ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

8 Ιουλίου 2005

# ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΟΡΓΑΝΩΣΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Εξετάσεις Ιουλίου 2005

| | |
|---|---|
| Επώνυμο: | |
| Όνομα: | |
| Εξάμηνο: | |

| Θέμα | Βαθμός |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |

Μην ξεχάσετε να γράψετε το ονοματεπώνυμο σας σε όλες τις κόλλες. Η εξέταση γίνεται με ανοικτά βιβλία και σημειώσεις, χωρίς χρήση Η/Υ ή PDAs. Διάρκεια εξέτασης 2 ½ ώρες. Το διαγώνισμα βαθμολογείται με άριστα 10 μονάδες. Στον τελικό βαθμό προαγωγής του μαθήματος προστίθεται 1 μονάδα (bonus) αν έχετε παραδώσει μία ικανοποιητική λύση στις δύο σειρές που μοιράσθηκαν στο μάθημα (0,5 μονάδες για κάθε σειρά ασκήσεων).

## Θέμα 1ο (30%):

Οι *κρυφές μνήμες χωρίς κλειδώματα* (lockup-free caches) και η *πρώιμη ανάκληση δεδομένων* στο υλικό (hardware prefetching) σχεδιάστηκαν για να επιταχύνουν το ρυθμό εκτέλεσης των εντολών στον επεξεργαστή, επικαλύπτοντας το χειρισμό των αστοχιών κρυφής μνήμης (cache misses) με την επεξεργασία άλλων εντολών ή άλλων αστοχιών κρυφής μνήμης. Στο πρόβλημα αυτό υποθέστε ότι η ιεραρχία μνήμης διαθέτει κρυφή μνήμη ενός επιπέδου, με μέγεθος cache line 32 bytes και διάδρομο επικοινωνίας κύριας μνήμης με την κρυφή μνήμη (memory bus) 8 bytes. Υποθέστε, επίσης, ότι ο χρόνος εύρεσης των ζητούμενων δεδομένων στη μνήμη (miss latency) είναι $t_{miss}$ = 2 cycles, και ο χρόνος μεταφοράς δεδομένων στη κρυφή μνήμη (transfer latency) είναι 1 cycle / 8 bytes, δηλαδή για την μεταφορά ολόκληρης της cache line: $t_{transfer}$ = 4 cycles. Τέλος, δίνεται ότι στο σύστημα 1word = 4 bytes.

Εξετάζουμε την εκτέλεση του εξής κώδικα:

```
for (i = 0; i < 1000; i+=4) {
        a += A[i] + B[i];
```

Θεωρούμε ότι οι τιμές των μεταβλητών `4·i`, `a` βρίσκονται στους καταχωρητές `$s4`, `$t0` και οι διευθύνσεις των πινάκων `A` και `B` στους `$s1` και `$s2`. Το περιεχόμενο του καταχωρητή `$s4` είναι αρχικά 0. Ο αντίστοιχος κώδικας σε assembly είναι ο εξής:

```
Loop:       add  $t1, $s1, $s4
            add  $t2, $s2, $s4
            lw   $t3, 0($t1)
```

```
            lw   $t4, 0($t2)
            add  $t0, $t0, $t3
            add  $t0, $t0, $t4
            addi $s4, $s4, 16
            bne  $s4, $s5, Loop
Exit:
```

Στον πίνακα που ακολουθεί φαίνεται η ροή εκτέλεσης του παραπάνω βρόχου, υποθέτοντας ότι έχουμε μοντέλο υλοποίησης single cycle και ότι κανένα από τα στοιχεία των πινάκων A και B δεν βρίσκονται ήδη στην κρυφή μνήμη (συμβολίζουμε με m = miss latency, t = transfer latency). Θεωρούμε ότι ο σχεδιασμός της κρυφή μνήμης επιτρέπει την προώθηση στον επεξεργαστή της ζητούμενης λέξης αμέσως μόλις αφιχθεί στην κρυφή μνήμη, χωρίς να χρειάζεται να ολοκληρωθεί η μεταφορά ολόκληρης της cache line. Επιπλέον, υποθέτουμε ότι μετά από μία αστοχία, όλες οι επόμενες εντολές πρέπει να περιμένουν έως η ζητούμενη λέξη φτάσει στον επεξεργαστή. Τα στοιχεία των πινάκων είναι ευθυγραμμισμένα στην κρυφή μνήμη, δηλαδή το στοιχείο A[0] καταλαμβάνει την πρώτη θέση μέσα σε κάποια cache line, ομοίως και το στοιχείο B[0].

Τέλος, ο μηχανισμός πρώιμης ανάκλησης δεδομένων, καταγράφει τις ακολουθίες προσπελάσεων στη μνήμη (αστοχίες της κρυφής μνήμης). Αν παρατηρηθούν τρεις συνεχόμενες προσπελάσεις ίδιου βήματος, τότε ενεργοποιείται η πρώιμη ανάκληση των επόμενων θέσεων μνήμης ίδιου βήματος (π.χ. οι αστοχίες δεδομένων που αναφέρονται στις θέσεις μνήμης a+0, a+32, a+64, ενεργοποιούν την πρώιμη ανάκληση των θέσεων μνήμης a+96, a+128, κλπ). Ο μηχανισμός πρώιμης ανάκλησης καλεί στην κρυφή μνήμη ένα δεδομένο (a+96) στον κύκλο που ακολουθεί αμέσως μετά τον πρώτο κύκλο ανάγνωσης από τη μνήμη της τρίτης αστοχίας (a+64). Επιπλέον, προκειμένου να μην κατακλυστεί η κρυφή μνήμη με δεδομένα που δεν χρειάζονται στον επεξεργαστή, κάθε επόμενη πρώιμη ανάκληση (a+128) περιμένει έως ότου κληθεί προς επεξεργασία το δεδομένο που έχει αποθηκευθεί στην κρυφή μνήμη από την προηγούμενη πρώιμη ανάκληση (a+96). Σημειώστε ότι επιτρέπεται η επικάλυψη κύκλων ανάγνωσης από την μνήμη (κύκλων m), αλλά ο διάδρομο επικοινωνίας της κύριας μνήμης με την κρυφή μνήμη (memory bus) μπορεί σε κάθε κύκλο να μεταφέρει δεδομένα μίας συγκεκριμένης cache line.

| | | | | | | | | | | | 1 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cycles | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| array A | | | | m | m | t | t | t | t | | | | | | | | | | | |
| array B | | | | | | | | m | m | t | t | t | t | | | | | | | |
| Prefetch A | | | | | | | | | | | | | | | | | | | | |
| Prefetch B | | | | | | | | | | | | | | | | | | | | |
| Execute | add | add | lwA | | | | lwB | | | | add | add | addi | bne | add | add | lwA | lwB | add | ⋮ |
| | 2 | | | | | | | | | | 3 | | | | | | | | | |
| cycles | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| array A | | | | | | | | | | | | | | | | | | | | |
| array B | | | | | | | | | | | | | | | | | | | | |
| Prefetch A | | | | | | | | | | | | | | | | | | | | |
| Prefetch B | | | | | | | | | | | | | | | | | | | | |
| Execute | | | | | | | | | | | | | | | | | | | | |

| cycles | 4 | | | | | | | | | | 5 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| array A | | | | | | | | | | | | | | | | | | | | |
| array B | | | | | | | | | | | | | | | | | | | | |
| Prefetch A | | | | | | | | | | | | | | | | | | | | |
| Prefetch B | | | | | | | | | | | | | | | | | | | | |
| Execute | | | | | | | | | | | | | | | | | | | | |

| cycles | 6 | | | | | | | | | | 7 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| array A | | | | | | | | | | | | | | | | | | | | |
| array B | | | | | | | | | | | | | | | | | | | | |
| Prefetch A | | | | | | | | | | | | | | | | | | | | |
| Prefetch B | | | | | | | | | | | | | | | | | | | | |
| Execute | | | | | | | | | | | | | | | | | | | | |

| cycles | 8 | | | | | | | | | | 9 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| array A | | | | | | | | | | | | | | | | | | | | |
| array B | | | | | | | | | | | | | | | | | | | | |
| Prefetch A | | | | | | | | | | | | | | | | | | | | |
| Prefetch B | | | | | | | | | | | | | | | | | | | | |
| Execute | | | | | | | | | | | | | | | | | | | | |

Συμπληρώστε το διάγραμμα χρονισμού του παραπάνω πίνακα για τους πρώτους 100 κύκλους εκτέλεσης. Πόσοι κύκλοι απαιτούνται για την εκτέλεση των 250 επαναλήψεων του `for loop`;

## Θέμα 2ο (35%):

Έστω ότι οι εντολές αναφοράς στη μνήμη (`load, store`) δεν επιτρέπεται να περιέχουν σταθερό όρισμα (offset). Δηλαδή έχουν τη μορφή, π.χ:

```
lw   $s1, ($s2)
```

Για τη λειτουργία: `$s1 = Mem[$s2];`

(i) Πώς επηρεάζει η συγκεκριμένη τροποποίηση του συνόλου εντολών (ISA) την υλοποίηση της αρχιτεκτονικής αγωγού; Θα μπορούσε να μειωθεί το βάθος της σωλήνωσης και πώς; Τι συνέπειες θα είχαν οι όποιες αλλαγές σε throughput και latency;

(ii) Καταγράψτε τις περιπτώσεις όπου ενδέχεται να εμφανισθούν κίνδυνοι δεδομένων καθώς και τις αντίστοιχες αναγκαίες συνθήκες προώθησης δεδομένων (forwarding); Υπενθύμιση: εξαρτήσεις δεδομένων μεταξύ εντολών αριθμητικών/λογικών με αριθμητικές/λογικές καθώς και μεταξύ lw/sw και αριθμητικών/λογικών. Σε ότι αφορά τη δεύτερη περίπτωση εξαρτήσεων, υπάρχει κάποια βελτίωση σε σχέση με την κλασσική σωλήνωση 5 σταδίων του MIPS;

## Θέμα 3ο (35%):

Θεωρούμε της εξής ακολουθία εντολών. Αρχικά, ισχύει: `$t4` = 16 + `$r1`;

```
loop:     lw   $t1, 0($r1)    #1
          lw   $t2, 0($r2)    #2
          lw   $t3, 0($r3)    #3
          add  $t1, $t1, $t2  #4
          add  $t1, $t1, $t3  #5
          sll  $t1, $t1, 2    #6
          sw   $t1, 0($r4)    #7
          addi $r1, $r1, 4    #8
          addi $r2, $r2, 4    #9
          addi $r3, $r3, 4    #10
          bne  $r1, $t4, loop #11
exit:
```

i) Υποθέτουμε ότι διαθέτουμε έναν επεξεργαστή με αρχιτεκτονική σωλήνωσης 5 σταδίων, όπου υπάρχουν όλα τα δυνατά σχήματα προώθησης. Δείξτε το διάγραμμα εκτέλεσης των επιμέρους φάσεων για κάθε εντολή (κατά την πρώτη σειρά επαναλήψεων), στην περίπτωση που οι εντολές εκτελούνται με τη σειρά που υπαγορεύει ο κώδικας της ρουτίνας. Σημειώστε όλες τις αναγκαίες προωθήσεις δεδομένων. Πόσους κύκλους χρειάζεται για να ολοκληρωθεί η εκτέλεση της ρουτίνας;

ii) Ο παραλληλισμός επιπέδου εντολής (instruction level parallelism - ILP) αναφέρεται στην έλλειψη εξαρτήσεων μεταξύ των εντολών ενός σειριακού προγράμματος και την εκμετάλλευση του γεγονότος για την παράλληλη εκτέλεσή τους, διατηρώντας την ορθότητα του τελικού αποτελέσματος. Για το σκοπό αυτό, στο εξής (για τα ερωτήματα α και β) υποθέτουμε ότι διαθέτουμε έναν επεξεργαστή με αρχιτεκτονική σωλήνωσης 5 σταδίων, πλάτους 2 σωληνώσεων. Επιπλέον, υπάρχουν όλα τα δυνατά σχήματα προώθησης.

α) Δείξτε το διάγραμμα εκτέλεσης των επιμέρους φάσεων για κάθε εντολή (κατά την πρώτη σειρά επαναλήψεων), στην περίπτωση που οι εντολές εκτελούνται με τη σειρά (in-order: αν οι διαδοχικές εντολές έχουν μεταξύ τους κάποιο είδος εξάρτησης δεδομένων, εισάγεται στη σωλήνωση μία μόνο εντολή/ κύκλο ρολογιού). Σημειώστε όλες τις αναγκαίες προωθήσεις δεδομένων. Πόσους κύκλους χρειάζεται για να ολοκληρωθεί η εκτέλεση της ρουτίνας;

β) Αν οι εντολές μπορούν να εκτελεστούν εκτός σειράς (out-of-order: αν οι διαδοχικές εντολές έχουν μεταξύ τους κάποιο είδος εξάρτησης δεδομένων, αναζητούνται ανεξάρτητες εντολές μεταξύ των επόμενων εντολών με στόχο να εισάγονται στη σωλήνωση 2 εντολές/ κύκλο ρολογιού), πόσους κύκλους κερδίζουμε; Κατασκευάστε το αντίστοιχο διάγραμμα εκτέλεσης και σημειώστε όλες τις αναγκαίες προωθήσεις δεδομένων.