ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

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

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Εξετάσεις Ιουλίου 2010
Διάρκεια 2:45' ώρες

Οι εξετάσεις θα πραγματοποιηθούν ΧΩΡΙΣ την παρουσία βιβλίων, βοηθημάτων ή άλλου είδους σημειώσεων. Το μόνο που επιτρέπεται να έχετε μαζί σας είναι ένα φύλλο Α4 στο οποίο μπορείτε να έχετε γράψει ό,τι έχετε κρίνει πιο σημαντικό για το μάθημα και θέλετε να το έχετε ως βοήθημά σας. Απαγορεύεται η ανταλλαγή οποιουδήποτε αντικειμένου κατά την ώρα της εξέτασης, ούτε και των φύλλων Α4 που είναι ατομικά.

Θέμα 1ο (15%)

Α. Θεωρήστε ένα πρόγραμμα το οποίο αποτελείται από δύο κατηγορίες εντολών. Στην $1^η$ κατηγορία (*γρήγορες*) οι εντολές δεν εξαρτώνται από οποιεσδήποτε άλλες και χρειάζονται 1 κύκλο για να εκτελεστούν. Στη $2^η$ κατηγορία (*αργές*) οι εντολές επίσης είναι ανεξάρτητες από όλες τις υπόλοιπες, αλλά χρειάζονται 20 κύκλους για την εκτέλεσή τους.
Υποθέστε ότι έχετε έναν επεξεργαστή με δυναμική δρομολόγηση εντολών ο οποίος υλοποιεί τον αλγόριθμο Tomasulo. Τα στάδια issue, write result και commit διαρκούν 1 κύκλο το καθένα, ενώ ο επεξεργαστής διαθέτει re-order buffer 64 θέσεων.

(i) Ποιο είναι το καλύτερο CPI που μπορεί να επιτύχει αυτός ο επεξεργαστής, αν το πρόγραμμα είναι δομημένο σε ομάδες των 20 εντολών εκ των οποίων οι 19 πρώτες είναι *γρήγορες* ενώ η τελευταία *αργή*; Υποθέστε ότι υπάρχει μεγάλο πλήθος τέτοιων ομάδων (η εκτέλεση δηλαδή περιλαμβάνει 19 *γρήγορες*, 1 *αργή*, 19 *γρήγορες*, 1 *αργή*, κ.ο.κ.).

CPI=1.0 (σε κατάσταση «ισορροπίας»):

IS	EX	WR	CMT
1	2	3	4
2	3	4	5
3	4	5	6
…			
19	20	21	22
20	**21-40**	**41**	**42**
21	22	23	43
22	23	24	44
…			
39	40	41	61
40	**41-60**	**61**	**62**
41	42	43	63
…			
59	60	61	81
60	**61-80**	**81**	**82**

(ii) Απαντήστε όπως και στο ερώτημα (i), με τη διαφορά ότι οι *αργές* εντολές απαιτούν 100 κύκλους, και το πρόγραμμα τώρα είναι δομημένο σε ομάδες των 40 εντολών, εκ των οποίων οι 39 πρώτες είναι *γρήγορες* ενώ η τελευταία *αργή*.

Id	ROB usage	IS	EX	WR	CMT
1:	1	1	2	3	4
2:	2	2	3	4	5
3:	3	3	4	5	6
4:	4	4	5	6	7
5:	4	5	6	7	8
6:	4	6	7	8	9
…					
39:	4	39	40	41	42
40:	**4**	**40**	**41-140**	**141**	**142**
1:	4	41	42	43	143
2:	4	42	43	44	144
3:	4	43	44	45	145
4:	5	44	45	46	146
5:	6	45	46	47	147
…					
39:	40	79	80	81	181
40:	**41**	**80**	**81-180**	**181**	**182**
1:	42	81	82	83	183
2:	43	82	83	84	184
…					
22:	63	102	103	104	204
23:	64	103	104	105	205
~~*24:*~~		~~*104*~~	*ROB full (24η εντολή από την 40-άδα)*		
24:	64	143	144	145	206
25:	64	144	145	146	207
…					
39:	64	158	159	160	221
40:	**64**	**159**	**160-259**	**260**	**261**

Το pattern των 2 τελευταίων 40-άδων θα επαναλαμβάνεται συνεχώς, δηλαδή ανά 2 επαναλήψεις θα έχουμε stalls λόγω γεμάτου ROB, ως αποτέλεσμα της καθυστέρησης της αργής εντολής.
CPI = (261-143+1)/80 = 1.4875

Χοντρικά, μπορούμε να πούμε το εξής:
Αφού εκτελεστεί η 1η αργή εντολή ο ROB θα είναι συνεχώς γεμάτος. Όταν όμως μια αργή εντολή στην κεφαλή του ROB stall-άρει, μία άλλη εντολή θα εκτελείται επίσης. Αυτή η εντολή δε θα stall-άρει. Δηλαδή θα έχουμε 100-63 stalls ανά 2 αργές εντολές, δηλ. CPI = 1 + (100-63)/2*40 = 1.4625

Β. Δίνεται ο παρακάτω κώδικας γραμμένος σε C :

```
int i, stride;
```

```
long A[N], sum=0;
for(i = 0; i < 1000; i++)
      for(j = 0; j < N; j += stride)
            sum += A[j];
return sum;
```

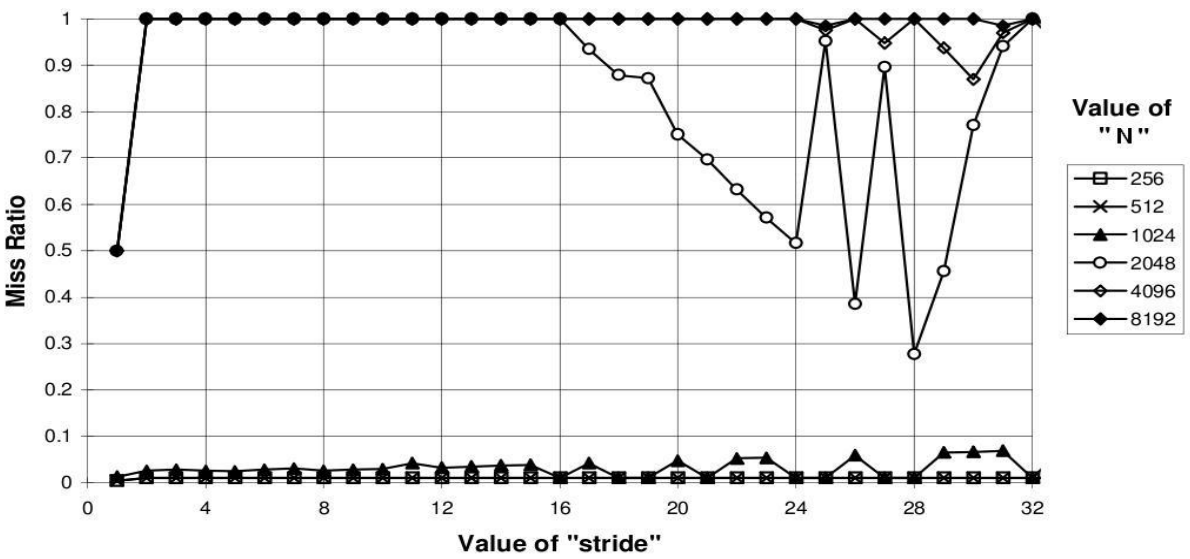
Ο κώδικας έχει μεταγλωττιστεί με τέτοιο τρόπο ώστε οι βαθμωτές μεταβλητές να αποθηκεύονται σε καταχωρητές ενώ το overhead των εντολών ελέγχου του loop (reads και writes) έχει απομακρυνθεί. Ο κώδικας εκτελέστηκε σε ένα προσομοιωμένο επεξεργαστή με μια direct-mapped cache για διαφορετικές τιμές του N και του stride. Τα αποτελέσματα για το Miss Rate της cache δίνονται στην παρακάτω γραφική. Θεωρήστε ότι δεν υπάρχει prefetching.

(i) Ποιο είναι το μέγεθος της cache σε KB; Δικαιολογήστε την απάντηση σας.

Size = 1024 * 8 = 8KB

(ii) Ποιο είναι το μέγεθος του cache bock της cache σε bytes; Δικαιολογήστε την απάντηση σας.

Για stride = 1, οι μεγάλοι πίνακες έχουν 50% miss rate, γεγονός που υποδεικνύει ότι εκμεταλλεύονται τη χωρική τοπικότητα. Αντίθετα για stride > 2, έχουν 100% miss rate, επομένως το block περιέχει 2 στοιχεία του πίνακα. Άρα block size = 2 * 8 = 16 bytes.



Θέμα 2ο (25%)

A. Δίνεται ο παρακάτω κώδικας :

```
0x0038      L3:   LD          R1, 0(R4)
0X003C            LD          R2, 0(R5)
0X0040            DSUBUI      R3, R1, #2
0X0044            BNEZ        R3, L1
0X0048            DADD        R1, R0, R0
0X004C      L1:   DUSBUI      R3, R2, #2
0X0050            BEQZ        R3, L2
0X0054            DADD        R2, R0, R0
0X0058      L2:   DSUBUI      R3, R1, R2
0X005C            BEQZ        R3, L3
```

Υποθέστε τη χρήση ενός πίνακα με n-bit predictors. Το παραπάνω loop έχει ήδη εκτελεστεί 12 φορές. Κατά την εκτέλεση βρέθηκε ότι οι εντολές άλματος που βρίσκονται στις θέσεις 0x0044 και 0x0050

εκτελέστηκαν (ήταν Taken) στο 50% των περιπτώσεων. Υποθέστε ότι αυτή τη στιγμή, ο επεξεργαστής βρίσκεται στο 0x0038 και είναι έτοιμος να εκτελέσει για 13η φορά το loop.
(i) Με βάση τον παραπάνω κώδικα πόσες θέσεις πρέπει να έχει ο πίνακας των predictors, ώστε η πιθανότητα εμφάνισης aliases (ψευδωνύμων) να είναι μηδενική;

min index length = 4 άρα minimum entries = 16

(ii) Αν πριν την πρώτη εκτέλεση του loop, οι προβλέπτες είχαν αρχικοποιηθεί στο 0, ποια μπορεί να είναι αυτή τη στιγμή η τιμή τους για τις 2 εντολές άλματος στα σημεία 0x0044 και 0x0050;

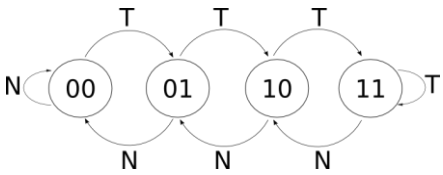
$0 \leq$ value ≤ 6

(iii) Υποθέστε ότι ο ίδιος κώδικας εκτελείται και από ένα σύστημα που χρησιμοποιεί ένα (3, n) global history predictor. Στο σημείο αυτό της εκτέλεσης (πριν την 13η εκτέλεση του loop) δώστε τις πιθανές επιτρεπτές τιμές του καταχωρητή που αποθηκεύει το history. Υποθέστε ότι το αποτέλεσμα της πιο πρόσφατης εντολής άλματος βρίσκεται στο λιγότερο σημαντικό bit.

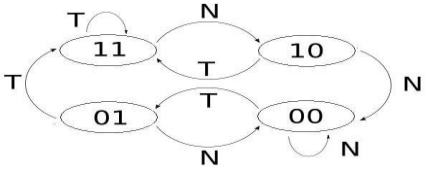
0x1, x01

Β. Δίνονται τα FSMs των εξής 2-bit predictors.

predA



predB



Υποθέτοντας ότι και οι 2 είναι αρχικοποιημένοι στο 00, δώστε μια αλληλουχία των αποτελεσμάτων 4 εντολών άλματος υπό συνθήκη (π.χ. T, NT, T, T), όπου ο predA θα είναι καλύτερος από τον predB.

T,T,NT,NT

Γ. Θεωρήστε ένα loop το οποίο εκτελείται αρκετές φορές σε ένα πρόγραμμα. Κάθε εκτέλεση του loop, περιλαμβάνει 10 επαναλήψεις. Κάθε επανάληψη περιέχει 4 εντολές άλματος υπό συνθήκη, τα αποτελέσματα των οποίων φαίνονται στον παρακάτω πίνακα :

	Επανάληψη									
	1	2	3	4	5	6	7	8	9	10
Branch1	N	N	N	N	N	N	N	N	N	T
Branch2	T	T	T	T	T	T	T	T	T	
Branch3	T	T	N	T	N	T	N	T	N	
Branch4	N	N	T	N	T	N	T	N	T	

Στην 10η επανάληψη το branch1 είναι T, η εκτέλεση του loop σταματά και για αυτό τα υπόλοιπα branches δεν εκτελούνται. Από τους dynamic predictors που είδαμε στο μάθημα διαλέξτε τον φθηνότερο που θα δώσει το καλύτερο ποσοστό πρόβλεψης για τις εξής εντολές :

(i) Branch1

Είναι σχεδόν πάντα NT, άρα ο 2-bit predictor θα έκανε, αφού θα είχαμε 1 mispred κατά την έξοδο. Με τον 1-bit αντίθετα θα είχαμε 2 mispreds, στο 1ο και τελευταίο branch.

(ii) Branch2

Πάντα T, ίδιο αποτέλεσμα με όλους τους predictors, άρα επιλέγουμε τον 1-bit

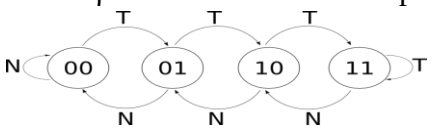
(iii) Branch 4

Πάντα αντίθετος του branch3, άρα (1,1) correlating

Θέμα 3ο (30%)

Δίνεται αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το pipeline του επεξεργαστή περιέχει τα στάδια Issue (IS), Execute (EX), Write Result (WR) και Commit (CMT), αγνοούμε δηλαδή τα IF και ID. Ισχύουν επίσης τα ακόλουθα :

- Τα IS, WR, CMT απαιτούν 1 κύκλο.
- Το σύστημα περιέχει *περιορισμένο* αριθμό από reservation stations (RS). Συγκεκριμένα, περιέχει 2 RS για προσθέσεις/αφαιρέσεις και 1 RS για πολλαπλασιασμούς διαιρέσεις floating point αριθμών. Αντίστοιχος αριθμός RS περιλαμβάνεται και για integer αριθμούς.
- Το σύστημα περιλαμβάνει 1 *non-pipelined* integer functional unit. Σε αυτό εκτελούνται οι αριθμητικές εντολές για ακεραίους, οι οποίες διαρκούν 1 κύκλο.
- Το σύστημα περιλαμβάνει 2 floating point functional units, ένα για ADDD / SUBD και ένα για MULD / DIVD. Το πρώτο functional unit είναι *πλήρως pipelined* στη συχνότητα του επεξεργαστή, ενώ το δεύτερο είναι *non-pipelined*. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 5 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 20 κύκλους.
- Για τις εντολές αναφοράς στη μνήμη, στο στάδιο EX γίνεται τόσο ο υπολογισμός της διεύθυνσης αναφοράς όσο και η προσπέλαση στη μνήμη. Το σύστημα περιλαμβάνει ένα Load και ένα Store Queue, τα οποία διαθέτουν 4 θέσεις. Οι εντολές αναφοράς στη μνήμη διαρκούν 6 κύκλους και χρησιμοποιούν ένα ξεχωριστό functional unit για τον υπολογισμό της διεύθυνσης.
- Οι εντολές διακλάδωσης υπό συνθήκη έχουν τη μορφή : *συνθήκη* <*reg1*>, <*reg2*>, <*offset*> και χρησιμοποιούν τα κατάλληλα RS και FU για αφαίρεση, προκειμένου να υπολογίσουν αν ισχύει η συνθήκη μεταξύ των 2 καταχωρητών που δίνονται ως όρισμα.
- Ο ROB έχει 8 θέσεις.
- Το σύστημα περιλαμβάνει 1 CDB. Σε περίπτωση που παραπάνω από μια εντολές θέλουν να το χρησιμοποιήσουν, τότε *προτεραιότητα αποκτά η ''παλαιότερη'' εντολή.*
- Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί έναν *(2,2) global history predictor* με συνολικά *16 εγγραφές (entries).* Ο καταχωρητής που αποθηκεύει το αποτέλεσμα των προηγούμενων εντολών άλματος είναι αρχικοποιημένος στο 00, ενώ και οι 2-bit predictors είναι αρχικοποιημένοι στο 00. Δίνεται το παρακάτω FSM του 2-bit predictor :



- Η πρόβλεψη για μια εντολή διακλάδωσης υπό συνθήκη γίνεται *ταυτόχρονα* με τη δρομολόγηση της εντολής. Ο έλεγχος της πρόβλεψης γίνεται αμέσως μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR (κύκλος k). Σε περίπτωση σφάλματος, σταματά η εκτέλεση των λάθος εντολών και στον επόμενο κύκλο (κύκλος k+1) δρομολογείται η σωστή εντολή.

Δίνεται ο παρακάτω κώδικας :

```
0x00880000        LOOP:       LD    F0, 0(R1)
0x00880004                    ADDD  F0, F1, F0
0x00880008                    SUBD  F2, F3, F2
0x0088000C                    DBEQ  F2, F4, L1
0x00880010                    DIVD  F4, F0, F5
```

```
0x00880014                  ADDI  R2, R2, #8
0x00880018                  LD    F0, 0(R2)
0x0088001C                  ADDD  F0, F0, F4
0x00880020                  ADDD  F0, F0, F7
0x00880024        L1:       MULD  F6, F0, F3
0x00880028                  ST    F6, 0(R1)
0x0088002C                  ADDI  R1, R1, #8
0x00880030                  BNEQ  R1, R4, LOOP
0x00880034                  ADDI  R1, R1, R4
0x00880038                  ADDI  R2, R4, R5
0x0088003C                  ST    R1, 0(R2)
```

Δίνεται ότι το αποτέλεσμα της εντολής άλματος στο 0x0088000C θα είναι πάντα T, καθώς και οι αρχικές τιμές 0 για τον R1 και 16 για τον R4. Εκτελέστε τον παραπάνω κώδικα και δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών σε έναν πίνακα όπως ο παρακάτω :

OP	IS	EX	WR	CMT	Σχόλιο
L.D F0, 0(R1)	1	2-7	8	9	
ADDD F0,F1,F0	2	9-13	14	15	
SUBD F2,F3,F2	3	4-8	9	16	
DBEQ F2,F4,L1	10	11-15	16	17	
DIVD F4,F0,F5	11	15-			
ADDI R2,R2,#8	12	13-13	15		
LD F0,0(R2)	13	16-	-		
ADDD F0,F0,F4	15	-			
ADDD F0,F0,F7	-				
MULD F6,F0,F3	17	18-37	38	39	
ST F6,0(R1)	18	39-44	45	46	
ADDI R1,R1,#8	19	20-20	21	47	
BNEQ R1,R4,LOOP	20	22-22	23	48	
ADDI R1,R1,R4	22	23-23	-		
ADDI R2,R4,R5	-				
LD F0,0(R1)	24	25-30	31	49	
ADDD F0,F1,F0	25	32-36	37	50	
SUBD F2,F3,F2	26	27-31	32	51	
DBEQ F2,F4,L1	33	34-38	39	52	
DIVD F4,F0,F5	-	-			
MULD F6,F0,F3	40	41-60	61	62	
ST F6,0(R1)	47	62-67	68	69	
ADDI R1,R1,#8	48	49-49	50	70	
BNEQ R1,R4,LOOP	49	51-51	52	71	
ADDI R1,R1,R4	51	52-52	53	72	
ADDI R2,R4,R5	53	54-54	55	73	
ST R1,0(R2)	54	56-61	62	74	

Στο πεδίο "Σχόλιο" δικαιολογήστε τυχόν καθυστερήσεις μεταξύ IS-EX, EX-WR και WR-CMT καθώς και ακυρώσεις εντολών.

Θέμα 4ο (20%)

Θεωρούμε το ακόλουθο κομμάτι κώδικα:

```
float A[10][256];
int i,j;
for(i=1; i<9; i++)
      for(j=0; j<256; j++)
            A[i][j] = A[i-1][j] + A[i+1][j];
```

Κάνουμε τις εξής υποθέσεις:

1. Το πρόγραμμα εκτελείται σε έναν επεξεργαστή με μόνο ένα επίπεδο κρυφής μνήμης δεδομένων, η οποία αρχικά είναι άδεια. Η κρυφή μνήμη είναι direct mappped, write-allocate, και έχει μέγεθος 2KB. Το μέγεθος του block είναι 32 bytes, ενώ το μέγεθος ενός float είναι 4 bytes.
2. Η σειρά με την οποία γίνονται οι *αναγνώσεις* είναι αυτή με την οποία φαίνονται στο πρόγραμμα.

Α. Βρείτε το συνολικό ποσοστό αστοχίας (miss rate) για τις αναφορές που γίνονται στην μνήμη.

Η cache περιέχει 2KB/32 = 64 blocks.
1 γραμμή του πίνακα εκτείνεται σε 256*4/32 = 32 blocks.
Άρα ανά 2 γραμμές θα έχω conflict.

Οι αναφορές που γίνονται είναι οι εξής:

A00 A20 A10 m m m comp comp comp
A01 A21 A11 m m h confl confl
A02 A22 A12 m m h confl confl
…
A07 A27 A17 m m h confl confl
A08 A28 A18 m m m comp comp comp
…
---------- 224 hits για i=1
A10 A30 A20 h m h comp
A11 A31 A21 m m h confl confl
…
A17 A37 A27 m m h confl confl
A18 A38 A28 h m h comp
…
288 hits για i=2,…,8

Άρα σύνολο 224+7*288 = 2240 hits ή (6144-2240)/6144 = 63.5% miss rate

Β. Ποιες από τις παρακάτω τεχνικές βελτιστοποίησης θα ακολουθούσατε προκειμένου να βελτιώσετε την απόδοση; Δικαιολογήστε την επιλογή ή μη-επιλογή κάθε τεχνικής. Για αυτές που επιλέξατε, υπολογίστε το νέο ποσοστό αστοχίας.

1. loop distribution

```
for(i=1;  i<9; i++)
   for(j=0;  j<256; j++)
      A[i][j]  = A[i-1][j];
```

A00 A10 m m
A01 A11 h h
A02 A12 h h
…
A07 A17 h h
…

A10 A20 h m
A11 A21 h h
A12 A22 h h
…
A17 A27 h h
…
2*32 + 7*32*1 misses

```
for(i=1; i<9; i++)
   for(j=0; j<256; j++)
      A[i][j] += A[i+1][j];
```

A20 A10 A10 m m h
A21 A11 A11 h h h
A22 A12 A12 h h h
…
A27 A17 h h h
…

A30 A20 A20 m h h
A31 A21 A21 h h h
A32 A22 A22 h h h
…
A37 A27 A27 h h h
…
2*32 + 7*32*1 misses

Miss rate = 576/(4096+6144) = 5.6%

2. αύξηση associativity σε 2-way (με διατήρηση της χωρητικότητας της cache)
   Ανά 1 γραμμή θα έχω mapping στο ίδιο set
   A00 A20 A10 m m m
   A01 A21 A11 m m m
   A02 A22 A12 m m m
   …
   A07 A27 A17 m m m
   …

   A10 A30 A20 h m m
   A11 A31 A21 m m m
   …
   A18 A38 A28 h m m
   A19 A39 A29 m m m

   32 hits για i=2,…,8, άρα 7*32 = 224 hits ή (6144-224)/6144 = 96.3% miss rate

3. hardware prefetching

A00 A20 A10 m m m comp comp comp
A01 A21 A11 m m h confl confl
A02 A22 A12 m m h confl confl
…
A07 A27 A17 m m h confl confl
A08 A28 A18 m **h h** **confl** ~~comp~~
…

A10 A30 A20 m **h h** **confl** ~~comp~~
A11 A31 A21 m m h confl confl
…
A17 A37 A27 m m h confl confl
A18 A38 A28 m **h h** **confl** ~~comp~~
…

Περίπου 9 hits ανά 8 επαναλήψεις, δηλαδή miss rate = (24-9)/24 = 62.5%