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 2008
Διάρκεια 2,5 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 (2 μονάδες)

Α. Είστε υπεύθυνοι σχεδιασμού ενός out-of-order επεξεργαστή με δυναμική δρομολόγηση (dynamic scheduling) και υποθετική εκτέλεση (speculative execution) εντολών. Ο επεξεργαστής που σχεδιάσατε έχει 8 Integer Functional Units (IUs), 4 Floating Point Units (FPUs), 256KB on-chip caches, 4 reservation stations για integer και 2 για floating point. Ο Reorder Buffer έχει 8 θέσεις, ενώ το pipeline αποτελείται από 25 στάδια. Τέλος, στο σχέδιο σας έχετε συμπεριλάβει ένα μικρό 2-bit branch predictor, ο οποίος αποδίδει *σχετικά καλά*.

Οι εφαρμογές για τις οποίες θα χρησιμοποιηθεί ο επεξεργαστής έχουν λίγο κώδικα και επεξεργάζονται δεδομένα μεγέθους περίπου 64KB. Την περισσότερη ώρα οι εφαρμογές εκτελούν loops, των οποίων οι επαναλήψεις είναι μεταξύ τους ανεξάρτητες, ενώ σε κάθε μια από αυτές υπάρχει περιορισμένη δυνατότητα εκμετάλλευσης του ILP.

Μετά την αξιολόγηση του σχεδίου, σας ενημερώνουν ότι υπάρχουν ακόμα διαθέσιμα transistors τα οποία θα μπορούσατε να χρησιμοποιήσετε με κάποιον από τους παρακάτω τρόπους :

(i) Βελτίωση της ακρίβειας πρόβλεψης του branch predictor.

Ο predictor αποδίδει "σχετικά καλά" και γνωρίζουμε ότι ο κώδικας μας περιέχει πολλά branches. Ταυτόχρονα, το pipeline είναι αρκετά μεγάλο καθιστώντας τις λάθος προβλέψεις πολύ ακριβές. Θα επιλέγαμε επομένως να αυξήσουμε την ακρίβεια του predictor (και μέσω αυτής την απόδοση του συστήματος).

(ii) Αύξηση του μεγέθους της cache.

Δεν θα το επιλέγαμε. Το σύστημα έχει ήδη αρκέτη μνήμη για να χωρέσει το working set των εφαρμογών, επομένως δε θα βλέπαμε σημαντική μείωση των miss rates.

(iii) Προσθήκη περισσότερων reservation stations.

Θα επιλέγαμε να προσθέσουμε περισσότερα RS, καθώς αυτό θα σήμαινε ότι το σύστημα θα είχε ένα μεγαλύτερο window στο οποίο θα έψαχνε για εντολές που θα μπορούσαν να εκτελεστούν παράλληλα. Θα

έδινε πχ. τη δυνατότητα στο σύστημα μας να εκμεταλλευτεί τυχόν παραλληλισμό μεταξύ διαφορετικών επαναλήψεων ενός loop.

(iv) Προσθήκη περισσότερων IUs και FPUs.

Δεν θα το επιλέγαμε, τουλάχιστον σε σύγκριση με τις υπόλοιπες επιλογές. Αν ο επεξεργαστής μας δεν έχει τη δυνατότητα να ανακαλύψει περισσότερο ILP τότε δεν έχει νόημα να προσθέσουμε περισσότερα execution units, καθώς θα μείνουν ανενεργά.

(v) Προσθήκη περισσότερων θέσεων στον ROB.

Ο ROB έχει πολύ λίγες θέσεις. Θα επιλέγαμε λοιπόν να τον μεγαλώσουμε, καθώς θα βοηθούσε να κρυφτούν οι μεγάλες καθυστερήσεις (λόγω long latency instructions) και θα έδινε τη δυνατότητα στον επεξεργαστή να ψάξει για παραλληλισμό σε ένα μεγαλύτερο window εντολών (όμοια με την (iii)).

Β. Για ποιο λόγο, οι αρχιτεκτονικές που χρησιμοποιούν τον αλγόριθμο Tomasulo χρειάζονται κάποιο μηχανισμό πρόβλεψης διακλάδωσης;

Ο αλγόριθμος Tomasulo χρησιμοποιείται για την υλοποίηση out-of-order εκτέλεσης εντολών. Τα συστήματα αυτά επιτυγχάνουν βελτίωση της απόδοσης τους εκτελώντας εντολές οι οποίες ακολουθούν άλλες, οι οποίες όμως είναι blocked περιμένοντας το αποτέλεσμα κάποιων προηγούμενων εντολών από τις οποίες εξαρτώνται. Για να γίνει εφικτό αυτό, οι αρχιτεκτονικές αυτές απαιτείται να δρομολογούν πολλές εντολές, οι οποίες βρίσκονται πιο μπροστά στο instruction stream. Όταν λοιπόν συναντούν ένα branch απαιτείται ένας μηχανισμός πρόβλεψης, ώστε να ξέρουν ποιες εντολές πρέπει να δρομολογήσουν στη συνέχεια. Διαφορετικά, θα πρέπει να περιμένουν το αποτέλεσμα του branch χάνοντας έτσι το κέρδος της ooo execution.

Γ. Υποθέστε ότι έχετε έναν επεξεργαστή ο οποίος υλοποιεί τον αλγόριθμο Tomasulo με τη βοήθεια ROB και ο οποίος έχει άπειρους επεξεργαστικούς πόρους (π.χ. άπειρα reservation units και άπειρα execution units). Υποθέστε επίσης ότι ο μηχανισμός πρόβλεψης διακλάδωσης είναι ιδανικός και πως δεν υπάρχουν dependencies μεταξύ των εντολών που εκτελούνται. Τέλος, το Fetch στάδιο μπορεί να φέρνει 12 εντολές ανά κύκλο, ενώ για τα υπόλοιπα στάδια δεν υπάρχουν περιορισμοί (π.χ. αποκωδικοποίηση άπειρων εντολών ανά κύκλο).

(i) Αν όλες οι εντολές απαιτούν 1 κύκλο εκτέλεσης και ο ROB έχει άπειρες θέσεις ποιος είναι ο μέσος ρυθμός εκτέλεσης εντολών ανά κύκλο (IPC) του επεξεργαστή;

Το IPC περιορίζεται μόνο από το ρυθμό με τον οποίο οι εντολές γίνονται fetched. Δεν υπάρχουν data dependencies ή branch mipredictions ενώ έχουμε και άπειρους επεξεργαστικούς πόρους. Επομένως

$$IPC = 12$$

(ii) Αν για το σύστημα του ερωτήματος (i) έχετε κάθε 48 εντολές ένα load το οποίο αποτυγχάνει και το miss χρειάζεται 500 κύκλους για να ικανοποιηθεί, ποιο θα είναι το IPC του επεξεργαστή ;

Ο ROB είναι άπειρος, δηλαδή συνεχίζουμε να κάνουμε fetch και issue 12 εντολές ανά κύκλο. Το κάθε miss χρειάζεται 500 κύκλους από το fetch μέχρι το commit, αλλά αυτές οι καθυστερήσεις "κρύβονται" μέσω του ROB και το average throughput παραμένει 12.

(iii) Αν στο ερώτημα (ii) περιορίσετε τις θέσεις του ROB από άπειρες σε 48, το IPC του επεξεργαστή μεταβάλλεται; Αν το IPC είναι μικρότερο από 12, ποιο το μέγεθος του ROB που απαιτείται ώστε IPC = 12;

Προφανώς το IPC θα είναι μικρότερο του 12, μιας και λόγω του μεγέθους του ROB θα έχουμε stalls. Έστω ότι στον 1ο κύκλο έχουμε ένα load που προκαλεί miss. Το σύστημα θα συνεχίσει να φέρνει και να δρομολογεί εντολές μέχρι να γεμίσει ο ROB, δηλαδή 47 ακόμα εντολές θα γίνουν issued και μετά θα έχουμε stall. Στον κύκλο 500 το load που βρίσκεται στην κορυφή του ROB θα είναι έτοιμο να γίνει commit, μαζί βέβαια με τις υπόλοιπες 47 εντολές. Στο σημείο αυτό γίνεται issued το επόμενο load που προκαλεί miss και ο κύκλος επαναλαμβάνεται. Δηλαδή κάθε 500 κύκλους μπορούμε να κάνουμε commit 48 εντολές κι άρα

$$IPC = 48 / 500$$

*Για να έχουμε IPC = 12, πρέπει το σύστημα να μπορεί να διατηρεί ρυθμό εκτέλεσης 12 εντολές ανά κύκλο κατά τη διάρκεια των 500 κύκλων όπου περιμένουμε να ικανοποιηθεί το load (οι εντολές αυτές βέβαια θα γίνονται commit μαζί με το load). Επομένως απαιτείται το ROB να έχει τουλάχιστον 12 * 500 = 6000 θέσεις.*

Θέμα 2ο (2 μονάδες)

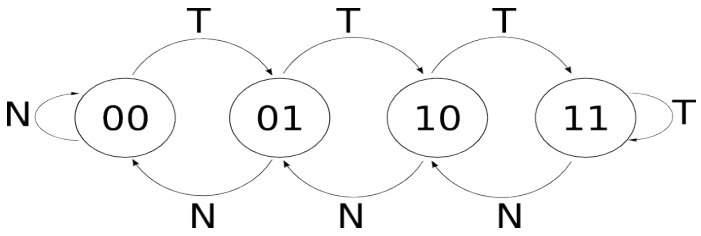
Α. Υποθέστε εντολές διακλάδωσης με τις παρακάτω συμπεριφορές :

(i) T, T, T, T, T, T, *N*, T, T, T, T, T, T, T, T, *N*, T, T, T, T, T, T, T, T, *N*, T, T, T, T, T, T, T, T, T, *N*
(ii) T, T, T, T, T, *N*, *N*, *N*, *N*, *N*, *N*, *N*, *N*, T, T, T, T, T, T, T, T, T, *N*, *N*, *N*, *N*, *N*, T, T, T, T, T, T, T
(iii) T, T, T, T, *N*, T, T, T, T, T, T, *N*, T, T, T, T, T, T, T, *N*, *N*, *N*, *N*, *N*, T, *N*, *N*, *N*, *N*, *N*, *N*, T, *N*, *N*, *N*

όπου T taken και N not-taken. Σας δίνονται επίσης οι επόμενοι μηχανισμοί πρόβλεψης :

1. static Taken
2. n-bit, όπου n = 1, 2

Ποιο μηχανισμό θα προτείνατε για κάθε μια από τις παραπάνω εντολές διακλάδωσης και γιατί; (Προφανώς αν 2 μηχανισμοί αποδίδουν το ίδιο καλά, τότε πρέπει να λάβετε υπόψη το κόστος του καθενός). Ο 2-bit predictor χρησιμοποιεί το παρακάτω FSM (finite state machine):



(i) *static Taken : 4 miss-predictions*
1-bit : 7 miss-predictions (2 λάθη σε κάθε T,N,T)
2-bit : 4 miss-predictions (1 λάθος σε κάθε T,N,T)

Επιλέγεται ο static Taken λόγω μικρότερου κόστους από τον 2-bit.

(ii) *static Taken : 13 miss-predictions*
1-bit : 4 miss-predictions (1 λάθος σε κάθε T,N,N ή N,T,T)
2-bit : 8 miss-predictions (2 λάθη σε κάθε T,N,N ή N,TT)

Επιλέγεται ο 1-bit.

(iii) *static Taken : 16 miss-predictions*
1-bit : 9 miss-predictions (1 λάθος σε κάθε T,N,N και 2 λάθη σε κάθε T,N,T ή N,T,N)
2-bit : 6 miss-predictions (2 λάθη σε κάθε T,N,N και 1 λάθος σε κάθε T,N,T ή N,T,N)

Επιλέγεται ο 2-bit predictor.

B. Δίνονται οι παρακάτω εντολές διακλάδωσης :

B1 : T, N, T, T, T, N, N, N, T, N, T, N, T, T, N, T, N
B2 : N, T, N, N, N, T, T, T, N, T, N, T, N, N, T, N, T

Εκτός από τους μηχανισμούς του ερωτήματος Α μπορείτε τώρα να χρησιμοποιήσετε και έναν (m,n) global history predictor, όπου m,n = 1,2. Ποιο μηχανισμό πρόβλεψης θα προτείνατε για τη διακλάδωση B2 και γιατί;

Παρατηρούμε πως το B2 είναι διαρκώς το αντίθετο του B1. Όταν δηλαδή το B1 είναι T το B2 είναι N και αντίστοιχα αν το B1 είναι N το B2 είναι T. Σε έναν global history predictor (m,n) θα αρκούσε ένας 1-bit local predictor, δηλαδή n = 1. Επίσης, επειδή το B2 εκτελείται πάντα μετά από το B1, αρκεί 1 bit ιστορίας (το οποίο θα κρατάει το αποτέλεσμα του B1) για να προβλέπουμε το B2, δηλαδή m = 1.

Χρησιμοποιώντας λοιπόν έναν (1,1) predictor με κατάλληλη αρχικοποίηση θα είχαμε 100% ακρίβεια πρόβλεψης για το B2.

Θέμα 3ο (2.5 μονάδες)

A. Έστω ένας επεξεργαστής που υλοποιεί τον αλγόριθμο Tomasulo με out-of-order commit εντολών, καθώς και οι παρακάτω εντολές :

```
1.   LD   R1, 0(R2)
2.   ADD  R3, R4, R1
3.   SUB  R5, R4, R1
4.   MUL  R5, R4, R8
```

Υποθέστε επίσης τα εξής :

- Η αρχιτεκτονική διαθέτει 6 reservation stations, τα Load1, Load2, Add1, Add2, Mult1 και Mult2
- Οι εντολές LD, ADD και SUB απαιτούν 5 κύκλους εκτέλεσης ενώ η MULT 7 κύκλους

(i) Δώστε την εικόνα των reservation stations όταν γίνει issued και η 4η εντολή, δηλαδή όταν cycles = 4, φτιάχνοντας έναν πίνακα όπως παρακάτω:

Name	Busy	Op	Vj	Vk	Qj	Qk
Add1	Yes	Add	R[R4]			Load1
Add2	Yes	Sub	R[R4]			Load1
Mult1	Yes	Mul	R[R4]	R[R8]		
Mult2						

	Busy	Address
Load1	Yes	M[0+R2]
Load2		

Reg. res.status	R1	R2	R3	R4	R5	R6	R7	R8
Qi	Load1		Add1		Mult1			

(ii) Δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών, συμπληρώνοντας έναν πίνακα όπως ο παρακάτω :

OP	Issue	Exec	WB	Σχόλιο
LD R1, 0(R2)	1	2 - ??	??	

Αν για κάποιες εντολές υπάρχει καθυστέρηση μεταξύ Issue-Exec ή Exec-WB, χρησιμοποιήστε το πεδίο "Σχόλιο" για να την εξηγήσετε.

Instruction		j	k	IS	EX	WB
LD	R1	0	R2	1	2 – 6	7
ADD	R3	R4	R1	2	8 – 12	13
SUB	R5	R4	R1	3	13 – 17	18
MUL	R5	R4	R8	4	5 – 11	12

Σημείωση: έχουμε υποθέσει 1 adder, οπότε η SUB θα πρέπει να περιμένει να τελειώσει πρώτα η ADD.

(iii) Αν η αρχιτεκτονική χρησιμοποιούσε έναν ROB τι θα άλλαζε στα πιο πάνω αποτελέσματα; Υποθέστε ότι ο ROB έχει άπειρες θέσεις.

IS	EX	WB	CO
1	2 – 6	7	8
2	8 – 12	13	14
3	13 – 17	18	19
4	5 – 11	12	20

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

1. Τα IS, WR, CMT απαιτούν 1 κύκλο.
2. Στο στάδιο EX εκτελείται και ο υπολογισμός της διεύθυνσης μιας προσπέλασης στη μνήμη καθώς και η ίδια προσπέλαση.
3. Το σύστημα έχει άπειρα reservation stations καθώς και άπειρες θέσεις στον ROB.
4. Υπάρχουν 2 Integer Functional Units τα οποία χρησιμοποιούνται για όλες τις πράξεις integer αριθμών καθώς και όλες τις προσπελάσεις στη μνήμη.
5. Υπάρχουν 2 Floating Point Functional Units, τα οποία χρησιμοποιούνται για όλες τις πράξεις μεταξύ floating point αριθμών (προσθέσεις, πολλαπλασιασμούς, διαιρέσεις).
6. Δεν υπάρχει προώθηση μεταξύ των διαφόρων FUs. Τα αποτελέσματα μεταφέρονται σε κάθε reservation station μέσω του CDB.

7. Οι εντολές που χρησιμοποιούν τα Integer FUs απαιτούν 1 κύκλο εκτέλεσης, ενώ για αυτές που χρησιμοποιούν τα Floating Point FUs απαιτούνται 5 κύκλοι για ADD, 10 κύκλοι για MULT και 14 κύκλοι για DIV.
8. Ο επεξεργαστής είναι 2-wide superscalar, επομένως σε κάθε κύκλο 2 εντολές μπορούν να γίνονται issued, να γίνονται commit ή να βρίσκονται στο WR στάδιο (επομένως υπάρχουν 2 CDBs).

Το σύστημα εκτελεί τις παρακάτω εντολές :

```
1.    LD        F0, 0(R1)
2.    MUL.D     F2, F1, F0
3.    DIV.D     F6, F2, F0
4.    ADD.D     F2, F4, F8
5.    LD        F5, 0(R3)
6.    LD        F10, 0(R2)
7.    DIV.D     F6, F1, F4
8.    MUL.D     F4, F0, F5
9.    ADD.D     F2, F7, F8
```

Δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών συμπληρώνοντας έναν πίνακα όπως ο παρακάτω :

OP	IS	EX	WR	CMT	Σχόλιο
LD F0, 0(R1)	1	2	3	4	
MUL.D F2,F1,F0	??	??	??	??	??

Στο πεδίο “Σχόλιο” δικαιολογήστε τυχόν καθυστερήσεις μεταξύ IS-EX, EX-WR και WR-CMT.

Instruction		j	k	IS	EX	WB	CO	Σχόλιο
LD	F0	0	R1	1	2 – 2	3	4	
MUL.D	F2	F1	F0	1	4 – 13	14	15	α
DIV.D	F6	F2	F0	2	22 – 35	36	37	β
ADD.D	F2	F4	F8	2	3 – 7	8	37	
LD	F5	0	R3	3	4 – 4	5	38	
LD	F10	0	R2	3	4 – 4	5	38	
DIV.D	F6	F1	F4	4	8 – 21	22	39	γ
MUL.D	F4	F0	F5	4	14 – 23	24	39	δ
ADD.D	F2	F7	F8	5	24 – 28	29	40	ε

Εκτός από τα stalls μεταξύ των σταδίων WB-CO που είναι απαραίτητα ώστε οι εντολές να γίνονται commit στη σειρά προγράμματος, τα υπόλοιπα stalls είναι τα εξής:

α) Stall λόγω αναμονής F0 από την LD
β) Αναμονή F2 από την 1η MUL. Μπορεί να ξεκινήσει από τον 15ο κύκλο και μετά, εφόσον βρει εκείνη τη στιγμή ελεύθερη fp μονάδα. Όμως στον 14ο κύκλο την μονάδα θα την πάρει η 2η MUL.
γ) Stall λόγω αναμονής fp μονάδας. Τελειώνει πρώτη η 1η ADD στον κύκλο 7, οπότε η 2η DIV μπορεί να ξεκινήσει να εκτελείται από τον κύκλο 8.
δ) Stall λόγω αναμονής fp μονάδας. Θα πάρει την μονάδα μόλις τελειώσει η 1η MUL.
ε) Stall λόγω αναμονής fp μονάδας. Θα πάρει την μονάδα μόλις τελειώσει η 2η MUL (θα μπορούσε να πάρει τη μονάδα της 2ης DIV μόλις εκείνη τελειώσει στον κύκλο 21, όμως η 1η DIV έχει προτεραιότητα).

Θέμα 4ο (2,5 μονάδες)

Α. Εξετάζουμε την εκτέλεση του ακόλουθου βρόχου ο οποίος αντιμεταθέτει έναν 256x256 πίνακα a και αποθηκεύει το αποτέλεσμα σε έναν πίνακα b.

```
for(i=0; i<256; i++)
     for(j=0; j<256; j++)
          b[i][j] = a[j][i];
```

Οι πίνακες περιέχουν στοιχεία κινητής υποδιαστολής διπλής ακρίβειας, μεγέθους 8 bytes το καθένα. Κάνουμε τις εξής υποθέσεις:

- Το πρόγραμμα εκτελείται σε έναν επεξεργαστή με μόνο ένα επίπεδο κρυφής μνήμης δεδομένων, η οποία αρχικά είναι άδεια. Η κρυφή μνήμη είναι πλήρως συσχετιστική (fully associative), write-allocate, έχει μέγεθος 16 KB, και έχει LRU πολιτική αντικατάστασης. Το μέγεθος του block είναι 64 bytes.
- Οι πίνακες είναι αποθηκευμένοι στην κύρια μνήμη κατά γραμμές. Επιπλέον, είναι "ευθυγραμμισμένοι" ώστε το πρώτο στοιχείο του καθενός να απεικονίζεται στην αρχή μιας γραμμής της κρυφής μνήμης.

(i) Βρείτε το συνολικό ποσοστό αστοχίας (miss rate) για τις αναφορές που γίνονται στην μνήμη στον παραπάνω κώδικα.

*Κάθε cache line είναι 64 bytes, άρα σε 1 cache line χωράνε 8 στοιχεία κάποιου πίνακα. Επιπλέον, για την αποθήκευση 1 γραμμής ενός πίνακα απαιτούνται 8*256 = 2048 bytes, ενώ για την αποθήκευση 1 στήλης απαιτούνται 64*256 = 16KB (η αναφορά σε κάθε στοιχείο της στήλης οδηγεί στην φόρτωση καινούριας cache line). Αυτό αμέσως σημαίνει ότι τα στοιχεία μιας στήλης δεν μπορούν να επαναχρησιμοποιηθούν αφού απαιτούν ολόκληρη την cache για να αποθηκευτούν.*

Για την 1η επανάληψη του εξωτερικού loop, οι αναφορές που γίνονται είναι οι ακόλουθες:

$a_{0,0}$ $b_{0,0}$ *m m*
$a_{1,0}$ $b_{0,1}$ *m h*
$a_{2,0}$ $b_{0,2}$ *m h*
...
$a_{7,0}$ $b_{0,7}$ *m h*
$a_{8,0}$ $b_{0,8}$ *m m (νέα cache line για τον b)*
$a_{9,0}$ $b_{0,9}$ *m h*
...
$a_{15,0}$ $b_{0,15}$ *m h*
$a_{16,0}$ $b_{0,16}$ *m m*
...
$a_{255,0}$ $b_{0,255}$ *m h*

*Το παραπάνω miss pattern επαναλαμβάνεται ανά 8 επαναλήψεις του εσωτερικού loop. Συνεπώς, θα έχουμε συνολικά 256/8 * 9 misses = 288 misses για 256 επαναλήψεις του εσωτερικού loop, δηλαδή για 1 επανάληψη του εξωτερικού loop.*
Στην επόμενη επανάληψη του εξωτερικού loop, δεν υπάρχει επαναχρησιμοποίηση σε κανέναν από τους δύο πίνακες. Συγκεκριμένα, η αναφορά στα τελευταία στοιχεία της 1ης γραμμής του b κατά την 1η επανάληψη του εξωτερικού loop, έχει οδηγήσει στον εκτοπισμό των πρώτων στοιχείων της 1ης (και μαζί της 2ης, 3ης,...,8ης) στήλης του a, τα οποία θα μπορούσαν να επαναχρησιμοποιηθούν στις επόμενες επαναλήψεις του εξωτερικού loop.
*Συνεπώς, και στις επόμενες επαναλήψεις του εξωτερικού loop έχουμε το ίδιο miss pattern, δηλαδή θα έχουμε συνολικά 256*288 = 73728 misses σε σύνολο 2*256*256 αναφορών, που δίνει miss rate ίσο με*

288/512 = 56.25%.

(ii) Εφαρμόστε την τεχνική του blocking στον παραπάνω κώδικα παρουσιάζοντας τον βελτιστοποιημένο κώδικα. Ποιο block size θα επιλέγατε ως καταλληλότερο και γιατί;

```
for(i=0; i<256; i+=bs)
     for(j=0; j<256; j+=bs)
          for(ii=i; ii<i+bs; ii++)
               for(jj=j; jj<j+bs; jj++)
                    b[ii][jj] = a[jj][ii];
```

Η επιλογή του block size πρέπει να γίνει με τέτοιο τρόπο ώστε τα 2 blocks, 1 για τον πίνακα a και 1 για τον πίνακα b, να χωράνε στην cache. Αυτό σημαίνει ότι κάθε block θα απαιτεί 8KB, δηλαδή θα απαρτίζεται από 1024 στοιχεία, επομένως θα έχει διάσταση 32x32. Επομένως bs=32.

(iii) Για το block size που απαντήσατε στο ερώτημα β, υπολογίστε το ποσοστό αστοχίας για τον blocked κώδικα.

Για την 1η επανάληψη του 2ου εσωτερικότερου loop (ii=0), οι αναφορές που γίνονται είναι οι εξής:

$a_{0,0}$ $b_{0,0}$ m m
$a_{1,0}$ $b_{0,1}$ m h
$a_{2,0}$ $b_{0,2}$ m h
...
$a_{7,0}$ $b_{0,7}$ m h
$a_{8,0}$ $b_{0,8}$ m m
$a_{9,0}$ $b_{0,9}$ m h
...
$a_{15,0}$ $b_{0,15}$ m h
$a_{16,0}$ $b_{0,16}$ m m
$a_{17,0}$ $b_{0,17}$ m h
...
$a_{23,0}$ $b_{0,23}$ m h
$a_{24,0}$ $b_{0,24}$ m m
$a_{25,0}$ $b_{0,25}$ m h
...
$a_{31,0}$ $b_{0,31}$ m h

*Ανά 8 επαναλήψεις του loop jj το miss pattern επαναλαμβάνεται, οπότε έχουμε συνολικά 4*9=36 misses. Στην επόμενη επανάληψη του 2ου εσωτερικότερου loop (ii=1), θα έχουμε επαναχρησιμοποίηση στα στοιχεία του a (την οποία δεν καταφέραμε να έχουμε στην non-blocked έκδοση του κώδικα λόγω του ότι δουλεύαμε με πίνακες που δεν χωρούσαν στην cache).*

$a_{0,1}$ $b_{1,0}$ h m
$a_{1,1}$ $b_{1,1}$ h h
$a_{2,1}$ $b_{1,2}$ h h
...
$a_{7,1}$ $b_{1,7}$ h h
$a_{8,1}$ $b_{1,8}$ h m
$a_{9,1}$ $b_{1,9}$ h h
...
$a_{15,1}$ $b_{1,15}$ h h
$a_{16,1}$ $b_{1,16}$ h m

$a_{17,1}$ $b_{1,17}$ h h
...
$a_{23,1}$ $b_{1,23}$ h h
$a_{24,1}$ $b_{1,24}$ h m
$a_{25,1}$ $b_{1,25}$ h h
...
$a_{31,1}$ $b_{1,31}$ h h

*Σε αυτή την περίπτωση έχουμε συνολικά 4*1=4 misses.*
Η επαναχρησιμοποίηση στα στοιχεία του a θα συνεχιστεί και για ii=2,3,...,7, όπου θα έχουμε και εκεί 4 misses ανά επανάληψη.
*Συνολικά λοιπόν, για ii=0,...,7, θα έχουμε 36 + 7*4 = 64 misses.*
Το pattern αυτό θα επαναλαμβάνεται για ii=8,...,15, ii=16,...,23, ii=24,...,31 (όπου αλλάζει η στήλη για τον πίνακα a ώστε να φορτώνεται κανούρια cache line).
*Συνολικά επομένως, για μια πλήρη εκτέλεση των 2 εσωτερικότερων loops (για την εκτέλεση δηλαδή του αλγορίθμου σε επίπεδο block πίνακα), θα έχουμε 4*64 = 256 misses σε σύνολο 2*32*32=2048 αναφορών.*

Το miss pattern αυτό επαναλαμβάνεται για όλες τις επαναλήψεις των 2 εξωτερικότερων loops, για όλους δηλαδή τους συνδυασμούς blocks των πινάκων a και b (δεν υπάρχει επαναχρησιμοποίηση σε επίπεδο block, μόνο σε επίπεδο στοιχείων εντός του block). Επομένως, το miss rate ισούται με 256/2048 = 12.5%.