Μικροεπεξεργαστές και Περιφερειακά

Εαρινό Εξάμηνο 2023

1η Εργασία/Εργαστήριο

Ο κώδικας της εργασίας περιλαμβάνει 2 αρχεία, ένα αρχείο σε γλώσσα C και ένα αρχείο σε γλώσσα assembly.

Στον κώδικα της C στην main συνάρτηση ο χρήστης εισάγει ένα αλφαριθμητικό με μία scanf και δίνεται ως όρισμα στην συνάρτηση myhash. Αφού επιστραφεί το αποτέλεσμα της myhash, εισάγεται ως όρισμα στην συνάρτηση factorial, η οποία επιστρέφει το ζητούμενο παραγοντικό. Οι συναρτήσεις myhash και factorial καλούνται μέσα στις συναρτήσεις sprintf, οι οποίες μετατρέπουν απευθείας τους integer που επιστρέφονται σε string. Έπειτα, χρησιμοποιούνται 3 uart_print, μια για την εκτύπωση του hash, μια για αλλαγή σειράς και μια για τη factorial.

```
sprintf(s1, "The hash of the string is  %d", myhash(str));
sprintf(s2, "The factorial of the one digit hash is %d", factorial(myhash(str)));
uart_print(s1);
uart_print("\r\n");
uart_print(s2);
```

Επίσης, υλοποιήσαμε μία συνάρτηση letterToNumber, η οποία δέχεται ως είσοδο μία μεταβλητή τύπου char και την μετατρέπει σε έναν integer χρησιμοποιώντας την switch case. Σύμφωνα με τον πίνακα που δίνεται στην εκφώνηση αν ο char είναι κεφαλαίο γράμμα η συνάρτηση επιστρέφει τον αντίστοιχο integer, αν η char μεταβλητή είναι αριθμός επιστρέφει την απόλυτη τιμή του με αρνητικό πρόσημο και αν είναι οτιδήποτε άλλο επιστρέφει μηδέν. Χρησιμοποιείται μόνο για μερικές επιπλέον printf που μας βοηθούν στο debugging.

Στο αρχείο της assembly έχουμε 2 συναρτήσεις που αναφέρθηκαν παραπάνω, την myhash και την factorial. Στην συνάρτηση myhash αρχικά αποθηκεύουμε στον link register την τρέχουσα διεύθυνση του προγράμματος. Στην συνέχεια μηδενίζουμε τον r2 και ξεκινάει το loop. Εκεί φορτώνουμε με την εντολή ldrb r1, [r0], #1 στον r1 ένα byte την διεύθυνση του ορίσματος r0, η οποία με κάθε επανάληψη αυξάνεται κατά 1, και ξεκινάει η loop. Έπειτα συγκρίνουμε το 0 με το στοιχείο για να ελέγξουμε αν είναι το τελευταίο του πίνακα. Αν αυτό ισχύει, πηγαίνει στο end_procedure. Αλλιώς, ξεκινάει να συγκρίνει την τιμή του καταχωρητή r1 με όλα τα γράμματα και τους μονοψήφιους αριθμούς. Αν ο r1 ταυτίζεται με κάποιο κεφαλαίο γράμμα, προσθέτει στο καταχωρητή r2 την ακέραια τιμή που αντιστοιχεί στο γράμμα, σύμφωνα με τον πίνακα της εκφώνησης και κάνει branch στη loop.

Π.χ

```
cmp     r1, #'A'
it eq
addeq   r2, r2, 10
beq     loop
```

Αν είναι αριθμός, αφαιρεί από τον r2 την ακέραια τιμή του αριθμού και κάνει branch στη loop. Π.χ

```
cmp     r1, #'1'
it eq
subeq   r2, r2, 1
beq     loop
```

Στο τέλος της σύγκρισης με τον αριθμό 9 κατευθείαν κάνει branch στη loop ώστε αν ο χαρακτήρας δεν είναι αριθμός ή κεφαλαίο γράμμα να μην επηρεάζεται ο r2 και να προχωράει στον επόμενο χαρακτήρα.

Στην end_procedure, συγκρίνει τον r2 με το 0 και ,αν είναι αρνητικό, βάζει στον r2 την τιμή 255. Στη συνέχεια, μετακινεί την τιμή του r2 στον r0 για να την επιστρέψει.

```
end_procedure:
        cmp r2, #0
        it lt
        movlt r2, #0xff

        mov r0, r2
        pop{lr}
        bx lr
```

Στην συνάρτηση factorial, αποθηκεύουμε την τρέχουσα διεύθυνση στον lr, μετακινούμε το όρισμα στον r1 και μηδενίζουμε τον r2. Στην συνέχεια καθώς μπαίνουμε στο label comp, συγκρίνουμε τον r1 με το 10 για να δούμε αν έχει παραπάνω από ένα ψηφίο και μηδενίζεται ο r3 για λόγο που θα εξηγηθεί στη συνέχεια.

```
comp:   cmp   r1, #10
        mov   r3, #0
        bge   loop1
        add   r1, r1, r2
```

```
loop1:  sub   r1, r1, #10
        add   r3, r3, #1
        cmp   r1, #10
        bge   loop1
        add   r2, r2, r1
        mov   r1, r3
        b     comp
```

Στη σύγκριση στην αρχή της comp:

- **Αν ο r1 είναι μεγαλύτερος του 10** κάνουμε branch στο label loop1. Εκεί, αφαιρεί το 10 από τον r1 και αυξάνει τον καταχωρητή r3, ο οποίος μετράει πόσες δεκάδες έχουν αφαιρεθεί από τον r1. Αν ο r1 συνεχίζει να παραμένει μεγαλύτερος του 10 επαναλαμβάνεται η loop1, ενώ αν δεν είναι, προστίθεται ο r1 στον r2 και μετακινείται ο r3 στον r1. Άρα πλέον στον r1 περιέχεται ο αριθμός των δεκάδων του αρχικού αριθμού ενώ στον r2 οι μονάδες του. Έπειτα, κάνει branch στην comp όπου ελέγχεται αν ο νέος r1 είναι μεγαλύτερος του 10. Αν είναι, επαναλαμβάνεται η ίδια διαδικασία με πριν θεωρώντας το περιεχόμενο του r1 έναν νέο αριθμό. Εκεί μηδενίζεται και ο r3, καθώς αν ξαναμπεί στην loop1 θα πρέπει να ξεκινάει να μετράει τις δεκάδες από το μηδέν.

```
        cmp  r1, #10
        bge   two_digits_after_addition
        mov   r2, #1
        b      fact

two_digits_after_addition:
        mov   r2, #0
        b comp
```

- **Αν ο r1 δεν είναι μεγαλύτερος του 10**, προστίθεται στον r1 το r2. Αν το αποτέλεσμα της αρχικής πρόσθεσης όλων των ψηφίων προκύψει διψήφιος, από την cmp γίνεται branch στην two_digits_after_addition. Εκεί, μηδενίζεται ο r2, γιατί θεωρούμε τον r1 ως ένα καινούριο διψήφιο, και γίνεται branch στην comp για να επαναλάβει την αρχική διαδικασία. Όπως φαίνεται παραπάνω.

```
fact:   mul   r2, r2, r1
        sub   r1, r1, #1
        cmp   r1, #0
        bne   fact
```

Αν έχουμε φτάσει σε μονοψήφιο αριθμό, βάζουμε στον r2 την τιμή 1 και κάνουμε branch στην fact. Στην fact πολλαπλασιάζουμε τον r2 με τον μονοψήφιο και αποθηκεύουμε το αποτέλεσμα στον r2. Έπειτα, μειώνουμε κατά 1 τον r1 και αν δεν έχει φτάσει στο μηδέν επαναλαμβάνουμε την fact. Τέλος μετακινούμε τον r2 στον r0 για να επιστρέψουμε το αποτέλεσμα και επιστρέφουμε στην διεύθυνση του κυρίως προγράμματος.

Testing:

Δοκιμάσαμε αρκετά αλφαριθμητικά με simulation debugging ώστε να καλύψουμε κάθε περίπτωση. Επίσης χρησιμοποιήσαμε αρκετές printf σε πολλά σημεία του κώδικα για να ελέγξουμε την πορεία του προγράμματος. Τέλος, στις διαθέσιμες ώρες των εργαστηρίων κάναμε testing σε ST-Link debugger.

Άννα Τσιτσάνου ΑΕΜ: 10051

Σιταρίδης Παναγιώτης ΑΕΜ: 10249