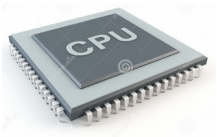
Εθνικό & Καποδιστριακό Πανεπιστημιο ων
Τμημα Πληροφορικης & Τηλεπικοινων
Μαθημα: **Αρχιτεκτονικη Υπολογιστω**



Εργασία 1 (*υποχρεωτική*) – Διοχέτευση

Ακαδημαϊκο Ετος 2019 - 2020
(εκφωνηση) Τεταρτη 20 Νοεμβριου 2019
(παραδοση στο eclass μεχρι) **Τριτη 10 Δεκεμβριου 2019**

Επώνυμο	Όνομα	Αριθμός Μητρώου	Email
Μαλέκας	Παντελεήμων	1115201600268	sdi1600268@di.uoa.gr
Πανταζή	Σωτηρία	1115201700241	sdi1700241@di.uoa.gr

Πληροφορίες για τις Υποχρεωτικές Εργασίες του μαθήματος

- Οι υποχρεωτικές εργασίες του μαθήματος είναι **δύο**. Σκοπός τους είναι η κατανόηση των εννοιών του μαθήματος με χρήση αρχιτεκτονικών προσομοιωτών. Η πρώτη υποχρεωτική εργασία (αυτή) αφορά τη διοχέτευση (pipelining) και η δεύτερη θα αφορά τις κρυφές μνήμες (cache memories).
- Οι δύο αυτές εργασίες είναι υποχρεωτικές και η βαθμολογία του μαθήματος θα προκύπτει από το γραπτό (60%), την εργασία της διοχέτευσης (20%), και την εργασία των κρυφών μνημών (20%). Αυτός είναι ο τρόπος εξέτασης του μαθήματος για όσους φοιτητές έχουν αριθμό μητρώου 2009 και μεταγενέστερο (δηλαδή πήραν το μάθημα για πρώτη φορά το εαρινό εξάμηνο του 2012 και μετά). Καθένας από τους τρεις βαθμούς πρέπει να είναι προβιβάσιμος για να περαστεί προβιβάσιμος βαθμός στη γραμματεία.
- Για τους παλαιότερους φοιτητές (με αριθμό μητρώου 2008 και παλαιότερο) οι δύο εργασίες (pipeline, cache) είναι προαιρετικές. Αν κάποιος παλαιότερος φοιτητής δεν τις παραδώσει θα βαθμολογηθεί με ποσοστό 100% στο γραπτό. Αν τις παραδώσει, θα βαθμολογηθεί με τον παραπάνω τρόπο (γραπτό + 2 εργασίες).
- Κάθε ομάδα μπορεί να αποτελείται **από 1 έως και 3 φοιτητές**. Συμπληρώστε τα στοιχεία όλων των μελών της ομάδας στον παραπάνω πίνακα. Όλα τα μέλη της ομάδας πρέπει να έχουν ισότιμη συμμετοχή και να γνωρίζουν τις λεπτομέρειες της υλοποίησης της ομάδας.
- Για την εξεταστική Σεπτεμβρίου δε θα δοθούν άλλες εργασίες. Το Σεπτέμβριο εξετάζεται μόνο το γραπτό.
- Σε περίπτωση αντιγραφής θα μηδενίζονται όλες οι ομάδες που μετέχουν σε αυτή.
- Η παράδοση της **Εργασίας Διοχέτευσης** πρέπει να γίνει μέχρι τα **μεσάνυχτα της προθεσμίας ηλεκτρονικά** και μόνο στο eclass (να ανεβάσετε ένα μόνο αρχείο zip ή rar με την τεκμηρίωσή σας σε PDF και τον κώδικά σας). **Μην περιμένετε μέχρι την τελευταία στιγμή. Δεν θα υπάρξει παράταση στην προθεσμία παράδοσης ώστε να διατεθεί αρκετός χρόνος και για την εργασία των κρυφών μνημών της οποίας η εκφώνηση θα δοθεί αμέσως μετά**.

Ζητούμενο

Το ζητούμενο της εργασίας είναι να υλοποιήσετε ένα πρόγραμμα σε συμβολική γλώσσα για τον προσομοιωτή WinMIPS64 με σκοπό την εκτέλεσή του **στον μικρότερο δυνατό χρόνο** (μικρότερο αριθμό κύκλων ρολογιού).

Το πρόγραμμα πρέπει να εκτελεί την εξής απλή λειτουργία: υπολογίζει το άθροισμα των 500 στοιχείων ενός πίνακα απρόσημων (unsigned) ακεραίων αριθμών K[i] (i = 0, 1, … 499) που βρίσκονται στην μνήμη (στο τμήμα .data του πηγαίου αρχείου συμβολικής γλώσσας). Οι αριθμοί K[i] έχουν τιμές μεταξύ του 0 και του 127, ο πίνακας δεν είναι ταξινομημένος και οποιαδήποτε τιμή μπορεί να εμφανίζεται οποιονδήποτε αριθμό φορών. Στο τέλος της εκτέλεσης του προγράμματος να εκτυπώνεται στο Terminal το άθροισμα: “Sum=…”.

Μπορείτε να χρησιμοποιήσετε *οποιεσδήποτε* ρυθμίσεις του προσομοιωτή (Enable/Disable Forwarding, Enable/Disable Branch Target Buffer, Enable/Disable Delay Slot) και να γράψετε τον κώδικά σας και τις δομές δεδομένων σας με *όποιον* τρόπο θέλετε – μόνος στόχος είναι να ελαχιστοποιήσετε τον χρόνο εκτέλεσης του προγράμματος. Οι μόνοι περιορισμοί είναι αυτοί της προηγούμενης παραγράφου.

Μεταξύ των προγραμμάτων που εκτελούνται σωστά, τα ταχύτερα (που διαρκούν τον μικρότερο αριθμό κύκλων ρολογιού) θα βαθμολογηθούν με μεγαλύτερο βαθμό.

Εκτός από το πρόγραμμά σας σε συμβολική γλώσσα, το οποίο πρέπει να παραδώσετε σε ξεχωριστό αρχείο, να συμπληρώσετε τον ακόλουθο πίνακα για το πρόγραμμά σας.

	Ενεργοποίηση Forwarding (Ναι / Όχι)	Ενεργοποίηση Branch Target Buffer (Ναι / Όχι)	Ενεργοποίηση Branch Delay Slot (Ναι / Όχι)	Κύκλοι ρολογιού εκτέλεσης
Απαντήσεις για το Πρόγραμμα μου	Ναι	Ναι	Όχι	1578

Τεκμηρίωση

[Σύντομη τεκμηρίωση της λύσης σας μέχρι **5 σελίδες ξεκινώντας από την επόμενη** – μην αλλάζετε τη μορφοποίηση του κειμένου (**και παραδώστε την τεκμηρίωση σε αρχείο PDF**). Η τεκμηρίωσή σας πρέπει να περιλαμβάνει παραδείγματα ορθής εκτέλεσης του προγράμματος και σχολιασμό για την επίλυση του προβλήματος και την επίτευξη μικρότερου χρόνου εκτέλεσης. Μπορείτε να χρησιμοποιήσετε εικόνες, διαγράμματα και ό,τι άλλο μπορεί να βοηθήσει στην εξήγηση της δουλειάς σας.]

Γενική σημείωση: Το Data Address Bus στο παράθυρο Architecture του Winmips έχει αλλάξει από 10 σε 12.

Το πρόγραμμα το οποίο αναπτύχθηκε υπολογίζει το άθροισμα από έναν πίνακα με 500 τυχαίους αριθμούς και εκτυπώνει το ζητούμενο άθροισμα. Ο πίνακας με τους 500 αριθμούς έχει οριστεί στο data τμήμα με όνομα Numbers και αποτελείται από 32 γραμμές και 16 στήλες (με εξαίρεση την τελευταία γραμμή που έχει 4 στήλες) από στοιχεία τύπου byte. Εφόσον οι αριθμοί είναι απρόσημοι στο διάστημα [0,127] θεωρήσαμε κατάλληλη την χρήση τύπου byte. Οι αριθμοί παράχθηκαν τυχαία με την χρήση του random.org.

Στο τμήμα κώδικα υπολογίζεται ο ζητούμενος αριθμός. Πρώτα, χωρίζουμε τον πίνακα Numbers σε 10 υποπίνακες των 50 ακεραίων, τοποθετώντας τους καταχωρητές t1-t9 στις κατάλληλες θέσεις του (ο t0 θε ξεκινήσει από το 0 οπότε δεν χρειάζεται αρχικοποίηση). Ο s0 παίρνει την τιμή 50 (αριθμός επαναλήψεων που πρέπει να γίνουν) για να χρησιμοποιηθεί στην συνθήκη ελέγχου του βρόχου. Επίσης στους καταχώρησες a1 και a2 φορτώνονται οι διευθύνσεις από τα CONTROL και DATA labels αντίστοιχα, που έχουν οριστεί στο data τμήμα του προγράμματος, γιατί θα χρησιμοποιηθούν αργότερα για τις εκτυπώσεις.

Ακολουθεί το loop label στο όποιο υπολογίζεται το άθροισμα των στοιχείων του πίνακα. Αρχικά, φορτώνουμε ένα byte από τον υποπίνακα που καλύπτει ο t0, στον καταχωρητή s1. Αυξάνουμε τον δείκτη t0 κατά 1 και μετά προσθέτουμε στον καταχωρητή s7 το byte που φορτώθηκε στον s1 για να βρούμε το άθροισμα. Αυτά τα 3 βήματα επαναλαμβάνονται αντίστοιχα για τους υπόλοιπους 9 υποπίνακες. Οι καταχωρητές s1-s5 επαναχρησιμοποιούνται για τους υποπίνακες 6-10.

Τέλος εκτελείται η συνθήκη ελέγχου. Για όσο είναι ψευδής, επισρέφουμε στο loop label για την επόμενη επανάληψη και όταν γίνει αληθής σημαίνει ότι έχουν προστεθεί όλα τα στοιχεία του Numbers και προχωράμε στην εκτύπωση των μηνυμάτων. Αρχικά φορτώνουμε από το data τμήμα το string “Sum = “ στον καταχωρητή s2 και το εκτυπώνουμε κάνοντας χρήση των CONTROL και DATA καταχωρητών. Μετά κάνουμε το ίδιο για τον καταχωρητή s7 ο οποίος περιέχει το άθροισμα. Για τον πίνακα που έχουμε ορίσει στο συγκεκριμένο πρόγραμμα έχουμε ως αποτέλεσμα να εκτυπώνεται: Sum = 30617, το οποίο είναι πράγματι το άθροισμα των στοιχείων του πίνακα.

Ακολουθεί εξήγηση για το πώς επιτεύχθηκε ο αριθμός των κύκλων του προγράμματος.

Παρατηρούμε πως χωρίς καμία μέθοδο αντιμετώπισης των pipeline hazards έχουμε 2125 συνολικούς κύκλους, καθώς και 502 RAW stalls και 49 branch taken stalls.

Τα RAW stalls βλέπουμε ότι πραγματοποιήθηκαν στα εξής σημεία: Στο loop, η πρώτη εντολή όπου υπολογίζεται το άθροισμα (συγκεκριμένα η dadd $s7, $s7, $s1) πραγματοποιεί ένα RAW stall

επειδή έχει ως είσοδο τον καταχωρητή s1 και δεν έχει ολοκληρωθεί το WB στάδιο από την προηγούμενη εντολή, lbu $s1, Numbers($t0) (στην οποία φορτώνουμε ένα στοιχείο του πρώτου υποπίνακα). Το ακριβώς ίδιο RAW stall πραγματοποιείται και στις υπόλοιπες dadd εντολές, όπου προσθέτουμε στοιχεία από τους υπόλοιπους υποπίνακες. Επειδή λοιπόν αυτό συμβαίνει σε κάθε στοιχείο που προστίθεται στο άθροισμα, έχουμε 500 RAW stalls σε όλο το loop. Επίσης, έχουμε δυο ακόμα στις ρουτίνες των εκτυπώσεων. Στην ρουτίνα όπου εκτυπώνουμε το string: Sum = , στην εντολή sd $s2, ($a2) έχουμε RAW stall επειδή δεν έχει ολοκληρωθεί το WB στάδιο της εντολής daddi $s2, $zero, sum όπου τοποθετείται η γραμματοσειρά στον καταχωρητή s2. Ομοίως έχουμε παρόμοιο RAW stall στην εντολή sd $v0, ($a1) το οποίο προκαλείται από την εντολή daddi $v0, $zero, 1. Άρα έχουμε συνολικά 502 RAW stalls.

Με την χρήση του forwarding, οι κύκλοι του προγράμματος γίνονται 1623, τα RAW stalls μηδενίζονται και τα branch taken stalls παραμένουν 49. Εφόσον έχει ενεργοποιηθεί το forwarding, οι απαραίτητες τιμές προωθούνται έγκαιρα στον κάθε καταχωρητή και έτσι, δεν χρειάζεται καμία επόμενη εντολή να περιμένει να ολοκληρωθεί το WB στάδιο από κάποια άλλη προηγούμενη. Εφόσον τα RAW stalls τακτοποιήθηκαν, μειώθηκαν και 502 κύκλοι από το πρόγραμμα και έτσι φτάσαμε στους 1623. Το forwarding δεν επηρεάζει με κάποιον τρόπο τις branch εντολές οπότε τα branch taken stalls έμειναν ίδια.

Όσον αφορά τα branch taken stalls βλέπουμε το εξής: Η τελευταία εντολή στο loop είναι η bne $t0, $s0, loop και μετά από αυτήν ακολουθεί η daddi $s2, $zero, sum η οποία είναι η πρώτη από τις ρουτίνες της εκτύπωσης. Όσο λοιπόν οι καταχωρητές t0 και s0 δεν είναι ίσοι, η εντολή bne $t0, $s0, loop μας επιστρέφει στην εντολή όπου φορτώνουμε ένα στοιχείο από τον πρώτο υποπίνακα, δηλαδή την lbu $s1, Numbers($t0) . Όμως λόγω του pipeline, στο IF στάδιο της bne $t0, $s0, loop έχει φορτωθεί στο ID στάδιο η επόμενη εντολή, δηλαδή η daddi $s2, $zero, sum. Εφόσον η bne $t0, $s0, loop μας πηγαίνει πίσω στην πρώτη εντολή του loop, διακόπτεται η εκτέλεση της εντολής daddi $s2, $zero, sum και άρα έχουμε branch taken stall. Επειδή το loop εκτελείται 50 φορές έχουμε 49 branch taken stalls επειδή αυτό συμβαίνει σε κάθε επανάληψη με εξαίρεση την τελευταία γιατί εκεί εκτελείται κανονικά η εντολή daddi $s2, $zero, sum. Άρα καταλήγουμε έτσι με 49 branch taken stalls.

Με την χρήση του Branch Target Buffer παρατηρούμε το εξής: Αρχικά, στην 1η επανάληψη πάλι θα έχει φορτωθεί στο ID στάδιο η εντολή daddi $s2, $zero, sum, αλλά επειδή με την bne $t0, $s0, loop επιστρέφουμε στην πρώτη εντολή της επανάληψης θα δημιουργηθεί πάλι ένα branch taken stall από την διακοπή της εντολής daddi $s2, $zero, sum καθώς και ένα επιπλέον στο ID στάδιο της bne $t0, $s0, loop. Όμως τώρα, το Branch Target Buffer προβλέπει ότι σε κάθε επόμενη σύγκριση οι καταχωρητές δεν θα είναι ίσοι, άρα δεν ξαναφορτώνεται στο ID στάδιο η εντολή daddi $s2, $zero, sum πριν την ώρα της. Όμως στην τελευταία επανάληψη, εφόσον οι καταχωρητές είναι ίσοι, το οποίο σημαίνει οτί θα είναι και η πρόβλεψη του Branch Target Buffer λανθασμένη, θα δημιουργηθεί ένα branch misprediction stall επειδή διακόπτεται η εκτέλεση της εντολής lbu $s1, Numbers($t0), καθώς και ένα ακόμα στο ID στάδιο της bne $t0, $s0, loop εντολής. Όπως έχουμε δει από την θεωρία το Winmips64 μετράει κάθε 1ο stall ως branch taken stall και κάθε 2ο ως branch misprediction stall οπότε ο αριθμός των stalls είναι αναμενόμενος. Άρα έτσι καταλήγουμε με 2 branch taken stalls και 2 branch misprediction stalls. Ωστόσο μιας και μειώθηκαν 47 branch taken stalls βλέπουμε ότι μειώθηκαν και οι κύκλοι του προγράμματος σε 1578 (μαζί με την χρήση forwarding).

Ο λόγος που δεν χρησιμοποιήθηκε το delay slot είναι ο εξής: Στο τέλος της επανάληψης όπου έχουμε την εντολή bne $t0, $s0, loop στο IF στάδιο και στο ID στάδιο την επόμενη εντολή, δηλαδή την daddi $s2, $zero, sum, σε αντίθεση με πριν δεν θα έχουμε stall άλλα με την χρήση του delay slot, αντί να διακοπεί η εκτέλεση της εντολής θα εκτελεστούν κανονικά όλα τα στάδια της. Οπότε μπορεί να γλιτώνουμε όλα τα branch taken stalls αλλά οι κύκλοι του προγράμματος παραμένουν ακριβώς ίδιοι για αυτό δεν είναι επιθυμητή η χρήση του delay slot.

Η αναδιάταξη στον κώδικα δεν χρειάστηκε ιδιαίτερα, αν και προσέξαμε σημεία όπως το εξής κομμάτι κώδικα στο loop:

```
lbu $s1, Numbers($t0)      #loading an element from the array
daddi $t0, $t0, 1          #counter++
dadd $s7, $s7, $s1         # add element to the sum
```

όπου πρώτα αυξάνουμε τον δείκτη του υποπίνακα και μετά προσθέτουμε το νέο ακέραιο στο άθροισμα, για να αποφύγουμε την άμεση χρήση του ίδιου καταχωρητή.

Επιπλέον, ο διαχωρισμός του Numbers σε υποπίνακες μείωσε σημαντικά τους κύκλους εκτέλεσης, εφόσον για 1 επανάληψη (άρα 1 εκτέλεση branch) έχουμε κατευθείαν 10 προσθέσεις στο άθροισμα αντί 1 που είχαμε στην αρχή.

Καταλήγουμε λοιπόν ότι με χρήση Forwarding και Branch Target Buffer έχουμε τον καλύτερο αριθμό κύκλων στο πρόγραμμα μας. Συγκεκριμένα έχουμε 1578 κύκλους, 2 branch taken stalls και 2 branch misprediction stalls.