ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ ΣΗΜΑΤΩΝ

Εργαστήριο 2

Ομάδα 09 – Group 1

ΕΠΩΝΥΜΟ	ΟΝΟΜΑ	ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ
ΔΑΣΟΥΛΑΣ	ΙΩΑΝΝΗΣ	1053711
ΔΟΥΡΔΟΥΝΑΣ	ΑΡΙΣΤΕΙΔΗΣ ΑΝΑΡΓΥΡΟΣ	1047398

Προγραμματισμός του C67x σε assembly(2/2)

Στόχος:

Στόχος της άσκησης είναι η συνέχιση της εκμάθησης εντολών assembly και της αρχιτεκτονικής του C6713 DSP.

Ασκήσεις:

Άσκηση 2.1

Φόρτωση σταθερών. Να γραφούν οι εντολές assembly για να γίνουν τα εξής:

(α) Φόρτωση της 16-bit σταθεράς 0xee13 στον A1.

(β) Φόρτωση της 32-bit σταθεράς 0xcdef5601 στο B0.

Πρόγραμμα:

```
            .def entry
            .text

entry:  MVKL  .S1 0xee13,A1
        MVKH  .S1 0xee13,A1
        MVKL  .S2 0xcdef5601 ,B0
        MVKH  .S2 0xcdef5601 ,B0
        IDLE
        .end
```

(α) Η 16-bit σταθερά 0xee13 φορτώνεται στα 4 LSB του A1 μέσω των εντολών MVKL, MVKH. Οι δύο αυτές εντολές προτιμώνται από την MVK διότι η MVK κάνει επέκταση προσήμου.

(β) Η 32-bit σταθερά 0xcdef5601 φορτώνεται στον A1 μέσω των εντολών MVKL, MVKH. Οι δύο αυτές εντολές προτιμώνται και πάλι από την MVK διότι η MVK κάνει επέκταση προσήμου.

Άσκηση 2.2

Οι ακόλουθες τιμές είναι αποθηκευμένες στις διευθύνσεις μνήμης:

100h fff4 7834h

104h 8f5c f34dh

108h cc2e fe87h

10ch f945 afc9h

110h 2faf fed9h

114h 4a7f a8f2h

118h ffad a1eah

Ισχύει A10 = 0000 0108h. Να βρεθεί το περιεχόμενο του A1 και A10 μετά από την εκτέλεση κάθε μιας από τις ακόλουθες εντολές.

1. LDH .D1 *A10, A1

2. LDB .D1 *A10, A1

3. LDW .D1 *A10, A1

4. LDW .D1 *-A10[1], A1

5. LDW .D1 *+A10[1], A1

6. LDW .D1 *+A10[2], A1

7. LDB .D1 *+A10[3], A1

8. LDW .D1 *++A10[1], A1

9. LDW .D1 *--A10[1], A1

10. LDW .D1 *A10--[1], A1

11. LDB .D1 *++A10[2], A1

12. LDB .D1 *--A10[1], A1

13. LDW .D1 *A10++[2], A1

1. LDH .D1 *A10, A1:

Φορτώνεται στον A1 η μισή λέξη που βρίσκεται στην διεύθυνση που δείχνει ο A10 και η υπόλοιπη λέξη συμπληρώνεται με ffff αφού γίνεται επέκταση προσήμου μιας και το fe87 είναι αρνητικό. Το A10 παραμένει ίδιο.
A1 = ffff fe87h, A10 = 108h.

2. LDB .D1 *A10, A1:

Φορτώνεται στον A1 το τελευταίο byte που βρίσκεται στην διεύθυνση που δείχνει ο A10 και η υπόλοιπη λέξη συμπληρώνεται με ffff ff αφού γίνεται επέκταση προσήμου μιας και το 87 είναι αρνητικό. Το A10 παραμένει ίδιο.
A1 = ffff ff87h, A10 = 108h.

3. LDW .D1 *A10, A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στην διεύθυνση που δείχνει ο A10. Το A10 παραμένει ίδιο.
A1 = cc2e fe87h, A10 = 108h.

4. LDW .D1 *-A10[1], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στο προηγούμενο μπλοκ μνήμης από την διεύθυνση που δείχνει ο A10. Το A10 παραμένει ίδιο.
A1 = 8f5c f34dh , A10 = 108h.

5. LDW .D1 *+A10[1], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στο επόμενο μπλοκ μνήμης από την διεύθυνση που δείχνει ο A10. Το A10 παραμένει ίδιο.
A1 = f945 afc9h , A10 = 108h.

6. LDW .D1 *+A10[2], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται δύο μπλοκ μνήμης μετά από την διεύθυνση που δείχνει ο A10. Το A10 παραμένει ίδιο.
A1 = 2faf fed9h, A10 = 108h.

7. LDB .D1   *+A10[3], A1:

Φορτώνεται στον A1 το τελευταίο byte της λέξης που βρίσκεται τρία μπλοκ μνήμης μετά από την διεύθυνση που δείχνει ο A10 και η υπόλοιπη λέξη συμπληρώνεται με ffff ff αφού γίνεται επέκταση προσήμου μιας και το f2 είναι αρνητικό.. Το A10

παραμένει ίδιο.
A1 = ffff fff2h, A10 = 108h.

8. LDW .D1 *++A10[1], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στο επόμενο μπλοκ μνήμης από την διεύθυνση που δείχνει ο A10. Το A10 παίρνει τη διεύθυνση του επόμενου μπλοκ μνήμης.
A1 = f945 afc9h , A10 = 10ch.

9. LDW .D1 *--A10[1], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στο προηγούμενο μπλοκ μνήμης από την διεύθυνση που δείχνει ο A10. Το A10 παίρνει τη διεύθυνση του προηγούμενου μπλοκ μνήμης.
A1 = 8f5c f34dh , A10 = 104h.

10. LDW .D1 *A10--[1], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στην διεύθυνση που δείχνει ο A10. Το A10 παίρνει τη διεύθυνση του προηγούμενου μπλοκ μνήμης.
A1 = cc2e fe87h, A10 = 104h.

11. LDB .D1 *++A10[2], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στο μεθεπόμενο μπλοκ μνήμης από την διεύθυνση που δείχνει ο A10. Το A10 παίρνει τη διεύθυνση του μεθεπόμενου μπλοκ μνήμης.
A1 =2faf fed9h, A10 = 110ch.

12. LDB .D1 *--A10[1], A1:

Φορτώνεται στον A1 το τελευταίο byte της λέξης που βρίσκεται στο προηγούμενο μπλοκ μνήμης από την διεύθυνση που δείχνει ο A10. Η υπόλοιπη λέξη συμπληρώνεται με 0000 00 αφού το 4d είναι θετικό (4 = 0100). Το A10 παίρνει τη διεύθυνση του προηγούμενου μπλοκ μνήμης.
A1 = 0000 004dh , A10 = 104h.

13. LDW .D1 *A10++[2], A1:

Φορτώνεται στον A1 η λέξη που βρίσκεται στην διεύθυνση που δείχνει ο A10. Το A10 παίρνει τη διεύθυνση του μεθεπόμενου μπλοκ μνήμης.
A1 = cc2e fe87h, A10 = 110h.

Άσκηση 2.3

Να γραφεί πρόγραμμα assembly για να αποθηκευτεί η 32-bit σταθερά fdae 12abh στη διεύθυνση μνήμης 0000 0fa2h . Επιπλέον, να επιβεβαιωθεί η ορθότητα της αποθήκευσης των δεδομένων, διαβάζοντας το περιεχόμενο της μνήμης στη συγκεκριμένη διεύθυνση.

Πρόγραμμα:

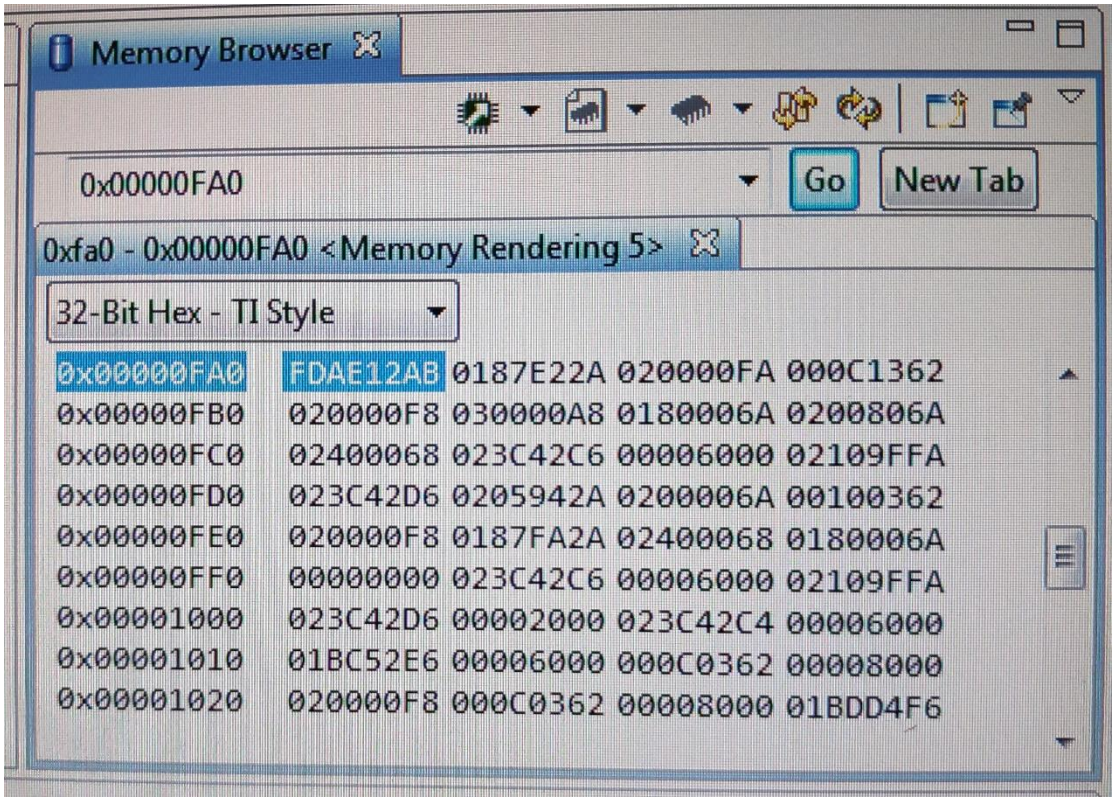
```
a           .set 0xfdae12ab
b           .set 0x00000fa2
            .def entry
            .text

entry:
            ZERO A1
            ZERO A2
            ZERO A3

            MVKL .S1 a,A1
            MVKH .S1 a,A1
            MVKL .S1 b,A2
            MVKH .S1 b,A2
            STW  .D1 A1,*A2
            LDW  .D1 *A2,A3
            NOP 4
            IDLE
            .end
```

Αρχικά, οι σταθερές 0xfdae12ab και 0x00000fa2 γίνονται set στις σταθερές a, b αντίστοιχα. Έπειτα, μηδενίζονται οι καταχωρητές A1, A2, A3. Με τις εντολές MVKL, MVKH φορτώνεται η τιμή που περιέχει το a στον A1 και το b στον A2. Ύστερα με την εντολή STW (store word) γίνεται φόρτωση της τιμής που έχει ο A1 στην διεύθυνση που δείχνει ο A2 και με την LDW (load word) φορτώνεται στον A3 η τιμή στην οποία δείχνει ο A2, δηλαδή η τιμή του A1. Η LDW απαιτεί 4 delay slots για να εκτελεστεί σωστά, εξού και χρησιμοποιείται η NOP 4 οδηγία.

Κατά την εκτέλεση του προγράμματος, παρατηρείται ότι η σταθερά 0xfdae12ab αποθηκεύεται στην διεύθυνση 0x00000fa0 και όχι στην διεύθυνση 0x00000fa2, όπως ήταν η εντολή διότι όταν γίνεται αποθήκευση μιας 32-bit λέξης, πρέπει να αποθηκευτεί σε διεύθυνση μνήμης που είναι ακέραια πολλαπλάσια του 4 (π.χ 0x00000fa0, 0x00000fa4, 0x00000fa8…) αφού απαιτούνται 4 bytes. Αυτό συμβαίνει για να μην δημιουργούνται προβλήματα στην μνήμη, σε πιο πολύπλοκες εντολές, οπότε by default ο επεξεργαστής τοποθετεί τις 32-bit λέξεις με αυτόν τον τρόπο και απαγορεύει την τοποθέτηση τους σε άλλη θέση μνήμης ακόμα κι αν ζητείται από το πρόγραμμα. Αντίστοιχα, τις 16-bit (halfword) τις τοποθετεί σε διευθύνσεις που είναι ακέραια πολλαπλάσια του 2 ενώ τις 8-bit (byte) σε οποιαδήποτε διεύθυνση.



Άσκηση 2.4

Ποια θα είναι η τιμή του A0 μετά από την εκτέλεση των ακόλουθων εντολών assembly (οι λειτουργικές μονάδες παραλείφθηκαν) ;

1. MVKL 0x80000000, A10
2. MVKH 0x80000000, A10
3. MVKL 0xfabc5ef8, A9
4. MVKH 0xfabc5ef8, A9
5. STW A9, *A10
6. LDB *+A10[2], A0

Ποια θα είναι η τιμή του A0 εάν το σύστημα χρησιμοποιεί το big endian τρόπο λειτουργίας μνήμης;

Η σταθερά 0xfabc5ef8 αποθηκεύεται στον A9 με τις MVKL, MVKH και με την εντολή STW ορίζεται η σταθερά του A10 (0x80000000) ως η διεύθυνσή της. Άρα, η 0xfabc5ef8 είναι αποθηκευμένη στην διεύθυνση 0x80000000 και με την LDB *+A10[2], A0 , παίρνουμε τα περιεχόμενα που βρίσκονται 2 bytes μετά από την

διεύθυνση A10. Όμως, ανάλογα με το endianness αλλάζει ο τρόπος που αναπαρίσταται ο αριθμός στις διευθύνσεις.

Little endian:

Στο little endian τρόπο, οι χαμηλότερες διευθύνσεις μνήμης περιέχουν το LSB μέρος των δεδομένων. Κατά συνέπεια, τα bytes που αποθηκεύονται στις τέσσερις διευθύνσεις byte θα είναι:

0x80000000 >> 0xf8

0x80000001 >> 0x5e

0x80000002 >> 0xbc

0x80000003 >> 0xfa

Άρα, με την εντολή LDB *+A10[2], A0 θα φορτωθεί στον καταχωρητή A0 η τιμή ffff ffbc. Αυτό συμβαίνει γιατί κάνει load τα bc από τη διεύθυνση 0x80000002 και ο αριθμός συμπληρώνεται με ffff ff μιας και γίνεται επέκταση προσήμου, αφού ο αριθμός b(hex) ξεκινάει με μονάδα (b = 1011).

Big endian:

Στο big endian τρόπο, οι χαμηλότερες διευθύνσεις μνήμης περιέχουν το MSB μέρος των δεδομένων. Κατά συνέπεια, τα bytes που αποθηκεύονται στις τέσσερις διευθύνσεις byte θα είναι:

0x80000000 >> 0xfa

0x80000001 >> 0xbc

0x80000002 >> 0x5e

0x80000003 >> 0xf8

Άρα, με την εντολή LDB *+A10[2], A0 θα φορτωθεί στον καταχωρητή A0 η τιμή 0000 005e. Αυτό συμβαίνει γιατί κάνει load τα 5e από τη διεύθυνση 0x80000002 και ο αριθμός συμπληρώνεται με 0000 00 αυτήν τη φορά αφού δεν γίνεται επέκταση προσήμου, μιας και ο αριθμός 5(hex) ξεκινάει με μηδέν (5 = 0101).

Άσκηση 2.5

Να γραφτεί ένα πρόγραμμα σε assembly που να υπολογίζει το:

(0000 ab33h + 0000 11abh - 0000 abcdh) * 0000 000ah

Πρόγραμμα:

```
a       .set  0x0000ab33
b       .set  0x000011ab
c       .set  0x0000abcd
d       .set  0x0000000a
        .def entry
```

```
        .text

entry:  ZERO A1
        ZERO A2
        ZERO A3
        ZERO A4
        MVKL a,A1
        MVKH a,A1
        MVKL b,A2
        MVKH b,A2
        MVKL c,A3
        MVKH c,A3
        MVKL d,A4
        MVKH d,A4
        ADD A1,A2,A1
        SUB A1,A3,A1
        MPYI A1,A4,A1
        NOP 8
        IDLE
        .end
```

Στην αρχή, φορτώνονται οι 4 αριθμοί που χρειάζονται στις σταθερές a, b, c, d με την set και μετά στους καταχωρητές A1, A2, A3, A4, αντίστοιχα, με MVKL, MVKH αφού πρώτα μηδενιστούν με την εντολή ZERO. Έπειτα, γίνεται η πρόσθεση, μετά η αφαίρεση και έπειτα ο τελικός πολλαπλασιασμός. Στον πολλαπλασιασμό, αντί για MPY, χρησιμοποιήθηκε η MPYI που εκτελεί πολλαπλασιασμό 32-bit x 32-bit λέξεων, αλλά απαιτεί nop 8 οδηγία μετά. Έτσι, αποφεύγεται ο κίνδυνος overflow που θα μπορούσε να υπάρχει αν χρησιμοποιούταν η MPY. Επίσης, θα μπορούσε να γίνει και με επιμεριστική ιδιότητα, δηλαδή να γίνουν πρώτα όλοι οι πολλαπλασιασμοί και μετά οι προσθέσεις και οι αφαιρέσεις ή και χωρίζοντας τις λέξεις σε halfwords και κάνοντας μετά 16-bit πράξεις.
Αποτέλεσμα = 0000 aaaa

Κατά την διάρκεια του εργαστηρίου ζητήθηκε να γίνει πρόγραμμα σε assembly που να υπολογίζει την παράσταση:
`(0x0000ef35 + 0x000033dc - 0x00001234) * 0x00000007`

Πρόγραμμα:

```
a       .set  0x0000ef35
b       .set  0x000033dc
c       .set  0x00001234
d       .set  0x00000007
        .def entry
        .text

entry:  ZERO A1
        ZERO A2
        ZERO A3
        ZERO A4
        MVKL a,A1
        MVKH a,A1
        MVKL b,A2
        MVKH b,A2
        MVKL c,A3
        MVKH c,A3
```

```
        MVKL d,A4
        MVKH d,A4
        ADD A1,A2,A1
        SUB A1,A3,A1
        MPYI A1,A4,A1
        NOP 8
        IDLE
        .end
```

Αποτέλεσμα = 7760b

Άσκηση 2.6

Να γραφεί ένα πρόγραμμα assembly που να υπολογίζει το παρακάτω άθροισμα με την εφαρμογή ενός απλού βρόχου: $y = \sum_1^{25} 2 * n - 1$.

Ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος:

```
    .def entry
            .text
entry:  MVKL 0x19,A1
            ZERO A2

loop:   MPY A1,2,A3
            NOP
            SUB A3,1,A3
            ADD A3,A2,A2
            SUB A1,1,A1

 [A1]   B loop

    IDLE
    .end
```

Αρχικά ορίζεται η τιμή 0x19h (25 στο δεκαδικό) στον register A1. Έπειτα , ορίζεται ο βρόχος επανάληψης μέσα στον οποίο γίνεται ο η πράξη του πολλαπλασιασμού του περιεχομένου του register A1 με το 2 και αποθηκεύεται στον register A3. Στη συνέχεια αφαιρείται από τον A3 το 1 και αποθηκεύεται η τιμή του στον A3. Μέσω της εντολής ADD A3,A2,A2 μεταφέρεται το αποτέλεσμα του A3 στον register A2 ,που λειτουργεί σαν αθροιστής ενώ μετά αφαιρείται 1 από τον register A1 που λειτουργεί σαν counter.

Τέλος έχουμε μία υπό συνθήκη διακλάδωση [A1] B loop η οποία εκτελεί την loop μέχρι η τιμή του counter να γίνει 0.

Άσκηση 2.7

Να δοθούν όλες οι λειτουργικές μονάδες που

μπορείτε να ορίσετε σε κάθε μια από αυτές τις εντολές:

(a) ADD .?? A0,A1,A2

(b) B .?? A1

(c) MVKL .?? 000023feh, B0

(d) LDW .?? *A10, A3

Κάθε εντολή έχει συγκεκριμένες λειτουργικές μονάδες που μπορεί να εκτελεσθεί. Μερικές εντολές μπορούν να εκτελεσθούν από διαφορετικές λειτουργικές μονάδες. Τα μονοπάτια προορισμού (που σημειώνονται ως dst) που εξέρχονται από τις .L1 .S1 .M1 και .D1 μονάδες, συνδέονται με το register file A. Αυτό σημαίνει ότι οποιαδήποτε εντολή που έχει προορισμό έναν από τους καταχωρητές A (δηλαδή, το αποτέλεσμα της λειτουργίας αποθηκεύεται σε έναν από τους καταχωρητές A), πρέπει να εκτελείται σε μία από αυτές τις 4 λειτουργικές μονάδες. Για τον ίδιο λόγο, εάν οι εντολές έχουν τους καταχωρητές B ως προορισμό, τότε οι μονάδες .L2 .S2 .M2 και .D2 πρέπει να χρησιμοποιηθούν. Με βάση το manual TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide οι παραπάνω εντολές μπορούν να εκτελεστούν από τις ακόλουθες λειτουργικές μονάδες.

(α) ADD .L1 / .S1 / .D1 A0,A1,A2

(b) B .S1 A1

(c) MVKL .S2 000023efh,B0

(d) LDW .D1 *A10,A3

Άσκηση 2.8

Ορισμένες εντολές μπορεί να απαιτούν να γίνουν πράξεις ανάμεσα σε δεδομένα από δύο διαφορετικούς καταχωρητές . Εάν τα δεδομένα έρχονται από έναν καταχωρητή A με προορισμό έναν καταχωρητή B έχουμε 1x διαγώνια πορεία , ενώ εάν τα δεδομένα έρχονται από έναν καταχωρητή B με προορισμό έναν καταχωρητή A έχουμε 2x διαγώνια.

(a) ADD .L2x / .D2x / .S2x B0,A1,B2 .Εδώ πρόκειται για 2x διαγώνια πορεία αφού η εντολή παίρνει δεδομένα και από A και από B καταχωρητή αλλά έχει ως καταχωρήτη προορισμού τον B2.
(b) MPY .M1x A1,B2,A4.1x διαγώνια πορεία αφού χρησιμοποιούνται δεδομένα από A και B καταχωρητές με καταχωρητή προορισμού τον A4.

Άσκηση 2.9

Να γραφεί πρόγραμμα assembly για τον υπολογισμό του $\sum_1^{12} a_n * x_n$, όπου a_n και x_n παίρνουν τις ακόλουθες τιμές: a[] = { 1, 2, 4, 7, 8, 6, 5, 3, 9, b, a, c },
x[] = { 3, e, 6, c, 5, a, 9, 8, a, 6 ,b,4}.

Ο κώδικας που αναπτύχθηκε για τον ερώτημα είναι ο ακόλουθος:

```
counter .set 12
consta .word  0x1, 0x2, 0x4, 0x7, 0x8, 0x6, 0x5, 0x3, 0x9, 0xb, 0xa, 0xc
constx .word  0x3, 0xe, 0x6, 0xc, 0x5, 0xa, 0x9, 0x8, 0xa, 0x6 ,0xb, 0x4

                .def entry
                .text

entry:
                MVKL counter,B0
                MVKH counter,B0
                MVKL consta,A1
                MVKH consta,A1
                MVKL constx,A2
                MVKH constx,A2

                ZERO A0

loop:           LDB *A1++[4], A3
                LDB *A2++[4], A4
                NOP 4
                MPY A3, A4, A5
                NOP
                ADD A5, A0, A0
                SUB B0,1,B0
                [B0]    B loop
```

```
        IDLE
        .end
```

Αρχικά , ορίζεται ο counter ο οποίος παίρνει τιμή ίση με τον αριθμό των στοιχείων που πρέπει να πολλαπλασιαστούν μεταξύ τους (12). Έπειτα δηλώνονται στην μνήμη ως λέξεις (consta .word, constx .word) τα στοιχεία των δύο πινάκων που πρέπει να πολλαπλασιαστούν καταλαμβάνοντας διαδοχικές θέσεις μνήμης. Μέσω της .word παρέχεται δηλαδή η δυνατότητα να προσπελαστούν τα στοιχεία κάθε σταθεράς προχωρώντας στην αμέσως επόμενη θέση μνήμης. Έπειτα, μέσω των εντολών MVKL, MVKH φορτώνονται οι διευθύνσεις των πρώτων στοιχείων της κάθε σταθεράς. Αφού μηδενιστεί ο register A0 ορίζεται μία επανάληψη μέσα στην οποία μέσω των εντολών LDB *A1++[4], A3 και LDB *A2++[4], A4, φορτώνονται στους A3 , A4 τα περιεχόμενα των διευθύνσεων που δείχνουν οι pointers *A1 και *A2 και έπειτα μέσω του τελεστή ++[4] οι pointers πάνε στην διεύθυνση του επόμενου στοιχείου. Έπειτα, μέσα στην επανάληψη γίνεται ο πολλαπλασιασμός των στοιχείων και το αποτέλεσμα του αποθηκεύεται μέσω της ADD A5, A0, A0 στον A0. Η SUB B0,1,B0 αφαιρεί 1 από τον counter σε κάθε επανάληψη. Η εντολή [B0] B loop δηλώνει μια υπό συνθήκη διακλάδωση η συνθήκη τερματισμού της οποίας δηλώνεται με το [B0] και λέει ότι η επανάληψη θα τελειώσει όταν ο register B0 πάρει την τιμή 0.

Άσκηση 3.6

Να γραφεί ένα πρόγραμμα σε assembly που να διαβάζει τη θέση των 3 τελευταίων διακοπτών (USER_SW 1,2,3) και ανοιγοκλείνει τα αντίστοιχα LEDs με βάση τη θέση των διακοπτών. Το 1ο led πρέπει να παραμένει πάντα κλειστό. Να τροποποιηθεί το πρόγραμμα έτσι ώστε η ΚΜΕ να ελέγχει τη θέση διακοπτών συνεχώς και να ενημερώνει τα LEDs. Ενώ το πρόγραμμά τρέχει, τα LEDs πρέπει να ανοιγοκλείνουν όταν αλλάζουν οι διακόπτες.

Το DSK board έχει 4 πράσινα LEDs (USER_LED 0, 1, 2, 3) που μπορούν να ελεγχθούν από την ΚΜΕ μέσω της μνήμης. Έχει επίσης έναν διακόπτη 4-θέσεων. Μπορείτε να διαβάσετε τους τρεις (USER_SW 0, 1, 2, 3) μέσω της διεύθυνσης μνήμης I/O. Η διεύθυνση μνήμης είναι η διεύθυνση 0x90080000h. Το περιεχόμενο της διεύθυνσης αυτής είναι ένας αριθμός μήκους 8 bit . Ο αριθμός αυτός μας δείχνει τις καταστάσεις των 4 διακοπτών του board καθώς και των 4 LED. Τα bit 0-3 αναφέρονται στην κατάσταση των LED (0-3). Το 0 σημαίνει ότι το LED είναι OFF και το 1 ότι το LED είναι ON. Αντίστοιχα , τα bit 4-7 αναφέρονται στις καταστάσεις των διακοπτών (1-3) με την διαφορά ότι 1 σημαίνει ότι ο διακόπτης είναι OFF και το 0 ότι ο διακόπτης είναι ON. Ο κώδικας που αναπτύχθηκες στο εργαστήριο είναι ο ακόλουθος:

```
a       .set 0x90080000
        .def entry
        .text
```

```
entry:      ZERO A2
            ZERO A3

            MVKL a,A2
            MVKH a,A2

loop:
            LDB *A2,A3
            NOP 4
            SHR A3,4,A3
            NOT A3,A3
            CLR A3,0,0,A3
            STB A3,*A2

            B loop
            NOP 5
            IDLE
            .end
```

Αρχικά , η μεταβλητή a ίση παίρνει την τιμή 0x90080000 (διεύθυνση του αριθμού που δείχνει τις καταστάσεις). Στη συνέχεια , μηδενίζονται οι 2 registers μέσω των MVKH και MVKL και έτσι φορτώνεται στον register A2 η a. Έπειτα , μέσα στον βρόχο επανάληψης loop με την χρήση της εντολής LDB φορτώνεται το περιεχόμενο της διεύθυνσης που υποδεικνύεται από τον pointer *A2. Το περιεχόμενο του A3 είναι ο 8-bit αριθμός που δείχνει τις καταστάσεις των διακοπτών και των LED. Προκειμένου να περαστούν οι καταστάσεις των διακοπτών στα LED χρησιμοποιείται η εντολή SHR A3,4,A3 η οποία κάνει shift δεξιά κατά 4 θέσεις τα bit του αριθμού. Έτσι , τα 4 MSB μεταφέρονται δεξιά και καταλαμβάνουν τις θέσεις 0-3 , και μεταφέρουν τις καταστάσεις τον διακοπτών στα LED . Επειδή όμως τα ON και OFF ορίζονται αντίθετα για τους διακόπτες από ότι στα LED χρησιμοποιείται την εντολή NOT A3,A3 η οποία κάνει την λογική πράξη NOT και τη ν αποθηκεύει στον A3. Επειδή ζητείται το 1^{ο} LED να είναι μονίμως κλειστό , χρησιμοποιείται η CLR A3,0,0,A3 η οποία μηδενίζει το 3^{ο} bit του αριθμού , το οποίο ελέγχει το τελευταίο LED. Τέλος, η τελική κατάσταση αποθηκεύεται στον A3 για να ενημερωθεί και να χρησιμοποιηθεί στην επόμενη επανάληψη.

Βιβλιογραφία:

- **TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide**
- **ΕΚΦΩΝΗΣΗ ΑΣΚΗΣΗΣ LAB-2 – ECLASS**

TMS320C6713 DATASHEET