ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

www.cslab.ntua.gr

3η ΑΣΚΗΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2013-2014, 5ο εξάμηνο Σχολή ΗΜ&ΜΥ

Ημερομηνία Παράδοσης: **09/03/2014**

Απορίες: **ca2013-2014t2@cslab.ece.ntua.gr**

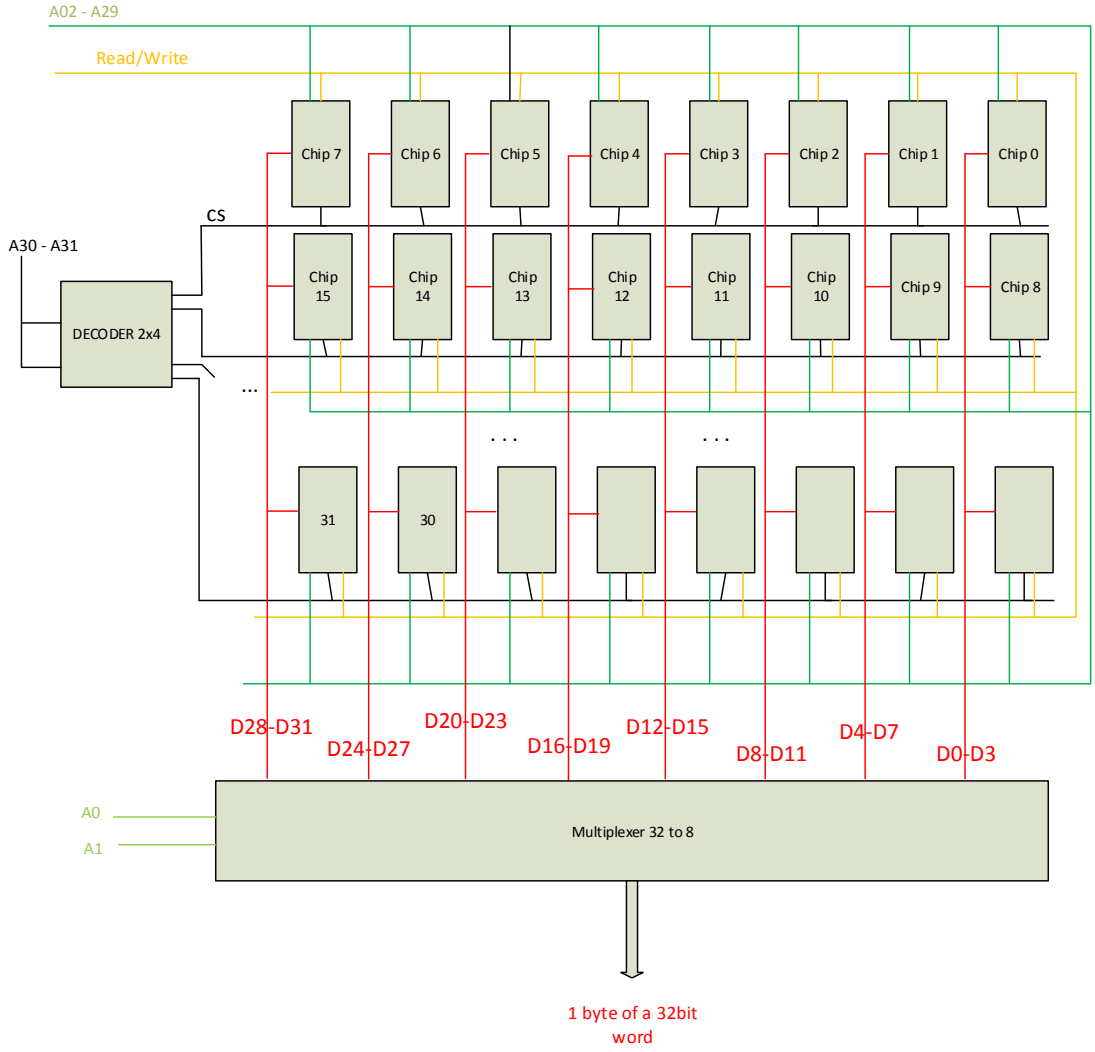
Θέμα 1ο

Να σχεδιαστεί πλήρης μονάδα μνήμης για υπολογιστικό σύστημα MIPS με διευθύνσεις 32 bit και λέξεις 32 bit, χρησιμοποιώντας ψηφίδες μνήμης RAM 256M x 4 bit. Κάθε ψηφίδα RAM διαθέτει, εκτός από τις αντίστοιχες γραμμές διευθύνσεων και δεδομένων, και τις εισόδους R/W και CS (chip select).

Λύση:

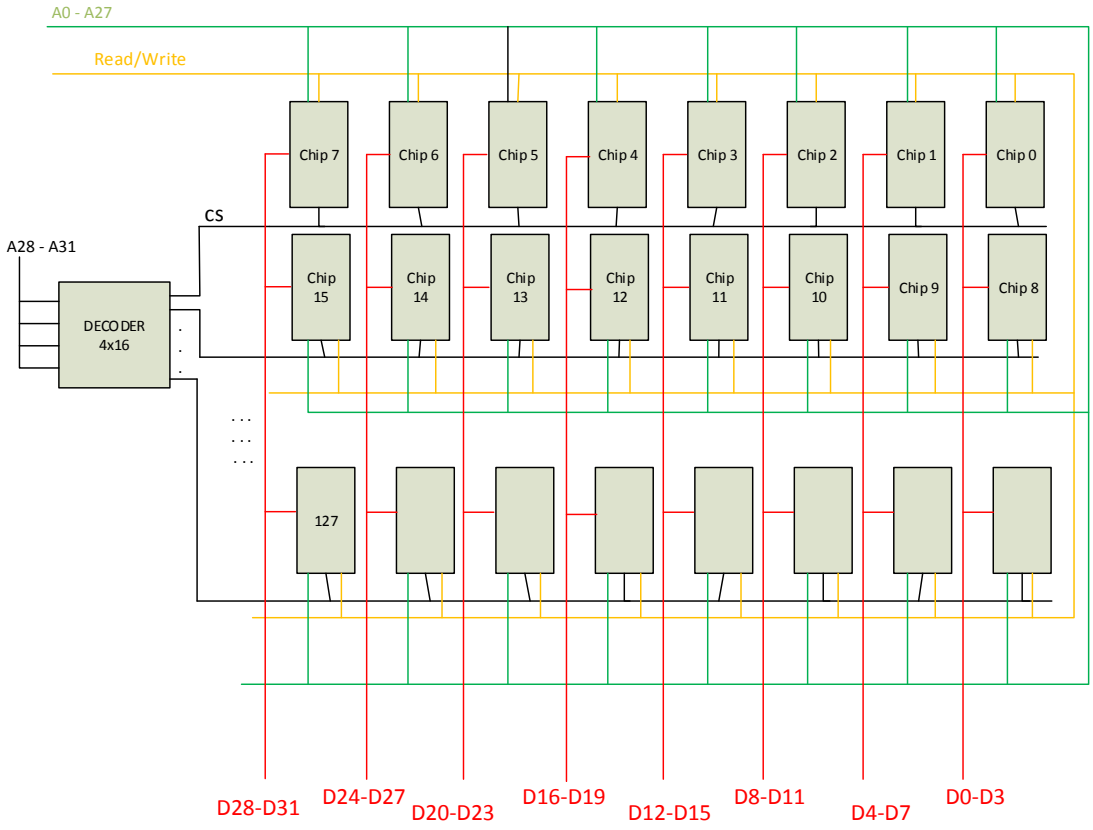
Λύση Α

Στην περίπτωση που η **ελάχιστη διευθυνσιοδοτούμενη ποσότητα μνήμης είναι λέξη των 8 bit (όπως συμβαίνει στον MIPS),** η υλοποίηση είναι η ακόλουθη:



Λύση Β

Στην περίπτωση που η **ελάχιστη διευθυνσιοδοτούμενη ποσότητα μνήμης είναι λέξη των 32 bits,** η αντίστοιχη υλοποίηση είναι η ακόλουθη:



Θέμα 2ο

Δίνεται το ακόλουθο κομμάτι κώδικα σε C:

```
float A[4][4], B[4][3], C[3][4];
for (i=0; i<3; i++)
     for(j=0; j<4; j++)
          B[j][i] = (A[i][j]+A[i+1][j]*C[i][j])/2;
```

Οι πίνακες A, B, C περιέχουν στοιχεία κινητής υποδιαστολής **μονής** ακριβείας (single precision), μεγέθους **4 bytes** το καθένα.

- Όλες οι μεταβλητές εκτός των στοιχείων των πινάκων αποθηκεύονται σε **καταχωρητές** του επεξεργαστή, οπότε οποιαδήποτε αναφορά σε αυτές δε συνεπάγεται προσπέλαση στη μνήμη.
- Σε επίπεδο εντολών assembly οι αναγνώσεις γίνονται με τη σειρά που εμφανίζονται στον κώδικα.
- Οι πίνακες **A**, **B**, **C** είναι αποθηκεμένοι ξεκινώντας από τις θέσεις **0xFF000800**, **0x0F000900** και **0xA0010700** αντίστοιχα.
- Το παραπάνω πρόγραμμα εκτελείται σε ένα επεξεργαστή, τύπου MIPS, με μόνο ένα επίπεδο κρυφής μνήμης δεδομένων η οποία αρχικά είναι άδεια. Η cache διαθέτει **8 blocks**, μεγέθους **8 byte** το καθένα, πολιτική αντικατάστασης **LRU** και πολιτικές εγγραφής **write-allocate, write-through**, ενώ η ελάχιστη ποσότητα δεδομένων που μπορεί να διευθυνσιοδοτηθεί είναι το **1** byte, με χρήση **32 bit** διευθύνσεων.

Υποθέτουμε τις ακόλουθες οργανώσεις μνήμης:

- direct mapped
- 2-way set associative

1. Για κάθε μία από τις παραπάνω οργανώσεις υπολογίστε:
    a. Τα μεγέθη των **TAG**, **INDEX**, **OFFSET** σε bit (δώστε διάγραμμα).
    b. Το συνολικό μέγεθος της μνήμης και το ποσοστό του μεγέθους των tags στο συνολικό μέγεθος
    c. Το συνολικό αριθμό των misses που θα συμβούν καθώς κα το είδος αυτών (**compulsory**, **conflict**, **capacity**)
    d. Πώς μεταβάλλεται ο αριθμός των misses στην περίπτωση αλλαγής της πολιτικής write-allocate σε write-no-allocate;

2. Υποθέτουμε ότι στην περίπτωση του **read/write miss** δαπανούνται **3 επιπλέον κύκλοι** για να έρθει το αντίστοιχο block στην cache, και την κλασική

αρχιτεκτονική των πέντε σταδίων, εφοδιασμένη με τα απαραίτητα σχήματα προώθησης.

a. Δώστε τα διαγράμματα χρονισμού για τις δύο περιπτώσεις.
b. Υπολογίστε τους κύκλους που απαιτούνται για την ολοκλήρωση του προγράμματος στην κάθε περίπτωση.

Λύση:

Direct Mapped

- Block offset: για 8 bytes απαιτούνται **3 bits**
- Index: η cache διαθέτει 8 blocks διατεταγμένα ως 1 block / cache line. Άρα διαθέτει 8 cache lines. Για την διευθυνσιοδότηση αυτών απαιτούνται **3 bits**.
- tagSize = addressSize – indexSize – blockOffest = **26 bits**.

Συνεπώς έχουμε:

TAG	index	offset
26 bits	3 bits	3 bits

Συνολικό μέγεθος tags: **cacheLines * tagSize = 8 * 26 = 208 bits**

Συνολικό μέγεθος μνήμης: **dataSize + (cacheLines * tagSize) = 720 bits**

Ποσοστό μεγέθους tags στο συνολικό μέγεθος cache: **P = 208 / 720 = 28.8 %**

Κατασκευάζουμε τον πίνακα αντιστοίχησης στοιχείων με τις θέσεις που αυτά αποθηκεύονται στην cache. Στον πίνακα αυτό κβαντίζεται και παρουσιάζεται ο κάθε πίνακας σε block τα οποία συνδέονται με τον αντίστοιχο δείκτη που τα διευθυνσιοδοτεί στην cache.

cache index	A elements	B elements	C elements
000	A[0][0] / A[0][1]	B[0][0] / B[0][1]	C[0][0] / C[0][1]
001	A[0][2] / A[0][3]	B[0][2] / B[1][0]	C[0][2] / C[0][3]
010	A[1][0] / A[1][1]	B[1][1] / B[1][2]	C[1][0] / C[1][1]
011	A[1][2] / A[1][3]	B[2][0] / B[2][1]	C[1][2] / C[1][3]
100	A[2][0] / A[2][1]	B[2][2] / B[3][0]	C[2][0[/ C[2][1]
101	A[2][2] / A[2][3]	B[3][1] / B[3][2]	C[2][2] / C[2][3]
110	A[3][0] / A[3][1]		
111	A[3][2] / A[3][3]		

Από τη βηματική εκτέλεση του δοθέντος τμήματος κώδικα προκύπτει:

Με τη μορφή R(element) παρουσιάζεται το ποιο στοιχείο και συνεπώς block, εκτοπίζεται από τη cache

A[i][j]	A[i+1][j]	C[i][j]	B[i][j]
A[0][0]: compM	**A[1][0]: compM**	**C[0][0] : compM R(A[0][0])**	**B[0][0]: compM R(C[0][0])**
A[0][1]: confM R(B[0][0])	**A[1][1]: hit**	**C[0][1]: confM R(A[0][1])**	**B[1][0]: compM**
A[0][2]: compM R (B[1][0])	**A[1][2]: compM**	**C[0][2]: compM R (A[0][2])**	**B[2][0]: compM R (A[1][2])**
A[0][3]: confM R(C[0][2])	**A[1][3]: confM R(B[2][0])**	**C[0][3]: confM R(A[0][3])**	**B[3][0]: compM**
A[1][0]: hit	**A[2][0]: compM R(B[3][0])**	**C[1][0]: compM R(A[1][0])**	B[0][1]: confM R (C[0][1])
A[1][1]: confM R(C[1][0])	**A[2][1]: hit**	**C[1][1]: confM R(A[1][1])**	B[1][1]: compM R(C[1][1])
A[1][2]: hit	**A[2][2]: compM**	**C[1][2]: compM R (A[1][2]/ A[1][3])**	**B[2][1]: confM R (C[1][2])**
A[1][3]: confM R(B[2][1])	**A[2][3]: hit**	C[1][3]: confM R (A[1][3])	**B[3][1]: compM R(A[2][2]/ A[2][3])**
A[2][0]: hit	A[3][0]: compM	**C[2][0]: compM R(A[2][0]/ A[2][1])**	B[0][2]: confM R(C[0][3])
A[2][1]: confM R(C[2][0])	A[3][1]: hit	**C[2][1]: confM R(A[2][1])**	B[1][2]: hit
A[2][2]: confM R(B[3][1])	A[3][2]: compM	**C[2][2]: compM R (A[2][2])**	B[2][2]: confM R(C[2][1])
A[2][3]: confM R(C[2][2])	A[3][3]: hit	**C[2][3]: confM R(A[2][3])**	B[3][2]: confM R(C[2][3])

Συνολικά έχουμε 39 misses εκ των οποίων 20 είναι compulsory και 19 conflict misses.

2-Way Set associative

- Block offset: για 8 bytes απαιτούνται **3 bits**
- Index: η cache διαθέτει 8 blocks διατεταγμένα ως 2 blocks / cache line. Άρα διαθέτει 4 cache lines. Για την διευθυνσιοδότηση αυτών απαιτούνται **2 bits**.
- tagSize = addressSize – indexSize – blockOffest = **26 bits**.

  Συνεπώς έχουμε:

TAG	index	offset
27 bits	2 bits	3 bits

Συνολικό μέγεθος tags: **cacheLines * tagSize = 8 * 27 = 216 bits**

Επισημαίνεται ότι στην περίπτωση της 2-way set associative cache έχουμε LRU πολιτική η οποία επιβαρύνει κατά 1 bit τα metadata του κάθε block.

Συνολικό μέγεθος μνήμης: **dataSize + (cacheLines * tagSize) + #blocks*1 = 736 bits**

Ποσοστό μεγέθους tags στο συνολικό μέγεθος cache: **P = 216 / 736 = 29.3 %**

Κατασκευάζουμε τον πίνακα αντιστοίχησης στοιχείων με τις θέσεις που αυτά αποθηκεύονται στην cache. Στον πίνακα αυτό κβαντίζεται και παρουσιάζεται ο κάθε πίνακας σε block τα οποία συνδέονται με τον αντίστοιχο δείκτη που τα τοποθετεί στην cache.

Cache index	A elements	A elements	B elements	B elements	C elements	C elements
00	A[0][0] / A[0][1]	A[2][0] / A[2][1]	B[0][0] / B[0][1]	B[2][2] / B[3][0]	C[0][0] / C[0][1]	C[2][0] / C[2][1]
01	A[0][2] / A[0][3]	A[2][2] / A[2][3]	B[0][2] / B[1][0]	B[3][1] / B[3][2]	C[0][2] / C[0][3]	C[2][2] / C[2][3]
10	A[1][0] / A[1][1]	A[3][0] / A[3][1]	B[1][1] / B[1][2]		C[1][0] / C[1][1]	
11	A[1][2] / A[1][3]	A[3][2] / A[3][3]	B[2][0] / B[2][1]		C[1][2] / C[1][3]	

Από τη βηματική εκτέλεση του δοθέντος τμήματος κώδικα προκύπτει:

Με τη μορφή R(element) παρουσιάζεται το ποιο στοιχείο και συνεπώς block, εκτοπίζεται από τη cache ενώ σε παρενθέσεις έχει τοποθετηθεί ο αντίστοιχος cache-line index.

A[i][j]	A[i+1][j]	C[i][j]	B[i][j]
A[0][0]: compM (00)	**A[1][0]: compM (10)**	**C[0][0] : compM (00)**	**B[0][0]: compM (00) R(A[0][0])**
A[0][1]: confM (00) R(C[0][0])	**A[1][1]: hit (10)**	**C[0][1]: confM (00) R(B[0][0])**	**B[1][0]: compM (01)**
A[0][2]: compM (01)	**A[1][2]: compM (11)**	**C[0][2]: compM (01) R (B[1][0])**	**B[2][0]: compM (11)**
A[0][3]: hit (01)	**A[1][3]: hit (11)**	**C[0][3]: hit (01)**	**B[3][0]: compM (00) R(A[0][1])**
A[1][0]: hit (10)	**A[2][0]: compM (00) R(C[0][1])**	**C[1][0]: compM (10)**	**B[0][1]: confM (00) R (B[3][0])**
A[1][1]: hit (10)	**A[2][1]: hit (00)**	**C[1][1]: hit (10)**	B[1][1]: compM (10) R(A[1][1]/ A[1][0])

A[1][2]: hit (11)	**A[2][2]: compM (01) R(A[0][3]/ A[0][2])**	**C[1][2]: compM (11) R (B[2][0])**	**B[2][1]: confM (11) R (A[1][2] /A[1][3])**
A[1][3]: confM (11) R(C[1][2])	**A[2][3]: hit (01)**	C[1][3]: confM (11) R (B[2][1])	**B[3][1]: compM (01) R(C[0][2]/ C[0][3])**
A[2][0]: hit (00)	A[3][0]: compM (10) R (C[1][0]/ C[1][1])	C[2][0]: compM (00) R(B[0][1])	**B[0][2]: confM (01) R(A[2][2]/A[2][3])**
A[2][1]: hit (00)	A[3][1]: hit (10)	C[2][1]: hit (00)	B[1][2]: hit (10)
A[2][2]: confM (01) R(B[3][1])	A[3][2]: compM (s11) R(A[1][3])	C[2][2]: compM (01) R (B[0][2])	B[2][2]: confM (s00) R(A[2][0]/ A[2][1])
A[2][3]: hit (01)	A[3][3]: hit (s11)	C[2][3]: hit (01)	B[3][2]: confM (s01) R(A[2][2]/A[2][3])

Από τον παραπάνω πίνακα προκύπτουν 18 hits και 30 misses εκ των οποίων 10 είναι conflict και 20 είναι compulsory.

Στην περίπτωση αλλαγής της πολιτικής **write-allocate σε write-no-allocate** ο συνολικός αριθμός hits θα αυξηθεί λόγω της απουσίας των στοιχείων του πίνακα B από την cache.

Μια πιθανή υλοποίηση της δοθείσας συνάρτησης σε MIPS assembly είναι η ακόλουθη:

```
            addi $3, $zero, 3
            addi $4, $zero, 16
            addi $5, $zero, $0
            addi $6, $zero, $zero
            addi $1, $zero, $zero
LOOPEX:
      addi $2, $zero, $zero
      LOOPIN:
            add $7, $6, $2
            addi $13, $0, 0xff000800
            addu $13, $7, $13
            lw $8, 0($13)
            lw $9, 0($13)
            addi $7, $7, 16
            addi $13, $0, 0xff000800
            addu $13, $7, $13
            lw $10, 0($13)
```

```
            add $10, $10, $8
            add $10, $10, $9
            srl $10, $10, 1
            mul $7, $2, $3
            muli $8, $1, 4
            add $7, $7, $8
            addi $13, $0, 0xA0010700
            addu $13, $8, $13
            sw $10, 0($13)
      addi $2, $2, 4
      blt $2, $4, LOOPIN
addi $1, $1, 1
addi $6, $6, 16
blt $1, $3, LOOPEX
```

Μπορούμε να χωρίσουμε τον παραπάνω κώδικα σε 3 τμήματα:

- τμήμα αρχικοποιήσεων (εκτελείται 1 φορά)
- τμήμα εξωτερικού αλλά όχι εσωτερικού βρόχου (εκτελείται 3 φορές)
- τμήμα εσωτερικού βρόχου (εκτελείται 12 φορές)

addi $3, $zero, 3		IF	ID	EX	MEM	WB																																													
addi $4, $zero, 16			IF	ID	EX	MEM	WB																																												
addi $5, $zero, $0				IF	ID	EX	MEM	WB																																											
addi $6, $zero, $zero					IF	ID	EX	MEM	WB																																										
addi $1, $zero, $zero						IF	ID	EX	MEM	WB																																									
LOOPEX: addi $2, $zero, $zero							IF	ID	EX	MEM	WB																																								
LOOPIN: add $7, $6, $2								IF	ID	EX	MEM	WB																																							
addi $13, $0, 0xff000800									IF	ID	EX	MEM	WB																																						
addu $13, $7, $13										IF	ID	EX	MEM	WB																																					
lw $8, 0($13)	MISS										IF	ID	EX	MEM	-	-	-	WB																																	
lw $9, 0($13)	MISS											IF	ID	EX	-	-	-	MEM	-	-	-	WB																													
addi $7, $7, 16													IF	ID	-	-	-	EX	-	-	-	MEM	WB																												
addi $13, $0, 0xff000800														IF	-	-	-	ID	-	-	-	EX	MEM	WB																											
addu $13, $7, $13																		IF	-	-	-	ID	EX	MEM	WB																										
lw $10, 0x($13)	MISS																					IF	ID	EX	MEM	-	-	-	WB																						
add $10, $10, $8																							IF	ID	EX	-	-	-	MEM	WB																					
add $10, $10, $9																								IF	ID	-	-	-	EX	MEM	WB																				
srl $10, $10, 1																									IF	-	-	-	ID	EX	MEM	WB																			
mul $7, $2, $3																													IF	ID	EX	MEM	WB																		
muli $8, $1, 4																														IF	ID	EX	MEM	WB																	
add $7, $7, $8																															IF	ID	EX	MEM	WB																
addi $13, $0, 0xA0010700																																IF	ID	EX	MEM	WB															
addu $13, $8, $13																																	IF	ID	EX	MEM	WB														
sw $10, 0($13)	MISS																																	IF	ID	EX	MEM	-	-	-	WB										
addi $2, $2, 4																																			IF	ID	EX	-	-	-	MEM	WB									
blt $2, $4, LOOPIN																																				IF	ID	-	-	-	EX	MEM	WB								
addi $1, $1, 1																																										IF	ID	EX	MEM	WB					
addi $6, $6, 16																																											IF	ID	EX	MEM	WB				
blt $1, $3, LOOPEX																																												IF	ID	EX	MEM	WB			
addi $2, $zero, $zero																																															IF	ID	EX	MEM	WB

Στο παραπάνω σχήμα παρουσιάζεται τα ενδεικτικά διαγράμματα χρονισμού των τριών περιπτώσεων.

Από το διάγραμμα φαίνεται ότι το σχήμα αρχικοποιήσεων απαιτεί 9 κύκλους.

Οι εντολές του εξωτερικού βρόχου απαιτούν 8 κύκλους. Άρα τα δύο αυτά τμήματα αθροιστικά

απαιτούν 9 + 3*8 = 33 κύκλους.

Ο εσωτερικότερος βρόχος απαιτεί αριθμό κύκλων ανάλογο των misses όπως αυτά απεικονίζονται

στους προηγούμενους πίνακες. Είναι φανερό ότι κάθε miss εισάγει καθυστέρηση 3 κύκλων.