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

19 Φεβρουαρίου 2003

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΟΡΓΑΝΩΣΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Εκφωνήσεις και Λύσεις Θεμάτων Φεβρουαρίου 2003

Θέμα 1ο (20%):

Έστω το τμήμα του C κώδικα:

```
while (save[i] == k)
     i = i + j;
```

Θεωρούμε ότι οι τιμές των μεταβλητών `i, j, k` βρίσκονται στους καταχωρητές `$s3, $s4, $s5` και η διεύθυνση του πίνακα `save` στον `$s6` Ο αντίστοιχος κώδικας σε assembly είναι ο εξής:

```
Loop:     add  $t1, $s3, $s3
          add  $t1, $t1, $t1
          add  $t1, $t1, $s6
          lw   $t0, 0($t1)
          bne  $t0, $s5, Exit
          add  $s3, $s3, $s4
          j    Loop
Exit:
```

Κάθε φορά που διατρέχει το loop χρησιμοποιούνται 2 εντολές άλματος (μία υπό συνθήκη και μία χωρίς συνθήκη). Ωστόσο μόνο μη βελτιστοποιημένοι compilers θα παρήγαγαν κώδικα με τέτοιο κόστος. Ξαναγράψτε το τμήμα αυτό του κώδικα έτσι ώστε να εκτελείται το πολύ μία εντολή άλματος (χωρίς ή με συνθήκη) εντός του βρόχου. Πόσες εντολές εκτελούνται πριν και μετά τη βελτιστοποίηση αν ο αριθμός των επαναλήψεων του βρόχου είναι 10. (Θεωρείστε ότι `save[i+10*j] ≠ k` και `save[i], … , save[i+9*j] = k`).

Λύση

Στον αρχικό κώδικα ο αριθμός των εντολών που εκτελούνται είναι:

(αρ. επαναλήψεων) × (αρ. εντολών όταν array[i]=k)

+

(αρ. εντολών κατά την τελευταία εκτέλεση όπου array[i]≠k)

=

$10 \times 7 + 5 = 75$

Ο βελτιστοποιημένος κώδικας είναι:

```
Loop:     add  $t1, $s3, $s3
          add  $t1, $t1, $t1
          add  $t1, $t1, $s6
          lw   $t0, 0($t1)
          add  $s3, $s3, $s4
          beq  $t0, $s5, Loop
          sub  $s3, $s3, $s4
Exit:
```

Στο βελτιστοποιημένο κώδικα ο αριθμός των εντολών που εκτελούνται είναι:

(αρ. επαναλήψεων) × (αρ. εντολών όταν array[i] = k)
+
(αρ. εντολών κατά την τελευταία εκτέλεση όπου array[i] ≠ k)
=
10 × 6 + 7 = 67

2η λύση :

```
            add  $t1, $s3, $s3
            add  $t1, $t1, $t1
            add  $t1, $t1, $s6
            lw   $t0, 0($t1)
            bne  $t0, $s5, Exit
Loop:       add  $s3, $s3, $s4
            add  $t1, $s3, $s3
            add  $t1, $t1, $t1
            add  $t1, $t1, $s6
            lw   $t0, 0($t1)
            beq  $t0, $s5, Loop
Exit:
```

Στο βελτιστοποιημένο κώδικα ο αριθμός των εντολών που εκτελούνται είναι:

(πρώτη εκτέλεση)
+
(αρ.υπολ.επαναλήψεων) × (αρ.εντολών όταν array[i] = k)
+
(αρ.εντολών κατά την τελευταία εκτέλεση όπου array[i] ≠ k)
=
5 + 9 × 6 + 6 = 65

Θέμα 2ο (35%):

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 (π.χ. 2 διαδοχικές επαναλήψεις, οπότε ο unrolled βρόχος έχει πλέον βήμα 2).

```
for (i=0;i<12;i++) a(i) = a(i) + 8;
```

γίνεται:

```
for (i=0;i<12;i+=2) {
                a(i) = a(i) + 8;
                a(i+1) = a(i+1) + 8;
                }
```

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).

Ο αρχικός μας κώδικας σε MIPS assembly είναι ο ακόλουθος (ο `$s1` έχει αρχικά την τιμή 48):

```
Loop:   lw   $t0, 0($s1)
        add  $t0, $t0, $s2
        sw   $t0, 0($s1)
        addi $s1, $s1, -4
        bne  $s1, $zero, Loop
```

A

α) Δείξτε το διάγραμμα εκτέλεσης των επιμέρους φάσεων για κάθε εντολή, μέσα στην αρχιτεκτονική αγωγού, υποθέτοντας την ύπαρξη σχήματος προώθησης (διάγραμμα εκτέλεσης όπως στις διαφάνειες του μαθήματος: IF - ID -…κλπ). Μπορείτε να το βελτιστοποιήσετε (αποφυγή τυχόν stalls)?

β) Παρακάτω είναι το τμήμα του αρχικού κώδικα, όπου ο βρόχος είναι ξεδιπλωμένος μία φορά:

```
Loop:   lw   $t0, 0($s1)
        add  $t0, $t0, $s2
        sw   $t0, 0($s1)
        lw   $t1, -4($s1)
        add  $t1, $t1, $s2
        sw   $t1, -4($s1)
        addi $s1, $s1, -8
        bne  $s1, $zero, Loop
```

B

Σχεδιάστε ξανά τον κώδικα ώστε να αποφύγετε τα υποχρεωτικά memory stalls. Λαμβάνοντας υπόψη τα αναγκαία memory 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

α) Χωρίς βελτιστοποίηση η εκτέλεση θα γίνει ως εξής:

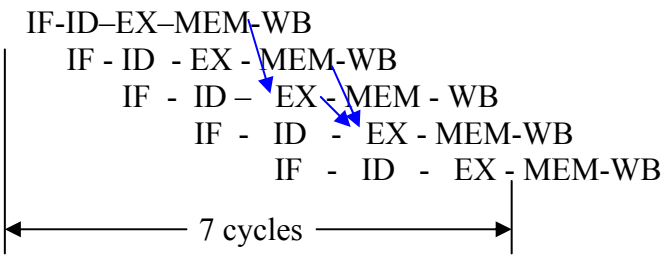
```
Loop: lw   $t0, 0($s1)      IF-ID–EX–MEM-WB
      add  $t0, $t0, $s2      IF – s - ID – EX – MEM - WB
      sw   $t0, 0($s1)              IF - ID – EX – MEM-WB
      addi $s1, $s1, -4                IF – ID – EX – MEM-WB
      bne  $s1, $zero, Loop                 IF - ID - EX – MEM-WB
                                  <-------- 8 cycles -------->
```

Για να αποφύγουμε το αναγκαίο memory stall αναδιατάσσουμε τη σειρά εκτέλεσης των εντολών:

```
Loop:   lw   $t0, 0($s1)
        addi $s1, $s1, -4
        add  $t0, $t0, $s2
        sw   $t0, 4($s1)
        bne  $s1, $zero, Loop
```

Αναλυτικότερα :

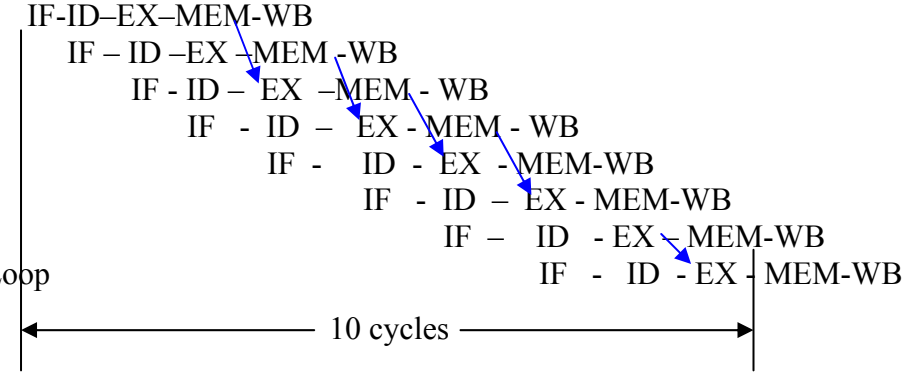
```
Loop:  lw    $t0, 0($s1)
       addi  $s1, $s1, -4
       add   $t0, $t0, $s2
       sw    $t0, 0($s1)
       bne   $s1, $zero, Loop
```



β)
```
Loop:   lw   $t0, 0($s1)
        lw   $t1, -4($s1)
        add  $t0, $t0, $s2
        add  $t1, $t1, $s2
        sw   $t0, 0($s1)
        sw   $t1, -4($s1)
        addi $s1, $s1, -8
        bne  $s1, $zero, Loop
```

Οπότε :

```
Loop:  lw    $t0, 0($s1)
       lw    $t1, -4($s1)
       add   $t0, $t0, $s2
       add   $t1, $t1, $s2
       sw    $t0, 0($s1)
       sw    $t1, -4($s1)
       addi  $s1, $s1, -8
       bne   $s1, $zero, Loop
```



Σε κάθε περίπτωση, ο αριθμός των κύκλων που εκτελούνται μέχρι την έξοδο από το loop είναι:
(αρ.επαναλήψεων)×(αρ.κύκλων βρόχου)+2 κύκλοι (το τελείωμα του τελευταίου bne)

Στον μη βελτιστοποιημένο αρχικό κώδικα :
12×8+2=98cycles (A1)

Στο βελτιστοποιημένο αρχικό κώδικα :
12×7+2=86cycles (A2)

Στον τελικό βελτιστοποιημένο κώδικα :
6×10+2=62cycles (B)

Επομένως η βελτιστοποίηση είναι :

$$A1 \rightarrow B : \frac{98}{62} = 1{,}58 \qquad \text{ή} \qquad A2 \rightarrow B : \frac{86}{62} = 1{,}38$$

Θέμα 3ο (25%):

Το ακόλουθο πρόγραμμα εκτελείται σε μηχάνημα με κρυφή μνήμη οργανωμένη σε blocks με 4 λέξεις (words, με μήκος word = 4 bytes) χωρητικότητας 256 bytes δεδομένων:

```
int i, j, c, stride, array[256];
…
for (i=0; i<10000; i++)
   for (j=0; j<256; j+=stride)
      array[j] = c + 5;
```

Εξετάζουμε τη δραστηριότητα της κρυφής μνήμης που αφορά μόνο τις αναφορές στα στοιχεία του πίνακα `array`. Αν θεωρήσετε επιπλέον ότι κάθε ακέραιος καταλαμβάνει χώρο ίσο με 1 λέξη, ποιος είναι ο αναμενόμενος αριθμός αναφορών στη κρυφή μνήμη που είναι επιτυχείς (hits), όταν `stride=33`; Τι συμβαίνει όταν `stride=30`; Ποιες αλλαγές θα προκύψουν αν η κρυφή μνήμη είναι συσχέτισης 2-δρόμων (2-way set associative); Θεωρούμε ότι το στοιχείο `array[0]` αποθηκεύεται στο block 0 της κρυφής μνήμης.

Λύση

Θεωρούμε ότι η μνήμη μας είναι write allocate και ότι αρχικά είναι άδεια.

Α) Διαθέτουμε 256 bytes μνήμη δεδομένων, δηλαδή μνήμη με $\frac{256}{4} = 64$ words.

Επομένως για οργάνωση direct mapped, με 4 λέξεις ανά block : $\frac{64}{4} = 16$ blocks

i) Για stride = 33 γράφονται οι εξής διευθύνσεις μνήμης : 0, 33, 66, 99, 132, 165, 198, 231
Αρχικά η κρυφή μνήμη είναι άδεια. Επομένως στην πρώτη επανάληψη όλες οι αναφορές στη μνήμη είναι miss (δηλαδή 8 misses).

0	~~(0-1-2-3)~~ **(64-65-66-67)**
1	~~(132-133-134-135)~~ **(196-197-198-199)**
2	
3	
4	
5	
6	
7	
8	~~(32-33-34-35)~~ **(96-97-98-99)**
9	~~(164-165-166-167)~~ **(228-229-230-231)**
10	
11	
12	
13	
14	
15	

Μετά το τέλος της πρώτης επανάληψης η κρυφή μνήμη θα έχει την παραπάνω μορφή. Επομένως και σε όλες τις επόμενες επαναλήψεις θα έχουμε από 8 misses
Συνολικά:

$8 \times 10.000 = 80.000$ misses

ii) Για stride = 30 γράφονται οι εξής διευθύνσεις μνήμης : 0, 30, 60, 90, 120, 150, 180, 210, 240
Αρχικά η κρυφή μνήμη είναι άδεια. Επομένως στην πρώτη επανάληψη όλες οι αναφορές στη μνήμη είναι miss (δηλαδή 9 misses).

0	(0-1-2-3)
1	
2	
3	
4	(208-209-210-211)
5	(148-149-150-151)
6	(88-89-90-91)
7	(28-29-30-31)
8	
9	
10	
11	
12	(240-241-242-243)
13	(180-181-182-183)
14	(120-121-122-123)
15	(60-61-62-63)

Μετά το τέλος της πρώτης επανάληψης η κρυφή μνήμη θα έχει την παραπάνω μορφή. Επομένως και σε όλες τις επόμενες επαναλήψεις θα έχουμε από 9 hits
Συνολικά :

$9 \times 9.999 = 89.991$ hits

και 9 misses

Β) Για οργάνωση 2-way set associative, με 4 λέξεις ανά block: $\frac{64}{4} = 16$ blocks

και $\frac{16}{2} = 8$ sets

i) Για stride = 33 γράφονται οι εξής διευθύνσεις μνήμης : 0, 33, 66, 99, 132, 165, 198, 231

0	~~(0-1-2-3)~~ **(64-65-66-67)**
	~~(32-33-34-35)~~ **(96-97-98-99)**
1	~~(132-133-134-135)~~ **(196-197-198-199)**
	~~(164-165-166-167)~~ **(228-229-230-231)**
2	
…	… … …
…	… … …
7	

Μετά το τέλος της πρώτης επανάληψης η κρυφή μνήμη θα έχει την παραπάνω μορφή. Επομένως και σε όλες τις επόμενες επαναλήψεις θα έχουμε από 8 misses

Συνολικά και πάλι:

$8 \times 10.000 = 80.000$ misses

ii) Για stride = 30 γράφονται οι εξής διευθύνσεις μνήμης: 0, 30, 60, 90, 120, 150, 180, 210, 240

0	(0-1-2-3)
1	
2	
3	
4	(208-209-210-211)
	(240-241-242-243)
5	(148-149-150-151)
	(180-181-182-183)
6	(88-89-90-91)
	(120-121-122-123)
7	(28-29-30-31)
	(60-61-62-63)

Μετά το τέλος της πρώτης επανάληψης η κρυφή μνήμη θα έχει την παραπάνω μορφή. Επομένως και σε όλες τις επόμενες επαναλήψεις (εκτός από την πρώτη) θα έχουμε από 9 hits. Συνολικά και πάλι:

$9 \times 9.999 = 89.991$ hits και 9 misses

Θέμα 4ο (20%):

Δίνεται μια σειρά αναφορών σε διευθύνσεις λέξεων στη μνήμη ενός υπολογιστή: 2, 7, 11, 5, 25, 32, 15, 49, 11, 10, 12, 2, 13, 7, 48, 3. Υποθέτουμε ότι έχουμε κρυφή μνήμη με οργάνωση:

i) απευθείας απεικόνισης (direct mapped) με 8 blocks, όπου κάθε block έχει μέγεθος μια λέξη (word),

ii) απευθείας απεικόνισης (direct mapped), με 8 λέξεις (words) συνολικό μέγεθος cache, όπου κάθε block έχει μέγεθος δύο (2) λέξεις.

iii) συνόλου συσχέτισης 2-δρόμων (2-way set associative) με συνολικό μέγεθος 8 λέξεις (words) όπου κάθε block έχει μέγεθος μια λέξη (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

i) Διαθέτουμε μνήμη με 8 blocks:

0	32 **48**
1	25 **49**
2	2 10 **2**
3	11 **3**
4	**12**
5	5 **13**
6	
7	7 15 **7**

2	miss
7	miss
11	miss
5	miss
25	miss
32	miss
15	miss
49	miss
11	hit
10	miss
12	miss
2	miss
13	miss
7	miss
48	miss
3	miss

ii) Η μνήμη διαθέτει 8 words. Αφού περιέχει 2 words / block, περιέχει $\frac{8}{2} = 4$ blocks

0	(24-25) (32-33) **(48-49)**
1	(2-3) (10-11) **(2-3)**
2	(4-5) **(12-13)**
3	(6-7) (14-15) **(6-7)**

2	miss
7	miss
11	miss
5	miss
25	miss
32	miss
15	miss
49	miss
11	hit
10	hit
12	miss
2	miss
13	hit
7	miss
48	hit
3	hit

iii) Η μνήμη διαθέτει 8 words. Αφού είναι 2-way set associative, περιέχει $\frac{8}{2} = 4$ sets

0	32				**12**
	12				**48**
1	5		25		**49**
	25		49		**13**
2	2				**10**
	10				**2**
3	7	11	15	11	**7**
	11	15	11	7	**3**

2	miss
7	miss
11	miss
5	miss
25	miss
32	miss
15	miss
49	miss
11	hit
10	miss
12	miss
2	hit
13	miss
7	miss
48	miss
3	miss