Άσκηση 4.12

- Στην άσκηση αυτή, εξετάζουμε την επίδραση της **διοχέτευσης** στο χρόνο κύκλου ρολογιού του επεξεργαστή. Τα προβλήματα αυτής της άσκησης θεωρούν ότι τα μεμονωμένα στάδια της διαδρομής δεδομένων έχουν τους επόμενους λανθάνοντες χρόνους:

	IF	ID	EX	MEM	WB
α.	300ps	400ps	350ps	500ps	100ps
β.	200ps	150ps	120ps	190ps	140ps

Άσκηση 4.12 (συνέχεια)

- **4.12.1** Ποιος είναι ο χρόνος κύκλου ρολογιού σε έναν επεξεργαστή με διοχέτευση και χωρίς διοχέτευση;
- Χωρίς διοχέτευση (σε έναν κύκλο) ο χρόνος του ρολογιού είναι ίσος με το άθροισμα των σταδίων. Με τη διοχέτευση ο χρόνος ρολογιού είναι ίσος με το χρόνο του μέγιστου σταδίου. Άρα:

	Χωρίς διοχέτευση (ενός κύκλου)	Με διοχέτευση
α.	1650ps	500ps
β.	800ps	200ps

Άσκηση 4.12 (συνέχεια)

- **4.12.2** Ποιος είναι ο συνολικός λανθάνων χρόνος μιας εντολής lw σε έναν επεξεργαστή με διοχέτευση και σε ένα χωρίς διοχέτευση;
- Χωρίς τη διοχέτευση είναι φυσικά ίσος με τον χρόνο του κύκλου ρολογιού (όπως και κάθε εντολή). Με τη διοχέτευση είναι ίσος με πέντε κύκλους ρολογιού (όπως και κάθε εντολή). Άρα:

	Χωρίς διοχέτευση (ενός κύκλου)	Με διοχέτευση
α.	1650ps	5x500=2500ps
β.	800ps	5x200=1000ps

Άσκηση 4.12 (συνέχεια)

- **4.12.3** Αν μπορούμε να διαιρέσουμε ένα στάδιο της διαδρομής δεδομένων με διοχέτευση σε δύο νέα στάδια, καθένα με το μισό λανθάνοντα χρόνο του αρχικού, ποιο στάδιο θα διαιρούσατε και ποιος είναι ο νέος χρόνος κύκλου ρολογιού του επεξεργαστή;
- Το μεγαλύτερο όφελος (μείωση του κύκλου της διοχέτευσης) θα προκύψει αν «σπάσουμε» στη μέση το μεγαλύτερο στάδιο. Στο α είναι το MEM και στο β είναι το IF. Άρα:

	Σταδίο που θα «σπάσει»	Νέο ρολόι
α.	MEM (αρχικά 500ps)	400ps (το ID)
β.	IF (αρχικά 200ps)	190ps (το MEM)

Άσκηση 4.12 (συνέχεια)

- Τα υπόλοιπα προβλήματα της άσκησης θεωρούν ότι οι εντολές που εκτελούνται από τον επεξεργαστή διαιρούνται ως εξής:

	ALU	beq	lw	sw
α.	50%	25%	15%	10%
β.	30%	25%	30%	15%

- **4.12.4** Με την παραδοχή ότι δεν υπάρχουν καθυστερήσεις (stalls) ή κίνδυνοι, ποια είναι η αξιοποίηση (utilization — το ποσοστό κύκλων που χρησιμοποιείται) της μνήμης δεδομένων;
- Αφού «αποχωρεί» (ολοκληρώνεται) από τη διοχέτευση μία εντολή σε κάθε κύκλο (δεν υπάρχουν χαμένοι κύκλοι) η αξιοποίηση είναι το άθροισμα των εντολών lw και sw

	Αξιοποίηση μνήμης δεδομένων
α.	25%
β.	45%

Άσκηση 4.12 (συνέχεια)

- **4.12.5** Με την παραδοχή ότι δεν υπάρχουν καθυστερήσεις (stalls) ή κίνδυνοι, ποια είναι η αξιοποίηση της θύρας «Καταχωρητής εγγραφής» της μονάδας «Καταχωρητές»;
- Όπως και πριν αφού δεν υπάρχουν χαμένοι κύκλοι, η απάντηση είναι το άθροισμα των ποσοστών των εντολών που γράφουν σε καταχωρητή. Δηλαδή των εντολών ALU (μορφή R) και των εντολών lw.

	Αξιοποίηση Καταχωρητή Εγγραφής
α.	50% + 15% = 65%
β.	30% + 30% = 60%

Άσκηση 4.12 (συνέχεια)

- **4.12.6** Αντί για μια οργάνωση ενός κύκλου, μπορούμε να χρησιμοποιήσουμε μια οργάνωση πολλών κύκλων όπου κάθε εντολή διαρκεί πολλούς κύκλους αλλά μια εντολή πρέπει να τελειώσει πριν προσκομιστεί μια άλλη. Στην οργάνωση αυτή, μια εντολή περνάει μόνο μέσω των σταδίων που πραγματικά χρειάζεται (π.χ. η sw διαρκεί μόνο τέσσερις κύκλους επειδή δε χρειάζεται το στάδιο WB). **Συγκρίνετε τους χρόνους κύκλου ρολογιού και τους χρόνους εκτέλεσης της οργάνωσης ενός κύκλου, της οργάνωσης πολλών κύκλων, και της οργάνωσης με διοχέτευση**.

Άσκηση 4.12 (συνέχεια)

- Ηδη υπολογίσαμε τους χρόνους ρολογιού για τις οργανώσεις με τη διοχέτευση και με τον ένα μεγάλο κύκλο στο 4.12.1. Η νέα οργάνωση πολλαπλών κύκλων έχει **ίδιο κύκλο ρολογιού με τη διοχέτευση**.

	Χωρίς διοχέτευση (ενός κύκλου)	Πολλών Κύκλων	Με διοχέτευση
α.	1650ps	500ps	500ps
β.	800ps	200ps	200ps

- Θα υπολογίσουμε τους χρόνους εκτέλεσης σε σχέση με την οργάνωση με διοχέτευση (που είναι αυτή με το μικρότερο χρόνο εκτέλεσης). Στον έναν κύκλο, κάθε εντολή διαρκεί έναν (μεγάλο) κύκλο. Στη διοχέτευση, ένα πρόγραμμα (χωρίς καθυστερήσεις) ολοκληρώνει μία εντολή σε κάθε κύκλο. Τέλος, σε μια οργάνωση πολλών κύκλων, μια εντολή sw θέλει 4 κύκλους (χωρίς WB), μια εντολή ALU σε 4 κύκλους (χωρίς MEM), και μία beq θέλει 3 κύκλους (χωρίς MEM και WB). Συνεπώς, η επιτάχυνση που δίνει η διοχέτευση είναι:

Άσκηση 4.12 (συνέχεια)

- «**Επιτάχυνση**» που προσφέρει η διοχέτευση έναντι των σχεδιάσεων ενός και πολλών κύκλων

	Χρόνος εκτέλεσης οργάνωσης πολλών κύκλων σε σχέση με την οργάνωση με διοχέτευση (ίδιος κύκλος ρολογιού άρα η διαφορά είναι το CPI)	**Χρόνος εκτέλεσης οργάνωσης ενός κύκλου σε σχέση με την οργάνωση με διοχέτευση** (ίδιο CPI=1 άρα η διαφορά είναι στη διάρκεια του κύκλου)
α.	0,15 x 5 + 0,60 x 4 + 0,25 x 3 = 3,90	1650ps/500ps = 3,30
β.	0,30 x 5 + 0,45 x 4 + 0,25x3 = 4,05	800ps/200ps = 4,00

Άσκηση 4.13

- Στην άσκηση αυτή, εξετάζουμε πώς επηρεάζουν την εκτέλεση οι εξαρτήσεις δεδομένων στη βασική διοχέτευση των πέντε σταδίων που περιγράψαμε στην Ενότητα 4.5. Τα προβλήματα αυτής της άσκησης αναφέρονται στην παρακάτω ακολουθία εντολών:

	Ακολουθία εντολών

α.	`lw $1,40($6)`
	`add $6,$2,$2`
	`sw $6,50($1)`

β.	`lw $5,-16($5)`
	`sw $5,-16($5)`
	`add $5,$5,$5`

Άσκηση 4.13 (συνέχεια)

- **4.13.1** Δείξτε τις εξαρτήσεις και τους τύπους τους.
- Συμβολισμοί:
- RAW – read after write
- WAR – write after read
- WAW – write after write

	Ακολουθία εντολών	Εξαρτήσεις

α.	`I1: lw $1,40($6)`
	`I2: add $6,$2,$2`
	`I3: sw $6,50($1)`
		RAW στον $1 από I1 προς I3
		RAW στον $6 από I2 προς I3
		WAR στον $6 από I1 προς I2

β.	`I1: lw $5,-16($5)`
	`I2: sw $5,-16($5)`
	`I3: add $5,$5,$5`
		RAW στον $5 από I1 προς I2 και I3
		WAR στον $5 από I1 και I2 προς I3
		WAW στον $5 από I1 προς I3

Άσκηση 4.13 (συνέχεια)

- **4.13.2** Υποθέστε ότι **δεν υπάρχει προώθηση** σε αυτόν τον επεξεργαστή με διοχέτευση. Δείξτε τους κινδύνους και προσθέστε εντολές **nop** (no operation – απραξία) για να τους εξαλείψετε.
- Στη βασική διοχέτευση των 5 σταδίων οι **εξαρτήσεις WAR και WAW δεν προκαλούν κινδύνους**. Χωρίς προώθηση, κάθε εξάρτηση RAW μεταξύ μιας εντολής και των δύο επόμενων εντολών αποτελεί κίνδυνο (αν η ανάγνωση κατ/τη συμβαίνει στο δεύτερο μισό του κύκλου και η εγγραφή στο πρώτο μισό). Ο κώδικας που εξαλείφει αυτούς τους κινδύνους με εισαγωγή nop είναι:

Άσκηση 4.13 (συνέχεια)

	Ακολουθία εντολών	
α.	I1: lw $1,40($6) I2: add $6,$2,$2 nop nop I3: sw $6,50($1)	Καθυστερεί η εκκίνηση της I3 για να αποφευχθεί ο κίνδυνος RAW στον $1 από την I1 και ο κίνδυνος RAW στον $6 από I2
β.	I1: lw $5,-16($5) nop nop I2: sw $5,-16($5) I3: add $5,$5,$5	Καθυστερεί η εκκίνηση της I2 για να αποφευχθεί ο κίνδυνος RAW στον $5 από την I1 Σημείωση: τώρα δεν υπάρχει κίνδυνος RAW στον $5 από τη I1

Άσκηση 4.13 (συνέχεια)

- **4.13.3** Υποθέστε ότι **υπάρχει πλήρης προώθηση**. Δείξτε τους κινδύνους και προσθέστε εντολές nop για να τους εξαλείψετε.

	Ακολουθία εντολών	

α.	`I1: lw $1,40($6)`
	`I2: add $6,$2,$2`
	`I3: sw $6,50($1)`
		Δεν υπάρχει κίνδυνος RAW στον $1 από τη I1 (προωθείται)

β.	`I1: lw $5,-16($5)`
	`nop`
	`I2: sw $5,-16($5)`
	`I3: add $5,$5,$5`
	Γιατί χρειάζεται το nop?
	
	Αφού υπάρχει προώθηση από lw σε sw??
		Καθυστέρηση της I2 για να αποφευχθεί ο κίνδυνος RAW στον $5 από τη I1
		
		Η τιμή του $5 προωθείται τώρα από τη I2
		
		Σημείωση: δεν υπάρχει τώρα κίνδυνος RAW στον $5 από τη I1

Άσκηση 4.13 (συνέχεια)

- Τα υπόλοιπα προβλήματα της άσκησης υποθέτουν τους επόμενους χρόνους κύκλου ρολογιού:

	Χωρίς προώθηση	Με πλήρη προώθηση	Μόνο με προώθηση από την ALU στην ALU (μερική προώθηση…)
α.	300ps	400ps	360ps
β.	200ps	250ps	220ps

Άσκηση 4.13 (συνέχεια)

- **4.13.4** Ποιος είναι ο συνολικός χρόνος εκτέλεσης των ακολουθιών εντολών της άσκησης αυτής χωρίς προώθηση και με πλήρη προώθηση;
- Ποια είναι η επιτάχυνση που επιτυγχάνεται με την προσθήκη πλήρους προώθησης σε μια διοχέτευση που δεν είχε καθόλου προώθηση;

Άσκηση 4.13 (συνέχεια)

- Ο χρόνος εκτέλεσης είναι ίσος με το χρόνο κύκλου ρολογιού επί τον αριθμό των κύκλων. Χωρίς καθυστερήσεις, μια ακολουθία τριών εντολών διαρκεί 7 κύκλους (3+4). Η εκτέλεση χωρίς προώθηση πρέπει να προσθέσει μία καθυστέρηση για κάθε nop που είδαμε στο 4.13.2 και η εκτέλεση με προώθηση πρέπει να προσθέσει μια καθυστέρηση για κάθε nop που είδαμε στο 4.13.3. Έτσι έχουμε:

	Χωρίς προώθηση	Με πλήρη προώθηση	Επιτάχυνση λόγω προώθησης
α.	(7+2) x 300ps = = 2700ps	7 x 400 = = 2800ps	0,964 (**επιβράδυνση…**)
β.	(7+2) x 200ps = = 1800ps	(7+1) x 250ps = = 2000ps	0,90 (**επιβράδυνση…**)

Άσκηση 4.13 (συνέχεια)

- **4.13.5** Στο κώδικα αυτόν προσθέστε εντολές nop για να εξαλείψετε τους κινδύνους στην περίπτωση που υπάρχει **μόνο** προώθηση από την ALU στην ALU (καθόλου προώθηση από το στάδιο MEM προς άλλα στάδια).
- Με προώθηση μόνο από την ALU προς την ALU, μια εντολή ALU μπορεί να προωθήσει στην επόμενη εντολή αλλά όχι σε μια δεύτερη εντολή που ακολουθεί (γιατί αυτό απαιτεί προώθηση από το MEM στο EX **που δεν υπάρχει**). Μια εντολή load δεν προωθεί καθόλου, γιατί καθορίζει την τιμή των δεδομένων στο στάδιο MEM, και είναι τότε πολύ αργά για να γίνει προώθηση από ALU σε ALU. Έχουμε:

Άσκηση 4.13 (συνέχεια)

	Ακολουθία εντολών	
α.	I1: lw $1,40($6) I2: add $6,$2,$2 nop I3: sw $6,50($1)	Δεν μπορεί να γίνει χρήση προώθησης ALU-ALU ($1 φορτώνεται από τη μνήμη στο στάδιο MEM).
β.	I1: lw $5,-16($5) nop nop I2: sw $5,-16($5) I3: add $5,$5,$5	Δεν μπορεί να γίνει χρήση προώθησης ALU-ALU ($5 φορτώνεται από τη μνήμη στο στάδιο MEM).

Άσκηση 4.13 (συνέχεια)

- **4.13.6** Ποιος είναι ο συνολικός χρόνος εκτέλεσης αυτής της ακολουθίας εντολών μόνο με προώθηση από την ALU στην ALU; Ποια είναι η επιτάχυνση σε σχέση με μια διοχέτευση χωρίς προώθηση;

	Χωρίς προώθηση	Με προώθηση ALU-ALU μόνο	Επιτάχυνση λόγω προώθησης από ALU-ALU
α.	(7+2) x 300ps = = 2700ps	(7+1) x 360ps = = 2880ps	0,94 (**επιβράδυνση…**)
β.	(7+2) x 200ps = = 1800ps	(7+2) x 220ps = = 1980ps	0,91 (**επιβράδυνση…**)

Άσκηση 4.14

- Στην άσκηση αυτή, εξετάζουμε πώς μπορούν να επηρεάσουν την εκτέλεση με διοχέτευση οι κίνδυνοι πόρων (ή δομής), οι κίνδυνοι ελέγχου, και η σχεδίαση της αρχιτεκτονικής συνόλου εντολών. Τα προβλήματα της άσκησης αναφέρονται στο τμήμα κώδικα MIPS:

	Ακολουθία εντολών

α.	`lw  $1,40($6)`
	`beq $2,$0,Label ; υποθέστε ότι $2 == $0`
	`sw  $6,50($2)`
	`Label: add $2,$3,$4`
	`sw  $3,50($4)`

β.	`lw $5,-16($5)`
	`sw $4,-16($4)`
	`lw $3,-20($4)`
	`beq $2,$0,Label ; υποθέστε ότι $2 != $0`
	`add $5,$1,$4`

Άσκηση 4.14 (συνέχεια)

- **4.14.1** Για το πρόβλημα αυτό, υποθέστε ότι όλες οι διακλαδώσεις προβλέπονται τέλεια (αυτό εξαλείφει όλους τους κινδύνους ελέγχου), και ότι δε χρησιμοποιούνται καθόλου υποδοχές καθυστέρησης (delay slots). Αν έχουμε **μόνο μία μνήμη** (και για τις εντολές και για τα δεδομένα), υπάρχει ένας δομικός κίνδυνος κάθε φορά που πρέπει να προσκομιστεί μια εντολή στον ίδιο κύκλο που μια άλλη εντολή προσπελάζει δεδομένα. Για να είναι εγγυημένη η πρόοδος του προγράμματος, αυτός ο κίνδυνος πρέπει να επιλύεται πάντα υπέρ της εντολής που προσπελάζει δεδομένα. Ποιος είναι ο συνολικός χρόνος εκτέλεσης αυτής της ακολουθίας εντολών σε μια διοχέτευση πέντε σταδίων που έχει μόνο μία μνήμη;

Άσκηση 4.14 (συνέχεια)

- Το *** δείχνει μια καθυστέρηση (stall) όταν μια εντολή δεν μπορεί να προσκομιστεί διότι μια εντολή load ή store χρησιμοποιεί τη μνήμη στον ίδιο κύκλο :

	Ακολουθία εντολών	Στάδια διοχέτευσης											
α.	`lw $1,40($6)`	IF	ID	EX	MEM	WB							
	`beq $2,$0,Label`		IF	ID	EX	MEM	WB						**11 κύκλοι**
	`sw $6,50($2)`			IF	ID	EX	MEM	WB					
	`add $2,$3,$4`				***	IF	ID	EX	MEM	WB			
	`sw $3,50($4)`						***	IF	ID	EX	MEM	WB	
β.	`lw $5,-16($5)`	IF	ID	EX	MEM	WB							
	`sw $4,-16($4)`		IF	ID	EX	MEM	WB						
	`lw $3,-20($4)`			IF	ID	EX	MEM	WB					**12 κύκλοι**
	`beq $2,$0,Label`				***	***	***	IF	ID	EX	MEM	WB	
	`add $5,$1,$4`								IF	ID	EX	MEM	WB

Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.14 (συνέχεια)

- Έχουμε δει ότι οι κίνδυνοι δεδομένων μπορούν να εξαλειφθούν **με την προσθήκη εντολών nop** στο κώδικα. **Μπορείτε να κάνετε το ίδιο με αυτόν το δομικό κίνδυνο; Γιατί**;
- **Δεν** μπορούμε να προσθέσουμε εντολές nop στον κώδικα για να εξαλείψουμε αυτό τον κίνδυνο – οι εντολές nop πρέπει να προσκομιστούν ακριβώς όπως κάθε εντολή.
- Αυτός ο κίνδυνος αντιμετωπίζεται μόνο με ειδικό hardware που τον ανιχνεύει και καθυστερεί την προσκόμιση της επόμενης εντολής.

Άσκηση 4.14 (συνέχεια)

- **4.14.2** Για το πρόβλημα αυτό, υποθέστε ότι όλες οι διακλαδώσεις προβλέπονται τέλεια (αυτό εξαλείφει όλους τους κινδύνους ελέγχου), και ότι δε χρησιμοποιούνται καθόλου υποδοχές καθυστέρησης (delay slots). Αν αλλάξουμε τις εντολές load και store ώστε να χρησιμοποιούν έναν καταχωρητή ως διεύθυνση (χωρίς σχετική διεύθυνση — offset), αυτές οι εντολές δε χρειάζεται πλέον να χρησιμοποιήσουν την ALU. Ως αποτέλεσμα, **τα στάδια MEM και EX μπορούν να επικαλυφθούν** και η διοχέτευση έχει μόνο τέσσερα στάδια. Αλλάξτε τον κώδικα λαμβάνοντας υπόψη αυτή την τροποποιημένη αρχιτεκτονική συνόλου εντολών. Με την παραδοχή ότι η αλλαγή δεν επιδρά στο χρόνο κύκλου ρολογιού, ποια επιτάχυνση επιτυγχάνεται γι' αυτή την ακολουθία εντολών;

Άσκηση 4.14 (συνέχεια)

	Ακολουθία εντολών	Στάδια διοχέτευσης

α.	`addi $6,$6,40`
	`lw $1,($6)`
	`beq $2,$0,Label`
	`addi $2,$2,50`
	`sw $6,($2)`
	`add $2,$3,$4`
	`addi $4,$4,50`
	`sw $3,($4)`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`

	Εντολές που εκτελούνται	Κύκλοι με 5 στάδια (χωρίς κινδ. δομής)	Κύκλοι με 4 στάδια	Επιτάχυνση ☹
α.	5 -> 8	5 + 4 = 9	8 + 3 = 11	9/11 = 0,818

Άσκηση 4.14 (συνέχεια)

- Ομοίως …

	Ακολουθία εντολών	Στάδια διοχέτευσης

β.	`addi $5,$5,-16`
	`lw $5,($5)`
	`addi $4,$4,-16`
	`sw $4,($4)`
	`addi $4,$4,-20`
	`lw $3,($4)`
	`beq $2,$0,Label`
	`add $5,$1,$4`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`
		`IF ID NEW WB`

	Εντολές που εκτελούνται	Κύκλοι με 5 στάδια (χωρίς κινδ. δομής)	Κύκλοι με 4 στάδια	Επιτάχυνση ☹
β.	5 -> 8	5 + 4 = 9	8 + 3 = 9	9/11 = 0,818

Άσκηση 4.14 (συνέχεια)

- **4.14.3** Με την παραδοχή ότι υπάρχει καθυστέρηση σε περίπτωση εντολής διακλάδωσης (stall-on-branch) και δεν υπάρχουν υποδοχές καθυστέρησης, ποια επιτάχυνση επιτυγχάνεται σε αυτόν το κώδικα αν τα αποτελέσματα των διακλαδώσεων προσδιορίζονται στο στάδιο ID, σε σχέση με την εκτέλεση όπου τα αποτελέσματα των διακλαδώσεων προσδιορίζονται στο στάδιο EX;

Άσκηση 4.14 (συνέχεια)

- Η λύση stall-on-branch καθυστερεί την προσκόμιση της επόμενης εντολής μέχρι να επιλυθεί η διακλάδωση. Όταν η επίλυση γίνεται στο στάδιο EXE, κάθε διακλάδωση προκαλεί δύο κύκλους καθυστέρησης. Όταν η επίλυση γίνεται στο ID προκαλεί μόνο έναν κύκλο καθυστέρησης. Αν δεν υπάρχουν καθυστερήσεις διακλάδωσης (τέλεια πρόβλεψη διακλάδωσης) δεν υπάρχουν χαμένοι κύκλοι.

	Εντολές που εκτελούνται	Διακλαδώσεις	Κύκλοι με επίλυση στο στάδιο EXE	Κύκλοι με επίλυση στο στάδιο ID	Επιτάχυνση
α.	5	1	11	10	11/10 = 1,1
β.	5	1	11	10	11/10 = 1,1

Άσκηση 4.14 (συνέχεια)

- Τα παρακάτω προβλήματα αυτής της άσκησης υποθέτουν ότι τα μεμονωμένα στάδια της διοχέτευσης έχουν τους επόμενους λανθάνοντες χρόνους:

	IF	ID	EX	MEM	WB
α.	100ps	120ps	90ps	130ps	60ps
β.	180ps	100ps	170ps	220ps	60ps

- **4.14.4** Με δεδομένους αυτούς τους χρόνους των σταδίων, επαναλάβετε τον υπολογισμό της επιτάχυνσης της Άσκησης 4.14.2, αλλά λάβετε υπόψη σας την (πιθανή) αλλαγή στο χρόνο κύκλου ρολογιού. Όταν τα EX και MEM γίνονται σε ένα στάδιο, η περισσότερη δουλειά τους μπορεί να γίνει παράλληλα. Ως αποτέλεσμα, το στάδιο EX/MEM (=NEW) που προκύπτει έχει λανθάνοντα χρόνο ίσο με το μεγαλύτερο των αρχικών δύο σταδίων, συν 20ps που χρειάζονται για ό,τι δεν μπορεί να γίνει παράλληλα.

Άσκηση 4.14 (συνέχεια)

- Ο αριθμός κύκλων για την κανονική (5 σταδια) και τη νέα (4 στάδια) διοχέτευση έχει υπολογιστεί στο 4.14.2.
- Ο κύκλος ρολογιού είναι ίσος με το χρόνο του μεγαλύτερου σταδίου. Ο συνδυασμός των EX και MEM επιδρά στο χρονο του κύκλου μόνο αν το συνδυασμένο στάδιο είναι το μεγαλύτερο στη διοχέτευση:

	Χρόνος κύκλου με 5 στάδια	Χρόνος κύκλου με 4 στάδια	Επιτάχυνση ☹
α.	130 ps (MEM)	150 ps (MEM + 20)	(9 x 130)/(11 x 150) = **0,709**
β.	220 ps (MEM)	240 ps (MEM + 20)	(9 x 220)/(11 x 240) = **0,750**

Άσκηση 4.14 (συνέχεια)

- **4.14.5** Με δεδομένους αυτούς τους λανθάνοντες χρόνους των σταδίων της διοχέτευσης, επαναλάβετε τον υπολογισμό της επιτάχυνσης της Άσκησης 4.14.3, αλλά λάβετε υπόψη σας την (πιθανή) αλλαγή στο χρόνο κύκλου ρολογιού. Υποθέστε ότι ο λανθάνων χρόνος του σταδίου ID αυξάνεται κατά 50% και ο λανθάνων χρόνος του σταδίου EX μειώνεται κατά 10ps όταν η επίλυση του αποτελέσματος της διακλάδωσης μεταφέρεται από το EX στο ID.

	Νέος χρόνος ID	Νέος χρόνος EX	Νέος χρόνος κύκλου	Παλιός χρόνος κύκλου	Επιτάχυνση
α.	180 ps (= 120ps + 50%)	80 ps	180 ps (ID)	130 ps (MEM)	**(11 x 130)/(10 x 180) = 0,794**
β.	150 ps (= 100ps + 50%	160 ps	220 ps (MEM)	220 ps (MEM)	**(11 x 220)/(10 x 220) = 1,10**

Άσκηση 4.14 (συνέχεια)

- **4.14.6** Με την παραδοχή καθυστέρησης σε περίπτωση διακλάδωσης (stall-on-branch) και χωρίς υποδοχές καθυστέρησης, ποιος είναι ο νέος χρόνος κύκλου ρολογιού και ο χρόνος εκτέλεσης αυτής της ακολουθίας εντολών αν ο υπολογισμός διεύθυνσης της beq μεταφερθεί στο στάδιο MEM; Ποια είναι η επιτάχυνση από αυτή την αλλαγή; Υποθέστε ότι ο λανθάνων χρόνος του σταδίου EX μειώνεται κατά 20ps και ο λανθάνων χρόνος του σταδίου MEM είναι αμετάβλητος όταν η κατεύθυνση της διακλάδωσης μεταφέρεται από το EX στο MEM.

Άσκηση 4.14 (συνέχεια)

- Ο κύκλος ρολογιού δεν αλλάζει: μια μείωση 20ps στο χρόνο του EX δεν επηρεάζει κάτι αφού το EX δεν είναι το χειρότερο στάδιο. Η αλλαγή **επηρεάζει το χρόνο εκτέλεσης** αφού προσθέτει έναν κύκλο καθυστέρησης σε κάθε διακλάδωση. Επειδή ο χρόνος κύκλου δε βελτιώνεται αλλά το πλήθος των κύκλων αυξάνεται, η επιτάχυνση από αυτή την αλλαγή θα είναι <1 (επιβράδυνση). Στο 4.14.3 υπολογίσαμε ήδη τους κύκλους όταν μια διακλάδωση επιλύεται στο EX. Έχουμε:

	Κύκλοι με επίλυση στο EX	Χρόνος εκτέλ. (επιλ. στο EX)	Κύκλοι με επίλυση στο MEM	Χρόνος εκτέλ. (επιλ. στο MEM)	Επιτάχυνση
α.	11 (9+2)	11 x 130 = 1430 ps	12 (9+3)	12 x 130 = 1560 ps	**0,917**
β.	11 (9+2)	11 x 220 = 2420 ps	12 (9+3)	12 x 220 = 2640 ps	**0,917**

Άσκηση 4.15

- Στην άσκηση αυτή, εξετάζουμε τον τρόπο που η **αρχιτεκτονική συνόλου εντολών** (ISA) επιδρά στη σχεδίαση της διοχέτευσης. Τα προβλήματα της άσκησης αναφέρονται στην παρακάτω νέα εντολή:

α.	`bezi (Rs),Label` **branch on zero word with index**	αν Mem[Rs] = 0 τότε PC=PC+Offs
β.	`swi Rd,Rs(Rt)` **store word with index**	Mem[Rs+Rt]=Rd

Άσκηση 4.15 (συνέχεια)

- **4.15.1** Τι πρέπει να αλλάξει στη διαδρομή δεδομένων με διοχέτευση για να προστεθεί αυτή η εντολή στην αρχιτεκτονική συνόλου εντολών του MIPS;
- **α.** `bezi (Rs),Label`
- Η εντολή λειτουργεί σαν **load με μηδενικό offset** μέχρι να προσκομίσει την τιμή από τη μνήμη δεδομένων. Για να γίνει αυτό **πρέπει ο πολυπλέκτης πριν την ALU να έχει κι άλλη μια είσοδο (μηδέν)**. Αφού διαβαστεί η τιμή από τη μνήμη πρέπει να συγκριθεί με το μηδέν. Αυτό πρέπει να γίνει είτε ταχύτατα στο στάδιο WB, είτε πρέπει να προστεθεί ένα επιπλέον στάδιο μεταξύ MEM και WB. Το αποτέλεσμα αυτής της σύγκρισης με το μηδέν πρέπει να χρησιμοποιηθεί για να ελέγξει τον πολ/κτη της δνσης διακλάδωσης. **Αυτό σημαίνει ότι η επιλογή τη δνσης διακλάδωσης θα καθυστερήσει μέχρι το στάδιο WB**.

Άσκηση 4.15 (συνέχεια)

- **β.** `swi Rd,Rs(Rt)`
- Πρέπει να υπολογίσουμε τη δνση μνήμης χρησιμοποιώντας δύο καταχωρητές, άρα ο υπολογισμός της δνσης είναι ίδιος με μια πρόσθεση στην ALU. Ωστόσο, χρειαζόμαστε τώρα να διαβάσουμε έναν τρίτο καταχωρητή, και **το αρχείο καταχωρητών πρέπει να επεκταθεί με μια τρίτη θύρα (έξοδο) ανάγνωσης**. Επίσης η είσοδος δεδομένων εγγραφής της μνήμης δεδομένων πρέπει να έχει έναν πολυπλέκτη στο στάδιο EX που θα επιλέγει μεταξύ αυτής της τρίτης τιμής από το αρχείο καταχωρητών και της κανονικής εισόδου εγγραφής μια εντολής sw.

Άσκηση 4.15 (συνέχεια)

- **4.15.2** Ποια νέα σήματα ελέγχου πρέπει να προστεθούν στη διοχέτευσή σας της Άσκησης 4.15.1;
- **α.** `bezi (Rs),Label`

Πρέπει να προστεθεί άλλο ένα bit ελέγχου για τον έλεγχο του πολυπλέκτη πριν την ALU (που προσθέτει το μηδενικό offset). Επίσης, χρειάζεται άλλο ένα σήμα παρόμοιο με το Branch σήμα για να ελέγχει αν το νέο αποτέλεσμα ελέγχου με το μηδέν επιτρέπεται να αλλάξει τον PC (η διακλάδωση είναι taken).

- **β.** `swi Rd,Rs(Rt)`

Χρειαζόμαστε ένα σήμα ελέγχου για τον έλεγχο του νέου πολυπλέκτη στο στάδιο EX (που επιλέγει ποιος καταχωρητής θα γραφεί στη μνήμη).

Άσκηση 4.15 (συνέχεια)

- **4.15.3** Εισάγει κάποιους νέους κινδύνους η υποστήριξη αυτής της εντολής; Γίνονται χειρότερες οι καθυστερήσεις εξαιτίας υπαρχόντων κινδύνων;
- **α.** `bezi (Rs),Label`
- Η εντολή αυτή εισάγει **έναν νέο κίνδυνο ελέγχου**. Ο νέος PC για τη διακλάδωση αυτή υπολογίζεται μόνο μετά το στάδιο MEM. Αν προστεθεί νέο στάδιο μετά το MEM, αυτό τότε είτε προσθέτει νέα μονοπάτια προώθησης (από το νέο στάδιο στο EX) ή (αν δεν υπάρχει προώθηση) κάνει τις καθυστερήσεις λόγω κινδύνων δεδομένων κατά έναν κύκλο μεγαλύτερες.
- **β.** `swi Rd,Rs(Rt)`
- Αυτή η εντολή **δεν επιδρά στους κινδύνους**. Δεν τροποποιεί καταχωρητές, άρα δεν προκαλεί κινδύνους δεδομένων. Δεν είναι εντολή διακλάδωσης, άρα δεν παράγει κινδύνους ελέγχου. Με την προσθήκη της τρίτης θύρας ανάγνωσης καταχωρητή, **δε δημιουργεί νέες πηγές κινδύνων**.

Άσκηση 4.15 (συνέχεια)

- **4.15.4** Δώστε ένα παράδειγμα του πού μπορεί να είναι χρήσιμη αυτή η εντολή, και μια ακολουθία υπαρχουσών εντολών MIPS που αντικαθίστανται από αυτή.

α.	`bezi (Rs),Label`	`lw Rtmp,0(Rs)` `beq Rtmp,$0,Label`	Η bezi μπορεί να χρησιμοποιηθεί για την εύρεση του μήκους μιας συμβολοσειράς που τερματίζεται με το μηδενικό χαρακτήρα (null).
β.	`swi Rd,Rs(Rt)`	`add Rtmp,Rs,Rt` `sw Rd,0(Rtmp)`	Η swi μπορεί να χρησιμοποιηθεί για αποθήκευση σε ένα στοιχείο πίνακα, όταν ο πίνακας αρχίζει στη δνση Rt και ο Rs χρησιμοποιείται σαν δείκτης στον πίνακα.

Άσκηση 4.15 (συνέχεια)

- **4.15.5** Αν η εντολή αυτή υπάρχει ήδη σε μια παλαιότερη (κληρονομημένη) αρχιτεκτονική συνόλου εντολών, εξηγήστε πώς θα εκτελούνταν σε ένα σύγχρονο επεξεργαστή όπως ο Barcelona της AMD.
- Η εντολή μπορεί να μεταφραστεί σε **απλές μικρολειτουργίες** (micro-operations) που μοιάζουν με MIPS (όπως στο 4.15.4 προηγούμενα). Αυτές μπορούν στη συνέχεια να εκτελεστούν από έναν επεξεργαστή με «κανονική» διοχέτευση.

Άσκηση 4.15 (συνέχεια)

- Το τελευταίο πρόβλημα αυτής της άσκησης υποθέτει ότι κάθε χρήση της νέας εντολής αντικαθιστά το δεδομένο αριθμό αρχικών εντολών, ότι η αντικατάσταση μπορεί να γίνει μία φορά κάθε δεδομένο αριθμό αρχικών εντολών, και ότι κάθε φορά που εκτελείται η νέα εντολή ο δεδομένος αριθμός επιπλέον κύκλων καθυστέρησης προστίθενται στο χρόνο εκτέλεσης του προγράμματος:

	Αντικαθιστά	Μία φορά κάθε	Επιπλέον κύκλοι καθυστέρησης
α.	2	20	1
β.	3	60	0

- **4.15.6** Ποια είναι η επιτάχυνση που επιτυγχάνεται με την προσθήκη αυτής της νέας εντολής; Στον υπολογισμό σας, υποθέστε ότι το CPI του αρχικού προγράμματος (χωρίς τη νέα εντολή) είναι 1.

Άσκηση 4.15 (συνέχεια)

- Θα υπολογίσουμε το χρόνο εκτέλεσης για κάθε διάστημα αντικατάστασης. Ο παλιός χρόνος εκτέλεσης είναι απλά ο αριθμός των εντολών στο διάστημα αντικατάστασης (CPI=1). Ο νέος χρόνος εκτέλεσης είναι ίσος με τον αριθμό των εντολών μετά την αντικατάσταση, συν τον αριθμό των πρόσθετων κύκλων καθυστέρησης. Ο νέος αριθμός εντολών είναι ο αριθμός εντολών στο αρχικό διάστημα αντικατάστασης, συν τη νέα εντολή, μείον τον αριθμό εντολών που αντικαθιστά:

	Νέος χρόνος εκτέλεσης	Παλιός χρόνος εκτέλεσης	Επιτάχυνση
α.	20 – (2 – 1) + 1 = 20	20	1.00
β.	60 – (3 – 1) + 0 = 58	60	1.03

Άσκηση 4.16

- Τα τρία πρώτα προβλήματα αυτής της άσκησης αναφέρονται στην εξής εντολή του MIPS:

	Εντολή
α.	`lw $1,40($6)`
β.	`add $5,$5,$5`

- **4.16.1** Καθώς εκτελείται η εντολή αυτή, τι περιέχει κάθε καταχωρητής που βρίσκεται μεταξύ δύο σταδίων της διοχέτευσης;

Άσκηση 4.16 (συνέχεια)

- **IF/ID**: κρατά το PC + 4 και την εντολή.
- **ID/EX**: κρατά όλα τα σήματα ελέγχου για τα στάδια EX, MEM, WB, το PC + 4, τις δύο τιμές που διαβάζονται από τους Καταχωρητές, τα 16 χαμηλά bit της εντολής (offset) μετά την επέκταση προσήμου, και τα πεδία Rd και Rt (ακόμη και για εντολές που δεν εχουν αυτά τα πεδία).
- **EX/MEM**: κρατά τα σήματα ελέγχου για τα στάδια MEM και WB, το PC + 4 + Offset ακόμη και για εντολές που δεν έχουν Offset), το αποτέλεσμα της ALU και την τιμή της εξόδου Zero, την τιμή που διαβάστηκε από τον δεύτερο καταχωρητή στο στάδιο ID (ακόμη και για εντολές που δε χρειάζονται αυτή την τιμή), και τον αριθμό του καταχωρητή προορισμού (ακόμη και για εντολές που δεν κάνουν εγγραφές σε καταχωρητή – γι' αυτές ο αριθμός αυτός είναι απλά μια τυχαία επιλογή μεταξύ του Rd ή του Rt).
- **MEM/WB**: κρατά τα σήματα ελέγχου του σταδίου WB, την τιμή που διαβάστηκε από τη μνήμη (ή μια τυχαία τιμή αν δεν υπήρχε ανάγνωση μνήμης), το αποτέλεσμα της ALU, και τον αριθμό καταχωρητή προορισμού.

Άσκηση 4.16 (συνέχεια)

- **4.16.2** Ποιοι καταχωρητές πρέπει να διαβαστούν και ποιοι καταχωρητές διαβάζονται πραγματικά;

	Πρέπει να διαβαστούν	Διαβάζονται πραγματικά
α.	`$6`	`$6, $1`
β.	`$5`	`$5 (δύο φορές)`

- **4.16.3** Τι κάνει η εντολή αυτή στα στάδια EX και MEM;

	EX	MEM
α.	`40+$6`	`Φόρτωση τιμής από τη μνήμη`
β.	`$5+$5`	`Τίποτε`

Άσκηση 4.16 (συνέχεια)

- Τα τρία υπόλοιπα προβλήματα αυτής της άσκησης αναφέρονται στον επόμενο βρόχο. Υποθέστε ότι χρησιμοποιείται τέλεια πρόβλεψη διακλάδωσης (καθόλου καθυστερήσεις λόγω κινδύνων ελέγχου), ότι δεν υπάρχουν υποδοχές καθυστέρησης, και ότι η διοχέτευση έχει πλήρη υποστήριξη προώθησης. Επίσης, υποθέστε ότι εκτελούνται πολλές επαναλήψεις του βρόχου πριν την έξοδο από αυτόν.

	Βρόχος		Βρόχος

α.	`Loop: lw $1,40($6)`
	`add $5,$5,$8`
	`add $6,$6,$8`
	`sw $1,20($5)`
	`beq $1,$0,Loop`
		β.	`Loop: add $1,$2,$3`
			`sw $0,0($1)`
			`sw $0,4($1)`
			`add $2,$2,$4`
			`beq $2,$0,Loop`

Άσκηση 4.16 (συνέχεια)

- **4.16.4** Δείξτε ένα διάγραμμα εκτέλεσης διοχέτευσης για την τρίτη επανάληψη αυτού του βρόχου, από τον κύκλο στον οποίο προσκομίζουμε την πρώτη εντολή αυτής της επανάληψης μέχρι και τον κύκλο στον οποίο μπορούμε να προσκομίσουμε την πρώτη εντολή της επόμενης επανάληψης (αλλά χωρίς αυτόν). Δείξτε όλες τις εντολές που βρίσκονται στη διοχέτευση κατά τη διάρκεια αυτών των κύκλων (όχι μόνον αυτές τις τρίτης επανάληψης).

Άσκηση 4.16 (συνέχεια)

	Βρόχος		Βρόχος				
α.	2:add $5,$5,$8		WB				
	2:add $6,$6,$8		**MEM**	WB			
	2:sw $1,20($5)		EX	MEM	**WB**		
	2:beq $1,$0,Loop		ID	EX	MEM	**WB**	
	3:lw $1,40($6)		IF	ID	EX	MEM	WB
	3:add $5,$5,$8			IF	ID	EX	**MEM**
	3:add $6,$6,$8				IF	ID	EX
	3:sw $1,20($5)					IF	ID
	3:beq $1,$0,Loop						IF

Άσκηση 4.16 (συνέχεια)

	Βρόχος		Βρόχος				
β.	2: sw $0,0($1)		WB				
	2: sw $0,4($1)		MEM	**WB**			
	2: add $2,$2,$4		EX	**MEM**	WB		
	2: beq $2,$0,Loop		ID	EX	MEM	**WB**	
	3: add $1,$2,$3		IF	ID	EX	**MEM**	WB
	3: sw $0,0($1)			IF	ID	EX	MEM
	3: sw $0,4($1)				IF	ID	EX
	3: add $2,$2,$4					IF	ID
	3: beq $2,$0,Loop						IF

Άσκηση 4.16 (συνέχεια)

- **4.16.5** Πόσο συχνά (ως ποσοστό όλων των κύκλων) έχουμε έναν κύκλο στον οποίο και τα πέντε στάδια της διοχέτευσης κάνουν χρήσιμο έργο;
- Σε δεδομένο κύκλο, ένα στάδιο δεν κάνει χρήσιμο έργο αν καθυστερεί ή αν η εντολή περνά από το στάδιο χωρίς να κάνει χρήσιμο έργο εκεί. Στο διάγραμμα του 4.16.4, ένα στάδιο καθυστερεί αν το όνομά του δε φαίνεται για κάποιο κύκλο, και τα στάδια στα οποία δεν κάνει κάτι χρήσιμο είναι με χρώμα. Σημειώστε ότι μια εντολή beq κάνει χρήσιμο έργο στο στάδιο MEM γιατί εκεί καθορίζει τη σωστή τιμή του PC για την επόμενη εντολή. Έχουμε:

	Κύκλοι ανά επανάληψη	Κύκλοι με όλα τα στάδια σε χρήσιμο έργο	% κύκλων με όλα τα στάδια σε χρήσιμο έργο
α.	5	1	20%
β.	5	2	40%

Άσκηση 4.16 (συνέχεια)

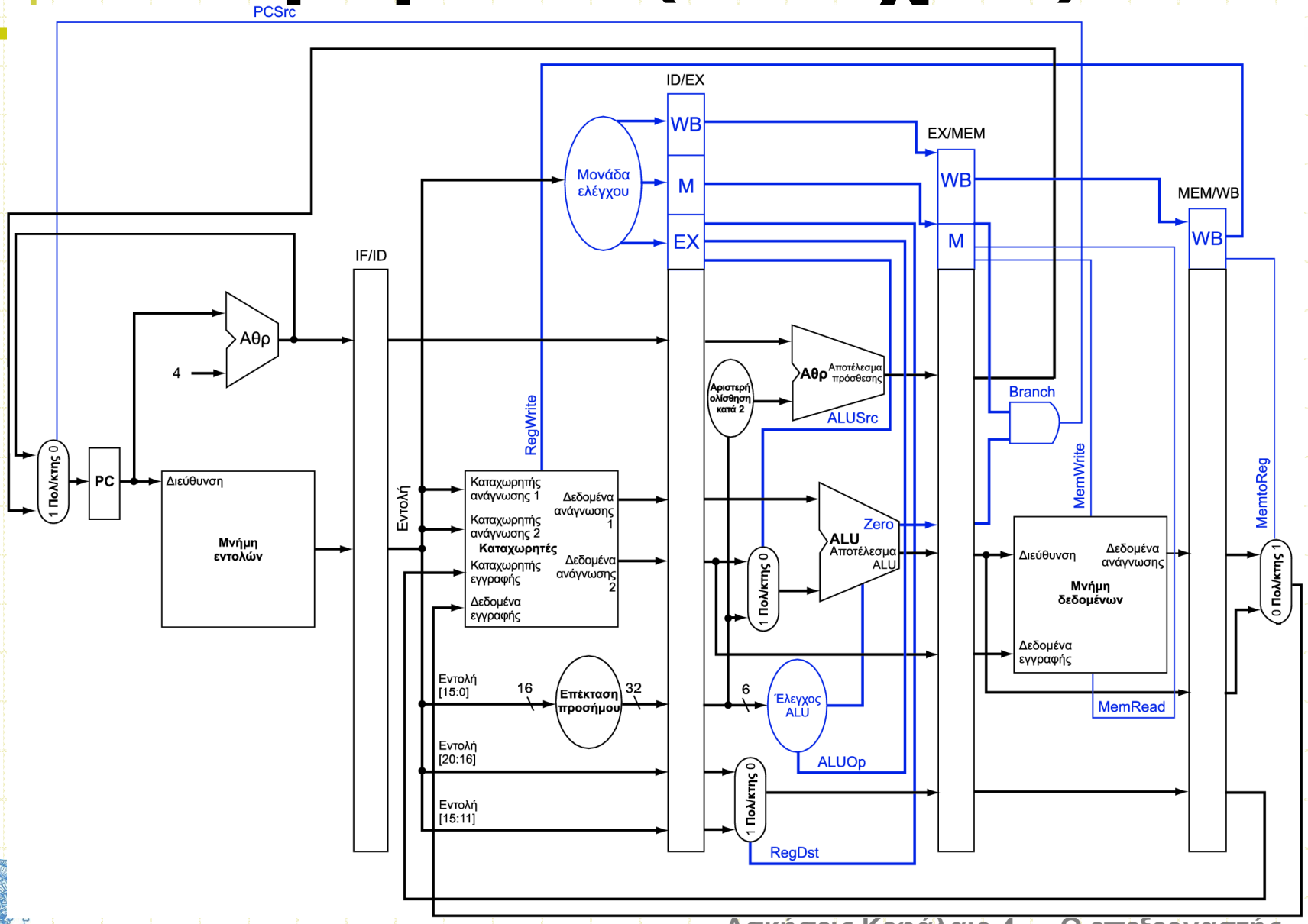
- **4.16.6** Στην αρχή του κύκλου στον οποίο προσκομίζουμε την πρώτη εντολή της τρίτης επανάληψης αυτού του βρόχου, τι αποθηκεύεται στον καταχωρητή IF/ID;
- Η δνση αυτής της πρώτης εντολής της τρίτης επανάληψης (PC+4 για την beq από την προηγούμενη επανάληψη) και η λέξη εντολής της beq από την προηγούμενη επανάληψη.

Άσκηση 4.18

- Τα τρία πρώτα προβλήματα αυτής της άσκησης αναφέρονται στην εκτέλεση της επόμενης εντολής στη διαδρομή δεδομένων με διοχέτευση της Εικόνας 4.51, και υποθέτουν τον επόμενο χρόνο κύκλου ρολογιού, λανθάνοντα χρόνο ALU, και λανθάνοντα χρόνο πολυπλέκτη:

	Εντολή	Χρόνος κύκλου ρολογιού	Λανθάνων χρόνος ALU	Λανθάνων χρόνος Πολυπλέκτη
α.	`add $1,$2,$3`	100ps	80ps	10ps
β.	`slt $2,$1,$3`	80ps	50ps	20ps

Άσκηση 4.18 (συνεχεια)



Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.18 (συνεχεια)

- **4.18.1** Για κάθε στάδιο της διοχέτευσης, ποιες είναι οι τιμές των σημάτων ελέγχου που ενεργοποιούνται από αυτή την εντολή στο συγκεκριμένο στάδιο;
- Στα στάδια IF και ID δεν υπάρχουν ενεργοποιημένα σήματα. Στα άλλα τρία στάδια έχουμε:

	EX	MEM	WB
α.	ALUSrc = 0, ALUOp = 10, RegDst = 1	Branch = 0, MemWrite = 0, MemRead = 0	MemtoReg = 1, RegWrite = 1
β.	ALUSrc = 0, ALUOp = 10, RegDst = 1	Branch = 0, MemWrite = 0, MemRead = 0	MemtoReg = 1, RegWrite = 1

Άσκηση 4.18 (συνεχεια)

- **4.18.2** Πόσο χρόνο διαθέτει η μονάδα ελέγχου για να δημιουργήσει το σήμα ελέγχου ALUSrc; Συγκρίνετε με την οργάνωση ενός κύκλου.

- Διαθέτει έναν κύκλο ρολογιού.

Άσκηση 4.18 (συνεχεια)

- **4.18.3** Ποια είναι η τιμή του σήματος PCSrc γι' αυτή την εντολή; Αυτό το σήμα δημιουργείται νωρίς στο στάδιο MEM (μόνο μία πύλη AND). Ποιος θα ήταν ένας λόγος για να γίνει αυτό στο στάδιο EX; Ποιος είναι ο λόγος να μη γίνει στο στάδιο EX;
- Το PCSrc είναι 0. Ο λόγος **ενάντια** στη δημιουργία του στο EX είναι ότι το AND πρέπει να γίνει αφού η ALU υπολογίσει την έξοδο Zero. Αν το EX είναι το μεγαλύτερο στάδιο και η έξοδος της ALU βρίσκεται στην κρίσιμη διαδρομή, ο πρόσθετος λανθάνων χρόνος μιας AND θα αυξήσει το κύκλο ρολογιού του επεξεργαστή.
- Ο λόγος **υπέρ** της δημιουργίας του στο στάδιο EX είναι ότι ο σωστός επόμενος PC για μια διακλάδωση με συνθήκη μπορεί να υπολογιστεί έναν κύκλο νωρίτερα, κι έτσι να αποφύγουμε έναν κύκλο καθυστέρησης σε περίπτωση κινδύνου ελέγχου.

Άσκηση 4.18 (συνεχεια)

- Τα υπόλοιπα προβλήματα αυτής της άσκησης αναφέρονται στα εξής σήματα της Εικόνας 4.48:

	Σήμα 1	Σήμα 2
α.	RegDst	RegWrite
β.	MemRead	RegWrite

- **4.18.4** Για καθένα από αυτά τα σήματα, προσδιορίστε το στάδιο της διοχέτευσης στο οποίο παράγεται και το στάδιο στο οποίο χρησιμοποιείται.

	Σήμα 1	Σήμα 2
α.	Παράγεται στο ID, χρήση στο EX	Παράγεται στο ID, χρήση στο WB
β.	Παράγεται στο ID, χρήση στο MEM	Παράγεται στο ID, χρήση στο WB

Άσκηση 4.18 (συνεχεια)

- **4.18.5** Σε ποιες εντολές MIPS έχουν και τα δύο αυτά σήματα την τιμή 1;

α.	Εντολές R
β.	Φορτώσεις

- **4.18.6** Ένα από αυτά τα σήματα επιστρέφει μέσω της διοχέτευσης. Ποιο σήμα είναι αυτό; Πρόκειται για το παράδοξο του ταξιδιού στο χρόνο; Εξηγήστε.
- Το σήμα 2 γυρίζει πίσω μέσω της διοχέτευσης. Επηρεάζει τις εντολές που εκτελούνται μετά από αυτή για την οποια δημιουργείται το σήμα, οπότε δεν πρόκειται για παράδοξο.

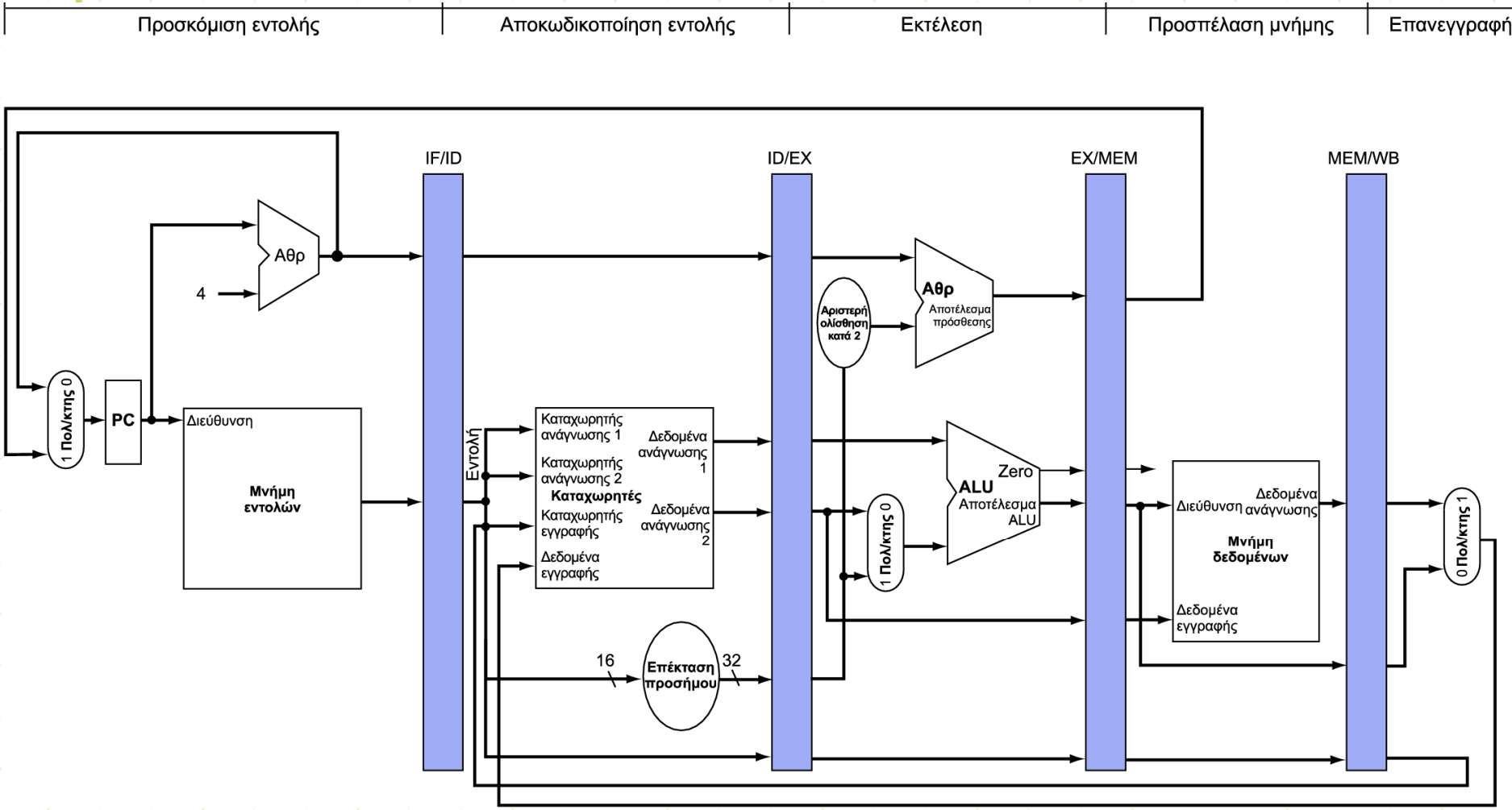
Άσκηση 4.19

- Η άσκηση έχει σκοπό την κατανόηση των συμβιβασμών μεταξύ κόστους, πολυπλοκότητας, και απόδοσης της προώθησης σε έναν επεξεργαστή με διοχέτευση. Αναφέρεται στις διαδρομές δεδομένων με διοχέτευση της Εικόνας 4.45.
- Από όλες τις εντολές που εκτελούνται σε έναν επεξεργαστή, το παρακάτω κλάσμα αυτών των εντολών έχει ένα συγκεκριμένο τύπο εξάρτησης δεδομένων **RAW** (**read after write** — ανάγνωση μετά την εγγραφή).

	EX προς 1η μόνο	EX προς 1η και 2η	EX προς 2η μόνο	MEM προς 1η
α.	10%	10%	5%	25%
β.	15%	5%	10%	20%

- Ο τύπος εξάρτησης RAW προσδιορίζεται από το στάδιο που παράγει το αποτέλεσμα (EX ή MEM) και την εντολή που καταναλώνει το αποτέλεσμα (1η εντολή που ακολουθεί αυτή η οποία παράγει το αποτέλεσμα, 2η εντολή που ακολουθεί, ή και οι δύο). Υποθέτουμε ότι η εγγραφή καταχωρητή γίνεται στο πρώτο μισό του κύκλου ρολογιού και ότι οι αναγνώσεις καταχωρητών γίνονται στο δεύτερο μισό του κύκλου, και έτσι οι εξαρτήσεις «EX προς 3η» και «MEM προς 2η» δε μετρούν επειδή δεν μπορούν να οδηγήσουν σε κίνδυνο δεδομένων. Επίσης, υποθέστε ότι το CPI του επεξεργαστή είναι 1 αν δεν υπάρχουν κίνδυνοι δεδομένων.

Άσκηση 4.19 (συνέχεια)



Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.19 (συνέχεια)

- **4.19.1** Αν δε χρησιμοποιήσουμε προώθηση, σε τι κλάσμα των κύκλων καθυστερούμε εξαιτίας κινδύνων δεδομένων;
- Οι εξαρτήσεις προς την 1η εντολή δίνουν 2 κύκλους καθυστέρησης, και το ίδιο ισχύει όταν υπάρχει εξάρτηση και προς τις δύο επόμενες εντολές. Εξαρτήσεις μόνο προς τη 2η εντολή οδηγούν σε έναν μόνο κύκλο καθυστέρησης. Έχουμε:

	CPI	% κύκλων καθυστέρησης
α.	1 + 0,10*2 + 0,10*2 + 0,05*1 + 0,25*2 = **1,95**	0,95/1,95 = **~49%**
β.	1 + 0,15*2 + 0,05*2 + 0,10*1 + 0,20*2 = **1,90**	0,90/1,90 = **~47%**

Άσκηση 4.19 (συνέχεια)

- **4.19.2** Αν χρησιμοποιήσουμε πλήρη προώθηση (προώθηση όλων των αποτελεσμάτων που μπορούν να προωθηθούν), σε τι κλάσμα των κύκλων καθυστερούμε εξαιτίας κινδύνων δεδομένων;
- Οι μόνες καθυστερήσεις (1 κύκλος) που δεν θα μπορούν να εξαλειφθούν θα οφείλονται στην εξάρτηση από μια εντολή στο στάδιο MEM προς την επόμενη. Άρα:

	CPI	% κύκλων καθυστέρησης
α.	1 + 0,25*1 = **1,25**	0,25/1,25 = **~20%**
β.	1 + 0,20*1 = **1,20**	0,20/1,20 = **~17%**

- Εδώ υποθέτουμε πως δεν υπάρχει προώθηση από MEM σε MEM (δηλαδή από load σε store) διότι δεν γνωρίζουμε τι ποσοστό εξαρτήσεων MEM στην επόμενη εντολή αφορά load σε store.

Άσκηση 4.19 (συνέχεια)

- **4.19.3** Ας υποθέσουμε ότι δεν μπορούμε να αντέξουμε το κόστος πολυπλεκτών με τρεις εισόδους, οι οποίοι χρειάζονται για την πλήρη προώθηση. Πρέπει να αποφασίσουμε αν είναι καλύτερο να προωθούμε μόνον από τον καταχωρητή EX/MEM της διοχέτευσης (προώθηση στον επόμενο κύκλο) ή μόνον από τον καταχωρητή MEM/WB της διοχέτευσης (προώθηση δύο κύκλων μετά). Ποια από τις δύο εναλλακτικές οδηγεί σε λιγότερους κύκλους καθυστέρησης δεδομένων;
- Με προώθηση μόνο από τον EX/MEM, οι εξαρτήσεις EX προς την 1η δεν θα έχουν καθυστερήσεις αλλά οι EX προς τη 2η και MEM προς 1η θα έχουν 1 κύκλο καθυστέρησης. Με προώθηση μόνο προς από τον MEM/WB, οι εξαρτήσεις EX προς 2η δεν θα έχουν καθυστερήσεις. Οι MEM προς 1η θα έχουν έναν κυκλο καθυστέρηση, και οι EX προς 1η θα έχουν έναν κύκλο διότι πρέπει να περιμένουμε την ολοκλήρωση του σταδίου MEM για να γίνει η προώθηση προς την επόμενη εντολή. Υπολογίζουνε τους κύκλους καθυστέρησης ανά εντολή για κάθε περίπτωση:

	EX/MEM	MEM/WB	Επιλέγουμε
α.	0,10+0,05+0,25 = 0,40	0,10+0,10+0,25 = 0,45	EX/MEM
β.	0,05+0,10+0,20 = 0,35	0,15+0,05+0,20 = 0,40	EX/MEM

Άσκηση 4.19 (συνέχεια)

- Τα τρία υπόλοιπα προβλήματα αυτής της άσκησης αναφέρονται στους επόμενους λανθάνοντες χρόνους για μεμονωμένα στάδια της διοχέτευσης. Για το στάδιο EX, οι λανθάνοντες χρόνοι δίνονται ξεχωριστά για έναν επεξεργαστή χωρίς προώθηση και για έναν επεξεργαστή με διαφορετικούς τύπους προώθησης.

	IF	ID	EX (χωρίς προώ-θηση)	EX (με πλήρη προώ-θηση)	EX (προώ-θηση από το EX/MEM μόνο)	EX (προώ-θηση από το MEM/WB μόνο)	MEM	WB
α.	100ps	50ps	75ps	110ps	100ps	100ps	100ps	60ps
β.	250ps	300ps	200ps	350ps	320ps	310ps	300ps	200ps

Άσκηση 4.19 (συνέχεια)

- **4.19.4** Για τις δεδομένες πιθανότητες κινδύνου και τους λανθάνοντες χρόνους των σταδίων της διοχέτευσης, ποια είναι η επιτάχυνση που επιτυγχάνεται με τη προσθήκη πλήρους προώθησης σε μια διοχέτευση που δεν είχε προώθηση;
- Στα 4.19.1 και 4.19.2 έχουμε ήδη υπολογίσει το CPI χωρίς προώθηση και με πλήρη προώθηση. Τώρα μπορούμε να υπολογίσουμε το χρόνο ανά εντολή λαμβάνοντας υπ' όψιν το χρόνο κύκλου ρολογιού, και έπειτα βρίσκουμε την επιτάχυνση:

	Χωρίς προώθηση	Με πλήρη προώθηση	Επιτάχυνση
α.	1.95 × **100ps** = 195ps	1.25 × **110ps** = 137.5ps	**1,42**
β.	1.90 × **300ps** = 570ps	1.20 × **350ps** = 420ps	**1,36**

Άσκηση 4.19 (συνέχεια)

- **4.19.5** Ποια θα ήταν η επιπλέον επιτάχυνση (σε σχέση με έναν επεξεργαστή με προώθηση) αν προσθέταμε ένα μηχανισμό προώθησης στο χρόνο που εξαλείφει όλους τους κινδύνους δεδομένων; Υποθέστε ότι το κύκλωμα ταξιδιού στο χρόνο, που δεν έχει ανακαλυφθεί ακόμη, προσθέτει 100ps στο λανθάνοντα χρόνο του σταδίου EX με πλήρη προώθηση.
- Το «ταξίδι στο χρόνο» θα έκανε το CPI = 1 αφού θα εξάλειφε και τις καθυστερήσεις λόγω load-use κινδύνων. Έτσι:

	Με πλήρη προώθηση	Με το «ταξίδι στο χρόνο»	Επιτάχυνση
α.	1.25 × **110ps** = 137.5ps	1 × **210ps** = 210ps	**0,65**
β.	1.20 × **350ps** = 420ps	1 × **450ps** = 450ps	**0,93**

Άδικα ξοδέψαμε ...

Άσκηση 4.19 (συνέχεια)

- **4.19.6** Επαναλάβετε την Άσκηση 4.19.3, αλλά αυτή τη φορά προσδιορίστε ποια από τις δύο εναλλακτικές οδηγεί σε μικρότερο χρόνο ανά εντολή.
- Έχουμε:

	EX/MEM	MEM/WB	Επιλέγουμε
α.	1,40 x 100ps = 140ps	1,45 x 100ps = 145ps	EX/MEM
β.	1,35 x 320ps = 432ps	1,40 x 310ps = 434ps	EX/MEM

Άσκηση 4.21

- Αυτή η άσκηση έχει σκοπό να σας βοηθήσει να κατανοήσετε τη σχέση μεταξύ προώθησης, ανίχνευσης κινδύνων, και σχεδίασης της αρχιτεκτονικής συνόλου εντολών. Τα προβλήματα της άσκησης αναφέρονται στις παρακάτω ακολουθίες εντολών, και υποθέτουν ότι εκτελούνται σε μια διαδρομή δεδομένων με διοχέτευση πέντε σταδίων:

	Ακολουθία εντολών

α.	`lw  $1,40($6)`
	`add $2,$3,$1`
	`add $1,$6,$4`
	`sw  $2,20($4)`
	`and $1,$1,$4`

β.	`add $1,$5,$3`
	`sw  $1,0($2)`
	`lw  $1,4($2)`
	`add $5,$5,$1`
	`sw  $1,0($2)`

Άσκηση 4.21 (συνέχεια)

4.21.1 Αν δεν υπάρχει προώθηση ή ανίχνευση κινδύνων, προσθέστε εντολές nop για να εξασφαλίσετε την σωστή εκτέλεση.

	Ακολουθία εντολών

α.	`lw  $1,40($6)`
	`nop`
	`nop`
	`add $2,$3,$1`
	`add $1,$6,$4`
	`nop`
	`sw  $2,20($4)`
	`and $1,$1,$4`

	Ακολουθία εντολών

β.	`add $1,$5,$3`
	`nop`
	`nop`
	`sw  $1,0($2)`
	`lw  $1,4($2)`
	`nop`
	`nop`
	`add $5,$5,$1`
	`sw  $1,0($2)`

Άσκηση 4.21 (συνέχεια)

- **4.21.2** Επαναλάβετε την Άσκηση 4.21.1 αλλά τώρα χρησιμοποιήστε εντολές nop μόνον όταν ένας κίνδυνος δεν μπορεί να αποφευχθεί με την αλλαγή ή την αναδιάταξη αυτών των εντολών. Μπορείτε να υποθέσετε ότι ο καταχωρητής $7 είναι δυνατόν να χρησιμοποιηθεί για να διατηρήσει προσωρινές τιμές στον τροποποιημένο σας κώδικα.
- Μπορούμε να μετακινήσουμε μια εντολή προς τα πάνω εναλλάσοντας τη τη θέση της με μια άλλη με την οποία δεν έχει εξάρτηση, κι έτσι μπορούμε να γεμίσουμε κάποιες nop με τέτοιες εντολές. Επίσης μπορούμε να χρησιμοποιήσουμε τον $7 για να εξαλείψουμε τις εξαρτήσεις WAW ή WAR ώστε να μπορούμε να μετακινήσουμε περισσότερες εντολές προς τα πάνω.

Άσκηση 4.21 (συνέχεια)

- **4.21.2** χρήση του $7 για αντικατάσταση του $1 όπου χρειάζεται – και μεταφορά εντολών προς τα πάνω.

	Ακολουθία εντολών		Ακολουθία εντολών
α.	I1: lw **$7**,40($6) I3: add $1,$6,$4 nop I2: add $2,$3,**$7** I5: and $1,$1,$4 nop I4: sw $2,20($4)	**β.**	I1: add **$7**,$5,$3 I3: lw $1,4($2) nop I2: sw **$7**,0($2) I4: add $5,$5,$1 I5: sw $1,0($2)

Άσκηση 4.21 (συνέχεια)

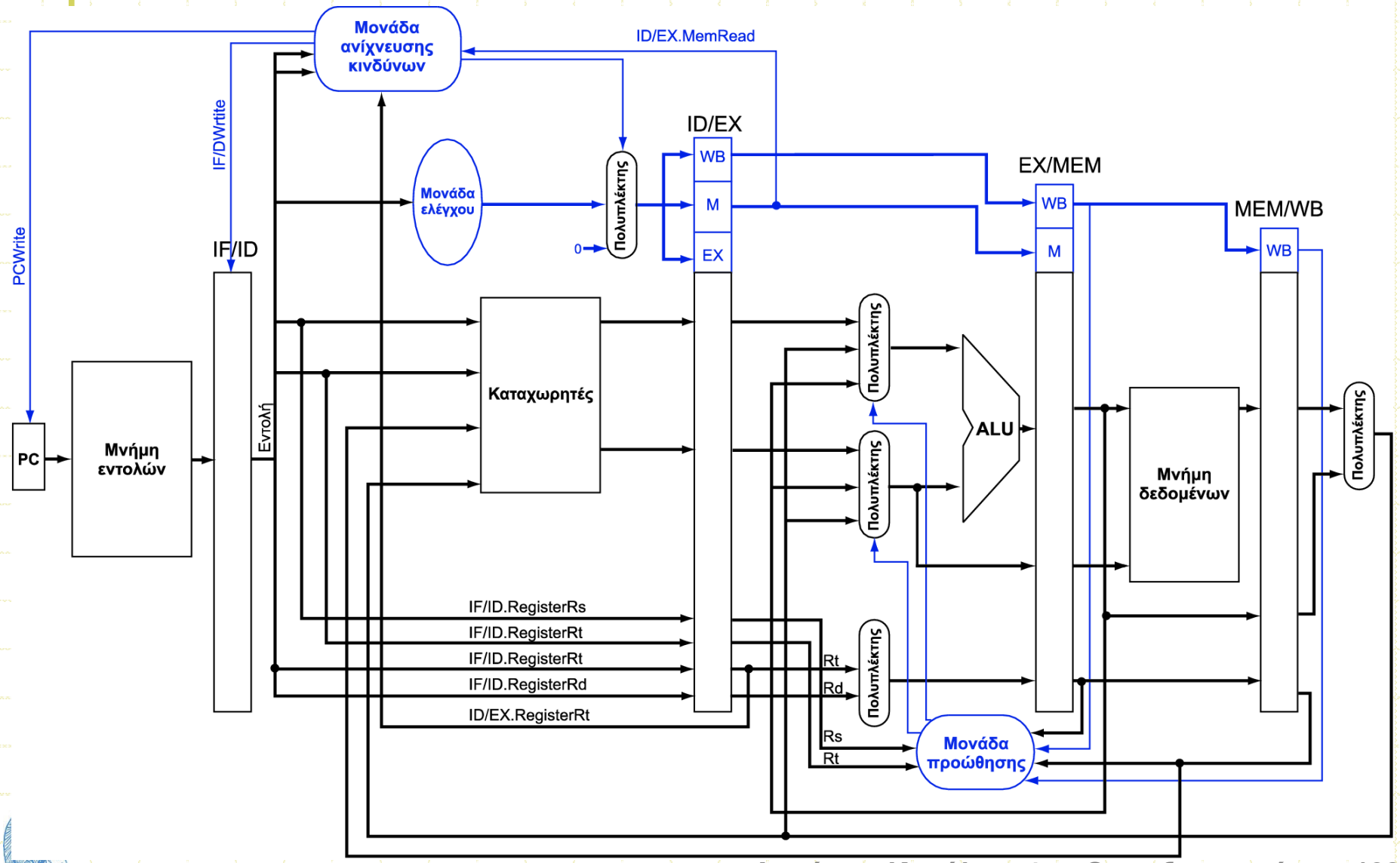
- **4.21.3** Αν ο επεξεργαστής διαθέτει προώθηση, αλλά ξεχάσαμε να υλοποιήσουμε τη μονάδα ανίχνευσης κινδύνων, τι συμβαίνει όταν εκτελείται ο κώδικας;
- Με την προώθηση, η μονάδα ανίχνευσης κινδύνων εξακολουθεί να είναι απαραίτητη, για να εισαγάγει ένα κύκλο καθυστέρησης όποτε μια εντολή load παρέχει μια τιμή στην εντολή που ακολουθεί αμέσως. Χωρίς τη μονάδα ανίχνευσης κινδύνου, η εντολή που εξαρτάται από ένα αμέσως προηγουμενο load παίρνει την «παλιά» τιμή που είχε ο καταχωρητής πριν την εντολή load.

α.	η I2 παίρνει την τιμή του είχε ο $1 πριν από την I1, και όχι από την I1 όπως θα έπρεπε
β.	η I4 παίρνει την τιμή που είχε ο $1 πριν από την I1, και όχι από την I3 όπως θα έπρεπε

Άσκηση 4.21 (συνέχεια)

- **4.21.4** Αν υπάρχει προώθηση, για τους πρώτους πέντε κύκλους κατά τη διάρκεια της εκτέλεσης αυτού του κώδικα, προσδιορίστε ποια σήματα ενεργοποιούνται σε κάθε κύκλο από τις μονάδες ανίχνευσης κινδύνων και προώθησης της Εικόνας 4.60.
- Οι έξοδοι της μονάδας ανίχνευσης κινδύνων είναι: PCWrite, IF/IDWrite, και ID/EXZero (που ελέγχει τον πολ/κτη μετά την έξοδο της μονάδας ελέγχου). Σημειώστε ότι το IF/IDWrite είναι πάντα ίσο με το PCWrite, και το ED/ExZero είναι πάντα το αντίθετο του PCWrite. Έτσι, θα δείξουμε μόνο την τιμή του PCWrite σε κάθε κύκλο. Οι έξοδοι της μονάδας προώθησης είναι ALUin1 και ALUin2, και ελέγχουν τους πολ/κτες που επιλέγουν την πρώτη και δεύτερη είσοδο της ALU. Οι τρεις πιθανές τιμές των ALUin1 ή ALUin2 είναι: 0 (όχι προώθηση), 1 (προώθηση της εξόδου της ALU από την προηγούμενη εντολή, ή 2 (προώθηση τιμής δεδομένων από την προ-προηγούμενη εντολή). Έχουμε:

Άσκηση 4.21 (συνέχεια)



Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.21 (συνέχεια)

	Ακολουθία εντολών	Πρώτοι πέντε κύκλοι 1 2 3 4 5	Σήματα

α.	`lw $1,40($6)`
	`add $2,$3,$1`
	`add $1,$6,$4`
	`sw $2,20($4)`
	`and $1,$1,$4`
		`IF ID EX MEM WB`
		`IF ID *** EX`
		`IF *** ID`
		`IF`
			1: PCWrite = 1, ALUin1 = X, ALUin2 = X
			2: PCWrite = 1, ALUin1 = X, ALUin2 = X
			3: PCWrite = 1, ALUin1 = 0, ALUin2 = 0
			4: PCWrite = 0, ALUin1 = X, ALUin2 = X
			5: PCWrite = 1, ALUin1 = 0, ALUin2 = 2

β.	`add $1,$5,$3`
	`sw $1,0($2)`
	`lw $1,4($2)`
	`add $5,$5,$1`
	`sw $1,0($2)`
		`IF ID EX MEM WB`
		`IF ID EX MEM`
		`IF ID EX`
		`IF ID`
		`IF`
			1: PCWrite = 1, ALUin1 = X, ALUin2 = X
			2: PCWrite = 1, ALUin1 = X, ALUin2 = X
			3: PCWrite = 1, ALUin1 = 0, ALUin2 = 0
			4: PCWrite = 1, ALUin1 = 0, ALUin2 = 1
			5: PCWrite = 1, ALUin1 = 0, ALUin2 = 0

Άσκηση 4.21 (συνέχεια)

- **4.21.5** Αν δεν υπάρχει προώθηση, τι νέα σήματα εισόδου και εξόδου χρειαζόμαστε για τη μονάδα ανίχνευσης κινδύνων της Εικόνας 4.60; Χρησιμοποιώντας αυτή την ακολουθία εντολών ως παράδειγμα, εξηγήστε γιατί χρειάζεται το κάθε σήμα.
- Η εντολή που βρίσκεται στο στάδιο ID πρέπει να καθυστερήσει αν εξαρτάται από την τιμή που παράγεται από την εντολή στο EX ή την εντολή στο MEM. Άρα πρέπει να ελέγξουμε τον καταχωρητή προορισμού αυτών των δύο εντολών. Για την εντολή στο στάδιο EX, πρέπει να ελέγξουμε το Rd των εντολών τύπου R και το Rt των φορτώσεων. Για την εντολή στο στάδιο MEM, ο καταχωρητής προορισμού έχει ήδη επιλεγεί (από τον πολ/κτη του EX) άρα πρέπει να ελέγξουμε εκείνο τον αριθμό καταχωρητή (είναι η κάτω έξοδος του καταχωρητή διοχέτευσης EX/MEM). Οι πρόσθετες είσοδοι στη μονάδα ανίχνευσης κινδύνου είναι ο καταχωρητής Rd από τον ID/EX και ο καταχωρητής εξόδου από τον EX/MEM. Το Rt πεδίο από τον ID/EX είναι πάντα είσοδος στη μονάδα ανίχνευσης κινδύνου της εικόνας 4.60.
- Δε χρειάζονται πρόσθετες έξοδοι. Μπορούμε να καθυστερήσουμε τη διοχέτευση χρησιμοποιώντας τα τρία σήματα εξόδου που έχουμε ήδη.

Άσκηση 4.21 (συνέχεια)

- **4.21.6** Για τη νέα μονάδα ανίχνευσης κινδύνων της Άσκησης 4.21.5, προσδιορίστε ποια σήματα εξόδου ενεργοποιεί σε καθέναν από τους πρώτους πέντε κύκλους κατά τη διάρκεια της εκτέλεσης αυτού του κώδικα.
- Όπως εξηγήσαμε στο 4.21.5, πρέπει μόνο να καθορίσουμε την τιμή του σήματος PCWrite, διότι το IF/IDWrite είναι ίσο με το PCWrite και το σήμα ID/EXzero είναι το αντίθετό του. Έτσι:

Άσκηση 4.21 (συνέχεια)

	Ακολουθία εντολών	Πρώτοι πέντε κύκλοι 1 2 3 4 5	Σήματα

α.	`lw $1,40($6)`
	`add $2,$3,$1`
	`add $1,$6,$4`
	`sw $2,20($4)`
	`and $1,$1,$4`
		`IF ID EX MEM WB`
		`IF ID *** ***`
		`IF *** ***`
		`***`
			1: PCWrite = 1
			2: PCWrite = 1
			3: PCWrite = 1
			4: PCWrite = 0
			5: PCWrite = 0

β.	`add $1,$5,$3`
	`sw $1,0($2)`
	`lw $1,4($2)`
	`add $5,$5,$1`
	`sw $1,0($2)`
		`IF ID EX MEM WB`
		`IF ID EX MEM`
		`IF ID EX`
		`IF ID`
		`IF`
			1: PCWrite = 1
			2: PCWrite = 1
			3: PCWrite = 1
			4: PCWrite = 0
			5: PCWrite = 0

Άσκηση 4.23

- Η σημασία μιας καλής διάταξης πρόβλεψης διακλάδωσης εξαρτάται από τη συχνότητα που εκτελούνται διακλαδώσεις υπό συνθήκη. Μαζί με την ακρίβεια της διάταξης πρόβλεψης, αυτό καθορίζει πόσος χρόνος ξοδεύεται σε καθυστέρηση εξαιτίας λανθασμένων προβλέψεων διακλάδωσης. Σε αυτή την άσκηση, υποθέστε ότι οι δυναμικές εντολές διαιρούνται σε διάφορες κατηγορίες εντολών ως εξής:

	Τύπου R	beq	j	lw	sw
α.	50%	15%	10%	15%	10%
β.	30%	10%	5%	35%	20%

- Επίσης, θεωρήστε τις επόμενες ακρίβειες των διατάξεων πρόβλεψης:

	Πάντα ληφθείσα	Πάντα μη ληφθείσα	2 bit
α.	40%	60%	80%
β.	60%	40%	95%

Άσκηση 4.23 (συνέχεια)

- **4.23.1** Οι κύκλοι καθυστέρησης εξαιτίας λάθος προβλέψεων διακλάδωσης αυξάνουν το CPI. Ποιο είναι το επιπλέον CPI λόγω των λανθασμένων προβλέψεων διακλάδωσης με τη διάταξη πρόβλεψης «πάντα ληφθείσα»; Υποθέστε ότι τα αποτελέσματα της διακλάδωσης καθορίζονται στο στάδιο EX, ότι δεν υπάρχουν κίνδυνοι δεδομένων, και δε χρησιμοποιούνται υποδοχές καθυστέρησης.
- Κάθε διακλάδωση που δεν προβλέπεται σωστά από τη διάταξη πρόβλεψης πάντα-ληφθείσα θα προκαλέσει καθυστέρηση 2 κύκλων, έτσι έχουμε:

	Επιπλέον CPI
α.	$2 \times (1 - 0.40) \times 0.15 = 0.18$
β.	$2 \times (1 - 0.60) \times 0.10 = 0.08$

Άσκηση 4.23 (συνέχεια)

- **4.23.2** Επαναλάβετε την Άσκηση 4.23.1 για τη διάταξη πρόβλεψης «πάντα μη ληφθείσα».
- Κάθε διακλάδωση που δεν προβλέπεται σωστά από τη διάταξη πρόβλεψης πάντα-μη-ληφθείσα θα προκαλέσει καθυστέρηση 2 κύκλων, έτσι έχουμε:

	Επιπλέον CPI
α.	$2 \times (1 - 0.60) \times 0.15 = 0.12$
β.	$2 \times (1 - 0.40) \times 0.10 = 0.12$

Άσκηση 4.23 (συνέχεια)

- **4.23.3** Επαναλάβετε την Άσκηση 4.23.1 για τη διάταξη πρόβλεψης των 2 bit.
- Κάθε διακλάδωση που δεν προβλέπεται σωστά από τη διάταξη πρόβλεψης των 2bit θα προκαλέσει καθυστέρηση 2 κύκλων, έτσι έχουμε:

	Επιπλέον CPI
α.	$2 \times (1 - 0.80) \times 0.15 = 0.06$
β.	$2 \times (1 - 0.95) \times 0.10 = 0.01$

Άσκηση 4.23 (συνέχεια)

- **4.23.4** Με τη διάταξη πρόβλεψης των 2 bit, ποια επιτάχυνση θα μπορούσε να επιτευχθεί αν μπορούσαμε να μετατρέψουμε τις μισές εντολές διακλάδωσης έτσι ώστε μια εντολή διακλάδωσης να αντικαθίσταται από μια εντολή ALU; Υποθέστε ότι οι σωστά και οι λανθασμένα προβλεφθείσες εντολές έχουν την ίδια πιθανότητα να αντικατασταθούν.
- Οι σωστά προβλεφθείσες διακλαδώσεις είχαν CPI = 1 και τώρα γίνονται εντολές ALU των οποίων το CPI είναι παλι 1. Οι λανθασμένα προβλεφθείσες εντολές γίνονται και πάλι εντολές ALU με CPI = 1, έτσι έχουμε:

	CPI χωρίς μετατροπή	CPI με μετατροπή	Επιτάχυνση
α.	$1 + 2 \times (1 - 0.80) \times 0.15 = 1.060$	$1 + 2 \times (1 - 0.80) \times 0.15 \times 0.5 = 1.030$	$1.060/1.030 = 1.029$
β.	$1 + 2 \times (1 - 0.95) \times 0.10 = 1.010$	$1 + 2 \times (1 - 0.95) \times 0.10 \times 0.5 = 1.005$	$1.010/1.005 = 1.005$

Άσκηση 4.23 (συνέχεια)

- **4.23.5** Με τη διάταξη πρόβλεψης των 2 bit, ποια επιτάχυνση θα μπορούσε να επιτευχθεί αν μπορούσαμε να μετατρέψουμε τις μισές εντολές διακλάδωσης έτσι ώστε μια εντολή διακλάδωση να αντικαθίσταται από δύο εντολές ALU; Υποθέστε ότι οι σωστά και οι λανθασμένα προβλεφθείσες εντολές έχουν την ίδια πιθανότητα να αντικατασταθούν.
- Κάθε εντολή διακλάδωσης που μετατρέπεται, τώρα διαρκεί έναν πρόσθετο κύκλο εκτέλεσης, άρα έχουμε:

	CPI χωρίς μετατροπή	Κύκλοι ανά αρχική εντολή με τη μετατροπή	Επιτάχυνση
α.	1.060	$1 + (1 + 2 \times (1 - 0.80)) \times 0.15 \times 0.5 = 1.105$	1.060/1.105 = 0.959
β.	1.010	$1 + (1 + 2 \times (1 - 0.95)) \times 0.10 \times 0.5 = 1.055$	1.010/1.055 = 0.957

Άσκηση 4.23 (συνέχεια)

- **4.23.6** Μερικές εντολές διακλάδωσης είναι πολύ πιο προβλέψιμες από άλλες. Αν γνωρίζουμε ότι το 80% όλων των εντολών που εκτελούνται είναι εύκολα προβλέψιμες διακλαδώσεις βρόχου προς τα πίσω (loop-back branches) που πάντα προβλέπονται σωστά, ποια είναι η ακρίβεια της διάταξης πρόβλεψης των 2 bit για το υπόλοιπο 20% των εντολών διακλάδωσης;
- Ας υποθέσουμε ότι ο συνολικός αριθμός εντολών διακλάδωσης που εκτελούνται από το πρόγραμμα είναι B. Τότε έχουμε:

	Σωστά προβλεφθείσες	Σωστά προβλεφθείσες όχι προς τα πίσω	Ακρίβεια των όχι προς τα πίσω διακλαδώσεων
α.	B × 0.80	B × 0.00	(B × 0.00)/(B × 0.20) = 0.00 (00%)
β.	B × 0.95	B × 0.15	(B × 0.15)/(B × 0.20) = 0.75 (75%)

Άσκηση 4.24

- Αυτή η άσκηση εξετάζει την ακρίβεια διαφόρων διατάξεων πρόβλεψης διακλάδωσης για την παρακάτω επαναλαμβανόμενη σειρά (π.χ., σε ένα βρόχο) αποτελεσμάτων διακλάδωσης (Λ = ληφθείσα, ΜΛ = μη ληφθείσα):

	Αποτέλεσμα διακλάδωσης
α.	Λ, Λ, ΜΛ, Λ
β.	Λ, Λ, Λ, ΜΛ, ΜΛ

Άσκηση 4.24 (συνέχεια)

- **4.24.1** Ποια είναι η ακρίβεια των διατάξεων πρόβλεψης «πάντα ληφθείσα» και «πάντα μη ληφθείσα» γι' αυτή την ακολουθία αποτελεσμάτων διακλάδωσης;

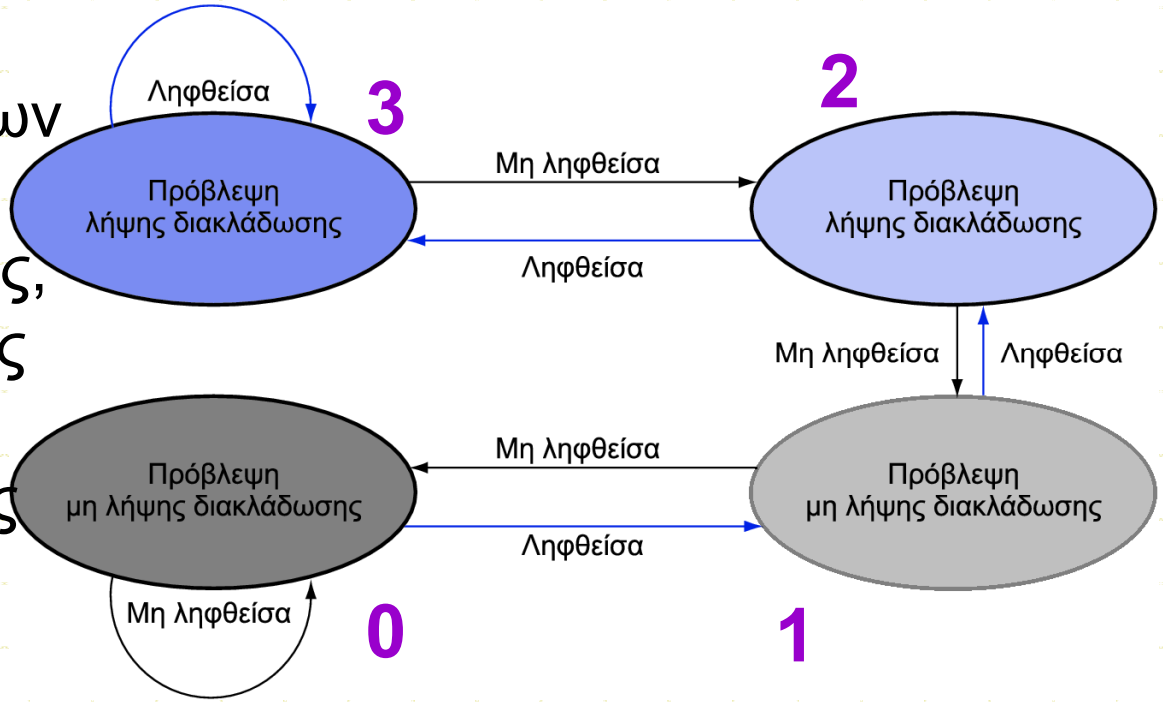
	Αποτέλεσμα διακλάδωσης
α.	Λ, Λ, ΜΛ, Λ
β.	Λ, Λ, Λ, ΜΛ, ΜΛ

- άρα:

	Ακρίβεια Πάντα ληφθείσας	Ακρίβεια Πάντα μη ληφθείσας
α.	3/4 = 75%	1/4 = 25%
β.	3/5 = 60%	2/5 = 40%

Άσκηση 4.24 (συνέχεια)

- **4.24.2** Ποια είναι η ακρίβεια της διάταξης των 2 bit για τις πρώτες 4 διακλαδώσεις της σειράς, αν η διάταξη πρόβλεψης αρχίζει από την κάτω αριστερή κατάσταση της Εικόνας (πρόβλεψη μη λήψης).



	Αποτελέσματα	Τιμή διάταξης πρόβλεψης τη στιγμή της πρόβλεψης	Σωστό ή Λάθος	Ακρίβεια
α.	Λ, Λ, ΜΛ, Λ	0, 1, 2, 1	Λ, Λ, Λ, Λ	0%
β.	Λ, Λ, Λ, ΜΛ	0, 1, 2, 3	Λ, Λ, Σ, Λ	25%

Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.24 (συνέχεια)

- **4.24.3** Ποια είναι η ακρίβεια της διάταξης πρόβλεψης των 2 bit αν η σειρά αυτή επαναλαμβάνεται για πάντα;
- Οι λίγες πρώτες επαναλήψεις της σειράς δεν έχουν την ίδια ακρίβεια με τις επόμενες επειδή ο predictor ακόμη «**προθερμαίνεται**». Για να καθοριστεί η ακρίβεια στην «**σταθερή κατάσταση**», πρέπει να προχωρήσουμε στις προβλέψεις μέχρι να αρχίσουν να επαναλαμβάνονται οι τιμές πρόβλεψης (δηλαδή, ο predictor να έχει την ίδια τιμή στην αρχή της τρέχουσας και της επόμενης επανάληψης της σειράς).

Άσκηση 4.24 (συνέχεια)

	Αποτελέσματα	Τιμή διάταξης πρόβλεψης τη στιγμή της πρόβλεψης	Σωστό ή Λάθος	Ακρίβεια
α.	Λ, Λ, ΜΛ, Λ	1η επαν. 0, 1, 2, 1 2η επαν. 2, 3, 2, 3 3η επαν. 3, 3, 3, 2 4η επαν. 3, 3, 3, 2	Σ, Σ, Λ, Σ	75%
β.	Λ, Λ, Λ, ΜΛ, ΜΛ	1η επαν. 0, 1, 2, 3, 2 2η επαν. 1, 2, 3, 3, 2 3η επαν. 1, 2, 3, 3, 2	Λ, Σ, Σ, Λ, Λ	40%

Άσκηση 4.24 (συνέχεια)

- **4.24.4** Σχεδιάστε μια διάταξη πρόβλεψης που θα επιτύγχανε μια τέλεια ακρίβεια αν αυτή η σειρά επαναλαμβάνεται για πάντα. Η διάταξή σας πρέπει να είναι ένα ακολουθιακό κύκλωμα, με μια έξοδο που παρέχει μια πρόβλεψη (1 για ληφθείσα, 0 για μη ληφθείσα), και χωρίς άλλες εισόδους εκτός του ρολογιού και του σήματος ελέγχου το οποίο δείχνει ότι η εντολή είναι μια διακλάδωση υπό συνθήκη.
- Η διάταξη πρέπει να είναι ένας **καταχωρητής ολίσθησης (shift register) των N-bit**, όπου N είναι ο αριθμός των αποτελεσμάτων των διακλαδώσεων στη σειρά. Ο καταχωρητής πρέπει να αρχικοποιείται με τη σειρά καθαυτή (0 για ΜΛ, 1 για Λ), και η πρόβλεψη είναι πάντα η τιμή στο αριστερότερο bit του καταχωρητή. Ο καταχωρητής πρέπει να ολισθαίνει μετά από κάθε πρόβλεψη διακλάδωσης.

Άσκηση 4.24 (συνέχεια)

- **4.24.5** Ποια είναι η ακρίβεια της διάταξης πρόβλεψης της Άσκησης 4.24.4 αν της δίνεται μια επαναλαμβανόμενη σειρά η οποία είναι το ακριβώς αντίθετο της παραπάνω;
- Αφού η έξοδος της διάταξης πρόβλεψης είναι πάντα το αντίθετο από το πραγματικό αποτέλεσμα της εντολής διακλάδωσης, **η ακρίβεια είναι 0%**.

Άσκηση 4.24 (συνέχεια)

- **4.24.6** Επαναλάβετε την Άσκηση 4.24.4, αλλά τώρα η διάταξη πρόβλεψής σας πρέπει να μπορεί τελικά (μετά από μια περίοδο «προθέρμανσης» στη διάρκεια της οποίας μπορεί να κάνει λάθος προβλέψεις) να αρχίσει να προβλέπει τέλεια και τη συγκεκριμένη σειρά και την αντίθετή της. Η διάταξη πρόβλεψής σας πρέπει να έχει μια είσοδο που να της λέει ποιο ήταν το πραγματικό αποτέλεσμα. Υπόδειξη: αυτή η είσοδος επιτρέπει στη διάταξη πρόβλεψής σας να προσδιορίζει ποια από τις δύο επαναλαμβανόμενες σειρές τής έχει δοθεί.
- Η διάταξη είναι η ίδια με της 4.24.4, εκτός του ότι πρέπει να συγκρίνει την πρόβλεψή της με το πραγματικό αποτέλεσμα και να αντιστρέφει (not) όλα τα bit του καταχωρητή αν η πρόβλεψη είναι λάθος. Αυτή η διάταξη εξακολουθεί να προβλέπει τέλεια τη δεδομένη σειρά. Για την αντίθετη σειρά, η πρώτη πρόβλεψη θα είναι λάθος, κι έτσι οι κατάσταση της διάταξης πρόβλεψης θα αντιστραφεί και από κει και πέρα οι προβλέψεις θα είναι πάντα σωστές. Συνολικά, δεν υπάρχει περίοδος προθέρμανσης για τη δεδομένη σειρά, και για την αντίστροφή σειρά η περίοδος προθέρμανσης είναι μόνο μία διακλάδωση.

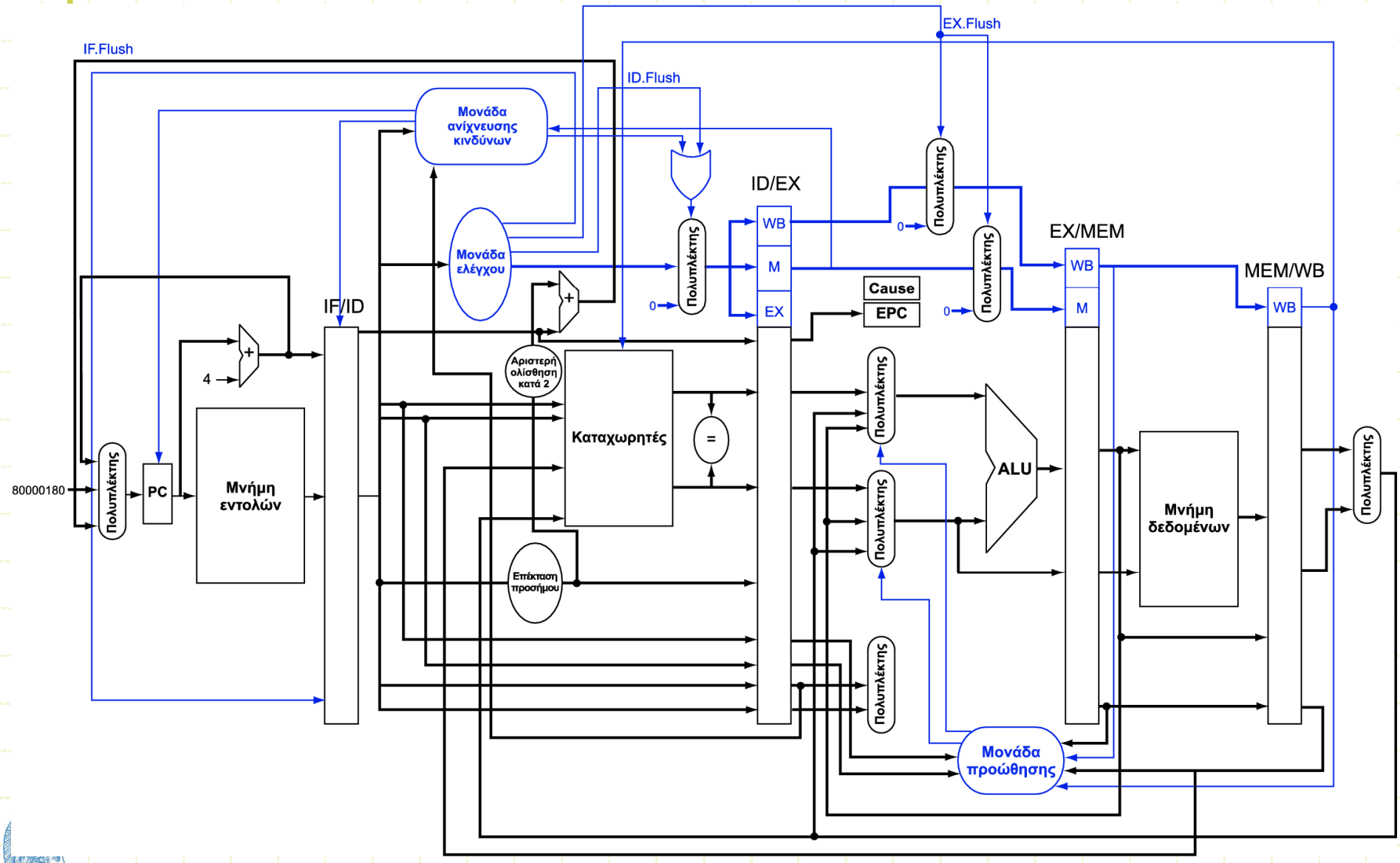
Άσκηση 4.26

- Η άσκηση αυτή διερευνά τον τρόπο με τον οποίο ο χειρισμός εξαιρέσεων επηρεάζει τη σχεδίαση της μονάδας ελέγχου και το χρόνο κύκλου ρολογιού του επεξεργαστή. Τα τρία πρώτα προβλήματα της άσκησης αναφέρονται στην επόμενη εντολή MIPS που προκαλεί μια εξαίρεση:

	Εντολή	Εξαίρεση
α.	`add $0,$1,$2`	Αριθμητική υπερχείλιση
β.	`lw $2,40($3)`	Μη έγκυρη διεύθυνση μνήμης δεδομένων

- **4.26.1** Για κάθε στάδιο της διοχέτευσης, προσδιορίστε τις τιμές των σημάτων ελέγχου της Εικόνας 4.66 που έχουν σχέση με τις εξαιρέσεις, καθώς αυτή η εντολή περνάει μέσα από το συγκεκριμένο στάδιο της διοχέτευσης.

Άσκηση 4.26 (συν.), Εικόνα 4.66



Άσκηση 4.26 (συνέχεια)

- Όλα τα σήματα ελέγχου που σχετίζονται με τις εξαιρέσεις είναι 0 σε όλα τα στάδια, εκτός από αυτό στο οποίο ανιχνεύεται η εξαίρεση. Για το στάδιο αυτό δείχνουμε τις τιμές των σημάτων Flush για διάφορα στάδιο, και επίσης την τιμή του σήματος που ελέγχει τον πολ/κτη που παρέχει την τιμή του PC.

	Στάδιο	Σήματα
α.	EX	IF.Flush = ID.Flush = EX.Flush = 1 PCSel = Exc
β.	MEM	IF.Flush = ID.Flush = EX.Flush = MEM.Flush = 1 PCSel = Exc Αφου η εξαίρεση ανιχνεύεται στο MEM, προσθέσαμε το MEM.Flush

Άσκηση 4.26 (συνέχεια)

- **4.26.2** Κάποια από τα σήματα ελέγχου που δημιουργούνται στο στάδιο ID αποθηκεύονται στον καταχωρητή διοχέτευσης ID/EX, και μερικά πηγαίνουν απευθείας στο στάδιο EX. Εξηγήστε γιατί συμβαίνει αυτό, χρησιμοποιώντας την εντολή αυτή ως παράδειγμα.
- Τα σήματα που αποθηκεύονται στον ID/EX απαιτούνται για να εκτελεστεί η εντολή αν δεν υπάρχουν εξαιρέσεις. Η εικόνα 4.66 δεν το δείχνει, αλλά οι συνθήκες εξαίρεσης των διαφόρων σταδίων παρέχονται επισης στη μονάδα ελέγχου. Το σήμα που πηγαίνει **απευθείας στο EX είναι το EX.Flush και βασίζεται σ' αυτές τις εισόδους συνθήκης εξαίρεσης, και όχι στο opcode της εντολής που είναι στο στάδιο ID**. Συγκεκριμένα, το σήμα EX.Flush γίνεται 1 όταν η εντολή στο στάδιο EX σηματοδοτεί μια εξαίρεση και **πρέπει να της απαγορευθεί να ολοκληρωθεί**.

Άσκηση 4.26 (συνέχεια)

- **4.26.3** Μπορούμε να κάνουμε το στάδιο EX ταχύτερο ελέγχοντας για εξαιρέσεις στο επόμενο στάδιο από αυτό στο οποίο προκύπτει η συνθήκη εξαίρεσης. Χρησιμοποιώντας αυτή την εντολή ως παράδειγμα, περιγράψτε το κύριο μειονέκτημα αυτής της προσέγγισης.
- Το μειονέκτημα είναι ότι **ο χειριστής εξαιρέσεων ξεκινά εκτέλεση έναν κύκλο αργότερα**.
- Επίσης, μια συνθήκη εξαίρεσης που κανονικά ελέγχεται στο MEM δεν μπορεί να καθυστερήσει στο WB, **διότι την ίδια ώρα η εντολή ενημερώνει καταχωρητές και δεν μπορεί να την αποτρέψουμε να το κάνει**.

Άσκηση 4.26 (συνέχεια)

- Τα τρία υπόλοιπα προβλήματα της άσκησης υποθέτουν ότι τα στάδια της διοχέτευσης έχουν τους παρακάτω λανθάνοντες χρόνους:

	IF	ID	EX	MEM	WB
α.	300ps	320ps	350ps	350ps	100ps
β.	200ps	170ps	210ps	210ps	150ps

Άσκηση 4.26 (συνέχεια)

- **4.26.4** Αν η εξαίρεση υπερχείλισης συμβαίνει μία φορά κάθε 100.000 εντολές που εκτελούνται, ποια είναι η συνολική επιτάχυνση αν μεταφέρουμε τον έλεγχο για υπερχείλιση μέσα στο στάδιο MEM; Υποθέστε ότι αυτή η αλλαγή μειώνει το λανθάνοντα χρόνο του EX κατά 30ns, και ότι το IPC που επιτυγχάνει ο επεξεργαστής με διοχέτευση είναι 1 όταν δεν υπάρχουν εξαιρέσεις.
- Όταν ανιχνεύεται υπερχείλιση στο EX, κάθε εξαίρεση έχει σαν αποτέλεσμα 3 κύκλους καθυστέρησης (οι IF, ID, EX ακυρώνονται). Με τη μεταφορά της υπερχείλισης στο MEM, προσθέτουμε ένα ακόμη κύκλο καθυστέρησης. Για να υπολογίσουμε την επιτάχυνση, υπολογίζουμε το χρόνο εκτέλεσης ανά 100.000 εντολές:

	Παλιός χρόνος κύκλου ρολογιού	Νέος χρόνος κύκλου ρολογιού	Παλιός χρόνος ανά 100.000 εντολές	Νέος χρόνος ανά 100.000 εντολές	Επιτάχυνση
α.	350ps	350ps	350ps × 100,003	350ps × 100,004	0,99999
β.	210ps	210ps	210ps × 100,003	210ps × 100,004	0,99999

Άσκηση 4.26 (συνέχεια)

- **4.26.5** Μπορούμε να δημιουργήσουμε σήματα ελέγχου εξαίρεσης στο στάδιο EX αντί για το ID; Εξηγήστε πώς θα δουλέψει ή γιατί δε θα δουλέψει αυτό, χρησιμοποιώντας ως παράδειγμα την εντολή «bne $4,$5,Label» και τους παραπάνω λανθάνοντες χρόνους της διοχέτευσης.
- Τα σήματα ελέγχου (Flush) δε δημιουργούνται στην πραγματικότητα στο στάδιο EX. Δημιουργούνται από τη μονάδα ελέγχου, που σχεδιάζεται σαν μέρος του ID, αλλά θα μπορούσαμε να έχουμε μια ξεχωριστή «Μονάδα εξαιρέσεων» για τη δημιουργία των σημάτων Flush και στην πραγματικότητα αυτή η μονάδα δεν είναι μέρος κανενός σταδίου.

Άσκηση 4.26 (συνέχεια)

- **4.26.6** Με την παραδοχή ότι κάθε πολυπλέκτης έχει λανθάνοντα χρόνο 40ps, προσδιορίστε πόσο χρόνο έχει η μονάδα ελέγχου για να παραγάγει τα σήματα εκκένωσης (flush); Ποιο σήμα είναι το πιο κρίσιμο;
- Τα σήματα Flush πρέπει να δημιουργηθούν ένα χρόνο πολυπλέκτη πριν το τέλος του κύκλου. Ωστόσο, η δημιουργία τους μπορεί να ξεκινήσει μόνο μετά τη δημιουργία των συνθηκών εξαίρεσης. Για παράδειγμα, η αριθμητική υπερχείλιση δημιουργείται μόνο μετά τη λειτουργία ALU στο στάδιο EX, συνήθως στο τελευταίο μέρος του κύκλου ρολογιού. Συνεπώς, η μονάδα ελέγχου έχει πολύ λίγο χρόνο για να δημιουργήσει αυτά τα σήματα, και μπορούν εύκολα να βρισκονται στην κρίσιμη διαδρομή που καθορίζει τον κύκλο ρολογιού.

Άσκηση 4.28

- Στην άσκηση αυτή συγκρίνουμε την απόδοση επεξεργαστών απλής και διπλής εκκίνησης εντολής (1-issue και 2-issue processors), λαμβάνοντας υπόψη μετασχηματισμούς προγραμμάτων που μπορούν να γίνουν για τη βελτιστοποίηση της εκτέλεσης σε επεξεργαστή διπλής εκκίνησης. Τα προβλήματα αυτής της άσκησης αναφέρονται στον επόμενο βρόχο (γραμμένο σε C):

	Κώδικας C
α.	`for(i=0;i!=j;i++)` `  b[i]=a[i];`
β.	`for(i=0;a[i]!=a[i+1];i++)` `  a[i]=0;`

Άσκηση 4.28 (συνέχεια)

- Όταν θα γράφετε κώδικα MIPS, υποθέστε ότι οι μεταβλητές διατηρούνται σε καταχωρητές όπως φαίνεται παρακάτω, και ότι όλοι οι καταχωρητές εκτός από αυτούς που σημειώνονται ως «Ελεύθεροι» χρησιμοποιούνται για να διατηρήσουν διάφορες μεταβλητές, οπότε δεν μπορούν να χρησιμοποιηθούν για οτιδήποτε άλλο.

	i	j	a	b	c	Ελεύθεροι
α.	$1	$2	$3	$4	$5	$6,$7,$8
β.	$4	$5	$6	$7	$8	$1,$2,$3

- **4.28.1** Μεταφράστε αυτόν τον κώδικα της C σε εντολές MIPS. Η μετάφρασή σας πρέπει να είναι άμεση, χωρίς αναδιάταξη εντολών για καλύτερη απόδοση.

Άσκηση 4.28 (συνέχεια)

	Κώδικας C	Κώδικας MIPS

α.	`for(i=0;i!=j;i++)`
	`  b[i]=a[i];`
		`  add $1,$0,$0`
		`Again: beq $1,$2,End`
		`  add $6,$3,$1`
		`  lw $7,0($6)`
		`  add $8,$4,$1`
		`  sw $7,0($8)`
		`  addi $1,$1,1`
		`  beq $0,$0,Again`
		`End:`

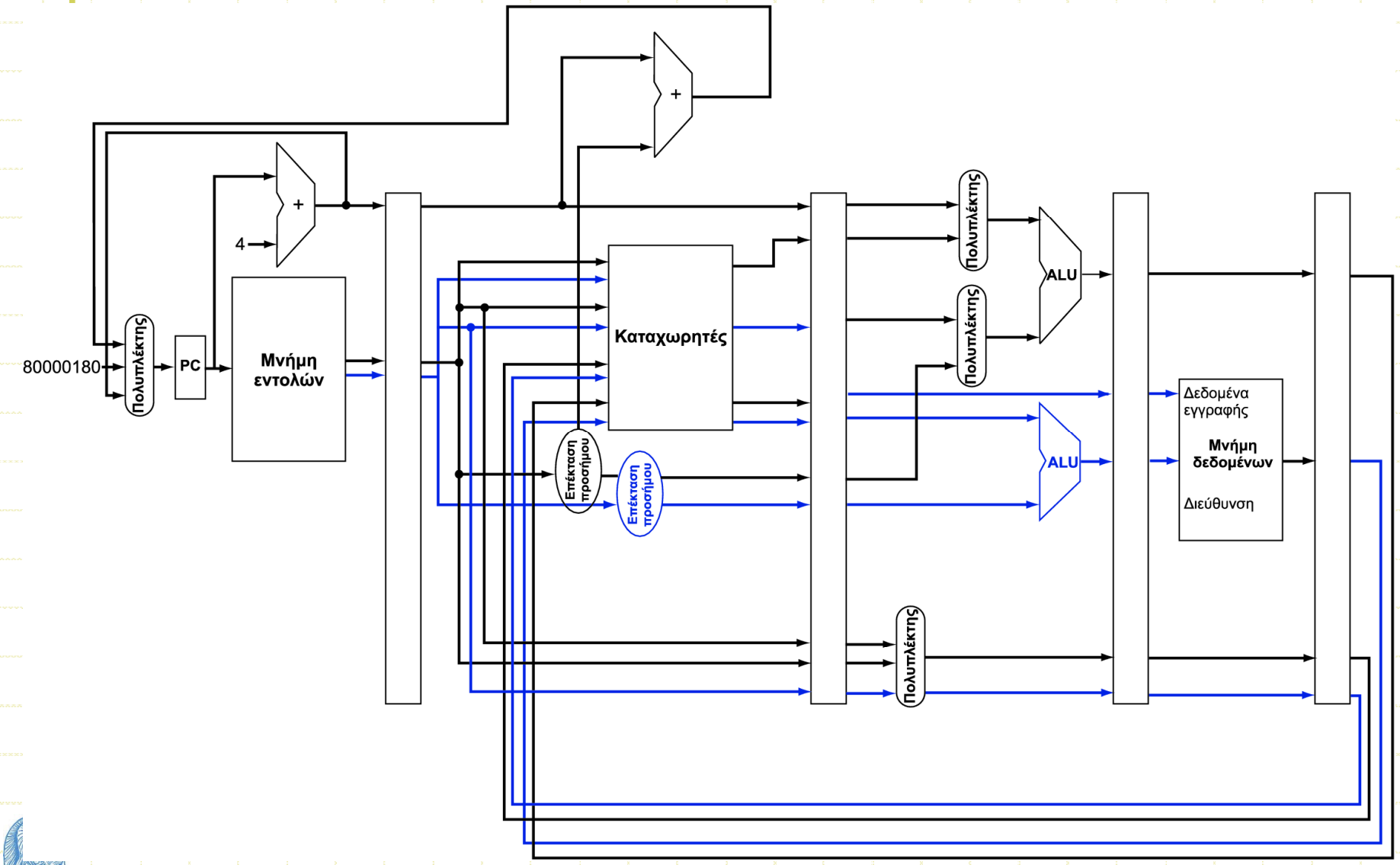
β.	`for(i=0;a[i]!=a[i+1];i++)`
	`  a[i]=0;`
		`  add $4,$0,$0`
		`Again: add $1,$4,$6`
		`  lw $2,0($1)`
		`  lw $3,1($1)`
		`  beq $2,$3,End`
		`  sw $0,0($1)`
		`  addi $4,$4,1`
		`  beq $0,$0,Again`
		`End:`

Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.28 (συνέχεια)

- **4.28.2** Αν ο βρόχος τερματίζεται μετά από δύο μόνον επαναλήψεις, σχεδιάστε ένα διάγραμμα διοχέτευσης για τον κώδικα MIPS που γράψατε στην Άσκηση 4.28.1, που να εκτελείται στον επεξεργαστή διπλής εκκίνησης ο οποίος φαίνεται στην Εικόνα 4.69. Υποθέστε ότι ο επεξεργαστής διαθέτει τέλεια πρόβλεψη διακλαδώσεων και μπορεί να προσκομίσει οποιεσδήποτε 2 εντολές (όχι μόνο διαδοχικές) στην ίδιο κύκλο.

Άσκηση 4.28 (συν), Εικόνα 4.69



Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών

Άσκηση 4.28 (συνέχεια)

	Εντολές	Διοχέτευση

α.	add $1,$0,$0
	beq $1,$2,End
	add $6,$3,$1
	lw $7,0($6)
	add $8,$4,$1
	sw $7,0($8)
	addi $1,$1,1
	beq $0,$0,Again
	beq $1,$2,End
	add $6,$3,$1
	lw $7,0($6)
	add $8,$4,$1
	sw $7,0($8)
	addi $1,$1,1
	beq $0,$0,Again
	beq $1,$2,End
		IF ID EX ME WB
		IF ID ** EX ME WB
		IF ** ID EX ME WB
		IF ** ID ** EX ME WB
		IF ** ID EX ME WB
		IF ** ID ** EX ME WB
		IF ** ID EX ME WB
		IF ** ID ** EX ME WB
		IF ** ID EX ME WB
		IF ** ID EX ME WB
		IF ID EX ME WB
		IF ID EX ME WB
		IF ID EX ME WB
		IF ID ** EX ME WB

- 18 κύκλοι, 16 εντολές
- CPI = 18/16 = 1,125

Άσκηση 4.28 (συνέχεια)

	Εντολές	Διοχέτευση
β.	add $4,$0,$0	IF ID EX ME WB
	add $1,$4,$6	IF ID ** EX ME WB
	lw $2,0($1)	IF ** ID EX ME WB
	lw $3,1($1)	IF ** ID ** EX ME WB
	beq $2,$3,End	IF ** ID ** EX ME WB
	sw $0,0($1)	IF ** ID ** EX ME WB
	addi $4,$4,1	IF ** ID EX ME WB
	beq $0,$0,Again	IF ** ID ** EX ME WB
	add $1,$4,$6	IF ** ID EX ME WB
	lw $2,0($1)	IF ** ID ** EX ME WB
	lw $3,1($1)	IF ** ID ** EX ME WB
	beq $2,$3,End	IF ** ID ** ** EX ME WB
	sw $0,0($1)	IF ** ** ID EX ME WB
	addi $4,$4,1	IF ** ** ID EX ME WB
	bew $0,$0,Again	IF ID EX ME WB
	add $1,$4,$6	IF ID ** EX ME WB
	lw $2,0($1)	IF ** ID EX ME WB
	lw $3,1($1)	IF ** ID ** EX ME WB
	beq $2,$3,End	IF ** ID ** EX ME WB

- 24 κύκλοι, 19 εντολές
- CPI = 20/19 = 1,263

Άσκηση 4.28 (συνέχεια)

- **4.28.3** Αναδιατάξτε τον κώδικα που γράψατε στην Άσκηση 4.28.1 για να επιτύχετε καλύτερη απόδοση στο στατικά χρονοπρογραμματιζόμενο επεξεργαστή διπλής εκκίνησης της Εικόνας 4.69.
- Ο μόνος τρόπος για να εκτελεστούν 2 εντολές πλήρως παράλληλα είναι να εκτελείται μια load/store μαζί με μια άλλη εντολή. Για να γίνει αυτό, θα προσπαθήσουμε να βάλουμε γύρω από μία load/store μια άλλη εντολή που δεν είναι load/store και δεν έχει εξάρτηση από τη load/store.

Άσκηση 4.28 (συνέχεια)

	Κώδικας MIPS	Σχόλιο

α.	`add $1,$0,$0`
	`Again: beq $1,$2,End`
	`add $6,$3,$1`
	`add $8,$4,$1`
	`lw $7,0($6)`
	`addi $1,$1,1`
	`sw $7,0($8)`
	`beq $0,$0,Again`
	`End:`	

β.	`add $4,$0,$0`
	`Again: add $1,$4,$6`
	`lw $2,0($1)`
	`lw $3,1($1)`
	`beq $2,$3,End`
	`sw $0,0($1)`
	`addi $4,$4,1`
	`beq $0,$0,Again`
	`End:`
		Δεν αλλάξαμε τίποτε. Σημειώστε ότι η μόνη εντολή χωρίς εξαρτήσεις προς ή από τις δύο load είναι η addi, και αυτή δεν μπορεί να μεταφερθεί πάνω από τη beq (τότε ο βρόχος θα τερματιζόταν με τη λανθασμένη τιμή του i).

Άσκηση 4.28 (συνέχεια)

- **4.28.4** Επαναλάβετε την Άσκηση 4.28.2, αλλά αυτή τη φορά χρησιμοποιήστε τον κώδικα MIPS που γράψατε στην Άσκηση 4.28.3.
- Όμοια με τη 4.28.2…
- Στο α. χρειάζονται 16 κύκλοι ρολογιού (αντί 18) και γίνεται το CPI=1
- Στο β. χρειάζονται 24 κύκλοι ρολογιού (όπως και πριν 24)

Άσκηση 4.28 (συνέχεια)

- **4.28.5** Ποια είναι η επιτάχυνση που επιτυγχάνει η μετάβαση από έναν επεξεργαστή απλής εκκίνησης σε έναν επεξεργαστή διπλής εκκίνησης όπως στην Εικόνα 4.69; Χρησιμοποιήστε τον κώδικα που γράψατε στην Άσκηση 4.28.1 και για την απλή εκκίνηση και για τη διπλή εκκίνηση, και υποθέστε ότι εκτελούνται 1.000.000 επαναλήψεις του βρόχου. Όπως στην Άσκηση 4.28.2, υποθέστε ότι ο επεξεργαστής έχει τέλειες προβλέψεις διακλαδώσεων, και ότι ένας επεξεργαστής διπλής εκκίνησης μπορεί να προσκομίσει οποιεσδήποτε 2 εντολές στον ίδιο κύκλο.

Άσκηση 4.28 (συνέχεια)

	CPI για εκκίνηση 1 εντολής	CPI για εκκίνηση 2 εντολών	Επιτά-χυνση
α.	1 (χωρίς κινδύνους δεδομένων)	0.86 (12 κ. για 14 εντολές). Στις άρτιες επαναλήψεις η lw και η sw μπορούν να εκτελεστούν παράλληλα με την επόμενη εντολή.	1,16
β.	1.14 (8 κ. για 7 εντολές). Υπάρχει 1 κύκλος καθυστέρησης σε κάθε επανάληψη εξαιτίας κινδύνου δεδομένων μεταξύ της lw και την επόμενης εντολής beq.	1 (14 κ. για 14 εντολές). Ούτε η lw μπορεί να εκτελεστεί παράλληλα με άλλη εντολή, και η beq μετά τη δεύτερη lw καθυστερεί διότι εξαρτάται από τη lw. Ωστόσο, η sw πάντα εκτελείται παράλληλα με μια άλλη εντολή (εναλλάσεται μεταξύ beq και addi).	1,14

Άσκηση 4.28 (συνέχεια)

- **4.28.6** Επαναλάβετε την Άσκηση 4.28.5, αλλά αυτή τη φορά υποθέστε ότι στον επεξεργαστή διπλής εκκίνησης μία από τις εντολές που θα εκτελεστεί σε έναν κύκλο μπορεί να είναι οποιουδήποτε τύπου, και η άλλη πρέπει να είναι μια εντολή που να μην έχει σχέση με τη μνήμη.

Άσκηση 4.28 (συνέχεια)

	CPI για εκκίνηση 1 εντολής	CPI για εκκίνηση 2 εντολών	Επιτά-χυνση
α.	1	0.64 (9 κ. για 14 εντολές). Στις περιττές επαναλήψεις οι add και lw δεν μπορούν να εκτελεστούν στον ίδιο κύκλο εξαιτίας εξάρτησης δεδομένων, και ακολούθως οι add και sw έχουν το ίδιο πρόβλημα. Οι υπόλοιπες εντολές μπορούν να εκτελεστούν σε ζεύγη.	1,56
β.	1.14	0.86 (12 κ. για 14 εντολές). Σε όλες τις επαναλήψεις η beq καθυστερεί διότι εξαρτάται από τη δεύτερη lw. Στις περιττές επαναλήψεις οι beq και sw εκτελούνται μαζί, και το ίδιο οι addi και η τελευταία beq. Στις άρτιες επαναλήψεις οι sw και addi εκτελούνται μαζί, και το ίδιο και η τελευταία beq και η πρώτη add της επόμενης επανάληψης.	1,33

Άσκηση 4.30

- Στην άσκηση αυτή, κάνουμε διάφορες παραδοχές. Πρώτον, υποθέτουμε ότι ένας υπερβαθμωτός επεξεργαστής Ν-πλής εκκίνησης μπορεί να εκτελεί οποιεσδήποτε Ν εντολές στον ίδιο κύκλο, ανεξάρτητα από τους τύπους τους. Δεύτερον, υποθέτουμε ότι κάθε εντολή επιλέγεται ανεξάρτητα, χωρίς να λαμβάνεται υπόψη αυτή που προηγείται ή ακολουθεί. Τρίτον, υποθέτουμε ότι δεν υπάρχουν καθυστερήσεις λόγω εξαρτήσεων δεδομένων, ότι δε χρησιμοποιούνται υποδοχές καθυστέρησης, και ότι οι διακλαδώσεις εκτελούνται στο στάδιο ΕΧ της διοχέτευσης. Τέλος, υποθέτουμε ότι οι εκτελούμενες εντολές έχουν την εξής κατανομή:

	ALU	Σωστά προβλεφθείσα beq	Λανθασμένα προβλεφθείσα beq	lw	sw
α.	50%	18%	2%	20%	10%
β.	40%	10%	5%	35%	10%

Άσκηση 4.30 (συνέχεια)

- **4.30.1** Ποιο είναι το CPI που επιτυγχάνει ένας στατικός υπερβαθμωτός επεξεργαστής διπλής εκκίνησης γι' αυτό το πρόγραμμα;
- Αν p είναι η πιθανότητα λάθους πρόβλεψης διακλάδωσης, τότε όποτε έχουμε λανθασμένη πρόβλεψη για μια beq που είναι πρώτη από τις δύο εντολές σε έναν κύκλο (πιθανότητα p), χάνουμε μια υποδοχή εκκίνησης (issue slot), δηλαδή μισό κύκλο και άλλους δύο ολόκληρους κύκλους που ακολουθούν (αφού η επίλυση γίνεται στο EX). Εάν η πρώτη εντολή ενός κύκλου δεν είναι λανθασμένα προβλεφθείσα beq αλλά είναι η δεύτερη (πιθανότητα (1 – p) x p), χάνουμε τους δύο επόμενους κύκλους. Χωρίς αυτές τις λάθος προβλέψεις, θα μπορούσαμε να εκτελούμε πάντα 2 εντολές σε κάθε κύκλο. Έτσι, έχουμε:

	CPI
α.	$0.5 + 0.02 \times 2.5 + 0.98 \times 0.02 \times 2 = \mathbf{0.589}$
β.	$0.5 + 0.05 \times 2.5 + 0.95 \times 0.05 \times 2 = \mathbf{0.720}$

Άσκηση 4.30 (συνέχεια)

- **4.30.2** Σε ένα στατικό υπερβαθμωτό επεξεργαστή διπλής εκκίνησης, του οποίου η διάταξη πρόβλεψης μπορεί να χειριστεί μόνο μία διακλάδωση ανά κύκλο, ποια επιτάχυνση επιτυγχάνεται με την προσθήκη της δυνατότητας πρόβλεψης δύο διακλαδώσεων ανά κύκλο; Υποθέστε πολιτική καθυστέρησης σε περίπτωση διακλάδωσης (stall-on-branch) για τις διακλαδώσεις τις οποίες δεν μπορεί να χειριστεί η διάταξη πρόβλεψης.
- Αδυναμία να προβλέψουμε το αποτέλεσμα μιας διακλάδωσης έχει την ίδια τιμή με ένα μια διακλάδωση που προβλέπεται λανθασμένα. Υπολογίζουμε το CPI όπως και στην 4.30.1, αλλά αυτή τη φορά έχουμε και μια ποινή 2 κύκλων αν υπάρχει σωστά προβλεφθείσα διακλάδωση στην πρώτη υποδοχή εκκίνησης και μια άλλη διακλάδωση που θα μπορούσε να προβλεφθεί σωστά (αλλά τώρα δεν μπορεί) στη δεύτερη υποδοχή. Έχουμε:

	CPI με 2 προβλεφθείσες διακλαδώσεις ανά κύκλο	CPI με 1 προβλεφθείσα διακλάδωση ανά κύκλο	Επιτάχυνση
α.	0.589	0.5 + 0.02 × 2.5 + 0.98 × 0.02 × 2 + 0.18 × 0.18 × 2 = 0.654	1.11
β.	0.720	0.5 + 0.05 × 2.5 + 0.95 × 0.05 × 2 + 0.10 × 0.10 × 2 = 0.740	1.03

Άσκηση 4.30 (συνέχεια)

- **4.30.3** Σε ένα στατικό υπερβαθμωτό επεξεργαστή διπλής εκκίνησης που έχει μόνο μία θύρα εγγραφής καταχωρητή, ποια επιτάχυνση επιτυγχάνεται με την προσθήκη μιας δεύτερης θύρας εγγραφής καταχωρητή;
- Έχουμε ποινή 1 κύκλου όποτε έχουμε κύκλο με δύο εντολές που χρειάζονται και οι δύο εγγραφή σε καταχωρητή. Τέτοιες εντολές είναι οι εντολές ALU και lw. Σημειώστε ότι η beq δε γράφει σε καταχωρητές, και γι' αυτό οι καθυστερήσεις που οφείλονται σε εγγραφές καταχωρητών κι αυτές που οφείλονται σε λάθος προβλέψεις διακλαδώσεων είναι ανεξάρτητα συμβάντα. Έχουμε:

	CPI με 2 εγγραφές καταχωρητή ανά κύκλο	CPI με 1 εγγραφή καταχωρητή ανά κύκλο	Επιτάχυνση
α.	0.589	0.5 + 0.02 × 2.5 + 0.98 × 0.02 × 2 + 0.70 × 0.70 × 1 = 1.079	1.83
β.	0.720	0.5 + 0.05 × 2.5 + 0.95 × 0.05 × 2 + 0.75 × 0.75 × 1 = 1.283	1.78

Άσκηση 4.30 (συνέχεια)

- **4.30.4** Για ένα στατικό υπερβαθμωτό επεξεργαστή διπλής εκκίνησης με κλασική διοχέτευση πέντε σταδίων, ποια επιτάχυνση επιτυγχάνουμε κάνοντας τέλεια την πρόβλεψη διακλαδώσεων;
- Έχουμε ήδη υπολογίσει το CPI με τη δεδομένη ακρίβεια πρόβλεψης διακλάδωσης, και γνωρίζουμε φυσικά ότι το CPI για την ιδανική πρόβλεψη είναι 0,5. Άρα:

	CPI με τη δεδομένη ακρίβεια	CPI με τέλεια πρόβλεψη διακλάδωσης	Επιτάχυνση
α.	0.589	0.5	1.18
β.	0.720	0.5	1.44

Άσκηση 4.30 (συνέχεια)

- **4.30.5** Επαναλάβετε την Άσκηση 4.30.4, αλλά για έναν επεξεργαστή **τετραπλής** εκκίνησης. Σε τι συμπέρασμα μπορείτε να καταλήξετε σχετικά με τη σημασία της καλής πρόβλεψης διακλαδώσεων όταν αυξάνεται το εύρος εκκίνησης του επεξεργαστή;
- Το CPI με τέλεια πρόβλεψη είναι τώρα 0.25 (4 εντολές/κύκλο). Μια λάθος πρόβλεψη στην πρώτη υποδοχή του κύκλου έχει ποινή 2.75 κύκλους (οι 3 επόμενες υποδοχές και 2 ολόκληροι κύκλοι), στη δεύτερη υποδοχή έχει ποινή 2.5 κύκλους, στην τρίτη υποδοχή, 2.25 κύκλους και στην τέταρτη υποδοχή 2. Έχουμε:

	CPI με τη δεδομένη πρόβλεψη	CPI με τέλεια πρόβλεψη διακλάδωσης	Επιτάχυνση
α.	$0.25 + 0.02 \times 2.75 + 0.98 \times 0.02 \times 2.5 + 0.98^2 \times 0.02 \times 2.25 + 0.98^3 \times 0.02 \times 2 = 0.435$	0.25	1.18
β.	$0.25 + 0.05 \times 2.75 + 0.95 \times 0.05 \times 2.5 + 0.95^2 \times 0.05 \times 2.25 + 0.95^3 \times 0.05 \times 2 = 0.694$	0.25	1.44

- Η επιτάχυνση από τη βελτιωμένη πρόβλεψη είναι πολύ μεγαλύτερη στον τετραπλής εκκίνησης σε σχέση με τον διπλής. Γενικά, οι επεξεργαστές με εκκίνηση περισσότερων εντολών ανά κύκλο κερδίζουν περισσότερα από τη βελτιωμένη πρόβλεψη επειδή κάθε λάθος πρόβλεψη κοστίζει περισσότερες χαμένες ευκαιρίες εντολής (π.χ. 4 ανά κύκλο έναντι 2 ανά κύκλο).

Άσκηση 4.30 (συνέχεια)

- **4.30.6** Επαναλάβετε την Άσκηση 4.30.5, αλλά τώρα υποθέστε ότι ο επεξεργαστής τετραπλής εκκίνησης έχει 50 στάδια διοχέτευσης. Υποθέστε ότι καθένα από τα αρχικά πέντε στάδια διαιρείται σε δέκα νέα στάδια, και οι διακλαδώσεις εκτελούνται στο πρώτο από τα δέκα νέα στάδια ΕΧ. Σε τι συμπέρασμα μπορείτε να καταλήξετε σχετικά με τη σημασία της καλής πρόβλεψης διακλαδώσεων όταν αυξάνεται το βάθος της διοχέτευσης του επεξεργαστή;

Άσκηση 4.30 (συνέχεια)

- Με αυτή τη διοχέτευση, η ποινή λάθους πρόβλεψης είναι 20 κύκλοι συν το κλάσμα του κύκλου λόγω απόρριψης εντολών που ακολουθούν τη διακλάδωση στον ίδιο κύκλο. Έχουμε

	CPI με τη δεδομένη πρόβλεψη	CPI με τέλεια πρόβλεψη διακλάδωσης	Επιτάχυνση
α.	$0.25 + 0.02 \times 20.75 + 0.98 \times 0.02 \times 20.5 + 0.98^2 \times 0.02 \times 20.25 + 0.98^3 \times 0.02 \times 20 = 1.832$	0.25	7.33
β.	$0.25 + 0.05 \times 20.75 + 0.95 \times 0.05 \times 20.5 + 0.95^2 \times 0.05 \times 20.25 + 0.95^3 \times 0.05 \times 20 = 4.032$	0.25	16.13

- Παρατηρούμε τεράστιες επιταχύνσεις όταν η πρόβλεψη βελτιώνεται σε επεξεργαστή με βαθιά διοχέτευσ. Γενικά οι επεξεργαστές με βαθύτερες διοχετεύσεις κερδίζουν περισσότερο από την βελτιωμένη πρόβλεψη γιατί αυτοί οι επεξεργαστές ακυρώνουν περισσότερες εντολές (π.χ. 20 στάδια έναντι μόνο 2) σε κάθε λάθος πρόβλεψη.

Άσκηση 4.32

- Τα προβλήματα αυτής της άσκησης υποθέτουν ότι οι διακλαδώσεις αποτελούν το παρακάτω κλάσμα όλων των εντολών που εκτελούνται, και έχουν την παρακάτω ακρίβεια πρόβλεψης διακλάδωσης. Υποθέστε ότι ο επεξεργαστής δεν καθυστερεί ποτέ λόγω εξαρτήσεων δεδομένων και πόρων, δηλαδή πάντα προσκομίζει και εκτελεί το μέγιστο αριθμό εντολών ανά κύκλο αν δεν υπάρχουν κίνδυνοι ελέγχου. Για τις εξαρτήσεις ελέγχου, ο επεξεργαστής χρησιμοποιεί πρόβλεψη διακλάδωσης και συνεχίζει την προσκόμιση από την προβλεφθείσα διαδρομή. Αν η διακλάδωση έχει προβλεφθεί λάθος, όταν υπολογίζεται το αποτέλεσμα της διακλάδωσης οι εντολές που προσκομίζονται μετά από τη λάθος προβλεφθείσα διακλάδωση απορρίπτονται, και στον επόμενο κύκλο ο επεξεργαστής ξεκινάει την προσκόμιση από τη σωστή διαδρομή.

	Διακλαδώσεις ως % ποσοστό όλων των εκτελούμενων εντολών	Ακρίβεια της πρόβλεψης διακλάδωσης
α.	20%	90%
β.	20%	99,5%

Άσκηση 4.32 (συνέχεια)

- **4.32.1** Πόσες εντολές αναμένεται να εκτελεστούν μεταξύ της στιγμής που ανιχνεύεται μια λανθασμένη πρόβλεψη διακλάδωσης και της στιγμής που ανιχνεύεται η επόμενη λανθασμένη πρόβλεψη διακλάδωσης;
- Ο αναμενόμενος αριθμός λάθος προβλέψεων ανά εντολή είναι ίσος με την πιθανότητα μια δεδομένη εντολή να είναι διακλάδωση που προβλέπεται λάθος. Ο αριθμός εντολών μεταξύ λανθασμένων προβλέψεων είναι ίσος με 1 διά τον αριθμό των λάθος προβλέψεων ανά εντολή. Έχουμε:

	Λάθος προβλέψεις ανά εντολή	Εντολές μεταξύ λανθασμένων προβλέψεων
α.	0,2 x (1 – 0,9)	50
β.	0,20 x (1 – 0,995)	1000

Άσκηση 4.32 (συνέχεια)

- Τα υπόλοιπα προβλήματα αυτής της άσκησης υποθέτουν το παρακάτω βάθος διοχέτευσης και ότι το αποτέλεσμα της διακλάδωσης προσδιορίζεται στο παρακάτω στάδιο διοχέτευσης (μετρώντας από το στάδιο 1):

	Βάθος διακλάδωσης	Αποτέλεσμα διακλάδωσης γνωστό στο στάδιο
α.	12	10
β.	25	18

Άσκηση 4.32 (συνέχεια)

- **4.32.2** Σε έναν επεξεργαστή τετραπλής εκκίνησης με αυτές τις παραμέτρους διοχέτευσης, πόσες εντολές διακλάδωσης μπορούμε να αναμένουμε ότι βρίσκονται «σε εξέλιξη» (ήδη έχουν προσκομιστεί αλλά δεν έχουν ακόμα υποβληθεί — commit) σε οποιαδήποτε δεδομένη στιγμή;
- Ο αριθμός των εντολών «σε εξέλιξη» είναι ίσος με το βάθος της διοχέτευσης επί το πλάτος εκκίνησης. Ο αριθμός των διακλαδώσεων σε «εξέλιξη» μπορεί να υπολογιστεί εύκολα διότι ξέρουμε ποιο ποσοστό όλων των εντολών είναι διακλαδώσεις. Έχουμε:

	Διακλαδώσεις σε εξέλιξη
α.	12 x 4 x 0,20 = 9,6
β.	25 x 4 x 0,20 = 20

Άσκηση 4.32 (συνέχεια)

- **4.32.3** Πόσες εντολές προσκομίζονται από τη λανθασμένη διαδρομή για κάθε λανθασμένη πρόβλεψη διακλάδωσης σε έναν επεξεργαστή τετραπλής εκκίνησης;
- Συνεχίζουμε την προσκόμιση από τη λάθος διαδρομή μέχρι να γίνει γνωστό το αποτέλεσμα της διακλάδωσης, προσκομίζοντας 4 εντολές σε κάθε κύκλο. Αν το αποτέλεσμα της διακλάδωσης είναι γνωρτό στο στάδιο N της διοχέτευσης, όλες οι εντολές σε N – 1 στάδια είναι από τη λάθος διαδρομή. Στο N-οστό στάδιο, όλες οι εντολές μετά τη διακλάδωση είναι από τη λάθος διαδρομή. Με την υπόθεση ότι η διακλάδωση είναι το ίδιο πιθανό να είναι η $1^η$, $2^η$, $3^η$ ή $4^η$ εντολή που προσκομίζεται στην κύκλο της, έχουμε κατά μέσο όρο 1.5 εντολές από τη λάθος διαδρομή στο στάδιο N (3 αν η διακ/ση είναι η $1^η$, 2 αν είναι η $2^η$, 1 αν είναι η $3^η$, 0 αν είναι η $4^η$). Έτσι:

	Εντολές από τη λάθος διαδρομή
α.	(10 − 1) × 4 × 1.5 = 37.5
β.	(18 − 1) × 4 × 1.5 = 69.5

Άσκηση 4.32 (συνέχεια)

- **4.32.4** Ποια είναι η επιτάχυνση που επιτυγχάνεται με την αλλαγή του επεξεργαστή από τετραπλής εκκίνησης σε οκταπλής εκκίνησης; Υποθέστε ότι οι επεξεργαστές οκταπλής και τετραπλής εκκίνησης διαφέρουν μόνο στον αριθμό εντολών ανά κύκλο, και είναι κατά τα άλλα πανομοιότυποι (βάθος διοχέτευσης, στάδιο υπολογισμού διακλάδωσης, κλπ.).
- Μπορούμε να υπολογίσουμε το CPI κάθε επεξεργαστή, μετά να υπολογίσουμε την επιτάχυνση. Για το CPI, σημειώνουμε ότι έχουμε καθορίσει τον αριθμό των χρήσιμων εντολών μεταξύ λάθος προβλέψεων διακ/σης (στο 4.32.1) και τον αριθμό των εντολών που κακώς προσκομίσθηκαν ανά λανθασμένη πρόβλεψη (στο 4.32.3), και ξέρουμε πόσες εντολές συνολικά προσκομίζονται ανά κύκλο (4 ή 8). Από αυτά μπορούμε να προσδιορίσουμε τον αριθμό κύκλων μεταξύ λανθασμένων προβλέψεων, και έπειτα το CPI (κύκλοι ανά χρήσιμη εντολή). Έχουμε:

	4πλης εκκίνησης		8πλής εκκίνησης			Επιτάχυνση
	Κύκλοι	CPI	Λαθος προβλεφθείσες	Κύκλοι	CPI	
α.	(37.5 + 50)/4 = 21.9	21.9/50 = 0.438	(10 – 1) × 8 × 3.5 = 75.5	(75.5 + 50)/8 = 15.7	15.7/50 = 0.314	1,39
β.	(69.5 + 1000)/4 = 267.4	267.4/1000 = 0.267	(18 – 1) × 8 × 3.5 = 139.5	(139.5 + 1000)/8 = 142.4	142.4/1000 = 0.142	1,88

Άσκηση 4.32 (συνέχεια)

- **4.32.5** Ποια είναι η επιτάχυνση από την εκτέλεση των διακλαδώσεων ένα στάδιο νωρίτερα σε έναν επεξεργαστή τετραπλής εκκίνησης;
- Όταν οι διακλαδώσεις εκτελούνται έναν κύκλο νωρίτερα, χρειάζεται ένας λιγότερος κύκλος για να εκτελεστούν εντολές μεταξύ δύο λάθος προβλέψεων διακλάδωσης. Έχουμε:

	Κανονικό CPI	Βελτιωμένο CPI	Επιτάχυνση
α.	21.9/50 = 0.438	20.9/50 = 0.418	1.048
β.	267.4/1000 = 0.267	266.4/1000 = 0.266	1.004

Άσκηση 4.32 (συνέχεια)

- **4.32.6** Ποια είναι η επιτάχυνση από την εκτέλεση των διακλαδώσεων ένα στάδιο νωρίτερα σε έναν επεξεργαστή οκταπλής εκκίνηση; Συζητήστε τη διαφορά μεταξύ αυτού του αποτελέσματος και του αποτελέσματος της Άσκησης 4.32.5.

	Κανονικό CPI	Βελτιωμένο CPI	Επιτάχυνση
α.	15.7/50 = 0.314	14.7/50 = 0.294	1.068
β.	142.4/1000 = 0.142	141.4/1000 = 0.141	1.007

- Οι επιταχύνσεις από αυτή τη βελτίωση είναι μεγαλύτερες για τον επεξεργαστή 8πλης εκκίνησης έναντι αυτού με την 4απλή. Αυτό ισχύει διότι ο επεξεργαστής 8πλης χρειάζεται λιγότερους κύκλους για να εκτελέσει τον ίδιο αριθμό εντολών, και έτσι η ίδια βελτίωση του 1 κύκλου αντιπροσωπεύει μεγαλύτερη σχετική βελτίωση (επιτάχυνση).