3ο Εργαστήριο Αρχιτεκτονικής Η/Υ: MIPS assembly: Υπορουτίνες υπολογισμού parity

Α. Ευθυμίου

Παραδοτέο: Παρασκευή 1 Νοέμβρη, 23:59

Το αντικείμενο αυτής της άσκησης είναι ένα πρόγραμμα που διατρέχει τα στοιχεία ενός πίνακα 32 λέξεων, υπολογίζει το bit περιττής ισοτιμίας (odd parity) για κάθε στοιχείο, και τα βάζει όλα μαζί σε μια λέξη. Κάτι αντίστοιχο θα μπορούσε να χρησιμοποιηθεί για έλεγχο λαθών κατά την μετάδοση της ακολουθίας των λέξεων του πίνακα.

Σε αυτή την εργαστηριακή άσκηση θα γράψετε ένα προγράμμα assembly που χρησιμοποιεί υπορουτίνες. Θα πρέπει να έχετε μελετήσει τα μαθήματα για τη γλώσσα assembly του MIPS που αντιστοιχούν μέχρι και την ενότητα 2.8 του βιβλίου (διάλεξη 6).

Για να ξεκινήσετε, ακολουθήστε τον σύνδεσμο https://classroom.github.com/a/8y1lz3ue. Κάνοντας κλικ στον σύνδεσμο, δημιουργείται έναν νέο αποθετήριο στον οργανισμό του μαθήματος. Μπορείτε να δείτε το νέο αποθετήριο αμέσως μετά το κλίκ στον σύνδεσμο.Το URL του αποθετηρίου θα έχει τη μορφή https://github.com/UoI-CSE-MYY505/lab03-ghUsername, όπου ghUsername το όνομα χρήστη που έχετε στο GitHub.

Κλωνοποιήστε το με την εντολή:

```
git clone https://github.com/UoI-CSE-MYY505/lab03-ghUsername.git
```

Για να πάρετε τα αρχεία της εργαστηριακής άσκησης, μεταβείτε στον κατάλογο που θα δημιουργηθεί από το παραπάνω βήμα και θα έχει το ίδιο όνομα με το αποθετήριο (αλλάζετε το ghUsername με το όνομα χρήστη).

1 Η άσκηση

Η περιττή ισοτιμία (odd parity) ενός δυαδικού αριθμού έχει την τιμή 1 αν το πλήθος των bits με τιμή 1 είναι περιττό. Για παράδειγμα, ο δυαδικός αριθμός 0101 δίνει αποτέλεσμα 0 ενώ ο 1000 δίνει 1. Χρησιμοποιείται για τη δημιουργία ένος bit ελέγχου το οποίο αποθηκεύται μαζί με τα υπόλοιπα bits δεδομένων. Αν, για κάποιο λόγο, 1, 3, ..., bits αντιστραφούν, τότε το λάθος ανιχνεύεται γιατί η περιττή ισοτιμία του αριθμού, συμπεριλαμβανομένου του bit ελέγχου, θα είναι 1, ενώ αν δεν υπάρχουν λάθη θα είναι 0. Αν βέβαια συμβούν 2, 4, ... αντιστροφές, το λάθος δεν ανιχνεύεται. Συνεχίζοντας το προηγούμενο παράδειγμα, το 0101 έχει parity 0, επομένως αποθηκεύεται ως 0101_0, με το επιπλέον bit να είναι το bit ισοτιμίας. Αν όμως, εξαιτίας ενός λάθους, γίνει 0100_0, θα το ανιχνεύσουμε γιατί η ισοτιμία του 0100_0 είναι 1, ενώ αυτή του σωστού 0101_0 θα ήταν 0.

Ο ένας τρόπος υπολογισμού της περιττής ισοτιμίας περιγράφηκε παραπάνω: μέτρηση του πλήθους των 1 στον δυαδικό αριθμό και έλεγχος αν είναι περιττό. Αυτό είναι ισοδύναμο με το αποκλειστικό ή όλων των bits του αριθμού μεταξύ τους. Υπάρχουν και άλλοι αλγόριθμοι, με διάφορα πλεονεκτήματα ως προς τους παραπάνω. Καλό θα είναι να τους διερευνήσετε και, για ανταμοιβή, 1 μονάδα από τον βαθμό της άσκησης θα δοθεί σε όσους χρησιμοποιήσουν έναν πιο «εξελιγμένο» αλγόριθμο. Στο πρώτο μέρος της άσκησης θα γράψετε μια υπορουτίνα, parity, που δέχεται ως είσοδο, στον `$a0`, έναν αριθμό 32 bit και επιστρέφει, στο λιγότερο σημαντικό bit του `$v0`, το bit ισοτιμίας.

Όταν αποθηκεύεται (ή μεταδίδεται) μια ακολουθία λέξεων, αντί για αποθήκευση του bit ισοτιμίας σε κάθε λέξη (που θα ήταν προβληματική καθώς θα χρειαζόταν 33 bits), μπορούν όλα αυτά τα bits ισοτιμίας να μαζευτούν και να ενωθούν σε λέξεις που αποθηκεύονται (ή μεταδίδονται) μετά από τις λέξεις δεδομένων. Στο δεύτερο μέρος της άσκησης θα γράψετε μια υπορουτίνα που θα υπολογίζει τα bits ισοτιμίας των 32 λέξεων ενός πίνακα και θα τα βάζει όλα μαζί, σχηματίζοντας στο αποτέλεσμα της υπορουτίνας (v0) μια νέα λέξη. Η υπορουτίνα, blockParity, παίρνει ως είσοδο (στον `$a0`) την διεύθυνση του πίνακα των 32 λέξεων και επιστρέφει (στον `$v0`) την λέξη με όλα τα bits ισοτιμίας των λέξεων

του πίνακα, χρησιμοποιώντας την υπορουτίνα partiy που περιγράφεται παραπάνω. Το πιο σημαντικό bit (31) του αποτελέσματος θα είναι το parity bit της 1ης λέξης του πίνακα (`array[0]` στο lab03.asm) κ.ο.κ.

Στο αρχείο `lab03.asm` θα βρείτε έναν μικρό σκελετό του κώδικα όπου υπάρχουν δηλωμένες οι υπορουτίνες parity, για τον υπολογισμό του bit περιττής ισοτιμίας μιας λέξης, και blockParity, για τον υπολογισμό των parity bits όλου του πίνακα. Οι υπορουτίνες, ως έχουν, επιστρέφουν αμέσως πίσω. Επίσης υπάρχει το κυρίως πρόγραμμα (main) που καλεί πρώτα την parity με μία τιμή δοκιμής στον a0 και μετά την blockParity με τη διεύθυνση του πρώτου στοιχείου του πίνακα στον a0.

Συμπληρώστε κώδικα στα σημεία που δείχνουν τα σχόλια, γιατί έτσι απαιτεί ο αυτόματος έλεγχος με το Lab03TestParity.java. Για να τρέξετε το αυτόματο τεστ, δώστε τις εντολές:

```
javac -cp munit.jar Lab03TestParity.java
java -jar munit.jar lab03.asm Lab03TestParity.class
```

2 Παραδοτέο και κριτήρια αξιολόγησης

Το παραδοτέο της άσκησης είναι το αρχείο lab03.asm που περιέχει το πρόγραμμά σας. Προεραιτικά αλλάξτε και το README.md για να βλέπετε το badge του Travis CI.

Πρέπει να κάνετε commit τις αλλαγές σας και να τις στείλετε (push) στο αποθετήριό σας στο GitHub για να βαθμολογηθούν πριν από την καταληκτική ημερομηνία!

Τα προγράμματά σας θα βαθμολογηθούν για την ορθότητά τους, την ποιότητα σχολίων και τη ταχύτητα εκτέλεσής τους. Είναι εξαιρετικά σημαντικό να ακολουθήσετε τις συμβάσεις του MIPS για τη χρήση των καταχωρητών που περιγραφηκαν στο μάθημα. Το αυτόματο τεστ προσπαθεί να το εξασφαλίσει αυτό, συνεπώς αν βλέπετε ότι ο αυτόματος έλεγχος αποτυγχάνει ενώ στον MARS τρέχει κανονικά, συνήθως το πρόβλημα είναι ότι έχετε παραβιάσει κάποια σύμβαση (ή δεν έχετε αρχικοποιήσει σωστά κάποιον καταχωρητή). Η ταχύτητα εκτέλεσης σχετίζεται με τον αριθμό εντολών, ειδικά μέσα σε βρόγχο. Επιπλέον, μία μονάδα θα διατεθεί για την υλοποίηση πιο εξελιγμένου αλγορίθμου για τον υπολογισμό της ισοτιμίας από αυτούς που περιγράφηκαν παραπάνω.