ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

www.cslab.ece.ntua.gr

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΟΡΓΑΝΩΣΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Θέματα και Λύσεις Εξετάσεων Μαρτίου 2004

Θέμα 1ο (30%):

Έστω το τμήμα του C κώδικα:

```
for (i = 0; i < 12; i++)
     if (array[i] == k)
          array[i] = j;
```

Θεωρούμε ότι οι τιμές των μεταβλητών `4·i, j, k` βρίσκονται στους καταχωρητές `$s1, $s2, $s3` και η διεύθυνση του πίνακα `array` στον `$s4`. Το περιεχόμενο των καταχωρητών `$s1` και `$s5` είναι αρχικά 0 και 48, αντίστοιχα. Ο αντίστοιχος κώδικας σε assembly είναι ο εξής:

```
Loop:     add  $t1, $s1, $s4
          lw   $t0, 0($t1)
          bne  $t0, $s3, Next
          sw   $s2, 0($t1)
Next:     addi $s1, $s1, 4
          bne  $s1, $s5, Loop
Exit:
```

A

α) Δείξτε το διάγραμμα εκτέλεσης των επιμέρους φάσεων για κάθε εντολή, μέσα στην αρχιτεκτονική αγωγού, χωρίς την ύπαρξη κανενός σχήματος προώθησης (διάγραμμα εκτέλεσης όπως στις διαφάνειες του μαθήματος: IF - ID -…κλπ). Υποθέστε ότι κάθε φορά θεωρούμε ότι δεν πρόκειται να γίνει διακλάδωση (branch not taken) και ότι στο 75% των περιπτώσεων ισχύει η ισότητα `array[i]==k`. Πόσους κύκλους χρειάζεται για να εκτελεστεί ο παραπάνω βρόχος;

β) Δείξτε το διάγραμμα εκτέλεσης των επιμέρους φάσεων για κάθε εντολή, μέσα στην αρχιτεκτονική αγωγού, υποθέτοντας την ύπαρξη σχήματος προώθησης. Ισχύουν οι υποθέσεις του ερωτήματος (α) σε ό,τι αφορά τις εντολές διακλάδωσης και τον έλεγχο της ισότητας `array[i]==k`. Πόσους κύκλους χρειάζεται για να εκτελεστεί ο παραπάνω βρόχος; Μπορείτε να βελτιστοποιήσετε τον κώδικα (αποφυγή τυχόν pipeline stalls);

(γ) Η τεχνική loop unrolling (ξεδίπλωμα βρόχου) είναι μια από τις πιο διαδεδομένες τεχνικές βελτιστοποίησης κώδικα. Στην τεχνική αυτή, διαδοχικές επαναλήψεις του αρχικού βρόχου γράφονται σαν ξεχωριστές εντολές του ίδιου βρόχου (π.χ. για 2 διαδοχικές επαναλήψεις, ο unrolled βρόχος έχει πλέον βήμα 2). Στο παράδειγμά μας, ο αρχικός βρόχος ξεδιπλώνεται ως εξής:

```
for (i = 0; i < 12; i+=2) {
     if (array[i] == k)
          array[i] = j;

     if (array[i+1] == k)
          array[i+1] = j;
```

```
}
```

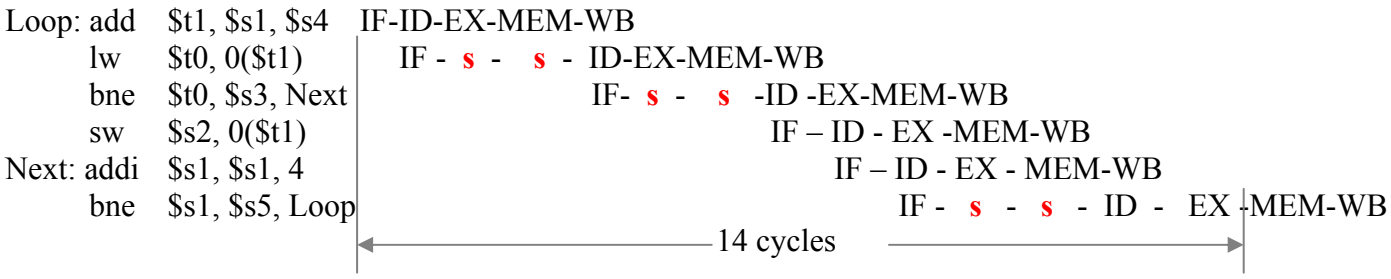
Εδώ, θέλουμε να αξιολογήσουμε τα πλεονεκτήματα υλοποίησης της τεχνικής ξεδιπλώματος βρόχων σε κώδικα που εκτελείται σε αρχιτεκτονική MIPS με χρήση αγωγού (pipeline datapath) και ύπαρξη σχήματος προώθησης (forwarding). Παρακάτω είναι το τμήμα του αρχικού κώδικα, όπου ο βρόχος είναι ξεδιπλωμένος μία φορά:

```
Loop:   add  $t1, $s1, $s4
        lw   $t0, 0($t1)
        bne  $t0, $s3, Next1
        sw   $s2, $0($t1)
Next1:  lw   $t2, 4($t1)                    B
        bne  $t2, $s3, Next2
        sw   $s2, 4($t1)
Next2:  addi $s1, $s1, 8
        bne  $s1, $s5, Loop
Exit:
```

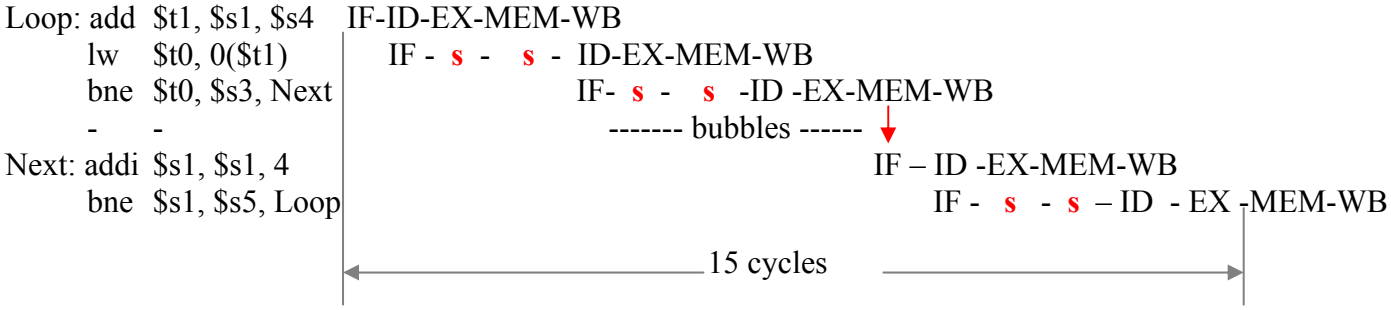
Σχεδιάστε ξανά τον κώδικα ώστε να αποφύγετε τις υποχρεωτικές καθυστερήσεις (pipeline stalls). Λαμβάνοντας υπόψη τα αναγκαία stalls συγκρίνετε τη διαφορά επίδοσης μεταξύ του αρχικού μη-ξεδιπλωμένου κώδικα (A) και του δικού σας (μετά το ξεδίπλωμα και τη βελτιστοποίηση του κώδικα B).

Λύση 1ου Θέματος

α) Χωρίς την ύπαρξη κανενός σχήματος προώθησης, η εκτέλεση θα γίνει ως εξής:



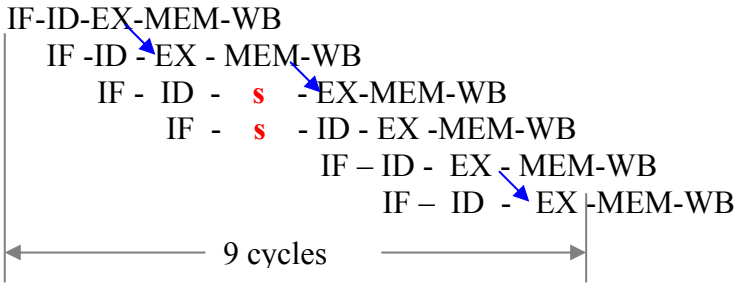
Η παραπάνω σειρά εκτέλεσης θα πραγματοποιηθεί για το 75% των επαναλήψεων (12×75% = 9 επαναλήψεις), όπου ισχύει η ισότητα `array[i]==k`. Κατά τις υπόλοιπες 12 – 9 = 3 επαναλήψεις, η σειρά εκτέλεσης θα είναι η ακόλουθη:



Επομένως, συνολικά χρειάζονται: 14×9 + 15×3 + 2= 173 κύκλοι για την ολοκλήρωση των 12 επαναλήψεων του βρόχου (+2 κύκλοι για το τελείωμα του τελευταίου bne).

β) Με προώθηση, χωρίς άλλη βελτιστοποίηση, η εκτέλεση θα γίνει ως εξής:

```
Loop:  add    $t1, $s1, $s4
       lw     $t0, 0($t1)
       bne    $t0, $s3, Next
       sw     $s2, 0($t1)
Next:  addi   $s1, $s1, 4
bne           $s1, $s5, Loop
```



Για το 75% των επαναλήψεων (9 επαναλήψεις) θα πραγματοποιηθεί η παραπάνω σειρά εκτέλεσης των εντολών. Κατά τις υπόλοιπες 3 επαναλήψεις, ομοίως με το ερώτημα (α), θα χρειαστεί 1 επιπλέον κύκλος, για την ολοκλήρωση της εκτέλεσης. Επομένως, στις 3 αυτές επαναλήψεις θα χρειαστούν 9+1 = 10 κύκλοι.

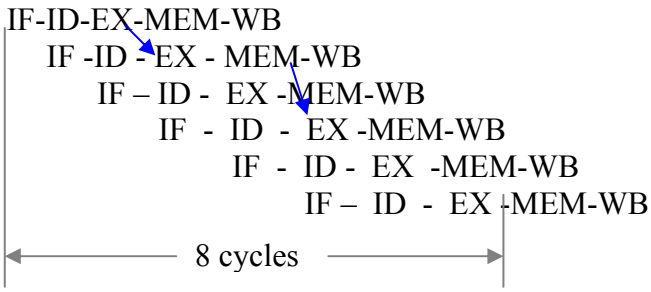
Συνολικά χρειάζονται: $9\times9 + 10\times3 + 2 = 113$ κύκλοι για την ολοκλήρωση των 12 επαναλήψεων του βρόχου.

Για να αποφύγουμε το αναγκαίο memory stall που εισάγει η εντολή φόρτωσης (lw), αναδιατάσσουμε τη σειρά εκτέλεσης των εντολών, προσέχοντας να μην επηρεάσουμε το τελικό αποτέλεσμα του βρόχου:

```
Loop:   add  $t1, $s1, $s4
        lw   $t0, 0($t1)
        addi $s1, $s1, 4
        bne  $t0, $s3, Next
        sw   $s2, 0($t1)
Next:   bne  $s1, $s5, Loop
Exit:
```

Αναλυτικότερα :

```
Loop:  add    $t1, $s1, $s4
       lw     $t0, 0($t1)
       addi   $s1, $s1, 4
       bne    $t0, $s3, Next
       sw     $s2, 0($t1)
Next:  bne    $s1, $s5, Loop
```



Επομένως, με τη βελτιστοποίηση, χρειάζονται 8 κύκλοι για το 75% των περιπτώσεων και 9 κύκλοι για το υπόλοιπο 25% των επαναλήψεων: $8\times9 + 9\times3 + 2 = 101$ κύκλοι

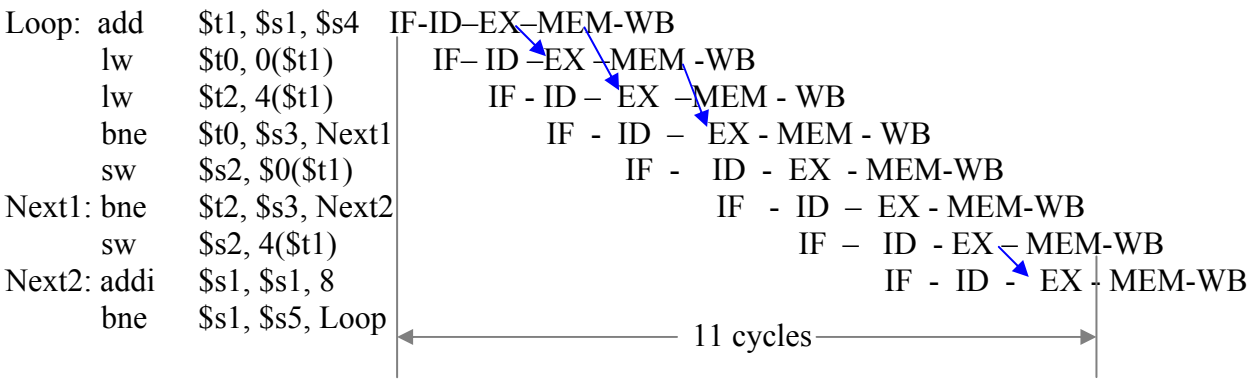
γ) Η αναδιάταξη των εντολών μπορεί να γίνει ως εξής:

```
Loop:   add  $t1, $s1, $s4
        lw   $t0, 0($t1)
        lw   $t2, 4($t1)
        bne  $t0, $s3, Next1
        sw   $s2, $0($t1)
Next1:  bne  $t2, $s3, Next2
        sw   $s2, 4($t1)
Next2:  addi $s1, $s1, 8
        bne  $s1, $s5, Loop
```

B2

```
        Exit:
```

Οπότε :



Ο παραπάνω βελτιστοποιημένος βρόχος θα εκτελεστεί 12/2 = 6 φορές. Κάθε ένας από τους βρόχους χρειάζεται για την εκτέλεσή του 11 κύκλους, εκτός από 3 περιπτώσεις (όπου δεν ισχύει η ισότητα `array[i]==k`), όπου θα χρειαστούν 3 επιπλέον κύκλοι: $6\times11 + 3 + 2 = 71$ κύκλοι

Η βελτιστοποίηση στη συνολική επίδοση είναι :

$$A\rightarrow B2 : \frac{173}{71} = 2{,}44$$

Θέμα 2ο (15%):

Έστω ότι επιθυμούμε να έχουμε τη δυνατότητα απευθείας προσπέλασης στη μνήμη από τις αριθμητικές εντολές, μέσω της εντολής:

```
        addm $t1, 100($t3)
δηλαδή: $t1 = $t1 + Mem[$t3+100]
```

Γιατί θα ήταν δύσκολο να προσθέσουμε την εντολή αυτή σε αρχιτεκτονική MIPS με χρήση αγωγού (pipelined datapath), όπως αυτή περιγράφεται στο κεφ. 6 των σημειώσεων; Τι συνέπειες θα είχαν οι αλλαγές που απαιτούνται για την υλοποίηση της νέας εντολής (σε throughput, latency και hazards);

Λύση 2ου Θέματος

Η εντολή `addm` χρειάζεται τα εξής 6 στάδια για την ολοκλήρωση των λειτουργιών της:

1) Ανάγνωση εντολής (IF)
2) Αποκωδικοποίηση εντολής και ανάγνωση των ορισμάτων της (ID)
3) Υπολογισμός της διεύθυνσης ανάγνωσης από τη μνήμη (EX)
4) Ανάγνωση από τη μνήμη (MEM)
5) Πρόσθεση του περιεχομένου της μνήμης με τον δεύτερο καταχωρητή (EX-2)
6) Αποθήκευση του αποτελέσματος στο register file (WB)

Από τα παραπάνω γίνεται φανερό ότι για την υλοποίηση της συγκεκριμένης εντολής θα χρειαστούμε ένα επιπλέον στάδιο στη σωλήνωση, και επομένως και όλες οι υπόλοιπες εντολές θα πρέπει αν περάσουν μέσα από το στάδιο αυτό. Επομένως, η προσθήκη μίας τέτοιας εντολής στο σύνολο των εντολών θα έχει ως αποτέλεσμα την αύξηση του latency (της αρχικής καθυστέρησης εξόδου της πρώτης εντολής από τη σωλήνωση), αλλά όχι του throughput (του ρυθμού εξόδου των

επόμενων εντολών από τη σωλήνωση), αφού οι επόμενες εντολές θα ολοκληρώνονται με ρυθμό μία ανά κύκλο ρολογιού, δεδομένου ότι δεν υπάρχουν stalls.

Στην πραγματικότητα όμως, στη μακρύτερη σωλήνωση υπάρχει μεγαλύτερη πιθανότητα να εμφανιστούν κίνδυνοι δεδομένων (hazards), και επιπλέον ένα hazard μπορεί να έχει μεγαλύτερο κόστος στην επίδοση (penalty). Επομένως, μπορεί τελικά να έχουμε μείωση του throughput με την προσθήκη της νέας εντολής.

Τέλος, το επιπλέον hardware που απαιτείται για την υλοποίηση του νέου σταδίου, για την ανίχνευση των hazards και για την πραγματοποίηση των αναγκαίων προωθήσεων δεδομένων, θα αυξήσει το κόστος της αρχιτεκτονικής.

Εναλλακτικά, θα μπορούσε το επιπλέον στάδιο που απαιτείται για την υλοποίηση της νέας εντολής να εκτελεστεί με τους υπάρχοντες πόρους του συστήματος, χωρίς την προσθήκη hardware. Κάτι τέτοιο θα υποχρέωνε την εντολή, ενώ ακολουθούσε τη συνηθισμένη ροή IF-ID-EX-MEM, να επιστρέψει από το στάδιο MEM πίσω στο στάδιο EX για την εκτέλεση της πρόσθεσης του περιεχομένου της μνήμης με τον δεύτερο καταχωρητή, στη συνέχεια να περάσει πάλι από το στάδιο MEM (χωρίς να κάνει τίποτα) και τέλος να εκτελέσει το WB.

Δηλαδή θα είχαμε την εξής ροή για τη συγκεκριμένη εντολή: IF-ID-EX-MEM-EX-MEM-WB.

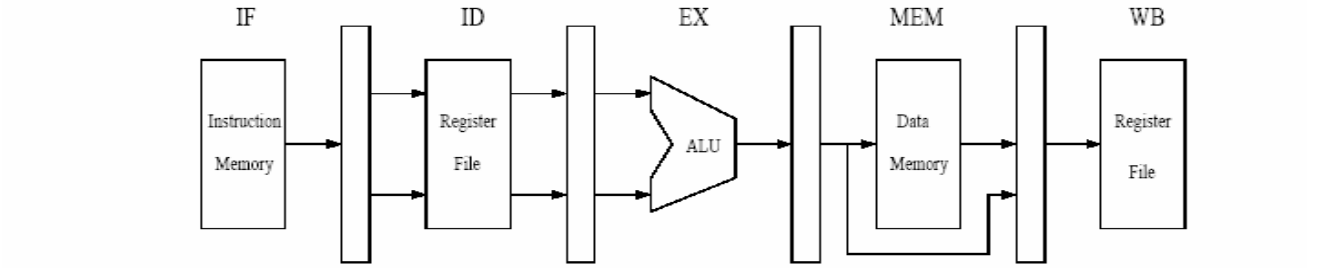
Άρα κάθε επόμενη εντολή δεν θα μπορούσε παρά να ευθυγραμμιστεί ως εξής:

IF-ID-EX-MEM-EX-MEM-WB
IF- ID- s - s - EX -MEM-WB

Η συγκεκριμένη υλοποίηση επιφέρει μείωση του throughput, και επιπρόσθετα hazards, τα οποία έχουν όλες τις συνέπειες που περιγράφτηκαν και για την προηγούμενη υλοποίηση.

Θέμα 3ο (25%):

Έστω η παρακάτω αρχιτεκτονική αγωγού (pipelined datapath) 5 σταδίων (IF-ID-EX-MEM-WB), όπως του επεξεργαστή MIPS του μαθήματος, όπου όμως το Register File **γράφεται στο δεύτερο μισό** του κύκλου ρολογιού και **διαβάζεται στο πρώτο μισό**. Ο εντολές μπορούν να καθυστερήσουν (stall) μέσα στην αρχιτεκτονική αγωγού (pipeline stalls) μόνο στα στάδια IF και ID.



α) για την παρακάτω ακολουθία εντολών, δείξτε την εκτέλεσή τους μέσα στο pipelined datapath, γεμίζοντας τον παρακάτω πίνακα. Όπου συμβαίνει καθυστέρηση (stall), βάλτε στο αντίστοιχο στάδιο τη λέξη bubble. (10%)

```
lw   $s2, 0($s1)    ; lw1
add  $s2,$s1,$s2    ; add1
add  $s3,$s1,$s3    ; add2
sw   $s2, 0($s4)    ; sw1
```

cycles	IF	ID	EX	MEM	WB
1	lw1				
2	add1	lw1			
3	add2	add1	lw1		
..					

β) προσπαθήστε να εξαφανίσετε καθυστερήσεις (stalls) που εμφανίζονται στην ακολουθία εντολών, προσθέτοντας σχήμα προώθησης (forwarding-bypass path), ώστε η παραπάνω αρχιτεκτονική αγωγού να συμπεριφέρεται όπως μια αρχιτεκτονική αγωγού όπου το register file γράφεται στο πρώτο μισό του κύκλου και διαβάζεται στο δεύτερο μισό. Δείξτε σε αναλυτικό διάγραμμα τα εναλλακτικά μονοπάτια δεδομένων που δημιουργεί το σχήμα προώθησης που επιλέξατε. Στον πίνακα του α) ερωτήματος, βάλτε το γράμμα B δίπλα σε κάθε bubble που εξαφανίζεται με το σχήμα προώθησης που επιλέξατε (Δηλ. bubble, B). Τέλος, γράψτε τις συνθήκες που πρέπει να ισχύουν για να γίνει το προτεινόμενο σχήμα προώθησης. (15%)

Λύση 3ου Θέματος

α) Παρατηρούμε ότι υπάρχουν δύο διαφορετικές εξαρτήσεις μεταξύ των εντολών:

i) η πρώτη βρίσκεται μεταξύ των `lw1` και `add1`. Δεδομένου (όπως φαίνεται από το σχήμα του datapath) ότι δεν υπάρχει κανενός είδους προώθησης, θα χρειαστεί να περιμένουμε να γράψει η `lw1` στο στάδιο WB τον καταχωρητή `$s2` και στον επόμενο κύκλο να διαβάζει η `add1`, η οποία θα πρέπει εν τω μεταξύ να παραμένει στο στάδιο ID, το περιεχόμενό του.

ii) η δεύτερη εξάρτηση βρίσκεται μεταξύ των `add1` και `sw1`. Ομοίως, θα περιμένουμε να γράψει η `add1` στο στάδιο WB τον καταχωρητή `$s2` και στον επόμενο κύκλο θα διαβάζει η `sw1`, η οποία θα παραμένει στο στάδιο ID, το περιεχόμενό του.

Η εκτέλεση της δοσμένης ακολουθίας εντολών θα γίνει ως εξής:

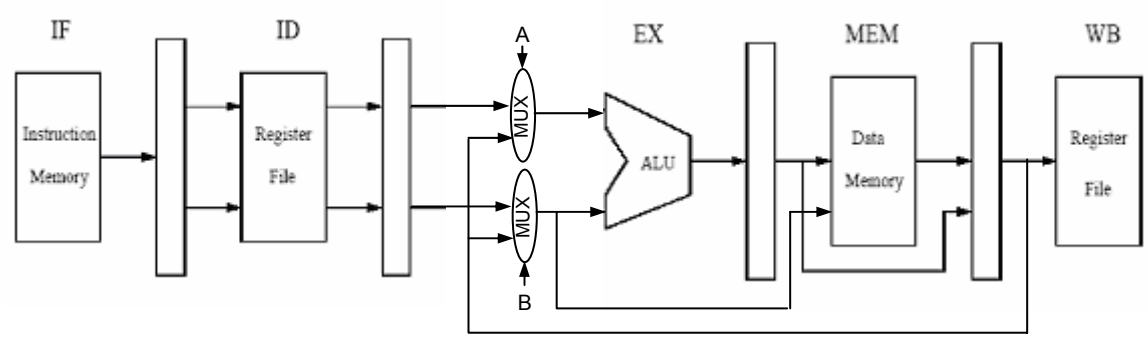
cycles	IF	ID	EX	MEM	WB
1	lw1				
2	add1	lw1			
3	add2	add1	lw1		
4	add2	add1	*bubble*	lw1	
5	add2	add1	*bubble*	*bubble*	lw1
6	add2	add1	*bubble*	*bubble*	*bubble*
7	sw1	add2	add1	*bubble*	*bubble*
8	-	sw1	add2	add1	*bubble*
9	-	sw1	*bubble*	add2	add1
10	-	sw1	*bubble*	*bubble*	add2
11	-	-	sw1	*bubble*	*bubble*
12	-	-	-	sw1	*bubble*
13	-	-	-	-	sw1

β) Οι εμφανιζόμενες καθυστερήσεις μπορούν να εξαφανιστούν χρησιμοποιώντας ένα κατάλληλο σχήμα προώθησης, εκτός από μία, εκείνη που παρεμβάλλεται μεταξύ των `lw1` και `add1` (όπως συμβαίνει εξάλλου και στη συνηθισμένη αρχιτεκτονική σωλήνωσης του MIPS). Δηλαδή, και στις δύο περιπτώσεις εξαρτήσεων, προωθούμε το προς εγγραφή (στον `$s2`) αποτέλεσμα, από την έξοδο του σταδίου MEM της πρώτης εντολής, στην είσοδο του σταδίου EX της δεύτερης-εξαρτημένης εντολής. Επομένως, ο παραπάνω πίνακας μπορεί να διορθωθεί ως εξής:

cycles	IF	ID	EX	MEM	WB
1	lw1				
2	add1	lw1			
3	add2	add1	lw1		
4	add2	add1	*bubble*	lw1	

5	add2	add1	*bubble*, B	*bubble*	lw1
6	add2	add1	*bubble*, B	*bubble*, B	*bubble*
7	sw1	add2	add1	*bubble*, B	*bubble*, B
8	-	sw1	add2	add1	*bubble*, B
9	-	sw1	*bubble*, B	add2	add1
10	-	sw1	*bubble*, B	*bubble*, B	add2
11	-	-	sw1	*bubble*, B	*bubble*, B
12	-	-	-	sw1	*bubble*, B
13	-	-	-	-	sw1

Οι αναγκαίες προσθήκες μονοπατιών και δομικών στοιχείων για την προώθηση δεδομένων φαίνονται στο σχήμα που ακολουθεί:



Για τον έλεγχο των πολυπλεκτών, χρησιμοποιούνται οι εξής συνθήκες:
για την προώθηση από `add1` σε `sw1`

If (MEM/WB.RegWrite) and
(MEM/WB.RegRd ≠ 0) and
(EX/MEM.RegRd ≠ ID/EX.RegRt) and
(MEM/WB.RegRd = ID/EX.RegRt) and
then forwardB = 1

και για την προώθηση από `lw1` σε `add1`

If (MEM/WB.RegWrite) and
(MEM/WB.RegRt ≠ 0) and
(EX/MEM.RegRt ≠ ID/EX.RegRt) and
(MEM/WB.RegRt = ID/EX.RegRt) and
then forwardB = 1

Θέμα 4ο (30%):

Δίνεται μια σειρά αναφορών σε διευθύνσεις λέξεων στη μνήμη ενός υπολογιστή: 5, 9, 12, 28, 32, 25, 6, 8, 11, 28, 31, 24, 5, 25, 4, 12. Υποθέτουμε ότι έχουμε κρυφή μνήμη με οργάνωση:

i) απευθείας απεικόνισης (direct mapped) με 16 blocks, όπου κάθε block έχει μέγεθος μια λέξη (word),

ii) απευθείας απεικόνισης (direct mapped), με 16 λέξεις (words) συνολικό μέγεθος cache, όπου κάθε block έχει μέγεθος τέσσερις (4) λέξεις.

iii) συνόλου συσχέτισης 2-δρόμων (2-way set associative) με συνολικό μέγεθος 16 λέξεις (words) όπου κάθε block έχει μέγεθος δύο (2) λέξεις (Υποθέστε LRU αλγόριθμο αντικατάστασης).

Δείξτε για τις παραπάνω περιπτώσεις οργάνωσης της κρυφής μνήμης, για κάθε αναφορά, αν είναι επιτυχής (hit) ή όχι (miss) καθώς και την τελικά περιεχόμενα της κρυφής μνήμης.

Λύση 4ου Θέματος

i) Διαθέτουμε μνήμη με 16 blocks, όπως φαίνεται στο σχήμα που ακολουθεί:

0	**32**
1	
2	
3	
4	**4**
5	**5**
6	**6**
7	
8	~~8~~ **24**
9	~~9~~ **25**
10	
11	**11**
12	~~12~~ ~~28~~ **12**
13	
14	
15	**31**

5	miss
9	miss
12	miss
28	miss
32	miss
25	miss
6	miss
8	miss
11	miss
28	hit
31	miss
24	miss
5	hit
25	hit
4	miss
12	miss

ii) Η μνήμη διαθέτει 16 words. Αφού περιέχει 4 words / block, περιέχει $\frac{16}{4} = 4$ blocks

0	**(32-33-34-35)**
1	**(4-5-6-7)**
2	~~(8-9-10-11) (24-25-26-27) (8-9-10-11)~~ **(24-25-26-27)**
3	~~(12-13-14-15) (28-29-30-31)~~ **(12-13-14-15)**

5	miss
9	miss
12	miss
28	miss
32	miss
25	miss
6	hit
8	miss
11	hit
28	hit
31	hit
24	miss
5	hit
25	hit
4	hit
12	miss

iii) Η μνήμη διαθέτει 168 words. Αφού είναι 2-way set associative με 2 words / block, περιέχει $\frac{16}{2 \cdot 2} = 4$ sets

0	(8-9)	(32-33)	(24-25)	**(8-9)**
	(32-33)	(24-25)	(8-9)	**(24-25)**
1	**(10-11)**			
2	(4-5)	(12-13)	(28-29)	**(4-5)**
	(12-13)	(28-29)	(4-5)	**(12-13)**
3	**(6-7)**			
	(30-31)			

5	miss
9	miss
12	miss
28	miss
32	miss
25	miss
6	miss
8	miss
11	miss
28	hit
31	miss
24	hit
5	miss
25	hit
4	hit
12	miss