HY425

Θεωρητική Άσκηση 2

Ανάθεση: 5/11/2008

Προθεσμία: 12/11/2008

Οδηγίες: Απαντήστε στα ερωτήματα χρησιμοποιώντας κειμενογράφο της αρεσκείας σας και υποβάλετε τις απαντήσεις σε ένα αρχείο μορφής .pdf, με e-mail στο βοηθό του μαθήματος (kavadias@ics.forth.gr). Χρησιμοποιήστε την επικεφαλίδα: `HY425: HW2` στο μήνυμά σας.

Άσκηση 1 (50%)

Δίδεται ο παρακάτω κώδικας, για τον οποίο υποθέτουμε ότι όλες οι τιμές των μεταβλητών έχουν ήδη οριστεί πριν τη χρήση τους, όλες οι τιμές μεταβλητών έχουν φορτωθεί από τη μνήμη σε καταχωρητές και μόνο οι μεταβλητές a, b και c χρησιμοποιούνται μετά το τέλος του κώδικα. Θεωρήστε ότι δεν υπάρχουν εξαιρέσεις ή διακοπές κατά την εκτέλεση των εντολών.

```
1. if (a > c) {
2.      d = d + 5;
3.      c = b + d + e;

   } else {

4.     e = e + 2;
5.     f = f + 2;
6.     a = c  + f;

   }

7. b = a + f;
```

a. Καταγράψτε όλες τις εξαρτήσεις ελέγχου (control dependencies) μεταξύ εντολών. Χρησιμοποιήστε τους αριθμούς των εντολών (1-7) για αυτό το σκοπό. Για κάθε εξάρτηση ελέγχου, αναφέρετε εάν η εξαρτημένη εντολή μπορεί να δρομολογηθεί πριν την εντολή if ή όχι, βάσει των δεδομένων που διαβάζει και γράφει κάθε εντολή **(12%)**
b. Θεωρήστε έναν υπερβαθμωτό (superscalar) επεξεργαστή με δυναμική δρομολόγηση εντολών (dynamic scheduling). Ο επεξεργαστής δεν χρησιμοποιεί τεχνικές speculation και έχει ένα παράθυρο εντολών που μπορεί να αποθηκεύσει ολόκληρη την ακολουθία που δίνεται στην άσκηση. Βρείτε τις εξαρτήσεις δεδομένων (data dependencies) της ακολουθίας και χρησιμοποιήστε αυτή την πληροφορία για να βρείτε ποιες ομάδες εντολών μπορούν να ξεκινήσουν (issue) στον ίδιο κύκλο από τον υπερβαθμωτό επεξεργαστή **(12%)**
c. Υποθέτετε ότι μονο οι τιμές των a, b, και c χρησιμοποιούνται στο πρόγραμμα μετά την εκτέλεση της παραπάνω ακολουθίας.

Χρησιμοποιούμε τον όρο «νεκρές » για μεταβλητές των οποίων η τιμή δεν χρειάζεται μετά την εκτέλεση της παραπάνω ακολουθίας. Εάν μία μεταβλητή είναι νεκρή, οι εντολές που ορίζουν αυτή την τιμή πιθανόν να μπορούν να διαγραφούν από την ακολουθία χωρίς να αλλάζουν το αποτέλεσμα του προγράμματος. Βρείτε ποιες μεταβλητές στον κώδικα που δίνεται είναι νεκρές και ποιες εντολές μπορούν να διαγραφούν χωρίς να αλλάζουν το αποτέλεσμα του προγράμματος (**12%)**

d. Εάν αφαιρέσετε τις εντολές ορισμού νεκρών μεταβλητών, πώς επηρεάζεται η επίδοση του επεξεργαστή; Σχολιάστε με αφορμή το συγκεκριμένο παράδειγμα πώς ο μεταφραστής και το υλικό μπορούν να επηρεάσουν τον παραλληλισμό σε επίπεδο εντολών. **(14%)**

Άσκηση 2 (50%)

Η χρήση μεγαλύτερων πινάκων στους branch predictors σημαίνει ότι μειώνεται η πιθανότητα 2 διαφορετικά branches να μοιραστούν τον ίδιο predictor. Εάν υπάρχει ένας και μοναδικός predictor για κάθε branch, αναμένεται να είναι πιο ακριβής από έναν predictor που διαμοιράζεται μεταξύ περισσότερων της μίας branch εντολών.

a. Υποθέστε τον απλούστερο branch predictor του ενός bit (δηλ. predictor που χρησιμοποιεί το αποτέλεσμα του τελευταίου branch σαν πρόβλεψη), ο οποίος διαμοιράζεται μεταξύ branches. Είναι δυνατόν αυτός ο predictor να πετύχει *καλύτερη* ακρίβεια πρόβλεψης από predictors ενός bit που δεν διαμοιράζονται μεταξύ branches; Εάν ναι, δώστε μία ακολουθία branch αποτελεσμάτων Taken-Not taken με την οποία ο συγκεκριμένος predictor *βελτιώνει* την ακρίβεια πρόβλεψης. Εάν όχι, εξηγήστε τους λόγους **(16%)**
b. Υποθέστε τον απλούστερο branch predictor του ενός bit, ο οποίος διαμοιράζεται μεταξύ branches. Είναι δυνατόν αυτός ο predictor να πετύχει *χειρότερη* ακρίβεια πρόβλεψης από predictors ενός bit που δεν διαμοιράζονται μεταξύ branches; Εάν ναι, δώστε μία ακολουθία branch αποτελεσμάτων Taken-Not taken με την οποία ο συγκεκριμένος predictor *μειώνει* την ακρίβεια πρόβλεψης. Εάν όχι, εξηγήστε τους λόγους **(16%)**
c. Εξηγήστε τους λόγους γιατί σε πραγματικά προγράμματα branch predictors που διαμοιράζονται μεταξύ branches αναμένεται να μειώνουν την ακρίβεια της πρόβλεψης σε σχέση με branch predictors που δεν διαμοιράζονται μεταξύ branches. **(17%)**