Οργάνωση Υπολογιστών

Επιμέλεια:

Γεώργιος Θεοδωρίδης, Επίκουρος Καθηγητής

Ανδρέας Εμερετλής, Υποψήφιος Διδάκτορας

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Άδειες Χρήσης

Ανάπτυξη

Το παρόν εκπαιδευτικό υλικό αναπτύχθηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

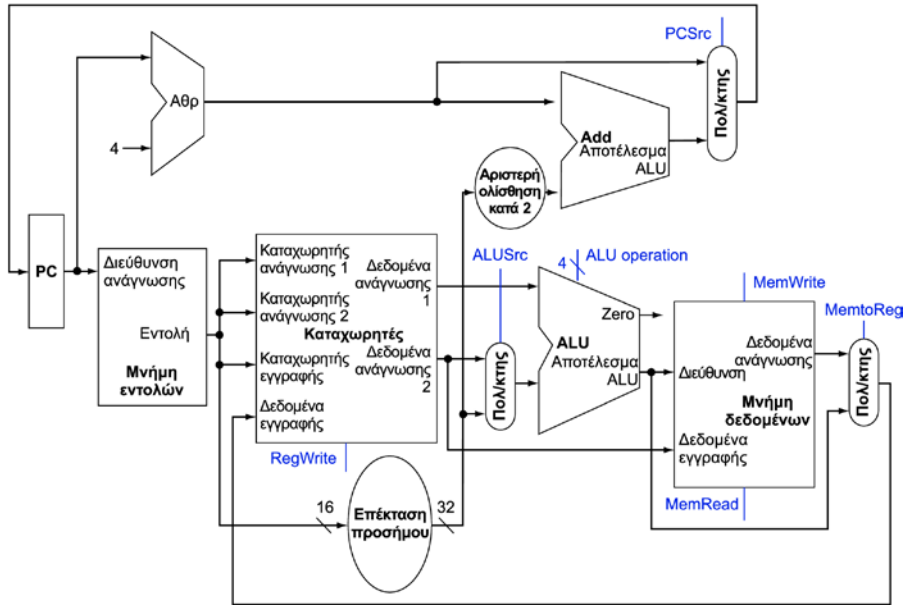
ΚΕΦΑΛΑΙΟ 4

Ο ΕΠΕΞΕΡΓΑΣΤΗΣ

ΑΣΚΗΣΗ 1

Θεωρείστε την παρακάτω απλοποιημένη υλοποίηση του MIPS, όπου οι καθυστερήσεις των μονάδων δίνονται στον ακόλουθο πίνακα:

Μονάδα	**Καθυστέρηση**	**Μονάδα**	**Καθυστέρηση**
Μνήμη Εντολών	450 ps	*Αρχείο Καταχωρητών*	200 ps
Αθροιστές	150 ps	*Μνήμη Δεδομένων*	350 ps
MUX	30 ps	*Επέκταση Προσήμου*	20 ps
ALU	120 ps	*x2 Αριστερή Ολίσθηση*	0 ps



Α. Ποια είναι η τιμή της περιόδου ρολογιού αν πρέπει να υποστηρίξουμε μόνο αριθμητικές και λογικές εντολές (and, add, …) ;

Β. Ποια είναι η τιμή της περιόδου ρολογιού αν πρέπει να υποστηρίξουμε μόνο την εντολή lw;

Γ. Ποια είναι η τιμή της περιόδου ρολογιού αν πρέπει να υποστηρίξουμε μόνο τις εντολές add, beq, lw, sw;

Δ. Θεωρείστε ότι η κατανομή των εντολών κατά την εκτέλεση ενός προγράμματος είναι:

Εντολή	**Ποσοστό (%)**	**Εντολή**	**Ποσοστό (%)**
add	20	beq	20
addi	20	lw	20
and	5	sw	15

Σε ποσοστό χρησιμοποιείται η μνήμη δεδομένων και σε τι ποσοστό χρησιμοποιείται το αποτέλεσμα της μονάδας επέκτασης προσήμου;

Ε. Αν μπορούσαμε να βελτιώσουμε κατά 10% την καθυστέρηση μίας από τις κυκλωματικές μονάδες της υλοποίησης, ποια θα επιλέγατε και ποια η βελτίωση που επιτυγχάνεται στην περίοδο;

ΛΥΣΗ

Α. Σε αυτήν τη υλοποίηση του επεξεργαστή τα υπολογιστικά μονοπάτια ροής δεδομένων (data-flow computation paths) είναι:

Μονοπάτι-1: **Μνήμη Εντολών → Καταχωρητές → MUX** (με σήμα ελέγχου ALUSrc για επιλογή εισόδου στην ALU) → **ALU → MUX** (με σήμα ελέγχου MemtoReg για επιλογή του καταχωρητή εγγραφής).

Μονοπάτι-2: **Μνήμη Εντολών → Καταχωρητές → MUX** (με σήμα ελέγχου ALUSrc για επιλογή εισόδου στην ALU) → **ALU → Μνήμη δεδομένων → MUX** (με σήμα ελέγχου MemtoReg για επιλογή του καταχωρητή εγγραφής).

Μονοπάτι-3: **Μνήμη Εντολών → Επέκταση προσήμου → MUX** (με σήμα ελέγχου ALUSrc για επιλογή εισόδου στην ALU) → **ALU → MUX** (με σήμα ελέγχου MemtoReg για επιλογή του καταχωρητή εγγραφής).

Μονοπάτι-4: **Μνήμη Εντολών → Επέκταση προσήμου → MUX** (με σήμα ελέγχου ALUSrc για επιλογή εισόδου στην ALU) → **ALU → Μνήμη δεδομένων → MUX** (με σήμα ελέγχου MemtoReg για επιλογή του καταχωρητή εγγραφής).

Μονοπάτι-5: **Αθροιστής** (PC+4) → **MUX** (με σήμα επιλογής το PCSrc).

Μονοπάτι-6: **Αθροιστής** (PC+4) → **Αθροιστής** → **MUX** (με σήμα επιλογής το PCSrc).

Μονοπάτι-7: **Μνήμη Εντολών → Επέκταση προσήμου → Ολίσθηση x2 → Αθροιστής → MUX** (με σήμα επιλογής το PCSrc).

Για κάθε τύπο εντολής που πρέπει να υποστηριχτεί, η περίοδος του ρολογιού πρέπει να είναι τουλάχιστον ίση με κρίσιμο μονοπάτι (critical path). Ως κρίσιμο μονοπάτι ορίζεται εκείνο που παρουσιάζει τη μεγίστη καθυστέρηση.

Για να εκτελεστεί μια αριθμητική/λογική εντολή πρέπει: α) να διαβαστεί η εντολή από τη μνήμη εντολών, β) να γίνει προσπέλαση στο αρχείο καταχωρητών για να διαβαστούν οι τιμές των δύο δεδομένων (τελεστέων) της εντολής, γ) να τροφοδοτηθούν στην ALU οι τιμές των τελεστέων δ) να εκτελέσει η ALU την πράξη, ε) να αποθηκευθεί το αποτέλεσμα της πράξης στον καταχωρητή προορισμού.

Επίσης, πρέπει να αυξηθεί και η τιμή του του PC κατά 4 (PC+4), το οποίο απαιτείται για κάθε τύπο εντολής.

Επομένως, για τις αριθμητικές εντολές χρησιμοποιούνται τα μονοπάτια 1, 5 και7.

Με βάση τις καθυστερήσεις των αντίστοιχων μονάδων, οι καθυστερήσεις των δύο αυτών μονοπατιών είναι:

Καθυστέρηση Μονοπάτι-1: (450 + 200 + 30 + 120 + 30) ps = 830 ps.

Καθυστέρηση Μονοπάτι-5 (150 + 30) ps = 180 ps.

Άρα, το κρίσιμο μονοπάτι είναι το Μονοπάτι-1 και επομένως **T = 780 ps**.

Β. Για να εκτελεστεί η εντολή lw (load) πρέπει:

- να διαβαστεί η εντολή από τη μνήμη εντολών,
- να προσπελαστεί το αρχείο καταχωρητών και να διαβαστούν τα δεδομένα του καταχωρητή πηγής (SrcReg), *[SrcReg]*, και η τιμή του offset,
- να γίνει επέκταση προσήμου του offset και να προκύψει η τιμή *offset-με-επέκταση_προσήμου*,
- να τροφοδοτηθεί η ALU με τις δύο εισόδους που είναι *[SrcReg]* και *offset-με-επέκταση_προσήμου*,
- να εκτελέσει η ALU την πράξη *Διεύθυνση_μνήμης_ανάγνωσης* = *[SrcReg]* + *offset-με-επέκταση_προσήμου*,
- να προσπελαστεί η μνήμη δεδομένων και να γίνει ανάγνωση της θέσης με διεύθυνση *Διεύθυνση_μνήμης_ανάγνωσης*,
- να αποθηκευθεί το αποτέλεσμα στον καταχωρητή προορισμού.

Με βάση τα παραπάνω, τα μονοπάτια που εμπλέκονται είναι τα 2, 4, 5, 6, 7. Από αυτά, με βάση τις καθυστερήσεις των κυκλωματικών μονάδων, το κρίσιμο μονοπάτι είναι το Μονοπάτι-2 με καθυστέρηση

Καθυστέρηση Μονοπάτι-2 (450 + 200 + 30 + 120 + 350+ 30) ps = 1180 ps.

Άρα, **T = 1180 ps**.

Γ. Από τα ερωτήματα Α και Β, συμπεραίνουμε ότι αν θέλουμε να υποστηρίξουμε τις εντολές add και lw, τότε την περίοδο ρολογιού την καθορίζει η εντολή lw, η οποία

παρουσιάζει τη μέγιστη καθυστέρηση. Αν θέλουμε να υποστηρίξουμε τις εντολές add, beq, lw και sw παρατηρούμε τα ακόλουθα. Η εντολή sw έχει μικρότερη καθυστέρηση από τη lw καθώς δε συμμετέχει στο κρίσιμο μονοπάτι η μνήμη δεδομένων. Το ίδιο συμβαίνει και με την εντολή beq. Επομένως, **T = 1130 ps.**

Δ. Η μνήμη δεδομένων χρησιμοποιείται μόνο από τις εντολές lw, sw. Επομένως, το ποσοστό χρήσης της μνήμης δεδομένων είναι 20% + 15% = 35%. Η μονάδα επέκταση προσήμου υπολογίζει αποτέλεσμα πάντα (για κάθε εντολή). Όμως, αυτό χρησιμοποιείται για όλες τις εντολές του ερωτήματος εκτός των add και and. Άρα, το ποσοστό χρήσης του αποτελέσματος της μονάδας επέκτασης προσήμου είναι ίσο με: 100-20-5= 75%.

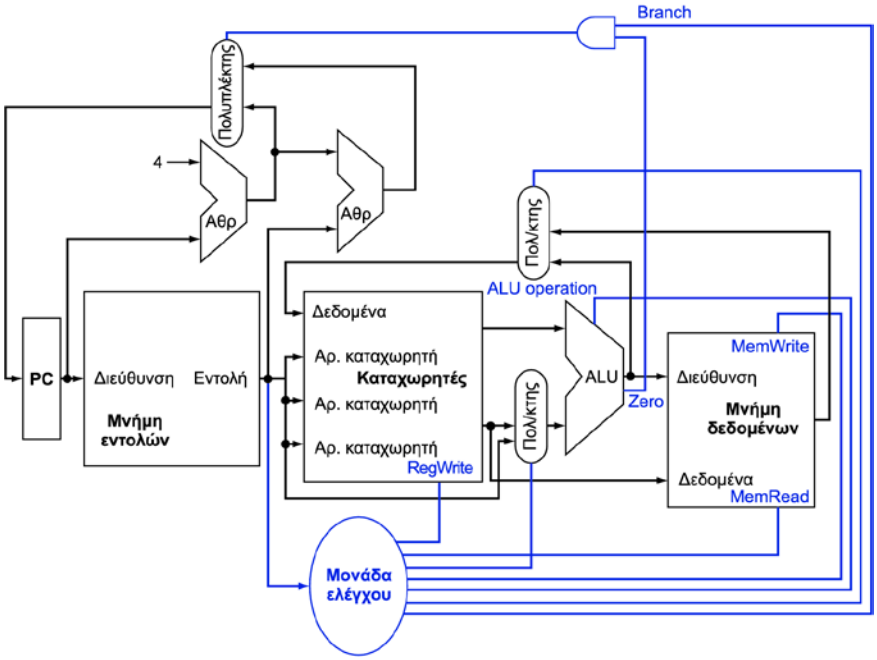
Ε. Όπως είδαμε σε προηγούμενο ερώτημα, η εντολή που παρουσιάζει τη μέγιστη καθυστέρηση και η οποία καθορίζει την περίοδο είναι η εντολή lw με κρίσιμο μονοπάτι το Μονοπάτι-2. Σε αυτό το μονοπάτι η μονάδα με τη μεγαλύτερη καθυστέρηση είναι μνήμη εντολών. Άρα, θα επιλέξουμε να μειώσουμε την καθυστέρηση της μνήμης εντολών. Αφού, η καθυστέρηση αυτής μειώνεται κατά 10%, η νέα καθυστέρηση είναι 405 ps. Επομένως, η νέα περίοδος γίνεται ίση με T_{new} = 1135 ps.

Όσον αφορά τη βελτίωση της επίδοσης του επεξεργαστή έχουμε: $\frac{T_{old}}{T_{new}} = \frac{1180}{1135} = 1.04$

ΑΣΚΗΣΗ 2

Θεωρείστε την παρακάτω απλοποιημένη υλοποίηση του MIPS, όπου οι καθυστερήσεις των μονάδων δίνονται στον ακόλουθο πίνακα:

Μνήμες: 300 ps	***Αθροιστές***: 80 ps	***MUX***: 30 ps
ALU: 140 ps	***Αρχείο Καταχωρητών***: 200 ps	***Μονάδα Ελέγχου***: 100 ps



Α. Πόση είναι η περίοδος του ρολογιού της παραπάνω υλοποίησης; Ποιες μονάδες συμμετέχουν στο κρίσιμο μονοπάτι (critical path);

Β. Με σκοπό τη βελτίωση της εκτέλεσης ενός προγράμματος εξετάζονται δύο αλλαγές στην υλοποίηση. Ποια η επίπτωση της κάθε αλλαγής στην εκτέλεση του προγράμματος; Ποια από τις δύο θα διαλέγατε;

Αλλαγές Υλοποίησης	**Νέα καθυστέρηση μονάδας**	**Επιπτώσεις στην εκτέλεση του προγράμματος**
1. *Ταχύτερος αθροιστής*	60 ps	Ίδιο πλήθος κύκλων ρολογιού
2. *Μεγαλύτερο αρχείο καταχωρητών*	700 ps	20% λιγότεροι κύκλοι ρολογιού

Αγνοείστε τις καθυστερήσεις του PC και των πυλών.

ΛΥΣΗ

Α. Για να βρεθεί η περίοδος του ρολογιού πρέπει να βρεθεί το κρίσιμο μονοπάτι δηλαδή, το μονοπάτι ροής και επεξεργασίας δεδομένων με τη μεγαλύτερη καθυστέρηση. Σε κάθε σύγχρονο ψηφιακό κύκλωμα, ένα τέτοιο μονοπάτι ξεκινά από καταχωρητή και τερματίζει σε καταχωρητή.

Στη συγκεκριμένη υλοποίηση, υπάρχουν αρκετά τέτοια μονοπάτια, όπως για παράδειγμα τα ακόλουθα:

Μονοπάτι-1: Μνήμη εντολών → Αρχείο καταχωρητών → MUX → ALU → Μνήμη δεδομένων → MUX

Μονοπάτι-2: Μνήμη εντολών → Μονάδα Ελέγχου → MUX → ALU → Μνήμη δεδομένων → MUX

Μονοπάτι-3: Αθροιστής → Αθροιστής → MUX

κλπ.

Όμως, το κρίσιμο μονοπάτι είναι αυτό που παρουσιάζει τη μεγαλύτερη καθυστέρηση. Έτσι, με βάση τις καθυστερήσεις των μονάδων, το κρίσιμο μονοπάτι είναι το μονοπάτι-1. Άρα, η περίοδος του ρολογιού είναι: T_{old} = 300+200+30+140+300+30 = 1000 ps. Όσον αφορά τις μονάδες που μονάδες συμμετέχουν στο κρίσιμο μονοπάτι, αυτές έχουν δοθεί στην περιγραφή του Μονοπάτι-1.

Β. Εδώ πρέπει να εξεταστούν δύο κυκλωματικές αλλαγές. Η 1η από αυτές συνίσταται στην αλλαγή των αθροιστών από ταχύτερους αθροιστές. Όμως, καθώς κανένας αθροιστής δε περιλαμβάνεται στο κρίσιμο μονοπάτι, η χρήση των νέων αθροιστών δε θα βελτιώσει την περίοδο του ρολογιού. Επίσης, ο νέος αθροιστής εισάγει επιπλέον κόστος ανάπτυξης αφού πρέπει να επανασχεδιασθεί και υλοποιηθεί με πιο έξυπνο τρόπο ώστε να παρουσιάζει μικρότερη καθυστέρηση. Άρα, λόγω του ότι δεν υπάρχει βελτίωση της περιόδου ρολογιού και του επιπλέον κόστους, η επιλογή αυτή απορρίπτεται.

Στη δεύτερη αλλαγή έχει αυξηθεί η καθυστέρηση του αρχείου καταχωρητών. Επομένως, το κρίσιμο μονοπάτι εξακολουθεί και είναι το Μονοπάτι-1. Συγκεκριμένα, αφού αρχικά το κρίσιμο μονοπάτι ήταν το Μονοπάτι-1, το οποίο περιέχει το αρχείο καταχωρητών, η αύξηση της καθυστέρησης του αρχείου καταχωρητών διατηρεί ως κρίσιμο μονοπάτι το Μονοπάτι-1. Όμως, η τιμή της περιόδου αλλάζει και τώρα ισούται με : T_{new} = 300+**700**+30+140+300+30 = 1500 ps.

Η αύξηση της περιόδου κάνει μεν αργότερο τον επεξεργαστή, αλλά μειώνει κατά 20% τους κύκλους ρολογιού που απαιτούνται για την εκτέλεση του προγράμματος. Επομένως, για αποτιμήσουμε την προτεινόμενη αλλαγή πρέπει να συγκρίνουμε τους χρόνους εκτέλεσης του προγράμματος για τις δύο περιπτώσεις.

Ο χρόνος εκτέλεσης, t, ισούται με : t = (κύκλοι ρολογιού) x (Περίοδος). Άρα, έχουμε:

$$\frac{t_{new}}{t_{old}} = \frac{(cycles_{new}) \times (T_{new})}{(cycles_{old}) \times (T_{old})} = \frac{(0{,}8 \times cycles_{old}) \times 1500}{(cycles_{old}) \times 1000} = 1{,}2 .$$

Επομένως, ο νέος χρόνος εκτέλεσης του προγράμματος είναι 1,2 φορές μεγαλύτερος από τον αρχικό. Άρα, και αυτή η αλλαγή απορρίπτεται.

ΑΣΚΗΣΗ 3

Θεωρείστε τις ακόλουθες καθυστερήσεις των σταδίων διαδρομής δεδομένων

IF	**ID**	**EX**	**MEM**	**WB**
200 ps	400 ps	300 ps	500 ps	100 ps

Α. Ποια είναι η περίοδος ρολογιού σε έναν επεξεργαστή με διοχέτευση 5 σταδίων και χωρίς διοχέτευση;

Β. Πόσος χρόνο διαρκεί η εκτέλεση της εντολής lw σε έναν επεξεργαστή με διοχέτευση και χωρίς διοχέτευση;

Γ. Αν είχατε να εκτελέσετε ένα πρόγραμμα με 3 εντολές και ένα δεύτερο πρόγραμμα με 100 εντολές, ποια υλοποίηση (με διοχέτευση, χωρίς διοχέτευση) θα επιλέγατε σε κάθε περίπτωση;

ΛΥΣΗ

Α. Στην περίπτωση που έχουμε διοχέτευση 5 σταδίων, η περίοδος καθορίζεται από το στάδιο (IF, ID, EX, MEM, WB) που έχει τη μεγαλύτερη καθυστέρηση. Επομένως, για τη συγκεκριμένη περίπτωση, το στάδιο με τη μεγαλύτερη καθυστέρηση είναι το MEM και άρα: T_{pipe} = 500 ps.

Στην περίπτωση που δεν έχουμε διοχέτευση και δεδομένου ότι η εντολή lw χρησιμοποιεί όλα τα στάδια, η περίοδος είναι ίση με: $T_{non\text{-}pipe}$ = (200 + 400+ 300 +500 + 100) ps = 1500 ps.

Β. Στην περίπτωση που έχουμε διοχέτευση 5 σταδίων, κάθε εντολή δαπανά 5 κύκλους ρολογιού για την εκτέλεση της. Επομένως,

Χρόνος εκτέλεσης lw με pipeline = 5 x 500 = 2500 ps.

Στην περίπτωση που δεν έχουμε διοχέτευση και δεδομένου ότι η εντολή lw χρησιμοποιεί όλα τα στάδια, ο χρόνος εκτέλεσης της lw ισούται με την αντίστοιχη περίοδο του ρολογιού. Συνεπώς,

Χρόνος εκτέλεσης lw χωρίς pipeline = 1500 ps.

Παρατηρούμε ότι ο χρόνος εκτέλεσης μίας εντολής είναι μεγαλύτερος στην περίπτωση που έχουμε διοχέτευση, το οποίο μπορεί να παραπλανήσει και να οδηγήσει στο συμπέρασμα ότι η διοχέτευση είναι άχρηστη. Όμως, όπως θα δούμε στο επόμενο ερώτημα, αυτό δεν είναι σωστό.

Γ. Για την περίπτωση του επεξεργαστή χωρίς διοχέτευση, κάθε εντολή δαπανά έναν κύκλο ρολογιού με περίοδο $T_{non\text{-}pipe}$. Άρα, ο χρόνος εκτέλεσης του προγράμματος είναι ίσος με:

Χρόνος εκτέλεσης χωρίς pipeline = $n \times T_{non\text{-}pipe}$ όπου n το πλήθος των εντολών του προγράμματος.

Για την περίπτωση του επεξεργαστή με διοχέτευση 5 σταδίων, το αποτέλεσμα της 1ης εντολής είναι διαθέσιμο μετά από 5 κύκλους ρολογιού, ενώ στη συνέχεια για όλες τις υπόλοιπες εντολές το αποτέλεσμα τους παράγεται σε κάθε κύκλο. Επομένως, οι συνολικοί κύκλοι που δαπανώνται (5 + (n-1)) και η περίοδος είναι ίση με T_{pipe}. Συνεπώς, ο χρόνος εκτέλεσης του προγράμματος είναι ίσος με:

Χρόνος εκτέλεσης με pipeline = $(5 + (n\text{-}1)) \times T_{pipe}$

Άρα, με βάση τα παραπάνω έχουμε:

Για την περίπτωση των 3 εντολών έχουμε:

Χρόνος εκτέλεσης χωρίς pipeline = 3×1500 = 4500 ps.

Χρόνος εκτέλεσης με pipeline = (5 + 2)×500 = 3500 ps.

Για την περίπτωση των 100 εντολών έχουμε:

Χρόνος εκτέλεσης χωρίς pipeline = 100×1500 = 150000 ps.

Χρόνος εκτέλεσης με pipeline = (5 + 99)×500 = 52000 ps.

Και στις δύο περιπτώσεις η υλοποίηση με διοχέτευση είναι ταχύτερη. Μάλιστα, όταν οι εντολές του προγράμματος είναι πολλές (εκατοντάδες), όπως συμβαίνει σε ρεαλιστικές εφαρμογές, η υλοποίηση με διοχέτευση είναι μακράν καλύτερη.

ΑΣΚΗΣΗ 4

Θεωρείστε τον παρακάτω κώδικα (εντολές I1, I2, I3), που εκτελείται στον επεξεργαστή MIPS με 5 στάδια pipeline:

I1: lw $4, 40 ($4)

I2: sw $4, 50 ($4)

I3: add $4, $4, $4

Α. Βρείτε τις εξαρτήσεις (κινδύνους) δεδομένων

Β. Επιλύστε τα προβλήματα που προκύπτουν λόγω των εξαρτήσεων δεδομένων θεωρώντας ότι ο επεξεργαστής δεν έχει μονάδα προώθησης (forwarding unit)

Γ. Επιλύστε τα προβλήματα που προκύπτουν λόγω των εξαρτήσεων δεδομένων θεωρώντας ότι ο επεξεργαστής έχει μονάδα προώθησης (forwarding unit).

ΛΥΣΗ

Α. Όπως είναι γνωστό, οι εξαρτήσεις (κίνδυνοι) δεδομένων είναι οι ακόλουθες:

RAW (Read-After-Write) που σημαίνει ότι ένας καταχωρητής γράφεται από μία εντολή και στη συνέχεια τα δεδομένα του διαβάζονται (δηλ, χρησιμοποιούνται) από μία επόμενη εντολή

WAR (Write-After-Read) που σημαίνει τα δεδομένα ενός καταχωρητή διαβάζονται (χρησιμοποιούνται) από μία εντολή και στην συνέχεια γράφονται (τροποποιούνται) από μία επόμενη

WAW (Write-After-Write) που σημαίνει ότι τα δεδομένα ενός καταχωρητή τροποποιούνται δύο φορές από δύο εντολές

Στον επεξεργαστή MIPS όπου οι εντολές εκτελούνται με τη σειρά που είναι γραμμένες (in order), ο μόνος κίνδυνος δεδομένων που μπορεί να προκαλέσει λανθασμένους υπολογισμούς κατά την εκτέλεση του προγράμματος είναι ο κίνδυνος WAR.

Πιο συγκεκριμένα, για δύο εντολές με εξάρτηση τύπου RAW, ο λανθασμένος υπολογισμός θα προκύψει ως εξής: όταν η 2η εντολή, που διαβάζει τα δεδομένα του καταχωρητή που προκαλεί την εξάρτηση, έχει ήδη εισέλθει και φτάσει στο 2ο στάδιο του pipeline, όπου γίνεται ανάγνωση των καταχωρητών, ενώ η 1η εντολή δεν έχει ακόμη φτάσει στο στάδιο WB, ώστε να ενημερώσει (να γράψει) το αρχείο καταχωρητών. Αποτέλεσμα αυτού είναι ότι η 2η εντολή διαβάσει παλιά δεδομένα για τον εν λόγω καταχωρητή και όχι αυτά που αντιστοιχούν στην εγγραφή του καταχωρητή από την 1η εντολή.

Ο κώδικας που πρέπει να μελετηθεί είναι:

I1: lw $4, 40 ($4)

I2: sw $4, 50 ($4)

I3: add $4, $4, $4

Λαμβάνοντας υπόψη τη λειτουργία της κάθε εντολής έχουμε:

Εντολή	**Ανάγνωση**			**Εγγραφή**		
	Καταχωρητές	**Στάδιο pipeline**	**Κύκλος ρολογιού**	**Καταχωρητές**	**Στάδιο pipeline**	**Κύκλος ρολογιού**
I1	$4	2ο (ID)	2+0 = 2	$4	5ο (WB)	5+0 = 5
I2	$4	2ο (ID)	2+1 = 3	–	–	–
I3	$4, $4	2ο (ID)	2+2 = 4	$4	5ο (WB)	5+2 = 7

Η εντολή I1 (lw $4, 40 ($4)) εκτελεί τη λειτουργία ($4) ← mem [40+($4)], δηλαδή, αποθηκεύει στον καταχωρητή $4 την τιμή που βρίσκεται στη μνήμη με διεύθυνση [40+($4)]. Για να το κάνει αυτό, διαβάζει την τιμή του $4 στο στάδιο ID (2ο στάδιο), εκτελεί την πράξη 40+($4) στο στάδιο EX (3ο στάδιο), προσπελαύνει (διαβάζει) την διεύθυνση μνήμης [40+($4)] στο στάδιο MEM (4ο στάδιο) και γράφει το δεδομένο της διεύθυνσης μνήμης στον $4 στο στάδιο WB (5ο στάδιο). Εφόσον, η I1 είναι η 1η εντολή, οι καταχωρητές διαβάζονται και γράφονται στον 2ο και 5ο κύκλο ρολογιού, αντίστοιχα. Με τον ίδιο τρόπο εξηγείται η λειτουργία και των επόμενων εντολών, ανάλογα βέβαια με τον τύπο της κάθε μίας από αυτές.

Πρέπει να είμαστε όμως προσεκτικοί με τους κύκλους ρολογιού που γίνονται η ανάγνωση και εγγραφή των καταχωρητών. Όλες οι εντολές στον MIPS διαβάζουν και γράφουν τους καταχωρητές στο 2ο και 5ο pipeline στάδιο, αντίστοιχα. Όμως, για τον καθορισμό του κύκλου ρολογιού όπου συμβαίνουν αυτές οι λειτουργίες, πρέπει να ληφθεί υπόψη η καθυστέρηση ως προς την 1η εντολή με την οποία οι εντολές εισέρχονται στο pipeline. Για παράδειγμα, η 3η εντολή (I3) εισέρχεται στο pipeline με καθυστέρηση 2 κύκλους ρολογιού ως προς την 1η εντολή. Επομένως, η ανάγνωση των καταχωρητών γίνεται στον κύκλο 4 (2 κύκλοι αφού η ανάγνωση γίνεται στο 2ο pipeline στάδιο και 2 επιπλέον κύκλοι αφού η εντολή έχει 2 κύκλους καθυστέρηση ως προς την 1η εντολή).

Μελετώντας τον παραπάνω πίνακα συμπεραίνουμε τα παρακάτω.

Η εντολή I2 διαβάζει τον καταχωρητή $4 στον κύκλο 3. Με βάση τη ροή εκτέλεσης των εντολών, η I2 θα πρέπει να διαβάσει από τον $4 το αποτέλεσμα της I1, αφού η I1 τροποποιεί (γράφει) τον $4. Όμως, η I1 ενημερώνει (γράφει) τον $4 στον κύκλο 5. Δηλαδή, στον κύκλο 3 η I2 δε διαβάζει από τον $4 το αποτέλεσμα της I1 αλλά την αρχική τιμή του $4. Άρα, υπάρχει κίνδυνος δεδομένων RAW μεταξύ των εντολών I1, I2 και του $4.

Επίσης, η εντολή I3 διαβάζει τον $4 στον κύκλο 5 και η εντολή που έχει αλλάξει τελευταία τα δεδομένα του $4 είναι η I1 (η I2 δεν αλλάζει τα δεδομένα του $4). Όμως,

η I1 ενημερώνει τον $4 στον κύκλο 7. Επομένως, υπάρχει κίνδυνος RAW μεταξύ των εντολών I1, I3 και του $4.

Β. Οι κίνδυνοι δεδομένων αντιμετωπίζονται με τρεις τρόπους, που είναι: α) αναδιάταξη εντολών, β) εισαγωγή εντολών NOP και γ) με χρήση της μονάδας προώθησης, αν υπάρχει.

Στη συγκεκριμένη περίπτωση δεν έχουμε μονάδα προώθησης, ενώ δε μπορεί να γίνει αναδιάταξη εντολών γιατί αλλάζει το τη λειτουργία του προγράμματος. Επομένως, πρέπει να γίνει εισαγωγή εντολών NOP.

Επίσης, πρέπει να σημειωθεί ότι ο MIPS είναι υλοποιημένος ώστε να επιτρέπεται εγγραφή και ανάγνωση καταχωρητή στον ίδιο κύκλο (η εγγραφή γίνεται στο πρώτο μισό και η ανάγνωση του ίδιου καταχωρητή μπορεί να γίνει στο δεύτερο μισό του ίδιου κύκλου).

Όπως είδαμε, υπάρχει ένας κίνδυνος RAW μεταξύ των εντολών I1, I2 και του $4. Καθώς η I1 γράφει τον $4 στον κύκλο 5 ενώ η I2 τον διαβάζει στον κύκλο 3, πρέπει η ανάγνωση του $4 από τη I2 να γίνει στον κύκλο 5. Συνεπώς, πρέπει να εισαχθούν 2 NOP εντολές μεταξύ των I1 και I2.

Επίσης, είδαμε στο ερώτημα Α ότι υπάρχει εξάρτηση δεδομένων μεταξύ των εντολών I1, I3 και του $4. Συγκεκριμένα, η I3 διαβάζει στον κύκλο 4 ενώ η I1 γράφει τον $4 στον κύκλο 5. Αυτό σημαίνει ότι η I3 πρέπει να καθυστερήσει έναν κύκλο. Όμως, αυτό έχει ήδη επιτευχθεί καθώς έχουμε βάλει δύο NOP μεταξύ των I1 και I2.

Με βάση τα παραπάνω, το νέο πρόγραμμα είναι το εξής

```
I1:   lw $4, 40 ($4)
      NOP
      NOP
I2:   sw $4, 50 ($4)
I3:   add $4, $4, $4
```

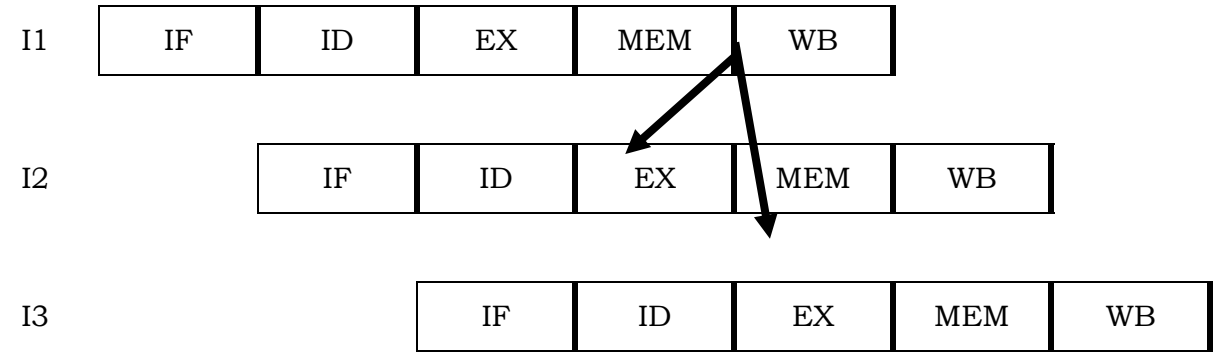
και ο αντίστοιχος πίνακας ο παρακάτω.

Εντολή	**Ανάγνωση**			**Εγγραφή**		
	Καταχωρητές	**Στάδιο pipeline**	**Κύκλος ρολογιού**	**Καταχωρητές**	**Στάδιο pipeline**	**Κύκλος ρολογιού**
I1	$4	2ο (ID)	2+0 = 0	$4	5ο (WB)	5+0 = 5

NOP						
NOP						
I2	$4	2ο (ID)	2+3= 5			
I3	$4, $4	2ο (ID)	2+4 = 6	$4	5ο (WB)	5+2 = 7

Γ. Όπως είναι γνωστό, με τη χρήση της μονάδας προώθησης είναι δυνατή η προώθηση αποτελεσμάτων που βρίσκονται στα στάδια EX και MEM στις εισόδους της ALU. Αν δούμε γραφικά την κατάσταση του pipeline κατά την εκτέλεση των τριών εντολών τότε έχουμε το ακόλουθο σχήμα

Εντολή	Κύκλος 1	Κύκλος 2	Κύκλος 3	Κύκλος 4	Κύκλος 5	Κύκλος 6	Κύκλος 7



Στο παραπάνω σχήμα τα βέλη δηλώνουν τις εξαρτήσεις δεδομένων λόγω του $4 μεταξύ των εντολών I1, I2 και I1, I3, ενώ οι έντονες μαύρες γραμμές αντιστοιχούν στο καταχωρητές μεταξύ των σταδίων pipeline. Καθώς η εντολή I1 είναι εντολή φόρτωσης, τα δεδομένα διαβάζονται από τη μνήμη στον 4ο κύκλο και είναι διαθέσιμα στον 5ο κύκλο.

Για να εκτελέσει την πράξη ($4) ← ($4) + ($4), η εντολή I3 χρειάζεται τα δεδομένα του $4 στο στάδιο EX δηλαδή, στον 5ο κύκλο. Επομένως, η μονάδα προώθησης μπορεί να προωθήσει τα δεδομένα αυτά χωρίς πρόβλημα.

Όμως, η εντολή I2 πρέπει να εκτελέσει τον υπολογισμό 50 +($4) στον κύκλο 4, όπου τα δεδομένα του $4 δεν είναι διαθέσιμα (αυτά όπως αναφέρθηκε είναι διαθέσιμα στον 5ο κύκλο ρολογιού). Άρα, ο συγκεκριμένος κίνδυνος εξάρτησης δεδομένων δε μπορεί να επιλυθεί ούτε με τη μονάδα προώθησης. Επομένως, για την αντιμετώπιση αυτού του κινδύνου απαιτείται η εισαγωγή μιας εντολής NOP μεταξύ των I1 και I2.

ΑΣΚΗΣΗ 5

Για τον παρακάτω κώδικα που εκτελείται στον επεξεργαστή MIPS με 5 στάδια pipeline

I1: lw $1, 40($8)

I2: lw $2, 30($7)

I3: or $3, $1, $2

I4: sw $3, 10($7)

I5: lw $4, 50($9)

I6: lw $5, 60($10)

I7: add $6, $4, $5

I8: lw $6, 60($7)

Α. Βρείτε τις εξαρτήσεις δεδομένων

Β. Επιλύστε τους κινδύνους δεδομένων με χρήση εντολών NOP

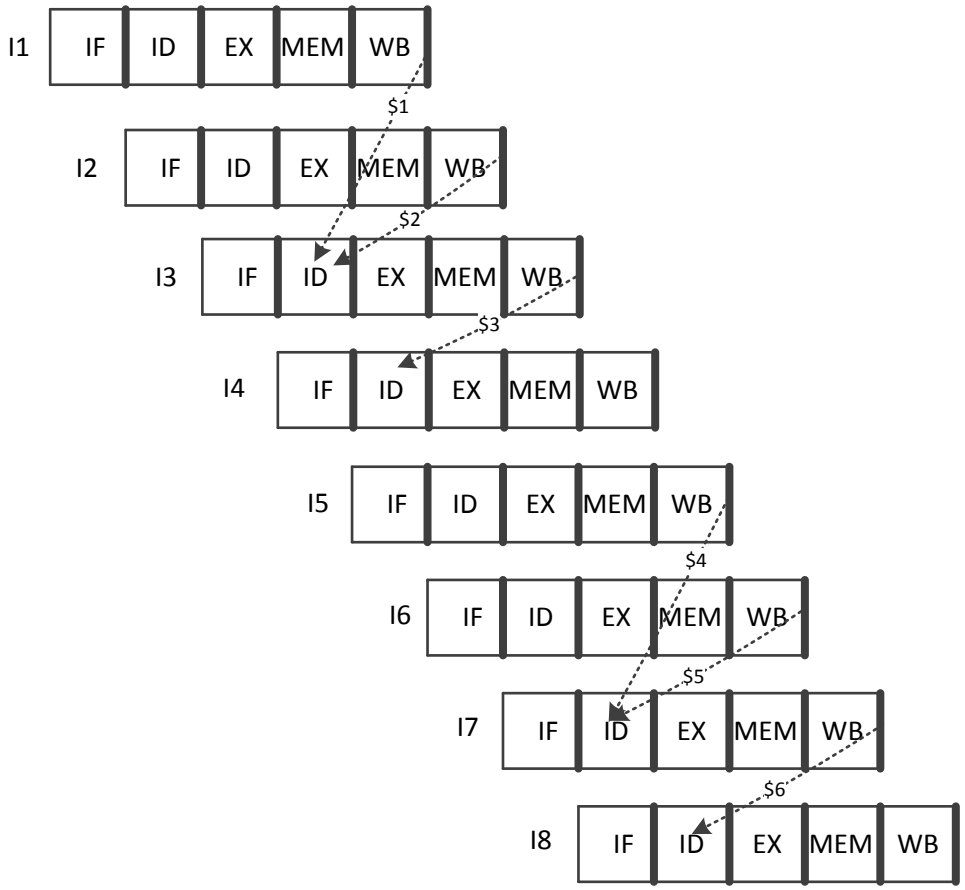
Γ. Επιλύστε τους κινδύνους δεδομένων με αναδιάταξη εντολών και συγκρίνετε το πλήθος των εντολών των προγραμμάτων των περιπτώσεων Β και Γ.

ΛΥΣΗ

Α. Από τη μελέτη του παραπάνω κώδικα προκύπτουν οι ακόλουθες εξαρτήσεις δεδομένων τύπου RAW.

Εξαρτώμενες εντολές	**Καταχωρητής εξάρτησης**	**Εξαρτώμενες εντολές**	**Καταχωρητής εξάρτησης**
I1, I3	$1	I5, I7	$4
I2, I3	$2	I6, I7	$5
I3, I4	$3	I7, I8	$6

Η γραφική απεικόνιση των εξαρτήσεων κατά την εκτέλεση των εντολών είναι:



Β.

Ο κίνδυνος (I1, I3, $1) επιλύεται με την εισαγωγή μίας NOP πριν τη I3. Επίσης, ο κίνδυνος (I2, I3, $2) επιλύεται με την εισαγωγή δύο NOP πριν τη I3. Συνεπώς, αν εισάγουμε 2 NOP πριν την I3 αντιμετωπίζουμε και τους δύο κινδύνους.

Ο κίνδυνος (I3, I4, $3) επιλύεται με την εισαγωγή δύο NOP πριν τη I4.

Επίσης, οι κίνδυνοι (I5, I7, $4) και (I6, I7, $5) επιλύονται με την εισαγωγή 2 NOP πριν την I7 (όπως έγινε για τους κινδύνους (I1, I3, $1) και (I2, I3, $2)).

Τέλος, ο κίνδυνος (I7, I8, $6) επιλύεται με την εισαγωγή δύο NOP πριν τη I8.

Επομένως, το νέο πρόγραμμα είναι:

```
I1:   lw    $1, 40($8)
I2:   lw    $2, 30($7)
      NOP
      NOP
I3:   or    $3, $1, $2
```

```
        NOP
        NOP
I4:     sw      $3, 10($7)
I5:     lw      $4, 50($9)
I6:     lw      $5, 60($10)
        NOP
        NOP
I7:     add     $6, $4, $5
        NOP
        NOP
I8:     lw      $6, 60($7)
```

και το πλήθος των εντολών έγινε 16 από 8, που ήταν αρχικά.

Γ.

Οι κίνδυνοι δεδομένων μπορούν επίσης να αντιμετωπιστούν κατά τη μεταγλώττιση με αναδιοργάνωση των εντολών. Όμως, πρέπει να δοθεί ιδιαίτερη προσοχή ώστε να μην αλλοιωθεί σε καμία περίπτωση η λειτουργικότητα του προγράμματος. Δηλαδή, τα αποτελέσματα των υπολογισμών μετά την αναδιοργάνωση των εντολών πρέπει να είναι ίδια με τα αποτελέσματα του αρχικού κώδικα.

Παρατηρώντας τους κινδύνους δεδομένων όπως μελετήθηκαν στο ερώτημα Α, προκύπτει ότι οι κίνδυνοι εμφανίζονται μεταξύ μίας εντολής lw και μίας επόμενης εντολής (εντολή Χ) που χρησιμοποιεί το αποτέλεσμα της lw. Ο κίνδυνος αυτού του είδους αντιμετωπίζεται εισάγοντας δύο επιπλέον εντολές μεταξύ της lw και της εντολής Χ. Αυτό έγινε συγκεκριμένα στο ερώτημα Β με την εισαγωγή των στολών NOP.

Στην προκειμένη περίπτωση όμως προσπαθούμε να μειώσουμε το πλήθος των NOP. Για να το πετύχουμε αυτό, μπορούμε να αντικαταστήσουμε (όσες το δυνατόν περισσότερες) εντολές NOP με επόμενες εντολές του προγράμματος. Μελετώντας το συγκεκριμένο κώδικα, παρατηρούμε ότι θα μπορούσαμε να κάνουμε τις φορτώσεις των καταχωρητών στην αρχή και μετά να εκτελέσουμε τις πράξεις μεταξύ των καταχωρητών φροντίζοντας οι εντολές που αφορούν την εκτέλεση των πράξεων να εκτελούνται τουλάχιστον δύο κύκλους αργότερα από τις αντίστοιχες εντολές lw.

Έτσι, με βάση τα παραπάνω, ένας τέτοιος κώδικας είναι ο ακόλουθος, οποίος αποτελείται από 10 εντολές.

```
I1:   lw    $1, 40($8)
I2:   lw    $2, 30($7)
I5:   lw    $4, 50($9)
I6:   lw    $5, 60($10)
I3:   or    $3, $1, $2
      NOP
I7:   add   $6, $4, $5
I4:   sw    $3, 10($7)
      NOP
I8:   lw    $6, 60($7)
```

ΑΣΚΗΣΗ 6

Θεωρείστε ένα υπολογιστικό σύστημα με επεξεργαστή MIPS με πέντε στάδια pipeline, το οποίο έχει μόνο μία φυσική μνήμη που χρησιμοποιείται τόσο για εντολές όσο και για δεδομένα. Επίσης, θεωρείστε ότι όλες οι διακλαδώσεις προβλέπονται τέλεια (δεν υπάρχουν κίνδυνοι ελέγχου) και ότι δε χρησιμοποιούνται υποδοχές καθυστέρησης (delay slots).

Για αυτό το σύστημα θεωρείστε τους παρακάτω κώδικες:

Α)

```
I1          lw  $1,40($6)
I2          beq $2,$0,Label ; υποθέστε ότι $2 == $0
I3          sw  $6,50($2)
Label: I4   add $2,$3,$4
I5          sw  $3,50($4)
```

Β)

```
I1    lw $5,-16($5)
I2    sw $4,-16($4)
I3    lw $3,-20($4)
I4    beq $2,$0,Label ; υποθέστε ότι $2 != $0
I5    add $5,$1,$4
```

Εξετάστε αν υπάρχουν δομικοί κίνδυνοι (structural hazards) που εμποδίζουν τη σωστή εκτέλεση των παραπάνω κωδίκων και εκτελέστε τις απαραίτητες τροποποιήσεις ώστε αυτοί να εκτελεστούν σωστά.

ΛΥΣΗ

Α. Για την 1η περίπτωση, καθώς οι εντολές εισέρχονται προς εκτέλεση, η κατάσταση του pipeline είναι η ακόλουθη:

Εντολή	Κύκλοι ρολογιού							
	1	2	3	4	5	6	7	8
I1	IF	ID	EX	**MEM**	WB			
I2		IF	ID	EX	MEM	WB		
I3	Δεν εκτελείται. Είναι πάντα $2==0 και επομένως η διακλάδωση στέλνει την εκτέλεση στην add							

I4	IF	ID	EX	MEM	WB	
I5		**IF**	ID	EX	MEM	WB

Παρατηρούμε ότι στον 4ο κύκλο ρολογιού γίνεται προσπέλαση της μνήμης δεδομένων από την εντολή I1 (ανάγνωση της διεύθυνσης [40+($6)]) και ταυτόχρονα προσκόμιση της εντολής I5 από τη μνήμη εντολών. Όμως, σύμφωνα με την εκφώνηση, το σύστημα περιέχει μία μνήμη και επομένως στον κύκλο 4 γίνεται ταυτόχρονη προσπέλαση της μνήμης από δύο εντολές (I1, I5). Είναι φανερό ότι αυτό δεν μπορεί να γίνει καθώς η μνήμη μπορεί να εξυπηρετήσει μία προσπέλαση κάθε φορά.

Η περίπτωση αυτή είναι ένας δομικός κίνδυνος/σύγκρουση (resource hazard/conflict). Πρέπει να σημειωθεί ότι στην πραγματική υλοποίηση του MIPS αυτό δε συμβαίνει γιατί οι μνήμες εντολών και δεδομένων υλοποιούνται από δύο ξεχωριστές φυσικές μνήμες.

Η επίλυση αυτού του κίνδυνου μπορεί να γίνει με την εισαγωγή εντολών NOP που έχει ως συνέπεια την καθυστέρηση της εκτέλεσης της I5 δηλαδή, τη μετακίνηση του σταδίου IF της I5 προς τα δεξιά (σε κύκλο μεγαλύτερο του 4). Έτσι, εισάγοντας μία NOP πριν τη I5 έχουμε:

Εντολή — Κύκλοι ρολογιού

	1	2	3	4	5	6	7	8	9
I1	IF	ID	EX	MEM	WB				
I2		IF	ID	EX	**MEM**	WB			
I3	Δεν εκτελείται. Είναι πάντα $2==0 και επομένως η διακλάδωση στέλνει την εκτέλεση στην add								
I4			IF	ID	EX	MEM	WB		
NOP									
I5					**IF**	ID	EX	MEM	WB

Τώρα το στάδιο IF της I5 εκτελείται στον 5ο κύκλο στον οποίο εκτελείται επίσης και το στάδιο MEM της I2, το οποίο δείχνει ότι πάλι έχουμε σύγκρουση κυκλωματικού πόρου (μνήμης). Όμως, αυτό δε συμβαίνει γιατί η I2 (beq) δεν προσπελαύνει τη μνήμη δεδομένων. Τέλος, οι κύκλοι που απαιτούνται για την εκτέλεση του κώδικα είναι 9. Το

πλήθος των κύκλων που χρειάζεται για να εκτελεστούν *n* εντολές υπολογίζεται στη γενική περίπτωση από τον ακόλουθο τύπο

$$Κύκλοι\_εκτέλεσης = 5 + (n-1)$$

Εδώ έχουμε 5 εντολές (I1, I2, I4, I5 και μία NOP) άρα Κύκλοι εκτέλεσης = 5 + (5-1)=9 κύκλοι.

Β. Για τη δεύτερη περίπτωση έχουμε:

Εντολή — Κύκλοι ρολογιού

	1	2	3	4	5	6	7	8	9
I1	IF	ID	EX	**MEM**	WB				
I2		IF	ID	EX	MEM	WB			
I3			IF	ID	EX	MEM	WB		
I4				**IF**	ID	EX	MEM	WB	
I5					IF	ID	EX	MEM	WB

Είναι φανερό ότι υπάρχει σύγκρουση κυκλωματικών πόρων (resource conflict) καθώς στον 4ο κύκλο, η εντολή I1 κάνει προσπέλαση δεδομένων και I4 εκτελεί προσκόμιση εντολής. Όπως είδαμε, για την αντιμετώπιση αυτού του προβλήματος πρέπει να καθυστερήσει η εκτέλεση στου σταδίου IF της I4.

Αν η I4 καθυστερήσει κατά ένα κύκλο (εισαγωγή μιας NOP), τότε το στάδιο IF εκτελείται στον 5ο κύκλο. Όμως τώρα υπάρχει σύγκρουση μεταξύ της I2 και I4, όπως φαίνεται στον επόμενο πίνακα, άρα χρειάζεται επιπλέον καθυστέρηση. Αν η I4 καθυστερήσει κατά δύο κύκλους (εισαγωγή δύο NOP), τότε το στάδιο IF της I4 εκτελείται στον 6ο κύκλο, όπου πάλι υπάρχει σύγκρουση μεταξύ των I3 και I4. Αν η I4 καθυστερήσει κατά 3 κύκλους (εισαγωγή τριών NOP) τότε το στάδιο IF της I4 εκτελείται στον 7ο κύκλο χωρίς να υπάρχει σύγκρουση κυκλωματικών πόρων με καμία από τις προηγούμενες εντολές, όπως φαίνεται στον επόμενο πίνακα. Όμως, η καθυστέρηση της I4 κατά τρεις κύκλους επιβάλει και την αντίστοιχη καθυστέρηση στην I5 που ακολουθεί και η οποία αρχίζει να εκτελείται στον 8ο κύκλο.

Εντολή — Κύκλοι ρολογιού

	1	2	3	4	5	6	7	8	9
I1	IF	ID	EX	**MEM**	WB				

I2	IF	ID	EX	**MEM**	WB			
I3		IF	ID	EX	**MEM**	WB		
I4			**IF**	**IF**	**IF**	IF	ID	…..
I5							IF	ID

Συμπερασματικά, για την αντιμετώπιση του κίνδυνου εισάγονται τρεις NOP μεταξύ της I3 και I4 και συνολικός αριθμός των εντολών είναι τώρα οκτώ. Επομένως, με βάση τον παραπάνω τύπο ο συνολικός αριθμός των κύκλων εκτέλεσης είναι ίσος με: Κύκλοι_εκτέλεσης = 5 + (8-1) = 12 κύκλοι.

ΑΣΚΗΣΗ 7

Θεωρείστε τους παρακάτω κώδικες που εκτελούνται σε έναν επεξεργαστή MIPS με 5 στάδια pipeline.

Για κάθε περίπτωση βρείτε τις εξάρτησης δεδομένων και μελετήστε πως αυτοί αντιμετωπίζονται όταν ο επεξεργαστής περιλαμβάνει μονάδα προώθησης (forwarding unit).

Α)

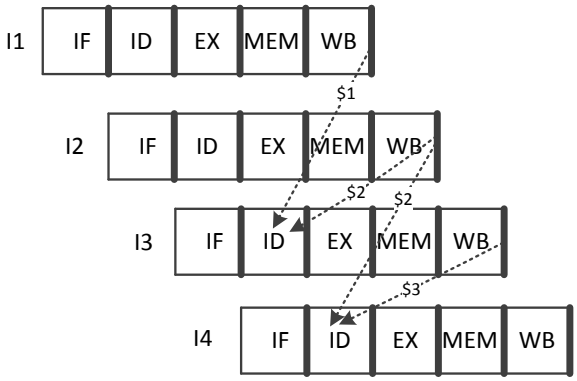
```
I1      lw  $1, 40($6)
I2      and $2, $4, $4
I3      sub $3, $1, $2
I4      add $1, $2, $3
```

Β)

```
I1    and $1, $1, $2
I2    add $2, $1, $2
I3    lw  $3,-20($3)
I4    sub $4, $2, $4
```

ΛΥΣΗ

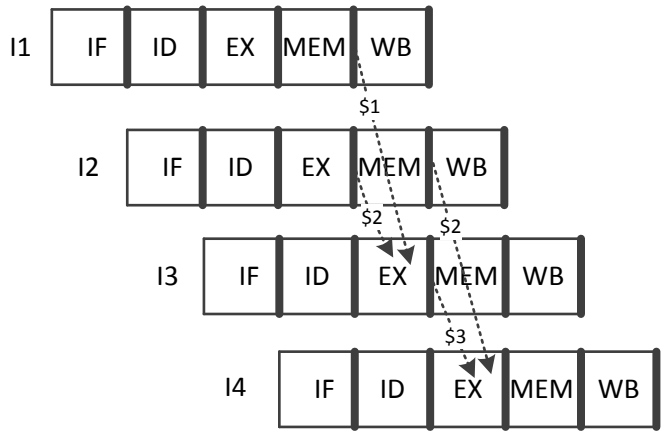
Α. Η εκτέλεση των εντολών οδηγεί στην ακόλουθη κατάσταση του pipeline



Σχ. 1

Είναι φανερό από το παραπάνω σχήμα ότι υπάρχουν οι ακόλουθες κίνδυνοι δεδομένων: (I1, I3, $1), (I2, I3, $2), (I2, I4, $2) και (I3, I4, $3).

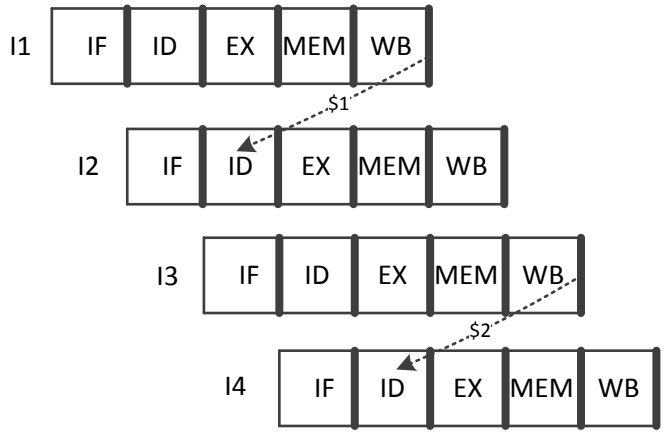
Με τη μονάδα προώθησης, η οποία επιτρέπει προώθηση δεδομένων από τους καταχωρητές EX/MEM και MEM/WB στο στάδιο EX, έχουμε την ακόλουθη εκτέλεση που επιλύει όλα τους κινδύνους.



Σχ. 2

Β. Εργαζόμενοι με τον ίδιο τρόπο, έχουμε τα ακόλουθα.

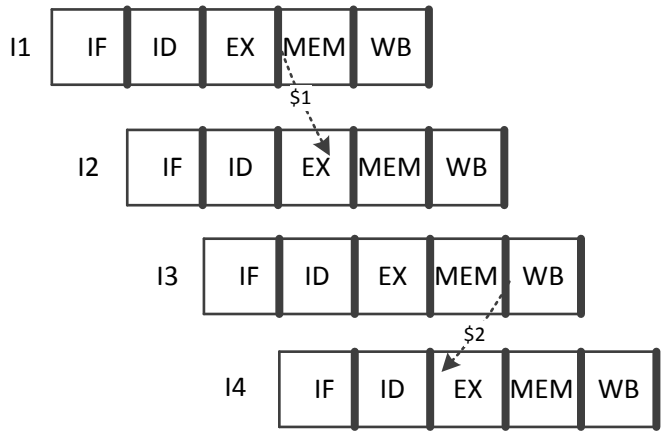
Για την περίπτωση όπου δεν υπάρχει μονάδα προώθησης, η εκτέλεση των εντολών και οι εξαρτήσεις δείχνονται στο ακόλουθο σχήμα:



Σχ. 3

Είναι φανερό ότι υπάρχει κίνδυνος δεδομένων μεταξύ των (I1, I2, $1) και (I3, I4, $2).

Για την περίπτωση όπου υπάρχει μονάδα προώθησης, η εκτέλεση των εντολών και οι εξαρτήσεις δείχνονται στο ακόλουθο σχήμα:



Σχ. 3

Εδώ βλέπουμε ότι κίνδυνος (I1, I2, $1) μπορεί να επιλυθεί με τη μονάδα προώθησης, ενώ ο κίνδυνος (I3, I4, $2) δε μπορεί να επιλυθεί. Για το σκοπό αυτό χρειάζεται η εισαγωγή μίας NOP μεταξύ των I3 και I4.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.