Μικροεπεξεργαστές και Περιφερειακά

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών ΑΠΘ
8ο εξάμηνο

Πιπινιάς Ιωάννης[1] και Φανάκης Απόστολος[2]

[1]*7965*, *pipinias@ece.auth.gr*
[2]*8261*, *apostolof@ece.auth.gr*

26 Απριλίου 2021

1 Εισαγωγή

Το παρόν έγγραφο είναι η αναφορά του πρώτου εργαστηρίου στο μάθημα Μικροεπεξεργαστές και Περιφερειακά. Στο πρώτο εργαστήριο μας ζητήθηκε να υλοποιήσουμε κώδικα που θα υπολογίζει την τιμή κατακερματισμού (hash) μίας αλφαριθμητικής ακολουθίας. Η βασική ρουτίνα main υλοποιήθηκε στη γλώσσα C και περιλαμβάνει την αρχικοποίηση του αλφαριθμητικού, την κλήση της συνάρτησης κατακερματισμού και την τύπωση της τιμής στην κονσόλα. Η συνάρτηση κατακερματισμού υλοποιήθηκε σε ARM Assembly, δέχεται ως ορίσματα το αλφαριθμητικό προς κατακερματισμό και τη διεύθυνση ενός lookup table με τις τιμές κατακερματισμού κάθε γράμματος που μας δόθηκαν στην εκφώνηση της άσκησης.

Ο κώδικας είναι διαθέσιμος στο παράρτημα καθώς και στο Gitlab repository εδώ.

2 Υλοποίηση

Στα πλαίσια του εργαστηρίου υλοποιήθηκε συνάρτησης σε ARM Assembly με την παρακάτω υπογραφή:

```
int generate_hash( const char *str, const uint8_t *hashtbl, int *hashptr )
```

Αρχικά, δόθηκαν ονόματα στους καταχωρητές οι οποίοι χρησιμοποιήθηκαν μέσα στη συνάρτηση ώστε αυτή να είναι πιο εύχρηστοι. Τονίζεται εδώ ότι ο καταχωρητής R0 είναι ο δείκτης (pointer) προς το αλφαριθμητικό (πρώτο όρισμα της συνάρτησης), ο καταχωρητής R1 είναι ο δείκτης προς το lookup table (δεύτερο όρισμα της συνάρτησης), ενώ ο καταχωρητής R3 είναι ο δείκτης προς τον ακέραιο αριθμό στον οποίο θα αποθηκευτεί το αποτέλεσμα της συνάρτησης (τρίτο όρισμα της συνάρτησης).

Για την αποθήκευση και εύρεση των τιμών κατακερματισμού κάθε γράμματος δημιουργήθηκε ένα lookup table που αποθηκεύει στη μνήμη πίνακα με τις τιμές των γραμμάτων σειριακά. Έτσι, για την εύρεση της κατάλληλης τιμής αρκεί ο pointer που δείχνει στον πίνακα να μετακινηθεί τόσες θέσεις όσες η απόσταση του εκάστοτε γράμματος από τον χαρακτήρα A στο ASCII table.

Έπειτα υλοποιήθηκε ένα τμήμα κώδικα με ετικέτα (label) hash_loop. Ο κώδικας αυτός φορτώνει στη μνήμη το επόμενο γράμμα και μετά από μία σειρά ελέγχων αποφασίζει τη σωστή διαχείριση:

- Αν ο χαρακτήρας είναι μικρότερος από τον χαρακτήρα "0" στο ASCII table ή είναι ίσος με αυτόν, τότε το πρόγραμμα προσπερνάει αυτόν τον χαρακτήρα και πηγαίνει στο label hash_skip
- Αν ο χαρακτήρας είναι μικρότερος ή ίσος με το "9" στο ASCII table (δηλαδή ο χαρακτήρας είναι 1-9) τότε η τιμή του αφαιρείται από το hash_val και έπειτα προστίθεται 48 διότι τα αριθμητικά ψηφία στο ASCII table ξεκινούν από το 48
- Αν ο χαρακτήρας είναι μικρότερος από τον χαρακτήρα "A", τότε το πρόγραμμα προσπερνάει τον χαρακτήρα και πηγαίνει στο label hash_skip
- Αν ο χαρακτήρας είναι μεγαλύτερος από τον χαρακτήρα "Z", τότε το πρόγραμμα προσπερνάει τον χαρακτήρα και πηγαίνει στο label hash_skip
- Σε κάθε άλλη περίπτωση (δηλαδή ο χαρακτήρας είναι A-Z), αφαιρούμε την τιμή 65 (η τιμή του "A" στο ASCII table) από τον χαρακτήρα ώστε να πάρουμε τη θέση του χαρακτήρα στο lookup table, έπειτα προσθέτουμε στη θέση τη διεύθυνση του lookup table, φορτώνουμε την τιμή της θέσης που έχει υπολογιστεί και την προσθέτουμε στο hash_val

Το τελευταίο τμήμα κώδικα που υλοποιήσαμε είναι αυτό με label hash_skip. Σε αυτόν τον κώδικα αυξάνουμε την τιμή του pointer του αλφαριθμητικού κατά ένα. Μετά ελέγχουμε αν ο τελευταίος χαρακτήρας που διαβάστηκε είναι ίσος με τον χαρακτήρα NUL (τερματισμός string της C, "/0") και αν η ισότητα ισχύει, άρα έχουμε φτάσει στο τέλος του αλφαριθμητικού επομένως μεταφέρουμε το hash_val στον καταχωρητή r0 ο οποίος είναι ο καταχωρητής που επιστρέφει και επιστρέφουμε από τη συνάρτηση. Ενώ, σε περίπτωση που η ισότητα δεν ισχύει κάνουμε branch εκ νέου στο label hash_loop.

3 Προβλήματα που αντιμετωπίστηκαν

Αρχικά έγινε προσπάθεια δημιουργίας του lookup table μέσα στο κομμάτι κώδικα της Assembly. Μετά από διάφορες δοκιμές και προσεγγίσεις, η προσπάθεια αυτή απέτυχε. Ως λύση στη δήλωση και αρχικοποίηση του lookup table, έγινε ορισμός του ως στατική (static), global σταθερά (const) της C και η διεύθυνση του πίνακα δόθηκε ως παράμετρος στη συνάρτηση.

4 Δοκιμές (testing)

Για τον έλεγχο της υλοποίησης έγινε δοκιμή του κώδικα καλώντας τη συνάρτηση κατακερματισμού και χρησιμοποιώντας την κονσόλα ως έξοδο του αποτελέσματος. Η υλοποίηση δοκιμάστηκε με το αλφαριθμητικό που περιλαμβάνει όλους τους χαρακτήρες (σε τυχαία σειρά) του ASCII table (εκτός του χαρακτήρα NUL που χρησιμοποιείται για τον τερματισμό). Αποδεικνύεται μαθηματικά ότι αυτός ο έλεγχος είναι επαρκής.

5 Παράρτημα Α

5.1 Κώδικας

```
#include <stdio.h>
#include <stdint.h>

static const uint8_t hashtbl[] = {
  18, 11, 10, 21, 7, 5, 9, 22, 17, 2, 12, 3, 19, 1, 14, 16, 20, 8, 23, 4, 26, 15, 6,
    24, 13, 25
};

__asm int generate_hash( const char *str, const uint8_t *hashtbl, int *hashptr )
{
input_str RN r0
hashtbl   RN r1
hashptr   RN r2
curr_char RN r6
hash_val  RN r3

  MOV   hash_val, #0
hash_loop
  LDRB  curr_char, [input_str]
  CMP   curr_char, #'0'
  BLS   hash_skip

  CMP   curr_char, #'9'
  SUBLS hash_val, curr_char
  ADDLS hash_val, #48
  BLS   hash_skip

  CMP   curr_char, #'A' - 1
  BLS   hash_skip

  CMP   curr_char, #'Z'
  BHI   hash_skip

  SUB   r4, curr_char, #65
  ADD   r4, hashtbl, r4
  LDRB  r5, [r4]
  ADD   hash_val, r5

hash_skip
  ADD   input_str, input_str, #
  CMP   curr_char, #0
  BNE   hash_loop
  MOVEQ r0, hash_val
  STR   hash_val, [hashptr]
  BX    lr
}

int main( void )
{
  static const char STRING_TO_HASH[] = "FD7N8JT!EGMIQH2@3ZWOABCRSLOYV45PK1#X6U!9";
  int hash = 0;
  int hash2;

  hash = generate_hash(STRING_TO_HASH, hashtbl, &hash2);
  printf("%d", hash);
  printf("%d", hash2);

  return 0;
}
```