ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

Προχωρημένα Θέματα Αρχιτεκτονικής Υπολογιστών

Μνήμη cache χρησιμοποιώντας τον προσομοιωτή PCSpimCache

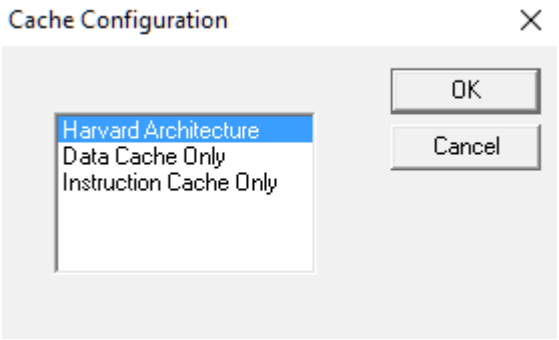
1η Εργασία
Εμμανουήλ Παπαδημητρίου
ΑΜ: mcse19021

Μεταπτυχιακό Πρόγραμμα "Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών"

Ερώτηση 1

α)

Για να επιλέξουμε την αρχιτεκτονική Harvard, επιλέγουμε
CacheSimulation → Επιλογή Cache Configuration → Και στη συνέχεια επιλέγουμε από το παράθυρο το Harvard Architecture



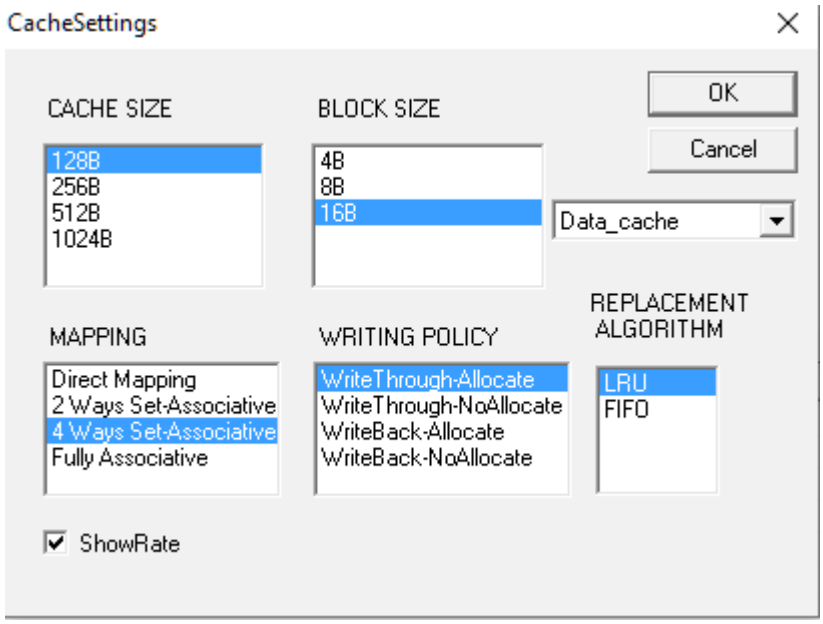
β)

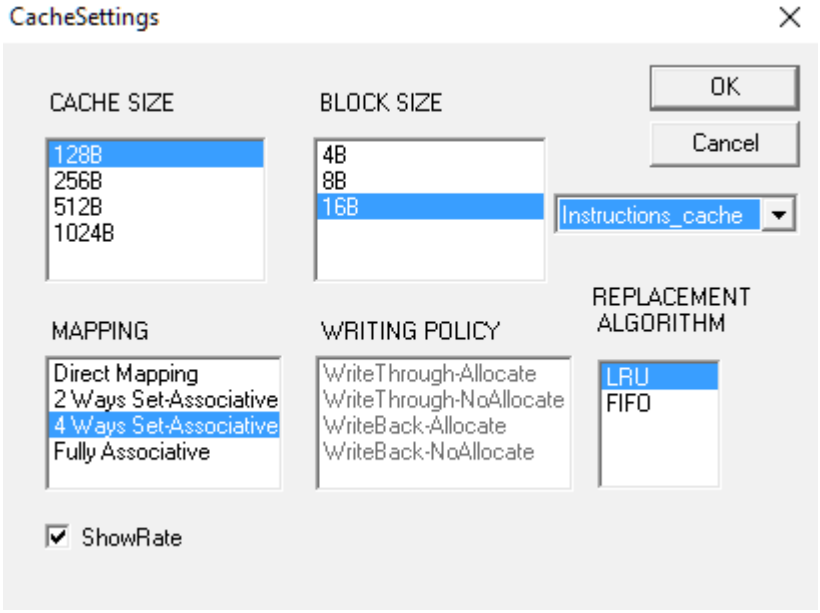
Για την ρύθμιση της cache δεδομένων και της cache εντολών, ακολοθούμε τα παρακάτω βήματα:

CacheSimulation → CacheSettings

Και στο παράθυρο επιλέγουμε και για Data_cache και για Instructions_cache

- Cache Size = 128B
- Block Size = 16B
- Mapping = 4 Ways Set-Associative
- Writing Policy = WriteThrough-Allocate (δεν έχει την επιλογή διαθέσιμη στην Instructions_cache)
- Replacement Algorithm = LRU


CacheSettings
CACHE SIZE
128B
256B
512B
1024B
BLOCK SIZE
4B
8B
16B
OK
Cancel
Data_cache
REPLACEMENT ALGORITHM
LRU
FIFO
MAPPING
Direct Mapping
2 Ways Set-Associative
4 Ways Set-Associative
Fully Associative
WRITING POLICY
WriteThrough-Allocate
WriteThrough-NoAllocate
WriteBack-Allocate
WriteBack-NoAllocate
ShowRate


CacheSettings
CACHE SIZE
128B
256B
512B
1024B
BLOCK SIZE
4B
8B
16B
OK
Cancel
Instructions_cache
REPLACEMENT ALGORITHM
LRU
FIFO
MAPPING
Direct Mapping
2 Ways Set-Associative
4 Ways Set-Associative
Fully Associative
WRITING POLICY
WriteThrough-Allocate
WriteThrough-NoAllocate
WriteBack-Allocate
WriteBack-NoAllocate
ShowRate

Ερώτηση 2

Αρχικά, επιλέξαμε στις ρυθμίσεις της cache (Cache Settings) στην οργάνωση (mapping), το 4 Ways Set-Associative. Αυτή μας η επιλογή, συμβολίζει τον αριθμό των block σε κάθε set, άρα έχοουμε 4 block σε κάθε set.

Συνολικά έχουμε:

- 8 block σε κάθε μνήμη
- 2 set σε κάθε μνήμη

Όπως παρατηρούμε στο παράθυρο της cache εντολών

Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc	V	LRU	Tag (h)	Instructions (h) Way 1	Acc
0	0	0				0	0			
1	0	0				0	0			

Acc	V	LRU	Tag (h)	Instructions (h) Way 2	Acc	V	LRU	Tag (h)	Instructions (h) Way 3	Acc
	0	0				0	0			
	0	0				0	0			

Όπως παρατηρούμε στο παράθυρο της cache δεδομένων

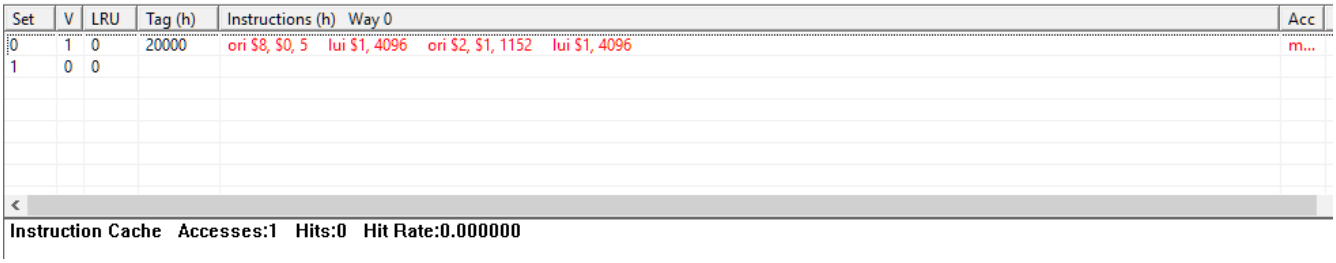
Set	V	LRU	Tag (h)	Data (h) Way 0	Acc	V	LRU	Tag (h)	Data (h) Way 1	Acc
0	0	0				0	0			
1	0	0				0	0			

Acc	V	LRU	Tag (h)	Data (h) Way 2	Acc	V	LRU	Tag (h)	Data (h) Way 3	Acc
	0	0				0	0			
	0	0				0	0			

Ερώτηση 3

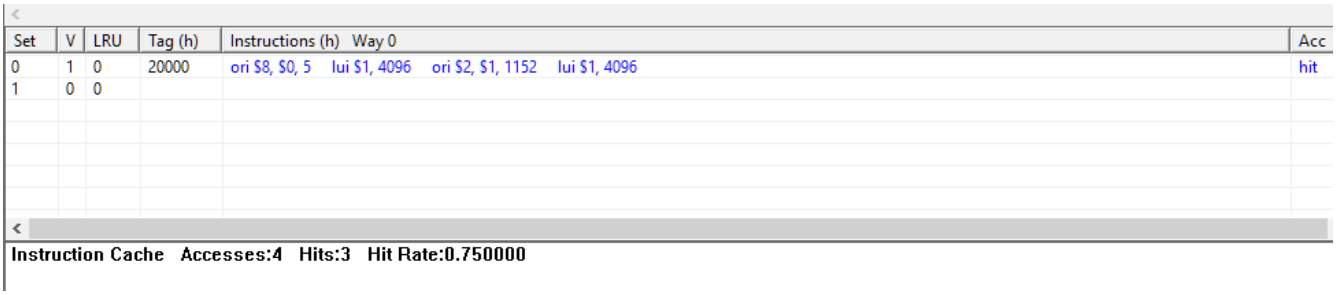
Μετά την φόρτωση του αρχείου της εργασίας, ξεκινάει η επανάληψη στην γραμμή 7.

-Με την εκτέλεση της εντολής **li $8, 5**, βλέπουμε ότι τοποθετεί στο πρώτο block του πρώτου set τις πρώτες 4 εντολές που θα εκτελεστούν και έχει την ένδειξη miss με Tag(h) ίσον με 20000. Tag είναι η διεύθυνση της εντολής που βρίσκεται στο instructions. Κάθε εντολή είναι 4 byte και χωράει 16 byte το block άρα 4 εντολές. Είχαμε ένδειξη miss, γιατί δεν υπήρχαν αυτές οι 4 εντολές στην μνήμη, άρα δεν τις βρήκε.

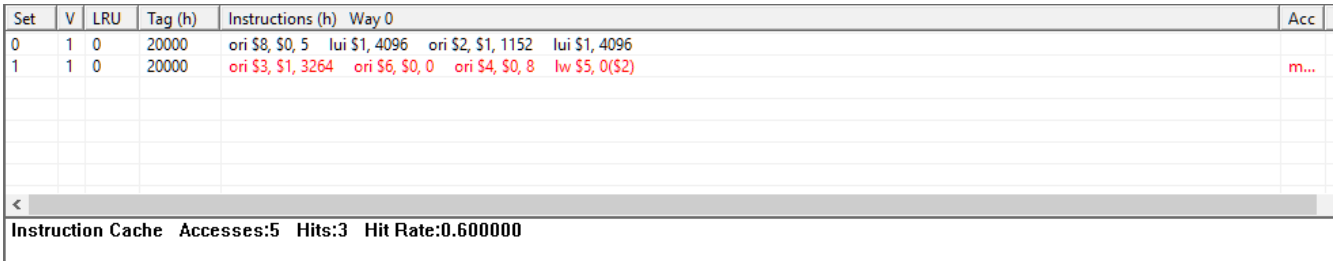


-Στη συνέχεια, γίνεται η εκτέλεση της εντολής **la $2, Array_A**, αλλά τώρα έχουμε ένδειξη hit, καθώς η εντολή αυτή υπάρχει στο block εντολών του πρώτου set με διεύθυνση Tag(h) = 20000.

-Στις επόμενες δύο εκτελέσεις έχουμε ξανά hit καθώς υπάρχουν μέσα στο block μας.



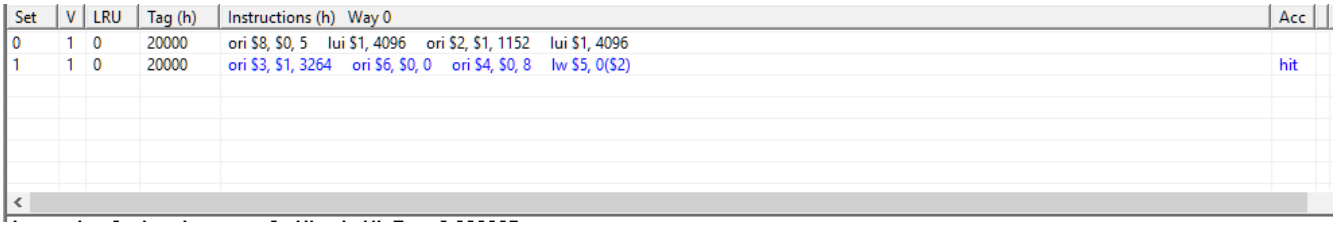
-Με την εκτέλεση της εντολής **la $3, Array_B**, έχουμε miss καθώς δεν υπάρχει αυτο το set εντολών στην μνήμη μας. Οπότε, θα τοποθετήσει τις επόμενες 4 εσωτερικές εντολές στην μνήμη. Όμως, υπάρχουν ήδη set εντολών στο πρώτο block του πρώτου set άρα θα τα τοποθετήσει στο δεύτερο block του δεύτερου set.



Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc
0	1	0	20000	ori $8, $0, 5 lui $1, 4096 ori $2, $1, 1152 lui $1, 4096	
1	1	0	20000	ori $3, $1, 3264 ori $6, $0, 0 ori $4, $0, 8 lw $5, 0($2)	m...

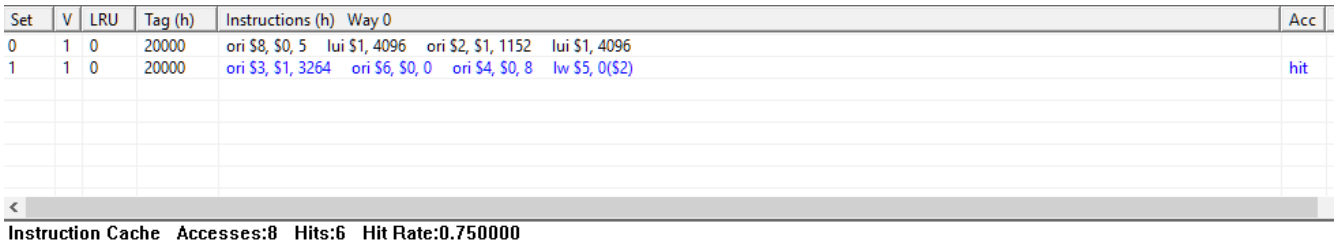
Instruction Cache Accesses:5 Hits:3 Hit Rate:0.600000

-Για την εντολή **li $4, 8**, θα έχουμε hit καθώς οι δύο εσωτερικές εντολές της υπάρχουν στο δεύτερο πρώτο block του δεύτερου set.



Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc
0	1	0	20000	ori $8, $0, 5 lui $1, 4096 ori $2, $1, 1152 lui $1, 4096	
1	1	0	20000	ori $3, $1, 3264 ori $6, $0, 0 ori $4, $0, 8 lw $5, 0($2)	hit

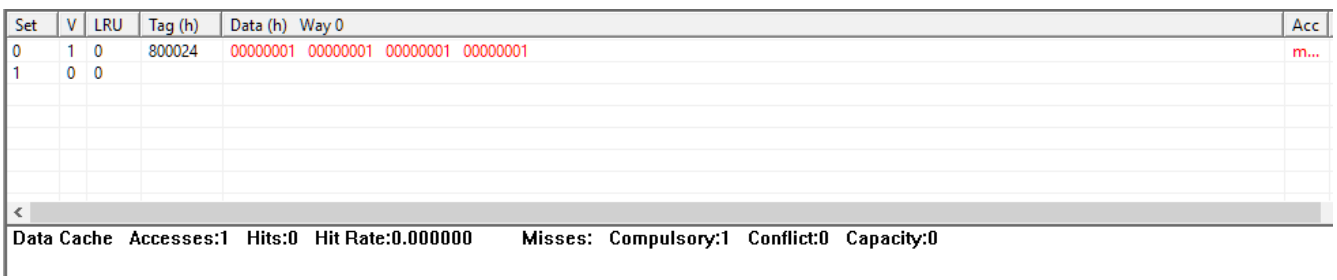
-Η εντολή **lw $5, 0($2)**, είναι η τελευταία εντολή που βρίσκεται στο block εντολών του δεύτερου set άρα θα έχουμε και εδώ την ένδειξη hit.



Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc
0	1	0	20000	ori $8, $0, 5 lui $1, 4096 ori $2, $1, 1152 lui $1, 4096	
1	1	0	20000	ori $3, $1, 3264 ori $6, $0, 0 ori $4, $0, 8 lw $5, 0($2)	hit

Instruction Cache Accesses:8 Hits:6 Hit Rate:0.750000

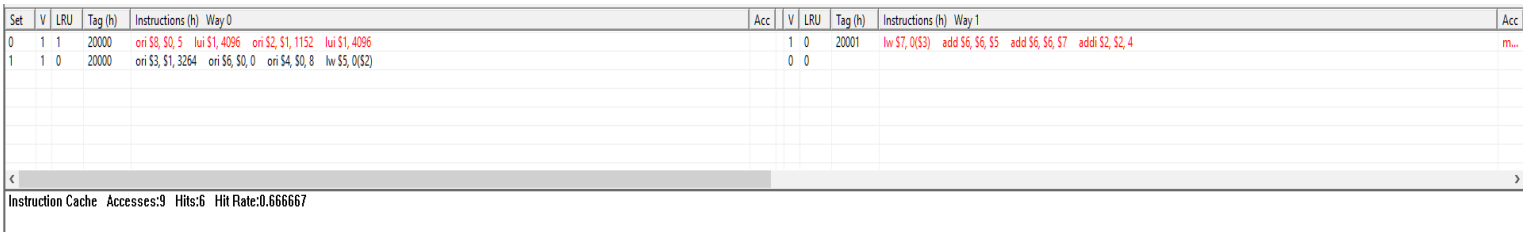
Επίσης, η εντολή αυτή γράφει στην cache δεδομένων γιατί παίρνει δεδομένα και το αποτέλεσμα είναι miss καθώς δεν υπήρχαν αυτές οι εντολές στο πρώτο block του πρώτου set που έψαξε.



Set	V	LRU	Tag (h)	Data (h) Way 0	Acc
0	1	0	800024	00000001 00000001 00000001 00000001	m...
1	0	0			

Data Cache Accesses:1 Hits:0 Hit Rate:0.000000 Misses: Compulsory:1 Conflict:0 Capacity:0

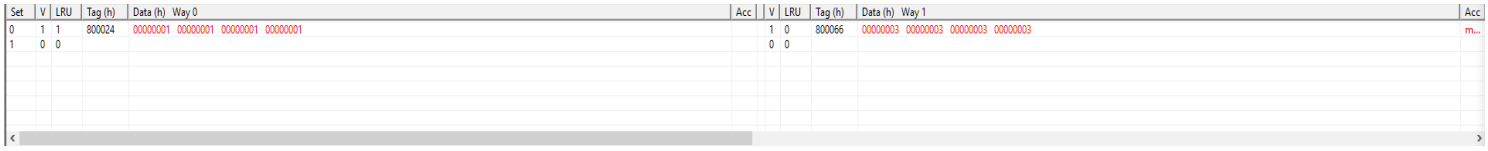
-Με την εκτέλεση της εντολής **lw $7 0($3),** βλέπουμε ότι ψάχνει την εντολή στο πρώτο block του πρώτου set αλλά δεν το βρίσκει και βγάζει miss, οπότε δημιουργεί νέα διεύθυνση Tag = 20001 και γράφει στο δεύτερο block του πρώτου set τις επόμενες 4 εσωτερικές εντολές.



Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc	V	LRU	Tag (h)	Instructions (h) Way 1	Acc
0	1	1	20000	ori $8, $0, 5 lui $1, 4096 ori $2, $1, 1152 lui $1, 4096		1	0	20001	lw $7, 0($3) add $6, $6, $5 add $6, $6, $7 addi $2, $2, 4	m...
1	1	0	20000	ori $3, $1, 3264 ori $6, $0, 0 ori $4, $0, 8 lw $5, 0($2)		0	0			

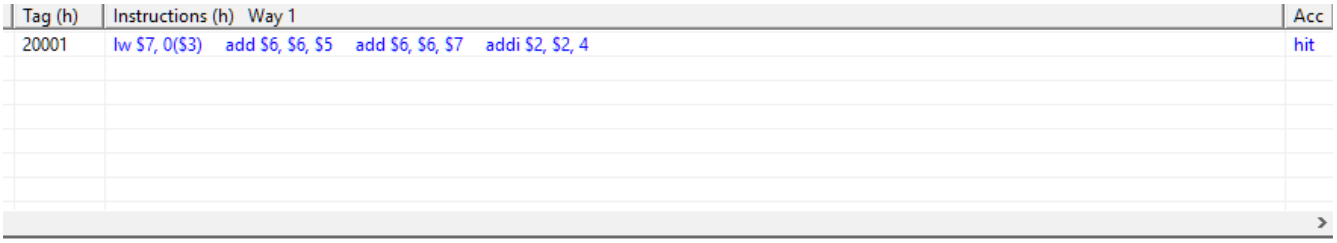
Instruction Cache Accesses:9 Hits:6 Hit Rate:0.666667

Επίσης και εδώ επειδή είναι εντολή που γράφει δεδομένα, βλέπουμε ότι τοποθετεί στην cache δεδομένω στο **αντίστοιχο** με την cache εντολών. Δηλαδή γράφει στο δεύτερο block του πρώτου set.



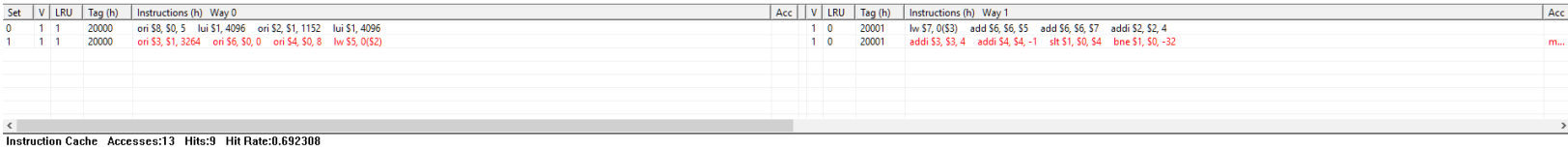
Set	V	LRU	Tag (h)	Data (h) Way 0	Acc	V	LRU	Tag (h)	Data (h) Way 1	Acc
0	1	1	800024	00000001 00000001 00000001 00000001		1	0	800066	00000003 00000003 00000003 00000003	m...
1	0	0				0	0			

-Για την εντολή **add $6, $6,$5,** βγάζει hit γιατί την βρίσκει στο set και συγκεκριμένα στο δεύτερο block του πρώτου set. Το ίδιο ισχύει και για τις εντολές **add $6, $6,$7** και **addi $2, $2,4.** Καθώς τις βρίσκ**ει** στο block με διεύθυνση Tag = 20001

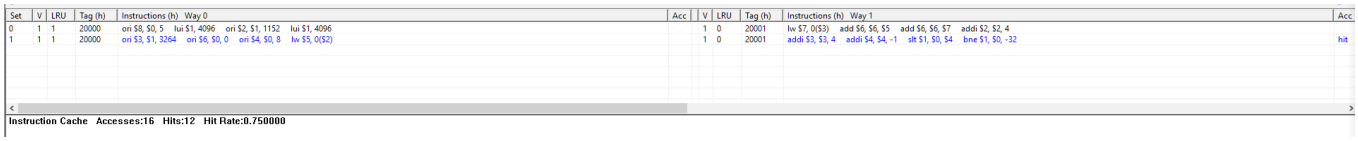


Tag (h)	Instructions (h) Way 1	Acc
20001	lw $7, 0($3) add $6, $6, $5 add $6, $6, $7 addi $2, $2, 4	hit

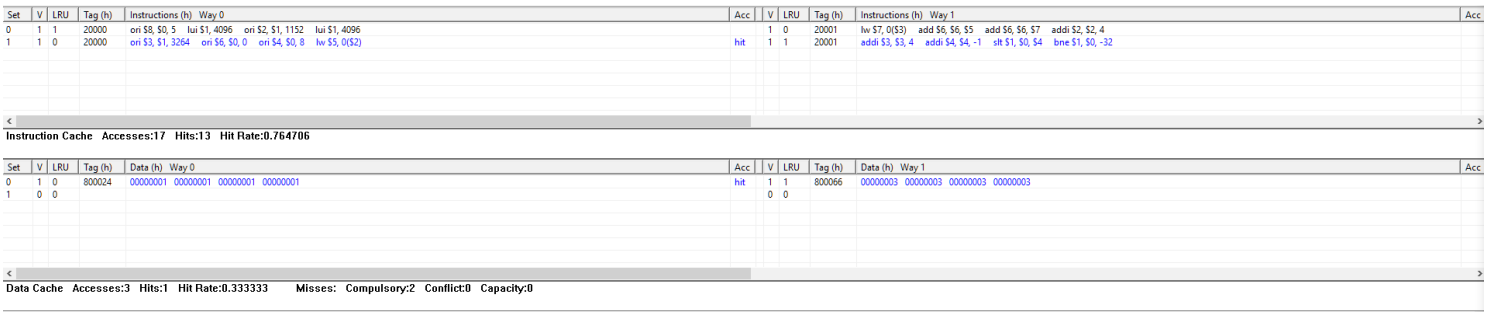
-Η επόμενη εντολή είναι η **addi $3, $3,4,** που δεν έχει τοποθετηθεί σε κάποιο block μέχρις στιγμής. Άρα το προφανές που γίνεται είναι να ψάξει στο δεύτερο set για την εντολή. Δεν την βρίσκει και βγάζει miss αρχικά καθώς έχουμε 4 άλλες εσωτερικές εντολές. Στην συνέχεια, τοποθετεί τις επόμενες 4 εσωτερικές εντολές με πρώτη την **addi $3, $3,4,** και τις βάζει στο δεύτερο block του δεύτερου set.



-Κατά σειρά εκτελεί τις επόμενες 3 εσωτερικές εντολές , πρώτα την **addi $4,$4, -1** και στη συνέχεια την **bgt $4,$0, loop** που περιέχει δύο εσωτερικές εντολές, οπότε στις επόμενες εκτελέσεις θα έχουμε ένδειξη hit καθώς τις βρίσκει μέσα στο block. Αυτές ήταν και οι τελευταίες εντολές, πριν ξαναξεκινήσουμε την επανάληψη μας.

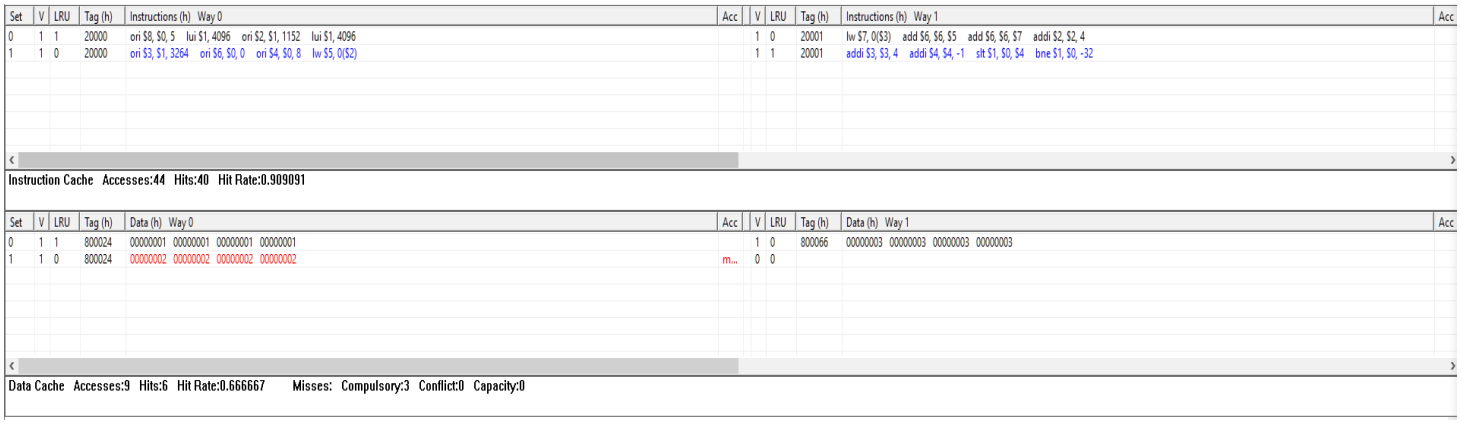


-Η δεύτερη επανάληψη ξεκινάει με την εντολή **lw $5,0($2)** και έχει την ένδειξη hit και στο instruction cache και στην data cache καθώς υπάρχει το set εντολών αυτών στο δεύτερο set του πρώτου block με Tag = 20000 και επίσης στην data cache έχει γράψει την τιμή στο πρώτο block του πρώτου set με Tag = 800024



-Για τις υπόλοιπες εντολές της επανάληψης δεν έχουμε κάτι καινούργιο, δηλαδή έχουμε πάντα hit καθώς τις βρίσκει στα block των sets.

-Στην τρίτη επανάληψη όμως, αν και η εντολή **lw $5,0($2)** υπάρχει στο δεύτερο set του πρώτου block, γράφει νέα τιμή στην cache δεδομένων άρα έχουμε miss ενώ στην cache εντολών έχουμε hit. Άρα γράφει στο δεύτερο set του πρώτου block της cache δεδομένων με Tag = 800024



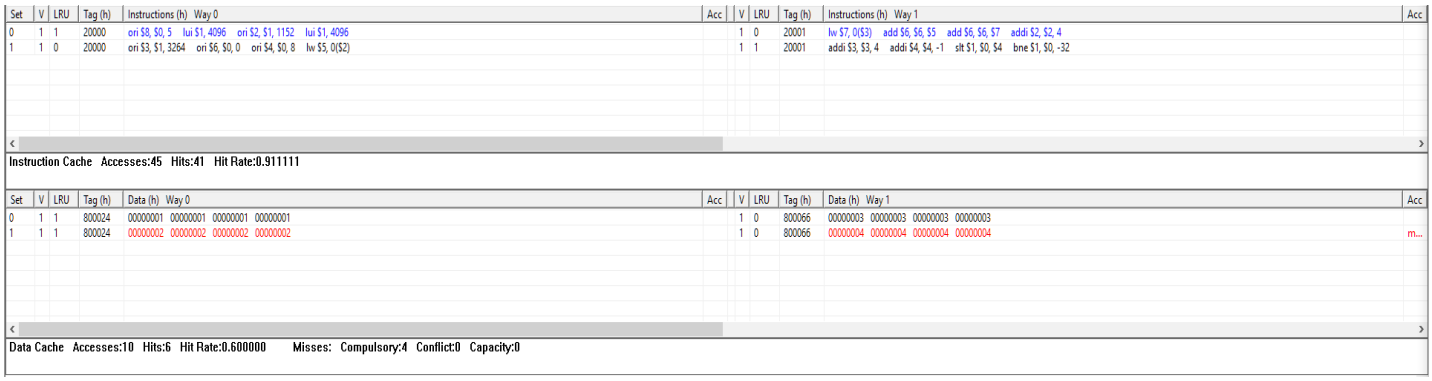
Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc	V	LRU	Tag (h)	Instructions (h) Way 1	Acc
0	1	1	20000	ori $8, $0, 5 lui $1, 4096 ori $2, $1, 1152 lui $1, 4096		1	0	20001	lw $7, 0($3) add $6, $6, $5 add $6, $6, $7 addi $2, $2, 4	
1	1	0	20000	ori $3, $1, 3264 ori $6, $0, 0 ori $4, $0, 8 lw $5, 0($2)		1	1	20001	addi $3, $3, 4 addi $4, $4, -1 slt $1, $0, $4 bne $1, $0, -32	

Instruction Cache Accesses:44 Hits:40 Hit Rate:0.909091

Set	V	LRU	Tag (h)	Data (h) Way 0	Acc	V	LRU	Tag (h)	Data (h) Way 1	Acc
0	1	1	800024	00000001 00000001 00000001 00000001		1	0	800066	00000003 00000003 00000003 00000003	
1	1	0	800024	00000002 00000002 00000002 00000002	m...	0	0			

Data Cache Accesses:9 Hits:6 Hit Rate:0.666667 Misses: Compulsory:3 Conflict:0 Capacity:0

-Το ίδιο ισχύει και για την **lw $7,0($3),** καθώς έχει νέα δεδομένα και θα γράψει στο δεύτερο block του δεύτερου set οπότε έχει ένδειξη miss στην cache δεδομένων και θα γράψει με διεύθυνση Tag = 800066 αλλά στην cache εντολών έχει hit καθώς υπάρχει στο δεύτερο block του δεύτερου set με Tag = 20001



Set	V	LRU	Tag (h)	Instructions (h) Way 0	Acc	V	LRU	Tag (h)	Instructions (h) Way 1	Acc
0	1	1	20000	ori $8, $0, 5 lui $1, 4096 ori $2, $1, 1152 lui $1, 4096		1	0	20001	lw $7, 0($3) add $6, $6, $5 add $6, $6, $7 addi $2, $2, 4	
1	1	0	20000	ori $3, $1, 3264 ori $6, $0, 0 ori $4, $0, 8 lw $5, 0($2)		1	1	20001	addi $3, $3, 4 addi $4, $4, -1 slt $1, $0, $4 bne $1, $0, -32	

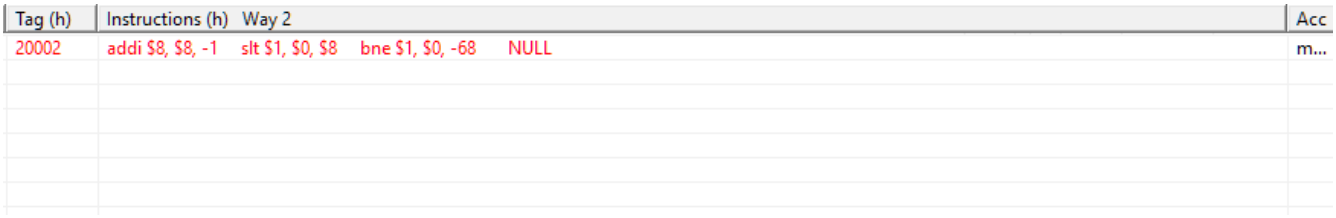
Instruction Cache Accesses:45 Hits:41 Hit Rate:0.911111

Set	V	LRU	Tag (h)	Data (h) Way 0	Acc	V	LRU	Tag (h)	Data (h) Way 1	Acc
0	1	1	800024	00000001 00000001 00000001 00000001		1	0	800066	00000003 00000003 00000003 00000003	
1	1	1	800024	00000002 00000002 00000002 00000002		1	0	800066	00000004 00000004 00000004 00000004	m...

Data Cache Accesses:10 Hits:6 Hit Rate:0.600000 Misses: Compulsory:4 Conflict:0 Capacity:0

-Στην συνέχεια έχουμε κιάλλες επαναλήψεις χωρίς αστοχίες (miss) για τις εντολές και για την cache εντολών και για την cache δεδομένων καθώς υπάρχουν στα blocks των set οπότε έχουμε πάντα hit μέχρι και την έξοδο μας από την επανάληψη.

-Μόλις βγούμε από την επανάληψη, γίνεται η εκτέλεση της εντολής **addi $8, $8, -1** η οποία όμως δεν βρίσκεται σε κάποιο block κάποιου set οπότε έχουμε miss. Έτσι, θα τοποθετήσει τις επόμενες 4 εσωτερικές εντολές στο **τρίτο** block του πρώτου set με διεύθυνση Tag = 20002.



Tag (h)	Instructions (h)	Way 2			Acc
20002	addi $8, $8, -1	slt $1, $0, $8	bne $1, $0, -68	NULL	m...

-Στη συνέχεια εκτελείται η εντολή bgt $8, $0, ext_loop που περιέχει δύο εσωτερικές εντολές και θα έχουμε ένδειξη hit καθώς υπάρχουν στο block που αναφέραμε, δηλαδή στο τρίτο block του πρώτου set.

Tag (h)	Instructions (h)	Way 2			Acc
20002	addi $8, $8, -1	slt $1, $0, $8	bne $1, $0, -68	NULL	hit

-Παρατηρούμε ότι τοποθέτησε NULL στην διεύθυνση Tag = 20002 καθώς όπως είπαμε, μπαίνουν 4 εντολές στο block και επειδή δεν υπήρχε άλλη εντολή μετά τις εσωτερικές τις τελευταίας έτσι γράφτηκε NULL.

-Στη συνέχεια έχουμε τις υπόλοιπες εκτέλεσεις και επαναλήψεις των εντολών μας και παρατηρούμε ότι έχουμε πάντα hit και στην cache εντολών και στην cache δεδομένων. Αυτό συμβαίνει γιατί όλες οι εντολές μας υπάρχουν σε blocks των sets και όλα τα δεδομένα μας υπάρχουν σε block των sets στην cache δεδομένων. Στη συνέχεια τερματίζεται το πρόγραμμα για N=5.

Ερώτηση 4

N	Hit Rate
5	0.987715
10	0.993842
100	0.999383

Παρατηρούμε ότι με την αύξηση του N έχουμε καλύτερο hit rate. Αυτό συμβαίνει γιατί έχουμε περισσότερες επαναλήψεις και έχουμε περισσότερα περάσματα από τα set εντολών που υπάρχουν ήδη στα blocks των set.

Τα είδη αστοχιών (miss) που συμβαίνουν:

- Όταν δεν βρίσκει την εντολή στην cache δεδομένων σε κάποιο block ενός set οπότε γράφει τις νέες 4 εντολές σε άλλο block.
- Στην φόρτωση δεδομένων στην cache δεδομένων, όλες οι καινούργιες τιμές χρειάζεται να μπουν σε νέο block καθώς είναι διαφορετικές κάθε φορά οπότε έχουμε miss για κάθε τέτοια περίπτωση.