Μικροελεγκτές

Παρουσιάσεις για το χειμερινό εξάμηνο 2022-23

Διεθνές Πανεπιστήμιο της Ελλάδος

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

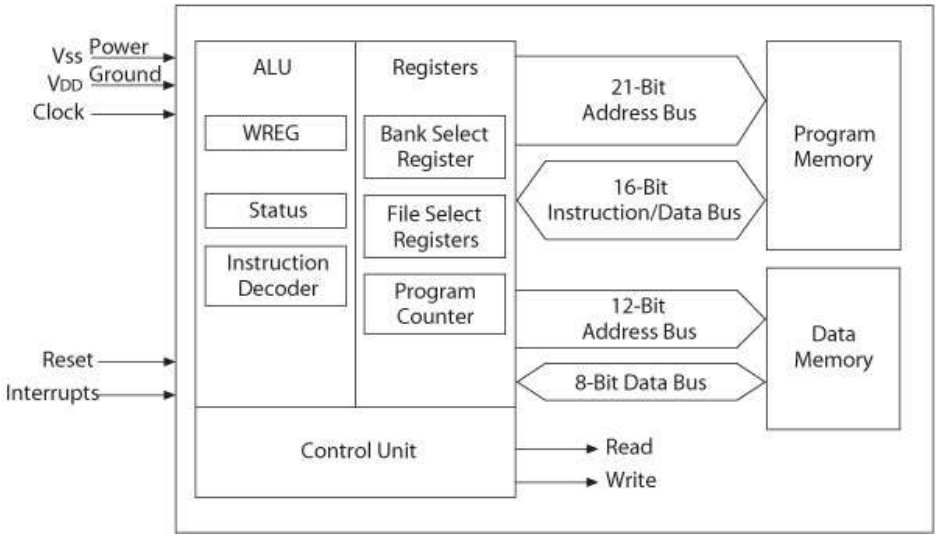
Μάθημα 5ου εξαμήνου: Μικροελγκτές

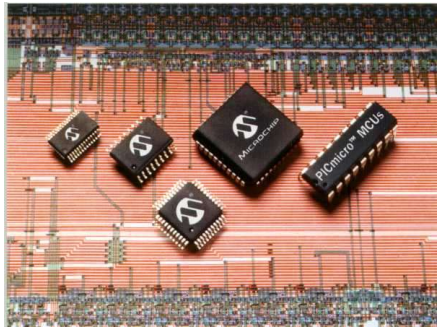
Απαιτούμενες γνώσεις: Ψηφιακά Ι, Ψηφιακά ΙΙ, Αρχιτεκτονική Υπολογιστικών Συστημάτων

Διδάσκοντες: Καζακόπουλος Αριστοτέλης, Καθηγητής

Αντικείμενο του μαθήματος

Το αντικείμενο του μαθήματος είναι η εισαγωγή στους μικροελεγκτές και ειδικότερα στον μικροελεγκτή PIC18F4550 της εταιρείας Microchip





Τι είναι ο μικροελεγκτής;

Βιβλία

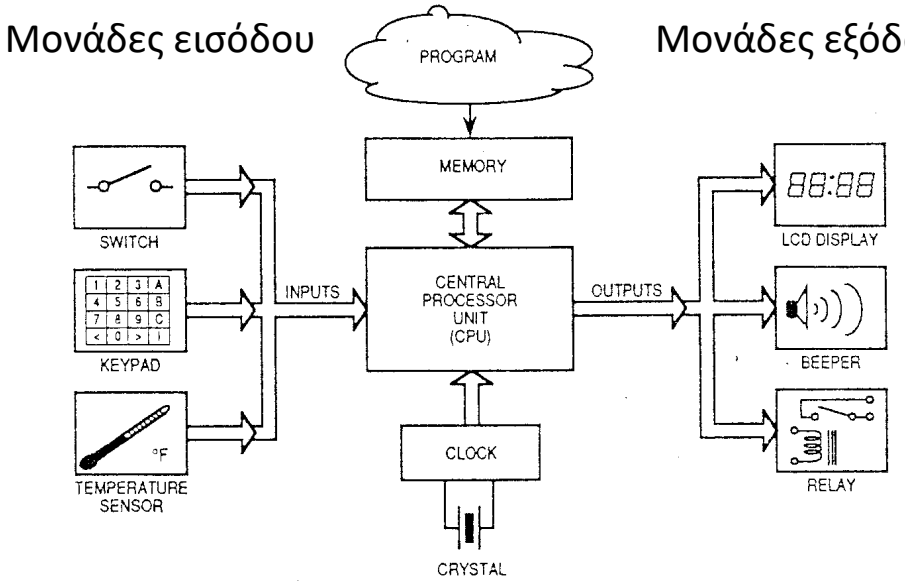
Η ύλη του μαθήματος θα καλύπτεται εξ ολοκλήρου από το υλικό που είναι αναρτημένο ή θα αναρτηθεί στο Moodle

Συνιστώμενα βιβλία, δεν είναι υποχρεωτικό να διαλέξετε από αυτά.

Βιβλίο [68405485]: Εισαγωγή στους Μικροελεγκτές PICmicro, Αλατσαθιανός Σ. (Είναι για μικρολεγκτές PIC, είναι ποιο κοντά στην ύλη του μαθηματος)

Βιβλίο [77119034]: Μικροελεγκτές, ΝΙΚΟΛΑΪΔΗΣ ΝΙΚΟΛΑΟΣ (Είναι για μικροελεγκτές AVR που χρησιμοποιούνται στη δημοφιλή πλατφόρμα Arduino)

Τυπικό υπολογιστικό σύστημα(1)



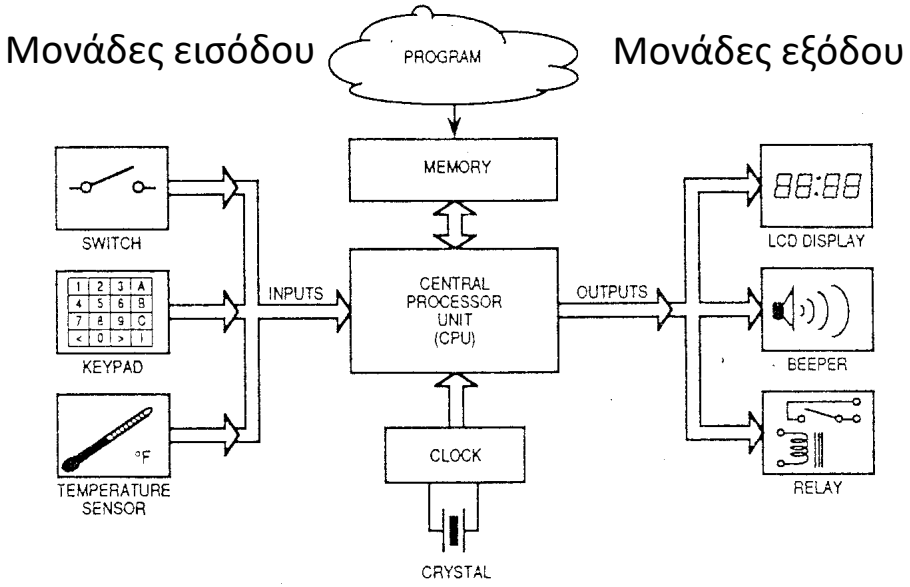
Μικροεπεξεργαστής (Microprocessor)
Μια Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ) σε ένα chip
CPU: Central Processing Unit

Μικροελεγκτής (Microcontroller)
CPU, μνήμη, μονάδες εισόδου εξόδου, άλλα συστήματα π.χ. A/D, χρονιστές κλπ,
Όλα σε ένα chip

Η Κεντρική Μονάδα Επεξεργασίας επεξεργάζεται τις πληροφορίες σύμφωνα με κάποιο πρόγραμμα το οποίο είναι αποθηκευμένο στη μνήμη προγράμματος σε μια γλώσσα που λέγεται γλώσσα μηχανής (Machine Language)

Το πρόγραμμα αποθηκεύεται στη μνήμη του υπολογιστικού συστήματος υπό μορφή δυαδικού κώδικα

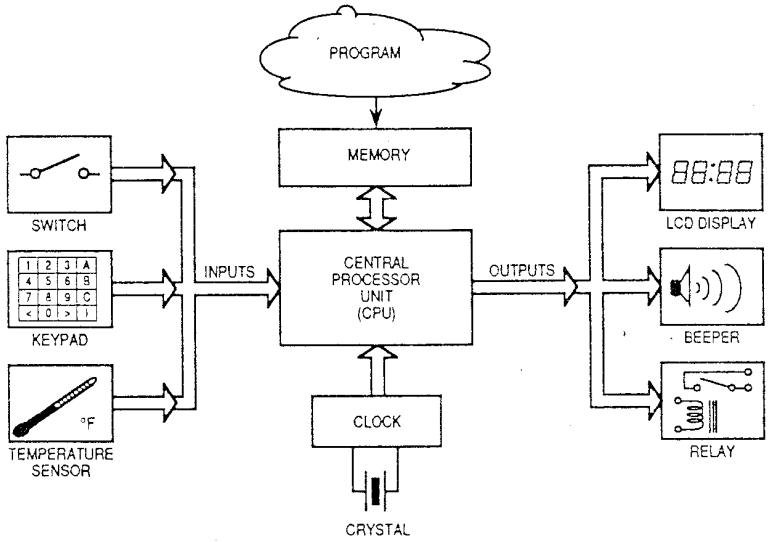
Typical computer system(2)



Οι **μονάδες εισόδου** προμηθεύουν με πληροφορίες την Κεντρική Μονάδα Επεξεργασίας (CPU). Κάποιες μονάδες εισόδου μετατρέπουν τα αναλογικά σήματα σε σήματα που μπορεί να επεξεργαστεί η Κεντρική Μονάδα Επεξεργασίας. Άλλες μονάδες εισόδου μετατρέπουν πληροφορίες του πραγματικού κόσμου σε σήματα 0 και +5 V τα οποία επεξεργάζεται η Κεντρικη Μονάδα Επεξεργασίας. Παραδείγματα τέτοιων μονάδων εισόδου είναι ο αισθητήρας θερμοκρασίας, ο διακόπτης, ένα πληκτρολόγιο, ένα ποντίκι.

Οι **μονάδες εξόδου** ελέγχονται από σήματα τα οποία αποστέλλονται από την Κεντρική Μονάδα Επεξεργασίας (CPU) προς τον εξωτερικό κόσμο. Ενδείκτες υγρών κρυστάλλων (LCD Displays), ηχητικά συστήματα, ρελέ, συστήματα οδήγησης θέρμανσης ή ψύξης είναι τέτοιες μονάδες εξόδου.

Typical computer system(3)



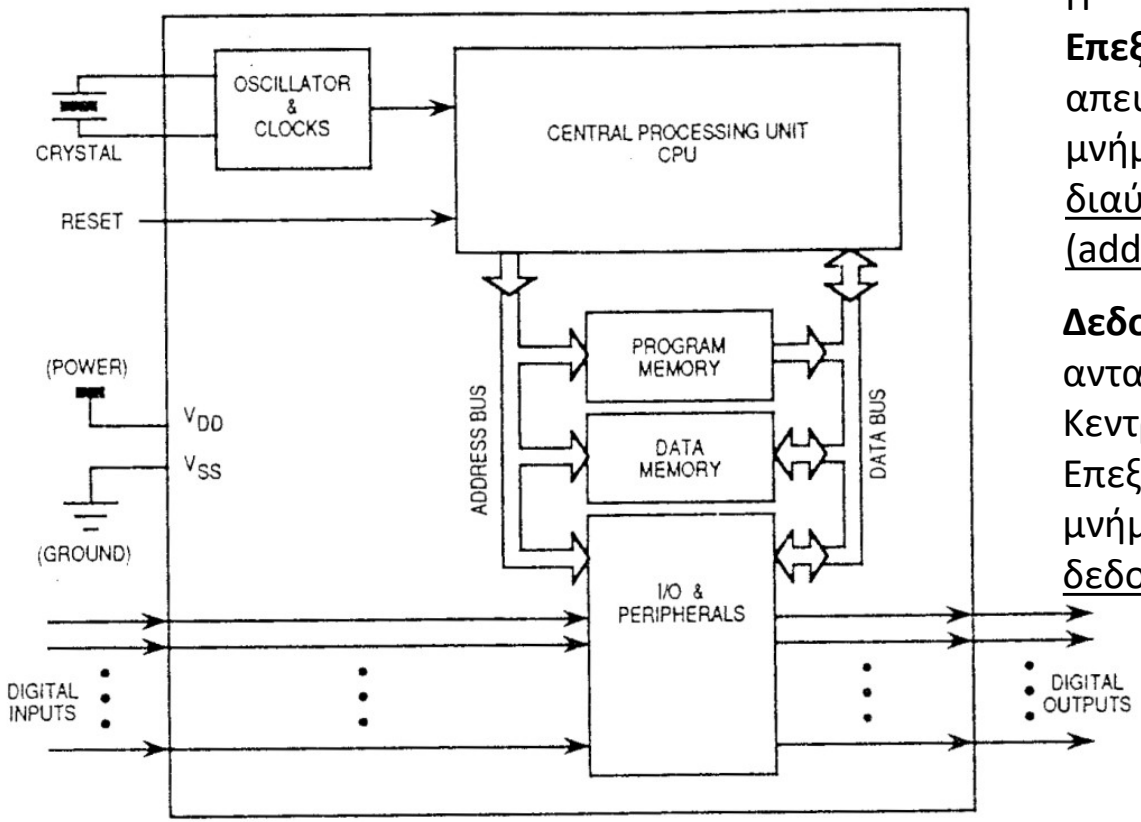
Στη **μνήμη** μπορούν να αποθηκεύονται πληροφορίες, συμπεριλαμβανομένου του προγράμματος που χρησιμοποιεί η Κεντρική Μονάδα Επεξεργασίας (CPU) Οι δύο βασικοί τύποι μνήμης είναι η μνήμη δεδομένων(Data Memory, συχνά αναφέρεται και σαν μνήμη RAM)) και η μνήμη μόνο ανάγνωσης (ROM, Read Only Memory)

Η Μνήμη **RAM** χρησιμοποιείται για προσωρινή αποθήκευση δεδομένων και τα δεδομένα χάνονται όταν σβήσει η τροφοδοσία

Η μνήμη **ROM** χρησιμοποιείται για τη μόνιμη αποθήκευση εντολών ή δεδομένων. Υπάρχουν διαφορετικοί τύποι μνήμης ROM όπως EPROM, EEPROM, Flash Memory.

Το ρολόι του υπολογιστικού συστήματος για τον χρονισμό των διαφόρων λειτουργιών. Ένας κρύσταλλος συνήθως χρησιμοποιείται για να δίνει συχνότητα αναφοράς

Μπλοκ διάγραμμα του μικροελεγκτή



Η **Κεντρική Μονάδα Επεξεργασίας (CPU)** απευθύνεται σε μια θέση της μνήμης δια μέσου του διαύλου διευθύνσεων (address bus)

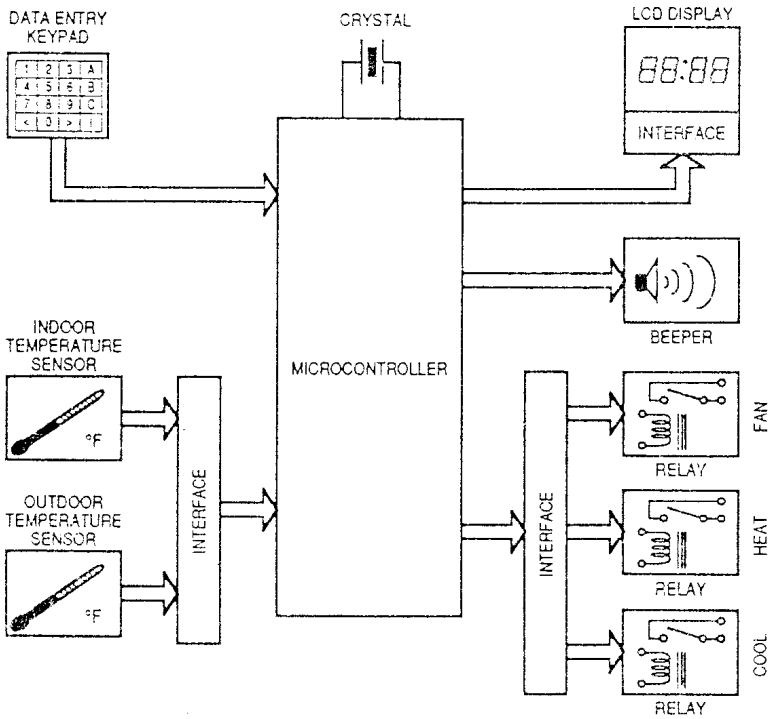
Δεδομένα (Data) ανταλάσσονται ανάμεσα στην Κεντρική Μονάδα Επεξεργασίας (CPU) και τη μνήμη δια μέσου του διαύλου δεδομένων (data bus).

Ένας μικροελεγκτής μπορεί επίσης να έχει αναλογικές εισόδους/ εξόδους (Analog inputs/outputs

Μπλοκ διάγραμμα ενός θερμοστάτη που χρησιμοποιεί μικροελεγκτή

Μονάδες εισόδου

Μονάδες εξόδου

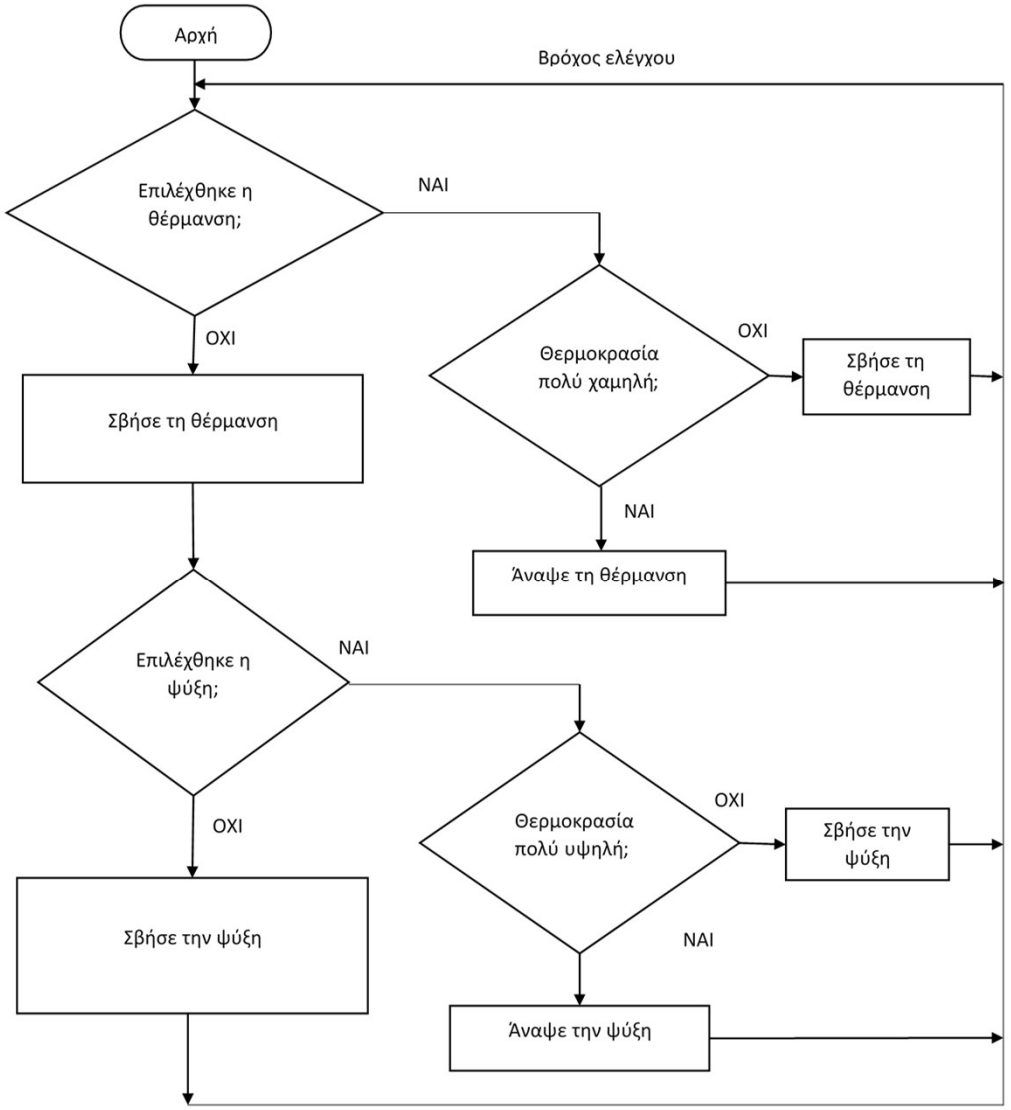


Μικροελεγκτής (Microcontroller):

CPU, Μνήμη, μονάδες εισόδου εξόδου σε ένα chip

Διεπαφές (Interfaces) μπορούν να χρησιμοποιούνται για να συνδέουν εξωτερικές μονάδες εισόδου εξόδου στις ψηφιακές εισόδους του μικροελεγκτή

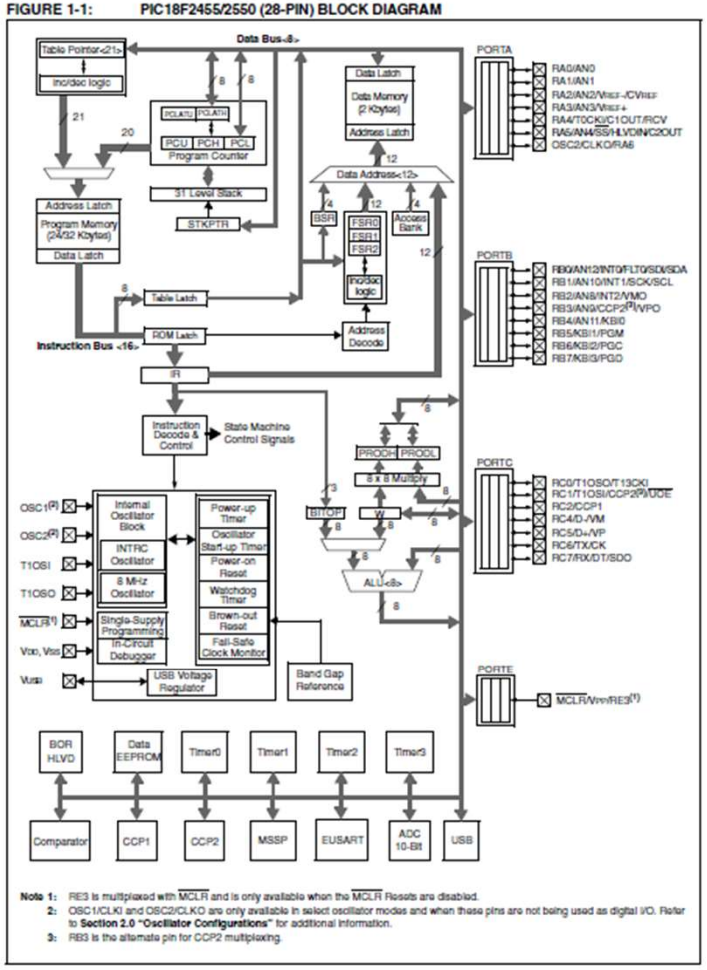
Διάγραμμα ροής του θερμοστάτη



Εσωτερική δομή του μικροελεγκτή PIC18F4550

PIC18F2455/2550/4455/4550

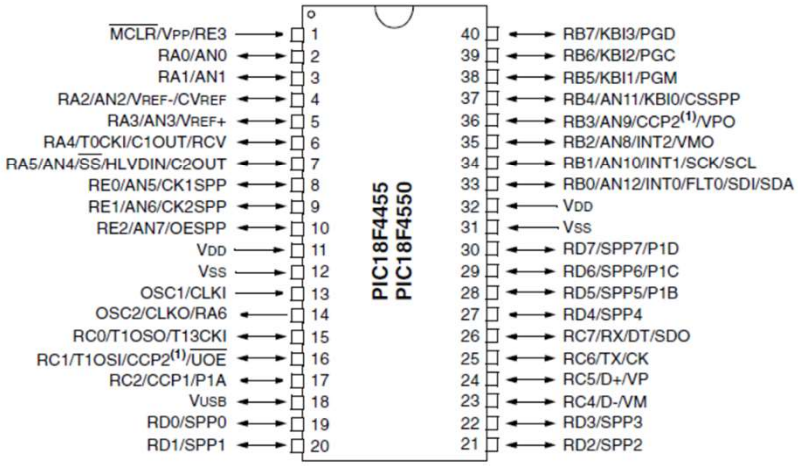
FIGURE 1-1: PIC18F2455/2550 (28-PIN) BLOCK DIAGRAM



Note 1: RE3 is multiplexed with MCLR and is only available when the MCLR Resets are disabled.
2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.
3: RB3 is the alternate pin for CCP2 multiplexing.

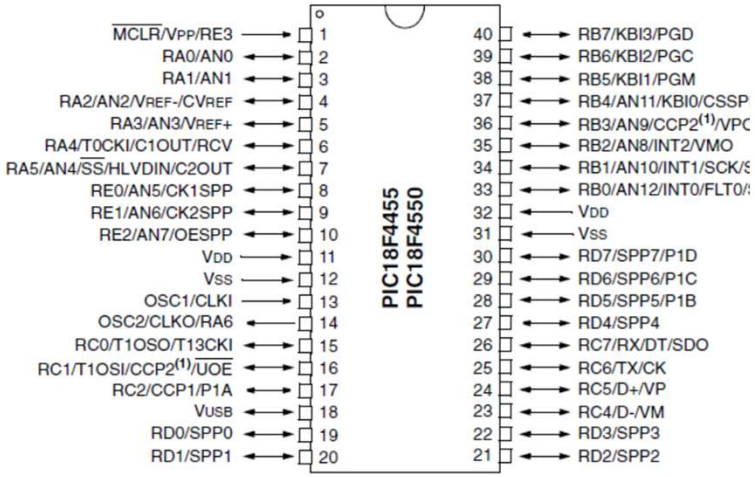
Τι είναι ορατό από έξω

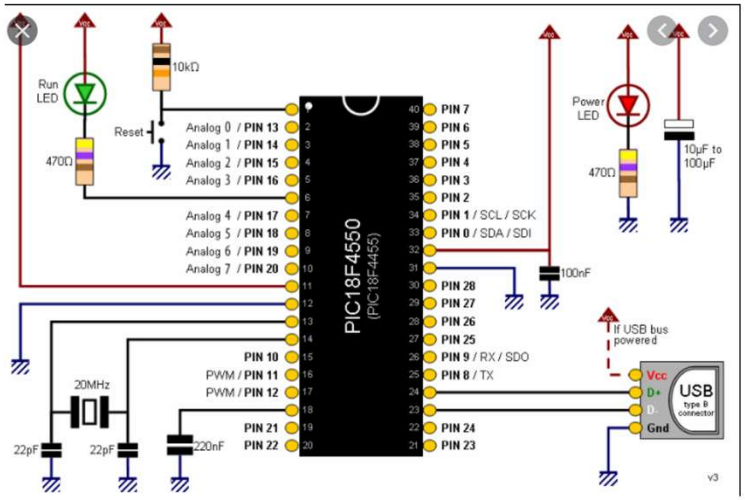
-Pin PDIP



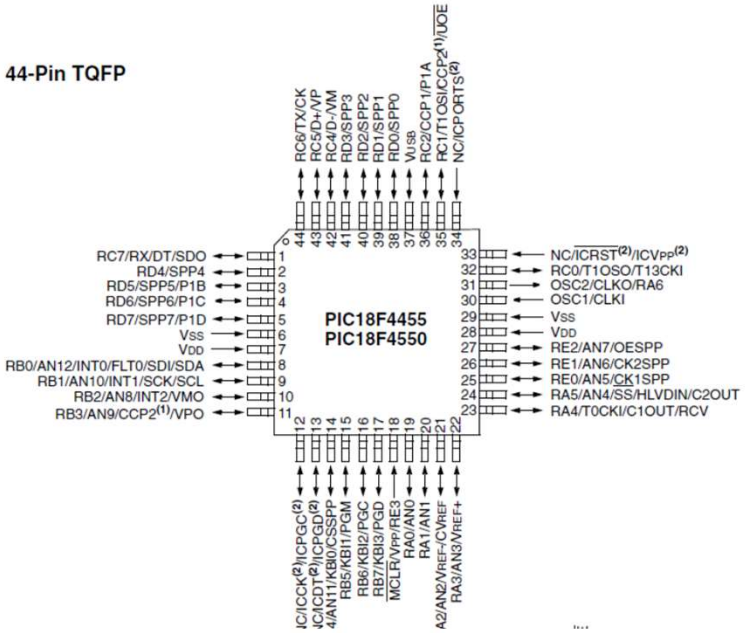
Chip PIC18F4550

40-Pin PDIP

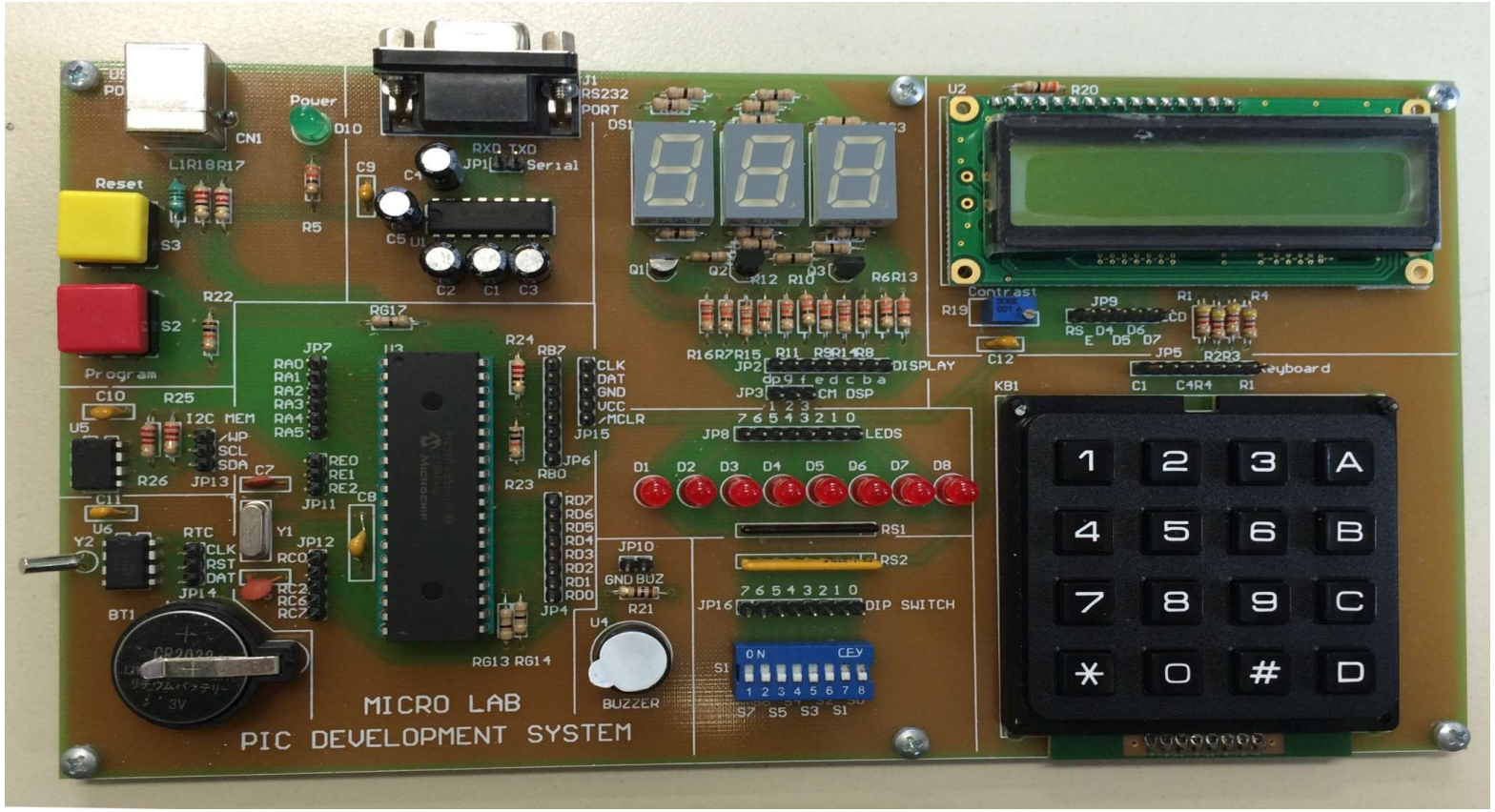




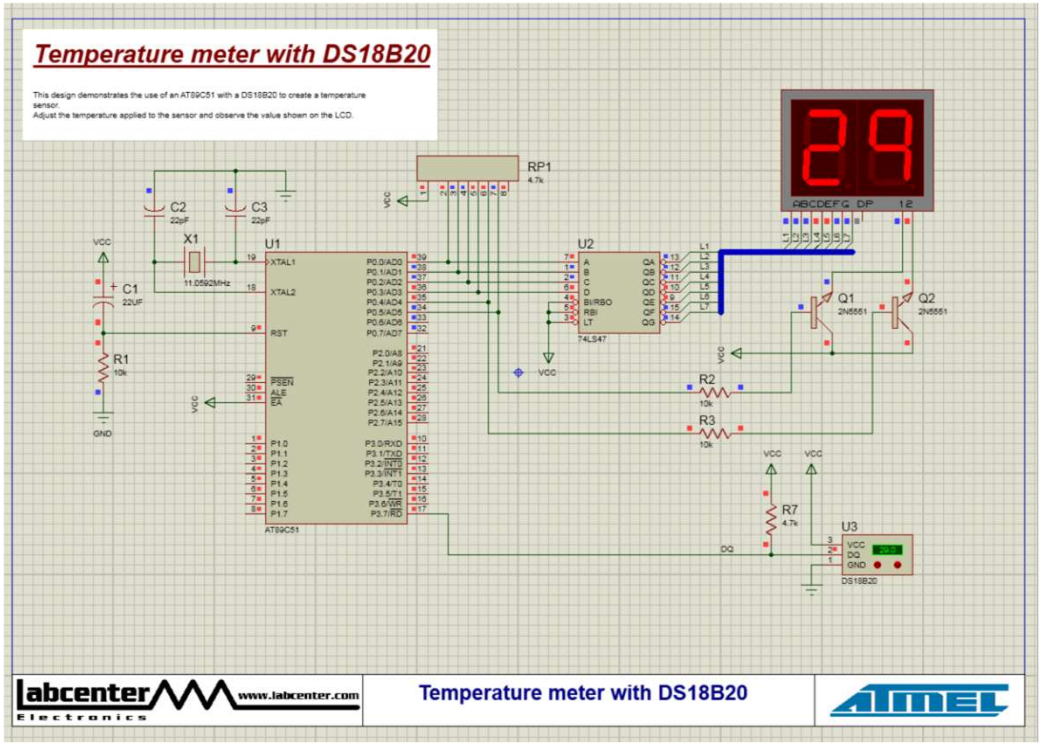
44-Pin TQFP



Πλακέτα ανάπτυξης εφαρμογών του εργαστηρίου Ενσωματωμένα Συστήματα

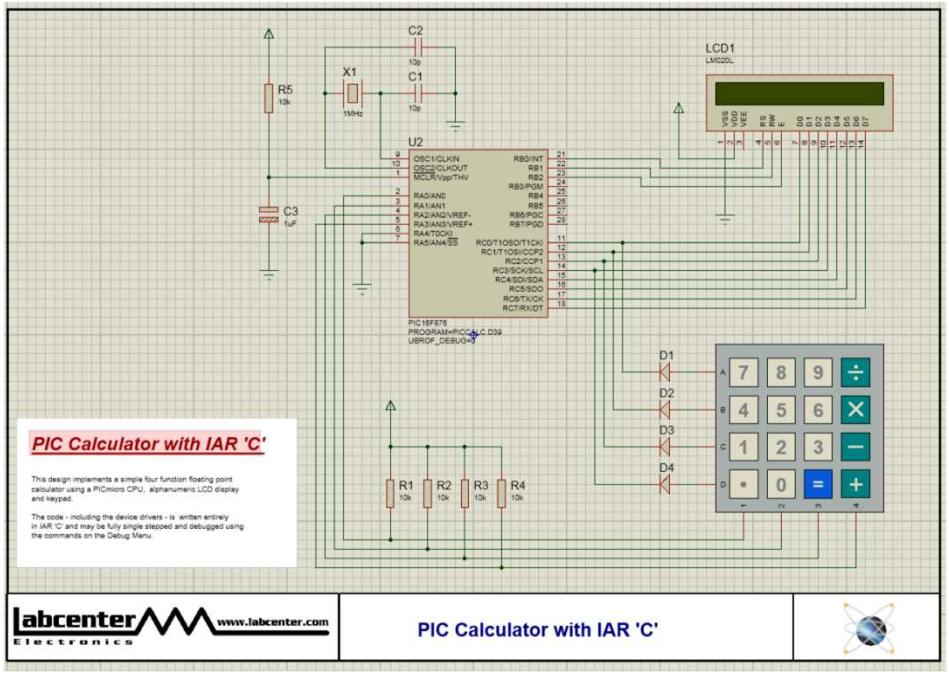


Παράδειγμα θερμομέτρου



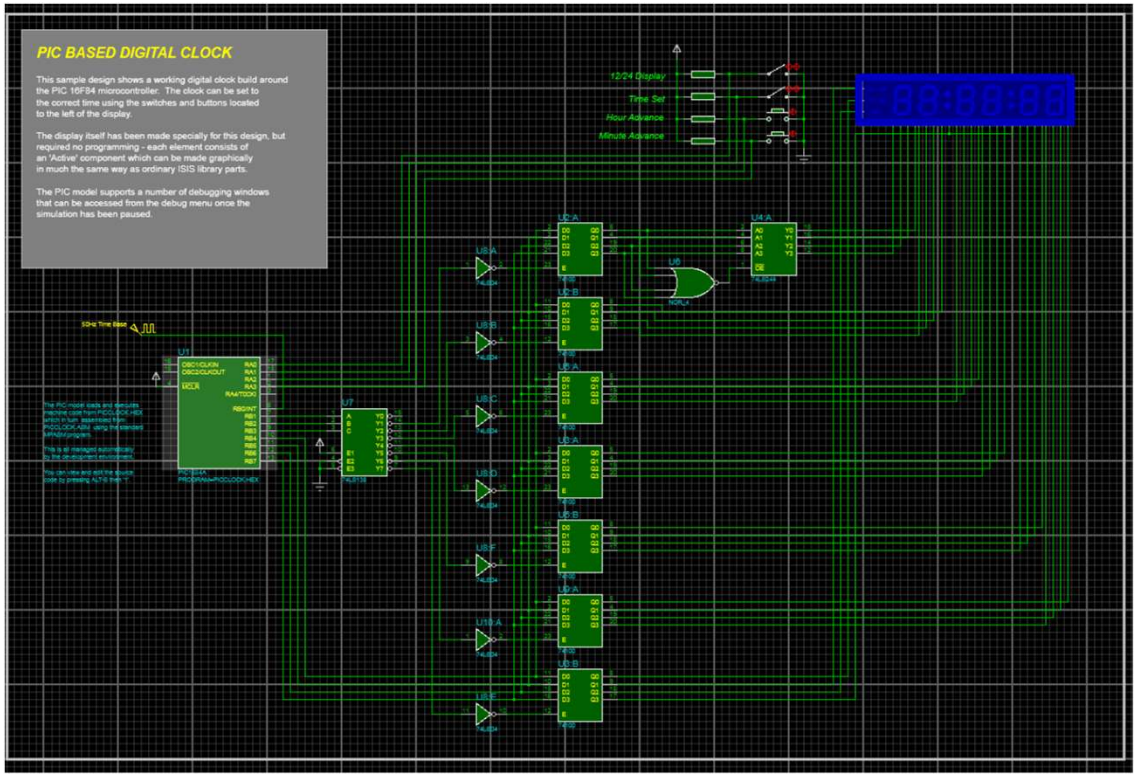
Σ’ αυτή την εφαρμογή ο μικροελεγκτής χρησιμοποιείται για να διαβάζει δεδομένα από ένα αισθητήρα θερμοκρασίας να τα επεξεργάζεται με σύμφωνα με κάποιο πρόγραμμα και να εμφανίζει τη θερμοκρασία με δύο ψηφία

Παράδειγμα αριθμομηχανής



Σ’ αυτή την εφαρμογή ο μικροελεγκτής χρησιμοποιείται σε μια εφαρμογή αριθμομηχανής.
Ο μικροελεγκτής έχει προγραμματιστεί για να διαβάζει από ένα πληκτρολόγιο 4X4 (είσοδος) και να εμφανίζει σε μια οθόνη υγρών κρυστάλλων LCD display (έξοδος).

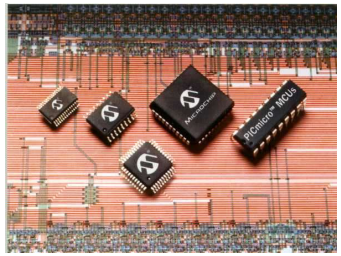
Ψηφιακό ρολόι με μικροελεγκτή PIC



Σ’ αυτή την εφαρμογή ο μικροελεγκτής χρησιμοποιείται σε μια εφαρμογή ψηφιακού ρολογιού.

Ο ταλαντωτής(Oscillator) του μικροελεγκτή χρησιμοποιείται σαν πηγή μέτρησης χρόνου.

Πως προγραμματίζουμε ένα μικροελεγκτή;



Ποια είναι η γλώσσα που μπορεί να καταλάβει ένα μικροελεγκτής;

Μπορούμε να προγραμματίσουμε κατευθείαν στη γλώσσα του μικροελεγκτή;

Η γλώσσα του μικροελεγκτή

- Η μοναδική γλώσσα που μπορεί να καταλάβει η Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit) του μικροελεγκτή είναι εντολές με μορφή δυαδικού κώδικα.

- Η γλώσσα αυτή λέγεται γλώσσα μηχανής

Instructions							
1	1	1	0	0	0	1	1
0	0	0	1	0	1	0	1
1	1	0	0	1	0	1	0
1	1	0	0	1	1	1	1
0	0	1	1	1	1	1	0
1	0	0	0	1	0	0	0

Παρουσίαση του υλικού του μαθήματος που είναι αναρτημένο στο Moodle

Έγινε παρουσίαση του υλικού του μαθήματος Μικροελεγκτές που είναι αναρτημένο στο Moodle.
Οι φοιτητές θα πρέπει να εγκαταστήσουν στους υπολογιστές τους το πρόγραμμα προσομοίωσης ηλεκτρονικών κυκλωμάτων Proteus και το πρόγραμμα MPLAB. Τα προγράμματα αυτά είναι αναρτημένα στο Moodle

Microcontroller Programming languages

C language

(Είναι ίδια με μικρές παραλλαγές για διαφορετικούς μικροελεγκτές)

Είναι εύκολο να γράψεις πρόγραμμα, δεν είναι απαραίτητη η γνώση της δομής του μικροελεγκτή

Assembly language

(Διαφορετική για διαφορετικές οικογένειες μικροελεγκτών)

Ποιο δύσκολο να γράψεις πρόγραμμα, είναι απαραίτητη κάποια γνώση της δομής του μικροελεγκτή

C Compiler

Ο C compiler είναι ένα πρόγραμμα με το οποίο γίνεται μετάφραση από γλώσσα C σε γλώσσα μηχανής

Assembler

Ο Assembler είναι ένα πρόγραμμα με το οποίο γίνεται μετάφραση από γλώσσα Assembly σε γλώσσα μηχανής

Machine Language

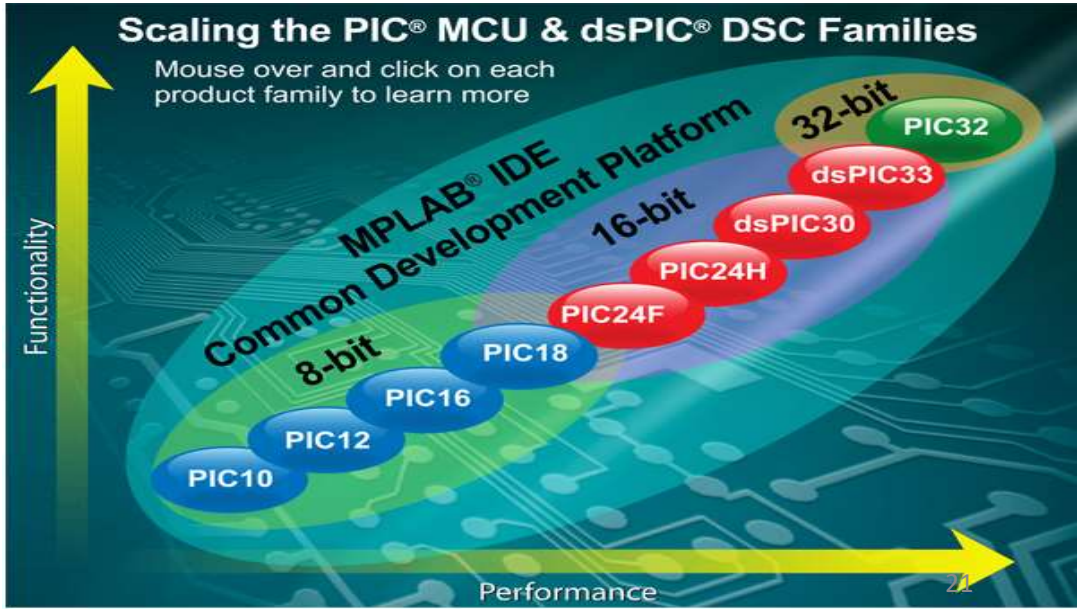
(Δυαδικός κώδικας, η μόνη γλώσσα που μπορεί να καταλάβει η Κεντρική Μονάδα Επεξεργασίας CPU)

Instructions							
1	1	1	0	0	0	1	1
0	0	0	1	0	1	0	1
1	1	0	0	1	0	1	0
1	1	0	0	1	1	1	1
0	0	1	1	1	1	1	0
1	0	0	0	1	0	0	0

Οικογένεια μικροελεγκτών PIC18F

- Οι μικροελεγκτές PIC έχουν αρχιτεκτονική Harvard και περιλαμβάνουν :

  - Μικροεπεξεργαστή(MPU Microprocessor Unit)
  - Μνήμη προγράμματος για εντολές (καταχωρητές των 16 bit)
  - Μνήμη δεδομένων(καταχωρητές των 8 bit)
  - Μονάδες εισόδου εξόδου (I/O ports)
  - Υποστηρικτικές μονάδες όπως, χρονιστές(timers) και μετατροπείς αναλογικού σήματος σε ψηφιακό.



Αριθμητικά συστήματα

Δεκαδικό (Decimal)

Δυαδικό (Binary)

Δεκαεξαδικό (Hexadecimal)

Δεκαδικό αριθμητικό σύστημα (1)

(Είναι ένα θεσιακό αριθμητικό σύστημα, η αξία του κάθε ψηφίου εξαρτάται από τη θέση του)

Στο δεκαδικό αριθμητικό σύστημα χρησιμοποιούνται 10 (0,1,2,…9)ψηφία και η αξία κάθε ψηφίου αυξάνει από τα δεξιά προς τα αριστερά κατά δύναμη του 10.

Το τελευταίο δεξιά ψηφίο είναι μονάδες (10^0)

Το προτελευταίο από δεξιά είναι δεκάδες (10^1)

Το τρίτο από δεξιά είναι εκατοντάδες(10^2)

Το τέταρτο από δεξιά είναι χιλιάδες(10^3)

Το πέμπτο από δεξιά είναι δεκάδες χιλιάδες (10^4)

κ.ο.κ. **1628 Ευρώ Ποια ψηφία έχουν μεγαλύτερη αξία;**

Δεκαδικό αριθμητικό σύστημα (2)

Παράδειγμα:

$15748 = 1X10^4+5x10^3+7x10^2+4X10^1+8x10^0$

$=1x10000+5x1000+7x100+4x10+8x1$

Η αξία του κάθε ψηφίου από δεξιά προς τα αριστερά είναι:

Μονάδες, δεκάδες, εκατοντάδες, χιλιάδες, δεκάδες χιλιάδες

Την αξία του κάθε ψηφίου μπορούμε να την γράψουμε στον παρακάτω πίνακα.

Ψηφίο	**1**	**5**	**7**	**4**	**8**
Αξία	10000	1000	100	10	1

Δυαδικό αριθμητικό σύστημα (1)

Στο δυαδικό αριθμητικό σύστημα χρησιμοποιούνται μόνο 2 ψηφία(0,1) και οι αριθμοί γράφονται με τέτοιο τρόπο που η αξία αυξάνει από δεξιά προς τα αριστερά κατά δύναμη του 2.

Παράδειγμα:

$10110010_b=1x2^7+0x2^6+1x2^5+1x2^4+0x2^3+0x2^2+1x2^1+0x2^0$

$=1x128+0x64+1x32+1x16+0x8+0x4+1x2+0x1$

Η αξία του κάθε ψηφίου από δεξιά προς τα αριστερά είναι:

$1(=2^0)$, $2(=2^1)$, $4(=2^2)$, $8(=2^3)$, $16(=2^4)$, $32(=2^5)$, $64(=2^6)$,$128(=2^7)$

Για να δείξουμε ότι ένας αριθμός είναι γραμμένος στο δυαδικό σύστημα βάζουμε τον δείκτη $_b$ (από την αγγλική λέξη binary για το δυαδικός).

10110010_b $11_b=3$ $111_b=7$ $1111_b=15$

Δυαδικό αριθμητικό σύστημα (2)

$(10110010)_b$

Η αξία του κάθε ψηφίου από δεξιά προς τα αριστερά είναι:

$1(=2^0)$, $2(=2^1)$, $4(=2^2)$, $8(=2^3)$, $16(=2^4)$, $32(=2^5)$, $64(=2^6)$,$128(=2^7)$

Συμπληρώνουμε τον παρακάτω πίνακα:

ψηφίο	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**
αξία	128	64	32	16	8	4	2	1

Δυαδικό αριθμητικό σύστημα (3)

ψηφίο	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
αξία	128	64	32	16	8	4	2	1

$11_b = 3$ $111_b = 7$ $1111_b = 15$

$1111\ 1111_b = 255_d$ (Να το θυμόμαστε, σημαντικό!)

Μετατροπή από το δυαδικό στο δεκαδικό αριθμητικό σύστημα(1)

Για να μετατρέψουμε έναν δυαδικό αριθμό σε δεκαδικό πρέπει να προσθέσουμε τις αξίες των αντίστοιχων θέσεων όπου υπάρχει 1

Ο αριθμός 10110010_b πως γράφεται στο δεκαδικό αριθμητικό σύστημα;

ψηφίο	1	0	1	1	0	0	1	0
αξία	128	64	32	16	8	4	2	1

$128+32+16+2=178_d$ στο δεκαδικό αριθμητικό σύστημα.

Μετατροπή από το δυαδικό στο δεκαδικό αριθμητικό σύστημα (2)

ψηφίο	1	0	0	0	1	0	0	0
αξία	128	64	32	16	8	4	2	1

Ασκήσεις:

1) Ποιος είναι ο αριθμός 11111111_b στο δεκαδικό αριθμητικό σύστημα?

Απάντηση: 128+64+32+16+8+4+2+1=255 **(Σημαντικό, να το θυμόμαστε !!!)**

Σημειώστε ότι ο παραπάνω αριθμός είναι ο μεγαλύτερος αριθμός που μπορούμε να γράψουμε στο δυαδικό σύστημα με 8 ψηφία.

1) Ποιος είναι ο αριθμός 11111100_b στο δεκαδικό αριθμητικό σύστημα?

Απάντηση: 128+64+32+16+8+4+0+0=252

1) Ποιος είναι ο αριθμός 11000000 στο δεκαδικό αριθμητικό σύστημα?

Απάντηση: 128+64+0+0+0+0+0+0=192

1) Ποιος είναι ο δυαδικός αριθμός 11100000 στο δεκαδικό αριθμητικό σύστημα?

Απάντηση: 128+64+32+0+0+0+0+0=224

Μετατροπή ενός αριθμού του δεκαδικού αριθμητικού συστήματος στο δυαδικό αριθμητικό σύστημα (1)

1) Γράψτε τον αριθμό 192_d στο δυαδικό αριθμητικό σύστημα

Απάντηση:

ψηφίο								
αξία	128	64	32	16	8	4	2	1

Πρέπει να βρούμε τις αξίες που όταν προστεθούν θα μας δώσουν τον αριθμό 192_d

Τον δείκτη $_d$ τον βάλαμε για να θυμίσουμε ότι είναι ένας αριθμός του δεκαδικού αριθμητικού συστήματος.

192=128+64

ψηφίο	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
αξία	128	64	32	16	8	4	2	1

Ο αριθμός 192_d στο δυαδικό αριθμητικό σύστημα γράφεται: 11000000_b

Μετατροπή ενός αριθμού του δεκαδικού αριθμητικού συστήματος στο δυαδικό αριθμητικό σύστημα (2)

2) Να γράψετε τον αριθμό 168_d στο δυαδικό αριθμητικό σύστημα.

Απάντηση:

ψηφίο	**1**		**1**		**1**			
αξία	128	64	32	16	8	4	2	1

Πρέπει να βρεθούν οι αξίες που όταν προστεθούν θα δώσουν την τιμή 168.

168=128+32+8

Ο αριθμός 168_d στο δυαδικό αριθμητικό σύστημα γράφεται: 10101000_b

Δεκαεξαδικό αριθμητικό σύστημα (Hexadecimal arithmetic system) (1)

Στο δεκαδικό αριθμητικό σύστημα χρησιμοποιούνται 16 (0,1,2,…9, A, B, C, D, E, F) ψηφία και η αξία κάθε ψηφίου αυξάνει από τα δεξιά προς τα αριστερά κατά δύναμη του 16.

Σ’ αυτό το αριθμητικό σύστημα η αξία του κάθε ψηφίου είναι μια δύναμη του 16.

Το τελευταίο δεξιά ψηφίο είναι μονάδες($16^0=1$)

Το προτελευταίο από δεξιά είναι δεκαεξάδες($16^1=16$)

Το τρίτο από δεξιά είναι διακοσιαπενταεξάδες($16^2=256$)

Το τέταρτο από δεξιά έχει αξία($16^3=4096$)

ψηφίο						**2**	**5**	**6**
αξία	16^7	16^6	16^5	16^4	16^3	16^2	16^1	1

$256_h = 2X16^2 + 5X16^1 + 6X16^0 = 2X256 + 5X16 + 6X1 = 598_d$

Δεκαεξαδικό αριθμητικό σύστημα (Hexadecimal arithmetic system) (2)

Ψηφία του δεκαεξαδικού συστήματος:

Πως γράφεται ο αριθμός 11h στο δεκαδικό αριθμητικό σύστημα;

1 δεκαεξάδα και 1 μονάδα, γράφεται 17_d

Πως γράφεται ο αριθμός A1h στο δεκαδικό αριθμητικό σύστημα;

10 δεκαεξάδες και 1 μονάδα, επομένως γράφεται 161_d

Πως γράφεται ο αριθμός FFh στο δεκαδικό αριθμητικό σύστημα;

15 δεκαεξάδες και 15 μονάδες, επομένως γράφεται 255_d
$(15X16+15=240+15=255_d)$

Πως γράφεται ο αριθμός (123)h στο δεκαδικό αριθμητικό σύστημα;
1 $(δεκαεξάδα)^2$ και 2 δεκαεξάδες και 3 μονάδες, επομένως γράφεται 291_d
$16^2 + 2X16 + 3 = 256 + 32 + 3 = 291_d$

Δεκαδικό σύστημα	Δεκαεξαδικό σύστημα
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Μετατροπή από το δεκαεξαδικό αριθμητικό σύστημα στο δεκαδικό αριθμητικό σύστημα

Παράδειγμα 1.

Να γραφεί ο αριθμός $320F_h$ στο δεκαδικό αριθμητικό σύστημα.

$320F_h=3x16^3+2x16^2+0x16^1+15x16^0=3x4096+2x256+0x16+15x1=12288+512+0+15=12815_d$

Επομένως: $320F_h=12815_d$

Παράδειγμα 2.

Να γραφεί ο αριθμός FF_h στο δεκαδικό αριθμητικό σύστημα.

$FF_h=15x16^1+15x16^0=15x16+15x1=240+15=255_d$

Παράδειγμα 3.

Να γραφεί ο αριθμός 100_h στο δεκαδικό αριθμητικό σύστημα.

$100_h=1x16^2+0x16^1+0x16^0=256+0+0=256_d$

Γιατί χρησιμοποιούμε το δεκαεξαδικό αριθμητικό σύστημα;

Γιατί χρησιμοποιείται το δεκαεξαδικό σύστημα στους υπολογιστές αφού οι υπολογιστές χρησιμοποιούν αποκλειστικά το δυαδικό αριθμητικό σύστημα και οι άνθρωποι στις καθημερινές τους συναλλαγές το δεκαδικό αριθμητικό σύστημα;

Ο λόγος για τον οποίο κατά τη μελέτη των υπολογιστικών συστημάτων χρησιμοποιείται το δυαδικό αριθμητικό σύστημα είναι ότι είναι πολύ εύκολη η μετατροπή από το δεκαεξαδικό αριθμητικό σύστημα στο δυαδικό και από το δυαδικό αριθμητικό σύστημα στο δεκαεξαδικό.

Επιπλέον μπορούμε να γράφουμε δυαδικούς αριθμούς με μεγάλο αριθμό ψηφίων χρησιμοποιώντας λίγα δεκαεξαδικά ψηφία(Το ¼ των ψηφίων που χρησιμοποιούνται για την γραφή του ίδιου αριθμού στο δυαδικό αριθμητικό σύστημα).

Δεν ξεχνάμε: Στα υπολογιστικά συστήματα χρησιμοποιούνται μόνο τα δύο ψηφία του δυαδικού αριθμητικού συστήματος και τίποτα άλλο.

Μετατροπή από το δυαδικό στο δεκαεξαδικό αριθμητικό σύστημα

Παράδειγμα 1, μετατροπή από το δυαδικό στο δεκαεξαδικό αριθμητικό σύστημα.

Να γραφεί ο αριθμός $100100100101010000111 1_b$ στο δεκαεξαδικό αριθμητικό σύστημα.

Απάντηση:

Χωρίζουμε τον αριθμό σε τετράδες ψηφίων από δεξιά προς τα αριστερά και σε κάθε τετράδα αντιστοιχούμε ένα δεκαεξαδικό ψηφίο.

$100100100101010000111 1_b$ → 10 0100 1001 0101 0000 1111_b → 2 4 9 5 0 F_h → $(24950F)_h$

Μετατροπή από το δεκαεξαδικό στο δυαδικό αριθμητικό σύστημα (1)

Παράδειγμα 2 μετατροπή από το δεκαεξαδικό στο δυαδικό αριθμητικό σύστημα.

Να γραφεί ο αριθμός $25C6FA_h$ στο δυαδικό αριθμητικό σύστημα.

Απάντηση: Σε κάθε δεκαεξαδικό ψηφίο αντιστοιχούμε 4 δυαδικά ψηφία σύμφωνα με τον παρακάτω πίνακα.

Ο αριθμός $25C6FA_h$ θα γραφεί στο δυαδικό σύστημα.

$0010\ 0101\ 1100\ 0110\ 1111\ 1010_b$

Τετράδα δυαδικών ψηφίων	Δεκαεξαδικό ψηφίο	Αντίστοιχος αριθμός του δεκαδικού συστήματος
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Μετατροπή από το δεκαεξαδικό στο δυαδικό αριθμητικό σύστημα

Παράδειγμα 3.

Να γραφεί ο αριθμός $3FF_h$ στο δυαδικό σύστημα.

$0011\ 1111\ 1111_b \rightarrow 11\ 1111\ 1111_b \rightarrow 1111111111_b$

Πόσοι αριθμοί υπάρχουν από $00\ 0000\ 0000_b$ έως $11\ 1111\ 1111_b$;

$11\ 1111\ 1111_b \rightarrow 3FF_h =$

$=3x16^2+15x16^1+15x16^0$

$=3x256+15x16+15x1=768+240+15=1023_d$

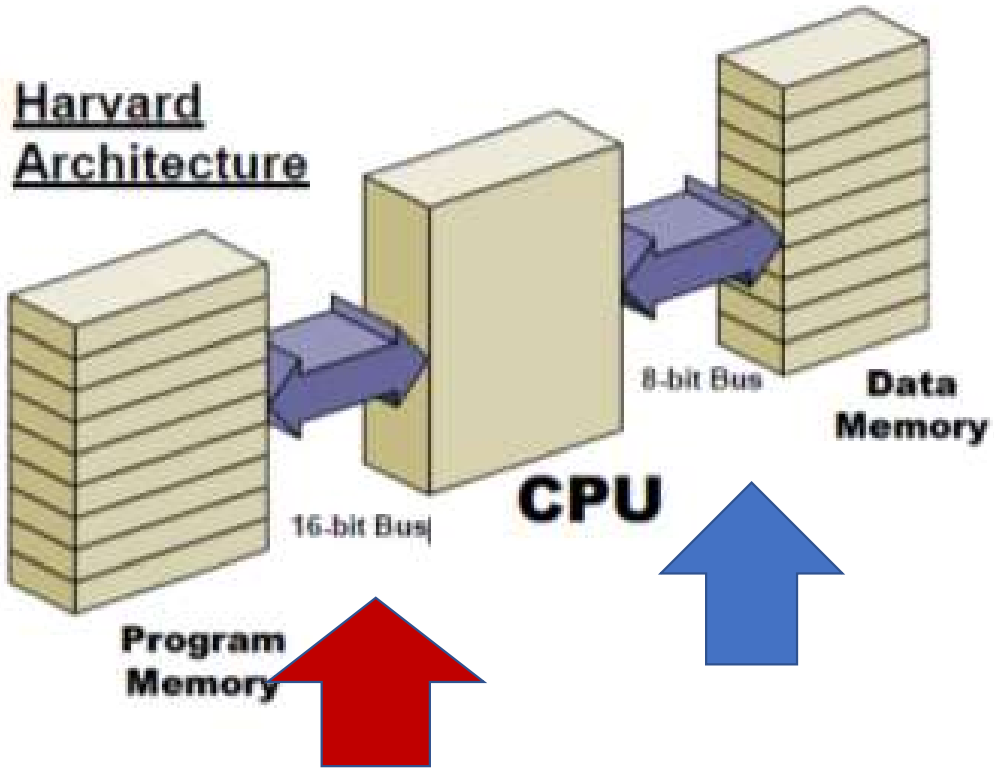
Υπάρχουν 1024_d αριθμοί(διότι περιλαμβάνεται και το μηδέν).

Προσέξτε: $1024= 2^{10}$

Με 10 δυαδικά ψηφία μπορούμε να σχηματίσουμ 2^{10} αριθμούς(=1024), 1Κ στα ψηφιακά.

Τετράδα δυαδικών ψηφίων	Δεκαεξαδικό ψηφίο	Αντίστοιχος αριθμός του δεκαδικού συστήματος
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Αρχιτεκτονική του μικροελεγκτή PIC18F4550



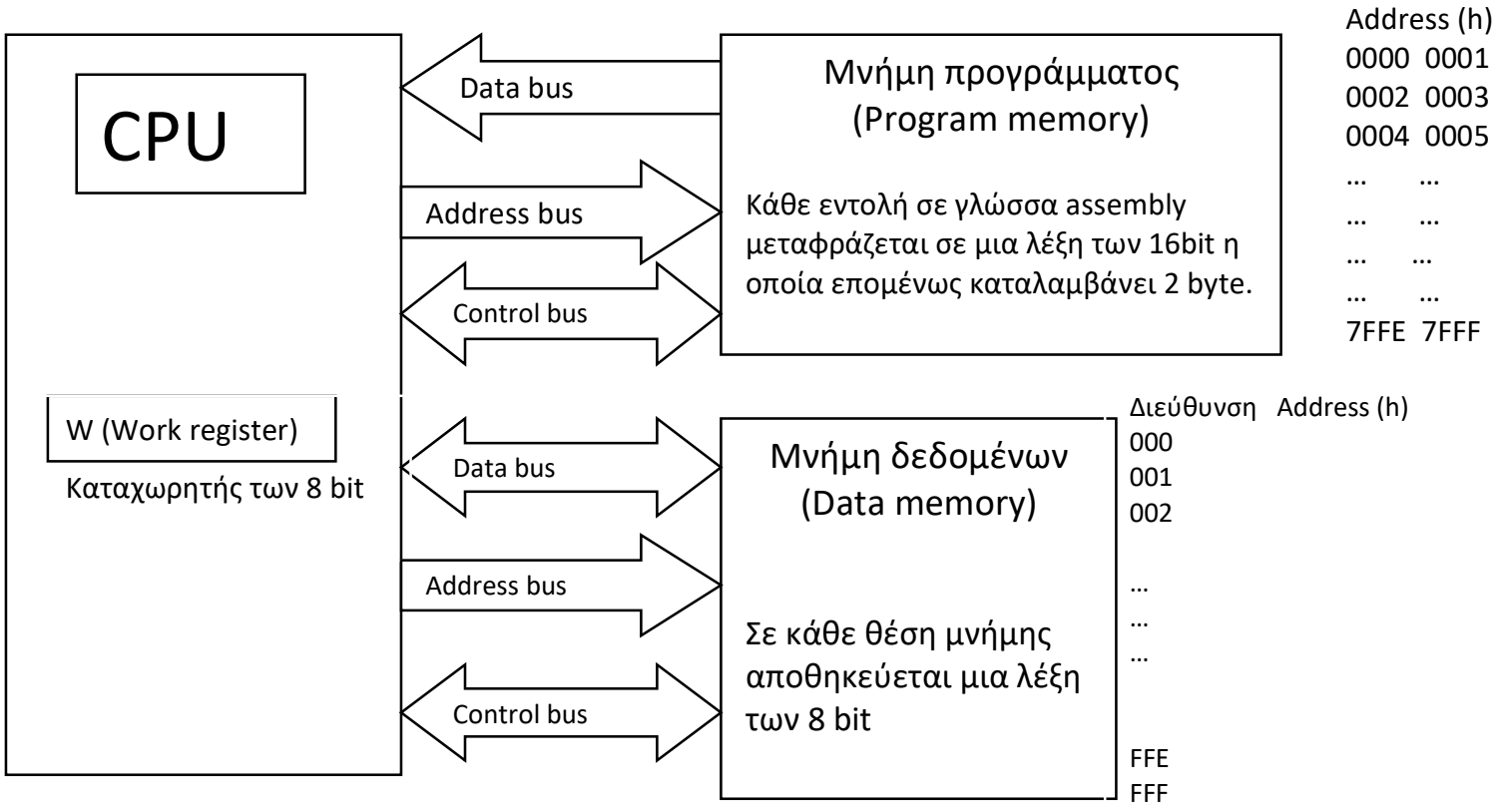
Ο δίαυλος δεδομένων προς τη μνήμη δεδομένων είναι των 8 bit

Ο δίαυλος δεδομένων προς τη μνήμη προγράμμαμος είναι 16 bit.

Επομένως μεταφέρονται συγχρόνως από τη μνήμη προγράμματος 16 bit.

Οι εντολές σε γλώσσα μηχανής είναι κατά κανόνα λέξεις των 16 bit

Σύνδεση της Κεντρικής Μονάδας Επεξεργασίας (CPU) με τη μνήμη προγράμματος και τη μνήμη δεδομένων



Γιατί στην αρίθμηση των θέσεων μνήμης χρησιμοποιούμε το δεκαεξαδικό αριθμητικό σύστημα;

Μπορούμε να τοποθετήσουμε περιεχόμενο σε μια θέση μνήμης ή να διαβάσουμε το περιεχόμενο από μια θέση μνήμης (Write / Read)

Data Bus
8 bit
Δίαυλος
δεδομένων

8 αγωγοί για μεταφορά του περιεχομένου της μνήμης

Περιεχόμενα θέσεων μνήμης (b)	Διευθύνσεις στο δεκαεξαδικό αριθμητικό σύστημα (h)	Διευθύνσεις στο δυαδικό αριθμητικό σύστημα (b)	Διευθύνσεις στο δεκαδική αριθμητικό σύστημα (d)
0101 0101	000	0000 0000 0000	0
1111 1111	001	0000 0000 0001	1
1000 1110	002	0000 0000 0010	2
. . .	. . .	. . .	. . .
1010 1111	0FE	0000 1111 1110	254
1001 1010	0FF	0000 1111 1111	255
1111 0000	100	0001 0000 0000	256
1111 1000	101	0001 0000 0001	257

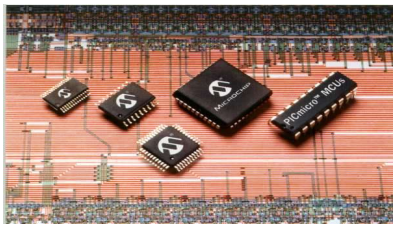
Address Bus
12 bit
Δίαυλος
διευθύνσεων

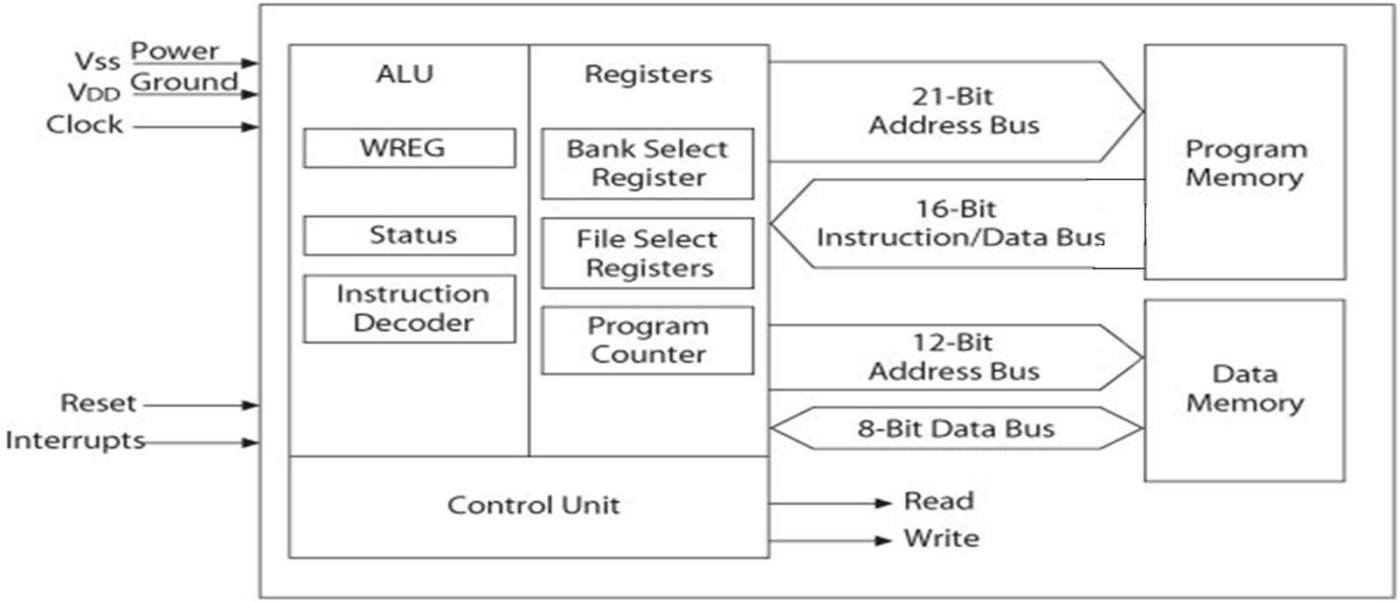
1----------
1----------
0----------
0----------
1----------
1----------
0----------
0----------
0----------
0----------
1----------
0----------

Διεύθυνση $CC2_h$ (Αποδίδεται από την Κεντρική Μονάδα επεξεργασίας, CPU)

12 αγωγοί για επιλογή της διεύθυνσης από την οποία θα γίνει η ανάγνωση ή η εγγραφή δεδομένων

PIC18F – MCU and Memory





Program Memory: 32 K (2^{15})

Περιοχή διευθύνσεων: 000000 to $007FFF_h$

Καταχωρητές των 16-bit

Data Memory: 2 K

Περιοχή διευθύνσεων: 000 to $7FF_h$

Καταχωρητές των 8 bit

Η μνήμη προγράμματος είναι οργανωμένη σε λέξεις των 16 bit

Ο δίαυλος διευθύνσεων των 21 bit θα μπορούσε να έχει πρόσβαση σε

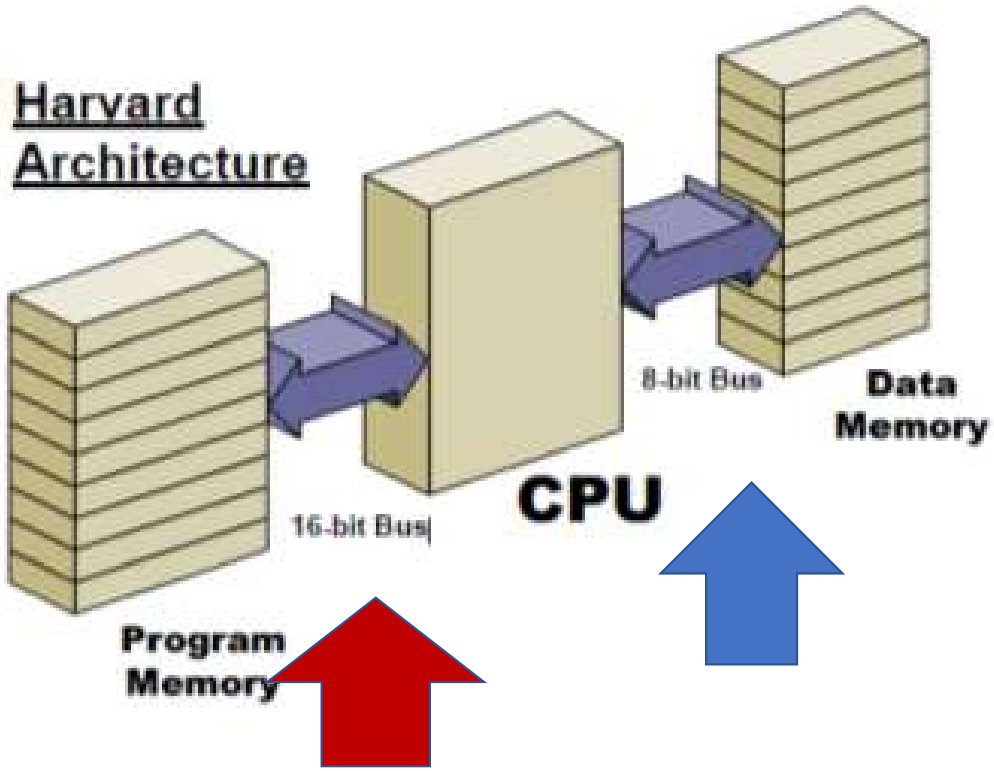
2 MB
2^{21}

Η μνήμη δεδομένων είναι οργανωμένη σε λέξεις των 8 bit

Ο δίαυλος διευθύνσεων των 12 bit θα μπορούσε να έχε πρόσβαση σε

4KB
2^{12}

Αρχιτεκτονική του μικροελεγκτή PIC18F4550



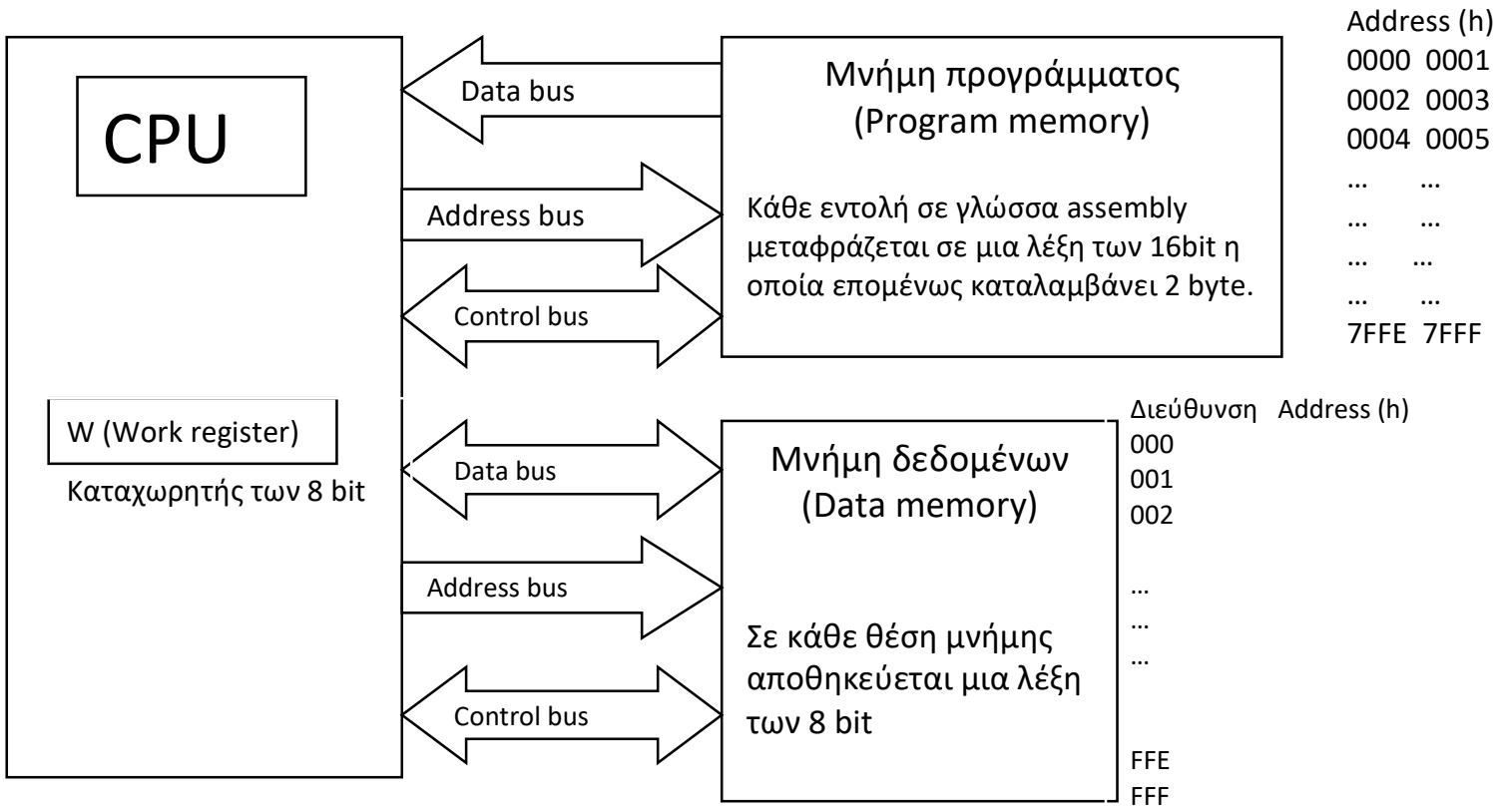
Ο δίαυλος δεδομένων προς τη μνήμη δεδομένων είναι των 8 bit

Ο δίαυλος δεδομένων προς τη μνήμη προγράμμαμος είναι 16 bit.

Επομένως μεταφέρονται συγχρόνως από τη μνήμη προγράμματος 16 bit.

Οι εντολές σε γλώσσα μηχανής είναι κατά κανόνα λέξεις των 16 bit

Σύνδεση της Κεντρικής Μονάδας Επεξεργασίας (CPU) με τη μνήμη προγράμματος και τη μνήμη δεδομένων



Γιατί στην αρίθμηση των θέσεων μνήμης χρησιμοποιούμε το δεκαεξαδικό αριθμητικό σύστημα;

Μπορούμε να τοποθετήσουμε περιεχόμενο σε μια θέση μνήμης ή να διαβάσουμε το περιεχόμενο από μια θέση μνήμης (Write / Read)

Data Bus
8 bit
Δίαυλος
δεδομένων

8 αγωγοί για μεταφορά του περιεχομένου της μνήμης

Περιεχόμενα θέσεων μνήμης (b)	Διευθύνσεις στο δεκαεξαδικό αριθμητικό σύστημα (h)	Διευθύνσεις στο δυαδικό αριθμητικό σύστημα (b)	Διευθύνσεις στο δεκαδική αριθμητικό σύστημα (d)
0101 0101	000	0000 0000 0000	0
1111 1111	001	0000 0000 0001	1
1000 1110	002	0000 0000 0010	2
. . .	. . .	. . .	. . .
1010 1111	0FE	0000 1111 1110	254
1001 1010	0FF	0000 1111 1111	255
1111 0000	100	0001 0000 0000	256
1111 1000	101	0001 0000 0001	257

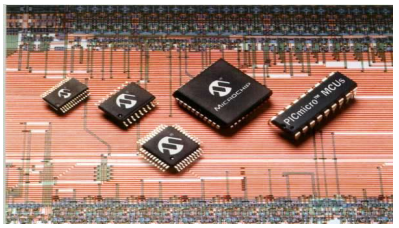
Address Bus
12 bit
Δίαυλος
διευθύνσεων

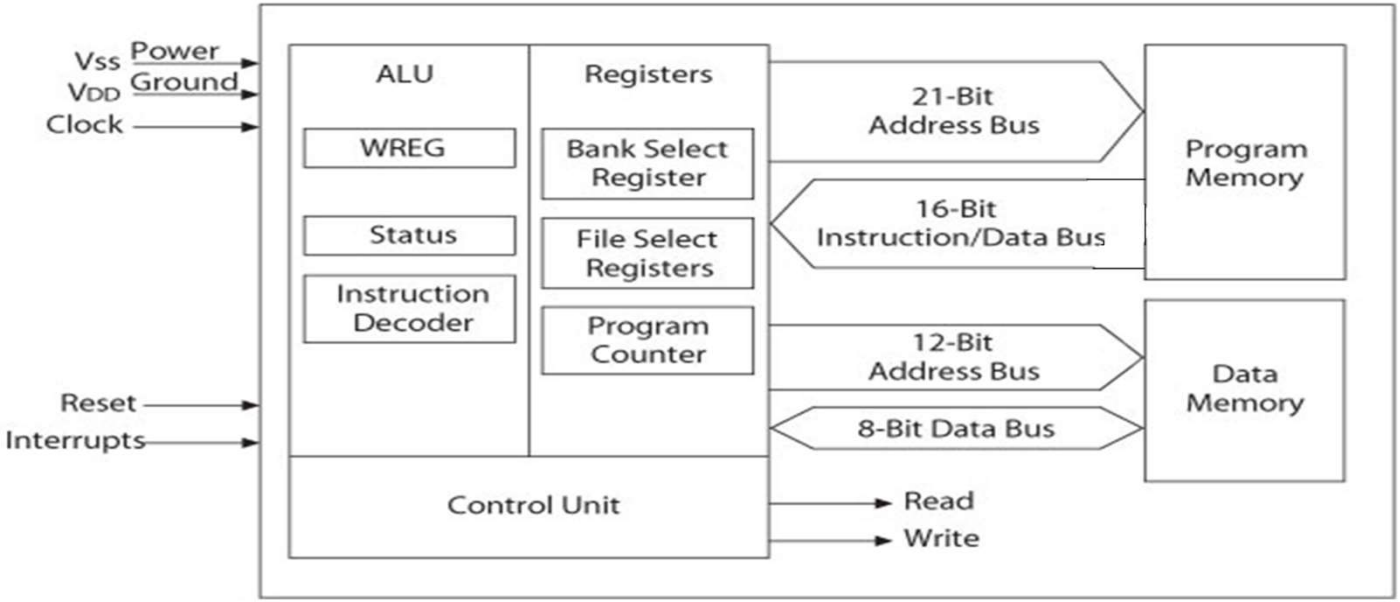
1----------
1----------
0----------
0----------
1----------
1----------
0----------
0----------
0----------
0----------
1----------
0----------

Διεύθυνση $CC2_h$ (Αποδίδεται από την Κεντρική Μονάδα επεξεργασίας, CPU)

12 αγωγοί για επιλογή της διεύθυνσης από την οποία θα γίνει η ανάγνωση ή η εγγραφή δεδομένων

PIC18F – MCU and Memory





Program Memory: 32 K (2^{15})

Περιοχή διευθύνσεων: 000000 to $007FFF_h$

Καταχωρητές των 16-bit

Data Memory: 2 K

Περιοχή διευθύνσεων: 000 to $7FF_h$

Καταχωρητές των 8 bit

Η μνήμη προγράμματος είναι οργανωμένη σε λέξεις των 16 bit

Ο δίαυλος διευθύνσεων των 21 bit θα μπορούσε να έχει πρόσβαση σε

2 MB
2^{21}

Η μνήμη δεδομένων είναι οργανωμένη σε λέξεις των 8 bit

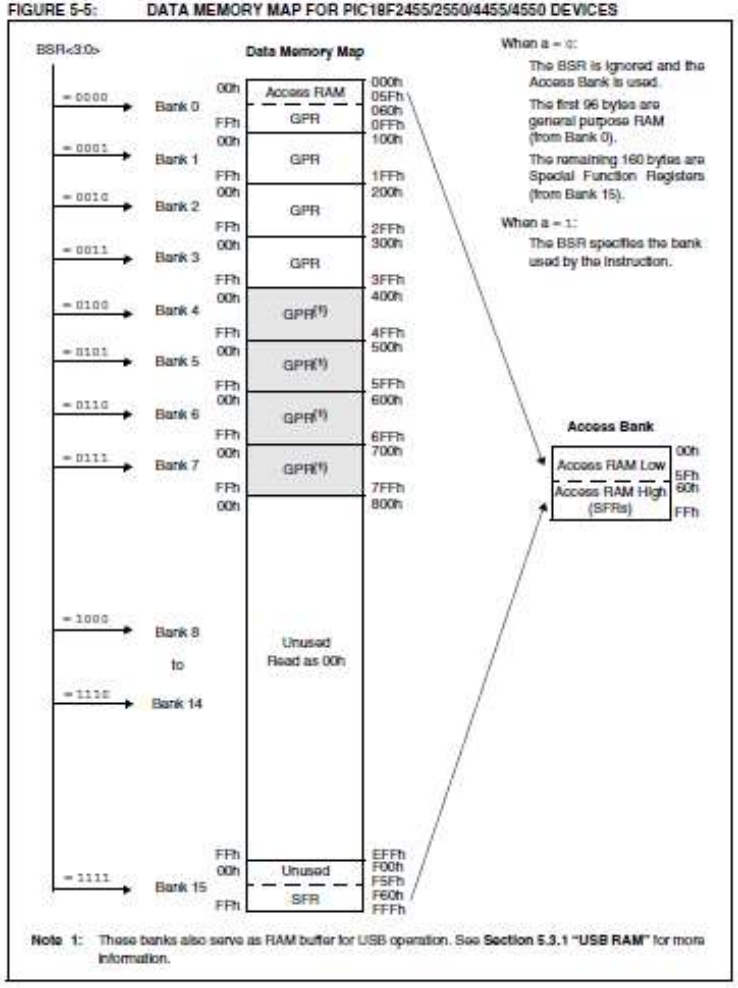
Ο δίαυλος διευθύνσεων των 12 bit θα μπορούσε να έχε πρόσβαση σε

4KB
2^{12}

Η μνήμη δεδομένων του μικροελεγκτή PIC18F4550
Data memory of the PIC18F4550 microcontroller

PIC18F2455/2550/4455/4550

FIGURE 5-5: DATA MEMORY MAP FOR PIC18F2455/2550/4455/4550 DEVICES



Note 1: These banks also serve as RAM buffer for USB operation. See Section 5.3.1 "USB RAM" for more information.

Η μνήμη δεδομένων του μικροελεγκτή είναι από τη Διεύθυνση (000)h έως (7FF)h.

Δηλαδή από 0000 0000 0000 έως 111 1111 1111

Το πλήθος των διευθύνσεων είναι $2^{11} = 2^1 \times 2^{10} = 2 \times 1024 = 2K$
Σε κάθε θέση μνήμης αποθηκεύεται ένα byte (8bit).

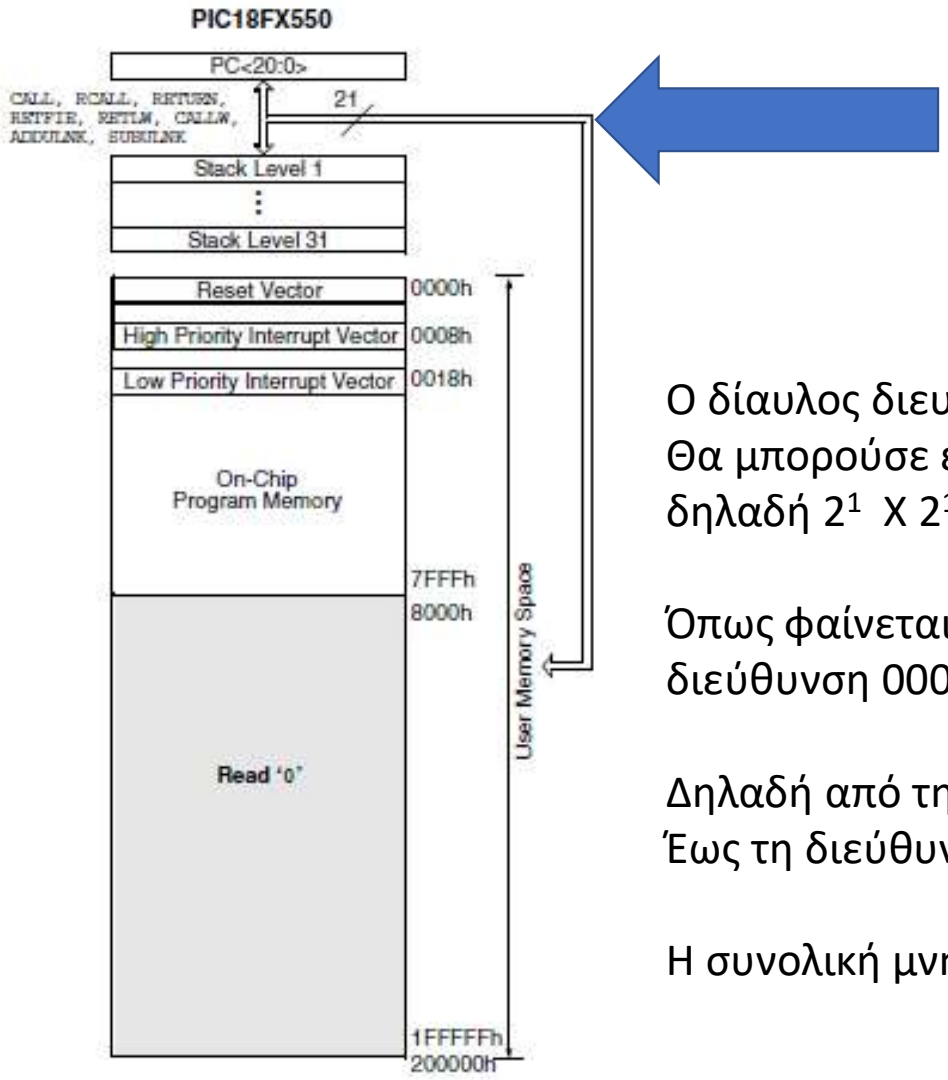
Η μνήμη δεδομένων του Μικροελεγκτή PIC 18F4550 είναι 2K

Προσοχή! $2^{10} = 1024 = 1K$ για τις μνήμες
$2^{20} = 1K \times 1K = 1\ M$ (1 048 576 θέσεις μνήμης)
$2^{30} = 1K \times 1K \times 1K = 1G$ (1 073 741 824 θέσεις μνήμης

Προσοχή, χρησιμοποιούνται και οι θέσεις μνήμης $(F60)_h$ έως $(FFF)_h$.

Αυτές οι θέσεις μνήμης χρησιμοποιούνται για τους Special Function Registers (SFR). Είναι θέσης μνήμης που έχουν μια ειδική λειτουργία.

Μνήμη προγράμματος του μικροελεγκτή PIC18F4550



Προσοχή: Οι τιμές στο δίαυλο διευθύνσεων της μνήμης προγράμματος τοποθετούνται από τον Program Counter . Ο Program Counter είναι ένας καταχωρητής τοποθετημένος μέσα στην Κεντρική Μονάδα Επεξεργασίας (CPU).

Ο δίαυλος διευθύνσεων περιλαμβάνει 21 αγωγούς.
Θα μπορούσε επομένως να έχει πρόσβαση σε 2^{21} θέσει μνήμης,
δηλαδή 2^1 X 2^{10} X 2^{10}=2 X 1K X 1K = 2M θέσεις μνήμης.

Όπως φαίνεται στο σχήμα το υλοποιημένο κομμάτι της μνήμης είναι από τη διεύθυνση 0000_h έως τη διεύθυνση $7FFF_h$

Δηλαδή από τη διεύθυνση 0000 0000 0000 0000
Έως τη διεύθυνση 111 1111 1111 1111

Η συνολική μνήμη προγράμματος είναι 2^{15} =2^5 X 2^{10} = 32 K

Υπολογισμός πλήθους θέσεων μνήμης, καθορισμός Bank

Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **000h** έως τη διεύθυνση **0FFh**;

Απάντηση: FF +1 (διότι περιλαμβάνεται και η μηδενική διεύθυνση)

FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 0**

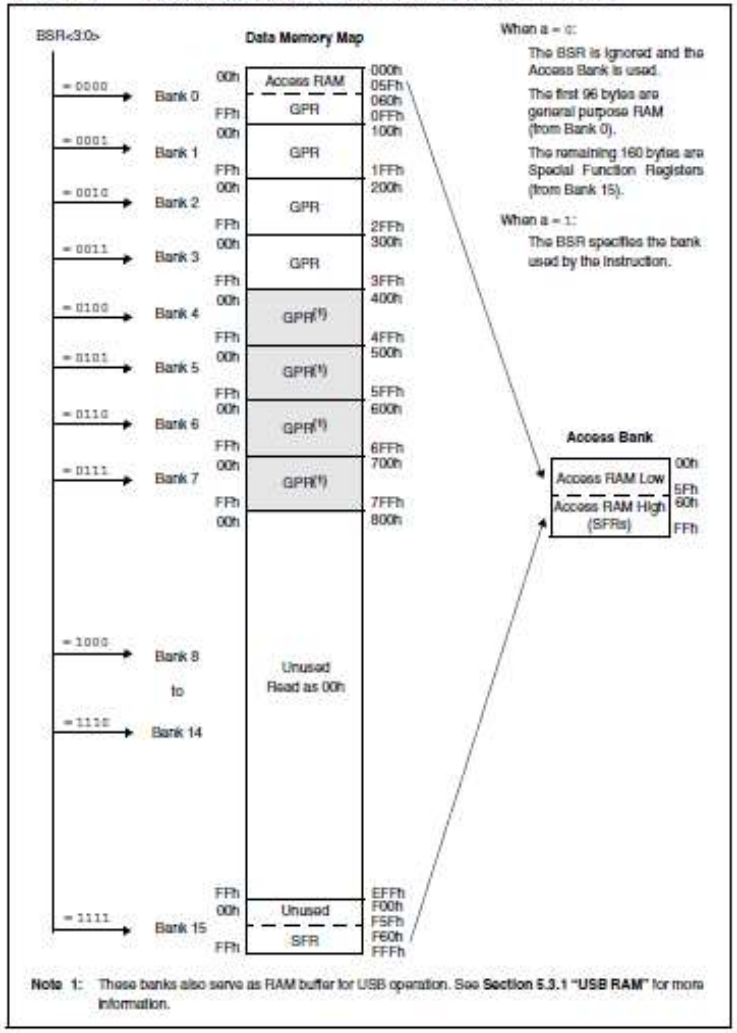
Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **100h** έως τη διεύθυνση **1FFh**;
Απάντηση: 1FF - 100= FF
FF +1 (διότι περιλαμβάνεται και η διεύθυνση 100)
Οι θέσεις μνήμης είναι:
FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 1**

Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **200h** έως τη διεύθυνση **2FFh**;
Απάντηση: 2FF - 200= FF
FF +1 (διότι περιλαμβάνεται και η διεύθυνση 100)
Οι θέσεις μνήμης είναι:
FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 2**

Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **300h** έως τη διεύθυνση **3FFh**;
Απάντηση: 3FF - 300= FF
FF +1 (διότι περιλαμβάνεται και η διεύθυνση 100)
Οι θέσεις μνήμης είναι:
FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 3**

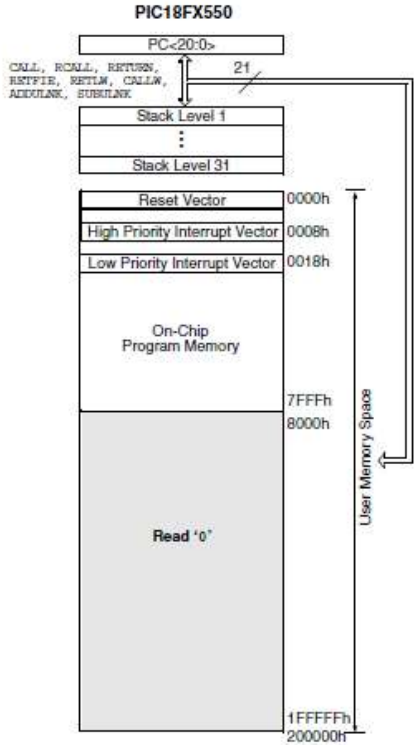
PIC18F2455/2550/4455/4550

FIGURE 5-5: DATA MEMORY MAP FOR PIC18F2455/2550/4455/4550 DEVICES



Δημιουργία προγράμματος σε γλώσσα Assembly (1)

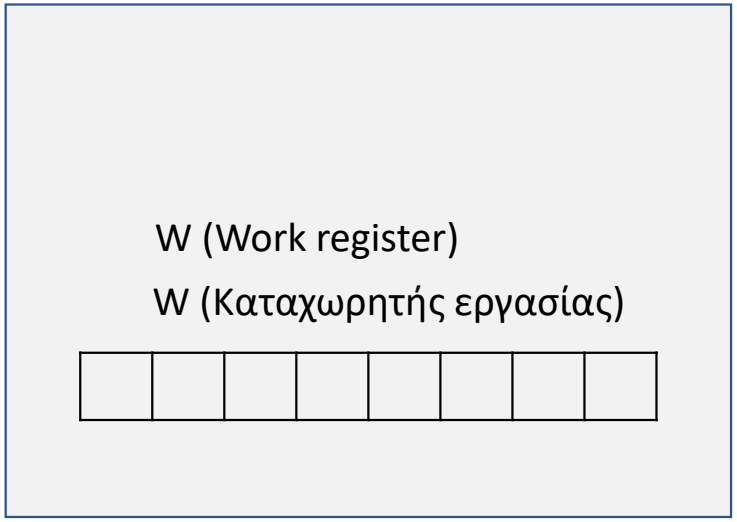
Να γραφεί πρόγραμμα με το οποίο στη θέση μνήμης 060h τοποθετείται η τιμή 2Ah και στη θέση μνήμης 061h η τιμή FF και στη συνέχεια το περιεχόμενο της θέσης μνήμης 060h μεταφέρεται στη θέση μνήμης 061h. Μετά την εκτέλεση των παραπάνω το πρόγραμμα να εκτελεί έναν ατέρμονα(χωρίς τέλος) βρόχο χωρίς να εκτελεί κάποια εντολή. Το πρόγραμμα να τοποθετηθεί στη μνήμη προγράμματος από τη διεύθυνση 0800h και μετά

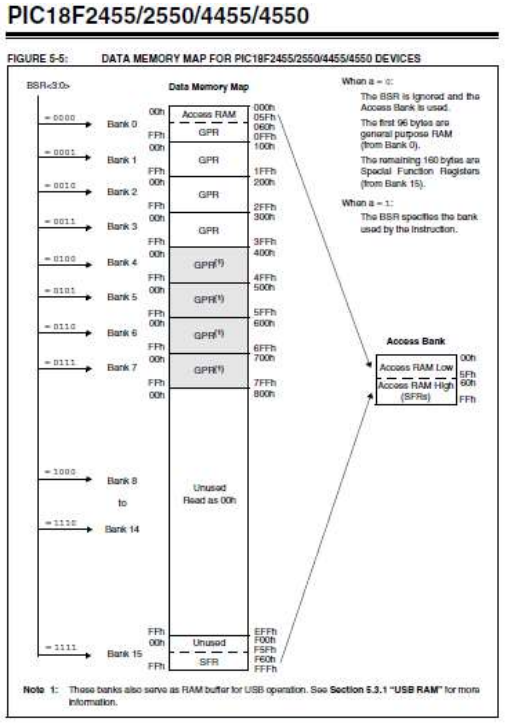


Μνήμη προγράμματος

Το πρόγραμμα θα γραφεί από τη διεύθυνση 0800 h και μετά. (Πώς θα το τοποθετήσουμε σ' αυτή τη θέση;

CPU (Central Processing Unit
ΚΜΕ (κεντρική Μονάδα Επεξεργασίας)





Μνήμη δεδομένων

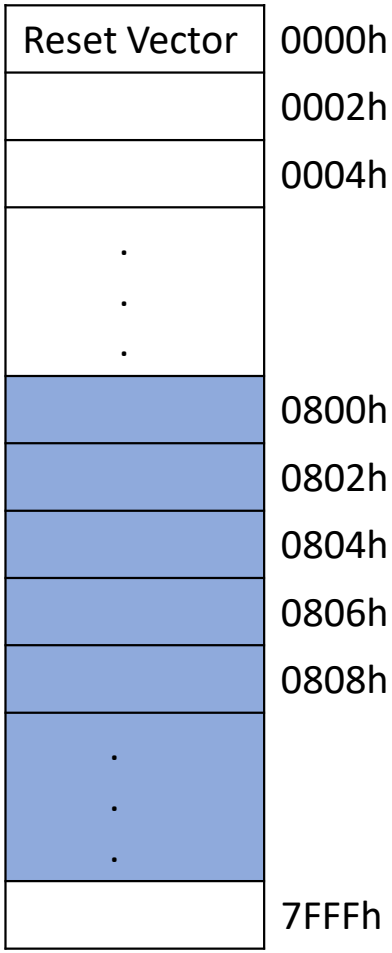
$2A_h$ →Να γίνει περιεχόμενο της θέσης μνήμης 060_h

FF_h → Να γίνει περιεχόμενο της θέσης μνήμης 061_h

Το περιεχόμενο της θέσης μνήμης 060_h να μεταφερθεί στο περιεχόμενο της θέσης μνήμης 061_h

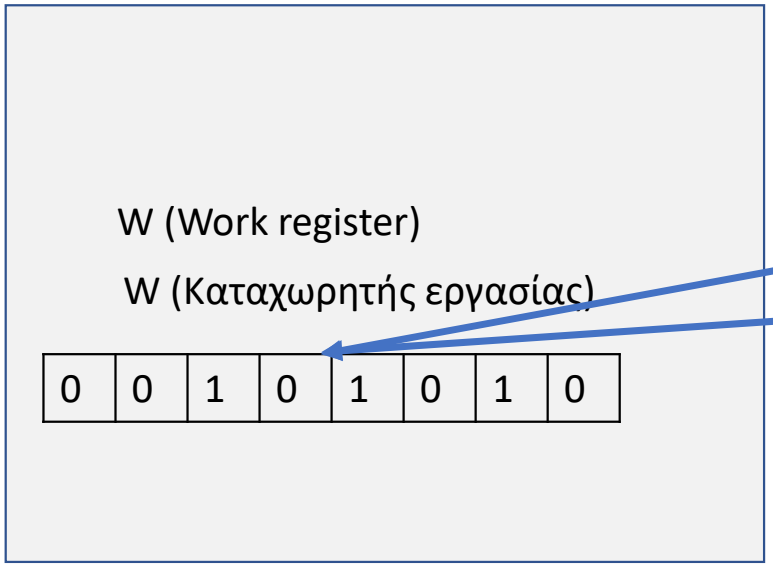
Δημιουργία προγράμματος σε γλωσσα Assembly (2)

Μνήμη προγράμματος
16 bit Instructions
Εντολές των 16 bit

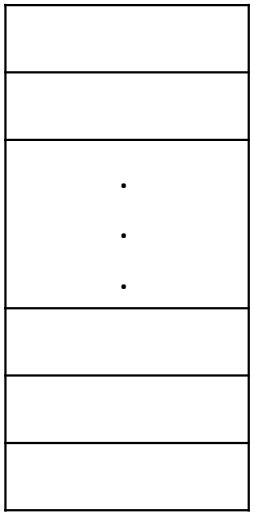


Το πρόγραμμα θα γραφεί από τη διεύθυνση 0800h και μετά

CPU (Central Processing Unit
ΚΜΕ (κεντρική Μονάδα Επεξεργασίας)



Μνήμη δεδομένων
Οργανωμένη σε λέξεις των 8 bit

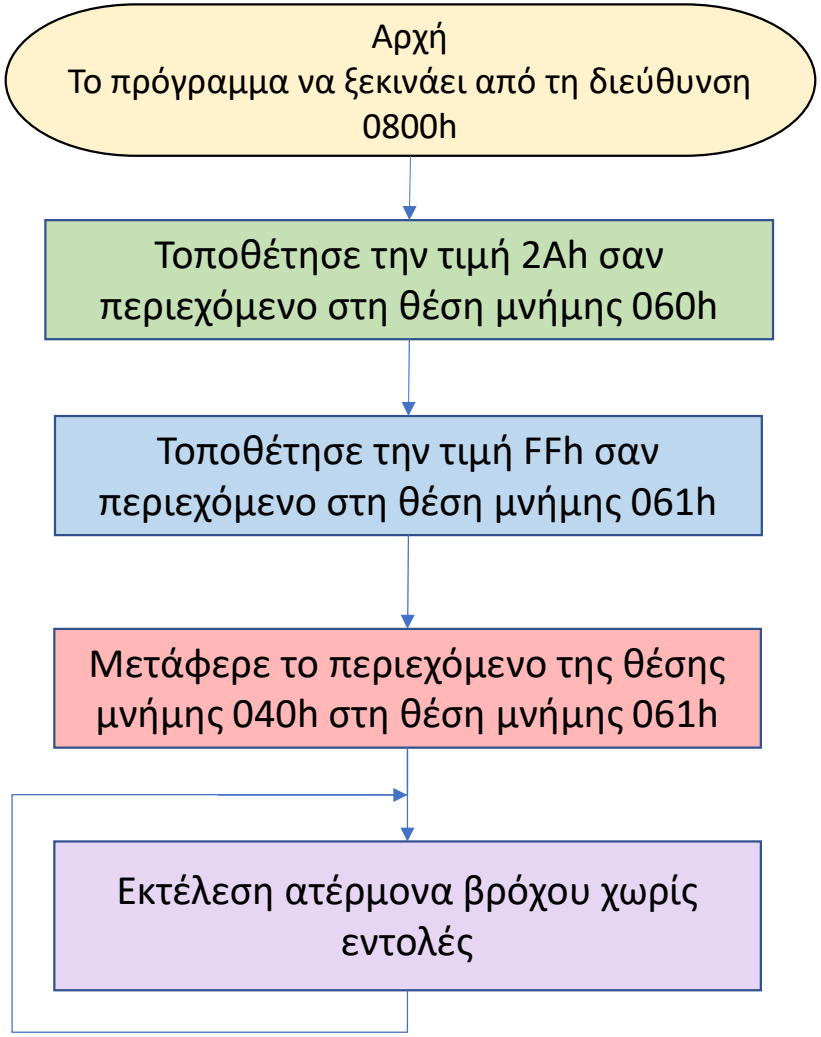


Μνήμη δεδομένων

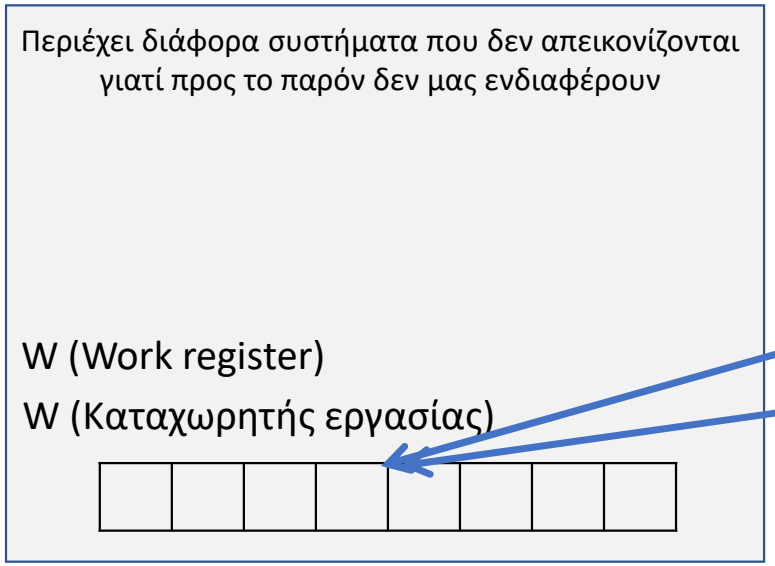
Η τοποθέτηση τιμών στις θέσεις τις μνήμης δεδομένων και η μεταφορά τιμών από και προς τις θέσεις της μνήμης δεδομένων γίνεται δια μέσου του καταχωρητή εργασίας W

Διάγραμμα ροής το προγράμματος (3)

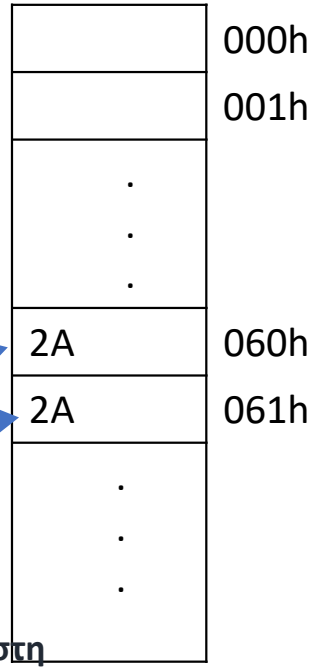
Διάγραμμα ροής του προγράμματος



CPU (Central Processing Unit
ΚΜΕ (κεντρική Μονάδα Επεξεργασίας)



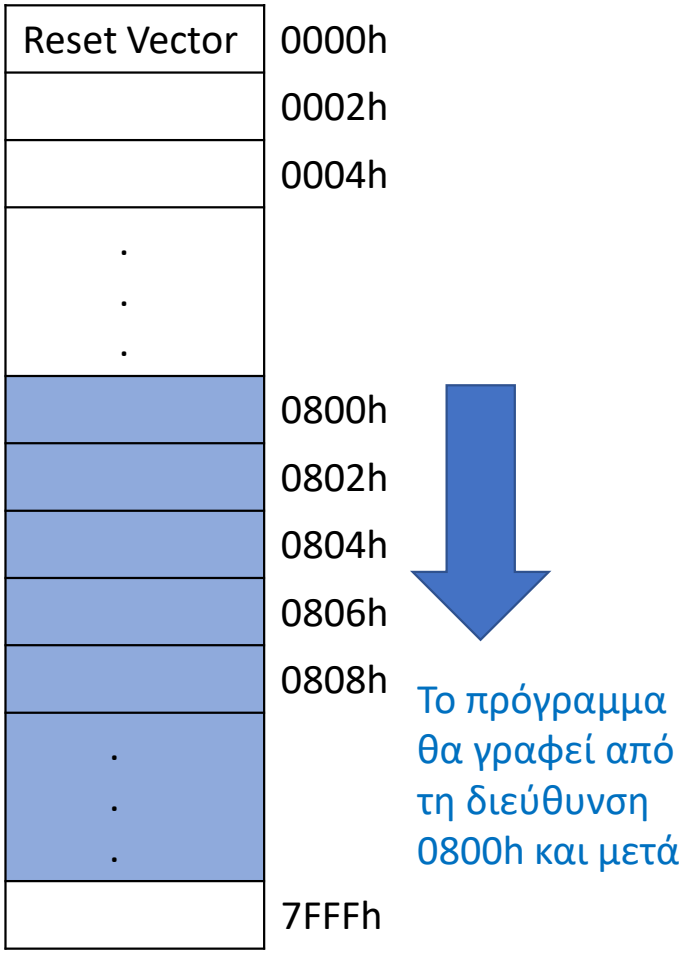
Μνήμη δεδομένων
Οργανωμένη σε λέξεις των 8 bit



Πρόγραμμα

```
org 0x800          ; Αρχική διεύθυνση του προγράμματος στη
                   ; μνήμη προγράμματος
        movlw 0x2A     ; move literal to W:  0x2A  → W
        movwf 0x060    ; move W to file 0x060:  W → 0x060
        movlw 0XFF     ; move literal to W:  0xFF → W
        movwf 0x061    ; move W to file :  W → 0x061
        movf 0x060, 0  ; move file 0x060 to W:  0x060 → W
        movwf 0x061    ; move w to file 0x061:  W →0x061
loop    goto loop      ; ατέρμων βρόχος χωρίς εντολή
END
```

Μνήμη προγράμματος
16 bit Instructions
Εντολές των 16 bit



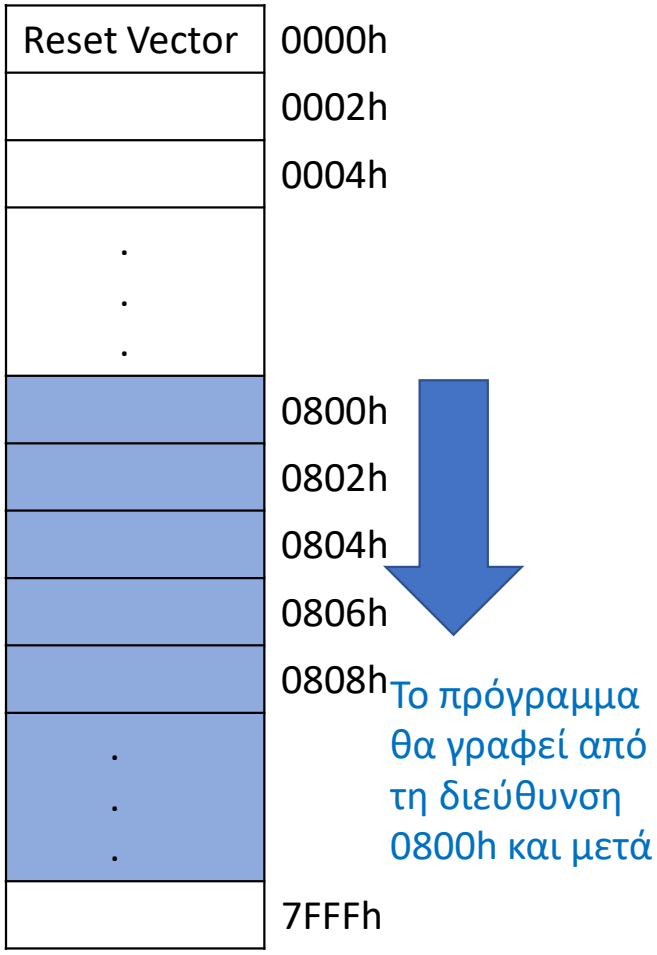
Ποιος θα είναι ο κώδικας του προγράμματος; (1)
Πως μπορούμε να δούμε τον κώδικα του προγράμματος στο MPLAB;

Πρόγραμμα

```
org 0x800        ; Τοποθετούμαι τη διεύθυνση στη μνήμη προγράμματος
                 ; από την οποία θέλουμε
                 ; να ξεκινάει το πρόγραμμα μας. Είναι οδηγία προς
                 ; τον Assembler
movlw 0x2A       ; move literal to W:  0x2A  → W
                 ;μεταφέρεται η τιμή 0x2A στον W
movwf 0x060      ; move W to file 0x060:  W → 0x060
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x060
movlw 0XFF       ; move literal to W:  0xFF → W
                 ; Μεταφέρεται η τιμή 0xFF στον καταχωρητή W
movwf 0x061      ; move W to file :  W → 0x061
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x041
movf 0x060, 0    ; move file 0x040 to W:  0x060 → W
                 ; Μεταφέρεται το περιεχόμενο της θέσης μνήμης 0x060 στον W
movwf 0x061      ; move w to file 0x041:  W →0x061
                 ; Μετάφερε το περιεχόμενο του W  στη θέση μνήμης 0X061
loop goto loop   ; ατέρμων βρόχος χωρίς εντολή
END              ; Οδηγία προς τον Assembler  ότι εδώ τελειώνει το πρόγραμμα
                 ; Που θα μεταφραστεί σε γλώσσα μηχανής
```

Ποιος θα είναι ο κώδικας του προγράμματος; (2)
Πως μπορούμε να δούμε τον κώδικα του προγράμματος στο MPLAB;

Μνήμη προγράμματος
16 bit Instructions
Εντολές των 16 bit



Κάθε εντολή του προγράμματος θα αντιστοιχηθεί σε έναν αριθμό των 16 bit. Υπάρχουν κάποιες εντολές που καταλαμβάνουν 32 bit όπως η εντολή goto

Τον δυαδικό κώδικα της κάθε εντολής μπορούμε αν θέλουμε να τον δούμε στο manual του μικροελεγκτή PIC 18F4550. Εκεί επεξηγείται το τι κάνει η κάθε εντολή, ποιες είναι οι παράμετρες της, πόσο χρόνο κάνει για να εκτελεστεί (σε κύκλους μηχανής) και υπάρχουν παραδείγματα χρήσης της.

Δημιουργία Project στο MPLAB

Που θα τοποθετήσουμε το project ;

Δημιουργούμε τον φάκελο:

C:\mikro\askisi-1

Σ’ αυτό τον φάκελο θα τοποθετήσουμε το Project

Τελικό σκοπός

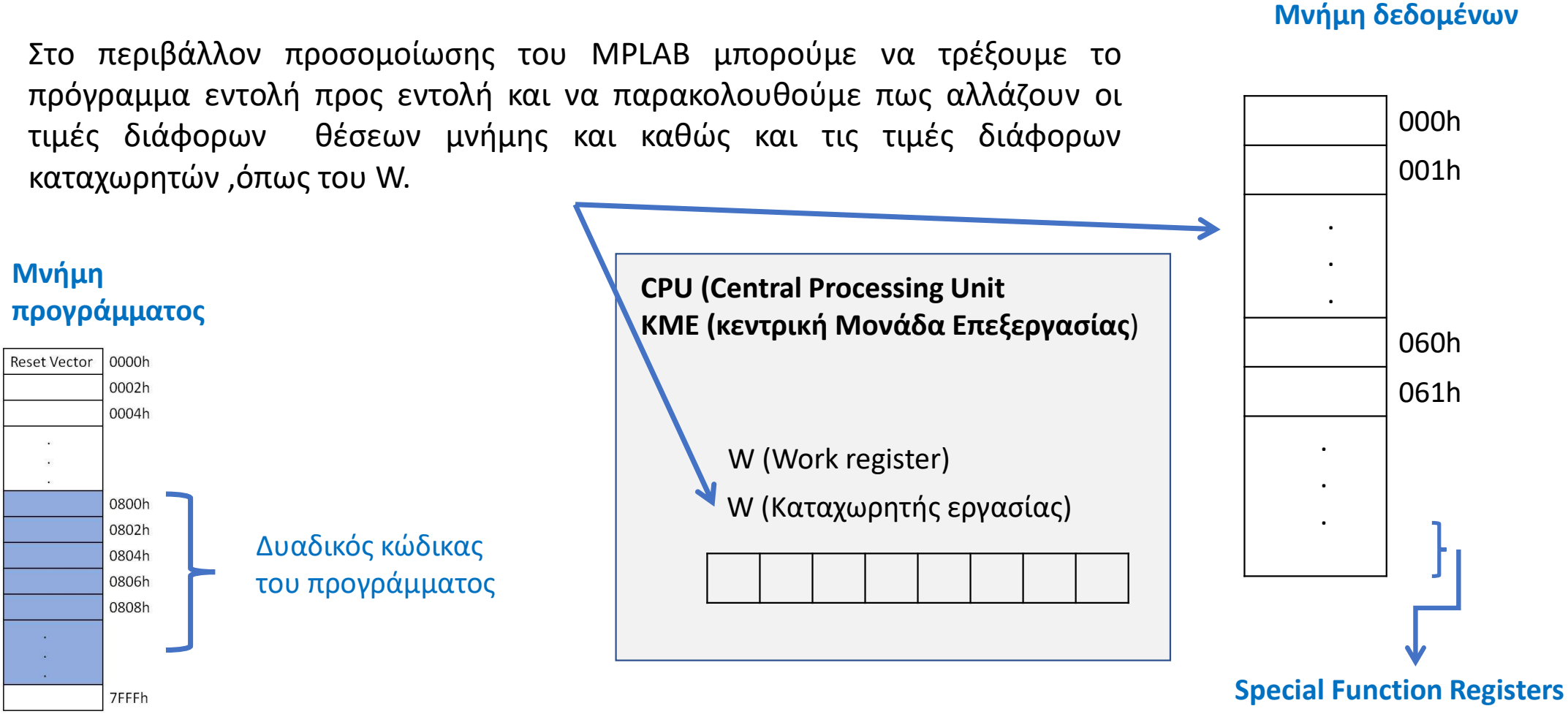
Ο τελικός σκοπός του Project είναι να μεταφραστεί το πρόγραμμα από γλώσσα Assembly σε γλώσσα μηχανής, να φορτωθεί στη μνήμη προγράμματος του μικροελεγκτή, και να εκτελεστεί.

Πρόγραμμα

```
org 0x800        ; Τοποθετούμαι τη διεύθυνση από την οποία θέλουμε
                 ; να ξεκινάει το πρόγραμμα μας.
movlw 0x2A       ; move literal to W:  0x2A  → W
                 ;μεταφέρεται η τιμή 0x2A στον W
movwf 0x060      ; move W to file 0x060:  W → 0x060
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x060
movlw 0XFF       ; move literal to W:  0xFF → W
                 ; Μεταφέρεται η τιμή 0xFF στον καταχωρητή W
movwf 0x061      ; move W to file :  W → 0x061
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x061
movf 0x060, 0    ; move file 0x060 to W:  0x060 → W
                 ; Μεταφέρεται το περιεχόμενο της θέσης μνήμης 0x040 στον W
movwf 0x061      ; move w to file 0x061:  W →0x061
                 ; Μετάφερε το περιεχόμενο του W  στη θέση μνήμης 0X061
loop goto loop   ; ατέρμων βρόχος χωρίς εντολή
END              ; Οδηγία προς τον Assembler  ότι εδώ τελειώνει το πρόγραμμα
                 ; Που θα μεταφραστεί σε γλώσσα μηχανής
```

Εκτός από τη μετάφραση σε γλώσσα μηχανής τι άλλο μπορούμε να δούμε με το MPLAB;

Στο περιβάλλον προσομοίωσης του MPLAB μπορούμε να τρέξουμε το πρόγραμμα εντολή προς εντολή και να παρακολουθούμε πως αλλάζουν οι τιμές διάφορων θέσεων μνήμης και καθώς και τις τιμές διάφορων καταχωρητών ,όπως του W.



Πως δημιουργούμε Project στο MPLAB;
Πως τρέχουμε το πρόγραμμα σε προσομοίωση(Simulation);

Στο Moodle, στο μάθημα μικροελεγκτές, υπάρχουν οδηγίες για δημιουργία Project στο MPLAB καθώς και οδηγίες για το τρέξιμο του προγράμματος σε περιβάλλον προσομοίωσης.

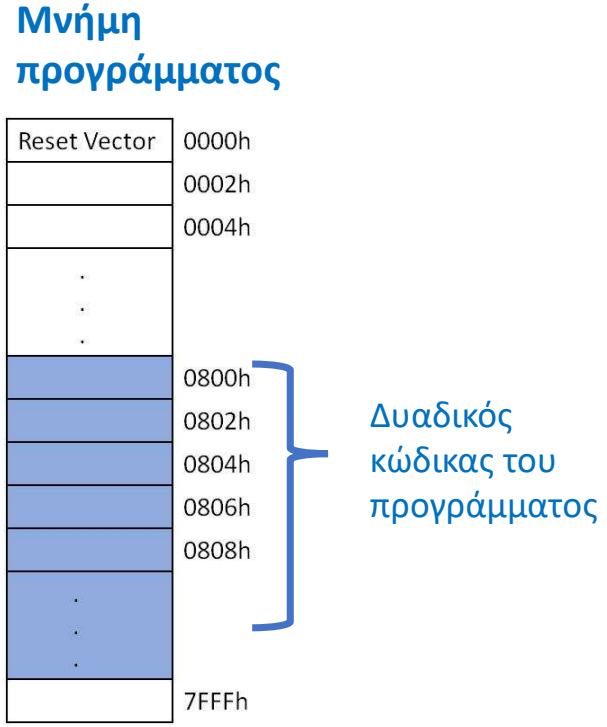
Προσοχή, θα δημιουργήσετε τον φάκελο:

C:\mikro\askisi-1

Σ' αυτό τον φάκελο θα τοποθετήσετε όλα τα αρχεία του Project της άσκησης 1.

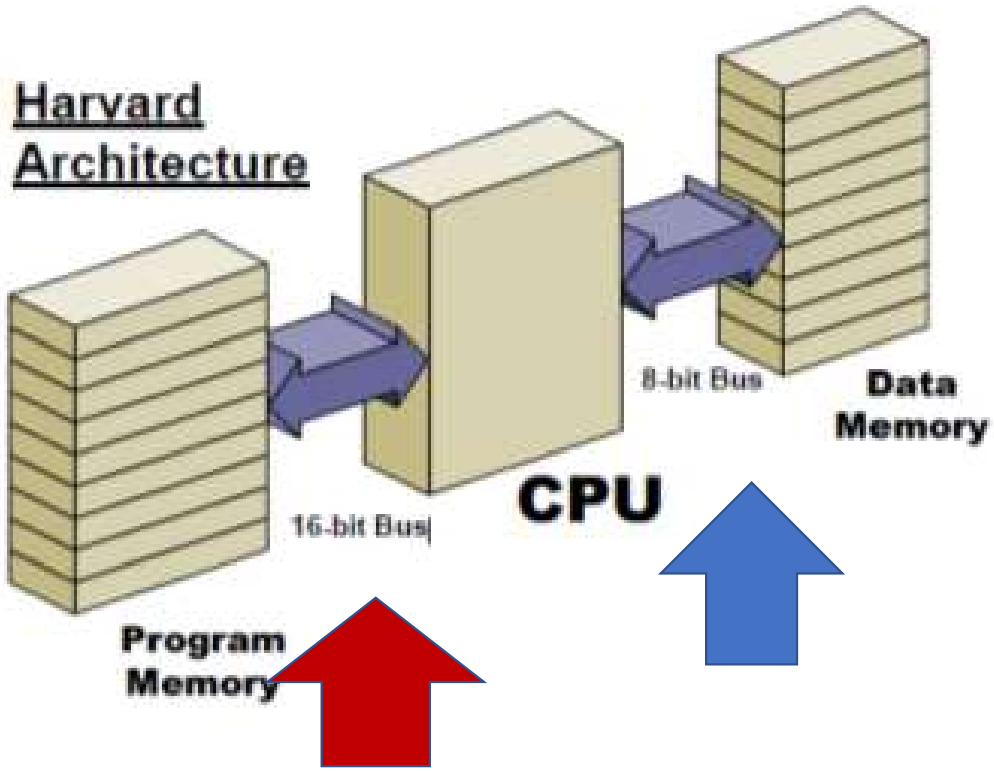
Όταν θα πάτε στην άσκηση 2 θα δημιουργήσετε τον φάκελο

C:\mikro\askisi-2 για να τοποθετήσετε όλα τα αρχεία του Project της άσκησης 2.



Προσοχή! Κατά τη διάρκεια εκτέλεσης του προγράμματος το περιεχόμενο της μνήμης προγράμματος μένει αμετάβλητο!!!

Αρχιτεκτονική του μικροελεγκτή PIC18F4550



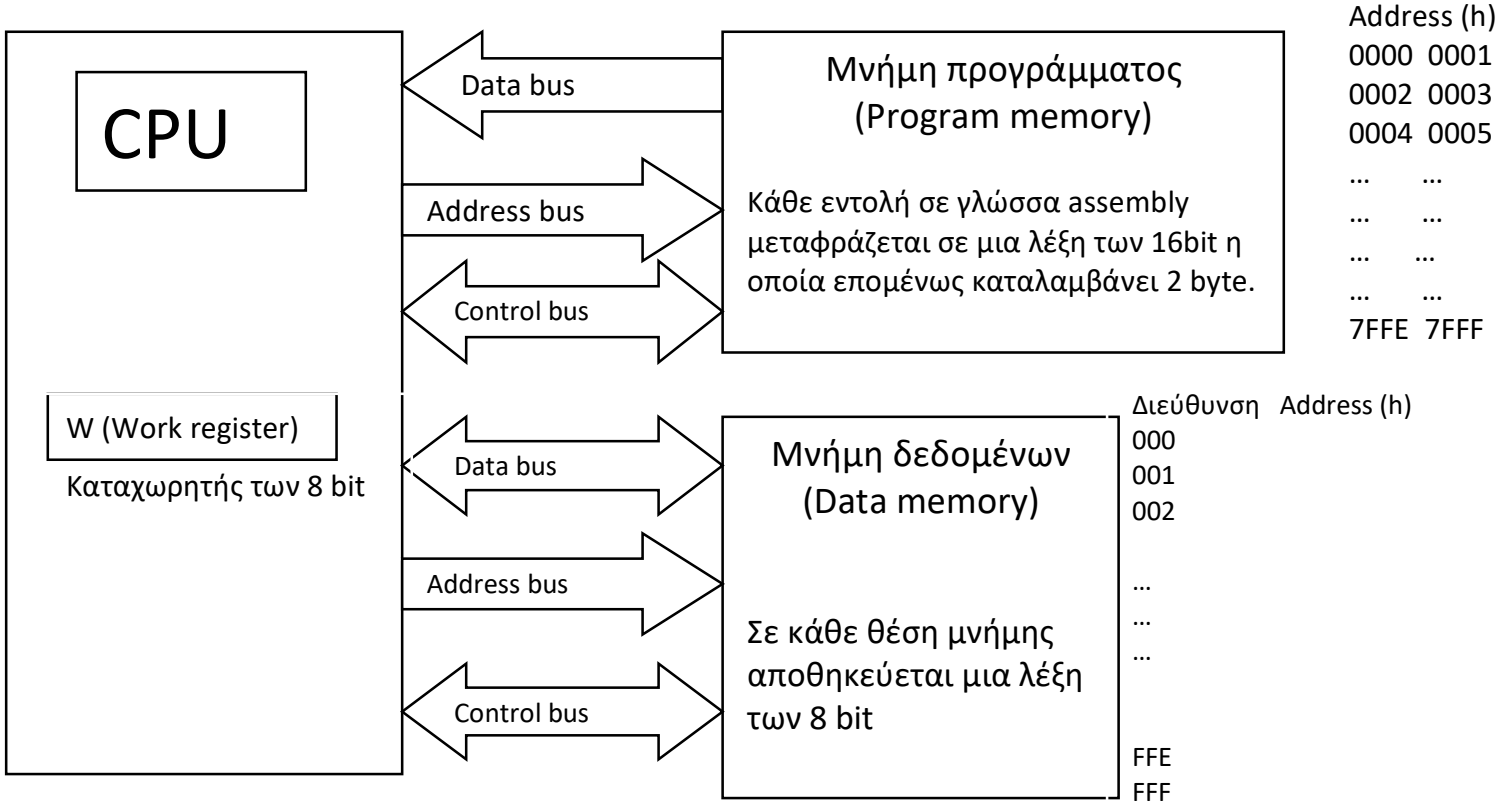
Ο δίαυλος δεδομένων προς τη μνήμη δεδομένων είναι των 8 bit

Ο δίαυλος δεδομένων προς τη μνήμη προγράμμαμος είναι 16 bit.

Επομένως μεταφέρονται συγχρόνως από τη μνήμη προγράμματος 16 bit.

Οι εντολές σε γλώσσα μηχανής είναι κατά κανόνα λέξεις των 16 bit

Σύνδεση της Κεντρικής Μονάδας Επεξεργασίας (CPU) με τη μνήμη προγράμματος και τη μνήμη δεδομένων



Γιατί στην αρίθμηση των θέσεων μνήμης χρησιμοποιούμε το δεκαεξαδικό αριθμητικό σύστημα;

Μπορούμε να τοποθετήσουμε περιεχόμενο σε μια θέση μνήμης ή να διαβάσουμε το περιεχόμενο από μια θέση μνήμης (Write / Read)

Data Bus
8 bit
Δίαυλος
δεδομένων

8 αγωγοί για μεταφορά του περιεχομένου της μνήμης

Περιεχόμενα θέσεων μνήμης (b)	Διευθύνσεις στο δεκαεξαδικό αριθμητικό σύστημα (h)	Διευθύνσεις στο δυαδικό αριθμητικό σύστημα (b)	Διευθύνσεις στο δεκαδική αριθμητικό σύστημα (d)
0101 0101	000	0000 0000 0000	0
1111 1111	001	0000 0000 0001	1
1000 1110	002	0000 0000 0010	2
. . .	. . .	. . .	. . .
1010 1111	0FE	0000 1111 1110	254
1001 1010	0FF	0000 1111 1111	255
1111 0000	100	0001 0000 0000	256
1111 1000	101	0001 0000 0001	257

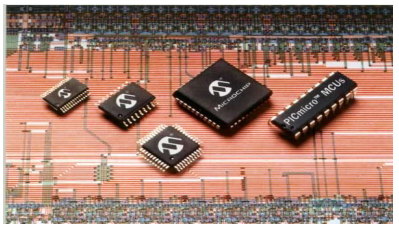
Address Bus
12 bit
Δίαυλος
διευθύνσεων

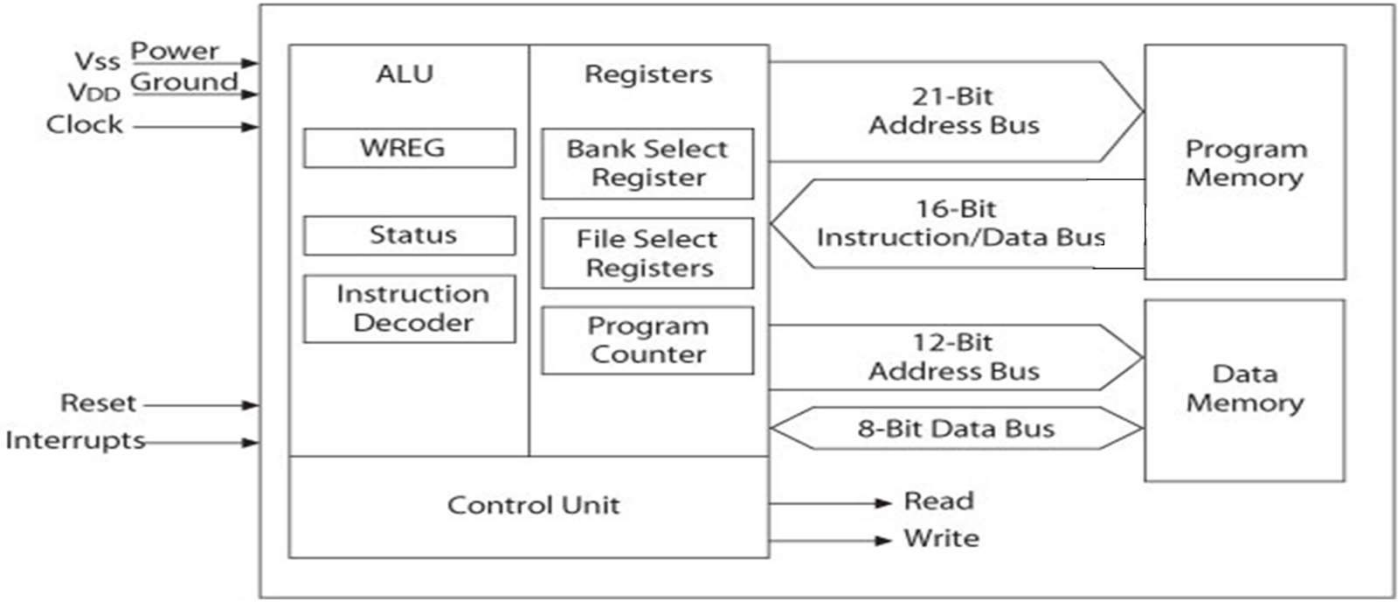
1----------
1----------
0----------
0----------
1----------
1----------
0----------
0----------
0----------
0----------
1----------
0----------

Διεύθυνση $CC2_h$ (Αποδίδεται από την Κεντρική Μονάδα επεξεργασίας, CPU)

12 αγωγοί για επιλογή της διεύθυνσης από την οποία θα γίνει η ανάγνωση ή η εγγραφή δεδομένων

PIC18F – MCU and Memory





Program Memory: 32 K (2^{15})

Περιοχή διευθύνσεων: 000000 to $007FFF_h$

Καταχωρητές των 16-bit

Data Memory: 2 K

Περιοχή διευθύνσεων: 000 to $7FF_h$

Καταχωρητές των 8 bit

Η μνήμη προγράμματος είναι οργανωμένη σε λέξεις των 16 bit

Ο δίαυλος διευθύνσεων των 21 bit θα μπορούσε να έχει πρόσβαση σε

2 MB
2^{21}

Η μνήμη δεδομένων είναι οργανωμένη σε λέξεις των 8 bit

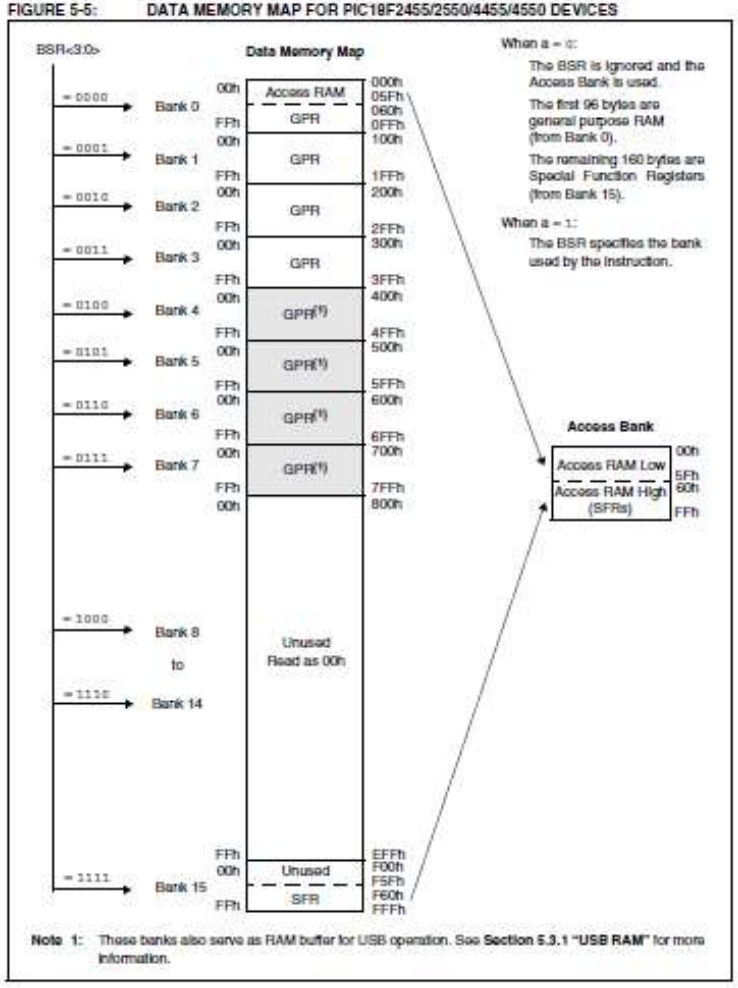
Ο δίαυλος διευθύνσεων των 12 bit θα μπορούσε να έχε πρόσβαση σε

4KB
2^{12}

Η μνήμη δεδομένων του μικροελεγκτή PIC18F4550
Data memory of the PIC18F4550 microcontroller

PIC18F2455/2550/4455/4550

FIGURE 5-5: DATA MEMORY MAP FOR PIC18F2455/2550/4455/4550 DEVICES



Note 1: These banks also serve as RAM buffer for USB operation. See Section 5.3.1 "USB RAM" for more information.

Η μνήμη δεδομένων του μικροελεγκτή είναι από τη Διεύθυνση (000)h έως (7FF)h.

Δηλαδή από 0000 0000 0000 έως 111 1111 1111

Το πλήθος των διευθύνσεων είναι $2^{11} = 2^1 \times 2^{10} = 2 \times 1024 = 2K$
Σε κάθε θέση μνήμης αποθηκεύεται ένα byte (8bit).

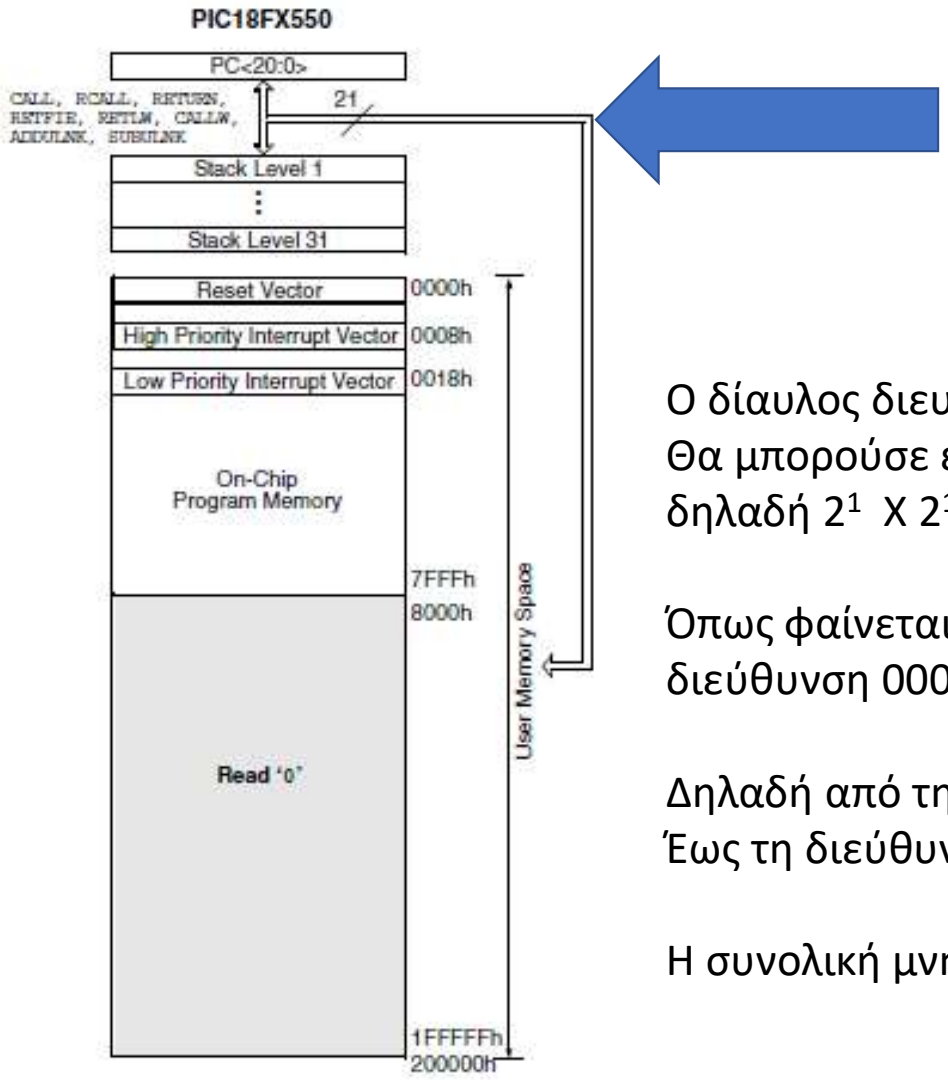
Η μνήμη δεδομένων του Μικροελεγκτή PIC 18F4550 είναι 2K

Προσοχή! $2^{10} = 1024 = 1K$ για τις μνήμες
$2^{20} = 1K \times 1K = 1\ M$ (1 048 576 θέσεις μνήμης)
$2^{30} = 1K \times 1K \times 1K = 1G$ (1 073 741 824 θέσεις μνήμης

Προσοχή, χρησιμοποιούνται και οι θέσεις μνήμης $(F60)_h$ έως $(FFF)_h$.

Αυτές οι θέσεις μνήμης χρησιμοποιούνται για τους Special Function Registers (SFR). Είναι θέσης μνήμης που έχουν μια ειδική λειτουργία.

Μνήμη προγράμματος του μικροελεγκτή PIC18F4550



Προσοχή: Οι τιμές στο δίαυλο διευθύνσεων της μνήμης προγράμματος τοποθετούνται από τον Program Counter . Ο Program Counter είναι ένας καταχωρητής τοποθετημένος μέσα στην Κεντρική Μονάδα Επεξεργασίας (CPU).

Ο δίαυλος διευθύνσεων περιλαμβάνει 21 αγωγούς.
Θα μπορούσε επομένως να έχει πρόσβαση σε 2^{21} θέσει μνήμης,
δηλαδή 2^1 X 2^{10} X 2^{10}=2 X 1K X 1K = 2M θέσεις μνήμης.

Όπως φαίνεται στο σχήμα το υλοποιημένο κομμάτι της μνήμης είναι από τη διεύθυνση 0000_h έως τη διεύθυνση $7FFF_h$

Δηλαδή από τη διεύθυνση 0000 0000 0000 0000
Έως τη διεύθυνση 111 1111 1111 1111

Η συνολική μνήμη προγράμματος είναι 2^{15} =2^5 X 2^{10} = 32 K

Υπολογισμός πλήθους θέσεων μνήμης, καθορισμός Bank

Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **000h** έως τη διεύθυνση **0FFh**;

Απάντηση: FF +1 (διότι περιλαμβάνεται και η μηδενική διεύθυνση)

FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 0**

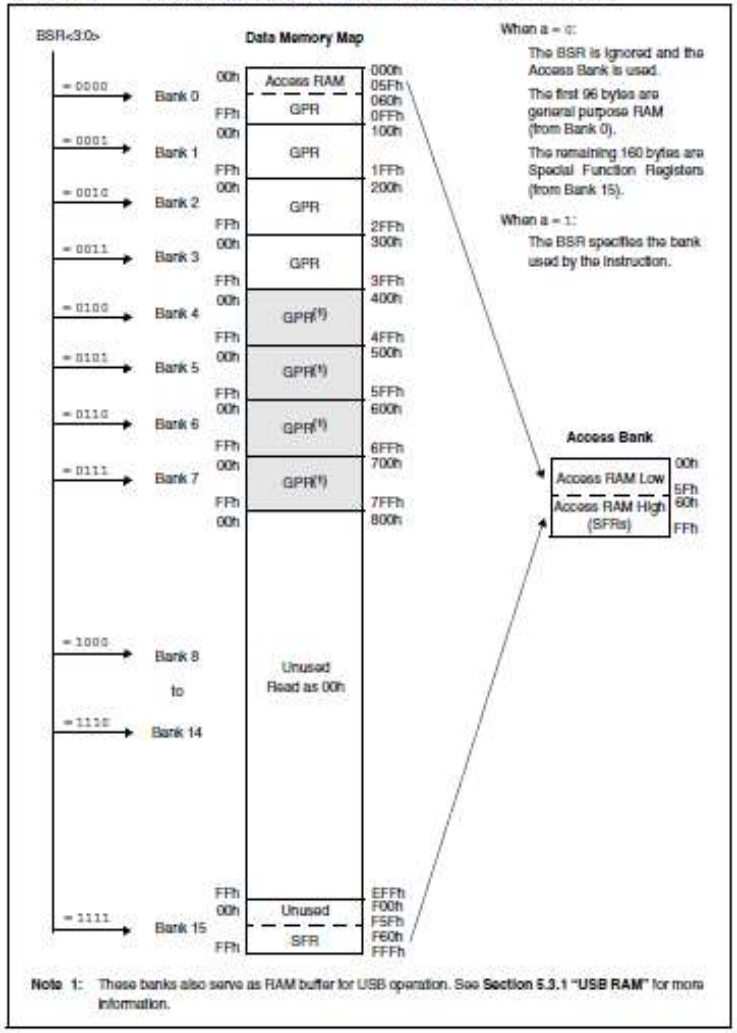
Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **100h** έως τη διεύθυνση **1FFh**;
Απάντηση: 1FF - 100= FF
FF +1 (διότι περιλαμβάνεται και η διεύθυνση 100)
Οι θέσεις μνήμης είναι:
FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 1**

Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **200h** έως τη διεύθυνση **2FFh**;
Απάντηση: 2FF - 200= FF
FF +1 (διότι περιλαμβάνεται και η διεύθυνση 100)
Οι θέσεις μνήμης είναι:
FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 2**

Πόσες είναι οι θέσεις μνήμης από τη διεύθυνση **300h** έως τη διεύθυνση **3FFh**;
Απάντηση: 3FF - 300= FF
FF +1 (διότι περιλαμβάνεται και η διεύθυνση 100)
Οι θέσεις μνήμης είναι:
FF + 1 = (100)h = (1 0000 0000)b = 2^8 =256 **Bank 3**

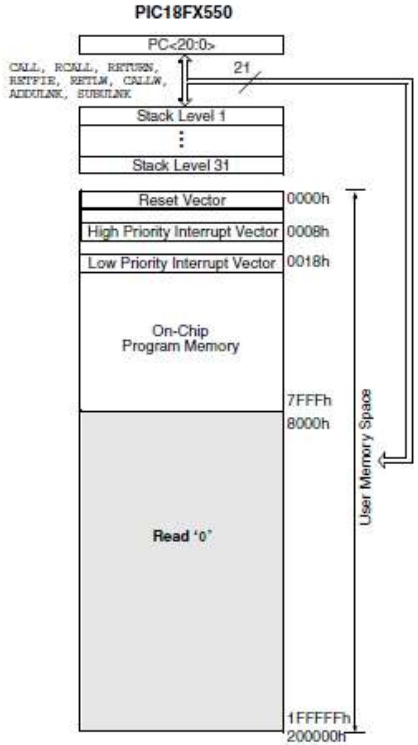
PIC18F2455/2550/4455/4550

FIGURE 5-5: DATA MEMORY MAP FOR PIC18F2455/2550/4455/4550 DEVICES



Δημιουργία προγράμματος σε γλώσσα Assembly (1)

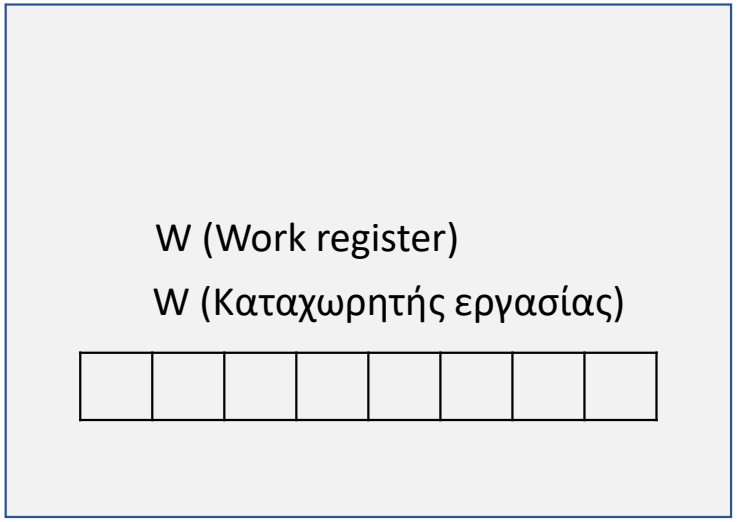
Να γραφεί πρόγραμμα με το οποίο στη θέση μνήμης 060h τοποθετείται η τιμή 2Ah και στη θέση μνήμης 061h η τιμή FF και στη συνέχεια το περιεχόμενο της θέσης μνήμης 060h μεταφέρεται στη θέση μνήμης 061h. Μετά την εκτέλεση των παραπάνω το πρόγραμμα να εκτελεί έναν ατέρμονα(χωρίς τέλος) βρόχο χωρίς να εκτελεί κάποια εντολή. Το πρόγραμμα να τοποθετηθεί στη μνήμη προγράμματος από τη διεύθυνση 0800h και μετά

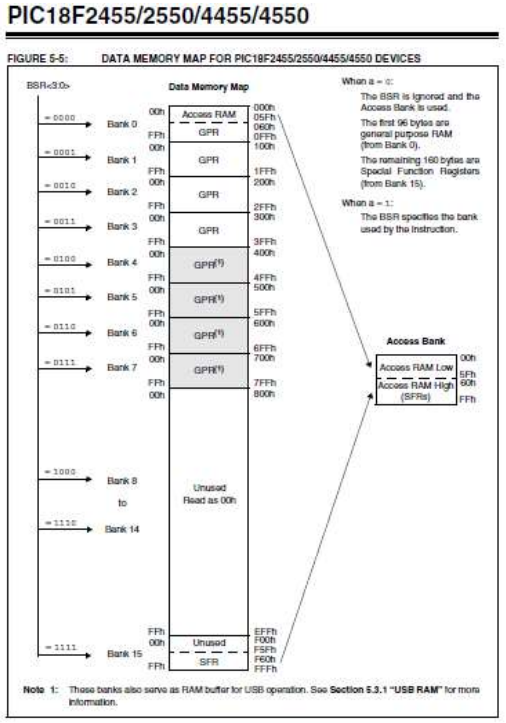


Μνήμη προγράμματος

Το πρόγραμμα θα γραφεί από τη διεύθυνση 0800 h και μετά. (Πώς θα το τοποθετήσουμε σ' αυτή τη θέση;

CPU (Central Processing Unit
ΚΜΕ (κεντρική Μονάδα Επεξεργασίας)





Μνήμη δεδομένων

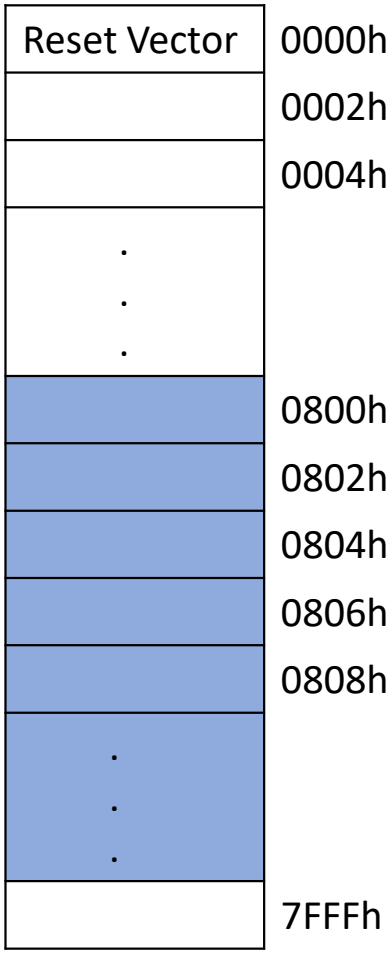
$2A_h$ →Να γίνει περιεχόμενο της θέσης μνήμης 060_h

FF_h → Να γίνει περιεχόμενο της θέσης μνήμης 061_h

Το περιεχόμενο της θέσης μνήμης 060_h να μεταφερθεί στο περιεχόμενο της θέσης μνήμης 061_h

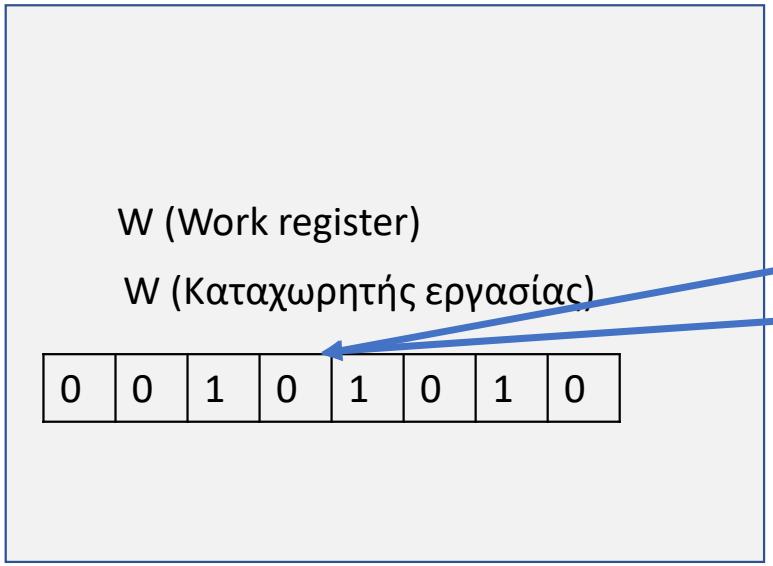
Δημιουργία προγράμματος σε γλωσσα Assembly (2)

Μνήμη προγράμματος
16 bit Instructions
Εντολές των 16 bit

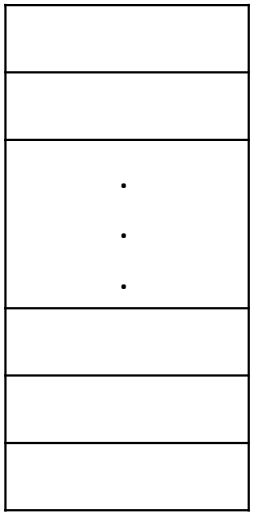


Το πρόγραμμα θα γραφεί από τη διεύθυνση 0800h και μετά

CPU (Central Processing Unit
ΚΜΕ (κεντρική Μονάδα Επεξεργασίας)



Μνήμη δεδομένων
Οργανωμένη σε λέξεις των 8 bit

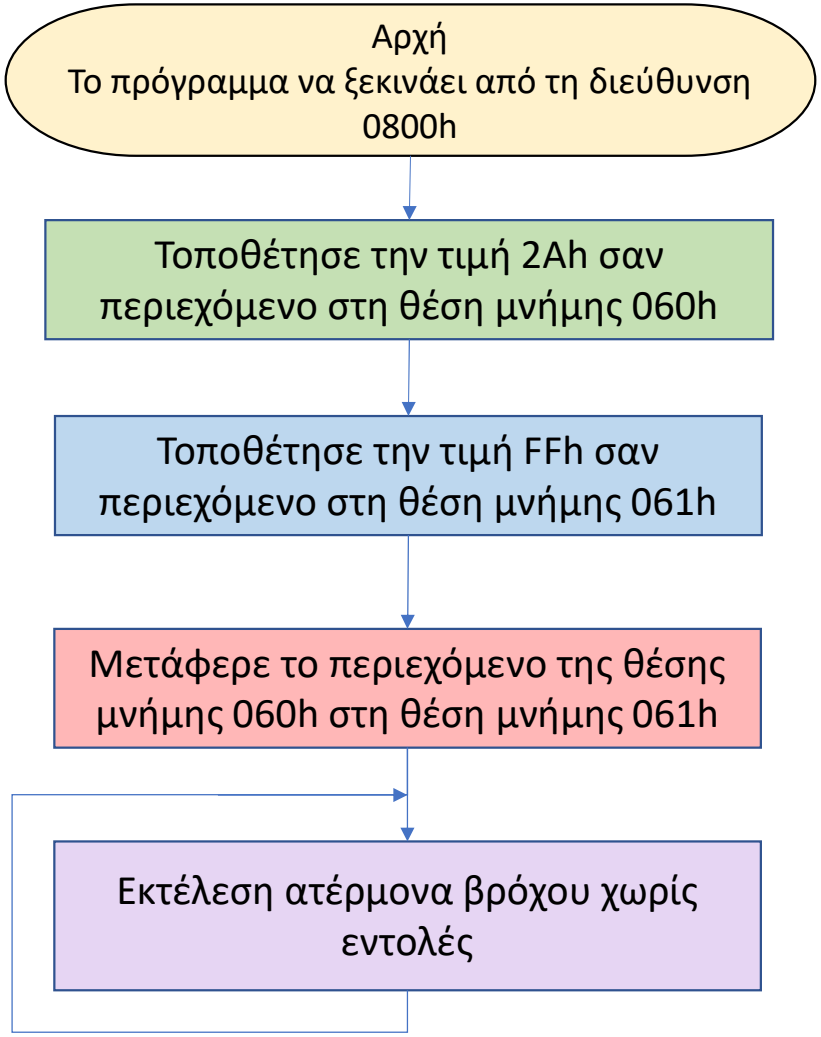


Μνήμη δεδομένων

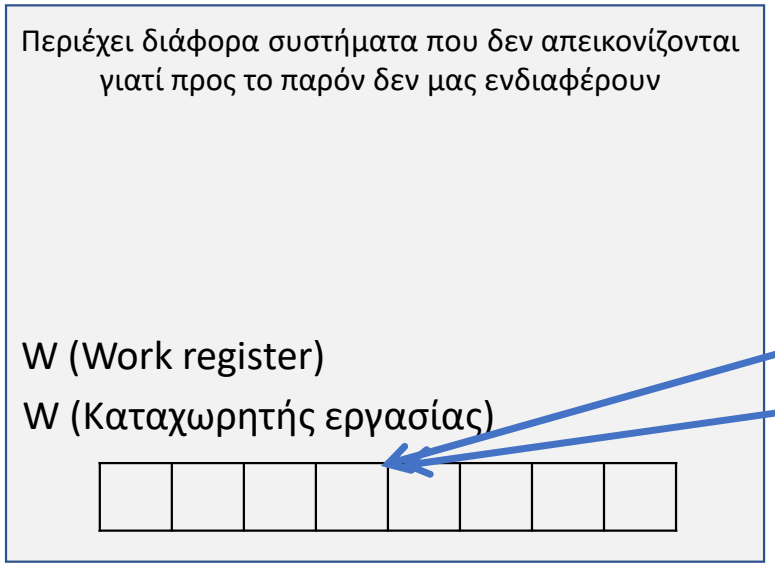
Η τοποθέτηση τιμών στις θέσεις τις μνήμης δεδομένων και η μεταφορά τιμών από και προς τις θέσεις της μνήμης δεδομένων γίνεται δια μέσου του καταχωρητή εργασίας W

Διάγραμμα ροής το προγράμματος (3)

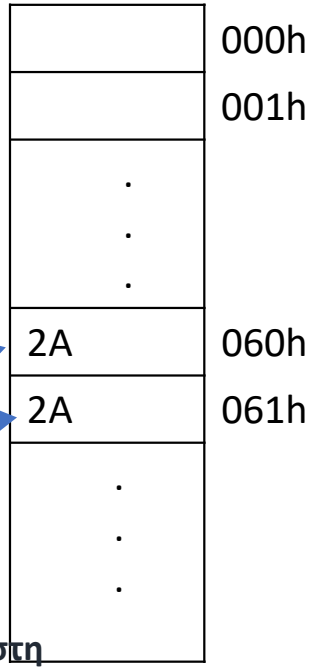
Διάγραμμα ροής του προγράμματος



CPU (Central Processing Unit
ΚΜΕ (κεντρική Μονάδα Επεξεργασίας)



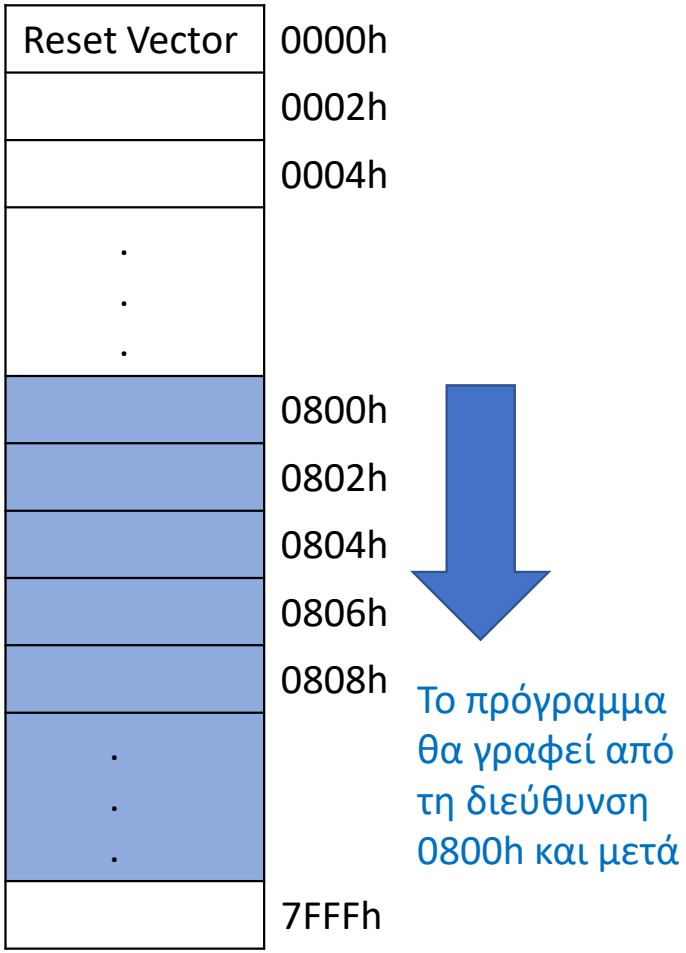
Μνήμη δεδομένων
Οργανωμένη σε λέξεις των 8 bit



Πρόγραμμα

```
org 0x800          ; Αρχική διεύθυνση του προγράμματος στη
                   ; μνήμη προγράμματος
        movlw 0x2A     ; move literal to W:  0x2A  → W
        movwf 0x060    ; move W to file 0x060:  W → 0x060
        movlw 0XFF     ; move literal to W:  0xFF → W
        movwf 0x061    ; move W to file :  W → 0x061
        movf 0x060, 0  ; move file 0x060 to W:  0x060 → W
        movwf 0x061    ; move w to file 0x061:  W →0x061
loop    goto loop      ; ατέρμων βρόχος χωρίς εντολή
END
```

Μνήμη προγράμματος
16 bit Instructions
Εντολές των 16 bit



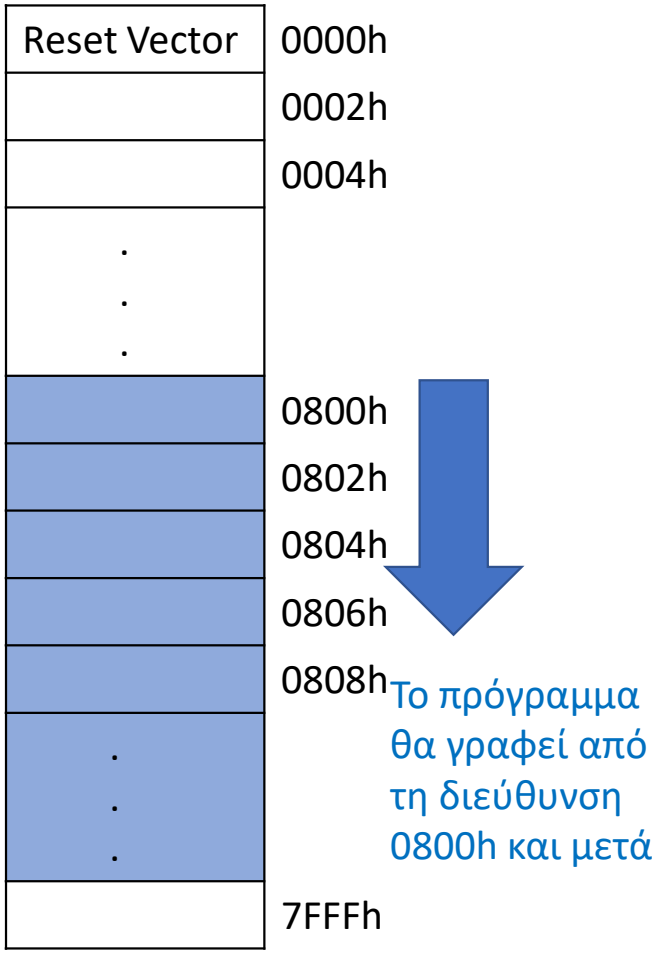
Ποιος θα είναι ο κώδικας του προγράμματος; (1)
Πως μπορούμε να δούμε τον κώδικα του προγράμματος στο MPLAB;

Πρόγραμμα

```
org 0x800        ; Τοποθετούμαι τη διεύθυνση στη μνήμη προγράμματος
                 ; από την οποία θέλουμε
                 ; να ξεκινάει το πρόγραμμα μας. Είναι οδηγία προς
                 ; τον Assembler
movlw 0x2A       ; move literal to W:  0x2A  →  W
                 ;μεταφέρεται η τιμή 0x2A στον W
movwf 0x060      ; move W to file 0x060:  W → 0x060
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x060
movlw 0XFF       ; move literal to W:  0xFF  →  W
                 ; Μεταφέρεται η τιμή 0xFF στον καταχωρητή W
movwf 0x061      ; move W to file :  W  →  0x061
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x061
movf 0x060, 0    ; move file 0x060 to W:  0x060  →  W
                 ; Μεταφέρεται το περιεχόμενο της θέσης μνήμης 0x060 στον W
movwf 0x061      ; move w to file 0x061:  W →0x061
                 ; Μετάφερε το περιεχόμενο του W  στη θέση μνήμης 0X061
loop goto loop   ; ατέρμων βρόχος χωρίς εντολή
END              ; Οδηγία προς τον Assembler  ότι εδώ τελειώνει το πρόγραμμα
                 ; Που θα μεταφραστεί σε γλώσσα μηχανής
```

Ποιος θα είναι ο κώδικας του προγράμματος; (2)
Πως μπορούμε να δούμε τον κώδικα του προγράμματος στο MPLAB;

Μνήμη προγράμματος
16 bit Instructions
Εντολές των 16 bit



Κάθε εντολή του προγράμματος θα αντιστοιχηθεί σε έναν αριθμό των 16 bit. Υπάρχουν κάποιες εντολές που καταλαμβάνουν 32 bit όπως η εντολή goto

Τον δυαδικό κώδικα της κάθε εντολής μπορούμε αν θέλουμε να τον δούμε στο manual του μικροελεγκτή PIC 18F4550. Εκεί επεξηγείται το τι κάνει η κάθε εντολή, ποιες είναι οι παράμετρες της, πόσο χρόνο κάνει για να εκτελεστεί (σε κύκλους μηχανής) και υπάρχουν παραδείγματα χρήσης της.

Δημιουργία Project στο MPLAB

Που θα τοποθετήσουμε το project ;

Δημιουργούμε τον φάκελο:

C:\mikro\askisi-1

Σ’ αυτό τον φάκελο θα τοποθετήσουμε το Project

Τελικό σκοπός

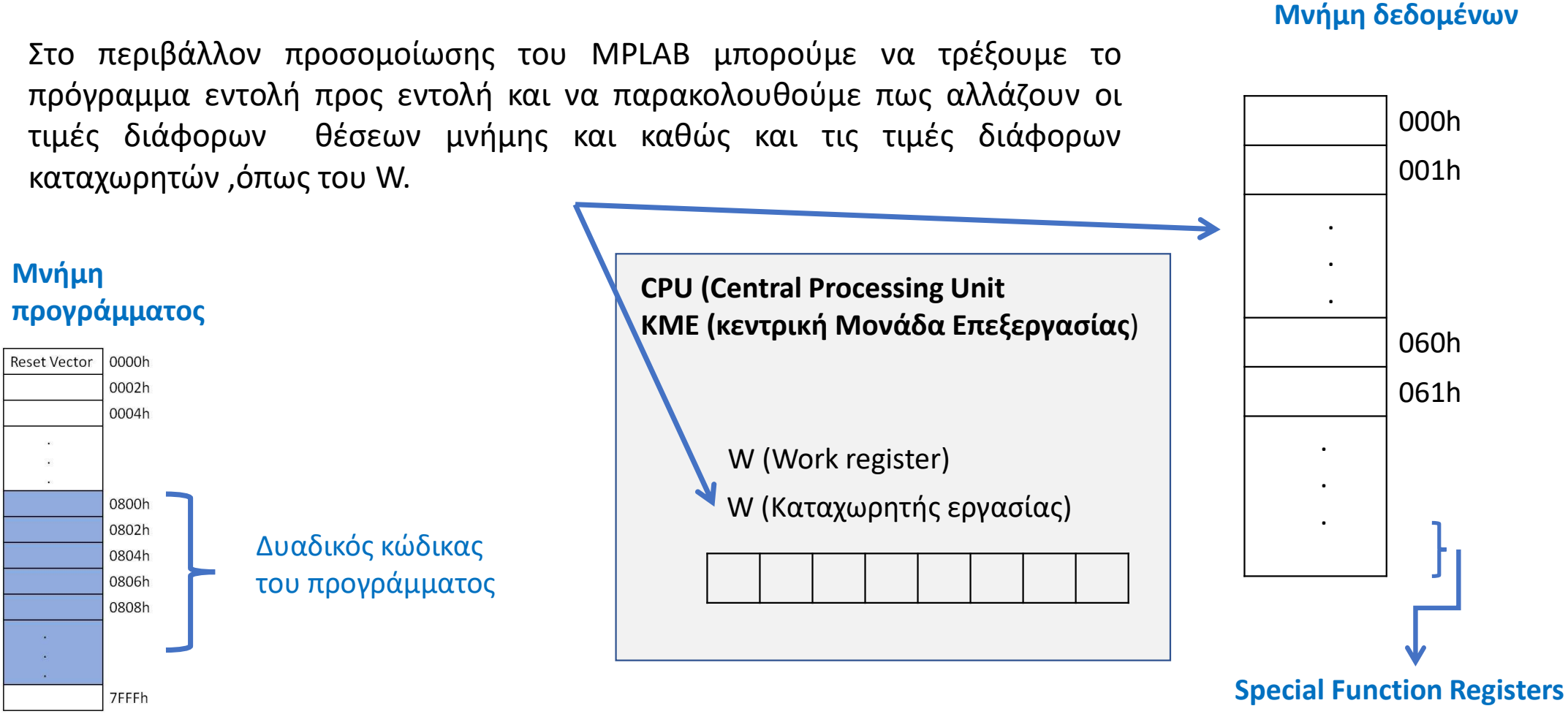
Ο τελικός σκοπός του Project είναι να μεταφραστεί το πρόγραμμα από γλώσσα Assembly σε γλώσσα μηχανής, να φορτωθεί στη μνήμη προγράμματος του μικροελεγκτή, και να εκτελεστεί.

Πρόγραμμα

```
org 0x800        ; Τοποθετούμαι τη διεύθυνση από την οποία θέλουμε
                 ; να ξεκινάει το πρόγραμμα μας.
movlw 0x2A       ; move literal to W:  0x2A  → W
                 ;μεταφέρεται η τιμή 0x2A στον W
movwf 0x060      ; move W to file 0x060:  W → 0x060
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x060
movlw 0XFF       ; move literal to W:  0xFF → W
                 ; Μεταφέρεται η τιμή 0xFF στον καταχωρητή W
movwf 0x061      ; move W to file :  W → 0x061
                 ; μεταφέρεται το περιεχόμενο του W στη θέση μνήμης 0x061
movf 0x060, 0    ; move file 0x060 to W:  0x060 → W
                 ; Μεταφέρεται το περιεχόμενο της θέσης μνήμης 0x060 στον W
movwf 0x061      ; move w to file 0x061:  W →0x061
                 ; Μετάφερε το περιεχόμενο του W  στη θέση μνήμης 0X061
loop goto loop   ; ατέρμων βρόχος χωρίς εντολή
END              ; Οδηγία προς τον Assembler  ότι εδώ τελειώνει το πρόγραμμα
                 ; Που θα μεταφραστεί σε γλώσσα μηχανής
```

Εκτός από τη μετάφραση σε γλώσσα μηχανής τι άλλο μπορούμε να δούμε με το MPLAB;

Στο περιβάλλον προσομοίωσης του MPLAB μπορούμε να τρέξουμε το πρόγραμμα εντολή προς εντολή και να παρακολουθούμε πως αλλάζουν οι τιμές διάφορων θέσεων μνήμης και καθώς και τις τιμές διάφορων καταχωρητών ,όπως του W.



Πως δημιουργούμε Project στο MPLAB;
Πως τρέχουμε το πρόγραμμα σε προσομοίωση(Simulation);

Στο Moodle, στο μάθημα μικροελεγκτές, υπάρχουν οδηγίες για δημιουργία Project στο MPLAB καθώς και οδηγίες για το τρέξιμο του προγράμματος σε περιβάλλον προσομοίωσης.

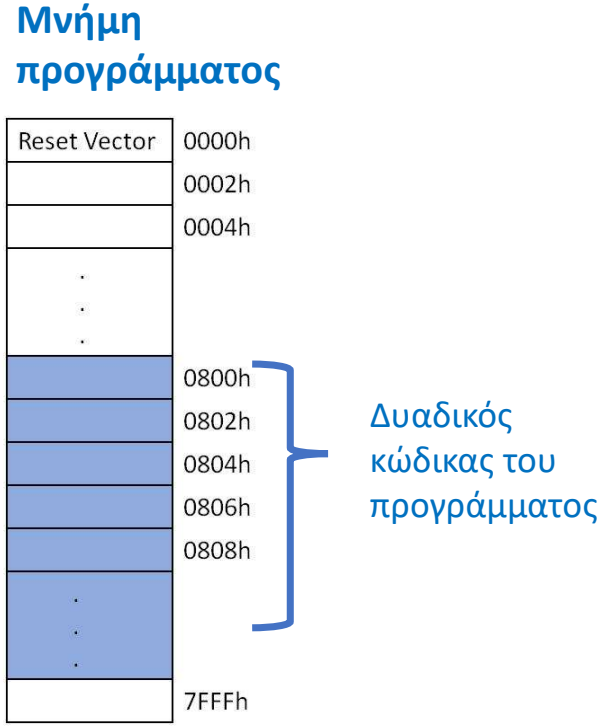
Προσοχή, θα δημιουργήσετε τον φάκελο:

C:\mikro\askisi-1

Σ' αυτό τον φάκελο θα τοποθετήσετε όλα τα αρχεία του Project της άσκησης 1.

Όταν θα πάτε στην άσκηση 2 θα δημιουργήσετε τον φάκελο

C:\mikro\askisi-2 για να τοποθετήσετε όλα τα αρχεία του Project της άσκησης 2.



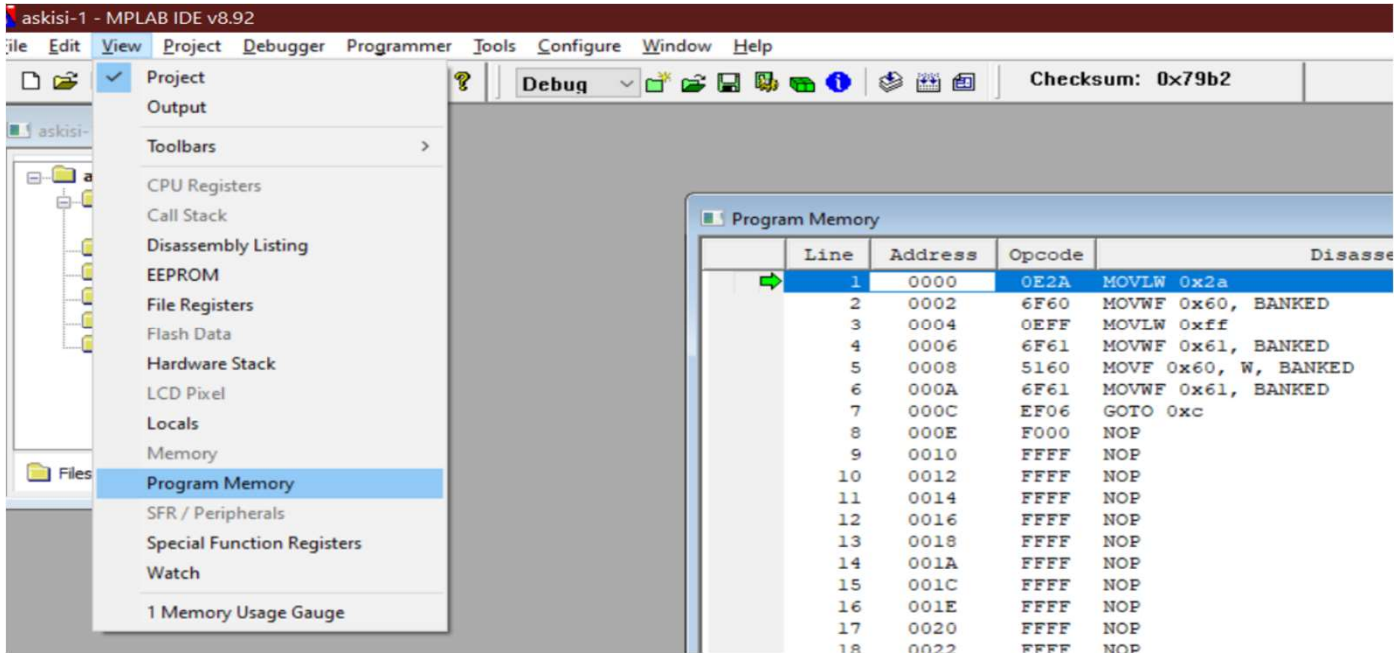
Προσοχή! Κατά τη διάρκεια εκτέλεσης του προγράμματος το περιεχόμενο της μνήμης προγράμματος μένει αμετάβλητο!!!

MPLAB, Άσκηση 1 (1)
Το πρόγραμμα στη μνήμη προγράμματος του Μικροελεγκτή

C:\mikro\askisi-1\askisi-1.asm

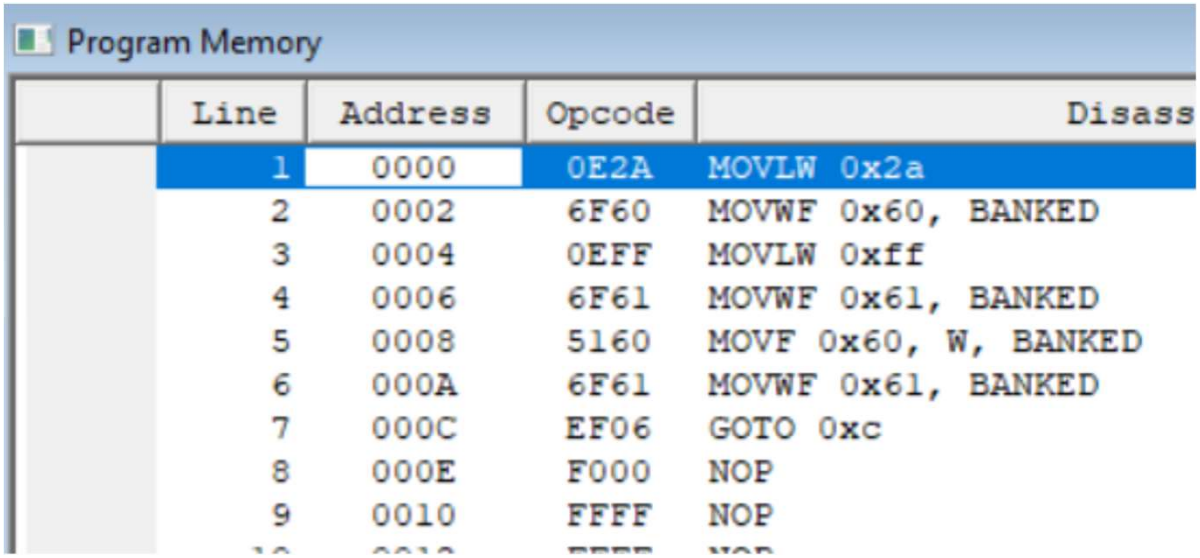
```
org 0x000               ;  Δηλώνουμε τη διεύθυνση όπου  θα τοποθετηθεί
                        ;  το πρόγραμμα μας στη μνήμη προγράμματος
        movlw 0x2A      ;  Μετάφερε την τιμή 0x2a στον καταχωρητή εργασίας w
        movwf 0x060     ;  Μετάφερε το περιεχόμενο του w στη θέση μνήμης 0χ060
        movlw 0XFF      ;  Μετάφερε την τιμή 0xFF στον καταχωρητή εργασίας w
        movwf 0x061     ;  Μετάφερε το περιεχόμενο του w στη θέση μνήμης 0χ061
        movf 0x060, 0   ;  Μετάφερε το περιεχόμενο της θέσης μνήμης 0χ060 στον w
        movwf 0x061     ;  Μετάφερε το περιεχόμενο του w στη θέση μνήμης στη θέση μνήμης 0χ061
 loop   goto loop       ;  Διακλάδωση της εντολής στον ίδιο τον εαυτό της. Ατέρμων βρόχος.
 END                    ; Δήλωση ότι εδώ τελειώνουν οι εντολές του προγράμματος
```

Για να δούμε το πρόγραμμα στη μνήμη προγράμματος επιλέγουμε:
View → Programm memory



MPLAB, Άσκηση 1 (2)

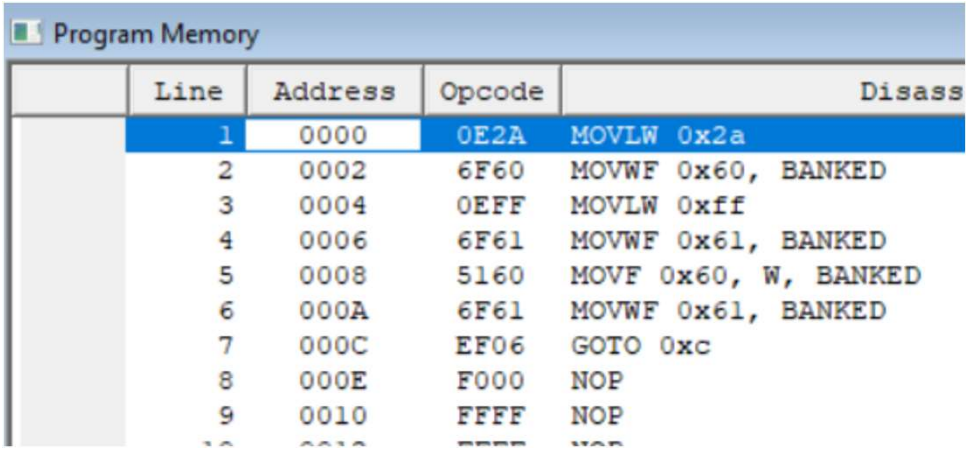
Ο κώδικας του προγράμματος του μικροελεγκτή σε γλώσσα μηχανής



1. Το πρόγραμμα έχει τοποθετηθεί στη διεύθυνση 0000 και μετά διότι δόθηκε η οδηγία:
   org 0x000
2. Ο κώδικας της εντολής movlw 0x2A είναι 0E2Ah δηλαδή 0000 1110 0010 1010b, δηλαδή είναι ένας αριθμός των 16 bit.
3. Η στήλη Opcode δείχνει τον δυαδικό κώδικα των εντολών σε γλώσσα Assembly. Τα περιεχόμενα είναι γραμμένα στο δεκαεξαδικό αριθμητικό σύστημα.

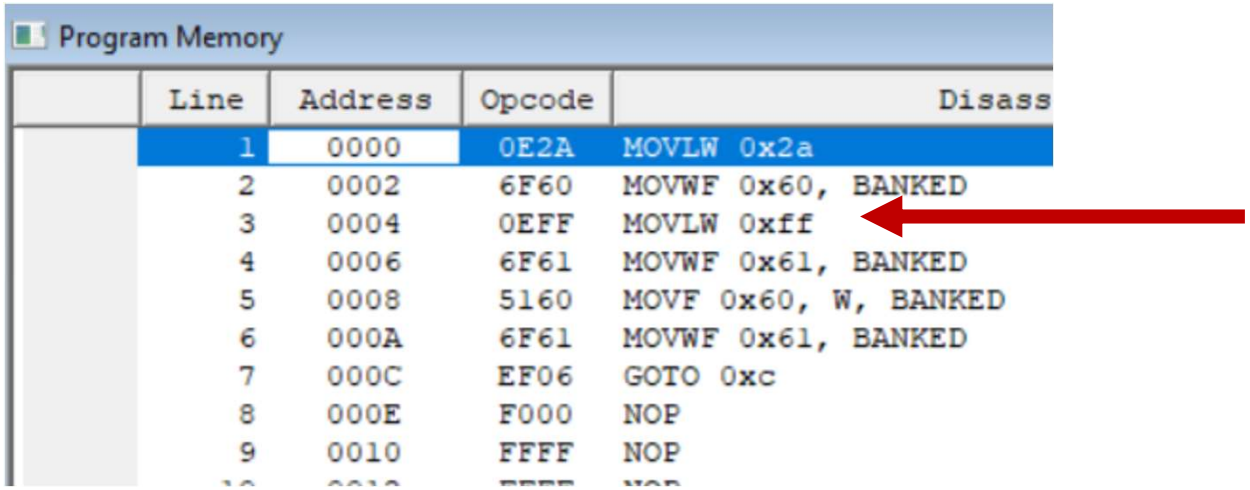
MPLAB, Άσκηση 1 (3)

Ο κώδικας του προγράμματος του μικροελεγκτή σε γλώσσα μηχανής



4. Η στήλη Address δείχνει τις διευθύνσεις στις οποίες είναι τοποθετημένοι οι δυαδικοί κώδικες των εντολών.
5. Ο κώδικας της εντολής movlw 0x2A καταλαμβάνει 2 byte στη μνήμη προγράμματος, ένα byte στη διεύθυνση 0000h και ένα byte στη διεύθυνση 0001h
6. Και τα 16 bit της εντολής μεταφέρονται συγχρόνως δια μέσω του διαύλου δεδομένων των 16 bit που συνδέει την Κεντρική Μονάδα Επεξεργασίας(CPU) με τη μνήμη προγράμματος

MPLAB, Άσκηση 1 (4)
Ο κώδικας του προγράμματος του μικροελεγκτή σε γλώσσα μηχανής



Ποιος είναι ο κώδικας της εντολής **movlw 0xFF** και σε ποιες θέσεις της μνήμης προγράμματος είναι τοποθετημένος;

Ο κώδικας είναι **0EFFh** και είναι τοποθετημένος στις θέσεις μνήμης **0004h** και **0005h**

Στη θέση μνήμης **0004h** είναι τοποθετημένη η τιμή **0Eh →0000 1110b**
Στη θέση μνήμης **0005h** είναι τοποθετημένη η τιμή **FFh →1111 1111b**

MPLAB, Άσκηση 1 (5)
Ο κώδικας του προγράμματος του μικροελεγκτή σε γλώσσα μηχανής

Program Memory

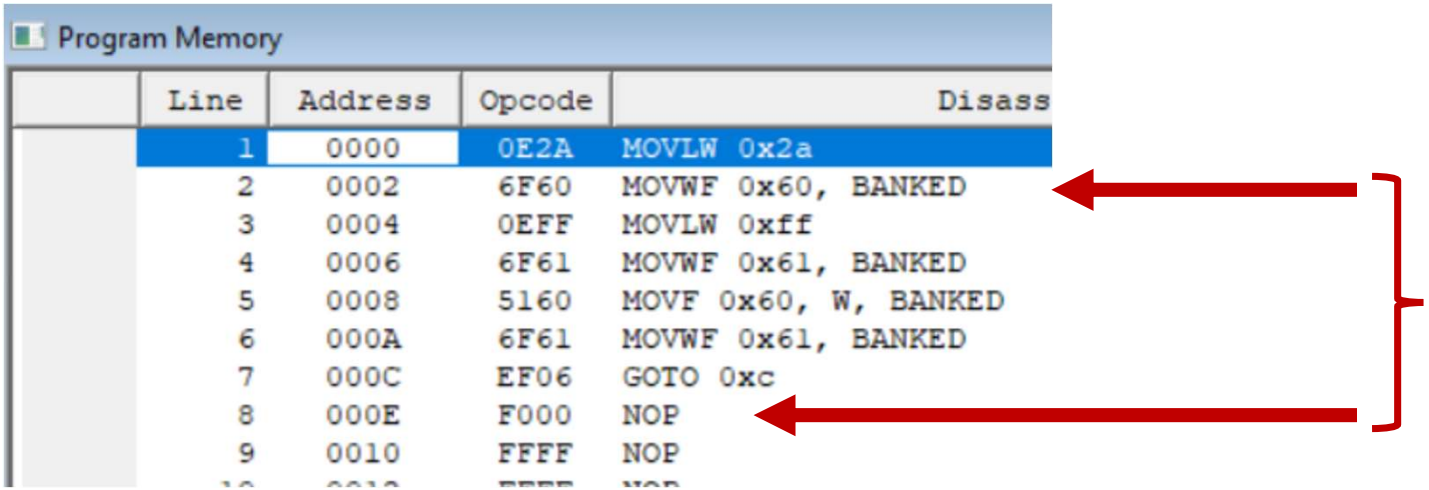
Line	Address	Opcode	Disass
1	0000	0E2A	MOVLW 0x2a
2	0002	6F60	MOVWF 0x60, BANKED
3	0004	0EFF	MOVLW 0xff
4	0006	6F61	MOVWF 0x61, BANKED
5	0008	5160	MOVF 0x60, W, BANKED
6	000A	6F61	MOVWF 0x61, BANKED
7	000C	EF06	GOTO 0xc
8	000E	F000	NOP
9	0010	FFFF	NOP

Ποιος είναι ο κώδικας της εντολής **goto 0x00C** και σε ποιες θέσεις της μνήμης προγράμματος είναι τοποθετημένος;

Η εντολή **goto 0xXXX** είναι από τις λίγες εντολές των 32 bit. Καταλαμβάνει τις θέσεις μνήμης **000Ch, 000Dh, 000Eh, 000Fh**.

Η θέση μνήμης **000Ch** έχει περιεχόμενο **EFh → 1110 1111b**
Η θέση μνήμης **000Dh** έχει περιεχόμενο **06h → 1111 0110b**
Η θέση μνήμης **000Eh** έχει περιεχόμενο **F0h → 1111 0000b**
Η θέση μνήμης **000Fh** έχει περιεχόμενο **00h → 0000 0000b**

MPLAB, Άσκηση 1 (6)
Πόσες θέσει μνήμης καταλαμβάνει το πρόγραμμα;



Το πρόγραμμα καταλαμβάνει από τη θέση μνήμης **0000h** έως τη θέση μνήμης **000Fh**.

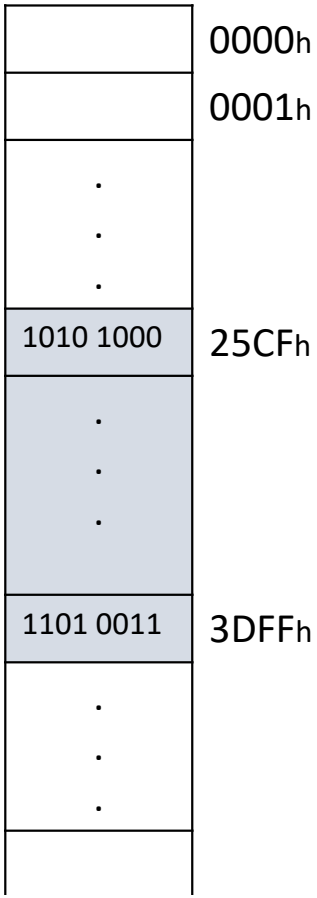
Δηλαδή καταλαμβάνει **000Fh +1** θέσεις μνήμης, δηλαδή **0010h** θέσεις μνήμης.

Δηλαδή καταλαμβάνει $1x16^1 + 0x16^0=16+0=$**16d** θέσεις μνήμης. Σε κάθε θέση μνήμης αποθηκεύεται ένα byte, επομένως το πρόγραμμα είναι 16 byte

MPLAB, Άσκηση 1 (7)

Παράδειγμα υπολογισμού των θέσεων μνήμης που καταλαμβάνει ένα πρόγραμμα

Ένα πρόγραμμα καταλαμβάνει τις θέσεις μνήμης από **25CFh** έως **3DFFh**. Εάν σε κάθε θέση αποθηκεύεται ένα byte (8 bit) πόσα byte είναι το πρόγραμμα; Η απάντηση να δοθεί στο δεκαδικό αριθμητικό σύστημα



$$\begin{array}{r} 3DFF_h \\ -\ \underline{25CF_h} \\ 1830_h \end{array}$$

Επειδή περιλαμβάνεται και η θέση μνήμης 25CFh οι θέσεις μνήμης είναι $1830_h + 1 = 1831_h$.

$1831h = 1 \times 16^3 + 8 \times 16^2 + 3 \times 16^1 + 1 \times 16^0 = 4096 + 8 \times 256 + 48 + 1$
$= 4096 + 2048 + 48 + 1 = 6193_d$ θέσεις μνήμης

Και επειδή σε κάθε θέση μνήμης αποθηκεύεται ένα byte η μνήμη που καταλαμβάνει το πρόγραμμα είναι 6193_d byte

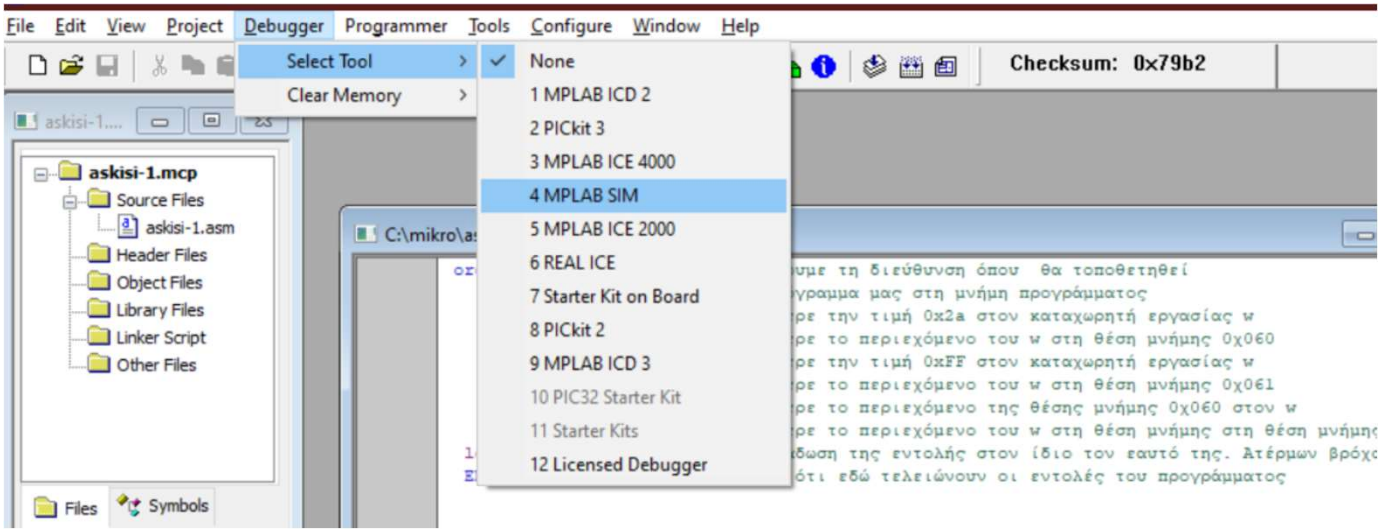
Θα μπορούσαμε να μετατρέψουμε τις διευθύνσεις στο δεκαδικό αριθμητικό σύστημα και να κάνουμε την αφαίρεση στο δεκαδικό αριθμητικό σύστημα

MPLAB, Άσκηση 1 (8)
Εκτέλεση του προγράμματος σε προσομοίωση

C:\mikro\askisi-1\askisi-1.asm

```
org 0x000               ;  Δηλώνουμε τη διεύθυνση όπου  θα τοποθετηθεί
                        ;  το πρόγραμμα μας στη μνήμη προγράμματος
        movlw 0x2A      ;  Μετάφερε την τιμή 0x2a στον καταχωρητή εργασίας w
        movwf 0x060     ;  Μετάφερε το περιεχόμενο του w στη θέση μνήμης 0x060
        movlw 0XFF      ;  Μετάφερε την τιμή 0xFF στον καταχωρητή εργασίας w
        movwf 0x061     ;  Μετάφερε το περιεχόμενο του w στη θέση μνήμης 0x061
        movf 0x060, 0   ;  Μετάφερε το περιεχόμενο της θέσης μνήμης 0x060 στον w
        movwf 0x061     ;  Μετάφερε το περιεχόμενο του w στη θέση μνήμης στη θέση μνήμης 0x061
loop    goto loop       ;  Διακλάδωση της εντολής στον ίδιο τον εαυτό της. Ατέρμων βρόχος.
END                     ; Δήλωση ότι εδώ τελειώνουν οι εντολές του προγράμματος
```

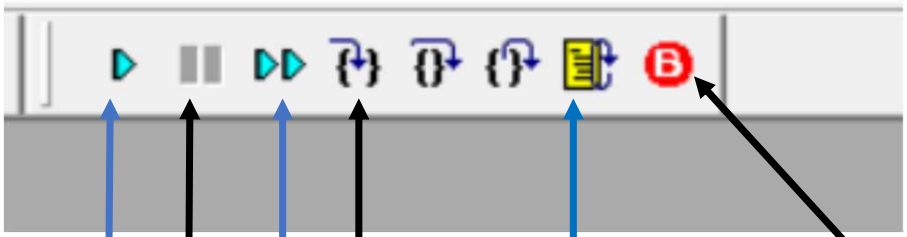
Για να τρέξουμε το πρόγραμμα μας σε προσομοίωνσ επιλέγουμε:
Debugger → Select Tool → MPLAB SIM όπως φαίνεται στην παρακάτω εικόνα



MPLAB, Άσκηση 1 (9)
Εκτέλεση του προγράμματος σε προσομοίωση

Debugger → Select Tool → MPLAB SIM

Μετά την επιλογή του MPLAB SIM εμφανίζονται οι παρακάτω επιλογές



Τρέξιμο του προγράμματος (Run)

Σταμάτημα του προγράμματος (Halt)

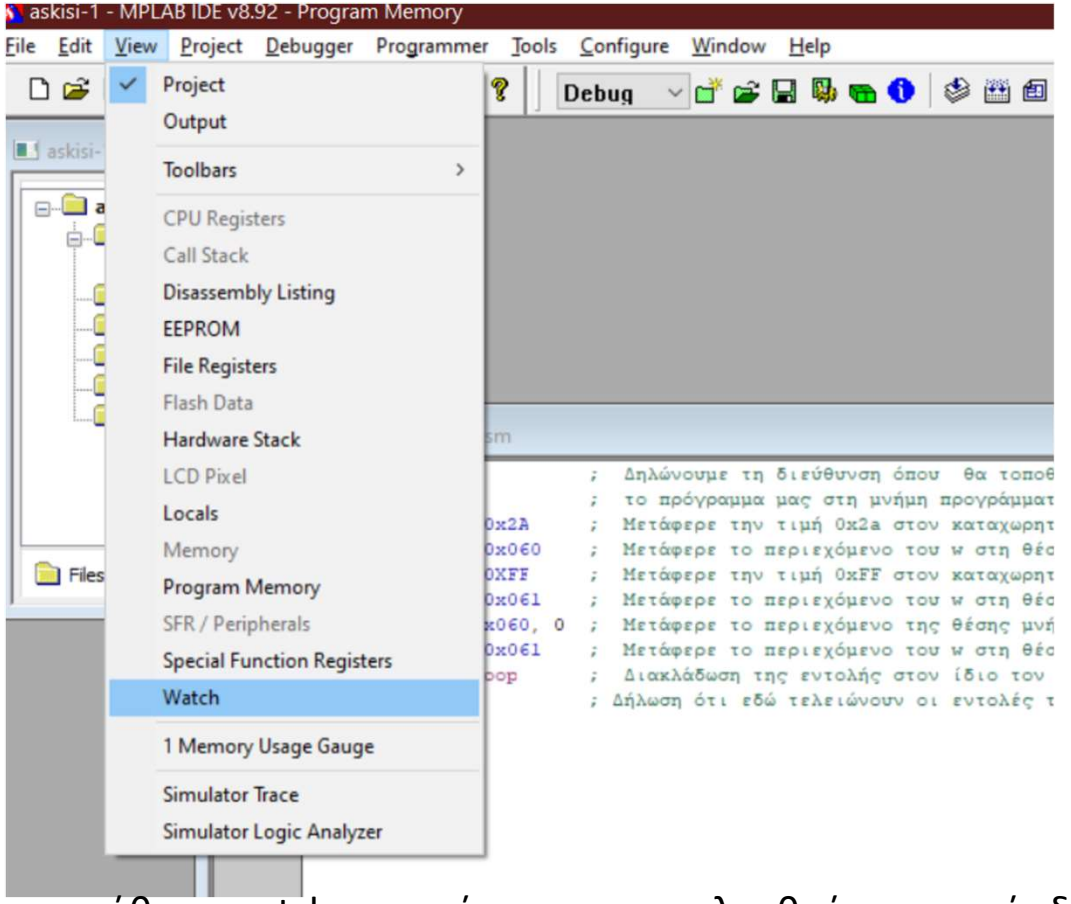
Τρέξιμο σε αργό ρυθμό (animate)

Τρέξιμο εντολή προς εντολή

Επανεκκίνηση (Reset)

Breakpoints
Σημεία διακοπής, μπορούμε να διακόψουμε το πρόγραμμα σε κάποιο σημείο και να εξετάσουμε τις τιμές διάφορων καταχωρητών

MPLAB Άσκηση 1 (10)
Παρακολούθηση των τιμών καταχωρητών και θέσεων μνήμης



Από το παράθυρο watch μπορούμε να παρακολουθούμε τις τιμές διάφορων καταχωρητών καθώς και θέσεων μνήμης ενώ εκτελείται το πρόγραμμα. Για παράδειγμα μπορούμε να παρακολουθούμε την τιμή του W (καταχωρητής εργασίας) και των θέσεων μνήμης 060h και 061h

Άσκηση 2 (1)

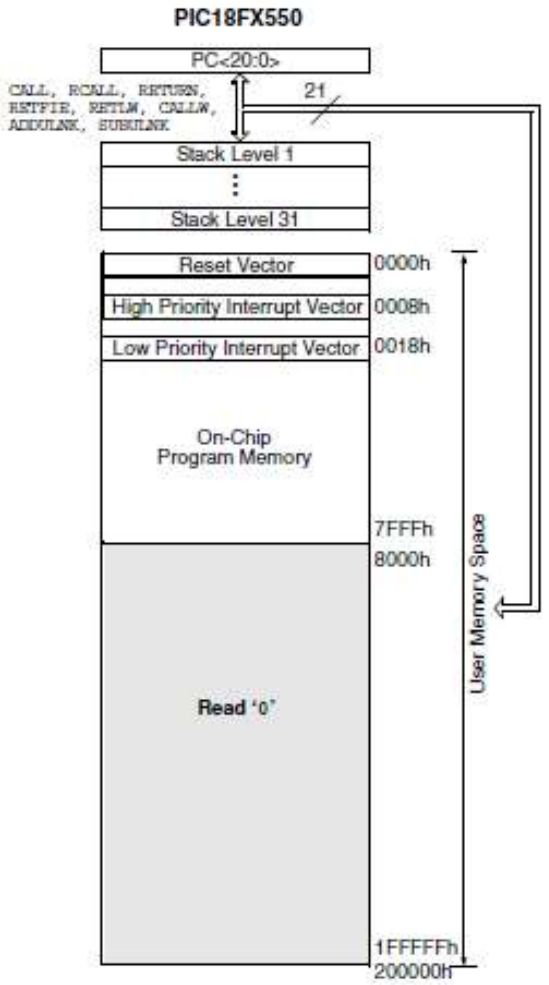
•Να γίνει προσομοίωση του παρακάτω προγράμματος και να παρακολουθηθεί η τιμή των καταχωρητών W, 0x060, 0x061 και 0x062 καθώς εκτελείται εντολή προς εντολή.

•Στη συνέχεια να γίνει προσομοίωση του ίδιου προγράμματος αλλά αντί για τις θέσεις μνήμης 0x060, 0x061 και 0x062 να χρησιμοποιηθούν οι θέσεις μνήμης 0x161, 0x262, 0x363.

•Εκτός από τις τιμές των θέσεων μνήμης τι επιπλέον αλλαγές θα πρέπει να γίνουν στο πρόγραμμα;

•Στη συνέχεια να τοποθετηθεί το πρόγραμμα στη μνήμη προγράμματος 0x800 και μετά. Σ' αυτή την περίπτωση τι θα γράψουμε στη διεύθυνση 0X000 (Reset Vector) ώστε όταν γίνει επανεκκίνηση του προγράμματος να γίνεται μετάβαση στη διεύθυνση 0x800;

•Τι είναι το Reset Vector;



Δημιουργία Project στο MPLAB για την άσκηση 2

Που θα τοποθετήσουμε το project ;

Δημιουργούμε τον φάκελο:

C:\mikro\askisi-2

Σ’ αυτό τον φάκελο θα τοποθετήσουμε το Project

Τελικό σκοπός

Ο τελικός σκοπός του Project είναι να μεταφραστεί το πρόγραμμα από γλώσσα Assembly σε γλώσσα μηχανής, να φορτωθεί στη μνήμη προγράμματος του μικροελεγκτή, και να εκτελεστεί.

```
        org 0x000          ; Δηλώνουμε τη θέση στη
                           ;μνήμη προγράμματος
                           ; στην οποία θέλουμε
                           ; να τοποθετηθεί το αποτέλεσμα
                           ; της  μετάφρασης  σε
                           ; γλώσσα μηχανής
                           ; των παρακάτω εντολών .
        movlw  B'11111101' ; Φόρτωσε στον W
                           ; την τιμή 253d
loop    movwf  0x060       ; Μετάφερε την τιμή του
                           ; W στη θέση
                           ; μνήμης 0x060
        movwf 0x061        ; Μετάφερε την τιμή του
                           ; W στη θέση μνήμης 0x061
        movwf 0x062        ; Μετάφερε την τιμή του W
                           ; στη θέση μνήμης 0x062
        addlw  0x01        ; Πρόσθεσε στον W την τιμή 1
        goto  loop         ; Πήγαινε στη διεύθυνση
                           ; loop
END                        ; Δηλώνουμε το τέλος του
                           ; προγράμματος
```

Άσκηση 2(2)

```
org 0x000                          ; Δηλώνουμε τη θέση στη μνήμη προγράμματος
                                   ; στην οποία θέλουμε να τοποθετηθεί το αποτέλεσμα
                                   ; της  μετάφρασης  σε γλώσσα μηχανής
                                   ;  των παρακάτω εντολών .
          movlw  B'11111101' ; Φόρτωσε στον W την τιμή 253d
loop      movwf  0x060        ; Μετάφερε την τιμή του W στη θέση μνήμης 0x060
          movwf 0x061         ; Μετάφερε την τιμή του W στη θέση μνήμης 0x061
          movwf 0x062         ; Μετάφερε την τιμή του W στη θέση μνήμης 0x062

          addlw  0x01         ; Πρόσθεσε στον W την τιμή 1
           goto  loop         ; Πήγαινε στη διεύθυνση loop
END                           ; Δηλώνουμε το τέλος του προγράμματος
```

****

Προσέξτε την τιμή που θα πάρει ο W μετά την τιμή 255d

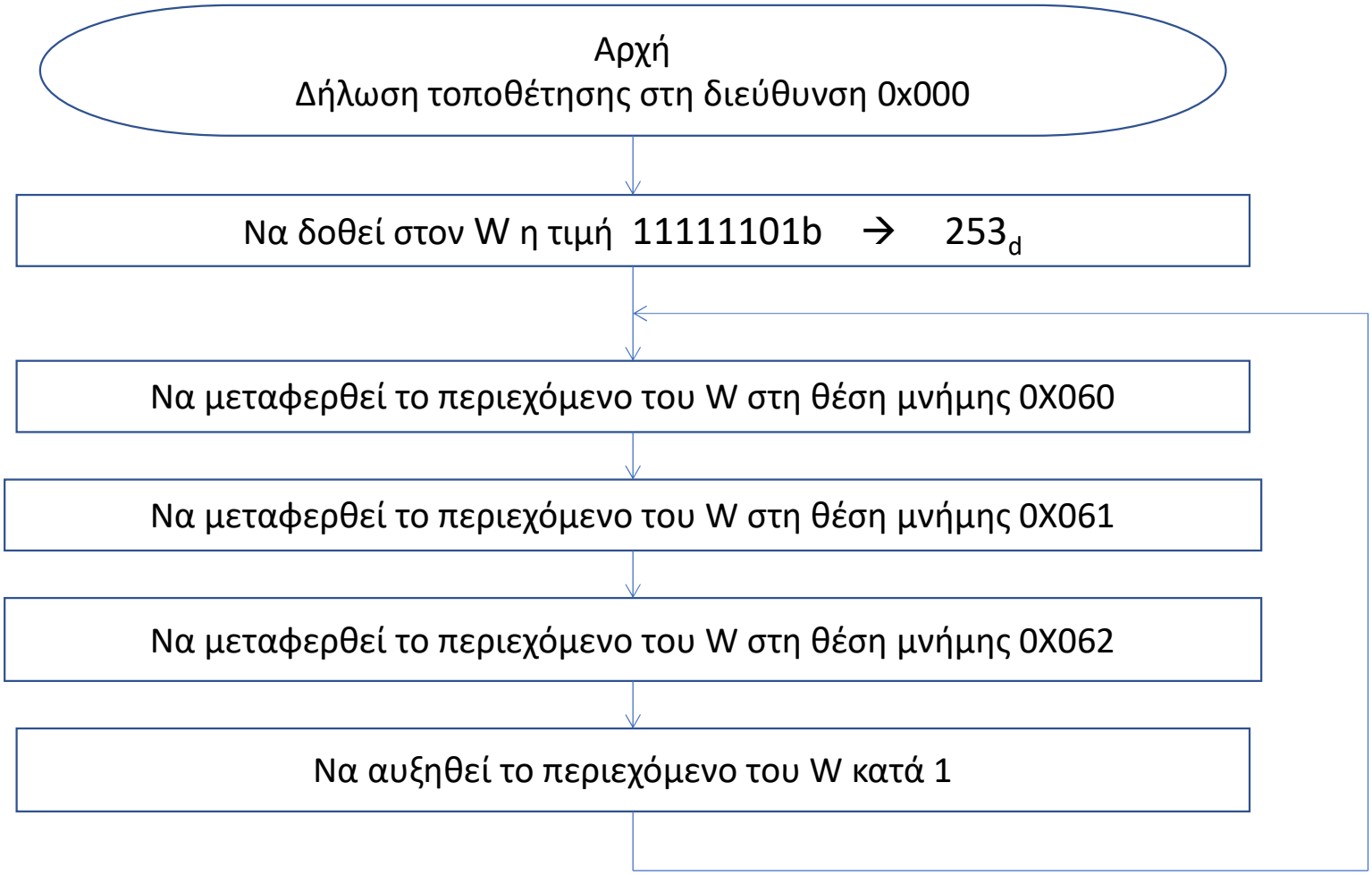
255+1 → 0

Προσέξτε τη μετάφραση σε γλώσσα μηχανής των εντολών :

```
movwf 0x060    ; Μετάφερε την τιμή του W στη θέση μνήμης 0x060
movwf 0x061    ; Μετάφερε την τιμή του W στη θέση μνήμης 0x061
movwf 0x062    ; Μετάφερε την τιμή του W στη θέση μνήμης 0x0602
```

Άσκηση 2 (3)

Διάγραμμα ροής του προγράμματος 2

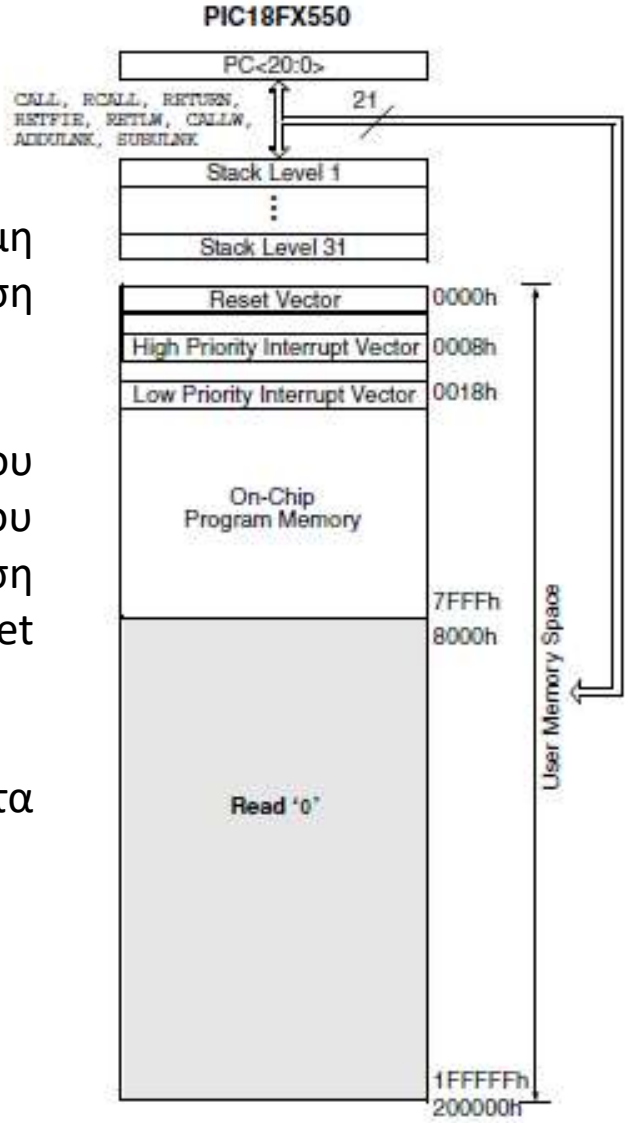


Τι είναι το Reset Vector;

•Το Reset Vector είναι η διεύθυνση στη μνήμη προγράμματος από την οποία ξεκινάει η εκτέλεση του προγράμματος ενός μικροελεγκτή .

•Όταν γίνει επανεκκίνηση (Reset) του προγράμματος ενός μικροελεγκτή η εκτέλεση του προγράμματος ξεκινάει επίσης από τη διεύθυνση στη μνήμη προγράμματος η οποία λέγεται Reset Vector.

•Το Reset Vector θα μπορούσε να μεταφραστεί στα Ελληνικά σαν «Διεύθυνση Επανεκκίνησης».



Τοποθέτηση του προγράμματος της άσκησης 2 στη διεύθυνση 0x800

```
org 0x000                          ; Δηλώνουμε τη θέση στη μνήμη προγράμματος
                                   ; στην οποία θέλουμε να τοποθετηθεί το αποτέλεσμα
                                   ; της  μετάφρασης  σε γλώσσα μηχανής
                                   ;  των παρακάτω εντολών .
            goto 0x800             ;  Πήγαινε στη διεύθυνση 0x800
                                   ;  Δίνουμε οδηγία στον Program Counter να πάρει την
                                   ;  τιμή 0x800, δηλαδή δίνουμε οδηγία να συνεχιστεί
                                   ;  η εκτέλεση του προγράμματος από τη διεύθυνση
                                   ;  0x800
org 0x800                          ; Δηλώνουμε τη θέση στη μνήμη προγράμματος
                                   ; στην οποία θέλουμε να τοποθετηθεί το αποτέλεσμα
                                   ; της  μετάφρασης  σε γλώσσα μηχανής
                                   ;  των παρακάτω εντολών .
            movlw  B'11111101' ; Φόρτωσε στον W την τιμή 253d
loop        movwf  0x060           ; Μετάφερε την τιμή του W στη θέση μνήμης 0x060
            movwf 0x061            ; Μετάφερε την τιμή του W στη θέση μνήμης 0x061
            movwf 0x062            ; Μετάφερε την τιμή του W στη θέση μνήμης 0x062
            addlw  0x01            ; Πρόσθεσε στον W την τιμή 1
             goto  loop            ; Πήγαινε στη διεύθυνση loop
END                                ; Δηλώνουμε το τέλος του προγράμματος
```

Χρήση των θέσεων μνήμης 0x160, 0x261, 0x362

•Δοκιμάστε το παρακάτω πρόγραμμα και ελέγξετε αν υπάρχει το επιθυμητό αποτέλεσμα. Παρακολουθείστε από το παράθυρο watch τα περιεχόμενα των θέσεων μνήμης 0x060, 0x061, 0x062, 0x160, 0x261, 0x362

•Θα διαπιστώσετε ότι δεν υπάρχει το επιθυμητό αποτέλεσμα!

•Το γιατί εξηγείται στην επόμενη διαφάνεια.

```
org 0x000                    ; Δηλώνουμε τη θέση στη μνήμη προγράμματος
                             ; στην οποία θέλουμε να τοποθετηθεί το αποτέλεσμα
                             ; της  μετάφρασης  σε γλώσσα μηχανής
                             ;  των παρακάτω εντολών .

          movlw  B'11111101' ; Φόρτωσε στον W την τιμή 253d
loop      movwf  0x160       ; Μετάφερε την τιμή του W στη θέση μνήμης 0x160
          movwf 0x261        ; Μετάφερε την τιμή του W στη θέση μνήμης 0x261
          movwf 0x362        ; Μετάφερε την τιμή του W στη θέση μνήμης 0x362

          addlw  0x01        ; Πρόσθεσε στον W την τιμή 1
           goto  loop        ; Πήγαινε στη διεύθυνση loop
END                          ; Δηλώνουμε το τέλος του προγράμματος
```

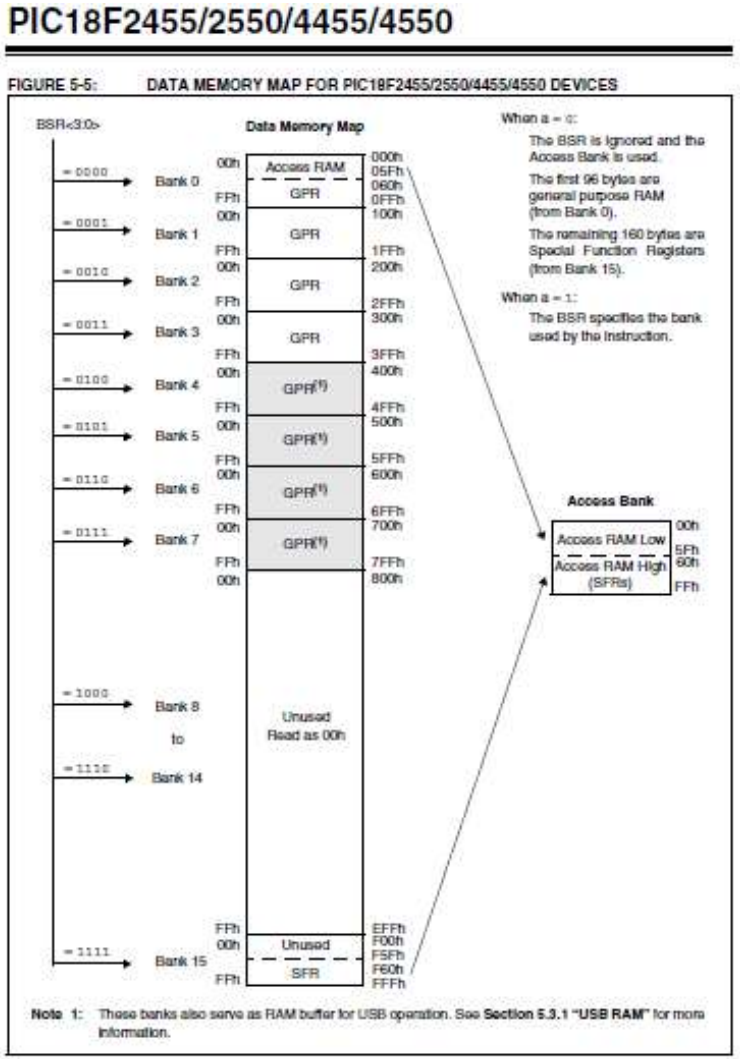
Ορισμός bank πριν τη χρήση θέσεων μνήμης

•Πριν χρησιμοποιήσουμε μια θέση μνήμης θα πρέπει να ορίσουμε το bank στο οποίο ανήκει. Αν στο πρόγραμμα δεν ορίσουμε καθόλου bank οι θέσεις μνήμης θεωρείται ότι ανήκουν στο bank 0.

•Από τη στιγμή που ορίσουμε το bank αυτό παραμένει το ίδιο έως ότου το αλλάξουμε.

•Παράδειγμα 1:
Πριν χρησιμοποιήσουμε την εντολή movwf 0x160 θα πρέπει να ορίσουμε το bank με την εντολή:
movlb 0x01 ; Δώσε την τιμή 1 στον bank register

•Παράδειγμα 2:
Πριν χρησιμοποιήσουμε την εντολή movwf 0x260 θα πρέπει να ορίσουμε το bank με την εντολή:
movlb 0x02 ; Δώσε την τιμή 2 στον bank register



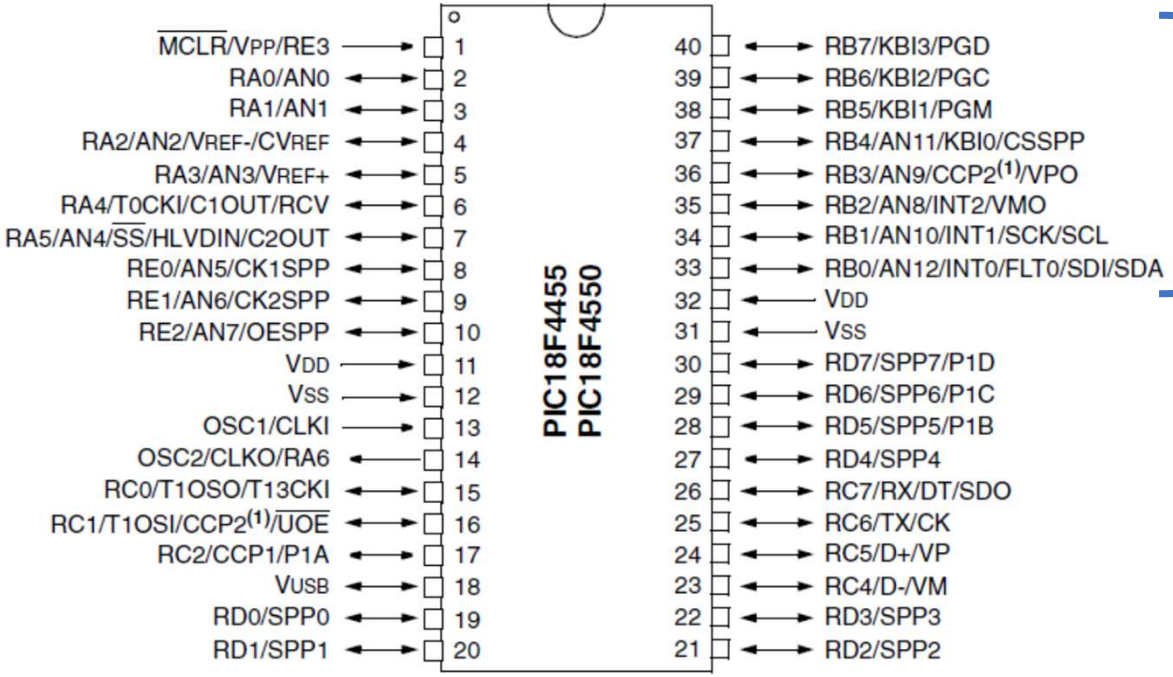
Το σωστό πρόγραμμα για τη χρήση των θέσεων μνήμης 0x160, 0x261, 0x362

```
org 0x000                    ; Δηλώνουμε τη θέση στη μνήμη προγράμματος
                             ; στην οποία θέλουμε να τοποθετηθεί το αποτέλεσμα
                             ; της  μετάφρασης  σε γλώσσα μηχανής
                             ;  των παρακάτω εντολών .

        movlw  B'11111101' ; Φόρτωσε στον W την τιμή 253dloop
        movlb   0X01         ;  Δώσε την τιμή 1 στον καταχωρητή bank (bank register)
        movwf  0x160         ; Μετάφερε την τιμή του W στη θέση μνήμης 0x160
        movlb   0x02         ; Δώσε την τιμή 2 στον καταχωρητή bank (bank register)
        movwf 0x261          ; Μετάφερε την τιμή του W στη θέση μνήμης 0x261
        movlb   0x03         ; Δώσε την τιμή 3 στον καταχωρητή bank (bank register)
        movwf 0x362          ; Μετάφερε την τιμή του W στη θέση μνήμης 0x362
        addlw  0x01          ; Πρόσθεσε στον W την τιμή 1
         goto  loop          ; Πήγαινε στη διεύθυνση loop
END                          ; Δηλώνουμε το τέλος του προγράμματος
```

****

Αφού ορίσουμε το bank 3, δεν έχει σημασία αν γράφουμε **movwf 0x362** ή movwf **0x62**.
Στο πρόγραμμα στο MPLAB να παρακολουθήσετε τις τιμές των θέσεων μνήμης 0x160, 0x261 και 0x362 από το παράθυρο watch.

Παράλληλες πόρτες μικροελεγκτή (Parallel ports)

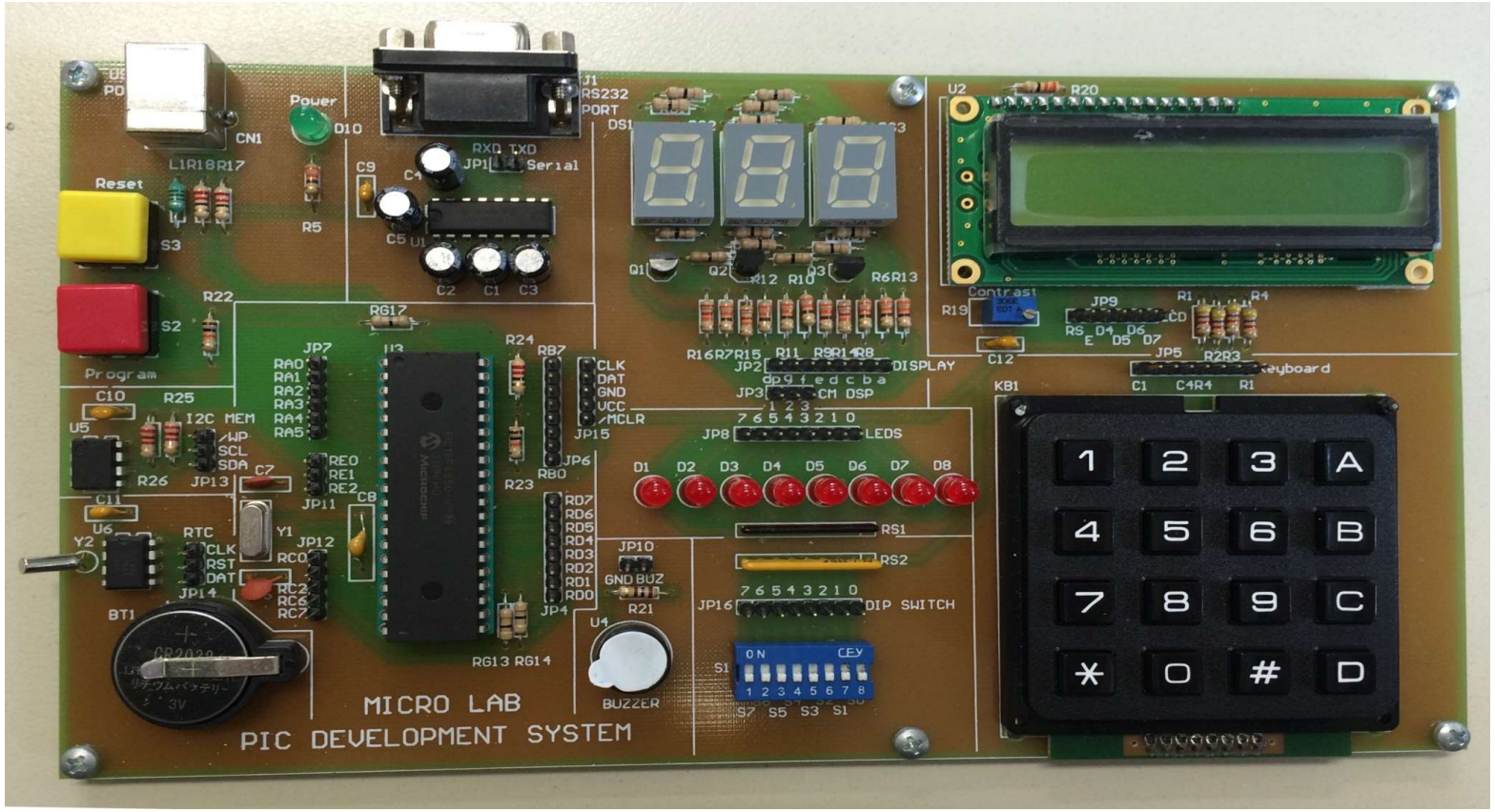


Παράλληλη πόρτα Β

Οι 8 ακροδέκτες
RB7, RB6, … RB1, RB0

Παράλληλη πόρτα είναι **ένα σύνολο ακροδεκτών** από το οποίο μπορεί να εισάγεται συγχρόνως στον μικροελεγκτή μια ψηφιακή λέξη ή να αποστέλλεται συγχρόνως από τον μικροελεγκτή μια ψηφιακή λέξη προς τον «έξω κόσμο».
Για παράδειγμα οι 8 ακροδέκτες RB7, RB6, RB4, . . . RB0 αποτελούν την παράλληλη πόρτα Β. Μπορεί να διαβάζεται μια λέξη των 8 bit ή να στέλνεται μια λέξη των 8 bit

Πλακέτα ανάπτυξης εφαρμογών του εργαστηρίου Ενσωματωμένα Συστήματα



Παράλληλη πόρτα Β (1)

Special Function Registers

Καταχωρητής κατεύθυνσης της πόρτας Β

PORTB Direction Register, TRISB, Είναι ένας Special Function Register

Είναι στη διεύθυνση F93h

1	1	0	0	0	0	0	1

F96h	TRISE(3)
F95h	TRISD(3)
F94h	TRISC
F93h	TRISB
F92h	TRISA
F91h	—(2)

Καταχωρητής δεδομένων της πόρτας Β

PORTB Data Register, PORTB, Είναι ένας Special Function Register

Είναι στη διεύθυνση F81h

F84h	PORTE
F83h	PORTD(3)
F82h	PORTC
F81h	PORTB
F80h	PORTA

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Είσοδος	Είσοδος	Έξοδος	Έξοδος	Έξοδος	Έξοδος	Έξοδος	Είσοδος

Ακροδέκτες της πόρτας Β

Οι τιμές που τοποθετούνται στον καταχωρητή κατεύθυνσης της πόρτας Β καθορίζουν ποιοι ακροδέκτες θα είναι είσοδοι και ποιοι θα είναι έξοδοι

1 → ο αντίστοιχος ακροδέκτης της πόρτας Β θα είναι είσοδος

0 → ο αντίστοιχος ακροδέκτης της πόρτας Β θα είναι έξοδος

Παράλληλη πόρτα Β(2)

Να γραφούν οι εντολές με τις οποίες οι ακροδέκτες RB7, RB6, RB5, RB4 της πόρτας Β γίνονται είσοδοι και οι ακροδέκτες RB3, RB2, RB1 και RB0 της πόρτας Β γίνονται έξοδοι.

```
movlw B'11110000' ; Φόρτωσε στον W την τιμή 11110000b
movwf TRISB       ; Στείλε το περιεχόμενο του W στον καταχωρητή κατεύθυνσης της Πόρτας Β
```

- *Τοποθετώντας την κατάλληλη τιμή στον καταχωρητή κατεύθυνσης της πόρτας Β καθορίζουμε ποιοι ακροδέκτες της πόρτας Β θα είναι είσοδοι και ποιοι θα είναι έξοδοι.*
- *Χρησιμοποιούμε το όνομα TRISB για τον καταχωρητή κατεύθυνσης της πόρτας Β και PORT Β για τον καταχωρητή δεδομένων της πόρτας Β*

PORTB Direction Register, TRISB , καταχωρητής κατεύθυνσης της πόρτας Β

1	1	1	1	0	0	0	0

PORTB Data Register, PORTB, καταχωρητής δεδομένων της πόρτας Β

Ακροδέκτες της πόρτας Β

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Είσοδος	Είσοδος	Είσοδος	Είσοδος	Έξοδος	Έξοδος	Έξοδος	Έξοδος

PIC 18F4550 Data Sheet, εντολές του PIC 18F4550

- Στο manual του Μικροελεγκτή PIC 18F4550 υπάρχει πίνακας με όλες τις εντολές του μικροελεγκτή σε γλώσσα Assembly. Περιγράφεται το τι κάνουν, καταγράφεται ο κώδικας τους σε γλώσσα μηχανής, πόσους κύκλους μηχανής χρειάζονται για να εκτελεστούν, και υπάρχουν παραδείγματα χρήσης τους.
- Δύο παραδείγματα από τις εντολές movlw k και την εντολή goto k

MOVLW **Move Literal to W**

Syntax:	MOVLW k
Operands:	$0 \le k \le 255$
Operation:	$k \rightarrow W$
Status Affected:	None
Encoding:	0000 1110 kkkk kkkk
Description:	The eight-bit literal 'k' is loaded into W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `MOVLW 5Ah`

After Instruction
W = 5Ah

GOTO **Unconditional Branch**

Syntax:	GOTO k
Operands:	$0 \le k \le 1048575$
Operation:	$k \rightarrow PC<20:1>$
Status Affected:	None
Encoding: 1st word (k<7:0>) 2nd word(k<19:8>)	1110 1111 $k_7$kkk kkkk kkkk$_0$ k_{19}kkk kkkk kkkk$_8$
Description:	GOTO allows an unconditional branch anywhere within the entire 2-Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.
Words:	2
Cycles:	2

Q Cycle Activity:

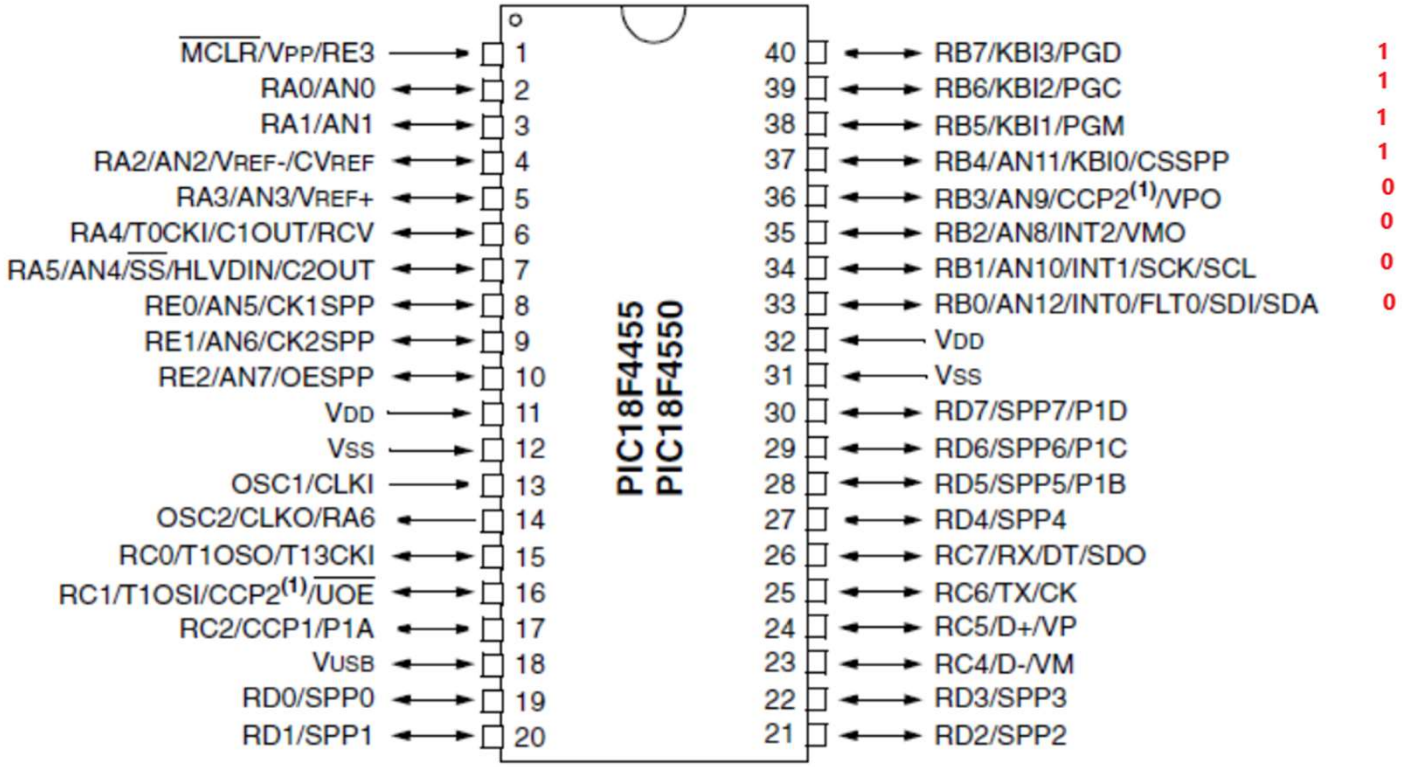
Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	No operation	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `GOTO THERE`

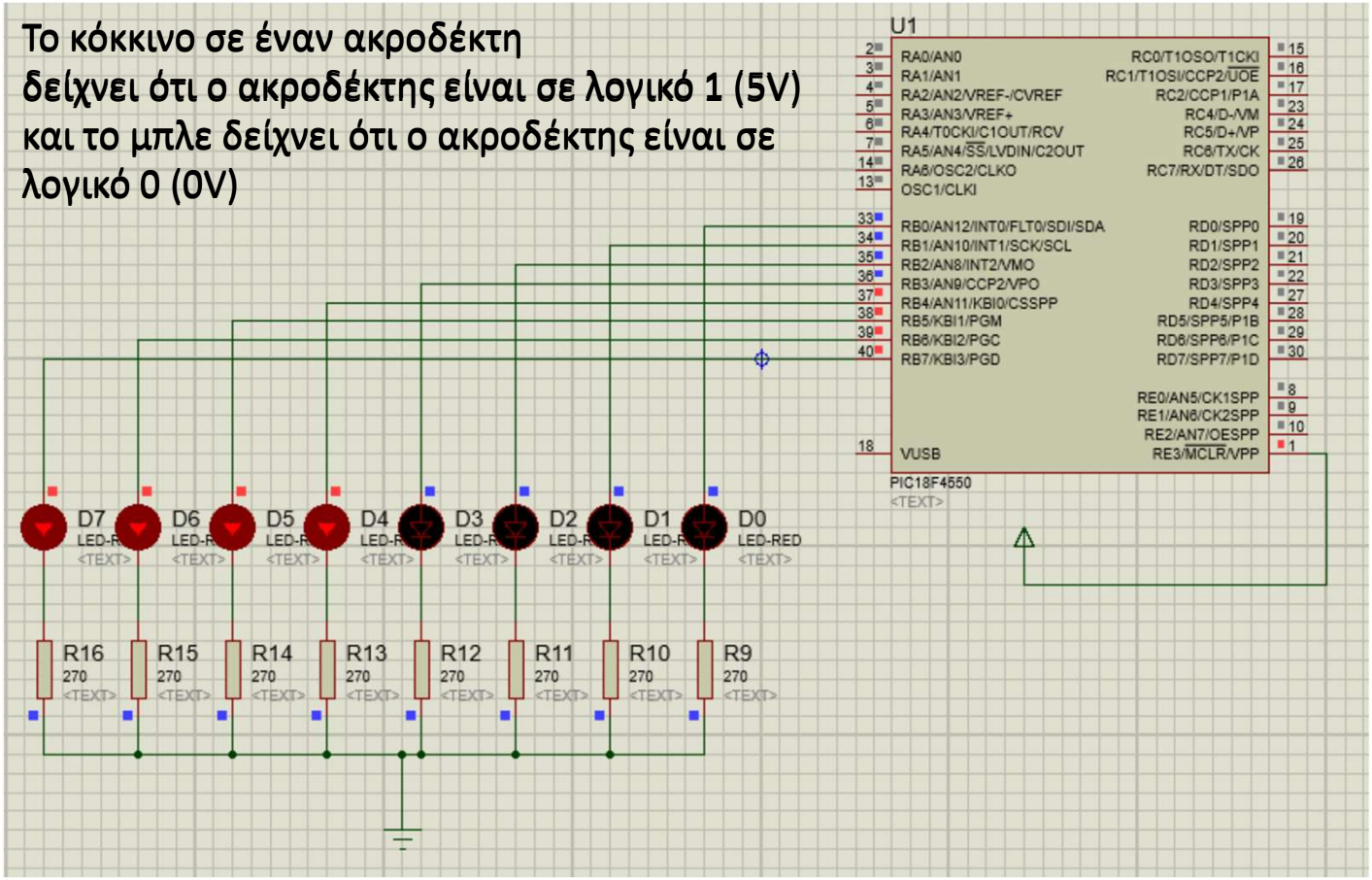
After Instruction
PC = Address (THERE)

Άσκηση 3 (1)
Χρήση της πόρτας Β σαν πόρτας εξόδου

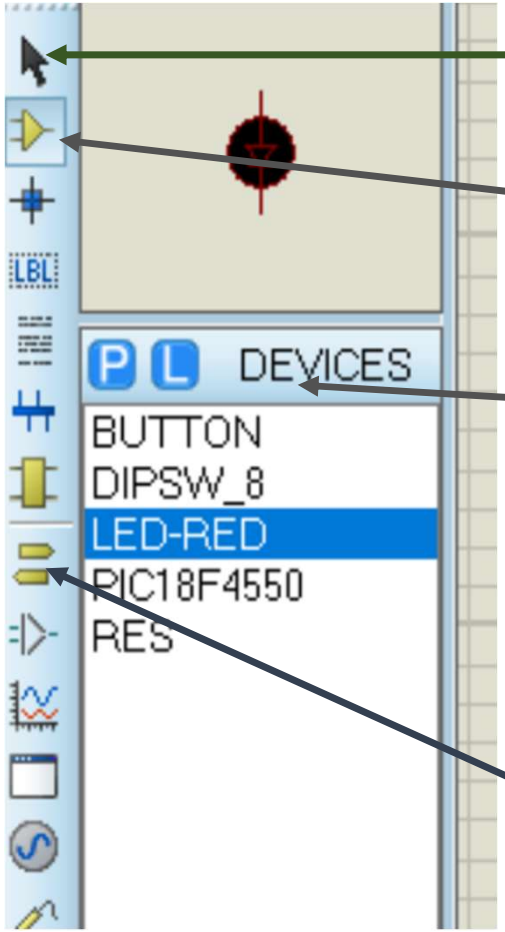
Να γραφεί πρόγραμμα ώστε να εμφανιστεί στην πόρτα Β του μικροελεγκτή PIC18F4550 η τιμή 11110000



Άσκηση 3(2)
Χρήση της πόρτας Β σαν πόρτας εξόδου



Πρόγραμμα σχεδίασης και προσομοίωσης ηλεκτρονικών κυκλωμάτων Proteus



Επιλογή περιοχής για αντιγραφή ή μετακίνηση. Selection mode.

Επιλογή εξαρτήματος . Component mode.
Πατάμε το κουμπί για εισαγωγή εξαρτημάτων και στη συνέχεια το P από το P L DEVICES για να επιλέξουμε το εξάρτημα που θέλουμε να εισάγουμε.
Αφού επιλέξουμε και εισάγουμε ένα εξάρτημα, αυτό εμφανίζεται στη λίστα P L DEVICES
Στη συνέχεια πιάνουμε το εξάρτημα από τη λίστα P L DEVICES και το μεταφέρουμε στην επιφάνεια εργασίας

Εισαγωγή τροφοδοσίας(POWER) και γείωσης (GROUND).
Πιάνουμε το POWER ή το GROUND και το μεταφέρουμε στην επιφάνεια εργασίας.

Άσκηση 3(3)

Ο κώδικας του προγράμματος με το οποίο εμφανίζεται η τιμή 11110000 στους ακροδέκτες της πόρτας Β

```
;**************** Πρόγραμμα για τοποθέτηση μιας τιμής στην πόρτα εξόδου Β *******
            org 0x000           ;Το πρόγραμμα ξεκινάει από την διεύθυνση (000)h της μνήμης
                                ;προγράμματος
TRISB       EQU 0xF93           ; Αποδίδουμε στη διεύθυνση 0xF93 το όνομα TRISB που κατά κανόνα
                                ; χρησιμοποιούμε για τον καταχωρητή κατεύθυνσης της πόρτας Β
PORTB       EQU 0xF81           ; Αποδίδουμε στη διεύθυνση 0xF81 το όνομα PORTB που κατά κανόνα
                                ; χρησιμοποιούμε για τον καταχωρητή δεδομένων της πόρτας Β
                                ; Με την οδηγία EQU αποδίδουμε ονόματα σε διευθύνσεις θέσεων μνήμης.
                                ; Το EQU είναι οδηγία και όχι εντολή. Επομένως δεν μεταφράζεται σε
                                ; κώδικα σε γλώσσα μηχανής

                                ; 1ον Ορίζουμε την πόρτα Β σαν πόρτα εξόδου
            movlw B'00000000'   ; φορτώνουμε στον w την τιμή 0000 0000
            movwf TRISB         ; μεταφέρουμε το περιεχόμενο του w στον καταχωρητή
                                ; κατεύθυνσης της πόρτας Β για να την κάνουμε έξοδο
                                ; Θα μπορούσαμε να αντικαταστήσουμε το TRISB με το 0xF93

                                ; 2ον Θα δώσουμε στην πόρτα Β την αρχική τιμή 1111 0000b
            movlw B'11110000'   ; φορτώνουμε στον W την τιμή 1000 0000
            movwf PORTB         ; στέλνουμε την τιμή 1111 0000 στην πόρτα Β για
                                ; ανάψουν τα 4 αριστερά LED (RB7, most significant bit).
                                ; Θα μπορούσαμε να αντικαταστήσουμε το PORTB με την διεύθυνση 0xF81
                                ; δηλαδή με τον καταχωρητή δεδομένων της πόρτας Β

loop        goto  loop          ; ατέρμων βρόχος (η εντολή goto κάνει διακλάδωση στον εαυτό της)
            END
```

Άσκηση 3(4)

Ο κώδικας του προγράμματος με το οποίο εμφανίζεται η τιμή 11110000 στους ακροδέκτες της πόρτας Β

```
;**************** Πρόγραμμα για τοποθέτηση μιας τιμής στην πόρτα εξόδου Β *******

#include  <set_fuse.inc>  ;Προσοχή! Στον φάκελο με το project σας θα
                          ;μεταφέρετε το αρχείο set_fuse.inc. Υπάρχει στο Moodle.
                          ;Περιλαμβάνει αρχικές ρυθμίσεις για τον μικροελεγκτή.
                          ; Σ' αυτό το αρχείο καθορίζεται ότι PORTB είναι η διεύθυνση
                         ; F81h δηλαδή η διεύθυνση του καταχωρητή δεδομένων της πόρτας Β και
                         ; ότι TRISB είναι η διεύθυνση 0xF93, δηλαδή η διεύθυνση του καταχωρητή
                         ; κατεύθυνσης της πόρτας Β
                         ; ΣΥΝΗΘΙΖΕΤΑΙ ΣΕ ΕΝΑ ΤΕΤΟΙΟ ΞΕΧΩΡΙΣΤΟ ΑΡΧΕΙΟ ΝΑ ΤΟΠΟΘΕΤΟΥΜΕ
                         ; ΤΙΣ ΑΡΧΙΚΕΣ ΡΥΘΜΙΣΕΙΣ ΤΟΥ ΜΙΚΡΟΕΛΕΓΚΤΗ.
org 0x000                ;Το πρόγραμμα ξεκινάει από την διεύθυνση (000)h της μνήμης
                         ;προγράμματος

                          ; 1ον Ορίζουμε την πόρτα Β σαν πόρτα εξόδου
movlw B'00000000'        ; φορτώνουμε στον w την τιμή 0000 0000
movwf TRISB               ; μεταφέρουμε το περιεχόμενο του w στον καταχωρητή
                          ; κατεύθυνσης της πόρτας Β για να την κάνουμε έξοδο
                          ; Θα μπορούσαμε να αντικαταστήσουμε το TRISB με το 0xF93

                          ; 2ον Θα δώσουμε στην πόρτα Β την αρχική τιμή 1111 0000b
movlw B'11110000'        ; φορτώνουμε στον W την τιμή 1000 0000
movwf PORTB               ; στέλνουμε την τιμή 1111 0000 στην πόρτα Β για
                          ; Ανάψουν τα 4 αριστερά LED (RB7, most significant bit)
                          ; Θα μπορούσαμε να αντικαταστήσουμε το PORTB με την διεύθυνση 0xF81
                          ; δηλαδή με τον καταχωρητή δεδομένων της πόρτας Β
```

Άσκηση 4(1)

Ο κώδικας του προγράμματος με το οποίο αναβοσβήνουν 4 LED που είναι συνδεδεμένα στους ακροδέκτες RB7, RB6, RB5, RB4 του μικροελεγκτή

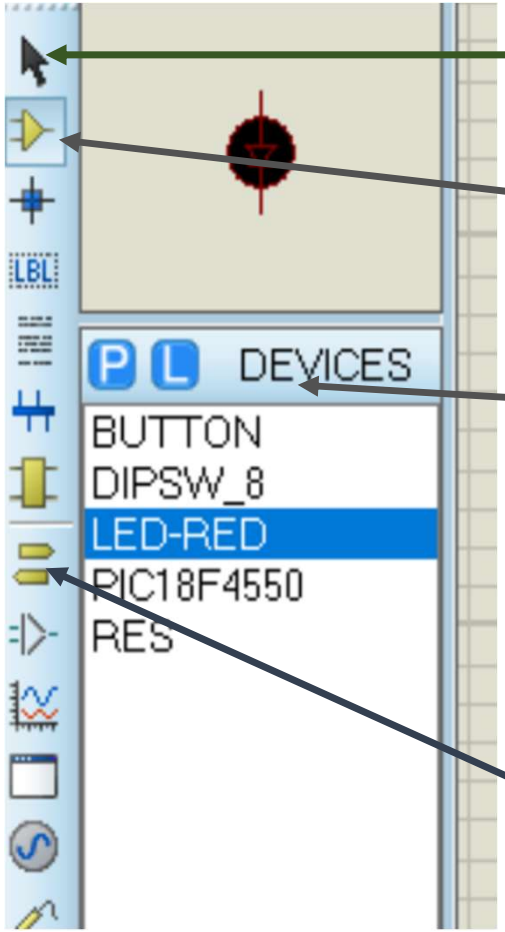
```
;**************** Πρόγραμμα για αναβόσβημα LED στην πόρτα εξόδου Β *******
            org 0x000           ;Το πρόγραμμα ξεκινάει από την διεύθυνση (000)h της μνήμης
                                ;προγράμματος
TRISB       EQU 0xF93           ; Αποδίδουμε στη διεύθυνση 0xF93 το όνομα TRISB που κατά κανόνα
                                ; χρησιμοποιούμε για τον καταχωρητή κατεύθυνσης της πόρτας Β
PORTB       EQU 0xF81           ; Αποδίδουμε στη διεύθυνση 0xF81 το όνομα PORTB που κατά κανόνα
                                ; χρησιμοποιούμε για τον καταχωρητή δεδομένων της πόρτας Β
                                ; Με την οδηγία EQU αποδίδουμε ονόματα σε διευθύνσεις θέσεων μνήμης.
                                ; Το EQU είναι οδηγία και όχι εντολή. Επομένως δεν μεταφράζεται σε
                                ; κώδικα σε γλώσσα μηχανής
reg1        EQU 0x60            ; Απόδοση του ονόματος reg1 στη θέση μνήμης 060h
reg2        EQU 0x61            ; Απόδοση του ονόματος reg2 στη θέση μνήμης 061h
reg3        EQU 0x62            ; Απόδοση του ονόματος reg3 στη θέση μνήμης 062h
                                ; 1ο Ορίζουμε την πόρτα Β σαν πόρτα εξόδου
            movlw B'00000000'   ; φορτώνουμε στον w την τιμή 0000 0000
            movwf TRISB         ; μεταφέρουμε το περιεχόμενο του w στον καταχωρητή
                                ; κατεύθυνσης της πόρτας Β για να την κάνουμε έξοδο
                                ; Θα μπορούσαμε να αντικαταστήσουμε το TRISB με το 0xF93

                                ; 2ο Θα ανάβουμε και θα σβήνουμε τα LED
loop        movlw B'11110000'   ; φορτώνουμε στον W την τιμή 1111 0000
            movwf PORTB         ; στέλνουμε την τιμή του W στην πόρτα Β (ανάβουν τα LED)
            call delay_1s       ; Κάλεσμα ρουτίνας καθυστέρησης ενός δευτερολέπτου
            movlw B'00000000'   ; φορτώνουμε στον W την τιμή 0000 0000
            movwf PORTB         ; Στέλνουμε την τιμή του W στην πόρτα Β (σβήνουν τα LED)
            call delay_1s       ; Κάλεσμα ρουτίνας καθυστέρησης ενός δευτερολέπτου
            goto  loop          ; Διακλάδωση για επανάληψη του αναβοσβησίματος
```

Άσκηση 4(2)

Συνέχεια του κώδικα της άσκησης 4 (η ρουτίνα καθυστέρησης τοποθετείται μετά το πρόγραμμα)

```
delay-1s          ;Αυτό είναι το όνομα της ρουτίνας καθυστέρησης.
                  ;Είναι ετικέτα και αντιστοιχεί στη διεύθυνση που είναι
                  ;γραμμένη η ρουτίνα καθυστέρησης.
        movlw 0x20    ;Προσοχή θα πρέπει να αντιστοιχήσουμε μια διεύθυνση
                  ;Στην προσομοίωση στο Proteus βάζουμε τιμή 0x02
                  ;γιατί το πρόγραμμα στο Proteus ποιο αργά απ' όσο στον
                  ;πραγματικό μικροελεγκτή.
        movwf reg3    ; στο όνομα reg3  (με EQU, στην αρχή του προγράμματος)
loop1             ; Το loop1 είναι ετικέτα, αντιστοιχεί στη διεύθυνση
                  ;Της επόμενης εντολής
        movlw 0x00    ;Προσοχή θα πρέπει να αντιστοιχήσουμε μια διεύθυνση
        movwf reg1    ;στο όνομα reg1 (με EQU στην αρχή του προγράμματος)
loop2             ;Το loop2 είναι ετικέτα, αντιστοιχεί στη διεύθυνση
                  ;της επόμενης εντολής
        movlw 0x00    ;Προσοχή θα πρέπει να αντιστοιχήσουμε μια διεύθυνση
        movwf reg2    ;στο όνομα reg2  (με EQU στην αρχή του προγράμματος)
loop3             ;Το loop3 είναι ετικέτα, αντιστοιχεί στη διεύθυνση
                  ; της επόμενης εντολής
      decfsz reg2,1
        goto loop3
        decfsz reg1,1
        goto loop2
        decfsz reg3,1
        goto loop1
        return        ; Στο τέλος μιας ρουτίνας γράφεται πάντα η εντολή return!!!
       END
```

Πρόγραμμα σχεδίασης και προσομοίωσης ηλεκτρονικών κυκλωμάτων Proteus



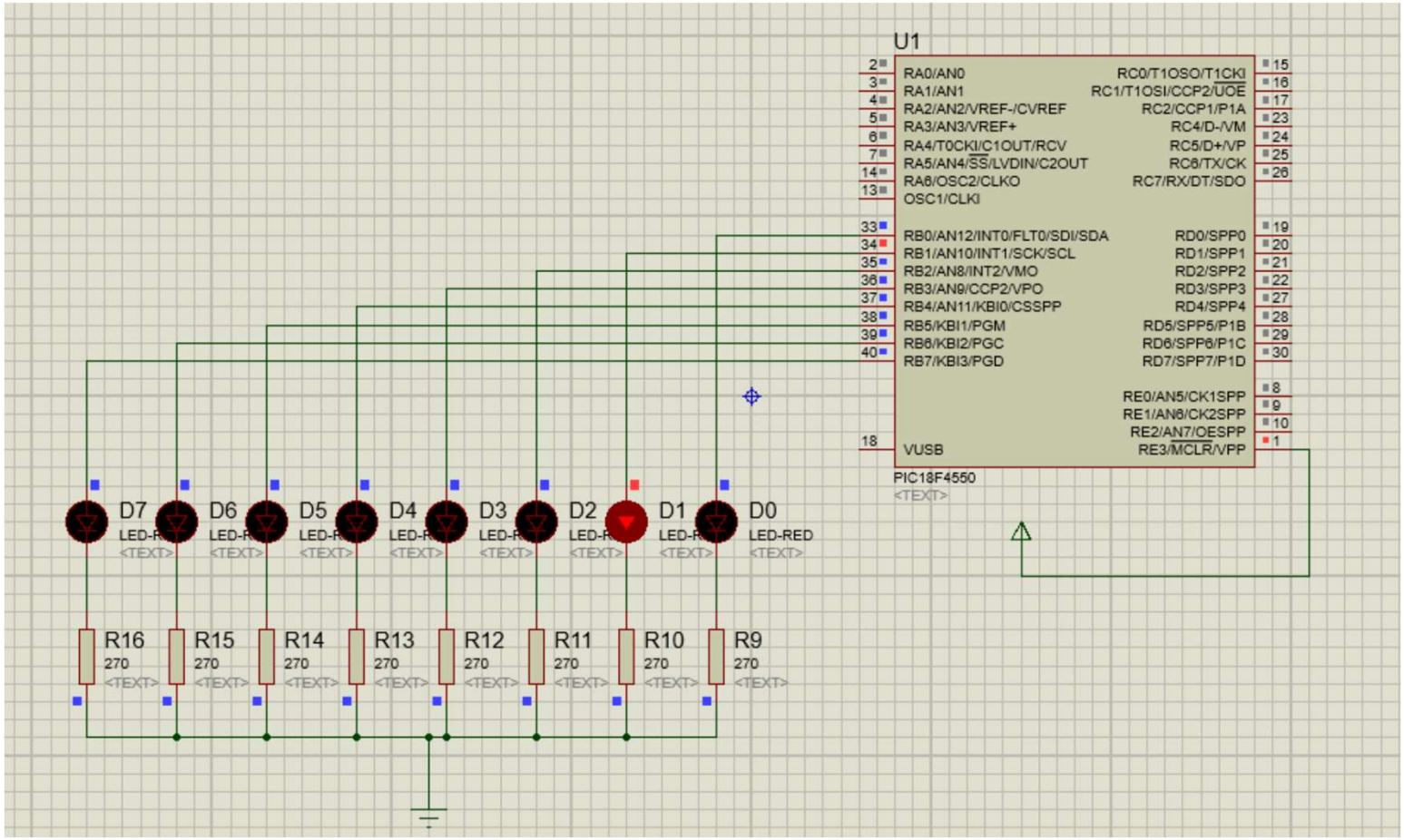
Επιλογή περιοχής για αντιγραφή ή μετακίνηση. Selection mode.

Επιλογή εξαρτήματος . Component mode.
Πατάμε το κουμπί για εισαγωγή εξαρτημάτων και στη συνέχεια το P από το P L DEVICES για να επιλέξουμε το εξάρτημα που θέλουμε να εισάγουμε.
Αφού επιλέξουμε και εισάγουμε ένα εξάρτημα, αυτό εμφανίζεται στη λίστα P L DEVICES
Στη συνέχεια πιάνουμε το εξάρτημα από τη λίστα P L DEVICES και το μεταφέρουμε στην επιφάνεια εργασίας

Εισαγωγή τροφοδοσίας(POWER) και γείωσης (GROUND).
Πιάνουμε το POWER ή το GROUND και το μεταφέρουμε στην επιφάνεια εργασίας.

Άσκηση 5

Να τροποποιηθεί η άσκηση 4 ώστε να εμφανίζει μια τελεία να κινείται από αριστερά προς τα δεξιά

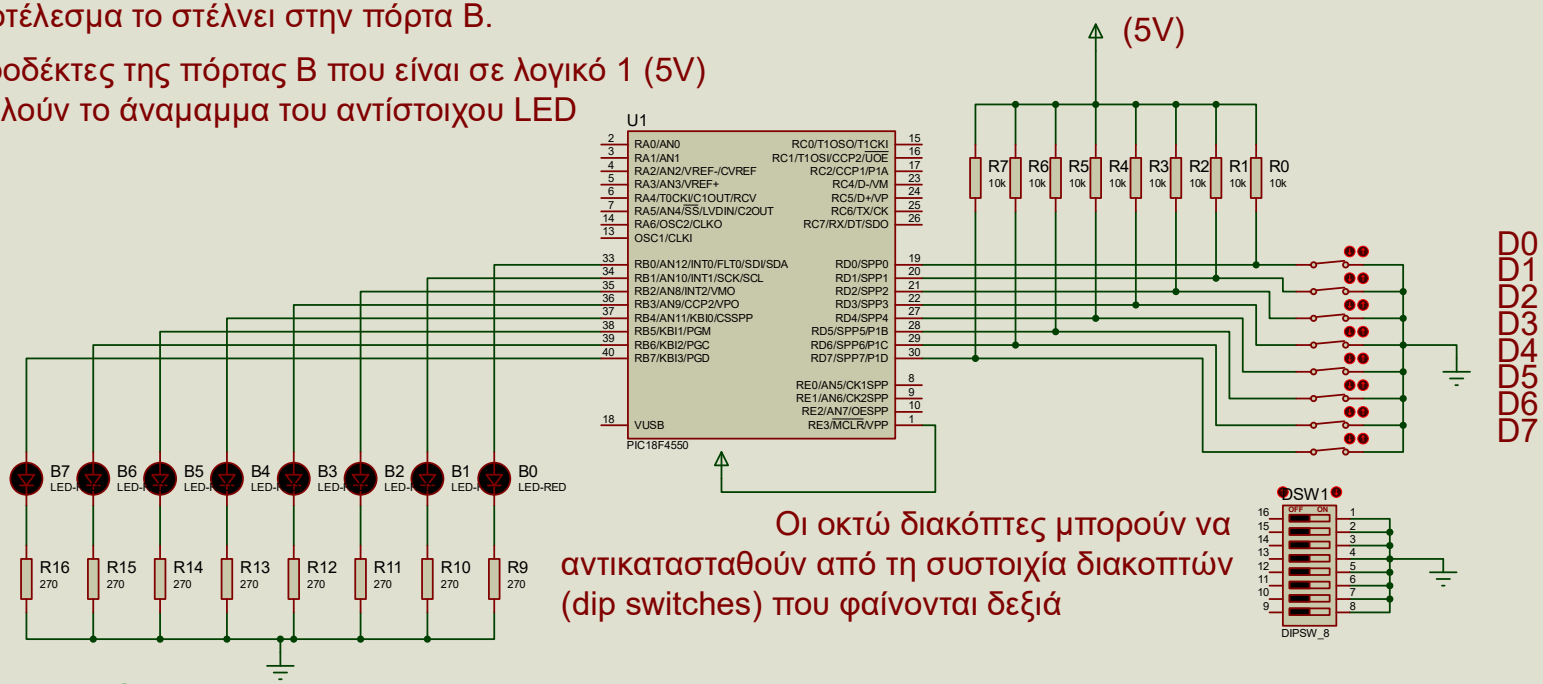


Άσκηση 6(1)

Χρήση της πόρτας D σαν πόρτας εισόδου και της πόρτας B σαν πόρτας εξόδου

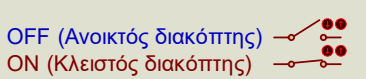
Η πόρτα D χρησιμοποιείται σαν είσοδος και η πόρτα B χρησιμοποιείται σαν έξοδος.
Ο μικροελεγκτής διαβάζει το περιεχόμενο της πόρτας D και το αποτέλεσμα το στέλνει στην πόρτα B.

Οι ακροδέκτες της πόρτας B που είναι σε λογικό 1 (5V) προκαλούν το άναμμα του αντίστοιχου LED



Οι οκτώ διακόπτες μπορούν να αντικατασταθούν από τη συστοιχία διακοπτών (dip switches) που φαίνονται δεξιά

Επεξηγήσεις για το τι σημαίνει ανοικτός και κλειστός διακόπτης



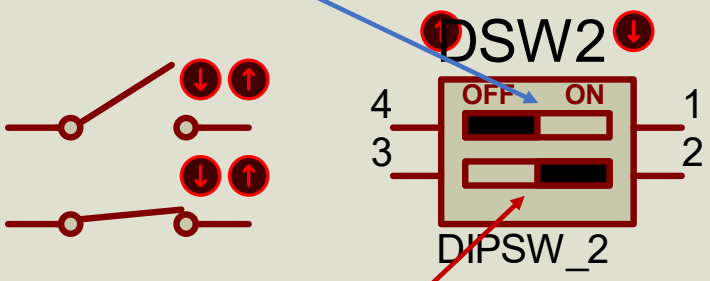
Διακόπτης σε θέση OFF σημαίνει ανοικτή επαφή
Δεν υπάρχει επαφή μεταξύ ακροδεκτών 4 και 1(ο επάνω διακόπτης είναι σε θέση OFF)

Διακόπτης σε θέση ON σημαίνει κλειστή επαφή
Υπάρχει επαφή μεταξύ των ακροδεκτών 3 και 2 (Ο κάτω διακόπτης είναι σε θέση ON)

Επεξηγήσεις για το τι σημαίνει ανοικτός και κλειστός διακόπτης

OFF (το μαύρο δείχνει τη θέση του διακόπτη)

OFF (Ανοικτός διακόπτης)
ON (Κλειστός διακόπτης)



ON (το μαύρο δείχνει τη θέση του διακόπτη)

Διακόπτης σε θέση OFF σημαίνει ανοικτή επαφή

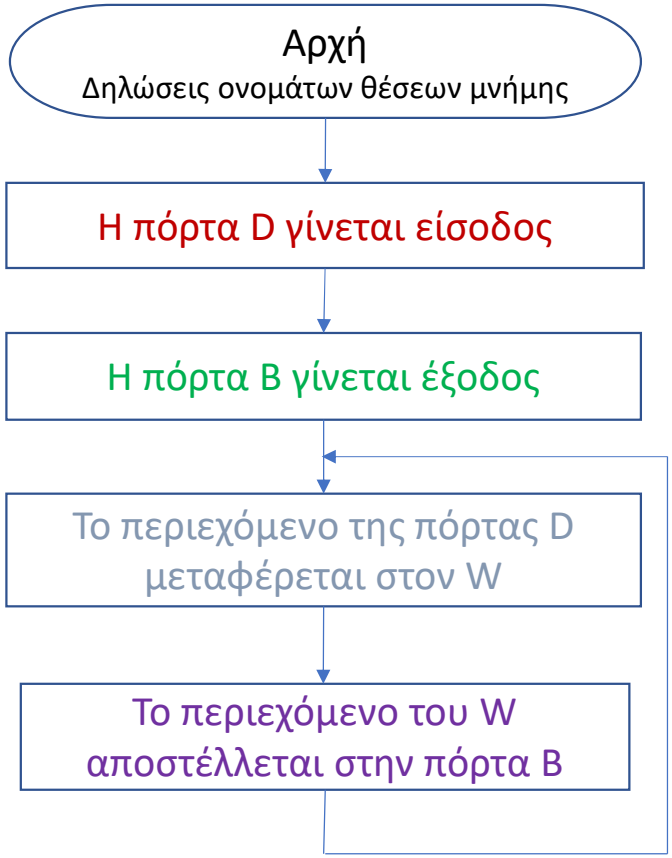
Δεν υπάρχει επαφή μεταξύ ακροδεκτών 4 και 1(ο επάνω διακόπτης είναι σε θέση OFF)

Διακόπτης σε θέση ON σημαίνει κλειστή επαφή

Υπάρχει επαφή μεταξύ των ακροδεκτών 3 και 2 (Ο κάτω διακόπτης είναι σε θέση ON)

Άσκηση 6(2)

Διάγραμμα ροής του προγράμματος και πρόγραμμα



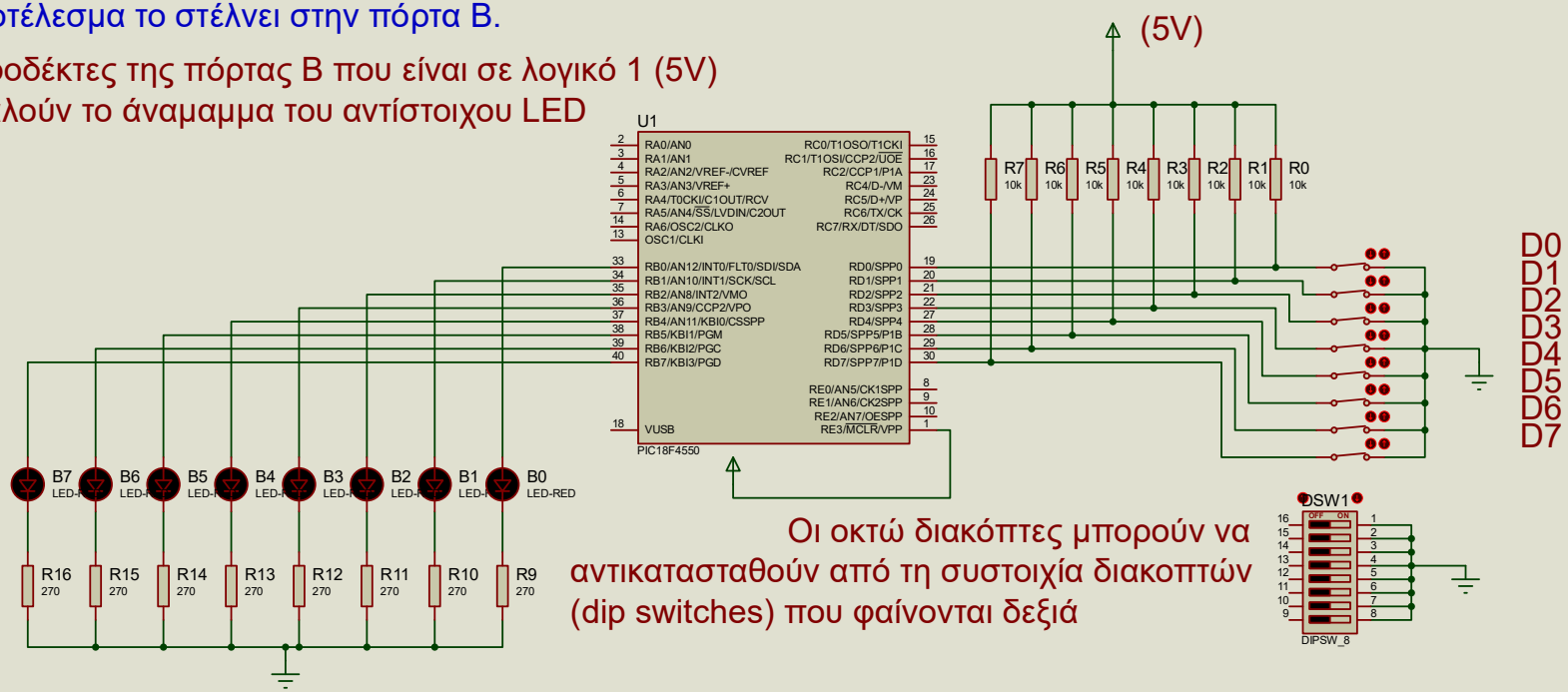

```
        org 0x000
        TRISB  EQU 0xF93
        TRISD  EQU 0xF95
        PORTB EQU 0xF81
        PORTD  EQU 0xF83

        movlw   B'11111111'
        movwf TRISD

        movlw B'00000000'
        movwf TRISB

loop    movf PORTD,0

        movwf PORTB

        goto loop
        END
```

Άσκηση 7(1)

Να γραφεί πρόγραμμα με το οποίο ο μικροελεγκτής διαβάζει το περιεχόμενο της πόρτας D, **προσθέτει την τιμή 1** και το αποτέλεσμα το εμφανίζει στην πόρτα B.

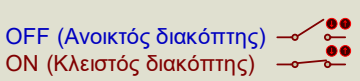
Η πόρτα D χρησιμοποιείται σαν είσοδος και η πόρτα B χρησιμοποιείται σαν έξοδος.
Ο μικροελεγκτής διαβάζει το περιεχόμενο της πόρτας D, προσθέτει την τιμή 1 και το αποτέλεσμα το στέλνει στην πόρτα B.

Οι ακροδέκτες της πόρτας B που είναι σε λογικό 1 (5V) προκαλούν το άναμμα του αντίστοιχου LED



Οι οκτώ διακόπτες μπορούν να αντικατασταθούν από τη συστοιχία διακοπτών (dip switches) που φαίνονται δεξιά

Επεξηγήσεις για το τι σημαίνει ανοικτός και κλειστός διακόπτης

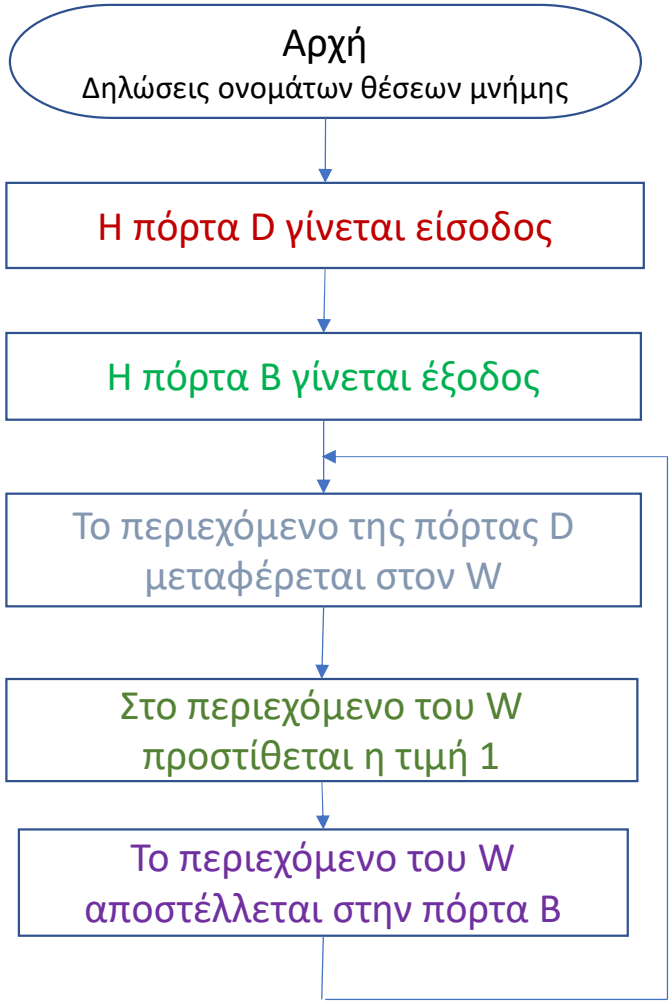


Διακόπτης σε θέση OFF σημαίνει ανοικτή επαφή
Δεν υπάρχει επαφή μεταξύ ακροδεκτών 4 και 1(ο επάνω διακόπτης είναι σε θέση OFF)

Διακόπτης σε θέση ON σημαίνει κλειστή επαφή
Υπάρχει επαφή μεταξύ των ακροδεκτών 3 και 2 (Ο κάτω διακόπτης είναι σε θέση ON)

Άσκηση 7(2)

Διάγραμμα ροής του προγράμματος και πρόγραμμα



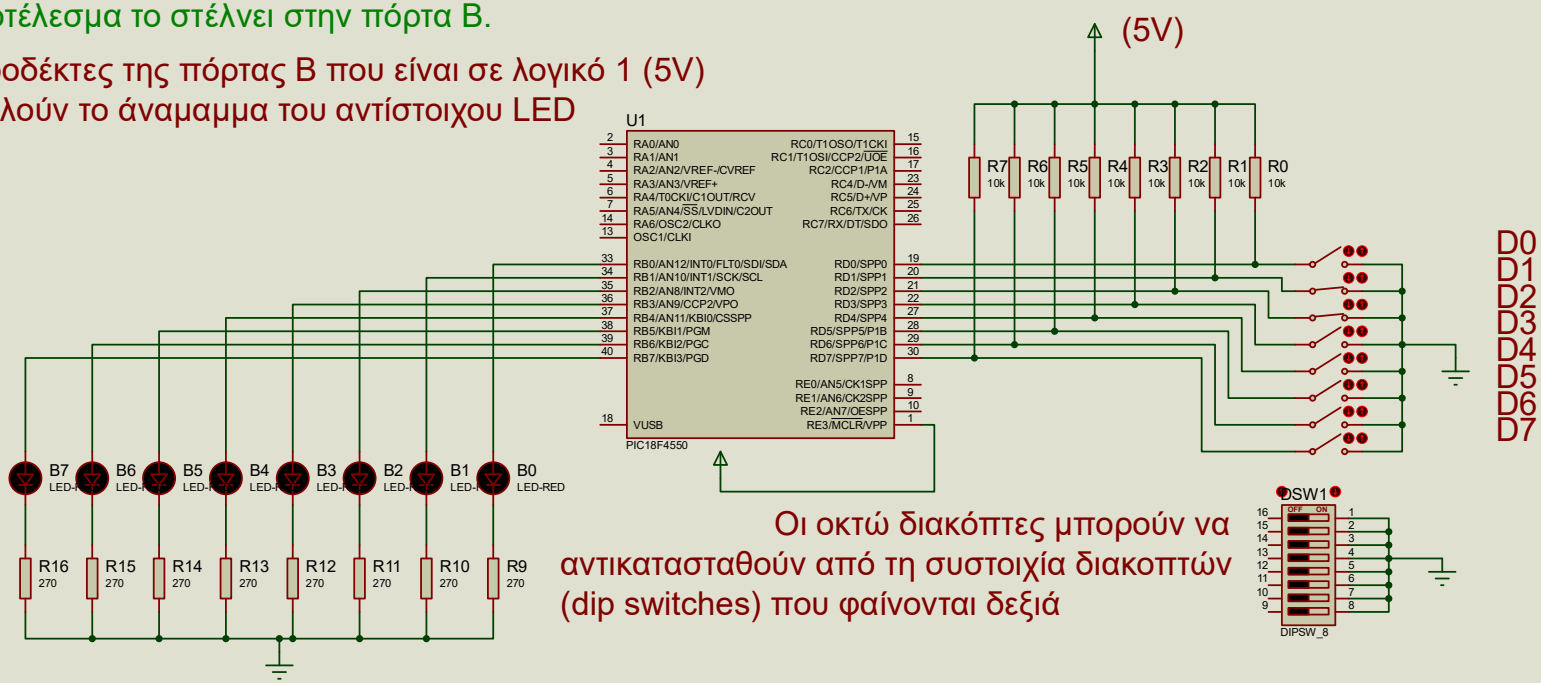

```
        org 0x000
        TRISB  EQU 0xF93
        TRISD  EQU 0xF95
        PORTB EQU 0xF81
        PORTD  EQU 0xF83

        movlw   B'11111111'
        movwf TRISD

        movlw B'00000000'
        movwf TRISB

loop    movf PORTD,0

        addlw 0x01

        movwf PORTB

        goto loop
        END
```

Άσκηση 8(1)

Να γραφεί πρόγραμμα με το οποίο ο μικροελεγκτής διαβάζει το περιεχόμενο της πόρτας D, **προσθέτει την τιμή 1** και το αποτέλεσμα το εμφανίζει στην πόρτα B.

Η πόρτα D χρησιμοποιείται σαν είσοδος και η πόρτα B χρησιμοποιείται σαν έξοδος.
Ο μικροελεγκτής διαβάζει το περιεχόμενο της πόρτας D, προσθέτει την τιμή 2 και το αποτέλεσμα το στέλνει στην πόρτα B.

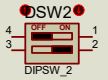
Οι ακροδέκτες της πόρτας B που είναι σε λογικό 1 (5V) προκαλούν το άναμμα του αντίστοιχου LED



Οι οκτώ διακόπτες μπορούν να αντικατασταθούν από τη συστοιχία διακοπτών (dip switches) που φαίνονται δεξιά

Επεξηγήσεις για το τι σημαίνει ανοικτός και κλειστός διακόπτης

OFF (Ανοικτός διακόπτης)
ON (Κλειστός διακόπτης)

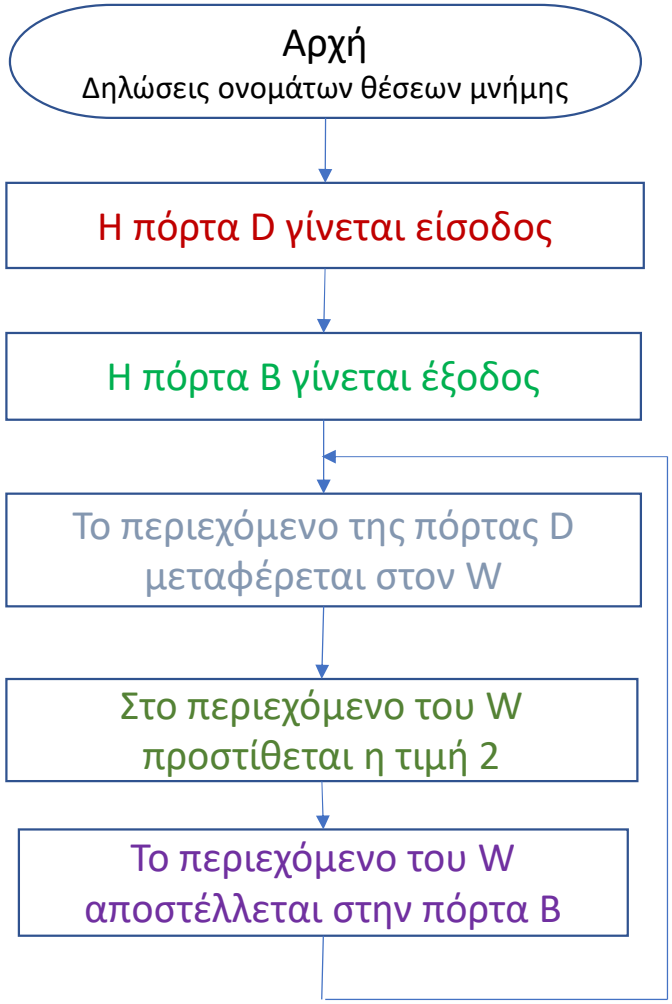


Διακόπτης σε θέση OFF σημαίνει ανοικτή επαφή
Δεν υπάρχει επαφή μεταξύ ακροδεκτών 4 και 1(ο επάνω διακόπτης είναι σε θέση OFF)

Διακόπτης σε θέση ON σημαίνει κλειστή επαφή
Υπάρχει επαφή μεταξύ των ακροδεκτών 3 και 2 (Ο κάτω διακόπτης είναι σε θέση ON)

Άσκηση 8(2)

Διάγραμμα ροής του προγράμματος και πρόγραμμα




```
        org 0x000
        TRISB  EQU 0xF93
        TRISD  EQU 0xF95
        PORTB EQU 0xF81
        PORTD  EQU 0xF83

        movlw   B'11111111'
        movwf TRISD

        movlw B'00000000'
        movwf TRISB

loop    movf PORTD,0

        addlw 0x02

        movwf PORTB

        goto loop
        END
```

Καταχωρητής κατάστασης
STATUS Register

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Ο καταχωρητής κατάστασης είναι μέσα στην Κεντρική Μονάδα Επεξεργασίας (CPU) και κάποια από τα bit του είναι αποτελούν ένδειξη για το αποτέλεσμα μιας πράξης που εκτελέστηκε από την Κεντρική μονάδα επεξεργασίας (CPU).

Αντίγραφο του STATUS REGISTER υπάρχει στη μνήμη δεδομένων στους Special Funcion Registers (SFR) στη διεύθυνση $FD8_h$

Για παράδειγμα το bit C (Carry flag, σημαία κρατουμένου) δείχνει αν κατά το αποτέλεσμα μιας πράξης δημιουργήθηκε κρατούμενο, το bit Z (Zero flag, σημαία μηδενός) δείχνει αν το αποτέλεσμα μιας πράξης ήταν μηδέν και το bit N δείχνει αν στο αποτέλεσμα μιας πράξης το bit 7 είναι 1 (Negative flag, σημαία αρνητικού)

Καταχωρητής κατάστασης STATUS Register Zero flag, Z (1)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Σημαία του μηδενός (Zero flag)

Αν το αποτέλεσμα μιας πράξης είναι 0 τότε η σημαία του μηδενός γίνεται 1.

Αν το αποτέλεσμα μιας πράξης δεν είναι 0 τότε η σημαία του μηδενός γίνεται 0

Παράδειγμα 1:

Μετά την εκτέλεση των παρακάτω εντολών ποια θα είναι η τιμή του Z;

```
movlw B'11111111' ; Μετάφερε στον w την τιμή 1111 1111b
addlw 0x01        ; Πρόσθεσε στο περιεχόμενο του W την τιμή 1
                  ; Το αποτέλεσμα αποθηκεύεται στον W
```

Μετά την εκτέλεση των παραπάνω εντολών ο καταχωρητής εργασίας **W** θα πάρει την τιμή **0** και επομένως θα γίνει **Z=1**

Καταχωρητής κατάστασης STATUS Register Zero flag, Z (2)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Σημαία του μηδενός (Zero flag)

Αν το αποτέλεσμα μιας πράξης είναι 0 τότε η σημαία του μηδενός γίνεται 1

Παράδειγμα 1:

Μετά την εκτέλεση των παρακάτω εντολών ποια θα είναι η τιμή του C;

```
movlw 0x0F          ; Φόρτωσε στον W την τιμή 0xOF
movwf 0X60          ; Μετάφερε το περιεχόμενο του W στη θέση μνήμης 060h
movlw B'00001111'   ; Φόρτωσε στον W την τιμή 00001111h
subwf 0x60          ; Αφαίρεσε τον W από το περιεχόμενο της θέσης μνήμης 060h
```

Μετά την εκτέλεση των παραπάνω εντολών θα γίνει **Z=1** διότι όπως προκύπτει από το manual του μικροελεγκτή η εντολή **sublwf** επηρρεάζει την σημαία του μηδενός (Zero flag) και το αποτέλεσμα της αφαίρεσης είναι **μηδέν**.

Καταχωρητής κατάστασης
STATUS Register Zero flag, Z (3)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Υπάρχουν εντολές με τις οποίες μπορούμε να επέμβουμε στη σημαία του μηδενός (Zero Flag) και να την κάνουμε 1 ή 0.

bcf STATUS, Z ; Clear bit Z at file Status Register
; Μηδένισε τη σημαία του μηδενός στον καταχωρητή κατάστασης

Υπάρχει εντολή με την οποία μπορούμε να κάνουμε 1 τη σημαία του μηδενός:
bsf STATUS,Z ; Set bit Z at file Status Register
; Κάνε 1 τη σημαία του μηδενός στον καταχωρητή κατάστασης

set σημαίνει στη γλώσσα των ψηφιακών κάνω 1
clear σημαίνει στη γλώσσα των ψηφιακών κάνω 0

Εντολές που σχετίζονται με τη σημαία του μηδενός (Zero Flag) (4)

Εντολές διακλάδωσης με έλεγχο της σημαίας του μηδενός (Zero flag)

BZ label Branch if Zero (Branch if Zero flag =1, κάνε διακλάδωση αν Zero flag=1)

BNZ label Branch if no Zero Flag (Branch if Zero flag=0, κάνε διακλάδωση αν Zero flag=0)

Εντολές με τις οποίες κάνουμε 1 (set) ή κάνουμε 0 (clear) τη σημαία του μηδενός

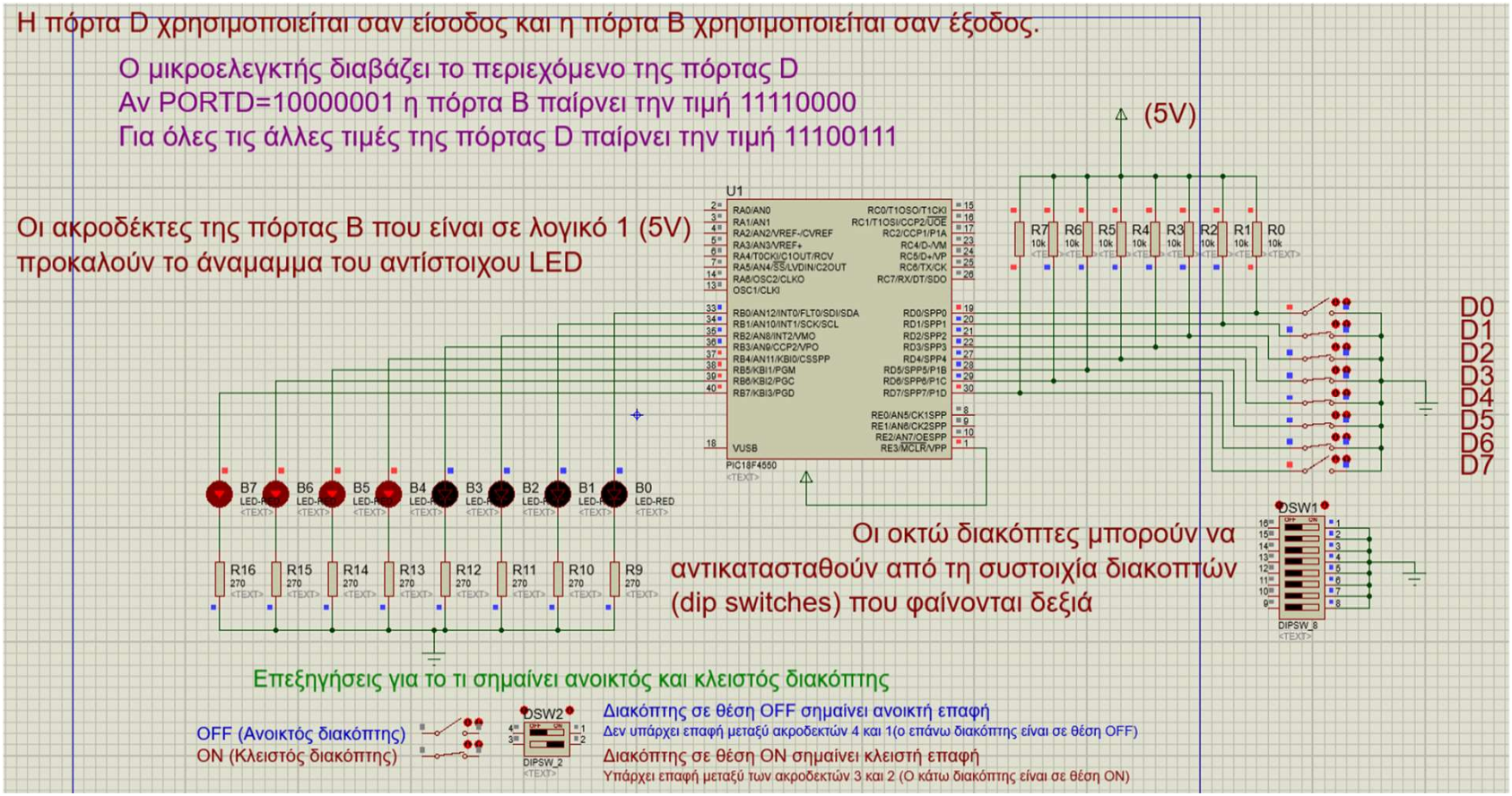
```
STATUS EQU 0xFD8   ; Αντιστοίχιση της λέξης STATUS με τη θέση μνήμης δεδομένων 0xFD8 (Special Function Registers)
Z      EQU D'2'    ; Στο γράμμα Z αποδίδεται η τιμή 2. Το Zero flag είναι το bit 2 στον καταχωρητή κατάστασης STATUS

bcf STATUS, Z  ;Clear Zero flag at STATUS , κάνε 0 τη σημαία του μηδενός στον καταχωρητή STATUS
bsf STATUS, Z  ;Set Zero flag at STATUS, κάνε 1 τη σημαία του μηδενός στον καταχωρητή STATUS
```

Αντίγραφο του STATUS REGISTER υπάρχει στη μνήμη δεδομένων στους Special Funcion Registers (SFR) στη διεύθυνση $FD8_h$

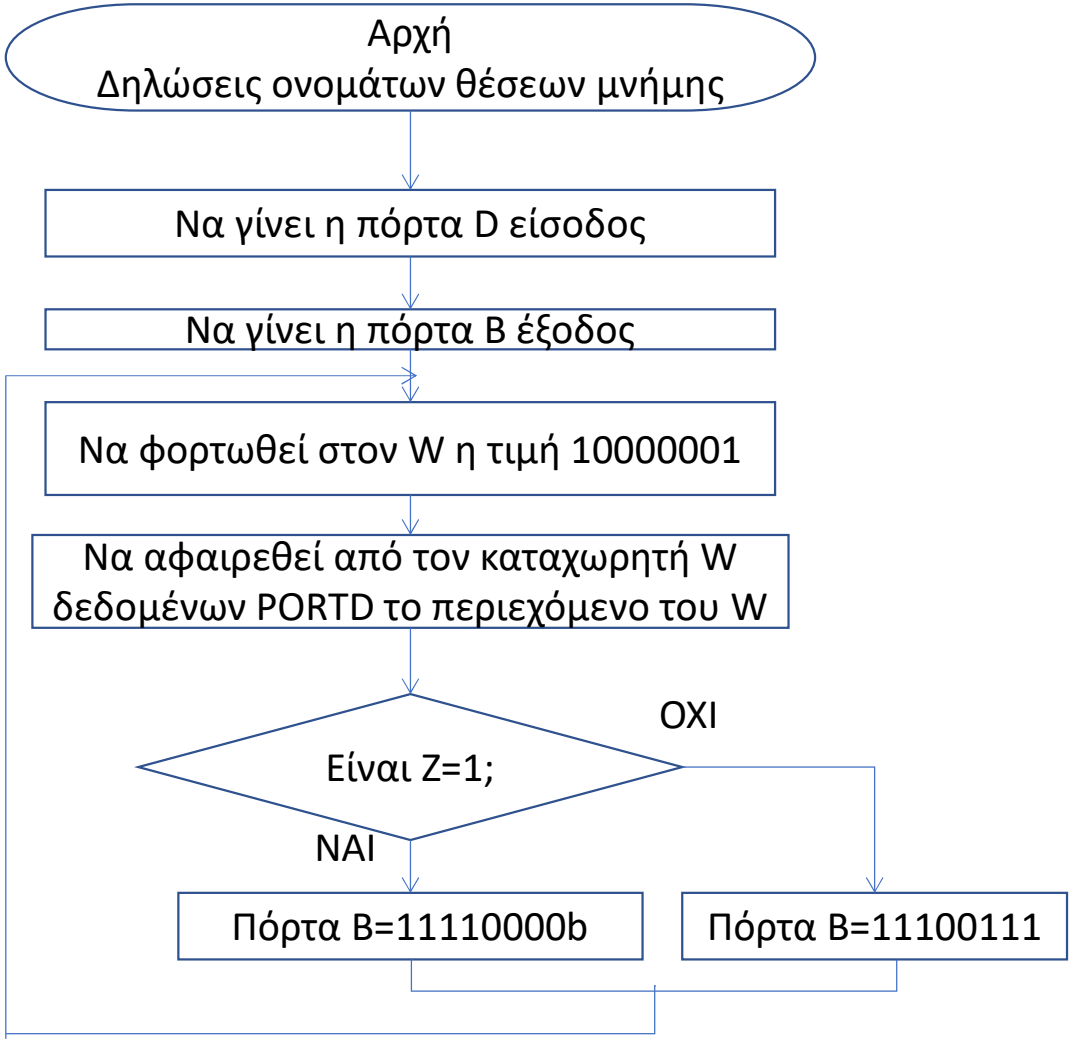
Άσκηση 9(1)

Παράδειγμα ελέγχου της ροής προγράμματος με τη βοήθεια της σημαίας του μηδενός (zero flag)(1)

Να γραφεί πρόγραμμα με το οποίο διαβάζεται συνεχώς το περιεχόμενο της πόρτας D.
Αν η πόρτα D έχει την τιμή **10000001b** να εμφανίζεται στην πόρτα B η τιμή **11110000b.**
Σε όλες τις άλλες περιπτώσεις να εμφανίζεται στην πόρτα B η τιμή **11100111**.



Παράδειγμα ελέγχου της ροής προγράμματος με τη βοήθεια της σημαίας του μηδενός (zero flag)(2) διάγραμμα ροής. ‘Ασκηση 9(2)



Παράδειγμα ελέγχου της ροής προγράμματος με τη βοήθεια της σημαίας του μηδενός (zero flag) Άσκηση 9(3) **Πρόγραμμα**

```
        org 0x000          ; Οδηγία για τοποθέτηση του προγράμματος θέση 000h στη μνήμη προγράμματος
TRISB   EQU 0xF93          ; Δήλωση των θέσεων των καταχωρητών κατεύθυνσης
TRISD   EQU 0xF95          ; και των καταχωρητών δεδομένων στη μνήμ
PORTB   EQU 0xF81          ; δεδομένων του μικροελεγκτή.
PORTD   EQU 0xF83          ;
STATUS  EQU 0xFD8          ; Θέση του STATUS register στους Special Function
                           ; Registers στη μνήμη δεδομένων του Μικροελεγκτή
Z       EQU D'2'           ; Θέση του Zero Flag στον STATUS Register. Είναι το bit 2

        movlw B'11111111'  ; Η πόρτα D γίνεται είσοδος
        movwf  TRISD       ; Δίνεται η τιμή 11111111b στον καταχωρητή κατεύθυνσης D

        movlw B'00000000'  ; Η πόρτα B γίνεται έξοδος
        movwf  TRISB       ; Δίνεται η τιμή 00000000b στον καταχωρητή κατεύθυνσης B

loop    movlw B'10000001'  ; Φορτώνεται η τιμή 10000001b στον καταχωρητή W
        subwf PORTD        ; Αφαιρείτα από τον καταχωρητή δεδομένων της
                           ; πόρτας D το περιεχόμενο του καταχωρητή W
        bz   equal         ; Αν γίνει 1 η σημαία του μηδενός (ισότητα) , τότε γίνεται
                           ;άλμα του προγράμματος προς την ετικέτα equal
                           ; (Branch on Zero flag=1 to label equal)
        movlw B'11100111'  ; Δίνεται η τιμή 11100111b στον καταχωρητή
        movwf PORTB        ; δεδομένων της πόρτας B
        goto loop          ; Άλμα προς την ετικέτα loop για νέο έλεγχο

equal   movlw B'11110000'  ;Δίνεται η τιμή 11110000b στον καταχωρητή
        movwf PORTB        ; Δεδομένων της πόρτας B
        goto   loop        ; Άλμα προς την ετικέτα loop για νέο έλεγχο
        END                ; Οδηγία προς τον Assembler για το τέλος του προγράμματος
```

Εντολές που επηρεάζουν τη σημαία του μηδενός

Κάποιες από τις εντολές που επηρεάζουν τη σημαία του μηδενός (zero flag)

ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	

Στο manual του μικροελεγκτή PIC18F4550 (PIC18F4550 datasheet, σελίδα 312) υπάρχει συνοπτικός πίνακας με όλες τις εντολές. Στον πίνακα αυτό εμφανίζονται τα bit στον καταχωρητή STATUS τα οποία επηρεάζονται από την κάθε μια εντολή.

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Καταχωρητής κατάστασης STATUS Register Carry flag, C (1)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Σημαία του κρατουμένου (Carry flag)

Αν στο αποτέλεσμα μιας πράξης υπάρχει κρατούμενο τότε η σημαία του κρατουμένου (Carry flag) γίνεται 1.

Αν στο αποτέλεσμα μιας πράξης δεν υπάρχει κρατούμενο τότε η σημαία του κρατουμένου (Carry flag) γίνεται 0.

Παράδειγμα 1: Μετά την εκτέλεση των παρακάτω εντολών ποια θα είναι η τιμή του C;

```
movlw B'11111111' ; Μετάφερε στον w την τιμή 1111 1111b
addlw 0x02        ; Πρόσθεσε στο περιεχόμενο του W την τιμή 2
                  ; Το αποτέλεσμα αποθηκεύεται στον W
```

Μετά την εκτέλεση των παραπάνω εντολών ο καταχωρητής εργασίας **W** θα πάρει την τιμή **0000 0001** και επομένως θα γίνει **C=1**

Καταχωρητής κατάστασης
STATUS Register Carry flag, C (2)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Υπάρχουν εντολές με τις οποίες μπορούμε να επέμβουμε στη σημαία του κρατουμένου (Carry Flag) και να την κάνουμε 1 ή 0.

bcf STATUS, C ; Clear bit C at file Status Register
; Μηδένισε τη σημαία του κρατουμένου στον καταχωρητή κατάστασης

bsf STATUS, C ; Set bit C at file Status Register
; Κάνε 1 τη σημαία του κρατουμένου στον καταχωρητή κατάστασης

set σημαίνει στη γλώσσα των ψηφιακών κάνω 1

clear σημαίνει στη γλώσσα των ψηφιακών κάνω 0

Εντολές που σχετίζονται με τη σημαία του κρατουμένου (Carry Flag) (3)

Εντολές διακλάδωσης με έλεγχο της σημαίας του κρατουμένου (Carry flag)

BC label Branch if Carry (Branch if Carry flag =1, κάνε διακλάδωση αν Carry flag=1)

BNC label Branch if no Carry flag (Branch if Carry flag=0, κάνε διακλάδωση αν Carry flag=0)

Εντολές με τις οποίες κάνουμε 1 (set) ή κάνουμε 0 (clear) τη σημαία του κρατουμένου (Carry flag)

```
STATUS EQU 0xFD8   ; Αντιστοίχιση της λέξης STATUS με τη θέση μνήμης δεδομένων 0xFD8 (Special Function Registers)
C      EQU D'0'    ; Στο γράμμα C αποδίδεται η τιμή 0. Το Carry flag είναι το bit 0 στον καταχωρητή κατάστασης STATUS
bcf STATUS, C  ; Clear Carry flag at STATUS , κάνε 0 τη σημαία του κρατουμένου στον καταχωρητή STATUS
bsf STATUS, C  ; Set Carry flag at STATUS, κάνε 1 τη σημαία του μηδενός στον καταχωρητή STATUS
```

Αντίγραφο του STATUS REGISTER υπάρχει στη μνήμη δεδομένων στους Special Funcion Registers (SFR) στη διεύθυνση $FD8_h$

Καταχωρητής κατάστασης STATUS Register Negative flag, N (1)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Σημαία του αρνητικού (negative flag)

Αν στο αποτέλεσμα μιας πράξης το bit 7 είναι 1 τότε η σημαία του αρνητικού (Negative flag) γίνεται 1. (Negative flag is set)

Αν στο αποτέλεσμα μιας πράξης το bit 7 είναι 0 τότε η σημαία του αρνητικού (Negative flag) γίνεται 0. (Negative flag is cleared)

Παράδειγμα 1: Μετά την εκτέλεση των παρακάτω εντολών ποια θα είναι η τιμή του N;

```
movlw B'01111111' ; Μετάφερε στον w την τιμή 01111111b
addlw 0x02        ; Πρόσθεσε στο περιεχόμενο του W την τιμή 2
                  ; Το αποτέλεσμα αποθηκεύεται στον W
```

Μετά την εκτέλεση των παραπάνω εντολών ο καταχωρητής εργασίας **W** θα πάρει την τιμή **1000 0001** και επομένως θα γίνει **N=1**

Καταχωρητής κατάστασης
STATUS Register Negative flag, N (2)

REGISTER 5-2: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(2)
bit 7							bit 0

Υπάρχουν εντολές με τις οποίες μπορούμε να επέμβουμε στη σημαία του αρνητικού (Negative Flag) και να την κάνουμε 1 ή 0.

bcf STATUS, N ; Clear bit N at file Status Register
; Μηδένισε τη σημαία του αρνητικού στον καταχωρητή κατάστασης

bsf STATUS, N ; Set bit N at file Status Register
; Κάνε 1 τη σημαία του κρατουμένου στον καταχωρητή κατάστασης

set σημαίνει στη γλώσσα των ψηφιακών κάνω 1

clear σημαίνει στη γλώσσα των ψηφιακών κάνω 0

Εντολές που σχετίζονται με τη σημαία του αρνητικού (Negative Flag) (3)

Εντολές διακλάδωσης με έλεγχο της σημαίας του κρατουμένου (Carry flag)

BN label Branch if Negative (Branch if Negative Flag=1, κάνε διακλάδωση αν Negative flag=1)

BNN label Branch if no Negative (Branch if Negative flag=0, κάνε διακλάδωση αν Negative flag=0)

Εντολές με τις οποίες κάνουμε 1 (set) ή κάνουμε 0 (clear) τη σημαία του αρνητικού (Negative flag)

```
STATUS EQU 0xFD8   ; Αντιστοίχιση της λέξης STATUS με τη θέση μνήμης δεδομένων 0xFD8  (Special Function Registers)
N      EQU D'4'    ; Στο γράμμα C αποδίδεται η τιμή 4. Το Negative flag είναι το bit 0 στον καταχωρητή κατάστασης
                   ; STATUS

bcf STATUS, C  ; Clear Carry flag at STATUS , κάνε 0 τη σημαία του κρατουμένου στον καταχωρητή STATUS
bsf STATUS, C  ; Set Carry flag at STATUS, κάνε 1 τη σημαία του μηδενός στον καταχωρητή STATUS
```

Αντίγραφο του STATUS REGISTER υπάρχει στη μνήμη δεδομένων στους Special Funcion Registers (SFR) στη διεύθυνση $FD8_h$

Πίνακας των εντολών στο Manual PIC18F4550 datasheet

Σελίδα 312 του manual του PIC18F4550 πίνακας με όλες τις εντολές όπου φαίνεται το εάν επηρεάζουν τις σημαίες.

Σελίδα 315 του manual του PIC18F4550, υπάρχουν αναλυτικά όλες οι εντολές όπου φαίνονται λεπτομέρειες για τη σύνταξη τους, πως επηρεάζουν τις σημαίες και παραδείγματα χρήσης τους.

Σελίδα 345
Παράδειγμα εντολής, σημασία των παραμέτρων d και a
SUBWF f {,d {,a}}

Σελίδα 317
Παράδειγμα εντολής, σημασία των παραμέτρων d και a
ANDWF f {,d {,a}}

Σελίδα 315
Παράδειγμα εντολής, σημασία των παραμέτρων d και a
ADDWF f {,d {,a}}

Άσκηση 11(1)

Σύγκριση της τιμής της πόρτας D με την τιμή 0x0F και εμφάνιση στην πόρτα B της τιμής 00000000b ή της τιμής 11111111b

Αν PORTD < 0x0F τότε PORTB = 00000000b

Αν PORTD >=0x0F τότε PORTB =11111111b

(Γίνεται χρήση της σημαίας του κρατουμένου)

Η πόρτα D χρησιμοποιείται σαν είσοδος και η πόρτα B χρησιμοποιείται σαν έξοδος.

Ο μικροελεγκτής διαβάζει το περιεχόμενο της πόρτας D και το συγκρίνει με την τιμή 0χ0F (00001111b)
Αν PORTD < 0x0F εμφανίζει στην πόρτα B την τιμή 0XFF (00000000b)
Αν PORTD >= 0x0F εμφανίζει στην πόρτα B την τιμή 0X00 (11111111b)

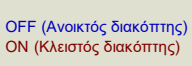
Οι ακροδέκτες της πόρτας B που είναι σε λογικό 1 (5V) προκαλούν το άναμμα του αντίστοιχου LED

Οι οκτώ διακόπτες μπορούν να αντικατασταθούν από τη συστοιχία διακοπτών (dip switches) που φαίνονται δεξιά

Το πρόγραμμα έχει διαμορφωθεί ώστε η σημαία κρατούμενου (Carry Flag), να ανάβει το αντίστοιχο LED

(PORTD-W) --> W

Επεξηγήσεις για το τι σημαίνει ανοικτός και κλειστός διακόπτης



Διακόπτης σε θέση OFF σημαίνει ανοικτή επαφή
Δεν υπάρχει επαφή μεταξύ ακροδεκτών 4 και 1(ο επάνω διακόπτης είναι σε θέση OFF)

Διακόπτης σε θέση ON σημαίνει κλειστή επαφή
Υπάρχει επαφή μεταξύ των ακροδεκτών 3 και 2 (Ο κάτω διακόπτης είναι σε θέση ON)

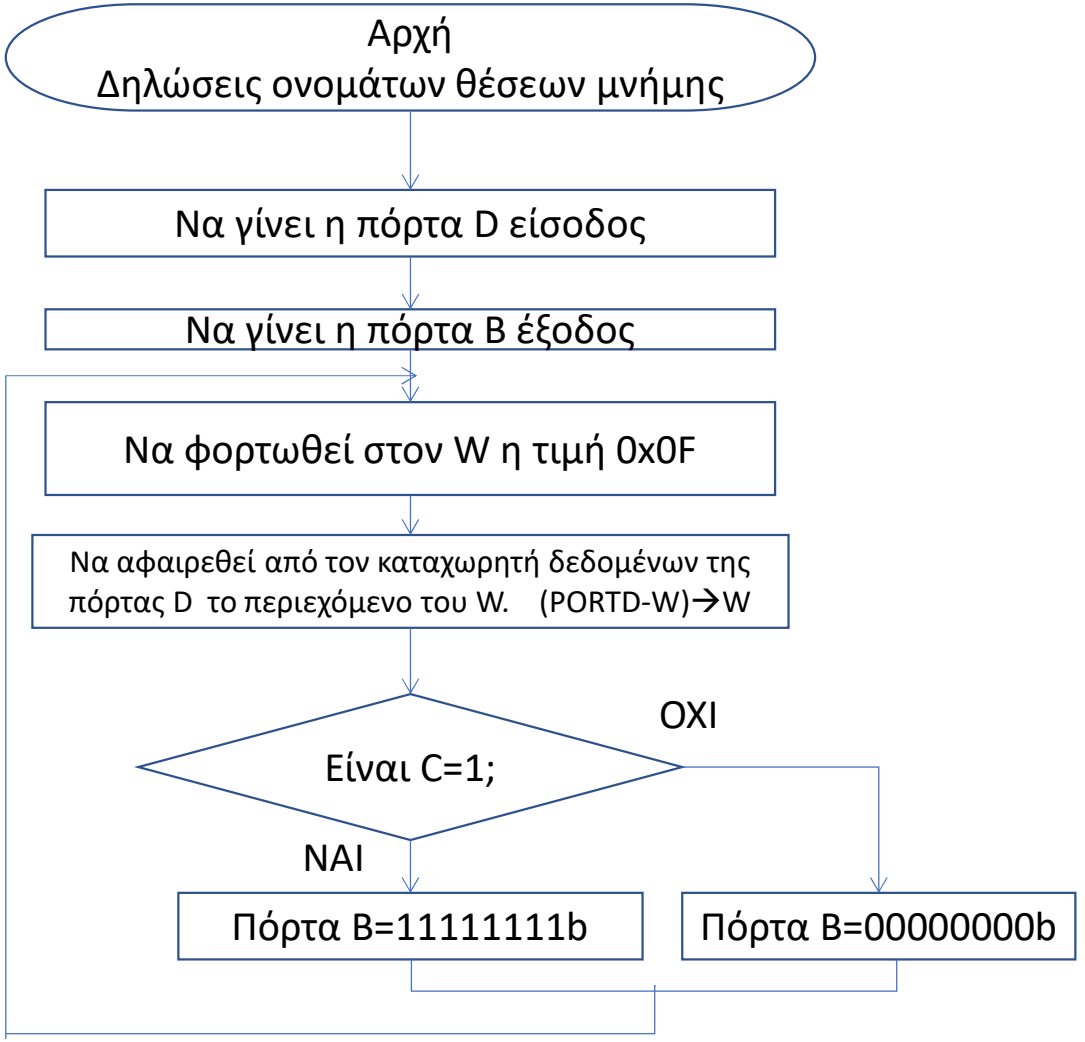
Σ' αυτή την άσκηση ο έλεγχος της τιμής της πόρτας D γίνεται με τη χρήση της σημαίας του κρατουμένου μετά την πράξη PORTD-W. Το W έχει την τιμή 0x0F

Εάν PORTD-W >=0 υπάρχει κρατούμενο (C=1) και αποδίδεται στην πόρτα B η τιμή 11111111b

Εάν PORTD-W<0 δεν υπάρχει κρατούμενο (C=0) και αποδίδεται στην πόρτα B η τιμή 00000000b

Άσκηση 11(2)

Παράδειγμα ελέγχου της ροής προγράμματος με τη βοήθεια της σημαίας του κρατουμένου (zero flag)(2) διάγραμμα ροής.



Άσκηση 11(3)

Σύγκριση δύο τιμών με χρήση της σημαίας του κρατουμένου (πρόγραμμα, 1ο μέρος)

```
            org 0x000   ; Δήλωση της αρχής του προγράμματος στη μνήμη προγράμματος

TRISB       EQU 0xF93   ; Δηλώσεις των καταχωρητών κατεύθυνσης των παράλληλων πορτών
TRISC       EQU 0xF94
TRISD       EQU 0xF95

PORTB       EQU 0xF81   ; Δηλώσεις των καταχωρητών δεδομένων των παράλληλων πορτών
PORTC       EQU 0xF82
PORTD       EQU 0xF83

STATUS      EQU 0xFD8   ; Δήλωση της θέσης του καταχωρητή κατάστασης στη μνήμη δεδομένων(Special Function Register)

            movlw   B'11111111' ; Η παράλληλη πόρτα D γίνεται είσοδος
            movwf TRISD         ; τοποθετώντας 1 σε όλα τα bit του αντίστοιχου καταχωρητή κατεύθυνσης

            movlw B'00000000'   ; Η παράλληλη πόρτα B γίνεται έξοδος
            movwf TRISB         ; τοποθετώντας 0 σε όλα τα bit του αντίστοιχου καταχωρητή κατεύθυνσης

            movlw B'00000000'   ; Η παράλληλη πόρτα C γίνεται έξοδος
            movwf TRISC         ; τοποθετώντας 0 σε όλα τα bit του αντίστοιχου καταχωρητή κατεύθυνσης
```

Άσκηση 11(4)

Σύγκριση δύο τιμών με χρήση της σημαίας του κρατουμένου (πρόγραμμα, 2ο μέρος)

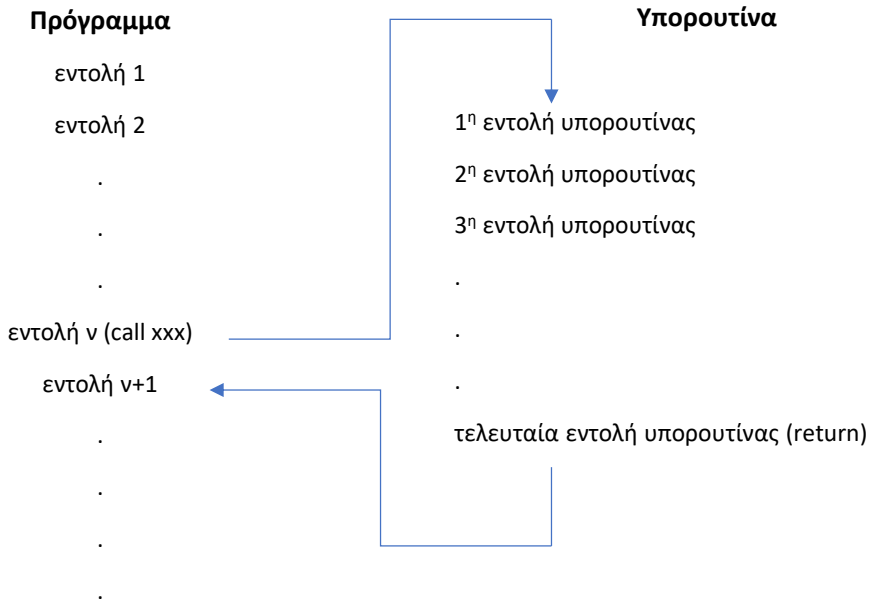
```
loop        movlw 0x0F      ; Φόρτωσε στον W την τιμή 0x0F
            subwf PORTD     ; Αφαίρεσε το περιεχόμενο του W από το περιεχόμενο του καταχωρητή δεδομένων της πόρτας D
                            ; Το αποτέλεσμα αποθηκεύεται στον W. Προσοχή! Ο W αφαιρείται από το PORTD. (PORTD-W)-->W
                            ; Ο καταχωρητής W έχει ήδη την τιμή 0x0F.

                            ; Προσοχή!!!!  μετά την αφαίρεση είναι C=1 αν το αποτέλεσμα είναι >=0

            BC  CarryFlag   ; Αν υπάρχει κρατούμενο(Branch if Carry) πήγαινε στην ετικέτα CarryFlag
                            ; Αν δεν υπάρχει κρατούμενο εκτελούνται οι παρακάτω εντολές

            movlw 0x00      ; Φόρτωσε στον W την τιμή 0xFF
            movwf PORTB     ; Μετάφερε το περιεχόμενο του W στον καταχωρητή δεδομένων της πόρτας B
                            ; Η πόρτα B παίρνει την τιμή 11111111

            movwf STATUS,0  ;Μετάφερε το περιεχόμενο του καταχωρητή STATUS στον W
            movwf PORTC     ;Μετάφερε το περιεχόμενο του W στον καταχωρητή δεδομένων της πόρτας C
            goto loop       ; Πήγοαινε στην ετικέττα loop

CarryFlag   movlw 0xFF      ; Φόρτωσε στον w την τιμή 0x00 (=00000000b)
            movwf PORTB     ; Στείλε το περιεχόμενο του W στον καταχωρητή δεδομένων της πόρτας
                            ; Η πόρτα B παίρνει την τιμή 00000000

            movwf STATUS,0  ;Μετάφερε το περιεχόμενο του καταχωρητή STATUS στον W
            movwf PORTC     ;Μετάφερε το περιεχόμενο του W στον καταχωρητή δεδομένων της πόρτας C
            goto loop       ; Πήγοαινε στην ετικέττα loop

            END
```

Υπορουτίνες (Subroutines)(1)

Τι εννοούμε με τον όρο Υπορουτίνα (Subroutine)

Υπορουτίνα είναι ένα κομμάτι προγράμματος του οποίο καλείται να εκτελεστεί με μια εντολή κλήσης (call xxxx ;όπου xxx το όνομα της υπορουτίνας).

Μετά την εκτέλεση της υπορουτίνας το πρόγραμμα συνεχίζεται από την εντολή που ακολουθεί την εντολή κλήσης της υπορουτίνας.



Προσοχή!

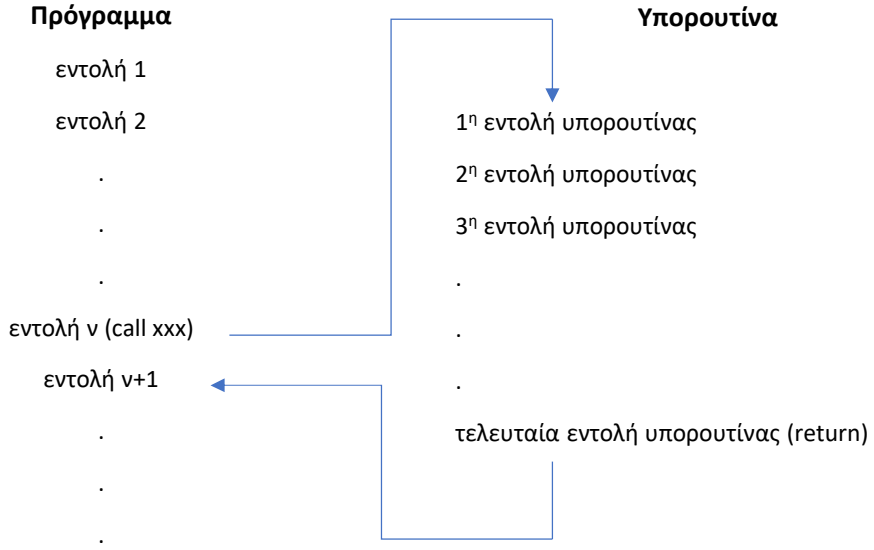
Η υπορουτίνα είναι ένα ανεξάρτητο κομμάτι προγράμματος που καλείται να εκτελεστεί με μια εντολή call είτε μέσα από το κύριο πρόγραμμα είτε μέσα από μια άλλη υπορουτίνα.

Στην γλώσσα Assembly μετά την εντολή call ακολουθεί η διεύθυνση στη μνήμη προγράμματος στην οποία είναι γραμμένη η υπορουτίνα.

Αντί για την διεύθυνση στη μνήμη προγράμματος στην οποία είναι γραμμένη η υπορουτίνα μπορούμε να βάλουμε και μια ετικέτα η οποία αντιστοιχεί στην διεύθυνση της μνήμης προγράμματος στην οποία είναι γραμμένη η υπορουτίνα

Π.χ. call 0x0800 ή call kostas

Υπορουτίνες (Subroutines) (2)

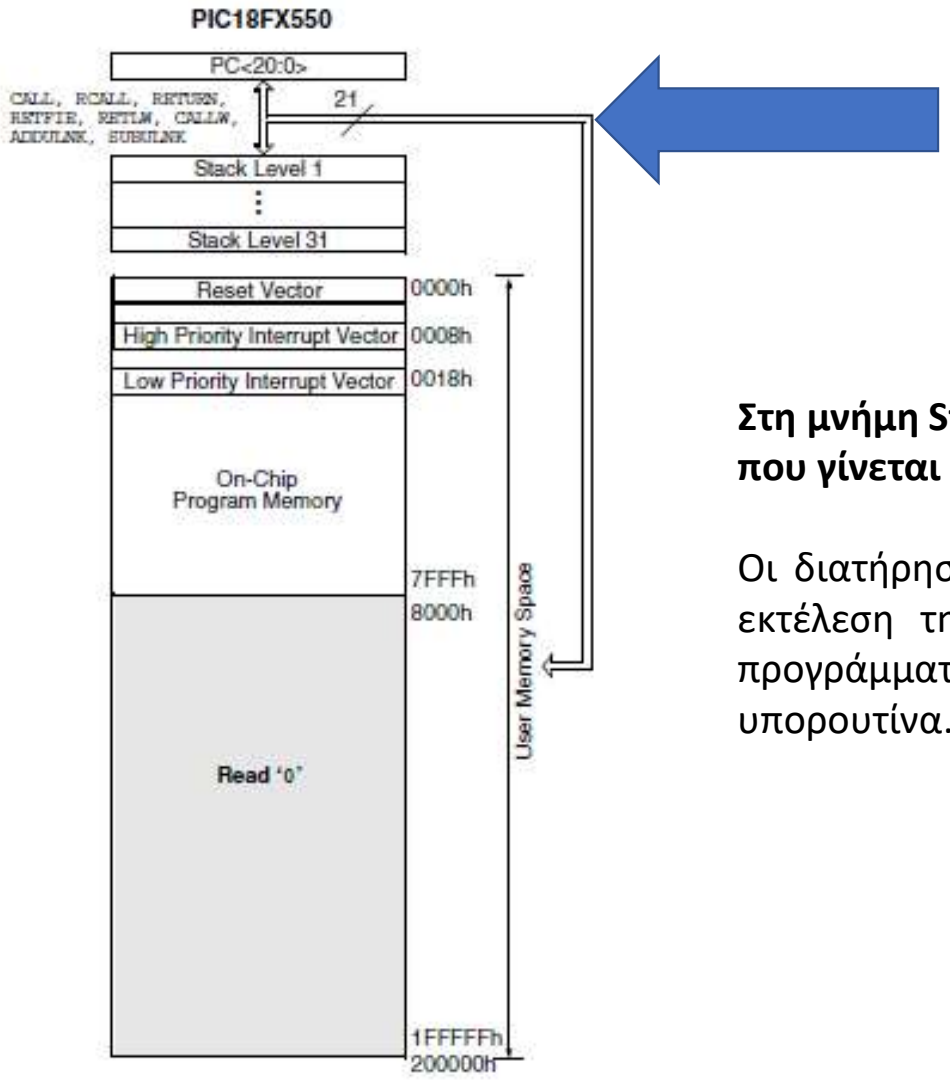


Η εντολή **call xxx** είναι εντολή 32 bit, και καταλαμβάνει 4 byte.
Κατά την εκτέλεση αυτής της εντολής η διεύθυνση επιστροφής στο πρόγραμμα σώζεται στην μνήμη στοίβας (ή σωρού, Stack memory). Στο παράδειγμα η διεύθυνση αυτή είναι η 0x00A.
Γίνεται άλμα στη διεύθυνση στην οποία είναι γραμμένη η πρώτη εντολή της υπορουτίνας.
Η ρουτίνα τελειώνει πάντα με την εντολή **return**. Κατά την εκτέλεση αυτής της εντολής διαβάζεται από την μνήμη stack η διεύθυνση επιστροφής στο πρόγραμμα (στο παράδειγμα η 0x00A), η τιμή αυτής της διεύθυνσης τοποθετείται στον Programm Counter και η εκτέλεση του προγράμματος συνεχίζεται από τη διεύθυνση επιστροφής.

Program memory

Address	Instruction code
	εντολές
	προγράμματος
	.....
0x006	call 0x800
0x008	
0x00A	επόμενη εντολή
...	...
...	...
...	...
0x800	1 εντολή υπορουτίνας
0x801	
	...
	...
	...
0x890	return

Υποροutίνες, Μνήμη Στοίβας (Stack memory) (3)



Προσοχή: Οι τιμές στο δίαυλο διευθύνσεων της μνήμης προγράμματος τοποθετούνται από τον Program Counter . Ο Program Counter είναι ένας καταχωρητής τοποθετημένος μέσα στην Κεντρική Μονάδα Επεξεργασίας (CPU).

Στη μνήμη Stack αποθηκεύονται οι διευθύνσεις επιστροφής κάθε φορά που γίνεται άλμα για να εκτελεστεί μια υποροutίνα.

Οι διατήρηση των διευθύνσεων αυτών είναι απαραίτητη ώστε μετά την εκτέλεση της υποροutίνας να μπορεί να συνεχιστεί η εκτέλεση του προγράμματος από το σημείο στο οποίο έγινε το άλμα προς την υποροutίνα.

Εντολή κλήσης υπορουτίνας call k{,s}

CALL	**Subroutine Call**
Syntax:	CALL k {,s}
Operands:	$0 \le k \le 1048575$ $s \in [0,1]$
Operation:	(PC) + 4 → TOS, k → PC<20:1>, if s = 1 (W) → WS, (STATUS) → STATUSS, (BSR) → BSRS
Status Affected:	None

Encoding:

1st word (k<7:0>)	1110	110s	$k_7$kkk	kkkk$_0$
2nd word(k<19:8>)	1111	k_{19}kkk	kkkk	kkkk$_8$

Description: Subroutine call of entire 2-Mbyte memory range. First, return address (PC + 4) is pushed onto the return stack. If 's' = 1, the W, STATUS and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = 0, no update occurs (default). Then, the 20-bit value 'k' is loaded into PC<20:1>. CALL is a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	Push PC to stack	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example:

```
HERE    CALL   THERE,1
```

Before Instruction
PC = address (HERE)
After Instruction
PC = address (THERE)
TOS = address (HERE + 4)
WS = W
BSRS = BSR
STATUSS = STATUS

TOS ---------- > TOP OF STACK, κορυφή της μνήμης στοίβας
WS ----------> STATUS SHADOW, αντίγραφο του W
STATUSS ------- -> STATUS SHADOW, αντίγραφο του STATUS
BSRS ---------->BANK SELECTION REGISTER SHADOW, αντίγραφο του Bank Selection Regster

Εντολή επιστροφής από υπορουτίνα (return)

RETURN	Return from Subroutine
Syntax:	RETURN {s}
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged
Status Affected:	None
Encoding:	0000 0000 0001 001s
Description:	Return from subroutine. The stack is popped and the top of the stack (TOS) is loaded into the program counter. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

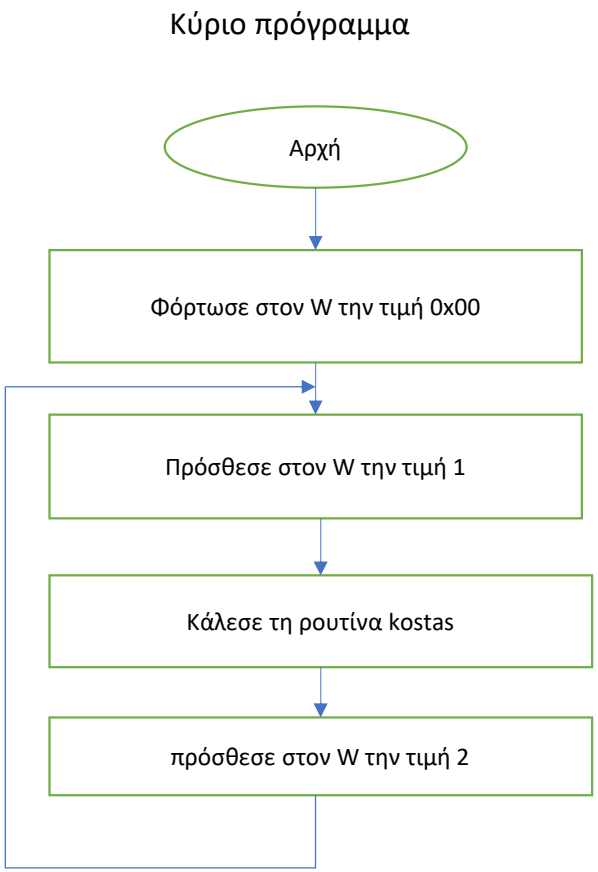
Q1	Q2	Q3	Q4
Decode	No operation	Process Data	Pop PC from stack
No operation	No operation	No operation	No operation

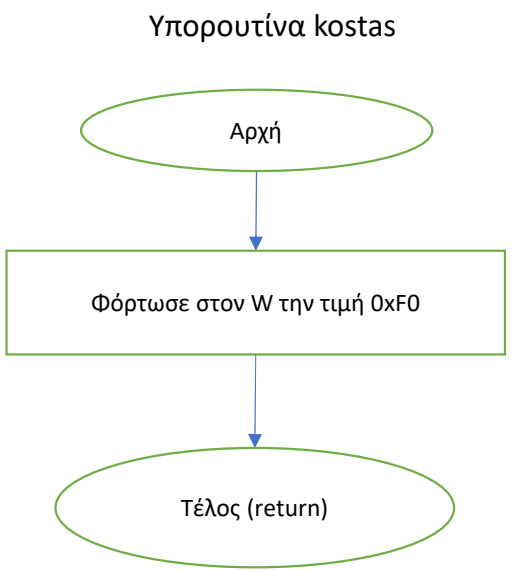
Example: RETURN

After Instruction:
PC = TOS

TOS ---------- > TOP OF STACK, κορυφή της μνήμης στοίβας
WS ----------> STATUS SHADOW, αντίγραφο του W
STATUSS ------- -> STATUS SHADOW, αντίγραφο του STATUS
BSRS ---------->BANK SELECTION REGISTER SHADOW, αντίγραφο του Bank Selection Register

Πρόγραμμα με υπορουτίνα (διάγραμμα ροής)





Προσοχή!

Το κύριο πρόγραμμα δεν έχει τέλος, εκτελείται συνεχώς ένας βρόχος.

Η υπορουτίνα έχει τέλος και η τελευταία εντολή είναι πάντα η εντολή **return.**

Παράδειγμα προγράμματος με υπορουτίνα (Παρακολούθηση της μνήμης stack)

kro\askisi-13 Subroutine 1\askisi-13.asm

```
        org 0x000       ; Οι παρακάτω εντολές θα γραφούν
                        ; από την διεύθυνση 000h και μετά
        movlw 0x00      ; φόρτωσε στον W την τιμή 00000000b
loop    addlw 0x01      ; πρόσθεσε την τιμή 1
        call kostas,1   ; κάλεσε την υποροutίνα που είναι
                        ; γραμμένη στην διεύθυνση που αντιστοιχεί
                        ; στην ετικέτα kostas
        addlw 0x02      ; πρόσθεσε στον w την τιμή 0χ02
        goto loop       ; πήγαινε στην ετικέττα loop, δηλαδή στη δ
                        ; που είναι γραμμένη η εντολή addlw

        org 0x01A       ; Οι παρακάτω εντολές θα γραφούν
                        ; από την διεύθυνση 01Ah και μετά
kostas  movlw 0XF0      ; φόρτωσε στον W την τιμή F0
        return 1        ; επιστροφή στο κύριο πρόγραμμα
        END
```

Program Memory

	Line	Address	Opcode	Disass
➡	1	0000	0E00	MOVLW 0
	2	0002	0F01	ADDLW 0x1
	3	0004	ED0D	CALL 0x1a, 0x1
	4	0006	F000	NOP
	5	0008	0F02	ADDLW 0x2
	6	000A	EF01	GOTO 0x2
	7	000C	F000	NOP
	8	000E	FFFF	NOP
	9	0010	FFFF	NOP
	10	0012	FFFF	NOP
	11	0014	FFFF	NOP
	12	0016	FFFF	NOP
	13	0018	FFFF	NOP
	14	001A	0EF0	MOVLW 0xf0
	15	001C	0013	RETURN 0x1
	16	001E	FFFF	NOP
	17	0020	FFFF	NOP
	18	0022	FFFF	NOP
	19	0024	FFFF	NOP

Την οδηγία προς τον Assembler **org 0x01A** θα μπορούσαμε να την παραλείψουμε.

Σ’ αυτή την περίπτωση ο κώδικας της υπορουτίνας θα γραφόταν ακριβώς κάτω από τον κώδικα του κύριου προγράμματος, δηλαδή στην περίπτωση του παραπάνω παραδείγματος, από τη διεύθυνση 000Eh και μετά.

Εκτέλεση του προγράμματος με υπορουτίνα και παρακολούθηση της μνήμης Stack, του Program Counter και του W από το παράθυρο Watch (1)

```
        org 0x000       ; Οι παρακάτω εντολές θα γραφούν
                        ; από την διεύθυνση 000h και μετά
        movlw 0x00      ; φόρτωσε στον W την τιμή 00000000b
loop    addlw 0x01      ; πρόσθεσε την τιμή 1
        call kostas,1   ; κάλεσε την υπορουτίνα που είναι
                        ; γραμμένη στην διεύθυνση που αντισ
                        ; στην ετικέτα kostas
        addlw 0x02      ; φόρτωσε στον w την τιμή 0χ02
        goto loop       ; πήγαινε στην ετικέττα loop, δηλαδ
                        ; που είναι γραμμένη η εντολή addlw

        org 0x01A       ; Οι παρακάτω εντολές θα γραφούν
                        ; από την διεύθυνση 01Ah και μετά
kostas  movlw 0XF0      ; φόρτωσε στον W την τιμή F0
        return 1        ; επιστροφή στο κύριο πρόγραμμα
        END
```

Program Memory

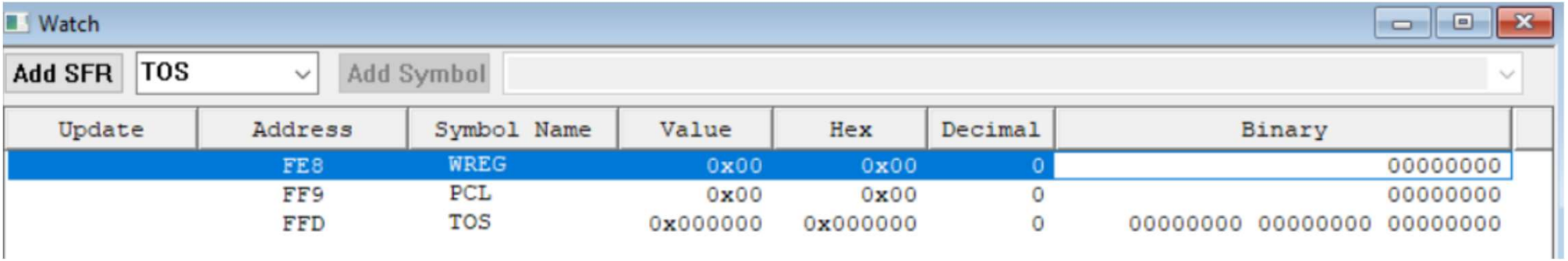
Line	Address	Opcode	Disass
1	0000	0E00	MOVLW 0
2	0002	0F01	ADDLW 0x1
3	0004	ED0D	CALL 0x1a, 0x1
4	0006	F000	NOP
5	0008	0F02	ADDLW 0x2
6	000A	EF01	GOTO 0x2
7	000C	F000	NOP
8	000E	FFFF	NOP
9	0010	FFFF	NOP
10	0012	FFFF	NOP
11	0014	FFFF	NOP
12	0016	FFFF	NOP
13	0018	FFFF	NOP
14	001A	0EF0	MOVLW 0xf0
15	001C	0013	RETURN 0x1
16	001E	FFFF	NOP
17	0020	FFFF	NOP
18	0022	FFFF	NOP
19	0024	FFFF	NOP

Watch

Add SFR | TOS | Add Symbol

Update	Address	Symbol Name	Value	Hex	Decimal	Binary
	FE8	WREG	0x00	0x00	0	00000000
	FF9	PCL	0x00	0x00	0	00000000
	FFD	TOS	0x000000	0x000000	0	00000000 00000000 00000000

Εκτέλεση του προγράμματος με υπορουτίνα και παρακολούθηση της μνήμης Stack, του Program Counter και του W από το παράθυρο Watch (2)



WREG -------- >W register
TOP --------- > Top of Stack Memory, Κορυφή της μνήμης στοίβας (stack memory)
PCL ---------- > Program Counter Low, Το λιγότερο σημαντικό byte του μετρητή προγράμματος

call kostas, 1 ; Αποθηκεύονται ο W, ο καταχωρητής STATUS και ο καταχωρητής BSR (Bank Selection register) στους αντίστοιχους καταχωρητές αποθήκευσης (Shadow registers) πριν το άλμα προς τη ρουτίνα.
call kostas, 0 ; Δεν αποθηκεύονται πουθενά οι καταχωρητές W, STATUS και BSR. Είναι η default κατάσταση, το 0 μπορεί να παραληφθεί.
return, 1 ; Κατά την επιστροφή από την υπορουτίνα **λαμβάνονται** οι τιμές του W, STATUS και BSR από τους αντίστοιχους shadow καταχωρητές. Δηλαδή παίρνουν τις τιμές που είχαν πριν την εκτέλεση της υπορουτίνας.
return, 0 ; Κατά την επιστροφή από την υπορουτίνα **δεν λαμβάνονται** οι τιμές του W, STATUS και BSR από τους αντίστοιχους shadow καταχωρητές. Είναι η default κατάσταση, το 0 μπορεί να παραληφθεί.

Εκτέλεση ενός συνόλου εντολών πολλές φορές (1)

Έστω ότι θέλουμε ένα σύνολο 10 εντολών να το επαναλάβουμε 5 φορές. Οι εντολές ας ονομαστούν εντολή 1, εντολή 2, … εντολή 10.

Δηλαδή κάτι αντίστοιχο με το παρακάτω που θα κάναμε στη γλώσσα C:

for(i=1; i<=5; i++) {εντολή 1, εντολή 2, … εντολή 10}

Θα φορτωθεί σε μια θέση μνήμης, π.χ. reg 1, την τιμή 5 και κάθε φορά που εκτελούνται οι 10 εντολές το περιεχόμενο της θέσης μνήμης θα ελαττώνεται κατά 1.

Όταν το περιεχόμενο της θέσης μνήμης γίνει 0 το πρόγραμμα θα προχωράει παρακάτω.

Ο έλεγχος ότι το περιεχόμενο της θέσης μνήμης έγινε 0 θα γίνεται με παρακολούθηση του Zero flag του καταχωρητή κατάστασης.

Εκτέλεση ενός συνόλου εντολών πολλές φορές (1)

Έστω ότι θέλουμε ένα σύνολο 10 εντολών να το επαναλάβουμε 5 φορές. Οι εντολές ας ονομαστούν εντολή 1, εντολή 2, … εντολή 10.

```
STATUS EQU FD8    ; FD8 είναι η διεύθυνση του καταχωρητή κατάστασης στη μνήμη δεδομένων του μικροελεγκτή PIC18F4550. (Το γράφουμε πριν το πρόγραμμα)
reg1   EQU 0x010 ; ονομάζουμε reg1 την διεύθυνση 010 στον χάρτη μνήμης. (Το γράφουμε πριν το πρόγραμμα)

        movlw 0x05
        movwf reg1

loop1   εντολή 1        ; Γράφουμε τις εντολές που θέλουμε να επαναλάβουμε 5 φορές
        εντολή 2
        ...
        …
        …
        εντολή 20
decfsz reg1,1           ; decrease file reg1 skip if zero
                        ; Ελάττωσε το περιεχόμενο της θέσης μνήμης reg1 και παράκαμψε την
                        ; επόμενη εντολή όταν το περιεχόμενο της θέσης μνήμης γίνει 0.
                        ;(η εντολή αυτή ελέγχει το αποτέλεσμα αν είναι 0)
                         ; το ,1 σημαίνει ότι το αποτέλεσμα της ελάττωσης του reg1
                        ; κατά 1 αποθηκεύεται στο reg1. Δηλαδή το reg1 θα παίρνει τιμές
                        ; 5(αρχική τιμή),4,3,2,1,0
        goto loop1

        επόμενες εντολές του προγράμματος
        .
        .
        .
```

Θεωρώντας δεδομένη μια ρουτίνα καθυστέρησης 50 ms να γράψετε μια ρουτίνα καθυστέρησης 1 sec. Έστω ότι το όνομα της ρουτίνας καθυστέρησης είναι delay_50ms. Η ρουτίνα καθυστέρησης 1 δευτερολέπτου θα έχει το όνομα delay_1s.

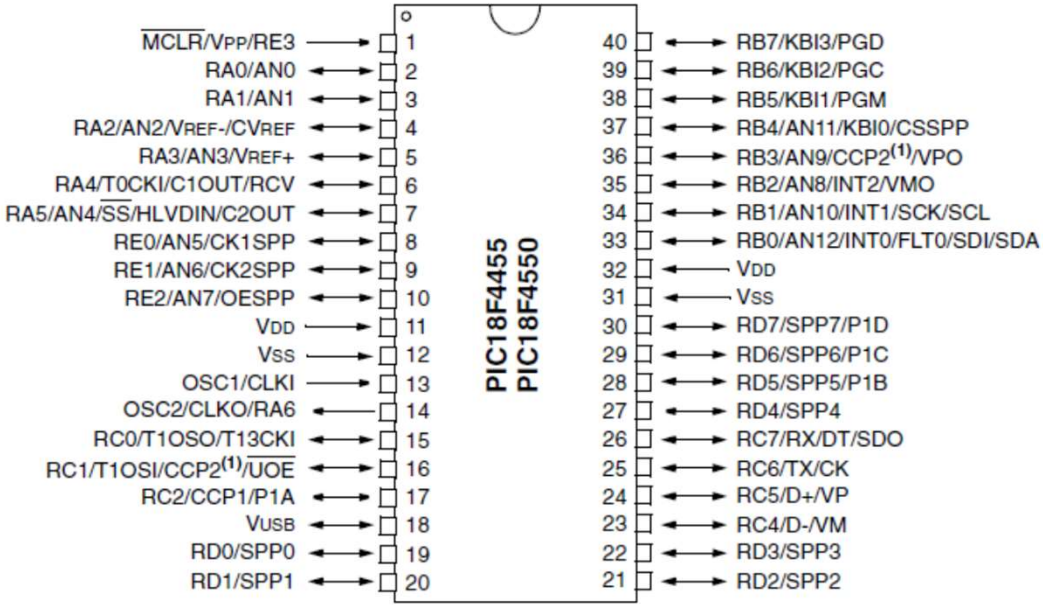
Για να έχουμε καθυστέρηση 1 sec θα πρέπει η ρουτίνα καθυστέρησης των 50 ms να εκτελεστεί $\frac{1000\text{ms}}{50\ ms} = 20\ \varphi o \rho \acute{\varepsilon} \varsigma.$

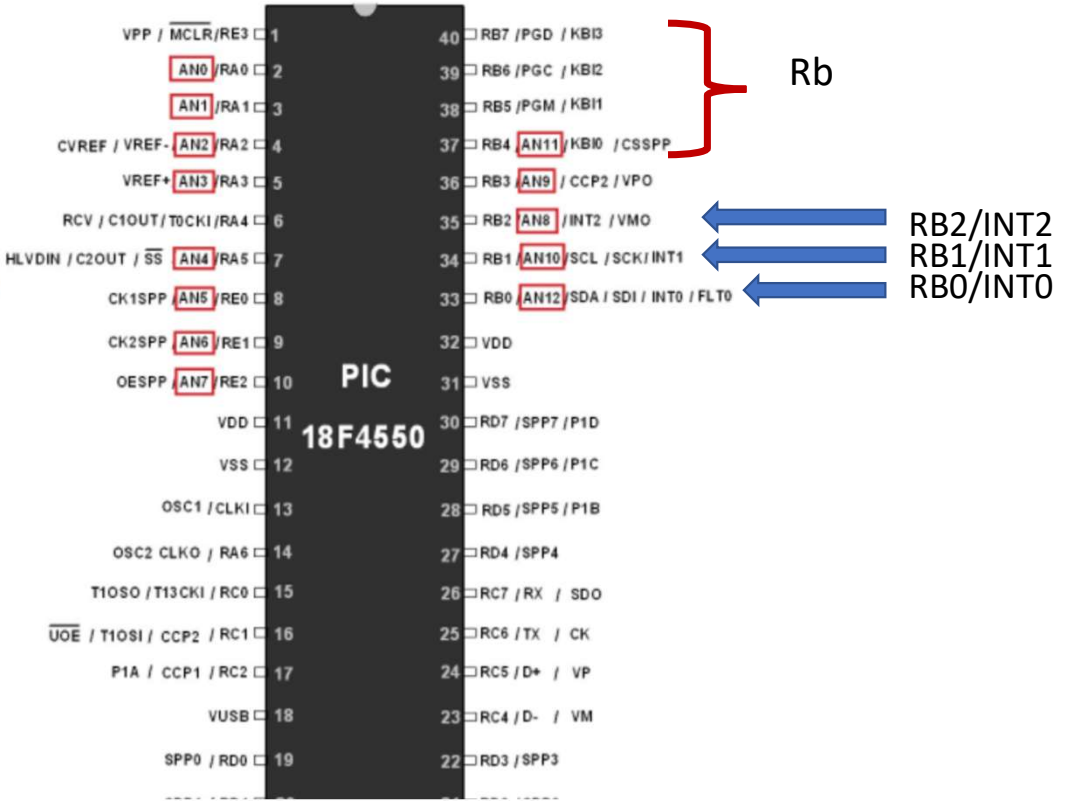
```
delay_1s
movlw D'20'      ; φορτώνεται στον W η τιμή (20)10 έτσι ώστε η ρουτίνα καθυστέρησης
                 ; των 50 sec  να εκτελεστεί 20 φορές.
                 ; delay_1s είναι το όνομα της ρουτίνας καθυστέρησης 1 δευτερολέπτου.
        movwf reg1          ; reg1 είναι μια θέση μνήμης που πρέπει να δηλώσουμε στην αρχή του
                            ; Προγράμματος.
loop1   call delay_50ms
        decfsz  reg1        ; decrease file reg1 skip if zero
                            ; Ελάττωσε το περιεχόμενο της θέσης μνήμης reg1 και παράκαμψε την
                            ; επόμενη εντολή(goto loop1) όταν το περιεχόμενο της θέσης μνήμης γίνει 0.
                            ;(η εντολή αυτή ελέγχει το αποτέλεσμα της μείωσης)
                            ; Δηλαδή όταν το reg1 θα γίνει 0 δεν θα εκτελεστεί η goto loop1 και
                            ; θα εκτελεστεί η εντολή return.
        goto loop1
        return              ; η ρουτίνα καθυστέρησης delay_1 s τελειώνει με την εντολή return
                            ; όπως και όλες οι ρουτίνες.
```

Διακοπές (Interrupts)

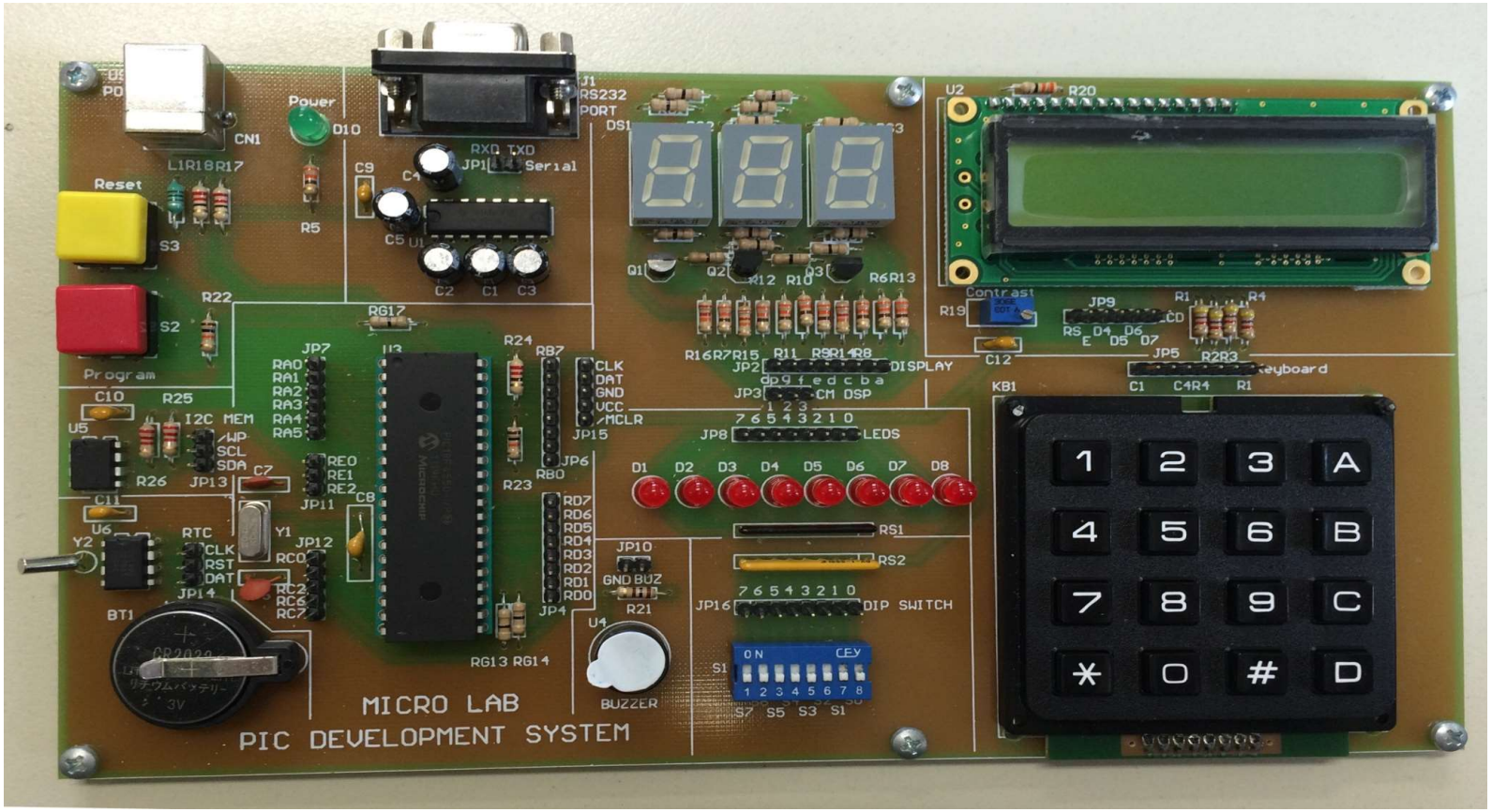
Εξωτερικές διακοπές από τους ακροδέκτες RB0/INT0, RB1/INT1, RB2INT2
Εξωτερικές διακοπές από τους ακροδέκτες Rb (RB4, RB5, RB6, RB7)

40-Pin PDIP





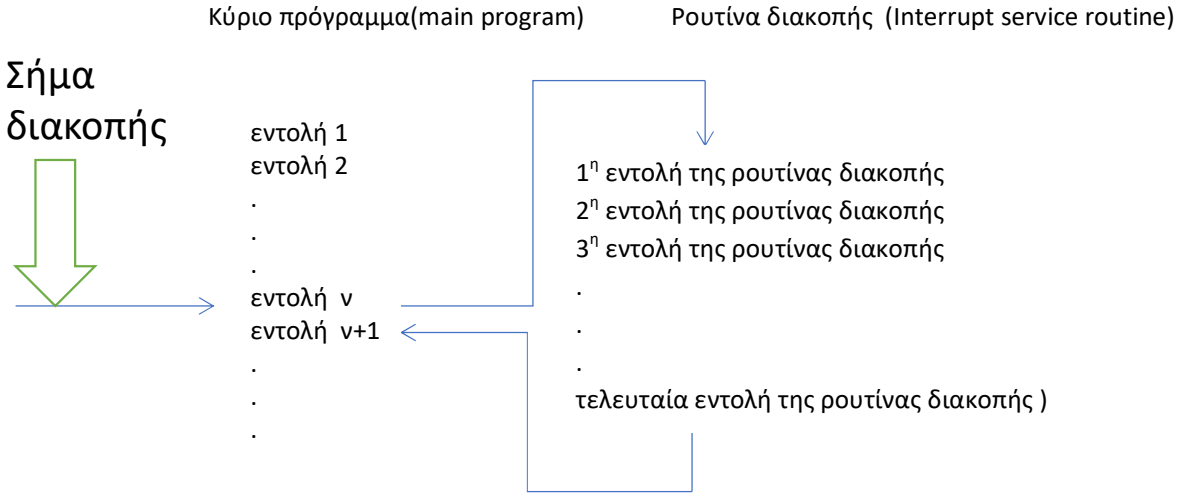
Πλακέτα ανάπτυξης εφαρμογών



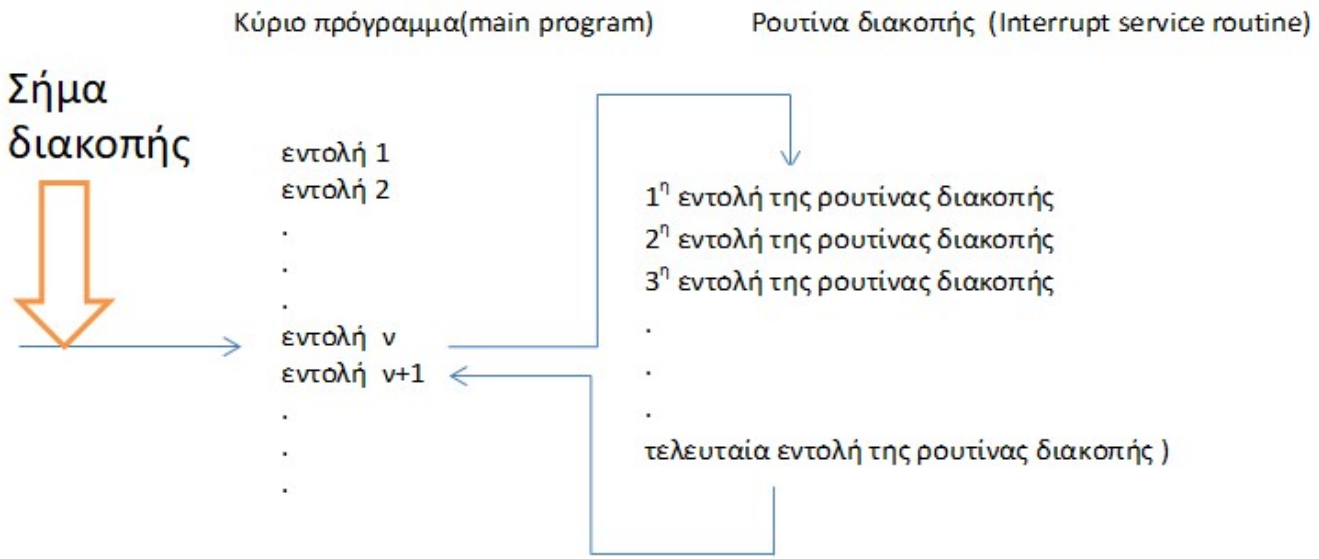
Διακοπές (Interrupts)

Τι εννοούμε με τον όρο διακοπή (Interrupt) στους μικροελεγκτές;
Διακοπή(Interrupt) είναι η διακοπή της εκτέλεσης της ροής του προγράμματος του μικροελεγκτή προκειμένου να εκτελεστεί ένα κομμάτι προγράμματος το οποίο ονομάζεται ρουτίνα διακοπής

Παράδειγμα:



Ένα σήμα προκαλεί την διακοπή της εκτέλεση του κύριου προγράμματος κατά τη διάρκεια της εκτέλεσης της εντολής ν προκειμένου να εκτελεστεί η ρουτίνα διακοπής. Όταν τελειώσει η ρουτίνα διακοπής εκτελείται η επόμενη εντολή του κύριου προγράμματος δηλαδή η εντολή ν+1.



Τι είναι το σήμα διακοπής;

Α) Το σήμα διακοπής μπορεί να είναι μια αλλαγή κατάστασης σε έναν εξωτερικό ακροδέκτη του μικροελεγτκή που χρησιμοποιείται σαν ακροδέκτης διακοπής.

Σ' αυτή την περίπτωση η διακοπή λέγεται εξωτερική διακοπή(external interrupt) γιατί ένα σήμα από τον εξωτερικό κόσμο αλλάζει την ροή εκτέλεσης του προγράμματος προκειμένου να εκτελεστεί η ρουτίνα εξυπηρέτησης αυτής της διακοπής

Β Το σήμα διακοπής μπορεί να προέρχεται από το εσωτερικό του μικροελεγκτή, για παράδειγμα από ένα από τα ρολόγια(timers) του μικροελεγκτή.

Σ' αυτή την περίπτωση η διακοπή λέγεται εσωτερική διακοπή(internal interrupt) γιατί ένα σήμα από το εσωτερικό του μικροελεγκτή αλλάζει την ροή εκτέλεσης του προγράμματος προκειμένου να εκτελεστεί η ρουτίνα εξυπηρέτησης αυτή της διακοπής

Διαφορές των υπορουτινών(subroutines) που καλούνται από το πρόγραμμα από τις ρουτίνες διακοπών(interrupt routines)

Οι υπορουτίνες καλούνται για να εκτελεστούν από συγκεκριμένα σημεία του προγράμματος με την εντολή call XXX, όπου XXX είναι μια ετικέτα που αντιστοιχεί σε μια διεύθυνση στη μνήμη προγράμματος.

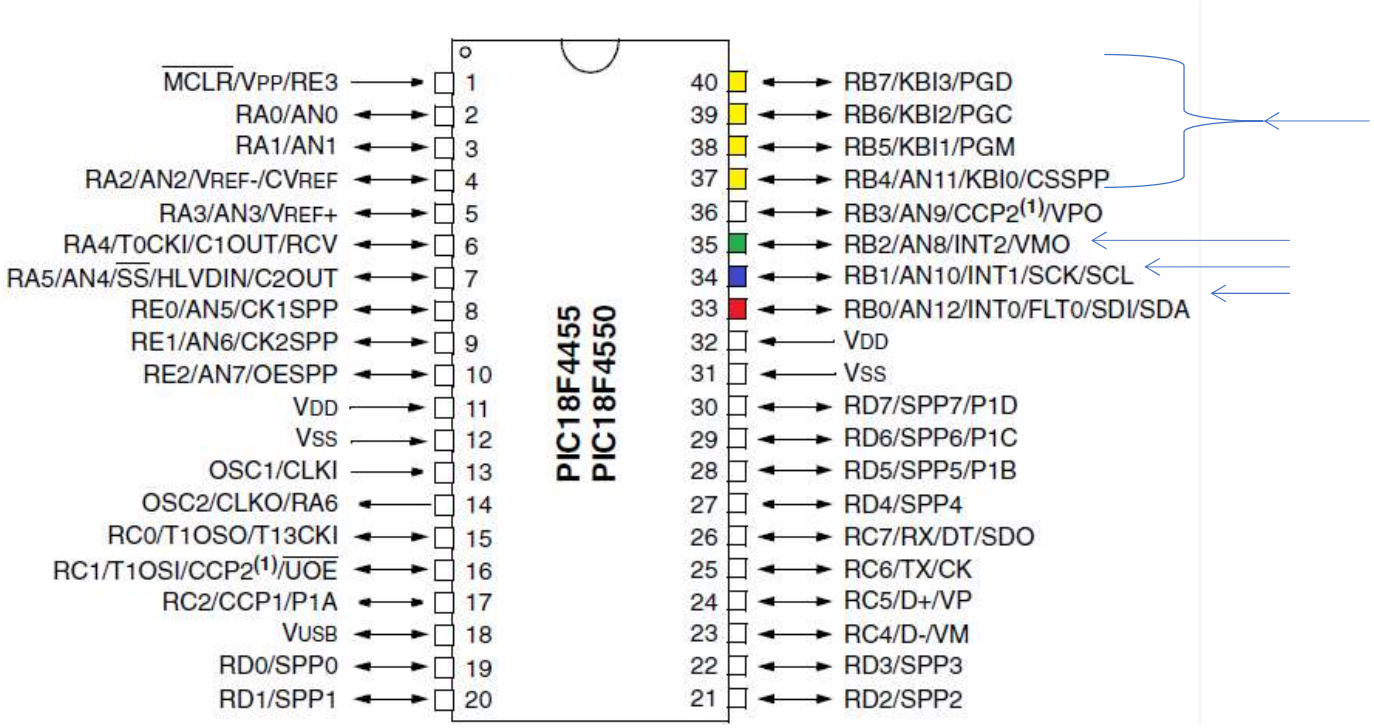
Η τελευταία εντολή των υπορρουτινών είναι η εντολή return.

Οι ρουτίνες διακοπών δεν καλούνται από συγκεκριμένα σημεία του προγράμματος με μια εντολή αλλά ενεργοποιούνται από σήματα που στέλνονται δια μέσου των ακροδεκτών προς τον μικροελεγκτή ή από συμβάντα στο εσωτερικό του μικροελεγκτή. Επομένως οι ρουτίνες διακοπών μπορούν να ενεργοποιηθούν σε οποιοδήποτε σημείο του προγράμματος.

Όταν συμβεί μια διακοπή το πρόγραμμα συνεχίζεται πάντα από συγκεκριμένη διεύθυνση η οποία ονομάζεται διάνυσμα διακοπής (Interrupt Vector)

Η τελευταία εντολή των ρουτινών διακοπών είναι πάντα η εντολή retfie

Ακροδέκτες εξωτερικών διακοπών του μικροελεκτή PIC 18F4550



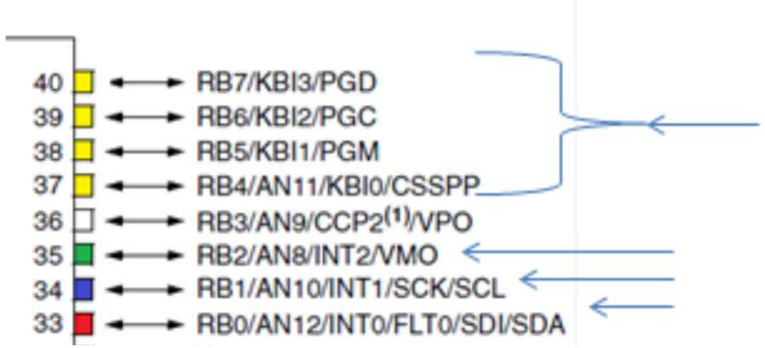
Τα σήματα εξωτερικών διακοπών θα πρέπει να εφαρμοστούν μόνο στους ακροδέκτες που έχουν προβλεφτεί από τον σχεδιαστή του μικροελεγκτή για τον σκοπό αυτό.

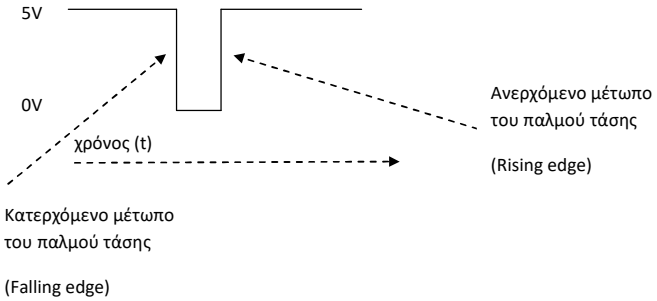
Είναι οι ακροδέκτες RB0, RB1, RB2 που φέρουν την ένδειξη INT0, INT1, INT2

Και οι ακροδέκτες RB4, RB5, RB6 και RB7 (Διακοπή Rb)

Τι είναι αυτό που προκαλεί την εξωτερική διακοπή ;

Αυτό που προκαλεί την εξωτερική διακοπή είναι η αλλαγή κατάστασης σε έναν ακροδέκτη διακοπών, π.χ. στον ακροδέκτη RB0/INT0

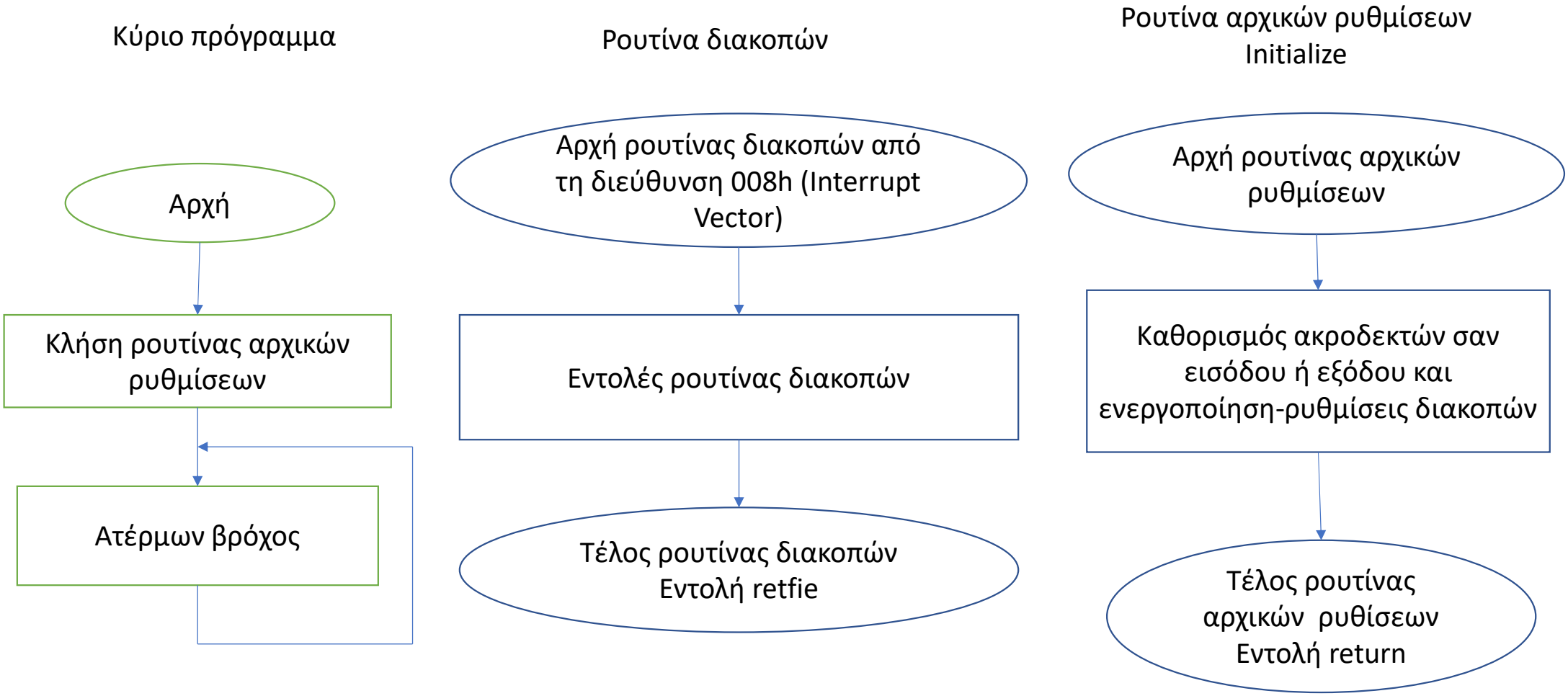




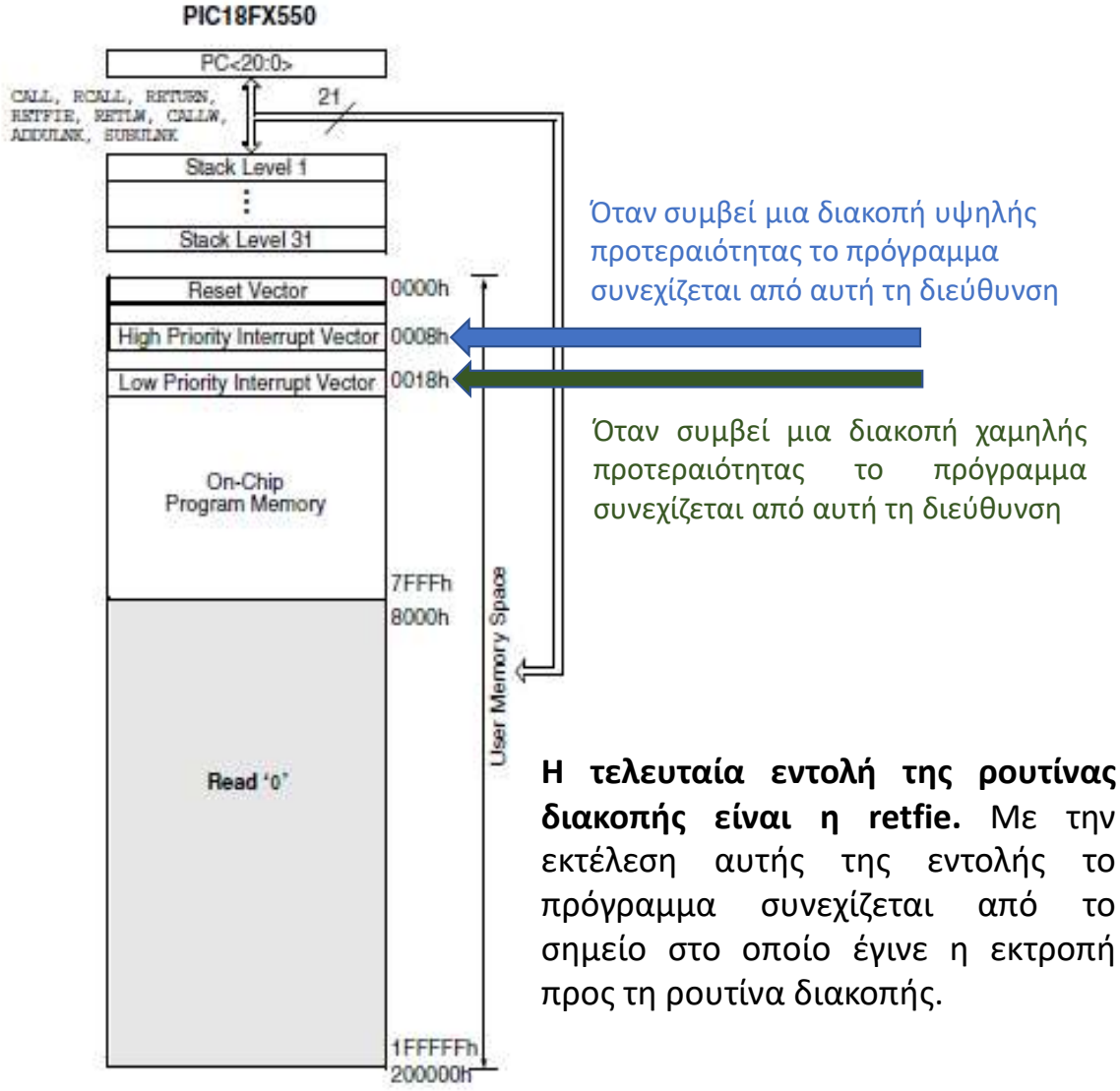
Την διακοπή την προκαλεί είτε το ανερχόμενο είτε το κατερχόμενο μέτωπο ενός παλμού που εφαρμόζουμε σε έναν από τους ακροδέκτες εξωτερικών διακοπών.

Εμείς στο πρόγραμμα μας, με μια συγκεκριμένη εντολή δηλώνουμε αν θέλουμε η διακοπή νε συμβαίνει στο ανερχόμενο ή στο κατερχόμενο μέτωπο του παλμού που εφαρμόζουμε στον ακροδέκτη διακοπών

Διάγραμμα ροής προγράμματος με ρουτίνα διακοπών



Διανύσματα διακοπών (Interrupt Vectors)



Η τελευταία εντολή της ρουτίνας διακοπής είναι η retfie. Με την εκτέλεση αυτής της εντολής το πρόγραμμα συνεχίζεται από το σημείο στο οποίο έγινε η εκτροπή προς τη ρουτίνα διακοπής.

Όταν συμβεί μια διακοπή υψηλής προτεραιότητας το πρόγραμμα συνεχίζεται από τη διεύθυνση του διανύσματος υψηλής προτεραιότητας (High Priority Interrupt Vector).
Στον PIC 18F4550 η διεύθυνση αυτή είναι: 0008h

Όταν συμβεί μια διακοπή χαμηλής προτεραιότητας το πρόγραμμα συνεχίζεται από τη διεύθυνση του διανύσματος υψηλής προτεραιότητας (Low Priority Interrupt Vector).
Στον PIC 18F4550 η διεύθυνση αυτή είναι: 0018h

Οι διακοπές από τον ακροδέκτη INT0 είναι πάντα υψηλής προτεραιτότητας. Επομένως δεν χρειάζεται να δηλωθεί με εντολή η προτεραιότητα τους.

Οι διακοπές από τους ακροδέκτες INT1 και INT2 μπορούνε να είναι είτε υψηλής είτε χαμηλής προτεραιότητας. Το είδος της προτεραιότητας το καθορίζουμε από την τιμή που δίνουμε στο αντίστοιχο bit ελέγχου προτεραιότητας στον καταχωρητή INTCON3.

Καταχωρητής ελέγχου διακοπών INTCON

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER INTCON EQU 0xFF2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all high priority interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit(1)
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Ο Καταχωρητής ελέγχου διακοπών ευρίσκεται στη θέση FF2h στους Special Function Registers στη μνήμη δεδομόνων του Μικροελεγκτή

GIE/GIEH Ενεργοποίηση-απενεργοποίηση όλων των διακοπών (General Interrupt Enable)

INT0IE Ενεργοποίηση των διακοπών από τον ακροδέκτη INT0 (INT0 Enable)

INT0IF (Σημαία διακοπής, γίνεται 1 όταν συμβεί διακοπή. Πρέπει να γίνει 0 με εντολή για να μπορεί να συμβεί νέα διακοπή)

Καταχωρητής ελέγχου διακοπών INTCON2

REGISTER 9-2: INTCON2: INTERRUPT CONTROL REGISTER 2 INTCON2 EQU 0xFF1

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP:** RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

Ενεργοποίηση διακοπών στο ανερχόμενο μέτωπο των παλμών που εφαρμόζονται στον ακροδέκτη INT0.

```
INTCON2 EQU 0xFF1
INTEDG0 EQU D'6'
bsf  INTCON2, INTEDG0
```

Ενεργοποίηση διακοπών στο κατερχόμενο μέτωπο των παλμών που εφαρμόζονται στον ακροδέκτη INT0

```
INTCON2 EQU 0xFF1
INTEDG0 EQU D'6'
bcf  INTCON2, INTEDG0
```

Καταχωρητής ελέγχου διακοπών INTCON3

REGISTER 9-3: INTCON3: INTERRUPT CONTROL REGISTER 3 INTCON3 EQU 0xFF0

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Note: Interrupt flag bits are set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global interrupt enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

Ορισμός της διακοπής από τον ακροδέκτη **INT1** σαν **υψηλής προτεραιότητας (High Priority Interrupt)**

```
INTCON3 EQU 0xFF0
INT1IP   EQU D'6'

bsf  INTCON3, INT1IP
```

Ορισμός της διακοπής από τον ακροδέκτη **INT1** σαν **χαμηλής προτεραιότητας (High Priority Interrupt)**

```
INTCON3 EQU 0xFF0
INT1IP   EQU D'6'

bcf  INTCON3, INT1IP
```

Παράδειγμα ενεργοποίησης εξωτερικών διακοπών από τον ακροδέκτη INT0 στο ανερχόμενο μέτωπο του παλμού εισόδου

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER INTCON EQU 0xFF2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

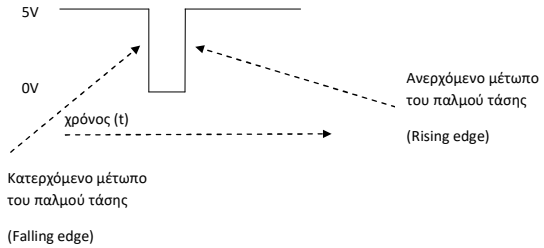
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 9-2: INTCON2: INTERRUPT CONTROL REGISTER 2 INTCON2 EQU 0xFF1

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

```
INTCON   EQU 0xFF2  ; Θέση του INTCON στη μνήμη δεδομένων του μικροελεγκτή (Special Function Registers, SFR)
INTCON2 EQU 0xFF1  ; Θέση του INTCON2 στη μνήμη δεδομένων του μικροελεγκτή (Special Function Registers, SFR)
GIE         EQU D'7'    ; Θέση του bit GIE στον αντίστοιχο καταχωρητή ελέγχου
INT0IE      EQU D'4'    ; Θέση του bit INT0IE στον αντίστοιχο καταχωρητή ελέγχου
INT0IF      EQU D'1'    ; Θέση του bit INT0IF στον αντίστοιχο καταχωρητή ελέγχου
INTEDG0  EQU D'6'    ; Θέση του bit INTEDG0 στον αντίστοιχο καταχωρητή ελέγχου
```




```
bsf INTCON,   GIE          ; Ενεργοποίηση όλες τις  διακοπές
bsf INTCON,   INT0IE       ; Ενεργοποίησε τις διακοπές από τον ακροδέκτη INT0
bcf INTCON,   INT0IF       ; Κατέβασε(δηλαδή κάνε 0) τη σημαία διακοπών από τον ακροδέκτη INT0
bsf  INTCON2, INTEDG0   ; Να συμβαίνουν διακοπές στο ανερχόμενο μέτωπο του παλμού εισόδου
```

Η τελευταία εντολή της ρουτίνας διακοπής retfie

RETFIE	**Return from Interrupt**
Syntax:	RETFIE {s}
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, 1 → GIE/GIEH or PEIE/GIEL, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged
Status Affected:	GIE/GIEH, PEIE/GIEL.
Encoding:	0000 0000 0001 000s

Description: Return from interrupt. Stack is popped and Top-of-Stack (TOS) is loaded into the PC. Interrupts are enabled by setting either the high or low priority global interrupt enable bit. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Pop PC from stack Set GIEH or GIEL
No operation	No operation	No operation	No operation

Example: `RETFIE 1`

After Interrupt

PC = TOS
W = WS
BSR = BSRS
STATUS = STATUSS
GIE/GIEH, PEIE/GIEL = 1

Καταχωρητές ελέγχου διακοπών, σημαία διακοπών (Interrupt flag)

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 9-2: INTCON2: INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 9-3: INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Η σημαία διακοπών από μια συγκεκριμένη πηγή σηκώνεται (is set) όταν προκληθεί μια διακοπή και εκτελείται η ρουτίνα διακοπής. Πριν την έξοδο από τη ρουτίνα διακοπής, δηλαδή πριν από την εντολή RETFIE, θα πρέπει να την κατεβάσουμε (clear) με μια εντολή.

Για την διακοπή που προκαλείται από τον ακροδέκτη INT0 αυτό γίνεται με την εντολή:

bcf INTCON, INT0IF

Για όσο διάστημα είναι ανεβασμένη η σημαία των διακοπών δεν μπορεί να γίνει διακοπή.

Δηλαδή αν δεν κατεβάσουμε τη σημαία της διακοπής από INT0 δεν μπορεί να προκληθεί νέα διακοπή από τον ακροδέκτη INT0

INT0IF --- > Interrupt 0 Interrupt Flag
INT1IF --- > Interrupt 1 Interrupt Flag
INT2IF --- > Interrupt 2 Interrupt Flag

Ορισμός της διακοπής από τον ακροδέκτη INT1 σαν χαμηλής προτεραιότητας (Low Priority)

Όταν δηλώσουμε τις διακοπές από σήμα στον ακροδέκτη INT1 σαν χαμηλής προτεραιότητας η ρουτίνα διακοπών θα ξεκινήσει να εκτελείται από τη διεύθυνση στη μνήμη προγράμματος Low Priority Interrupt Vector, δηλαδή τη διεύθυνση 0x0018.

```
;*** Δηλώνουμε τις διακοπές από INT1 σαν χαμηλής προτεραιότητας*****************************************
bsf INTCON, GIEL        ;Επιτρέπονται όλα τα peripheral interrupts. Προσοχή να μην ξεχαστεί!!!
bsf RCON, IPEN          ;Ενεργοποίηση προτεραιοτήτων.  Προσοχή να μην ξεχαστεί!!!
bcf INTCON3,   INT1IP   ; Κάνουμε 0 το bit προτεραιτότητας για διακοπές από INT1 στον καταχωρητή
                        ; ελέγχου INTCON3. Έτσι οι διακοπές από INT1 γίνονται χαμηλής προετεραιότητας
                        ; και κατευθύνονται στη διεύθυνση 0x018 (Low priority interrupt vector) και
                        ; όχι στη διεύθυνση 0x008

bsf INTCON3,   INT1IE   ; Ενεργοποίησε τις διακοπές από τον ακροδέκτη INT1
bcf INTCON3,   INT1IF   ; Κατέβασε(δηλαδή κάνε 0) τη σημαία διακοπών από τον ακροδέκτη INT1
bsf INTCON2,  INTEDG1  ; Να συμβαίνουν διακοπές στο ανερχόμενο μέτωπο του παλμού εισόδου
```

REGISTER 4-1: RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(2)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX Compatibility mode)

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Πως γράφουμε τη ρουτίνα υψηλής προτεραιότητας στο πρόγραμμα μας (High priority service routine)

```
;*************************************Ρουτίνα διακοπής υψηλής προτεραιότητας ************************************
            org 0x008           ;Στη διεύθυνση 0x008(High priority Interrupt Vector) γράφουμε την πρώτη εντολή
                                ;της ρουτίνας διακοπής. Όταν συμβεί διακοπή η ροή του προγράμματος συνεχίζεται
                                ;από τη διεύθυνση 0x008 (High priority Interrupt Vector)
            goto int_sr_H           ;Αυτή είναι η πρώτη εντολή της ρουτίνας διακοπής, γίνεται εκτροπή προς την
                                ;διεύθυνση που αντιστοιχεί στην ετικέττα int_sr όπου είναι γραμμένος ο κώδικας
                                ;της ρουτίνας διακοπής.
                                ;Τον κώδικα της ρουτίνας διακοπής θα μπορούσαμε να τον γράψουμε όλον εδώ
                                ;και να μην βάλουμε την εντολή goto.
                                ;Συνηθίζεται όμως να οργανώνουμε το πρόγραμμα μας με τον τρόπο που το κάνουμε
                                ;σε αυτό το παράδειγμα έτσι ώστε να φαίνονται ξεκάθαρα οι ενότητες από τις
                                ;οποίες αποτελείται. "Κύριο πρόγραμμα", "αρχικές ρυθμίσεις", "ρουτίνα διακοπών".
;*****************************************************************************************************************


;********************************** interrupt service routine ****************************************************
int_sr_H    incf PORTD,1            ; Από εδώ συνεχίζεται η ρουτίνα διακοπών που ξεκίνησε από τη
                                    ; διεύθυνση 0x008
            bcf INTCON,   INT0IF    ; Κατέβασε(δηλαδή κάνε 0) τη σημαία διακοπών από τον ακροδέκτη INT0.
                                    ; Αυτό είναι απαραίτητο ώστε να μπορεί να συμβεί νέα διακοπή από INT0.
                                    ; Όταν είναι ανεβασμένη η σημαία διακποπώνς από INT0 δεν μπορούν να
                                    ; συμβούν διακοπές από INT0.
            bcf INTCON3,  INT1IF
            retfie                  ; Τελευταία εντολή της ρουτίνας διακοπών.
;*******************************Τέλος ρουτίνας διακοπών από INT0*****************************************************
```

Πως γράφουμε τη ρουτίνα χαμηλής προτεραιότητας στο πρόγραμμα μας (Low Priority Service Routine)

```
;*********************************Αρχή ρουτίνας διακοπής χαμηλής προτεραιότητας **********************************
            org 0x0018              ;Στη διεύθυνση 0x018(Low priority Interrupt Vector) γράφουμε την πρώτη εντολή
                                    ;της ρουτίνας διακοπής. Όταν συμβεί διακοπή η ροή του προγράμματος συνεχίζεται
                                    ;από τη διεύθυνση 0x018 (Low priority Interrupt Vector)
            goto int_sr_L           ;Αυτή είναι η πρώτη εντολή της ρουτίνας διακοπής, γίνεται εκτροπή προς την
                                    ;διεύθυνση που αντιστοιχεί στην ετικέττα int_sr_L όπου είναι γραμμένος ο κώδικας
                                    ;της ρουτίνας διακοπής χαμηλής προτεραιότητας.
                                    ;Τον κώδικα της ρουτίνας διακοπής θα μπορούσαμε να τον γράψουμε όλον εδώ
                                    ;και να μην βάλουμε την εντολή goto.
                                    ;Συνηθίζεται όμως να οργανώνουμε το πρόγραμμα μας με τον τρόπο που το κάνουμε
                                    ;σε αυτό το παράδειγμα έτσι ώστε να φαίνονται ξεκάθαρα οι ενότητες από τις
                                    ;οποίες αποτελείται. "Κύριο πρόγραμμα", "αρχικές ρυθμίσεις", "ρουτίνα διακοπών".
;*****************************************************************************************************************


;********************************* interrupt service routine από INT1*************************************************
int_sr_L    decf PORTD,1            ; Από εδώ συνεχίζεται η ρουτίνα διακοπών που ξεκίνησε από τη
                                    ; διεύθυνση 0x018 (Low priority interrupt vector)
            bcf INTCON3,  INT1IF    ; Κατέβασε(δηλαδή κάνε 0) τη σημαία διακοπών από τον ακροδέκτη INT1.
                                    ; Αυτό είναι απαραίτητο ώστε να μπορεί να συμβεί νέα διακοπή από INT1.
                                    ; Όταν είναι ανεβασμένη η σημαία διακποπώνς από INT0 δεν μπορούν να
                                    ; συμβούν διακοπές από INT0
            retfie                  ; Τελευταία εντολή της ρουτίνας διακοπών
;*******************************Τέλος ρουτίνας διακοπών από INT1*************************************************
```

Δομή προγράμματος με μια ρουτίνα διακοπής υψηλής προτεραιότητας, (high priority interrupt)

```
                #include <set_fuse.inc>      ; Αρχείο που περιλαμβάνει τις θέσεις των καταχωρητών κατεύθυνσης και δεδομένων
                                             ; των πορτών καθώς και των καταχωρητών κατάστασης και ελέγχου των διακοπών.
                                             ; Το αρχείο αυτό περιλαμβάνει και άλλες ρυθμίσεις του μικροελεγκτή.
                org 0x000                    ; Δηλώνουμε την διεύθυνση στη μνήμη προγράμματος απ' όπου θα
                                             ; ξεκινήσει να γράφεται ο κώδικας του προγράμματος
                 call  init                  ; Καλείται η ρουτίνα που περιλαμβάνει τις αρχικές ρυθμίσεις του
                                             ; μικροελεγκτή. Η ρουτίνα αυτή είναι γραμμένη στη διεύθυνση
                                             ; που αντιστοιχεί στην ετικέτα init.
                goto  main                   ; Γίνεται μετάβαση στη διεύθυνση που αντιστοιχεί στην ετικέτα main
                                             ; όπου γράφεται το κύριο πρόγραμμα

                org 0x008                    ; Στη διεύθυνση 0x008 γράφεται η ρουτίνα διακοπής
                goto  int_sr_hp              ; Γίνεται μετάβαση στην ετικέτα int_sr_hp που αντιστοιχεί στη
                                             ; διεύθυνση όπου γράφεται

                                             ; η ρουτίνα διακοπής
main            ……..                         ;Αρχή εντολών κύριου προγράμματος **********
                ……..                         ; ……εντολές…..
                ……..                         ; Τέλος εντολών κύριου προγράμματος**********
int_sr_hp       ……..                         ; Αρχή εντολών ρουτίνας διακοπών      **********
                ……..                         ;  ……εντολές….
                retfie                       ; Τέλος εντολών ρουτίνας διακοπών      **********
init            ……..                         ;Αρχή εντολών της ρουτίνας αρχικών ρυθμίσεων *****
                ……..                         ; ……εντολές….
                return                       ;Τέλος εντολών της ρουτίνας αρχικών ρυθμίσεων *****
```

Το πρόγραμμα αποτελείται από 3 ενότητες: 1. Κύριο πρόγραμμα, 2. Ρουτίνα διακοπής, 3. Ρουτίνα αρχικών ρυθμίσεων

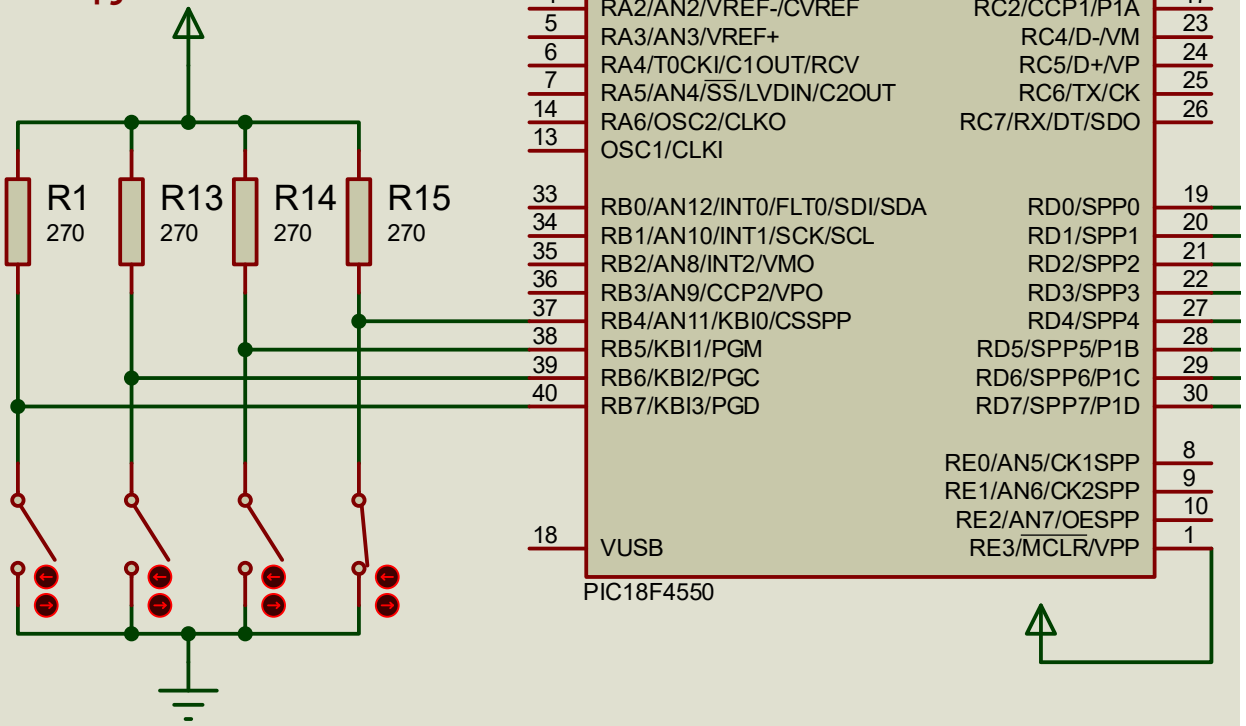
Δομή προγράμματος με μια ρουτίνα διακοπής υψηλής προτεραιότητας και μια ρουτίνα χαμηλής προτεραιότητας

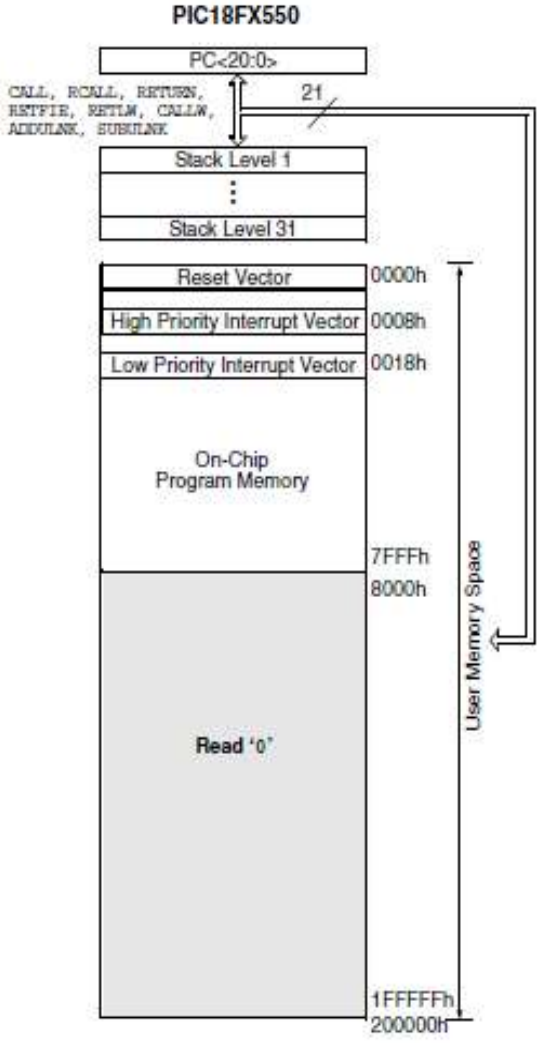
```
                #include <set_fuse.inc>      ; Αρχείο που περιλαμβάνει τις θέσεις των καταχωρητών κατεύθυνσης και δεδομένων
                                             ; των πορτών καθώς και των καταχωρητών κατάστασης και ελέγχου των διακοπών.
                                             ; Το αρχείο αυτό περιλαμβάνει και άλλες ρυθμίσεις του μικροελεγκτή.
                org 0x000                    ; Δηλώνουμε την διεύθυνση στη μνήμη προγράμματος απ' όπου θα
                                             ; ξεκινήσει να γράφεται ο κώδικας του προγράμματος
                 call  init                  ; Καλείται η ρουτίνα που περιλαμβάνει τις αρχικές ρυθμίσεις του
                                             ; μικροελεγκτή. Η ρουτίνα αυτή είναι γραμμένη στη διεύθυνση
                                             ; που αντιστοιχεί στην ετικέτα init.
                goto  main                   ; Γίνεται μετάβαση στη διεύθυνση που αντιστοιχεί στην ετικέτα main
                                             ; όπου γράφεται το κύριο πρόγραμμα

                org 0x008                    ; Στη διεύθυνση 0x008 γράφεται η ρουτίνα διακοπής υψηλής προτεραιότητας
                goto  int_sr_hp              ; Γίνεται μετάβαση στην ετικέτα int_sr_hp που αντιστοιχεί στη
                                             ; διεύθυνση όπου γράφεται η ρουτίνα διακοπής υψηλής προτεραιότητας
                org 0x018                    ; Στη διεύθυνση 0x018 γράφεται η ρουτίνα διακοπής χαμηλής προτεραιότητας
                goto  int_sr_lp              ; Γίνεται μετάβαση στην ετικέτα int_sr_lp που αντιστοιχεί στη
                                             ; διεύθυνση όπου γράφεται η ρουτίνα διακοπής χαμηλής προτεραιότητας

main            ……..                         ;Αρχή εντολών κύριου προγράμματος ******************************
                ……..                         ; ……εντολές…..
                ……..                         ; Τέλος εντολών κύριου προγράμματος******************************
int_sr_hp       ……..                         ; Αρχή εντολών ρουτίνας διακοπών υψηλής προτεραιότητας************
                ……..                         ;  ……εντολές….
                retfie                       ; Τέλος εντολών ρουτίνας διακοπών υψηλής προτεραιότητας************
Int_sr_lp       ……..                         ; Αρχή ρουτίνας διακοπών χαμηλής προτεραιότητας ******************
                ……..                         ;……..εντολές…….
                retfie                       ; Τέλος ρουτίνας διακοπών χαμηλής προτεραιότητας*******************
init            ……..                         ;Αρχή εντολών της ρουτίνας αρχικών ρυθμίσεων ***********************
                ……..                         ; ……εντολές….
                return                       ;Τέλος εντολών της ρουτίνας αρχικών ρυθμίσεων ***********************
```

Το πρόγραμμα αποτελείται από 4 ενότητες: 1. Κύριο πρόγραμμα, 2. Ρουτίνα διακοπής υψηλής προτεραιότητας, 3. Ρουτίνα διακοπής χαμηλής προτεραιότητας, 4. Ρουτίνα αρχικών ρυθμίσεων.

Ρουτίνα διακοπών Rb (από αλλαγή κατάστασης στους ακροδέκτες RB4, RB5, RB6, RB7)

Ανεξάρτητα από το ποιος διακόπτης αλλάξει κατάσταση εκτελείται η ρουτίνα διακοπής





Καταχωρητές για τον έλεγχο διακοπών Rb (Από αλλαγή κατάστασης στους ακροδέκτες RB4, RB5, RB6, RB7

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note 1: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

REGISTER 9-2: INTCON2: INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 4-1: RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(2)	R/W-0
IPEN	SBOREN	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

Ενεργοποίηση διακοπών από Rb

```
bsf  INTCON, RBIE        ; Να γίνει 1 το bit RBIE στον
                         ;καταχωρητή κατάστασης
                         ;INTCON
                         ; (Rb Interrupt Enable)
```

Μηδενισμός της σημαίας διακοπών από Rb

```
movf PORTB,0             ; Πρέπει πρώτα να διαβαστεί
                         ; η πόρτα B, δες Note 1.
bcf  INTCON, RBIF,       ; Να γίνει μηδέν το bit RBIF
                         ; στον καταχωρητή
                         ; κατάστασης INTCON
                         ; (Rb Interrupt Flag)
```

Διακοπές υψηλής προτεραιότητας από Rb

```
bsf  INTCON, RBIP        ; Να γίνει 1 το bit RBIP
                         ; στον καταχωρητή
                         ; κατάστασης INTCON
                         ;Rb Interrupt priority
```

Διακοπές χαμηλής προτεραιότητας από Rb

```
bsf  RCON, IPEN          ; Ενεργοποίησε τις προτεραιότητες
                         ; Interrupt Priority Enable
bsf INTCON, GIEL         ; Ενεργοποίησε τις διακοπές
                         ; χαμηλής προτεραιότητας
                         ;(General Interrupt Enable Low)
bcf  INTCON2, RBIP       ; Κάνε μηδέν το bit RBIP
                         ; Rb Interrupt Priority
                         ; Όρισε την διακοπή από Rb σαν
                         ; διακοπή χαμηλής προτεραιότητας
```

Σύστημα συναγερμού με διακοπές Rb (από τους ακροδέκτες RB4, RB5, RB6, Rb7)

Όταν αλλάξει κατάσταση ένας από τους ακροδέκτες RB4, RB5, RB6, RB7 ανάβει το LED και χτυπάει το Buzzer. Με το μπουτόν μπορούμε να σβήσουμε το LED και να σταματήσουμε το Buzzer.

Σύστημα συναγερμού με χρήση διακοπών από τους ακροδέκτες Rb (RB4, RB5, RB6, RB7)

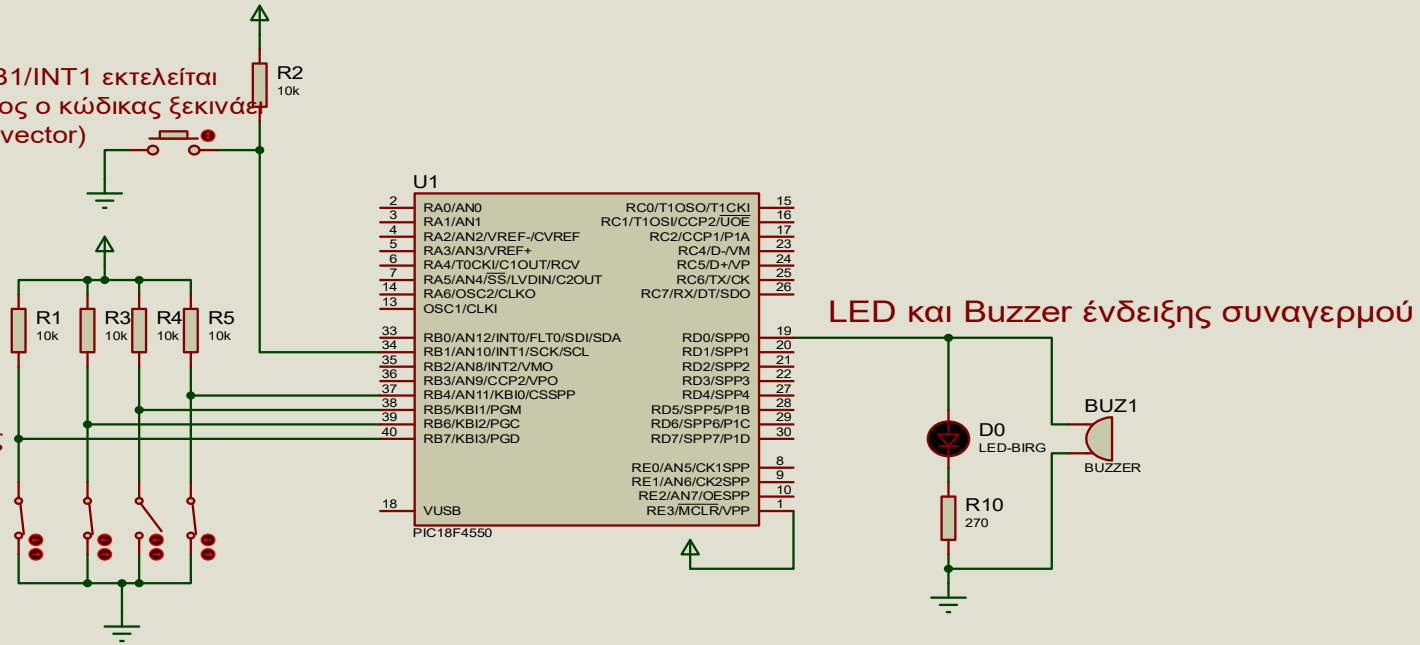
Όταν αλλάξει η κατάσταση σε έναν από τους ακροδέκτες RB4, RB5, RB6, RB7
ανάβει το LED και χτυπάει η σειρήνα

Όταν αλλάξει η κατάσταση στον ακροδέκτη RB1/INT1 εκτελείται μια διακοπή χαμηλής προτεραιότητας της οποίος ο κώδικας ξεκινάει από τη διεύθυνση 0χ018 (Low priority interrupt vector)

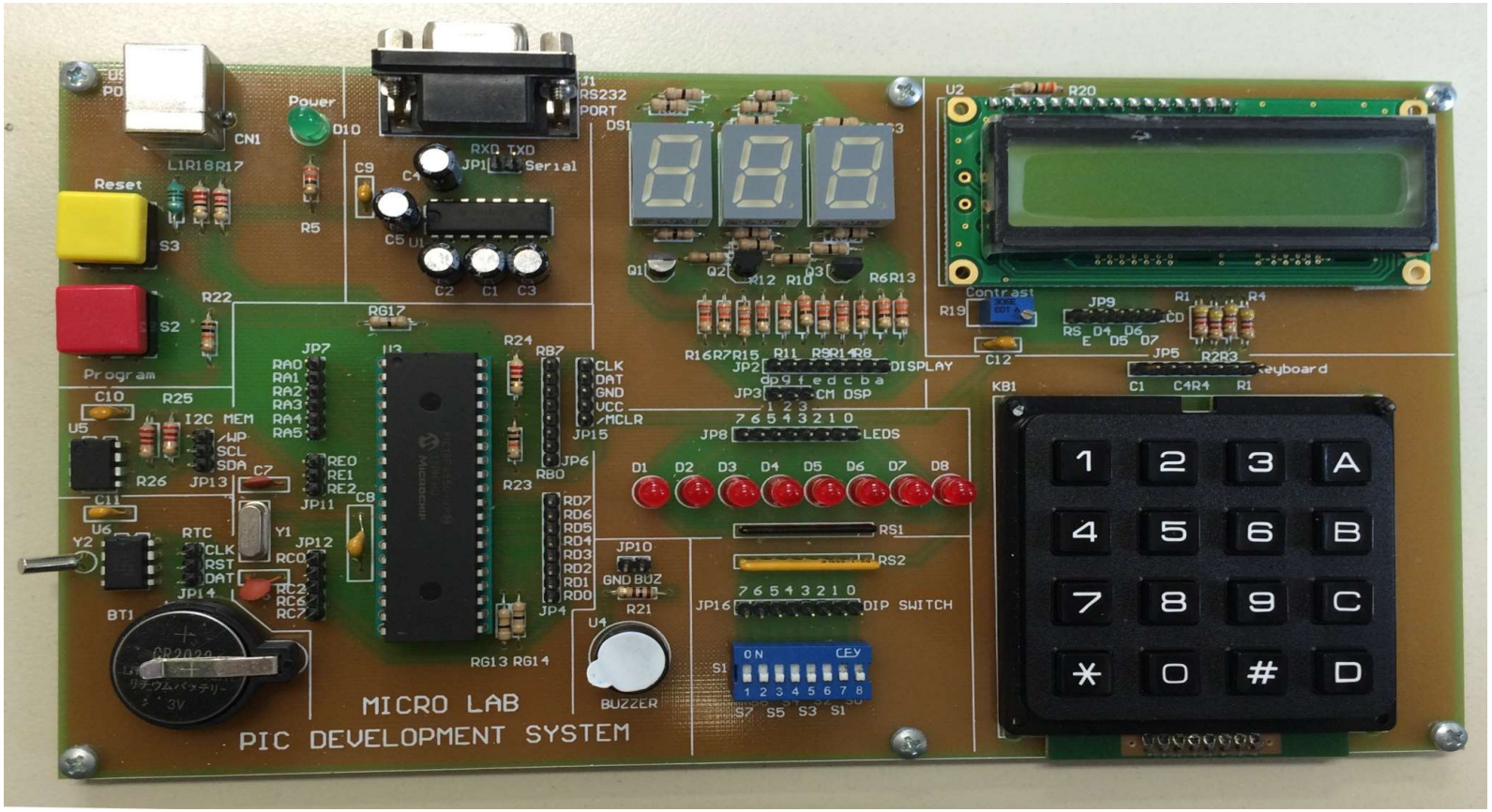
Σβήνει το LED που σημαίνει τον συναγερμό
Σταματάει το σφύριγμα της σειρήνας

Όταν αλλάξει η κατάσταση σε έναν από τους ακροδέκτες RB4, RB5, RB6, RB7 εκτελείται μια διακοπής υψηλής προτεραιότητας της οποίας ο κώδικας ξεκινάει από τη διεύθυνση 0x008 (High priority interrupt vector)

Ανάβει το LED που σημαίνει συναγερμό
Ενεργοποιεί την σειρήνα



Πλακέτα ανάπτυξης εφαρμογών

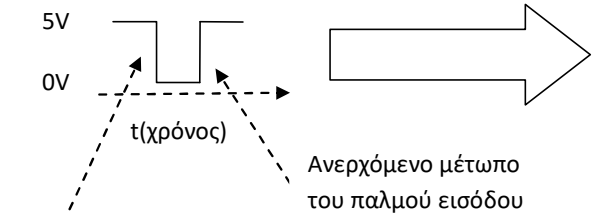


Χρονιστής
Timer

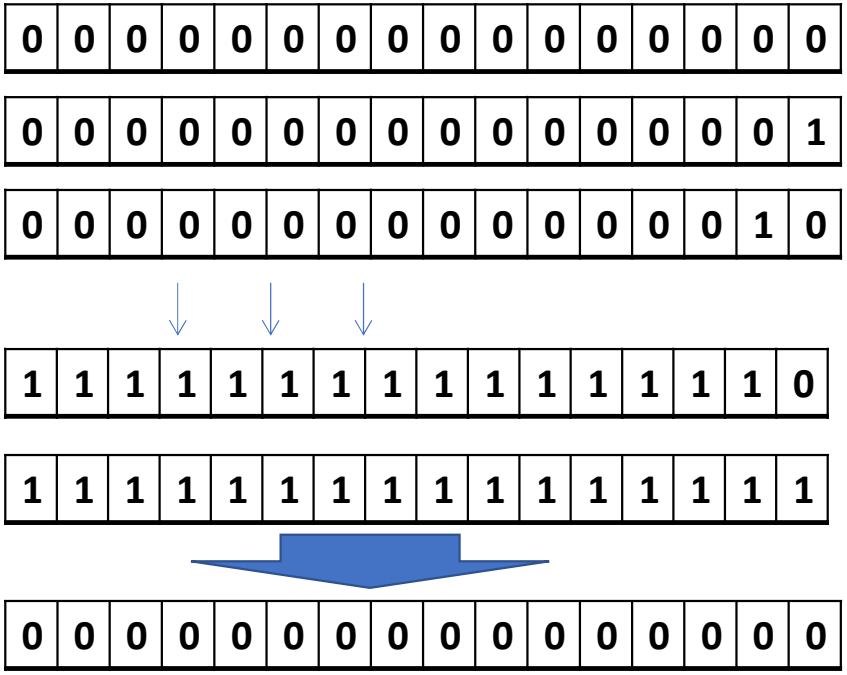
Τι είναι οι Timers(Χρονιστές) του μικροελεγκτή;

Είναι κάποιοι καταχωρητές των 8 bit ή των 16 bit των οποίων το περιεχόμενο αυξάνει ανά συγκεκριμένο χρονικό διάστημα.

Κάθε παλμός αυξάνει το περιεχόμενο του timer κατά 1



Στο πρόγραμμα μας δηλώνουμε αν θέλουμε το περιεχόμενο του timer να αυξάνεται κατά το ανερχόμενο ή το κατερχόμενο μέτωπο του παλμού εισόδου.

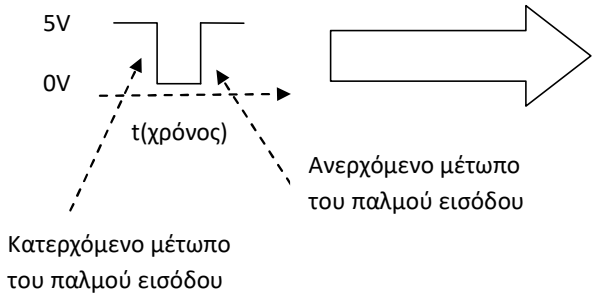


- **Προσοχή, μετά την τιμή 111 … 111 (=65535) ο timer παίρνει την τιμή 000 ... 000.**

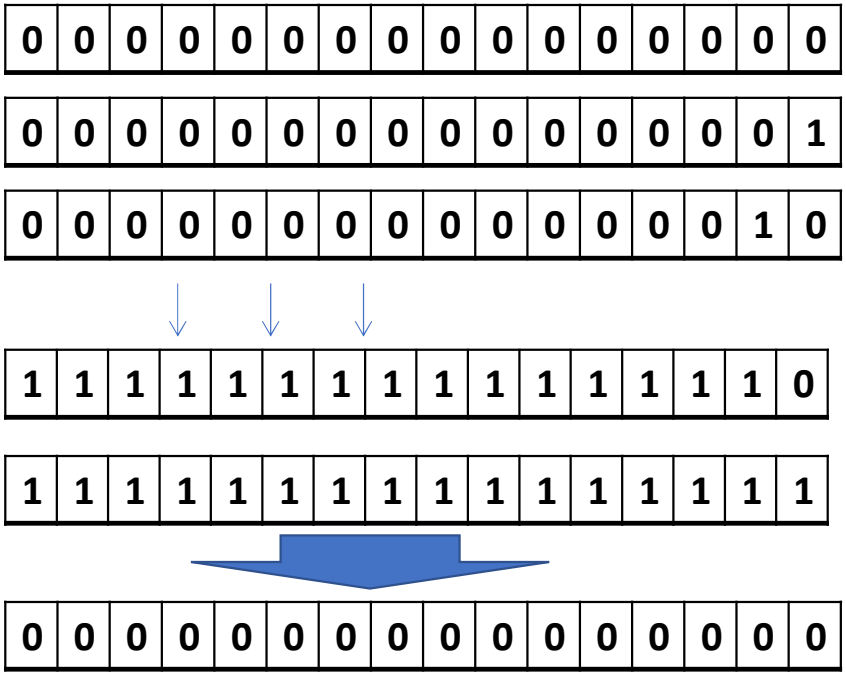
- **Σ’ αυτή την μετάβαση μπορούμε να προγραμματίσουμε να συμβαίνει μια διακοπή (interrupt).**

Χρονιστής
Timer

Κάθε παλμός αυξάνει το περιεχόμενο του timer κατά 1



Στο πρόγραμμα μας δηλώνουμε αν θέλουμε το περιεχόμενο του timer να αυξάνεται κατά το ανερχόμενο ή το κατερχόμενο μέτωπο του παλμού εισόδου.



Ερώτηση:

Από πού προέρχονται οι παλμοί που αυξάνουν την τιμή του Timer;

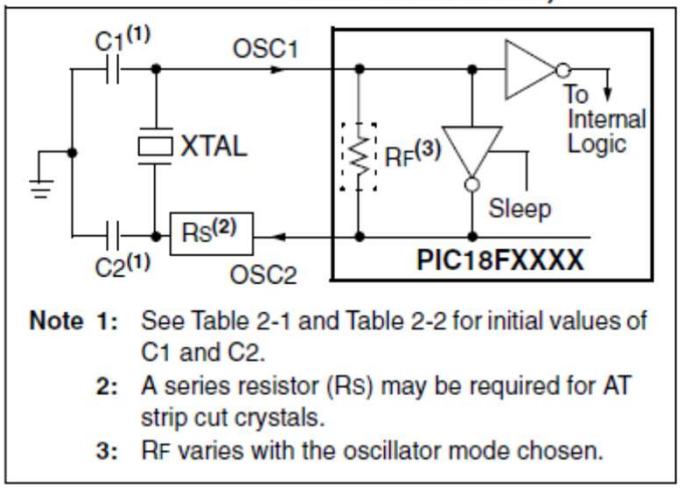
Απάντηση:

- Μπορεί να προέρχονται από μια εξωτερική πηγή και να εφαρμόζονται σε έναν συγκεκριμένο ακροδέκτη του μικροελεγκτή
- Ή να προέρχονται από ένα κύκλωμα στο εσωτερικό του μικροελεγκτή το οποίο ονομάζεται εσωτερικός ταλαντωτής(Oscillator)
- Στο πρόγραμμα μας με μια εντολή δηλώνουμε την πηγή των παλμών που θέλουμε να χρησιμοποιηθεί

Τι είναι ο εσωτερικό ταλαντωτής(internal oscillator) του μικροελεκτή;

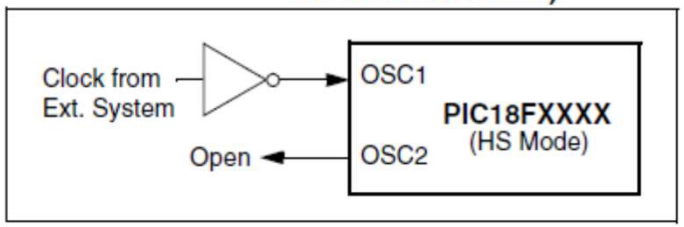
Manual PIC18F4550 σελίδα 25

FIGURE 2-2: CRYSTAL/CERAMIC RESONATOR OPERATION (XT, HS OR HSPLL CONFIGURATION)



Note 1: See Table 2-1 and Table 2-2 for initial values of C1 and C2.
2: A series resistor (RS) may be required for AT strip cut crystals.
3: RF varies with the oscillator mode chosen.

FIGURE 2-3: EXTERNAL CLOCK INPUT OPERATION (HS OSC CONFIGURATION)

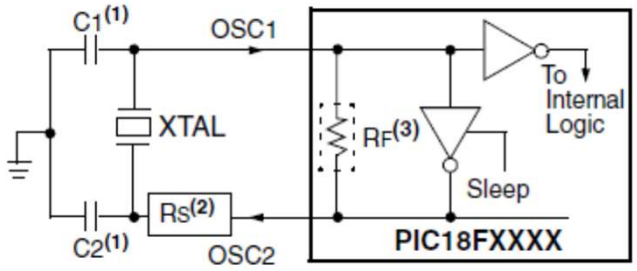


Μπορεί να χρησιμοποιηθεί και εξωτερικός ταλαντωτής για την παραγωγή παλμών.
Clock from External System

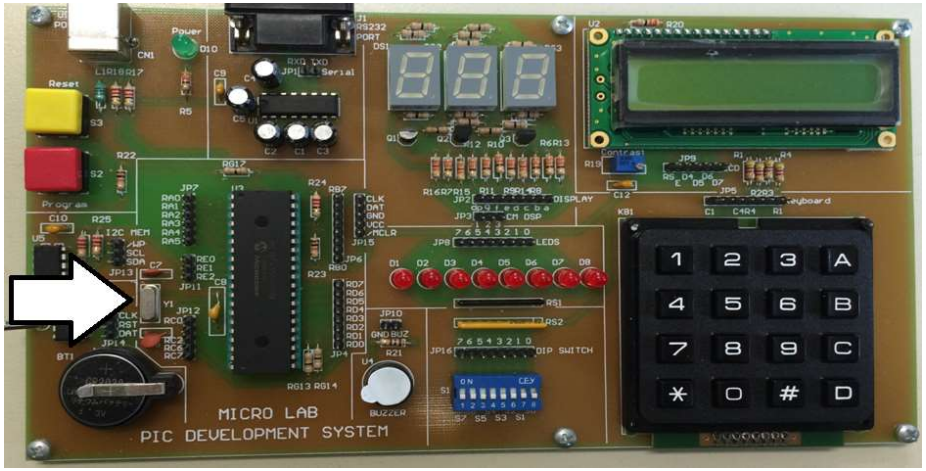
Ο ταλαντωτής είναι κύκλωμα στο εσωτερικό του μικροελεγκτή οποίο παράγει τετραγωνικούς παλμούς οι οποίοι χρησιμοποιούνται για την λειτουργία των λογικών κυκλωμάτων του μικροελεγκτή.
Το κύκλωμα αυτό χρησιμοποιεί ένα εξάρτημα έξω από τον μικροελεγκτή το οποίο λέγεται κρύσταλλος (XTAL).

Τι είναι ο κρύσταλλος(quartz) που χρησιμοποιείται στον ταλαντωτή του μικροελεγκτή;

FIGURE 2-2: CRYSTAL/CERAMIC RESONATOR OPERATION (XT, HS OR HSPLL CONFIGURATION)



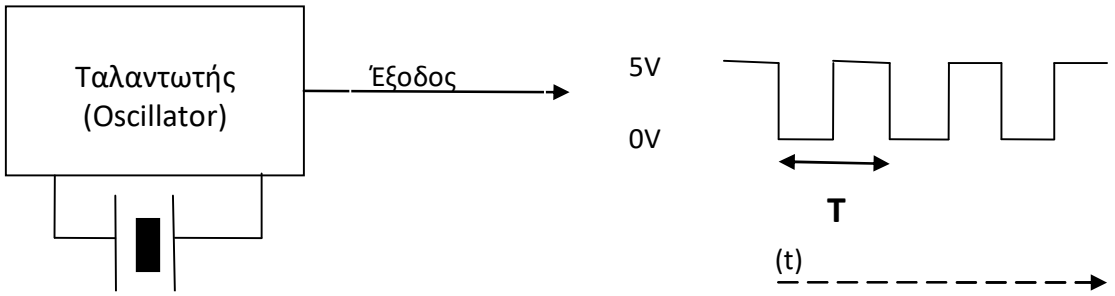
Note 1: See Table 2-1 and Table 2-2 for initial values of C1 and C2.
2: A series resistor (RS) may be required for AT strip cut crystals.
3: RF varies with the oscillator mode chosen.



Το βέλος δείχνει τον εξωτερικό κρύσταλλο(quarz). Δίπλα στον κρύσταλλο φαίνονται σαν φακές οι πυκνωτές C1 και C2

Ο κρύσταλλος(quartz) είναι ηλεκτρονικό εξάρτημα το οποίο χρησιμοποιείται σε κάποιο κύκλωμα το οποίο λέγεται ταλαντωτής κρυστάλλου.
Ο ταλαντωτής κρυστάλλου παράγει παλμούς εξαιρετικά σταθερής συχνότητας. Π.χ. συχνότητας F=20,237123 MHz.
Χωρίς κρύσταλλο(quartz) δεν μπορεί να παραχθεί σταθεροποιημένη συχνότητα.
Οι ταλαντωτές κρυστάλλου χρησιμοποιούνται και σε όλα τα ηλεκτρονικά ρολόγια. Αποτελούν την λεγόμενη πηγή χρόνου.

Τι είναι ο ταλαντωτής κρυστάλλου



Κρύσταλλος
(Quartz)
Ο κρύσταλλος χαρακτηρίζεται
από τη συχνότητα του
Π.χ f=20 MHz

$$f = \frac{1}{T}$$

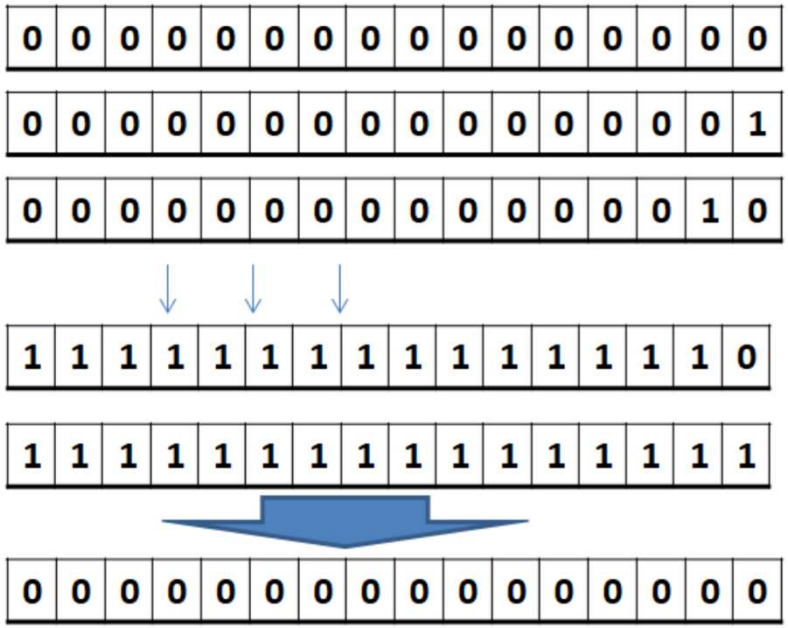
•Ο ταλαντωτής είναι ένα κύκλωμα το οποίο παράγει μια περιοδική κυματομορφή, όπως για παράδειγμα του τετραγωνικούς παλμούς τάσης του σχήματος.

•Η τετραγωνική αυτή περιοδική κυματομορφή χρησιμοποιείται στους μικροεπεξεργαστές για το συγχρονισμό της λειτουργίας των λογικών κυκλωμάτων τους.

•Η έννοια του ταλαντωτή είναι γενική και περιλαμβάνει και τα κυκλώματα τα οποία παράγουν ημιτονοειδείς κυματομορφές τάσης όπως αυτές που χρησιμοποιούνται στις τηλεπικοινωνίες.

Που χρησιμοποιούνται οι timers(ρολόγια) του μικροελεγκτή;

1. Για να μετράμε χρόνο.
2. Για να προγραμματίσουμε σε μια χρονική στιγμή ο μικρολεγκτής να εκτελέσει μια συγκεκριμένη λειτουργία
3. Για να προγραμματίσουμε ανά ορισμένα χρονικά διαστήματα ο μικροελεγκτής να εκτελεί μια συγκεκριμένη λειτουργία, π.χ. να μετράει και να καταγράφει τη θερμοκρασία κάθε 10 sec.



•Μπορούμε να προγραμματίσουμε ώστε σε κάθε μετάβαση από 111…111 σε 000…000 να εκτελείται μια ρουτίνα διακοπής από τον Timer (Timer Interrupt)

•Αυτή η διακοπή είναι μια εσωτερική διακοπή(Internal Interrupt) γιατί προκαλείται από το εσωτερικό του μικροελεγκτή και όχι από ένα σήμα που εφαρμόζεται σε έναν εξωτερικό ακροδέκτη του μικροελεγκτή.

Πόσα ρολόγια(timers) έχει ο Μικρολεγκτής PIC 18F4550;

Timer0 module
Timer1 module
Timer2 module
Timer 3 module

Ο καθένας από αυτούς του τέσσερις Timers έχει διαφορετικά υποσυστήματα για ειδικές εφαρμογές.

Timer0 module

module → υποσύστημα, ενότητα

Ο Timer0 είναι ένας timer των 8 bit ή των 16 bit.

Με μια εντολή ρυθμίζουμε πως θα χρησιμοποιηθεί ο timer0. Εάν θέλουμε να χρησιμοποιηθεί σαν timer των 8 bit θα πρέπει να συμπληρώσουμε τη σχετική παράμετρο στην εντολή ρύθμισης του timer0

Καταχωρητής ελέγχου διακοπών από τον Timer0

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT:** Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA:** Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0:** Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Παράδειγμα ρύθμισης: Να ρυθμιστεί ο Timer0 να είναι 8 bit.

```
bsf T0CON, T08BIT          ; Κάνε 1 το bit T08BIT στον καταχωρητή ελέγχου T0CON
```

Άλλοι καταχωρητές που σχετίζονται με τον Timer0

TABLE 11-1: REGISTERS ASSOCIATED WITH TIMER0

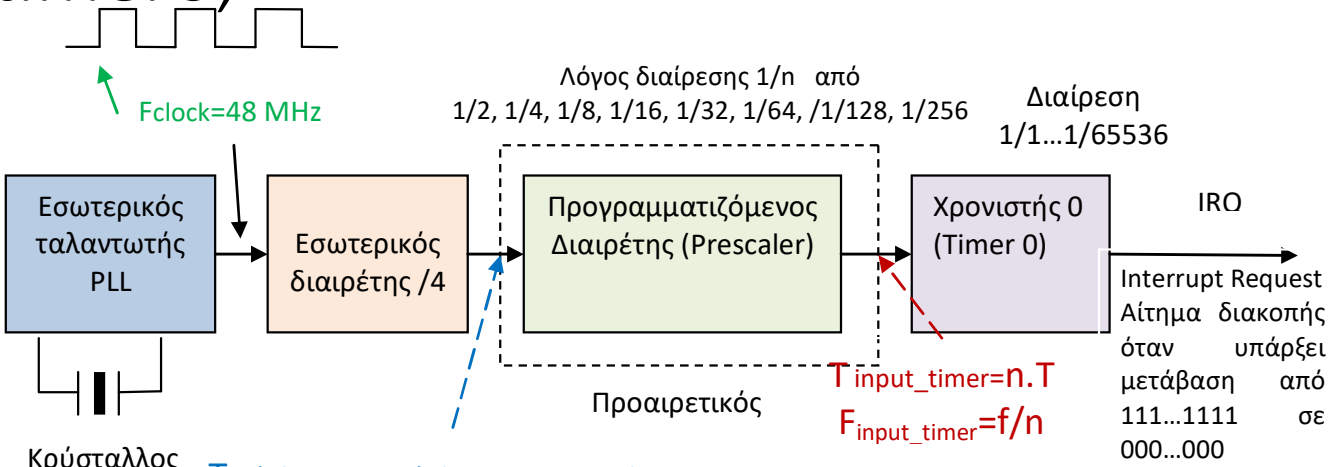
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register Low Byte								52
TMR0H	Timer0 Register High Byte								52
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	51
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	51
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	52
TRISA	—	TRISA6(1)	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	54

Legend: — = unimplemented locations, read as '0'. Shaded cells are not used by Timer0.

Note 1: RA6 is configured as a port pin based on various primary oscillator modes. When the port pin is disabled, all of the associated bits read '0'.

Τ κ ύ κ λ ο ς μ η χ α ν ή ς (M a c h i n e C y c l e)

Με τι ρυθμό αυξάνεται το περιεχόμενο του timer0;



T κύκλος μηχανής(Machine Cycle)

$$T = \frac{1}{12MHz} = \frac{1}{12x10^6 Hz} = 0{,}08333x10^{-6} = 83{,}33x10^{-9} = 83{,}33\ ns$$

Ο κύκλος μηχανής MC είναι 83,33 ns MC → Machine Cycle

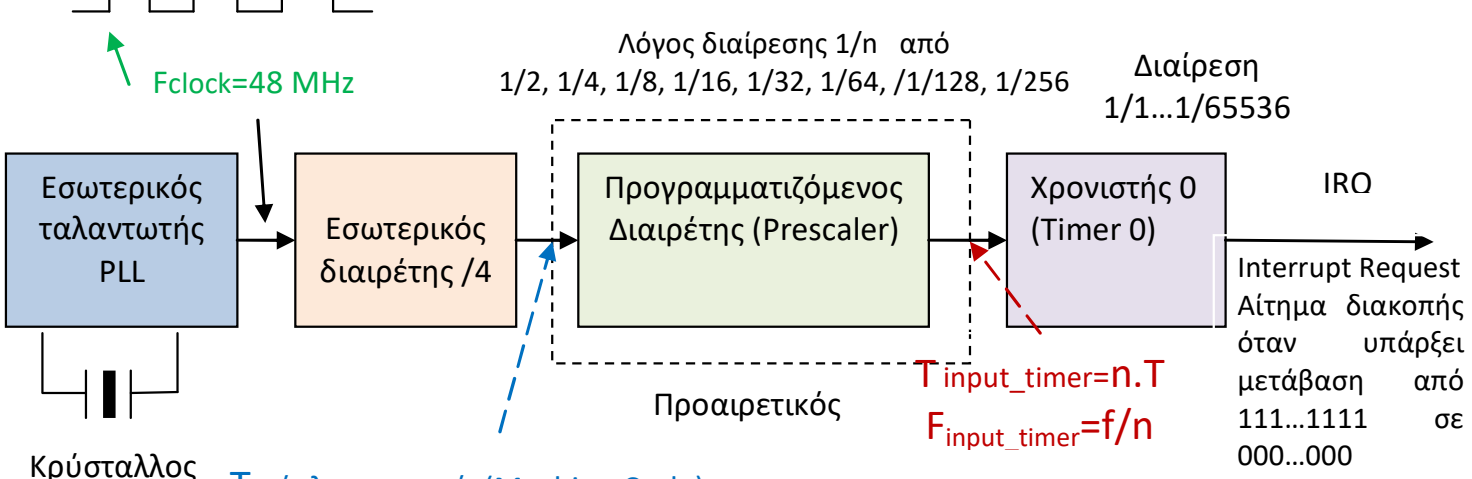
$f = \frac{1}{T}$ =12 MHz συχνότητα στην είσοδο του προγραμματιζόμενου διαιρέτη (prescaler)

Όταν δεν χρησιμοποιείται ο προγραμματιζόμενος διαιρέτης ο timer0 αυξάνει κατά 1 κάθε 83,33 ns.

Όταν ο προγραμματιζόμενος διαιρέτης τεθεί στην τιμή ½ ο timer0 αυξάνει κατά 1 κάθε 2χ83,33 ns=166,66 ns.

Όταν ο προγραμματιζόμενος διαιρέτης τεθεί στην τιμή 1/16 ο timer0 αυξάνει κατά 1 κάθε 16χ83,33 ns=1333,28ns=1,33328 μs.

Πως ρυθμίζεται η τιμή του προγραμματιζόμενου διαιρέτη συχνότητας;



T κύκλος μηχανής(Machine Cycle)

$$T = \frac{1}{12MHz} = \frac{1}{12x10^6Hz} = 0{,}08333x10^{-6} = 83{,}33x10^{-9} = 83{,}33\ ns$$

Ο κύκλος μηχανής MC είναι 83,33 ns MC → Machine Cycle

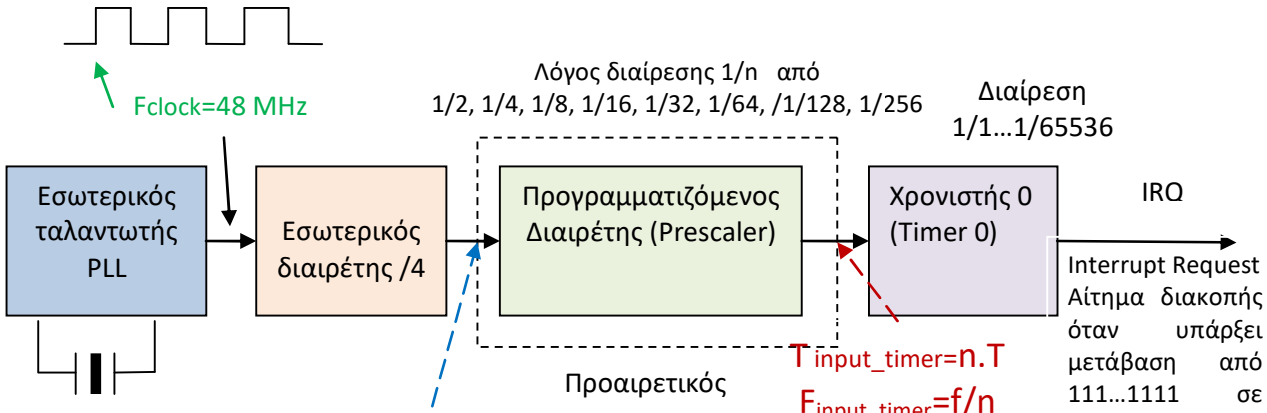
$f = \frac{1}{T}$ =12 MHz συχνότητα στην είσοδο του προγραμματιζόμενου διαιρέτη (prescaler)

Ρύθμιση του προγραμματιζόμενου διαιρέτη σε διαίρεση συχνότητας 1/256

Σύμφωνα με τον σχετικό πίνακα στο manual του PIC 18F4550, σελίδα 127 (T0CON)

```
bsf T0CON, T0PS0        ; Κάνε 1 το bit T0PS0 στον καταχωρητή T0CON
bsf T0CON, T0PS1        ; Κάνε 1 το bit T0PS1 στον καταχωρητή T0CON
bsf T0CON, T0PS2        ; Κάνε 1 το bit T0PS2 στον καταχωρητή T0CON
```

Ανά πόσο χρόνο θα συμβαίνουν διακοπές αν θέσουμε τον προγραμματιζόμενο διαιρέτη συχνότητας στην τιμή 1 /256;



T κύκλος μηχανής(Machine Cycle)

$$T = \frac{1}{12MHz} = \frac{1}{12x10^6 Hz} = 0{,}08333x10^{-6} = 83{,}33x10^{-9} = 83{,}33\ ns$$

Ο κύκλος μηχανής MC είναι 83,33 ns MC → Machine Cycle

$f = \frac{1}{T}$ =12 MHz συχνότητα στην είσοδο του προγραμματιζόμενου διαιρέτη (prescaler)

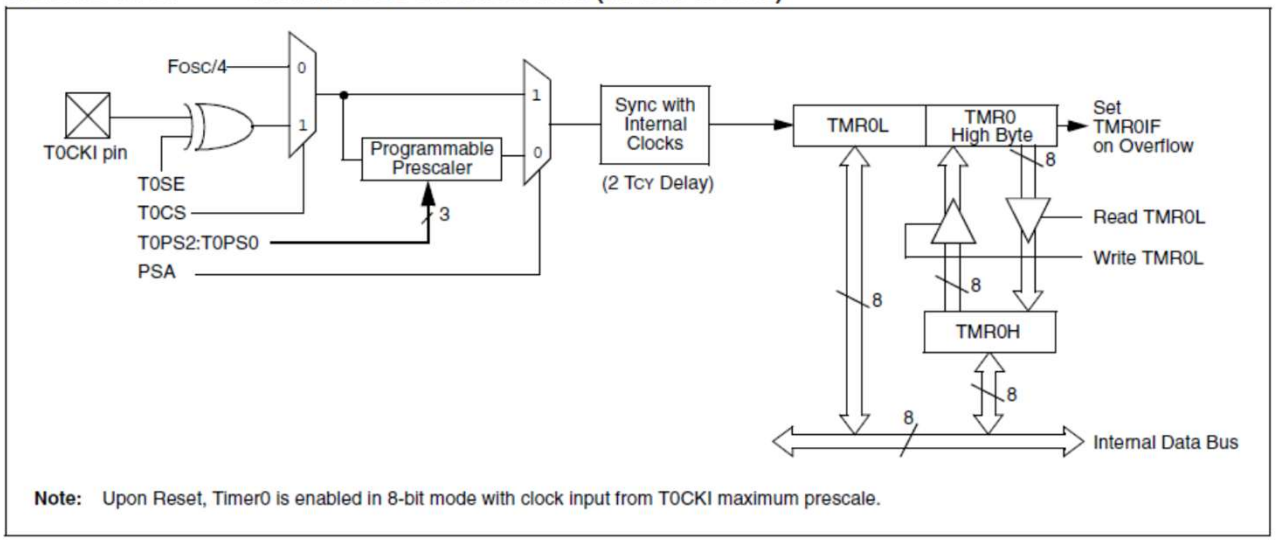
Θα συμβαίνουν διακοπές κάθε: 83,33ns X 256X65536 = 1398045409ns=1,398s

Πως ενεργοποιούμε τις διακοπές από τον Timer0; Απάντηση στο manual του PIC 18F4550

```
bsf INTCON, GIE        ; Ενεργοποίηση όλων των διακοπών (General Interrupt Enable)
bsf INTCON, TMR0IE     ; Ενεργοποίηση των διακοπών από τον Timer0
```

Προτοποθέτηση αρχικής τιμής στον Timer0 ώστε να μην ξεκινάει από την αρχική τιμή 0000 0000 0000 0000

FIGURE 11-2: TIMER0 BLOCK DIAGRAM (16-BIT MODE)



Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI maximum prescale.

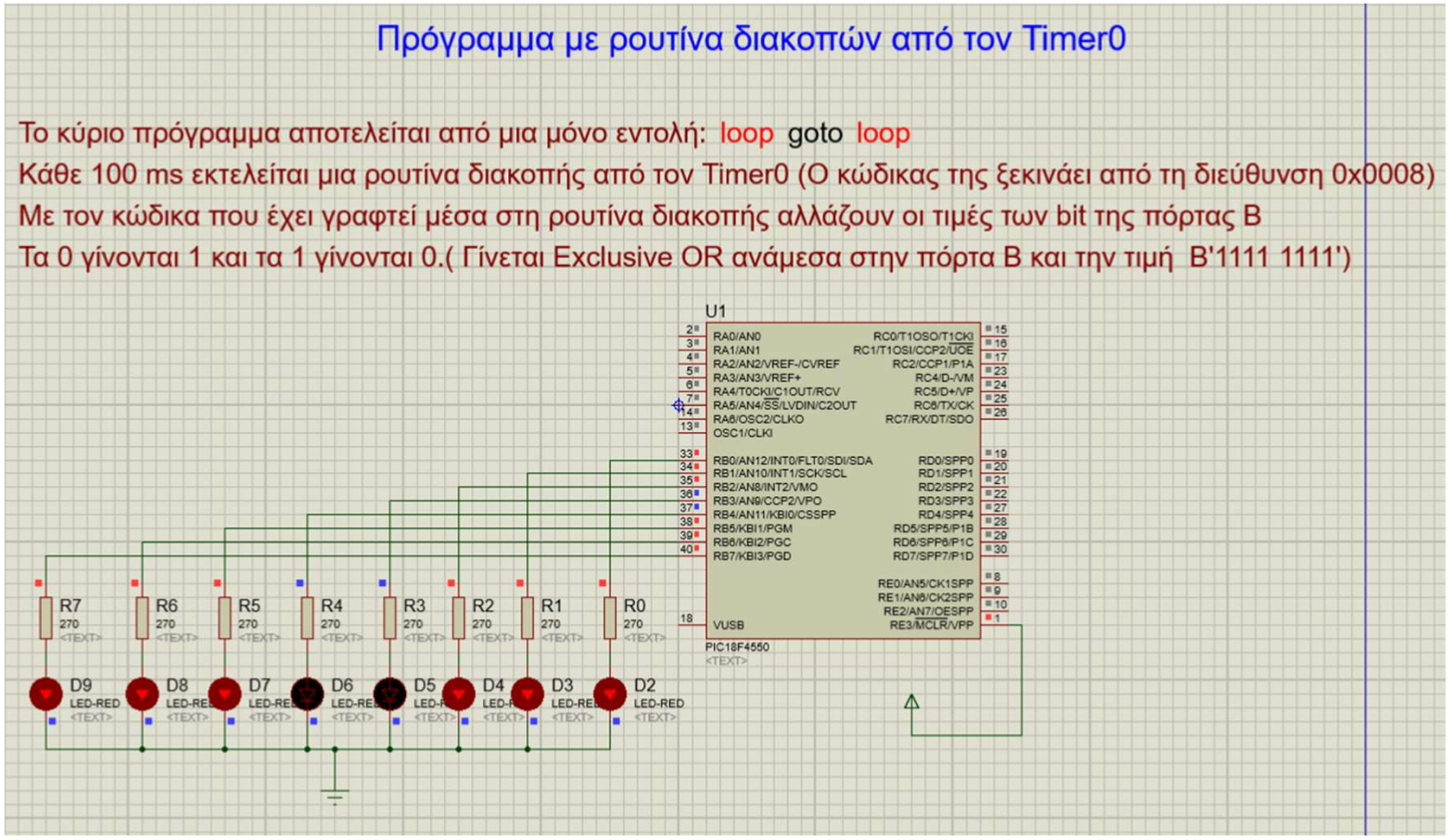
Με τις 4 παρακάτω εντολές δίνεται αρχική τιμή 6D82h στον Timer0

```
movlw 0x6D      ; εγγράφεται η τιμή (6D)h στον w
movwf TMR0H     ; μεταφέρεται η τιμή (6D)h στο υψηλό byte του Timer0
movlw 0x82      ; εγγράφεται η τιμή (82)h στον  w
movwf TMR0L     ; μεταφέρεται η τιμή (82)h στο χαμηλό byte του Timer0
                ; προσοχή και τα δύο byte του timer0 ενημερώνονται
                ;συγχρόνως με την εγγραφή του
                ; χαμηλού byte του timer0. Σελίδα 126 του manual
                ;του PIC 18F4550
```

Δομή προγράμματος με ρουτίνα διακοπών από τον χρονιστή Timer0

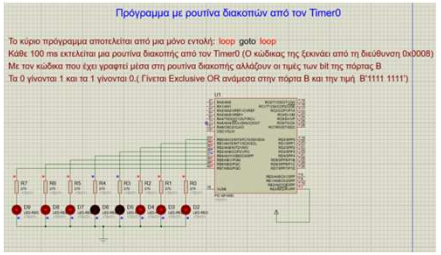
```
                    #include <set_fuse.inc>     ; Αρχείο που περιλαμβάνει τις θέσεις των
                                                ;καταχωρητών κατεύθυνσης και δεδομένων
                                                ; των πορτών καθώς και των καταχωρητών κατάστασης
                                                ;και ελέγχου των διακοπών.
                                                ; Το αρχείο αυτό περιλαμβάνει και άλλες ρυθμίσεις
                                                ;του μικροελεγκτή.
                    org 0x000                   ; Δηλώνουμε την διεύθυνση στη μνήμη
                                                ;προγράμματος απ' όπου θα
                                                ; ξεκινήσει να γράφεται ο κώδικας του προγράμματος
                     call  init                 ; Καλείται η ρουτίνα που περιλαμβάνει τις αρχικές
                                                ;ρυθμίσεις του
                                                ; μικροελεγκτή. Η ρουτίνα αυτή είναι γραμμένη
                                                ;στη διεύθυνση
                                                ; που αντιστοιχεί στην ετικέτα init.
                    goto  main                  ; Γίνεται μετάβαση στη διεύθυνση που αντιστοιχεί
                                                ;στην ετικέτα main
                                                ; όπου γράφεται το κύριο πρόγραμμα
                    org 0x008                   ; Στη διεύθυνση 0x008 γράφεται η ρουτίνα διακοπής
                    goto  timer0_sr             ; Γίνεται μετάβαση στην ετικέτα timer0_sr που
                                                ;αντιστοιχεί στη διεύθυνση όπου γράφεται
                                                ; η ρουτίνα διακοπής
main                ……..                        ;Αρχή εντολών κύριου προγράμματος **********
                    ……..                        ; ……εντολές…..
                    ……..                        ; Τέλος εντολών κύριου προγράμματος**********
timer0_sr           ……..                        ; Αρχή εντολών ρουτίνας διακοπών     **********
                    ……..                        ;  ……εντολές….
                    retfie                      ; Τέλος εντολών ρουτίνας διακοπών από τον timer0
init                movlw '10100000'            ; Ρυθμίσεις στον καταχωρητή ελέγχου INTCON που
                    movwf INTCON                ; αφορούν τον Timer0
                    movlw '10000XXX'            ; Ρυθμίσεις στον καταχωρητή ελέγχου T0CON που
                    movwf T0CON                 ; Αφορούν τον Timer0
                    ……..                        ; Υπόλοιπες εντολές αρχικών ρυθμίσεων
                    return                      ;Τέλος εντολών της ρουτίνας αρχικών ρυθμίσεων *****
                    END
```

Αναβόσβημα LED με ρουτίνα διακοπών από τον Timer0 (Την έχουμε ρυθμίσει να εκτελείται κάθε 100 ms)



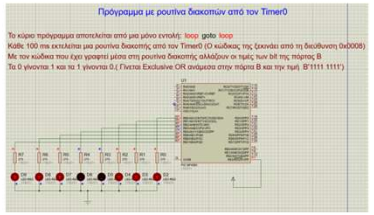
Πρόγραμμα με ρουτίνα διακοπής από timer0 (1)

```
;Εάν η τιμή του προγραμματιζόμενου διαιρέτη συχνότητας είναι 1/32 να υπολογισθεί η
;αρχική τιμή που θα πρέπει να δίνεται στον Timer0(σε λειτουργία 16bit) ώστε να
;προκαλούνται διακοπές κάθε 100ms.

;Στη συνέχεια να γραφεί πρόγραμμα με το οποίο αντιστρέφεται η κατάσταση της πόρτας B
;ή οποία χρησιμοποιείται σαν έξοδος κάθε 100ms. Η αρχική τιμή της πόρτας B να είναι: 11100111


        #include <set_fuse.inc>
        org 0x0000

        call init           ; Κλήση της ρουτίνας αρχικοποίησης
                            ; call initialize
loop    goto loop           ; το κύριο πρόγραμμα δεν κάνει τίποτα, εκτελεί μόνο αυτόν τον βρόχο

        org 0x0008          ; Σε αυτή τη διεύθυνση μεταβαίνει το πρόγραμμα όταν γίνει μια
                            ; διακοπή υψηλής προτεραιότητας όπως από τον ακροδέκτη RB0/INT0
                            ; η από τον Timer0. Στη διεύθυνση αυτή γράφουμε την εντολή για την
                            ; κλήση της υπορρουτίνας.
        goto timer0_isr     ; μετάβαση στην ρουτίνα εξυπηρέτησης διακοπής από τον Timer0
```



Πρόγραμμα με ρουτίνα διακοπής από timer0 (2)

```
;**********************Αρχή ρουτίνας εξυπηρέτησης διακοπής από τον timer0

timer0_isr                  ; ετικέτα όπου είναι γραμμένο το κύριο πρόγραμμα
; Με τις 4 παρακάτω εντολές δίνεται αρχική τιμή (6D82)h  στον Timer0 ώστε
; να γεμίσει μετά από 100ms
        ;*************
        movlw 0x6D          ; εγγράφεται η τιμή (6D)h στον w
        movwf TMR0H         ; μεταφέρεται η τιμή (6D)h στο υψηλό byte του Timer0
        movlw 0x82          ; εγγράφεται η τιμή (82)h στον  w
        movwf TMR0L         ; μεταφέρεται η τιμή (82)h στο χαμηλό byte του Timer0
        ;**************     ; προσοχή και τα δύο byte του timer0 ενημερώνονται συγχρόνως με την εγγραφή του
                            ; χαμηλού byte του timer0. Σελίδα 126 του manual του PIC 18F4550

        movlw B'11111111'   ;φόρτωσε στον w την τιμή 1111 1111
                            ;move literal value 1111 1111 to w

        xorwf PORTB,1       ;Κάνε αποκλειστικό ή(Exclusive OR, XOR) ανάμεσα στο περιεχόμενο του w
                            ;και το περιεχόμενο της πόρτας B
                            ; Με την πράξη αυτή αντιστρέφεται το περιεχόμενο της πόρτας B,
                            ; δηλαδή τα 0 γίνονται 1 και τα 1 γίνονται 0.
                            ; Exclusive OR between W register and PORTB. Result is stored at PORTB.

        bcf INTCON, TMR0IF  ; Μηδενίζουμε τη σημαία διακοπών του Timer0 ;
                            ;Αν δεν το κάνουμε αυτό ο μικροελεγκτής μόλις βγει θα
                            ;ξαναμπεί στην διακοπή γιατί θα δει σηκωμένη την σημαία
                            ;του μηδενός.
                            ; bit clear TMR0IF at INTCON register

        retfie              ;επιστροφή από την ρουτίνα εξυπηρέτησης της διακοπής
                            ;return from interrupt
;********************** Τέλος εξυπηρέτησης διακοπής από τον Timer0
```



Πρόγραμμα με ρουτίνα διακοπής από timer0 (3)

```
;**********************Αρχή ρουτίνας αρχικοποίησης
init                ;
        movlw B'00000000'   ; φορτώνεται στον w η τιμή 0000 0000
        movwf TRISB         ; Η πόρτα B γίνεται έξοδος(TRISB=0000 0000)

        movlw B'11100111'   ;φορτώνεται στον w η τιμή 11100111
        movwf PORTB         ;Στην πόρτα B δίνεται η αρχική τιμή 11100111

        movlw B'10100000'   ; Γίνεται ενεργοποίηση των διακοπών και
        movwf INTCON        ; μηδενίζεται η σημαία διακοπών από τον timer0
                            ; GIE=1, TMR0IE=1, TMR0IF=0.

        movlw B'10000100'   ;Ο Timer0 τίθεται σε λειτουργία 16 bit(τρία λιγότερο σημαντικά bit=110)
        movwf T0CON         ; και ο διαιρέτης σε τιμή 1:32. Ελέγξτε από τον πίνακα του T0CON.


        return              ; Επιστροφή από τη ρουτίνα αρχικοποίησης
;**********************Τέλος ρουτίνας αρχικοποίησης

        END                 ; Τέλος εντολών όλου του προγράμματος
```