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 2012
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 (20%)

Α. Εξηγήστε για ποιο λόγο οι αρχιτεκτονικές που χρησιμοποιούν τον αλγόριθμο Tomasulo χρειάζονται κάποιο μηχανισμό πρόβλεψης διακλαδώσεων.

Ο Tomasulo υλοποιεί out-of-order εκτέλεση εντολών προκειμένου να βελτιώσει την απόδοση του συστήματος. Αυτό επιτυγχάνεται δρομολογώντας εντολές που με βάση το program order ακολουθούν άλλες, οι οποίες όμως είναι blocked περιμένοντας το αποτέλεσμα άλλων προηγούμενων εντολών. Απαιτείται λοιπόν η δρομολόγηση πολλών εντολών που βρίσκονται πιο μπροστά στο instruction stream. Έτσι όταν συναντούν ένα branch απαιτείται ένας μηχανισμός πρόβλεψης ώστε να επιλεγεί το path από το οποίο θα έρθουν οι επόμενες εντολές. Διαφορετικά, αν δεν υπήρχε ο μηχανισμός πρόβλεψης, θα έπρεπε να περιμένουν το αποτέλεσμα του branch περιορίζοντας έτσι τα οφέλη της ooo εκτέλεσης.

Β. Σας αναθέτουν να σχεδιάσετε έναν επεξεργαστή που θα υλοποιεί ένα ISA το οποίο ορίζει K καταχωρητές (architecture registers). Για να βελτιώσετε την απόδοση, αποφασίζετε να σχεδιάσετε ένα out-of-order επεξεργαστή που θα υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ένα Reorder Buffer N θέσεων καθώς και M Reservation Stations. Ποιος είναι ο μέγιστος αριθμός φυσικών καταχωρητών (physical registers) που θα ενσωματώσετε στο σχέδιο σας;

Καταρχάς, χρειαζόμαστε τουλάχιστον όσους ορίζει το ISA, δηλαδή K. Εφόσον το σύστημα χρησιμοποιεί ένα ROB N θέσεων μπορούμε να έχουμε το πολύ N εντολές in-flight (να εκτελούνται ταυτόχρονα) οι οποίες γράφουν σε κάποιο καταχωρητή. Για να μπορούμε λοιπόν να κάνουμε το register renaming απαιτούνται N έξτρα φυσικοί καταχωρητές.

$$\text{physical registers} \leq N + K$$

Γ. Υποθέστε μια αρχιτεκτονική που υλοποιεί τον αλγόριθμο Tomasulo με χρήση Reorder Buffer. Σε ποιο στάδιο πραγματοποιούνται τα loads (δηλ. έρχεται στον επεξεργαστή από την μνήμη η τιμή που θα χρησιμοποιηθεί από κάποια επόμενη εντολή); Σε ποιο στάδιο πραγματοποιούνται τα stores (δηλ. στέλνεται στην μνήμη η τιμή που έχει γράψει ο επεξεργαστής); Αν τα στάδια είναι διαφορετικά εξηγήστε το γιατί.

Τα loads πραγματοποιούνται στο EX (όταν ολοκληρωθεί το EX η τιμή έχει έρθει στον επεξεργαστή). Αντίθετα τα stores πραγματοποιούνται στο CMT. Η διαφορά οφείλεται στις επιπτώσεις που έχουν οι εντολές αυτές κατά την υποθετική εκτέλεση (speculative execution). Τα loads δεν επηρεάζουν το architectural state και για αυτό το λόγο επιτρέπουμε να πραγματοποιούνται στο EX. Αντίθετα, τα stores

μεταβάλλουν την κατάσταση του συστήματος και για αυτό το λόγο τα πραγματοποιούμε όταν ξέρουμε ότι η εντολή επιτρέπεται να εκτελεστεί με βάση τη σειρά του προγράμματος (χωρίς δηλαδή να αντιμετωπίζουμε κινδύνους λόγω exceptions/interrupts ή mispredicted branches).

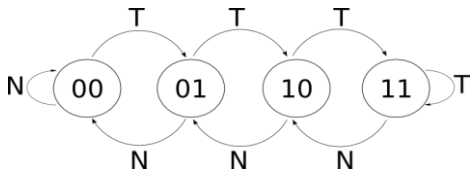
Δ. Δίνεται ο παρακάτω κώδικας σε C καθώς και η αντίστοιχη assembly.

```
int array[N]= {0, 1, -3, 4, 1};

for(int i = 0; i < N; i++)
    if (array[i] != 0)
        array[i] = array[i] + 1;
```

```
0x00880004         addi  $n, $0, N
0x00880008         addi  $i, $0, 0
0x0088000C  loop:  ld    $a, array($i)
0x00880010         beqz  $a, endif
0x00880014         st    $a, array($i)
0x00880018  endif:
                   addi  $i, $i, 4
0x0088001C         addi  $n, $n, -1
0x00880020         bnez  $n, loop
```

(i) Έστω ότι το σύστημα περιέχει ένα πίνακα BHT με 2-bit predictors αρχικοποιημένους στο 00. Αν ο πίνακας περιέχει 64 εγγραφές και δεικτοδοτείται από τα low order bits του PC, υπολογίστε την ακρίβεια πρόβλεψης για τις 2 εντολές άλματος του παραπάνω κώδικα. Κατά την δεικτοδότηση αγνοούνται προφανώς τα 2 λιγότερα σημαντικά ψηφία του PC. Δίνεται επίσης το FSM του 2-bit predictor.



64 εγγραφές → 6 bits index.
Branch1: PC = 0000 0000 1000 1000 0000 0000 0001 0000 → index = 000100 = 4.
Branch2: index = 001000 = 8.
Τα 2 branches χρησιμοποιούν διαφορετικούς predictors. Έτσι έχουμε :

	Branch 1	**Branch 2**
Πρόβλεψη	N / N / N / N / N	N / N / T / T / T
Αποτέλεσμα	T / N / N / N / N	T / T / T / T / N
Ακρίβεια πρόβλεψης	4/5 = 80%	2/5 = 40%

(ii) Ποιος είναι ο ελάχιστος αριθμός εγγραφών του BHT για τον οποίο δεν επηρεάζεται η ακρίβεια της πρόβλεψης; Ποια θα είναι η ακρίβεια πρόβλεψης από εκεί και κάτω;

Όσο δεν έχουμε aliases η απόδοση των predictors παραμένει η ίδια. Τα 2 branches θα χρησιμοποιούν τον ίδιο predictor εφόσον το index έχει μήκος μικρότερο ή ίσο του 2. Επομένως ο ελάχιστος αριθμός θέσεων για τον οποίο δεν επηρεάζεται η απόδοση είναι 8 (Branch1 index = 100 και Branch2 index = 000). Από το σημείο αυτό και κάτω, δηλαδή για 1, 2 ή 4 θέσεις η ακρίβεια θα είναι :

	Branch 1	**Branch 2**
Πρόβλεψη	N / T / T / T / T	N / N / N / N / N
Αποτέλεσμα	T / N / N / N / N	T / T / T / T / N
Ακρίβεια πρόβλεψης	0/5 = 0%	1/5 = 20%

Θέμα 2ο (20%)

Α. Ποια η διαφορά μεταξύ snooping και directory πρωτοκόλλων; Ποιος από τους 2 παραπάνω τύπους πρωτοκόλλων κλιμακώνει καλύτερα; Εξηγήστε γιατί.

Στα snooping πρωτόκολλα απαιτείται οι controllers όλων των caches να παρακολουθούν τις λειτουργίες όλων. Αντίθετα, στα directory πρωτόκολλα, επειδή ο κατάλογος γνωρίζει ποιοι είναι οι sharers για κάθε block, καθορίζει αυτός ποιες caches θα πρέπει να ενημερωθούν για κάποια λειτουργία μνήμης. Το γεγονός ότι δεν απαιτούνται broadcasts κάνει τα directory πρωτόκολλα να κλιμακώνουν καλύτερα από τα snooping πρωτόκολλα.

Β. Πώς πλεονεκτεί μια υλοποίηση *Test-and-Test-and-Set* μιας λειτουργίας κλειδώματος (lock) έναντι μιας υλοποίησης *Test-and-Set* σε ένα πολυεπεξεργαστικό σύστημα που χρησιμοποιεί write-invalidate cache coherence protocol; Εξηγήστε την απάντησή σας.

Στην test-and-set υλοποίηση, κάθε φορά που ένας επεξεργαστής που δεν έχει το lock επιχειρεί να το πάρει, αναγκάζεται να γράψει στη θέση μνήμης που αντιστοιχεί στη διεύθυνση του lock. Αυτό συμβαίνει σε κάθε επανάληψη του test-and-set loop, με αποτέλεσμα όταν οι διεκδικητές του lock είναι πολλοί να προκαλούνται αλλεπάλληλα invalidations της cache line που περιέχει το lock, και συνεπώς να εισάγεται αυξημένη κίνηση στο δίαυλο. Όσο μεγαλύτερος είναι ο αριθμός των διεκδικητών ή όσο περισσότερος ο χρόνος που το lock είναι κρατημένο, τόσο μεγαλύτερη η συμφόρηση που υπάρχει στο δίαυλο.

Αντίθετα, στην test-and-test-and-set υλοποίηση, σε κάθε επανάληψη του loop ο διεκδικητής του lock *διαβάζει* την τιμή του lock, και μόνο όταν αυτό φανεί ότι απελευθερώθηκε, επιχειρεί να το πάρει. Οι διεκδικητές, κατά συνέπεια, κάνουν spinning τοπικά στις caches τους, με αποτέλεσμα να μην εισάγεται άσκοπα κίνηση στον δίαυλο.

Γ. Θεωρήστε ένα πολυεπεξεργαστικό σύστημα κοινής μνήμης 3 επεξεργαστών καθώς και τον ακόλουθο κώδικα για 3 διεργασίες που εκτελούνται παράλληλα στο σύστημα. Αρχικά ισχύει ότι A=B=0.

P1	**P2**	**P3**
a1: A = 1	b1: u = A b2: B = 1	c1: v = B c2: w = A

(i) Βρείτε ποιοι συνδυασμοί τιμών για τις μεταβλητές u,v,w είναι ακολουθιακά συνεπείς (sequentially consistent) και ποιοι όχι. Εξηγήστε τις απαντήσεις σας δείχνοντας πώς μπορούν να παραχθούν ή πώς παραβιάζουν το SC.

u	**v**	**w**	
0	0	0	SC: b1, c1, c2, ...
0	0	1	SC: b1, c1, a1, c2, ...
0	1	0	SC: b1, b2, c1, c2, a1
0	1	1	SC: b1, a1, b2, c1, c2
1	0	0	SC: c1, c2, a1, b1, b2
1	0	1	SC: c1, a1, b1, c2, b2
1	1	0	not SC: w=0: c2→a1, όμως u=1,v=1: a1→b1, b2→c1 (→c2), άρα δε γίνεται ταυτόχρονα
1	1	1	SC: a1, b1, b2, c1, c2

(ii) Θεωρήστε ότι το σύστημα υλοποιεί το πλέον relaxed memory model (π.χ. RMO). Ποιοι από τους συνδυασμούς του ερωτήματος α που *δεν* επιτρέπονται στο SC, μπορούν να εμφανιστούν τώρα; Εισάγετε τον ελάχιστο αριθμό εντολών memory fence στον παραπάνω κώδικα ώστε να αποτρέπεται η εμφάνισή τους.

Η (u,v,w)=(1,1,0) μπορεί να εμφανιστεί σε ένα relaxed μοντέλο που επιτρέπει αναδιάταξη των writes, ως εξής: **c2**, a1, b1, b2, **c1** . Μπορούμε να την αποφύγουμε με την τοποθέτηση ενός memory fence ανάμεσα στις c1 και c2.

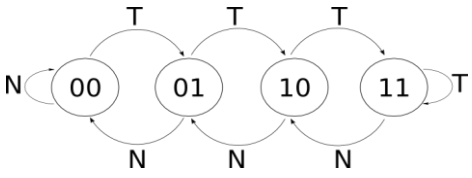
Ακόμα όμως και αν διατηρηθεί η σωστή σειρά στις εντολές του P3, θα μπορούσε να εμφανιστεί και ως εξής: **b2**, c1, c2, a1, **b1** . Άρα, θα πρέπει να τοποθετηθεί και δεύτερο memory fence ανάμεσα στις b1 και b2.

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 2 RS για προσθέσεις/αφαιρέσεις και 1 RS για πολλαπλασιασμούς/διαιρέσεις floating point αριθμών. Αντίστοιχα, για integer αριθμούς περιλαμβάνονται 4 RS για εντολές διακλάδωσης, αριθμητικές και λογικές εντολές.
- Το σύστημα περιλαμβάνει 1 non-pipelined functional unit για πράξεις integer αριθμών. Όλες οι εντολές μεταξύ integer αριθμών διαρκούν 1 κύκλο.
- Το σύστημα περιλαμβάνει 2 non-pipelined floating point functional units, ένα για ADDD / SUBD και ένα για MULD / DIVD. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 2 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 5 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, το καθένα από τα οποία διαθέτει 2 θέσεις. Οι εντολές χρησιμοποιούν ένα ξεχωριστό, pipelined functional unit για τον υπολογισμό της διεύθυνσης και διαρκούν 1 κύκλο στην περίπτωση Hit στην cache και 5 κύκλους σε περίπτωση Miss.
- Η πρόβλεψη μιας εντολή διακλάδωσης υπό συνθήκη γίνεται *ταυτόχρονα* με τη δρομολόγηση της εντολής. Ο έλεγχος της πρόβλεψης γίνεται μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR (κύκλος k). Σε περίπτωση σφάλματος, σταματά η εκτέλεση των εντολών του miss-predicted execution path και στον επόμενο κύκλο (κύκλος k+1) δρομολογείται η σωστή εντολή.
- Ο ROB έχει 6 θέσεις.
- Το σύστημα περιλαμβάνει 1 CDB. Σε περίπτωση που παραπάνω από μια εντολές θέλουν να το χρησιμοποιήσουν, τότε προτεραιότητα αποκτά η ‘‘*παλαιότερη*’’ εντολή (αυτή που έγινε issued πρώτη). Θεωρήστε ότι τα branches δεν χρησιμοποιούν το CDB κατά τη διάρκεια του WR σταδίου τους.
- Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί τον παρακάτω πίνακα από 2-bit predictors. Δίνεται επίσης το FSM διάγραμμα του 2-bit predictor.

11
00
01
10



Η δεικτοδότηση του πίνακα γίνεται χρησιμοποιώντας τον κατάλληλο αριθμό low order bits από το PC της εντολής. Ο επεξεργαστής υλοποιεί το ISA του MIPS, οι εντολές απέχουν μεταξύ τους 4 bytes και επομένως κατά τη δεικτοδότηση θα πρέπει να αγνοήσετε τα 2 λιγότερα σημαντικά bits του PC.

- Το σύστημα περιλαμβάνει μια fully associative cache με 2 cache lines και μέγεθος block = 16 bytes, η οποία χρησιμοποιεί πολιτική αντικατάστασης LRU. Αρχικά η cache είναι άδεια.
- Οι καταχωρητές R1, R2 περιέχουν τις διευθύνσεις των πρώτων στοιχείων δύο πινάκων A, B αριθμών διπλής ακρίβειας (μήκους 8 bytes ο καθένας). Οι πίνακες είναι ευθυγραμμισμένοι.

Δίνεται ο παρακάτω κώδικας :

```
0x00880004  LOOP: LD    F0, 0(R1)
0x00880008        ADDD  F1, F1, F0
0x0088000C        LD    F2, 0(R2)            /* Ο αντίστοιχος κώδικας σε C */
0x00880010        ADDD  F1, F1, F2
0x00880014        LD    F0, 8(R1)            for(int i=0,j=0; i<4; i+=2, j++)
0x00880018        MULD  F1, F1, F0              temp = (temp+A[i]+B[j])*A[i+1];
0x0088001C        ADDI  R2, R2, #8
0x00880020        ADDI  R1, R1, #16          temp = 2*(temp + temp2);
0x00880024        SUB   R6, R5, R1           B[0] = B[0] + temp;
0x00880028        BNEZ  R6, LOOP
0x0088002C        ADDD  F1, F1, F6
0x00880030        ADDD  F1, F1, F1
0x00880034        LD    F0, -16(R2)
0x00880038        ADDD  F0, F0, F1
0x0088003C        ST    F0, -16(R2)
```

Δίνεται επίσης ότι $R_5 = R_1 + 32$. 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-??	??	??	

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

OP	IS	EX	WR	CMT	Σχόλιο
LD F0,0(R1)	1	2 - 6	7	8	Miss A[0] → Fetch A[0]A[1] in block0, LRU=1
ADDD F1, F0, F0	2	8 - 9	10	11	
LD F2, 0(R2)	3	4 – 8	9	12	Miss B[0] → Fetch B[0]B[1] in block1, LRU=0
ADDD F1, F1, F2	4	11 - 12	13	14	
LD F0, 8(R1)	8	9	11	15	Structural hazard, Hit A[1], LRU=1
MULD F1, F1, F0	9	14 - 18	19	20	
ADDI R2, R2, 8	10	11	12	21	ROB FULL
ADDI R1, R1, 16	12	13	14	22	
SUB R6, R5, R1	13	15	16	23	R6 = 16
BNEZ R6, LOOP	15	17	18	24	PRED NT, RES = T
ADDD F6, F1, F6	16				FLUSH @ 18
LD F0, 0(R1)	19	20 - 24	25	26	Miss A[2] → Fetch A[2]A[3] in block1, LRU=0
ADDD F1, F0, F0	21	26 - 27	28	29	
LD F2, 0(R2)	22	23 - 27	29	30	Miss B[1] → Fetch B[0]B[1] in block 0, LRU=1
ADDD F1, F1, F2	23	30 - 31	32	33	
LD F0, 8(R1)	26	27	30	34	Structural Hazard, Hit A[3], LRU = 0
MULD F1, F1, F0	27	33 - 37	38	39	
ADDI R2, R2, 8	28	29	31	40	ROB FULL
ADDI R1, R1, 16	30	31	33	41	
SUB R6, R5, R1	31	34	35	42	R6 = 0
BNEZ R6, LOOP	34	36	37	43	PRED = T, RES=NT
LD F0, 0(R1)	35	36 – 40			Miss A[4] → Fetch A[4]A[5] in block 0, LRU = 1, FLUSH @ 37
ADDD F6, F1, F6	38	39 - 40	41	44	
ADDD F1, F6, F6	40	42 - 43	44	45	
LD F0, -16(R2)	41	42 - 46	47	48	Miss B[0] → Fetch B[0]B[1] in block1, LRU = 0
ADDD F0, F0, F1	42	48 - 49	50	51	
ST F0, -16(R2)	43	51	52	53	Hit B[0]

Θέμα 4ο (20%)

Υποθέστε ένα σύστημα παράλληλης επεξεργασίας αποτελούμενο από 3 επεξεργαστές που χρησιμοποιεί το MESI cache coherence protocol. Κάθε επεξεργαστής διαθέτει ένα μόνο επίπεδο κρυφής μνήμης δεδομένων, για την οποία ισχύουν τα εξής:

- Αρχικά είναι άδεια.
- Είναι συσχέτισης δύο δρόμων (2-way set associative), write-allocate, αποτελούμενη από 32 blocks δεδομένων μεγέθους 32 bytes, με LRU πολιτική αντικατάστασης.
- Η μικρότερη μονάδα δεδομένων που μπορεί να διευθυνσιοδοτηθεί είναι το 1 byte.

Θεωρήστε ένα C πρόγραμμα που εκτελείται παράλληλα στους 3 επεξεργαστές, και το οποίο περιλαμβάνει τους εξής ορισμούς μεταβλητών:

```
struct my_s {
  double x,y,z;
};
double a[M];
struct my_s s[N];
double b[L];
```

Για το πρόγραμμα ισχύουν τα εξής:

- Χρησιμοποιεί τιμές κινητής υποδιαστολής διπλής ακρίβειας, μεγέθους 8 bytes η κάθε μία
- Τα M, N, L είναι τέτοια ώστε τα a[0], s[0], b[0] να απεικονίζονται στο 1ο set της cache

(i) Για την παρακάτω σειρά προσβάσεων, βρείτε τον αριθμό των hits και misses του κάθε επεξεργαστή. Υποδείξτε ποια misses είναι compulsory, capacity, conflict ή coherence (true sharing ή false sharing). Αρχικά οι caches είναι άδειες. Όλες οι αναφορές πλην αυτών που περιέχονται στα σκιασμένα κελιά, είναι αναγνώσεις.

Κάθε cache line χωράει 32/8 = 4 τιμές διπλής ακρίβειας. Αναφορικά με τον πίνακα s, η διάταξη των στοιχείων στην μνήμη γίνεται ως εξής:

P0	P1	P2	Hit / Miss
a[0]			comp. miss
b[0]			comp. miss
	a[0]		comp. miss
	b[0]		comp. miss
		a[0]	comp. miss
		b[0]	comp. miss
s[0].y			comp. miss
	s[1].y		comp. miss
		s[2].y	comp. miss
	s[1].x		comp. miss
s[0].x			coh. miss
		s[2].x	hit
		a[1]	hit
	a[1]		conf. miss

a[1]			conf. miss
	b[1]		conf. miss
b[1]			conf. miss
		b[1]	hit
s[0].y			conf. miss
	s[1].y		coh. miss
	s[1].x		coh. miss / conf. miss
s[0].x			coh. miss
		s[2].y	hit
		s[2].x	hit

(ii) Πώς θα αλλάζατε το παραπάνω πρόγραμμα ώστε να μειώσετε τον αριθμό των misses; Δικαιολογήστε την απάντησή σας.

Θα κάναμε padding στην δομή my_s ώστε να εξασφαλίσουμε ότι κάθε τριάδα x,y,z απεικονίζεται σε διαφορετική cache line. Αυτό θα οδηγούσε στην εξάλειψη των coherence misses.

```
struct my_s {
  double x,y,z;
  double dummy;
};
```