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 2009
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 μονάδες)

Α (10%). Δίνεται επεξεργαστής με βαθιά σωλήνωση, ο οποίος χρησιμοποιεί Branch-Target Buffer (BTB) για την πρόβλεψη διακλαδώσεων. Ισχύουν τα εξής:

- 30% των εντολών είναι εντολές διακλάδωσης.
- Ο BTB έχει hit rate 80% ενώ η ακρίβεια πρόβλεψης του είναι 95%.
- Η ποινή σε περίπτωση λανθασμένης πρόβλεψης του BTB είναι 6 κύκλοι
- Σε περίπτωση αστοχίας του BTB, ο επεξεργαστής χρησιμοποιεί έναν two-level predictor (1,1). Η ποινή σε περίπτωση λανθασμένης πρόβλεψης του two-level predictor είναι 4 κύκλοι.
- Για κάθε εντολή διακλάδωσης η αποκωδικοποίηση και η πρόβλεψη γίνονται στο πρώτο στάδιο της σωλήνωσης. Ο έλεγχος ορθότητας της πρόβλεψης γίνεται στο τελευταίο στάδιο και σε περίπτωση λάθους η εκτέλεση της σωστής εντολής ξεκινά από τον επόμενο κύκλο.
- Οι εντολές που δεν προκαλούν stalls απαιτούν 1 κύκλο για να εκτελεστούν.

Υπολογίστε το ελάχιστο ποσοστό ακρίβειας πρόβλεψης του two-level predictor, ώστε ο επεξεργαστής αυτός να είναι πιο γρήγορος από έναν άλλο επεξεργαστή όπου δεν υπάρχει πρόβλεψη διακλάδωσης και κάθε εντολή διακλάδωσης κάνει stall την σωλήνωση για 2 κύκλους.

Για να συγκρίνουμε τους 2 επεξεργαστές θα πρέπει να υπολογίσουμε τον μέσο αριθμό κύκλων που απαιτεί η εκτέλεση κάθε εντολής.

$CPI_A = 1 + 0.3*(0.8*0.05*6 + 0.2*(1-h)*4) = 1.312 – 0.24h$
$CPI_B = 1 + 0.3 * 2 = 1.6$

Για να είναι πιο γρήγορος ο πρώτος υπολογιστής θα πρέπει να ισχύει :

$1.312 – 0.24h \leq 1.6 \rightarrow h \geq -1.2$ *που ισχύει ούτως ή άλλως αφού* $h \geq 0$. *Δηλαδή, ο πρώτος επεξεργαστής με αυτά τα χαρακτηριστικά είναι πάντα πιο γρήγορος από τον* 2^o, *ανεξάρτητα από το ποσοστό επιτυχίας του two-level predictor. Το πόσο πιο γρήγορος θα είναι προφανώς εξαρτάται από το h.*

Β (5%). Δίνονται οι εξής 3 τεχνικές : α) Out-of-order issue with renaming, β) Branch prediction, γ) Superscalar

Για τα παρακάτω κομμάτια κώδικα, εξηγήστε ποια τεχνική θα επιλέγατε να εντάξετε στο σύστημά σας. Εξηγήστε επίσης τους λόγους για τους οποίους θα απορρίπτατε τις υπόλοιπες τεχνικές.

(i)
```
ADD F0, F1, F8
ADD F2, F3, F8
ADD F4, F5, F8
ADD F6, F7, F8
```

(ii)
```
loop: ADD  R3, R4, R0
      LD   R4, 8(R4)
      BNEQZ R4, loop
```

(iii)
```
LD R1, 0(R2)      #cache miss
ADD  R2, R1, R1
LD  R1, 0(R3)     #cache hit
LD  R3, 0(R4)     #cache hit
ADD  R3, R1, R3
ADD  R1, R2, R3
```

(i) Θα επιλέγαμε Superscalar μιας και οι εντολές δεν έχουν dependencies μεταξύ τους και μπορούν να εκτελεστούν παράλληλα εφόσον η αρχιτεκτονική διαθέτει πολλαπλά functional units. To branch prediction δε θα προσέφερε κάτι μιας και δεν υπάρχει εντολή άλματος. Αντίστοιχα, και η τεχνική του out-of-order δε θα προσέφερε κάτι μιας και δεν υπάρχει κάποιο dependency, η επίλυση του οποίου να καθυστερεί την εκτέλεση κάποιας εντολής.
(ii) Προφανώς θα επιλέγαμε να υλοποιήσουμε το branch prediction καθώς ο συγκεκριμένος κώδικας είναι ένα loop και η εντολή διακλάδωσης θα εκτελεστεί αρκετές φορές. Η superscalar αρχιτεκτονική δε θα βοηθούσε μιας και υπάρχουν dependencies μεταξύ των εντολών και άρα δεν μπορούν να εκτελεστούν παράλληλα. Η out-of-order τεχνική θα επίλυε τα WAR αλλά τα RAW θα παρέμεναν. Ταυτόχρονα, καθώς δεν υπάρχει πρόβλεψη εντολή άλματος, η απόδοση θα περιοριζόταν καθώς θα έπρεπε να περιμένει να μάθει το αποτέλεσμα της διακλάδωσης. Έτσι δε θα βελτίωνε την απόδοση και ουσιαστικά οι εντολές θα κατέληγαν να εκτελεστούν in-order.
(iii) Σε αυτή την περίπτωση θα επιλέγαμε out-of-order, καθώς θα μας επέτρεπε να εκτελέσουμε την $3^{η}$, $4^{η}$ και $5^{η}$ εντολή καθώς περιμένουμε να ικανοποιηθεί η αστοχία μνήμης της $1^{ης}$ εντολής. Το branch prediction προφανώς δεν βοηθά καθώς δεν υπάρχουν branches, ενώ η superscalar αρχιτεκτονική δεν μπορεί να εκτελέσει παράλληλα τις εντολές λόγω των dependencies που υπάρχουν και οδηγούν σε hazards.

Γ (5%). Δίνεται επεξεργαστής με ένα επίπεδο κρυφής μνήμης. Πώς θα επηρεαστούν τα Compulsory, Conflict και Capacity misses, αν γίνει ξεχωριστά το καθένα από τα παρακάτω; Θα αυξηθούν, θα μειωθούν ή θα μείνουν αμετάβλητα; Εξηγήστε γιατί.
(i) Διπλασιασμός του associativity, διατηρώντας σταθερό το μέγεθος του block και της cache.

Η συνέπεια της κίνησης αυτής είναι η μείωση στο μισό του αριθμού των sets. Τα compulsory misses παραμένουν σταθερά καθώς η αύξηση του associativity δεν επηρεάζει το πότε θα έρθουν για πρώτη φορά δεδομένα μέσα στην cache. Σταθερά παραμένουν και τα capacity misses, μιας και το capacity της cache παραμένει σταθερό. Τα conflict misses περιμένουμε όμως να μειωθούν, καθώς το μεγαλύτερο associativity σημαίνει ότι υπάρχει περισσότερος χώρος για να αποθηκευθούν στοιχεία που είναι mapped στο ίδιο set.

(ii) Υποδιπλασιασμός του μεγέθους του block, διατηρώντας σταθερό το associativity και τον αριθμό των sets.

Η συνέπεια της κίνησης αυτής είναι η μείωση στο μισό της χωρητικότητας της cache. Καθώς ο αριθμός των sets και ways παραμένει σταθερός τα conflict misses δεν επηρεάζονται. Η μείωση στο μισό όμως της χωρητικότητας προφανώς θα επιφέρει αύξηση των capacity misses. Επίσης, θα υπάρξει αύξηση των compulsory misses, καθώς μικρότερο block σημαίνει μικρότερη αξιοποίηση της spatial locality και ‘‘λιγότερο’’ prefetching μιας και τώρα θα έρχονται λιγότερα στοιχεία στην cache για κάθε miss.

(iii) Διπλασιασμός του αριθμού των sets διατηρώντας σταθερό το μέγεθος του block καθώς και το συνολικό μέγεθος της cache.

Η κίνηση αυτή συνεπάγεται μείωση στο μισό του associativity. Τα compulsory misses παραμένουν σταθερά καθώς δεν επηρέαζεται το πότε θα φορτωθούν δεδομένα στην cache. Αντίστοιχα σταθερά παραμένουν και τα capacity misses, μιας και η συνολική χωρητικότητα παραμένει σταθερή. Η μείωση όμως του associativity θα επιφέρει αύξηση των conflict misses.

(iv) Εφαρμογή προφόρτωσης δεδομένων (prefetching).

Το prefetching μπορεί να μειώσει τα compulsory misses καθώς αν λειτουργεί σωστά θα φορτώνει δεδομένα στην cache πριν αυτά ζητηθούν για πρώτη φορά. Τα υπόλοιπα misses παραμένουν σταθερά. Υπάρχει βέβαια περίπτωση σε κάποιο μη αποδοτικό prefetching, να έχουμε φαινόμενα cache pollution/thrashing, στην οποία περίπτωση θα αυξηθούν και τα conflict και τα capacity misses.

(v) Εφαρμογή τεχνικής blocking.

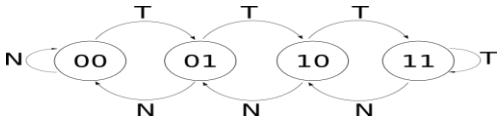
Το blocking μειώνει τα capacity misses, ενώ αφήνει ανεπηρέαστα τα conflict misses καθώς δεν μπορεί να κάνει κάτι για τα στοιχεία εκείνα που είναι mapped στο ίδιο set. Αντίστοιχα, δεν επηρεάζει το πότε θα φορτωθούν για πρώτη φορά στοιχεία στην cache και άρα τα compulsory misses παραμένουν σταθερά.

Θέμα 2^{ο} (15 μονάδες)

Υποθέστε εντολή διακλάδωσης με την παρακάτω συμπεριφορά :

T, T, T, *N, N*, T, T, T, *N, N*, T, T, T, *N, N*

όπου T taken και *N* not-taken.
(i) Ποια τα ποσοστά επιτυχίας ενός static Taken predictor, ενός 1-bit predictor με αρχική τιμή 1 (κατάσταση T) και ενός 2-bit predictor, με αρχική τιμή 3 (κατάσταση T). Ποιο predictor θα επιλέγατε να υλοποιήσετε; Ο 2-bit predictor χρησιμοποιεί το παρακάτω FSM (finite state machine):



Ο static Taken κάνει 6/15 λάθη. Ο 1-bit κάνει 5/15 λάθη. Ο 2-bit κάνει 8/15 λάθη. Θα επιλέγαμε λοιπόν τον 1-bit.

(ii) Έστω ένας (m,1) local-history predictor. Ποιο είναι το ελάχιστο m που απαιτείται προκειμένου ο predictor να έχει ιδανική συμπεριφορά; Για το m αυτό, δείξτε τα περιεχόμενα του PHT όπως στον παρακάτω πίνακα.

Παρατηρούμε ότι όταν έχουμε TTT το αποτέλεσμα είναι πάντα N, για TTN είναι N, για TNN είναι T, για NNT είναι T και για NTT είναι T. Οι περιπτώσεις αυτές καλύπτουν πλήρως τη συμπεριφορά που έχει δοθεί. Ο ελάχιστος λοιπόν αριθμός history bits που απαιτείται είναι 3.

PHT Index	***Prediction***
000	*X*
001	*T*
010	*X*
011	*T*
100	*T*
101	*X*
110	*N*
111	*N*

Θέμα 3ο (30 μονάδες)

Α (10%). Έστω ένας επεξεργαστής που υλοποιεί τον αλγόριθμο Tomasulo με out-of-order commit εντολών. Δίνονται οι παρακάτω εντολές :

```
1.    ADD   R1, R2, R1
2.    MUL   R3, R4, R6
3.    SUB   R5, R4, R1
4.    SD    R5, 0(R3)
```

Υποθέστε επίσης τα εξής :

1. Η αρχιτεκτονική μπορεί να κάνει fetch και issue 2 instructions ανά κύκλο, ενώ διαθέτει 1 CDB.
2. Η αρχιτεκτονική διαθέτει 5 reservation stations, τα Store1, Add1, Add2, Mult1 και Mult2.
3. Οι εντολές SD, ADD και SUB απαιτούν 3 κύκλους εκτέλεσης ενώ η MULT 8 κύκλους.

(i) Δώστε την εικόνα των reservation stations, καθώς και του register result status, όταν cycles = 3, φτιάχνοντας έναν πίνακα όπως ο παρακάτω:

Name	***Busy***	***OP***	V_i	V_k	Q_i	Q_k
Add1	*Y*	*ADD*	*R[R2]*	*R[R1]*		
Add2	*Y*	*SUB*	*R[R4]*			*Add1*
Mult1	*Y*	*MUL*	*R[R4]*	*R[R6]*		
Mult2	*N*					

Name	***Busy***	***Address***	***Value***
Store 1	*Y*	*M[0 + Mult1]*	*Add2*

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

OP	**Issue**	**Exec**	**WB**	**Σχόλιο**
ADD R1, R2, R1	1	2 - 4	5	
MUL R3, R4, R6	1	2 - 9	10	
SUB R5, R4, R1	2	6 – 8	9	RAW στον R1. Περιμένει την ADD.
SD R5, 0(R3)	2	11 - 13	14	RAW στον R5. Περιμένει την SUB.

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

Αυτό που θα άλλαζε είναι πως θα είχαμε in-order commit των εντολών. Έτσι, ο παραπάνω πίνακας θα παρέμενε ίδιος για το IS, EX, WR ενώ το commit θα γινόταν στους κύκλους 6, 11, 12, 15 με βάση τη σειρά των εντολών.

Β (20%). 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 :

1. Τα IS, WR, CMT απαιτούν 1 κύκλο.
2. Το σύστημα περιέχει *άπειρα* reservation stations.

3. Το σύστημα περιλαμβάνει 2 *non-pipelined* integer functional units. Σε αυτά εκτελούνται οι αριθμητικές εντολές για ακεραίους, καθώς και οι εντολές διακλάδωσης. Οι εντολές αυτές διαρκούν 2 κύκλους.
4. Το σύστημα περιλαμβάνει 2 floating point functional units, ένα για ADDD, SUBD και ένα για MULD, DIVD. Τα functional units αυτά είναι *πλήρως pipelined* στη συχνότητα του επεξεργαστή. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 4 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 6 κύκλους.
5. Για τις εντολές αναφοράς στη μνήμη, στο στάδιο EX, γίνεται τόσο ο υπολογισμός της διεύθυνσης αναφοράς όσο και η προσπέλαση στη μνήμη. Σε περίπτωση που η προσπέλαση στη μνήμη καταλήξει σε *HIT η εντολή διαρκεί 1 κύκλο*. Αντίθετα, αν υπάρξει *MISS η εντολή διαρκεί 6 κύκλους*.
6. Ο ROB έχει 8 θέσεις.
7. Το σύστημα περιλαμβάνει 1 CDB. Σε περίπτωση που παραπάνω από μια εντολή θέλουν να το χρησιμοποιήσουν, τότε *προτεραιότητα αποκτά η ''παλαιότερη'' εντολή*.
8. Για τις εντολές διακλάδωσης χρησιμοποιείται ένας 1-bit predictor, ο οποίος είναι αρχικοποιημένος στο 0. Υπενθυμίζεται η σύμβαση 0→NT, 1→T.
9. Η πρόβλεψη για μια εντολή διακλάδωσης γίνεται *ταυτόχρονα* με τη δρομολόγηση της εντολής.
10. Ο έλεγχος της πρόβλεψης για μια εντολή διακλάδωσης γίνεται αμέσως μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR. Σε περίπτωση λανθασμένης πρόβλεψης, σταματά η εκτέλεση των λάθος εντολών και στον επόμενο κύκλο δρομολογείται η σωστή εντολή.

Δίνεται ο παρακάτω κώδικας :

```
1.    LOOP:     LD    F0, 0(R1)
2.              MULD  F1, F0, F4
3.              ADD   F0, F0, F1
4.              SD    F0, 0(R1)
5.              ADD   R1, R1, #8
6.              SUB   R2, R2, #1
7.              BNEZ  R2, LOOP
8.              MULD  F5, F6, F7
9.              ADD   R5, R6, R7
```

Ο καταχωρητής R1 περιέχει τη διεύθυνση του πρώτου στοιχείου ενός πίνακα A, στον οποίο είναι αποθηκευμένοι αριθμοί διπλής ακρίβειας (μεγέθους 8 bytes ο καθένας). Ο πίνακας είναι *ευθυγραμμισμένος*, δηλαδή το πρώτο στοιχείο του βρίσκεται στην αρχή του block. Το μέγεθος του κάθε cache block είναι 16 bytes. Στην αρχή της εκτέλεσης, η cache είναι άδεια.

Αν η αρχική τιμή του καταχωρητή R2 είναι 2, εκτελέστε τον παραπάνω κώδικα και δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών σε έναν πίνακα όπως ο παρακάτω :

No.	**OP**	**IS**	**EX**	**WR**	**CMT**	**Σχόλιο**
1	L.D F0, 0(R1)	1	2 - 7	8	9	Το πρώτο load είναι miss. Άρα χρειάζεται 6 κύκλους.
2	MULD F1, F0, F4	2	9 - 14	15	16	RAW με την εντολή 1.
3	ADD F0, F0, F1	3	16 - 19	20	21	Καθυστέρηση λόγω RAWs με τις 2 προηγούμενες εντολές.
4	SD F0, 0(R1)	4	21 - 21	22	23	Το στοιχείο βρίσκεται στην cache και άρα έχουμε hit.
5	ADD R1, R1, #8	5	6 - 7	9	24	
6	SUB R2, R2, #1	6	7 - 8	10	25	
7	BNEZ R2, LOOP	7	11 - 12	13	26	RAW με την SUB. Η πρόβλεψη είναι NT. Ο predictor πάει στο 1. Στον κύκλο 13 μαθαίνουμε ότι είναι λάθος και κάνουμε reset τις επόμενες εντολές (8,9).
8	MULD F5, F6, F7	8	10 – 13			Μπορεί να ξεκινήσει να εκτελείται στον κύκλο 10 μιας και το fp unit είναι pipelined.
9	ADD R5, R6, R7	10	11 - 12			Issue στο 10 λόγω γεμάτου ROB. Δεν κάνει WR γιατί γίνεται flush.
10	LD F0, 0(R1)	14	15 - 15	16	27	Το στοιχείο που θέλουμε ανήκει στο ίδιο block που ανήκει και το στοιχείο του 1^{ου} Load (εντολή 1). Άρα έχουμε hit.
11	MULD F1, F0, F4	15	17 - 22	23	28	RAW με την εντολή 10. Σε αυτό το σημείο γεμίζει ξανά ο ROB.
12	ADD F0, F0, F1	17	24 - 27	28	29	Καθυστέρηση στο IS μέχρι να αδειάσει μια θέση στο ROB. RAW με τις 2 προηγούμενες εντολές. Ξανά γεμάτος ο ROB.
13	SD F0, 0(R1)	22	29 - 29	30	31	Καθυστέρηση στο IS μέχρι να αδειάσει μια θέση στο ROB. RAW με την εντολή 12. Ξανά γεμάτος ο ROB. Θεωρούμε ότι δεν υπάρχει πρόβλημα με την εντολή 17 η οποία επίσης προσπαθεί να προσπελάσει τη μνήμη.
14	ADD R1, R1, #8	24	25 - 26	27	32	Καθυστέρηση στο IS μέχρι να αδειάσει το ROB.
15	SUB R2, R2, #1	25	26 - 27	29	33	Καθυστέρηση στο WR, λόγω του ότι το CDB χρησιμοποιείται από την εντολή 12.
16	BNEZ R2, LOOP	26	30 – 31	32	34	Η πρόβλεψη θα είναι T. Στον κύκλο 32 μαθαίνουμε ότι η πρόβλεψη ήταν λάθος και κάνουμε flush τις εντολές 17-22
17	LD F0, 0(R1)	27	28 – 32			Η εντολή προσπαθεί να φορτώσει το 3^{ο} στοιχείο του πίνακα, το οποίο είναι σε άλλο block. Άρα miss.
18	MULD F1, F0, F4	28				
19	ADD F0, F0, F1	29				
20	SD F0, 0(R1)	30				Γεμίζει ο ROB.
21	ADD R1, R1, #8	32				Καθυστέρηση στο IS λόγω γεμάτου ROB. Στον κύκλο αυτό κάνουμε flush.
22	MULD F5, F6, F7	33	34 - 39	40	41	
23	ADD R5, R6, R7	34	35 - 36	37	42	

Θέμα 4ο (25 μονάδες)

Εξετάζουμε την εκτέλεση του ακόλουθου κώδικα:

```
#define N 16
double v[N], A[N][N], u[N] ;
int i,j;
for(j=0; j<N; j++)
      for(i=0; i<N; i++)
            v[j] += A[i][j] * u[i];
```

Όλοι 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, μεγέθους 192 bytes. Η κρυφή μνήμη είναι πλήρως συσχετιστική (fully associative) με LRU πολιτική αντικατάστασης. Το μέγεθος του block της κρυφής μνήμης είναι 32 bytes.
2. Σε επίπεδο εντολών assembly, η σειρά με την οποία γίνονται οι αναφορές στους πίνακες είναι η εξής: A, u, v, v.
4. Αρχικά, η κρυφή μνήμη δεδομένων είναι άδεια.

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

(ii) Εφαρμόστε την τεχνική του blocking στον κώδικα για να μειώσετε τον αριθμό των misses. Ποιο block size θα επιλέγατε ως καταλληλότερο και γιατί; Πόσο γίνεται τώρα το ποσοστό αστοχίας;

(iii) Το σύνολο εντολών της αρχιτεκτονικής του επεξεργαστή διαθέτει μία ειδική εντολή *prefetch*(*addr). Η εντολή αυτή προ-φορτώνει στην κρυφή μνήμη ολόκληρο το μπλοκ που περιέχει τη λέξη που βρίσκεται στη διεύθυνση μνήμης addr. Στον κώδικα που προέκυψε από το ερώτημα ii, καλείστε να εφαρμόσετε την τεχνική του prefetching, με τις εξής υποθέσεις/περιορισμούς:

- Μπορείτε να εισάγετε 1 μόνο κλήση στην prefetch οπουδήποτε μέσα στον κώδικα.
- Για να καλυφθεί ο χρόνος που απαιτείται για να έρθουν τα δεδομένα που ζητά η prefetch στην cache, αρκούν 2 εκτελέσεις του βασικού σώματος του βρόχου (συμπεριλαμβανομένων των όποιων κλήσεων στην εντολή).
- Θα πρέπει να προσέχετε ώστε τα δεδομένα που προφορτώνονται να μην εκτοπίζουν άλλα χρήσιμα δεδομένα. Για το σκοπό αυτό μπορείτε να χρησιμοποιήσετε εντολές ελέγχου για την υπό συνθήκη εκτέλεση της prefetch.

Πόσο γίνεται τώρα το ποσοστό αστοχίας;