Αναφορά 1ου εργαστηρίου

Παρούσης Σταύρος Δήμος 8892 / Πουνιώτης Σταύρος 8995

Στα πλαίσια της εργασίας χρειάστηκε να υλοποιήσουμε δύο ρουτίνες, την βασική ρουτίνα του προγράμματος σε c και την υπορουτίνα που θα υπολογίζει το hash του αλφαριθμητικού σε arm assembly.

Όσον αφορά τη βασικη ρουτίνα, αρχικά ορίσαμε στατικά το αλφαριθμητικό μας.

```
char str[] = " Ar, PE 2!";
```

Στη συνέχεια μέσα στη main ορίσαμε την ακέραια μεταβλητή out στην οποία θα αποθηκευτεί η τιμή του hash και καλέσαμε τη συνάρτηση hash με όρισμα τον πίνακα με το αλφαριθμητικό.

```
int main (void){
int out; out = hash(str);
printf("%d",out);
return 0; }
```

Προκειμένου να γίνει η σύνδεση μεταξύ της c ρουτίνας και της arm assembly ρουτίνας χρησιμοποιούμε το keyword extern.

```
extern int hash(char* str);
```

Σχετικά με την υπορουτίνα, αρχικά στην περιοχή των δεδομένων όριζουμε σε ένα πίνακα map 26 words, οι οποίες είναι οι τιμές που αντιστοιχούν σε κάθε κεφαλαίο γράμμα και θα προστίθενται στο hash κάθε φορά που εμφανίζονται στο αλφαριθμητικό.

```
      AREA myData, DATA, READWRITE
map   DCD 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 ;
```

Στη συνέχεια στην περιοχή κώδικα κάνουμε τη σύνδεση με την ρουτίνα μέσω του keyword export και ορίζεται η αρχή της συνάρτησης με την εντολή PROC.

```
     AREA myCode, CODE
     EXPORT hash
     ALIGN
hash PROC
```

Αφού γίνουν Push ορισμένοι καταχωρητές σε μια στοίβα αποθηκεύται η διεύθυνση του char pointer του αλφαριθμητικού στον καταχωρητή 2 και η διεύθυνση που δείχνει στο map στον καταχωρητή 3 ενώ ο καταχωρητής 4 αρχικοποιείται ως μηδέν

```
PUSH {r4-r7,lr}
MOV r2,r0 ;copies the value of R0 into R2 (address)
LDR r3, =map ;storing map array of integers to register3
MOV r4,#0 ;initialize r4=0
```

Έπειτα μέσω της loop υπολογίζεται το μέγεθος του αλφαριθμητικού και στη συνέχεια αρχικοποιούνται ο καταχωρητής 0 ως μηδέν (ο οποίος θα είναι η έξοδος της συνάρτησης), ο καταχωρητής 4 ως μηδέν (σαν i της for που ακολουθεί) και ο καταχωρητής 1 ως μέγεθος αλφαριθμητικού - 1 (που θα χρησιμοποιηθεί για τη for)

```
loop LDRB r1, [r2, r4] ;calculating length(str)
     CBZ r1, exit ; Compare and Branch on Zero
     ADD r4, r4, #1 ; r4 = r4 + 1
     B loop
exit SUB r1, r4, #1 ;setting r1 equal to length(str)-1 for the loop
     MOV r0, #0 ;set output = 0
     MOV r4, #0 ;set i = 0
```

Τέλος, για κάθε στοιχείο το αλφαριθμητικού γίνεται έλεγχος αν ανήκει μεταξύ A-Z ή 1-9 και αναλόγως προστίθεται η αντίστοιχη τιμή του map αν είναι κεφαλαίο γράμμα ή αφαιρείται αν είναι αριθμός.

Λόγω μεγαλης έκτασης του συγκεκριμένου κομματιού κώδικα αναφέρουμε πως είναι ανεβασμένος στο [Github - 1st Lab codes and report](#).

Τα προβλήματα που αντιμετωπίσαμε αφορούσαν κυρίως την επιλογή των απαραίτητων πακέτων από το Run-Time Environment, την εκμάθηση της arm assembly (υπήρχε παρ'ολ'αυτά ήδη μια ενασχόληση στο μάθημα της Αρχιτεκτονικής Υπολογιστών με mips assembly) και την προσαρμογή με το λογισμικό Keil uVision.

Τέλος όσον αφορά το testing, χρησιμοποιήσαμε το debug session που προσφέρει το keil για να παρακολουθήσουμε την ορθή λειτουργία του κώδικα. Ιδιαίτερα βοηθητική ήταν η καρτέλα registers (φαίνεται στην παρκάτω εικόνα) για να παρακολουθούμε τις τιμές των διευθύνσεων αλλά και των απλών τιμών που λαμβάνουν οι καταχωρητές. Επίσης μέσω των πακέτων STDERR, STDIN, STDOUT που βρίσκονται στο Run-Time Environment-> Compiler ->I/O χρησιμοποιήσαμε το View -> Serial Windows -> Debug (printf) Viewer για να δούμε τις τιμές που επιστρέφονται στη βασική ρουτίνα μας.