Μιχάλης Ψυχής
el22094, michalis.psychis@gmail.com
3η Σειρά Ασκήσεων, Αρχιτεκτονική Υπολογιστών. Τμήμα 3

Μέρος Α

Έστω ότι Hit_Rate = x. Τότε Miss_Rate = 1 - x. Εάν ο μέσος χρόνος πρόσβασης στη μνήμη είναι 2.4 κύκλοι, τότε ισχύει: Hit_Rate * Hit_Time + Miss_Rate * Miss_Penalty = 2.4 => 1 * x + 80 * (1-x) = 2.4 => 79x = 77.6 => **Hit_Rate = 0.9823** και **Miss_Rate = 0.0177**.

Προσθέτουμε ένα δεύτερο επίπεδο κρυφής μνήμης, L2. Εάν η επιτάχυνση του μέσου χρόνου πρόσβασης στην μνήμη, λόγω της νέας αρχιτεκτονικής, είναι 1.65, τότε για τον νέο μέσο χρόνο πρόσβασης θα ισχύει: New_Access_Time * 1.65 = Old_Access_Time => New_Access_Time = 1.455 κύκλοι.

Επομένως με χρησιμοποιώντας την παρακάτω σχέση, μπορούμε να υπολογίσουμε το Hit_Rate_2 της L2. Hit_Rate_1*Hit_Time_1 + Miss_Penalty_1*(Hit_Rate_2*Hit_Time_2 + Miss_Rate_2*Miss_Penalty_2) = 1.455 => 0.9823*1 + 0.0177(6*y + 80*(1-y)) = 1.455 => 0.0177(6*y + 80*(1-y)) = 0.4727 => 6*y + 80*(1-y) = 26.71 => 74y = 53.294 => y = 0.72 . Άρα **Hit_Rate_2 = 0.72** .

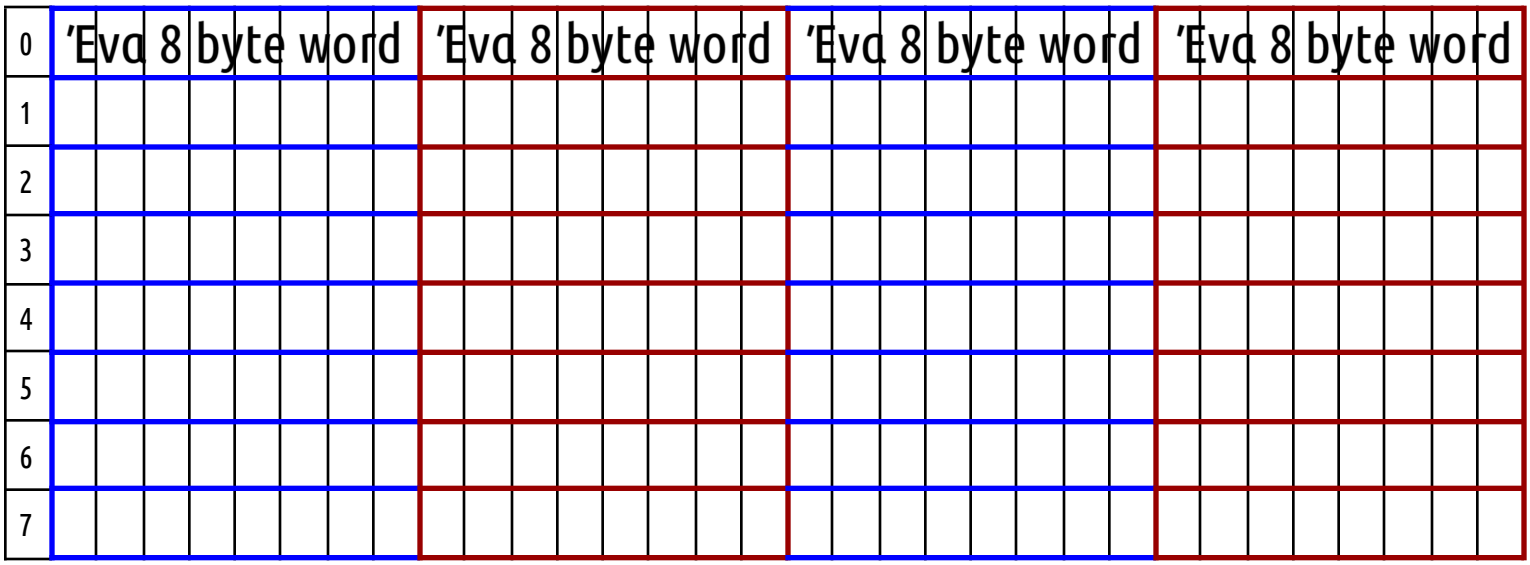
Μέρος Β

Α) Σύμφωνα με την αρχιτεκτονική του επεξεργαστή, κάθε word αποτελείται από 8 bytes. Έτσι στο 32 byte block, χωράνε 4 words. Το block_offset θα πρέπει να μπορεί να αντιστοιχεί σε κάθε byte του block. Επομένως, απαιτούνται 5bits για να έχουμε 32 θέσεις. Πιο συγκεκριμένα τα 2 bits αφορούν την επιλογή ενός εκ των τεσσάρων words, ενώ τα υπόλοιπα 3, σε ένα εκ των οκτώ επιμέρους byte του κάθε word.

Η cache μνήμη έχει χωρητικότητα 256 bytes. Κάθε block έχει μέγεθος 32 bytes. Άρα έχουμε 256/32 = 8 διαφορετικά blocks, δηλαδή χρειαζόμαστε ένα 3 bit index. Τα υπόλοιπα 32 - 5 - 3 = 24 bits αντιστοιχούν στο tag.

Άρα έχουμε (address):

tag (24bits)	index (3bits)	block_offset (5bits)



B) Πρώτο βήμα για να κατανοήσουμε πως λειτουργεί η cache μνήμη είναι να γίνουν ξεκάθαρα τα index και tag κάθε τιμής του πίνακα a και b. Ο πίνακας a αρχικοποιείται πρώτος, με την εντολή a[32][8]. Στην τιμή a[0][0] αντιστοιχεί λοιπόν η διεύθυνση 0. Στην τιμή a[0][1], η διεύθυνση 8 κ.ο.κ. . Οι τιμές a[0][0], a[0][1], a[0][2], a[0][3] αποτελούν ένα block. Το block αυτό έχει index 0 και tag 0. Γιατί έχει αυτές τις τιμές? Διότι είναι οι τιμές αυτές είναι οι πρώτες τιμές του πίνακα, οπότε θα μπουν πρώτες (tag 0) στην πρώτη θέση (index 0) της cache. Δύο τύποι για να υπολογίσουμε τα index/tag κάθε τιμής του a[A][B] είναι οι εξής:

index = floor(address/32)mod8
tag = floor(address/256) όπου address = A * 64 + B * 8

Τα στοιχεία του πίνακα b[] αποθηκεύονται και αυτά σε block στην cache. Συγκεκριμένα κάθε block αποτελείται από τις τετράδες b[0], b[1], b[2], b[3]. Ο πίνακας b[] αρχικοποιείται μετά τον a[][]. Ο a[][] καταλαμβάνει συνολικά 8 στήλες σε 32 γραμμές, δηλαδή 32*8 = 256 κελιά-τιμές. Επομένως η διεύθυνση του b[0] είναι ίση με 256*8 bytes = 2048. Άρα μπορούμε να υπολογίσουμε τα index/tag κάθε τιμής του b[A] με τους παραπάνω τύπους όπου όμως address = A*8 + 2048. Εδώ αξίζει να σημειωθεί πως κάθε στοιχείο ενός block, έχει το ίδιο tag και index. Έτσι αν ψάχνουμε το index του a[4][5] μπορούμε να υπολογίσουμε το index του a[4][4].

Στον παρακάτω πίνακα φαίνονται οι πρώτες 20 εντολές του προγράμματος, από τις συνολικά 64 που θα εκτελεστούν. Σε κάθε στήλη φαίνεται το στοιχείο που προσπελάται από την μνήμη και τα (index, tag) του. (Και οι 64 γραμμές του προγράμματος) Ακόμη, ο χρωματικός κώδικας δείχνει τα hits, τα compulsory misses και τα conflict misses.

a[0][0] = a[2][0] + a[1][0] + a[0][0] + b[0];	a[2][0]: (4, 0)	a[1][0]: (2, 0)	a[0][0]: (0, 0)	b[0]: (0, 8)	a[0][0]: (0, 0)
a[0][1] = a[2][1] + a[1][1] + a[0][1] + b[1];	a[2][1]: (4, 0)	a[1][1]: (2, 0)	a[0][1]: (0, 0)	b[1]: (0, 8)	a[0][1]: (0, 0)
a[0][2] = a[2][2] + a[1][2] + a[0][2] + b[2];	a[2][2]: (4, 0)	a[1][2]: (2, 0)	a[0][2]: (0, 0)	b[2]: (0, 8)	a[0][2]: (0, 0)
a[0][3] = a[2][3] + a[1][3] + a[0][3] + b[3];	a[2][3]: (4, 0)	a[1][3]: (2, 0)	a[0][3]: (0, 0)	b[3]: (0, 8)	a[0][3]: (0, 0)
a[0][4] = a[2][4] + a[1][4] + a[0][4] + b[4];	a[2][4]: (5, 0)	a[1][4]: (3, 0)	a[0][4]: (1, 0)	b[4]: (1, 8)	a[0][4]: (1, 0)
a[0][5] = a[2][5] + a[1][5] + a[0][5] + b[5];	a[2][5]: (5, 0)	a[1][5]: (3, 0)	a[0][5]: (1, 0)	b[5]: (1, 8)	a[0][5]: (1, 0)
a[0][6] = a[2][6] + a[1][6] + a[0][6] + b[6];	a[2][6]: (5, 0)	a[1][6]: (3, 0)	a[0][6]: (1, 0)	b[6]: (1, 8)	a[0][6]: (1, 0)
a[0][7] = a[2][7] + a[1][7] + a[0][7] + b[7];	a[2][7]: (5, 0)	a[1][7]: (3, 0)	a[0][7]: (1, 0)	b[7]: (1, 8)	a[0][7]: (1, 0)
a[1][0] = a[3][0] + a[2][0] + a[1][0] + b[8];	a[3][0]: (6, 0)	a[2][0]: (4, 0)	a[1][0]: (2, 0)	b[8]: (2, 8)	a[1][0]: (2, 0)
a[1][1] = a[3][1] + a[2][1] + a[1][1] + b[9];	a[3][1]: (6, 0)	a[2][1]: (4, 0)	a[1][1]: (2, 0)	b[9]: (2, 8)	a[1][1]: (2, 0)
a[1][2] = a[3][2] + a[2][2] + a[1][2] + b[10];	a[3][2]: (6, 0)	a[2][2]: (4, 0)	a[1][2]: (2, 0)	b[10]: (2, 8)	a[1][2]: (2, 0)
a[1][3] = a[3][3] + a[2][3] + a[1][3] + b[11];	a[3][3]: (6, 0)	a[2][3]: (4, 0)	a[1][3]: (2, 0)	b[11]: (2, 8)	a[1][3]: (2, 0)

a[1][4] = a[3][4] + a[2][4] + a[1][4] + b[12];	a[3][4]: (7, 0)	a[2][4]: (5, 0)	a[1][4]: (3, 0)	b[12]: (3, 8)	a[1][4]: (3, 0)
a[1][5] = a[3][5] + a[2][5] + a[1][5] + b[13];	a[3][5]: (7, 0)	a[2][5]: (5, 0)	a[1][5]: (3, 0)	b[13]: (3, 8)	a[1][5]: (3, 0)
a[1][6] = a[3][6] + a[2][6] + a[1][6] + b[14];	a[3][6]: (7, 0)	a[2][6]: (5, 0)	a[1][6]: (3, 0)	b[14]: (3, 8)	a[1][6]: (3, 0)
a[1][7] = a[3][7] + a[2][7] + a[1][7] + b[15];	a[3][7]: (7, 0)	a[2][7]: (5, 0)	a[1][7]: (3, 0)	b[15]: (3, 8)	a[1][7]: (3, 0)
a[2][0] = a[4][0] + a[3][0] + a[2][0] + b[16];	a[4][0]: (0, 1)	a[3][0]: (6, 0)	a[2][0]: (4, 0)	b[16]: (4, 8)	a[2][0]: (4, 0)
a[2][1] = a[4][1] + a[3][1] + a[2][1] + b[17];	a[4][1]: (0, 1)	a[3][1]: (6, 0)	a[2][1]: (4, 0)	b[17]: (4, 8)	a[2][1]: (4, 0)
a[2][2] = a[4][2] + a[3][2] + a[2][2] + b[18];	a[4][2]: (0, 1)	a[3][2]: (6, 0)	a[2][2]: (4, 0)	b[18]: (4, 8)	a[2][2]: (4, 0)
a[2][3] = a[4][3] + a[3][3] + a[2][3] + b[19];	a[4][3]: (0, 1)	a[3][3]: (6, 0)	a[2][3]: (4, 0)	b[19]: (4, 8)	a[2][3]: (4, 0)

Ας δούμε τι γίνεται στην cache στην πρώτη εντολή:

a[2][0]: (4, 0) Το block της a[2][0] δεν έχει ξανακληθεί. Η γραμμή 4 της cache είναι κενή, γεμίζει με το block a[2][0] | a[2][1] | a[2][2] | a[2][3]. Έχουμε compulsory miss.

a[1][0]: (2, 0) Το block της a[1][0] δεν έχει ξανακληθεί. Η γραμμή 2 της cache είναι κενή, γεμίζει με το block a[1][0] | a[1][1] | a[1][2] | a[1][3]. Έχουμε compulsory miss.

a[0][0]: (0, 0) Το block της a[0][0] δεν έχει ξανακληθεί. Η γραμμή 0 της cache είναι κενή, γεμίζει με το block a[0][0] | a[0][1] | a[0][2] | a[0][3]. Έχουμε compulsory miss.

b[0]: (0, 8) Το block της b[0] δεν έχει ξανακληθεί. Η γραμμή 0 της cache δεν είναι κενή, αντικαθίσταται με το block b[0] | b[1] | b[2] | b[3]. Έχουμε compulsory miss.

a[0][0]: (0, 0) Το block της a[0][0] έχει ξανακληθεί. Η γραμμή 0 της cache όμως περιέχει το block της b[0], αντικαθιστάται με το block a[0][0] | a[0][1] | a[0][2] | a[0][3]. Έχουμε conflict miss.

Στην δεύτερη γραμμή έχουμε τα πρώτα hit:

a[2][1]: (4, 0) Το block της a[2][1] έχει ξανακληθεί. Η γραμμή 4 της cache περιέχει το block a[2][0] | a[2][1] | a[2][2] | a[2][3]. Έχουμε hit.

a[1][1]: (2, 0) Το block της a[1][1] έχει ξανακληθεί. Η γραμμή 2 της cache περιέχει το block a[1][0] | a[1][1] | a[1][2] | a[1][3]. Έχουμε hit.

a[0][1]: (0, 0) Το block της a[0][1] έχει ξανακληθεί. Η γραμμή 0 της cache περιέχει το block a[0][0] | a[0][1] | a[0][2] | a[0][3]. Έχουμε hit.

b[1]: (0, 8) Το block της b[1] έχει ξανακληθεί. Η γραμμή 0 της cache όμως περιέχει το block της a[0][0], αντικαθίσταται με το block b[0] | b[1] | b[2] | b[3]. Έχουμε conflict miss.

a[0][1]: (0, 0) Το block της a[0][1] έχει ξανακληθεί. Η γραμμή 0 της cache όμως περιέχει το block της b[0], αντικαθιστάται με το block a[0][0] | a[0][1] | a[0][2] | a[0][3]. Έχουμε conflict miss.

Εκτελούμε την ίδια διαδικασία για τις υπόλοιπες 16 γραμμές. Τα hits και τα misses ακολουθούν ένα μοτίβο. Για τα πρώτα 2 ζεύγη τεσσάρων εντολών έχουμε 9 hits και 11 misses (4 compulsory, 7 conflict). Για τα υπόλοιπα 14 ζεύγη, έχουμε 11 hits και 9 misses (2 compulsory, 7 conflict). Άρα συνολικά έχουμε **172hits** και **148misses** (36 compulsory, 112 conflict).

Γ) Έστω ότι η cache είναι write-no-allocate. Αυτό σημαίνει ότι εάν το δεδομένο που θέλουμε να γράψουμε δεν υπάρχει ήδη στην cache (miss), τότε η cache δεν φορτώνει το block από την κύρια μνήμη. Η εγγραφή γίνεται απευθείας στην κύρια μνήμη (write-through ή write-back ανάλογα με την πολιτική εγγραφής στη μνήμη). Πρακτικά, στο δικό μας κώδικα, έχουμε writes όταν θέλουμε να αποθηκεύσουμε στον πίνακα a[]. Αυτό που αλλάζει λοιπόν είναι ότι τα blocks του a[] δεν θα τοποθετούνται στην cache, όταν έχουμε write.

Στον παρακάτω πίνακα φαίνονται οι πρώτες 20 εντολές του προγράμματος, από τις συνολικά 64 που θα εκτελεστούν. Σε κάθε στήλη φαίνεται το στοιχείο που προσπελάται από την μνήμη και τα (index, tag) του. Ακόμη, ο χρωματικός κώδικας δείχνει τα hits, τα compulsory misses, τα conflict misses και τα misses λόγω write-no-allocate.

a[0][0] = a[2][0] + a[1][0] + a[0][0] + b[0];	a[2][0]: (4, 0)	a[1][0]: (2, 0)	a[0][0]: (0, 0)	b[0]: (0, 8)	a[0][0]: (0, 0)
a[0][1] = a[2][1] + a[1][1] + a[0][1] + b[1];	a[2][1]: (4, 0)	a[1][1]: (2, 0)	a[0][1]: (0, 0)	b[1]: (0, 8)	a[0][1]: (0, 0)
a[0][2] = a[2][2] + a[1][2] + a[0][2] + b[2];	a[2][2]: (4, 0)	a[1][2]: (2, 0)	a[0][2]: (0, 0)	b[2]: (0, 8)	a[0][2]: (0, 0)
a[0][3] = a[2][3] + a[1][3] + a[0][3] + b[3];	a[2][3]: (4, 0)	a[1][3]: (2, 0)	a[0][3]: (0, 0)	b[3]: (0, 8)	a[0][3]: (0, 0)
a[0][4] = a[2][4] + a[1][4] + a[0][4] + b[4];	a[2][4]: (5, 0)	a[1][4]: (3, 0)	a[0][4]: (1, 0)	b[4]: (1, 8)	a[0][4]: (1, 0)
a[0][5] = a[2][5] + a[1][5] + a[0][5] + b[5];	a[2][5]: (5, 0)	a[1][5]: (3, 0)	a[0][5]: (1, 0)	b[5]: (1, 8)	a[0][5]: (1, 0)
a[0][6] = a[2][6] + a[1][6] + a[0][6] + b[6];	a[2][6]: (5, 0)	a[1][6]: (3, 0)	a[0][6]: (1, 0)	b[6]: (1, 8)	a[0][6]: (1, 0)
a[0][7] = a[2][7] + a[1][7] + a[0][7] + b[7];	a[2][7]: (5, 0)	a[1][7]: (3, 0)	a[0][7]: (1, 0)	b[7]: (1, 8)	a[0][7]: (1, 0)
a[1][0] = a[3][0] + a[2][0] + a[1][0] + b[8];	a[3][0]: (6, 0)	a[2][0]: (4, 0)	a[1][0]: (2, 0)	b[8]: (2, 8)	a[1][0]: (2, 0)
a[1][1] = a[3][1] + a[2][1] + a[1][1] + b[9];	a[3][1]: (6, 0)	a[2][1]: (4, 0)	a[1][1]: (2, 0)	b[9]: (2, 8)	a[1][1]: (2, 0)
a[1][2] = a[3][2] + a[2][2] + a[1][2] + b[10];	a[3][2]: (6, 0)	a[2][2]: (4, 0)	a[1][2]: (2, 0)	b[10]: (2, 8)	a[1][2]: (2, 0)
a[1][3] = a[3][3] + a[2][3] + a[1][3] + b[11];	a[3][3]: (6, 0)	a[2][3]: (4, 0)	a[1][3]: (2, 0)	b[11]: (2, 8)	a[1][3]: (2, 0)
a[1][4] = a[3][4] + a[2][4] + a[1][4] + b[12];	a[3][4]: (7, 0)	a[2][4]: (5, 0)	a[1][4]: (3, 0)	b[12]: (3, 8)	a[1][4]: (3, 0)
a[1][5] = a[3][5] + a[2][5] + a[1][5] + b[13];	a[3][5]: (7, 0)	a[2][5]: (5, 0)	a[1][5]: (3, 0)	b[13]: (3, 8)	a[1][5]: (3, 0)
a[1][6] = a[3][6] + a[2][6] + a[1][6] + b[14];	a[3][6]: (7, 0)	a[2][6]: (5, 0)	a[1][6]: (3, 0)	b[14]: (3, 8)	a[1][6]: (3, 0)
a[1][7] = a[3][7] + a[2][7] + a[1][7] + b[15];	a[3][7]: (7, 0)	a[2][7]: (5, 0)	a[1][7]: (3, 0)	b[15]: (3, 8)	a[1][7]: (3, 0)
a[2][0] = a[4][0] + a[3][0] + a[2][0] + b[16];	a[4][0]: (0, 1)	a[3][0]: (6, 0)	a[2][0]: (4, 0)	b[16]: (4, 8)	a[2][0]: (4, 0)
a[2][1] = a[4][1] + a[3][1] + a[2][1] + b[17];	a[4][1]: (0, 1)	a[3][1]: (6, 0)	a[2][1]: (4, 0)	b[17]: (4, 8)	a[2][1]: (4, 0)
a[2][2] = a[4][2] + a[3][2] + a[2][2] + b[18];	a[4][2]: (0, 1)	a[3][2]: (6, 0)	a[2][2]: (4, 0)	b[18]: (4, 8)	a[2][2]: (4, 0)
a[2][3] = a[4][3] + a[3][3] + a[2][3] + b[19];	a[4][3]: (0, 1)	a[3][3]: (6, 0)	a[2][3]: (4, 0)	b[19]: (4, 8)	a[2][3]: (4, 0)

Ας δούμε τι γίνεται στην cache στην πρώτη εντολή:

a[2][0]: (4, 0) Το block της a[2][0] δεν έχει ξανακληθεί. Η γραμμή 4 της cache είναι κενή, γεμίζει με το block a[2][0] | a[2][1] | a[2][2] | a[2][3]. Έχουμε compulsory miss.

a[1][0]: (2, 0) Το block της a[1][0] δεν έχει ξανακληθεί. Η γραμμή 2 της cache είναι κενή, γεμίζει με το block a[1][0] | a[1][1] | a[1][2] | a[1][3]. Έχουμε compulsory miss.

a[0][0]: (0, 0) Το block της a[0][0] δεν έχει ξανακληθεί. Η γραμμή 0 της cache είναι κενή, γεμίζει με το block a[0][0] | a[0][1] | a[0][2] | a[0][3]. Έχουμε compulsory miss.

b[0]: (0, 8) Το block της b[0] δεν έχει ξανακληθεί. Η γραμμή 0 της cache δεν είναι κενή, αντικαθίσταται με το block b[0] | b[1] | b[2] | b[3]. Έχουμε compulsory miss.

a[0][0]: (0, 0) Το block της a[0][0] δεν υπάρχει στην cache. Λόγω της write-no-allocate γίνεται write απευθείας στην μνήμη, χωρίς να αλλάξει η cache. Έχουμε write-no-allocate miss.

Στην δεύτερη γραμμή έχουμε τα πρώτα hit:

a[2][1]: (4, 0) Το block της a[2][1] έχει ξανακληθεί. Η γραμμή 4 της cache περιέχει το block a[2][0] | a[2][1] | a[2][2] | a[2][3]. Έχουμε hit.

a[1][1]: (2, 0) Το block της a[1][1] έχει ξανακληθεί. Η γραμμή 2 της cache περιέχει το block a[1][0] | a[1][1] | a[1][2] | a[1][3]. Έχουμε hit.

a[0][1]: (0, 0) Το block της a[0][1] έχει ξανακληθεί. Η γραμμή 0 της cache περιέχει το block της b[0], αντικαθίσταται με το block a[0][0] | a[0][1] | a[0][2] | a[0][3]. Έχουμε conflict miss.

b[1]: (0, 8) Το block της b[1] έχει ξανακληθεί. Η γραμμή 0 της cache όμως περιέχει το block της a[0][0], αντικαθίσταται με το block b[0] | b[1] | b[2] | b[3]. Έχουμε conflict miss.

a[0][1]: (0, 0) Το block της a[0][1] δεν υπάρχει στην cache. Λόγω της write-no-allocate γίνεται write απευθείας στην μνήμη, χωρίς να αλλάξει η cache. Έχουμε write-no-allocate miss.

Εκτελούμε την ίδια διαδικασία για τις υπόλοιπες 16 γραμμές. Τα hits και τα misses ακολουθούν ένα μοτίβο. Για τα πρώτα 2 σετ τεσσάρων εντολών έχουμε 6 hits και 14 misses (4 compulsory, 6 conflict, 4 write-no-allocate). Για τα υπόλοιπα 14 σετ, έχουμε 8 hits και 12 misses (2 compulsory, 6 conflict, 4 write-no-allocate). Άρα συνολικά έχουμε **124hits** και **196misses** (36 compulsory, 96 conflict , 64 write-no-allocate).