ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

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

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Κανονική Εξέταση Φεβρουαρίου 2011
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 (20%)

Α. Δίνονται οι παρακάτω functions γραμμένες σε assembly MIPS.

(i)
```
f:  sub   $s0,$a0,$a3
    sll   $v0,$s0,0x1
    add   $v0,$a2,$v0
    sub   $v0,$v0,$a1
    jr    $ra
```

(ii)
```
h:  addi  $sp,$sp,-8
    sw    $ra,4($sp)
    sw    $a0,0($sp)
    slti  $t0,$a0,2
    beq   $t0,$0,L1
    addi  $v0,$0,1
    addi  $sp,$sp,8
    jr    $ra
L1: addi  $a0,$a0,-1
    jal   h
    lw    $a0,4($sp)
    lw    $ra,0($sp)
    addi  $sp,$sp,8
    mul   $v0,$a0,$v0
    jr    $ra
```

(iii)
```
g:  addi  $sp,$sp,8
    sw    $ra,4($sp)
    sw    $s0,0($sp)
    move  $s0,$a2
    jal   h
    add   $v0,$v0,$s0
    lw    $ra,4($sp)
    lw    $s0,0($sp)
    addi  $sp,$sp,-8
    jr    $ra
```

Σε κάθε μια από αυτές τις συναρτήσεις υπάρχει ένα λάθος. Βρείτε το λάθος. Πώς θα το διορθώνατε;

(i) Σύμφωνα με τις συμβάσεις που ακολουθούν οι συναρτήσεις για τον MIPS, οι καταχωρητές $s διατηρούν τις τιμές τους κατά την κλίση μιας συνάρτησης. Εδώ όμως η f αλλάζει την τιμή του καταχωρητή $s0. Υπάρχουν 2 λύσεις για το πρόβλημα αυτό. Είτε θα πρέπει να σωθεί ο καταχωρητής στη στοίβα και να φορτωθεί ξανά πριν την επιστροφή από την f, είτε αντί για τον $s0 θα πρέπει να χρησιμοποιηθεί κάποιος $t καταχωρητής, πχ. ο $t0. Από αυτές θα προτιμούσαμε τη 2η καθώς θα έδινε πιο συμπτυγμένο (compact) κώδικα.

(ii) Εδώ γίνονται λάθος τα push και pop από τη στοίβα. Συγκεκριμένα, ο $ra σώνεται στο [$sp+4] και ο $a0 στο [$sp] ενώ η φόρτωση στο τέλος γίνεται για τον $ra από το [$sp] και για τον $a0 από το [$sp+4]. Προφανώς η φόρτωση στο τέλος θα πρέπει να γίνει από την ίδια διεύθυνση στην οποία έγινε η αποθήκευση.

(iii) Η συνάρτηση g επεκτείνει την στοίβα για να αποθηκεύσει τους καταχωρητές που χρειάζεται προς τη λάθος κατεύθυνση. Συγκεκριμένα, αυξάνει τον $sp κατά 8 (στην προσπάθεια να δημιουργήσει 2 νέες θέσεις), μετακινώντας έτσι το δείκτη προς τα "πάνω". Σύμφωνα με τις συμβάσεις του MIPS όμως, η

στοίβα μεγαλώνει προς τα "κάτω". Θα έπρεπε λοιπόν στην αρχή να προστίθεται το -8 στον $sp και στο τέλος το +8 (αντιμετάθεση της 1ης και της προτελευταίας εντολής).

Β. Ως επικεφαλής της ομάδας σχεδίασης του νέου επεξεργαστή MIPS, αποφασίζετε να αλλάξετε τη κλασσική σωλήνωση 5 σταδίων που διδαχθήκατε στο μάθημα. Συγκεκριμένα, αποφασίζετε να αντικαταστήσετε το EX στάδιο με το AC (address calculation), όπου ένας απλός αθροιστής υπολογίζει τις διευθύνσεις που χρησιμοποιούνται στις εντολές load και store. Η προσπέλαση της ALU από τις αριθμητικές και λογικές εντολές γίνεται πλέον στο 4ο στάδιο, όπου στην κλασσική σωλήνωση γίνεται η πρόσβαση στη μνήμη δεδομένων. Έτσι, κατά την εκτέλεση load/store εντολών στο 4ο στάδιο ενεργοποιείται η μνήμη και μένει ανενεργή η ALU, ενώ κατά την εκτέλεση αριθμητικών/λογικών εντολών ενεργοποιείται η ALU και μένει ανενεργή η μνήμη. Η νέα σωλήνωση έχει λοιπόν την εξής μορφή:

IF	ID	AC	EX/MEM	WB
Instruction Fetch	Instruction Decode and Register Read	Address Calculation	ALU execution or Memory Access	Write back to Register File

(i) Δώστε μια αλληλουχία εντολών όπου στην κλασσική σωλήνωση απαιτείται stall, ενώ στην καινούρια όχι.

Instructions	MIPS Classic Pipeline							New Pipeline					
lw r2,0(r1)	IF	ID	EX	M	WB			IF	ID	AC	E/M	WB	
add r2,r2,r3		IF	ID	-	EX	M	WB		IF	ID	AC	E/M	WB

(ii) Δώστε μια αλληλουχία εντολών όπου στην καινούρια σωλήνωση απαιτείται stall, ενώ στην κλασσική όχι.

Instructions	MIPS Classic Pipeline						New Pipeline						
add r1,r2,r3	IF	ID	EX	M	WB		IF	ID	AC	E/M	WB		
lw r4,0(r1)		IF	ID	EX	M	WB		IF	ID	-	AC	E/M	WB

Γ. Πώς ορίζονται οι WAW και WAR κίνδυνοι ανάμεσα σε 2 εντολές; Εξηγήστε γιατί αυτοί οι κίνδυνοι δεν μπορούν να εμφανιστούν στην κλασσική αρχιτεκτονική σωλήνωσης 5 σταδίων του MIPS.

Ο WAW κίνδυνος εμφανίζεται όταν μια εντολή προσπαθεί να γράψει σε έναν καταχωρητή πριν γράψει στον ίδιο καταχωρητή μια προηγούμενη εντολή.
πχ. ADD R1, R2, R3
ADD R1, R4, R5
Στον MIPS ο κίνδυνος αυτός δεν εμφανίζεται γιατί εγγραφή επιτρέπεται μόνο σε ένα στάδιο και οι εντολές εκτελούνται πάντα in-order. Επομένως πάντα θα γράφει η πρώτη εντολή.

Ο WAR κίνδυνος εμφανίζεται όταν μια εντολή προσπαθεί να γράψει σε έναν καταχωρητή πριν τον διαβάσει μια προηγούμενη εντολή.
πχ. ADD R1, R2, R3
ADD R3, R4, R5
Στον MIPS ο κίνδυνος αυτός δεν εμφανίζεται γιατί ο MIPS εκτελεί τις εντολές in-order και οι εντολές διαβάζουν τα ορίσματα τους στο στάδιο ID, το οποίο προηγείται του σταδίου WB όπου γίνονται οι εγγραφές στους καταχωρητές. Επομένως πάντα θα διαβάζει τον καταχωρητή η πρώτη εντολή.

Θέμα 2ο (30%)

Υποθέτουμε ότι έχουμε την παρακάτω αρχιτεκτονική σωλήνωσης 7 σταδίων.

IF	ID	X1	X2	X3	X4	WB
Instruction Fetch	Instruction Decode	Add/Sub/Logical Operations	Begin D$	Finish D$	Finish Mult/Div	Write Back

Στην αρχιτεκτονική αυτή οι λογικές και αριθμητικές εντολές, *εκτός από τον πολλαπλασιασμό και τη διαίρεση*, εκτελούνται στο στάδιο X1. Οι εντολές πολλαπλασιασμού και διαίρεσης απαιτούν 4 κύκλους και επομένως παράγουν το αποτέλεσμα τους στο τέλος του σταδίου X4. Οι εντολές πρόσβασης στη μνήμη απαιτούν 2 κύκλους (στάδια X2, X3) ενώ ο υπολογισμός της διεύθυνσης γίνεται στο στάδιο X1. Επιπλέον, η επίλυση των εντολών διακλάδωσης υπό συνθήκη γίνεται στο στάδιο X1 και σε περίπτωση που η διακλάδωση εκτελεστεί, η σωλήνωση εκκενώνεται (flush) και από τον επόμενο κύκλο έρχεται η σωστή εντολή. Τέλος, υποθέτουμε ότι η εγγραφή σε ένα καταχωρητή γίνεται στο πρώτο μισό ενός κύκλου ενώ η ανάγνωση από τον ίδιο καταχωρητή στο δεύτερο μισό του κύκλου.

Δίνεται το ακόλουθο κομμάτι κώδικα :

```
1. Loop:    lw   $3, 100($5)
2.          add  $1, $2, $3
3.          lw   $4, 0($1)
4.          add  $5, $4, $6
5.          mul  $7, $5, $4
6.          sw   $7, 0($1)
7.          addi $9, $9, -4
8.          bne  $9, #0, Loop
```

Α. Βρείτε όλες τις εξαρτήσεις στο παραπάνω κομμάτι κώδικα καθώς και την κατηγορία στην οποία ανήκουν (true / output / anti / control dependence).

RAW (True)	WAR (anti)	Control
2 → 1 ($3)	4 → 1 ($5)	8 → 1
3 → 2 ($1)		
4 → 3 ($4)		
5 → 4 ($5)		
5 → 3 ($4)		
6 → 5 ($7)		
6 → 2 ($1)		
8 → 7 ($9)		

Β. Υποθέστε ότι δεν υπάρχουν σχήματα προώθησης. Εκτελέστε την $1^{η}$ επανάληψη του βρόχου (μέχρι και το load της $2^{ης}$ επανάληψης) και χρησιμοποιήστε ένα διάγραμμα χρονισμού για να δείξετε τα διάφορα στάδια της σωλήνωσης από τα οποία διέρχονται οι παραπάνω εντολές. Πόσοι κύκλοι απαιτούνται για την εκτέλεση ολόκληρου του βρόχου; Υποθέστε ότι ο καταχωρητής $9 έχει αρχική τιμή 64.

Γ. Υποθέστε τώρα ότι υπάρχουν όλα τα δυνατά σχήματα προώθησης. Οι εντολές load και store απαιτείται να έχουν έτοιμα όλα τους τα ορίσματα στην αρχή του σταδίου X1. Δείξτε όπως και πριν το διάγραμμα χρονισμού για την $1^{η}$ επανάληψη του βρόχου, υποδεικνύοντας τις προωθήσεις που γίνονται. Πόσοι κύκλοι απαιτούνται τώρα για την εκτέλεση του κώδικα; Υποθέστε ότι ο καταχωρητής $9 έχει και πάλι αρχική τιμή 64.

Β. Το loop θα εκτελεστεί συνολικά 16 φορές. Επομένως συνολικά απαιτούνται **15*34 + 38 = 548 κύκλοι**.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**
lw $3,100($5)	F	D	X1	X2	X3	X4	WB																					
add $1,$2,$3		F	D	-	-	-	-	X1	X2	X3	X4	WB																
lw $4,0($1)			F	-	-	-	-	D	-	-	-	-	X1	X2	X3	X4	WB											
add $5,$4,$6								F	-	-	-	-	D	-	-	-	-	X1	X2	X3	X4	WB						
mul $7,$5,$4													F	-	-	-	-	D	-	-	-	-	X1	X2	X3	X4	WB	
sw $7,0($1)																		F	-	-	-	-	D	-	-	-	-	X1
addi $9,$9,-4																							F	-	-	-	-	D
bne $9,#0,Loop																												F

	29	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**
sw $7,0($1)	X2	X3	X4	WB									
addi $9,$9,-4	X1	X2	X3	X4	WB								
bne $9,#0,Loop	D	-	-	-	-	X1	X2	X3	X4	WB			
lw $3,100($5)							F	D	X1	X2	X3	X4	WB

Γ. Το loop θα εκτελεστεί συνολικά 16 φορές. Επομένως συνολικά απαιτούνται **15*17 + 21 = 276 κύκλοι**.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
lw $3,100($5)	F	D	X1	X2	X3	X4	WB																	
add $1,$2,$3		F	D	-	-	X1	X2	X3	X4	WB														
lw $4,0($1)			F	-	-	D	X1	X2	X3	X4	WB													
add $5,$4,$6						F	D	-	-	X1	X2	X3	X4	WB										
mul $7,$5,$4							F	-	-	D	X1	X2	X3	X4	WB									
sw $7,0($1)										F	D	-	-	-	X1	X2	X3	X4	WB					
addi $9,$9,-4											F	-	-	-	D	X1	X2	X3	X4	WB				
bne $9,#0,Loop															F	D	X1	X2	X3	X4	WB			
lw $3,100($5)																		F	D	X1	X2	X3	X4	WB

Θέμα 3ο (25%)

Α. Τι είναι η χωρική τοπικότητα των αναφορών και πως την εκμεταλλεύεται μια cache;

Χωρική τοπικότητα είναι η αρχή σύμφωνα με την οποία αν πραγματοποιηθεί μια πρόσβαση σε μια τοποθεσία της μνήμης, τότε είναι πιθανό σε σύντομο χρονικό διάστημα να πραγματοποιηθούν προσβάσεις και σε γειτονικές τοποθεσίες. Η cache εκμεταλλεύεται την αρχή αυτή, χρησιμοποιώντας blocks με μέγεθος μεγαλύτερο από μια λέξη, οπότε όταν φέρει μια λέξη που ζήτησε ο επεξεργαστής, φέρνει μαζί και γειτονικές της.

Β. Έστω μια cache αποτελούμενη από 4 γραμμές, με κάθε γραμμή να χωρά μια λέξη. Δώστε μια αλληλουχία προσβάσεων στη μνήμη για την οποία μια direct-mapped cache θα εμφάνιζε *μεγαλύτερο hit rate* από μια fully associative cache που χρησιμοποιεί πολιτική αντικατάστασης LRU. Στη συνέχεια δώστε και μια αλληλουχία προσβάσεων στη μνήμη για την οποία μια direct-mapped cache θα εμφάνιζε *μεγαλύτερο miss rate* από μια fully associative cache που χρησιμοποιεί πολιτική αντικατάστασης LRU.

Έστω οι λέξεις με διευθύνσεις 0, 1, 2, 3, 4 και η εξής αλληλουχία προσβάσεων:

	0	1	2	3	4	0	1	2	3	4
DM	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss
FA	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss

Δηλαδή για την αλληλουχία αυτή, η direct-mapped cache έχει 30% hit rate, ενώ η fully-associative 0%.
Έστω τώρα οι λέξεις με διευθύνσεις 0, 4, 8, 16 και η εξής αλληλουχία προσβάσεων:

	0	4	8	16	0	4	8	16	0	4
DM	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss
FA	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Δηλαδή για την αλληλουχία αυτή η direct-mapped cache έχει 100% miss rate, ενώ η fully-associative 40%.

Γ. Δίνεται μια 8-way associative cache μεγέθους 16KB με 64 sets. Η μικρότερη μονάδα που μπορεί να διευθυνσιοδοτηθεί είναι το 1 byte. Υπολογίστε το συνολικό μέγεθος του tag array για μια αρχιτεκτονική των 64 bits. Αν για την ίδια αρχιτεκτονική αλλάξουμε την οργάνωση της cache σε direct-mapped (διατηρώντας σταθερή τη χωρητικότητα και το μέγεθος του cache line) ποιο θα είναι το νέο μέγεθος του tag array;

$$sets = 64 \rightarrow index = 6\ bits$$
$$block\_size = \frac{size}{sets*assoc} = \frac{16*2^{10}}{64*8} = 32\ bytes \rightarrow block\_offset = 5\ bits$$

Άρα $tag = 64 - 6 - 5 = 53\ bits$. Κάθε block έχει ένα tag και άρα το συνολικό μέγεθος του tag array είναι $tag\_array\_size = 53 * 64 * 8 = 27136\ bits = 3392\ bytes = 3.3125\ KB$.

Αφού διατηρούμε σταθερό το μέγεθος του cache line (άρα και το block_offset) και τη χωρητικότητα της cache, θα διατηρηθεί σταθερός και ο αριθμός των blocks. Επειδή πλέον όμως η cache είναι direct-mapped, έχουμε πλέον 64*8 = 512 sets και άρα χρειαζόμαστε 9 bits για το index. Επομένως το πεδίο tag μειώνεται σε 50 bits και το νέο μέγεθος του tag array θα είναι

$$tag\_array\_size = 50 * 512 = 25600\ bits = 3200\ bytes = 3.125\ KB.$$

Δ. Αναφέρατε δύο λόγους για τους οποίους είναι επιθυμητή η χρήση εικονικής μνήμης. Εξηγήστε ποιο το πλεονέκτημα της χρήσης μιας virtually-indexed, physically-tagged L1 cache σε σχέση με μια physically-indexed, physically tagged L1 cache.

1) Isolation : Προστασία των διεργασιών από προσβάσεις άλλων διεργασιών στη μνήμη τους.
2) Relocation : Η virtual memory επιτρέπει σε κάθε πρόγραμμα να τρέξει οπουδήποτε στη φυσική μνήμη.

Στην virtually-indexed, physically-tagged cache, το index βρίσκεται από τη virtual address χωρίς μετάφραση. Επομένως, μπορεί το σύστημα να ψάξει το block στην cache παράλληλα με την πρόσβαση στο TLB, το οποίο θα δώσει το φυσικό tag. Με αυτό τον τρόπο, μειώνεται το hit time σε σχέση με τη physically-indexed, physically-tagged cache, όπου πρέπει πρώτα να γίνει η μετάφραση από το TLB και μετά να γίνει η αναζήτηση στην cache.

Θέμα 4ο (25%)

Δίνεται ο παρακάτω κώδικας γραμμένος σε C.

```
double A[11][12], B[11][12];

for(i = 0; i < 5; i++)
      for(j = 0; j < 12; j++)
            A[i][j] = A[i][j]*B[i][j%4];
```

Οι πίνακες περιέχουν στοιχεία κινητής υποδιαστολής διπλής ακρίβειας, μεγέθους 8 bytes το καθένα. 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-mapped, write-allocate και έχει χωρητικότητα 1KB. Το μέγεθος του block είναι 32 bytes, ενώ η μικρότερη μονάδα δεδομένων που μπορεί να διευθυνσιοδοτηθεί είναι το 1 byte.
2. Υποθέτουμε ότι όλες οι μεταβλητές, πλην των στοιχείων των πινάκων, μπορούν να αποθηκευτούν σε καταχωρητές του επεξεργαστή, οπότε οποιαδήποτε αναφορά σε αυτές δεν συνεπάγεται προσπέλαση στην κρυφή μνήμη. Επίσης, δήλωση συνεχόμενων μεταβλητών συνεπάγεται αποθήκευση σε διαδοχικές θέσεις στη μνήμη. Τέλος, σε επίπεδο εντολών assembly οι αναγνώσεις γίνονται με τη σειρά που εμφανίζονται στον κώδικα.
3. Οι πίνακες είναι ευθυγραμμισμένοι και αποθηκευμένοι στην κύρια μνήμη κατά γραμμές.

Α. Βρείτε ποιες από τις αναφορές στα στοιχεία των πινάκων, για όλη την εκτέλεση του παραπάνω κώδικα, καταλήγουν σε misses. Υπολογίστε το ρυθμό αστοχίας. Υποδείξτε τα compulsory misses.

Κάθε block περιέχει 4 στοιχεία από έναν πίνακα. Ο πίνακας είναι ευθυγραμμισμένος και επομένως σε κάποιο set x της cache γίνονται mapped τα A[0][0]-A[0][3]. Κάθε γραμμή του πίνακα έχει 12 στοιχεία, επομένως χρειάζεται 3 blocks. Άρα συνολικά, ο πίνακας A χρειάζεται 11 * 3 = 33 blocks (ή sets μιας και η cache είναι direct-mapped). Η cache όμως έχει 1024/32 = 32 block και άρα το τελευταίο block του A που περιέχει τα A[10][8]-A[10][11] θα γίνεται και αυτό mapped στο set x όπου γίνονται mapped τα A[0][0]-A[0][3].

Οι πίνακες είναι αποθηκευμένοι σε συνεχόμενες θέσεις στη μνήμη, κι επομένως το πρώτο block του B (B[0][0]-B[0][3]) θα γίνεται mapped στο επόμενο set από εκεί που τελειώνει ο πίνακας A, δηλαδή στο set x+1. Άρα έχουμε conflicts μεταξύ A[0][4]-A[0][7] και B[0][0]-B[0][3], A[0][8]-A[0][11] και B[0][4]-B[0][7], A[1][0]-A[1][3] και B[0][8]-B[0][1], κτλ.

Στον κώδικα έχουμε 2 αναγνώσεις και μετά μια εγγραφή. Για i = 0, έχουμε:

$A_{0,0}$ $B_{0,0}$ $A_{0,0}$	*m m* h	$A_{0,4}$ $B_{0,0}$ $A_{0,4}$	*m* m m	$A_{0,8}$ $B_{0,0}$ $A_{0,8}$	*m* m h
$A_{0,1}$ $B_{0,1}$ $A_{0,1}$	h h h	$A_{0,5}$ $B_{0,1}$ $A_{0,5}$	h m m	$A_{0,9}$ $B_{0,1}$ $A_{0,9}$	h h h
$A_{0,2}$ $B_{0,2}$ $A_{0,2}$	h h h	$A_{0,6}$ $B_{0,2}$ $A_{0,6}$	h m m	$A_{0,10}$ $B_{0,2}$ $A_{0,10}$	h h h
$A_{0,3}$ $B_{0,3}$ $A_{0,3}$	h h h	$A_{0,7}$ $B_{0,3}$ $A_{0,7}$	h m m	$A_{0,11}$ $B_{0,3}$ $A_{0,11}$	h h h

Δηλαδή έχουμε 2+9+2 = 13 misses, εκ των οποίων 4 είναι compulsory (σε italics).
Ακριβώς το ίδιο pattern επαναλαμβάνεται και για i = 1,2,3,4. Ο ρυθμός αστοχίας λοιπόν είναι ίσος με 13/36 = 36.11% και συνολικά έχουμε 20 compulsory misses.

Β. Αν η κρυφή μνήμη γίνει write-no-allocate, διατηρώντας τα υπόλοιπα χαρακτηριστικά της ίδια, ποιός θα είναι ο νέος ρυθμός αστοχίας;

Αν η μνήμη γίνει write-no-allocate, τότε σε περίπτωση write-miss δεν έρχεται το block στην cache. Επομένως για i = 0, έχουμε:

$A_{0,0}$ $B_{0,0}$ $A_{0,0}$	*m m* h	$A_{0,4}$ $B_{0,0}$ $A_{0,4}$	*m* m m	$A_{0,8}$ $B_{0,0}$ $A_{0,8}$	*m* h h
$A_{0,1}$ $B_{0,1}$ $A_{0,1}$	h h h	$A_{0,5}$ $B_{0,1}$ $A_{0,5}$	m m m	$A_{0,9}$ $B_{0,1}$ $A_{0,9}$	h h h
$A_{0,2}$ $B_{0,2}$ $A_{0,2}$	h h h	$A_{0,6}$ $B_{0,2}$ $A_{0,6}$	m m m	$A_{0,10}$ $B_{0,2}$ $A_{0,10}$	h h h
$A_{0,3}$ $B_{0,3}$ $A_{0,3}$	h h h	$A_{0,7}$ $B_{0,3}$ $A_{0,7}$	m m m	$A_{0,11}$ $B_{0,3}$ $A_{0,11}$	h h h

Δηλαδή έχουμε 2+12+1 = 15 misses.
Ακριβώς το ίδιο pattern επαναλαμβάνεται και για i = 1,2,3,4. Ο νέος ρυθμός αστοχίας λοιπόν θα είναι ίσος με 15/36 = 41.67%.

Γ. Αν η κρυφή μνήμη αντικατασταθεί με μια κρυφή μνήμη 2-way associative, write-allocate, με ίδιο συνολικό αριθμό blocks δεδομένων και ίδιο μέγεθος blocks, ποιος θα είναι ο νέος ρυθμός αστοχίας; Υποθέστε ότι η cache χρησιμοποιεί LRU πολιτική αντικατάστασης.

Αφού η cache γίνεται 2-way associative με τον ίδιο αριθμό blocks και το ίδιο block size, ο αριθμός των sets θα υποδιπλασιαστεί και έτσι θα έχουμε τώρα 16 sets. Αφού το block size παραμένει σταθερό, ο Α θα χρειάζεται και πάλι 33 sets και έτσι τα blocks του Β θα εξακολουθήσουν να κάνουν conflict με τα blocks του Α όπως και πριν. Συγκεκριμένα τα conflicts, θα είναι τώρα :

- A[0][4]-A[0][7], A[5][8]-A[5][11], B[0][0]-B[0][3], B[5][4]-B[5][7]
- A[0][8]-A[0][11], A[6][0]-A[6][3], B[0][4]-B[0][7], B[5][8]-B[5][11], κτλ.

Έτσι για i = 0, έχουμε:

$A_{0,0}$ $B_{0,0}$ $A_{0,0}$	*m m* h	$A_{0,4}$ $B_{0,0}$ $A_{0,4}$	*m* h h	$A_{0,8}$ $B_{0,0}$ $A_{0,8}$	*m* h h
$A_{0,1}$ $B_{0,1}$ $A_{0,1}$	h h h	$A_{0,5}$ $B_{0,1}$ $A_{0,5}$	h h h	$A_{0,9}$ $B_{0,1}$ $A_{0,9}$	h h h
$A_{0,2}$ $B_{0,2}$ $A_{0,2}$	h h h	$A_{0,6}$ $B_{0,2}$ $A_{0,6}$	h h h	$A_{0,10}$ $B_{0,2}$ $A_{0,10}$	h h h
$A_{0,3}$ $B_{0,3}$ $A_{0,3}$	h h h	$A_{0,7}$ $B_{0,3}$ $A_{0,7}$	h h h	$A_{0,11}$ $B_{0,3}$ $A_{0,11}$	h h h

Δηλαδή έχουμε 2+1+1 = 4 misses.
Ακριβώς το ίδιο pattern επαναλαμβάνεται και για i = 1,2,3,4. Ο νέος ρυθμός αστοχίας λοιπόν θα είναι ίσος με 4/36 = 11.11%.