ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ ΣΗΜΑΤΩΝ

Εργαστήριο 3

Ομάδα 09 – Group 1

ΕΠΩΝΥΜΟ	ΟΝΟΜΑ	ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ
ΔΑΣΟΥΛΑΣ	ΙΩΑΝΝΗΣ	1053711
ΔΟΥΡΔΟΥΝΑΣ	ΑΡΙΣΤΕΙΔΗΣ ΑΝΑΡΓΥΡΟΣ	1047398

Αιτήσεις διακοπών

Στόχος:

Η εξοικείωση με το μηχανισμό των interrupts για την οικογένεια επεξεργαστών C67x της TI καθώς και με τη χρήση μονάδων της CPU, όπως οι timers καθώς και με ορισμένα περιφερειακά που βρίσκονται επάνω στο DSK, τα οποία θα χειρισθούν με κατάλληλο τρόπο από τον κώδικα.

Ασκήσεις:

Άσκηση 3.1

Ποιες είναι οι τιμές που πρέπει να γράφουν στους IMH και IML για την ακόλουθη αρχικοποίηση και επιλογή των διακοπών;
INT4 ← TINT0
INT5 ← TINT1
INT6 ← XINT0
INT7 ← RINT0
INT8 ← XINT1
INT9 ← RINT1
Να ορισθούν και οι INT10 με INT15 αντιστοίχως. Να γραφεί ρουτίνα assembly που γράφει τις κατάλληλες τιμές στους IMH και IML.

Η C67x KME παρέχει συνολικά 14 διαφορετικές διακοπές (interrupts): δύο σταθερές (RESET και NMI) και 12 κατ' επιλογήν του χρήστη (INT4 - INT15). Επίσης υπάρχουν και δύο διακοπές (INT2 και INT3) που είναι κρατημένες για εσωτερική χρήση.

Οι RESET και NMI έχουν την υψηλότερη προτεραιότητα και ακολουθούν οι INT4 - INT15 κατ’ ελάσσονα σειρά στην προτεραιότητα. Μπορούν επίσης να εξυπηρετηθούν αιτήσεις όταν ήδη εξυπηρετείται αίτηση χαμηλότερης προτεραιότητας. Το αντίθετο δεν μπορεί να συμβεί. Ο ακόλουθος πίνακας (Πίνακας 1) παρουσιάζει τις διαθέσιμες πηγές διακοπών της KME C67x/C62x.

INTERRUPT SELECTOR VALUE (BINARY)	INTERRUPT EVENT	MODULE
00000	DSPINT	HPI
00001	TINT0	Timer 0
00010	TINT1	Timer 1
00011	SDINT	EMIF
00100	GPINT4†	GPIO
00101	GPINT5†	GPIO
00110	GPINT6†	GPIO
00111	GPINT7†	GPIO
01000	EDMAINT	EDMA
01001	EMUDTDMA	Emulation
01010	EMURTDXRX	Emulation
01011	EMURTDXTX	Emulation
01100	XINT0	McBSP0
01101	RINT0	McBSP0
01110	XINT1	McBSP1
01111	RINT1	McBSP1

Πίνακας 1 TMS 67xx/62xx πηγές διακοπών

Στις INT4 - INT15 μπορούν να αντιστοιχηθούν οποιεσδήποτε πηγές διακοπών ανάλογα με τις ανάγκες του χρήστη (ποια πηγή διακοπών θέλουμε να έχει τη μεγαλύτερη προτεραιότητα), χρησιμοποιώντας δύο 32-bit καταχωρητές μνήμης: IMH (Interrupt Multiplexer High, διεύθυνση μνήμης 019c 0000h) και IML (Interrupt Multiplexer Low, διεύθυνση μνήμης 019c 0004h). Αυτοί συνήθως αρχικοποιούνται στην έναρξη του προγράμματος, πριν ενεργοποιηθούν οποιεσδήποτε αιτήσεις διακοπής.

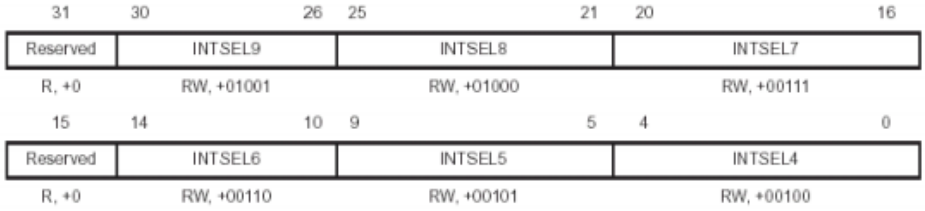
Πρόγραμμα:

```
IMH_pointer                 .set 0x019c0000
IML_pointer                 .set 0x019c0004

                .def entry
                .text
entry:          MVKL IML_pointer,A1
                        MVKH IML_pointer, A1
                        MVKL IMH_pointer, A2
                        MVKH IMH_pointer, A2
                        LDW *A1, A3
                        LDW *A2, A4
                        NOP 4

                        ; CLEAR IML, IMH

                        CLR A3, 0, 30, A3
```

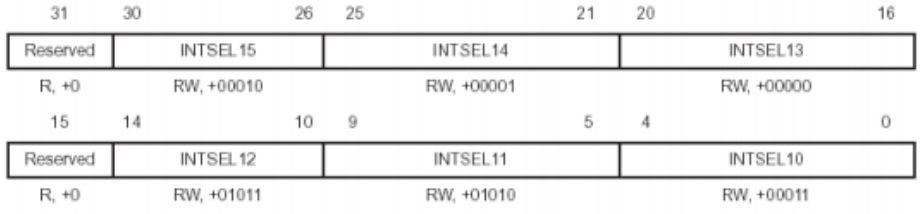
```
        CLR A4, 0, 30, A4

        ; IML

        SET A3, 0, 0, A3;               INT4 <-- TINT0

        SET A3, 6, 6, A3;               INT5 <-- TINT1

        SET A3, 12, 13, A3;     INT6 <-- XINT0

        SET A3, 16, 16, A3;     INT7 <-- RINT0
        SET A3, 18, 19, A3;     INT7 <-- RINT0

        SET A3, 22, 24, A3;     INT8 <-- XINT1

        SET A3, 26, 29, A3;     INT9 <-- RINT1

        ; IMH

        SET A4, 0, 0, A4;               INT 10 <-- EMUDTDMA
        SET A4, 3, 3, A4;               INT 10 <-- EMUDTDMA

        SET A4, 6, 6, A4;       INT 11 <-- EMURTDXRX
        SET A4, 8, 8, A4;       INT 11 <-- EMURTDXRX

        SET A4, 12, 12, A4;     INT 12 <-- GPINT4

        SET A4, 16, 16, A4;     INT 13 <-- GPINT5
        SET A4, 18, 18, A4 ;    INT 13 <-- GPINT5

        SET A4, 22, 23, A4;     INT 14 <-- GPINT6

        SET A4, 26, 28, A4;     INT 15 <-- GPINT7

        STW A3,*A1; Store Values to IML, IMH
        STW A4,*A2

IDLE
.end
```

Αρχικά, γίνεται set τις διευθύνσεις IML_pointer, IMH_pointer και τις μεταφέρουμε στους καταχωρητές A1, A2 αντίστοιχα. Έπειτα, φορτώνεται το περιεχόμενο της διεύθυνσης του A1 στον A3 και της διεύθυνσης του A2 στον A4 και μηδενίζουμε τους A3, A4 με την CLR εντολή. Μετά, με την εντολή SET ορίζονται οι αντιστοιχίες των interrupts, όπως ζητούνται, στον A3, και με τυχαίες τιμές (φαίνονται στα comments) στον A4. Αυτό γίνεται βάσει του διαγράμματος του IML και IMH

καταχωρητή (εικόνα 2 και εικόνα 3).



Εικόνα 2 Διάγραμμα του IML καταχωρητή



Εικόνα 3 Διάγραμμα του IMH καταχωρητή

Για παράδειγμα, για την αντιστοίχιση του INT4 με το TINT0 event (00001) εκτελείται η SET A3, 0, 0, A3 δηλαδή γίνεται το LSB 1. Στο τέλος γίνεται αποθήκευση των A3, A4 πίσω στην διεύθυνση των A1, A2 αντίστοιχα.

Άσκηση 3.2

Να δημιουργηθεί ένα αρχείο διανυσμάτων διακοπών χρησιμοποιώντας την ίδια αντιστοίχιση που χρησιμοποιήθηκε στην προηγούμενη άσκηση. Να υποτεθεί ότι οι διευθύνσεις (labels) των ISRs είναι TINT0_ISR, TINT1_ISR, κ.λπ. Το αρχείο σας πρέπει να περιέχει ακριβώς 128 εντολές.

Αναφέρθηκε ότι, κατά την έλευση κάποιας διακοπής, η ΚΜΕ μεταφέρει τον έλεγχο και την εκτέλεση από την τρέχουσα θέση της σε μια άλλη θέση που εξυπηρετεί τη συγκεκριμένη αίτηση διακοπής. Πού βρίσκεται λοιπόν ο κώδικας που εξυπηρετεί την αίτηση διακοπής; Αυτός ο κώδικας καλείται ρουτίνα εξυπηρέτησης διακοπής (Interrupt Service Routine - ISR) και πρέπει να ορίσουμε στην ΚΜΕ τις θέσεις των ISR της κάθε διακοπής. Αυτό γίνεται χρησιμοποιώντας το αρχείο διανυσμάτων διακοπών (Interrupt vector file - vectors.asm) που παρουσιάσαμε στην εργαστηριακή άσκηση 1 (εκεί περιγράφονταν μόνο η διακοπή Reset). Στην C67xKME, οι 8x16 = 128 εντολές που αρχίζουν από τη θέση 0x0000 0000h της μνήμης, καθορίζουν τα διανύσματα διακοπών. Για κάθε μία διακοπή υπάρχουν 8 καθορισμένες θέσεις για εντολές. Θυμηθείτε ότι είχαμε το αρχείο vectors.asm στο προηγούμενο εργαστήριο που καθορίζει τι κάνει η ΚΜΕ όταν λαμβάνει το σήμα RESET. Το RESET είναι ένα σήμα διακοπής (πολύ ειδικό) και όταν λαμβάνεται, ο έλεγχος του προγράμματος μεταφέρεται στη διεύθυνση 0x0000 0000h. Ομοίως, για κάθε μία διακοπή, υπάρχει μια συγκεκριμένη διεύθυνση στην οποία μεταφέρεται ο έλεγχος προγράμματος. Για την NMI, η θέση της ISR της είναι 0x0000 0020h, ακριβώς 32 bytes (8 εντολές - κάθε εντολή είναι μια διπλή λέξη 4 byte) απόσταση από την 0x0000 0000h. Έτσι, αρχικά μπορούμε να έχουμε 8 εντολές (επειδή κάθε εντολή είναι μια 32-bit λέξη) στην 0x0000 0000h που μπορούν να χρησιμοποιηθούν στη ρουτίνα RESET. Με τον ίδιο τρόπο, η NMI μπορεί να εξυπηρετηθεί χρησιμοποιώντας 8 εντολές από τη διεύθυνση μνήμης 0x0000 0020h μέχρι την 0x0000 003fh. Ο ακόλουθος πίνακας παρουσιάζει τις διευθύνσεις στις οποίες ο έλεγχος μεταφέρεται για κάθε μια αίτηση διακοπής:

NAME	START ADDRESS
RESET	0x0000 0000
NMI	0x0000 0020
reserved	0x0000 0040
reserved	0x0000 0060
INT4	0x0000 0080
INT5	0x0000 00A0
INT6	0x0000 00C0
INT7	0x0000 00E0
INT8	0x0000 0100
INT9	0x0000 0120
INT10	0x0000 0140
INT11	0x0000 0160
INT12	0x0000 0180
INT13	0x0000 01A0
INT14	0x0000 01C0
INT15	0x0000 01E0

Πίνακας 2 Διευθύνσεις εξυπηρέτησης διακοπών

Πρόγραμμα:

```
.title  "interrupt_vectors.asm"

           .ref TINT0_ISR
           .ref TINT1_ISR
           .ref XINT0_ISR
           .ref RINT0_ISR
           .ref XINT1_ISR
           .ref RINT1_ISR
           .ref EMUDTDMA_ISR
           .ref EMURTDXRX_ISR
           .ref GPINT4_ISR
           .ref TINT0_ISR
           .ref GPINT5_ISR
           .ref GPINT6_ISR
           .ref GPINT7_ISR

        .sect   "vectors"

INT_0:
           nop
           nop
           nop
           nop
           nop
           nop
           nop
           nop

INT_1:
```

```
nop
nop
nop
nop
nop
nop
nop
nop

.

.

.

.
```

Το αρχείο συνεχίζεται με τον ίδιο τρόπο μέχρι το INT15 interruption. Γίνεται η αντιστοίχιση των διακοπών με τα labels των γεγονότων. Για παράδειγμα, αν χρησιμοποιηθεί κάποιο interrupt, επειδή πρέπει να έχει μήκος ακριβώς 8 εντολές, χρησιμοποιείται μία branch που παραπέμπει στο αντίστοιχο label. Το άλμα μέσω branch γίνεται διότι στις 8 διαθέσιμες εντολές, πιθανότατα δεν θα μπορούσε να ολοκληρωθεί η διεργασία που το interrupt είναι υπεύθυνο να κάνει. Έπειτα, γίνεται η επιστροφή μέσω του καταχωρητή IRP, με την εντολή B IRP, όπου αποθηκεύεται η διεύθυνση της επόμενης εντολής. Εδώ, αρχικοποιούνται όλα τα interrupts με 8 nop εντολές, γιατί ακόμα δεν χρησιμοποιούνται κάπου από το πρόγραμμα. Τα labels ορίζονται σε ένα άλλο αρχείο asm, στο οποίο γίνεται ο ορισμός τους (.def), ενώ η αναφορά τους γίνεται στο “interrupt_vectors” (.ref).

Άσκηση 3.3

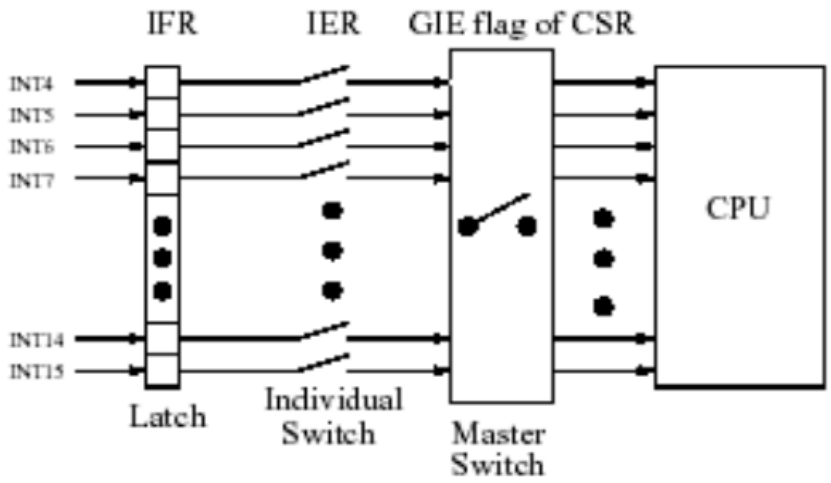
Να γραφεί ένα πρόγραμμα σε assembly που να ενεργοποιεί το INT6. Πρέπει επίσης να ενεργοποιηθούν τα NMI και GIE για να μπορεί να χρησιμοποιηθεί το INT6. ΠΡΟΣΟΧΗ: Όταν η ΚΜΕ βρίσκεται σε κατάσταση Idle κανένα interrupt δεν μπορεί να εξυπηρετηθεί!

Ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος

```
.def entry
                .text
entry:          MVKL 0X43,B0 ;
                MVC B0,IER
                MVKL 1,B1 ;THETOUME 1 TO BIT TOU GIE
                MVC B1,CSR
                IDLE
```

.end

Για να μπορέσει να εξυπηρετήσει η ΚΜΕ να εξυπηρετήσει τις διακοπές πρέπει πρώτα να ενεργοποιηθούν τα bit από συγκεκριμένους registers. Η λειτουργία και ο ρόλος των καταχωρητών φαίνεται στην Εικόνα 1.



Εικόνα 1 Καταχωρητές ενεργοποίησης διακοπών

Όταν συμβεί μία αίτηση διακοπής τίθεται σε 1 το αντίστοιχο bit του IFR (Interrupt Flag Register). Ο IER (Interrupt Enable Register) λειτουργεί ως μεμονωμένος διακόπτης που μπορεί να «ανοίξει» και να «κλείσει» κάθε διακοπή. Το Global Interrupt Enable bit (GIE) του καταχωρητή CSR (Control Status Register) δουλεύει ως γενικός διακόπτης που θέτει εκτός ή εντός λειτουργίας όλες τις διακοπές. Για να μπορούν να χρησιμοποιηθούν οι διακοπές που ενεργοποιούνται από το χρήστη, το NMI πρέπει να είναι ενεργοποιημένο. Για να γραφούν τιμές σε αυτούς τους καταχωρητές, χρησιμοποιείται η εντολή MVC. Επειδή αυτοί οι καταχωρητές είναι καταχωρητές ελέγχου δεν γίνεται να γράφει πληροφορία κατευθείαν σε αυτούς. Αντ' αυτού οι τιμές μεταφέρονται μέσω ενός καταχωρητή από το αρχείο καταχωρητών Β προς έναν καταχωρητή ελέγχου.

31														16
Reserved														
R-0														

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IE15	IE14	IE13	IE12	IE11	IE10	IE9	IE8	IE7	IE6	IE5	IE4	Reserved		NMIE	1
R/W-0												R-0		R/W-0	R-1

Legend: R = Readable by the **MVC** instruction; W = Writeable by the **MVC** instruction; *-n* = value after reset

Στην παραπάνω εικόνα φαίνεται η δομή του καταχωρητή IER. Το bit 0 είναι πάντα 1. Το NMIE πρέπει να τεθεί σε 1 καθώς και το IE6 έτσι ώστε να ενεργοποιηθεί το $6^{ο}$ interrupt. Έτσι, η λέξη που πρέπει να μεταφερθεί στο IER είναι η 1000011 που αντιστοιχεί στον δεκαεξαδικό 0X43. Αντίστοιχα , το bit του GIE τίθεται σε 1 μέσω της εντολής MVC B1,CSR.

Άσκηση 3.4

Να γραφεί μια ρουτίνα assembly που καθαρίζει όλες τις εκκρεμείς διακοπές.

Ο κώδικας που αναπτύχθηκε είναι ο ακόλουθος:

```
                .def entry
                .text

entry:          MVKL 0xFFFF,B2
                MVKH 0xFFFF,B2
                MVC B2,ICR
                NOP
                IDLE
                .end
```

Ισχύει ότι τα bits στον καταχωρητή IFR τίθενται σε 1 και 0 από την KME όταν δέχεται αίτηση διακοπής και μεταφέρει τον έλεγχο στα διανύσματα διακοπής. Ωστόσο δεν υπάρχει η δυνατότητα αλλαγής ενός bit κατευθείαν πάνω στον IFR. Προκειμένου να τεθούν τα bit του IFR σε 0 η 1 χρησιμοποιούνται οι registers ISR (Interrupt Set Register) και ICR (Interrupt Clear Register). Για να λάβει ένα bit του IFR την τιμή 1 πρέπει να τεθεί το αντίστοιχο bit του ISR σε 1. Έτσι , στον επόμενο κύκλο ρολογιού το αντίστοιχο bit του IFR γίνεται 1. Αντίστοιχα, για να καθαριστεί ένα bit του IFR πρέπει να γραφεί 1 στον αντίστοιχο bit του IFR. Με βάση αυτή τη λογική , φορτώθηκε στον B2 η λέξη FFFFFFFF και έπειτα μέσω της MVC φορτώθηκε στον καταχωρητή ICR έτσι ώστε να τεθούν σε 0 τα bit του IFR σε 0.

Άσκηση 3.5

Να γραφεί ένα πρόγραμμα σε assembly, που να προγραμματίζει το χρονόμετρο 0 με τις ακόλουθες ιδιότητες. Συμβουλευτείτε το manual για την αρχικοποίηση του χρονομέτρου:
•Το TOUT είναι ένα pin εξόδου γενικής χρήσης

• Το DATOUT οδηγείται στο TOUT

• Κατάσταση λειτουργίας χρονομέτρου για TSTAT

• KME clock/4 ως πηγή ρολογιού για εισαγωγή στο χρονόμετρο

• Επίσης γράψτε μια ρουτίνα σε assembly που θέτει την περίοδο του χρονομέτρου να είναι περίπου 1 δευτερόλεπτο. Το ρολόι της ΚΜΕ του DSK board είναι 225MHz. Κατόπιν, γράψτε επίσης μια ρουτίνα σε assembly που να θέτει σε εκκίνηση το χρονόμετρο 0 για να μετρήσει.

• Χρησιμοποιώντας το χρονόμετρο που προγραμματίσατε, γράψτε μια ρουτίνα σε assembly ώστε ένα από τα LED (επιλέξτε εσείς ποιο) να ανάβει για ένα δευτερόλεπτο, έπειτα να σβήνει για ένα δευτερόλεπτο κ.ο.κ (να αναβοσβήνει δηλαδή ένα LED με περίοδο λειτουργίας 1 δευτερόλεπτο).

Main πρόγραμμα:

```
                    .def entry
                    .text
a                   .set 0x1940000 ; TIMER CONTROL ADDRESS
b                   .set 0x1940004 ; PERIOD ADDRESS
c                   .set 0x35A4E90 ; T=56250000
d                   .set 0x90080000 ; LED ADDRESS
IML_pointer     .set 0x019c0004
IMH_pointer     .set 0x019c0000

entry:
                    MVKL 0xFFFF, B0; CLEAR ALL PREVIOUS INTERRUPTS
                    MVKH 0xFFFF, B0
                    MVC B0, ICR
                    NOP

                    MVKL IML_pointer, A1
                    MVKH IML_pointer, A1
                    LDW *A1, A3;
                    Nop 4
                    CLR A3, 0, 30, A3
                    SET A3, 0, 0, A3; Set timer TINTT0 --> INT 4
                    STW A3,*A1

                    MVKL IMH_pointer, A2
                    MVKH IMH_pointer, A2
                    LDW *A2, A4
                    Nop 4
                    CLR A4, 0, 30, A4
                    SET A4, 21, 21, A4
                    STW A4,*A2

                    MVKL a, B5; POINTS TO TIMER CONTROL ADDRESS
                    MVKH a, B5

                    MVKL b, B3; SET PERIOD
                    MVKH b, B3
                    MVKL c, B7;
                    MVKH c, B7
                    STW B7,*B3

                    MVKL 0X13, B1; SET NMI AND INT4 TO 1
                    MVC B1, IER
```

```
                MVKL 1, B2; SET GIE TO 1
                MVC B2, CSR

                MVKL d, B8; LED ADDRESS REGISTER
                MVKH d, B8
                MVKL 0x01, B9
                MVKH 0x01, B9

                ZERO B6

                SET B6, 6, 7, B6; GO AND HOLD SET TO 1
                SET B6, 9, 9, B6; CLCKSRC --> CPU_CLOCK/4
                CLR B6, 0, 0, B6; FUNC=0 --> TOUT = GEN. PUR
        CLR B6, 2, 2, B6; DATOUT--> TOUT
                STW B6,*B5

loop:
                B loop
                NOP 5

                IDLE
                .end
```

Αρχικά, γίνεται set στις μεταβλητές a, b, c, d η διεύθυνση του timer control, η διεύθυνση της περιόδου, η τιμή της περιόδου η διεύθυνση της λέξης που ελέγχει το led. Επίσης, φορτώνονται οι διευθύνσεις IMH_pointer και IML_pointer.

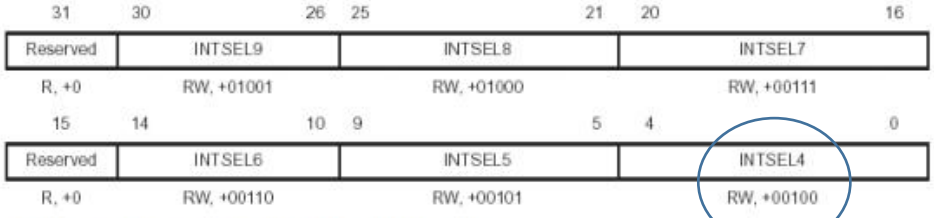
Στην συνέχεια, καθαρίζονται όλες οι προηγούμενες διακοπές μέσω των εντολών :

```
MVKL 0xFFFF,B0 ; CLEAR ALL PREVIOUS INTERRUPTS
MVKH 0xFFFF,B0
MVC B0,ICR
NOP
```

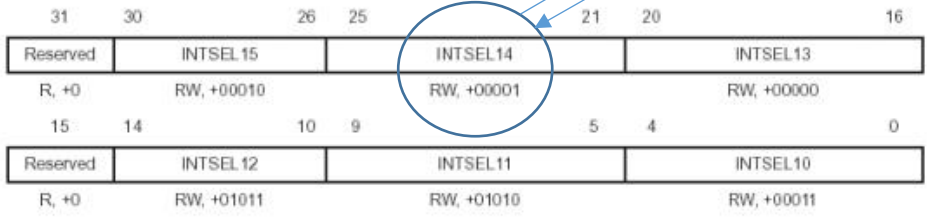
Έπειτα, φορτώνονται το TINT0 στο INT4 όπως αναφέρθηκε και στο ερώτημα 3.1:

```
MVKL IML_pointer,A1
MVKH IML_pointer,A1
LDW *A1,A3
nop 4
CLR A3,0,30,A3
SET A3,0,0,A3 ;Set timer TINTT0 --> INT 4
STW A3,*A1
```

Επειδή by default το TINT0 αντιστοιχίζεται στο INT14, εάν δεν γίνει κάποια επιπλέον αλλαγή στον IMH, το INT4 θα αντιστοιχεί σε δύο διαφορετικά interrupts τα οποία θα τρέχουν ταυτόχρονα. Για αυτό, αντικαθιστούμε την τιμή 00100 (GPINT4) στο INT14 το οποίο αρχικά είχε το event TINT0, ώστε να επιλυθεί αυτό το πρόγραμμα.



Εικόνα 2 Διάγραμμα του IML καταχωρητή



Εικόνα 3 Διάγραμμα του IMH καταχωρητή

Αυτό γίνεται με τις εντολές:

```
MVKL IMH_pointer,A2
MVKH IMH_pointer,A2
LDW *A2,A4
nop 4
CLR A4,0,30,A4
SET A4,21,21,A4
STW A4,*A2
```

Μετά, μεταφέρεται η τιμή των a, b, c στους καταχωρητές B5, B3, B7 αντίστοιχα και γίνεται αποθήκευση της τιμής της περιόδου στη διεύθυνση που δείχνει ο B3.

```
MVKL a,B5 ; POINTS TO TIMER CONTROL ADDRESS
MVKH a,B5

MVKL b, B3 ; SET PERIOD
MVKH b ,B3
MVKL c, B7 ;
MVKH c ,B7
STW B7,*B3
```

Οι εντολές που ακολουθούν ενεργοποιούν το INT4 καθώς και τα interrupts γενικότερα με λογική όμοια με αυτή που χρησιμοποιήθηκε στο ερώτημα 3.3:

```
MVKL 0X13,B1 ;SET  NMI AND INT4 TO 1
MVC B1,IER

MVKL 1,B2 ;SET GIE TO 1
MVC B2,CSR
```

Κατόπιν, φορτώνεται η διεύθυνση του led στον καταχωρητή B8 και ο αριθμός 1 στον B9:

```
MVKL d,B8 ;LED ADDRESS REGISTER
MVKH d,B8
MVKL 0x01,B9
MVKH 0x01,B9
```

Ύστερα, γίνονται οι αρχικοποιήσεις του timer 0 όπως ζητούνται από την εκφώνηση της άσκησης, σύμφωνα με το manual της του Lab 3 και αποθηκεύονται στην διεύθυνση του register control:

```
SET B6,6,7,B6 ; GO AND HOLD SET TO 1
SET B6,9,9,B6 ; CLCKSRC --> CPU_CLOCK/4
CLR B6,0,0,B6 ; FUNC=0 -->TOUT = GENERAL PURPOSE
CLR B6,2,2,B6 ; DATOUT--> TOUT
STW B6,*B5
```

Έπειτα, το πρόγραμμα εισάγεται σε έναν ατέρμονα βρόχο (loop), έτσι ώστε το πρόγραμμα να εκτελείται συνεχώς:

```
loop:
                B loop
                NOP 5

                IDLE
                .end
```

Πριν τρέξει το πρόγραμμα, θα γίνει τροποποίηση του αρχείου “interrupt_vectors” έτσι ώστε να παραπέμπει στο στις εντολές που πρέπει να εκτελεστούν όταν εκτελείται το INT4. Αυτό θα γίνει με branch εντολή. Συγκεκριμένα η τροποποίηση:

```
INT4:
            B TINT0_ISR
            nop
            nop
            nop
            nop
            nop
            nop
            nop
```

Με αυτή την branch εντολή οδηγείται το πρόγραμμα στο επιπλέον αρχείο που έχει δημιουργηθεί με τον ακόλουθο κώδικα του interrupt:

```
            .def TINT0_ISR
            .text

TINT0_ISR:
            LDW *B8, B9
            NOP 4
            XOR B9,1,B9
            STW B9,*B8
            B IRP
            nop 5
```

Ο κώδικας αυτός τον αριθμό 1 (B9) στην διεύθυνση των led (B8) και μέσω της εντολής XOR, γίνεται εναλλαγή του B9 από 0 σε 1 και αντίστροφα σε κάθε διακοπή. Η τιμή αυτή έπειτα αποθηκεύεται στη διεύθυνση των led και γίνεται επιστροφή στο entry πρόγραμμα μέσω της B IRP όπως εξηγήθηκε στην άσκηση 3.2

Βιβλιογραφία:

- **TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide**
- **ΕΚΦΩΝΗΣΗ ΑΣΚΗΣΗΣ LAB-3 – ECLASS**
- **TMS320C6713 DATASHEET**