ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

www.cslab.ece.ntua.gr

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ
Εξετάσεις Μαρτίου 2009

Θέμα 1ο (25 %)

Α(6%). Δίνεται επεξεργαστής με αρχιτεκτονική σωλήνωσης. Πώς θα επηρεαστούν τα μεγέθη Instructions/Program, Cycles/Instruction και Seconds/Cycle αν γίνει ξεχωριστά το καθένα από τα παρακάτω; Θα αυξηθούν, θα μειωθούν ή θα μείνουν αμετάβλητα; Γιατί;

(i) Ένωση 2 σταδίων της σωλήνωσης σε 1.

Instructions/Program : Δε μεταβάλεται μιας και δεν πραγματοποιείται καμιά αλλαγή στο instruction set και το πρόγραμμα παραμένει το ίδιο.
Cycles/Instruction : Μειώνεται, καθώς η κάθε εντολή χρειάζεται πλέον λιγότερα στάδια (και άρα κύκλους). Επίσης, μειώνεται και η πιθανότητα εμφάνισης κινδύνων μεταξύ των εντολών.
Seconds/Cycle : Μπορεί να αυξηθεί. Αν το νέο στάδιο που προκύπτει επηρεάζει το critical path (ή αλλιώς απαιτεί μεγαλύτερο κύκλο από αυτόν πριν την αλλαγή), τότε το cycle time θα πρέπει να αυξηθεί.

(ii) Αύξηση της συχνότητας του ρολογιού.

Instructions/Program : Καμία μεταβολή μιας και δεν υπάρχει καμιά αλλαγή στο instruction set και άρα στο πρόγραμμα.
Cycles/Instruction : Καμία αλλαγή, μιας και δεν αλλάζει τίποτα στο pipeline.
Seconds/Cycle : Θα μειωθεί μιας και αύξηση της συχνότητας σημαίνει ότι ό κάθε κύκλος θα χρειάζεται λιγότερα seconds.

(iii) Χρησιμοποιήση ενός καλύτερου compiler.

Instructions/Program : Συνήθως μειώνεται, μιας και οι καλύτεροι compilers έχουν τη δυνατότητα να παράγουν πιο "συμπυκνωμένο" κώδικα. Υπάρχει όμως και η πιθανότητα να αυξηθούν, αν ο compiler παρατηρήσει πως πιο απλές εντολές μειώνουν τα hazards.
Cycles/Instruction : Μπορεί να μειωθεί, αν ο compiler καταφέρει να αποφύγει hazards (αναδιατάσσοντας πχ κάποιες εντολές).
Seconds/Cycles : Καμία αλλαγή.

Β(9%). Ως επικεφαλής σχεδιασμού του επεξεργαστή MIPS αποφασίζετε την αναδιάταξη των σταδίων της σωλήνωσης, τοποθετώντας το στάδιο MEM πριν το στάδιο EX. Η καινούρια σωλήνωση διαθέτει και αυτή πλήρες σχήμα προώθησης.

(i) Αλλάζει ο τρόπος διευθυνσιοδότησης των δεδομένων στη μνήμη; Αν ναι, πώς;

Πλέον το σύστημα θα υποστηρίζει απλά register indirect addressing. Δεν υπάρχει πλέον η δυνατότητα χρήσης κάποιου offset (το οποίο στην προηγούμενη περίπτωση μπορούσε να προστεθεί στη τιμή του καταχωρητή.

(ii) Δώστε ένα παράδειγμα αλληλουχίας εντολών, όπου η κλασική σωλήνωση του MIPS θα εισήγαγε καθυστερήσεις (stalls) ενώ η καινούρια όχι.

LW **r2**, 0(r3)
ADD r4, **r2**, r5

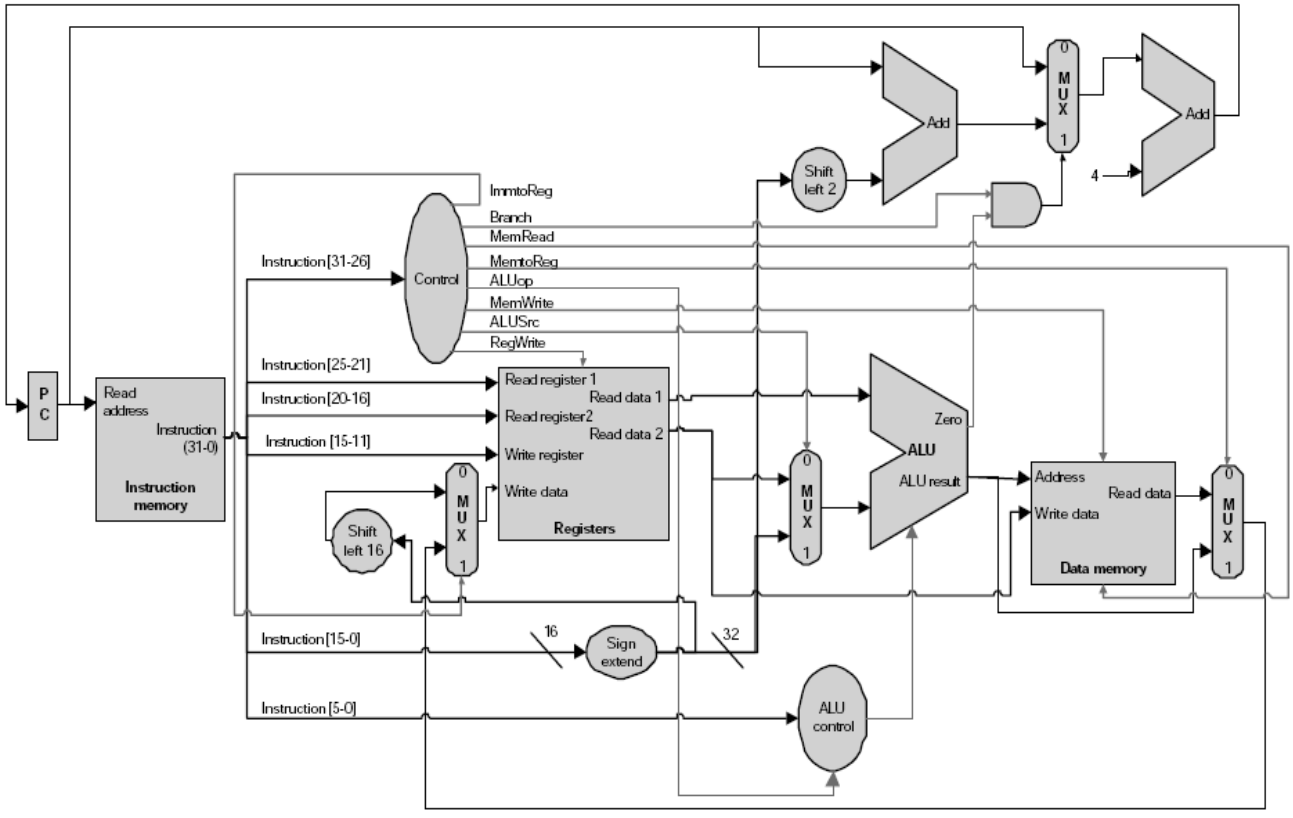
Στο κλασσικό pipeline η ADD θα είχε 1 stall. Αντίθετα στην καινούρια 0.

(iii) Δώστε ένα παράδειγμα αλληλουχίας εντολών, όπου η καινούρια σωλήνωση του MIPS θα εισήγαγε καθυστερήσεις (stalls) ενώ η κλασική όχι.

ADD **r2**, r3, r4
LW r5, 0(r2)

Στο καινούριο pipeline θα έχουμε 1 stall, ενώ στο παλιό 0.

Γ(10%). Δίνεται η παρακάτω δίοδος δεδομένων (datapath) καθώς και μια σειρά εντολών μαζί με τις δομές τους και το μήκος των πεδίων τους. Μπορεί να εκτελεστεί κάθε μια από τις εντολές αυτές σε αυτό το datapath; Αν όχι, γιατί;



(i) `add rd, rs, rt`

0x0	rs	rt	rd	0	0x20
6	5	5	5	5	6

Εκτελείται κανονικά.

(ii) `lw rt, offset(rs)`

0x23	rs	rt	offset
6	5	5	16

Δεν μπορεί να εκτελεστεί, γιατί ο rt (εδώ bits 17-21) δεν χρησιμοποιείται σαν Write Register στο Register File (λείπει ο MUX του κλασσικού datapath).

(iii) `j target`

0x2	target
6	26

Δεν μπορεί να εκτελεστεί, γιατί το target δεν φορτώνεται στο PC.

(iv) `lui rt, imm`

0xf	0x0	rt	imm
6	5	5	16

Δεν μπορεί να εκτελεστεί, γιατί ο rt (εδώ bits 17-21) δεν χρησιμοποιείται σαν Write Register στο Register File (λείπει ο MUX του κλασσικού datapath).

(v) `bne rs, rt, label`

0x5	rs	rt	label
6	5	5	16

Δεν μπορεί να εκτελεστεί. Το datapath αυτό μπορεί να κάνει μόνο branch on equal.

Θέμα 2ο (30 %)

Υποθέτουμε ότι έχουμε αρχιτεκτονική σωλήνωσης αποτελούμενη από τα εξής στάδια: IF ID AGU MEM ALU WB. Το *AGU* (address generation unit) χρησιμοποιείται για τον υπολογισμό τελικών διευθύνσεων μνήμης (effective address calculation), καθώς και για τον υπολογισμό διευθύνσεων-στόχων (targets) σε εντολές διακλάδωσης. Το ALU χρησιμοποιείται για όλους τους υπόλοιπους υπολογισμούς, καθώς και για την επίλυση των διακλαδώσεων υπό συνθήκη. Επιπλέον, οι αριθμητικές εντολές έχουν τη δυνατότητα να προσπελαύνουν απευθείας μια θέση μνήμης. Όλα τα στάδια διαρκούν 1 κύκλο ρολογιού.

Α(6%). Βρείτε όλες τις εξαρτήσεις στο ακόλουθο κομμάτι κώδικα καθώς και την κατηγορία στην οποία ανήκουν (true dependence, output dependence, anti-dependence, control dependence).

```
1.        Repeat:       lw $1,100($2)
2.                      add $3,$1,100($4)
3.                      or $6,$5,$3
4.                      sw $6,100($2)
5.                      mul $4,$4,#100
6.                      sub $2,$2,#8
7.                      bne $2, $0, Repeat
8.                      sub $1,$1,#100
```

True dependence	Output dependence	Anti-dependence	Control dependence
2 από την 1 (στον $1) 3 από την 2 (στον $3) 4 από την 3 (στον $6) 7 από την 6 (στον $2) 8 από την 1 (στον $1)	8 από την 1 (στον $1)	5 από την 2 (στον $4) 6 από την 1 (στον $2) 6 από την 4 (στον $2) 8 από την 2 (στον $1)	1,2,3,4,5,6,8 από την 7

Β(9%). Υποθέστε ότι δεν υπάρχει σχήμα προώθησης. Πόσοι κύκλοι απαιτούνται για την εκτέλεση μιας επανάληψης του loop; Ο καταχωρητής $2 έχει τέτοια τιμή ώστε το branch να είναι TAKEN.

Κύκλος	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*
lw $1,100($2)	IF	ID1	AG	M	AL	WB													
add $3,$1,100($4)		IF	ID	-	-	-	AG	M	AL	WB									
or $6,$5,$3			IF	-	-	-	ID	-	-	-	AG	M	AL	WB					
sw $6,100($2)							IF	-	-	-	ID	-	-	-	AG	M	AL	WB	
mul $4,$4,#100											IF	-	-	-	ID	AG	M	AL	WB
sub $2,$2,#8															IF	ID	AG	M	AL
bne $2,$0,Repeat																IF	ID	-	-
lw $1,100($2)																	-	-	-

Κύκλος	*20*	*21*	*22*	*23*	*24*
lw $1,100($2)					
add $3,$1,100($4)					
or $6,$5,$3					
sw $6,100($2)					
mul $4,$4,#100					
sub $2,$2,#8	WB				
bne $2,$0,Repeat	-	AG	M	AL	WB
lw $1,100($2)	-	-	-	-	IF

Συνολικά απαιτούνται 24 κύκλοι για την εκτέλεση μιας επανάληψης.

Γ(9%). Υποθέστε τώρα ότι υπάρχουν όλα τα δυνατά σχήματα προώθησης. Πόσοι κύκλοι απαιτούνται για την εκτέλεση μιας επανάληψης του loop; Ο καταχωρητής $2 έχει τέτοια τιμή ώστε το branch να είναι TAKEN.

Κύκλος	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*2*
lw $1,100($2)	IF	ID	AG	M	AL	WB															
add $3,$1,100($4)		IF	ID	AG	M	AL	WB														
or $6,$5,$3			IF	ID	AG	M	AL	WB													
sw $6,100($2)				IF	ID	AG	-	M	AL	WB											
mul $4,$4,#100					IF	ID	-	AG	M	AL	WB										
sub $2,$2,#8						IF	-	ID	AG	M	AL	WB									
bne $2,$0,Repeat								IF	ID	AG	M	AL	WB								
lw $1,100($2)									-	-	-	-	IF	ID	AG	M	AL	WB			

Στο παραπάνω διάγραμμα, τα κανονικά βέλη δείχνουν τις προωθήσεις που γίνονται, ενώ τα διακεκομμένα δείχνουν τα στάδια όπου χρησιμοποιούνται οι προωθούμενες τιμές.
Στην πρώτη προώθηση, η τιμή που θα αποθηκευτεί στον $1 (και η οποία γίνεται διαθέσιμη στο τέλος του 4ου κύκλου) προωθείται στις εισόδους του ίδιου σταδίου (MEM), έτσι ώστε να προχωρήσει στο pipeline και να χρησιμοποιηθεί όταν ακριβώς την χρειάζεται η add, δηλαδή στον μεθεπόμενο κύκλο, στο στάδιο ALU.
Στη δεύτερη περίπτωση προώθησης, ο $3 προωθείται από την έξοδο της ALU σε κάποια από τις εισόδους της, ώστε να χρησιμοποιηθεί στον επόμενο κύκλο από την mul.
Στην τρίτη περίπτωση, η τιμή του $6 προωθείται αμέσως μόλις γίνει διαθέσιμη (τέλος 7ου κύκλου) στις εισόδους του σταδίου MEM. Δεν μπορεί να γίνει διαθέσιμη νωρίτερα, γι'αυτό και αναγκαστικά εισάγεται stall στον 7ο κύκλο για την εντολή sw $6,100($2) καθώς και όσες επόμενες βρίσκονται στο pipeline.
Ομοίως, στην τέταρτη περίπτωση, η τιμή του $2 προωθείται από την έξοδο της ALU σε κάποια από τις εισόδους της.
Συνολικά απαιτούνται 13 κύκλοι για την εκτέλεση μιας επανάληψης.

Δ(6%). Θεωρώντας την ίδια σωλήνωση με το ερώτημα γ, μπορείτε να επιτύχετε ακόμα καλύτερη επίδοση για την εκτέλεση μιας επανάληψης του loop αναδιατάσσοντας εντολές;

Μια αναδιάταξη των εντολών η οποία διατηρεί τη σημασιολογία του κώδικα (σωστή ροή εντολών) και η οποία αποφεύγει την ανάγκη εισαγωγής stalls είναι η ακόλουθη:

```
Repeat:
lw $1,100($2)
add $3,$1,100($4)
or $6,$5,$3
mul $4,$4,#100
sw $6,100($2)
sub $2,$2,#8
bne $2, $0, Repeat
```

Η λογική είναι να εκμεταλλευτούμε το stall της προηγούμενης περίπτωσης παρεμβάλλοντας την ανεξάρτητη εντολή mul. Έτσι γλιτώνουμε έναν κύκλο εκτέλεσης.

Θέμα 3ο (20 %)

Α(5%). Εξηγήστε συνοπτικά τις πολιτικές write-through, write-back, write-allocate, no-write-allocate. Γιατί ο συνδυασμός write-through με write-allocate δεν προτιμάται στην πράξη;

write-through : Όταν γίνεται εγγραφή, ενημερώνονται αμέσως και τα χαμηλότερα επίπεδα μνήμης.
write-back : Όταν γίνεται η εγγραφή ενημερώνεται μόνο το επίπεδο μνήμης στο οποίο βρίσκεται το block. Όταν το block αυτό απομακρυνθεί από το επίπεδο αυτό, τότε πρέπει να ενημερωθούν και τα χαμηλότερα επίπεδα.
write-allocate : Σε περίπτωση write-miss, το block ενημερώνεται στη μνήμη και μεταφέρεται στη cache.
write-no-allocate: Σε περίπτωση ενός write-miss, το block ενημερώνεται στη μνήμη αλλά δε μεταφέρεται στην cache.

Ο συνδυασμός αυτός δεν προτιμάται στην πράξη καθώς δεν προσφέρει κάποιο κέρδος. Ακόμα και αν υπάρχουν επόμενες εγγραφές στην ίδια διεύθυνση, οι εγγραφές αυτές θα πρέπει να ενημερώνουν όλα τα επίπεδα της μνήμης.

Β(10%). Δίνεται μια κρυφή μνήμη 64KB με blocks των 128 bytes. Οι διευθύνσεις έχουν μήκος 32 bits και η μικρότερη μονάδα διευθυνσιοδότησης είναι το 1 byte. Σχεδιάστε το συνολικό σύστημα της κρυφής μνήμης για τις παρακάτω οργανώσεις :

(i) direct-mapped

TAG	INDEX	BLOCK OFFSET
16	9	7

+σχήμα

(ii) 4-way set associative

TAG	INDEX	BLOCK OFFSET
18	7	7

+σχήμα

Γ(5%). Έστω σύστημα εικονικής μνήμης. Είναι δυνατόν μια αναζήτηση διεύθυνσης να προκαλέσει *ευστοχία (hit) στη κρυφή μνήμη αναζήτησης μετάφρασης (TLB)* και ταυτόχρονα *σφάλμα σελίδας (page fault)*;

Όχι δεν είναι δυνατόν. Αν έχουμε hit στο TLB σημαίνει ότι υπάρχει και η αντίστοιχη σελίδα. Αν αυτή η σελίδα για κάποιο λόγο δεν υπάρχει, θα πρέπει να έχει γίνει invalid και το αντίστοιχο entry στο TLB.

Θέμα 4ο (25 %)

Θεωρήστε την εκτέλεση του ακόλουθου τμήματος κώδικα γραμμένο σε γλώσσα C:

```
double y[32][160], x[160];
j=2;
for (i=0; i<128; i++)
     x[i] = x[i+2] + y[j][i+32];
```

Κάθε στοιχείο των πινάκων x και y είναι 8 bytes. Έστω ότι για τα στοιχεία x[0] και y[0][0], οι διευθύνσεις του πρώτου στοιχείου τους είναι οι 0x00004000 και 0x00008300, αντίστοιχα. Υποθέτουμε ότι έχουμε ένα επίπεδο κρυφής μνήμης δεδομένων. Επιπλέον, οι εντολές σε επίπεδο assembly του παραπάνω βρόχου είναι διατεταγμένες με τέτοιο τρόπο ώστε γίνεται πρώτα ανάγνωση του x και έπειτα του y, οι οποίοι είναι αποθηκευμένοι κατά γραμμές.

Για κάθε μία από τις ακόλουθες περιπτώσεις οργάνωσης της cache, βρείτε τον αριθμό από read hits, read misses, write hits και write misses για τις διάφορες αναφορές που γίνονται στη μνήμη, εξηγώντας σε κάθε περίπτωση τις απαντήσεις σας. Υποδείξτε τις κατηγορίες στις οποίες εντάσσονται τα misses που βρίσκετε (compulsory, conflict, capacity).

Α(9%). Κρυφή μνήμη μεγέθους 1KB, συσχέτισης 2 δρόμων (2-way set associative), write allocate, με μέγεθος block ίσο με 32 bytes και πολιτική αντικατάστασης LRU.

Η cache θα έχει συνολικά 1KB/32B = 32 blocks, και επομένως 16 sets.
Αφού το μέγεθος του block είναι 32 bytes, το block offset θα είναι log2(32)=5 bits.
Ομοίως, αφού έχουμε 16 sets το index θα είναι log2(16)=4 bits.
Η διεύθυνση του πρώτου στοιχείου του x[0] είνα η 0x4000= 0100 0000 0000 0000, που σημαίνει ότι θα απεικονιστεί στο set 0, και επιπλέον θα βρίσκεται στην αρχή του block.
Η διεύθυνση του y[2][32] είναι η $0x8300 + (160*2*8 + 32*8)_{10} = 0x8E00 = 1000\ 1110\ 0000\ 0000$, που σημαίνει ότι θα απεικονιστεί και αυτό στο set 0.
Κάθε block χωράει 32/8=4 διαδοχικά στοιχεία των πινάκων, τα οποία έρχονται όλα μαζί στην cache με την αναφορά οποιουδήποτε στοιχείου από αυτά.

Άρα, η ακολουθία αναφορών, μαζί με το αποτέλεσμα της κάθε αναφοράς, είναι η ακόλουθη:

$x_2\ y_{2,32}\ x_0$	m m h
$x_3\ y_{2,33}\ x_1$	h h h
$x_4\ y_{2,34}\ x_2$	m h h
$x_5\ y_{2,35}\ x_3$	h h h
$x_6\ y_{2,36}\ x_4$	h m h
$x_7\ y_{2,37}\ x_5$	h h h
$x_8\ y_{2,38}\ x_6$	m h h
$x_9\ y_{2,39}\ x_7$	h h h
$x_{10}\ y_{2,40}\ x_8$	h m h
$x_{11}\ y_{2,41}\ x_9$	h h h

x_{12} $y_{2,42}$ x_{10} m h h
x_{13} $y_{2,43}$ x_{11} h h h
x_{14} $y_{2,44}$ x_{12} h m h
x_{15} $y_{2,45}$ x_{13} h h h
...
x_{124} $y_{2,154}$ x_{122} m h h
x_{125} $y_{2,155}$ x_{123} h h h
x_{126} $y_{2,156}$ x_{124} h m h
x_{127} $y_{2,157}$ x_{125} h h h

x_{128} $y_{2,158}$ x_{126} m h h
x_{129} $y_{2,159}$ x_{127} h h h

Σύνολο: 2 + 31*2 + 1 = 65 misses.

65 read misses, 191 read hits, 128 write hits, 0 write misses

Δεν έχουμε conflicts, αφού τα blocks των x και y απεικονίζονται μεν στο ίδιο set, αλλά σε διαφορετικά cache lines. Capacity misses δεν έχουμε, διότι αν και στα μισά του loop γεμίζει η cache από δεδομένα των x και y, δεν υπάρχει επαναχρησιμοποίηση δεδομένων που θα μπορούσε να ευνοηθεί από cache μεγαλύτερης χωρητικότητας. Δηλαδή, όλα τα misses είναι compulsory.

Β(8%). Κρυφή μνήμη μεγέθους 512 bytes, ευθείας αντιστοίχισης (direct mapped), write allocate, με μέγεθος block ίσο με 32 bytes.

Η cache θα έχει συνολικά 512B/32B = 16 blocks.
Αφού το μέγεθος του block είναι 32 bytes, και υποθέτοντας ότι η μικρότερη μονάδα δεδομένων που μπορεί να διευθυνσιοδοτηθεί είναι το 1 byte, το block offset θα είναι log2(32)=5 bits.
Ομοίως, αφού έχουμε 16 blocks το index θα είναι log2(16)=4 bits.
Η διεύθυνση του πρώτου στοιχείου του x[0] είνα η 0x4000= 0100 0000 0000 0000, που σημαίνει ότι θα απεικονιστεί στο block 0.
Η διεύθυνση του y[2][32] είναι η $0x8300 + (160*2*8 + 32*8)_{10} = 0x8E00 = 1000\ 1110\ 0000\ 0000$, που σημαίνει ότι θα απεικονιστεί και αυτό στο block 0.

Άρα, η ακολουθία αναφορών, μαζί με το αποτέλεσμα της κάθε αναφοράς, είναι η ακόλουθη:

x_2 $y_{2,32}$ x_0 m m m
x_3 $y_{2,33}$ x_1 h m m

x_4 $y_{2,34}$ x_2 m m m
x_5 $y_{2,35}$ x_3 h m m
x_6 $y_{2,36}$ x_4 h m m
x_7 $y_{2,37}$ x_5 h m m

x_8 $y_{2,38}$ x_6 m m m
x_9 $y_{2,39}$ x_7 h m m
x_{10} $y_{2,40}$ x_8 h m m
x_{11} $y_{2,41}$ x_9 h m m

x_{12} $y_{2,42}$ x_{10} m m m
x_{13} $y_{2,43}$ x_{11} h m m
x_{14} $y_{2,44}$ x_{12} h m m
x_{15} $y_{2,45}$ x_{13} h m m

...

x_{124} $y_{2,154}$ x_{122}	m m m
x_{125} $y_{2,155}$ x_{123}	h m m
x_{126} $y_{2,156}$ x_{124}	h m m
x_{127} $y_{2,157}$ x_{125}	h m m
x_{128} $y_{2,158}$ x_{126}	m m m
x_{129} $y_{2,159}$ x_{127}	h m m

Σύνολο: 5 + 31*9 + 5 = 289 misses.

95 read hits, 161 read misses, 0 write hits, 128 write misses

Όλα τα misses της αναφοράς x[i+2] είναι compulsory. Compulsory είναι επίσης τα misses για τις αναφορές του y, y[2][32], y[2][36], y[2][40], y[2][44], ..., καθώς σε αυτές τις αναφορές φορτώνεται για πρώτη φορά το αντίστοιχο block του y. Όλα τα υπόλοιπα misses της αναφοράς στον y, καθώς και της αναφοράς x[i], είναι conflict, αφού τα αντίστοιχα blocks έχουν φορτωθεί σε προηγούμενες χρονικές στιγμές, αλλά λόγω απεικόνισης στο ίδιο block, έχουν αλληλο-εκτοπιστεί από την cache. Και σε αυτή την περίπτωση δεν υπάρχουν capacity misses για τον ίδιο λόγο όπως και στο προηγούμενο ερώτημα.

Γ(8%). Κρυφή μνήμη μεγέθους 512 bytes, ευθείας αντιστοίχισης, no-write allocate, με μέγεθος block ίσο με 32 bytes.

Σε αυτή την περίπτωση, οι εγγραφές στον x οι οποίες αποτυγχάνουν, δε θα φέρνουν το αντίστοιχο block στην cache αλλά θα πηγαίνουν κατευθείαν στη μνήμη.
Όσον αφορά τις απεικονίσεις των πρώτων στοιχείων των πινάκων, ισχύουν τα ίδια με το προηγούμενο ερώτημα.

Η ακολουθία αναφορών, μαζί με το αποτέλεσμα της κάθε αναφοράς, είναι η ακόλουθη:

x_2 $y_{2,32}$ x_0	m m m
x_3 $y_{2,33}$ x_1	m m m
x_4 $y_{2,34}$ x_2	m h m
x_5 $y_{2,35}$ x_3	h h m
x_6 $y_{2,36}$ x_4	h m m
x_7 $y_{2,37}$ x_5	m m m
x_8 $y_{2,38}$ x_6	m h m
x_9 $y_{2,39}$ x_7	h h m
x_{10} $y_{2,40}$ x_8	h m m
x_{11} $y_{2,41}$ x_9	m m m
x_{12} $y_{2,42}$ x_{10}	m h m
x_{13} $y_{2,43}$ x_{11}	h h m
x_{14} $y_{2,44}$ x_{12}	h m m
x_{15} $y_{2,45}$ x_{13}	m m m
...	
x_{124} $y_{2,154}$ x_{122}	m h m
x_{125} $y_{2,155}$ x_{123}	h h m
x_{126} $y_{2,156}$ x_{124}	h m m
x_{127} $y_{2,157}$ x_{125}	m m m

x_{128} $y_{2,158}$ x_{126}	m h m
x_{129} $y_{2,159}$ x_{127}	h h m

Σύνολο: 6 + 31*8 + 3 = 257 misses.

127 read hits, 129 read misses, 0 write hits, 128 write misses

Τα misses x[2], x[4], x[8], x[12], ... της αναφοράς x[i+2] είναι compulsory, καθώς σε αυτές τις αναφορές φορτώνεται για πρώτη φορά το αντίστοιχο block του x. Compulsory είναι επίσης τα misses για τις αναφορές του y, y[2][32], y[2][36], y[2][40], y[2][44], ..., για τον ίδιο λόγο. Όλα τα υπόλοιπα misses στα x και y είναι conflict, αφού τα αντίστοιχα blocks έχουν φορτωθεί σε προηγούμενες χρονικές στιγμές, αλλά λόγω απεικόνισης στο ίδιο block, έχουν αλληλο-εκτοπιστεί από την cache. Και σε αυτή την περίπτωση δεν υπάρχουν capacity misses για τον ίδιο λόγο που αναφέραμε στο ερώτημα α.