

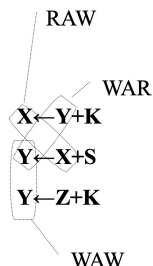
Θέμα 1

Ζητούμενο 1

Α) Ποια η διαφορά μεταξύ μιας εξάρτησης και ενός κινδύνου; Δώστε τους ορισμούς των WAW, WAR και RAW εξαρτήσεων. Εξηγήστε ποιες από τις παραπάνω εξαρτήσεις οδηγούν σε κινδύνους και ποιες όχι στην κλασική αρχιτεκτονική σωλήνωσης 5 σταδίων του MIPS.

Κίνδυνοι Σωλήνωσης (Pipeline Hazards)

- **Δομικοί Κίνδυνοι** (structural hazards): Το υλικό δεν μπορεί να υποστηρίξει το συνδυασμό των εντολών που θέλουμε να εκτελέσουμε στον ίδιο κύκλο μηχανής. (π.χ. ενιαία L1 \$ για I & D)
- **Κίνδυνοι Ελέγχου** (control hazards): Το υλικό δεν μπορεί να προχωρήσει την εκτέλεση επόμενων εντολών καθώς αναμένεται η ολοκλήρωση της εκτέλεσης μιας εντολής (π.χ. Branches)
- **Κίνδυνοι Δεδομένων** (data hazards): Εμφανίζονται όταν οι εντολές έχουν εξαρτήσεις δεδομένων (RAW, WAR, WAW).

**Εξαρτήσεις Δεδομένων**

- **RAW** (Read-After-Write) (true-dependence): Η ανάγνωση ενός καταχωρητή πρέπει να ακολουθεί την εγγραφή στον ίδιο καταχωρητή από προηγούμενη εντολή
-> οδηγεί σε data hazards: η εντολή που διαβάζει από τον καταχωρητή μπορεί να διαβάσει την παλιά τιμή αν η γραφή δεν έχει ολοκληρωθεί ακόμη
- **WAR**: (Write-After-Read) (anti-dependence): Η εγγραφή σε ένα καταχωρητή πρέπει να ακολουθεί την ανάγνωσή του από προηγούμενη εντολή
-> δεν οδηγεί σε data hazards: το Register Read είναι το 2ο στάδιο ενώ το Register Write το 5ο.
- **WAW**: (Write-After-Write) (output-dependence): Η εγγραφή σε ένα καταχωρητή πρέπει να ακολουθεί όλες τις εγγραφές στον ίδιο καταχωρητή από προηγούμενες εντολές
-> δεν οδηγεί σε data hazards: οι γραφές γίνονται σειριακά στο 5ο στάδιο κάθε εντολής

Ζητούμενο 2

B) Αναλαμβάνετε επικεφαλής σχεδίασης του νέου υπολογιστικού συστήματος της εταιρίας σας. Το προηγούμενο σύστημα αποτελούνταν από έναν single-cycle επεξεργαστή, Instruction και Data L1 caches και την κύρια μνήμη. Η μελέτη του με τη χρήση κατάλληλων μετροπρογραμμάτων, έχει φανερώσει τα παρακάτω χαρακτηριστικά:

Η ομάδα σχεδίασης σας έχει προτείνει 2 εναλλακτικές προκειμένου να βελτιώσετε την απόδοση:

- Αντικατάσταση του επεξεργαστή με έναν πιο καινούριο (πάντα single-cycle), ο οποίος λειτουργεί με 2.5 φορές μεγαλύτερη ταχύτητα. Οι L1 caches του επεξεργαστή αυτού λειτουργούν επίσης 2.5 φορές πιο γρήγορα από τις προηγούμενες, προκειμένου να τροφοδοτούν τον επεξεργαστή με τον ίδιο ρυθμό. Η κύρια μνήμη δεν αλλάζει.
- Αντικατάσταση της κύριας μνήμης με καινούρια η οποία λειτουργεί σε διπλάσια ταχύτητα. Ο επεξεργαστής και οι L1 caches δεν αλλάζουν.

- Ποσοστό memory instructions: 20%
- Instruction cache miss rate: 2%
- Data cache miss rate: 8%
- L1 cache miss penalty: 30 κύκλοι
- L1 cache hit penalty: 0 κύκλοι

instructions → 20%
data → 80%

current:

$$\begin{aligned} t_{instr} &= t_{hit} + \text{miss rate}^{inst} \cdot t_{miss} \text{ penalty} = 0 + 0,02 \cdot 30 = 0,6 \text{ cc} \\ t_{data} &= t_{hit} + \text{miss rate}^{data} \cdot t_{miss} \text{ penalty} = 0 + 0,08 \cdot 30 = 2,4 \text{ cc} \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{avg} = 20\% \cdot t_{instr} + 80\% \cdot t_{data} = 20\% \cdot 0,6 + 80\% \cdot 2,4 = 2,04 \text{ cc}$$

proposal 1:

ταχύτητα επεξ $\times 2,5$ ⊕ ταχύτητα L1 cache $\times 2,5 \Rightarrow \text{miss penalty}' = 30/2,5 = 12 \text{ cc}$

$$\begin{aligned} t_{instr}' &= 0 + 0,02 \cdot 12 = 0,24 \text{ cc} \\ t_{data}' &= 0 + 0,08 \cdot 12 = 0,96 \text{ cc} \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{avg}' = 20\% \cdot 0,24 + 80\% \cdot 0,96 = 0,816 \text{ cc}$$

proposal 2:

ταχύτητα κύριας μνήμης $\times 2 \Rightarrow \text{penalty miss}' = 30/2 = 15 \text{ cc}$

$$\begin{aligned} t_{instr}' &= 0 + 0,02 \cdot 15 = 0,3 \text{ cc} \\ t_{data}' &= 0 + 0,08 \cdot 15 = 1,2 \text{ cc} \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{avg}'' = 20\% \cdot 0,3 + 80\% \cdot 1,2 = 1,02 \text{ cc}$$

$\text{avg}' < \text{avg}'' < \text{avg} \Rightarrow \text{η πρόταση 1 είναι καλύτερη}$

Ζητούμενο 3

- $\sim \Sigma_{\text{gas}}$

single cycle: $CPI = 1$ (naïve)

pipelined: $CPI = 1$ (for execution x86 instructions) + stall α ogw branch + stall α ogw LW
 α ogw and RAW hazard.

Summary, on given $CP_{singlecycle} = 1 < CP_{pipelined}$

- Nidos

single cycle: ο κύκλος είναι ίσος με την πιο αργή εντολή (nx lw)

pipelined: 0 KUKKOs eval 100s μ sec to nio appo radio (nx Memory, ALU)

Juvenius, $T_{\text{single-cycle}} > T_{\text{pipelined}}$

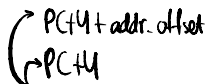
- ~ Nidox

multi cycle: οι εντολές εκτελούνται σειριακά, και κάθε εντολή

αποκρίνωται να αρχίσει η επόμενη → όχι stable λόγω έλλειψης δεδομένων

pipelined: nonres. errors exceed output capacity $\Rightarrow$ no own equation
still allow error detection

- $\sim \Sigma_{\text{max}}$



σε αριθμόν με single cycle που χρησιμοποιούμε 1 AW και 2 απορίες

Θέμα 2

Υποθέστε την κλασική αρχιτεκτονική σωλήνωσης του MIPS αποτελούμενη από τα στάδια IF, ID, EX, MEM, WB. 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 ότι η αρχική τιμή του καταχωρητή \$t9 (ακριβώς πριν το loop) είναι ίση με 0x400 καθώς και το ακόλουθο κομμάτι κώδικα:

```
1.  LOOP: LW    $t0, 0($t1)
2.      ADD    $t0, $t0, $t2
3.      LW     $t3, 0($t0)
4.      SUBI   $t1, $t3, 1
5.      SW     $t3, 400($t0)
6.      LW     $t2, 4($t1)
7.      ADDI   $t9, $t9, -4
8.      BNEZ   $t9, LOOP
```

Ζητούμενο 1

A) Υποθέστε ότι δεν υπάρχουν σχήματα προώθησης. Εκτελέστε την 1η επανάληψη του βρόχου (μέχρι και την 1η εντολή της 2ης 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 κώδικα;

$$\begin{aligned} \$9 &= 0 \times 400 = 4 \cdot 16^2 = 2^0 \\ \text{κάθε } \text{loop} &: \$t9 = -4 \end{aligned} \quad \left. \begin{aligned} & \\ & \end{aligned} \right\} \# \text{loops} = 2^0 / 4 = 2^8 = 256$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
lw	F	D	X	M	W																			
add		F	D	-	-	X	M	W																
lw			F	-	-	D	-	-	X	M	W													
subi					F	-	-	D	-	-	X	M	W											
sw							F	-	-	D	X	M	W											
lw									F	D	-	X	M	W										
addi										F	-	D	X	M	W									
bnez											F	D	-	-	X	M	W							
lw													F	D	X	M	W							

$$\text{total cycles: } 255 \cdot 19 \text{cc} + 21 \text{cc} = 4866 \text{cc}$$

Ζητούμενο 2

B) Υποθέστε τώρα ότι υπάρχουν όλα τα δυνατά σχήματα προώθησης. Δείξτε όπως και πριν το διάγραμμα χρονισμού για την 1η επανάληψη του βρόχου, υποδεικνύοντας τις προωθήσεις που γίνονται. Πόσοι κύκλοι απαιτούνται τώρα για την εκτέλεση του κώδικα;

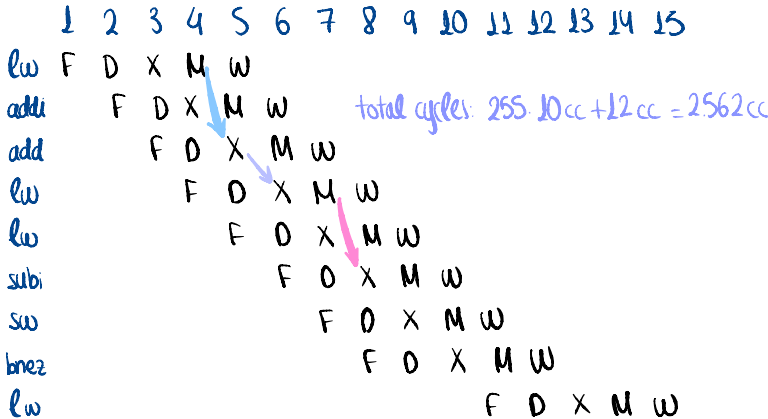
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
lw	F	D	X	M	W												
add		F	D	-	X	M	W										
lw			F	-	D	X	M	W									
subi				F	D	-	X	M	W								
sw					F	-	D	X	M	W							
lw						F	D	X	M	W							
addi							F	D	X	M	W						
bnez								F	D	X	M	W					
lw									F	D	X	M	W				

$$\text{total cycles: } 255 \cdot 12 \text{cc} + 14 \text{cc} = 3074 \text{cc}$$

Ζητούμενο 3

Γ) Θεωρώντας την ίδια σωλήνωση με το ερώτημα Β, μπορείτε να επιτύχετε τη βέλτιστη επίδοση αναδιατάσσοντας τον κώδικα (με τις απαραίτητες βέλτιστες μετατροπές για να μην αλλάξετε την σημασιολογία του προγράμματος); Δείξτε όπως και πριν το διάγραμμα χρονισμού για την 1η επανάληψη του βρόχου, υποδεικνύοντας τις προωθήσεις που γίνονται. Πόσοι κύκλοι απαιτούνται τώρα για την εκτέλεση του βρόχου;

1.	LOOP: LW	\$t0, 0(\$t1)	1.	LOOP: LW	\$t0, 0(\$t1)
2.	ADD	\$t0, \$t0, \$t2	7.	ADDI	\$t9, \$t9, -4
3.	LW	\$t3, 0(\$t0)	2.	ADD	\$t0, \$t0, \$t2
4.	SUBI	\$t1, \$t3, 1	3.	LW	\$t3, 0(\$t0)
5.	SW	\$t3, 400(\$t0)	6.	LW	\$t2, 4(\$t1)
6.	LW	\$t2, 4(\$t1)	4.	SUBI	\$t1, \$t3, 1
7.	ADDI	\$t9, \$t9, -4	5.	SW	\$t3, 400(\$t0)
8.	BNEZ	\$t9, LOOP	8.	BNEZ	\$t9, LOOP



Θέμα 3

Ζητούμενο 1

Α) Δίνεται το παρακάτω πρόγραμμα σε C που υπολογίζει το άθροισμα των μικρότερων στοιχείων δύο πινάκων. Δίνεται επίσης μια μετάφραση αυτού του προγράμματος σε assembly MIPS. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής \$0 (ή \$zero) είναι πάντα μηδέν.

```
int sum_min_arrays (int *x, int *y,
int n) {
    int min1 = x[0];
    int min2 = y[0];
    for (int i = 0; i < n-1; i++) {
        x++;
        y++;
        if (*x < min1)
            min1 = *x;
        if (*y < min2)
            min2 = *y;
    }
    return min1+min2;
}
```

LOOP:

CHECK2:

NEXT:

END:

```
addi $sp, $sp, -8
sw $s1, 4($sp)
sw $s0, 0($sp)
lw $s0, 0($a0)
lw $s1, 0($a1)
add $t3, $0, $0
subi $t2, $t2, 1
slt $t4, $t3, $a2
beq $t4, $0, END
addi $a0, $a0, 4
addi $a1, $a1, 4
lw $t0, 0($a0)
lw $t1, 0($a1)
slt $t4, $t0, $s0
beq $t4, $zero, CHECK2
add $s0, $t0, $0
slt $t4, $t1, $s1
beq $t4, $zero, NEXT
add $s1, $t1, $0
NEXT:
addi $t3, $t3, 1
j LOOP
END:
add $v0, $s1, $s2
lw $s0, 0($sp)
lw $s1, 4($sp)
addi $sp, $sp, 8
jr $ra
```

\$a0 → &x[i]
\$a1 → &y[i+1]
\$s0 → min1
\$s1 → min2
\$t3 → i-1
\$a2 → n-1
if (i < n-1), end
\$a0 → &x[i+1]
\$a1 → &y[i+1]
\$t0 → x[i]
\$t1 → y[i]
if (*x < min1), check2
else min1 = *x
if (*y < min2), next
else min2 = *y
i++
\$v0 → min1+min2
restore stack
return

Ζήτημα 2

B) Δίνεται επεξεργαστής με L1 data cache με συνολικά 4 blocks η οποία χρησιμοποιεί πολιτική αντικατάστασης LRU. Η cache αρχικά είναι άδεια και έστω $x, x+1, x+2, x+3, \dots$, οι διευθύνσεις διαδοχικών μπλοκ της μνήμης του συστήματος. Εκτελείτε ένα πρόγραμμα, το οποίο πραγματοποιεί τις παρακάτω προσβάσεις (με τη σειρά που δίνονται): $x, x+2, x+4$

Κάνοντας μόνο 2 ακόμα προσβάσεις στη μνήμη και παρακολουθώντας το hit rate βρείτε ποια είναι η οργάνωση της L1 cache;

4 blocks: $\begin{cases} 1 \text{ set} \times 4 \text{ blocks} \rightarrow \text{fully associative} \\ 2 \text{ sets} \times 2 \text{ blocks} \rightarrow \text{2-way associative} \\ 4 \text{ sets} \times 1 \text{ block} \rightarrow \text{directed map} \end{cases}$

αδεια cache
 $\rightarrow$ access $x, x+2, x+4 \rightarrow$ compulsory misses
 $\rightarrow$ access $x, x+4$

① fully associative:

set 0				
-------	--	--	--	--

access $x, x+2, x+4 \rightarrow$ compulsory misses

↓ LRU

set 0	x	x+2	x+4	
-------	---	-----	-----	--

access $x, x+4 \rightarrow$ hits

↓ LRU

set 0	x	x+2	x+4	
-------	---	-----	-----	--

② two-way associative:

access $x, x+2, x+4 \rightarrow$ comp misses

access $x, x+4 \rightarrow$ contl miss
 $\rightarrow$ hit

set 0		
set 1		

x+4	x+2

← LRU

x+4	x

← LRU

③ directed map

access $x, x+2, x+4 \rightarrow$ comp misses

access $x, x+4 \rightarrow$ contl misses

block 0	
block 1	
block 2	
block 3	

block 0	x+4
block 1	
block 2	x+2
block 3	

block 0	x+4
block 1	
block 2	x+2
block 3	

access $x, x+2, x+4$

access $x, x+4$

$\rightarrow$ compulsory misses

$\sim$ (hit, hit) $\rightarrow$ fully associative

$\sim$ (miss, hit) $\rightarrow$ 2-way associative

$\sim$ (miss, miss) $\rightarrow$ directed map

(hit, miss) $\rightarrow$ advance

Θέμα 4

Οι πίνακες περιέχουν στοιχεία κινητής υποδιαστολής διπλής ακρίβειας, μεγέθους 8 bytes το καθένα. Κάνουμε τις εξής υποθέσεις:

- > Το πρόγραμμα εκτελείται σε έναν 32-bit επεξεργαστή με ένα μόνο επίπεδο κρυφής μνήμης δεδομένων, η οποία αρχικά είναι άδεια. Η κρυφή μνήμη είναι 2-way set associative, write-allocate, έχει χωρητικότητα 4KB και χρησιμοποιεί πολιτική αντικατάστασης LRU. Το μέγεθος του block είναι 64 bytes, ενώ η μικρότερη μονάδα δεδομένων που μπορεί να διευθυνοδοτηθεί είναι το 1 byte.
- > Όλες οι μεταβλητές πλην των στοιχείων των πινάκων αποθηκεύονται σε καταχωρητές του επεξεργαστή κι επομένως οποιαδήποτε αναφορά σε αυτές δεν συνεπάγεται προσπάθεια στην κρυφή μνήμη.
- > Σε επίπεδο εντολών assembly οι αναγνώσεις γίνονται με τη σειρά που εμφανίζονται στον κώδικα.
- > Οι πίνακες είναι ευθυγραμμισμένοι και αποθηκεύονται κατά γραμμές.

Ζητούμενο 1

A) Βρείτε το συνολικό αριθμό hits και misses για όλη την εκτέλεση του παραπάνω κώδικα.

$$\begin{aligned} \text{στοιχείο size} &= 8 \text{ bytes} \\ \text{block size} &= 64 \text{ bytes} \end{aligned} \left\{ \begin{aligned} \text{στοιχεία/block} &= 64/8 = 8 \\ &+ \text{αρχ. κατά γραμμές} \end{aligned} \right. \Rightarrow \begin{aligned} a[i][0-7] &\sim \text{ίδιο block} \\ b[i][0-7] &\sim \text{ίδιο block} \end{aligned}$$

$$\begin{aligned} \text{cache size} &= 4 \text{K bytes} \\ \text{block size} &= 64 \text{ bytes} \end{aligned} \left\{ \begin{aligned} \# \text{blocks} &= 2^{10}/2^6 = 2^4 = 16 \\ \# \text{sets} &= \# \text{blocks} / 2 = 2^3 = 8 \end{aligned} \right. \Rightarrow \begin{aligned} a &\text{ χωράει } 128/32 = 4 \text{ φορές στη cache} \\ b &\text{ χωράει } 64/32 = 2 \text{ φορές στη cache} \end{aligned}$$

block 1: $a[i][0-7], a[i+32][0-7], a[i+64][0-7], a[i+96][0-7], b[i][0-7], b[i+32][0-7]$

$$\begin{aligned} \text{block size} &= 2^6 \text{ bytes} \Rightarrow \text{index} = 6 \text{ bits} \\ \# \text{sets} &= 2^3 \Rightarrow \text{offset} = 3 \text{ bits} \\ \text{address size} &= 32 \text{ bits} \Rightarrow \text{tag} = 23 \text{ bits} \end{aligned}$$



$i=0, j=0-7$

$a[0][0]$	$b[0][0]$	$a[32][0]$	$a[0][0]$	m	m	m	m
$a[0][1]$	$b[0][1]$	$a[32][1]$	$a[0][1]$	h	m	m	m
$a[0][2]$	$b[0][2]$	$a[32][2]$	$a[0][2]$	h	m	m	m
$a[0][3]$	$b[0][3]$	$a[32][3]$	$a[0][3]$	h	m	m	m
$a[0][4]$	$b[0][4]$	$a[32][4]$	$a[0][4]$	h	m	m	m
$a[0][5]$	$b[0][5]$	$a[32][5]$	$a[0][5]$	h	m	m	m
$a[0][6]$	$b[0][6]$	$a[32][6]$	$a[0][6]$	h	m	m	m
$a[0][7]$	$b[0][7]$	$a[32][7]$	$a[0][7]$	h	m	m	m

7 hits $\left\{ \begin{aligned} 3 \text{ compulsory} \\ 22 \text{ conflict} \end{aligned} \right\} = 25 \text{ misses}$

