2

ΠΑΡΑΛΛΗΛΙΣΜΟΣ ΕΠΙΠΕΔΟΥ ΕΝΤΟΛΩΝ ΚΑΙ ΑΞΙΟΠΟΙΗΣΗ ΤΟΥ

«Ποιος είναι πρώτος;»
«Η Αμερική.»
«Ποιος είναι δεύτερος;»
«Κύριε, κανείς δεν είναι δεύτερος»

Διάλογος μεταξύ δύο θεατών του ιστιοπλοϊκού αγώνα που μετέπειτα ονομάστηκε «Τρόπαιο Αμερικής», ο οποίος διεξαγόταν κάθε λίγα χρόνια - αποτέλεσε έμπνευση για τον John Cocke, ο οποίος έδωσε στον ερευνητικό επεξεργαστή της IBM την ονομασία "America". Ο επεξεργαστής αυτός αποτέλεσε προπομπό της σειράς μικροεπεξεργαστών RS/6000 και του πρώτου υπερβαθμωτού μικροεπεξεργαστή.

2.1 Παραλληλισμός Επιπέδου Εντολών: Έννοιες και Προκλήσεις

Όλοι οι υπολογιστές, περίπου από το 1985 και έπειτα, χρησιμοποιούν τη διασωλήνωση για την επικάλυψη της εκτέλεσης των εντολών (instructions) και τη βελτίωση της απόδοσης. Η ενδεχόμενη αυτή επικάλυψη των εντολών ονομάζεται *παραλληλισμός επιπέδου εντολών* (instruction level parallelism, ILP), καθώς οι εντολές μπορούν να αξιολογούνται παράλληλα. Στο κεφάλαιο αυτό, καθώς και στο Παράρτημα Ζ, εξετάζουμε πληθώρα τεχνικών για την επέκταση των βασικών εννοιών της διασωλήνωσης, αυξάνοντας το εύρος του παραλληλισμού που αξιοποιούν οι εντολές.

Το κεφάλαιο αυτό είναι σε αρκετά υψηλότερο επίπεδο από την ύλη της βασικής διασωλήνωσης του Παραρτήματος Α. Αν ο αναγνώστης δεν είναι εξοικειωμένος με τις έννοιες του Παραρτήματος Α, καλό θα ήταν να ανατρέξει σε αυτό, προτού επιχειρήσει την ανάγνωση του κεφαλαίου αυτού.

Ξεκινούμε στο κεφάλαιο αυτό εξετάζοντας τους περιορισμούς που επιβάλλουν οι κίνδυνοι δεδομένων και ελέγχου και, στη συνέχεια, στρέφουμε το ενδιαφέρον μας στο ζήτημα της αύξησης της ικανότητας του μεταγλωττιστή και του επεξεργαστή όσον αφορά την αξιοποίηση του παραλληλισμού. Οι ενότητες αυτές εισάγουν μεγάλο αριθμό εννοιών, στις οποίες στηριζόμαστε καθ' όλη τη διάρκεια του κεφαλαίου αυτού και του επομένου. Παρόλο που ένα μέρος της βασικής ύλης του κεφαλαίου αυτού θα μπορούσε να γίνει κατανοητό χωρίς τη μελέτη *όλων των εννοιών των δύο πρώτων ενοτήτων*, το βασικό αυτό υλικό είναι πολύ σημαντικό για την κατανόηση των επόμενων ενοτήτων του κεφαλαίου, καθώς και του Κεφαλαίου 3.

Υπάρχουν δύο διαμετρικά αντίθετες προσεγγίσεις αξιοποίησης του ILP: η μία προσέγγιση βασίζεται στο υλικό (hardware) για τη δυναμική ανίχνευση και αξιοποίηση του παραλληλισμού, ενώ η δεύτερη προσέγγιση βασίζεται σε τεχνολογίες λογισμικού (software) για την εύρεση του παραλληλισμού, στατικά, κατά τη διάρκεια της μεταγλώττισης. Οι επεξεργαστές που χρησιμοποιούν τη δυναμική προσέγγιση που βασίζεται στο υλικό, συμπεριλαμβανομένων και αυτών που ανήκουν στη σειρά Pentium, είναι αυτοί που κυριαρχούν στην αγορά. Αυτοί που χρησιμοποιούν τη στατική προσέγγιση, συμπεριλαμβανομένου του Pentium Itanium, έχουν πιο περιορισμένες χρήσεις σε επιστημονικά περιβάλλοντα ή σε περιβάλλοντα που χρησιμοποιούνται συγκεκριμένες εφαρμογές.

Τα τελευταία έτη, πολλές από τις τεχνικές που έχουν αναπτυχθεί για τη μία προσέγγιση έχουν αξιοποιηθεί από σχεδιάσεις που βασίζονται ως επί το πλείστον στην άλλη προσέγγιση. Το κεφάλαιο αυτό εισάγει τις βασικές έννοιες και των δύο προσεγγίσεων. Το επόμενο κεφάλαιο εστιάζει στο σημαντικό ζήτημα των περιορισμών που υφίστανται κατά την αξιοποίηση του ILP.

Στην ενότητα αυτή, εξετάζουμε χαρακτηριστικά τόσο των προγραμμάτων όσο

και των επεξεργαστών τα οποία περιορίζουν το εύρος του παραλληλισμού που μπορεί να αξιοποιηθεί μεταξύ των εντολών, καθώς και το σημαντικό ζήτημα της αντιστοίχισης ανάμεσα στη δομή του προγράμματος και τη δομή του υλικού, ζήτημα κεφαλαιώδους σημασίας για να κατανοήσει κανείς αν η ιδιότητα κάποιου προγράμματος θα περιορίσει στην πραγματικότητα την απόδοση και υπό ποιες προϋποθέσεις.

Η τιμή του CPI (cycles per instruction, κύκλων ανά εντολή) για τον επεξεργαστή που υποστηρίζει τη διασωλήνωση ισούται με το άθροισμα του CPI βάσης και όλων των επιπλέον κύκλων που προσθέτουν τα διαστήματα αδράνειας (stalls):

$$\text{CPI διασωλήνωσης} = \text{Ιδανικό CPI διασωλήνωσης} + \text{Δομικά διαστ. αδράνειας} + \text{Διαστήματα αδράνειας κινδύνων δεδομένων} + \text{Διαστήματα αδράν. ελέγχου}$$

Το *ιδανικό CPI της διασωλήνωσης* (ideal pipeline CPI) αποτελεί μονάδα μέτρησης της μέγιστης απόδοσης που μπορεί να επιτευχθεί μέσω της υλοποίησης. Μειώνοντας καθέναν από τους όρους που βρίσκονται στο δεξί μέλος της εξίσωσης, μπορούμε να ελαχιστοποιήσουμε το συνολικό CPI της διασωλήνωσης ή, εναλλακτικά, να αυξήσουμε το IPC (instructions per clock, εντολές ανά κύκλο). Η παραπάνω εξίσωση μάς επιτρέπει να αξιολογήσουμε τις διάφορες τεχνικές με βάση το συστατικό στοιχείο του συνολικού CPI στο οποίο επέρχεται μείωση. Το Σχήμα 2.1 παρουσιάζει τις τεχνικές που εξετάζουμε στο κεφάλαιο αυτό και στο Παράρτημα Ζ, καθώς και τα ζητήματα που καλύπτονται στο εισαγωγικό υλικό του Παραρτήματος Α. Στο κεφάλαιο αυτό θα διαπιστώσουμε ότι οι τεχνικές που παραθέτουμε για τη μείωση του ιδανικού CPI της διασωλήνωσης μπορεί να αυξήσουν τη σπουδαιότητα που έχει το ζήτημα της αντιμετώπισης των κινδύνων.

Τι Είναι ο Παραλληλισμός Επιπέδου Εντολών;

Όλες οι τεχνικές του κεφαλαίου αυτού αξιοποιούν τον παραλληλισμό μεταξύ των εντολών. Το εύρος του διαθέσιμου παραλληλισμού εντός ενός *βασικού block* (basic block) - μιας ευθύγραμμης ακολουθίας κώδικα που δεν μπορεί να περιλαμβάνει εντολή η οποία αποτελεί προορισμό διακλάδωσης παρά μόνο στην είσοδό της ούτε εντολή η οποία αποτελεί αφετηρία διακλάδωσης παρά μόνο στην έξοδό της - είναι σχετικά μικρό. Στα συνήθη προγράμματα της MIPS, η μέση συχνότητα δυναμικής διακλάδωσης συχνά κυμαίνεται ανάμεσα στο 15% και το 25%, κάτι που συνεπάγεται ότι τρεις έως έξι εντολές εκτελούνται μεταξύ ενός ζεύγους εντολών διακλάδωσης. Καθώς οι εντολές αυτές είναι πιθανό να εξαρτώνται η μία από την άλλη, το μέγεθος της επικάλυψης που μπορεί να επιτευχθεί εντός ενός βασικού block είναι πιθανό να είναι μικρότερο από το μέσο μέγεθος του βασικού block. Προκειμένου να έχουμε ουσιαστική βελτίωση της απόδοσης, πρέπει να αξιοποιήσουμε τον ILP μεταξύ πολλαπλών βασικών blocks.

Ο πιο απλός και συνήθης τρόπος αύξησης του ILP αφορά την αξιοποίηση του παραλληλισμού μεταξύ των επαναλήψεων ενός βρόχου. Αυτός ο τύπος παραλ-

Τεχνική	Αποτέλεσμα	Ενότητα
Προώθηση και παράκαμψη	Ενδεχόμενα διαστήματα αδράνειας κινδύνων δεδομένων	A.2
Καθυστερημένες διακλαδώσεις και απλή χρονοδρομολόγηση διακλάδωσης	Διαστήματα αδράνειας κινδύνων ελέγχου	A.2
Βασική δυναμική χρονοδρομολόγηση (πίνακας παρακολούθησης)	Διαστήματα αδράνειας κινδύνου δεδομένων εξαιτίας πραγματικών εξαρτήσεων	A.7
Δυναμική χρονοδρομολόγηση με μετονομασία	Διαστήματα αδράνειας κινδύνων δεδομένων εξαιτίας αντεξαρτήσεων και εξαρτήσεων εξόδου	2.4
Πρόβλεψη διακλάδωσης	Διαστήματα αδράνειας ελέγχου	2.3
Έκδοση πολλαπλών εντολών ανά κύκλο	Ιδανικό CPI	2.7, 2.8
Εικασία υλικού	Διαστήματα αδράνειας κινδύνων δεδομένων και ελέγχου	2.6
Δυναμική αποσαφήνιση μνήμης	Διαστήματα αδράνειας κινδύνων δεδομένων με τη μνήμη	2.4, 2.6
Ξεδίπλωμα βρόχου	Διαστήματα αδράνειας κινδύνων ελέγχου	2.2
Βασική χρονοδρομολόγηση διασωλήνωσης μέσω του μεταγλωττιστή	Διαστήματα αδράνειας κινδύνων δεδομένων	A.2, 2.2
Ανάλυση εξαρτήσεων μέσω του μεταγλωττιστή, διασωλήνωση λογισμικού, χρονοδρομολόγηση ίχνους	Ιδανικό CPI, διαστήματα αδράνειας κινδύνων δεδομένων	Z.2, Z.3
Υποστήριξη υλικού για την εικασία του μεταγλωττιστή	Ιδανικό CPI, διαστήματα αδράνειας κινδύνων δεδομένων, διαστήματα αδράνειας κινδύνων διακλάδωσης	Z.4, Z.5

Σχήμα 2.1 Οι βασικές τεχνικές που εξετάζονται στο Παράρτημα Α, στο Κεφάλαιο 2 ή στο Παράρτημα Ζ παρουσιάζονται σε συνδυασμό με τον όρο της εξίσωσης του CPI ο οποίος επηρεάζεται από κάθε τεχνική.

ληλισμού συχνά ονομάζεται *παραλληλισμός επιπέδου βρόχου* (loop-level parallelism). Παρακάτω ακολουθεί ένα απλό παράδειγμα βρόχου, ο οποίος προσθέτει δύο πίνακες των 1000 στοιχείων, διαδικασία που είναι πλήρως παράλληλη:

```
f    (i=1; i<=1000; i=i+1)
         x[i] = x[i] + y[i];
```

Κάθε επανάληψη του βρόχου μπορεί να επικαλύπτεται με οποιαδήποτε άλλη επανάληψη, παρόλο που εντός της επανάληψης είτε δεν υπάρχει καμία είτε υπάρχουν λίγες ευκαιρίες επικάλυψης.

Υπάρχει πληθώρα τεχνικών τις οποίες θα εξετάσουμε για τη μετατροπή αυτού του είδους του παραλληλισμού επιπέδου βρόχου σε παραλληλισμό επιπέδου εντολών. Βασικά, αυτού του είδους οι τεχνικές λειτουργούν ξεδιπλώνοντας το βρόχο είτε στατικά με τη βοήθεια του μεταγλωττιστή (όπως στην επόμενη ενότητα) είτε δυναμικά με τη συνδρομή του υλικού (όπως στις Ενότητες 2.5 και 2.6).

Σημαντική εναλλακτική μέθοδο για την αξιοποίηση του παραλληλισμού επιπέδου βρόχου αποτελεί η χρήση διανυσματικών εντολών (vector instructions)

(δείτε το Παράρτημα ΣΤ). Οι διανυσματικές εντολές αξιοποιούν τον παραλληλισμό επιπέδου δεδομένων, μέσω παράλληλης εκτέλεσης λειτουργιών χρησιμοποιώντας δεδομένα. Για παράδειγμα, σε μερικούς διανυσματικούς επεξεργαστές, η παραπάνω ακολουθία κώδικα θα μπορούσε να εκτελεστεί με τέσσερις εντολές: δύο εντολές για τη φόρτωση των διανυσμάτων x και y από τη μνήμη, μία εντολή για την πρόσθεση των δύο διανυσμάτων και μία εντολή για την αποθήκευση του αποτελέσματος στο διάνυσμα αποτελεσμάτων. Βεβαίως, οι εντολές αυτές θα χρησιμοποιούσαν διασωλήνωση και θα είχαν υψηλούς λανθάνοντες χρόνους, ωστόσο αυτοί οι λανθάνοντες χρόνοι μπορεί να επικαλύπτονται.

Παρόλο που η ανάπτυξη των ιδεών χρήσης διανυσμάτων προηγήθηκε της εφεύρεσης πολλών από τις τεχνικές αξιοποίησης του ILP, οι επεξεργαστές που αξιοποιούν τον ILP έχουν σχεδόν αντικαταστήσει πλήρως τους διανυσματικούς επεξεργαστές στην αγορά των επεξεργαστών γενικού σκοπού. Ωστόσο, τα σύνολα των διανυσματικών εντολών έχουν σημειώσει μεγάλη επιτυχία, τουλάχιστον σε ό,τι αφορά τη χρήση τους στα γραφικά, στην επεξεργασία ψηφιακών σημάτων και σε εφαρμογές πολυμέσων.

Εξαρτήσεις Δεδομένων και Κίνδυνοι

Ο προσδιορισμός του τρόπου με τον οποίο η μία εντολή εξαρτάται από την άλλη είναι σημαντικός για τον προσδιορισμό του εύρους του παραλληλισμού που υπάρχει σε κάποιο πρόγραμμα και του βαθμού στον οποίο μπορεί να αξιοποιηθεί ο παραλληλισμός αυτός. Συγκεκριμένα, για την αξιοποίηση του παραλληλισμού επιπέδου εντολών πρέπει να προσδιορίσουμε ποιες είναι οι εντολές που μπορούν να εκτελούνται παράλληλα. Αν δύο εντολές είναι *παράλληλες* (parallel), μπορούν να εκτελούνται ταυτόχρονα σε διασωλήνωση αυθαίρετου βάθους χωρίς να προκαλούν διαστήματα αδράνειας, με την προϋπόθεση ότι η διασωλήνωση διαθέτει επαρκείς πόρους (και ως εκ τούτου δεν υφίστανται δομικοί κίνδυνοι). Αν δύο εντολές είναι εξαρτώμενες, τότε δεν είναι παράλληλες και πρέπει να εκτελεστούν ακολουθώντας τη σειρά τους, παρόλο που συχνά μπορεί εν μέρει να επικαλύπτονται. Και στις δύο περιπτώσεις, το πιο βασικό στοιχείο είναι να προσδιοριστεί αν κάποια εντολή εξαρτάται από κάποια άλλη.

Εξαρτήσεις Δεδομένων

Υπάρχουν τρεις διαφορετικοί τύποι εξαρτήσεων: οι *εξαρτήσεις δεδομένων* (data dependencies) (επίσης ονομάζονται πραγματικές εξαρτήσεις δεδομένων[1]), οι *εξαρτήσεις ονομάτων* (name dependencies) και οι *εξαρτήσεις ελέγχου* (control dependencies). Η εντολή j έχει *εξάρτηση δεδομένων* από την εντολή i, αν ισχύει οποιοδήποτε από τα παρακάτω:

[1] Σ.τ.Μ. Ο αντίστοιχος αγγλικός όρος είναι "true data dependencies"

- αν η εντολή i παράγει αποτέλεσμα το οποίο μπορεί να χρησιμοποιηθεί από την εντολή j ή
- αν η εντολή j έχει εξάρτηση δεδομένων από την εντολή k και η εντολή k έχει εξάρτηση δεδομένων από την εντολή i.

Η δεύτερη συνθήκη απλά υποδηλώνει ότι μία εντολή εξαρτάται από κάποια άλλη, αν υφίσταται αλυσίδα εξαρτήσεων του πρώτου τύπου ανάμεσα σε δύο εντολές. Αυτή η αλυσίδα εξαρτήσεων μπορεί να έχει μήκος ίσο με το μήκος ολόκληρου του προγράμματος. Αξίζει να σημειωθεί ότι οι εξαρτήσεις εντός μίας μόνο εντολής (όπως η `ADDD R1, R1, R1`) δεν θεωρούνται εξαρτήσεις.

Για παράδειγμα, ας εξετάσουμε την παρακάτω ακολουθία κώδικα που αυξάνει τις τιμές ενός διανύσματος το οποίο βρίσκεται στη μνήμη (ξεκινά από τη θέση `0(R1)`, ενώ το τελευταίο του στοιχείο βρίσκεται στη θέση `8(R2)`) προσθέτοντας σε αυτές την τιμή ενός βαθμωτού πεδίου (scalar) η οποία βρίσκεται στον καταχωρητή `F2` (για λόγους απλότητας, σε όλο το κεφάλαιο αυτό, τα παραδείγματά μας δεν λαμβάνουν υπόψη την επίδραση των καθυστερημένων διακλαδώσεων).

```
Loop: L.D     F0,0(R1)     ;F0 = στοιχείο πίνακα
      ADD.D   F4,F0,F2     ;πρόσθεση του βαθμωτού πεδίου που βρίσκεται στον F2
      S.D     F4,0(R1)     ;αποθήκευση αποτελέσματος
      DADDUI  R1,R1,#-8    ;μείωση δείκτη κατά 8 bytes
      BNE     R1,R2,Loop   ;διακλάδωση αν R1!=R2
```

Οι εξαρτήσεις δεδομένων σε αυτήν την ακολουθία συμπεριλαμβάνουν τόσο δεδομένα κινητής υποδιαστολής:

```
Loop: L.D     F0,0(R1)     ;F0 = στοιχείο πίνακα
      ADD.D   F4,F0,F2     ;πρόσθεση του βαθμωτού πεδίου που βρίσκεται στον F2
      S.D     F4,0(R1)     ;αποθήκευση αποτελέσματος
```

όσο και δεδομένα ακεραίων:

```
      DADDUI  R1,R1,#-8    ;μείωση δείκτη κατά 8 bytes
      BNE     R1,R2,Loop   ;διακλάδωση R1!=R2
```

Και στις δύο παραπάνω ακολουθίες εξαρτήσεων, όπως καταδεικνύεται από τα σχετικά βέλη, κάθε εντολή εξαρτάται από την αμέσως προηγούμενη. Τα βέλη στο παράδειγμα αυτό, αλλά και στα επόμενα, υποδηλώνουν τη σειρά που πρέπει να διατηρηθεί, έτσι ώστε να υπάρχει ορθή εκτέλεση. Τα βέλη έχουν ως αφετηρία την εντολή που πρέπει να προηγείται αυτής προς την οποία δείχνει η αιχμή κάθε βέλους.

Αν δύο εντολές έχουν μεταξύ τους εξάρτηση δεδομένων, δεν μπορούν να εκτελεστούν ταυτόχρονα ή να επικαλυφθούν πλήρως. Η εξάρτηση υποδηλώνει ότι υπάρχει μια αλυσίδα που περιλαμβάνει έναν ή περισσότερους κινδύνους δεδομένων μεταξύ των δύο εντολών (δείτε το Παράρτημα Α για συνοπτική περιγραφή

των κινδύνων δεδομένων, για τους οποίους θα παραθέσουμε ακριβή ορισμό στις επόμενες σελίδες). Η ταυτόχρονη εκτέλεση των εντολών οδηγεί τους επεξεργαστές που διαθέτουν ενδοασφάλεια (interlock) διασωλήνωσης (και βάθος διασωλήνωσης μεγαλύτερο από την απόσταση, σε κύκλους, των δύο εντολών) στην ανίχνευση του κινδύνου και την εισαγωγή διαστημάτων αδράνειας, κάτι που ως εκ τούτου μειώνει ή εξαλείφει την επικάλυψη. Στους επεξεργαστές που δεν διαθέτουν ενδοασφάλειες και βασίζονται στη χρονοδρομολόγηση του μεταγλωττιστή (compiler scheduling), ο μεταγλωττιστής δεν είναι σε θέση να χρονοδρομολογείται εξαρτώμενες εντολές, με τέτοιο τρόπο ώστε να υπάρχει πλήρης επικάλυψη, καθώς το πρόγραμμα δεν θα εκτελείται ορθά. Η παρουσία εξάρτησης δεδομένων στην ακολουθία εντολών αντικατοπτρίζει την παρουσία εξάρτησης δεδομένων στον πηγαίο κώδικα από τον οποίο έχει παραχθεί αυτή η ακολουθία εντολών. Πρέπει να διατηρηθεί το αποτέλεσμα της αρχικής εξάρτησης δεδομένων.

Οι εξαρτήσεις αποτελούν ιδιότητα των *προγραμμάτων* (programs). Το αν κάποια δεδομένη εξάρτηση έχει ως αποτέλεσμα την ανίχνευση ενός πραγματικού κινδύνου, καθώς και το αν ο κίνδυνος αυτός στην πραγματικότητα προκαλεί την εισαγωγή διαστήματος αδράνειας είναι στοιχεία που αποτελούν ιδιότητες της *οργάνωσης της διασωλήνωσης* (pipeline organization). Η διαφορά αυτή είναι σημαντική για την κατανόηση του τρόπου αξιοποίησης του παραλληλισμού επιπέδου εντολών.

Οι εξαρτήσεις δεδομένων υποδηλώνουν τρία πράγματα: (1) την πιθανότητα εμφάνισης κινδύνου, (2) τη σειρά με την οποία πρέπει να υπολογίζονται τα αποτελέσματα και (3) το ανώτερο όριο του βαθμού στον οποίο μπορεί να αξιοποιηθεί ο παραλληλισμός. Αυτού του είδους οι περιορισμοί εξετάζονται στο Κεφάλαιο 3.

Αφού οι εξαρτήσεις δεδομένων μπορούν να περιορίσουν το εύρος του παραλληλισμού επιπέδου εντολών που μπορούμε να αξιοποιήσουμε, βασικό αντικείμενο ενδιαφέροντος του κεφαλαίου αυτού είναι η υπέρβαση των περιορισμών αυτών. Οι εξαρτήσεις μπορούν να υπερκεραστούν με δύο διαφορετικούς τρόπους: με τη διατήρηση της εξάρτησης και την αποφυγή του κινδύνου, καθώς και με την εξάλειψη της εξάρτησης αλλάζοντας τη μορφή του κώδικα. Η χρονοδρομολόγηση του κώδικα αποτελεί την κύρια μέθοδο που χρησιμοποιείται για την αποφυγή των κινδύνων χωρίς να τροποποιείται η εξάρτηση και αυτού του είδους η χρονοδρομολόγηση μπορεί να πραγματοποιηθεί τόσο από το μεταγλωττιστή όσο και από τον επεξεργαστή.

Μια τιμή δεδομένων μπορεί να μετακινείται μεταξύ εντολών είτε διαμέσου των καταχωρητών είτε μέσω των θέσεων μνήμης. Όταν αυτή η μετακίνηση δεδομένων πραγματοποιείται σε κάποιον καταχωρητή, η ανίχνευση της εξάρτησης είναι σαφής, καθώς τα ονόματα των καταχωρητών στις εντολές είναι σταθερά, παρόλο που μερικές φορές η διαδικασία αυτή περιπλέκεται όταν μεσολαβούν διακλαδώσεις και όταν τα ζητήματα ορθότητας επιβάλλουν στο μεταγλωττιστή ή στον επεξεργαστή να λειτουργεί συντηρητικά.

Οι εξαρτήσεις που υφίστανται μεταξύ θέσεων της μνήμης είναι πιο δύσκολο

να ανιχνευθούν, καθώς δύο εντολές μπορεί να αναφέρονται στην ίδια θέση άλλα να έχουν διαφορετική μορφή: Για παράδειγμα, η `100(R4)` και η `20(R6)` μπορεί να αποτελούν την ίδια διεύθυνση μνήμης. Επιπλέον, η ενεργός διεύθυνση (effective address) κάποιας εντολής φόρτωσης ή αποθήκευσης μπορεί να διαφοροποιείται από τη μία εκτέλεση της εντολής στην επόμενη (έτσι ώστε οι `20(R4)` και `20(R4)` να αποτελούν διαφορετικές διευθύνσεις), κάτι που περιπλέκει ακόμα περισσότερο την ανίχνευση της εξάρτησης.

Στο κεφάλαιο αυτό, εξετάζουμε το υλικό για την ανίχνευση εξαρτήσεων δεδομένων, οι οποίες περιλαμβάνουν θέσεις μνήμης, ωστόσο θα διαπιστώσουμε ότι οι τεχνικές αυτές έχουν, επίσης, περιορισμούς. Οι τεχνικές μεταγλωττιστή για την ανίχνευση τέτοιων εξαρτήσεων είναι κρίσιμες για το ξετύλιγμα του παραλληλισμού επιπέδου βρόχου, όπως θα διαπιστώσουμε στο Παράρτημα Z.

Εξαρτήσεις Ονόματος

Ο δεύτερος τύπος εξαρτήσεων είναι οι *εξαρτήσεις ονόματος* (name dependencies). Οι εξαρτήσεις ονόματος προκύπτουν όταν δύο εντολές χρησιμοποιούν τον ίδιο καταχωρητή ή την ίδια θέση μνήμης, δηλαδή το ίδιο *όνομα* (name), χωρίς ωστόσο να υπάρχει κάποια ροή δεδομένων ανάμεσα στις εντολές που έχουν συσχετισθεί με το όνομα αυτό. Υπάρχουν δύο είδη εξαρτήσεων ονόματος ανάμεσα στην εντολή i και την εντολή j, όπου η i *προηγείται* της j στη σειρά του προγράμματος:

1. *Αντεξάρτηση* (antidependence) μεταξύ της εντολής i και της εντολής j προκύπτει, όταν η εντολή j εγγράφει κάποιον καταχωρητή ή κάποια θέση μνήμης την οποία διαβάζει η εντολή i. Πρέπει να διατηρείται η αρχική σειρά των εντολών, έτσι ώστε να διασφαλίζεται ότι η i διαβάζει την ορθή τιμή. Στο παράδειγμα της σελίδας 94, υπάρχει αντεξάρτηση ανάμεσα στην `S.D` και την `DADDIU` στον καταχωρητή R1.

2. *Εξάρτηση εξόδου* (output dependence) προκύπτει όταν η εντολή i και η εντολή j εγγράφουν τον ίδιο καταχωρητή ή την ίδια θέση μνήμης. Πρέπει να διατηρείται η σειρά των εντολών, έτσι ώστε να διασφαλίζεται ότι η τιμή που εγγράφεται τελευταία αντιστοιχεί στην εντολή j.

Τόσο οι αντεξαρτήσεις όσο και οι εξαρτήσεις εξόδου αποτελούν εξαρτήσεις ονόματος και όχι πραγματικές εξαρτήσεις δεδομένων, καθώς δεν υφίσταται καμία τιμή η οποία μεταφέρεται από τη μία εντολή στην άλλη. Αφού οι εξαρτήσεις ονόματος δεν αποτελούν πραγματικές εξαρτήσεις, οι εντολές που εμπλέκονται στις εξαρτήσεις ονόματος μπορούν να εκτελούνται ταυτόχρονα ή να αναδιατάσσονται, με την προϋπόθεση ότι θα αλλάξει το όνομα (ο αριθμός του καταχωρητή ή η θέση μνήμης) που χρησιμοποιείται στις εντολές, έτσι ώστε αυτές να μην έρχονται σε σύγκρουση.

Η μετονομασία αυτή μπορεί εύκολα να πραγματοποιηθεί για τα ορίσματα καταχωρητών, διαδικασία που είναι γνωστή ως *μετονομασία καταχωρητών* (register renaming). Η μετονομασία καταχωρητών μπορεί να πραγματοποιηθεί είτε στατικά με τη βοήθεια του μεταγλωττιστή είτε δυναμικά με τη συνδρομή του υλικού. Πριν προχωρήσουμε στην περιγραφή των εξαρτήσεων που προκύπτουν από τις διακλαδώσεις, ας εξετάσουμε τη σχέση ανάμεσα στις εξαρτήσεις και τους κινδύνους δεδομένων της διασωλήνωσης.

Κίνδυνοι Δεδομένων

Κίνδυνος προκύπτει οποτεδήποτε υφίσταται εξάρτηση ανάμεσα σε εντολές, οι οποίες συγχρόνως βρίσκονται σε τόσο κοντινή απόσταση ώστε η επικάλυψη κατά τη διάρκεια της εκτέλεσης να οδηγεί στην τροποποίηση της σειράς της προσπέλασης των ορισμάτων που εμπλέκονται στην εξάρτηση. Εξαιτίας της εξάρτησης, πρέπει να διατηρούμε αναλλοίωτη τη *σειρά προγράμματος* (program order), δηλαδή, τη σειρά με την οποία θα εκτελούνταν οι εντολές, αν εκτελούνταν μία μία ακολουθιακά, όπως θα υποδείκνυε ο αρχικός πηγαίος κώδικας του προγράμματος. Στόχος τόσο των τεχνικών υλικού όσο και των τεχνικών λογισμικού είναι η αξιοποίηση του παραλληλισμού, διατηρώντας τη σειρά του προγράμματος *μόνο στις περιπτώσεις όπου επηρεάζεται το αποτέλεσμα του προγράμματος*. Η ανίχνευση και η αποφυγή των κινδύνων διασφαλίζει τη διατήρηση της απαραίτητης σειράς του προγράμματος.

Οι κίνδυνοι δεδομένων, που περιγράφονται συνοπτικά στο Παράρτημα Α, μπορούν να ομαδοποιηθούν σε τρεις κατηγορίες, ανάλογα με τη σειρά που λαμβάνουν χώρα οι προσπελάσεις ανάγνωσης και εγγραφής των εντολών. Οι κίνδυνοι λαμβάνουν την ονομασία τους με βάση τη σειρά των εντολών του προγράμματος, η οποία πρέπει να διατηρηθεί από τη διασωλήνωση. Ας εξετάσουμε δύο εντολές, τις i και j, όπου η i προηγείται της j στη σειρά προγράμματος. Οι πιθανοί κίνδυνοι δεδομένων είναι:

- RAW[2] (*ανάγνωση μετά από εγγραφή*): Η εντολή j επιχειρεί να διαβάσει το όρισμα αφετηρίας προτού αυτό να εγγραφεί από την i και ως εκ τούτου λαμβάνει την *παλαιά* τιμή. Αυτός ο κίνδυνος αποτελεί την πιο συνηθισμένη περίπτωση και αντιστοιχεί στην πραγματική εξάρτηση δεδομένων. Η σειρά του προγράμματος πρέπει να διατηρείται, έτσι ώστε να διασφαλίζεται ότι η εντολή j λαμβάνει την τιμή από την i.

- WAW[3] (*εγγραφή μετά από εγγραφή*): Η εντολή j επιχειρεί να εγγράψει κάποιο όρισμα προτού αυτό να εγγραφεί από την εντολή i. Το τελικό αποτέλεσμα είναι το γεγονός ότι οι λειτουργίες εγγραφής πραγματοποιούνται

[2] Σ.τ.Μ. Ακρωνύμιο του αγγλικού όρου "Read After Write".
[3] Σ.τ.Μ. Ακρωνύμιο του αγγλικού όρου "Write After Write".

με λάθος σειρά, κάτι που σημαίνει ότι το όρισμα προορισμού εν τέλει διαθέτει την τιμή που έχει εγγραφεί από την i και όχι από την j. Ο κίνδυνος αυτός αντιστοιχεί στην εξάρτηση εξόδου. Οι κίνδυνοι WAW παρουσιάζονται μόνο στις διασωληνώσεις που πραγματοποιούνται ενέργειες εγγραφής σε περισσότερα από ένα στάδια της σωλήνωσης (pipe stages) ή σε διασωληνώσεις που επιτρέπουν στις εντολές να προχωρούν ακόμα και αν κάποια προηγούμενη εντολή έχει τεθεί σε αδράνεια.

- WAR[4] (*εγγραφή μετά από ανάγνωση*): Η εντολή j επιχειρεί να εγγράψει το όρισμα προορισμού, προτού αυτό διαβαστεί από την i και ως εκ τούτου η i λαμβάνει λανθασμένα τη *νέα* τιμή. Ο κίνδυνος αυτός προκύπτει από αντεξάρτηση. Οι κίνδυνοι WAR δεν μπορούν να εμφανιστούν στις περισσότερες διασωληνώσεις στατικής έκδοσης (static issue) - ακόμα και σε διασωληνώσεις μεγαλύτερου βάθους ή διασωληνώσεις κινητής υποδιαστολής - επειδή οι ενέργειες ανάγνωσης πραγματοποιούνται νωρίς (στο στάδιο ID), ενώ οι ενέργειες εγγραφής αργότερα (στο στάδιο WB) (δείτε το Παράρτημα Α για περισσότερες πληροφορίες). Οι κίνδυνοι WAR προκύπτουν είτε όταν υπάρχουν μερικές εντολές που εγγράφουν τα αποτελέσματα στα πρώτα στάδια της διασωλήνωσης εντολών *και* κάποιες άλλες που διαβάζουν τα ορίσματα αφετηρίας στα τελευταία στάδια της διασωλήνωσης είτε όταν οι εντολές αναδιατάσσονται, όπως θα δούμε στο κεφάλαιο αυτό.

Αξίζει να σημειωθεί ότι η περίπτωση RAR[5] (*ανάγνωσης μετά από ανάγνωση*) δεν αποτελεί κίνδυνο.

Εξαρτήσεις Ελέγχου

Ο τελευταίος τύπος εξαρτήσεων είναι οι *εξαρτήσεις ελέγχου* (control dependencies). Η εξάρτηση ελέγχου προσδιορίζει τη σειρά μιας εντολής i, σε σχέση με μια εντολή διακλάδωσης, έτσι ώστε η εντολή i αφενός να εκτελείται με ορθή σειρά προγράμματος και αφετέρου μόνο όταν αυτό πρέπει να συμβαίνει. Κάθε εντολή, εκτός από αυτές που ανήκουν στο πρώτο βασικό block του προγράμματος, έχει εξάρτηση ελέγχου από κάποιο σύνολο διακλαδώσεων και, σε γενικές γραμμές, αυτές οι εξαρτήσεις ελέγχου πρέπει να διατηρούνται, έτσι ώστε να διατηρείται η σειρά του προγράμματος. Ένα από τα πιο απλά παραδείγματα εξάρτησης ελέγχου είναι η εξάρτηση των προτάσεων (statements) που βρίσκονται στο τμήμα "then" της πρότασης if της διακλάδωσης. Για παράδειγμα, στο παρακάτω τμήμα κώδικα

[4] Σ.τ.Μ. Ακρωνύμιο του αγγλικού όρου "Write After Read".
[5] Σ.τ.Μ. Ακρωνύμιο του αγγλικού όρου "Read After Read".

```
if p1 {
      S1;
};
if p2 {
      S2;
}
```

η S1 έχει εξάρτηση ελέγχου από την p1 και η S2 έχει εξάρτηση ελέγχου από την p2, αλλά όχι από την p1.

Σε γενικές γραμμές, υπάρχουν δύο περιορισμοί που επιβάλλουν οι εξαρτήσεις ελέγχου:

1. Οι εντολές που έχουν εξάρτηση ελέγχου από κάποια διακλάδωση δεν μπορούν να μεταφερθούν *πριν* τη διακλάδωση, έχοντας ως στόχο να *μην ελέγχεται πλέον* η εκτέλεσή τους από τη διακλάδωση. Για παράδειγμα, δεν μπορούμε να πάρουμε μια εντολή από το τμήμα "then" μιας πρότασης if και να τη μεταφέρουμε πριν την πρόταση if.

2. Οι εντολές που δεν έχουν εξάρτηση ελέγχου από κάποια διακλάδωση δεν μπορούν να μεταφερθούν *μετά* τη διακλάδωση, έχοντας ως στόχο *να ελέγχεται* η εκτέλεσή τους από τη διακλάδωση. Για παράδειγμα, δεν μπορούμε να λάβουμε μια πρόταση που προηγείται της πρότασης if και να τη μεταφέρουμε στο τμήμα "then".

Όταν οι επεξεργαστές διατηρούν αυστηρά τη σειρά του προγράμματος, διασφαλίζουν ότι, συγχρόνως, διατηρούνται και οι εξαρτήσεις ελέγχου. Παρ' όλα αυτά, ενδέχεται, *σε περίπτωση* που μπορεί να επιτευχθεί αυτό χωρίς να επηρεάζεται η ορθότητα του προγράμματος, να θέλουμε να εκτελέσουμε εντολές οι οποίες δεν θα έπρεπε να εκτελεστούν και ως εκ τούτου να παραβιάσουμε τις εξαρτήσεις ελέγχου. Η εξάρτηση ελέγχου δεν αποτελεί τη σημαντική ιδιότητα που πρέπει να διατηρείται. Αντιθέτως, οι δύο ιδιότητες που είναι σημαντικές για την ορθότητα του προγράμματος - και συνήθως διατηρούνται διασφαλίζοντας τόσο τις εξαρτήσεις δεδομένων όσο και τις εξαρτήσεις ελέγχου - είναι η *συμπεριφορά των εξαιρέσεων* (exception behavior) και η *ροή δεδομένων* (data flow).

Η διατήρηση της συμπεριφοράς των εξαιρέσεων συνεπάγεται ότι οι όποιες αλλαγές ενδέχεται να πραγματοποιηθούν στη διάταξη της εκτέλεσης των εντολών δεν πρέπει να διαφοροποιούν τον τρόπο με τον οποίο προκύπτουν οι εξαιρέσεις στα προγράμματα. Συνήθως, εφαρμόζεται μια πιο ελαστική μορφή του παραπάνω κανόνα σύμφωνα με την οποία η αναδιάταξη της εκτέλεσης των εντολών δεν πρέπει να προκαλεί καμία νέα εξαίρεση στο πρόγραμμα. Το παρακάτω απλό παράδειγμα καταδεικνύει τον τρόπο με τον οποίο η διατήρηση των εξαρτήσεων ελέγχου και δεδομένων μπορεί να παρεμποδίσει τέτοιου είδους καταστάσεις. Ας εξετάσουμε την παρακάτω ακολουθία κώδικα:

```
        DADDU   R2,R3,R4
        BEQZ    R2,L1
        LW      R1,0(R2)
L1:
```

Στην περίπτωση αυτή είναι εύκολο να διαπιστώσει κανείς ότι αν δεν διατηρήσουμε την εξάρτηση δεδομένων που περιλαμβάνει τον καταχωρητή R2, τότε μπορούμε να τροποποιήσουμε το αποτέλεσμα του προγράμματος. Λιγότερο εμφανές είναι το γεγονός ότι αν αγνοήσουμε την εξάρτηση ελέγχου και μεταφέρουμε την εντολή φόρτωσης πριν τη διακλάδωση, η εντολή φόρτωσης μπορεί να προκαλέσει εξαίρεση προστασίας της μνήμης. Αξίζει να σημειωθεί ότι *καμία εξάρτηση δεδομένων* δεν παρεμποδίζει την εναλλαγή των εντολών BEQZ και LW. Μόνο η εξάρτηση ελέγχου μπορεί να το προκαλέσει αυτό. Προκειμένου να επιτραπεί η αναδιάταξη των εντολών αυτών (και παράλληλα να διατηρηθεί η εξάρτηση δεδομένων), θα θέλαμε απλά, όταν ακολουθείται η διακλάδωση, να μη λαμβάνουμε υπόψη την εξαίρεση. Στην Ενότητα 2.6, θα εξετάσουμε μια τεχνική υλικού, γνωστή ως *εικασία* (speculation), η οποία επιτρέπει την υπέρβαση αυτού του προβλήματος των εξαιρέσεων. Το Παράρτημα Z εξετάζει τεχνικές λογισμικού για την υποστήριξη της εικασίας.

Η δεύτερη ιδιότητα που διατηρείται μέσω της διασφάλισης των εξαρτήσεων δεδομένων και των εξαρτήσεων ελέγχου είναι η ροή δεδομένων. Η *ροή δεδομένων* ισοδυναμεί με την πραγματική ροή των τιμών των δεδομένων μεταξύ των εντολών που παράγουν τα αποτελέσματα και αυτών που τα καταναλώνουν. Οι διακλαδώσεις καθιστούν τη ροή δεδομένων δυναμική, καθώς επιτρέπουν η αφετηρία δεδομένων κάποιας συγκεκριμένης εντολής να προέρχεται από πολλά σημεία. Με άλλα λόγια, δεν επαρκεί απλά η διατήρηση των εξαρτήσεων δεδομένων, διότι οι εντολές μπορεί να έχουν εξάρτηση δεδομένων από περισσότερες από μία προηγούμενες εντολές. Η σειρά προγράμματος είναι αυτή που προσδιορίζει ποια από τις προηγούμενες εντολές θα είναι αυτή που στην πραγματικότητα θα παραδώσει κάποια τιμή δεδομένων σε μία εντολή. Η σειρά προγράμματος διασφαλίζεται διατηρώντας τις εξαρτήσεις ελέγχου.

Για παράδειγμα, ας εξετάσουμε το ακόλουθο τμήμα κώδικα:

```
        DADDU   R1,R2,R3
        BEQZ    R4,L
        DSUBU   R1,R5,R6
L:      ...
        OR      R7,R1,R8
```

Στο παράδειγμα αυτό, η τιμή του R1 που χρησιμοποιείται από την εντολή OR εξαρτάται από το αν η διακλάδωση θα ακολουθηθεί ή όχι. Η εξάρτηση δεδομένων αυτή καθ' εαυτήν δεν επαρκεί για τη διατήρηση της ορθότητας. Η εντολή OR έχει

εξάρτηση δεδομένων τόσο από την εντολή DADDU όσο και από την εντολή DSUBU, ωστόσο η διατήρηση της σειράς από μόνη της δεν επαρκεί για την ορθή εκτέλεση.

Αντιθέτως, όταν οι εντολές εκτελούνται, η ροή δεδομένων πρέπει να διατηρείται. Αν δεν ακολουθηθεί η διακλάδωση, τότε η OR πρέπει να χρησιμοποιήσει την τιμή του R1 που υπολογίζεται από την DSUBU, ενώ αν ακολουθηθεί η διακλάδωση, τότε η OR πρέπει να χρησιμοποιήσει την τιμή του R1 που υπολογίζεται από την DADDU. Η διατήρηση της εξάρτησης ελέγχου που έχει η OR από τη διακλάδωση οδηγεί στην παρεμπόδιση της πραγματοποίησης μη επιτρεπτών αλλαγών στη ροή δεδομένων. Για παρόμοιους λόγους, η εντολή DSUBU δεν μπορεί να μεταφερθεί επάνω από τη διακλάδωση. Όπως θα δούμε στην Ενότητα 2.6, η εικασία, που βοηθά στην επίλυση του προβλήματος των εξαιρέσεων, θα μας επιτρέψει, επίσης, να μειώσουμε την επίδραση της εξάρτησης ελέγχου, διατηρώντας συγχρόνως τη ροή δεδομένων.

Μερικές φορές μπορούμε να διαπιστώσουμε ότι η παραβίαση της εξάρτησης ελέγχου δεν μπορεί να επηρεάσει ούτε τη συμπεριφορά των εξαιρέσεων ούτε τη ροή δεδομένων. Ας εξετάσουμε την παρακάτω ακολουθία κώδικα:

```
        DADDU   R1,R2,R3
        BEQZ    R12,skip
        DSUBU   R4,R5,R6
        DADDU   R5,R4,R9
skip:   OR      R7,R8,R9
```

Ας υποθέσουμε ότι γνωρίζουμε ότι ο καταχωρητής προορισμού της εντολής DSUBU (R4) δεν χρησιμοποιείται μετά την εντολή με την ετικέτα skip (η ιδιότητα που σχετίζεται με το αν κάποια τιμή θα χρησιμοποιηθεί από κάποια επόμενη εντολή ονομάζεται *βιωσιμότητα*[6]). Αν ο R4 δεν χρησιμοποιείται, τότε η αλλαγή της τιμής του ακριβώς πριν τη διακλάδωση δεν θα επηρεάσει τη ροή δεδομένων, καθώς ο R4 είναι *νεκρός* (dead) (και όχι ενεργός) για το τμήμα του κώδικα που έπεται της εντολής skip. Συνεπώς, αν ο R4 είναι νεκρός και η υφιστάμενη εντολή DSUBU δεν μπορεί να παραγάγει εξαίρεση (διαφορετική από αυτές κατά τις οποίες ο επεξεργαστής συνεχίζει εκ νέου την εκτέλεση από την ίδια διεργασία), τότε μπορούμε να μετακινήσουμε την εντολή DSUBU πριν τη διακλάδωση, καθώς η ροή δεδομένων δεν μπορεί να επηρεαστεί.

Αν η διακλάδωση ακολουθηθεί, η εντολή DSUBU θα εκτελεστεί χωρίς να έχει καμία χρησιμότητα και ως εκ τούτου δεν θα επηρεάσει τα αποτελέσματα του προγράμματος. Αυτός ο τύπος χρονοδρομολόγησης κώδικα αποτελεί, επίσης, μορφή εικασίας, που συχνά ονομάζεται εικασία λογισμικού (software speculation), καθώς ο μεταγλωττιστής στοιχηματίζει στο αποτέλεσμα της διακλάδωσης. Στην περίπτωση αυτή, σύμφωνα με το στοιχηματισμό η διακλάδωση συνήθως δεν ακολουθείται. Πιο φιλόδοξοι μηχανισμοί εικασίας λογισμικού εξετάζονται

[6] Σ.τ.Μ. Ο αντίστοιχος αγγλικός όρος είναι "liveness".

στο Παράρτημα Ζ. Τις περισσότερες φορές που θα χρησιμοποιούμε τις λέξεις «εικασία» (speculation) ή «υποθετική» (speculative), θα είναι εμφανές αν χρησιμοποιείται μηχανισμός υλικού ή λογισμικού. Όταν δεν είναι εμφανές, τότε είναι προτιμότερο να χρησιμοποιούμε τη φράση «εικασία υλικού» ή «εικασία λογισμικού».

Η εξάρτηση ελέγχου διατηρείται μέσω της υλοποίησης της ανίχνευσης των κινδύνων ελέγχου, η οποία προκαλεί την εισαγωγή διαστημάτων αδράνειας ελέγχου. Τα διαστήματα αδράνειας ελέγχου μπορούν να εξαλειφθούν ή να μειωθούν χρησιμοποιώντας διάφορες τεχνικές υλικού και λογισμικού, οι οποίες εξετάζονται στην Ενότητα 2.3.

2.2 Βασικές Τεχνικές Μεταγλωττιστή για την Αξιοποίηση του ILP

Η ενότητα αυτή εξετάζει τη χρήση απλών τεχνολογιών μεταγλωττιστή για τη βελτίωση της ικανότητας του επεξεργαστή να αξιοποιεί τον ILP. Οι τεχνικές αυτές είναι σημαντικές για επεξεργαστές που χρησιμοποιούν στατική έκδοση ή στατική χρονοδρομολόγηση. Έχοντας ως εφόδιο αυτές τις τεχνολογίες μεταγλωττιστή, θα εξετάσουμε εν συντομία τη σχεδίαση και την απόδοση των επεξεργαστών που χρησιμοποιούν τη στατική έκδοση. Το Παράρτημα Ζ διερευνά πιο εξεζητημένες προσεγγίσεις μεταγλωττιστή που έχουν σχεδιαστεί για την παροχή της δυνατότητας στους επεξεργαστές να αξιοποιούν περισσότερο τον παραλληλισμό επιπέδου εντολών.

Βασική Χρονοδρομολόγηση Διασωλήνωσης και Ξεδίπλωμα Βρόχου

Για να διατηρείται η διασωλήνωση πλήρης, πρέπει να αξιοποιείται ο παραλληλισμός μεταξύ των εντολών, εντοπίζοντας ακολουθίες εντολών που δεν σχετίζονται μεταξύ τους, έτσι ώστε να είναι εφικτή η επικάλυψη της εκτέλεσής τους εντός της διασωλήνωσης. Για την αποφυγή της εισαγωγής διαστημάτων αδράνειας στη διασωλήνωση, μια εξαρτώμενη εντολή πρέπει να έχει απόσταση από την εντολή αφετηρίας, απόσταση που υπολογίζεται σε κύκλους ρολογιού και είναι ίση με το λανθάνοντα χρόνο διασωλήνωσης αυτής της εντολής αφετηρίας. Η ικανότητα του μεταγλωττιστή σε ό,τι αφορά την εκτέλεση αυτής της χρονοδρομολόγησης εξαρτάται τόσο από το εύρος του διαθέσιμου ILP του προγράμματος αυτού όσο και από τους λανθάνοντες χρόνους των λειτουργικών μονάδων της διασωλήνωσης. Το Σχήμα 2.2 παρουσιάζει τους λανθάνοντες χρόνους των μονάδων FP που, στο κεφάλαιο αυτό, θεωρούμε ότι ισχύουν πάντα, εκτός από περιπτώσεις στις οποίες θα αναγράφονται ρητά διαφορετικοί λανθάνοντες χρόνοι. Υποθέτουμε ότι χρησιμοποιείται η τυπική διασωλήνωση ακεραίων των πέντε σταδίων, έτσι ώστε οι διακλαδώσεις να έχουν καθυστέρηση της τάξης του ενός κύκλου. Υποθέτουμε

Εντολή που παράγει το αποτέλεσμα	Εντολή που χρησιμοποιεί το αποτέλεσμα	Λανθάνων χρόνος σε κύκλους ρολογιού
Λειτουργία FP της ALU	Άλλη μία Λειτουργία FP της ALU	3
Λειτουργία FP της ALU	Αποθήκευση διπλής ακρίβειας	2
Φόρτωση διπλής ακρίβειας	Λειτουργία FP της ALU	1
Φόρτωση διπλής ακρίβειας	Αποθήκευση διπλής ακρίβειας	0

Σχήμα 2.2 Οι λανθάνοντες χρόνοι των λειτουργιών FP που χρησιμοποιούνται στο κεφάλαιο αυτό. Η τελευταία στήλη αντικατοπτρίζει το πλήθος των κύκλων ρολογιού που απαιτείται να μεσολαβήσουν για την αποφυγή της εισαγωγής διαστημάτων αδράνειας. Οι αριθμοί αυτοί είναι παρόμοιοι με τους μέσους λανθάνοντες χρόνους που συναντούμε στις μονάδες FP. Ο λανθάνων χρόνος ανάμεσα στη φόρτωση και την αποθήκευση FP ισούται με 0, καθώς το αποτέλεσμα της φόρτωσης μπορεί να προωθηθεί χωρίς να τεθεί η αποθήκευση σε αδράνεια. Εξακολουθούμε να θεωρούμε ότι ο λανθάνων χρόνος της φόρτωσης ακεραίων ισούται με 1 και ότι ο λανθάνων χρόνος των πράξεων της ALU με ακεραίους ισούται με 0.

ότι οι λειτουργικές μονάδες υποστηρίζουν πλήρως τη διασωλήνωση ή διαθέτουν πολλαπλά αντίγραφα (τόσα σε αριθμό όσο το βάθος της διασωλήνωσης), έτσι ώστε να είναι εφικτή η έκδοση οποιουδήποτε τύπου λειτουργίας σε κάθε κύκλο ρολογιού, χωρίς να υφίστανται δομικοί κίνδυνοι.

Στην υποενότητα αυτή, εξετάζουμε τον τρόπο με τον οποίο ο μεταγλωττιστής μπορεί να αυξήσει το εύρος του διαθέσιμου ILP αλλάζοντας τη μορφή των βρόχων. Το παράδειγμα αυτό αποτελεί σημείο αναφοράς τόσο για την περιγραφή της σημαντικής αυτής τεχνικής όσο για την κατανόηση των πιο σύνθετων τεχνικών που περιγράφονται στο Παράρτημα Z. Θα βασιστούμε στο ακόλουθο τμήμα κώδικα, στο οποίο προστίθεται μια βαθμωτή μεταβλητή σε ένα διάνυσμα:

```
f    (i=1000; i>0; i=i-1)
         x[i] = x[i] + s;
```

Μπορούμε να διαπιστώσουμε ότι ο βρόχος είναι παράλληλος, παρατηρώντας ότι το σώμα κάθε επανάληψης είναι ανεξάρτητο. Θα τυποποιήσουμε την έννοια αυτή στο Παράρτημα Z και θα περιγράψουμε με ποιον τρόπο μπορούμε να ελέγχουμε κατά το χρόνο μεταγλώττισης αν οι επαναλήψεις του βρόχου είναι ανεξάρτητες. Πρώτα, ας μελετήσουμε την απόδοση του βρόχου αυτού, εξετάζοντας τον τρόπο με τον οποίο μπορούμε να χρησιμοποιήσουμε τον παραλληλισμό για τη βελτίωση της απόδοσης της διασωλήνωσης της MIPS, που διαθέτει τους λανθάνοντες χρόνους που έχουν περιγραφεί παραπάνω.

Το πρώτο βήμα που πρέπει να πραγματοποιηθεί είναι η μετατροπή του παραπάνω τμήματος κώδικα στη συμβολική γλώσσα της MIPS. Στο ακόλουθο τμήμα

κώδικα, ο R1 περιλαμβάνει αρχικά τη διεύθυνση του στοιχείου του πίνακα με την υψηλότερη διεύθυνση, ενώ ο F2 περιέχει τη βαθμωτή τιμή *s*. Ο καταχωρητής R2 έχει προϋπολογιστεί, έτσι ώστε η 8(R2) να αντικατοπτρίζει τη διεύθυνση του τελευταίου στοιχείου το οποίο πρέπει να χρησιμοποιηθεί κατά την εκτέλεση.

Ο κώδικας της MIPS, χωρίς να έχει χρονοδρομολογηθεί εντός της διασωλήνωσης, έχει την παρακάτω μορφή:

```
Loop: L.D     F0,0(R1)    ; F0 = στοιχείο πίνακα
      ADD.D   F4,F0,F2    ; πρόσθεση του βαθμωτού πεδίου που βρίσκεται στον F2
      S.D     F4,0(R1)    ; αποθήκευση αποτελέσματος
      DADDUI  R1,R1,#-8   ; μείωση δείκτη κατά 8 bytes
      BNE     R1,R2,Loop  ; διακλάδωση R1!=R2
```

Ας ξεκινήσουμε να εξετάζουμε πόσο ικανοποιητικά θα εκτελεστεί ο βρόχος αυτός, όταν χρονοδρομολογηθεί στην απλή διασωλήνωση της MIPS που διαθέτει τους λανθάνοντες χρόνους του Σχήματος 2.2.

Παράδειγμα: Ας παρουσιάσουμε την εικόνα της MIPS, τόσο στην περίπτωση εφαρμογής της χρονοδρομολόγησης όσο και στην περίπτωση μη εφαρμογής της, συμπεριλαμβανομένων των διαστημάτων αδράνειας ή των αδρανών κύκλων ρολογιού. Θα χρονοδρομολογήσουμε τις καθυστερήσεις των λειτουργιών κινητής υποδιαστολής, ωστόσο πρέπει να έχουμε κατά νου ότι δεν θα λάβουμε υπόψη τις καθυστερημένες διακλαδώσεις.

Απάντηση: Χωρίς χρονοδρομολόγηση, ο βρόχος θα εκτελεστεί εντός 9 κύκλων ως ακολούθως:

			Κύκλος ρολογιού έκδοσης
Loop:	L.D	F0,0(R1)	1
	διάστημα αδράνειας		2
	ADD.D	F4,F0,F2	3
	διάστημα αδράνειας		4
	διάστημα αδράνειας		5
	S.D	F4,0(R1)	6
	DADDUI	R1,R1,#-8	7
	διάστημα αδράνειας		8
	BNE	R1,R2,Loop	9

Μπορούμε να χρονοδρομολογήσουμε το βρόχο, έτσι ώστε να υφίστανται μόνο δύο διαστήματα αδράνειας και να μειώσουμε το χρόνο στους 7 κύκλους:

```
Loop:   L.D     F0,0(R1)
        DADDUI  R1,R1,#-8
```

```
ADD.D    F4,F0,F2
διάστημα αδράνειας
διάστημα αδράνειας
S.D      F4,8(R1)
BNE      R1,R2,Loop
```

Τα διαστήματα αδράνειας μετά την εντολή ADD.D είναι απαραίτητα για τη χρήση της S.D.

Στο προηγούμενο παράδειγμα, ολοκληρώνουμε μία επανάληψη του βρόχου και αποθηκεύουμε ένα στοιχείο του πίνακα κάθε 7 κύκλους ρολογιού, ωστόσο η πραγματική εργασία για την εκτέλεση των απαραίτητων λειτουργιών στο στοιχείο του πίνακα χρειάζεται μόνο 3 από αυτούς τους 7 κύκλους (μόνο τους κύκλους φόρτωσης, πρόσθεσης και αποθήκευσης). Οι εναπομένοντες 4 κύκλοι ρολογιού αντιστοιχούν στην επιβάρυνση του βρόχου - στις εντολές DADDUI και BNE - και σε δύο διαστήματα αδράνειας. Για να εξαλειφθούν οι 4 αυτοί κύκλοι ρολογιού πρέπει να έχουμε μεγαλύτερο αριθμό λειτουργιών σε σχέση με τον αριθμό των εντολών επιβάρυνσης.

Μια απλή προσέγγιση αύξησης του αριθμού των εντολών σε σχέση με τις εντολές διακλάδωσης και επιβάρυνσης είναι το *ξεδίπλωμα βρόχου* (loop unrolling). Το ξεδίπλωμα απλά δημιουργεί πολλαπλά αντίγραφα του σώματος του βρόχου και προσαρμόζει κατάλληλα τον κώδικα τερματισμού του βρόχου.

Το ξεδίπλωμα βρόχου μπορεί, επίσης, να χρησιμοποιηθεί για τη βελτίωση της χρονοδρομολόγησης. Επειδή εξαλείφει τη διακλάδωση, επιτρέπει τη συνδυαστική χρονοδρομολόγηση εντολών που ανήκουν σε διαφορετικές επαναλήψεις. Στην περίπτωση αυτή, μπορούμε να εξαλείψουμε τα διαστήματα αδράνειας της χρήσης δεδομένων, δημιουργώντας επιπρόσθετες ανεξάρτητες εντολές εντός του σώματος του βρόχου. Αν απλά δημιουργούσαμε αντίγραφα των εντολών, τότε, όταν θα ξεδιπλώναμε το βρόχο, η χρήση των ίδιων καταχωρητών θα μπορούσε να παρεμποδίσει την ουσιαστική χρονοδρομολόγηση του βρόχου. Κατά συνέπεια, χρειάζεται να χρησιμοποιούμε διαφορετικούς καταχωρητές σε κάθε επανάληψη, κάτι που αυξάνει τον απαιτούμενο αριθμό των καταχωρητών.

Παράδειγμα: Ας παρουσιάσουμε το ξεδίπλωμα του βρόχου μας, έτσι ώστε να υπάρχουν τέσσερα αντίγραφα του σώματος βρόχου, με την προϋπόθεση ότι το R1 - R2 (το μέγεθος δηλαδή του πίνακα) είναι πολλαπλάσιο του 32, κάτι που σημαίνει ότι το πλήθος των επαναλήψεων του βρόχου είναι πολλαπλάσιο του 4. Θα εξαλείψουμε τους όποιους περιττούς υπολογισμούς και δεν θα χρησιμοποιήσουμε εκ νέου κανέναν από τους καταχωρητές.

Απάντηση: Παρακάτω παρατίθεται το αποτέλεσμα μετά τη συνένωση των εντολών DADDUI και την παράλειψη των περιττών λειτουργιών BNE που αναπαρά-

γονται κατά τη διάρκεια του ξεδιπλώματος. Αξίζει να σημειωθεί ότι ο R2 πρέπει λάβει την κατάλληλη τιμή, έτσι ώστε η 32(R2) να αποτελεί τη διεύθυνση αφετηρίας των τεσσάρων τελευταίων στοιχείων.

```
Loop:   L.D     F0,0(R1)
        ADD.D   F4,F0,F2
        S.D     F4,0(R1)      ;παράλειψη των DADDUI και BNE
        L.D     F6,-8(R1)
        ADD.D   F8,F6,F2
        S.D     F8,-8(R1)     ;παράλειψη των DADDUI και BNE
        L.D     F10,-16(R1)
        ADD.D   F12,F10,F2
        S.D     F12,-16(R1)   ;παράλειψη των DADDUI και BNE
        L.D     F14,-24(R1)
        ADD.D   F16,F14,F2
        S.D     F16,-24(R1)
        DADDUI  R1,R1,#-32
        BNE     R1,R2,Loop
```

Έχουμε εξαλείψει τρεις διακλαδώσεις και τρεις εντολές μείωσης του R1. Οι διευθύνσεις των εντολών φόρτωσης και αποθήκευσης έχουν αντικατασταθεί, έτσι ώστε να είναι εφικτή η συνένωση των εντολών DADDUI οι οποίες χρησιμοποιούν τον R1. Η βελτιστοποίηση αυτή μπορεί να φαίνεται, αλλά δεν είναι, απλή. Απαιτεί τη συμβολική αντικατάσταση και απλοποίηση (symbolic substitution and simplification). Η συμβολική αντικατάσταση και απλοποίηση αναδιατάσσει τις παραστάσεις, έτσι ώστε να είναι δυνατή η σύντμηση των σταθερών, κάτι που επιτρέπει σε παραστάσεις, όπως η "$((i+1)+1)$", να γράφονται εκ νέου ως "$(i+(1+1))$ και στη συνέχεια να απλοποιούνται στη μορφή "$i+2$". Στο Παράρτημα Z θα εξετάσουμε πιο γενικές μορφές των βελτιστοποιήσεων αυτών, οι οποίες μπορούν να εξαλείφουν τους εξαρτώμενους υπολογισμούς.

Χωρίς χρονοδρομολόγηση, κάθε λειτουργία του ξεδιπλωμένου βρόχου ακολουθείται από μια εξαρτώμενη λειτουργία και επομένως προκαλεί την εισαγωγή διαστήματος αδράνειας. Ο βρόχος αυτός θα εκτελεστεί εντός 27 κύκλων ρολογιού - κάθε εντολή LD έχει 1 διάστημα αδράνειας, κάθε ADDD 2, η DADDUI 1 και σε αυτά προστίθενται οι 14 κύκλοι έκδοσης εντολών - ή εντός 6.75 κύκλων ρολογιού για καθένα από τα τέσσερα στοιχεία, ωστόσο μπορεί να χρονοδρομολογηθεί έχοντας ως στόχο τη σημαντική βελτίωση της απόδοσης. Το ξεδίπλωμα βρόχου συνήθως πραγματοποιείται στα πρώτα βήματα της διαδικασίας μεταγλώττισης, έτσι ώστε να αποκαλύπτονται οι περιττοί υπολογισμοί και να εξαλείφονται από το βελτιστοποιητή (optimizer).

Στα πραγματικά προγράμματα, συνήθως δεν γνωρίζουμε το ανώτερο όριο του βρόχου. Ας υποθέσουμε ότι το όριο αυτό ισούται με n και ότι θέλουμε

να ξεδιπλώσουμε το βρόχο, προκειμένου να δημιουργήσουμε k αντίγραφα του σώματός του. Αντί να δημιουργήσουμε ένα και μοναδικό ξεδιπλωμένο βρόχο, παράγουμε ένα ζεύγος διαδοχικών βρόχων. Ο πρώτος εκτελείται ($n \bmod k$) φορές και έχει το σώμα του αρχικού βρόχου. Ο δεύτερος αποτελεί το σώμα του ξεδιπλωμένου βρόχου το οποίο περιβάλλεται από έναν εξωτερικό βρόχο, ο οποίος επαναλαμβάνεται (n/k) φορές. Για μεγάλες τιμές του n, το μεγαλύτερο μέρος του χρόνου εκτέλεσης δαπανάται στο σώμα του ξεδιπλωμένου βρόχου.

Στο προηγούμενο παράδειγμα, το ξεδίπλωμα βελτιώνει την απόδοση του βρόχου μέσω της εξάλειψης των εντολών επιβάρυνσης, παρόλο που αυξάνει σημαντικά το μέγεθος του κώδικα. Ποια θα είναι η απόδοση του ξεδιπλωμένου βρόχου, όταν χρονοδρομολογείται εντός της διασωλήνωσης που περιγράψαμε νωρίτερα;

Παράδειγμα: Ας παρουσιάσουμε τον ξεδιπλωμένο βρόχο του προηγούμενου παραδείγματος, αφότου έχει χρονοδρομολογηθεί εντός της διασωλήνωσης που έχει τους λανθάνοντες χρόνους του Σχήματος 2.2.

Απάντηση:

```
Loop:   L.D      F0,0(R1)
        L.D      F6,-8(R1)
        L.D      F10,-16(R1)
        L.D      F14,-24(R1)
        ADD.D     F4,F0,F2
        ADD.D    F8,F6,F2
        ADD.D    F12,F10,F2
        ADD.D    F16,F14,F2
        S.D      F4,0(R1)
        S.D      F8,-8(R1)
        DADDUI   R1,R1,#-32
        S.D      F12,16(R1)
        S.D      F16,8(R1)
        BNE      R1,R2,Loop
```

Ο χρόνος εκτέλεσης του ξεδιπλωμένου βρόχου έχει μειωθεί στους 14 συνολικά κύκλους ρολογιού ή σε 3.5 κύκλους ρολογιού ανά στοιχείο, σε αντίθεση με τους 9 κύκλους ανά στοιχείο που απαιτούνταν πριν το ξεδίπλωμα ή τη χρονοδρομολόγηση και τους 7 κύκλους που απαιτούνταν στην περίπτωση εφαρμογής της χρονοδρομολόγησης, χωρίς την παράλληλη εφαρμογή του ξεδιπλώματος.

Τα οφέλη της χρονοδρομολόγησης του ξεδιπλωμένου βρόχου είναι μεγαλύτερα απ' ό,τι της χρονοδρομολόγησης του αρχικού βρόχου. Η αύξηση αυτή προκύπτει, διότι το ξεδίπλωμα του βρόχου έχει ως αποτέλεσμα την αποκάλυψη

γονται κατά τη διάρκεια του ξεδιπλώματος. Αξίζει να σημειωθεί ότι ο R2 πρέπει λάβει την κατάλληλη τιμή, έτσι ώστε η 32(R2) να αποτελεί τη διεύθυνση αφετηρίας των τεσσάρων τελευταίων στοιχείων.

```
Loop:   L.D     F0,0(R1)
        ADD.D   F4,F0,F2
        S.D     F4,0(R1)      ;παράλειψη των DADDUI και BNE
        L.D     F6,-8(R1)
        ADD.D   F8,F6,F2
        S.D     F8,-8(R1)     ;παράλειψη των DADDUI και BNE
        L.D     F10,-16(R1)
        ADD.D   F12,F10,F2
        S.D     F12,-16(R1)   ;παράλειψη των DADDUI και BNE
        L.D     F14,-24(R1)
        ADD.D   F16,F14,F2
        S.D     F16,-24(R1)
        DADDUI  R1,R1,#-32
        BNE     R1,R2,Loop
```

Έχουμε εξαλείψει τρεις διακλαδώσεις και τρεις εντολές μείωσης του R1. Οι διευθύνσεις των εντολών φόρτωσης και αποθήκευσης έχουν αντικατασταθεί, έτσι ώστε να είναι εφικτή η συνένωση των εντολών DADDUI οι οποίες χρησιμοποιούν τον R1. Η βελτιστοποίηση αυτή μπορεί να φαίνεται, αλλά δεν είναι, απλή. Απαιτεί τη συμβολική αντικατάσταση και απλοποίηση (symbolic substitution and simplification). Η συμβολική αντικατάσταση και απλοποίηση αναδιατάσσει τις παραστάσεις, έτσι ώστε να είναι δυνατή η σύνιμηση των σταθερών, κάτι που επιτρέπει σε παραστάσεις, όπως η "$((i+1)+1)$", να γράφονται εκ νέου ως "$(i+(1+1))$ και στη συνέχεια να απλοποιούνται στη μορφή "$i+2$". Στο Παράρτημα Z θα εξετάσουμε πιο γενικές μορφές των βελτιστοποιήσεων αυτών, οι οποίες μπορούν να εξαλείφουν τους εξαρτώμενους υπολογισμούς.

Χωρίς χρονοδρομολόγηση, κάθε λειτουργία του ξεδιπλωμένου βρόχου ακολουθείται από μια εξαρτώμενη λειτουργία και επομένως προκαλεί την εισαγωγή διαστήματος αδράνειας. Ο βρόχος αυτός θα εκτελεστεί εντός 27 κύκλων ρολογιού - κάθε εντολή LD έχει 1 διάστημα αδράνειας, κάθε ADDD 2, η DADDUI 1 και σε αυτά προστίθενται οι 14 κύκλοι έκδοσης εντολών - ή εντός 6.75 κύκλων ρολογιού για καθένα από τα τέσσερα στοιχεία, ωστόσο μπορεί να χρονοδρομολογηθεί έχοντας ως στόχο τη σημαντική βελτίωση της απόδοσης. Το ξεδίπλωμα βρόχου συνήθως πραγματοποιείται στα πρώτα βήματα της διαδικασίας μεταγλώττισης, έτσι ώστε να αποκαλύπτονται οι περιττοί υπολογισμοί και να εξαλείφονται από το βελτιστοποιητή (optimizer).

Στα πραγματικά προγράμματα, συνήθως δεν γνωρίζουμε το ανώτερο όριο του βρόχου. Ας υποθέσουμε ότι το όριο αυτό ισούται με n και ότι θέλουμε

να ξεδιπλώσουμε το βρόχο, προκειμένου να δημιουργήσουμε k αντίγραφα του σώματός του. Αντί να δημιουργήσουμε ένα και μοναδικό ξεδιπλωμένο βρόχο, παράγουμε ένα ζεύγος διαδοχικών βρόχων. Ο πρώτος εκτελείται ($n \mod k$) φορές και έχει το σώμα του αρχικού βρόχου. Ο δεύτερος αποτελεί το σώμα του ξεδιπλωμένου βρόχου το οποίο περιβάλλεται από έναν εξωτερικό βρόχο, ο οποίος επαναλαμβάνεται (n/k) φορές. Για μεγάλες τιμές του n, το μεγαλύτερο μέρος του χρόνου εκτέλεσης δαπανάται στο σώμα του ξεδιπλωμένου βρόχου.

Στο προηγούμενο παράδειγμα, το ξεδίπλωμα βελτιώνει την απόδοση του βρόχου μέσω της εξάλειψης των εντολών επιβάρυνσης, παρόλο που αυξάνει σημαντικά το μέγεθος του κώδικα. Ποια θα είναι η απόδοση του ξεδιπλωμένου βρόχου, όταν χρονοδρομολογείται εντός της διασωλήνωσης που περιγράψαμε νωρίτερα;

Παράδειγμα: Ας παρουσιάσουμε τον ξεδιπλωμένο βρόχο του προηγούμενου παραδείγματος, αφότου έχει χρονοδρομολογηθεί εντός της διασωλήνωσης που έχει τους λανθάνοντες χρόνους του Σχήματος 2.2.

Απάντηση:

```
Loop:   L.D      F0,0(R1)
        L.D      F6,-8(R1)
        L.D      F10,-16(R1)
        L.D      F14,-24(R1)
        ADD.D     F4,F0,F2
        ADD.D    F8,F6,F2
        ADD.D    F12,F10,F2
        ADD.D    F16,F14,F2
        S.D      F4,0(R1)
        S.D      F8,-8(R1)
        DADDUI   R1,R1,#-32
        S.D      F12,16(R1)
        S.D      F16,8(R1)
        BNE      R1,R2,Loop
```

Ο χρόνος εκτέλεσης του ξεδιπλωμένου βρόχου έχει μειωθεί στους 14 συνολικά κύκλους ρολογιού ή σε 3.5 κύκλους ρολογιού ανά στοιχείο, σε αντίθεση με τους 9 κύκλους ανά στοιχείο που απαιτούνταν πριν το ξεδίπλωμα ή τη χρονοδρομολόγηση και τους 7 κύκλους που απαιτούνταν στην περίπτωση εφαρμογής της χρονοδρομολόγησης, χωρίς την παράλληλη εφαρμογή του ξεδιπλώματος.

Τα οφέλη της χρονοδρομολόγησης του ξεδιπλωμένου βρόχου είναι μεγαλύτερα απ' ό,τι της χρονοδρομολόγησης του αρχικού βρόχου. Η αύξηση αυτή προκύπτει, διότι το ξεδίπλωμα του βρόχου έχει ως αποτέλεσμα την αποκάλυψη

περισσότερων υπολογισμών, οι οποίοι μπορούν να χρονοδρομολογηθούν για τη μείωση των διαστημάτων αδράνειας. Ο παραπάνω κώδικας δεν περιλαμβάνει κανένα διάστημα αδράνειας. Η χρονοδρομολόγηση του βρόχου χρησιμοποιώντας την παραπάνω μέθοδο καθιστά αναγκαίο να αντιληφθεί κανείς ότι οι εντολές φόρτωσης και αποθήκευσης είναι ανεξάρτητες και επομένως μπορούν να εναλλάσσονται.

Σύνοψη του Ξεδιπλώματος Βρόχου και της Χρονοδρομολόγησης

Σε όλο το κεφάλαιο αυτό, καθώς και στο Παράρτημα Ζ, θα εξετάσουμε διάφορες τεχνικές υλικού και λογισμικού, οι οποίες επιτρέπουν την αξιοποίηση του παραλληλισμού επιπέδου εντολών για την πλήρη εκμετάλλευση των δυνατοτήτων των λειτουργικών μονάδων του επεξεργαστή. Το πιο σημαντικό ζητούμενο για τις τεχνικές αυτές είναι η γνώση του χρόνου και του τρόπου με τον οποίο μπορεί να τροποποιηθεί η διάταξη των εντολών. Στο παράδειγμά μας, πραγματοποιήσαμε πολλές τέτοιου είδους αλλαγές, οι οποίες για εμάς, που είμαστε ανθρώπινα όντα, ήταν εμφανές ότι μπορούσαν να πραγματωθούν. Στην πράξη, η διαδικασία αυτή πρέπει να εκτελείται με μεθοδικό τρόπο είτε από το μεταγλωττιστή είτε από το υλικό. Προκειμένου να φτάσουμε στον τελικό ξεδιπλωμένο κώδικα έπρεπε να λάβουμε τις ακόλουθες αποφάσεις και να πραγματοποιήσουμε τους παρακάτω μετασχηματισμούς:

- Να προσδιορίσουμε ότι το ξεδίπλωμα του βρόχου θα αποτελέσει χρήσιμη διαδικασία, διαπιστώνοντας ότι οι επαναλήψεις του βρόχου είναι ανεξάρτητες, εκτός από τις εντολές του κώδικα που σχετίζονται με τη διατήρηση του βρόχου.

- Να χρησιμοποιήσουμε διαφορετικούς καταχωρητές, ώστε να αποφύγουμε περιττούς περιορισμούς οι οποίοι θα επιβάλλονταν εξαιτίας της χρήσης των ίδιων καταχωρητών για διαφορετικούς υπολογισμούς.

- Να εξαλείψουμε τις επιπλέον εντολές ελέγχου και εντολές διακλάδωσης και να προσαρμόσουμε τον κώδικα τερματισμού και πραγματοποίησης επαναλήψεων.

- Να προσδιορίσουμε ότι οι εντολές φόρτωσης και αποθήκευσης του ξεδιπλωμένου βρόχου μπορούν να εναλλάσσονται, παρατηρώντας ότι οι εντολές φόρτωσης και αποθήκευσης που ανήκουν σε διαφορετικές επαναλήψεις είναι ανεξάρτητες μεταξύ τους. Ο μετασχηματισμός αυτός απαιτεί την ανάλυση των διευθύνσεων της μνήμης και τη διαπίστωση ότι οι εντολές αυτές δεν έχουν αναφορές στην ίδια διεύθυνση.

- Να χρονοδρομολογήσουμε τον κώδικα, διατηρώντας όλες τις εξαρτήσεις που απαιτούνται, έτσι ώστε να έχουμε το ίδιο αποτέλεσμα με τον αρχικό

κώδικα.

Αυτό που αποτελεί πρωταρχική απαίτηση την οποία όλοι αυτοί οι μετασχηματισμοί υποδηλώνουν είναι η ανάγκη κατανόησης του τρόπου με τον οποίο η μία εντολή εξαρτάται από την άλλη, καθώς και του τρόπου με τον οποίο οι εντολές μπορούν να τροποποιηθούν ή να αναδιαταχθούν με βάση τις δεδομένες εξαρτήσεις.

Υπάρχουν τρεις διαφορετικοί τύποι περιορισμών που τίθενται στα οφέλη τα οποία μπορούν να προκύψουν μέσω του ξεδιπλώματος βρόχου: η μείωση του μεγέθους της επιβάρυνσης που επιτυγχάνεται σε κάθε ξεδίπλωμα, οι περιορισμοί που αφορούν το μέγεθος του κώδικα και οι περιορισμοί του μεταγλωττιστή. Ας εξετάσουμε πρώτα το ζήτημα της επιβάρυνσης του βρόχου. Όταν ξεδιπλώσαμε το βρόχο τέσσερις φορές, επιτύχαμε ικανοποιητικό παραλληλισμό μεταξύ των εντολών του βρόχου, οι οποίες μπορούσαν να χρονοδρομολογηθούν χωρίς την εισαγωγή κύκλων αδράνειας. Στην πραγματικότητα, από τους 14 κύκλους ρολογιού, μόνο οι 2 κύκλοι αντιστοιχούσαν στην επιβάρυνση του βρόχου: ο κύκλος της εντολής `DADDUI`, η οποία διατηρεί την τιμή του δείκτη (index) και ο κύκλος της `BNE`, η οποία τερματίζει το βρόχο. Αν ο βρόχος ξεδιπλωθεί οκτώ φορές, τότε η επιβάρυνση μειώνεται από $1/2$ κύκλους ανά αρχική επανάληψη σε $1/4$.

Ο δεύτερος περιορισμός του ξεδιπλώματος αφορά την αύξηση του μεγέθους του τελικού κώδικα. Στους μεγαλύτερους βρόχους, η αύξηση του μεγέθους του κώδικα μπορεί να αποτελέσει ζήτημα για το οποίο πρέπει να ληφθεί μέριμνα, ιδίως στην περίπτωση που προκαλεί αύξηση στο ρυθμό αστοχίας της κρυφής μνήμης (cache miss).

Άλλος ένας παράγοντας, συνήθως πιο σημαντικός από το μέγεθος του κώδικα, είναι το ενδεχόμενο έλλειμμα καταχωρητών που μπορεί να δημιουργηθεί εξαιτίας της εντατικής εφαρμογής του ξεδιπλώματος και της χρονοδρομολόγησης. Το δευτερεύον αυτό φαινόμενο που είναι αποτέλεσμα της χρονοδρομολόγησης εντολών σε μεγάλα τμήματα κώδικα ονομάζεται *πίεση καταχωρητών* (register pressure). Το αποτέλεσμα αυτό προκύπτει, διότι η χρονοδρομολόγηση του κώδικα που έχει ως απώτερο στόχο την αύξηση του ILP προκαλεί την αύξηση του αριθμού των ενεργών τιμών. Μετά από την εντατική χρονοδρομολόγηση των εντολών, ενδέχεται να μην είναι εφικτή η ανάθεση όλων των ενεργών τιμών στους καταχωρητές. Ο μετασχηματισμένος κώδικας, παρόλο που θεωρητικά είναι ταχύτερος, μπορεί να απολέσει τα πλεονεκτήματά του ή ένα μέρος αυτών, επειδή οδηγεί σε έλλειψη καταχωρητών. Χωρίς το ξεδίπλωμα, η εντατική χρονοδρομολόγηση περιορίζεται επαρκώς από τις διακλαδώσεις και ως εκ τούτου η πίεση καταχωρητών σπάνια αποτελεί πρόβλημα. Ο συνδυασμός του ξεδιπλώματος και της εντατικής χρονοδρομολόγησης μπορεί, ωστόσο, να προκαλέσει το πρόβλημα αυτό. Το πρόβλημα αυτό αποτελεί τεράστια πρόκληση στους επεξεργαστές πολλαπλής έκδοσης, οι οποίοι απαιτούν την αξιοποίηση περισσότερων ανεξάρτητων ακολουθιών με εντολές, των οποίων η εκτέλεση μπορεί να επικαλύπτεται. Σε γε-

νικές γραμμές, η χρήση εξελιγμένων μετασχηματισμών υψηλού επιπέδου, των οποίων ο βαθμός της ενδεχόμενης βελτίωσης που μπορούν να επιφέρουν είναι δύσκολο να μετρηθεί πριν την παραγωγή του λεπτομερούς κώδικα, έχει οδηγήσει σε σημαντικές αυξήσεις στην πολυπλοκότητα των σύγχρονων μεταγλωττιστών.

Το ξεδίπλωμα βρόχου αποτελεί απλή, αλλά χρήσιμη, μέθοδο για την αύξηση του μεγέθους των ευθύγραμμων τμημάτων κώδικα που μπορούν να χρονοδρομολογηθούν με αποτελεσματικό τρόπο. Ο μετασχηματισμός αυτός είναι χρήσιμος για διάφορους επεξεργαστές, από αυτούς που υποστηρίζουν απλές διασωληνώσεις, όπως αυτές που μέχρι στιγμής έχουμε εξετάσει, έως και υπερβαθμωτούς επεξεργαστές πολλαπλής έκδοσης και επεξεργαστές VLIW τους οποίους θα μελετήσουμε αργότερα στο κεφάλαιο αυτό.

2.3 Μείωση του Κόστους των Διακλαδώσεων μέσω Πρόβλεψης

Εξαιτίας της ανάγκης ενίσχυσης των εξαρτήσεων ελέγχου μέσω των κινδύνων διακλάδωσης και της εισαγωγής διαστημάτων αδράνειας, οι διακλαδώσεις έχουν αρνητικές επιπτώσεις στην απόδοση. Το ξεδίπλωμα βρόχου είναι μία από τις μεθόδους μείωσης του αριθμού των κινδύνων διακλάδωσης. Μπορούμε, επίσης, να μειώσουμε τις απώλειες της απόδοσης μέσω πρόβλεψης του τρόπου συμπεριφοράς τους.

Η συμπεριφορά των διακλαδώσεων μπορεί να προβλεφθεί τόσο στατικά, κατά το χρόνο μεταγλώττισης όσο και δυναμικά από το υλικό κατά το χρόνο εκτέλεσης. Οι στατικοί μηχανισμοί πρόβλεψης (static predictors) της διακλάδωσης μερικές φορές χρησιμοποιούνται σε επεξεργαστές, όπου η προσδοκώμενη συμπεριφορά της διακλάδωσης μπορεί να προβλεφθεί με μεγάλη επιτυχία κατά το χρόνο μεταγλώττισης. Η στατική πρόβλεψη μπορεί, επίσης, να χρησιμοποιηθεί για την παροχή βοήθειας στους δυναμικούς μηχανισμούς πρόβλεψης.

Στατική Πρόβλεψη Διακλάδωσης

Στο Παράρτημα Α, εξετάζουμε τις καθυστερημένες διακλαδώσεις, οι οποίες αποτελούν βασικό χαρακτηριστικό της αρχιτεκτονικής η οποία υποστηρίζει τη στατική πρόβλεψη διακλάδωσης. Η δυνατότητα ακριβούς πρόβλεψης της διακλάδωσης κατά το χρόνο μεταγλώττισης είναι, επίσης, χρήσιμη για τη χρονοδρομολόγηση των κινδύνων δεδομένων. Το ξεδίπλωμα βρόχου αποτελεί άλλο ένα παράδειγμα τεχνικής που εξαρτάται από την πρόβλεψη των διακλαδώσεων και έχει ως στόχο τη βελτίωση της χρονοδρομολόγησης του κώδικα.

Προκειμένου να αναδιατάξει κανείς τον κώδικα που περιβάλλει τις διακλαδώσεις, έτσι ώστε να εκτελείται ταχύτερα, χρειάζεται να προβλέψει στατικά τη διακλάδωση κατά τη μεταγλώττιση του προγράμματος. Υπάρχουν αρκετές δια-

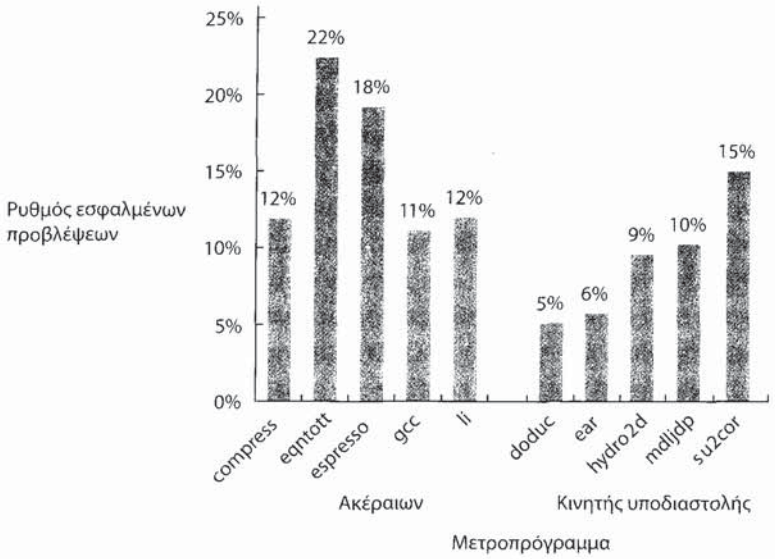
φορετικές μέθοδοι για τη στατική πρόβλεψη της συμπεριφοράς των διακλαδώσεων. Στην πιο απλή προσέγγιση μπορεί κανείς να προβλέψει ότι η διακλάδωση ακολουθείται. Η προσέγγιση αυτή έχει μέσο ρυθμό εσφαλμένων προβλέψεων που ισούται με τη συχνότητα των διακλαδώσεων που δεν ακολουθούνται, η οποία για τα προγράμματα του SPEC ισούται με 34%. Δυστυχώς, ο ρυθμός εσφαλμένων προβλέψεων των προγραμμάτων του SPEC κυμαίνεται μεταξύ τιμών που δεν υποδηλώνουν ιδιαίτερη ακρίβεια (59%) έως και τιμών που υποδηλώνουν απόλυτη ακρίβεια (9%).

Μία τεχνική που είναι πιο ακριβής αφορά την πρόβλεψη της διακλάδωσης με βάση τις πληροφορίες του προφίλ (profile information), οι οποίες συλλέγονται κατά τη διάρκεια προηγούμενων εκτελέσεων. Το βασικό στοιχείο που καθιστά αξιόλογη την προσέγγιση αυτή είναι το γεγονός ότι έχει παρατηρηθεί πως η συμπεριφορά των διακλαδώσεων συχνά ακολουθεί κατανομή δύο κορυφών (bimodal distribution). Δηλαδή, η μεμονωμένη διακλάδωση συχνά εμφανίζει την ίδια συμπεριφορά, κάτι που σημαίνει είτε ότι ακολουθείται σχεδόν πάντα είτε ότι δεν ακολουθείται σχεδόν πάντα. Το Σχήμα 2.3 παρουσιάζει την επιτυχία της πρόβλεψης διακλάδωσης χρησιμοποιώντας τη στρατηγική αυτή. Χρησιμοποιήθηκαν τα ίδια δεδομένα εισόδου τόσο για την εκτέλεση όσο και για τη συλλογή των πληροφοριών του προφίλ. Σύμφωνα με άλλες μελέτες, η αλλαγή των δεδομένων εισόδου, έτσι ώστε το προφίλ να αντιστοιχεί σε διαφορετική εκτέλεση, έχει ως αποτέλεσμα μια μικρή μόνο αλλαγή στην ακρίβεια της πρόβλεψης που βασίζεται στο προφίλ.

Η αποτελεσματικότητα κάθε προσέγγισης πρόβλεψης διακλάδωσης εξαρτάται τόσο από την ακρίβειά της όσο και από τη συχνότητα των διακλαδώσεων υπό συνθήκη, οι οποίες κυμαίνονται για το SPEC μεταξύ του 3% και του 24%. Το γεγονός ότι ο ρυθμός των εσφαλμένων προβλέψεων για τα προγράμματα ακεραίων είναι υψηλότερος, καθώς και το ότι τα προγράμματα αυτά συνήθως έχουν υψηλότερη συχνότητα διακλαδώσεων αποτελεί σημαντικό παράγοντα περιορισμού της στατικής πρόβλεψης διακλάδωσης. Στην επόμενη ενότητα, θα εξετάσουμε τους δυναμικούς μηχανισμούς πρόβλεψης διακλάδωσης, τους οποίους εφαρμόζουν οι πιο σύγχρονοι επεξεργαστές.

Δυναμική Πρόβλεψη Διακλάδωσης και Απομονωτές Πρόβλεψης Διακλάδωσης

Η πιο απλή προσέγγιση δυναμικής πρόβλεψης διακλάδωσης χρησιμοποιεί τον *απομονωτή πρόβλεψης διακλάδωσης* (branch prediction buffer) ή τον *πίνακα ιστορικού διακλάδωσης* (branch history table). Ο απομονωτής πρόβλεψης διακλάδωσης αποτελεί μικρή μνήμη, η οποία δεικτοδοτείται από το λιγότερο σημαντικό τμήμα της διεύθυνσης των εντολών διακλάδωσης. Η μνήμη περιέχει ένα bit το οποίο καταδεικνύει αν η διακλάδωση έχει πρόσφατα ακολουθηθεί ή όχι. Η προσέγγιση αυτή αποτελεί την πιο απλή μορφή απομονωτή. Δεν περι-

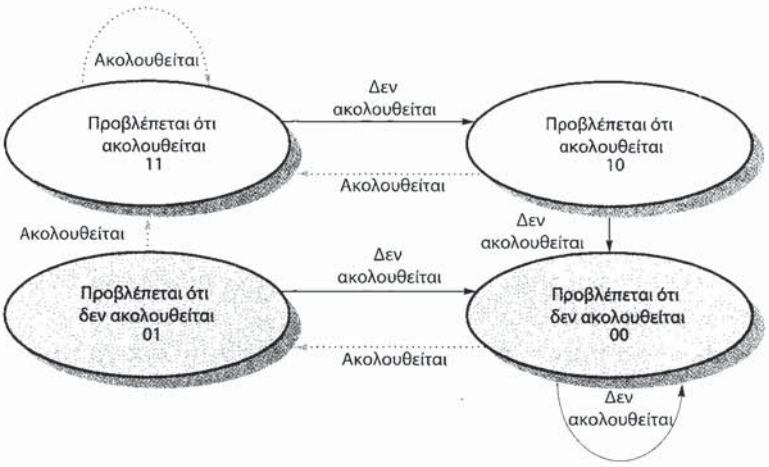


Σχήμα 2.3 Ο ρυθμός εσφαλμένων προβλέψεων του SPEC92 για μηχανισμούς πρόβλεψης που βασίζονται σε προφίλ διαφέρει αρκετά από περίπτωση σε περίπτωση, ωστόσο σε γενικές γραμμές είναι καλύτερος για προγράμματα FP, που έχουν μέσο ρυθμό εσφαλμένων προβλέψεων της τάξης του 9% και τυπική απόκλιση της τάξης του 4%, απ' ό,τι για προγράμματα ακεραίων, τα οποία έχουν μέσο ρυθμό εσφαλμένων προβλέψεων της τάξης του 15% και τυπική απόκλιση της τάξης του 5%. Η πραγματική απόδοση εξαρτάται τόσο από την ακρίβεια της πρόβλεψης όσο και από τη συχνότητα διακλάδωσης, οι οποίες κυμαίνονται μεταξύ του 4% και του 24%.

λαμβάνει ετικέτες και είναι χρήσιμη μόνο για τη μείωση της καθυστέρησης της διακλάδωσης, όταν αυτή χρειάζεται περισσότερο χρόνο απ' ό,τι ο υπολογισμός των πιθανών PCs προορισμού.

Σε αυτού του είδους τους απομονωτές, δεν γνωρίζουμε, στην πραγματικότητα, αν η πρόβλεψη είναι ορθή - μπορεί να έχει τοποθετηθεί εκεί από κάποια άλλη διακλάδωση η οποία έχει τα ίδια λιγότερο σημαντικά bits διεύθυνσης. Παρ' όλα αυτά, αυτό δεν αποτελεί πρόβλημα. Η πρόβλεψη είναι απλά μια υπόδειξη, για την οποία κάνουμε την υπόθεση ότι είναι ορθή και ως εκ τούτου η ανάκτηση ξεκινά από την προβλεπόμενη αυτή κατεύθυνση. Αν η υπόδειξη αποδειχθεί ότι είναι εσφαλμένη, τότε το bit πρόβλεψης λαμβάνει την αντίστροφη τιμή και αποθηκεύεται εκ νέου.

Ο απομονωτής αποτελεί στην πραγματικότητα κρυφή μνήμη, κάθε προσπέλαση της οποίας αποτελεί ευστοχία και, όπως θα διαπιστώσουμε, η απόδοση του απομονωτή εξαρτάται τόσο από το πόσο συχνά η πρόβλεψη αφορά τη διακλάδωση που μας ενδιαφέρει όσο και από το βαθμό ακρίβειας της πρόβλεψης, όταν όντως αφορά τη σωστή διακλάδωση. Πριν αναλύσουμε την απόδοση, είναι



Σχήμα 2.4 Οι καταστάσεις της προσέγγισης πρόβλεψης με χρήση 2 bits. Η χρησιμοποίηση 2 και όχι ενός bit έχει ως αποτέλεσμα τη λιγότερο συχνή εσφαλμένη πρόβλεψη των διακλαδώσεων, που είτε σχεδόν πάντα ακολουθούνται είτε σχεδόν πάντα δεν ακολουθούνται - όπως συμβαίνει στις περισσότερες περιπτώσεις - απ' ό,τι στην περίπτωση του μηχανισμού πρόβλεψης που χρησιμοποιεί 1 bit. Τα 2 bits χρησιμοποιούνται για την κωδικοποίηση των τεσσάρων καταστάσεων του συστήματος. Η προσέγγιση των 2 bits αποτελεί στην πραγματικότητα ειδίκευση μιας πιο γενικής προσέγγισης, η οποία περιλαμβάνει έναν μετρητή κορεσμού (saturating counter) των n bits για κάθε εγγραφή του απομονωτή πρόβλεψης. Οι μετρητές των n bits μπορούν να λάβουν τιμές μεταξύ του 0 και του $2^n - 1$: Όταν ο μετρητής είναι μεγαλύτερος ή ίσος με το μισό της μέγιστης τιμής του ($2^n - 1$), τότε η διακλάδωση προβλέπεται ότι ακολουθείται. Σε διαφορετική περίπτωση, η διακλάδωση προβλέπεται ότι είναι δεν ακολουθείται. Σύμφωνα με μελέτες που αφορούν τους μηχανισμούς πρόβλεψης των n bits, οι μηχανισμοί πρόβλεψης των 2 bits έχουν σχεδόν το ίδιο ικανοποιητικά αποτελέσματα και ως εκ τούτου τα περισσότερα συστήματα βασίζονται σε μηχανισμούς διακλάδωσης των 2 bits και όχι στους πιο γενικούς μηχανισμούς διακλάδωσης των n bits.

χρήσιμο να προβούμε σε μια μικρή, αλλά ιδιαίτερα σημαντική, βελτίωση της ακρίβειας της προσέγγισης της πρόβλεψης διακλάδωσης.

Αυτή η απλή προσέγγιση πρόβλεψης του ενός bit έχει ένα μειονέκτημα που αφορά την απόδοση: Ακόμα και αν η διακλάδωση ακολουθείται σχεδόν πάντα, πιθανότατα θα πραγματοποιείται λανθασμένη πρόβλεψη δύο φορές και όχι μόνο μία όταν και δεν ακολουθείται η διακλάδωση, καθώς η εσφαλμένη πρόβλεψη προκαλεί την αντιστροφή του bit πρόβλεψης.

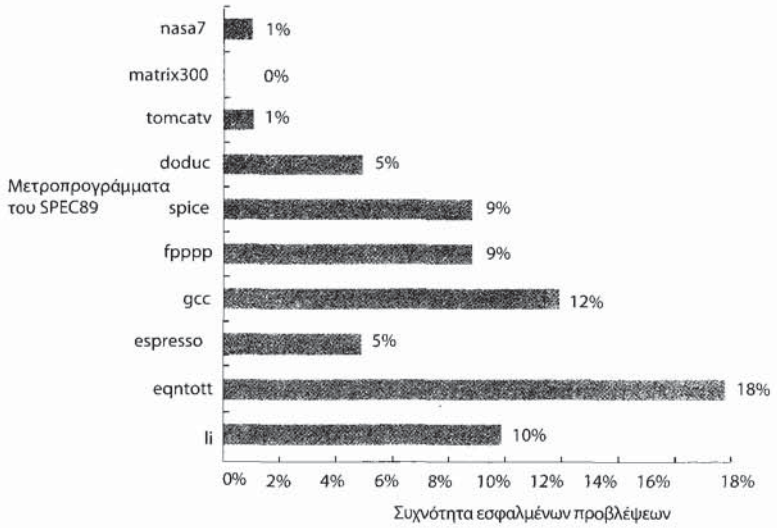
Για την αντιμετώπιση του μειονεκτήματος αυτού, συχνά χρησιμοποιούνται προσεγγίσεις πρόβλεψης με χρήση 2 bits. Στην προσέγγιση των 2 bits, η πρόβλεψη πρέπει να αποδειχθεί λανθασμένη δύο φορές, προτού να αλλάξει. Το Σχήμα 2.4 παρουσιάζει τον επεξεργαστή πεπερασμένων καταστάσεων της προσέγγισης πρόβλεψης με χρήση 2 bits.

Οι απομονωτές πρόβλεψης διακλάδωσης μπορούν να υλοποιηθούν είτε ως

μικρή, ειδική «κρυφή μνήμη», η οποία προσπελάζεται με τη βοήθεια της διεύθυνσης της εντολής κατά το στάδιο IF της σωλήνωσης είτε ως ζεύγος από bits τα οποία επισυνάπτονται σε κάθε block της κρυφής μνήμης εντολών και ανακτώνται μαζί με κάθε εντολή. Αν κατά την αποκωδικοποίηση της εντολής αποδειχθεί ότι πρόκειται για διακλάδωση, καθώς και αν η διακλάδωση προβλέπεται ότι ακολουθείται, τότε, μόλις γίνει γνωστός ο PC, ξεκινά η ανάκτηση από την εντολή προορισμού. Σε διαφορετική περίπτωση, συνεχίζεται η ακολουθιακή ανάκτηση και εκτέλεση. Όπως φαίνεται στο Σχήμα 2.4, αν η πρόβλεψη αποδειχθεί ότι είναι εσφαλμένη, τότε αλλάζουν τα bits πρόβλεψης.

Τι είδους ακρίβεια μπορεί να αναμένει κανείς, για τις πραγματικές εφαρμογές, από τη χρήση του απομονωτή πρόβλεψης διακλάδωσης που χρησιμοποιεί 2 bits ανά καταχώριση (entry); Σύμφωνα με το Σχήμα 2.5, όσον αφορά τα μετροπρογράμματα του SPEC89, η χρήση απομονωτή πρόβλεψης διακλάδωσης που διαθέτει 4096 καταχωρίσεις έχει ως αποτέλεσμα ακρίβεια πρόβλεψης η οποία κυμαίνεται ανάμεσα στο 99% και το 82% ή *ρυθμό εσφαλμένων προβλέψεων* (misprediction rate) που κυμαίνεται ανάμεσα στο 1% και το 18%. Σύμφωνα με τα κριτήρια του 2005, οι απομονωτές που διαθέτουν 4K καταχωρίσεις, όπως αυτός που χρησιμοποιήθηκε για αυτά τα αποτελέσματα, θεωρούνται μικροί, καθώς οι μεγαλύτεροι απομονωτές μπορούν να αποφέρουν, σε κάποιο βαθμό, καλύτερα αποτελέσματα.

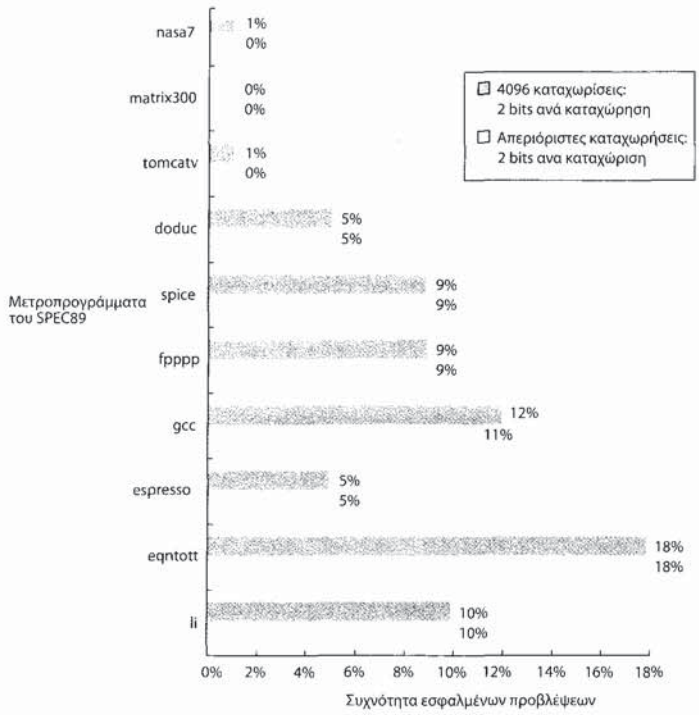
Όσο επιχειρούμε να αξιοποιούμε ολοένα και περισσότερο τον παραλληλισμό, η ακρίβεια της πρόβλεψης διακλάδωσης καθίσταται όλο και πιο σημαντική. Όπως μπορεί να διαπιστώσει κανείς εξετάζοντας το Σχήμα 2.5, η ακρίβεια των μηχανισμών πρόβλεψης για τα προγράμματα ακεραίων, τα οποία συνήθως έχουν, επίσης, υψηλότερη συχνότητα διακλαδώσεων, είναι χαμηλότερη από αυτή των επιστημονικών προγραμμάτων που χρησιμοποιούν εντατικά βρόχους. Μπορούμε να αντιμετωπίσουμε το πρόβλημα αυτό με δύο τρόπους: μέσω αύξησης του μεγέθους του απομονωτή και διαμέσου αύξησης της ακρίβειας της προσέγγισης που χρησιμοποιούμε για κάθε πρόβλεψη. Οι απομονωτές που διαθέτουν 4K καταχωρίσεις, ωστόσο, όπως φαίνεται στο Σχήμα 2.6, έχουν απόδοση συγκρίσιμη με αυτή των απομονωτών απεριόριστου μεγέθους, τουλάχιστον όσον αφορά μετροπρογράμματα, όπως αυτά του SPEC. Τα δεδομένα του Σχήματος 2.6 καθιστούν σαφές ότι ο ρυθμός ευστοχίας (hit rate) του απομονωτή δεν αποτελεί τον κύριο περιοριστικό παράγοντα. Όπως αναφέρθηκε και προηγουμένως, η απλή αύξηση του αριθμού των bits ανά μηχανισμό πρόβλεψης χωρίς την παράλληλη αλλαγή της δομής του μηχανισμού αυτού έχει, επίσης, χαμηλό αντίκτυπο. Αντιθέτως, χρειάζεται να εξετάσουμε τον τρόπο με τον οποίο μπορούμε να αυξήσουμε την ακρίβεια κάθε μηχανισμού πρόβλεψης.



Σχήμα 2.5 Ακρίβεια πρόβλεψης του απομονωτή που περιλαμβάνει 4096 καταχωρίσεις των 2 bits για τα μετροπρογράμματα του SPEC89. Ο ρυθμός εσφαλμένων προβλέψεων των μετροπρογραμμάτων ακεραίων (gcc, espresso, eqntott και li) είναι σημαντικά υψηλότερος (με μέσο όρο της τάξης του 11%) απ' ό,τι ο αντίστοιχος ρυθμός των προγραμμάτων FP (με μέσο όρο της τάξης του 4%). Ακόμα και αν παραλείψουμε τους πυρήνες FP (nasa7, matrix300 και tomcatv), τα μετροπρογράμματα FP εξακολουθούν να έχουν υψηλότερη ακρίβεια απ' ό,τι τα μετροπρογράμματα ακεραίων. Τα δεδομένα αυτά, καθώς και τα υπόλοιπα δεδομένα της ενότητας αυτής, έχουν ληφθεί από μελέτη, σχετική με την πρόβλεψη διακλάδωσης, που εκπονήθηκε χρησιμοποιώντας την αρχιτεκτονική IBM Power, καθώς και βελτιστοποιημένο κώδικα για το σύστημα αυτό (ανατρέξτε στους Pan, So και Rameh [1992]). Παρόλο που τα δεδομένα αυτά αφορούν παλαιότερη έκδοση του υποσυνόλου των μετροπρογραμμάτων του SPEC, τα νεότερα μετροπρογράμματα και ιδίως τα μετροπρογράμματα ακεραίων, είναι μεγαλύτερα και θα παρουσίαζαν ελαφρά χειρότερη συμπεριφορά.

Συσχετικοί Μηχανισμοί Πρόβλεψης Διακλάδωσης

Οι προσεγγίσεις των μηχανισμών πρόβλεψης των 2 bits αξιοποιούν μόνο την πρόσφατη συμπεριφορά μίας μονάχα διακλάδωσης για την πρόγνωση της μελλοντικής συμπεριφοράς της διακλάδωσης αυτής. Μπορεί να είναι εφικτή η βελτίωση της ακρίβειας της πρόβλεψης, αν εξετάσουμε, επίσης, την πρόσφατη συμπεριφορά *άλλων* διακλαδώσεων και όχι μόνο τη συμπεριφορά της διακλάδωσης που επιχειρούμε να προβλέψουμε. Ας εξετάσουμε ένα μικρό τμήμα κώδικα που προέρχεται από το μετροπρόγραμμα eqntott, μέλος των πρώτων σουιτών μετροπρογραμμάτων του SPEC, το οποίο επεδείκνυε ιδιαίτερα άσχημη συμπεριφορά όσον αφορά την πρόβλεψη διακλάδωσης:



Σχήμα 2.6 Ακρίβεια πρόβλεψης του απομονωτή πρόβλεψης των 2 bits και των 4096 καταχωρίσεων σε σύγκριση με τον απομονωτή απεριόριστου μεγέθους για τα μετροπρογράμματα του SPEC89. Παρόλο που τα δεδομένα αυτά αφορούν παλαιότερη έκδοση του υποσυνόλου των μετροπρογραμμάτων του SPEC, τα αποτελέσματα θα ήταν συγκρίσιμα με αυτά των νεότερων εκδόσεων, σύμφωνα με τα οποία οι απομονωτές με 8K καταχωρίσεις θα μπορούσαν να έχουν παρόμοια αποτελέσματα με τους απομονωτές απεριόριστου μεγέθους των 2 bits.

```
if (aa==2)
    aa=0;
if (bb==2)
    bb=0;
if (aa!=bb) {
```

Παρακάτω παρατίθεται ο κώδικας της MIPS που παράγουμε για το παραπάνω τμήμα κώδικα, με την προϋπόθεση ότι οι μεταβλητές aa και bb ανατίθενται στους καταχωρητές R1 και R2:

```
        DADDIU  R3,R1,#-2
        BNEZ    R3,L1          ;διακλάδωση b1 (aa!=2)
        DADD    R1,R0,R0       ;aa=0
L1:     DADDIU  R3,R2,#-2
```

```
        BNEZ    R3,L2          ;διακλάδωση b2  (bb!=2)
        DADD    R2,R0,R0       ;bb=0
L2:     DSUBU   R3,R1,R2       ;R3=aa-bb
        BEQZ    R3,L3          ;διακλάδωση b3  (aa==bb)
```

Ας δώσουμε στις διακλαδώσεις αυτές τις ονομασίες b1, b2 και b3. Το κύριο στοιχείο που παρατηρεί κανείς είναι το γεγονός ότι η συμπεριφορά της διακλάδωσης b3 είναι συσχετισμένη με τη συμπεριφορά των διακλαδώσεων b1 και b2. Όπως είναι εμφανές, αν δεν ακολουθηθεί καμία από τις διακλαδώσεις b1 και b2 (δηλ. αν και οι δύο συνθήκες αξιολογηθούν ως αληθείς και ανατεθεί τόσο στη μεταβλητή aa όσο και στην bb η τιμή 0), τότε θα ακολουθηθεί η b3, καθώς οι aa και bb είναι προφανώς ίσες. Οι μηχανισμοί πρόβλεψης που αξιοποιούν τη συμπεριφορά μόνο της μίας διακλάδωσης για την πρόβλεψη του αποτελέσματος της διακλάδωσης αυτής δεν μπορούν ποτέ να καταγράψουν αυτήν τη συμπεριφορά.

Οι μηχανισμοί πρόβλεψης διακλάδωσης που αξιοποιούν τη συμπεριφορά άλλων διακλαδώσεων για την πραγματοποίηση των προβλέψεών τους ονομάζονται *συσχετικοί μηχανισμοί πρόβλεψης* (correlating predictors) ή *μηχανισμοί πρόβλεψης δύο επιπέδων* (two-level predictors). Οι υφιστάμενοι συσχετικοί μηχανισμοί πρόβλεψης, προκειμένου να λάβουν απόφαση σχετικά με την πρόβλεψη κάποιας διακλάδωσης, αξιοποιούν πληροφορίες που αφορούν τη συμπεριφορά των πιο πρόσφατων διακλαδώσεων. Για παράδειγμα, για την πρόβλεψη μιας συγκεκριμένης διακλάδωσης, οι μηχανισμοί πρόβλεψης $(1,2)$ αξιοποιούν τη συμπεριφορά της τελευταίας διακλάδωσης για να επιλέξουν έναν από τους δύο διαθέσιμους μηχανισμούς πρόβλεψης διακλάδωσης. Γενικά, οι μηχανισμοί πρόβλεψης (m,n) αξιοποιούν τη συμπεριφορά των τελευταίων m διακλαδώσεων για να επιλέξουν έναν από τους 2^m διαθέσιμους μηχανισμούς πρόβλεψης, καθένας εκ των οποίων αποτελεί μηχανισμό πρόβλεψης μίας μόνο διακλάδωσης ο οποίος χρησιμοποιεί n bits. Η ελκυστικότητα αυτού του τύπου των συσχετικών μηχανισμών πρόβλεψης διακλάδωσης έγκειται στο γεγονός ότι μπορούν να σημειώσουν υψηλότερους ρυθμούς ακρίβειας απ' ό,τι οι προσεγγίσεις των 2 bits, χωρίς να απαιτούνται ιδιαίτερα σημαντικές προσθήκες στο υλικό.

Η απλότητα του υλικού οφείλεται σε μια απλή παρατήρηση: Το καθολικό ιστορικό (global history) των πιο πρόσφατων m διακλαδώσεων μπορεί να καταγραφεί σε καταχωρητή ολίσθησης των m bits, στον οποίο σε κάθε bit καταγράφεται αν η διακλάδωση έχει ακολουθηθεί ή όχι. Ο απομονωτής πρόβλεψης διακλάδωσης μπορεί στη συνέχεια να δεικτοδοτηθεί, συνενώνοντας τα λιγότερο σημαντικά bits της διεύθυνσης διακλάδωσης με το καθολικό ιστορικό των m bits. Για παράδειγμα, στους απομονωτές $(2,2)$ που διαθέτουν 64 συνολικά καταχωρίσεις, τα 4 λιγότερο σημαντικά bits της διεύθυνσης της διακλάδωσης (διεύθυνσης λέξης) και τα 2 καθολικά bits, τα οποία αναπαριστούν τη συμπεριφορά των δύο πιο πρόσφατα εκτελεσμένων διακλαδώσεων, σχηματίζουν δείκτη των 6 bits, ο οποίος μπορεί να χρησιμοποιηθεί για τη δεικτοδότηση των 64 μετρητών.

Πόσο καλύτερα αποδίδουν οι συσχετικοί μηχανισμοί πρόβλεψης διακλάδωσης σε σύγκριση με την τυποποιημένη προσέγγιση των 2 bits; Προκειμένου η διαδικασία αυτή να είναι αξιόπιστη, πρέπει να συγκρίνουμε μηχανισμούς πρόβλεψης οι οποίοι χρησιμοποιούν τον ίδιο αριθμό bits κατάστασης. Ο αριθμός των bits στους μηχανισμούς πρόβλεψης (m, n) ισούται με

$$2^m \times n \times \text{Πλήθος καταχωρίσεων πρόβλεψης που επιλέγονται από τη διεύθυνση διακλάδωσης}$$

Οι μηχανισμοί πρόβλεψης των 2 bits που δεν χρησιμοποιούν καθολικό ιστορικό είναι απλά μηχανισμοί $(0, 2)$.

Παράδειγμα: Πόσα bits έχουν οι μηχανισμοί πρόβλεψης διακλάδωσης $(0, 2)$ που διαθέτουν 4K καταχωρίσεις; Πόσες καταχωρίσεις διαθέτουν οι μηχανισμοί πρόβλεψης $(2, 2)$ που έχουν τον ίδιο αριθμό bits;

Απάντηση: Οι μηχανισμοί πρόβλεψης που διαθέτουν 4K καταχωρίσεις έχουν

$$2^0 \times 2 \times 4\text{K} = 8\text{K bits}$$

Πόσες επιλεγμένες από τη διακλάδωση καταχωρίσεις διαθέτουν οι μηχανισμοί πρόβλεψης $(2, 2)$ που έχουν συνολικά 8K bits στον απομονωτή πρόβλεψης; Γνωρίζουμε ότι

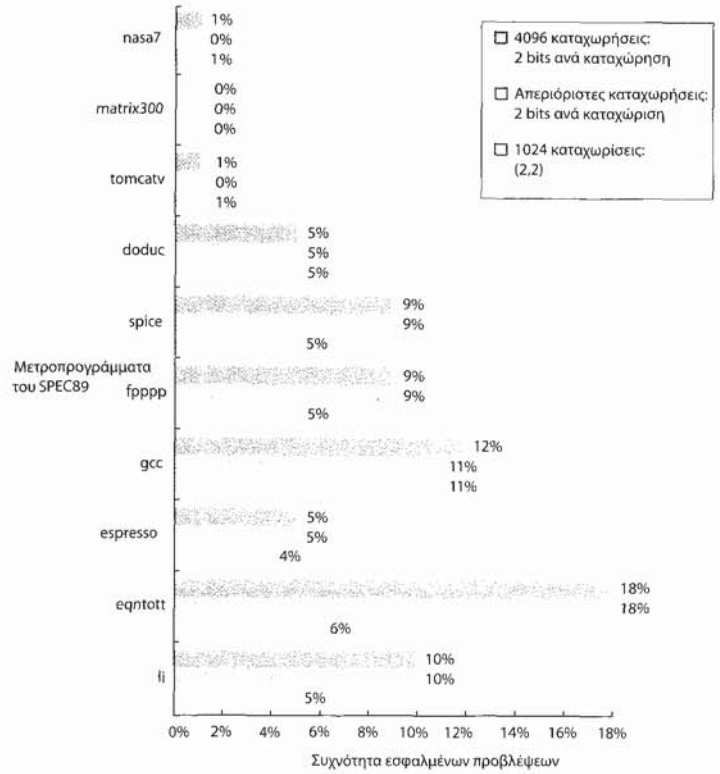
$$2^2 \times 2 \times \text{Πλήθος καταχωρίσεων πρόβλεψης που επιλέγονται από τη διακλάδωση} = 8\text{K bits}$$

Επομένως, ο αριθμός των καταχωρίσεων πρόβλεψης που επιλέγονται από τη διακλάδωση ισούται με 1K.

Το Σχήμα 2.7 συγκρίνει το ρυθμό εσφαλμένων προβλέψεων του προηγούμενου μηχανισμού πρόβλεψης $(0, 2)$ που διαθέτει 4K καταχωρίσεις με το μηχανισμό πρόβλεψης $(2, 2)$ που διαθέτει 1K καταχωρίσεις. Όπως μπορεί να διαπιστώσει κανείς, αυτός ο συσχετικός μηχανισμός πρόβλεψης, όχι μόνο έχει υψηλότερη απόδοση από τον απλό μηχανισμό πρόβλεψης των 2 bits που διαθέτει τον ίδιο συνολικό αριθμό από bits κατάστασης, αλλά συχνά έχει υψηλότερη απόδοση από το μηχανισμό πρόβλεψης των 2 bits που διαθέτει απεριόριστο αριθμό καταχωρίσεων.

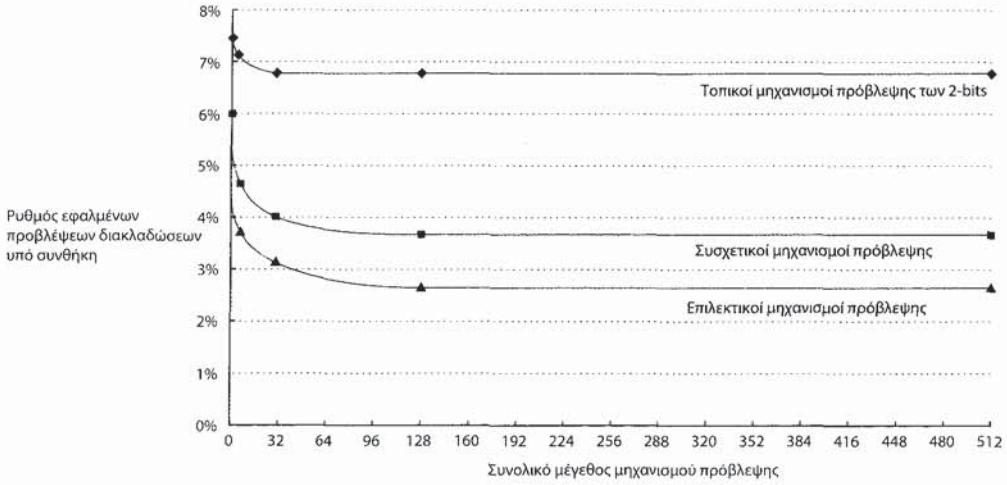
Επιλεκτικοί Μηχανισμοί Πρόβλεψης: Προσαρμοζόμενος Συνδυασμός Τοπικών και Καθολικών Μηχανισμών Πρόβλεψης

Αυτό που αποτέλεσε το βασικό παράγοντα δημιουργίας των συσχετικών μηχανισμών πρόβλεψης διακλάδωσης είναι το γεγονός ότι οι τυπικοί μηχανισμοί των 2 bits που χρησιμοποιούν μόνο τοπικές πληροφορίες (local information) δεν σημείωναν επιτυχία σε μερικές περιπτώσεις σημαντικών διακλαδώσεων, καθώς



Σχήμα 2.7 Σύγκριση των μηχανισμών πρόβλεψης των 2 bits. Πρώτος στη λίστα βρίσκεται ένας μη συσχετικός μηχανισμός πρόβλεψης των 4096 bits, ακολουθεί ένας μη συσχετικός μηχανισμός πρόβλεψης των 2 bits που διαθέτει απεριόριστο αριθμό καταχωρίσεων και τελευταίος βρίσκεται ένας μηχανισμός πρόβλεψης των 2 bits που διαθέτει καθολικό ιστορικό των 2 bits και 1024 καταχωρίσεις. Παρόλο που τα δεδομένα αυτά αφορούν παλαιότερη έκδοση του SPEC, τα δεδομένα που αφορούν τις νεότερες εκδόσεις του παρουσιάζουν τις ίδιες διαφορές σε επίπεδο ακρίβειας των μετροπρογραμμάτων.

και το ότι η προσθήκη καθολικών πληροφοριών (global information) οδηγεί στη βελτίωση της απόδοσης. Οι *επιλεκτικοί μηχανισμοί πρόβλεψης* (tournament predictors) αξιοποιούν ακόμα περισσότερο το παραπάνω φαινόμενο, χρησιμοποιώντας πολλαπλούς μηχανισμούς πρόβλεψης, συνήθως έναν με βάση τις καθολικές πληροφορίες και έναν με βάση τις τοπικές πληροφορίες, τους οποίους και συνδυάζουν με τη βοήθεια του μηχανισμού επιλογής (selector). Οι επιλεκτικοί μηχανισμοί πρόβλεψης μπορούν να σημειώνουν υψηλότερη απόδοση για μεσαία μεγέθη (της τάξης των 8K - 32K bits), καθώς και να χρησιμοποιούν αποτελεσματικά πολύ μεγάλο αριθμό bits πρόβλεψης. Οι υφιστάμενοι επιλεκτικοί μηχανισμοί πρόβλεψης χρησιμοποιούν ένα μετρητή κορεσμού των 2 bits ανά διακλάδωση, προκειμένου να επιλέξουν έναν από τους δύο διαφορετικούς μηχανισμούς πρόβλεψης με βάση το βαθμό αποτελεσματικότητας καθενός από



Σχήμα 2.8 Ρυθμός εσφαλμένων προβλέψεων τριών διαφορετικών μηχανισμών πρόβλεψης για το SPEC89 όσο αυξάνεται ο συνολικός αριθμός των bits. Στο σχήμα παρατίθεται ένας τοπικός μηχανισμός πρόβλεψης των 2 bits, ένας συσχετικός μηχανισμός πρόβλεψης, ο οποίος διαθέτει τη βέλτιστη δομή για τη χρήση των καθολικών και τοπικών πληροφοριών σε κάθε σημείο του γραφήματος, καθώς και ένας επιλεκτικός μηχανισμός πρόβλεψης. Παρόλο που τα δεδομένα αυτά αφορούν παλαιότερη έκδοση του SPEC, τα δεδομένα των πιο πρόσφατων μετροπρογραμμάτων του θα παρουσίαζαν παρόμοια συμπεριφορά, πιθανότατα συγκλίνοντας στο ασυμπτωτικό όριο σε ελαφρά μεγαλύτερα μεγέθη μηχανισμών πρόβλεψης.

αυτούς (τοπικού, καθολικού ή ακόμα και κάποιου συνδυασμού τους) σε ό,τι αφορά τις πιο πρόσφατες προβλέψεις τους. Όπως στην περίπτωση των απλών μηχανισμών πρόβλεψης των 2 bits, έτσι και σε αυτήν την περίπτωση ο μετρητής κορεσμού απαιτεί την ύπαρξη δύο εσφαλμένων προβλέψεων πριν την αλλαγή της ταυτότητας του επιλεγμένου μηχανισμού πρόβλεψης.

Το πλεονέκτημα των επιλεκτικών μηχανισμών πρόβλεψης έγκειται στην ικανότητά τους να επιλέγουν τον ορθό μηχανισμό πρόβλεψης μιας συγκεκριμένης διακλάδωσης, κάτι που είναι ιδιαίτερα σημαντικό για τα μετροπρογράμματα ακεραίων. Ο συνήθης επιλεκτικός μηχανισμός πρόβλεψης θα επιλέξει τον καθολικό μηχανισμό πρόβλεψης στο 40% των περιπτώσεων για τα μετροπρογράμματα ακεραίων του SPEC και σε λιγότερο από το 15% των περιπτώσεων για τα μετροπρογράμματα FP του SPEC.

Το Σχήμα 2.8 εξετάζει την απόδοση τριών διαφορετικών μηχανισμών πρόβλεψης (ενός τοπικού μηχανισμού πρόβλεψης των 2 bits, ενός συσχετικού και ενός επιλεκτικού μηχανισμού πρόβλεψης) για διαφορετικό αριθμό bits, χρησιμοποιώντας ως μετροπρόγραμμα το SPEC89. Όπως έχουμε ήδη διαπιστώσει, η ικανότητα πρόβλεψης του τοπικού μηχανισμού πρόβλεψης δεν βελτιώνεται από

ένα σημείο και πέρα. Ο συσχετικός μηχανισμός πρόβλεψης επιφέρει σημαντική βελτίωση, ενώ ο επιλεκτικός μηχανισμός πρόβλεψης παρουσιάζει ελαφρά υψηλότερη απόδοση. Στις νεότερες εκδόσεις του SPEC, τα αποτελέσματα είναι παρόμοια, ωστόσο δεν επιτυγχάνεται ασυμπτωτική συμπεριφορά αν δεν χρησιμοποιηθούν μηχανισμοί πρόβλεψης μεγαλύτερου μεγέθους.

Το 2005, οι επεξεργαστές, όπως οι Power5 και Pentium 4, διέθεταν επιλεκτικούς μηχανισμούς πρόβλεψης που χρησιμοποιούσαν περίπου 30K bits. Οι πιο εξελιγμένοι μηχανισμοί πρόβλεψης είναι αυτοί του Alpha 21264, παρόλο που οι μηχανισμοί πρόβλεψης τόσο του Pentium 4 όσο και του Power5 είναι παρόμοιοι. Ο επιλεκτικός μηχανισμός πρόβλεψης του 21264 χρησιμοποιεί 4K μετρητές των 2 bits, οι οποίοι δεικτοδοτούνται με τη βοήθεια της τοπικής διεύθυνσης διακλάδωσης, έτσι ώστε να πραγματοποιείται η επιλογή ανάμεσα σε έναν καθολικό και σε έναν τοπικό μηχανισμό πρόβλεψης. Ο καθολικός μηχανισμός πρόβλεψης διαθέτει, επίσης, 4K καταχωρίσεις και δεικτοδοτείται με τη βοήθεια του ιστορικού των 12 τελευταίων διακλαδώσεων. Κάθε καταχώριση του καθολικού μηχανισμού πρόβλεψης αποτελεί τυπικό μηχανισμό πρόβλεψης των 2 bits.

Ο τοπικός μηχανισμός πρόβλεψης αποτελείται από ένα μηχανισμό δύο επιπέδων. Το πρώτο επίπεδο αποτελεί τοπικό πίνακα ιστορικού που περιλαμβάνει 1024 καταχωρίσεις των 10 bits. Κάθε καταχώριση των 10 bits αντιστοιχεί στα 10 πιο πρόσφατα αποτελέσματα της διακλάδωσης της καταχώρισης αυτής. Δηλαδή, αν η διακλάδωση έχει ακολουθηθεί 10 ή περισσότερες φορές στη σειρά, τότε η καταχώριση στον τοπικό πίνακα ιστορικού θα αποτελείται μόνο από μονάδες. Αν η διακλάδωση τη μία φορά ακολουθείται και την επόμενη δεν ακολουθείται, τότε η καταχώριση του πίνακα ιστορικού περιλαμβάνει εναλλασσόμενα μηδενικά και μονάδες. Αυτό το ιστορικό των 10 bits επιτρέπει την ανακάλυψη και πρόβλεψη προτύπων που περιλαμβάνουν έως και δέκα διακλαδώσεις. Η επιλεγμένη καταχώριση του τοπικού πίνακα ιστορικού χρησιμοποιείται για τη δεικτοδότηση ενός πίνακα με 1K καταχωρίσεις που περιλαμβάνουν μετρητές κορεσμού των 3 bits, οι οποίοι παρέχουν την τοπική πρόβλεψη. Ο συνδυασμός αυτός, που χρησιμοποιεί συνολικά 29K bits, οδηγεί σε υψηλή ακρίβεια των προβλέψεων διακλάδωσης.

Για να εξετάσουμε την επίδραση της παραπάνω προσέγγισης στην απόδοση, χρειάζεται να γνωρίζουμε τόσο την ακρίβεια της πρόβλεψης όσο και τη συχνότητα των διακλαδώσεων, καθώς η ακριβής πρόβλεψη είναι πιο σημαντική για τα προγράμματα που διαθέτουν υψηλότερη συχνότητα διακλαδώσεων. Για παράδειγμα, τα προγράμματα ακεραίων της σουίτας του SPEC έχουν υψηλότερη συχνότητα διακλαδώσεων απ' ό,τι τα πιο εύκολα, όσον αφορά τη δυνατότητα πρόβλεψής τους, προγράμματα FP. Στο μηχανισμό πρόβλεψης του 21264, τα μετροπρογράμματα του SPECfp95 έχουν λιγότερο από μία εσφαλμένη πρόβλεψη ανά 1000 ολοκληρωμένες εντολές, ενώ τα μετροπρογράμματα του SPECint95 έχουν περίπου 11.5 εσφαλμένες προβλέψεις ανά 1000 ολοκληρωμένες εντολές. Αυτά αντιστοιχούν σε ρυθμούς εσφαλμένης πρόβλεψης της τάξης του 0.5% για

τα προγράμματα κινητής υποδιαστολής και του 14% για τα προγράμματα ακεραίων.

Μεταγενέστερες εκδόσεις του SPEC περιλαμβάνουν προγράμματα με μεγαλύτερα σύνολα δεδομένων και περισσότερο κώδικα, κάτι που έχει ως αποτέλεσμα υψηλότερους ρυθμούς αστοχίας. Συνεπώς, η σπουδαιότητα της πρόβλεψης των διακλαδώσεων έχει αυξηθεί. Στην Ενότητα 2.11, θα εξετάσουμε την απόδοση του μηχανισμού πρόβλεψης του Pentium 4 με τη βοήθεια των προγραμμάτων της σουίτας SPEC2000 και θα διαπιστώσουμε ότι, παρά το γεγονός ότι εφαρμόζεται πιο εντατική πρόβλεψη διακλάδωσης, οι ρυθμοί αστοχίας των προβλέψεων διακλάδωσης για τα προγράμματα ακεραίων εξακολουθούν να είναι σημαντικοί.

2.4 Αντιμετώπιση των Κινδύνων Δεδομένων με Δυναμική Χρονοδρομολόγηση

Η απλή διασωλήνωση που χρονοδρομολογείται στατικά ανακτά μια εντολή και την εκδίδει, εκτός και αν υφίσταται κάποια εξάρτηση δεδομένων ανάμεσα σε μια εντολή που βρίσκεται ήδη εντός της διασωλήνωσης και την ανακτώμενη εντολή, εξάρτηση που δεν μπορεί να εξαλειφθεί μέσω προώθησης ή παράκαμψης (η λογική προώθησης μειώνει τον ενεργό λανθάνοντα χρόνο της διασωλήνωσης, έτσι ώστε συγκεκριμένες εξαρτήσεις να μην οδηγούν σε κινδύνους). Αν υπάρχει κάποια εξάρτηση δεδομένων η οποία δεν μπορεί να πάψει να υφίσταται, τότε το υλικό ανίχνευσης κινδύνων θέτει τη διασωλήνωση σε αδράνεια ξεκινώντας με την εντολή που χρησιμοποιεί το αποτέλεσμα. Καμία νέα εντολή δεν ανακτάται ή εκδίδεται, μέχρις ότου να πάψει να υφίσταται η εξάρτηση.

Στην ενότητα αυτή, εξετάζουμε τη *δυναμική χρονοδρομολόγηση* (dynamic scheduling), κατά την οποία το υλικό αναδιατάσσει την εκτέλεση των εντολών για τη μείωση των διαστημάτων αδράνειας, διατηρώντας παράλληλα τη ροή των δεδομένων και τη συμπεριφορά των εξαιρέσεων. Η δυναμική χρονοδρομολόγηση διαθέτει αρκετά πλεονεκτήματα: Παρέχει τη δυνατότητα αντιμετώπισης κάποιων περιπτώσεων, στις οποίες οι εξαρτήσεις δεν είναι γνωστές κατά το χρόνο μεταγλώττισης (για παράδειγμα, επειδή μπορεί να περιλαμβάνουν μια αναφορά μνήμης) και συγχρόνως απλοποιεί το μεταγλωττιστή. Εξίσου σημαντικό στοιχείο είναι ίσως το γεγονός ότι επιτρέπει στον επεξεργαστή να είναι ανεκτικός σε μη αναμενόμενες καθυστερήσεις, όπως οι αστοχίες της κρυφής μνήμης, εκτελώντας άλλα τμήματα κώδικα την ώρα που αναμένει την επίλυση της αστοχίας. Στην Ενότητα 2.6, θα διερευνήσουμε την εικασία υλικού (hardware speculation), που αποτελεί τεχνική με σημαντικά πλεονεκτήματα απόδοσης, πάνω στα οποία εδράζεται η δυναμική χρονοδρομολόγηση. Όπως θα διαπιστώσουμε, η δυναμική χρονοδρομολόγηση διαθέτει πλεονεκτήματα, τα οποία για να τα καρπωθεί κανείς πρέπει να αυξήσει σημαντικά την πολυπλοκότητα του υλικού.

Οι επεξεργαστές που χρονοδρομολογούνται δυναμικά, παρόλο που δεν μπο-

ρούν να τροποποιήσουν τη ροή των δεδομένων, προσπαθούν να αποφύγουν την εισαγωγή διαστημάτων αδράνειας όταν υφίστανται εξαρτήσεις. Αντιθέτως, η στατική χρονοδρομολόγηση της διασωλήνωσης που πραγματοποιείται από το μεταγλωττιστή (η οποία καλύπτεται στην Ενότητα 2.2) επιχειρεί να ελαχιστοποιήσει τα διαστήματα αδράνειας, προβαίνοντας στο διαχωρισμό των εξαρτώμενων εντολών, έτσι ώστε αυτές να μην οδηγούν στην εμφάνιση κινδύνων. Βεβαίως, η χρονοδρομολόγηση της διασωλήνωσης από το μεταγλωττιστή μπορεί, επίσης, να χρησιμοποιηθεί σε κώδικα που προορίζεται για εκτέλεση σε επεξεργαστές που υποστηρίζουν διασωλήνωση δυναμικής χρονοδρομολόγησης.

Δυναμική Χρονοδρομολόγηση: Η Ιδέα

Ένας από τους βασικούς περιορισμούς των απλών τεχνικών διασωλήνωσης είναι το γεγονός ότι εφαρμόζουν έκδοση και εκτέλεση εντολών εντός σειράς (in-order): Οι εντολές εκδίδονται με βάση τη σειρά προγράμματος και στην περίπτωση που κάποια εντολή τεθεί σε αδράνεια εντός της διασωλήνωσης, τότε δεν μπορεί να προχωρήσει καμία επόμενη εντολή. Συνεπώς, αν υφίσταται κάποια εξάρτηση ανάμεσα σε δύο εντολές οι οποίες βρίσκονται σε κοντινή απόσταση εντός της διασωλήνωσης, τότε αυτό θα έχει ως αποτέλεσμα την εμφάνιση κινδύνου και την εισαγωγή διαστήματος αδράνειας. Αν υπάρχουν πολλαπλές λειτουργικές μονάδες, τότε οι μονάδες αυτές μπορεί να παραμένουν αδρανείς. Αν η εντολή j εξαρτάται από την εντολή μακράς διάρκειας εκτέλεσης i, η οποία εκτελείται την τρέχουσα χρονική στιγμή στη διασωλήνωση, τότε όλες οι εντολές που έπονται της j πρέπει να τεθούν σε αδράνεια, μέχρις ότου να ολοκληρωθεί η i και να μπορεί να εκτελεστεί η j. Για παράδειγμα, ας εξετάσουμε τον παρακάτω κώδικα:

```
DIV.D    F0,F2,F4
ADD.D    F10,F0,F8
SUB.D    F12,F8,F14
```

Η εντολή SUB.D δεν μπορεί να εκτελεστεί, διότι η εξάρτηση της ADD.D από την DIV.D προκαλεί την αδράνεια της διασωλήνωσης. Ωστόσο, η SUB.D δεν έχει εξάρτηση δεδομένων από καμία άλλη εντολή της διασωλήνωσης. Ο κίνδυνος αυτός εισάγει περιορισμούς σε επίπεδο απόδοσης, οι οποίοι μπορούν να εξαλειφθούν αν δεν υφίσταται η απαίτηση της εκτέλεσης των εντολών με τη σειρά προγράμματος.

Στην κλασική διασωλήνωση των πέντε σταδίων, τόσο οι δομικοί κίνδυνοι όσο και οι κίνδυνοι δεδομένων είναι εφικτό να ελέγχονται κατά τη διάρκεια της αποκωδικοποίησης εντολής (instruction decode, ID): Όταν η εντολή μπορεί να εκτελεσιεί χωρίς να προκύπτουν κίνδυνοι, τότε εκδίδεται κατά το στάδιο ID έχοντας γνώση ότι έχουν επιλυθεί όλοι οι κίνδυνοι δεδομένων.

Στο παραπάνω παράδειγμα, για να υπάρξει η δυνατότητα έναρξης της εκτέλεσης της SUB.D, πρέπει να διαχωρίσουμε τη διαδικασία έκδοσης σε δύο

μέρη: στον έλεγχο για κάθε είδους δομικό κίνδυνο και στην αναμονή για την απουσία κινδύνου δεδομένων. Επομένως, εξακολουθούμε να χρησιμοποιούμε την έκδοση εντολών εντός σειράς (δηλ. οι εντολές εκδίδονται με βάση τη σειρά προγράμματος), ωστόσο θέλουμε η εκτέλεση των εντολών να ξεκινά μόλις είναι διαθέσιμα τα ορίσματα των δεδομένων τους. Αυτού του είδους η διασωλήνωση προβαίνει σε *εκτέλεση εκτός σειράς* (out-of-order execution), κάτι που συνεπάγεται και την *ολοκλήρωση εκτός σειράς* (out-of-order completion).

Η εκτέλεση εκτός σειράς καθιστά εφικτή την πιθανότητα εμφάνισης κινδύνων WAR και WAW, κάτι που δεν είναι δυνατό να προκύψει ούτε στη διασωλήνωση ακεραίων των πέντε σταδίων και ούτε στη λογική επέκτασή της σε διασωλήνωση κινητής υποδιαστολής εντός σειράς. Ας εξετάσουμε την παρακάτω ακολουθία κώδικα κινητής υποδιαστολής της MIPS:

```
DIV.D    F0,F2,F4
ADD.D    F6,F0,F8
SUB.D    F8,F10,F14
MUL.D    F6,F10,F8
```

Υπάρχει αντιεξάρτηση ανάμεσα στις εντολές `ADD.D` και `SUB.D` και στην περίπτωση που η διασωλήνωση εκτελέσει τη `SUB.D` πριν την `ADD.D` (η οποία αναμένει την `DIV.D`) θα έχει παραβιάσει την αντιεξάρτηση, κάτι που θα έχει ως αποτέλεσμα την εμφάνιση κινδύνου WAR. Παρόμοια, για την αποφυγή των εξαρτήσεων εξόδου, όπως της εγγραφής του `F6` από την `MUL.D`, πρέπει να αντιμετωπίζονται οι κίνδυνοι WAW. Όπως θα διαπιστώσουμε και οι δύο αυτοί κίνδυνοι μπορούν να αποφευχθούν με τη χρήση της μετονομασίας καταχωρητών.

Η ολοκλήρωση εκτός σειράς προκαλεί, επίσης, σοβαρές επιπλοκές στο χειρισμό των εξαιρέσεων. Η δυναμική χρονοδρομολόγηση που προβαίνει σε ολοκλήρωση εκτός σειράς πρέπει να διατηρεί τη συμπεριφορά των εξαιρέσεων, με την έννοια ότι προκύπτουν *ακριβώς* εκείνες οι εξαιρέσεις που θα προέκυπταν στην *πραγματικότητα*, αν οι εντολές εκτελούνταν ακολουθώντας αυστηρά τη σειρά προγράμματος. Οι επεξεργαστές που υποστηρίζουν τη δυναμική χρονοδρομολόγηση διατηρούν τη συμπεριφορά των εξαιρέσεων διασφαλίζοντας ότι καμία εντολή δεν μπορεί να παραγάγει κάποια εξαίρεση, μέχρις ότου να γνωρίζει ο επεξεργαστής ότι η εντολή που προκαλεί την εξαίρεση θα εκτελεστεί. Σύντομα θα διαπιστώσουμε τον τρόπο με τον οποίο διασφαλίζεται η παραπάνω προϋπόθεση.

Παρόλο που η διατήρηση της συμπεριφοράς των εξαιρέσεων αποτελεί απαραίτητη προϋπόθεση, οι επεξεργαστές δυναμικής χρονοδρομολόγησης ενδέχεται να παράγουν *ανακριβείς* (imprecise) εξαιρέσεις. Η εξαίρεση θεωρείται *ανακριβής*, στην περίπτωση που, όταν προκύπτει κάποια εξαίρεση, η κατάσταση του επεξεργαστή δεν είναι ακριβώς ίδια με αυτήν που θα υπήρχε στην περίπτωση κατά την οποία οι εντολές εκτελούνταν ακολουθιακά τηρώντας αυστηρά τη σειρά προγράμματος. Οι ανακριβείς εξαιρέσεις μπορούν να προκύψουν εξαιτίας δύο πιθανών αιτιών:

1. Η διασωλήνωση μπορεί να έχει *ήδη ολοκληρώσει* τις εντολές οι οποίες βρίσκονται σε *μεταγενέστερη* θέση στη σειρά προγράμματος από την εντολή που προκαλεί την εξαίρεση.

2. Η διασωλήνωση μπορεί να μην έχει *ολοκληρώσει ακόμα* κάποιες εντολές οι οποίες βρίσκονται σε πρωτύτερη θέση στη σειρά προγράμματος από την εντολή που προκαλεί την εξαίρεση.

Οι ανακριβείς εξαιρέσεις δυσχεραίνουν την επανεκκίνηση της εκτέλεσης μετά από την εμφάνιση κάποιας εξαίρεσης. Αντί να εξετάσουμε την επίλυση των προβλημάτων αυτών στην ενότητα αυτή, θα διερευνήσουμε στην Ενότητα 2.6 μία λύση η οποία παρέχει ακριβείς (precise) εξαιρέσεις στο πλαίσιο χρήσης ενός επεξεργαστή που υποστηρίζει την εικασία. Όσον αφορά τις εξαιρέσεις κινητής υποδιαστολής, έχουν χρησιμοποιηθεί άλλες λύσεις, οι οποίες μελετώνται στο Παράρτημα Ι.

Για να επιτραπεί η εκτέλεση εκτός σειράς, στην ουσία διαχωρίζουμε το στάδιο ID της διασωλήνωσης των πέντε σταδίων σε δύο επιμέρους στάδια:

- *Έκδοση* (issue): Αποκωδικοποίηση εντολών, έλεγχος για δομικούς κινδύνους.

- *Ανάγνωση ορισμάτων* (read operands): Αναμονή έως ότου να μην υφίστανται κίνδυνοι δεδομένων και, στη συνέχεια, ανάγνωση των ορισμάτων.

Το στάδιο της ανάκτησης εντολών προηγείται του σταδίου έκδοσης και η ανάκτηση μπορεί να οδηγεί στην αποθήκευση κάθε εντολής στον καταχωρητή εντολών ή σε μια ουρά που περιλαμβάνει τις εκκρεμείς εντολές. Στη συνέχεια, οι εντολές εκδίδονται είτε από τον καταχωρητή είτε από την ουρά. Το στάδιο ΕΧ ακολουθεί το στάδιο της ανάγνωσης των ορισμάτων, όπως ακριβώς συμβαίνει και στη διασωλήνωση των πέντε σταδίων. Η εκτέλεση μπορεί να απαιτεί πολλαπλούς κύκλους, ανάλογα με τη λειτουργία.

Διαχωρίζουμε το χρονικό σημείο κατά το οποίο *ξεκινά η εκτέλεση* από το χρονικό σημείο κατά το οποίο *ολοκληρώνεται η εκτέλεση*. Ανάμεσα στα δύο αυτά χρονικά σημεία η εντολή είναι σε *εκτέλεση*. Η διασωλήνωσή μας επιτρέπει την ταυτόχρονη εκτέλεση πολλαπλών εντολών, δυνατότητα που αν δεν υπήρχε, τότε ένα από τα κύρια πλεονεκτήματα της δυναμικής χρονοδρομολόγησης θα έπαυε να υφίσταται. Η ταυτόχρονη εκτέλεση πολλαπλών εντολών απαιτεί την ύπαρξη είτε πολλαπλών λειτουργικών μονάδων είτε λειτουργικών μονάδων που υποστηρίζουν τη διασωλήνωση ή συνδυασμό και των δύο. Καθώς οι δύο αυτές δυνατότητες - οι λειτουργικές μονάδες που υποστηρίζουν τη διασωλήνωση και οι πολλαπλές λειτουργικές μονάδες - είναι ουσιαστικά ισοδύναμες, για τις ανάγκες του ελέγχου της διασωλήνωσης υποθέτουμε ότι ο επεξεργαστής διαθέτει πολλαπλές λειτουργικές μονάδες.

Στη διασωλήνωση δυναμικής χρονοδρομολόγησης, όλες οι εντολές διέρχονται από το στάδιο της έκδοσης με τη σειρά τους (έκδοση εντός σειράς). Ωστόσο, μπορεί είτε να τεθούν σε αδράνεια είτε η μία να παρακάμψει την άλλη στο δεύτερο στάδιο (στο στάδιο ανάγνωσης ορισμάτων) και επομένως να εισέλθουν στο στάδιο εκτέλεσης εκτός σειράς. Η χρήση *πίνακα παρακολούθησης* (scoreboarding) αποτελεί τεχνική που επιτρέπει την εκτέλεση των εντολών εκτός σειράς, όταν υπάρχουν επαρκείς πόροι και δεν υφίστανται εξαρτήσεις δεδομένων. Η τεχνική αυτή έχει πάρει την ονομασία της από τον πίνακα παρακολούθησης του CDC 6600, όπου και αναπτύχθηκε αυτή η δυνατότητα, κάτι που εξετάζεται στο Παράρτημα Α. Εδώ, δίνουμε έμφαση σε μια πιο εξελιγμένη τεχνική, γνωστή ως *αλγόριθμος του Tomasulo* (Tomaluso's algorithm), η οποία έχει σημαντικά πλεονεκτήματα σε σχέση με τη χρήση του πίνακα παρακολούθησης.

Δυναμική Χρονοδρομολόγηση Χρησιμοποιώντας την Προσέγγιση του Tomasulo

Η μονάδα κινητής υποδιαστολής του IBM 360/91 χρησιμοποιούσε προηγμένο σύστημα το οποίο παρείχε τη δυνατότητα εκτέλεσης εκτός σειράς. Η προσέγγιση αυτή, της οποίας εφευρέτης υπήρξε ο Robert Tomasulo, παρακολουθεί τα χρονικά σημεία κατά τα οποία είναι διαθέσιμα τα ορίσματα για τις εντολές, έτσι ώστε να ελαχιστοποιούνται οι κίνδυνοι RAW, ενώ εισάγει τη μετονομασία των καταχωρητών για την ελαχιστοποίηση των κινδύνων WAW και WAR. Στους σύγχρονους επεξεργαστές, υπάρχουν αρκετές παραλλαγές της προσέγγισης αυτής, παρόλο που οι βασικές έννοιες της παρακολούθησης των εξαρτήσεων των εντολών για την παροχή της δυνατότητας εκτέλεσης μόλις τα ορίσματα καταστούν διαθέσιμα, καθώς και της μετονομασίας των καταχωρητών για την αποφυγή των κινδύνων WAR και WAW αποτελούν κοινά χαρακτηριστικά.

Στόχος της IBM ήταν η επίτευξη υψηλής απόδοσης κινητής υποδιαστολής χρησιμοποιώντας σύνολο εντολών και μεταγλωττιστές που είχαν σχεδιαστεί για ολόκληρη την οικογένεια των υπολογιστών 360 και όχι χρησιμοποιώντας εξειδικευμένους μεταγλωττιστές που προορίζονταν για τους προηγμένους επεξεργαστές. Η αρχιτεκτονική 360 διέθετε μόνο τέσσερις καταχωρητές κινητής υποδιασιολής διπλής ακρίβειας, κάτι που περιόριζε την αποτελεσματικότητα της χρονοδρομολόγησης του μεταγλωττιστή. Το γεγονός αυτό αποτέλεσε άλλο ένα κίνητρο για την προσέγγιση Tomasulo. Επιπλέον, ο IBM 360/91 διέθετε μεγάλο χρόνο προσπέλασης της μνήμης, καθώς και μεγάλες καθυστερήσεις κινητής υποδιαστολής, τις οποίες ο αλγόριθμος του Tomasulo σχεδιάστηκε να τις αντιμετωπίσει. Στο τέλος της ενότητας, θα διαπιστώσουμε ότι ο αλγόριθμος του Tomasulo μπορεί να υποστηρίξει, επίσης, την επικαλυπτόμενη εκτέλεση πολλαπλών επαναλήψεων ενός βρόχου.

Θα εξηγήσουμε τον αλγόριθμο, ο οποίος εστιάζει στη μονάδα κινητής υποδιαστολής και στη μονάδα φόρτωσης - αποθήκευσης, στο πλαίσιο του συνόλου

εντολών της MIPS. Η κύρια διαφορά ανάμεσα στη MIPS και στην 360 είναι το γεγονός ότι η δεύτερη αρχιτεκτονική περιλαμβάνει εντολές καταχωρητή - μνήμης. Επειδή ο αλγόριθμος του Tomasulo χρησιμοποιεί μία λειτουργική μονάδα φόρτωσης, δεν χρειάζονται σημαντικές αλλαγές για την προσθήκη τρόπων διευθυνσιοδότησης καταχωρητή - μνήμης. Ο IBM 360/91 διέθετε, επίσης, λειτουργικές μονάδες που υποστήριζαν τη διασωλήνωση και όχι πολλαπλές λειτουργικές μονάδες, ωστόσο περιγράφουμε τον αλγόριθμο κάνοντας την υπόθεση εργασίας ότι υπάρχουν πολλαπλές λειτουργικές μονάδες. Είναι απλή η εννοιολογική προέκταση της παραπάνω προσέγγισης, ώστε να περιλαμβάνεται η διασωλήνωση αυτών των λειτουργικών μονάδων.

Όπως θα διαπιστώσουμε, οι κίνδυνοι RAW μπορούν να αποφευχθούν μέσω της εκτέλεσης εντολών μόνο όταν τα ορίσματά τους είναι διαθέσιμα. Οι κίνδυνοι WAR και WAW, οι οποίοι προκύπτουν εξαιτίας εξαρτήσεων ονομάτων, παύουν να υφίστανται με την εφαρμογή της μετονομασίας των καταχωρητών. Η *μετονομασία καταχωρητών* εξαλείφει τους κινδύνους αυτούς, μετονομάζοντας όλους τους καταχωρητές προορισμού, συμπεριλαμβανομένων και αυτών που έχουν σε εκκρεμότητα μια ανάγνωση ή εγγραφή κάποιας προηγούμενης εντολής, έτσι ώστε η εγγραφή εκτός σειράς να μην επηρεάζει οποιεσδήποτε ενιολές εξαρτώνται από προηγούμενη τιμή ενός ορίσματος.

Για την καλύτερη κατανόηση του τρόπου με τον οποίο η μετονομασία καταχωρητών εξαλείφει τους κινδύνους WAR και WAW, ας εξετάσουμε την παρακάτω ακολουθία κώδικα η οποία περιλαμβάνει τόσο έναν ενδεχόμενο κίνδυνο WAR όσο και έναν WAW:

```
DIV.D    F0,F2,F4
ADD.D    F6,F0,F8
S.D      F6,0(R1)
SUB.D    F8,F10,F14
MUL.D    F6,F10,F8
```

Υπάρχει αντεξάρτηση ανάμεσα στην `ADD.D` και τη `SUB.D`, καθώς και εξάρτηση εξόδου ανάμεσα στην `ADD.D` και τη `MUL.D`, κάτι που οδηγεί σε δύο πιθανούς κινδύνους: σε κίνδυνο WAR κατά τη χρήση του `F8` από την `ADD.D` και σε κίνδυνο WAW, καθώς η `ADD.D` ενδέχεται να ολοκληρωθεί αργότερα από τη `MUL.D`. Υπάρχουν, επίσης, τρεις πραγματικές εξαρτήσεις δεδομένων: ανάμεσα στην `DIV.D` και την `ADD.D`, ανάμεσα στην `SUB.D` και τη `MUL.D` και ανάμεσα στην `ADD.D` και την `S.D`.

Αυτές οι δύο εξαρτήσεις ονομάτων μπορούν να εξαλειφθούν μέσω της μετονομασίας καταχωρητών. Για να το απλοποιήσουμε, ας υποθέσουμε ότι υπάρχουν δύο προσωρινοί καταχωρητές, οι `S` και `T`. Χρησιμοποιώντας τους `S` και `T`, η ακολουθία μπορεί να συγγραφεί εκ νέου χωρίς να υπάρχουν εξαρτήσεις:

```
DIV.D    F0,F2,F4
ADD.D    S,F0,F8
S.D      S,0(R1)
SUB.D    T,F10,F14
MUL.D    F6,F10,T
```

Επιπλέον, οποιεσδήποτε επακόλουθες χρήσεις του F8 πρέπει να αντικατασταθούν από τον καταχωρητή T. Σε αυτό το τμήμα κώδικα, η διαδικασία μετονομασίας μπορεί να πραγματοποιηθεί στατικά από το μεταγλωττιστή. Για την εύρεση κάθε χρήσης του F8, η οποία έπεται στον κώδικα, απαιτείται είτε εξελιγμένη ανάλυση μεταγλωττιστή είτε υποστήριξη υλικού, καθώς μπορεί να μεσολαβούν διακλαδώσεις ανάμεσα στο παραπάνω τμήμα κώδικα και σε κάποια μεταγενέστερη χρήση του F8. Όπως θα διαπιστώσουμε, ο αλγόριθμος του Tomasulo μπορεί να χειριστεί τη μετονομασία μεταξύ διακλαδώσεων.

Στην προσέγγιση του Tomasulo, η μετονομασία των καταχωρητών παρέχεται από τους *σταθμούς κράτησης* (reservation stations), που αποθηκεύουν σε απομονωτή τα ορίσματα των εντολών οι οποίες βρίσκονται σε αναμονή έκδοσης. Σύμφωνα με τη βασική ιδέα ο σταθμός κράτησης ανακτά και αποθηκεύει σε απομονωτή κάποιο όρισμα μόλις αυτό καταστεί διαθέσιμο, εξαλείφοντας με τον τρόπο αυτό την ανάγκη λήψης του ορίσματος από κάποιον καταχωρητή. Επιπλέον, οι εκκρεμείς εντολές καθορίζουν το σταθμό κράτησης ο οποίος θα παράσχει την είσοδό τους. Τέλος, όταν υπάρχει επικάλυψη διαδοχικών ενεργειών εγγραφής κάποιου καταχωρητή, στην πραγματικότητα χρησιμοποιείται μόνο η τελευταία εγγραφή η οποία και ενημερώνει τον καταχωρητή. Καθώς εκδίδονται οι εντολές, οι προσδιοριστές των καταχωρητών για τα εκκρεμή ορίσματα μετονομάζονται, λαμβάνοντας την ονομασία του σταθμού κράτησης, ο οποίος παρέχει τη μετονομασία των καταχωρητών.

Καθώς μπορούν να υφίστανται περισσότεροι σταθμοί κράτησης απ' ό,τι πραγματικοί καταχωρητές, η τεχνική αυτή μπορεί να εξαλείψει ακόμα και τους κινδύνους που προκύπτουν από εξαρτήσεις ονομάτων, οι οποίες δεν μπορούν να αντιμετωπιστούν από τους μεταγλωττιστές. Κατά τη διερεύνηση των συστατικών στοιχείων της προσέγγισης Tomasulo, θα επιστρέψουμε στο ζήτημα της μετονομασίας των καταχωρητών και θα εξετάσουμε ακριβώς τον τρόπο με τον οποίο υλοποιείται η μετονομασία, καθώς και τον τρόπο με τον οποίο εξαλείφονται οι κίνδυνοι WAR και WAW.

Η χρήση των σταθμών κράτησης, αντί της χρήσης μιας συγκεντρωτικής συστοιχίας καταχωρητών, οδηγεί σε δύο σημαντικές ιδιότητες. Πρώτον, τόσο ο έλεγχος για κινδύνους όσο και ο έλεγχος εκτέλεσης είναι κατανεμημένος: Οι πληροφορίες που διατηρούνται στους σταθμούς κράτησης σε κάθε λειτουργική μονάδα προσδιορίζουν το χρονικό σημείο κατά το οποίο μπορεί να ξεκινήσει η εκτέλεση της εντολής στη μονάδα αυτή. Δεύτερον, τα δεδομένα μεταφέρονται απευθείας από τους σταθμούς κράτησης, στους οποίους έχουν απομονωθεί

(buffered), στις λειτουργικές μονάδες και δεν διέρχονται μέσω των καταχωρητών. Η παράκαμψη αυτή πραγματοποιείται με τη βοήθεια ενός κοινού διαύλου αποτελεσμάτων (result bus), ο οποίος επιτρέπει σε όλες τις μονάδες που αναμένουν κάποιο όρισμα να φορτώνονται ταυτόχρονα (στον 360/91 αυτό ονομάζεται *κοινός δίαυλος δεδομένων* ή CDB[7]). Στις διασωληνώσεις που διαθέτουν πολλαπλές μονάδες εκτέλεσης και προβαίνουν σε πολλαπλή έκδοση εντολών ανά κύκλο, χρειάζονται περισσότεροι από έναν δίαυλοι.

Το Σχήμα 2.9 παρουσιάζει τη βασική δομή των επεξεργαστών που βασίζονται στην προσέγγιση Tomasulo, συμπεριλαμβανομένης τόσο της μονάδας κινητής υποδιαστολής όσο και της μονάδας φόρτωσης - αποθήκευσης. Ωστόσο, δεν παρουσιάζεται κανένας από τους πίνακες ελέγχου εκτέλεσης. Κάθε σταθμός κράτησης διατηρεί μια εντολή η οποία έχει εκδοθεί και αναμένει να εκτελεστεί σε κάποια λειτουργική μονάδα, ενώ συγχρόνως διατηρεί είτε τις τιμές των ορισμάτων της εντολής αυτής, με την προϋπόθεση ότι έχουν ήδη υπολογιστεί είτε τα ονόματα των σταθμών κράτησης οι οποίοι θα παράσχουν τις τιμές των ορισμάτων.

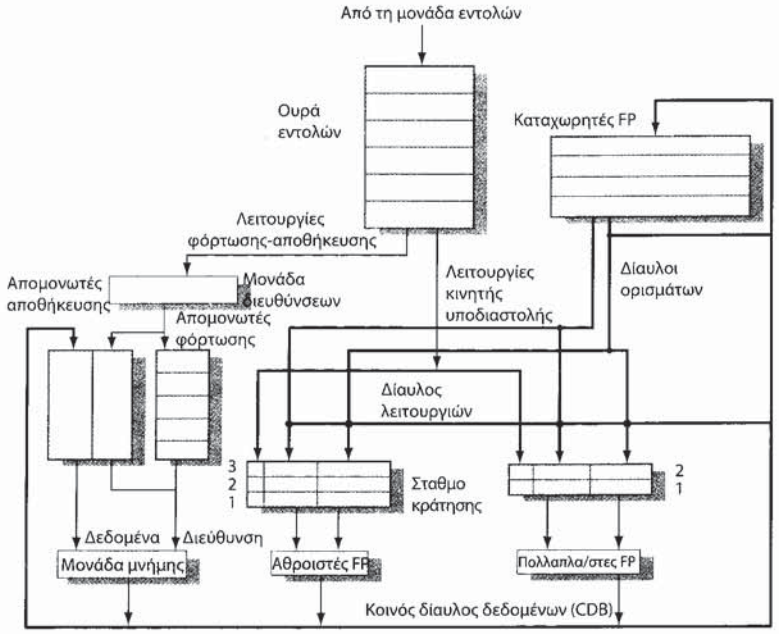
Οι απομονωτές φόρτωσης και οι απομονωτές αποθήκευσης διατηρούν δεδομένα ή διευθύνσεις που προέρχονται από τη μνήμη και οδεύουν προς αυτή, ενώ συγχρόνως λειτουργούν σχεδόν πανομοιότυπα με τους σταθμούς κράτησης, επομένως τους διαχωρίζουμε μόνο όταν αυτό είναι απαραίτητο. Οι καταχωρητές κινητής υποδιαστολής διασυνδέονται με τις λειτουργικές μονάδες διαμέσου ενός ζεύγους διαύλων και με τους απομονωτές αποθήκευσης διαμέσου ενός μόνο διαύλου. Όλα τα αποτελέσματα που προέρχονται από τις λειτουργικές μονάδες και τη μνήμη αποστέλλονται στον κοινό δίαυλο δεδομένων, ο οποίος επικοινωνεί με τα πάντα εκτός από τον απομονωτή φόρτωσης. Όλοι οι σταθμοί κράτησης διαθέτουν πεδία ετικέτας (tag fields), τα οποία χρησιμοποιούνται για τον έλεγχο της διασωλήνωσης.

Πριν περιγράψουμε τις λεπτομέρειες των σταθμών κράτησης και του αλγορίθμου, ας εξετάσουμε τα βήματα τα οποία ακολουθούν οι εντολές. Υπάρχουν τρία μόνο βήματα, παρόλο που πλέον καθένα από αυτά μπορεί να απαιτεί αυθαίρετο αριθμό κύκλων ρολογιού:

1. *Έκδοση*: Λήψη της επόμενης εντολής από την κεφαλή της ουράς των εντολών, η οποία διατηρείται ακολουθώντας σειρά FIFO[8], έτσι ώστε να διασφαλίζεται η διατήρηση της ορθής ροής δεδομένων. Αν υπάρχει κάποιος κατάλληλος σταθμός κράτησης που είναι κενός, τότε πραγματοποιείται έκδοση της εντολής η οποία τοποθετείται στο σταθμό αυτό μαζί με τις τιμές των ορισμάτων, αν αυτές βρίσκονται στους καταχωρητές την τρέχουσα χρονική στιγμή. Αν δεν υπάρχει κάποιος κενός σταθμός κράτησης, τότε υπάρχει δομικός κίνδυνος και η εντολή τίθεται σε αδράνεια έως ότου ο

[7]Σ.τ.Μ. Ο αντίστοιχος αγγλικός όρος είναι "common data bus"

[8]Σ.τ.Μ. Ακρωνύμιο του αγγλικού όρου "first-in-first-out" που σημαίνει «πρώτη μέσα πρώτη έξω».



Σχήμα 2.9 Η βασική δομή μιας μονάδας κινητής υποδιαστολής της MIPS χρησιμοποιώντας τον αλγόριθμο του Tomasulo. Οι εντολές αποστέλλονται από τη μονάδα εντολών στην ουρά εντολών από την οποία και έχουν εκδοθεί με σειρά FIFO. Οι σταθμοί κράτησης περιλαμβάνουν τη λειτουργία και τα πραγματικά ορίσματα, καθώς και πληροφορίες που χρησιμοποιούνται για την ανίχνευση και την αντιμετώπιση κινδύνων. Οι απομονωτές φόρτωσης διαθέτουν τρεις λειτουργίες: διατηρούν τα συστατικά στοιχεία της ενεργού διεύθυνσης μέχρι αυτή να υπολογιστεί, παρακολουθούν τις εκκρεμείς λειτουργίες φόρτωσης οι οποίες βρίσκονται σε αναμονή εντός της μνήμης και διατηρούν τα αποτελέσματα των ολοκληρωμένων λειτουργιών φόρτωσης οι οποίες αναμένουν τον CDB. Παρόμοια, οι απομονωτές αποθήκευσης διαθέτουν τρεις λειτουργίες: διατηρούν τα συστατικά στοιχεία της ενεργού διεύθυνσης μέχρι αυτή να υπολογιστεί, διατηρούν τις διευθύνσεις μνήμης προορισμού των εκκρεμών λειτουργιών αποθήκευσης οι οποίες αναμένουν την τιμή των δεδομένων έτσι ώστε να την αποθηκεύσουν και διατηρούν τη διεύθυνση και την τιμή που πρόκειται να αποθηκευτεί, μέχρις ότου η μονάδα μνήμης καταστεί διαθέσιμη. Όλα τα αποτελέσματα, είτε προέρχονται από τις μονάδες FP είτε από τη μονάδα φόρτωσης, τοποθετούνται στον CDB, ο οποίος επικοινωνεί τόσο με τη συστοιχία καταχωρητών FP όσο και με τους σταθμούς κράτησης και τους απομονωτές αποθήκευσης. Οι αθροιστές FP πραγματοποιούν την πρόσθεση και την αφαίρεση, ενώ οι πολλαπλασιαστές Tomasulo πραγματοποιούν τον πολλαπλασιασμό και τη διαίρεση.

σταθμός ή ο απομονωτής να ελευθερωθεί. Αν τα ορίσματα δεν βρίσκονται στους καταχωρητές, τότε παρακολουθούνται οι λειτουργικές μονάδες οι οποίες θα παραγάγουν τα ορίσματα. Στο βήμα αυτό μετονομάζονται οι καταχωρητές, κάτι που οδηγεί στην εξάλειψη των κινδύνων WAR και WAW (το στάδιο αυτό, στους επεξεργαστές δυναμικής χρονοδρομολόγησης, μερικές

φορές ονομάζεται *διαβίβαση*[9]).

2. *Εκτέλεση*: Αν ένα ή περισσότερα ορίσματα δεν είναι ακόμα διαθέσιμα, τότε, την ώρα που αναμένουμε τον υπολογισμό των ορισμάτων, παρακολουθείται ο κοινός δίαυλος δεδομένων. Όταν κάποιο όρισμα καθίσταται διαθέσιμο, τοποθετείται σε οποιονδήποτε σταθμό κράτησης το αναμένει. Όταν όλα τα ορίσματα είναι διαθέσιμα, η λειτουργία μπορεί να εκτελεστεί στην αντίστοιχη λειτουργική μονάδα. Η καθυστέρηση της εκτέλεσης των εντολών, έως ότου όλα τα ορίσματα να καταστούν διαθέσιμα, οδηγεί στην αποφυγή των κινδύνων RAW (μερικοί επεξεργαστές δυναμικής χρονοδρομολόγησης ονομάζουν το βήμα αυτό ως «έκδοση», ωστόσο εμείς χρησιμοποιούμε την ονομασία «εκτέλεση», η οποία χρησιμοποιήθηκε στον πρώτο επεξεργαστή δυναμικής χρονοδρομολόγησης, στον CDC 6600).

   Αξίζει να σημειωθεί ότι αρκετές εντολές μπορεί να καθίστανται διαθέσιμες για την ίδια λειτουργική μονάδα στον ίδιο κύκλο ρολογιού. Παρόλο που οι ανεξάρτητες λειτουργικές μονάδες μπορούν να ξεκινούν την εκτέλεση διαφορετικών εντολών στον ίδιο κύκλο ρολογιού, στην περίπτωση που περισσότερες από μία εντολές είναι έτοιμες να εκτελεστούν σε μία λειτουργική μονάδα, τότε η μονάδα αυτή πρέπει επιλέξει μία από αυτές. Στους σταθμούς κράτησης κινητής υποδιαστολής, η επιλογή αυτή μπορεί να πραγματοποιηθεί αυθαίρετα. Ωστόσο, οι λειτουργίες φόρτωσης και αποθήκευσης παρουσιάζουν επιπρόσθετες δυσκολίες.

   Οι εντολές φόρτωσης και αποθήκευσης απαιτούν διαδικασία εκτέλεσης που αποτελείται από δύο βήματα. Στο πρώτο βήμα, όταν ο καταχωρητής βάσης είναι διαθέσιμος, υπολογίζεται η ενεργός διεύθυνση, η οποία στη συνέχεια τοποθετείται στον απομονωτή φόρτωσης ή αποθήκευσης. Οι εντολές φόρτωσης που βρίσκονται στον απομονωτή φόρτωσης εκτελούνται μόλις καταστεί διαθέσιμη η μονάδα μνήμης. Οι εντολές αποθήκευσης που βρίσκονται στον απομονωτή αποθήκευσης αναμένουν την τιμή που πρόκειται να αποθηκευτεί, πριν σταλεί στη μονάδα μνήμης. Οι εντολές φόρτωσης και αποθήκευσης ακολουθούν τη σειρά προγράμματος κατά τη διάρκεια υπολογισμού της ενεργού διεύθυνσης, κάτι που, όπως θα διαπιστώσουμε σύντομα, βοηθά στην παρεμπόδιση εμφάνισης κινδύνων μέσω της μνήμης.

   Για τη διατήρηση της συμπεριφοράς των εξαιρέσεων, δεν επιτρέπεται η έναρξη της εκτέλεσης καμίας εντολής, μέχρις ότου να ολοκληρωθούν όλες οι διακλαδώσεις που προηγούνται της εντολής αυτής στη σειρά προγράμματος. Ο περιορισμός αυτός διασφαλίζει ότι η εντολή η οποία προκαλεί εξαίρεση κατά τη διάρκεια της εκτέλεσης θα κατορθώσει, στην πραγματικότητα, να εκτελεστεί. Στην περίπτωση των επεξεργαστών που χρησιμοποιούν πρόβλεψη διακλάδωσης (κάτι που συμβαίνει σε όλους τους επεξεργαστές

[9]Σ.τ.Μ. Ο αντίστοιχος αγγλικός όρος είναι ο "dispatch".

δυναμικής χρονοδρομολόγησης), αυτό συνεπάγεται ότι ο επεξεργαστής, πριν να επιτρέψει την έναρξη της εκτέλεσης κάποιας εντολής που έπεται της διακλάδωσης, πρέπει να γνωρίζει ότι η πρόβλεψη της διακλάδωσης είναι ορθή. Σε περίπτωση που ο επεξεργαστής καταγράψει την εμφάνιση της εξαίρεσης, χωρίς ωστόσο να προβεί στην υλοποίησή της, τότε μπορεί να ξεκινήσει η εκτέλεση κάποιας εντολής, ωστόσο η εντολή αυτή δεν μπορεί να τεθεί σε αδράνεια πριν εισέλθει στο στάδιο Εγγραφής Αποτελέσματος (Write Result).

Όπως θα διαπιστώσουμε, η εικασία παρέχει πιο ευέλικτη και πολύπλοκη μέθοδο χειρισμού των εξαιρέσεων και ως εκ τούτου δεν θα προβούμε στην προσθήκη των βελτιώσεων αυτών, ενώ θα παρουσιάσουμε αργότερα τον τρόπο με τον οποίο η εικασία χειρίζεται το πρόβλημα αυτό.

3. *Εγγραφή αποτελέσματος:* Όταν το αποτέλεσμα καταστεί διαθέσιμο, τότε πραγματοποιείται η εγγραφή του στον CDB και από εκεί στους καταχωρητές και σε οποιουσδήποτε σταθμούς κράτησης (συμπεριλαμβανομένων και των απομονωτών αποθήκευσης) το αναμένουν. Οι εντολές αποθήκευσης τοποθετούνται στον απομονωτή αποθήκευσης, έως ότου να είναι διαθέσιμη τόσο η τιμή που πρόκειται να αποθηκευτεί όσο και η διεύθυνση αποθήκευσης και, εν συνεχεία, πραγματοποιείται η εγγραφή του αποτελέσματος μόλις η μονάδα μνήμης ελευθερωθεί.

Οι δομές δεδομένων οι οποίες ανιχνεύουν και εξαλείφουν τους κινδύνους είναι ενσωματωμένες στους σταθμούς κράτησης, στη συστοιχία καταχωρητών και στους απομονωτές φόρτωσης και αποθήκευσης, όπου ελαφρά διαφορετικές πληροφορίες ενσωματώνονται σε διαφορετικά αντικείμενα. Οι ετικέτες αυτές αποτελούν στην ουσία τα ονόματα ενός εκτεταμένου συνόλου εικονικών καταχωρητών (virtual registers), οι οποίοι χρησιμοποιούνται για τη μετονομασία. Στο παράδειγμά μας, το πεδίο ετικέτας αποτελεί μέγεθος των 4 bits το οποίο υποδηλώνει είτε έναν από τους πέντε σταθμούς κράτησης είτε έναν από τους πέντε απομονωτές φόρτωσης. Όπως θα διαπιστώσουμε, το παραπάνω ισοδυναμεί με 10 καταχωρητές που μπορούν να οριστούν ως καταχωρητές αποτελέσματος (σε αντίθεση με τους τέσσερις καταχωρητές διπλής ακρίβειας που περιλαμβάνει η αρχιτεκτονική 360). Σε επεξεργαστές που διαθέτουν περισσότερους πραγματικούς καταχωρητές, θα επιθυμούσαμε η μετονομασία να παρέχει ακόμα μεγαλύτερο σύνολο εικονικών καταχωρητών. Το πεδίο ετικέτας περιγράφει ποιος είναι ο σταθμός κράτησης ο οποίος περιέχει την εντολή η οποία θα παραγάγει κάποιο αποτέλεσμα που χρειάζεται να χρησιμοποιηθεί ως όρισμα αφετηρίας.

Μόλις η εντολή εκδοθεί και βρεθεί σε κατάσταση αναμονής κάποιου ορίσματος αφετηρίας, η αναφορά στο όρισμα πραγματοποιείται χρησιμοποιώντας τον αριθμό του σταθμού κράτησης, ο οποίος ανατέθηκε στην εντολή που θα εγγράψει τον καταχωρητή. Οι μη χρησιμοποιημένες τιμές, όπως αυτή του μηδενός,

υποδηλώνουν ότι το όρισμα είναι ήδη διαθέσιμο εντός των καταχωρητών. Επειδή υπάρχουν περισσότεροι σταθμοί κράτησης απ' ό,τι υπαρκτοί αριθμοί καταχωρητών, οι κίνδυνοι WAW και WAR εξαλείφονται μέσω της μετονομασίας των αποτελεσμάτων που χρησιμοποιούν τους αριθμούς των σταθμών κράτησης. Παρόλο που στην προσέγγιση του Tomasulo οι σταθμοί κράτησης χρησιμοποιούνται ως εκτεταμένοι εικονικοί καταχωρητές, άλλες προσεγγίσεις μπορούν να χρησιμοποιούν σύνολο καταχωρητών που περιλαμβάνει επιπρόσθετους καταχωρητές ή δομή όπως αυτή του απομονωτή αναδιάταξης, τον οποίο θα εξετάσουμε στην Ενότητα 2.6.

Στην προσέγγιση Tomasulo, καθώς και στις επόμενες μεθόδους που θα εξετάσουμε για την υποστήριξη της εικασίας, τα αποτελέσματα μεταδίδονται σε κάποιον δίαυλο (στον CDB), ο οποίος παρακολουθείται από τους σταθμούς κράτησης. Ο συνδυασμός του κοινού διαύλου αποτελεσμάτων και της ανάκτησης των αποτελεσμάτων που βρίσκονται στο δίαυλο από τους σταθμούς κράτησης υλοποιεί τους μηχανισμούς προώθησης και παράκαμψης που χρησιμοποιούνται στις διασωληνώσεις στατικής χρονοδρομολόγησης. Ωστόσο, για να επιτευχθεί αυτό, η προσέγγιση της δυναμικής χρονοδρομολόγησης εισάγει έναν κύκλο λανθάνοντος χρόνου ανάμεσα στην αφετηρία και το αποτέλεσμα, καθώς ο συνδυασμός κάποιου αποτελέσματος και της χρήσης του δεν μπορεί να πραγματοποιηθεί πριν το στάδιο της Εγγραφής Αποτελέσματος. Κατά συνέπεια, στη διασωλήνωση δυναμικής χρονοδρομολόγησης, ο ενεργός λανθάνων χρόνος μεταξύ μιας εντολής που οδηγεί στην παραγωγή κάποιου αποτελέσματος και κάποιας άλλης εντολής που καταναλώνει το αποτέλεσμα αυτό είναι τουλάχιστον κατά ένα κύκλο μεγαλύτερος από το λανθάνοντα χρόνο της λειτουργικής μονάδας που παράγει το αποτέλεσμα.

Κατά την περιγραφή της λειτουργίας της προσέγγισης αυτής, αντί να εισαγάγουμε νέα ορολογία, χρησιμοποιούμε την ορολογία της προσέγγισης παρακολούθησης πίνακα (scoreboard) του CDC (δείτε το Παράρτημα Α), παραθέτοντας μόνο για ιστορικούς λόγους την ορολογία που χρησιμοποιήθηκε από τον IBM 360/91. Είναι σημαντικό να έχει κανείς κατά νου ότι οι ετικέτες της προσέγγισης Tomasulo αναφέρονται στον απομονωτή ή στη μονάδα η οποία θα παραγάγει το αποτέλεσμα. Τα ονόματα των καταχωρητών παύουν να χρησιμοποιούνται όταν η εντολή εκδίδεται εντός του σταθμού κράτησης.

Κάθε σταθμός κράτησης διαθέτει επτά πεδία:

- Op: Αφορά τη λειτουργία που πρόκειται να εκτελεστεί στα ορίσματα αφετηρίας S1 και S2

- Qj, Qk: Αφορούν τους σταθμούς κράτησης οι οποίοι θα παραγάγουν το αντίστοιχο όρισμα αφετηρίας. Η μηδενική τιμή υποδηλώνει είτε ότι το όρισμα αφετηρίας είναι ήδη διαθέσιμο στο Vj ή Vk είτε ότι είναι περιττό (ο IBM 360/91 τα ονομάζει αυτά ως SINKunit και SOURCEunit)

- Vj, Vk: Αφορούν την τιμή των ορισμάτων αφετηρίας. Αξίζει να σημειωθεί ότι μόνο ένα από τα πεδία V ή Q είναι έγκυρο για κάθε όρισμα. Για τις εντολές φόρτωσης, το πεδίο Vk χρησιμοποιείται για τη διατήρηση του πεδίου σχετικής απόστασης (τα πεδία αυτά ονομάζονται SINK και SOURCE στον IBM 360/91.)

- A - Χρησιμοποιείται για τη διατήρηση πληροφοριών που χρειάζονται για τον υπολογισμό της διεύθυνσης μνήμης για κάποια φόρτωση ή αποθήκευση. Αρχικά, εδώ αποθηκεύεται το άμεσο πεδίο της εντολής. Μετά τον υπολογισμό της διεύθυνσης, εδώ αποθηκεύεται η ενεργός διεύθυνση.

- Απασχολημένος: Υποδηλώνει ότι ο συγκεκριμένος σταθμός κράτησης, καθώς και οι συνοδευτικές λειτουργικές μονάδες είναι κατειλημμένες.

Η συστοιχία καταχωρητών διαθέτει το πεδίο Qi:

- Qi: Πρόκειται για τον αριθμό του σταθμού κράτησης ο οποίος περιέχει τη λειτουργία της οποίας το αποτέλεσμα πρέπει να αποθηκευτεί στον καταχωρητή αυτό. Αν η τιμή του Qi είναι κενή (ή 0), τότε καμία ενεργή εντολή δεν υπολογίζει την τρέχουσα χρονική στιγμή κάποιο αποτέλεσμα το οποίο προορίζεται για τον καταχωρητή αυτό, κάτι που σημαίνει ότι η τιμή απλά συμβολίζει τα περιεχόμενα του καταχωρητή.

Καθένας από τους απομονωτές φόρτωσης και αποθήκευσης διαθέτει ένα πεδίο, το A, το οποίο, μόλις ολοκληρωθεί το πρώτο βήμα εκτέλεσης, διατηρεί το αποτέλεσμα της ενεργού διεύθυνσης.

Στην επόμενη ενότητα, θα εξετάσουμε αρχικά κάποια παραδείγματα τα οποία καταδεικνύουν τον τρόπο λειτουργίας των μηχανισμών αυτών και εν συνεχεία θα εξετάσουμε λεπτομερώς τον αλγόριθμο.

2.5 Δυναμική Χρονοδρομολόγηση: Παραδείγματα και Αλγόριθμος

Πριν μελετήσουμε λεπτομερώς τον αλγόριθμο του Tomasulo, ας εξετάσουμε μερικά παραδείγματα, τα οποία θα βοηθήσουν στην κατανόηση του τρόπου λειτουργίας του αλγορίθμου.

Παράδειγμα: Ας παρουσιάσουμε την εικόνα των πινάκων πληροφοριών της παρακάτω ακολουθίας κώδικα, όταν μόνο η πρώτη φόρτωση έχει ολοκληρωθεί και εγγράψει τα αποτελέσματά της.

```
1.   L.D      F6,32(R2)
2.   L.D      F2,44(R3)
```

```
3.  MUL.D   F0,F2,F4
4.  SUB.D   F8,F2,F6
5.  DIV.D   F10,F0,F6
6.  ADD.D   F6,F8,F2
```

Απάντηση: Το Σχήμα 2.10 παρουσιάζει το αποτέλεσμα εντός τριών πινάκων. Οι αριθμοί που έχουν επισυναφθεί στις ονομασίες add, mult και load αντιστοιχούν στην ετικέτα του σταθμού κράτησης - το Add1 αντιπροσωπεύει το αποτέλεσμα που προέρχεται από την πρώτη μονάδα πρόσθεσης. Επιπλέον, έχουμε συμπεριλάβει έναν πίνακα κατάστασης εντολών. Ο πίνακας αυτός συμπεριλαμβάνεται μόνο για να βοηθήσει τον αναγνώστη στην κατανόηση του αλγορίθμου. *Δεν* αποτελεί στην πραγματικότητα μέρος του υλικού. Αντιθέτως, ο σταθμός κράτησης διατηρεί την κατάσταση κάθε εντολής που έχει εκδοθεί.

Η προσέγγιση του Tomasulo διαθέτει δύο σημαντικά πλεονεκτήματα σε σύγκριση με τις παλαιότερες και πιο απλές προσεγγίσεις: (1) την κατανομή της λογικής της ανίχνευσης κινδύνων και (2) την εξάλειψη των διαστημάτων αδράνειας για κινδύνους WAW και WAR.

Το πρώτο πλεονέκτημα προκύπτει από τους κατανεμημένους σταθμούς κράτησης και τη χρήση του Κοινού Διαύλου Δεδομένων (CDB). Αν πολλαπλές εντολές αναμένουν ένα συγκεκριμένο αποτέλεσμα και, συγχρόνως, κάθε εντολή έχει ήδη στη διάθεσή της το άλλο όρισμά της, τότε οι εντολές μπορούν να εκδοθούν ταυτόχρονα με τη μετάδοση του αποτελέσματος στον CDB. Αν χρησιμοποιούνταν μια συγκεντρωτική συστοιχία καταχωρητών, τότε οι μονάδες θα έπρεπε να διαβάζουν τα αποτελέσματά τους από τους καταχωρητές, όταν οι δίαυλοι καταχωρητών θα ήταν διαθέσιμοι.

Η εξάλειψη των κινδύνων WAW και WAR, που αποτελεί το δεύτερο πλεονέκτημα, ολοκληρώνεται τόσο μέσω της μετονομασίας των καταχωρητών οι οποίοι χρησιμοποιούν τους σταθμούς κράτησης όσο και διαμέσου της διαδικασίας αποθήκευσης των ορισμάτων στο σταθμό κράτησης, μόλις αυτά καταστούν διαθέσιμα.

Για παράδειγμα, η ακολουθία κώδικα του Σχήματος 2.10 εκδίδει τόσο την εντολή `DIV.D` όσο και την εντολή `ADD.D`, παρόλο που υφίσταται κίνδυνος WAR που περιλαμβάνει τον καταχωρητή F6. Ο κίνδυνος εξαλείφεται με έναν από τους δύο διαθέσιμους τρόπους. Πρώτον, αν έχει ολοκληρωθεί η εντολή που παρέχει την τιμή την οποία χρειάζεται η `DIV.D`, τότε το αποτέλεσμα αποθηκεύεται στο Vk, γεγονός που επιτρέπει στην `DIV.D` να εκτελεστεί ανεξαρτήτως του τι θα πράξει η `ADD.D` (αυτή είναι και η περίπτωση που παρουσιάζεται). Από την άλλη πλευρά, αν η εντολή `L.D` δεν είχε ολοκληρωθεί, τότε το Qk θα παρέπεμπε στο σταθμό κράτησης Load1 και η `DIV.D` θα ήταν ανεξάρτητη από την `ADD.D`. Κατά συνέπεια, σε κάθε περίπτωση, η `ADD.D` μπορεί να εκδοθεί και να ξεκινήσει να εκτελείται. Οποιαδήποτε χρήση του αποτελέσματος της `DIV.D` θα παρέπεμπε

	Κατάσταση εντολής		
Εντολή	**Έκδοση**	**Εκτέλεση**	**Εγγραφή αποτελέσματος**
L.D F6,32(R2)	√	√	√
L.D F2,44(R3)	√	√	
MUL.D F0,F2,F4	√		
SUB.D F8,F2,F6	√		
DIV.D F10,F0,F6	√		
ADD.D F6,F8,F2	√		

	Σταθμοί κράτησης							
Ονομασία	**Απασχολημένος**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**A**
Load1	όχι						
Load2	ναι	Load					45 + Regs[R3]
Add1	ναι	SUB		Mem[34 + Regs[R2]]	Load2		
Add2	ναι	ADD			Add1	Load2	
Add3	όχι						
Mult1	ναι	MUL		Regs[F4]	Load2		
Mult2	ναι	DIV		Mem[34 + Regs[R2]]	Mult1		

	Κατάσταση καταχωρητών								
Πεδίο	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
Qi	Mult1	Load2		Add2	Add1	Mult2			

Σχήμα 2.10 Παρουσιάζονται οι σταθμοί κράτησης και οι ετικέτες καταχωρητών αφότου όλες οι εντολές έχουν εκδοθεί, ωστόσο μόνο η πρώτη εντολή φόρτωσης έχει ολοκληρωθεί και εγγράψει το αποτέλεσμά της στον CDB. Η δεύτερη φόρτωση έχει ολοκληρώσει τον υπολογισμό της ενεργού διεύθυνσης, ωστόσο αναμένει τη μονάδα μνήμης. Χρησιμοποιούμε τον πίνακα Regs[] για να αναφερόμαστε στη συστοιχία καταχωρητών και τον πίνακα Mem[] για να αναφερόμαστε στη μνήμη. Ας έχουμε κατά νου ότι τα ορίσματα, ανά πάσα στιγμή, προσδιορίζονται είτε από το πεδίο V είτε από το πεδίο Q. Αξίζει να σημειωθεί ότι η εντολή `ADD.D`, η οποία προκαλεί κίνδυνο WAR στο στάδιο WB, έχει εκδοθεί και μπορεί να ολοκληρωθεί πριν την αρχικοποίηση της `DIV.D`.

στο σταθμό κράτησης, κάτι που θα επέτρεπε την ολοκλήρωση της `ADD.D` και την αποθήκευση της τιμής της στους καταχωρητές, χωρίς να επηρεάζεται η `DIV.D`.

Σύντομα, θα εξετάσουμε παράδειγμα που σχετίζεται με την εξάλειψη των κινδύνων WAW. Ας διερευνήσουμε, όμως, αρχικά τον τρόπο με τον οποίο συνεχίζεται η εκτέλεση του προηγούμενου παραδείγματος. Στο παράδειγμα αυτό, καθώς και σε αυτά που ακολουθούν στο κεφάλαιο αυτό, υποθέτουμε ότι ισχύουν οι παρακάτω λανθάνοντες χρόνοι: η φόρτωση έχει λανθάνοντα χρόνο της τάξης του 1 κύκλου ρολογιού, η πρόσθεση της τάξης των 2 κύκλων ρολογιού, ο πολ-

	Κατάσταση εντολών		
Εντολή	**Έκδοση**	**Εκτέλεση**	**Εγγραφή αποτελέσματος**
L.D F6,32(R2)	√	√	√
L.D F2,44(R3)	√	√	√
MUL.D F0,F2,F4	√	√	
SUB.D F8,F2,F6	√	√	√
DIV.D F10,F0,F6	√		
ADD.D F6,F8,F2	√	√	√

	Σταθμοί κράτησης							
Ονομασία	**Απασχολημένος**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**A**
Load1	όχι						
Load2	όχι						
Add1	όχι						
Add2	όχι						
Add3	όχι						
Mult1	ναι	MUL	Mem[45 + Regs[R3]]	Regs[F4]			
Mult2	ναι	DIV		Mem[34 + Regs[R2]]	Mult1		

	Κατάσταση καταχωρητών								
Πεδίο	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
Qi	Mult1					Mult2			

Σχήμα 2.11 Ο πολλαπλασιασμός και η διαίρεση είναι οι μόνες εντολές που δεν έχουν ολοκληρωθεί.

λαπλασιασμός της τάξης των 6 κύκλων ρολογιού και η διαίρεση της τάξης των 12 κύκλων ρολογιού.

Παράδειγμα: Χρησιμοποιώντας το ίδιο τμήμα κώδικα του προηγούμενου παραδείγματος (σελίδα 134), θα παρουσιάσουμε την εικόνα των πινάκων κατάστασης τη στιγμή που η `MUL.D` είναι έτοιμη να εγγράψει το αποτέλεσμά της.

Απάντηση: Το αποτέλεσμα παρουσιάζεται στους τρεις πίνακες του Σχήματος 2.11. Αξίζει να σημειωθεί ότι η `ADD.D` έχει ολοκληρωθεί, καθώς έχουν αντιγραφεί τα ορίσματα της `DIV.D` και ως εκ τούτου έχει υπερκεραστεί ο κίνδυνος WAR. Επίσης, αξίζει να επισημανθεί ότι ακόμα και αν καθυστερούσε η φόρτωση του F6, θα εκτελούνταν η πρόσθεση της οποίας το αποτέλεσμα θα αποθηκευόταν στον καταχωρητή F6, χωρίς αυτό να προκαλούσε κάποιον κίνδυνο WAR.

Αλγόριθμος του Tomasulo: Λεπτομερής Αναφορά

Στο Σχήμα 2.12 προσδιορίζονται οι έλεγχοι και τα βήματα από τα οποία πρέπει να διέλθει κάθε εντολή. Όπως έχει ήδη αναφερθεί, οι εντολές φόρτωσης και αποθήκευσης, πριν προχωρήσουν στους ανεξάρτητους απομονωτές φόρτωσης ή αποθήκευσης, διέρχονται από κάποια λειτουργική μονάδα για τον υπολογισμό της ενεργού διεύθυνσης. Οι εντολές φόρτωσης χρειάζονται και δεύτερο βήμα εκτέλεσης προκειμένου προσπελάσουν τη μνήμη και, εν συνεχεία, εισέρχονται στο στάδιο Εγγραφής Αποτελέσματος για την αποστολή της τιμής από τη μνήμη στη συστοιχία καταχωρητών και/ή σε οποιουσδήποτε σταθμούς κράτησης την αναμένουν. Οι εντολές αποθήκευσης ολοκληρώνουν την εκτέλεσή τους στο στάδιο Εγγραφής Αποτελέσματος, κατά το οποίο το αποτέλεσμα εγγράφεται στη μνήμη. Αξίζει να σημειωθεί ότι όλες οι ενέργειες εγγραφής πραγματοποιούνται κατά το στάδιο Εγγραφής Αποτελέσματος, ανεξαρτήτως του αν ο προορισμός είναι κάποιος καταχωρητής ή η μνήμη. Ο περιορισμός αυτός απλοποιεί τον αλγόριθμο του Tomasulo και, όπως θα διαπιστώσουμε στην Ενότητα 2.6, είναι ιδιαίτερα σημαντικός για την επέκτασή του ενσωματώνοντας την προσέγγιση της εικασίας.

Αλγόριθμος του Tomasulo: Παράδειγμα Βρόχου

Για την κατανόηση της σπουδαιότητας που έχει η εξάλειψη των κινδύνων WAW και WAR μέσω της δυναμικής μετονομασίας των καταχωρητών, πρέπει να μελετήσουμε την περίπτωση των βρόχων. Ας εξετάσουμε την παρακάτω απλή ακολουθία εντολών για τον πολλαπλασιασμό των στοιχείων ενός πίνακα με μια βαθμωτή τιμή που βρίσκεται στον καταχωρητή F2:

```
Loop:   L.D     F0,0(R1)
        MUL.D   F4,F0,F2
        S.D     F4,0(R1)
        DADDIU  R1,R1,-8
        BNE     R1,R2,Loop ; διακλάδωση αν R1≠R2
```

Αν προβούμε στην πρόβλεψη ότι οι διακλαδώσεις ακολουθούνται, τότε η χρήση των σταθμών κράτησης επιτρέπει την ταυτόχρονη πραγματοποίηση πολλαπλών εκτελέσεων του βρόχου αυτού. Το όφελος αυτό προκύπτει χωρίς να χρειάζεται η αλλαγή του κώδικα - στην πραγματικότητα, ο βρόχος ξεδιπλώνεται δυναμικά από το υλικό, μέσω μεταχείρισης των σταθμών κράτησης που χρησιμοποιούνται κατά τη διαδικασία της μετονομασίας ως επιπρόσθετων καταχωρητών.

Ας υποθέσουμε ότι έχουμε εκδώσει όλες τις εντολές που περιλαμβάνονται σε δύο διαδοχικές επαναλήψεις του βρόχου, ωστόσο δεν έχει ολοκληρωθεί καμία λειτουργία φόρτωσης-αποθήκευσης ή κινητής υποδιαστολής. Το Σχήμα 2.13 παρουσιάζει τους σταθμούς κράτησης, τους πίνακες κατάστασης καταχωρητών και τους απομονωτές φόριωσης και αποθήκευσης σε αυτήν τη χρονική στιγμή

Κατάσταση εντολής	Αναμονή έως ότου	Ενέργεια ή καταγραφή

Εκδοση
Λειτουργία FP
	Ο σταθμός r να είναι κενός	if (RegisterStat[rs].Qi ≠0)
		{RS[r].Qj ← RegisterStat[rs].Qi}
		else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0};
		if (RegisterStat[rt].Qi ≠0)
		{RS[r].Qk ← RegisterStat[rt].Qi
		else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};
		RS[r].Busy ← yes; RegisterStat[rd].Q ← r;

Φόρτωση ή αποθήκευση	Ο απομονωτής r να είναι κενός	if (RegisterStat[rs].Qi ≠0) {RS[r].Qj ← RegisterStat[rs].Qi} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0}; RS[r].A ← imm; RS[r].Busy ← yes;
Μόνο φόρτωση		RegisterStat[rt].Qi ← r;
Μόνο αποθήκευση		if (RegisterStat[rt].Qi ≠0) {RS[r].Qk ← RegisterStat[rs].Qi} else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};

Σχήμα 2.12 **Βήματα του αλγορίθμου και περιγραφή των απαιτήσεων κάθε βήματος.** Σε ό,τι αφορά την εντολή που εκδίδεται, ο `rd` αποτελεί τον αριθμό του καταχωρητή προορισμού, τα `rs` και `rt` αποτελούν τους αριθμούς των καταχωρητών αφετηρίας, το `imm` αποτελεί το άμεσο πεδίο στο οποίο έχει εφαρμοσιεί επέκταση προσήμου και ο `r` αποτελεί το σταθμό κράτησης ή τον απομονωτή, ο οποίος έχει ανατεθεί στην εντολή. Η `RS` αποτελεί τη δομή δεδομένων των σταθμών κράτησης. Η τιμή που επιστρέφεται από τη μονάδα FP ή από τη μονάδα φόρτωσης ονομάζεται `result`. Η `RegisterStat` αποτελεί τη δομή δεδομένων της κατάστασης των καταχωρητών (όχι τη συστοιχία καταχωρητών η οποία είναι η `Regs[]`). Όταν εκδίδεται κάποια εντολή, ο καταχωρητής προορισμού θέτει στο πεδίο Qi τον αριθμό του απομονωτή ή του σταθμού κράτησης στον οποίο έχει εκδοθεί η εντολή. Αν τα ορίσματα είναι διαθέσιμα εντός των καταχωρητών, τότε αποθηκεύονται στα πεδία V. Διαφορετικά, τα πεδία Q λαμβάνουν τιμή η οποία υποδεικνύει το σταθμό κράτησης ο οποίος θα παραγάγει τις τιμές που χρειάζονται ως ορίσματα αφετηρίας. Η εντολή βρίσκεται σε αναμονή στο σταθμό κράτησης, μέχρις ότου να είναι διαθέσιμα και τα δύο ορίσματά της, κάτι που επισημαίνεται τοποθετώντας την τιμή μηδέν στα πεδία Q. Τα πεδία Q λαμβάνουν μηδενική τιμή είτε όταν εκδίδεται η εντολή αυτή είτε όταν κάποια εντολή, από την οποία εξαρτάται η εντολή αυτή, ολοκληρώνεται και εκτελεί το στάδιο επανεγγραφής. Όταν κάποια εντολή έχει ολοκληρώσει την εκτέλεση της και συγχρόνως ο CDB είναι διαθέσιμος, τότε μπορεί να προβεί στην εκτέλεση του σταδίου επανεγγραφής. Όλοι οι απομονωτές, οι καταχωρητές και οι σταθμοί κράτησης, των οποίων τα πεδία Qj ή Qk έχουν τιμή ίδια με αυτή του σταθμού κράτησης που ολοκληρώνει την εκτέλεσή του, ενημερώνουν τις τιμές τους μέσω του CDB και θέτουν στα πεδία Q την κατάλληλη τιμή, έτσι ώστε να καταδεικνύεται ότι οι τιμές αυτές έχουν ληφθεί. Επομένως, ο CDB μπορεί να μεταδίδει το αποτέλεσμά του προς πολλούς προορισμούς εντός ενός μόνο κύκλου ρολογιού και, σε περίπτωση, που οι εντολές που βρίσκονται σε αναμονή διαθέτουν τα ορίσματά τους, τότε μπορούν όλες να ξεκινήσουν την εκτέλεσή τους στον επόμενο κύκλο ρολογιού. Οι εντολές φόρτωσης πραγματοποιούν δύο βήματα κατά το στάδιο Εκτέλεσης, ενώ οι εντολές αποθήκευσης λειτουργούν με ελαφρά διαφορετικό τρόπο κατά τη διάρκεια του σταδίου Εγγραφής Αποτελέσματος, κατά το οποίο ενδέχεται να πρέπει να αναμείνουν την αποθήκευση της τιμής. Ας έχουμε κατά νου ότι για τη διατήρηση της συμπεριφοράς των εξαιρέσεων, δεν πρέπει να επιτρέπεται η εκτέλεση των εντολών, αν δεν έχει ολοκληρωθεί η εκτέλεση κάποιας διακλάδωσης η οποία προηγείται των εντολών αυτών στη σειρά προγράμματος. Επειδή η έννοια της σειράς προγράμματος παύει να ισχύει μετά το στάδιο έκδοσης, στην περίπτωση που υπάρχει κάποια εκκρεμής διακλάδωση εντός της διασωλήνωσης, ο περιορισμός αυτός συνήθως υλοποιείται μέσω της παρεμπόδισης της εξόδου κάθε εντολής από το βήμα Έκδοσης. Στην Ενότητα 2.6, θα εξετάσουμε τον τρόπο με τον οποίο η προσέγγιση της εικασίας οδηγεί στην εξάλειψη του περιορισμού αυτού.

Εκτέλεση Λειτουργία FP	(RS[r].Qj = 0) και (RS[r].Qk = 0)	Υπολογισμός αποτελέσματος: τα ορίσματα βρίσκονται στους Vj και Vk
Φόρτωση-αποθήκευση πρώτο βήμα	RS[r].Qj = 0 & o r βρίσκεται στην κεφαλή της ουράς φόρτωσης-αποθήκευσης	RS[r].A ← RS[r].Vj + RS[r].A;
Δεύτερο βήμα φόρτωσης	Ολοκλήρωση πρώτου βήματος φόρτωσης	Ανάγνωση από Mem[RS[r].A]
Εγγραφή αποτελέσματος Λειτουργία FP ή φόρτωση	Η εκτέλεση έχει ολοκληρωθεί στον r & ο CDB είναι διαθέσιμος	∀x(if (RegisterStat[x].Qi=r) {Regs[x] ← result; RegisterStat[x].Qi ← 0}); ∀x(if (RS[x].Qj=r) {RS[x].Vj ← result;RS[x].Qj ← 0}); ∀x(if (RS[x].Qk=r) {RS[x].Vk ← result;RS[x].Qk ← 0}); RS[r].Busy ← no;
Αποθήκευση	Η εκτέλεση έχει ολοκληρωθεί στον r & RS[r].Qk = 0	Mem[RS[r].A] ← RS[r].Vk; RS[r].Busy ← no;

Σχήμα 2.12 Βήματα του αλγορίθμου και περιγραφή των απαιτήσεων κάθε βήματος (Συνέχεια)

(η πράξη της ALU με ακεραίους δεν λαμβάνεται υπόψη, ενώ συγχρόνως υποθέτουμε ότι η διακλάδωση προβλέφθηκε ότι θα ακολουθηθεί). Μόλις το σύστημα φτάσει στο σημείο αυτό, είναι εφικτή η υποστήριξη δύο αντιγράφων του βρόχου έχοντας CPI που προσεγγίζει τη μονάδα, με την προϋπόθεση ότι οι πολλαπλασιασμοί μπορούν να ολοκληρωθούν εντός 4 κύκλων ρολογιού. Έχοντας λανθάνοντα χρόνο της τάξης των 6 κύκλων, χρειάζεται η επεξεργασία των επιπρόσθετων επαναλήψεων πριν να είναι εφικτό να φτάσουμε στη σταθερή κατάσταση. Αυτό απαιτεί τη χρήση περισσότερων σταθμών κράτησης για τη διατήρηση εντολών οι οποίες εκτελούνται. Όπως θα διαπιστώσουμε αργότερα στο κεφάλαιο αυτό, η προσέγγιση του Tomasulo, με την ενσωμάτωση της δυνατότητας της πολλαπλής έκδοσης εντολών, θα είναι σε θέση να υποστηρίζει περισσότερες από μία εντολές ανά κύκλο.

Οι εντολές φόρτωσης και αποθήκευσης μπορούν να εκτελούνται με ασφάλεια εκτός σειράς, με την προϋπόθεση ότι προσπελάζουν διαφορετικές διευθύνσεις. Αν κάποια εντολή φόρτωσης και κάποια εντολή αποθήκευσης προσπελάζουν την ίδια διεύθυνση τότε είτε

- η εντολή φόρτωσης προηγείται στη σειρά προγράμματος της εντολής αποθήκευσης και επομένως η αντιμετάθεσή τους έχει ως αποτέλεσμα την εμφάνιση κινδύνου WAR είτε
- η εντολή αποθήκευσης προηγείται στη σειρά προγράμματος της εντολής φόρτωσης και επομένως η αντιμετάθεσή τους έχει ως αποτέλεσμα την εμφάνιση κινδύνου RAW.

Παρόμοια, η αντιμετάθεση δύο εντολών αποθήκευσης που προσπελάζουν την ίδια διεύθυνση οδηγεί στην εμφάνιση κινδύνου WAW.

		Κατάσταση εντολών			
Εντολή		Από επανάληψη	Έκδοση	Εκτέλεση	Εγγραφή αποτελέσματος
L. D	F0, 0(R1)	1	√	√	
ML. D	F4, F0, F2	1	√		
S. D	F4, 0(R1)	1	√		
L. D	F0, 0(R1)	2	√	√	
ML. D	F4, F0, F2	2	√		
S. D	F4, 0(R1)	2	√		

		Σταθμοί κράτησης					
Ονομασία	Απασχολημένος	Op	Vj	Vk	Qj	Qk	A
Load1	ναι	Load					Regs[R1] + 0
Load2	ναι	Load					Regs[R1] – 8
Add1	όχι						
Add2	όχι						
Add3	όχι						
Mult1	ναι	MUL		Regs[F2]	Load1		
Mult2	ναι	MUL		Regs[F2]	Load2		
Store1	ναι	Store	Regs[R1]			Mult1	
Store2	ναι	Store	Regs[R1] – 8			Mult2	

	Κατάσταση καταχωρητών								
Πεδίο	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Load2		Mult2						

Σχήμα 2.13 Δύο ενεργές επαναλήψεις του βρόχου, χωρίς να έχει ολοκληρωθεί ακόμα καμία εντολή. Οι καταχωρίσεις των σταθμών κράτησης υποδηλώνουν ότι οι εκκρεμείς λειτουργίες φόρτωσης αποτελούν τα ορίσματα αφετηρίας. Οι σταθμοί κράτησης των εντολών αποθήκευσης υποδηλώνουν ότι ο προορισμός του πολλαπλασιασμού αποτελεί την αφετηρία της τιμής που πρέπει να αποθηκευτεί.

Κατά συνέπεια, για να προσδιοριστεί αν κάποια εντολή φόρτωσης μπορεί να εκτελεστεί τη δεδομένη χρονική στιγμή, ο επεξεργαστής μπορεί να ελέγξει αν υπάρχει κάποια μη ολοκληρωμένη εντολή αποθήκευσης η οποία αφενός προηγείται στη σειρά προγράμματος της εντολής φόρτωσης και αφετέρου διαμοιράζεται την ίδια διεύθυνση μνήμης με αυτή. Παρόμοια, η εντολή αποθήκευσης πρέπει να βρίσκεται σε αναμονή έως ότου να μην υφίστανται εντολές φόρτωσης ή αποθήκευσης οι οποίες προηγούνται στη σειρά προγράμματος, διαμοιράζονται την ίδια διεύθυνση μνήμης και δεν έχουν ακόμα ολοκληρωθεί. Στην Ενότητα 2.9, εξετάζουμε *μέθοδο εξάλειψης του περιορισμού αυτού*.

Για την ανίχνευση των κινδύνων αυτού του είδους, ο επεξεργαστής πρέπει να έχει υπολογίσει τις διευθύνσεις της μνήμης δεδομένων που σχετίζονται με οποιαδήποτε προηγούμενη λειτουργία της μνήμης. Ένας απλός, αλλά όχι απαραίτητα βέλτιστος, τρόπος για να διασφαλιστεί ότι ο επεξεργαστής έχει στη διάθεσή του όλες αυτού του είδους τις διευθύνσεις σχετίζεται με την εκτέλεση των υπολογισμών των ενεργών διευθύνσεων ακολουθώντας τη σειρά προγράμματος (στην πραγματικότητα χρειάζεται μόνο να διατηρούμε τη σχετική σειρά μεταξύ των εντολών αποθήκευσης και των υπόλοιπων αναφορών στη μνήμη, κάτι που σημαίνει ότι οι εντολές φόρτωσης μπορούν ελεύθερα να αναδιατάσσονται).

Ας εξετάσουμε πρώτα την περίπτωση της φόρτωσης. Αν εκτελέσουμε τον υπολογισμό της ενεργού διεύθυνσης με τη σειρά προγράμματος, τότε, όταν η φόρτωση ολοκληρώσει τον υπολογισμό της ενεργού διεύθυνσης, μπορούμε να ελέγξουμε αν υφίσταται κάποια σύγκρουση διευθύνσεων, εξετάζοντας το πεδίο Α όλων των ενεργών απομονωτών αποθήκευσης. Αν η διεύθυνση φόρτωσης είναι ίδια με οποιαδήποτε ενεργό καταχώριση του απομονωτή αποθήκευσης, τότε αυτή η εντολή φόρτωσης δεν αποστέλλεται στον απομονωτή φόρτωσης, πριν την ολοκλήρωση της αποθήκευσης που προκαλεί τη σύγκρουση (μερικές υλοποιήσεις προωθούν την τιμή κάποιας εκκρεμούς εντολής αποθήκευσης απευθείας στην εντολή φόρτωσης, μειώνοντας με τον τρόπο αυτό την καθυστέρηση αυτού του κινδύνου RAW).

Οι εντολές αποθήκευσης λειτουργούν με παρόμοιο τρόπο, με τη διαφορά ότι ο επεξεργαστής πρέπει να διενεργεί έλεγχο για την εύρεση συγκρούσεων τόσο στους απομονωτές φόρτωσης όσο και στους απομονωτές αποθήκευσης, καθώς οι εντολές αποθήκευσης που προκαλούν κάποια σύγκρουση δεν μπορούν να αναδιαταχθούν σε σχέση με τη θέση είτε κάποιας εντολής φόρτωσης είτε κάποιας εντολής αποθήκευσης.

Η διασωλήνωση δυναμικής χρονοδρομολόγησης μπορεί να επιδείξει ιδιαίτερα υψηλή απόδοση, με την προϋπόθεση ότι οι διακλαδώσεις προβλέπονται με ακρίβεια - ζήτημα που εξετάσαμε στην προηγούμενη ενότητα. Το κύριο μειονέκτημα της μεθόδου αυτής είναι η πολυπλοκότητα της προσέγγισης Tomasulo, η οποία απαιτεί τη χρήση υλικού υψηλών δυνατοτήτων. Συγκεκριμένα, κάθε σταθμός κράτησης πρέπει να περιέχει έναν σχετικό με αυτόν απομονωτή, οποίος με τη σειρά του πρέπει να λειτουργεί σε υψηλή ταχύτητα, ενώ συγχρόνως πρέπει να διαθέτει και πολύπλοκη λογική ελέγχου. Η απόδοση μπορεί, επίσης, να περιορίζεται από την ύπαρξη ενός μόνο CDB. Παρόλο που είναι εφικτή η προσθήκη επιπλέον CDBs, κάθε CDB πρέπει να αλληλεπιδρά με καθένα από τους σταθμούς κράτησης και, επομένως, θα έπρεπε να υπάρχουν αντίγραφα του υλικού που είναι αρμόδιο για την αντιστοίχιση των ετικετών σε κάθε σταθμό για καθένα από τα CDBs.

Στην προσέγγιση του Tomasulo υπάρχει συνδυασμός δύο διαφορετικών τεχνικών: της μετονομασίας των καταχωρητών της αρχιτεκτονικής με τη βοήθεια ενός μεγαλύτερου συνόλου καταχωρητών και της αποθήκευσης των ορισμάτων

αφετηρίας, που βρίσκονται στη συστοιχία καταχωρητών, σε απομονωτές. Η αποθήκευση των ορισμάτων αφετηρίας σε απομονωτές έχει ως αποτέλεσμα την αντιμετώπιση των κινδύνων WAR, οι οποίοι προκύπτουν όταν το όρισμα είναι διαθέσιμο στους καταχωρητές. Όπως θα διαπιστώσουμε αργότερα, είναι, επίσης, εφικτή η εξάλειψη των κινδύνων WAR μέσω της μετονομασίας ενός καταχωρητή και της αποθήκευσης του αποτελέσματος σε κάποιον απομονωτή, έως ότου να μην υφίσταται καμία εκκρεμής αναφορά στην προηγούμενη έκδοση του καταχωρητή. Η προσέγγιση αυτή θα χρησιμοποιηθεί κατά την εξέταση της εικασίας υλικού.

Η προσέγγιση του Tomasulo δεν χρησιμοποιούνταν για πολλά χρόνια μετά από την ανάπτυξη του 360/91, ωστόσο υιοθετήθηκε ευρέως από τους επεξεργαστές πολλαπλής έκδοσης στις αρχές της δεκαετίας του 1990 για διάφορους λόγους:

1. Μπορεί να σημειώσει υψηλή απόδοση χωρίς να απαιτείται από το μεταγλωττιστή να διαμορφώνει τον κώδικα στα μέτρα και σταθμά κάποιας συγκεκριμένης δομής διασωλήνωσης, κάτι που αποτελεί ιδιαίτερα χρήσιμη υπηρεσία στη σημερινή εποχή της μαζικής αγοράς λογισμικού σε μορφή συσκευασίας.

2. Παρόλο που ο αλγόριθμος του Tomasulo σχεδιάστηκε πριν τη χρήση της κρυφής μνήμης, η παρουσία της κρυφής μνήμης, με τις εγγενείς μη αναμενόμενες καθυστερήσεις που υφίστανται σε αυτή, έχει αποτελέσει ένα από τα πιο σημαντικά κίνητρα για την υιοθέτηση της δυναμικής χρονοδρομολόγησης. Η εκτέλεση εκτός σειράς επιτρέπει στους επεξεργαστές να συνεχίζουν να εκτελούν εντολές, ενώ αναμένουν την ολοκλήρωση κάποιας αστοχίας της κρυφής μνήμης, κάτι που ως εκ τούτου οδηγεί στην εξάλειψη της ποινής που σχετίζεται με την αστοχία της κρυφής μνήμης.

3. Όσο οι επεξεργαστές αποκτούν όλο και περισσότερες δυνατότητες έκδοσης εντολών και οι σχεδιαστές ενδιαφέρονται για ζητήματα που σχετίζονται με την απόδοση τμημάτων κώδικα που είναι δύσκολο να χρονοδρομολογηθούν (όπως οι περισσότερες περιπτώσεις μη αριθμητικού κώδικα), οι προσεγγίσεις, όπως αυτή της μετονομασίας των καταχωρητών και της δυναμικής χρονοδρομολόγησης, αποκτούν ολοένα και μεγαλύτερη σπουδαιότητα.

4. Η δυναμική χρονοδρομολόγηση, επειδή αποτελεί ένα από τα βασικά στοιχεία της εικασίας, έχει υιοθετηθεί, σε συνδυασμό με την εικασία υλικού, από τα μέσα της δεκαετίας του 1990.

2.6 Εικασία που Βασίζεται στο Υλικό

Καθώς προσπαθούμε να αξιοποιήσουμε ολοένα και περισσότερο τον παραλληλισμό επιπέδου εντολών, η διατήρηση των εξαρτήσεων ελέγχου αποτελεί σημαντική επιβάρυνση. Η πρόβλεψη διακλάδωσης μειώνει τα άμεσα διαστήματα αδράνειας που σχετίζονται με τις διακλαδώσεις, ωστόσο για τους επεξεργαστές που εκτελούν πολλαπλές εντολές ανά κύκλο, η ακριβής μονάχα πρόβλεψη των διακλαδώσεων μπορεί να μην επαρκεί για την επίτευξη του επιθυμητού εύρους του παραλληλισμού επιπέδου εντολών. Οι επεξεργαστές ευρείας έκδοσης μπορεί να χρειάζεται να εκτελούν μια διακλάδωση σε κάθε κύκλο ρολογιού για τη διατήρηση της μέγιστης απόδοσης. Κατά συνέπεια, η αξιοποίηση του παραλληλισμού ολοένα και περισσότερο προϋποθέτει την επιτυχή αντιμετώπιση του περιορισμού της εξάρτησης ελέγχου.

Η αντιμετώπιση της εξάρτησης ελέγχου υλοποιείται κάνοντας μια εικασία για το αποτέλεσμα των διακλαδώσεων και εκτελώντας το πρόγραμμα υποθέτοντας ότι η πρόβλεψή μας είναι ορθή. Ο μηχανισμός αυτός αναπαριστά μια δυσδιάκριτη, αλλά σημαντική, επέκταση της χρήσης της δυναμικής χρονοδρομολόγησης και της πρόβλεψης διακλάδωσης. Συγκεκριμένα, στην εικασία, οι εντολές ανακτώνται, εκδίδονται και *εκτελούνται* υποθέτοντας ότι οι προβλέψεις των διακλαδώσεων είναι πάντοτε ορθές. Αντιθέτως, η δυναμική χρονοδρομολόγηση το μόνο στο οποίο προβαίνει είναι η ανάκτηση και έκδοση των εντολών αυτών. Βεβαίως, χρειαζόμαστε μηχανισμούς για την αντιμετώπιση της περίπτωσης κατά την οποία η εικασία είναι λανθασμένη. Το Παράρτημα Ζ εξετάζει διάφορους μηχανισμούς για την υποστήριξη της εικασίας από το μεταγλωττιστή. Στην ενότητα αυτή, εξετάζουμε την *εικασία υλικού* (hardware speculation), η οποία επεκτείνει τις έννοιες της δυναμικής χρονοδρομολόγησης.

Η εικασία που βασίζεται στο υλικό συνδυάζει τρεις ιδιαίτερα σημαντικές ιδέες: τη δυναμική πρόβλεψη διακλάδωσης για την επιλογή των εντολών που εκτελούνται, την εικασία για την παροχή της δυνατότητας εκτέλεσης εντολών πριν την επίλυση των εξαρτήσεων ελέγχου (σε συνδυασμό με τη δυνατότητα αναίρεσης των ακολουθιών που είναι αποτέλεσμα λανθασμένης εικασίας) και τη δυναμική χρονοδρομολόγηση για την αντιμετώπιση του ζητήματος της χρονοδρομολόγησης διαφορετικών συνδυασμών από βασικά blocks (αντιθέτως, η δυναμική χρονοδρομολόγηση χωρίς τη χρήση της εικασίας οδηγεί μόνο στην εν μέρει επικάλυψη των βασικών blocks, διότι απαιτεί την πραγματοποίηση της λήψης της απόφασης σχετικά με την κατεύθυνση της διακλάδωσης πριν την υλοποίηση της εκτέλεσης οποιασδήποτε εντολής ανήκει στο αμέσως επόμενο βασικό block).

Η εικασία που βασίζεται στο υλικό ακολουθεί την προβλεπόμενη ροή των τιμών των δεδομένων για την επιλογή του χρονικού σημείου εκτέλεσης των εντολών. Αυτή η μέθοδος εκτέλεσης προγραμμάτων αποτελεί στην ουσία μια *εκτέλεση ροής δεδομένων* (data flow execution): Οι λειτουργίες εκτελούνται μόλις τα

ορίσματά τους καθίστανται διαθέσιμα.

Για να προεκτείνουμε τον αλγόριθμο του Tomasulo έτσι ώστε να υποστηρίζει την εικασία, πρέπει να διαχωρίσουμε την προώθηση των αποτελεσμάτων μεταξύ των εντολών, η οποία είναι απαραίτητη για την εκτέλεση της εντολής ακολουθώντας την προσέγγιση της εικασίας, από την πραγματική ολοκλήρωση της εντολής. Με την πραγματοποίηση του διαχωρισμού αυτού, μπορούμε να επιτρέπουμε την εκτέλεση κάποιας εντολής, καθώς και την προώθηση των αποτελεσμάτων της σε άλλες εντολές, χωρίς να παρέχουμε τη δυνατότητα στην εντολή να πραγματοποιεί ενημερώσεις οι οποίες δεν μπορούν να αναιρεθούν, πριν να γνωρίζουμε ότι η εντολή δεν είναι πλέον υποθετική.

Η χρήση της προωθημένης τιμής είναι ισοδύναμη με την υποθετική ανάγνωση κάποιου καταχωρητή, καθώς δεν γνωρίζουμε αν η εντολή, που παρέχει την τιμή του ορίσματος αφετηρίας, παρέχει όντως την ορθή τιμή, μέχρις ότου η εντολή αυτή να πάψει να είναι υποθετική. Όταν η εντολή πάψει να είναι υποθετική, έχει τη δυνατότητα να ενημερώσει τη συστοιχία καταχωρητών ή τη μνήμη. Το επιπρόσθετο αυτό βήμα της ακολουθίας εκτέλεσης των εντολών ονομάζεται *επικύρωση εντολών* (instruction commit).

Η κεντρική ιδέα γύρω από την υλοποίηση της εικασίας είναι η παροχή της δυνατότητας στις εντολές να εκτελούνται εκτός σειράς, όντας ωστόσο υποχρεωμένες να επικυρώνονται *εντός σειράς* και να παρεμποδίζουν την εκτέλεση κάθε αμετάκλητης ενέργειας (όπως η ενημέρωση της κατάστασης ή η λήψη κάποιας εξαίρεσης) μέχρι την επικύρωση της εντολής. Επομένως, όταν προστίθεται η εικασία, πρέπει να διαχωρίζεται η διαδικασία της ολοκλήρωσης της εκτέλεσης από την επικύρωση της εκτέλεσης, καθώς οι εντολές ενδέχεται να ολοκληρώνουν την εκτέλεσή τους πολύ πιο πριν από τη χρονική στιγμή κατά την οποία θα είναι σε θέση να επικυρωθούν. Η προσθήκη αυτής της φάσης επικύρωσης στην ακολουθία εκτέλεσης των εντολών απαιτεί ένα επιπλέον σύνολο απομονωτών υλικού, στους οποίους διατηρούνται τα αποτελέσματα των εντολών που έχουν ολοκληρώσει την εκτέλεσή τους, χωρίς ωστόσο να έχουν επικυρωθεί. Αυτός ο απομονωτής υλικού, που ονομάζεται *απομονωτής αναδιάταξης* (reorder buffer, ROB), χρησιμοποιείται, επίσης, για τη μεταβίβαση των αποτελεσμάτων μεταξύ των εντολών, που ενδέχεται να είναι υποθετικές.

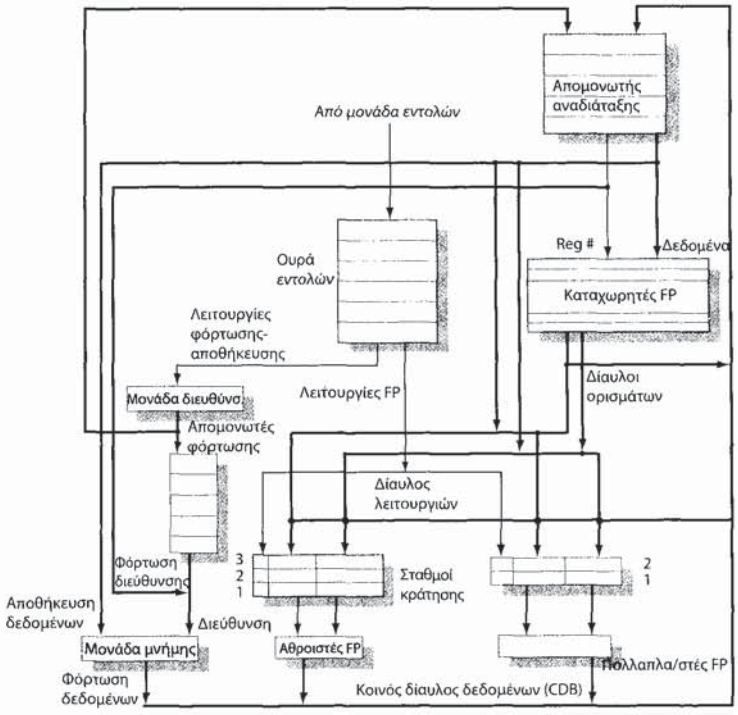
Ο απομονωτής αναδιάταξης παρέχει επιπρόσθετους καταχωρητές με τον ίδιο τρόπο που οι σταθμοί κράτησης του αλγορίθμου του Tomasulo επεκτείνουν το σύνολο των καταχωρητών. Ο ROB διατηρεί το αποτέλεσμα κάποιας εντολής για το χρονικό διάστημα που μεσολαβεί ανάμεσα στην ολοκλήρωση της λειτουργίας που σχετίζεται με την εντολή αυτή και την επικύρωσή της. Συνεπώς, ο ROB αποτελεί αφετηρία ορισμάτων για τις εντολές, με τον ίδιο ακριβώς τρόπο που παρέχουν τα ορίσματα οι σταθμοί κράτησης του αλγορίθμου του Tomasulo. Η βασική διαφορά έγκειται στο γεγονός ότι, στον αλγόριθμο του Tomasulo, μόλις κάποια εντολή εγγράψει το αποτέλεσμά της, τότε οποιεσδήποτε εντολές εκδίδονται μεταγενέστερα μπορούν να βρουν το αποτέλεσμα στη συστοιχία κα-

ταχωρητών. Στην εικασία, η συστοιχία καταχωρητών δεν ενημερώνεται πριν την επικύρωση της εντολής (και πριν να γνωρίζουμε οριστικά ότι η εντολή πρέπει να εκτελεστεί). Κατά συνέπεια, ο ROB παρέχει τα ορίσματα κατά το διάστημα που μεσολαβεί ανάμεσα στην ολοκλήρωση της εκτέλεσης της εντολής και την επικύρωσή της. Ο ROB είναι παρόμοιος με τον απομονωτή αποθήκευσης του αλγορίθμου του Tomasulo και για λόγους απλότητας έχουμε ενσωματώσει τη λειτουργία του απομονωτή αποθήκευσης στον ROB.

Κάθε καταχώριση του ROB περιέχει τέσσερα πεδία: τον τύπο εντολής, το πεδίο προορισμού, το πεδίο τιμής και το πεδίο ετοιμότητας. Το πεδίο του τύπου εντολής (instruction type) υποδηλώνει αν η εντολή είναι διακλάδωση (και δεν έχει κανένα αποτέλεσμα προορισμού), αποθήκευση (η οποία έχει μια διεύθυνση μνήμης προορισμού) ή λειτουργία καταχωρητή (πράξη ή φόρτωση της ALU, περιπτώσεις στις οποίες υπάρχουν καταχωρητές προορισμού). Το πεδίο προορισμού (destination field) παρέχει είτε τον αριθμό του καταχωρητή (στην περίπτωση λειτουργιών φόρτωσης και σε πράξεων της ALU) είτε τη διεύθυνση μνήμης (στην περίπτωση των εντολών αποθήκευσης), στην οποία πρέπει να εγγραφεί το αποτέλεσμα της εντολής. Το πεδίο τιμής (value field) χρησιμοποιείται για τη διατήρηση της τιμής του αποτελέσματος της εντολής μέχρι την επικύρωσή της. Σύντομα, θα εξετάσουμε σχετικό παράδειγμα όπου παρουσιάζονται οι καταχωρίσεις του ROB. Τέλος, το πεδίο ετοιμότητας (ready field) υποδηλώνει ότι η εντολή έχει ολοκληρώσει την εκτέλεσή της και ότι η τιμή είναι έτοιμη.

Το Σχήμα 2.14 παρουσιάζει τη δομή του υλικού του επεξεργαστή που περιλαμβάνει τον ROB. Οι καταχωρητές αποθήκευσης υπάγονται στον ROB. Οι εντολές αποθήκευσης εξακολουθούν να εκτελούνται σε δύο βήματα, ωστόσο το δεύτερο βήμα εκτελείται κατά την επικύρωση της εντολής. Παρόλο που η λειτουργία της μετονομασίας που εκτελούνταν με τη βοήθεια των σταθμών κράτησης έχει αντικατασταθεί από τον ROB, εξακολουθούμε να χρειαζόμαστε ένα χώρο στον οποίο θα αποθηκεύονται προσωρινά οι εντολές (και τα ορίσματα) για το χρονικό διάστημα που μεσολαβεί ανάμεσα στην έκδοση και την έναρξη της εκτέλεσής τους. Η λειτουργία αυτή εξακολουθεί να παρέχεται από τους σταθμούς κράτησης. Καθώς κάθε εντολή, μέχρι να επικυρωθεί, καταλαμβάνει μία θέση στον ROB, τα αποτελέσματα επισημαίνονται με τη βοήθεια του αριθμού καταχώρισης του ROB και όχι με τη συνδρομή του αριθμού του σταθμού κράτησης. Αυτή η διαδικασία επισήμανσης προϋποθέτει την παρακολούθηση του ROB, ο οποίος έχει ανατεθεί σε κάποια εντολή, από το σταθμό κράτησης. Αργότερα, στην ενότητα αυτή θα εξερευνήσουμε μια εναλλακτική υλοποίηση, η οποία χρησιμοποιεί επιπλέον καταχωρητές για τη μετονομασία και τον ROB μόνο για την ανίχνευση του χρονικού σημείου κατά το οποίο οι εντολές μπορούν να επικυρωθούν.

Ακολούθως παρατίθενται τα τέσσερα βήματα που περιλαμβάνονται στην εκτέλεση των εντολών:



Σχήμα 2.14 **Η βασική δομή της μονάδας FP χρησιμοποιώντας τον αλγόριθμο του Tomasulo, η οποία έχει επεκταθεί έτσι ώστε να υποστηρίζει την εικασία.** Συγκρίνοντας το σχήμα αυτό με το Σχήμα 2.9 στη σελίδα 130, στο οποίο υλοποιείται ο αλγόριθμος του Tomasulo, μπορούμε να διακρίνουμε ότι η βασική διαφοροποίηση είναι η προθήκη του ROB και η εξάλειψη του απομονωτή αποθήκευσης, η λειτουργικότητα του οποίου έχει ενσωματωθεί στον ROB. Ο μηχανισμός αυτός μπορεί να επεκταθεί ώστε να υποστηρίζει την πολλαπλή έκδοση, διευρύνοντας τον CDB, έτσι ώστε να παρέχεται η δυνατότητα ολοκλήρωσης πολλαπλών εντολών ανά κύκλο.

1. *Έκδοση*: Παραλαβή κάποιας εντολής από την ουρά των εντολών. Έκδοση της εντολής στην περίπτωση που υπάρχει τόσο κάποιος κενός σταθμός κράτησης όσο και μια κενή θέση στον ROB. Αποστολή των ορισμάτων στο σταθμό κράτησης, στην περίπτωση που αυτά είναι διαθέσιμα είτε εντός των καταχωρητών είτε εντός του ROB. Ενημέρωση των καταχωρίσεων ελέγχου, έτσι ώστε να υποδηλώνουν ότι οι απομονωτές χρησιμοποιούνται. Ο αριθμός της καταχώρισης του ROB που έχει δεσμευτεί για το αποτέλεσμα αποστέλλεται, επίσης, στο σταθμό κράτησης, έτσι ώστε ο αριθμός αυτός να μπορεί να χρησιμοποιηθεί για την επισήμανση του αποτελέσματος, όταν τοποθετηθεί στον CDB. Σε περίπτωση που είτε όλοι οι σταθμοί κράτησης είτε ο ROB είναι πλήρης, τότε η έκδοση της εντολής τίθεται σε αδράνεια, έως ότου και οι δύο παραπάνω χώροι να έχουν διαθέσιμες θέσεις.

2. *Εκτέλεση*: Αν ένα ή περισσότερα από τα ορίσματα δεν είναι ακόμα διαθέσιμα, τότε πραγματοποιείται παρακολούθηση του CDB για το χρονικό διάστημα που υφίσταται αναμονή για τον υπολογισμό του καταχωρητή. Στο βήμα αυτό διενεργείται ο έλεγχος για κινδύνους RAW. Όταν και τα δύο ορίσματα είναι διαθέσιμα σε κάποιον σταθμό κράτησης, τότε πραγματοποιείται η εκτέλεση της λειτουργίας. Οι εντολές μπορεί, στο στάδιο αυτό, να χρειαστούν πολλαπλούς κύκλους ρολογιού, ενώ οι εντολές φόρτωσης εξακολουθούν στο στάδιο αυτό να χρειάζονται δύο βήματα. Στο βήμα αυτό, οι εντολές αποθήκευσης χρειάζεται μόνο να έχουν στη διάθεσή τους τον καταχωρητή βάσης, καθώς η εκτέλεση των εντολών αποθήκευσης σε αυτό το χρονικό σημείο αφορά μονάχα τον υπολογισμό της ενεργού διεύθυνσης.

3. *Εγγραφή αποτελέσματος*: Όταν το αποτέλεσμα είναι διαθέσιμο, τότε πραγματοποιείται η εγγραφή του στον CDB (με την ετικέτα του ROB να έχει αποσταλεί κατά την έκδοση της εντολής) και από εκεί στον ROB, καθώς και σε όσους σταθμούς κράτησης αναμένουν το αποτέλεσμα αυτό. Ο σταθμός κράτησης στη συνέχεια χαρακτηρίζεται ως διαθέσιμος. Σε ό,τι αφορά τις εντολές αποθήκευσης απαιτούνται ειδικές ενέργειες. Αν η τιμή που πρόκειται να αποθηκευτεί είναι διαθέσιμη, τότε αυτή εγγράφεται στο πεδίο Τιμής της καταχώρισης του ROB που αφορά την εντολή αποθήκευσης. Σε περίπτωση που η τιμή που πρόκειται να αποθηκευτεί δεν είναι διαθέσιμη ακόμα, τότε πρέπει να παρακολουθείται ο CDB μέχρις ότου να εκπεμφθεί η τιμή αυτή, χρονικό σημείο κατά το οποίο ενημερώνεται το πεδίο Τιμής της καταχώρισης του ROB που αφορά την εντολή αποθήκευσης. Για λόγους απλότητας, θεωρούμε ότι αυτό πραγματοποιείται κατά τη διάρκεια του σταδίου Εγγραφής Αποτελέσματος της αποθήκευσης. Θα εξετάσουμε αργότερα τρόπους με τους οποίους μπορούμε να υπερκεράσουμε τον περιορισμό αυτό.

4. *Επικύρωση*: Πρόκειται για το τελικό στάδιο ολοκλήρωσης μιας εντολής, μετά το οποίο απομένει μόνο το αποτέλεσμά της (σε μερικούς επεξεργαστές αυτή η φάση επικύρωσης ονομάζεται «ολοκλήρωση» ή «αποφοίτηση»). Υπάρχουν τρεις διαφορετικές ακολουθίες ενεργειών στο στάδιο επικύρωσης ανάλογα με τον τύπο της εντολής που επικυρώνεται, ανάλογα δηλαδή με τον αν πρόκειται για διακλάδωση με εσφαλμένη πρόβλεψη, αποθήκευση ή κάποια άλλη εντολή (κανονική επικύρωση). Η περίπτωση της κανονικής επικύρωσης παρουσιάζεται όταν κάποια εντολή φτάνει στην κεφαλή του ROB και το αποτέλεσμά της βρίσκεται εντός του απομονωτή. Στο σημείο αυτό, ο επεξεργαστής ενημερώνει τον καταχωρητή χρησιμοποιώντας το σχετικό αποτέλεσμα και αφαιρεί την εντολή από τον ROB. Η επικύρωση των εντολών αποθήκευσης είναι παρόμοια με τη διαφορά ότι ενημερώνεται η μνήμη και όχι κάποιος καταχωρητής αποτελεσμάτων. Το ενδεχόμενο να καταφτάσει στην κεφαλή του ROB κάποια διακλάδωση με εσφαλμένη

πρόβλεψη συνεπάγεται ότι η εικασία ήταν λανθασμένη. Ο ROB εκκενώνεται και η εκτέλεση ξεκινά εκ νέου με την ορθή εντολή που ακολουθεί τη διακλάδωση. Σε περίπτωση που έχει πραγματοποιηθεί ορθή πρόβλεψη διακλάδωσης, τότε η διακλάδωση ολοκληρώνεται.

Μόλις επικυρωθεί η εντολή, χρησιμοποιείται εκ νέου η καταχώρισή της στον ROB και ενημερώνεται είτε ο καταχωρητής είτε η μνήμη προορισμού, κάτι που εξαλείφει την ανάγκη ύπαρξης της καταχώρισης στον ROB. Αν ο ROB είναι πλήρης, απλά διακόπτουμε την έκδοση εντολών, έως ότου να ελευθερωθεί κάποια θέση. Ας εξετάσουμε, τώρα, τον τρόπο με τον οποίο θα λειτουργούσε η προσέγγιση αυτή με το ίδιο παράδειγμα που χρησιμοποιήσαμε στον αλγόριθμο του Tomasulo.

Παράδειγμα: Ας υποθέσουμε ότι για τις λειτουργικές μονάδες κινητής υποδιαστολής ισχύουν οι ίδιοι λανθάνοντες χρόνοι με αυτούς των προηγούμενων παραδειγμάτων: η πρόσθεση έχει λανθάνοντα χρόνο της τάξης των 2 κύκλων ρολογιού, ο πολλαπλασιασμός της τάξης των 6 κύκλων ρολογιού και η διαίρεση της τάξης των 12 κύκλων ρολογιού. Χρησιμοποιώντας το παρακάτω τμήμα κώδικα, που είναι ίδιο με αυτό που χρησιμοποιήσαμε στο Σχήμα 2.11, θα παρουσιάσουμε την εικόνα των πινάκων κατάστασης, όταν η `MUL.D` είναι έτοιμη να εισέλθει στο στάδιο επικύρωσης.

```
L.D     F6,32(R2)
L.D     F2,44(R3)
MUL.D   F0,F2,F4
SUB.D   F8,F6,F2
DIV.D   F10,F0,F6
ADD.D   F6,F8,F2
```

Απάντηση: Το Σχήμα 2.15 παρουσιάζει το αποτέλεσμα των τριών πινάκων. Αξίζει να σημειωθεί ότι παρόλο που η εντολή `SUB.D` έχει ολοκληρώσει την εκτέλεσή της, δεν επικυρώνεται πριν την επικύρωση της `MUL.D`. Οι σταθμοί κράτησης και το πεδίο κατάστασης καταχωρητών περιέχουν τις ίδιες βασικές πληροφορίες που περιείχαν και στην περίπτωση του αλγορίθμου του Tomasulo (δείτε τη σελίδα 133 όπου περιλαμβάνεται περιγραφή των πεδίων αυτών). Οι διαφορές που υπάρχουν αφορούν το γεγονός ότι, τόσο στα πεδία Qj και Qk όσο και στα πεδία κατάστασης καταχωρητών, οι αριθμοί των σταθμών κράτησης αντικαθίστανται από τους αριθμούς των καταχωρίσεων του ROB, καθώς και το ότι έχουμε προσθέσει το πεδίο Dest στους σταθμούς κράτησης. Το πεδίο Dest προσδιορίζει την καταχώριση του ROB η οποία αποτελεί τον προορισμό του αποτελέσματος που παράγεται από αυτήν την καταχώριση του σταθμού κράτησης.

		Απομονωτής αναδιάταξης				
Καταχώριση	Απασχολημένος	Εντολή		Κατάσταση	Προορισμός	Τιμή
1	όχι	L.D	F6,32(R2)	Επικύρωση	F6	Mem[34 + Regs[R2]]
2	όχι	L.D	F2,44(R3)	Επικύρωση	F2	Mem[45 + Regs[R3]]
3	ναι	MUL.D	F0,F2,F4	Εγγραφή αποτ.	F0	#2 × Regs[F4]
4	ναι	SUB.D	F8,F2,F6	Εγγραφή αποτ.	F8	#2 – #1
5	ναι	DIV.D	F10,F0,F6	Εκτέλεση	F10	
6	ναι	ADD.D	F6,F8,F2	Εγγραφή αποτ.	F6	#4 + #2

		Σταθμοί κράτησης						
Ονομασία	Απασχολημένος	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	όχι							
Load2	όχι							
Add1	όχι							
Add2	όχι							
Add3	όχι							
Mult1	όχι	MUL.D	Mem[45 + Regs[R3]]	Regs[F4]			#3	
Mult2	ναι	DIV.D		Mem[34 + Regs[R2]]	#3		#5	

	Κατάσταση καταχωρητή FP									
Πεδίο	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	ναι	όχι	όχι	όχι	όχι	όχι	ναι	...	ναι	ναι

Σχήμα 2.15 **Τη χρονική στιγμή κατά την οποία η MUL.D είναι έτοιμη να επικυρωθεί, έχουν επικυρωθεί μόνο οι δύο εντολές L.D, παρόλο που αρκετές άλλες εντολές έχουν ολοκληρώσει την εκτέλεσή τους.** Η MUL.D βρίσκεται στην κεφαλή του ROB, ενώ οι δύο εντολές L.D βρίσκονται εκεί μόνο για τη διευκόλυνση της κατανόησης της κατάστασης από το χρήστη. Οι εντολές SUB.D και ADD.D δεν θα επικυρωθούν πριν την επικύρωση της εντολής MUL.D, παρόλο που τα αποτελέσματα των εντολών είναι διαθέσιμα και μπορούν να χρησιμοποιηθούν ως αφετηρία άλλων εντολών. Η DIV.D βρίσκεται σε εκτέλεση και δεν έχει ολοκληρωθεί απλά εξαιτίας του υψηλότερου λανθάνοντος χρόνου της σε σύγκριση με τη MUL.D. Η στήλη με την ονομασία «Τιμή» υποδηλώνει την τιμή η οποία διατηρείται. Η μορφή #X χρησιμοποιείται προκειμένου να γίνεται αναφορά στην τιμή πεδίου της καταχώρισης X του ROB. Οι απομονωτές αναδιάταξης 1 και 2 έχουν στην πραγματικότητα ολοκληρωθεί, ωστόσο παρουσιάζονται για ενημερωτικούς λόγους. Δεν παρουσιάζονται οι καταχωρίσεις της ουράς φόρτωσης - αποθήκευσης, ωστόσο οι καταχωρίσεις αυτές διατηρούν τη σειρά τους.

Το παραπάνω παράδειγμα απεικονίζει την πιο σημαντική διαφορά ανάμεσα στους επεξεργαστές εικασίας και τους επεξεργαστές δυναμικής χρονοδρομολόγησης. Ας συγκρίνουμε τα περιεχόμενα του Σχήματος 2.15 με αυτά του Σχήμα-

τος 2.11 στη σελίδα 137, όπου παρουσιάζεται η εκτέλεση της ίδιας ακολουθίας κώδικα σε επεξεργαστή που υλοποιεί τον αλγόριθμο του Tomasulo. Η βασική διαφορά έγκειται στο γεγονός ότι, στο παραπάνω παράδειγμα, δεν επιτρέπεται η ολοκλήρωση καμίας εντολής που έπεται της πρωτύτερης μη ολοκληρωμένης εντολής (της MUL.D στην παραπάνω περίπτωση). Αντιθέτως, στο Σχήμα 2.11 οι εντολές SUB.D και ADD.D έχουν, επίσης, ολοκληρωθεί.

Μία από τις επιπλοκές που προκύπτουν εξαιτίας της διαφοράς αυτής είναι το γεγονός ότι ο επεξεργαστής που διαθέτει τον ROB μπορεί να εκτελέσει δυναμικά τον κώδικα, διατηρώντας συγχρόνως ένα ακριβές μοντέλο διακοπών. Για παράδειγμα, σε περίπτωση που η εντολή MUL.D προκαλέσει διακοπή, μπορούμε απλά να περιμένουμε έως ότου να φτάσει στην κεφαλή του ROB και να εκτελέσουμε τη διακοπή, αφαιρώντας από τον ROB οποιεσδήποτε εκκρεμείς εντολές. Επειδή η επικύρωση των εντολών πραγματοποιείται εντός σειράς, αυτό έχει ως αποτέλεσμα την ύπαρξη ακριβών εξαιρέσεων.

Αντιθέτως, στο παράδειγμα στο οποίο χρησιμοποιείται ο αλγόριθμος του Tomasulo, τόσο η εντολή SUB.D όσο και η ADD.D μπορούν να ολοκληρωθούν πριν να προκύψει η εξαίρεση της MUL.D. Φυσικό επακόλουθο είναι το γεγονός ότι είναι εφικτή η αντικατάσταση (overwrite) των καταχωρητών F6 και F8 (προορισμοί των εντολών SUB.D και ADD.D) και, επομένως, η διακοπή είναι ανακριβής.

Μερικοί χρήστες και αρχιτέκτονες έχουν αποφασίσει ότι οι ανακριβείς εξαιρέσεις κινητής υποδιαστολής δεν είναι αποδεκτές σε επεξεργαστές υψηλής απόδοσης, καθώς το πρόγραμμα κατά πάσα πιθανότητα θα τερματιστεί (δείτε το Παράρτημα Z για περισσότερες λεπτομέρειες γύρω από το ζήτημα αυτό). Άλλοι τύποι εξαιρέσεων, όπως τα σφάλματα σελίδας, είναι πολύ πιο δύσκολο να αντιμετωπιστούν στην περίπτωση που είναι ανακριβείς, καθώς το πρόγραμμα πρέπει, χωρίς να γίνεται αντιληπτό, από το χρήστη να συνεχίζει την εκτέλεση μετά την αντιμετώπιση τέτοιων εξαιρέσεων.

Όπως θα δούμε στο παρακάτω παράδειγμα, η χρήση του ROB σε συνδυασμό με την επικύρωση των εντολών εντός σειράς παρέχει, πέρα από την υποστήριξη της υποθετικής εκτέλεσης, ακριβείς εξαιρέσεις.

Παράδειγμα: Ας εξετάσουμε το παράδειγμα κώδικα που χρησιμοποιήθηκε νωρίτερα για τον αλγόριθμο του Tomasulo, του οποίου η εκτέλεση παρουσιάστηκε στο Σχήμα 2.13.

```
Loop:   L.D      F0,0(R1)
        MUL.D    F4,F0,F2
        S.D      F4,0(R1)
        DADDIU   R1,R1,-8
        BNE      R1,R2,Loop ; διακλάδωση αν R1≠R2
```

Ας υποθέσουμε ότι όλες οι εντολές του βρόχου έχουν εκδοθεί δύο φορές. Ας υποθέσουμε, επίσης, ότι οι εντολές L.D και MUL.D της πρώτης επανάληψης έχουν

		Απομονωτής αναδιάταξης			
Καταχώριση	**Απασχολημένος**	**Εντολή**	**Κατάσταση**	**Προορισμός**	**Τιμή**
1	όχι	L.D F0,0(R1)	Επικύρωση	F0	Mem[0 + Regs[R1]]
2	όχι	MUL.D F4,F0,F2	Επικύρωση	F4	#1 × Regs[F2]
3	ναι	S.D F4,0(R1)	Εγγραφή αποτελ.	0 + Regs[R1]	#2
4	ναι	DADDIU R1,R1,#-8	Εγγραφή αποτελ.	R1	Regs[R1] – 8
5	ναι	BNE R1,R2,Loop	Εγγραφή αποτελ.		
6	ναι	L.D F0,0(R1)	Εγγραφή αποτελ.	F0	Mem[#4]
7	ναι	MUL.D F4,F0,F2	Εγγραφή αποτελ.	F4	#6 × Regs[F2]
8	ναι	S.D F4,0(R1)	Εγγραφή αποτελ.	0 + #4	#7
9	ναι	DADDIU R1,R1,#-8	Εγγραφή αποτελ.	R1	#4 – 8
10	ναι	BNE R1,R2,Loop	Εγγραφή αποτελ.		

	Κατάσταση καταχωρητή FP								
Πεδίο	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Reorder #	6				7				
Busy	ναι	όχι	όχι	όχι	ναι	όχι	όχι	...	όχι

D **Σχήμα 2.16 Μόνο οι εντολές L.D και MUL.D έχουν επικυρωθεί, παρόλο που όλες οι υπόλοιπες έχουν ολοκληρώσει την εκτέλεσή τους.** Συνεπώς, κανένας σταθμός κράτησης δεν είναι απασχολημένος και κανένας από αυτούς δεν παρουσιάζεται. Οι επόμενες εντολές θα επικυρωθούν το ταχύτερο δυνατόν. Οι πρώτοι δύο απομονωτές αναδιάταξης είναι κενοί, ωστόσο συμπεριλαμβάνονται στο σχήμα για να υπάρχει πιο ολοκληρωμένη εικόνα.

επικυρωθεί, καθώς και ότι όλες οι υπόλοιπες εντολές έχουν ολοκληρώσει την εκτέλεσή τους. Κατά κανόνα, η εντολή αποθήκευσης βρίσκεται εντός του ROB σε αναμονή τόσο του ορίσματος της ενεργού διεύθυνσης (του R1 στο παράδειγμα αυτό) όσο και της τιμής (του F4 στο παράδειγμα αυτό). Καθώς εξετάζουμε μόνο τη διασωλήνωση κινητής υποδιαστολής, ας υποθέσουμε ότι η ενεργός διεύθυνση που χρησιμοποιείται από την εντολή αποθήκευσης υπολογίζεται κατά το χρόνο έκδοσης της εντολής.

Απάντηση: Το Σχήμα 2.16 παρουσιάζει τα αποτελέσματα σε δύο πίνακες.

Επειδή πριν την επικύρωση της εντολής ούτε οι τιμές των καταχωρητών ούτε οι τιμές της μνήμης έχουν στην πραγματικότητα εγγραφεί, ο επεξεργαστής μπορεί με ευκολία να αναιρέσει τις υποθετικές ενέργειές του, στην περίπτωση που διαπιστώνεται ότι κάποια διακλάδωση έχει προβλεφθεί εσφαλμένα. Ας υποθέσουμε ότι, στο Σχήμα 2.16, την πρώτη φορά η διακλάδωση BNE δεν ακο-

λουθείται. Οι εντολές που προηγούνται της διακλάδωσης απλά θα επικυρωθούν όταν καθεμία από αυτές θα φτάσει στην κεφαλή του ROB. Όταν η διακλάδωση φτάσει στην κεφαλή του απομονωτή αυτού, ο απομονωτής απλά εκκαθαρίζεται και ο επεξεργαστής ξεκινά την ανάκτηση των εντολών από την άλλη διαδρομή.

Στην πράξη, οι επεξεργαστές που υποστηρίζουν την εικασία επιχειρούν να ανακάμψουν (recover) όσο το δυνατόν συντομότερα μετά από μια εσφαλμένη πρόβλεψη διακλάδωσης. Η ανάκαμψη αυτή μπορεί να πραγματοποιηθεί αφενός μέσω της εκκαθάρισης του ROB από όλες τις καταχωρίσεις οι οποίες ακολουθούν τη διακλάδωση με την εσφαλμένη πρόβλεψη, κάτι που επιτρέπει στις καταχωρίσεις του ROB που προηγούνται της διακλάδωσης να συνεχίσουν και αφετέρου διαμέσου της εκ νέου έναρξης της ανάκτησης χρησιμοποιώντας την ορθή εντολή που ακολουθεί τη διακλάδωση. Στους υποθετικούς επεξεργαστές, η απόδοση εξαρτάται σε μεγαλύτερο βαθμό από την πρόβλεψη της διακλάδωσης, καθώς οι επιπτώσεις των εσφαλμένων προβλέψεων είναι μεγαλύτερες. Κατά συνέπεια, όλα τα ζητήματα που σχετίζονται με το χειρισμό των διακλαδώσεων - ακρίβεια πρόβλεψης, λανθάνων χρόνος της ανίχνευσης εσφαλμένων προβλέψεων και χρόνος ανάκαμψης από εσφαλμένες προβλέψεις - είναι ακόμα πιο σημαντικά.

Ο χειρισμός των εξαιρέσεων πραγματοποιείται μέσω της μη αναγνώρισής τους μέχρις ότου να είναι σε θέση να επικυρωθούν. Αν κάποια υποθετική εντολή προκαλέσει εξαίρεση, τότε η εξαίρεση καταγράφεται στον ROB. Αν προκύψει κάποια εσφαλμένη πρόβλεψη, κάτι που συνεπάγεται ότι η εντολή δεν θα έπρεπε να εκτελεστεί, τότε η εξαίρεση μαζί με την εντολή αφαιρούνται κατά την εκκαθάριση του ROB. Αν η εντολή φτάσει στην κεφαλή του ROB, τότε γνωρίζουμε ότι έχει πάψει πλέον να είναι υποθετική και ως εκ τούτου η εξαίρεση πρέπει πραγματικά να εκτελεστεί. Επίσης, ο χειρισμός των εξαιρέσεων μπορεί να πραγματοποιηθεί κατά τη χρονική στιγμή που προκύπτουν, εφόσον έχουν επιλυθεί όλες οι προηγούμενες διακλαδώσεις, ωστόσο η προσέγγιση αυτή παρουσιάζει περισσότερες προκλήσεις στην περίπτωση των εξαιρέσεων απ' ό,τι στην περίπτωση της εσφαλμένης πρόβλεψης των διακλαδώσεων, ενώ είναι λιγότερο σημαντική, καθώς προκύπτει πιο σπάνια.

Το Σχήμα 2.17 παρουσιάζει τα βήματα της εκτέλεσης κάποιας εντολής, καθώς και τις συνθήκες που πρέπει να ικανοποιούνται για τη μετάβαση από το ένα βήμα στο επόμενο και την εκτέλεση των ενεργειών. Παρουσιάζεται η περίπτωση κατά την οποία οι διακλαδώσεις με εσφαλμένη πρόβλεψη δεν έχουν επιλυθεί μέχρι την επικύρωση. Παρόλο που η εικασία φαίνεται να αποτελεί μια απλή προσθήκη στη δυναμική χρονοδρομολόγηση, η σύγκριση του Σχήματος 2.17 με το Σχήμα 2.12 στο οποίο παρουσιάζεται ο αλγόριθμος του Tomasulo, καταδεικνύει ότι η εικασία περιπλέκει σε σημαντικό βαθμό τον έλεγχο. Επιπλέον, αξίζει να θυμηθούμε ότι οι εσφαλμένες προβλέψεις των διακλαδώσεων είναι, επίσης, σε κάποιο βαθμό πιο πολύπλοκες.

Υπάρχει μια σημαντική διαφορά όσον αφορά τον τρόπο με τον οποίο οι υποθετικοί επεξεργαστές χειρίζονται τις εντολές αποθήκευσης σε σύγκριση με αυτούς

Κατάσταση	Αναμονή έως ότου	Ενέργεια ή καταγραφή

Έκδοση όλων των εντολών	Διαθεσιμότητα τόσο του σταθμού κράτησης (r) όσο και του ROB (b)	if (RegisterStat[rs].Busy)/*in-flight instr. writes rs*/
		{h ← RegisterStat[rs].Reorder;
		if (ROB[h].Ready)/* Instr completed already */
		{RS[r].Vj ← ROB[h].Value; RS[r].Qj ← 0;}
		else {RS[r].Qj ← h;} /* wait for instruction */
		} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0;};
		RS[r].Busy ← yes; RS[r].Dest ← b;
		ROB[b].Instruction ← opcode; ROB[b].Dest ← rd;ROB[b].Ready ← no;

Λειτουργίες FP και αποθήκευσης		if (RegisterStat[rt].Busy) /*in-flight instr writes rt*/
		{h ← RegisterStat[rt].Reorder;
		if (ROB[h].Ready)/* Instr completed already */
		{RS[r].Vk ← ROB[h].Value; RS[r].Qk ← 0;}
		else {RS[r].Qk ← h;} /* wait for instruction */
		} else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0;};

Λειτουργίες FP		RegisterStat[rd].Reorder ← b; RegisterStat[rd].Busy ← yes; ROB[b].Dest ← rd;
Λειτουργίες φόρτωσης		RS[r].A ← imm; RegisterStat[rt].Reorder ← b; RegisterStat[rt].Busy ← yes; ROB[b].Dest ← rt;
Λειτουργίες αποθήκευσης		RS[r].A ← imm;
Εκτέλεση FP op	(RS[r].Qj == 0) και (RS[r].Qk == 0)	Υπολογισμός αποτελεσμάτων – τα ορίσματα βρίσκονται στους Vj και Vk
Πρώτο βήμα φόρτωσης	(RS[r].Qj == 0) and και δεν υπάρχουν λειτουργίες αποθήκευσης νωρίτερα στην ουρά	RS[r].A ← RS[r].Vj + RS[r].A;
Δεύτερο βήμα φόρτωσης	Έχει ολοκληρωθεί το πρώτο βήμα φόρτωσης και όλες οι λειτουργίες φόρτωσης που βρίσκονταν στον ROB διαθέτουν διαφορετικές διευθύνσεις	Ανάγνωση από τη Mem[RS[r].A]
Αποθήκευση	(RS[r].Qj == 0) και αποθήκευση στην κεφαλή της ουράς	ROB[h].Address ← RS[r].Vj + RS[r].A;
Εγγραφή όλων των αποτελεσμάτων εκτός της αποθήκευσης	Η εκτέλεση έχει ολοκληρωθεί στον r & ο CDB είναι διαθέσιμος	b ← RS[r].Dest; RS[r].Busy ← no; ∀x(if (RS[x].Qj==b) {RS[x].Vj ← result; RS[x].Qj ← 0}); ∀x(if (RS[x].Qk==b) {RS[x].Vk ← result; RS[x].Qk ← 0}); ROB[b].Value ← result; ROB[b].Ready ← yes;
Αποθήκευση	Η εκτέλεση έχει ολοκληρωθεί στον r και (RS[r].Qk == 0)	ROB[h].Value ← RS[r].Vk;

Σχήμα 2.17 **Βήματα αλγορίθμου και απαιτήσεις κάθε βήματος.** Σε ό,τι αφορά την εντολή που εκδίδεται, το `rd` αποτελεί τον προορισμό, τα `rs` και `rt` αποτελούν τους δεσμευμένους σταθμούς κράτησης, το `b` συμβολίζει τη δεσμευμένη καταχώριση του ROB και το `h` συμβολίζει την καταχώριση της κεφαλής του ROB. Η `RS` συμβολίζει τη δομή δεδομένων των σταθμών κράτησης. Η τιμή που επιστρέφει κάθε σταθμός κράτησης ονομάζεται `result`. Η `RegisterStat` συμβολίζει τη δομή δεδομένων των καταχωρητών, το `Regs` αναπαριστά τους πραγματικούς καταχωρητές, ενώ η `ROB` συμβολίζει τη δομή δεδομένων του απομονωτή αναδιάταξης.

Επικύρωση	Η εντολή βρίσκεται στην κεφαλή του ROB (καταχώριση h) και ROB[h].ready == ναι	d ← ROB[h].Dest; /* register dest, if exists */
		if (ROB[h].Instruction==Branch)
		{if (branch is mispredicted)
		{clear ROB[h], RegisterStat; fetch branch dest;};}
		else if (ROB[h].Instruction==Store)
		{Mem[ROB[h].Destination] ← ROB[h].Value;}
		else /* put the result in the register destination */
		{Regs[d] ← ROB[h].Value;};
		ROB[h].Busy ← no; /* free up ROB entry */
		/* free up dest register if no one else writing it */
		if (RegisterStat[d].Reorder==h) {RegisterStat[d].Busy ← no;};

Σχήμα 2.17 **Βήματα αλγορίθμου και απαιτήσεις κάθε βήματος (Συνέχεια)**

που εφαρμόζουν τον αλγόριθμο του Tomasulo. Στον αλγόριθμο του Tomasulo, η εντολή αποθήκευσης μπορεί να ενημερώσει τη μνήμη, όταν φτάσει στο στάδιο Εγγραφής Αποτελέσματος (που διασφαλίζει ότι έχει υπολογιστεί η ενεργός διεύθυνση) και η τιμή των δεδομένων είναι διαθέσιμη. Στους υποθετικούς επεξεργαστές, η εντολή αποθήκευσης μπορεί να ενημερώσει την μνήμη μόνο όταν φτάσει στην κεφαλή του ROB. Η διαφορά αυτή διασφαλίζει ότι η μνήμη δεν ενημερώνεται, πριν η εντολή πάψει να είναι υποθετική.

Το Σχήμα 2.17 διαθέτει μια σημαντική απλοποίηση που αφορά τις εντολές αποθήκευσης, η οποία στην πράξη είναι περιττή. Στο Σχήμα 2.17, οι εντολές αποθήκευσης πρέπει κατά το στάδιο Εγγραφής Αποτελέσματος να αναμένουν το όρισμα του καταχωρητή αφετηρίας, του οποίου η τιμή πρόκειται να αποθηκευτεί. Στη συνέχεια, η τιμή αυτή μεταφέρεται από το πεδίο Vk του σταθμού κράτησης της εντολής αποθήκευσης στο πεδίο Τιμής της καταχώρισης της αποθήκευσης στον ROB. Παρ' όλα αυτά, στην πραγματικότητα, η τιμή που πρόκειται να αποθηκευτεί δεν είναι απαραίτητη παρά μόνο *ακριβώς πριν* την επικύρωση της εντολής αποθήκευσης και, επομένως, μπορεί να τοποθετηθεί απευθείας στην καταχώριση του ROB από την εντολή αφετηρίας. Αυτό επιτυγχάνεται ρυθμίζοντας το υλικό με τέτοιο τρόπο, ώστε να παρατηρεί πότε η τιμή αφετηρίας που πρόκειται να αποθηκευτεί είναι διαθέσιμη στην καταχώριση της εντολής αποθήκευσης στον ROB και διενεργώντας, σε κάθε ολοκλήρωση εντολής, αναζήτηση στον ROB για την εύρεση εξαρτώμενων εντολών αποθήκευσης.

Η παραπάνω προσθήκη δεν είναι πολύπλοκη, ωστόσο επιφέρει δύο αποτελέσματα: είναι απαραίτητη η προσθήκη ενός πεδίου στον ROB και επομένως το Σχήμα 2.17, το οποίο ήδη παρατίθεται χρησιμοποιώντας μικρή γραμματοσειρά, θα ήταν ακόμα μεγαλύτερο! Παρόλο που το Σχήμα 2.17 προβαίνει σε αυτήν την απλοποίηση, στα παραδείγματά μας θα επιτρέπουμε στην εντολή αποθήκευσης να διέρχεται από το στάδιο Εγγραφής Αποτελέσματος και απλά να αναμένει την απαραίτητη τιμή κατά την επικύρωσή της.

Όπως στην περίπτωση του αλγορίθμου του Tomasulo, έτσι και εδώ πρέπει να αποφεύγουμε τους κινδύνους που προκύπτουν μέσω της μνήμης. Οι κίνδυνοι

WAW και WAR που προκύπτουν μέσω της μνήμης εξαλείφονται με τη χρήση της εικασίας, διότι, όταν μια εντολή αποθήκευσης βρίσκεται στην κεφαλή του ROB, η πραγματική ενημέρωση της μνήμης πραγματοποιείται εντός σειράς και, ως εκ τούτου, δεν μπορεί να βρίσκεται σε εκκρεμότητα καμία προηγούμενη εντολή φόρτωσης ή αποθήκευσης. Οι κίνδυνοι RAW που προκύπτουν μέσω της μνήμης μπορούν να αποφευχθούν τηρώντας δύο περιορισμούς:

1. απαγορεύοντας στις εντολές φόρτωσης να ξεκινούν το δεύτερο βήμα της εκτέλεσής τους, αν οποιαδήποτε ενεργή καταχώριση του ROB η οποία αντιστοιχεί σε κάποια εντολή αποθήκευσης διαθέτει πεδίο Προορισμού το οποίο είναι ίσο με την τιμή του πεδίου A της εντολής φόρτωσης και

2. διατηρώντας τη σειρά προγράμματος για τον υπολογισμό της ενεργού διεύθυνσης κάποιας εντολής φόρτωσης σε σχέση με όλες τις όλες τις προηγούμενες εντολές αποθήκευσης.

Ο συνδυασμός των δύο αυτών περιορισμών διασφαλίζει ότι οποιαδήποτε εντολή φόρτωσης προσπελάζει μια θέση μνήμης η οποία έχει εγγραφεί από κάποια προγενέστερη εντολή αποθήκευσης δεν μπορεί έχει πρόσβαση στη μνήμη πριν η εντολή αποθήκευσης να εγγράψει τα δεδομένα. Μερικοί υποθετικοί επεξεργαστές στην πραγματικότητα, όταν προκύπτει αυτού του είδους ο κίνδυνος RAW, προωθούν απευθείας την τιμή από την εντολή αποθήκευσης στην εντολή φόρτωσης. Εναλλακτική προσέγγιση είναι η πρόβλεψη των ενδεχόμενων συγκρούσεων χρησιμοποιώντας κάποια μορφή προσέγγισης τιμών. Αυτό θα το εξετάσουμε στην Ενότητα 2.9.

Παρόλο που αυτή η επεξήγηση της υποθετικής εκτέλεσης εστιάζει στις λειτουργίες κινητής υποδιαστολής, οι τεχνικές μπορούν εύκολα να επεκταθούν ώστε να υποστηρίζουν καταχωρητές και λειτουργικές μονάδες ακεραίων, όπως θα διαπιστώσουμε στην ενότητα με την ονομασία «Συνδυασμός Όλων των Δεδομένων». Πράγματι, η εικασία μπορεί να είναι πιο χρήσιμη σε προγράμματα ακεραίων, καθώς αυτού του είδους τα προγράμματα έχουν την τάση να διαθέτουν κώδικα όπου η συμπεριφορά των διακλαδώσεων είναι λιγότερο προβλέψιμη. Επιπλέον, οι τεχνικές αυτές μπορούν να επεκταθούν ώστε να υποστηρίζονται από επεξεργαστές πολλαπλής έκδοσης, παρέχοντας τη δυνατότητα σε πολλαπλές εντολές να εκδίδονται και να επικυρώνονται σε κάθε κύκλο ρολογιού. Στην πραγματικότητα, η εικασία αποτελεί πιθανότατα το πιο ενδιαφέρον χαρακτηριστικό αυτού του είδους των επεξεργαστών, καθώς οι λιγότερο φιλόδοξες τεχνικές μπορούν, με τη βοήθεια του μεταγλωττιστή, να αξιοποιήσουν σε ικανοποιητικό βαθμό τον ILP εντός των βασικών blocks.

2.7 Αξιοποίηση του ILP Χρησιμοποιώντας Πολλαπλή Έκδοση και Στατική Χρονοδρομολόγηση

Οι τεχνικές των προηγούμενων ενοτήτων μπορούν να χρησιμοποιηθούν τόσο για την εξάλειψη των διαστημάτων αδράνειας δεδομένων και ελέγχου όσο και για την επίτευξη του ιδανικού CPI, που ισούται με τη μονάδα. Προκειμένου να βελτιώσουμε περαιτέρω την απόδοση θα θέλαμε να μειώσουμε το CPI σε τιμές κάτω από τη μονάδα. Ωστόσο, το CPI δεν μπορεί να μειωθεί κάτω από τη μονάδα, αν εκδίδουμε μόνο μία εντολή σε κάθε κύκλο ρολογιού.

Ο στόχος των *επεξεργαστών πολλαπλής έκδοσης* (multiple-issue processors), ζήτημα που εξετάζεται στις επόμενες ενότητες, είναι η παροχή της δυνατότητας έκδοσης πολλαπλών εντολών σε έναν κύκλο ρολογιού. Υπάρχουν τρεις κατηγορίες επεξεργαστών πολλαπλής έκδοσης:

1. οι στατικά χρονοδρομολογημένοι υπερβαθμωτοί επεξεργαστές,
2. οι επεξεργαστές VLIW (very long instruction word, πολύ μεγάλης λέξης εντολής) και
3. οι δυναμικά χρονοδρομολογημένοι υπερβαθμωτοί επεξεργαστές.

Οι δύο τύποι των υπερβαθμωτών επεξεργαστών (superscalar processors) εκδίδουν μεταβαλλόμενο αριθμό εντολών ανά κύκλο ρολογιού και εφαρμόζουν είτε εκτέλεση εντός σειράς σε περίπτωση που χρονοδρομολογούνται στατικά είτε εκτέλεση εκτός σειράς σε περίπτωση που χρονοδρομολογούνται δυναμικά.

Αντιθέτως, οι επεξεργαστές VLIW εκδίδουν σταθερό αριθμό εντολών, οι οποίες είναι διαμορφωμένες είτε ως μία μεγάλη εντολή είτε ως ένα σταθερό πακέτο εντολών, όπου ο παραλληλισμός μεταξύ των εντολών υποδηλώνεται ρητά από την ίδια την εντολή. Οι επεξεργαστές VLIW από τη φύση τους χρονοδρομολογούνται στατικά από το μεταγλωττιστή. Όταν η Intel και η HP δημιούργησαν την αρχιτεκτονική IA-64, η οποία περιγράφεται στο Παράρτημα Z, χρησιμοποίησαν, επίσης, την ονομασία EPIC[10] - που σημαίνει υπολογιστής ρητά παράλληλων εντολών - για αυτήν την τεχνοτροπία αρχιτεκτονικής.

Παρόλο που οι στατικά χρονοδρομολογημένοι υπερβαθμωτοί επεξεργαστές εκδίδουν μεταβαλλόμενο και όχι σταθερό αριθμό εντολών ανά κύκλο ρολογιού, στην πραγματικότητα έχουν περισσότερες ομοιότητες με τους VLIWs, καθώς και οι δύο προσεγγίσεις βασίζονται στο μεταγλωττιστή για τη χρονοδρομολόγηση του κώδικα του επεξεργαστή. Οι στατικά χρονοδρομολογημένοι υπερβαθμωτοί, εξαιτίας του γεγονότος ότι έχουν πλεονεκτήματα που φθίνουν όσο το εύρος της έκδοσης αυξάνεται, χρησιμοποιούνται πρωτίστως για περιορισμένα εύρη έκδοσης, που συνήθως περιλαμβάνουν δύο μόνο εντολές. Όταν το εύρος πρέπει να

[10] Σ.τ.Μ. Ο αντίστοιχος αγγλικός όρος είναι "explicitly parallel instruction computer".

Σύνηθες όνομα	Δομή έκδοσης	Ανίχνευση κινδύνων	Χρονοδρομολόγηση	Διακριτά χαρακτηριστικά	Παραδείγματα
Υπερβαθμωτός (στατικός)	δυναμική	από το υλικό	στατική	εκτέλεση εντός σειράς	κυρίως σε ενσωματωμένα περιβάλλοντα: MIPS και ARM
Υπερβαθμωτός (δυναμικός)	δυναμική	από το υλικό	δυναμική	εκτέλεση εκτός σειράς, χωρίς εικασία	κανένα έως σήμερα
Υπερβαθμωτός (υποθετικός)	δυναμική	από το υλικό	δυναμική εικασία	εκτέλεση εκτός σειράς με εικασία	Pentium 4, MIPS R12K, IBM Power5
VLIW/LIW	στατική	κυρίως από λογισμικό	στατική	όλοι οι κίνδυνοι προσδιορίζονται και επισημαίνονται από το μεταγλωττιστή (συνήθως με έμμεσο τρόπο)	τα περισσότερα παραδείγματα ανήκουν στο χώρο των ενσωματωμένων συστημάτων, όπως το T1 C6x
EPIC	πρωτίστως στατική	κυρίως από λογισμικό	κυρίως στατική	όλοι οι κίνδυνοι προσδιορίζονται από το μεταγλωττιστή	Itanium

Σχήμα 2.18 Οι πέντε κύριες προσεγγίσεις που χρησιμοποιούνται για τους επεξεργαστές πολλαπλής έκδοσης και τα βασικά διακριτά χαρακτηριστικά τους. Το κεφάλαιο αυτό έχει εστιάσει σε τεχνικές εντατικής χρήσης του υλικού, όλες εκ των οποίων αποτελούν κάποια μορφή υπερβαθμωτής προσέγγισης. Το Παράρτημα Ζ εστιάζει σε προσεγγίσεις που βασίζονται σε μεταγλωττιστή. Η προσέγγιση EPIC, όπως έχει ενσωματωθεί στην αρχιτεκτονική IA-64, προεκτείνει πολλές από τις έννοιες των πρώτων προσεγγίσεων VLIW, παρέχοντας με τον τρόπο αυτό ένα μίγμα στατικών και δυναμικών προσεγγίσεων.

είναι υψηλότερο, οι περισσότεροι σχεδιαστές προτιμούν την υλοποίηση είτε των VLIWs είτε των δυναμικά χρονοδρομολογημένων υπερβαθμωτών. Στην ενότητα αυτή εστιάζουμε στους VLIWs, εξαιτίας των ομοιοτήτων που υφίστανται όσον αφορά το υλικό και την απαιτούμενη τεχνολογία του μεταγλωττιστή. Οι βασικές έννοιες του κεφαλαίου αυτού μπορούν εύκολα να μεταφερθούν στους στατικά χρονοδρομολογημένους υπερβαθμωτούς.

Το Σχήμα 2.18 συνοψίζει τις βασικές προσεγγίσεις που σχετίζονται με την πολλαπλή έκδοση σε συνδυασμό με τα ιδιαίτερα χαρακτηριστικά τους, ενώ συγχρόνως παρουσιάζει τους επεξεργαστές οι οποίοι εφαρμόζουν κάθε προσέγγιση.

Η Βασική Προσέγγιση VLIW

Οι VLIWs χρησιμοποιούν πολλαπλές ανεξάρτητες λειτουργικές μονάδες. Αντί να προβαίνουν στην έκδοση πολλαπλών ανεξάρτητων εντολών στις μονάδες, είτε ενοποιούν τις πολλαπλές εντολές σε μία πολύ μεγάλη εντολή είτε έχουν την απαίτηση οι εντολές του πακέτου έκδοσης να ικανοποιούν τους ίδιους περιορισμούς. Αφού δεν υπάρχει καμία θεμελιώδης διαφορά ανάμεσα στις δύο προσεγγίσεις, υποθέτουμε απλά ότι ισχύει η περίπτωση κατά την οποία πολλαπλές λειτουργίες τοποθετούνται εντός μίας εντολής, όπως στην αρχική προσέγγιση VLIW.

Καθώς αυτό το πλεονέκτημα των VLIWs αυξάνεται παράλληλα με την αύξηση του μέγιστου ρυθμού έκδοσης, εστιάζουμε σε επεξεργαστές που διαθέτουν μεγαλύτερο εύρος έκδοσης. Πράγματι, στους απλούς επεξεργαστές διπλής έκδοσης, η επιβάρυνση του υπερβαθμωτού είναι κατά πάσα πιθανότητα αμελητέα. Πολλοί σχεδιαστές πιθανότατα θα υποστήριζαν ότι η διαχείριση των επεξεργαστών τετραπλής έκδοσης είναι εφικτή, ωστόσο, όπως θα διαπιστώσουμε στο επόμενο κεφάλαιο, η αύξηση της επιβάρυνσης αποτελεί σημαντικό παράγοντα περιορισμού των επεξεργαστών μεγαλύτερου εύρους έκδοσης.

Ας εξετάσουμε έναν επεξεργαστή VLIW που διαθέτει εντολές οι οποίες περιέχουν πέντε λειτουργίες, μεταξύ των οποίων περιλαμβάνονται μία λειτουργία ακεραίων (η οποία θα μπορούσε, επίσης, να είναι διακλάδωση), δύο λειτουργίες κινητής υποδιαστολής και δύο αναφορές μνήμης. Η εντολή διαθέτει σύνολο πεδίων για κάθε λειτουργική μονάδα - ίσως μεταξύ 16-24 bits ανά μονάδα, κάτι που έχει ως αποτέλεσμα τη δημιουργία εντολών με μήκος μεταξύ των 80 και των 120 bits. Αντιθέτως, οι Intel Itanium I και II περιλαμβάνουν 6 λειτουργίες ανά πακέτο εντολών.

Προκειμένουν να διατηρούνται απασχολημένες οι λειτουργικές μονάδες, οι ακολουθίες κώδικα πρέπει να διαθέτουν επαρκή παραλληλισμό για τη συμπλήρωση των διαθέσιμων θυρίδων λειτουργίας. Ο παραλληλισμός αυτός επιτυγχάνεται με το ξεδίπλωμα των βρόχων και τη χρονοδρομολόγηση του κώδικα εντός ενός μεγαλύτερου σώματος βρόχου. Σε περίπτωση που το ξεδίπλωμα παραγάγει ευθύγραμμο κώδικα, τότε μπορούν να χρησιμοποιηθούν τεχνικές *τοπικής χρονοδρομολόγησης* (local scheduling), οι οποίες εφαρμόζονται σε ένα μόνο βασικό block. Αν η εύρεση και η αξιοποίηση του παραλληλισμού απαιτεί τη χρονοδρομολόγηση κώδικα μεταξύ διακλαδώσεων, τότε πρέπει να χρησιμοποιηθεί κάποιος αλγόριθμος *καθολικής χρονοδρομολόγησης* (global scheduling). Οι αλγόριθμοι καθολικής χρονοδρομολόγησης δεν είναι μόνο περισσότερο πολύπλοκοι στη δομή τους, αλλά συγχρόνως πρέπει να αντιμετωπίζουν σημαντικά πιο περίπλοκες καταστάσεις βελτιστοποίησης, καθώς η μεταφορά κώδικα μεταξύ διακλαδώσεων είναι δαπανηρή διαδικασία.

Στο Παράρτημα Ζ, θα εξετάσουμε τη *χρονοδρομολόγηση ίχνους* (trace scheduling), που αποτελεί μία από τις τεχνικές καθολικής χρονοδρομολόγησης που αναπτύχθηκαν ειδικά για τους VLIWs. Θα διερευνήσουμε, επίσης, την υποστήριξη που παρέχεται από ειδικό υλικό το οποίο επιτρέπει την απαλοιφή κάποιων διακλαδώσεων υπό συνθήκη, γεγονός που οδηγεί στην επέκταση της χρησιμότητας της τοπικής χρονοδρομολόγησης και στη βελτίωση της απόδοσης της καθολικής χρονοδρομολόγησης.

Προς το παρόν, θα βασιστούμε στο ξεδίπλωμα βρόχου για την παραγωγή μακροσκελών ευθύγραμμων ακολουθιών κώδικα, έτσι ώστε να μπορούμε να χρησιμοποιούμε την τοπική χρονοδρομολόγηση για τη δημιουργία εντολών VLIW και να εστιάζουμε στο βαθμό λειτουργίας των επεξεργαστών αυτών.

Παράδειγμα: Ας υποθέσουμε ότι έχουμε έναν VLIW ο οποίος σε κάθε κύκλο ρολογιού είναι σε θέση να εκδίδει δύο αναφορές μνήμης, δύο λειτουργίες FP και μία λειτουργία ακεραίων ή διακλάδωσης. Θα παρουσιάσουμε την ξεδιπλωμένη έκδοση του βρόχου $x[i] = x[i] + s$ (ανατρέξτε στη σελίδα 104 για να δείτε τον κώδικα της MIPS) για έναν τέτοιου είδους επεξεργαστή. Το ξεδίπλωμα πραγματοποιείται όσες φορές είναι απαραίτητο για την εξάλειψη των διαστημάτων αδράνειας. Οι καθυστερημένες διακλαδώσεις δεν λαμβάνονται υπόψη.

Απάντηση: Το Σχήμα 2.19 παρουσιάζει τον κώδικα. Ο βρόχος έχει ξεδιπλωθεί, έτσι ώστε να δημιουργηθούν επτά αντίγραφα του σώματος του κώδικα, κάτι το οποίο οδηγεί στην εξάλειψη όλων των διαστημάτων αδράνειας (των εντελώς, δηλαδή, κενών κύκλων έκδοσης) και εκτελείται εντός 9 κύκλων. Ο κώδικας αυτός έχει ως αποτέλεσμα ρυθμό εκτέλεσης της τάξης των επτά αποτελεσμάτων εντός 9 κύκλων ή των 1.29 κύκλων ανά αποτέλεσμα, ο οποίος είναι σχεδόν διπλάσιος από αυτόν του υπερβαθμωτού επεξεργαστή διπλής έκδοσης της Ενότητας 2.2, ο οποίος χρησιμοποίησε ξεδιπλωμένο και χρονοδρομολογημένο κώδικα.

Στο αρχικό μοντέλο του VLIW υπήρχαν τόσο τεχνικά όσο και λογικά προβλήματα τα οποία καθιστούσαν την προσέγγιση αυτή λιγότερο αποτελεσματική. Στα τεχνικά προβλήματα συγκαταλέγεται η αύξηση του μεγέθους του κώδικα και οι περιορισμοί της επακόλουθης λειτουργίας. Δύο διαφορετικά στοιχεία συνδυάζονται για την ουσιαστική αύξηση του κώδικα στον VLIW. Πρώτον, η παραγωγή επαρκούς αριθμού λειτουργιών εντός τμημάτων ευθύγραμμου κώδικα απαιτεί φιλόδοξες προσπάθειες ξεδιπλώματος βρόχων (όπως στα προηγούμενα παραδείγματα), γεγονός που οδηγεί στην αύξηση του μεγέθους του κώδικα. Δεύτερον, οποτεδήποτε οι εντολές δεν είναι πλήρεις, οι λειτουργικές μονάδες που δεν χρησιμοποιούνται αντιστοιχούν σε χαμένα bits της κωδικοποίησης εντολών. Στο Παράρτημα Z εξετάζουμε προσεγγίσεις χρονοδρομολόγησης σε επίπεδο λογισμικού, όπως η διασωλήνωση λογισμικού, η οποία μπορεί να αποφέρει τα οφέλη του ξεδιπλώματος χωρίς να απαιτείται αυτού του μεγέθους η επέκταση του κώδικα.

Για την αντιμετώπιση αυτής της αύξησης του μεγέθους του κώδικα, μερικές φορές χρησιμοποιούνται έξυπνες κωδικοποιήσεις. Για παράδειγμα, ενδέχεται μόνο ένα μεγάλο άμεσο πεδίο να μπορεί να χρησιμοποιηθεί από οποιαδήποτε λειτουργική μονάδα. Άλλη μία τεχνική σχετίζεται με τη συμπίεση των εντολών ενός της κύριας μνήμης και την επέκτασή τους όταν διαβάζονται εντός της κρυφής μνήμης ή όταν αποκωδικοποιούνται. Στο Παράρτημα Z, παρουσιάζονται και άλλες τεχνικές, ενώ συγχρόνως παρατίθεται η σημαντική επέκταση κώδικα που παρατηρείται στην IA-64.

Οι πρώτοι VLIWs λειτουργούσαν ακολουθιακά. Δεν υπήρχε κανενός είδους υλικό ανίχνευσης κινδύνων. Η δομή αυτή υπαγόρευε ότι η εισαγωγή διαστή-

Αναφορά μνήμης 1	Αναφορά μνήμης 2	Λειτουργία FP 1	Λειτουργία FP 2	Λειτουργία ακέραιων/διακλαδωση
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L. D F26, - 48(R1)		ADD. D F12, F10, F2	ADD. D F16, F14, F2	
		ADD.D F20,F18,F2	ADD.D F24,F22,F2	
S.D F4,0(R1)	S.D F8,-8(R1)	ADD.D F28,F26,F2		
S. D F12, - 16(R1)	S. D F16, - 24(R1)			DADDU R1, R1, #- 56
S.D F20,24(R1)	S.D F24,16(R1)			
S.D F28,8(R1)				BNE R1,R2,Loop

Σχήμα 2.19 Εντολές VLIW οι οποίες καταλαμβάνουν τον εσωτερικό βρόχο και αντικαθιστούν την ξεδιπλωμένη ακολουθία. Ο κώδικας αυτός εκτελείται εντός 9 κύκλων, κάνοντας την υπόθεση ότι δεν υφίσταται καθυστέρηση διακλάδωσης. Κανονικά, η καθυστέρηση διακλάδωσης θα έπρεπε, επίσης, να χρονοδρομολογηθεί. Ο ρυθμός έκδοσης ισούται με 23 λειτουργίες εντός 9 κύκλων ή με 2.5 λειτουργίες ανά κύκλο. Η αποτελεσματικότητα, το ποσοστό, δηλαδή, των διαθέσιμων θυρίδων που περιέχουν μια λειτουργία, ισούται περίπου με 60%. Για την επίτευξη αυτού του ρυθμού έκδοσης απαιτείται μεγαλύτερος αριθμός καταχωρητών από αυτόν που η MIPS θα χρησιμοποιούσε κανονικά για αυτόν το βρόχο. Η παραπάνω ακολουθία κώδικα του VLIW απαιτεί τουλάχιστον οκτώ καταχωρητές FP, ενώ η ίδια ακολουθία κώδικα μπορεί να χρησιμοποιήσει κατ' ελάχιστο δύο καταχωρητές FP στο βασικό επεξεργαστή της MIPS ή το πολύ πέντε καταχωρητές σε περίπτωση που εφαρμοστεί ξεδίπλωμα και χρονοδρομολόγηση.

ματος αδράνειας σε οποιαδήποτε διασωλήνωση λειτουργικής μονάδας προκαλεί την αδράνεια ολόκληρου του επεξεργαστή, καθώς όλες οι λειτουργικές μονάδες έπρεπε να διατηρούνται συγχρονισμένες. Παρόλο που ο μεταγλωττιστής μπορεί να είναι σε θέση να χρονοδρομολογήσει τις προσδιοριστικές λειτουργικές μονάδες με απώτερο σκοπό την αποφυγή των διαστημάτων αδράνειας, η πρόβλεψη όσον αφορά τις προσπελάσεις των δεδομένων που θα συναντήσουν διάστημα αδράνειας, καθώς και η χρονοδρομολόγησή τους είναι πολύ δύσκολη. Συνεπώς, οι κρυφές μνήμες έπρεπε να είναι απαγορευτικές (blocking) και να θέτουν *όλες* τις λειτουργικές μονάδες σε αδράνεια. Καθώς ο ρυθμός έκδοσης και ο αριθμός των αναφορών στη μνήμη γίνονται ολοένα και μεγαλύτεροι, αυτός ο περιορισμός συγχρονισμού δεν μπορεί να είναι αποδεκτός. Στους πιο πρόσφατους επεξεργαστές, οι λειτουργικές μονάδες διαθέτουν μεγαλύτερη ανεξαρτησία και ο μεταγλωττιστής χρησιμοποιείται για την αποφυγή των κινδύνων κατά το χρόνο έκδοσης, ενόσω οι έλεγχοι που διενεργούνται από το υλικό επιτρέπουν την ασύγχρονη εκτέλεση των εντολών, μόλις αυτές εκδοθούν.

Η συμβατότητα του δυαδικού κώδικα αποτέλεσε, επίσης, σημαντικό πρόβλημα λογικής για τους VLIWs. Στην αυστηρή προσέγγιση των VLIWs, η ακολουθία

κώδικα χρησιμοποιεί τόσο τον ορισμό του συνόλου εντολών όσο και τη λεπτομερή δομή της διασωλήνωσης, συμπεριλαμβανομένων τόσο των λειτουργικών μονάδων όσο και των λανθανόντων χρόνων τους. Επομένως, οι διαφορετικές προσεγγίσεις όσον αφορά τον αριθμό των λειτουργικών μονάδων και του ύψους του λανθάνοντος χρόνου τους απαιτούν διαφορετικές εκδόσεις του κώδικα. Η απαίτηση αυτή καθιστά τόσο τη μετάβαση από τη μία επιτυχημένη υλοποίηση στην επόμενη όσο και τη μετάβαση από μία συγκεκριμένη υλοποίηση σε κάποια επόμενη που διαθέτει διαφορετικό εύρος έκδοσης πιο δύσκολη απ' ό,τι στην περίπτωση της υπερβαθμωτής σχεδίασης. Βεβαίως, η επίτευξη υψηλής απόδοσης μέσω κάποιας νέας υπερβαθμωτής σχεδίασης μπορεί να απαιτεί την εκ νέου μεταγλώττιση. Ωστόσο, η δυνατότητα εκτέλεση παλαιών δυαδικών αρχείων αποτελεί πρακτικό πλεονέκτημα της υπερβαθμωτής προσέγγισης.

Η προσέγγιση EPIC, βασικό παράδειγμα της οποίας αποτελεί η αρχιτεκτονική IA-64, παρέχει λύσεις σε πολλά από τα προβλήματα που παρατηρήθηκαν στις πρώτες σχεδιάσεις της VLIW, λύσεις όπως η πιο εντατική χρήση της εικασίας λογισμικού και η εφαρμογή μεθόδων για την αντιμετώπιση του περιορισμού της εξάρτησης υλικού, διατηρώντας παράλληλα τη δυαδική συμβατότητα.

Η πιο σημαντική πρόκληση για όλους τους επεξεργαστές πολλαπλής έκδοσης είναι η προσπάθεια αξιοποίησης μεγάλου εύρους του ILP. Στην περίπτωση που ο παραλληλισμός επιτυγχάνεται μέσω του ξεδιπλώματος απλών βρόχων σε προγράμματα FP, ο αρχικός βρόχος πιθανότατα θα μπορούσε να εκτελεστεί αποτελεσματικά σε κάποιο διανυσματικό επεξεργαστή (κάτι που περιγράφεται στο Παράρτημα ΣΤ). Σε αυτού του είδους τις εφαρμογές δεν είναι σαφές αν προτιμότερη λύση είναι η χρήση επεξεργαστών πολλαπλής έκδοσης ή η χρήση διανυσματικών επεξεργαστών. Τα κόστη είναι παρόμοια και ο διανυσματικός επεξεργαστής, συνήθως, είτε διαθέτει την ίδια ταχύτητα είτε είναι γρηγορότερος. Τα ενδεχόμενα πλεονεκτήματα των επεξεργαστών πολλαπλής έκδοσης σε σύγκριση με τους διανυσματικούς επεξεργαστές σχετίζονται με την ικανότητά τους να επιτυγχάνουν την εφαρμογή παραλληλισμού σε λιγότερο δομημένες ακολουθίες κώδικα, καθώς και με τη δυνατότητα τους να αποθηκεύουν εύκολα στην προσωρινή μνήμη όλες τις μορφές δεδομένων. Για τους λόγους αυτούς, οι προσεγγίσεις πολλαπλής έκδοσης αποτελούν την κύρια μέθοδο αξιοποίησης του παραλληλισμού επιπέδου εντολών, ενώ οι διανυσματικές προσεγγίσεις αποτελούν κυρίως τρόπο επέκτασης των επεξεργαστών αυτών.

2.8 Αξιοποίηση του ILP Χρησιμοποιώντας Δυναμική Χρονοδρομολόγηση, Πολλαπλή Έκδοση και Εικασία

Μέχρι τώρα, έχουμε διερευνήσει τον τρόπο λειτουργίας των μεμονωμένων μηχανισμών της δυναμικής χρονοδρομολόγησης, της πολλαπλής έκδοσης και τη

εικασίας. Στην ενότητα αυτή, συνδυάζουμε όλες τις παραπάνω προσεγγίσεις, γεγονός που οδηγεί σε μικροαρχιτεκτονική που είναι σχεδόν παρόμοια με αυτή των σύγχρονων μικροεπεξεργαστών. Για λόγους απλότητας, θα εξετάσουμε μονάχα ένα ρυθμό έκδοσης της τάξης των δύο εντολών ανά κύκλο, ωστόσο οι έννοιες που θίγονται δεν διαφέρουν από όσα ισχύουν στους σύγχρονους επεξεργαστές που εκδίδουν τρεις ή περισσότερες εντολές ανά κύκλο ρολογιού.

Ας υποθέσουμε ότι θέλουμε να επεκτείνουμε τον αλγόριθμο του Tomasulo ώστε να υποστηρίζει την υπερβαθμωτή διασωλήνωση διπλής έκδοσης που διαθέτει ξεχωριστή μονάδα ακεραίων και μονάδα κινητής υποδιαστολής, καθεμία εκ των οποίων μπορεί να εκκινεί μια λειτουργία σε κάθε κύκλο ρολογιού. Στους σταθμούς κράτησης, δεν θέλουμε να πραγματοποιείται έκδοση εντολών εκτός σειράς, καθώς αυτό μπορεί να οδηγήσει σε παραβίαση της σημασιολογίας του προγράμματος. Για τη μέγιστη δυνατή αξιοποίηση της δυναμικής χρονοδρομολόγησης, η διασωλήνωση επιτρέπεται σε κάθε κύκλο ρολογιού να εκδίδει οποιοδήποτε συνδυασμό δύο εντολών, χρησιμοποιώντας το υλικό χρονοδρομολόγησης, έτσι ώστε να πραγματοποιείται η ανάθεση λειτουργιών στη μονάδα ακεραίων και στη μονάδα κινητής υποδιαστολής. Επειδή η αλληλεπίδραση των ακέραιων εντολών και των εντολών κινητής υποδιαστολής είναι ιδιαίτερα σημαντική, θα επεκτείνουμε, επίσης, την προσέγγιση του Tomasulo έτσι ώστε να ασχολείται με τις λειτουργικές μονάδες και τους καταχωρητές ακεραίων και κινητής υποδιαστολής, ενσωματώνοντας συγχρόνως την υποθετική εκτέλεση.

Έχουν χρησιμοποιηθεί δύο διαφορετικές προσεγγίσεις για την έκδοση πολλαπλών εντολών ανά κύκλο ρολογιού στους δυναμικά χρονοδρομολογημένους επεξεργαστές και οι δύο εκ των οποίων βασίζονται στην παρατήρηση ότι το πιο σημαντικό στοιχείο είναι η ανάθεση κάποιου σταθμού κράτησης και η ενημέρωση των πινάκων ελέγχου της διασωλήνωσης. Η μία προσέγγιση αφορά την εκτέλεση του βήματος αυτού σε μισό κύκλο ρολογιού, έτσι ώστε σε κάθε κύκλο ρολογιού να είναι εφικτή η επεξεργασία δύο εντολών. Η δεύτερη εναλλακτική λύση αφορά τη δόμηση της απαραίτητης λογικής για τον ταυτόχρονο χειρισμό δύο εντολών, συμπεριλαμβανομένων και των οποιονδήποτε πιθανών εξαρτήσεων μεταξύ των εντολών. Οι σύγχρονοι υπερβαθμωτοί επεξεργαστές που εκδίδουν τέσσερις ή περισσότερες εντολές ανά κύκλο ρολογιού συχνά υποστηρίζουν και τις δύο προσεγγίσεις: Προβαίνουν τόσο στη διασωλήνωση όσο και στη διεύρυνση της λογικής της έκδοσης.

Ο συνδυασμός της υποθετικής δυναμικής χρονοδρομολόγησης με την πολλαπλή έκδοση απαιτεί την αντιμετώπιση μίας επιπλέον πρόκλησης που σχετίζεται με τη διασωλήνωση: πρέπει να είμαστε σε θέση να ολοκληρώνουμε και να επικυρώνουμε πολλαπλές εντολές ανά κύκλο ρολογιού. Όπως στην περίπτωση της έκδοσης πολλαπλών εντολών, έτσι και εδώ οι έννοιες είναι απλές, παρόλο που η υλοποίηση μπορεί να είναι το ίδιο απαιτητική με τη διαδικασία της έκδοσης και της μετονομασίας των καταχωρητών. Παρακάτω, παρουσιάζουμε τις σχετικές έννοιες με τη βοήθεια ενός παραδείγματος.

Παράδειγμα: Ας εξετάσουμε την εκτέλεση του παρακάτω βρόχου, ο οποίος αυξάνει κατά μία μονάδα την τιμή κάθε στοιχείου ενός πίνακα ακεραίων σε επεξεργαστή διπλής έκδοσης, τόσο υποστηρίζοντας όσο και μη υποστηρίζοντας την εικασία:

```
Loop:   LD      R2,0(R1)      ;R2=στοιχείο πίνακα
        DADDIU  R2,R2,#1      ;αύξηση του R2
        SD      R2,0(R1)      ;αποθήκευση αποτελέσματος
        DADDIU  R1,R1,#8      ;αύξηση δείκτη
        BNE     R2,R3,LOOP    ;διακλάδωση αν δεν πρόκειται
                              για το τελευταίο στοιχείο
```

Ας υποθέσουμε ότι υπάρχουν ξεχωριστές λειτουργικές μονάδες ακεραίων για τον υπολογισμό της ενεργού διεύθυνσης, για τις πράξεις της ALU και για την αξιολόγηση της συνθήκης διακλάδωσης. Θα δημιουργήσουμε έναν πίνακα για τις τρεις πρώτες επαναλήψεις του βρόχου αυτού και για τους δύο επεξεργαστές. Υποθέτουμε ότι σε κάθε κύκλο ρολογιού μπορούν να επικυρωθούν έως δύο εντολές κάθε τύπου.

Απάντηση: Τα Σχήματα 2.20 και 2.21 παρουσιάζουν την απόδοση ενός δυναμικά χρονοδρομολογημένου επεξεργαστή διπλής έκδοσης τόσο με εφαρμογή όσο και χωρίς εφαρμογή της εικασίας. Στην περίπτωση αυτή, όπου μια διακλάδωση μπορεί να αποτελέσει σημαντικό περιοριστικό παράγοντα της απόδοσης, η εικασία βοηθά σε σημαντικό βαθμό. Στον υποθετικό επεξεργαστή, η τρίτη διακλάδωση εκτελείται στον κύκλο ρολογιού 13, ενώ στη μη υποθετική διασωλήνωση εκτελείται στον κύκλο 19. Επειδή στη μη υποθετική διασωλήνωση ο ρυθμός ολοκλήρωσης από τα πρώτα βήματα δεν μπορεί να ακολουθήσει το ρυθμό έκδοσης, η διασωλήνωση θα τεθεί σε αδράνεια μετά από την έκδοση μερικών ακόμα επαναλήψεων. Η απόδοση του μη υποθετικού επεξεργαστή μπορεί να βελτιωθεί παρέχοντας τη δυνατότητα στις εντολές φόρτωσης να ολοκληρώνουν τον υπολογισμό της ενεργού διεύθυνσης πριν τη λήψη της απόφασης για κάποια διακλάδωση, ωστόσο αν δεν επιτρέπεται η πραγματοποίηση υποθετικών προσπελάσεων, η βελτίωση αυτή θα αποφέρει κέρδος ίσο μόνο με 1 κύκλο ρολογιού ανά επανάληψη.

Το παράδειγμα αυτό απεικονίζει ξεκάθαρα τον τρόπο με τον οποίο η εικασία παρουσιάζει πλεονεκτήματα στην περίπτωση που υπάρχουν διακλαδώσεις με εξάρτηση δεδομένων, οι οποίες σε διαφορετική περίπτωση θα περιόριζαν την απόδοση. Το πλεονέκτημα αυτό, ωστόσο, εξαρτάται από την ακρίβεια της πρόβλεψης της διακλάδωσης. Η εσφαλμένη εικασία δεν πρόκειται να οδηγήσει στη βελτίωση της απόδοσης, αλλά στην πραγματικότητα θα οδηγήσει στη μείωσή της.

Αριθμός επανάληψης	Εντολές	Έκδοση κατά τον κύκλο ρολογιού	Εκτέλεση κατά τον κύκλο ρολογιού	Προσπέλαση μνήμης κατά τον κύκλο ρολογιού	Εγγραφή του CDB κατά τον κύκλο ρολογιού	Σχόλιο
1	LD R2,0(R1)	1	2	3	4	Πρώτη έκδοση
1	DADDIU R2, R2, #1	1	5		6	Αναμονή για LW
1	SD R2,0(R1)	2	3	7		Αναμονή για DADDIU
1	DADDIU R1, R1, #8	2	3		4	Άμεση εκτέλεση
1	BNE R2,R3,LOOP	3	7			Αναμονή για DADDIU
2	LD R2,0(R1)	4	8	9	10	Αναμονή για BNE
2	DADDIU R2, R2, #1	4	11		12	Αναμονή για LW
2	SD R2,0(R1)	5	9	13		Αναμονή για DADDIU
2	DADDIU R1, R1, #8	5	8		9	Αναμονή για BNE
2	BNE R2,R3,LOOP	6	13			Αναμονή για DADDIU
3	LD R2,0(R1)	7	14	15	16	Αναμονή για BNE
3	DADDIU R2, R2, #1	7	17		18	Αναμονή για LW
3	SD R2,0(R1)	8	15	19		Αναμονή για DADDIU
3	DADDIU R1, R1, #8	8	14		15	Αναμονή για BNE
3	BNE R2,R3,LOOP	9	19			Αναμονή για DADDIU

Σχήμα 2.20 Χρόνος έκδοσης, εκτέλεσης και εγγραφής αποτελέσματος για την περίπτωση της διασωλήνωσης διπλής έκδοσης *χωρίς* εικασία. Αξίζει να σημειωθεί ότι η LD η οποία ακολουθεί την BNE δεν μπορεί να ξεκινήσει να εκτελείται νωρίτερα, διότι πρέπει να περιμένει έως ότου να προσδιοριστεί το αποτέλεσμα της διακλάδωσης. Αυτού του είδους το πρόγραμμα, που περιλαμβάνει διακλαδώσεις με εξάρτηση δεδομένων οι οποίες δεν μπορούν να επιλυθούν νωρίτερα, καθιστά σαφή την ισχύ της εικασίας. Η παρουσία ξεχωριστών λειτουργικών μονάδων για τον υπολογισμό διευθύνσεων, για πράξεις ALU και για την αξιολόγηση της συνθήκης διακλάδωσης επιτρέπει την εκτέλεση πολλαπλών εντολών στον ίδιο κύκλο. Το Σχήμα 2.21 παρουσιάζει το ίδιο παράδειγμα, υποστηρίζοντας την εικασία.

2.9 Εξελιγμένες Τεχνικές Παράδοσης Εντολών και Εικασίας

Στις διασωληνώσεις υψηλής απόδοσης, ιδίως σε αυτές που υποστηρίζουν πολλαπλή έκδοση, η ορθή πρόβλεψη των διακλαδώσεων δεν είναι αρκετή. Στην πραγματικότητα χρειάζεται να υπάρχει η δυνατότητα παράδοσης ροής εντολών υψηλού εύρους ζώνης (high bandwidth). Στους πρόσφατους επεξεργαστές πολλαπλής έκδοσης, αυτό μεταφράζεται σε παράδοση 4-8 εντολών σε κάθε κύκλο ρολογιού. Πρώτα, θα εξετάσουμε τις μεθόδους για την αύξηση του εύρους ζώνης παράδοσης των εντολών. Στη συνέχεια θα στρέψουμε το ενδιαφέρον μας σε αρκετά βασικά ζητήματα που σχετίζονται με την υλοποίηση εξελιγμένων τεχνικών εικασίας, μεταξύ των οποίων περιλαμβάνονται η χρήση της μετονομασίας καταχωρητών σε σύγκριση με τη χρήση των απομονωτών αναδιάταξης, η εντατική

Αριθμός επανάληψης	Εντολές	Έκδοση κατά τον κύκλο ρολογιού	Εκτέλεση κατά τον κύκλο ρολογιού	Προσπέλαση ανάγνωσης κατά τον κύκλο ρολογιού	Εγγραφή του CDB κατά τον κύκλο ρολογιού	Επικύρωση κατά τον κύκλο ρολογιού	Σχόλιο
1	LD R2,0(R1)	1	2	3	4	5	Πρώτη έκδοση
1	DADDIU R2,R2,#1	1	5		6	7	Αναμονή για LW
1	SD R2,0(R1)	2	3			7	Αναμονή για DADDIU
1	DADDIU R1,R1,#8	2	3		4	8	Επικύρωση εντός σειράς
1	BNE R2,R3,LOOP	3	7			8	Αναμονή για DADDIU
2	LD R2,0(R1)	4	5	6	7	9	Καθυστέρηση μη εκτέλεσης
2	DADDIU R2,R2,#1	4	8		9	10	Αναμονή για LW
2	SD R2,0(R1)	5	6			10	Αναμονή για DADDIU
2	DADDIU R1,R1,#8	5	6		7	11	Επικύρωση εντός σειράς
2	BNE R2,R3,LOOP	6	10			11	Αναμονή για DADDIU
3	LD R2,0(R1)	7	8	9	10	12	Όσο το δυνατό νωρίτερα
3	DADDIU R2,R2,#1	7	11		12	13	Αναμονή για LW
3	SD R2,0(R1)	8	9			13	Αναμονή για DADDIU
3	DADDIU R1,R1,#8	8	9		10	14	Πρωτύτερη εκτέλεση
3	BNE R2,R3,LOOP	9	13			14	Αναμονή για DADDIU

Σχήμα 2.21 **Χρόνος έκδοσης, εκτέλεσης και εγγραφής αποτελέσματος για την περίπτωση της διασωλήνωσης διπλής έκδοσης *με* εικασία.** Αξίζει να σημειωθεί ότι η LD που ακολουθεί την BNE μπορεί να ξεκινήσει να εκτελείται νωρίς, διότι πρόκειται για υποθετική εκτέλεση.

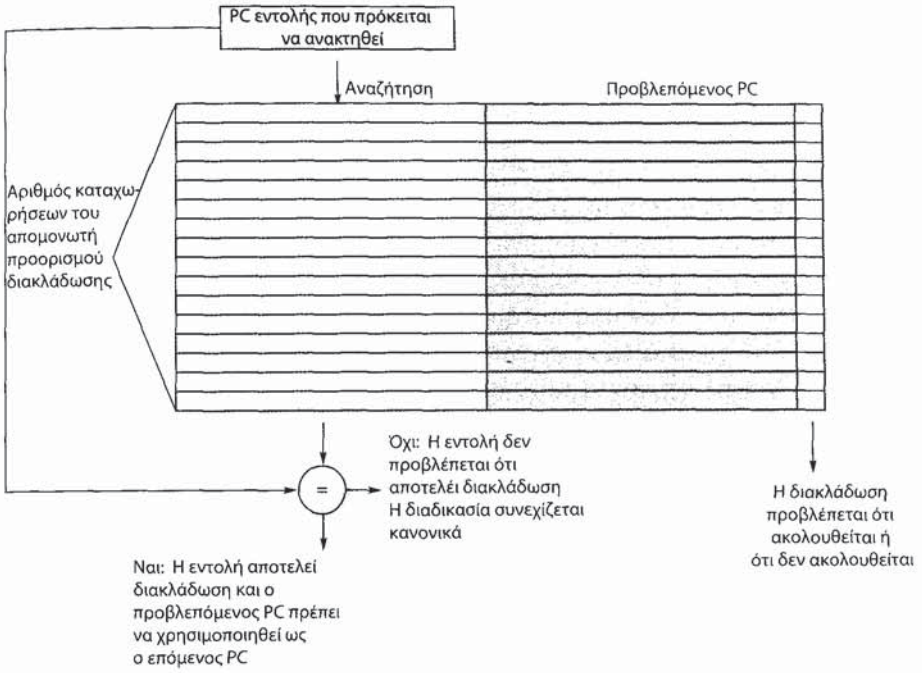
χρήση της εικασίας και η παρουσίαση μιας τεχνικής που είναι γνωστή ως πρόβλεψη τιμής (value prediction), η οποία μπορεί να βελτιώσει ακόμα περισσότερο τον ILP.

Αύξηση του Εύρους Ζώνης Ανάκτησης Εντολών

Οι επεξεργαστές πολλαπλής έκδοσης απαιτούν ο μέσος αριθμός των εντολών που ανακτώνται σε κάθε κύκλο ρολογιού να είναι τουλάχιστον ίσος με τη μέση ρυθμοαπόδοση (throughput). Βεβαίως, η ανάκτηση των εντολών αυτών απαιτεί την ύπαρξη διαδρομών μεγάλου εύρους προς την κρυφή μνήμη εντολών, ωστόσο οι μεγαλύτερες δυσκολίες εγείρονται από τον τρόπο χειρισμού των διακλαδώσεων. Στην ενότητα αυτή θα διερευνήσουμε δύο μεθόδους για την αντιμετώπιση των διακλαδώσεων και στη συνέχεια θα εξετάσουμε τον τρόπο με τον οποίο οι σύγχρονοι επεξεργαστές συνδυάζουν την πρόβλεψη εντολών με τις λειτουργίες προανάκτησης.

Απομονωτές Προορισμού Διακλάδωσης

Για να μειώσουμε την ποινή διακλάδωσης στην απλή διασωλήνωσή μας των πέντε σταδίων, καθώς και σε διασωληνώσεις μεγαλύτερου βάθους, χρειάζεται να γνωρί-



Σχήμα 2.22 Απομονωτής προορισμού διακλάδωσης. Ο PC της εντολής που ανακτάται συγκρίνεται με σύνολο διευθύνσεων εντολών οι οποίες βρίσκονται αποθηκευμένες στην πρώτη στήλη. Αυτές αντικατοπτρίζουν τις διευθύνσεις γνωστών διακλαδώσεων. Αν ο PC είναι ίδιος με κάποια από αυτές τις καταχωρίσεις, τότε η εντολή που ανακτάται είναι διακλάδωση που ακολουθείται και το δεύτερο πεδίο, το πεδίο δηλαδή του προβλεπόμενου PC, περιλαμβάνει την πρόβλεψη που αφορά την τιμή του επόμενου PC μετά τη διακλάδωση. Η ανάκτηση ξεκινά αμέσως από τη διεύθυνση αυτή. Το τρίτο πεδίο, το οποίο είναι προαιρετικό, μπορεί να χρησιμοποιηθεί για την εισαγωγή επιπλέον bits κατάστασης για την πρόβλεψη.

ζουμε αν η εντολή που δεν έχει ακόμα αποκωδικοποιηθεί είναι διακλάδωση και στην περίπτωση που αυτό ισχύει ποια είναι η τιμή του επόμενου PC. Αν η εντολή είναι διακλάδωση και γνωρίζουμε την τιμή του επόμενου PC, τότε μπορούμε να έχουμε ποινή διακλάδωσης ίση με το μηδέν. Η κρυφή μνήμη πρόβλεψης διακλάδωσης η οποία αποθηκεύει την προβλεπόμενη διεύθυνση της εντολής που ακολουθεί μια διακλάδωση ονομάζεται *απομονωτής προορισμού διακλάδωσης* (branch-target buffer) ή *κρυφή μνήμη προορισμού διακλάδωσης* (branch-target cache). Στο Σχήμα 2.22 παρουσιάζεται ένας απομονωτής προορισμού διακλάδωσης.

Επειδή ο απομονωτής προορισμού διακλάδωσης προβλέπει τη διεύθυνση της επόμενης εντολής και την αποστέλλει *πριν* την αποκωδικοποίηση της εντολής, *πρέπει* να γνωρίζουμε αν η ανακτηθείσα εντολή έχει προβλεφθεί ότι αποτελεί διακλάδωση που πρόκειται να ακολουθηθεί. Αν ο PC της εντολής που έχει ανα-

κτηθεί είναι ίδιος με κάποιον PC του απομονωτή πρόβλεψης, τότε ο επόμενος PC λαμβάνει την τιμή του αντίστοιχου προβλεπόμενου PC. Το υλικό αυτού του απομονωτή προορισμού διακλάδωσης είναι στην ουσία ίδιο με το υλικό της κρυφής μνήμης.

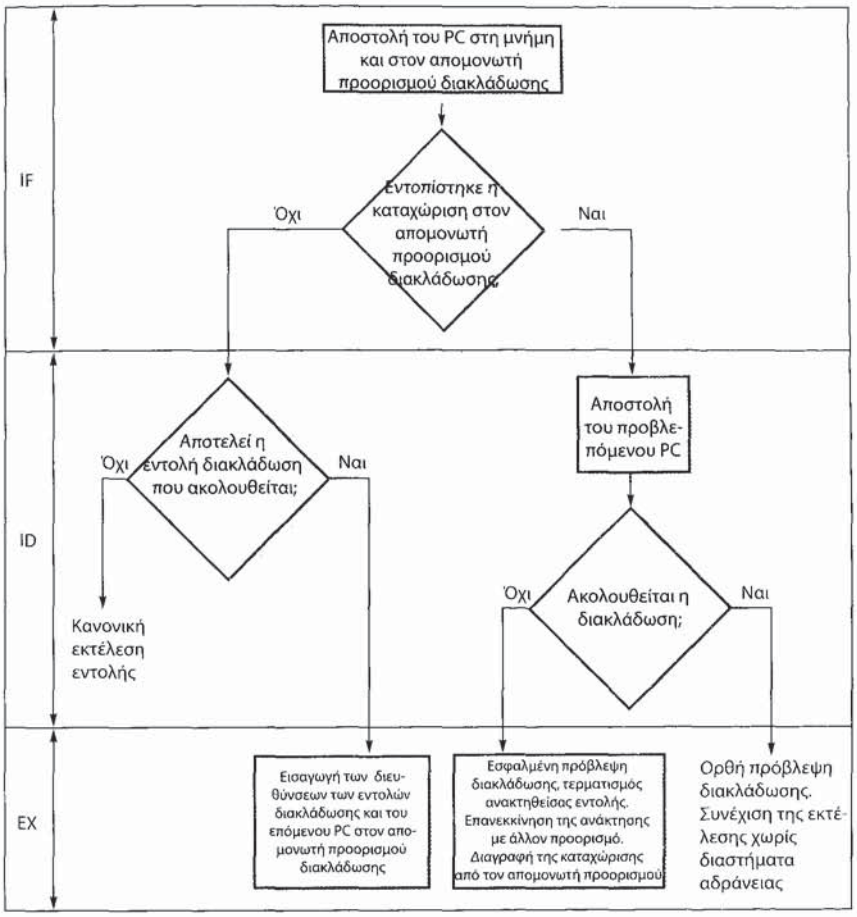
Σε περίπτωση που ο απομονωτής προορισμού διακλάδωσης διαθέτει καταχώριση με διεύθυνση ίδια με αυτή της εντολής που έχει ανακτηθεί, τότε η ανάκτηση ξεκινά αμέσως από τον προβλεπόμενο PC. Αξίζει να σημειωθεί ότι σε αντίθεση με κάποιον απομονωτή πρόβλεψης διακλάδωσης, η καταχώριση πρόβλεψης πρέπει να είναι ίδια με την εντολή αυτή, διότι ο προβλεπόμενος PC θα αποσταλεί, πριν να είναι καν γνωστό αν η εντολή είναι διακλάδωση. Σε περίπτωση που ο επεξεργαστής δεν ήλεγχε αν η καταχώριση είναι ίδια με αυτόν τον PC, τότε θα αποστελλόταν λανθασμένος PC για εντολές που δεν είναι διακλαδώσεις, γεγονός που θα είχε ως αποτέλεσμα ο επεξεργαστής να είναι πιο αργός. Χρειάζεται απλά να αποθηκεύουμε στον απομονωτή προορισμού διακλάδωσης τις διακλαδώσεις που προβλέπεται ότι ακολουθούνται, καθώς αυτού του είδος οι διακλαδώσεις απλά οδηγούν στην ανάκτηση της επόμενης ακολουθιακής εντολής, όπως θα συνέβαινε στην περίπτωση οποιασδήποτε εντολής δεν ανήκει στην κατηγορία των διακλαδώσεων.

Το Σχήμα 2.23 παρουσιάζει τα λεπτομερή βήματα της χρήσης του απομονωτή προορισμού διακλάδωσης για την απλή διασωλήνωση των πέντε σταδίων. Από το σχήμα αυτό μπορεί να διαπιστώσει κανείς ότι, σε περίπτωση που κάποια καταχώριση πρόβλεψης διακλάδωσης βρεθεί εντός του απομονωτή και η πρόβλεψη είναι ορθή, δεν θα υπάρχει καμία καθυστέρηση διακλάδωσης. Διαφορετικά, θα υπάρχει ποινή τουλάχιστον δύο κύκλων. Η αντιμετώπιση των εσφαλμένων προβλέψεων και των αστοχιών αποτελεί σημαντική πρόκληση, καθώς συνήθως χρειάζεται να προκαλέσουμε την παύση της ανάκτησης εντολών, ενώ συγχρόνως εκτελούμε τη διαδικασία επανεγγραφής της καταχώρισης του απομονωτή. Κατά συνέπεια, θα θέλαμε να καταστεί η διαδικασία αυτή ταχύτερη, έτσι ώστε να ελαχιστοποιείται η ποινή.

Για την αξιολόγηση της λειτουργίας των απομονωτών προορισμού διακλάδωσης, πρέπει πρώτα να προσδιορίσουμε τις ποινές για όλες τις πιθανές περιπτώσεις. Το Σχήμα 2.24 περιέχει τις πληροφορίες αυτές για την απλή διασωλήνωση των πέντε σταδίων.

Παράδειγμα: Ας προσδιορίσουμε τη συνολική ποινή διακλάδωσης του απομονωτή προορισμού διακλάδωσης, κάνοντας την υπόθεση ότι ισχύουν οι κύκλοι ποινής που αφορούν τις μεμονωμένες εσφαλμένες προβλέψεις οι οποίοι παρατίθενται στο Σχήμα 2.24. Θα προβούμε στις παρακάτω προβλέψεις όσον αφορά την ακρίβεια της πρόβλεψης και το ρυθμό ευστοχίας:

- *Η ακρίβεια πρόβλεψης ισούται με 90%* (για τις εντολές εντός του απομονωτή)



Σχήμα 2.23 **Τα βήματα που περιλαμβάνονται στο χειρισμό των εντολών με τη βοήθεια του απομονωτή προορισμού διακλάδωσης.**

- Ο ρυθμός ευστοχίας ισούται με 90% (για τις διακλαδώσεις που έχουν προβλεφθεί ότι θα ακολουθηθούν)

Απάντηση: Υπολογίζουμε την ποινή εξετάζοντας την πιθανότητα πραγματοποίησης των δύο γεγονότων: στην πρώτη περίπτωση η διακλάδωση έχει προβλεφθεί ότι θα ακολουθηθεί, αλλά εν τέλει δεν ακολουθείται και στη δεύτερη περίπτωση η διακλάδωση ακολουθείται, αλλά δεν βρίσκεται εντός του απομονωτή. Και οι δύο περιπτώσεις συνεπάγονται ποινή της τάξης των 2 κύκλων.

$$
\begin{aligned}
\text{Πιθανότητα (διακλάδωσης εντός απομονωτή, χωρίς να ακολουθείται)} &= \\
\text{Ποσοστό ρυθμού ευστοχίας απομονωτή} \times \text{Ποσοστό εσφ. προβλέψεων} &= \\
90\% \times 10\% &= 0.09 \\
\text{Πιθανότητα (διακλάδωσης εκτός απομονωτή, η οποία ακολουθείται)} &= 10\% \\
\text{Ποινή διακλάδωσης} &= (0.09 + 0.10) \times 2 \\
\text{Ποινή διακλάδωσης} &= 0.38
\end{aligned}
$$

Η ποινή αυτή έρχεται σε σύγκριση με την ποινή διακλάδωσης των καθυστερημένων διακλαδώσεων που ανέρχεται σε 0.5 κύκλους ανά διακλάδωση, κάτι το οποίο αξιολογούμε στο Παράρτημα Α. Ωστόσο, χρειάζεται να έχουμε κατά νου ότι η βελτίωση που επιτυγχάνεται μέσω της δυναμικής πρόβλεψης διακλάδωσης μεγεθύνεται παράλληλα με το μήκος της διασωλήνωσης και ως εκ τούτου μεγαλώνει η καθυστέρηση διακλάδωσης. Επιπλέον, καλύτεροι μηχανισμοί πρόβλεψης αποφέρουν υψηλότερη απόδοση.

Μία παραλλαγή του απομονωτή προορισμού διακλάδωσης αφορά την αποθήκευση μίας ή περισσότερων *εντολών προορισμού* (target instructions) είτε στη θέση είτε συνοδευτικά της προβλεπόμενης *διεύθυνσης προορισμού*. Η παραλλαγή αυτή έχει δύο ενδεχόμενα πλεονεκτήματα. Πρώτον, η προσπέλαση του

Εντολή εντός του απομονωτή	Πρόβλεψη	Πραγματική διακλάδωση	Κύκλοι ποινής
ναι	ακολουθείται	ακολουθείται	0
ναι	ακολουθείται	δεν ακολουθείται	2
όχι		ακολουθείται	2
όχι		δεν ακολουθείται	0

Σχήμα 2.24 Ποινές για όλους τους πιθανούς συνδυασμούς που αφορούν τόσο το αν η διακλάδωση βρίσκεται εντός του απομονωτή όσο και αν στην πραγματικότητα ακολουθείται ή όχι, κάνοντας την υπόθεση ότι στον απομονωτή αποθηκεύουμε μόνο τις διακλαδώσεις που ακολουθούνται. Δεν υπάρχει καμία ποινή διακλάδωσης αν τα πάντα έχουν προβλεφθεί ορθά και η διακλάδωση βρίσκεται εντός του απομονωτή προορισμού. Αν η διακλάδωση δεν έχει προβλεφθεί ορθά, τότε η ποινή είναι ίση με 1 κύκλο ρολογιού για την ενημέρωση του απομονωτή με τις ορθές πληροφορίες (κατά τη διάρκεια του οποίου δεν μπορεί να ανακτηθεί καμία εντολή) και 1 κύκλο ρολογιού, σε περίπτωση που αυτό είναι απαραίτητο, για την εκ νέου εκκίνηση της ανάκτησης της επόμενης ορθής εντολής της διακλάδωσης. Σε περίπτωση που η διακλάδωση δεν βρεθεί και ακολουθηθεί, τότε έχουμε ποινή της τάξης των δύο κύκλων, χρονικό διάστημα κατά το οποίο ενημερώνεται ο απομονωτής.

απομονωτή προορισμού διακλάδωσης επιτρέπεται να δαπανά περισσότερο χρόνο από αυτόν που μεσολαβεί μεταξύ διαδοχικών ανακτήσεων εντολών, κάτι που πιθανότατα επιτρέπει την ύπαρξη μεγαλύτερου απομονωτή προορισμού διακλάδωσης. Δεύτερον, η αποθήκευση των πραγματικών εντολών προορισμού μάς παρέχει τη δυνατότητα να εκτελούμε μια διαδικασία βελτιστοποίησης, γνωστή ως *αναδίπλωση διακλάδωσης* (branch folding). Η αναδίπλωση διακλάδωσης μπορεί να χρησιμοποιηθεί, έτσι ώστε να επιτευχθούν διακλαδώσεις άνευ συνθήκης που απαιτούν 0 κύκλους και μερικές φορές και διακλαδώσεις υπό συνθήκη που απαιτούν 0 κύκλους. Ας εξετάσουμε τον απομονωτή προορισμού διακλάδωσης που αποθηκεύει εντολές από την προβλεπόμενη διαδρομή, του οποίου η προσπέλαση πραγματοποιείται χρησιμοποιώντας τη διεύθυνση μιας διακλάδωσης άνευ συνθήκης. Η μόνη λειτουργία της διακλάδωσης άνευ συνθήκης αφορά την τροποποίηση του PC. Συνεπώς, όταν ο απομονωτής προορισμού διακλάδωσης σηματοδοτεί την ύπαρξη ευστοχίας και επισημαίνει ότι πρόκειται για διακλάδωση άνευ συνθήκης, η διασωλήνωση μπορεί απλά να τοποθετήσει την εντολή του απομονωτή προορισμού διακλάδωσης στη θέση της εντολής που επιστρέφεται από την κρυφή μνήμη (η οποία είναι η διακλάδωση άνευ συνθήκης). Αν ο επεξεργαστής εκδίδει πολλαπλές εντολές ανά κύκλο, τότε ο απομονωτής χρειάζεται να παρέχει πολλαπλές εντολές για την επίτευξη των μέγιστων δυνατών ωφελειών. Σε μερικές περιπτώσεις, όταν οι κωδικοί συνθήκης έχουν τεθεί εκ των προτέρων, μπορεί να είναι εφικτή η εξάλειψη του κόστους που συνοδεύει τις διακλαδώσεις υπό συνθήκη.

Μηχανισμοί Πρόβλεψης Διεύθυνσης Επιστροφής

Όσο προσπαθούμε να αυξήσουμε τις ευκαιρίες εφαρμογής και την ακρίβεια της εικασίας, ερχόμαστε αντιμέτωποι με την πρόκληση των έμμεσων αλμάτων (indirect jumps), των αλμάτων, δηλαδή, των οποίων η διεύθυνση προορισμού μεταβάλλεται κατά το χρόνο εκτέλεσης. Παρόλο που τα προγράμματα γλωσσών υψηλού επιπέδου θα παραγάγουν τέτοιου είδους άλματα για την κλήση έμμεσων διαδικασιών, για προτάσεις select ή case και εντολές goto της FORTRAN, η συντριπτική πλειονότητα των έμμεσων αλμάτων σχετίζεται με τις επιστροφές των διαδικασιών (procedure returns). Για παράδειγμα, στα μετροπρογράμματα του SPEC95, οι επιστροφές των διαδικασιών σχετίζονται με περισσότερο από το 15% των διακλαδώσεων, καθώς και με τη συντριπτική πλειονότητα των έμμεσων αλμάτων. Στις αντικειμενοστρεφείς γλώσσες, όπως η C++ και η Java, οι επιστροφές των διαδικασιών είναι ακόμα πιο συχνές. Επομένως, φαίνεται ότι είναι χρήσιμο να εστιάσει κανείς στις επιστροφές των διαδικασιών.

Παρόλο που οι επιστροφές των διαδικασιών μπορούν να προβλεφθούν με τη βοήθεια του απομονωτή προορισμού διακλάδωσης, η ακρίβεια αυτής της τεχνικής πρόβλεψης μπορεί να είναι χαμηλή στην περίπτωση που η διαδικασία

καλείται από πολλαπλά σημεία και οι κλήσεις που προέρχονται από κάποιο σημείο δεν πραγματοποιούνται στο ίδιο χρονικό διάστημα. Για παράδειγμα, στο SPEC CPU95, οι εντατικοί μηχανισμοί πρόβλεψης επιτυγχάνουν ακρίβεια μικρότερη του 60% για αυτού του είδους τις διακλαδώσεις επιστροφής. Για την αντιμετώπιση του προβλήματος αυτού, μερικές σχεδιάσεις χρησιμοποιούν ένα μικρό απομονωτή διευθύνσεων επιστροφής, ο οποίος λειτουργεί ως στοίβα. Η δομή αυτή αποθηκεύει σε κρυφή μνήμη τις πιο πρόσφατες διευθύνσεις επιστροφής: εφαρμόζει την ώθηση (push) μιας διεύθυνσης επιστροφής στη στοίβα κατά την κλήση και την απώθηση (pop) μίας εξ αυτών κατά την επιστροφή. Αν η κρυφή μνήμη είναι αρκετά μεγάλη (δηλαδή τόσο μεγάλη όσο το μέγιστο βάθος κλήσεων), τότε θα προβλέψει τις επιστροφές με απόλυτη ορθότητα. Το Σχήμα 2.25 παρουσιάζει την απόδοση ενός τέτοιου είδους απομονωτή επιστροφής που διαθέτει 0-16 στοιχεία, όταν αξιολογήθηκε με τη βοήθεια μερικών μετροπρογραμμάτων του SPEC CPU95. Στην Ενότητα 3.2, όπου θα εξετάσουμε τις μελέτες που έχουν εκπονηθεί γύρω από τον ILP θα χρησιμοποιήσουμε έναν παρόμοιο μηχανισμό πρόβλεψης επιστροφών.

Ολοκληρωμένες Μονάδες Ανάκτησης Εντολών

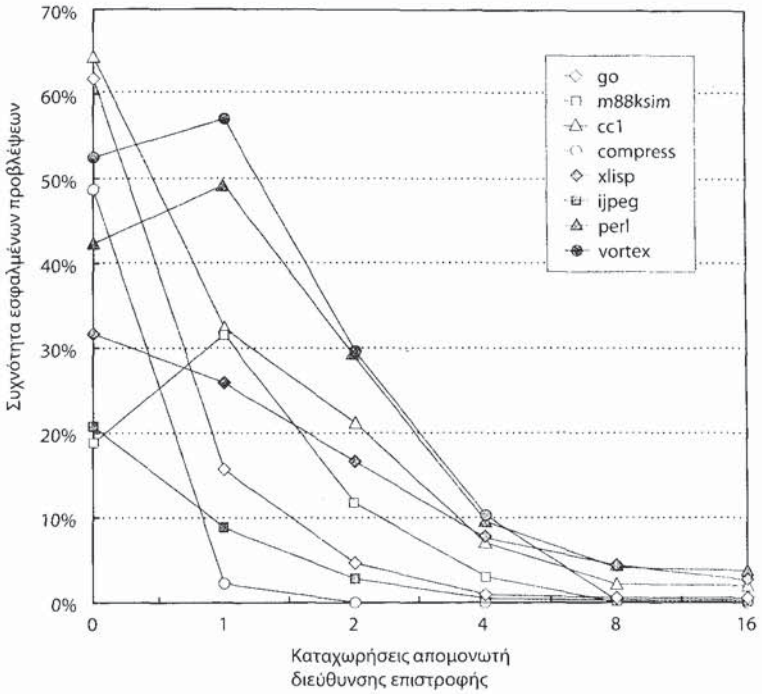
Για την ικανοποίηση των απαιτήσεων των επεξεργαστών πολλαπλής έκδοσης, πολλοί σχεδιαστές τελευταία έχουν επιλέξει να υλοποιήσουν μια ολοκληρωμένη μονάδα ανάκτησης εντολών (integrated instruction fetch unit) ως μια ξεχωριστή αυτόνομη μονάδα, η οποία τροφοδοτεί με εντολές την υπόλοιπη διασωλήνωση. Ουσιαστικά, με αυτόν τον τρόπο γίνεται η παραδοχή ότι η χρήση της ανάκτησης εντολών ως ένα απλό βήμα της σωλήνωσης, έχοντας ως δεδομένη την υψηλή πολυπλοκότητα της πολλαπλής έκδοσης, δεν αποτελεί πλέον έγκυρη διαδικασία.

Αντιθέτως, οι πρόσφατες σχεδιάσεις χρησιμοποιούν μια ολοκληρωμένη μονάδα ανάκτησης εντολών στην οποία έχουν ενσωματωθεί αρκετές λειτουργίες:

1. *Ενσωματωμένη πρόβλεψη διακλάδωσης* (integrated branch prediction): Ο μηχανισμός πρόβλεψης διακλάδωσης αποτελεί πλέον τμήμα της μονάδας ανάκτησης εντολών και προβαίνει αδιαλείπτως στην πρόβλεψη διακλαδώσεων, έτσι ώστε να καθοδηγεί τη διαδικασία ανάκτησης της διασωλήνωσης.

2. *Προανάκτηση εντολών* (instruction prefetch): Για την παράδοση πολλαπλών εντολών ανά κύκλο ρολογιού, η μονάδα ανάκτησης εντολών πιθανότατα χρειάζεται να προβαίνει στην εκ των προτέρων ανάκτηση. Η μονάδα διαχειρίζεται αυτόνομα την προανάκτηση των εντολών (δείτε στο Κεφάλαιο 5 την εξέταση των τεχνικών που το επιτυγχάνουν αυτό), ενσωματώνοντάς την στην πρόβλεψη διακλάδωσης.

3. *Προσπέλαση μνήμης εντολών και αποθήκευση σε απομουωτή* (instruction memory access and buffering): Κατά τη διαδικασία ανάκτησης πολλα-



Σχήμα 2.25 **Ακρίβεια πρόβλεψης του απομονωτή διευθύνσεων επιστροφής που λειτουργεί ως στοίβα με τη βοήθεια αρκετών μετροπρογραμμάτων του SPEC 95.** Η ακρίβεια υπολογίζεται με βάση το ποσοστό των διευθύνσεων επιστροφής που έχουν προβλεφθεί με ακρίβεια. Όταν ο απομονωτής περιέχει 0 καταχωρίσεις σημαίνει ότι χρησιμοποιείται η τυποποιημένη πρόβλεψη διακλάδωσης. Καθώς το βάθος των κλήσεων συνήθως δεν είναι ιδιαίτερα μεγάλο, εκτός από κάποιες συγκεκριμένες εξαιρέσεις, οι απομονωτές μεσαίου μεγέθους λειτουργούν σε ικανοποιητικό βαθμό. Τα δεδομένα αυτά προέρχονται από τους Skadron et al. (1999) και χρησιμοποιούν διορθωτικό μηχανισμό για την παρεμπόδιση της αλλοίωσης των αποθηκευμένων σε κρυφή μνήμη διευθύνσεων επιστροφής.

πλών εντολών ανά κύκλο ρολογιού συναντά κανείς πληθώρα επιπλοκών, συμπεριλαμβανομένου και του γεγονότος ότι η ανάκτηση πολλαπλών εντολών μπορεί να απαιτεί την προσπέλαση πολλαπλών γραμμών της κρυφής μνήμης. Η μονάδα ανάκτησης εντολών αντιμετωπίζει αυτές τις επιπλοκές, χρησιμοποιώντας την προανάκτηση ως προσπάθεια μείωσης του κόστους προσπέλασης διαφορετικών blocks της κρυφής μνήμης. Επίσης, η μονάδα ανάκτησης εντολών παρέχει τη δυνατότητα αποθήκευσης σε απομονωτή, ενεργώντας ουσιαστικά ως μονάδα παροχής εντολών κατ' απαίτηση στο στάδιο έκδοσης με βάση την ποσότητα που απαιτείται.

Καθώς οι σχεδιαστές προσπαθούν να αυξήσουν τον αριθμό των εντολών που εκτελούνται ανά κύκλο ρολογιού, η ανάκτηση εντολών καθίσταται ολοένα και πιο

σημαντικός ανασχετικός παράγοντας, γεγονός που καθιστά αδήριτη την ανάγκη αξιοποίησης νέων έξυπνων ιδεών για την παράδοση των εντολών στον απαραίτητο ρυθμό. Μία από τις νέες αυτές ιδέες, που έχει την ονομασία *κρυφές μνήμες ίχνους* (trace caches) και χρησιμοποιείται στον Pentium 4, εξετάζεται στο Παράρτημα Γ.

Εικασία: Ζήτημα Υλοποίησης και Προεκτάσεις τους

Στην ενότητα αυτή διερευνούμε τρία ζητήματα τα οποία σχετίζονται με την υλοποίηση της εικασίας, ξεκινώντας με τη χρήση της μετονομασίας των καταχωρητών, της προσέγγισης δηλαδή που έχει σχεδόν αντικαταστήσει εξ ολοκλήρου τη χρήση των απομονωτών αναδιάταξης. Στη συνέχεια εξετάζουμε μια σημαντική πιθανή προέκταση της εικασίας όσον αφορά τον έλεγχο ροής: μια ιδέα που είναι γνωστή ως πρόβλεψη τιμής.

Υποστήριξη Εικασίας: Σύγκριση Μετονομασίας Καταχωρητών και Απομονωτών Αναδιάταξης

Μία εναλλακτική λύση της χρήσης του απομονωτή αναδιάταξης (reorder buffer, ROB) είναι η ρητή χρησιμοποίηση ενός μεγαλύτερου φυσικού συνόλου καταχωρητών σε συνδυασμό με τη μετονομασία των καταχωρητών. Η προσέγγιση αυτή εδράζεται στην έννοια της μετονομασίας που χρησιμοποιήθηκε στον αλγόριθμο του Tomasulo, την οποία και προεκτείνει. Στον αλγόριθμο του Tomasulo, οι τιμές των *ορατών, σε επίπεδο αρχιτεκτονικής, καταχωρητών* (R0, . . . , R31 και F0, . . . , F31) περιλαμβάνονται, οποιαδήποτε χρονική στιγμή της εκτέλεσης, εντός κάποιου συνδυασμού του συνόλου των καταχωρητών και των σταθμών κράτησης. Με την προσθήκη της εικασίας, οι τιμές των καταχωρητών μπορεί, επίσης, να τοποθετούνται προσωρινά στον ROB. Σε κάθε περίπτωση, αν ο επεξεργαστής δεν εκδώσει νέες εντολές για κάποια χρονική περίοδο, όλες οι υφιστάμενες εντολές θα επικυρωθούν και οι τιμές των καταχωρητών θα εμφανίζονται στη συστοιχία των καταχωρητών, η οποία αντιστοιχεί στους ορατούς, σε επίπεδο αρχιτεκτονικής, καταχωρητές.

Στην προσέγγιση της μετονομασίας των καταχωρητών, χρησιμοποιείται ένα εκτεταμένο σύνολο φυσικών καταχωρητών για τη διατήρηση τόσο των ορατών, σε επίπεδο αρχιτεκτονικής, καταχωρητών όσο και των προσωρινών τιμών. Επομένως, το εκτεταμένο σύνολο καταχωρητών αντικαθιστά τη λειτουργία τόσο του ROB όσο και των σταθμών κράτησης. Κατά τη διάρκεια της έκδοσης εντολών, η διαδικασία μετονομασίας αντιστοιχίζει τα ονόματα των καταχωρητών της αρχιτεκτονικής σε αριθμούς φυσικών καταχωρητών του εκτεταμένου συνόλου καταχωρητών, αναθέτοντας ένα νέο μη χρησιμοποιημένο καταχωρητή ως προορισμό. Οι κίνδυνοι WAW και WAR μπορούν να αποφευχθούν μέσω της μετονομασίας του καταχωρητή προορισμού, ενώ ο χειρισμός της ανάκαμψης μετά από εσφαλμένη

εικασία είναι εφικτός διότι ο φυσικός καταχωρητής που διατηρεί τον προορισμό κάποιας εντολής δεν καθίσταται καταχωρητής αρχιτεκτονικής πριν την επικύρωση της εντολής αυτής. Ο χάρτης μετονομασίας (renaming map) αποτελεί απλή δομή δεδομένων η οποία παρέχει τον αριθμό του φυσικού καταχωρητή ο οποίος την τρέχουσα χρονική στιγμή αντιστοιχεί στο συγκεκριμένο καταχωρητή αρχιτεκτονικής. Η δομή αυτή είναι παρόμοια όσον αφορά τη συγκρότηση και τη λειτουργία της με τον πίνακα κατάστασης καταχωρητών του αλγορίθμου του Tomasulo. Όταν επικυρώνεται κάποια εντολή, ο πίνακας μετονομασίας ενημερώνεται οριστικά έτσι ώστε να καταδεικνύεται ότι ο φυσικός καταχωρητής αντιστοιχεί στον πραγματικό καταχωρητή αρχιτεκτονικής, γεγονός που συνεπώς οδηγεί στην ολοκλήρωση της ενημέρωσης της κατάστασης του επεξεργαστή.

Ένα από τα πλεονεκτήματα της προσέγγισης της μετονομασίας σε σύγκριση με την προσέγγιση του ROB είναι το γεγονός ότι απλοποιείται η επικύρωση των εντολών, καθώς απαιτεί μόνο δύο απλές ενέργειες: την καταγραφή του ότι η αντιστοιχία ανάμεσα στον αριθμό του καταχωρητή αρχιτεκτονικής και στον αριθμό του φυσικού καταχωρητή έχει πάψει να είναι υποθετική και την αποδέσμευση όλων των φυσικών καταχωρητών που χρησιμοποιούνταν για τη διατήρηση της «παλαιότερης» τιμής του καταχωρητή αρχιτεκτονικής. Στις σχεδιάσεις που διαθέτουν σταθμούς κράτησης, οι σταθμοί αποδεσμεύονται όταν η εντολή που τους χρησιμοποιεί ολοκληρώνει την εκτέλεσή της, ενώ η καταχώριση του ROB ελευθερώνεται όταν επικυρώνεται η αντίστοιχη εντολή.

Στη μετονομασία των καταχωρητών, η αποδέσμευση των καταχωρητών είναι πιο πολύπλοκη, διότι προτού ελευθερώσουμε κάποιο φυσικό καταχωρητή πρέπει να γνωρίζουμε ότι αυτός έχει πάψει να αντιστοιχεί σε κάποιον καταχωρητή αρχιτεκτονικής, καθώς και ότι δεν υφίσταται καμία εκκρεμής χρήση του φυσικού καταχωρητή. Ένας φυσικός καταχωρητής αντιστοιχεί σε κάποιον καταχωρητή αρχιτεκτονικής μέχρι την επανεγγραφή του καταχωρητή αρχιτεκτονικής, γεγονός που οδηγεί τον πίνακα μετονομασίας να δείχνει προς άλλη κατεύθυνση. Δηλαδή, αν δεν υπάρχει καμία καταχώριση μετονομασίας η οποία να καταδεικνύει κάποιον συγκεκριμένο φυσικό καταχωρητή, τότε έχει πάψει να αντιστοιχεί σε κάποιον καταχωρητή αρχιτεκτονικής. Ωστόσο, μπορεί να εξακολουθούν να υφίστανται εκκρεμείς χρήσεις του φυσικού καταχωρητή. Ο επεξεργαστής μπορεί να προσδιορίσει αν αυτό ισχύει, εξετάζοντας τους προσδιοριστές των καταχωρητών αφετηρίας όλων των εντολών που βρίσκονται στις ουρές των λειτουργικών μονάδων. Αν κάποιος δεδομένος φυσικός καταχωρητής δεν εμφανίζεται ως αφετηρία και δεν αντιστοιχεί σε κάποιον καταχωρητή αρχιτεκτονικής, τότε μπορεί να χρησιμοποιηθεί και να ανατεθεί εκ νέου.

Εναλλακτικά, ο επεξεργαστής μπορεί απλά να αναμένει έως ότου να επικυρωθεί κάποια άλλη εντολή η οποία εγγράφει τον ίδιο καταχωρητή αρχιτεκτονικής. Στο σημείο αυτό, δεν μπορούν να υφίστανται επιπλέον εκκρεμείς χρήσεις της παλαιότερης τιμής. Η μέθοδος αυτή, παρόλο που μπορεί να δεσμεύει κάποιον φυσικό καταχωρητή για ελαφρά μεγαλύτερο χρονικό διάστημα από αυτό

που είναι απαραίτητο, μπορεί να υλοποιηθεί εύκολα και ως εκ τούτου χρησιμοποιείται σε αρκετούς σύγχρονους υπερβαθμωτούς επεξεργαστές.

Μία από τις ερωτήσεις που μπορεί να θέσει κανείς είναι η εξής: Πώς μπορούμε να γνωρίζουμε ποιοι είναι οι καταχωρητές αρχιτεκτονικής αν αυτοί αλλάζουν συνεχώς; Τις περισσότερες φορές όταν εκτελείται το πρόγραμμα αυτό δεν έχει καμία σημασία. Παρ' όλα αυτά, υπάρχουν προφανώς περιπτώσεις στις οποίες κάποια άλλη διεργασία, όπως για παράδειγμα το λειτουργικό σύστημα, πρέπει να είναι σε θέση να γνωρίζει επακριβώς που βρίσκονται τα περιεχόμενα κάποιου συγκεκριμένου καταχωρητή αρχιτεκτονικής. Για να κατανοήσουμε τον τρόπο με τον οποίο παρέχεται η δυνατότητα αυτή, ας υποθέσουμε ότι ο επεξεργαστής δεν εκδίδει εντολές για κάποια χρονική περίοδο. Εν τέλει, όλες οι εντολές διασωλήνωσης θα επικυρωθούν και η αντιστοιχία ανάμεσα στους ορατούς, σε επίπεδο αρχιτεκτονικής καταχωρητές και στους φυσικούς καταχωρητές θα παραμείνει σταθερή. Στο σημείο αυτό, υποσύνολο των φυσικών καταχωρητών περιέχει τους ορατούς, σε επίπεδο αρχιτεκτονικής, καταχωρητές, ενώ η τιμή οποιουδήποτε φυσικού καταχωρητή δεν είναι συσχετισμένος με κάποιον καταχωρητή αρχιτεκτονικής είναι πλέον περιττή. Στη συνέχεια είναι εύκολη η μεταφορά των καταχωρητών αρχιτεκτονικής σε κάποιο σταθερό υποσύνολο φυσικών καταχωρητών, έτσι ώστε να μπορεί να πραγματοποιηθεί η μεταβίβαση των τιμών σε κάποια άλλη διεργασία.

Τα τελευταία έτη οι περισσότεροι υπερβαθμωτοί επεξεργαστές υψηλού επιπέδου, συμπεριλαμβανομένων αυτών της σειράς Pentium, του MIPS R12000 και των επεξεργαστών Power και PowerPC, έχουν επιλέξει να χρησιμοποιήσουν τη μετονομασία των καταχωρητών, προσθέτοντας για το λόγο αυτό από 20 έως 80 επιπλέον καταχωρητές. Αφού ένας νέος εικονικός καταχωρητής ανατίθεται σε όλα τα αποτελέσματα, μέχρι την επικύρωσή τους, οι επιπλέον αυτοί καταχωρητές αντικαθιστούν μία από τις βασικές λειτουργίες του ROB και προσδιορίζουν σε μεγάλο βαθμό τον τρόπο με τον οποίο μπορεί να πραγματοποιείται η ταυτόχρονη εκτέλεση (ανάμεσα στην έκδοση και την επικύρωση) πολλών εντολών.

Επιτρεπτός Βαθμός Εφαρμογής της Εικασίας

Ένα από τα πιο σημαντικά πλεονεκτήματα της εικασίας είναι η ικανότητά της να αποκαλύπτει γεγονότα τα οποία σε διαφορετική περίπτωση θα έθεταν από νωρίς τη διασωλήνωση σε αδράνεια, γεγονότα όπως οι αστοχίες της κρυφής μνήμης. Ωστόσο, το ενδεχόμενο αυτό πλεονέκτημα συνοδεύεται από ένα σημαντικό μειονέκτημα. Η εικασία δεν είναι ανέξοδη: απαιτεί χρόνο και ενέργεια, ενώ η ανάκαμψη μετά από εσφαλμένη εικασία μειώνει περισσότερο την απόδοση. Επιπλέον, για την υποστήριξη του υψηλότερου ρυθμού εκτέλεσης εντολών που απαιτείται για την αξιοποίηση των ωφελειών που προκύπτουν από την εικασία, ο επεξεργαστής πρέπει να διαθέτει επιπλέον πόρους, οι οποίοι απαιτούν επι-

πλέον ποσότητες πυριτίου και καταναλώνουν ενέργεια. Τέλος, σε περίπτωση που η εικασία προκαλέσει την εμφάνιση κάποιου γεγονότος εξαίρεσης, όπως η αστοχία της κρυφής μνήμης ή του TLB, αυξάνεται η πιθανότητα να προκύψει σημαντική μείωση της απόδοσης σε σύγκριση με την περίπτωση κατά την οποία θα προέκυπτε το ίδιο γεγονός χωρίς να υποστηρίζεται η εικασία.

Για τη διατήρηση των περισσότερων πλεονεκτημάτων και την ταυτόχρονη ελαχιστοποίηση των μειονεκτημάτων, οι περισσότερες διασωληνώσεις που υποστηρίζουν την εικασία επιτρέπουν την υποθετική αντιμετώπιση μόνο των γεγονότων εξαίρεσης χαμηλού κόστους (όπως η αστοχία της κρυφής μνήμης πρώτου επιπέδου). Σε περίπτωση που προκύψει κάποιο δαπανηρό γεγονός εξαίρεσης, όπως η αστοχία της κρυφής μνήμης δεύτερου επιπέδου ή η αστοχία του παράπλευρου απομονωτή μετάφρασης (translation lookaside buffer, TLB), ο επεξεργαστής, πριν χειριστεί το γεγονός, θα αναμένει έως ότου πάψει η εντολή που προκάλεσε το γεγονός να είναι υποθετική. Η παραπάνω προσέγγιση, παρόλο που ενδέχεται να υποβαθμίσει ελαφρά την απόδοση μερικών προγραμμάτων, αποφεύγει την εμφάνιση σημαντικών απωλειών σε επίπεδο απόδοσης άλλων εφαρμογών, ιδίως αυτών που εμφανίζουν υψηλή συχνότητα τέτοιων γεγονότων σε συνδυασμό με *πρόβλεψη διακλάδωσης η οποία δεν προσεγγίζει το απόλυτο.*

Στη δεκαετία του 1990, τα ενδεχόμενα μειονεκτήματα της εικασίας ήταν λιγότερο προφανή. Με την εξέλιξη των επεξεργαστών, άρχισε να γίνεται εμφανές ολοένα και περισσότερο τόσο το πραγματικό κόστος της εικασίας όσο και οι περιορισμοί του συνδυασμού της εικασίας με την ευρύτερη έκδοση εντολών. Θα επιστρέψουμε σε αυτό το ζήτημα σε επόμενο κεφάλαιο.

Εικασία μέσω Πολλαπλών Διακλαδώσεων

Στα παραδείγματα που έχουμε εξετάσει στο κεφάλαιο αυτό, ήταν εφικτή η επίλυση μιας διακλάδωσης προτού χρειαστεί να εφαρμόσουμε εικασία σε κάποια άλλη. Σε τρεις διαφορετικές περιπτώσεις μπορεί να υπάρξουν οφέλη από την ταυτόχρονη εφαρμογή της εικασίας σε πολλαπλές διακλαδώσεις: στην πολύ υψηλή συχνότητα διακλαδώσεων, στη σημαντική συσταδοποίηση των διακλαδώσεων και στις μεγάλες καθυστερήσεις στις λειτουργικές μονάδες. Στις δύο πρώτες περιπτώσεις, η επίτευξη υψηλής απόδοσης μπορεί να σημαίνει ότι υποστηρίζεται η εικασία σε πολλαπλές διακλαδώσεις, κάτι που με τη σειρά του συνεπάγεται ότι μπορεί να πραγματοποιείται ο χειρισμός περισσότερων από μία διακλαδώσεων ανά κύκλο ρολογιού. Τα προγράμματα βάσεων δεδομένων, καθώς και άλλοι λιγότερο δομημένοι ακέραιοι υπολογισμοί, συχνά διαθέτουν τέτοιες ιδιότητες, γεγονός που καθιστά σημαντική την εφαρμογή της εικασίας σε πολλαπλές διακλαδώσεις. Παρόμοια, οι μεγάλες καθυστερήσεις στις λειτουργικές μονάδες μπορούν να καταστήσουν σαφή τη σπουδαιότητα της εφαρμογής της εικασίας *σε πολλαπλές διακλαδώσεις ως ένα μέσο αποφυγής εισαγωγής διαστημάτων α-*

δράνειας εξαιτίας των μεγαλύτερων καθυστερήσεων της διασωλήνωσης.

Η εφαρμογή της εικασίας σε πολλαπλές διακλαδώσεις περιπλέκει ελαφρά τη διαδικασία ανάκαμψης μετά από εσφαλμένη εικασία, η οποία ωστόσο είναι αρκετά σαφής. Μία πιο πολύπλοκη τεχνική σχετίζεται με την πρόβλεψη και την εφαρμογή εικασίας σε περισσότερες από μία διακλαδώσεις ανά κύκλο ρολογιού. Ο IBM Power2 μπορούσε να επιλύσει δύο διακλαδώσεις ανά κύκλο, ωστόσο δεν εφάρμοζε εικασία σε καμία άλλη εντολή. Έως το 2005, κανένας επεξεργαστής δεν συνδύαζε την πλήρη εφαρμογή της εικασίας με την επίλυση πολλαπλών διακλαδώσεων ανά κύκλο.

Πρόβλεψη Τιμής

Μία τεχνική αύξησης του διαθέσιμου εύρους του ILP ενός προγράμματος είναι η πρόβλεψη τιμής. Η *πρόβλεψη τιμής* (value prediction) επιχειρεί να προβλέψει την τιμή που πρόκειται να παραχθεί από κάποια εντολή. Όπως είναι φανερό, καθώς οι περισσότερες εντολές παράγουν διαφορετική τιμή κάθε φορά που εκτελούνται (ή τουλάχιστον διαφορετική τιμή από κάποιο σύνολο τιμών), η πρόβλεψη τιμής μπορεί να παρουσιάσει περιορισμένη μόνο επιτυχία. Παρ' όλα αυτά, υπάρχουν συγκεκριμένες εντολές για τις οποίες είναι ευκολότερη η πρόβλεψη της τιμής του αποτελέσματος - όπως, για παράδειγμα, οι εντολές φόρτωσης που είτε φορτώνουν τιμές από κάποιο σταθερό σύνολο είτε φορτώνουν μια τιμή η οποία σπάνια αλλάζει. Επιπλέον, όταν κάποια εντολή παράγει τιμή η οποία επιλέγεται από κάποιο μικρό σύνολο ενδεχόμενων τιμών, τότε μπορεί να είναι εφικτή η απευθείας πρόβλεψη της τιμής.

Η πρόβλεψη τιμής είναι χρήσιμη αν αυξάνει σημαντικά το εύρος του διαθέσιμου ILP. Το ενδεχόμενο αυτό είναι πιο πιθανό όταν η τιμή χρησιμοποιείται ως αφετηρία μιας αλυσίδας εξαρτώμενων υπολογισμών, όπως είναι η φόρτωση. Η ακρίβεια της πρόβλεψης είναι ιδιαίτερα σημαντική, επειδή η πρόβλεψη τιμής χρησιμοποιείται για τη βελτίωση της εικασίας και η εσφαλμένη εικασία έχει αρνητικές συνέπειες όσον αφορά την απόδοση.

Το μεγαλύτερο μέρος των ερευνών που σχετίζονται με την πρόβλεψη τιμής αφορά τις εντολές φόρτωσης. Μπορούμε να αξιολογήσουμε τη μέγιστη ακρίβεια των μηχανισμών πρόβλεψης τιμής για τις εντολές φόρτωσης εξετάζοντας τη συχνότητα με την οποία οι εντολές φόρτωσης επιστρέφουν κάποια τιμή, η οποία είναι ίδια με την τιμή που επεστράφη σε κάποια *πρόσφατη* εκτέλεσή τους. Το πιο απλό που μπορεί να εξετάσει κανείς αφορά την περίπτωση κατά την οποία η φόρτωση επιστρέφει τιμή που είναι ίδια με αυτήν της τελευταίας εκτέλεσής της. Σε αρκετά μειροπρογράμματα του SPEC CPU2000, το φαινόμενο αυτό παρουσιάζεται στο 5% έως και το 80% των περιπτώσεων. Αν επιτρέψουμε τη σύγκριση της φόρτωσης με τις 16 πιο πρόσφατες τιμές που έχουν επιστραφεί, τότε αυξάνεται η πιθανότητα εύρεσης ίδιας επιστρεφόμενης τιμής, κάτι που επι-

βεβαιώνεται από πολλά μετροπρογράμματα που παρουσιάζουν ρυθμό επιτυχίας της τάξης του 80%. Βεβαίως, η εύρεση μίας ίδιας τιμής μεταξύ των 16 πιο πρόσφατων τιμών δεν μπορεί να αποτελέσει κριτήριο σε ό,τι αφορά την επιλογή της προβλεπόμενης τιμής, ωστόσο καταδεικνύει ότι, ακόμα και αν χρησιμοποιηθούν επιπρόσθετες πληροφορίες, η ακρίβεια της πρόβλεψης είναι αδύνατο να υπερβεί το 80%.

Εξαιτίας του υψηλού κόστους των εσφαλμένων προβλέψεων, καθώς του ενδεχομένου ο ρυθμός των εσφαλμένων προβλέψεων να είναι υψηλός (20% - 50%), οι ερευνητές έχουν επικεντρωθεί στο να αξιολογήσουν ποιες εντολές φόρτωσης μπορούν να προβλεφθούν σε ικανοποιητικό βαθμό, έτσι ώστε να επιχειρούν την πρόβλεψη μόνο αυτών των εντολών. Αυτό έχει ως αποτέλεσμα την ύπαρξη χαμηλότερου ρυθμού εσφαλμένων προβλέψεων, έχοντας ωστόσο λιγότερες υποψήφιες εντολές μέσω των οποίων να μπορεί να επιταχυνθεί η πρόβλεψη. Αν επιχειρήσουμε να προβλέψουμε μόνο τις εντολές φόρτωσης που επιστρέφουν πάντα την ίδια τιμή, ενδέχεται να είναι εφικτή η πρόβλεψη μόνο του 10% έως 15% των εντολών φόρτωσης. Η έρευνα γύρω από το ζήτημα της πρόβλεψης τιμής συνεχίζεται. Ωστόσο, τα αποτελέσματα έως σήμερα δεν είναι επαρκώς πειστικά ώστε να συνηγορούν ότι υπάρχει κάποιος εμπορικός επεξεργαστής ο οποίος έχει ενσωματώσει την τεχνική αυτή.

Μια απλή ιδέα που έχει υιοθετηθεί και σχετίζεται με την πρόβλεψη τιμής είναι η πρόβλεψη ψευδωνυμίας διευθύνσεων. Η *πρόβλεψη ψευδωνυμίας διευθύνσεων* (address aliasing prediction) αποτελεί απλή τεχνική η οποία προβλέπει αν είτε δύο εντολές αποθήκευσης είτε μία εντολή φόρτωσης και μία εντολή αποθήκευσης διαθέτουν αναφορά στην ίδια διεύθυνση μνήμης. Αν οι δύο εντολές δεν διαθέτουν αναφορά στην ίδια διεύθυνση, τότε η εναλλαγή τους μπορεί να πραγματοποιηθεί με ασφάλεια. Διαφορετικά, πρέπει να περιμένουμε έως ότου να γίνουν γνωστές οι διευθύνσεις της μνήμης τις οποίες προσπελάζουν οι εντολές. Επειδή στην πραγματικότητα δεν χρειάζεται να προβλέψουμε τις τιμές των διευθύνσεων, παρά μόνο να διαπιστώσουμε αν οι τιμές αυτές έρχονται σε σύγκρουση μεταξύ τους, η διαδικασία πρόβλεψης είναι τόσο πιο απλή όσο και πιο σταθερή. Κατά συνέπεια, η περιορισμένη αυτή μορφή εικασίας της τιμής των διευθύνσεων έχει χρησιμοποιηθεί σε λίγους επεξεργαστές.

2.10 Συνδυασμός Όλων των Δεδομένων: Ο Intel Pentium 4

Ο Pentium 4 είναι επεξεργαστής που διαθέτει διασωλήνωση με μεγάλο βάθος η οποία υποστηρίζει την πολλαπλή έκδοση και την εικασία. Στην ενότητα αυτή, περιγράφουμε τα βασικά χαρακτηριστικά της μικροαρχιτεκτονικής του Pentium 4 και εξετάζουμε την απόδοσή του με τη βοήθεια των μετροπρογραμμάτων SPEC CPU. Ο Pentium 4 υποστηρίζει, επίσης, την πολυνημάτωση, ζήτημα το οποίο θα

εξετάσουμε στο επόμενο κεφάλαιο.

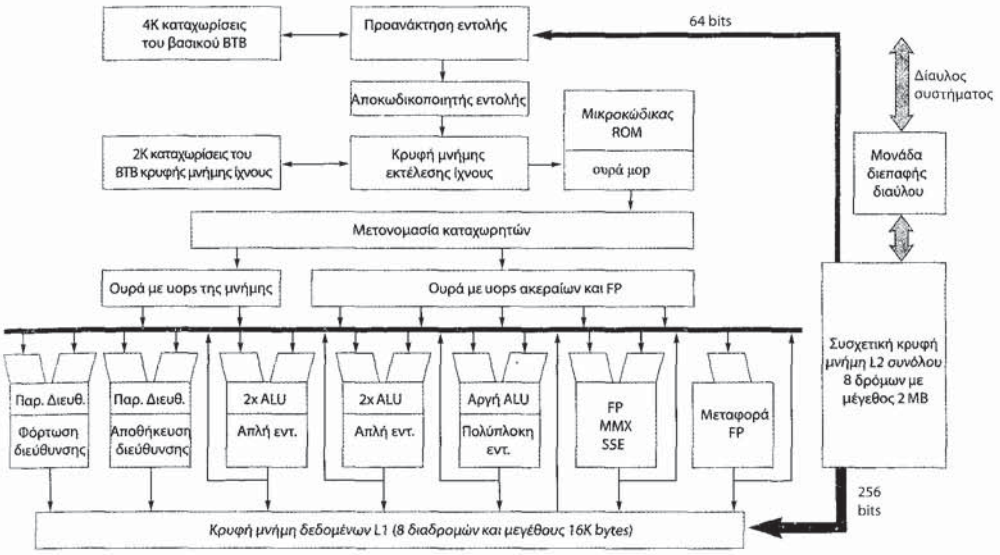
Ο Pentium 4 χρησιμοποιεί υποθετική εντατική μικροαρχιτεκτονική εκτός σειράς, η οποία ονομάζεται Netburst, που υποστηρίζει πλήρως τη διασωλήνωση, έχοντας ως στόχο την επίτευξη υψηλής ρυθμοαπόδοσης εντολών, συνδυάζοντας την πολλαπλή έκδοση με υψηλό ρυθμό ρολογιού. Όπως στη μικροαρχιτεκτονική που χρησιμοποιείται στον Pentium III, έτσι και σε αυτήν την περίπτωση χρησιμοποιείται ένας αποκωδικοποιητής ο οποίος μεταφράζει κάθε εντολή της IA-32 σε σειρά μικρολειτουργιών (micro-operations, uops), οι οποίες είναι παρόμοιες με τις συνήθεις εντολές της RISC. Οι uops στη συνέχεια εκτελούνται με τη βοήθεια της δυναμικά χρονοδρομολογημένης υποθετικής διασωλήνωσης.

Ο Pentium 4 χρησιμοποιεί μία νέα *κρυφή μνήμη ίχνους εκτέλεσης* (execution trace cache) για την παραγωγή της ροής των εντολών uop, σε αντίθεση με τη συμβατική κρυφή μνήμη εντολών η οποία θα διατηρούσε τις εντολές της IA-32. Η *κρυφή μνήμη ίχνους* (trace cache) αποτελεί τύπο κρυφής μνήμης εντολών που διατηρεί ακολουθίες εντολών που πρόκειται να εκτελεστούν, οι οποίες περιλαμβάνουν μη γειτονικές εντολές διαχωρισμένες μέσω διακλαδώσεων. Η κρυφή μνήμη ίχνους προσπαθεί να αξιοποιήσει τη χρονική ακολουθία της εκτέλεσης των εντολών και όχι τη χωρική τοπικότητα την οποία αξιοποιούν οι συμβατικές κρυφές μνήμες. Η κρυφή μνήμη ίχνους εξετάζεται λεπτομερώς στο Παράρτημα Γ.

Η κρυφή μνήμη ίχνους εκτέλεσης του Pentium 4 είναι κρυφή μνήμη που περιλαμβάνει uops, οι οποίες αντιστοιχούν στην αποκωδικοποιημένη ροή εντολών της IA-32. Ο Pentium 4, τροφοδοτώντας τη διασωλήνωση με εντολές από την κρυφή μνήμη ίχνους εκτέλεσης, αποφεύγει την ανάγκη της εκ νέου αποκωδικοποίησης των εντολών της IA-32, οποτεδήποτε σημειώνεται ευστοχία της κρυφής μνήμης ίχνους. Οι εντολές της IA-32 ανακτώνται από την κρυφή μνήμη L2 και αποκωδικοποιούνται έτσι ώστε να τροφοδοτήσουν την κρυφή μνήμη ίχνους εκτέλεσης, μόνο στην περίπτωση που σημειώνεται αστοχία σε αυτήν. Σε κάθε κύκλο μπορεί να αποκωδικοποιηθούν και να μεταφραστούν έως και τρεις εντολές της IA-32, κάτι που οδηγεί στην παραγωγή έως και έξι uops. Όταν κάποια εντολή της IA-32 απαιτεί περισσότερες από τρεις uops, τότε η ακολουθία των uops παράγεται από τη ROM μικροκώδικα.

Η κρυφή μνήμη ίχνους εκτέλεσης διαθέτει το δικό της απομονωτή προορισμού διακλάδωσης, ο οποίος προβλέπει το αποτέλεσμα δύο διακλαδώσεων uop. Ο υψηλός ρυθμός ευστοχίας της κρυφής μνήμης ίχνους εκτέλεσης (για παράδειγμα, ο ρυθμός αστοχίας της κρυφής μνήμης ίχνους, στα μειροπρογράμματα του SPEC CPUINT2000, είναι μικρότερος από 0,15%) συνεπάγεται ότι η ανάκτηση και η αποκωδικοποίηση των εντολών της IA-32 είναι κάτι που σπάνια χρειάζεται να πραγματοποιηθεί.

Οι uops, μετά την ανάκτησή τους από την κρυφή μνήμη ίχνους εκτέλεσης, εκτελούνται μέσω μιας υποθετικής διασωλήνωσης εκτός σειράς, παρόμοια με αυτήν της Ενότητας 2.6, χρησιμοποιώντας, ωστόσο, την προσέγγιση της μετο-



Σχήμα 2.26 Η μικροαρχιτεκτονική του Pentium 4. Τα μεγέθη των κρυφών μνημών αντιστοιχούν σε αυτά του Pentium 4 640. Αξίζει να σημειωθεί ότι οι εντολές συνήθως προέρχονται από την κρυφή μνήμη ίχνους. Μόνο όταν σημειώνεται αστοχία της κρυφής μνήμης ίχνους χρησιμοποιείται η βασική μονάδα προανάκτησης εντολών. Το σχήμα προέρχεται από τους Boggs et al. [2004].

νομασίας καταχωρητών και όχι αυτήν του απομονωτή αναδιάταξης. Σε κάθε κύκλο ρολογιού μπορούν να μετονομάζονται και να διαβιβάζονται έως και τρεις uops στις ουρές των λειτουργικών μονάδων, ενώ ο ίδιος αριθμός uops μπορεί να επικυρώνεται σε κάθε κύκλο ρολογιού. Υπάρχουν τέσσερις θύρες διαβίβασης (dispatch ports), οι οποίες, σε κάθε κύκλο ρολογιού, επιτρέπουν τη διαβίβαση συνολικά έξι uops στις λειτουργικές μονάδες. Η μονάδα φόρτωσης και η μονάδα αποθήκευσης διαθέτουν καθεμία τη δική της θύρα διαβίβασης, οι υπόλοιπες πράξεις της ALU καλύπτονται από κάποια άλλη θύρα, ενώ η τέταρτη θύρα χειρίζεται τις λειτουργίες κινητής υποδιαστολής και ακεραίων. Το Σχήμα 2.26 παρουσιάζει σε διάγραμμα αυτήν τη μικροαρχιτεκτονική.

Καθώς η μικροαρχιτεκτονική του Pentium 4 υποστηρίζει δυναμική χρονοδρομολόγηση, οι uops δεν ακολουθούν κατά την εκτέλεσή τους το απλό στατικό σύνολο των σταδίων της διασωλήνωσης. Αντιθέτως, διάφορα στάδια εκτέλεσης (ανάκτηση εντολών, αποκωδικοποίηση, έκδοση uop, μετονομασία, χρονοδρομολόγηση, εκτέλεση και απόσυρση) μπορούν να καταναλώσουν μεταβαλλόμενο αριθμό κύκλων ρολογιού. Στον Pentium III ο ελάχιστος χρόνος που έπρεπε να μεσολαβήσει ώστε μια εντολή να μεταφερθεί από το στάδιο ανάκτησης σε αυτό της απόσυρσης (retire) ήταν ίσος με 11 κύκλους, ενώ οι εντολές που χρειάζονταν πολλαπλούς κύκλους κατά το στάδιο εκτέλεσης κατανάλωναν περισσότερους κύ-

κλους. Όπως σε κάθε διασωλήνωση δυναμικής χρονοδρομολόγησης, έτσι και σε αυτήν την περίπτωση οι εντολές μπορεί να χρειαστούν πολύ περισσότερο χρόνο, αν πρέπει να αναμένουν τα ορίσματα. Όπως έχει ήδη περιγραφεί, ο Pentium 4 χρησιμοποιεί διασωλήνωση πολύ μεγαλύτερου βάθους, κατατέμνοντας τα στάδια της διασωλήνωσης του Pentium III σε επιμέρους τμήματα, με απώτερο στόχο την επίτευξη υψηλότερου ρυθμού ρολογιού. Στην αρχική έκδοση του Pentium 4, που κυκλοφόρησε το 2000, ο ελάχιστος αριθμός των κύκλων που απαιτούνταν για τη διάσχιση της διασωλήνωσης είχε αυξηθεί σε 21 κύκλους, γεγονός που επέτρεπε ρυθμό ρολογιού της τάξης του 1.5 GHz. Το 2004, η Intel παρήγαγε έκδοση του Pentium 4 με ρυθμό ρολογιού της τάξης των 3.2 GHz. Για την επίτευξη του υψηλού αυτού ρυθμού, το βάθος της διασωλήνωσης αυξήθηκε σε τέτοιο βαθμό, ώστε μία απλή εντολή χρειαζόταν 31 κύκλους ρολογιού προκειμένου να μεταφερθεί από το στάδιο της ανάκτησης σε αυτό της απόσυρσης. Αυτή η αύξηση του βάθους της διασωλήνωσης, σε συνδυασμό με βελτιώσεις της ταχύτητας του transistor, κατέστησαν εφικτό το διπλασιασμό του ρυθμού του ρολογιού σε σύγκριση με την πρώτη έκδοση του Pentium 4.

Όπως είναι φανερό, έχοντας διασωληνώσεις με τόσο μεγάλο βάθος, καθώς και τόσο εντατικούς ρυθμούς ρολογιού, το κόστος τόσο των αστοχιών της κρυφής μνήμης και όσο των εσφαλμένων προβλέψεων των διακλαδώσεων είναι ιδιαίτερα υψηλό. Για τη μείωση της συχνότητας προσπέλασης της DRAM χρησιμοποιείται κρυφή μνήμη δύο επιπέδων. Η πρόβλεψη διακλάδωσης πραγματοποιείται με τη βοήθεια ενός απομονωτή προορισμού διακλάδωσης, που χρησιμοποιεί μηχανισμό πρόβλεψης δύο επιπέδων, ο οποίος διαθέτει τόσο τοπικό όσο και καθολικό ιστορικό διακλαδώσεων. Στον πιο πρόσφατο Pentium 4, έχει αυξηθεί το μέγεθος του απομονωτή προορισμού διακλάδωσης, ενώ, επίσης, έχει βελτιωθεί ο στατικός μηχανισμός πρόβλεψης που χρησιμοποιείται όταν σημειώνεται αστοχία του απομονωτή προορισμού διακλάδωσης. Στο Σχήμα 2.27 συνοψίζονται τα βασικά χαρακτηριστικά της μικροαρχιτεκτονικής, ενώ στον επεξηγηματικό τίτλο του σχήματος γίνεται αναφορά σε μερικές από τις αλλαγές που έχουν πραγματοποιηθεί από την πρώτη έκδοση του Pentium 4 το 2000.

Ανάλυση της Απόδοσης του Pentium 4

Η διασωλήνωση μεγάλου βάθους του Pentium 4 καθιστά ιδιαίτερα σημαντική τη χρήση της εικασίας, καθώς και της εξάρτησής της από την πρόβλεψη διακλάδωσης, έτσι ώστε να είναι εφικτή η επίτευξη υψηλής απόδοσης. Παρόμοια, η απόδοση εξαρτάται σε μεγάλο βαθμό από το σύστημα μνήμης. Παρόλο που τόσο η δυναμική χρονοδρομολόγηση όσο και ο μεγάλος αριθμός εκκρεμών εντολών φόρτωσης και αποθήκευσης οδηγούν στην απόκρυψη του λανθάνοντος χρόνου των αστοχιών της κρυφής μνήμης, ο εντατικός ρυθμός ρολογιού που ισούται με 3.2 GHz συνεπάγεται ότι οι αστοχίες της κρυφής μνήμης L2 είναι πιθανό να

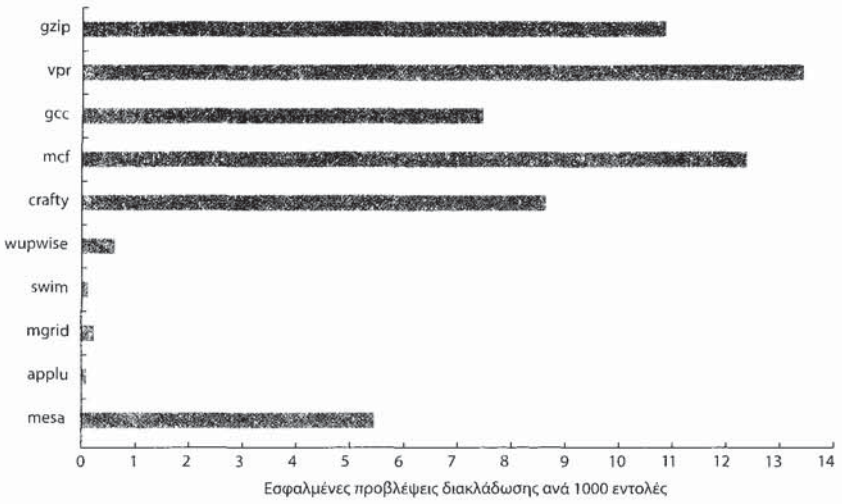
Χαρακτηριστικό	Μέγεθος	Σχόλια
Βασικός απομονωτής προορισμού διακλάδωσης	4K καταχωρίσεις	Προβλέπει την επόμενη εντολή IA-32 που πρόκειται να ανακτηθεί. Χρησιμοποιείται μόνο όταν σημειώνεται αστοχία στην κρυφή μνήμη ίχνους.
Κρυφή μνήμη ίχνους εκτέλεσης	12K uops	Η κρυφή μνήμη ίχνους χρησιμοποιείται για τις uops
Απομονωτής προορισμού διακλάδωσης κρυφής μνήμης ίχνους	2K entries	Προβλέπει την επόμενη uop
Καταχωρητές για μετονομασία	Συνολικά 128	Μπορούν να εκτελούνται 128 uops με έως και 48 λειτουργίες φόρτωσης και 32 αποθήκευσης
Λειτουργικές μονάδες	Συνολικά 7: 2 απλές ALU, μία πολύπλοκη ALU, μία φόρτωσης, μία αποθήκευσης, μία μεταφοράς FP και μία αριθμητική FP.	Οι απλές μονάδες ALU λειτουργούν σε διπλάσιο ρυθμό ρολογιού, αποδεχόμενες έως δύο απλές uops ALU σε κάθε κύκλο ρολογιού. Αυτό επιτρέπει την εκτέλεση δύο εξαρτώμενων λειτουργιών ALU εντός ενός κύκλου ρολογιού.
Κρυφή μνήμη δεδομένων L1	16 KB. Συσχετική 8 δρόμων. Blocks των 64 bytes. Ταυτόχρονη εγγραφή	Ο λανθάνων χρόνος της φόρτωσης ακεραίων ισούται με 4 κύκλους. Ο λανθάνων χρόνος της φόρτωσης FP ισούται με 12 κύκλους. Έως και 8 εκκρεμείς αστοχίες φόρτωσης
Κρυφή μνήμη L2.	2 MB. Συσχετική 8 δρόμων. Blocks των 128 bytes. Ετερόχρονη εγγραφή	256 bits προς την L1, που παρέχουν 108 GB/sec. Χρόνος προσπέλασης ίσος με 18 κύκλους. 64 bits προς τη μνήμη που παρέχουν 6.4 GB/sec. Οι αστοχίες της L2 δεν προκαλούν την αυτόματη ενημέρωση της L1.

Σχήμα 2.27 Σημαντικά χαρακτηριστικά της πρόσφατης υλοποίησης του Pentium 4, 640 (με την κωδική ονομασία Prescott), χρησιμοποιώντας τεχνολογία 90 nm. Ο νεότερος Pentium 4 χρησιμοποιεί μεγαλύτερες κρυφές μνήμες και απομονωτές πρόβλεψης διακλάδωσης, επιτρέπει την ύπαρξη περισσότερων εκκρεμών εντολών φόρτωσης και αποθήκευσης και διαθέτει υψηλότερο εύρος ζώνης μεταξύ των επιπέδων του συστήματος μνήμης. Αυτό που χρήζει ιδιαίτερης προσοχής είναι η νέα χρήση των ALUs διπλής ταχύτητας, οι οποίες επιτρέπουν την εκτέλεση διαδοχικών εξαρτώμενων λειτουργιών της ALU εντός ενός κύκλου ρολογιού. Μία εναλλακτική σχεδίαση, έχοντας στη διάθεσή της διπλάσιο αριθμό ALUs, δεν θα επέτρεπε αυτήν τη δυνατότητα. Ο αρχικός Pentium 4 χρησιμοποιούσε μία κρυφή μνήμη ίχνους BTB με 512 καταχωρίσεις, μια κρυφή μνήμη L1 των 8 KB και μία κρυφή μνήμη L2 των 256 KB.

προκαλέσουν την εισαγωγή διαστημάτων αδράνειας, καθώς οι ουρές γεμίζουν ενώ αναμένουν την ολοκλήρωση της αστοχίας.

Εξαιτίας της σπουδαιότητας τόσο της πρόβλεψης διακλάδωσης όσο και των αστοχιών της κρυφής μνήμης, εστιάζουμε την προσοχή μας στα δύο αυτά ζητήματα. Στα διαγράμματα της ενότητας αυτής παρουσιάζονται τα σχετικά αποτελέσματα με τη βοήθεια πέντε εκ των μετροπρογραμμάτων ακεραίων και πέντε εκ των μετροπρογραμμάτων FP του SPEC CPU2000, ενώ τα δεδομένα έχουν καταγραφεί με τη βοήθεια των μετρητών του Pentium 4, οι οποίοι έχουν σχεδιαστεί για την παρακολούθηση της απόδοσης. Ο επεξεργαστής είναι Pentium 4 640 στα 3.2 GHz, με δίαυλο συστήματος στα 800 MHz και κύρια μνήμη DDR2 DRAM στα 667 MHz.

Το Σχήμα 2.28 παρουσιάζει το ρυθμό των εσφαλμένων προβλέψεων των δια-

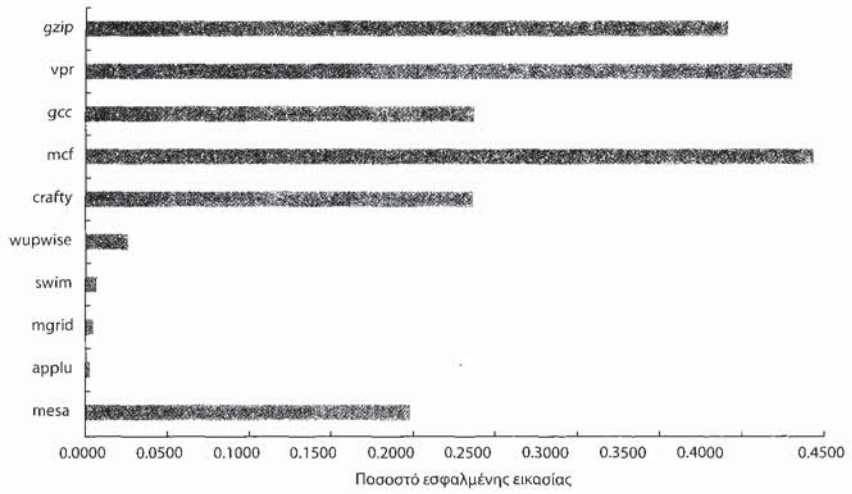


Σχήμα 2.28 Ρυθμός εσφαλμένων προβλέψεων διακλαδώσεων ανά 1000 εντολές για τα πέντε μετροπρογράμματα ακεραίων και FP της σουίτας SPEC CPU2000. Τα δεδομένα αυτά, καθώς και τα υπόλοιπα δεδομένα της ενότητας αυτής έχουν καταγραφεί με τη βοήθεια των John Holm και Dileep Bhandarkar της Intel.

κλαδώσεων έχοντας ως βάση μέτρησης τον αριθμό των εσφαλμένων προβλέψεων ανά 1000 εντολές. Ας έχουμε κατά νου ότι σε ό,τι αφορά την απόδοση της διασωλήνωσης, αυτό που διαδραματίζει πρωτεύοντα ρόλο είναι ο αριθμός των εσφαλμένων προβλέψεων ανά εντολή. Τα μετροπρογράμματα FP έχουν, σε γενικές γραμμές, λιγότερες διακλαδώσεις ανά εντολή (48 διακλαδώσεις ανά 1000 εντολές) απ' ό,τι τα μετροπρογράμματα ακεραίων (186 διακλαδώσεις ανά 1000 εντολές), καθώς και καλύτερο ρυθμό πρόβλεψης (98% έναντι 94%). Το αποτέλεσμα, όπως παρουσιάζεται στο Σχήμα 2.28, είναι το γεγονός ότι ο ρυθμός εσφαλμένων προβλέψεων ανά εντολή είναι 8 φορές υψηλότερος για τα μετροπρογράμματα ακεραίων σε σύγκριση με τα μετροπρογράμματα FP.

Η ακρίβεια της πρόβλεψης διακλάδωσης είναι ιδιαίτερα σημαντική στους υποθετικούς επεξεργαστές, καθώς η εσφαλμένη εικασία απαιτεί χρόνο ανάκαμψης, ενώ, συγχρόνως, σπαταλά ενέργεια ακολουθώντας τη λανθασμένη διαδρομή. Το Σχήμα 2.29 παρουσιάζει το ποσοστό των uops που έχουν εκτελεστεί και αποτελούν αποτέλεσμα εσφαλμένης εικασίας. Όπως θα υποπτευόταν κανείς, τα αποτελέσματα του ρυθμού εσφαλμένων εικασιών είναι σχεδόν ίδια με αυτά του ρυθμού εσφαλμένων προβλέψεων.

Με ποιον τρόπο συμβάλλουν οι αστοχίες της κρυφής μνήμης στις πιθανές απώλειες της απόδοσης; Ο ρυθμός αστοχίας της κρυφής μνήμης ίχνους αποτελεί σχεδόν αμελητέα ποσότητα για αυτό το σύνολο των μετροπρογραμμάτων του SPEC, καθώς μόνο ένα μετροπρόγραμμα (το 186.craft) παρουσιάζει σημαντικό ρυθμό αστοχίας (0.6%). Ο ρυθμός αστοχίας των κρυφών μνημών L1 και L2 είναι



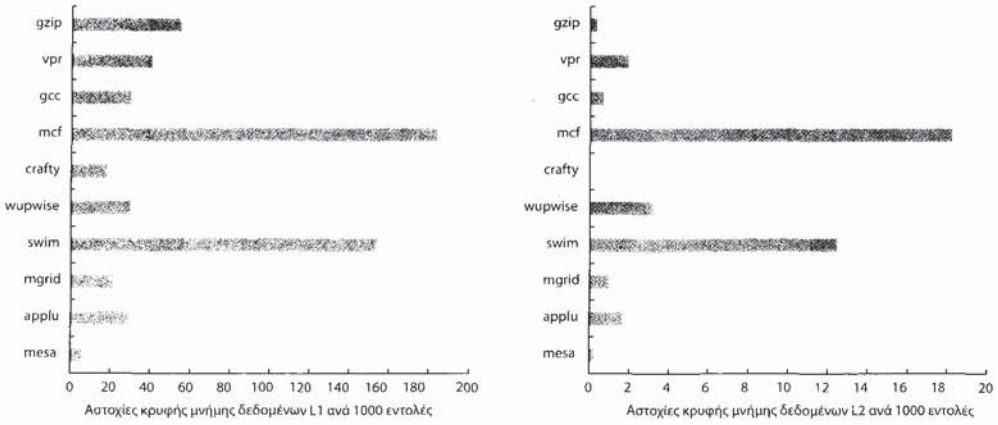
Σχήμα 2.29 Ποσοστό εντολών uop που εκδίδονται και αποτελούν αποτέλεσμα εσφαλμένης εικασίας.

πιο σημαντικός. Το Σχήμα 2.30 παρουσιάζει το ρυθμό αστοχίας των κρυφών μνημών L1 και L2 για τα 10 αυτά μετροπρογράμματα. Παρόλο που ο ρυθμός αστοχίας της L1 είναι περίπου 14 φορές υψηλότερος από το ρυθμό αστοχίας της L2, η ποινή αστοχίας της L2 είναι αρκετά υψηλότερη, ενώ η αδυναμία της μικροαρχιτεκτονικής να αποκρύψει αυτές τις ιδιαίτερα μεγάλες αστοχίες συνεπάγεται ότι οι αστοχίες της L2 είναι πιθανότατα υπεύθυνες για απώλειες ίδιου μεγέθους ή μεγαλύτερες από αυτές που επιφέρουν οι αστοχίες της L1, ιδίως για μετροπρογράμματα, όπως τα mcf και swim.

Με ποιον τρόπο τα αποτελέσματα της εσφαλμένης εικασίας και των αστοχιών της κρυφής μνήμης γίνονται αισθητά στο επίπεδο της πραγματικής απόδοσης; Το Σχήμα 2.31 παρουσιάζει το ενεργό CPI για τα 10 μετροπρογράμματα του SPEC CPU2000. Υπάρχουν τρία μετροπρογράμματα των οποίων η απόδοση ξεχωρίζει και χρήζει ιδιαίτερης ερμηνείας:

1. Το mcf έχει CPI το οποίο είναι περισσότερο από τέσσερις φορές μεγαλύτερο απ' ό,τι αυτό των τεσσάρων άλλων μετροπρογραμμάτων ακεραίων. Διαθέτει το χειρότερο ρυθμό εσφαλμένων εικασιών. Εξίσου σημαντικό είναι το γεγονός ότι το mcf διαθέτει το χειρότερο ρυθμό αστοχίας τόσο της L1 όσο και της L2 από όλα τα μετροπρογράμματα, ακεραίων ή κινητής υποδιαστολής, της σουίτας SPEC. Ο υψηλός ρυθμός αστοχίας των κρυφών μνημών καθιστά αδύνατη για τον επεξεργαστή την απόκρυψη του σημαντικά μεγάλου λανθάνοντος χρόνου αστοχίας.

2. Το vpr διαθέτει CPI το οποίο είναι 1.6 φορές υψηλότερο από αυτό των τριών εκ των πέντε μετροπρογραμμάτων (συμπεριλαμβανομένου και του mcf).

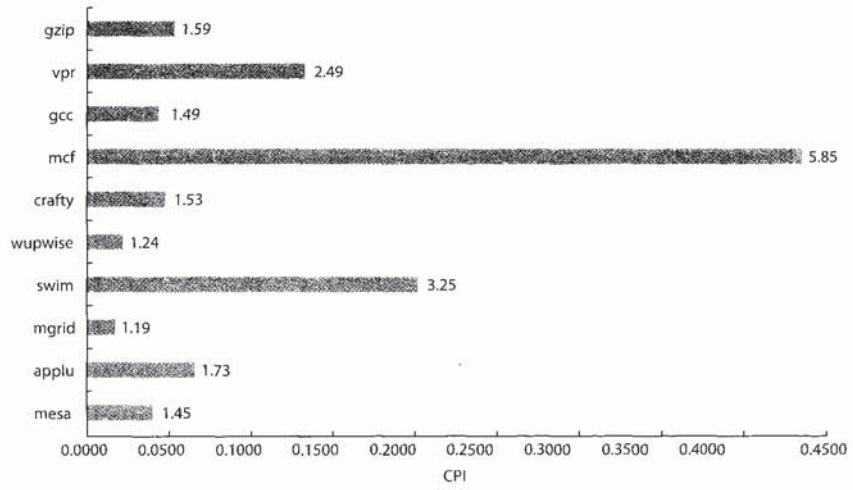


Σχήμα 2.30 **Αστοχίες των κρυφών μνημών δεδομένων L1 και L2 ανά 1000 εντολές για τα 10 μετροπρογράμματα του SPEC CPU2000.** Αξίζει να σημειωθεί ότι η κλίμακα των αστοχιών της L1 είναι 10 φορές μεγαλύτερη από αυτήν των αστοχιών της L2. Επειδή η ποινή αστοχίας της L2 πιθανότατα είναι τουλάχιστον 10 φορές μεγαλύτερη από αυτήν της L1, τα σχετικά μεγέθη των ράβδων αποτελούν ένδειξη της σχετικής ποινής της απόδοσης για τις αστοχίες σε κάθε κρυφή μνήμη. Η αδυναμία να αποκρυφτούν οι μεγάλες αστοχίες της L2 μέσω επικαλυπτόμενης εκτέλεσης αυξάνει επιπλέον τα διαστήματα αδράνειας που προκαλούνται εξαιτίας των αστοχιών της L2 σε σύγκριση με τις αστοχίες της L1.

Αυτό φαίνεται ότι προκύπτει εξαιτίας του ρυθμού εσφαλμένων προβλέψεων διακλάδωσης ο οποίος είναι ο χειρότερος μεταξύ των μετροπρογραμμάτων ακεραίων (αν και δεν είναι πολύ χειρότερος από το μέσο όρο), καθώς και του υψηλού ρυθμού αστοχίας της L2, του δεύτερου υψηλότερου, μετά από αυτόν του mcf, μεταξύ των μετροπρογραμμάτων ακεραίων.

3. Το swim αποτελεί το μετροπρόγραμμα FP με τη χειρότερη απόδοση και διαθέτει CPI το οποίο είναι δύο φορές υψηλότερο από το μέσο όρο των τεσσάρων άλλων μετροπρογραμμάτων FP. Τα προβλήματα του swim είναι ο υψηλός ρυθμός αστοχίας των κρυφών μνημών L1 και L2, που είναι ο υψηλότερος μετά από αυτόν του mcf. Αξίζει να σημειωθεί ότι το swim παρουσιάζει εξαιρετικά αποτελέσματα όσον αφορά την εικασία, ωστόσο η επιτυχία αυτή δεν είναι ικανή ώστε να αποκρύψει του υψηλούς ρυθμούς αστοχίας, ιδίως της L2. Αντιθέτως, αρκετά μετροπρογράμματα που παρουσιάζουν λογικό ρυθμό αστοχίας της L1 και χαμηλό ρυθμό αστοχίας της L2 (όπως το mgrid και το gzip) διαθέτουν καλή απόδοση.

Κλείνοντας την ενότητα αυτή, ας εξετάσουμε τη σχετική απόδοση των Pentium 4 και AMD Opteron για αυτό το υποσύνολο των μετροπρογραμμάτων του SPEC. Τα αποτελέσματα των AMD Opteron και Intel Pentium 4 παρουσιάζουν

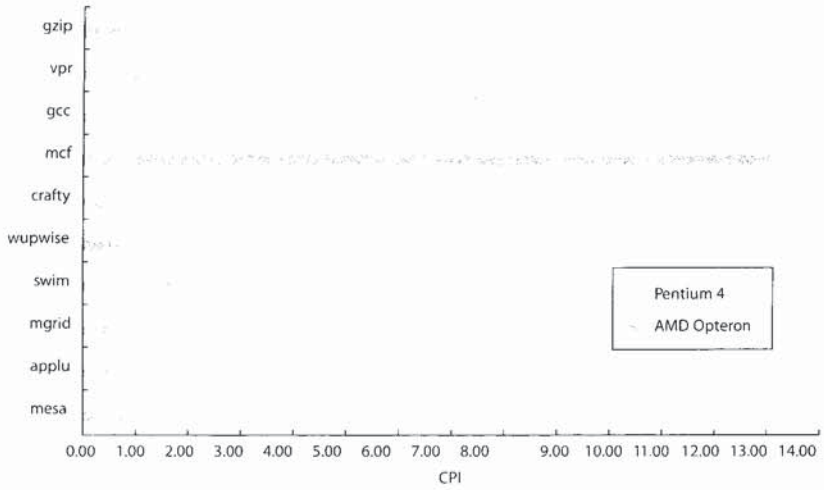


Σχήμα 2.31 **Το CPI των 10 μετροπρογραμμάτων του SPEC CPU.** Η αύξηση του CPI κατά 1.29 φορές προέρχεται από τη μετατροπή των εντολών της IA-32 σε uops, γεγονός που έχει, κατά μέσο όρο για τα 10 αυτά μετροπρογράμματα, ως αποτέλεσμα 1.29 uops ανά εντολή IA-32.

αρκετές ομοιότητες:

- Και οι δύο χρησιμοποιούν δυναμικά χρονοδρομολογημένη, υποθετική διασωλήνωση, η οποία είναι σε θέση να εκδίδει και να επικυρώνει τρεις εντολές IA-32 ανά κύκλο ρολογιού.
- Και οι δύο χρησιμοποιούν δομή κρυφής μνήμης δύο επιπέδων η οποία είναι τοποθετημένη στο chip, παρόλο που ο Pentium 4 χρησιμοποιεί κρυφή μνήμη ίχνους για το πρώτο επίπεδο κρυφής μνήμης εντολών, ενώ οι πρόσφατες υλοποιήσεις του Pentium 4 διαθέτουν μεγαλύτερες κρυφές μνήμες δεύτερου επιπέδου.
- Διαθέτουν παρόμοιο αριθμό transistors, μέγεθος κύβου και ισχύ, ενώ ο Pentium 4, σε όλες αυτές τις μετρήσεις, είναι κατά 7% έως 10% υψηλότερος όταν εφαρμόζονται οι υψηλότεροι (διαθέσιμοι κατά το 2005) ρυθμοί ρολογιού και στους δύο αυτούς επεξεργαστές.

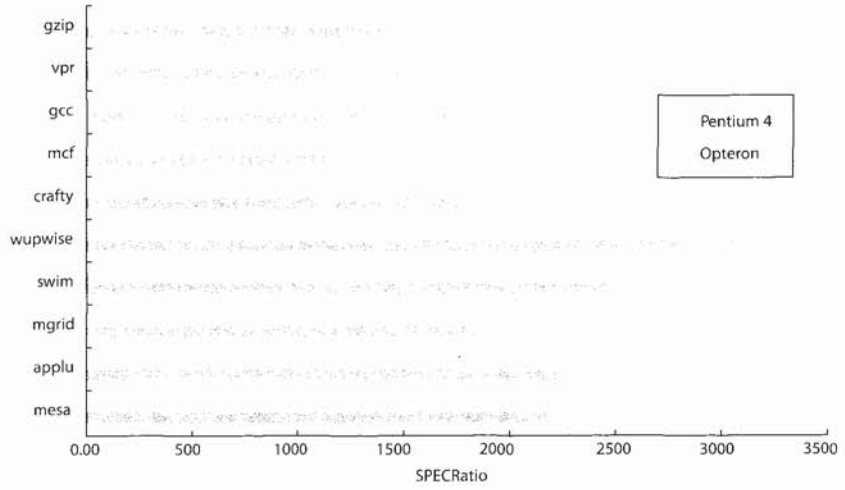
Η πιο σημαντική διαφορά έγκειται στο πολύ μεγάλο βάθος της διασωλήνωσης της μικροαρχιτεκτονικής του Intel Netburst, ο οποίος έχει σχεδιαστεί με τέτοιο τρόπο ώστε να επιτρέπει υψηλότερους ρυθμούς ρολογιού. Παρόλο που οι μεταγλωττιστές οι οποίοι είναι ρυθμισμένοι ώστε να λειτουργούν με το βέλτιστο τρόπο για καθεμία από τις δύο αρχιτεκτονικές παράγουν ελαφρά διαφορετικές ακολουθίες κώδικα, η αντιπαραβολή των μετρήσεων του CPI μπορεί να παράσχει σημαντικές πληροφορίες όσον αφορά τον τρόπο σύγκρισης των δύο αυτών



Σχήμα 2.32 Ο AMD Opteron στα 2.6 GHz έχει 1.27 φορές χαμηλότερο CPI από τον Pentium 4 στα 3.2 GHz.

επεξεργαστών. Ας έχουμε κατά νου ότι οι διαφορές στην ιεραρχία της μνήμης, καθώς και οι διαφορές στη δομή της διασωλήνωσης επηρεάζουν τις μετρήσεις αυτές. Η ανάλυση των διαφορών στην απόδοση του συστήματος της μνήμης αναλύονται στο Κεφάλαιο 5. Το Σχήμα 2.32 παρουσιάζει τις μετρήσεις του CPI για σύνολο μετροπρογραμμάτων του SPEC CPU2000 που έχουν εκτελεστεί σε Pentium 4 στα 3.2 GHz και σε AMD Opteron στα 2.6 GHz. Σε αυτούς τους ρυθμούς ρολογιού, ο επεξεργαστής Opteron παρουσιάζει κατά μέσο όρο, σε σύγκριση με τον Pentium 4, βελτίωση του CPI της τάξης του 1.27.

Βεβαίως, θα περιμέναμε ότι ο Pentium 4, ο οποίος διαθέτει διασωλήνωση με πολύ μεγαλύτερο βάθος, θα παρουσίαζε σε κάποιο βαθμό υψηλότερο CPI σε σύγκριση με τον AMD Opteron. Το βασικό ερώτημα που πρέπει να τεθεί και αφορά τη διασωλήνωση μεγάλου βάθους της σχεδίασης Netburst σχετίζεται με το αν η αύξηση του ρυθμού του ρολογιού, κάτι το οποίο επιτρέπει η διασωλήνωση μεγαλύτερου βάθους, είναι ικανή να υπερκεράσει τα μειονεκτήματα του υψηλότερου CPI. Θα το εξετάσουμε αυτό παρουσιάζοντας την απόδοση του SPEC CPU2000 για τους δύο αυτούς επεξεργαστές με βάση το μέγιστο διαθέσιμο κατά το 2005 ρυθμό ρολογιού για κάθε περίπτωση: ρυθμό της τάξης των 2.8 GHz για τον Opteron και των 3.8 GHz για τον Pentium 4. Αυτοί οι υψηλότεροι ρυθμοί ρολογιού οδηγούν στην αύξηση του ενεργού CPI σε σύγκριση με το CPI του Σχήματος 2.32, καθώς έχει αυξηθεί το κόστος των αστοχιών της κρυφής μνήμης. Το Σχήμα 2.33 παρουσιάζει τη σχετική απόδοση χρησιμοποιώντας το ίδιο υποσύνολο του SPEC με αυτό που χρησιμοποιείται στο Σχήμα 2.32. Ο Opteron είναι ελαφρά ταχύτερος, γεγονός που συνεπάγεται ότι ο υψηλότερος ρυθμός ρολογιού του Pentium 4 δεν επαρκεί για την αντιμετώπιση του υψηλότερου CPI



Σχήμα 2.33 **Η απόδοση του AMD Opteron στα 2.8 GHz σε σύγκριση με τον Intel Pentium 4 στα 3.8 GHz παρουσιάζει το ελαφρύ προβάδισμα, της τάξης του 1.08, του Opteron.**

που προκύπτει εξαιτίας του μεγαλύτερου αριθμού των διαστημάτων αδράνειας της διασωλήνωσης.

Συνεπώς, παρόλο που ο Pentium 4 λειτουργεί ικανοποιητικά, είναι σαφές ότι η προσπάθεια επίτευξης αφενός υψηλού ρυθμού ρολογιού μέσω της χρησιμοποίησης διασωλήνωσης μεγάλου βάθους και αφετέρου υψηλής ρυθμοαπόδοσης εντολών μέσω πολλαπλής έκδοσης δεν μπορεί να στεφθεί με επιτυχία στο βαθμό που κάποτε οι σχεδιαστές πίστευαν ότι είναι εφικτό. Στο επόμενο κεφάλαιο, θα εξετάσουμε το ζήτημα αυτό με περισσότερες λεπτομέρειες.

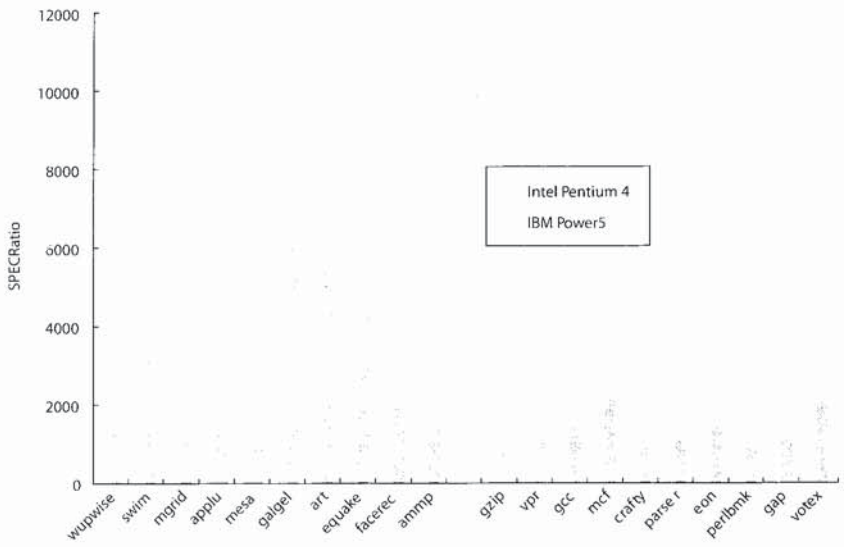
2.11 Πλάνες και Παγίδες

Η πρώτη πλάνη που παρουσιάζουμε διαθέτει δύο σκέλη: Πρώτον, δεν ισχύουν οι απλοί κανόνες και, δεύτερον, η επιλογή των μετροπρογραμμάτων διαδραματίζει σημαντικό ρόλο.

Πλάνη: *Οι επεξεργαστές που διαθέτουν χαμηλότερο CPI είναι πάντα ταχύτεροι.*

Πλάνη: *Οι επεξεργαστές που διαθέτουν υψηλότερο ρυθμό ρολογιού είναι πάντα ταχύτεροι.*

Παρόλο που το χαμηλότερο CPI είναι ασφαλώς καλύτερο, οι προηγμένες διασωληνώσεις πολλαπλής έκδοσης έχουν χαμηλότερο ρυθμό ρολογιού απ' ό,τι οι επεξεργαστές με απλή διασωλήνωση. Στις εφαρμογές που διαθέτουν περιορισμέ-



Σχήμα 2.34 **Η σύγκριση του επεξεργαστή Power 5 της IBM στα 1.9 GHz με τον Pentium 4 της Intel στα 3.8 GHz με τη βοήθεια 20 μετροπρογραμμάτων του SPEC (10 μετροπρογραμμάτων ακεραίων στο αριστερό τμήμα του σχήματος και 10 μετροπρογραμμάτων κινητής υποδιαστολής στο δεξιό τμήμα) καταδεικνύει ότι ο Pentium 4, που διαθέτει υψηλότερο ρυθμό ρολογιού, είναι σε γενικές γραμμές ταχύτερος σε ό,τι αφορά τις εντολές ακεραίων, ενώ ο Power 5, που διαθέτει χαμηλότερο CPI, είναι συνήθως ταχύτερος όσον αφορά τις εντολές κινητής υποδιαστολής.**

νο βαθμό ILP ή στις οποίες δεν είναι εφικτή η αξιοποίηση του παραλληλισμού από τους πόρους του υλικού, η προτιμώμενη επιλογή είναι ο υψηλότερος ρυθμός ρολογιού. Ωστόσο, όταν υπάρχει σημαντικός βαθμός ILP, τότε οι επεξεργαστές που αξιοποιούν τον ILP μπορεί να είναι καλύτεροι.

Ο επεξεργαστής της IBM Power 5 έχει σχεδιαστεί με στόχο την επίτευξη υψηλής απόδοσης για εντολές ακεραίων και FP. Διαθέτει δύο πυρήνες επεξεργαστών, καθένας εκ των οποίων είναι ικανός να εκτελεί τέσσερις εντολές ανά κύκλο ρολογιού, συμπεριλαμβανομένων δύο εντολών FP και δύο εντολών φόρτωσης - αποθήκευσης. Το 2005, ο υψηλότερος ρυθμός ρολογιού του επεξεργαστή Power 5 ανερχόταν στα 1.9 GHz. Από την άλλη πλευρά, ο Pentium 4 παρέχει έναν επεξεργαστή που υποστηρίζει την πολυνημάτωση (ανατρέξτε στο επόμενο κεφάλαιο). Ο επεξεργαστής μπορεί να εκτελεί τρεις εντολές ανά κύκλο ρολογιού και διαθέτει διασωλήνωση ιδιαίτερα μεγάλου βάθους, ενώ ο μέγιστος διαθέσιμος ρυθμός ρολογιού κατά το 2005 ισούτο με 3.8 GHz.

Επομένως, ο Power 5 θα είναι ταχύτερος, αν το γινόμενο του αριθμού των εντολών με το CPI είναι μικρότερο από το μισό του ίδιου γινομένου για τον Pentium 4. Όπως φαίνεται στο Σχήμα 2.34, το προβάδισμα του Power 5 που προκύ-

πτει από το γινόμενο CPI×αριθμός εντολών είναι σημαντικό για τα προγράμματα FP, προβάδισμα που σε αρκετές περιπτώσεις έχει διπλάσιο ή και ακόμα μεγαλύτερο μέγεθος, ενώ σε ό,τι αφορά τα προγράμματα ακεραίων το προβάδισμα του Power 5 που προκύπτει από το γινόμενο CPI × αριθμός εντολών συνήθως δεν επαρκεί για να υπερκεράσει το πλεονέκτημα που διαθέτει ο Pentium 4 εξαιτίας του υψηλού ρυθμού ρολογιού του. Συγκρίνοντας τα αποτελέσματα του SPEC διαπιστώνει κανείς ότι το προβάδισμα του Power 5 που προκύπτει από το γινόμενο CPI × αριθμός εντολών είναι 3.1 φορές μεγαλύτερο για τα προγράμματα κινητής υποδιαστολής, ενώ μόλις 1.5 φορές μεγαλύτερο για τα προγράμματα ακεραίων. Επειδή ο μέγιστος ρυθμός ρολογιού του Pentium 4 το 2005 ήταν ακριβώς διπλάσιος από αυτόν του Power 5, ο Power 5 είναι 1.5 φορές ταχύτερος από τον του Pentium 4 για το SPECfp2000, ενώ ο Pentium 4 είναι 1.3 φορές ταχύτερος από τον Power 5 για το SPECint2000.

Παγίδα: *Μερικές φορές το μεγαλύτερο μέγεθος, χωρίς να συνοδεύεται από υψηλό βαθμό ευφυΐας, είναι καλύτερο.*

Οι προηγμένες διασωληνώσεις έχουν εστιάσει την προσοχή τους σε νέες και ολοένα και πιο εκλεπτυσμένες προσεγγίσεις βελτίωσης του CPI. Ο 21264 χρησιμοποιεί έναν προηγμένο επιλεκτικό μηχανισμό πρόβλεψης που διαθέτει συνολικά 29K bits (δείτε στη σελίδα 121), ενώ η προηγούμενη έκδοση, αυτή του 21164, χρησιμοποιεί έναν απλό μηχανισμό πρόβλεψης με 2K καταχωρίσεις (ή συνολικά 4K bits). Στα μετροπρογράμματα του SPEC95, ο πιο προηγμένος μηχανισμός πρόβλεψης διακλάδωσης του 21264 έχει υψηλότερη απόδοση από την πιο απλή προσέγγιση των 2 bits σε όλα τα μετροπρογράμματα, εκτός από ένα. Κατά μέσο όρο, στο SPECint95, ο 21264 παρουσιάζει 11.5 εσφαλμένες προβλέψεις ανά 1000 εντολές που επικυρώνονται, ενώ ο 21164 παρουσιάζει περίπου 16.5 εσφαλμένες προβλέψεις.

Αυτό που προκαλεί σε κάποιο βαθμό έκπληξη είναι το γεγονός ότι η πιο απλή προσέγγιση των 2 bits παρουσιάζει καλύτερη απόδοση για τις εργασίες επεξεργασίας συναλλαγών από την προηγμένη προσέγγιση του 21264 (17 έναντι 19 εσφαλμένων προβλέψεων ανά 1000 ολοκληρωμένες εντολές)! Πώς είναι δυνατό να έχει στην πραγματικότητα υψηλότερη απόδοση ο μηχανισμός πρόβλεψης που διαθέτει λιγότερο από το 1/7 του αριθμού των bits και ακολουθεί πιο απλή προσέγγιση; Η δομή του φορτίου εργασιών δίνει την απάντηση στο παραπάνω ερώτημα. Το φορτίο εργασιών της επεξεργασίας συναλλαγών διαθέτει πολύ μεγάλο μέγεθος κώδικα (κώδικα μίας τάξης μεγέθους μεγαλύτερο από οποιοδήποτε μετροπρόγραμμα του SPEC95) με μεγάλη συχνότητα διακλαδώσεων. Η ικανότητα του μηχανισμού πρόβλεψης του 21164 να διατηρεί διπλάσιο αριθμό προβλέψεων διακλάδωσης με βάση μόνο την τοπική συμπεριφορά (2K έναντι του 1K του τοπικού μηχανισμού πρόβλεψης του 21264) φαίνεται ότι παρέχει ένα ελαφρύ πλεονέκτημα.

Η παγίδα αυτή μας υπενθυμίζει ότι διαφορετικές εφαρμογές μπορεί να έχουν ως αποτέλεσμα διαφορετικές συμπεριφορές. Όσο οι επεξεργαστές αποκτούν ολοένα και πιο εκλεπτυσμένες ιδιότητες, συμπεριλαμβανομένων και συγκεκριμένων χαρακτηριστικών της μικροαρχιτεκτονικής τα οποία έχουν ως στόχο την επίτευξη συγκεκριμένης συμπεριφοράς από την πλευρά των προγραμμάτων, είναι πιθανό διαφορετικές εφαρμογές να παρουσιάζουν περισσότερο αποκλίνουσα συμπεριφορά.

2.12 Συμπερασματικές Παρατηρήσεις

Το τεράστιο ενδιαφέρον στις προσεγγίσεις πολλαπλής έκδοσης προέκυψε εξαιτίας της ανάγκης βελτίωσης της απόδοσης χωρίς να επηρεαστεί το τυποποιημένο μοντέλο προγραμματισμού του μονού επεξεργαστή (uniprocessor). Παρόλο που η αξιοποίηση του ILP είναι εννοιολογικά απλή, τα προβλήματα σχεδίασης που προκύπτουν στην πράξη είναι απίστευτα πολύπλοκα. Είναι εξαιρετικά δύσκολο να επιτευχθεί η απόδοση που θα μπορούσε να αναμένει κανείς προβαίνοντας σε μια απλή ανάλυση πρώτου επιπέδου.

Τα τελευταία 10 χρόνια οι περισσότερες προσπάθειες δεν έχουν επικεντρωθεί στην ανάπτυξη νέων σημαντικών προσεγγίσεων σε επίπεδο μικροαρχιτεκτονικής, αλλά αντιθέτως έχουν εστιασθεί στην αύξηση του ρυθμού του ρολογιού των επεξεργαστών πολλαπλής έκδοσης και στη γεφύρωση του χάσματος ανάμεσα στην υψηλότερη και τη μέση απόδοση. Οι δυναμικά χρονοδρομολογημένοι επεξεργαστές πολλαπλής έκδοσης που έχουν ανακοινωθεί τα τελευταία 5 έτη (οι Pentium 4, IBM Power 5, καθώς και οι AMD Athlon και Opteron) διαθέτουν την ίδια βασική δομή και παρόμοιους μέσους ρυθμούς έκδοσης (τριών έως τεσσάρων εντολών ανά κύκλο ρολογιού) με τους πρώτους δυναμικά χρονοδρομολογημένους επεξεργαστές πολλαπλής έκδοσης που κυκλοφόρησαν το 1995! Ωστόσο ο ρυθμός ρολογιού είναι 10-20 φορές υψηλότερος, η κρυφή μνήμη 4-8 φορές μεγαλύτερη, ενώ υπάρχουν 2-4 φορές περισσότεροι καταχωρητές μειονομασίας και διπλάσιος αριθμός μονάδων φόρτωσης - αποθήκευσης. Αποτέλεσμα είναι η επίτευξη απόδοσης που είναι 8-16 φορές υψηλότερη.

Η σχέση που υφίσταται ανάμεσα στην αύξηση της ταχύτητας του ρολογιού και τη μείωση του CPI μέσω της πολλαπλής έκδοσης είναι πολύ δύσκολο να ποσοτικοποιηθεί. Στην έκδοση του βιβλίου αυτού που κυκλοφόρησε το 1995 είχαμε υποστηρίξει τα παρακάτω:

> Παρόλο που θα ανέμενε κανείς ότι είναι εφικτή η ανάπτυξη προηγμένων επεξεργαστών πολλαπλής έκδοσης με υψηλό ρυθμό ρολογιού, ο ρυθμός ρολογιού των επεξεργαστών υψηλού ρυθμού ήταν ανέκαθεν 1.5 έως 2 φορές υψηλότερος από αυτόν που διέθεταν οι πιο εκλεπτυσμένοι επεξεργαστές πολλαπλής έκδοσης. Είναι απλά πολύ νωρίς να διακρίνει κανείς αν η διαφορά αυτή οφείλεται σε θεμελιώδεις αλληλοεξαρτώμενες σχέσεις που αφορούν την υλοποίηση, στη δυσκολία

αντιμετώπισης των πολύπλοκων ζητημάτων που αφορούν τους επεξεργαστές πολλαπλής έκδοσης ή απλά στην έλλειψη εμπειρίας ανάπτυξης τέτοιων επεξεργαστών.

Έχοντας ως δεδομένες τις δυνατότητες του Pentium 4 στα 3.8 GHz, είναι πλέον σαφές ότι ο βασικός περιορισμός ήταν η κατανόησή μας όσον αφορά τον τρόπο ανάπτυξης αυτού του είδους των επεξεργαστών. Ωστόσο, όπως θα διαπιστώσουμε στο επόμενο κεφάλαιο, δεν είναι ακόμα σαφές αν η αρχική επιτυχία δημιουργίας επεξεργαστών υψηλού ρυθμού ρολογιού που εκδίδουν τρεις έως τέσσερις εντολές ανά κύκλο ρολογιού μπορεί να εξελιχθεί περαιτέρω, εξαιτίας περιορισμών που αφορούν το διαθέσιμο ILP και την αποτελεσματικότητα της αξιοποίησης αυτού του ILP, καθώς και περιορισμών που σχετίζονται με την ισχύ. Επιπλέον, όπως διαπιστώσαμε από τη σύγκριση του Opteron και του Pentium 4, φαίνεται ότι τα πλεονεκτήματα που επιφέρουν οι διασωληνώσεις ιδιαίτερα μεγάλου βάθους (20-30 σταδίων) όσον αφορά την απόδοση σε υψηλούς ρυθμούς ρολογιού, εκμηδενίζονται εξαιτίας της εισαγωγής επιπρόσθετων διαστημάτων αδράνειας στη διασωλήνωση. Στο επόμενο κεφάλαιο, αναλύουμε τη συμπεριφορά αυτή σε μεγαλύτερο βαθμό.

Ένα χαρακτηριστικό, που ήταν εμφανές το 1995 και έχει γίνει ακόμα πιο προφανές το 2005, αφορά το γεγονός ότι ο λόγος της υψηλότερης προς τη μέση απόδοση για τους επεξεργαστές πολλαπλής έκδοσης είναι συχνά μεγάλος και συνήθως μεγαλώνει ακόμα περισσότερο παράλληλα με την αύξηση του βαθμού έκδοσης. Η γνώση που έχουμε αποκτήσει από τη σύγκριση του Power 5 και του Pentium 4 ή του Pentium 4 και του Pentium III (οι οποίοι διαφέρουν πρωτίστως ως προς το βάθος της διασωλήνωσης και ως εκ τούτου ως προς το ρυθμό του ρολογιού και όχι ως προς το ρυθμό έκδοσης) μας υπενθυμίζει ότι είναι δύσκολη η διατύπωση γενικευμένων κανόνων αφενός όσον αφορά τη σχέση του ρυθμού του ρολογιού και του CPI και αφετέρου σε ό,τι αφορά την αλληλεπιδραστική σχέση ανάμεσα στο βάθος της διασωλήνωσης, το ρυθμό έκδοσης και άλλα χαρακτηριστικά.

Είναι εμφανές ότι επίκειται κάποια αλλαγή σε επίπεδο προσέγγισης. Έχουν εγκαταλειφθεί οι εκδόσεις του Pentium 4 με υψηλό ρυθμό ρολογιού. Η IBM έχει στρέψει το ενδιαφέρον της στην ενσωμάτωση δύο επεξεργαστών εντός ενός chip στις σειρές Power 4 και Power 5, ενώ τόσο η Intel όσο και η AMD έχουν ήδη θέσει σε κυκλοφορία κάποιες αρχικές εκδόσεις chips που διαθέτουν δύο επεξεργαστές. Θα επιστρέψουμε στο ζήτημα αυτό στο επόμενο κεφάλαιο, στο οποίο και θα καταδειχθεί γιατί η εικοσάχρονη ταχύτατη επιδίωξη αξιοποίησης του ILP φαίνεται πως έχει φτάσει στο τέλος της.

2.13 Ιστορική Προοπτική και Πρόσθετες Πηγές

Η Ενότητα Κ.4 στο συνοδευτικό CD περιλαμβάνει τη διερεύνηση της ανάπτυξης της διασωλήνωσης και του παραλληλισμού επιπέδου εντολών. Παρέχουμε πλη-

θώρα πρόσθετων πηγών στις οποίες μπορεί να ανατρέξει ο αναγνώστης για την περαιτέρω διερεύνηση των ζητημάτων αυτών.

Μελέτες Περίπτωσης με Ασκήσεις από τον Robert P. Colwell

Μελέτη Περίπτωσης 1: Διερεύνηση των Επιπτώσεων των Τεχνικών σε Επίπεδο Μικροαρχιτεκτονικής

Έννοιες που αναλύονται σε αυτήν τη μελέτη περίπτωσης

- Βασική Χρονοδρομολόγηση Εντολών, Αναδιάταξη, Διαβίβαση
- Πολλαπλή Έκδοση και Κίνδυνοι
- Μετονομασία Καταχωρητών
- Υποθετική Εκτέλεση και Εκτέλεση Εκτός Σειράς
- Πώς Μπορούν να Δαπανηθούν οι Πόροι Εκτός Σειράς

Σας ζητείται να σχεδιάσετε μία νέα μικροαρχιτεκτονική επεξεργαστών και προσπαθείτε να διαπιστώσετε ποιος είναι ο βέλτιστος τρόπος ανάθεσης των πόρων υλικού. Ποιες από τις τεχνικές υλικού και λογισμικού που μάθατε στο Κεφάλαιο 2 πρέπει να εφαρμόσετε; Στη διάθεσή σας έχετε λίστα με τους λανθάνοντες χρόνους των λειτουργικών μονάδων και της μνήμης, καθώς και κάποιον αντιπροσωπευτικό κώδικα. Ο προϊστάμενός σας είναι κατά κάποιον τρόπο ασαφής όσον αφορά τις απαιτήσεις της νέας σας σχεδίασης στο επίπεδο της απόδοσης, ωστόσο γνωρίζετε με βάση την εμπειρία σας ότι, αν όλα τα υπόλοιπα χαρακτηριστικά βρίσκονται στο ίδιο επίπεδο, η προτιμώμενη λύση είναι συνήθως η ταχύτερη. Ξεκινήστε από τα βασικά. Το Σχήμα 2.35 παρέχει ακολουθία εντολών και παραθέτει τη λίστα με τους λανθάνοντες χρόνους.

2.1 [10]<1.8, 2.1, 2.2> Ποια θα ήταν η αρχική απόδοση (σε κύκλους ανά βρόχο επανάληψης) της ακολουθίας κώδικα του Σχήματος 2.35, αν δεν ήταν εφικτή η εκκίνηση της εκτέλεσης καμίας εντολής πριν την ολοκλήρωση της εκτέλεσης της προηγούμενης εντολής; Μη λάβετε υπόψη την ανάκτηση και την αποκωδικοποίηση. Θεωρήστε, προς το παρόν, ότι η εκτέλεση δεν τίθεται σε αδράνεια εξαιτίας έλλειψης επόμενης εντολής, ωστόσο μπορεί να εκδίδεται μόνο μία εντολή/κύκλο. Υποθέστε, επίσης, ότι η διακλάδωση ακολουθείται, καθώς και ότι υπάρχει θυρίδα καθυστέρησης διακλάδωσης του ενός κύκλου.

```
Loop: LD     F2,0(Rx)
IO:   MULTD  F2,F0,F2
I1:   DIVD   F8,F2,F0
I2:   LD     F4,0(Ry)
I 3:  ADDD   F4, F0, F4
I 4:  ADDD   F10, F8, F2
I 5:  SD     F4, 0( Ry)
I6:   ADDI   Rx,Rx,#8
I7:   ADDI   Ry,Ry,#8
I8:   SUB    R20,R4,Rx
I9:   BNZ    R20,Loop
```

Λανθάνοντες χρόνοι που ξεπερνούν τον ένα μόνο κύκλο	
LD μνήμης	+3
SD μνήμης	+1
ADD, SUB ακεραίων	+0
Διακλαδώσεις	+1
ADDD	+2
MULTD	+4
DI VD	+10

Σχήμα 2.35 **Κώδικας και λανθάνοντες χρόνοι για τις Ασκήσεις 2.1 έως 2.6.**

2.2 [10]<1.8, 2.1, 2.2> Σκεφτείτε τι συμβολίζουν στην πραγματικότητα οι λανθάνοντες χρόνοι - υποδηλώνουν μόνο τον αριθμό των κύκλων που χρειάζεται μια δεδομένη συνάρτηση για την παραγωγή του αποτελέσματός της και τίποτα παραπάνω. Αν συνολικά η διασωλήνωση τίθεται σε αδράνεια για τους λανθάνοντες κύκλους κάθε λειτουργικής μονάδας, τότε είναι διασφαλισμένη η ορθή εκτέλεση οποιουδήποτε ζεύγους διαδοχικών εξαρτώμενων εντολών (μιας εντολής «παραγωγού» που ακολουθείται από εντολή «καταναλωτή»). Ωστόσο, δεν έχουν όλα τα ζεύγη εντολών τη σχέση παραγωγού/καταναλωτή. Αρκετές φορές, δύο γειτονικές εντολές δεν έχουν καμία σχέση μεταξύ τους. Πόσους κύκλους θα χρειαζόταν το σώμα του βρόχου στην ακολουθία κώδικα του Σχήματος 2.35, αν η διασωλήνωση ανίχνευε τις πραγματικές εξαρτήσεις δεδομένων και ετίθετο σε αδράνεια μόνο σε αυτήν την περίπτωση, αντί να τίθεται σε αδράνεια αδιακρίτως απλά και μόνο επειδή μια λειτουργική μονάδα είναι απασχολημένη; Παραθέστε τον κώδικα εισαγάγοντας την πρόταση `<stall>` όπου αυτό είναι απαραίτητο για την ικανοποίηση των διατυπωμένων λανθανόντων χρόνων (*Συμβουλή:* Οι εντολές που έχουν λανθάνοντα χρόνο «+2» χρειάζονται την εισαγωγή δύο κύκλων `<stall>` εντός της ακολουθίας του κώδικα. Σκεφτείτε το ως εξής: οι εντολές του ενός κύκλου έχουν λανθάνοντα χρόνο $1+0$, κάτι που συνεπάγεται μηδενικό αριθμό καταστάσεων αναμονής. Συνεπώς, ο λανθάνων χρόνος $1+1$ συνεπάγεται 1 κύκλο αδράνειας και ως εκ τούτου ο λανθάνων χρόνος $1+N$ διαθέτει N κύκλους αδράνειας).

2.3 [15]<2.6, 2.7> Εξετάστε τη σχεδίαση πολλαπλής έκδοσης. Θεωρήστε ότι έχετε στη διάθεσή σας δύο διασωληνώσεις εκτέλεσης, καθεμία εκ των οποίων είναι σε θέση να ξεκινά την εκτέλεση μίας εντολής ανά κύκλο, καθώς και επαρκές εύρος ζώνης ανάκτησης/αποκωδικοποίησης, έτσι ώστε η εκτέλεση να μην τίθεται σε αδράνεια. Υποθέστε ότι τα αποτελέσματα μπορούν να προωθηθούν αμέσως από τη μία μονάδα εκτέλεσης στην επόμενη ή και προς τον εαυτό της. Επιπλέον, θεωρήστε ότι ο μοναδικός λόγος για τον οποίο οι διασωληνώσεις εκτέλεσης θα μπορούσαν να τεθούν σε αδράνεια είναι η ύπαρξη πραγματικής εξάρτησης δεδομένων. Πόσους κύκλους χρειάζεται πλέον ο βρόχος;

2.4 [10]<2.6, 2.7> Στη σχεδίαση πολλαπλής έκδοσης της Άσκησης 2.3, μπορεί να έχετε εντοπίσει μερικά δυσδιάκριτα ζητήματα. Οι δύο διασωληνώσεις εκτέλεσης, παρόλο που διαθέτουν το ίδιο ακριβώς ρεπερτόριο εντολών, ούτε είναι πανομοιότυπες ούτε μπορούν να εναλλάσσονται μεταξύ τους, διότι πρέπει να είναι εμμέσως ταξινομημένες μεταξύ τους με διάταξη η οποία αντικατοπτρίζει τη διάταξη των εντολών του αρχικού προγράμματος. Αν ξεκινήσει η εκτέλεση της εντολής $N + 1$ στη Σωλήνωση Εκτέλεσης 1 ταυτόχρονα με την έναρξη της εκτέλεσης της εντολής N στη Σωλήνωση 0 και, παράλληλα, η $N+1$ τυγχάνει να απαιτεί μικρότερο λανθάνοντα χρόνο εκτέλεσης απ' ό,τι η N, τότε η $N + 1$ θα ολοκληρωθεί πριν τη N (παρόλο που η διάταξη του προγράμματος υποδηλώνει το αντίθετο). Απαριθμήστε τουλάχιστον δύο λόγους για τους οποίους το παραπάνω φαινόμενο θα μπορούσε να είναι επικίνδυνο και θα απαιτούσε ιδιαίτερη αντιμετώπιση σε επίπεδο μικροαρχιτεκτονικής. Παραθέστε ένα παράδειγμα δύο εντολών του κώδικα του Σχήματος 2.35 που εμφανίζουν τον κίνδυνο αυτό.

2.5 [20]<2.7> Αναδιατάξτε τις εντολές για τη βελτίωση της απόδοσης του κώδικα του Σχήματος 2.35. Θεωρήστε ότι έχετε στη διάθεσή σας τη μηχανή των δύο διασωληνώσεων της Άσκησης 2.3, καθώς και ότι τα ζητήματα της εκτός σειράς ολοκλήρωσης της Άσκησης 2.4 έχουν αντιμετωπιστεί με επιτυχία. Προς το παρόν, το μόνο που σας ενδιαφέρει είναι η παρατήρηση των πραγματικών εξαρτήσεων δεδομένων και των λανθανόντων χρόνων των λειτουργικών μονάδων. Πόσους κύκλους χρειάζεται ο αναδιαταγμένος κώδικάς σας;

2.6 [10/10]<2.1, 2.2> Κάθε κύκλος που δεν εκκινεί μια νέα λειτουργία στη σωλήνωση αποτελεί μία χαμένη ευκαιρία, με την έννοια ότι το υλικό «δεν αποδίδει τα μέγιστα των δυνατοτήτων του».

   α. [10]<2.1, 2.2> Στον αναδιαταγμένο κώδικα της Άσκησης 2.5, ποιο είναι το ποσοστό εκ του συνόλου των κύκλων που δαπανήθηκαν άσκοπα (κατά τους οποίους δεν ξεκίνησε μία νέα op), αν λάβουμε υπόψη

και τις δύο σωληνώσεις;

β. [10]<2.1, 2.2> Το ξεδίπλωμα βρόχου αποτελεί μία από τις τυποποιημένες τεχνικές των μεταγλωττιστών για την ανίχνευση μεγαλύτερου εύρους παραλληλισμού εντός του κώδικα, έτσι ώστε να ελαχιστοποιηθούν οι χαμένες ευκαιρίες που μειώνουν την απόδοση.

γ. Ξεδιπλώστε στο χαρτί σας δύο επαναλήψεις του βρόχου του αναδιαταγμένου κώδικα της Άσκησης 2.5. Ποιος είναι ο βαθμός επιτάχυνσης (speedup) που προκύπτει; (Στην άσκηση αυτή, απλά χρωματίστε με πράσινο χρώμα τις εντολές επανάληψης της θέσης $N+1$, ώστε να τις ξεχωρίζετε από τις Nιοστές επαναλήψεις. Αν ξεδιπλώνατε το βρόχο, στην πραγματικότητα, θα έπρεπε να προβείτε στην εκ νέου ανάθεση των καταχωρητών για την παρεμπόδιση εμφάνισης συγκρούσεων μεταξύ των επαναλήψεων.)

2.7 [15]<2.1> Οι υπολογιστές δαπανούν το μεγαλύτερο μέρος του χρόνου τους στους βρόχους και ως εκ τούτου οι πολλαπλές επαναλήψεις βρόχων αποτελούν την καλύτερη περίπτωση μέσω της οποίας μπορούμε με υποθετικό τρόπο να διατηρούμε τους πόρους της CPU απασχολημένους. Ωστόσο, η διαδικασία αυτή δεν είναι καθόλου εύκολη. Ο μεταγλωττιστής έχει παραγάγει μόνο ένα αντίγραφο του κώδικα του βρόχου και, επομένως, παρόλο που οι πολλαπλές επαναλήψεις διαχειρίζονται ξεχωριστά μεταξύ τους δεδομένα, εμφανίζονται να χρησιμοποιούν τους ίδιους καταχωρητές. Προκειμένου να αποφευχθεί η σύγκρουση των πολλαπλών επαναλήψεων που χρησιμοποιούν τους ίδιους καταχωρητές, προβαίνουμε στη μετονομασία των καταχωρητών τους. Το Σχήμα 2.36 παρουσιάζει ενδεικτικό κώδικα τον οποίο επιθυμούμε να μετονομάσει το υλικό μας.

Ο μεταγλωττιστής θα μπορούσε απλά να ξεδιπλώσει το βρόχο και να χρησιμοποιήσει διαφορετικούς καταχωρητές για την αποφυγή των συγκρούσεων, ωστόσο αν το υλικό μας απαιτείται να ξεδιπλώσει το βρόχο, πρέπει, επίσης, να είναι σε θέση να πραγματοποιήσει τη μετονομασία των καταχωρητών. Με ποιον τρόπο; Θεωρήστε ότι το υλικό σας διαθέτει σύνολο προσωρινών καταχωρητών (τους οποίους ονομάζουμε καταχωρητές T και κάνουμε την υπόθεση ότι υπάρχουν 64 τέτοιοι καταχωρητές που έχουν τις ονομασίες T0 έως και T63), οι οποίοι μπορούν να αντικαταστήσουν τους καταχωρητές που έχει καθορίσει ο μεταγλωττιστής. Αυτό το υλικό μετονομασίας δεικτοδοτείται μέσω του ορισμού του καταχωρητή αφετηρίας και η τιμή του πίνακα αντιπροσωπεύει τον καταχωρητή T του τελευταίου προορισμού που έδειχνε προς αυτόν τον καταχωρητή (Αντιμετωπίστε αυτές τις τιμές του πίνακα ως παραγωγούς και τους καταχωρητές αφετηρίας ως καταναλωτές. Δεν έχει ιδιαίτερη σημασία πού τοποθετεί ο παραγωγός το αποτέλεσμά του, αρκεί οι καταναλωτές του να μπορούν να το εντοπί-

```
Loop: LD     F2,0(Rx)
I0:   MULTD  F5,F0,F2
I1:   DIVD   F8,F0,F2
I2:   LD     F4,0(Ry)
I3:   ADDD   F6,F0,F4
I4:   ADDD   F10,F8,F2
I5:   SD     F4,0(Ry)
```

Σχήμα 2.36 **Ενδεικτικός κώδικας για πρακτική άσκηση στη μετονομασία καταχωρητών.**

ζουν). Εξετάστε την ακολουθία κώδικα του Σχήματος 2.36. Κάθε φορά που διακρίνετε στον κώδικα κάποιον καταχωρητή προορισμού, αντικαταστήστε τον με τον επόμενο διαθέσιμο καταχωρητή Τ, ξεκινώντας από τον Τ9. Στη συνέχεια ενημερώστε αναλόγως τους καταχωρητές αφετηρίας, έτσι ώστε να διατηρούνται οι πραγματικές εξαρτήσεις δεδομένων. Παρουσιάστε τον τελικό κώδικα (*Συμβουλή:* Δείτε το Σχήμα 2.37).

2.8 [20]<2.4> Στην Άσκηση 2.7 διερευνήσαμε την απλή μετονομασία καταχωρητών: όταν το υλικό μετονομασίας καταχωρητών συναντά κάποιον καταχωρητή αφετηρίας, τον αντικαθιστά με τον καταχωρητή Τ προορισμού της τελευταίας εντολής που είχε ως προορισμό αυτόν τον καταχωρητή αφετηρίας. Όταν ο πίνακας μετονομασίας συναντά έναν καταχωρητή προορισμού, τον αντικαθιστά με τον επόμενο διαθέσιμο για αυτόν καταχωρητή Τ. Ωστόσο, οι υπερβαθμωτές σχεδιάσεις χρειάζεται να χειρίζονται πολλαπλές εντολές ανά κύκλο ρολογιού σε κάθε στάδιο της μηχανής, συμπεριλαμβανομένης και της μετονομασίας καταχωρητών. Επομένως, οι απλοί βαθμωτοί επεξεργαστές θα εξέταζαν και τις δύο αντιστοιχίες των καταχωρητών αφετηρίας κάθε εντολής και θα ανέθεταν μία νέα αντιστοίχιση σε κάθε κύκλο ρολογιού. Οι υπερβαθμωτοί επεξεργαστές πρέπει, επίσης, να είναι σε θέση να εκτελούν την παραπάνω λειτουργία, ωστόσο πρέπει, συνάμα,

```
I0: LD     T9,0(Rx)
I1: MULTD  T10,F0,T9
...
```

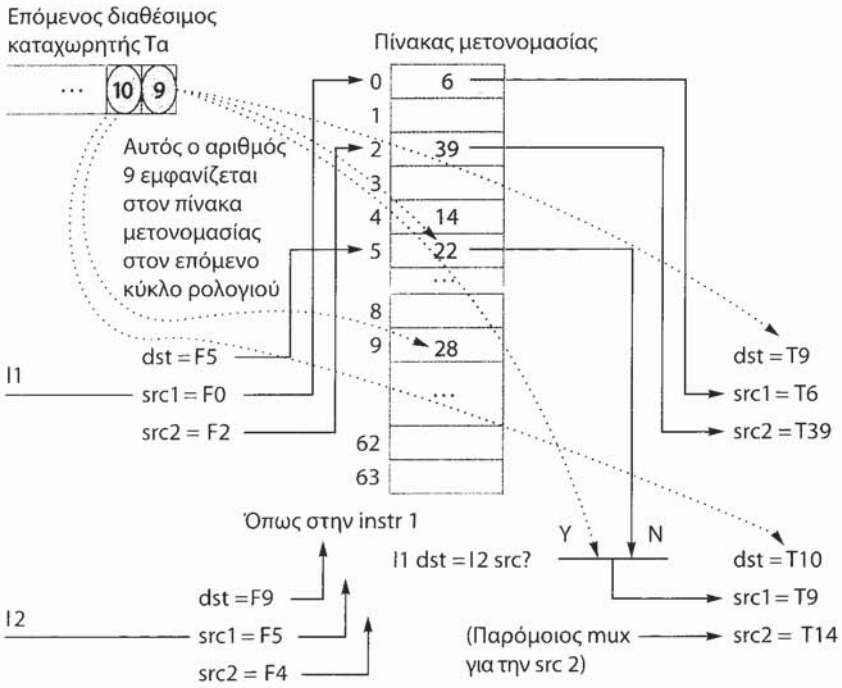
Σχήμα 2.37 **Αναμενόμενο αποτέλεσμα της μετονομασίας καταχωρητών.**

```
I0: MULTD F5,F0,F2
I1: ADDD  F9,F5,F4
I2: ADDD  F5,F5,F2
I3: DIVD  F2,F9,F0
```

Σχήμα 2.38 Ενδεικτικός κώδικας για την υπερβαθμωτή μετονομασία καταχωρητών.

να διασφαλίζουν ότι ο χειρισμός οποιωνδήποτε σχέσεων προορισμού προς αφετηρία ανάμεσα στις δύο ταυτοχρονικές (concurrent) εντολές πραγματοποιείται με ορθό τρόπο. Εξετάστε την ενδεικτική ακολουθία του Σχήματος 2.38. Θεωρήστε ότι θα θέλαμε να μετονομάσουμε ταυτόχρονα τις δύο πρώτες εντολές. Επιπλέον, υποθέστε ότι οι επόμενοι δύο διαθέσιμοι προς χρήση καταχωρητές Τ είναι γνωστοί κατά την έναρξη του κύκλου ρολογιού, εντός του οποίου μετονομάζονται οι δύο αυτές εντολές. Αυτό, λοιπόν, που θέλουμε είναι η πρώτη εντολή να προβεί στις κατάλληλες αναζητήσεις στον πίνακα μετονομασίας και στη συνέχεια να ενημερώσει τον πίνακα σχετικά με τον καταχωρητή Τ του προορισμού της. Στη συνέχεια, η δεύτερη εντολή θα υλοποιήσει την ίδια ακριβώς λειτουργία και ως εκ τούτου ο χειρισμός οποιασδήποτε εξάρτησης εντολών πραγματοποιείται με ορθό τρόπο. Παρ' όλα αυτά, δεν υπάρχει αρκετός χρόνος, εντός του ίδιου κύκλου ρολογιού, για την εγγραφή του ορισμού του καταχωρητή Τ στον πίνακα μετονομασίας και στη συνέχεια για την εκ νέου εξέτασή του για τις ανάγκες της δεύτερης εντολής. Αυτή η αντικατάσταση καταχωρητή πρέπει, αντιθέτως να πραγματοποιηθεί ζωντανά (παράλληλα με την ενημέρωση του πίνακα μετονομασίας καταχωρητών). Το Σχήμα 2.39 παρουσιάζει διάγραμμα κυκλωμάτων, χρησιμοποιώντας πολυπλέκτες και συγκριτές, το οποίο επιτυγχάνει την απαραίτητη ζωντανή μετονομασία των καταχωρητών. Στόχος σας είναι να παρουσιάσετε την κατάσταση του πίνακα μετονομασίας, ανά κύκλο ρολογιού, για κάθε εντολή του κώδικα. Θεωρήστε ότι όλες οι καταχωρίσεις του πίνακα αρχικά έχουν τιμή ίση με το δείκτη τους (Τ0 = 0, Τ1 = 1, . . .).

2.9 [5]<2.4> Αν σε κάποια στιγμή δεν σας είναι ξεκάθαρες οι ενέργειες που πρέπει να πραγματοποιήσει το υλικό μετονομασίας καταχωρητών, ανατρέξτε στο συμβολικό κώδικα (assembly code) που εκτελείτε και αναρωτηθείτε τι πρέπει να συμβεί προκειμένου να οδηγηθείτε στο ορθό αποτέλεσμα. Για παράδειγμα, εξετάστε την υπερβαθμωτή μηχανή που μετονομάζει ταυτοχρονικά τις τρεις αυτές εντολές:



Σχήμα 2.39 **Πίνακας μετονομασίας και λογική ζωντανής αντικατάστασης καταχωρητών για τις υπερβαθμωτές μηχανές (Σημείωση: το "src" συμβολίζει την αφετηρία και το "dst" τον προορισμό).**

```
ADDI R1, R1, R1
ADDI R1, R1, R1
ADDI R1, R1, R1
```

Αν η τιμή του R1 αρχικά ισούται με 5, ποια πρέπει να είναι η τιμή του μετά την εκτέλεση της ακολουθίας αυτής;

2.10 [20]<2.4, 2.9> Οι σχεδιαστές του VLIW πρέπει να υλοποιήσουν μερικές βασικές επιλογές για τους κανόνες αρχιτεκτονικής όσον αφορά τη χρήση των καταχωρητών. Θεωρήστε ότι ο VLIW έχει σχεδιαστεί ώστε να διαθέτει διασωληνώσεις εκτέλεσης που αδειάζουν μόνες τους: μόλις ξεκινήσει κάποια λειτουργία, τα αποτελέσματά της θα βρίσκονται στον καταχωρητή προορισμού το πολύ L κύκλους αργότερα (όπου το L συμβολίζει το λανθάνοντα χρόνο της λειτουργίας). Ποτέ δεν υπάρχουν αρκετοί καταχωρητές, επομένως αποτελεί δέλεαρ η επίτευξη της μέγιστης δυνατής χρήσης των καταχωρητών που υπάρχουν. Εξετάστε το Σχήμα 2.40. Με την προϋπόθεση ότι οι εντολές φόρτωσης έχουν λανθάνοντα χρόνο ίσο με $1 + 2$ κύκλους, ξεδιπλώστε μία φορά το βρόχο αυτό και παρουσιάστε τον τρόπο με τον

```
Loop: LW    R1,0(R2) ;  LW    R3,8(R2)
      <stall>
      <stall>
      ADDI  R10,R1,#1;  ADDI  R11,R3,#1
      SW    R1,0(R2) ;  SW    R3,8(R2)
      ADDI  R2,R2,#8
      SUB   R4,R3,R2
      BNZ   R4,Loop
```

Σχήμα 2.40 **Ενδεικτικός κώδικας του VLIW που περιλαμβάνει δύο εντολές πρόσθεσης, δύο φόρτωσης και δύο διαστήματα αδράνειας.**

οποίο ένας VLIW, ικανός να εκτελεί δύο εντολές φόρτωσης και δύο προσθέσεις σε κάθε κύκλο, μπορεί να χρησιμοποιήσει τον ελάχιστο δυνατό αριθμό καταχωρητών, έχοντας ως δεδομένο ότι η διασωλήνωση δεν διαθέτει διακοπές ή διαστήματα αδράνειας. Παραθέστε ένα ενδεικτικό γεγονός, το οποίο θα μπορούσε, χρησιμοποιώντας διασωληνώσεις που αδειάζουν μόνες τους, να διαταράξει την ομαλή λειτουργία της διαδικασίας διασωλήνωσης και να φέρει λανθασμένα αποτελέσματα.

2.11 [10/10/10]<2.3> Υποθέστε ότι έχετε στη διάθεσή σας μικροαρχιτεκτονική μονής διασωλήνωσης πέντε σταδίων (ανάκτησης, αποκωδικοποίησης, εκτέλεσης, μνήμης, επανεγγραφής), καθώς και τον κώδικα του Σχήματος 2.41. Όλες οι ops χρειάζονται 1 κύκλο, εκτός από τις LW και SW, οι οποίες χρειάζονται $1 + 2$ κύκλους και τις διακλαδώσεις, οι οποίες χρειάζονται $1 + 1$ κύκλους. Δεν υπάρχει η δυνατότητα της προώθησης. Παρουσιάστε τις φάσεις κάθε εντολής ανά κύκλο ρολογιού για μία επανάληψη του βρόχου.

α. [10]<2.3> Πόσοι κύκλοι ρολογιού ανά επανάληψη βρόχου δαπανώνται εξαιτίας της επιβάρυνσης της διακλάδωσης;

β. [10]<2.3> Υποθέστε ότι έχετε στη διάθεσή σας ένα στατικό μηχανισμό πρόβλεψης διακλάδωσης, που είναι ικανός να αναγνωρίζει διακλαδώσεις οπισθοδρόμησης (backwards branches) κατά το στάδιο αποκωδικοποίησης. Πλέον, πόσοι κύκλοι ρολογιού δαπανώνται εξαιτίας της επιβάρυνσης της διακλάδωσης;

γ. [10]<2.3> Υποθέστε ότι έχετε στη διάθεσή σας ένα δυναμικό μηχανισμό πρόβλεψης διακλάδωσης. Πόσοι κύκλοι δαπανώνται για την ορθή πρόβλεψη;

```
Loop: LW    R1,0(R2)
      ADDI  R1,R1,#1
      SW    R1,0(R2)
      ADDI  R2,R2,#4
      SUB   R4,R3,R2
      BNZ   R4,Loop
```

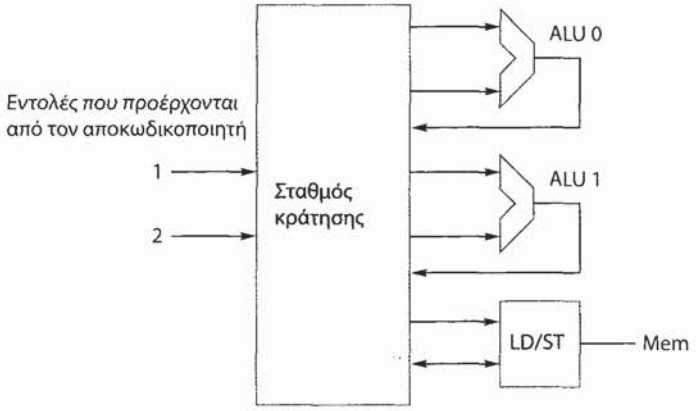
Σχήμα 2.41 **Κώδικας βρόχου για την Άσκηση 2.11.**

2.12 [20/20/20/10/20]<2.4, 2.7, 2.10> Στην άσκηση αυτή ας εξετάσουμε τι μπορεί να επιτευχθεί μέσω της δυναμικής χρονοδρομολόγησης. Υποθέστε ότι έχετε στη διάθεσή σας τη μικροαρχιτεκτονική που παρουσιάζεται στο Σχήμα 2.42. Θεωρήστε ότι όλες οι ALUs μπορούν να εκτελέσουν αριθμητικές ops (`MULTD, DIVD, ADDD, ADDI, SUB`) και διακλαδώσεις, καθώς και ότι ο Σταθμός Κράτησης (Reservation Station, RS) μπορεί να διαβιβάζει το πολύ μία λειτουργία σε κάθε λειτουργική μονάδα ανά κύκλο ρολογιού (μία op σε κάθε ALU, καθώς και μία op μνήμης στη μονάδα LD/ST).

α. [15]<2.4> Θεωρήστε ότι όλες οι εντολές της ακολουθίας του Σχήματος 2.35 βρίσκονται εντός του RS, χωρίς να έχει εκτελεστεί καμία ενέργεια μετονομασίας. Υπογραμμίστε τις εντολές του κώδικα στις οποίες η εφαρμογή της μετονομασίας των καταχωρητών θα οδηγούσε σε βελτίωση της απόδοσης. *Συμβουλή:* Αναζητήστε κινδύνους RAW και WAW. Θεωρήστε ότι ισχύουν οι ίδιοι λανθάνοντες χρόνοι με αυτούς του Σχήματος 2.35.

β. [20]<2.4> Θεωρήστε ότι εντός του RS, κατά τον κύκλο ρολογιού N, βρίσκεται η έκδοση του κώδικα του υποερωτήματος (α) αφότου έχει εφαρμοστεί η μετονομασία καταχωρητών, χρησιμοποιώντας τους λανθάνοντες χρόνους του Σχήματος 2.35. Παρουσιάστε τον τρόπο με τον οποίο ο RS πρέπει, ανά κύκλο ρολογιού, να διαβιβάσει τις εντολές αυτές εκτός σειράς, έτσι ώστε ο κώδικας αυτός να αποφέρει τη βέλτιστη απόδοση. (Θεωρήστε ότι ισχύουν οι ίδιοι περιορισμοί όσον αφορά τον RS που ίσχυαν στο υποερώτημα (α). Επίσης, υποθέστε ότι τα αποτελέσματα πρέπει να έχουν εγγραφεί στον RS προτού να είναι διαθέσιμα για χρήση, δηλ. δεν επιτρέπεται η παράκαμψη.) Πόσους κύκλους ρολογιού θα χρειαστεί η ακολουθία του κώδικα;

γ. [20]<2.4> Το υποερώτημα (β) επιτρέπει στον RS να επιχειρήσει τη βέλτιστη χρονοδρομολόγηση των εντολών αυτών. Ωστόσο, στην πραγ-



Σχήμα 2.42 **Ενδεικτικό μορφότυπο του αρχείου εισόδου history.txt.**

ματικότητα, δεν βρισκεται συνήθως εντός του RS ολόκληρη η ακολουθία κώδικα που μας ενδιαφέρει. Αντιθέτως, διάφορα γεγονότα οδηγούν στην εκκαθάριση του RS και καθώς μία νέα ακολουθία κώδικα εισέρχεται στον RS από τον αποκωδικοποιητή, ο RS πρέπει να επιλέξει να διαβιβάσει ό,τι έχει στη διάθεσή του. Υποθέστε ότι ο RS είναι κενός. Οι δύο πρώτες εντολές στις οποίες έχει εφαρμοστεί μετονομασία καταχωρητών εμφανίζονται στον RS στον κύκλο 0. Θεωρήστε ότι χρειάζεται η παρέλευση ενός κύκλου ρολογιού για τη διαβίβαση οποιασδήποτε op και υποθέστε ότι οι λανθάνοντες χρόνοι των λειτουργικών μονάδων ισούνται με αυτούς της Άσκησης 2.2. Επιπλέον, θεωρήστε το υλικό των αρχικών σταδίων (αποκωδικοποιητής/υλικό μετονομασίας καταχωρητών) εξακολουθεί να παρέχει δύο νέες εντολές ανά κύκλο ρολογιού. Παρουσιάστε τη σειρά της διαβίβασης των εντολών του RS ανά κύκλο ρολογιού. Πόσους κύκλους ρολογιού απαιτεί πλέον αυτή η ακολουθία κώδικα;

δ. [10]<2.10> Αν θέλατε να βελτιώσετε τα αποτελέσματα του υποερωτήματος (γ), ποια από τις παρακάτω επιλογές θα ήταν η καλύτερη: (1) άλλη μία ALU, (2) άλλη μία μονάδα LD/ST, (3) πλήρης παράκαμψη των αποτελεσμάτων της ALU σε επακόλουθες λειτουργίες, (4) μείωση κατά το ήμισυ του υψηλότερου λανθάνοντος χρόνου; Ποια είναι η επιτάχυνση;

ε. [20]<2.7> Ας εξετάσουμε, τώρα, την εικασία, τη διαδικασία ανάκτησης, αποκωδικοποίησης και εκτέλεσης περισσότερων από μία διακλαδώσεις υπό συνθήκη. Το κίνητρό μας έχει διττή υπόσταση: η χρονοδρομολόγηση της διαβίβασης που εφαρμόσαμε στο υποερώτημα (γ) διέθετε αρκετές nops, ενώ, συγχρόνως, γνωρίζουμε ότι οι υπολογιστές δαπανούν το μεγαλύτερο μέρος του χρόνου τους εκτελώντας

βρόχους (κάτι που συνεπάγεται ότι η διακλάδωση που βρίσκεται στην κορυφή του βρόχου είναι αρκετά προβλέψιμη). Οι βρόχοι μάς ενημερώνουν σχετικά με το πού μπορούν να προκύψουν περισσότερες εργασίες προς εκτέλεση. Η αραιή χρονοδρομολόγηση της διαβίβασης υποδηλώνει ότι υπάρχει η ευκαιρία να εκτελεστεί ένα μέρος των εργασιών νωρίτερα απ' ό,τι προηγουμένως. Στο υποερώτημα (δ) εντοπίσατε την κρίσιμη διαδρομή μέσω του βρόχου. Υποθέστε ότι έχετε στη διάθεσή σας δεύτερο αντίγραφο της διαδρομής που προέκυψε από τη χρονοδρομολόγηση του υποερωτήματος (β). Πόσοι περισσότεροι κύκλοι ρολογιού θα απαιτούνταν για να εκτελεστεί εργασία που αντιστοιχεί σε δύο βρόχους (με την προϋπόθεση ότι οι εντολές βρίσκονται εντός του RS); (Θεωρήστε ότι όλες οι λειτουργικές μονάδες υποστηρίζουν πλήρως τη διασωλήνωση.)

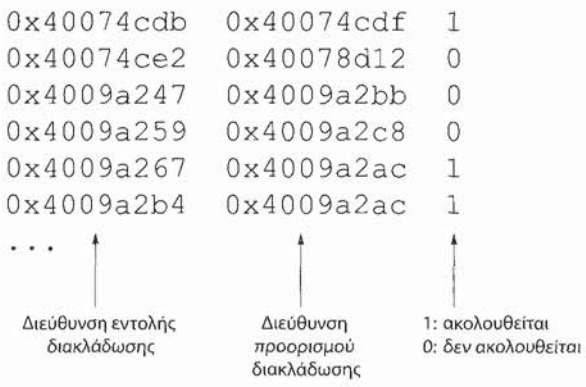
Μελέτη Περίπτωσης 2: Μοντελοποίηση Μηχανισμού Πρόβλεψης Διακλάδωσης

Έννοιες που αναλύονται σε αυτήν τη μελέτη περίπτωσης

- Μοντελοποίηση Μηχανισμού Πρόβλεψης Διακλάδωσης

Η πραγματική κατανόηση της αρχιτεκτονικής των υπολογιστών προϋποθέτει, πέρα από την εξέταση των τεχνικών σε επίπεδο μικροαρχιτεκονικής, τον προγραμματισμό των υπολογιστών. Ωστόσο, η άμεση ενασχόληση με τη μοντελοποίηση διάφορων ιδεών σε επίπεδο μικροαρχιτεκονικής αποτελεί καλύτερη λύση. Γράψτε ένα πρόγραμμα σε C ή Java για τη μοντελοποίηση ενός μηχανισμού πρόβλεψης διακλάδωσης (2,1). Το πρόγραμμά σας θα διαβάζει μια ακολουθία γραμμών από αρχείο με ονομασία history.txt (το οποίο διατίθεται στο συνοδευτικό CD - δείτε το Σχήμα 2.43).

Κάθε γραμμή του αρχείου αυτού διαθέτει τρία στοιχεία δεδομένων, τα οποία είναι διαχωρισμένα μέσω σιηλοθέτη (tab). Η πρώτη στήλη κάθε γραμμής περιλαμβάνει δεδομένα τα οποία αφορούν τη διεύθυνση της εντολής διακλάδωσης σε δεκαεξαδική μορφή. Η δεύτερη στήλη αντιπροσωπεύει τη διεύθυνση προορισμού διακλάδωσης σε δεκαεξαδική μορφή. Η τρίτη στήλη περιλαμβάνει είτε την τιμή 1 είτε την τιμή 0. Η μονάδα υποδηλώνει διακλάδωση που ακολουθείται, ενώ το μηδέν υποδηλώνει διακλάδωση που δεν ακολουθείται. Ο συνολικός αριθμός των διακλαδώσεων τις οποίες το μοντέλο σας εξετάζει είναι, βεβαίως, ίσος με το πλήθος των γραμμών του αρχείου. Υποθέστε ότι έχετε στη διάθεσή σας απομονωτή προορισμού διακλάδωσης (branch target buffer, BTB) με απευθείας αντιστοίχιση και μην προβληματίζεστε για ζητήματα που αφορούν το μήκος ή τη στοίχιση των εντολών (δηλ. αν ο BTB διαθέτει τέσσερις καταχωρίσεις, τότε οι εντολές διακλάδωσης των διευθύνσεων 0x0, 0x1, 0x2 και 0x3 θα βρίσκονται στις



Σχήμα **2.43 Ενδεικτικό μορφότυπο της εξόδου.**

τέσσερις αυτές καταχωρίσεις, ωστόσο η διεύθυνση διακλάδωσης της διεύθυνσης 0x4 θα αντικαταστήσει το πεδίο BTB[0]). Για κάθε γραμμή του αρχείου εισόδου, το μοντέλο σας θα διαβάζει το ζεύγος των τιμών των δεδομένων, θα προσαρμόζει τους διάφορους πίνακες του μηχανισμού πρόβλεψης διακλάδωσης που μοντελοποιείται και θα συλλέγει τα πιο σημαντικά στατιστικά της απόδοσης. Η τελική έξοδος του προγράμματός σας θα έχει μορφή παρόμοια με αυτήν του Σχήματος 2.44.

Το μοντέλο σας πρέπει να δίνει τη δυνατότητα επιλογής του αριθμού των καταχωρίσεων του BTB μέσω της γραμμής εντολών.

2.13 [20/10/10/10/10/10/10]<2.3> Γράψτε το μοντέλο ενός απλού απομονωτή προορισμού διακλάδωσης των τεσσάρων καταστάσεων που διαθέτει 64 καταχωρίσεις.

α. [20]<2.3> Ποιος είναι ο συνολικός ρυθμός ευστοχίας του BTB (το ποσοστό των επιτυχημένων προσπαθειών εύρεσης μιας διακλάδωσης εντός του BTB);

β. [10]<2.3> Ποιος είναι ο συνολικός ρυθμός εσφαλμένων προβλέψεων διακλάδωσης κατά την ψυχρή εκκίνηση (το ποσοστό των επιτυχημένων προσπαθειών πρόβλεψης ότι κάποια διακλάδωση είτε έχει είτε δεν έχει ακολουθηθεί, ανεξαρτήτως του αν η πρόβλεψη αυτή «ανήκε» στη διακλάδωση που αποτελούσε το αντικείμενο της πρόβλεψης);

γ. [10]<2.3> Εντοπίστε την πιο συνήθη διακλάδωση. Ποια ήταν η συνεισφορά της στο συνολικό αριθμό των ορθών προβλέψεων; (*Συμβουλή:* Μειρήστε το πλήθος των περιπτώσεων που εμφανίζεται η διακλάδωση στο αρχείο history.txt και στη συνέχεια παρακολουθήστε τον τρόπο συμπεριφοράς της διακλάδωσης αυτής εντός του μοντέλου BTB).

Άσκηση 2.13 (α)
Αριθμός ευστοχιών του BTB: 54390. Συνολικές διακ.: 55493. Ρυθμός ευστοχίας: 99.8%

Άσκηση 2.13 (β)
Εσφαλμένες προβλέψεις: 1562 από τις 55493 ή 2.8%

Άσκηση 2.13 (γ)
<ένα απλό αρχείο σεναρίου με εντολές της γραμμής εντολών του unix θα επιστρέψει την πιο συνήθη διακλάδωση...παρουσιάστε τον τρόπο με τον οποίο το επιτύχατε αυτό>
Η πιο σημαντ. διακλάδωση συναντήθηκε 15418 από τις 55493 συν. διακ., 27.8%
Διακλάδωση MS = 0x80484ef, ορθές προβλέψεις = 19151 (από τις 36342 συνολικά ορθές προβ.) ή 52.7%

Άσκηση 2.13 (δ)
Συνολικές μοναδικές διακλαδώσεις (1 υποχρεωτική αστοχία ανά διακ.): 121
Συνολικός αριθμός αστοχιών: 104
Άρα, συνολική χωρητικότητα αστοχιών = σύνολο αστοχιών – υποχρεωτικές αστοχίες = 17

Άσκηση 2.13 (ε)
Αριθμός ευστοχιών του BTB: 54390. Συνολικές διακ.: 55493. Ρυθμός ευστοχίας: 99.8%
Εσφαλμένες προβλέψεις: 1103 από τις 54493 ή 2%

Άσκηση 2.13 (στ)

Μήκος του BTB	ρυθμός εσφαλμένων προβλέψεων
1	32.91%
2	6.42%
4	0.28%
8	0.23%
16	0.21%
32	0.20%
64	0.20%

Σχήμα 2.44 Ενδεικτικό μορφότυπο της εξόδου του προγράμματος.

δ. [10]<2.3> Πόσες αστοχίες χωρητικότητας σημειώνονται στο μηχανισμό πρόβλεψης διακλάδωσης;

ε. [10]<2.3> Ποια είναι η επίδραση της ψυχρής εκκίνησης (cold start) σε σύγκριση με τη θερμή εκκίνηση (warm start); Για τον υπολογισμό του παραπάνω μεγέθους, εκτελέστε μία φορά το ίδιο σύνολο δεδομένων για την αρχικοποίηση του πίνακα ιστορικού και στη συνέχεια συλλέξτε εκ νέου το ίδιο σύνολο στατιστικών δεδομένων.

στ. [10]<2.3> Προβείτε τέσσερις περισσότερες φορές στην ψυχρή εκκίνηση του BTB, έχοντας BTBs με μεγέθη 16, 32 και 64. Παρουσιάστε με τη βοήθεια γραφήματος τους πέντε ρυθμούς εσφαλμένων προβλέψεων. Παρουσιάστε, επίσης, με τη βοήθεια γραφήματος τουω πέντε ρυθμούς ευστοχίας.

ζ. [10] Παραδώστε σε καλογραμμένη μορφή τον κώδικά σας, εισαγάγοντας τα κατάλληλα σχόλια όπου είναι απαραίτητο, για το μοντέλο του απομονωτή προορισμού διακλάδωσης.