 400
   Action: change all four functions info_msg() einfo() minfo()
   finfo() like this:

   343 void info_msg(const char* fmt, va_list ap)
   344 //va_dcl
   345 {
   ```

```
346 char *fmt;
347 va_list arg;
348 /* va_start(arg);
349 fmt = va_arg(arg, char *);
350 vfinfo(stdout, fmt, arg);
351 va_end(arg); */
352 }
```

gcc-2.6.3/cccp.c
Line: 194
Action: comment out
gcc-2.6.3/cp/g++.c
Line: 90
Action: comment out
gcc-2.6.3/sdbout.c
Line: 56
Action: replace the whole line with ”#if 0”
gcc-2.6.3/gcc.c
Line: 172
Action: comment out

3. Μετά από τις παραπάνω αλλαγές, εγκαθιστούμε κανονικά τα binutils και τους προσομοιωτές με τον τρόπο που περιγράψαμε στο κεφάλαιο 6.

4. Τέλος, πριν την εγκατάσταση του cross-compiler εφαρμόζουμε τις παρακάτω αλλαγές:
gcc-2.6.3/insn-output.c
Line: 675 750 823
Action: and a ‘\’ to the end of the line, and it will be

```
675 return "FIXME\n\
750 return "FIXME\n\
```

823 return "FIXME\n\
note that insn-output.c is generated by flex automatically, and your modification would be lost every time you make clean

Edit line 60 of protoize.c, and replace “#include <varargs.h>” with
“#include <stdarg.h>”

To fix an error message about decl.c, saying “invalid lvalue in increment”, do the following: Edit
obstack.h at line 341 and change:

- *((void **)__o->next_free)++=((void *)datum);\

with

- *((void **)__o->next_free++)=((void *)datum);\

Βιβλιογραφία

[1] Todd Austin, Doug Burger, Simplescalar Tutorial.

[2] Meiyappan, Somasundaram, Adding custom instructions to the Simplescalar.

[3] Paul R. Gray, Paul J. Hurst, Stephen H. Lewis, Robert G. Meyer, Analysis and Design of Analog Integrated Circuits, Forth edition.

[4] John L. Hennessy, David A. Patterson, Computer Architecture, Forth Edition: A Quantitative Approach.

[5] Sohi, Gurindar S., Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers.

[6] Athanassios Tziouvaras, Georgios Dimitriou, Rapid, Low-power Loop Execution in a Network of Functional Units.

[7] Doug Burger, Todd M. Austin, The SimpleScalar Tool Set, Version 2.0.

[8] David Brooks, Vivek Tiwari, Margaret Martonosi, Wattch:A Framework for Architectural-Level Power Analysis and Optimizations.

[9] Song Li, Zizhen Yao, Wattch-alpha: A Power Analysis Simulator for Alpha Architecture.

[10] Oguz, Cenk, A SimpleScalar-based full system simulator with a symmetric multiprocessor extension.

[11] Robert McDonald, Doug Burger, Steve Keckler, The Design and Implementation of the TRIPS Prototype Chip.

[12] Doug Burger, Stephen W. Keckler, Kathryn S. McKinley, Mike Dahlin, Lizy K. John, Calvin Lin, Charles R. Moore, James Burrill, Robert G. McDonald, William Yoder and the TRIPS team, Scaling to the End of Silicon with EDGE Architectures.

[13] Written by Maggie Johnson and revised by Julie Zelenski, Processor Architectures.

[14] Panagiotis Kritikakos, Low-Power High Performance Computing.

[15] Marco A. Ramírez, Adrian Cristal, Alexander V. Veidenbaum, Luis Villa, Mateo Valero, Direct Instruction Wakeup for Out-Of-Order Processors.