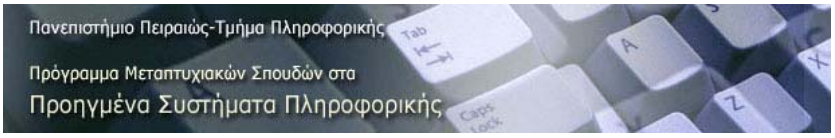
Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Αρχιτεκτονικές Υπερβαθμωτών Μικροεπεξεργαστών με Δυναμική και Εκτός Σειράς Εκτέλεση Εντολών και Αξιολόγηση της Απόδοσής τους**
Ονοματεπώνυμο Φοιτητή	**Βασίλειος Δήμητσας του Παύλου**
Αριθμός Μητρώου	**ΜΠΣΠ/08033**
Κατεύθυνση	**Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων (ΤΕΥΣ)**
Επιβλέπων	**Δημήτριος Γκιζόπουλος, Αναπληρωτής Καθηγητής**



Ημερομηνία Παράδοσης **Ιούνιος 2010**

Τριμελής Εξεταστική Επιτροπή

Δημήτριος Γκιζόπουλος Αναπληρωτής Καθηγητής	Ιωάννης Θεοδωρίδης Αναπληρωτής Καθηγητής	Μιχαήλ Ψαράκης Λέκτορας

Περιεχόμενα

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στους καθηγητές μου στο Πανεπιστήμιο Πειραιώς, κ. Γκιζόπουλο Δημήτριο και κ. Ψαράκη Μιχαήλ για τις πολύτιμες συμβουλές τους και τη συμπαράστασή τους όχι μόνο στη συγκεκριμένη μεταπτυχιακή εργασία αλλά και όλα τα χρόνια που ήμουν φοιτητής στο τμήμα Πληροφορικής. Ακόμη, θα ήθελα να ευχαριστήσω ιδιαίτερα τον υποψήφιο διδάκτορα του Πανεπιστημίου Πειραιώς Φουτρή Νικόλαο για τις καίριες και εύστοχες παρεμβάσεις του, κυρίως όσον αφορά στη συγγραφή της τεκμηρίωσης αλλά και καθ' όλη τη διάρκεια εκπόνησης της διατριβής αυτής.

Επίσης, ευχαριστώ την οικογένειά μου που συνεχίζει να μου παρέχει όλα τα εφόδια που χρειάζομαι για να πορεύομαι και να κοιτάζω με αισιοδοξία το μέλλον.

Τέλος, δε θα μπορούσα να ξεχάσω τη Βιβή, η οποία με την αμέριστη υποστήριξή της και θετική της ενέργεια μου δίνει κουράγιο και δύναμη να συνεχίσω στο δύσκολο δρόμο που έχω επιλέξει.

Περιεχόμενα του CD

Το CD που συνοδεύει τη διατριβή περιλαμβάνει τους εξής φακέλους:

documentation: Περιέχει την τεκμηρίωση της μεταπτυχιακής εργασίας σε μορφή εγγράφου Word και σε μορφή εγγράφου pdf.

applications: Σε αυτό το φάκελο βρίσκονται όλες οι εφαρμογές που εκτελέστηκαν και παρουσιάστηκαν στην τεκμηρίωση. Οι εφαρμογές είναι κατηγοριοποιημένες στους εξής υποφακέλους:

bubblesort: Περιέχει σε C (bubblesort.c) και σε συμβολική γλώσσα (bubblesort.s) το πρόγραμμα ταξινόμησης με φυσαλίδα, καθώς επίσης και το σχετικό αρχείο (results_bubble.txt) με τα αποτελέσματα της προσομοίωσης εκτέλεσης του συγκεκριμένου προγράμματος.

check_mem_deps: Περιέχει το αρχείο check_mem_deps.s και το σχετικό αρχείο (check_mem_depRes.txt) με τα αποτελέσματα της προσομοίωσης εκτέλεσης του συγκεκριμένου προγράμματος.

check_mem_deps2: Περιέχει το αρχείο check_mem_deps2.s και το σχετικό αρχείο (check_mem_depRes2.txt) με τα αποτελέσματα της προσομοίωσης εκτέλεσης του συγκεκριμένου προγράμματος.

check_memA: Περιέχει το αρχείο check_memA.s και το σχετικό αρχείο (check_memResA.txt) με τα αποτελέσματα της προσομοίωσης εκτέλεσης του συγκεκριμένου προγράμματος.

check_memB: Περιέχει το αρχείο check_memB.s και το σχετικό αρχείο (check_memResB.txt) με τα αποτελέσματα της προσομοίωσης εκτέλεσης του συγκεκριμένου προγράμματος.

processor_models: Στο φάκελο αυτό περιλαμβάνονται οι εκδόσεις του IVM (1.0 και 1.1), που έχουν αναρτηθεί στην ιστοσελίδα [7], καθώς επίσης και η τροποποιημένη του έκδοση που χρησιμοποιήθηκε στη συγκεκριμένη διατριβή. Πιο συγκεκριμένα ο φάκελος project περιλαμβάνει τους ακόλουθους υποφακέλους:

IVM-1.0: Περιέχει τα αρχεία Verilog του IVM-1.0 που βρίσκεται και στην [7].
IVM-1.1: Περιέχει τα αρχεία Verilog του IVM-1.1 που βρίσκεται και στην [7].
IVM-1.0modified: Περιέχει τον IVM-1.0 που χρησιμοποιήθηκε στη διατριβή.

Περίληψη

Σε αυτή τη μεταπτυχιακή διατριβή γίνεται μελέτη των αρχιτεκτονικών των υπερβαθμωτών επεξεργαστών. Οι υπερβαθμωτοί επεξεργαστές έχουν τη δυνατότητα να επιλύουν τις διακλαδώσεις και να καθορίζουν τη ροή των εντολών με δυναμικό τρόπο. Επίσης, ορισμένες κατηγορίες αυτών εκτελούν τις εντολές ενός προγράμματος εκτός σειράς.

Ως αντικείμενο έρευνας χρησιμοποιήθηκε ο επεξεργαστής IVM-1.0, ο οποίος σχεδιάστηκε στο Πανεπιστήμιο του Illinois, στην περιοχή Urbana-Champaign. Ο IVM είναι ο μοναδικός υπερβαθμωτός επεξεργαστής ο οποίος διατίθεται ελεύθερα στην επιστημονική κοινότητα. Υλοποιεί διοχέτευση 12 σταδίων και διαθέτει όλα εκείνα τα χαρακτηριστικά (μονάδα πρόβλεψης διακλαδώσεων, μονάδα πρόβλεψης εξαρτήσεων μνήμης, τεχνικές μετονομασίας, εκ των προτέρων προσκόμιση εντολών και άλλα) που τον κατατάσσουν στην κατηγορία των υπερβαθμωτών αρχιτεκτονικών.

Εκτός από την αναλυτική μελέτη της αρχιτεκτονικής του IVM, έγινε και η πειραματική αξιολόγησή του εκτελώντας διάφορα προγράμματα, υλοποιημένα στη γλώσσα προγραμματισμού C και σε συμβολική γλώσσα. Με την εκτέλεση των προγραμμάτων παρουσιάζονται στον αναγνώστη οι διάφορες πτυχές της λειτουργίας ενός υπερβαθμωτού επεξεργαστή.

Η μεταπτυχιακή εργασία έχει την ακόλουθη δομή: Αρχικά γίνεται μία εισαγωγική περιγραφή της μελέτης. Στο 1ο κεφάλαιο λαμβάνει χώρα μία αναλυτική περιγραφή της οργάνωσης των υπερβαθμωτών επεξεργαστών, επεξηγούνται οι λόγοι εφεύρεσής τους, αναφέρονται τα πλεονεκτήματά τους έναντι των βαθμωτών και τα γενικά χαρακτηριστικά τους. Στο 2ο κεφάλαιο παρουσιάζεται ο μηχανισμός πρόβλεψης εξαρτήσεων μνήμης με τη χρήση των συνόλων αποθήκευσης (store sets), που έχει σαν στόχο την αύξηση της απόδοσης του επεξεργαστή. Στο 3ο κεφάλαιο περιγράφεται διεξοδικά ο επεξεργαστής IVM και αναφέρονται με λεπτομέρεια όλα τα στάδια της διοχέτευσής του. Στο 4ο κεφάλαιο παρατίθεται η διαδικασία που πρέπει να ακολουθηθεί για την προσομοίωση της εκτέλεσης προγραμμάτων, υλοποιημένων στη γλώσσα προγραμματισμού C, στον IVM. Στο 5ο κεφάλαιο παρουσιάζεται η πειραματική αξιολόγηση του επεξεργαστή με τη βοήθεια προσομοιώσεων των εκτελέσεων κάποιων προγραμμάτων υλοποιημένων στη C και στη συμβολική γλώσσα του Alpha. Τέλος, ακολουθούν τα συμπεράσματα που εξήχθησαν από την παρούσα διατριβή και παρέχονται δύο παραρτήματα: Το παράρτημα Α περιέχει τα scripts που χρησιμοποιούνται κατά τη φάση παραγωγής του εκτελέσιμου προγράμματος και το Β περιέχει μία συνοπτική παρουσίαση της αρχιτεκτονικής συνόλου εντολών Alpha.

Abstract

In the current thesis the architectures of superscalar processors are studied. Superscalar processors are capable of out-of-order instructions' execution, resolving branches, as well as jumps and determining the correct instruction stream dynamically.

The research was focused mainly on IVM processor model. It was designed in University of Illinois, at Urbana-Champaign and it is the only superscalar processor that is freely available to the scientific community. It consists of a 12-stage pipeline and contains all of those characteristics (branch predictor, memory dependence predictor, renaming techniques, prefetching techniques and so on) that classify it in the category of superscalar architectures.

Furthermore, there has been an experimental evaluation of IVM by executing several programs, which were implemented in C programming language and in Alpha assembly language. The reader is thus able to comprehend the various aspects of the operation of a superscalar processor.

The content of the master thesis is structured as follows: Initially, there is an introductory description of the study which took place during the elaboration of the specific dissertation. The 1st chapter contains a detailed presentation of the organization of superscalar processors and their typical characteristics. In the 2nd chapter the store sets mechanism is described for predicting memory dependences. The 3rd chapter includes a thorough analysis of IVM processor and its pipeline stages. The 4th chapter describes the steps which are necessary in order to simulate a program execution in IVM. The 5th chapter contains the experimental evaluation of the processor. Finally, the thesis ends with the conclusions that were extracted from this research and two appendices. In appendix A the scripts, which

are used during the construction of the executable program, are available, and in appendix B Alpha Instruction Set Architecture is briefly described.

Εισαγωγή

Στόχος της παρούσης μεταπτυχιακής διατριβής είναι η εκ βαθέων αναλυτική μελέτη των αρχιτεκτονικών των υπερβαθμωτών μικροεπεξεργαστών με δυναμική και εκτός σειράς εκτέλεση εντολών. Οι υπερβαθμωτοί επεξεργαστές προσελκύουν το ενδιαφέρον των επιστημόνων και των ερευνητών στον τομέα της Αρχιτεκτονικής Υπολογιστών λόγω της πληθώρας τεχνικών που χρησιμοποιούνται με σκοπό την αύξηση της απόδοσης. Οι αρχιτεκτονικές αυτές χαρακτηρίζονται με τον όρο «επιθετικές» επειδή δε δίνεται σημασία σε παραμέτρους όπως η ενέργεια και το κόστος, οι οποίες μπορούν να περιορίσουν μία σχεδίαση, παρά μόνο στην απόδοση. Επομένως, είναι προφανές ότι ένας υπερβαθμωτός επεξεργαστής μπορεί να ενσωματώνει μία εξαιρετικά μεγάλη ποικιλία τεχνολογιών και να υλοποιεί ένα μεγάλο αριθμό τεχνικών αύξησης της απόδοσης.

Επιπρόσθετα, παρόλο που οι υπερβαθμωτές αρχιτεκτονικές είναι σχετικά παλαιότερες από αυτές των πολυπύρηνων και πολυνηματικών, η έρευνα στο χώρο της Αρχιτεκτονικής Υπολογιστών κάνει τον κύκλος της. Οι υπερβαθμωτές σχεδιάσεις έχουν επιστρέψει στο προσκήνιο και αυτό μπορεί να το καταλάβει κάποιος αν ενημερωθεί για τις ερευνητικές τάσεις των εταιριών Intel, AMD (για παράδειγμα η αρχιτεκτονική Intel Nehalem [13] και ο επεξεργαστής AMD phenom [14]) καθώς επίσης και άλλων που σχεδιάζουν πολυπύρηνα συστήματα με τον κάθε πυρήνα να είναι υπερβαθμωτός ή τουλάχιστον να υποστηρίζει την εκτέλεση εκτός σειράς των εντολών.

Ως εκ τούτων, υπάρχει μεγάλο ενδιαφέρον και κινητικότητα τα τελευταία χρόνια σε αυτές τις αρχιτεκτονικές. Στην παρούσα μεταπτυχιακή εργασία έγινε μια εκτενής έρευνα στις υπερβαθμωτές σχεδιάσεις. Φυσικά, όσο μεγάλη προσπάθεια και αν καταβλήθηκε και όσος χρόνος και αν δαπανήθηκε μελετώντας σχετικές δημοσιεύσεις και διαβάζοντας επιστημονικά συγγράμματα δε θα μπορούσε να γίνει τίποτα αν δεν υπήρχε ως σημείο αναφοράς ένα πραγματικό μοντέλο επεξεργαστή που περιλαμβάνει όλα εκείνα τα χαρακτηριστικά που μπορεί να διαθέτει μια υπερβαθμωτή αρχιτεκτονική. Διότι, συνδυάζοντας το θεωρητικό υπόβαθρο που μπορεί να αποκομίσει κάποιος με την παράλληλη μελέτη ενός αληθινού μοντέλου μόνο τότε μπορεί να αποκτήσει βαθειά γνώση της δυσπρόσιτης και δυσνόητης περιοχής των υπερβαθμωτών αρχιτεκτονικών.

Δυστυχώς η πρόσβαση σε τέτοιες σχεδιάσεις είναι εξαιρετικά περιορισμένη. Οι υπερβαθμωτοί επεξεργαστές ανήκουν στη συντριπτική πλειοψηφία σε εταιρίες και δεν είναι ελεύθεροι προς χρήση και μελέτη στην ακαδημαϊκή κοινότητα. Ο μόνος επεξεργαστής που διατίθεται χωρίς περιορισμούς είναι ο IVM. Η σχεδίασή του είναι υλοποιημένη σε Verilog και περιγράφεται σε επίπεδο RTL. Επίσης διατίθεται και ο προσομοιωτής Simplescalar, ο οποίος απλά προσομοιώνει τη λειτουργία υπερβαθμωτών επεξεργαστών και δεν αποτελεί ένα μοντέλο που θα παρέχει ακρίβεια προσομοιώσεων των εκτελέσεων προγραμμάτων σε κύκλο ρολογιού.

Για να μπορέσει ο αναγνώστης να κατανοήσει καλύτερα την αρχιτεκτονική του IVM το πρώτο κεφάλαιο αφιερώνεται σε μία γενική, αναλυτική παρουσίαση της οργάνωσης των υπερβαθμωτών επεξεργαστών. Δηλαδή παρατίθενται οι διαφορές μεταξύ των υπερβαθμωτών και των βαθμωτών διοχετεύσεων και οι λόγοι που οδήγησαν στην εφεύρεση των συγκεκριμένων αρχιτεκτονικών. Επίσης παρουσιάζονται τα προτερήματα της συγκεκριμένης αρχιτεκτονικής σε σχέση με την απλή, τη βαθμωτή.

Στο δεύτερο κεφάλαιο περιγράφεται μία τεχνική πρόβλεψης των εξαρτήσεων μνήμης με τη χρήση των συνόλων αποθήκευσης (store sets). Η τεχνική αυτή είναι από τις πιο δημοφιλείς για πρόβλεψη εξαρτήσεων μνήμης και επειδή υλοποιείται στον IVM της αφιερώθηκε ένα ξεχωριστό κεφάλαιο με σκοπό να μπορέσει ο αναγνώστης να την κατανοήσει και να καταλάβει πώς λειτουργεί μέσα σε έναν επεξεργαστή. Είναι εξαιρετικά σημαντικό να προβλέπονται οι εξαρτήσεις μνήμης, διότι η απόδοση ενός προγράμματος-έχει επιβεβαιωθεί στατιστικά-εξαρτάται από τις εντολές μνήμης και ιδιαίτερα από τις εντολές φόρτωσης, αφού χάρη σε αυτές προσκομίζονται τα δεδομένα από τη μνήμη, επί των οποίων γίνονται οι διάφοροι υπολογισμοί. Οπότε όσο πιο γρήγορα επιλύονται οι εξαρτήσεις των εντολών φόρτωσης τόσο πιο γρήγορα εκτελούνται και αυτό έχει θετική επίδραση σε όσες άλλες εντολές εξαρτώνται από τις συγκεκριμένες εντολές μνήμης.

Έχοντας παραθέσει στα δύο πρώτα κεφάλαια το απαιτούμενο θεωρητικό υπόβαθρο που θα διεγείρει το ενδιαφέρον του αναγνώστη για περαιτέρω μελέτη και θα του παρέχει την απαιτούμενη γνώση έτσι ώστε να μπορέσει να καταλάβει σε βάθος την αρχιτεκτονική του IVM, στο τρίτο κεφάλαιο αναλύεται σε επίπεδο RTL η δομή και η λειτουργία των σταδίων της διοχέτευσης του επεξεργαστή.

Έπειτα από πολλή σκέψη θεωρήθηκε πιο σωστός ο ακόλουθος τρόπος παρουσίασης των υπερβαθμωτών σχεδιάσεων: Μέσα από την εκτενή επεξήγηση του IVM, ο αναγνώστης μαθαίνει τα ποικίλα χαρακτηριστικά και τις επεκτάσεις των υπερβαθμωτών μικροεπεξεργαστών. Για παράδειγμα, όταν αναλύεται το στάδιο προσκόμισης, ο αναγνώστης εκτός του ότι κατανοεί τον IVM ενημερώνεται και για τους μηχανισμούς πρόβλεψης διακλαδώσεων, για την πρόωρη προσκόμιση των εντολών και άλλες τεχνικές αύξησης της απόδοσης, οι οποίες χρησιμοποιούνται κατά κόρον στις υπερβαθμωτές αρχιτεκτονικές.

Επίσης θεωρήθηκε σωστό να γίνει μία εκτενής αναφορά στη διαδικασία παραγωγής του κατάλληλου για τον IVM, εκτελέσιμου προγράμματος ($4^{ο}$ κεφάλαιο), όπως επίσης και στον τρόπο που μπορεί κάποιος να εκτελέσει μία προσομοίωση. Στο τέλος παρατίθεται μία πειραματική αξιολόγηση του επεξεργαστή διότι ο αναγνώστης πρέπει να παρατηρήσει πώς εφαρμόζονται οι τεχνικές αύξησης της απόδοσης και πώς λειτουργεί το μοντέλο στην πράξη.

Συνοψίζοντας, η παρούσα διατριβή έχει ως στόχο τη θεωρητική μελέτη και την πειραματική αξιολόγηση των υπερβαθμωτών αρχιτεκτονικών. Πιο συγκεκριμένα, γίνεται μία διεξοδική ανάλυση των σταδίων της υπερβαθμωτής διοχέτευσης του μοντέλου IVM και επιδεικνύονται τα χαρακτηριστικά και οι διάφορες πτυχές της λειτουργίας του μέσω της προσομοίωσης της εκτέλεσης διαφόρων προγραμμάτων.

1. Οργάνωση υπερβαθμωτών επεξεργαστών

Η διοχέτευση (pipeline) είναι αποδεδειγμένα η πιο επιτυχημένη μέθοδος αύξησης της απόδοσης ενός μικροεπεξεργαστή με παραλληλία επιπέδου εντολής (**I**nstruction **L**evel **P**arallelism - **ILP**). Παρόλα αυτά υπάρχει ένα πλήθος περιορισμών, που δεν επιτρέπουν την μεγιστοποίηση αυτής της αύξησης. Λόγω της δίψας που διακατέχει τους αρχιτέκτονες των υπολογιστικών συστημάτων για συνεχή βελτίωση της απόδοσης, έπρεπε αυτοί οι περιορισμοί να αρθούν. Τη λύση την έδωσε η υπερβαθμωτή (superscalar) διοχέτευση.

Οι υπερβαθμωτοί επεξεργαστές μπορούν να διαχειρίζονται πολλές εντολές ταυτόχρονα σε κάθε στάδιο της διοχέτευσης. Περιλαμβάνουν πολλές μονάδες εκτέλεσης με σκοπό την μεγιστοποίηση της ταυτόχρονης επεξεργασίας πολλών εντολών και την αύξηση της διεκπεραιωτικής ικανότητας (throughput) των εντολών. Επίσης, ένα μεγάλο κέρδος είναι ότι οι εντολές μπορούν να εκτελούνται δυναμικά και εκτός σειράς (out-of-order), δηλαδή σε διαφορετική σειρά από αυτή που καθορίζεται από το πρόγραμμα.

1.1. Περιορισμοί των βαθμωτών διοχετεύσεων

Το κύρια χαρακτηριστικά των βαθμωτών διοχετεύσεων είναι ότι η διοχέτευση αποτελείται από k σταδία και σε κάθε στάδιο μπορεί να εκτελείται μία μόνο εντολή. Επίσης, κάθε εντολή μένει σε κάθε στάδιο του pipeline για έναν κύκλο ρολογιού. Σε τέτοιου είδους επεξεργαστές υπάρχουν οι ακόλουθοι τρεις περιορισμοί:

1. Η μέγιστη διαπερατότητα είναι μία εντολή ανά κύκλο.
2. Η επεξεργασία όλων των τύπων των εντολών στο ίδιο pipeline κάνει τη σχεδίαση αναποτελεσματική.
3. Το stall σε μία διοχέτευση προκαλεί ανούσια καθυστέρηση στις εντολές που ακολουθούν αλλά και αδρανοποίηση σε όλα τα επόμενα στάδια της.

1.1.1. Άνω όριο στη διεκπεραιωτική ικανότητα των βαθμωτών τεχνικών διοχέτευσης

Η απόδοση ενός επεξεργαστή δίνεται από τον ακόλουθο τύπο:

Απόδοση = (1/αριθμός εντολών)*(εντολές/κύκλο)*(1/διάρκεια κύκλου) = ((IPC*συχνότητα)/αριθμός εντολών)

Με βάση τον παραπάνω τύπο, η απόδοση του επεξεργαστή μπορεί να αυξηθεί κάνοντας τον επεξεργαστή να εκτελεί περισσότερες εντολές ανά κύκλο ή αυξάνοντας τη συχνότητα ή μειώνοντας τον αριθμό των εντολών.

Η συχνότητα μπορεί να αυξηθεί αυξάνοντας το βάθος της διοχέτευσης, δηλαδή αυξάνοντας τον αριθμό των σταδίων της. Μία διοχέτευση, όσο περισσότερα στάδια έχει, τόσο λιγότερη λογική διαθέτει σε κάθε στάδιο. Ως εκ τούτου, ο κύκλος ρολογιού έχει μικρότερη διάρκεια και άρα η συχνότητα είναι μεγαλύτερη.

Παρόλα αυτά, μία διοχέτευση με πολλά στάδια αυξάνει τους κύκλους καθυστέρησης λόγω πιθανών εξαρτήσεων των εντολών. Για παράδειγμα, έστω ότι ένας τελεστέος, μιας εντολής ADD, εξαρτάται από μία εντολή LOAD. Έστω επίσης, ότι το στάδιο MEM, στο οποίο εκτελείται η εντολή LOAD, έχει διάρκεια τρεις κύκλους ρολογιού. Τότε η καθυστέρηση που προκύπτει για την ADD είναι τρεις κύκλοι. Στην περίπτωση που οι δύο εντολές εκτελούνταν σε έναν επεξεργαστή που υποστήριζε μία διοχέτευση με μικρότερο βάθος, όπου το στάδιο MEM θα διαρκούσε έναν κύκλο, τότε η εντολή ADD θα καθυστερούσε 2 κύκλους λιγότερο. Επομένως, σε μία διοχέτευση με πολλά στάδια, όπου η διάρκεια του κύκλου ρολογιού είναι σχετικά μικρή υπάρχει σοβαρή πιθανότητα να ακυρωθεί το κέρδος αυτό εξαιτίας των επιπλέον κύκλων καθυστέρησης που προκύπτουν σε περίπτωση εξαρτήσεων των εντολών.

Για να είναι το IPC (**I**nstructions **P**er **C**ycle – **Ε**ντολές **Α**νά **Κ**ύκλο) μεγαλύτερο από 1, θα πρέπει ένας επεξεργαστής με διοχέτευση, να έχει τη δυνατότητα να επεξεργάζεται περισσότερες από μία εντολές

σε κάθε κύκλο ρολογιού. Αυτό επιβάλλει την αύξηση του πλάτους της διοχέτευσης. Οι διοχετεύσεις που έχουν αυτό το χαρακτηριστικό ονομάζονται **παράλληλες διοχετεύσεις** (parallel pipelines).

1.1.2. Αναποτελεσματική ενοποίηση όλων των λειτουργιών στη διοχέτευση

Οι εντολές που περιλαμβάνονται στο αρχιτεκτονική συνόλου εντολών που υποστηρίζει ένας επεξεργαστής είναι διαφόρων τύπων και κάθε τύπος εντολής απαιτεί διαφορετικό είδος επεξεργασίας. Στις κλασσικές, βαθμωτές τεχνικές διοχέτευσης, όλες οι εντολές, ανεξαρτήτως τύπου, εκτελούνται στην ίδια διοχέτευση. Αυτό όμως μπορεί να προκαλέσει σημαντικές δυσκολίες και να κάνει τη σχεδίαση αναποτελεσματική.

Τα πρώτα στάδια σε μία διοχέτευση παρουσιάζουν μεγάλη ομοιομορφία. Δηλαδή, σχεδόν για κάθε τύπο εντολής γίνονται οι ίδιες λειτουργίες. Αντίθετα στα στάδια εκτέλεσης (δηλαδή στα στάδια όπου εκτελούνται οι αριθμητικές εντολές ή στα στάδια όπου εκτελούνται οι λειτουργίες της μνήμης), παρουσιάζεται εξαιρετική ανομοιομορφία. Για παράδειγμα μία εντολή πολλαπλασιασμού αριθμών κινητής υποδιαστολής μπορεί να διαρκεί περισσότερους κύκλους από μία εντολή πρόσθεσης ακέραιων αριθμών.

Συνεπώς, οι εντολές που προκαλούν μεγάλες και πολλές φορές μεταβλητού μήκους καθυστερήσεις είναι ασύμφορο να εκτελούνται στην ίδια διοχέτευση με εντολές που απαιτούν μόνο ένα κύκλο για την εκτέλεσή τους. Ένα ακόμη πρόβλημα είναι η διεύρυνση του χάσματος μεταξύ ταχύτητας επεξεργαστή και μνήμης. Αυτό έχει σαν συνέπεια, ο λανθάνων χρόνος εκτέλεσης των εντολών μνήμης να μεγαλώνει.

Εκτός όμως από τις διακυμάνσεις στο λανθάνοντα χρόνο εκτέλεσης των διαφόρων εντολών υπάρχουν και διαφορές στους πόρους που απαιτούνται από τον κάθε τύπο εντολής. Λόγω των αυξανόμενων απαιτήσεων για γρηγορότερο υλικό, ολοένα και πιο εξειδικευμένες μονάδες ενσωματώνονται στο κύκλωμα του επεξεργαστή για να εκτελούν γρήγορα πολύπλοκες εντολές.

Όλα τα παραπάνω καθιστούν ακόμα πιο αναγκαία την ανεξαρτητοποίηση των λειτουργιών στα στάδια εκτέλεσης. Στις παράλληλες διοχετεύσεις το κίνητρο είναι ακόμα πιο ισχυρό και προτιμάται η υλοποίηση διαφόρων λειτουργικών μονάδων στο στάδιο εκτέλεσης. Οι παράλληλες διοχετεύσεις που περιλαμβάνουν αυτή την ιδιότητα ονομάζονται ποικιλόμορφες διοχετεύσεις.

1.1.3. Απώλεια απόδοσης λόγω της συντηρητικής λειτουργίας της βαθμωτής διοχέτευσης

Οι βαθμωτές διοχετεύσεις λειτουργούν στατικά και συντηρητικά. Όλες οι εντολές εισέρχονται στη διοχέτευση με βάση τη σειρά που είναι στο πρόγραμμα. Όταν δεν υπάρχουν καθυστερήσεις, όλες οι εντολές προωθούνται ταυτόχρονα από το στάδιο που βρίσκονται προς το επόμενο και έτσι η σειρά εκτέλεσής τους διατηρείται χωρίς να δημιουργείται κάποιο πρόβλημα.

Όταν μία εντολή καθυστερήσει σε ένα στάδιο διοχέτευσης λόγω εξαρτήσεων, τότε όλες οι εντολές που προηγούνται χρονικά συνεχίζουν την εκτέλεσή τους. Αν η εντολή καθυστερήσει στο στάδιο i, τότε όλες οι εντολές που βρίσκονται στα στάδια 1, 2, …, i – 1 και ακολουθούν, θα καθυστερήσουν και αυτές. Όλα τα στάδια i καθυστερούν μέχρι η εντολή που βρίσκεται στο στάδιο i να προωθήσει το αποτέλεσμά της.

Κατά συνέπεια, η διάδοση της καθυστέρησης στα προηγούμενα στάδια της διοχέτευσης προκαλεί ανούσια στασιμότητα. Για παράδειγμα αν μία εντολή που ακολουθεί μπορεί να συνεχίσει κανονικά την εκτέλεσή της, ανεξάρτητα από την καθυστέρηση της εντολής που προηγείται (διότι δεν υπάρχουν εξαρτήσεις) δυστυχώς δε θα επιτραπεί η προώθηση της σε επόμενο στάδιο της διοχέτευσης και θα χρειαστεί να περιμένει χωρίς κανένα λόγο.

Αντίθετα αν αυτή η επόμενη εντολή είχε τη δυνατότητα να προσπεράσει την εντολή που καθυστερεί και να συνεχίσει την εκτέλεσή της, τότε θα είχε εξαλειφθεί ένας κύκλος καθυστέρησης. Κατ' επέκταση αν πολλές εντολές, που ακολουθούν την εντολή που καθυστερεί και δεν έχουν εξαρτήσεις,

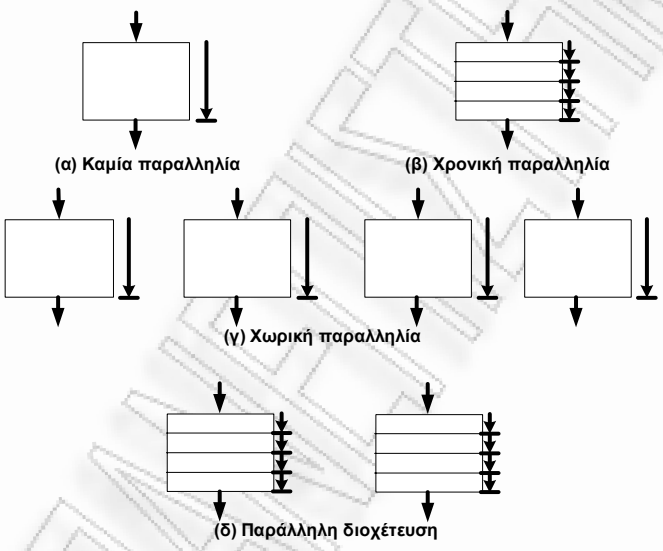
μπορούσαν να συνεχίσουν ανεξάρτητα την εκτέλεσή τους, τότε θα είχαν εξαλειφθεί πολλοί κύκλοι καθυστέρησης. Αυτή η λειτουργία ονομάζεται εκτέλεση εκτός σειράς (**O**ut-**O**f-**O**rder execution). Οι βαθμωτές διοχετεύσεις δεν την ενσωματώνουν και αυτό έχει σαν αποτέλεσμα την μείωσης της απόδοσης του επεξεργαστή. Οι παράλληλες διοχετεύσεις που υποστηρίζουν αυτή τη δυνατότητα ονομάζονται δυναμικές διοχετεύσεις (dynamic pipelines).

1.2. Το πέρασμα από τις βαθμωτές διοχετεύσεις στις υπερβαθμωτές

Οι υπερβαθμωτές διοχετεύσεις μπορούν να χαρακτηριστούν ως οι φυσικοί απόγονοι των βαθμωτών διοχετεύσεων και περιέχουν διάφορες επεκτάσεις με στόχο την άρση των τριών περιορισμών που περιγράφτηκαν στην προηγούμενη παράγραφο. Δηλαδή, οι υπερβαθμωτές διοχετεύσεις μπορούν να επεξεργαστούν πολλαπλές εντολές σε κάθε στάδιο, είναι ποικιλόμορφες διότι εμπεριέχουν πολλαπλές και ετερογενείς λειτουργικές μονάδες και είναι δυναμικές.

1.2.1. Παράλληλες διοχετεύσεις

Ο βαθμός της παραλληλίας ενός επεξεργαστή μπορεί να μετρηθεί από το μέγιστο αριθμό των εντολών που μπορεί να βρίσκονται ταυτόχρονα υπό επεξεργασία σε μία χρονική στιγμή. Ένας επεξεργαστής που αποτελείται από μία βαθμωτή, k-σταδίων διοχέτευση μπορεί να χειρίζεται k εντολές ταυτόχρονα και έτσι η απόδοσή του να είναι k φορές καλύτερη σε σχέση με έναν επεξεργαστή που δεν έχει καθόλου διοχέτευση. Εναλλακτικά, η ίδια βελτίωση στην απόδοση μπορεί να επιτευχθεί αν χρησιμοποιηθούν παράλληλα k ίδιες μηχανές χωρίς διοχέτευση. Αυτές οι δύο μορφές παραλληλίας φαίνονται και στην ακόλουθη εικόνα. Η μία, στην οποία χρησιμοποιεί η διοχέτευση, ονομάζεται χρονική παραλληλία (temporal parallelism) και η άλλη ονομάζεται χωρική παραλληλία (spatial parallelism).



Εικόνα 1.1: (α) Μία μηχανή χωρίς παραλληλία, (β) μία μηχανή με χρονική παραλληλία, (γ) μία μηχανή με χωρική παραλληλία και (δ) παράλληλη διοχέτευση

Είναι προφανές ότι μία μηχανή που υποστηρίζει χρονική παραλληλία μέσω διοχέτευσης έχει λιγότερο υλικό από μία μηχανή που υποστηρίζει τη χωρική παραλληλία. Ο λόγος είναι ότι σε μία μηχανή με χωρική παραλληλία απαιτούνται πολλαπλές, ίδιες επεξεργαστικές μονάδες. Οι παράλληλες διοχετεύσεις υποστηρίζουν και τα δύο είδη παραλληλίας (εικόνα 1.1, δ) με σκοπό την υψηλότερη διεκπεραιωτική ικανότητα.

Η απόδοσης που προκύπτει από την παράλληλη διοχέτευση μετράται σε σύγκριση με την απόδοση ενός επεξεργαστή που υποστηρίζει τη βαθμωτή διοχέτευση. Κύριο στοιχείο της απόδοσης μιας παράλληλης διοχέτευσης είναι το πλάτος της (width). Μια παράλληλη διοχέτευση με πλάτος s μπορεί να επεξεργαστεί ταυτόχρονα s εντολές σε κάθε στάδιο της. Αυτό έχει σαν αποτέλεσμα την αύξηση της απόδοσής της, σε σχέση με τη βαθμωτή διοχέτευση, κατά ένα παράγοντα s.

Για να μπορέσει να υλοποιηθεί μια παράλληλη διοχέτευση απαιτείται σημαντική προσθήκη υλικού. Για να έχει κάθε στάδιο τη δυνατότητα να επεξεργάζεται s εντολές ταυτόχρονα, θα πρέπει η πολυπλοκότητα της λογικής κάθε σταδίου να αυξηθεί κατά ένα παράγοντα s. Για παράδειγμα, οι προσωρινοί αποθηκευτικοί χώροι κάθε σταδίου θα πρέπει να είναι s για να μπορούν να αποθηκεύουν όλες τις εντολές. Στη χειρότερη περίπτωση, η λογική που απαιτείται για την πλήρη διασύνδεση μεταξύ δύο συνεχόμενων σταδίων πρέπει να αυξηθεί κατά έναν παράγοντα s^2 με σκοπό την πλήρη διασύνδεση όλων των αποθηκευτικών χώρων των δύο σταδίων. Επίσης για να υποστηριχθεί η παράλληλη πρόσβαση σε ένα αρχείο καταχωρητών θα πρέπει οι θύρες ανάγνωσης και εγγραφής του αρχείου να αυξηθούν κατά ένα παράγοντα s. Τα ίδια ισχύουν και για τις κρυφές μνήμες εντολών και δεδομένων (I-cache και D-cache).

1.2.2. Ποικιλόμορφες (diversified) διοχετεύσεις

Οι λειτουργικές μονάδες που απαιτούνται για να υποστηρίξουν την εκτέλεση διαφόρων τύπων εντολών μπορεί να διαφέρουν σημαντικά. Έστω ότι σε μία βαθμωτή διοχέτευση, το στάδιο εκτέλεσης των εντολών (EX) αποτελείται από πολλά στάδια, για να μπορέσει να υποστηρίξει την εκτέλεση όλων των ειδών των εντολών. Επομένως, όλες οι εντολές, ανεξαρτήτως τύπου, θα πρέπει να περάσουν από όλα τα στάδια του EX για να εκτελεστούν. Αυτό έχει σαν αποτέλεσμα, ο λανθάνων χρόνος εκτέλεσης για κάθε εντολή να είναι ίσος με τον αριθμό των σταδίων του EX.

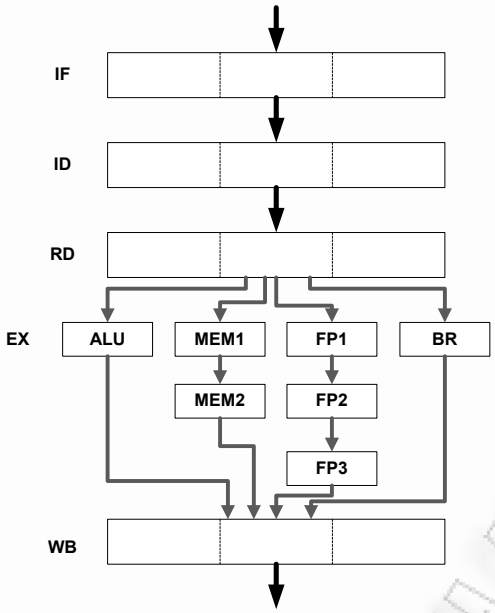
Όμως, το συγκεκριμένο είδος σχεδίασης είναι εξαιρετικά αναποτελεσματικό, διότι μία απλή εντολή, για παράδειγμα μία πρόσθεση μεταξύ δύο ακεραίων, δε χρειάζεται να περάσει από όλα τα στάδια για να εκτελεστεί με επιτυχία. Αυτό το μειονέκτημα είναι ιδιαίτερα εμφανές στις παράλληλες διοχετεύσεις, αφού στα στάδια εκτέλεσης περιλαμβάνονται πολλαπλές, διαφόρων ειδών, λειτουργικές μονάδες.

Σε μία παράλληλη διοχέτευση με πλάτος s, είναι προτιμότερο, στο κομμάτι που αφορά την εκτέλεση των εντολών, να υλοποιηθούν ποικιλόμορφες διοχετεύσεις αντί να υλοποιηθούν s ίδιες διοχετεύσεις, Μία ποικιλόμορφη διοχέτευση 4 σταδίων φαίνεται στο σχήμα 1.2. Στο στάδιο EX, είναι υλοποιημένες τέσσερις διοχετεύσεις, όπου κάθε μία προορίζεται για συγκεκριμένο είδος εντολής και αποτελείται από διαφορετικό αριθμό σταδίων. Συγκεκριμένα (από αριστερά προς τα δεξιά), περιλαμβάνονται: Μία μονάδα εκτέλεσης απλών αριθμητικών εντολών, η οποία έχει ένα στάδιο, μία μονάδα εκτέλεσης εντολών μνήμης, η οποία αποτελείται από 2 στάδια, μία μονάδα εκτέλεσης εντολών κινητής υποδιαστολής, η οποία αποτελείται από τρία στάδια και μία μονάδα επίλυσης διακλαδώσεων, η οποία αποτελείται από 1 στάδιο. Το στάδιο RD διεκπεραιώνει κάθε εντολή στην κατάλληλη διοχέτευση εκτέλεσης, ανάλογα με τον τύπο της.

Τα πλεονεκτήματα των ποικιλόμορφων διοχετεύσεων είναι τα εξής: Κάθε διοχέτευση στο στάδιο εκτέλεσης μπορεί να προσαρμοστεί καταλλήλως για ένα συγκεκριμένο τύπο εντολής οπότε η σχεδίαση γίνεται περισσότερο αποτελεσματική. Ο λανθάνων χρόνος για κάθε εντολή είναι διαφορετικός και είναι αυτός που απαιτείται για να εκτελεστεί σωστά. Επιπρόσθετα, επιτρέπεται η κατανεμημένη και ανεξάρτητη διαχείριση κάθε διοχέτευσης.

Για να σχεδιαστεί σωστά μία ποικιλόμορφη διοχέτευση θα πρέπει να ληφθούν υπόψη κάποιοι περιορισμοί. Πρώτα απ' όλα θα πρέπει να προσεχθεί ο αριθμός και ο συνδυασμός των λειτουργικών μονάδων. Στην ιδανικότερη περίπτωση, ο αριθμός των λειτουργικών μονάδων θα πρέπει να είναι ίσος με τη διαθέσιμη παραλληλία επιπέδου εντολής. Για παράδειγμα οι υπερβαθμωτοί επεξεργαστές πρώτης γενιάς, απλά ενσωμάτωναν μία δεύτερη διοχέτευση στο στάδιο εκτέλεσης για την επεξεργασία των εντολών κινητής υποδιαστολής. Ο λόγος ήταν ότι οι υπερβαθμωτοί επεξεργαστές πρώτης γενιάς έστελναν σε κάθε κύκλο ρολογιού το πολύ δύο εντολές προς τις διοχετεύσεις του σταδίου εκτέλεσης. Όταν όμως η σχεδίαση των υπερβαθμωτών επεξεργαστών εξελίχθηκε και ο αριθμός των εντολών που

αποστέλλονταν διπλασιάστηκε, τότε οι διοχετεύσεις στο στάδιο εκτέλεσης διπλασιάστηκαν και αυτές. Μερικές πρόσφατες σχεδιάσεις περιλαμβάνουν ακόμα περισσότερες διοχετεύσεις στο στάδιο εκτέλεσης.



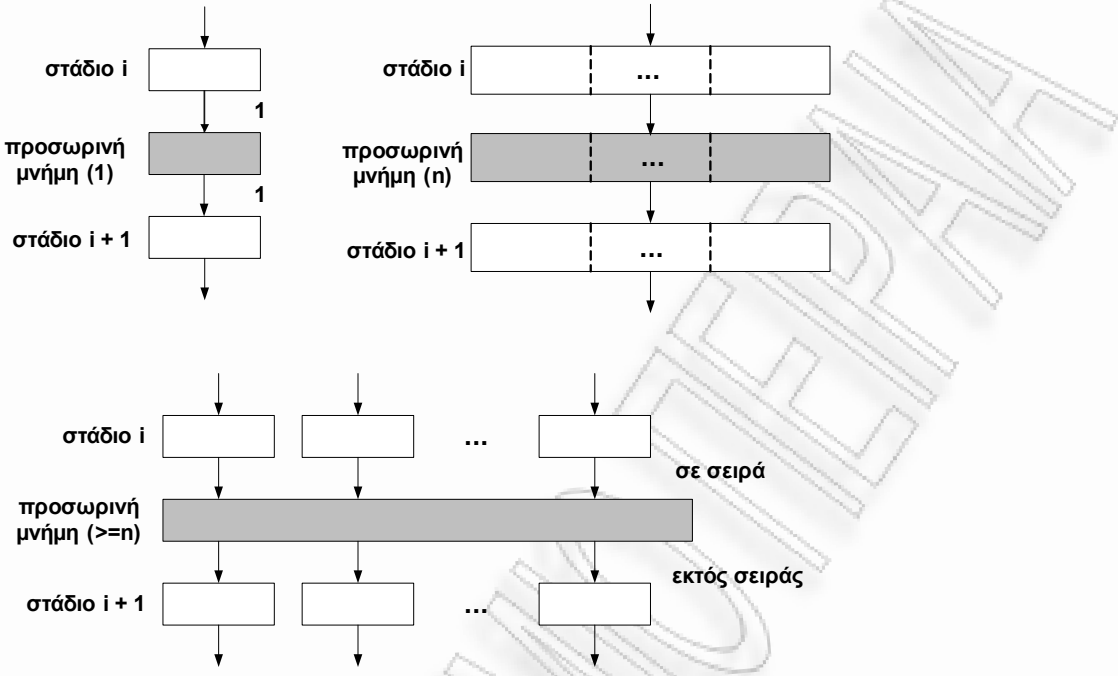
Εικόνα 1.2: Μία ποικιλόμορφη διοχέτευση με τέσσερις διοχετεύσεις στο στάδιο εκτέλεσης

1.2.3. Δυναμικές διοχετεύσεις

Σε μία διοχέτευση απαιτούνται μνήμες προσωρινής αποθήκευσης μεταξύ των σταδίων. Σε μία βαθμωτή, διοχέτευση τοποθετείται μία προσωρινή μνήμη (buffer) μιας καταχώρησης (εικόνα 1.3, α) μεταξύ δύο συνεχόμενων σταδίων (ας υποτεθεί ότι είναι τα στάδια k και k+1). Η προσωρινή μνήμη κρατάει αποθηκευμένα όλα τα απαραίτητα bit ελέγχου και δεδομένων της εντολής που βρίσκεται στο στάδιο k και θα μεταβεί στο στάδιο k+1 στον επόμενο κύκλο ρολογιού. Σε κάθε κύκλο το περιεχόμενο της προσωρινής μνήμης χρησιμοποιείται σαν είσοδος στο στάδιο k+1 και στο τέλος του κύκλου αποθηκεύεται σε αυτή η κατάλληλη πληροφορία που παράχθηκε από το στάδιο k. Κατ' αυτόν τον τρόπο, όλες οι εντολές εισέρχονται και εξέρχονται από τις προσωρινές μνήμες της διοχέτευσης με βάση τη σειρά στην οποία ορίζονται στο πρόγραμμα. Η περίπτωση όπου τα δεδομένα των προσωρινών μνημών δε προωθούνται στα επόμενα στάδια είναι όταν οι εντολές πρέπει να καθυστερήσουν εξαιτίας κινδύνων (hazards).

Σε μία παράλληλη διοχέτευση χρειάζονται μεταξύ δύο συνεχόμενων σταδίων μνήμες προσωρινής αποθήκευσης πολλαπλών καταχωρήσεων (εικόνα 1.3, β) έτσι ώστε οι πολλαπλές εντολές να μπορούν να αποθηκεύονται στο τέλος κάθε κύκλου ρολογιού και να μεταφέρονται, στην αρχή του επόμενου κύκλου, στο επόμενο στάδιο. Αν απαιτείται όλες οι εντολές που βρίσκονται σε όλα τα στάδια της παράλληλης διοχέτευσης να μεταβαίνουν ταυτόχρονα στο επόμενο στάδιο, τότε η λειτουργία των προσωρινών μνημών πολλαπλών καταχωρίσεων είναι ίδια με τη λειτουργία των προσωρινών μνημών μίας καταχώρησης. Δηλαδή, είτε θα προωθούν τις εντολές στο επόμενο στάδιο, στην αρχή κάθε κύκλου, είτε θα τις κρατούν λόγω κινδύνων.

Παρόλα αυτά, ο συγκεκριμένος τρόπος λειτουργίας των προσωρινών μνημών πολλαπλών καταχωρήσεων μπορεί να προκαλέσει ανούσια καθυστέρηση σε κάποιες εντολές, οι οποίες δεν έχουν κανένα λόγο να σταματήσουν την εκτέλεσή τους. Συνεπώς, για την αποτελεσματικότερη λειτουργία μιας παράλληλης διοχέτευσης χρειάζονται πιο πολύπλοκες προσωρινές μνήμες.

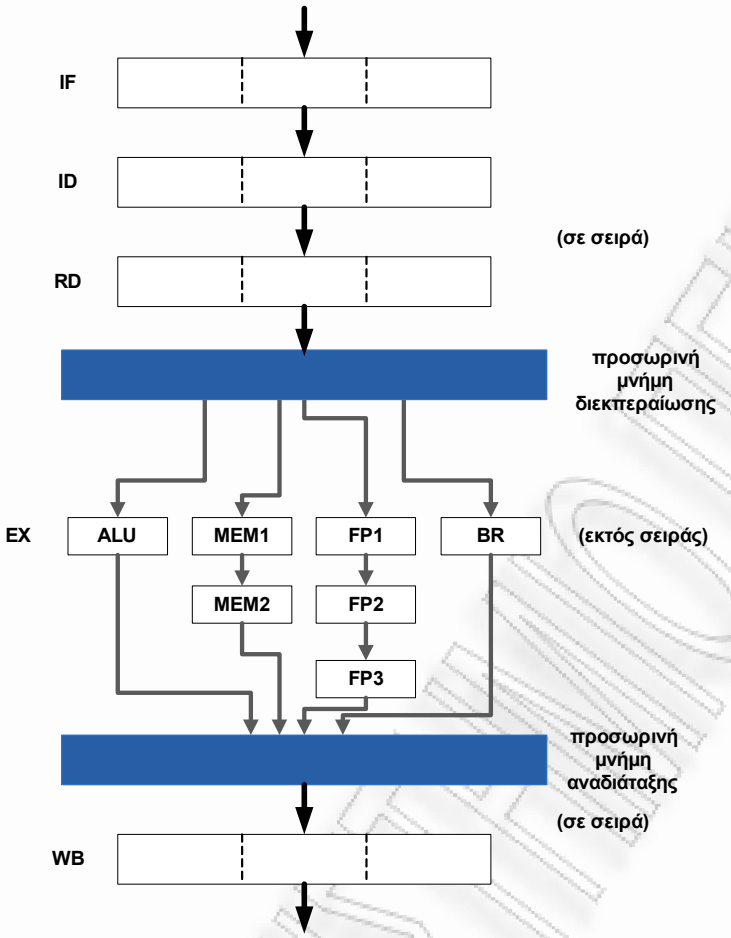


Εικόνα 1.3: Πάνω: (α) Αριστερά μία προσωρινή μνήμη μιας εισόδου, (β) δεξιά μία προσωρινή μνήμη πολλαπλών εισόδων. Κάτω: (γ) Μία προσωρινή μνήμη πολλαπλών εισόδων με δυνατότητα ανακατάταξης των εντολών

Κάθε καταχώριση σε μία μνήμη προσωρινής μνήμη διαθέτει μία θύρα εγγραφής και μία θύρα ανάγνωσης. Επίσης δεν υπάρχει αλληλεπίδραση μεταξύ των καταχωρίσεων. Μία βελτίωση σε αυτό την απλή προσωρινή μνήμη θα ήταν να συνδεθούν οι είσοδοι μεταξύ τους έτσι ώστε να προστεθεί η δυνατότητα διακίνησης των δεδομένων. Για παράδειγμα θα μπορούσαν οι είσοδοι να συνδέονται σε μία γραμμική αλυσίδα, η οποία θα λειτουργούσε σαν ουρά FIFO. Μια άλλη βελτίωση θα μπορούσε να ήταν η ανεξάρτητη προσπέλαση των καταχωρίσεων. Αυτό βέβαια απαιτεί την αποκλειστική τους διευθυνσιοδότηση και ανεξάρτητο έλεγχο για εγγραφή και ανάγνωση σε κάθε καταχώριση. Με άλλα λόγια, η βελτίωση αυτή απαιτεί τη μετατροπή της προσωρινής μνήμης σε μία μνήμη RAM πολλών θυρών. Επιπλέον, αντί να χρησιμοποιηθεί ο συμβατικός τρόπος διευθυνσιοδότησης και προσπέλασης των καταχωρίσεων, θα μπορούσε το περιεχόμενο κάθε καταχώρισης να αποτελέσει την ετικέτα (tag) με την οποία θα δεικτοδοτείται. Έτσι, με αυτόν το μηχανισμό πρόσβασης η προσωρινή μνήμη πολλαπλών καταχωρίσεων μετατρέπεται στην ουσία σε μία μικρή συσχετιστική κρυφή μνήμη.

Οι υπερβαθμωτές διοχετεύσεις διαφέρουν από τις βαθμωτές διοχετεύσεις σε ένα βασικό σημείο, το οποίο είναι η χρήση πολύπλοκων προσωρινών μνημών για να κρατάνε τις εντολές που βρίσκονται υπό επεξεργασία. Σε μία παράλληλη διοχέτευση, για να ελαχιστοποιηθεί το ποσοστό των ανούσιων καθυστερήσεων, θα πρέπει να επιτρέπεται στις εντολές που ακολουθούν μια εντολή που καθυστερεί να την παρακάμπτουν και να συνεχίζουν κανονικά την εκτέλεσή τους. Αυτό έχει σαν αποτέλεσμα να αλλάζει η σειρά εκτέλεσης των εντολών, η οποία ορίζεται αρχικά από το πρόγραμμα. Με την εκτέλεση εκτός σειράς υπάρχει αρκετά καλή πιθανότητα να προσεγγιστεί το όριο ροής των δεδομένων διότι οι εντολές εκτελούνται αμέσως μόλις γίνουν διαθέσιμοι οι τελεστέοι τους. Μία παράλληλη διοχέτευση, η οποία υποστηρίζει την εκτέλεση των εντολών εκτός σειράς ονομάζεται δυναμική διοχέτευση (dynamic pipeline).

Μία διοχέτευση επιτυγχάνει την εκτέλεση των εντολών εκτός σειράς χρησιμοποιώντας πολύπλοκες προσωρινές μνήμες, πολλαπλών καταχωρίσεων. Οι μνήμες αυτές επιτρέπουν την είσοδο και την έξοδο των εντολών μεμονωμένα και σε διαφορετική σειρά (εικόνα 1.3, γ). Η εικόνα 1.4 δείχνει μία παράλληλη, ποικιλόμορφη διοχέτευση, πλάτους s=3, η οποία είναι επίσης και δυναμική.



Εικόνα 1.4: Μία δυναμική διοχέτευση πλάτους s=3

Το κομμάτι της διοχέτευσης που αφορά την εκτέλεση των εντολών (στάδιο EX), αποτελείται από τέσσερις λειτουργικές μονάδες με διοχέτευση. Πριν και μετά το στάδιο EX βρίσκονται δύο προσωρινές μνήμες, οι οποίες αναδιατάσσουν τις εντολές. Η πρώτη προσωρινή μνήμη, η οποία βρίσκεται πριν το στάδιο EX, ονομάζεται προσωρινή μνήμη διεκπεραίωσης (dispatch buffer) και αποθηκεύει τις αποκωδικοποιημένες εντολές σε σειρά. Όμως, η προσωρινή μνήμη αυτή έχει τη δυνατότητα να αποστέλλει τις εντολές στις λειτουργικές μονάδες με διαφορετική σειρά από αυτή με την οποία εισήχθησαν.

Λόγω της προώθησης εκτός σειράς των εντολών και των διαφορετικών λανθανόντων χρόνων των λειτουργικών μονάδων, μία εντολή μπορεί να τελειώσει επίσης εκτός σειράς. Για να διασφαλιστεί η συνέπεια της εκτέλεσης (όπως για παράδειγμα οι εξαιρέσεις να προκαλούνται και να διαχειρίζονται με βάση τη σειρά που ορίζεται από το πρόγραμμα), θα πρέπει οι εντολές να ολοκληρωθούν (completed - δηλαδή να αλλάξουν την κατάσταση της μηχανής) σε σειρά. Όταν οι εντολές τελειώσουν την εκτέλεσή τους εκτός σειράς, χρειάζεται μία άλλη προσωρινή μνήμη για να διασφαλίσει την ολοκλήρωση των εντολών στη σωστή σειρά. Συνεπώς, μία δυναμική διοχέτευση επιτυγχάνει την εκτός σειράς εκτέλεση των εντολών, για να μειώσει το συνολικό χρόνο εκτέλεσης του προγράμματος, αλλά είναι επίσης ικανή να διατηρεί τη συνέπεια όσον αφορά την αλλαγή στην κατάσταση της μηχανής και την αλλαγή στην κατάσταση της μνήμης.

2. Εξαρτήσεις μνήμης

Οι υπερβαθμωτοί επεξεργαστές, έχοντας σαν στόχο την αύξηση της απόδοσης, κάνουν χρήση των τεχνικών της εικασίας (speculation) και της εκτέλεσης εκτός σειράς έτσι ώστε να επιτύχουν τη μέγιστη παραλληλία επιπέδου εντολής και το μέγιστο όριο ροής των δεδομένων. Όμως αυτές οι τεχνικές απαιτούν ιδιαίτερη διαχείριση όταν υπάρχουν εξαρτήσεις μεταξύ των εντολών ή όταν γίνει κάποια λανθασμένη πρόβλεψη. Στη δεύτερη περίπτωση ο επεξεργαστής θα πρέπει να επανέλθει στη σωστή κατάσταση και να εκτελέσει τη σωστή ροή εντολών. Στην ενότητα αυτή γίνεται αναφορά στις εξαρτήσεις μνήμης, την αρνητική επίδρασή τους στη συνολική απόδοση του επεξεργαστή και πώς μπορεί να γίνει η πρόβλεψή τους έτσι ώστε η επίδραση αυτή να είναι όσο το δυνατόν μικρότερη.

Δύο εντολές έχουν εξάρτηση μνήμης όταν μία εντολή αποθήκευσης και μία εντολή φόρτωσης κάνουν αναφορά στην ίδια διεύθυνση. Πιο συγκεκριμένα, η εντολή φόρτωσης ψευδωνυμεί (alias) με την εντολή αποθήκευσης. Αυτό σημαίνει ότι μία εντολή φόρτωσης (αποθήκευσης) διαβάζει (γράφει) από (σε) μία διεύθυνση μνήμης στην (από την) οποία γράφει (διαβάζει) μία εντολή αποθήκευσης (φόρτωσης) που προηγείται χρονικά στο πρόγραμμα. Γενικά για τις εξαρτήσεις μνήμης ισχύουν τα ίδια με τις εξαρτήσεις μεταξύ των δεδομένων των καταχωρητών (RAW, WAR, WAW και RAR αν και η τελευταία στην ουσία δεν υφίσταται). Δύο εντολές αποθήκευσης που γράφουν στην ίδια περιοχή έχουν εξάρτηση WAW μεταξύ τους. Σε αυτού του είδους την εξάρτηση πρέπει να δοθεί μεγάλη προσοχή έτσι ώστε να διατηρηθεί η συνέπεια της μνήμης.

Ένας απλός τρόπος για να επιλυθούν οι εξαρτήσεις μνήμης είναι να εκτελεστούν όλες οι εντολές μνήμης με τη σειρά που ορίζονται στο πρόγραμμα. Όμως ο τρόπος αυτός είναι συντηρητικός και μπορεί να θέσει ανούσια όρια στην απόδοση του επεξεργαστή. Το ακόλουθο παράδειγμα [1] το αποδεικνύει.

```
for (i=0; i<64; i++)
      Y(i) = A*X(i) + Y(i);

F0 ← LD,a
R4 ← ADDI,Rx,#512       ;last address

Loop:
      F2 ← LD,0(Rx)             ;load X(i)
      F2 ← MULTD,F0,F2          ;A*X(i)
      F4 ← LD,0(Ry)             ;load Y(i)
      F4 ← ADDD,F2,F4           ;A*X(i) + Y(i)
      0(Ry) ← SD,F4             ;store into Y(i)
      Rx ← ADDI,Rx,#8           ;increment index of X
      Ry ← ADDI,Ry,#8           ;increment index of Y
      R20 ← SUB,R4,Rx           ;compute loop limit
      BNZ,R20,Loop              ;check if it is OK to loop again
```

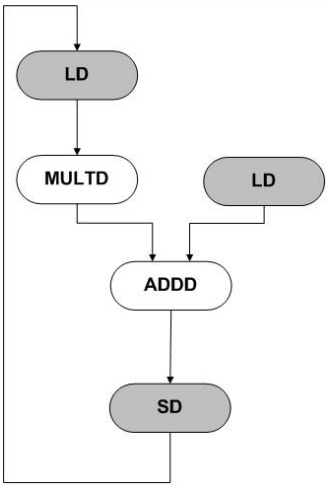
Το συγκεκριμένο κομμάτι κώδικα πολλαπλασιάζει τα στοιχεία ενός πίνακα με μία σταθερά και προσθέτει το αποτέλεσμα με τα στοιχεία ενός άλλου πίνακα. Οι επαναλήψεις του βρόχου δεν παρουσιάζουν εξαρτήσεις δεδομένων μεταξύ τους, οπότε μπορούν να εκτελεστούν παράλληλα (loop unrolling – ξεδίπλωμα βρόχου). Παρόλα αυτά, αν υιοθετηθεί ο περιορισμός της εκτέλεσης των εντολών μνήμης σε σειρά, τότε, ανά ζεύγη επαναλήψεων, η πρώτη εντολή φόρτωσης από τη δεύτερη επανάληψη θα πρέπει να περιμένει την εντολή αποθήκευσης της πρώτης επανάληψης για να εκτελεστεί και έτσι ο βρόχος δε θα μπορέσει να ξεδιπλωθεί.

Αν επιτραπεί η εκτέλεση εκτός σειράς των εντολών φόρτωσης και αποθήκευσης, χωρίς βέβαια να παραβιάζονται οι εξαρτήσεις μνήμης, τότε μπορεί να σημειωθεί μεγάλη αύξηση στην απόδοση.

Ο γράφος στην εικόνα 2.1 αναπαριστά τις αληθινές εξαρτήσεις δεδομένων που υπάρχουν μεταξύ των εντολών μιας επανάληψης του βρόχου και είναι προφανές ότι δεν υπάρχουν εξαρτήσεις μεταξύ των επαναλήψεων. Η εντολή διακλάδωσης που είναι τελευταία είναι σε μεγάλο βαθμό προβλέψιμη και ως εκ τούτου η προσκόμιση των εντολών, των επαναλήψεων που ακολουθούν, μπορεί να γίνει πολύ γρήγορα. Επίσης οποιεσδήποτε εξαρτήσεις καταχωρητών μπορούν να επιλυθούν εύκολα λόγω της δυναμικής επαναχρησιμοποίησης αυτών. Συνεπώς, αν οι εντολές μνήμης μπορούσαν να εκτελεστούν εκτός σειράς

τότε οι εντολές φόρτωσης μιας επανάληψης που ακολουθεί θα μπορούσαν να εκτελεστούν νωρίτερα από την εντολή αποθήκευσης μιας επανάληψης που προηγείται χρονικά και έτσι θα σημειωνόταν αύξηση στην απόδοση του επεξεργαστή.

Όμως τα μοντέλα μνήμης θέτουν κάποιους περιορισμούς στην εκτός σειράς εκτέλεση των εντολών φόρτωσης και αποθήκευσης. Πρώτον, για να λειτουργεί σωστά ο μηχανισμός των εξαιρέσεων θα πρέπει να αλλάζει με τη σωστή σειρά η κατάσταση της μνήμης. Το ίδιο ισχύει και στα πολυπήρυνα συστήματα, στα οποία πρέπει να διατηρείται η συνέπεια της μνήμης. Όπως είναι γνωστό, οι εντολές που αλλάζουν την κατάσταση είναι οι εντολές αποθήκευσης. Επομένως, οι εντολές αυτές θα πρέπει να εκτελούνται σε σειρά. Με βάση αυτόν τον περιορισμό δε υπάρχει λόγος ανησυχίας για τις εξαρτήσεις μνήμης WAW και WAR. Συνεπώς θα πρέπει να γίνεται μέριμνα μόνο για τις εξαρτήσεις μνήμης RAW.



Εικόνα 2.1: Οι εξαρτήσεις δεδομένων σε μια επανάληψη του κώδικα DAXPY

2.1. Πρόβλεψη εξαρτήσεων μνήμης με χρήση των συνόλων αποθήκευσης (store sets)

[6] Με στόχο τη μέγιστη απόδοση, ένας υπερβαθμωτός επεξεργαστής πρέπει να ξεκινάει όσο το δυνατόν νωρίτερα τις εντολές φόρτωσης ενώ παράλληλα πρέπει να αποφεύγονται οι παραβιάσεις της διάταξης μνήμης με τις εντολές αποθήκευσης που προηγούνται χρονικά και γράφουν στην ίδια διεύθυνση. Μία μέθοδος για να επιτευχθεί αυτό είναι να προβλέπονται οι εντολές αποθήκευσης από τις οποίες εξαρτάται μία εντολή φόρτωσης και να μεταδίδεται η πληροφορία αυτή στο διεκπεραιωτή εντολών έτσι ώστε να είναι σε θέση να διεκπεραιώνει ή όχι τις εντολές φόρτωσης.

Το σύνολο των εντολών αποθήκευσης από το οποίο εξαρτάται μία εντολή φόρτωσης ονομάζεται σύνολο αποθήκευσης (store set). Ο επεξεργαστής μπορεί να ανακαλύπτει και να χρησιμοποιεί ένα σύνολο αποθήκευσης, για να προβλέπει με ακρίβεια το νωρίτερο χρόνο κατά τον οποίο μπορεί μια εντολή φόρτωσης να εκτελεστεί.

Όσον αφορά όμως τις εξαρτήσεις μνήμης υπάρχει ένα πρόβλημα. Ενώ οι εξαρτήσεις καταχωρητών μπορούν να καθοριστούν είτε στο στάδιο αποκωδικοποίησης των εντολών, είτε στο στάδιο μετονομασίας, οι εξαρτήσεις μνήμης δε γίνεται να είναι γνωστές τόσο νωρίς ή τουλάχιστον νωρίτερα από το στάδιο δρομολόγησης των εντολών. Επομένως αν μία εντολή φόρτωσης ξεκινήσει και εκτελεστεί νωρίτερα από μία εντολή αποθήκευσης που προηγείται χρονικά και από την οποία εξαρτάται, τότε η εντολή φόρτωσης θα φορτώσει τη λάθος τιμή. Έτσι θα πρέπει αυτή και όλες οι ακόλουθες, εξαρτώμενες εντολές να ξανά εκτελεστούν και αυτό θα έχει σαν αποτέλεσμα την επιβάρυνση της απόδοσης. Για να αποφευχθούν αυτού του είδους οι παραβιάσεις, θα μπορούσε ο διεκπεραιωτής να εκτελέσει πρώτα όλες τις εντολές αποθήκευσης που προηγούνται και ύστερα τις εντολές φόρτωσης. Όμως αποδείχτηκε στην εισαγωγή του 2^{ου} κεφαλαίου αυτή η μέθοδος επίσης μειώνει δραματικά την απόδοση. Συνεπώς το ακόλουθο δίλημμα δημιούργησε την ανάγκη για την πρόβλεψη των εξαρτήσεων μνήμης: *Είναι*

*προτιμότερο να εκτελούνται νωρίτερα οι εντολές φόρτωσης και αν υπάρχει παραβίαση των εξαρτήσεων να ακυρώνονται **ή** να ακολουθηθεί μία πιο συντηρητική προσέγγιση όπου δε θα εκτελείται καμία εντολή φόρτωσης μέχρις ότου εκτελεστούν όλες οι εντολές αποθήκευσης που προηγούνται χρονικά;*

Οι στόχοι της πρόβλεψης των εξαρτήσεων μνήμης είναι δύο:

- Να προβλέπονται οι εντολές φόρτωσης, οι οποίες αν εκτελούνταν θα γίνονταν παραβιάσεις.
- Να καθυστερεί η εκτέλεση των παραπάνω εντολών όσο χρειάζεται με σκοπό την αποφυγή των παραβιάσεων των εξαρτήσεων.

Όταν ένας μηχανισμός πρόβλεψης εξαρτήσεων μνήμης αποτυγχάνει να ικανοποιήσει έναν από τους δύο στόχους, τότε έχει αποτύχει.

2.1.1. Η βασική ιδέα των συνόλων αποθήκευσης

Η βασική ιδέα των συνόλων αποθήκευσης στηρίζεται σε δύο υποθέσεις: α) Η παρελθοντική συμπεριφορά των εξαρτήσεων μνήμης (αν παραβιάστηκαν ή όχι) αποτελεί έναν καλό οδηγό για την πρόβλεψη των μελλοντικών εξαρτήσεων. β) Είναι σημαντικό να προβλέπεται η εξάρτηση μιας εντολής φόρτωσης από πολλαπλές εντολές αποθήκευσης, όπως επίσης και η εξάρτηση πολλών εντολών φόρτωσης από μία εντολή αποθήκευσης.

Μία εντολή φόρτωσης, σε ένα πρόγραμμα, συνδέεται με ένα σύνολο από εντολές αποθήκευσης από τις οποίες εξαρτάται. Το σύνολο αυτό ονομάζεται σύνολο αποθήκευσης. Ο επεξεργαστής έχει την ευθύνη να κατασκευάσει το σύνολο αποθήκευσης μιας εντολής φόρτωσης παρακολουθώντας ποιες εντολές αποθήκευσης την έκαναν να προκαλέσει, με την εκτέλεσή της, παραβιάσεις όσον αφορά τις εξαρτήσεις μνήμης.

Όταν ένα πρόγραμμα ξεκινήσει την εκτέλεσή του, όλες οι εντολές φόρτωσης έχουν άδεια σύνολα αποθήκευσης. Όταν μία εντολή φόρτωσης και μία εντολή αποθήκευσης εκτελεστούν σε λάθος σειρά και παραβιάσουν την εξάρτηση, τότε η διεύθυνση της εντολής αποθήκευσης προστίθεται στο σύνολο αποθήκευσης της εντολής φόρτωσης, για παράδειγμα το σύνολο Α. Αν η ίδια εντολή φόρτωσης παραβιάσει την εξάρτησή της με κάποια άλλη εντολή αποθήκευσης, τότε η διεύθυνση και αυτής της εντολής αποθήκευσης προστίθεται επίσης στο σύνολο αποθήκευσης Α. Αυτό έχει σαν αποτέλεσμα, την επόμενη φορά που ο επεξεργαστής συναντήσει τη συγκεκριμένη εντολή φόρτωσης, θα απαιτήσει να εκτελεστεί εφόσον προηγηθεί η εκτέλεση κάποιας ή κάποιων από τις εντολές αποθήκευσης του συνόλου αποθήκευσης Α. Δηλαδή όταν προσκομιστεί η εντολή φόρτωσης, ο επεξεργαστής ελέγχει ποιες από τις εντολές αποθήκευσης στο σύνολο αποθήκευσης Α έχουν προσκομιστεί πιο πρόσφατα αλλά δεν έχουν ξεκινήσει και δημιουργεί εξάρτηση της εντολής φόρτωσης με αυτές. Οι εντολές φόρτωσης που δε παραβιάζουν τις εξαρτήσεις μνήμης μπορούν να εκτελεστούν όσο το δυνατόν νωρίτερα. Όμως για αυτές που παραβιάζουν τις εξαρτήσεις μνήμης δημιουργείται εξάρτηση μόνο με τις εντολές αποθήκευσης με τις οποίες έχουν εξαρτηθεί και στο παρελθόν. Τέλος, αν μία εντολή αποθήκευσης προκαλεί παραβίαση των εξαρτήσεων μνήμης με κάποια άλλη εντολή φόρτωσης, τότε προστίθεται επίσης και στο άλλο σύνολο αποθήκευσης. Οι εντολές φόρτωσης και οι εντολές αποθήκευσης αναγνωρίζονται με βάση τη διεύθυνση μνήμης τους.

Στο παράδειγμα που ακολουθεί [6]:

```
PC
0     store C
4     store A
8     store B
12    store C
28    load B store set { PC 8 }
32    load D store set { (null) }
36    load C store set { PC 0, PC 12 }
40    load B store set { PC 8 }
```

Τα σύνολα αποθήκευσης των εντολών φόρτωσης είναι σημειωμένα δεξιά. Αν παρατηρήσετε, μία εντολή φόρτωσης μπορεί να έχει εξαρτήσεις με πολλαπλές εντολές αποθήκευσης. Για παράδειγμα η εντολή φόρτωσης στη διεύθυνση μνήμης 36 εξαρτάται από τις εντολές αποθήκευσης στις διευθύνσεις 0 και 12. Επίσης πολλαπλές εντολές φόρτωσης εξαρτώνται από την ίδια εντολή αποθήκευσης. Για παράδειγμα οι

εντολές φόρτωσης στις διευθύνσεις 28 και 40 εξαρτώνται από την εντολή αποθήκευσης της διεύθυνσης 8.

Η περίπτωση όπου πολλαπλές εντολές φόρτωσης εξαρτώνται από την ίδια εντολή αποθήκευσης είναι κατανοητή και συνήθης. Για παράδειγμα, έστω ότι στον κώδικα υπάρχει μία εντολή που γράφει σε μια μεταβλητή και πολλές εντολές που διαβάζουν την τιμή της μεταβλητής αυτής. Η περίπτωση όμως όπου μία εντολή φόρτωσης εξαρτάται από πολλαπλές εντολές αποθήκευσης δεν είναι τόσο προφανής. Συμβαίνει σε τρεις περιπτώσεις: 1) Μία εντολή φόρτωσης να εξαρτάται από εντολές αποθήκευσης που βρίσκονται σε διαφορετικά μονοπάτια εντολών, για παράδειγμα *if (expr) then x=a; else x=b; ... c=x;* 2) Όταν έχουμε μία δομή στην οποία γίνονται αποθηκεύσεις σε διάφορα μέρη της αλλά η ίδια φορτώνεται ολόκληρη. 3) Στη περίπτωση των εξαρτήσεων WAW, τότε μία εντολή φόρτωσης μπορεί να εξαρτάται από ένα σύνολο εντολών αποθήκευσης που αποθηκεύουν στην ίδια περιοχή μνήμης.

2.1.2. Η υλοποίηση των συνόλων αποθήκευσης

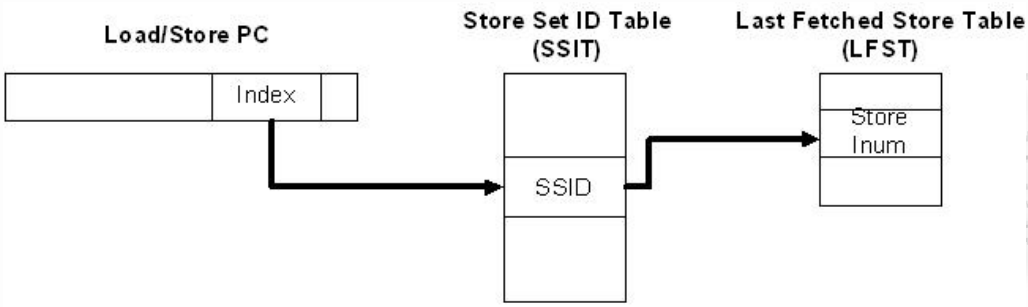
Για να υλοποιηθεί η συγκεκριμένη μέθοδος στο υλικό, σίγουρα θα πρέπει να τεθούν κάποιοι περιορισμοί. Το υλικό έχει πεπερασμένες διαστάσεις, οπότε δε μπορούν όλα να είναι ιδανικά. Περιορίζοντας το μέγεθος των δομών, αυτό σημαίνει ότι άσχετες εντολές φόρτωσης θα μοιράζονται το ίδια σύνολα αποθήκευσης με αποτέλεσμα να προκαλούνται λανθασμένες εξαρτήσεις. Επίσης η διεύθυνση μιας εντολής αποθήκευσης επιτρέπεται να είναι σε ένα μόνο σύνολο αποθήκευσης. Με αυτόν τον περιορισμό, μία εντολή φόρτωσης μπορεί να εξαρτάται από πολλαπλές εντολές αποθήκευσης αλλά πολλαπλές εντολές φόρτωσης δε μπορούν να εξαρτώνται από μία εντολή αποθήκευσης. Για να αντιμετωπιστεί αυτό το φαινόμενο χρησιμοποιήθηκε ένας αλγόριθμος συγχώνευσης, ο οποίος επιτρέπει σε δύο εντολές φόρτωσης που εξαρτώνται από την ίδια εντολή αποθήκευσης, να μοιραστούν το ίδιο σύνολο αποθήκευσης.

Αφού λοιπόν οι εντολές φόρτωσης μπορούν να εξαρτώνται από πολλαπλές εντολές αποθήκευσης, θα έπρεπε, θεωρητικά, να υπάρχει ένας μηχανισμός όπου θα καθυστερούσε την εντολή φόρτωσης μέχρις ότου εκτελεστούν όλες οι εντολές αποθήκευσης που βρίσκονται στο σύνολο αποθήκευσής της. Η κατασκευή όμως ενός τέτοιου μηχανισμού θα ήταν πολύ ακριβή. Για να επιλυθεί αυτό το πρόβλημα έπρεπε να ακολουθηθεί μία πιο συντηρητική οδός. Έτσι όλες οι εντολές αποθήκευσης σε ένα σύνολο αποθήκευσης περιορίζονται στο να εκτελούνται σε σειρά. Αυτό επιτυγχάνεται κάνοντας κάθε μία από τις εντολές αποθήκευσης να εξαρτώνται από αυτήν η οποία προσκομίστηκε τελευταία. Κάθε εντολή αποθήκευσης λοιπόν προσδιορίζει μία εξάρτηση, η εντολή φόρτωσης προσδιορίζει και αυτή την εξάρτησή της και έτσι δημιουργείται μια αλυσίδα εξαρτήσεων που συμβάλλει στη σωστή αλλαγή της κατάστασης της μνήμης και στη σωστή εκτέλεση των εντολών του προγράμματος.

Η απαίτηση της εκτέλεσης των εντολών αποθήκευσης σε σειρά περιορίζει την περίπλοκη εξάρτηση WAW. Για παράδειγμα, αν δύο συνεχόμενες εντολές αποθήκευσης, που είναι στο ίδιο σύνολο αποθήκευσης, γράφουν στην ίδια τοποθεσία, ας είναι η X, τότε η εντολή φόρτωσης έχει πραγματική εξάρτηση μόνο με τη τελευταία (από τις δύο συνεχόμενες) εντολή αποθήκευσης.

Η υλοποίηση των συνόλων αποθήκευσης αποτελείται από δύο πίνακες (εικόνα 2.2). Ο πρώτος πίνακας, ο οποίος δεικτοδοτείται από κάποια bits της διεύθυνσης της εντολής, ονομάζεται **S**tore **S**et **I**dentifier **T**able (**SSIT**) και διατηρεί τα σύνολα αποθήκευσης χρησιμοποιώντας μία κοινή ετικέτα για την εντολή φόρτωσης και τις εντολές αποθήκευσης, οι οποίες περιλαμβάνονται στο σύνολο αποθήκευσής του. Ο δεύτερος πίνακας ονομάζεται **L**ast **F**etched **S**tore **T**able (**LFST**) και διατηρεί δυναμικές πληροφορίες (έναν κωδικό - inum) για την εντολή αποθήκευσης, κάθε σύνολο αποθήκευσης, που προσκομίστηκε πιο πρόσφατα.

Οι εντολές φόρτωσης που προσκομίστηκαν πιο πρόσφατα προσπελαύνουν τον πίνακα SSIT με βάση κάποια από τα bits της διεύθυνσής τους και παίρνουν το αναγνωριστικό για το σύνολο αποθήκευσής τους (**SSID** – **S**tore **S**et **Id**entifier). Αν η εντολή φόρτωσης έχει έγκυρο SSID, τότε αυτό σημαίνει ότι έχει έγκυρο σύνολο αποθήκευσης. Επομένως, θα προσπελάσει, με βάση το SSID, τον πίνακα LFST και θα πάρει τον κωδικό για την εντολή αποθήκευσης που προσκομίστηκε πιο πρόσφατα και ανήκει στο σύνολο αποθήκευσης του.



Εικόνα 2.2: [6]Υλοποίηση του μηχανισμού των συνόλων αποθήκευσης που προβλέπει τις εξαρτήσεις μνήμης

Οι εντολές αποθήκευσης που έχουν προσκομιστεί πιο πρόσφατα προσπελάζουν και αυτές τον πίνακα SSIT. Αν μια εντολή αποθήκευσης έχει έγκυρο SSID, αυτό σημαίνει ότι ανήκει σε ένα έγκυρο σύνολο αποθήκευσης. Τότε, πρέπει να γίνουν δύο πράγματα: Πρώτα, θα πρέπει να προσπελαστεί ο πίνακας LFST (με βάση το SSID) και να εξαρτηθεί η εντολή αποθήκευσης που μόλις προσκομίστηκε με την εντολή αποθήκευσης που θα βρεθεί εκεί. Στη συνέχεια θα πρέπει να ανανεωθεί ο πίνακας LFST, εισάγοντας το νέο κωδικό της εντολής αποθήκευσης που μόλις προσκομίστηκε.

Όταν ξεκινήσει μία εντολή αποθήκευσης, προσπελάζεται ο πίνακας LFST και ακυρώνεται η σχετική καταχώριση αν εξακολουθεί να αναφέρεται σε αυτή την εντολή. Αυτό εξασφαλίζει ότι οι εντολές φόρτωσης και αποθήκευσης θα εξαρτώνται σε εντολές φόρτωσης, οι οποίες δεν έχουν ξεκινήσει.

Όταν ο επεξεργαστής επανέρχεται από μία λανθασμένη πρόβλεψη, ο πίνακας SSIT δε χρειάζεται να τροποποιηθεί. Για τον πίνακα LFST θα πρέπει, στην ιδανικότερη περίπτωση, να επαναφερθούν οι καταχωρήσεις του στους κωδικούς των τελευταίων, έγκυρων εντολών αποθήκευσης. Βέβαια πειράματα που μοντελοποίησαν τη συγκεκριμένη συμπεριφορά έδειξαν ότι δεν έχει μεγάλη διαφορά από το να ακυρωθούν απλά όλες οι καταχωρίσεις.

Για να γίνει ακόμη πιο κατανοητός ο μηχανισμός των συνόλων αποθήκευσης παρατίθεται ένα παράδειγμα. Ας υποθέσουμε ότι στην αρχή του προγράμματος, όλες οι καταχωρήσεις στον πίνακα SSIT είναι άκυρες. Αρχικά, οι εντολές φόρτωσης και αποθήκευσης προσπελαύνουν τον πίνακα SSIT και δεν παίρνουν καμία έγκυρη πληροφορία για εξαρτήσεις μνήμης. Έστω μία εντολή φόρτωσης Α και μία εντολή αποθήκευσης Β. Όταν η εντολή φόρτωσης Α διαπράξει μία παραβίαση εξάρτησης μνήμης με τη Β, δημιουργείται ένα σύνολο αποθήκευσης στον SSIT. Για τις εντολές Α και Β, που αναμίχθηκαν στην παραβίαση της εξάρτησης μνήμης, τους ανατίθεται ένα αναγνωριστικό του συνόλου αποθήκευσης (SSID) που δημιουργήθηκε, ας είναι το Χ. Τα SSID δημιουργούνται με διάφορους τρόπους, όπως για παράδειγμα με συνάρτηση κατακερματισμού (hash) στη διεύθυνση μνήμης της εντολής φόρτωσης. Το SSID Χ θα αποθηκευτεί σε δύο καταχωρήσεις του πίνακα SSIT. Η πρώτη καταχώρηση δεικτοδοτείται από τη διεύθυνση της εντολής φόρτωσης Α ενώ η δεύτερη καταχώρηση δεικτοδοτείται από τη διεύθυνση της εντολής αποθήκευσης Β. Την επόμενη φορά που θα προσκομιστεί η Β, θα προσπελαστεί η καταχώρηση του πίνακα SSIT που δεικτοδοτείται από τη διεύθυνσή της. Αφού τότε το αναγνωριστικό του συνόλου αποθήκευσης θα είναι έγκυρο (και ίσο με Χ), χρησιμοποιείται για να προσπελαστεί ο πίνακας LFST όπου δε υπάρχει καμία έγκυρη καταχώρηση. Έτσι, η εντολή αποθήκευσης Β δε θα εξαρτηθεί σε άλλη εντολή αποθήκευσης και απλά γράφεται στη σχετική καταχώρηση του LFST ο κωδικός της. Όταν προσκομιστεί η εντολή φόρτωσης Α, προσπελάζεται ο πίνακας SSIT και στη συνέχεια ο LFST με βάση το SSID της εντολής φόρτωσης που είναι το Χ. Τώρα όμως στη συγκεκριμένη καταχώρηση του LFST υπάρχει ο κωδικός της εντολής αποθήκευσης Β, οπότε ο LFST ενημερώνει το διεκπεραιωτή εντολών ότι η εντολή φόρτωσης Α εξαρτάται από την εντολή αποθήκευσης Β.

Αν αργότερα η εντολή φόρτωσης Α παραβιάσει την εξάρτηση της με μία άλλη εντολή αποθήκευσης, ας πούμε τη Γ, ενημερώνεται ο πίνακας SSIT και συγκεκριμένα το SSID Χ αποθηκεύεται στην καταχώρηση που δεικτοδοτείται από την εντολή αποθήκευσης Γ. Έτσι στον πίνακα SSIT υπάρχουν πλέον τρεις καταχωρήσεις που έχουν το SSID Χ. Την επόμενη φορά που οι δύο εντολές αποθήκευσης Β και Γ και η εντολή φόρτωσης Α θα προσκομιστούν, η εντολή αποθήκευσης Γ θα εξαρτάται από την εντολή αποθήκευσης Β και η εντολή φόρτωσης Α θα εξαρτάται από την εντολή φόρτωσης Γ.

Θα πρέπει να σημειωθεί ότι οι εντολές αποθήκευσης μπορούν να βρίσκονται μόνο σε ένα σύνολο αποθήκευσης κάθε φορά έτσι ώστε οι πίνακες SSIT και LFST να είναι σχετικά απλοί. Μία εντολή αποθήκευσης θα μπορούσε να ανήκει σε δύο τουλάχιστον σύνολα αποθήκευσης αν επιτρεπόταν στον SSIT να διατηρεί πολλαπλά SSID σε κάθε καταχώρησή του. Σε αυτή την περίπτωση όμως τα πολλαπλά SSID θα έπρεπε να προσπελάσουν τον πίνακα LFST, αυξάνοντας τον αριθμό των θυρών ανάγνωσης που απαιτούνται. Επίσης κάθε εντολή φόρτωσης θα ήταν εξαρτώμενη με τουλάχιστον δύο εντολές αποθήκευσης αυξάνοντας το μέγεθος και την πολυπλοκότητα του διεκπεραιωτή εντολών.

Οι πίνακες που φαίνονται στην εικόνα 2.2 και υλοποιούν το μηχανισμό των συνόλων αποθήκευσης επιτρέπουν σε μία εντολή αποθήκευσης να βρίσκεται μόνο σε ένα σύνολο αποθήκευσης. Αυτό προφανώς παραβιάζει την προϋπόθεση ότι πολλαπλές εντολές φόρτωσης μπορούν να εξαρτώνται σε ίδιες εντολές αποθήκευσης. Για το λόγο αυτό χρησιμοποιείται ο ακόλουθος αλγόριθμος συγχώνευσης έτσι ώστε να υποστηρίζεται αυτή η ιδιότητα.

Όταν συμβεί μία παραβίαση εξάρτησης μνήμης, ανανεώνονται οι σχετικές καταχωρήσεις του πίνακα SSIT. Όταν το SSID της εντολής φόρτωσης αλλά και της εντολής αποθήκευσης δεν είναι έγκυρα, δημιουργείται ένα καινούριο SSID και δίνεται και στις δύο εντολές. Αν το SSID της εντολής φόρτωσης είναι έγκυρο και της εντολής αποθήκευσης δεν είναι, τότε η εντολή αποθήκευσης κληρονομεί το έγκυρο SSID και ουσιαστικά συμπεριλαμβάνεται στο σύνολο αποθήκευσης της εντολής φόρτωσης. Όμως, δεν έχει εξετασθεί η περίπτωση μία εντολή αποθήκευσης να έχει ήδη ένα έγκυρο SSID. Αν απλά αντικαθιστούσαμε το SSID της εντολής αποθήκευσης με το καινούριο αυτό θα ήταν μη αποδεκτό διότι τότε μία εντολή αποθήκευσης θα μπορούσε να είναι μόνο σε ένα σύνολο αποθήκευσης και αυτό θα προκαλούσε σοβαρά προβλήματα. Για παράδειγμα, πολλαπλές εντολές φόρτωσης μπορεί να δημιουργούσαν παραβιάσεις των εξαρτήσεων μνήμης με την ίδια εντολή αποθήκευσης, και αφού η εντολή αποθήκευσης μπορεί να ανήκει μόνο σε ένα σύνολο αποθήκευσης, τότε θα υπήρχε ανταγωνισμός μεταξύ των εντολών φόρτωσης για να προσθέσουν την εντολή αποθήκευσης στο δικό τους σύνολο αποθήκευσης. Αυτή η συνθήκη ανταγωνισμού μοιάζει με το φαινόμενο που συναντάται στις κρυφές μνήμες όπου δύο εντολές φόρτωσης προκαλούν αστοχίες η μία στην άλλη (cache thrash). Οπότε για να γίνεται ορθή αντικατάσταση και μετακίνηση των εντολών αποθήκευσης στα σύνολα αποθήκευσης, έχουν καθιερωθεί συγκεκριμένοι κανόνες.

- Αν ούτε στην εντολή φόρτωσης, ούτε στην εντολή αποθήκευσης έχει ανατεθεί κάποιο σύνολο αποθήκευσης τότε δεσμεύεται ένα καινούργιο και δίνεται και στις δύο.
- Αν η εντολή φόρτωσης έχει ήδη κάποιο έγκυρο σύνολο αποθήκευσης αλλά η εντολή αποθήκευσης δεν ανήκει σε κάποιο, τότε η εντολή αποθήκευσης συμπεριλαμβάνεται στο σύνολο αποθήκευσης της εντολής φόρτωσης.
- Αν η εντολή αποθήκευσης είναι σε κάποιο σύνολο αποθήκευσης αλλά η εντολή φόρτωσης όχι, τότε στην εντολή φόρτωσης ανατίθεται το σύνολο αποθήκευσης της εντολής αποθήκευσης.
- Αν στις εντολές φόρτωσης και αποθήκευσης έχει ήδη ανατεθεί κάποιο έγκυρο σύνολο αποθήκευσης (σε διαφορετικά το καθένα) τότε ένα από τα δύο σύνολα αποθήκευσης ορίζεται ως «νικητής» σύμφωνα με κάποιον κανόνα διαιτησίας. Η εντολή που ανήκει στο σύνολο αποθήκευσης του «ηττημένου» ανατίθεται στο σύνολο αποθήκευσης του «νικητή».

Ο τέταρτος κανόνας αναφέρεται σε έναν αλγόριθμο διαιτησίας που διαλέγει ένα «νικητή» μεταξύ δύο συνόλων αποθήκευσης. Συγκεκριμένα, επιλέγεται ως νικητής το σύνολο αποθήκευσης που έχει το μικρότερο αύξοντα αριθμό. Αυτός ο αλγόριθμος κάποια στιγμή συγκλίνει και τα σύνολα αποθήκευσης θα έχουν συγχωνευθεί μετά από ένα μικρό αριθμό παγιδεύσεων.

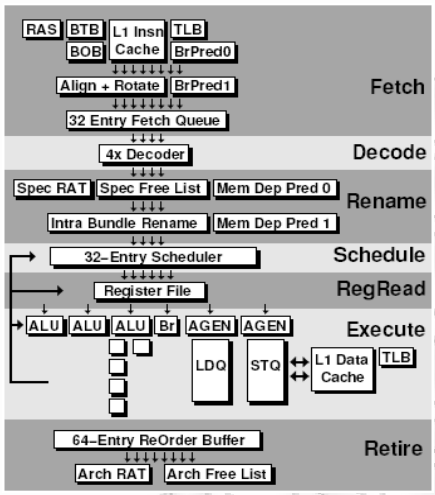
Όταν εκτελείται ένα πρόγραμμα, ο πίνακας SSIT περιέχει καταχωρημένα όλα τα έγκυρα SSIDs που αντιστοιχούν σε έγκυρα σύνολα αποθήκευσης. Πώς όμως ακυρώνονται οι καταχωρήσεις του SSIT; Επειδή δεν υπάρχουν ετικέτες στον πίνακα, κάθε εντολή φόρτωσης και αποθήκευσης τον προσπελάζει και χρησιμοποιεί την πληροφορία που βρίσκει. Δύο εντολές φόρτωσης, μία που χρειάζεται ένα σύνολο αποθήκευσης και μία που δε χρειάζεται, μπορεί να δεικτοδοτούν στην ίδια καταχώρηση του πίνακα SSIT. Είναι κατανοητό πως τέτοιου είδους περιπτώσεις μπορεί να έχουν αρνητική επίδραση στην απόδοση αφού μπορεί να προκληθούν λανθασμένες εξαρτήσεις για τις εντολές φόρτωσης που θα μπορούσαν να εκτελεστούν απευθείας χωρίς να παραβιάζουν τη σειρά εκτέλεσης των εντολών μνήμης. Επίσης, αν δεν ακυρώνονται οι καταχωρήσεις του πίνακα SSIT, υπάρχει το ενδεχόμενο κάποια στιγμή να γίνουν όλες έγκυρες και να προκαλούν εξαρτήσεις για άσχετες εντολές φόρτωσης ή ακόμα χειρότερα για εντολές φόρτωσης που μπορεί να βρίσκονται σε οποιοδήποτε πρόγραμμα. Ο πιο ιδανικός και εύκολος τρόπος

ακύρωσης των καταχωρήσεων είναι να τίθενται όλα τα valid bits τους στο λογικό 0 ανά περιοδικά διαστήματα, για παράδειγμα κάθε 1,000,000 κύκλους, και ο πίνακας SSIT να κατασκευάζεται ξανά από την αρχή. Η υλοποίηση αυτής της μεθόδου είναι πολύ απλή, αφού απαιτούνται μονάχα δύο μετρητές, ένας που θα μετράει το χρονικό διάστημα που απαιτείται για να ακυρωθούν οι καταχωρήσεις του SSIT και ένας που θα ακυρώνει τις καταχωρήσεις του SSIT.

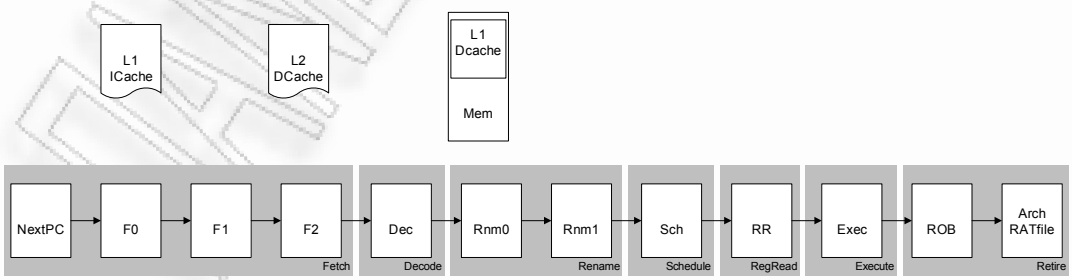
3. Γενική περιγραφή της αρχιτεκτονικής του IVM

Έχοντας αφιερώσει τα δύο πρώτα κεφάλαια στο θεωρητικό υπόβαθρο των υπερβαθμωτών αρχιτεκτονικών, στην ενότητα αυτή θα αναλυθεί το υλικό και η λειτουργία ενός υπερβαθμωτού επεξεργαστή. Ο IVM-1.0 είναι ένας επεξεργαστής, ο οποίος σχεδιάστηκε από το Πανεπιστήμιο του Illinois, Urbana – Champaign. Υλοποιήθηκε στη γλώσσα περιγραφής υλικού Verilog, σε επίπεδο RTL (**R**egister **T**ransfer **L**evel) και αποτελεί το συνδυασμό των επεξεργαστών Alpha 21264 και AMD Athlon, με μεγαλύτερη έμφαση στον επεξεργαστή Alpha 21264.

[7] Η μικροαρχιτεκτονική του IVM-1.0 είναι υπερβαθμωτή. Ο επεξεργαστής υλοποιεί διοχέτευση που αποτελείται από 12 στάδια. Μέχρι και 132 εντολές μπορεί να βρίσκονται υπό επεξεργασία και η δρομολόγηση αυτών γίνεται με δυναμικό τρόπο. Συνολικά περιέχει, εξαιρουμένων των κρυφών μνημών και των πινάκων πρόβλεψης, 50,000 bits στοιχείων μνήμης (pipeline latches και RAM). Το σύνολο των αρχείων Verilog, από τα οποία αποτελείται το μοντέλο, είναι 80 και το μέγεθός τους κυμαίνεται από 10 γραμμές κώδικα περίπου που είναι το μικρότερο αρχείο μέχρι και 2460 γραμμές κώδικα περίπου που είναι το μεγαλύτερο.



Εικόνα 3.1: [8] Ο επεξεργαστής IVM-1.0



Εικόνα 3.2: Η διοχέτευση του επεξεργαστή IVM-1.0

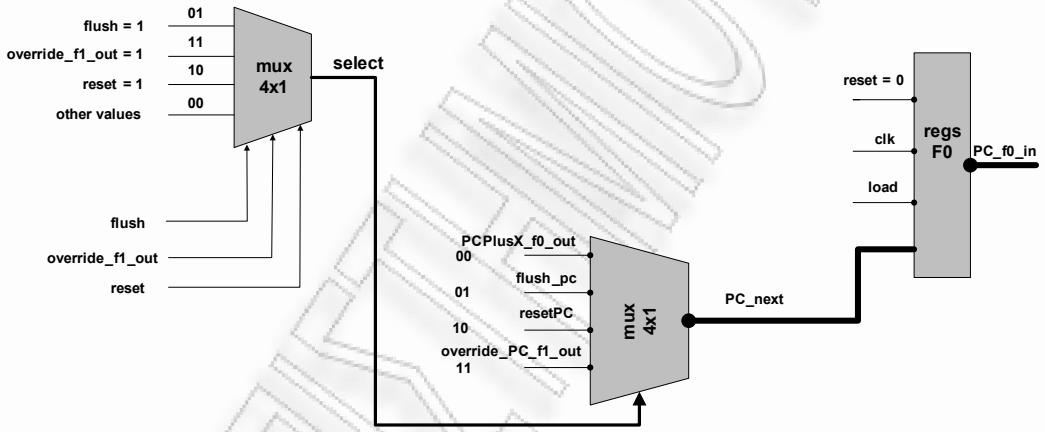
Από το πανεπιστήμιο του Illinois διατίθεται και η έκδοση του IVM-1.1. Όμως η έκδοση αυτή είναι για σύνθεση και έχουν αφαιρεθεί αρκετά συστατικά, όπως οι κρυφές μνήμες των εντολών και των δεδομένων. Επομένως, προτιμήθηκε η αναλυτική περιγραφή του IVM-1.0 που χρησιμοποιείται για προσομοιώσεις και είναι πιο πλήρης. Στις επόμενες ενότητες θα γίνει μία αναλυτική παρουσίαση της διοχέτευσης του IVM και θα επεξηγηθούν οι διάφορες οντότητες που περιέχει.

3.1. Στάδιο προσκόμισης (FETCH)

Το στάδιο αυτό είναι το πρώτο στη διοχέτευση του IVM. Εκτός της προσκόμισης των εντολών από τη κρυφή μνήμη εντολών, περιλαμβάνει και όλες τις οντότητες που κάνουν τον επεξεργαστή να λειτουργεί με εικασία (speculative). Δηλαδή στο στάδιο προσκόμισης γίνονται εκτιμήσεις όσον αφορά τη λήψη των εντολών διακλάδωσης (υπό συνθήκη, χωρίς συνθήκη, επιστροφή από ρουτίνα, μετάβαση σε ρουτίνα) και προβλέπεται η επόμενη ροή εντολών που θα εκτελεστεί. Αποτελείται από τα τέσσερα υποστάδια NextPC, F0, F1 και F2.

3.1.1. Στάδιο NextPC

Σε αυτό το υποστάδιο, όπως είναι προφανές και από το όνομά του, επιλέγεται η επόμενη διεύθυνση προσκόμισης των εντολών. Όπως φαίνεται από την Εικόνα 3.3:



Εικόνα 3.3: Το στάδιο NextPC

Το υποστάδιο NextPC αποτελείται από δύο πολυπλέκτες. Με τη βοήθεια αυτών των πολυπλεκτών επιλέγεται η επόμενη διεύθυνση προσκόμισης των εντολών και προωθείται στο επόμενο υποστάδιο του σταδίου fetch και στη κρυφή μνήμη, στον επόμενο κύκλο ρολογιού, αφού παρεμβάλλεται ο καταχωρητής regsF0 (που δεν αποτελεί μέρος του NextPC, απλά προστέθηκε για να δείξει την καθυστέρηση που προκύπτει στη μετάδοση των δεδομένων). Η επόμενη διεύθυνση προσκόμισης μπορεί να προέρχεται από το στάδιο F0 (PCPlusX_f0_out) ή από το στάδιο F1 (override_PC_f1_out) ή να είναι κάποια διεύθυνση από την οποία πρέπει να γίνει προσκόμιση μετά από κάποια εκκένωση (flush_pc) ή μετά από επαναφορά (resetPC).

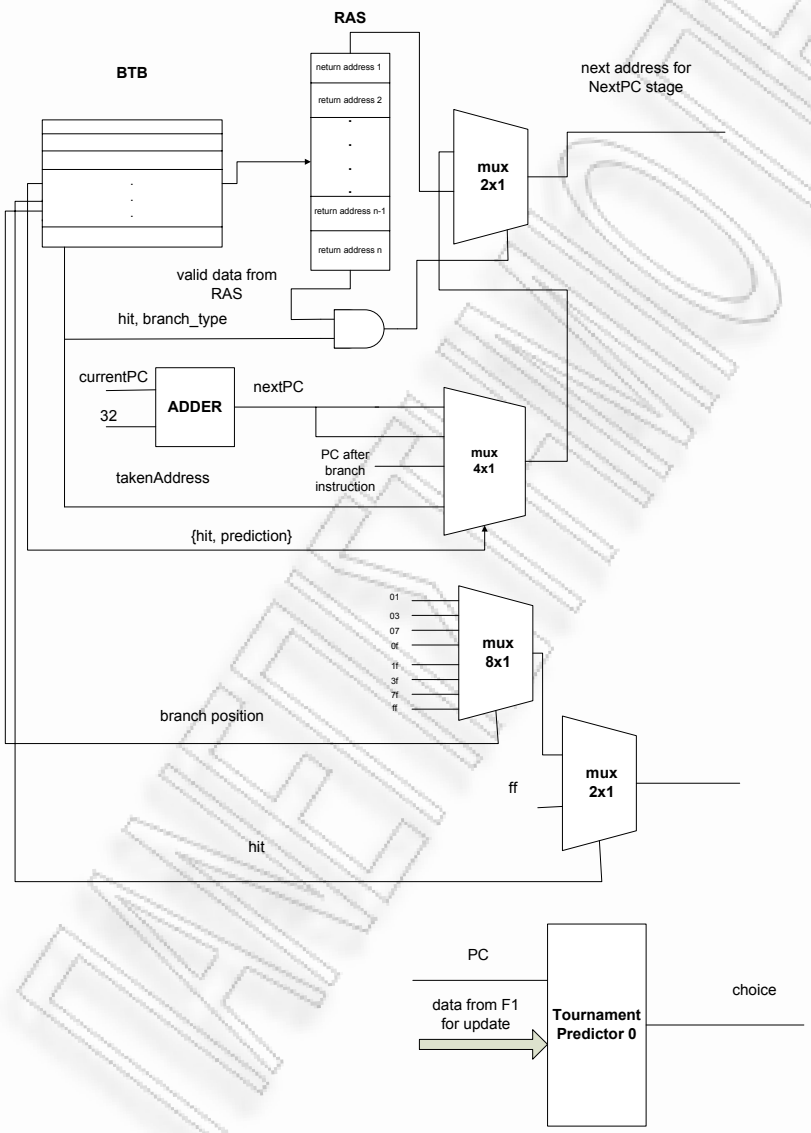
3.1.2. Στάδιο F0

Σε αυτό το υποστάδιο γίνεται μία αρχική εκτίμηση όσον αφορά τη λήψη των διακλαδώσεων και με βάση την εκτίμηση αυτή υπολογίζεται η επόμενη διεύθυνση προσκόμισης των εντολών. Να σημειωθεί πως μία διεύθυνση αντιστοιχεί σε μία οκτάδα εντολών, δηλαδή οι εντολές προσκομίζονται ανά οκτώ από την κρυφή μνήμη. Όταν φτάσει από το στάδιο NextPC στο στάδιο F0 η διεύθυνση προσκόμισης, ελέγχεται από το **B**ranch **T**arget **B**uffer (**BTB**) αν υπάρχει κάποια εντολή διακλάδωσης στην αντίστοιχη οκτάδα. Βέβαια, για να είναι σε θέση να κάνει τέτοιο έλεγχο ο BTB, θα πρέπει να περιέχει πληροφορίες για αυτή τη διεύθυνση μνήμης, δηλαδή θα πρέπει το συγκεκριμένο σύνολο εντολών να είχε προσκομιστεί ξανά

στο παρελθόν, να είχε αναλυθεί (στο υποστάδιο F1, όπως θα δούμε παρακάτω) και να είχαν αποθηκευτεί οι σχετικές πληροφορίες στον BTB.

Αν ο BTB δεν έχει καμία πληροφορία για τη συγκεκριμένη διεύθυνση τότε σαν επόμενη διεύθυνση προσκόμισης προβλέπεται και επιλέγεται αυτή της επόμενης οκτάδας εντολών (PC + 32, αφού είναι οκτώ εντολές και κάθε εντολή έχει μέγεθος τέσσερα bytes).

Αν όμως υπάρχει πληροφορία στο BTB για τη διεύθυνση τότε πρέπει να εξεταστούν κάποιοι παράμετροι όπως: σε ποια θέση βρίσκεται η διακλάδωση μέσα στην οκτάδα, τι τύπος είναι, η πρόβλεψή της (αν λαμβάνεται ή όχι) και άλλες. Για παράδειγμα, αν στην οκτάδα των εντολών υπάρχει διακλάδωση και έχει προβλεφθεί ότι λαμβάνεται, τότε: Αν η διακλάδωση είναι επιστροφή από ρουτίνα, τότε σαν επόμενη διεύθυνση προβλέπεται να είναι η διεύθυνση που βρίσκεται στην κορυφή της στοίβας RAS. Αν η διακλάδωση είναι υπό συνθήκη ή είναι ένα άλμα, τότε σαν επόμενη διεύθυνση επιλέγεται η διεύθυνση



Εικόνα 3.4: Το στάδιο F0

προορισμού που έχει προβλεφθεί και είναι αποθηκευμένη στον BTB. Επίσης, μία εντολή διακλάδωσης μπορεί να είναι οποιαδήποτε από τις οκτώ εντολές. Μεγάλο ρόλο λοιπόν παίζει και η θέση της μέσα στην

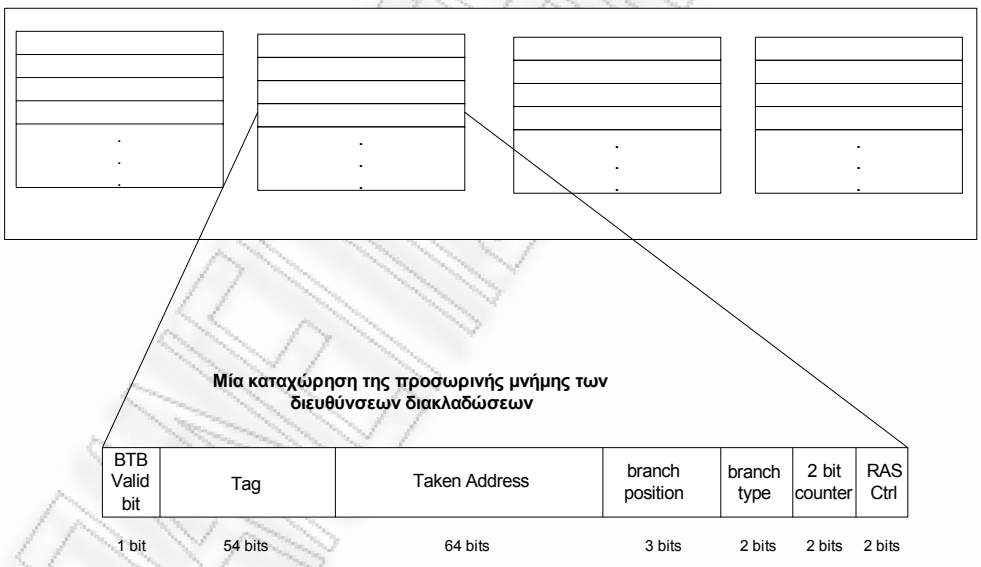
οκτάδα. Αν σε μία οκτάδα βρίσκονται περισσότερες από μία διακλαδώσεις, τότε ελέγχεται πάντα η πρώτη. Επιπρόσθετα, αν από το BTB έχει προβλεφθεί ότι η διακλάδωση δε λαμβάνεται τότε σαν επόμενη διεύθυνση επιλέγεται η διεύθυνση της εντολής που ακολουθεί αμέσως μετά την εντολή διακλάδωσης.

Εκτός όμως από την εκτίμηση της επόμενης διεύθυνσης, στο υποστάδιο F0 παράγονται τα ενδιάμεσα αποτελέσματα του tournament predictor. Όπως θα αναλυθεί παρακάτω, ο επεξεργαστής IVM περιέχει αυτόν τον υβριδικό μηχανισμό για την αξιόπιστη πρόβλεψη των διακλαδώσεων υπό συνθήκη. Ακριβώς επειδή είναι αξιόπιστος, χρειάζεται δύο κύκλους για να παράγει την πρόβλεψη (είναι σχετικά αργός). Οπότε στο υποστάδιο F0 παράγονται τα πρώτα αποτελέσματα που θα χρησιμοποιηθούν στο υποστάδιο F1 για να παραχθεί η τελική πρόβλεψη.

Προσωρινή μνήμη διευθύνσεων διακλαδώσεων (Branch Target Buffer – BTB)

Η προσωρινή μνήμη διευθύνσεων διακλαδώσεων είναι μία συνολοσυσχετιστική, τεσσάρων δρόμων κρυφή μνήμη η οποία χρησιμοποιείται για τη σύντομη πρόβλεψη των διακλαδώσεων. Δηλαδή η πρώτη, γρήγορη πρόβλεψη που γίνεται για την επόμενη διεύθυνση προσκόμισης των εντολών βασίζεται στον BTB.

Ο BTB περιέχει πληροφορίες για τις διευθύνσεις των δεσμών των εντολών, όπου μία δέσμη αποτελείται από οκτώ εντολές. Οι πληροφορίες αυτές σχετίζονται με τις διακλαδώσεις. Πιο συγκεκριμένα περιλαμβάνουν: Τον τύπο της διακλάδωσης, τη θέση της εντολής διακλάδωσης μέσα στην οκτάδα, αν λαμβάνεται ή όχι (εκτίμηση με βάση τις παλιότερες εκτελέσεις της εντολής), την προβλεφθείσα διεύθυνση προορισμού και τη λειτουργία που γίνεται στη στοίβα επιστροφής διευθύνσεων (**R**eturn **A**ddress **S**tack – **RAS**). Για παράδειγμα, αν η διακλάδωση είναι κλήση σε ρουτίνα, τότε η λειτουργία που απαιτείται να γίνει είναι αυτή της τοποθέτησης στη στοίβα (push) ώστε να αποθηκευτεί η διεύθυνση επιστροφής. Αν η διακλάδωση είναι επιστροφή από ρουτίνα, τότε η λειτουργία που απαιτείται να γίνει είναι αυτή της εξαγωγής από τη στοίβα (pop) ώστε να προσκομιστεί η διεύθυνση επιστροφής. Επίσης, δύο άλλες πληροφορίες που περιέχονται για μία διεύθυνση δέσμης είναι: Ένα valid bit, όπου αν είναι στο λογικό 1 σημαίνει ότι η καταχώρηση στον BTB είναι έγκυρη ενώ στην αντίθετη περίπτωση άκυρη και μία ετικέτα (tag) που χρησιμεύει στην αναζήτηση της καταχώρησης.



Εικόνα 3.5: Η προσωρινή μνήμη διευθύνσεων διακλαδώσεων και οι πληροφορίες που περιέχει μία καταχώριση

Ο BTB αποτελείται από 4 δρόμους και κάθε δρόμος περιέχει 256 καταχωρίσεις. Μία καταχώρηση έχει μέγεθος 128 bits, οπότε ένας δρόμος έχει μέγεθος 128x256 = 32768 bits και συνολικά ο BTB έχει μέγεθος 32768x4 = 131072 bits, δηλαδή 128 KB. Για την εύρεση μιας καταχώρησης γίνεται παράλληλη αναζήτηση και στους τέσσερις δρόμους ενώ για την αντικατάσταση ή την εγγραφή μιας καταχώρησης χρησιμοποιείται η πολιτική LRU.

RAS

Στην ιδανική περίπτωση, ένας επεξεργαστής με διοχέτευση μπορεί να εκτελεί εντολές με μέγιστο ρυθμό αυτόν που ορίζεται από το πιο αργό στάδιο. Παρόλα αυτά, οι εντολές διακλάδωσης χαλάνε τη ροή των εντολών στη διοχέτευση και μειώνουν την απόδοση του επεξεργαστή. Από τη στιγμή που τα φορτία εργασίας περιέχουν ένα μεγάλο ποσοστό από διακλαδώσεις που λαμβάνονται (γιατί σε αυτή την περίπτωση μόνο αλλάζει η ροή εκτέλεσης των εντολών), τότε χρειάζεται η εφαρμογή κάποιων τεχνικών για να περιορίσουν ή ακόμη και να εξαλείψουν το συγκεκριμένο φαινόμενο.

Για τη μείωση της επιβάρυνσης που προκαλείται από τις εντολές διακλάδωσης, έχουν προταθεί διάφορες τεχνικές, όπως η καθυστερημένη διακλάδωση, το δίπλωμα διακλάδωσης, η στατική πρόβλεψη διακλάδωσης που βασίζεται στον κωδικό λειτουργίας (opcode) της εντολής και η πρόωρη προσκόμιση του προορισμού διακλάδωσης.

Όλες οι προαναφερθείσες τεχνικές έχουν ένα κοινό πρόβλημα το οποίο είναι ότι η πρόβλεψη ή η ανακατεύθυνση γίνεται στο στάδιο αποκωδικοποίησης ή ακόμη χειρότερα στο στάδιο εκτέλεσης. Επομένως, ένας επεξεργαστής μέχρι εκείνη την ώρα θα έχει προσκομίσει και θα έχει αποκωδικοποιήσει εντολές που πιθανότατα δε θα εκτελεστούν. Για το λόγο αυτό προτάθηκε ο πίνακας ιστορικού διακλάδωσης (**B**ranch **H**istory **T**able - **BHT**) ο οποίος κάνει αυτόνομη πρόβλεψη και πρόωρη προσκόμιση των προορισμών των διακλαδώσεων. Το σημαντικότερο είναι ότι ο BHT έχει τη δυνατότητα να κάνει προβλέψεις πολύ νωρίτερα, ακόμη και στο στάδιο FETCH, σε σχέση με τις άλλες διατάξεις. Είναι της ίδιας φιλοσοφίας, ίσως και πιο απλός, με τον BTB που επεξηγήθηκε παραπάνω.

Είναι κατανοητό πως μία διάταξη πρόβλεψης διακλάδωσης μπορεί να κάνει λάθος προβλέψεις. Για παράδειγμα, ο BHT βασίζει την πρόβλεψή του για μία διακλάδωση στην ιστορία της, δηλαδή στις αποφάσεις που έχουν ληφθεί στο παρελθόν για το αν λαμβάνεται η συγκεκριμένη διακλάδωση ή όχι. Έχει παρατηρηθεί λοιπόν πως οι λανθασμένες προβλέψεις που προκαλούνται λόγω της μεταβαλλόμενης ιστορίας των διακλαδώσεων είναι πάνω από τις μισές περιπτώσεις. Όμως για τις υπόλοιπες περιπτώσεις λανθασμένων προβλέψεων που αναλύθηκαν, βρέθηκε ότι ένα μεγάλο ποσοστό προκλήθηκε λόγω του μεταβαλλόμενου προορισμού διακλάδωσης. Μεγάλο ρόλο σε αυτό παίζει ο μηχανισμός ΚΛΗΣΗΣ/ΕΠΙΣΤΡΟΦΗΣ από ρουτίνες. Για να γίνει πιο σαφές παρατίθεται το ακόλουθο παράδειγμα.

Έστω τμήμα κώδικα:

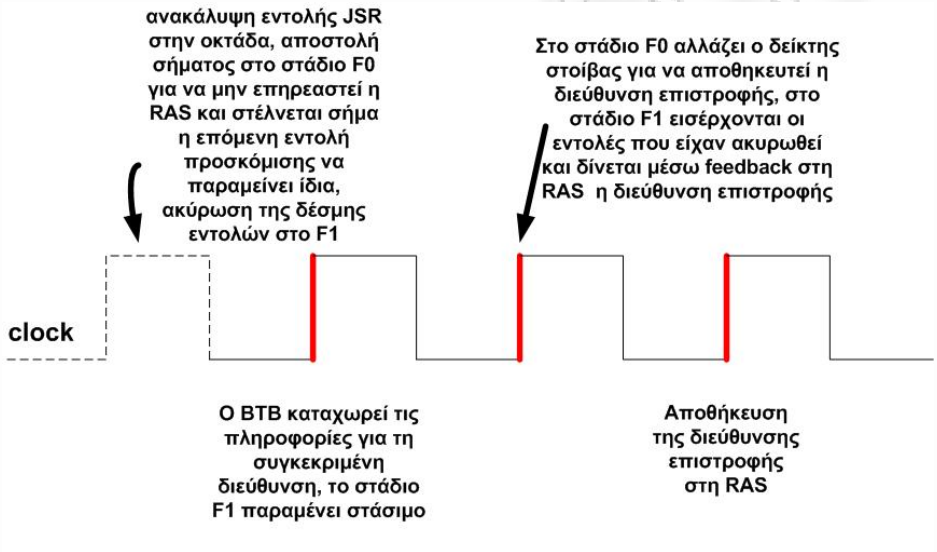
```
Διεύθυνση εντολής
0                   JSR Rout
4                   .
8                   .
12                  .
16                  JSR Rout
20                  .
.
.

Rout:
2000                .
2004                .
.                   .
.                   .
2100                RET
```

Από τη διεύθυνση 0, γίνεται διακλάδωση στην υπορουτίνα Rout με την εντολή JSR, η οποία βρίσκεται στη διεύθυνση 2000. Έτσι στον BHT προστίθεται μία καταχώρηση για τη διεύθυνση 0 με διεύθυνση προορισμού τη 2000. Επιστρέφοντας από τη ρουτίνα Rout με την εντολή RET, προστίθεται στον BHT η διεύθυνση 2100 με διεύθυνση προορισμού τη 4. Όταν εκτελεστεί η εντολή με διεύθυνση μνήμης 16, γίνεται ακόμα μία διακλάδωση στην υπορουτίνα Rout, οπότε προστίθεται πάλι στον BHT η διεύθυνση 16 με διεύθυνση προορισμού τη 2000. Όταν φτάσουμε στο τέλος της υπορουτίνας Rout, θα πρέπει να γίνει επιστροφή, μέσω της εντολής RET, για να συνεχιστεί η εκτέλεση στο κύριο πρόγραμμα. Όμως, στη συγκεκριμένη περίπτωση θα γίνει έλεγχος στο BHT και θα βρεθεί μία έγκυρη εγγραφή (είναι αυτή που είχε καταχωρηθεί από την προηγούμενη εκτέλεση της εντολής RET) η οποία προβλέπει διακλάδωση στην εντολή με διεύθυνση μνήμης 4. Προφανώς η πρόβλεψη είναι λανθασμένη, όσον αφορά τη

διεύθυνση προορισμού, διότι μετά την επιστροφή η εντολή που πρέπει να εκτελεστεί είναι αυτή με διεύθυνση μνήμης 20.

Επομένως για να επιλυθεί το συγκεκριμένο πρόβλημα, προτάθηκε η υλοποίηση της στοίβας διευθύνσεων επιστροφών (**RAS** - **R**eturn **A**ddress **S**tack). Η στοίβα αυτή χρησιμοποιείται για την πρόβλεψη των επιστροφών από ρουτίνες. Όταν καλείται μια ρουτίνα (JSR), τοποθετείται στη RAS η διεύθυνση επιστροφής, ενώ όταν εκτελείται η εντολή επιστροφής από ρουτίνα (RET) εξάγεται η αντίστοιχη διεύθυνση επιστροφής από τη RAS.

Η RAS στον IVM περιέχει 16 καταχωρήσεις με 65 bits μέγεθος κάθε μία, δηλαδή το μέγεθος συνολικά της RAS είναι 130 bytes. Λειτουργεί ως εξής: Έστω ότι μία δέσμη εντολών, που προσκομίζεται για πρώτη φορά από την κρυφή μνήμη εντολών, περιέχει μία εντολή JSR. Εφόσον δεν υπάρχει καμιά καταχώρηση στον BTB, δεν υπάρχει κάποια αρχική πρόβλεψη. Οπότε στο στάδιο F1, ανιχνεύεται η ύπαρξη διακλάδωσης μέσω των αποκωδικοποιητών διακλαδώσεων και στέλνεται σήμα να σταματήσει η κανονική προσκόμιση των εντολών και να γίνει προσκόμιση από την ίδια διεύθυνση. Στον επόμενο κύκλο (ας είναι ο t), στο στάδιο F0, καταγράφεται στον BTB η διεύθυνση της συγκεκριμένης οκτάδας εντολών και ότι πρέπει να γίνει τοποθέτηση της διεύθυνσης επιστροφής στη RAS. Στον κύκλο t+1, γίνεται αλλαγή στο δείκτη της στοίβας ώστε να υποδεχθεί τη νέα διεύθυνση επιστροφής και στον κύκλο t+2 αποθηκεύεται η διεύθυνση επιστροφής στη RAS, η οποία έχει σταλεί μέσω ανάδρασης (feedback) από το στάδιο F1.



Εικόνα 3.6: Χρονική ανάλυση στη λειτουργία της στοίβας RAS στην περίπτωση που πρέπει να αποθηκεύσει μία νέα διεύθυνση επιστροφής, όταν ο BTB δεν περιέχει καμία πληροφορία για τη συγκεκριμένη δέσμη των εντολών

Αν προσκομιστεί πάλι η ίδια δέσμη εντολών (λόγω κάποιας νέας επανάληψης) για εκτέλεση, τότε στο BTB θα υπάρχει έγκυρη πληροφορία για αυτήν. Επομένως, οι εντολές θα προωθηθούν κανονικά στο στάδιο F1, δε θα γίνει καμία διακοπή της προβλεφθείσας ροής των εντολών και θα σταλεί μέσω ανάδρασης η διεύθυνση επιστροφής για την αποθήκευσή της στη RAS.

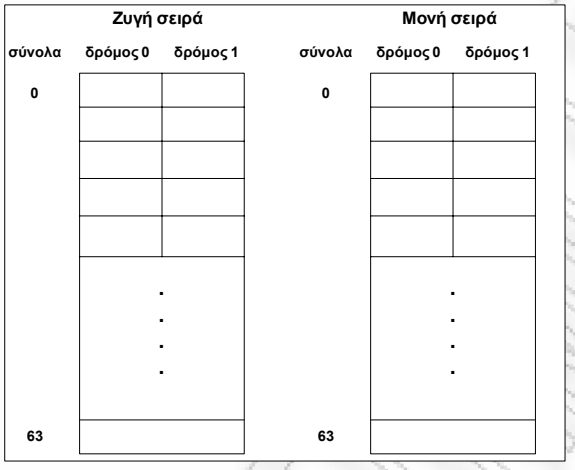
Αν προσκομιστεί για πρώτη φορά μία δέσμη εντολών που περιέχει την εντολή για επιστροφή από τη ρουτίνα (RET), τότε δε θα υπάρχει πάλι κάποια καταχώρηση στο BTB, του σταδίου F0, οπότε θα ακολουθηθεί η ίδια διαδικασία για την εγγραφή των σχετικών πληροφοριών στο BTB και την εξαγωγή της διεύθυνσης επιστροφής από τη RAS. Στην περίπτωση που η συγκεκριμένη δέσμη, που περιέχει την εντολή RET, προσκομιστεί ξανά, τότε θα υπάρχει έγκυρη καταχώρηση στο BTB και έτσι δε θα συμβεί καμία καθυστέρηση.

Η κρυφή μνήμη εντολών L1

Η κρυφή μνήμη εντολών L1 του IVM είναι συνολοσυσχετιστική (set-associative), δύο δρόμων (ways) και αποτελείται από δύο σειρές (banks). Κάθε σειρά περιέχει 64 καταχωρήσεις και σε κάθε καταχώρηση

βρίσκονται δύο γραμμές (μία για κάθε δρόμο). Κάθε γραμμή μπορεί να έχει μέχρι και οκτώ εντολές. Το συνολικό μέγεθος της κρυφής μνήμης εντολών είναι ίσο με: 2 σειρές x (64 σύνολα x 2 δρόμοι σε κάθε σύνολο x 32 bytes σε κάθε δρόμο) = 8192 bytes = 8KB.

Η πολιτική αντικατάστασης που χρησιμοποιείται είναι η LRU, αλλά δεν παίζει κάποιο ρόλο στο μοντέλο του IVM διότι κρυφή μνήμη L2 ή κάποιο άλλο είδος μνήμης, χαμηλότερου επιπέδου στην ιεραρχία, δεν υπάρχει. Συνεπώς απαιτείται μεγάλη προσοχή, όταν φορτώνεται ένα πρόγραμμα και εκτελείται, να μη δημιουργούνται αστοχίες ώστε να χρειάζεται να γίνουν αντικαταστάσεις. Διαφορετικά, θα πρέπει να χρησιμοποιηθούν διαφορετικές τακτικές για να μην υπάρχει πρόβλημα στην προσομοίωση. Επίσης, θα πρέπει να τονισθεί ότι οι δύο σειρές της κρυφής μνήμης είναι πλεκτές (interleaved). Δηλαδή, στη μία σειρά αποθηκεύονται οι δέσμες των εντολών με ζυγή διεύθυνση και στην άλλη σειρά αποθηκεύονται οι δέσμες των εντολών με μονή διεύθυνση μνήμης. Για τους δρόμους ισχύουν τα εξής: Στον ένα δρόμο αποθηκεύονται οι δέσμες των εντολών που η διεύθυνση τους είναι μέχρι και 128 και στον άλλο δρόμο αποθηκεύονται οι δέσμες των εντολών που η διεύθυνσή τους είναι πάνω από 128 μέχρι και 256 (αφού η κρυφή μνήμη έχει μέγεθος 8 KB, μπορούν να αποθηκευτούν συνολικά 256 οκτάδες).



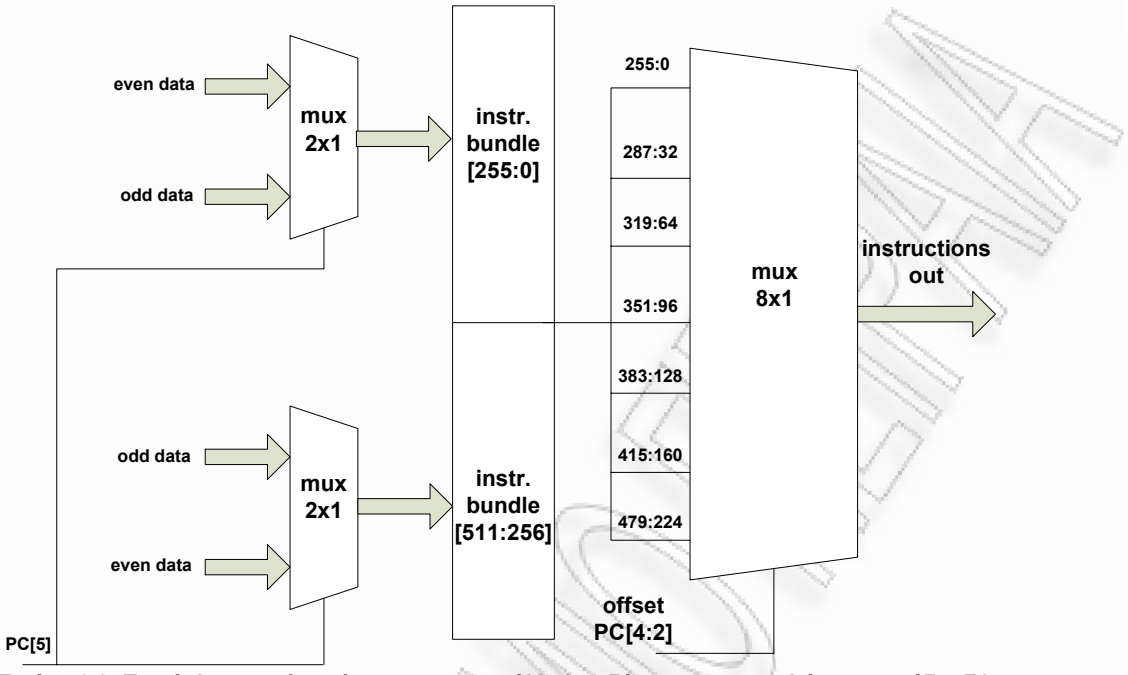
Περιγραφή των bits της διεύθυνσης
Μέγεθος διεύθυνσης: 64 bits

tag 50 bits - PC[63:12]	set 6 bits - PC[11:6]	bank 1 bit PC[5]	offset 3 bits PC[4:2]	bytes Instr. 2 bits PC[1:0]

Εικόνα 3.7: Η κρυφή μνήμη των εντολών L1

Η προσκόμιση των εντολών γίνεται πάντα ανά δέσμες των οκτώ και οι διευθύνσεις είναι πολλαπλάσιες του 32. Αν προκύψει κάποια διακλάδωση σε εντολή, η οποία έχει διεύθυνση που δεν είναι πολλαπλάσια του 32, τότε και πάλι η διεύθυνση προσκόμισης μετατρέπεται σε πολλαπλάσια του 32, μηδενίζοντας τα πέντε δεξιότερα bits. Παράλληλα με την τρέχουσα διεύθυνση προσκόμισης της δέσμης των εντολών, υπολογίζεται και η διεύθυνση της επόμενης δέσμης (είναι η τρέχουσα συν 32, αφού κάθε δέσμη έχει οκτώ εντολές και κάθε εντολή έχει μέγεθος 4 bytes) που επίσης φορτώνεται και αυτή από την κρυφή μνήμη.

Στο στάδιο F1 μεταφέρονται οι δύο οκτάδες των εντολών. Εκεί υπάρχει ένα κύκλωμα ευθυγράμμισης (alignment circuit), το οποίο επιλέγει από τη δεκαεξάδα, την κατάλληλη οκτάδα προς εκτέλεση.



Εικόνα 3.8: Το κύκλωμα ευθυγράμμισης της κατάλληλης δέσμης των εντολών στο στάδιο F1

Η λειτουργία του κυκλώματος, που φαίνεται στην εικόνα 3.8, μπορεί να εξηγηθεί καλύτερα με ένα παράδειγμα. Έστω ότι γίνεται διακλάδωση σε μία εντολή με διεύθυνση 0x44457328. Επειδή η διεύθυνση πρέπει να είναι πολλαπλάσια του 32, θα μηδενιστούν τα δεξιότερα πέντε bits οπότε θα γίνει ίση με 0x44457320. Από την κρυφή μνήμη προσκομίζονται οι δέσμες των εντολών με διευθύνσεις: 0x44457320 και 0x44457340 (η τρέχουσα και η επόμενη). Οι δύο οκτάδες μεταφέρονται στο στάδιο F1 και μέσω του κυκλώματος ευθυγράμμισης, σχηματίζεται η κατάλληλη οκτάδα προς εκτέλεση. Παρατηρώντας αναλυτικότερα τις διευθύνσεις, το bit 5 της παρούσας διεύθυνσης είναι 1, οπότε τα δεδομένα έχουν προσκομιστεί από τη σειρά 1 της κρυφής μνήμης (είναι τα «μονά» δεδομένα επειδή η διεύθυνσή τους είναι μονή) και καταλαμβάνουν τα bits [255:0] του καταχωρητή (που βρίσκεται ανάμεσα στους 2 πολυπλέκτες 2x1 και του πολυπλέκτη 8x1). Το bit 5 της επόμενης διεύθυνσης είναι ίσο με 0. Οπότε η προσκόμιση της αντίστοιχης δέσμης έγινε από τη σειρά 0 (είναι τα «ζυγά» δεδομένα) και καταλαμβάνουν τα bits [511:256] του καταχωρητή. Έπειτα, με βάση το πεδίο σχετική απόσταση (offset) της αρχικής, χωρίς στρογγυλοποίηση, διεύθυνσης (bits 4:2) επιλέγεται η κατάλληλη οκτάδα εντολών προς εκτέλεση. Για τη διεύθυνση 0x44457328, το πεδίο αυτό είναι ίσο με το δυαδικό 010. Άρα οι οκτώ εντολές που επιλέγονται είναι τα bits [319:64], με πρώτη την εντολή στην οποία έγινε η διακλάδωση.

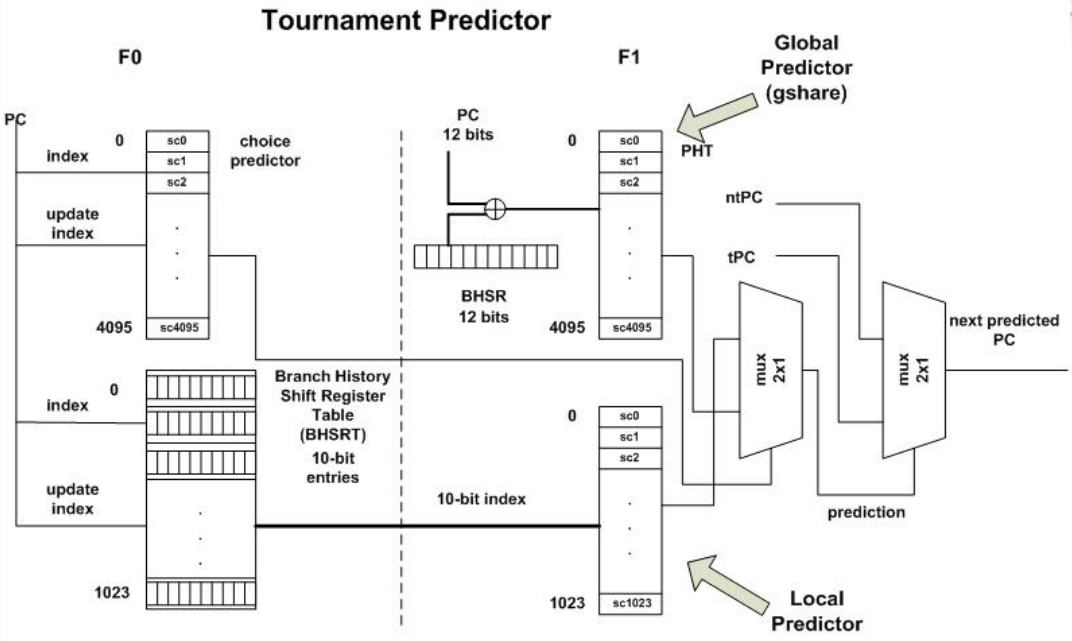
Ο λόγος που ακολουθείται η συγκεκριμένη διαδικασία είναι επειδή πρέπει να είναι μεγιστοποιημένο το εύρος προσκόμισης. Δηλαδή θα πρέπει σε κάθε κύκλο να προωθούνται από το στάδιο FETCH οκτώ εντολές.

Η διάταξη πρόβλεψης διακλάδωσης στον IVM

Αποτελεί μία από τις πιο σημαντικές οντότητες του επεξεργαστή, αφού χάρη σε αυτήν μπορεί να εκτελεί εντολές με εικασία. Στον IVM έχει σχεδιαστεί μία υβριδική διάταξη πρόβλεψης διακλάδωσης, η οποία αποτελείται από τη μικρή διάταξη πρόβλεψης διακλάδωσης (**SmP**) και από τη μεγάλη διάταξη πρόβλεψης διακλάδωσης (**LaP**). Η SmP χρησιμοποιείται για να κάνει γρήγορες προβλέψεις, σε χρόνο λιγότερο από έναν κύκλο ρολογιού, διότι πρέπει σε κάθε νέο κύκλο να γίνεται προσκόμιση μίας νέας δέσμης εντολών. Αντίθετα, η LaP αποτελεί την αξιόπιστη διάταξη και κάνει πιο ασφαλείς και ακριβείς προβλέψεις. Λειτουργεί πιο αργά αφού χρειάζονται 2 κύκλοι για να παραχθεί η πρόβλεψη και χρησιμοποιείται για την πιστοποίηση των προβλέψεων του SmP.

Η SmP είναι ο BTB, όπου κάθε καταχώρησή του περιέχει και έναν μετρητή κορεσμού μεγέθους 2 bit. Ο μετρητής αυτός αποτελεί μία διάταξη πρόβλεψης διακλάδωσης. Όταν είναι πάνω από ένα τότε η

διακλάδωση προβλέπεται να ληφθεί, αλλιώς δε λαμβάνεται. Επίσης ο μετρητής αυτός έχει την ιδιότητα ότι όταν φτάνει στη μέγιστη τιμή του (3) και δίνεται εντολή αύξησης κατά ένα, τότε η τιμή του παραμένει αμετάβλητη και δε μηδενίζεται. Παρομοίως, αν ο μετρητής είναι στην ελάχιστη τιμή του και δίνεται εντολή για μείωση κατά ένα, τότε η τιμή του παραμένει ξανά αμετάβλητη. Ο μετρητής κορεσμού μιας καταχώρισης του BTB ανανεώνεται αν ολοκληρώθηκε κάποια εντολή διακλάδωσης. Τότε, ελέγχεται αν η διακλάδωση λήφθηκε ή όχι. Αν ναι, τότε η τιμή του saturating counter αυξάνεται κατά 1, αλλιώς μειώνεται κατά 1.



Εικόνα 3.9: Η μεγάλη διάταξη πρόβλεψης διακλάδωσης (LaP)

Η LaP (tournament predictor) αποτελείται από μία τοπική διάταξη πρόβλεψης διακλάδωσης (local predictor – SAg), από μία σφαιρική διάταξη πρόβλεψης διακλάδωσης (global predictor – gshare) και μία διάταξη πρόβλεψης της διάταξης πρόβλεψης της διακλάδωσης (choice predictor ή αλλιώς meta-predictor), η οποία προβλέπει την επιλογή μεταξύ SAg και gshare για να βγει πιο σωστή η πρόβλεψη. Ανάλογα με την εκτίμηση που κάνει καθορίζει αν πρέπει να σταματήσει η κανονική ροή των εντολών λόγω λάθος αρχικής εκτίμησης (από τη μικρή διάταξη πρόβλεψης διακλάδωσης). Επίσης προβλέπει και την επόμενη διεύθυνση προσκόμισης εντολών και αν πρέπει να ανανεωθεί ο choice predictor επειδή οι SAg και gshare βγάζουν διαφορετικές τιμές.

Ο meta-predictor εκτελείται στο υποστάδιο F0, έτσι ώστε στο υποστάδιο F1, με βάση την πρόβλεψή του, να επιλεγεί η κατάλληλη διάταξη (τοπική ή σφαιρική) για την πρόβλεψη μιας διακλάδωσης. Αποτελείται από έναν πίνακα 4096 θέσεων, ο οποίος δεικτοδοτείται με βάση κάποια bit (12) της διεύθυνσης μνήμης της δέσμης των εντολών. Κάθε καταχώρηση του πίνακα, περιέχει ένα saturating counter, μεγέθους δύο bit. Το αριστερό bit καθορίζει την επιλογή της SAg ή της gshare. Αν το bit είναι ίσο με 1, τότε επιλέγεται ο gshare, αλλιώς η SAg.

Ανανέωση σε κάποια θέση του πίνακα γίνεται όταν ολοκληρωθεί κάποια εντολή διακλάδωσης και το αντίστοιχο σήμα από το BOB (**B**ranch **O**rder **B**uffer - θα επεξηγηθεί αργότερα) είναι ίσο με 1. Όμως η τιμή της ανανέωσης είναι ίση με: *update_value = branch_value ^ local_predictor_value*. Δηλαδή, αν η SAg έχει την ίδια πρόβλεψη με την έκβαση της διακλάδωσης, τότε η τιμή ανανέωσης είναι 0, έτσι ώστε ο αντίστοιχος saturating counter να μειωθεί κατά 1 και ο meta-predictor να έχει την τάση να επιλέγει την SAg, αφού κάνει σωστές προβλέψεις. Αντίθετα, αν η έκβαση της διακλάδωσης είναι διαφορετική από την πρόβλεψη της SAg, τότε γίνεται η τιμή ανανέωσης γίνεται 1 έτσι ώστε να αυξηθεί η τιμή του saturating counter και να έχει την τάση να επιλέγει τη gshare.

Η διάταξη πρόβλεψης SAg επεκτείνεται στα υποστάδια F0 και F1. Στο υποστάδιο F0 περιέχει έναν πίνακα 1024 θέσεων που ονομάζεται **BHSRT** (**B**ranch **H**istory **S**hift **R**egister **T**able). Σε κάθε θέση βρίσκεται ένας καταχωρητής αριστερής ολίσθησης, μεγέθους 10 bits, που ονομάζεται **BHSR** (**B**ranch **H**istory **S**hift **R**egister) και περιέχει τις 10 πιο πρόσφατες εκβάσεις μιας εντολής διακλάδωσης (ή παραπάνω από μία λόγω ψευδωνυμίας - aliasing). Όταν ολοκληρωθεί κάποια διακλάδωση και πρέπει να γίνει ανανέωση, τότε όλα τα bits του αντίστοιχου καταχωρητή ολισθαίνουν κατά μία θέση αριστερά και η τιμή της ολοκληρωμένης διακλάδωσης αποθηκεύεται στη δεξιότερη θέση.

Στο στάδιο F1, η SAg περιέχει έναν πίνακα (**PHT** – **P**attern **H**istory **T**able), ο οποίος έχει 1024 καταχωρίσεις και σε κάθε καταχώριση περιλαμβάνεται ένας saturating counter των δύο bits. Για την παραγωγή της πρόβλεψης της SAg χρησιμοποιείται το αριστερό bit ενός από τους saturating counters που επιλέγεται. Ο πίνακας PHT δεικτοδοτείται από την τιμή ενός BHSR που εξάγεται από τον πίνακα BHSRT του σταδίου F0 και ανανεώνεται όταν ολοκληρωθεί κάποια εντολή διακλάδωσης υπό συνθήκη.

Έχοντας παρουσιάσει τις οντότητες και αναλύσει τη λειτουργία της SAg, καλό είναι να αναφερθεί πώς προέκυψε το όνομά της. Το A προκύπτει από τη λέξη Adaptive. Οι Yeh και Patt έχουν ονομάσει έτσι όλα αυτά τα σχήματα επειδή χρησιμοποιούν σαν διάταξη πρόβλεψης διακλάδωσης τους saturating counters των 2 bits. Το S προκύπτει από τη λέξη shared διότι κάθε καταχώρηση στον πίνακα BHSRT μπορεί να δεικτοδοτείται από ένα σύνολο διευθύνσεων δεσμών εντολών (ψευδωνυμία) και το g προκύπτει από τη λέξη global επειδή διατίθεται ένας σφαιρικός PHT στον οποίο έχουν πρόσβαση όλες οι εντολές διακλάδωσης.

Η διάταξη πρόβλεψης διακλάδωσης gshare βασίζεται στο gshare Correlated Branch Predictor που προτάθηκε από τον Scott McFarling το 1993. Το κύριο πλεονέκτημά της είναι ότι πετυχαίνει υψηλά ποσοστά επιτυχούς πρόβλεψης με λιγότερο υλικό, σε σχέση με άλλες διατάξεις πρόβλεψης διακλαδώσεων, όπως αυτή των Yeh και Patt, που προτάθηκε το 1991. Η gshare βρίσκεται στο υποστάδιο F1 και περιλαμβάνει έναν πίνακα PHT 4096 θέσεων. Τα 12 bits της διεύθυνσης της δέσμης εντολών γίνονται XORed με το περιεχόμενο ενός σφαιρικού BHSR, μεγέθους 12 bits, και το αποτέλεσμα που προκύπτει χρησιμοποιείται για τη δεικτοδότηση του PHT. Το όνομα gshare προκύπτει από το g και το share. Επειδή ο πίνακας PHT είναι άμεσης απεικόνισης μπορεί να προκύψει ψευδωνυμία (share) και λόγω του ότι χρησιμοποιείται ένας σφαιρικός BHSR προστέθηκε στο όνομα το g.

Ανανέωση στον πίνακα PHT της gshare γίνεται όταν ολοκληρωθεί μια εντολή διακλάδωσης υπό συνθήκη. Στον καταχωρητή BHSR γίνεται ανανέωση στις εξής περιπτώσεις:

- Αν έχει δοθεί εντολή για εκκένωση (flush) της διοχέτευσης και έχει ολοκληρωθεί η εκτέλεση κάποιας εντολής διακλάδωσης υπό συνθήκη, τότε ανανεώνεται περιλαμβάνοντας στο δεξιότερό του bit την απόφαση για τη συγκεκριμένη διακλάδωση.
- Αν έχει δοθεί εντολή για εκκένωση της διοχέτευσης τότε ο BHSR ανανεώνεται με βάση την πιο πρόσφατη, έγκυρη πληροφορία του BOB.
- Αν η πιο πρόσφατα προσκομισθείσα δέσμη εντολών περιέχει μία εντολή διακλάδωσης υπό συνθήκη, τότε ο BHSR ανανεώνεται περιλαμβάνοντας στο δεξιότερό του bit την πρόβλεψη της gshare για τη συγκεκριμένη εντολή διακλάδωσης.

Κλείνοντας την παρουσίαση του μηχανισμού πρόβλεψης διακλαδώσεων του επεξεργαστή IVM, θα πρέπει να τονισθεί ότι η παραγωγή κάποιας πρόβλεψης από τη μεγάλη διάταξη πρόβλεψης διακλαδώσεων έχει νόημα και λαμβάνεται υπόψη μόνο όταν η δέσμη εντολών, που βρίσκεται στο υποστάδιο F1, περιέχει εντολή διακλάδωσης υπό συνθήκη.

3.1.3. Στάδιο F1

Στο στάδιο F1, εισέρχονται 16 εντολές που έχουν προσκομιστεί από την κρυφή μνήμη εντολών. Μέσω του κυκλώματος ευθυγράμμισης, επιλέγονται οι οκτώ κατάλληλες που θα προωθηθούν στα υπόλοιπα στάδια της διοχέτευσης για να εκτελεστούν. Επίσης, υπάρχει μία μεγάλη συνδυαστική λογική η οποία αποτελείται από οκτώ αποκωδικοποιητές διακλαδώσεων. Οι αποκωδικοποιητές αυτοί ελέγχουν τις εντολές της δέσμης ταυτόχρονα για να ανιχνευτεί αν είναι κάποια από αυτές εντολή διακλάδωσης.

Κάποιες από τις πληροφορίες που προκύπτουν από τους αποκωδικοποιητές διακλαδώσεων προωθούνται, μέσω ανάδρασης, στο υποστάδιο F0 για να δημιουργηθεί και να αποθηκευτεί μία νέα εγγραφή στο BTB. Οι πληροφορίες αυτές είναι: Ένα bit που δηλώνει αν θα πρέπει να γίνει εγγραφή στον

BTB (1) ή όχι (0), η θέση της εντολής διακλάδωσης μέσα στη δέσμη εντολών, ο τύπος της και η προβλεφθείσα διεύθυνση διακλάδωσης.

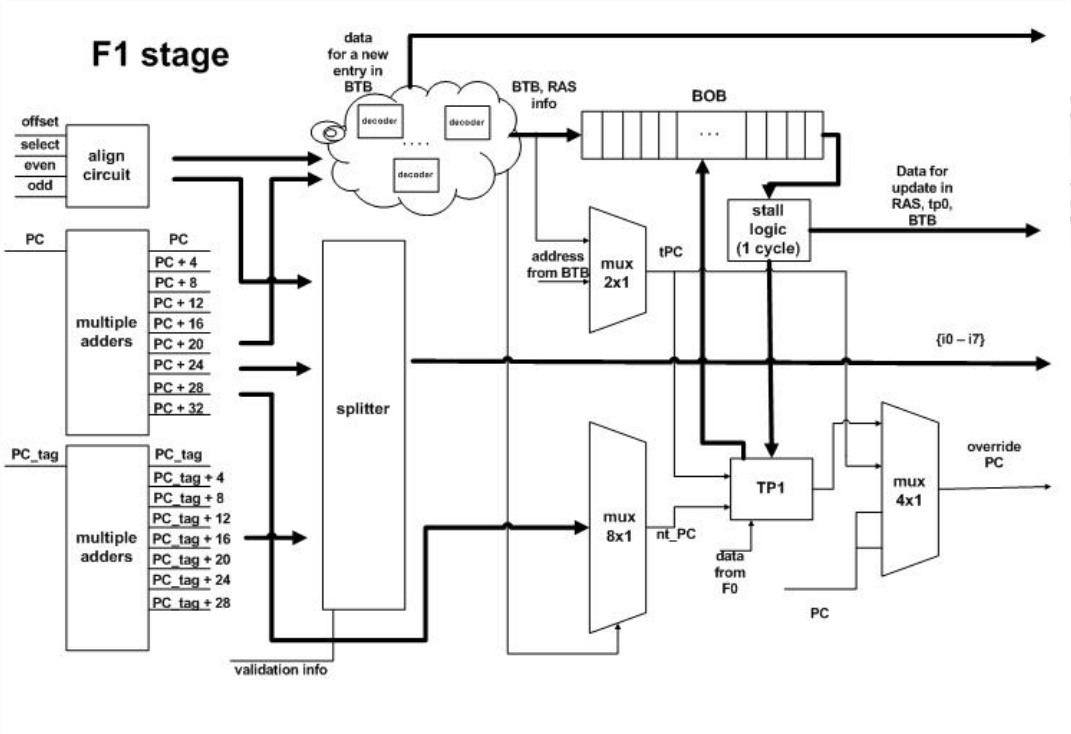
Επιπρόσθετα, στο υποστάδιο F1 συνεχίζεται η παραγωγή της πρόβλεψης από τη μεγάλη διάταξη πρόβλεψης διακλαδώσεων. Τα αποτελέσματα που παράγει είναι οι προβλέψεις για τη διακλάδωση και την επόμενη διεύθυνση προσκόμισης εντολών. Η επόμενη διεύθυνση προσκόμισης εντολών μπορεί να προκύψει από τις ακόλουθες περιπτώσεις:

- Αν η διακλάδωση έχει προβλεφθεί ότι δε λαμβάνεται, τότε ως επόμενη διεύθυνση ορίζεται η διεύθυνση της εντολής που ακολουθεί τη διακλάδωση.
- Αν η διακλάδωση έχει προβλεφθεί ότι λαμβάνεται τότε:
    - Αν η εντολή διακλάδωσης είναι επιστροφή από ρουτίνα, τότε σαν επόμενη διεύθυνση επιλέγεται αυτή που είναι αποθηκευμένη στην σχετική καταχώρηση του BTB, του σταδίου F0.

    - Αν η εντολή είναι οποιοδήποτε άλλο είδος διακλάδωσης, τότε επιλέγεται σαν επόμενη διεύθυνση προσκόμισης η διεύθυνση που παράγεται από τους αποκωδικοποιητές διακλαδώσεων.

Να σημειωθεί πως τα αποτελέσματα που εξάγονται από τη μεγάλη διάταξη πρόβλεψης διακλαδώσεων λαμβάνονται υπ' όψιν μόνο όταν στη δέσμη εντολών υπάρχει κάποια εντολή διακλάδωσης υπό συνθήκη και όταν οι προβλέψεις της μικρής και της μεγάλης διάταξης διακλαδώσεων είναι διαφορετικές. Τότε, λαμβάνεται υπόψη η πρόβλεψη της μεγάλης διάταξης που είναι και η πιο αξιόπιστη.

Οι πληροφορίες που παράγονται από τη μεγάλη διάταξη πρόβλεψης διακλαδώσεων μαζί με κάποιες άλλες που αφορούν την εντολή διακλάδωσης, μιας δέσμης εντολών, αποθηκεύονται στην προσωρινή μνήμη σειράς διακλαδώσεων (**BOB**-**B**ranch **O**rder **B**uffer). Ο BOB έχει μέγεθος (16 καταχωρίσεις)x(93 bit σε κάθε καταχώριση) = 1488 bits = 186 bytes = 0.18 KB και διατηρεί όλες τις εντολές διακλάδωσης που είναι σε εκτέλεση μέχρις ότου ολοκληρωθούν ή δοθεί εντολή για εκκένωση ή επαναφορά. Το σημαντικότερο πλεονέκτημα είναι ότι οι εντολές μπορούν να εκτελούνται με εικασία και να γίνεται ανάνηψη του συστήματος όταν προκύψει κάποια λανθασμένη πρόβλεψη διακλάδωσης. Στην περίπτωση αυτή πρέπει να ακυρωθεί το σύνολο των εντολών που προσκομίστηκαν και εκτελούνται με βάση αυτή την πρόβλεψη. Στον BOB δημιουργείται μια καινούρια καταχώρηση όταν βρεθεί από τους αποκωδικοποιητές διακλαδώσεων ότι υπάρχει μία εντολή διακλάδωσης. Οι πληροφορίες που αποθηκεύονται και αφορούν την εντολή διακλάδωσης είναι:



Εικόνα 3.10: Το στάδιο F1

- Η διεύθυνση της δέσμης των οκτώ εντολών, μέσα στην οποία βρίσκεται η εντολή διακλάδωσης.
- Αν θα πρέπει να γίνει ανανέωση ο choice predictor.
- Η πρόβλεψη του SAg.
- Το περιεχόμενο ενός BHSR που χρησιμοποιείται για δεικτοδότηση του PHT, του SAg.
- Η πρόβλεψη για τη συγκεκριμένη εντολή διακλάδωσης (από τη μεγάλη διάταξη πρόβλεψης διακλαδώσεων).
- Το περιεχόμενο του καθολικού BHSR που χρησιμοποιείται για δεικτοδότηση στον πίνακα PHT του gshare.
- Το δείκτη της στοίβας RAS.

Μία καταχώρηση αφαιρείται από τον BOB όταν ολοκληρωθεί κάποια εντολή διακλάδωσης. Επίσης, ο BOB αδειάζει όταν δοθεί εντολή για εκκένωση ή επαναφορά.

Η προσκόμιση των εντολών από την κρυφή μνήμη εντολών γίνεται με βάση τη διεύθυνση που προβλέπεται από το στάδιο F0. Οι περιπτώσεις όπου η προσκόμιση των εντολών γίνεται με βάση τη διεύθυνση που δίνεται από το στάδιο F1 είναι:

- Σε μία δέσμη εντολών να υπάρχει εντολή μετάβασης σε ή επιστροφής από ρουτίνα και να μην υπάρχει καμία σχετική πληροφορία αποθηκευμένη στον BTB. Τότε, δίνεται εντολή από το υποστάδιο F1 να σταματήσει η προβλεπόμενη προσκόμιση εντολών και να προσκομιστεί πάλι η ίδια δέσμη έτσι ώστε να δημιουργηθεί μία έγκυρη καταχώρηση στον BTB και να χρησιμοποιηθεί σωστά η στοίβα RAS.
- Σε μία δέσμη εντολών να υπάρχει εντολή διακλάδωσης χωρίς συνθήκη και να μην υπάρχει έγκυρη καταχώρηση στον BTB. Τότε δίνεται εντολή από το υποστάδιο F1 να συνεχιστεί η προσκόμιση των εντολών από τη διεύθυνση της διακλάδωσης, η οποία έχει εξαχθεί από τους αποκωδικοποιητές διακλαδώσεων.
- Η μεγάλη διάταξη διακλάδωσης να έχει παραγάγει μία πρόβλεψη, για μία εντολή διακλάδωσης υπό συνθήκη, που είναι διαφορετική από την πρόβλεψη της μικρής διάταξης διακλάδωσης. Τότε δίνεται εντολή να προσκομιστεί η επόμενη δέσμη εντολών με βάση τη διεύθυνση που έχει προβλεφθεί από τη μεγάλη διάταξη.

Τέλος, αν λαμβάνεται η διακλάδωση, το valid bit της εντολής διακλάδωσης αλλά και τα valid bit των εντολών που προηγούνται αυτής, τίθενται στο λογικό 1, που σημαίνει ότι οι συγκεκριμένες εντολές προβλέπεται να εκτελεστούν κανονικά. Αντίθετα, τα valid bit των εντολών που ακολουθούν την εντολή διακλάδωσης τίθενται στο λογικό 0 και ακυρώνονται επειδή λόγω της διακλάδωσης αλλάζει η ροή εκτέλεσης. Αν δεν υπάρχει εντολή διακλάδωσης μέσα στη δέσμη εντολών, τότε τα valid bit όλων τίθενται στο 1.

3.1.4. Στάδιο F2

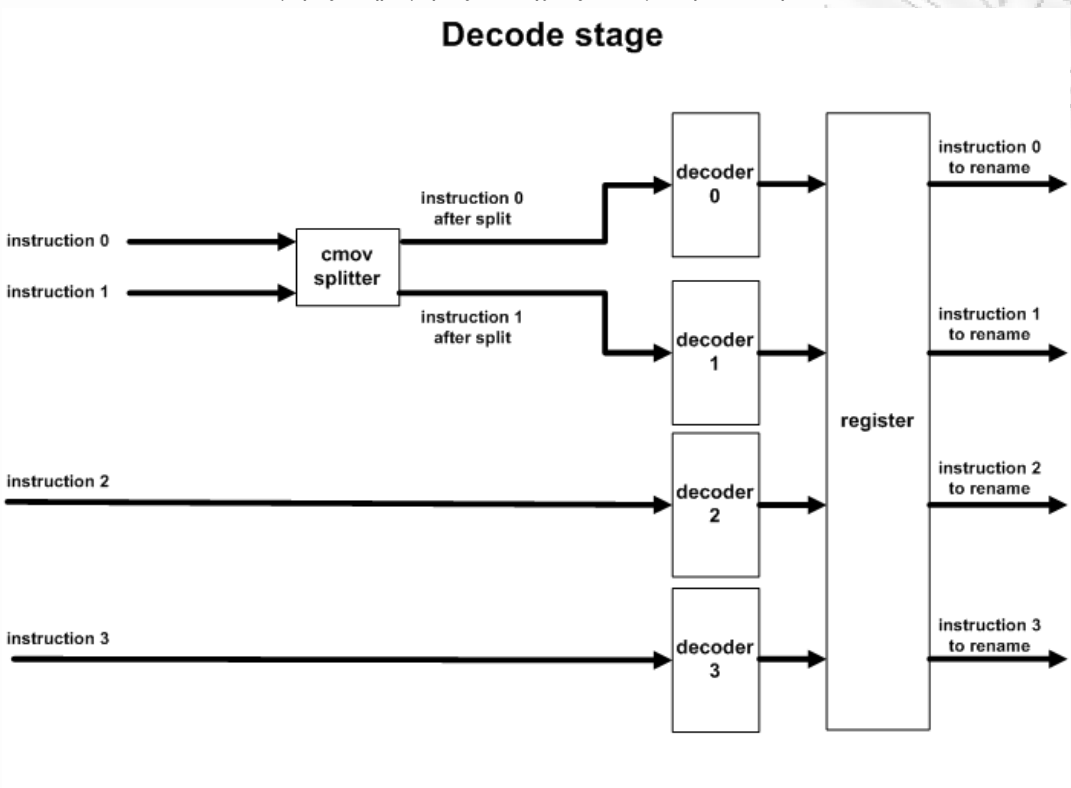
Σε αυτό το υποστάδιο εισάγονται σε μία προσωρινή μνήμη (buffer), η οποία αποτελείται από 32 θέσεις, το πολύ οκτώ εντολές που προέρχονται από το υποστάδιο F1. Ο ακριβής αριθμός των εντολών που αποθηκεύονται καθορίζεται από το ποιες είναι έγκυρες και ποιες όχι (δηλαδή από το valid bit τους). Επίσης από την προσωρινή μνήμη εξάγονται το πολύ τέσσερις εντολές.

Η προσωρινή μνήμη έχει υλοποιηθεί σαν μία ουρά FIFO, οπότε οι εντολές εισάγονται στο τέλος της ουράς και εξάγονται από την κεφαλή της. Θα πρέπει να σημειωθεί πως από το υποστάδιο αυτό αλλάζει και το πλάτος της παράλληλης διοχέτευσης. Ενώ μέχρι το στάδιο F1, το πλάτος ήταν οκτώ, τώρα γίνεται τέσσερα, αφού από την προσωρινή μνήμη εξάγονται το πολύ τέσσερις εντολές και προωθούνται στα υπόλοιπα στάδια της διοχέτευσης.

Ο αριθμός των εντολών που εξάγονται εξαρτάται από το συνολικό αριθμό αυτών που υπάρχουν στο buffer καθώς επίσης και από το είδος της εντολής. Για παράδειγμα αν υπάρχει η εντολή cmov (**c**onditional **mov**e - **cmov**), η οποία είναι ειδική περίπτωση εντολής και είναι πρώτη στην τετράδα εξαγωγής, τότε προωθείται μόνη της στο στάδιο αποκωδικοποίησης. Αν η εντολή αυτή είναι στην τετράδα εξαγωγής αλλά όχι πρώτη, τότε εξάγονται μόνο οι εντολές που προηγούνται της cmov. Σε κάθε άλλη περίπτωση εξάγονται από την τετράδα αυτή όλες οι εντολές μέχρι την πρώτη που δεν είναι έγκυρη. Ο λόγος που ακολουθείται η συγκεκριμένη πολιτική εξαγωγής των εντολών είναι επειδή πρέπει να εξέρχονται από το στάδιο F2 σε σειρά.

3.2. Στάδιο αποκωδικοποίησης (DECODE)

Σε αυτό το στάδιο αναγνωρίζονται οι μεμονωμένες εντολές, καθορίζονται οι τύποι τους και αποκωδικοποιούνται οι διάφορες πληροφορίες που σχετίζονται με την εντολή.



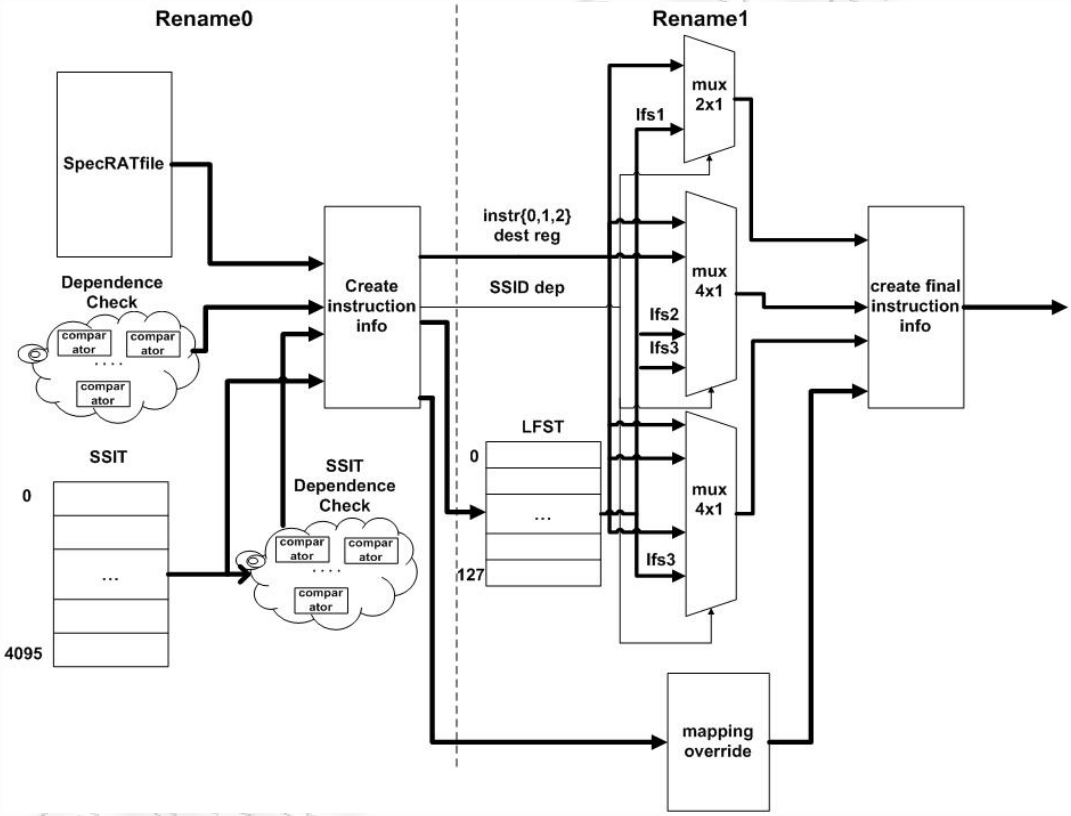
Εικόνα 3.11: Το στάδιο αποκωδικοποίησης (DECODE)

Όπως φαίνεται και από την εικόνα 3.11, το στάδιο DECODE περιέχει τέσσερις αποκωδικοποιητές, έναν για κάθε εντολή, έτσι ώστε η αποκωδικοποίηση να γίνεται παράλληλα (οι εντολές που εξέρχονται από το στάδιο F2 είναι τέσσερις). Ιδιαίτερη μέριμνα γίνεται για τις εντολές cmov. Η εντολή cmov εξέρχεται του υποσταδίου F2 μόνη της. Οπότε στο στάδιο DECODE, οι εντολές instruction 0 και instruction 1 περνάνε, πριν από τους αποκωδικοποιητές, από ένα ειδικό κύκλωμα που ανακαλύπτει αν υπάρχει εντολή cmov και κάνει τους κατάλληλους χειρισμούς για να μη δημιουργηθεί πρόβλημα.

3.3. Στάδιο μετονομασίας (RENAME)

Στο στάδιο RENAME καθορίζονται οι εξαρτήσεις μεταξύ των εντολών και ελέγχονται ποιες είναι ανεξάρτητες έτσι ώστε να διεκπεραιωθούν όσο το δυνατό νωρίτερα. Το βασικότερο πρόβλημα για τον καθορισμό των εξαρτήσεων είναι οι ψευδοεξαρτήσεις δεδομένων. Πιο συγκεκριμένα, υπάρχουν δύο ειδών ψευδοεξαρτήσεις: Η αντεξάρτηση (WAR – Write After Read ή anti-dependence) και η εξάρτηση εξόδου μεταξύ δύο εντολών (WAW – Write After Write ή output dependence). Η πρώτη συμβαίνει όταν η εντολή που ακολουθεί πρέπει να γράψει σε έναν καταχωρητή από τον οποίο πρέπει πρώτα να διαβάσει την τιμή του η εντολή που προηγείται. Η δεύτερη συμβαίνει όταν δύο εντολές γράφουν στον ίδιο καταχωρητή. Γενικά, η ύπαρξη των ψευδοεξαρτήσεων οφείλεται στην επαναχρησιμοποίηση των καταχωρητών (register reuse ή register recycling) από τους μεταγλωττιστές.

Για την εξάλειψη των ψευδοεξαρτήσεων, οι οποίες μπορεί να προκαλέσουν σημαντικές απώλειες στην απόδοση, χρησιμοποιούνται διάφορες τεχνικές μετονομασίας των καταχωρητών. Επίσης στο στάδιο RENAME, με χρήση των συνόλων αποθήκευσης, γίνεται πρόβλεψη και καθορισμός των εξαρτήσεων μεταξύ των εντολών μνήμης. Οπότε με βάση αυτές τις εξαρτήσεις γίνονται και οι αντίστοιχες μετονομασίες. Το στάδιο RENAME αποτελείται από δύο υποστάδια, τα **Rename0** και **Rename1**.



Εικόνα 3.12: Το στάδιο μετονομασίας (RENAME)

3.3.1. Υποστάδιο Rename0

Στο υποστάδιο Rename0 γίνονται οι ακόλουθες διεργασίες:

- Προσκόμιση από τον αρχιτεκτονικό πίνακα καταχωρητών με εικασία (speculative register architected table ή specRAT) των διευθύνσεων των φυσικών καταχωρητών που αντιστοιχούν στις διευθύνσεις των αρχιτεκτονικών καταχωρητών.
- Έλεγχος των εξαρτήσεων

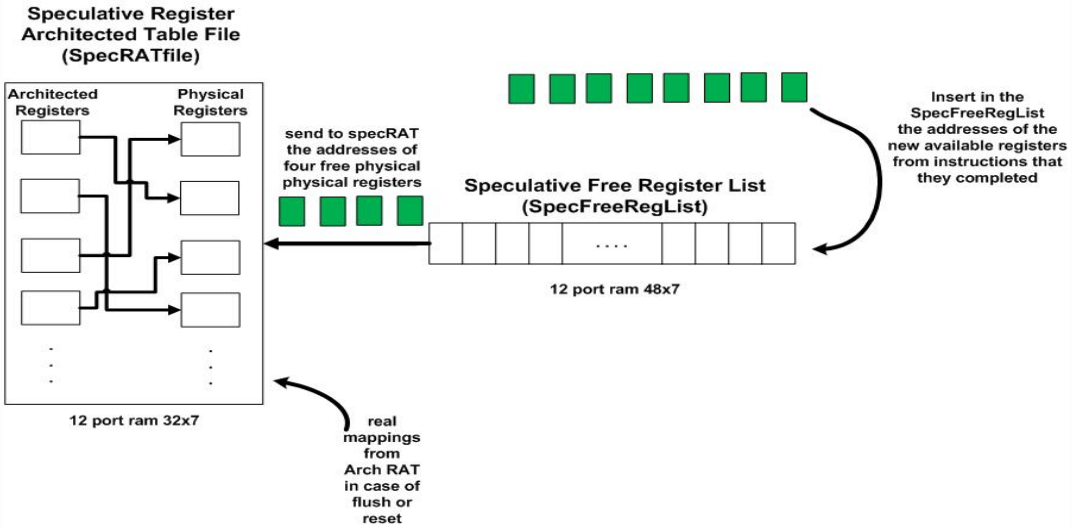
- Προσκόμιση των SSIDs των εντολών από τον πίνακα SSIT.
- Έλεγχος των εξαρτήσεων όσον αφορά τις εντολές μνήμης.

Ο αρχιτεκτονικός πίνακας καταχωρητών με εικασία (Speculative Register Architected Table File ή specRATfile) και η λίστα των ελεύθερων καταχωρητών με εικασία (Speculative Free Register List ή specFreeRegList)

Το κυριότερο πρόβλημα όσον αφορά τις εξαρτήσεις είναι οι ψευδοεξαρτήσεις δεδομένων (WARs και WAWs). Για την εξάλειψη τους χρησιμοποιείται ένας πιο επιθετικός τρόπος αντιμετώπισης που είναι η μετονομασία των καταχωρητών προορισμού των εντολών.

Οι καταχωρητές που είναι ορατοί στον προγραμματιστή είναι 32. Όμως αυτός ο αριθμός είναι εξαιρετικά μικρός για να μπορούν πολλές εντολές να εκτελούνται παράλληλα και εκτός σειράς. Συνεπώς, από τον επεξεργαστή διατίθεται ένα σύνολο 80 φυσικών καταχωρητών με τους οποίους αντιστοιχίζονται οι 32 καταχωρητές της αρχιτεκτονικής.

Για την υλοποίηση της τεχνικής της μετονομασίας χρησιμοποιείται ένα αρχείο που αποθηκεύει τις προσωρινές συνδέσεις μεταξύ των αρχιτεκτονικών και των φυσικών καταχωρητών. Το αρχείο αυτό υλοποιείται μέσω μίας μνήμης RAM 32x7, των 12 θυρών. Ο αριθμός των θυρών προκύπτει ως εξής: Από το στάδιο DECODE φτάνουν τέσσερις εντολές. Ο μέγιστος αριθμός καταχωρητών που διαθέτει κάθε εντολή είναι τρεις, δύο τελεστέοι και ένας καταχωρητής προορισμού. Οπότε θα πρέπει συνολικά να αποθηκευτούν(4) και να φορτωθούν(8) 12 φυσικές διευθύνσεις. Για τους τελεστέους των εντολών ισχύουν τα εξής: Προσκομίζονται από το αρχείο οι διευθύνσεις (ή τα ονόματα, εξάλλου αυτά τα δύο ταυτίζονται στη συγκεκριμένη υλοποίηση) των φυσικών καταχωρητών, οι οποίοι αντιστοιχούν στους αρχιτεκτονικούς καταχωρητές των εντολών. Η προσκόμιση γίνεται χρησιμοποιώντας ως διευθύνσεις τις δυαδικές κωδικοποιήσεις των αντίστοιχων αρχιτεκτονικών καταχωρητών.



Εικόνα 3.13: Το κύκλωμα αντιστοίχησης των διευθύνσεων των αρχιτεκτονικών καταχωρητών με τις διευθύνσεις των φυσικών

Όσες εντολές περιέχουν έναν αρχιτεκτονικό καταχωρητή προορισμού (για παράδειγμα μία εντολή πρόσθεσης ή μία εντολή φόρτωσης), τότε η αντιστοίχησή του γίνεται με έναν ελεύθερο, φυσικό καταχωρητή. Αυτή επιτυγχάνεται ως εξής: Η διεύθυνση του ελεύθερου, φυσικού καταχωρητή διατίθεται από μία λίστα (Speculative Free Register List ή SpecFreeRegList), η οποία λειτουργεί σαν ουρά και διαθέτει τις διευθύνσεις όλων των φυσικών καταχωρητών που δεν έχουν δεσμευτεί από εντολές. Έστω ότι μια εντολή διαθέτει έναν αρχιτεκτονικό καταχωρητή προορισμού με το όνομα Α. Τότε στο αρχείο, στη διεύθυνση Α, θα αποθηκευτεί η διεύθυνση του ελεύθερου, φυσικού καταχωρητή. Η λίστα υλοποιείται από μία RAM 48x7 των 12 θυρών. Από τη λίστα εξάγονται οι διευθύνσεις τεσσάρων, το πολύ, φυσικών καταχωρητών και εισάγονται σε αυτή το πολύ οκτώ διευθύνσεις νέων, ελεύθερων φυσικών καταχωρητών που έχουν αποδεσμευτεί από εντολές που έχουν ολοκληρωθεί.

Επιπρόσθετα, στο στάδιο Rename0 γίνεται έλεγχος των εξαρτήσεων μεταξύ των τεσσάρων εντολών. Οι εξαρτήσεις που ελέγχονται είναι οι RAWs και οι WAWs και χρησιμοποιούνται 18 συγκριτές γι' αυτόν τον σκοπό. Πιο αναλυτικά γίνεται:

1. Έλεγχος για εξάρτηση RAW της δεύτερης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι ο $1^{ος}$ τελεστέος της δεύτερης εντολής με τον καταχωρητή προορισμού της $1^{ης}$ εντολής.
2. Έλεγχος για εξάρτηση RAW της δεύτερης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι ο $2^{ος}$ τελεστέος της δεύτερης εντολής με τον καταχωρητή προορισμού της $1^{ης}$ εντολής.
3. Έλεγχος για εξάρτηση RAW της τρίτης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι ο $1^{ος}$ τελεστέος της τρίτης εντολής με τον καταχωρητή προορισμού της $1^{ης}$ εντολής.
4. Έλεγχος για εξάρτηση RAW της τρίτης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι ο $2^{ος}$ τελεστέος της τρίτης εντολής με τον καταχωρητή προορισμού της $1^{ης}$ εντολής.
5. Έλεγχος για εξάρτηση RAW της τέταρτης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι ο $1^{ος}$ τελεστέος της τέταρτης εντολής με τον καταχωρητή προορισμού της $1^{ης}$ εντολής.
6. Έλεγχος για εξάρτηση RAW της τέταρτης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι ο $2^{ος}$ τελεστέος της τέταρτης εντολής με τον καταχωρητή προορισμού της $1^{ης}$ εντολής.
7. Έλεγχος για εξάρτηση RAW της τρίτης εντολής με την δεύτερη. Ελέγχεται αν είναι ίδιοι ο $1^{ος}$ τελεστέος της τρίτης εντολής με τον καταχωρητή προορισμού της $2^{ης}$ εντολής.
8. Έλεγχος για εξάρτηση RAW της τρίτης εντολής με την δεύτερη. Ελέγχεται αν είναι ίδιοι ο $2^{ος}$ τελεστέος της τρίτης εντολής με τον καταχωρητή προορισμού της $2^{ης}$ εντολής.
9. Έλεγχος για εξάρτηση RAW της τέταρτης εντολής με την δεύτερη. Ελέγχεται αν είναι ίδιοι ο $1^{ος}$ τελεστέος της τέταρτης εντολής με τον καταχωρητή προορισμού της $2^{ης}$ εντολής.
10. Έλεγχος για εξάρτηση RAW της τέταρτης εντολής με την δεύτερη. Ελέγχεται αν είναι ίδιοι ο $2^{ος}$ τελεστέος της τέταρτης εντολής με τον καταχωρητή προορισμού της $2^{ης}$ εντολής.
11. Έλεγχος για εξάρτηση RAW της τέταρτης εντολής με την τρίτη. Ελέγχεται αν είναι ίδιοι ο $1^{ος}$ τελεστέος της τέταρτης εντολής με τον καταχωρητή προορισμού της $3^{ης}$ εντολής.
12. Έλεγχος για εξάρτηση RAW της τέταρτης εντολής με την τρίτη. Ελέγχεται αν είναι ίδιοι ο $2^{ος}$ τελεστέος της τέταρτης εντολής με τον καταχωρητή προορισμού της $3^{ης}$ εντολής.
13. Έλεγχος για εξάρτηση WAW της δεύτερης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι οι καταχωρητές προορισμού τους.
14. Έλεγχος για εξάρτηση WAW της τρίτης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι οι καταχωρητές προορισμού τους.
15. Έλεγχος για εξάρτηση WAW της τέταρτης εντολής με την πρώτη. Ελέγχεται αν είναι ίδιοι οι καταχωρητές προορισμού τους.
16. Έλεγχος για εξάρτηση WAW της τρίτης εντολής με την δεύτερη. Ελέγχεται αν είναι ίδιοι οι καταχωρητές προορισμού τους.
17. Έλεγχος για εξάρτηση WAW της τέταρτης εντολής με την δεύτερη. Ελέγχεται αν είναι ίδιοι οι καταχωρητές προορισμού τους.
18. Έλεγχος για εξάρτηση WAW της τέταρτης εντολής με την τρίτη. Ελέγχεται αν είναι ίδιοι οι καταχωρητές προορισμού τους.

Οι έξοδοι του κυκλώματος ελέγχου των εξαρτήσεων είναι δώδεκα, όσοι είναι δηλαδή και οι καταχωρητές συνολικά των τεσσάρων εντολών. Κάθε έξοδος έχει μέγεθος δύο bits και δείχνει για έναν καταχωρητή με ποιας εντολής τον καταχωρητή προορισμού είναι ίδιοι. Για παράδειγμα έστω ότι η έξοδος του $2^{ου}$ τελεστέου, της τρίτης εντολής είναι 00. Αυτό σημαίνει ότι ο συγκεκριμένος καταχωρητής είναι ίδιος με τον καταχωρητή προορισμού τις πρώτης εντολής (οι εντολές από 1 – 4 αριθμούνται στο δυαδικό σύστημα αρίθμησης από 00 – 11). Αν η έξοδος του $2^{ου}$ τελεστέου, της τρίτης εντολής ήταν 10, τότε αυτό σημαίνει πως ο καταχωρητής αυτός δεν είναι ίδιος με κανέναν καταχωρητή προορισμού των άλλων τριών εντολών.

Στο υποστάδιο Rename0 περιλαμβάνεται και ο πίνακας SSIT του μηχανισμού των συνόλων αποθήκευσης. Από τον πίνακα αυτόν προσκομίζονται τα SSID των εντολών και με βάση αυτά καθορίζονται οι εξαρτήσεις των εντολών μνήμης. Το κύκλωμα που ανακαλύπτει και παράγει τις εξαρτήσεις των εντολών μνήμης λειτουργεί σχεδόν με τον ίδιο τρόπο που λειτουργεί και το κύκλωμα που ελέγχει τις εντολές για εξαρτήσεις δεδομένων. Πιο συγκεκριμένα:

1. Αν τα SSID των εντολών 1 και 2[1] είναι έγκυρα και ίδια, πράγμα που σημαίνει ότι οι εντολές ανήκουν στο ίδιο σύνολο αποθήκευσης, τότε αν η εντολή 1 είναι εντολή αποθήκευσης δημιουργείται εξάρτηση της εντολής 2 από την 1.
2. Αν τα SSID των εντολών 1 και 3 είναι έγκυρα και ίδια, πράγμα που σημαίνει ότι οι εντολές ανήκουν στο ίδιο σύνολο αποθήκευσης, τότε αν η εντολή 1 είναι εντολή αποθήκευσης δημιουργείται εξάρτηση της εντολής 3 από την 1.
3. Αν τα SSID των εντολών 1 και 4 είναι έγκυρα και ίδια, πράγμα που σημαίνει ότι οι εντολές ανήκουν στο ίδιο σύνολο αποθήκευσης, τότε αν η εντολή 1 είναι εντολή αποθήκευσης δημιουργείται εξάρτηση της εντολής 4 από την 1.
4. Αν τα SSID των εντολών 2 και 3 είναι έγκυρα και ίδια, πράγμα που σημαίνει ότι οι εντολές ανήκουν στο ίδιο σύνολο αποθήκευσης, τότε αν η εντολή 2 είναι εντολή αποθήκευσης δημιουργείται εξάρτηση της εντολής 3 από την 2.
5. Αν τα SSID των εντολών 2 και 4 είναι έγκυρα και ίδια, πράγμα που σημαίνει ότι οι εντολές ανήκουν στο ίδιο σύνολο αποθήκευσης, τότε αν η εντολή 2 είναι εντολή αποθήκευσης δημιουργείται εξάρτηση της εντολής 4 από την 2.
6. Αν τα SSID των εντολών 3 και 4 είναι έγκυρα και ίδια, πράγμα που σημαίνει ότι οι εντολές ανήκουν στο ίδιο σύνολο αποθήκευσης, τότε αν η εντολή 3 είναι εντολή αποθήκευσης δημιουργείται εξάρτηση της εντολής 4 από την 3.

3.3.2. Υποστάδιο Rename1

Όπως αναφέρθηκε στην προηγούμενη ενότητα, στο υποστάδιο Rename0 γίνεται ο καθορισμός των εξαρτήσεων. Στο υποστάδιο Rename1 εξετάζεται ο τύπος των εντολών, οι εξαρτήσεις αυτών με τις υπόλοιπες και γίνεται η σωστή αντιστοίχηση των αρχιτεκτονικών καταχωρητών τους με τους φυσικούς καταχωρητές του επεξεργαστή.

Έστω για παράδειγμα μία εντολή πρόσθεσης. Όταν αυτή εισέλθει στο υποστάδιο Rename0, θα φορτωθούν από το SpecRATfile οι διευθύνσεις των φυσικών καταχωρητών που αντιστοιχούν στους τελεστέους της εντολής. Επίσης, για τον καταχωρητή προορισμού της εντολής, θα δοθεί η διεύθυνση ενός ελεύθερου φυσικού καταχωρητή από τη SpecFreeRegList. Παράλληλα, θα καθοριστούν οι εξαρτήσεις όλων των εντολών. Έστω ότι ο πρώτος τελεστέος της εντολής πρόσθεσης είναι το αποτέλεσμα μιας άλλης αριθμητικής εντολής που προηγείται. Στο στάδιο Rename1 θα ελεγχθεί αυτή η εξάρτηση με αποτέλεσμα η διεύθυνση του φυσικού καταχωρητή που θα αντιστοιχηθεί με τον 1ο τελεστέο να μην είναι αυτή που φορτώθηκε από το SpecRATfile αλλά να είναι η διεύθυνση του φυσικού καταχωρητή που αντιστοιχεί στον αρχιτεκτονικό καταχωρητή προορισμού της αριθμητικής εντολής που προηγείται.

Στο υποστάδιο Rename1 προσκομίζονται τα inums από τον πίνακα LFST, ο οποίος δεικτοδοτείται από τα SSID που εξήχθησαν στο υποστάδιο Rename0. Επίσης, με βάση μόνο τις εξαρτήσεις δεδομένων προσδιορίζονται οι διευθύνσεις των φυσικών καταχωρητών για τους αρχιτεκτονικούς καταχωρητές των εντολών. Αφού ολοκληρωθούν αυτές οι δύο λειτουργίες, συνδυάζονται οι εξαρτήσεις δεδομένων, οι εξαρτήσεις μνήμης και οι τύποι των εντολών έτσι ώστε να αντιστοιχηθούν σωστά οι αρχιτεκτονικοί καταχωρητές με τους φυσικούς καταχωρητές.

Συνοψίζοντας, ο λόγος ύπαρξης του σταδίου RENAME είναι να μετονομαστούν με τέτοιο τρόπο οι καταχωρητές των εντολών για να αποφευχθούν οι ανούσιες ψευδοεξαρτήσεις που προκαλούν μεγάλη απώλεια στην απόδοση του επεξεργαστή. Επίσης, στο στάδιο αυτό προστίθενται οι τελευταίες πληροφορίες που αφορούν τις εντολές και στον επόμενο κύκλο προωθούνται προς τα στάδια MEM και έκδοσης (ISSUE), ώστε να αρχίσει η εκτέλεσή τους. Επιπλέον, οι εντολές προωθούνται στον επόμενο κύκλο και στην προσωρινή μνήμη αναδιάταξης (**Re**-**O**rder **B**uffer – **ROB**), όπου παραμένουν αποθηκευμένες μέχρι να τελειώσει η εκτέλεσή τους για να ολοκληρωθούν σε σειρά και να αλλάξει σωστά η κατάσταση της μηχανής.

[1] Οι εντολές εισέρχονται σε σειρά από το υποστάδιο F2 του σταδίου FETCH στο υποστάδιο Rename0 του σταδίου RENAME. Οπότε εκτελούνται αντίστοιχα με τη σειρά που δηλώνει ο αύξων αριθμός τους, δηλαδή πρώτα εκτελείται η εντολή 1, μετά η εντολή 2 και ούτω καθεξής.

3.4. Στάδιο εκκίνησης (ISSUE)

Πριν αρχίσει η επεξήγηση του σταδίου έκδοσης καλό θα ήταν να αποσαφηνιστούν οι όροι διεκπεραίωση (dispatch) και εκκίνηση (issue). Με τον όρο διεκπεραίωση υπονοείται η συσχέτιση των τύπων των εντολών με τις διάφορες λειτουργικές μονάδες, ενώ ο όρος εκκίνηση σημαίνει η αρχικοποίηση της εκτέλεσης μιας εντολής στη λειτουργική μονάδα. Για λόγους που θα εξηγηθούν παρακάτω, στην αρχιτεκτονική του IVM οι όροι αυτοί ταυτίζονται.

Η διεκπεραίωση των εντολών (dispatching) είναι απαραίτητη στις υπερβαθμωτές διοχετεύσεις. Σε μία βαθμωτή διοχέτευση όλες οι εντολές, ανεξαρτήτως τύπου, περνούν μέσα από την ίδια διοχέτευση. Οι υπερβαθμωτές διοχετεύσεις όμως είναι ποικιλόμορφες και διαθέτουν πολλαπλές, ετερογενείς, λειτουργικές μονάδες. Συνεπώς, διαφορετικοί τύποι εντολών εκτελούνται από διαφορετικές μονάδες. Μόλις λοιπόν, αποκωδικοποιηθεί η εντολή και γίνει η απαραίτητη μετονομασία των καταχωρητών της, τότε πρέπει να δρομολογηθεί προς την κατάλληλη λειτουργική μονάδα για εκτέλεση.

Σε μία υπερβαθμωτή διοχέτευση, το είδος επεξεργασίας που γίνεται στα πρώτα στάδια διαφέρει σε σχέση με τα επόμενα. Ποιο συγκεκριμένα, στον IVM, από το στάδιο FETCH μέχρι και το RENAME ακολουθείται μία κεντρική επεξεργασία όλων των εντολών. Δηλαδή η προσκόμισή τους, η αποκωδικοποίησή τους και η μετονομασία των καταχωρητών τους γίνεται ακριβώς με τον ίδιο τρόπο για όλες τις εντολές.

Αντίθετα λόγω της ποικιλομορφίας, οι λειτουργικές μονάδες μπορούν να λειτουργούν κατανεμημένα και να εκτελούν τις εντολές ανεξάρτητα, εκτός βέβαια και αν υπάρχουν εξαρτήσεις μεταξύ των εντολών. Άρα, στο στάδιο EXECUTE αλλάζει το είδος της επεξεργασίας και από κεντρική γίνεται κατανεμημένη. Επομένως, λόγω αυτής της αλλαγής είναι χρήσιμη η παρεμβολή του σταδίου ISSUE, μεταξύ των σταδίων RENAME και EXECUTE.

Οι εντολές θα πρέπει να έχουν έτοιμα όλα τα δεδομένα των τελεστέων τους πριν εκτελεστούν. Στο στάδιο ISSUE και συγκεκριμένα στο υποστάδιο RegRead, οι τελεστέοι προσκομίζονται από το αρχείο των καταχωρητών. Παρόλα αυτά, υπάρχει πιθανότητα αυτοί οι τελεστέοι να μην είναι έτοιμοι διότι μπορεί προηγούμενες εντολές που τους ανανεώνουν να μην έχουν τελειώσει την εκτέλεσή τους ή να μην έχουν ολοκληρωθεί. Έτσι θα πρέπει οι εντολές που περιέχουν τους συγκεκριμένους τελεστέους να περιμένουν. Αυτό είναι και το μεγαλύτερο όφελος από την ύπαρξη του σταδίου ISSUE. Δηλαδή, το ότι περιέχει μία προσωρινή μνήμη αποθήκευσης όπου αποθηκεύονται οι εντολές μέχρις ότου ετοιμαστούν οι τελεστέοι τους. Αυτή η προσωρινή μνήμη αποθήκευσης ονομάζεται σταθμός κράτησης (reservation station) και αποσυνδέει το στάδιο RENAME από το στάδιο EXECUTE.

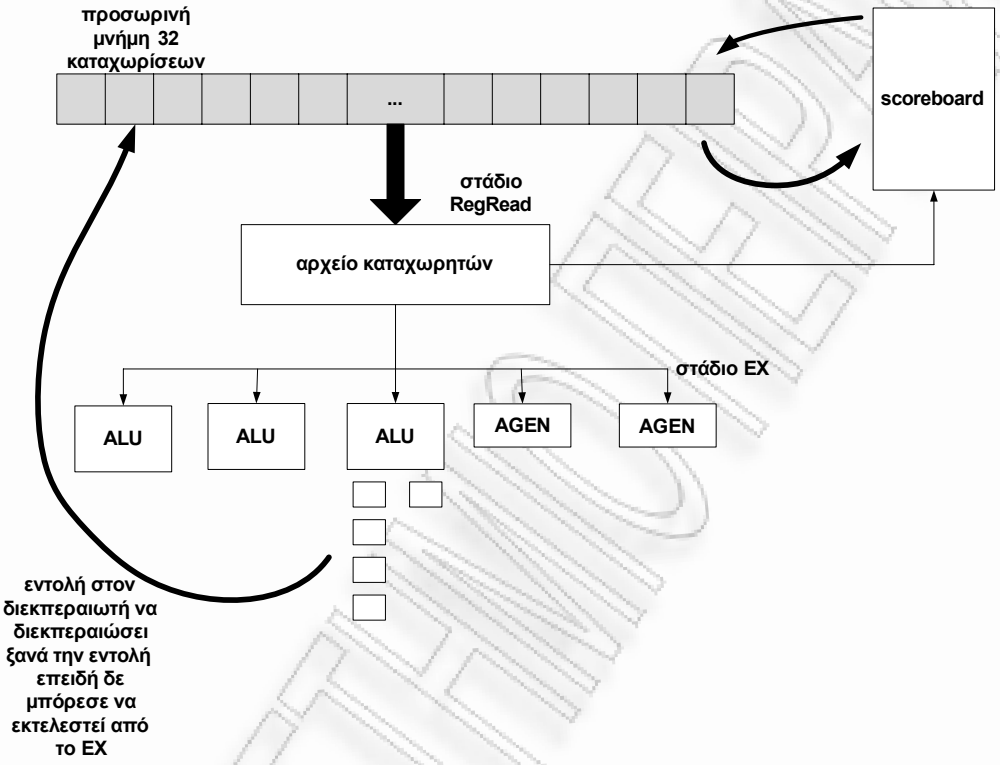
Γενικότερα, αν χρησιμοποιείται μία μοναδική προσωρινή μνήμη αποθήκευσης των εντολών πριν το στάδιο EXECUTE, τότε αυτή ονομάζεται κεντρικός σταθμός κράτησης (centralized reservation station). Αντίθετα, αν στο στάδιο EXECUTE, βρίσκονται πολλαπλές μνήμες προσωρινής αποθήκευσης, όπου κάθε μία αντιστοιχεί σε μία λειτουργική μονάδα τότε αυτές ονομάζονται κατανεμημένοι σταθμοί κράτησης (distributed reservation stations). Χρησιμοποιώντας το συγκεκριμένο μοντέλο, οι εντολές διεκπεραιώνονται (dispatched) από την προσωρινή μνήμη αποθήκευσης στους σταθμούς κράτησης των λειτουργικών μονάδων. Όταν οι τελεστέοι τους είναι έτοιμοι, τότε ξεκινούν.

Στην περίπτωση που χρησιμοποιείται το μοντέλο του κεντρικού σταθμού κράτησης, οι εντολές, μόλις ετοιμαστούν οι τελεστέοι τους, προωθούνται απευθείας στις λειτουργικές μονάδες. Συνεπώς, οι έννοιες της διεκπεραίωσης και της εκκίνησης ταυτίζονται. Ο κεντρικός σταθμός κράτησης έχει υιοθετηθεί από τον IVM και γι' αυτό το λόγο υπάρχει μόνο το στάδιο ISSUE και δεν υπάρχει στάδιο DISPATCH ή κάποιο άλλο παρεμφερές.

Το στάδιο ISSUE αποτελείται από τα υποστάδια Δρομολόγησης (Schedule) και Ανάγνωσης καταχωρητών (RegRead). Το υποστάδιο Schedule αναλαμβάνει την προσωρινή αποθήκευση των εντολών μέχρις ότου ετοιμαστούν οι τελεστέοι τους, καθώς επίσης και τη δρομολόγηση των εντολών στις κατάλληλες λειτουργικές μονάδες. Η κύρια λειτουργία του υποσταδίου RegRead είναι η διαχείριση του αρχείου καταχωρητών.

3.4.1. Υποστάδιο Δρομολόγησης (Schedule)

Αποτελείται από μία προσωρινή μνήμη αποθήκευσης (buffer) 32 θέσεων, στην οποία παραμένουν αποθηκευμένες οι εντολές μέχρις ότου εκκινηθούν/διεκπεραιωθούν στις λειτουργικές μονάδες για εκτέλεση.



Εικόνα 3.14: Τα υποστάδια Δρομολόγησης, RegRead και EXE

Η προσωρινή μνήμη αποθήκευσης λειτουργεί περίπου σαν ουρά FIFO, οπότε υπάρχουν δύο δείκτες, ο head και ο tail. Οι δείκτες αυτοί αλλάζουν σε κάθε ανοδική ακμή του ρολογιού. Πιο συγκεκριμένα, ο δείκτης head δείχνει στην εντολή που είναι υποψήφια να εξαχθεί από την ουρά και ο δείκτης tail δείχνει στη θέση που πρόκειται να εισαχθεί μια εντολή. Ο μέγιστος αριθμός των εντολών που μπορούν να εξαχθούν είναι έξι:

- Δύο απλές, αριθμητικές εντολές
- Δύο εντολές μνήμης
- Μία πολύπλοκη, αριθμητική εντολή (όπως για παράδειγμα πολλαπλασιασμός)
- Μία εντολή διακλάδωσης

Παρόλα αυτά ο τύπος του buffer δεν είναι ακριβώς FIFO διότι οι εντολές δεν εξάγονται σε σειρά. Σε μία τυπική ουρά FIFO θα εξαγόταν η εντολή που δεικτοδοτείται από το δείκτη head καθώς και οι πέντε επόμενες. Όμως στην προσωρινή μνήμη αποθήκευσης ελέγχονται όλες οι καταχωρήσεις και όσες εντολές πληρούν τις προϋποθέσεις ξεκινούν.

Για την εισαγωγή των εντολών ακολουθείται η γνωστή διαδικασία. Δηλαδή, από το στάδιο RENAME εισέρχονται τέσσερις εντολές και όποιες είναι έγκυρες, εισάγονται στις τελευταίες θέσεις της προσωρινής μνήμης και αυξάνεται αντίστοιχα ο δείκτης tail. Βέβαια, πριν αλλάξει ο δείκτης tail και εισαχθούν οι εντολές, ελέγχεται αν η προσωρινή μνήμη είναι γεμάτη ή αν πρόκειται να γεμίσει με την προσθήκη μιας, δύο, τριών ή τεσσάρων εντολών. Σε αυτή την περίπτωση, τίθεται ένα σήμα στο λογικό 1 και δεν επιτρέπεται η εισαγωγή καμιάς εντολής μέχρις ότου αδειάσει η προσωρινή μνήμη.

Ένας έλεγχος που γίνεται ακόμα είναι αν ξεκίνησαν οι εντολές με επιτυχία. Το στάδιο EXE στέλνει στο στάδιο ISSUE και συγκεκριμένα στο υποστάδιο Schedule πληροφορίες για τις έξι εντολές. Αν αυτές ξεκίνησαν με επιτυχία, τότε ελευθερώνονται οι αντίστοιχες καταχωρίσεις στην προσωρινή μνήμη, αλλιώς δίνεται εντολή να ξεκινήσουν ξανά μόλις είναι έτοιμες. Στην περίπτωση που γίνει reset, αδειάζει όλη η προσωρινή μνήμη.

Μια εντολή είναι έτοιμη να ξεκινήσει όταν:

- Είναι έγκυρη
- Δεν έχει ξεκινήσει προηγουμένως
- Είναι έτοιμοι οι τελεστέοι που χρειάζεται

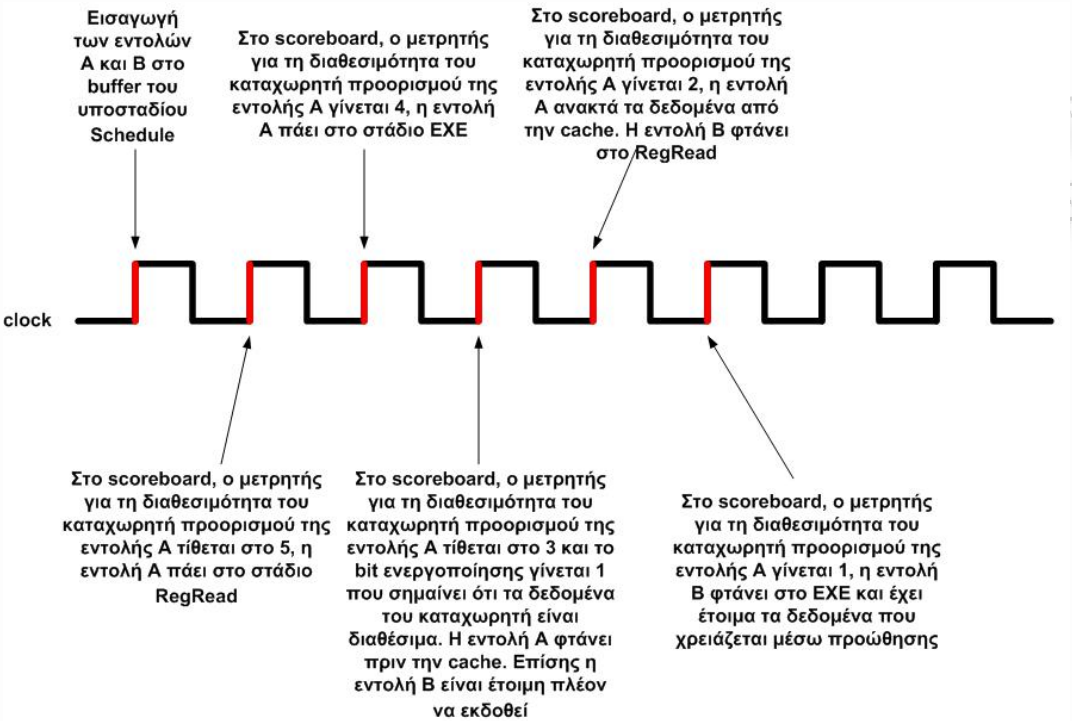
Το υποστάδιο Schedule επικοινωνεί και με το scoreboard. Το scoreboard είναι ένα κύκλωμα που παράγει πληροφορία για τη διαθεσιμότητα των δεδομένων ενός καταχωρητή. Είναι στην ουσία ένας πίνακας 80 καταχωρίσεων, όσοι είναι και οι φυσικοί καταχωρητές του επεξεργαστή, όπου κάθε καταχώρηση έχει τη μορφή που φαίνεται στην ακόλουθη εικόνα:

μετρητής (κύκλοι ρολογιού που έχουν μείνει μέχρι τη διαθεσιμότητά του)	**bit ενεργοποίησης (δηλώνει αν είναι έτοιμα τα δεδομένα του καταχωρητή ή όχι)**

Εικόνα 3.15: Μία καταχώριση του scoreboard

Στο αριστερό πεδίο είναι ένας μετρητής που με την τιμή του δείχνει πόσοι κύκλοι μένουν μέχρι να γίνουν διαθέσιμα τα δεδομένα του καταχωρητή. Το δεξιό πεδίο περιλαμβάνει ένα bit ενεργοποίησης (activation bit) και δείχνει αν είναι έτοιμα τα δεδομένα του καταχωρητή ή όχι. Τα δεδομένα δε θεωρούνται διαθέσιμα μόνο όταν είναι αποθηκευμένα στον αντίστοιχο καταχωρητή αλλά και στην εξής περίπτωση: Όταν μια εντολή βρίσκεται στο υποστάδιο Schedule και εκτιμηθεί ότι θα είναι έτοιμος ο τελεστέος της όταν αυτή θα βρίσκεται στο στάδιο EXE, τότε το bit ενεργοποίησης τίθεται στο 1 και η εντολή μπορεί να ξεκινήσει κανονικά. Δηλαδή, αν εκτιμηθεί ότι η εντολή θα έχει έτοιμα τα δεδομένα των καταχωρητών που χρειάζεται στο στάδιο EXE τότε δεν είναι ανάγκη να περιμένει μέχρι τα δεδομένα αυτά να αποθηκευτούν στους αντίστοιχους καταχωρητές αλλά συνεχίζει και τα βρίσκει στο στάδιο που πραγματικά τα χρειάζεται.

Ένα παράδειγμα θα κάνει ακόμα πιο κατανοητή αυτή την περίπτωση. Έστω μία εντολή φόρτωσης Α και μία εντολή πρόσθεσης Β, όπου ο ένας τελεστέος της Β είναι το αποτέλεσμα της εντολής Α ενώ ο άλλος δεν εξαρτάται από κάποια εντολή. Χωρίς απώλεια της γενικότητας, έστω ότι οι δύο εντολές είναι συνεχόμενες και ότι η Β είναι η αμέσως επόμενη εντολή από την Α. Επίσης, έστω ότι η Α δεν έχει εξάρτηση με καμία εντολή και ότι δε θα προκαλέσει αστοχία στην κρυφή μνήμη η προσκόμιση του επιθυμητού δεδομένου.



Εικόνα 3.16: Χρονική ανάλυση που δείχνει πότε η εντολή Β θα πάρει τα δεδομένα που χρειάζεται από την εντολή Α από την οποία εξαρτάται

Όπως φαίνεται και από την χρονική ανάλυση που απεικονίζεται στην εικόνα 3.16 το αποτέλεσμα της εντολής Α, από τη στιγμή που αυτή θα εισαχθεί στο υποστάδιο Schedule, θα είναι έτοιμο και θα μπορέσει να προωθηθεί σε 5 κύκλους (οι 5 κύκλοι προκύπτουν λόγω του τρόπου που ο IVM επεξεργάζεται τις εντολές μνήμης και επικοινωνεί με την κρυφή μνήμη). Δηλαδή τη στιγμή που η εντολή Β εισέρχεται στο στάδιο EXE ο τελεστέος της είναι έτοιμος και έτσι μπορεί να γίνει από την αριθμητική λογική μονάδα η πράξη της πρόσθεσης.

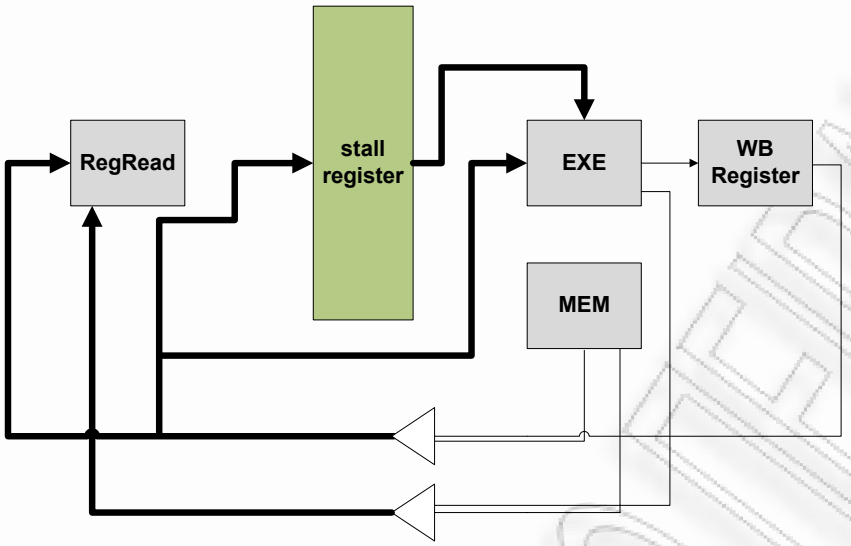
Τέλος, να σημειωθεί ότι το κύκλωμα scoreboard σαν έξοδο έχει ένα διάνυσμα των 80 bits, όπου κάθε bit αντιστοιχεί σε ένα φυσικό καταχωρητή. Όταν τα bits είναι στο λογικό 1, τότε αυτό σημαίνει ότι τα δεδομένα των αντίστοιχων φυσικών καταχωρητών είναι διαθέσιμα ενώ όταν τα bits είναι στο λογικό 0 αυτό σημαίνει ότι τα δεδομένα των φυσικών καταχωρητών δεν είναι διαθέσιμα.

3.4.2. Υποστάδιο Ανάγνωσης καταχωρητών (RegRead)

Στο υποστάδιο αυτό γίνονται οι ακόλουθες λειτουργίες:

- Διαχείριση του αρχείου των φυσικών καταχωρητών που περιέχει τα δεδομένα τους
- Προσκόμιση των τελεστέων των εντολών ώστε να είναι έτοιμες προς εκτέλεση στο στάδιο EXE
- Ελέγχει τις προωθήσεις που πρέπει να γίνουν για τους τελεστέους των εντολών
- Για τις αριθμητικές εντολές που απαιτείται πράξη με άμεσο τελεστέο, στο στάδιο αυτό υπολογίζεται ο άμεσος τελεστέος

Τα στάδια από τα οποία εξάγονται αποτελέσματα εντολών και κατ' επέκταση δεδομένα καταχωρητών, είναι το EXE και το MEM (λόγω των εντολών φόρτωσης). Για το στάδιο EXE συγκεκριμένα, τα δεδομένα που παράγονται από την εκτέλεση των εντολών αποθηκεύονται σε έναν καταχωρητή (WB) και εξάγονται στην επόμενη ανοδική ακμή του ρολογιού. Αυτό γίνεται κυρίως για λόγους συγχρονισμού. Όταν παραχθούν τα αποτελέσματα, προωθούνται μέσω μηχανισμού ανάδρασης πίσω στο στάδιο RegRead για να ανανεωθεί το αρχείο καταχωρητών και στο στάδιο EXE. Επίσης αποθηκεύονται και σε έναν άλλο καταχωρητή έτσι ώστε να είναι διαθέσιμα στο στάδιο EXE, με έναν κύκλο καθυστέρηση.



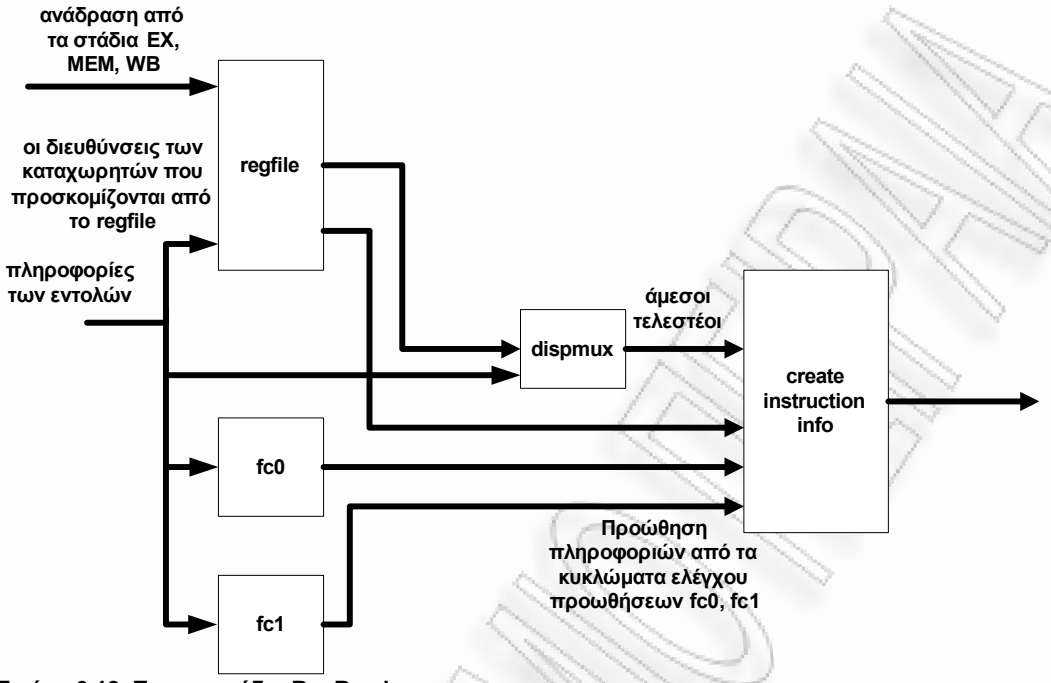
Εικόνα 3.17: Ο μηχανισμός των προωθήσεων δεδομένων του IVM

Η βασικότερη λειτουργία στο υποστάδιο RegRead είναι η ανάγνωση και η ανανέωση των δεδομένων των φυσικών καταχωρητών. Τα δεδομένα τους είναι αποθηκευμένα σε ένα αρχείο. Το αρχείο αυτό δέχεται σαν είσοδο τις διευθύνσεις των φυσικών καταχωρητών και γίνεται είτε εγγραφή (ανανέωση) είτε ανάγνωση των περιεχομένων τους.

Πιο συγκεκριμένα, το κύκλωμα που διαχειρίζεται το αρχείο καταχωρητών δέχεται σαν είσοδο ένα διάνυσμα το οποίο περιέχει όλες τις διευθύνσεις των φυσικών καταχωρητών που αποτελούν τους τελεστέους των εντολών. Με το διάνυσμα αυτό γίνεται ανάγνωση των τελεστέων. Επίσης, το αρχείο δέχεται σαν είσοδο τις τιμές των καταχωρητών προορισμού των εντολών που έχουν εξέλθει του σταδίου EXE και τις διευθύνσεις αυτών έτσι ώστε να ανανεωθούν οι αντίστοιχοι φυσικοί καταχωρητές. Επιπροσθέτως, δέχεται ένα διάνυσμα που έχει μέγεθος 80 bits, όσος και ο αριθμός των φυσικών καταχωρητών. Το διάνυσμα αυτό τροποποιείται όταν εξέλθουν τέσσερις εντολές από το στάδιο RENAME, όπου γίνεται ο εξής έλεγχος: Αν μία εντολή χρησιμοποιεί καταχωρητή προορισμού και είναι έγκυρη τότε το αντίστοιχο bit τίθεται στο λογικό 1. Το διάνυσμα αυτό αποτελεί στην ουσία μία μάσκα που χρησιμοποιείται στη δημιουργία ενός άλλου διανύσματος που απεικονίζει την κατάσταση των καταχωρητών (1 αν έχει έτοιμα τα δεδομένα του, 0 σε αντίθετη περίπτωση). Τέλος, το αρχείο έχει σαν είσοδο ένα διάνυσμα που δείχνει ποιες από τις εντολές που εξέρχονται από το στάδιο EXE και από το στάδιο MEM είναι εντολές που γράφουν σε καταχωρητή.

Οι σημαντικότεροι έξοδοι του κυκλώματος που διαχειρίζεται το αρχείο των καταχωρητών είναι ένα διάνυσμα με τις τιμές των τελεστέων των εντολών, ένα διάνυσμα με τις τιμές όλων των φυσικών καταχωρητών του επεξεργαστή, ένα διάνυσμα που δείχνει τη διαθεσιμότητα των δεδομένων των καταχωρητών. Τέλος, το αρχείο των καταχωρητών είναι μία RAM 80x64 bits (80 το σύνολο των καταχωρητών, 64 bits το μέγεθος καθενός), 17 θυρών. Από τις 17 θύρες, οι 11 χρησιμοποιούνται για την ανάγνωση των τελεστέων και οι άλλες 6 για την αποθήκευση των δεδομένων των καταχωρητών προορισμού.

Στο υποστάδιο RegRead υπάρχουν δύο κυκλώματα ελέγχου προωθήσεων. Τα κυκλώματα αυτά ελέγχουν αν χρειάζεται να γίνει προώθηση δεδομένου κάποιου καταχωρητή προορισμού (μιας εντολής που έχει ήδη εκτελεστεί) σε έναν τελεστέο μιας εντολής που πρόκειται να εισαχθεί στο στάδιο EXE στον επόμενο κύκλο ρολογιού. Ο τρόπος με τον οποίο γίνεται αυτό είναι ο ακόλουθος: Αρχικά κάθε τελεστέος εξετάζεται αν είναι ίδιος με κάποιον από τους καταχωρητές προορισμού. Αν είναι ίδιος και ο καταχωρητής προορισμού είναι έγκυρος, τότε σημειώνεται να γίνει προώθηση της τιμής του καταχωρητή προορισμού στον καταχωρητή τελεστέο όταν η εντολή εισέλθει στο στάδιο EXE.



Εικόνα 3.18: Το υποστάδιο RegRead

Ο ένας ελεγκτής προωθήσεων (εικόνα 3.18, fc1) ελέγχει αν πρέπει να προωθηθούν στους τελεστέους των εντολών, που πρόκειται να εισαχθούν στο στάδιο EXE, τα δεδομένα των καταχωρητών προορισμού των εντολών που εξέρχονται του καταχωρητή WB, ενώ ο άλλος (εικόνα 3.18, fc0) ελέγχει αν πρέπει να προωθηθούν στους τελεστέους των εντολών, που πρόκειται να εισαχθούν στο στάδιο EXE, τα δεδομένα των καταχωρητών προορισμού των εντολών που εξέρχονται από το στάδιο EXE. Αν βρεθεί από τον ελεγκτή fc0 ότι πρέπει να γίνουν προωθήσεις, οι σχετικοί τελεστέοι των εντολών θα πάρουν τα επιθυμητά δεδομένα από την έξοδο του καταχωρητή WB, στον επόμενο κύκλο ρολογιού και αν βρεθεί από τον ελεγκτή fc1 ότι πρέπει να γίνουν προωθήσεις τότε οι σχετικοί τελεστέοι των εντολών θα πάρουν τα επιθυμητά δεδομένα από την έξοδο του καταχωρητή καθυστέρησης (βλέπε Εικόνα 3.17, stall register).

Τέλος, στο υποστάδιο RegRead δημιουργείται ο άμεσος τελεστέος των εντολών, οι οποίες είναι πράξεις τιμών καταχωρητών με άμεσους τελεστέους, έτσι ώστε όταν αυτού του είδους οι εντολές εισέλθουν στο στάδιο EXE να έχουν έτοιμους τους τελεστέους και να εκτελεστεί η πράξη χωρίς καμία επιπρόσθετη καθυστέρηση. Οι μόνες εντολές που έχουν πιθανότητα να διαθέτουν άμεσους τελεστέους είναι αυτές που προορίζονται για τις δύο απλές αριθμητικές λογικές μονάδες και αυτή που προορίζεται για την πολύπλοκη αριθμητική λογική μονάδα. Έτσι από το κύκλωμα ελέγχονται τα δυαδικά των συγκεκριμένων εντολών και ανάλογα την εντολή γίνονται και οι προβλεπόμενες ενέργειες. Για παράδειγμα αν μία εντολή είναι η AND, τότε ο άμεσος τελεστέος έχει μέγεθος 64 bits και είναι τα τελευταία 16 bit της εντολής επεκταμένα προς τα αριστερά με βάση την τιμή του 16^{ου} bit. Αν για παράδειγμα είναι μία εντολή LDA, τότε σαν άμεσος τελεστέος ορίζονται τα τελευταία 16 bits της εντολής επεκταμένα προς τα δεξιά κατά 16 μηδενικά bits. Στο σύνολο αυτό, των 32 bits, γίνεται επέκταση πρόσημου κατά 32 bits ακόμη, για να δημιουργηθεί μία τιμή των 64 bit.

3.5. Στάδιο μνήμης (MEM)

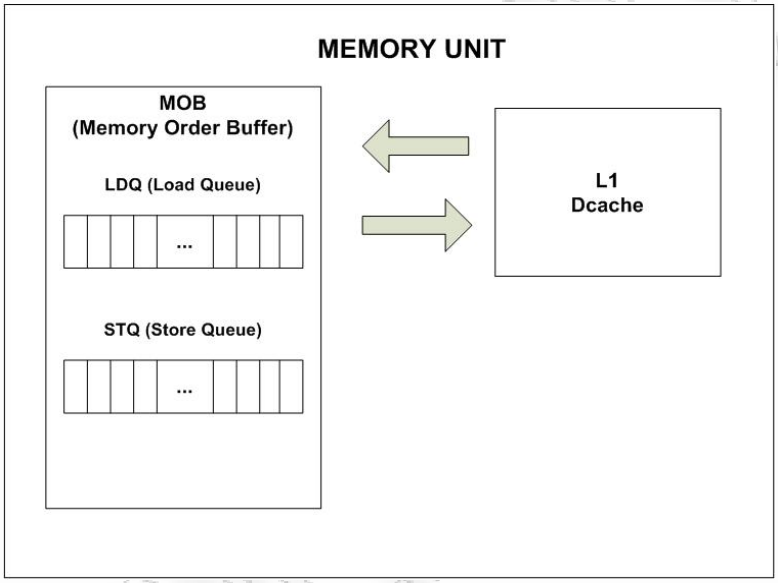
Οι εντολές μνήμης είναι πολύ χρήσιμες για τη μεταφέρουν δεδομένα μεταξύ της κύριας μνήμης και του αρχείου καταχωρητών ενός επεξεργαστή. Υποστηρίζουν σημαντικά την εκτέλεση αριθμητικών και λογικών εντολών διότι οι τελεστέοι που χρειάζεται η αριθμητική λογική μονάδα πρέπει πρώτα να φορτωθούν από τη μνήμη.

Οι εντολές μνήμης εκτελούνται σε τρία στάδια: Πρώτον παράγεται η εικονική διεύθυνση μνήμης, δεύτερον μεταφράζεται η εικονική διεύθυνση στην αντίστοιχη φυσική της και τρίτον γίνεται προσκόμιση/αποθήκευση των επιθυμητών δεδομένων από και προς τη μνήμη. Να σημειωθεί ότι το δεύτερο στάδιο, όσον αφορά τον επεξεργαστή IVM, παραλείπεται διότι δεν υποστηρίζεται ο μηχανισμός εικονικής μνήμης, ούτε υπάρχει κάποιο TLB.

Για μία εντολή φόρτωσης που παραμένει αποθηκευμένη στην προσωρινή μνήμη, στο υποστάδιο δρομολόγησης, μόλις γίνουν διαθέσιμα τα δεδομένα του καταχωρητή βάσης, ο οποίος χρησιμοποιείται στον υπολογισμό της διεύθυνσης προσκόμισης, τότε η εντολή ξεκινάει έτσι ώστε να υπολογιστεί και να δημιουργηθεί η διεύθυνση μνήμης. Στη συνέχεια γίνεται προσκόμιση των δεδομένων από την κρυφή μνήμη και η εντολή φόρτωσης θεωρείται ότι έχει τελειώσει την εκτέλεσή της. Μόλις η εντολή φόρτωσης εξέλθει από την προσωρινή μνήμη αναδιάταξης (ROB) τότε έχει ολοκληρωθεί (completed).

Αντίθετα, ο IVM επεξεργάζεται τις εντολές αποθήκευσης με διαφορετικό τρόπο. Μία εντολή αποθήκευσης θεωρείται ότι έχει τελειώσει την εκτέλεσή της όταν παραχθεί επιτυχώς η διεύθυνση μνήμης. Όταν ολοκληρωθεί εισάγεται σε μία ουρά όπου περιμένει μέχρι να έρθει η σειρά της να αλλάξει την κατάσταση της μνήμης. Όταν γίνει και αυτό η εντολή αποθήκευσης θεωρείται ως αποσυρθείσα (retired).

Το στάδιο MEM οργανώνεται χωρικά ως εξής: Υπάρχει μία προσωρινή μνήμη διάταξης εντολών μνήμης (Memory Order Buffer – MOB) η οποία επικοινωνεί με την κρυφή μνήμη δεδομένων. Πριν παρουσιαστεί όμως η πλήρης λειτουργία του συγκεκριμένου σταδίου είναι προτιμότερο να παρουσιαστούν οι επιμέρους οντότητες.



Εικόνα 3.19: Μία αφαιρετική αναπαράσταση του σταδίου MEM

3.5.1. Προσωρινή μνήμη διάταξης εντολών μνήμης (Memory Order Buffer – MOB)

Στο στάδιο MEM υπάρχει μία προσωρινή μνήμη η οποία χρησιμοποιείται για τη διατήρηση της σωστής σειράς και εκτέλεσης των εντολών μνήμης. Αυτή η προσωρινή μνήμη λέγεται MOB και χάρη σε αυτή οι εντολές φόρτωσης μπορούν να εκτελούνται εκτός σειράς. Αυτό έχει σαν αποτέλεσμα τη μεγιστοποίηση της απόδοσης λειτουργίας του επεξεργαστή.

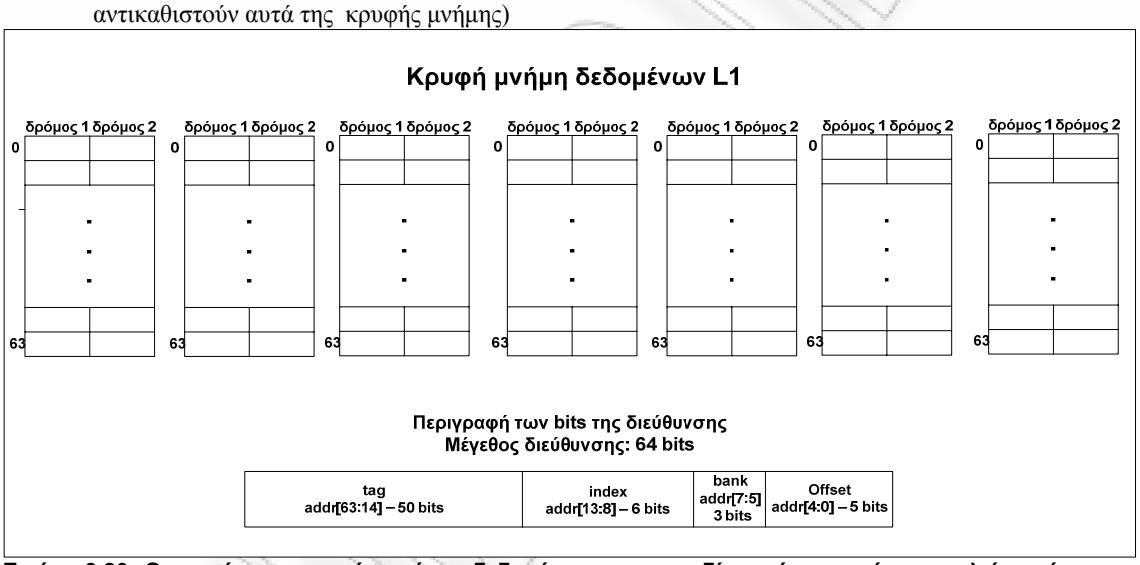
Το MOB αποτελείται από δύο ουρές, τις **LDQ** (**L**oa**D** **Q**ueue) και **STQ** (**ST**ore **Q**ueue). Η LDQ είναι η ουρά που κρατάει όλες τις εντολές φόρτωσης που είναι υπό επεξεργασία. Η περίπτωση κατά την οποία μία εντολή φόρτωσης φεύγει από την ουρά είναι όταν ολοκληρωθεί. Η STQ είναι η ουρά που περιλαμβάνει τις εντολές αποθήκευσης που έχουν τελειώσει την εκτέλεσή τους και αυτές που έχουν

ολοκληρωθεί. Στην ουσία, η ουρά που περιέχει τις ολοκληρωμένες εντολές αποθήκευσης που πρόκειται να αλλάξουν την κατάσταση της μνήμης και να αποσυρθούν είναι ενσωματωμένη στην STQ.

3.5.2. Η κρυφή μνήμη δεδομένων L1

Τα χαρακτηριστικά της κρυφής μνήμης δεδομένων δύο εισόδων, του επεξεργαστή IVM είναι τα ακόλουθα:

- 32 bytes ανά γραμμή cache (κάθε byte προσπελάζεται με τα τελευταία 5 bits της διεύθυνσης - address[4:0])
- 8 σειρές (η επιλογή μιας σειράς καθορίζεται από τα bits της διεύθυνσης [7:5])
- Συνολοσυσχετιστική δύο δρόμων (2 way set-associative)
- 64 γραμμές για κάθε δρόμο
- Συνολικό μέγεθος: 8 σειρές x 64 σύνολα σε κάθε σειρά x 2 δρόμοι σε κάθε σύνολο x 32 bytes σε κάθε δρόμο = 32768 bytes = 32KB
- Πολιτική αντικατάστασης LRU (δεν παίζει βέβαια κάποιο ρόλο διότι δεν υπάρχει υλοποιημένη κάποια δευτερεύουσα ή κύρια μνήμη από την οποία θα προσκομίζονται blocks και θα αντικαθιστούν αυτά της κρυφής μνήμης)



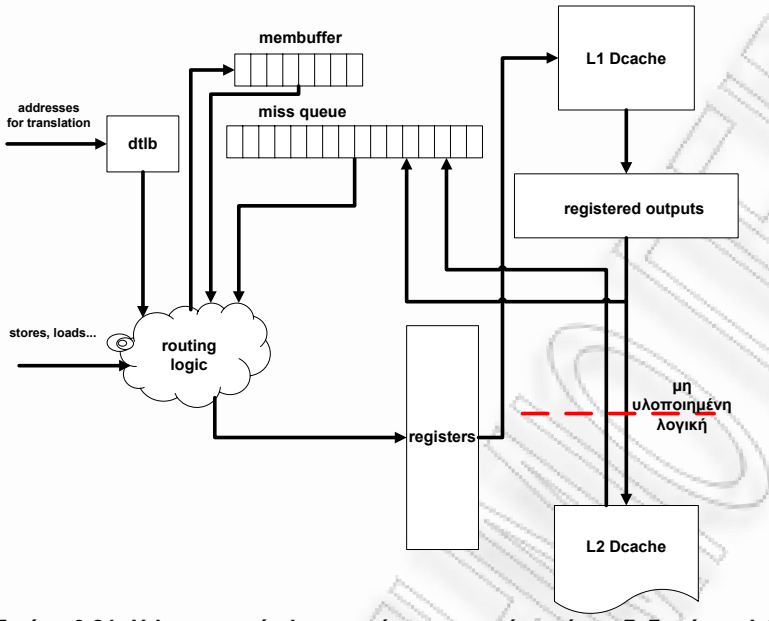
Εικόνα 3.20: Οι σειρές της κρυφής μνήμης δεδομένων και τα πεδία από τα οποία αποτελείται μία διεύθυνση μνήμης

Εσωτερικά η λειτουργία της κρυφής μνήμης είναι πιο περίπλοκη. Στο συνολικό κύκλωμα που την περιστοιχίζει, περιλαμβάνεται μια προσωρινή μνήμη, η οποία κρατάει όσες εντολές δεν έχουν εξυπηρετηθεί και μία ουρά αστοχιών, η οποία κρατάει τις εντολές που έχουν προκαλέσει αστοχία.

Η προσωρινή μνήμη αποτελείται από οκτώ καταχωρίσεις και μπορεί να αποθηκεύσει ταυτόχρονα μέχρι και δύο εντολές οι οποίες περιμένουν να εξυπηρετηθούν από την κρυφή μνήμη. Παρόλο που οι σχεδιαστές του συγκεκριμένου επεξεργαστή την ονόμασαν έτσι, η δομή αυτή λειτουργεί σαν ουρά FIFO. Οι νέες καταχωρήσεις αποθηκεύονται στο τέλος της ουράς και η προσωρινή μνήμη βγάζει σαν έξοδο την πιο παλιά καταχωρημένη εντολή, αυτή δηλαδή που βρίσκεται στην κεφαλή της ουράς. Η προσωρινή μνήμη δίνει σήμα καθυστέρησης της ροής των εντολών όταν έχει το πολύ μία άδεια θέση.

Η ουρά αστοχιών αποτελείται από 16 καταχωρίσεις. Επικοινωνεί με τη δευτερεύουσα μνήμη (να τονισθεί ότι δευτερεύουσα μνήμη δεν υπάρχει απλά έχει δημιουργηθεί η διασύνδεση) για να παραλαμβάνει τα μπλοκς που προορίζονται για αντικατάσταση των αντίστοιχων στη κρυφή μνήμη. Παράλληλα, για λόγους συγχρονισμού, τα μπλοκς αυτά τα κρατάει για δύο κύκλους. Η ουρά εξάγει δεδομένα με την ακόλουθη προτεραιότητα: Αν έλθει ένα καινούριο μπλοκ από τη δευτερεύουσα μνήμη (προφανώς θα πρέπει να έχει σημειωθεί ευστοχία) τότε το εξάγει απευθείας για να αντικατασταθεί το ήδη

υπάρχον στην κρυφή μνήμη. Αλλιώς δίνεται προτεραιότητα στις εντολές αποθήκευσης. Αν είναι κάποια εντολή αποθήκευσης έτοιμη τότε προωθείται προς την κρυφή μνήμη έτσι ώστε να εξυπηρετηθεί. Αν δεν είναι έτοιμη κάποια εντολή αποθήκευσης, τότε εξετάζεται αν είναι έτοιμη κάποια εντολή φόρτωσης. Αν είναι, τότε δίνεται στην κρυφή μνήμη για εξυπηρέτηση. Σε κάθε άλλη περίπτωση, δεν εξάγεται τίποτα. Η ουρά μπορεί να αποθηκεύσει το πολύ δύο εντολές ταυτόχρονα. Εξετάζει αν υπάρχουν ελεύθερες θέσεις που μπορεί να γίνει αποθήκευση και τις τοποθετεί σε αυτές.



Εικόνα 3.21: Η λεπτομερής λειτουργία της κρυφής μνήμης δεδομένων L1

Η λειτουργία της κρυφής μνήμης δεδομένων L1 έχει ως εξής (βλέπε εικόνα 3.21): Συνεχώς καταφθάνουν δύο εντολές από το MOB και καθορίζεται ποιες θα είναι οι δύο είσοδοι της κρυφής μνήμης. Για την πρώτη είσοδο προτεραιότητα έχουν οι εντολές ή τα μπλοκς που εξάγονται από την ουρά αστοχιών (missqueue). Αν υπάρχει ένα μπλοκ ή μία εντολή από την ουρά αστοχιών που είναι έτοιμο/η να εξυπηρετηθεί, τότε επιλέγεται για την πρώτη είσοδό της κρυφής μνήμης και η εντολή που προέρχεται από το MOB αποθηκεύεται στην προσωρινή μνήμη (membuffer) για να εξυπηρετηθεί αργότερα. Αν δεν υπάρχει κάποια έτοιμη εντολή στη missqueue, τότε για την πρώτη είσοδο της κρυφής μνήμης επιλέγεται η εντολή από το MOB. Για τη δεύτερη είσοδο, προτεραιότητα έχουν οι εντολές που είναι στο membuffer και η άλλη εντολή που προέρχεται από το MOB αποθηκεύεται στο membuffer επίσης για να εξυπηρετηθεί αργότερα. Αν δεν υπάρχει έτοιμη εντολή στο membuffer, τότε για τη δεύτερη είσοδο της κρυφής μνήμης επιλέγεται απευθείας η εντολή από το MOB. Οι δύο είσοδοι στη συνέχεια υπόκεινται σε επεξεργασία έτσι ώστε να παραχθούν οι οκτώ είσοδοι των σειρών.

Τα δεδομένα που παράγονται από την επεξεργασία των δύο εισόδων, προωθούνται στις οκτώ εισόδους των σειρών στον επόμενο κύκλο ρολογιού (στην Εικόνα 3.21 μπορεί κάποιος να παρατηρήσει ότι οι έξοδοι του κυκλώματος routing logic αποθηκεύονται σε ένα σύνολο καταχωρητών - registers). Έπειτα γίνεται η προσκόμιση ή η αποθήκευση στην κρυφή μνήμη και με καθυστέρηση ενός ακόμη κύκλου εξάγονται τα αποτελέσματα (registered outputs). Αν έχει προκύψει αστοχία, τότε η εντολή αποθηκεύεται στη missqueue και ταυτόχρονα στέλνεται σήμα να προσκομιστεί το ζητούμενο μπλοκ από τη δευτερεύουσα μνήμη.

Στην εικόνα 3.21 φαίνεται επίσης το κύκλωμα **dtlb**. Το κύκλωμα αυτό μεταφράζει μία εικονική διεύθυνση στην αντίστοιχη φυσική της. Στον IVM όμως υπάρχει μόνο η σχετική διασύνδεση και όχι η υλοποίησή της, οπότε δεν αξίζει να αναφερθεί τίποτα περισσότερο.

Προσκόμιση μπορεί να γίνει οποιουδήποτε quadword (στην αρχιτεκτονική Alpha 1 word = 16 bits, οπότε 1 quadword = 64 bits) από μία γραμμή της κρυφής μνήμης. Φορτώνεται όλη η γραμμή και στη

συνέχεια επιλέγεται το κατάλληλο quadword ανάλογα με τα bits [4:3] της διεύθυνσης. Αποθήκευση μπορεί να γίνει σε οποιαδήποτε διεύθυνση byte μιας γραμμής της κρυφής μνήμης.

Δευτερεύουσα μνήμη δεν υπάρχει. Αυτό σημαίνει ότι σε περίπτωση αστοχίας δε γίνεται τίποτα και γι' αυτό θα πρέπει να μη δημιουργούνται αστοχίες κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Επίσης, λόγω απουσίας δευτερεύουσας ή άλλου είδους μνήμης, χαμηλότερου επιπέδου, θα πρέπει να φορτώνονται τα αρχικά δεδομένα (για παράδειγμα σφαιρικές μεταβλητές, δείκτης στοίβας, σφαιρικός δείκτης) στην κρυφή μνήμη στην αρχή της προσομοίωσης.

3.5.3. Περιγραφή της πλήρους λειτουργίας του σταδίου MEM

Έχοντας αναφέρει τις δύο βασικές οντότητες που συνθέτουν το στάδιο MEM, την προσωρινή μνήμη διάταξης εντολών μνήμης και την κρυφή μνήμη δεδομένων, στην ενότητα αυτή περιγράφεται η πλήρης λειτουργία του σταδίου.

Στο στάδιο MEM γίνονται οι ακόλουθες λειτουργίες:

- Αποστολή των εντολών στην κρυφή μνήμη δεδομένων.
- Έλεγχος για εξαρτήσεις (aliases) μεταξύ των εντολών φόρτωσης και αποθήκευσης.
- Ενημέρωση του ROB για το ποιες εντολές τελείωσαν την εκτέλεσή τους.
- Προώθηση στα υπόλοιπα τμήματα της διοχέτευσης τα αποτελέσματα των εντολών φόρτωσης που έχουν τελειώσει την εκτέλεσή τους.

Χειρισμός των νέων εντολών που εισέρχονται στο στάδιο MEM

Όταν εισέρχονται οι νέες εντολές στο στάδιο MEM, γίνεται αρχικά μία δρομολόγηση των διαφόρων πληροφοριών των εντολών προς ένα σύνολο καταχωρητών. Στη συνέχεια (στον ίδιο κύκλο) γίνεται διαχωρισμός των εντολών φόρτωσης από τις εντολές αποθήκευσης και δημιουργούνται οι κατάλληλες καταχωρήσεις για τις ουρές του MOB, LDQ και STQ. Οι νέες εντολές προέρχονται από την έξοδο του σταδίου **Rename**. Χρονικά, δηλαδή τη στιγμή που οι εντολές εξέρχονται του σταδίου Rename και εισέρχονται στο στάδιο MEM, δεν έχει ακόμη υπολογιστεί η διεύθυνση μνήμης διότι αυτή υπολογίζεται στο στάδιο EXE, από τις μονάδες παραγωγής διευθύνσεων (**AGUs** – **A**ddress **G**enaration **U**nits).

Προετοιμασία των θυρών της κρυφής μνήμης δεδομένων

Όπως αναφέρθηκε στην προηγούμενη ενότητα, η κρυφή μνήμη δεδομένων διαθέτει δύο θύρες. Οπότε από το MOB μεταδίδονται το πολύ δύο εντολές που προορίζονται για τις θύρες αυτές. Για την πρώτη θύρα προτεραιότητα έχουν οι εντολές φόρτωσης, των οποίων οι διευθύνσεις έχουν παραχθεί και έχουν μόλις έρθει από το στάδιο EXE και συγκεκριμένα από τη μονάδα **AGU0** (**A**ddress **G**eneration **U**nit 0). Αν δεν υπάρχει τέτοια εντολή φόρτωσης, της οποίας η διεύθυνση να έχει μόλις έρθει από το στάδιο EXE, τότε προτεραιότητα έχουν οι εντολές αποθήκευσης που έχουν ολοκληρωθεί έτσι ώστε να αποσυρθούν και να αλλάξουν την κατάσταση της μνήμης. Για τη δεύτερη θύρα, προτεραιότητα έχουν οι εντολές φόρτωσης των οποίων οι διευθύνσεις έχουν παραχθεί και έχουν μόλις έρθει από το στάδιο EXE και συγκεκριμένα από τη μονάδα **AGU1** (**A**ddress **G**eneration **U**nit 1). Αν δεν υπάρχει τέτοια εντολή φόρτωσης τότε σειρά έχουν οι εντολές φόρτωσης που βρίσκονται στο MOB (στην ουρά LDQ).

Εξαρτήσεις μεταξύ των εντολών φόρτωσης και αποθήκευσης

Με σκοπό την εξασφάλιση και τη διατήρηση της ακολουθιακής συνέπειας μνήμης (sequential memory consistency) λαμβάνονται υπόψη οι χρονικές εξαρτήσεις μεταξύ των εντολών φόρτωσης και αποθήκευσης. Δηλαδή δημιουργείται μία χρονική εξάρτηση μεταξύ μιας εντολής φόρτωσης με την αμέσως προηγούμενη εντολή αποθήκευσης. Αυτό σημαίνει ότι μία εντολή φόρτωσης δε μπορεί να τελειώσει την εκτέλεσή της αν δεν έχει ολοκληρωθεί η πιο πρόσφατη εντολή αποθήκευσης που προηγείται χρονικά. Παράλληλα με τις χρονικές εξαρτήσεις, γίνεται έλεγχος και για εξαρτήσεις RAW, όπου μία εντολή αποθήκευσης προσπελαύνει την ίδια θέση μνήμης με μία εντολή φόρτωσης, η οποία έπεται χρονικά.

Πιο συγκεκριμένα, όταν εισέρχονται τέσσερις εντολές από το στάδιο RENAME στο στάδιο MEM, καθορίζονται οι χρονικές εξαρτήσεις μεταξύ των εντολών φόρτωσης και αποθήκευσης. Κατά τη διάρκεια εξυπηρέτησης μιας εντολής φόρτωσης από την κρυφή μνήμη, ελέγχονται οι εξαρτήσεις RAW

με όλες τις εντολές αποθήκευσης που προηγούνται χρονικά. Στον έλεγχο αυτό διακρίνονται οι ακόλουθες περιπτώσεις:

- Αν οι διευθύνσεις προσπέλασης τετραπλής λέξης (quadword) είναι ίδιες (μεταξύ των εντολών φόρτωσης και αποθήκευσης), τότε:
  - Αν οι διευθύνσεις προσπέλασης byte είναι ίδιες και το μέγεθος των δεδομένων που προσπελαύνουν είναι ίδια, σημειώνεται να γίνει προώθηση του δεδομένου από την εντολή αποθήκευσης στην εντολή φόρτωσης.
  - Αλλιώς σημειώνεται ότι υπάρχει εξάρτηση RAW. Σε αυτή την περίπτωση η εντολή φόρτωσης περιμένει μέχρις ότου αποσυρθεί η εντολή αποθήκευσης και ύστερα τελειώνει την εκτέλεσή της.

Υπάρχει πιθανότητα, κατά τη διάρκεια ελέγχου των RAW εξαρτήσεων, μία εντολή φόρτωσης να έχει την ίδια διεύθυνση προσπέλασης τετραπλής λέξης με δύο τουλάχιστον εντολές αποθήκευσης. Για παράδειγμα, έστω οι εντολές:

```
.
.
.
store A+2
store A
load A
.
.
.
```

Οι εντολές εκτελούνται με τη σειρά, ξεκινώντας από πάνω προς τα κάτω. Επομένως, η εντολή φόρτωσης εξαρτάται χρονικά από την εντολή αποθήκευσης store A. Επίσης με τη συγκεκριμένη εντολή αποθήκευσης υπάρχει εξάρτηση RAW και μάλιστα σημειώνεται να γίνει προώθηση των δεδομένων της στην εντολή φόρτωσης. Όσον αφορά την εντολή αποθήκευσης store A+2, η εντολή φόρτωσης έχει την ίδια διεύθυνση τετραπλής λέξης. Παρόλο που μόνο η RAW εξάρτηση με την εντολή store A έχει νόημα, λόγω της συντηρητικής σχεδίασης του μηχανισμού, η εντολή φόρτωσης θα πρέπει να περιμένει την απόσυρση της εντολής αποθήκευσης store A+2 για να μπορέσει να τελειώσει την εκτέλεσή της.

Όταν ολοκληρωθούν κάποιες εντολές ελέγχονται από τη μονάδα μνήμης μήπως υπάρχει σε αυτές μία τουλάχιστον εντολή αποθήκευσης η οποία να έχει εξάρτηση RAW με μία εντολή φόρτωσης που έχει τελειώσει την εκτέλεσή της. Αυτό συμβαίνει έτσι ώστε να εξασφαλιστεί ότι έχει σημειωθεί η εξάρτηση μεταξύ αυτών των εντολών και η προώθηση των δεδομένων από την εντολή αποθήκευσης στην εντολή φόρτωσης. Αν δεν έχει σημειωθεί, τότε όταν έρθει η σειρά ολοκλήρωσης της εντολής φόρτωσης, θα πρέπει να δοθεί εντολή για εκκένωση της διοχέτευσης. Ακολουθεί ένα παράδειγμα που επεξηγεί καλύτερα την συγκεκριμένη περίπτωση.

```
.
.
.
store a,B
.
.
.
load r,B
.
.
.
```

Στον κώδικα αυτόν γίνεται αποθήκευση του δεδομένου του καταχωρητή a στη διεύθυνση B και φόρτωση δεδομένου από την ίδια διεύθυνση. Όταν η εντολή φόρτωσης εισέλθει για πρώτη φορά στο στάδιο MEM ανιχνεύεται η χρονική εξάρτηση με την εντολή αποθήκευσης. Ας υποτεθεί ότι λόγω εξαρτήσεων της εντολής αποθήκευσης με άλλες εντολές, καθυστερεί η διεκπεραίωση/εκκίνησή της. Έτσι διεκπεραιώνεται/ξεκινάει πρώτα η εντολή φόρτωσης. Επειδή δεν έχει παραχθεί η διεύθυνση προσπέλασης για την εντολή αποθήκευσης δεν ανιχνεύεται η εξάρτηση RAW μεταξύ της εντολής

φόρτωσης και της εντολής αποθήκευσης. Οπότε μόλις παραχθεί η διεύθυνση προσπέλασης της εντολής φόρτωσης, στέλνεται για εξυπηρέτηση στην κρυφή μνήμη.

Έστω ότι σε δύο κύκλους, που τελειώνει η εξυπηρέτηση της εντολής φόρτωσης από την κρυφή μνήμη, δεν έχει ακόμα διεκπεραιώνεται/ξεκινάει η εντολή αποθήκευσης. Αυτό έχει σαν συνέπεια να μη σταλεί η εντολή φόρτωσης προς ολοκλήρωση στο στάδιο Retire του IVM και να περιμένει την ολοκλήρωση της εντολής αποθήκευσης.

Μόλις επιλυθούν όλες οι εξαρτήσεις της εντολής αποθήκευσης διεκπεραιώνεται/ξεκινάει για να παραχθεί η διεύθυνση προσπέλασής της. Μετά από τρεις κύκλους παράγεται η διεύθυνση προσπέλασης, ενώ στον επόμενο κύκλο στέλνεται η εντολή στο στάδιο Retire για να ολοκληρωθεί. Όταν ολοκληρωθεί γίνεται έλεγχος στο στάδιο MEM αν υπάρχει εξάρτηση RAW με κάποια εντολή φόρτωσης που έχει τελειώσει την εκτέλεσή της. Συνεπώς εντοπίζεται η εξάρτηση μεταξύ των δύο εντολών του κώδικα. Όμως επειδή η εντολή φόρτωσης έχει τελειώσει νωρίτερα την εκτέλεσή της, έχουν προωθηθεί τα λανθασμένα δεδομένα στα υπόλοιπα στάδια της διοχέτευσης. Οπότε, όταν έρθει η σειρά ολοκλήρωσης της εντολής φόρτωσης δε θα ολοκληρωθεί και θα δοθεί εντολή για εκκένωση της διοχέτευσης. Μετά την εκκένωση, η προσκόμιση των εντολών θα αρχίσει από τη διεύθυνση της εντολής φόρτωσης.

Ενημέρωση της προσωρινής μνήμης αναδιάταξης (ROB)

Ο μέγιστος αριθμός εντολών που μπορούν να σταλούν ταυτόχρονα στο ROB είναι τέσσερις. Δύο εντολές φόρτωσης και δύο εντολές αποθήκευσης. Μία εντολή φόρτωσης στέλνεται στο ROB όταν:

- Είναι έγκυρη
- Έχει τελειώσει την εκτέλεσή της (μία εντολή φόρτωσης τελειώνει την εκτέλεσή της όταν προσκομιστούν τα δεδομένα από την κρυφή μνήμη και δεν εξαρτάται χρονικά από κάποια εντολή αποθήκευσης)
- Η εντολή αποθήκευσης από την οποία εξαρτιόταν χρονικά έχει ολοκληρωθεί
- Δε δημιουργεί κάποια αστοχία στο dtlb

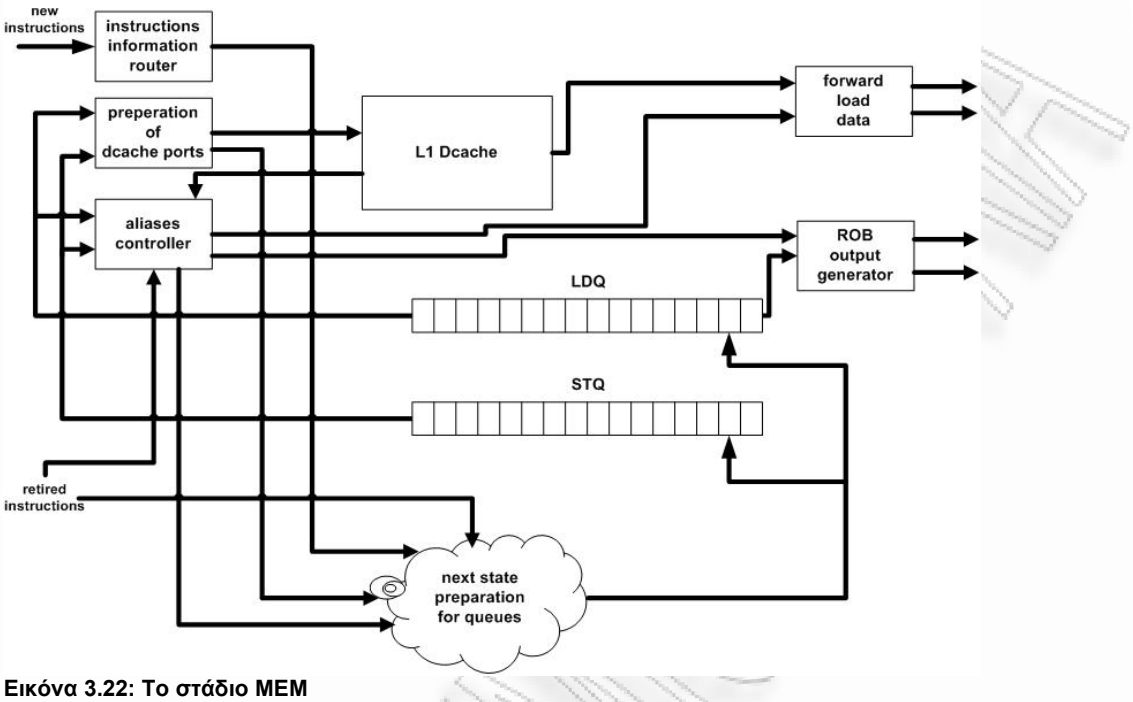
Υπάρχουν και κάποιες άλλες συνθήκες, δευτερεύουσας σημασίας, από τις οποίες εξαρτάται για το αν θα σταλεί ή όχι κάποια εντολή φόρτωσης στο ROB αλλά βασίζονται στη σχεδίαση του επεξεργαστή και δεν αξίζει να αναφερθούν.

Μία εντολή αποθήκευσης, μόλις ετοιμαστεί η διεύθυνση προσπέλασής της από το στάδιο EXE, στέλνεται κατευθείαν στο ROB για να ολοκληρωθεί. Μόλις εξέλθει από το ROB περιμένει σε μία ουρά για να αποσυρθεί και να αλλάξει σωστά την κατάσταση της μνήμης.

Προώθηση των δεδομένων που έχουν προσκομιστεί από την κρυφή μνήμη

Το στάδιο MEM προωθεί στα υπόλοιπα στάδια τα αποτελέσματα των εντολών φόρτωσης (ο αριθμός των αποτελεσμάτων είναι το πολύ δύο αφού η κρυφή μνήμη περιέχει 2 θύρες). Τα δεδομένα που προωθούνται είναι με εικασία και η εγκυρότητά τους καθορίζεται από το αν εξαρτάται η εντολή φόρτωσης χρονικά από μία εντολή αποθήκευσης. Αν υπάρχει τέτοια εξάρτηση τότε τα δεδομένα ακυρώνονται. Αν δεν υπάρχει καμία εξάρτηση τότε τα δεδομένα στέλνονται κανονικά. Χωρίς πρόβλημα στέλνονται επίσης και τα δεδομένα μιας εντολής φόρτωσης που έχει εξάρτηση RAW με μία εντολή αποθήκευσης. Διότι σε αυτή την περίπτωση προωθούνται απευθείας τα δεδομένα της εντολής αποθήκευσης.

Κλείνοντας τη συγκεκριμένη ενότητα, ακολουθεί μία αφαιρετική αναπαράσταση του σταδίου MEM.



Εικόνα 3.22: Το στάδιο MEM

Συνοπτικά οι οντότητες που απεικονίζονται είναι οι:

- **instructions information router**: Λογική η οποία συλλέγει και διαχωρίζει από τις νέες εντολές μνήμης τις διάφορες πληροφορίες.
- **preparation of dcache ports**: Το συγκεκριμένο τμήμα κυκλώματος επιλέγει και προετοιμάζει τα δεδομένα για τις εισόδους της κρυφής μνήμης δεδομένων.
- **aliases controller**: Το σύνολο των κυκλωμάτων που ελέγχουν τις εξαρτήσεις μεταξύ των εντολών μνήμης, χρονικές και RAW.
- **L1 Dcache**: Η κρυφή μνήμη δεδομένων.
- **next state preparation for queues**: Συνδυαστική λογική που προετοιμάζει την επόμενη κατάσταση των δεδομένων των ουρών LDQ και STQ.
- **LDQ, STQ**: Οι ουρές LDQ και STQ (δες παράγραφο 3.5.1).
- **forward load data**: Το κύκλωμα που προωθεί τα δεδομένα των εντολών φόρτωσης που έχουν προσκομιστεί από την κρυφή μνήμη δεδομένων.
- **ROB output generator**: Το κύκλωμα που παράγει τις καταχωρήσεις που αφορούν τις εντολές μνήμης για την προσωρινή μνήμη αναδιάταξης.

3.6. Στάδιο Εκτέλεσης (EX)

Στο στάδιο αυτό γίνεται η εκτέλεση όλων των εντολών, επιλέγονται οι κατάλληλοι τελεστέοι και εκτελούνται οι σχετικές λειτουργίες που απαιτούν οι διάφορες εντολές. Οι τελεστέοι μπορεί να προέρχονται είτε από το αρχείο καταχωρητών είτε από αποτελέσματα άλλων εντολών τα οποία έχουν προωθηθεί.

Το στάδιο περιλαμβάνει 6 βασικές λειτουργικές μονάδες οι οποίες λειτουργούν ταυτόχρονα και είναι: Δύο αριθμητικές λογικές μονάδες που εκτελούν απλές αριθμητικές εντολές (simple ALUs), μία αριθμητική λογική μονάδα που εκτελεί εντολές που απαιτούν πολύπλοκους υπολογισμούς (complex ALU), μία μονάδα επίλυσης διακλαδώσεων και δύο μονάδες παραγωγής διευθύνσεων (AGU0 και AGU1), οι οποίες παράγουν τις διευθύνσεις προσπέλασης των εντολών μνήμης.

Μαζί με αυτές τις έξι λειτουργικές μονάδες, περιλαμβάνεται και ένας πολλαπλασιαστής ακέραιων αριθμών, του οποίου η διάρκεια εκτέλεσης είναι 5 κύκλοι ρολογιού. Δυστυχώς η σχεδίαση του

συγκεκριμένου πολλαπλασιαστή δεν είναι ιδιαίτερα αποδοτική διότι δεν εφαρμόζει την τεχνική συνεχούς διοχέτευσης στο ίδιο το κύκλωμα, το οποίο είναι εξαιρετικά μεγάλο και πολύπλοκο, αλλά έχει τέσσερις καταχωρητές από τους οποίους διέρχονται οι τελεστέοι που χρησιμοποιούνται για την πράξη. Αυτό γίνεται για να προσομοιωθεί με ακρίβεια ο χρόνος εκτέλεσης.

Στην περίπτωση της complex ALU μία εντολή εκτελείται σε 2 κύκλους. Συνεπώς, λόγω αυτής της διαφοράς στο χρόνο εκτέλεσης μεταξύ του πολλαπλασιαστή και της complex ALU υπάρχει μεγάλη πιθανότητα σε κάποιον κύκλο ρολογιού να είναι έτοιμα τα αποτελέσματα και από τις δύο λειτουργικές μονάδες. Το πρόβλημα όμως είναι ότι μπορεί να εξαχθεί μόνο ένα εκ των δύο αποτελεσμάτων από το στάδιο EXE. Για αυτές τις περιπτώσεις εφαρμόζεται ο ακόλουθος κανόνας διαιτησίας: Αν και οι δύο λειτουργικές μονάδες έχουν παραγάγει τα αποτελέσματά τους στον ίδιο κύκλο ρολογιού, τότε προτεραιότητα έχει αυτό του πολλαπλασιαστή και το αποτέλεσμα της complex ALU εισάγεται σε μία προσωρινή μνήμη. Αν έχει παραγάγει μόνο ο πολλαπλασιαστής αποτέλεσμα, τότε επιλέγεται να εξαχθεί από το στάδιο EXE. Αν έχει παραγάγει μόνο η complex ALU αποτέλεσμα, τότε επιλέγεται να εξαχθεί από το στάδιο EXE. Αν δεν έχει παραχθεί κανέναν αποτέλεσμα, από καμία από τις δύο λειτουργικές μονάδες, τότε σαν έξοδος επιλέγεται η πιο παλιά καταχώρηση της προσωρινής μνήμης.

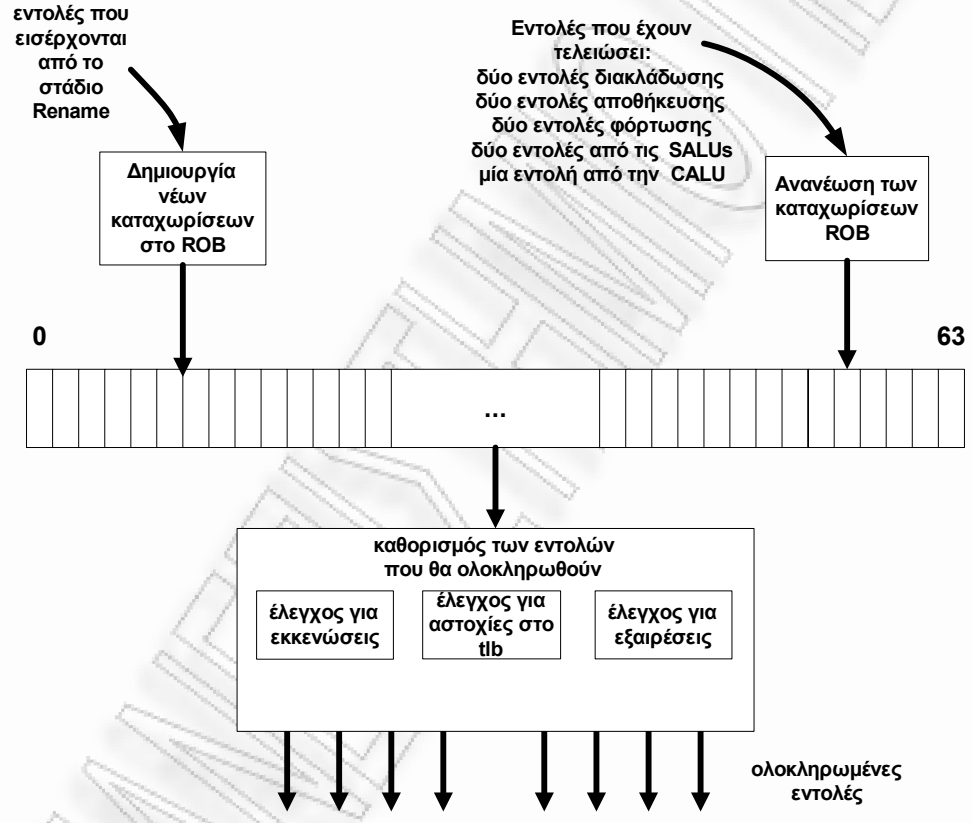
Εξαιρουμένων των πολλαπλασιαστή και complex ALU, οι υπόλοιπες λειτουργικές μονάδες έχουν έτοιμα τα αποτελέσματά τους στον επόμενο κύκλο ρολογιού.

3.7. Στάδιο Απόσυρσης (Retire)

Στο στάδιο αυτό οι εντολές ολοκληρώνουν την εκτέλεσή τους (complete) και αλλάζουν μόνιμα την κατάσταση του επεξεργαστή. Βέβαια στην περίπτωση των εντολών αποθήκευσης χρειάζεται ένα ακόμη βήμα έτσι ώστε να αλλάξουν σωστά και μόνιμα την κατάσταση της μνήμης. Για την αποφυγή οποιονδήποτε παρεξηγήσεων, το όνομα που έχει δοθεί για το συγκεκριμένο στάδιο έχει την ίδια έννοια με αυτήν που έχει η χρήση του όρου complete. Το στάδιο Retire περιλαμβάνει δύο υποστάδια, το ROB και το ArchRATfile.

3.7.1. Υποστάδιο ROB

Αυτό το υποστάδιο περιλαμβάνει μία προσωρινή μνήμη αναδιάταξης (ROB) των εντολών, 64 καταχωρίσεων. Η προσωρινή μνήμη αναδιάταξης χρησιμοποιείται για να αποθηκεύει πληροφορίες για τις εντολές που είναι ακόμα στον επεξεργαστή και περιμένουν να εκτελεστούν, καθώς επίσης και για τη ολοκλήρωση σε σειρά των εντολών έτσι ώστε να διατηρείται η συνέπεια της εκτέλεσης και να αλλάζει σωστά η κατάσταση της μηχανής.



Εικόνα 3.23: Το υποστάδιο ROB

Στο υποστάδιο ROB εισέρχονται οι πληροφορίες τεσσάρων νέων εντολών από το στάδιο Rename και δημιουργούνται τέσσερις καινούριες καταχωρήσεις στην προσωρινή μνήμη αναδιάταξης. Εκτός όμως από τη δημιουργία καινούριων καταχωρήσεων γίνεται και αναβάθμιση των υπαρχόντων. Η αναβάθμιση προκύπτει από εντολές που έχουν τελειώσει την εκτέλεσή τους και μπορεί να είναι:

- Μία εντολή διακλάδωσης που προέρχεται από τη μονάδα επίλυσης διακλαδώσεων του σταδίου EXE.
- Δύο εντολές αποθήκευσης από το στάδιο MEM.
- Δύο εντολές φόρτωσης από το στάδιο MEM.

- Δύο εντολές που προέρχονται από τις αριθμητικές λογικές μονάδες του σταδίου EXE, οι οποίες εκτελούν απλές αριθμητικές εντολές.
- Μία εντολή που μπορεί να έχει εκτελεστεί είτε από τη complex ALU, είτε από τον πολλαπλασιαστή.

Όλες οι εντολές ολοκληρώνονται σε σειρά και σε κάθε κύκλο ρολογιού μπορούν να ολοκληρώνονται το πολύ οκτώ. Για να μπορέσει μία εντολή να ολοκληρωθεί επιτυχώς και άρα να υπάρχει πιθανότητα να ολοκληρωθούν και οι επόμενες, χρονικά, εντολές θα πρέπει να γίνουν οι ακόλουθοι έλεγχοι:

- Η εντολή να είναι έγκυρη.
- Η επόμενη από αυτήν εντολή να είναι έγκυρη.
- Η εντολή να έχει τελειώσει την εκτός σειράς ή με εικασία εκτέλεσή της.
- Έλεγχος αν είναι εντολή cmov (αν είναι, έλεγχος αν έχει τελειώσει την εκτέλεσή της η επόμενη από αυτήν εντολή).
- Να μην είναι εντολή διακλάδωσης ή αντιθέτως να είναι η πρώτη εντολή διακλάδωσης μέσα στην υποψήφια οκτάδα εντολών.
- Έλεγχος αν δε μπορεί να ολοκληρωθεί λόγω κάποιας εντολής διακλάδωσης μέσα στην υποψήφια οκτάδα εντολών, η οποία προηγείται χρονικά.
- Έλεγχος αν δε μπορεί να ολοκληρωθεί λόγω ύπαρξης κάποιας εντολής φόρτωσης που εξαρτάται από κάποια εντολή αποθήκευσης.
- Να μη δημιουργεί ή να μην επηρεάζεται από κάποια εξαίρεση.
- Να μη δημιουργεί ή να μην επηρεάζεται από κάποια αστοχία στο TLB.

Χειρισμός των διακλαδώσεων

Σε μία οκτάδα εντολών που είναι υποψήφια να ολοκληρωθεί, ελέγχεται, για κάθε εντολή i, αν είναι αυτή η πρώτη εντολή διακλάδωσης. Αν δεν είναι, τότε δε μπορεί να ολοκληρωθεί. Αν είναι, τότε μόνο αυτή και όσες εντολές προηγούνται χρονικά μπορούν να ολοκληρωθούν, οι άλλες όχι. Για μία διακλάδωση δίνεται εντολή για εκκένωση της διοχέτευσης όταν:

target of inst (i) ≠ **address** of inst (i+1)

Χειρισμός των εντολών φόρτωσης που έχουν εξάρτηση RAW με κάποια εντολή αποθήκευσης

Αν κάποια εντολή φόρτωσης μέσα στην οκτάδα έχει εξάρτηση RAW με κάποια εντολή αποθήκευσης, τότε επιτρέπεται να ολοκληρωθούν όλες οι εντολές που προηγούνται χρονικά της συγκεκριμένης εντολής φόρτωσης και δίνεται εντολή για εκκένωση της διοχέτευσης.

Χειρισμός των εξαιρέσεων και των αστοχιών στο TLB

Σε μία υποψήφια για ολοκλήρωση, οκτάδα εντολών, για να μπορέσει μία εντολή να ολοκληρωθεί δεν πρέπει να δημιουργεί κάποιου είδους εξαίρεση, αλλά ούτε και κάποια αστοχία στο TLB. Αν κάποια εντολή δημιουργεί εξαίρεση ή αστοχεί στο TLB δε μπορεί να ολοκληρωθεί αλλά ούτε και οι επόμενες από αυτή εντολές, παρά μόνο οι προηγούμενες. Οι συνέπειες είναι μόνο σε θεωρητικό επίπεδο. Όταν βρεθεί εντολή που προκαλεί εξαίρεση ή αστοχία απλά σταματάει η προσομοίωση. Επίσης, θα πρέπει να επισημανθεί ότι αστοχία στο TLB δεν πρόκειται να δημιουργηθεί ποτέ.

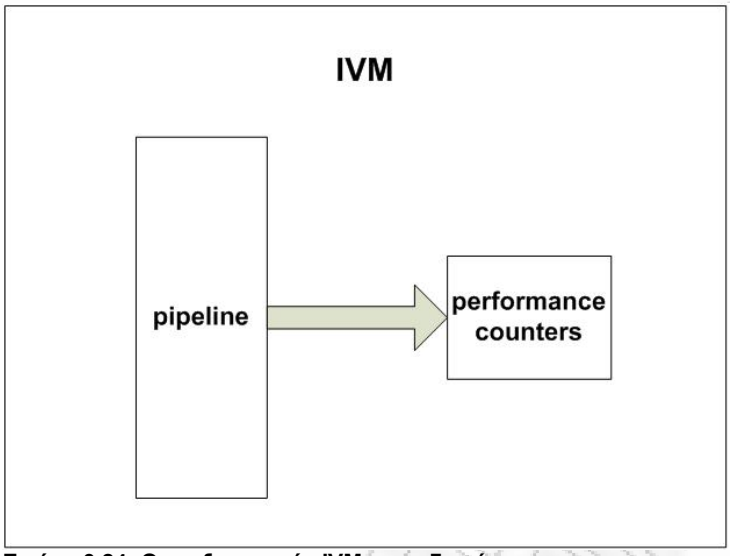
3.7.2. Υποστάδιο ArchRATfile

Όταν οι εντολές τελειώσουν την εκτέλεσή τους, τότε τα αποτελέσματά τους αποθηκεύονται στους φυσικούς καταχωρητές του επεξεργαστή. Επομένως, όταν οι εντολές ολοκληρωθούν θα πρέπει να γίνει η μόνιμη αντιστοίχηση των αρχιτεκτονικών καταχωρητών με τους αντίστοιχους φυσικούς καταχωρητές.

Σε αυτό το υποστάδιο γίνεται η συγκεκριμένη εργασία. Το ArchRATfile περιλαμβάνει τον έγκυρο αρχιτεκτονικό πίνακα καταχωρητών όπου υπάρχουν οι συνδέσεις των αρχιτεκτονικών καταχωρητών με τους φυσικούς. Ο πίνακας τροποποιείται εφόσον ολοκληρωθεί μια εντολή, διότι στην περίπτωση αυτή μπορεί να αλλάξει την κατάσταση της μηχανής. Ο αντίστοιχος πίνακας υπάρχει και στο υποστάδιο Rename0, με τη μόνη διαφορά ότι οι συνδέσεις εκεί είναι με εικασία. Όταν δοθεί εντολή για εκκένωση, ο αρχιτεκτονικός πίνακας καταχωρητών αντιγράφεται στον αρχιτεκτονικό πίνακα με εικασία.

3.8. Καταχωρητές μέτρησης απόδοσης (Performance Counters)

Ο επεξεργαστής IVM διαθέτει κάποιους καταχωρητές στατιστικών στοιχείων που χρησιμεύουν στη μέτρηση διαφόρων ποσοτικών δεδομένων. Το περιεχόμενο των συγκεκριμένων καταχωρητών ανανεώνεται με βάση τα σήματα που εξάγει η διοχέτευση. Για παράδειγμα, ένας καταχωρητής που έχει αποθηκευμένο των αριθμό των εκκενώσεων ενημερώνεται από το σήμα εκκένωσης που εξάγει η διοχέτευση. Αν το σήμα είναι στο λογικό 1, αυτό σημαίνει ότι έχει δοθεί εντολή για εκκένωση και ο καταχωρητής αυξάνει το περιεχόμενό του κατά 1, αλλιώς δε γίνεται καμία αλλαγή. Η επόμενη λίστα τους παραθέτει με τα ονόματά τους αριστερά και δεξιά μία σύντομη επεξήγηση για το περιεχόμενό τους.



Εικόνα 3.24: Ο επεξεργαστής IVM με τη διοχέτευση και τους καταχωρητές μέτρησης της απόδοσης

- **retired_inst_count**: Οι εντολές που έχουν ολοκληρωθεί.
- **cpu_cycles**: Οι κύκλοι ρολογιού που λειτουργεί ο επεξεργαστής.
- **branch_mispred**: Τα direct branches που δεν προβλέφθηκαν σωστά.
- **total_branches**: Ο συνολικός αριθμός των άμεσων διακλαδώσεων.
- **total_indirects**: Ο συνολικός αριθμός των έμμεσων διακλαδώσεων.
- **flushes:** Ο συνολικός αριθμός των εκκενώσεων.
- **indirect_mispred**: Ο αριθμός των έμμεσων διακλαδώσεων που δεν προβλέφθηκαν σωστά.
- **aliasflushes**: Ο αριθμός των εκκενώσεων που προκλήθηκαν λόγω των εξαρτήσεων μνήμης.
- **poppushmispreds**: Ο αριθμός των λανθασμένων ωθήσεων που έγιναν στη στοίβα RAS μετά από λανθασμένες προβλέψεις επιστροφής από ρουτίνα.
- **l1dcache_accesses**: Ο αριθμός των έγκυρων προσπελάσεων στην κρυφή μνήμη δεδομένων.
- **l1dcache_hits**: Ο αριθμός των ευστοχιών στην κρυφή μνήμη δεδομένων.
- **l1dcache_tlbmisses**: Ο αριθμός των αστοχιών στο TLB της μνήμης δεδομένων.

4. Προσομοίωση εκτέλεσης C προγραμμάτων στον επεξεργαστή IVM

Για να πιστοποιηθεί η σωστή λειτουργία του επεξεργαστή IVM προσομοιώθηκε η εκτέλεση προγραμμάτων, υλοποιημένα στη γλώσσα προγραμματισμού C, στον επεξεργαστή. Για το σκοπό αυτό χρησιμοποιήθηκε η εφαρμογή Icarus και συγκεκριμένα η έκδοση 0.9.1. Η εφαρμογή αυτή υποστηρίζεται από το λειτουργικό σύστημα Linux και αποτελεί έναν προσομοιωτή λειτουργίας κυκλωμάτων σχεδιασμένα στη γλώσσα περιγραφής υλικού Verilog.

Ο επεξεργαστής IVM, υποστηρίζει την αρχιτεκτονική συνόλου εντολών Alpha ev6. Η συντριπτική πλειοψηφία όμως των υπολογιστών διαθέτουν επεξεργαστή αρχιτεκτονικής x86. Συνεπώς, στο λειτουργικό σύστημα Linux που είναι συνήθως προεγκατεστημένος ο μεταγλωττιστής GCC δε μπορούν τα προγράμματα να μεταγλωττιστούν σε γλώσσα μηχανής Alpha. Αυτό είχε σαν αποτέλεσμα την εγκατάσταση ενός ετερομεταγλωττιστής, ο οποίος μεταγλωττίζει προγράμματα, υλοποιημένα στη γλώσσα προγραμματισμού C, σε γλώσσα μηχανής Alpha. Η εγκατάστασή του έγινε με τη βοήθεια της εφαρμογής crosstool-ng 1.5.0.

Επιπρόσθετα, έπρεπε να ληφθούν υπόψη διάφοροι περιορισμοί όπως η αρχικοποίηση του καταχωρητή στοίβας. Ένα πρόγραμμα που εκτελείται χρησιμοποιεί το μοντέλο στοίβας για την αποθήκευση των διαφόρων δεδομένων του και για τις κλήσεις διαδικασιών. Άρα έπρεπε να γίνει η αρχικοποίηση του στη σωστή τιμή. Στα περισσότερα προγραμματιστικά πρότυπα (θα πρέπει να σημειωθεί πως δεν ακολουθήθηκαν πιστά, διότι σκοπός της διατριβής είναι η μελέτη του IVM και όχι η εκτέλεση προγραμμάτων σε αυτόν), η στοίβα επεκτείνεται από τις υψηλές διευθύνσεις στις χαμηλές. Συνεπώς ο δείκτης στοίβας πρέπει να έχει αποθηκευμένη την υψηλότερη διεύθυνση μνήμης. Ο IVM όταν ξεκινά τη λειτουργία του έχει τον καταχωρητή $30 στο 0. Επομένως έπρεπε με κάποιο τρόπο να δίνεται η σωστή τιμή στον καταχωρητή $30, προτού ξεκινήσει η εκτέλεση του προγράμματος. Η λύση που προτάθηκε είναι να εισάγεται η εντολή που αρχικοποιεί το δείκτη στοίβας πριν την εκτέλεση της πρώτης εντολής, της ρουτίνας main. Για να επιτευχθεί αυτό πρέπει να τροποποιείται το αρχείο σε συμβολική γλώσσα. Επομένως η διαδικασία μεταγλώττισης πρέπει να σταματάει πριν την παραγωγή του αντικειμενικού αρχείου. Η εντολή φλοιού που χρησιμοποιείται για την παραγωγή του αρχείου σε συμβολική γλώσσα είναι:

```
alphaev56-unknown-linux-gnu-gcc -O0 -mcpu=ev6 -S <όνομα_αρχείου>.c
                                <όνομα_αρχείου>.s
```

Εφόσον γίνει η προσθήκη της εντολής αρχικοποίησης του δείκτη στοίβας στο αρχείο συμβολικής γλώσσας (*.s), τότε πρέπει να συνεχιστεί η διαδικασία μεταγλώττισης με την ακόλουθη εντολή:

```
alphaev56-unknown-linux-gnu-as -o <όνομα_αρχείου>.o <όνομα_αρχείου>.s
```

Η εντολή αυτή δέχεται σαν είσοδο το αρχείο συμβολικής γλώσσας και παράγει το αντίστοιχο αντικειμενικό αρχείο (*.o).

Έπειτα πρέπει να χρησιμοποιηθεί ο πρόγραμμα σύνδεσης για να παραχθεί το τελικό, εκτελέσιμο πρόγραμμα. Δηλαδή το αντικειμενικό αρχείο να συνδεθεί με τις βιβλιοθήκες και να γίνει η σωστή αντιστοίχηση των τμημάτων του στις διευθύνσεις μνήμης. Ως εκ τούτου, έπρεπε να ληφθεί υπόψη ότι δεν υπάρχει κάποιο είδος δευτερεύουσας ή κύριας μνήμης, παρά μόνο οι κρυφές μνήμες δεδομένων και εντολών. Έτσι, δόθηκε προσοχή στα όρια των κρυφών μνημών δεδομένων και εντολών και στη μορφή που έπρεπε να είναι το δυαδικό αρχείο ώστε να φορτωθεί σωστά. Η κρυφή μνήμη εντολών έχει μέγεθος 8KB και αποτελείται από 2 σειρές, οι οποίες είναι πλεκτές. Συνεπώς διευθυνσιοδοτείται από το 0 έως το 8191. Η κρυφή μνήμη δεδομένων έχει μέγεθος 32 KB και αποτελείται από 8 σειρές, οι οποίες είναι πλεκτές. Συνεπώς διευθυνσιοδοτείται από το 0 έως το 32767. Για να μπορέσει να συνδεθεί σωστά το αντικειμενικό αρχείο, δημιουργήθηκε ένα linker script (δες παράρτημα Α) το οποίο περιγράφει το μοντέλο μνήμης που χρησιμοποιείται για την αποθήκευση των προγραμμάτων. Ορίστηκαν λοιπόν δύο περιοχές, η RAM και η ROM (απλά ονόματα, καμία σχέση με τις γνωστές RAM και ROM), στις οποίες αποθηκεύονται τα δεδομένα και οι εντολές αντίστοιχα. Η ROM διευθυνσιοδοτείται από το 0 έως το 3071. Ο λόγος που δε χρησιμοποιήθηκε όλος ο δυνατός χώρος διευθύνσεων της κρυφής μνήμης δεδομένων

είναι επειδή στα πειράματα που έγιναν δε δοκιμάστηκαν προγράμματα μεγέθους πάνω από 3KB. Το εύρος διευθύνσεων της RAM είναι από 3072 έως 32767. Συνεπώς ο δείκτης στοίβας πρέπει να αρχικοποιηθεί με την τιμή 32768. Όμως, στην περίπτωση διευθυνσιοδότησης της RAM υπάρχει μια μικρή διαφοροποίηση με τη διευθυνσιοδότηση της κρυφής μνήμης δεδομένων διότι δε συμπεριλαμβάνονται οι διευθύνσεις 0 – 3071 της τελευταίας. Ο λόγος είναι επειδή στο linker script δε μπορούν να οριστούν δύο περιοχές που να ξεκινούν από την ίδια διεύθυνση και να επικαλύπτονται. Δυστυχώς αυτό είναι και το μειονέκτημα της συγκεκριμένης μεθόδου εκτέλεσης προγραμμάτων αφού οι διευθύνσεις αυτές παραμένουν αχρησιμοποίητες. Στο σημείο αυτό θα πρέπει να τονισθεί επίσης ότι το συγκεκριμένο μοντέλο μνήμης που ορίστηκε μέσω του linker script δεν είναι υποχρεωτικό να έχει την παραπάνω μορφή. Ο καθένας μπορεί να αλλάξει τα όρια διευθυνσιοδότησης αρκεί να μη συμβούν επικαλύψεις ή να μην ξεπεραστούν τα μεγέθη των κρυφών μνημών εντολών και δεδομένων.

Η εντολή φλοιού που χρησιμοποιήθηκε για τη σύνδεση του αντικειμενικού αρχείου είναι:

```
alphaev56-unknown-linux-gnu-ld -T<όνομα_linker_script> -emain -nostartfiles -
nodefaultlibs -o <όνομα_εκτελέσιμου_προγράμματος>
<όνομα_αντικειμενικού_αρχείου>
```

Επειδή όμως το τελικό εκτελέσιμο αρχείο που παράγεται περιέχει άχρηστη πληροφορία για τον επεξεργαστή IVM, έπρεπε να χρησιμοποιηθεί η επόμενη εντολή έτσι ώστε να αποκοπούν οι τομείς που δεν περιέχουν εκτελέσιμο κώδικα ή δεδομένα για τον επεξεργαστή.

```
alphaev56-unknown-linux-gnu-strip --remove-section=.comment --remove-
section=.eh_frame --remove-section=.eh_frame_hdr
<όνομα_εκτελέσιμου_προγράμματος>
```

Αφού γίνει και η αφαίρεση των ανούσιων, για τον επεξεργαστή, πληροφοριών μπορεί ο χρήστης να αποκωδικοποιήσει το εκτελέσιμο πρόγραμμα με την εντολή:

```
alphaev56-unknown-linux-gnu-objdump -D
<όνομα_εκτελέσιμου_προγράμματος> <όνομα_αρχείου_log>
```

Με το εργαλείο objdump, το οποίο παρέχεται από το crosscompiler, ο χρήστης μπορεί να δει το εσωτερικό του εκτελέσιμου προγράμματος καθώς επίσης και σε ποιες εικονικές διευθύνσεις έχουν αντιστοιχηθεί οι διάφοροι τομείς του.

Όμως το εκτελέσιμο πρόγραμμα, στη μορφή που είναι, δε μπορεί να εκτελεστεί ακόμα από τον επεξεργαστή IVM. Ο μεταγλωττιστής, για να το παράγει, ακολουθεί συγκεκριμένους κανόνες και πρότυπα. Με βάση τα πρότυπα αυτά προστίθενται σαν κεφαλίδες διάφορες πληροφορίες και σύμβολα που δεν αναγνωρίζονται από τον επεξεργαστή. Με τη χρήση του εργαλείου objcopy δημιουργείται ένα νέο εκτελέσιμο, δυαδικό αρχείο το οποίο αποτελεί αντιγραφή του εκτελέσιμου προγράμματος χωρίς όμως τις διάφορες κεφαλίδες. Η εντολή είναι:

```
alphaev56-unknown-linux-gnu-objcopy -O binary
<όνομα_εκτελέσιμου_προγράμματος> <όνομα_δυαδικού_αρχείου>
```

Τέλος, αφού έχει παραχθεί το δυαδικό αρχείο δε μένει παρά να φορτωθεί στις κρυφές μνήμες εντολών και δεδομένων. Όπως αναφέρθηκε σε προηγούμενες ενότητες οι κρυφές μνήμες αποτελούνται από σειρές, οι οποίες είναι πλεκτές. Συνεπώς, χρειάζεται η συμβολή ενός ειδικού script, υλοποιημένο στη γλώσσα perl (δες παράρτημα Α), το οποίο δημιουργεί αρχεία κατάλληλα για τις σειρές των κρυφών μνημών εντολών και δεδομένων. Σαν είσοδο δέχεται το εκτελέσιμο, δυαδικό αρχείο το οποίο πρέπει να είναι στη μορφή που είναι και το δυαδικό αρχείο που παρατίθεται στην επόμενη σελίδα.

```
201f0001
203f0002
205f0003
207f0004
209f0005
20bf0006
20df0006
203f0007
201f0008
203f0009
```

```
201f0001
203f0002
205f0003
207f0004
209f0005
20bf0006
20df0006
203f0007
201f0008
203f0009
201f0001
203f0002
205f0003
207f0004
209f0005
20bf0006
20df0006
203f0007
201f0008
203f0009
201f0001
203f0002
205f0003
207f0004
209f0005
20bf0006
20df0006
203f0007
201f0008
203f0009
```

Όταν παράγεται το δυαδικό αρχείο από το εργαλείο objcopy, ο τρόπος αποθήκευσης των δεδομένων είναι μικρού άκρου (little endian) και τα δεδομένα είναι όλα σε σειρά. Όπως φαίνεται όμως και από το παραπάνω παράδειγμα δυαδικού, εκτελέσιμου αρχείου η κάθε εντολή, των 32 bits, πρέπει είναι στη δική της σειρά και ο τρόπος αποθήκευσής της να είναι μεγάλου άκρου. Το ίδιο ισχύει και για τα δεδομένα που μπορεί να περιέχει.

Συνεπώς, για να μπορέσει να μετατραπεί το δυαδικό αρχείο, που προκύπτει από το εργαλείο objcopy, στη μορφή που απαιτείται από το perl script, χρησιμοποιήθηκε ένα άλλο script υλοποιημένο στη γλώσσα tcl (δες παράρτημα Α). Επομένως για να εκτελεστεί το tcl script δίνεται η εντολή:

tcl <όνομα_script>.tcl <όνομα_δυαδικού_αρχείου> <όνομα_νέου_αρχείου>

Το νέο δυαδικό αρχείο που προκύπτει, δίνεται σαν είσοδος στο perl script το οποίο εκτελείται με την εντολή:

./<όνομα_script>.pl <όνομα_δυαδικού_αρχείου>

Από την εκτέλεση παράγονται 20 αρχεία που περιέχουν τα δεδομένα των σειρών των κρυφών μνημών (16 για την κρυφή μνήμη δεδομένων και 4 για την κρυφή μνήμη εντολών αλλά και για τους δύο δρόμους).

Τα δεδομένα αυτά φορτώνονται στις κρυφές μνήμες στην αρχή της προσομοίωσης, πριν αρχίσει η λειτουργία του επεξεργαστή. Για τη φόρτωση χρησιμοποιείται η έτοιμη συνάρτηση της Verilog $readmemh. Να σημειωθεί ότι για να λειτουργήσει σωστά η προσομοίωση και για να μπορεί να γίνεται με επιτυχία η προσκόμιση των δεδομένων από τη κρυφή μνήμη δεδομένων, θα πρέπει όλα τα έγκυρα πεδία των γραμμών της κρυφής μνήμης να είναι στο λογικό 1.

Τέλος θα πρέπει, για να αρχίσει η προσομοίωση, να δοθεί πρώτα η επόμενη εντολή στο τερματικό του Linux:

make target_icarus

και μετά

```
make
```

Στη συνέχεια πληκτρολογείται η εντολή:

```
./pipeline
```

και αρχίζει η προσομοίωση.

5. Εκτέλεση προγραμμάτων στον IVM

Σε αυτό το κεφάλαιο θα παρουσιαστεί η εκτέλεση προγραμμάτων στον IVM. Σκοπός είναι να δοθεί μία εικόνα στον αναγνώστη της ροής εκτέλεσης των εντολών στον επεξεργαστή και να επεξηγηθούν διάφορες καταστάσεις που λαμβάνουν χώρα κατά τη διάρκεια λειτουργίας των σταδίων της διοχέτευσης. Να σημειωθεί ότι όλα τα παρακάτω προγράμματα περιλαμβάνονται στο CD, καθώς επίσης και τα αποτελέσματα προσομοιώσεων της εκτέλεσης αυτών.

5.1. Εκτέλεση ενός προγράμματος ταξινόμησης δεδομένων με τον αλγόριθμο της φυσαλίδας

Το πρόγραμμα που επιλέχθηκε είναι η υλοποίηση του αλγορίθμου φυσαλίδας (bubblesort.c). Ο αλγόριθμος αυτός ταξινομεί τα στοιχεία μιας δομής δεδομένων κατά αύξουσα ή φθίνουσα σειρά. Ο κώδικας στη γλώσσα προγραμματισμού C για την υλοποίηση είναι:

```
int A[5] = {3, 8, 4, 2, 21};
int main(int argc, char *argv[])
{
      int i = 0;
      int j = 0;
      int temp = 0;

      for (i=1; i<5; i++)
      {
            for(j=4; j>=i; j--)
            {
                  if (A[j-1] < A[j])
                  {
                        temp = A[j-1];
                        A[j-1] = A[j];
                        A[j] = temp;
                  }
            }
      }

      return 0;
}
```

Όπως φαίνεται και από τον κώδικα ο αλγόριθμος ταξινόμησης θα εφαρμοστεί στους ακέραιους αριθμούς 3, 8, 4, 2 και 21 του πίνακα Α.

Αφού υλοποιήθηκε ο αλγόριθμος ταξινόμησης στη C, σειρά είχε η μεταγλώττισή του. Ο κώδικας μηχανής και συμβολικής γλώσσας που προέκυψε από τη μεταγλώττιση είναι ο ακόλουθος:

```
0000000000000000 <.text>:
   0: 01 00 bb 27       ldah  gp,1(t12)
   4: 00 80 bd 23       lda   gp,-32768(gp)
   8: 00 00 3d 24       ldah  t0,0(gp)
   c: 01 00 bf 20       lda   t4,1
  10: 00 8c 21 20       lda   t0,-29696(t0)
  14: 0c 00 01 21       lda   t7,12(t0)
  18: 0d 00 e0 c3       br    0x50
  1c: 00 00 c2 a0       ldl   t5,0(t1)
  20: 04 00 82 a0       ldl   t3,4(t1)
  24: 23 31 60 40       subl  t2,0x1,t2
  28: a7 09 65 40       cmplt t2,t4,t6
  2c: a1 09 c4 40       cmplt t5,t3,t0
```

```
  30: 02 00 20 e4       beq   t0,0x3c
  34: 00 00 82 b0       stl   t3,0(t1)
  38: 04 00 c2 b0       stl   t5,4(t1)
  3c: fc ff 42 20       lda   t1,-4(t1)
  40: f6 ff ff e4       beq   t6,0x1c
  44: 05 30 a0 40       addl  t4,0x1,t4
  48: a1 b5 a0 40       cmpeq t4,0x5,t0
  4c: 03 00 20 f4       bne   t0,0x5c
  50: 02 04 e8 47       mov   t7,t1
  54: 04 00 7f 20       lda   t2,4
  58: f0 ff ff c3       br    0x1c
  5c: 00 04 ff 47       clr   v0

Disassembly of section .data:

0000000000000c00 <.data>:
 c00: 03 00 00 00
 c04: 08 00 00 00
 c08: 04 00 00 00
 c0c: 02 00 00 00
 c10: 15 00 00 00
```

Στο τμήμα .text είναι το σύνολο των εντολών που θα εκτελεστούν στον επεξεργαστή. Στην αριστερή στήλη είναι οι διευθύνσεις τους, στη μεσαία στήλη είναι η συγκεκριμένη εντολή σε γλώσσα μηχανής (δεκαεξαδική αναπαράσταση) και στη δεξιά στήλη παρουσιάζεται η εντολή σε συμβολική γλώσσα. Το τμήμα .text βρίσκεται στη διεύθυνση 0x0. Το τμήμα .data, ο οποίος περιέχει τα καθολικά δεδομένα του πίνακα Α, βρίσκεται στη διεύθυνση 0xC00. Στην αριστερή στήλη είναι οι διευθύνσεις των δεδομένων και στη δεξιά στήλη είναι οι τιμές τους σε δεκαεξαδική αναπαράσταση. Να σημειωθεί πως επειδή ο τρόπος αποθήκευσης των δεδομένων στην αρχιτεκτονική Alpha είναι μικρού άκρου, οι εντολές και τα δεδομένα αναπαρίστανται παραπάνω με αυτόν τον τρόπο. Για παράδειγμα η εντολή 0100bb27 είναι στην ουσία η εντολή 27bb0001 σε αναπαράσταση μικρού άκρου.

Στη συνέχεια τα tcl και perl scripts επεξεργάζονται το παραγόμενο δυαδικό αρχείο και παράγουν τα επιμέρους αρχεία, τα περιεχόμενα των οποίων θα φορτωθούν στις κρυφές μνήμες εντολών και δεδομένων. Θα πρέπει να σημειωθεί πως η προσομοίωση εκτέλεσης έγινε στο περιβάλλον Icarus Verilog και τα αποτελέσματα αυτής εμφανίζονται σε τερματικό.

Αρχικά δίνεται το περιεχόμενο της γραμμής 12, της σειράς 0, του δρόμου 0 της κρυφής μνήμης δεδομένων. Στη γραμμή αυτή περιέχεται ο πίνακας Α του προγράμματος. Όπως φαίνεται οι αριθμοί δεν είναι ταξινομημένοι σωστά.

```
**********************
*     ARRAY DATA     *
**********************
12th row of bank0, way0:
00000000000000000000000000001500000002000000040000000800000003
```

Στους τρεις πρώτους κύκλους δε συμβαίνει κάτι το ιδιαίτερο αφού μεταφέρονται οι εντολές στα υποστάδια του σταδίου FETCH. Στον 4ο κύκλο όμως:

```
#################################################################
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
-----------------------------------------------
27bb0001
23bd8000
243d0000
20bf0001
#################################################################
```

Οι πρώτες τέσσερις εντολές του προγράμματος εξέρχονται του σταδίου FETCH και εισέρχονται του σταδίου DECODE. Στον 5ο κύκλο:

```
########################################################################
CPU cycle:                    5
---------

########################################################################
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
-----------------------------------------------
20218c00
2101000c
c3e0000d
00000000
########################################################################
INSTRUCTIONS FROM DECODE TO RENAME0 STAGE:
------------------------------------------
277d0001
23bd8000
27a10000
23e50001
```

Οι πρώτες τέσσερις εντολές εισέρχονται στο στάδιο rename0 ενώ οι εντολές με διευθύνσεις 0x10 – 0x18 εξέρχονται του σταδίου FETCH και εισέρχονται στο στάδιο DECODE. Εδώ υπάρχουν δύο σημεία τα οποία πρέπει να επεξηγηθούν. Το πρώτο είναι ότι η εντολή ldah gp, 1(t12) με δεκαεξαδική αναπαράσταση 27bb0001, έχει αλλάξει σε 27d0001 όταν εισέρχεται στο υποστάδιο Rename0 και αυτό γιατί στο στάδιο Decode, έγινε εναλλαγή των καταχωρητών βάσης και προορισμού έτσι ώστε να διευκολυνθεί η περαιτέρω επεξεργασία της εντολής στα επόμενα στάδια. Το δεύτερο είναι ότι στον 5ο κύκλο, στο στάδιο Decode εισέρχονται τρεις εντολές και όχι τέσσερις που είναι το μέγιστο δυνατό. Αυτό συνέβη για τον εξής λόγο: Οι εντολές που προσκομίζονται κάθε φορά από την κρυφή μνήμη είναι οκτώ. Έτσι λοιπόν αρχικά προσκομίστηκαν οι πρώτες οκτώ εντολές. Η έβδομη από αυτές είναι εντολή διακλάδωσης χωρίς συνθήκη (br 0x50). Συνεπώς η όγδοη δε θα πρέπει να προωθηθεί στα επόμενα στάδια και να εκτελεστεί. Συνεπώς, όταν οι εντολές εισέρχονται στο υποστάδιο F1, ανιχνεύεται η εντολή διακλάδωσης από τους αποκωδικοποιητές διακλαδώσεων και έτσι ακυρώνεται η εντολή που την ακολουθεί ώστε να μην εκτελεστεί. Άρα, το σύνολο των εντολών που εισέρχονται στο υποστάδιο F2 και αποθηκεύονται στην προσωρινή μνήμη είναι επτά. Στον κύκλο 4 εξάγονται οι πρώτες τέσσερις και σε αυτόν τον κύκλο οι υπόλοιπες τρεις.

Στον επόμενο κύκλο:

```
########################################################################
CPU cycle:                    6
---------

########################################################################
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
-----------------------------------------------
47e80402
207f0004
c3fffff0
00000000
########################################################################
INSTRUCTIONS FROM DECODE TO RENAME0 STAGE:
------------------------------------------
20218c00
2028000c
c3e0000d
00000000
```

```
PHYSICAL MAPPINGS OF THE ARCHITECTED REGISTERS OF THE INSTRUCTIONS TO
THE REGISTERS OF IVM:
-----------------------------------------------------------------------
rename1 inst0: 277d0001 dest_arch: 29, dest_phys:  32, old_dest_phys:
29
rename1 inst0: 277d0001 srca_arch: 27, srca_phys:  27, srcb_arch: 29,
srcb_phys:  29
rename1 inst1: 23bd8000 dest_arch: 29, dest_phys:  33, old_dest_phys:
32
rename1 inst1: 23bd8000 srca_arch: 29, srca_phys:  32, srcb_arch: 29,
srcb_phys:  32
rename1 inst2: 27a10000 dest_arch:  1, dest_phys:  34, old_dest_phys:
1
rename1 inst2: 27a10000 srca_arch: 29, srca_phys:  33, srcb_arch:  1,
srcb_phys:   1
rename1 inst3: 23e50001 dest_arch:  5, dest_phys:  35, old_dest_phys:
5
rename1 inst3: 23e50001 srca_arch: 31, srca_phys:  31, srcb_arch:  5,
srcb_phys:   5
#######################################################################
```

Από το στάδιο FETCH εισέρχονται εντολές στο στάδιο DECODE με διεύθυνση μνήμης 0x50 και μεγαλύτερη λόγω της διακλάδωσης που έγινε πριν. Για τον ίδιο λόγο που αναφέρθηκε προηγουμένως, με την εντολή διακλάδωσης, ο αριθμός των εντολών που εισέρχονται στο στάδιο DECODE είναι τρεις και όχι τέσσερις. Οι τρεις εντολές με διευθύνσεις 0x10 – 0x18 εισέρχονται κανονικά στο υποστάδιο Rename0 και οι πρώτες τέσσερις εντολές του προγράμματος εξέρχονται του υποσταδίου Rename1 και εισέρχονται στο υποστάδιο Schedule. Αυτό που αξίζει να σχολιαστεί είναι το πώς έχουν μετονομαστεί οι καταχωρητές του.

Όπως αναφέρεται σε προηγούμενη ενότητα, ο λόγος μετονομασίας των καταχωρητών είναι η εξάλειψη των ψευδοεξαρτήσεων. Ο επεξεργαστής IVM περιέχει 80 φυσικούς καταχωρητές, οι 0 – 31 είναι οι ορατοί στον προγραμματιστή, αρχιτεκτονικοί καταχωρητές και οι 32 – 79 είναι οι υπόλοιποι. Αρχικά, στη λίστα SpecFreeRegList, βρίσκονται οι διευθύνσεις των φυσικών καταχωρητών 32 – 79. Ο αρχιτεκτονικός καταχωρητής προορισμού $gp (`dest_arch`), της εντολής 0x277d0001, αντιστοιχίζεται με τον πρώτο ελεύθερο, φυσικό καταχωρητή, ο οποίος είναι ο $32 (`dest_phys` - να σημειωθεί πως τα ονόματα και οι διευθύνσεις των καταχωρητών ταυτίζονται, οπότε γίνεται έμμεση αναφορά στις διευθύνσεις αυτών). Στον καταχωρητή 32 πλέον θα αποθηκευτεί το αποτέλεσμα που προκύπτει από την εκτέλεση της εντολής. Οι αρχιτεκτονικοί καταχωρητές της, οι οποίοι αποτελούν τους τελεστέους (`srca_arch`, `srcb_arch`), μένουν ως έχουν.

Η εντολή 0x23bd8000 έχει και αυτή ως αρχιτεκτονικό καταχωρητή προορισμού τον gp. Αυτός αντιστοιχίζεται στον επόμενο ελεύθερο φυσικό καταχωρητή, ο οποίος είναι ο $33. Ο αρχιτεκτονικός τελεστέος της εντολής είναι επίσης ο $gp. Όμως το περιεχόμενο του τελεστέου αυτού είναι προφανές ότι εξαρτάται από την προηγούμενη εντολή, την 0x277d0001 (ldah gp,1(t12)). Συνεπώς εντοπίζεται η εξάρτηση της εντολής 0x23bd8000 με την προηγούμενή της και έτσι ο αρχιτεκτονικός τελεστέος της αντιστοιχίζεται με το φυσικό καταχωρητή 32. Η μετονομασία των καταχωρητών των υπόλοιπων δύο εντολών, από τις πρώτες τέσσερις του προγράμματος, γίνεται με την ίδια τακτική.

Στον 7^{ο} κύκλο:

```
#######################################################################
CPU cycle:                    7
---------

#######################################################################
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
-----------------------------------------------
00000000
00000000
```

```
00000000
00000000
########################################################################
INSTRUCTIONS FROM DECODE TO RENAME0 STAGE:
----------------------------------------
47e80402
23e30004
c3fffff0
00000000
########################################################################
INSTRUCTIONS FROM RENAME TO SCHEDULE AND MEM:
--------------------------------------------
277d0001
23bd8000
27a10000
23e50001
```

Δεν εισέρχεται καμιά εντολή στο στάδιο DECODE. Μετά την εντολή διακλάδωσης χωρίς συνθήκη 0xc3e0000d, προσκομίστηκαν από την κρυφή μνήμη εντολών οκτώ εντολές με την πρώτη να είναι στη διεύθυνση 0x50. Όμως η τρίτη από αυτές είναι επίσης εντολή διακλάδωσης χωρίς συνθήκη (0xc3fffff0) και έτσι προέκυψε καθυστέρηση ενός κύκλου για να προσκομιστούν οι επόμενες οκτώ από τη σωστή διεύθυνση. Οπότε, ενώ στον 6^{ο} κύκλο εξάγονται από το στάδιο F2 οι τρεις εντολές, στις οποίες συμπεριλαμβάνεται η εντολή διακλάδωσης 0xc3fffff0, στον 7^{ο} δεν εξάγεται καμία διότι δεν υπάρχει άλλη στην προσωρινή μνήμη του σταδίου F2. Οι τέσσερις πρώτες εντολές του προγράμματος προωθούνται στο υποστάδιο Schedule και στο στάδιο μνήμης.

Στον 8^{ο} κύκλο:

```
########################################################################
CPU cycle:                    8
---------

########################################################################
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
----------------------------------------------
a0c20000
a0820004
40603123
406509a7
########################################################################
INSTRUCTIONS FROM DECODE TO RENAME0 STAGE:
----------------------------------------
00000000
00000000
00000000
00000000
########################################################################
INSTRUCTIONS FROM RENAME TO SCHEDULE AND MEM:
--------------------------------------------
20218c00
2028000c
c3e0000d
00000000


INSTRUCTIONS WHICH WERE SELECTED TO BE FORWARDED TO FUNCTIONAL UNITS:
--------------------------------------------------------------------
277d0001
```

```
23e50001
00000000
00000000

retired          0 instructions this cycle

Scheduler_Full: 0
first n locs in scheduler(head 0: 0d, tail 0: 4d):
0: valid: 01, inst: 277d0001   0, dest_phys:  32, srca:  27:1:1.1,
srcb:  29:0:1.1
1: valid: 01, inst: 23bd8000   1, dest_phys:  33, srca:  32:1:0.0,
srcb:  32:0:0.0
2: valid: 01, inst: 27a10000   2, dest_phys:  34, srca:  33:1:0.0,
srcb:   1:0:1.1
3: valid: 01, inst: 23e50001   3, dest_phys:  35, srca:  31:1:1.1,
srcb:   5:0:1.1
4: valid: 00, inst: 00000000   0, dest_phys:   0, srca:   0:0:1.1,
srcb:   0:0:1.1
5: valid: 00, inst: 00000000   0, dest_phys:   0, srca:   0:0:1.1,
srcb:   0:0:1.1
 6: valid: 00, inst: 00000000   0, dest_phys:   0, srca:   0:0:1.1,
srcb:   0:0:1.1
 7: valid: 00, inst: 00000000   0, dest_phys:   0, srca:   0:0:1.1,
srcb:   0:0:1.1
 8: valid: 00, inst: 00000000   0, dest_phys:   0, srca:   0:0:1.1,
srcb:   0:0:1.1
 9: valid: 00, inst: 00000000   0, dest_phys:   0, srca:   0:0:1.1,
srcb:   0:0:1.1
```

Η ροή εκτέλεσης συνεχίζεται κανονικά. Εξάγονται οι εντολές από το στάδιο F2 με αρχική διεύθυνση 0x1c. Οι τέσσερις αρχικές εντολές του προγράμματος έχουν εισαχθεί στον πίνακα δρομολόγησης και περιμένουν να διεκπεραιωθούν/ξεκινήσουν. Οι εντολές που αποφασίζεται να διεκπεραιωθούν/ξεκινήσουν στις λειτουργικές μονάδες είναι οι 0x277d0001 (ldah gp, 1(t12)) και 0x23e50001 (lda t4,1) διότι δεν εξαρτώνται από καμία άλλη. Πιο συγκεκριμένα, επειδή και οι δύο είναι απλές αριθμητικές εντολές πρόσθεσης θα διεκπεραιωθούν/ξεκινήσουν στις λειτουργικές μονάδες ALU0 και ALU1.

Στον 9ο κύκλο:

```
########################################################################
CPU cycle:                    9
---------

########################################################################
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
-----------------------------------------------
40c409a1
e4200002
2042fffc
e4fffff6
########################################################################
INSTRUCTIONS FROM DECODE TO RENAME0 STAGE:
------------------------------------------
a3e20000
a3e20004
40603123
406509a7
```

```
######################################################################
INSTRUCTIONS FROM RENAME TO SCHEDULE AND MEM:
---------------------------------------------
47e80402
23e30004
c3fffff0
00000000


INSTRUCTIONS WHICH WERE SELECTED TO BE FORWARDED TO FUNCTIONAL UNITS:
---------------------------------------------------------------------
23bd8000
00000000
00000000
c3e0000d
00000000
00000000
######################################################################
INSTRUCTIONS THAT FETCHED THEIR OPERANDS FROM REGREAD STAGE:
-----------------------------------------------------------
277d0001
23e50001
00000000
00000000
00000000
00000000
```

Οι δύο εντολές που αποφασίστηκε να διεκπεραιωθούν/ξεκινήσουν, ανακτούν τα περιεχόμενά τους από το υποστάδιο RegRead και περιμένουν να εκτελεστούν στον επόμενο κύκλο από τις λειτουργικές μονάδες ALU0 και ALU1. Παράλληλα, από το στάδιο προσκόμισης εξάγονται δύο εντολές που ανήκουν στην οκτάδα με διεύθυνση μνήμης 0x1c (0x40c409a1, 0xe4200002) και άλλες δύο (0x2042fffc, 0xe4ffffff6) που ανήκουν στην οκτάδα με διεύθυνση μνήμης 0x3c. Η 0xe4200002 είναι εντολή διακλάδωσης υπό συνθήκη (beq t0,0x3c) και παρόλο που στην οκτάδα που ανήκει είναι η $6^{η}$ κατά σειρά εντολή οι άλλες δύο που ακολουθούν παρακάμπτονται και συνεχίζεται η προώθηση των εντολών από τη διεύθυνση 0x3c. Ο λόγος παράκαμψης είναι επειδή ένα τμήμα της σχεδίασης είναι εξαιρετικά συντηρητικό και αφορά το στάδιο F1. Πιο συγκεκριμένα, μόλις οι αποκωδικοποιητές διακλαδώσεων ανιχνεύσουν ότι υπάρχει στην οκτάδα εντολών μία εντολή διακλάδωσης ακυρώνονται όλες όσες ακολουθούν αυτής. Όμως, αν μελετήσει κανείς το συγκεκριμένο κώδικα θα παρατηρήσει ότι στο συγκεκριμένο στιγμιότυπο εκτέλεσης δεν πρέπει να παρακαμφθούν οι δύο εντολές που ακολουθούν την εντολή διακλάδωσης. Έτσι, αυτή η στατική προσέγγιση επιβαρύνει με επιπλέον κύκλους την εκτέλεση του προγράμματος. Διότι από το σημείο αυτό και μετά αλλάζει λανθασμένα η ροή των εντολών.

Στον $10^{ο}$ κύκλο η ροή εκτέλεσης συνεχίζεται από το λανθασμένο μονοπάτι εντολών. Δηλαδή, ενώ είχε προσκομιστεί πιο πριν από την κρυφή μνήμη η οκτάδα εντολών με διεύθυνση 0x3c, στην οκτάδα αυτή η δεύτερη εντολή είναι διακλάδωση υπό συνθήκη (εντολή 0xe4ffffff6, διεύθυνση 0x40). Έτσι οι υπόλοιπες έξι εντολές ακυρώθηκαν και η μόνη που απέμεινε μετά την εντολή διακλάδωσης για να εκτελεστεί είναι η 0x47ff0400, η οποία είναι τελευταία στο πρόγραμμα.

Στον $11^{ο}$ κύκλο:

```
######################################################################
CPU cycle:                  11
----------

######################################################################
…
…
…
```

```
######################################################################
INSTRUCTIONS WHICH FINISHED THEIR EXECUTION:
------------------------------------------
SIMPLE ALU 0 instruction: 277d0001
SIMPLE ALU 1 instruction: 23e50001
BRANCH instruction: 00000000
COMPLEX ALU instruction: 00000000
LOAD 1 instruction: 000
LOAD 2 instruction: 000
STORE 1 instruction: 000
STORE 2 instruction: 000
```

Η πρώτη και η τέταρτη εντολή τελειώνουν με επιτυχία την εκτέλεσή τους και στον 12ο κύκλο τελειώνει η εντολή 0x23bd8000, η οποία είναι δεύτερη χρονικά στο πρόγραμμα. Εδώ φαίνεται η εκτός σειράς εκτέλεση των εντολών.

Στον 13ο κύκλο ολοκληρώνεται η πρώτη εντολή με διεύθυνση 0x0. Η εντολή 0x23e50001 δεν ολοκληρώνεται, παρόλο που έχει τελειώσει και αυτή, διότι είναι τέταρτη στο πρόγραμμα και επειδή η ολοκλήρωση των εντολών γίνεται σε σειρά πρέπει να περιμένει να ολοκληρωθούν πρώτα η δεύτερη και η τρίτη εντολή. Επίσης, στον κύκλο αυτόν τελειώνει την εκτέλεσή της, μαζί με άλλες, η τρίτη εντολή (0x27a10000).

```
######################################################################
CPU cycle:                    13
---------

######################################################################
…
…
…
######################################################################
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
----------------------------------------------------------------------
0000000000000000
…
…
…
######################################################################
INSTRUCTIONS WHICH FINISHED THEIR EXECUTION:
------------------------------------------
SIMPLE ALU 0 instruction: 27a10000
SIMPLE ALU 1 instruction: 23e30004
BRANCH instruction: c3fffff0
COMPLEX ALU instruction: 00000000
LOAD 1 instruction: 000
LOAD 2 instruction: 000
STORE 1 instruction: 000
STORE 2 instruction: 000
######################################################################
```

Στον 14ο κύκλο ολοκληρώνεται η δεύτερη, χρονικά, εντολή.

```
######################################################################
CPU cycle:                    14
---------
######################################################################
…
```

```
…
…
######################################################################
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
----------------------------------------------------------------------
0000000000000004
```

Στον 15ο κύκλο ολοκληρώνονται επίσης η τρίτη και η τέταρτη εντολή.

```
######################################################################
CPU cycle:                    15
---------

######################################################################
…
…
…
######################################################################
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
----------------------------------------------------------------------
0000000000000008
000000000000000c
```

Στον 27ο κύκλο:

```
######################################################################
CPU cycle:                    27
---------

######################################################################
…
…
…
######################################################################
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
----------------------------------------------------------------------
0000000000000030
```

Ολοκληρώνεται η εντολή διακλάδωσης με διεύθυνση μνήμης 0x30 (0xe4200002). Όμως λόγω εσφαλμένης πρόβλεψης της διακλάδωσης (η πρόβλεψη ήταν ότι λαμβάνεται, ενώ δε λαμβάνεται τελικά) δίνεται εντολή για εκκένωση και οι πρώτες τέσσερις εντολές, μετά την εκκένωση, εισέρχονται στο στάδιο DECODE στον 32ο κύκλο.

Με παρόμοιο τρόπο συνεχίζεται η εκτέλεση του προγράμματος. Όταν ολοκληρωθούν οι εντολές και αποσυρθούν οι εντολές αποθήκευσης, το πρόγραμμα τελειώνει την εκτέλεσή του και όπως φαίνεται και ακολούθως από τα αποτελέσματα της προσομοίωσης:

```
**********************
*     ARRAY DATA     *
**********************
A[0] Address: 3072 12th row of bank0, way0:
00000000000000000000000000000002000000030000000400000008000000 15
######################################################################

TOTAL BRANCHES:
---------------
total branches:
00000000000000000000000000000000000000000000000000000011000
```

```
TOTAL FLUSHES:
--------------
total flushes:
00000000000000000000000000000000000000000000000000000000001100
flushes due to direct mispredictions:
00000000000000000000000000000000000000000000000000000000001100
flushes due to aliases:
00000000000000000000000000000000000000000000000000000000000000
```

Τα δεδομένα έχουν ταξινομηθεί απολύτως σωστά. Ο συνολικός αριθμός κύκλων που διήρκησε η εκτέλεση του προγράμματος ήταν **276**. Αξιοσημείωτο είναι το γεγονός πως δόθηκε 12 φορές εντολή για εκκένωση της διοχέτευσης λόγω εσφαλμένων προβλέψεων των αποτελεσμάτων των άμεσων διακλαδώσεων (δες από πάνω, flushes due to direct mispredictions και total flushes), ενώ ο συνολικός αριθμός διακλαδώσεων (δες στην προηγούμενη σελίδα, total branches) ήταν 24. Δηλαδή το ποσοστό αποτυχίας των προβλέψεων έφτασε το 50%. Οι λανθασμένες προβλέψεις οφείλονται κυρίως στη στατική, συντηρητική πολιτική πρόβλεψης των διακλαδώσεων όπου θεωρούνται όλες ληφθείσες.

5.2. Εκτέλεση εντολών μνήμης

Το πιο δυσνόητο σημείο στην κατανόηση της λειτουργίας του σταδίου MEM είναι η ανίχνευση και ο χειρισμός των εξαρτήσεων μεταξύ των εντολών μνήμης. Για το λόγο αυτό, στην παρούσα ενότητα, θα παρουσιαστούν οι εκτελέσεις μικρών προγραμμάτων, υλοποιημένα στη συμβολική γλώσσα Alpha, που αναδεικνύουν αυτά τα χαρακτηριστικά.

Όπως είχε αναφερθεί στη σχετική ενότητα, το στάδιο MEM ελέγχει αν υπάρχουν μεταξύ των εντολών μνήμης χρονικές και RAW εξαρτήσεις. Για να εξασφαλιστεί η αξιόπιστη λειτουργία της εκτός σειράς εκτέλεσης και της ολοκλήρωσης σε σειρά των εντολών αυτών σχεδιάστηκε ένας συντηρητικός μηχανισμός που διατηρεί την ακολουθιακή συνέπεια μνήμης.

Ακολουθεί το πρώτο πρόγραμμα (check_mem_deps.s):

```
Disassembly of section .text:

0000000000000000 <.text>:
   0: 01 94 20 40       addq  t0,0x4,t0
   4: 02 94 45 40       addq  t1,0x2c,t1
   8: 03 34 40 40       addq  t1,0x1,t2
   c: 04 b4 60 40       addq  t2,0x5,t3
  10: 07 b4 81 40       addq  t3,0xd,t6
  14: 08 34 00 41       addq  t7,0x1,t7
  18: 06 d4 a0 40       addq  t4,0x6,t5
  1c: 40 01 ff b3       stl   zero,320(zero)
  20: 40 01 ff b3       stl   zero,320(zero)
  24: 40 01 ff b3       stl   zero,320(zero)
  28: 40 01 ff b3       stl   zero,320(zero)
  2c: 40 01 ff b3       stl   zero,320(zero)
  30: 40 01 ff b3       stl   zero,320(zero)
  34: 40 01 ff b3       stl   zero,320(zero)
  38: 40 01 ff b3       stl   zero,320(zero)
  3c: 0c 0c 3f b0       stl   t0,3084(zero)
  40: 08 0c 5f b4       stq   t1,3080(zero)
  44: 08 0c 7f a4       ldq   t2,3080(zero)

Disassembly of section .data:

0000000000000c00 <.data>:
 c00: 03 00 00 00       call_pal    0x3
 c04: 08 00 00 00       call_pal    0x8
 c08: 04 00 00 00       call_pal    0x4
```

```
 c0c: 02 00 00 00       draina
 c10: 15 00 00 00       call_pal    0x15
```

Σκοπός της επίδειξης της εκτέλεσής του είναι για να καταλάβει ο αναγνώστης στην πράξη πως λειτουργεί ο μηχανισμός με τις RAW εξαρτήσεις. Όπως φαίνεται και από τον κώδικα το ενδιαφέρον εστιάζεται στις τρεις τελευταίες εντολές (αυτές που έχουν διευθύνσεις μνήμης 0x3c, 0x40 και 0x44). Η εντολή ldq έχει εξάρτηση RAW με την εντολή stq διότι γίνεται και από τις δύο προσπέλαση στην ίδια διεύθυνση μνήμης και επίσης διαχειρίζονται το ίδιο μέγεθος δεδομένων. Όμως η εντολή ldq έχει εξάρτηση RAW και με την stl, διότι προσπελάζουν την ίδια διεύθυνση τετραπλής λέξης. Παρόλο που για την εντολή ldq έχει νόημα μόνο η RAW εξάρτησή της από την stq, τα αποτελέσματα της προσομοίωσης δείχνουν ότι καθυστερεί σημαντικά η ολοκλήρωση της ldq, εξαιτίας της stl.

Πιο συγκεκριμένα:

```
######################################################################
CPU cycle:                    16
---------

######################################################################
.
.
.
finished1_in: v: 1, robid:  14, addr: 0000000000000140, data:
0000000000000000, type: 1, destreg:  46, size: 2, writedest: 0
```

Στον 16ο κύκλο παράγεται η διεύθυνση της εντολής stl.

Στον 17ο κύκλο:

```
######################################################################
CPU cycle:                    17
---------

######################################################################
.
.
.
finished1_in: v: 1, robid:  16, addr: 0000000000000c08, data:
000000000000002c, type: 1, destreg:  48, size: 3, writedest: 0

finished2_in: v: 1, robid:  17, addr: 0000000000000c08, data:
0000000000000000, type: 0, destreg:  49, size: 3, writedest: 1

Values sent to data cache:
valid2: 1, addr2: 0000000000000c08, data2: 0000000000000000, size2: 3,
type2: 0, mobid2:  0, destreg2:  49, writedest2: 1
```

Η εντολή stl στέλνεται στο ROB για ολοκλήρωση και παράγεται η διεύθυνση προσπέλασης της εντολής stq και της εντολής ldq. Επομένως, η εντολή ldq στέλνεται στην κρυφή μνήμη δεδομένων για να εξυπηρετηθεί.

Στον 18ο κύκλο:

```
######################################################################
CPU cycle:                    18
---------

######################################################################
.
.
.
stb_alias2: 0000000100000000
```

Η εντολή stq στέλνεται στο ROB να ολοκληρωθεί, ανιχνεύεται για πρώτη φορά ότι η ldq έχει εξάρτηση RAW με την stl. Με την stq δεν έχει ανιχνευτεί η εξάρτηση RAW διότι δεν είναι διαθέσιμη ακόμα η διεύθυνση προσπέλασης της συγκεκριμένης εντολής αποθήκευσης.

Στον 19ο κύκλο:

```
######################################################################
CPU cycle:                    19
---------

######################################################################
.
.
.
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
----------------------------------------------------------------------
0000000000000038
000000000000003c

Values returned from data cache:
hit2: 1, data2: 0000000200000004, mobid2:  0, addr2: 0000000000000c08,
size2: 3, destreg2:  49
```

Η εντολή stl ολοκληρώνεται και στη συνέχεια περιμένει να αλλάξει την κατάσταση της μνήμης και να αποσυρθεί. Επίσης ολοκληρώνεται η εξυπηρέτηση της ldq από την κρυφή μνήμη δεδομένων. Είναι προφανές ότι τα δεδομένα αυτά που προσκομίστηκαν δεν είναι τα σωστά διότι η εντολή stq, από την οποία εξαρτάται η ldq, δεν έχει αποσυρθεί έτσι ώστε να αλλάξει την κατάσταση της μνήμης. Παρόλο που η ldq έχει τελειώσει την εκτέλεσή της και μπορεί να σταλεί στο ROB για ολοκλήρωση, η εξάρτησή της (ψευδο-εξάρτηση στην ουσία) από την stl της επιβάλλει να εκτελεστεί ξανά και να μην υπάρξει καμία πρόοδος. Το ίδιο πράγμα επαναλαμβάνεται μέχρις ότου αποσυρθεί η εντολή stl στον 28ο κύκλο. Τότε η εντολή ldq δεν έχει πλέον RAW εξάρτηση με την stl και προωθούνται τα δεδομένα της στον επόμενο κύκλο στα υπόλοιπα στάδια της διοχέτευσης. Η ίδια ολοκληρώνεται στον 31ο κύκλο και τότε τελειώνει και η εκτέλεση του προγράμματος.

```
######################################################################
CPU cycle:                    31
---------

######################################################################
.
.
.
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
----------------------------------------------------------------------
0000000000000044
```

Αν δεν υπήρχε η εξάρτηση RAW με την εντολή stl, η ldq θα είχε προωθήσει τα δεδομένα της στον 22ο κύκλο και όχι στον 29ο. Η εντολή θα ολοκληρωνόταν στον 24ο κύκλο, ενώ ο συνολικός αριθμός των κύκλων που θα απαιτούνταν για την ολοκλήρωση της εκτέλεσης του προγράμματος θα ήταν 28. Η διαφορά δεν είναι ιδιαίτερα μεγάλη, σε σχέση με τους 31, αλλά αυτό οφείλεται στο γεγονός ότι η εντολή ldq είναι τελευταία στο πρόγραμμα και δεν εξαρτώνται άλλες εντολές από την εκτέλεσή της.

Έστω για παράδειγμα το πρόγραμμα B (check_memB.s), το οποίο είναι παρόμοιο με το προηγούμενο που παρουσιάστηκε:

```
Disassembly of section .text:

0000000000000000 <.text>:
```

```
   0: 01 94 20 40       addq  t0,0x4,t0
   4: 02 94 45 40       addq  t1,0x2c,t1
   8: 03 34 40 40       addq  t1,0x1,t2
   c: 04 b4 60 40       addq  t2,0x5,t3
  10: 07 b4 81 40       addq  t3,0xd,t6
  14: 08 34 00 41       addq  t7,0x1,t7
  18: 06 d4 a0 40       addq  t4,0x6,t5
  1c: 40 01 ff b3       stl   zero,320(zero)
  20: 40 01 ff b3       stl   zero,320(zero)
  24: 40 01 ff b3       stl   zero,320(zero)
  28: 40 01 ff b3       stl   zero,320(zero)
  2c: 40 01 ff b3       stl   zero,320(zero)
  30: 40 01 ff b3       stl   zero,320(zero)
  34: 40 01 ff b3       stl   zero,320(zero)
  38: 40 01 ff b3       stl   zero,320(zero)
  3c: 0c 0c 3f b0       stl   t0,3084(zero)
  40: 08 0c 5f b4       stq   t1,3080(zero)
  44: 1f 14 e0 43       addq  zero,0,zero
  48: 1f 14 e0 43       addq  zero,0,zero
  4c: 1f 14 e0 43       addq  zero,0,zero
  50: 1f 14 e0 43       addq  zero,0,zero
  54: 1f 14 e0 43       addq  zero,0,zero
  58: 1f 14 e0 43       addq  zero,0,zero
  5c: 1f 14 e0 43       addq  zero,0,zero
  60: 1f 14 e0 43       addq  zero,0,zero
  64: 1f 14 e0 43       addq  zero,0,zero
  68: 1f 14 e0 43       addq  zero,0,zero
  6c: 1f 14 e0 43       addq  zero,0,zero
  70: 1f 14 e0 43       addq  zero,0,zero
  74: 1f 14 e0 43       addq  zero,0,zero
  78: 1f 14 e0 43       addq  zero,0,zero
  7c: 1f 14 e0 43       addq  zero,0,zero
  80: 08 0c 7f a4       ldq   t2,3080(zero)
  84: 01 94 60 40       addq  t2,0x4,t0
  88: 02 94 25 40       addq  t0,0x2c,t1
  8c: 04 94 40 40       addq  t1,0x4,t3
  90: 05 94 85 40       addq  t3,0x2c,t4
  94: 06 94 a0 40       addq  t4,0x4,t5
  98: 07 94 c5 40       addq  t5,0x2c,t6

Disassembly of section .data:

0000000000000c00 <.data>:
 c00: 03 00 00 00       call_pal    0x3
 c04: 08 00 00 00       call_pal    0x8
 c08: 04 00 00 00       call_pal    0x4
 c0c: 02 00 00 00       draina
 c10: 15 00 00 00       call_pal    0x15
```

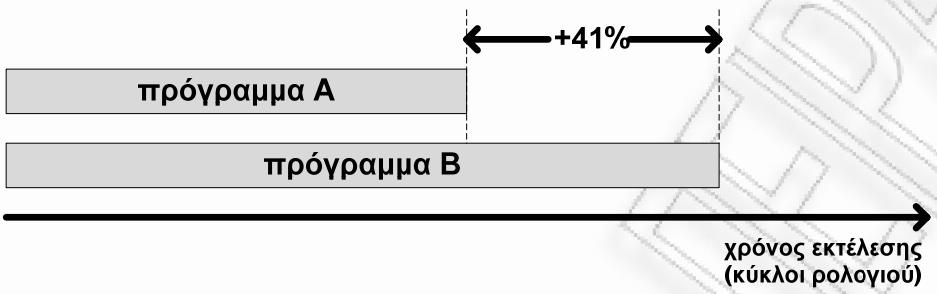
Δηλαδή, υπάρχουν πάλι οι τρεις εντολές μνήμης (στις διευθύνσεις 0x3c, 0x40 και 0x80) για τις οποίες ισχύουν οι προαναφερθείσες εξαρτήσεις RAW που ίσχυαν και για το προηγούμενο πρόγραμμα. Η κύρια διαφορά είναι ότι μετά την εντολή ldq ακολουθούν έξι εντολές addq στις οποίες υπάρχει αλυσιδωτή εξάρτηση RAW, με την πρώτη εντολή πρόσθεσης να εξαρτάται από το αποτέλεσμα της εντολής φόρτωσης. Τότε, ο χρόνος εκτέλεσής του είναι **41** κύκλοι.

Έστω ένα πρόγραμμα Α (check_memA.s), το οποίο έχει τον ίδιο αριθμό εντολών, οι εντολές είναι του ίδιου τύπου και βρίσκονται στην ίδια χρονική σειρά με αυτές του Β. Η μόνη διαφορά είναι ότι η ldq δεν έχει RAW εξάρτηση με την stl παρά μόνο με την stq.

```
Disassembly of section .text:

0000000000000000 <.text>:
   0: 01 94 20 40       addq  t0,0x4,t0
   4: 02 94 45 40       addq  t1,0x2c,t1
   8: 03 34 40 40       addq  t1,0x1,t2
   c: 04 b4 60 40       addq  t2,0x5,t3
  10: 07 b4 81 40       addq  t3,0xd,t6
  14: 08 34 00 41       addq  t7,0x1,t7
  18: 06 d4 a0 40       addq  t4,0x6,t5
  1c: 40 01 ff b3       stl   zero,320(zero)
  20: 40 01 ff b3       stl   zero,320(zero)
  24: 40 01 ff b3       stl   zero,320(zero)
  28: 40 01 ff b3       stl   zero,320(zero)
  2c: 40 01 ff b3       stl   zero,320(zero)
  30: 40 01 ff b3       stl   zero,320(zero)
  34: 40 01 ff b3       stl   zero,320(zero)
  38: 40 01 ff b3       stl   zero,320(zero)
  3c: 40 01 3f b0       stl   t0,320(zero)
  40: 08 0c 5f b4       stq   t1,3080(zero)
  44: 1f 14 e0 43       addq  zero,0,zero
  48: 1f 14 e0 43       addq  zero,0,zero
  4c: 1f 14 e0 43       addq  zero,0,zero
  50: 1f 14 e0 43       addq  zero,0,zero
  54: 1f 14 e0 43       addq  zero,0,zero
  58: 1f 14 e0 43       addq  zero,0,zero
  5c: 1f 14 e0 43       addq  zero,0,zero
  60: 1f 14 e0 43       addq  zero,0,zero
  64: 1f 14 e0 43       addq  zero,0,zero
  68: 1f 14 e0 43       addq  zero,0,zero
  6c: 1f 14 e0 43       addq  zero,0,zero
  70: 1f 14 e0 43       addq  zero,0,zero
  74: 1f 14 e0 43       addq  zero,0,zero
  78: 1f 14 e0 43       addq  zero,0,zero
  7c: 1f 14 e0 43       addq  zero,0,zero
  80: 08 0c 7f a4       ldq   t2,3080(zero)
  84: 01 94 60 40       addq  t2,0x4,t0
  88: 02 94 25 40       addq  t0,0x2c,t1
  8c: 04 94 40 40       addq  t1,0x4,t3
  90: 05 94 85 40       addq  t3,0x2c,t4
  94: 06 94 a0 40       addq  t4,0x4,t5
  98: 07 94 c5 40       addq  t5,0x2c,t6

Disassembly of section .data:

0000000000000c00 <.data>:
 c00: 03 00 00 00       call_pal    0x3
 c04: 08 00 00 00       call_pal    0x8
 c08: 04 00 00 00       call_pal    0x4
 c0c: 02 00 00 00       draina
 c10: 15 00 00 00       call_pal    0x15
```

Τότε ο χρόνος εκτέλεσής του είναι **29** κύκλοι.

Είναι προφανές ότι ο χρόνος εκτέλεσης του προγράμματος Α είναι σαφώς μικρότερος σε σχέση με αυτόν του Α. Αυτό συμβαίνει γιατί στο πρόγραμμα Α δεν υπάρχει εξάρτηση RAW της εντολής ldq με την stl, παρά μόνο με την stq, η οποία είναι αυτή που έχει σημασία. Αυτό έχει σαν αποτέλεσμα να μη δημιουργείται ανούσια καθυστέρηση στο τελείωμα της εκτέλεσης της ldq και κατ' επέκταση στην προώθηση του δεδομένου της στις εντολές πρόσθεσης που το χρειάζονται για να εκτελεστούν. Έτσι, οι εντολές πρόσθεσης ολοκληρώνονται και αυτές πολύ πιο γρήγορα και επομένως η εκτέλεση του προγράμματος τελειώνει πιο σύντομα.



Εικόνα 5.1: Σύγκριση των χρόνων εκτέλεσης των προγραμμάτων Α και Β. Η εκτέλεση του Β είναι κατά 41% μεγαλύτερη λόγω της εξάρτησης της εντολής ldq με την stl

Στο επόμενο πρόγραμμα (check_mem_deps2.s) αναδεικνύεται η περίπτωση όπου ανιχνεύεται η εξάρτηση RAW μιας εντολής φόρτωσης με μία εντολή αποθήκευσης, εφόσον η τελευταία ολοκληρώσει την εκτέλεσή της.

```
Disassembly of section .text:

0000000000000000 <.text>:
   0: 01 94 20 40       addq  t0,0x4,t0
   4: 02 94 25 40       addq  t0,0x2c,t1
   8: 03 94 40 40       addq  t1,0x4,t2
   c: 04 b4 60 40       addq  t2,0x5,t3
  10: 05 34 80 40       addq  t3,0x1,t4
  14: 07 74 a0 40       addq  t4,0x3,t6
  18: 08 94 e5 40       addq  t6,0x2c,t7
  1c: 08 0c 1f b5       stq   t7,3080(zero)
  20: 01 94 20 40       addq  t0,0x4,t0
  24: 02 94 25 40       addq  t0,0x2c,t1
  28: 03 94 40 40       addq  t1,0x4,t2
  2c: 04 b4 60 40       addq  t2,0x5,t3
  30: 05 34 80 40       addq  t3,0x1,t4
  34: 07 74 a0 40       addq  t4,0x3,t6
  38: 08 94 e5 40       addq  t6,0x2c,t7
  3c: 08 0c 1f b5       stq   t7,3080(zero)
  40: 01 94 20 40       addq  t0,0x4,t0
  44: 02 94 25 40       addq  t0,0x2c,t1
  48: 03 94 40 40       addq  t1,0x4,t2
  4c: 04 b4 60 40       addq  t2,0x5,t3
  50: 05 34 80 40       addq  t3,0x1,t4
  54: 07 74 a0 40       addq  t4,0x3,t6
  58: 08 94 e5 40       addq  t6,0x2c,t7
  5c: 08 0c df a6       ldq   t8,3080(zero)

Disassembly of section .data:

0000000000000c00 <.data>:
 c00: 03 00 00 00       call_pal    0x3
 c04: 08 00 00 00       call_pal    0x8
```

```
 c08: 04 00 00 00       call_pal    0x4
 c0c: 02 00 00 00       draina
 c10: 15 00 00 00       call_pal    0x15
```

Το ενδιαφέρον επικεντρώνεται για μια ακόμη φορά στις εντολές μνήμης και συγκεκριμένα αυτές που βρίσκονται στις θέσεις μνήμης 0x1c, 0x3c (οι δύο εντολές stq) και 0x5c αντίστοιχα. Οι εντολές stq εισέρχονται στο υποστάδιο Schedule και στο στάδιο MEM στον $8^{ο}$ και $10^{ο}$ κύκλο αντίστοιχα.

```
######################################################################
CPU cycle:                      8
---------

######################################################################
.
.
.
INSTRUCTIONS FROM RENAME TO SCHEDULE AND MEM:
-------------------------------------------
40803405
40a07407
40e59408
b51f0c08
.
.
.
######################################################################
CPU cycle:                     10
---------

######################################################################
.
.
.
INSTRUCTIONS FROM RENAME TO SCHEDULE AND MEM:
-------------------------------------------
40803405
40a07407
40e59408
b51f0c08
```

Ενώ οι αντίστοιχες καταχωρήσεις στην ουρά STQ, του σταδίου MEM δημιουργούνται έναν κύκλο μετά, λείπουν οι διευθύνσεις προσπέλασης των εντολών. Αυτό συμβαίνει λόγω των RAW εξαρτήσεων. Δηλαδή, όπως φαίνεται από τον κώδικα, τα περιεχόμενα των καταχωρητών βάσης των εντολών αποθήκευσης εξαρτώνται από τις πιο πρόσφατες εντολές πρόσθεσης που προηγούνται χρονικά (αυτές που είναι στις διευθύνσεις 0x18 και 0x38 αντίστοιχα) και οι οποίες με τη σειρά τους εξαρτώνται επίσης από τις πιο πρόσφατες, προηγούμενες εντολές πρόσθεσης (αυτές που είναι στις διευθύνσεις 0x14 και 0x34) και ούτω καθεξής. Έτσι, αυτή η αλυσίδα εξαρτήσεων RAW δημιουργεί σημαντική καθυστέρηση (10 κύκλους) στη διεκπεραίωση/εκκίνηση των εντολών αποθήκευσης.

Η εντολή φόρτωσης ldq, δεν έχει καμία εξάρτηση όσον αφορά τον καταχωρητή βάσης και έτσι στον $13^{ο}$ κύκλο:

```
######################################################################
CPU cycle:                     13
---------

######################################################################
.
.
```

```
.

INSTRUCTIONS WHICH WERE SELECTED TO BE FORWARDED TO FUNCTIONAL UNITS:
--------------------------------------------------------------------
40a07407
4060b404
00000000
00000000
a7ff0c08
00000000
```

Διεκπεραιώνεται/ξεκινάει. Στον 16ο κύκλο παράγεται η διεύθυνσή της και αποστέλλεται στην κρυφή μνήμη δεδομένων για να εξυπηρετηθεί.

```
####################################################################
CPU cycle:                    16
---------

####################################################################
.
.
.
finished1_in: v: 1, robid:  23, addr: 0000000000000c08, data:
0000000000000000, type: 0, destreg:  55, size: 3, writedest: 1
```

Στον 18ο κύκλο:

```
####################################################################
CPU cycle:                    18
---------

####################################################################
.
.
.
Values returned from data cache:
hit1: 1, data1: 0000000200000004, mobid1:  0, addr1: 0000000000000c08,
size1: 3, destreg1:  55
```

Τελειώνει η εξυπηρέτηση της εντολή φόρτωσης από την κρυφή μνήμη δεδομένων και έτσι θεωρείται ότι έχει τελειώσει την εκτέλεσή της και η ίδια η εντολή. Οι εξαρτήσεις RAW που έχει η εντολή φόρτωσης με τις εντολές αποθήκευσης δεν έχουν εντοπιστεί διότι όταν έγινε ο έλεγχος, στον 17ο κύκλο, δεν ήταν έτοιμες οι διευθύνσεις προσπέλασης των εντολών stq. Παρόλο που η εντολή φόρτωσης έχει τελειώσει την εκτέλεσή της δε στέλνεται στο ROB για να ολοκληρωθεί διότι δεν έχει ολοκληρωθεί ακόμη εντολή αποθήκευσης από την οποία εξαρτάται χρονικά.

Στον 21ο κύκλο:

```
####################################################################
CPU cycle:                    21
---------

####################################################################
.
.
.
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
--------------------------------------------------------------------
000000000000001c
0000000000000020
```

```
0000000000000024
0000000000000028
000000000000002c
0000000000000030
0000000000000034
0000000000000038
.
.
.
ldq_write_replay: 0000000000000001
```

Ολοκληρώνονται οι εντολές με διευθύνσεις μνήμης από 0x1c έως 0x38. Μέσα σε αυτές συμπεριλαμβάνεται και η εντολή stq (διεύθυνση 0x1c). Στο στάδιο MEM λοιπόν εντοπίζεται για πρώτη φορά η εξάρτηση RAW μεταξύ της εντολής φόρτωσης και της συγκεκριμένης εντολής αποθήκευσης. Αυτό έχει σαν αποτέλεσμα να δοθεί εντολή εκκένωσης της διοχέτευσης όταν φτάσει η σειρά ολοκλήρωσης της ldq.

Στον 23^{ο} κύκλο:

```
##################################################################
CPU cycle:                    23
---------

##################################################################
.
.
.
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
------------------------------------------------------------------
000000000000003c
0000000000000040
0000000000000044
0000000000000048
000000000000004c
.
.
.
ldq_write_replay: 0000000000000001
```

Ολοκληρώνονται οι εντολές με διευθύνσεις μνήμης από 0x3c έως 0x4c. Μέσα σε αυτές συμπεριλαμβάνεται και η δεύτερη εντολή stq του προγράμματος (διεύθυνση 0x3c – από την οποία εξαρτάται ουσιαστικά η ldq). Στο στάδιο MEM λοιπόν εντοπίζεται για πρώτη φορά η εξάρτηση RAW μεταξύ της εντολής φόρτωσης και της συγκεκριμένης εντολής αποθήκευσης. Αυτό έχει σαν αποτέλεσμα να δοθεί εντολή εκκένωσης της διοχέτευσης όταν φτάσει η σειρά ολοκλήρωσης της ldq. Ο λόγος που πρέπει να συμβεί αυτό είναι επειδή στον 18^{ο} κύκλο που τελείωσε η εντολή φόρτωσης την εκτέλεσή της προώθησε στα υπόλοιπα στάδια της διοχέτευσης μη έγκυρα δεδομένα.

Με την ολοκλήρωση της δεύτερης εντολής αποθήκευσης, στέλνεται η εντολή ldq στο ROB για να ολοκληρωθεί, διότι η τρέχουσα stq είναι αυτή από την οποία εξαρτιόταν χρονικά. Όντως στον 27^{ο} κύκλο:

```
##################################################################
CPU cycle:                    27
---------

##################################################################
.
.
.
```

```
TOTAL FLUSHES:
--------------
total flushes:
00000000000000000000000000000000000000000000000000000000000000001
flushes due to direct mispredictions:
00000000000000000000000000000000000000000000000000000000000000000
flushes due to aliases:
00000000000000000000000000000000000000000000000000000000000000001
```

Είναι η σειρά ολοκλήρωσης της ldq. Όμως δεν ολοκληρώνεται και γίνεται εκκένωση της διοχέτευσης. Η προσκόμιση των εντολών συνεχίζει από τη διεύθυνση 0x5c και στον 32ο κύκλο η εντολή ldq εισέρχεται στο στάδιο DECODE.

```
######################################################################
CPU cycle:                  32
---------

######################################################################
.
.
.
INSTRUCTIONS THAT WILL BE FORWARDED FROM FETCH:
-----------------------------------------------
a6df0c08
00000000
00000000
00000000
```

Η ροή συνεχίζεται κανονικά και τελικά η ldq ολοκληρώνεται στον 43ο κύκλο και μαζί με αυτήν τελειώνει η εκτέλεση του προγράμματος.

```
######################################################################
CPU cycle:                  43
---------

######################################################################
.
.
.
THE ADDRESSES OF THE INSTRUCTIONS WHICH HAVE BEEN COMPLETED THIS
CYCLE:
-------------------------------------------------------------------
000000000000005c
```

Με την παρουσίαση της εκτέλεσης του συγκεκριμένου κώδικα αναδείχθηκε η χρησιμότητα ελέγχου των RAW εξαρτήσεων μεταξύ των εντολών φόρτωσης που έχουν τελειώσει την εκτέλεσή τους και των εντολών αποθήκευσης που έχουν ολοκληρωθεί. Αν δεν γινόταν αυτός ο έλεγχος θα υπήρχε μεγάλη πιθανότητα παραβιάσεων και εν τέλει, λανθασμένων εκτελέσεων των εντολών.

Συμπεράσματα

Η μελέτη της μεταπτυχιακής διατριβής παρέχει ένα ισχυρό υπόβαθρο σχετικά με τις υπερβαθμωτές αρχιτεκτονικές. Οι δύσκολες έννοιες παρουσιάζονται και επεξηγούνται μέσω της ανάλυσης ενός πραγματικού υπερβαθμωτού μοντέλου, κάνοντας έτσι τη μελέτη πιο κατανοητή και προσεγγίσιμη ακόμα και σε ερευνητές οι οποίοι δεν έχουν ασχοληθεί σε βάθος με τις υπερβαθμωτές σχεδιάσεις.

Επίσης, έγινε ίσως για πρώτη φορά απόπειρα «αποκρυπτογράφησης» του υπερβαθμωτού επεξεργαστή IVM. Το μοντέλο αυτό διατίθεται χωρίς κανένα κόστος από το Πανεπιστήμιο του Illinois, ωστόσο δεν υπάρχει ούτε στοιχειώδης αναφορά ώστε να βοηθήσει πολλούς που ασχολούνται μ' αυτό ερευνητικά να κατανοήσουν πιο εύκολα τη λειτουργία του.

Πέρα, όμως, από τα εμφανή οφέλη δόθηκε η ευκαιρία να μελετηθεί ο ετερομεταγλωττιστής GCC, η αρχιτεκτονική συνόλου εντολών Alpha καθώς επίσης και ο επεξεργαστής Alpha 21264, ο οποίος ανήκει στην εταιρεία DEC και χρησιμοποιούνταν παλιότερα σε υπολογιστές του εμπορίου. Για να γίνει εφικτή η προσομοίωση εκτέλεσης των προγραμμάτων στον IVM, έπρεπε ο επεξεργαστής να ληφθεί υπόψη ως μέλος ενός ευρύτερου συστήματος και να μην υπάρξει προσκόλληση μονάχα στα πολλαπλά κυκλώματά του. Έτσι, μελετήθηκαν ως ένα βαθμό αρκετά εργαλεία (όπως tcl, shell script) και διάφορες δυνατότητες του λειτουργικού συστήματος Linux που συνέβαλαν προς την κατεύθυνση αυτή.

Εντούτοις, πολλές είναι και οι προκλήσεις που προέκυψαν. Πρώτα απ' όλα, ο IVM-1.0 έχει αρκετές ατέλειες στη σχεδίασή του, όπως για παράδειγμα στο στάδιο μνήμης ή η μη χρησιμοποίηση του μηχανισμού πρόβλεψης διακλαδώσεων και η στατική αντιμετώπιση αυτού του είδους των εντολών ή η ανυπαρξία μονάδων αριθμητικής κινητής υποδιαστολής. Επίσης, πρέπει να βρεθεί ένας τρόπος για ν' αντιμετωπιστούν οι δυσκολίες εκτέλεσης των μετροπρογραμμάτων SPEC. Τα μετροπρογράμματα αυτά διαθέτουν πολλαπλές κλήσεις συστήματος που φυσικά δεν μπορούν να υποστηριχθούν από το μοντέλο. Μέχρι στιγμής οι λύσεις που έχουν προταθεί είναι άκομψες και χωρίς να παρέχουν ουσιαστική επίλυση του προβλήματος. Αυτό αποτελεί την κύρια προτεραιότητα για μελλοντική ενασχόληση.

Παρ' όλες τις δυσκολίες και τα προβλήματα η εκπόνηση της εργασίας ήταν πλήρης στο μέγιστο βαθμό. Για πρώτη φορά υπάρχει διαθέσιμη η τεκμηρίωση του μοντέλου IVM με την οποία ανοίγουν νέοι δρόμοι για τη χρήση του, τη βελτίωσή του, ακόμη και την επέκτασή του.

Βιβλιογραφία

[1] Modern Processor Design: Fundamentals of Superscalar Processors, John Paul Shen, Mikko H. Lipasti, McGraw-Hill Series in Electrical and Computer Engineering

[2] Computer Architecture: A Quantitative Approach, 4th Edition, John L. Henessy, David A. Patterson

[3] Οργάνωση και Σχεδίαση Υπολογιστών: Η Διασύνδεση Υλικού και Λογισμικού, D. A. Patterson, J. L. Hennessy (4η Έκδοση, Elsevier. Τίτλος Πρωτοτύπου: Computer Organization and Design: The Hardware/Software Interface). Επιστημονική επιμέλεια και μετάφραση: Δ. Γκιζόπουλος

[4] The Alpha 21264 Microprocessor Architecture, R. E. Kessler, E. J. McLellan, and D. A. Webb

[5] Alpha Assembly Language Guide, Randal E. Bryant, Carnegie Mellon University

[6] Memory dependence prediction using store sets, George Z. Chrysos, Joel S. Emer, Proceedings of the 25th annual international symposium on Computer architecture (ISCA)

[7] http://www.crhc.illinois.edu/ACS/tools/

[8] ReStore: Symptom Based Soft Error Detection in Microprocessors, Nicholas J. Wang and Sanjay J. Patel, Proceedings of the 2005 International Conference on Dependable Systems and Networks, Yokohama, Japan, June 2005

[9] Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline, Nicholas J. Wang, Justin Quek, Todd M. Rafacz, and Sanjay J. Patel, Proceedings of the 2004 International Conference on Dependable Systems and Networks, Florence, Italy, June 2004

[10] Alpha 21264 Microprocessor Hardware Reference Manual, Compaq

[11] Alpha Architecture Handbook, Compaq

[12] http://en.wikipedia.org/wiki/DEC_Alpha#Instruction_formats

[13] Μεταγλώττιση και εκτέλεση κώδικα γλώσσας C σε ενσωματωμένο επεξεργαστή αρχιτεκτονικής MIPS, Βασίλειος Δήμητσας

[14] First the Tick, Now the Tock: Intel® Microarchitecture (Nehalem) (white paper)

[15] http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15332,00.html

ΠΑΡΑΡΤΗΜΑ Α: Τα scripts που χρησιμοποιούνται για τη δημιουργία των αρχείων δεδομένων

Στο παράρτημα αυτό παρατίθενται τα scripts που χρησιμοποιήθηκαν για την παραγωγή των αρχείων δεδομένων, των κρυφών μνημών. Αρχικά δίνεται το linker script.

```
MEMORY
  {
    rom (rx) : ORIGIN = 0, LENGTH = 3K
    ram (!rx) : org = 3072, l = 24K
  }
SECTIONS
  {
         .text :{
           *(.text)
         }>rom
         .data :{
           *(.data)
             *(.sdata)
             *(.rdata)
             *(.rodata)
         }>ram
         .bss :{
             *(.bss)
             *(.sbss)
         }>ram
         .got :{
             *(.got)
         }>ram
  }
```

Ακολουθεί το tcl script, το οποίο ανακτήθηκε από το διαδίκτυο και τροποποιήθηκε έτσι ώστε να παράγεται το δυαδικό αρχείο στη μορφή που απαιτούσε το perl script:

```
#!/bin/tcl
proc string'reverse str {
    set res {}
    #set res1 {}
    set i [string length $str]
    while {$i > 0} {
      set j [incr i 0]
      append res [string range $str [incr j -2] [incr j +1]]
      set i [incr i -2]
      #append res1 [string index $str [incr i -1]]
      #append res [format $res1]
      #set res1 {}
    }
    set res
 } ;# RS


proc dumpFile { fileName { channel stdout } } {

     # Open the file, and set up to process it in binary mode.

     set f [open $fileName r]
```

```
fconfigure $f \
    -translation binary \
    -encoding binary \
    -buffering full -buffersize 16384

while { 1 } {

    # Record the seek address.  Read 16 bytes from the file.

    set addr [tell $f]
    set s [read $f 16]

    # Convert the data to hex and to characters.

    binary scan $s H*@0a* hex ascii

    # Replace non-printing characters in the data.

    #regsub -all -- {[^[:graph:] ]} $ascii {.} ascii

    # Split the 16 bytes into two 8-byte chunks

    set hex1   [string range $hex   0 7]
    set hex2   [string range $hex  8 15]
  set hex3   [string range $hex  16 23]
  set hex4   [string range $hex  24 31]
    #set ascii1 [string range $ascii 0  7]
    #set ascii2 [string range $ascii 8 16]

    # Convert the hex to pairs of hex digits

    regsub -all -- {..} $hex1 {&} hex1
    regsub -all -- {..} $hex2 {&} hex2
    regsub -all -- {..} $hex3 {&} hex3
    regsub -all -- {..} $hex4 {&} hex4

    # Put the hex and Latin-1 data to the channel

    #puts $channel [format {%08x  %-24s %-24s %-8s %-8s} \
                        #$addr $hex1 $hex2 $ascii1 $ascii2]

  #puts $channel [format {%-24s %-24s} \
                        #$hex1 $hex2]

 puts $channel [string'reverse $hex1]

 puts $channel [string'reverse $hex2]

 puts $channel [string'reverse $hex3]

 puts $channel [string'reverse $hex4]

    # Stop if we've reached end of file

    if { [string length $s] == 0 } {
        break
```

```
            }
        }

        # When we're done, close the file.

        close $f
        return
    }
    #--------------------------------------------------------------------
--
    #
    # Main program
    #
    #--------------------------------------------------------------------
--
    if { [info exists argv0] && [string equal $argv0 [info script]] } {
        foreach file $argv {
            #puts "$file:"
            dumpFile $file
        }
    }
```

Τέλος, δίνεται το perl script, το οποίο ανακτήθηκε από τη διαδικτυακή σελίδα του εργαστηρίου CHRC (Center for Reliable and High-Performance Computing), του Πανεπιστημίου του Illinois, Urbana-Champaign. Το script αυτό παρήγαγε τα αρχεία δεδομένων των σειρών, μόνο για την κρυφή μνήμη εντολών και τροποποιήθηκε έτσι ώστε να παράγει επιπρόσθετα τα αρχεία δεδομένων των σειρών, της κρυφής μνήμης δεδομένων.

```
#!/usr/bin/perl
sub bin2dec {
    return unpack("N", pack("B32", substr("0" x 32 . shift, -32)));
}


sub dec2bin {
    my $str = unpack("B32", pack("N", shift));
    $str =~ s/^0+(?=\d)//;   # otherwise you'll get leading zeros
    return $str;
}

open(FILE, $ARGV[0]);

$count = 0;
$count2 = 0;
$count3[14] = '00000000000000';
$countAddressLines = 0;

#creates vector files to be loaded into caches

#expects input as 1 instruction per line in ascii hex,
#outputs 1 cache line per line in ascii hex

open(FILE0_0, ">bank0_way0");
open(FILE0_1, ">bank0_way1");
open(FILE1_0, ">bank1_way0");
open(FILE1_1, ">bank1_way1");
```

```
open(DATAFILE0_0, ">dataBank0_way0");
open(DATAFILE0_1, ">dataBank0_way1");
open(DATAFILE1_0, ">dataBank1_way0");
open(DATAFILE1_1, ">dataBank1_way1");
open(DATAFILE2_0, ">dataBank2_way0");
open(DATAFILE2_1, ">dataBank2_way1");
open(DATAFILE3_0, ">dataBank3_way0");
open(DATAFILE3_1, ">dataBank3_way1");
open(DATAFILE4_0, ">dataBank4_way0");
open(DATAFILE4_1, ">dataBank4_way1");
open(DATAFILE5_0, ">dataBank5_way0");
open(DATAFILE5_1, ">dataBank5_way1");
open(DATAFILE6_0, ">dataBank6_way0");
open(DATAFILE6_1, ">dataBank6_way1");
open(DATAFILE7_0, ">dataBank7_way0");
open(DATAFILE7_1, ">dataBank7_way1");
while ( !eof(FILE) ) {
    $cacheline = "";
    $iword_count = 0;
    while ($iword_count < 8) {
        chop($_ = <FILE>);
      if ( !eof(FILE) && !($_ =~ m/\W/) ) {
            $cacheline = "$_$cacheline";
      } else {
            $cacheline = "00000000$cacheline";
      }
        $iword_count++;
    }
    if ( ($countAddressLines < 3072) && ($count < 256) ) {
      if ( ($count & 0x1) == 0 ) {

            if ( $count > 128 ) {

                        print FILE0_1 "$cacheline\n";

            } else {

                        print FILE0_0 "$cacheline\n";

            }

      } else {

            if ( $count > 128 ) {

                        print FILE1_1 "$cacheline\n";

            } else {

                        print FILE1_0 "$cacheline\n";

            }

      }

      $count = $count + 1;
```

```
}
elsif ( ($countAddressLines >= 3072) && ($count2 < 1024) ) {
  $count3 = dec2bin($count2);
  if ( ($count3[7..5] & 0x0) == 0x0 ) {
        if ($count2 > 512) {
              print DATAFILE0_1 "$cacheline\n";
        }
        else {
              print DATAFILE0_0 "$cacheline\n";
        }
  }
  elsif ( ($count3[7..5] & 0x1) == 0x1 ) {
        if ($count2 > 512 ) {
              print DATAFILE1_1 "$cacheline\n";
        }
        else {
              print DATAFILE1_0 "$cacheline\n";
        }
  }
  elsif ( ($count3[7..5] & 0x2) == 0x2 ) {
        if ($count2 > 512 ) {
              print DATAFILE2_1 "$cacheline\n";
        }
        else {
              print DATAFILE2_0 "$cacheline\n";
        }
  }
  elsif ( ($count3[7..5] & 0x3) == 0x3 ) {
        if ($count2 > 512 ) {
              print DATAFILE3_1 "$cacheline\n";
        }
        else {
              print DATAFILE3_0 "$cacheline\n";
        }
  }
  elsif ( ($count3[7..5] & 0x4) == 0x4 ) {
        if ($count2 > 512 ) {
              print DATAFILE4_1 "$cacheline\n";
        }
        else {
              print DATAFILE4_0 "$cacheline\n";
        }
  }
  elsif ( ($count3[7..5] & 0x5) == 0x5 ) {
        if ($count2 > 512 ) {
              print DATAFILE5_1 "$cacheline\n";
        }
        else {
              print DATAFILE5_0 "$cacheline\n";
        }
  }
  elsif ( ($count3[7..5] & 0x6) == 0x6 ) {
        if ($count2 > 512 ) {
              print DATAFILE6_1 "$cacheline\n";
        }
        else {
```

```
                print DATAFILE6_0 "$cacheline\n";
            }
        }
        else {
            if ($count2 > 512 ) {
                print DATAFILE7_1 "$cacheline\n";
            }
            else {
                print DATAFILE7_0 "$cacheline\n";
            }
        }

        $count2 = $count2 + 1;
    }

    $countAddressLines = $countAddressLines + 32;
}
```

ΠΑΡΑΡΤΗΜΑ Β: Η συμβολική γλώσσα Alpha

Η αρχιτεκτονική Alpha δημιουργήθηκε από τη Digital Equipment Corporation (DEC) σαν μία αρχιτεκτονική RISC δεύτερης γενιάς. Περιλαμβάνει ένα επαρκή αριθμό εντολών που καλύπτουν κοινές λειτουργίες ενώ παράλληλα αποφεύγει πολλά χαρακτηριστικά που θα μπορούσαν να μειώσουν την ταχύτητα του επεξεργαστή. Στις επόμενες σελίδες ακολουθεί η συνοπτική παρουσίαση και περιγραφή της αρχιτεκτονικής συνόλου εντολών του Alpha. Θα πρέπει να σημειωθεί ότι για την επεξήγηση της λειτουργίας που κάνουν οι εντολές χρησιμοποιείται της γλώσσας προγραμματισμού C.

Το αξιοσημείωτο χαρακτηριστικό του Alpha είναι ότι είναι μία μηχανή των 64 bits. Όλοι οι ακέραιοι καταχωρητές έχουν μέγεθος 64 bits και μπορεί να διαχειριστεί διευθύνσεις και ακεραίους του μεγέθους αυτού. Επίσης, υποστηρίζεται και η διαχείριση ακεραίων μεγέθους 32 bits. Παρακάτω δίνεται ένας πίνακας με τους τύπους δεδομένων που υποστηρίζει ο Alpha και το μέγεθος που εκφράζουν.

Τύπος δεδομένων στην αρχιτεκτονική Alpha	Μέγεθος (σε bytes)
byte	1
word	2
long word	4
quad word	8
S_floating (για αριθμούς κινητής υποδιαστολής)	4
T_ floating (για αριθμούς κινητής υποδιαστολής)	8

Πίνακας Β1: Οι τύποι δεδομένων που υποστηρίζονται από την αρχιτεκτονική Alpha

Η αρχιτεκτονική συνόλου του επεξεργαστή Alpha είναι σχετικά απλή. Οι αριθμητικές λειτουργίες εφαρμόζονται μόνο σε δεδομένα καταχωρητών. Για τη μεταφορά δεδομένων μεταξύ των καταχωρητών του επεξεργαστή και της μνήμης απαιτούνται οι λειτουργίες φόρτωσης/αποθήκευσης από/στη μνήμη και οι διακλαδώσεις υπό συνθήκη ελέγχουν τη σχέση μεταξύ ενός καταχωρητή και του 0.

Αριθμητικές λειτουργίες

Η αρχιτεκτονική Alpha υποστηρίζει λειτουργίες για ακεραίους μεγέθους 4 και 8 bytes. Ο πίνακας Β2 παραθέτει το σύνολο των αριθμητικών εντολών. Οι εντολές με κατάληξη "l" αναφέρονται σε ακεραίους των 4 bytes ενώ οι εντολές με κατάληξη "q" αναφέρονται σε ακεραίους των 8 bytes.

Οι αριθμητικές εντολές μπορούν να έχουν τρεις τελεστέους. Δύο τελεστέοι πηγής και ένας τελεστέος προορισμού. Για παράδειγμα η μορφή μιας εντολής πρόσθεσης είναι η εξής:

```
addq Ra, Rb, Rc
addq Ra, Litb, Rc
```

Τα Ra, Rb, Rc αποτελούν τους δύο καταχωρητές πηγές και τον καταχωρητή προορισμού αντίστοιχα. Το Litb αντιστοιχεί σε μία σταθερά που παίρνει τιμές από 0 έως 255. Όπως φαίνεται από τα δύο παραδείγματα, ο πρώτος τελεστέος προέρχεται από τιμή καταχωρητή, ο δεύτερος τελεστέος μπορεί να προέρχεται από τιμή καταχωρητή ή να είναι μία σταθερά και ο τρίτος τελεστέος αντιστοιχεί στον προορισμό που αποθηκεύεται το αποτέλεσμα, ο οποίος είναι καταχωρητής.

Long Word	Quad Word	Περιγραφή	Πράξη
addl	addq	Πρόσθεση	c = a + b
s4addl	s4addq	Βαθμωτή πρόσθεση	c = 4*a + b
s8addl	s8addq	Βαθμωτή πρόσθεση	c = 8*a + b
subl	subq	Αφαίρεση	c = a – b
s4subl	s4subl	Βαθμωτή αφαίρεση	c = 4*a – b
s8subl	s8subq	Βαθμωτή αφαίρεση	c = 8*a – b
mull	mulq	Πολλαπλασιασμός	c = a * b
divl	divq	Διαίρεση	c = a / b
reml	remq	Υπόλοιπο	c = a % b

Πίνακας Β2: Οι αριθμητικές εντολές της αρχιτεκτονικής συνόλου εντολών του Alpha

Λειτουργίες σύγκρισης

Στην αρχιτεκτονική συνόλου εντολών Alpha όλες οι συγκρίσεις εφαρμόζονται σε quad words. Οι εντολές σύγκρισης έχουν την ίδια μορφή με τις αριθμητικές εντολές. Το αποτέλεσμα το γράφουν στον καταχωρητή Rc και μπορεί να είναι 1 (αν η σύγκριση βγει αληθής) ή 0 (αν η σύγκριση είναι ψευδής).

Εντολή	Περιγραφή	Υπολογισμός
cmpeq	Ισότητα	c = (a == b)
cmple	Μικρότερο ή ίσο	c = (a <= b)
cmplt	Μικρότερο	c = (a < b)
cmpule	Μικρότερο ή ίσο με μη προσημασμένους αριθμούς	c = (ua <= ub)
cmpult	Μικρότερο με μη προσημασμένους αριθμούς	c = (ua < ub)

Πίνακας Β3: Οι εντολές σύγκρισης

Λογικές λειτουργίες και λειτουργίες σε επίπεδο bit

Όλες οι λογικές λειτουργίες και οι λειτουργίες σε επίπεδο bit εφαρμόζονται σε quad words. Έχουν την ίδια μορφή όπως και οι αριθμητικές εντολές. Η εντολή της αριστερής ολίσθησης εισάγει μηδενικά στις δεξιότερες θέσεις ενώ η εντολή της δεξιάς ολίσθησης εισάγει μηδενικά στις αριστερότερες θέσεις. Ο αριθμός των θέσεων που μπορεί να ολισθήσουν τα bits ενός αριθμού μπορεί να είναι από 0 έως 63. Αν είναι παραπάνω τότε ο αριθμός αυτός μειώνεται κατά modulo 64.

Εντολή	Περιγραφή	Πράξη
and	And	c = a & b
bic	Καθαρισμός bit	c = a & ~b
bis	Θέση bit	c = a | b
eqv	Λογική ισότητα	c = ~ (a ^ b)
xor	XOR	c = a ^ b
ornot	OR – NOT	c = a | ~ b
sra	Δεξιά αριθμητική ολίσθηση	c = a >> (b % 64)
sll	Αριστερή ολίσθηση	c = a << (b % 64)
srl	Δεξιά λογική ολίσθηση	c = ua >> (b % 64)

Πίνακας Β4: Οι λογικές εντολές

Εντολές φόρτωσης και αποθήκευσης

Οι εντολές φόρτωσης και αποθήκευσης μεταφέρουν δεδομένα μεταξύ των καταχωρητών του επεξεργαστή και της μνήμης. Χρησιμοποιούνται ξεχωριστές εντολές για δεδομένα των 4 και των 8 bytes. Επιπρόσθετα, οι εντολές lda και ldah έχουν την ίδια μορφή με μία εντολή φόρτωσης αλλά στην ουσία είναι πράξεις πρόσθεσης, δεδομένων καταχωρητών με σταθερές. Οι εντολές αυτές προτιμούνται όταν χρησιμοποιούνται σταθερές οι οποίες έχουν μέγεθος πάνω από οκτώ bits.

Οι εντολές φόρτωσης και αποθήκευσης έχουν την ακόλουθη μορφή (παρατίθεται η εντολή ldq αλλά ισχύει για όλες το ίδιο):

```
ldq Ra, Disp(Rb)
```

Οι τελεστέοι Ra, Rb είναι καταχωρητές και το Disp αποτελεί τη σταθερά μετατόπισης, η οποία μπορεί να πάρει τιμές από -32768 έως 32767. Για μία εντολή φόρτωσης, ο καταχωρητής Ra είναι αυτός στον οποίο θα αποθηκευτεί το αποτέλεσμα από την προσκόμιση των δεδομένων. Για μία εντολή αποθήκευσης, ο καταχωρητής Ra περιέχει τα δεδομένα που θα αποθηκευτούν στη μνήμη.

Οι εντολές φόρτωσης και αποθήκευσης πρέπει να προσπελαύνουν δεδομένα σε ευθυγραμμισμένες διευθύνσεις. Για παράδειγμα όταν πρέπει να προσκομιστεί από τη μνήμη ένα δεδομένο μεγέθους τεσσάρων bytes, τότε η διεύθυνση προσκόμισης πρέπει να είναι πολλαπλάσια του 4. Το αντίστοιχο ισχύει και για δεδομένα μεγέθους 8 bytes.

Εντολή	Περιγραφή	Προσπέλαση bytes	Διεύθυνση προσπέλασης (Effective Address – EA)	Αποτέλεσμα
ldl	Load long	4	EA = b + D	a = *EA
ldq	Load quad	8	EA = b + D	a = *EA
stl	Store long	4	EA = b + D	*EA = a
stq	Store quad	8	EA = b + D	*EA = a
lda	Load address	0	EA = b + D	a = EA
ldah	Load address high	0	EA = b + D*65536	a = EA
ldq_u	Load quad unaligned	8	EA = (b + D) & ~0x7	a = *EA
stq_u	Store quad unaligned	8	EA = (b + D) & ~0x7	*EA = a

Πίνακας Β5: Οι εντολές φόρτωσης και αποθήκευσης

Εξαίρεση αποτελούν οι εντολές ldq_u και stq_u. Οι εντολές χρησιμοποιούνται τυπικά όταν γίνονται λειτουργίες σε επίπεδο byte. Δηλαδή, όταν πρέπει για παράδειγμα να φορτωθεί ένας χαρακτήρας μεγέθους ενός byte. Αν αυτός ο χαρακτήρας είναι μέσα σε μία συμβολοσειρά που έχει μέγεθος 8 bytes, τότε με την εντολή ldq_u φορτώνεται ολόκληρη η συμβολοσειρά διότι μηδενίζονται τα τρία τελευταία bits της διεύθυνσης.

Λειτουργίες μεταφοράς υπό συνθήκη (Conditional Moves)

Οι λειτουργίες αυτές που υποστηρίζονται από την αρχιτεκτονική συνόλου εντολών Alpha παρέχουν τη δυνατότητα ανανέωσης του περιεχομένου ενός καταχωρητή όταν ισχύει μια συνθήκη χωρίς να χρησιμοποιούνται λειτουργίες διακλάδωσης.

Οι εντολές αυτές έχουν την μορφή όπως και οι αριθμητικές λειτουργίες. Για παράδειγμα η εντολή cmoveq συντάσσεται ως εξής:

```
cmoveq Ra, Rb, Rc
cmoveq Ra, Litb, Rc
```

Ο καταχωρητής Ra περιέχει το περιεχόμενο που εξετάζεται, ο καταχωρητής Rb ή η σταθερά Litb αποτελούν τον τελεστέο πηγή και ο καταχωρητής Rc είναι ο καταχωρητής προορισμού στον οποίο θα αποθηκευτεί ο τελεστέος πηγή, αν ισχύει η συνθήκη. Αναπαριστάνοντας σε γλώσσα C τη λειτουργία της εντολής cmoveq, θα ήταν:

```
if (a == 0)
      c = b;
```

Εντολή	Περιγραφή	Συνθήκη μεταφοράς
cmoveq	Conditional move on equal	a == 0
cmovne	Conditional move on not equal	a != 0
cmovgt	Conditional move on greater than	a > 0
cmovge	Conditional move on greater than or equal	a >= 0
cmovlt	Conditional move on less than	a < 0
cmovle	Conditional move on less than or equal	a <= 0

cmovlbc	Conditional move on lower bit clear	!(a & 0x1)
cmovlbs	Conditional move on lower bit set	a & 0x1

Πίνακας Β6: Εντολές μεταφοράς υπό συνθήκη

Λειτουργίες χειρισμού των bytes

Η αρχιτεκτονική Alpha δεν υποστηρίζει απευθείας λειτουργίες σε επίπεδο byte, όπως τη μεταφορά μεμονωμένων bytes μεταξύ των καταχωρητών και της μνήμης. Για το λόγο αυτό έχουν προστεθεί κάποιες εντολές, οι οποίες διαχειρίζονται bytes. Ο πίνακας Β7 περιέχει τέτοιες εντολές και η μορφή τους είναι ίδια με αυτή των αριθμητικών εντολών. Οι τελεστέοι a, b είναι οι πηγές και αντιστοιχούν στους καταχωρητές Ra, Rb (ο τελεστέος b μπορεί να αντιστοιχεί και στη σταθερά Litb). Το αποτέλεσμα αποθηκεύεται στον τελεστέο c, ο οποίος αντιστοιχεί στον καταχωρητή Rc.

Εντολή	Περιγραφή	Byte(c, i) == 0	Nonzero Byte(c, i)
extbl	Extract byte low	i != 0	Byte(a, bl)
mskbl	Mask byte low	i == bl	Byte(a, i)
insbl	Insert byte low	i != bl	Byte(a, 0)
extqh	Extract quad word high	bl != 0 && i+bl<8	Byte(a, (i+bl-8)&0x7)
zap	Zero bytes	Bit(b, i)	Byte(a, i)
zapnot	Zero bytes not	!Bit(b, i)	Byte(a, i)

Πίνακας Β7: Εντολές διαχείρισης bytes. Η τιμή του δείκτη bl είναι ίση με b & 0x7

Η τρίτη στήλη δηλώνει την κατάσταση κάτω από την οποία το byte i του αποτελέσματος θα τεθεί στο μηδέν. Η τέταρτη στήλη δηλώνει ποιο byte του τελεστέου πηγής θα είναι το byte i στον καταχωρητή προορισμού, εφόσον βέβαια το byte i επιτρέπεται να είναι μη μηδενικό. Επίσης, χρησιμοποιείται η σημειογραφία Byte(x, j) όταν γίνεται αναφορά στο byte j του quad word x. Επίσης, θεωρείται ότι ο τρόπος αποθήκευσης των δεδομένων είναι μικρού άκρου, δηλαδή τα bytes αριθμούνται από το 0 (το λιγότερο σημαντικό) έως το 7 (το σημαντικότερο).

Οι δύο τελευταίες εντολές στον πίνακα Β7 επιτρέπουν το μηδενισμό των bytes του τελεστέου a χρησιμοποιώντας μία μάσκα, η οποία προκύπτει από τα οκτώ χαμηλότερης τάξης bits του τελεστέου b. Κάθε bit της μάσκας δηλώνει αν θα πρέπει να αντιγραφεί το αντίστοιχο byte από τον τελεστέο a στον καταχωρητή προορισμού ή να τεθεί το byte εκείνο, του καταχωρητή προορισμού, στο μηδέν. Οι δύο εντολές, zap και zapnot, έχουν διαφορά στην ερμηνεία των bits της μάσκας.

Ειδικές εντολές

Για να είναι η συμβολική γλώσσα Alpha πιο προσιτή στον προγραμματιστή και πιο εύκολη στην ανάγνωση προγραμμάτων, χρησιμοποιούνται κάποιες ειδικές ψευδοεντολές, οι οποίες προκύπτουν στην ουσία από τις υπάρχουσες εντολές της αρχιτεκτονικής συνόλου εντολών Alpha.

Ψευδοεντολή	Περιγραφή	Πραγματική εντολή
nop	No operation	`bis $31, $31, $31`
-	Εναλλακτικό no operation	`ldq_u $31, 0($sp)`
mov $1, $2	Μεταφορά καταχωρητή	`bis $31, $1, $2`
mov 17, $2	Μεταφορά σταθεράς	`bis $31, 17, $2`
sextl $1, $2	Μεταφορά long word και επέκταση προσήμου	`addl $31, $1, $2`

Πίνακας Β8: Οι ψευδοεντολές της συμβολικής γλώσσας Alpha

Εντολές μεταφοράς ελέγχου

Υπάρχουν δύο τύποι εντολών μεταφοράς ελέγχου: Διακλαδώσεις και άλματα. Οι περισσότερες διακλαδώσεις είναι υπό συνθήκη. Αν θα ληφθούν ή όχι εξαρτάται από τη σύγκριση ενός καταχωρητή τελεστέου με το 0. Η μορφή τους είναι η ακόλουθη (σαν παράδειγμα δίνεται η εντολή beq αλλά το ίδιο ισχύει και για τις άλλες):

```
beq Ra, Label
```

Το Ra δηλώνει ο καταχωρητής τελεστέος που ελέγχεται και το Label δηλώνει μία διεύθυνση μνήμης σχετική ως προς το μετρητή προγράμματος. Στον πίνακα B9 παρατίθενται όλες οι εντολές διακλάδωσης και χρησιμοποιείται η σύνταξη της C για να αναπαρασταθεί η συνθήκη. Όπου a αντιστοιχεί στον καταχωρητή Ra.

Εντολή	Περιγραφή	Συνθήκη διακλάδωσης
beq	Branch on equal	a == 0
bne	Branch on not equal	a != 0
bgt	Branch on greater than	a > 0
bge	Branch on greater than or equal	a >= 0
blt	Branch on less than	a < 0
ble	Branch on less than or equal	a <= 0
blbc	Branch on lower bit clear	!(a & 0x1)
blbs	Branch on lower bit set	a & 0x1
br	Branch	1
bsr	Branch to subroutine	1

Πίνακας B9: Εντολές διακλάδωσης

Οι εντολές br και bsr διακλαδώνουν χωρίς συνθήκη. Ο καταχωρητής Ra της εντολής br είναι ο $31. Στην εντολή bsr ο καταχωρητής Ra χρησιμοποιείται με έναν τελείως διαφορετικό τρόπο. Είναι ο $26 και περιέχει την παρούσα τιμή του μετρητή προγράμματος έτσι ώστε να γίνει σωστά η επιστροφή από την υπορουτίνα.

Οι εντολές άλματος παρέχουν μεταφορές ελέγχου χωρίς συνθήκη και η διεύθυνση προορισμού καθορίζεται από έναν καταχωρητή. Υπάρχουν τρία διαφορετικά είδη:

```
jmp $31,(Rb), Hint
jsr Ra, (Rb), Hint
ret $31, (Rb), Hint
```

Το Hint είναι κάποια προαιρετική πληροφορία που δίνεται από το μεταγλωττιστή για να βοηθήσει τον επεξεργαστή να προβλέψει την διεύθυνση προορισμού. Η εντολή jmp περιέχει τη διεύθυνση προορισμού στον καταχωρητή προορισμού Rb. Η εντολή jsr, η οποία είναι άλμα σε υπορουτίνα, περιέχει τη διεύθυνση προορισμού στον καταχωρητή Rb και η παρούσα τιμή του μετρητή προγράμματος είναι αποθηκευμένη στον καταχωρητή Ra, που συνήθως είναι ο καταχωρητής $26. Η ret είναι εντολή επιστροφής από ρουτίνα και η διεύθυνση επιστροφής περιλαμβάνεται στον καταχωρητή Rb, που συνήθως είναι ο $26 (στον καταχωρητή αυτόν έχουν αποθηκεύσει την παρούσα τιμή του μετρητή προγράμματος προηγούμενες εντολές bsr ή jsr).

Εντολές κινητής υποδιαστολής

Οι εντολές κινητής υποδιαστολής χρησιμοποιούν ένα σύνολο 32 καταχωρητών κινητής υποδιαστολής, με ονόματα $f0-$f31. Υποστηρίζονται τέσσερις τύποι, αλλά δύο είναι οι πιο δημοφιλείς: Ο τύπος S_floating, που είναι IEEE απλής ακρίβειας και ο T_floating είναι IEEE διπλής ακρίβειας. Κάθε καταχωρητής κινητής υποδιαστολής έχει μέγεθος 8 bytes και μπορεί να έχει αποθηκευμένη μια τιμή μονής ή διπλής ακρίβειας.

S_floating	T_floating	Περιγραφή	Υπολογισμός
adds	addt	Πρόσθεση	c = a + b
subs	subt	Αφαίρεση	c = a – b
muls	mult	Πολλαπλασιασμός	c = a * b
divs	divt	Διαίρεση	c = a / b

Πίνακας B10: Λειτουργίες αριθμών κινητής υποδιαστολής: Κάθε εντολή έχει έκδοση για μονή (S_floating) και διπλή (T_floating) ακρίβεια

Γενικά, οι εντολές κινητής υποδιαστολής έχουν την ίδια συμπεριφορά και τύπο με τις αντίστοιχες εντολές ακεραίων αριθμών. Για παράδειγμα, η εντολή πρόσθεσης addt:

```
addt Fa, Fb, Fc
```

όπου Fa και Fb αποτελούν τους καταχωρητές πηγές και ο Fc τον καταχωρητή προορισμού. Στον πίνακα B10, καθώς και στους υπόλοιπους που ακολουθούν, χρησιμοποιείται η σημειογραφία της C και οι a, b και c αντιστοιχούν στους Fa, Fb και Fc.

Ο πίνακας B11 παρουσιάζει κάποιες από τις εντολές σύγκρισης αριθμών κινητής υποδιαστολής. Αυτές θέτουν τον καταχωρητή Fc στο 2.0 αν η συνθήκη είναι αληθής και στο 0.0 όταν η συνθήκη είναι ψευδής.

Εντολή	Περιγραφή	Υπολογισμός
cmpteq	Ισότητα	c = (a == b) ? 2.0 : 0.0
cmptle	Μικρότερο από ή ίσο	c = (a <= b) ? 2.0 : 0.0
cmptlt	Μικρότερο από	c = (a < b) ? 2.0 : 0.0

Πίνακας B11: Οι εντολές σύγκρισης για αριθμούς κινητής υποδιαστολής

Οι εντολές φόρτωσης και αποθήκευσης κινητής υποδιαστολής έχουν την ίδια μορφή όπως οι αντίστοιχες εντολές για ακεραίους. Για παράδειγμα η μορφή της εντολής lds είναι:

```
lds Fa, Disp(Rb)
```

Οι τελεστέοι Fa και Rb υποδεικνύουν καταχωρητές και το Disp υποδεικνύει μία σταθερή μετατόπιση, η οποία έχει εύρος τιμών από -32,768 έως 32,767. Ο καταχωρητής κινητής υποδιαστολής Fa αποδεικνύει τον καταχωρητή προορισμού (για εντολή φόρτωσης) ή τον καταχωρητή πηγής (για εντολή αποθήκευσης) ενώ ο συνδυασμός των Rb και Disp σχηματίζουν τη διεύθυνση προσπέλασης. Οι εντολές φόρτωσης και αποθήκευσης κινητής υποδιαστολής πρέπει να προσπελαύνουν δεδομένα σε ευθυγραμμισμένες διευθύνσεις.

Εντολή	Περιγραφή	Αριθμός των bytes που προσπελαύνονται	Διεύθυνση προσπέλασης	Αποτέλεσμα
lds	Φόρτωση S_floating	4	EA = b + D	a = *EA
ldt	Φόρτωση T_floating	8	EA = b + D	a = *EA
sts	Αποθήκευση S_floating	4	EA = b + D	*EA = a
stt	Αποθήκευση T_floating	8	EA = b + D	*EA = a

Πίνακας B12: Εντολές φόρτωσης και αποθήκευσης κινητής υποδιαστολής

Ο πίνακας B13, περιέχει εντολές μεταφοράς υπό συνθήκη για αριθμούς κινητής υποδιαστολής. Αυτές έχουν την ίδια μορφή με τις αριθμητικές εντολές. Για παράδειγμα, η εντολή fcmomveq:

```
fcmoveq Fa, Fb, Fc
```

Εντολή	Περιγραφή	Συνθήκη μεταφοράς
fcmoveq	Conditional move on equal	a == 0.0
fcmovne	Conditional move on not equal	a != 0.0
fcmovgt	Conditional move on greater than	a > 0.0
fcmovge	Conditional move on greater than or equal	a >= 0.0
fcmovlt	Conditional move on less than	a < 0.0
fcmovle	Conditional move on less than or equal	a <= 0.0

Πίνακας B13: Εντολές μεταφοράς υπό συνθήκη

Στον επόμενο πίνακα παρατίθενται οι εντολές διακλάδωσης υπό συνθήκη για κινητή υποδιαστολή. Έχουν μορφή παρόμοια με τις εντολές διακλάδωσης ακεραίων. Για παράδειγμα η εντολή fbeq:

```
fbeq Fa, Label
```

Η απόφαση για το αν λαμβάνεται η διακλάδωση ή όχι εξαρτάται από το αποτέλεσμα της σύγκρισης του καταχωρητή fa με το 0.0.

Εντολή	Περιγραφή	Συνθήκη μεταφοράς
fcmoveq	Branch on equal	a == 0.0
fcmovne	Branch on not equal	a != 0.0
fcmovgt	Branch on greater than	a > 0.0
fcmovge	Branch on greater than or equal	a >= 0.0
fcmovlt	Branch on less than	a < 0.0
fcmovlt	Branch on less than or equal	a <= 0.0
fcmovle		

Πίνακας B14: Εντολές διακλάδωσης κινητής υποδιαστολής

Τέλος, η αρχιτεκτονική συνόλου εντολών Alpha υποστηρίζει λειτουργίες για μετατροπή μεταξύ διαφορετικών αριθμητικών τύπων. Κάθε μία από τις εντολές έχουν την ίδια μορφή. Για παράδειγμα η εντολή cvtqs:

```
cvts Fb, Fc
```

Εντολή	Μετατροπή από	Μετατροπή σε
cvtqs	Quad integer	S_floating
cvtqt	Quad integer	T_floating
cvtsq	S_floating	Quad integer
cvttq	T_floating	Quad integer
cvtts	T_floating	S_floating
cvtst	S_floating	T_floating

Πίνακας B15: Εντολές μετατροπής κινητής υποδιαστολής

Ο αναγνώστης θα πρέπει να γνωρίζει πως ο επεξεργαστής IVM δεν υποστηρίζει εκτέλεση εντολών κινητής υποδιαστολής. Παρόλα αυτά, παρατίθενται στο συγκεκριμένο παράρτημα για να είναι πιο πλήρες και αναλυτικό.

Στο τελευταίο κομμάτι του παραρτήματος δίνονται οι πίνακες με όλους τους αρχιτεκτονικούς, ακέραιους και κινητής υποδιαστολής καταχωρητές που χρησιμοποιούνται για την αποθήκευση διαφόρων τιμών και αποτελεσμάτων καθώς επίσης και για την υποστήριξη άλλων λειτουργιών όπως κλήση διαδικασιών, διατήρηση της στοίβας μιας διεργασίας, τη δέσμευση χώρου για δομές δεδομένων και άλλες. Στους πίνακες αυτούς, εκτός του ότι περιέχονται τα ονόματα των καταχωρητών, εξηγείται και η χρήση για την οποία προορίζονται σύμφωνα με κάποιες κοινές προγραμματιστικές συμβάσεις.

Όνομα καταχωρητή	Όνομα στο λογισμικό	Χρήση
$0	v0	Η επιστρεφόμενη τιμή από ακέραιες συναρτήσεις
$1-$8	t0 - t7	Προσωρινοί
$9-$14	s0 – s5	Αποθηκευμένοι καταχωρητές
$15	s6	Αποθηκευμένος καταχωρητής
$15 ή $fp	fp	Δείκτης πλαισίου
$16-$21	a0 – a5	Ορίσματα ακεραίων
$22-$25	t8 – t11	Προσωρινοί
$26	ra	Διεύθυνση επιστροφής
$27	pv	Η διεύθυνση της παρούσας διαδικασίας
$27	t12	Προσωρινός
$28 ή $at	AT	Δεσμευμένος για χρήση από το συμβολομεταφραστή
$29 ή $gp	gp	Καθολικός δείκτης
$30 ή $sp	sp	Δείκτης στοίβας
$31	zero	Έχει πάντα τιμή 0

Πίνακας B16: Οι προγραμματιστικές συμβάσεις για τους ακέραιους αρχιτεκτονικούς καταχωρητές

Όνομα καταχωρητή	Χρήση
$f0	Επιστρεφόμενη τιμή από συναρτήσεις κινητής υποδιαστολής
$f1	Επιστρεφόμενη φανταστική τιμή από πολύπλοκες συναρτήσεις
$f2-$f9	Αποθηκευμένοι καταχωρητές
$f10-$f15	Προσωρινοί
$f16-$f21	Ορίσματα κινητής υποδιαστολής
$f22-$f30	Προσωρινοί
$f31	Έχει πάντα τιμή 0.0

Πίνακας Β17: Οι προγραμματιστικές συμβάσεις για τους αρχιτεκτονικούς καταχωρητές κινητής υποδιαστολής