

Μικροεπεξεργαστές και Περιφερειακά -Εργαστήριο 1-

Αλέξανδρος Πετρίδης 9288
Αλέξανδρος Οικονόμου 9260
Ομάδα 35

Τελευταία ενημέρωση: 8 Μαΐου 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 τρεις 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.

18	11	10	21	7	5	9	22	17	2
A	B	C	D	E	F	G	H	I	J

12	3	19	1	14	16	20	8	23	4
K	L	M	N	O	P	Q	R	S	T

26	15	6	24	13	25
U	V	W	X	Y	Z

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 αλφαριθμητικού ξεχωριστά και εφόσον δεν είναι το '\0', δηλαδή ο τελευταίος χαρακτήρας ενός αλφαριθμητικού, τότε συνεχίζουμε στον υπολογισμό 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:

- 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*.

Code Generation
ARM Compiler: Use default compiler version 5

☒ Use Simulator [with restrictions](#) Settings

☐ Limit Speed to Real-Time

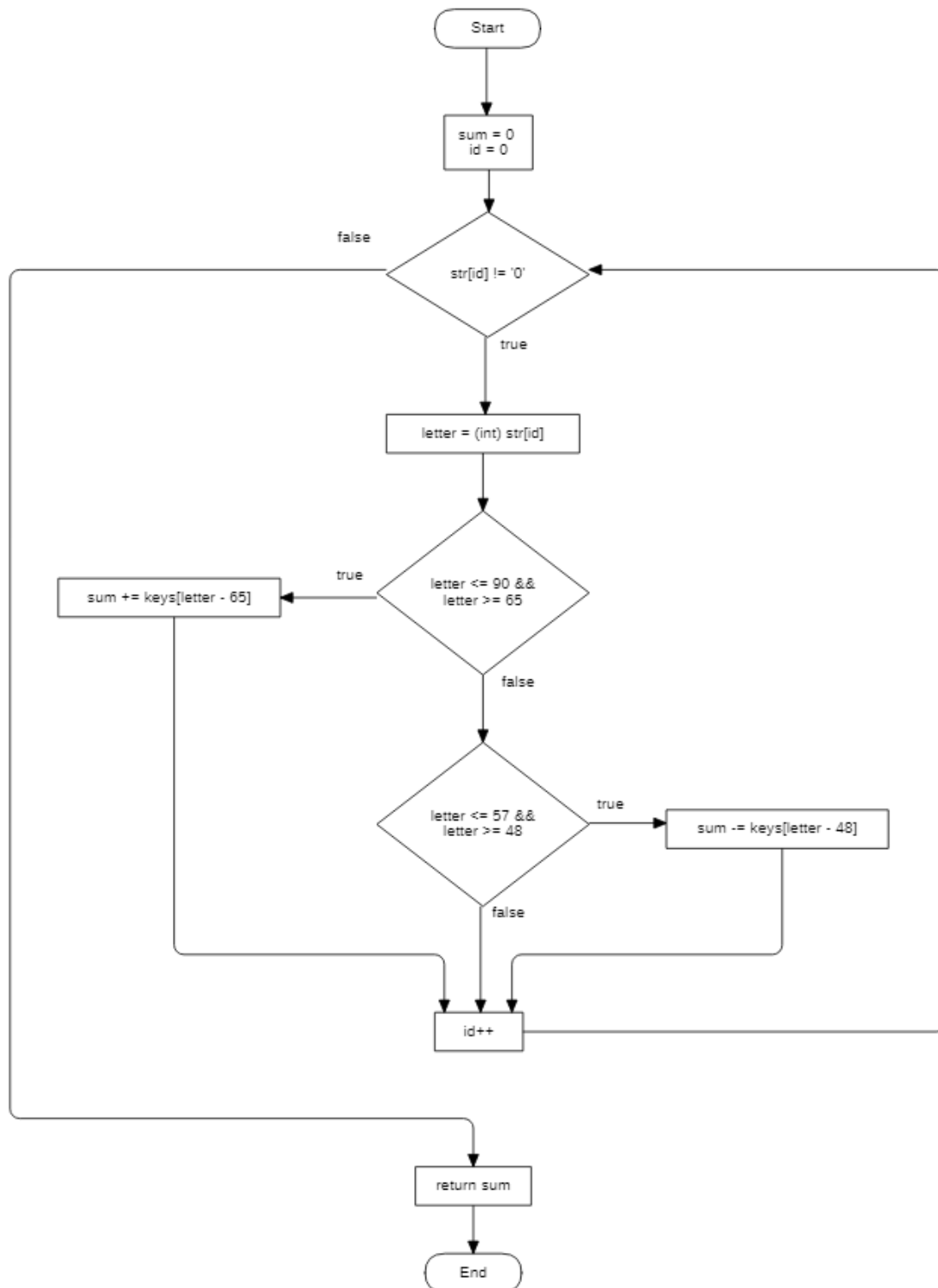
☒ Load Application at Startup ☒ Run to main()

STM32F401RE
STM32F401RETx

Software Component	Sel.	Variant	Version
Board Support		32F469DISCOVERY	1.0.0
CMSIS			
CORE	<input checked="" type="checkbox"/>		5.4.0
DSP	<input type="checkbox"/>	Source	1.8.0
NN Lib	<input type="checkbox"/>		1.3.0
RTOS (API)			1.0.0
RTOS2 (API)			2.1.3
CMSIS Driver			
Compiler		ARM Compiler	1.6.0
Event Recorder	<input type="checkbox"/>	DAP	1.4.0
I/O			
File	<input type="checkbox"/>	File System	1.2.0
STDERR	<input type="checkbox"/>	Breakpoint	1.2.0
STDIN	<input type="checkbox"/>	Breakpoint	1.2.0
STDOUT	<input checked="" type="checkbox"/>	Breakpoint	1.2.0
TTY	<input type="checkbox"/>	Breakpoint	1.2.0
Device			
Startup	<input checked="" type="checkbox"/>		2.6.3
STM32Cube Framework (API)			1.0.0
STM32Cube HAL			
STM32Cube LL			
File System		MDK-Plus	6.13.8
Graphics		MDK-Plus	6.10.8
Graphics Display			
Network		MDK-Plus	7.14.0
USB		MDK-Plus	6.14.1

- Πριν αρχίσουμε το *debugging* έχουμε υπολογίσει με το χέρι το αποτέλεσμα που θα έπρεπε να βγάλει σαν σωστό η συνάρτησή μας για το αλφαριθμητικό "aA1d", το οποίο είναι $18 - 1 = 17$.
- 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 τιμή 0x00000011 που μεταφράζεται από 16-δικό σε 10-δικό στην τιμή 17, όπως περιμέναμε.

3 Διάγραμμα Ροής



Σχήμα 1: Διάγραμμα Ροής Αλγορίθμου