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 Σεπτεμβρίου 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. Απαντήστε όσο το δυνατόν πιο συνοπτικά στα παρακάτω:
i) Για ποιο λόγο θα θέλαμε οι επεξεργαστές να έχουν τεράστια register files; Για ποιο λόγο δεν έχουν;
ii) Πως μπορεί ο ROB να επηρεάσει αρνητικά την απόδοση του επεξεργαστή;
iii) Μπορείτε να μειώσετε τη πολυπλοκότητα σε έναν 1-instruction-wide, out-of-order επεξεργαστή όπου κάθε εντολή εκτελείται σε 1 κύκλο, χωρίς να αλλάξει η απόδοσή του;

Β. Σχεδιάζετε ένα κύκλωμα το οποίο κάθε φορά που εντοπίζεται ένα mispredicted branch ακυρώνει τα memory requests που δημιουργήθηκαν από εντολές load που βρίσκονταν στο mispredicted path. Αξιολογώντας το, διαπιστώνετε πως για το πρόγραμμα Α η απόδοση βελτιώνεται, ενώ για το πρόγραμμα Β παρατηρείτε μείωση της απόδοσης του επεξεργαστή. Υποθέτοντας πως δεν υπάρχει κάποιο λάθος στο σχεδιασμό του κυκλώματος καθώς και στην μεθοδολογία και εκτέλεση της πειραματικής αξιολόγησης, μπορείτε να εξηγήσετε που οφείλονται τα παραπάνω αποτελέσματα;

Γ. Έστω ένας 2-bit predictor με 1024 εγγραφές και ένας global history predictor (5, 1).
i) Σχεδιάστε τον global history predictor, ώστε οι 2 predictors να έχουν το ίδιο hardware overhead. *[Αγνοήστε τα bits του history register]*
ii) Δώστε μια ακολουθία αποτελεσμάτων ενός άλματος, η οποία όταν επαναλαμβάνεται μπορεί να προβλεφθεί 100% σωστά από τον global history predictor αλλά όχι από τον 2-bit predictor. Εξηγήστε γιατί αποτυγχάνει ο 2-bit predictor. Η ακολουθία δε χρειάζεται να υπερβαίνει τα 10-15 στοιχεία.

Δ. Δίνεται επεξεργαστής με IPC 0.8. Σας προτείνουν να αλλάξετε τον τρόπο εκτέλεσης των εντολών άλματος μειώνοντας κατά 2 κύκλους το branch mispredict penalty. Η αλλαγή όμως αυτή θα αυξήσει τον κύκλο του ρολογιού κατά 10%. Αν το 20% των εντολών είναι εντολές άλματος, ποια θα πρέπει να είναι η ακρίβεια του predictor ώστε να βελτιωθεί η απόδοση του επεξεργαστή;

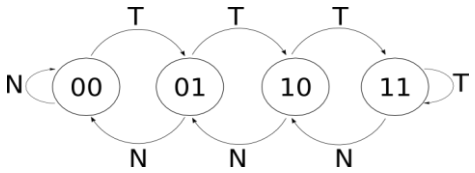
Θέμα 2^{ο} (30%)

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.
- Για integer αριθμούς το σύστημα περιέχει 4 RS για εντολές διακλάδωσης, αριθμητικές και λογικές εντολές καθώς και 4 non-pipelined functional units. Οι integer εντολές διαρκούν 1 κύκλο.
- Για floating point αριθμούς, το σύστημα περιέχει 1 RS για προσθέσεις/αφαιρέσεις, 1 RS για πολλαπλασιασμούς/διαιρέσεις και αντίστοιχο αριθμό non-pipelined functional units. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 1 κύκλο, 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 7 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 και τα stores δεν δημιουργούν conflicts στο CDB.
- 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.

00
11
10
01



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 R_1, R_2 δείχνουν και οι δύο στο πρώτο στοιχείο ενός πίνακα A, ο οποίος περιέχει στοιχεία μήκους 8 bytes και είναι ευθυγραμμισμένος.

Δίνεται ο παρακάτω κώδικας :

```
0x00880004  LOOP: LD    F0, 0(R1)
0x00880008        ADDI  R3, R1, 0x10
0x0088000C        LD    F1, 0(R3)
0x00880010        ADDD  F0, F0, F1          /* Ο αντίστοιχος κώδικας σε C */
0x00880014        DIVD  F0, F0, 0x2         k = 0;
0x00880018        SD    F0, 0(R2)           for(int i=0; i < 6; i+=3) {
0x0088001C        ADDI  R2, R2, 0x8               A[k] = (A[i] + A[i+2])/2;
0x00880020        ADDI  R1, R1, 0x18              k ++ ;
0x00880024        SUBI  R8, R8, 0x1         }
0x00880028        BNEZ  R8, LOOP            A[k] = A[k+2] - A[k+1];
0x0088002C        ADDI  R3, R2, 0x10
0x00880030        LD    F0, 0(R3)
0x00880034        ADDI  R4, R2, 0x8
0x00880038        LD    F1, 0(R4)
0x0088003C        SUBD  F0, F0, F1
0x00880040        SD    F0, 0(R2)
```

Δίνεται επίσης ότι $R_8 = 2$. Εκτελέστε τον 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 τις καθυστερήσεις μεταξύ σταδίων καθώς και ακυρώσεις εντολών.

Θέμα 3ο (20%)

A. Αναλαμβάνετε τη σχεδίαση ενός directory-based cache coherent συστήματος χρησιμοποιώντας το MESI invalidation πρωτόκολλο. Κατά την αξιολόγηση της σχεδίασης διαπιστώνετε ότι για συγκεκριμένες εφαρμογές η απόδοση του συστήματος είναι πολύ μικρή. Μετά από λεπτομερείς προσομοιώσεις, διαπιστώνετε ότι υπάρχει μια σταθερή ροή ακυρώσεων (invalidation requests) μεταξύ τεσσάρων από τους κόμβους του συστήματος για ένα συγκεκριμένο cache block.

i) Εξηγήστε που οφείλεται αυτή η συμπεριφορά.

ii) Εξηγήστε με ποιο τρόπο θα λύσετε το πρόβλημα.

B. Θεωρήστε ένα πολυεπεξεργαστικό σύστημα κοινής μνήμης 3 επεξεργαστών, καθώς και τον ακόλουθο κώδικα για 3 διεργασίες που εκτελούνται παράλληλα στο σύστημα. Οι μεταβλητές A και flag είναι αποθηκευμένες στην κοινή μνήμη, με αρχική τιμή ίση με 0.

P1	**P2**	**P3**
1a: A = 1 1b: print flag	2a: while (flag == 0) ; 2b: print A	3a: flag = 1 3b: print A

i) Θεωρώντας ότι κάθε γραμμή στον παραπάνω κώδικα εκτελείται ατομικά, βρείτε ποιοι συνδυασμοί εκτυπώσεων στους 3 επεξεργαστές είναι ακολουθιακά συνεπείς (sequentially consistent) και ποιοι όχι. Εξηγήστε τις απαντήσεις σας δείχνοντας πώς μπορούν να παραχθούν ή πώς παραβιάζουν το SC.

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

Θέμα 4ο (20%)

Θεωρήστε ένα σύστημα παράλληλης επεξεργασίας αποτελούμενο από 4 επεξεργαστές που χρησιμοποιεί το MESI cache coherence protocol. Κάθε επεξεργαστής διαθέτει μια μόνο L1 data cache μεγέθους 256KB, με μέγθος block 32 bytes. Αρχικά, όλα τα blocks είναι σε κατάσταση I. Στους επεξεργαστές εκτελείται παράλληλα ένα C πρόγραμμα το οποίο περιλαμβάνει τα εξής δεδομένα:

```
struct protected_data {
  int lock;       /* size of int = 4 bytes */
  char data[24]; /* size of char = 1 byte */
};
struct protected_data pd[N];
```

Δίνεται ότι ο πίνακας pd είναι ευθυγραμμισμένος. Ο κώδικας που εκτελείται είναι ο ακόλουθος:

P1, P4	P2,P3

`lock(pd[0].lock);`
`read(pd[0].data);`
`/* computations with pd[0].data */`
`write(pd[0].data);`
`unlock(pd[0].lock);`
	`lock(pd[1].lock);`
	`read(pd[1].data);`
	`/* computations with pd[1].data */`
	`write(pd[1].data);`
	`unlock(pd[1].lock);`

i) Έστω ότι η lock() υλοποιείται με **Test-And-Set (TAS)**. Ο πίνακας που ακολουθεί περιέχει τις προσβάσεις στη μνήμη ως αποτέλεσμα της εκτέλεσης του κώδικα. Δείξτε την κατάσταση των αντίστοιχων cache lines στις caches των επεξεργαστών, καθώς και το αν κάθε πρόσβαση περιλαμβάνει μεταφορά δεδομένων από/προς την κύρια μνήμη ή μεταξύ των caches.

CPU	Λειτουργία	L1 Data Caches επεξεργαστών				Μεταφορά Δεδομένων	
		P1	P2	P3	P4	Από/Προς κύρια μνήμη	Μεταξύ caches
1	TAS(pd[0].lock)	M	I	I	I	√	
3	TAS(pd[1].lock)	?	?	?	?	?	?
4	TAS(pd[0].lock)	?	?	?	?	?	?
2	TAS(pd[1].lock)	?	?	?	?	?	?
1	READ(pd[0].data)	?	?	?	?	?	?
4	TAS(pd[0].lock)						
2	TAS(pd[1].lock)						
3	READ(pd[1].data)						
1	WRITE(pd[0].data)						
4	TAS(pd[0].lock)						
2	TAS(pd[1].lock)						
3	WRITE(pd[1].data)						
3	UNLOCK(pd[1].lock)						
1	UNLOCK(pd[0].lock)						
2	TAS(pd[1].lock)						
4	TAS(pd[0].lock)						

ii) Έστω ότι η lock() υλοποιείται με **Test-And-Test-And-Set (**η TEST στον πίνακα αντιστοιχεί στην πρώτη ανάγνωση του Test-And-Test-And-Set). Συμπληρώστε τον πίνακα. Σχολιάστε την επίδοση σε σχέση με την πρώτη. Θα μπορούσε να βελτιωθεί παραπάνω;

CPU	Λειτουργία	L1 Data Caches επεξεργαστών				Μεταφορά Δεδομένων	
		P1	P2	P3	P4	Από/Προς κύρια μνήμη	Μεταξύ caches
1	TEST(pd[0].lock)	E	I	I	I	√	
4	TEST(pd[0].lock)	?	?	?	?	?	?
3	TEST(pd[1].lock)	?	?	?	?	?	?
1	TAS(pd[0].lock)	?	?	?	?	?	?
3	TAS(pd[1].lock)	?	?	?	?	?	?
4	TAS(pd[0].lock)						
2	TEST(pd[1].lock)						
1	READ(pd[0].data)						
4	TEST(pd[0].lock)						
2	TEST(pd[1].lock)						
3	READ(pd[1].data)						
1	WRITE(pd[0].data)						
4	TEST(pd[0].lock)						
2	TEST(pd[1].lock)						
3	WRITE(pd[1].data)						
3	UNLOCK(pd[1].lock)						
1	UNLOCK(pd[0].lock)						
2	TEST(pd[1].lock)						
4	TEST(pd[0].lock)						