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 Μαρτίου 2012
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) Ένας επεξεργαστής διαθέτει 32 καταχωρητές, χρησιμοποιεί immediate μήκους 16 bits και διαθέτει συνολικά 142 εντολές στο ISA του. Έστω ένα πρόγραμμα με τα εξής χαρακτηριστικά:

a. 20% των εντολών έχουν 1 καταχωρητή προέλευσης και 1 καταχωρητή προορισμού
b. 30% έχουν 2 καταχωρητές προέλευσης και 1 προορισμού
c. 25% έχουν 1 καταχωρητή προέλευσης, 1 προορισμού και 1 immediate ως τελεστέο προέλευσης
d. 25% έχουν 1 immediate ως τελεστέο προέλευσης και 1 καταχωρητή προορισμού.

(i) Πόσα bits απαιτούνται για την κωδικοποίηση καθενός από τους παραπάνω τύπους εντολών; Υποθέστε ότι το ISA απαιτεί το μήκος των εντολών να είναι πολλαπλάσιο των 8 bits.

Για το id του κάθε καταχωρητή χρειαζόμαστε 5 bits, για κάθε immediate 16 και για κάθε opcode 8. Επομένως για κάθε τύπο εντολής χρειαζόμαστε συνολικά :

a. 8 + 5 + 5 = 18 → 24 bits
b. 8 + 5 + 5 + 5 = 23 → 24 bits
c. 8 + 5 + 5 + 16 = 34 → 40 bits
d. 8 + 16 + 5 = 29 → 32 bits

(ii) Αν σας έλεγαν ότι ο επεξεργαστής αυτός διαθέτει μια πολύ μικρή instruction cache και σας ζητούσαν να αποφασίσετε αν το ISA θα είναι RISC ή CISC, ποια θα ήταν η επιλογή σας; Γιατί;

Αν ο επεξεργαστής είναι RISC, τότε όλες οι εντολές πρέπει να έχουν το ίδιο μήκος, δηλαδή 40 bits. Αντίθετα στην περίπτωση του CISC, οι εντολές έχουν μεταβλητό μήκος. Επομένως, κατά μέσο όρο θα χρειαζόμασταν:

$$0.2 * 24 + 0.3 * 24 + 0.25 * 40 + 0.25 * 32 = 30 \text{ bits}$$

Δηλαδή για ένα CISC επεξεργαστή θα χρειαζόμασταν το 75% του χώρου που θα χρειαζόμασταν για να αποθηκεύσουμε το ίδιο πρόγραμμα σε ένα RISC επεξεργαστή. Καθώς λοιπόν ο επεξεργαστής μας διαθέτει πολύ μικρή instruction cache, επιλέγουμε το ISA μας να είναι CISC.

Β) Θεωρήστε έναν επεξεργαστή με multi-cycle datapath. Δίνεται ότι οι αριθμητικές εντολές απαιτούν 4 κύκλους, οι εντολές μνήμης 6 κύκλους και οι εντολές διακλάδωσης 2 κύκλους, ενώ η κατανομή τους σε ένα μέσο πρόγραμμα είναι 50%, 35% και 15% αντίστοιχα. Δίνεται επίσης ότι η επέκταση του επεξεργαστή για την υποστήριξη pipelining προσθέτει επιπλέον καθυστέρηση 0.34 nsec στον κύκλο του ρολογιού.

Ποιο θα πρέπει να είναι το μέγιστο μέγεθος του κύκλου του ρολογιού ώστε η εκτέλεση ενός μέσου προγράμματος να είναι πιο γρήγορη στον non-pipelined επεξεργαστή σε σχέση με τον pipelined;

$T_1 < T_2 \rightarrow$ Instructions * CPI_1 * T_{cycle} < Instructions * CPI_2 * (T_{cycle} + 0.34)
$\rightarrow$ (0.5 * 4 + 0.35 * 6 + 0.15 * 2) * T_{cycle} < 1 * (T_{cycle} + 0.34)
$\rightarrow$ 4.4 * T_{cycle} < T_{cycle} + 0.34
$\rightarrow$ T_{cycle} < 0.1 nsec

Γ) Ποια η διαφορά μεταξύ μιας εξάρτησης και ενός κινδύνου; Δώστε τους ορισμούς των WAW, WAR και RAW εξαρτήσεων. Εξηγήστε ποιοι από τους κινδύνους αυτούς εμφανίζονται και ποιοι όχι στην κλασσική αρχιτεκτονική σωλήνωσης 5 σταδίων του MIPS.

Η εξάρτηση είναι χαρακτηριστικό του κώδικα ενώ ο κίνδυνος της αρχιτεκτονικής στην οποία εκτελείται ο κώδικας αυτός. Μια εξάρτηση δεν είναι υποχρεωτικό να οδηγήσει σε κίνδυνο.

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.

Ο RAW κίνδυνος εμφανίζεται όταν μια εντολή προσπαθεί να διαβάσει από έναν καταχωρητή πριν τον γράψει μια προηγούμενη εντολή.
πχ. ADD R1, R2, R3
ADD R5, R4, R1
Στον MIPS ο κίνδυνος αυτός εμφανίζεται και επιλύεται με χρήση forwarding και stalls.

Θέμα 2ο (30%)

Υποθέστε την αρχιτεκτονική σωλήνωσης αποτελούμενη από τα εξής στάδια : IF, ID, MEM, EX, WB. Στην αρχιτεκτονική αυτή δεν υποστηρίζεται χρήση offset κατά την έμμεση διευθυνσιοδότηση, καθώς οι εντολές μνήμης δε χρησιμοποιούν την ALU. Επίσης, οι αριθμητικές και λογικές εντολές έχουν τη δυνατότητα να προσπελάσουν απευθείας μια θέση μνήμης (ως τελεστέο προέλευσης). Όλα τα στάδια διαρκούν ένα κύκλο. Κατά τον εντοπισμό μιας εντολής άλματος υπό συνθήκη, ο επεξεργαστής κάνει stall τη σωλήνωση μέχρι την επίλυση η οποία πραγματοποιείται στο στάδιο EX. Τέλος, υποθέστε ότι η εγγραφή σε ένα καταχωρητή γίνεται στο πρώτο μισό ενός κύκλου, ενώ η ανάγνωση από τον ίδιο καταχωρητή πραγματοποιείται στο δεύτερο μισό του κύκλου.

Δίνεται το ακόλουθο κομμάτι κώδικα :

```
1.              ADDI  R2, R1, #200
2.              ADDI  R3, R1, #196
3.    LOOP:     ADDI  R3, R3, #4
4.              LD    R5, (R1)
5.              MUL   R4, R5, (R3)
6.              ADD   R4, R4, R5
7.              ST    R4, (R1)
8.              ADDI  R1, R1, #4
9.              SUB   R6, R2, R1
10.             BNEZ  R6, LOOP
```

Α) 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;

Β) Υποθέστε τώρα ότι υπάρχουν όλα τα δυνατά σχήματα προώθησης. 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;

Γ) Θεωρώντας την ίδια σωλήνωση με το ερώτημα Β, μπορείτε να επιτύχετε καλύτερη επίδοση αναδιατάσσοντας τον κώδικα (με τις απαραίτητες βέβαια αλλαγές για να μην αλλάξετε την σημασιολογία του προγράμματος); 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 βρόχου;

Α. Ο βρόχος θα εκτελεστεί **200/4 = 50** φορές. Συνολικά για ολόκληρο τον κώδικα θα χρειαστούμε **2 + 23 + 48 * 21 + 22 = 1055 κύκλους**.

	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	29	30	31
ADDI R2,R1,#200	F	D	M	X	W																										
ADDI R3,R1,#196		F	D	M	X	W																									
ADDI R3,R3,#4			F	D	-	-	M	X	W																						
LD R5,(R1)				F	-	-	D	M	X	W																					
MUL R4,R5,(R3)							F	D	-	-	M	X	W																		
ADD R4,R4,R5								F	-	-	D	-	-	M	X	W															
ST R4,(R1)											F	-	-	D	-	-	M	X	W												
ADDI R1,R1,#4														F	-	-	D	M	X	W											
SUB R6,R2,R1																	F	D	-	-	M	X	W								
BNEZ R6,LOOP																		F	-	-	D	-	-	M	X	W					
ADDI R3,R3,#4																										F	D	M	X	W	
LD R5,(R1)																											F	D	M	X	W

B. Ο βρόχος θα εκτελεστεί **200/4 = 50** φορές. Συνολικά για ολόκληρο τον κώδικα θα χρειαστούμε **2 + 49 * 12 + 13= 603 κύκλους.**

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
ADDI R2,R1,#200	F	D	M	X	W															
ADDI R3,R1,#196		F	D	M	X	W														
ADDI R3,R3,#4			F	D	M	X	W													
LD R5,(R1)				F	D	M	X	W												
MUL R4,R5,(R3)					F	D	M	X	W											
ADD R4,R4,R5						F	D	M	X	W										
ST R4,(R1)							F	D	-	M	X	W								
ADDI R1,R1,#4								F	-	D	M	X	W							
SUB R6,R2,R1										F	D	M	X	W						
BNEZ R6,LOOP											F	D	M	X	W					
ADDI R3,R3,#4															F	D	M	X	W	
LD R5,(R1)																F	D	M	X	W

Γ. Ο βρόχος θα εκτελεστεί **200/4 = 50** φορές. Συνολικά για ολόκληρο τον κώδικα θα χρειαστούμε **2 + 49 * 11 + 12= 553 κύκλους.**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ADDI R2,R1,#200	F	D	M	X	W													
ADDI R3,R1,#200		F	D	M	X	W												
LD R5,(R1)			F	D	M	X	W											
MUL R4,R5,(R3)				F	D	M	X	W										
ADD R4,R4,R5					F	D	M	X	W									
ADDI R3,R3,#4						F	D	M	X	W								
ST R4,(R1)							F	D	M	X	W							
ADDI R1,R1,#4								F	D	M	X	W						
SUB R6,R2,R1									F	D	M	X	W					
BNEZ R6,LOOP										F	D	M	X	W				
LD R5,(R1)														F	D	M	X	W

Θέμα 3ο (25%)

Α) Έστω ένας επεξεργαστής με 2 επίπεδα ιεραρχίας μνήμης: μια L1 cache και την κύρια μνήμη. Τα 2 επίπεδα αυτά συνδέονται μέσω ενός 32-bit διαδρόμου. Ένα hit στην cache ολοκληρώνεται μέσα σε 1 κύκλο του ρολογιού. Αντίθετα, για την εξυπηρέτηση ενός miss απαιτείται η φόρτωση ενός block, η οποία πραγματοποιείται στέλνοντας τη διεύθυνση (32 bits) στην κύρια μνήμη, η προσπέλαση της οποίας διαρκεί 4 κύκλους προτού να είναι έτοιμη να στείλει στην cache το ζητούμενο block. Το miss ικανοποιείται εφόσον έρθει στην cache ολόκληρο το ζητούμενο block. Κάθε μεταφορά δεδομένων στο διάδρομο διαρκεί 1 κύκλο. Δίνονται τα παρακάτω miss rates για διαφορετικά μεγέθη του block:

Block Size (Bytes)	Miss Ratio (%)
4	4.5
16	2.4
32	1.6
64	1.0
128	0.64

Ποιο από τα παραπάνω μεγέθη block μας δίνει το πιο γρήγορο σύστημα;

Αυτό που θα αλλάζει κάθε φορά είναι το κόστος των misses, το οποίο, με δεδομένο ότι το bus έχει δυνατότητα μεταφοράς 4 bytes σε κάθε κύκλο, θα είναι:

Cost = miss_rate * [1 + 4 + (block_size_bytes / 4)]

Έτσι για κάθε περίπτωση θα έχουμε:

a. Cost = 0.045 * (1 + 4 + 4/4) = 0.27
b. Cost = 0.024 * (1 + 4 + 16/4) = 0.216
c. Cost = 0.016 * (1 + 4 + 32/4) = 0.208
d. Cost = 0.010 * (1 + 4 + 64/4) = 0.21
e. Cost = 0.0064 * (1 + 4 + 128/4) = 0.2368

Επομένως το πιο γρήγορο σύστημα το έχουμε για block_size = 32 Bytes.

Β) Δίνεται μια direct mapped cache μεγέθους 2KB με block size 64 bytes. Η μικρότερη μονάδα που μπορεί να διευθυνσιοδοτηθεί είναι το 1 byte. Υπολογίστε το συνολικό μέγεθος του tag array για μια αρχιτεκτονική που χρησιμοποιεί διευθύνσεις μήκους 48 bits. Αν για την ίδια αρχιτεκτονική αλλάξουμε την οργάνωση της cache σε 4-way associative (διατηρώντας σταθερή τη χωρητικότητα και το μέγεθος του cache line) πόσο θα μεταβληθεί το tag array;

$$sets = \frac{size}{block\_size} = \frac{2 * 2^{10}}{2^6} = 2^5 = 32 \rightarrow index = 5\ bits$$

$$block_{size} = 64\ bytes \rightarrow block\_offset = 6\ bits$$

Άρα $tag = 48 - 5 - 6 = 37\ bits$. Κάθε block έχει ένα tag και άρα το συνολικό μέγεθος του tag array είναι $tag\_array\_size = 37 * 32 = 1184\ bits = 148\ bytes = 0.1445\ KB$.

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 4-way asssociative θα έχουμε το ¼ των sets και άρα χρειαζόμαστε 3 bits για το index. Επομένως το πεδίο tag αυξάνεται σε 39 bits και το νέο μέγεθος του tag array θα είναι

$$tag\_array\_size = 39 * 32 = 1248\ bits = 156\ bytes = 0.1523\ KB.$$

Γ) Αναφέρατε ένα πλεονέκτημα και ένα μειονέκτημα των παρακάτω βελτιστοποιήσεων μιας cache:
(i) Μείωση μεγέθους της cache
(ii) Αύξηση συσχετιστικότητας
(iii) Χρήση block μεγαλύτερου μεγέθους

Πλεονεκτήματα
(i) Μικρότερο access time / hit time, μικρότερη κατανάλωση ενέργειας
(ii) Μείωση capacity misses
(iii) Μείωση compulsory misses, καλύτερη εκμετάλλευση της χωρικής τοπικότητας

Μειονεκτήματα
(i) Αύξηση των capacity misses
(ii) Αύξηση του access time/hit time, αύξηση της πολυπλοκότητας
(iii) Πιθανή αύξηση capacity misses, μεγαλύτερο miss penalty καθώς χρειάζεσαι περισσότερους κύκλους για να φέρεις το block στην cache

Δ) Πως επηρεάζει το TLB την απόδοση της ιεραρχίας μνήμης αν η cache είναι physically ή virtually addressed;

Στην περίπτωση της physically addressed cache, η δεικτοδότηση της cache γίνεται με την real address. Έτσι, πριν από κάθε access πρέπει η διεύθυνση να περάσει πρώτα από το TLB προκειμένου να βρούμε τη real address. Επομένως, το TLB βρίσκεται στο critical path.

Αντίθετα, στην περίπτωση των virtual addressed caches, το σύστημα χρησιμοποιεί τη virtual address για τη δεικτοδότηση της cache. Αυτό σημαίνει, ότι ταυτόχρονα με την προσπέλαση του TLB και πριν ολοκληρωθεί ακόμα η μετάφραση και παραχθεί η real address, το σύστημα μπορεί να προσπελάσει την cache και να βρει τα blocks εκείνα που θα ελέγξει προκειμένου να διαπιστώσει αν πρόκειται για hit ή miss. Επομένως, σε σύγκριση με τις physically addressed caches, μειώνεται ο χρόνος πρόσβασης.

Θέμα 4ο (25%)

Δίνεται ο παρακάτω κώδικας γραμμένος σε C.

```
double c[32], a[8][32], b[34];

for (i=0; i<8; i++)
   for (j=0; j<32; j++)
      c[j] = c[j] + a[i][j]*b[j+2];
```

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 (L1), η οποία αρχικά είναι άδεια. Η κρυφή μνήμη είναι 2-way associative, write-allocate, χρησιμοποιεί LRU πολιτική αντικατάστασης και έχει χωρητικότητα 256B. 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 αποθηκευμένοι στην κύρια μνήμη κατά γραμμές. Το πρώτο στοιχείο του πίνακα c βρίσκεται στη διεύθυνση 0x00001F80.

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

#blocks = 256/32 = 8, άρα #sets = 4, άρα index=2bits
byte offset = log(32) = 5 bits
Άρα το 1^{ο} στοιχείο του c απεικονίζεται στο set 0 (0001 1111 1**00**0 0000)

c0 a00 b2 c0 m m m m
c1 a01 b3 c1 h m m m
c2 a02 b4 c2 h m m h
c3 a03 b5 c3 h h h h

c4 a04 b6 c4 m m m m
c5 a05 b7 c5 h m m m
c6 a06 b8 c6 h m m h
c7 a07 b9 c7 h h h h

c8 a08 b10 c8 m m m m
c9 a09 b11 c9 h m m m
...
c31 a0,31 b33 c31 h h h h

miss rate = 9/16 = 56.25%

Β) Αν αντικαταστήσουμε τη L1 cache με μία 4-way associative ίδιου συνολικού μεγέθους και ίδιου μεγέθους block, ποιο θα είναι το νέο ποσοστό αστοχίας;

c0 a00 b2 c0 m m m h
c1 a01 b3 c1 h h h h
c2 a02 b4 c2 h h m h
c3 a03 b5 c3 h h h h

c4 a04 b6 c4 m m h h
c5 a05 b7 c5 h h h h
c6 a06 b8 c6 h h m h
c7 a07 b9 c7 h h h h

c8 a08 b10 c8 m m h h
c9 a09 b11 c9 h h h h
...

miss rate = (4 + 7*3)/8*16 = 19.5%

Γ) Έστω ότι ο χρόνος πρόσβασης στην 2-way associative cache του ερωτήματος Α είναι 1 nsec, ενώ σε περίπτωση miss η πρόσβαση στα επόμενα επίπεδα της ιεραρχίας απαιτεί 11 nsec. Ποιος είναι ο μέγιστος επιτρεπόμενος χρόνος πρόσβασης για τη 4-way associative cache του ερωτήματος Β, ώστε να συμφέρει η επιλογή της από άποψη χρόνου εκτέλεσης (ο χρόνος πρόσβασης στα επόμενα επίπεδα της ιεραρχίας μνήμης παραμένει σταθερός);
AMAT(2-way) = 1 + 0.5625*11 = 7.1875
Θέλουμε AMAT(4-way) ≤ AMAT(2-way) ↔ x + 0.195*11 ≤ 7.1875 ↔ x ≤ 5.0425 nsec