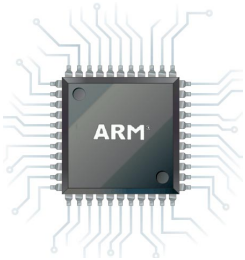


Μικροεπεξεργαστές και Περιφερειακά
-Εργαστήριο 1-

Αλέξανδρος Πετρίδης 9288
Αλέξανδρος Οικονόμου 9260

Τελευταία ενημέρωση: 28 Απριλίου 2021

1 Συνάρτηση *hash*

Στα πλαίσια της παρούσας εργασίας έχει υλοποιηθεί μια ρουτίνα σε *assembly*, η οποία δημιουργεί το *hash* από μία αλφαριθμητική ακολουθία. Η συνάρτηση ακολουθεί τους παρακάτω τρείς κανόνες:

1. Για κάθε κεφαλαίο λατινικό γράμμα προστίθεται στο *hash* ο αριθμός που αντιστοιχεί στον παρακάτω πίνακα.
2. Για κάθε αριθμητικό ψηφίο αφαιρείται η τιμή του από το *hash*.
3. Το *hash* δεν επηρεάζεται από οποιοδήποτε άλλο στοιχείο.

$$\begin{bmatrix} 18 & 11 & 10 & 21 & 7 & 5 & 9 & 22 & 17 & 2 \\ A & B & C & D & E & F & G & H & I & J \end{bmatrix} \begin{bmatrix} 12 & 3 & 19 & 1 & 14 & 16 & 20 & 8 & 23 & 4 \\ K & L & M & N & O & P & Q & R & S & T \end{bmatrix} \begin{bmatrix} 26 & 15 & 6 & 24 & 13 & 25 \\ U & V & W & X & Y & Z \end{bmatrix}$$

Η συνάρτηση μας δέχεται σαν ορίσματα το αλφαριθμητικό του οποίου το *hash* θέλουμε να υπολογίσουμε και τον παραπάνω πίνακα ο οποίος περιέχει τις αντίστοιχες τιμές για το *hash* των κεφαλαίων γραμμάτων. Η βασική ιδέα είναι να ελέγχουμε κάθε χαρακτήρα του αλφαριθμιτικού ξεχωριστά και εφόσον δεν είναι το $'\backslash 0'$, δηλαδή ο τελευταίος χαρακτήρας ενός αλφαριθμητικού, τότε συνεχίζουμε στον υπολογισμού του *hash*. Ο υπολογισμός του *hash* γίνεται σύμφωνα με τους παραπάνω κανόνες. Η αντιστοίχιση των γραμμάτων γίνεται κάνοντας *typecast* αρχικά σε *int* τον χαρακτήρα που ελέγχουμε και μέσω του πίνακα *ascii* καταλαβαίνουμε αν μιλάμε για κεφαλαίο γράμμα, αριθμητικό ψηφίο ή κάτι άλλο.
Στο Σχήμα 1 φαίνεται αναλυτικότερα η ροή της διαδικασίας που ακολουθείται.

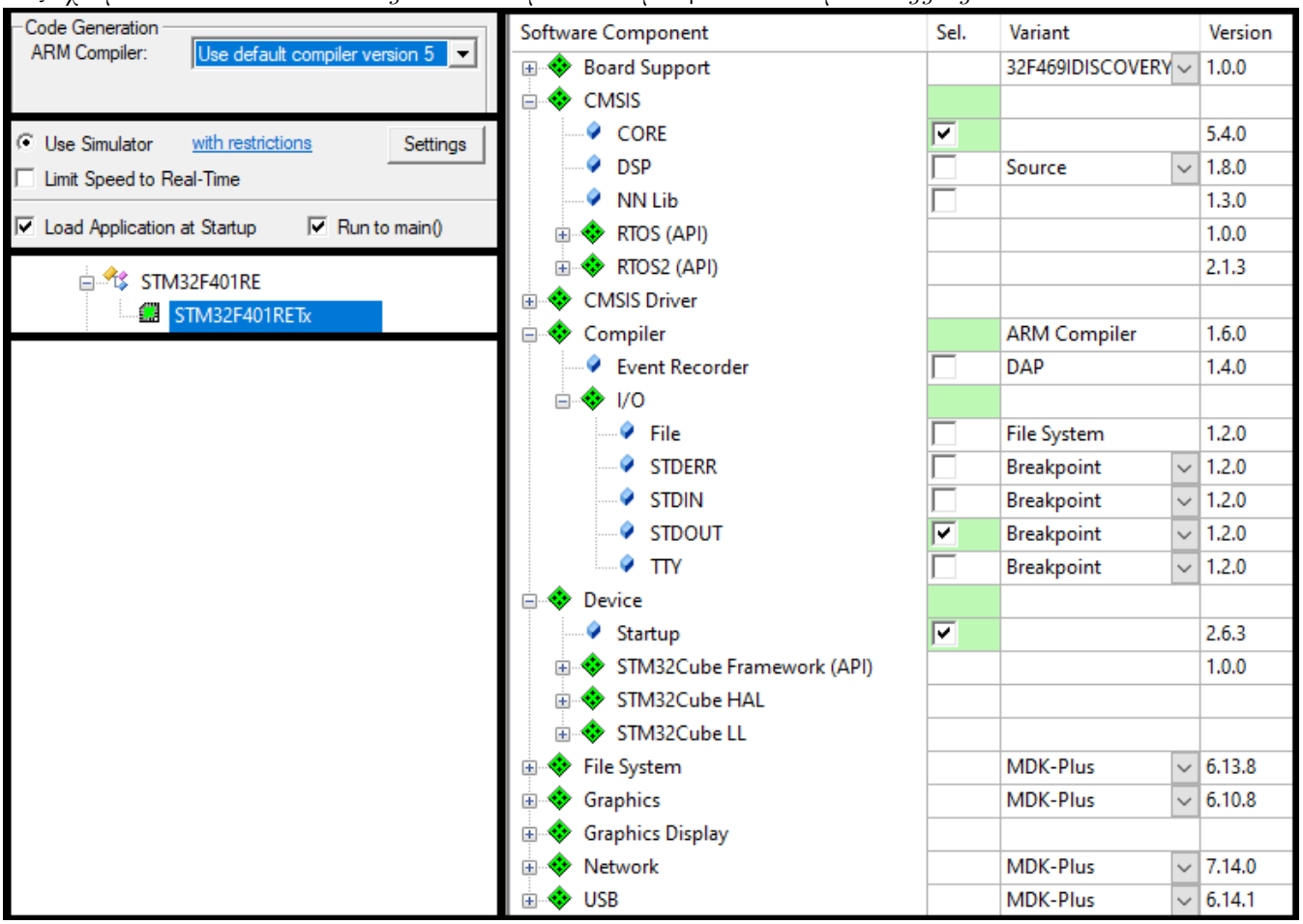
2 Διαδικασία στο *Keil*

Τα βήματα που ακολουθήσαμε είναι τα ακόλουθα:

1. Εφόσον έχουμε γράψει την συνάρτηση σε *assembly* και την *main* μας σε *c*, όπως και μας ζητείται, αρχικά παρόλο που στον *editor* του *Keil* μας βγάζει *error* στην πρώτη και την τέταρτη γραμμή, στον ορισμό και στο *push* της ρουτίνας μας όπως φαίνεται και παρακάτω

```
1  __asm int hash(char *str, int *keys)
2  {
3    //*r0 = &str[0], *r1 = &keys[0]
4    push {r4-r5, lr}  //Save r4-r5 (preserved registers) and link register (return address)
```

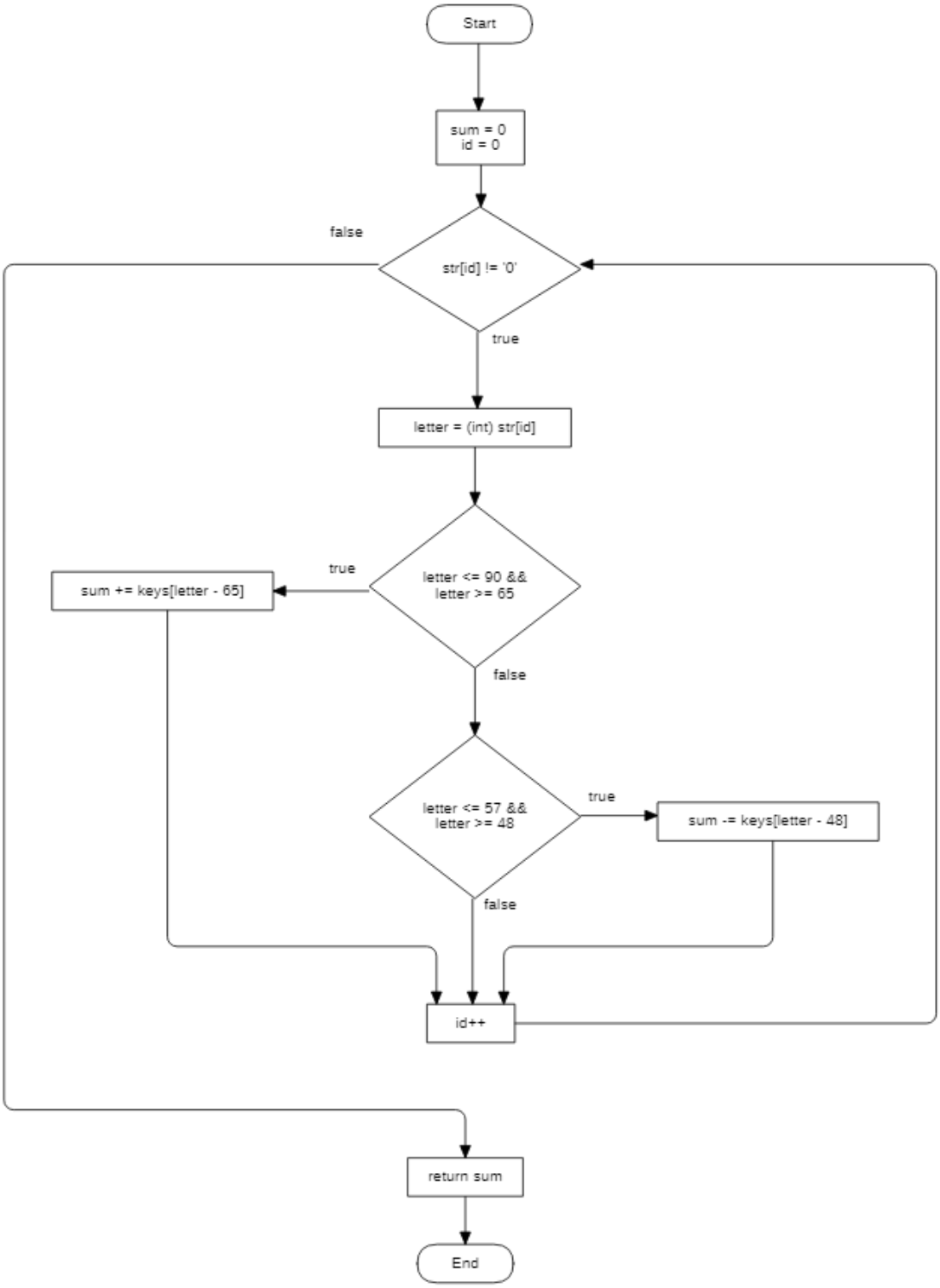
κάνοντας *translate* και *build* με τις παρακάτω ρυθμίσεις και τους παρακάτω *components* το *project* μας, μας εμφανίζει πως έχουμε 0 *errors* και 0 *warnings*. Οπότε είμαστε έτοιμοι για να κάνουμε *debugging*.



2. Πριν αρχίσουμε το *debugging* έχουμε υπολογίσει με το χέρι το αποτέλεσμα που θα έπρεπε να βγάλει σαν σωστό η συνάρτησή μας για το αλφαριθμιτικό "$aA1d$", το οποίο είναι $18 - 1 = 17$.

3. Αρχίζοντας το *debugging* γραμμή γραμμή, παρατηρούμε πως οι καταχωρητές μας παίρνουν τις τιμές που πρέπει και στο τέλος όταν επιστρέφουμε από την *hash* στην *main* ο καταχωρητής $r0$, έχει την τιμη $0x00000011$ που μεταφράζεται από 16-δικό σε 10-δικό στην τιμή 17, όπως περιμέναμε.

3 Διάγραμμα Ροής



Σχήμα 1: Διάγραμμα Ροής Αλγορίθμου