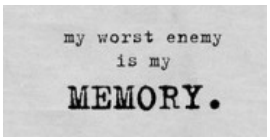
ΕΘΝΙΚΟ & ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΜΑΘΗΜΑ: **ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ ΙΙ**



Εργασία 2 (υποχρεωτική) – Κρυφές Μνήμες

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2018 – 2019

(ΕΚΦΩΝΗΣΗ) ΠΑΡΑΣΚΕΥΗ 04 ΙΑΝΑΟΥΡΙΟΥ 2019

(ΠΑΡΑΔΟΣΗ ΣΤΟ ECLASS ΜΕΧΡΙ) **ΠΑΡΑΣΚΕΥΗ 25 ΙΑΝΟΥΑΡΙΟΥ 2019**

Επώνυμο	**Όνομα**	**Αριθμός Μητρώου**	**Email**
Αθανασάκου	Αντωνία	1115201400004	sdi1400004@di.uoa.gr
Πάτσου	Στεφανία	1115201400156	sdi1400156@di.uoa.gr

Πληροφορίες για τις Υποχρεωτικές Εργασίες του μαθήματος

- Οι υποχρεωτικές εργασίες του μαθήματος είναι δύο. Σκοπός τους είναι η πλήρης κατανόηση των εννοιών του μαθήματος με χρήση αρχιτεκτονικών προσομοιωτών (architectural simulators). Η πρώτη υποχρεωτική εργασία αφορούσε τη διοχέτευση (pipelining) και η δεύτερη (αυτή) αφορά τις κρυφές μνήμες (cache memories).
- Η παράδοση των δύο εργασιών του μαθήματος είναι υποχρεωτική για τους φετινούς τριτοετείς φοιτητές καθώς επίσης και για όσους φοιτητές έχουν αριθμό μητρώου 2009 και μεταγενέστερο (δηλαδή πήραν το μάθημα για πρώτη φορά ως τριτοετείς από το εαρινό εξάμηνο του 2012 και μετά). Η βαθμολογία του μαθήματος θα προκύπτει από το γραπτό (60%), την εργασία της διοχέτευσης (20%), και την εργασία των κρυφών μνημών (20%). Καθένας από τους τρεις βαθμούς πρέπει να είναι προβιβάσιμος για να περαστεί προβιβάσιμος βαθμός στη γραμματεία. Οι προαιρετικές ασκήσεις που κατά καιρούς δίνονται δε βαθμολογούνται και δίνονται μόνο για να διευκολυνθεί η κατανόηση του μαθήματος.
- 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.
- 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 του Σεπτεμβρίου δε θα δοθούν άλλες εργασίες. Το Σεπτέμβριο θα εξεταστεί μόνο το γραπτό.
- Σε περίπτωση αντιγραφής θα μηδενίζονται όλες οι ομάδες που μετέχουν σε αυτή.
- Η παράδοση της **Εργασίας 2** πρέπει να γίνει μέχρι τα μεσάνυχτα της **Παρασκευής 25 Ιανουαρίου 2019** ηλεκτρονικά στο eclass (να ανεβάσετε ένα αρχείο zip ή rar με την τεκμηρίωσή σας σε Word ή PDF και/ή τους κώδικές σας). Μόνο ένα από τα μέλη της ομάδας να ανεβάσει την εργασία στο eclass.

Ζητούμενο

Γράψτε πρόγραμμα assembly MIPS για εκτέλεση στον προσομοιωτή Spim-Cache (που διδάχθηκε στο εργαστήριο και μπορείτε να κατεβάσετε από το site του μαθήματος στο eclass), το οποίο να εκτελεί τον εξής απλό υπολογισμό:

- Ταξινομεί σε αύξουσα σειρά 1000 προσημασμένους ακεραίους αριθμούς ο καθένας από τους οποίους αποθηκεύεται σε *ένα byte* στο τμήμα δεδομένων. Η ταξινόμηση να γίνεται με βάση έναν από τους παρακάτω πέντε (5) αλγόριθμους ταξινόμησης και η τελική ταξινομημένη λίστα να αποθηκεύεται στις ίδιες θέσεις μνήμης με την αρχική μη ταξινομημένη λίστα. Δεν πρέπει να γίνεται ανάγνωση των αριθμών από την κονσόλα ούτε εκτύπωση της τελικής λίστας στην κονσόλα. Οι αλγόριθμοι ταξινόμησης που μπορείτε να χρησιμοποιήσετε είναι οι παρακάτω:

    1. *gnome*
    2. *bubble sort*
    3. *merge sort*
    4. *insertion sort*
    5. *heapsort*

Φυσικά, το πρωταρχικό ζητούμενο είναι το πρόγραμμα να εκτελείται σωστά. Πέρα όμως από την ορθή εκτέλεση πρέπει να μειώσετε και το χρόνο εκτέλεσης του προγράμματός σας *γράφοντας κατάλληλα τον κώδικά* σας αλλά και *ρυθμίζοντας τις παραμέτρους* της κρυφής μνήμης στον Spim-Cache με βάση τα παρακάτω δεδομένα.

- Υπάρχει ξεχωριστή κρυφή μνήμη εντολών (instruction cache) και ξεχωριστή κρυφή μνήμη δεδομένων (data cache) δηλαδή Harvard architecture σύμφωνα και με την ορολογία του προσομοιωτή.

- Δεν χρησιμοποιούνται delayed branches ή delayed loads (ρυθμίστε τον προσομοιωτή κατάλληλα).
- Ο ρυθμός του ρολογιού του επεξεργαστή είναι 50 MHz αν τόσο η κρυφή μνήμη δεδομένων όσο και η κρυφή μνήμη εντολών έχουν μέγεθος 128 byte καθεμία. Αν έστω η μία από αυτές έχει μέγεθος 256 byte τότε ο ρυθμός ρολογιού είναι 48 MHz. Αν έστω μία από αυτές έχει μέγεθος 512 byte τότε ο ρυθμός είναι 46 MHz και τέλος αν έστω μία από αυτές έχει μέγεθος 1024 byte τότε ο ρυθμός είναι 45 MHz.
- Κάθε εντολή που εκτελείται διαρκεί 1 κύκλο ρολογιού αν ευστοχεί (hit) στην κρυφή μνήμη εντολών (instruction cache) και επιβαρύνεται με ποινή 25 επιπλέον κύκλων ρολογιού αν αστοχεί (miss) σε αυτή. Αυτό ισχύει (οι 25 κύκλοι ποινή) μόνο αν η instruction cache έχει μέγεθος μπλοκ ίσο με 4 byte. Αν το μέγεθος μπλοκ είναι 8 byte τότε η ποινή αυτή είναι 27 κύκλοι, ενώ αν το μέγεθος είναι 16 byte η ποινή αυτή είναι 30 κύκλοι.
- Εάν η εντολή έχει αναφορά δεδομένων και ευστοχεί (hit) στη data cache δεν έχει καμία πρόσθετη επιβάρυνση. Εάν όμως η εντολή με αναφορά δεδομένων αστοχήσει (miss) στη data cache επιβαρύνεται με ποινή 25 επιπλέον κύκλων ρολογιού όταν η data cache έχει μέγεθος μπλοκ ίσο με 4 byte. Όπως και στην instruction cache αν το μέγεθος μπλοκ της data cache είναι 8 byte τότε η ποινή αυτή είναι 27 κύκλοι, ενώ αν το μέγεθος είναι 16 byte η ποινή αυτή είναι 30 κύκλοι.
- Το μέγεθος του μπλοκ κάθε κρυφής μνήμης μπορεί να είναι οποιοδήποτε από αυτά που υποστηρίζει ο προσομοιωτής (4, 8, ή 16 byte).
- Η οργάνωση κάθε κρυφής μνήμης μπορεί να είναι οποιαδήποτε από τις παρακάτω: direct mapped, 2-way set associative, 4-way set associative.
- Η πολιτική εγγραφής πρέπει να είναι write-back-allocate.
- Ο αλγόριθμος αντικατάστασης πρέπει να είναι LRU και στις δύο κρυφές μνήμες (όταν φυσικά υπάρχει θέμα αντικατάστασης).

Αν χρησιμοποιήσετε έτοιμο κώδικα assembly mips να αναφέρετε ρητά την πηγή και τις αλλαγές που έχετε κάνει σε αυτόν ώστε να βελτιστοποιήσετε την απόδοσή του για τις ρυθμίσεις της cache που θα επιλέξετε. Κώδικας χωρίς καμία τροποποίηση από την πηγή δεν θα γίνει δεκτός.

Συμπληρώστε τον παρακάτω πίνακα με τις ρυθμίσεις που επιλέξατε.

Κρυφή μνήμη εντολών: **μέγεθος** **οργάνωση** **μέγεθος μπλοκ**	**Κρυφή μνήμη δεδομένων:** **μέγεθος** **οργάνωση** **μέγεθος μπλοκ**
μέγεθος μνήμης 128 bytes, 4-way set associative, μέγεθος μπλόκ 16 bytes	μέγεθος μνήμης 128 bytes, 4-way set associative, μέγεθος μπλόκ 16 bytes

Συμπληρώστε τον παρακάτω πίνακα με τις πληροφορίες του τελικού σας προγράμματος (σκεφθείτε τον τρόπο που θα συλλέξετε τις πληροφορίες αυτές με βάση τα στατιστικά από τον προσομοιωτή και τα παραπάνω δεδομένα της εργασίας).

Συνολικό πλήθος εντολών που εκτελούνται	**Συνολικοί κύκλοι ρολογιού για την πλήρη εκτέλεση του προγράμματος**	**CPI**	**Ρυθμός ευστοχίας κρυφής μνήμης εντολών**	**Ρυθμός ευστοχίας κρυφής μνήμης δεδομένων**	**Συνολικός χρόνος εκτέλεσης**
9150925	9778255	1.082430356	0.999999	0.986435	0.1981047801 sec

Τεκμηρίωση

[Σύντομη τεκμηρίωση της λύσης σας μέχρι 7 σελίδες – μην αλλάζετε τη μορφοποίηση του κειμένου. Η τεκμηρίωσή σας μπορεί να περιλαμβάνει παραδείγματα ορθής εκτέλεσης του προγράμματος, διαγράμματα και γενικά οποιονδήποτε σχολιασμό για την ανάπτυξη του κώδικά σας.]

Περιγραφή Κώδικα

Στην εργασία αυτή κληθήκαμε να υλοποιήσουμε ένα πρόγραμμα σε assembly MIPS το οποίο να ταξινομεί ένα πίνακα 1000 προσημασμένων ακεραίων με την προϋπόθεση ότι η τελική ταξινομημένη λίστα θα αποθηκεύεται στις ίδιες θέσεις μνήμης με την αρχική μη ταξινομημένη λίστα.

Δημιουργήσαμε, λοιπόν, έναν πίνακα από 1000 bytes με όνομα array που αποτελείται από τυχαίους προσημασμένους ακεραίους όπου για κάθε ακέραιο num ισχύει -128 <= num <= 127. Για την ταξινόμηση του πίνακα χρησιμοποιήσαμε τον αλγόριθμο gnome, ο οποίος παρουσιάζεται παρακάτω. Ο λόγος που επιλέξαμε τον συγκεκριμένο αλγόριθμο ήταν, διότι ήταν πιο εύκολα υλοποιήσιμος, δεν απαιτούσε τη χρήση επιπλέον πίνακα για την ταξινόμηση και η χρονική πολυπλοκότητα ήταν στην χειρότερη περίπτωση O(N^2), αλλά μπορούσε να φτάσει το O(N).

```
int index = 0;
while (index < n) {
  if (index == 0)
      index++;
  if (arr[index] >= arr[index - 1])
      index++;
  else {
       swap(arr[index], arr[index - 1]);
      index--;
  }
}
```

Για να ελέγξουμε ότι ο πίνακας είχε ταξινομηθεί, τον εκτυπώσαμε, αλλά αυτό το κομμάτι κώδικα έχει αφαιρεθεί διότι στην εκφώνηση ζητήθηκε να μην υπάρξει εκτύπωση. Παρακάτω φαίνεται το κομμάτι κώδικα αυτό. (Εάν το χρησιμοποιήσετε, θα πρέπει στο αρχείο sortingArray.s να αλλάξετε τη γραμμή **beq $t3, $t1, end** με **beq $t3, $t1, printArray**

```
printArray:   la $t0, array          # load array to temporary variable t0
              li $t1, 1000           # load immediate into t1, n: number of elements in array
              lb $t5, 0($t0)         # load first element of array to t4
              li $t3, 0              # load index
loop:
              li $v0, 1              # load appropriate system call code into register $v0, code for int is 1
              move $a0, $t5          # move integer to be printed into $a0:  $a0 = $t4
              syscall                # call operating system to perform operation
              addi $t3, $t3, 1       # add next position of array to t3
              beq $t3, $t1, end      # go to end if index == n
              add $t4, $t3, $t0
              lb $t5, 0($t4)         # load value of array element
              j loop
```

Για την γρηγορότερη εκτέλεση του προγράμματος μετακινήσαμε διάφορες εντολές ώστε να μην υπάρχει έλεγχος (beq, beqz…) μετά από καταχώρηση αποτελέσματος στον ίδιο καταχωρητή ή load.

Πειράματα και έλεγχοι

Για την εκτέλεση του προγράμματος ρυθμίσαμε τις παρακάτω παραμέτρους:

- Υπάρχει ξεχωριστή κρυφή μνήμη εντολών (instruction cache) και ξεχωριστή κρυφή μνήμη δεδομένων (data cache) δηλαδή Harvard architecture σύμφωνα και με την ορολογία του προσομοιωτή. **(CacheSimulation -> Cache Configuration -> Harvard Architecture)**
- Δεν χρησιμοποιούνται delayed branches ή delayed loads. **(Simulator -> Settings -> uncheck Delayed branches/loads)**
- Η πολιτική εγγραφής πρέπει να είναι write-back-allocate. **(CacheSimulation -> CacheSettings -> Writing Policy -> Write-back-allocate)**
- Ο αλγόριθμος αντικατάστασης πρέπει να είναι LRU και στις δύο κρυφές μνήμες (όταν φυσικά υπάρχει θέμα αντικατάστασης). **(CacheSimulation -> CacheSettings -> LRU)**

Στους παρακάτω πίνακες φαίνονται τα αποτελέσματα των πειραμάτων που εκτελέσαμε.

Instruction Cache

Block Size	Cache Size	Mapping	Accesses	Hits	Hit Rate	Misses
4B	128	Direct Mapping	7610026	7609995	0.999996	31
4B	256	Direct Mapping	7610026	7609995	0.999996	31
4B	512	Direct Mapping	7610026	7609995	0.999996	31
4B	1024	Direct Mapping	7610026	7609995	0.999996	31
8B	128	Direct Mapping	7610026	7610010	0.999998	16
8B	256	Direct Mapping	7610026	7610010	0.999998	16
8B	512	Direct Mapping	7610026	7610010	0.999998	16
8B	1024	Direct Mapping	7610026	7610010	0.999998	16
16B	128	Direct Mapping	7610026	7610017	0.999999	9
16B	256	Direct Mapping	7610026	7610017	0.999999	9
16B	512	Direct Mapping	7610026	7610017	0.999999	9
16B	1024	Direct Mapping	7610026	7610017	0.999999	9
4B	128	2-way set associative	7610026	7609995	0.999996	31
4B	256	2-way set associative	7610026	7609995	0.999996	31
4B	512	2-way set associative	7610026	7609995	0.999996	31
4B	1024	2-way set associative	7610026	7609995	0.999996	31
8B	128	2-way set associative	7610026	7610010	0.999998	16
8B	256	2-way set associative	7610026	7610010	0.999998	16

8B	512	2-way set associative	7610026	7610010	0.999998	16
8B	1024	2-way set associative	7610026	7610010	0.999998	16
16B	128	2-way set associative	7610026	7610017	0.999999	9
16B	256	2-way set associative	7610026	7610017	0.999999	9
16B	512	2-way set associative	7610026	7610017	0.999999	9
16B	1024	2-way set associative	7610026	7610017	0.999999	9
4B	128	4-way set associative	7610026	7609995	0.999996	31
4B	256	4-way set associative	7610026	7609995	0.999996	31
4B	512	4-way set associative	7610026	7609995	0.999996	31
4B	1024	4-way set associative	7610026	7609995	0.999996	31
8B	128	4-way set associative	7610026	7610010	0.999998	16
8B	256	4-way set associative	7610026	7610010	0.999998	16
8B	512	4-way set associative	7610026	7610010	0.999998	16
8B	1024	4-way set associative	7610026	7610010	0.999998	16
16B	128	4-way set associative	7610026	7610017	0.999999	9
16B	256	4-way set associative	7610026	7610017	0.999999	9
16B	512	4-way set associative	7610026	7610017	0.999999	9
16B	1024	4-way set associative	7610026	7610017	0.999999	9

Data Cache

Block Size	Cache Size	Mapping	Accesses	Hits	Hit Rate	Compulsory Misses	Conflict Misses	Capacity Misses	Misses
4B	128	Direct Mapping	1540899	1461094	0.948209	250	0	79554	79804
4B	256	Direct Mapping	1540899	1494056	0.969600	250	0	46592	46842
4B	512	Direct Mapping	1540899	1529238	0.992432	250	0	11410	11660
4B	1024	Direct Mapping	1540899	1540648	0.999837	250	0	0	250

8B	128	Direct Mapping	1540899	1500387	0.973709	125	0	40386	40511
8B	256	Direct Mapping	1540899	1517083	0.984544	125	0	23690	23815
8B	512	Direct Mapping	1540899	1534901	0.996107	125	0	5872	5997
8B	1024	Direct Mapping	1540899	1540773	0.999918	125	0	0	125
16B	128	Direct Mapping	1540899	1519997	0.986435	63	0	20838	20901
16B	256	Direct Mapping	1540899	1528553	0.991988	63	0	12282	12345
16B	512	Direct Mapping	1540899	1537731	0.997944	63	0	3104	3167
16B	1024	Direct Mapping	1540899	1540835	0.999958	63	0	0	63
4B	128	2-way set associative	1540899	1461094	0.948209	250	0	79554	79804
4B	256	2-way set associative	1540899	1494056	0.969600	250	0	46592	46842
4B	512	2-way set associative	1540899	1529238	0.992432	250	0	11410	11660
4B	1024	2-way set associative	1540899	1540648	0.999837	250	0	0	250
8B	128	2-way set associative	1540899	1500387	0.973709	125	0	40386	40511
8B	256	2-way set associative	1540899	1517083	0.984544	125	0	23690	23815
8B	512	2-way set associative	1540899	1534901	0.996107	125	0	5872	5997
8B	1024	2-way set associative	1540899	1540773	0.999918	125	0	0	125
16B	128	2-way set associative	1540899	1519997	0.986435	63	0	20838	20901
16B	256	2-way set associative	1540899	1528553	0.991988	63	0	12282	12345
16B	512	2-way set associative	1540899	1537731	0.997944	63	0	3104	3167

16B	1024	2-way set associative	1540899	1540835	0.999958	63	0	0	63
4B	128	4-way set associative	1540899	1461094	0.948209	250	0	79554	79804
4B	256	4-way set associative	1540899	1494056	0.969600	250	0	46592	46842
4B	512	4-way set associative	1540899	1529238	0.992432	250	0	11410	11660
4B	1024	4-way set associative	1540899	1540648	0.999837	250	0	0	250
8B	128	4-way set associative	1540899	1500387	0.973709	125	0	40386	40511
8B	256	4-way set associative	1540899	1517083	0.984544	125	0	23690	23815
8B	512	4-way set associative	1540899	1534901	0.996107	125	0	5872	5997
8B	1024	4-way set associative	1540899	1540773	0.999918	125	0	0	125
16B	128	4-way set associative	1540899	1519997	0.986435	63	0	20838	20901
16B	256	4-way set associative	1540899	1528553	0.991988	63	0	12282	12345
16B	512	4-way set associative	1540899	1537731	0.997944	63	0	3104	3167
16B	1024	4-way set associative	1540899	1540835	0.999958	63	0	0	63

Τύποι & Αποτελέσματα

Με βάση την εκφώνηση, κάθε εντολή διαρκεί (1 κύκλο + επιβάρυνση), η οποία επιβάρυνση επηρεάζεται από το μέγεθος του block. Ο ρυθμός ρολογιού επηρεάζεται από το μέγεθος της κρυφής μνήμης. Όσον αφορά την κρυφή μνήμη δεδομένων, αν η αναφορά μίας τιμής «hits», τότε δεν υπάρχει επιβάρυνση, διαφορετικά, όπως και στην μνήμη εντολών, υπάρχει η αντίστοιχη επιβάρυνση. Για να υπολογίσουμε, λοιπόν, τα στατιστικά της εργασίας, χρησιμοποιήσαμε τους παρακάτω τύπους:

- **Εντολές** = Instruction Accesses + Data Accesses
- **Κύκλοι Ρολογιού** = Instructions + (Instruction Misses) * (Instruction Miss Penalty) + (Data Accesses – Hits Data) * (Data Miss Penalty)
- **CPI** = 1 + (Instruction Miss Rate) *** (**Instruction Miss Penalty) + (Data Accesses/Instruction Accesses) * (Data Miss Penalty) * (Data Miss Rate)
- **Χρόνος Εκτέλεσης Προγράμματος** = (CPI * Instructions) / Clock Rate

Οι συνολικές εντολές που εκτελέστηκαν κατά το πρόγραμμα είναι:
Εντολές = 7610026 + 1540899 = 9150925

Πριν τον υπολογισμό τω στατιστικών και την επιλογή της πιο βέλτιστης αρχιτεκτονικής, έχουμε να κάνουμε κάποιες παρατηρήσεις για τα στατιστικά των πινάκων που παράχθηκαν από την πολλαπλή εκτέλεση του προγράμματος. Όσον αφορά την κρυφή μνήμη εντολών, παρατηρήθηκε ότι αλλάζει το Hit Rate μόνο όταν αλλάζει τιμή το μέγεθος του μπλοκ. Έχουμε σχεδόν 100% Hit Rate με μέγεθος μπλοκ 16 bytes. Ακόμη, η κρυφή μνήμη δεδομένων επηρεάζεται τόσο από το μέγεθος του μπλοκ, όσο και από το μέγεθος της ολικής κρυφής μνήμης. Αν και με αρχιτεκτονική με μέγεθος μπλοκ 16 bytes και μέγεθος μνήμης 1024, επιτυγχάνουμε σχεδόν κανένα Miss, εντούτοις δεν προτείνεται, διότι ο ρυθμός ρολογιού αρχίζει και μειώνεται σημαντικά. Ακόμη, τα περισσότερα bytes στο μέγεθος μπλοκ, προσδίδουν ποινή στους κύκλους ρολογιού. Οπότε, θα ήταν ορθό να επιλέξουμε μία αρχιτεκτονική που δεν θα χρησιμοποιεί πολύ μνήμη αλλά και δεν θα είναι αργή στον υπολογισμό και δεν θα έχει μεγάλο χρόνο εκτέλεσης. Τέλος, και στις δύο κρυφές μνήμες, παρατηρείται ότι η οργάνωση δεν παίζει μεγάλο ρόλο στην αλλαγή του Hit Rate, λόγω της μεγαλύτερης σημασίας της αλλαγής μεγέθους μπλοκ ή μνήμης. Πληροφοριακά, θα χρησιμοποιηθεί η 4-way set associative, επειδή, όπως ξέρουμε από την θεωρία, μειώνει σημαντικά τις αστοχίες. Παρακάτω φαίνονται ενδεικτικοί υπολογισμοί για την κάθε περίσταση. Μέσω αυτών, θα επιλεχθεί η καλύτερη αρχιτεκτονική για κάθε κρυφή μνήμη. Για να γίνει πιο εύκολα αυτό, οι γενικοί τύποι θα χωριστούν για κάθε κρυφή μνήμη. Πιο συγκεκριμένα, για την κρυφή μνήμη εντολών θα χρησιμοποιηθεί ο τύπος:

Κύκλοι Ρολογιού Εντολών = (Instruction Misses) * (Instruction Miss Penalty)

Για κάθε διαφορετικό μέγεθος μπλοκ. Έχουμε:

- Για μέγεθος μπλοκ 4 bytes: Κύκλοι Ρολογιού Εντολών = 31 * 25 = 775.
- Για μέγεθος μπλοκ 8 bytes: Κύκλοι Ρολογιού Εντολών = 16 * 27 = 432.
- Για μέγεθος μπλοκ 16 bytes: Κύκλοι Ρολογιού Εντολών = 9 * 30 = 270.

Άρα, για την συγκεκριμένη υλοποίηση, προτιμούμε αρχιτεκτονική με μέγεθος μπλόκ 16 bytes, μέγεθος μνήμης 128 bytes και 4(ή 2)-way set associative για την κρυφή μνήμη εντολών.

Όσον αφορά την κρυφή μνήμη δεδομένων, οι περιπτώσεις εδώ είναι πολλές. Πρέπει να υπολογιστεί για κάθε περίπτωση, ο συνολικός χρόνος εκτέλεσης, πριν τον καθορισμό της αρχιτεκτονικής της. Για τα παραπάνω δεδομένα που αφορούν την αρχιτεκτονική της κρυφής μνήμης εντολών, θα χρησιμοποιηθούν τα στατιστικά από την επιλογή που έγινε, δηλαδή από την αρχιτεκτονική με μέγεθος μπλόκ 16 bytes, μέγεθος μνήμης 128 bytes και 4-way set associative για την κρυφή μνήμη εντολών.

Data Cache						**Program**		
Block Size	**Cache Size**	**Accesses**	**Hits**	**Hit Rate**	**Misses**	**Cycles**	**CPI**	**CPU TIME (sec)**
4B	128	1540899	1461094	0.948209	79805	11146320	1.262199604	0.2310058782
4B	256	1540899	1494056	0.969600	46843	10322270	1.153916891	0.2199876443
4B	512	1540899	1529238	0.992432	11661	9442720	1.038339736	0.2065601967
4B	1024	1540899	1540648	0.999837	251	9157470	1.000855117	0.2035277803
8B	128	1540899	1500387	0.973709	40512	10245019	1.143763798	0.2093299347
8B	256	1540899	1517083	0.984544	23816	9794227	1.143763798	0.2180520153
8B	512	1540899	1534901	0.996107	5998	9313141	1.084528482	0.2157486695
8B	1024	1540899	1540773	0.999918	126	9154597	1.000478297	0.2034511524
16B	128	1540899	1519997	0.986435	20902	9778255	1.082430356	0.1981047801
16B	256	1540899	1528553	0.991988	12346	9521575	1.048698754	0.1999284093
16B	512	1540899	1537731	0.997944	3168	9246235	1.012519136	0.2014236234

16B	1024	1540899	1540835	0.999958	64	9153115	1.000285128	0.2034118708

Με βάση τον παραπάνω πίνακα, φαίνεται πως το πρόγραμμα είναι πιο βέλτιστο όταν η κρυφή μνήμη δεδομένων είναι της αρχιτεκτονικής με μέγεθος μπλοκ 16 bytes, μέγεθος μνήμης 128 bytes. Επιλέγουμε, πληροφοριακά, να είναι οργάνωσης 4-way set associative όπως της κρυφής μνήμης εντολών. Οπότε, μπορούμε με ασφάλεια να συμπληρώσουμε και τα υπόλοιπα κενά των στατιστικών για την συγκεκριμένη υλοποίηση.