Εθνικο & Καποδιστριακο Πανεπιστημιο Αθηνων
Τμημα Πληροφορικης & Τηλεπικοινωνιων
Μαθημα: **Αρχιτεκτονικη Υπολογιστων II**


# Εργασία 1 (*υποχρεωτική*) – Διοχέτευση

**Ακαδημαϊκο Ετος 2016 – 2017**
(εκφωνηση) Τεταρτη 23 Νοεμβριου 2016
(παραδοση στο eclass μεχρι) **Τεταρτη 14 Δεκεμβριου 2016**

| Επώνυμο | Όνομα | Αριθμός Μητρώου | Email |
|---|---|---|---|
| Αθανασόπουλος | Γεώργιος | 1115201300002 | sdi1300002@di.uoa.gr |
| Καρατσενίδης | Κωνσταντίνος | 1115201300064 | sdi1300064@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 Word ή PDF και τους κώδικές 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 και να επιλέξετε τη βέλτιστη διαμόρφωση της CPU για την εκτέλεσή του στον μικρότερο δυνατό χρόνο.

Το πρόγραμμα πρέπει να εκτελεί την εξής λειουργία: αναζητά σε έναν μη ταξινομημένο πίνακα 200 προσημασμένων ακεραίων αριθμών A[i] (i = 0, 1, … 199) ένα συγκεκριμένο ακέραιο X και να μετράει το πλήθος K των εμφανίσεών του. Το πρόγραμμα πρέπει επίσης να μετράει πόσοι από τους αριθμούς του πίνακα είναι αρνητικοί (N), πόσοι θετικοί (P), πόσοι μηδέν (Z), πόσοι είναι μεγαλύτεροι του ζητούμενου αριθμού (B) και πόσοι είναι μικρότεροί του (S). Τα A, X, K, N, P, Z, B, S πρέπει να είναι μεταβλητές στο τμήμα δεδομένων του προγράμματός σας. Στο τέλος της εκτέλεσης του προγράμματος να εκτυπώνονται στο Terminal οι τιμές των αποτελεσμάτων K, N, P, Z, B, S σε μία γραμμή: "K=..., N=..., P=..., Z=…, B=…, S=…"

Μπορείτε να κάνετε αλλαγές στο Configuration του προσομοιωτή (Enable Forwarding, Enable Branch Target Buffer, Enable Delay Slot) και αλλαγές στον κώδικα ώστε να μειώσετε τις καθυστερήσεις (stalls) που αυξάνουν το χρόνο εκτέλεσης του προγράμματος.

Θα πρέπει να υποθέσετε ότι ο ρυθμός του ρολογιού του μικροεπεξεργαστή είναι διαφορετικός ανάλογα με την διαμόρφωση που θα επιλέξετε. Ο πίνακας που ακολουθεί δίνει το ρυθμό του ρολογιού σε συνάρτηση με την κάθε διαμόρφωση.

| Forwarding | Branch Target Buffer | Branch Delay Slot | Ρυθμός ρολογιού |
|---|---|---|---|
| – | – | – | 500 MHz |
| – | – | ✓ | 485 MHz |
| – | ✓ | – | 470 MHz |
| ✓ | – | – | 445 MHz |
| ✓ | – | ✓ | 430 MHz |
| ✓ | ✓ | – | 415 MHz |

Μεταξύ των προγραμμάτων που εκτελούνται σωστά, τα ταχύτερα θα βαθμολογηθούν με μεγαλύτερο βαθμό. Συμπληρώσετε τον ακόλουθο πίνακα για το πρόγραμμά σας.

| | Επιλογή Forwarding (Ναι / Όχι) | Επιλογή Branch Target Buffer (Ναι / Όχι) | Επιλογή Branch Delay Slot (Ναι / Όχι) | Περίοδος ρολογιού (nsec) | Κύκλοι ρολογιού εκτέλεσης | Χρόνος εκτέλεσης (sec) |
|---|---|---|---|---|---|---|
| Απαντήσεις για το Πρόγραμμα μου | Όχι | Όχι | Όχι | 2 nsec | 1585 | 3,17 * 10^-6 sec |

**Τεκμηρίωση**

Σκοπός της παρούσας εργασίας ήταν να μειώσουμε όσο το δυνατόν περισσότερο τον χρόνο εκτέλεσης του συγκεκριμένου προβλήματος. Η διαδικασία που ακολουθήθηκε είναι η εξής:
Σε πρώτη φάση γράφτηκε ο κώδικας έτσι ώστε το μόνο που πετυχαίνει να είναι ο σωστός υπολογισμός των αποτελεσμάτων χωρίς κάποια διαμόρφωση στον επεξεργαστή.

**Περιγραφή του Αλγορίθμου**

Στην πορεία των πειραματισμών μας υλοποιήθηκαν 2 αλγόριθμοι. Ο πρώτος διάβαζε τα δεδομένα από το data κομμάτι και στην συνέχεια για κάθε στοιχείο του πίνακα αύξανε τους counter των θετικών, μηδενικών και μεγαλύτερων, ίσων στοιχείων του X. Για τα αρνητικά και τα μικρότερα του X κάναμε την αφαίρεση 200 - θετικά – μηδενικά και 200 – μεγαλύτερα – ίσα αντίστοιχα. Με αυτό τον τρόπο δεν αυξάναμε όλους τους μετρητές μέσα στην λούπα και άρα είχαμε λιγότερες πράξεις προς εκτέλεση.

Η δεύτερη και τελική υλοποίηση εκμεταλλεύεται το γεγονός ότι όταν το X που αναζητούμε είναι μεγαλύτερο του 0 τότε μετρώντας τα N μετράμε ταυτόχρονα και ένα μέρος των S.
Επομένως ο αλγόριθμος ελέγχει αρχικά αν το X>0.
Αν είναι τότε όταν το τρέχων στοιχείο A είναι αρνητικό ή αν είναι μηδέν αυξάνεται μόνο ο μετρητής N ή Z αντίστοιχα.
Αλλιώς όταν A>0, μόνο τότε ελέγχετε αν το A>X ή A==X και έτσι καλύπτουμε το πλήθος των μικρότερων που ξεφεύγουν από το A<0, δηλαδή το διάστημα (0,X).
Αντίστοιχα όταν X<=0, όταν το τρέχων στοιχείο A είναι θετικό ή αν είναι μηδέν αυξάνεται μόνο ο μετρητής P ή Z αντίστοιχα.
Αλλιώς όταν A<0, μόνο τότε ελέγχετε αν το A>X ή A==X και έτσι καλύπτουμε το πλήθος των μεγαλύτερων που ξεφεύγουν από το A>0, δηλαδή το διάστημα (X,0).
Όταν X==0 τότε η λύση είναι τετριμμένη στην δεύτερης περίπτωση όπου X<=0.
Όταν τελειώσει η λούπα κάνουμε προσθαφαιρέσεις για να βρούμε τα αποτελέσματα που μας λείπουν όπως στον πρώτο αλγόριθμο.

Ο δεύτερος αλγόριθμος υπερισχύει έναντι του πρώτου γιατί εκτελεί λιγότερους ελέγχους και προσθέσεις στους μετρητές. Αυτό γίνεται επειδή δεν ελέγχουμε και δεν προσθέτουμε θετικούς ή αρνητικούς όταν X>0 ή X<0 αντίστοιχα.
Επειδή όμως ο δεύτερος αλγόριθμος αποτελείται από δυο ξεχωριστές περιπτώσεις, είναι διπλάσιος σε μέγεθος.
Οι χρόνοι σε αυτή την φάση ήταν:
Αλγόριθμος 1: 2002/(500*10^6) = 4004 nsec
Αλγόριθμος 2: 1693/(500*10^6) = 3386 nsec

```
Execution

2002 Cycles
1559 Instructions
1.284 Cycles Per Instruction (CPI)

Stalls

0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
439 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size

464 Bytes
```

```
Execution

1693 Cycles
1299 Instructions
1.303 Cycles Per Instruction (CPI)

Stalls

0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
390 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size

836 Bytes
```

**Forwarding - RAW Stalls**

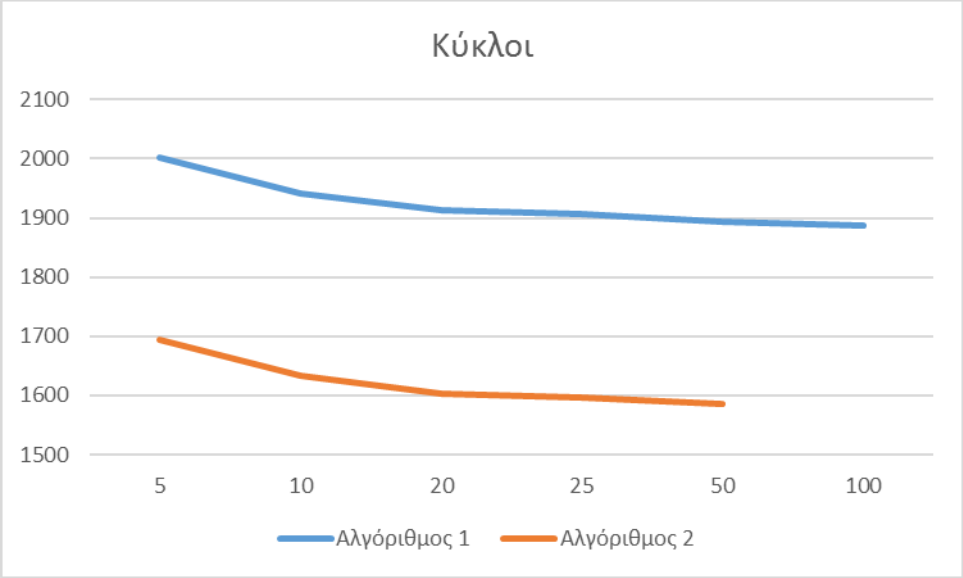
Η παραπάνω υλοποίηση περιείχε πολλά RAW και Branch stalls. Επομένως ο επόμενος στόχος ήταν να μειώσουμε αυτά τα stalls.
Για την μείωση των RAW stalls, αλλάξαμε θέση στις εντολές οι οποίες αλληλοσυγκρούονταν, όπως για παράδειγμα οι φορτώσεις δεδομένων των οποίων οι τιμές μετά χρησιμοποιούνταν στον έλεγχο των συνθηκών. Οι φορτώσεις δεδομένων γίνονται μαζί έτσι ώστε οι επόμενες δύο εντολές κάθε φόρτωσης να μην χρησιμοποιούν την τιμή που φέρνεται εκείνη την στιγμή από την μνήμη. Με αυτή την μέθοδο εξαλείφθηκαν όλα τα RAW stalls μιας και η φύση του προβλήματος ήταν τέτοια που δεν περιελάβανε πολλές εξαρτήσεις δεδομένων. Επίσης με 0 RAW stalls, η επιλογή Forwarding ήταν αχρείαστη επειδή όλα τα δεδομένα, βρίσκονται στους καταχωρητές όταν ζητούνται. Το ότι δεν χρησιμοποιούμε Forwarding έχει το πλεονέκτημα ότι ο ρυθμός του ρολογιού μας είναι μεγαλύτερος.

**Branch Target Buffer - Branch Stalls**

Όσο αναφορά τα Branch stalls η πρώτη προσέγγιση που δοκιμάσαμε για την μείωση τους ήταν η χρήση Branch Target Buffer. Η μέθοδος αυτή μας μείωσε κατά πολύ τους κύκλους όταν ο πίνακας δεδομένων ήταν ταξινομημένος γιατί οι predictors έκαναν σωστές προβλέψεις μιας και στις πρώτες επαναλήψεις η ροή εκτέλεσης πήγαινε στις branch για τα <X και <0 στοιχεία και στην συνέχεια στις =X και =0 . Αστοχίες είχαμε μόνο στο σημείο που τα δεδομένα από μικρότερα έγιναν ίσα με X και 0. Μετά από αυτό το σημείο ξανά υπολογίστηκαν οι προβλέψεις που στην συνέχεια ήταν σωστές. Σύμφωνα με την εκφώνηση της εργασίας όμως, τα δεδομένα που μας δίνονται δεν είναι ταξινομημένα. Στην περίπτωση των μη ταξινομημένων δεδομένων η χρήση του Branch Target Buffer μείωνε ελάχιστα τους κύκλους εκτέλεσης, πράγμα που σε αντιδιαστολή με την μείωση του ρυθμού του ρολογιού, δεν μείωνε τον χρόνο εκτέλεσης. Η σκέψη για ταξινόμηση των δεδομένων πριν τον υπολογισμό των τιμών δεν συμφέρει γιατί μια ταξινόμηση απαιτεί πολύ χρόνο και σε σχέση με τον χρόνο μιας αναζήτησης

**Loop Unrolling**

Μια ιδέα για την εκτέλεση λιγότερων branch αλλά και των μετακινήσεων στον πίνακα ήταν η προσθήκη Loop Unrolling. Επειδή το μέγεθος του πίνακα είναι σταθερό (200 στοιχεία) σύμφωνα με την εκφώνηση και μας ενδιαφέρει περισσότερο η ταχύτητα από ότι η γενικότητα του προγράμματος, αυτή η μέθοδος ευνοείτε. Έτσι κάναμε το μέγιστο αριθμό (100 για τον πρώτο αλγόριθμο και 50 για τον τελικό) Loop Unrolls που μας επέτρεπε το bus των εντολών του επεξεργαστή (13 μέγιστο). Με αυτό τον τρόπο μειώσαμε τον αριθμό των branches (από 200 σε 2 και 4 αντίστοιχα) με αποτέλεσμα να μειωθούν και τα Branch stalls και το πλήθος των εντολών που εκτελούνται, παράγοντες που επηρεάζουν την ταχύτητα.



Οι χρόνοι σε αυτή την φάση ήταν:
Αλγόριθμος 1: 1888/(500*10^6) = 3776 nsec
Αλγόριθμος 2: 1585/(500*10^6) = 3170 nsec

```
Execution
1888 Cycles
1483 Instructions
1.273 Cycles Per Instruction (CPI)

Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
401 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
5404 Bytes
```

```
Execution
1585 Cycles
1227 Instructions
1.292 Cycles Per Instruction (CPI)

Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
354 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
5516 Bytes
```

**Delay Slot - Branch Stalls**

Αρχικά βάλαμε εντολές nop κάτω από κάθε εντολή branch και jump. Στο δεδομένο πρόβλημα το ποσοστό των εντολών ελέγχου, είναι μεγάλο. (Γίνονται έλεγχοι για A>0,A=0,A<0,A>X,A=X,A<X). Όμως, ο αριθμός των υπολοίπων εντολών είναι εξαιρετικά μικρός και είναι στρατηγικά τοποθετημένες ώστε να μην υπάρχουν RAW Stalls. Συνεπώς, δεν υπάρχουν αρκετές εντολές που μπορούν να αντικαταστήσουν τις εντολές nop. Συγκεκριμένα, υπολογίσαμε ότι πρέπει να μειωθούν κατά 250 οι κύκλοι του προγράμματος (1500 δλδ) ώστε να κάνει ίδιους χρόνους αυτός ο επεξεργαστείς, αλλά οι κύκλοι που γλυτώναμε ήταν οριακά 200.

**Συμπεράσματα**

Τα συμπεράσματα στα οποία καταλήξαμε μετά από την υλοποίηση αυτής της εργασίας είναι ότι για την μείωση του χρόνου εκτέλεσης κάθε προβλήματος πρέπει να σταθείς στις ιδιαιτερότητες του και όχι να το προσεγγίσεις με μία γενική λύση. Έτσι μειώνονται όσον το δυνατόν περισσότερο οι κύκλοι. Επίσης η προσέγγιση που ξεκινάς με ένας επεξεργαστή χωρίς καμία επιπλέον ρύθμιση και προσπαθείς να καλύψεις τα Stalls με αναδιατάξεις του κώδικα είναι προτιμητέα επειδή αποφασίζεις να χρησιμοποιήσεις μόνο όσες ρυθμίσεις είναι χρήσιμες και έτσι έχει τον μεγαλύτερο δυνατό ρυθμό ρολογιού.