ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

1η ΑΣΚΗΣΗ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2023-2024, 8ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Τελική Ημερομηνία Παράδοσης: **20/04/2024**

1. Εισαγωγή

Στα πλαίσια της παρούσας άσκησης θα χρησιμοποιήσετε το εργαλείο *"PIN"* για να μελετήσετε την επίδραση διαφόρων παραμέτρων της ιεραρχίας μνήμης στην απόδοση ενός συνόλου εφαρμογών. Στόχος της άσκησης είναι η εξοικείωση με το εργαλείο PIN καθώς και με την διαδικασία διεξαγωγής πειραματικών μετρήσεων με σκοπό την εξαγωγή χρήσιμων συμπερασμάτων.

2. Το εργαλείο PIN

Το PIN είναι ένα εργαλείο το οποίο αναπτύσσεται και συντηρείται από την Intel και χρησιμοποιείται για την ανάλυση εφαρμογών. Χρησιμοποιείται για dynamic binary instrumentation, δηλαδή για την εισαγωγή κώδικα δυναμικά (την στιγμή που εκτελείται η εφαρμογή) ανάμεσα στις εντολές της εφαρμογής με σκοπό την συλλογή πληροφοριών σχετικά με την εκτέλεση (π.χ. cache misses, total instructions κλπ.).

Περισσότερες πληροφορίες σχετικά με το PIN καθώς και εγχειρίδια χρήσης μπορείτε να βρείτε εδώ:

https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

3. Λήψη και εγκατάσταση του PIN

Για να εγκαταστήσετε το PIN στο σύστημα σας κατεβάστε το από το παρακάτω link:

https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-binary-instrumentation-tool-downloads.html

Το PIN εξαρτάται άμεσα από τον πυρήνα του λειτουργικού συστήματος, οπότε υπάρχει πιθανότητα κάποιες εκδόσεις του PIN να έχουν ασυμβατότητα με συγκεκριμένες εκδόσεις του πυρήνα Linux. Τα βήματα που παρουσιάζονται παρακάτω για την εκτέλεση του PIN έχουν δοκιμαστεί επιτυχώς χρησιμοποιώντας την πιο πρόσφατη του έκδοση (3.30) σε Ubuntu 22.04 (με έκδοση πυρήνα 6.5).

Αφού ολοκληρωθεί η λήψη του αρχείου θα πρέπει να το αποσυμπιέσετε δίνοντας σε ένα τερματικό την παρακάτω εντολή:

```
$ tar xvfz pin-3.30-98830-g1d7b601b3-gcc-linux.tar.gz
```

Τώρα μπορείτε να περιηγηθείτε στα περιεχόμενα του PIN:

```
$ cd pin-3.30-98830-g1d7b601b3-gcc-linux
$ ls -aF

./ ../ README doc/ extras/ ia32/ intel64/ licensing/ pin pin.sig source/
```

Το **pin** είναι το εκτελέσιμο που θα χρησιμοποιήσετε για την εκτέλεση των εφαρμογών στα πειράματά σας. Για να δείτε τον τρόπο χρήσης του pin.sh μπορείτε να το εκτελέσετε χωρίς ορίσματα:

```
$ ./pin

E: Missing application name
Pin: pin-3.30-98830-g1d7b601b3
Copyright 2002-2020 Intel Corporation.

Usage: pin [OPTION] [-t <tool> [<toolargs>]] -- <command line>
Use -help for a description of options
```

Με το όρισμα **-t** λέτε στο PIN ποιο pintool να χρησιμοποιήσει, ενώ ώς **<command line>** δίνεται το εκτελέσιμο το οποίο θα αναλυθεί από το PIN καθώς και τα ορίσματά του. Ένα παράδειγμα εκτέλεσης του pin δίνεται παρακάτω:

```
$ cd source/tools/ManualExamples/
$ make obj-intel64/inscount0.so
$ cd ../../../
$ ./pin -t ./source/tools/ManualExamples/obj-intel64/inscount0.so \
                                -o ls.inscount0.output -- /bin/ls -aF
./    ../    README*    doc/    extras/    ia32/    intel64/    licensing/
ls.inscount0.output  pin*  pin.log  pin.sig*  source/
$ cat ls.inscount0.output
Count 455956
```

Στο παραπάνω παράδειγμα χρησιμοποιήθηκε το pintool **inscount0.so** το οποίο αθροίζει το σύνολο των εντολών που εκτελούνται. Τα pintools είναι προγράμματα γραμμένα σε C++ που χρησιμοποιούνται από το PIN και επικοινωνούν με αυτό μέσω του API του για να κατευθύνουν την ανάλυση των εφαρμογών. Μπορείτε να γράψετε τα δικά σας pintools αλλά υπάρχουν και αρκετά που παρέχονται μαζί με το PIN. Θα τα βρείτε στον φάκελο **pin-3.30-98830-g1d7b601b3-gcc-linux/source/tools/**. Για να τα μεταγλωττίσετε μπορείτε να εκτελέσετε την εντολή make στον φάκελο που σας ενδιαφέρει.

4. Μετροπρογράμματα

Το PIN μπορεί να χρησιμοποιηθεί για την εκτέλεση οποιασδήποτε εφαρμογής. Στα πλαίσια της παρούσας άσκησης θα χρησιμοποιήσετε τα PARSEC benchmarks για τα οποία μπορείτε να βρείτε περισσότερες πληροφορίες εδώ:

https://www.cs.princeton.edu/techreports/2008/811.pdf

Κατεβάστε την έκδοση 3.0 της σουίτας από το παρακάτω link:

https://github.com/cirosantilli/parsec-benchmark/releases/download/3.0/parsec-3.0-core.tar.gz

Επίσης, θα χρειαστείτε τα αρχεία εισόδου για τα benchmarks τα οποία μπορείτε να κατεβάσετε από εδώ:

https://github.com/cirosantilli/parsec-benchmark/releases/download/3.0/parsec-3.0-input-sim.tar.gz

Στη συνέχεια αποσυμπιέστε τα ληφθέντα αρχεία:

```
$ tar xvfz parsec-3.0-core-tar.gz
$ tar xvfz parsec-3.0-input-sim.tar.gz
```

Η σουίτα περιλαμβάνει 13 benchmarks και διάφορα αρχεία εισόδου. Για τους σκοπούς της άσκησης θα χρησιμοποιήσετε τα παρακάτω 8 benchmarks με τα simlarge αρχεία εισόδου:

1. blackscholes
2. bodytrack
3. canneal
4. fluidanimate
5. freqmine
6. raytrace
7. swaptions
8. streamcluster

Στον κώδικα κάθε benchmark έχουν οριστεί οι περιοχές του κώδικα που παρουσιάζουν το μεγαλύτερο ενδιαφέρον (Regions of Interest, ROI). Τα ROI ορίζονται από τις κλήσεις των συναρτήσεων **__parsec_roi_begin()** και **__parsec_roi_end()**.

Για την διαχείριση (μεταγλώττιση, εκτέλεση κ.λ.π.) των PARSEC benchmarks παρέχεται το script bin/parsecmgmt του οποίου οι σημαντικότερες παράμετροι είναι:

- -a action : η ενέργεια που θέλουμε να γίνει, π.χ. build, run, status
- -p benchmark : το όνομα του benchmark που θέλουμε να μεταγλωττίσουμε ή να εκτελέσουμε
- -c config-file : το αρχείο ρυθμίσεων που θα χρησιμοποιηθεί για την μεταγλώττιση. Παρέχονται μεταξύ άλλων:
  - gcc-serial: μεταγλώττιση σειριακών benchmarks
  - gcc-pthreads: μεταγλώττιση παράλληλων benchmarks με χρήση των pthreads
  - gcc-hooks: μεταλώττιση παράλληλων benchmarks με χρήση pthreads και hooks που ορίζουν τα ROI

Επειδή δεν παρέχεται κάποιο config αρχείο που να μεταγλωττίζει τη σειριακή έκδοση των benchmarks με ταυτόχρονη χρήση των hooks για τα ROI θα χρειαστεί να τροποποιηθούν τα config αρχεία της έκδοσης gcc-serial. Το script **cslab_process_parsec_benchmarks.sh** που παρέχεται με τον βοηθητικό κώδικα της άσκησης κάνει όλες τις κατάλληλες τροποποιήσεις και θα πρέπει να το εκτελέσετε μέσα στον φάκελο parsec-3.0.

Για να μεταγλωττίσετε τα benchmarks που θα χρησιμοποιηθούν στο πειραματικό σκέλος της άσκησης εκτελέστε τις παρακάτω εντολές:

```
$ ~/advcomparch-ex1-helpcode/cslab_process_parsec_benchmarks.sh
$ sudo apt-get update
$ sudo apt-get install make g++ libx11-dev libxext-dev libxaw7 \
x11proto-xext-dev libglu1-mesa-dev libxi-dev libxmu-dev
$ ./bin/parsecmgmt –a build –c gcc-serial –p blackscholes bodytrack canneal
fluidanimate freqmine raytrace streamcluster swaptions
```

Στη συνέχεια εκτελέστε το script **cslab_create_parsec_workspace.sh** το οποίο θα δημιουργήσει έναν φάκελο *parsec_workspace* που θα περιέχει όλα τα εκτελέσιμα που θα χρειαστείτε καθώς και όλα τα αρχεία εισόδου για τα benchmarks. Τέλος, στον βοηθητικό κώδικα σας δίνεται το αρχείο **cmds_simlarge.txt** που περιέχει τις εντολές για την εκτέλεση κάθε benchmark. Έχουμε επιλέξει να μην χρησιμοποιηθεί το script parsecgmgt για την εκτέλεση των benchmarks, καθώς με τη χρήση του

δημιουργούνται επιπλέον διεργασίες κατά την εκτέλεση των benchmarks, γεγονός που μπορεί να επηρεάσει τις μετρήσεις σας.

```
$ ~/advcomparch-ex1-helpcode/cslab_create_parsec_workspace.sh
$ cd parsec_workspace
$ cp ~/advcomparch-ex1-helpcode/cmds_simlarge.txt .
$ cat cmds_simlarge.txt
./executables/blackscholes 1 inputs/in_64K.txt prices.txt
./executables/bodytrack inputs/sequenceB_4 4 4 4000 5 0 1
./executables/canneal 1 15000 2000 inputs/400000.nets 128
./executables/fluidanimate 1 5 inputs/in_300K.fluid out.fluid
./executables/freqmine inputs/kosarak_990k.dat 790
./executables/rtview inputs/happy_buddha.obj -automove -nthreads 1 -frames
3 -res 1920 1080
./executables/streamcluster 10 20 128 16384 16384 1000 none output.txt 1
./executables/swaptions -ns 64 -sm 40000 -nt 1
```

Πριν την εκτέλεση οποιουδήποτε benchmark *θα πρέπει* να έχετε ορίσει την μεταβλητή περιβάλλοντος **LD_LIBRARY_PATH** ώστε να δείχνει στο PATH που περιέχει την βιβλιοθήκη των hooks:

```
$ export LD_LIBRARY_PATH=~/parsec-3.0/pkgs/libs/hooks/inst/amd64-
linux.gcc-serial/lib
```

5. PINTOOL: simulator

Στον βοηθητικό κώδικα της άσκησης θα βρείτε το pintool **simulator.cpp**, το οποίο θα χρησιμοποιήσετε για την προσομοίωση της εκτέλεσης εφαρμογών σε ένα περιβάλλον με έναν in-order επεξεργαστή, κρυφή μνήμη δύο επιπέδων (L1-Data + L2 caches) και μνήμη μετάφρασης διευθύνσεων (TLB). Το simulator.cpp είναι γραμμένο έτσι ώστε να ενεργοποιείται μόνο κατά την διάρκεια των PARSEC ROI. Για την μεταγλώττισή του, κατευθυνθείτε στο directory του pintool που σας παρέχεται:

```
$ cd advcomparch-ex1-helpcode/pintool
```

Τροποποιήστε κατάλληλα το αρχείο makefile ώστε το PIN_ROOT να δείχνει στο path του pin, δηλαδή:

PIN_ROOT ?= /path/to/pin-3.30-98830-g1d7b601b3-gcc-linux

Και εκτελέστε:

```
$ make clean; make
```

Το simulator pintool δέχεται τα παρακάτω ορίσματα:

- -o <filename> : το αρχείο εξόδου όπου θα αποθηκευτούν οι παράμετροι και τα στατιστικά της προσομοίωσης
- -L1c : το μέγεθος της L1 (KB)
- -L1a : το associativity της L1
- -L1b : το μέγεθος του block της L1 (bytes)
- -L2c : το μέγεθος της L2 (KB)
- -L2a : το associativity της L2

- -L2b : το μέγεθος του block της L2 (bytes)
- -L2prf : ο αριθμός των blocks που γίνονται prefetched στην L2 (η default τιμή 0 απενεργοποιεί το prefetching)
- -TLBe : το μέγεθος σε καταχωρίσεις (entries) του TLB
- -TLBa : το associativity του TLB
- -TLBp : το μέγεθος της σελίδας μνήμης (pagesize) που χρησιμοποιεί το TLB (bytes)

Με τις κατάλληλες τροποποιήσεις, το simulator pintool μπορεί επίσης να προσομοιώσει διαφορετικές στρατηγικές ως προς το write allocation ή διαφορετικές πολιτικές αντικατάστασης στις caches και στο TLB.

Παράδειγμα χρήσης του simulator pintool:

```
$ /path/to/pin-3.30-98830-g1d7b601b3-gcc-linux/pin \
 -t /path/to/advcomparch-ex1-helpcode/pintool/obj-intel64/simulator.so \
-o my_output.out -L1c 64 -L1a 8 -L1b 64 -L2c 256 -L2a 8 -L2b 64 \
  -TLBe 64 -TLBa 4 -TLBp 4096 -- \
 /path/to/parsec_workspace/executables/blackscholes 1 \
 /path/to/parsec_workspace/inputs/in_64K.txt prices.txt
```

6. Προσομοίωση Ιεραρχίας μνήμης

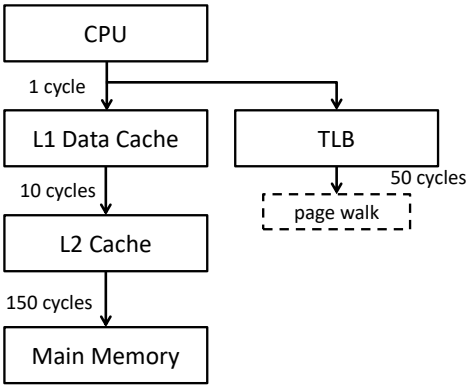
Η ιεραρχία κρυφής μνήμης που θα χρησιμοποιήσουμε στα πλαίσια αυτής της άσκησης απεικονίζεται στο παρακάτω σχήμα. Πιο συγκεκριμένα, το simulator pintool προσομοιώνει έναν in-order επεξεργαστή με την inclusive ιεραρχία μνήμης και μετάφραση διευθύνσεων που φαίνεται στο παραπάνω σχήμα. Για τον υπολογισμό της επίδοσης των εφαρμογών που χρησιμοποιούνται στις προσομοιώσεις, χρησιμοποιείται ένα απλό μοντέλο, όπου θεωρούμε ότι κάθε εντολή απαιτεί 1 κύκλο για την εκτέλεσή της (IPC=1).

Επιπρόσθετα, οι εντολές που πραγματοποιούν πρόσβαση στη μνήμη (είτε load είτε store) προκαλούν επιπλέον καθυστερήσεις ανάλογα με το αν οι μεταφράσεις διευθύνσεων βρίσκονται στο TLB και με το πού βρίσκονται τα δεδομένα τους. Θεωρούμε ότι η L1 είναι υλοποιημένη ως **Virtually indexed, Physically tagged (VIPT)** cache, δηλαδή οι προσβάσεις στο TLB και στην L1 cache επικαλύπτονται και πραγματοποιούνται παράλληλα. Αν η ζητούμενη μετάφραση δεν βρίσκεται στο TLB (TLB miss), τότε εισάγεται μία καθυστέρηση 100 κύκλων που προσομοιώνει την καθυστέρηση ανάκλησης της μετάφρασης διεύθυνσης από την μνήμη (page walk), και στη συνέχεια εκτελείται η πρόσβαση στην ιεραρχία μνήμης μέσω της L1 cache.

Το TLB module του simulator χρησιμοποιείται μόνο για την μελέτη της επίδρασης της μετάφρασης διευθύνσεων στην επίδοση των προγραμμάτων όσον αφορά τον χρόνο εκτέλεσης, και όχι για να παρέχει πραγματική μετάφραση διευθύνσεων στον φυσικό χώρο. Δηλαδή, κατά την προσομοίωση το cache module χρησιμοποιεί εικονικές διευθύνσεις για την πρόσβαση στην μνήμη δεδομένων και τον έλεγχο των cache hits/misses.

Συνεπώς, όπως μπορείτε να διακρίνετε και στο παρακάτω σχήμα έχουμε τις εξής περιπτώσεις:

1. TLB hit: 0 cycles (η πρόσβαση πραγματοποιείται παράλληλα με την L1 cache)
2. TLB miss: 50 cycles
3. L1 hit: 1 cycle
4. L2 hit: 10 cycles
5. Main memory access: 150 cycles



Οι παραπάνω τιμές κύκλων μπορούν να δοθούν και ως παράμετροι κατά την αρχικοποίηση της ιεραρχίας μνήμης στο simulator pintool.

Συνολικά, ο αριθμός των κύκλων υπολογίζεται ως:

Cycles = Inst + TLB_Misses*TLB_miss_cycles + L1_Accesses * L1_hit_cycles + L2_Accesses * L2_hit_cycles + Mem_Accesses * Mem_acc_cycles

7. Πειραματική Αξιολόγηση

Στα πλαίσια της εργασίας αυτής, θα διερευνηθεί η επίδραση των βασικότερων παραμέτρων ιεραρχίας κρυφής μνήμης στην επίδοση της εφαρμογής.

7.1 L1 cache

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, οι παράμετροι της L2 cache και του TLB θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με:

- L2 size = 2048 KB
- L2 associativity = 16
- L2 block size = 256 B
- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

Εκτελέστε τα benchmarks για τις παρακάτω L1 caches:

L1 size (KB)	**L1 associativity**	**L1 cache block size (B)**
32	4, 8	32, 64, 128
64	4	64
64, 128	8	32, 64, 128

7.1.2 L2 cache

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, χρησιμοποιήστε τις παραμέτρους της καλύτερης L1 cache που βρήκατε στο ερώτημα 7.1 (την cache με το υψηλότερο IPC). Για το TLB χρησιμοποιήστε:

- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

Εκτελέστε τα benchmarks για τις παρακάτω L2 caches:

L2 size (KB)	**L2 associativity**	**L2 cache block size (B)**
512	4	128, 256
512	8	64, 128, 256
1024	8, 16	64, 128, 256
2048	16	64, 128, 256

7.1.3 TLB

Για όλες τις περιπτώσεις που εξετάζονται στο πείραμα αυτό, οι παράμετροι της L1 και L2 cache θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με τις βέλτιστες παραμέτρους από τα 7.1 και 7.2 αντίστοιχα.

Εκτελέστε τα benchmarks για τα παρακάτω TLB (υποθέτοντας πολιτική αντικατάστασης LRU για associativity > 1):

TLB size (entries)	**TLB associativity**	**TLB page size (B)**
32	4, 8	4096
64	1, 2, 4, 8, 16, 32, 64	4096
128, 256	4	4096

7.1.4 Προανάκληση (prefetching)

Σε όλες τις προηγούμενες προσομοιώσεις, η προανάκληση (prefetching) είναι απενεργοποιημένη. Επεκτείνετε κατάλληλα τον κώδικα ώστε το σύστημα να προσομοιώνει NEXT-N-LINE prefetching στη L2 cache. Πιο συγκεκριμένα, για κάθε L2 miss θα πρέπει η cache να φέρνει εκτός από το ζητούμενο block και τα επόμενα N blocks, όπου N ο αριθμός που ορίζεται από την παράμετρο L2 prf.

Εκτελέστε τα benchmarks *blackscholes, canneal, fluidanimate* και *rtview* για N = 1, 2, 4, 8, 16, 32, 64. Οι παράμετροι των L1, L2, TLB θα διατηρηθούν σταθερές και συγκεκριμένα ίσες με τις βέλτιστες τιμές που βρήκατε στα ερωτήματα 7.1 και 7.2:

- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

7.1.5 Πολιτικές Αντικατάστασης (replacement policies)

Σε όλες τις προηγούμενες προσομοιώσεις, η ιεραρχία μνήμης χρησιμοποιούσε την πολιτική αντικατάστασης Least Recently Used (LRU). Τροποποιήστε/επεκτείνετε κατάλληλα τον κώδικα που σας έχει δοθεί, ώστε το σύστημα να προσομοιώνει πολιτική αντικατάστασης **στις L1 και L2 caches** (όχι στο TLB) που επιλέγει τυχαία το block που θα αντικατασταθεί.
Ως seed για την συνάρτηση τυχαιότητας που θα χρησιμοποιήσετε στον κώδικά σας χρησιμοποιήστε τα 5 τελευταία νούμερα του ΑΜ σας. Για τη συνάρτηση rand() αυτό θα γινόταν με τις δύο εξής εντολές:

```
srand(10242) // Ο ΑΜ μου τελειώνει σε 10242.
int my_random_number = rand();
```

Τροποποιήστε την κλάση LRU που αφορά τις caches L1 και L2 με τον δικό σας κώδικα. Εκτελέστε τα benchmarks *blackscholes, canneal, fluidanimate* και *rtview* με την νέα πολιτική αντικατάστασης, χρησιμοποιώντας την βέλτιστη L2 του 7.2 και θέτοντας:

- TLB size = 64 entr.
- TLB associativity = 4
- TLB page size = 4096 B

για τις παρακάτω περιπτώσεις:

L1 size (KB)	**L1 associativity**	**L1 cache block size (B)**
32	4,8	64
64	4,8	64
128	4	64

7.1.6 Ζητούμενα

1) Σαν βασική μετρική επίδοσης θα χρησιμοποιήσετε το **IPC (Instructions Per Cycle)**. Με την προϋπόθεση ότι ο κύκλος μηχανής και ο εκτελούμενος αριθμός εντολών παραμένουν σταθεροί κάθε φορά, μεγαλύτερες τιμές στο IPC υποδεικνύουν καλύτερη επίδοση.

1.1) Για κάθε μια από τις παραπάνω περιπτώσεις μελετήστε τις μεταβολές στο IPC και στην επίδοση της cache της οποίας τις παραμέτρους μεταβάλλετε. Παρουσιάστε σε γραφικές παραστάσεις τις μεταβολές αυτές για κάθε περίπτωση. Συνοψίστε τα συμπεράσματα των προηγούμενων ερωτημάτων. Τι συμπεράσματα μπορείτε να εξάγετε για τα μετροπρογράμματα που εκτελέσατε; Ποιες από τις παραμέτρους που εξετάσατε έχουν τη μεγαλύτερη επίδραση στην επίδοση;

1.2) Αν είστε ο υπεύθυνος αρχιτέκτοντας για τη σχεδίαση του συστήματος, ποια L1 και L2 cache θα διαλέγατε για την υλοποίηση;

2) Στην προηγούμενη ανάλυση υποθέσαμε ότι ο κύκλος του ρολογιού παραμένει σταθερός. Στην πράξη όμως, οι διάφορες τροποποιήσεις στα μικροαρχιτεκτονικά χαρακτηριστικά του επεξεργαστή επιφέρουν συνήθως αλλαγές και στην **διάρκεια του κύκλου**.

Επιστρέψτε στα προηγούμενα ερωτήματα, θεωρώντας κάθε φορά ως αρχικό σημείο αναφοράς την πρώτη προσομοίωση που καλείστε να εκτελέσετε στο καθένα από αυτά. Αν κάθε διπλασιασμός του associativity ή του μεγέθους προκαλεί αύξηση του κύκλου κατά 5% και 15% αντίστοιχα, πως επηρεάζονται τα συμπεράσματα που εξάγετε; Μπορείτε να συμπεράνετε και πάλι ποια L1 και L2 cache θα διαλέγατε για την υλοποίηση του συστήματος; Αν όχι, τι μπορείτε να κάνετε προκειμένου να ολοκληρώσετε σωστά το σχεδιασμό του συστήματος;

Παραδοτέο της άσκησης θα είναι ένα ηλεκτρονικό κείμενο (**pdf**, **docx** ή **odt**). *Στο ηλεκτρονικό κείμενο να αναφέρετε στην αρχή τα στοιχεία σας (Όνομα, Επώνυμο, ΑΜ).*

Η άσκηση θα παραδοθεί ηλεκτρονικά στο moodle του μαθήματος:

https://helios.ntua.gr/course/view.php?id=1039

Δουλέψτε ατομικά. Έχει ιδιαίτερη αξία για την κατανόηση του μαθήματος να κάνετε μόνοι σας την εργασία. Μην προσπαθήσετε να την αντιγράψετε από άλλους συμφοιτητές σας.

Μην αφήσετε την εργασία για το τελευταίο Σαββατοκύριακο, απαιτεί αρκετό χρόνο για οργάνωση και την εκτέλεση όλων των προσομοιώσεων, οπότε ξεκινήστε αμέσως!

Καθώς τα ίδια εργαλεία και μετροπρογράμματα είχαν χρησιμοποιηθεί και τις προηγούμενες χρονιές, είναι εξαιρετικά πιθανό αρκετές (αν όχι όλες οι) απορίες που μπορεί να προκύψουν να έχουν ήδη απαντηθεί. Σας συμβουλεύουμε λοιπόν να ψάξετε για απαντήσεις/βοήθεια στα archives της mailing list του μαθήματος, τα οποία βρίσκονται εδώ:

http://lists.cslab.ece.ntua.gr/pipermail/advcomparch/