Πανεπιστήμιο Θεσσαλίας
Τμήμα Πληροφορικής

Αρχιτεκτονική Υπολογιστών

Λυμένες Ασκήσεις

Άσκηση 7:

Έστω ένας επεξεργαστής MIPS αρχιτεκτονικής VLIW, ο οποίος στη λέξη εντολής του κωδικοποιεί πέντε απλές εντολές MIPS, ως εξής: δύο εντολές προσπέλασης μνήμης, δύο εντολές κινητής υποδιαστολής, και μία εντολή σταθερής υποδιαστολής χωρίς προσπέλαση μνήμης. Για την εκτέλεση των εντολών VLIW, η ΜΕΔ του επεξεργαστή αυτού περιλαμβάνει δύο υπομονάδες υπολογισμού τελικής διεύθυνσης προσπέλασης μνήμης, δύο υπομονάδες πρόσθεσης/αφαίρεσης/ πολλαπλασιασμού κινητής υποδιαστολής, μία υπομονάδα λοιπών ακέραιων λογικών/αριθμητικών πράξεων, καθώς και μία κρυφή μνήμη δεδομένων, δύο προσπελάσεων ανά κύκλο μηχανής, που σε επιτυχία ολοκληρώνονται σε έναν κύκλο μετά τον υπολογισμό της τελικής διεύθυνσης προσπέλασης. Ο υπολογισμός της τελικής διεύθυνσης προσπέλασης για εντολές προσπέλασης μνήμης, καθώς και όλες οι υπόλοιπες ακέραιες πράξεις απαιτούν έναν κύκλο μηχανής για την εκτέλεσή τους, ενώ οι υπομονάδες κινητής υποδιαστολής εκτελούν πρόσθεση/αφαίρεση σε δύο και πολλαπλασιασμό σε τέσσερις κύκλους μηχανής με μερική επικάλυψη διαδοχικών πράξεων. Η ΜΕΔ του επεξεργαστή περιλαμβάνει τέλος και ένα φάκελο με 32 καταχωρητές σταθερής και 32 καταχωρητές κινητής υποδιαστολής, με όσες θύρες ανάγνωσης/εγγραφής απαιτούνται για εξυπηρέτηση μιας εντολής VLIW ανά κύκλο μηχανής.
Αν ο παραπάνω επεξεργαστής, ακολουθώντας την αρχιτεκτονική VLIW, δεν ανιχνεύει εξαρτήσεις από δεδομένα μεταξύ των επιμέρους εντολών MIPS της κάθε εντολής VLIW που εκτελεί, δώστε μια κωδικοποίηση εντολών VLIW για τον πιο κάτω βρόχο απλών εντολών MIPS, έτσι ώστε η δρομολόγηση των εντολών VLIW να γίνεται στον ελάχιστο χρόνο χωρίς κανένα χαμένο κύκλο μηχανής:

```
loop: ldc1  $f0,0($1)        (S1)
      ldc1  $f1,0($2)        (S2)
      add.d $f2,$f0,$f1      (S3)
      mul.d $f2,$f2,$f0      (S4)
      sub.d $f3,$f2,$f4      (S5)
      sdc1  $f3,0($3)        (S6)
      addiu $1,$1,8          (S7)
      addiu $2,$2,8          (S8)
      addiu $3,$3,8          (S9)
      bne   $1,$4,loop       (S10)
```

ξεδιπλώνοντας το βρόχο όσες φορές κρίνετε απαραίτητο. Βρείτε το μέσο ρυθμό ολοκλήρωσης των επαναλήψεων του αρχικού βρόχου στον παραπάνω επεξεργαστή, και συγκρίνετε με το μέσο ρυθμό ολοκλήρωσης επαναλήψεων του βρόχου σε ένα βαθμωτό επεξεργαστή MIPS με στατική δρομολόγηση εντολών. Ποιο είναι το ποσοστό πλήρωσης θέσεων στις εντολές VLIW που δώσατε; Στη συνέχεια προσπαθήστε να βρείτε μια νέα κωδικοποίηση εντολών VLIW για τον πιο πάνω βρόχο, έτσι ώστε να επιτυγχάνετε και τη μεγαλύτερη δυνατή πλήρωση θέσεων στις εντολές VLIW. Βρείτε ένα άνω φράγμα στο ποσοστό πλήρωσης και εξηγήστε πώς μπορείτε να το επιτύχετε. Ξεδιπλώστε το βρόχο όσο χρειάζεται, θεωρώντας ότι ο φάκελος καταχωρητών διαθέτει απεριόριστους καταχωρητές. Ποιος είναι ο ρυθμός ολοκλήρωσης των επαναλήψεων του αρχικού βρόχου στην περίπτωση αυτή;
Και για τις δύο πιο πάνω περιπτώσεις υποθέστε επιτυχημένη πρόβλεψη διακλαδώσεων με εκτέλεση χωρίς καθυστέρηση.

Απάντηση

Ξεκινώντας από τον κώδικα MIPS που μας δίνεται, θα δημιουργήσουμε VLIW εντολές, τοποθετώντας εντολές MIPS στις θέσεις που αντιστοιχούν στις επιμέρους εντολές των VLIW εντολών, έτσι ώστε να μην παραβιάζονται οι εξαρτήσεις που υπάρχουν στον αρχικό κώδικα. Ειδικότερα, επειδή ο επεξεργαστής μας δεν εξετάζει εξαρτήσεις από δεδομένα μεταξύ των επιμέρους εντολών MIPS μιας VLIW εντολής, θα πρέπει να είμαστε προσεκτικοί στην κωδικοποίηση των εντολών VLIW και να μην τοποθετούμε σε αυτές εντολές MIPS που εμφανίζουν μεταξύ τους εξαρτήσεις τύπου ΑΜΕ, ή να μετονομάζουμε τους καταχωρητές που εμπλέκονται σε εξαρτήσεις, αν αυτές είναι τύπου ΕΜΑ και ΕΜΕ.
Όσο αφορά κινδύνους από εξαρτήσεις δεδομένων μεταξύ διαφορετικών εντολών VLIW, υποθέτουμε ότι η ΜΕΔ διαθέτει μηχανισμό ανίχνευσης εξαρτήσεων τύπου ΑΜΕ, και κατ' επέκταση και μηχανισμό παροχέτευσης για όλες τις υπομονάδες εκτέλεσης. Με το μηχανισμό παροχέτευσης, κάθε εντολή MIPS μπορεί να ξεκινήσει την εκτέλεσή της τουλάχιστον τόσους κύκλους αργότερα από μια εντολή από την οποία έχει εξάρτηση δεδομένων, όσους απαιτεί η δεύτερη για την εκτέλεσή της. Ειδικότερα, μια εντολή αποθήκευσης μπορεί να ξεκινήσει έναν κύκλο μηχανής νωρίτερα από τον κύκλο στον οποίο παράγεται το δεδομένο που αποθηκεύει, επειδή ο μηχανισμός παροχέτευσης στέλνει το δεδομένο αυτό κατ' ευθείαν στη μονάδα προσπέλασης μνήμης, και όχι στην υπομονάδα υπολογισμού τελικής διεύθυνσης προσπέλασης.
Εφ' όσον η δρομολόγηση των εντολών είναι στατική, αν μια εντολή MIPS βρίσκεται σε μικρότερη απόσταση εντολών VLIW από την εντολή από την οποία αυτή εξαρτάται, η ΜΕΔ του επεξεργαστή παγώνει όσο χρόνο είναι απαραίτητο, ώστε να συμπληρωθεί ο αριθμός κύκλων που πρέπει να μεσολαβήσει μεταξύ των δύο εντολών. Για να μην έχουμε απώλεια χρόνου στη δρομολόγηση κάποιου κώδικα, δε θα πρέπει επομένως να τοποθετούμε εντολές MIPS που εμφανίζουν εξαρτήσεις τύπου ΑΜΕ σε εντολές VLIW, με τρόπο που να δημιουργείται πάγωμα στη ΜΕΔ κατά τη δρομολόγησή τους. Δε θα πρέπει όμως να τοποθετούμε τέτοιες εντολές ούτε σε εντολές VLIW που βρίσκονται σε απόσταση μεγαλύτερη από τον ελάχιστο αριθμό κύκλων, αν αυτό επιμηκύνει το χρόνο δρομολόγησης του κώδικα.
Για απλούστευση θα αγνοήσουμε τις δομικές εξαρτήσεις στην εγγραφή του φακέλου καταχωρητών από εντολές κινητής υποδιαστολής που έχουν διαφορετικούς χρόνους εκτέλεσης. Έτσι, θα υποθέσουμε ότι δύο εντολές που ολοκληρώνουν την εκτέλεσή τους στον ίδιο κύκλο και προέρχονται από την ίδια θέση της εντολής VLIW, έχοντας όμως ξεκινήσει από διαφορετικές εντολές VLIW, μπορούν να γράψουν το αποτέλεσμά τους ταυτόχρονα στο φάκελο καταχωρητών. Θα πρέπει όμως να προσέχουμε για εξαρτήσεις τύπου ΕΜΕ που θα μπορούσαν να εμφανιστούν λόγω των διαφορετικών χρόνων εκτέλεσης των εντολών κινητής υποδιαστολής, και να μην κατασκευάζουμε εντολές VLIW με κινδύνους από τέτοιες εξαρτήσεις.

Για να βρούμε μια κωδικοποίηση του δεδομένου βρόχου σε εντολές VLIW που να οδηγεί σε εκτέλεση χωρίς απώλεια κύκλων μηχανής, θα πρέπει (α) να εξασφαλίσουμε ότι η κωδικοποίηση δε θα οδηγήσει σε πάγωμα της ΜΕΔ για αντιμετώπιση εξαρτήσεων από δεδομένα τύπου ΑΜΕ, αλλά ούτε και σε καθυστέρηση στη δρομολόγηση των επιμέρους εντολών, και (β) να ξεδιπλώσουμε το βρόχο όσο χρειάζεται, ώστε να έχουμε διαθέσιμες τις εντολές MIPS που χρειαζόμαστε για την κωδικοποίηση των εντολών VLIW.
Έτσι, για να επιτύχουμε τον πρώτο στόχο, θα πρέπει να ξεκινήσουμε με εντολές VLIW, που αν περιέχουν επιμέρους εντολές MIPS με κινδύνους από εξαρτήσεις τύπου ΑΜΕ, να βρίσκονται σε ικανή απόσταση μεταξύ τους, σύμφωνα με όσα αναφέραμε πιο πάνω. Επιτρέποντας κατ' αρχήν κενές εντολές VLIW, τις οποίες θα συμπληρώσουμε αργότερα προς επίτευξη του δεύτερου στόχου, σχηματίζουμε την ακόλουθη διάταξη εντολών VLIW:

VLIWloop:	S1	S2			
			S3		
			S4		

		S5		
S6				S7
				S8
				S9
				S10

όπου κάθε γραμμή αντιστοιχεί σε μία εντολή VLIW, με τις επιμέρους εντολές MIPS να δίνονται με το συμβολικό όνομα που έχουν στην εκφώνηση, και να καταλαμβάνουν θέσεις ως εξής: εντολές προσπέλασης μνήμης στις δύο πρώτες στήλες, εντολές κινητής υποδιαστολής στις δύο επόμενες στήλες και εντολές διακλάδωσης και λοιπών αριθμητικών/λογικών πράξεων στην τελευταία στήλη.
Παρατηρούμε ότι σε δύο περιπτώσεις, εντολές MIPS που είναι ανεξάρτητες μεταξύ τους τοποθετήθηκαν στην ίδια εντολή VLIW. Από την άλλη μεριά, εντολές MIPS που εμφανίζουν εξαρτήσεις από δεδομένα τύπου AME τοποθετήθηκαν σε κατάλληλη απόσταση μεταξύ τους. Για παράδειγμα, οι εντολές MIPS S3 και S4 έχουν μια κενή εντολή VLIW μεταξύ τους, εφ' όσον η δεύτερη μπορεί να ξεκινήσει την εκτέλεσή της δύο κύκλους μηχανής μετά την πρώτη, και όχι νωρίτερα. Αν δεν υπήρχε η εντολή VLIW μεταξύ των S3 και S4, δε θα είχαμε λανθασμένο κώδικα, αλλά η δρομολόγηση των εντολών στον επεξεργαστή θα δημιουργούσε πάγω-μα στη ΜΕΔ, με αποτέλεσμα την απώλεια ενός κύκλου μηχανής, που για μεγάλο αριθμό επαναλήψεων του βρόχου γίνεται πολύ σημαντικός!

Για να ολοκληρώσουμε τον πρώτο στόχο, θα πρέπει να προσπαθήσουμε να συμπτύξουμε τον κώδικα, ώστε να πάρουμε τον ελάχιστο χρόνο δρομολόγησης κάθε επανάληψης του βρόχου. Ο χρόνος αυτός ταυτίζεται με το χρόνο που απαιτείται για την εκτέλεση μιας επανάληψης, και καθορίζεται από τις εξαρτήσεις από δεδομένα τύπου AME που υπάρχουν στο σώμα του βρόχου. Τέτοιες εξαρτήσεις είναι οι εξαρτήσεις από τις εντολές S1 και S2 προς την S3, από τις S1 και S3 προς την S4, από την S4 προς την S5, από την S5 προς την S6, καθώς και από την S7 προς την S10. Από αυτές, η τελευταία δε σχετίζεται με τις προηγούμενες. Έτσι, οι υπόλοιπες εξαρτήσεις δημιουργούν την αλυσίδα εντολών S1-S3-S4-S5-S6 που δρομολογείται σειριακά, δίνοντας τον ελάχιστο χρόνο των 10 κύκλων μηχανής για κάθε επανάληψη του βρόχου. Οι εντολές S7, S8 και S9 μπορούν να τοποθετηθούν πριν από την εντολή S6, προσέχοντας τις εξαρτήσεις τύπου EMA στους καταχωρητές $1, $2 και $3, οι κίνδυνοι από τις οποίες αποφεύγονται με αλλαγή στη μετατόπιση των εντολών προσπέλασης μνήμης, όπου αυτό χρειάζεται. Φυσικά η εντολή διακλάδωσης S10 θα πρέπει να βρίσκεται πάντα στο τέλος του βρόχου, αλλά μπορεί να βρίσκεται στην ίδια εντολή VLIW με την S6.
Σύμφωνα με τα παραπάνω, λαμβάνουμε τον εξής νέο κώδικα VLIW:

VLIWloop:	S1(0)	S2(0)			
			S3		
			S4		
					S7
					S8
					S9
			S5		
	S6(-8)				S10

όπου σε παρένθεση δείχνουμε τις μετατοπίσεις στις αντίστοιχες εντολές προσπέλασης μνήμης. Οι εντολές S7, S8 και S9 θα μπορούσαν να τοποθετηθούν οπουδήποτε στην τελευταία στήλη, εκτός από την τελευταία γραμμή, και επιλέξαμε γραμμές που πριν ήταν κενές. Παρατηρούμε ότι με τη νέα κωδικοποίηση παραμένουν δύο μόνο κενές εντολές VLIW.

Στη συνέχεια, για να επιτύχουμε το δεύτερο στόχο, δηλαδή την πλήρωση με τουλάχιστον μια εντολή MIPS όλων των εντολών VLIW, εφαρμόζουμε ξεδίπλωμα στον αρχικό βρόχο, και τοποθετούμε στις πιο πάνω εντολές VLIW τις εντολές MIPS που αντιστοιχούν στη δεύτερη επανάληψη του βρόχου. Επειδή η πρώτη εντολή VLIW δε διαθέτει χώρο για άλλες δύο εντολές φόρτωσης, οι δύο νέες εντολές φόρτωσης θα τοποθετηθούν αναγκαστικά στη δεύτερη εντολή VLIW, κι επομένως όλος ο κώδικας VLIW θα επιμηκυνθεί κατά μία εντολή:

VLIWloop:	S1(0)	S2(0)			
	S1(8)	S2(8)			
			S3		
			S3		
			S4		
			S4		S7
					S8
					S9
			S5		
	S6(-16)		S5		
	S6(-8)				S10

Προφανώς οι εντολές S7, S8, S9 θα πρέπει τώρα να μεταβάλλουν τα τελούμενά τους κατά το διπλάσιο της προηγούμενης μεταβολής, δηλαδή κατά 16. Η εντολή S10 παραμένει στο τέλος του βρόχου.
Επίσης, οι εντολές της δεύτερης επανάληψης θα πρέπει να χρησιμοποιούν διαφορετικούς καταχωρητές κινητής υποδιαστολής από τις εντολές της πρώτης επανάληψης, ώστε να αποφεύγονται παραβιάσεις των εξαρτήσεων τύπου EMA και EME που εμφανίζονται σαν αποτέλεσμα του ξεδιπλώματος. Αυτό είναι εφικτό, αφού κάθε επανάληψη χρησιμοποιεί μόνο τέσσερις καταχωρητές κινητής υποδιαστολής, με τον $f4 να είναι κοινός για όλες τις επαναλήψεις, και η αρχιτεκτονική μας διαθέτει 32 καταχωρητές κινητής υποδιαστολής.
Τώρα δεν έχουμε καμία κενή εντολή VLIW στον κώδικά μας, αλλά έχουμε πολύ χαμηλό ποσοστό πλήρωσης θέσεων, όπως θα δούμε πιο κάτω.

Ο κώδικας VLIW που βρήκαμε οδηγεί σε δρομολόγηση δύο επαναλήψεων του αρχικού βρόχου σε χρόνο 11 κύκλων μηχανής, απαιτεί δηλαδή κατά μέσο όρο 5,5 κύκλους μηχανής ανά επανάληψη. Αν είχαμε ένα βαθμωτό επεξεργαστή MIPS με στατική δρομολόγηση εντολών, θα απαιτούνταν 15 κύκλοι μηχανής ανά επανάληψη, χρόνος που προκύπτει από την αρχική σειρά των εντολών και τον ελάχιστο αριθμό κύκλων μηχανής που μεσολαβούν μεταξύ εξαρτημένων εντολών. Για τον υπολογισμό αυτό, λαμβάνουμε υπ' όψη μόνο τη φάση εκτέλεσης των εντολών, μια που ζητάμε το ρυθμό ολοκλήρωσης επαναλήψεων, και όχι το συνολικό χρόνο για μια επανάληψη από την προσκόμιση της πρώτης μέχρι την ολοκλήρωση της τελευταίας εντολής. Αν λοιπόν η πρώτη εντολή εκτελείται στον πρώτο κύκλο μηχανής, ο χρόνος εκτέλεσης των εντολών μιας επανάληψης θα είναι:

Εντολή:	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Χρόνος:	1	2	4	6	10	11	12	13	14	15

Αν όμως αναδιατάσσαμε τις εντολές σύμφωνα με όσα αναφέραμε νωρίτερα, θα μπορούσαμε να μειώσουμε το χρόνο σε 12 κύκλους:

Εντολή:	S1	S2	S3	S4	S7	S8	S9	S5	S6	S10
Χρόνος:	1	2	4	6	7	8	9	10	11	12

Τώρα, οι εντολές S7, S8 και S9 δρομολογούνται κατά τη διάρκεια εκτέλεσης του πολλαπλασιασμού κινητής υποδιαστολής της εντολής S4.

Βλέπουμε λοιπόν ότι με την κωδικοποίηση εντολών VLIW που δώσαμε, ο ρυθμός ολοκλήρωσης των επαναλήψεων του βρόχου υπερδιπλασιάστηκε. Θα μπορούσαμε όμως να πάρουμε και καλύτερο ρυθμό, δεδομένου ότι ο βρόχος είναι παράλληλος, ενώ το ποσοστό πλήρωσης θέσεων των εντολών VLIW είναι 16 θέσεις στις 55, δηλαδή μόλις 29%, λίγο παραπάνω από το ελάχιστο ποσοστό 20% που αντιστοιχεί σε κώδικα με μία ακριβώς συμπληρωμένη θέση για κάθε εντολή VLIW. Υψηλότερος ρυθμός ολοκλήρωσης επαναλήψεων επιτυγχάνεται με περαιτέρω ξεδίπλωμα του βρόχου.

Ένα άνω φράγμα στο ποσοστό πλήρωσης εντολών VLIW, αλλά και στο ρυθμό ολοκλήρωσης επαναλήψεων του αρχικού βρόχου, υπολογίζεται από το εύρος και τα πεδία της VLIW εντολής, σε συνδυασμό με τον αριθμό εντολών του σώματος, και αγνοώντας τις εντολές ελέγχου του βρόχου. Έτσι, με δεδομένο ότι τα πεδία της VLIW εντολής περιλαμβάνουν δύο επιμέρους εντολές προσπέλασης μνήμης, μπορούμε να έχουμε κωδικοποίηση μέχρι δύο επαναλήψεων ανά εντολή VLIW, αν το σώμα του βρόχου περιέχει μία εντολή προσπέλασης μνήμης, μέχρι μιας επανάληψης ανά εντολή VLIW, αν το σώμα του βρόχου περιέχει δύο εντολές προσπέλασης μνήμης, και γενικά μέχρι 2/N επαναλήψεις ανά εντολή VLIW, αν το σώμα του βρόχου περιέχει N εντολές προσπέλασης μνήμης. Το ίδιο άνω φράγμα στον αριθμό επαναλήψεων ανά εντολή VLIW υπολογίζεται και για εντολές κινητής υποδιαστολής, αφού τα πεδία της VLIW εντολής περιλαμβάνουν δύο τέτοιες επιμέρους εντολές. Όμως, αν το σώμα του βρόχου περιλαμβάνει και εντολές σταθερής υποδιαστολής εκτός προσπελάσεων μνήμης, τότε εφ' όσον τα πεδία της VLIW εντολής περιλαμβάνουν μόνο μία τέτοια επιμέρους εντολή, το άνω φράγμα μειώνεται στο μισό από το προηγούμενο, δηλαδή 1/N επαναλήψεις ανά εντολή VLIW, για N εντολές σταθερής υποδιαστολής εκτός προσπελάσεων μνήμης στο σώμα του βρόχου.
Στην περίπτωσή μας, το άνω φράγμα στον αριθμό επαναλήψεων που κωδικοποιούνται ανά εντολή VLIW υπολογίζεται στο 2/3, το οποίο σημαίνει ότι μπορούμε να κωδικοποιήσουμε μέχρι δύο επαναλήψεις σε κάθε τρεις εντολές VLIW. Η τιμή αυτή προκύπτει τόσο από τις εντολές προσπέλασης μνήμης, όσο και από τις εντολές κινητής υποδιαστολής, που είναι τρεις στο σώμα του βρόχου και για τις δύο κατηγορίες. Επομένως, για κ επαναλήψεις του αρχικού βρόχου, που αντιστοιχούν σε 6κ+4 εντολές MIPS, συμπεριλαμβανομένων των τεσσάρων εντολών ελέγχου του βρόχου που υπολογίζονται μία φορά, θέλουμε $\lceil 1{,}5\kappa \rceil$ εντολές VLIW, που διαθέτουν $5*\lceil 1{,}5\kappa \rceil$ θέσεις επιμέρους εντολών MIPS. Το μέγιστο ποσοστό πλήρωσης θέσεων θα είναι $(6\kappa+4)/(5*\lceil 1{,}5\kappa \rceil)$, που φτάνει στο 93,33% για $\kappa = 4$. Για μεγαλύτερο αριθμό επαναλήψεων το ποσοστό μειώνεται προς το 80%, λόγω των κενών θέσεων που προκύπτουν για τις εντολές σταθερής υποδιαστολής. Για $\kappa \leq 2$ ο αριθμός εντολών VLIW δεν είναι αρκετός για την τοποθέτηση των εντολών ελέγχου του βρόχου.
Επειδή κάθε εντολή VLIW δρομολογείται σε έναν κύκλο μηχανής, το άνω φράγμα στον αριθμό επαναλήψεων ανά εντολή VLIW αντιστοιχεί σε άνω φράγμα στο ρυθμό ολοκλήρωσης επαναλήψεων ίσο με δύο επαναλήψεις σε τρεις κύκλους μηχανής.

Επίτευξη του άνω φράγματος στο ποσοστό πλήρωσης θέσεων και ρυθμού ολοκλήρωσης επαναλήψεων μπορούμε να έχουμε, αν ο συνολικός αριθμός επαναλήψεων είναι αρκετά μεγάλος, ώστε τα κενά στη δρομολόγηση MIPS εντολών που οφείλονται στις εξαρτήσεις τύπου AME να περιορίζονται στις αρχικές επαναλήψεις. Βγάζοντας τις αρχικές επαναλήψεις έξω από το βρόχο, διατηρούμε μέσα σε αυτόν μόνο τον αριθμό επαναλήψεων που αντιστοιχεί στο άνω φράγμα, ενώ συμπληρώνουμε τον κώδικα με εντολές που ολοκληρώνουν το συνολικό αριθμό επαναλήψεων του αρχικού βρόχου. Με τον τρόπο αυτό λαμβάνουμε ένα νέο βρόχο που υλοποιεί ουσιαστικά συνδυασμό πραγματικού με συμβολικό ξεδίπλωμα του αρχικού βρόχου.
Για να τοποθετήσουμε τις εντολές MIPS στις VLIW εντολές με τον βέλτιστο τρόπο, μπορούμε να χρησιμοποιήσουμε την τεχνική που μάθαμε για βελτιστοποίηση στη μερική επικάλυψη μονάδων εκτέλεσης πράξεων. Έτσι, ξεκινώντας με τις εντολές VLIW που είχαμε παραπάνω, εισάγουμε καθυστερήσεις στη δρομολόγηση εντολών MIPS, όταν αυτό γίνεται απαραίτητο, ώστε οι κενές θέσεις που δημιουργούμε να καλύπτονται από άλλες εντολές MIPS επόμενων επαναλήψεων. Για παράδειγμα, η τοποθέτηση της εντολής S6 πρέπει να γίνεται 11 εντολές

VLIW μετά την πρώτη S1, και όχι 9 εντολές, όπως είχαμε πιο πάνω, ώστε να μεσολαβήσουν δύο εντολές VLIW, όπου θα τοποθετηθούν οι εντολές S1 και S2 επόμενων επαναλήψεων.

Η διάταξη εντολών VLIW που προκύπτει, και η οποία δίνει δρομολόγηση εντολών και πλήρωση θέσεων στο άνω φράγμα, είναι η ακόλουθη:

	S1(0)	S2(0)			
	S1(8)	S2(8)			
			S3		
	S1(16)	S2(16)	S3		
	S1(24)	S2(24)	S4		
			S4	S3	
	S1(32)	S2(32)		S3	
	S1(40)	S2(40)		S4	
			S3	S4	
VLIWloop:	S1(48)	S2(48)	S3	S5	
	S1(56)	S2(56)	S4	S5	
	S6(0)	S6(8)	S4	S3	S7
	S1(32)	S2(64)	S5	S3	S8
	S1(40)	S2(40)	S5	S4	S9
	S6(-16)	S6(-8)	S3	S4	S10
			S3	S5	
			S4	S5	
	S6(0)	S6(8)	S4		
			S5		
			S5		
	S6(16)	S6(24)			
			S5		
			S5		
	S6(32)	S6(40)			

Παρατηρούμε ότι στον παραπάνω κώδικα έχουν ξεκινήσει 6 επαναλήψεις του αρχικού βρόχου πριν την είσοδο στο βρόχο που σηματοδοτείται με την ετικέτα VLIWloop. Έτσι, ενώ ο βρόχος των έξι εντολών VLIW φαίνεται να περιέχει 4 επαναλήψεις του αρχικού βρόχου, στην πραγματικότητα διαχειρίζεται 10 επαναλήψεις, με 8 από αυτές να είναι ταυτόχρονα ενεργές! Έτσι, ο νέος βρόχος υλοποιεί πραγματικό ξεδίπλωμα 4 επαναλήψεων του αρχικού βρόχου, και συμβολικό ξεδίπλωμα 2 φορές πάνω στο πραγματικό. Μπορούμε να δούμε πώς το σώμα του νέου βρόχου περιέχει εντολές από 10 επαναλήψεις του αρχικού βρόχου, αν σημειώσουμε δίπλα σε κάθε εντολή – πλην των τεσσάρων εντολών ελέγχου του βρόχου – τον αριθμό αρχικής επανάληψης στον οποίο αντιστοιχεί, υποθέτοντας ότι η πρώτη αποθήκευση αναφέρεται στην επανάληψη i:

S1(48) (i+6)	S2(48) (i+6)	S3 (i+5)	S5 (i)
S1(56) (i+7)	S2(56) (i+7)	S4 (i+4)	S5 (i+1)
S6(0) (i)	S6(8) (i+1)	S4 (i+5)	S3 (i+6)
S1(32) (i+8)	S2(64) (i+8)	S5 (i+2)	S3 (i+7)
S1(40) (i+9)	S2(40) (i+9)	S5 (i+3)	S4 (i+6)
S6(-16) (i+2)	S6(-8) (i+3)	S3 (i+8)	S4 (i+7)

Έτσι, στις 3 πρώτες εντολές ενεργές είναι οι επαναλήψεις i έως i+7, ενώ στις 3 τελευταίες εντολές ενεργές είναι οι επαναλήψεις i+2 έως i+9.
Ένας τέτοιος βαθμός συνολικού ξεδιπλώματος απαιτεί μεγάλο αριθμό καταχωρητών κινητής υποδιαστολής, όταν πρέπει να γίνει μετονομασία για μη παραβίαση των εξαρτήσεων τύπου ΕΜΕ που υπάρχουν τόσο στον αρχικό, όσο και στον ξεδιπλωμένο κώδικα. Από την άλλη μεριά, δεν απαιτούνται περισσότεροι καταχωρητές σταθερής υποδιαστολής από όσους συναντάμε στον αρχικό κώδικα MIPS, όμως οι $1, $2 και $3 τώρα μεταβάλλονται κατά 32 μέσα στο βρόχο, γι' αυτό και οι μετατοπίσεις των εντολών προσπέλασης μνήμης έχουν τροποποιηθεί

κατάλληλα. Μέσα στο νέο βρόχο, το ποσοστό πλήρωσης θέσεων εντολών VLIW γίνεται ίσο με το άνω φράγμα που βρήκαμε προηγουμένως για κ = 4. Ο ρυθμός ολοκλήρωσης επαναλήψεων του αρχικού βρόχου γίνεται ίσος με τέσσερις επαναλήψεις κάθε έξι κύκλους μηχανής, ή 2/3, όπως και αναμέναμε.
Ο παραπάνω κώδικας προϋποθέτει ένα συνολικό αριθμό επαναλήψεων της μορφής 6+4ρ, για ρ ≥ 1. Για την περίπτωση που ο συνολικός αριθμός επαναλήψεων του αρχικού βρόχου είναι μικρότερος από 10, ή όχι στη μορφή 6+4ρ, θα πρέπει να υπάρχει επιπλέον κώδικας, ο οποίος να υλοποιεί μεμονωμένες επαναλήψεις του αρχικού βρόχου, ώστε να συμπληρώνεται ο απαιτούμενος αριθμός επαναλήψεων.

Πρέπει να παρατηρήσουμε ότι η επιλογή της τιμής του κ δεν γίνεται με μόνο κριτήριο το άνω φράγμα στο ποσοστό πλήρωσης θέσεων εντολών VLIW. Ένας παράγοντας που πρέπει να λαμβάνουμε υπόψη είναι η μέγιστη απόσταση από την παραγωγή μιας τιμής με την εγγραφή της σε κάποιον καταχωρητή, μέχρι τη χρήση της με την ανάγνωσή της από τον ίδιο καταχωρητή. Ο χρόνος μιας επανάληψης του νέου βρόχου θα πρέπει να είναι μεγαλύτερος από την απόσταση αυτή, διαφορετικά θα κινδυνεύσουμε να χρησιμοποιήσουμε τιμή που παράγεται από λάθος επανάληψη. Πιο συγκεκριμένα, από τη στιγμή που μια επανάληψη του νέου βρόχου περιέχει εντολές από διαφορετικές επαναλήψεις του αρχικού βρόχου, πρέπει να είμαστε σίγουροι ότι η παραγωγή και η χρήση μιας τιμής γίνονται σε εντολές της σωστής επανάληψης του αρχικού βρόχου. Αυτό ακριβώς εξασφαλίζεται αν η μέγιστη απόσταση μεταξύ παραγωγής και χρήσης, συμπεριλαμβανομένων των εντολών παραγωγής και χρήσης της τιμής, δεν υπερβαίνει τη διάρκεια μιας επανάληψης του νέου βρόχου.
Μπορούμε να διακρίνουμε δύο περιπτώσεις. Αν η απόσταση αυτή οφείλεται μόνο στο χρόνο καθυστέρησης της εντολής που παράγει την τιμή, αν δηλαδή η εντολή χρήσης της τιμής είναι δρομολογημένη αμέσως μετά το χρόνο καθυστέρησης της εντολής παραγωγής, τότε δεν έχουμε άλλη επιλογή από το να αναγκάσουμε το κ να δώσει αριθμό εντολών VLIW τουλάχιστον ίσο με την απόσταση αυτή. Έτσι, αν λ είναι ο χρόνος καθυστέρησης της εντολής παραγωγής, τότε δεν μπορούμε να υλοποιήσουμε το νέο βρόχο με λιγότερες από λ+2 εντολές VLIW, ώστε να υπάρχει ικανή απόσταση μεταξύ των εμπλεκόμενων εντολών στο χρόνο μιας επανάληψης. Αν από την άλλη μεριά η απόσταση αυτή δεν οφείλεται μόνο στο χρόνο καθυστέρησης, αν δηλαδή η εντολή χρήσης είναι δρομολογημένη πιο κάτω από το σημείο στο οποίο η τιμή γίνεται διαθέσιμη, μπορούμε μετά το χρόνο καθυστέρησης να εισάγουμε μια εντολή αντιγραφής της τιμής σε άλλον καταχωρητή, οπότε στην ουσία σπάμε την απόσταση σε δύο μικρότερες. Βέβαια, θα πρέπει να υπάρχει η κατάλληλη θέση στις εντολές VLIW, αλλιώς η προσθήκη της εντολής αντιγραφής θα μας αναγκάσει να αρχίσουμε από την αρχή τη διαδικασία υπολογισμού του βέλτιστου κ, διότι θα αλλάξει το άνω φράγμα των αριθμών επαναλήψεων ανά εντολή VLIW.
Στον παραπάνω κώδικα, η εντολή με τη μέγιστη καθυστέρηση είναι η εντολή πολλαπλασιασμού S4 με λ = 3. Όμως, η εξαρτημένη από αυτήν εντολή S5 δεν δρομολογείται στην ελάχιστη απόσταση, αλλά έναν κύκλο αργότερα, επειδή δεν υπάρχει διαθέσιμη θέση εντολής VLIW στην απόσταση αυτή. Θα πρέπει επομένως να ισχύει 1,5κ ≥ λ+2+1, ή 1,5κ ≥ 6. Από την άλλη μεριά, και η ίδια η S4 είναι δρομολογημένη 4 κύκλους μακριά από την S1 από την οποία λαμβάνει τιμή, παρόλο που η S1 παράγει την τιμή με καθυστέρηση μόνο 1 κύκλου. Η απόσταση αυτή είναι μικρότερη από την προηγούμενη, οπότε δε μας απασχολεί. Αν όμως ήταν μεγαλύτερη, θα μπορούσαμε να σκεφτούμε την εναλλακτική επιλογή της αντιγραφής του καταχωρητή $f0 που περιέχει αυτή την τιμή σε κάποιον άλλο, αν βέβαια είχαμε περιθώριο στις θέσεις των εντολών VLIW που κατασκευάζουμε. Σύμφωνα με τα παραπάνω, η ελάχιστη τιμή του κ δεν είναι μεγαλύτερη από αυτή που επιλέξαμε με κριτήριο τη μεγιστοποίηση του ποσοστού πλήρωσης θέσεων εντολών VLIW, κι επομένως η τιμή που επιλέξαμε είναι αποδεκτή.

Άσκηση 8:

Έστω ο πιο κάτω βρόχος κώδικα C:

```
for (i=1; i<1000; i++) {
```

```
        a[i] = a[i+1] + b[i];    (S1)
        b[i] = a[i] + c[i];      (S2)
}
```

Βρείτε τις εξαρτήσεις από δεδομένα μεταξύ των εντολών του σώματός του, και αποφανθείτε αν ο βρόχος είναι παράλληλος. Αν δεν είναι παράλληλος, εξηγήστε αν μπορεί να γίνει παράλληλος ή μια σειριακή διάταξη παράλληλων βρόχων, και πώς αυτό επιτυγχάνεται.

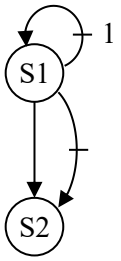
Απάντηση

Ένας βρόχος είναι παράλληλος, αν οι επαναλήψεις του μπορούν να εκτελεστούν με οποιαδήποτε σειρά. Κάτι τέτοιο μπορεί να συμβεί, αν δεν υπάρχουν εξαρτήσεις από δεδομένα που μεταφέρονται μεταξύ επαναλήψεων του βρόχου. Μπορεί να υπάρχουν εξαρτήσεις από δεδομένα μεταξύ των εντολών της ίδιας επανάληψης, κάτι που περιορίζει το βαθμό παραλληλισμού σε επίπεδο εντολών του σώματος του βρόχου, αλλά αυτό δε σχετίζεται με τον παραλληλισμό σε επίπεδο επαναλήψεων του βρόχου που μας απασχολεί σε αυτή την άσκηση.

Οι εξαρτήσεις από δεδομένα στο σώμα του βρόχου σχετίζονται με τις αναφορές στα στοιχεία των πινάκων a και b. Με την υπόθεση ότι οι πίνακες δεν επικαλύπτονται – διαφορετικά η ανάλυση δεν έχει νόημα, οι εξαρτήσεις είναι οι εξής:

- AME από την εντολή S1 προς την S2, λόγω αναφοράς στο στοιχείο a[i].
- EMA από την εντολή S1 προς την S2, λόγω αναφοράς στο στοιχείο b[i].
- EMA από την εντολή S1 προς την S1 της επόμενης επανάληψης, λόγω ανάγνωσης του στοιχείου a[i+1] και εγγραφής του στοιχείου a[i].

Μπορούμε να απεικονίσουμε τις παραπάνω εξαρτήσεις από δεδομένα στον ακόλουθο γράφο εξαρτήσεων:



όπου οι εξαρτήσεις τύπου EMA σημειώνονται με μια κάθετη γραμμή πάνω στην αντίστοιχη ακμή, και ο αριθμός δίπλα σε μια εξάρτηση υποδηλώνει την απόσταση επαναλήψεων στην οποία μεταφέρεται η εξάρτηση.
Εξαιτίας της ύπαρξης μιας εξάρτησης που μεταφέρεται σε διαδοχικές επαναλήψεις, και συγκεκριμένα της εξάρτησης από την εντολή S1 προς τον εαυτό της, ο βρόχος δεν είναι παράλληλος με τη μορφή που μας δόθηκε. Για να διαπιστώσουμε αν ο βρόχος μετατρέπεται σε παράλληλο ή σε κάποια σειρά παράλληλων βρόχων, θα εξετάσουμε το γράφο εξαρτήσεων.

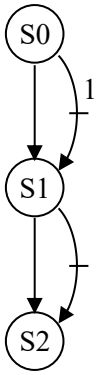
Γενικά, αν στο γράφο εξαρτήσεων κάποιου βρόχου σχηματίζεται κάποιος κύκλος, όπου όλες οι εξαρτήσεις είναι τύπου AME, τότε ο βρόχος δε μπορεί να μετασχηματιστεί σε παράλληλο. Αν ο κύκλος αυτός δεν καλύπτει όλες τις εντολές του σώματος του βρόχου, τότε είναι πιθανό να μπορούμε να διασπάσουμε το βρόχο σε δύο επιμέρους βρόχους, έτσι ώστε στον ένα να συμμετέχουν οι εντολές που σχηματίζουν τον κύκλο, και ο οποίος θα είναι αναγκαστικά σειριακός, και στον άλλο να συμμετέχουν οι υπόλοιπες εντολές, και ο οποίος να είναι παράλληλος. Για να επιτρέπεται διάσπαση του αρχικού βρόχου, θα πρέπει οι υπόλοιπες εντολές να μη συμμετέχουν σε άλλον κύκλο που να έχει κοινές εντολές με τον πρώτο, ενώ για να είναι ο δεύτερος βρόχος παράλληλος, θα πρέπει αυτό να το επιτρέπουν οι εξαρτήσεις μεταξύ των εντολών που συμμετέχουν στο σώμα του.
Σε κάθε περίπτωση κύκλου εξαρτήσεων όπου συμμετέχει εξάρτηση τύπου EMA, είναι δυνατό να γίνει μετασχηματισμός στον κώδικα μέσω μετονομασίας, ο οποίος θα διασπά τον κύκλο και

θα μεταφέρει αυτή την εξάρτηση έξω από αυτόν. Πιο συγκεκριμένα, θα δημιουργείται νέα εντολή, η οποία θα αντιγράφει την τιμή ανάγνωσης σε προσωρινό πίνακα, και στη συνέχεια η τιμή θα διαβάζεται από τον πίνακα αυτόν. Παρόμοιος μετασχηματισμός εφαρμόζεται και για εξαρτήσεις τύπου ΕΜΕ.
Από τη στιγμή που έχουμε απαλείψει τους κύκλους εξαρτήσεων σε ένα βρόχο, και αν υπάρχουν εξαρτήσεις που μεταφέρονται μεταξύ επαναλήψεων, διασπάμε το βρόχο ανάμεσα στις εντολές που συμμετέχουν στις εξαρτήσεις αυτές, και εκτελούμε τους επιμέρους βρόχους με τη σειρά που επιβάλλουν οι εξαρτήσεις, αλλά με κάθε έναν από αυτούς να είναι πια παράλληλος.

Σύμφωνα με τα παραπάνω, ο κύκλος που έχουμε στον κώδικά μας μπορεί να απαλειφτεί, αφού δημιουργείται από εξάρτηση τύπου ΕΜΑ. Επομένως, ο βρόχος που μας δόθηκε μπορεί να μετατραπεί σε παράλληλο, ή σε μια σειρά από παράλληλους βρόχους. Για το σκοπό αυτό, δημιουργούμε έναν προσωρινό πίνακα, στον οποίο αντιγράφουμε την τιμή του στοιχείου a[i+1], και ξαναγράφουμε τον κώδικα C ως εξής:

```
for (i=1; i<1000; i++) {
      t[i] = a[i+1];          (S0)
      a[i] = t[i] + b[i];     (S1)
      b[i] = a[i] + c[i];     (S2)
}
```

Ο γράφος εξαρτήσεων του νέου βρόχου θα είναι ο ακόλουθος:



όπου βλέπουμε πώς ο κύκλος που είχαμε προηγουμένως έχει τώρα μετατραπεί σε δύο εξαρτήσεις από την εντολή S0 προς την εντολή S1, και ο νέος γράφος δεν περιέχει κύκλο.
Επειδή ο βρόχος περιέχει μια εξάρτηση που μεταφέρεται από μία επανάληψη στην επόμενη, συνεχίζει να μην είναι παράλληλος. Αυτό μπορεί να επαληθευτεί, θεωρώντας τις δύο πρώτες επαναλήψεις του βρόχου, και παρατηρώντας ότι με παράλληλη εκτέλεσή τους, το στοιχείο a[2] θα μπορούσε να πάρει νέα τιμή στη δεύτερη επανάληψη πριν διαβαστεί στην πρώτη.
Εφ' όσον όμως ο νέος βρόχος δεν περιέχει κύκλο εξαρτήσεων, μπορεί να διασπαστεί σε δύο παράλληλους βρόχους forall[1], ως εξής:

```
forall (i=1; i<1000; i++)
      t[i] = a[i+1];          (S0)
forall (i=1; i<1000; i++) {
      a[i] = t[i] + b[i];     (S1)
      b[i] = a[i] + c[i];     (S2)
}
```

Οι δύο παράλληλοι βρόχοι εκτελούνται ο ένας μετά τον άλλο.
Αξίζει να σημειωθεί ότι η αντιγραφή των αρχικών τιμών του πίνακα που μετονομάζουμε δεν είναι πάντα αναγκαία. Έτσι, για το συγκεκριμένο παράδειγμα, μπορούμε να χρησιμοποιούμε τον πίνακα t για κάθε αναφορά στον a σε κώδικα που προηγείται του βρόχου που μελετάμε,

[1] Ο βρόχος forall είναι ένας παράλληλος βρόχος for και ανήκει σε επέκταση της C για παράλληλο προγραμματισμό.

ώστε να έχουμε τις τιμές των στοιχείων a[i+1] ήδη στα στοιχεία t[i] πριν την έναρξη του βρόχου. Η δυνατότητα μιας τέτοιας βελτιστοποίησης εξαρτάται βέβαια από τη ροή ελέγχου και δεδομένων του συνολικού κώδικα, μια που οι τιμές των στοιχείων a[i] μπορεί να μην παράγονται μόνο μέσα από το βρόχο, αλλά και από άλλα μονοπάτια ελέγχου!

Άσκηση 9:

Θεωρήστε ένα βαθμωτό επεξεργαστή MIPS με στατική δρομολόγηση εντολών. Η ΜΕΔ του επεξεργαστή αυτού διαθέτει μία υπομονάδα πρόσθεσης/αφαίρεσης κινητής υποδιαστολής που απαιτεί 2 κύκλους μηχανής για κάθε πράξη, μία υπομονάδα πολλαπλασιασμού κινητής υποδιαστολής που απαιτεί 8 κύκλους μηχανής για κάθε πράξη και μία υπομονάδα σταθερής υποδιαστολής που απαιτεί έναν κύκλο μηχανής για κάθε πράξη. Ακόμα, διαθέτει μία υπομονάδα προσπέλασης μνήμης που απαιτεί 3 κύκλους μηχανής για κάθε προσπέλαση, μετά τον υπολογισμό της τελικής διεύθυνσης προσπέλασης που γίνεται στην υπομονάδα σταθερής υποδιαστολής. Οι διακλαδώσεις εκτελούνται στην υπομονάδα σταθερής υποδιαστολής χωρίς καθυστέρηση. Οι υπομονάδες διαθέτουν μηχανισμό παροχέτευσης για προώθηση σε άλλες υπομονάδες των δεδομένων που παράγουν, αλλά καμία υπομονάδα δεν υποστηρίζει μερική επικάλυψη στην εκτέλεση διαδοχικών πράξεων. Τέλος, ο επεξεργαστής διαθέτει 32 καταχωρητές σταθερής και 32 καταχωρητές κινητής υποδιαστολής.
Θεωρήστε στη συνέχεια τον ακόλουθο κώδικα MIPS:

```
loop: ldc1  $f0,0($1)        (S1)
      ldc1  $f1,0($2)        (S2)
      add.d $f2,$f0,$f0      (S3)
      add.d $f2,$f2,$f1      (S4)
      mul.d $f3,$f0,$f1      (S5)
      sub.d $f3,$f3,$f2      (S6)
      sdc1  $f3,0($3)        (S7)
      addiu $1,$1,8          (S8)
      addiu $2,$2,8          (S9)
      addiu $3,$3,8          (S10)
      bne   $1,$4,loop       (S11)
```

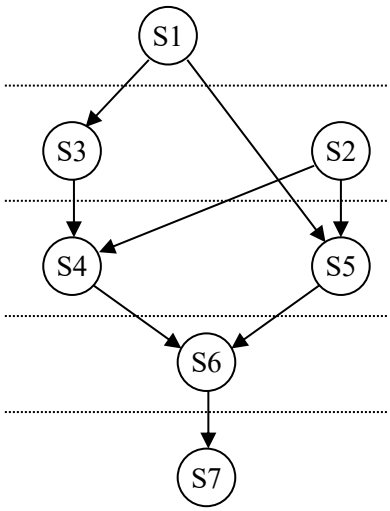
Αν οι περιοχές μνήμης που προσπελαύνονται από τις εντολές προσπέλασης μνήμης δεν επικαλύπτονται, εφαρμόστε την τεχνική του συμβολικού ξεδιπλώματος στον παραπάνω κώδικα, έτσι ώστε να αυξηθεί η απόσταση δρομολόγησης των εντολών κινητής υποδιαστολής και προσπέλασης μνήμης που εμφανίζουν κινδύνους από εξαρτήσεις δεδομένων, και ο κώδικας να δρομολογείται στον επεξεργαστή χωρίς παγώματα.

Απάντηση

Σε ένα βαθμωτό επεξεργαστή με στατική δρομολόγηση εντολών, οι εντολές εκτελούνται με τη σειρά που ανακαλούνται από τη μνήμη εντολών. Σε περίπτωση εμφάνισης κινδύνου από εξάρτηση δεδομένων που δεν καλύπτεται με παροχέτευση, η ΜΕΔ παγώνει, μέχρι το δεδομένο που απαιτείται να γίνει διαθέσιμο. Επομένως, αν θέλουμε η σειρά ανάκλησης των εντολών να οδηγεί σε δρομολόγηση χωρίς παγώματα, η παραγωγή κώδικα θα πρέπει να γίνεται με κατάλληλες τεχνικές αναδιάταξης των εντολών.
Με την τεχνική του συμβολικού ξεδιπλώματος δημιουργούμε ένα νέο βρόχο που, ενώ φαινομενικά εκτελεί εντολές μιας επανάληψης, περιέχει εντολές από πολλές διαδοχικές επαναλήψεις. Οι εντολές είναι τοποθετημένες με τέτοιον τρόπο, ώστε αν εμφανίζουν κάποιον κίνδυνο από εξάρτηση τύπου ΑΜΕ, να μη βρίσκονται σε θέσεις που να οδηγούν σε πάγωμα στην εκτέλεση του κώδικα. Αν το πάγωμα δεν αποφεύγεται στον αρχικό βρόχο, με αντιστοίχηση των εντολών σε εντολές διαδοχικών επαναλήψεων, η απόσταση στη δρομολόγησή τους αυξάνεται μέχρι το μέγεθος του σώματος του βρόχου.

Για την εφαρμογή του συμβολικού ξεδιπλώματος απαιτείται η ανάλυση των εξαρτήσεων από δεδομένα στο σώμα του αρχικού βρόχου. Οι κίνδυνοι από τις εξαρτήσεις τύπου AME αντιμετωπίζονται με το συμβολικό ξεδίπλωμα, ενώ οι εξαρτήσεις τύπου EME και EMA απαλείφονται με μετονομασία καταχωρητών.
Για να μην υπάρχει πάγωμα στη δρομολόγηση δύο εντολών που εμφανίζουν κάποια εξάρτηση τύπου AME από την πρώτη προς τη δεύτερη, πρέπει η απόσταση δρομολόγησής τους να είναι τουλάχιστον ίση με το χρόνο που απαιτείται για την εκτέλεση της πρώτης. Η μέγιστη απόσταση δρομολόγησης δύο τέτοιων εντολών επιτυγχάνεται, αν η δεύτερη εντολή είναι τοποθετημένη αμέσως πριν από την πρώτη. Αυτό συμβαίνει, μια που για σώμα βρόχου N εντολών, η μέγιστη απόσταση στη δρομολόγηση χωρίς πάγωμα εντολών της ίδιας επανάληψης του αρχικού βρόχου με κάποια εξάρτηση τύπου AME θα είναι N-1. Δεδομένου ότι σε απόσταση N από μια εντολή θα δρομολογηθεί η ίδια εντολή της επόμενης επανάληψης, η εντολή που λαμβάνει κάποιο δεδομένο δε μπορεί να δρομολογηθεί σε απόσταση μεγαλύτερη από N από την εντολή που το παράγει, διότι τότε θα λάμβανε λανθασμένα το δεδομένο που παράγεται στην επόμενη επανάληψη. Εκτός από την ειδική περίπτωση που η εντολή που λαμβάνει το δεδομένο είναι η τελευταία εντολή στο σώμα του βρόχου, οπότε η εντολή που το παράγει θα βρίσκεται στην αρχή του, οι δύο εντολές θα πρέπει να τοποθετηθούν στο νέο βρόχο με την πρώτη να βρίσκεται αμέσως πριν τη δεύτερη, ώστε να δρομολογούνται σε διαδοχικές επαναλήψεις του νέου βρόχου.
Φυσικά, αν κάποια εντολή έχει χρόνο εκτέλεσης μεγαλύτερο από N-1, δε μπορούμε να επιτύχουμε δρομολόγηση χωρίς πάγωμα των εντολών που εξαρτώνται από αυτή, εκτός εάν η διαδικασία κατασκευής του νέου βρόχου απαιτήσει την εισαγωγή εντολών αντιγραφής, όπως θα δούμε πιο κάτω, και το μέγεθος του σώματος του νέου βρόχου ξεπεράσει το χρόνο εκτέλεσης της εντολής.

Οι εξαρτήσεις από δεδομένα τύπου AME στο σώμα του βρόχου που μας δίνεται, χωρίς να λάβουμε υπ' όψη τις εντολές ελέγχου του βρόχου που σαν εντολές σταθερής υποδιαστολής δε δημιουργούν παγώματα στην εκτέλεσή του, απεικονίζονται στον παρακάτω γράφο εξαρτήσεων. Η τοποθέτηση των εντολών του κώδικά μας σε ένα νέο βρόχο που να είναι σε συμφωνία με τα παραπάνω μπορεί να γίνει ευκολότερα, αν διαμερίσουμε το γράφο σε επίπεδα από κάτω προς τα επάνω, με βάση την απόσταση των εντολών από την S7. Οι εντολές κάθε επιπέδου θα τοποθετηθούν ανάμεσα στις εντολές των γειτονικών επιπέδων και θα αντιστοιχούν στην ίδια επανάληψη του αρχικού βρόχου. Εντολές διαδοχικών επιπέδων της ίδιας επανάληψης του αρχικού βρόχου θα εκτελούνται σε διαδοχικές επαναλήψεις του νέου βρόχου.



Έτσι, μπορούμε να δούμε ότι η εντολή S6 θα τοποθετηθεί αμέσως μετά την S7, ως εξής:

```
sdc1  $f3,0($3)          (S7)
sub.d $f3,$f3,$f2        (S6)
```

Εφ' όσον ο καταχωρητής $f3 λαμβάνει τιμή στην S6, είναι εύκολα κατανοητό γιατί η εντολή S7 που λαμβάνει την τιμή αυτή – κι επομένως είναι εντολή της ίδιας επανάληψης του αρχικού βρόχου, θα εκτελεστεί στην επόμενη επανάληψη του νέου βρόχου. Η απόσταση στη δρομολόγηση των δύο εντολών είναι η μεγαλύτερη δυνατή, αφού μεταξύ τους θα μεσολαβήσουν όλες οι υπόλοιπες εντολές του σώματος του βρόχου που θα εισάγουμε στη συνέχεια.
Δεδομένου ότι η S6 είναι εξαρτημένη από δύο εντολές, τις S4 και S5, μόνο μία από αυτές θα βρεθεί αμέσως μετά την S6, ενώ η άλλη θα ακολουθήσει, κι επομένως θα δρομολογηθεί σε απόσταση κατά μία θέση μικρότερη από τη μέγιστη. Σε μια τέτοια περίπτωση, μπορούμε να τοποθετήσουμε στη μέγιστη απόσταση την εντολή που απαιτεί τον περισσότερο χρόνο για την εκτέλεσή της, που εδώ είναι η S5:

```
sdc1  $f3,0($3)          (S7)
sub.d $f3,$f3,$f2        (S6)
mul.d $f3,$f0,$f1        (S5)
```

Ο παραπάνω κώδικας είναι όμως λανθασμένος ύστερα από την αναδιάταξη που κάναμε, διότι η εξάρτηση τύπου ΕΜΕ στον καταχωρητή $f3 μεταξύ των S5 και S6 οδηγεί σε αποθήκευση στη μνήμη κατά την εκτέλεση της εντολής S7 της τιμής που παράγει η S5, και όχι της τιμής που παράγει η εντολή S6. Στο βαθμωτό επεξεργαστή MIPS, η εξάρτηση αυτή δε δημιουργεί κίνδυνο στον αρχικό βρόχο, όμως δεν επιτρέπει αναδιάταξη των S5 και S6, γι' αυτό και θα πρέπει να μετονομάσουμε τον καταχωρητή $f3 στη δεύτερη εγγραφή του. Ο νέος κώδικας θα είναι ο ακόλουθος:

```
sdc1  $f13,0($3)         (S7)
sub.d $f13,$f3,$f2       (S6)
mul.d $f3,$f0,$f1        (S5)
```

όπου σωστά πια, η εντολή S7 αποθηκεύει στη μνήμη την τιμή που παράγει η S6, η οποία τώρα μεταφέρεται σ' αυτήν μέσω του καταχωρητή $f13.

Η εντολή S4 τοποθετείται κάτω από την S5, και οι εντολές του επόμενου επιπέδου θα τοποθετηθούν μετά τις S4 και S5. Παρ' όλο που δεν υπάρχει εξάρτηση μεταξύ των εντολών S3 και S5, δε μπορούμε να τοποθετήσουμε την S3 μεταξύ των S4 και S5, για το λόγο που θα εξηγήσουμε πιο κάτω. Μέχρι το σημείο αυτό λοιπόν, έχουμε κατασκευάσει το ακόλουθο τμήμα από το σώμα του νέου βρόχου:

```
sdc1  $f13,0($3)         (S7)
sub.d $f13,$f3,$f2       (S6)
mul.d $f3,$f0,$f1        (S5)
add.d $f2,$f2,$f1        (S4)
```

Το επόμενο επίπεδο περιέχει τις εντολές S2 και S3. Από τις δύο εντολές επιλέγουμε να τοποθετήσουμε πρώτα την S2 που έχει μεγαλύτερο χρόνο εκτέλεσης. Όμως, θα πρέπει να διορθώσουμε τη μετατόπισή της, ώστε να ανταποκρίνεται στο νέο βρόχο. Δεδομένου ότι η εντολή S2 βρίσκεται τρία επίπεδα πάνω από την S7, θα χρειαστούν τρεις επαναλήψεις του νέου βρόχου μέχρι την εκτέλεση της εντολής S7 της ίδιας επανάληψης του αρχικού βρόχου, ή ισοδύναμα, η εντολή S2 θα αντιστοιχεί σε τρεις επαναλήψεις του αρχικού βρόχου μετά την επανάληψη στην οποία βρίσκεται η S7 της ίδιας επανάληψης του νέου βρόχου. Επομένως, με μέγεθος δεδομένου ίσο με 8, η μετατόπιση της εντολής S2 θα αυξηθεί κατά 24:

```
sdc1  $f13,0($3)         (S7)
sub.d $f13,$f3,$f12      (S6)
mul.d $f3,$f0,$f1        (S5)
add.d $f12,$f2,$f1       (S4)
ldc1  $f1,24($2)         (S2)
add.d $f2,$f0,$f0        (S3)
```

Στον παραπάνω κώδικα εφαρμόσαμε μετονομασία για τη δεύτερη εγγραφή του καταχωρητή $f2 που δημιουργεί εξάρτηση τύπου ΕΜΕ μεταξύ των εντολών S3 και S4.

Μέχρι τώρα οι εντολές που τοποθετήσαμε στο νέο βρόχο είχαν εξαρτήσεις που εκτείνονταν σε δύο μόνο επίπεδα. Παρατηρούμε όμως ότι η εξάρτηση από την εντολή S1 προς την S5 εκτείνεται σε τρία επίπεδα, γεγονός που είναι αποτέλεσμα της μορφής που έχει ο γράφος εξαρτήσεων.
Ας δοκιμάσουμε να τοποθετήσουμε την εντολή S1 απλά κάτω από τις S2 και S3:

```
            sdc1  $f13,0($3)        (S7)
            sub.d $f13,$f3,$f12     (S6)
            mul.d $f3,$f0,$f1       (S5)
            add.d $f12,$f2,$f1      (S4)
            ldc1  $f1,24($2)        (S2)
            add.d $f2,$f0,$f0       (S3)
            ldc1  $f0,32($1)        (S1)
```

όπου η μετατόπιση της S1 τροποποιήθηκε κατάλληλα, ώστε να αντιστοιχεί σε τέσσερις επαναλήψεις του αρχικού βρόχου μετά την S7.
Μελετώντας τη δρομολόγηση των εντολών μιας επανάληψης, έστω i, του αρχικού βρόχου, παρατηρούμε τα εξής: Αν η S1 της επανάληψης i του αρχικού βρόχου που βρίσκεται σε κάποια επανάληψη, έστω j, του νέου βρόχου διαβάσει κάποια τιμή από τη μνήμη, η τιμή αυτή θα χρησιμοποιηθεί από τις εντολές S3 και S5 της επόμενης επανάληψης j+1 του νέου βρόχου. Στην επανάληψη j+1 όμως θα εκτελεστεί και η εντολή S2 της επανάληψης i του αρχικού βρόχου, η οποία θα διαβάσει από τη μνήμη κάποια τιμή που πρέπει να λάβει η S5 της ίδιας επανάληψης του αρχικού βρόχου. Η τελευταία όμως είναι πάνω από την S2 και έχει ήδη εκτελεστεί, με λανθασμένη τιμή του καταχωρητή $f1.
Το παραπάνω σφάλμα συνέβη επειδή η S5 παρέλαβε το δεδομένο που παρήγαγε η S1 μια επανάληψη νωρίτερα απ' ό,τι έπρεπε. Επειδή οι S1 και S2 βρίσκονται σε διαφορετικά επίπεδα του γράφου εξαρτήσεων, τα δεδομένα που παράγουν για την ίδια επανάληψη του αρχικού βρόχου, λαμβάνονται από την S5 σε διαφορετικές επαναλήψεις του νέου βρόχου, ενώ για σωστή εκτέλεσή του, θα έπρεπε να ληφθούν στην ίδια επανάληψη.
Το πρόβλημα αυτό δε θα αντιμετωπιζόταν με μεταφορά της εντολής S5 στο παραπάνω επίπεδο. Τότε, θα αυξανόταν η απόσταση μεταξύ των εντολών S5 και S6, αλλά και μεταξύ των εντολών S2 και S4, μια που και η εντολή S2 αναγκαστικά θα μεταφερόταν ένα επίπεδο πιο πάνω, και το πρόβλημα θα ξαναεμφανιζόταν σε άλλα σημεία του κώδικα. Μάλιστα, αυτός είναι και ο λόγος που δεν τοποθετήσαμε την εντολή S3 μεταξύ των εντολών S4 και S5, μια που αυτό θα ισοδυναμούσε με άνοδο της S5 στο επίπεδο της S3!
Για να αντιμετωπίσουμε το παραπάνω πρόβλημα, είμαστε αναγκασμένοι να εισάγουμε μια εντολή αντιγραφής, έστω S12, η οποία απλά να μεταφέρει το δεδομένο που παράγει η εντολή S1 σε έναν άλλο καταχωρητή, από τον οποίο στη συνέχεια να το παραλάβει η S5. Η εντολή αυτή διασπά την εξάρτηση τύπου ΑΜΕ από την εντολή S1 προς την εντολή S5 σε δύο μέρη, και τοποθετείται έτσι στο επίπεδο των S2 και S3 του γράφου εξαρτήσεων. Με τον τρόπο αυτό εξασφαλίζουμε ότι καμία εξάρτηση δεν εκτείνεται σε παραπάνω από δύο επίπεδα του γράφου.
Ο διορθωμένος κώδικας, με την προσθήκη και των εντολών ελέγχου του βρόχου, θα είναι ο ακόλουθος:

```
swploop:    sdc1  $f13,0($3)        (S7)
            sub.d $f13,$f3,$f12     (S6)
            mul.d $f3,$f10,$f1      (S5)
            add.d $f12,$f2,$f1      (S4)
            ldc1  $f1,24($2)        (S2)
            add.d $f2,$f0,$f0       (S3)
            mov.d $f10,$f0          (S12)
            ldc1  $f0,32($1)        (S1)
            addiu $1,$1,8           (S8)
            addiu $2,$2,8           (S9)
            addiu $3,$3,8           (S10)
            bne   $1,$4,swploop     (S11)
```

Ο παραπάνω κώδικας συμπληρώνεται με τους κώδικες προλόγου και επιλόγου, οι οποίοι ολοκληρώνουν τις επαναλήψεις του αρχικού βρόχου που πρέπει να έχουν ξεκινήσει πριν την αρχή, και τις επαναλήψεις που πρέπει να ολοκληρωθούν μετά το τέλος του νέου βρόχου. Έτσι, για πέντε επίπεδα στο γράφο εξαρτήσεων, έχουν ξεκινήσει και πρέπει να ολοκληρωθούν τέσσερις ακόμα επαναλήψεις του αρχικού βρόχου. Ο αντίστοιχος κώδικας προλόγου και επιλόγου προκύπτει με αντιγραφή τμημάτων του σώματος του βρόχου όσες φορές χρειάζεται, αφαιρώντας κάθε φορά τις εντολές που εκτελούνται μέσα στο βρόχο. Έτσι, ο συνολικός κώδικας θα είναι ο εξής:

```
prologue:   ldc1  $f0,0($1)         (S1)
            ldc1  $f1,0($2)         (S2)
            add.d $f2,$f0,$f0       (S3)
            mov.d $f10,$f0          (S12)
            ldc1  $f0,8($1)         (S1)
            mul.d $f3,$f10,$f1      (S5)
            add.d $f12,$f2,$f1      (S4)
            ldc1  $f1,8($2)         (S2)
            add.d $f2,$f0,$f0       (S3)
            mov.d $f10,$f0          (S12)
            ldc1  $f0,16($1)        (S1)
            sub.d $f13,$f3,$f12     (S6)
            mul.d $f3,$f10,$f1      (S5)
            add.d $f12,$f2,$f1      (S4)
            ldc1  $f1,16($2)        (S2)
            add.d $f2,$f0,$f0       (S3)
            mov.d $f10,$f0          (S12)
            ldc1  $f0,24($1)        (S1)
swploop:    sdc1  $f13,0($3)        (S7)
            sub.d $f13,$f3,$f12     (S6)
            mul.d $f3,$f10,$f1      (S5)
            add.d $f12,$f2,$f1      (S4)
            ldc1  $f1,24($2)        (S2)
            add.d $f2,$f0,$f0       (S3)
            mov.d $f10,$f0          (S12)
            ldc1  $f0,32($1)        (S1)
            addiu $1,$1,8           (S8)
            addiu $2,$2,8           (S9)
            addiu $3,$3,8           (S10)
            bne   $1,$4,swploop     (S11)
epilogue:   sdc1  $f13,0($3)        (S7)
            sub.d $f13,$f3,$f12     (S6)
            mul.d $f3,$f10,$f1      (S5)
            add.d $f12,$f2,$f1      (S4)
            ldc1  $f1,24($2)        (S2)
            add.d $f2,$f0,$f0       (S3)
            mov.d $f10,$f0          (S12)
            sdc1  $f13,8($3)        (S7)
            sub.d $f13,$f3,$f12     (S6)
            mul.d $f3,$f10,$f1      (S5)
            add.d $f12,$f2,$f1      (S4)
            sdc1  $f13,16($3)       (S7)
            sub.d $f13,$f3,$f12     (S6)
            sdc1  $f13,24($3)       (S7)
```

Είναι φανερό ότι ο διαμερισμός του γράφου εξαρτήσεων σε επίπεδα δεν είναι μοναδικός. Κάποιες φορές μάλιστα προτιμάμε να διατηρήσουμε στο ίδιο επίπεδο εντολές που έχουν κάποια εξάρτηση τύπου AME μεταξύ τους, οπότε αυτές τοποθετούνται στο νέο βρόχο με τη σειρά που είχαν και στον αρχικό. Κάτι τέτοιο είναι επιθυμητό, αν έτσι παράγεται μικρότερος κώδικας, λόγω πολυπλοκότητας των εξαρτήσεων τύπου AME που υπαγορεύει την εισαγωγή πολλών εντολών αντιγραφής, ή ακόμα, αν έτσι ο κώδικας χρησιμοποιεί λιγότερους καταχωρητές. Αν

για παράδειγμα στον κώδικα που μας δόθηκε τοποθετούσαμε την εντολή S1 στο ίδιο επίπεδο με την S3, οπότε στο νέο βρόχο θα την τοποθετούσαμε πριν την S3, όπως δηλαδή βρίσκεται στον αρχικό κώδικα, δε θα χρειαζόταν να αντιγράψουμε την τιμή που διαβάζεται από τη μνήμη σε κάποιον προσωρινό καταχωρητή, γιατί η S1 θα βρισκόταν σε απόσταση ενός επιπέδου από την S5. Αυτό θα το προτιμούσαμε, αν η εντολή S1 είχε μικρότερο χρόνο εκτέλεσης. Προφανώς, με τον τρόπο αυτό δεν επιτυγχάνουμε τη μεγιστοποίηση στην απόσταση δρομολόγησης εξαρτημένων εντολών, αλλά αυτό δε μας πειράζει, ειδικά για εντολές που δεν έχουν μεγάλο χρόνο εκτέλεσης, όπως για παράδειγμα εντολές σταθερής υποδιαστολής.
Γενικά λοιπόν, το πρόβλημα της παραγωγής κώδικα με συμβολικό ξεδίπλωμα δεν έχει μονοσήμαντη λύση, και απαιτείται χρήση ευριστικών κανόνων για παραγωγή κώδικα που να δίνει τη βέλτιστη δρομολόγηση εντολών.

Ας σημειωθεί ότι η πιο πάνω εφαρμογή της τεχνικής του συμβολικού ξεδιπλώματος δε θα ήταν επιτρεπτή, αν οι θέσεις μνήμης στις οποίες γίνονται εγγραφές μπορούσαν να επικαλύπτονται με τις θέσεις μνήμης από τις οποίες γίνονται αναγνώσεις. Πιο γενικά, το συμβολικό ξεδίπλωμα δε θα επιτρεπόταν, αν ο βρόχος εμφάνιζε κύκλο εξαρτήσεων τύπου AME, είτε μέσω καταχωρητή, είτε μέσω της μνήμης, το οποίο στην ουσία θα σήμαινε ότι ο βρόχος δεν είναι, ούτε μπορεί να γίνει παράλληλος. Στην πραγματικότητα όμως, το συμβολικό ξεδίπλωμα δεν εκτελεί το βρόχο σε πλήρη παραλληλία, αλλά μόνο σε παραλληλία ενός αριθμού διαδοχικών επαναλήψεων. Έτσι, αν γνωρίζουμε ότι στον κύκλο εξαρτήσεων η ελάχιστη απόσταση επαναλήψεων στην οποία μεταφέρονται εξαρτήσεις είναι μεγαλύτερη από τον αριθμό διαδοχικών επαναλήψεων που εκτελούνται παράλληλα, η εφαρμογή της πιο πάνω τεχνικής γίνεται δυνατή!

Άσκηση 10:

Θεωρήστε τον ακόλουθο κώδικα MIPS:

```
        lw    $3,0($7)
        slt   $1,$3,$4
        beq   $1,$0,L1
        addu  $3,$3,$2
L1:     sw    $3,0($8)
```

Μετασχηματίστε τον κώδικα με τη βοήθεια (α) εντολών μεταφοράς υπό συνθήκη, και (β) βεβαιωμένης εκτέλεσης, ώστε να μην περιέχει εντολή διακλάδωσης.

Απάντηση

Στην πρώτη περίπτωση, η εντολή πρόσθεσης θα εκτελείται ανεξάρτητα από την τιμή που λαμβάνει ο καταχωρητής $1. Άρα, θα πρέπει να γράφει το αποτέλεσμά της σε κάποιον προσωρινό καταχωρητή, απ' όπου αυτό θα μεταφερθεί στον $3 μέσω κάποιας εντολής μεταφοράς υπό συνθήκη. Η συνθήκη αυτή θα καθορίζεται από το περιεχόμενο του καταχωρητή $1:

```
lw     $3,0($7)
slt    $1,$3,$4
addu   $13,$3,$2
cmovn  $3,$13,$1
sw     $3,0($8)
```

και είναι η μη μηδενική τιμή του, ώστε η μεταφορά να συμβαίνει, ακριβώς όταν η αρχική διακλάδωση δεν εκτελούσε άλμα.

Στη δεύτερη περίπτωση, η εντολή πρόσθεσης θα εκτελείται με συνθήκη που καθορίζεται από κάποιον καταχωρητή βεβαίωσης. Η εντολή σύγκρισης θα πρέπει να τροποποιηθεί, ώστε να δίνει τιμή στον καταχωρητή βεβαίωσης, μια που τώρα δε χρειαζόμαστε τον καταχωρητή $1:

```
lw     $3,0($7)
cmp.lt p0,p1=$3,$4
```

```
(p0)  addu   $3,$3,$2
      sw     $3,0($8)
```

όπου p0 και p1 καταχωρητές βεβαίωσης. Όταν ο καταχωρητής p0 έχει μηδενική τιμή, το οποίο θα συμβαίνει όταν το περιεχόμενο του $3 δεν είναι μικρότερο από το περιεχόμενο του $4, η εντολή πρόσθεσης εκτελείται σαν εντολή nop, και το περιεχόμενο του $3 παραμένει αμετάβλητο.