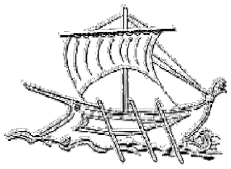


Α.Ε.Ι. ΠΕΙΡΑΙΑΤ.Τ.
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ Τ.Ε.

"ΑΣΥΡΜΑΤΟΣ ΕΛΕΓΧΟΣ ΟΙΚΙΑΚΩΝ ΣΥΣΚΕΥΩΝ ΜΕ ΜΙΚΡΟΕΛΕΓΚΤΗ ΜΕΣΩ ΜΗΝΥΜΑΤΩΝ"

Επιβλέπων Καθηγητής: Γεώργιος Ξερογιαννάκης

Σπουδαστής:
Αντύπας Θεόδωρος ΑΜ: 41923

ΑΙΓΑΛΕΩ 2017

ΠΕΡΙΕΧΟΜΕΝΑ

ΛΙΣΤΑ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Στη παρούσα πτυχιακή εργασία παρουσιάζουμε μια εφαρμογή, που ασύρματα, είναι δυνατή η ενεργοποίηση και η απενεργοποίηση οποιασδήποτε οικιακής συσκευής όπως παραδείγματος χάρη ένα κύκλωμα φωτισμού, ψυγείο, ηλεκτρική κουζίνα κ.α.. Αυτό επιτυγχάνεται στέλνοντας γραπτό μήνυμα (sms) από οποιοδήποτε απομακρυσμένο κινητό τηλέφωνο. Η εφαρμογή κατασκευάστηκε χρησιμοποιώντας μία πλακέτα με ενσωματωμένο έναν μικροελεγκτή PIC18F4520 της Microchip , ένα Module με οθόνη LCD για να δείχνει διάφορα μηνύματα προς τον χρήστη , ένα GSM Module και λίγες φωτοδιόδους LED ως προσομοίωση ρελέ για εναλλαγή εναλλασσομένων 230V (AC) φορτίων. Το GSM Module δέχεται κάρτα SIM , όπως ένα κανονικό κινητό τηλέφωνο , συνδέεται σειριακά με τη πλακέτα του μικροελεγκτή με ενσωματωμένο σύστημα έτσι ώστε να δέχεται αλλά και να στέλνει γραπτά μηνύματα.

Keywords: λέξεις κλειδιά της πτυχιακής εργασίας, λέξεις που την κατατάσσουν σε θεματικές περιοχές και καθιστούν ευκολότερη την ηλεκτρονική της αναζήτηση (στα αγγλικά)

ΕΙΣΑΓΩΓΗ

Η ραγδαία εξέλιξη της τεχνολογίας και οι εφαρμογές της τον τελευταίο αιώνα, συντέλεσαν στην πρόοδο των επιστημών και συνεπώς στη βελτίωση των συνθηκών διαβίωσης του ανθρώπου. Στη ραγδαία αυτή εξέλιξη, μεγάλη συμβολή είχε και η ανακάλυψη των ολοκληρωμένων κυκλωμάτων που διευκόλυναν τον αυτοματιστικό έλεγχο.
Αρχικά τα ολοκληρωμένα κυκλώματα κατασκευάστηκαν ως πολεμικά εργαλεία ενώ με την πάροδο των χρόνων εξελίχθηκαν και έχουν μπει στη καθημερινότητα του ανθρώπου ενώ είναι απαραίτητα εργαλεία για κάθε είδους επιστημονική έρευνα. Είναι απαραίτητα διότι η κατασκευή τους είναι τέτοια ώστε να εξοικονομούν μέγεθος και ισχύ, άρα χρήμα και αυτό έχει ως αποτέλεσμα πάνω στην εξέλιξή τους να πραγματοποιούνται εκατομμύρια έρευνες.
Ένα παράδειγμα ολοκληρωμένου, που με αυτό θα ασχοληθούμε, είναι ο μικροελεγκτής έχοντας αμέτρητες εφαρμογές πάνω στον αυτοματισμό και τη ρομποτική. Ανάμεσα σε αυτές τις εφαρμογές είναι κι εκείνες που αφορούν στην αυτοματοποίηση της λειτουργίας των σπιτιών.
Το τρανό παράδειγμα είναι η σχεδίαση ενός "έξυπνου σπιτιού" που στην ουσία πρόκειται για ένα ψηφιακό σπίτι όπου με τον κατάλληλο προγραμματισμό του ολοκληρωμένου(π.χ. μικροελεγκτής, PLC κλπ) , είναι δυνατό να γίνει πλήρης έλεγχος όλων των ηλεκτρικών κυκλωμάτων του σπιτιού, δηλαδή από το απλούστερο παράδειγμα για το αν είναι ανοιχτή η πόρτα μέχρι το πιο πολυσύνθετο.
Εν κατακλείδι, η τεχνολογία έχει διευκολύνει την καθημερινότητα του ανθρώπου ,όμως είναι απαραίτητο το μέτρο πάνω στη χρήση των εφαρμογών διότι τα αποτελέσματά τους δεν θα είναι τα αναμενόμενα και στη πραγματικότητα θα βλάψουν την ανθρώπινη ζωή.

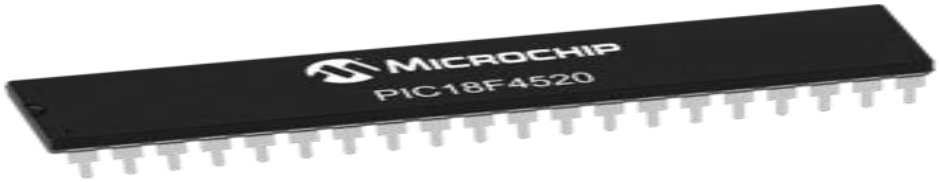
Λέξεις κλειδιά: λέξεις που χαρακτηρίζουν το αντικείμενο της πτυχιακής εργασίας και κάνουν την ηλεκτρονική αναζήτησή της πιο εύκολη.

1ΟΚΕΦΑΛΑΙΟ

“ ΜΙΚΡΟΕΛΕΓΚΤΕΣ PIC”

1.1 Εισαγωγή στους μικροελεγκτές PIC

Όλοι οι μικροελεγκτές PIC της εταιρίας Microchip έχουν αρχιτεκτονική RISC (Reduced Unstruction Set Code), με ξεχωριστή μνήμη εντολών και δεδομένων και επιπλέον με ξεχωριστούς διαύλους (Harvard αρχιτεκτονική) προσπέλασης αυτών των μνημών εντολών. Οι PIC διαθέτουν : μνήμη RAM, μνήμη EEPROM, μνήμη ROM και θύρες I/O (Input/Output). Αναλόγως την εφαρμογή που επιλέγεται ένας μικροελεγκτής PIC, υπάρχει και ο κατάλληλος καθώς η γκάμα τους είναι τεράστια, και διαθέτουν πολλά περισσότερα περιφερειακά εκτός από μνήμες και I/O, όπως κρύσταλλο 4MHz, κύκλωμα reset, δυνατότητα οδήγησης εξωτερικών συσκευών από τις θύρες I/O με ρεύμα τάξεως 20mA, programming, μνήμη Flash κ.α..



1.1.1 Οικογένειες των μικροελεγκτών PIC

Οι μικροελεγκτές PIC συνοψίζοντας και ομαδοποιώντας τους,
χωρίζονται σε πέντε κατηγορίες αναλόγως τα χαρακτηριστικά τους.

1. **PIC12CXXX / PIC12FXXX** (8-bits)
2. **PIC16C5X** (12-bits)
3. **PIC16CXXX / PIC16FXXX** (14-bits)
4. **PIC17CXXX** (16-bits)
5. **PIC18CXXX / PIC18FXXX** (improved 16-bits)

1.1.1.1 Σύντομη ανάπτυξη μικροελεγκτών PIC

1. PIC12CXXX / PIC12FXXX

Η οικογένεια αυτή έχει μικροελεγκτές με χαμηλό κόστος, έχουν 8 ακίδες, λειτουργούν και σε τάση 2,5 V, διαθέτουν μνήμες προγράμματος FLASH,OTP,ROM, και μνήμες δεδομένων RAM 64 bytes και EEPROM 128 bytes.

2. PIC16C5X

Η οικογένεια αυτή έχει μικροελεγκτές με μνήμη προγράμματος OTP, διατίθενται σε συσκευασίες των 14,18,20 και 28 ακίδων, λειτουργούν και σε τάση 2V ενώ PIC16HV5XX λειτουργεί και σε τάση έως 15V σε σύνδεση απευθείας με μπαταρία.

3. PIC16CXXX / PIC16FXXX

Οι μικροελεγκτές αυτοί διατίθενται σε συσκευασίας από 18 μέχρι 68 ακίδες.

4. PIC17CXXX

Το μόνο επιπλέον από τις προηγούμενες οικογένειες είναι ότι διαθέτουν , συγκριτικά, βελτιωμένες εντολές στο προγραμματισμό τους.

5. PIC18CXXX / PIC18FXXX

Οι μικροελεγκτές αυτοί αποτελούν μία οικογένεια υψηλής απόδοσης CMOS με ενσωματωμένο μεταροπέα A/D και 16-ψήφιο μήκος εντολής. Διαθέτουν 32 επιπέδων σωρό

(συγκριτικά με άλλους μικρότερους που διαθέτουν 8 επιπέδων) καθώς και πολλαπλές πηγές πρόκλησης εσωτερικών και εξωτερικών διακοπών. Τέλος, λόγω των επιπρόσθετων καταχωρητών που διαθέτουν φτάνουν την ταχύτητα εκτέλεσης εντολών των 10 MIPS (Million Instructions Per Second).

1.1.1.2 Δομή του μικροελεγκτή PIC

Η δομή του μικροελεγκτή PIC μπορεί να χωριστεί σε δύο μέρη, τον πυρήνα (core) και τις περιφερειακές μονάδες του (peripheral units). Ο πυρήνας του μικροελεγκτή αποτελείται από όλα εκείνα τα στοιχεία, τα οποία είναι απολύτως απαραίτητα για την λειτουργία του. Οι περιφερειακές μονάδες βρίσκονται πάντα ενσωματωμένες στον μικροελεγκτή και είναι αυτές που τον κάνουν να διαφέρει από έναν μικροεπεξεργαστή.

Στον πυρήνα του PIC ανήκουν :

1. Κεντρική μονάδα επεξεργασίας
2. Μνήμη
3. Εντολές
4. Λειτουργίες διακοπών

Εδώ αξίζει να σημειωθεί ότι , λόγω της μεγάλης σημαντικότητάς τους, έχουμε συμπεριλάβει τις εντολές στον πυρήνα του PIC, παρά το γεγονός ότι πρόκειται για κάποιο λογικό παρά υλικό στοιχείο του μικροελεγκτή.

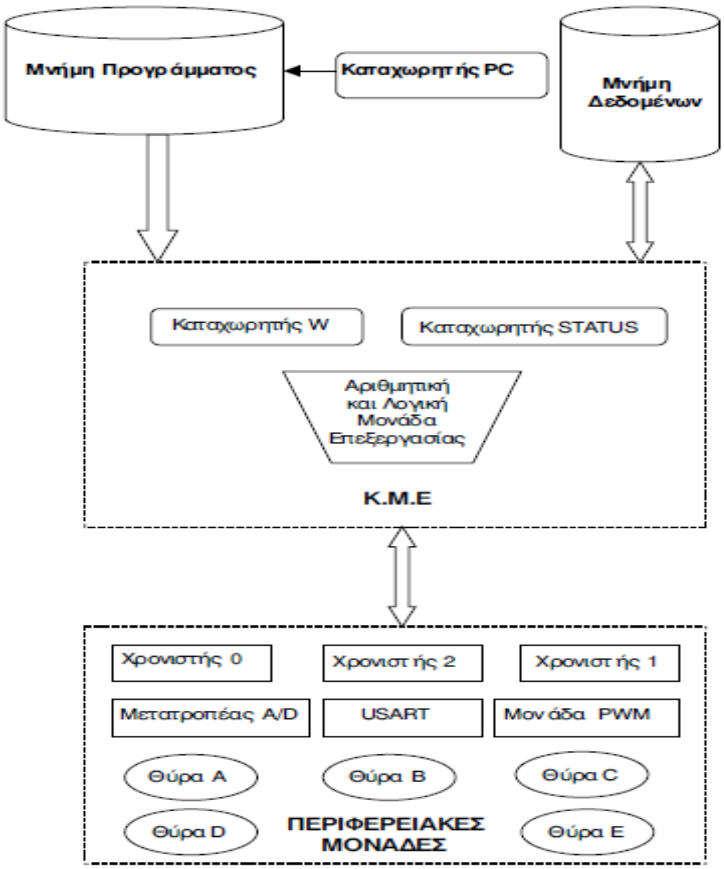
Στις περιφερειακές μονάδες ανήκουν :

1. Οι θύρες εισόδου/εξόδου γενικής χρήσης
2. Οι μετρητές χρόνου
3. Η μονάδα διαμόρφωσης πλάτους
4. Οι θύρες σειριακής επικοινωνίας
5. Η μονάδα παραγωγής τάσης αναφοράς
6. Οι συγκριτές
7. Ο μετατροπέας αναλογικού σήματος σε ψηφιακό

Κεντρική μονάδα επεξεργασίας

Η κεντρική μονάδα επεξεργασίας (CPU) εκτελεί τις εντολές του προγράμματος που έχει αποθηκευτεί στη μνήμη προγράμματος. Από τη μνήμη αυτή, η κεντρική μονάδα επεξεργασίας φέρνει, με τη σειρά, τις εντολές του προγράμματος , τις αποκωδικοποιεί και μετέπειτα τις εκτελεί. Πρέπει να σημειωθεί, ότι ο PIC αναγνωρίζει (35) εντολές προγραμματισμού.

Εντός της κεντρικής μονάδας επεξεργασίας, βρίσκεται και η αριθμητική και λογική μονάδα (Α.Λ.Μ.). Το σχήμα 1.2 παρουσιάζει την κεντρική μονάδα επεξεργασίας μαζί με τα άμεσα με αυτή συνδεόμενα στοιχεία του PIC. Οι αριθμητικές πράξεις που μπορεί να εκτελεί είναι η πρόσθεση και η αφαίρεση, ενώ έχει τη δυνατότητα να πραγματοποιεί και λογικές πράξεις (AND, NOT, OR, XOR, κτλ). Η μονάδα επεξεργάζεται δεδομένα μήκους 8 bit (8 δυαδικά ψηφία).

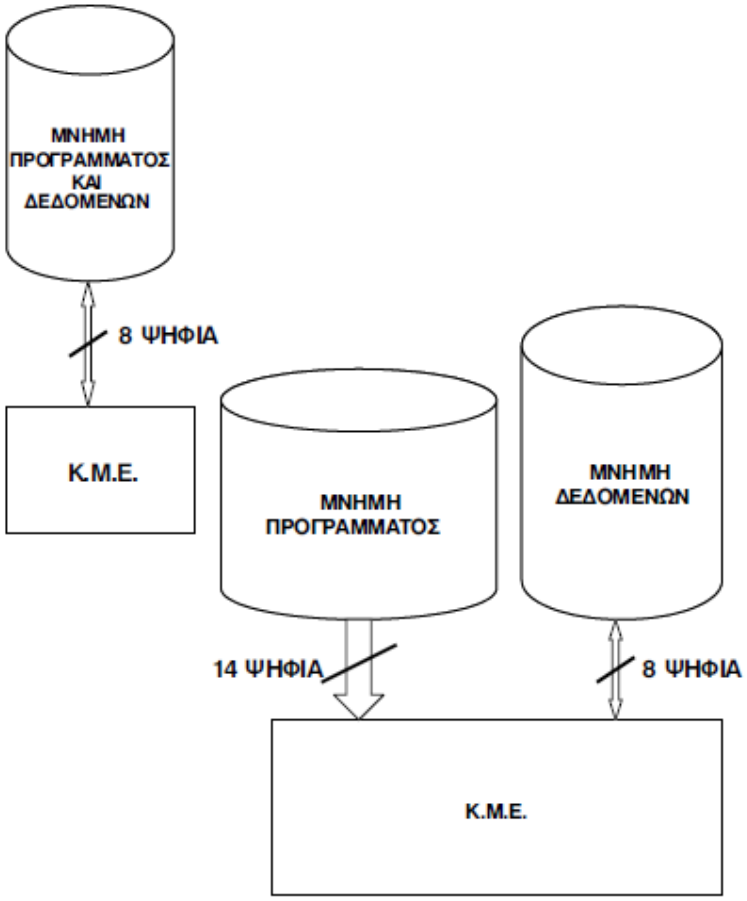


Σχήμα **Error! Use the Home tab to apply StyleHeading 1 to the text that you want to appear here.**.1 (Η δομή μικροελεγκτή PIC)

Τέλος, έχοντας ως στόχο τη σωστή ανεύρεση των εντολών που πρέπει να εκτελεστούν, η κεντρική μονάδα επεξεργασίας χρησιμοποιεί έναν μετρητή. Στην πραγματικότητα, πρόκειται για έναν καταχωρητή, οποίος περιέχει τη διεύθυνση μνήμης στην οποία βρίσκεται αποθηκευμένη η εντολή που πρέπει να εκτελεστεί. Ο καταχωρητής αυτός ονομάζεται Program Counter , γνωστός και ως (PC).

Μνήμη

Στη σχεδίαση των μικροελεγκτών χρησιμοποιούνται δύο αρχιτεκτονικές. Στη πρώτη χρησιμοποιείται μία μνήμη τόσο για την αποθήκευση του προγράμματος όσο και για την αποθήκευση των δεδομένων, ενώ στη δεύτερη δύο ξεχωριστές μνήμες. Η μία χρησιμοποιείται για την αποθήκευση του προγράμματος (Μνήμη προγράμματος), ενώ η άλλη για την αποθήκευση των δεδομένων (Μνήμη δεδομένων). Στο σχήμα 1.3 απεικονίζονται οι δύο αρχιτεκτονικές.



Σχήμα **Error! Use the Home tab to apply StyleHeading 1 to the text that you want to appear here.**.2

Ο μικροελεγκτής PIC έχει την δεύτερη αρχιτεκτονική, όπου εντολές και δεδομένα κινούνται σε ξεχωριστούς διαδρόμους με αποτέλεσμα αυτό να μπορεί να γίνει όχι μόνο με πολύ μεγαλύτερη ταχύτητα αλλά ακόμη και την ίδια χρονική στιγμή. Αντιθέτως, στην πρώτη αρχιτεκτονική, εντολές και δεδομένα μοιράζονται τον ίδιο διάδρομο με αποτέλεσμα να ελαττώνεται η ταχύτητα μεταφοράς τους. Επιπροσθέτως, το πλεονέκτημα της δεύτερης αρχιτεκτονικής δίνει τη δυνατότητα χρησιμοποίησης μνήμης με διαφορετικό μήκος λέξης. Έτσι, στην περίπτωση του PIC, η μνήμη προγράμματος έχει μήκος λέξης 14 διαδικών ψηφίων (bits) αντί των 8 , της μνήμης των δεδομένων, με σκοπό όλες οι εντολές να κωδικοποιούνται σε μία λέξη.
Το μέγεθος της μνήμης προγράμματος κυμαίνεται από 2 ως 8 KByteskαι συνήθως είναι τύπου Flash. Η συγκεκριμένη τεχνολογία επιτρέπει όχι μόνο την εγγραφή αλλά και το σβήσιμο της μνήμης να πραγματοποιείται με ηλεκτρονικό τρόπο. Αυτό σημαίνει ότι ο προγραμματισμός του μικροελεγκτή γίνεται εύκολα ενώ είναι συνδεδεμένος στο κύκλωμα της εφαρμογής.
Το μέγεθος της μνήμης αποτελείται από τρία τμήματα 128 Bytes έκαστος, δηλαδή 384 Bytes συνολικά. Το κάθε τμήμα αποτελείται από καταχωρητές γενικού αλλα και ειδικού σκοπού, όπου κάποιοι χρησιμοποιούνται για τον έλεγχο του πυρήνα του PIC και άλλοι για τον έλεγχο των περιφερειακών του.

Εντολές

Οι μικροελεγκτές ακολουθούν την αρχιτεκτονική RISC και την αρχιτεκτονική CISC. Ενδεικτικά οι μεγάλοι κεντρικοί ηλεκτρονικοί υπολογιστές βασίζονται σε μικροεπεξεργαστές αρχιτεκτονικής RISC, ενώ ο προσωπικός μας ηλεκτρονικός υπολογιστής βασίζεται σε μικροεπεξεργαστή αρχιτεκτονικής CISC.
Συγκεκριμένα ο PIC, ακολουθεί την RISK όπως αναφέρθηκε παραπάνω, ενώ έχει συνολικά (35) εντολές μήκους μιας λέξης 14 bits. Κατά συνέπεια, σε αντίθεση με τους μικροελεγκτές αρχιτεκτονικής CISC, ο PIC εκτελεί κάθε εντολή σε έναν κύκλο μηχανής, με αποτέλεσμα την πολύ μεγάλη βελτίωση της ταχύτητας επεξεργασίας.

Λειτουργίες Διακοπών

Την ώρα που ο PIC εκτελεί ένα πρόγραμμα που του έχουμε φορτώσει, μπορεί να συμβούν διάφορα γεγονότα στο περιβάλλον που ελέγχει. Αυτό σημαίνει ότι ανάλογα με τη σημασία

του κάθε γεγονότος, ενδεχομένως να χρειαστεί διακοπή της εκτέλεσης του κυρίως προγράμματος και ο μικροελεγκτής να ασχοληθεί με το γεγονός αυτό. Ο PIC για να μπορέσει να αντιληφθεί την ύπαρξη των γεγονότων αυτών έχει αναθέσει σε κάποιες από τις περιφερειακές μονάδες του να κάνουν διάφορους χρήσιμους ελέγχους. Όταν μία από τις περιφερειακές μονάδες εντοπίσει το γεγονός, τότε στέλνει σήμα στην κεντρική μονάδα επεξεργασίας του μικροελεγκτή να διακόψει την εκτέλεση του προγράμματος που εκτελεί. Το σήμα αυτό λέγεται διακοπή και προκαλεί άμεση εκτέλεση ενός τμήματος κώδικα, το οποίο λέγεται ρουτίνα εξυπηρέτησης διακοπής.
Ως παράδειγμα μπορούμε να αναφέρουμε τη χρήση ενός μικροϋπολογιστικού συστήματος βασισμένο στον PIC, που χρησιμοποιείται για τον έλεγχο μίας κατεργασίας μετάλλου σε βιομηχανικό περιβάλλον. Εκτός από τις άλλες ασχολίες του ο PIC, πρέπει να ρυθμίζει τη θερμοκρασία σε κάποιον κλιβανο, ενώ για λόγους ασφαλείας μετρά τη θερμοκρασία σε διάφορα σημεία της κατεργασίας. Κατά τη διάρκεια της εκτέλεσης μιας συνηθισμένης διαδικασίας όπως η αποστολή κάποιων δεδομένων από τη μνήμη του σε έναν κεντρικό ηλεκτρονικό υπολογιστή, διαπιστώνεται επικίνδυνη αύξηση της θερμοκρασίας σε κάποιο σημείο της κατεργασίας. Τότε η περιφερειακή μονάδα που το εντόπισε προκαλεί διακοπή στην κεντρική μονάδα επεξεργασίας, κι έτσι η κεντρική μονάδα επεξεργασίας καλείται να εκτελέσει τη ρουτίνα σε περίπτωση διακοπής. Η ρουτίνα αυτή περιέχει εντολές που σβήνουν τον κλίβανο , ενεργοποιούν ανεμιστήρες ψύξεως και ανάβουν alarm (σηματοδοσία κινδύνου).
Ο PIC δέχεται ένα πλήθος διακοπών, οι οποίες κατά βάση προέρχονται από τις διάφορες περιφερειακές μονάδες. Συνήθως μία περιφερειακή μονάδα μπορεί να δώσει ένα σήμα διακοπής, ωστόσο υπάρχει μονάδες που δίνουν περισσότερες διακοπές.

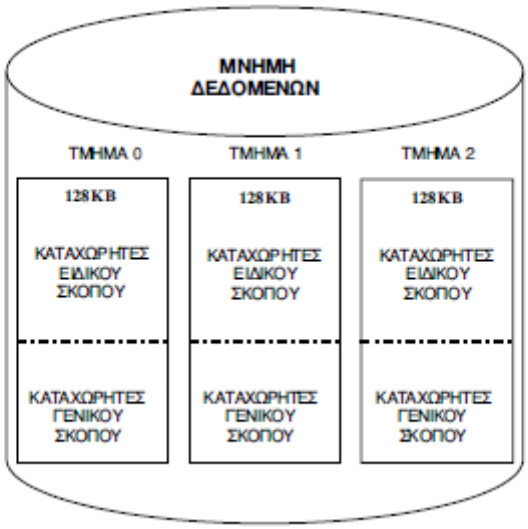
Περιφερειακές Μονάδες

Η κύρια διαφορά των μικροελεγκτών από τους μικροεπεξεργαστές είναι ότι οι πρώτοι έχουν ενσωματωμένα περιφερειακά. Αυτό τους κάνει κατάλληλους για πολλές εφαρμογές όπου το ζητούμενο είναι η χρήση περιφερειακών μονάδων, όπως A/D μετατροπέας (Analog to Digital) , η θύρα σειριακής επικοινωνίας , κτλ. Πολλές φορές , ένας μικροελεγκτής επιλέγεται για μία εφαρμογή με γνώμονα το είδος και τις δυνατότητες των περιφερειακών που διαθέτει. Ο μικροελεγκτής PIC έχει ενσωματωμένα αρκετά περιφερειακά που του δίνουν τη δυνατότητα να χρησιμοποιηθεί για ένα πλήθος εφαρμογών.

1.1.1.3 Καταχωρητές

Οι καταχωρητές είναι ένα από τα βασικότερα στοιχεία της αρχιτεκτονικής ενός μικροελεγκτή. Η ευκολία και οι δυνατότητες προγραμματισμού του μικροελεγκτή έχουν άμεση σχέση με το πλήθος, το είδος και τις δυνατότητες των καταχωρητών του. Κάθε εντολή ενός προγράμματος χρησιμοποιεί έναν τουλάχιστον καταχωρητή.

Ο μικροελεγκτής PIC χρησιμοποιεί δύο ομάδες καταχωρητών, τις οποίες μπορούμε να αναζητήσουμε στη μνήμη δεδομένων του PIC, όπως φαίνεται στο σχήμα 1.4.



Σχήμα 1.3

Η πρώτη ομάδα , που βρίσκεται στις χαμηλότερες διευθύνσεις, περιέχει καταχωρητές ειδικών λειτουργιών (special function registers) , όπως αυτών του ελέγχου των περιφερειακών που βρίσκονται ενσωματωμένα στον μικροελεγκτή. Η δεύτερη ομάδα περιέχει καταχωρητές γενικής χρήσης και αναφέρεται ως αρχείο καταχωρητών γενικού σκοπού (general purpose register file). Εκτός από αυτούς , ο μικροελεγκτής διαθέτει και τους καταχωρητές W και PC.

Ο καταχωρητής W λέγεται και καταχωρητής εργασίας, ενώ είναι ανεξάρτητος από τους υπόλοιπους και βρίσκεται άμεσα συνδεδεμένος με την αριθμητική και λογική μονάδα του PIC. Αυτό του δίνει κάποια πλεονεκτήματα, με αποτέλεσμα να είναι απαραίτητος για την εκτέλεση κάποιον εντολών. Παραδείγματος χάρη μπορούμε να δώσουμε εντολή πρόσθεσης ή αφαίρεσης δύο καταχωρητών μόνο εάν ο ένας από αυτούς είναι ο W.

Ο μετρητής προγράμματος PC, είναι κοινό στοιχείο της αρχιτεκτονικής όλων των μικροελεγκτών. Στην ουσία, είναι ο μόνος τρόπος με τον οποίο η κεντρική μονάδα επεξεργασίας μπορεί να βρει στη μνήμη την επόμενη εντολή που πρέπει να εκτελέσει. Ο PC σε κάθε εντολή που πρόκειται να εκτελεστεί , αυξάνεται κατά 1 ώστε να κρατάει την διεύθυνση της κάθε εντολής.

Πολλές φορές είναι αναγκαίο να εκτελεστεί μία εντολή που βρίσκεται αποθηκευμένη αρκετές θέσεις μνήμης μακριά από την τελευταία εντολή που εκτελέστηκε. Αυτό επιτυγχάνεται με μία εντολή άλματος , ο οποία αλλάζει τη ροή εκτέλεσης του προγράμματος, κι έτσι ο PC καταχωρεί τη διεύθυνση της εντολής που πρέπει να εκτελεστεί.

Ο καταχωρητής STATUS είναι επίσης πολύ σημαντικός και ανήκει στην κατηγορία των καταχωρητών ειδικών λειτουργιών. Στο σχήμα 1.5 μπορούμε να δούμε τη μορφή του.

7	6	5	4	3	2	1	0
0	RP1	RP0	-	-	Z	DC	C

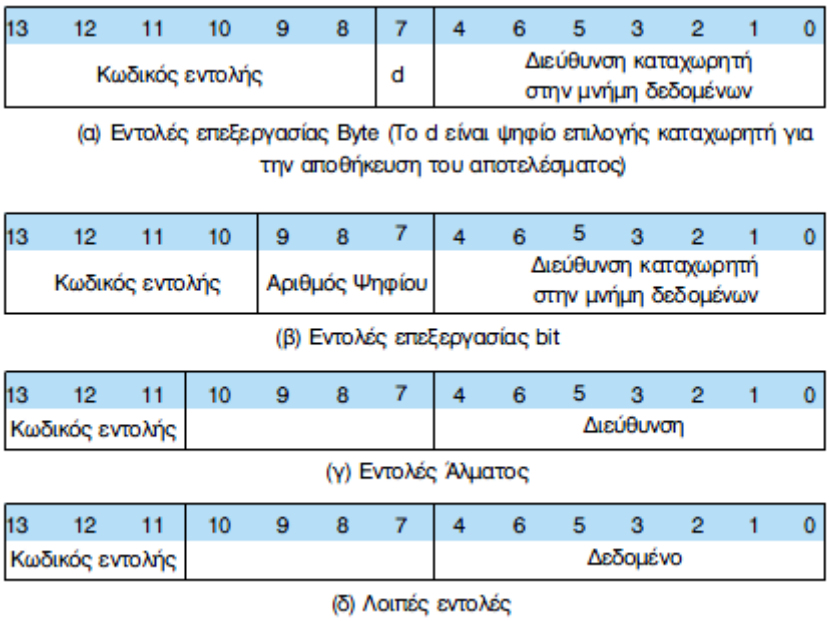
Σχήμα 1.4

1.1.1.4 Τύποι εντολών του PIC

Ο μικροελεγκτής PIC έχει εντολές μήκους μιας λέξης . Η δομή της λέξης διαφέρει από εντολή σε εντολή , όμως σε όλες τις λέξεις το πρώτο τμήμα περιέχει τον κωδικό της εντολής (OPCODE), ενώ το υπόλοιπο έχει πληροφορίες για την εκτέλεση της συγκεκριμένης εντολής. Αυτός ο κωδικός είναι καθορισμένος από την αρχιτεκτονική του συστήματος και είναι μοναδικός για κάθε εντολή. Η κεντρική μονάδα του επεξεργαστή διαβάζοντας τον κωδικό μιας εντολής γνωρίζει ακριβώς τις εργασίες που πρέπει να κάνει , ενώ οι υπόλοιπες πληροφορίες βρίσκονται στην υπόλοιπη λέξη. Τους κωδικούς των εντολών κάθε μικροελεγκτή PIC, μπορούμε να τους βρούμε στο manual του.
Εν τέλει , οι εντολές των PIC χωρίζονται σε τέσσερις κατηγορίες :

1. Εντολές επεξεργασίας Byte (byte - oriented)
2. Εντολές επεξεργασίας Bit (bit - oriented)
3. Εντολές άλματος (αλλαγή ροής προγράμματος)
4. Υπόλοιπες εντολές

Οι δομές τους αποτυπώνονται στο παρακάτω σχήμα



Σχήμα 1.5

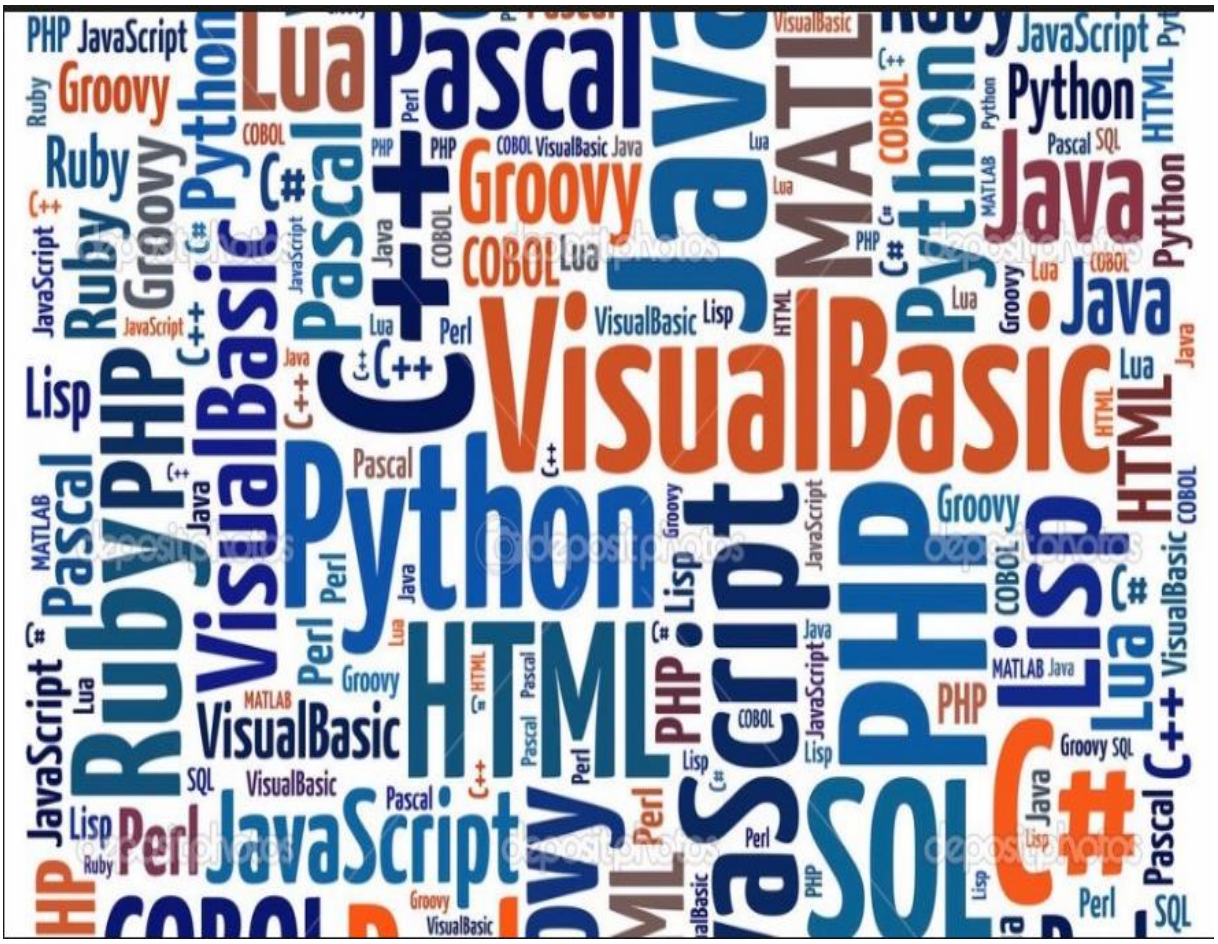
Αναλυτικότερα για τον προγραμματισμό του μικροελεγκτή, θα εμβαθύνουμε στα επόμενα κεφάλαια της πτυχιακής όπου θα είναι αναπτυγμένο το πρόγραμμα της εφαρμογής μας, με πλήρη ανάλυση της κάθε εντολής.

2ΟΚΕΦΑΛΑΙΟ

“ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΤΟΝ PIC”

2.1 Εισαγωγή στις γλώσσες προγραμματισμού

Η επιτυχία μιας οικογένειας μικροελεγκτών καθορίζεται σε μεγάλο βαθμό από τη διαθεσιμότητα και την ευχρηστία των σχετικών εργαλείων ανάπτυξης, όπως μεταφραστές από γλώσσες υψηλού επιπέδου σε γλώσσα κατανοητή από τον μικροελεγκτή (C, C++, assembly), προγραμματιστές της εσωτερικής μνήμης και εργαλεία εκσφαλμάτωσης (debuggers). Στους μικροελεγκτές, τα εργαλεία αυτά δεν αποτελούνται ποτέ μόνο λογισμικό, καθώς δεν υπάρχει τυποποιημένος τρόπος επικοινωνίας με αυτούς. Στον τομέα των εργαλείων ανάπτυξης, δραστηριοποιούνται όχι μόνο οι ίδιοι οι κατασκευαστές μικροελεγκτών αλλά και εξειδικευμένες εταιρείες.



Τα ψηφιακά κυκλώματα μας παρέχουν μία πολύ σημαντική δυνατότητα. Μπορούν να προγραμματίζονται, έτσι ώστε να κάνουν αυτό ακριβώς που απαιτείται από την ανάλογη εφαρμογή που θέλουμε να πραγματοποιήσουμε.
Αρχικά, ο προγραμματισμός ήταν μεγάλες σειρές από δυαδικούς αριθμούς, δηλαδή όλο το πρόγραμμα γραφόταν σε γλώσσα μήχανής. Αυτό ήταν πολύ δύσκολο, ιδίως όταν είχαμε να κάνουμε με μεγάλες και απαιτητικές εφαρμογές. Επίσης, αν ήθελαν οι προγραμματιστές να κάνουν διορθώσεις, προσθήκες και διαγραφές εντολών σε πρόγραμμα γραμμένο σε γλώσσα μηχανής, αντιμετώπιζαν τεράστιες δυσκολίες σε μια διαδικασία πολύ ευάλωτη από λάθη. Επινοήθηκε λοιπόν, στις αρχές τις δεκαετίας του 50’, μια συμβολική γλώσσα για τις εντολές που καταλάβαινε ο υπολογιστής και γράφτηκε ένα συμβολομεταφραστικό πρόγραμμα (assembler), που μετέτρεπε ένα πρόγραμμα συμβολικής γλώσσας σε ένα πρόγραμμα σε γλώσσα μηχανής. Με την πάροδο των χρόνων, ο προγραμματισμός δέυτερης γενιάς εξελίχθηκε σε μια μορφή ακόμα πιο φιλική προς τον χρήστη. Έτσι δημιουργήθηκαν οι γλώσσες τρίτης γενιάς, όπως είναι η Fortran, όπου το σύνολο των εντολών ήταν πιο απλουστευμένο και ένας μεταφραστής (compiler) αναλάμβανε να κάνει τη μετατροπή σε περισσότερες εντολές σε συμβολική γλώσσα και σε γλώσσα μηχανής.

Πλέον ο πραγραμματισμός έχει φτάσει σε μία μορφή που πλησιάζει πάρα πολύ τη φυσική γλώσσα και οι γλώσσες που έχουν δημιουργηθεί καλούνται ανωτέρου επιπέδου. Σήμερα υπάρχουν χιλιάδες γλώσσες προγραμματισμού και κάθε μία έχει το δικό της σύνολο τυπικών προδιαγραφών (ή κανόνων) που αφορούν το συντακτικό, το λεξιλόγιο και το νόημα της. Για τις περισσότερες γλώσσες που χρησιμοποιούνται ευρέως και έχουν χρησιμοποιηθεί για αρκετό χρονικό διάστημα, υπάρχουν ειδικοί οργανισμοί τυποποίησης οι οποίοι μέσα από τακτές συναντήσεις δημιουργούν, τροποποιούν ή επεκτείνουν τις τυπικές προδιαγραφές που διέπουν την χρήση μιας γλώσσας προγραμματισμού

Η πιο διαδεδομένη γλώσσα προγραμματισμού των μικροελεγκτών είναι η C, η C++ και οι παραλλαγές τους. Σε τμήματα του λογισμικού όπου απαιτείται ταχύτητα η μικρό μέγεθος χρησιμοποιούμενης μνήμης, μπορεί να χρησιμοποιείται η Assembly. Όμως οι μεγαλύτερες απαιτήσεις σε λειτουργικότητα και η ευκολία προγραμματισμού της C έναντι της assembly, σε συνδυασμό με την επάρκεια μνήμης των σύγχρονων μικροελεγκτών, έχουν γενικά εκτοπίσει την Assembly από τις περισσότερες εφαρμογές. Η εφαρμογή της παρούσας πτυχιακής εργασίας , γίνεται με γλώσσα C, η οποία έχει σπουδαία ιστορία και θα αναλυθεί παρακάτω.

2.1.1 Γλώσσα Assembly

Μια συμβολική γλώσσα (assembly language) είναι μια χαμηλού επιπέδου γλώσσα προγραμματισμού, δηλαδή μια γλώσσα πολύ κοντά στη γλώσσα μηχανής και στο υλικό του υπολογιστή. Κάθε συγκεκριμένη αρχιτεκτονική συνόλου εντολών, δηλαδή κάθε οικογένεια επεξεργαστών έχει τη δική της συμβολική γλώσσα, η οποία δίνεται συνήθως από τον κατασκευαστή της. Ένα πρόγραμμα σε γλώσσα μηχανής είναι ένα μοτίβο από bits στα οποία κωδικοποιούνται εντολές του επεξεργαστή και δεδομένα. Αυτό γίνεται πιο ευανάγνωστο αντικαθιστώντας τις ακολουθίες των bits με μνημονικά σύμβολα. Για παράδειγμα ένας επεξεργαστής της αρχιτεκτονικής x86/IA-32 θα καταλάβει την εντολή σε γλώσσα μηχανής:

10110000 01100001

Ένας προγραμματιστής όμως είναι πιο εύκολο να θυμάται την ισοδύναμη συμβολική αναπαράσταση, για παράδειγμα μια τυπική εντολή σε συμβολική γλώσσα είναι η εξής:

mov al, 061h

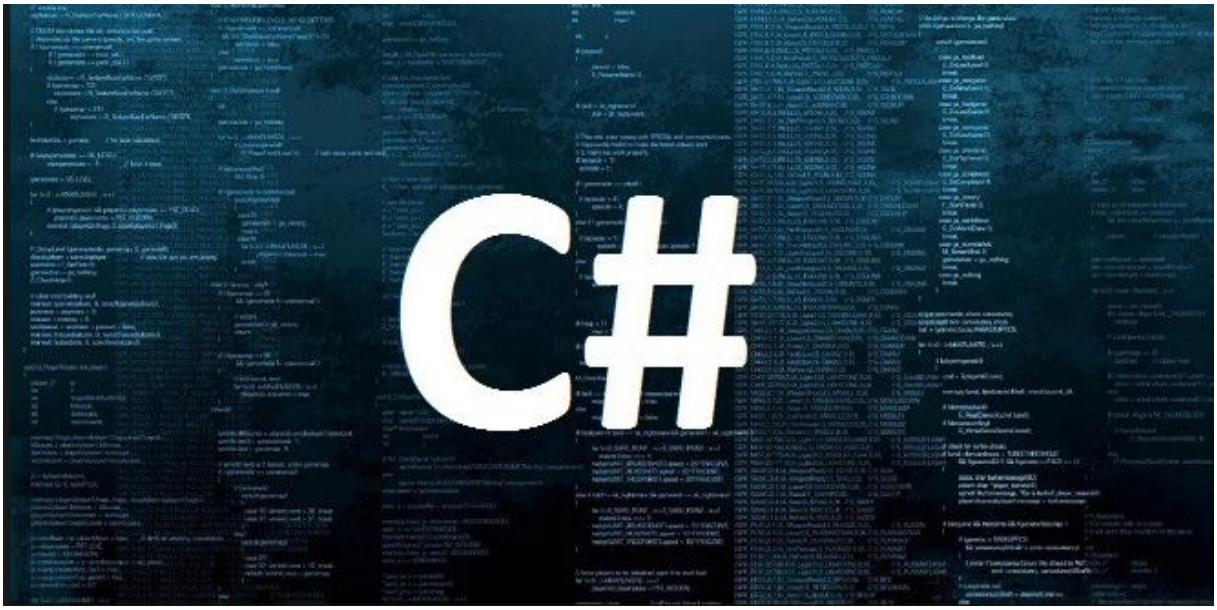
που είναι συντομογραφία της αγγλικής λέξης move ("μετακίνησε"). Η εντολή αυτή μετακινεί τη δεκαεξαδική τιμή 61 (97 στο δεκαδικό σύστημα) στον καταχωρητή με το όνομα "al". Η μετατροπή ενός προγράμματος από συμβολική γλώσσα σε γλώσσα μηχανής γίνεται από ένα συμβολομεταφραστή (assembler) και το αντίστροφο γίνεται από έναν αντισυμβολομεταφραστή (disassembler).

2.1.1.1 Μειονεκτήματα

- Προγραμματιστής πρέπει να σκέφτεται πάλι σε γλώσσα μηχανής (απλά αλλάζει η αναπαράσταση και η σύνταξη των εντολών)
- Η assembly είναι εγγενώς εξαρτημένη από τη μηχανή (οι εντολές ενός προγράμματος σε assembly εκφράζονται με ιδιότητες εγγενείς στην τεχνολογία της συγκεκριμένης μηχανής)
- Δύσκολη έως αδύνατη φορητότητα (portability) (μεταφορά του ίδιου προγράμματος ώστε να είναι σωστό σε μια άλλη μηχανή)
- Τα αρχέτυπα (εντολές) είναι χαμηλού επιπέδου (πολύ κοντά στο hardware)
    - Δεν διευκολύνει στον γενικό σχεδιασμό ενός προγράμματος (γιατί αποτελείται από μικροεντολές που συνδέονται με πολύ μικρές λειτουργίες του Η/Υ)

2.2 Γλώσσα C

Η γλώσσα C σχεδιάστηκε κι αναπτύχθηκε από τον Dennis Ritchie στις αρχές της δεκαετίας του 1970 στα εργαστήρια Bell Labs της εταιρείας AT&T. Η γλώσσα ονομάστηκε έτσι, γιατί πολλά από τα χαρακτηριστικά της προήλθαν από μια παλαιότερη γλώσσα, η οποία είχε αναπτυχθεί από το συνάδελφο του Ritchie, Ken Thompson, και ονομαζόταν B.
Η C δημιουργήθηκε με σκοπό να χρησιμοποιηθεί για την ανάπτυξη του λειτουργικού συστήματος Unix, το οποίο ήταν υπό ανάπτυξη εκείνη την περίοδο στα Bell Labs από την ίδια, πάνω-κάτω, ομάδα επιστημόνων. Για την ακρίβεια, πολύ μεγάλο τμήμα του Unix (πάνω από 90%) γράφτηκε χρησιμοποιώντας τη C. Ο σκοπός αυτός επηρέασε σε πολύ μεγάλο βαθμό τα χαρακτηριστικά της γλώσσας, η οποία από την αρχή προσανατολίστηκε στην ανάπτυξη συστημάτων, επιτρέποντας τη δημιουργία ιδιαίτερα αποτελεσματικού, αλλά και μερικές φορές «επικίνδυνου», κώδικα.
Η C δε γνώρισε άμεση αποδοχή, κυρίως λόγω έλλειψης καλών μεταγλωττιστών έξω από τα Bell Laboratories. Αυτό όμως ήταν ένα προσωρινό πρόβλημα και προς τα τέλη της δεκαετίας του ΄70 άρχισαν να εμφανίζονται αρκετοί μεταγλωττιστές. Σε συνδυασμό με την εξάπλωση χρήσης του Unix, του οποίου η C αποτελούσε βασικό συστατικό, η δημοτικότητα της γλώσσας αυξήθηκε κατακόρυφα.
Αυτό, παρ’ ότι θετικό, δημιούργησε ένα σημαντικό πρόβλημα. Η υλοποίηση της γλώσσας στους περισσότερους μεταγλωττιστές, βασίστηκε σε ένα παράρτημα του πρώτου βιβλίου για τη C (*The C Programming Language* των *Brian Kernighan* και *Dennis Ritchie*). Δυστυχώς, το παράρτημα δεν παρείχε πλήρη και μονοσήμαντο ορισμό της γλώσσας, με αποτέλεσμα οι κατασκευαστές των μεταγλωττιστών να υιοθετούν ο καθένας τις δικές του συμβάσεις. Η ύπαρξη πολλών εκδόσεων της γλώσσας με αρκετές (έστω και μικρές) διαφορές είχε σαν συνέπεια να εμφανιστούν προβλήματα ασυμβατότητας, δηλαδή προγράμματα τα οποία μεταγλωττίζονταν επιτυχώς με μία έκδοση να μην μεταγλωττίζονται με κάποια άλλη.



Η ανάγκη για την ύπαρξη ενός σαφούς ορισμού της γλώσσας έγινε αντιληπτή από τους κατασκευαστές. Έτσι, το 1983 ορίστηκε μια επιτροπή από το Αμερικάνικο Ινστιτούτο Προτύπων (ANSI – American National Standard Institute) με σκοπό τη δημιουργία ενός προτύπου της C, το οποίο να καθορίζει πλήρως τους κανόνες, τα χαρακτηριστικά και τη λειτουργικότητα της γλώσσας.

Οι διεργασίες της επιτροπής ολοκληρώθηκαν το 1989 και το αποτέλεσμα δημοσιεύτηκε το 1990, οδηγώντας στο πρώτο επίσημο πρότυπο της C που έγινε γνωστό ως ANSI C (και αργότερα όταν εμφανίστηκα καινούριες εκδόσεις ως C89 ή C90). Από τότε (και μετά το απαραίτητο διάστημα προσαρμογής) όλοι οι C μεταγλωττιστές υποστηρίζουν το συγκεκριμένο πρότυπο.

Το πρότυπο ANSI C επανεξετάστηκε στα τέλη της δεκαετίας του ’90 προκειμένου να ληφθούν υπ’ όψιν οι εξελίξεις στον τομέα των υπολογιστών, της πληροφορικής και των γλωσσών προγραμματισμού. Οι αλλαγές που έγιναν, οδήγησαν στη δημιουργία ενός νεότερου προτύπου, γνωστού ως C99. Σε αντίθεση ωστόσο με το ANSI C, του οποίου η αποδοχή υπήρξε αρκετά γρήγορη, η αποδοχή του C99 υπήρξε αρκετά πιο αργή. Αυτό οφείλεται, αφ’ ενός μεν, στο γεγονός ότι οι αλλαγές που περιελάμβανε δεν έχουν μεγάλη επίδραση στη γλώσσα, αφ’ ετέρου δε, στο ότι η ίδια η γλώσσα C είχε να αντιμετωπίσει και τον ανταγωνισμό καινούριων γλωσσών (π.χ. C++, Java, C#). Σε κάθε περίπτωση, σήμερα, σχεδόν όλοι οι μοντέρνοι μεταγλωττιστές υποστηρίζουν τα περισσότερα χαρακτηριστικά του C99.

Κλείνοντας την ιστορική αναδρομή, πρέπει να αναφερθούμε και στο C11, το οποίο είναι το τελευταίο πρότυπο για τη γλώσσα C, που δημοσιεύτηκε το 2011. Το πρότυπο αυτό, είχε ως

κύριο στόχο να εξομαλύνει τις διαφορές που υπήρχαν ανάμεσα στη C και τη C++, η οποία προέκυψε από τη C και με την οποία μοιράζονται ένα πολύ μεγάλο μέρος των κανόνων και του συντακτικού. Έτσι, δημιουργήθηκαν δύο πρότυπα (C11 και C++11 αντίστοιχα) τα οποία βρίσκονται σε συμφωνία μεταξύ τους προκειμένου για τη διευκόλυνση των προγραμματιστών αλλά και των κατασκευαστών μεταγλωττιστών.

2.2.1 Χαρακτηριστικά

Στη C δεν επιβάλλεται κάποια συγκεκριμένη μορφή στον πηγαίο κώδικα (όπως, για παράδειγμα, συνέβαινε στις αρχικές εκδόσεις τηςFortran). Ο προγραμματιστής, χωρίς να αγνοεί φυσικά το συντακτικό της γλώσσας, είναι ελεύθερος να δώσει όποια μορφή θέλει στον κώδικα που γράφει (free-formatsource). Το ελληνικό ερωτηματικό (;) χρησιμοποιείται ως τερματιστής εντολών (και όχι ως διαχωριστής, όπως στην Pascal, παραδείγματος χάριν) και τα άγκιστρα ({}) χρησιμοποιούνται για την ομαδοποίηση εντολών (όπως τα begin/end στην Pascal).

Ακόμα, στη C όλος ο εκτελέσιμος κώδικας περιέχεται σε υπορουτίνες οι οποίες ονομάζονται «συναρτήσεις» (όχι με την αυστηρή έννοια του συναρτησιακού προγραμματισμού).
Οι παράμετροι περνιούνται στις συναρτήσεις πάντα με τιμή (pass-by-value). Το πέρασμα με αναφορά (pass-by-reference) γίνεται έμμεσα στην ουσία, περνώντας, ως παραμέτρους των συναρτήσεων, δείκτες στις μεταβλητές των οποίων θέλουμε να αλλάζουμε τις τιμές μέσα από τις συναρτήσεις.
Η φιλοσοφία της C είναι να περιλαμβάνει μόνο τα εντελώς απαραίτητα χαρακτηριστικά, προωθώντας έτσι την ευελιξία και την αποδοτικότητα. Έτσι λοιπόν, η γλώσσα δεν περιλαμβάνει εντολές ακόμα και για πολύ κοινές λειτουργίες (π.χ. είσοδο/έξοδο) στη λογική ότι υπάρχουν προγράμματα που δεν τις χρησιμοποιούν και για τις οποίες η ύπαρξή τους θα ήταν μια αχρείαστη επιβάρυνση. Αντίθετα, για το σκοπό της υποστήριξης τέτοιων κοινών λειτουργιών (εισόδος/εξόδος, διαχείριση δυναμικής μνήμης, κ.λπ.) η C χρησιμοποιεί εξωτερικά υλοποιημένες συναρτήσεις. Το σύνολο αυτών των συναρτήσεων καθορίζονται από τις διάφορες εκδόσεις του προτύπου ANSI C (C89, C99, C11) και είναι γνωστό ως ***C Standard Library.***

Για να μπορούν να γραφτούν προγράμματα C σε έναν υπολογιστή (αλλά και σε ορισμένες περιπτώσεις να εκτελεστούν), θα πρέπει σε αυτόν να υπάρχει εγκατεστημένη μια υλοποίηση της C Standard Library (π.χ. GNU C Library, Microsoft C Run-time Library). Η απαίτηση αυτή, αν και φαίνεται περιοριστική, στην πράξη έχει ελάχιστη σημασία, καθώς εκ των πραγμάτων σχεδόν όλα τα συστήματα περιλαμβάνουν ένα αντίγραφο της βιβλιοθήκης

2.2.2 Πλεονεκτήματα της C

- Ένα από τα κυριότερα πλεονεκτήματα της C που προκύπτει ως αποτέλεσμα των σχεδιαστικών της στόχων είναι η μεγάλη ευελιξία. Καθώς σχεδιάστηκε για προγραμματιστές και με κύριο σκοπό την ανάπτυξη συστημάτων, περιέχει πολύ λίγους περιορισμούς σχετικά με το τι επιτρέπεται και τι όχι. Επιπλέον, αν και (όπως αναφέρθηκε νωρίτερα) είναι μια γλώσσα υψηλού επιπέδου και παρέχει τις αντίστοιχες διευκολύνσεις στον προγραμματιστή, του δίνει επίσης τη δυνατότητα να χρησιμοποιήσει και μηχανισμούς χαμηλού επιπέδου («κοντά» στο επίπεδο του υλικού) αν αυτός το επιθυμεί. Χαρακτηριστικό είναι, ότι μέσω των κατάλληλων εντολών, υποστηρίζει ακόμα και την ενσωμάτωση κώδικα Assembly στον πηγαίο κώδικα των προγραμμάτων.
- Η C είναι μια σχετικά απλή από άποψη μεγέθους γλώσσα. Το συντακτικό της περιέχει λίγες δεσμευμένες λέξεις (λέξεις κλειδιά – keywords) κι έτσι είναι εύκολο για τον προγραμματιστή να μάθει τους βασικούς της κανόνες και να αρχίσει να δημιουργεί προγράμματα αρκετά γρήγορα.
- Άμεση συνέπεια της ευελιξίας της γλώσσας, αλλά και της απλότητάς της, είναι η αποδοτικότητα. Η C παράγει μερικά από τα πιο αποδοτικά προγράμματα και πολύ συχνά χρησιμοποιείται ως μέτρο σύγκρισης για την αποδοτικότητα άλλων γλωσσών προγραμματισμού. Σε αυτό συνεισφέρει επίσης και η ύπαρξη πλειάδας μακροχρόνια εξελιγμένων μεταγλωττιστών.
- Η φορητότητα του κώδικα (code portability) αναφέρεται στη δυνατότητα του ίδιου (σε επίπεδο πηγαίου κώδικα)προγράμματος να μεταγλωττίζεται και να τρέχει σε διαφορετικές πλατφόρμες (διαφορετικό υλικό, λειτουργικό σύστημα, κ.λπ.) χωρίς

αλλαγές. Δηλαδή, στη δυνατότητα υποστήριξης πολλών πλατφορμών γράφοντας μόνο μια φορά τον απαραίτητο κώδικα, κάτι που προφανώς αποτελεί ένα μεγάλο πλεονέκτημα παραγωγικότητας.

```c
#include <stdio.h>

int main(void)
{
    int year, month, day;

    printf("Welcome...\n\n");

    printf("What year were you born? ");
    scanf("%d", &year);

    printf("What month? ");
    scanf("%d", &month);

    printf("What day? ");
    scanf("%d", &day);

    printf("\nOur \"calculations\" show that you were born on %d/%d/%d.\n\n", day, month, year);

    return 0;
}
```

Σχήμα 2.1 Παράδειγμα ενός απλού προγράμματος σε C

- Η C και λόγω της προτυποποίησής της (ANSI C και C99) επιτυγχάνει σε πολύ μεγάλο βαθμό τη φορητότητα του κώδικα. Να σημειωθεί βέβαια, ότι οι διαφορές ανάμεσα στα λειτουργικά συστήματα επιβάλλουν τις περισσότερες φορές, μικρές, έστω, διαφοροποιήσεις, κάτι που η C υποστηρίζει μέσω ειδικών οδηγιών προεπεξεργαστή (preprocessor directives).
- Η C, αν και δεν είναι καινούρια γλώσσα, είναι μια σύγχρονη γλώσσα προγραμματισμού. Περιλαμβάνει όλα εκείνα τα στοιχεία ελέγχου που θεωρούνται επιθυμητά από τη θεωρία αλλά και την πράξη της επιστήμης των υπολογιστών. Υποστηρίζει (και προωθεί) το δομημένο προγραμματισμό, επιτρέποντας τον ορισμό και τη κλήση συναρτήσεων που υλοποιούν μικρά και αυστηρά καθορισμένα τμήματα λειτουργικότητας.
- Όπως αναφέρθηκε και νωρίτερα, η C είναι μια ιδιαίτερα δημοφιλής γλώσσα. Αυτό σημαίνει ότι ο προγραμματιστής μπορεί να βρει μια πολύ μεγάλη ποικιλία από έτοιμες και καλά ελεγμένες βιβλιοθήκες συναρτήσεων για σχεδόν οποιαδήποτε ανάγκη για την κωδικοποίηση, αποκωδικοποίηση, αναπαραγωγή κι εγγραφή video και ήχου). Επιπλέον, ο προγραμματιστής θα βρει μια μεγάλη κοινότητα (σε blogs, fora , κ.λπ.) όπου μπορεί να ανακαλύψει απαντήσεις σε προβλήματα που αντιμετωπίζει.
- Τέλος, να σημειωθεί ότι η C έχει πάρα πολλά κοινά χαρακτηριστικά (keywords, συντακτικό, οργάνωση κώδικα) με πολλές σύγχρονες (και πιο πολύπλοκες) γλώσσες προγραμματισμού (με τη C++ ειδικά, αλλά και με τη Java, τη C#, κ.α.). Συνεπώς, η καλή εκμάθηση της C μπορεί να αποτελέσει ένα ιδανικό πρώτο βήμα για την εκμάθηση κάποιας από αυτές τις γλώσσες.

2.2.3 Μειονεκτήματα της C

- Η C, όπως όλα τα εργαλεία, έχει κι αυτή τα μειονεκτήματά της. Σε πολλές περιπτώσεις, τα ίδια σχεδιαστικά χαρακτηριστικά και δυνατότητες που οδηγούνε σε μερικά από τα σημαντικότερα πλεονεκτήματα της, μπορούν να οδηγήσουνε και σε μερικές από τις βασικότερες αδυναμίες της.
- Η ευελιξία της γλώσσας, η έλλειψη περιορισμών και η δυνατότητα χρήσης χαρακτηριστικών «κοντά» στο επίπεδο του υλικού (π.χ. δείκτες), επιτρέπουν πολλές φορές τη συγγραφή «επικίνδυνου» κώδικα η πλήρης λειτουργικότητα του οποίου δεν είναι δυνατόν να ελεγχθεί απόλυτα από το μεταγλωττιστή.

- Το μικρό λεξιλόγιο, η εκφραστικότητα της γλώσσας και το πλήθος των τελεστών δίνει τη δυνατότητα στους προγραμματιστές, να δημιουργήσουν μικρά και πυκνογραμμένα προγράμματα των οποίων η λογική είναι πολύ δύσκολο να ακολουθηθεί ακόμα και σε λίγες γραμμές κώδικα.

2.2.4 Διαδικασία δημιουργίας και εκτέλεσης ενός προγράμματος C

Συγγραφή Κώδικα – Edit

Ο προγραμματιστής, χρησιμοποιώντας τον editor της επιλογής του, συντάσσει τον κώδικα του προγράμματος και τον αποθηκεύει στο δίσκο. Ο κώδικας αυτός ονομάζεται ***πηγαίος κώδικας*** (***source code***). Κατά σύμβαση, τα αρχεία πηγαίου κώδικα της C έχουν επέκταση ***.c*** (αν και δεν αποκλείεται να τα δείτε και με επέκταση ***.cpp*** αν ο editor είναι προσανατολισμένος στη C++).

Μεταγλώττιση – Compile

Το πρόγραμμα μεταγλωττίζεται σε ***δυαδικό κώδικα μηχανής*** (***binary code*** ή ***machine code***) που καταλαβαίνει ο υπολογιστής και αποθηκεύεται στο δίσκο. Τα ενδιάμεσα αυτά αρχεία που παράγονται από το μεταγλωττιστή, ονομάζονται ***αρχεία αντικειμενικού κώδικα*** (***object file***). Συνήθως έχουν το ίδιο όνομα με το αρχείο πηγαίου κώδικα κι επέκταση ***.o*** ή ***.obj***, αν και αυτό μπορεί να αλλαχθεί δίνοντας τις κατάλληλες παραμέτρους.

Σύνδεση – Linking

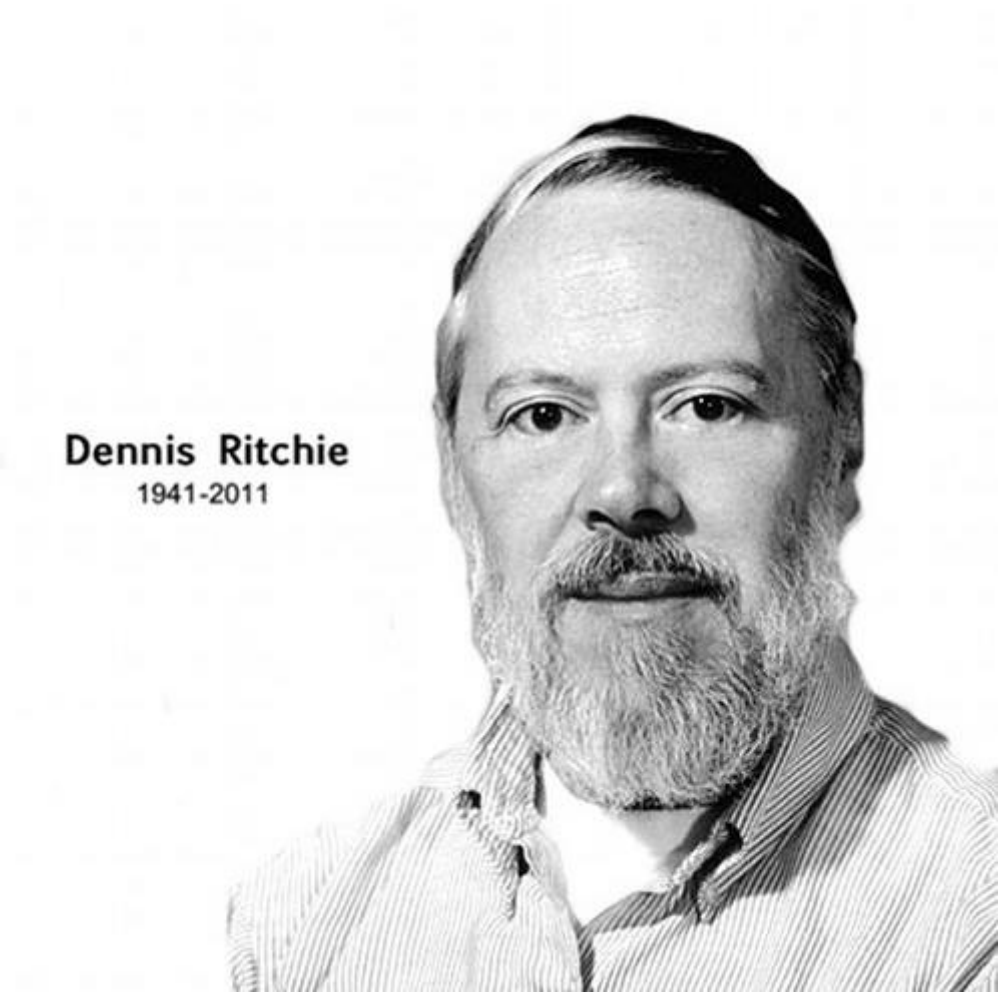
Στη συνέχεια, το πρόγραμμα σύνδεσης (linker) αναλαμβάνει να συνδυάσει όλα τα αρχεία αντικειμενικού κώδικα που παρήγαγε ο compiler, καθώς και όσες προ-μεταγλωττισμένες βιβλιοθήκες συναρτήσεων χρειάζονται προκειμένου να παράγει το τελικό εκτελέσιμο κώδικα και να τον αποθηκεύσει στο δίσκο. Τα εκτελέσιμα αρχεία, όπως και τα αρχεία κώδικα μηχανής, έχουν συνήθως το ίδιο όνομα με το αρχικό αρχείο πηγαίου κώδικα. Ωστόσο, αν έχουν προέλθει από πολλαπλά object files, θα πρέπει ο προγραμματιστής να επιλέξει το τελικό όνομα. Στα Windows τα εκτελέσιμα αρχεία έχουν επέκταση ***.exe***, ενώ στο Linux και το Mac OS X συνήθως δεν έχουν καθόλου επέκταση ή (κυρίως σε παλιότερες εκδόσεις) μπορεί να δείτε την επέκταση ***.out***.

Τέλος, να σημειωθεί ότι στα περισσότερα σύγχρονα IDEs (αλλά και stand-alone compilers) η διαδικασία της σύνδεσης συνήθως εκτελείται αυτόματα μετά την επιτυχή μεταγλώττιση χωρίς

να χρειάζονται περαιτέρω ενέργειες από τον προγραμματιστή. Πάντα, βέβαια, υπάρχει η δυνατότητα με τις κατάλληλες επιλογές ή/και παραμέτρους να γίνει ξεχωριστά.
Τυπικά, σε αυτό το σημείο η διαδικασία δημιουργίας του προγράμματος έχει ολοκληρωθεί. Ωστόσο, και για καλύτερη κατανόηση κάποιων βασικών χαρακτηριστικών της λειτουργίας των υπολογιστών που μπορούν να βοηθήσουν και στην κατανόηση της C, παραθέτουμε και τα δύο επόμενα βήματα που αφορούν στην εκτέλεση του προγράμματος.

Φόρτωση – Load

Το εκτελέσιμο αρχείο φορτώνεται στη μνήμη από το loader του λειτουργικού συστήματος. Κυριολεκτικά, τα bytes που αποτελούν τον εκτελέσιμο κώδικα διαβάζονται από το δίσκο και αντιγράφονται στη RAM, ενώ παράλληλα εκτελούνται και μερικές επιπλέον εργασίες προετοιμασίας (π.χ. δέσμευση της απαραίτητης μνήμης).



Εκτέλεση – Execute

Η CPU (ΚΜΕ – Κεντρική Μονάδα Επεξεργασίας) παίρνοντας την κατάλληλη οδηγία από το λειτουργικό σύστημα, αρχίζει να εκτελεί σειριακά τις εντολές του προγράμματος. Καθώς το πρόγραμμα εκτελείται και αλληλεπιδρά με το χρήστη (ή άλλες εφαρμογές και υπηρεσίες) τα δεδομένα αλλάζουν και οι νέες τιμές αποθηκεύονται μέχρι την ολοκλήρωσή του.

Debugging (Αποσφαλμάτωση) – Εύρεση Λαθών και Διόρθωση

Στην παραπάνω διαδικασία, και για λόγους απλότητας, θεωρήθηκε ότι ο κώδικας ο οποίος δημιουργείται από τον προγραμματιστή είναι σωστός τόσο συντακτικά, όσο και λογικά. Στην πράξη, ωστόσο, πολύ σπάνια θα δημιουργήσετε ένα απόλυτα σωστό πρόγραμμα με τη μια (εκτός αν αναφερόμαστε σε προγράμματα 10 γραμμών, αλλά και πάλι δεν είναι σπάνιο να υπάρχουν μικρά λάθη). Αυτό σημαίνει ότι μετά την ολοκλήρωση της διαδικασίας της μεταγλώττισης ή/και της σύνδεσης πρέπει να ελέγχουμε για τυχόν μηνύματα λαθών από τον compiler και το linker αντίστοιχα. Αν υπάρχουν τέτοιου είδους μηνύματα ακολουθείται μια επαναληπτική διαδικασία διόρθωσης και μεταγλώττισης μέχρι να εξαλειφθούν όλα. Στο τέλος, θα πρέπει πάντα να ελέγξουμε αν το πρόγραμμα εκτελείται κανονικά κι αν επιτελεί το σκοπό για τον οποίο σχεδιάστηκε. Αν όχι, θα πρέπει ακολουθήσει νέος κύκλος διορθώσεων μέχρι να φτάσουμε στο επιθυμητό αποτέλεσμα

.

3ο Κεφάλαιο

“ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ (MPLAB) ”

3.1 Εισαγωγή στο περιβάλλον του MPLAB

Από την εταιρία Microchip διατίθεται στο εμπόριο δωρεάν το πρόγραμμα εξομοίωσης MPLAB. Το MPLAB είναι στην πραγματικότητα ένα πολλαπλό εργαλείο ανάπτυξης στο οποίο εμπεριέχονται τα προγράμματα :

- Assembler για τη μετάφραση του αρχείου προγράμματος .asm σε γλώσσα μηχανής
- Μετατροπέας του αρχείου στην κατάλληλη μορφή .hex για τον προγραμματισμό του - μικροελεγκτή
- Εξομοιωτής προγραμματισμού της μνήμης ROM του μικροελεγκτή

Ενώ για την παρούσα εργασία χρησιμοποιήθηκε ο compiler XC8 της Microchip, για την μεταγλώττιση του προγράμματος σε γλώσσα μηχανής.

Ο εξομοιωτής (simulator) ως γνωστό είναι ένα πρόγραμμα το οποίο τρέχει σε έναν ηλεκτρονικό υπολογιστή ή άλλο τερματικό εξομοιώνοντας τη λειτουργία του μικροελεγκτή. Το πρόγραμμα αυτό μιμείται ακριβώς τη συμπεριφορά του συγκεκριμένου μικροελεγκτή. Κατά την εκτέλεση του προγράμματος με τον εξομοιωτή γίνεται κυρίως έλεγχος ως προς τη λογική.

3.1.1.1 Μειονεκτήματα

Τα μειονεκτήματα είναι :

- Η εκτέλεση του προγράμματος στον εξομοιωτή δεν γίνεται στον ίδιο χρόνο που θα γινόταν αν εκτελείτο από τον ίδιο τον μικροελεγκτή.
- Δεν μπορεί να γίνει είσοδος ή έξοδος πραγματικού σήματος από και προς το μικροελεγκτή υπό μορφή τάσης.

3.1.1.2 Πλεονεκτήματα

- Ορισμένες από τις διευκολύνσεις που παρέχονται από τους εξομοιωτές είναι :
- Η εκτέλεση του προγράμματος με τον εξομοιωτή μπορεί να γίνει πολύ αργά, έτσι ώστε να μπορέσει ο χρήστης να βλέπει τα ενδιάμεσα αποτελέσματα της κάθε εντολής.
- Ο χρήστης μπορεί να τοποθετεί οπουδήποτε σημεία σταματήματος της εκτέλεσης του προγράμματος (break points) ή να εκτελεί τις εντολές βήμα προς βήμα ή να παρεμβάλει σε κάποια σημεία νέες εντολές και πολλά ακόμα.
- Ο χρήστης μπορεί να δει τις τιμές των καταχωρητών καθώς και την κατάσταση των ακροδεκτών I/O χρησιμοποιώντας παράθυρα προβολής (watch windows).
- Υπάρχει δυνατότητα εμφάνισης της κατάστασης του μικροελεγκτή μετά από την υποτιθέμενη μεταβολή του επιπέδου σε κάποια ακίδα ή κατά τη ζήτηση διακοπής.

3.2 Προγραμματίζοντας στο MPLAB

Κατά την ανάπτυξη της κάθε εφαρμογής δημιουργούνται αρκετά αρχεία από τα οποία τα πιο σημαντικά είναι τα παρακάτω:

Αρχεία .C

Είναι τα αρχεία που περιέχουν τον κώδικα του προγράμματος σε γλώσσα Assembly για τους μικροελεγκτές της MICROCHIP. Τα αρχεία αυτά δημιουργούνται με ένα πρόγραμμα συντάκτη (editor) και συνήθως χρησιμοποιείται ο ενσωματωμένος συντάκτης του MPLAB.

Αρχεία . ASM

Είναι τα αρχεία που περιέχουν τον κώδικα του προγράμματος σε γλώσσα Assembly για τους μικροελεγκτές της MICROCHIP. Τα αρχεία αυτά δημιουργούνται με ένα πρόγραμμα συντάκτη (editor) και συνήθως χρησιμοποιείται ο ενσωματωμένος συντάκτης του MPLAB.

Αρχεία . HEX

Είναι τα αρχεία που περιέχουν τον κώδικα του προγράμματος σε γλώσσα μηχανής (δεκαεξαδική μορφή) μετά την συμβολομετάφρασή τους από τον συμβολομεταφραστή (assembler) που είναι εμπεριέχεται στο MPLAB. Αυτό πετυχαίνεται με την επιλογή BuildAll από το μενού Project και είναι απαραίτητα για την προσομοίωση της εκτέλεσης του προγράμματος καθώς και για την διαδικασία προγραμματισμού του μικροελεγκτή.

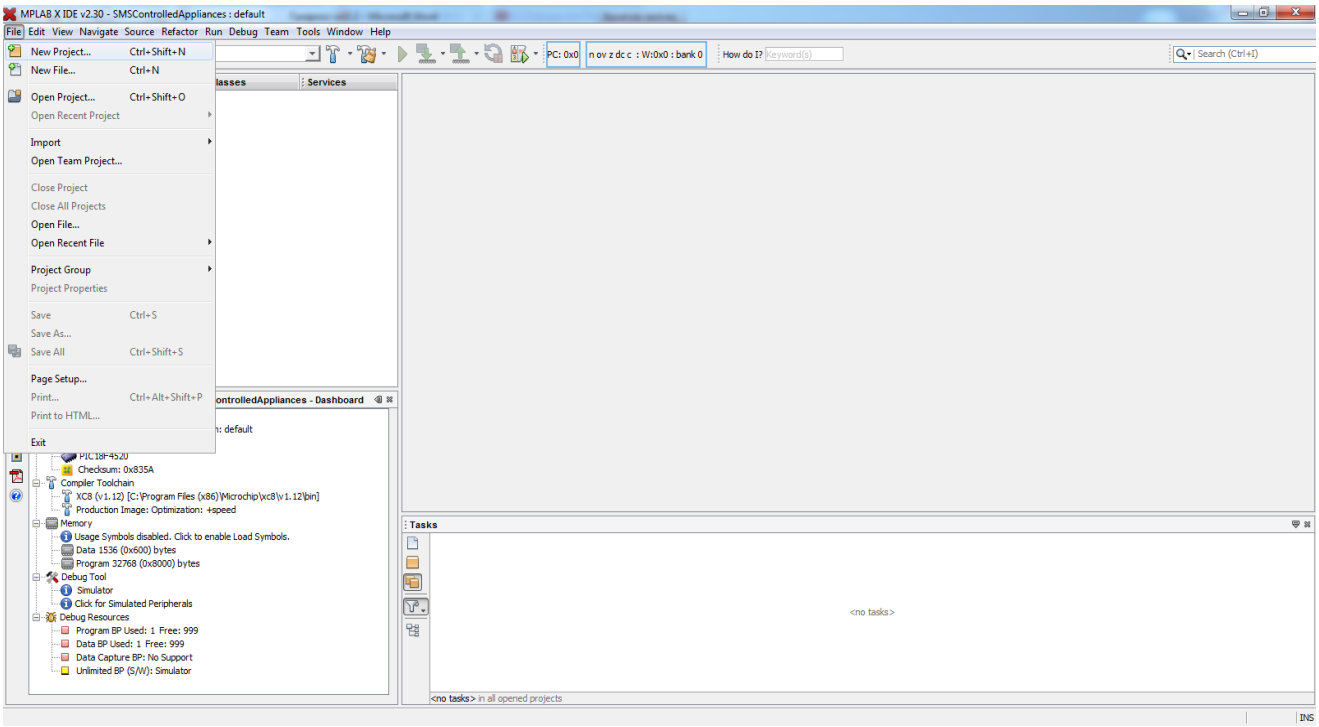
Αρχεία . MCP

Είναι τα αρχεία που περιέχουν τις πληροφορίες για την εφαρμογή (project) που θέλουμε να αναπτύξουμε και στο οποίο ενσωματώνονται τα αρχεία τύπου .ASM που περιέχουν τον κώδικα σε γλώσσα assembly.

Αρχεία..MCW

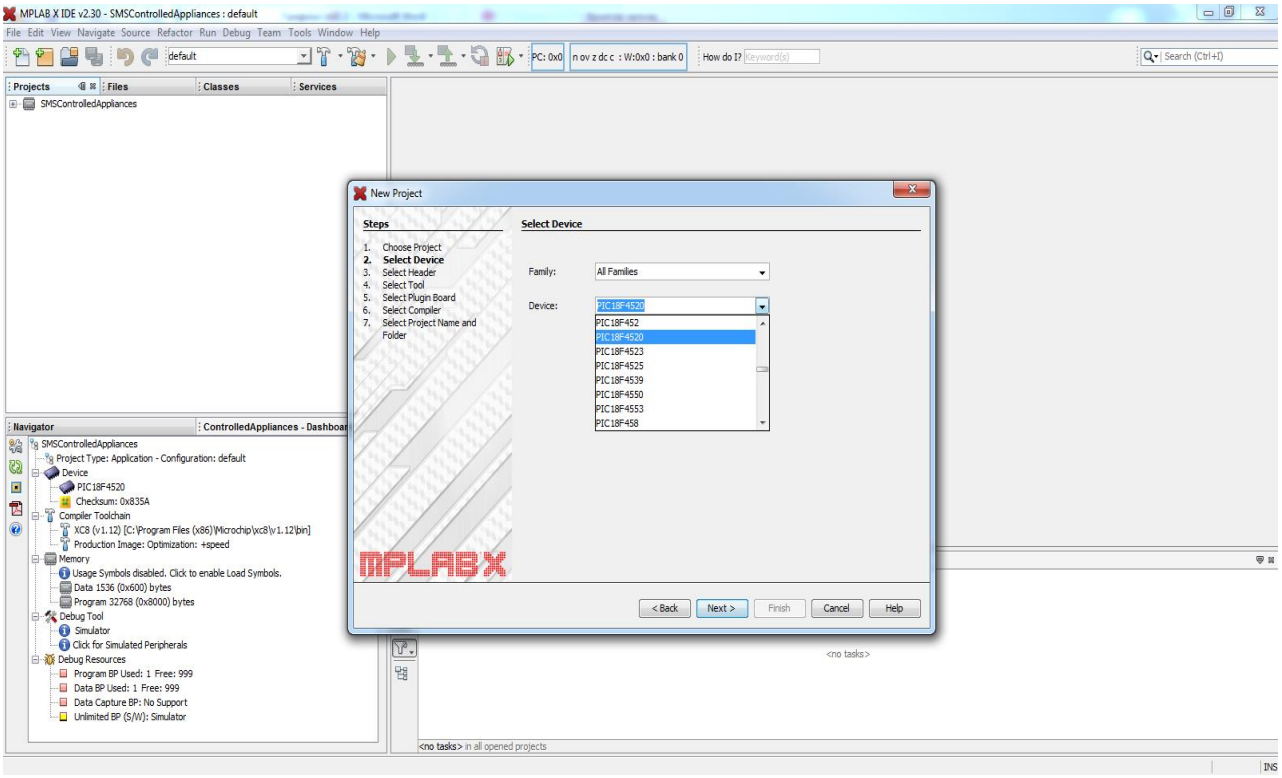
Είναι τα αρχεία που περιέχουν γενικότερες πληροφορίες για το χώρο εργασίας μας (workspace) όπως για παράδειγμα ο τύπος του μικροελεγκτή ή οι ρυθμίσεις για τον συντάκτη και τον συμβολομεταφραστή όπως αυτές χρησιμοποιούνται από καθέναν προγραμματιστή- ηλεκτρονικό που δουλεύει με το MPLAB. Τα αρχεία τύπου workspace (.mcw) περιέχουν τις πιο γενικές πληροφορίες και ακολουθούν τα αρχεία τύπου project (.mcp) στα οποία ενσωματώνονται ως αρχεία πηγαίου κώδικα τα αρχεία τύπου assembly (.asm)

Ξεκινώντας το λογισμικό MPLAB, ρυθμίζουμε τις λεπτομέρειες του παραθύρου έργου επιλέγοντας από το μενού NEW PROJECT.



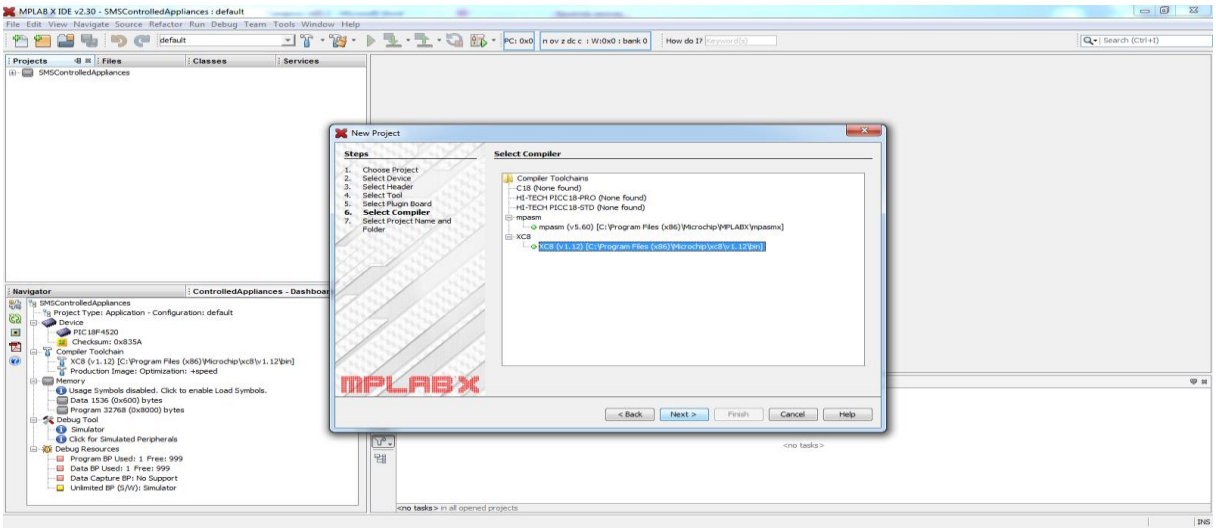
Εικόνα 3.1 : Το παράθυρο έναρξης project στο MPLAB

Στη συνέχεια θα πρέπει να επιλέξουμε την οικογένεια και τον τύπο του μικροελεγκτή που θα χρησιμοποιήσουμε όπως φαίνεται στην εικόνα 3.2



Εικόνα 3.2 : Το παράθυρο επιλογής μικροελεγκτή

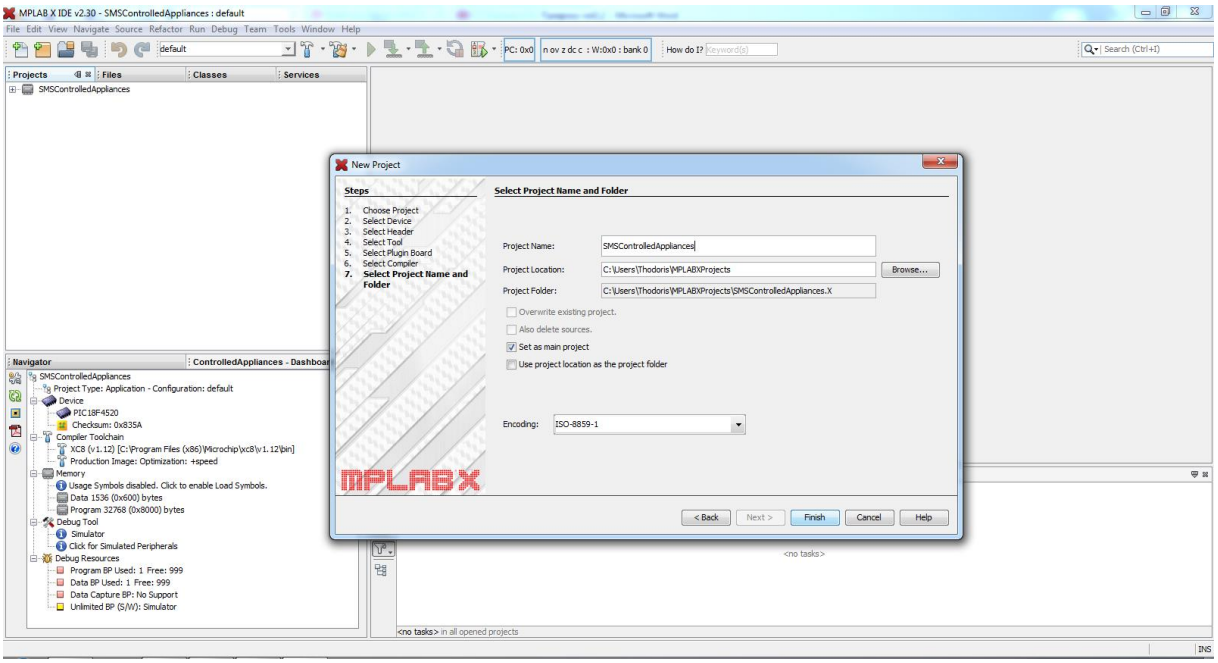
Με το επόμενο βήμα, καθορίζουμε την γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε που στην περίπτωσή μας είναι η γλώσσα C για τους μικροελεγκτές PIC και για το λόγο αυτό επιλέγουμε να φορτωθεί ο συμβολομεταφραστής XC8 με καθορισμένη διαδρομή αναζήτησης που οδηγεί στο φάκελο που έχει εγκατασταθεί το λογισμικό MPLAB.



Εικόνα 3.3 : Το παράθυρο επιλογής της γλώσσας προγραμματισμού

Τέλος, δίνουμε όνομα στο έργο που θα δημιουργήσουμε καθώς και τον φάκελο στο οποίο το έργο θα αποθηκευτεί σύμφωνα με το παράθυρο της εικόνας.

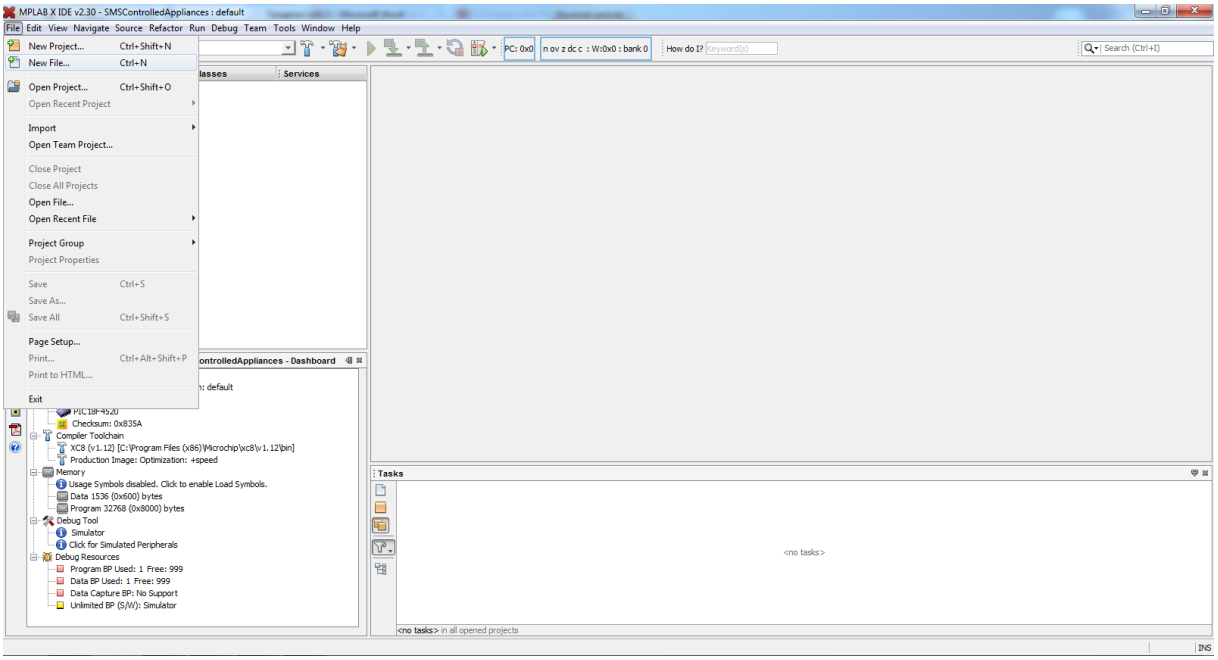
Στον παραπάνω φάκελο αυτό, θα αποθηκεύουμε και όλα τα αρχεία με τους κώδικες C, assembly κτλ που θα δημιουργήσουμε στην συνέχεια. Η διαδικασία αυτή ολοκληρώνεται με την διαδοχική επιλογή των πλήκτρων NEXT και FINISH οπότε εμφανίζεται ένα παράθυρο με την περίληψη όλων των ρυθμίσεων που δώσαμε έως τώρα.



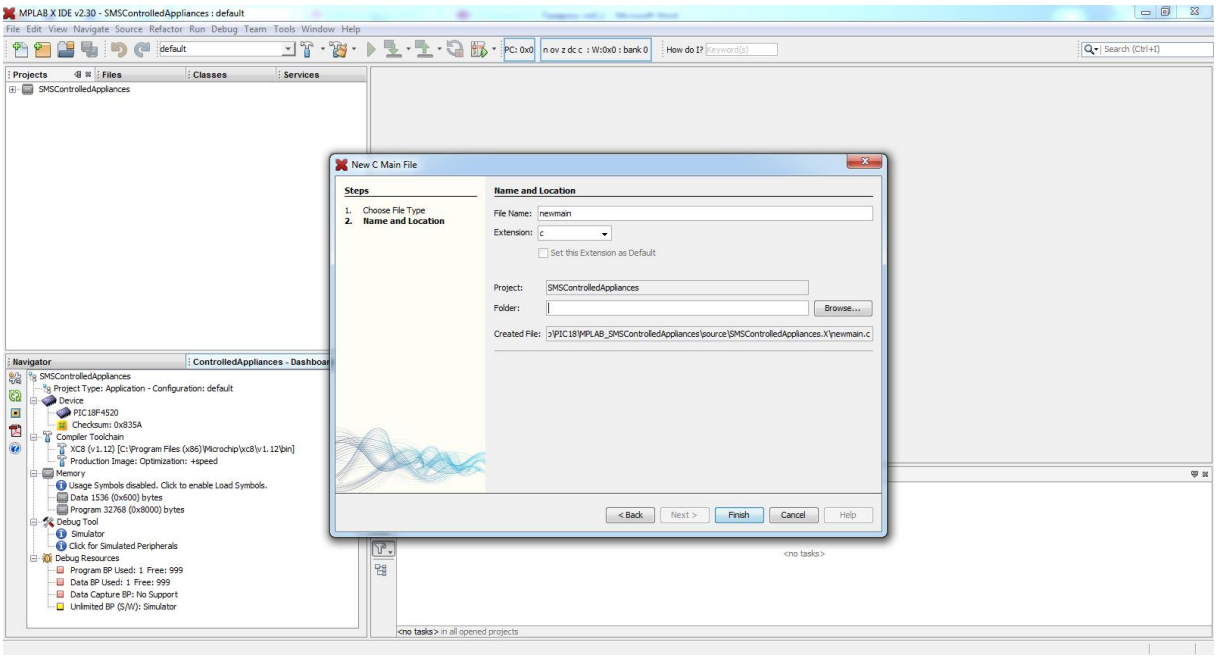
Εικόνα 3.4

3.2.1 Δημιουργία του αρχείου κώδικα

Για να γράψουμε τον κώδικα πρέπει να δημιουργήσουμε ένα αρχείο με την βοήθεια του συντάκτη (editor) του MPLAB. Αυτό γίνεται από το μενού FILE και την επιλογή NEW, οπότε ανοίγει ένα λευκό παράθυρο του συντάκτη στο οποίο γράφουμε τις εντολές του κώδικα.

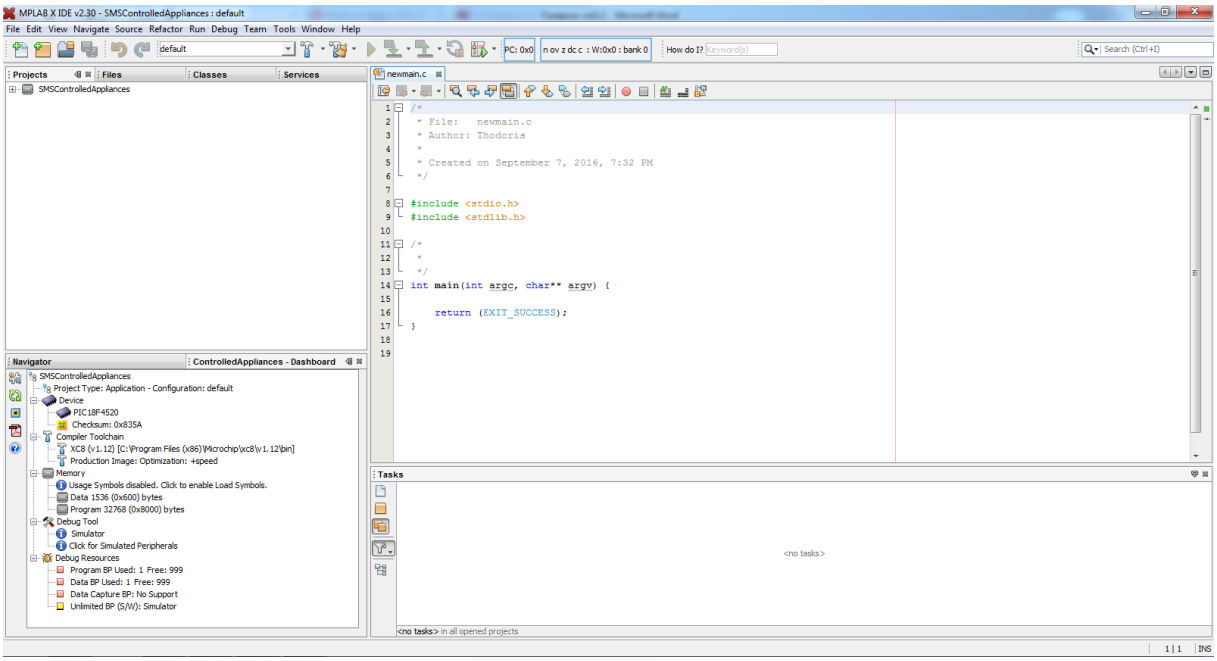


Εικόνα 3.5: Το παράθυρο απόδοσης ονόματος στο έργο.



Εικόνα 3.6: Το παράθυρο περίληψης των ρυθμίσεων του έργου

Στο ίδιο μενού διακρίνουμε τις γνωστές επιλογές των WINDOWS για την δημιουργία νέου αρχείου (NEW) , για την φόρτωση ήδη αποθηκευμένου αρχείου (OPEN) και για την αποθήκευση αρχείου (SAVE). Επίσης στο ίδιο μενού προβλέπεται και η δημιουργία, φόρτωση και αποθήκευση σε αρχείο του χώρου εργασίας που έχουμε δημιουργήσει (Workspace, αρχεία με προέκταση .mcw) που θα περιλαμβάνει όλες τις πληροφορίες τόσο για το έργο (project με προέκταση .c)) .



Εικόνα 3.7 : Το παράθυρο του συντάκτη (editor) για την συγγραφή κώδικα

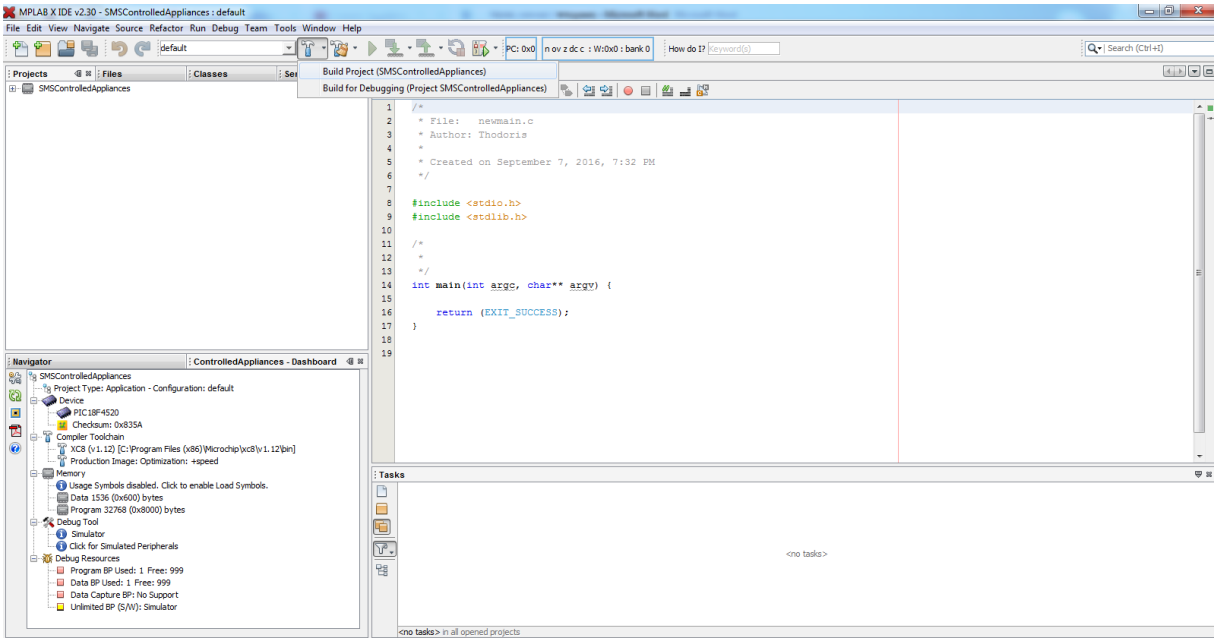
3.2.1.1 Συμβολομετάφραση

Απαραίτητη προϋπόθεση για την συμβολομετάφραση του κώδικα ας πουμε από γλώσσα assembly ή C σε γλώσσα μηχανής είναι να έχουν ολοκληρωθεί όλα τα παραπάνω βήματα με την τελική ενσωμάτωση των αρχείων μέσα στο έργο. Η συμβολομετάφραση γίνεται στην συνέχεια, με την επιλογή BuildAll ή Build project (ctrl+F10) του μενού Project, όπως φαίνεται στην εικόνα Εικόνα 3.8.

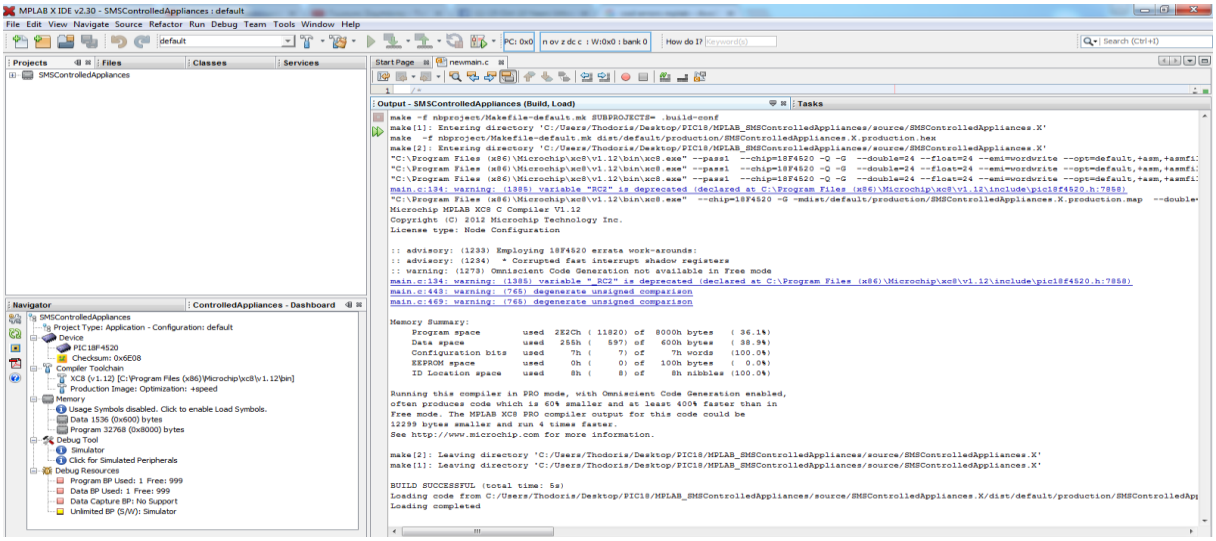
Το αποτέλεσμα της ελέγχου της συμβολομετάφρασης και τα συντακτικά λάθη που τυχόν υπάρχουν, εμφανίζονται σε ένα παράθυρο εξόδου (output, Εικόνα 3.9). Κάνοντας κλικ πάνω σε κάθε συντακτικό λάθος, το MPLAB μας μεταφέρει στην αντίστοιχη γραμμή του παραθύρου κώδικα για να το διορθώσουμε.

Από την διαδικασία συμβολομετάφρασης, παράγεται ένα αρχείο που περιέχει τον κώδικα του προγράμματος σε γλώσσα μηχανής (δεκαεξαδική μορφή) και το οποίο μπορούμε εύκολα να δούμε με την εφαρμογή Notepad των Windows. Το αρχείο αυτό έχει το ίδιο όνομα

με εκείνο που περιέχει τον κώδικα assembly (.asm) αλλά με διαφορετική επέκταση , την .hex. Το αρχείο αυτό είναι πολύ σημαντικό γιατί είναι το αρχείο με τον κώδικα μηχανής που θα φορτωθεί στο ολοκληρωμένο του μικροελεγκτή κατά την διαδικασία προγραμματισμού. Με τον όρο ‘προγραμματισμό’ εννοούμε την διαδικασία με την οποία μεταφέρεται ο κώδικας μηχανής από τον προσωπικό Η/Υ στη μνήμη προγράμματος του μικροελεγκτή και η οποία θα παρουσιαστεί στην συνέχεια.



Εικόνα 3.8: Η διαδικασία συμβολομετάφρασης του κώδικα assembly.



Εικόνα 3.9: Το παράθυρο εξόδου με τα αποτελέσματα της συμβολομετάφρασης του κώδικα assembly.

3.2.2 Προσομοίωση της εκτέλεσης του προγράμματος και προγραμματισμός

Η διαδικασία πραγματοποιείται εφόσον στο περιβάλλον του MPLAB έχει δημιουργηθεί όπως στην προηγούμενη ενότητα, ή έχει φορτωθεί, ένα αρχείο τύπου .hex. Η ενεργοποίηση του προσομοιωτή γίνεται επιλέγοντας το εργαλείο MPLABSIM διαδοχικά από τα μενού : Debugger/SelectTool.

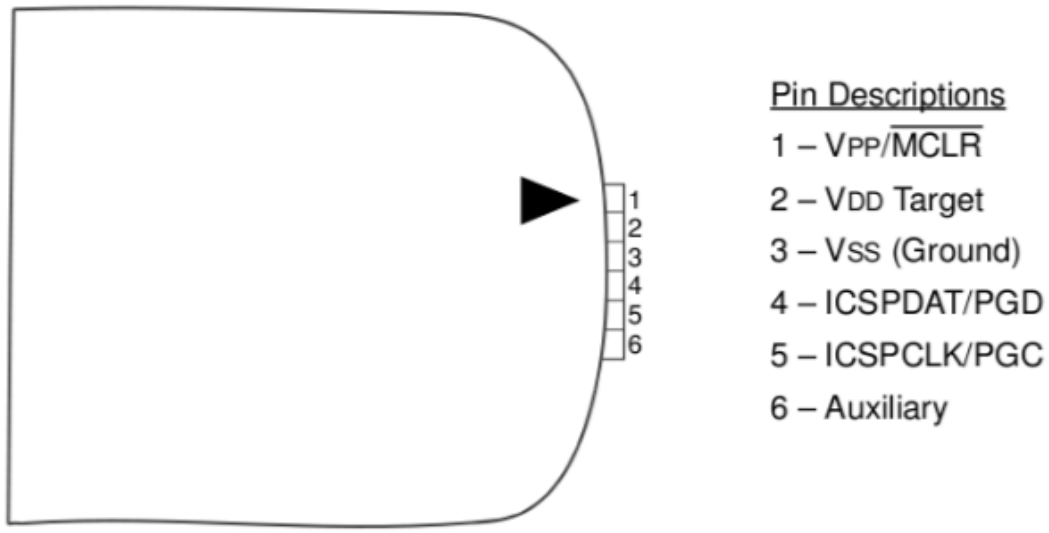
Προγραμματισμός του μικροελεγκτή

Όπως αναφέρθηκε στην προηγούμενη παράγραφο, για τον προγραμματισμό του μικροελεγκτή πρέπει απαραίτητα να έχει δημιουργηθεί ένα αρχείο τύπου .Hex που περιέχει τον κώδικα του προγράμματος σε γλώσσα μηχανής.

Με το προγραμματισμό, μεταφέρουμε το αρχείο τύπου .hex από τον Η/Υ στην μνήμη προγράμματος του μικροελεγκτή. Συνήθως, η εκτέλεση του προγράμματος αρχίζει με την τροφοδότηση του ολοκληρωμένου του μικροελεγκτή. Από εκεί και πέρα ο προγραμματισμός μπορεί να γίνει με διάφορες μονάδες προγραμματισμού οι οποίες συνδέονται στον Η/Υ είτε μέσω της σειριακής είτε μέσω της παράλληλης διασύνδεσης είτε τέλος μέσω της διασύνδεσης USB. Στην παρούσα εργασία χρησιμοποιήθηκε ο Pickit2 της Microchip για τον προγραμματισμό του μικροελεγκτή, ο οποίος στην ουσία φόρτωσε τον κώδικα C σε αρχείο .Hex δηλαδή σε γλώσσα μηχανής ώστε να προγραμματιστεί ο PIC.

Εικόνα 3.10 : Ο Debugger Pickit2

Η πλακέτα μας δεν διαθέτει είσοδο USB, μόνο σειριακή οπότε στην παρακάτω εικόνα φαίνονται οι ακίδες του Pickit2.



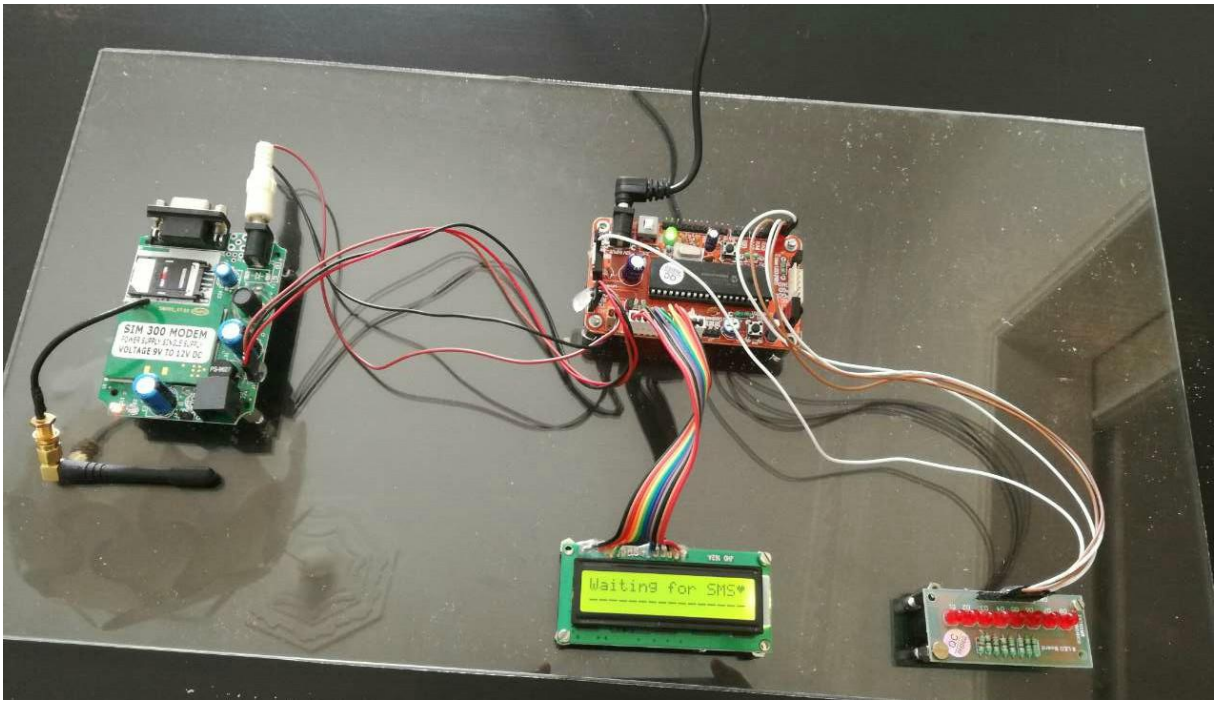
Εικόνα 3.11 : Τα output pins του Pickit2 (Σειριακή σύνδεση)

4Ο ΚΕΦΑΛΑΙΟ

"ΑΝΑΛΥΣΗ ΚΑΙ ΠΑΡΟΥΣΙΑΣΗ ΤΗΣ ΚΑΤΑΣΚΕΥΗΣ"

4.1 Εισαγωγή στη λειτουργία της κατασκευής

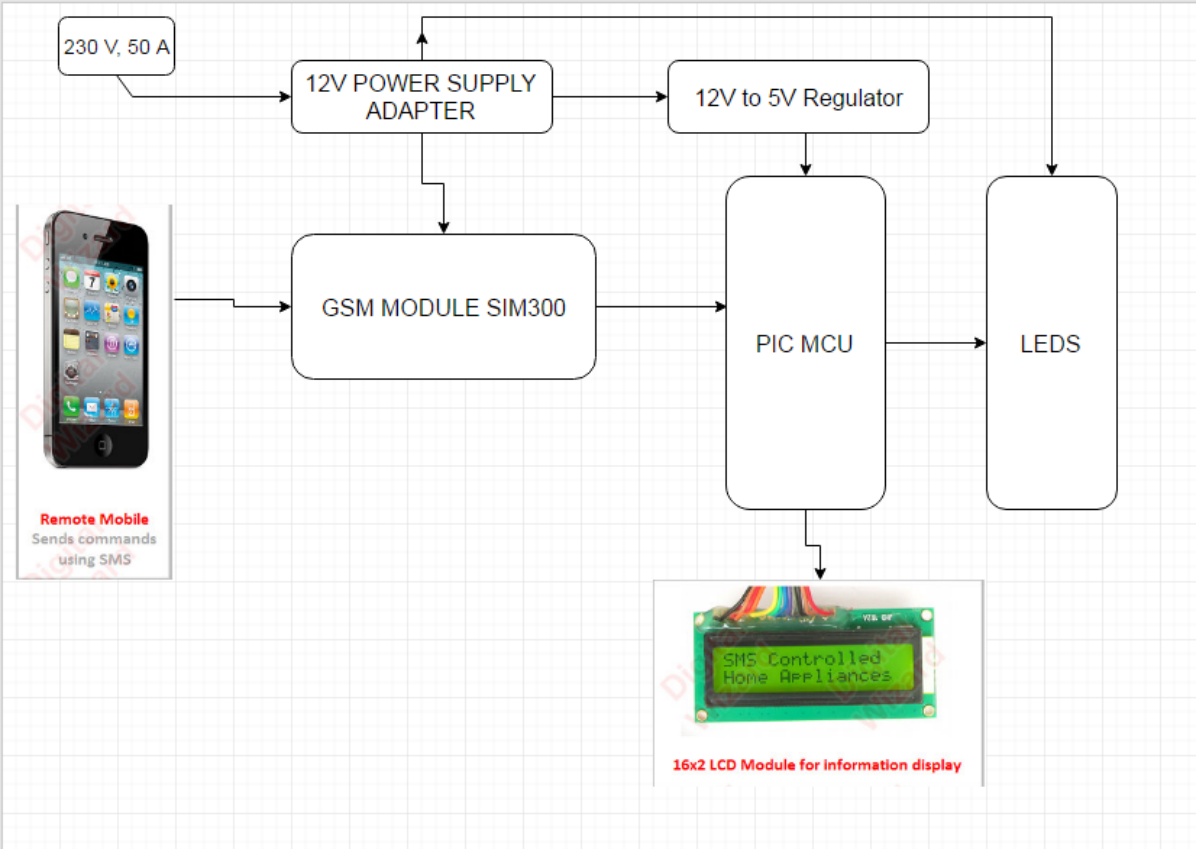
Όπως αναφέρθηκε και στην περίληψη της εργασίας πρόκειται για μια συσκευή, που ασύρματα, θα είναι δυνατή η ενεργοποίηση και η απενεργοποίηση οποιασδήποτε οικιακής συσκευής όπως παραδείγματος χάρη φώτα, ψυγείο, ηλεκτρική κουζίνα κ.α.. Αυτό επιτυγχάνεται στέλνοντας γραπτό μήνυμα (sms) από οποιοδήποτε απομακρυσμένο κινητό τηλέφωνο. Η συσκευή κατασκευάζεται χρησιμοποιώντας μία πλακέτα με έναν μικροελεγκτή PIC18F4520 της Microchip , ένα module με οθόνη LCD για να δείχνει διάφορα μηνύματα προς τον χρήστη , ένα GSM Module και λίγες φωτοδιόδους LED ως προσομοίωση ρελλέ για εναλλαγή εναλλασσομένων 230V (AC) φορτίων. Το GSM Module δέχεται κάρτα SIM , όπως ένα κανονικό κινητό τηλέφωνο , συνδέεται σειριακά με τη πλακέτα του μικροελεγκτή με ενσωματωμένο σύστημα έτσι ώστε να δέχεται αλλά και να στέλνει γραπτά μηνύματα.



Η αρχή λειτουργίας της εφαρμογής φαίνεται εικονικά στο Σχήμα 4.1 και αναλυτικά είναι η εξής :

Αρχικά ο χρήστης στέλνει κωδικοποιημένο γραπτό μήνυμα (sms) στον αριθμό της κάρτας Sim που έχει τοποθετηθεί GSM MODULE ώστε να ανάψει το LED ή να το σβήσει.

Ύστερα το GSM αφού επεξεργάζεται το μήνυμα, το στέλνει στη πλακέτα με τον μικροελεγκτή , εκείνος με τη σειρά του ενημερώνει τον χρήστη μέσω της LCD οθόνης σε ποιο σημείο της διαδικασίας βρίσκεται και παράλληλα επεξεργάζεται το μήνυμα. Μετά την επεξεργασία του μηνύματος , ως έξοδο δίνει 1 ή 0 και έτσι ανάβει ή σβήνει το LED.



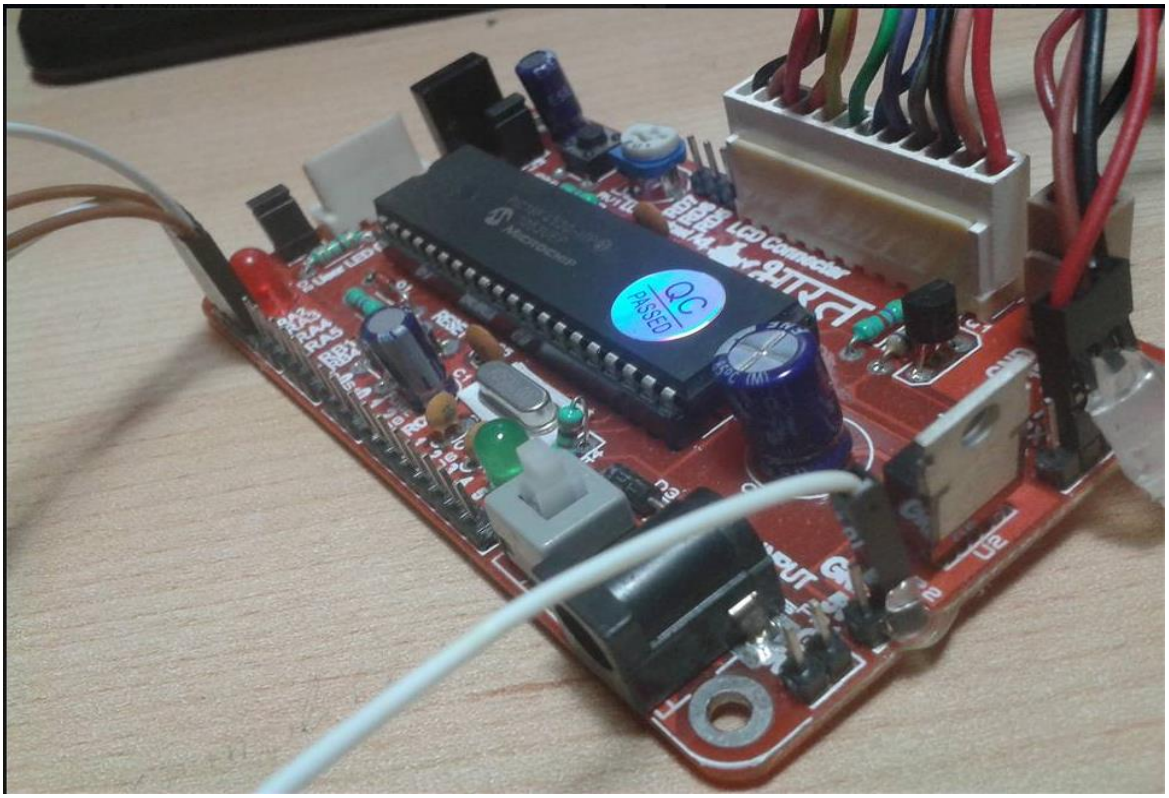
Σχημα 4.1 μπλοκ διάγραμμα ροής

4.1.1 Παρουσίαση των εξαρτημάτων της εφαρμογής

Τα εξαρτήματα που πλαισιώνουν την εφαρμογή είναι μία πλακέτα ειδικά σχεδιασμένη με μικροελεγκτή PIC18F4520, ένα S300 SMS Module, μία hd44780 LCD οθόνη , ένας μετατροπέας εναλλασσόμενης τάσης σε συνεχή (AC to DC Adaptor) και τέσσερα LEDs ως έξοδοι.

4.1.2 Πλακέτα Μικροελεγκτή

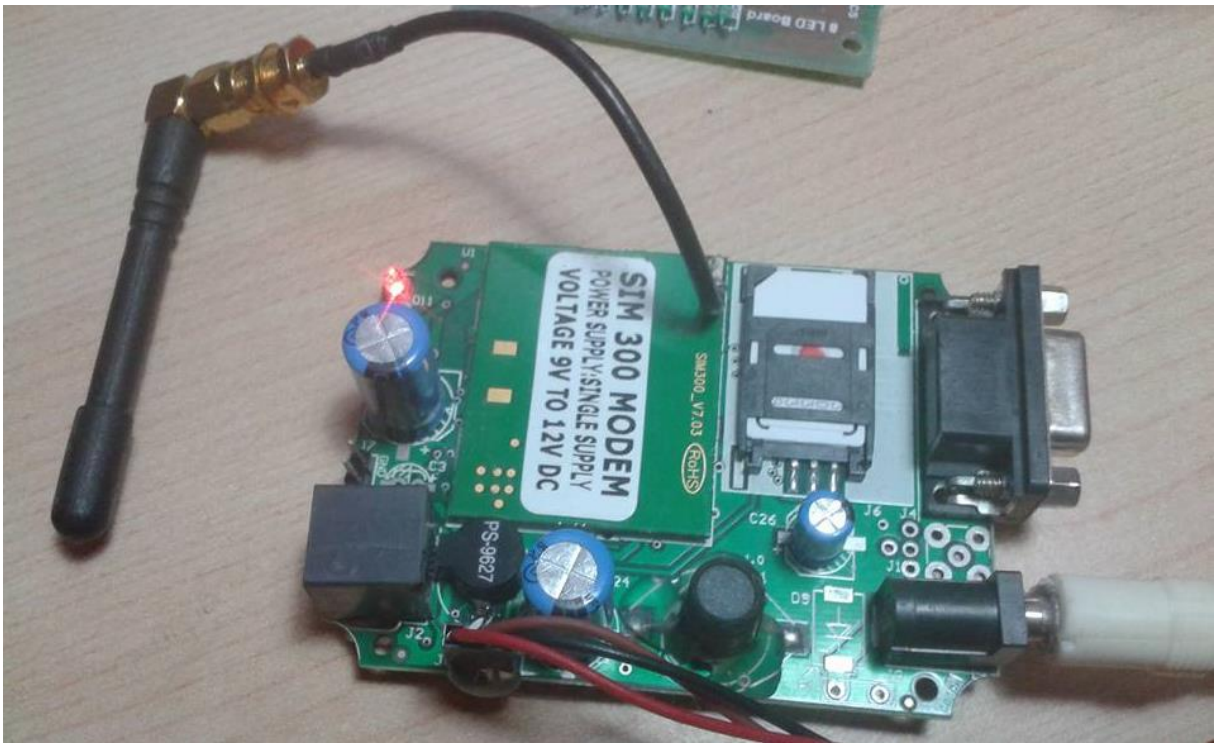
Η πλακέτα φαίνεται στην εικόνα που βρίσκεται από κάτω :



Τα χαρακτηριστικά της πλακέτας αναλύονται στη συνέχεια με γνώμονα την εικόνα 4.2.

4.1.3 SIM300 GSM Module

Όπως φαίνεται στη παρακάτω φωτογραφία, βρίσκεται η πλακέτα σε κατάσταση stand by , ώστε να δεχτεί κάποιο μήνυμα.



Χαρακτηριστικά του SIM300 GSM MODULE :

Κατασκευαστής : SIMCOM

1. Τάση τροφοδοσίας : 9 V - 12 V (Μονή πηγή τροφοδοσίας)

2. Εξοικονόμηση ενέργειας : Τυπική κατανάλωση ρεύματος σε SLEEP MODE στα 2.5 mA

3. Ζώνες Συχνοτήτων : SIM300 Tri-band: EGSM 900, DCS 1800, PCS 1900. Ενώ είναι συμβατό με GSM Phase 2/2 +

4. Κατηγορία GSM : Small MS

5. Μετάδοση ενέργειας :
(1) Κατηγορία 4 (2W) σε EGSM900
(2) Κατηγορία 1 (1W) σε DCS1800 και PCS 1900

6. Συνδεσιμότητα GPRS :

1) GPRS multi-slot τάξη 10

(2) GPRS mobile station τάξη B

- Το GPRS (General Packet Radio Service) είναι μία υπηρεσία μεταφοράς δεδομένων , σύμφωνα με την οποία είναι δυνατή η μεταφορά δεδομένων χρήστη σε πολύ υψηλούς ρυθμούς μετάδοσης, μέσω δικτύου κινητής τηλεφωνίας.

7. Διαστάσεις : 40mm x 30mm x 2,85mm

8. Pins του SIM 300

Table 31: Connection diagrams SIMCOM

PIN NO.	PIN NAME	I/O	PIN NO.	PIN NAME	I/O
2	VBAT	I	1	VBAT	I
4	VBAT	I	3	VBAT	I
6	VBAT	I	5	VBAT	I
8	VBAT	I	7	VBAT	I
10	GND		9	GND	
12	GND		11	GND	
14	GND		13	GND	
16	SIM_PRESENCE	I	15	VRTC	I/O
18	SPI_DATA	I/O	17	VDD_EXT	O
20	SPI_CLK	O	19	SIM_VDD	O
22	SPI_CS	O	21	SIM_I/O	I/O
24	SPI_D/C	O	23	SIM_CLK	O
26	SPI_RST	O	25	SIM_RST	O
28	DCD/GPIO0	O	27	KBC0	O
30	Network LED /GPIO1	O	29	KBC1	O
32	GPIO5	I/O	31	KBC2	O
34	PWRKEY	I	33	KBC3	O
36	Buzzer/GPIO8		35	KBC4	O
38	DTR	I	37	KBR0	I
40	RXD	I	39	KBR1	I
42	TXD	O	41	KBR2	I
44	RTS	I	43	KBR3	I
46	CTS	O	45	KBR4	I
48	RI	O	47	DBGRX	I
50	AGND	I/O	49	DBGTX	O
52	ADC0	I	51	AGND	I/O
54	SPK1P	O	53	MIC1P	I
56	SPK1N	O	55	MIC1N	I
58	SPK2P	O	57	MIC2P	I
60	SPK2N	O	59	MIC2N	I

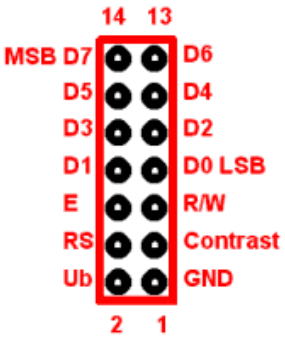
4.1.4 LCD οθόνη hd44780

Η οθόνη , όπως φαίνεται στην παρακάτω φωτογραφία, είναι έτοιμη ώστε να δεχτεί γραπτό μήνυμα.

Κατασκευαστής : Hitachi

Η οθόνη έχει 2 γραμμές που η καθεμία χωράει 16 χαρακτήρες (2X16).

Τα Pins του module της οθόνης είναι τα παρακάτω :



Pin#	Name		In/Out/Pwr
1	GND	Ground	Power
2	VCC	LCD Controller Power (+3 to +5V)	Power
3	VLCD	LCD Display Bias (+5 to -5V *see text)	Analog
4	RS	Register Select: H: Data L: Command	Input
5	R/W	H: Read L: Write	Input
6	E	Enable (Data strobe, active high)	Input
7	DB0	Data LSB	I/O
8	DB1	Data	I/O
9	DB2	Data	I/O
10	DB3	Data	I/O
11	DB4	Data	I/O
12	DB5	Data	I/O
13	DB6	Data	I/O
14	DB7	Data MSB	I/O
15	A	LED Backlight Anode (optional)	Power
16	K	LED Backlight Cathode (optional)	Power

4.1.5 Μετατροπέας τάσης

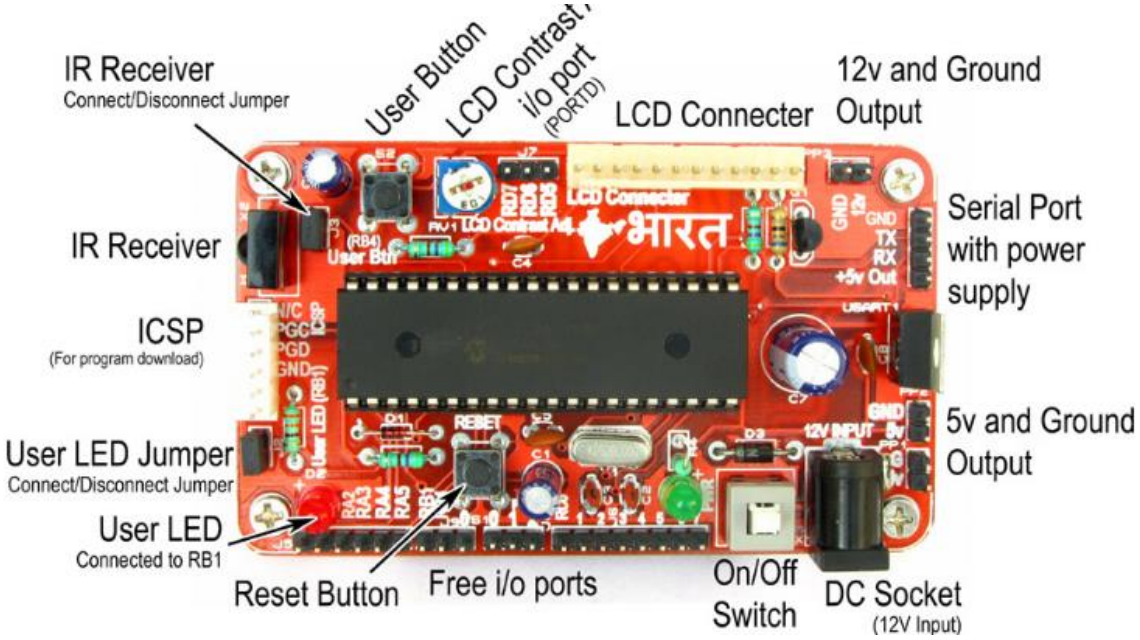
Ο μετατροπέας της τάσης μετασχηματίζει την τάση , δηλαδή είτε την υποβιβάζει είτε την ανυψώνει. Υπάρχουν μετατροπείς που ως είσοδο λαμβάνουν εναλλασσόμενη τάση και τη μετατρέπουν στην έξοδο σε συνεχή και το αντίστροφο. Ο συγκεκριμένος μετατροπέας έχει τα παρακάτω χαρακτηριστικά.

Χαρακτηριστικά

Τάση εισόδου : 90-270 V AC
Ρεύμα εισόδου : 0,8 A
Συχνότητα : 50-60 Hz
Τάση εξόδου : 12 V DC
Ρεύμα εξόδου : 2 A

4.2 Ανάλυση εξαρτημάτων της πλακέτας του μικροελεγκτή

Η πλακέτα είναι εξοπλισμένη με αρκετά εξαρτήματα ώστε ο χρήστης να έχει τη δυνατότητα εύκολου χειρισμού και εύκολης τόσο διαχείρισης όσο και σύνδεσης των περιφερειακών μονάδων.



Σχήμα 4.1

4.2.1 Μικροελεγκτής PIC18F4520

Για την εφαρμογή μας επιλέχθηκε ο μικροελεγκτής της Microchip PIC18F4520 με τα εξής χαρακτηριστικά :

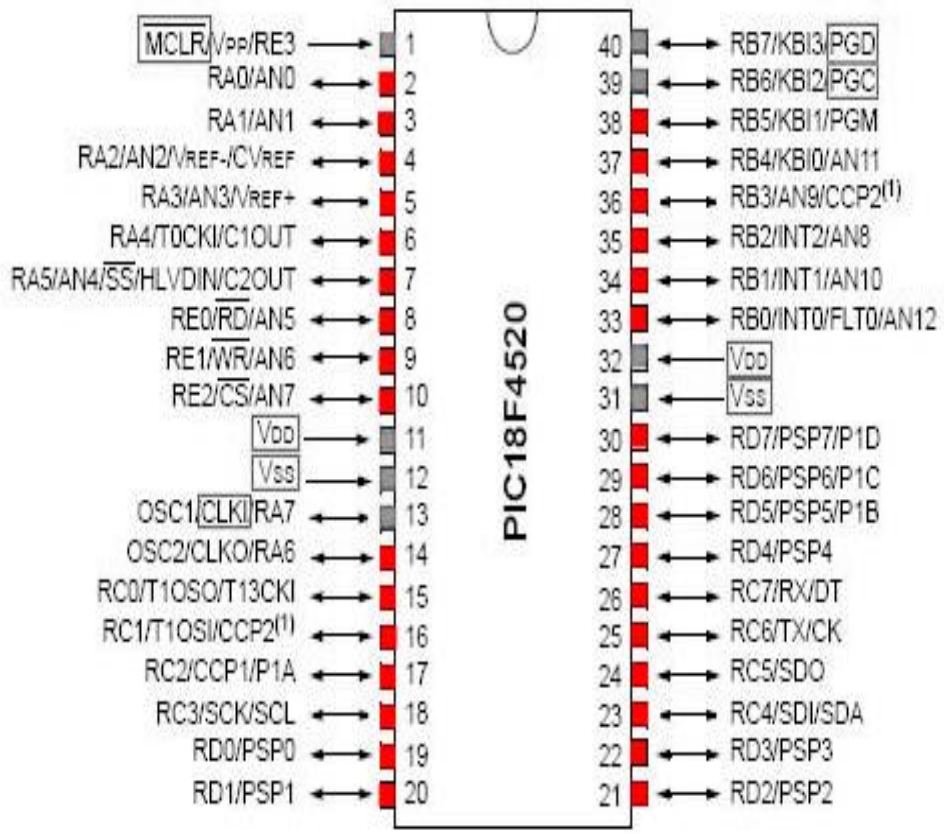
Τύπος Μνήμης προγράμματος	Flash
Μέγεθος μνήμης προγράμματος (KB)	32
Ταχύτητα επεξεργαστή CPU (MIPS)	10
Μέγεθος μνήμης RAM (bytes)	1536

Μέγεθος Data EEPROM (bytes)	256
Ψηφιακή περιφερειακή επικοινωνία	1-UART, 1-A/E/USART, 1-SPI, 1-I2C1-MSSP(SPI/I2C)
Capture/ Compare επικοινωνία	1 CCP, 1 ECCP
Χρονιστές - Timers	1 x 8-bit, 3 x 16-bit
Συγκρητές - Comparators	2
Εύρος Θερμοκρασίας (C)	-40 έως 125
Εύρος Τάσης Λειτουργίας (Volts)	2 έως 5.5
Αριθμός Ακιδών (Pins)	40

Πίνακας 4.1

Διαθέσιμες ακίδες του PIC18F4520

Με την εξαίρεση των θετικών pins παροχής τάσης και της γείωσης , όλες ακίδες του PIC18F4520 μπορεί να χρησιμοποιηθούν ως ψηφιακές I / O (Input/Output) , ωστόσο μερικές άλλες ακίδες (φαίνονται με γκρι χρώμα παρακάτω) χρησιμοποιούνται συνήθως για την επικοινωνία αντί των ψηφιακών I / O.



4.2.2 Υπόλοιπα εξαρτήματα της πλακέτας

Αναλύουμε βλέποντας το Σχήμα 4.2

- DC Socket(12Vinput) : Είναι η είσοδος που τροφοδοτείται η πλακέτα μέσω του μετατροπέα τάσης. Δέχεται τάση 12 Volts και ρεύμα 1 Ampere.
- ON/OFF : Αυτό είναι το μπουτόν ελέγχου ON/OFF της πλακέτας
- Free i/o ports :Αυτές είναι οι ελεύθερες θύρες του μικροελεγκτή που είναι διαθέσιμες για σύνδεση οποιονδήποτε άλλων περιφερειακών εκτός των 4 led που εμείς έχουμε χρησιμοποιήσει για την εφαρμογή μας.
- User LED : Αυτό το LED μπορεί να ενεργοποιηθεί ή να απενεργοποιηθεί υπό τον έλεγχο του λογισμικού. Είναι συνδεδεμένο με PORTB bit 1. Για να το ενεργοποιήσετε, κάνεις τη θύρα I / HIGH. Εάν επιθυμείτε να αποσυνδέσετε αυτό το LED από τον πείρο I / O, μπορείτε να αφαιρέσετε LED Jumper.
- ICSP : Είναι συντομογραφία του In Curcuit Serial Programming και είναι η θύρα που είναι δυνατή η σύνδεση ενός Programmer(Pickit) για να γίνει φόρτωση κώδικα στη πλακέτα.
- User Button : Αυτό το κουμπί μπορεί να χρησιμοποιηθεί για οποιοδήποτε σκοπό που απαιτείται από το χρήστη. Είναι συνδεδεμένο με PORTB bit 4 (δηλαδή RB4).
- LCD Contrast Adjust Preset : Εδώ ρυθμίζεται με τη βοήθεια ενός κατσαβιδιού το πόσο καθαρά μπορεί να δείξει η οθόνη LCD.
- LCD Connector :Είναι η θύρα που συνδέεται η οθόνη LCD.
- 12V and Ground : Ορισμένες εξωτερικές συσκευές χρειάζονται 12v για να λειτουργήσουν, αυτό το σημείο θα σας βοηθήσει να τροφοδοτήσει αυτές τις συσκευές π.χ. το GSM Module.
- 5V Output and Ground : Από εδώ τροφοδοτούνται τα περιφερειακά που χρειάζονται 5V.

5Ο ΚΕΦΑΛΑΙΟ

"ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ"

5.1 Εισαγωγή και ανάλυση κώδικα

Η επιλογή της γλώσσας C για την εφαρμογή έγινε διότι είναι πιο προσιτή από την assembly και θα παρουσιάζονταν πολλά προβλήματα κατά τη διατύπωση του κώδικα, καθώς οι βιβλιοθήκες και οι συναρτήσεις των εξαρτημάτων που χρησιμοποιήθηκαν είναι σε γλώσσα C.

Το πρόγραμμα επομένως είναι γραμμένο σε γλώσσα C και έχει μεταγλωττιστεί χρησιμοποιώντας το μεταγλωττιστή XC8 ο οποίος είναι ελεύθερα διαθέσιμος από την Microchip. ενώ για την δημιουργία του προγράμματος χρησιμοποιήθηκε το MPLAB IDE X .

Το πρόγραμμα κάνει χρήση της βιβλιοθήκης της οθόνης LCD και της βιβλιοθήκης SIM300 Module για την διασύνδεση. Στο κύριο πρόγραμμα χρησιμοποιούμε συναρτήσεις υψηλού επιπέδου για τον έλεγχο της οθόνης LCD και του SMS Module, οι οποίες είναι διαθέσιμες από τους κατασκευαστές.

Αρχικά το πρόγραμμα ξεκινάει με την προετοιμασία της οθόνης και του SMS Module, ώστε να είναι έτοιμα για τους ελέγχους που θα πραγματοποιηθούν στη συνέχεια έτσι ώστε να καταλήξει το σύστημα σε κατάσταση όπου θα είναι έτοιμο για οποιαδήποτε ενέργεια του δώσει εντολή να κάνει ο χρήστης. Όπως ειπώθηκε λοιπόν, εν συνεχεία γίνονται οι απαραίτητοι έλεγχοι όπως εάν είναι συνδεδεμένο το μόντεμ και ανταποκρίνεται σωστά, εάν η κάρτα SIM είναι σωστά τοποθετημένη στο modem και έχει ελεύθερο χώρο για νέο μήνυμα ενώ επίσης εάν το μόντεμ είναι συνδεδεμένο με το δίκτυο ή όχι. Αν το σύστημα εντοπίσει οποιοδήποτε πρόβλημα, ενημερώνει το χρήστη για αυτό, έτσι ώστε αυτός να έχει τη δυνατότητα να προβεί στις ανάλογες διορθώσεις ώστε το σύστημα να είναι σε ορθή λειτουργία.

Εφόσον όλα είναι έτοιμα, το σύστημα είναι σε κατάσταση αναμονής για το κωδικοποιημένο μήνυμα του χρήστη, δηλαδή εάν λάβει μήνυμα ON 1, κάνοντας τους απαραίτητους ελέγχους για τη σωστή ορθογραφία του μηνύματος, ανάβει το LED με νούμερο 1 (στη περίπτωση του μικροελεγκτή κάνει "μετάφραση" του 1 σε 0 διότι τα ποδαράκια του PIC είναι αριθμημένα

από 0 έως 3 ενώ το αλφαριθμητικό μήνυμα για λόγους κατανόησης απαριθμεί τις εξόδους από 1 μέχρι 4). Ομοίως εάν λάβει μήνυμα OFF 1, σβήνει το LED με νούμερο 1.

Στη συνέχεια ελέγχοντας κάποιες παραμέτρους για τα στοιχεία του μηνύματος και του αποστολέα του μηνύματος, ο μικροελεγκτής απαντάει στον αποστολέα για να τον ενημερώσει εάν το αίτημά του πραγματοποιήθηκε ή προέκυψε κάποιο πρόβλημα.

5.1.1 Κώδικας εφαρμογής σε C

Ξεκινάμε με τις απαραίτητες ρυθμίσεις (configurates) ώστε το πρόγραμμά μας και η εκτέλεσή του στην εφαρμογή μας να μην παρουσιάζει προβλήματα. Τα απαραίτητα configs για την εφαρμογή μας επιλέχθηκαν βάσει των χαρακτηριστικών του μικροελεγκτή PIC18F4520 από το manual του και των απαιτήσεων του κώδικάς μας.

Configuration Settings

PIC18F4520

Oscillator Selection:

OSC = LP	LP
OSC = XT	XT
OSC = HS	HS
OSC = RC	RC
OSC = EC	EC-OSC2 as Clock Out
OSC = ECIO6	EC-OSC2 as RA6
OSC = HSPLL	HS-PLL Enabled
OSC = RCIO6	RC-OSC2 as RA6
OSC = INTIO67	INTRC-OSC2 as RA6, OSC1 as RA7
OSC = INTIO7	INTRC-OSC2 as Clock Out, OSC1 as RA7

Fail Safe Clock Monitor:

FCMEN = OFF	Disabled
FCMEN = ON	Enabled

Internal External Osc. Switch Over:

IESO = OFF	Disabled
IESO = ON	Enabled

Power Up Timer:

PWRT = ON	Enabled
PWRT = OFF	Disabled

Brown Out Reset:

BOREN = OFF	Disabled
BOREN = ON	SBOREN Enabled
BOREN = NOSLP	Enabled except SLEEP, SBOREN Disabled
BOREN = SBORDIS	Enabled, SBOREN Disabled

Brown Out Voltage:

BORV = 46	4.6V
BORV = 43	4.3V
BORV = 28	2.8V
BORV = 21	2.1V

Watchdog Timer:

WDT = OFF	Disabled
WDT = ON	Enabled

Watchdog Postscaler:

WDTPS = 1	1:1
WDTPS = 2	1:2
WDTPS = 4	1:4
WDTPS = 8	1:8
WDTPS = 16	1:16
WDTPS = 32	1:32
WDTPS = 64	1:64
WDTPS = 128	1:128
WDTPS = 256	1:256
WDTPS = 512	1:512
WDTPS = 1024	1:1024
WDTPS = 2048	1:2048
WDTPS = 4096	1:4096
WDTPS = 8192	1:8192
WDTPS = 16384	1:16384
WDTPS = 32768	1:32768

MCLR Enable:

MCLRE = OFF	Disabled
MCLRE = ON	Enabled

T1 Oscillator Enable:

LPT1OSC = OFF	Disabled
LPT1OSC = ON	Enabled

Port B A/D Enable:

PBADEN = OFF	Port B<4:0> digital on RESET
PBADEN = ON	Port B<4:0> analog on RESET

CCP2 Mux:

CCP2MX = PORTBE	Muxed with RB3
CCP2MX = PORTC	Muxed with RC1

Stack Overflow Reset:

STVREN = OFF	Disabled
STVREN = ON	Enabled

Low Voltage ICSP:

LVP = OFF	Disabled
LVP = ON	Enabled

XINST Enable:

XINST = OFF	Disabled
XINST = ON	Enabled

Background Debugger Enable:

DEBUG = ON	Enabled
DEBUG = OFF	Disabled

Code Protection Block 0:

CP0 = ON	Enabled
CP0 = OFF	Disabled

Code Protection Block 1:

CP1 = ON	Enabled
CP1 = OFF	Disabled

Code Protection Block 2:

CP2 = ON	Enabled
CP2 = OFF	Disabled

Code Protection Block 3:

CP3 = ON	Enabled
CP3 = OFF	Disabled

Boot Block Code Protection:

CPB = ON	Enabled
CPB = OFF	Disabled

Data EEPROM Code Protection:

CPD = ON	Enabled
CPD = OFF	Disabled

Write Protection Block 0:

WRT0 = ON	Enabled
WRT0 = OFF	Disabled

Write Protection Block 1:

WRT1 = ON	Enabled
WRT1 = OFF	Disabled

Write Protection Block 2:

WRT2 = ON	Enabled
WRT2 = OFF	Disabled

Write Protection Block 3:

WRT3 = ON	Enabled
WRT3 = OFF	Disabled

Boot Block Write Protection:

WRTB = ON	Enabled
WRTB = OFF	Disabled

Configuration Register Write Protection:

WRTC = ON	Enabled
WRTC = OFF	Disabled

PIC18 Configuration Settings Addendum

Table Read Protection Block 0:

EBTR0 = ON	Enabled
EBTR0 = OFF	Disabled

Table Read Protection Block 1:

EBTR1 = ON	Enabled
EBTR1 = OFF	Disabled

Table Read Protection Block 2:

EBTR2 = ON	Enabled
EBTR2 = OFF	Disabled

Table Read Protection Block 3:

EBTR3 = ON	Enabled
EBTR3 = OFF	Disabled

Boot Block Table Read Protection:

EBTRB = ON	Enabled
EBTRB = OFF	Disabled

Κώδικας Προγράμματος

```
#include <xc.h>
#include <string.h>
#include <stdio.h>

#include "lcd_hd44780_pic18.h" // Το αρχείο όπου περιλαμβάνονται βιβλιοθήκες για την lcd οθόνη  και τον μικροελεγκτή, ξαθώς και διάφορες συναρτήσεις συναρτήσεις για να γίνει το interface.
#include "progsm.h" // Το αρχείο όπου υπάρχουν διάφορες συναρτήσεις που χρησιμοποιούνται στους ελέγχους του προγράμματος(λίστες με τα error και με τις καταστάσεις) καθώς και public interface.

// CONFIG1H
#pragma config OSC = HS         // Oscillator Selection bits (HS oscillator)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)
```

```
// CONFIG2L
#pragma config PWRT = OFF       // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 3         // Brown Out Reset Voltage bits (Minimum setting)

// CONFIG2H
#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer1 Oscillator Enable bit (Timer1 configured for higher power operation)
#pragma config MCLRE = ON       // MCLR Pin Enable bit (MCLR pin enabled; RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection bit (Block 0 (000800-001FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection bit (Block 1 (002000-003FFFh) not code-protected)
```

```
#pragma config CP2 = OFF        // Code Protection bit (Block 2 (004000-005FFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection bit (Block 3 (006000-007FFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection bit (Block 0 (000800-001FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection bit (Block 1 (002000-003FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection bit (Block 2 (004000-005FFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection bit (Block 3 (006000-007FFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection bit (Block 0 (000800-001FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from table reads executed in other blocks)
```

```
#pragma config EBTR2 = OFF      // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot block (000000-0007FFh) not protected from table reads executed in other blocks)

//Διακοπή και δήλωση απαραίτητων οκτα-bitων συναρτήσεων .
void Halt();

#define RELAY_LAT   LATA
#define RELAY_TRIS  TRISA

void RelaysInit();
void RelayOn(uint8_t n);
void RelayOff(uint8_t n);

// Δημικουργία ενός delay για μελλοντική χρήση στο πρόγραμμα
void _delay_ms(uint16_t ms) //Δήλωση μιας συνάρτησης που περιέχει χαρακτήρες uint 16 bit
{
   ms=ms/10; // Διαίρεση του ms με το 10
   for(uint16_t i=0;i<ms;i++) // Για όσο το i τύπου( uint16_t ) είναι μικρότερο του ms τότε να αυξάνεται κατά 1
   {
      __delay_ms(10); // Να γίνει καθυστέρηση για 10 ms
   }
}
```

Κύριο πρόγραμμα

Σημειώνεται ότι δίπλα από τον κώδικα και όπου χρειάζεται υπάρχουν τα κατάλληλα σχόλια για την επεξήγηση του κώδικα.

```
void main()
{
      RelaysInit();

      //Initialization - Εδώ γίνεται η αρχικοποίηση
         LCDInit(LS_NONE);

      //LCD Backlight on ( ορίζουμε στα Pin της οθόνης)

      TRISCbits.TRISC2=0;     //Backlight driver pin ως output
      RC2=1;                  //Backlight on

     //Μήνυμα καλωσορίσματος
         LCDClear();
         LCDWriteString("SMS Controlled"); // Χρησιμοποίηση της συνάρτησης
```

LCDWriteString που υπάρχει για να εμφανίζει μηνύματα η οθόνη , ώστε να γράψει το μήνυμα αυτό.

```
         LCDWriteStringXY(0,1,"Home Appliances"); // Χρησιμοποίηση της
```

LCDWriteStringXY ώστε να γράψει στην στην πρώτη γραμμή της οθόνης και ξεκινώντας από τον πρώτο χαρακτήρα το συγκεκριμένο μήνυμα.

```
         _delay_ms(3000); // Καθυστέρηση για 3000 ms ( 3 seconds )

         int8_t r; //Δήλωση της μεταβλητής r ως int8_t
```

```
//Initialize - ( Αρχικοποίηση) του SIM300 module
    LCDClear();
    LCDWriteString("Initializing ..."); // Χρησιμοποίηση της συνάρτησης
```
LCDWriteString που υπάρχει για να εμφανίζει μηνύματα η οθόνη , ώστε να γράψει το μήνυμα αυτό για το GSM που στην ουσία δηλώνει οτι προετοιμάζεται.

```
    r= SIM300Init(); // Δίνεται τιμή στην μεταβλητή r όπου μέσω αυτής δίνει τα error
```
αλλιώς ξεκινάει το gsm

```
    _delay_ms(1000); // Καθυστέρηση για 1000 ms ( 1 second )


//Έλεγχος της κατάστασης της αρχικοποίησης με χρήση της case χρησιμοποιώντας
```
σταθερές ώστε να γίνει σύγκριση αναλόγως με τις τιμες που προκύπτουν.
```
    switch(r)
    {
        case SIM300_OK: // Έλεγχος της σταθερής SIM300_OK
        LCDWriteStringXY(0,1,"OK !"); //Αν όλα είναι οκ εμφανίζει OK
        break;
        case SIM300_TIMEOUT: // Έλεγχος της σταθερής  SIM300_TIMEOUT
        LCDWriteStringXY(0,1,"No response"); // Αν δεν υπάρχει καμία απάντηση
```
εμφανίζει No response
```
        Halt();
        case SIM300_INVALID_RESPONSE: // Έλεγχος της σταθερής
```
SIM300_INVALID_RESPONSE
```
        LCDWriteStringXY(0,1,"Inv response"); //Αν δεν υπάχει ανταπόκριση
```
εμφανίζει Inv response
```
        Halt();
        case SIM300_FAIL: // Έλεγχος της σταθερής  SIM300_FAIL
        LCDWriteStringXY(0,1,"Fail"); // Αν υπάρχει αποτυχία στη σύνδεση
```
εμφανίζει Fail

```
                Halt();
                default:
                LCDWriteStringXY(0,1,"Unknown Error"); // Για οποιοδήποτε άλλο σφάλμα
Unknown Error
                Halt();
        }

        _delay_ms(1000);

   //Ρύθμιση της μορφής του μηνύματος σε κείμενο
        r= SIM300SetTextMode();

        LCDClear();
        LCDWriteString("Set Text Mode ..");

        _delay_ms(1000);









        //Ελεγχος παρουσίας SIM CARD εντός του SIM300 κάνοντας χρήση της If
        LCDClear();
        LCDWriteString("Checking SIMCard");

        _delay_ms(1000);

        r=SIM300IsSIMInserted();

        if (r==SIM300_SIM_NOT_PRESENT)
        {
                //Όταν δεν υπάρχει SIM CARD
                LCDWriteStringXY(0,1,"No SIM Card !");
```

```
                Halt();}


        else if(r==SIM300_TIMEOUT)
        {
//Όταν υπάρχει σφάλμα επικοινωνίας και γενικά οποιοδήποτε άλλο σφάλμα
                LCDWriteStringXY(0,1,"Comm Error !");

                Halt();
        }
        else if(r==SIM300_SIM_PRESENT)
        {
                //Όταν υπάρχει SIM CARD
                LCDWriteStringXY(0,1,"SIM Card Present");

                _delay_ms(1000);
        }
 //Αναζήτηση κενής μνήμης για τα μηνύματα
        for(uint8_t try=0;try<30;try++)
        {
                LCDClear();
                LCDWriteString("Checking Memory");

                r=SIM300GetFreeSMSMemory();

                if(r==SIM300_CMS_ERROR)
                {
//Αν δεν υπάρχει κενή μνήμη για τα μηνύματα

                        LCDWriteStringXY(0,1,"SIM not ready!");
                        LCDWriteIntXY(7,1,sim300_error_code,4);
                }
                else if(r==SIM300_NO_RESPONSE)
```

```
		{
			LCDWriteStringXY(0,1,"No response");
			Halt();
		}
		else if(r>=0)
		{
//Επιτυχής έλεγχος και εύρεση κενής μνήμης
			LCDWriteStringXY(4,1,"slots free"); // Εφόσον έχει βρει μνήμη εμφανίζει στην οθόνη " slots free "
			LCDWriteIntXY(0,1,r,3); // Εμφανίζει την τιμή της r , δηλαδή πόσο μνήμη έχει

			_delay_ms(2000); // Καθυστέρηση για 2000 ms ( 2 seconds )

			if(r==0) Halt(); // Αν είναι γεμάτη η μνήμη τότε το πρόγραμμα κάνει Halt

			break;
		}

		_delay_ms(1500); // Καθυστέρηση 1500 ms ( 1,5 second )

	}



	//Αναζήτηση δικτύου για την sim card κάνοντας χρήση της  επανάληψης while
	LCDClear();
	LCDWriteStringXY(0,0,"SearchingNetwork");

	uint8_t		nw_found=0;
	uint16_t	tries=0;
	uint8_t		x=0;
```

```
while(!nw_found)
{
        r=SIM300GetNetStat();

        if(r==SIM300_NW_SEARCHING)
        {
//Εμφάνιση συνεχόμενων παύλων ως ένδειξη αναμονής για εύρεση δικτύου

LCDWriteStringXY(0,1,"%0%0%0%0%0%0%0%0%0%0%0%0%0%0%0%0");
                LCDWriteStringXY(x,1,"%1");
                LCDGotoXY(17,1);

                x++;

                if(x==16) x=0; // Μετρητής προσπαθειών για να βρει network

                _delay_ms(50);

                tries++;

                if(tries==600)
                break; // Εαν οι προσπάθειες (tries) είναι 600 τότε το πρόγραμμα
διακόπτεται
        }
        else
        break;

}
LCDClear();

//SIM300_NW_REGISTERED_HOME - Για εγχώριο δίκτυο &
SIM300_NW_REGISTED_ROAMING - Για δίκτυο εξωτερικού
```

```
	if(r==SIM300_NW_REGISTERED_HOME ||
r==SIM300_NW_REGISTED_ROAMING) // Έλεγχος τι δίκτυο ( network) ανιχνεύει η sim
card
	{
		LCDWriteString("Network Found");
	}
	else
	{
		LCDWriteString("Cant Connt to NW!");
		Halt();
	}


	_delay_ms(1000);


	LCDClear();

	uint8_t id; //Το id κρατάει τον αριθμό από τον οποίο είχε φτάσει τελευταία φορά
κάποιο μήνυμα
	char oa[20]; //Διεύθυνση Προέλευσης ( Origin Address) : Κρατάει τον αριθμό από τον
οποίο ήρθε το μήνυμα
	uint8_t ref; //Αναφορά του μηνύματος που εστάλη ως απάντηση.


	while(1)
	{

//Αναμονή για κάποιο αίτημα.
/Εμφανίζει , ενώ περιμένει το σύστημα για κάποιο μήνυμα από τον χρήστη ,γεμίζοντας και
αδιάζοντας, την καρδούλα για καθαρά οπτικό θέμα και για διασκεδαστικότερη αναμονή !
		while(SIM300WaitForMsg(&id)!=SIM300_OK)
		{
			if(x)
```

```
                {
                        //Ζωγραφίζει τη γεμάτη καρδιά
                        LCDWriteStringXY(15,0,"%3");

                        //Αλλαγή κατάστασης της καρδιάς σε άδεια
                        x=0;
                }
                else
                {
                        //Ζωγραφίζει την άδεια καρδιά
                        LCDWriteStringXY(15,0,"%4");

                        //Αλλαγή κατάστασης της καρδιάς σε γεμάτη.
                        x=1;
                }

                //Αναμονή για λήψη μηνύματος από τον χρήστη

                LCDWriteStringXY(0,0,"Waiting for SMS");
                LCDWriteStringXY(0,1,"----------------");


        }

//Εμφανίζεται ένα κάτω βέλος για να δείξει ότι έχει έρθει κάποιο μήνυμα
                LCDWriteStringXY(15,0,"%2");

                _delay_ms(1000);

//Τώρα διαβάζει το μήνυμα και κάνει τους απαραίτητους ελέγχους.
                char msg[30];
            char reply_text[80];
```

```
r=SIM300ReadMsg(id,msg,oa); // Δίνεται στην r η συνάρτηση
```
SIM300ReadMsg όπου περιλαμβάνει τον αριθμό μνήμης του τελευταίου μηνύματος, το ίδιο το μήνυμα και τον αριθμο από τον οποίο ήρθε το συγκεκριμένο μήνυμα

```
if(r==SIM300_OK)
{
//Όταν το αίτημα είναι να ανάψει κάποιο led ( ON n )
    if(strnicmp(msg,"ON",2)==0) // Χρήση της strnicmp ώστε να
```
συγκρίνει εάν το στο μήνυμα οι 2 πρώτοι χαρακτήρες είναι ON.
```
    {
  char load_index = msg[3];
```
//Εδώ γίνεται ο έλεγχος και η μετάφραση του 1 σε 0 , του 2 σε 1 , του 3 σε 2 και του 4 σε 3 ώστε ο μικροελεγκτής να καταλαβει ποιο led πρέπει να ανάψει

```
  if(load_index=='1') // Αν το load_index = με τον χαρακτήρα 1
    load_index=0; // Τότε το Load_index = 0 για τον μικροελεγκτή
  else if(load_index=='2')
    load_index=1;
  else if(load_index=='3')
    load_index=2;
  else if(load_index=='4')
    load_index=3;
  else
  {
    load_index=4; // Σε κάθε άλλη περίπτωση το Load_index παίρνει την
```
τιμή 4 (ενώ ο μικροελεγκτής εφόσον έχει συνδεθεί με 4 leds δεν παίρνει τιμή 4)
```
    strcpy(reply_text,"Invalid load number! Range is (1 to 4) only");// Χρήση
```
της strcpy ώστε να γίνει αντιγραφή του μηνύματος ,"Invalid load number! Range is (1 to 4) only" στο reply_text και να έρθει στον χρήστη το μήνυμα αυτό διότι έχει υπάρξει σφάλμα με την εντολή που είχε στείλει ο ίδιος.
```
  }

  if(load_index>=0 && load_index<4) //Αν το load_index είναι μεγαλύτερο ή
```
ίσο με το 0 και μικρότερο από 4

```
                {
                  RelayOn(load_index);// Τότε να ανάψει το led που έχει δωθεί εντολή να
ανάψει
                  sprintf(reply_text,"Load %d turned on successfully",load_index+1);
// Να σταλεί μήνυμα στον χρήστη " ,"Load %d turned on successfully" κάνοντας την
μετατροπή στο load_index προσθέτοντας 1 ώστε να είναι οπτικά ορθό.
                }
              }

              //Όταν το αίτημα είναι να σβήσει κάποιο led ( OFF n )
              else if(strnicmp(msg,"OFF",3)==0)
                    {
                char load_index = msg[4];

//Εδώ γίνεται η μετάφραση του 1 σε 0 , του 2 σε 1 , του 3 σε 2 και του 4 σε 3 ώστε ο
μικροελεγκτής να καταλαβει ποιο led πρέπει να σβήσει.


        if(load_index=='1')
                  load_index=0;
              else if(load_index=='2')
                  load_index=1;
              else if(load_index=='3')
                  load_index=2;
              else if(load_index=='4')
                  load_index=3;
              else
              {
                  load_index=4; // Σε κάθε άλλη περίπτωση το Load_index παίρνει την
τιμή 4 ( ενώ ο μικροελεγκτής εφόσον έχει συνδεθεί με 4 leds δεν παίρνει τιμή 4 )
                  strcpy(reply_text,"Invalid load number! Range is (1 to 4) only");");//
Χρήση της strcpy ώστε να γίνει αντιγραφή του μηνύματος ,"Invalid load number! Range is (1
to 4) only" στο reply_text και να έρθει στον χρήστη το μήνυμα αυτό διότι έχει υπάρξει
σφάλμα με την εντολή που είχε στείλει ο ίδιος.
```

```
                }

                if(load_index>=0 && load_index<4)
                {
                   RelayOff(load_index);
                   sprintf(reply_text,"Load %d turned off successfully",load_index+1); // Να σταλεί μήνυμα στον χρήστη " ,"Load %d turned on successfully" κάνοντας την μετατροπή στο load_index προσθέτοντας 1 ώστε να είναι οπτικά ορθό.
                }
             }
             else
             {
                strcpy(reply_text,"Invalid request! Send ON n or OFF n. Where n is between 1 to 4");
             }
```

```
 //Εμφανίζεται  βέλος στην οθόνη LCD για να δείξει ότι στέλνει ένα μήνυμα.
          LCDWriteStringXY(15,0,"%6");

//Προϋποθέσεις για να σταλεί απάντηση στο κινητό
//Στέλνεται απάντηση
          r=SIM300SendMsg(oa,reply_text,&ref); // τα στοιχεία του κινητού στην r

          if(r==SIM300_OK) // Εάν η r είναι  η σταθερή SIM300_OK
          {
             //Δείχνει το σήμα τικ ότι όλη η διαδικασία έγινε με επιτυχία.
             LCDWriteStringXY(15,0,"%7");
```

```
}
else if(r==SIM300_TIMEOUT) // Εάν η r είναι η σταθερή SIM300_TIMEOUT
{
        // Εαν υπάρχει time out εμφανίζει "Can't Send Reply!" και "Time out !"
   LCDClear();
   LCDWriteStringXY(0,0,"Can't Send Reply!");
   LCDWriteStringXY(0,1,"Time out !");

   _delay_ms(3000);
}
else // Για οποιδήποτε άλλο πρόβλημα εμφανίζει ,"Can't Send Reply!" και "Fail"
{
   LCDClear();
            LCDWriteStringXY(0,0,"Can't Send Reply!");
   LCDWriteStringXY(0,1,"Fail  !");

   _delay_ms(3000);
}

   }
   else // Και τελικά εμφανίζει "Err Reading Msg !"
   {
            LCDClear();
            LCDWriteString("Err Reading Msg !");

            _delay_ms(3000);

   }
```

// Σβήσιμο των μηνυμάτων για να κρατήσει μνήμη για μελλοντική χρήση εφόσον τα μηνύματα αποθηκεύονται στην sim card.

```
if (SIM300DeleteMsg(id)!=SIM300_OK) // Αν η συνάρτηση αυτή έχει την τιμή της σταθεράς SIM300_OK
                {
                        LCDWriteString("Err Deleting Msg !"); // Εμφανίζει " Err Deleting Msg"  και σβήνει το μήνυμα

                        _delay_ms(3000);
                }

                _delay_ms(250);
        }

}
void Halt()
{
        while(1); // Κόβεται το πρόγραμμα και εποστρέφει στην αρχή της επανάληψης while.
}

        void RelaysInit()
{
   RELAY_TRIS&=0xF0;
   RELAY_LAT&=0xF0;
}

void RelayOn(uint8_t n)
{
   RELAY_LAT |=(1<<n);
}
```

```
void RelayOff(uint8_t n)
{
   RELAY_LAT &=~(1<<n);
}
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

Οι στόχοι της εργασίας κατά βάση υλοποιήθηκαν σε καλό βαθμό, εφόσον το σύστημα λειτουργεί. Η κύρια διευκόλυνση που δίνει είναι η εξοικονόμηση χρόνου που στις ημέρες μας ο χρόνος είναι μία μεταβλητή που συνεχώς γίνεται όλο και πιο σημαντική. Επιπροσθέτως προσδίδει μία αίσθηση σιγουριάς σε περιπτώσεις όπου ο χρήστης απομακρυνθεί από το σπίτι του και δεν θυμάται εάν έχει κλείσει κάποιο κύκλωμα.

Συμπερασματικά , όπως αναφέρθηκε, οι στόχοι υλοποιήθηκαν σε καλό βαθμό όμως υπάρχουν περιθώρια βελτίωσης και περαιτέρω ανάπτυξης της εφαρμογής καθώς είναι δυνατή η σύνδεση περισσότερων κυκλωμάτων για καλύτερο έλεγχο αλλά και επίσης είναι δυνατή , κάνοντας κάποιες μετατροπές , η σύνδεση της εφαρμογής με το διαδίκτυο το οποίο πλέον είναι θα έλεγα το πιο σημαντικό εργαλείο της τεχνολογίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Σταμάτης Αλατσαθιανός, "Εισαγωγή στους Μικροελεγκτές PICmicro" , Αθήνα 2007

2. Microchip Technology Inc. , "PICmicro Mid - Range MCU Family" Reference Manual, 1997

3. Ιωάννης Καλομοίρος, "Εισαγωγή στους μικροελεγκτές PIC", ΤΕΙ Κεντρικής Μακεδονίας

4. Γεώργιος Μπάρδης , "Εισαγωγή στη γλώσσα προγραμματισμού C και εφαρμογές", Digital Academy

5. Microchip Technology Inc. , "PIC18F4520 - 8-bit PIC Microcontrollers"

6. Probots - India's DIY Electronics and Robotics Superstore, " SIM300 Datasheet "

7. Hitachi , " HD44780 LCD Datasheet"

8. https://en.wikipedia.org/wiki/C_(programming_language)

Αιγάλεω

Απρίλιος - 2017