

LAB1 REPORT

ΓΚΟΥΜΑ ΒΑΣΙΛΙΚΗ 9755 , ΚΩΣΤΑΣ ΑΝΔΡΟΝΙΚΟΣ 9754

Main.c

Έχουμε αρχικά δύο πίνακες, τον `table[26] = {10, 42, ...}` που αποθηκεύουμε τους ακεραίους που αντιστοιχούν σε κάθε χαρακτήρα σύμφωνα με την εκφώνηση και τον `input[100]` για να γίνει η αποθήκευση του string που δίνει ο χρήστης.

```
#include <stdio.h>
#include <stdlib.h>
#include <uart.h>

static char input[100] = {'\0'};

static uint8_t table[26] = {10, 42, 12, 21, 7, 5, 67, 48, 69, 2, 36, 3, 19, 1, 14, 51, 71, 8, 26, 54, 75, 15, 6, 59, 13, 25};
static char hash_string[9];

extern int hash(char *s, uint8_t *table);
extern int factorial(char *s);
```

Υπολογίζουμε το `hash_num` μέσω της `hash(char *s, uint8_t *t)` και τυπώνουμε το αποτέλεσμα. Για τον υπολογισμό του `factorial` μετατρέπουμε πρώτα τον ακέραιο σε string (`hash_string[9]`) για να μπορούμε να έχουμε πρόσβαση στο κάθε ψηφίο του `hash` αν για παράδειγμα έχει πάνω από 1 ψηφία. Έπειτα καλούμε την συνάρτηση `int factorial(char *s)`

```
int main()
{
    printf("enter string : ");
    //
    scanf("%s", input);
    // CHECK THE INPUT STRING WITH UART
    printf("\n");
    printf("given string = %s", input);
    printf("\n");
    uart_init(115200);
    uart_enable();
    uart_print("string = ");
    uart_print(input);
    //

    int hash_num = 0;
    hash_num = hash(input, table);

    printf("hash number = %d\n", hash_num);

    // convert hash_number to string
    sprintf(hash_string, "%d", hash_num);

    printf("factorial = %d", factorial(hash_string));

    return 0;
}
```

int hash(char *s, uint8_t *t) :

Code Explanation in C code

Διατρέχουμε τον πίνακα από χαρακτήρες μέσω της `while(s[i] != '\0')` μέχρι να διατρέξουμε όλους τους χαρακτήρες. Επειδή οι πίνακες αποθηκεύονται σε θέσεις μνήμης του ενός byte όλοι οι counters που χρησιμοποιούνται για το πέρασμα της μνήμης αυξάνονται κατά 1. Ακολουθούν δύο if έλεγχοι όπου ελέγχουμε σε ποιο εύρος ανήκει ο χαρακτήρας. Αν ανήκει στο εύρος [a,z] πραγματοποιούμε την πράξη : `hashResult = hashResult + t[s[i] - 97]` . Στην ουσία, προσθέτουμε στο `hashResult` τον αριθμό που αντιστοιχεί στον συγκεκριμένο χαρακτήρα σύμφωνα με τον πίνακα της εκφώνησης. Οι αριθμοί 97 και 122 είναι οι αντίστοιχοι αριθμοί ASCII των χαρακτήρων a και z. Με αντίστοιχο τρόπο υλοποιείται και η else-if που ακολουθεί όπου πραγματοποιεί την πράξη : `hashResult -= s[i] - 48;`

```
int hash(char *s, uint8_t *t)
{
    int hashResult = 0;
    int i = 0;
    while(s[i] != '\0')
    {
        if(s[i] >= 97 && s[i] <= 122) // a - z
        {
            hashResult += t[s[i] - 97];
        }
        else if(s[i] >= 48 && s[i] <= 57)
        {
            hashResult -= s[i] - 48;
        }

        i++;
    }

    return hashResult;
}
```

Code Explanation in assembly code – assembly.s

Γνωρίζουμε ότι τα ορίσματα αντιστοιχούν στους καταχωρητές R0 και R1, καθώς και το return value της συνάρτησης είναι το περιεχόμενο του R0. LDRB R7, [R0, R6] // R7 = s[i] (1) Μέσω της (1) φορτώνουμε στον R7 το περιεχόμενο της διεύθυνσης μνήμης (R0 + R6). Επειδή, ο R0 έχει το address του string που λάβαμε ως όρισμα και το στον R6 είναι η τιμή του i, μπορούμε να έχουμε την i-οστή τιμή του πίνακα string. Η υλοποίηση των if-else statements πραγματοποιείται μέσω της εντολής CMP και των αντίστοιχων εντολών branch. Για παράδειγμα :

```
loop:
    LDRB R7, [R0, R6] // R7 = s[i]
    CMP   R7, #0x00    // compare s[i] with '\0'
    BEQ   endOfWhile   // if s[i] == '\0' go to endOfWhile
    CMP   R7, #0x61    // compare s[i] with 97
    BLT   elseIfLabel  // go to elseLabel if s[i] < 97
    CMP   R7, #0x7A    // compare s[i] with 122
    BGT   label        // if s[i] > 122 go to i++

    // if the code reach here we are inside if(s[i] >= 97 && s[i] <= 122)

    SUB   R8, R7, #0x61 // s[i] - 97
    LDRB   R9, [R1, R8] // t[s[i] - 97]
    ADD   R5, R5, R9    // hashResult += t[s[i] - 97]
    b     label        // go to i++

elseIfLabel:
    CMP   R7, #0x30    // compare R7 to 48
    BLT   label        // if s[i] < 48 go to i++
    CMP   R7, #0x39    // compare R7 to 57
    BGT   label        // if s[i] > 57 go to i++
    SUB   R10, R7, #48 // s[i] - 48;
    SUB   R5, R5, R10  // hashResult -= s[i] - 48;

label:
    ADD   R6, R6, #0x01 // i = i + 1
    b     loop         // go to loop
endOfWhile:
    MOV   R0, R5
    bx    lr

.fnend
```

int factorial(char *s):

Code Explanation in C code

Δεχόμαστε ως είσοδο το hash_number αλλά ως string, όποτε έχουμε πρόσβαση στο κάθε ψηφίο ξεχωριστά. Σε κάθε επανάληψη προσθέτουμε στο sum το i-οστό στοιχείο αφού το μετατρέψουμε από ASCII στο δεκαδικό σύστημα. Για παράδειγμα αν το όρισμα s είναι το "521", θα πάρουμε το 8, αφού : $sum = 0$

$sum = 0 + 5 = 5$

$sum = 5 + 2 = 7$

$sum = 7 + 1 = 8$

Σε περίπτωση που σε κάποιο iteration προκύψει διψήφιος αριθμός βρίσκουμε ξεχωριστά τα ψηφία του και τα προσθέτουμε δηλαδή σε C :

```
if(sum >= 10)
{
    // sum belongs to [10,18]
    // first digit is always 1
    sd = sum - 10; // calculate the second digit
    sum = 1 + sd;  // calculate the sum
}
i++;
```

Εφόσον, ο διψήφιος προκύπτει από το άθροισμα δύο μονοψήφιων το maximum που μπορούμε να έχουμε είναι το 18 (9+9) και ο ελάχιστος το 10. Άρα για να βρούμε ξεχωριστά τα δύο ψηφία προκειμένου να το προσθέσουμε για να καταλήξουμε σε μονοψήφιο παρατηρούμε ότι το 1ο ψηφίο είναι πάντα 1 και το 2ο προκύπτει από την πράξη $sd = sum - 10$ Άρα, καταλήγουμε στον τελικό μονοψήφιο αριθμό, του οποίου πρέπει να υπολογίσουμε το παραγοντικό με την παρακάτω υλοποίηση, έχοντας και οριακές συνθήκες : $0! = 1$ και το παραγοντικό αρνητικού αριθμού επιστρέφει 0.

```
if(sum < 0)
{
    factorial = 0;
}
// we will calculate the factorial of the sum
while(sum > 1){
    factorial *= sum; // (1*n)(n-1)....
    sum -= 1;
}
```

Code Explanation in assembly code – assembly.s

Αρχικά φορτώνουμε στον R1 το i-οστό στοιχείο του πίνακα hash_string που έχουμε δώσει ως όρισμα. Χρησιμοποιώντας δηλαδή τον μετρητή R4 μπορούμε να διατρέξουμε όλο τον πίνακα μέσω της εντολής LDRB όπως φαίνεται παρακάτω:

```
LDRB    R1, [R0, R4]    // Load to R1 the value on the address R0+R4 //
CMP     R1, #0
BEQ     factorial_calculation
```

Αφού υλοποιούμε βήμα προς βήμα τις εντολές C σε assembly με χρήση CMP και branch εντολές, φορτώνουμε τον R5 που έχει αποθηκεύσει το παραγοντικό στον R0 που τελικά είναι και το return value της συνάρτησης.

```
loop_f:
    LDRB    R1, [R0, R4]    // Load to R1 the value on the address R0+R4 //
    CMP     R1, #0
    BEQ     factorial_calculation
    SUB     R1, #48
    ADD     R2, R2, R1    // sum += s[i] - 48 ;
    CMP     R2, #10
    BLT     label_f
    SUB     R3, R2, #10    // sd = sum - 10;
    ADD     R2, R3, #1    // sum = 1 + sd; // calculate the sum

label_f:
    ADD     R4, R4, #1
    B       loop_f

factorial_calculation:
    LDRB    R6, [R0]
    CMP     R6, #45    // compare sum with '-'
    BNE     label_greater_or_equal_than_zero

    // if(sum < 0) //
    MOV     R5, #0
    B       return_label

label_greater_or_equal_than_zero:
    CMP     R2, #0    // compare sum with 0
    BNE     loop_is_greater_than_zero
    // if(sum = 0) //
    MOV     R5, #1
    B       return_label

loop_is_greater_than_zero:
    CMP     R2, #1
    BEQ     return_label
    MUL     R5, R5, R2
    SUB     R2, R2, #1
    B       loop_is_greater_than_zero

return_label:
    MOV     R0, R5
    BX     LR

.fnend
```

Κρίσιμο είναι το σημείο που ελέγχουμε αρνητικό αριθμό! Επειδή ως είσοδο έχουμε string αν είναι αρνητικός θα αποθηκευτεί το - στο 1ο στοιχείο του πίνακα το οποίο είναι αποθηκευμένο στον R6. Πχ το -12 είναι αποθηκευμένο ως {'-', '1', '2'}.

```
factorial_calculation:
    LDRB    R6, [R0]
    CMP     R6, #45    // compare sum with '-'
    BNE     label_greater_or_equal_than_zero

    // if(sum < 0) //
    MOV     R5, #0
    B       return_label
```

Testing

- Χρήση breakpoints για να παρατηρήσουμε step by step πως τρέχουν οι εντολές σε συνδυασμό με το window που φαίνονται οι τιμές των registers και φυσικά μέσω και των εκτυπώσεων στο Debug (printf) Viewer. Χρήσιμο επίσης ήταν και η χρήση των watch παραθύρων.
- Γενικά ήταν πιο εύκολο πρώτα να υλοποιηθεί ο κώδικας σε C και έπειτα η μετατροπή του σε ARM assembly αρχιτεκτονική.

Challenges

- Δεν μπορούσαμε να καταλάβουμε πώς διοχετεύουμε τον πίνακα ως όρισμα
- Μετατροπή ASCII
- Πώς θα βρούμε ξεχωριστά το κάθε ψηφίο για να τα προσθέσουμε μεταξύ τους
- Μετατροπή C σε Assembly

RESULTS

EXAMPLE 1:

COM6

string = abc

Send

Autoscroll Show timestamp Carriage return 115200 baud Clear output

```
22 // scanf("%s", input);
23
CMP R2, #10
BLT label_f
SUB R3, R2, #10 // sd
ADD R2, R3, #1 // sum
label_f:
ADD R4, R4, #1
B label_f
// factorial calculation:
LDRB R6, [R0]
CMP R6, #45 // compare
BNE label_greater_or_equal_1
// if (sum < 0) //
MOV R5, #0
B return_label
// greater_or_equal_than_zero:
CMP R2, #0 // compare
BNE loop_is_greater_than_zero
// if (sum = 0) //
MOV R5, #1
B return_label
// is greater than zero:
```

Debug (printf) Viewer

enter string :
given string = abc
hash number = 64
factorial = 1

EXAMPLE 2:

COM6

string = 12345

Send

Autoscroll Show timestamp Carriage return 115200 baud Clear output

```
23 scanf("%s", input);
180
BX R0
ENDP
Dummy Exception Handlers (in:
_Handler PROC
EXPORT NMI_Handler
B .
ENDP
dFault_Handler\
PROC
EXPORT HardFault_Handler
B .
ENDP
Manage_Handler\
PROC
EXPORT MemManage_Handler
B .
ENDP
onfiguration Wizard /
```

Debug (printf) Viewer

enter string :
given string = 12345
hash number = -15
factorial = 0