ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Μοντελοποίηση Υπερβαθμωτού Επεξεργαστή σε Γλώσσα Περιγραφής Υλικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Γ. Πηλίτσος

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2006

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Μοντελοποίηση Υπερβαθμωτού Επεξεργαστή σε Γλώσσα Περιγραφής Υλικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Γ. Πηλίτσος

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή.

Αθήνα, Οκτώβριος 2006

.....	.....	.....
Κιαμάλ Πεκμεστζή	Γεώργιος Οικονομάκος	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

.....
Δημήτριος Γ. Πηλίτσος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

στην οικογένειά μου και σε όσους με στήριξαν

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κύριο Κιαμάλ Πεκμεστζή, Καθηγητή Ε.Μ.Π, και τον υποψήφιο διδάκτορα Ισίδωρο Σιδερή για τη συμβολή τους στην διεκπεραίωση αυτής της εργασίας.

Περίληψη

Τις τελευταίες δεκαετίες έχει σημειωθεί τεράστια πρόοδος στον τομέα των μικροεπεξεργαστών. Οι μικροεπεξεργαστές είναι υπεύθυνοι για τη δημιουργία μερικών από τις μεγαλύτερες καινοτομίες στα συστήματα υπολογιστών και χρησιμοποιούνται σε πολλές γνωστές και σημαντικές εφαρμογές. Οι έρευνες που γίνονται επικεντρώνονται κυρίως στις βελτιστοποιήσεις που μπορούν να γίνουν στην μικροαρχιτεκτονική σχεδίαση και στις τεχνικές που πρέπει να χρησιμοποιηθούν. Αυτά είναι και τα θέματα στα οποία επικεντρώνεται αυτή η διπλωματική.

Στόχος της διπλωματικής είναι η σχεδίαση και μοντελοποίηση ενός υπερβαθμωτού επεξεργαστή με τη χρήση γλώσσας περιγραφής υλικού. Στο πρώτο κεφάλαιο αναφέρονται κάποια ιστορικά στοιχεία για τους μικροεπεξεργαστές και γίνεται μία δικαιολόγηση των πλεονεκτημάτων των υπερβαθμωτών επεξεργαστών έναντι των βαθμωτών. Στο επόμενο κεφάλαιο μελετούνται με λεπτομέρεια οι βασικές τεχνικές σχεδίασης των υπερβαθμωτών επεξεργαστών. Στο τρίτο και στο τέταρτο κεφάλαιο παρουσιάζεται ο επεξεργαστής MICROPROC, που είναι ένας υπερβαθμωτός RISC μικροεπεξεργαστής, με ιδιαίτερη έμφαση στην μικροαρχιτεκτονική του. Στο πέμπτο κεφάλαιο γίνεται μία σύντομη αναφορά στη μεθοδολογία σχεδίασης που χρησιμοποιείται την τελευταία δεκαετία στην σχεδίαση μικροεπεξεργαστών. Τέλος υπάρχουν και αποτελέσματα από την προσομοίωση του επεξεργαστή MICROPROC. Η γλώσσα περιγραφής υλικού που χρησιμοποιήθηκε ήταν η VHDL και υπάρχει ξεχωριστό παράρτημα που περιέχει την υλοποίηση του επεξεργαστή σε VHDL.

Abstract

In the last few decades the microprocessor has undergone phenomenal advances. The microprocessor is responsible for facilitating some of the major innovations in computer systems and is used in many popular and important applications. Recent researches are focused mostly on the optimizations regarding the microarchitecture design and the best possible techniques that need to be used. This diploma thesis focuses exactly on these issues.

The purpose of this diploma thesis is the design of a superscalar processor using a hardware description language. In the first chapter we mention some historical clues about the microprocessor and we analyse the advantages of the superscalar over the scalar microprocessors. In the next chapter we study briefly the superscalar processor design techniques. In both the third and the fourth chapter we present the MICROPROC processor, which is a superscalar RISC microprocessor, with focus on the microarchitecture. In the fifth chapter we mention shortly the design methods that are usually used in the last decade in microarchitecture designing. Finally we have some results from the simulation of the MICROPROC processor. The hardware description language that we used is VHDL and there is an individual appendix that contains the implemenation of the MICROPROC processor in VHDL.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Πίνακας Σχημάτων

1 ΕΙΣΑΓΩΓΗ

Στην σχετικά σύντομη διάρκεια ζωής του, που δεν υπερβαίνει κατά πολύ τα 30 χρόνια, ο μικροεπεξεργαστής (microprocessor) έχει κάνει τεράστιες προόδους. Η απόδοσή του έχει βελτιωθεί αισθητά αφού διπλασιάζεται κάθε 18 μήνες. Στις τελευταίες τρεις δεκαετίες οι μικροεπεξεργαστές είναι υπεύθυνοι για την έμπνευση και τη δημιουργία μερικών από τις μεγαλύτερες καινοτομίες στα συστήματα υπολογιστών. Αυτές οι καινοτομίες περιλαμβάνουν τους embedded μικροελεγκτές, τους προσωπικούς υπολογιστές, τους σύγχρονους σταθμούς εργασίας, συσκευές χειρός και κινητές συσκευές (όπως τους επεξεργαστές των κινητών τηλεφώνων), servers (εξυπηρετητές) εφαρμογών και αρχείων, web servers για το Internet, υπερυπολογιστές χαμηλού κόστους και ευρείας κλίμακας δίκτυα υπολογιστών. Σήμερα πωλούνται κάθε χρόνου πάνω από 100 εκατομμύρια μικροεπεξεργαστές στις αγορές κινητής τηλεφωνίας, επεξεργαστών για εργασίες γραφείου και server. Αν συνυπολογίσουμε και τους embedded μικροεπεξεργαστές και μικροελεγκτές τότε ο συνολικός αριθμός των μικροεπεξεργαστών που πωλούνται κάθε χρόνο υπερβαίνει κατά πολύ το ένα δισεκατομμύριο.

Οι μικροεπεξεργαστές είναι επεξεργαστές συνόλου εντολών (instruction set processors, ISPs). Ένας ISP εκτελεί εντολές ενός προκαθορισμένου συνόλου εντολών (ή αλλιώς σετ εντολών). Η λειτουργικότητα ενός μικροεπεξεργαστή εξαρτάται πλήρως από το σύνολο εντολών που είναι ικανός να εκτελέσει. Όλα τα προγράμματα που τρέχουν σε έναν μικροεπεξεργαστή κωδικοποιούνται σε αυτό το σύνολο εντολών. Αυτό το προκαθορισμένο σύνολο εντολών ονομάζεται επίσης αρχιτεκτονική συνόλου εντολών (Instruction Set Architecture, ISA). Το ISA χρησιμεύει ως μία διασύνδεση ανάμεσα στο λογισμικό (software) και το υλικό (hardware), δηλαδή ανάμεσα στα προγράμματα και στους επεξεργαστές. Σε ορολογία της μεθοδολογίας σχεδίασης επεξεργαστών, το ISA είναι ο καθορισμός της σχεδίασης ενώ ο επεξεργαστής ή ο ISP είναι η υλοποίηση της σχεδίασης.

1.1 Η εξέλιξη των μικροεπεξεργαστών

Ο πρώτος μικροεπεξεργαστής, ο Intel 4004, παρουσιάστηκε το 1971. Ο Intel 4004 ήταν ένας 4bit επεξεργαστής που αποτελούταν από περίπου 2300 τρανζίστορς με συχνότητα ρολογιού λίγο πάνω από τα 100 KHz. Η αρχική του εφαρμογή ήταν η δημιουργία αριθμομηχανών. Το έτος 2001σήμανε την τριακοστή επέτειο της γέννησης των μικροεπεξεργαστών. Μικροεπεξεργαστές υψηλού-τέλους (high-end microprocessors), οι οποίοι αποτελούνται από εκατοντάδες εκατομμύρια τρανζίστορς με συχνότητα ρολογιού που άγγιζε τα 2 GHz, ήταν οι δομικές μονάδες για τα συστήματα υπερυπολογιστών και πανίσχυρων συστημάτων πελάτη-εξυπηρετητή (client-server systems) που υπάρχουν στο Internet. Στη σημερινή εποχή η συχνότητα ρολογιού των μικροεπεξεργαστών υπερβαίνει τα 3,5 GHz και μέσα στα επόμενα χρόνια αναμένεται να φτάσει τα 10 GHz ενώ κάθε μικροεπεξεργαστής θα αποτελείται από αρκετές εκατοντάδες εκατομμύρια τρανζίστορς.

Οι τρεις δεκαετίες της ιστορίας των μικροεπεξεργαστών λένε μία πραγματικά αξιοπρόσεκτη ιστορία όσων αφορά την τεχνολογική πρόοδο στην βιομηχανία υπολογιστών. Αυτή η πρόοδος φαίνεται καλύτερα στον πίνακα 1 που ακολουθεί:

	1970-1980	1980-1990	1990-2000	2000-2010
Πλήθος τρανζίστορ (transistor count)	2K-100K	100K-1M	1M-100M	100M-2B
Συχνότητα ρολογιού (clock frequency)	0.1-3 MHz	3-30 MHz	30 MHz-1GHz	1-15 GHz
Εντολές/κύκλο (Instructions/cycle, IPC)	0.1	0.1-0.9	0.9-1.9	1.9-2.9

Πίνακας 1.1 Οι δεκαετίες εξέλιξης των μικροεπεξεργαστών.

Η εξέλιξη των μικροεπεξεργαστών έχει κατά βάση ακολουθήσει τον περίφημο νόμο του Moore (Moore's law), που παρατηρήθηκε από τον Gordon Moore το 1965 και σύμφωνα με τον οποίο ο αριθμός των συσκευών που μπορούν να ολοκληρωθούν (με την έννοια της ολοκλήρωσης στην ορολογία της σχεδίασης ολοκληρωμένων κυκλωμάτων) σε ένα απλό κομμάτι πυριτίου διπλασιάζεται κάθε 18-24 μήνες. Σε λιγότερο από 30 χρόνια μάλιστα, ο αριθμός των τρανζίστορς σε ένα chip μικροεπεξεργαστή έχει αυξηθεί κατά πάνω από 4 τάξεις μεγέθους όπως φαίνεται και στον πίνακα. Στην ίδια χρονική περίοδο, η απόδοση του μικροεπεξεργαστή έχει αυξηθεί κατά πάνω από 5 τάξεις μεγέθους.. Επίσης, στις τελευταίες δύο δεκαετίες η απόδοση του μικροεπεξεργαστή διπλασιάζεται κάθε 18 μήνες, ή διαφορετικά εκατονταπλασιάζεται (x100) σε κάθε δεκαετία. Αυτή η εντυπωσιακή βελτίωση της απόδοσης των μικροεπεξεργαστών αποτελεί μοναδικό φαινόμενο και δεν έχει συναντηθεί σε καμία άλλη βιομηχανία.

Κατά τη διάρκεια της πρώτης δεκαετίας η έλευση του 4bit μικροεπεξεργαστή οδήγησε γρήγορα στην παρουσίαση του 8bit μικροεπεξεργαστή. Αυτοί εξελίχθηκαν σε μικροελεγκτές που χρησιμοποιήθηκαν σε embedded εφαρμογές, από μηχανές πλυσίματος μέχρι ανελκυστήρες και μηχανές jet. Ο 8bit μικροεπεξεργαστής έγινε επίσης η καρδιά μίας νέας διάσημης υπολογιστικής πλατφόρμας, γνωστής ως προσωπικός υπολογιστής (Personal Computer, PC) και εισήγαγε την εποχή των PCs.

Η δεκαετία του 1980 υπήρξε μάρτυρας πολλών σημαντικών αλλαγών στην αρχιτεκτονική και την μικροαρχιτεκτονική των 32bit μικροεπεξεργαστών. Τα προβλήματα που αφορούν τη σχεδίαση του συνόλου εντολών έγιναν το σημαντικότερο αντικείμενο των ακαδημαϊκών και βιομηχανικών ερευνών. Το pipelining των εντολών και οι γρήγορες κρυφές μνήμες (cache memories) έγιναν αναντικατάστατες τεχνικές μικροαρχιτεκτονικής.

Κατά τη διάρκεια της δεκαετίας του 1990 οι μικροεπεξεργαστές έγιναν η πιο πανίσχυρη και δημοφιλής μορφή υπολογιστών. Η συχνότητα ρολογιού των ταχύτερων μικροεπεξεργαστών ξεπέρασαν τις συχνότητες ρολογιού των ταχύτερων υπερυπολογιστών. Οι προσωπικοί υπολογιστές και οι σταθμοί εργασίας έγιναν σημαντικά και αναντικατάστατα εργαλεία της παραγωγικότητας και των επικοινωνιών. Χρησιμοποιήθηκαν ιδιαίτερα επιθετικές τεχνικές μικροαρχιτεκτονικής για την επίτευξη απόδοσης μικροεπεξεργαστών σε επίπεδα που δεν είχαν επιτευχθεί ποτέ μέχρι τότε. Επίσης έγιναν δημοφιλείς βαθιά pipelined μηχανές (δηλαδή μηχανές με σωληνώσεις πολλών σταδίων), ικανές να επιτύχουν εξαιρετικά υψηλές συχνότητες ρολογιού και να εκτελέσουν πολλές εντολές ανά κύκλο ρολογιού. Ακόμη χρησιμοποιήθηκαν πολύ επιθετικές τεχνικές πρόβλεψης διακλαδώσεων όπως επίσης και η out-of-order εκτέλεση των εντολών ώστε να μειωθούν στο ελάχιστο οι χαμένοι κύκλοι ρολογιού λόγω καθυστερήσεων του pipeline (pipeline stalls).

Πλέον βρισκόμαστε ήδη στη τέταρτη δεκαετία των μικροεπεξεργαστών και φαίνεται ότι η ροπή αυτή δεν δείχνει σημάδια ελάττωσης. Οι περισσότεροι

τεχνολόγοι συμφωνούν ότι ο νόμος του Moore θα συνεχίσει να ισχύει για τουλάχιστον 10 με 15 χρόνια. Ως το 2010 μπορούμε να αναμένουμε ότι οι μικροεπεξεργαστές θα αποτελούνται από περισσότερα του ενός δισεκατομμύρια τρανζίστορς με συχνότητα ρολογιού μεγαλύτερη από 10 GHz. Η εστίαση που τις πρώτες τρεις δεκαετίες ήταν μόνο στον παραλληλισμό στο επίπεδο της εντολής (Instruction-level parallelism, ILP) έχει ήδη περάσει στον παραλληλισμό στο επίπεδο του νήματος (Thread-level parallelism, TLP) και στον παραλληλισμό στο επίπεδο της μνήμης (Memory-level parallelism, MLP). Επίσης πολλά ζητήματα που αφορούσαν παραδοσιακά την «μακροαρχιτεκτονική» γίνονται ζητήματα που αφορούν τη μικροαρχιτεκτονική (δηλαδή χαρακτηριστικά που αφορούσαν μεγάλα συστήματα τώρα υλοποιούνται πάνω σε ένα chip). Η κατανάλωση ισχύος έχει γίνει επίσης ένας πολύ σημαντικός παράγοντας της απόδοσης και απαιτούνται νέες λύσεις σε όλα τα επίπεδα της ιεραρχίας σχεδίασης, που περιλαμβάνουν την διαδικασία παραγωγής, τη λογική σχεδίαση, τη σχεδίαση σε επίπεδο μικροαρχιτεκτονικής και το rum-time περιβάλλον του λογισμικού, έτσι ώστε να διατηρηθούν τα ίδια επίπεδα βελτίωσης της απόδοσης που επιτεύχθηκαν κατά τις προηγούμενες τρεις δεκαετίες.

1.2 Σχεδίαση συνόλου εντολών επεξεργαστή (instruction set processor design)

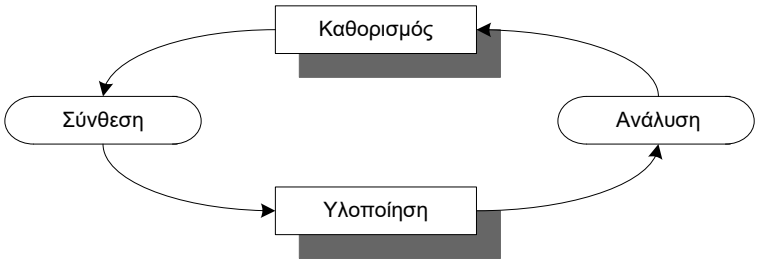
Ιδιαίτερα κρίσιμο ρόλο στη σχεδίαση του συνόλου εντολών ενός επεξεργαστή παίζει η αρχιτεκτονική του συνόλου εντολών, η οποία καθορίζει τη λειτουργικότητα που πρέπει να έχει ο επεξεργαστής συνόλου εντολών.

1.2.1 Σχεδίαση ψηφιακών συστημάτων

Τυπικά η διαδικασία σχεδίασης περιλαμβάνει δύο σημαντικές εργασίες:

- τη σύνθεση,
- την ανάλυση.

Η σύνθεση προσπαθεί να βρει μία υλοποίηση βασισμένη στον καθορισμό της σχεδίασης (specification) ενώ η ανάλυση εξετάζει την υλοποίηση της σχεδίασης (implementation) για να καθοριστεί το αν και κατά πόσο πληρούνται οι απαιτήσεις του καθορισμού σχεδίασης. Η σύνθεση είναι η περισσότερο δημιουργική εργασία από τις δύο , ενώ γενικά αναζητεί για πιθανές λύσεις και κάνει διάφορους συμβιβασμούς (tradeoffs, δηλαδή ενέργειες που ωφελούν στην βελτιστοποίηση ενός χαρακτηριστικού αλλά προκαλούν την χειροτέρευση ενός άλλου χαρακτηριστικού) και βελτιστοποιήσεις σχεδίασης ώστε να βρεθεί η καλύτερη δυνατή λύση. Η κρίσιμη εργασία της ανάλυσης είναι απαραίτητη για την απόφαση όσων αφορά την ορθότητα και την αποτελεσματικότητα μίας σχεδίασης. Συχνά χρησιμοποιούνται εργαλεία προσομοίωσης (simulation tools) για τον έλεγχο εγκυρότητας της σχεδίασης (design validation) και για την εκτίμηση απόδοσης (performance evaluation). Μία τυπική διαδικασία σχεδίασης μπορεί να απαιτεί την εναλλαγή ανάλυσης-σύνθεσης αρκετές φορές προτού βρεθεί η καλύτερη δυνατή λύση. Αυτός ο κύκλος φαίνεται στο ακόλουθο σχήμα.



Σχήμα 1.1 Μηχανική σχεδίαση.

Ο καθορισμός μίας σχεδίασης μικροεπεξεργαστή είναι η αρχιτεκτονική συνόλου εντολών, η οποία καθορίζει ένα σύνολο εντολών το οποίο ο συγκεκριμένος μικροεπεξεργαστής πρέπει να είναι ικανός να εκτελεί. Η υλοποίηση είναι η ίδια η σχεδίαση υλικού, που περιγράφεται με τη χρήση μίας γλώσσας περιγραφής υλικού (hardware definition language, HDL). Τα δομικά στοιχεία που χρησιμοποιεί μία γλώσσα περιγραφής υλικού μπορούν να είναι από λογικές πύλες και flip-flops μέχρι περισσότερο σύνθετα μοντέλα, όπως αποκωδικοποιητές και πολυπλέκτες, ή μέχρι ακόμα και ολόκληρες λειτουργικές οντότητες, όπως αθροιστές και πολλαπλασιαστές. Μία σχεδίαση μπορεί να περιγραφεί ως ένα κύκλωμα, ή εσωτερικά συνδεδεμένη οργάνωση, αυτών των δομικών στοιχείων της γλώσσας.

Η διαδικασία σχεδίασης ενός μοντέρνου high-end μικροεπεξεργαστή περιέχει δύο βασικά βήματα:

- την μικροαρχιτεκτονική σχεδίαση,
- την λογική σχεδίαση.

Η μικροαρχιτεκτονική σχεδίαση εμπερικλείει την ανάπτυξη και τον ορισμό των βασικών τεχνικών που θα χρησιμοποιηθούν για την επίτευξη της επιθυμητής απόδοσης. Συνήθως χρησιμοποιείται ένα μοντέλο απόδοσης ως εργαλείο ανάλυσης για τον έλεγχο της αποτελεσματικότητας των τεχνικών αυτών. Αυτά τα μοντέλα απόδοσης ελέγχουν τη συμπεριφορά της μηχανής κατά τις εναλλαγές κύκλων ρολογιού και είναι ικανά να μετρήσουν των αριθμό των κύκλων μηχανής που απαιτούνται για την εκτέλεση ενός benchmark προγράμματος. Το τελικό αποτέλεσμα της μικροαρχιτεκτονικής σχεδίασης είναι μία περιγραφή της οργάνωσης του μικροεπεξεργαστή σε υψηλό επίπεδο. Αυτή η περιγραφή χρησιμοποιεί συνήθως μία γλώσσα μεταφοράς καταχωρητών (register transfer language, RTL) για τον καθορισμό όλων των κύριων μοντέλων στην οργάνωση της μηχανής και τον καθορισμό των ενεργειών ανάμεσα σε αυτά τα μοντέλα-μονάδες. Κατά τη διάρκεια του βήματος λογικής σχεδίασης, η RTL περιγραφή επανακαθορίζεται επιτυχώς αφού πρώτα ληφθούν υπόψη ορισμένες λεπτομέρειες της υλοποίησης, και έτσι δημιουργείται μία HDL περιγραφή της πραγματικής hardware σχεδίασης (υπάρχει έτσι ως αποτέλεσμα το σχέδιο σε επίπεδο hardware του μικροεπεξεργαστή). Οι RTL και HDL περιγραφές μπορούν και οι δύο να χρησιμοποιούν την ίδια γλώσσα περιγραφής. Η Verilog είναι ένα τέτοιο παράδειγμα γλώσσας. Στην συγκεκριμένη διπλωματική έγινε χρήση της γλώσσας περιγραφής υλικού VHDL και το περιβάλλον εργασίας που χρησιμοποιήθηκε ήταν το Active-HDL 3.6.

1.2.2 Αρχιτεκτονική συνόλου εντολών

Η αρχιτεκτονική συνόλου εντολών παίζει έναν εξαιρετικά σημαντικό ρόλο και θεωρείται ως ένας σύνδεσμος ανάμεσα στο software και το hardware, ή ανάμεσα στο πρόγραμμα και τη μηχανή. Με αυτόν τον τρόπο είναι δυνατό τα προγράμματα και η μηχανή να αναπτυχθούν ανεξάρτητα. Προγράμματα που έχουν ως στόχο μία

συγκεκριμένη αρχιτεκτονική συνόλου εντολών (ISA) μπορούν να δημιουργηθούν χωρίς να απαιτείται γνώση της πραγματικής υλοποίησης της μηχανής στην οποία θα εκτελεστούν. Από την άλλη μεριά, εντελώς παρόμοια, μηχανές που έχουν σαν στόχο να υλοποιήσουν ένα συγκεκριμένο ISA μπορούν να σχεδιαστούν χωρίς να ληφθούν υπόψη τα προγράμματα που θα εκτελεστούν σε αυτές.

Η ύπαρξη του ISA διασφαλίζει και την δυνατότητα για φορητό λογισμικό. Ένα πρόγραμμα που γράφεται για ένα συγκεκριμένο ISA μπορεί να εκτελεστεί σε όλες τις υλοποιήσεις αυτού του ISA. Αυτή η δυνατότητα μειώνει αισθητά το κόστος της ανάπτυξης λογισμικού και αυξάνει παράλληλα το χρόνο ζωής του λογισμικού.

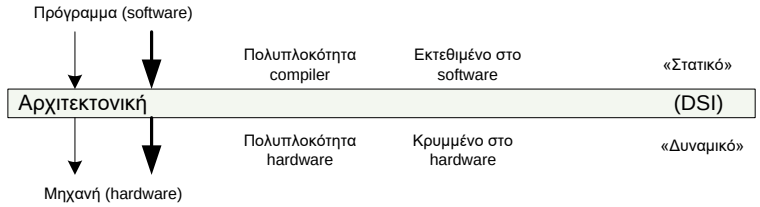
Η σχεδίαση του μικροεπεξεργαστή αρχίζει με το ISA και παράγει μία μικροαρχιτεκτονική που καλύπτει τις απαιτήσεις του καθορισμού σχεδίασης, δηλαδή του specification. Κάθε νέα μικροαρχιτεκτονική πρέπει να ελεγχθεί ως προς την εγκυρότητά της όσων αφορά το ISA, έτσι ώστε να συμφωνεί με τις λειτουργικές απαιτήσεις του συγκεκριμένου ISA. Αυτό είναι ιδιαίτερα σημαντικό γιατί πρέπει να εξασφαλιστεί ότι το υπάρχον software μπορεί να εκτελεστεί σωστά στην νέα μικροαρχιτεκτονική.

Μέχρι στιγμής έχουν αναπτυχθεί και χρησιμοποιηθεί πολλά ISA. Αυτά διαφέρουν ως προς τον καθορισμό των λειτουργιών των εντολών και των ορισμάτων τους. Γενικά ένα ISA καθορίζει ένα σύνολο εντολών που ονομάζονται εντολές assembly. Κάθε εντολή καθορίζει με τη σειρά της μία λειτουργία (ή αλλιώς πράξη) και ένα ή περισσότερα ορίσματα. Κάθε ISA ορίζει με μοναδικό τρόπο μία γλώσσα assembly. Φυσικά ένα πρόγραμμα assembly απαρτίζεται από μία σειρά εντολών assembly. Πολλά αρχικά ISA χρησιμοποιούσαν έναν συσσωρευτή (accumulator) ως υπονοούμενο όρισμα. Σε μία αρχιτεκτονική βασισμένη σε συσσωρευτή ο μοναδικός συσσωρευτής χρησιμοποιείται ταυτόχρονα ως το υπονοούμενο όρισμα και ως προορισμός. Άλλα ISA θεωρούσαν ότι τα αναγνωριστικά σώζονται σε μία δομή στοίβας (last in, first out (LIFO)) και οι λειτουργίες εκτελούνται στην πάνω ή στις δύο πάνω εγγραφές της στοίβας. Τα πιο σύγχρονα ISA θεωρούν ότι τα ορίσματα σώζονται σε ένα αρχείο καταχωρητών πολλών εγγραφών και ότι όλες οι αριθμητικές και λογικές πράξεις εκτελούνται πάνω σε ορίσματα που βρίσκονται στους καταχωρητές. Ειδικές εντολές, όπως οι εντολές φόρτωσης και αποθήκευσης δεδομένων από και προς της μνήμη αντίστοιχα (load/store instructions), δημιουργήθηκαν για να υπάρχει δυνατότητα μετακίνησης ορισμάτων από το αρχείο καταχωρητών στην κύρια μνήμη και ανάποδα. Μερικά παραδοσιακά ISA επιτρέπουν τα ορίσματα να έρχονται απευθείας και από το αρχείο καταχωρητών και από την κύρια μνήμη.

Όπως είναι λογικό τα διάφορα ISA εξελίσσονται πολύ αργά λόγω της δυσκολίας της ανάπτυξης καινούριου software (για το νέο ISA απαιτείται νέο πρόγραμμα) και του νέου compilation που απαιτείται. Η ανάπτυξη νέων compilers και λειτουργικών συστημάτων για ένα νέο ISA χρειάζεται συνήθως περισσότερο από 10 χρόνια. Όσο περισσότερο υπάρχει ένα ISA τόσο μεγαλύτερη είναι η βάση του software που είναι βασισμένο πάνω σε αυτό το ISA και επομένως τόσο πιο δύσκολο είναι να αντικατασταθεί αυτό το ISA. Παρόλα αυτά μπορούν να υπάρξουν πολλές μικροαρχιτεκτονικές για το ίδιο ISA και οι έρευνες στη σημερινή εποχή συγκεντρώνονται στην υλοποίηση όλο και πιο γρήγορων αρχιτεκτονικών που υλοποιούν ήδη υπάρχοντα ISA.

1.2.3 Δυναμική-στατική διασύνδεση (dynamic-static interface)

Το ISA εκτός από τον διπλό ρόλο που έχει ως σύνδεσμος ανάμεσα στο hardware και το software και ως καθορισμός μίας σχεδίασης μικροεπεξεργαστή, έχει και έναν τρίτο ρόλο. Παράλληλα με τον ορισμό ενός ISA καθορίζεται και μία διασύνδεση (interface) που διαχωρίζει αυτά που γίνονται στατικά κατά τη διάρκεια του compilation και αυτά που γίνονται δυναμικά κατά τη διάρκεια της εκτέλεσης. Αυτή η διασύνδεση ονομάστηκε δυναμική-στατική διασύνδεση (dynamic-static interface, DSI) από τον Yale Patt και φαίνεται στο ακόλουθο σχήμα.

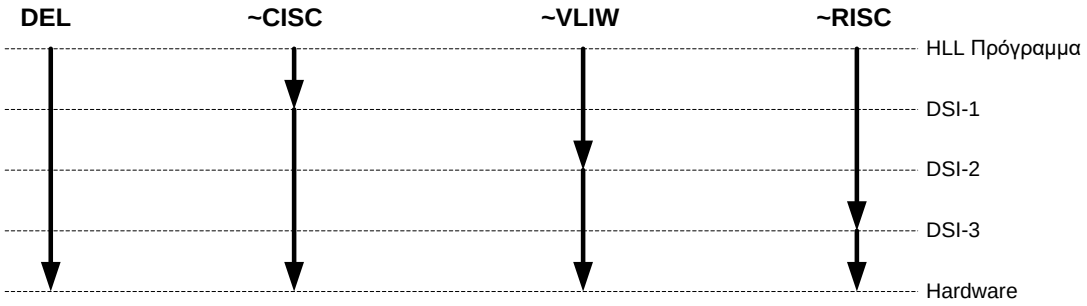


Σχήμα 1.2 Η δυναμική-στατική διασύνδεση (dynamic-static interface).

Το DSI είναι ένα φυσικό επακόλουθο της χρήσης του ISA ως σύνδεσμος ανάμεσα στο software και το hardware. Παραδοσιακά, όλες οι εργασίες και βελτιστοποιήσεις που γίνονται στο στατικό επίπεδο κατά τη διάρκεια του compilation σχετίζονται με το software και τον compiler και θεωρείται ότι βρίσκονται «πάνω» από το DSI. Αντίθετα, όλες οι εργασίες και βελτιστοποιήσεις που γίνονται στο δυναμικό επίπεδο κατά τη διάρκεια της εκτέλεσης σχετίζονται με το hardware και θεωρείται ότι βρίσκονται «κάτω» από το DSI.

Υπάρχουν γενικά διάφορα επίπεδα αφαίρεσης όπου το DSI μπορεί να τοποθετηθεί. Ιδεατά το DSI πρέπει να τοποθετηθεί σε ένα επίπεδο όπου επιτυγχάνεται η μέγιστη συνέργια ανάμεσα στις στατικές τεχνικές και τις δυναμικές τεχνικές, δηλαδή σε ένα επίπεδο όπου ο συνδυασμός της πολυπλοκότητας του compiler και του hardware βρίσκεται σε τέτοιο επίπεδο που προκαλεί την επιθυμητή απόδοση.

Στην ιστορία της σχεδίασης των ISA έχουν προταθεί πολλές τοποθετήσεις του DSI, μερικές από τις οποίες φαίνονται στο επόμενο σχήμα.



Σχήμα 1.3 Επίδειξη των πιθανών τοποθετήσεων του DSI σε μία σχεδίαση ISA.

Ο Mike Flynn έχει προτείνει την τοποθέτηση του DSI πολύ ψηλά. Με αυτόν τον τρόπο βέβαια όλα πρέπει να γίνονται κάτω από το DSI και επομένως κάθε πρόγραμμα που είναι γραμμένο σε μία γλώσσα υψηλού επιπέδου μπορεί να εκτελεστεί απευθείας από μία απευθείας εκτελέσιμη γλώσσα (directly executable

language, DEL). Ένα ISA ενός υπολογιστή πολύπλοκου συνόλου εντολών (complex instruction set computer, CISC) τοποθετεί το DSI στο επίπεδο της παραδοσιακής γλώσσας assembly (ή αλλιώς στο επίπεδο μακροκώδικα). Στον αντίποδα, ένα ISA ενός υπολογιστή μειωμένου συνόλου εντολών (reduced instruction set computer, RISC) κατεβάζει το DSI και απαιτεί την εκτέλεση των περισσότερων βελτιστοποιήσεων πάνω από το DSI μέσω του compiler. Το κατέβασμα του DSI εκθέτει και ανεβάζει αυτά που κανονικά θα θεωρούνταν μικροαρχιτεκτονικά στοιχεία σε ένα CISC ISA στο επίπεδο του ISA. Ο σκοπός αυτής της επιλογής είναι η μείωση της πολυπλοκότητας του hardware και ως εκ τούτου η δημιουργία μίας πολύ ταχύτερης μηχανής. Αυτός είναι και ο λόγος που στη συγκεκριμένη διπλωματική επιλέχθηκε η σχεδίαση και υλοποίηση ενός RISC επεξεργαστή.

1.3 Αρχές της απόδοσης επεξεργαστή

Κάθε νέα γενιά της μικροαρχιτεκτονικής έχει ως στόχο να βελτιώσει την απόδοση της προηγούμενης γενιάς. Τα τελευταία χρόνια η μείωση της κατανάλωσης ισχύος έχει αναδειχθεί επίσης σε μείζον θέμα, όμως η απαίτηση για καλύτερη απόδοση θα υπάρχει πάντοτε και η απόδοση επεξεργαστή θα συνεχίζει να είναι ένα πολύ βασικό ζητούμενο της σχεδίασης.

1.3.1 Εξίσωση απόδοσης επεξεργαστή (processor performance equation)

Κατά τη διάρκεια της δεκαετίας του 1980 αρκετοί ερευνητές ανακάλυψαν μία ισότητα που ορίζει ξεκάθαρα την απόδοση του επεξεργαστή και χαρακτηρίζει με σαφή τρόπο τους βασικούς παράγοντες από τους οποίους εξαρτάται η απόδοση. Αυτή η εξίσωση έχει γίνει ευρέως γνωστή με τον όρο «σιδερένιος νόμος» (“iron law”) της απόδοσης επεξεργαστή και δίνεται από την εξής σχέση:

$$\frac{1}{\text{Απόδοση}} = \frac{\text{χρόνος}}{\text{πρόγραμμα}} = \frac{\text{εντολές}}{\text{πρόγραμμα}} \times \frac{\text{κύκλοι}}{\text{εντολή}} \times \frac{\text{χρόνος}}{\text{κύκλο}} \quad \text{(Σχέση 1.1)}$$

Καταρχάς πρέπει να παρατηρηθεί ότι η εξίσωση απόδοσης επεξεργαστή δείχνει ότι η απόδοση εξαρτάται από το πόσος χρόνος χρειάζεται για να εκτελεστεί ένα συγκεκριμένο πρόγραμμα (χρόνος/πρόγραμμα). Επίσης παρατηρούμε ότι ο όρος χρόνος/πρόγραμμα ή χρόνος εκτέλεσης (execution time) μπορεί να γραφεί ως το γινόμενο των εξής τριών όρων:

- εντολές/πρόγραμμα,
- κύκλοι/εντολή,
- χρόνος/κύκλο.

Ο πρώτος όρος δείχνει τον συνολικό αριθμό των δυναμικών εντολών (σε χρόνο εκτέλεσης δηλαδή και όχι του compilation) που πρέπει να εκτελεστούν για ένα συγκεκριμένο πρόγραμμα. Αυτός ο όρος ονομάζεται και αριθμός εντολών (instruction count). Ο δεύτερος όρος δείχνει κατά μέσο όρο (μέσος όρος πάνω σε ολόκληρη την εκτέλεση του προγράμματος) πόσοι κύκλοι μηχανής απαιτούνται για την εκτέλεση της κάθε εντολής. Αυτός ο όρος είναι γνωστός ως CPI (cycles per instruction). Ο τρίτος όρος δείχνει τη χρονική διάρκεια κάθε κύκλου μηχανής, δηλαδή τον χρόνο κύκλου (cycle time) της μηχανής.

Όσο μικρότερος είναι ο χρόνος εκτέλεσης του προγράμματος τόσο καλύτερη είναι η απόδοση. Από την εξίσωση γίνεται εύκολα αντιληπτό ότι για να αυξηθεί η απόδοση θα πρέπει να αυξηθεί ένας ή περισσότεροι από τους τρεις παράγοντες που

αναφέραμε. Αν ο αριθμός εντολών μειωθεί, τότε θα υπάρχουν λιγότερες εντολές προς εκτέλεση και ο χρόνος εκτέλεσης του προγράμματος θα μειωθεί. Αν το CPI μειωθεί, τότε κατά μέσο όρο κάθε εντολή θα απαιτεί λιγότερους κύκλους μηχανής για να εκτελεστεί. Αν ο χρόνος κύκλου μειωθεί, τότε κάθε κύκλος θα καταναλώνει λιγότερο χρόνο και ο συνολικός χρόνος εκτέλεσης θα μειωθεί. Όλα αυτά οδηγούν σε αύξηση της απόδοσης του επεξεργαστή. Δυστυχώς όμως αυτοί οι τρεις όροι δεν είναι μεταξύ τους ανεξάρτητοι και υπάρχουν πολύπλοκες εξαρτήσεις μεταξύ τους. Η μείωση κάθε ενός από τους τρεις αυτούς όρους μπορεί να προκαλέσει την αύξηση των άλλων δύο όρων. Η σχέση μάλιστα μεταξύ των τριών όρων δεν μπορεί εύκολα να χαρακτηριστεί και να μοντελοποιηθεί. Η βελτίωση της απόδοσης γίνεται έτσι μία πραγματική πρόκληση και απαιτεί λεπτούς συμβιβασμούς και ιδιαίτερα προσεκτικές ενέργειες εξισορρόπησης.

1.3.2 Βελτιστοποιήσεις της απόδοσης επεξεργαστή

Μερικές τεχνικές μπορούν να μειώσουν έναν όρο αφήνοντας τους άλλους δύο ανεπηρέαστους. Για παράδειγμα, όταν ο compiler κάνει βελτιστοποιήσεις οι οποίες εξαλείφουν πλεονάζουσες και άχρηστες εντολές στον εκτελέσιμο κώδικα, τότε ο αριθμός εντολών μειώνεται χωρίς να επηρεάζονται το CPI ή ο χρόνος κύκλου. Ακόμα, όταν χρησιμοποιείται μία τεχνολογία ταχύτερων κυκλωμάτων ή μία περισσότερο ανεπτυγμένη διαδικασία παραγωγής του πυριτίου, τότε μειώνονται οι καθυστερήσεις μετάδοσης του σήματος ανάμεσα στις λογικές πύλες και αυτό έχει ως αποτέλεσμα τη μείωση του χρόνου κύκλου χωρίς παράλληλα να επηρεάζονται οι άλλοι δύο όροι. Τέτοιου είδους τεχνικές βελτιστοποίησης είναι πάντα επιθυμητές και πρέπει να εφαρμόζονται όταν το κόστος τους δεν είναι απαγορευτικό.

Υπάρχουν φυσικά και διάφορες τεχνικές οι οποίες ενώ μειώνουν έναν από τους όρους, αυξάνουν ταυτόχρονα τον έναν από τους άλλους δύο όρους ή και τους δύο. Για αυτές τις τεχνικές υπάρχει κέρδος απόδοσης μόνο αν η μείωση του όρου δεν επικαλύπτεται από τις αυξήσεις των άλλων δύο όρων.

Υπάρχουν διάφοροι τρόποι να μειωθεί ο αριθμός εντολών. Καταρχάς, το σύνολο εντολών μπορεί να περιέχει πιο σύνθετες εντολές που να εκτελούν περισσότερες λειτουργίες ανά εντολή. Ο συνολικός αριθμός των εντολών που εκτελούνται μπορεί με αυτόν τον τρόπο να μειωθεί σημαντικά. Για παράδειγμα, ένα πρόγραμμα σε ένα RISC ISA μπορεί να απαιτεί τις διπλάσιες εντολές από το αντίστοιχο πρόγραμμα σε ένα CISC ISA. Παρότι ο αριθμός των εντολών όμως μειώνεται, η πολυπλοκότητα της μονάδας επεξεργασίας αυξάνεται, κάτι που οδηγεί σε πιθανή αύξηση του χρόνου κύκλου. Επίσης, αν χρησιμοποιείται βαθύ pipelining για να αποφευχθεί η αύξηση του χρόνου κύκλου, τότε μία υψηλότερη ποινή λανθασμένης πρόβλεψης διακλαδώσεως μπορεί να έχει ως αποτέλεσμα υψηλότερο CPI. Ένας δεύτερος τρόπος μείωσης του αριθμού εντολών είναι μερικές συγκεκριμένες βελτιστοποιήσεις από τον compiler. Για παράδειγμα τα λεγόμενα μη-περιστρεφόμενα loops (unrolling loops) μπορούν να μειώσουν τον αριθμό εντολών, όμως μπορεί να οδηγήσουν σε αύξηση του μεγέθους του στατικού κώδικα, κάτι που θα μειώσει το ποσοστό επιτυχίας στις αναγνώσεις της μνήμης εντολών (I-cache hit rate) και επομένως θα αυξήσει το CPI. Έτσι παρατηρούμε ότι η μείωση του αριθμού εντολών είναι πιθανό να οδηγήσει στην αύξηση του CPI και/ή του χρόνου κύκλου.

Η ανάγκη μείωσης του CPI έχει γίνει πηγή έμπνευσης πολλών αρχιτεκτονικών και μικροαρχιτεκτονικών εντολών. Ένα από τα σημαντικότερα κίνητρα για την δημιουργία των RISC επεξεργαστών ήταν η μείωση της πολυπλοκότητας κάθε εντολής με σκοπό της μείωση του αριθμού των κύκλων μηχανής που απαιτούνται για

την εκτέλεση κάθε εντολής. Όπως έχει ήδη αναφερθεί, αυτό έχει ως αρνητικό αποτέλεσμα την αύξηση του αριθμού εντολών. Άλλη μία βασική τεχνική μείωσης του CPI είναι το pipelining των εντολών (ή αλλιώς «σωλήνωση»).Ένας pipelined επεξεργαστής μπορεί να επικαλύψει την εκτέλεση πολλαπλών εντολών, δηλαδή όταν μία εντολή βρίσκεται σε μία φάση της επεξεργασίας της, κάποια άλλη μπορεί να βρίσκεται σε κάποια άλλη φάση επεξεργασίας. Οι φάσεις αυτές μάλιστα λέγονται στάδια του pipeline. Σε σύγκριση με μία μη-pipelined σχεδίαση και υποθέτοντας ίσο χρόνο κύκλου, μία pipelined σχεδίαση μπορεί να μειώσει αισθητά το CPI. Ένα πιο ρηχό pipeline, δηλαδή ένα pipeline με λιγότερα στάδια, μπορεί να παραγάγει χαμηλότερο CPI από ένα βαθύτερο pipeline, άλλα με το κόστος του αυξημένου χρόνου κύκλου. Η χρήση της μνήμης cache (δηλαδή της «κρυφής» μνήμης) με σκοπό τη μείωση της μέσης καθυστέρησης πρόσβασης μνήμης (σε όρους αριθμού κύκλων ρολογιού) μειώνει επίσης το CPI. Όταν γίνει μία διακλάδωση υπό συνθήκη (conditional branch) δημιουργούνται χαμένοι κύκλοι (stall cycles) λόγω της ανάγκης φόρτωσης των νέων εντολών από μη-συνεχόμενες θέσεις μνήμης της I-cache. Οι μονάδες πρόβλεψης διακλάδωσης (branch predictors) μπορούν να μειώσουν των αριθμό αυτών των χαμένων κύκλων, οδηγώντας στη μείωση του CPI. Παρόλα αυτά η πρόσθεση τέτοιων μονάδων μπορεί να αυξήσει τον χρόνο κύκλου λόγω της αύξησης της πολυπλοκότητας του hardware στο στάδιο φόρτωσης των εντολών (στάδιο fetch) του pipeline ή να αυξήσει το CPI αν απαιτείται ένα βαθύτερο pipeline για την διατήρηση του ίδιου χρόνου κύκλου. Η εμφάνιση των υπερβαθμωτών επεξεργαστών (superscalar processors) επέτρεψε στο pipeline του επεξεργαστή να επεξεργάζεται ταυτόχρονα πολλές εντολές σε κάθε στάδιό του. Με αυτήν την πολλαπλή επεξεργασία εντολών σε κάθε κύκλο μηχανής το CPI μειώνεται σημαντικά. Φυσικά όμως, η πολυπλοκότητα του κάθε σταδίου του pipeline αυξάνει με αποτέλεσμα την αύξηση του χρόνου κύκλου ή του βάθους του pipeline, που οδηγεί με της σειρά του σε αύξηση του CPI.

Η κύρια μικροαρχιτεκτονική τεχνική για τη μείωση του χρόνου κύκλου είναι το pipelining. Το pipelining διαχωρίζει το έργο της επεξεργασίας μίας εντολής σε πολλά στάδια. Η καθυστέρηση (σε όρους καθυστέρησης μετάδοσης σήματος) κάθε σταδίου της «σωλήνωσης» καθορίζει τον χρόνο κύκλου μηχανής. Με τη χρήση βαθύτερων pipeline η καθυστέρηση κάθε σταδίου, και επομένως και ο χρόνος κύκλου, μπορεί να μειωθεί. Τα τελευταία χρόνια το επιθετικό pipelining αποτελεί την βασικότερη τεχνική που χρησιμοποιείται για να επιτευχθούν τρομερές αυξήσεις της συχνότητας ρολογιού των high-end επεξεργαστών. Όπως φαίνεται και από τον Πίνακα 1, κατά την τελευταία δεκαετία (1990-2000) η σημαντικότερη αύξηση της απόδοσης οφειλόταν στην αύξηση της συχνότητας ρολογιού.

Υπάρχει πάντως και ένα αρνητικό όσων αφορά την αύξηση της συχνότητας ρολογιού μέσω βαθύτερου pipelining. Όταν το pipeline γίνεται βαθύτερο, το CPI μπορεί να αυξηθεί με τρεις διαφορετικούς τρόπους:

- όταν το αρχικό κομμάτι του pipeline γίνεται βαθύτερο, ο αριθμός των σταδίων ανάμεσα στο στάδιο φόρτωσης (fetch) και το στάδιο εκτέλεσης (execute) αυξάνεται. Αυτό βασικά αυξάνει και τον αριθμό των κύκλων ποινής (penalty cycles) που προκαλούνται όταν υπάρχει λανθασμένη πρόβλεψη διακλάδωσης, κάτι που έχει ως άμεσο αποτέλεσμα την αύξηση του CPI.
- αν το pipeline είναι τόσο βαθύ που χρειάζεται οι λειτουργίες της βασικής αριθμητικής-λογικής μονάδας (arithmetic-logic unit, ALU) να απαιτούν πολλαπλούς κύκλους, τότε η απαραίτητη καθυστέρηση ανάμεσα σε δύο εξαρτημένες μεταξύ τους εντολές θα είναι πολλαπλοί κύκλοι, ακόμα και αν

χρησιμοποιείται hardware προώθησης αποτελέσματος από τη μία εντολή στην άλλη. Αυτό φυσικά αυξάνει το CPI.

- όταν η συχνότητα ρολογιού αυξάνεται με βαθύτερα pipeline κεντρικής μονάδας επεξεργασίας (central processing unit, CPU), τότε η καθυστέρηση μνήμης, σε όρους αριθμού κύκλων ρολογιού, μπορεί να αυξηθεί σημαντικά. Αυτό μπορεί να αυξήσει την μέση καθυστέρηση των λειτουργιών της μνήμης και έτσι να αυξηθεί το συνολικό CPI.

Τέλος αξίζει να σημειωθεί ότι υπάρχει περισσότερο hardware και καθυστέρηση λόγω του pipelining που μπορούν να μειώσουν τα κέρδη στην απόδοση. Γενικά η τεχνική αύξησης της συχνότητας ρολογιού μέσω του βαθύτερου pipelining βοήθησε αρκετά για περισσότερο από μία δεκαετία, δεν είναι όμως καθαρό το κατά πόσο μπορεί να αυξηθεί ακόμα μέχρι η απαιτούμενη πολυπλοκότητα και η κατανάλωση ισχύος να γίνουν απαγορευτικά.

1.4 Παράλληλη επεξεργασία σε επίπεδο εντολής

Η παράλληλη επεξεργασία σε επίπεδο εντολής μπορεί πρακτικά να οριστεί ως η ταυτόχρονη επεξεργασία πολλών εντολών. Οι παραδοσιακοί σειριακοί επεξεργαστές εκτελούν μόνο μία εντολή κάθε φορά. Μία εντολή που προηγείται πρέπει να ολοκληρωθεί προτού αρχίσει η εκτέλεση της επόμενης εντολής. Οι pipelined επεξεργαστές επιτυγχάνουν ένα είδος παράλληλης επεξεργασίας σε επίπεδο εντολής με την επικάλυψη της επεξεργασίας πολλών εντολών. Στο pipeline μπορούν να βρίσκονται ταυτόχρονα (in flight όπως λέγεται) τόσες εντολές όσα είναι και τα στάδια του pipeline. Παραδοσιακοί σειριακοί επεξεργαστές (CISC) απαιτούν κατά μέσο όρο 10 κύκλους μηχανής για την επεξεργασία κάθε εντολής, κάτι που σημαίνει ότι ισχύει CPI = 10. Με τους pipelined επεξεργαστές (RISC), παρότι κάθε εντολή μπορεί να χρειάζεται αρκετούς κύκλους για να ολοκληρωθεί, το μέσο CPI μπορεί να μειωθεί κοντά στη μονάδα, λόγω της επικάλυψης της επεξεργασίας πολλών εντολών στο pipeline, αν σε κάθε κύκλο μπορεί να αρχίσει η επεξεργασία μίας νέας εντολής.

Με τους βαθμωτούς pipelined επεξεργαστές (scalar pipelined processors) υπάρχει πάντα ο περιορισμός της φόρτωσης και έναρξης επεξεργασίας μίας μόνο εντολής στο pipeline σε κάθε κύκλο μηχανής. Εξ' αιτίας αυτού του περιορισμού το CPI δεν μπορεί να μειωθεί κάτω από τη μονάδα, ή διαφορετικά, το καλύτερο δυνατό throughput ενός βαθμωτού επεξεργαστή ισούται με μία εντολή ανά κύκλο (instruction per cycle, IPC). Μία πιο επιθετική μορφή παράλληλης επεξεργασίας σε επίπεδο εντολής είναι η φόρτωση και έναρξη επεξεργασίας πολλών εντολών μέσα σε ένα πλατύτερο pipeline σε ένα κύκλο μηχανής. Έτσι, ενώ ο στόχος για τη δεκαετία του 1980 ήταν το CPI να ισούται με 1 για τους μικροεπεξεργαστές ενός chip, ο στόχος για τη δεκαετία του 1990 ήταν η μείωση του CPI κάτω από τη μονάδα, δηλαδή η επίτευξη ενός throughput για το IPC μεγαλύτερο της μονάδας. Οι επεξεργαστές που είναι ικανοί για IPC μεγαλύτερο της μονάδας ονομάζονται υπερβαθμωτοί επεξεργαστές (superscalar processors).

1.4.1 Από τους scalar στους superscalar επεξεργαστές

Οι scalar επεξεργαστές είναι pipelined επεξεργαστές οι οποίοι είναι σχεδιασμένοι να φορτώνουν και να προωθούν προς επεξεργασία το πολύ μία εντολή σε κάθε κύκλο μηχανής. Οι superscalar επεξεργαστές είναι pipelined επεξεργαστές οι οποίοι όμως είναι σχεδιασμένοι να φορτώνουν και να προωθούν προς επεξεργασία πολλές εντολές σε κάθε κύκλο μηχανής. Στα υποκεφάλαια που ακολουθούν γίνεται

μία παρουσίαση της βάσης και των κινήτρων της εξέλιξης από τις scalar στις superscalar υλοποιήσεις επεξεργαστών.

1.4.1.1 Απόδοση επεξεργαστή

Είδαμε προηγουμένως την σχέση που δίνει την απόδοση ενός επεξεργαστή. Η εξίσωση της Σχέσης 1.1 δίνει για την ακρίβεια το αντίστροφο της απόδοσης ως το γινόμενο του αριθμού εντολών, του μέσου CPI και του χρόνου κύκλου ρολογιού. Η εξίσωση αυτή μπορεί να ξαναγραφεί έτσι ώστε να παρουσιάζει την ίδια την απόδοση και όχι το αντίστροφό της. Η απόδοση παρατηρούμε ότι ισούται τελικά με το γινόμενο του αντίστροφου του αριθμού εντολών, του μέσου IPC (IPC=1/CPI) και της συχνότητας ρολογιού, όπως φαίνεται και στην ακόλουθη σχέση:

$$\text{Απόδοση} = \frac{1}{\text{αριθμός εντολών}} \times \frac{\text{εντολές}}{\text{κύκλο}} \times \frac{1}{\text{χρόνος κύκλου}} = \frac{IPC \times \text{συχνότητα}}{\text{αριθμός εντολών}}$$

(Σχέση 1.2)

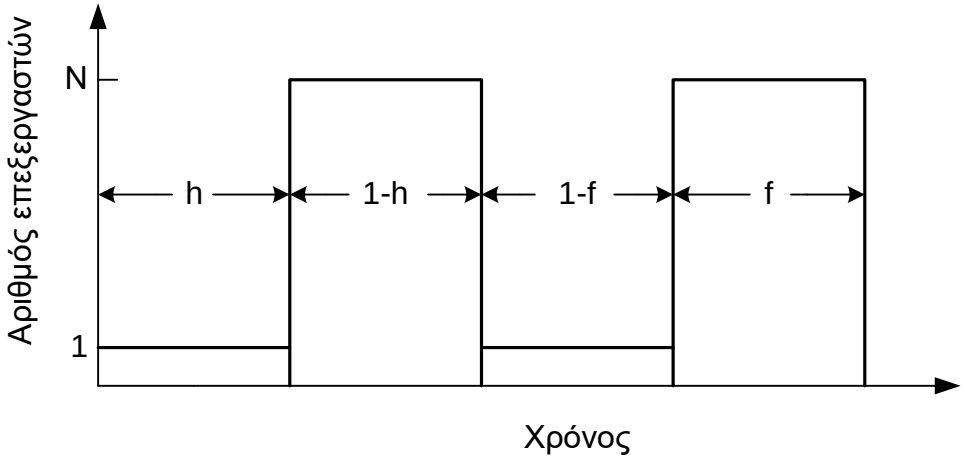
Ο αριθμός εντολών καθορίζεται από τρεις βασικούς παράγοντες:

- την αρχιτεκτονική συνόλου εντολών,
- τον compiler,
- το λειτουργικό σύστημα.

Το μέσο IPC δείχνει το μέσο throughput των εντολών που επιτυγχάνεται από τον επεξεργαστή και είναι ένα πολύ σημαντικό εργαλείο μέτρησης της αποδοτικότητας της μικροαρχιτεκτονικής σχεδίασης. Ιστορικά, γινόταν χρησιμοποίηση του CPI για την μέτρηση του μέσου όρου των κύκλων μηχανής που απαιτούνταν για τη επεξεργασία της κάθε εντολής. Η χρήση του CPI ήταν πολύ δημοφιλής την εποχή των scalar pipelined επεξεργαστών. Καθώς όμως προχωρήσαμε στην εποχή των superscalar pipelined επεξεργαστών, έγινε πιο βολική η χρησιμοποίηση του IPC. Έτσι, ο στόχος, που παλαιότερα ήταν η μείωση του CPI προς τη μονάδα, έγινε η επίτευξη ενός μέσου IPC πάνω από τη μονάδα (όσο περισσότερο γίνεται). Η συχνότητα ρολογιού εξαρτάται όπως έχουμε ήδη αναφέρει από την τεχνολογία παρασκευής των υλικών και από τις κυκλωματικές τεχνικές. Η αύξηση του αριθμού των σταδίων του pipeline μπορεί επίσης να προκαλέσει μεγαλύτερες συχνότητες ρολογιού αλλά όπως ήδη έχει εξηγηθεί στην προσπάθεια να γίνει ένα pipeline φαρδύτερο, για να κρατήσουμε την ίδια συχνότητα ρολογιού πρέπει το pipeline να γίνει και βαθύτερο. Επομένως υπάρχει συμβιβασμός ανάμεσα στο να γίνουν τα pipeline φαρδύτερα και στο να γίνουν βαθύτερα.

1.4.1.2 Απόδοση παράλληλου επεξεργαστή

Για την απόδοση των παράλληλων επεξεργαστών ισχύει ο λεγόμενος «νόμος του Amdahl» (“Amdahl’s law”). Παραδοσιακοί υπερυπολογιστές είναι παράλληλοι επεξεργαστές που εκτελούν βαθμωτές (scalar) και διανυσματικούς (vector) υπολογισμούς. Κατά τον βαθμωτό υπολογισμό χρησιμοποιείται μόνο ένας επεξεργαστής. Κατά τον διανυσματικό υπολογισμό χρησιμοποιούνται όλοι οι επεξεργαστές, οι οποίοι υποθέτουμε ότι είναι N το πλήθος, για την εκτέλεση των απαραίτητων λειτουργιών στα δεδομένα του πίνακα. Ο υπολογισμός που γίνεται από μία τέτοια παράλληλη μηχανή μπορεί να παρασταθεί όπως φαίνεται στο Σχήμα 1.4, που ακολουθεί, όπου το N είναι ο αριθμός των επεξεργαστών και h είναι το χρονικό διάστημα που η μηχανή εκτελεί τον βαθμωτό υπολογισμό. Φυσικά το 1-h είναι το χρονικό διάστημα στο οποίο η μηχανή εκτελεί τον διανυσματικό υπολογισμό.



Σχήμα 1.4 Βαθμωτός και διανυσματικός υπολογισμός σε έναν παραδοσιακό υπερυπολογιστή.

Μία μορφή του νόμου του Amdahl δηλώνει ότι η αποδοτικότητα Ε της παράλληλης μηχανής υπολογίζεται από τη συνολική χρησιμοποίηση των Ν επεξεργαστών ή από το χρονικό διάστημα στο οποίο οι Ν επεξεργαστές είναι απασχολημένοι, δηλαδή εκτελούν έργο. Η αποδοτικότητα Ε ισούται με:

$$E = \frac{h + N \times (1-h)}{N} = \frac{h + N - Nh}{N} = 1 - h \times \left(1 - \frac{1}{N}\right) \quad \text{(Σχέση 1.3)}$$

Καθώς ο αριθμός Ν των επεξεργαστών γίνεται πολύ μεγάλος, η αποδοτικότητα Ε πλησιάζει το 1-h, το οποίο είναι το τμήμα του χρόνου όπου η μηχανή εκτελεί τον διανυσματικό υπολογισμό. Όσο το Ν γίνεται μεγάλο, ο χρόνος που δαπανάται για τον διανυσματικό υπολογισμό γίνεται μικρότερος και πλησιάζει στο μηδέν. Επομένως, όταν το Ν γίνεται πολύ μεγάλο, η αποδοτικότητα Ε τείνει στο μηδέν. Αυτό σημαίνει ότι σχεδόν όλος ο χρόνος υπολογισμού της μηχανής δαπανάται για τον βαθμωτό υπολογισμό και επομένως το Ν παίζει πλέον μικρότερο ρόλο στη μείωση του συνολικού χρόνου εκτέλεσης.

Μία άλλη μορφή του νόμου βασίζεται στο μέγεθος του έργου που επιτελείται από τη μηχανή κατά τη διάρκεια του διανυσματικού υπολογισμού, ή στην ικανότητα διανυσματοποίησης (vectorizability) του προγράμματος. Όπως φαίνεται και στο Σχήμα 1.4, το f αντιπροσωπεύει το τμήμα του προγράμματος που μπορεί να παραλληλιστεί και να εκτελεστεί σε μορφή διανυσματικού υπολογισμού. Έτσι, το 1-f αντιπροσωπεύει το τμήμα του προγράμματος που πρέπει να εκτελεστεί σειριακά. Αν το Τ είναι ο συνολικός χρόνος που απαιτείται για να εκτελεστεί το πρόγραμμα, τότε η σχετική επιτάχυνση (speedup) S μπορεί να υπολογιστεί από τον εξής τύπο:

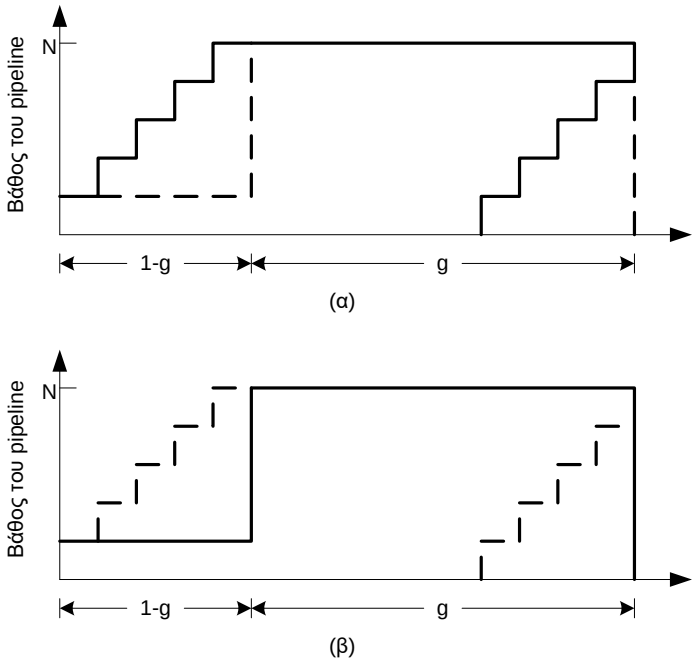
$$S = \frac{1}{T} = \frac{1}{(1-f) + (f/N)} \quad \text{(Σχέση 1.4)}$$

όπου το Τ είναι το άθροισμα του (1-f), δηλαδή του χρόνου που απαιτείται για την εκτέλεση του σειριακού τμήματος, και του f/N, δηλαδή του χρόνου που απαιτείται για την εκτέλεση του παράλληλου τμήματος του προγράμματος. Καθώς το Ν γίνεται αρκετά μεγάλο, ο δεύτερος όρος αυτού του αθροίσματος τείνει στο μηδέν, και ο συνολικός χρόνος εκτέλεσης εξαρτάται από τον χρόνο που απαιτείται για την εκτέλεση του σειριακού τμήματος. Αυτό συχνά αναφέρεται ως σειριακός στενωπός (sequential bottleneck), κάτι που σημαίνει ότι ο χρόνος που δαπανάται για την σειριακή εκτέλεση γίνεται ένα όριο της βελτίωσης της συνολικής απόδοσης μέσω της χρήσης του παραλληλισμού.

Η αποδοτικότητα ενός παράλληλου επεξεργαστή πέφτει πολύ απότομα καθώς ο αριθμός των επεξεργαστών αυξάνεται. Έτσι, καθώς το τμήμα του προγράμματος που μπορεί να παραλληλιστεί πέφτει λίγο κάτω από το 100%, το ποσοστό πτώσης της αποδοτικότητας αυξάνεται. Παρόμοια, το συνολικό speedup πέφτει πολύ απότομα όταν το f, η ικανότητα διανυσματοποίησης του προγράμματος, πέφτει πολύ λίγο κάτω από το 100%. Επομένως, η βελτίωση της συνολικής απόδοσης είναι πολύ ευαίσθητη στην ικανότητα διανυσματοποίησης του προγράμματος. Θέτοντάς το διαφορετικά μπορούμε να πούμε ότι το συνολικό speedup λόγω παράλληλης επεξεργασίας εξαρτάται ισχυρά από το σειριακό κομμάτι του προγράμματος καθώς ο παραλληλισμός της μηχανής αυξάνεται.

1.4.1.3 Απόδοση pipelined επεξεργαστή

Ένα τυπικό προφίλ επεξεργασίας ενός pipelined επεξεργαστή φαίνεται στο Σχήμα 1.5(α). Η παράμετρος παραλληλισμού της μηχανής N είναι τώρα το βάθος του pipeline, δηλαδή ο αριθμός των σταδίων του pipeline. Υπάρχουν τρεις φάσεις σε αυτό το προφίλ εκτέλεσης. Η πρώτη φάση είναι η φάση γεμίσματος του pipeline, κατά τη διάρκεια της οποίας η πρώτη ακολουθία των N το πλήθος εντολών εισέρχονται στο pipeline. Η δεύτερη φάση είναι η φάση του γεμάτου pipeline, κατά τη διάρκεια της οποίας το pipeline είναι γεμάτο και η οποία αντιπροσωπεύει την σταθερή κατάσταση του pipeline. Αυτή βέβαια η φάση βέβαια συμβαίνει όταν δεν υπάρχουν διακοπές στο pipeline, και επομένως αντιπροσωπεύει το τέλειο προφίλ εκτέλεσης στο pipeline. Η τρίτη φάση είναι η φάση αδειάσματος (draining) του pipelined, κατά τη διάρκεια της οποίας καμία νέα εντολή δεν εισέρχεται στο pipeline και το pipeline τελειώνει την εκτέλεση των εντολών που βρίσκονται ήδη στα στάδια του pipeline.



Σχήμα 1.5 Ιδεατό προφίλ pipelined εκτέλεσης: (α) Πραγματικό (β) Μοντελοποιημένο.

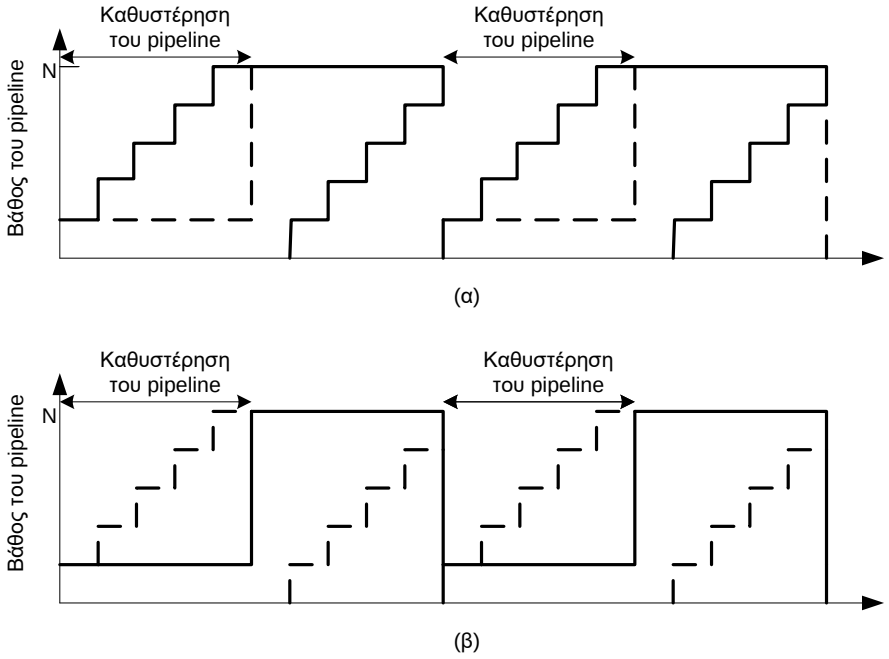
Για σκοπούς μοντελοποίησης μπορούμε να μετατρέψουμε το προφίλ εκτέλεσης του Σχήματος 1.5(α) στο προφίλ εκτέλεσης του Σχήματος 1.5(β) μετακινώντας μερική από το έργο που γίνεται κατά τη διάρκεια της φάσης

γεμίσματος του pipeline στη φάση αδειάσματος τους pipeline. Η συνολική ποσότητα έργου παραμένει η ίδια, αφού τα εμβαδά των δύο προφίλ είναι ίσα. Ο αριθμός των σταδίων του pipeline ισούται με N, το χρονικό διάστημα στο οποίο και τα N στάδια του pipeline χρησιμοποιούνται είναι ίσο με g, ενώ με 1-g ισούται το χρονικό διάστημα στο οποίο χρησιμοποιείται μόνο ένα στάδιο του pipeline. Επομένως, μπορεί να θεωρηθεί ότι με 1-g ισούται το χρονικό διάστημα κατά το οποίο μόνο μία εντολή κινείται στο pipeline και επομένως δεν υπάρχει επικάλυψη εντολών.

Αντίθετα με το ιδεατό προφίλ της εκτέλεσης στο pipeline, το ρεαλιστικό προφίλ πρέπει να λάβει υπόψη του και τους χαμένους κύκλους (stalling cycles). Αυτό μπορεί να γίνει όπως φαίνεται στο Σχήμα 1.6(α). Αντί να παραμένει στη φάση του γεμάτου pipeline για ολόκληρη την διάρκεια της εκτέλεσης, η σταθερή κατάσταση διακόπτεται από καθυστερήσεις του pipeline (pipeline stalls). Κάθε καθυστέρηση προκαλεί μία νέα φάση αδειάσματος του pipeline και μία νέα φάση γεμίσματος του pipeline, όπως φαίνεται στο Σχήμα 1.6(α), λόγω της διακοπής της φάσης γεμάτου pipeline. Παρόμοιες μετατροπές με πριν μπορούν να γίνουν και σε αυτό το προφίλ και οι οποίες έχουν ως αποτέλεσμα το τροποποιημένο προφίλ του Σχήματος 1.6(β). Τώρα βέβαια το τροποποιημένο προφίλ έχει την ίδια ακριβώς μορφή με το προφίλ εκτέλεσης των παράλληλων επεξεργαστών του Σχήματος 1.4.

Λόγω της ομοιότητας των δύο αυτών προφίλ μπορούμε πλέον να δανειστούμε το μοντέλο απόδοσης των παράλληλων επεξεργαστών και να το χρησιμοποιήσουμε για τους pipelined επεξεργαστές. Αντί για τον αριθμό των επεξεργαστών, το N τώρα είναι ο αριθμός των σταδίων του pipeline ή το μέγιστο δυνατό speedup. Η παράμετρος g γίνεται πλέον το τμήμα του χρόνου στο οποίο το pipeline είναι γεμάτο και η παράμετρος 1-g το τμήμα του χρόνου στο οποίο το pipeline καθυστερεί. Το speedup ισούται πλέον με:

$$S = \frac{1}{(1-g)+(g/N)} \quad \text{(Σχέση 1.5).}$$



Σχήμα 1.6 Ρεαλιστικό προφίλ pipelined εκτέλεσης: (α) Πραγματικό (β) Μοντελοποιημένο.

Να σημειωθεί εδώ ότι το g, δηλαδή το χρονικό διάστημα στο οποίο το pipeline είναι γεμάτο, είναι ανάλογο με το f, δηλαδή το διάστημα στο οποίο εκτελείται ο παράλληλος υπολογισμός και από τους N επεξεργαστές στο μοντέλο παράλληλης επεξεργασίας. Έτσι, ο νόμος του Amdahl ισχύει αναλογικά και για τους pipelined επεξεργαστές. Καθώς το g πέφτει ελάχιστα κάτω από το 100%, το speedup ή η απόδοση ενός pipelined επεξεργαστή πέφτει πολύ απότομα. Με άλλα λόγια, το πραγματικό κέρδος απόδοσης που μπορεί να επιτευχθεί μέσω του pipelining μπορεί να υποβαθμιστεί ισχυρά από λίγους μόνο χαμένους κύκλους ρολογιού. Επομένως, οι χαμένοι κύκλοι (stall cycles) αποτελούν τον σειριακό στενωπό του pipelined επεξεργαστή.

Η Σχέση 1.5 αποτελεί ουσιαστικά ένα απλό μοντέλο απόδοσης για τους pipelined επεξεργαστές που είναι βασισμένο στον νόμο του Amdahl για τους παράλληλους επεξεργαστές. Σε αυτό το μοντέλο θεωρείται ότι όποτε το pipeline καθυστερεί, υπάρχει μόνο μία εντολή στο pipeline, ή διαφορετικά το pipeline μετατρέπεται σε έναν σειριακό μη-pipelined επεξεργαστή. Αυτό συμβαίνει γιατί θεωρείται ότι όταν το pipeline καθυστερεί δεν επιτρέπεται καμία επικάλυψη εντολών. Κάτι τέτοιο βέβαια είναι ισοδύναμο με την καθυστέρηση του pipeline για N κύκλους έτσι ώστε να επιτραπεί η πλήρης ολοκλήρωση της εντολής που προκάλεσε την καθυστέρηση. Είναι γνωστό, παρόλα αυτά, ότι με έξυπνη σχεδίαση του pipeline, όπως για παράδειγμα με τη χρήση προώθησης δεδομένων που, με σκοπό την αντιμετώπιση των κινδύνων που προκαλούν καθυστερήσεις στο pipeline, ο αριθμός των κύκλων ποινής που προκλήθηκαν δεν είναι απαραίτητα ίσος με N αλλά πιθανότατα μικρότερος από N. Με βάση αυτή την παρατήρηση, μία τροποποίηση στο μοντέλο της Σχέσης 1.5 είναι δυνατή. Έτσι προκύπτει η εξής σχέση:

$$S = \frac{1}{\frac{g_1}{1} + \frac{g_2}{2} + \ldots + \frac{g_N}{N}} \quad \text{(Σχέση 1.6)}$$

Η Σχέση 1.6 είναι μία γενίκευση της Σχέσης 1.5 και δίνει ένα ακριβέστερο μοντέλο για την απόδοση των pipelined επεξεργαστών. Σε αυτό το μοντέλο, το g_i αναπαριστά το χρονικό διάστημα στο οποίο υπάρχουν i το πλήθος εντολές μέσα στο pipeline. Με άλλα λόγια, το g_i αναπαριστά το χρονικό διάστημα στο οποίο το pipeline καθυστερεί για (N-i) κύκλους ποινής. Φυσικά, το g_N είναι το χρονικό διάστημα στο οποίο το pipeline είναι γεμάτο.

1.4.1.4 Η superscalar πρόταση

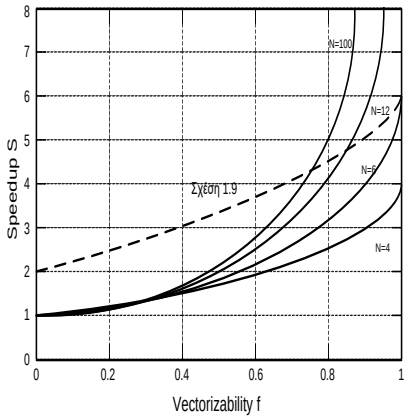
Η Σχέση 1.4 όπως είπαμε αποτελεί τον νόμο του Amdahl που μοντελοποιεί την απόδοση ενός παράλληλου επεξεργαστή. Εδώ υπενθυμίζουμε τον τύπο:

$$S = \frac{1}{(1-f)+(f/N)} \quad \text{(Σχέση 1.7)}$$

Αυτό το μοντέλο δίνει την απόδοση ή την επιτάχυνση ενός παράλληλου συστήματος σε σχέση με ένα μη-παράλληλο σύστημα. Ο παραλληλισμός μηχανής (machine parallelism) μετριέται με το N, δηλαδή τον αριθμό των επεξεργαστών σε μία μηχανή, και αντικατοπτρίζει το μέγιστο αριθμό εργασιών που μπορούν να εκτελεστούν ταυτόχρονα από ένα σύστημα. Η παράμετρος f, από την άλλη πλευρά, είναι η ικανότητα διανυσματοποίησης (vectorizability) του προγράμματος η οποία αναπαριστά τον παραλληλισμό προγράμματος (program parallelism). Η μορφή αυτού του μοντέλου επηρεάζεται από τους παραδοσιακούς υπερυπολογιστές που περιέχουν μία scalar μονάδα και μία διανυσματική μονάδα. Η διανυσματική μονάδα, που

αποτελείται από N επεξεργαστές, εκτελεί το διανυσματοποιημένο τμήμα του προγράμματος εκτελώντας N εργασίες κάθε φορά. Το μη-διανυσματοποιημένο τμήμα του προγράμματος εκτελείται από την scalar μονάδα με ένα σειριακό τρόπο. Έχουμε ήδη παρατηρήσει την αρνητική επίπτωση του μη-διανυσματοποιημένου τμήματος στην συνολική απόδοση.

Η υπόθεση ότι το μη-διανυσματοποιημένο τμήμα του προγράμματος πρέπει να εκτελεστεί σειριακά είναι αρκετά απαισιόδοξη και όχι απαραίτητη. Αν μπορεί να επιτευχθεί κάποιο, έστω και μικρό, επίπεδο παραλληλισμού για αυτό το κομμάτι του προγράμματος, τότε η πολλή μεγάλη αρνητική επίδραση του σειριακού στενωπού μπορεί να μειωθεί σημαντικά. Στο Σχήμα 1.7 αυτό φαίνεται καθαρά.



Σχήμα 1.7 Εξομάλυνση του σειριακού στενωπού με παραλληλισμό σε επίπεδο εντολής για μη-διανυσματοποιήσιμο κώδικα.

Αυτή η γραφική παράσταση, που πάρθηκε από μία τεχνική αναφορά της IBM, αναπαριστά το speedup ως μία συνάρτηση του f για αρκετές τιμές του N. Ας πάρουμε το παράδειγμα της περίπτωσης όπου N = 6. Το speedup τότε ισούται με:

$$S = \frac{1}{(1-f)+(f/6)} \quad \text{(Σχέση 1.8)}$$

Εξετάζοντας την καμπύλη της Σχέσης 1.8 στο Σχήμα 1.7 παρατηρούμε ότι το speedup ισούται με 6 αν το f ισούται με 100%. Καθώς το f πέφτει κάτω από το 100%, το speedup πέφτει πολύ απότομα. Όταν το f ισούται με 0% το speedup γίνεται ίσο με τη μονάδα, δηλαδή δεν υπάρχει καμία επιτάχυνση. Για μεγάλες τιμές του N, το ποσοστό πτώσης του speedup γίνεται αισθητά χειρότερο και καθώς το f πλησιάζει στο 0%, όλα τα speedup πλησιάζουν τη μονάδα, ανεξάρτητα από την τιμή του N. Τώρα ας υποθέσουμε ότι μπορεί να επιτευχθεί ο ελάχιστος βαθμός παραλληλισμού, δηλαδή βαθμός παραλληλισμού ίσος με 2, για το μη-διανυσματοποιημένο κομμάτι του προγράμματος. Το speedup γίνεται πλέον ίσο με:

$$S = \frac{1}{\frac{(1-f)}{2}+\frac{f}{6}} \quad \text{(Σχέση 1.9)}$$

Εξετάζοντας την καμπύλη της Σχέσης 1.9 στο Σχήμα 1.7 παρατηρούμε ότι το speedup ισούται πάλι με 6 όταν το f ισούται με 100%, αλλά πέφτει πολύ πιο αργά από την καμπύλη της Σχέσης 1.8, όταν το f πέφτει κάτω από το 100%. Μάλιστα, η καμπύλη τέμνει την καμπύλη της Σχέσης 1.8 για N = 100 όταν το f είναι περίπου ίσο με 75%. Αυτό σημαίνει ότι για f μικρότερο από 75% συμφέρει περισσότερο να υπάρχει ένα σύστημα με μέγιστο βαθμό παραλληλισμού ίσο με μόλις 6, δηλαδή με N = 6, αλλά με βαθμό παραλληλισμού για το μη-διανυσματοποιημένο τμήμα ίσο με

2, σε σχέση με ένα σύστημα με μέγιστο βαθμό παραλληλισμού ίσο με N = 100 με σειριακή εκτέλεση για το μη-διανυσματοποιημένο τμήμα. Η δυνατότητα διανυσματοποίησης ενός προγράμματος f είναι μία πολύπλοκη συνάρτηση που εμπεριέχει τον αλγόριθμο εφαρμογής, τη γλώσσα προγραμματισμού, τον compiler και την αρχιτεκτονική. Πάντως, πρέπει να σημειωθεί ότι τα περισσότερα προγράμματα γενικού σκοπού έχουν f μικρότερα του 75% και αρκετά από αυτά έχουν σημαντικά μικρότερα f.

Ένα βασικό κίνητρο για τη σχεδίαση των superscalar επεξεργαστών είναι η δημιουργία επεξεργαστών γενικού σκοπού οι οποίοι μπορούν να επιτύχουν ένα επίπεδο παραλληλισμού για μία μεγάλη γκάμα προγραμμάτων εφαρμογών. Ο στόχος είναι να σιγουρευτεί η επίτευξη κάποιου βαθμού παραλληλισμού σε επίπεδο εντολής για όλα τα τμήματα του προγράμματος, έτσι ώστε να αντιμετωπιστεί η επίδραση του σειριακού στενωπού.

1.4.2 Όρια του παραλληλισμού επιπέδου εντολής

Στην Σχέση 1.9 μπορεί να επιτευχθεί παραλληλισμός βαθμού 6 για το f κομμάτι του προγράμματος και βαθμού 2 για το υπόλοιπο 1-f κομμάτι. Το speedup S μπορεί να θεωρηθεί ως ο μέσος βαθμός παραλληλισμού που μπορεί να επιτευχθεί για όλο το πρόγραμμα. Για παράδειγμα, αν η παράμετρος f είναι 50% και ο μέγιστος παραλληλισμός (peak) N είναι 6, τότε το speedup ή ο μέσος βαθμός παραλληλισμού ισούται με:

$$S = \frac{1}{\frac{(1-f)}{2} + \frac{f}{6}} = \frac{1}{\frac{0.5}{2} + \frac{0.5}{6}} = 3 \quad \text{(Σχέση 1.10)}$$

Η Σχέση 1.10 μας οδηγεί στο συμπέρασμα ότι ένας συνολικός ή μέσος βαθμός παραλληλισμού βαθμού 3 επιτυγχάνεται για ολόκληρο το πρόγραμμα. Εφαρμόζοντας αυτό το αποτέλεσμα στο επίπεδο εντολής, παρατηρούμε ότι η Σχέση 1.10 δείχνει ότι κατά μέσο όρο μπορούν να εκτελούνται ταυτόχρονα 3 εντολές.

Ο παραλληλισμός επιπέδου εντολής μπορεί πρακτικά να οριστεί ως ο μέσος βαθμός παραλληλισμού (μετρημένος σε αριθμός εντολών) που μπορεί να επιτευχθεί από την ταυτόχρονη εκτέλεση πολλών εντολών. Πιθανά όρια του ILP (Instruction-Level Parallelism) έχουν μελετηθεί για περίπου τρεις δεκαετίες. Αρκετές πειραματικές μελέτες έχουν γίνει όλα αυτά τα χρόνια, τα αποτελέσματα των οποίων ποικίλουν για την τιμή του ILP. Πολλές από τις μελέτες του ορίου αυτού βασίστηκαν σε ιδεατές υποθέσεις και παραδοχές. Η πραγματική πρόκληση είναι η επίτευξη αυτών των ILP σε πραγματικές σχεδιάσεις.

1.4.2.1 Ο στενωπός του Flynn (Flynn's Bottleneck)

Μία από τις παλαιότερες μελέτες που έγινε στο Πανεπιστήμιο του Stanford από τους Tjaden και Flynn το 1970 είχε ως τελικό συμπέρασμα ότι το ILP για τα περισσότερα προγράμματα είναι μικρότερο από 2. Αυτό το όριο αναφέρεται ανεπίσημα ως «ο στενωπός του Flynn» ("Flynn's Bottleneck"). Αυτή η μελέτη εστίασε στον παραλληλισμό επιπέδου εντολής που μπορεί να επιτευχθεί στα όρια βασικών μπλοκ εντολών. Από τη στιγμή που το πέρασμα ανάμεσα από αυτά τα όρια (δηλαδή από το ένα κομμάτι κώδικα στο άλλο) εμπεριέχει εξαρτήσεις ελέγχου, οι οποίες μπορεί να είναι εξαρτήσεις σε δεδομένα χρόνου εκτέλεσης, θεωρείται ότι τα βασικά αυτά μπλοκ πρέπει να εκτελεστούν σειριακά. Λόγω του μικρού μεγέθους των

περισσότερων βασικών μπλοκ, συνήθως ο βαθμός παραλληλισμού είναι μικρότερος από 2. Το αποτέλεσμα αυτό έχει διαπιστωθεί και από αρκετές άλλες μελέτες.

Μία τέτοια μελέτη έγινε το 1972 από τους Riseman και Foster. Παρόλα αυτά, οι δύο ερευνητές προχώρησαν τη μελέτη τους και εξέτασαν την περίπτωση στην οποία όλες οι εξαρτήσεις ελέγχου θα μπορούσαν με κάποιο τρόπο να επιλυθούν. Σε αυτή τη μελέτη αναφέρονται ποικίλοι βαθμοί παραλληλισμού που μπορούν να επιτευχθούν αν ποικίλοι αριθμοί εξαρτήσεων ελέγχου μπορούν να αντιμετωπιστούν. Αν ο αριθμός των εξαρτήσεων ελέγχου που αντιμετωπίζονται είναι απεριόριστος, τότε η τιμή του ILP είναι κοντά στο 51. Αυτή η μελέτη υπογραμμίζει την πολύ σημαντική επιρροή των εξαρτήσεων ελέγχου στα όρια του ILP.

1.4.2.2 Η αισιοδοξία του Fisher (Fisher's Optimism)

Στο άλλο άκρο κυμάνθηκε μία μελέτη που έγινε από τους Νικολάου και Fisher το 1984 στο Πανεπιστήμιο Yale. Αυτή η μελέτη υπαινίσσεται σχεδόν απεριόριστα μεγέθη του ILP σε πολλά προγράμματα. Τα προγράμματα ελέγχου (benchmarks) που χρησιμοποιήθηκαν σε αυτή τη μελέτη είναι κυρίως αριθμητικά, και μερικοί από τους παραλληλισμούς που μετρήθηκαν οφείλονται στους παραλληλισμού δεδομένων εξαιτίας των συνόλων δεδομένων τύπου πίνακα. Επίσης θεωρήθηκε μία ιδεατή μηχανή ικανή για ταυτόχρονη εκτέλεση πολλών εντολών. Παρότι στη μελέτη αυτή έγιναν πολλές ιδεατές παραδοχές, παρουσιάζεται σε αυτή μία αισιόδοξη ματιά στο θέμα του μεγέθους του ορίου του ILP. Ανεπίσημα αναφερόμαστε στο όριο αυτό για το ILP, το οποίο κυμαίνεται γύρω στο 90, ως «η αισιοδοξία του Fisher ("Fisher's Optimism").

Αρχικά αυτή η αισιοδοξία αντιμετωπίστηκε με μεγάλο σκεπτικισμό, όμως μία σειρά από γεγονότα υποστήριξαν κατά κάποιο τρόπο αυτή την μελέτη. Αρχικά δημιουργήθηκε ένα πρωτότυπο μοντέλο μηχανής, που ονομάστηκε VLIW (very long instruction word) επεξεργαστής, μαζί με έναν compiler που υποστήριζε τον επεξεργαστή αυτό. Στη συνέχεια η εταιρία Multiflow Inc. κατασκεύασε μία ρεαλιστική VLIW μηχανή, η οποία κατέληξε στον Multiflow TRACE υπολογιστή. Οι μηχανές TRACE συνοδεύονταν από έναν πανίσχυρο VLIW compiler που εφάρμοζε καταγραφή ιχνών για την επίτευξη παραλληλισμού επιπέδου εντολής. Το μοντέλο ήταν επιτυχημένο, και κατασκευάστηκαν πάνω από 100 μηχανές. Το πιο σημαντικό όμως ήταν ότι επιτεύχθηκε μέσο IPC μεγαλύτερο της μονάδας. Παρότι τα πραγματικά επίπεδα ILP που επιτεύχθηκαν από τις μηχανές αυτές ήταν πολύ κάτω από τα όρια της έρευνας, επιβεβαίωσαν τους ισχυρισμούς ότι υπάρχουν σημαντικά μεγέθη ILP που μπορούν να επιτευχθούν πάνω από το προηγούμενο αποδεκτό όριο του 2.

2 ΤΕΧΝΙΚΕΣ ΥΠΕΡΒΑΘΜΩΤΩΝ ΕΠΕΞΕΡΓΑΣΤΩΝ

Οι βαθμωτοί (scalar) επεξεργαστές έχουν ορισμένα μειονεκτήματα έναντι των υπερβαθμωτών (superscalar) επεξεργαστών. Οι τεχνικές σωληνώσεως που χρησιμοποιούνται στους scalar επεξεργαστές έχουν τρεις σημαντικούς περιορισμούς. Αυτοί οι τρεις περιορισμοί είναι οι ακόλουθοι:

- Το μέγιστο throughput για το scalar pipeline (βαθμωτή σωλήνωση) έχει ως άνω όριο την μία εντολή ανά κύκλο ρολογιού. Το throughput αποτελεί μέγεθος μέτρησης της απόδοσης ενός επεξεργαστή και ουσιαστικά μετράει το πλήθος των εντολών που μπορούν να ολοκληρωθούν σε ένα κύκλο ρολογιού του επεξεργαστή.
- Η ενοποίηση όλων των τύπων εντολών (αριθμητικών, load/store, branch) σε μία σωλήνωση μπορεί να οδηγήσει σε μία μη αποδοτική σχεδίαση του επεξεργαστή.
- Η καθυστέρηση (stall) του άκαμπτου pipeline που χαρακτηρίζει τους scalar επεξεργαστές προκαλεί χαμένους κύκλους ρολογιού που δεν είναι απαραίτητοι (δεν είναι αναγκαίο να χαθούν) και μειώνουν την απόδοση.

Οι υπερβαθμωτές σωληνώσεις(superscalar pipelines) είναι παράλληλα pipelines αντί για scalar pipelines. Αυτό σημαίνει ότι έχουν την δυνατότητα να εκκινούν την επεξεργασία πολλαπλών εντολών σε κάθε κύκλο μηχανής. Επιπλέον τα superscalar pipelines είναι τροποποιημένα pipelines αφού παρέχουν πολλαπλές και ετερογενείς μονάδες επεξεργασίας στα στάδια επεξεργασίας τους. Τέλος τα superscalar pipelines μπορούν να υλοποιηθούν ως δυναμικά pipelines με στόχο να επιτευχθεί η καλύτερη δυνατή απόδοση χωρίς την απαίτηση αναδιάταξης (reordering) των εντολών από τον compiler. Τώρα θα περάσουμε στην περαιτέρω ανάλυση αυτών των τριών χαρακτηριστικών.

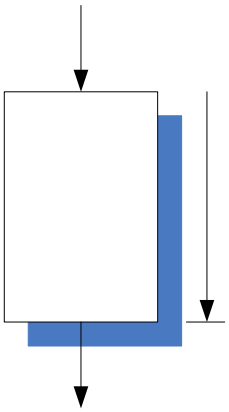
2.1 Παράλληλα pipelines

Ο βαθμός παραλληλισμού μίας μηχανής μπορεί να μετρηθεί από τον μέγιστο αριθμό των εντολών που μπορούν να βρίσκονται σε εξέλιξη (επεξεργασία) σε κάθε χρονική στιγμή. Ένα scalar pipeline κ σταδίων μπορεί να έχει κ εντολές ταυτόχρονα στη μηχανή (η κάθε εντολή θα βρίσκεται σε διαφορετικό στάδιο επεξεργασίας) και μπορεί να επιτύχει ένα συντελεστή επιτάχυνσης κ έναντι μίας μη pipelined μηχανής. Η ίδια επιτάχυνση (speedup) μπορεί να επιτευχθεί με κ αντίγραφα μίας μη pipelined μηχανής που επεξεργάζεται παράλληλα κ εντολές. Για αυτές τις δύο μορφές παραλληλισμού μηχανής χρησιμοποιούνται οι όροι χρονικός παραλληλισμός μηχανής (temporal machine parallelism) και χωρικός παραλληλισμός μηχανής (spatial machine parallelism) αντίστοιχα.

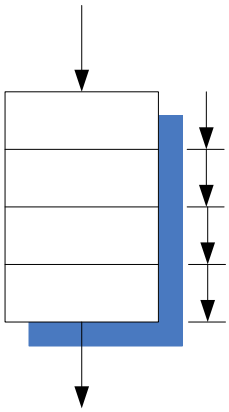
Για τα superscalar pipelines η επιτάχυνση μετριέται συνήθως σε σχέση με ένα scalar pipeline και αρχικά καθορίζεται από το πλάτος (width) του παράλληλου pipeline (παράλληλο pipeline = superscalar pipeline). Ένα παράλληλο pipeline με πλάτος s μπορεί να επεξεργαστεί ταυτόχρονα μέχρι και s εντολές σε κάθε ένα από τα στάδιά του, γεγονός το οποίο μπορεί να οδηγήσει σε πιθανό συντελεστή επιτάχυνσης s έναντι ενός scalar pipeline.

Όπως γίνεται εύκολα αντιληπτό οι απαιτήσεις σε hardware ενός παράλληλου pipeline είναι σημαντικά μεγαλύτερες από τις απαιτήσεις ενός scalar pipeline.

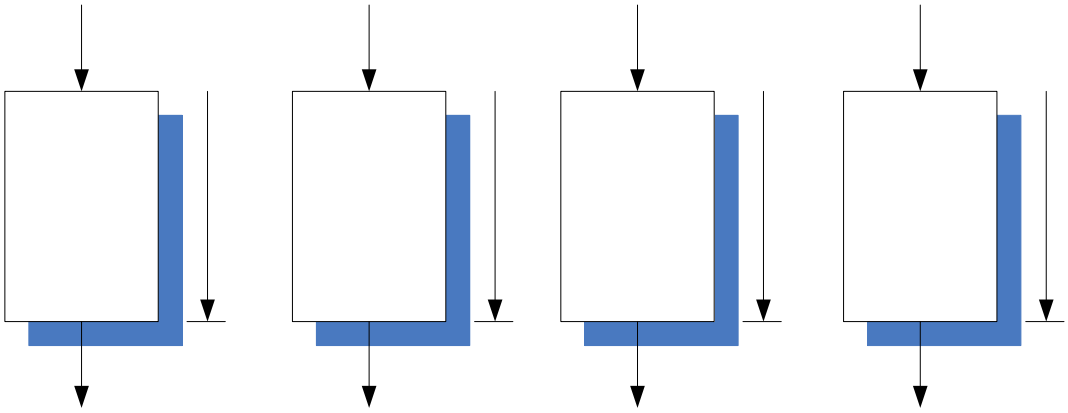
Τα διάφορα είδη pipeline φαίνονται στο ακόλουθο σχήμα:



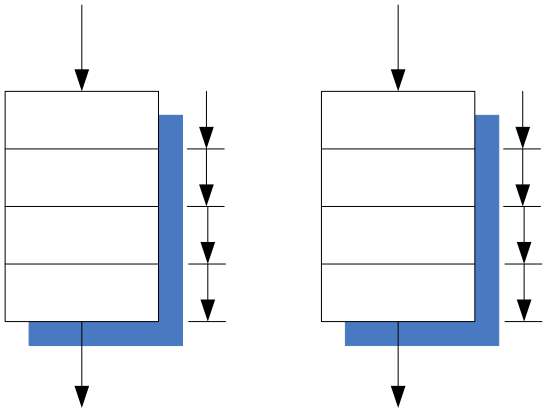
(α) Μη παραλληλισμός



(β) Χρονικός παραλληλισμός



(γ) Χωρικός παραλληλισμός



(δ) Παράλληλο pipeline

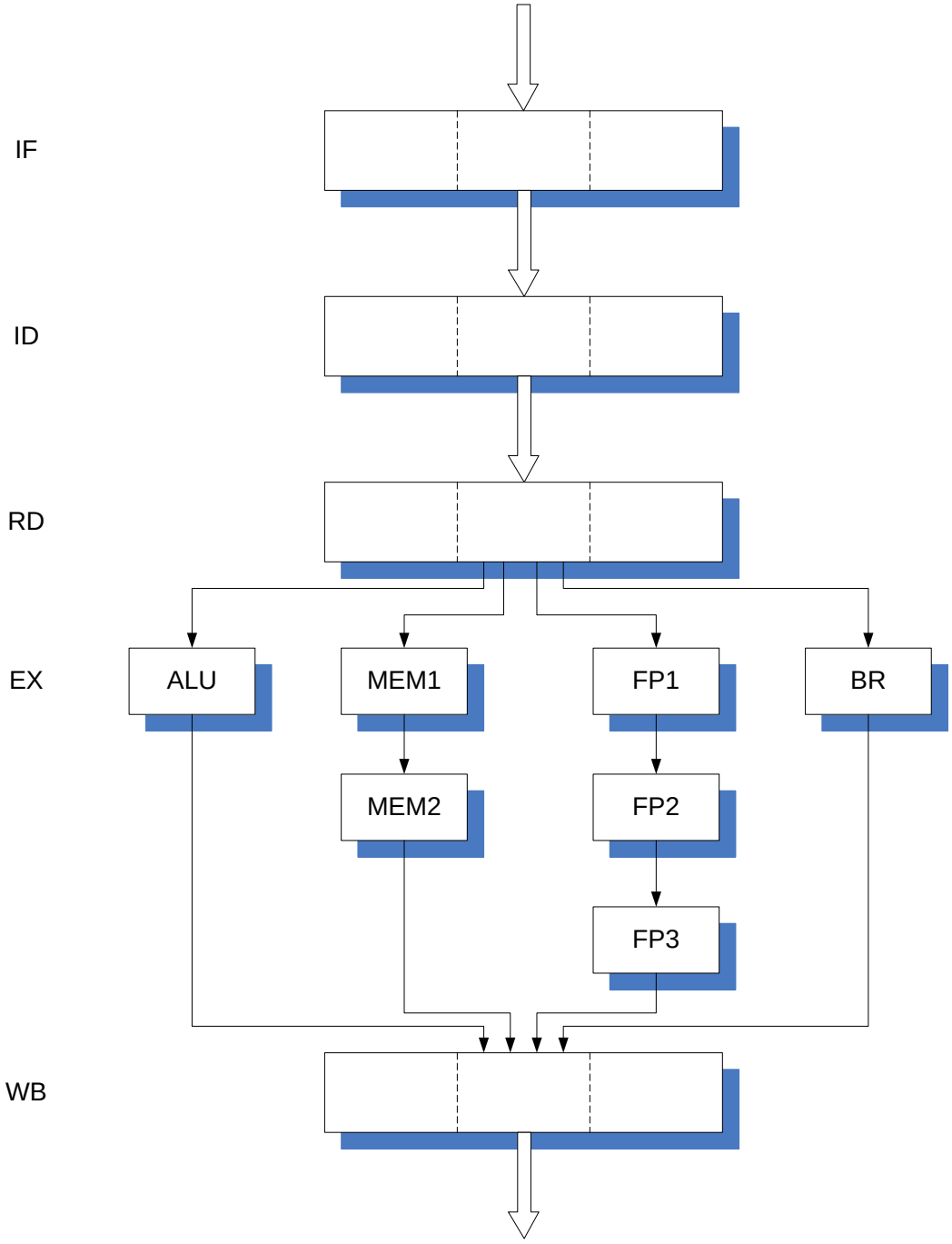
Σχήμα 2.1 Είδη παραλληλισμού.

2.2 Τροποποιημένα pipelines

Για ένα scalar pipeline όλες οι απαιτήσεις για την εκτέλεση όλων των ειδών εντολών πρέπει να ενοποιηθούν σε ένα ενιαίο pipeline. Το pipeline που προκύπτει είναι ιδιαίτερα ανεπαρκές και αναποτελεσματικό. Κάθε τύπος εντολών απαιτεί μόνο ένα μέρος του συνόλου των σταδίων εκτέλεσης αλλά πρέπει να διατρέξει όλα τα στάδια εκτέλεσης, δηλαδή ακόμα και αυτά που δεν είναι χρήσιμα. Κάθε εντολή μένει ανενεργή (ή αλλιώς «οκνηρή») (idle) καθώς διαπερνά τα μη απαραίτητα για την εκτέλεσή της στάδια εκτέλεσης και έτσι προκαλεί σημαντική δυναμική εξωτερική τμηματοποίηση (fragmentation). Ο χρόνος εκτέλεσης (execution latency) για όλα τα είδη εντολών είναι ίσος με το συνολικό αριθμό των σταδίων εκτέλεσης. Αυτό μπορεί να προκαλέσει ανεπιθύμητη καθυστέρηση συρόμενων εντολών (η μία εντολή έρχεται πίσω από την άλλη) και/ή να απαιτήσει πρόσθετα μονοπάτια μεταβίβασης (δηλαδή μεταβίβασης δεδομένων ανάμεσα στα στάδια) (forwarding paths).

Η αναποτελεσματικότητα λόγω της ενοποίησης σε ένα ενιαίο pipeline αντιμετωπίζεται στα παράλληλα pipelines με τη χρήση πολλαπλών διαφορετικών λειτουργικών μονάδων (functional units) στο στάδιο εκτέλεσης (στο execution stage(s)). Αντί για τη χρησιμοποίηση s πανομοιότυπων σωλήνων (pipes) σε ένα παράλληλο pipeline πλάτους s στο τμήμα εκτέλεσης του pipeline, μπορούν να χρησιμοποιηθούν τροποποιημένα execution pipes. Αυτό φαίνεται καθαρά στο Σχήμα 2.2.

Κάθε pipe μπορεί να σχετιστεί με ένα συγκεκριμένο είδος εντολών και να σχεδιαστεί έτσι ώστε να καλύπτει απόλυτα τις απαιτήσεις του συγκεκριμένου είδους εντολών. Αυτό έχει ως αποτέλεσμα μία αποτελεσματική σχεδίαση του hardware που απαιτείται. Κάθε τύπος εντολών προκαλεί με αυτόν τον τρόπο μόνο την απολύτως αναγκαία χρονική καθυστέρηση και κάνει χρήση όλων των σταδίων ενός execution type. Αυτή η λύση είναι σαφώς πιο αποτελεσματική από τη χρήση s πανομοιότυπων αντιγράφων ενός ενιαίου execution pipe καθένα από τα οποία μπορεί να εκτελέσει εντολές όλων των τύπων. Μάλιστα, αν όλες οι εσωτερικές εξαρτήσεις ανάμεσα στις εντολές διαφορετικών τύπων εντολών λυθούν πριν το στάδιο dispatch, τότε από τη στιγμή που οι εντολές θα μπουν στα ξεχωριστά execution pipes δεν θα προκύψει καμία περαιτέρω καθυστέρηση λόγω των εντολών των άλλων pipes. Αυτό επιτρέπει τον κατανεμημένο και ανεξάρτητο έλεγχο του κάθε execution pipe. Ο αριθμός των λειτουργικών μονάδων στην ιδανική περίπτωση πρέπει να ισούται με το βαθμό παραλληλισμού σε επίπεδο εντολών.



Σχήμα 2.2 Τροποποιημένο παράλληλο pipeline με τέσσερις σωληνώσεις εκτέλεσης.

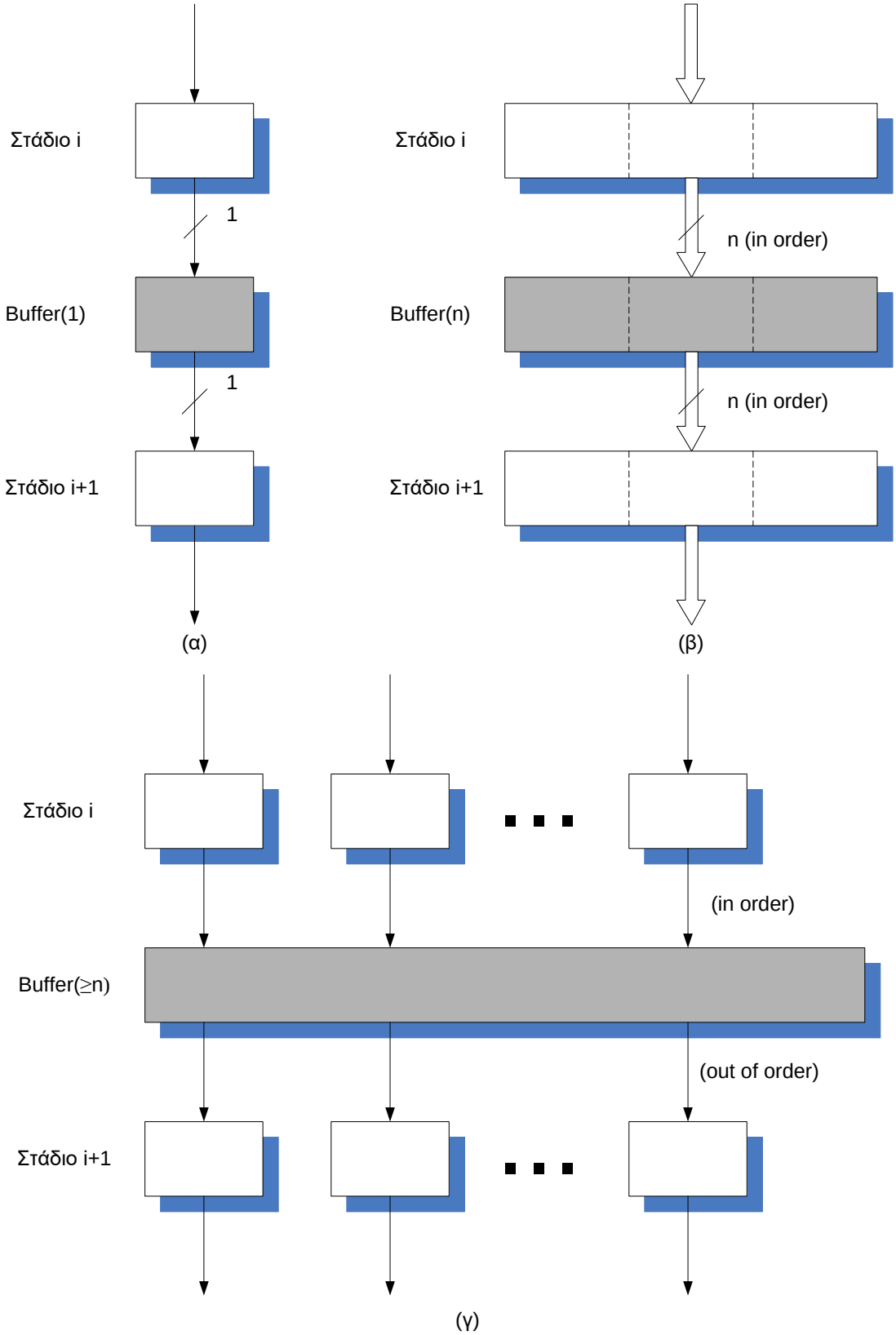
2.3 Δυναμικά pipelines

Σε κάθε pipelined σχεδίαση απαιτούνται καταχωρητές (buffers) ανάμεσα στα στάδια του pipeline. Σε ένα άκαμπτο scalar pipeline ένας καταχωρητής μίας θύρας τοποθετείται ανάμεσα σε δύο συνεχόμενα στάδια του pipeline (στάδια i και i+1). Ο καταχωρητής περιέχει όλα τα απαραίτητα bit ελέγχου (control bits) και δεδομένων (data bits) για την εντολή που έχει μόλις διατρέξει το στάδιο i και είναι έτοιμη να μπει στο στάδιο i+1 του pipeline στον επόμενο κύκλο μηχανής. Σε κάθε κύκλο μηχανής το τρέχον περιεχόμενο του καταχωρητή χρησιμοποιείται ως είσοδος για το στάδιο i+1, ενώ στο τέλος του κύκλου ο καταχωρητής λατσάρει (latch) το αποτέλεσμα που παράγεται στο στάδιο i. Βασικά λοιπόν ο καταχωρητής παίρνει νέα τιμή σε κάθε

κύκλο μηχανής. Η μοναδική εξαίρεση γίνεται στην περίπτωση που η εντολή πρέπει να περιμένει και να αποτραπεί η είσοδός της στο στάδιο i+1. Σε αυτή την περίπτωση ο καταχωρητής δεν παίρνει νέα τιμή στον κύκλο και η εντολή καθυστερεί στον καταχωρητή (γίνεται stall της εντολής). Προφανώς αν έχουμε stall του καταχωρητή σε ένα άκαμπτο scalar pipeline όλα τα στάδια που προηγούνται του σταδίου i πρέπει επίσης να καθυστερήσουν. Έτσι, σε ένα scalar pipeline, αν δεν υπάρχει stalling, κάθε εντολή παραμένει σε κάθε καταχωρητή για ακριβώς ένα κύκλο μηχανής και μετά προωθείται στον επόμενο καταχωρητή.

Στην περίπτωση του παράλληλου pipeline χρειάζονται καταχωρητές πολλαπλών θυρών ανάμεσα σε δύο συνεχόμενα στάδια του pipeline. Οι καταχωρητές πολλαπλών θυρών (multientry buffers) μπορούν να θεωρηθούν ως απλή επέκταση των καταχωρητών μίας θύρας. Πολλαπλές εντολές λατσάρονται σε κάθε καταχωρητή πολλαπλών θυρών σε κάθε κύκλο μηχανής. Στον επόμενο κύκλο αυτές οι εντολές μπορούν να προωθηθούν στο επόμενο στάδιο του pipeline. Αν όλες οι εντολές που βρίσκονται σε έναν καταχωρητή πολλαπλών θυρών απαιτούνται για να προχωρήσουν ταυτόχρονα με ένα βηματικό τρόπο, τότε ο έλεγχος του καταχωρητή πολλαπλών θυρών είναι παρόμοιος με τον έλεγχο του καταχωρητή μίας θύρας. Ολόκληρος ο καταχωρητής παίρνει νέα τιμή ή καθυστερεί σε κάθε κύκλο μηχανής. Όμως μία τέτοια λειτουργία του παράλληλου pipeline μπορεί να προκαλέσει μη αναγκαία καθυστέρηση για μερικές από τις εντολές ενός καταχωρητή πολλαπλών θυρών. Γίνεται λοιπόν εύκολα αντιληπτό ότι απαιτούνται πιο πολύπλοκοι καταχωρητές. Αυτοί οι καταχωρητές φαίνονται καλύτερα στο Σχήμα 2.3.

Στο superscalar pipeline επιτρέπεται οι εντολές που ακολουθούν μια προπορευόμενη εντολή που καθυστερεί να την παρακάμπτουν (bypassing), με σκοπό να ελαχιστοποιηθεί η μη αναγκαία καθυστέρηση. Ένα τέτοιο bypassing των εντολών μπορεί να αλλάξει τη σειρά εκτέλεσης των εντολών από την αρχική αυθεντική ακολουθιακή σειρά (σειρά εκτέλεσης των εντολών σύμφωνα με τον κώδικα του προγράμματος). Με αυτήν την out-of-order εκτέλεση των εντολών υπάρχει η δυνατότητα προσέγγισης του ορίου ροής δεδομένων (data flow limit) της εκτέλεσης των εντολών, δηλαδή οι εντολές εκτελούνται με το που θα γίνουν διαθέσιμα τα ορίσματά τους. Ένα παράλληλο pipeline που υποστηρίζει την out-of-order εκτέλεση των εντολών ονομάζεται δυναμικό pipeline (dynamic pipeline). Το δυναμικό pipeline επιτυγχάνει out-of-order εκτέλεση μέσω της χρήσης πολύπλοκων καταχωρητών πολλαπλών θυρών που επιτρέπουν στις εντολές να εισέρχονται στους καταχωρητές και να εξέρχονται από αυτούς με διαφορετική σειρά (Σχήμα 2.3.c).



Σχήμα 2.3 Καταχωρητές μεταξύ σταδίων του pipeline: (α) Καταχωρητής μίας θύρας (β) Καταχωρητής πολλαπλών θυρών (γ) Καταχωρητής πολλαπλών θυρών με Reordering.

2.4 Superscalar Pipeline

Σε αυτό το μέρος θα φανούν ορισμένα χαρακτηριστικά του superscalar pipeline και θα δοθεί έμφαση στα στάδια του pipeline καθώς επίσης και στους τρόπους με τους οποίους αυτά διασυνδέονται. Ουσιαστικά αυτό το κεφάλαιο επικεντρώνεται στη σχεδίαση της οργάνωσης του επεξεργαστή.

Ένα τυπικό superscalar pipeline μπορεί να έχει έξι στάδια. Αυτά τα στάδια είναι τα fetch, decode, dispatch, execute, complete και retire. Φυσικά υπάρχουν pipelines με διαφορετικά στάδια αλλά εδώ θα δούμε τη σχεδίαση με βάση αυτά τα έξι που προαναφέρθηκαν. Το στάδιο execute (στο οποίο γίνεται η εκτέλεση της εντολής) μπορεί να περιέχει πολλαπλές λειτουργικές μονάδες (ALUs, branch units κτλ) διαφορετικών τύπων με διαφορετικούς χρόνους απόκρισης, δηλαδή με διαφορετικές καθυστερήσεις. Αυτές οι μονάδες χρησιμοποιούν επίσης την τεχνική του pipelining. Αυτό βέβαια σημαίνει ότι στο στάδιο dispatch πρέπει οι εντολές διαφορετικών τύπων να προωθούνται στις κατάλληλες λειτουργικές μονάδες. Για παράδειγμα οι εντολές διακλάδωσης θα πρέπει να προωθούνται στο branch unit, οι αριθμητικές εντολές στην integer unit (όπου υπάρχει ALU) κ.ο.κ. Φυσικά λόγω της out-of-order εκτέλεσης των εντολών στο στάδιο execute στο στάδιο complete υπάρχει η ανάγκη αναδιάταξης των εντολών και της διασφάλισης με αυτόν τον τρόπο της ολοκλήρωσης των εντολών με βάση την σειρά στην οποία αυτές βρίσκονται στον πηγαίο κώδικα. Η διασύνδεση ανάμεσα στα στάδια επιτυγχάνεται με χρήση καταχωρητών πολλαπλών θυρών (multientry buffers) οι οποίοι χρειάζονται για την προώθηση των εντολών από ένα στάδιο του pipeline στο επόμενο. Ακολουθεί μία σύντομη περιγραφή των έξι σταδίων ενός τυπικού superscalar pipeline.

2.4.1 Στάδιο fetch

Αν το superscalar pipeline έχει πλάτος s τότε το στάδιο fetch θα πρέπει να είναι ικανό να φέρει s εντολές από την I-cache μνήμη (I-cache μνήμη είναι η instruction cache μνήμη, δηλαδή η κρυφή μνήμη στην οποία είναι αποθηκευμένες κάθε φορά οι εντολές του εκάστοτε προγράμματος). Αυτό σημαίνει ότι η φυσική οργάνωση της I-cache πρέπει να είναι αρκετά «πλατιά» έτσι ώστε κάθε γραμμή της μνήμης να μπορεί να αποθηκεύσει s εντολές και παράλληλα να μπορεί να γίνει προσπέλαση ολόκληρης της γραμμής σε μία χρονική στιγμή. Το ζητούμενο στο στάδιο fetch είναι η μεγιστοποίηση του εύρους ζώνης των εντολών που φέρνει το στάδιο από την I-cache. Το throughput του fetch σταδίου ούτως ή άλλως είναι αυτό που περιορίζει την τιμή του συνολικού throughput του superscalar pipeline δίνοντας του μία μέγιστη τιμή. Το throughput των επόμενων σταδίων (άρα και το συνολικό) δεν μπορεί να ξεπεράσει την τιμή του throughput του σταδίου fetch και για αυτό το λόγο το στάδιο αυτό και το εύρος ζώνης του φορτώματος των εντολών (fetching) (instruction-fetching bandwidth) είναι πολύ σημαντικό για την αποδοτικότητα του επεξεργαστή. Τα δύο προβλήματα που επιδρούν αρνητικά σε αυτό το throughput είναι:

- Η μη-ευθυγράμμιση των s εντολών που φέρνει από τη μνήμη το στάδιο fetch σε σχέση με την οργάνωση των γραμμών της μνήμης.
- Η παρουσία εντολών ελέγχου-αλλαγής ροής του προγράμματος (όπως οι εντολές διακλάδωσης) μέσα στην ομάδα των εντολών που αφίχθησαν.

Το στάδιο fetch χρησιμοποιεί την τιμή του καταχωρητή PC (Programm Counter) για να φέρει την εντολή στην οποία δείχνει ο PC μαζί με τις επόμενες s-1 εντολές από την I-cache. Αν η τιμή του PC δεν αντιστοιχεί στο πρώτο από τα s στοιχεία της

γραμμής της μνήμης (υποθέτουμε ότι η κάθε γραμμή της I-cache έχει s εντολές) τότε θα χρειαστούν 2 κύκλοι για να έρθουν και οι s εντολές από την μνήμη. Αυτό το πρόβλημα είναι το πρόβλημα της μη-ευθυγράμμισης που αναφέρθηκε νωρίτερα. Οι τρόποι αντιμετώπισης αυτού του προβλήματος είναι είτε η ύπαρξη πληροφοριών στον compiler για την οργάνωση της I-cache και της περαιτέρω κατάλληλης ευθυγράμμισης από τον ίδιο τον compiler είτε λύση με χρήση κατάλληλου πρόσθετου hardware. Όσων αφορά το δεύτερο πρόβλημα θα πρέπει να γίνεται κατάλληλος έλεγχος στις εντολές διακλάδωσης ούτως ώστε να μην μειώνεται το εύρος ζώνης.

2.4.2 Στάδιο decode

Το στάδιο decode (στάδιο αποκωδικοποίησης) σχετίζεται με την αναγνώριση των ξεχωριστών εντολών (αναγνώριση της κάθε εντολής ξεχωριστά), τον καθορισμό του τύπου της κάθε εντολής και την ανακάλυψη των εσωτερικών εξαρτήσεων ανάμεσα στις εντολές που έχουν αφιχθεί από το στάδιο fetch αλλά δεν έχουν ακόμα περάσει στο στάδιο dispatch. Η πολυπλοκότητα της αποκωδικοποίησης των εντολών εξαρτάται από δύο βασικούς παράγοντες, το ISA (Instruction Set Architecture) και το πλάτος του παράλληλου pipeline. Το ISA είναι η αρχιτεκτονική του συνόλου εντολών του επεξεργαστή, δηλαδή το format των εντολών (περιλαμβάνει για παράδειγμα το μήκος της εντολής που μπορεί να είναι είτε σταθερό είτε μεταβλητό, τους διαφορετικούς τύπους των εντολών, την αντιστοιχία των bits της εντολής με τα διάφορα ορίσματα και το operation code των εντολών κ.ο.κ.). Για ένα τυπικό RISC συνόλου εντολών με εντολές σταθερού μήκους και με απλά format εντολών, η λειτουργία αποκωδικοποίησης είναι αρκετά απλή και με δεδομένα βήματα. Δεν χρειάζεται ιδιαίτερη προσπάθεια για τον καθορισμό της αρχής και του τέλους της κάθε εντολής ενώ τα σχετικά λίγα διαφορετικά format εντολών καθιστούν τον διαχωρισμό των τύπων εντολών εύκολη. Με απλή αποκωδικοποίηση των λιγοστών bits του operation code μίας εντολής μπορεί να καθοριστεί ο τύπος της εντολής και να προσδιοριστούν τα υπόλοιπα πεδία της εντολής (ορίσματα, καταχωρητής προορισμός κτλ). Οι δύο βασικές λειτουργίες που πρέπει να γίνονται στο στάδιο decode είναι η εύρεση των εσωτερικών εξαρτήσεων των εντολών μετά τον καθορισμό των καταχωρητών ορισμάτων και προορισμού καθώς επίσης και η εύρεση των εντολών διακλάδωσης. Εσωτερικές εξαρτήσεις υπάρχουν στις περιπτώσεις που ένας καταχωρητής εισόδου μίας εντολής αποτελεί καταχωρητή εξόδου μίας άλλης εντολής που προηγείται της πρώτης. Αυτές οι εξαρτήσεις είναι του τύπου WAR (write after read), που σημαίνει ανάγνωση μετά από εγγραφή (του ίδιου προφανώς καταχωρητή). Υπάρχουν και εξαρτήσεις των τύπων WAW (write after write) και RAW (read after write), δηλαδή εγγραφής μετά από εγγραφή και ανάγνωσης μετά από εγγραφή που όμως είναι ουσιαστικά ψευδοεξαρτήσεις και αντιμετωπίζονται με άλλες τεχνικές στα μετέπειτα στάδια του pipeline.

2.4.3 Στάδιο dispatch

Η λειτουργία του σταδίου dispatch είναι ουσιαστικά η προώθηση της κάθε εντολής με βάση τον τύπο της, που έχει καθοριστεί από το στάδιο αποκωδικοποίησης στην κατάλληλη λειτουργική μονάδα που αντιστοιχεί στο συγκεκριμένο τύπο εντολής., αφού όπως έχουμε αναφέρει διαφορετικοί τύποι εντολών εκτελούνται σε διαφορετικές μονάδες εκτέλεσης ή αλλιώς λειτουργικές μονάδες (functional units). Οι λειτουργίες που επιτελούνται τόσο στο στάδιο fetch όσο και στο στάδιο decode γίνονται συνήθως με έναν συγκεντρωμένο κεντρικό τρόπο, δηλαδή όλες οι εντολές

διαχειρίζονται από τις ίδιες μονάδες ελέγχου. Από την άλλη πλευρά όλες οι λειτουργικές μονάδες σε ένα τροποποιημένο pipeline λειτουργούν ανεξάρτητα η μία από την άλλη και έτσι οι λειτουργίες μπορούν να γίνουν με ένα κατανεμημένο τρόπο. Επομένως μπορούμε να πούμε ότι περνώντας από την αποκωδικοποίηση των εντολών (στάδιο decode) στην εκτέλεση των εντολών (στάδιο execute) γίνεται μία αλλαγή από κεντρική επεξεργασία σε κατανεμημένη επεξεργασία. Αυτή η δουλειά επιτελείται στο στάδιο dispatch.

Ένας ακόμα μηχανισμός που είναι απολύτως απαραίτητος ανάμεσα στην αποκωδικοποίηση και στην εκτέλεση των εντολών είναι η προσωρινή αποθήκευση (φύλαξη) των εντολών σε καταχωρητές. Κάθε εντολή πριν την εκτέλεσή της χρειάζεται όλα τα ορίσματα της, τα οποία έρχονται κατά το στάδιο αποκωδικοποίησης από τα register files (αρχεία καταχωρητών) και τα οποία ενδέχεται να μην είναι έτοιμα εξαιτίας εντολών που προηγούνται, οι οποίες δεν έχουν ολοκληρώσει ακόμα την εκτέλεσή τους και έχουν ως καταχωρητή εξόδου κάποιον από τους καταχωρητές-ορίσματα της εντολής αυτής. Μία λύση στο πρόβλημα αυτό είναι το stall του σταδίου decode μέχρι όλοι οι καταχωρητές ορίσματα να είναι έτοιμοι. Αυτή η λύση βέβαια δεν είναι επιθυμητή. Μία άλλη λύση είναι να φέρουμε όλους τους καταχωρητές ορίσματα που είναι έτοιμοι και να προωθήσουμε τις εντολές αυτές σε ένα ξεχωριστό καταχωρητή όπου θα περιμένουν τις τιμές των καταχωρητών ορισμάτων που δεν είναι ακόμη έτοιμα. Όταν όλα τα ορίσματα μιας εντολής είναι έτοιμα τότε η εντολή αυτή θα προωθείται στην κατάλληλη λειτουργική μονάδα για να εκτελεστεί (issuing). Ένας καταχωρητής όπως αυτός που περιγράψαμε τώρα ονομάζεται σταθμός αναμονής (reservation station). Υπάρχουν δύο τεχνικές χρήσης των σταθμών αναμονής. Ο πρώτος είναι να χρησιμοποιήσουμε ένα κεντρικό σταθμό αναμονής (centralized reservation station), δηλαδή ένας καταχωρητής στην είσοδο του σταδίου dispatch και ο δεύτερος είναι να χρησιμοποιηθούν πολλοί σταθμοί αναμονής (distributed reservation stations), δηλαδή πολλοί καταχωρητές στην έξοδο του σταδίου dispatch. Στην πρώτη περίπτωση από τον μοναδικό σταθμό αναμονής γίνεται έκδοση των έτοιμων προς εκτέλεση εντολών στις κατάλληλες λειτουργικές μονάδες, ενώ στη δεύτερη κάθε λειτουργική μονάδα έχει το δικό της σταθμό αναμονής στην πλευρά εισόδου της μονάδας. Αξίζει να σημειωθεί ότι υπάρχουν και υβριδικές λύσεις. Εδώ πρέπει να δοθεί ιδιαίτερη προσοχή και στη διαφορά ανάμεσα στην έννοιες dispatching και issuing. Ο πρώτος όρος εννοεί τη συσχέτιση των τύπων εντολών με τις αντίστοιχες λειτουργικές μονάδες ενώ ο δεύτερος σημαίνει την εκκίνηση της εκτέλεσης της εντολής μέσα στη λειτουργική μονάδα.

2.4.4 Στάδιο execute

Το στάδιο εκτέλεσης των εντολών είναι η καρδιά ενός superscalar επεξεργαστή. Η σημερινή τάση στη σχεδίαση superscalar pipeline είναι προς περισσότερο τροποποιημένα και παράλληλα pipelines. Αυτό συνεπάγεται ότι πρέπει να υπάρχουν περισσότερες λειτουργικές μονάδες οι οποίες πρέπει να κάνουν ακόμα πιο συγκεκριμένες λειτουργίες από ότι σε παλαιότερα συστήματα. Με αυτή την εξειδίκευση των λειτουργικών μονάδων έτσι ώστε να εκτελούν συγκεκριμένους μόνο τύπους εντολών αυξάνεται αισθητά η απόδοσή τους.

Στους αρχικούς superscalar επεξεργαστές (πρώτης γενιάς) υπήρχαν δύο λειτουργικές μονάδες, μία που εκτελούσε τις εντολές ακεραίων (integer unit) και μία που εκτελούσε τις εντολές κινητής υποδιαστολής (floating point unit). Οι σημερινοί superscalar επεξεργαστές διαθέτουν πολλές μονάδες για εντολές ακεραίων και μερικοί έχουν πάνω από μία μονάδες για εντολές κινητής υποδιαστολής (τις

περισσότερες φορές όλες αυτές οι μονάδες είναι pipelined, δηλαδή εκτελούν τις πράξεις σε πάνω από ένα στάδια). Οι integer units πολλές φορές μπορούν να χρησιμοποιηθούν για την εκτέλεση και εντολών διακλάδωσης (branch) ή φόρτωσης/αποθήκευσης δεδομένων (load/store). Παρόλα αυτά στις περισσότερες σχεδιάσεις πλέον υπάρχουν ξεχωριστές λειτουργικές μονάδες για αυτές τις λειτουργίες, δηλαδή μονάδα για εντολές διακλάδωσης (branch unit) και μονάδα για εντολές load/store (load/store unit). Η load/store unit είναι απευθείας συνδεδεμένη με την D-cache μνήμη (D-cache είναι η Data-cache μνήμη, δηλαδή η κρυφή μνήμη όπου είναι αποθηκευμένα τα δεδομένα), ενώ η branch unit είναι υπεύθυνη για την ενημέρωση του καταχωρητή PC. Όπως γίνεται εύκολα αντιληπτό ο συνολικός αριθμός των λειτουργικών μονάδων πρέπει να υπερβαίνει το πλάτος του superscalar pipeline ώστε να αποφευχθεί ο κίνδυνος να γίνει το στάδιο εκτέλεσης των εντολών ο στενωπός του συστήματος λόγω της μη διαθεσιμότητας συγκεκριμένων τύπων λειτουργικών μονάδων. Τέλος πρέπει να προστεθεί ότι πολλές φορές οι έξοδοι λειτουργικών μονάδων συνδέονται απευθείας με εισόδους άλλων λειτουργικών μονάδων για ταχύτητα λειτουργίας, όμως κάτι τέτοιο απαιτεί πολυπλοκότητα διαδρόμων επικοινωνίας και μονάδων διαιτησίας και ελέγχου των διαδρόμων αυτών.

2.4.5 Στάδια complete και retire

Μία εντολή θεωρείται ολοκληρωμένη (completed) όταν ολοκληρώνει την εκτέλεσή της και ενημερώνει κατάλληλα την κατάσταση της μηχανής. Η ολοκλήρωση της εκτέλεσης ουσιαστικά γίνεται όταν η εντολή εξέρχεται της λειτουργικής μονάδας και εισέρχεται στον καταχωρητή ολοκλήρωσης (completion buffer). Στη συνέχεια η εντολή βγαίνει από τον completion buffer και τότε γίνεται ολοκληρωμένη. Όταν μία εντολή ολοκληρώσει την εκτέλεσή της το αποτέλεσμά της μπορεί να βρίσκεται μόνο σε μη-αρχιτεκτονικούς καταχωρητές. Όμως όταν η εντολή είναι completed το αποτέλεσμά της γράφεται σε έναν αρχιτεκτονικό καταχωρητή. Όσων αφορά εντολές οι οποίες ενημερώνουν θέσεις μνήμης μπορεί να υπάρχει ένα μικρό χρονικό διάστημα από τη στιγμή που η εντολή ολοκληρωθεί αρχιτεκτονικά μέχρι τη στιγμή που θα ενημερωθεί η κατάλληλη θέση μνήμης. Για παράδειγμα, μία εντολή αποθήκευσης μπορεί να έχει ολοκληρωθεί αρχιτεκτονικά όταν εξέλθει από τον completion buffer και μπει στον store buffer για να βρει διαθέσιμο ένα κύκλο διαδρόμου (bus) έτσι ώστε να γράψει στην D-cache. Η εντολή αυτή θεωρείται retired όταν εξέλθει από τον store buffer και ενημερώσει την D-cache. Έτσι, το completion έχει σχέση με την ενημέρωση της κατάστασης μηχανής ενώ το retiring έχει σχέση με την ανανέωση της κατάστασης της μνήμης.

2.5 Τεχνικές Superscalar

Στους superscalar επεξεργαστές είναι αρκετά βολικό να βλέπουμε την επεξεργασία των εντολών σαν έναν συνδυασμό τριών ροών εντολών και/ή δεδομένων. Αυτές οι τρεις ροές είναι η ροή εντολών (instruction flow), η ροή δεδομένων καταχωρητών (register data flow) και η ροή δεδομένων μνήμης (memory data flow). Αυτά τα τρία μονοπάτια ροής ανταποκρίνονται στην επεξεργασία των τριών κύριων τύπων εντολών, δηλαδή των εντολών διακλάδωσης (branch), των εντολών ALU και των εντολών φόρτωσης/αποθήκευσης (load/store). Έτσι η μεγιστοποίηση του throughput των τριών ροών ουσιαστικά επιτυγχάνεται με την

ελαχιστοποίηση των ποινών των branch, ALU και load/store εντολών. Η αντιστοιχία ροών και τύπων εντολών είναι η ακόλουθη:

- Ροή εντολών ⇔ Επεξεργασία εντολών branch.
- Ροή δεδομένων καταχωρητών ⇔ Επεξεργασία εντολών ALU.
- Ροή δεδομένων μνήμης ⇔ Επεξεργασία εντολών load/store.

Στη συνέχεια θα αναφερθούν ορισμένες τεχνικές που χρησιμοποιούνται για την ελαχιστοποίηση των προβλημάτων που σχετίζονται με τις τρεις ροές εντολών και δεδομένων που σαν στόχο έχουν τη μεγιστοποίηση της αποδοτικότητας τους επεξεργαστή.

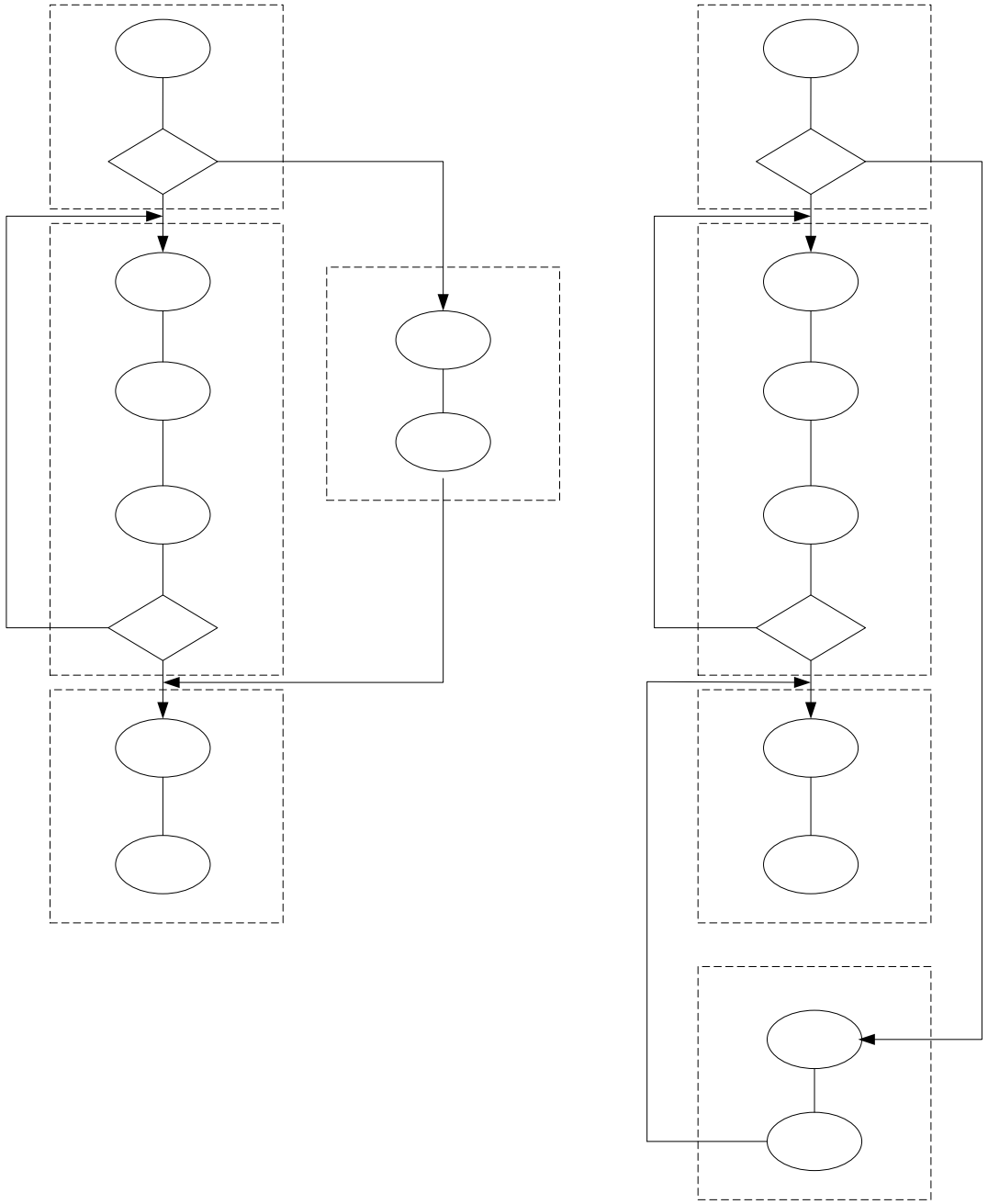
2.5.1 Τεχνικές ροής εντολών (instruction flow techniques)

Οι τεχνικές ροής εντολών σχετίζονται με τα αρχικά στάδια ενός superscalar pipeline, δηλαδή με τα στάδια fetch και decode. Το throughput των αρχικών αυτών σταδίων θέτει ένα άνω όριο για το throughput όλων των επόμενων σταδίων (και επομένως και για το συνολικό throughput). Ο πρωταρχικός στόχος αυτών των τεχνικών είναι η μεγιστοποίηση της παροχής εντολών στο superscalar pipeline που υπόκειται στις απαιτήσεις του διαγράμματος ροής ελέγχου του προγράμματος.

Ακολουθούν οι σημαντικότερες τεχνικές ροής εντολών.

2.5.1.1 Ροή ελέγχου και εξαρτήσεις ελέγχου προγράμματος

Σε αυτήν την τεχνική χρησιμοποιούνται γράφοι ελέγχου ροής για την αναπαράσταση του ελέγχου ροής του προγράμματος. Υπάρχουν ορισμένα βασικά block εντολών τα οποία αποτελούνται από συνεχόμενες εντολές που δεν είναι εντολές ελέγχου ροής, δηλαδή δεν είναι εντολές διακλάδωσης υπό συνθήκη (conditional branch). Αυτά τα block αποθηκεύονται στην I-cache σε συνεχόμενες θέσεις για να είναι πιο εύκολη η διαδικασία του fetching από την I-cache μνήμη. Με αυτόν τον τρόπο επιτυγχάνεται να βρίσκονται σε συνεχόμενες θέσεις στην μνήμη εντολές που βρίσκονται σε συνεχόμενες θέσεις και στον πηγαίο κώδικα του προγράμματος. Επίσης πρέπει να προστεθεί ότι τα block εντολών διακόπτονται λόγω unconditional branches που οδηγούν σε νέα block εντολών. Αυτά τα πρόσθετα block εντολών αποθηκεύονται στην μνήμη στις αμέσως επόμενες θέσεις από το block εντολών που αντιστοιχούν στην μία από τις δύο αληθοτιμές του branch (δηλαδή μετά την εντολή διακλάδωσης που αποτελεί την τελευταία εντολή ενός block ακολουθούν οι εντολές του block που αντιστοιχούν στο branch taken και αμέσως μετά βρίσκονται οι εντολές του block που αντιστοιχούν στο branch not-taken). Αυτά φαίνονται καλύτερα στο Σχήμα 2.4 που ακολουθεί.

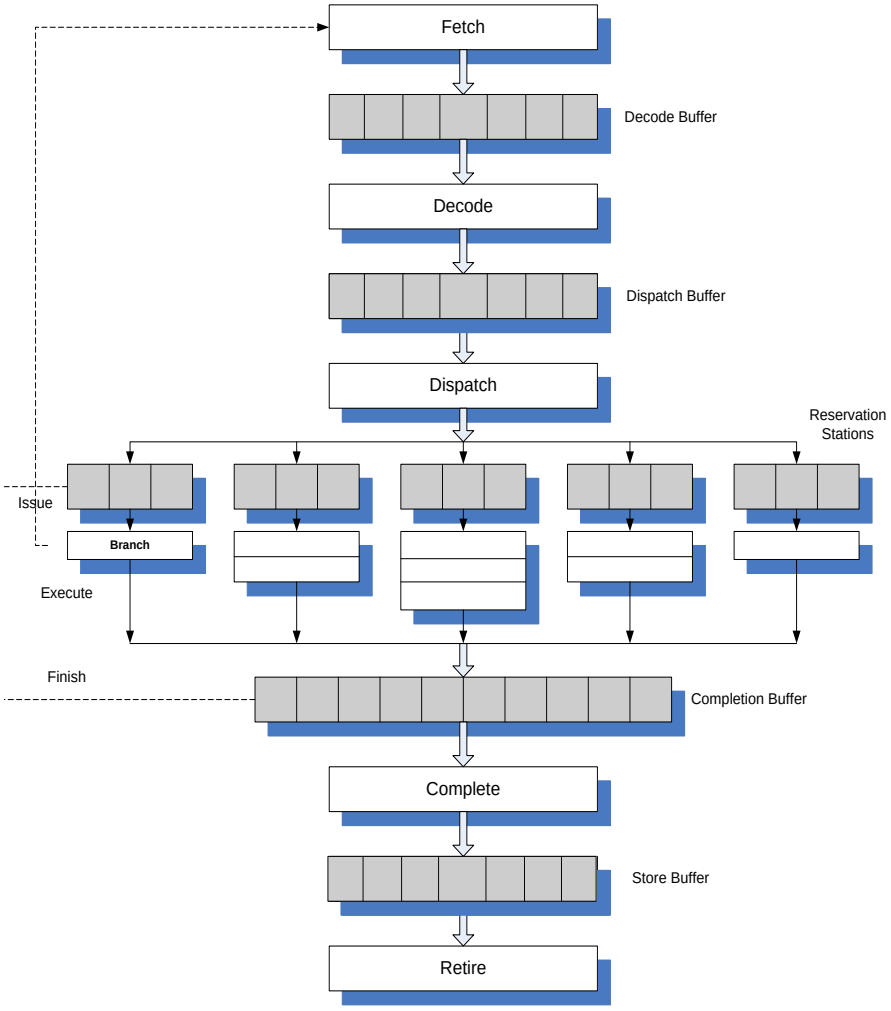


Σχήμα 2.4 Ροή ελέγχου προγράμματος: (α) Ο γράφος ελέγχου ροής (Control Flow Graph, CFG) (β) Αντιστοίχηση του CFG σε συνεχόμενες θέσεις μνήμης.

2.5.1.2 Υποβάθμιση απόδοσης λόγω branches

Στο στάδιο fetch η κανονική λειτουργία επιτυγχάνεται με το συνεχόμενο φόρτωμα εντολών που βρίσκονται σε συνεχόμενες θέσεις στη μνήμη του προγράμματος. Για τα unconditional branches όμως οι επόμενες εντολές δεν μπορούν να έρθουν μέχρι να καθοριστεί η διεύθυνση-στόχος (τελική διεύθυνση ή target address) του branch. Για τα conditional branches ο επεξεργαστής πρέπει να περιμένει μέχρι την επίλυση της συνθήκης διακλάδωσης, και αν η διακλάδωση πρέπει να γίνει θα πρέπει να υπάρχει περαιτέρω αναμονή μέχρι να γίνει διαθέσιμη η τελική

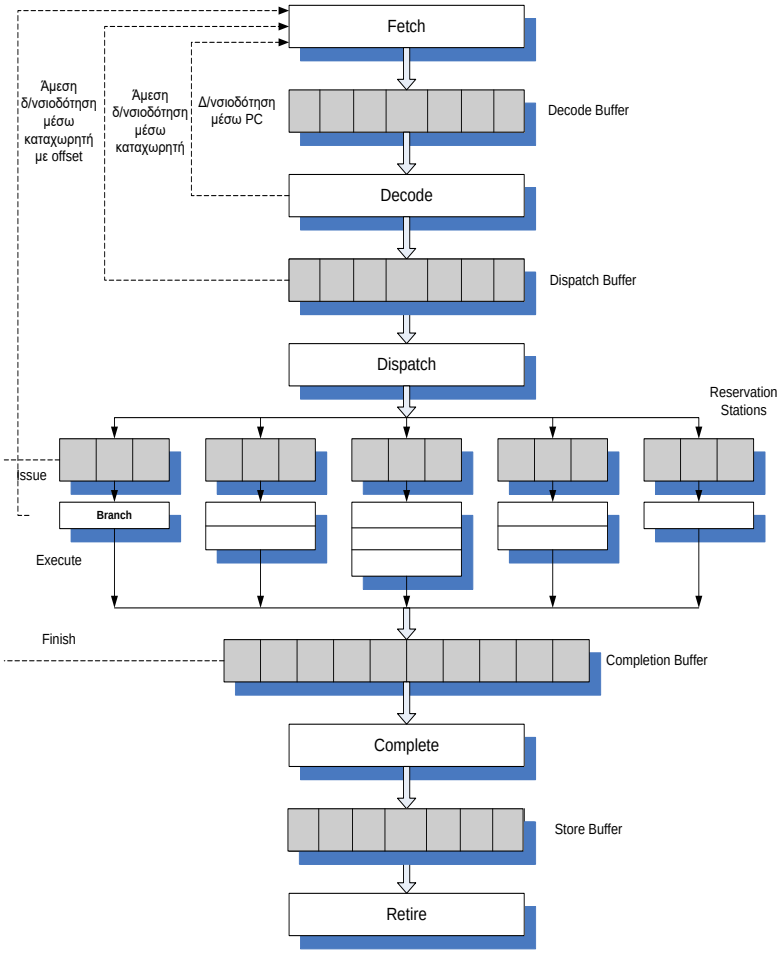
διεύθυνση του branch. Τα branches εκτελούνται ως γνωστόν στην branch unit. Για ένα conditional branch το στάδιο fetch μπορεί να φέρει τις εντολές που έπονται του branch αφού αυτή η εντολή εξέλθει από το branch unit και όταν η συνθήκη διακλάδωσης και η τελική διεύθυνση του branch γίνουν γνωστές. Αυτή η καθυστέρηση στην επεξεργασία των conditional branches επιφέρει μία ποινή της τάξης των τριών κύκλων, όπως φαίνεται και στο Σχήμα 2.5, μέχρι να έρθει η επόμενη εντολή από το στάδιο fetch, αφού θα πρέπει η εντολή διακλάδωσης να περάσει από τα στάδια decode, dispatch και execute για να επιλυθεί η διακλάδωση και να υπολογιστεί η τιμή του PC και όταν η εντολή διακλάδωσης μπει στο στάδιο complete να μπει και η επόμενη στο στάδιο fetch. Αυτοί βέβαια οι τρεις κύκλοι θα πρέπει να πολλαπλασιαστούν με το πλάτος του superscalar pipeline για να υπολογιστούν τα χαμένα slots εντολών στο pipeline (αφού σε κάθε χαμένο κύκλο θα έμπαιναν τόσες εντολές για εκτέλεση όσες και το πλάτος του pipeline). Για παράδειγμα, για μία μηχανή με πλάτος ίσο με 4 η συνολική ποινή είναι 12 «φούσκες» εντολών (bubbles) στο superscalar pipeline. Ουσιαστικά πάντως η ποινή μπορεί να οφείλεται στην παραγωγή της τελικής διεύθυνσης του branch (περίπτωση unconditional branches) ή στην παραγωγή της τελικής διεύθυνσης του branch και στην καθυστέρηση της επίλυσης της διακλάδωσης (περίπτωση conditional branches).



Σχήμα 2.5 Καταστροφή της σειριακής ροής ελέγχου από εντολές διακλάδωσης.

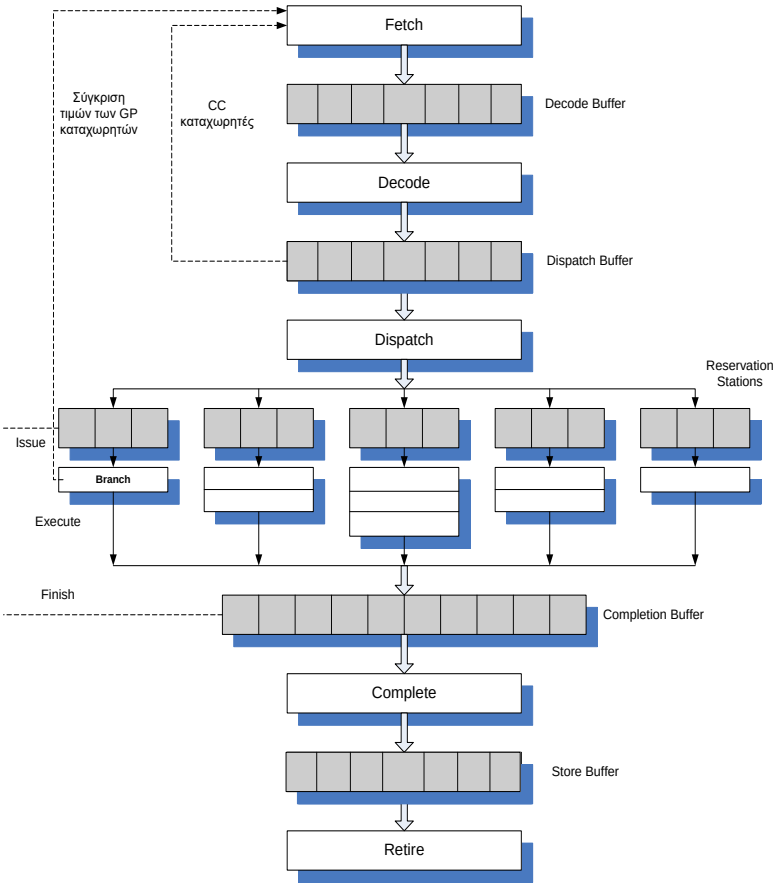
Για διευθυνσιοδότηση σχετική με το PC η τελική διεύθυνση μπορεί να παραχθεί κατά τη διάρκεια του fetch σταδίου οπότε υπάρχει ποινή ενός κύκλου. Αν

έχουμε έμμεση διευθυνσιοδότηση μέσω καταχωρητή έχουμε ποινή δύο κύκλων αφού υπάρχει πρόσβαση στον καταχωρητή στο στάδιο decode. Αν έχουμε έμμεση διευθυνσιοδότηση μέσω καταχωρητή με κάποιο offset (offset είναι κάποια τιμή που προστίθεται στην υπολογιζόμενη μέσω του καταχωρητή τιμή) τότε προκύπτει ποινή τριών κύκλων. Αυτές οι τρεις περιπτώσεις φαίνονται καθαρά στο Σχήμα 2.6.



Σχήμα 2.6 Ποινές υπολογισμού της διεύθυνσης στόχου της διακλάδωσης.

Όσων αφορά τέλος τις ποινές επίλυσης της συνθήκης υπάρχει η περίπτωση χρησιμοποίησης CC καταχωρητών (ειδικοί καταχωρητές κωδικού συνθήκης, Condition Code registers), όπου υποθέτοντας ότι σε αυτούς τους καταχωρητές υπάρχει πρόσβαση στο στάδιο dispatch, υπάρχει ποινή δύο κύκλων, καθώς επίσης και η περίπτωση χρησιμοποίησης καταχωρητών γενικού σκοπού (General Purpose registers, GP registers) όπου χρειάζεται μία σύγκριση των περιεχομένων των καταχωρητών στην ALU για την επίλυση, και επομένως υπάρχει ποινή τριών κύκλων. Αυτές οι δύο περιπτώσεις φαίνονται στο Σχήμα 2.7.



Σχήμα 2.7 Ποινές επιλύσεων των συνθηκών διακλάδωσης.

2.5.1.3 Τεχνικές πρόβλεψης διακλάδωσης (branch prediction techniques)

Μία λύση στο πρόβλημα των ποινών των διακλαδώσεων είναι η πρόβλεψη των branches. Δύο βασικοί συντελεστές της πρόβλεψης διακλάδωσης είναι η πρόβλεψη στόχου διακλάδωσης (branch target speculation) και η πρόβλεψη συνθήκης διακλάδωσης (branch condition speculation).

Η πρόβλεψη στόχου διακλάδωσης, δηλαδή η πρόβλεψη της τελικής διεύθυνσης του branch στηρίζεται στην χρήση ενός καταχωρητή στον οποίο φυλάγονται προηγούμενες τιμές τελικών διευθύνσεων των branches. Ένας τέτοιος καταχωρητής λέγεται BTB (branch target buffer). Κάθε στοιχείο αυτού του καταχωρητή αποτελείται από δύο πεδία, την διεύθυνση της εντολής branch (BIA) και την τελική διεύθυνση του branch (BTA). Όταν ένα branch εκτελεστεί για πρώτη φορά γίνεται μία εγγραφή στον BTB. Εν συνεχεία στα υπόλοιπα branches γίνεται το ίδιο αλλά κάθε φορά γίνεται έλεγχος στον BTB και αν το τρέχον PC ισούται με το πεδίο BIA ενός στοιχείου του BTB η τελική διεύθυνση του branch γίνεται ίση με το αντίστοιχο πεδίο του στοιχείου αυτού. Αν μάλιστα το branch προβλεφθεί ως σωστό και αυτή η πρόβλεψη αποδειχθεί σωστή τότε δεν θα υπάρξει καμία ποινή λόγω του branch.

Η πρόβλεψη συνθήκης διακλάδωσης από την άλλη μεριά μπορεί να γίνει με διάφορους τρόπους. Ο πιο απλός είναι να ακολουθηθεί κάποια πολιτική για τα branches, δηλαδή πολιτική branch taken όπου όλα τα branches θεωρούνται σωστά ή πολιτική branch not-taken όπου όλα τα branches θεωρούνται λανθασμένα και δεν γίνονται. Άλλος τρόπος είναι με τη χρήση ενός πρόσθετου bit που η τιμή του τίθεται

από τον compiler που υπολογίζει την τιμή του bit αυτού με βάση πληροφορίες που αποκτά για το branch κατά τη διάρκεια του compilation. Ένας τρίτος τρόπος είναι η πρόβλεψη με βάση το offset της τελικής διεύθυνσης της διακλάδωσης. Πάντως ο πιο συνήθης και διαδεδομένος τρόπος πρόβλεψης branch είναι αυτός που κάνει χρήση πίνακα με την ιστορία των προηγούμενων εκτελέσεων branches. Ουσιαστικά σε αυτόν τον τρόπο η πρόβλεψη, είτε taken είτε not-taken, γίνεται βασισμένη σε προηγούμενα παρατηρημένες κατευθύνσεις διακλάδωσης. Εδώ χρησιμοποιούνται συνήθως ντετερμινιστικές μηχανές καταστάσεων (finite state machines, FSMs) με καταστάσεις των κ bit, όπου το κ είναι ο αριθμός των τελευταίων branches που έχουν επιλυθεί και κάθε bit έχει τιμή ανάλογα με το αν το αντίστοιχο branch έγινε ή όχι. Για παράδειγμα, για ένα πίνακα ιστορίας branch με 2 bits οι καταστάσεις είναι: NN, NT, TT και TN. Αυτές λοιπόν οι μηχανές καταστάσεων ανάλογα με την κατάσταση στην οποία βρίσκονται κάνουν πρόβλεψη για το επόμενο branch, αλλάζουν κατάσταση και στο τέλος κάθε branch ανανεώνουν τον πίνακα ιστορίας (history table).

2.5.1.4 Επανόρθωση λανθασμένης πρόβλεψης διακλάδωσης (branch misprediction recovery)

Η πρόβλεψη ενός branch οδηγεί σε μία σειρά από εντολές οι οποίες μπορούν να εμπεριέχουν νέα branches. Σε αυτήν την περίπτωση τα επιπλέον branches μπορούν να εκτελεστούν με τον ίδιο τρόπο (γίνεται επίσης πρόβλεψη) αλλά δημιουργούνται έτσι μονοπάτια πρόβλεψης με πάνω του ενός branches που δεν έχουν ακόμα επιλυθεί. Σε κάθε εντολή ενός speculation path (όπως λέγεται το μονοπάτι πρόβλεψης) δίνεται η ίδια αναγνωριστική ετικέτα. Μία εντολή με τέτοια ετικέτα δείχνει ότι είναι μία εντολή που έχει προβλεφθεί και η ετικέτα υποδηλώνει σε ποιο μονοπάτι ανήκει αυτή η εντολή. Η επικύρωση του branch (branch validation) γίνεται όταν το branch εκτελεστεί κανονικά στο branch unit. Αν η πρόβλεψη αποδειχτεί σωστή τότε η ετικέτα απομακρύνεται από τις αντίστοιχες εντολές και όλες οι εντολές που σχετίζονται με αυτή την ετικέτα δεν θεωρείται πλέον ότι έχει προβλεφθεί και μπορεί να ολοκληρωθεί. Αν η πρόβλεψη αποδειχθεί λανθασμένη τότε το λανθασμένο μονοπάτι πρέπει να τερματιστεί και η διαδικασία fetch από νέο σωστό μονοπάτι πρέπει να αρχίσει. Για αυτό το λόγο το PC θα πρέπει να πάρει νέα σωστή τιμή. Αν η λανθασμένη πρόβλεψη ήταν μία not-taken πρόβλεψη τότε το PC ενημερώνεται με τη υπολογισμένη τελική διεύθυνση του branch. Αν η λανθασμένη πρόβλεψη ήταν μία taken πρόβλεψη τότε το PC ενημερώνεται με την επόμενη διεύθυνση εντολής (την επόμενη της branch εντολής που τελικά δεν είναι taken). Για τον τερματισμό του λανθασμένου μονοπατιού χρησιμοποιούνται οι ετικέτες πρόβλεψης. Οι ετικέτες που σχετίζονται με την λανθασμένη πρόβλεψη χρησιμοποιούνται για την αναγνώριση των λανθασμένων εντολών και έτσι αυτές οι εντολές τερματίζουν. Επίσης οι εντολές με αυτήν την ετικέτα που βρίσκονται ακόμα σε αρχικά στάδια ακυρώνονται. Τέλος οι αντίστοιχες θέσεις στον reorder buffer ελευθερώνονται.

2.5.1.5 Προχωρημένες τεχνικές πρόβλεψης διακλάδωσης (advanced branch prediction techniques)

Υπάρχουν νέες ανεπτυγμένες τεχνικές πρόβλεψης διακλάδωσης οι οποίες λαμβάνουν σοβαρά υπ' όψιν τους εκτός από την ιστορία των διακλαδώσεων και το δυναμικό περιεχόμενο μέσα στο οποίο εκτελείται το branch. Αυτές οι τεχνικές δεν θα αναπτυχθούν περισσότερο σε αυτήν την εργασία αφού αποτελούν εξειδικευμένες τεχνικές που δεν μας αφορούν στην συγκεκριμένη φάση.

2.5.2 Τεχνικές ροής δεδομένων καταχωρητών (register data flow techniques)

Οι τεχνικές ροής δεδομένων καταχωρητών αφορούν την αποτελεσματική εκτέλεση των εντολών τύπου ALU (ή αλλιώς καταχωρητή-καταχωρητή, register-register). Τυπικά μία εντολή ALU προσδιορίζει μία δυαδική πράξη, δύο καταχωρητές εισόδου που αποτελούν τα ορίσματα της εντολής και έναν καταχωρητή προορισμό όπου πρέπει να τοποθετηθεί το αποτέλεσμα της πράξης. Μία εντολή ALU μπορεί να παρασταθεί με τον εξής τρόπο: $R_i \leftarrow F_n(R_j,R_k)$, η εκτέλεση της οποίας προυποθέτει την διαθεσιμότητα των α) F_n , που αποτελεί τη λειτουργική μονάδα (functional unit), β) R_j και R_k , που αποτελούν τους δύο καταχωρητές εισόδου (ορίσματα) και γ) R_i , που αποτελεί τον καταχωρητή προορισμού. Αν η λειτουργική μονάδα δεν είναι διαθέσιμη τότε υπάρχει μία δομική εξάρτηση που μπορεί να οδηγήσει σε έναν δομικό κίνδυνο. Αν ο ένας ή και οι δύο καταχωρητές εισόδου δεν είναι διαθέσιμοι τότε μπορεί να προκληθεί κίνδυνος εξαιτίας αληθινής εξάρτησης δεδομένων. Αν τέλος δεν είναι διαθέσιμος ο καταχωρητής προορισμού τότε μπορεί να προκληθεί ένας κίνδυνος εξαιτίας αντιεξάρτησης ή εξάρτησης εξόδου.

Ακολουθούν οι σημαντικότερες τεχνικές ροής δεδομένων καταχωρητών.

2.5.2.1 Επαναχρησιμοποίηση καταχωρητών και ψευδοεξαρτήσεις δεδομένων (register reuse and false data dependencies)

Αν οι καταχωρητές δεν επαναχρησιμοποιούνται ποτέ για την αποθήκευση ορισμάτων, τότε δεν προκύπτουν ψευδοεξαρτήσεις δεδομένων. Συνήθως αυτή η επαναχρησιμοποίηση των καταχωρητών αναφέρεται ως ανακύκλωση καταχωρητών (register recycling), και μπορεί να συμβεί σε δύο διαφορετικές μορφές, μία στατική και μία δυναμική. Η στατική μορφή οφείλεται στις ενέργειες του compiler για την βελτιστοποίηση του κώδικα. Κατά το compilation δύο είναι οι βασικές ενέργειες που επιτελούνται:

- παραγωγή κώδικα,
- διανομή καταχωρητών (register allocation).

Βασικά η παραγωγή κώδικα θεωρεί ότι υπάρχει διαθεσιμότητα ενός απεριόριστου πλήθους συμβολικών καταχωρητών στους οποίους αποθηκεύει όλα τα προσωρινά δεδομένα. Κάθε τέτοιος συμβολικός καταχωρητής χρησιμοποιείται μόνο μία φορά για αποθήκευση δεδομένων και αυτό είναι γνωστό ως κώδικας μονής ανάθεσης. Το ISA (instruction set architecture) διαθέτει όμως έναν περιορισμένο αριθμό αρχιτεκτονικών καταχωρητών και επομένως το εργαλείο διανομής καταχωρητών χρησιμοποιείται για να αντιστοιχήσει τον απεριόριστο αριθμό των συμβολικών καταχωρητών με τον περιορισμένο και σταθερό αριθμό των αρχιτεκτονικών καταχωρητών. Ο διανομέας καταχωρητών (register allocator) προσπαθεί να κρατήσει όσες το δυνατόν περισσότερες από τις προσωρινές τιμές σε καταχωρητές για την αποφυγή της ανάγκης μετακίνησης των δεδομένων σε θέσεις μνήμης και το εν συνεχεία φόρτωμα των δεδομένων αυτών από τις θέσεις αυτές. Αυτό το επιτυγχάνει με την επαναχρησιμοποίηση καταχωρητών (register reuse). Αυτό σημαίνει ότι ένας καταχωρητής επανεγγράφεται με μία νέα τιμή όταν η παλιά τιμή του δεν είναι πλέον απαραίτητη. Με αυτόν τον τρόπο κάθε καταχωρητής ανακυκλώνεται για να μπορεί έτσι να κρατάει πολλαπλές τιμές.

Το γράψιμο ενός καταχωρητή ονομάζεται συνήθως ορισμός του καταχωρητή (register definition) και το διάβασμα χρήση του καταχωρητή (use of the register). Μετά από κάθε ορισμό μπορούν να γίνουν πολλές χρήσεις ενός καταχωρητή, δηλαδή

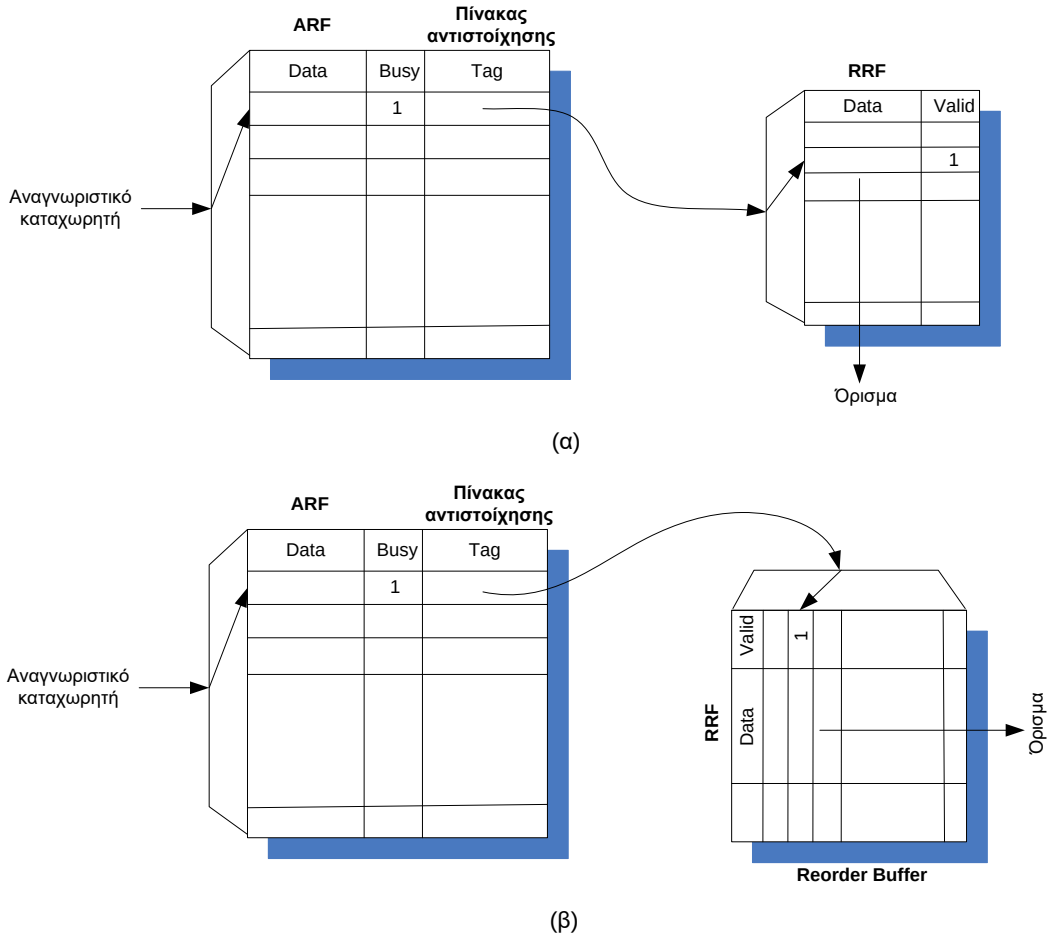
μετά από κάθε εγγραφή μπορούν να γίνουν πολλές αναγνώσεις (ίσως όμως και καμία). Η διάρκεια ανάμεσα στον ορισμό και στην τελευταία χρήση της τιμής ονομάζεται βελησνεκές ζωής της τιμής αυτής. Οι διεργασίες της διανομής καταχωρητών προσπαθούν να αντιστοιχήσουν μη επικαλυπτόμενα βελησνεκή ζωής στους ίδιους αρχιτεκτονικούς καταχωρητές και να μεγιστοποιήσουν έτσι την επαναχρησιμοποίηση καταχωρητών. Αν οι εντολές εκτελούνται ακολουθιακά και μία επανεγγραφή καταχωρητή δεν επιτρέπεται πριν από την προηγούμενη εγγραφή του ίδιου καταχωρητή τότε τα βελησνεκή ζωής που μοιράζονται τον ίδιο καταχωρητή δεν θα επικαλυφθούν ποτέ κατά τη διάρκεια της εκτέλεσης και η ανακύκλωση των καταχωρητών δεν δημιουργεί κανένα πρόβλημα. Παρ' όλα αυτά σε έναν superscalar επεξεργαστή, ειδικά λόγω της out-of-order εκτέλεσης των εντολών, οι λειτουργίες ανάγνωσης και εγγραφής των καταχωρητών μπορεί να γίνουν με διαφορετική σειρά από αυτήν που υποδηλώνει το πρόγραμμα. Έτσι η ένα προς ένα αντιστοίχηση ανάμεσα στις τιμές και τους καταχωρητές δεν υφίσταται πλέον. Για να επιτευχθεί λοιπόν η ορθότητα του προγράμματος όλες οι αντιεξαρτήσεις και οι εξαρτήσεις εξόδου (anti- και output dependencies) πρέπει να ανακαλυφθούν και να εξαλειφθούν. Η out-of-order ανάγνωση/εγγραφή των καταχωρητών μπορεί να επιτραπεί όσο όλες οι αντιεξαρτήσεις και οι εξαρτήσεις εξόδου εξαλείφονται. Αν μία αντιεξάρτηση (antidependency, WAR(write after read)), δηλαδή μία εγγραφή ενός καταχωρητή μετά από ανάγνωσή του, υφίσταται ανάμεσα σε ένα ζευγάρι εντολών τότε η δεύτερη σε σειρά εντολή (αυτή που ενημερώνει τον καταχωρητή) πρέπει να περιμένει μέχρι η προπορευόμενη εντολή να διαβάσει τον εν λόγω καταχωρητή. Αν μία εξάρτηση εξόδου (output dependency, WAW(write after write)), δηλαδή μία εγγραφή ενός καταχωρητή μετά από προηγούμενη εγγραφή του, υφίσταται ανάμεσα σε ένα ζευγάρι εντολών τότε η δεύτερη σε σειρά εντολή (αυτή που ενημερώνει τον καταχωρητή) πρέπει να καθυστερήσει μέχρι η προπορευόμενη εντολή να ενημερώσει πρώτα τον καταχωρητή. Αυτές βέβαια οι καθυστερήσεις μπορεί να οδηγήσουν σε σημαντική μείωση της απόδοσης του pipeline. Αυτά τα προβλήματα μπορούν να αντιμετωπιστούν αποδοτικά με την τεχνική μετονομασίας καταχωρητών που θα αναπτυχθεί στην επόμενη παράγραφο.

2.5.2.2 Τεχνικές μετονομασίας καταχωρητών (register renaming techniques)

Ένας περισσότερο δυναμικός τρόπος αντιμετώπισης των ψευδοεξαρτήσεων είναι η δυναμική ανάθεση ονομάτων στους πολλαπλούς ορισμούς ενός αρχιτεκτονικού καταχωρητή με την οποία τερματίζεται η παρουσία των διαφόρων ψευδοεξαρτήσεων. Αυτό ονομάζεται μετονομασία καταχωρητών και απαιτεί τη χρήση ορισμένων hardware μηχανισμών κατά τη διάρκεια της εκτέλεσης για να αποτρέψει τα αποτελέσματα της ανακύκλωσης καταχωρητών με την αναπαραγωγή της ένα προς ένα αντιστοιχίας ανάμεσα στους καταχωρητές και τις τιμές για όλες τις εντολές που μπορεί να είναι ταυτόχρονα σε εκτέλεση (αυτές οι εντολές λέγεται ότι είναι in flight). Έτσι με τη μετονομασία καταχωρητών δεν υφίστανται πλέον αντιεξαρτήσεις ή εξαρτήσεις εξόδου και πλέον όλες οι εντολές που αρχικά είχαν ψευδοεξαρτήσεις μεταξύ τους μπορούν να εκτελούνται παράλληλα.

Ο πιο συνηθισμένος τρόπος μετονομασίας καταχωρητών είναι η χρήση ενός ξεχωριστού αρχείου καταχωρητών μετονομασίας (RRF, Renaming Register File) που έρχεται να προστεθεί στο ήδη υπάρχων αρχείο αρχιτεκτονικών καταχωρητών (ARF, Architected Register File). Πολύ συχνά το RRF είναι μία «σκία» του ARF, ένα απλό αντίγραφό του. Αυτό επιτρέπει σε κάθε αρχιτεκτονικό καταχωρητή να μετονομαστεί μία φορά. Στις περισσότερες περιπτώσεις όμως δίνεται η δυνατότητα σε κάθε

καταχωρητή του RRF να μετονομάσει οποιονδήποτε από τους καταχωρητές του ARF αυξάνοντας έτσι την αποτελεσματικότητα της τεχνικής (υπάρχει όμως ανάγκη ύπαρξης πίνακα αντιστοίχησης όπου κρατούνται οι απαραίτητοι δείκτες). Υπάρχουν φυσικά και άλλες επιλογές, όπως η ύπαρξη του RRF μέσα στον reorder buffer. Σε αυτήν την περίπτωση υπάρχουν επιπλέον bits σε κάθε στοιχείο του reorder buffer που δίνει την δυνατότητα να υπάρχει ένας καταχωρητής μετονομασίας για κάθε εντολή in flight. Αυτό είναι και το σενάριο χειρότερης κατάστασης που μπορεί να συμβεί (worst case scenario). Όλα αυτά φαίνονται καλύτερα στο Σχήμα 2.8 που ακολουθεί.



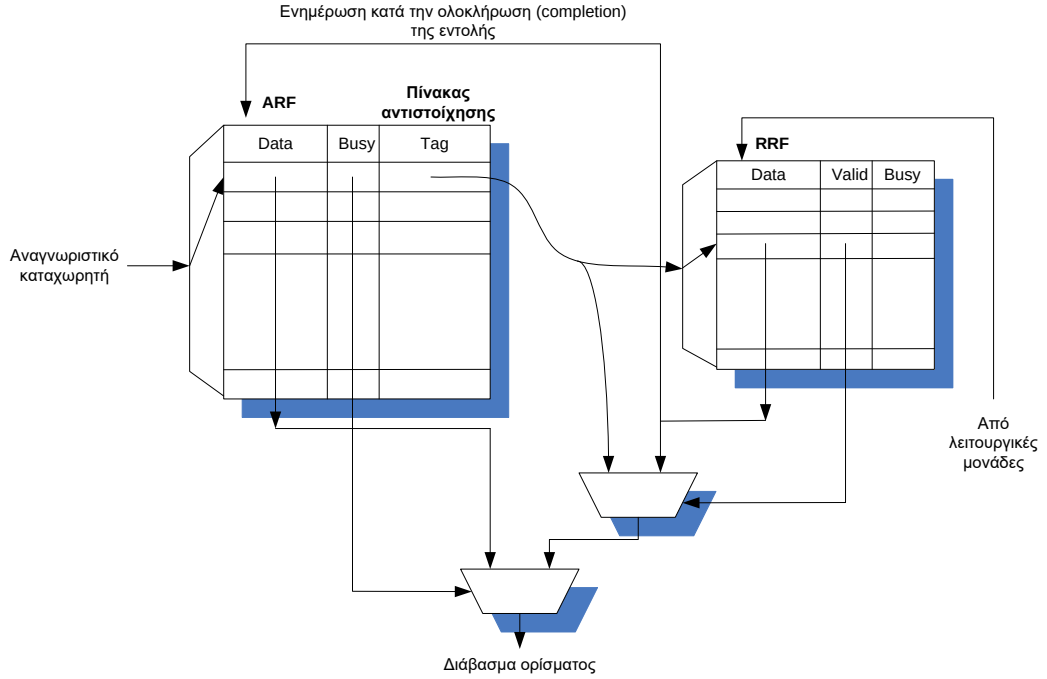
Σχήμα 2.8 Υλοποιήσεις του RRF: (α) Μόνο του (β) Προσαρτημένο στον Reorder Buffer.

Η μετονομασία καταχωρητών εμπεριέχει τρεις λειτουργίες όπως φαίνεται και στο Σχήμα 2.9:

- ανάγνωση εισόδων.
- διάθεση πόρων για τον καταχωρητή προορισμό.
- ενημέρωση καταχωρητή.

Η πρώτη λειτουργία τυπικά γίνεται κατά το στάδιο decode. Όταν μία εντολή αποκωδικοποιηθεί, τα αναγνωριστικά (specifiers) των καταχωρητών ορισμάτων της εντολής χρησιμοποιούνται για να δείξουν (με χρήση των αναγνωριστικών ως κανονικοί δείκτες) μέσα στο πολύπορτο ARF για να έρθουν οι καταχωρητές ορίσματα. Τρεις πιθανότητες υπάρχουν για κάθε καταχωρητή όρισμα. Αν το busy bit (είναι το bit εκείνο που δείχνει αν ο καταχωρητής εγγράφεται από κάποια εντολή (όταν είναι 1) ή όχι (όταν είναι 0) και υπάρχει σε κάθε εγγραφή του ARF) δεν έχει τεθεί (είναι δηλαδή ίσο με το μηδέν) τότε δεν υπάρχει εν ενεργεία εγγραφή πάνω

στον καταχωρητή και επομένως ο αρχιτεκτονικός καταχωρητής περιέχει την τιμή του ορίσματος που απαιτείται. Αν το busy bit έχει τεθεί, κάτι που υποδηλώνει εγγραφή από εντολή που ακόμα εκτελείται, το περιεχόμενο της αντίστοιχης εγγραφής του ARF δεν περιέχει το σωστό δεδομένο και επομένως γίνεται πρόσβαση στην αντίστοιχη εγγραφή του πίνακα αντιστοίχησης για να δοθεί η ετικέτα μετονομασίας. Αυτή η ετικέτα μετονομασίας καθορίζει έναν καταχωρητή μετονομασίας και χρησιμοποιείται ως δείκτης μέσα στο RRF. Όταν βρεθεί η αντίστοιχη εγγραφή του RRF δύο πιθανότητες υπάρχουν. Αν το valid bit (που δείχνει αν η εγγραφή του RRF είναι έγκυρη) έχει τεθεί τότε αυτό σημαίνει ότι η εντολή ενημέρωσης του καταχωρητή έχει ήδη ολοκληρώσει την εκτέλεσή της αλλά περιμένει να γίνει completed. Σε αυτή την περίπτωση το όρισμα εισόδου είναι έτοιμο και παίρνεται από την αντίστοιχη εγγραφή του RRF. Αν το valid bit δεν έχει τεθεί τότε υπάρχει μία εν εξελίξει ενημέρωση αυτού του καταχωρητή μετονομασίας και η ετικέτα μετονομασίας περνά στον αντίστοιχο σταθμό αναμονής αντί για την τιμή του ορίσματος. Αυτή η ετικέτα χρησιμοποιείται στη συνέχεια για την απόκτηση του ορίσματος, όταν αυτό γίνει διαθέσιμο.



Σχήμα 2.9 Λειτουργίες μετονομασίας καταχωρητών: ανάγνωση εισόδων, διάθεση πόρων για τον καταχωρητή προορισμό και ενημέρωση καταχωρητή.

Η λειτουργία της διάθεσης πόρων για τον καταχωρητή προορισμού γίνεται επίσης κατά το στάδιο decode και έχει τρεις υπολειτουργίες:

- την ανάθεση του busy bit,
- την ανάθεση ετικέτας,
- την ενημέρωση του πίνακα μετονομασίας.

Κατά την αποκωδικοποίηση μιας εντολής το αναγνωριστικό του καταχωρητή προορισμού χρησιμοποιείται ως δείκτης για να επιλεγεί ο κατάλληλος καταχωρητής μέσα στο ARF. Το busy bit της εγγραφής του καταχωρητή αυτού πρέπει να τεθεί στη μονάδα αφού υπάρχει πλέον εγγραφή σε εξέλιξη. Επίσης ο αρχιτεκτονικός αυτός καταχωρητής θα πρέπει να μετονομαστεί και να του ανατεθεί ένας καταχωρητής

μετονομασίας. Γίνεται λοιπόν επιλογή ενός καταχωρητή μετονομασίας (του οποίου το busy bit πρέπει να είναι μηδέν) και το busy bit της συγκεκριμένης εγγραφής του RRF που αντιστοιχεί σε αυτόν τον καταχωρητή τίθεται στην μονάδα έτσι ώστε αυτός ο καταχωρητής μετονομασίας να μην χρησιμοποιηθεί από κάποια άλλη εντολή προτού ολοκληρωθεί η τρέχουσα εντολή. Ακόμα, ο δείκτης της συγκεκριμένης εγγραφής του RRF χρησιμοποιείται ως ετικέτα μετονομασίας. Αυτή η ετικέτα πρέπει να γραφτεί στην αντίστοιχη εγγραφή του πίνακα μετονομασίας για να περιέχει ο πίνακας τις σωστές πληροφορίες για τις επόμενες εντολές.

Η ενημέρωση του καταχωρητή μπορεί να συμβεί σε δύο ξεχωριστά βήματα. Όταν μία εντολή ενημέρωσης καταχωρητή τελειώσει την εκτέλεσή της το αποτέλεσμά της γράφεται στην εγγραφή του RRF που υποδηλώνεται από την ετικέτα μετονομασίας. Αργότερα, όταν η εντολή ολοκληρωθεί, το αποτέλεσμα αντιγράφεται από το RRF στο ARF. Έτσι, η διαδικασία ενημέρωσης του καταχωρητή περιλαμβάνει αρχικά την ενημέρωση του RRF και εν συνεχεία την ενημέρωση του ARF. Όταν ο καταχωρητής μετονομασίας αντιγραφεί στον αντίστοιχο αρχιτεκτονικό καταχωρητή το busy bit της εγγραφής του RRF γίνεται μηδέν έτσι ώστε ο καταχωρητής μετονομασίας να μπορεί να ξαναχρησιμοποιηθεί για να μετονομάσει άλλους αρχιτεκτονικούς καταχωρητές.

2.5.2.3 Αληθινές εξαρτήσεις δεδομένων και το όριο ροής δεδομένων (true data dependences and the data flow limit)

Μία RAW (read after write) εξάρτηση, δηλαδή μία ανάγνωση ενός καταχωρητή μετά την εγγραφή του, ονομάζεται αληθινές εξάρτηση δεδομένων λόγω της σχέσης παραγωγού-καταναλωτή ανάμεσα στις δύο εντολές. Η εντολή ανάγνωσης (εντολή καταναλωτής) δεν μπορεί να αποκτήσει το όρισμά της μέχρι τη στιγμή που η προπορευόμενη εντολή εγγραφής του καταχωρητή αυτού (που αποτελεί όρισμα της εντολής ανάγνωσης) θα παραγάγει το αποτέλεσμά της (δηλαδή θα εγγράψει τον καταχωρητή). Αυτό σημαίνει ότι όταν υπάρχει μία αληθινή εξάρτηση ανάμεσα σε δύο εντολές η δεύτερη στη σειρά εντολή θα πρέπει να περιμένει την ολοκλήρωση της πρώτης για να μπορέσει να αποκτήσει την τιμή του ορίσματος που της χρειάζεται και να συνεχίσει να εκτελείται. Αυτός ο περιορισμός δεν μπορεί να αντιμετωπιστεί με κάποιον τρόπο (όπως η μετονομασία που είδαμε ότι λύνει το πρόβλημα των ψευδοεξαρτήσεων). Σε αυτή την περίπτωση το μόνο που μπορεί να γίνει είναι η καθυστέρηση της δεύτερης εντολής μέχρι την παραγωγή του αποτελέσματος της προπορευόμενης εντολής. Για αυτό το πρόβλημα υπάρχουν γράφοι που απεικονίζουν τη ροή των δεδομένων και αναδεικνύουν καθαρά την εξάρτηση δεδομένων. Οι γράφοι αυτοί ονομάζονται γράφοι ροής δεδομένων (data dependence graph DDG). Οι κόμβοι στους γράφους αυτούς υποδηλώνουν τις εντολές και οι κατευθυνόμενες ακμές (όπου αυτές υπάρχουν) υποδηλώνουν την ύπαρξη μίας αληθινής εξάρτησης δεδομένων ανάμεσα στις δύο εντολές που ενώνει η ακμή. Το νούμερο στην ακμή δείχνει τον καταχωρητή που προκαλεί την εξάρτηση ή/και την καθυστέρηση λόγω της εξάρτησης σε κύκλους μηχανής. Το μονοπάτι με την μεγαλύτερη συνολική καθυστέρηση (που ισούται με το άθροισμα των καθυστερήσεων των ακμών) δίνει το κρίσιμο μονοπάτι του κώδικα (critical path), και δίνει ένα κάτω όριο καθυστέρησης για την εκτέλεσή του, αφού ο κώδικας δεν μπορεί να εκτελεστεί γρηγορότερα από αυτό το χρονικό διάστημα. Αυτό ονομάζεται συνήθως ως όριο ροής δεδομένων (data flow limit).

2.5.2.4 Ο κλασσικός αλγόριθμος Tomasulo (the classic Tomasulo algorithm)

Στους μοντέρνους superscalar επεξεργαστές χρησιμοποιείται πλέον για την αντιμετώπιση των διαφόρων προβλημάτων της ροής δεδομένων ο αλγόριθμος Tomasulo (Tomasulo algorithm). Ο αλγόριθμος Tomasulo αποτελείται από τρεις νέους μηχανισμούς που προστίθονται στην σχεδίαση της αρχικής μονάδας κινητής υποδιαστολής, δηλαδή στην FPU (floating point unit), των superscalar επεξεργαστών πρώτης γενιάς. Αυτοί οι τρείς μηχανισμοί είναι οι σταθμοί αναμονής (reservation stations), ο κοινός διάδρομος δεδομένων (common data bus) και οι ετικέτες καταχωρητών (register tags). Στην αρχική σχεδίαση κάθε λειτουργική μονάδα έχει έναν μοναδικό καταχωρητή στην είσοδό της που κρατάει την εντολή που εκτελείται σε κάθε δεδομένη χρονική στιγμή. Αν μία λειτουργική μονάδα είναι απασχολημένη με την εκτέλεση κάποιας εντολής τότε η έκδοση των εντολών από την στοίβα των ορισμάτων κινητής υποδιαστολής (FLOS, floating point operand stack) καθυστερεί όταν η επόμενη εντολή χρειάζεται την ίδια λειτουργική μονάδα. Με τον αλγόριθμο Tomasulo πολλαπλοί καταχωρητές, που ονομάζονται σταθμοί αναμονής τοποθετούνται στην είσοδο κάθε λειτουργικής μονάδας (όπως έχουμε ήδη αναφέρει σε προηγούμενο κεφάλαιο). Αυτοί οι σταθμοί αναμονής μπορούν να θεωρηθούν ως εικονικές λειτουργικές μονάδες. Όσο υπάρχει ένας ελεύθερος σταθμός αναμονής η στοίβα FLOS μπορεί να προωθήσει μία εντολή σε αυτό το σταθμό αναμονής ακόμα και εάν η λειτουργική μονάδα που του αντιστοιχεί είναι απασχολημένη με την εκτέλεση κάποιας άλλης εντολής.

Λόγω αυτής της διαθεσιμότητας των σταθμών αναμονής μπορεί να υπάρξει έκδοση εντολών σε λειτουργικές μονάδες ακόμα και αν όλα τα ορίσματά τους δεν είναι ακόμα διαθέσιμα. Αυτές οι εντολές μπορούν να περιμένουν στους σταθμούς αναμονής για τα ορίσματά τους και επιτρέπεται να αρχίσουν την εκτέλεσή τους μόνο όταν τα ορίσματα αυτά γίνουν διαθέσιμα. Ο κοινός διάδρομος δεδομένων (common data bus CDB) ενώνει τις εξόδους των λειτουργικών μονάδων με τους σταθμούς αναμονής έτσι ώστε να μπορεί να γίνει η γνωστοποίηση των τιμών των ορισμάτων στις εντολές που περιμένουν τις τιμές αυτές (υπάρχει ένα είδος broadcasting δηλαδή των αποτελεσμάτων των λειτουργικών μονάδων στον διάδρομο με παράλληλο έλεγχο στους σταθμούς αναμονής για τους καταχωρητές ορίσματα που τους ενδιαφέρουν οι οποίοι λατσάρουν τα δεδομένα από τον διάδρομο). Φυσικά την ώρα που γίνεται το broadcasting του αποτελέσματος από μία λειτουργική μονάδα γίνεται παράλληλα και η ενημέρωση του καταχωρητή προορισμού της εντολής που εκτελέστηκε. Υπάρχει πάντως και η περίπτωση το όρισμα μίας εντολής που περιμένει να έρχεται από μία θέση μνήμης οπότε και πρέπει να φορτωθεί στους buffers κινητής υποδιαστολής που συνδέονται με τη μνήμη (floating point buffers FLBs) αφού πρώτα γίνει μία προσπέλαση στη μνήμη. Έτσι χρειάζεται να επικοινωνεί και το FLB με τον κοινό διάδρομο.

Όταν η FLOS προωθεί μία εντολή σε μία λειτουργική μονάδα δεσμεύει έναν σταθμό αναμονής και ελέγχει την διαθεσιμότητα των ορισμάτων της εντολής. Αν ένα όρισμα είναι διαθέσιμο στους καταχωρητές κινητής υποδιαστολής (floating point registers FLRs) τότε το περιεχόμενο του καταχωρητή του FLR αντιγράφεται στον σταθμό αναμονής. Σε διαφορετική περίπτωση αντιγράφεται στον σταθμό αναμονής μία ετικέτα. Η ετικέτα δείχνει από πού πρόκειται να έρθει το όρισμα, του οποίου ο υπολογισμός της τιμής βρίσκεται σε εξέλιξη (pending operand). Αυτό το όρισμα μπορεί να έρθει από οποιαδήποτε λειτουργική μονάδα του σταδίου execute, ή από κάποιον buffer του FLB (δηλαδή από τη μνήμη). Η ετικέτα επομένως θα πρέπει να περιέχει όλες τις επιλογές, δηλαδή αν οι επιλογές είναι κ (οι επιλογές είναι το

άθροισμα των λειτουργικών μονάδων και του πλήθους των FLBs) θα πρέπει να έχει πλήθος bits ίσο με $\log_2 μ$ όπου μ η αμέσως μεγαλύτερη ή ίση του κ δύναμη του 2. Όταν ένα τέτοιο όρισμα υπολογιστεί από κάποια λειτουργική μονάδα (ή έρθει από κάποιον FLB) τότε η τιμή του μαζί με την ετικέτα του καταχωρητή οδηγείται στον κοινό διάδρομο. Οι εντολές που περιμένουν στους σταθμούς αναμονής επιβλέπουν τον κοινό διάδρομο δεδομένων και αν ανακαλύψουν ότι η ετικέτα που βρίσκεται στον διάδρομο ταιριάζει με κάποια δικιά τους ετικέτα λατσάρουν την τιμή του ορίσματος. Κάθε σταθμός αναμονής έχει δύο ορίσματα άρα πρέπει να έχει και δύο πεδία ετικέτας αφού και τα δύο ορίσματα μπορεί να είναι σε φάση υπολογισμού. Οι καταχωρητές FLRs όπως και οι καταχωρητές αποθήκευσης δεδομένων (store data buffers SDBs) πρέπει επίσης να έχουν πεδίο ετικέτας. Η ετικέτα μηδενίζεται όταν το πεδίο ορίσματος έχει την πραγματική τους τιμή. Επίσης το πρώτο όρισμα ονομάζεται sink (καταβόθρα) αφού έχει και την έννοια του αναγνωριστικού του καταχωρητή προορισμού ενώ το δεύτερο ονομάζεται source (πηγή). Τέλος υπάρχουν επιπλέον bits που υποδηλώνουν (busy bits) που δείχνουν αν τα ορίσματα είναι σε εν εξελίξει ενημέρωση από άλλες εντολές.

2.5.2.5 Δυναμικός πυρήνας εκτέλεσης (dynamic execution core)

Ο πυρήνας της out-of-order εκτέλεσης (που συχνά καλείται δυναμικός πυρήνας εκτέλεσης), χρησιμοποιώντας τον αλγόριθμο Tomasulo, μπορεί να θεωρηθεί ως μία μηχανή μικρο-ροής δεδομένων που προσπαθεί να φτάσει το όριο ροής δεδομένων της εκτέλεσης εντολών. Οι λειτουργίες που επιτελούνται σε αυτόν τον πυρήνα είναι η προώθηση των εντολών (dispatching), η εκτέλεσή τους (execution) και η ολοκλήρωσή τους (completion).

Το dispatching των εντολών αποτελείται από την μετονομασία των καταχωρητών προορισμού, τη δέσμευση των εγγραφών του σταθμού αναμονής και του reorder buffer και την προώθηση των εντολών από τον dispatch buffer στους σταθμούς αναμονής. Για να μπορεί να τελειώσει βέβαια αυτή η φάση απαιτείται η διαθεσιμότητα ενός καταχωρητή μετονομασίας, μίας εγγραφής στον αντίστοιχο σταθμό αναμονής και μίας εγγραφής στον reorder buffer. Αν κάποιο από αυτά δεν είναι διαθέσιμο τότε είναι αναγκαία η καθυστέρηση (stall) της φάσης αυτής.

Η φάση της εκτέλεσης των εντολών αποτελείται από την έκδοση των εντολών στις λειτουργικές μονάδες (issuing), την εκτέλεση αυτών των εντολών και την προώθηση των αποτελεσμάτων. Κάθε σταθμός αναμονής είναι υπεύθυνος να αναγνωρίζει τις εντολές που είναι έτοιμες να προωθηθούν στις λειτουργικές μονάδες. Με τον τρόπο που περιγράφτηκε σε προηγούμενη παράγραφο ο σταθμός αναμονής αναμένει για τα ορίσματά της εντολής τα οποία δεν είναι ακόμα έτοιμα (με έλεγχο ετικέτας του ορίσματος και λατσάρισμα της τιμής από τον κοινό διάδρομο δεδομένων όταν το όρισμα γίνει διαθέσιμο) και όταν η εντολή έχει όλα της τα ορίσματα έτοιμα ο την προωθεί στην αντίστοιχη λειτουργική μονάδα για να εκτελεστεί. Αν σε μία δεδομένη χρονική στιγμή είναι έτοιμες προς εκτέλεση πάνω από μία εντολές σε έναν σταθμό αναμονής τότε η εντολή που προωθείται επιλέγεται με συγκεκριμένο αλγόριθμο που διαφέρει από επεξεργαστή σε επεξεργαστή και από σταθμό αναμονής σε σταθμό αναμονής (μπορεί να χρησιμοποιείται διαφορετική τακτική για τις εντολές load/store από ότι για τα branches για παράδειγμα) και εξαρτάται από τη σχεδίαση. Τυπικά, από τη στιγμή που μία εντολή εκδοθεί σε μία λειτουργική μονάδα (η οποία μπορεί επίσης να έχει στάδια στην εκτέλεσή των εντολών, να είναι δηλαδή pipelined) δεν υπάρχει περαιτέρω καθυστέρηση.

Όταν τελειώσει η εκτέλεση μίας εντολής γνωστοποιεί ην ετικέτα του καταχωρητή προορισμού της καθώς επίσης και το αποτέλεσμα της εντολής στον κοινό διάδρομο. Παράλληλα γίνεται και ενημέρωση του RRF.

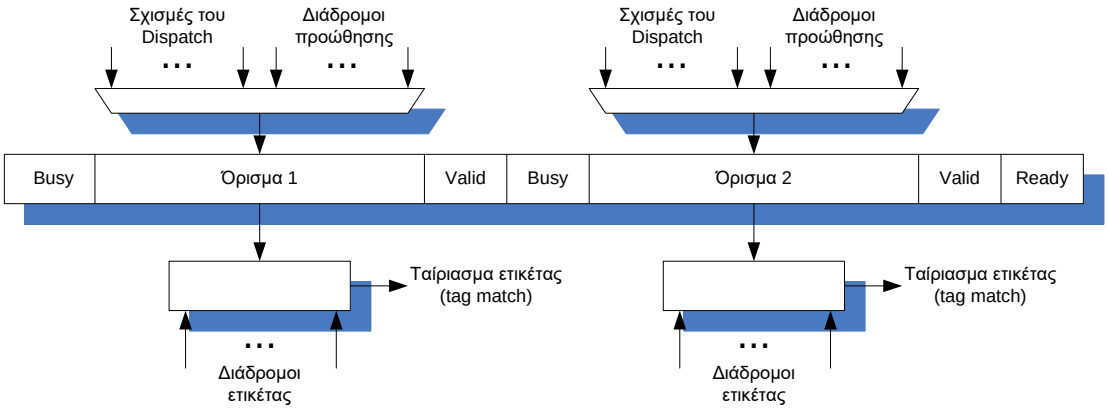
2.5.2.6 Σταθμοί αναμονής και καταχωρητής επαναδιάταξης (reservation stations and reorder buffer)

Εκτός από τις λειτουργικές μονάδες υπάρχουν δύο πολύ κρίσιμα στοιχεία του δυναμικού πυρήνα εκτέλεσης. Αυτά είναι οι σταθμοί αναμονής (reservation stations) και ο καταχωρητής επαναδιάταξης (reorder buffer, ο αγγλικός όρος είναι περισσότερο δόκιμος και θα χρησιμοποιηθεί αντί του ελληνικού όπως έχει γίνει και με άλλους όρους μέχρι στιγμής).

Υπάρχουν τρεις εργασίες που σχετίζονται με έναν σταθμό αναμονής, η προώθηση των εντολών από το στάδιο αποκωδικοποίησης στο στάδιο εκτέλεσης (dispatching), η αναμονή (waiting) και η έκδοση των έτοιμων προς εκτέλεση εντολών στις κατάλληλες λειτουργικές μονάδες (issuing). Ένας τυπικός σταθμός αναμονής φαίνεται στο Σχήμα 2.10. Κάθε εγγραφή ενός σταθμού αναμονής έχει ένα busy bit, που δείχνει αν η εγγραφή έχει δεσμευθεί από κάποια εντολή, και ένα ready bit, που δείχνει αν η εντολή της εγγραφής είναι έτοιμη για εκτέλεση, δηλαδή αν έχει έτοιμα όλα τα ορίσματά της. Το dispatching εμπεριέχει την φόρτωση μίας εντολής από τον dispatch buffer σε μία εγγραφή του σταθμού αναμονής. Ουσιαστικά αυτή η διαδικασία απαιτεί τα εξής τρία βήματα:

- την επιλογή μίας ελεύθερης (δηλαδή με busy bit ίσο με μηδέν) θέσης του σταθμού αναμονής,
- την ακόλουθη φόρτωση των ορισμάτων και/ή των ετικετών στην επιλεγμένη εγγραφή,
- τον ορισμό του busy bit της εγγραφής στη μονάδα.

Η επιλογή της ελεύθερης θέσης στον σταθμό αναμονής γίνεται από τον διανομέα (allocator) με κάποιον αλγόριθμο που εξαρτάται από τη σχεδίαση του pipeline. Κατά τα λοιπά ισχύουν αυτά που έχουν προαναφερθεί και σε προηγούμενες παραγράφους με τα πεδία που αφορούν τα ορίσματα και τις ετικέτες.



Σχήμα 2.10 Εγγραφή σταθμού αναμονής.

Η αναμονή για τα ορίσματα της εντολής γίνεται με τον τρόπο που έχει περιγραφεί σε προηγούμενη παράγραφο, δηλαδή με τον έλεγχο των ετικετών των ορισμάτων από τον σταθμό αναμονής και το λατσάρισμα των δεδομένων από τον κοινό διάδρομο δεδομένων σε περίπτωση που η ετικέτα που δημοσιοποιείται στον διάδρομο ταιριάζει με κάποια από τις δύο ετικέτες της εγγραφής του σταθμού

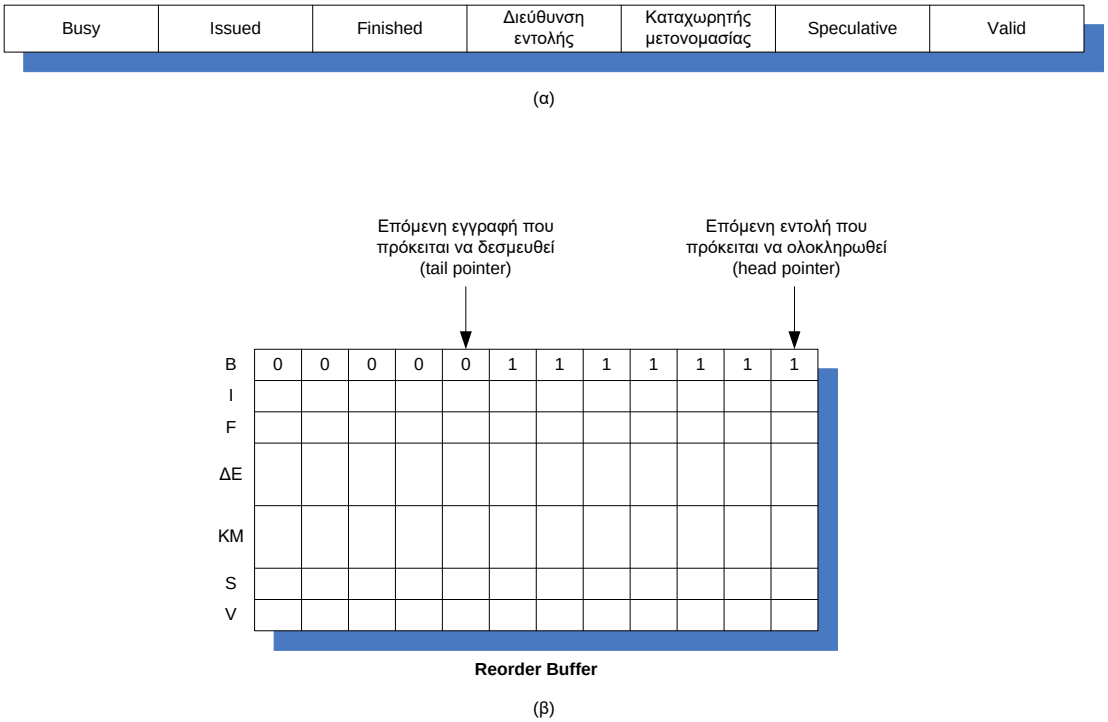
αναμονής. Όταν και τα δύο ορίσματα είναι έγκυρα τότε το ready bit τίθεται στη μονάδα και η εντολή είναι έτοιμη να εκδοθεί στην λειτουργική μονάδα.

Η έκδοση στη λειτουργική μονάδα (issuing) εμπεριέχει την επιλογή μίας έτοιμης εντολής από τον σταθμό αναμονής και την προώθησή της στην αντίστοιχη λειτουργική μονάδα. Αν υπάρχουν πολλές έτοιμες εντολές η επιλογή αυτή γίνεται με βάση κάποιον επιλεγμένο αλγόριθμο που πρέπει σε γενικές γραμμές να είναι δίκαιος (συνήθως επιλέγεται η εντολής που περίμενε την περισσότερη ώρα στον σταθμό αναμονής). Με το που επιλεγεί μία εντολή και εκδοθεί για εκτέλεση, το busy bit της αντίστοιχης εγγραφής στον σταθμό αναμονής μηδενίζεται.

Ο reorder buffer διατηρεί όλες τις εντολές που βρίσκονται in flight, δηλαδή όλες εκείνες τις εντολές που έχουν γίνει dispatched αλλά δεν έχουν ακόμα ολοκληρωθεί αρχιτεκτονικά. Αυτές οι εντολές περικλείουν τις εντολές που περιμένουν στους σταθμούς αναμονής, τις εντολές που εκτελούνται στις λειτουργικές μονάδες και τις εντολές που έχουν τελειώσει την εκτέλεσή τους αλλά περιμένουν να ολοκληρωθούν με βάση την σειρά του προγράμματος. Μία τυπική εγγραφή του reorder buffer παρουσιάζεται στο Σχήμα 2.11. Υπάρχουν αρκετά bits που υποδηλώνουν την κατάσταση μιας εντολής στις εγγραφές του reorder buffer. Αυτά τα bits είναι τα εξής:

- busy,
- issued,
- finished,
- speculative,
- valid.

Το speculative bit είναι αυτό που δείχνει αν η εντολή ανήκει σε κάποιο μονοπάτι πρόβλεψης (speculation path) λόγω κάποιου προγενέστερου branch που δεν έχει ακόμα επιλυθεί.. Τα άλλα bits έχουν αυτονόητη σημασία και μεταβάλλονται όταν η εντολή αλλάζει κατάσταση. Μόνο οι finished και non-speculative εντολές μπορούν να ολοκληρωθούν. Μία άκυρη (invalid, όταν η πρόβλεψη του speculation path της εντολής αποδειχθεί λανθασμένη) εντολή δεν ολοκληρώνεται αρχιτεκτονικά όταν βγει από τον reorder buffer. Σε κάθε εγγραφή του reorder buffer υπάρχει επίσης η διεύθυνση της εντολής (Instruction address στην I-cache) και ο καταχωρητής μετονομασίας. Όταν μία εντολή ολοκληρωθεί ο καταχωρητής μετονομασίας και η εγγραφή του reorder buffer αποδεσμεύονται. Από μία έννοια ο reorder buffer μπορεί να θεωρηθεί ως η καρδιά του κεντρικού ελέγχου του δυναμικού πυρήνα εκτέλεσης γιατί η κατάσταση όλων των in flight εντολών κρατιέται εκεί. Ο χειρισμός του reorder buffer γίνεται συνήθως με χρήση ουρών και δεικτών αλλά αυτό δεν είναι κάτι το δεσμευτικό.



Σχήμα 2.11 (α) Εγγραφή reorder buffer (β) Οργάνωση του reorder buffer.

2.5.3 Τεχνικές ροής δεδομένων μνήμης (memory data flow techniques)

Οι εντολές μνήμης είναι υπεύθυνες για τη μετακίνηση δεδομένων ανάμεσα στην κύρια μνήμη και στο αρχείο καταχωρητών, και είναι απαραίτητες για την υποστήριξη της εκτέλεσης των εντολών ALU. Οι καταχωρητές-ορίσματα που χρειάζονται από τις εντολές ALU πρέπει πρώτα να φορτωθούν από τη μνήμη. Λόγω του περιορισμένου αριθμού καταχωρητών είναι λογικό κατά τη διάρκεια της εκτέλεσης ενός προγράμματος να μην είναι δυνατόν όλα τα ορίσματα να μπορούν να κρατούνται στο αρχείο καταχωρητών. Έτσι ο compiler δημιουργεί έναν καινούριο κώδικα (ονομάζεται συχνά spill code) ο οποίος τοποθετεί προσωρινά συγκεκριμένα ορίσματα στην κύρια μνήμη και στη συνέχεια τα επαναφορτώνει όταν χρειάζονται. Αυτός ο κώδικας δημιουργείται με χρήση εντολών load/store. Οι εντολές αυτές προκαλούν γενικά μεγάλη καθυστέρηση εξαιτίας της ανάγκης του υπολογισμού της διεύθυνσης μνήμης και της προσπέλασης μίας θέσης μνήμης. Για την υποστήριξη της εικονικής μνήμης η υπολογισμένη διεύθυνση μνήμης (που ονομάζεται εικονική διεύθυνση, virtual address) χρειάζεται επίσης να μεταφραστεί σε μία φυσική διεύθυνση για να μπορεί να γίνει η προσπέλαση στην φυσική μνήμη. Οι κρυφές μνήμες ή όπως είναι ευρύτερα γνωστές cache μνήμες, είναι πολύ αποτελεσματικές στην μείωση της καθυστέρησης της προσπέλασης της κύριας μνήμης.

Ακολουθούν ορισμένες από τις τεχνικές ροής δεδομένων μνήμης.

2.5.3.1 Εντολές προσπέλασης μνήμης (memory accessing instructions)

Η εκτέλεση των εντολών ροής δεδομένων μνήμης γίνεται σε τρία βήματα:

- την παραγωγή της διεύθυνσης μνήμης,
- την μετάφραση της διεύθυνσης μνήμης,
- την προσπέλαση της μνήμης δεδομένων.

Η κύρια μνήμη ορίζεται από την αρχιτεκτονική συνόλου εντολών (ISA) του επεξεργαστή και αποτελεί μία συλλογή 2^n θέσεων μνήμης με δυνατότητα τυχαίας προσπέλασης, δηλαδή κάθε θέση μνήμης προσδιορίζεται από μία διεύθυνση των n bits και μπορεί να μπορεί να προσπελαστεί απευθείας με την ίδια καθυστέρηση. Όπως ακριβώς και το ARF, η κύρια μνήμη είναι μία αρχιτεκτονική οντότητα η οποία είναι ορατή από τις εντολές λογισμικού. Η διεύθυνση μνήμης πάντως παράγεται συνήθως με βάση έναν καταχωρητή και ένα offset που καθορίζεται από την εντολή. Έτσι είναι απαραίτητη η παραγωγή διεύθυνσης που απαιτεί την προσπέλαση του καθορισμένου καταχωρητή και την πρόσθεση της τιμής του offset.

Επιπρόσθετα απαιτείται και μετάφραση αυτής της διεύθυνσης μνήμης όταν χρησιμοποιείται εικονική μνήμη στο σύστημα. Η αρχιτεκτονική κύρια μνήμη συνθέτει το χώρο εικονικών διευθύνσεων του προγράμματος και θεωρείται από κάθε πρόγραμμα ως ο ιδιωτικός του χώρος διευθύνσεων. Η φυσική μνήμη που υπάρχει σε μία μηχανή συνθέτει το χώρο φυσικών διευθύνσεων, που μπορεί να είναι μικρότερος από το χώρο εικονικών διευθύνσεων και μπορεί επίσης να μοιράζεται από πολλά ξεχωριστά προγράμματα. Η εικονική μνήμη είναι εν ολίγοις ένας μηχανισμός που αντιστοιχίζει τον εικονικό χώρο μνήμης ενός προγράμματος με τον φυσικό χώρο μνήμης της μηχανής. Αυτός ο μηχανισμός υλοποιείται συνήθως με τη χρήση ενός πίνακα αντιστοίχησης (mapping table) και η μετάφραση γίνεται με τη χρήση αυτού του πίνακα.

Το τρίτο βήμα της επεξεργασίας των εντολών load/store είναι η προσπέλαση της μνήμης. Για τις εντολές φόρτωσης (load) τα δεδομένα διαβάζονται από μία θέση μνήμης και αποθηκεύονται σε έναν καταχωρητή, ενώ για τις εντολές αποθήκευσης (store) μία τιμή καταχωρητή αποθηκεύεται σε μία θέση μνήμης. Σε αντίθεση με τα δύο πρώτα βήματα που είναι ακριβώς τα ίδια για τα δύο είδη εντολών, το τρίτο βήμα (αυτό της προσπέλασης) είναι διαφορετικό για τις εντολές φόρτωσης και τις εντολές αποθήκευσης σε ένα superscalar pipeline.

Μία εντολή φόρτωσης προωθείται στην pipelined λειτουργική μονάδα τη στιγμή που ο καταχωρητής διεύθυνσης όρισμα είναι διαθέσιμος και η εικονική διεύθυνση παράγεται στο πρώτο στάδιο σωλήνωσης της μονάδας. Μία εντολή αποθήκευσης πρέπει να περιμένει την διαθεσιμότητα και του καταχωρητή διεύθυνσης και του καταχωρητή δεδομένων, δηλαδή και των δύο ορισμάτων της, προτού μπει στη λειτουργική μονάδα. Το δεύτερο στάδιο της σωλήνωσης στη συνέχεια μεταφράζει την εικονική διεύθυνση σε μία φυσική διεύθυνση. Αυτό γίνεται με την προσπέλαση του πίνακα μετάφρασης (translation lookaside buffer, TLB) που είναι ένας ελεγχόμενος από hardware πίνακας που περιέχει τις αντιστοιχίσεις ανάμεσα στις εικονικές διευθύνσεις και τις φυσικές διευθύνσεις. Φυσικά υπάρχει η περίπτωση η εικονική διεύθυνση που ζητείται να μεταφραστεί να ανήκει σε μία σελίδα που δεν υπάρχει στον TLB. Αυτό ονομάζεται TLB αποτυχία (TLB miss). Η αντιστοίχηση που λείπει σε αυτήν την περίπτωση προστίθεται στον TLB αφού πρώτα βρεθεί στον πίνακα σελίδων (που είναι ο πίνακας με τις σελίδες αντιστοιχήσεων και ο TLB περιέχει κάθε δεδομένη χρονική στιγμή μόνο ένα κομμάτι του). Σε περίπτωση που η αντιστοίχηση δεν βρεθεί ούτε στον πίνακα σελίδων αντιστοίχησης τότε αυτό σημαίνει ότι η αντιστοίχηση δεν υπάρχει στην κύρια μνήμη και θα προκληθεί επομένως ένα σφάλμα σελίδας (page fault). Τελικά αυτό το σφάλμα θα οδηγήσει στην διακοπή της εκτέλεσης του συγκεκριμένου προγράμματος. Μετά την επιτυχημένη μετάφραση στο δεύτερο στάδιο σωλήνωσης η εντολή φόρτωσης προσπελαύνει τη μνήμη δεδομένων κατά τη διάρκεια του τρίτου σταδίου. Στο τέλος του κύκλου μηχανής τα δεδομένα έχουν συλλεχθεί από την μνήμη και εγγράφονται είτε στον καταχωρητή μετονομασίας είτε στον reorder buffer. Σε αυτό το σημείο η εκτέλεση της εντολής

τελειώνει. Εδώ υποθέσαμε βέβαια ότι η προσπέλαση της μνήμης μπορεί να γίνει σε ένα κύκλο. Αυτό πάντως είναι εφικτό μόνο στην περίπτωση ύπαρξης cache μνήμης όπου υπάρχει παρόμοια λογική αποτυχίας προσπέλασης (data cache miss) με αυτήν που αναφέρθηκε για τον TLB και η αποτυχία αυτή μπορεί να προκαλέσει καθυστέρηση στο load/store pipeline.

Σε αντίθεση με τις εντολές φόρτωσης μία εντολή αποθήκευσης θεωρείται ότι έχει τελειώσει την εκτέλεσή της στο τέλος του δεύτερου σταδίου της σωλήνωσης όταν έχει γίνει μία σωστή μετάφραση της εικονικής διεύθυνσης σε φυσική διεύθυνση. Τα δεδομένα του καταχωρητή που πρέπει να αποθηκευθούν στη μνήμη φυλάσσονται στον reorder buffer. Τη στιγμή που η εντολή αποθήκευσης ολοκληρώνεται τα δεδομένα αυτά εγγράφονται στη μνήμη. Ο λόγος για αυτή την καθυστερημένη προσπέλαση στη μνήμη είναι η αποφυγή της πρώιμης και πιθανότατα λανθασμένης ενημέρωσης της μνήμης σε περίπτωση που η εντολή αποθήκευσης αποδειχθεί άκυρη (πρέπει να γίνει flushed) εξαιτίας μίας εξαίρεσης ή μίας λανθασμένης εκτίμησης για ένα branch. Αντίθετα, στις εντολές φόρτωσης μία τέτοια περίπτωση δεν επιδρά αρνητικά στην κατάσταση της μνήμης. Πάντως για μία εντολή αποθήκευσης μπορεί αντί για ενημέρωση της μνήμης κατά την ολοκλήρωση να γίνει μεταφορά των δεδομένων στον καταχωρητή αποθήκευσης (store buffer) κατά την ολοκλήρωση. Αυτός ο buffer είναι ουσιαστικά μία ουρά στην οποία κρατούνται οι αρχιτεκτονικά ολοκληρωμένες εντολές αποθήκευσης. Κάθε μία από αυτές τις εντολές γίνεται εν συνεχεία retired και ενημερώνει τη μνήμη όταν ο διάδρομος μνήμης είναι διαθέσιμος. Ο σκοπός αυτού του buffer είναι το να επιτραπεί σε εντολές αποθήκευσης να γίνουν retired όταν ο διάδρομος μνήμης δεν είναι απασχολημένος ούτως ώστε να δοθεί προτεραιότητα στις εντολές ανάγνωσης να απασχολήσουν τον διάδρομο μνήμης.

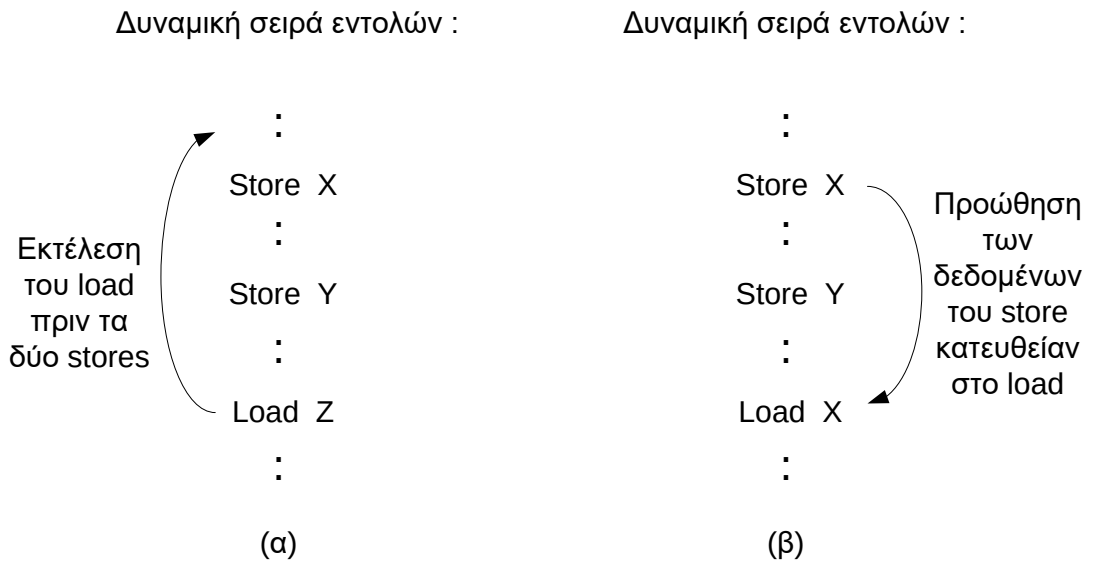
2.5.3.2 Ταξινόμηση των προσπελάσεων μνήμης (ordering of memory access)

Μία εξάρτηση δεδομένων μνήμης υπάρχει ανάμεσα σε δύο load/store εντολές αν και οι δύο αναφέρονται στην ίδια θέση μνήμης, δηλαδή αν υπάρχει συνωνυμία (aliasing) ή σύγκρουση (collision) των δύο θέσεων μνήμης. Μία εντολής φόρτωσης προκαλεί μία ανάγνωση από μία θέση μνήμης ενώ μία εντολή αποθήκευσης προκαλεί μία εγγραφή σε μία θέση μνήμης. Με παρόμοιο τρόπο με τις εξαρτήσεις δεδομένων καταχωρητών υπάρχουν και εδώ τρεις τύποι εξαρτήσεων, RAW (read after write), WAR (write after read) και WAW (write after write). Μία εντολή αποθήκευσης (φόρτωσης) ακολουθούμενη από μία εντολή φόρτωσης (αποθήκευσης) που αφορά την ίδια θέση μνήμης προκαλεί μία RAW (WAR) εξάρτηση. Δύο εντολές αποθήκευσης στην ίδια θέση μνήμης προκαλούν μία WAW εξάρτηση.

Μία αντιμετώπιση των εξαρτήσεων αυτών είναι η ταξινόμηση των load/store εντολών με βάση τον κώδικα του προγράμματος (με τις απαραίτητες καθυστερήσεις). Πάντως είναι εφικτή και η out-of-order εκτέλεση των εντολών η οποία μπορεί να αυξήσει την ταχύτητα εκτέλεσης αλλά σε κάθε περίπτωση θα πρέπει οι εντολές αποθήκευσης να εκτελούνται με τη σειρά που υποδηλώνει το πρόγραμμα, ή τουλάχιστον η μνήμη θα πρέπει να ενημερώνεται με αυτόν τον τρόπο, αλλιώς η λογική ορθότητα του προγράμματος καταστρέφεται. Αν λοιπόν οι εντολές αποθήκευσης απαιτηθεί να εκτελούνται σε σειρά προγράμματος τότε οι WAW και WAR εξαρτήσεις δεδομένων μνήμης παύουν να υφίστανται και πρέπει να αντιμετωπιστούν μόνο οι RAW εξαρτήσεις.

2.5.3.3 Load Bypassing και Load Forwarding

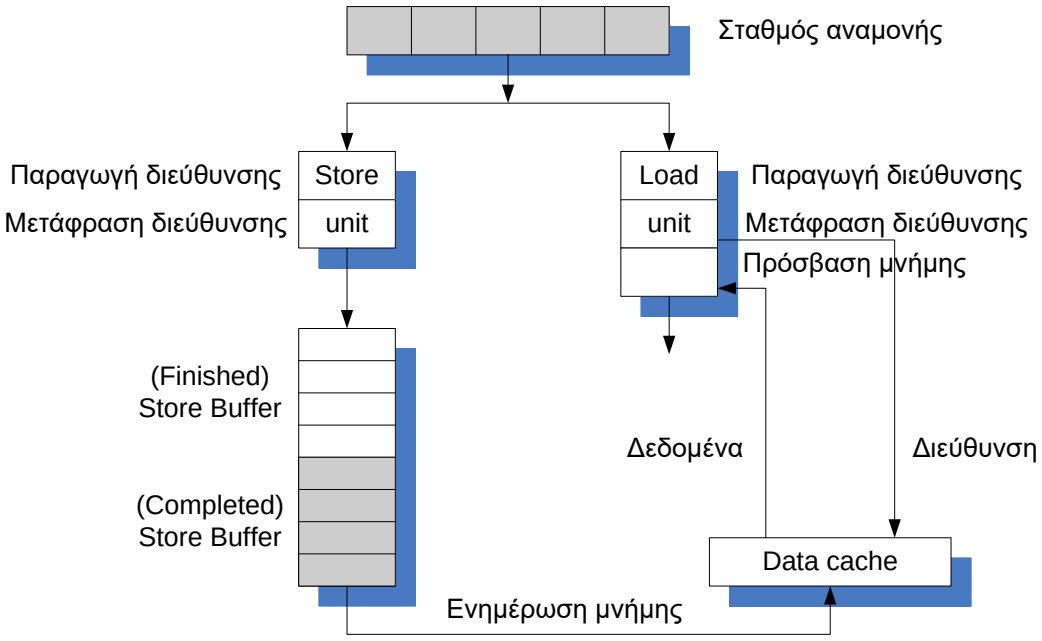
Οι εντολές φόρτωσης φαίνονται πως κάνουν ενέργειες ανάγνωσης σε θέσεις μνήμης, όμως στην πραγματικότητα κάνουν ενέργειες εγγραφής στους καταχωρητές προορισμού τους. Με τις εντολές φόρτωσης να είναι εντολές καθορισμού καταχωρητή (register defining), έχουμε συχνά το φαινόμενο να ακολουθούνται αμέσως από άλλες εντολές που σχετίζονται με τη χρήση του ίδιου εξαρτημένου καταχωρητή. Ο στόχος είναι το να επιτραπεί η εκτέλεση των εντολών φόρτωσης όσο το δυνατόν γρηγορότερα, πιθανότατα με το να αρχίσουν να εκτελούνται πριν από εντολές αποθήκευσης που προηγούνται στον κώδικα, αλλά ταυτόχρονα χωρίς να παραβιάζονται εξαρτήσεις δεδομένων μνήμης τύπου RAW και με τη μνήμη να ενημερώνεται με βάση το ακολουθιακό μοντέλο συνέπειας της μνήμης. Δύο συγκεκριμένες τεχνικές για γρήγορη out-of-order εκτέλεση των εντολών φόρτωσης αποτελούν τα load bypassing (παράκαμψη φόρτωσης) και load forwarding (προώθηση φόρτωσης). Το load bypassing επιτρέπει σε μία εντολή φόρτωσης που ακολουθεί να εκτελεστεί νωρίτερα από προπορευόμενες εντολές αποθήκευσης αν η διεύθυνση της εντολής φόρτωσης δεν συμπίπτει με αυτές των εντολών αποθήκευσης (δεν έχουμε δηλαδή RAW εξάρτηση). Στην περίπτωση που η διεύθυνση της εντολής φόρτωσης συμπίπτει με κάποια από τις διευθύνσεις των εντολών αποθήκευσης και έχουμε εξάρτηση RAW το load forwarding δίνει την δυνατότητα στην εντολή φόρτωσης να αποκτήσει τα δεδομένα της απευθείας από την εντολή αποθήκευσης που προηγείται (και με την οποία υπάρχει η εξάρτηση) χωρίς έτσι να αναγκαστεί να προσπελάσει τη μνήμη. Αυτά φαίνονται και στο ακόλουθο σχήμα, το Σχήμα 2.12:



Σχήμα 2.12 Πρώιμη εκτέλεση των load εντολών: (α) Load Bypassing (β) Load Forwarding.

Ας θεωρήσουμε την οργάνωση του Σχήματος 2.13. Σε αυτήν υπάρχει μία μονάδα αποθήκευσης (store unit, με δύο στάδια σωλήνωσης) και μία μονάδα φόρτωσης (load unit, με τρία στάδια σωλήνωσης). Και οι δύο τροφοδοτούνται από έναν κοινό σταθμό αναμονής, ενώ για αρχή θεωρούμε ότι οι εντολές load/store έρχονται με τη σειρά του προγράμματος. Η μονάδα αποθήκευσης υποστηρίζεται επίσης από έναν store buffer. Η μονάδα φόρτωσης και ο store buffer έχουν πρόσβαση στην cache μνήμη δεδομένων (data cache, D-cache).

Μία εντολή αποθήκευσης μπορεί να είναι σε μία από αρκετές καταστάσεις ενώ είναι in flight. Όταν η εντολή προωθηθεί στον σταθμό αναμονής δεσμεύεται για αυτήν μία εγγραφή στον reorder buffer, ενώ η εντολή περιμένει στον σταθμό αναμονής μέχρι να γίνουν διαθέσιμα όλα τα ορίσματά της και να εκδοθεί στην pipelined μονάδα επεξεργασίας. Όταν η διεύθυνση μνήμης παραχθεί και μεταφραστεί επιτυχημένα, η εντολή θεωρείται ότι έχει τελειώσει την εκτέλεσή της και τοποθετείται στο finished τμήμα του store buffer (ενώ ενημερώνεται αντίστοιχα και ο reorder buffer). Ο store buffer λειτουργεί ως ουρά με δύο τμήματα, το finished (με τις εντολές αποθήκευσης που έχουν τελειώσει την εκτέλεσή τους αλλά δεν έχουν ολοκληρωθεί ακόμα αρχιτεκτονικά) και το completed (με τις ολοκληρωμένες αρχιτεκτονικά εντολές αποθήκευσης που όμως δεν έχουν ενημερώσει ακόμα τη μνήμη). Η υλοποίηση της ουράς και των άκρων των τμημάτων μπορεί να γίνει με κατάλληλους δείκτες ή με κατάλληλα bit κατάστασης. Μία εντολή αποθήκευσης στο finished κομμάτι μπορεί να είναι σε μονοπάτι πρόβλεψης, δηλαδή speculative εντολή, και τότε αν ανακαλυφθεί λάθος πρόβλεψη θα πρέπει να διαγραφεί από τον store buffer. Όταν μία finished εντολή γίνει completed από τον reorder buffer τότε η εντολή περνάει από το finished στο completed κομμάτι του store buffer. Όταν μία αρχιτεκτονικά ολοκληρωμένη εντολή αποθήκευσης βγει τελικά από τον store buffer και ενημερώσει τη μνήμη, θεωρείται πλέον retired. Αν επομένως δει κάποιος την εντολή από την πλευρά της κατάστασης της μνήμης, μία εντολή αποθήκευσης τελειώνει πραγματικά την εκτέλεσή της όταν γίνει retired.



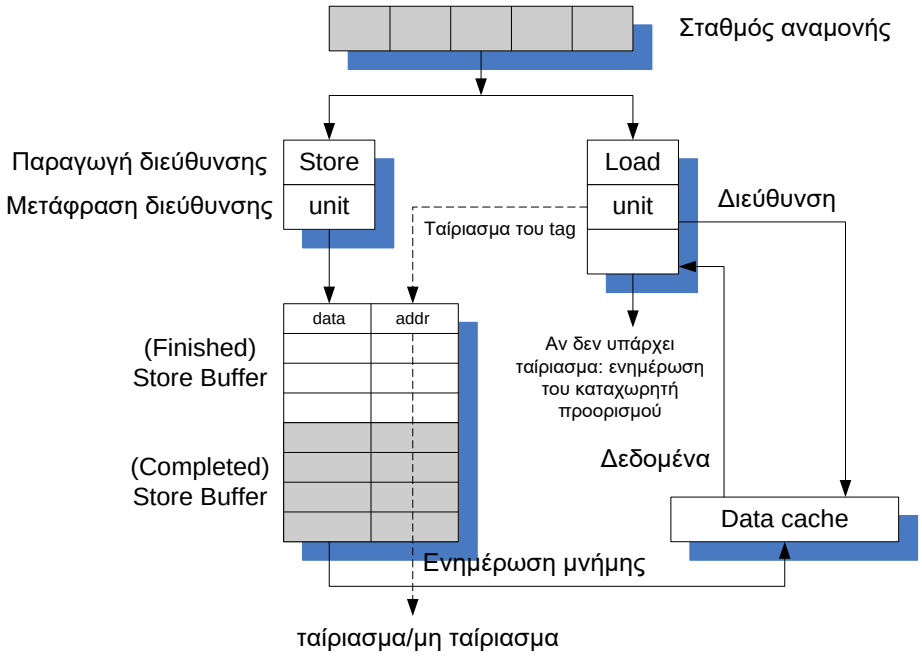
Σχήμα 2.13 Μηχανισμοί επεξεργασίας των load/store εντολών: Ξεχωριστές load και store μονάδες με in-order issuing από έναν κοινό σταθμό αναμονής.

Μία εντολή φόρτωσης θεωρείται ότι παρακάμπτει (bypass) μία προπορευόμενη εντολή αποθήκευσης αν διαβάσει από τη μνήμη προτού η προπορευόμενη εντολή γράψει στη μνήμη. Πριν λοιπόν επιτραπεί στην εντολή αυτή να εκτελέσει το διάβασμα της μνήμης πρέπει να εξακριβωθεί ότι δεν υπάρχει σύγκρουση διεύθυνσης με τις προπορευόμενες εντολές αποθήκευσης που είναι ακόμα in flight. Υποθέτοντας in-order issuing των εντολών load/store από τον σταθμό αναμονής, όλες αυτές οι εντολές αποθήκευσης πρέπει να βρίσκονται στον store

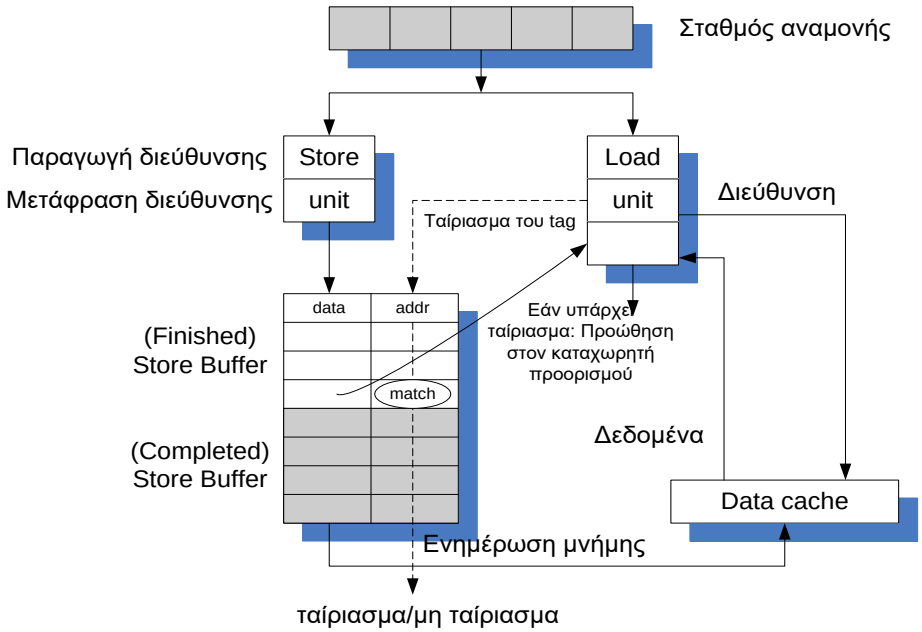
buffer. Ο έλεγχος των διευθύνσεων γίνεται με τη βοήθεια του store buffer, αφού σε κάθε στοιχείο μπορεί να υπάρχει ένα πεδίο ετικέτας που να περιέχει την διεύθυνση μνήμης της εντολής αποθήκευσης. Όταν λοιπόν το πεδίο διεύθυνσης της εντολής φόρτωσης είναι διαθέσιμο μπορεί εύκολα να γίνει ο απαραίτητος έλεγχος για τυχόν aliasing. Αν υπάρχει ταίριασμα διευθύνσεων τότε δεν επιτρέπεται η out-of-order εκτέλεση της εντολής φόρτωσης και δεν έχουμε load bypassing, διαφορετικά η εντολή φόρτωσης εκτελείται νωρίτερα από τις εντολές αποθήκευσης.

Η τεχνική του load forwarding μετατρέπει και συμπληρώνει την λογική του load bypassing. Στην περίπτωση που αναφέραμε πριν όπου κατά τη διάρκεια του ελέγχου για RAW εξάρτηση έχουμε ταίριασμα διευθύνσεων (δηλαδή αν η διεύθυνση του load συμπίπτει με τη διεύθυνση κάποιου από τα προπορευόμενα stores) υπάρχει η δυνατότητα προώθησης των δεδομένων από την εντολή αποθήκευσης στην εντολή φόρτωσης χωρίς η εντολή φόρτωσης να αναγκαστεί να προσπελάσει τη μνήμη. Το μόνο που απαιτείται είναι η προώθηση του πεδίου δεδομένων της εγγραφής του store buffer όπου βρέθηκε το aliasing στο πεδίο του μετονομασμένου καταχωρητή προορισμού της εντολής φόρτωσης. Για όλες αυτές τις λειτουργίες χρειάζονται φυσικά ορισμένες προσθήκες hardware στις οντότητες που περιγράφτηκαν.

Φυσικά υπάρχει γενικά και η δυνατότητα της υλοποίησης των τεχνικών αυτών με out-of-order issuing των load/store εντολών αλλά με σαφώς πιο πολύπλοκη σχεδίαση. Ακολουθούν τα σχήματα που δείχνουν τις τεχνικές όταν έχουμε in-order issuing (Σχήματα 2.14 και 2.15).



Σχήμα 2.14 Load Bypassing.



Σχήμα 2.15 Load Forwarding.

3 ΕΠΕΞΕΡΓΑΣΤΗΣ MICROPROC

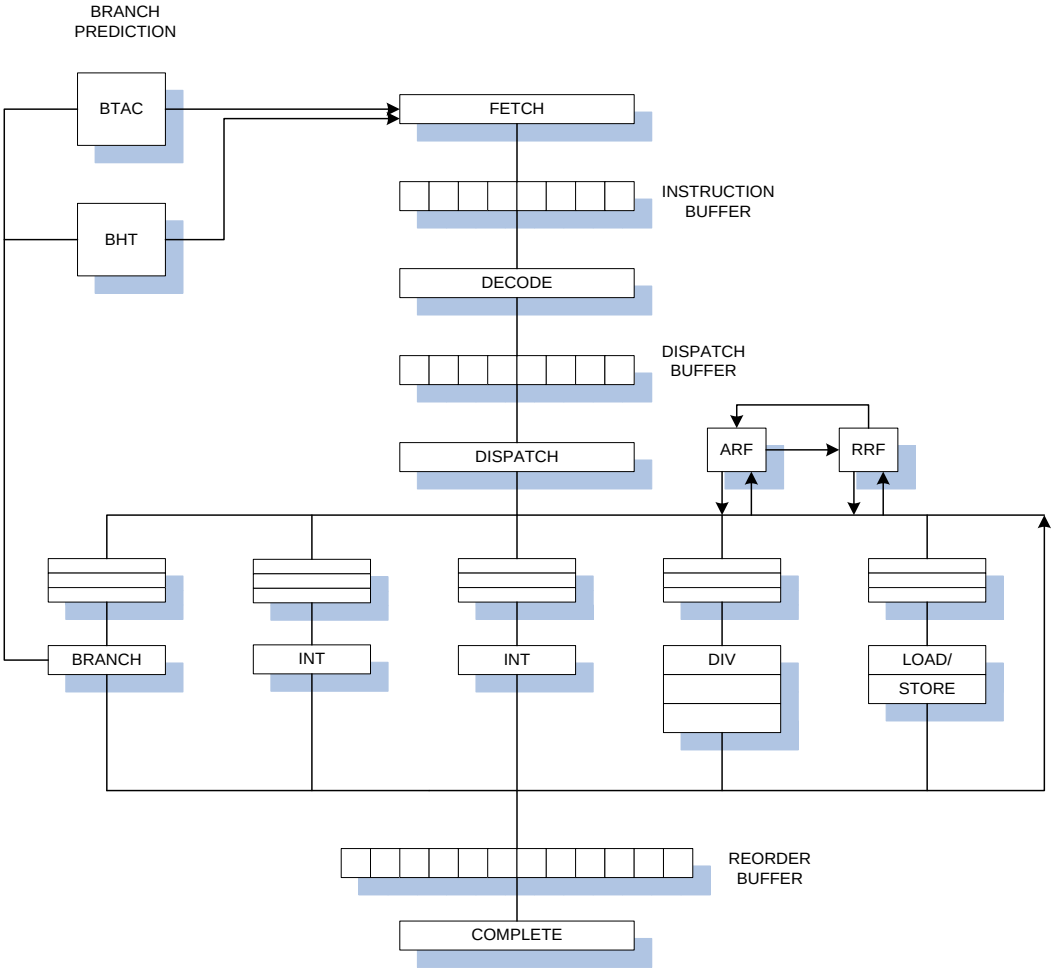
Σε αυτό το κεφάλαιο της διπλωματικής θα παρουσιαστεί ο επεξεργαστής MICROPROC ο οποίος σχεδιάστηκε σε επίπεδο μικροαρχιτεκτονικής με βάση τις τεχνικές σχεδίασης που αναλύθηκαν στο προηγούμενο κεφάλαιο. Φυσικά η σχεδίαση διαφέρει σε ορισμένα σημεία από αυτά που αναφέρθηκαν αλλά πρόκειται για διαφορές σε λεπτομέρειες σχεδίασης και τρόπους υλοποίησης χωρίς να αλλάζουν τα γενικά στοιχεία της σχεδίασης των superscalar επεξεργαστών τύπου RISC όπως είναι ο MICROPROC. Εδώ θα περιγραφεί η αρχιτεκτονική συνόλου εντολών (ISA) του επεξεργαστή και η μικροαρχιτεκτονική σε υψηλό επίπεδο. Η περιγραφή της μικροαρχιτεκτονικής του MICROPROC σε υψηλό εμπεριέχει την αναφορά των σταδίων του superscalar pipeline και την περιγραφή των τεχνικών που εφαρμόζονται και της αλληλεπίδρασης των διαφόρων δομικών μονάδων χωρίς όμως να δοθεί έμφαση σε θέματα χρονισμών και στις ειδικές λεπτομέρειες κάθε σταδίου.

3.1 Γενική περιγραφή

Ένα απλοποιημένο διάγραμμα της αρχιτεκτονικής του επεξεργαστή φαίνεται στο Σχήμα 3.1. Στο διάγραμμα αυτό απεικονίζονται τα στάδια του υπερβαθμωτού επεξεργαστή, οι buffers, οι μονάδες εκτέλεσης, οι σταθμοί αναμονής, τα register files (αρχιτεκτονικό και μετονομασίας) και οι μνήμες πρόβλεψης διακλάδωσης.

Στο στάδιο fetch φορτώνεται ένας αριθμός εντολών (με βάση τον μετρητή προγράμματος, δηλαδή τον PC) από την instruction cache. Οι εντολές αποθηκεύονται στον instruction buffer, ο οποίος έχει διπλάσιο μέγεθος από τον μέγιστο αριθμό εντολών που φορτώνονται ανά κύκλο. Ο αριθμός των εντολών που ανακτώνται από την instruction cache ανά κύκλο, καθορίζεται από τη γραμμή της instruction cache.

Επειδή έχουμε 4-way υπερβαθμωτό επεξεργαστή και 32bit σύνολο εντολών τύπου RISC, ορίζουμε το μέγεθος της γραμμής 128bit, το μέγιστο bandwidth ανάκτησης εντολών 4 ανά κύκλο και το μέγεθος του buffer 8bytes. Η ύπαρξη του instruction buffer κατά κάποιο τρόπο αποσυμπλέκει τον μηχανισμό ανάκτησης εντολών από το υπόλοιπο pipeline και επιτρέπει στο τελευταίο να λειτουργεί με το μέγιστο ρυθμό.



Σχήμα 3.1 Αρχιτεκτονική του επεξεργαστή MICROPROC.

Στο στάδιο decode γίνεται αποκωδικοποίηση των εντολών. Επειδή έχουμε σύνολο εντολών τύπου RISC η αποκωδικοποίηση μίας εντολής είναι απλή. Επειδή δεν αποκωδικοποιούμε μία εντολή ανά κύκλο, αλλά 4, στο στάδιο αυτό πρέπει επιπλέον να εντοπισθούν οι εξαρτήσεις (τύπου RAW) μεταξύ αυτών. Στο στάδιο αυτό μπορούν να διαβαστούν και τα ορίσματα από το register file (τιμές ή ετικέτες). Το σύνολο των εντολών που αποκωδικοποιούνται τοποθετείται στον dispatch buffer.

Το στάδιο dispatch μαζί με τα στάδια εκτέλεσης και ολοκλήρωσης αποτελούν την καρδιά του υπερβαθμωτού επεξεργαστή. Οι μονάδες εκτέλεσης αποτελούν παράλληλες ανεξάρτητες σωληνώσεις με διαφορετικό αριθμό σταδίων η κάθε μία. Ο μηχανισμός dispatch και εκτέλεσης λειτουργεί με βάση τον αλγόριθμο Tomasulo. Το στάδιο dispatch επεξεργάζεται τις εντολές με τη σειρά και για κάθε εντολή που έχει καταχωρητή προορισμού δεσμεύει έναν καταχωρητή μετονομασίας. Διατηρείται μία αντιστοίχιση αρχιτεκτονικού καταχωρητή – καταχωρητή μετονομασίας. Αυτή η διαδικασία λέγεται μετονομασία καταχωρητή (register renaming) και έχει ήδη εξηγηθεί σε προηγούμενο κεφάλαιο. Αφού γίνει μετονομασία (αν είναι απαραίτητη) η εντολή τοποθετείται στον σταθμό αναμονής της αντίστοιχης προς το opcode (opcode

είναι το 5-bito αναγνωριστικό πεδίο της εντολής, δηλαδή υποδηλώνει για ποια εντολή πρόκειται) μονάδας εκτέλεσης. Μία θέση του σταθμού αναμονής περιέχει τον τύπο της εντολής, τα ορίσματά της (αν δεν είναι διαθέσιμα περιέχει ετικέτες (καταχωρητές μετονομασίας)) και την ετικέτα (όνομα καταχωρητή μετονομασίας). Μία εντολή που βρίσκεται σε ένα σταθμό αναμονής και περιέχει ετικέτα αντί για την τιμή ενός ορίσματος, δεν μπορεί να εκτελεστεί αν δεν γίνει διαθέσιμη η τιμή του ορίσματος. Κατά το τέλος της εκτέλεσης μίας εντολής με καταχωρητή προορισμού, το αποτέλεσμα και η ετικέτα (tag) μεταδίδεται σε ένα διάδρομο, στον οποίο ακούν όλοι οι σταθμό αναμονής. Έτσι, όταν ο σταθμός αναμονής περιέχει εγγραφή με ετικέτα ίδια με αυτή που μεταδίδεται στο διάδρομο, λαμβάνει την αντίστοιχη τιμή και καθιστά την αντίστοιχη εντολή έτοιμη προς εκτέλεση (αν δεν έχει ετικέτα και για το άλλο όρισμα βέβαια). Μία έτοιμη εντολή (ready) μπορεί να εκτελεστεί. Αν υπάρχουν περισσότερες της μίας έτοιμης εντολής, με βάση κάποιο αλγόριθμο επιλογής (πχ. LRU) επιλέγεται η κατάλληλη και εκδίδεται στη μονάδα εκτέλεσης (φάση issue). Για κάποιους τύπους μονάδας εκτέλεσης είναι απαραίτητη ή διευκολύνει την υλοποίηση in order issue (πχ. διακλάδωση, load/store μονάδα). Όταν η εντολή τελειώσει (φάση finish) μεταδίδεται στο εν λόγω διάδρομο η ετικέτα μετονομασίας και η τιμή υπολογισμού, για να σαρωθεί από κάποιον «ενδιαφερόμενο» σταθμό αναμονής, αλλά και για να ενημερωθεί το renaming register file. Το τελείωμα της εκτέλεσης μίας εντολής δε σημαίνει ότι η εντολή έχει ολοκληρωθεί. Μία εντολή ολοκληρώνεται (αρχιτεκτονικά) όταν ενημερωθεί το αρχιτεκτονικό register file. Και αυτό γιατί μία εντολή που τελείωσε την εκτέλεσή της και έγραψε τον αντίστοιχο καταχωρητή μετονομασίας, μπορεί να βρίσκεται σε ένα υποθετικό μονοπάτι εκτέλεσης με βάση κάποια πρόβλεψη διακλάδωσης που έγινε. Αυτές οι εντολές πρέπει να απομακρυνθούν και ο επεξεργαστής να ανακάμψει από την λανθασμένη υπόθεση. Η ολοκλήρωση των εντολών γίνεται με τη σειρά (in order), εν αντιθέσει με την εκτέλεση που γίνεται out of order.

Κατά το dispatch μαζί με τη δρομολόγηση μίας εντολής στον κατάλληλο σταθμό αναμονής, δεσμεύεται για αυτήν και μία εγγραφή στον reorder buffer. Ο reorder buffer είναι ο σημαντικότερος buffer στον υπερβαθμωτό επεξεργαστή και στην ουσία διατηρεί όλες τις εντολές που βρίσκονται in flight (μαζί με την κατάστασή τους). Περιέχει όλες τις εντολές που έχουν γίνει dispatch και δεν έχουν γίνει complete. Υλοποιείται σαν μία ουρά fifo. Εγγραφές εισάγονται στην ουρά με τη σειρά κατά το dispatch και βγαίνουν απ' την ουρά με τη σειρά κατά το complete. Για κάθε εγγραφή περιέχεται πληροφορία για την κατάσταση της εντολής (wait, issued, executing, finished), για την ετικέτα μετονομασίας, το μονοπάτι υποθετικής εκτέλεσης ή μονοπάτι πρόβλεψης όπως αναφέρθηκε στο προηγούμενο κεφάλαιο (αν έχει γίνει πρόβλεψη διακλάδωσης) κα.

3.2 Σύνολο εντολών επεξεργαστή MICROPROC (MICROPROC ISA)

Το σύνολο εντολών του MICROPROC αποτελείται από τις βασικές εντολές του MIPS. Πρόκειται για RISC σύνολο εντολών. Το μοντέλο προγραμματισμού περιλαμβάνει ένα register file 32 καταχωρητών 32bit. Οι αριθμητικές και λογικές εντολές προσπελαύνουν μόνο καταχωρητές και όχι θέσεις μνήμης. Για να εκτελεστεί λειτουργία σε δεδομένα που βρίσκονται αποθηκευμένα στη μνήμη πρέπει αυτά να μεταφερθούν σε καταχωρητές. Αυτό γίνεται με τις εντολές μεταφοράς δεδομένων από/προς τη μνήμη. Τέλος, για τη μεταβίβαση της ροής υπάρχουν οι εντολές διακλάδωσης και κλήσης υπορουτινών.

3.2.1 Μορφή εντολών

Οι εντολές έχουν σταθερό μέγεθος (32bits). Η εντολή βρίσκεται σε «σχεδόν» αποκωδικοποιημένη μορφή, αφού είναι οργανωμένη σε διάφορα πεδία. Η ομαδοποίηση των πεδίων δεν είναι η ίδια για όλες τις εντολές, αλλά διακρίνονται 3 περιπτώσεις:

R type

op	rs	rt	rd	shamt	funct
6	5	5	5	5	6

Το format αυτό χρησιμοποιείται κυρίως από αριθμητικές & λογικές εντολές χωρίς άμεσο όρισμα. rd είναι ο καταχωρητής προορισμού, ενώ rs, rt οι καταχωρητές τελεστέοι (rd <- rs op rd). Το πεδίο shamt περιέχει ποσό ολίσθησης. Το πεδίο funct διακρίνει μεταξύ των λειτουργιών για το συγκεκριμένο opcode.

I type

op	rs	rt	addr or immed
6	5	5	16

Το format αυτό χρησιμοποιείται από αριθμητικές και λογικές εντολές με άμεσο όρισμα και σε εντολές μεταφοράς από/προς τη μνήμη (load/store). Στην πρώτη περίπτωση rt είναι ο καταχωρητής προορισμού, ενώ rs ο καταχωρητής τελεστής και immed ο τελεστής που βρίσκεται σε άμεση μορφή (rt<-rs+immed). Στην περίπτωση μεταφοράς από/προς τη μνήμη ο rs χρησιμοποιείται σαν καταχωρητής βάσης και το πεδίο immed σαν 16bit offset (έμμεση δεικτοδοτημένη διευθυνσιοδότηση). Ο rt χρησιμοποιείται σαν καταχωρητής προέλευσης ή προορισμού (mem[rs+immed]<-rt ή rt<-mem[rs+immed] αντίστοιχα).

Τέτοιο format χρησιμοποιείται και από τις εντολές διακλάδωσης υπό συνθήκη. Στην περίπτωση αυτή γίνεται σύγκριση των καταχωρητών rs, rt και σε περίπτωση που ικανοποιείται η συνθήκη γίνεται διακλάδωση στη διεύθυνση pc+addr (pc relative addressing).

J type

op	target address
6	26

Το format αυτό χρησιμοποιείται για εντολές διακλάδωσης χωρίς συνθήκη. Τα 4 MSB της 32bit δ/νσης στόχο διακλάδωσης είναι τα 4MSB του PC. Τα επόμενα 26bit εμπεριέχονται σαν άμεσο όρισμα στην εντολή. Τα 2 LSB είναι μηδενικά (δηλ. έχουμε δ/νση λέξης και όχι byte). Εδώ έχουμε ψευδό- άμεση διευθυνσιοδότηση.

Οι δυνατοί τρόποι διευθυνσιοδότησης που χρησιμοποιούνται από τις εντολές είναι οι εξής:

- Διευθυνσιοδότηση καταχωρητή. Το όρισμα είναι καταχωρητής.

- Έμμεση δεικτοδοτημένη διευθυνσιοδότηση . Γίνεται προσπέλαση στη μνήμη σε θέση που καθορίζεται από το περιεχόμενο καταχωρητή βάσης προσαυξημένο κατά άμεσο όρισμα (offset).
- Άμεση διευθυνσιοδότηση. Το όρισμα παρέχεται σε άμεση μορφή (εμπεριέχεται στην εντολή).
- Διευθυνσιοδότηση σε σχέση με τον καταχωρητή PC. Άμεσο όρισμα που εμπεριέχεται στην εντολή (με μορφή συμπληρώματος ως προς 2) προστίθεται στον καταχωρητή PC.
- Ψευδοάμεση διευθυνσιοδότηση. Επειδή δεν υπάρχει χώρος στην 32bit εντολή για 32bit διεύθυνση, τα 4 MSB λαμβάνονται από τον PC.

Οι εντολές, ανάλογα με τη λειτουργία τους , ομαδοποιούνται στις κατηγορίες:
1) Αριθμητικές & Λογικές εντολές,
2)Μετακίνησης δεδομένων,
3)Χειρισμού σταθερών,
4)Εντολές Άλματος,
5) Διακλάδωσης υπό συνθήκη.

Αναλυτικά έχουμε:

Αριθμητικές & Λογικές εντολές

add
opcode: 0x00 (funct=0x20)
Format: R
Λειτουργία: add rd, rs, rt (rs←rs+rt)
Σχόλια: -

sub
opcode: 0x00 (funct=0x22)
Format: R
Λειτουργία: sub rd, rs, rt (rs←rs-rt)
Σχόλια: -

subu
opcode: 0x00 (funct=0x23)
Format: R
Λειτουργία: subu rd, rs, rt (rs←rs-rt)
Σχόλια: χωρίς υπερχείλιση

addi
opcode: 0x08
Format: I
Λειτουργία: addi rt, rs,
sign_extend(immed)
(rt←rs+sign_extend(immed))
Σχόλια: -

addiu
opcode: 0x09
Format: I
Λειτουργία: addi rt, rs,
sign_extend(immed)
(rt←rs+sign_extend(immed))
Σχόλια: χωρίς υπερχείλιση

and
opcode: 0x00 (funct=0x24)
Format: I
Λειτουργία:
and rd, rs, rt (rd←rs&rt)
Σχόλια: -

andi
opcode: 0x0c
Format: I
Λειτουργία: andi rt, rs, zero_extend(immed)
(rt←rs&zero_extend(immed))
Σχόλια: -

div (δεν υλοποιήθηκε)

mult (δεν υλοποιήθηκε)

nor
opcode: 0x00 (funct=0x27)
Format: R
Λειτουργία: nor rd, rs, rt (rs←!(rs&rt))
Σχόλια: -

or
opcode: 0x00 (funct=0x24)
Format: R
Λειτουργία: or rd, rs, rt (rs←rs|rt)
Σχόλια: -

ori
opcode: 0x0d
Format: I
Λειτουργία: ori rt, rs,
zero_extend(immed)
(rt←rs|zero_extend(immed))
Σχόλια: -

sll rd, rt, shamt **(shift left logical)**
opcode: 0x00 (funct=0x00)
Format: R
Λειτουργία:
rd←rt<<shamt
Σχόλια: ο rs δεν χρησιμοποιείται

sllv rd, rt, shamt **(shift left logical variable)**
opcode: 0x00 (funct=0x04)
Format: R
Λειτουργία: rd←rt<<rs
Σχόλια: -

sra rd, rt, shamt **(shift right arithmetic)**
opcode: 0x00 (funct=0x03)
Format: R
Λειτουργία: rd←rt>>shamt
Σχόλια: ο rs δεν χρησιμοποιείται

srav rd, rt, rs **(shift right arithmetic variable)**
opcode: 0x00 (funct=0x07)
Format: R
Λειτουργία: rd←rt>>rs
Σχόλια: -

srl rd, rt, shamt **(shift left right)**
opcode: 0x00 (funct=0x02)
Format: R
Λειτουργία: rd←rt>>>shamt
Σχόλια: ο rs δεν χρησιμοποιείται

srlv rd, rt, rs **(shift right logical variable)**
opcode: 0x00 (funct=0x06)
Format: R
Λειτουργία: rd←rt>>>rs
Σχόλια: -

xor
opcode: 0x00 (funct=0x26)
Format: R
Λειτουργία: xor rd, rs, rt (rs←rs^rt)
Σχόλια: -

xori
opcode: 0x0e
Format: I
Λειτουργία: xori rt, rs, zero_extend(immed)
(rt←rs^zero_extend(immed))
Σχόλια: -

Μετακίνησης δεδομένων (load/store)

lw rt, rs, offset **(load word)**
opcode: 0x23
Format: I
Λειτουργία:
rt←word mem[rs+sign_extend(offset)]
Σχόλια: alignment

lb rt, rs, offset (**load byte**)
opcode: 0x20
Format: I
Λειτουργία:
rt←sign_extend(byte
mem[rs+sign_extend(offset)])
Σχόλια: alignment

lbu rt, rs, offset (**load unsigned byte**)
opcode: 0x24
Format: I
Λειτουργία:
rt←zero_extend(byte
mem[rs+sign_extend(offset)])
Σχόλια: alignment

lh rt, rs, offset (**load halfword**)
opcode: 0x21
Format: I
Λειτουργία:
rt←sign_extend(halfword
mem[rs+sign_extend(offset)])
Σχόλια: alignment

lhu rt, rs, offset (**load unsigned halfword**)
opcode: 0x25
Format: I
Λειτουργία:
rt←zero_extend(halfword mem[rs+sign_extend(offset)])
Σχόλια: alignment

[lwl I0x22 load word left
lwr I0x26 load word right]

sw rt, rs, offset **(store word)**
opcode: 0x2b
Format: I
Λειτουργία:
mem[rs+sign_extend(offset)] ←rt
Σχόλια: alignment

sb rt, rs, offset **(store byte)**
opcode: 0x28
Format: I
Λειτουργία:
mem[rs+sign_extend(offset)] ←rt[7:0]
Σχόλια:alignment

sh rt, rs, offset **(store halfword)**
opcode: 0x29
Format: I
Λειτουργία:
mem[rs+sign_extend(offset)] ←rt[15:0]
Σχόλια: alignment

[swl I0x2a store word left
swr I0x2e store word right]

Εντολές χειρισμού σταθερών

lui rt, imm
opcode: 0x0f
Format: I
Λειτουργία: rt[31:16] ←imm, rt[15:0] ←0
Σχόλια: -

Εντολές Άλματος

j target **(jump)**
opcode: 2
Format: J
Λειτουργία:
PC←PC[31:27]&target&00
Σχόλια: -

jr rs **(jump register)**
opcode: 0 (funct=0x8)
Format: R
Λειτουργία: PC←rs
Σχόλια: -

jal target **(jump and link)**
opcode: 3
Format: J
Λειτουργία: $ra←PC+4,
PC← PC[31:27]&target&00
Σχόλια: -

jalr rs, rd **(jump and link register)**
opcode: 0 (funct=9)
Format: R
Λειτουργία: rd←PC+4, PC← rs
Σχόλια: -

Εντολές Διακλάδωσης

beq rs, rt, offset **(branch on equal)**
opcode: 4
Format: I
Λειτουργία:
PC←(rs==rt)?PC+sign_extend(offset):
PC+4
Σχόλια: -

bne rs, rt, offset **(branch on not equal)**
opcode: 5
Format: I
Λειτουργία:
PC←(rs!=rt)?PC+sign_extend(offset):
PC+4
Σχόλια: -

bgez rs, 1, offset **(branch on greater than equal zero)**
opcode: 1 (rt=1)
Format: I
Λειτουργία:
PC←(rs>=0)?PC+sign_extend(offset):
PC+4
Σχόλια: -

bgezal rs, 0x11, offset **(branch on greater than equal zero and link)**
opcode: 1 (rt=0x11)
Format: I
Λειτουργία: r31←PC,
PC←(rs>=0)?PC+sign_extend(offset):
PC+4
Σχόλια: -

bgtz rs, 0, offset **(branch on greater than zero)**
opcode: 7 (rt=0)
Format: I
Λειτουργία:
PC←(rs>0)?PC+sign_extend(offset):
PC+4
Σχόλια: -

blez rs, 0, offset **(branch on less than equalzero)**
opcode: 6 (rt=0)
Format: I
Λειτουργία:
PC←(rs<=0)?PC+sign_extend(offset):
PC+4
Σχόλια: -

bltzal rs, 0x10, offset **(branch on less than zero and link)**
opcode: 1 (rt=0x10)
Format: I
Λειτουργία: r31←PC,
PC←(rs<0)?PC+sign_extend(offset):P
C+4
Σχόλια: -

bltz rs, 0, offset **(branch on less than zero)**
opcode: 1 (rt=0)
Format: I
Λειτουργία:
PC←(rs<0)?PC+sign_extend(offset):P
C+4
Σχόλια: -

Εντολές σύγκρισης

slt rd, rs, rt **(set less than)**
opcode: 0 (funct=0x2a)
Format: R
Λειτουργία: rd←(rs<rt)?1:0
Σχόλια: -

sltu rd, rs, rt **(set less than unsigned)**
opcode: 0 (funct=0x2b)
Format: R
Λειτουργία: rd←(rs<rt)?1:0
Σχόλια: unsigned σύγκριση

slti rt, rs, imm **(set less than immediate)**
opcode: 0xa
Format: I
Λειτουργία:
rt←(rs<sign_extended(imm)?1:0
Σχόλια: -

sltiu rt, rs, imm **(set less than immediate unsigned)**
opcode: 0xb
Format: I
Λειτουργία:
rt←(rs<sign_extended(imm)?1:0
Σχόλια: -

Συγκεντρωτικός πίνακας με την αντιστοίχηση των εντολών με opcodes.

Πίνακας 3.1 Αντιστοίχηση εντολών με opcodes.

Opcode instr[31:26]	εντολή	format	λειτουργία	κατηγορία
0x00	-	R	ανάλογα με το πεδίο fucnt (βλ. πίνακα 2)	
0x01	-	I	ανάλογα με το πεδίο rt (βλ. πίνακα 3)	
0x02	j	J	PC←PC[31:27]&target&00	Unconditional branch
0x03	jal	J	$ra←PC+4, PC← PC[31:27]&target&00	Unconditional branch + link
0x04	beq	I	PC←(rs==rt)? PC+sign_extend(offset):PC+4	Conditional branch
0x05	bne	I	PC←(rs!=rt)? PC+sign_extend(offset):PC+4	Conditional branch
0x06	blez	I	PC←(rs<=0)? PC+sign_extend(offset):PC+4	Conditional branch
0x07	bgtz	I	PC←(rs>0)? PC+sign_extend(offset):PC+4	Conditional branch
0x08	addi	I	addi rt, rs, sign_extend(immed) (rt←rs+sign_extend(immed))	integer/logical
0x09	addiu	I	addi rt, rs, sign_extend(immed) (rt←rs+sign_extend(immed))	integer/logical
0x0a	slti	I	rt←(rs<sign_extended(imm)?1:0	integer/logical
0x0b	sltiu	I	rt←(rs<sign_extended(imm)?1:0	integer/logical
0x0c	andi	I	andi rt, rs, zero_extend(immed) (rt←rs&zero_extend(immed))	integer/logical
0x0d	ori	I	ori rt, rs, zero_extend(immed) (rt←rs|zero_extend(immed))	integer/logical
0x0e	xori	I	xori rt, rs, zero_extend(immed) (rt←rs^zero_extend(immed))	integer/logical
0x0f	lui	I	rt[31:16]←imm, rt[15:0]←0	integer/logical
0x10-0x1f	-			
0x20	lb	I	rt←sign_extend(byte mem[rs+sign_extend(offset)])	load
0x21	lh	I	rt←sign_extend(halfword mem[rs+sign_extend(offset)])	load
0x22	lwl			
0x22	lw	I	rt←mem[rs+sign_extend(offset)]	load
0x23	lbu	I	rt←zero_extend(byte mem[rs+sign_extend(offset)])	load
0x24	lbu	I	rt←zero_extend(byte mem[rs+sign_extend(offset)])	load
0x25	lhu	I	rt←zero_extend(halfword mem[rs+sign_extend(offset)])	load
0x26	lwr			
0x27	-			
0x28	sb	I	mem[rs+sign_extend(offset)]←rt[7:0]	store
0x29	sh	I	mem[rs+sign_extend(offset)]←rt[15:0]	store
0x2a	swl			
0x2b	sw	I	mem[rs+sign_extend(offset)]←rt	store
0x2c-0x2d	-			
0x2e	swr			
0x2f-0x3f	-			

Πίνακας 3.2 Εντολές με opcode 0x00. Διαφοροποίηση με βάση το πεδίο funct.

Opcode instr[31:26]	funct instr[5:0]	εντολη	format	Λειτουργία	κατηγορία
0x00	0x00	sll	R	rd←rt<<shamt	integer/logical
0x00	0x01	-			
0x00	0x02	srl	R	rd←rt>>>shamt	integer/logical
0x00	0x03	sra	R	rd←rt>>shamt	integer/logical
0x00	0x04	sllv	R	rd←rt<<rs	integer/logical
0x00	0x05	-			
0x00	0x06	srlv	R	rd←rt>>>rs	integer/logical
0x00	0x07	srav	R	rd←rt>>rs	integer/logical
0x00	0x08	jr	R	PC←rs	unconditional branch (register indirect)
0x00	0x09	jalr	R	rd←PC+4, PC← rs	unconditional branch (register indirect) +link
0x00	0x0a- 0x0b	-			
0x00	0x0c	syscall			
0x00	0x0d	break			
0x00	0x0e- 0x1f	-			
0x00	0x20	add	R	add rd, rs, rt (rs←rs+rt)	integer/logical
0x00	0x21				
0x00	0x22	sub	R	sub rd, rs, rt (rs←rs-rt)	integer/logical
0x00	0x23	subu	R	subu rd, rs, rt (rs←rs-rt)	integer/logical
0x00	0x24	and	R	and rd, rs, rt (rd←rs&rt)	integer/logical
0x00	0x25	or	R	or rd, rs, rt (rs←rs|rt)	integer/logical
0x00	0x26	xor	R	xor rd, rs, rt (rs←rs^rt)	integer/logical
0x00	0x27	nor	R	nor rd, rs, rt (rs←!(rs&rt))	integer/logical
0x00	0x28- 0x29	-			
0x00	0x2a	slt	R	rd←(rs<rt)?1:0	integer/logical
0x00	0x2b	sltu	R	rd←(rs<rt)?1:0	integer/logical
0x00	0x2c-0x3f	-			

Πίνακας 3.3 Εντολές με opcode 0x01. Διαφοροποίηση με βάση το πεδίο rt.

Opcode instr[31:26]	rt instr[20:16]	εντολη	format	Λειτουργία	κατηγορία
0x01	0x00	bltz	I	PC←(rs<0)? PC+sign_extend(offset): PC+4	conditional branch
0x01	0x01	bgez	I	PC←(rs>=0)? PC+sign_extend(offset): PC+4	conditional branch
0x01	0x02-0x0f	-			
0x01	0x10	bltzal	I	r31←PC, PC←(rs<0)? PC+sign_extend(offset): PC+4	conditional branch + link
0x01	0x11	bgezal	I	r31←PC, PC←(rs>=0)? PC+sign_extend(offset): PC+4	conditional branch + link
0x01	0x12-0x1f	-			

4 ΜΙΚΡΟΑΡΧΙΤΕΚΤΟΝΙΚΗ MICROPROC

Σε αυτό το κεφάλαιο της διπλωματικής θα γίνει μία εκτενέστερη περιγραφή της μικροαρχιτεκτονικής του επεξεργαστή MICROPROC. Αυτή θα περιλαμβάνει την περιγραφή σε επίπεδο μικροαρχιτεκτονικής κάθε σταδίου-δομικής μονάδας του επεξεργαστή ξεχωριστά και τις αντίστοιχες αναλύσεις των χρονισμών των σημάτων κάθε σταδίου. Το κεφάλαιο αυτό θα είναι και το μεγαλύτερο και σημαντικότερο κεφάλαιο της εργασίας και σε αυτό θα γίνει προσπάθεια εξοικείωσης με τις τεχνικές σχεδίασης ενός superscalar επεξεργαστή σε επίπεδο μικροαρχιτεκτονικής.

Όπως έχουμε ήδη αναφέρει πρωτύτερα, ένα superscalar pipeline αποτελείται συνήθως από τα στάδια fetch, decode, dispatch, execute, complete και retire. Το superscalar pipeline του επεξεργαστή MICROPROC αποτελείται από τα στάδια fetch, decode, dispatch, execute και complete (το στάδιο retire υλοποιείται ουσιαστικά μαζί με το complete), τα οποία επικοινωνούν μεταξύ τους με τη χρήση κατάλληλων buffers που κρατούν τα απαραίτητα δεδομένα. Στη συνέχεια ακολουθεί η περιγραφή των 5 σταδίων του επεξεργαστή MICROPROC.

4.1 Fetch στάδιο

Στο στάδιο αυτό ανανεώνεται η τιμή του PC και γίνεται ανάγνωση από την instruction cache. Αρχικά δεν θα γίνει χρήση TLB. Η instruction cache είναι 2-way set associative με 16bytes μέγεθος γραμμής (16bytes line size) (δηλ. 4 εντολές).

Το στάδιο αυτό σπάει σε πολλούς κύκλους. Στον πρώτο κύκλο γίνεται ο υπολογισμός του PC. Στην αρνητική ακμή εγγράφεται. Από εκεί και πέρα λαμβάνει χώρα ανάγνωση από την cache, η οποία βγάζει έξοδο μετά από λίγο λιγότερο από ένα κύκλο. Η γραμμή εξόδου της cache μαζί με τα tags λατσάρονται με αρνητικά ακμοπυροδότητους καταχωρητές. Στο δεύτερο μισό του τρίτου κύκλου γίνεται ο έλεγχος του tag, αν υπάρχει hit. Αν η σύγκριση είναι επιτυχής, οι εντολές που ανακτήθηκαν προωθούνται στο στάδιο αποκωδικοποιήσης. Αν δεν είναι επιτυχής θα πρέπει να stallάρει το pipeline και να γεμίσει η I-cache από την level2 cache.

Κάθε γραμμή της cache περιέχει 4 εντολές. Αν η τιμή του PC δεν είναι πολλαπλάσιο του 16, τότε ο PC δεν δείχνει στην αρχή της γραμμής και επομένως δε μπορούν να χρησιμοποιηθούν όλες οι εντολές της γραμμής. Στην χειρότερη περίπτωση, αν ο PC δείχνει στη τελευταία εντολής της γραμμής, μόνο μία εντολή θα είναι χρήσιμη και θα προωθηθεί στο στάδιο decode. Τα bits [3:2] του PC καθορίζουν το πόσες εντολές θα είναι χρήσιμες και προωθείται κι αυτό στο στάδιο αποκωδικοποιήσης.

4.1.1 Υλοποίηση

Τώρα θα γίνει η περιγραφή των αρχιτεκτονικών μονάδων του σταδίου fetch καθώς επίσης και των διασυνδέσεων των σημάτων του. Στο στάδιο αυτό υπάρχει ένας πολυπλέκτης (pc_mux) ο οποίος επιλέγει την επόμενη τιμή του καταχωρητή pc ο οποίος είναι αρνητικά ακμοπυροδότητος. Οι είσοδοι του πολυπλέκτη είναι η παλιά τιμή του pc (η τρέχουσα δηλαδή τιμή του καταχωρητή pc), το σήμα pcfromdec που αποτελεί την υπολογισμένη τιμή του pc από το στάδιο decode με βάση την εντολή

διακλάδωσης και τα σήματα branch και stall και branch_incorrect. Ο πολυπλέκτης δίνει ως έξοδο του την τρέχουσα τιμή του καταχωρητή pc όταν το σήμα stall ισούται με τη μονάδα, κάτι που σημαίνει ότι το στάδιο fetch πρέπει να καθυστερήσει και να μην φορτώσει καινούριες εντολές. Όταν το σήμα branch ισούται με μονάδα τότε η έξοδος του πολυπλέκτη γίνεται ίση με το σήμα pcfromdec δεδομένου ότι ακολουθείται πολιτική branch taken στην πρόβλεψη διακλάδωσης. Αυτό σημαίνει ότι όταν έχουμε μία διακλάδωση, είτε υπό συνθήκη είτε χωρίς, ο καταχωρητής pc γίνεται ίσος με τη διεύθυνση στόχο της διακλάδωσης. Στην περίπτωση που η πρόβλεψη της διακλάδωσης αποδειχθεί στο στάδιο execute λανθασμένη και το branch_icorrect ισούται με τη μονάδα ο πολυπλέκτης θα δώσει ως έξοδό του το παλιό pc αυξημένο κατά τέσσερα, δηλαδή την διεύθυνση της επόμενης εντολής από την διακλάδωση που αποδείχθηκε λανθασμένη. Τέλος αν δεν ισχύει τίποτα από αυτά, η έξοδος του πολυπλέκτη θα πάρει την τρέχουσα τιμή του pc αυξημένη κατά δεκαέξι (δηλαδή κατά 4 εντολές) (λύνοντας τα προβλήματα του misalignment, δηλαδή έρχεται η πρώτη εντολής από την επόμενη τετράδα εντολών).

Αυτή η έξοδος του πολυπλέκτη αποτελεί όπως είπαμε είσοδος του αρνητικά ακμοπυροδότητου καταχωρητή PC ο οποίος λατσάρεται στην κατάλληλη τιμή στην αρνητική ακμή του κύκλου ρολογιού, στην αρχή του οποίου έγινε ο υπολογισμός της τιμής εισόδου του. Σε αυτόν τον καταχωρητή όμως εκτός από την είσοδο υπάρχει και ένα σήμα επίτρεψης. Το σήμα αυτό είναι το wr_en, το οποίο αποτελεί έξοδο ενός αρνητικά ακμοπυροδότητου D flip-flop, που δέχεται ως είσοδο το σήμα nfhold, που αποτελεί αντιστροφή του σήματος fhold (fetch hold). Έτσι όσο το σήμα fhold ισούται με το μηδέν, το σήμα nfhold ισούται με τη μονάδα και ως εκ τούτου το σήμα wr_en ισούται επίσης με τη μονάδα. Αυτό έχει ως αποτέλεσμα να μπορεί ο καταχωρητής PC να λατσάρει στην τιμή της εισόδου τους στην αρνητική ακμή του κύκλου ρολογιού. Στην περίπτωση που το fhold γίνει μονάδα (κάτι που σημαίνει ότι το στάδιο fetch θα πρέπει να stallάρει τη λειτουργία του) ο καταχωρητής pc δεν θα αλλάξει τιμή στην αρνητική ακμή του ρολογιού αφού το σήμα επίτρεψής του θα είναι ίσο με μηδέν.

Η έξοδος του καταχωρητή PC (η τρέχουσα τιμή του PC) χρησιμοποιείται για την διευθυνσιοδότηση μέσα στην instruction cache μνήμη (I-cache) από όπου διαβάζεται κάθε φορά μία τετράδα εντολών (μία γραμμή δηλαδή της μνήμης, cache line), ή λιγότερες (όταν το pc δεν δείχνει στην πρώτη εντολή της γραμμής). Η δομή της instruction cache έχει ήδη αναφερθεί και εδώ το μόνο που θα πρέπει να προστεθεί είναι ότι η μνήμη αυτή αποκρίνεται σε πολύ μικρό χρόνο (λιγότερο του ενός κύκλου ρολογιού). Το σήμα εξόδου της I-cache είναι το data και η τιμή του λατσάρεται με αρνητικά ακμοπυροδότητο καταχωρητή, τον INSTRd. Αυτό σημαίνει ότι αν η τιμή του pc είναι σταθερή λίγο μετά την αρνητική ακμή του κύκλου ρολογιού, η μνήμη αποκρίνεται λίγο μετά δίνοντας σταθερή τιμή στο σήμα data, το οποίο λατσάρεται μέσω του καταχωρητή instrd (δίνοντας τιμή στο σήμα instrd που αποτελεί σήμα εξόδου του καταχωρητή) στην αρνητική ακμή του αμέσως επόμενου κύκλου ρολογιού. Παράλληλα όμως η έξοδος του καταχωρητή PC (δηλαδή το σήμα pc) αποτελεί το σήμα εισόδου του καταχωρητή PCd (pc delayed), οποίος λατσάρει την τιμή του pc στην επόμενη αρνητική ακμή ρολογιού αφού είναι και αυτός αρνητικά ακμοπυροδότητος. Το σήμα εξόδου του PCd ονομάζεται pcd. Εδώ θα πρέπει να σημειωθεί ότι οι καταχωρητές INSTRd και PCd έχουν επίσης ως σήμα επίτρεψης το wr_en.

Όπως γίνεται εύκολα αντιληπτό, τα σήματα pcd και instrd παίρνουν τιμή στην ίδια αρνητική ακμή ρολογιού και έτσι χρησιμοποιούνται ως σήματα ελέγχου για τις μονάδες ελέγχου ετικέτας (tag_check) και εξαγωγής εντολών (instruction extractor).

Αυτός είναι ούτως ή άλλως ο λόγος ύπαρξης των σημάτων αυτών, δηλαδή χρησιμοποιούνται για κατάλληλο συγχρονισμό και έλεγχο. Κάτι τέτοιο δεν θα μπορούσε να επιτευχθεί με τις τιμές των σημάτων pc και data αφού το data δεν είναι εντελώς συγχρονισμένο με το pc (το pc αποτελεί είσοδο της μονάδας που δίνει ως έξοδο το data). Αντίθετα τα σήματα pcd και instrd είναι όπως είπαμε συγχρονισμένα στην ίδια ακμή και μπορούν να χρησιμοποιηθούν μαζί.

Η μονάδα ελέγχου ετικέτας (tag_check) είναι αυτή που ελέγχει αν οι εντολές που φορτώθηκαν από την instruction cache έχουν τη σωστή ετικέτα. Αν την έχουν τότε η τιμή του σήματος hit, που αποτελεί έξοδο της μονάδας αυτής, τίθεται στη μονάδα και όλα μπορούν να προχωρήσουν κανονικά. Σε αντίθετη περίπτωση πρέπει να καθυστερηθεί το pipeline και να γεμίσει η instruction cache από την level2 cache.

Η μονάδα εξαγωγής των εντολών (instruction extractor) είναι ουσιαστικά αυτή που δίνει τα σήματα εξόδου του σταδίου fetch τα οποία προωθούνται στο στάδιο decode. Αυτά τα σήματα είναι οι τέσσερις 32bit εντολές ni0, ni1, ni2, ni3 και το σήμα num_of_ni (#ni), δηλαδή το πλήθος των καινούριων εντολών που φορτώθηκαν. Το σήμα pcd χρησιμοποιείται για τον υπολογισμό του num_of_ni, δηλαδή του πλήθους των εντολών που προωθούνται στο επόμενο στάδιο. Αυτό γίνεται με τη χρήση των bits [3:2] του pcd τα οποία καθορίζουν το πόσες εντολές είναι χρήσιμες. Όταν τα bits αυτά είναι ίσα με “00” έχουμε 4 καινούριες εντολές, όταν είναι ίσα με “01” έχουμε 3, όταν είναι ίσα με “10” έχουμε 2, και όταν είναι ίσα με “11” έχουμε 1 καινούρια εντολή. Στην περίπτωση που όλες οι εντολές έχουν όλα τα bits ίσα με μηδέν το πλήθος των εντολών τίθεται ίσο με μηδέν. Τέλος τα σήματα ni0, ni1, ni2, ni3 παίρνουν τις τιμές τους από το σήμα instrd που περιέχει τις εντολές με τη σειρά που αυτές φορτώθηκαν από την instruction cache. Για την ακρίβεια τα bits [31:0] αντιστοιχούν στην $1^{η}$ εντολή (ni0), τα bits [63:32] αντιστοιχούν στην $2^{η}$ εντολή (ni1), τα bits [95:64] αντιστοιχούν στην $1^{η}$ εντολή (ni2) και τα bits [127:96] αντιστοιχούν στην $4^{η}$ εντολή (ni3).

Τώρα ακολουθεί ένα κεφάλαιο που αφορά τον χρονισμό των σημάτων του σταδίου fetch.

4.1.2 Χρονισμός σημάτων σταδίου fetch

Το στάδιο fetch ως γνωστόν είναι υπεύθυνο για τη φόρτωση των εντολών από την instruction cache (I-cache). Ο μέγιστος αριθμός εντολών που μπορούν να φορτωθούν σε κάθε κύκλο είναι τέσσερις, δηλαδή μία ολόκληρη γραμμή μνήμης της I-cache. Το στάδιο αυτό έχει ως εισόδους τα σήματα pcfromdec, stall, branch και fhold. Τα σήματα pcfromdec, branch και fhold αποτελούν εξόδους του σταδίου decode και λαμβάνουν σταθερή τιμή αμέσως μετά από θετική ακμή κύκλου ρολογιού (δηλαδή αλλάζουν τιμές σε θετικές ακμές κύκλων ρολογιού). Το σήμα stall αποτελεί έξοδο του σταδίου dispatch και λαμβάνει επίσης τιμή αμέσως μετά από θετική ακμή κύκλου ρολογιού (δηλαδή αλλάζει τιμές σε θετικές ακμές κύκλων ρολογιού). Σε αυτό το σημείο θα εξετάσουμε το χρονισμό των σημάτων του σταδίου fetch.

Αρχικά θα δούμε το χρονισμό των σημάτων pc, new_pc και pcd που αποτελούν ουσιαστικά τον PC (program counter), την καινούρια τιμή του PC (new PC) και την καθυστερημένη κατά ένα κύκλο τιμή του PC (delayed PC) αντίστοιχα. Ουσιαστικά ο PC λατσάρεται στην καινούρια τιμή του PC με την χρήση αρνητικά ακμοπυροδότητου καταχωρητή. Ο delayed PC λατσάρεται στην τιμή του PC με αρνητικά ακμοπυροδότητο καταχωρητή και επομένως αποτελεί αντίγραφο του PC καθυστερημένο κατά ένα κύκλο ρολογιού.

Για να αντιληφθούμε καλύτερα τον χρονισμό των σημάτων του σταδίου πρέπει να ορίσουμε τον κύκλο ρολογιού στου οποίου την αρνητική ακμή λαμβάνει σταθερή τιμή το σήμα new_pc ως τον προηγούμενο του πρώτου κύκλο ρολογιού του σταδίου fetch. Η τιμή του σήματος pc σε αυτήν την περίπτωση σταθεροποιείται αμέσως μετά την αρνητική ακμή του επόμενου κύκλου ρολογιού, δηλαδή στην αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου. Το σήμα pcd λαμβάνει τέλος σταθερή τιμή (αυτήν του σήματος pc) στην αρνητική ακμή του επόμενου κύκλου ρολογιού, δηλαδή στην αρνητική ακμή του δεύτερου κύκλου ρολογιού του σταδίου.

Εν συνεχεία θα εξετάσουμε τον χρονισμό των σημάτων data και instrd. Το σήμα data λαμβάνει σταθερή τιμή στην ίδια αρνητική ακμή όπου σταθεροποιείται η τιμή του σήματος pc αφού ουσιαστικά αποτελεί τα δεδομένα εξόδου της I-cache με διευθυνσιοδότηση μέσω του καταχωρητή PC. Επομένως το σήμα data λατσάρεται σε σταθερή τιμή λίγο μετά την αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου fetch. Το σήμα instrd παίρνει την τιμή του σήματος data μετά από έναν ακριβώς κύκλο ρολογιού αφού λατσάρεται με αρνητικά ακμοπυροδότητο καταχωρητή ο οποίος έχει ως είσοδο το σήμα data. Άρα η τιμή του instrd σταθεροποιείται λίγο μετά την αρνητική ακμή του δεύτερου κύκλου ρολογιού του σταδίου.

Όσων αφορά τα σήματα εξόδου του σταδίου fetch, τα οποία είναι τα pc0, num_of_ni, ni0, ni1, ni2 και ni3, τα πράγματα είναι αρκετά απλά όσων αφορά το χρονισμό τους. Το σήμα pc0 είναι ακριβώς το ίδιο σήμα με το σήμα pc και επομένως παίρνει σταθερή τιμή αμέσως μετά την αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου. Ακόμα τα σήματα ni0, ni1, ni2 και ni3, που αποτελούν τις καινούριες εντολές που προωθούνται από το στάδιο fetch στο στάδιο decode, ουσιαστικά συνθέτουν όλα μαζί το σήμα instrd. Αυτό σημαίνει ότι η τιμή των σημάτων αυτών σταθεροποιείται λίγο μετά την αρνητική ακμή του δεύτερου κύκλου ρολογιού του σταδίου. Το σήμα num_of_ni που περιέχει το πλήθος των καινούριων εντολών που προωθούνται στο επόμενο στάδιο λαμβάνει επίσης σταθερή τιμή την ίδια χρονική στιγμή με τα σήματα των καινούριων εντολών, δηλαδή λίγο μετά την αρνητική ακμή του δεύτερου κύκλου ρολογιού του σταδίου fetch, ο οποίος είναι ουσιαστικά ο τελευταίος κύκλος ρολογιού του σταδίου fetch (αυτή η αρίθμηση για τους κύκλους ρολογιού του σταδίου θα ακολουθηθεί και στο κεφάλαιο που αναφέρεται στον χρονισμό του σταδίου decode).

Τέλος θα δούμε με ποιον τρόπο επηρεάζεται ο χρονισμός των σημάτων του σταδίου fetch λόγω των αλλαγών των τιμών των σημάτων εισόδου του. Αρχικά θα ελέγξουμε τι γίνεται όταν αλλάζει τιμή το σήμα branch. Σε αυτήν την περίπτωση αλλάζει τιμή την ίδια χρονική στιγμή το σήμα new_pc. Επειδή το σήμα branch όμως όπως προείπαμε λαμβάνει σταθερή τιμή αμέσως μετά από θετική ακμή κύκλου ρολογιού συμπεραίνουμε ότι και το σήμα new_pc λαμβάνει την (αντίστοιχη στην αλλαγή του σήματος branch, δηλαδή την τιμή του pcfromdec αν το branch έγινε ίσο με μονάδα στην θετική ακμή ή την παλιά τιμή του PC συν 4 (εννοείται συν 4 εντολές) αν το branch έγινε ίσο με μηδέν) σταθερή τιμή του λίγο μετά την ίδια θετική ακμή ρολογιού. Ακριβώς αντίστοιχα είναι τα πράγματα με το σήμα stall. Και σε αυτήν την περίπτωση το σήμα new_pc λαμβάνει την σταθερή τιμή του (αντίστοιχη της αλλαγής του σήματος stall, δηλαδή την τιμή του παλιού PC αν το stall έγινε ίσο με μονάδα στην θετική ακμή ή την παλιά τιμή του PC συν 4 (εννοείται συν 4 εντολές) αν το stall έγινε ίσο με μηδέν). Τέλος, όταν αλλάζει τιμή το σήμα fhold το σήμα wr_en αλλάζει τιμή μετά από έναν κύκλο, αφού το wr_en λατσάρεται από θετικά ακμοπυροδότητο καταχωρητή με είσοδο το σήμα not_fhold (δηλαδή ουσιαστικά το σήμα fhold).

4.2 Decode στάδιο

Στο στάδιο της αποκωδικοποίησης ανακτούμε την πληροφορία που υπάρχει στις εντολές (πληροφορίες σχετικές με τον τύπο του branch σε περίπτωση branch, καταχωρητές που διαβάζονται, καταχωρητές που εγγράφονται, μονάδα εκτέλεσης που θα χρησιμοποιηθεί, format εντολής). Έχουμε 4 πανομοιότυπες μονάδες που εξάγουν την πληροφορία αυτή για καθεμία από τις μέχρι 4 εντολές που ανακτώνται από τον instruction buffer.

Στη συνέχεια ελέγχουμε εξαρτήσεις RAW καταχωρητών μέσα στην 4άδα εντολών. Από το στάδιο αυτό προκύπτει ο μέγιστος αριθμός εντολών που μπορούν να οδηγηθούν στον dispatch buffer. Σε περίπτωση που υπάρχει branch (conditional ή unconditional) δεν προωθείται εντολή μετά απ' αυτό. Επεξεργαζόμαστε μόνο 1 branch ανά κύκλο και αγνοούμε τις εντολές μετά από αυτό. Έτσι ο αριθμός των εντολών που προωθούνται περιορίζεται.

Σε περίπτωση που γίνει speculation σε ένα conditional branch φορτώνεται ένας καταχωρητής με το id του υποθετικού μονοπατιού και παράλληλα αυξάνεται ένας μετρητής, που αν είναι μη μηδενικός σημαίνει ότι υπάρχει υποθετική εκτέλεση. Το id του τελευταίου υποθετικού μονοπατιού τοποθετείται σε κάθε εντολή που αποκωδικοποιείται. Έτσι μπορούμε αργότερα να ανακαλέσουμε τις εντολές ή να τις επιβεβαιώσουμε.

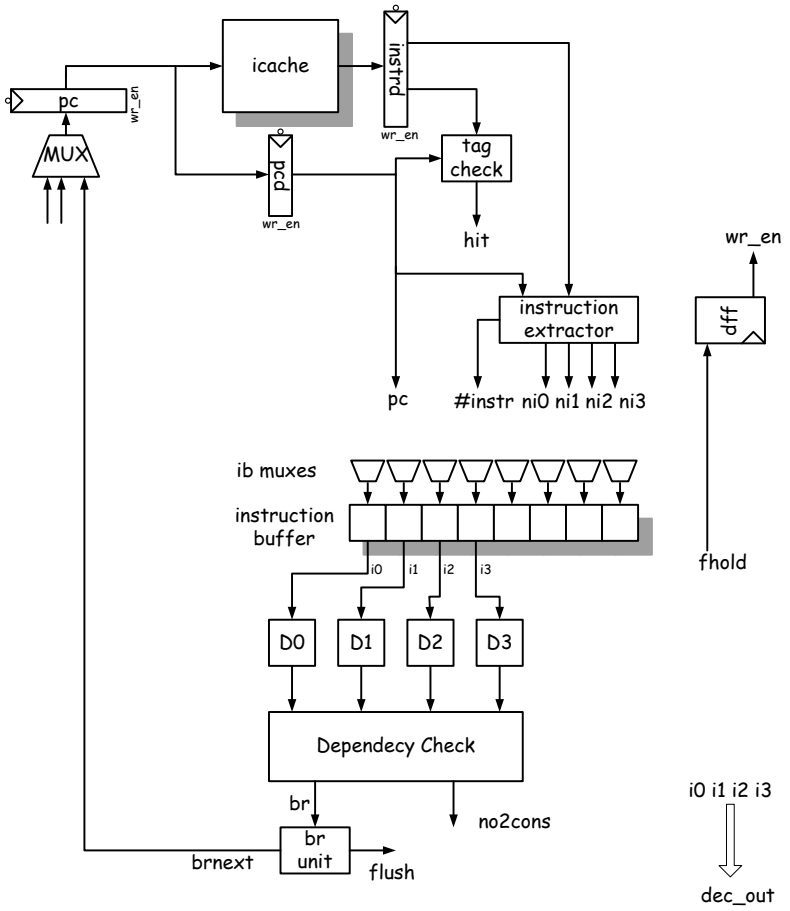
Στην αρχική έκδοση θα έχουμε πολιτική not taken στα conditional branches. Στη συνέχεια θα χρησιμοποιήσουμε branch history. Για τα μη conditional branches δεν θα χρησιμοποιηθούν BTAC αρχικά. Στα pc relative unconditional branches η διεύθυνση ανάκλησης εντολών είναι έτοιμη στο στάδιο decode, οπότε γίνεται flush ο instruction buffer και χάνεται ένας κύκλος. Στα register indirect branches unconditional branches η διεύθυνση παράγεται στο στάδιο dispatch (αφού πρέπει πρώτα να διαβαστεί ο καταχωρητής). Εκεί χάνονται δύο κύκλοι.

Το στάδιο αυτό παίρνει μέχρι 4 εντολές από τον instruction buffer και αφού αποκωδικοποιήσει μέχρι 4 εντολές τις τοποθετεί στον dispatch buffer.

4.2.1 Υλοποίηση

Η βασική λειτουργία του αποκωδικοποιητή έχει ως εξής: Διατηρούμε έναν instruction buffer που χωράει έως και 8 εντολές. Αυτές έρχονται από το στάδιο fetch. Η επικοινωνία του σταδίου fetch και decode θα εξηγηθεί αναλυτικά πιο κάτω.

Ο instruction buffer μπορεί κάθε φορά να έχει από μηδέν έως και 8 εντολές. Ο καταχωρητής-δείκτης length δείχνει στην τελευταία εντολή στον instruction buffer.



Σχήμα 4.1 Στάδια fetch και decode και η διασύνδεσή τους.

Εντολές καταναλώνονται από τη θέση 0 έως και τη θέση 3, προκαλώντας αντίστοιχη μετατόπιση (n2cons) στον instruction buffer. Ο αριθμός των καταναλισκόμενων εντολών n2cons προκύπτει από τη μονάδα εντοπισμού εξαρτήσεων (dependency check unit), όπως θα περιγραφεί πιο κάτω. Εντολές από το στάδιο fetch εισέρχονται στις θέσεις length-no2cons+1 έως και length-no2cons+4, με το συνολικό πλήθος να μην ξεπερνάει τις 8. Οι εντολές που εισέρχονται από το στάδιο fetch μπορεί να μην είναι 4, λόγω προβλημάτων alignment (βλ πιο πάνω & πιο κάτω). Το πλήθος καθορίζεται από τα bits [3:2] του PC, με βάση τα οποία επιλέγονται και οι χρήσιμες εντολές (από τη μονάδα instruction extractor) όπως ήδη αναλύσαμε στο κεφάλαιο όπου περιγράφηκε το στάδιο fetch.

Σε κάθε κύκλο ρολογιού τα σήματα επιλογής των 8 θέσεων του instruction buffer παράγονται από μία μονάδα επιλογής εντολής (sel unit), η οποία με βάση τον αριθμό των καταναλισκόμενων εντολών (n2cons), τον καταχωρητή-δείκτη length και τον αριθμό των καινούριων εντολών που προωθήθηκαν από το στάδιο fetch (#ni) βγάζει ως έξοδο τα σήματα επιλογής sel0, sel1,…, sel7 (για την επιλογή της τιμής των θέσεων 0, 1,…, 7 του instruction buffer). Η τιμή κάθε μίας από τις 8 θέσεις του instruction buffer μπορεί να αποκτήσει μία από τις ακόλουθες τιμές:

- την τιμή της πρώτης καινούριας εντολής που προωθήθηκε από το προηγούμενο στάδιο,
- την τιμή της δεύτερης καινούριας εντολής που προωθήθηκε από το προηγούμενο στάδιο,
- την τιμή της τρίτης καινούριας εντολής που προωθήθηκε από το προηγούμενο στάδιο,

- την τιμή της τέταρτης καινούριας εντολής που προωθήθηκε από το προηγούμενο στάδιο,
- την τιμή που είχε στον προηγούμενο κύκλο η θέση αυτή,
- την τιμή που είχε στον προηγούμενο κύκλο η επόμενη θέση,
- την τιμή που είχε στον προηγούμενο κύκλο η μεθεπόμενη θέση,
- την τιμή που είχε στον προηγούμενο κύκλο η τρίτη επόμενη θέση,
- τιμή που είχε στον προηγούμενο κύκλο η τέταρτη επόμενη θέση.

Αυτές οι τιμές αποτελούν την είσοδο των 8 πολυπλεκτών που με βάση την τιμή του σήματος seli (το σήμα αυτό είναι το συνολικό σήμα επιλογής και αποτελείται από τα 8 σήματα sel0, sel1,…, sel7) επιλέγουν την νέα τιμή των 8 θέσεων του instruction buffer.

Οι εντολές που εξάγονται από τον instruction buffer αποκωδικοποιούνται η κάθε μια ξεχωριστά από μία μονάδα αποκωδικοποιήσης εντολής D. Η μονάδα D (βλ. σχήμα) βγάζει ως έξοδο τους καταχωρητές προέλευσης και προορισμού της εντολής, πληροφορία αν είναι branch κα, με βάση φυσικά το πεδίο opcode της 32bit εντολής. Οι έξοδοι αυτοί χρησιμοποιούνται από τη μονάδα εντοπισμού εξαρτήσεων, η οποία ελέγχει αν υπάρχουν RAW εξαρτήσεις. Η μονάδα αυτή αποκρίνεται με τον αριθμό των εντολών που μπορούν να προωθηθούν στο επόμενο στάδιο (n2cons). Η ομάδα εντολών που προωθείται είναι η υποακολουθία που ξεκινάει από την παλιότερη εντολή του instruction buffer (θέση 0) και όσο το δυνατό περισσότερες μη εξαρτώμενες (με RAW) εντολές. Επίσης λόγω του περιορισμού, αν υπάρχει branch, αυτό να είναι η τελευταία εντολή στην ομάδα, έχουμε επιπλέον περιορισμό στον αριθμό των προωθούμενων εντολών.

Συνοπτικά η μονάδα αυτή βγάζει τον αριθμό των εντολών που προωθούνται και κατά συνέπεια αφαιρούνται από τον instruction buffer, και μία σημαία που δείχνει αν υπάρχει branch μέσα στην υποακολουθία χωρίς RAW.

Επίσης η μονάδα αυτή βγάζει ως έξοδο την σημαία cond_br που υποδεικνύει διακλάδωση υπό συνθήκη και η οποία αποτελεί είσοδο μαζί με τις σημαίες br_correct και br_incorrect (αυτές προέρχονται από το στάδιο execute) της μηχανής καταστάσεων του σταδίου decode (dec state machine). Η μονάδα αυτή έχει τρεις καταστάσεις, τις NO_BRANCHES, στην οποία βρίσκεται η μηχανή όταν δεν υπάρχει κανένα branch speculated, ONE_BRANCH_SPEC, στην οποία βρίσκεται η μηχανή όταν υπάρχει ένα branch speculated) και TWO_BRANCHES_SPEC, στην οποία βρίσκεται η μηχανή όταν υπάρχουν δύο branches speculated. Σαν έξοδο η μηχανή καταστάσεων βγάζει την σημαία two_nested_br που υποδηλώνει 2 φωλιασμένα speculated branches και έτσι αποτρέπεται speculation άλλου branch (υπάρχει πολιτική speculation μέχρι 2 φωλιασμένων διακλαδώσεων υπό συνθήκη). Επίσης βγαίνει ως έξοδος και το speculation id (parent_id και kid_id), δηλαδή το id του υποθετικού μονοπατιού, που προσκολλάται στις εντολές που προωθούνται στο επόμενο στάδιο. Για την ακρίβεια, αφού το conditional branch εκ των πραγμάτων αποτελεί την τελευταία εντολή από μία ομάδα καταναλισκόμενων εντολών, το speculation id παίρνει τιμή μετά την προώθηση της ομάδας αυτής εντολών και προσκολλάται στις επόμενες εντολές (δηλαδή στις εντολές που προωθούνται στον επόμενο κύκλο ρολογιού) που ουσιαστικά αποτελούν εντολές που ανήκουν σε speculation path.

Τέλος υπάρχει και η μονάδα διακλάδωσης του decode σταδίου (br_unit) η οποία με βάση το n2cons, τη σημαία branch, τις σημαίες j0, j1, j2 και j3 (που αποτελούν εξόδους της μονάδας εντοπισμού εξαρτήσεων και υποδηλώνουν ποια εντολή από αυτές που προωθούνται είναι εντολή διακλάδωσης, αν υπάρχει φυσικά) και τα πεδία της εντολής διακλάδωσης που απαιτούνται για τον υπολογισμό της

διεύθυνσης στόχου της διακλάδωσης, δίνει ως έξοδο τη διεύθυνση pcfromdec, που όπως ήδη έχουμε περιγράψει χρησιμοποιείται στο στάδιο fetch για τον υπολογισμό του PC.

Στη συνέχεια θα δούμε την διαλειτουργικότητα μεταξύ των μονάδων fetch και decode και θα μελετήσουμε τον χρονισμό των σημάτων και την εφικτότητα του μηχανισμού επικοινωνίας από απόψη κρίσιμου μονοπατιού.

4.2.2 Διασύνδεση με το στάδιο fetch και χρονισμός των σημάτων

Το στάδιο fetch, όπως σημειώθηκε και παραπάνω, είναι χωρισμένο σε 2 κύκλους. Στον πρώτο κύκλο, στη δεύτερη φάση (δηλ. στην αρνητική ακμή) ο καταχωρητής PC παίρνει την τιμή που υπολογίσθηκε στην πρώτη φάση (ή προήλθε από άλλο στάδιο στη φάση αυτή). Η τιμή του PC αποτελεί τη δ/νση της instruction cache, η οποία είναι ασύγχρονη SRAM. Θεωρούμε ότι αποκρίνεται σε λίγο λιγότερο χρόνο από την περίοδο του ρολογιού. Δηλαδή το αποτέλεσμα (= 4άδα εντολών) θα είναι διαθέσιμο λίγο πριν την αρνητική ακμή(ξεκίνημα δεύτερης φάσης) του δευτέρου κύκλου. Οι εντολές αυτές λατσάρονται σε αρνητικά ακμοπυροδότητους καταχωρητές. Το ίδιο και τα tags.

Στην αρνητική ακμή λατσάρεται και η προηγούμενη τιμή του PC στον καταχωρητή PCd. Κατά τη δεύτερη φάση του δευτέρου κύκλου γίνονται οι εξής παράλληλες εργασίες. Συγκρίνεται το tag που διαβάστηκε με το κατάλληλο πεδίο της δ/νσης του PC, για να δούμε αν υπάρχει hit. Παράλληλα με βάση τα bits 3:2 του PC ο instruction extractor εξάγει τις χρήσιμες εντολές στον αποκωδικοποιητή. Βγάζει επίσης και τον αριθμό τους (0 ως 4).

Σε κανονικές λειτουργίες η σύγκριση του tag θα έχει οδηγήσει σε hit, οπότε ο instruction buffer θα δέχεται συνέχεια (ιδανικά) τετράδες εντολών και θα εξάγει (ιδανικά) τετράδες εντολών. Αν δεν υπάρχει hit, ένας controller θα αναλαμβάνει να κάνει πρόσβαση στην L2 cache. Στους κύκλους που θα χρειαστούν, ο instruction buffer δεν θα παίρνει νέες εντολές, αλλά θα αδειάζει μόνο τις ήδη υπάρχουσες.

Από το στάδιο fetch μεταφέρονται στο στάδιο decode από 1 έως 4 εντολές, τις οποίες εδώ θα αναφέρουμε ως καινούριες εντολές (new instructions) καθώς επίσης και το πλήθος των καινούριων αυτών εντολών που φορτώθηκαν από την I-cache κατά την διάρκεια του fetch σταδίου. Τα δύο αυτά σήματα, δηλαδή οι καινούριες εντολές το πλήθος των καινούριων εντολών (#ni) αποκτούν σταθερές τιμές αμέσως μετά την αρνητική ακμή του τελευταίου κύκλου ρολογιού του σταδίου fetch. Αυτό σημαίνει ότι τα σήματα των καινούριων εντολών και το σήμα του πλήθους των εντολών λατσάρονται με αρνητικά ακμοπυροδότητους καταχωρητές. Σε αυτό το σημείο θα εξετάσουμε το χρονισμό των σημάτων του σταδίου decode.

Αρχικά πρέπει να παρατηρηθεί ο χρονισμός του σήματος εισόδου του Instruction Buffer (IB). Αυτό το σήμα αποτελεί την έξοδο των πολυπλεκτών του σταδίου. Αυτοί οι πολυπλέκτες με τη σειρά τους έχουν ως είσοδο τα σήματα των καινούριων εντολών από το στάδιο fetch αλλά και το σήμα εξόδου του instruction buffer και το σήμα seli (instruction select, σήμα που καθορίζει την επόμενη τιμή του σήματος εισόδου του instruction buffer και επιλέγει για κάθε εγγραφή του IB την επόμενή της τιμή). Φυσικά το σήμα εξόδου του instruction buffer έχει ήδη λατσάρει σε μία τιμή πριν από την αρνητική ακμή του τελευταίου κύκλου ρολογιού του σταδίου fetch και αυτή η παλιά τιμή του είναι που χρησιμοποιείται από τους πολυπλέκτες. Το σήμα seli υπολογίζεται με βάση τις τιμές των σημάτων n2cons, length και #ni (#ni είναι το πλήθος των καινούριων εντολών). Φυσικά χρησιμοποιείται εδώ η παλιά τιμή του n2cons που υποδηλώνει το πλήθος των

εντολών που προωθήθηκαν στο επόμενο στάδιο στον προηγούμενο κύκλο και επομένως η τιμή του είναι διαθέσιμη πριν την αρνητική ακμή του τελευταίου κύκλου ρολογιού του σταδίου fetch. Το ίδιο ακριβώς ισχύει και για το σήμα length. Το σήμα #ni παίρνει την τελική τους τιμή στην αρνητική ακμή του τελευταίου κύκλου ρολογιού του σταδίου fetch και έτσι έχει σταθερή τιμή λίγο μετά την αρνητική ακμή αυτού του κύκλου ρολογιού. Άρα το σήμα seli έχει σταθερή τιμή λίγο μετά την αρνητική ακμή. Αυτό όμως σημαίνει ότι τα σήματα εισόδου των πολυπλεκτών λατσάρονται στην αρνητική ακμή αυτού του κύκλου ρολογιού αφού όπως είπαμε στην προηγούμενη παράγραφο οι καινούριες εντολές λατσάρονται και αυτές σε αυτήν την αρνητική ακμή και επομένως η έξοδός των πολυπλεκτών λαμβάνουν σταθερή τιμή λίγο μετά από αυτήν την αρνητική ακμή ρολογιού. Τελικά το σήμα εισόδου του Instruction Buffer παίρνει τιμή λίγο μετά την τελευταία αρνητική ακμή του τελευταίου κύκλου ρολογιού του σταδίου fetch.

Το σήμα εξόδου του Instruction Buffer από την πλευρά του σταθεροποιείται σε μία τελική τιμή λίγο μετά την αρνητική ακμή του επόμενου κύκλου ρολογιού, τον οποίο θα ονομάζουμε πρώτο κύκλο ρολογιού του σταδίου decode (δηλαδή παίρνει την τιμή του σήματος εισόδου του Instruction Buffer ένα ακριβώς κύκλο ρολογιού μετά τη σταθεροποίηση της τιμής του σήματος εισόδου). Αυτό συμβαίνει επειδή ο Instruction Buffer είναι ένας αρνητικά ακμοπυροδότητος καταχωρητής. Εδώ πρέπει να διευκρινιστεί ότι όταν λέμε ότι λατσάρεται ένας καταχωρητής σε μία ακμή ρολογιού, αυτό σημαίνει ότι η τιμή του εν λόγω καταχωρητή σταθεροποιείται λίγο μετά από την ακμή αυτή. Αυτό μέχρι τώρα έχει εννοηθεί από τα συμφραζόμενα αλλά καλό θα είναι να γίνει αντιληπτό λόγω της λεπτομέρειας του ζητήματος.

Το σήμα εξόδου του Instruction Buffer αποτελεί την είσοδο των τεσσάρων decodes, δηλαδή των αποκωδικοποιητών που δίνουν ως έξοδο τους καταχωρητές εισόδου και εξόδου των εντολών. Επομένως το σήμα εισόδου των αποκωδικοποιητών έχει σταθερή τιμή λίγο μετά την αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου decode. Τα σήματα εξόδου των αποκωδικοποιητών (δηλαδή τα dest, srca, srcb για κάθε μία από τις τέσσερις το πολύ εντολές) παίρνουν την τιμή τους σχεδόν απευθείας μετά από ορισμένους υπολογισμούς που απαιτούν όμως ελάχιστο χρόνο. Θεωρούμε λοιπόν ότι και η δικιά τους τιμή σταθεροποιείται λίγο μετά την αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου αυτού.

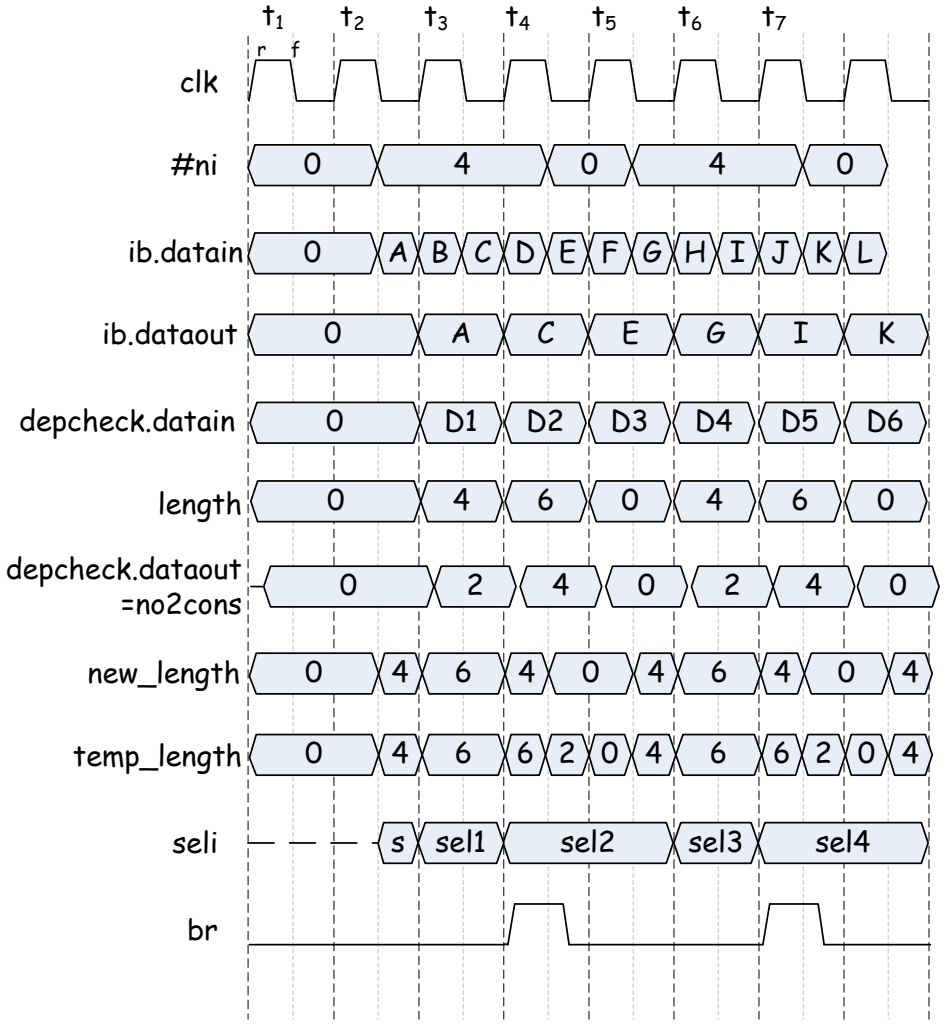
Τα σήματα εξόδου των τεσσάρων αποκωδικοποιητών αποτελούν με τη σειρά τους σήματα εισόδου για την οντότητα dependency check, δηλαδή για τον ελεγκτή εξαρτήσεων τύπου RAW, ο οποίος παράλληλα ελέγχει και για την ύπαρξη διακλάδωσης. Τα σήματα εξόδου του ελεγκτή εξαρτήσεων είναι τα n2cons και branch. Όσων αφορά το χρονισμό των σημάτων εισόδου και εξόδου μπορούμε και εδώ να δούμε ότι τα σήματα εισόδου είναι έτοιμα όπως προείπαμε λίγο μετά την αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου decode και επομένως και τα σήματα εξόδου, δηλαδή τα n2cons και branch, παίρνουν και αυτά την τιμή του αμέσως μετά αλλά ουσιαστικά λίγο μετά την αρνητική αυτή ακμή. Εδώ επίσης θα πρέπει να διευκρινιστεί ότι επειδή οι υπολογισμοί που απαιτούνται για τον υπολογισμό των εξόδων των οντοτήτων που υπάρχουν σε αυτό το στάδιο είναι πολύ γρήγοροι, ουσιαστικά θεωρούμε ότι τη στιγμή που αλλάζουν οι είσοδοι της οντότητας αλλάζουν και οι έξοδοι. Γενικά για τις οντότητες στα διάφορα στάδια που δεν είναι καταχωρητές, δηλαδή η τιμή τους δεν σταθεροποιείται σε ακμές ρολογιού, αλλά η έξοδοί τους είναι απλός υπολογισμός μα βάση τις τιμές των εισόδων τους και δεν έχουν καμία σχέση με ακμές ρολογιού, θα θεωρούμε ότι οι έξοδοί τους αλλάζουν ακριβώς την ίδια στιγμή που αλλάζουν οι είσοδοί τους. Όπως διευκρινίσαμε βέβαια

στην πραγματικότητα η αλλαγή της τιμής των εξόδων γίνεται μετά από ελάχιστο χρονικό διάστημα το οποίο όμως δεν μας ενοχλεί.

Στο στάδιο υπάρχουν όμως όπως προείπαμε και μία οντότητα επιλογέα αλλά και 8 πολυπλέκτες των οποίων οι είσοδοι και οι έξοδοι έχουν ήδη αναφερθεί. Στην οντότητα επιλογέα τα σήματα εισόδου έχουν σταθερή τιμή αμέσως μετά την αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου decode (δηλαδή το length, που όπως θα περιγράψουμε σε λίγο αλλάζει τιμή στην αρνητική αυτή ακμή, το #ni και το n2cons) και επομένως και η τιμή της εξόδου seli σταθεροποιείται εκείνη τη στιγμή. Ακριβώς το ίδιο ισχύει και για τις εισόδους των πολυπλεκτών (seli, καινούριες εντολές, έξοδος του IB) που έχουν σταθερή τιμή αμέσως μετά την αρνητική αυτή ακμή. Επομένως και η έξοδος των πολυπλεκτών αποκτά σταθερή τιμή σε αυτή τη αρνητική ακμή.

Τέλος, επειδή το σήμα length λατσάρεται με αρνητικά ακμοπυροδότητο καταχωρητή, η τιμή του σήματος σταθεροποιείται όπως προείπαμε στην αρνητική ακμή του πρώτου κύκλου ρολογιού του σταδίου decode (με μία τιμή που υπολογίζεται στον προηγούμενο κύκλο).

Ακολουθεί ένα διάγραμμα χρονισμού των σημάτων που αναφέραμε. Στο παράδειγμα αυτό υπάρχει ένα loop που γίνεται δύο φορές (τα 2 branches του παραδείγματος). Το λατσάρισμα των σημάτων του σταδίου γίνεται στις ακμές που αναφέρθηκαν στην ανάλυση που προηγήθηκε.



Σχήμα 4.2 Χρονισμός των σημάτων του σταδίου decode.

Η λειτουργία του decoder σε συνεργασία με το στάδιο fetch θα εξηγηθεί καλύτερα με ένα παράδειγμα. Οι χρονισμοί των σημάτων στο παράδειγμα αυτό φαίνονται στο Σχήμα 4.3.

Στο παράδειγμα μεταξύ των άλλων, περιγράφεται κα ο τρόπος που αντιδρά ο αποκωδικοποιητής σε περίπτωση που δε μπορεί να δεχθεί παραπάνω εντολές ο instruction buffer, καθώς επίσης και σε περίπτωση που βρεθεί unconditional direct branch.

Υποθέτουμε ότι όλες οι προσβάσεις στην instruction cache είναι hit.

Ξεκινάμε με άδειο τον instruction buffer. Ο PC παίρνει την τιμή Α (δ/νση) στην αρνητική ακμή. Η instruction cache (I-cache) παίρνει ως είσοδο την τιμή του PC και σε λίγο λιγότερο από ένα κύκλο, δηλ. λίγο πριν την αρνητική ακμή του δεύτερου κύκλου βγάζει έξοδο (tag+instructions). Η έξοδος αυτή λατσάρεται από αρνητικά ακμοπυροδότητους καταχωρητές (instrd). Έτσι από την αρνητική ακμή (t2f) ο instrd περιέχει τις εντολές (για έναν κύκλο). Ο PCd είναι αρνητικά ακμοπυροδότητος και αποθηκεύει την τιμή που είχε ο PC ένα κύκλο πριν. Δηλ. η τιμή του PCd συμβαδίζει με τις εντολές στον instrd. Το τμήμα tag του instrd συγκρίνεται με το αντίστοιχο τμήμα του PCd, για να ελεγχθεί αν έχουμε hit ή όχι. Το σήμα hit σηκώνεται λίγο μετά την t2f και χρησιμοποιείται από τον decoder στην θετική ακμή t3r. Εδώ να σημειωθεί ότι τόσο ο instrd, όσο και ο PCd έχουν write enable που καθορίζεται από το flip flop wren που είναι θετικά ακμοπυροδότητο και παίρνει τιμή από το σήμα hold που βγάζει ο decoder σε περιπτώσεις που είναι γεμάτος ο instruction buffer και δε μπορεί να δεχθεί εντολές.

Οι εντολές που κρατούνται στον instrd από την αρνητική ακμή t2f, περνάνε από τον instruction extractor και στη συνέχεια περνούν από τους πολυπλέκτες του instruction buffer. Θα πρέπει η συνολική συνδυαστική καθυστέρηση να είναι μικρότερη από την ημιπερίοδο του ρολογιού.

Η υλοποίηση του shifting στον instruction buffer γίνεται με το να έχουμε 8 διαφορετικούς καταχωρητές, έναν για κάθε εντολή, και κάθε τέτοιος καταχωρητής τροφοδοτείται από έναν πολυπλέκτη ο οποίος διαλέγει μεταξύ των 4 νέων εντολών, της ίδιας ή των 4 δεξιότερων (=νεότερων). Τα σήματα επιλογών στους πολυπλέκτες καθορίζονται από μία μονάδα καθορισμού αυτών, η οποία παίρνει ως είσοδο τον pointer length, την έξοδο n2cons και τον αριθμό των νέων εντολών. Όλες αυτές οι πληροφορίες είναι διαθέσιμες κατά το δεύτερο μισό του κύκλου και έτσι (αν η περίοδος είναι κατάλληλη) η έξοδος των πολυπλεκτών λίγο πριν τη θετική ακμή θα έχει σταθεροποιηθεί. Έτσι στην αρνητική ακμή (στο παράδειγμά μας στην t3r), ο instruction buffer θα ανανεωθεί κατάλληλα. Την ίδια στιγμή θα ανανεωθεί και ο pointer length, που δείχνει στην τελευταία (=νεότερη) χρήσιμη εντολή στον instruction buffer.

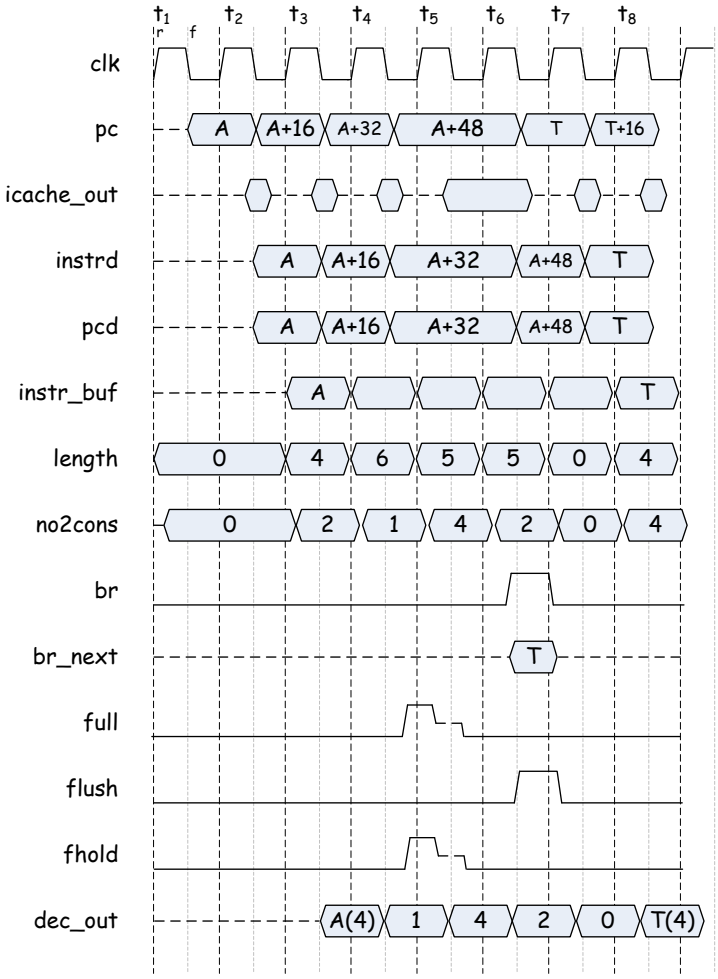
Στον κύκλο t3, από την θετική ακμή t3r, λειτουργεί η αποκωδικοποίηση. Οι εντολές στις θέσεις 1 έως 4 οδηγούνται στους αποκωδικοποιητές εντολών (D0 έως D3).

Οι έξοδοι αυτών, που μεταξύ των άλλων, περιέχουν τους καταχωρητές προέλευσης και προορισμού των εντολών και το αν είναι ή όχι branch, οδηγούνται στον ελεγκτή εξαρτήσεων, ο οποίος βγάζει ως έξοδο (πριν την αρνητική ακμή t3f) τον αριθμό των εντολών που μπορούν να καταναλωθούν-προωθηθούν (n2cons) και το αν υπάρχει branch. Σε περίπτωση που υπάρχει unconditional direct branch, μία μονάδα υπολογισμού διεύθυνσης διακλάδωσης αποκρίνεται πριν την αρνητική ακμή (t3f) με τη διεύθυνση διακλάδωσης που προωθείται στον πολυπλέκτη του PC μαζί με το αντίστοιχο σήμα ελέγχου.

Στο παράδειγμα μας δεν υπάρχει branch, ούτε εξαρτήσεις και έτσι ο ελεγκτής εξαρτήσεων αποκρίθηκε με n2cons=4. Σε συνδυασμό με το length=4 και το #instr=2, που προήλθε από την ανάκληση της επόμενης γραμμής εντολών, το νέο length γίνεται 6 και ανάλογα καθορίζονται και τα σήματα που ελέγχουν τους πολυπλέκτες του instruction buffer. Με την θετική ακμή (t4r) ο instruction buffer περιέχει 6 εντολές και ο καταχωρητής length έχει την τιμή 6.

Η διαδικασία συνεχίζεται. Στο τέλος του τρέχοντος κύκλου (t4) έχουμε length=6, no2cons=1 και #instr=4, δηλ length=9 που δεν χωράνε στον instruction buffer. Γι αυτό πριν την θετική ακμή t5r σηκώνεται το σήμα full που υποδεικνύει στους πολυπλέκτες να μην πάρουν νέες εντολές, και το σήμα f_hold που τροφοδοτεί το flip flop wr_en του τμήματος fetch. Έτσι στον νέο κύκλο (t5) έχουμε length=6-1=5 και οι καταχωρητές PC, instrd και PCd δεν αλλάζουν στην αρνητική ακμή (t5f).
Στον κύκλο t5 καταναλώνονται 4 εντολές. Έτσι στον t6 ο instruction buffer έχει 5 (δεδομένου ότι εισήλθαν 4 νέες εντολές).

Στον κύκλο t6, υπάρχει εντολή unconditional direct branch. Αυτό εντοπίζεται από τον ελεγκτή εξαρτήσεων πριν την αρνητική ακμή t6f και τροφοδοτείται ο pc με τη δ/νση διακλάδωσης. Ο PC την από την αρνητική ακμή t6f έχει την δ/νση διακλάδωση TARGET. Ο αποκωδικοποιητής αντιδρά στο branch σηκώνοντας το σήμα flush, το οποίο αγνοεί τις εντολές στον instruction buffer στον επόμενο κύκλο. Στην ουσία θέτει το length σε 0 στην θετική ακμή του t7. Στον κύκλο t8 ο instruction buffer γεμίζει με εντολές από τη δ/νση TARGET.



Σχήμα 4.3 Χρονισμός των σημάτων των σταδίων fetch και decode.

4.3 Dispatch στάδιο

Το στάδιο dispatch είναι η καρδιά του υπερβαθμωτού επεξεργαστή. Είναι το στάδιο της μετάβασης από in-order σε out of order. Η σχεδίαση στηρίζεται στον αλγόριθμο Tomasulo και την τεχνική register renaming. Διατηρείται ένα πλήθος φυσικών καταχωρητών, οι οποίοι αποτελούν το Renaming Register File (RRF). Για κάθε εντολή που έχει καταχωρητή προορισμού (αρχιτεκτονικό), δεσμεύεται ένας ελεύθερος renaming register από το RRF που θα αποτελέσει τον φυσικό καταχωρητή στον οποίο θα αποθηκευθεί το αποτέλεσμα. Θα κρατηθεί η αντιστοίχιση αρχιτεκτονικού καταχωρητή και καταχωρητή μετονομασίας. Θα τεθεί και μία σημαία busy στον αρχιτεκτονικό καταχωρητή, που σημαίνει ότι δεν είναι διαθέσιμο ακόμα το αποτέλεσμα. Αργότερα, κατά το στάδιο της ολοκλήρωσης η τιμή του καταχωρητή μετονομασίας θα αντιγραφεί στον αρχιτεκτονικό καταχωρητή. Η μετονομασία γίνεται για να εξαλειφθούν οι μη πραγματικές εξαρτήσεις (WAW και WAR) που πηγάζουν από τον περιορισμένο αριθμό καταχωρητών λόγω της επαναχρησιμοποίησης.

Η διαδικασία του dispatch μίας εντολής περιλαμβάνει τις εξής ενέργειες:

- Δέσμευση εγγραφής στον reorder buffer. Ο reorder buffer περιέχει όλες τις εντολές που βρίσκονται εν πτήσει (in flight) στο out of order pipeline. Εκεί διατηρούνται με τη σειρά, έτσι ώστε να μπορούν να ολοκληρωθούν αρχιτεκτονικά στο στάδιο complete. Ο reorder buffer υλοποιείται σαν μία κυκλική ουρά με head και tail. Στη θέση που δείχνει ο δείκτης tail εισέρχεται η νέα εντολή που γίνεται dispatch. Από τη θέση head εξέρχεται η πιο παλιά εντολή που τελείωσε την εκτέλεση της και ολοκληρώνεται αρχιτεκτονικά. Μία εγγραφή στον reorder buffer περιγράφει πλήρως την κατάσταση της εντολής, δηλαδή υπάρχουν πληροφορίες για το αν βρίσκεται σε σταθμό αναμονής, αν έχει εκδοθεί, αν έχει τελειώσει, αν είναι σε speculation path. Περιέχει επίσης το id του αρχιτεκτονικού καταχωρητή και του μετονομασίας.
- Έλεγχος αν υπάρχει ελεύθερη θέση σε σταθμό αναμονής για εκτέλεση εντολών συγκεκριμένου τύπου. Σπάνια δεν θα υπάρχει ελεύθερη θέση.
- Διάβασμα καταχωρητών – ορισμάτων. Κατά την ανάγνωση των αρχιτεκτονικών καταχωρητών που υπάρχουν στις εντολές, αν βρεθεί σημαία busy σημαίνει ότι το αποτέλεσμα δεν είναι διαθέσιμο. Αντί της τιμής θα ληφθεί το tag – διεύθυνση του αντίστοιχου καταχωρητή μετονομασίας.
- Μετονομασία αρχιτεκτονικού καταχωρητή προορισμού με καταχωρητή μετονομασίας από το RRF. Στο αρχιτεκτονικό register file, διατηρούνται πεδία busy και tag. Το πεδίο busy δείχνει ότι ο καταχωρητής έχει μετονομαστεί, ενώ το πεδίο tag υποδεικνύει τον καταχωρητή μετονομασίας.
- Μετακίνηση στον κατάλληλο σταθμό αναμονής (προώθηση δηλαδή της εντολής στο στάδιο execute) και συμπλήρωση της κατάστασης της εντολής στην εγγραφή της στον reorder buffer.

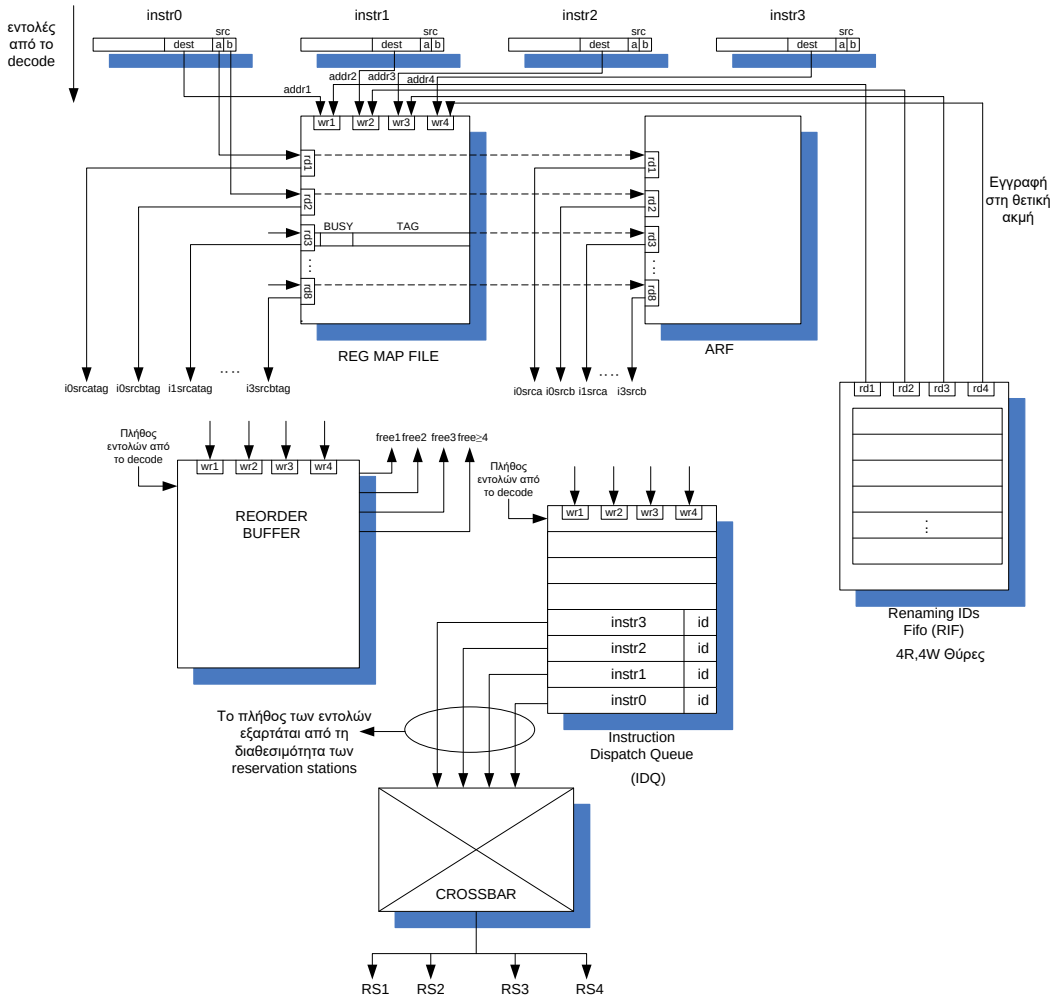
Η διαδικασία όπως περιγράφηκε πιο πάνω απευθύνεται στο dispatch μίας μόνο εντολής. Στο 4-way υπερβαθμωτό επεξεργαστή που σχεδιάζουμε θα πρέπει να γίνεται dispatch έως και 4 εντολών ταυτόχρονα αν είναι δυνατό. Οι εντολές αυτές θα βόλευε να είναι ανεξάρτητες. Ο decoder βγάζει εντολές χωρίς RAW εξαρτήσεις. Θα μπορούσε να χρησιμοποιηθεί μία FIFO ουρά για την μεταβίβαση τετράδων (το πολύ) εντολών από τον decoder. Έτσι το στάδιο dispatch σε κάθε κύκλο θα παίρνει από 1 έως 4 εντολές (ανάλογα με τις RAW εξαρτήσεις) και θα τις κάνει dispatch όπως περιγράφηκε παραπάνω. Θα απαιτηθούν λοιπόν 8 πόρτες ανάγνωσης στο

αρχιτεκτονικό register file (για τα ορίσματα, 2 για κάθε μία από τις 4 εντολές) και 4 πόρτες εγγραφής. Ο reorder buffer θα πρέπει κι αυτός να έχει 4 πόρτες εγγραφής.

4.3.1 Υλοποίηση

Οι αρχιτεκτονικές μονάδες από τις οποίες αποτελείται το στάδιο αυτό είναι το Register Map File (RMF), το Architected Register File (ARF), η ουρά μετονομασίας καταχωρητών (Renaming IDs Fifo, RIF), ο reorder buffer, η Instruction Dispatch Queue (IDQ) και τέλος η μονάδα crossbar. Ακόμα βέβαια υπάρχει και η μηχανή καταστάσεων του σταδίου Dispatch.

Ακολουθεί το Σχήμα 4.4 που απεικονίζει την αρχιτεκτονική του σταδίου dispatch.



Σχήμα 4.4 Το στάδιο dispatch.

Το Register Map File είναι ένα αρχείο αντιστοίχησης των αρχιτεκτονικών καταχωρητών με τους καταχωρητές μετονομασίας (τις ετικέτες δηλαδή των καταχωρητών). Υλοποιείται σαν μία RAM μνήμη με 8 πόρτες ανάγνωσης και 4 πόρτες εγγραφής. Οι 8 θύρες ανάγνωσης χρειάζονται για τα 8 (το μέγιστο) ορίσματα των 4 εντολών που προωθούνται σε κάθε κύκλο ρολογιού από το στάδιο Decode. Για κάθε ένα από τα ορίσματα που υπάρχει (αυτό φαίνεται από κατάλληλη σημαία ύπαρξης του ορίσματος) γίνεται ανάγνωση της αντίστοιχης εγγραφής του RMF. Για

τη διευθυνσιοδότηση μέσα στο RMF γίνεται χρήση των specifiers των ορισμάτων, δηλαδή των αναγνωριστικών τους. Υπενθυμίζουμε ότι τα αναγνωριστικά αυτά είναι 5bit αφού έχουμε 32 αρχιτεκτονικούς καταχωρητές, και αποτελούν πεδία των εντολών που έχουν προωθηθεί από το προηγούμενο στάδιο. Το busy bit των εγγραφών του RMF όταν είναι ίσο με τη μονάδα υποδεικνύει ότι ο αντίστοιχος αρχιτεκτονικός καταχωρητής δεν είναι διαθέσιμος και θα πρέπει να χρησιμοποιηθεί η τιμή της εγγραφής ως ετικέτα για τον καταχωρητή αυτό. Οι ετικέτες αυτές ουσιαστικά αποτελούν αναγνωριστικά των καταχωρητών μετονομασίας (renaming register specifier) τα οποία εγγράφονται στις θέσεις του RMF με βάση τα αναγνωριστικά των καταχωρητών προορισμού-στόχου των 4 εντολών. Επομένως γίνεται διευθυνσιοδότηση στο RMF με βάση τον καταχωρητή αποτελέσματος της εντολής και στην αντίστοιχη θέση εγγράφεται ένα αναγνωριστικό καταχωρητή μετονομασίας, τα οποία αποτελούν εισόδους της μονάδας αυτής. Παράλληλα το busy bit τίθεται στη μονάδα. Με αυτόν τον τρόπο επιτυγχάνεται η αντιστοίχηση αρχιτεκτονικού καταχωρητή και καταχωρητή μετονομασίας. Οι έξοδοι του RMF είναι τα σήματα i0srcatag, i0srcbtag,…, i3srcatag και i3srcbtag, δηλαδή οι ετικέτες των ορισμάτων.

Παράλληλα γίνεται ανάγνωση και από το Architected Register File (ARF). Το ARF υλοποιείται επίσης ως RAM μνήμη και έχει όπως προείπαμε 8 πόρτες ανάγνωσης και 4 πόρτες εγγραφής. Η ενημέρωση του ARF γίνεται σε επόμενο στάδιο (όταν το RRF αντιγράφεται στο ARF) οπότε δεν θα ασχοληθούμε καθόλου με αυτήν σε αυτό το σημείο. Η ανάγνωση αφορά τις τιμές των 8 (το πολύ) ορισμάτων των 4 (το πολύ) εντολών και οι τιμές των καταχωρητών αποθηκεύονται στα σήματα εξόδου του ARF, δηλαδή τα i0srca, i0srcb,…, i3rsca, i3srcb. Το αν θα χρησιμοποιηθούν αυτές οι τιμές ή οι ετικέτες για τους καταχωρητές εξαρτάται όπως έχει ήδη αναλυθεί από το busy bit στις αντίστοιχες θέσεις του RMF.

Η ουρά μετονομασίας καταχωρητών (Renaming IDs Fifo, RIF) είναι μία ουρά τύπου FIFO (First In-First Out) η οποία έχει 4 πόρτες ανάγνωσης και 4 πόρτες εγγραφής. Σε κάθε κύκλο ρολογιού διαβάζονται τόσα id καταχωρητών μετονομασίας όσοι είναι και οι καταχωρητές προορισμού των 4 εντολών (κάθε εντολή μπορεί να έχει έναν καταχωρητή προορισμού ή και κανέναν). Για λόγους απλότητας πάντως υπάρχει περίπτωση ανάγνωσης 4 id καταχωρητών μετονομασίας (σε αυτήν την περίπτωση μπορεί να χαραμίζονται ορισμένα ids). Η RIF ενημερώνεται όταν μία εντολή ολοκληρώσει την εκτέλεσή της και επιστρέψει το id του καταχωρητή μετονομασίας που χρησιμοποίησε κατά την εκτέλεσή της. Αυτό επομένως γίνεται σε επόμενο στάδιο και θα αναλυθεί αργότερα.

Ο reorder buffer υλοποιείται επίσης ως μία FIFO ουρά (κυκλική με head και tail όπως εξηγήθηκε και νωρίτερα) η οποία όμως έχει 8 πόρτες ανάγνωσης και 8 πόρτες εγγραφής, όμως οι 4 πόρτες ανάγνωσης και οι 4 πόρτες εγγραφής είναι τύπου RAM, δηλαδή είναι πόρτες τυχαίας προσπέλασης και χρειάζονται για την δυνατότητα της ανάκαμψης (recovery) σε περίπτωση λανθασμένου branch speculation. Ο μηχανισμός αυτής της ανάκαμψης θα εξηγηθεί εκτενώς σε ειδικό κεφάλαιο. Κάθε εγγραφή του reorder buffer περιέχει το speculation id της εντολής, 5 bits κατάστασης της εντολής (speculation bit, ready bit, completed bit, finished bit και issued bit), το id του αρχιτεκτονικού καταχωρητή και το id του καταχωρητή μετονομασίας. Τα 2 από τα 5 bits κατάστασης (το issued και το finished) αποτελούν εισόδους του σταδίου Dispatch από το στάδιο Execute μαζί με τα αντίστοιχα σήματα επίτρεψης εγγραφής και διεύθυνσης εγγραφής για τα δύο συγκεκριμένα bits. Κάθε φορά έρχεται η πληροφορία για 4 το πολύ εντολές όσων αφορά αυτά τα bits και γίνεται η κατάλληλη ανανέωση στις εγγραφές του reorder buffer με random access εγγραφή των bits

αυτών. Για τα υπόλοιπα 3 bits κατάστασης γίνεται εσωτερικός υπολογισμός μέσα στον reorder buffer.

Η Instruction Dispatch Queue (IDQ) είναι μία κυκλική ουρά τύπου FIFO η οποία έχει 4 πόρτες ανάγνωσης (αφού 4 το πολύ εντολές μπορούν να εισέλθουν στο στάδιο Dispatch) και 4 πόρτες εγγραφής (αφού 4 το πολύ εντολές μπορούν να εξέλθουν από το στάδιο Dispatch και να προωθηθούν στους σταθμούς αναμονής). Αυτή η ουρά έχει ως είσοδο τα σήματα των πλήρως αποκωδικοποιημένων εντολών που έχουν προωθηθεί από το στάδιο Decode. Για την ακρίβεια η κάθε εγγραφή της IDQ αποτελείται από το 2bito σήμα επιλογής της μονάδας επεξεργασίας που αντιστοιχεί στο είδος της εντολής (“00” για την 1η integer unit, “01” για την 2η integer unit, “10” για την branch unit και “11” για την load/store unit), το PC του conditional branch αν αυτό υπάρχει (χρειάζεται για τη σωστή λειτουργία του branch unit), το speculation id της εντολής, το id του καταχωρητή μετονομασίας, σημαίες για τη χρήση ή μη των ετικετών, τις τιμές των αρχιτεκτονικών καταχωρητών, τις ετικέτες για τα ορίσματα της εντολής και τέλος την ίδια την 32bit εντολή. Το πλήθος των εντολών που διαβάζονται από την IDQ και προωθούνται προς τους σταθμούς αναμονής υπολογίζεται με βάση το 2bito σήμα επιλογής μονάδας επεξεργασίας. Όταν στις εντολές που έρχονται από το προηγούμενο στάδιο υπάρχουν πάνω από μία για την ίδια μονάδα επεξεργασίας, δηλαδή για τον ίδιο σταθμό αναμονής, το σήμα no_of_idq_rd (που υποδηλώνει το πλήθος των εντολών που διαβάζονται από την IDQ και προωθούνται στο επόμενο στάδιο) μειώνεται έτσι ώστε να εξαλειφθεί το παραπάνω πρόβλημα. Ακόμα, στις περιπτώσεις που το σήμα stall_disp που αποτελεί είσοδο του σταδίου dispatch και έξοδο του σταδίου execute ισούται με τη μονάδα, τότε το no_of_idq_rd γίνεται ίσο με τη μονάδα ενώ και τα σήματα εξόδου του σταδίου dispatch (δηλαδή οι εντολές που προωθούνται προς στους σταθμούς αναμονής κρατούν την τιμή που είχαν για όσους κύκλους το σήμα αυτό ισούται με τη μονάδα. Το σήμα stall_disp είναι ίσο με τη μονάδα στην περίπτωση που κάποιος από τους σταθμούς αναμονής είναι γεμάτος, κάτι που σημαίνει ότι δεν θα πρέπει να προωθηθούν εντολές στους σταθμούς αναμονής μέχρι να δημιουργηθεί κενή θέση στον γεμάτο σταθμό αναμονής.

Το 2bito σήμα επιλογής της μονάδας επεξεργασίας αποτελεί όπως είπαμε έξοδο της μονάδας crossbar. Αυτή αποτελείται από 4 μικρότερες crossbar μονάδες και μία μεγαλύτερη μονάδα. Κάθε μία από τις 4 μονάδες crossbar αναλαμβάνει μία εντολή και με βάση το opcode της δίνει την κατάλληλη τιμή στο σήμα εξόδου της μονάδας, ex_unit_choice (“00” για integer unit, “10” για branch unit και “11” για load/store unit). Η κεντρική μονάδα crossbar δέχεται ως εισόδους τα σήματα ex_unit_choice1, ex_unit_choice2, ex_unit_choice3, ex_unit_choice4 και ελέγχουν αν υπάρχουν πάνω από μία εντολές για integer unit και στην περίπτωση αυτή μεταβάλει τις τιμές αυτών των σημάτων έτσι ώστε αν για παράδειγμα υπάρχουν 2 εντολές τύπου integer η μία να πηγαίνει στην 1η integer unit και η άλλη στη 2η.

Τέλος όσων αφορά την μηχανή καταστάσεων του σταδίου Dispatch (dispatch state machine) πρέπει να πούμε ότι δέχεται ως εισόδους τα σήματα rb_entry, ren_reg, no_instruction, flush και rb_check_end και βγάζει ως εξόδους τα σήματα stall, stall_rrf και rb_check. Ουσιαστικά όλα αυτά είναι σήματα του ενός bit (σημαίες). Οι καταστάσεις της μηχανής είναι οι OK, NO_INSTR, NO_RESOURCE και RB_CHECKING. Αρχικά η μηχανή βρίσκεται στην κατάσταση NO_INSTR αφού δεν υπάρχουν εντολές στο στάδιο. Για τα σήματα εξόδου ισχύει:

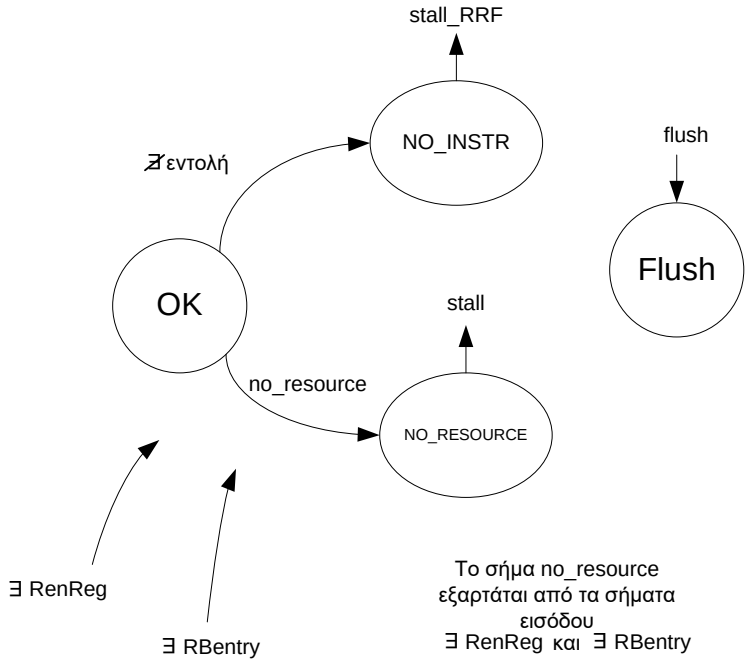
```
stall_rrf<='1' when state=NO_INSTR else '0';
stall<='1' when state=NO_RESOURCE else '0';
rb_check<='1' when state=RB_CHECKING else '0';
```

no_resources<=((not rb_entry) or (not ren_reg));

Το σήμα no_resources ουσιαστικά υποδεικνύει το αν υπάρχουν πηγές του συστήματος ή όχι. Αν δηλαδή δεν υπάρχει ελεύθερη θέση στον reorder buffer ή αν είναι άδεια η RIF δεν μπορεί να συνεχιστεί η τροφοδότηση του σταδίου Dispatch από το pipeline και έτσι το τελευταίο πρέπει να stallάρει. Το σήμα stall σε αυτή την περίπτωση γίνεται μονάδα. Αυτό το σήμα το είχαμε δει και σε προηγούμενο στάδιο ως σήμα εισόδου και τώρα γίνεται κατανοητό από πού προέρχεται και τι υποδηλώνει. Στην περίπτωση που δεν υπάρχουν εντολές πρέπει να καθυστερήσουν τα επόμενα στάδια, δηλαδή να μην κάνουν άσκοπη χρήση των πηγών του συστήματος (πχ να μην διαβάζονται id από την RIF). Τα σήματα rb_check και rb_check_end έχουν σχέση με το recovery από λανθασμένες προβλέψεις διακλαδώσεων υπό συνθήκη και θα αναλυθούν αργότερα.

Στο στάδιο Dispatch υπάρχουν και άλλες μονάδες που σχετίζονται με το recovery από λανθασμένες προβλέψεις διακλαδώσεων υπό συνθήκη και θα αναλυθούν επίσης αργότερα. Τέλος υπάρχουν και οι σταθμοί αναμονής οι οποίοι όμως θα αναλυθούν εκτενέστερα στο στάδιο Execute.

Τώρα ακολουθεί το σχήμα της state machine του σταδίου.



Σχήμα 4.5 State machine του σταδίου dispatch.

4.3.2 Χρονισμός των σημάτων του σταδίου Dispatch

Από το στάδιο Decode μεταφέρονται στο στάδιο Dispatch από 1 έως 4 εντολές χωρίς να υπάρχουν εξαρτήσεις τύπου RAW (read after write) οι οποίες είναι και αυτές που υφίστανται στην πραγματικότητα. Αυτό συμβαίνει λόγω του ελέγχου αυτού του είδους εξαρτήσεων στο στάδιο Decode και την απάλειψη τους, αφού προωθούνται στο επόμενο στάδιο μόνο εντολές που δεν έχουν RAW εξαρτήσεις μεταξύ τους. Το πλήθος αυτών των εντολών περιέχεται στο σήμα dec_num που

ουσιαστικά αποτελεί το n2cons (number to consume) του σταδίου Decode. Το dec_num και οι dec_num το πλήθος εντολές (instructions) που εισέρχονται στο στάδιο Dispatch για περαιτέρω επεξεργασία έχουν σταθερή τιμή μετά την θετική ακμή του ρολογιού του τελευταίου κύκλου του σταδίου Decode. Σε αυτό το σημείο θα εξετάσουμε τον χρονισμό των σημάτων του σταδίου Dispatch.

Αρχικά πρέπει να δούμε πότε λαμβάνουν τιμή τα σήματα no_of_rd και no_of_wr τα οποία περιέχουν το πλήθος των αναγνώσεων και των εγγραφών αντίστοιχα στο Register Map File. Όπως είναι λογικό η τιμή των σημάτων αυτών εξαρτάται από το πλήθος των εντολών dec_num που είναι όπως προείπαμε έτοιμο λίγο μετά την θετική ακμή του ρολογιού. Έτσι τα σήματα no_of_rd και no_of_wr παίρνουν τιμή λίγο αργότερα αλλά σίγουρα πριν τη θετική ακμή του επόμενου κύκλου ρολογιού και αυτό γιατί δεν απαιτείται κάποιος χρονοβόρος υπολογισμός. Πρέπει εδώ να τονίσουμε ότι η απαίτηση που υπάρχει είναι ότι τα σήματα αυτά πρέπει να έχουν σταθερή τιμή πριν την αρνητική ακμή του επόμενου κύκλου ρολογιού για να έχουν υπολογιστεί και τα αντίστοιχα σήματα rd και wr (enables για τις αναγνώσεις και τις εγγραφές) πριν την αρνητική ακμή έτσι ώστε να μπορούν να γίνουν οι αντίστοιχες εγγραφές και αναγνώσεις στην αρνητική ακμή. Εντελώς αντίστοιχα παίρνουν τιμή και τα σήματα enables που χρειάζονται για να δείξουν την ύπαρξη ή μη της κάθε μίας από τις 4 εντολές καθώς επίσης και των αντίστοιχων dest, srca και srcb καταχωρητών. Αυτές οι τιμές έχουν υπολογιστεί από το προηγούμενο στάδιο και μεταφέρονται σε ορισμένα bits των εισόδων instr0, instr1, instr2 και instr3 του σταδίου Dispatch, των τεσσάρων δηλαδή εντολών. Προφανώς ανάλογα με το dec_num μπορούν εύκολα να εντοπιστούν οι εντολές με en = 1 και αυτές με en = 0 (πχ αν dec_num = 2 τότε τα enables των instr0, instr1 θα ισούνται με 1 ενώ τα enables των instr2 και instr3 θα ισούνται με 0 αφού δεν θα υπάρχουν στην πραγματικότητα αυτές οι εντολές λόγω RAW εξαρτήσεων ή λόγω κάποιου branch). Επομένως τα σήματα enables είναι και αυτά έτοιμα λίγο μετά το dec_num και τις instructions αλλά και σαφώς πριν τη θετική ακμή του επόμενου κύκλου ρολογιού.

Όσων αφορά το Register Map File μπορούμε να παρατηρήσουμε ότι τα δεδομένα ανάγνωσης παίρνουν τιμή (ή αλλιώς «λατσάρονται») στην αρνητική ακμή εκείνου του κύκλου ρολογιού στον οποίο έχει, πριν την αρνητική ακμή και μέχρι τουλάχιστον την αρνητική ακμή, σταθερή μη μηδενική τιμή το σήμα no_of_rd. Έτσι τα δεδομένα ανάγνωσης έχουν σταθερή τιμή λίγο μετά την αρνητική ακμή. Τα δεδομένα εγγραφής του Register Map File είναι έτοιμα επίσης μετά την αρνητική ακμή του κύκλου ρολογιού. Αυτό οφείλεται στο γεγονός ότι τα δεδομένα αυτά προέρχονται από τη Renaming Ids Fifo (RIF) και αποτελούν ουσιαστικά δεδομένα ανάγνωσης αυτής της ουράς, τα οποία είναι έτοιμα όπως θα δούμε αμέσως μετά την αρνητική ακμή του κύκλου ρολογιού (την ίδια αρνητική ακμή όπου λατσάρονται και τα δεδομένα ανάγνωσης του RMF). Η εγγραφή τελικά αυτών των δεδομένων εγγραφής στο Register Map File γίνεται στην αμέσως επόμενη θετική ακμή του ρολογιού.

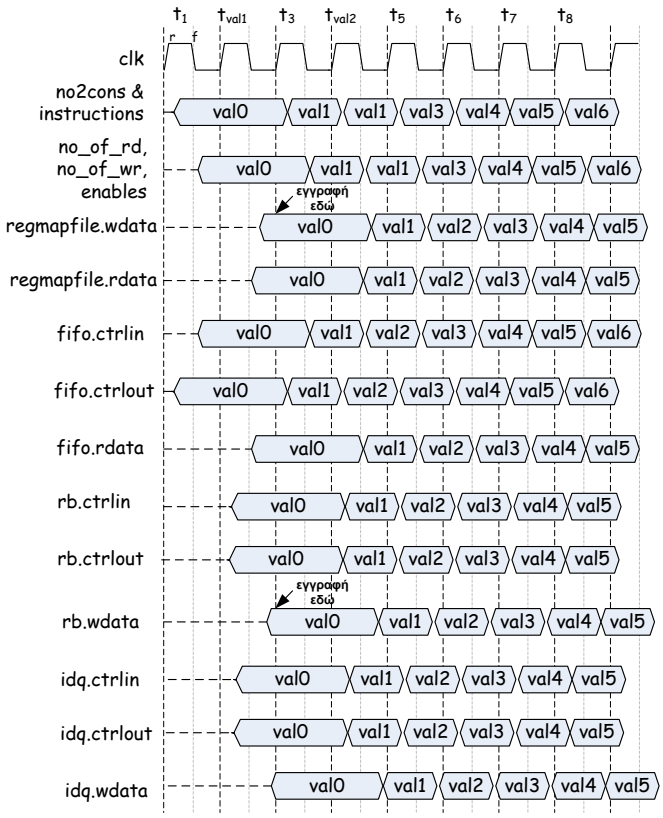
Για την Renaming Ids Fifo είναι εύκολο να διαπιστωθεί ότι τα σήματα ελέγχου εισόδου, όπως είναι τα σήματα enables που αναφέραμε νωρίτερα είναι ήδη έτοιμα πριν την θετική ακμή του πρώτου κύκλου ρολογιού (όπου είναι έτοιμα τα σήματα no_of_rd, no_of_wr και enables). Τα σήματα ελέγχου εξόδου της ουράς αυτής, δηλαδή τα σήματα full και empty (τα οποία δείχνουν αν η ουρά είναι γεμάτη ή άδεια αντιστοίχως), έχουν την παλιά τους τιμή πριν την ίδια θετική ακμή (και χρησιμοποιούνται αυτές οι τιμές τους για τη σωστή λειτουργία της state machine που έχει υλοποιηθεί για το στάδιο Dispatch) και αποκτούν τη νέα τους τιμή λίγο μετά τη θετική ακμή του ρολογιού. Τα δεδομένα ανάγνωσης από την άλλη πλευρά της RIF

είναι έτοιμα μετά την αρνητική ακμή αυτού του κύκλου ρολογιού και όπως ήδη έχουμε πει αποτελούν τα δεδομένα εγγραφής του Register Map File.

Στο στάδιο Dispatch υπάρχει ακόμα και ο Reorder Buffer στον οποίο κρατούνται οι εντολές που βρίσκονται εν πτήσει (in flight) στο out of order pipeline με διάφορα στοιχεία για την κατάστασή τους (αν βρίσκονται σε speculation path, κατάσταση εντολής, ετικέτα μετονομασίας αν υπάρχει κτλ.). Στο συγκεκριμένο στάδιο τα σήματα ελέγχου εισόδου του Reorder Buffer (σε αυτή την περίπτωση είναι το dec_num) είναι έτοιμα λίγο πριν την αρνητική ακμή του κύκλου ρολογιού, ενώ τα αντίστοιχα σήματα ελέγχου εξόδου, δηλαδή τα σήματα full και empty έχουν την παλιά τους τιμή πριν την ίδια αρνητική ακμή (και χρησιμοποιούνται αυτές οι τιμές τους για τη σωστή λειτουργία της state machine που έχει υλοποιηθεί για το στάδιο Dispatch) και αποκτούν τη νέα τους τιμή λίγο μετά τη θετική ακμή του ρολογιού. Παρατηρείται λοιπόν μία ολίσθηση μισού κύκλου ρολογιού στα σήματα του Reorder Buffer και αυτό γιατί είναι απαραίτητα τα δεδομένα ανάγνωσης του Register Map File (δηλαδή τα tags για τους καταχωρητές ορίσματα των εντολών) τα οποία όμως έχουν σταθεροποιηθεί στην κατάλληλη τιμή τους αμέσως μετά την αρνητική ακμή του κύκλου ρολογιού. Έτσι τελικά τα δεδομένα εγγραφής του Reorder Buffer είναι έτοιμα λίγο πριν την θετική ακμή του 2^{ου} κύκλου ρολογιού και η εγγραφή γίνεται αμέσως μετά την θετική ακμή του κύκλου αυτού.

Τέλος ακριβώς αντίστοιχη με τον Reorder Buffer είναι η περίπτωση της Instruction Dispatch Queue (IDQ). Ο χρονισμός για τους ίδιους λόγους παραμένει ο ίδιος. Έτσι υπάρχει και εδώ ολίσθηση των σημάτων κατά μισό κύκλο ρολογιού (παρότι και εδώ τα σήματα είναι έτοιμα από πιο νωρίς, εκτός φυσικά από τα δεδομένα εγγραφής που είναι και εδώ η αιτία της ολίσθησης). Στην IDQ τα δεδομένα εγγραφής, δηλαδή οι εντολές, περιέχουν τα tags τα οποία αποτελούν όπως προείπαμε δεδομένα ανάγνωσης του Register Map File και είναι διαθέσιμα λίγο μετά την αρνητική ακμή του 1^{ου} κύκλου ρολογιού. Επομένως τα σήματα ελέγχου εισόδου είναι έτοιμα πριν την αρνητική ακμή του κύκλου, τα σήματα ελέγχου εξόδου είναι έτοιμα λίγο μετά την αρνητική ακμή και τέλος τα δεδομένα εγγραφής είναι έτοιμα λίγο πριν την θετική ακμή του 2^{ου} κύκλου. Η εγγραφή γίνεται και σε αυτήν την περίπτωση με την θετική ακμή του 2^{ου} κύκλου ρολογιού.

Όλα τα παραπάνω φαίνονται με ακρίβεια στο παρακάτω σχήμα.



Σχήμα 4.6 Χρονισμός των σημάτων του σταδίου dispatch.

Φυσικά οι ακολουθίες των τιμών val0, val1,…, val6 είναι διαφορετικές για κάθε σήμα και χρησιμοποιούνται για την αντιστοιχία των χρονικών στιγμών που πάιρνουν την τιμή τους τα διάφορα σήματα.

4.3.3 Branch recovery

Το branch misprediction recovery, η ανάκαμψη δηλαδή από λανθασμένη πρόβλεψη διακλάδωσης, είναι μία πολύ σημαντική λειτουργία του superscalar επεξεργαστή. Στον MICROPROC αυτή η λειτουργία επιτελείται από δύο κατάλληλες μονάδες οι οποίες εκτελούν τις ενέργειες που απαιτούνται για το recovery με έξοδο κατάλληλων σημάτων και ανανέωση των εγγραφών των reorder buffer, RIF και RMF. Οι δύο αυτές μονάδες είναι η μονάδα rb checker (reorder buffer check unit) και η μονάδα speculation id checker (speculation id check unit).

Η μονάδα ελέγχου του reorder buffer (rb checker) δίνει ως έξοδο τις διευθύνσεις ανάγνωσης τυχαίας προσπέλασης του reorder buffer που χρησιμοποιεί η μονάδα ελέγχου του speculation id (speculation id checker) για τον έλεγχο των εγγραφών του reorder buffer. Ο έλεγχος των εγγραφών του reorder buffer ξεκινά από τη θέση tail και τελειώνει στην θέση head του reorder buffer, όταν και ο rb checker θέτει στην μονάδα τη σημαία rb_check_end, κάτι το οποίο σημαίνει την λήξη του ελέγχου του reorder buffer και ουσιαστικά τη λήξη της διαδικασίας ανάκαμψης από τη λανθασμένη πρόβλεψη. Οι διευθύνσεις αυτές βγαίνουν ως έξοδος ανά τετράδες, αφού γίνεται ταυτόχρονος έλεγχος 4 θέσεων του reorder buffer. Έτσι, στον πρώτο κύκλο της διαδικασίας ανάκαμψης οι 4 διευθύνσεις ανάγνωσης τυχαίας προσπέλασης του reorder buffer (που αποτελούν όπως είπαμε έξοδοι του rb checker) θα είναι οι tail, tail+1, tail+2 και tail+3 στην περίπτωση φυσικά που κάποια από αυτές τις διευθύνσεις δεν είναι η head. Από την τετράδα των διευθύνσεων που περιέχουν την head,

προωθούνται στον speculation id checker οι διευθύνσεις έως και την head (δηλαδή λιγότερες ή ίσες από 4 διευθύνσεις).

Ο speculation id checker από την πλευρά του δέχεται ως σήμα εισόδου το wrong_spec_id, δηλαδή το λανθασμένο speculation id του μονοπατιού για το οποίο υπήρξε λανθασμένη πρόβλεψη. Στην συνέχεια ελέγχει τις εγγραφές του reorder buffer ,οι οποίες διαβάστηκαν με βάση τις τιμές των διευθύνσεων ανάγνωσης τυχαίας προσπέλασης του reorder buffer, και δίνει ως έξοδο τις αντίστοιχες σημαίες επίτρεψης εγγραφής του RMF και το πλήθος των αναγνωριστικών καταχωρητών μετονομασίας που μπορούν να επιστραφούν στην RIF. Ο έλεγχος περιλαμβάνει αρχικό έλεγχο του speculation bit των εγγραφών (4 το πολύ όπως εξηγήθηκε σε κάθε κύκλο ρολογιού) του reorder buffer και εν συνεχεία σύγκριση των speculation id (αν οι εντολές ανήκουν σε κάποιο μονοπάτι πρόβλεψης) με το wrong_spec_id. Αν μία εντολή ανήκει σε λανθασμένο μονοπάτι τότε με βάση την ειδική σημαία επίτρεψης εγγραφής του RMF θα μηδενιστεί το busy bit της αντίστοιχης εγγραφής για να μην υπάρξει λάθος χρήση του RMF από εντολές που έπονται ενώ θα επιστραφεί και ο αντίστοιχος καταχωρητής μετονομασίας στην RIF. Επίσης καθορίζονται και οι εγγραφές της IDQ που πρέπει να γίνουν flush.

Φυσικά αυτό προϋποθέτει ύπαρξη πολυπλεκτών για τα σήματα εισόδου και επίτρεψης εγγραφής και ανάγνωσης των μονάδων RMF, reorder buffer,IDQ και RIF που επιλέγουν τις κατάλληλες τιμές με βάση τις σημαίες rb_check και rb_check_end. Προφανώς όταν η σημαία rb_check ισούται με τη μονάδα δεν υπάρχει κανονική λειτουργία του συστήματος και δεν εγγράφονται νέα στοιχεία στις παραπάνω μονάδες, ενώ stallάρουν και τα προηγούμενα στάδια.

Τέλος πρέπει να αναφερθεί ότι και στο στάδιο Execute ελέγχονται και απομακρύνονται οι αντίστοιχες εντολές που ανήκουν σε λανθασμένο μονοπάτι διακλάδωσης αλλά αυτό θα αναλυθεί εκτενέστερα στην αντίστοιχη παράγραφο.

4.4 Execute στάδιο

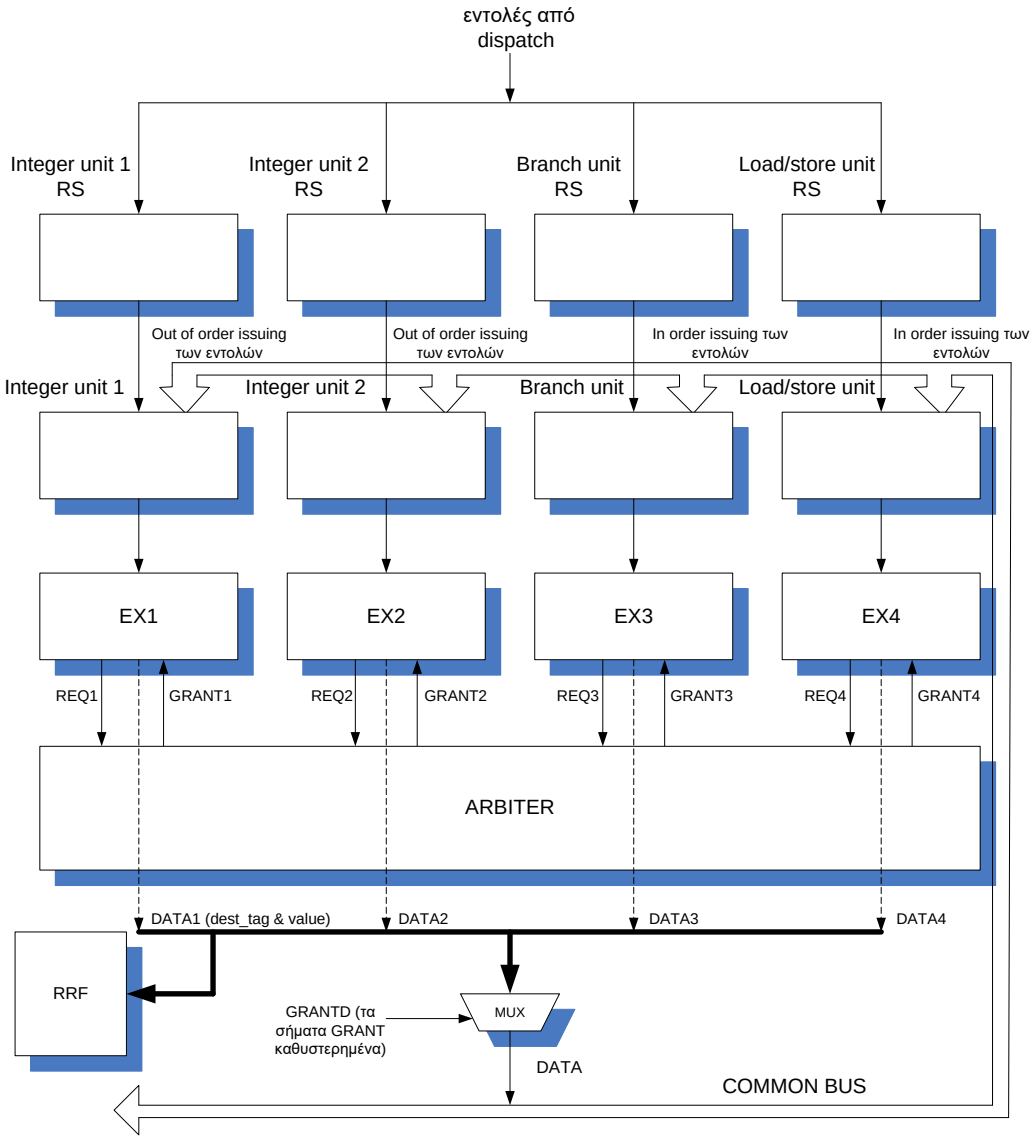
Το στάδιο execute είναι αυτό στο οποίο οι εντολές εκτελούνται κανονικά με τις σωστές τιμές των ορισμάτων τους και ενημερώνουν το renaming register file (RRF). Στο στάδιο execute προωθούνται μόνο εντολές που είναι έτοιμες, δηλαδή έχουν έτοιμα όλα τους τα ορίσματα. Ουσιαστικά αυτός ο έλεγχος των ορισμάτων λαμβάνει χώρα στους σταθμούς αναμονής (reservation stations) οι οποίοι αποτελούν τον καταχωρητή διασύνδεσης των σταδίων dispatch και execute.

Ένας σταθμός αναμονής αντιστοιχεί σε μία μονάδα εκτέλεσης και επιτελεί 3 βασικές λειτουργίες. Η πρώτη είναι να δέχεται εντολές μαζί με τα ορίσματα τους, όπως περιγράφηκε παραπάνω. Αν κάποιο όρισμα δεν είναι διαθέσιμο, διατηρείται το tag του renaming register. Σε τέτοια περίπτωση η εντολή δεν είναι έτοιμη προς εκτέλεση (έκδοση). Εδώ λαμβάνει χώρα η δεύτερη λειτουργία του σταθμού αναμονής. Όπως έχει αναφερθεί, υπάρχει ένα κοινό bus με οδηγούς τις μονάδες εκτέλεσης και δέκτες τους σταθμούς αναμονής. Αν κάποια εντολή εκτελεστεί (τελειώσει η εκτέλεσή της), η μονάδα εκτέλεσης οδηγεί το κοινό bus με το αποτέλεσμα μαζί με το αντίστοιχο tag (=id καταχωρητή μετονομασίας αρχιτεκτονικού προορισμού εντολής που εκτελέστηκε). Η δεύτερη λειτουργία του σταθμού αναμονής έγκειται στο διαρκές έλεγχο του common bus για «δημοσιοποιημένο» αποτέλεσμα που τον ενδιαφέρει. Το κάνει αυτό διαβάζοντας το tag που «δημοσιεύεται» και συγκρίνοντας το με τα tags των εγγραφών του. Αν βρεθεί

ταίριασμα διαβάζεται η αντίστοιχη τιμή και ενημερώνεται η εγγραφή. Μία εγγραφή χωρίς tag, αλλά μόνο με τιμές, είναι έτοιμη προς εκτέλεση (έκδοση). Αυτή ακριβώς είναι η τρίτη λειτουργία. Με βάση κάποια πολιτική (πχ Last First) επιλέγεται μία από τις ready εντολές στο σταθμό αναμονής και εκδίδεται. Η έκδοση μπορεί να γίνεται κάθε κύκλο, ή κάθε συγκεκριμένο αριθμό κύκλων, ανάλογα με τη μονάδα εκτέλεσης και με το αν είναι ή όχι pipelined. Για παράδειγμα κάποια floating point μονάδα εκτέλεσης μπορεί να μην επέτρεπε έκδοση κάθε κύκλο.

Οι εντολές που προωθούνται στις αντίστοιχες μονάδες εκτέλεσης εκτελούνται και βγάζουν τα αποτελέσματα τους στο common bus, ενώ παράλληλα ενημερώνουν το RRF αρχείο. Η κάθε μία από τις δύο integer units αποτελείται από μία ALU η οποία με βάση το opcode (και συχνά το πεδίο funct) της αριθμητικής/λογικής εντολής επιτελεί την κατάλληλη λειτουργία και βγάζει ως έξοδο την τιμή του αποτελέσματος και τον καταχωρητή μετονομασίας ο οποίος πρέπει να ενημερωθεί. Όπως είπαμε τα αποτελέσματα αυτά οδηγούνται στο common bus και στο RRF. Αυτά τα αποτελέσματα βγαίνουν και από τις branch και load/store units (υπάρχει μία branch και μία load/store unit) όταν αυτές έχουν καταχωρητή στόχου-προορισμού. Φυσικά η branch unit βγάζει ως αποτέλεσμα και πληροφορία για το αν ένα branch είναι σωστό ή λανθασμένο, ενώ στην περίπτωση που το branch είναι λανθασμένο εξάγεται πληροφορία και για το λανθασμένο speculation id και για την σωστή τιμή του PC που πρέπει να χρησιμοποιηθεί από τον πολυπλέκτη επιλογής του PC του σταδίου fetch για την εξαγωγή της σωστής τιμής του PC. Αυτή η σωστή τιμή είναι φυσικά η διεύθυνση στην I-cache της επόμενης εντολής από την εντολή διακλάδωσης που αποδείχθηκε λανθασμένη. Υπενθυμίζεται εδώ ότι όλα αυτά ισχύουν λόγω του ότι ακολουθείται πολιτική branch taken για τις διακλαδώσεις υπό συνθήκη. Η load/store unit από την άλλη πλευρά διαβάζει από την D-cache τις τιμές των καταχωρητών σε κάθε περίπτωση μίας εντολής φόρτωσης δεδομένων (load instruction) ενώ εγγράφει σε μία διεύθυνση της D-cache μόνο αν η εντολή γίνει non-speculative στην περίπτωση που αυτή ήταν τέτοια. Ακόμα αξίζει να αναφερθεί ότι χρησιμοποιούνται οι μέθοδοι load bypassing και load forwarding που έχουν αναλυθεί σε προηγούμενο κεφάλαιο και οι οποίες αυξάνουν την ταχύτητα επεξεργασίας της συγκεκριμένης μονάδας.

Ακολουθεί το Σχήμα 4. με την αρχιτεκτονική του σταδίου Execute.



Σχήμα 4.7 Το στάδιο execute.

4.4.1 Υλοποίηση

Στον επεξεργαστή MICROPROC υπάρχουν τέσσερις σταθμοί αναμονής. Δύο από αυτούς αντιστοιχούν στις δύο integer units (μονάδα ακεραίων), μία αντιστοιχεί στην branch unit (μονάδα διακλάδωσης) και μία αντιστοιχεί στην load/store unit (μονάδα φόρτωσης/αποθήκευσης δεδομένων). Οι δύο σταθμοί αναμονής των δύο integer units αποτελούνται από τέσσερις θέσεις ενώ χρησιμοποιούν out-of-order issuing των εντολών στις μονάδες επεξεργασίας. Αντίθετα οι σταθμοί αναμονής των branch και load/store unit αποτελούνται από δύο μόλις θέσεις και χρησιμοποιούν in-order issuing των εντολών στις αντίστοιχες μονάδες επεξεργασία. Στους σταθμούς αναμονής τελούνται οι διαδικασίες που ήδη έχουν αναφερθεί με διαρκή έλεγχο του common bus και σύγκριση των tags των ορισμάτων των εντολών που βρίσκονται στους σταθμούς αναμονής με τα αναγνωριστικά των καταχωρητών μετονομασίας που «δημοσιοποιούνται» στο common bus μαζί με την τιμή τους. Σε περίπτωση που κάποιο tag είναι το ίδιο με το αναγνωριστικό του καταχωρητή μετονομασίας που «δημοσιοποιήθηκε» στο common bus, η τιμή του καταχωρητή μετονομασίας (που έχει επίσης «δημοσιοποιηθεί» στο bus) εγγράφεται στο αντίστοιχο πεδίο της

αντίστοιχης εγγραφής του σταθμού αναμονής και το bit που δείχνει αν ισχύει το tag ή η τιμή του καταχωρητή μηδενίζεται, κάτι που σημαίνει ότι το αντίστοιχο όρισμα της εντολής είναι έτοιμο και επομένως η τιμή του πεδίου της τιμής καταχωρητή είναι έγκυρη και μπορεί να χρησιμοποιηθεί. Παράλληλα γίνεται έλεγχος για τα ορίσματα της εντολής. Αρχικά ελέγχεται το αν η εντολή έχει δύο ορίσματα ή ένα με τον έλεγχο των bits που υποδηλώνουν την ύπαρξη ή μη των ορισμάτων και εν συνεχεία ελέγχεται για τα ορίσματα που υπάρχουν, το κατά πόσο αυτά είναι έτοιμα με τον έλεγχο των bits που δείχνουν αν ισχύουν τα tags ή οι τιμές των καταχωρητών (όπως είναι φανερό υπάρχουν πεδία με τα tags και πεδία με την τιμή των καταχωρητών ορισμάτων στις εγγραφές των σταθμών αναμονής). Στους σταθμούς αναμονής των branch και load/store unit που γίνεται in-order issuing των εντολών αυτοί οι έλεγχοι γίνονται μόνο για την εντολή που βρίσκεται στην πρώτη θέση του σταθμού αναμονής. Αντίθετα, στους σταθμούς αναμονής των integer units αυτός ο έλεγχος γίνεται σε όλες τις busy εγγραφές των σταθμών αναμονής και μόλις μία εντολή γίνει ready και μέσω του αλγορίθμου επιλογής ανάμεσα στις ready εντολές γίνει issued μηδενίζεται το busy bit της αντίστοιχης εγγραφής. Ο αλγόριθμος επιλογής ανάμεσα στις ready εντολές ουσιαστικά προωθεί την παλιότερη εντολή και γίνεται με βάση το πεδίο time των εγγραφών των σταθμών αναμονής το οποίο αυξάνεται σε κάθε κύκλο ρολογιού. Ταυτόχρονα βγαίνουν ως έξοδοι και τα σήματα data_ready_aux1,… ,data_ready_aux4 τα οποία δείχνουν ότι θα υπάρξουν έτοιμα δεδομένα για «δημοσιοποίηση» στο common bus. Ουσιαστικά για αυτό το σκοπό χρησιμοποιούνται τα αντίστοιχα σήματα data_ready που αποτελούν ολισθήσεις των προηγούμενων σημάτων (για λόγους σωστού χρονισμού). Ακόμα έξοδοι αποτελούν και τα σήματα stall1, stall2, stall3 και stall4 που χρησιμοποιούνται από την IDQ και γίνονται ίσα με τη μονάδα όταν και τα αντίστοιχα σήματα full γίνουν ίσα με τη μονάδα.

Όλοι οι σταθμοί αναμονής βγάζουν τις εξόδους τους στη μονάδα allocator που είναι υπεύθυνη για την δέσμευση των εγγραφών των σταθμών αναμονής με βάση τα σήματα ex_unit_choice των εντολών που έχουν έρθει από το στάδιο dispatch και τα σήματα full των σταθμών αναμονής. Η μονάδα αυτή βγάζει ως εξόδους τα issued_bit1, issued_bit2, issued_bit3, issued_bit4, issued_wren1, issued_wren2, issued_wren3, issued_wren4 και issued_rb_wr_addr1, issued_rb_wr_addr2, issued_rb_wr_addr3, issued_rb_wr_addr4 που αποτελούν εξόδους και τους σταδίου Execute προς το στάδιο Dispatch για την σωστή ενημέρωση των status bits στον reorder buffer. Επίσης βγάζει ως εξόδους τα σήματα integer_unit1_data_in, integer_unit2_data_in, br_unit_data_in, load_store_unit_data_in. Αυτά τα σήματα αποτελούν τις εισόδους των αντίστοιχων μονάδων επεξεργασίας και όπως προείπαμε έχουν έτοιμες τις τιμές των ορισμάτων τους. Επίσης έξοδοι του allocator είναι και τα σήματα data_ready_aux τα οποία προαναφέρθηκαν. Τέλος υπάρχουν και τα σήματα stall1, stall2, stall3, stall4 που αποτελούν εξόδους των σταθμών αναμονής και χρησιμοποιούνται για την καθυστέρηση όσο κάποιο από αυτά τα σήματα ισούται με την μονάδα της IDQ στο στάδιο dispatch. Όταν κάποιος σταθμός αναμονής επομένως είναι γεμάτος το συνολικό σήμα stall_disp του σταδίου execute γίνεται ίσο με τη μονάδα και μέσω κατάλληλων πολυπλεκτών στην IDQ δεν προωθούνται εντολές ούτως ώστε αυτές να μην χαθούν.

Οι μονάδες επεξεργασίας των εντολών ακεραίων (integer units) ανάλογα με τα πεδία opcode και funct επιτελούν στην ALU τους την κατάλληλη ενέργεια εκτελώντας την πράξη της εντολής. Έτσι μπορεί να επιτελέσει την πράξη της άθροισης των τιμών των ορισμάτων, της αφαίρεσης, της ολίσθησης κ.ο.κ. Όλες αυτές οι πράξεις και οι λειτουργίες τους φαίνονται αναλυτικά στον πίνακα του ISA του

MICROPROC που υπάρχει σε προηγούμενο κεφάλαιο. Ως έξοδος βγαίνει επίσης και το σήμα rrf_wren που αποτελεί επίτρεψη εγγραφής του RRF και τίθεται στη μονάδα όταν υπάρχει έτοιμο αποτέλεσμα.

Η μονάδα επεξεργασίας των εντολών διακλάδωσης (branch unit) ανάλογα πάλι με τα πεδία opcode και funct εκτελούν το branch με έλεγχο της συνθήκης διακλάδωσης στην περίπτωση των conditional branches και εν συνεχεία τον κατάλληλο υπολογισμό των εξόδων ή με την απλή ανάθεση της μονάδας στο σήμα br_correct στην περίπτωση των unconditional branches. Παράλληλα υπολογίζονται η τιμή του καταχωρητή προορισμού και τίθεται το σήμα rrf_wren όταν υπάρχει καταχωρητής προορισμού και η τιμή αυτού έχει υπολογισθεί. Έξοδοι της μονάδας αυτής αποτελούν φυσικά και τα σήματα br_correct, br_incorrect, flush, wrong_spec_id και pc_from_branch_unit που υπολογίζονται με βάση την επίλυση του branch (δηλαδή τον έλεγχο της συνθήκης διακλάδωσης αν αυτή υπάρχει, δηλαδή αν το branch δεν είναι unconditional). Φυσικά με βάση το σήμα flush και το wrong_spec_id γίνεται και misspredict recovery στο στάδιο αυτό με κατάλληλο έλεγχο των εγγραφών των σταθμών αναμονής και μηδενισμό των αντίστοιχων busy bits.

Η μονάδα επεξεργασίας εντολών φόρτωσης/αποθήκευσης δεδομένων (load/store unit) με βάση το πεδίο opcode της εντολής εκτελεί την κατάλληλη λειτουργία. Στην περίπτωση της φόρτωσης δεδομένων γίνεται αρχικά έλεγχος στις εγγραφές μία ουράς που υπάρχει στην μονάδα και η οποία περιέχει όλες τις εντολές αποθήκευσης (δηλαδή τη διεύθυνση όπου πρέπει να γίνει εγγραφή στην D-cache και τα δεδομένα εγγραφής) οι οποίες μπορεί να είναι είτε completed είτε finished. Όταν πρέπει να γίνει φόρτωση από μία διεύθυνση που υπάρχει σε αυτή την ουρά τότε τα δεδομένα προωθούνται απευθείας από την ουρά χωρίς να χρειάζεται προσπέλαση στη μνήμη δεδομένων D-cache. Αυτή όπως έχουμε αναλύσει σε προηγούμενο κεφάλαιο είναι η τεχνική load forwarding. Σε διαφορετική περίπτωση γίνεται προσπέλαση στη μνήμη. Το load bypassing επιτυγχάνεται με την κατάλληλη προώθηση των εντολών φόρτωσης πριν από εντολές αποθήκευσης που έχουν διαφορετική διεύθυνση (όπου δηλαδή δεν υπάρχει σύγκρουση). Επίσης στην περίπτωση flush έχουμε πάλι τις κατάλληλες λειτουργίες στην ουρά με τις εντολές αποθήκευσης που είναι finished αλλά όχι ακόμα completed. Το σήμα rrf_wren τίθεται και εδώ στη μονάδα στην περίπτωση που υπάρχουν δεδομένα έτοιμα για εγγραφή στο RRF.

Εδώ πρέπει να προστεθεί ότι όλες οι μονάδες επεξεργασίας εντολών (integer unit1, integer unit2, branch unit και load/store unit) βγάζουν ως εξόδους τα σήματα finished_bit1, finished_bit2, finished_bit3, finished_bit4, finished_wren1, finished_wren2, finished_wren3, finished_wren4 και finished_rb_wr_addr1, finished_rb_wr_addr2, finished_rb_wr_addr3, finished_rb_wr_addr4 (όπου το 1 αντιστοιχεί στην integer unit1, το 2 στην integer unit2, το 3 στην branch unit και το 4 στην load/store unit) που αποτελούν εξόδους και τους σταδίου Execute προς το στάδιο Dispatch για την σωστή ενημέρωση των status bits στον reorder buffer.

Μία άλλη μονάδα του σταδίου είναι η μονάδα που επιτελεί την ολίσθηση κατά ένα κύκλο ρολογιού των σημάτων data_ready_aux και έχει ως εξόδους τα σήματα data_ready. Αυτή η ολίσθηση γίνεται για τη σωστή λειτουργία της μονάδα execution unit που είναι υπεύθυνη για τη «δημοσιοποίηση» των αποτελεσμάτων των εντολών στο common bus.

Η μονάδα execution unit δέχεται ως εισόδους τις εξόδους των μονάδων επεξεργασίας και τα data_ready1, data_ready2, data_ready3, data_ready4 και βγάζει ως έξοδο το σήμα data που αποτελεί ουσιαστικά το σήμα που προωθείται στο common bus (αποτελείται από το renaming tag και την τιμή αποτελέσματος του

καταχωρητή μετονομασίας). Στη μονάδα αυτή υπάρχουν τέσσερις μηχανές καταστάσεων ex1, ex2, ex3 και ex4, μία για κάθε μονάδα επεξεργασίας (το 1 σε όλα αυτά τα σήματα που έχουν αναφερθεί αντιστοιχεί στην integer unit1,το 2 στην integer unit2, το 3 στην branch unit και το 4 στην load/store unit). Οι καταστάσεις της κάθε μίας μηχανής καταστάσεων είναι οι IDLE, REQUEST και GRANTED. Αρχικά και γενικά όταν δεν υπάρχουν δεδομένα προς «δημοσιοποίηση» κάθε μηχανή καταστάσεων βρίσκεται στην κατάσταση IDLE. Όταν μία από αυτές τις μονάδες έχει δεδομένα προς «δημοσιοποίηση» (κάτι που ελέγχεται από το αντίστοιχο σήμα data_ready) θέτει το σήμα request στη μονάδα, μεταβαίνει στην κατάσταση REQUEST και περιμένει μέχρι το αντίστοιχο σήμα grant να γίνει μονάδα από την μονάδα διαιτησίας arbiter που είναι υπεύθυνη για τον έλεγχο του bus. Όταν το σήμα grant γίνει μονάδα τα δεδομένα «δημοσιοποιούνται στο bus και η μηχανή περνάει στην κατάσταση GRANTED για ένα κύκλο. Αμέσως μετά επιστρέφει στην κατάσταση IDLE αν το data_ready είναι μηδέν ή στην κατάσταση REQUEST αν το data_ready είναι μονάδα (που σημαίνει ότι υπάρχει νέα αίτηση χρήσης του διαδρόμου). Για την σωστή υλοποίηση (δηλαδή για να μην χάνονται δεδομένα προς «δημοσιοποίηση» γίνεται χρήση ουράς για την προσωρινή αποθήκευση των δεδομένων που καθυστερούν να «δημοσιοποιηθούν».

Η μονάδα arbiter ουσιαστικά παίρνει ως εισόδους τα σήματα req1, req2, req3 και req4 και βγάζει ως εξόδους τα σήματα grant1, grant2, grant3 και grant4. Ο αλγόριθμος διαιτησίας που χρησιμοποιείται είναι αρκετά δίκαιος και δεν υπάρχουν μεγάλες καθυστερήσεις. Τα σήματα grant1, grant2, grant3 και grant4 ολισθαίνουν κατά μισό κύκλο ρολογιού με χρήση τεσσάρων κατάλληλων μονάδων ολίσθησης οι οποίες βγάζουν ως έξοδο σε αρνητικές ακμές κύκλων ρολογιού τα σήματα grantd1, grantd2, grantd3 και grantd4 (grant delayed).

Αυτά τα σήματα μαζί με τα σήματα εξόδου των ex1, ex2, ex3, ex4 (δηλαδή τα δεδομένα προς «δημοσιοποίηση», τα οποία όμως έχουν έγκυρη τιμή όταν το αντίστοιχο σήμα grantd ισούται με τη μονάδα) αποτελούν τις εισόδους του πολυπλέκτη δεδομένων που υπάρχει στην μονάδα execution unit. Αυτός ο πολυπλέκτης είναι που τελικά βγάζει ως έξοδο το σωστό σήμα data, που αποτελεί και τα τελικά δεδομένα που προωθούνται στο common bus.

Τέλος στο στάδιο αυτό υπάρχει και το αρχείο RRF το οποίο έχει τέσσερις πόρτες εγγραφής και τέσσερις πόρτες ανάγνωσης. Η χρήση του έχει εξηγηθεί ενδελεχώς και δεν είναι απαραίτητο να γίνουν περαιτέρω σχόλια.

4.4.2 Χρονισμός των σημάτων του σταδίου Execute

Το στάδιο Execute έχει ως σήματα εισόδου τα σήματα data_in1, data_in2, data_in3, data_in4 και no_of_idq_rd που αποτελούν εξόδους του σταδίου Dispatch και τα σήματα rrf_rd_addr1, rrf_rd_addr2, rrf_rd_addr3, rrf_rd_addr4 που αποτελούν εξόδους του σταδίου Complete και θα αναλυθούν αργότερα. Τα σήματα data_in1, data_in2, data_in3, data_in4 και no_of_idq_rd έχουν σταθερή τιμή αμέσως μετά την αρνητική ακμή του τελευταίου κύκλου ρολογιού του σταδίου Dispatch. Τα σήματα rrf_rd_addr1, rrf_rd_addr2, rrf_rd_addr3, rrf_rd_addr4 έχουν σταθερή τιμή λίγο μετά την θετική ακμή του τελευταίου κύκλου ρολογιού του σταδίου Complete.

Τα σήματα εισόδου της μονάδας allocator αποτελούν ουσιαστικά τις εισόδους του σταδίου, αφού είναι τα data_in1, data_in2, data_in3, data_in4 και no_of_idq_rd και όπως είπαμε έχουν σταθερή τιμή μετά την αρνητική ακμή του τελευταίου κύκλου του σταδίου Dispatch. Τα σήματα εξόδου της μονάδας αυτής εξαρτώνται από τον χρόνο που περιμένει η κάθε εντολή στον αντίστοιχο σταθμό αναμονής μέχρι να

διαβάσει τα κατάλληλα δεδομένα από τον κοινό διάδρομο και να γίνει έτοιμη για εκτέλεση. Πάντως αν είναι ελεύθερος ο αντίστοιχος σταθμός αναμονής η εγγραφή γίνεται μετά ακριβώς από έναν κύκλο (στην αρνητική ακμή του επόμενου κύκλου) και τα αντίστοιχα σήματα issued_bit, wren και wr_addr αποκτούν σταθερή τιμή στην αμέσως επόμενη θετική ακμή ρολογιού. Επίσης τα σήματα integer_unit1_data_in, integer_unit2_data_in, br_unit_data_in και load_store_unit_data_in που εξαρτώνται από τον χρόνο αναμονής στον σταθμό αναμονής αποκτούν σταθερή τιμή πάντα αμέσως μετά από κάποια θετική ακμή ρολογιού (λατσάρονται δηλαδή με θετικά ακμοπυροδότητους καταχωρητές).

Τα σήματα αυτά αποτελούν εισόδους των τεσσάρων αντίστοιχων μονάδων εκτέλεσης εντολών. Τα σήματα εξόδων κάθε μονάδας από αυτές, δηλαδή τα value dest_tag, rrf_wren, finished_bit και finshed_rb_wr_addr, αποκτούν σταθερή τιμή στην θετική ακμή του επόμενου κύκλου ρολογιού αφού λατσάρονται και αυτά με θετικά ακμοπυροδότητους καταχωρητές. Σε αυτή την ακμή (δηλαδή ένα ακριβώς κύκλο αφού αποκτήσει τιμή το σήμα εισόδου της εκάστοτε μονάδας επεξεργασίας) λατσάρονται και τα υπόλοιπα σήματα εξόδου, όπως είναι τα σήματα br_correct, br_incorrect, flush, pc_from_br_unit και wrong_spec_id για την branch unit και η μνήμη D-cache για την load/store unit.

Όσων αφορά τα σήματα της μονάδας execution unit, που είναι υπεύθυνη για τον έλεγχο του κοινού διαδρόμου, τα σήματα εισόδου της είναι ουσιαστικά τα value και dest_tag οπότε είναι έτοιμα μετά από θετική ακμή κύκλου ρολογιού αλλά και τα σήματα data_ready που λατσάρονται όπως έχουμε ξαναπεί με αρνητικά ακμοπυροδότητους καταχωρητές και ουσιαστικά έχουν σταθερή τιμή αμέσως μετά την αρνητική ακμή του αμέσως προηγούμενου κύκλου ρολογιού. Τα σήματα req είναι έτοιμα αμέσως μετά από θετικές ακμές κύκλων ρολογιού και τα αντίστοιχα grant αμέσως μετά από αρνητικές ακμές κύκλων ρολογιού. Το σήμα grantd(i) αποκτά σταθερή τιμή ακριβώς ένα κύκλο μετά το αντίστοιχο granti, i = 1, 2, 3, 4 (αφού είναι το ίδιο σήμα ολισθημένο κατά ένα κύκλο). Στην αρνητική ακμή του επόμενου κύκλου ρολογιού παίρνει την τελική τιμή του και το σήμα data_from_listener, δηλαδή το σήμα του διαδρόμου (δηλαδή δύο κύκλους ρολογιού μετά την αλλαγή του σήματος grant το αντίστοιχο data «δημοσιοποιείται» στον διάδρομο).

Τέλος στο RRF τα rrf_wren και τα σήματα των διευθύνσεων παίρνουν σταθερή τιμή σε θετικές ακμές κύκλων ρολογιού και η εγγραφή γίνεται στην αμέσως επόμενη θετική ακμή κύκλου ρολογιού. Ο χρονισμός σε σχέση με τα υπόλοιπα σήματα του σταδίου καθορίζεται από τα rrf_wren που είδαμε προηγουμένως πότε λαμβάνουν σταθερή τιμή.

Λόγω της εξάρτησης των χρονισμών των σημάτων αυτού του σταδίου από τους χρόνους αναμονής στους σταθμούς αναμονής δεν θα παρουσιαστεί εδώ διάγραμμα χρονισμού. Οι χρονισμοί των σημάτων και οι εξαρτήσεις τους γίνονται εύκολα αντιληπτές από το παρών υποκεφάλαιο.

4.5 Complete στάδιο

Κατά το στάδιο της ολοκλήρωσης επιτελείται αρχιτεκτονική ολοκλήρωση έως και 4 εντολών. Στην συγκεκριμένη σχεδίαση γίνεται αρχιτεκτονική ολοκλήρωση μίας μόνο εντολής. Το στάδιο αυτό είναι in-order και ολοκληρώνει εντολές με σειρά παλαιότητας. Η αρχιτεκτονική ολοκλήρωση έγκειται στην αντιγραφή του καταχωρητή μετονομασίας στον αντίστοιχο αρχιτεκτονικό. Μετά το πέρας της

ολοκλήρωσης αποδεσμεύονται οι αντίστοιχες εγγραφές στον reorder buffer και ενημερώνεται κατάλληλα το Register Map File.

4.5.1 Υλοποίηση και χρονισμός των σημάτων του σταδίου Complete

Στο στάδιο αυτό υπάρχει μόνο μία μονάδα, η complete_unit που έχει ως είσοδο το σήμα compl_unit_data_in και το no_of_rb_rd που αποτελούν εξόδους του σταδίου Dispatch. Βέβαια το no_of_rb_rd είναι πάντα ίσο με τη μονάδα αφού ολοκληρώνεται όπως είπαμε μία μόνο εντολή σε κάθε κύκλο. Το compl_unit_data_in είναι ουσιαστικά η εγγραφή της θέσης head (έτσι επιτυγχάνεται η σειρά παλαιότητας) του reorder buffer που διαβάζεται όταν η εντολή είναι finished. Τα σήματα εξόδου της μονάδας αυτής είναι τα rrf_rd_addr1, rrf_rd_addr2, rrf_rd_addr3, rrf_rd_addr4, arf_wr_addr1, arf_wr_addr2, arf_wr_addr3, arf_wr_addr4 και rmf_and_rif_update που αποτελούν εισόδους των σταδίων Dispatch και Execute. Τα σήματα αυτά αποκτούν τιμή στην θετική ακμή του κύκλου ρολογιού που ακολουθεί τον κύκλο ρολογιού στον οποίο αποκτά σταθερή τιμή το σήμα no_of_rb_rd και το compl_unit_data_in. Το σήμα rrf_rd_addr1 χρειάζεται για να διαβαστεί η αντίστοιχη εγγραφή του RRF και το σήμα arf_wr_addr1 για να γίνει στον επόμενο κύκλο η εγγραφή του ARF. Τέλος το σήμα rmf_and_rif_update χρησιμοποιείται στο στάδιο Execute για να μηδενιστεί το busy bit της κατάλληλης εγγραφής του Register Map File και να επιστραφεί στην RIF ο καταχωρητής μετονομασίας.

5 Μεθοδολογία σχεδίασης

Στο κεφάλαιο αυτό γίνεται η περιγραφή της μεθοδολογίας σχεδίασης του επεξεργαστή. Αυτή η μεθοδολογία περιλαμβάνει τις εξής ενέργειες:

- Εκτίμηση επιδόσεων.
- Συγγραφή RTL.
- Verification strategy.
- Debugging.
- Σύνθεση.

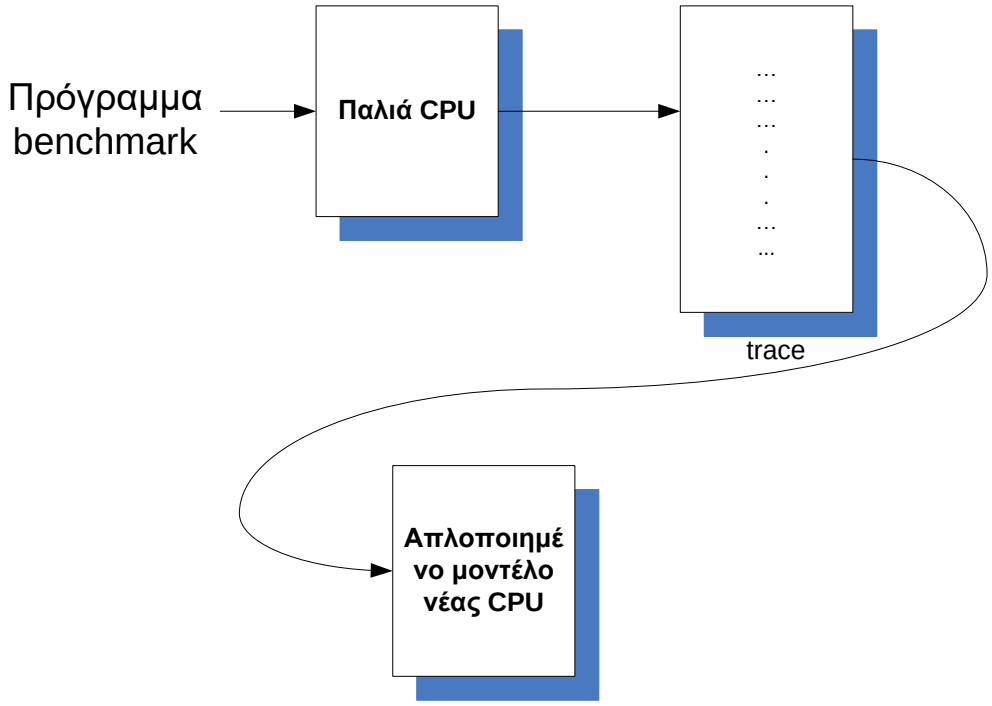
Στη συνέχεια γίνεται μία μικρή ανάλυση των ενεργειών αυτών.

5.1 Εκτίμηση επιδόσεων (Performance evaluation)

Στη σύγχρονη σχεδίαση μικροεπεξεργαστών, η απευθείας σχεδίαση υλικού (hardware prototyping) είναι ακατόρθωτη. Οι περισσότεροι σχεδιαστές σήμερα χρησιμοποιούν προσομοιωτές (simulators) για να κάνουν εκτιμήσεις επιδόσεων και για να διασφαλίσουν την λειτουργική ορθότητα κατά τη διάρκεια της διαδικασίας σχεδίασης. Συνήθως χρησιμοποιούνται δύο είδη simulators, οι functional simulators (προσομοιωτές λειτουργικότητας) και οι performance simulators (προσομοιωτές επίδοσης). Οι functional simulators μοντελοποιούν μία μηχανή στο επίπεδο της αρχιτεκτονικής συνόλου εντολών (ISA) και χρησιμοποιούνται για την πιστοποίηση της ορθής εκτέλεσης των προγραμμάτων. Αυτοί οι προσομοιωτές ουσιαστικά εκτελούν τις εντολές ενός προγράμματος. Οι performance simulators μοντελοποιούν

την μικροαρχιτεκτονική μίας σχεδίασης και χρησιμοποιούνται για την μέτρηση των κύκλων μηχανής που απαιτούνται για την εκτέλεση ενός προγράμματος.

Οι performance simulators μπορεί να λειτουργούν είτε με χρήση ίχνους (trace driven) είτε με χρήση εκτέλεσης (execution driven). Οι trace-driven performance simulators επεξεργάζονται προπαρασκευασμένα ίχνη για να καθορίσουν τον αριθμό κύκλων που απαιτούνται για την εκτέλεση των εντολών των ιχνών. Στην ουσία μία παλιά CPU του ίδιου ISA εκτελεί ένα τεράστιο benchmark πρόγραμμα (πρόγραμμα ελέγχου που αποτελείται από χιλιάδες εντολές) και δημιουργεί ένα ίχνος εντολών το οποίο εκτελείται από το απλοποιημένο μοντέλο της νέας CPU που σχεδιάζεται. Κατά την εκτέλεση αυτή ουσιαστικά μετράται το πλήθος των εντολών που ολοκληρώνονται αρχιτεκτονικά κατά μέσο όρο σε κάθε κύκλο, δηλαδή το IPC. Το benchmark πρόγραμμα που χρησιμοποιείται είναι πολύ μεγάλο έτσι ώστε να μπορεί να δοθεί μία πλήρης πληροφορία για το IPC της νέας CPU. Αυτή η διαδικασία παρουσιάζεται στο Σχήμα 5.1. που ακολουθεί.



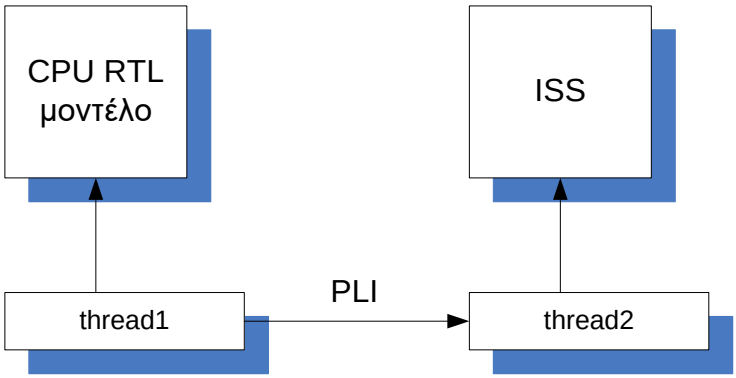
Σχήμα 5.1 Trace driven performance simulator.

5.2 Συγγραφή RTL

Η συγγραφή του RTL είναι το βασικό κομμάτι της σχεδίασης αφού κατά τη διάρκεια αυτού ορίζονται οι αρχιτεκτονικές οντότητες του επεξεργαστή και οι σχέσεις των εισόδων/εξόδων τους. Για τον καθορισμό των αρχιτεκτονικών οντοτήτων της σχεδίασης χρησιμοποιείται μία γλώσσα περιγραφής υλικού. Τέτοιες γλώσσες είναι για παράδειγμα η Verilog και η VHDL. Σε αυτή τη διπλωματική έγινε χρήση της γλώσσας VHDL. Κατά τη συγγραφή του RTL μοντέλου δίνεται έμφαση στις μεθόδους υλοποίησης της αρχιτεκτονικής. Ο κώδικας από τον οποίο αποτελείται το RTL μοντέλο δίνει μια πλήρη περιγραφή για όλα τα στοιχεία που χρησιμοποιούνται στον επεξεργαστή, όπως είναι οι διάφοροι buffers, οι caches κτλ. Επίσης μέσω του κώδικα υλοποιούνται οι χρονισμοί των σημάτων που απαιτούνται για τη σωστή λειτουργία και σύνδεση των οντοτήτων αυτών.

5.3 Verification strategy

Η verification strategy ή αλλιώς στρατηγική επαλήθευσης της σχεδίασης είναι ένα πολύ σημαντικό κομμάτι της συνολικής διαδικασίας σχεδίασης. Σε αυτήν την περίπτωση γίνεται η επαλήθευση της ορθής λειτουργίας του επεξεργαστή με χρήση του RTL μοντέλου του επεξεργαστή και ενός Instruction Set Simulator (ISS) ο οποίος είναι συμμορφωμένος με το ISA του σχεδιασμένου επεξεργαστή. Στο τέλος κάθε κύκλου το RTL μοντέλο της CPU ανανεώνει τη μνήμη και τους αρχιτεκτονικούς καταχωρητές της CPU την ίδια ώρα που και ο Instruction Set Simulator ανανεώνει τη δικιά του μνήμη και τους δικούς του αρχιτεκτονικούς καταχωρητές. Ουσιαστικά κάθε φορά που το RTL μοντέλο της CPU τελειώνει την εκτέλεση μίας εντολής καλείται μία συνάρτηση plitask() και μέσω του PLI interface που υπάρχει ανάμεσα στο RTL μοντέλο και τον Instruction Set Simulator, ο τελευταίος εκτελεί με τη σειρά του την ίδια εντολή. Στο τέλος κάθε κύκλου συγκρίνονται τα δύο ARF ενώ η σύγκριση της μνήμης, η οποία λόγω του μεγάλου μεγέθους της μνήμης είναι χρονοβόρα και απαιτεί πολλούς υπολογισμούς, γίνεται στο τέλος της εκτέλεσης του προγράμματος ελέγχου (πρόγραμμα testbench). Ουσιαστικά βέβαια οι εντολές του προγράμματος που χρησιμοποιείται έχουν τη μορφή ενός ίχνους (thread1) το οποίο τροφοδοτεί το RTL μοντέλο και εν συνεχεία μέσω του PLI δίνει την εντολή στο άλλο ίχνος (thread2) για να τροφοδοτήσει τον ISS. Στο Σχήμα 5.2 που ακολουθεί φαίνεται καθαρά η διασύνδεση που υπάρχει ανάμεσα στο RTL μοντέλο της CPU και τον ISS.



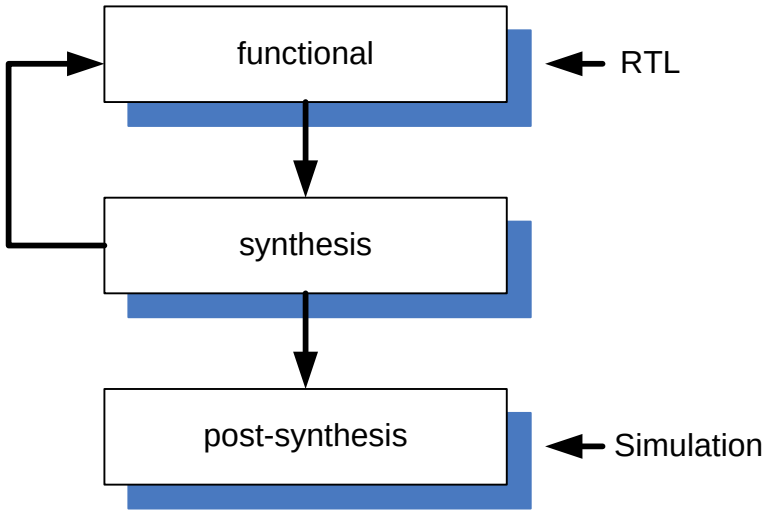
Σχήμα 5.2 Διασύνδεση RTL μοντέλου και ISS.

5.4 Debugging

Το debugging είναι ένα πολύ σημαντικό στάδιο της σχεδίασης και είναι απαραίτητο για την ολοκλήρωση της σχεδίασης. Κατά τη διάρκεια του debugging γίνεται η διόρθωση των λαθών που υπάρχουν στον κώδικα. Αυτή η διαδικασία γίνεται με τον λεπτομερή έλεγχο των σημάτων των διαφόρων σταδίων και την προσπάθεια επικέντρωσης στα σημεία όπου υπάρχουν λάθη, τα σημεία δηλαδή όπου τα σήματα δεν αποκτούν τις επιθυμητές και προβλεπόμενες από τη σχεδίαση τιμές. Συνήθη λάθη στη σχεδίαση επεξεργαστών είναι τα λάθη χρονισμών, όταν για παράδειγμα χρησιμοποιείται λανθασμένα ως είσοδος σε μία μονάδα η είσοδος ενός καταχωρητή και όχι η έξοδός του.

5.5 Σύνθεση

Όταν το μοντέλο RTL έχει τελειώσει τότε αυτό περνάει από εργαλεία σύνθεσης για την παραγωγή του κυκλώματος της σχεδίασης, με την παράλληλη μέτρηση των αριθμών των πυλών που απαιτούνται για το κύκλωμα, του χρόνου του κρίσιμου μονοπατιού (critical path) και της ισχύος που καταναλώνει το κύκλωμα. Στη σύνθεση οι μνήμες παίρνονται συνήθως έτοιμες ως megacells, ενώ τα εργαλεία σύνθεσης χρησιμοποιούν για τα behavioral μοντέλα του κώδικα τις διάφορες βιβλιοθήκες που περιέχουν. Τις περισσότερες φορές γίνεται σύνθεση κατά τη διάρκεια της συγγραφής του RTL έτσι ώστε να ανατροφοδοτείται η διαδικασία της συγγραφής με βάση τα αποτελέσματα της σύνθεσης. Τελικά αφού τελειώσει το κομμάτι της συγγραφής του RTL μοντέλου και έχει ολοκληρωθεί η σύνθεση των επιμέρους τμημάτων του επεξεργαστή, γίνεται και μία συνολική σύνθεση ολόκληρης της σχεδίασης (post-synthesis). Όλα αυτά φαίνονται στο Σχήμα 5.3 που ακολουθεί.



Σχήμα 5.3 Η διαδικασία σύνθεσης.

6 ΠΑΡΑΡΤΗΜΑ

6.1 Προσομοίωση

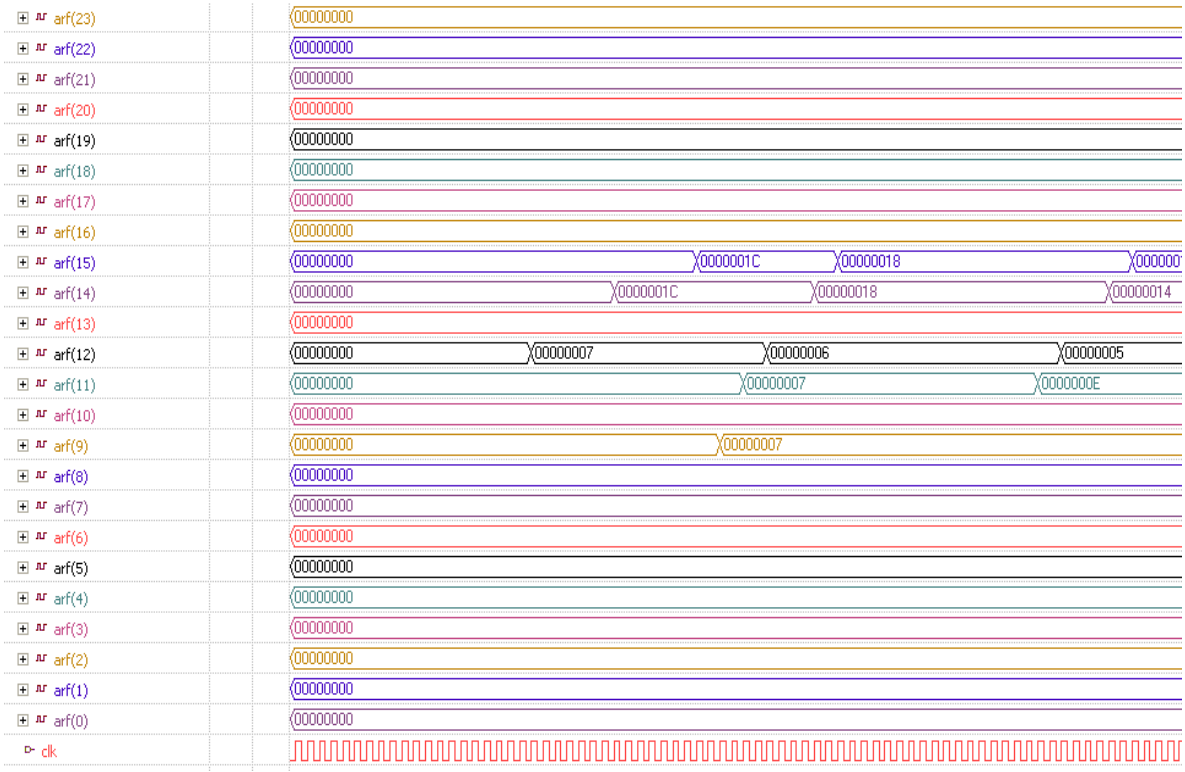
Ακολουθούν ορισμένες εικόνες από την προσομοίωση ενός προγράμματος σε γλώσσα assembly από το Active-HDL 3.6. Το πρόγραμμα που εκτελέστηκε είναι το ακόλουθο :

```
[0x00400000]   0x3c010000  lui $1, 0 [count]          ; 4: lw $t4,count
[0x00400004]   0x8c2c0000  lw $12, 0($1) [count]
[0x00400008]   0x21ce0001  addi $14, $14, 1            ; 5: add $t6,$t6,1
[0x0040000c]   0x8dc90000  lw $9, 0($14)              ; 6: lw $t1,($t6)
[0x00400010]   0x01695820  add $11, $11, $9           ; 7: add $t3,$t3,$t1
[0x00400014]   0x218cffff  addi $12, $12, -1          ; 8: sub $t4,$t4,1
```

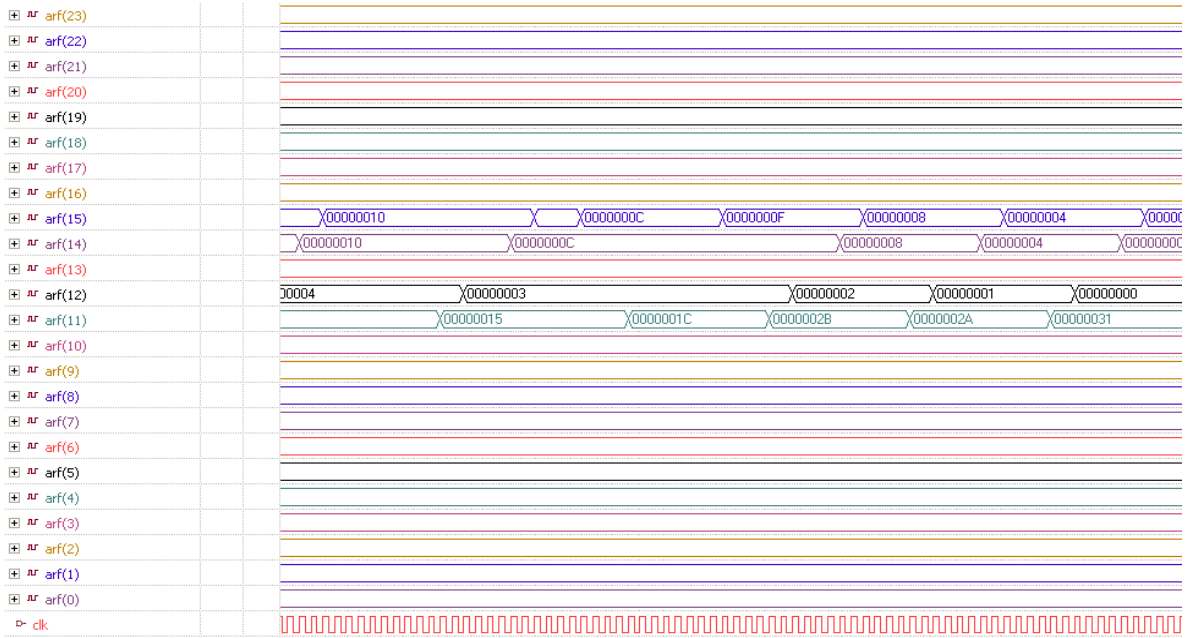
[0x00400018] 0x0581fffc bgez $12 -16 [loop-0x00400018] ; 9: bgez $t4,loop
[0x0040001c] 0x000b58c2 srl $11, $11, 3 ; 10: srl $t3,$t3,3

με τη μνήμη να είναι αρχικοποιημένη. Το πρόγραμμα δίνει το τρέχον άθροισμα συνεχόμενων θέσεων μνήμης σε έναν καταχωρητή.

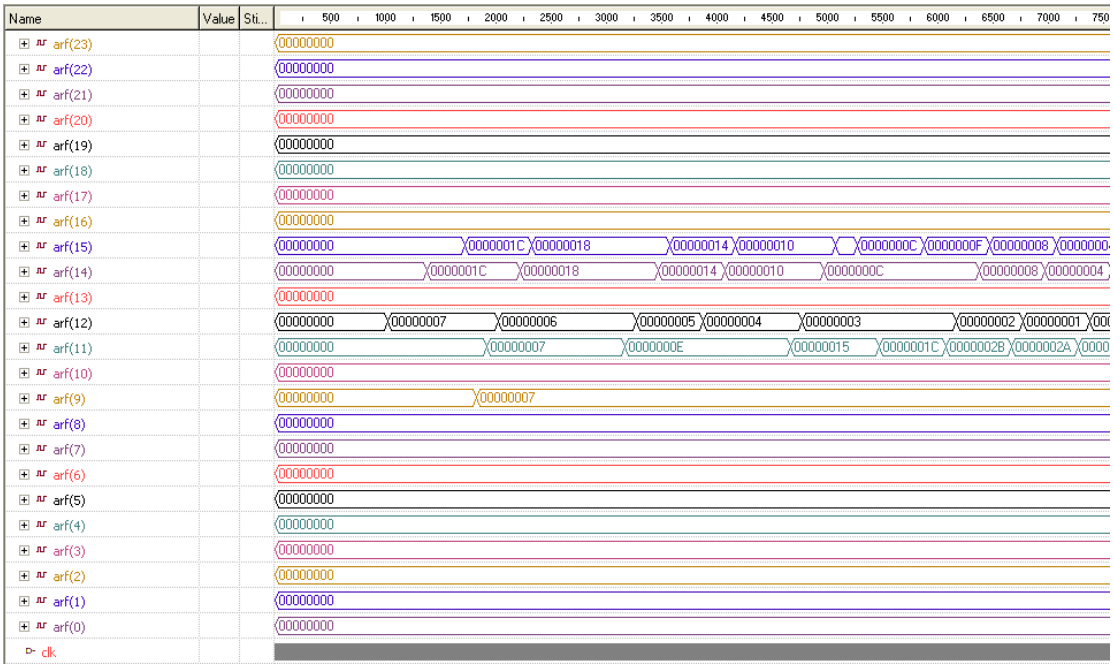
Ακολουθούν τα waveforms του top_level_unit, του RRF και του ARF από την προσομοίωση :



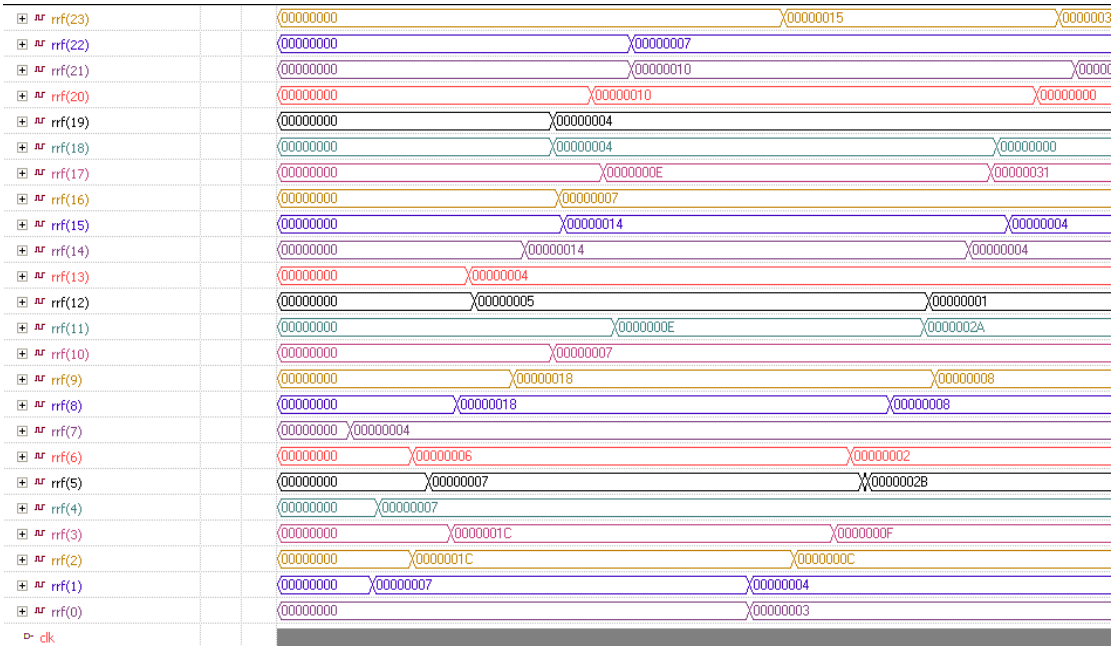
Εικόνα 6.1 ARF από 0-4000ns.



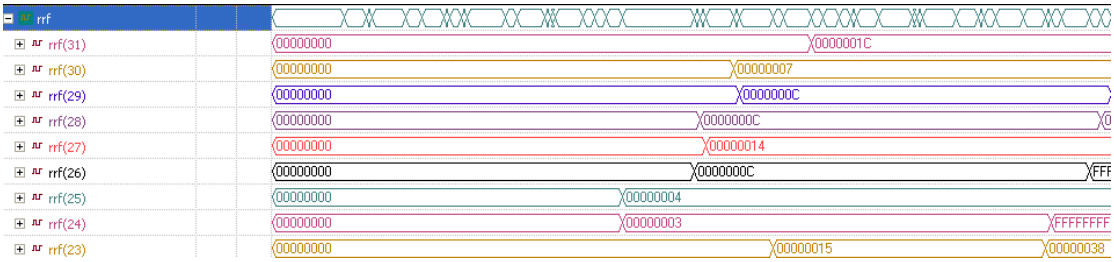
Εικόνα 6.2 ARF από 4000-8000ns.



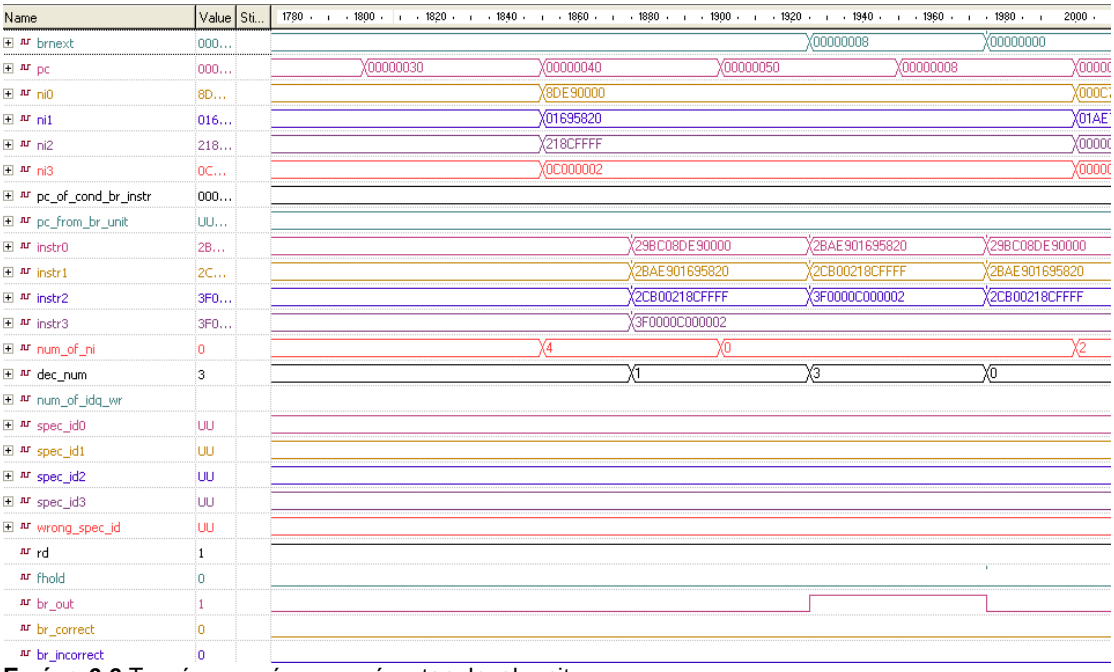
Εικόνα 6.3 ARF από 0-8000ns.



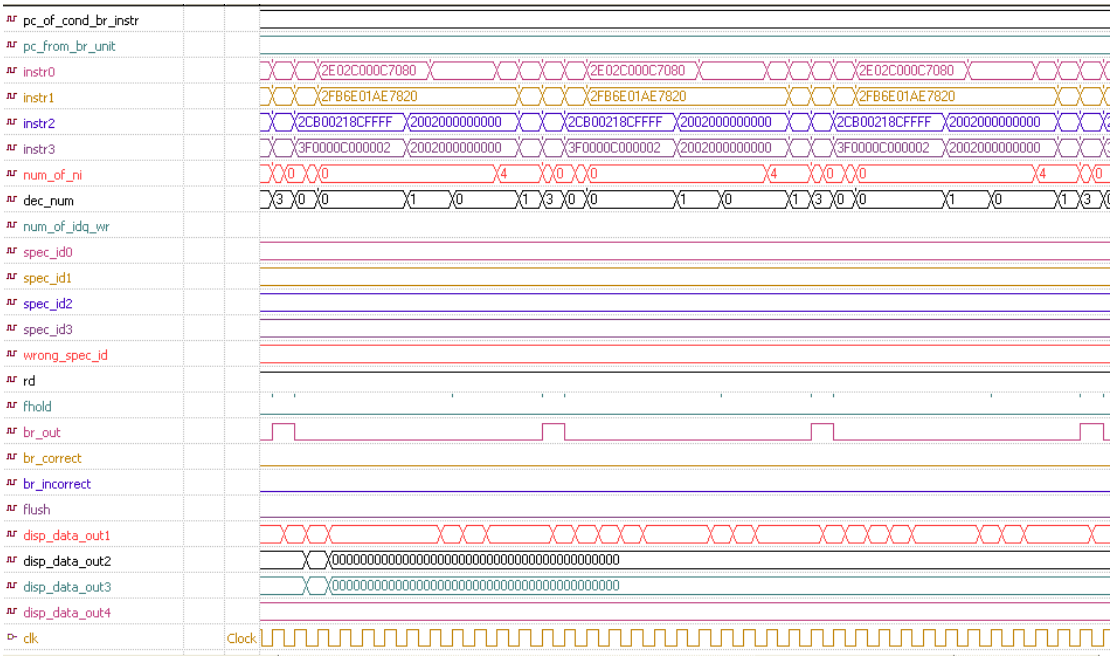
Εικόνα 6.4 RRF (καταχωρητές 0-23) από 0-8000ns.



Εικόνα 6.5 RRF (καταχωρητές 23-31) από 0-8000ns.



Εικόνα 6.6 Τυχαίο στιγμιότυπο από το top_level_unit.



Εικόνα 6.7 Τυχαίο στιγμιότυπο από το top_level_unit που δείχνει συνεχόμενα branches.

Παρατηρούμε όσων αφορά την απόδοση ότι ένα branch με 7 εντολές με εξαρτήσεις ανάμεσα στα ζευγάρια $1^{η}$-$2^{η}$, $3^{η}$-$4^{η}$, $4^{η}$-$5^{η}$ και $6^{η}$-$7^{η}$ (όλες οι εξαρτήσεις RAW) εκτελείται μέσα σε 12 κύκλους ρολογιού, γεγονός που δείχνει ότι η απόδοση του επεξεργαστή είναι εμφανώς βελτιωμένη σε σχέση με τη θεωρητική απόδοση ενός scalar pipelined επεξεργαστή που λόγω και τω εξαρτήσεων θα ήταν πολύ μεγαλύτερη.

7 ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Ψηφιακά Συστήματα VLSI, K.Z. Πεκμεστζή.
2. Modern Processor Design: Fundamentals of Superscalar Processors by John Shen, Carnegie Mellon University and Mikko Lipasti, University of Wisconsin-Madison.
1. Computer Organization and Design/Software Interface, Third Edition (The Morgan Kaufmann Series in Computer Architecture and Design) by David A. Patterson.
2. Computer Architecture: A Quantitative Approach, Third Edition (The Morgan Kaufmann Series in Computer Architecture and Design) by David A. Patterson.
3. Hardware Design Verification: Simulation and Formal Method-Based Approaches (Prentice Hall Modern Semiconductor Design Series) by William K. Lam.
4. VDHL: Programming By Example by Douglas L. Perry, McGraw Hill Book Company.