3ο Εργαστήριο Αρχιτεκτονικής Η/Υ: MIPS assembly: Υπολογισμός αριθμού δυαδικών μονάδων σε πίνακα ακεραίων με αναδρομή

Α. Ευθυμίου

Παραδοτέο: Τετάρτη 15 Μάρτη, 23:00

Το αντικείμενο αυτής της άσκησης είναι δύο υπορουτίνες. Η πρώτη, popc, υπολογίζει τον αριθμό των δυαδικών μονάδων ενός ακέραιου 32 bit και η δεύτερη, sum_popc, υπολογίζει το ίδιο για έναν ολόκληρο πίνακα ακεραίων. Ο υπολογισμός του αριθμού των δυαδικών μονάδων σε έναν ακέραιο είναι γνωστός ως *population count* και μερικοί επεξεργαστές διαθέτουν ειδική εντολή για το σκοπό αυτό.

Σε αυτή την εργαστηριακή άσκηση θα γράψετε ένα προγράμμα assembly που χρησιμοποιεί υπορουτίνες και αναδρομή. Θα πρέπει να έχετε μελετήσει τα μαθήματα για τη γλώσσα assembly του MIPS που αντιστοιχούν μέχρι και την ενότητα 2.8 του βιβλίου (διάλεξη 7).

Για να πάρετε τα αρχεία της εργαστηριακής άσκησης, μεταβείτε στον κατάλογο εργασίας που είχατε κλωνοποιήσει από το αποθετήριό σας στο GitHub. (cd <όνομα χρήστη GitHub>-labs). Μετά, κάνετε τα παρακάτω βήματα:

```
git remote add lab03_starter https://github.com/UoI-CSE-MYY402/lab03_starter.git
git fetch lab03_starter
git merge lab03_starter/master -m "Fetched lab03 starter files"
```

Στον κατάλογο lab03 σας δίνεται το αρχείο lab03.asm. Υπάρχουν οι ετικέτες των υπορουτινών popc, sum_popc, όπου και πρέπει να γράψετε κώδικα. Επίσης υπάρχει η main που καλεί τις υπορουτίνες που θα γράψετε.

Μετά από κάθε κλήση η main αποθηκεύει το αποτέλεσμα σε διαφορετικό καταχωρητή. Όταν τελειώσει η εκτέλεση **ολόκληρου** του προγράμματος οι τιμές αυτών των καταχωρητών θα πρέπει να είναι οι αναμενόμενες. Μην αλλάξετε τον κώδικα της main ούτε τις θέσεις ή την σειρά των ετικετών δεδομένων, γιατί χρησιμοποιούνται από τον αυτόματο έλεγχο. Δοκιμάστε το πρόγραμμα με άλλους αριθμούς αν θέλετε, αλλά στο τέλος παραδώστε το με τα αρχικά δεδομένα.

Στον MARS για να παρατηρείτε τί συμβαίνει στην περιοχή της μνήμης που αντιστοιχεί στη στοίβα, αλλάξτε την επιλογή στο κάτω μέρος του παραθύρου Data Segment σε "current $sp".

1 Μέρος 1: Population count, 40% του βαθμού

Για το πρώτο μέρος της άσκησης θα γράψετε μια (μη-αναδρομική) υπορουτίνα, με όνομα popc, που υπολογίζει τον αριθμό των δυαδικών μονάδων ενός ακεραίου, μια πράξη γνωστή ως population count. Πιο συγκεκριμένα μετράει τον αριθμό των 1 που υπάρχουν στην αναπαράσταση του αριθμού-εισόδου στο δυαδικό σύστημα. Για παράδειγμα το popc(0) = 0, αφού ο αριθμός 0 δεν έχει καμία μονάδα στην δυαδική του αναπαράσταση, ενώ popc(1) = 1 και popc(256) = 1, αφού οι αριθμοί αυτοί έχουν μόνο 1 μονάδα στην δυαδική τους αναπαράσταση. Το popc(3) = 2, γιατί το 3 είναι 011, συνεπώς έχει δύο μονάδες στην δυαδική του αναπαράσταση.

Υπάρχουν διάφοροι αλγόριθμοι για population count, από απλούς που χρησιμοποιούν μάσκες και ολίσθηση μέχρι αρκετά περίπλοκους που χρησιμοποιούν εντολές διαίρεσης (υπόλοιπο) και πολλαπλασιασμού. Ξεκινήστε με έναν απλό αλγόριθμο και αφού τελειώσετε και το 2ο μέρος, εξετάστε κάποιον καλύτερο-ταχύτερο αλγόριθμο αν μένει χρόνος. Σε καμία περίπτωση όμως μην χρησιμοποιήσετε εντολές διαίρεσης και πολλαπλασιασμού του MIPS (div, divu, mult, multu).

Θα πρέπει να χρησιμοποιήσετε τις γνωστές συμβάσεις του MIPS ως προς τους καταχωρητές που χρησιμοποιούνται για να περάσουν παραμέτρους εισόδου (τους `$a0`-`$a3`), αυτούς που χρησιμοποιούνται για επιστροφή τιμών (`$v0`-`$v1`), καθώς και ποιούς καταχωρητές μπορεί να αλλάξει μια υπορουτίνα χωρίς

να χρειάζεται να επαναφέρει τις προηγούμενες τιμές τους όταν επιστρέφει. Συγκεκριμένα η είσοδος θα είναι η τιμή του καταχωρητή `$a0` και το αποτέλεσμα θα δίνεται στον `$v0`. Μπορείτε να αλλάξετε τιμές σε καταχωρητές `$t0-$t9` χωρίς να χρειάζεται να επαναφέρετε τις προηγούμενες τιμές τους όταν επιστρέφει η υπορουτίνα. Επίσης μπορείτε να αλλάξετε τους `$a0-a4`, `$v0`, `$v1`. Αν όμως χρησιμοποιήσετε τους καταχωρητές `$s0-$s7`, θα πρέπει να επαναφέρετε τις προηγούμενες τιμές τους κατά την επιστροφή.

Η υπορουτίνα popc δεν χρειάζεται να καλεί άλλες υπορουτίνες, επομένως δεν θα χρειαστεί να αποθηκεύσετε τον `$ra`. Γενικά, αν δεν χρησιμοποιήσετε κάποιον από τους `$s0-$s7`, δεν θα χρειαστεί να κάνετε οτιδήποτε με τη στοίβα. Η υπορουτίνα δεν θα πρέπει να στηρίζεται ή να υποθέτει οποιαδήποτε αρχική ή άλλη τιμή καταχωρητή εκτός του `$a0` (και προφανώς των `$sp`, `$ra`, `$zero`). Για παράδειγμα μπορεί ο καταχωρητής `$v0` να μην έχει την τιμή 0 όταν κληθεί η popc.

Τέλος, όταν υλοποιείτε το δεύτερο μέρος της άσκησης μήν μπείτε στον πειρασμό να αλλάξετε κάτι σε αυτή την υπορουτίνα ώστε να γλιτώσετε κάποιες εντολές στην υπορουτίνα sum_popc. Ένα παράδειγμα θα ήταν να χρησιμοποιήσει κανείς αντί για τον `$a0`, τον `$a3` που δεν χρειάζεται στην sum_popc και να γλυτώσει μερικές μεταφορές τιμών μεταξύ καταχωρητών. Κάτι τέτοιο δεν επιτρέπεται γιατί η popc είναι ανεξάρτητη υπορουτίνα και μπορεί να χρησιμοποιηθεί και έξω από την sum_popc.

2 Μέρος 2: Αναδρομικός υπολογισμός population count των στοιχείων ενός πίνακα

2.1 Εισαγωγή

Η αναδρομή είναι μια τεχνική που λύνει ένα "μεγάλο" πρόβλημα λύνοντας μικρότερες εκδοχές του ίδιου προβλήματος και χρησιμοποιώντας τα αποτελέσματα των λύσεων αυτών υπολογίζει το αποτέλεσμα του μεγάλου προβλήματος. Η κεντρική ιδέα είναι ότι τα μικρότερα προβλήματα, σπάνε σε ακόμη μικρότερα, έως ότου η λύση τους είναι τετριμένη.

Για παράδειγμα μπορεί να χρησιμοποιηθεί αναδρομή για την ταξινόμιση ενός πίνακα με την μέθοδο της συγχώνευσης (merge sort): ο πίνακας χωρίζεται σε δύο τμήματα, μισού μεγέθους, τα οποία πρώτα ταξινομούνται το καθένα ξεχωριστά και μετά συγχωνεύονται σε έναν ενιαίο πίνακα, συγκρίνοντας τα στοιχεία των υποπινάκων ανά ζεύγη με τη σειρά. Το μικρότερο στοιχείο κάθε υποπίνακα θα είναι πρώτο στον υποπίνακα, άρα συγκρίνοντας τα δύο πρώτα στοιχεία κάθε φορά μπορούμε εύκολα να συγχωνεύσουμε τους δύο υποπίνακες σε έναν.

Η αναδρομή επιτρέπει πολύ συμπαγείς και κομψές λύσεις ενός προβλήματος αν και συχνά υστερούν σε ταχύτητα εκτέλεσης. Για να λυθεί ένα προβλημα με αναδρομή πρέπει να μπορεί να εκφραστεί με τέτοιο τρόπο ώστε η λύση ενός προβλήματος μεγέθους N είναι ακριβώς η ίδια με τη λύση ενός προβλήματος μικρότερου μεγέθους.

2.2 Αναδρομή σε assembly

Οι αναδρομικές υπορουτίνες σε assembly δεν έχουν κάτι το ιδιαίτερο σε σχέση με τις απλές υπορουτίνες. Ένα συχνό λάθος είναι ότι όταν μια αναδρομική συνάρτηση φτάσει στην τετριμένη περίπτωση, τελειώνει και η αρχική κλήση της υπορουτίνας. Χρησιμοποιώντας το παράδειγμα του μαθήματος, έστω ότι η main καλεί την fact(3), που καλεί την fact(2), που καλεί την fact(1), που τελικά καλεί την fact(0). Η fact(0) είναι η τετριμένη περίπτωση και απλά επιστρέφει την τιμή 1. Η επιστροφή δεν γίνεται πίσω στην main, αλλά στην fact(1). Δηλαδή δεν τελειώνει η εκτέλεση της υπορουτίνας fact συνολικά αλλά μόνο της τελευταίας κλήσης της.

Το ίδιο συμβαίνει με απλές υπορουτίνες, π.χ. μια υπορουτίνα f() που καλεί μία g() που καλεί μία h(). Όταν τελειώσει, με μία `jr $ra`, η h() θα επιστρέψει στην g() στην αμέσως επόμενη εντολή από εκεί που καλέστηκε (`jal h`) και όταν η g() τελειώσει θα επιστρέψει στην f().

Εφόσον τηρούνται οι γνωστές συμβάσεις για αποθήκευση τιμών καταχωρητών (συμπεριλαμβανομένου του καταχωρητή διεύθυνσης επιστροφής, `$ra`), μία υπορουτίνα μπορεί να καλέσει τον εαυτό της και, όταν επιστρέψει, θα συνεχίσει την εκτέλεση από την εντολή μετά την `jal`. Για να δουλέψει αυτό σωστά

χρησιμοποιούμε την στοίβα. Όταν καλούμε μια υπορουτίνα, τοποθετούμε πληροφορία στη στοίβα, η οποία μεγαλώνει. Όταν επιστρέφει μια υπορουτίνα, ο χώρος που χρειάστηκε στη στοίβα για να αποθηκεύσει πληροφορίες είναι πλέον άχρηστος. Επομένως η στοίβα μικραίνει.

Σε αναδρομικές συναρτήσεις όταν μεγαλώνει η στοίβα (δηλαδή καλούμε τον εαυτό μας) προχωράμε προς την τετριμένη περίπτωση. Όταν αυτή βρεθεί και λυθεί, επιστρέφει προς τα πίσω, μικραίνοντας τη στοίβα και συνθέτοντας σιγά-σιγά την λύση. Αυτό επαναλαμβάνεται μέχρι να βρεθούμε στην πρώτη κλήση της αναδρομικής υπορουτίνας (το fact(3) στο παράδειγμα), όπου και υπολογίζεται η τελική τιμή.

2.3 Αναδρομική υπορουτίνα sum_popc

Στο δεύτερο μέρος, θα χρησιμοποιήσετε την popc για να γράψετε την υπορουτίνα sum_popc που υπολογίζει το population count όλων των αριθμών (32bit ακέραιοι) ενός πίνακα. Ο αναδρομικός αλγόριθμος που θα χρησιμοποιηθεί είναι ο ακόλουθος. Δεν επιτρέπεται να χρησιμοποιήσετε διαφορετικό αλγόριθμο για την άσκηση.

```
int sum_popc(int array[], int size) {
   if (size == 0)
      return 0;
   else
      return sum_popc(array, size-1) + popc(array[size-1]);
}
```

Η πρώτη είσοδος είναι η διεύθυνση του πίνακα και θα είναι στον καταχωρητή `$a0`. Όπως όλες οι διευθύνσεις πινάκων, είναι η διεύθυνση του πρώτου στοιχείου (του `array[0]`), ενώ τα υπόλοιπα στοιχεία είναι σε συνεχόμενες, αύξουσες διευθύνσεις μνήμης. Η δεύτερη είσοδος είναι το μέγεθος του πίνακα, size, ο συνολικός αριθμός στοιχείων που περιέχει, και θα είναι στον καταχωρητή `$a1`. Αφού το πρώτο στοιχείο του πίνακα είναι το `array[0]`, το τελευταίο θα είναι το `array[size-1]`. Κάθε στοιχείο του πίνακα είναι ένας αριθμός 32 bit.

Ο αλγόριθμος υπολογίζει αναδρομικά το population count του πίνακα, αλλά με μέγεθος μειωμένο κατά 1 καθε φορά και σε αυτό προσθέτει το population count του τελευταίου στοιχείου του πίνακα. Στην τετριμένη περίπτωση, που τελειώνει η αναδρομή, το μέγεθος του πίνακα είναι 0, συνεπώς ο πίνακας είναι άδειος και επιστρέφεται η τιμή 0. Προσοχή, όταν τελειώνει η αναδρομή επιστρέφεται το 0 και όχι το `popc(array[0])`!

Η τελευταία γραμμή έχει κάποιες ομοιώτητες, αλλά δεν είναι ίδια, με την αντίστοιχη του factorial που εξηγήθηκε σε προηγούμενο φυλλάδιο που θα βρείτε αναρτημένο στην ιστοσελίδα του μαθήματος στο ecourse. Προσέξτε με ποιά σειρά θα γίνουν οι υπολογισμοί, οι αναφορές στην μνήμη και οι κλήσεις υπορουτινών.

Όπως έχει τονιστεί επανειλημμένα, η sum_popc θα πρέπει να ακολουθεί τις γνωστές συμβάσεις του MIPS ως προς τις κλήσεις υπορουτινών: Για παράδειγμα μην περνάτε τιμές σπό μια συνάρτηση σε άλλη έξω από τους `$a0, $a1`, ακόμη και για αναδρομικές κλήσεις της ίδιας υπορουτίνας.

3 Παραδοτέο

Το παραδοτέο της άσκησης είναι το αρχείο lab03.asm που περιέχει το πρόγραμμά σας. Στον κατάλογο test το lab03_tester.py περιέχει έλεγχο για το πρόγραμμα χωρίς να αλλάζει τα δεδομένα, γι'αυτό και δεν πρέπει να αλλάξετε τον κώδικα της main και τις υπάρχουσες τιμές δεδομένων. Δεν χρειάζεται να κάνετε αλλαγές στο lab03_tester.py, παρά μόνο να επιβεβαιώσετε ότι το πρόγραμμά σας περνάει τον έλεγχο.

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

Το πρόγραμμά σας θα βαθμολογηθεί για την ορθότητά του, την ποιότητα σχολίων και την συνοπτικότητά του.