

H19Y

Μικροεπεξεργαστές & Εφαρμογές

Προαιρετική Εργασία

ΔΠΘ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών



Παναγιώτης Ρήγας
56841

panariga@ee.duth.gr

Υπεύθυνος Καθηγητής Γ. Συρακούλης

Περιεχόμενα

Περιεχόμενα.....	1
Εισαγωγή	2
Θεωρητική Ανάλυση Εργασίας	3
Πρόγραμμα	6
Σχεδιασμός Αλγόριθμου	6
Υλοποίηση Κώδικα.....	10
Ανάλυση αποτελεσμάτων	15
Κωδικοποίηση-Coder	15
Αποκωδικοποίηση-Decoder.....	17
Υπολογιστική Μηχανή-MyFun	18
Προβλήματα και Διορθώσεις.....	23
Προβλήματα.....	23
Διορθώσεις.....	24
Βιβλιογραφία	26

Εισαγωγή

Στόχος της παρούσας εργασίας είναι η δημιουργία προγράμματος σε assembly στο Keil Uvision για τον ARM επεξεργαστή, με σκοπό την εκτέλεση βασικών μαθηματικών πράξεων δύο αριθμών χρησιμοποιώντας πάνελ αφής για την εισαγωγή αριθμών και επιλογή της πράξης.

Έτσι, αρχικά έγινε ανάλυση της εκφώνησης του κώδικα και αποτύπωση διάφορων προβλημάτων που προέκυψαν. Εκεί, μετά από συνεννόηση με τον αρμόδιο καθηγητή και τους διδακτορικούς δόθηκαν κατευθύνσεις και ιδέες για το πώς να συνεχιστή η εργασία. Έπειτα δημιουργήθηκε διάγραμμα ροής για τον κώδικα με βάση την ανάλυση που έγινε στην εργασία.

Στη συνέχεια, έγινε ο κώδικας, προσομοιώθηκε και αναλύθηκαν τα αποτελέσματά του. Τέλος, αναλύθηκαν τα προβλήματα αλλά και διορθώσεις που πραγματοποιήθηκαν στην εργασία. Στην εγγραφή της χρησιμοποιήθηκε βιβλιογραφία η οποία υπάρχει στο τέλος του παρόντος έγγραφου.

Θεωρητική Ανάλυση Εργασίας

Ο στόχος της εργασίας είναι η δημιουργία προγράμματος ηλεκτρολόγησης αριθμών, και επιλογής βασικών πράξεων σε assembly. Ουσιαστικά φτιάχνουμε ένα απλό κομπιουτεράκι. Το πάνελ φαίνεται στην παρακάτω εικόνα:

		Στήλες			
		0	0	1	0
Γραμμές	0	0	1	2	3
	1	4	5	6	7
	0	8	9	A	B
	0	C	D	E	F

Εικόνα 3.1: Πάνελ επαφής. Πηγή: e-class H19Y

Ο χρήστης θα εισάγει τους αριθμούς των 8 bit, από 0 έως F. Όταν πατάει ένα κουμπί, παράδειγμα το 6, θα ανάβει η 3^η στήλη και η 2^η γραμμή. Έτσι θα δημιουργούνται δύο λέξεις 4 bit ή μια των 8. Στη συγκεκριμένη περίπτωση θα έχουμε στην γραμμή την κωδική λέξη 0100 και στην στήλη την 0010. Αυτά θα δημιουργούν τελικά την λέξη 01000010. Έτσι μπορούμε να πούμε ότι ισχύει:

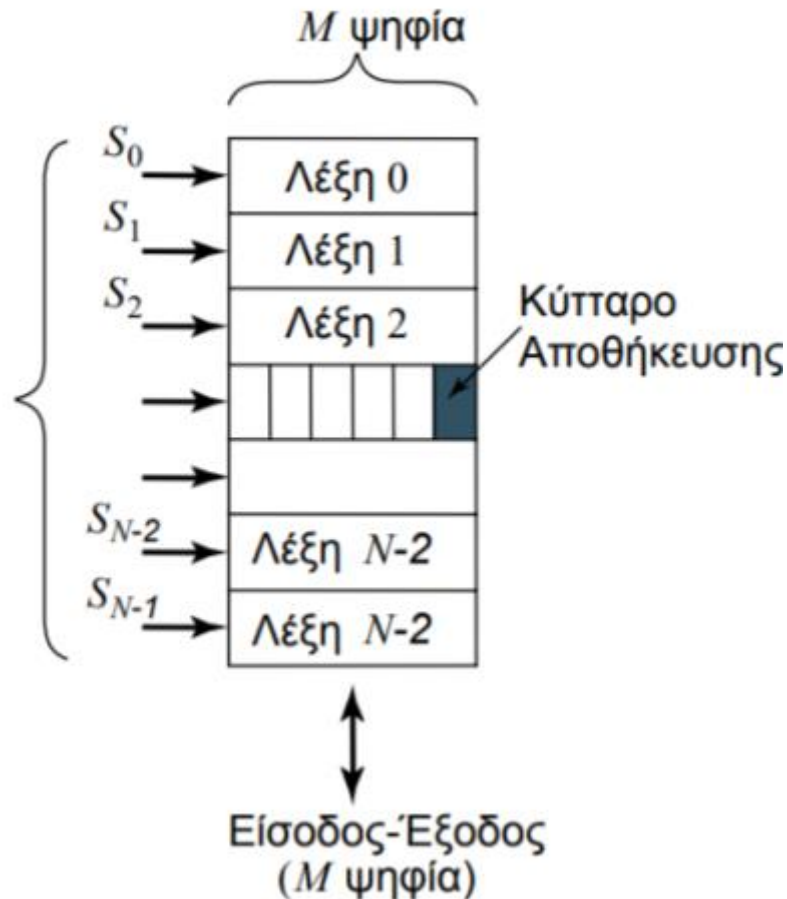
$$(06)_{hex} = row \times column = 0100 \times 0010 = 01000010$$

Ο παραπάνω αριθμός κωδικοποιείται με βάση τα παραπάνω και μεταδίδεται. Μόλις φτάσει στην μνήμη αποθηκεύεται και αποκωδικοποιείται. Εκεί καλείται ο αριθμός (06)_{hex} από έναν πίνακα και χρησιμοποιείται. Επιπλέον ο χρήστης θα επιλέγει την πράξη που θέλει να κάνει. Συγκεκριμένα θα έχει 4 επιλογές:

1. Πρόσθεση
2. Αφαίρεση
3. Διαίρεση
4. Πολλαπλασιασμό

Κάθε μία από τις επιλογές θα αντιστοιχεί σε έναν 4bit κώδικα 1000 0100 0010 και 0001.

Σε αυτό το στάδιο κρίνεται χρήσιμο να γίνει αναφορά στην κωδικοποίηση. Αν αυτή γίνει με απλή αρχιτεκτονική, δηλαδή χωρίς, τότε το αποτέλεσμα θα είναι πως για N σειρές θα έχουμε N σήματα επιλογής.



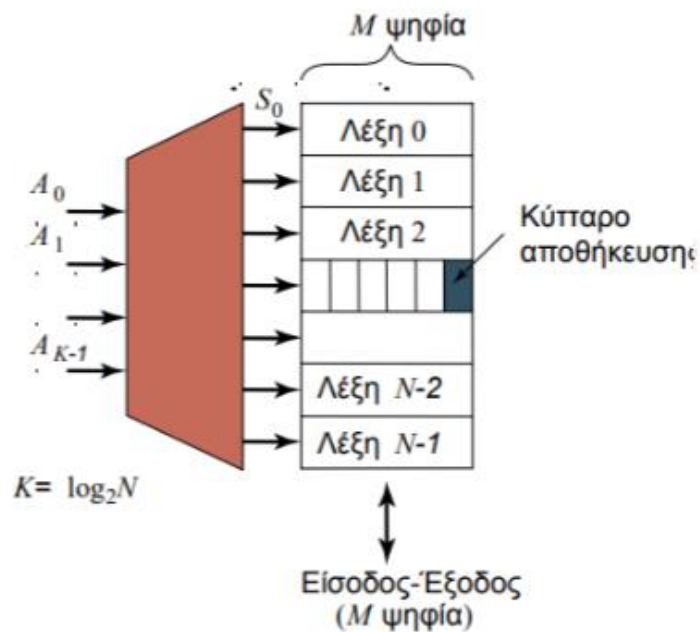
Εικόνα 3.2: Διαισθητική αρχιτεκτονικής μνήμης $N \times M$. Πηγή: e-class Μικροηλεκτρονική.

Εδώ το αρνητικό ξεκάθαρο· έχουμε πολλά σήματα επιλογής, πράγμα που σημαίνει πιο μεγάλα κυκλώματα και χειρότερη χρήση των πόρων μας γενικά. Σαν απάντηση σε αυτό έρχεται ένας απλό αποκωδικοποιητής. Από την εικόνα 3.3, βλέπουμε πως ουσιαστικά κωδικοποιούμε την επιλογή μας, έχοντας έτσι μειώσει αισθητά τα σήματα επιλογής, που πλέον είναι:

$$K = \log_2 N$$

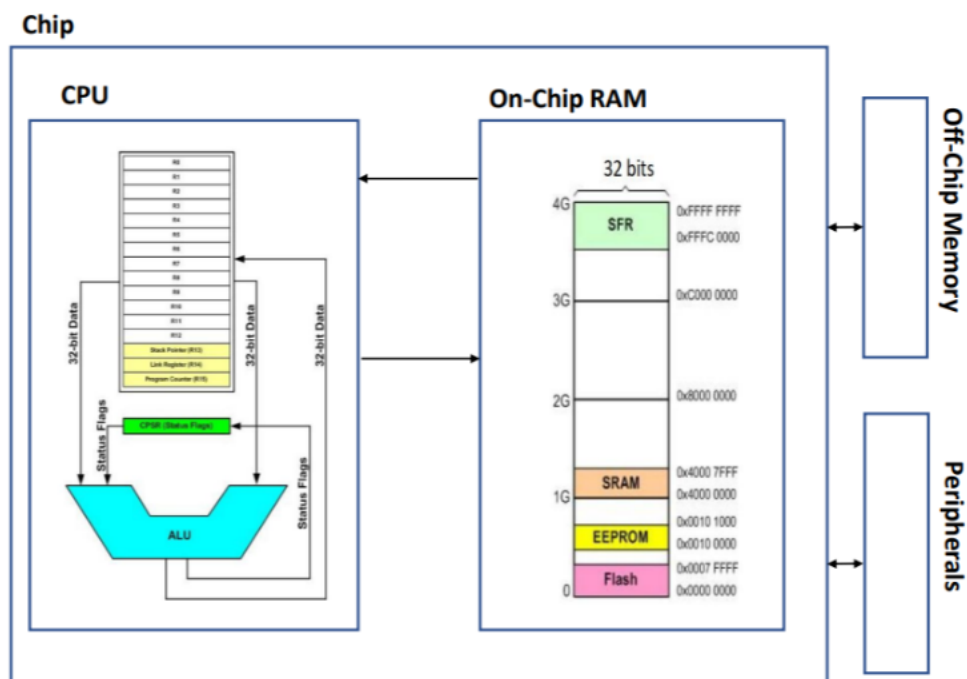
Όπου N οι λέξεις. Έτσι κάνουμε καλύτερη χρήση των πόρων μας, αν και κάνουμε λίγο πιο πολύπλοκο το σύστημά μας. Εάν εφαρμόσουμε την λογική αυτή και στις στήλες έχουμε ουσιαστικά μια δομή πίνακα όπου κάθε στοιχείο χαρακτηρίζεται από δύο μοναδικές λέξεις.

Μόλις ο χρήστης πραγματοποιήσει την εισαγωγή των αριθμών και των πράξεων, το σύστημα θα ανασύρει τα δεδομένα από την μνήμη στους καταχωριτές και θα πραγματοποιήσει την πράξη. Το αποτέλεσμα της θα αποθηκευτεί στη μνήμη. Βασικό θέμα σε αυτό είναι πως η εκφώνηση ζητάει τα δεδομένα να αποθηκευτούν στην διεύθυνση $0x103$ που όμως δεν μπορεί να γίνει, καθώς και σύμφωνα με τη θεωρία χρησιμοποιείται ήδη από το σύστημα και συγκεκριμένα από την Flash. Έτσι επιλέχθηκε ένα μέρος της μνήμης που είναι ελεύθερο για read/write εντολές.



Εικόνα 3.3: Αρχιτεκτονική επιλογής με αποκωδικοποιητή. Πηγή: e-class

ARM Chip



Εικόνα 3.4: Σχηματικό διάγραμμα ARM chip. Πηγή: e-class Μικροεπεξεργαστές.

Πρόγραμμα

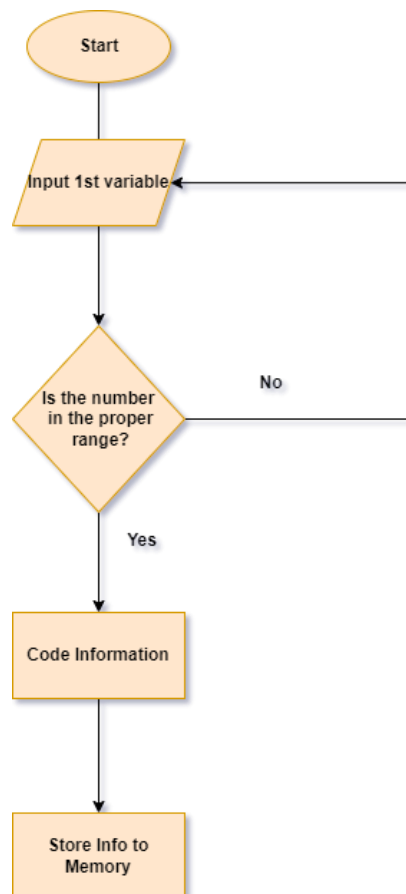
Ο κώδικας πραγματοποιήθηκε σε assembly KEIL. Η δημιουργία του χωρίστηκε σε τρία στάδια :

1. Σχεδιασμός Αλγόριθμου
2. Υλοποίηση
3. Ανάλυση Αποτελεσμάτων

Το πρώτο στάδιο είναι ο σχεδιασμός του αλγόριθμου. Εκεί κρίθηκε σημαντικό η δημιουργία διαγράμματος ροής για διευκόλυνση.

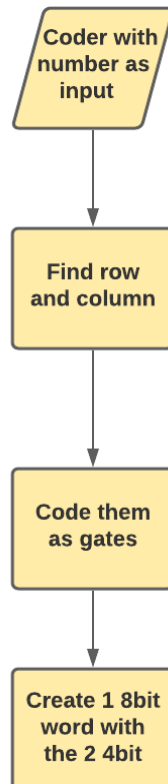
Σχεδιασμός Αλγόριθμου

Όπως αναφέραμε η διαδικασία σχεδιασμού αφορά τη δημιουργία διαγράμματος ροής. Αυτό φαίνεται παρακάτω:



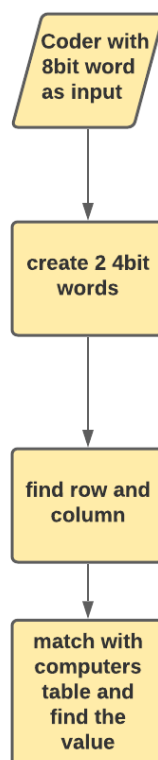
Εικόνα 4.1: Διάγραμμα ροής για εισαγωγή μεταβλητών

Βλέπουμε πως στην αρχή ζητείται η 1^η μεταβλητή, η οποία αφού ελεγχθεί από το σύστημα κωδικοποιείται και αποθηκεύεται στη μνήμη. Το ίδιο φυσικά συμβαίνει και με τις άλλες δύο, με κάποια διαφορά στην κωδικοποίηση της πράξης.



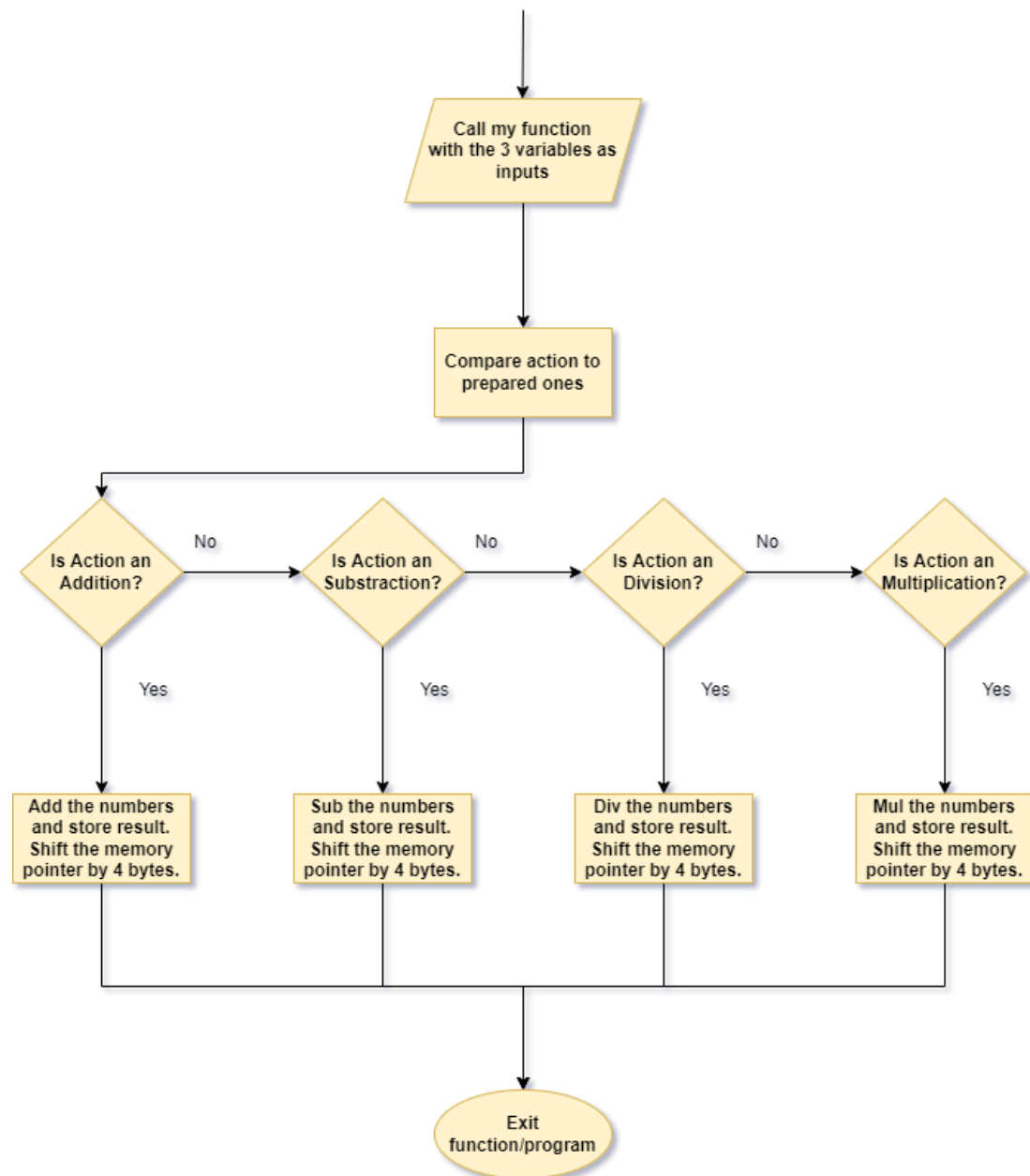
Εικόνα 4.2α: Διάγραμμα ροής για κωδικοποίηση

Αντίστοιχα έχουμε και την αποκωδικοποίηση της λέξης.



Εικόνα 4.2β Διάγραμμα ροής για αποκωδικοποίηση

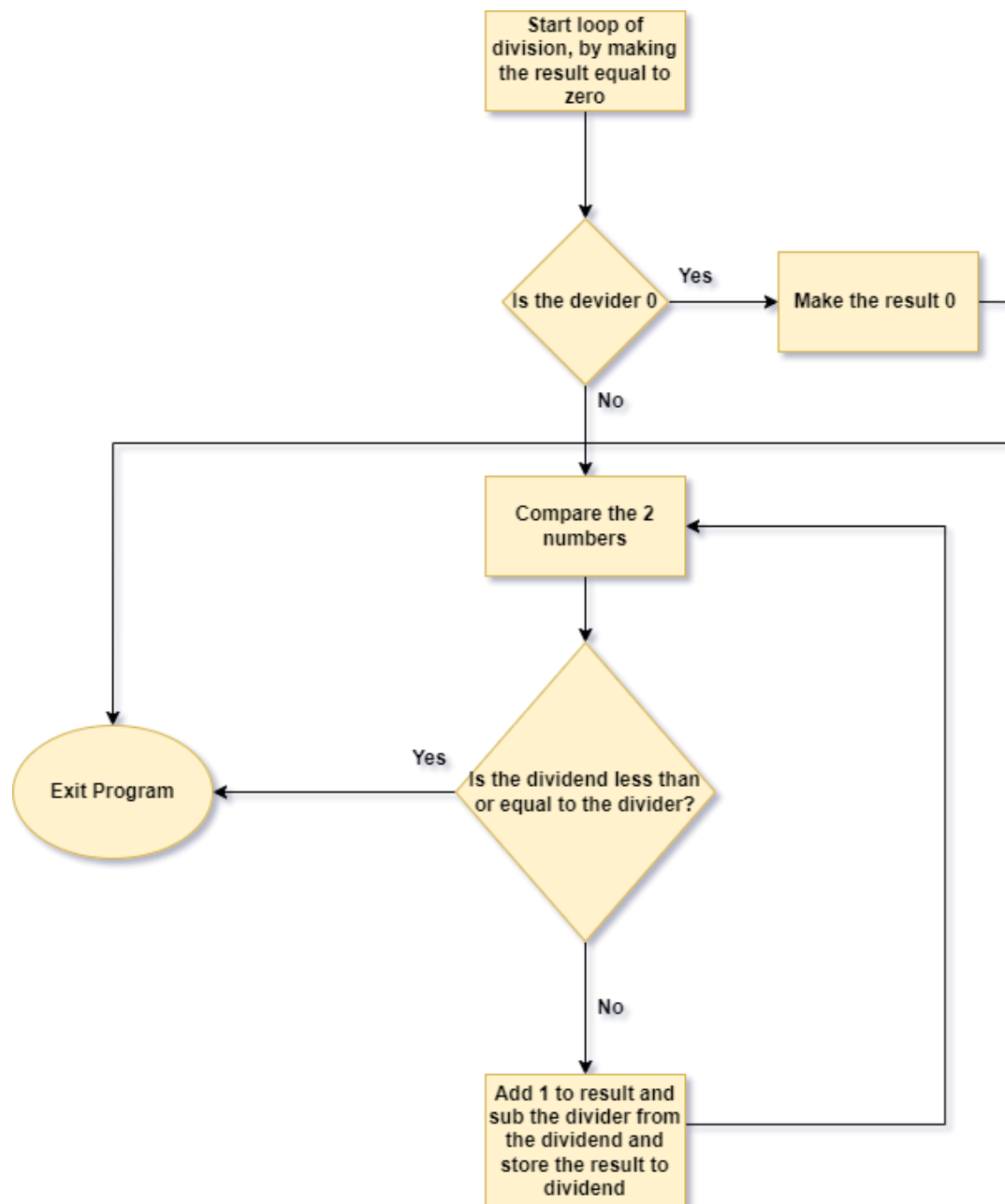
Στη συνέχεια:



Εικόνα 4.3: Διάγραμμα ροής συνάρτησης `myfun`.

Παρατηρούμε έπειτα το διάγραμμα της συνάρτησης των πράξεων. Αρχικά περνάμε τις τρεις μεταβλητές, τους δύο αριθμούς και την πράξη, ως μεταβλητές. Το πρόγραμμα σειριακά συγκρίνει την αποκωδικοποιημένη πράξη με τις αποθηκευμένες έως ότου βρει ποια είναι. Εδώ δεν υπάρχει θέμα καθώς στο όρισμα της πράξης υπάρχει δικλίδα ασφαλείας για την σωστή επιλογή της.¹ Θα πρέπει εδώ να αναφερθούμε στη διαίρεση. Ο συγκεκριμένος επεξεργαστής που προσομοιώνουμε δεν δύνάτε να κάνει την πράξη αυτή κατευθείαν. Έτσι πρέπει να υλοποιήσουμε έναν αλγόριθμο ειδικά για αυτήν την πράξη.

¹ Παρόλα αυτά στον κώδικα εισάγεται ένα flag για επιπλέον ασφάλεια.



Εικόνα 4.4: Ροϊκό διάγραμμα αλγόριθμου ευκλείδειας διαίρεσης.

Στόχος του αλγόριθμου διαίρεσης είναι να δώσει το πηλίκο και όχι το υπόλοιπο. Επιπλέον σε περίπτωση λάθους, όπως το να είναι ο διαιρέτης μηδέν, το σύστημα βγάζει αποτέλεσμα μηδέν και κλείνει το πρόγραμμα.

Τέλος θα αναφερθούμε στους αλγόριθμους κωδικοποίησης και αποκωδικοποίησης. Η λογική είναι πως ο χρήστης επιλέγει από το panel έναν αριθμό. Φυσικά ο υπολογιστής έχει δημιουργήσει την διεπαφή αλλά δεν ξέρει το «6» καθ' αυτό τι σημαίνει. Έτσι το κωδικοποιεί μέσω πυλών και κωδικοποιητών στήλης και γραμμής και στο τέλος δημιουργεί μία λέξη την οποία και μεταδίδει. Αυτή εν τέλει αποκωδικοποιείτε και αναπαράγει, αφού ταυτοποιηθεί, τον αριθμό που ζητάμε.

Υλοποίηση Κώδικα

Ξεκινάμε την Υλοποίηση με την δήλωση της προσομοίωσης και του των βασικών μεταβλητών.

```
/*-----  
Μικροεπεξεργαστές Project  
Panagiotis Rigas  
56841  
panariga@ee.duth.gr  
-----*/  
#include <MKL2524.H>  
// A is the table that the user sees and means nothing to the pc  
int A[4][4] = {{0x0, 0x1, 0x2, 0x3}, {0x4, 0x5, 0x6, 0x7}, {0x8, 0x9, 0xA, 0xB}, {0xC, 0xD, 0xE, 0xF}};  
// Apc is the table seen by the pc and to which will match the information  
int Apc[4][4] = {{0x0, 0x1, 0x2, 0x3}, {0x4, 0x5, 0x6, 0x7}, {0x8, 0x9, 0xA, 0xB}, {0xC, 0xD, 0xE,  
0xF}};  
// binary is presented in hex form since c has rejected any binary representation  
// b is the table used for action  
int B[4] = {0x0001, 0x0010, 0x0100, 0x1000};  
// c is for storing the results  
int C[2] = {0x0, 0x0};
```

Παρατηρούμε πως έχουμε 2 πίνακες, έναν όπου ουσιαστικά «βλέπει» ο χρήστης, και έναν όπου έχει ο υπολογιστής για να ταυτοποιεί τα αποκωδικοποιημένα στο τέλος δεδομένα. Επιπλέον έχουμε έναν πίνακα με τις πύλες, σε μορφή δεκαεξαδικού αντί για δυαδικού, επειδή η c δεν δέχεται δυαδική αναπαράσταση αριθμών. Τέλος έχουμε τον πίνακα C όπου αναπαριστούμε τα αποτελέσματά μας.

```
int main(void)  
{  
    int n1, n2, a;  
    int *num1, *num2, *num3;  
    int *p, *pdec;  
    // assign pointers  
    num1 = &n1;  
    num2 = &n2;  
    num3 = &a;  
    p = &A[0][0];  
    pdec = &Apc[0][0];  
    // input values  
    /*-----  
    printf("\nHello!, this programs accepts three numbers!\n1. The first number,\n2. The action to be  
    calculated,\n3. The second number.);  
    printf("Please enter the first number: ");  
    scanf("%x", &n1);  
    printf("\n Provide an action. \n4 for addition,\n3 for subtraction\n2 for multiplication,\n1 for  
    division\n");  
    scanf("%d", &a);  
    printf("\n Provide the second number: ") ;  
    scanf("%x", &n2)  
    -----*/  
    n1 = 6;  
    n2 = 2;  
    a = 2 ;  
    // code value  
    coder(num1,p,C,B);  
    // num1 is now coded  
    // and transmitted  
    // them decoded  
    decoder(C, B, C, pdec);  
    // and stored  
    n1 = C[0];  
    //call my fun  
    myFun(num1, num2, num3);  
    while(1);  
}
```

Εδώ βλέπουμε την main function του προγράμματός μας. Ξεκινάμε με την δήλωση των βασικών μεταβλητών και τον δεικτών(pointers) που αναθέτουμε σε κάθε μία από αυτές. Επιπλέον αναθέτουμε δείκτες και στους πίνακες (2d) καθώς κρίθηκε πιο απλό στην υλοποίηση του κώδικα. Έπειτα πηγαίνουμε και ενημερώνουμε τον χρήστη για το πρόγραμμα και ζητάμε να εισάγει μεταβλητές. Εδώ επίτηδες κρατήθηκε απλή η εισαγωγή καθώς δεν μπόρεσε να επεξεργαστεί από το σύστημα. Έτσι τυχόν flags για ενημέρωση εισαγωγής λάθος τιμή, ή μια switch case κρίθηκε υπερβολή και εκτός ουσίας. Στη συνέχεια καλούμε την κωδικοποίηση της εισαχθείσας τιμής, την μεταφέρουμε και την αποκωδικοποιούμε. Την αποθηκεύουμε και καλούμε την συνάρτηση.

```
__asm void coder(int* num, int* list, int* dst, int* code)
{
    LDR r5,[r0] ; load value
    MOVS r6,#0 ; flag column
    MOVS r7,#0 ; flag row

    // MOVS r5,#0
loopfor
    LDR r4,[r1] ; load pointer of array
    CMP r5,r4
    BEQ done // when done row and col will be loaded to r6r7
    ADDS r1,#4 ; go to next value
    ADDS r6,#1 ; add 1 to flag
    CMP r6,#4 ; check if flag exceeds col range
    BEQ goto1 ; if yes goto1
    B loopfor

goto1
    MOVS r6,#0 ; make col flag zero
    ADDS r7,#1 ; add 1 to row flag
    B loopfor

done
    // r0 r5 r1 r4 are now free for use
    //MOVS r0,#0
    //MOVS r1,#0
    MOVS r4,#0

    // loopa is for row
    // loopa is for 1st 4bit word
    // it matches the number of row to a word in B matrix
loopa
    LDR r5,[r3]
    CMP r7,#0 ; check if r7 is 0
    BEQ goto2
    ADDS r3,#4 ; go to next value
    ADDS r4,#4 ; add 4 to r4 so you can go back once finished
    SUBS r7,#1 ; decrease flag
    B loopa

goto2
    LSL r5,#16 ; here u l shift by 16 which is 4*4 because its hex and not binary. We are preparing to
create the 8bit word
    STR r5,[r2] ; store value
    SUBS r3,r4 ; the reason behind this is that we wanna use this pointer again
    // loopb is for column word
loopb
    LDR r5,[r3]
    CMP r6,#0 ; check if r6 is 0
    BEQ goto3
    ADDS r3,#4 ; go to next value
    SUBS r6,#1 ; decrease flag
    B loopb
    //store values
goto3
    LDR r4,[r2] ; load value to r4
    ADDS r4,r5 ; add the other 4bit
    STR r4,[r2] ; store back to r2
    LDR r4,=0x1FFFF04C
    BX lr ; exit subroutine
}
```

Η επόμενη συνάρτηση είναι αυτή της κωδικοποίησης. Αρχικά φορτώνουμε τους αριθμούς και αρχικοποιούμε τα flags που θα μετρήσουν σε πια γραμμή και ποια στήλη βρισκόμαστε. Ξεκινάμε έτσι την εύρεση αυτών. Ουσιαστικά παίρνουμε την τιμή και την συγκρίνουμε με τον 2d πίνακα. Εδώ το κόλπο είναι ότι ο 2d πίνακας είναι κατ' ουσία ένας μονοδιάστατος με 16 θέσεις. Έτσι απλά μετακινούμε την μεταβλητή-δείκτη. Όμως αυτό πρέπει να αποτυπωθεί στο σύστημα ως δύο μεταβλητές. Έτσι για κάθε 4 θέσεις στήλης, μηδενίζουμε και βάζουμε μια γραμμή. Συνεχίζουμε με το να τακτοποιήσουμε τις θέσεις αυτές σε 0000 σύμβολα. Αυτό γίνεται με τη βοήθεια του πίνακα B. Έτσι παίρνουμε την τιμή 0 και πάμε και παίρνουμε το B[0] και ούτε καθεξής. Τέλος ενώνουμε τις δυο 4-bit λέξεις κάνοντας shift την πρώτη και προσθέτοντας τις. Το αποτέλεσμα το αποθηκεύουμε στον πίνακα C.

```
__asm void decoder(int* codednum, int* Binary, int* deocder, int* Apc)
{
    LDR r4,[r0]; load value to be decoded
    MOVS r6,#0 ; make flags 0
    MOVS r7,#0
    //split the 2 numbers
    LSLS r4,#16 ; take 8bit word and make it 2 4-bits
    LSRS r4,#16
    //match with binary positioning
    //make 0001 into 0 0010 into 1 etc.
loop1
    LDR r5,[r1]
    CMP r4,r5
    BEQ goto4
    ADDS r1,#4 ; go to next value
    ADDS r6,#4 ; hold value so u can come back
    ADDS r7,#1 // r7 holds the row flag
    B loop1
goto4
    SUBS r1,r6
    MOVS r6,#0
    // we dont need r6 anymore
loop2
    LDR r4,[r0];
    LSRS r4,#16 ; create the second 4bit word
    LDR r5,[r1]

    CMP r4,r5
    BEQ goto5 ; if equal proceed
    ADDS r1,#4
    ADDS r6,#1 // r6 holds the column flag
    B loop2
goto5
    ADDS r6,#1 ; add 1 so no zeros exist. We do this to find the positioning in the 4x4 2d
array
    ADDS r7,#1
    MULS r7,r6,r7 ; find the positioning as if 2d is 1d
    MOVS r6,#4
    MULS r7,r6,r7 ; multiplie by 4 so you can shift. We can also lsls 2 to make it better
    ADDS r3,r7 ; add value to inital pointer
    LDR r6,[r3] ; get value
    ADDS r2,#4
    STR r6,[r2] ; store
    LDR r4,=0x1FFFF050
    BX lr ; exit
}
```

Συνεχίζουμε με την αποκωδικοποίηση με το αντίστροφο τρόπο. Ουσιαστικά παίρνουμε την 8bit λέξη, την σπάμε σε δύο 4bit και τις αντιστοιχούμε σε αριθμούς σειρών/στηλών με τη βοήθεια του πίνακα B. Στη συνέχεια με βάση αυτά βρίσκουμε την τιμή από τον πίνακα αντιστοίχισης του υπολογιστή. Αυτό το κάνουμε, πάλι

εκμεταλλεύοντας την ιδιότητα του 2d πίνακα , που ουσιαστικά είναι 1d.² Τέλος αποθηκεύουμε την τιμή με την οποία τακτοποιήσαμε την αποκωδικοποιημένη πληροφορία, την οποία δίνουμε στην μεταβλητή για εισαγωγή στην myfun.

```
__asm void myFun(int* x,int* y,int* z)
{
    LDR r3,[r0] ; load value pointed by r0 to r3, num1
    LDR r4,[r1] ; num2
    LDR r5,[r2] ; action
    LDR r7,=0x1ffff004 ; where results will be stored
    MOVS r6,#0 ; init result to zero

    //
    // first we need to address the operator which is stored into r2
    //
    CMP r5,#0
    BEQ division

    CMP r5,#1
    BEQ multiplication

    CMP r5,#2
    BEQ subtraction

    CMP r5,#3
    BEQ addition
    // in case of error in r5
    B err

division
    CMP r4,#0 ; is the divider 0?
    BEQ err
    CMP r3,r4 ; compare the diff
    BLT ready ; if r3 is less than r4 we are ready
    ADDS r6,r6,#1 ; add 1 to result
    SUBS r3,r3,r4 ; sub r3 = r3 - r4 r6 times so far
    B division

multiplication
    MOVS r6,r4
    MULS r6,r3,r6 ; multiplie r3xr4 and store to r6
    B ready

subtraction
    CMP r3,r4
    BLT err
    SUBS r6,r3,r4 ; sub r3-r4 and store to r6
    B ready

addition
    ADDS r6,r3,r4 ; add r3+r4 and store to r6
    B ready

ready
    STR r6,[r7] ;store result to value pointed by r7
    ADDS r7,#4 ; go to next value
    BX lr ; exit function

err
    MOVS r6,#0
    STR r6,[r7]
    BX lr
}

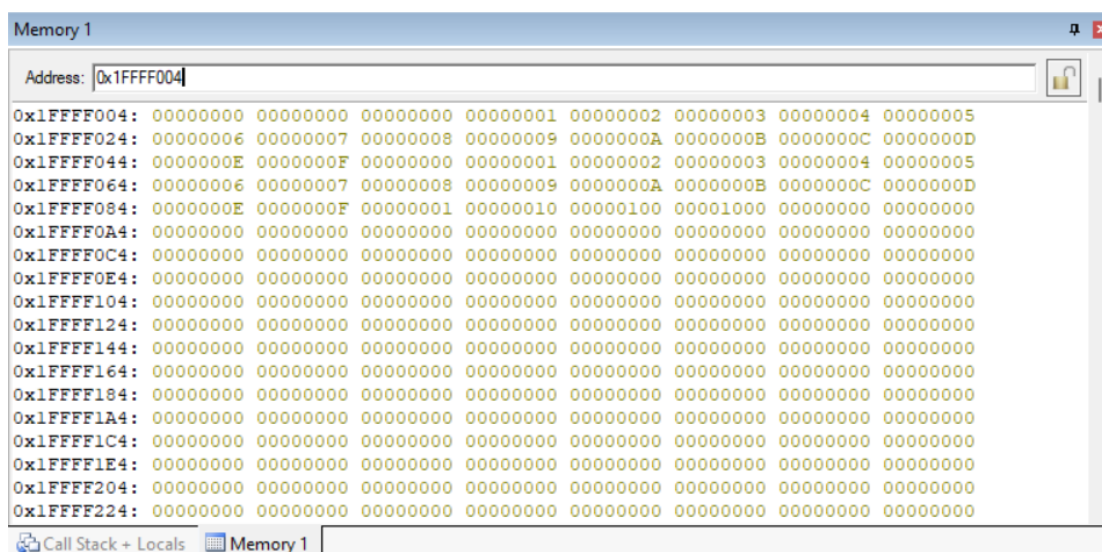
// Photo by Pawel Czerwinski on Unsplash
```

² Δες κεφάλαιο **διορθώσεις** για αλλαγές στην ταυτοποίηση της τιμής.

Η συνάρτηση-κομπιοτεράκι λειτουργεί ως εξής. Αρχικά εισάγουμε τις μεταβλητές τις οποίες και φορτώνουμε στον υπολογιστή. Τότε συγκρίνουμε την τιμή της action μεταβλητής για να δούμε ποια πράξη θα πραγματοποιήσουμε. Σε όλες τις πράξεις η διαδικασία είναι η ίδια. Κάνουμε την πράξη και αποθηκεύουμε το αποτέλεσμα. Η μόνη διαφορά έρχεται από την διαίρεση. Εκεί κάνουμε ευκλείδεια διαίρεση. Δηλαδή συγκρίνουμε τους δύο αριθμούς και κάνουμε αφαίρεση έως ότου το αποτέλεσμα είναι μικρότερο του μηδέν. Ταυτόχρονα προσθέτουμε στο πηλίκο +1 για κάθε αφαίρεση που κάνουμε. Έτσι στο τέλος παίρνουμε το πηλίκο. Σε περίπτωση σφάλματος βάζουμε σαν αποτέλεσμα μηδέν και βγαίνουμε από την συνάρτηση.

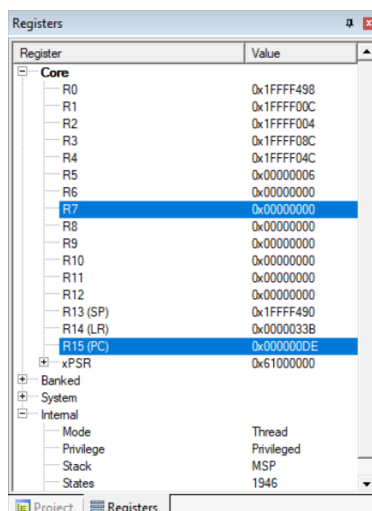
Ανάλυση αποτελεσμάτων

Η ανάλυση των αποτελεσμάτων θα γίνει σε τρία μέρη, κάθε ένα από τα οποία θα φορά τις τρεις βασικές συναρτήσεις. Έτσι θα ασχοληθούμε με την κωδικοποίηση, την αποκωδικοποίηση και την συνάρτηση πράξεων. Επιπλέον, αν και είναι πιο αναλυτική ή step-by-step ανάλυση(δηλαδή σε κάθε f11) αυτή κρίνεται πολύ μεγάλη δεδομένου του κώδικα(γύρω στις 225 σειρές). Έτσι θα γίνει αναφορά σε βασικά μέρη των συναρτήσεων κωδικοποίησης και αποκωδικοποίησης, ενώ εκτενέστερη θα γίνει στην συνάρτηση που αφορά τις πράξεις. Ξεκινάμε λοιπόν βλέποντας την αρχικοποιημένη θέση μνήμης. Εκεί φαίνονται, οι μνήμες που αφορούν C, A, Arc και B.

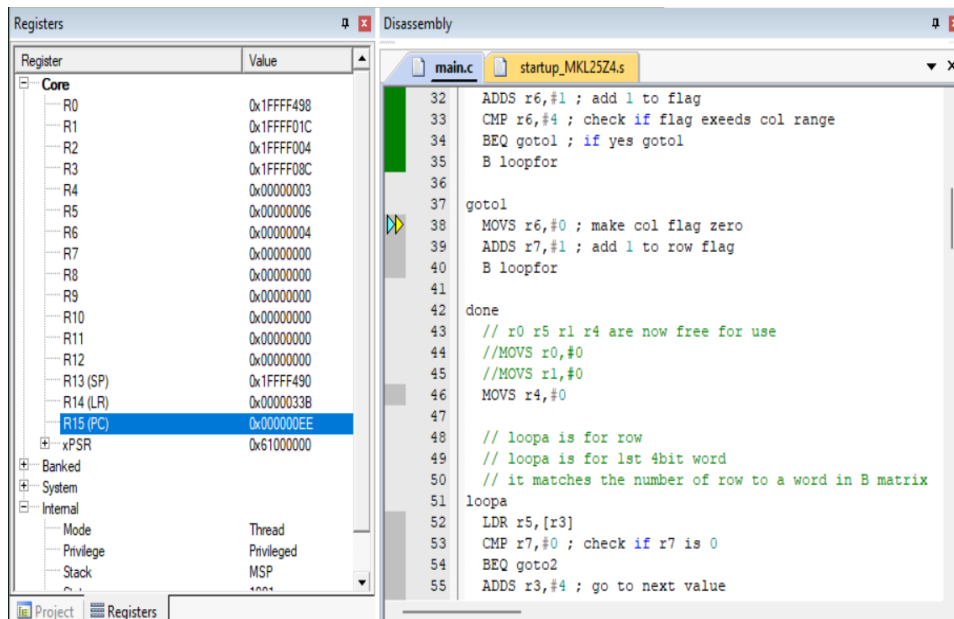


Κωδικοποίηση-Coder

Βλέπουμε αρχικά την ανάθεση της τιμής στον καταχωρητή r5. Επιπλέον οι r6, r7 γίνονται μηδέν, καθώς ετοιμάζονται για να χρησιμοποιηθούν ως counter για τα rows και columns.

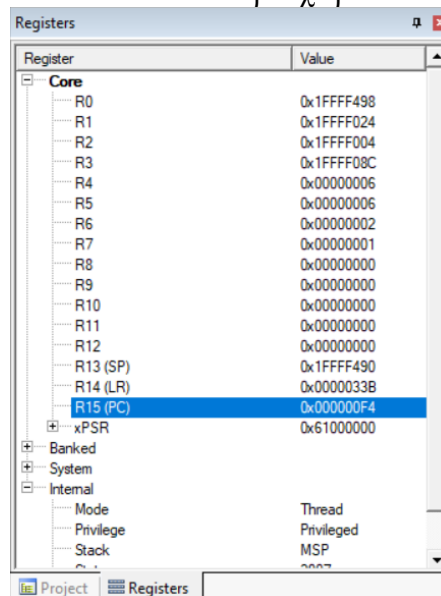


Εικόνα 6.1.1



Εικόνα 6.1.2

Στη συνέχεια βλέπουμε ένα στιγμιότυπο από την αποτύπωση του σημείου του αριθμού που πατήθηκε. Δηλαδή ο χρήστης πατάει το «6». Αυτό, μεταφράζεται σε 2^η σειρά και 3^η στήλη. Βλέπουμε πως ο R6 μετράει έως ότου φτάσει την τιμή. Κάθε φορά όμως που πιάνει 4, μηδενίζει και η γραμμή αυξάνεται κατά ένα. Ο κώδικας δείχνει στην goto1, όπου ακριβώς η παραπάνω διαδικασία θα λάβει χώρα.



Εικόνα 6.1.3

Στο τέλος βλέπουμε στην εικόνα 6.1.3 πως τα flags r6,r7 έχουν 2 και 1 αντιστοιχα. Αυτό, εφόσον μετράμε από το μηδέν και όχι το ένα αντιστοιχεί στην 3^η στήλη και 2^η γραμμή. Στη συνέχεια ξεκινώντας από τη γραμμή την αντιστοιχούμε με τιμή του πίνακα B. Έτσι το 1 θα γίνει 0010 το οποίο θα γίνει 0010 0000 ώστε να ενωθεί με τον κώδικα στήλης.

0x1FFFF004: 00100000

Εικόνα 6.1.4

Για τις στήλες γίνεται ακριβώς το ίδιο πράγμα. Στο τέλος οι δύο λέξεις προστίθενται και αποθηκεύονται στη μνήμη. Το αποτέλεσμα φαίνεται στην εικόνα 6.1.5.

0x1FFFF004: 00100100

Εικόνα 6.1.5

Αποκωδικοποίηση-Decoder

Η αποκωδικοποίηση ξεκινάει με την εισαγωγή της κωδικής λέξης στον καταχωρητή r4. Στη συνέχεια τον χωρίζουμε στις δύο λέξεις κάνοντας τις κατάλληλες μετατοπίσεις κάθε φορά. Έτσι σχηματίζεται η 1^η λέξη:

R4 0x00000100

Εικόνα 6.2.1

Η οποία αφορά τη στήλη. Αυτό θα μεταφραστεί σε αριθμό στήλης «2», δηλαδή στην 3^η στήλη. Αυτό γίνεται ταυτίζοντας την λέξη με τα περιεχόμενα της B. Στο τέλος παίρνουμε έναν counter που λέει σε ποιο στοιχείο του B ήμαστε. Το ίδιο κάνουμε και για την γραμμή.

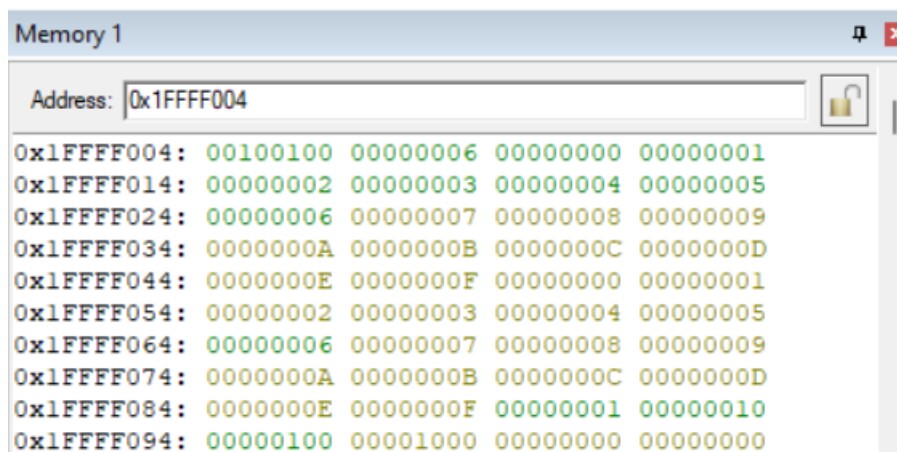
R6 0x00000001
R7 0x00000002

Εικόνα 6.2.2

Στη συνέχεια με βάση αυτά βρίσκουμε την θέση στον πίνακα του υπολογιστή και με αυτή το ψηφίο που πάτησε ο χρήστης. Αυτό γίνεται καθώς ο 2d πίνακας αριθμείται ως 1d. Έτσι τα ψηφία που ζητάμε είναι στην :

$$(r6 + 1) * (r7 + 1) = 2 * 3 = 6η \text{ θέση}$$

Το οποίο αντιστοιχεί στην τιμή «6». Στο τέλος αποθηκεύουμε το ψηφίο αυτό στη μνήμη.

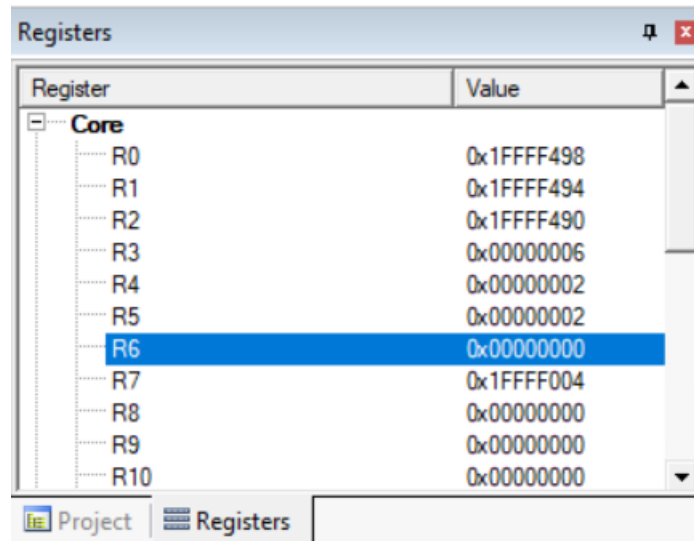


Εικόνα 6.2.3

Στην εικόνα 6.2.3 βλέπου με πράσινο τις διευθύνσεις που έχουμε προσπελάσει. Χαρακτηριστικά βλέπουμε το 6 από τον πίνακα του χρήστη, την προσπέλαση στον πίνακα του υπολογιστή, και στις τιμές του B. Τέλος η αποθήκευση γίνεται στο C.

Υπολογιστική Μηχανή-MyFun

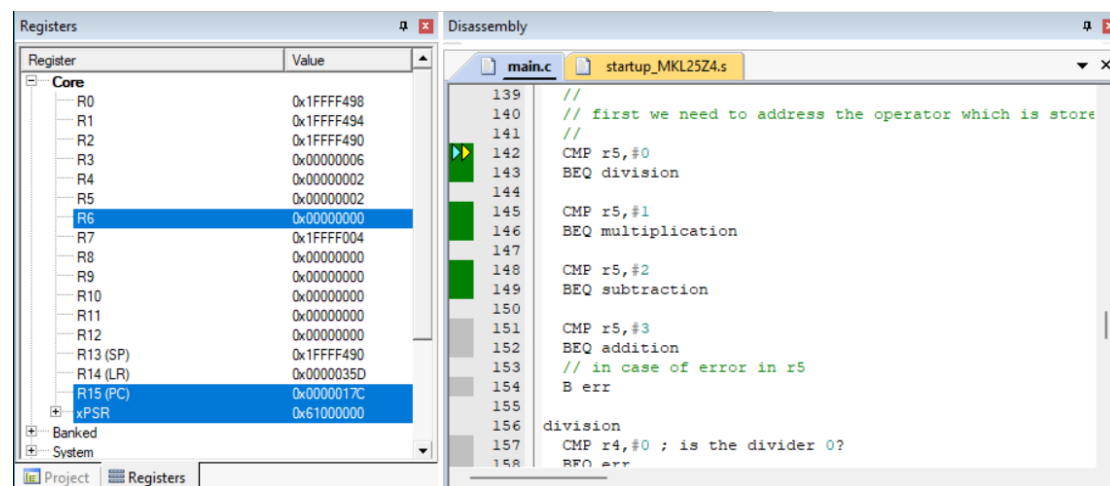
Η συνάρτηση ξεκινάει με την φόρτωση των μεταβλητών και αρχικοποίηση του καταχωρητή αποτελέσματος.



Register	Value
Core	
R0	0x1FFFF498
R1	0x1FFFF494
R2	0x1FFFF490
R3	0x00000006
R4	0x00000002
R5	0x00000002
R6	0x00000000
R7	0x1FFFF004
R8	0x00000000
R9	0x00000000
R10	0x00000000

Εικόνα 6.3.1

Στη συνέχεια συγκρίνουμε τον καταχωρητή που αποθηκεύεται η πράξη ώστε να δούμε τι πράξη θα κάνουμε:

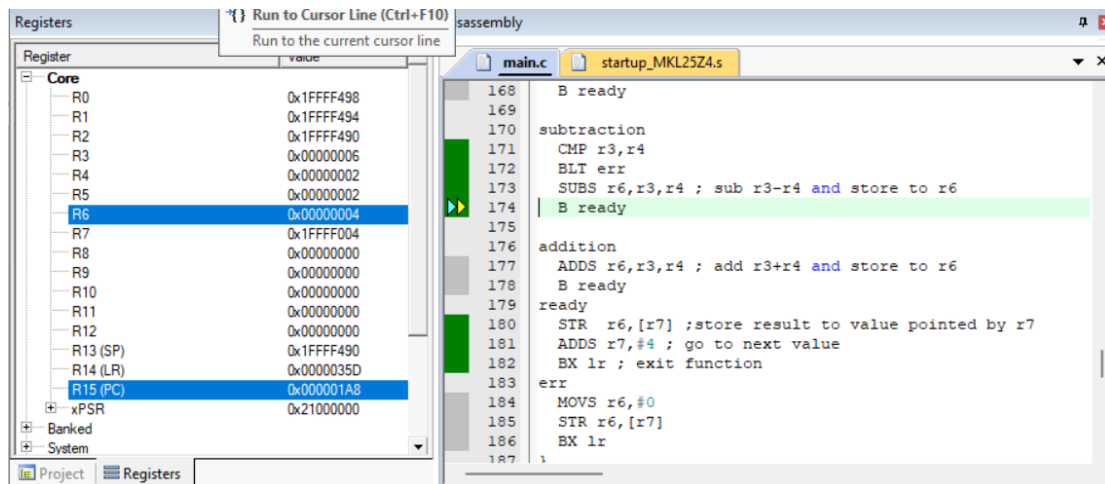


Register	Value
Core	
R0	0x1FFFF498
R1	0x1FFFF494
R2	0x1FFFF490
R3	0x00000006
R4	0x00000002
R5	0x00000002
R6	0x00000000
R7	0x1FFFF004
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x1FFFF490
R14 (LR)	0x0000035D
R15 (PC)	0x0000017C
xPSR	0x61000000

main.c	startup_MKL25Z4.s
139	//
140	// first we need to address the operator which is store
141	//
142	CMP r5,#0
143	BEQ division
144	
145	CMP r5,#1
146	BEQ multiplication
147	
148	CMP r5,#2
149	BEQ subtraction
150	
151	CMP r5,#3
152	BEQ addition
153	// in case of error in r5
154	B err
155	
156	division
157	CMP r4,#0 ; is the divider 0?
158	BEQ err

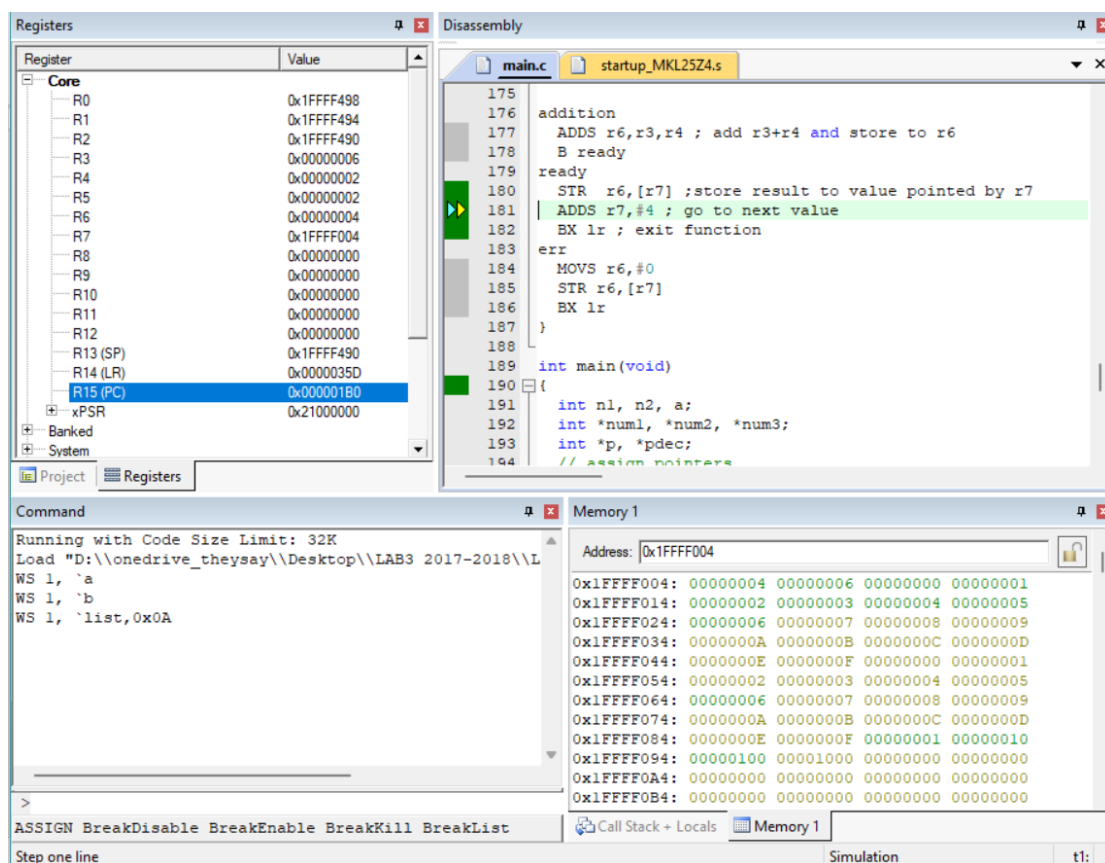
Εικόνα 6.3.2

Στο συγκεκριμένο παράδειγμα γίνεται αφαίρεση. Μόλις πάμε στο αντίστοιχο branch αρχικά συγκρίνουμε του δύο αριθμούς, έτσι ώστε να βεβαιωθούμε ότι το αποτέλεσμα θα είναι θετικό.



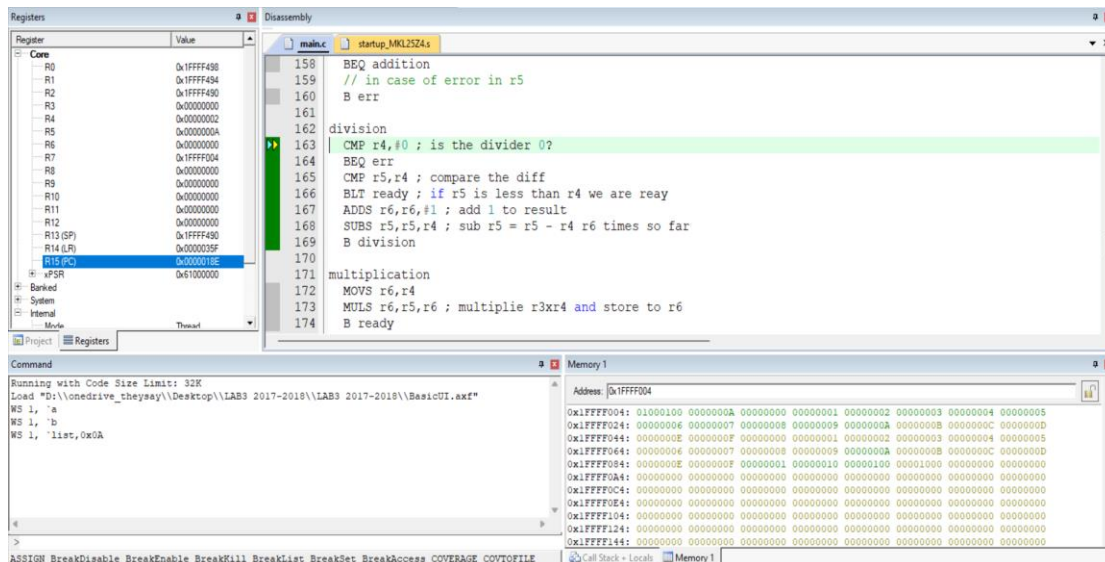
Εικόνα 6.3.3

Βλέπουμε εδώ ότι το αποτέλεσμα της πράξης, αποθηκεύεται στον καταχωρητή r6. Μόλις γίνει η πράξη πάμε στο branch ready.



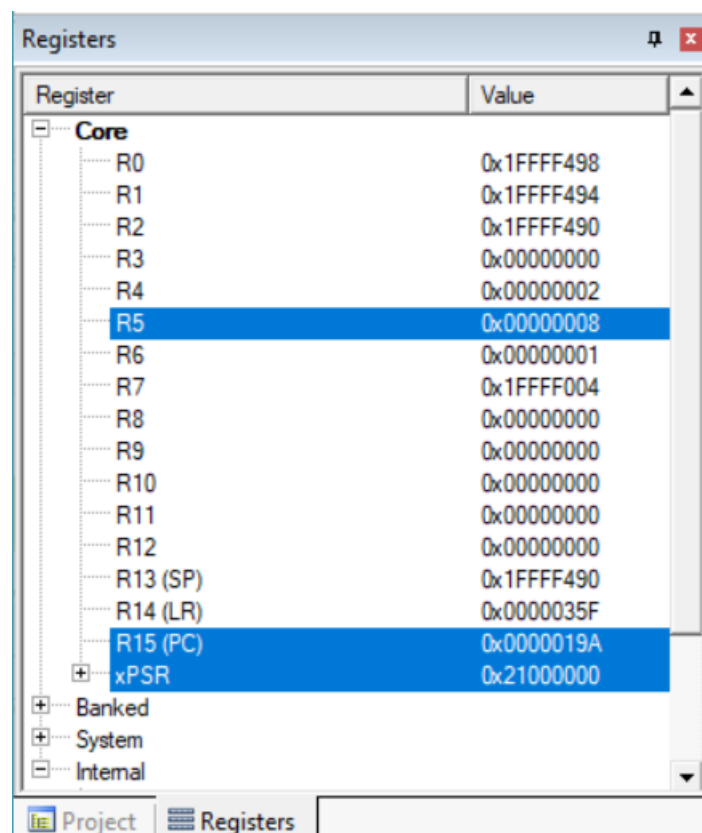
Εικόνα 6.3.4

Εκεί βλέπουμε πως το αποτέλεσμα αποθηκεύεται στην θέση μνήμης που δείχνει ο R7. Έτσι στο 0x1FFF004 έχουμε το αποτέλεσμα της αφαίρεσης, δηλαδή $6 - 2 = 4$. Στη συνέχεια θα γίνει μια ειδική αναφορά στην περίπτωση της διαίρεσης. Αυτό θα γίνει με τους αριθμούς A και 2. Το αποτέλεσμα έτσι αναμένουμε να είναι 5.



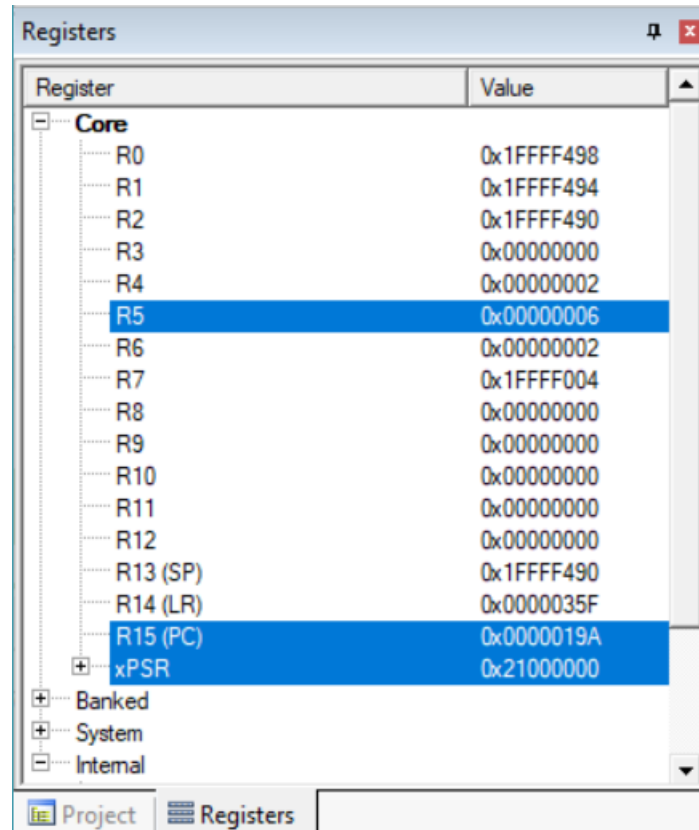
Εικόνα 6.3.5

Βλέπουμε την αρχή της διαίρεσης. Ξεκινάμε με την σύγκριση του r4 register με την τιμή 0. Αυτό γίνεται καθώς δεν θέλουμε να διαιρέσουμε έναν αριθμό με το 0. Σε περίπτωση που ισχύει κάτι τέτοιο πάμε στο branch err όπου μηδενίζουμε την έξοδο και τερματίζουμε το πρόγραμμα. Από εκεί και πέρα θα συγκρίνουμε τις τιμές r5 και r4. Αν η r4 είναι μεγαλύτερη έχουμε τελιώσει. Αυτό συμβαίνει διότι κάθε φορά αφαιρούμε την r4 από την r5 και κρατάμε έναν άσσο έως ότου ισχύσει. Έτσι ουσιαστικά έχουμε το πηλίκιο.



Εικόνα 6.3.6

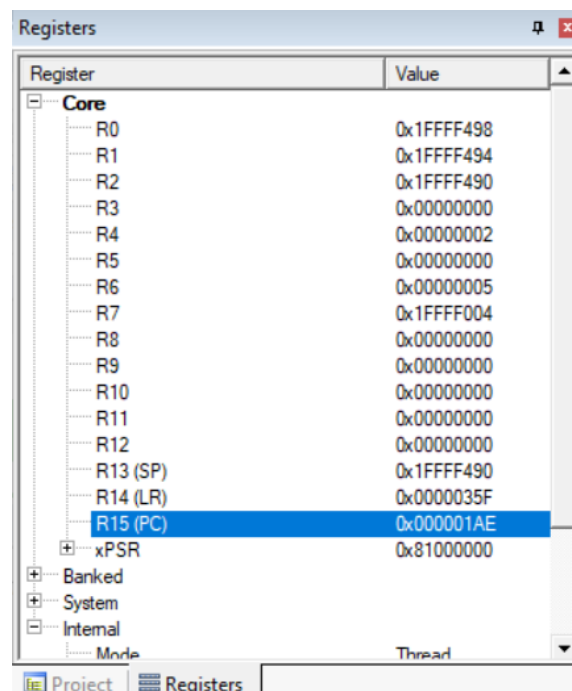
Στην εικόνα 6.3.5, φαίνεται αυτή η διαδικασία στους registers. Ο r6, όπου θα εμφανιστεί το αποτέλεσμα $A+1$, ενώ ο r5 έγινε από A, $A-2$ δηλαδή 8. Πάμε στον επόμενο κύκλο.



Register	Value
Core	
R0	0x1FFFF498
R1	0x1FFFF494
R2	0x1FFFF490
R3	0x00000000
R4	0x00000002
R5	0x00000006
R6	0x00000002
R7	0x1FFFF004
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x1FFFF490
R14 (LR)	0x0000035F
R15 (PC)	0x0000019A
xPSR	0x21000000
Banked	
System	
Internal	

Εικόνα 6.3.7

Μετά από έναν ακόμη κύκλο ο R5 έχει γίνει $8-2=A-2*2$, ενώ το αποτέλεσμα, που ουσιαστικά μετράει πόσες φορές αφαιρέσαμε έως τώρα, είναι δύο. Στο τέλος φθάνουμε στην εικόνα 6.3.8.



Register	Value
Core	
R0	0x1FFFF498
R1	0x1FFFF494
R2	0x1FFFF490
R3	0x00000000
R4	0x00000002
R5	0x00000000
R6	0x00000005
R7	0x1FFFF004
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x1FFFF490
R14 (LR)	0x0000035F
R15 (PC)	0x000001AE
xPSR	0x81000000
Banked	
System	
Internal	
Mode	

Εικόνα 6.3.8

Εκεί βλέπουμε πως το αποτέλεσμα έγινε 5. Επιπλέον ο r5 έγινε 0. Αυτό είναι φυσικό καθώς $A - 5 * 2 = 0$. Στο τέλος αυτό το αποτέλεσμα θα αποθηκευτεί στην μνήμη και το πρόγραμμα θα κλείσει.

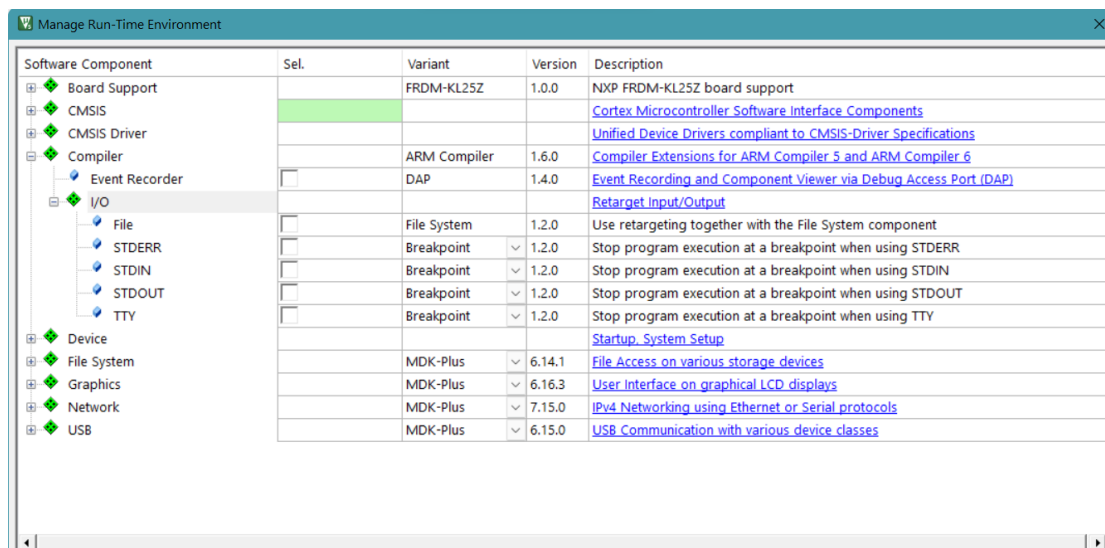
0x1FFFF004: 00000005

Εικόνα 6.3.9

Προβλήματα και Διορθώσεις

Προβλήματα

Βασικό πρόβλημα της εργασίας είναι η μη δυνατότητα χρήσης εξόδων, δηλαδή του πληκτρολογίου και της οθόνης για είσοδο και απεικόνιση αποτελεσμάτων. Συγκεκριμένα όταν εισάγουμε τις εντολές printf ή scanf, το πρόγραμμα πετάει error. Όταν πηγαίνουμε στο window run-time environment βλέπουμε πως τα stdin και stdout λειτουργούν ως breakpoint, δηλαδή το πρόγραμμα σταματάει όταν χρησιμοποιούνται.



Ως λύση σε αυτό είναι η δημιουργία προγραμμάτων και η επιλογή variant/user. Έτσι ο χρήστης επιλέγει να εκτελούνται αυτές. Το θέμα είναι πως δεν γνωρίζω τι πρέπει να περιέχουν αυτές, ή οποίες είναι γραμμένες σε c.

Ως εναλλακτική λύση δοκιμάστηκε η δημιουργία κώδικα assembly για αντικατάσταση των scanf και printf. Εδώ, ενώ γενικά υπάρχει αυτή η δυνατότητα, βρέθηκε πως στα συγκεκριμένα μοντέλα που προσομοιώνουμε στο εργαστήριο αυτό δεν είναι εφικτό, τουλάχιστον σε αρχικό επίπεδο. Έτσι ενώ παρουσιάζεται ο κώδικας των printf, scanf, αυτές δεν χρησιμοποιούνται.

Το δεύτερο πρόβλημα που παρουσιάστηκε αφορά την εισαγωγή μεταβλητών στις συναρτήσεις. Παρατηρήθηκε πως όταν καλείται μία συνάρτηση, η επόμενη που θα καλεστεί εμφανίζει λάθος κατανομημένες της μεταβλητές. Δηλαδή, αντί στο r0 να είναι η 1^η μεταβλητή, είναι μία άσχετη θέση μνήμης. Παρατηρώντας τον κώδικα αυτό οφείλεται στους καταχωρητές 4-7. Παρόλα αυτά δεν βρέθηκε λύση, αφού, η απευθείας ανάθεση δεν δουλεύει για διαφορετικές μεταβλητές. Έτσι οι συναρτήσεις της κωδικοποίησης και αντίστοιχα της αποκωδικοποίησής εισήχθησαν για την πρώτη μεταβλητή. Ως τελική σημείωση, στην όποια προσπάθεια επίλυσης του προβλήματος προκύπταν αρκετά άλλα errors.

Διορθώσεις

Εκτός από τα προβλήματα, κρίθηκε απαραίτητο να δημιουργηθεί ένα κεφάλαιο για τις διορθώσεις και συγκεκριμένα για την διόρθωση στην ταυτοποίηση της τιμής στην συνάρτηση decoder. Παρουσιάζεται η διόρθωση και αναλύεται ο λόγος που έγινε.

```
__asm void decoder(int* codednum, int* Binary, int* deocder, int* Apc)
{
    LDR r4,[r0]; load value to be decoded
    MOVS r6,#0 ; make flags 0
    MOVS r7,#0
    //split the 2 numbers
    LSLS r4,#16 ; take 8bit word and make it 2 4-bits
    LSRS r4,#16
    //match with binary positioning
    //make 0001 into 0 0010 into 1 etc.
loop1
    LDR r5,[r1]
    CMP r4,r5
    BEQ goto4
    ADDS r1,#4 ; go to next value
    ADDS r6,#4 ; hold value so u can come back
    ADDS r7,#1 // r7 holds the row flag
    B loop1
goto4
    SUBS r1,r6
    MOVS r6,#0
    // we dont need r6 anymore
loop2
    LDR r4,[r0];
    LSRS r4,#16 ; create the second 4bit word
    LDR r5,[r1]

    CMP r4,r5
    BEQ goto5 ; if equal procced
    ADDS r1,#4
    ADDS r6,#1 // r6 holds the column flag
    B loop2
goto5
    // this loop is left blank. It is not erased for the purpose of showing
    // the correction
    // ADDS r6,#1 ;add 1 so no zeros exist. We do this to find the positioning
    // in the 4x4 2d array
    // ADDS r7,#1 dont add since we need the last full rows
loop
    CMP r6,#0
    BEQ move
    ADDS r7,#4 ; add one row to r7
    SUBS r6,#1
    B loop
move
    LSLS r7,#2
    ADDS r3,r7 ; add value to inital pointer
    LDR r7,[r3] ; get value
    ADDS r2,#4
    STR r7,[r2] ; store
    LDR r4,=0xFFFF050
    BX lr ; exit
}
```

Συγκεκριμένα η goto5 έμεινε κενή πλέον. Αυτό συμβαίνει διότι η προηγούμενη θεώρηση που έλεγε κάνε τα flags+1 ώστε να μετράνε από 1 έως 4 και όχι από 0 έως 3 και πολλαπλασιάσέ τα είναι λάθος. Η νέα επιλογή πλέον είναι να πάρουμε το counter της γραμμής από 0 έως 3 και να προσθέσουμε 4, τόσες φορές, όσες είναι οι γραμμές. Όταν φθάσουμε στην γραμμή των στηλών, απλά θα προσθέσουμε την στήλη. Έτσι θα έχουμε το σωστό pointer. Μετά πολλαπλασιάζουμε *4, ώστε να πάμε από counter σε pointer και πάμε στην τιμή που θέλουμε.

Βιβλιογραφία

- [1] J. R. C., Μικροηλεκτρονική Σχεδίαση κυκλωμάτων.
- [2] W. Stallings, Οργάνωση & Αρχιτεκτονική Υπολογιστών, Εκδόσεις Τζιόλα.
- [3] Π. Θ. Πληροφορική, Εισαγωγή στους Αλγόριθμους και στα διαγράμματα ροής.
- [4] «<https://www.keil.com/dd/vtr/3694/3775.htm>,» [Ηλεκτρονικό].
- [5] «Input/output library,» [Ηλεκτρονικό].
- [6] «https://www.keil.com/support/man/docs/uv4/uv4_db_exp_pvar_io.htm,» [Ηλεκτρονικό].
- [7] «https://www.keil.com/support/man/docs/uv4/uv4_sm_di_overview.htm,» [Ηλεκτρονικό].
- [8] «https://www.keil.com/support/man/docs/armcc/armcc_pge1358787046598.htm,» [Ηλεκτρονικό].