ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
Τμήμα Μηχανικών Πληροφορικής (ΤΕ)

Αρχές Προγραμματισμού Πραγματικού Χρόνου

Ιωάννης Καλόμοιρος

Σεπτέμβριος 2015

Άδειες Χρήσης

Χρηματοδότηση

Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

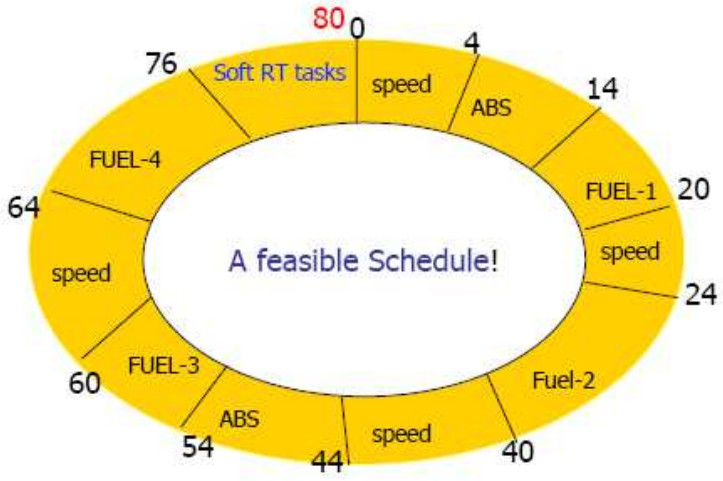
Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ΤΕΙ ΣΕΡΡΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΟΜΕΑΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΒΙΟΜΗΧΑΝΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Αρχές Προγραμματισμού Πραγματικού Χρόνου

Εφαρμογές σε μικρά ενσωματωμένα συστήματα

(Σημειώσεις για τους σπουδαστές του Ζ΄ εξαμήνου)



Ιωάννης Καλόμοιρος, Επ. Καθηγητής

ΣΕΡΡΕΣ 2012

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Οι σημειώσεις αυτές αποτελούν βοηθητικό διδακτικό υλικό για το μάθημα «Προγραμματισμός Συστημάτων σε Πραγματικό χρόνο», που διδάσκεται στους σπουδαστές του Ζ΄ εξαμήνου του τμήματος Πληροφορικής και Επικοινωνιών του ΤΕΙ Σερρών. Το μάθημα εντάσσεται στην κατεύθυνση της Αρχιτεκτονικής Υπολογιστών και έχει σκοπό να εξοικειώσει τους σπουδαστές με τον προγραμματισμό εφαρμογών πραγματικού χρόνου. Η έμφαση δίνεται σε εφαρμογές απλών μικροϋπολογιστικών συστημάτων, όπως είναι οι μικροελεγκτές.

Στο πρώτο κεφάλαιο δίνονται παραδείγματα συστημάτων πραγματικού χρόνου, ώστε να γίνει φανερή η τεράστια διείσδυση και σημασία τους. Στη συνέχεια, στα κεφάλαια 2-5 γίνεται μια εισαγωγή στην αρχιτεκτονική και τον προγραμματισμό μικροελεγκτών. Ιδιαίτερη αναφορά γίνεται στους μικροελεγκτές 8-bit της σειράς 16F της εταιρίας Microchip. Στα κεφάλαια 6 και 7 παρουσιάζονται θέματα διασύνδεσης με αισθητήρες και με κανάλια επικοινωνίας, που αποτελούν αναπόσπαστο μέρος των εφαρμογών πραγματικού χρόνου.

Στο κεφάλαιο 8 γίνεται αναφορά στις βασικές έννοιες συστημάτων πραγματικού χρόνου. Στο κεφάλαιο 9 παρουσιάζονται οι αρχές πολυεπεξεργασίας στα ενσωματωμένα συστήματα καθώς και τα λειτουργικά συστήματα πραγματικού χρόνου (RTOS).

Το παρόν υλικό συμπληρώνεται από εργαστηριακά φύλλα έργου και ασκήσεις, που διανέμονται στους σπουδαστές με τη μορφή φωτοτυπιών.

Ιωάννης Καλόμοιρος

Επ. Καθηγητής ΤΕΙ Σερρών

1. Παραδείγματα συστημάτων πραγματικού χρόνου

1.1 Τι είναι μια εφαρμογή πραγματικού χρόνου

Ένα σύστημα πραγματικού χρόνου έχει ως σκοπό να ολοκληρώνει το έργο του και να αποδίδει τις υπηρεσίες για τις οποίες σχεδιάστηκε, μέσα σε καθορισμένα χρονικά όρια.

Εφόσον, λοιπόν, η εφαρμογή θέτει συγκεκριμένες χρονικές προθεσμίες για την ολοκλήρωση των προβλεπόμενων διεργασιών (tasks), το σύστημα ονομάζεται «σύστημα πραγματικού χρόνου» (real time system).

Συστήματα ψηφιακού ελέγχου, επεξεργασίας σήματος, συλλογής δεδομένων (data acquisition), τηλεπικοινωνιακά συστήματα, συστήματα πολυμέσων, είναι συνήθως συστήματα πραγματικού χρόνου. Βρίσκονται ενσωματωμένα σε εφαρμογές καθημερινής χρήσης και διαθέτουν κρυμμένο κάποιο ισχυρό σύστημα επεξεργασίας δεδομένων, δηλαδή κάποιον μικροϋπολογιστή. Τέτοια ενσωματωμένα συστήματα επεξεργασίας είναι συνήθως μικροελεγκτές ή DSP επεξεργαστές, ολοκληρωμένα κυκλώματα ειδικού σκοπού (ASICs), ή υλικό προγραμματιζόμενης (ή διαμορφούμενης) λογικής (FPGAs), ανάλογα με τον τύπο της εφαρμογής.

Στα συστήματα πραγματικού χρόνου, η ορθότητα των υπολογισμών δεν εξαρτάται μόνον από την ακρίβεια του αποτελέσματος, αλλά και από τον χρόνο κατά τον οποίο παράγεται το αποτέλεσμα. Με άλλα λόγια, στα συστήματα αυτά, ένα καθυστερημένο αποτέλεσμα είναι λανθασμένο αποτέλεσμα.

Ένα πρώτο παράδειγμα συστήματος πραγματικού χρόνου προέρχεται από τον βιομηχανικό έλεγχο. Ας φανταστούμε ένα σύστημα ελεγχόμενο από υπολογιστή, στην γραμμή παραγωγής ενός εργοστασίου εμφιάλωσης. Η λειτουργία του συστήματος είναι να τοποθετεί το καπάκι σε κάθε φιάλη, καθώς αυτή κινείται πάνω σε μια ζώνη μεταφοράς. Η εφαρμογή του πώματος γίνεται με έναν ρομποτικό βραχίονα (manipulator). Αν ο βραχίονας κινηθεί πολύ γρήγορα για να τοποθετήσει το πώμα, η φιάλη δεν θα έχει φτάσει ακόμη στη σωστή θέση. Αν κινηθεί πολύ αργά, η φιάλη θα έχει ήδη απομακρυνθεί αρκετά, ώστε δεν θα μπορεί να την φτάσει ο ρομποτικός βραχίονας. Αν χρειάζεται να σταματούμε την ζώνη μεταφοράς κάθε τόσο, ώστε να προλαβαίνει ο βραχίονας, επιβαρύνουμε σημαντικά το κόστος των εργασιών. Άρα, η ακτίνα της κίνησης του βραχίονα σε συνδυασμό με την ταχύτητα της ζώνης μεταφοράς, δημιουργεί ένα παράθυρο ευκαιρίας για τη μηχανή προκειμένου να τοποθετηθεί το καπάκι.

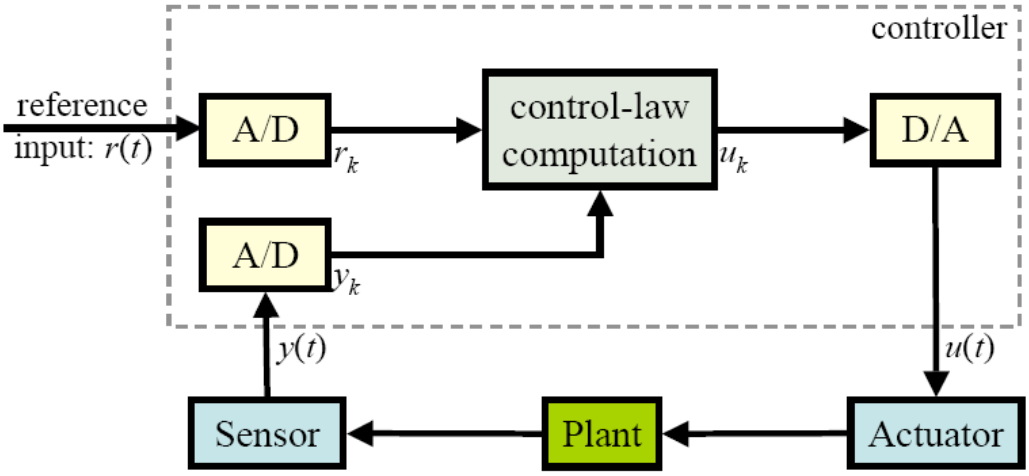
Αυτό το παράθυρο της σωστής λειτουργίας επιβάλει χρονικούς περιορισμούς στην λειτουργία της μηχανής. Ακριβώς αυτή η ύπαρξη χρονικών περιορισμών κάνει ώστε το σύστημα αυτό να θεωρείται πραγματικού χρόνου.

Παρακάτω θα δοθεί η περιγραφή ορισμένων τυπικών εφαρμογών πραγματικού χρόνου.

1.2 Τύποι συστημάτων πραγματικού χρόνου

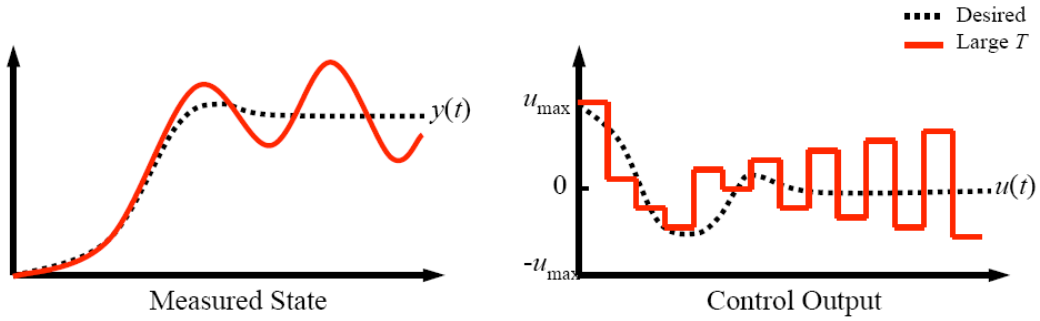
1.2.1 Ψηφιακός ελεγκτής

Κάθε αυτόματο σύστημα ελέγχεται από έναν ελεγκτή, που η δουλειά του είναι να συγκρίνει κάθε στιγμή την πραγματική κατάσταση ενός συστήματος με την επιθυμητή και να παράγει ένα διορθωτικό σήμα που να οδηγεί το σύστημα προς την επιθυμητή κατάσταση. Ένας ελεγκτής μπορεί να υλοποιηθεί και με αναλογικό τρόπο, σήμερα όμως οι ελεγκτές σχεδιάζονται με βάση τις αρχές των ψηφιακών συστημάτων. Για παράδειγμα, ας θεωρήσουμε την επιθυμητή τροχιά ενός ρομποτικού μηχανισμού. Έστω *r(t)* η επιθυμητή τροχιά, που τη σχεδιάζουμε με την βοήθεια υπολογισμών, έχοντας κατά νου πού θέλουμε να οδηγήσουμε στην πραγματικότητα το ρομπότ. Το σήμα *r(t)* είναι, λοιπόν, το «σήμα αναφοράς». Με τη βοήθεια αισθητήρων είναι δυνατό να μετρά το ρομπότ κατά την διάρκεια της κίνησης, ποια είναι η πραγματική του θέση. Έστω *y(t)* η μετρούμενη θέση του ρομποτικού μηχανισμού. Η διαφορά *e(t)=r(t)-y(t)* αντιπροσωπεύει το σφάλμα της κίνησης, δηλαδή την διαφορά ανάμεσα στην επιθυμητή και τη μετρούμενη τιμή. Ο ελεγκτής αναλαμβάνει με βάση το σφάλμα *e(t)* να παράγει ένα διορθωτικό σήμα *u(t)* που θα οδηγήσει τους κινητήρες του ρομπότ κατάλληλα, ώστε να διορθωθεί το σφάλμα της θέσης. Κατόπιν, οι αισθητήρες μετρούν την νέα θέση και ο κύκλος της διόρθωσης επαναλαμβάνεται. Το σχήμα 1.1 δείχνει το διάγραμμα βαθμίδων ενός ψηφιακού ελεγκτή.



Σχ. 1.1 Διάγραμμα βαθμίδων ψηφιακού ελεγκτή

Για την ψηφιακή εκδοχή του ελεγκτή θα πρέπει να λαμβάνονται δείγματα του σήματος αναφοράς *r(t)* και να συγκρίνονται με δείγματα του σήματος που παράγουν οι αισθητήρες της θέσης. Τη λειτουργία αυτή την αποκαλούμε δειγματοληψία των σημάτων. Ένας μετατροπέας αναλογικού σήματος σε ψηφιακό (ADC) αναλαμβάνει να κάνει την μετατροπή των σημάτων του αισθητήρα σε κατάλληλη ψηφιακή μορφή. Έτσι, τα συνεχή σήματα γίνονται διακριτά. Μια σημαντική παράμετρος του ψηφιακού συστήματος, λοιπόν, είναι η **περίοδος T** της δειγματοληψίας. Από την συχνότητα με την οποία λαμβάνουμε δείγματα προκειμένου να εκτελέσουμε τις εργασίες που προαναφέρθηκαν, εξαρτάται η καλή λειτουργία και το κόστος του συστήματος. Αν ο χρόνος T είναι μεγάλος, τότε θα έχουμε ένα φθηνό σύστημα, που όμως δεν θα οδηγείται σωστά. Αν ο χρόνος T είναι πολύ μικρός, τότε θα έχουμε πολύ καλή οδήγηση του ρομπότ, όμως το όλο σύστημα θα έχει μεγάλο κόστος.



Σχ. 1.2 Ένα σήμα ελέγχου u(t) (δεξιά) παράγει μια μεταβολή στη θέση y(t) (αριστερά). Η επιθυμητή απόκριση του συστήματος φαίνεται με τελείες και η απόκριση για μεγάλο T με συνεχή γραμμή.

Σαν ένα παράδειγμα του παραπάνω ελεγκτή, ας σκεφτούμε ένα χειριστήριο που μεταδίδει εντολές προς το σύστημα κατεύθυνσης ενός οχήματος. Το χειριστήριο παράγει τα σήματα αναφοράς στα οποία πρέπει να υπακούει το σύστημα στροφής του οχήματος. Ένας αισθητήρας μετρά την πραγματική στροφή του άξονα. Ανάλογα με τη διαφορά ανάμεσα στην ζητούμενη θέση και στην πραγματική θέση, παράγεται το σήμα οδήγησης *u* που ενεργεί σε ένα μοτέρ και στρίβει τον άξονα.

Όσο πιο γρήγορα πρέπει να αποκρίνεται ένα σύστημα στις αλλαγές του σήματος αναφοράς, τόσο πιο γρήγορα πρέπει να μεταβάλλονται τα σήματα οδήγησης προς τους ενεργοποιητές (μοτέρ, ηλεκτρονόμοι κλπ). Για να συμβαίνει αυτό, η περίοδος δειγματοληψίας πρέπει να είναι μικρή. Αν η περίοδος T είναι μεγαλύτερη από ένα όριο, τότε η χειριστής θα αισθάνεται ότι το σύστημα δεν υπακούει άμεσα στο χειριστήριο και θα δυσκολεύεται να το κατευθύνει. Στο σχ. 1.2 φαίνεται αριστερά η μεταβολή της θέσης ενός ρομποτικού μηχανισμού για μια απότομη (βηματική) μεταβολή του σήματος αναφοράς (π.χ. απότομη μετακίνηση του χειριστηρίου). Επίσης φαίνεται δεξιά η μορφή

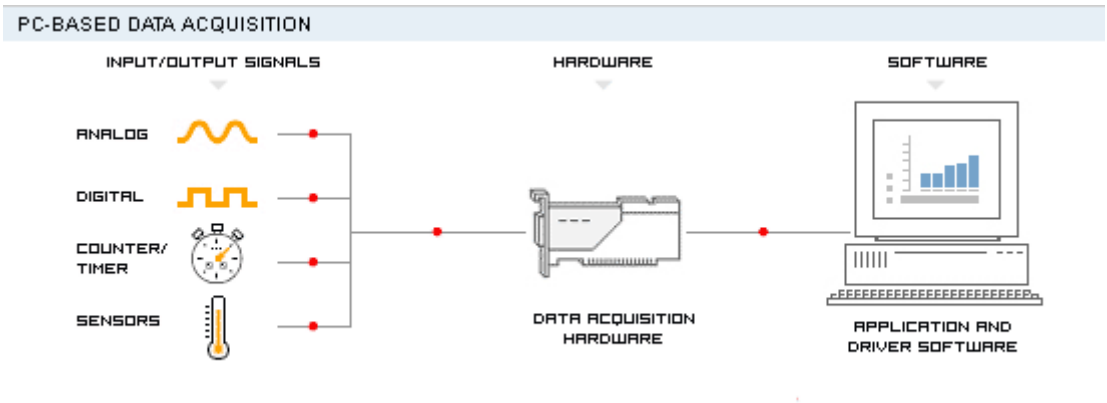
του σήματος οδήγησης *u*. Για μεγάλη περίοδο T το σύστημα θα γίνει ασταθές, όπως φαίνεται στο σχ. 1.2 με την συνεχή γραμμή, οπότε θα ελέγχεται δύσκολα.

Ας σημειωθεί ότι ένας σύνθετος ψηφιακός ελεγκτής μπορεί να περιλαμβάνει υποσυστήματα που το καθένα να χρειάζεται τη δική του περίοδο για τον βρόγχο ελέγχου. Τέτοια συστήματα ονομάζονται πολλαπλού ρυθμού (multirate systems).

1.2.2 Ψηφιακό σύστημα συλλογής δεδομένων

Ένα σύστημα συλλογής δεδομένων αποτελείται από ένα σύνολο αισθητηρίων που μετατρέπουν τα προς μέτρηση φυσικά μεγέθη σε σήματα τάσης ή ρεύματος. Ένας μετατροπέας αναλογικού σήματος σε ψηφιακό κάνει τη μετατροπή της αναλογικής τάσης του αισθητηρίου σε ψηφιακή, λαμβάνοντας δείγματα του αναλογικού σήματος σε τακτά χρονικά διαστήματα. Η μετατροπή αυτή μπορεί να γίνει με τις λεγόμενες κάρτες DAQ που είναι ειδικό υλικό ψηφιακών μετρήσεων. Κατόπιν, η μέτρηση μεταφέρεται προς έναν υπολογιστή, όπου αποθηκεύεται με την μορφή αρχείου ή καταγράφεται σε πραγματικό χρόνο. Επίσης, το σήμα της μέτρησης μπορεί να υποστεί επεξεργασία σε πραγματικό χρόνο.

Στο παράδειγμα αυτό, όπως και στο προηγούμενο, η περίοδος της δειγματοληψίας παίζει σημαντικό ρόλο. Ανάλογα με τη φύση του σήματος που πρέπει να καταγραφεί σε πραγματικό χρόνο, η περίοδος αυτή μπορεί να είναι μικρότερη ή μεγαλύτερη. Για παράδειγμα, αν το σήμα που θέλουμε να καταγράψουμε περιέχει διάφορες συχνότητες από τις οποίες η μεγαλύτερη είναι τα 20 KHz, τότε σύμφωνα με το κριτήριο του Nyquist η συχνότητα της δειγματοληψίας δεν πρέπει να είναι μικρότερη από $2f_{max}$ = 40 KHz. Άρα, η περίοδος της δειγματοληψίας πρέπει να είναι μικρότερη ή ίση από 1/40KHz = 25μs. Αν πάλι το σήμα που θέλουμε να καταγράψουμε μεταβάλλεται πολύ αργά, όπως συμβαίνει, για παράδειγμα, με το σήμα που παράγει ένα θερμόμετρο δωματίου, τότε αρκεί μια περίοδος δειγματοληψίας της τάξης του ενός δευτερολέπτου. Άρα, και πάλι ο σχεδιασμός και το κόστος του συστήματος συνδέονται με τους χρονικούς περιορισμούς



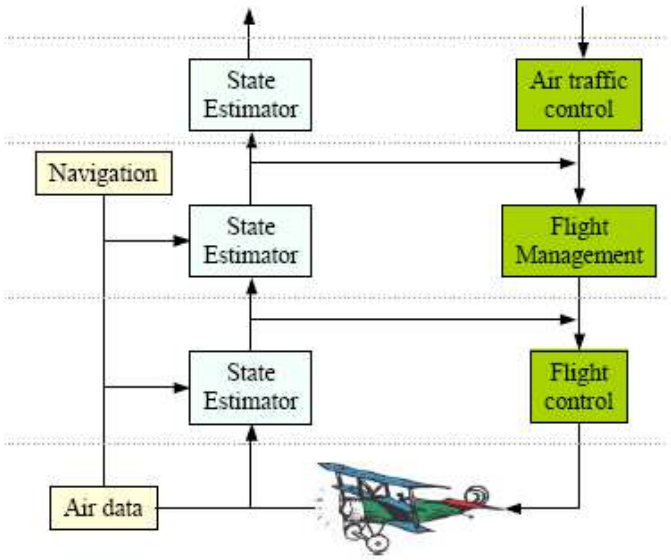
Σχ. 1.3 Ψηφιακό σύστημα συλλογής δεδομένων

της συγκεκριμένης εφαρμογής. Στο σχήμα 1.3 φαίνονται οι βασικές βαθμίδες ενός ψηφιακού συστήματος συλλογής δεδομένων.

Εκτός από το ρυθμό δειγματοληψίας, οι ανάγκες επεξεργασίας σε πραγματικό χρόνο βάζουν τους δικούς τους χρονικούς περιορισμούς. Για παράδειγμα, έστω ότι καταγράφουμε ένα ηχητικό σήμα που προέρχεται από ένα μικρόφωνο. Ας υποθέσουμε ότι επιθυμούμε να παράγουμε το φάσμα Fourier του σήματος σε πραγματικό χρόνο. Με τον τρόπο αυτό θα ξέρουμε πώς μεταβάλλεται το συχνοτικό περιεχόμενο σε κάθε στιγμή. Η ανάλυση Fourier είναι αρκετά απαιτητική υπολογιστική διαδικασία και η εκτέλεση των αντίστοιχων αλγορίθμων είναι χρονοβόρα. Θα πρέπει λοιπόν, να σχεδιάσουμε κατάλληλα τους αλγόριθμους, ώστε να εκτελούνται κατά το δυνατόν γρηγορότερα, μέσα στα χρονικά όρια που επιβάλει η εφαρμογή. Σε ορισμένες περιπτώσεις θα πρέπει να αναθέσουμε αυτό το μέρος της επεξεργασίας σε εξειδικευμένο υλικό.

1.2.3 Ιεραρχικός έλεγχος

Όσο πιο σύνθετο γίνεται το σύστημα ελέγχου, τόσο πιο πολλά επίπεδα ελέγχου χρειάζεται. Πολλές φορές, όλα αυτά τα επίπεδα ελέγχου χρειάζεται να εκτελούνται σε πραγματικό χρόνο, αλλά το καθένα έχει τους δικούς του χρονικούς περιορισμούς. Ένα τυπικό παράδειγμα είναι ο έλεγχος της πτήσης ενός αεροπλάνου, που περιλαμβάνει τα εξής επίπεδα: στο χαμηλότερο επίπεδο υπάρχει ο έλεγχος των πτητικών συστημάτων του αεροσκάφους (flight control), σε αμέσως ψηλότερο επίπεδο υπάρχει το σύστημα διαχείρισης της πτήσης (flight management) και τέλος, στο ανώτερο ιεραρχικά επίπεδο υπάρχει το σύστημα ελέγχου της εναέριας κυκλοφορίας (air-traffic control).



Σχ. 1.4 Ιεραρχικός έλεγχος αεροσκάφους

Ο εναέριος έλεγχος ρυθμίζει την ροή των αεροσκαφών σε κάθε αεροδρόμιο, θέτοντας χρονικά όρια στην άφιξη κάθε αεροσκάφους σε προκαθορισμένα γεωγραφικά σημεία. Το σύστημα διαχείρισης της πτήσης σε κάθε αεροσκάφος χαράζει την καλύτερη δυνατή πορεία του αεροπλάνου, ώστε αυτό να φτάσει στο προκαθορισμένο σημείο στον προβλεπόμενο χρόνο. Φυσικά, λαμβάνει υπόψη του τους κανόνες εναέριας κυκλοφορίας, τον καιρό και την κατανάλωση καυσίμων. Τέλος, ο ελεγκτής πτήσης ρυθμίζει τα υποσυστήματα του αεροπλάνου (ταχύτητα, στροφή, ρυθμός ανόδου/καθόδου) ώστε το αεροπλάνο να ανταποκριθεί στην προβλεπόμενη πορεία.
Προφανώς, τα διάφορα επίπεδα ελέγχου στο παράδειγμά μας έχουν διαφορετικούς χρονικούς περιορισμούς. Συνήθως τα ψηλότερα επίπεδα ελέγχου έχουν λιγότερο αυστηρές προθεσμίες για την εκτέλεση μιας εργασίας. Για παράδειγμα, το στίγμα και οι «μεταβλητές κατάστασης» κάθε αεροσκάφους πρέπει να ανανεώνονται στις οθόνες των ραντάρ περίπου κάθε ένα ή δύο δευτερόλεπτα. Το σύστημα διαχείρισης πτήσης μπορεί να καταστρώνει το σχέδιο πτήσης ακόμη και πριν την απογείωση, ενώ οι σχετικοί υπολογισμοί που είναι απαραίτητοι κατά την διάρκεια της πτήσης έχουν ένα περιθώριο αρκετών δευτερολέπτων ή λεπτών. Οι ψηφιακοί ελεγκτές που ελέγχουν τα πτητικά υποσυστήματα του αεροσκάφους, όμως, έχουν πολύ αυστηρούς περιορισμούς, καθώς οι έλεγχοι πρέπει να γίνονται δεκάδες ή εκατοντάδες φορές το δευτερόλεπτο.

1.2.4 Επεξεργασία σήματος και πολυμεσικές εφαρμογές

Η ψηφιακή επεξεργασία σήματος είναι αναπόσπαστο μέρος εφαρμογών εικόνας και ήχου, που αναφέρονται συχνά ως πολυμεσικές εφαρμογές. Παραδείγματα τέτοιων εφαρμογών είναι η εφαρμογή φίλτρων θορύβου, προκειμένου να απαλλάξουμε το σήμα μας από διάφορους τύπους θορύβου, η εύρεση χαρακτηριστικών (features) και η αναγνώριση προτύπων. Άλλες σημαντικές εφαρμογές είναι ο υπολογισμός μετασχηματισμών που αναδημιουργούν το σήμα στο πεδίο της συχνότητας, όπως ο μετασχηματισμός Fourier και ο μετασχηματισμός DCT, η συμπίεση και η αποσυμπίεση (compression-decompression) δεδομένων κ. ά.

Τυπικά, ένα σύστημα DSP πραγματικού χρόνου υπολογίζει σε κάθε περίοδο δειγματοληψίας μία ή περισσότερες εξόδους *x(k)*, χρησιμοποιώντας ένα σύνολο από *n* δείγματα εισόδου *y(i)*:

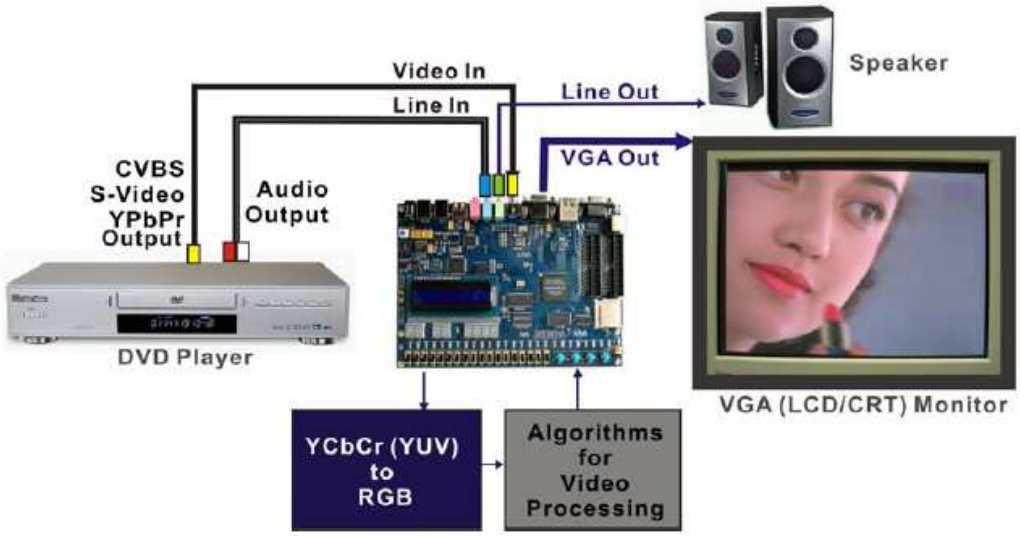
$$x(k) = \sum_{i=1}^{n} a(k,i)\,y(i) \qquad (1.1)$$

Για παράδειγμα, ένα φίλτρο θορύβου, μπορεί να απαλλάξει ένα ηχητικό σήμα από τον υψίσυχνο θόρυβο, υπολογίζοντας για κάθε δείγμα *y(k)* του σήματος το παραπάνω άθροισμα, μέσα σε μια γειτονιά *n* σημείων γύρω από το δείγμα, σύμφωνα με κατάλληλα επιλεγμένο παράθυρο. Οι συντελεστές *α* αποτελούν κατάλληλα επιλεγμένα βάρη, που αλλού τονίζουν και αλλού εξασθενούν τον ρόλο των δειγμάτων μέσα στη

γειτονιά. Ο χρόνος των υπολογισμών για κάθε δείγμα, είναι προφανώς ανάλογος του *n*. Όσο πιο μεγάλο το παράθυρο των υπολογισμών, τόσο μεγαλύτερη η καθυστέρηση.

Αν η εφαρμογή απαιτεί επεξεργασία εικόνας, οι υπολογιστικές απαιτήσεις είναι ακόμη πιο υψηλές. Στην περίπτωση αυτή, οι υπολογισμοί γίνονται στις δύο διαστάσεις, καθώς η ψηφιακή εικόνα είναι ένα διακριτό διδιάστατο σήμα. Έτσι, ένα φίλτρο θορύβου απαιτεί για κάθε εικονοστοιχείο n^2 πολλαπλασιασμούς και προσθέσεις, όπου *n* το εύρος του παραθύρου στη μία διάσταση. Προφανώς, η διδαδικασία που περιγράφουμε είναι μια τυπική συνέλιξη (convolution). Αν πρόκειται για επεξεργασία βίντεο σε πραγματικό χρόνο, οι υπολογισμοί πρέπει να γίνουν για όλο το πλαίσιο (frame) και για 25 πλαίσια το δευτερόλεπτο, σύμφωνα με τις απαιτήσεις του PAL. Αν το πλαίσιο έχει ανάλυση 640x480 εικονοστοιχεία, που είναι η ανάλυση VGA, τότε χρειαζόμαστε $640x480x25xn^2$ πολλαπλασιασμούς και προσθέσεις το δευτερόλεπτο. Σε ένα τυπικό φίλτρο θορύβου εικόνας, το παράθυρο μπορεί να έχει μέγεθος 15x15. Άρα χρειαζόμαστε περίπου $2x10^9$ πολλαπλασιασμούς και προσθέσεις το δευτερόλεπτο, κάτι που αποτελεί ιδιαίτερα μεγάλη απαίτηση, ακόμη και με τα σημερινά υπολογιστικά συστήματα. Ας σημειωθεί ότι οι παραπάνω απαιτήσεις αναφέρονται σε εικόνα αποχρώσεων του γκρι και όχι σε έγχρωμη εικόνα. Στην περίπτωση που έχουμε χρώμα, πρέπει να ληφθούν υπόψη τα τρία επίπεδα χρώματος και οι υπολογιστικές απαιτήσεις αυξάνουν τουλάχιστο κατά μία τάξη μεγέθους.

Οι παραπάνω συλλογισμοί φανερώνουν τις μεγάλες υπολογιστικές απαιτήσεις που υπάρχουν συχνά σε εφαρμογές επεξεργασίας σήματος πραγματικού χρόνου. Όσο πιο περίπλοκη είναι η εφαρμογή, τόσο πιο δύσκολη η υλοποίησή της σε πραγματικό χρόνο.



Σχ. 1.4 Χρήση πλακέτας διαμορφούμενου υλικού (FPGA) για επεξεργασία βίντεο

Συνήθως, οι πολυμεσικές εφαρμογές πραγματικού χρόνου υλοποιούνται με κατάλληλο υλικό (hardware), όπως είναι ειδικοί επεξεργαστές DSP, αφιερωμένα ολοκληρωμένα κυκλώματα ASICs ή επαναδιαμορφούμενο υλικό τύπου FPGAs. Πολλές φορές, επίσης, τέτοιες εφαρμογές υλοποιούνται με κατάλληλο προγραμματισμό σε χαμηλό επίπεδο (assembly), με χρήση ειδικών εντολών των επεξεργαστών MMX, όπως είναι οι διάφοροι Pentium.

Ας σημειώσουμε εδώ, ότι σε πολλές πολυμεσικές εφαρμογές ο χρόνος της επεξεργασίας ανάμεσα σε διαδοχικές εκτελέσεις μιας εργασίας μπορεί να μην είναι πάντα ο ίδιος, αλλά μεταβάλλεται ανάλογα με το περιεχόμενο. Όμως, υπάρχει πάντα μια προθεσμία, που σχετίζεται με τον μέγιστο χρόνο εκτέλεσης, μέσα στην οποία πρέπει να αποδοθεί η εργασία, ώστε να μην μειωθεί με απαράδεκτο τρόπο η ποιότητα της εφαρμογής.

Από την άλλη μεριά, ανάλογα με την εφαρμογή μπορεί να υιοθετούμε διαφορετικά κριτήρια ποιότητας. Για παράδειγμα, άλλες απαιτήσεις έχουμε από την τηλεόραση υψηλής ευκρίνειας (HD) και άλλες από μια απλή τηλεδιάσκεψη. Παρομοίως, άλλες είναι οι απαιτήσεις στον ήχο υψηλής πιστότητας (hi-fi) και άλλες σε μια τηλεφωνική συνομιλία.

1.3 Μερικά ακόμη παραδείγματα

Το ηλεκτρονικό σύστημα ελέγχου της τροφοδοσίας (injection) μιας μηχανής εσωτερικής καύσης είναι ένα σύστημα πραγματικού χρόνου υψηλών απαιτήσεων, χωρίς περιθώρια χρονικών συμβιβασμών. Άλλα παραδείγματα είναι το σύστημα των φρένων ABS (anti-blocking system) σε ένα αυτοκίνητο. Εκεί, ο χρόνος ανάμεσα στο σήμα που δίνει το πηδάλιο φρένου και στην ενεργοποίηση του ηλεκτρονικού μηχανισμού, δεν μπορεί να ξεπερνά τα μερικά χιλιοστά του δευτερολέπτου. Αντίστοιχα, κρίσιμος είναι ο χρόνος ενεργοποίησης ενός αερόσακκου κατά τη διάρκεια μιας σύγκρουσης, όπου οι χρονικές απαιτήσεις είναι απόλυτες, χωρίς περιθώρια λάθους. Επίσης κρίσιμος, αλλά όχι τόσο μικρός, είναι ο χρόνος μέσα στον οποίο πρέπει να ενεργήσει το αυτόματο σύστημα πέδησης για να σταματήσει ένα τρένο, από τη στιγμή που λάβει μια σχετική οπτική ειδοποίηση. Ο χρόνος αυτός δεν είναι απόλυτος, αλλά εξαρτάται από την ταχύτητα του τρένου, το φορτίο του κλπ.

Σε ένα σύστημα ηλεκτρονικών συναλλαγών υπάρχουν επίσης προθεσμίες μέσα στις οποίες πρέπει να έχει ολοκληρωθεί μια συναλλαγή από τη στιγμή που ξεκινά. Εδώ βέβαια δεν κινδυνεύουν ζωές, όμως η ποιότητα της εξυπηρέτησης είναι κι αυτή σημαντικός παράγων.

1.4 Συμπεράσματα

Με βάση τα παραπάνω μπορούμε να κάνουμε τις εξής παρατηρήσεις. Κατ' αρχάς, σύστημα πραγματικού χρόνου δεν σημαίνει αναγκαστικά «γρήγορο» σύστημα. Όπως

είδαμε, άλλες εφαρμογές απαιτούν ολοκλήρωση μιας εργασίας σε χιλιοστά ή εκατοστά του δευτερολέπτου, όπως συμβαίνει στα συστήματα πολυμέσων, ενώ σε άλλες περιπτώσεις ο χρόνος μπορεί να είναι μερικά δευτερόλεπτα ή και λεπτά. Σε όλες τις περιπτώσεις, όμως, υπάρχει χρονικό όριο μέσα στο οποίο το σύστημα πρέπει να έχει ολοκληρώσει μια συγκεκριμένη εργασία, αλλιώς το σύστημα αποτυγχάνει, σε μικρότερο ή μεγαλύτερο βαθμό.

Μια άλλη παρατήρηση αφορά στη φύση της επεξεργασίας που απαιτούν τα συστήματα πραγματικού χρόνου. Είδαμε συστήματα τα οποία εκτελούν κάποιον βρόγχο επεξεργασίας, όπως οι ψηφιακοί ελεγκτές. Οι απαιτήσεις επεξεργασίας σε κάθε επανάληψη του κύκλου παραμένουν λίγο-πολύ οι ίδιες. Τέτοια συστήματα χαρακτηρίζονται ως **περιοδικά**. Μπορούν να υλοποιηθούν επαναλαμβάνοντας αυστηρά τον ίδιο κύκλο επεξεργασίας, λαμβάνοντας περιοδικά δεδομένα από τις εισόδους (polling).

Άλλα συστήματα αντιδρούν σε ασύγχρονα συμβάντα και εκτελούν την εργασία τους μέσα σε συγκεκριμένη προθεσμία, όπως συμβαίνει με το σύστημα του αερόσακκου ή με το σύστημα των φρένων ABS. Τέτοια συστήματα ονομάζονται **απεριοδικά**.

Από την σκοπιά του προγραμματισμού τους, ένα περιοδικό σύστημα μπορεί να υλοποιηθεί με περιοδική συλλογή δεδομένων από τις εισόδους (polling) ενώ ένα απεριοδικό σύστημα μπορεί να υλοποιηθεί με σήματα διακοπών (interrupts). Τέλος, όπως θα δούμε, ένας τρίτος τρόπος υλοποίησης είναι η χρήση λειτουργικού συστήματος πραγματικού χρόνου.

Ας σημειωθεί ότι είναι δυνατό να μετατρέψουμε απεριοδικά συστήματα σε περιοδικά. Αντί το σύστημα να αντιδρά σε ένα εξωτερικό γεγονός την στιγμή που αυτό συμβαίνει, παράγοντας ένα σήμα διακοπής, είναι δυνατό να συλλέγει δεδομένα σε τακτική βάση από τους αισθητήρες, δεκάδες ή εκατοντάδες φορές το δευτερόλεπτο και να ενεργοποιεί την επεξεργασία αν ανιχνευτεί το γεγονός.

Από την σκοπιά των υπολογιστικών συστημάτων για την επεξεργασία των εφαρμογών πραγματικού χρόνου, τα πράγματα ποικίλλουν. Συνήθως, ένα σύστημα πραγματικού χρόνου στηρίζεται σε έναν ή περισσότερους μικροεπεξεργαστές ή αυτόνομους μικροελεγκτές, που είναι ενσωματωμένοι σε μια εφαρμογή. Έτσι υλοποιούνται οι αυτόματοι ψηφιακοί ελεγκτές. Συστήματα πολυμέσων ή επεξεργασίας σήματος στηρίζονται σε DSP επεξεργαστές, ολοκληρωμένα κυκλώματα ASICs ή διαμορφούμενο υλικό (FPGAs). Όμως, μεγάλα κατανεμημένα συστήματα, όπως το σύστημα εναέριας κυκλοφορίας ή τηλεπικοινωνιακά συστήματα, στηρίζονται σε μεγάλα δίκτυα υπολογιστών.

Στα επόμενα κεφάλαια θα προσεγγίσουμε το πρόβλημα της επεξεργασίας στα συστήματα πραγματικού χρόνου κυρίως από την σκοπιά των ενσωματωμένων συστημάτων, δηλαδή από την σκοπιά ενσωματωμένων επεξεργαστών, όπως είναι οι μικροελεγκτές. Για το λόγο αυτό ακολουθεί μια αρκετά εκτεταμένη εισαγωγή στον προγραμματισμό μικροελεγκτών.

2. Εισαγωγή στους μικροελεγκτές

2.1 Τι είναι ένας Μικροελεγκτής

Ένας μικροελεγκτής είναι ένα μικρό *υπολογιστικό κύκλωμα*, σχεδιασμένο σε ένα και μόνο ολοκληρωμένο κύκλωμα υψηλής κλίμακας ολοκλήρωσης. Όπως κάθε υπολογιστικό κύκλωμα, περιέχει κεντρική μονάδα επεξεργασίας, έναν αριθμό καταχωρητών, κυκλώματα μνήμης και κυκλώματα ελέγχου περιφερειακών συσκευών. Κάθε μικροελεγκτής είναι λοιπόν ικανός να ανταλλάξει σήματα με το εξωτερικό περιβάλλον, να εκτελέσει πράξεις ανάμεσα σε μεταβλητές και να καταχωρήσει κάποιες τιμές στη μνήμη RAM που διαθέτει.

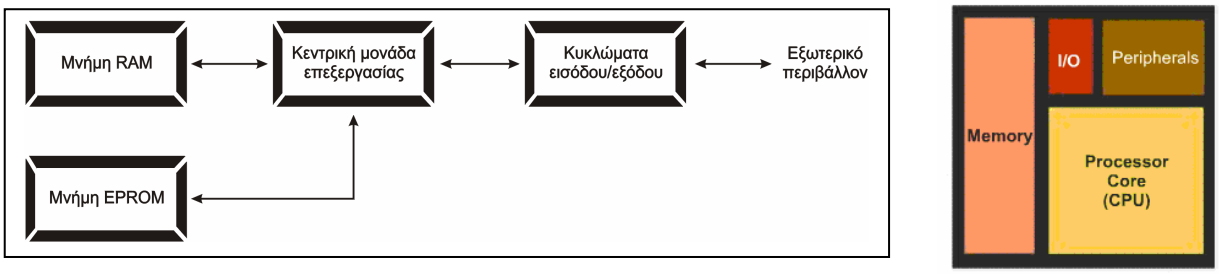
Κάθε μικροελεγκτής περιέχει μέσα σε ένα και μοναδικό ολοκληρωμένο κύκλωμα τα παρακάτω στοιχεία:

- έναν αριθμό από καταχωρητές ειδικού σκοπού (συσσωρευτή, καταχωρητή κατάστασης, μετρητή προγράμματος, καταχωρητή εντολών, καταχωρητή δείκτη).
- εσωτερικούς χρονιστές - απαριθμητές.
- αριθμητική και λογική μονάδα (ALU).
- μονάδα αποκωδικοποίησης εντολών.

Βασικά στοιχεία ενός μικροελεγκτή αποτελούν:

- η μνήμη προγράμματος (ROM ή EPROM) και
- η μνήμη καταχωρητών / μεταβλητών (RAM).

Στο Σχ. 2.1 φαίνεται η βασική δομή ενός μικροελεγκτή.



Σχήμα 2.1 Βασική δομή ενός μικροελεγκτή

Στους μικροελεγκτές διακρίνουμε επίσης

- τα κυκλώματα χρονισμού και ελέγχου

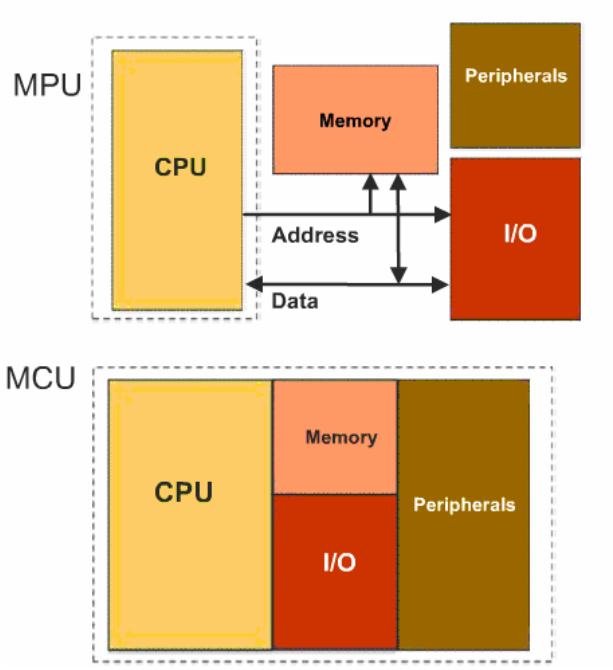
Τέλος, βασικά μέρη ενός μικροελεγκτή είναι

- παράλληλες θύρες εισόδου/εξόδου
- άλλα περιφερειακά κυκλώματα (UART, A/D μετατροπείς κλπ.)

Μέσα από τις θύρες I/O ένας μικροελεγκτής μπορεί να δέχεται σήματα εισόδου με τη μορφή λογικών ψηφιακών καταστάσεων, χαρακτήρες ή bytes δεδομένων με την τεχνική της ασύγχρονης ή της σύγχρονης σειριακής επικοινωνίας, σήματα διακοπών, ή σε ορισμένες περιπτώσεις και αναλογικά σήματα, τα οποία στη συνέχεια μετατρέπονται σε ψηφιακά. Επίσης μπορεί να αποστέλλει σήματα σε άλλες συσκευές μέσα από θύρες εξόδου, να οδηγεί ηλεκτρονόμους, διόδους LED και άλλα κατάλληλα κυκλώματα, που συνήθως περιλαμβάνονται σε κάθε μορφής αυτοματισμό.

Οι μικροελεγκτές χαρακτηρίζονται από ένα περιορισμένο ρεπερτόριο εντολών, οι οποίες μπορούν να γραφούν σε συμβολική μορφή (*assembly*), με τη βοήθεια μνημονικών ονομάτων. Στους μικροελεγκτές PIC μεσαίας τάξης (midrange), το μήκος της εντολής σε γλώσσα μηχανής είναι 14 bits, τα οποία καταχωρούνται στη μνήμη προγράμματος, τύπου EEPROM. Για τα εργαλεία που χρησιμοποιούνται για τον σκοπό αυτό θα μιλήσουμε σε επόμενη παράγραφο.

Σε τι διαφέρει ένας μικροελεγκτής από έναν συνηθισμένο μικροεπεξεργαστή; Ο μικροελεγκτής είναι ένα μικρό *αυτόνομο* υπολογιστικό σύστημα, προγραμματισμένο να



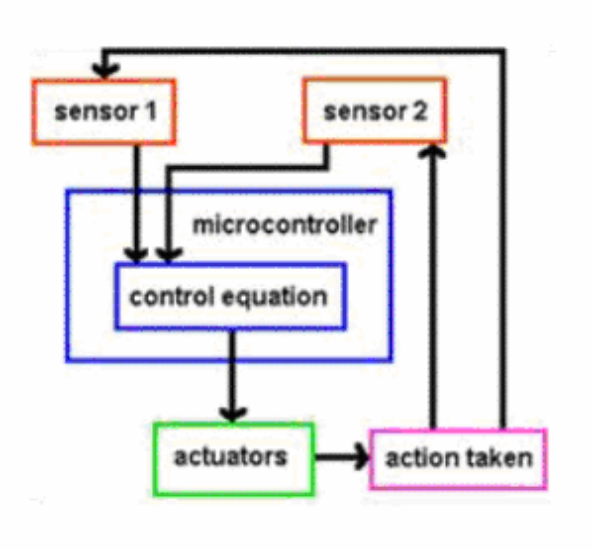
Σχ. 2.2 Διαφορές ανάμεσα σε σύστημα μικροεπεξεργαστή (MPU) και μικροελεγκτή (MCU).

εκτελεί μία συγκεκριμένη λογική ακολουθία εντολών, οι οποίες έχουν καταχωρηθεί στην προγραμματιζόμενη μόνιμη μνήμη του. Κάθε φορά που θα επανεκκινείται ο μικροελεγκτής, θα εκτελεί την ίδια λογική. Θα ανακαλεί τα δεδομένα, θα τα επεξεργάζεται και με βάση τα αποτελέσματα της επεξεργασίας θα ελέγχει το περιβάλλον του. Πρόκειται, δηλαδή, για σύστημα **ειδικού σκοπού**, *αφιερωμένο* (*dedicated*) στον έλεγχο και την εξυπηρέτηση ενός συγκεκριμένου αυτοματισμού.

Αντίθετα, ένας μικροεπεξεργαστής μετά την εκκίνησή του δεν είναι από μόνος του σε θέση να εκτελέσει κάποια λογική ακολουθία. Αν και μπορεί να συνδεθεί με μνήμες RAM και ROM, αυτές αποτελούν ξεχωριστές μονάδες, που συνήθως δεν ολοκληρώνονται μέσα στον ίδιο τον μικροεπεξεργαστή. Οι διαφορές αυτές φαίνονται στο Σχ. 2.2.

Το παρακάτω σχήμα 2.3 δείχνει σχηματικά το βρόχο ελέγχου που εκτελεί ένας μικροελεγκτής, σε μια τυπική ενσωματωμένη εφαρμογή. Ο μικροελεγκτής λειτουργεί ως ψηφιακός ελεγκτής συλλέγοντας δεδομένα από τους αισθητήρες και ενεργοποιώντας εξωτερικά κυκλώματα, όπως κινητήρες, ηλεκτρονόμους, κυκλώματα ισχύος κλπ. Αντίστοιχα με όσα περιγράψαμε στην παράγραφο 1.2.1, ο μικροελεγκτής λαμβάνει ως είσοδο την τρέχουσα κατάσταση του συστήματος που ελέγχει, συνήθως με τη μορφή ενός αναλογικού σήματος, που προέρχεται από έναν ή περισσότερους αισθητήρες. Το σήμα αυτό ψηφιοποιείται, με τη βοήθεια ενός μετατροπέα αναλογικού σήματος σε ψηφιακό και συγκρίνεται με μια τιμή αναφοράς, που περιγράφει την επιθυμητή κατάσταση του συστήματος. Με βάση την απόκλιση της τρέχουσας κατάστασης από την επιθυμητή κατάσταση αναφοράς, λαμβάνεται μια απόφαση και παράγεται ένα σήμα «οδήγησης», που επιδρά πάνω στο ελεγχόμενο σύστημα και μεταβάλλει την κατάστασή του. Για το σκοπό αυτό, ο μικροελεγκτής διαθέτει κατάλληλες θύρες εισόδου/εξόδου, με τις οποίες διασυνδέεται με τον εξωτερικό κόσμο. Επίσης διαθέτει περιφερειακά, όπως κυκλώματα παραγωγής παλμών μεταβαλλόμενου εύρους (PWM) για οδήγηση κινητήρων, μετατροπής αναλογικού σήματος σε ψηφιακό (ADC), θύρες επικοινωνίας κ.ά. Ο ψηφιακός ελεγκτής είναι συνήθως σύστημα πραγματικού χρόνου.

Μια άλλη κατηγορία ενσωματωμένων συστημάτων είναι οι επεξεργαστές ψηφιακού σήματος (Digital Signal Processors ή DSP). Αυτά τα κυκλώματα είναι αφιερωμένα στην γρήγορη επεξεργασία σημάτων, συνήθως ήχου και εικόνας. Ο σκοπός τους είναι να επιτελέσουν μέσα σε συγκεκριμένη χρονική προθεσμία τις μαθηματικές πράξεις που απαιτούνται από τις ανάγκες επεξεργασίας του σήματος. Τέτοιες πράξεις είναι συχνά πολλαπλασιασμοί και αθροίσεις, όπως αυτές της σχέσης (1.1), η οποία περιγράφει ένα τυπικό ψηφιακό φίλτρο. Για το σκοπό αυτό, οι DSP επεξεργαστές διαθέτουν κατάλληλη αρχιτεκτονική και αριθμητικές μονάδες ώστε να εκτελούν με ταχύτητα πολλαπλασιασμούς και συσσωρεύσεις (Multiply Accumulate ή MAC). Εκτός από το κατάλληλο data path οι DSP επεξεργαστές διαθέτουν και περιφερειακά κυκλώματα για διακίνηση δεδομένων μεγάλου όγκου και με την κατάλληλη ψηφιακή κωδικοποίηση, σύμφωνα με πρωτόκολλα που χρησιμοποιούνται για δεδομένα ήχου και



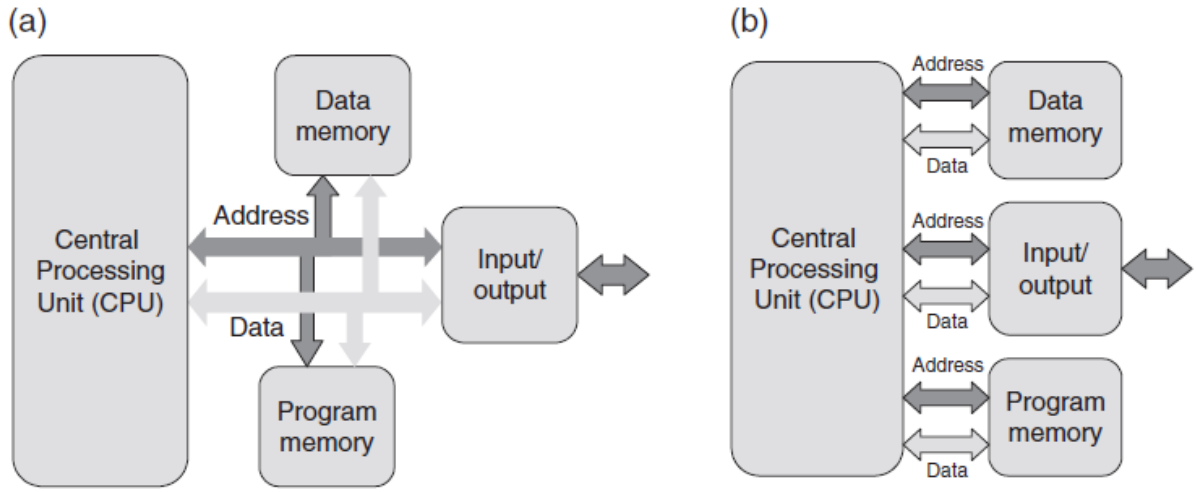
Σχ. 2.3 Σχηματική παράσταση του βρόχου ελέγχου που εκτελεί ένας μικροελεγκτής

βίντεο. Οι DSP επεξεργαστές ανήκουν στην κατηγορία των συστημάτων επεξεργασίας σήματος και πολυμέσων, που εξετάσαμε στην παράγραφο 1.2.4

2.2 Αρχιτεκτονική Harvard

Βασικό χαρακτηριστικό της αρχιτεκτονικής πολλών μικροελεγκτών, που δεν απαντάται στους συνηθισμένους μικροεπεξεργαστές, είναι ότι έχουν διαφορετικό **διάδρομο για τις εντολές** (*instructions bus*) και διαφορετικό **διάδρομο δεδομένων** (*data bus*), για όλα τα υπόλοιπα δεδομένα και αποτελέσματα της επεξεργασίας. Η αρχιτεκτονική αυτή αναφέρεται και ως αρχιτεκτονική Harvard. Βλέπε σχετικά και το Σχήμα 2.4.

Ας σημειωθεί εδώ ότι οι συνηθισμένοι μικροϋπολογιστές είναι δομημένοι σύμφωνα με τη λεγόμενη *αρχιτεκτονική von Neumann*. Σε αυτούς, το πρώτο βήμα για την εκτέλεση μιας λογικής ακολουθίας είναι να καταχωρηθούν στη μνήμη RAM οι εντολές του προγράμματος, μέσω μίας περιφερειακής μονάδας (πληκτρολόγιο ή μαγνητική μνήμη). Το πρόγραμμα, δηλαδή, καταλαμβάνει ένα μέρος στην ίδια μνήμη που διατίθεται επίσης για τα δεδομένα και τα αποτελέσματα της επεξεργασίας. Όταν διακοπεί η τροφοδοσία, τα δεδομένα της μνήμης RAM χάνονται, μαζί με το πρόγραμμα. Την επόμενη φορά ο μικροπεξεργαστής μπορεί να φορτώσει στη μνήμη RAM και να επεξεργαστεί ένα διαφορετικό πρόγραμμα. Οι διάφορες μονάδες του υπολογιστικού συστήματος επικοινωνούν παράλληλα μέσω των ίδιων διαδρόμων δεδομένων, διευθύνσεων και ελέγχου.

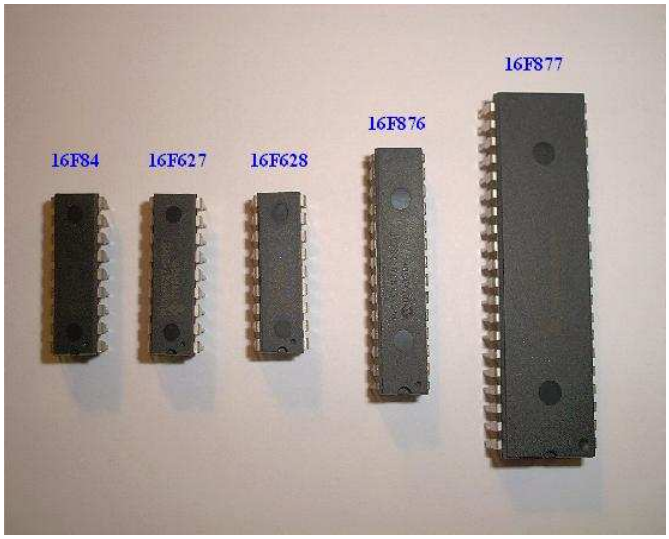


Σχήμα 2.4: Παρουσίαση των δυο τύπων αρχιτεκτονικής α) Ενιαία μνήμη προγράμματος και δεδομένων (αρχιτεκτονική Von-Neumann) β) Ξεχωριστή μνήμη προγράμματος και δεδομένων (αρχιτεκτονική Harvard).

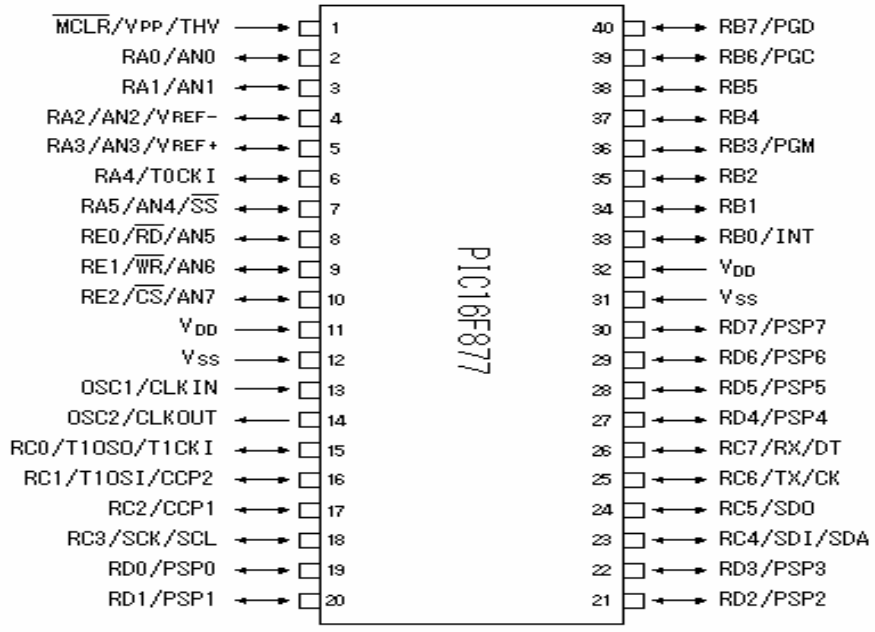
2.3 Οι μικροελεγκτές PIC – Ο PIC 16F877

Στα επόμενα θα αναφερθούμε ειδικά στους μικροελεγκτές PIC της εταιρείας Microchip. Πρόκειται για μία οικογένεια κυκλωμάτων που καλύπτουν ένα εύρος δυνατοτήτων και κόστους. Ειδικότερα, θα μελετήσουμε τους μικροελεγκτές PIC16F87x, που ανήκουν στη μεσαία κατηγορία της οικογένειας. Ένα χαρακτηριστικό που κάνει έναν μικροελεγκτή, όπως τον PIC16F877, ιδιαίτερα ελκυστικό και κατάλληλο για απλές εφαρμογές είναι η ιδιότητα της μνήμης του προγράμματος να διαγράφεται και να επαναπρογραμματίζεται μέσω ηλεκτρικών σημάτων. Με άλλα λόγια, η μνήμη προγράμματος είναι *ηλεκτρικά διαγραφόμενη και προγραμματιζόμενη* (*electrically erasable – programmable*) και γι' αυτό ονομάζεται ***Flash EEPROM*** (η λέξη *flash* δηλώνει ότι η μνήμη λειτουργεί με υψηλή ταχύτητα κατά τη διαγραφή και τον προγραμματισμό). Δεν απαιτείται, δηλαδή, κάποιο ακριβό μηχάνημα διαγραφής των περιεχομένων της μνήμης με τη βοήθεια υπεριώδους ακτινοβολίας, όπως συμβαίνει με άλλες μνήμες. Έτσι, το ίδιο ολοκληρωμένο κύκλωμα μπορεί να χρησιμοποιηθεί για την ανάπτυξη του πρωτότυπου κυκλώματος και του πιλοτικού προγράμματος, καθώς και για την τελική υλοποίηση του κυκλώματος που θα τεθεί σε χρήση ή και θα παραχθεί μαζικά. Η διαγραφή και ο προγραμματισμός του ολοκληρωμένου μπορούν να γίνουν χιλιάδες φορές.

Επιπλέον, ο μικροελεγκτής PIC16F877 έχει τη δυνατότητα να προγραμματίζεται μέσα από εξαιρετικά απλούς **προγραμματιστές**, δηλαδή κυκλώματα που από τη μια μεριά συνδέονται στη σειριακή ή στη θύρα USB ενός προσωπικού υπολογιστή και από την άλλη συνδέονται σε ορισμένους από τους ακροδέκτες (pins) του μικροελεγκτή,



(α)

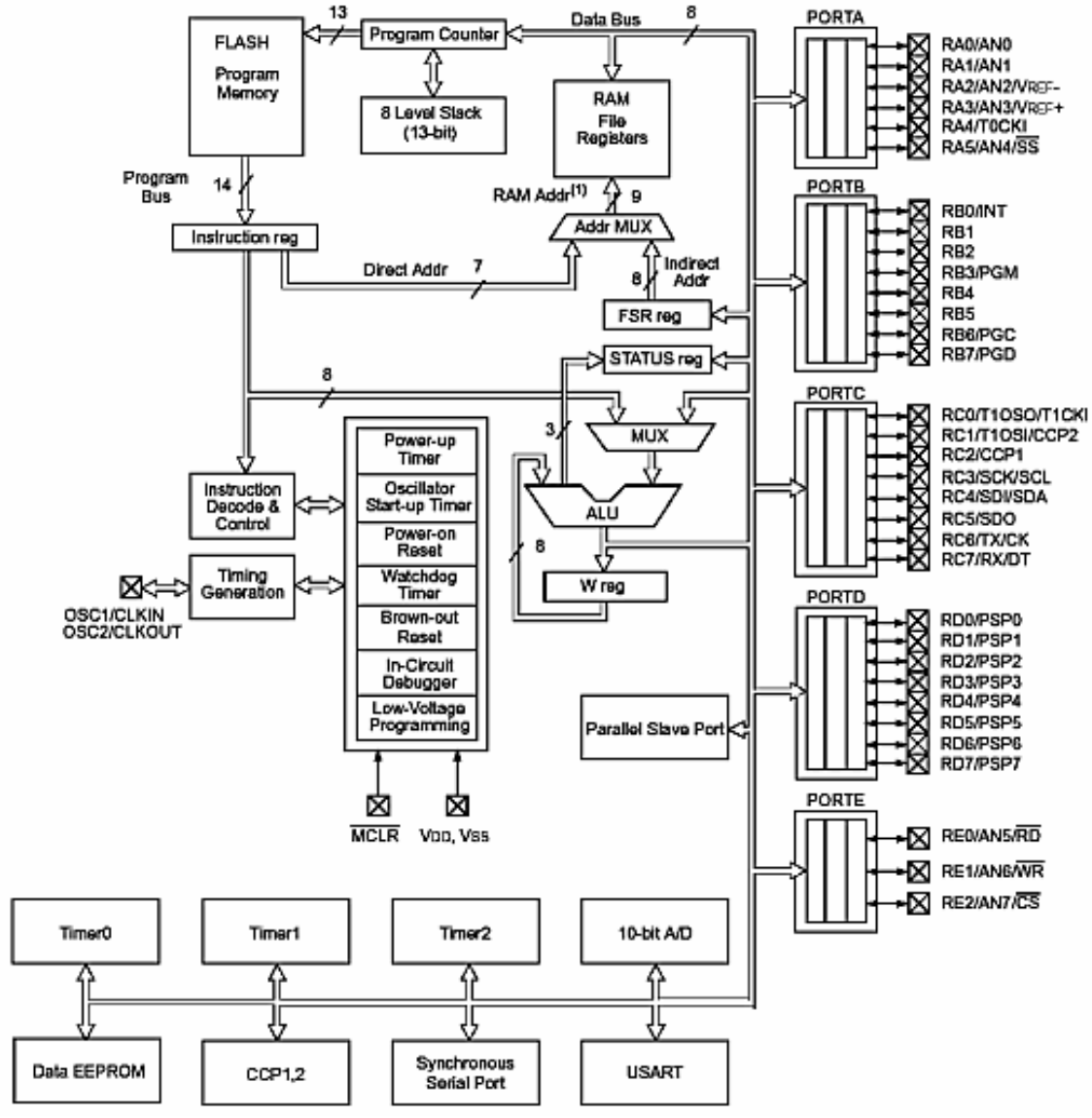


(β)

Σχήμα 2.5 α) **Η** οικογένεια τον μικροελεγκτών PIC 16fXXX και β) Διάγραμμα ακροδεκτών του μικροελεγκτή PIC16F877

προκειμένου να προγραμματίσουν τη μνήμη EEPROM. Η εταιρία Microchip διαθέτει λογισμικό για την ανάπτυξη εφαρμογών καθώς και όλες τις απαραίτητες πληροφορίες για τους μικροελεγκτές PIC, στην ιστοσελίδα της στο Διαδίκτυο, στη διεύθυνση http://www.microchip.com.

Στο Σχήμα 2.5 (β) παρουσιάζεται η αντιστοιχία των 40 συνολικά ακροδεκτών του μικροελεγκτή. Το κύκλωμα διατίθεται σε πλαστική συσκευασία DIP (*Dual In-line Package – διάταξης ακροδεκτών σε δύο σειρές*), είναι τεχνολογίας CMOS και έχει χαμηλό κόστος.



Σχήμα 2.6 Γενικό διάγραμμα της αρχιτεκτονικής του μικροελεγκτή PIC16F877

2.4 Αρχιτεκτονική του Μικροελεγκτή PIC16F877

Όπως όλοι οι μικροελεγκτές PIC, το μοντέλο 16F877 ενσωματώνει την *αρχιτεκτονική Harvard*, της οποίας όπως είδαμε, βασικό χαρακτηριστικό είναι ο διαφορετικός διάδρομος εντολών και δεδομένων και η συνακόλουθη διαφορά των μνημών προγράμματος και καταχωρητών/μεταβλητών. Η αρχιτεκτονική των μικροελεγκτών PIC υπακούσει στους κανόνες της λεγόμενης αρχιτεκτονικής RISC. Ο διάδρομος εντολών έχει εύρος 14 bits. Κάθε εντολή αποτελείται από μία λέξη 14 bits και συνεπώς εκτελείται σε έναν και μοναδικό κύκλο εκτέλεσης, κάτι που αποτελεί βασικό χαρακτηριστικό της αρχιτεκτονικής RISC.

Ο διάδρομος δεδομένων έχει εύρος 8 bits. Ο μικροελεγκτής μπορεί να προγραμματιστεί με 35 συνολικά εντολές και με τη βοήθεια δεκαεπτά καταχωρητών ειδικού σκοπού, που αντιστοιχούν σε συγκεκριμένες θέσεις της μνήμης RAM. Κάθε εντολή εκτελείται σε τέσσερις συνολικά κύκλους του εξωτερικού ωρολογιακού σήματος, που συνιστούν έναν **κύκλο εκτέλεσης** ή **κύκλο εντολής** (*αποκωδικοποίηση εντολής*, *ανάγνωση τελεστέου καταχωρητή*, *επεξεργασία δεδομένων* και *εγγραφή αποτελεσμάτων στον τελικό προορισμό*). Έτσι, εάν ο εξωτερικός ταλαντωτής είναι ένας κρύσταλλος με ιδιοσυχνότητα 4MHz, τότε κάθε εντολή θα εκτελείται σε χρόνο 1 μs (= $^{1}/_{1MHz}$).

Ο μικροελεγκτής PIC16F877 διαθέτει 368 bytes μνήμης RAM, όπου ο χρήστης μπορεί να ορίσει μεταβλητές και να αποθηκεύσει δεδομένα. Επίσης έχει τριαντατρείς ακροδέκτες εισόδου/εξόδου, μοιρασμένους σε πέντε θύρες, που ονομάζονται **PORTA** (6 ακροδέκτες) και **PORTB** (8 ακροδέκτες), **PORTC** (8 ακροδέκτες), **PORTD** (8 ακροδέκτες) και **PORTE** (3 ακροδέκτες), αντίστοιχα. Διαθέτει τρεις χρονιστές, που λειτουργούν και ως απαριθμητές (**Timer0, Timer1, Timer2**) κι έναν ακόμη, εσωτερικά ταλαντούμενο χρονιστή, που επιτηρεί και επαναφέρει τον μικροελεγκτή, σε περίπτωση που βρεθεί εκτός ελέγχου. Ο τελευταίος χρονιστής ονομάζεται **WDT** ή *Watchdog Timer*. Παρακάτω θα αναφερθούμε με κάποια λεπτομέρεια στον χρονιστή **Timer0** (8-bits).

Η μνήμη προγράμματος (EEPROM) του μικροελεγκτή 16F877 έχει χωρητικότητα 2K ή 2048 bytes. Εκτός από τη μνήμη αυτή, ο 16F877 διαθέτει άλλα 64 bytes μνήμης EEPROM, τα οποία προορίζονται για τη μόνιμη αποθήκευση δεδομένων.

Τα βασικά αυτά χαρακτηριστικά της αρχιτεκτονικής του μικροελεγκτή PIC16F877 φαίνονται στο διάγραμμα βαθμίδων του Σχήματος 2.6. Το σχεδιάγραμμα αυτό αναφέρεται σε όλη την οικογένεια των μικροελεγκτών PIC16F87x, ενώ δηλώνονται και οι μεταξύ τους διαφορές.

Ας δούμε μερικά αρχιτεκτονικά στοιχεία με μεγαλύτερη λεπτομέρεια.

2.5 Οργάνωση της Μνήμης RAM

Ένα βασικό θέμα που πρέπει να καταλάβει κανείς, προκειμένου να εργαστεί με κάποιον μικροελεγκτή είναι η δομή της μνήμης RAM. Στον μικροελεγκτή PIC16F877 η μνήμη RAM είναι ένας πίνακας τεσσάρων σελίδων, 128 θέσεων η κάθε μια. Η πρώτη

σελίδα ξεκινά από τη διεύθυνση 0x000 και φτάνει στη διεύθυνση 0x07F (0 έως 127 στο δεκαδικό). Η πρόσβαση στις θέσεις αυτές και η αλλαγή του περιεχομένου τους γίνεται αποκλειστικά μέσω εντολών. Δηλαδή, η μνήμη RAM δεν «φορτώνεται» από περιφερειακές διατάξεις, όπως συμβαίνει στους «κανονικούς» υπολογιστές.

Η εταιρία Microchip ονομάζει τις θέσεις μνήμης «*αρχεία καταχωρητών*» (*files* ή *registers*) και όταν αναφέρεται σ' αυτές, στις διάφορες εντολές της συμβολικής γλώσσας, χρησιμοποιεί το γράμμα **f**. Κάθε θέση μνήμης αποτελείται από 8 bits.

Αφού η μνήμη δεδομένων του PIC16F877 αποτελείται από τέσσερα τμήματα (banks 0-3), με μέγεθος 128 Bytes το καθένα, η συνολική της χωρητικότητα είναι 512 Bytes. Το κάθε τμήμα (σελίδα) μνήμης αποτελείται τόσο από καταχωρητές γενικού όσο και ειδικού σκοπού.

Οι πρώτες θέσεις μνήμης αντιστοιχούν σε εσωτερικούς καταχωρητές, που ρυθμίζουν εσωτερικές λειτουργίες του μικροελεγκτή (Καταχωρητές ειδικού σκοπού - *Special Function Registers*). Μερικοί από τους καταχωρητές ειδικού σκοπού χρησιμοποιούνται για τον έλεγχο του πυρήνα του PIC ενώ άλλοι για τον έλεγχο των περιφερειακών του. Η τιμή τους μπορεί να αλλάξει μέσω εντολών, αλλά ο καθένας καταλαμβάνει συγκεκριμένη θέση στον πίνακα μνήμης RAM. Στις πρώτες αυτές θέσεις δεν μπορούμε να αποθηκεύσουμε δικές μας μεταβλητές.

Οι επόμενες θέσεις (368 συνολικά στον PIC16F877), αντιστοιχούν στην περιοχή *Καταχωρητών Γενικού Σκοπού* (*General Purpose Registers* ή *GPR*). Αυτή είναι η περιοχή όπου μπορούμε να δηλώσουμε και να αποθηκεύσουμε τις δικές μας μεταβλητές.

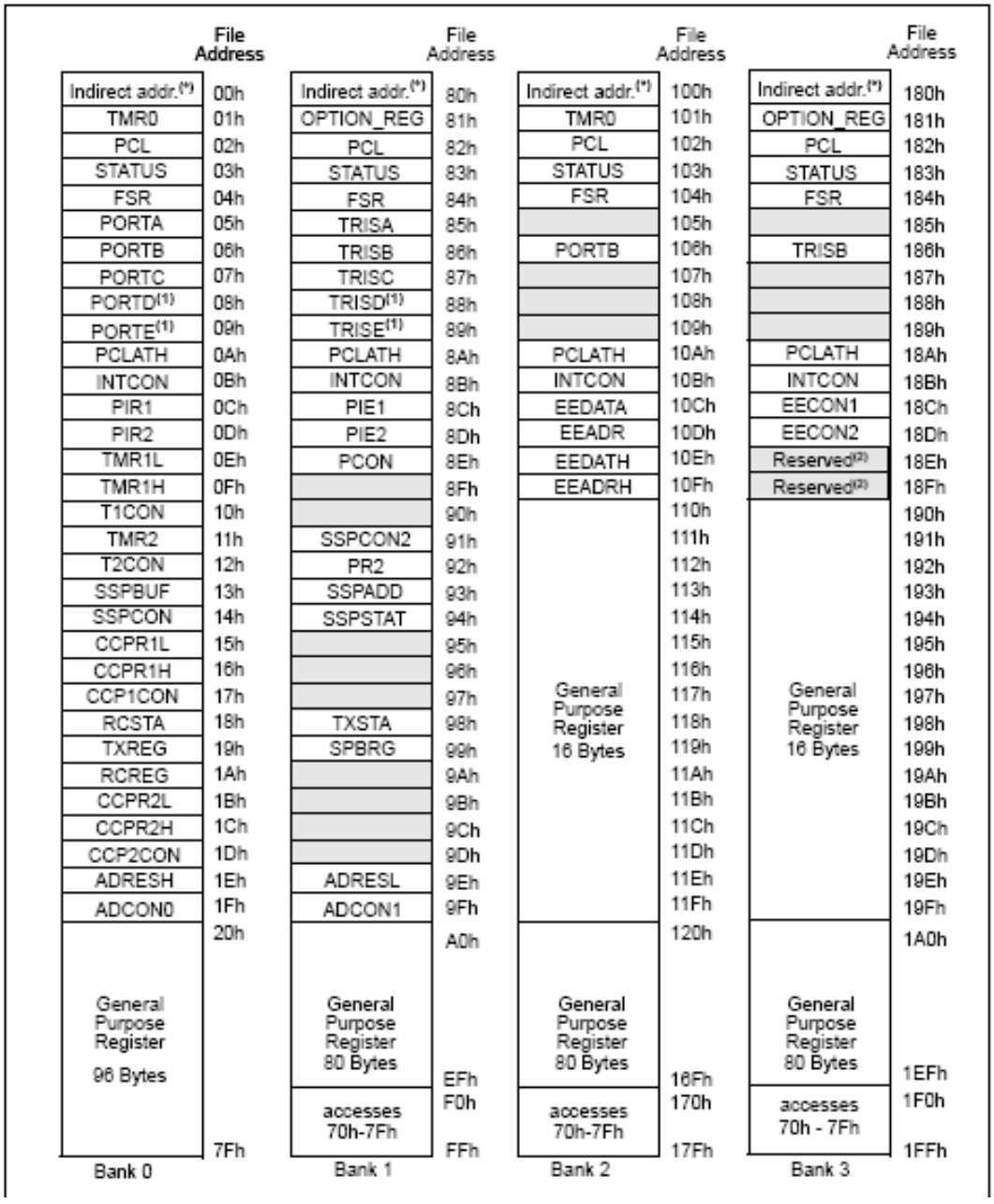
Η διάταξη των θέσεων μνήμης παρουσιάζεται στο Σχήμα 2.7. Το σχήμα αυτό ονομάζεται και *χάρτης της μνήμης*. Στο σχήμα αυτό φαίνεται ένα χαρακτηριστικό της μνήμης που χρειάζεται κάποια προσοχή. Όπως σημειώσαμε, η μνήμη αποτελείται από τέσσερις σελίδες, με έναν αριθμό θέσεων η κάθε μία, οι οποίες είναι χρήσιμες για τον χρήστη. Ορισμένοι από τους καταχωρητές ειδικού σκοπού, όπως ο STATUS και ο FSR χαρτογραφούνται σε περισσότερες από μία σελίδες, και συνεπώς μπορούμε να απευθυνθούμε σ' αυτούς είτε βρισκόμαστε στη μία είτε στην άλλη σελίδα μνήμης. Όμως, για τους περισσότερους από τους καταχωρητές ειδικού σκοπού, που βρίσκονται στις πρώτες διευθύνσεις, έχει σημασία σε ποια σελίδα μνήμης βρισκόμαστε. Έτσι, οι καταχωρητές **OPTION**, **TRISA**, **TRISB**, **TRISC, TRISD, TRISE,** χαρτογραφούνται ***μόνον*** σε ορισμένες σελίδες μνήμης και συνεπώς μπορούμε να τους προσπελάσουμε μόνον μέσα από τις αντίστοιχες διευθύνσεις. Στις αντίστοιχες θέσεις στις άλλες σελίδες υπάρχουν άλλοι ειδικοί καταχωρητές.

Κατά την εκκίνηση, ο μικροελεγκτής βλέπει εξ' ορισμού τη μηδενική σελίδα μνήμης (bank0). Εάν χρειαστεί να προσπελάσουμε καταχωρητές που βρίσκονται στη σελίδα μνήμης 1 (bank1), θα πρέπει να το ορίσουμε αυτό με την εντολή:

BSF STATUS, RP0

η οποία θέτει σε λογικό ένα το bit RP0 του καταχωρητή κατάστασης (*STATUS Register*). Αυτό είναι το bit 5 του καταχωρητή **STATUS**. Το λεπτομερές νόημα της εντολής **BSF f,b**

θα το παρουσιάσουμε σε επόμενη παράγραφο. Μετά την προσπέλαση της σελίδας μνήμης 1 (bank1) οφείλουμε να επιστρέψουμε ξανά στην αρχική σελίδα, όπου βρίσκεται και η πλειονότητα των θέσεων μνήμης που μας χρειάζονται. Ας σημειωθεί εδώ, ότι ο διάδρομος διευθύνσεων (*address bus*) της μνήμης RAM έχει εύρος 7 bits (βλέπε και Σχήμα 2.4). Από τα 14 bits του κωδικού των εντολών, τα επτά προορίζονται για τη δήλωση της διεύθυνσης των καταχωρητών. Άρα, συνολικά μπορούμε να προσπελάσουμε 128 καταχωρητές (2^7), μέσω του διαδρόμου διευθύνσεων της μνήμης



Σχήμα 2.7 Χάρτης της μνήμης RAM του μικροελεγκτή PIC16F877

RAM. Αυτός είναι ο λόγος που, για να προσπελάσουμε τη δεύτερη σελίδα μνήμης RAM (bank1), δηλαδή διευθύνσεις καταχωρητών μεγαλύτερες του 128 (δεκαεξαδικό 0x07F), χρειάζεται να θέσουμε σε λογικό ένα το bit RP0 του καταχωρητή **STATUS**. Δηλαδή, αυτό λειτουργεί ως το περισσότερο σημαντικό bit μιας διεύθυνσης εύρους 8 bits.

2.6 Οι εντολές των μικροελεγκτών PIC.

Όπως αναφέρθηκε, οι μικροελεγκτές PIC διαθέτουν ένα περιορισμένο σύνολο από 35 εντολές, που η κάθε μια αποτελείται από 14 bits. Επειδή και ο διάδρομος εντολών έχει μήκος 14 bits κάθε εντολή μπορεί να εκτελεστεί σε έναν μοναδικό κύκλο εκτέλεσης.
Ο πίνακας 2.1 παραθέτει τις 35 εντολές που αποτελούν τη γλώσσα Assembly των μικροελεγκτών PIC.

Οι εντολές των ελεγκτών PIC μπορούν να χωριστούν σε τέσσερις κατηγορίες.

1. Αριθμητικές εντολές
2. Ελέγχου εκτέλεσης
3. Ελέγχου του μικροεπεξεργαστή
4. Χειρισμού των bit των καταχωρητών

Η πρώτη κατηγορία αποτελείται από τις «αριθμητικές» εντολές, που περιλαμβάνουν τη πράξη της πρόσθεσης και της αφαίρεσης ανάμεσα στα περιεχόμενα καταχωρητών, καθώς και πράξεις αύξησης ή μείωσης των τιμών τους και πράξεις σε επίπεδο bit.

Στην επόμενη κατηγορία εντολών, ανήκουν οι εντολές «ελέγχου εκτέλεσης». Αυτές, τις αποτελούν οι εντολές άλματος (goto), κλήσης υπορουτίνας (call) και οι εντολές επιστροφής (return) από κάποια υπορουτίνα, καθώς επίσης και οι εντολές διακλάδωσης υπό συνθήκη (btfss, btfsc, decfsz κλπ).

Στην επόμενη κατηγορία ανήκουν οι εντολές «ελέγχου του μικροεπεξεργαστή». Οι εντολές αυτές επηρεάζουν βασικά τη λειτουργία του επεξεργαστή και τα κυκλώματα που συσχετίζονται με αυτόν (π.χ. clrwdt, sleep, clrf).

Η τελευταία κατηγορία αποτελείται από τις εντολές «χειρισμού των bit των καταχωρητών» (τοποθέτηση ή μηδενισμός bit). Με τις εντολές αυτές ελέγχουμε άμεσα, κάθε ένα ξεχωριστό bit των καταχωρητών. Η πιο προφανής χρήση των εντολών αυτών είναι ο άμεσος έλεγχος επιμέρους κυκλωμάτων και ακροδεκτών του μικροελεγκτή.

2.7 Τρόποι προσπέλασης της μνήμης

Στο σημείο αυτό αναφέρουμε ότι τις θέσεις της μνήμης RAM μπορούμε να τις προσπελάσουμε με *απευθείας* (*direct*) και *έμμεσο* (*indirect*) τρόπο (ή αλλιώς *διευθυνσιοδότηση*), όπως και στους συνηθισμένους μικροεπεξεργαστές. Στην απευθείας διευθυνσιοδότηση, η διεύθυνση της μνήμης του καταχωρητή που θέλουμε να προσπελάσουμε αναφέρεται με την αριθμητική ή τη συμβολική της ονομασία στη

Πίνακας 2.1: Οι εντολές του μικροελεγκτή PIC

Μνημονικό / Τελεστής		Λειτουργία	Σημαία – Flag που επηρεάζεται
Εντολές χειρισμού ψηφιολέξεων - Byte oriented file register operations			
ADDWF	f, d	Πρόσθεσε το W και το f	C, DC, Z
ANDWF	f, d	Κάνε την λογική πράξη AND ανάμεσα στο W και το f	Z
CLRF	f	Μηδένισε το f	Z
CLRW	-	Μηδένισε το W	Z
COMF	f, d	Φτιάξε το συμπλήρωμα του f και αποθήκευσέ το στο d	Z
DECF	f, d	Μείωσε την τιμή του f	Z
DECFSZ	f, d	Μείωσε την τιμή του f, παρέκαμψε την επόμενη εντολή αν ο f γίνει 0	
INCF	f, d	Αύξησε την τιμή του f	Z
INCFSZ	f, d	Αύξησε την τιμή του f, παρέκαμψε την επόμενη εντολή αν ο f γίνει 0	
IORWF	f, d	Κάνε την λογική πράξη IOR ανάμεσα στο W και το f	Z
MOVF	f, d	Μετέφερε το περιεχόμενο του f	Z
MOVWF	f	Μετέφερε το περιεχόμενο του W στο f	
NOP	-	Εντολή δίχως λειτουργία (απλή χρονική καθυστέρηση ενός κύκλου μηχανής)	
RLF	f, d	Μετέφερε προς τα αριστερά το περιεχόμενο του f μέσο του ψηφίου Carry	C
RRF	f, d	Μετέφερε προς τα δεξιά το περιεχόμενο του f μέσο του ψηφίου Carry	C
SUBWF	f, d	Αφαίρεσε το W από το f	C, DC, Z

SWAPF	f, d	Αντιμετάθεσε τα δύο μισά της ψηφιολέξης (Byte) στο f	
Εντολές χειρισμού ψηφίων - Bit oriented file register operations			
BCF	f, b	Μηδένισε το ψηφίο b του καταχωρητή f	
BSF	f, b	Κάνε λογικό 1 το ψηφίο b του καταχωρητή f	
BTFSC	f, b	Εξέτασε το ψηφίο b του καταχωρητή f, παρέκαμψε την επόμενη εντολή αν είναι 0	
BTFSS	f, b	Εξέτασε το ψηφίο b του καταχωρητή f, παρέκαμψε την επόμενη εντολή αν είναι 1	
Εντολές πράξεων με σταθερούς αριθμούς. Εντολές ελέγχου προγράμματος			
ADDLW	k	Πρόσθεσε τον σταθερό αριθμό k με το W	C, DC, Z
ANDLW	k	Κάνε την λογική πράξη AND ανάμεσα στο k και το W	Z
CALL	k	Κάλεσε την υπορουτίνα k	
CLRWDT	-	Μηδένισε τον επιτηρητή Watchdog Timer	$\overline{TO}$, $\overline{PD}$
GOTO	k	Πήγαινε και εκτέλεσε την εντολή που υπάρχει στην διεύθυνση k	
IORLW	k	Κάνε την λογική πράξη IOR ανάμεσα στο k και το W	Z
MOVLW	k	Μετέφερε το περιεχόμενο του k στο W	
RETFIE	-	Επέστρεψε στην διεύθυνση που ήσουν πριν συμβεί η διακοπή (interrupt)	
RETLW	k	Επέστρεψε από υπορουτίνα και φόρτωσε τον σταθερό αριθμό k στο W	
RETURN	-	Επέστρεψε από υπορουτίνα	
SLEEP	-	Ενεργοποίησε την λειτουργία χαμηλής κατανάλωσης (Sleep - κατανάλωση 2μA)	$\overline{TO}$, $\overline{PD}$
SUBLW	k	Αφαίρεσε το περιεχόμενο του W από το σταθερό αριθμό k	C, DC, Z
XORLW	k	Κάνε την λογική πράξη XOR ανάμεσα στο k και το W	Z

θέση του *τελεστέου* (*operand*). Η σύνταξη της εντολής είναι **movwf *F***, όπου *F* η θέση μνήμης όπου θα καταχωρηθεί το περιεχόμενο του *w*. Στην έμμεση διευθυνσιοδότηση, ο τελεστέος δίνει μια διεύθυνση μνήμης, στην οποία ευρίσκεται αποθηκευμένη η πραγματική διεύθυνση που θέλουμε να προσπελάσουμε.

2.8 Καταχωρητές Ειδικού Σκοπού

Δεν είναι δυνατό να επεκταθούμε σε λεπτομερή περιγραφή όλων των καταχωρητών ειδικού σκοπού (SFR) στο πλαίσιο αυτής της περιληπτικής αναφοράς. Εντελώς εισαγωγικά θα κάνουμε μια νύξη για το νόημα και τη χρήση των πιο απαραίτητων. Αναγκαστικά, ο χρήστης θα πρέπει να ανατρέξει στα εξειδικευμένα εγχειρίδια της Microchip για επιπλέον λεπτομέρειες. Τα 14 bits των εντολών του 16F877, αν και επιτρέπουν την εγγραφή κάθε εντολής σε μια μοναδική *λέξη*, που εκτελείται συνήθως σε έναν κύκλο, οδηγούν σε κάποιους περιορισμούς. Έτσι, δεν μπορούμε να ορίσουμε παραπάνω από έναν καταχωρητή ως τελεστέο σε κάθε εντολή. Παρακάτω θα γνωρίσουμε δύο βασικούς καταχωρητές, που παίζουν καίριο ρόλο σε κάθε εφαρμογή.

2.8.1 Ο καταχωρητής w

Ο καταχωρητής μέσω του οποίου πραγματοποιούνται οι περισσότερες πράξεις και λειτουργίες είναι ο λεγόμενος *καταχωρητής εργασίας* (*working register*), που αναφέρεται ως *καταχωρητής **w***. Αυτός είναι το αντίστοιχο του *συσσωρευτή* (*accumulator*), που χρησιμοποιείται σε δημοφιλείς μικροεπεξεργαστές, όπως τους 6502, 8085 και 8051.

Ο καταχωρητής **w** έχει εύρος 8 bits και δεν διευθυνσιοδοτείται. Μπορούμε να φανταζόμαστε ότι ενσωματώνεται στην αριθμητική μονάδα. Περιλαμβάνεται στις περισσότερες εντολές των μικροελεγκτών PIC, διότι μέσω αυτού γίνονται οι μετακινήσεις δεδομένων και οι πράξεις ανάμεσα στους καταχωρητές. Στον καταχωρητή **w** μπορεί να γίνει προσπέλαση άμεσα και να φορτωθεί κάποιο αριθμητικό δεδομένο, με την εντολή **movlw k**, όπου k ο αριθμός (literal ή κυριολέκτημα). Κάθε αριθμητική πράξη που επιτελείται στον PIC, χρησιμοποιεί τον καταχωρητή **w**. Παραδείγματος χάριν, αν θέλουμε να προσθέσουμε τα περιεχόμενα δύο καταχωρητών, πρέπει να μεταφέρουμε το περιεχόμενο του πρώτου καταχωρητή στον **w** και στη συνέχεια να το προσθέσουμε με το περιεχόμενο του δεύτερου καταχωρητή.

Οι ελεγκτές PIC διαθέτουν αρκετά ισχυρή αρχιτεκτονική από την άποψη ότι το αποτέλεσμα μιας αριθμητικής πράξης μπορεί να αποθηκευτεί ή στον καταχωρητή "**w**", ή στον καταχωρητή προέλευσης των δεδομένων. Αποθηκεύοντας το αποτέλεσμα στον καταχωρητή προέλευσης εξαλείφεται ουσιαστικά η ανάγκη χρήσης πρόσθετων εντολών για την αποθήκευση αυτή. Με το τρόπο αυτό, η μεταφορά των αποτελεσμάτων απλουστεύεται και γίνεται αποδοτικότερη.

2.8.2 Ο καταχωρητής STATUS

Ο καταχωρητής **STATUS** (Καταχωρητής Κατάστασης) αποτελεί το βασικό καταχωρητή που χρησιμοποιείται για τον έλεγχο της εκτέλεσης του προγράμματος. Ο καταχωρητής αυτός χωρίζεται σε τρία τμήματα. Το πρώτο τμήμα περιέχει τις σημαίες (Flags) ή bits κατάστασης της εκτέλεσης (τις "**Z**", "**dc**" και "**C**"). Τα τρία αυτά bits απεικονίζουν τη κατάσταση της εκτέλεσης του προγράμματος. Το bit "**Z**", ή η σημαία του μηδενός (Zero Flag), τίθεται σε λογικό "1" όταν το αποτέλεσμα κάποιας πράξης γίνει ίσο με το μηδέν (add, sub, clear, πράξεις λογικής επεξεργασίας). Η σημαία κρατουμένου (Carry Flag) "**C**" τίθεται σε λογικό "1" όταν το αποτέλεσμα κάποιας πράξης γίνει μεγαλύτερο από 255 (0x0FF), και χρησιμοποιείται για να δηλώσει ότι πρέπει να ενημερωθούν και τα υψηλότερης τάξης bytes που είναι σχετικά με το αποτέλεσμα.

Η σημαία δεκαδικού κρατουμένου (Decimal Carry Flag) "**dc**", τίθεται σε λογικό "1" όταν τα τέσσερα λιγότερο σημαντικά bits (nibble) του αποτελέσματος μιας αριθμητικής πράξης, δώσουν αριθμό μεγαλύτερο από 15.

Αυτά τα bits, που αντιπροσωπεύουν οι σημαίες κατάστασης, μπορούν να διαβαστούν και να εγγραφούν από το χρήστη, ενώ επίσης ενημερώνεται η κατάστασή τους, κάθε φορά που εκτελείται μια εντολή.

Τα επόμενα δύο bits του καταχωρητή **STATUS** δείχνουν το τρόπο με τον οποίο ο επεξεργαστής ανταποκρίθηκε κατά την εκτέλεση της διαδικασίας έναρξης του προγράμματος ή κατά την επιστροφή του από έναν ανενεργό κύκλο (sleep). Ο σκοπός για τον οποίο χρησιμοποιούνται τα bits αυτά είναι να μπορεί το πρόγραμμα της εφαρμογής να αντιλαμβάνεται την αιτία για την οποία ο έλεγχος του επεξεργαστή βρέθηκε στην αρχική θέση εκτέλεσης του προγράμματος.

Τα άλλα δύο bits, «*RP0»* και «*RP1»* χρησιμοποιούνται αποκλειστικά για τη προσπέλαση των δύο υψηλότερων σελίδων της μνήμης. Είναι δυνατή τόσο η ανάγνωση όσο και η εγγραφή σε αυτά. Το bit *"irp",* χρησιμοποιείται για την επιλογή έμμεσης διευθυνσιοδότησης.

Πίνακας 2.2: Ο καταχωρητής STATUS και τα bits που τον αποτελούν

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

3. Γενικά θέματα προγραμματισμού των μικροελεγκτών PIC16F

3.1 Βασικές εντολές αντιγραφής και μεταφοράς δεδομένων

Η εντολή **MOVWF** Reg

αντιγράφει το περιεχόμενο του καταχωρητή εργασίας **w** στον καταχωρητή **Reg**.

Η εντολή **MOVF** Reg, w

αντιγράφει το περιεχόμενο του καταχωρητή **Reg** στον καταχωρητή εργασίας **w**.

Η εντολή **MOVLW** k

μεταφέρει την ποσότητα **k** στον καταχωρητή εργασίας **w**, με τη διαφορά, ότι εδώ το **k** είναι ο συγκεκριμένος αριθμός και δεν αντιπροσωπεύει μία θέση μνήμης, όπως προηγουμένως ο **Reg**. Ο αριθμός k αναφέρεται ως literal ή κυριολέκτημα.

Με βάση τα παραπάνω, η εντολή

MOVLW 0C

μεταφέρει τον δεκαεξαδικό αριθμό 0C στον καταχωρητή εργασίας **w**, ενώ η εντολή

MOVLW Reg

θα μεταφέρει στον **w** όχι το περιεχόμενο της διεύθυνσης **Reg**, αλλά την ίδια τη διεύθυνση **Reg**. Ας ονομάσουμε, λοιπόν, τον καταχωρητή στη διεύθυνση hex0F με το ψευδώνυμο **START**:

START **equ** 0F

Τότε η εντολή

MOVLW START

έχει σαν αποτέλεσμα να μεταφερθεί στον καταχωρητή εργασίας **w** ο αριθμός 0F, ενώ η εντολή

MOVF START, w

Αντιγράφει στον **w** το περιεχόμενο της διεύθυνσης 0F.

3.2 Έλεγχος της ροής του προγράμματος

Η ροή του προγράμματος στους μικροελεγκτές PIC ελέγχεται μέσω εντολών διακλάδωσης χωρίς συνθήκη και μέσω εντολών διακλάδωσης υπό συνθήκη. Επίσης, ελέγχεται με την κλήση υπορουτινών. Τις περιπτώσεις αυτές εξετάζουμε στις επόμενες παραγράφους.

3.2.1 Εντολή goto

Η εντολή goto διακόπτει την ομαλή ροή του προγράμματος και φορτώνει μια νέα τιμή στον απαριθμητή προγράμματος (Program counter ή PC). Η σύνταξη της εντολής είναι:

goto *label* ή **goto** *address*

παράδειγμα:

goto *main* ; πήγαινε στην εντολή με ετικέτα *main*

ή

goto 08 ; πήγαινε στη διεύθυνση προγράμματος 08

3.2.2 Εντολές διακλάδωσης υπό συνθήκη

Οι εντολές αυτές είναι εντολές ελέγχου bit. Δηλαδή, ελέγχουν τη λογική κατάσταση ενός bit κάποιου καταχωρητή και διακλαδίζουν τη ροή του προγράμματος, ανάλογα με το αποτέλεσμα του ελέγχου. Αν η συνθήκη που ελέγχουμε συντρέχει, τότε δεν εκτελείται η αμέσως επόμενη εντολή, αλλά η μεθεπόμενη (“skip”). Αν η συνθήκη δεν συντρέχει, τότε εκτελείται κανονικά η επόμενη εντολή. Τέτοιες εντολές είναι οι εξής:

btfss Reg, b ;κάνε έλεγχο του bit b του καταχωρητή Reg και να παρακάμψεις την

;επόμενη εντολή εάν το bit είναι λογικό 1.

Η εντολή διαβάζεται ως εξής: Bit Test f Skip on Set

Παράδειγμα:

btfss STATUS, Z

Ο έλεγχος του προγράμματος θα παρακάμψει την επόμενη εντολή, εάν η σημαία του μηδενός (Z) στον καταχωρητή **STATUS** είναι 1.

Αντίστοιχα, η εντολή **btfsc** μεταφέρει τον έλεγχο στην μεθεπόμενη εντολή, αν το bit b του καταχωρητή Reg είναι λογικό 0:

btfsc Reg, b ;κάνε έλεγχο στο bit b του καταχωρητή Reg και να παρακάμψεις

;την αμέσως επόμενη εντολή, εάν το bit b είναι λογικό 0.

Η εντολή διαβάζεται ως εξής: Bit Test f Skip on Clear

Παράδειγμα:

btfsc PORTD, 1

Ο έλεγχος του προγράμματος θα παρακάμψει την επόμενη εντολή εάν ο ακροδέκτης RD1 της PORTD είναι 0.
Ο συνδυασμός των παραπάνω εντολών με μια εντολή **goto $-1** (πήγαινε στην αμέσως προηγούμενη εντολή), δημιουργεί ένα βρόχο ελέγχου στον οποίο παγιδεύεται η εκτέλεση του προγράμματος έως ότου εκπληρωθεί η συνθήκη:

btfsc PORTD, 1

goto $-1

επόμενη εντολή...

Στον παραπάνω βρόχο θα γίνεται διαρκώς ο έλεγος της λογικής κατάστασης του ακροδέκτη RD1, μέχρι να βρεθεί ότι ο ακροδέκτης έλαβε λογικό 0. Τότε, η εντολή **goto $-1** θα παρακαμφθεί και η εκτέλεση του προγράμματος θα μεταβεί στην *επόμενη εντολή* και θα συνεχίσει εκτελώντας με τη σειρά όσες έπονται.

Δύο ακόμη εντολές διακλάδωσης υπό συνθήκη, αυξάνουν και μειώνουν, αντίστοιχα, το περιεχόμενο ενός καταχωρητή *f* και παρακάπτουν την επόμενη αντολή αν το αποτέλεσμα της εντολής είναι μηδέν (δηλαδή η σημαία του μηδενός Z γίνει 1):

incfsz Reg ;αύξησε το περιεχόμενο του καταχωρητή Reg και παρέκαμψε
;την επόμενη εντολή αν το αποτέλεσμα είναι μηδέν

decfsz Reg ;μείωσε το περιεχόμενο του καταχωρητή Reg και παρέκαμψε
;την επόμενη εντολή αν το αποτέλεσμα είναι μηδέν

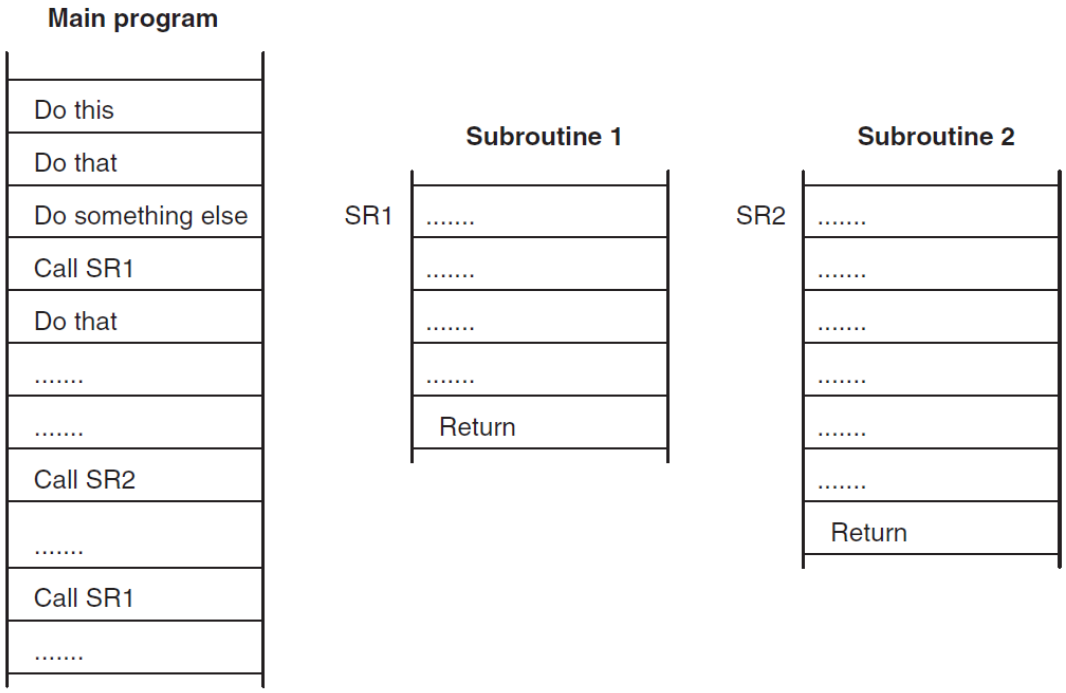
π.χ.

incfsz 20h

3.3 Υπορουτίνες

Οι υπορουτίνες επιτρέπουν να γράφουμε δομημένα προγράμματα και να επαναχρησιμοποιούμε χρήσιμο κώδικα. Μια υπορουτίνα ονομάζεται με μια ετικέτα που προηγείται της πρώτης εντολής της υπορουτίνας και τελειώναι με μια εντολή **return**. Η κλήση της γίνεται με μια εντολή **call**. Όταν ολοκληρωθεί η εκτέλεση της υπορουτίνας η εκτέλεση του προγράμματος επιστρέφει στην εντολή που ακολουθεί την **call**. Σε ένα πρόγραμμα μπορούμε να έχουμε πολλές υπορουτίνες, όπως φαίνεται στο παρακάτω σχήμα 3.1. Επίσης, είναι δυνατό να καλούμε μια υπορουτίνα μέσα από μια άλλη υπορουτίνα.

Κάθε φορά που καλείται μια υπορουτίνα, το περιεχόμενο του απαριθμητή εντολών (Program Counter) αποθηκεύεται στη στοίβα (stack). Όταν εκτελείται η εντολή **return**, τότε ανακτάται η τελευταία διεύθυνση που αποθηκεύτηκε στο σωρό και η εκτέλεση του προγράμματος μεταβαίνει στο σημείο αυτό.



Σχήμα 3.1 Οργάνωση προγράμματος με υπορουτίνες

3.4 Δομή ενσωματωμένων εφαρμογών

Ο κώδικας ενός μικροελεγκτή περιλαμβάνει τις απαραίτητες ενέργειες για την εκτέλεση του βρόχου ελέγχου που γνωρίσαμε σε προηγούμενες παραγράφους (παράγραφοι 1.2.1 και 2.1). Ο βρόχος ελέγχου είναι μια περιοδική διεργασία, και πρέπει να επαναλαμβάνεται κυκλικά. Άρα, το κυρίως πρόγραμμα (main) αντιπροσωπεύει έναν ατέρμονα βρόχο της μορφής

```
        initialize peripherals
main    do this
        do that
        read current sensor value
        compare current sensor value with reference value
        if current value < reference value
                do this
        else
                do something else
        goto main
```

Ο παραπάνω κώδικας είναι ψευδοκώδικας. Γενικά, ένας απλός κώδικας σε assembly έχει την εξής γενική δομή:

- Σύντομη τεκμηρίωση του κώδικα με τη μορφή σχολίων
- Συμπερίληψη (include) αρχείων δηλώσεων ανάλογα με τον μικροελεγκτή
- Οδηγία (macro-directive) ορισμού των ψηφίων διαμόρφωσης (configuration bits)
- Δηλώσεις ψευδωνύμων θέσεων μνήμης για ευκολία στη χρήση μεταβλητών
- Δηλώσεις μετάβασης στην αρχική θέση μνήμης προγράμματος (reset vector)
- Αρχικοποίηση περιφερειακών, όπως θύρες, χρονιστές κλπ.
- Κυρίως πρόγραμμα, όπου επιτελείται ο κυρίως βρόχος του προγράμματος

Ας σημειωθεί ότι τα bits διαμόρφωσης μπορούν να οριστούν και εκτός του κώδικα της εφαρμογής, μέσω του μενού εντολών του περιβάλλοντος ανάπτυξης. Τα bits αυτά διαμορφώνουν συγκεκριμένες γενικές ρυθμίσεις για τον τρόπο λειτουργίας της συσκευής και δεν μπορούν να αλλάξουν μέσω εντολών του προγράμματος.
Παρακάτω δίνεται ένα απλός κώδικας, που περιλαμβάνει τα παραπάνω βασικά στοιχεία για μια εφαρμογή, που απλά εισάγει τα δεδομένα που υπάρχουν στους ακροδέκτες της θύρας D και τα εξάγει αυτούσια στους ακροδέκτες της θύρας B. Επίσης, αποθηκεύει τα δεδομένα σε μια θέση μνήμης (20h), για ενδεχόμενη περαιτέρω χρήση. Στη θέση μνήμης έχουμε δώσει το μνημονικό ψευδώνυμο TEMP. Ότι ακολουθεί το ελληνικό ερωτηματικό σε μια γραμμή είναι σχόλιο και δεν εκτελείται.

```
;-------------------------------------------------------------------------------
; project_name: DIR \ test1
;TITLE: this program reads PORTD and transfers data to PORTB
;6.10.2012 by John

#include "p16f877.inc"

; Here we optionally set Configuration Word bits:
__CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF & _CPD_OFF &
_WRT_ENABLE_ON & _BODEN_ON & _LVP_OFF
;WDT OFF, Power-up timer ON, code protect OFF, XT oscillator,
;Low-Voltage Programming OFF
TEMP equ 20h                    ;declare an alias for memory position 20h
        Org 00                  ;define the Reset vector
;Initialization of peripherals
start   bsf STATUS, RP0         ;select memory bank 1
        movlw b'00000111'
        movwf TRISD             ;the three first bits of portA are input
        movlw 00
        movwf TRISB             ;all pins of portB are output
        bcf STATUS, RP0         ;return to bank0
        clrf PORTB
```

```
;main code starts here
main        clrf PORTB
            movf PORTD, w
            movwf TEMP
            movwf PORTB
            goto main
end
```

Κώδικας 3.1 Δομή ενσωματωμένης εφαρμογής

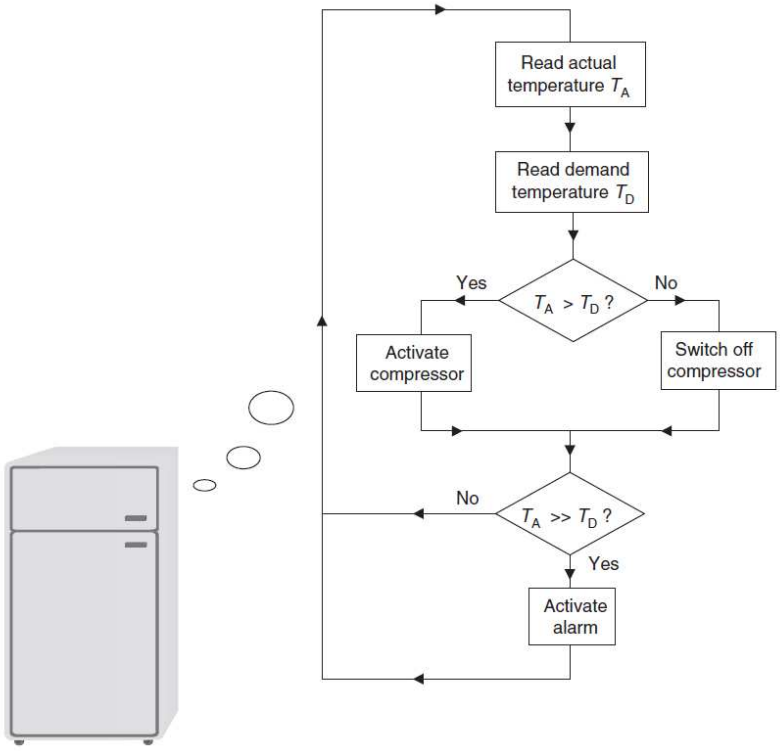
Ένας αποτελεσματικός τρόπος οργάνωσης μιας ενσωματωμένης εφαρμογής σε γλώσσα assembly είναι να οργανώνουμε τον κώδικα με βάση υπορουτίνες. Έτσι, το παραπάνω πρόγραμμα θα μπορούσε να γραφεί ως εξής

```
; project_name: DIR \ test1
;TITLE: this program reads PORTD and transfers data to PORTB
;6.10.2012 by John

#include "p16f877.inc"

__CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF & _CPD_OFF &
_WRT_ENABLE_ON & _BODEN_ON & _LVP_OFF
TEMP equ 20h                    ;declare an alias for memory position 20h
            Org 00              ;define the Reset vector
;main code starts here
main        call init
loop        call read_display
            goto loop
;subroutines
init        bsf STATUS, RP0     ;select memory bank 1
            movlw b'00000111'
            movwf TRISD         ;the three first bits of portA are input
            movlw 00
            movwf TRISB         ;all pins of portB are output
            bcf STATUS, RP0     ;return to bank0
            clrf PORTB
            return
read_display movf PORTD, w
            movwf TEMP
            movwf PORTB
            return
end
```

Κώδικας 3.2 Δόμηση της εφαρμογής με χρήση υπορουτινών

3.5 Υλοποίηση διακλάδωσης υπό συνθήκη

Στο παρακάτω σχήμα 3.2 φαίνεται ένα τυπικό σύστημα που ενσωματώνει ένα μικροελεγκτή προκειμένου να υλοποιήσει το βρόχο ελέγχου. Το ψυγείο ενεργοποιεί το συμπιεστή, που η λειτουργία του στηρίζεται σε έναν ηλεκτρικό κινητήρα, προκειμένου να μειώσει τη θερμοκρασία του χώρου της ψύξης. Αντίστοιχα, σταματά τη λειτουργία του συμπιεστή, όταν η θερμοκρασία μειωθεί αρκετά. Αν, πάλι, η πόρτα του ψυγείου μείνει ανοιχτή για αρκετή ώρα, ώστε η θερμοκρασία ανεβεί πολύ, τότε το σύστημα παράγει μια ηχητική ειδοποίηση (alarm). Η θερμοκρασία κάθε στιγμή συγκρίνεται με τη θερμοκρασία αναφοράς, προκειμένου να ληφθούν οι αποφάσεις. Το διάγραμμα ροής της εφαρμογής αποτυπώνει τις παραπάνω ανάγκες επεξεργασίας.



Σχήμα 3.2 Διάγραμμα ροής του ελεγκτή του ψυγείου

Ένα κρίσιμο ζητούμενο είναι η υλοποίηση της διακλάδωσης υπό συνθήκη, που απορρέει από τη σύγκριση της τρέχουσας θερμοκρασίας με τη θερμοκρασία αναφοράς. Παρακάτω παρουσιάζεται ένα τμήμα κώδικα που υλοποιεί τη δομή IF…THEN…ELSE σε γλώσσα assembly.

```
movf  TA, w          ;transfer TA into w
subwf TD, w          ;compare TD with TA (TD-TA)
 btfsc STATUS,C      ;if Carry=1 then result is negative (TD<TA)
do this              ;in which case do this
```

```
        btfss STATUS,C      ;if Carry=0 then subwf result is positive (TD>TA)
        do that             ;in which case do something else
```

Κώδικας 3.3 Υλοποίηση διακλάδωσης IF…THEN…ELSE

Τα βασικά χαρακτηριστικά είναι τα εξής: α. Η σύγκριση των δύο θερμοκρασιών γίνεται με αφαίρεση. β. Αν το αποτέλεσμα της αφαίρεσης είναι αρνητικό, τότε το bit **C** του καταχωρητή **STATUS** γίνεται μονάδα. Σε αντίθετη περίπτωση το bit είναι μηδέν γ. Η διπλή χρήση της εντολής διακλάδωσης υπό συνθήκη (btfsc και btfss) εξασφαλίζει ότι θα εκτελεστεί η εντολή *do this* ή η εντολή *do that* και ποτέ και οι δύο. Φυσικά, οι *do this* και *do that* αντιστοιχούν σε ψευδοκώδικα και παριστάνουν ό,τι πρέπει να εκτελεστεί στη μία ή στην άλλη περίπτωση.
Αν αυτό που εκετελείται με τις ψευδοεντολές *do this* και *do that* περιλαμβάνει παραπάνω από μία εντολές, τότε μπορεί να γραφεί με τη μορφή υπορουτίνας και να κληθεί με εντολή call:

```
        call do_this
```

3.5 Έμμεση προσπέλαση με τους καταχωρητές FSR και INDF

Ο ρόλος της έμμεσης διευθυνσιοδότησης είναι πολύ σημαντικός στην προσπέλαση πινάκων, δηλαδή συνόλου δεδομένων που έχουν διαταχθεί στη σειρά μέσα στη μνήμη, καταλαμβάνοντας πολλές θέσεις. Για το σκοπό αυτό οι μικροελεγκτές PIC διαθέτουν έναν καταχωρητή «δείκτη», τον **FSR**. Οι τιμές που καταχωρούνται στον **FSR** είναι θέσεις μνήμης δεδομένων. Έτσι, οι εντολές

```
        movlw 20h

        movwf FSR
```

καταχωρούν στον **FSR** τη διεύθυνση 20h. Αν στη συνέχεια, γράψουμε την εντολή

```
        movf INDF, w
```

τότε προσπελάζεται η διεύθυνση που έχει καταχωρηθεί στον **FSR** και το περιεχόμενό της εγγράφεται στον καταχωρητή εργασίας **w**. Ο INDF στην παραπάνω εντολή δεν είναι πραγματικός καταχωρητής, αλλά η χρήση του όπως παραπάνω είναι μια συμβολική έκφραση, που όταν εκτελείται προσπελάζει τη διεύθυνση που περιέχεται στον **FSR** και μεταφέρει το περιεχόμενό της στον **w**.

Παρακάτω θα δώσουμε ένα παράδειγμα προσπέλασης πίνακα, με τη βοήθεια του καταχωρητή «δείκτη» **FSR**.

Έστω ο παρακάτω πίνακας τιμών στη μνήμη RAM

Mnemonic name	Address in RAM	Example value
startf	0x20	0x09
	0x21	0x07
	0x22	0x05
	0x23	0x1A
endf	0x24	0x2B

Τότε οι παρακάτω εντολές προσπελάζουν τον πίνακα από την αρχική διεύθυνση μέχρι την τελική και προβάλλουν το περιεχόμενο στη θύρα B:

```
        movlw startf
        movwf FSR          ;load start memory location in FSR
loop2   movf INDF,w        ;access memory location in FSR
        movwf PORTB        ;display memory content on PORTB
        movf FSR,w
        sublw endf         ;check if current memory is last
        btfsc STATUS,Z
        goto label
        incf FSR,1         ;access next memory location
        goto loop2
label   goto label         ;endless loop
```

Κώδικας 3.4 Προσπέλαση θέσεων μνήμης με τον καταχωρητή **FSR**.

Προφανώς, ο παραπάνω κώδικας δεν είναι ολοκληρωμένος, καθώς δεν περιέχει παρά μόνον το τμήμα του κυρίως προγράμματος που προσπελάζει τις θέσεις μνήμης. Η αφαίρεση που επιτελείται ανάμεσα στην τρέχουσα θέση μνήμης και στην τελευταία (endf) είναι ένας μηχανισμός με τον οποίο ελέγχεται αν ολοκληρώθηκε η προσπέλαση του πίνακα ή πρέπει να επαναληφθεί ο βρόχος loop2.

3.4 Εργαλεία Λογισμικού για τον Προγραμματισμό του Μικροελεγκτή PIC16F877

Έχοντας αποκτήσει κανείς μια εικόνα για την αρχιτεκτονική του μικροελεγκτή PIC16F84 και για τις βασικές εντολές της μνημονικής γλώσσας (*assembly*), επόμενο είναι να αναρωτιέται με ποια εργαλεία θα μπορέσει να αναπτύξει πρακτικές εφαρμογές, προγραμματίζοντας κατάλληλα τον PIC. Η παράγραφος αυτή περιέχει ορισμένες απαντήσεις, που θα καθοδηγήσουν τον αρχάριο χρήστη στην αντιμετώπιση πρακτικών προβλημάτων.

- **Ο κειμενογράφος (text editor):** Με τον κειμενογράφο συντάσσουμε το πηγαίο αρχείο σε μνημονική γλώσσα, το οποίο κατόπιν θα μεταφράσουμε και θα αποθηκεύσουμε στη μνήμη ROM του μικροελεγκτή. Τέτοιο πρόγραμμα μπορεί να είναι ένας οποιοσδήποτε κειμενογράφος χαρακτήρων ASCII, για παράδειγμα ο

κειμενογράφος EDIT του DOS ή το Notepad των Windows. Το τελικό αρχείο κειμένου πρέπει να έχει επέκταση .ASM γι' αυτό πρέπει να αποθηκεύσει κανείς το αρχείο που πληκτρολόγησε δίνοντας ως επέκταση τα αρχικά ".ASM".

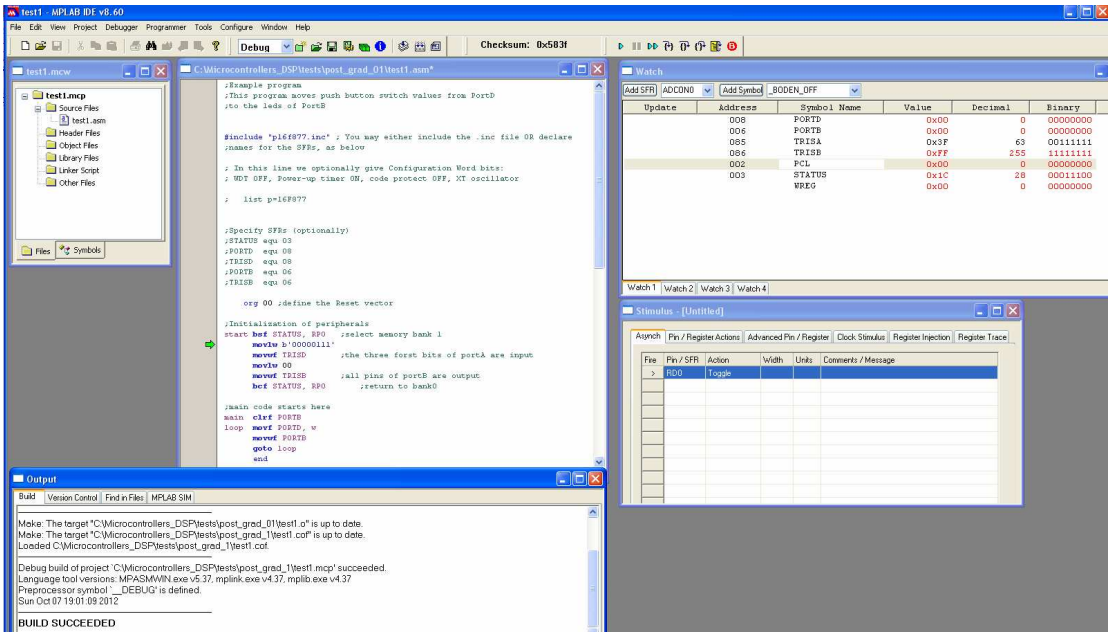
- **Ο συμβολομεταφραστής (assembler):** Πρόκειται για το πρόγραμμα που μεταφράζει το πηγαίο αρχείο .ASM σε δεκαεξαδική μορφή, κατάλληλη να τη διαχειριστεί ο προγραμματιστής της μνήμης EEPROM του μικροελεγκτή. Αν και κυκλοφορούν πολλοί συμβολομεταφραστές, μία εύκολη και καλή λύση είναι να χρησιμοποιήσει κανείς το πρόγραμμα MPASM της εταιρείας Microchip, το οποίο αποτελεί μέρος του ολοκληρωμένου περιβάλλοντος MPLAB και διατίθεται χωρίς χρέωση στην ιστοσελίδα της εταιρείας. Κατά τη μετάφραση παράγονται τα μηνύματα σφαλμάτων, μετά τη διόρθωση των οποίων ο συμβολομεταφραστής παράγει ένα αρχείο με επέκταση .HEX, το οποίο μπορεί να χρησιμοποιηθεί για τα επόμενα βήματα. Επίσης, παράγονται αρχεία που περιέχουν αναφορές σφαλμάτων καθώς και αρχεία εισόδου για τον προσομοιωτή (simulator).

- **Ο προσομοιωτής (simulator):** Αυτός είναι ένα βοηθητικό και όχι υποχρεωτικό πρόγραμμα. Τέτοιο πρόγραμμα είναι το MSIM της εταιρείας Microchip. Το πρόγραμμα αυτό χρησιμοποιεί σαν αρχείο εισόδου ένα αρχείο με επέκταση .INI που παράγεται από το πρόγραμμα MPASM κατά τη διαδικασία της συμβολομετάφρασης. Το πρόγραμμα MSIM παράγει μία βήμα προς βήμα προσομοίωση της εκτέλεσης του προγράμματος και επιδεικνύει τις τιμές που λαμβάνουν οι διάφοροι καταχωρητές ειδικού σκοπού και οι θέσεις της μνήμης RAM. Με τον τρόπο αυτόν ο χρήστης μπορεί να βεβαιωθεί ότι το πρόγραμμα επιτελεί ακριβώς αυτό για το οποίο προορίζεται και να ανιχνεύσει τα τυχόν σφάλματα που υπάρχουν.

Τα παραπάνω εργαλεία μπορεί να τα βρει κανείς ενσωματωμένα σε ολοκληρωμένα αναπτυξιακά εργαλεία λογισμικού, όπως είναι το πρόγραμμα MPLAB της εταιρείας Microchip. Το πρόγραμμα αυτό αναφέρεται ως *Ολοκληρωμένο Περιβάλλον Ανάπτυξης Εφαρμογών* (*IDE – Integrated Development Environment*). Είναι ένα σπονδυλωτό παραθυρικό πρόγραμμα που περιέχει σε ενιαίο γραφικό περιβάλλον τον κειμενογράφο, τον συμβολομεταφραστή MPASM και τον προσομοιωτή MSIM της εταιρείας Microchip. Το πρόγραμμα MPLAB δημιουργεί *σχέδια εργασίας* (*projects*) που περιέχουν όλα τα απαραίτητα αρχεία κάθε εφαρμογής. Για τον προγραμματισμό της συσκευής μπορεί να συνδεθεί με κυκλώματα προγραμματισμού, όπως ο PICSTART Plus ή ο PICkit3. Το πρόγραμμα διατίθεται δωρεάν από την εταιρεία Microchip, στην ιστοσελίδα www.microchip.com (βλέπε και Παράρτημα Α).

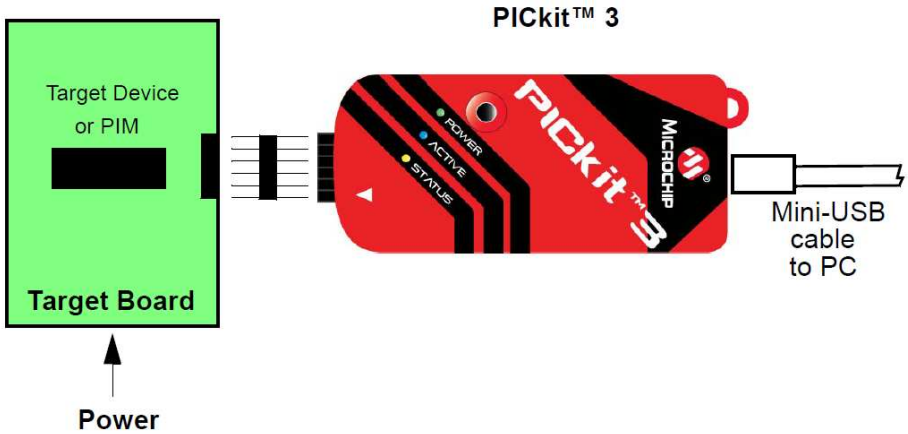
Τα παραπάνω λογισμικό μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών με κάθε μικροελεγκτή της εταιρείας Microchip και όχι μόνον με τον PIC16F877.

Στην παρακάτω εικόνα φαίνεται το γραφικό περιβάλλον του προγράμματος MPLAB, όπου τα διάφορα παράθυρα είναι ο κειμενογράφος, η κονσόλα όπου παράγονται τα μηνύματα εξόδου (Output) και παράθυρα προσομοίωσης του κώδικα.



Εικόνα 3.1: Το περιβάλλον MPLAB

Ο προγραμματισμός του μικροελεγκτή γίνεται με ειδικά κυκλώματα, που συνδέονται στον υπολογιστή. Τα κυκλώματα αυτά καταλήγουν σε ακροδέκτες του μικροελεγκτή, που διασυνδέονται με τη μνήμη προγράμματος. Η συσκευή της εικόνας 3.2 είναι ο προγραμματιστής PICkit3, που προγραμματίζει τον μικροελεγκτή πάνω στο κύκλωμα (in-circuit programmer)



Εικόνα 3.2: In-circuit programmer-debugger PICkit3

4. Είσοδος/έξοδος και χρονισμός του μικροελεγκτή PIC

4.1 Είσοδος/έξοδος και καταχωρητές θυρών

Οι καταχωρητές **PORTB** και **TRISB**, **PORTD** και **TRISD**, καθώς και οι αντίστοιχοι καταχωρητές των υπόλοιπων θυρών, χρησιμεύουν στη λειτουργία εισόδου/εξόδου, δηλαδή στην ανάγνωση και εγγραφή δεδομένων από και προς τις θύρες. Η εγγραφή σε έναν καταχωρητή **TRIS** ορίζει ποιοι ακροδέκτες της αντίστοιχης θύρας λειτουργούν σαν είσοδοι και ποιοι σαν έξοδοι.

Εγγράφοντας στον καταχωρητή **TRISB** τον δυαδικό αριθμό 11110000, ορίζουμε τα τέσσερα λιγότερο σημαντικά bits της θύρας B ως εξόδους και τα τέσσερα περισσότερο σημαντικά bits ως εισόδους (0 = έξοδος, 1 = είσοδος):

movlw b'11110000'
movwf TRISB

Ας θυμηθούμε ότι η θύρες **B, C, D** έχουν οκτώ ακροδέκτες, η θύρα **A** έχει έξι και η θύρα **E** έχει τρεις (στον μικροελεγκτή PIC16F877). Όλοι αυτοί μπορούν να οριστούν σε κάθε στιγμή ως είσοδοι ή έξοδοι. Επίσης, ας θυμηθούμε ότι η πρόσβαση στον καταχωρητή **TRISB** για τον ορισμό της διεύθυνσης λειτουργίας της θύρας **B**, απαιτεί μετάβαση στη σελίδα 1 της μνήμης (bit RP0 του καταχωρητή **STATUS** σε λογικό ένα). Το ίδιο, βέβαια, συμβαίνει και με τους υπόλοιπους καταχωρητές **TRIS**. Όταν τελειώσει η διαδικασία αρχικοποίησης των θυρών, πρέπει να επιστρέψουμε στη βασική σελίδα μνήμης. Ο ολοκληρωμένος κώδικας για την αρχικοποίηση της θύρας B έχει ως εξής:

bsf STATUS, RP0
movlw b'11110000'
movwf TRISB
bcf STATUS, RP0

Η εντολή ανάγνωσης ενός καταχωρητή θύρας, π.χ. του **PORTB**, διαβάζει την κατάσταση των ακροδεκτών της θύρας και τη μεταφέρει στον καταχωρητή εργασίας **w**:

movf PORTB, w

Η εντολή εγγραφής του **w** στον καταχωρητή **PORTB** εμφανίζει το περιεχόμενο του **w** στους αντίστοιχους ακροδέκτες:

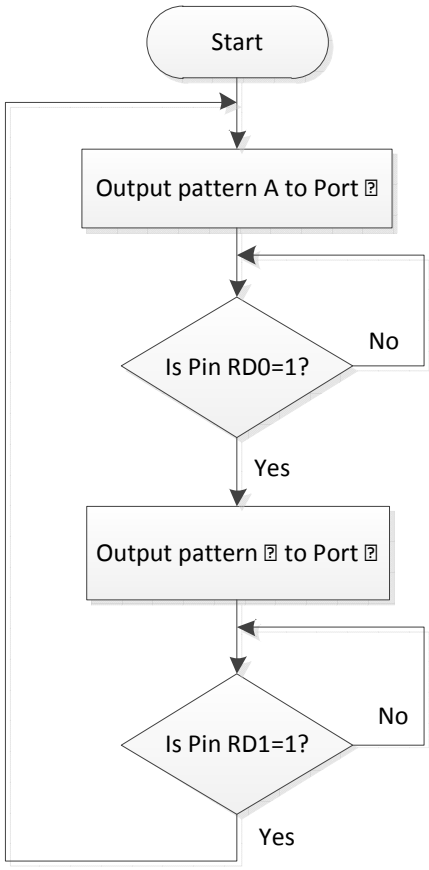
movwf PORTB

Οι μικροελεγκτές PIC έχουν τη δυνατότητα μέσω ειδικών εντολών να εγγράφουν ένα μόνο bit κάποιου καταχωρητή, αγνοώντας το υπόλοιπο μέρος του συγκεκριμένου byte:

bsf PORTB, 2 ;θέσε σε λογικό 1 το bit RB2

4.1.2 Προγραμματισμός των θυρών PIO του PIC16F877.

Το παρακάτω πρόγραμμα επιδεικνύει τον τρόπο προγραμματισμού των θυρών Parallel I/O (PIO) του PIC για είσοδο και για έξοδο. Δύο διακόπτες πιέσεως (push buttons) συνδέονται με τους ακροδέκτες RD0 και RD1 και όταν πιέζονται μεταφέρουν στους ακροδέκτες τάση 5V, που αντιστοιχεί σε λογικό 1. Στην αρχή, ο κώδικας εξάγει ένα φωτεινό μοτίβο στη θύρα B και αναμένει την πίεση του πρώτου διακόπτη. Στη συνέχεια, μόλις ο διακόπτης πιεστεί, αλλάζει το φωτεινό μοτίβο και αναμένει την πίεση του δεύτερου διακόπτη. Όταν πιεστεί κι αυτός, η διαδικασία επαναλαμβάνεται, με την εμφάνιση του πρώτου μοτίβου. Το διάγραμμα ροής του κώδικα παρουσιάζεται στο σχήμα 4.1.



Σχήμα 4.1 Το διάγραμμα ροής της εφαρμογής εισόδου/εξόδου

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες παρατίθεται παρακάτω:

```
#include "p16F877.inc"
        ;Initialization
        Org 0                   ;Start from program memory address 0
        bsf STATUS, RP0         ;Change to bank1
```

```
        movlw b'00000000'    ;Make all pins of portB Outputs
        movwf TRISB
        movlw b'00011111'    ;Make 5 pins of portD inputs
        movwf TRISD
        bcf STATUS, RP0      ;Return to bank0

main    movlw b'01010101'    ;Output to portB
        movwf PORTB
        btfss PORTD, 0       ;Wait until pin0 of portD receives 1
        goto $-1
        movlw b'10101010'    ;If pin0 of portD is zero, change output combination to
                             ;portB
        movwf PORTB
        btfss PORTD, 1       ;Wait until pin1 of portD receives 1
        goto $-1
        goto main
        end
```

Κώδικας 4.1 Παράδειγμα προγραμματισμού θυρών εισόδου/εξόδου

4.2 Χρονισμός

4.2.1 Κύκλος εκτέλεσης εντολής

Οι περισσότερες εντολές των μικροελεγκτών PIC εκτελούνται σε έναν κύκλο εκτέλεσης (instruction cycle). Αυτό είναι αποτέλεσμα της αρχιτεκτονικής RISC που υποστηρίζουν αυτές οι μονάδες μικροελεγκτών. Κάθε κύκλος εκτέλεσης περιλαμβάνει τα βήματα fetch, decode, execute και store, οπότε κάθε εντολή ολοκληρώνεται σε τέσσερις περιόδους του εξωτερικού ρολογιού. Γενικά αναφερόμαστε στη συχνότητα του εξωτερικού ρολογιού ως f_{osc} και στην περίοδο ως T_{osc}. Άρα

fetch, decode, execute and store = 1 instruction cycle

Διάρκεια ενός κύκλου εντολής = $4*T_{osc}$ (4.1)

Ο κύκλος εκτέλεσης εντολής ορίζει την περίοδο μιας εσωτερικής γεννήτριας παλμών, που ονομάζεται ρολόϊ εντολών του PIC.

Γνωρίζοντας πόσους εξωτερικούς παλμούς χρειάζεται κάθε εντολή για να ανακτηθεί από την μνήμη και να εκτελεστεί μπορούμε να υπολογίσουμε το συνολικό χρόνο που χρειάζεται ένα πρόγραμμα ή μια ρουτίνα για να εκτελεστεί.

Άρα, τη χρονική διάρκεια κάθε εντολής μπορούμε να την υπολογίσουμε αν γνωρίζουμε την συχνότητα του εξωτερικού ταλαντωτή, σύμφωνα με τη σχέση:

Διάρκεια κύκλου εντολής = 4 / (Συχνότητα εξωτερικού κρυστάλλου) (4.2)

Τυπική συχνότητα λειτουργίας των μικροελεγκτών PIC είναι 4MHz, οπότε η διάρκεια κάθε κύκλου εντολής είναι **1μsec.** Για κρύσταλλο 8MHz η διάρκεια του κύκλου εντολής είναι 0,5 μsec.

Όπως είναι εύκολα κατανοητό, η διάρκεια εκτέλεσης ενός προγράμματος, ισούται με τον αριθμό τον εντολών που το αποτελούν επί τη διάρκεια του κύκλου εκτέλεσης της εντολής. Κάποιες εντολές χρειάζονται παραπάνω από έναν κύκλο εντολής για να εκτελεστούν. Τέτοιες είναι οι εντολές αλλαγής της ροής του προγράμματος (Π.χ. goto, call) που χρειάζονται δύο κύκλους εντολής για να εκτελεστούν.
Με βάση τα παραπάνω μπορούν να δημιουργηθούν κατάλληλες υπορουτίνες καθυστέρησης.

4.2.2 Θέματα χρονισμού και καθυστέρησης της εκτέλεσης (delay)

Όπως αναφέρθηκε στην προηγούμενη παράγραφο, αν η συχνότητα του εξωτερικού κρυστάλλου είναι f_{osc} = 4MHz, τότε η περίοδος είναι

$T_{osc} = 1/f_{osc} = 0{,}25$ μs.

Άρα, ένας πλήρης κύκλος εκτέλεσης εντολής διαρκεί 1μs.

Συνεπώς, σε ένα τέτοιο σύστημα η εκτέλεση των περισσότερων εντολών διαρκεί 1μs.

Εξαίρεση αποτελούν οι εντολές διακλάδωσης, που αναφέρθηκαν στις προηγούμενες παραγράφους. Έτσι, η εντολή goto, η εντολή call και η εντολή return διαρκούν δύο κύκλους εκτέλεσης εντολής, δηλαδή για εξωτερικό κρύσταλλο με συχνότητα 4MHz αυτές οι εντολές διαρκούν 2μs.

Οι εντολές ελέγχου bit διαρκούν 1 κύκλο εντολής αν δεν εκτελούν παράκαμψη της επόμενης εντολής (δηλαδή δεν συντρέχει η συνθήκη τους), αλλά διαρκούν 2 κύκλους, αν εκτελούν την παράκαμψη.

Έτσι, η παρακάτω υπορουτίνα delay παράγει μια συνολική καθυστέρηση 1ms.

```
;With crystal XT and fosc=4MHz, instruction cycle is 1 μs
;branch instructions (call, goto) take 2μs

delay   movlw 0F9       ;load decimal 249
        nop             ;first two instructions take 2μs      } 4μs
micro4  addlw 0FF       ;add -1 in 2's complement. 1μs        }
```

```
        btfss STATUS, Z     ;no skip: 1μs. With skip: 2μs
        goto micro4         ;2μs
        return              ;2μs
```

Κώδικας 4.2: Υπορουτίνα καθυστέρησης, για συνολική καθυστέρηση 1ms

Αρχικά φορτώνεται ο W με τον δεκαδικό 249 (διάρκεια 1μs) και ακολουθεί μια εντολή nop που απλώς διαρκεί 1μs χωρίς να επιτελεί τίποτε (no operation). Ο βρόχος micro4 - goto micro4 διαρκεί ακριβώς 4μs, διότι η πρώτη εντολή (addlw) διαρκεί 1μs, η εντολή btfss διαρκεί 1μs όταν δεν συντρέχει η συνθήκη και η goto διαρκεί 2 μs. Όταν συντρέχει η συνθήκη Z=1, τότε η btfss διαρκεί 2 μs. Άρα σε κάθε περίπτωση ο βρόχος διαρκεί 4 μs. Σε κάθε εκτέλεση του βρόχου, η εντολή addlw προσθέτει στο 249 το -1 (FF ως προσημασμένος, με το συμπλήρωμα ως προς 2), οπότε το αποτέλεσμα της άθροισης θα γίνει μηδέν σε 249 κύκλους εκτέλεσης του βρόχου, δηλαδή σε συνολική διάρκεια:

249 x 4 μs = 996 μs

Στη διάρκεια αυτή πρέπει να προστεθούν και τα 2 μs των δύο πρώτων εντολών, καθώς και τα 2μs της εντολής call delay. Σύνολο, 1000 μs, δηλαδή 1ms.

Η παρακάτω παραλλαγή είναι μια υπορουτίνα που ονομάζεται delay_ms και δημιουργεί καθυστέρηση όσα ms έχουν εγγραφεί στον καταχωρητή msec (από 1 έως 255).

```
delay_ms    movwf msec
loop        movlw 0F8       ;decimal 248
            call micro4     ;248x4+2=994μs
            nop
            nop
            decfsz msec, f
            goto loop
            return
```

Κώδικας 4.3 Υπορουτίνα καθυστέρησης έως 255 ms

Η υπορουτίνα delay_ms καλεί την υπορουτίνα micro4 που περιγράψαμε παραπάνω 248 φορές (συνολική διάρκεια 994 μs) και επαναλαμβάνει την κλήση αυτή μέσα σε ένα βρόχο που εκτελείται msec φορές. Ο βρόχος προσθέτει κάθε φορά τα μs που υπολείπονται, μέχρι τα 1000μs. Έτσι, ο κώδικας

```
        msec equ 30h
        movlw d'35'
        call delay_ms
```

παράγει εξαιτίας της delay_ms μια καθυστέρηση 35 ms.

4.2.3 Χρονιστές (timers) στους PIC

Ο PIC διαθέτει τρεις μετρητές: δύο οκτάμπιτους (8bits) χρονιστές/μετρητές (timer/counter), τους Timer0,Timer2 και έναν δεκαξάμπιτο (16bits), τον Timer1.

Ο Timer0, είναι ένας πλήρως προγραμματιζόμενος απαριθμητής, του οποίου το περιεχόμενο προσπελάζεται μέσω του ειδικού καταχωρητή TMR0. Σε κάθε παλμό ρολογιού, το περιεχόμενο του TMR0 αυξάνει κατά ένα. Το κύκλωμα μπορεί να λειτουργήσει με δύο τρόπους.

α) *Ως χρονιστής (timer)*: Στην περίπτωση αυτή η πηγή χρονισμού είναι ο εσωτερικός κύκλος των εντολών. Το περιεχόμενο του TMR0 σε λειτουργία χρονιστή αυξάνει κατά ένα σε κάθε κύκλο εντολής. Ο κύκλος της εντολής διαρκεί όσο η περίοδος του εξωτερικού κρυστάλλου επί τέσσερα ($4*T_{osc}$). Η αντίστοιχη συχνότητα είναι ίση με την συχνότητα f_{osc} του κρυστάλλου διά 4. Χρησιμοποιώντας τον Timer0 σε λειτουργία εσωτερικού χρονισμού, έχουμε στη διάθεσή μας ένα αρκετά αξιόπιστο σήμα χρονισμού που μπορεί να χρησιμοποιηθεί για τον ακριβή προσδιορισμό χρονικών διαστημάτων.

β) *Ως απαριθμητής εξωτερικών συμβάντων (counter)*: Στην περίπτωση αυτή χρησιμοποιείται ένας εξωτερικός παλμός χρονισμού. Το εξωτερικό σήμα χρονισμού συνδέεται με τον ακροδέκτη TOCKI (RA4). Η χρήση εξωτερικής πηγής χρονισμού επιτρέπει την απαρίθμηση εξωτερικών συμβάντων, με τη μορφή παλμών. Το περιεχόμενο του Timer0 αυξάνει κατά ένα σε κάθε παλμό της εξωτερικής πηγής χρονισμού.

Και στις δύο περιπτώσεις, ο **TMR0** μετρά από την αρχική τιμή που καταχωρήσαμε έως 0xFF (στο δεκαεξαδικό σύστημα).

Όπως θα δούμε, η επιλογή του είδους της πηγής χρονισμού του TMR0, επιτυγχάνεται με τη χρήση του bit T0CS του ειδικού καταχωρητή OPTION_REG.

Τα bits του καταχωρητή **OPTION_REG** καθορίζουν εάν θα μπει σε λειτουργία ο απαριθμητής και με ποιον από όλους τους δυνατούς τρόπους θα εργαστεί. Ο πίνακας 4-1 παρουσιάζει τα bits του καταχωρητή OPTION_REG και τη σημασία τους.

Πίνακας 4-1
Τα bits του καταχωρητή OPTION_REG, ο οποίος ρυθμίζει τη λειτουργία του χρονιστή TMR0

b7	b6	b5	b4	b3	b2	b1	b0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

- **Bits 7-6**: Δεν έχουν σημασία για την λειτουργία του χρονιστή (τα θέτουμε 11).
- **T0CS**: Επιλογή πηγής ρολογιού
  1 = Πηγή συνδεδεμένη στον ακροδέκτη T0CKI (Λειτουργία μετρητή παλμών)

  0 = Εσωτερική πηγή ρολογιού (Λειτουργία χρονιστή).
- **T0CE:** Επιλογή θετικού ή αρνητικού μετώπου παλμού (αναφέρεται σε εξωτερική πηγή χρονισμού)
  1 = Αύξηση με κατερχόμενο μέτωπο του παλμού στον ακροδέκτη T0CLK

  0 = Αύξηση με ανερχόμενο μέτωπο του παλμού στον ακροδέκτη T0CLK
- **PSA**: "0", όταν θέλουμε να συνδέσουμε τον διαιρέτη συχνότητας.
  "1", όταν θέλουμε να αποσυνδέσουμε τον διαιρέτη συχνότητας.
- **PS2-PS0***:* Επιλογή λόγου διαίρεσης

Εάν επιθυμούμε να αυξηθεί ο χρόνος τον οποίο μετρά ο απαριθμητής **TMR0**, μπορούμε να χρησιμοποιήσουμε έναν διαιρέτη συχνότητας (*prescaler*). Αυτός μπορεί να ρυθμιστεί κατάλληλα, ώστε να διαιρεθεί η συχνότητα αύξησης του μετρητή κατά μία επιθυμητή τιμή. (1:2, 1:4, 1:8, ... 1:256).

Πίνακας 4-2

Διαίρεση της συχνότητας με την οποία αυξάνεται ο TMR0 και ο WDT

Τιμή προμετρητή			Ρυθμός TMR0	Ρυθμός WDT
PS2	PS1	PS0		
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Η ρύθμιση του διαιρέτη συχνότητας γίνεται ορίζοντας κάποια bits του καταχωρητή **OPTION_REG** (Πίνακας 4-1). Συγκεκριμένα, ορίζουμε τα τρία πρώτα bits (PS0, PS1, και PS2) του καταχωρητή **OPTION_REG** από p=000 έως p=111 (δηλαδή στο δεκαδικό σύστημα από 0 έως 7), και έτσι μεταβάλλουμε τον ρυθμό του απαριθμητή **TMR0** κατά τον παράγοντα $1:2^{p+1}$

Εάν το bit PSA του καταχωρητή **OPTION_REG** τεθεί σε λογικό ένα, τότε ο διαιρέτης αποσυνδέεται από τον απαριθμητή **TMR0** και συνδέεται με τον χρονιστή επιτήρησης (**WDT**, βλέπε παρακάτω). Στην περίπτωση αυτή δεν συμβαίνει καμία καθυστέρηση στο χρόνο αύξησης του περιεχομένου του απαριθμητή **TMR0** και αυτός αυξάνει κατά ένα σε κάθε κύκλο εντολής.

Εάν το bit PSA του καταχωρητή **OPTION_REG** τεθεί σε λογικό μηδέν, τότε ο διαιρέτης συχνότητας συνδέεται με τον απαριθμητή **TMR0** και διαιρεί τη συχνότητα αύξησης κατ' ελάχιστο διά δύο.

Όταν ο Timer0 λειτουργεί ως χρονιστής, τότε ο χρόνος που χρειάζεται για να υπερχειλίσει ο καταχωρητής TMR0 δίνεται από τη σχέση:

$$\text{Delay} = \frac{(256 - \text{αρχικό περιεχόμενο του TMR0}) * \text{λόγος διαίρεσης συχνότητας} * 4}{\text{Συχνότητα κρυστάλλου}} \quad (4.3)$$

Όταν ο απαριθμητής **Timer0** υπερχειλίσει, δηλαδή όταν ο καταχωρητής του **TMR0** μεταβεί από την ανώτερη τιμή 0xFF στην αρχική τιμή 0x00, τότε μια σημαία σηματοδοτεί την υπερχείλιση μεταβαίνοντας σε λογικό 1. Η σημαία αυτή ονομάζεται T0IF και βρίσκεται στον καταχωρητή INTCON (b2). Στο σημείο αυτό, και εφόσον έχει ενεργοποιηθεί η αντίστοιχη διακοπή (*interrupt*), δημιουργείται ένα *σήμα διακοπής*. Η εκτέλεση του προγράμματος διακόπτεται και καλείται η ρουτίνα εξυπηρέτησης της διακοπής. Πιο λεπτομερής αναφορά στα σήματα διακοπών στους PIC γίνεται σε επόμενη παράγραφο. Ας σημειωθεί ότι η συγκεκριμένη χρήση του απαριθμητή/χρονιστή **Timer0** για τη δημιουργία διακοπών χρησιμοποιείται σε εφαρμογές στις οποίες η καταμέτρηση του χρόνου έχει κρίσιμη σημασία. Παράδειγμα τέτοιας εφαρμογής είναι η χρήση του μικροελεγκτή για λήψη και εκπομπή σημάτων, με τη μέθοδο της ασύγχρονης σειριακής επικοινωνίας.

Στο σημείο αυτό θα γίνει μια σύντομη αναφορά σε έναν ακόμη χρονιστή που διαθέτει ο PIC και ονομάζεται χρονιστής επιτήρησης (Watch-Dog Timer ή WDT). Ο χρονιστής επιτήρησης, όταν ενεργοποιείται, επανεκκινεί (reset) τη CPU σε τακτά χρονικά διαστήματα, ώστε να αποκλείεται η παγίδευση της εκτέλεσης του κώδικα σε κάποιο αδιέξοδο. Η ενεργοποίηση και η απενεργοποίηση του χρονιστή επιτήρησης γίνεται με τη βοήθεια των bits διαμόρφωσης, που εγγράφονται στη λέξη διαμόρφωσης όταν η συσκευή προγραμματίζεται. Ανάμεσα στα bits διαμόρφωσης (configuration bits) υπάρχει ένα που ενεργοποιεί ή απενεργοποιεί τον WDT (WDT_ON ή WDT_OFF). Περισσότερα για τα bits διαμόρφωσης αναφέρονται στην παράγραφο 3.4 και στα εργαστηριακά φύλλα έργου.

Όταν ο διαιρέτης συχνότητας αποσυνδέεται από το κύκλωμα του χρονιστή, τότε συνδέεται με τον WDT (PSA=1). Στην περίπτωση αυτή, εφόσον έχει ενεργοποιηθεί ο

WDT η συχνότητα επανεκκίνησης υποδιαιρείται με τον τρόπο που εμφανίζεται στον πίνακα 4-2.

4.3 Προσομοίωση της λειτουργίας του χρονιστή

Ο παρακάτω κώδικας προγραμματίζει τον χρονιστή Timer0 και επιτρέπει να παρατηρήσουμε την υπερχείλιση του καταχωρητή **TMR0** σε περιβάλλον προσομοίωσης (βλέπε και εργαστηριακά φύλλα έργου).

```
#include "P16F877.inc"
        Org 0                   ;Το πρόγραμμα ξεκινά από τη θέση μνήμης προγρ. 0
        bsf STATUS, RP0         ;Μεταβαίνουμε στην σελίδα 1 της μνήμης
        movlw b'11010001'       ;Διαίρεση συχνότητας διά 4
        movwf OPTION_REG        ;Εγγράφουμε τον OPTION REG
        bcf STATUS, RP0         ;Μεταβαίνουμε στην σελίδα 0 της μνήμης
        movlw 0F0h              ;Θέτουμε στον TMR0 αρχική τιμή 240
        movwf TMR0
        bcf INTCON, T0IF        ;Μηδενίζουμε την σημαία T0IF
loop    goto loop               ;Βρόχος αναμονής
        END
```

Κώδικας 4.4 Ρύθμιση και φόρτωση του χρονιστή, με σκοπό την προσομοίωση της διαδικασίας

Στον παραπάνω κώδικα, θέτοντας το PSA bit του OPTION_REG μηδέν, συνδέουμε τον χρονιστή TIMER0 με τον διαιρέτη συχνότητας. Θέτοντας τα bits PS2:PS0=001 ορίζουμε τη συχνότητα αύξησης του TIMER0 ίση με τη συχνότητα ρολογιού εντολών διά 4. Όταν η εκτέλεση του προγράμματος φτάσει στον βρόχο αναμονής «loop GOTO loop», τότε ο TIMER0 αυξάνεται κατά 1 για κάθε δύο εκτελέσεις της εντολής βρόχου. Ο λόγος είναι ότι η εντολή GOTO διαρκεί δύο κύκλους εντολής (2x2=4 ο λόγος διαίρεσης).

4.4 Ρολόϊ πραγματικού χρόνου

Το παρακάτω πρόγραμμα μετρά με ακρίβεια το χρονικό διάστημα που ορίζουμε στον καταχωρητή SEC. Ο εξωτερικός ταλαντωτής έχει συχνότητα 8MHz. Ο προγραμματισμός του χρονιστή Timer0 γίνεται με βάση τη σχέση (4.3).

```
#include "P16F877.INC"
        Org 0
CENT equ 20h                    ; Δίνουμε στη θέση μνήμης 20h το όνομα CENT
SEC equ 21h                     ; Δίνουμε στη θέση μνήμης 21h το όνομα SEC
```

```
        movlw d'5'
        movwf SEC           ;Ορίζουμε χρόνο 5 sec και το αποθηκεύουμε στην θέση SEC
        clrf CENT           ;Μηδενίζουμε την θέση CENT (εκατοστά του δευτερολέπτου)
        bsf STATUS, RP0     ;Μεταβαίνουμε στην σελίδα μνήμης 1
        movlw b'11111100'
        movwf TRISB          ; Κάνουμε τα δύο χαμηλότερα bits της θύρας B έξοδο
        movlw b'11010111'
        movwf OPTION_REG    ;Ορίζουμε διαίρεση συχνότητας 1/256 PS2:PS0=111
        bcf STATUS, RP0      ; Επιστρέφουμε στην σελίδα μνήμης 0
        movlw b'00000010'    ; Ανάβουμε το δεύτερο LED της PORTB
        movwf PORTB
loop1   movlw d'178'
        movwf TMR0           ; Χρόνος=(256-178)*256*4/8 σε μsec (=0,01 sec). Σχέση (4.3).
        bcf INTCON, T0IF     ; Μηδενίζουμε τη σημαία T0IF
loop2   btfss INTCON, T0IF   ; Αναμονή 0,01 sec
        goto loop2
        incf CENT,1
        movlw d'100'
        subwf CENT,w
        btfss STATUS,Z
        goto loop1           ; Επαναλαμβάνει την αναμονή 100 φορές
        clrf CENT
        decfsz SEC,f
        goto loop1           ; Επαναλαμβάνει τη συνολική αναμονή SEC φορές
        movlw b'00000001'
        movwf PORTB          ; Ανάβει το πρώτο LED
loop3   goto loop3
        END
```

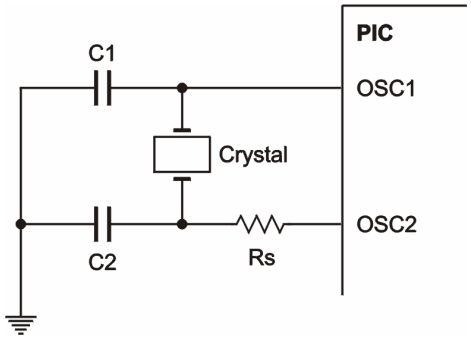
Κώδικας 4.5 Μέτρηση δευτερολέπτων με χρήση του Timer0

4.4 Εξωτερικός ταλαντωτής

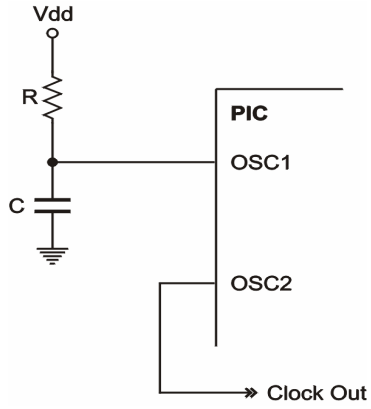
Θα κλείσουμε τη σύντομη αυτή παρουσίαση των κυκλωμάτων χρονισμού με μία αναφορά στα εξωτερικά κυκλώματα που μπορούμε να χρησιμοποιήσουμε, προκειμένου να εξασφαλίσουμε τους κύκλους του ωρολογιακού σήματος για τα κυκλώματα του μικροελεγκτή. Το Σχήμα 4.2 παρουσιάζει πώς συνδέουμε έναν κρυσταλλικό ταλαντωτή με τη βοήθεια δύο πυκνωτών. Ας σημειωθεί ότι μία τιμή 22-33pF για τους πυκνωτές C1 και C2 είναι κατάλληλη στις περισσότερες περιπτώσεις.

Ένας ταλαντωτής με ιδιοσυχνότητα 4MHz είναι μια συνηθισμένη και φθηνή λύση, η οποία μάλιστα εξασφαλίζει ικανοποιητικές ταχύτητες για τις περισσότερες εφαρμογές. Επιπλέον έχει το πλεονέκτημα ότι μας βοηθά να προγραμματίζουμε εφαρμογές στις

οποίες ο χρονισμός αποτελεί κρίσιμο παράγοντα. Ο λόγος είναι ότι με αυτήν την εξωτερική πηγή χρονισμού, η εκτέλεση κάθε εντολής διαρκεί ακριβώς 1μs, αφού όπως προαναφέραμε ο κύκλος εκτέλεσης κάθε εντολής διαρκεί τέσσερις συνολικά κύκλους του εξωτερικού ρολογιού. Έτσι διευκολύνεται σημαντικά η καταμέτρηση του χρόνου κατά την εκτέλεση του προγράμματος.



Σχήμα 4.2 Σύνδεση κρυσταλλικού ταλαντωτή



Σχήμα 4.3 Σύνδεση ταλαντωτή RC

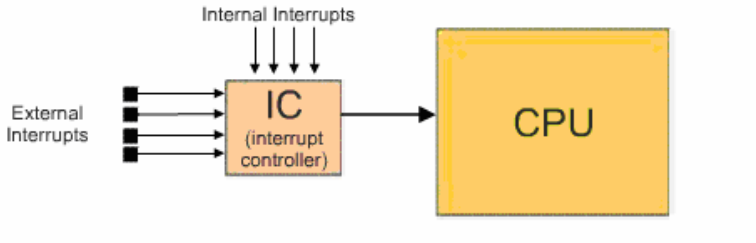
Σε απλούστερες εφαρμογές, όπου η ταχύτητα και η ακρίβεια της συχνότητας του ωρολογιακού σήματος δεν παίζουν ιδιαίτερο ρόλο, είναι δυνατό να χρησιμοποιούμε για τον χρονισμό απλούς ταλαντωτές RC. Μία τέτοια τυπική συνδεσμολογία παρουσιάζεται στο Σχήμα 4.3.

Ο χρονισμός των διαφόρων εφαρμογών απαιτεί συχνά την αναμονή σε κάποιο βρόγχο καθυστέρησης. Η χρονική ακρίβεια με την οποία εκτελείται μια υπορουτίνα καθυστέρησης επιτρέπει την υλοποίηση εφαρμογών με ιδιαίτερες απαιτήσεις στην καταμέτρηση χρονικών διαστημάτων.

5. Σήματα διακοπής στους μικροελεγκτές PIC16F

5.1 Σήματα Διακοπής (Interrupts)

Τα σήματα διακοπής αποτελούν γενικά τον πιο αποδοτικό τρόπο χρήσης της κεντρικής μονάδας επεξεργασίας (CPU), ειδικά όταν αυτή πρόκειται να εξυπηρετήσει περισσότερες από μία ανάγκες. Ας υποθέσουμε ότι ο μικροελεγκτής πρέπει να εκτελέσει μία συγκεκριμένη ακολουθία εντολών, όταν κάποιος ακροδέκτης εισόδου δεχτεί λογικό 1. Ένας τρόπος είναι η CPU να ελέγχει διαρκώς τον συγκεκριμένο ακροδέκτη, κάτι που την υποχρεώνει να αφιερώνει χρήσιμο χρόνο σε μια ελάχιστα παραγωγική εργασία. Ο άλλος τρόπος είναι το λογικό 1 να απευθύνεται σε κάποιον από τους ακροδέκτες που ο μικροελεγκτής χρησιμοποιεί για να δέχεται σήματα διακοπών. Μέχρι να δεχτεί κάποια διακοπή, η CPU μπορεί να εκτελεί τη βασική αλληλουχία εντολών του κυρίως προγράμματος, για παράδειγμα να παίρνει μετρήσεις κάποιου φυσικού μεγέθους. Όταν δεχτεί ένα σήμα διακοπής, η CPU σταματά την εκτέλεση του προγράμματος, αποθηκεύει τη διεύθυνση της επόμενης εντολής στη *στοίβα*, πηγαίνει στην εντολή 4 του κυρίως προγράμματος και συνεχίζει την εκτέλεση από εκεί. Συνήθως, στη θέση 4 βρίσκεται μια εντολή GOTO που στέλνει το πρόγραμμα στην υπορουτίνα που εξυπηρετεί τις ανάγκες της διακοπής. Για παράδειγμα, η υπορουτίνα αυτή μπορεί να θέσει για λίγο σε λειτουργία κάποιον βηματικό κινητήρα, πριν επιστρέψει ο έλεγχος στο κυρίως πρόγραμμα και το σύστημα συνεχίσει να εκτελεί μετρήσεις.



Σχ. 5.1 Ο ελεγκτής διακοπών ενός μικροελεγκτή

Στο σχ. 5.1 φαίνεται η σχέση του ελεγκτή διακοπών ενός μικροελεγκτή με την CPU και του εξωτερικούς ακροδέκτες του κυκλώματος.

Ο μικροελεγκτής PIC16F877 μπορεί να δεχτεί σήματα διακοπών κυρίως από τις εξής πηγές.

- Εξωτερική διακοπή από τον ακροδέκτη RB0/INT.
- Υπερχείλιση του απαριθμητή-χρονιστή **Timer0**.

- Κάποια αλλαγή της κατάστασης των ακροδεκτών RB7-RB4 της θύρας **B**.

5.2 Ο καταχωρητής ελέγχου διακοπών (INTCON)

Ο καταχωρητής που ρυθμίζει την ενεργοποίηση των διακοπών και καταγράφει ποιες διακοπές σημειώθηκαν είναι ο καταχωρητής **INTCON** (*Interrupt Control*). Η ονομασία και η σειρά των bits αυτού του καταχωρητή παρουσιάζεται στον Πίνακα 4-1.

Πίνακας 5-1
Τα bits του καταχωρητή ελέγχου διακοπών INTCON

b7	b6	b5	b4	b3	b2	b1	b0
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Το σημαντικότερο bit του καταχωρητή **INTCON**, το bit b7, είναι το bit της γενικής ενεργοποίησης των διακοπών (*General Interrupt Enable*, *GIE*) και όταν είναι μηδέν δεν επιτρέπει να συμβεί καμία διακοπή. Τα υπόλοιπα bits του καταχωρητή διακρίνονται σε δύο κατηγορίες, αυτά που ενεργοποιούν τις επιμέρους διακοπές (IE) και αυτά που δηλώνουν ότι σημειώθηκε κάποια διακοπή (*σημαίες διακοπών*, IF). Καθένας από τους βασικούς τύπους διακοπών, που αναφέραμε προηγουμένως έχει το δικό του ζευγάρι IE και IF. Έτσι, για να ενεργοποιηθεί η δυνατότητα να εμφανιστεί διακοπή εξαιτίας της υπερχείλισης του καταχωρητή **TMR0**, του χρονιστή **Timer0,** πρέπει πρώτα να τεθεί σε λογικό 1 το bit GIE και κατόπι να τεθεί σε λογικό 1 το bit 5 (T0IE). Μόλις εμφανιστεί η υπερχείλιση, η αντίστοιχη σημαία διακοπής T0IF τίθεται σε λογικό 1 και ακολουθεί η διαδικασία που περιγράψαμε: η τιμή του απαριθμητή προγράμματος, που περιέχει τη διεύθυνση της επόμενης εντολής αποθηκεύεται στη *στοίβα* (*stack*) και το πρόγραμμα μεταβαίνει στη διεύθυνση 0x004 της μνήμης του προγράμματος, προκειμένου να εξυπηρετηθεί η διακοπή.

Το bit που ενεργοποιεί τις διακοπές που προέρχονται από τα τέσσερα ανώτερα bits της θύρας **B** είναι το bit RBIE και η αντίστοιχη σημαία είναι το bit RBIF. Παρομοίως, οι διακοπές που προέρχονται από σήματα στον ακροδέκτη RB0/INT ενεργοποιούνται και σημειώνονται από το bit 4 (INTE) και το bit 1 (INTF) αντίστοιχα, του καταχωρητή **INTCON**. Το είδος του μετώπου (θετικό ή αρνητικό) που προκαλεί τη διακοπή στον ακροδέκτη RB0/INT ορίζεται από το bit INTEDG του καταχωρητή **OPTION_REG**.

Όταν καλείται η υπορουτίνα εξυπηρέτησης της διακοπής, απενεργοποιούνται οι επιπλέον διακοπές και έτσι το σύστημα δεν μπορεί να δεχτεί άλλη διακοπή μέχρι να επιστρέψει από αυτήν που ήδη άρχισε να εξυπηρετεί. Στο τέλος της υπορουτίνας της διακοπής ο χρήστης οφείλει να μηδενίσει εκ νέου τη σημαία της διακοπής και να ενεργοποιήσει ξανά τις διακοπές, τοποθετώντας σε λογικό 1 το bit GIE. Αυτό επιτυγχάνεται με χρήση της εντολής **RETFIE** στο τέλος της υπορουτίνας της διακοπής. Η εντολή αυτή είναι μια παραλλαγή της εντολής **RETURN**, και με αυτήν αφενός επιτυγχάνεται η επιστροφή στο κυρίως πρόγραμμα, στη διεύθυνση της εντολής που

αποθηκεύτηκε στη στοίβα, και αφετέρου ενεργοποιούνται ξανά οι διακοπές, με τοποθέτηση του GIE σε λογικό 1.

5.3 Κώδικας επίδειξης σημάτων διακοπής

Το επόμενο πρόγραμμα επιδεικνύει τη λειτουργία των σημάτων διακοπής στους μικροελεγκτές PIC. Το κυρίως πρόγραμμα βρίσκεται μετά την ετικέτα Start. Αυτό, κάνει απλά τις κατάλληλες αρχικοποιήσεις στις τιμές των καταχωρητών και στη συνέχεια μπαίνει σε έναν βρόγχο αναμονής (loop goto loop).

Η υπορουτίνα διακοπής ακολουθεί αμέσως μετά την δήλωση Org 4. Τοποθετείται στη μνήμη προγράμματος ξεκινώντας από την θέση μνήμης 04. Η λειτουργία της είναι να αυξάνει έναν μετρητή κατά ένα, κάθε φορά που συμβαίνει εξωτερική διακοπή. Το αποτέλεσμα της απαρίθμησης των διακοπών εμφανίζεται στη θύρα C.

Η πηγή των σημάτων διακοπής ορίζεται να είναι ο ακροδέκτης RB0 της θύρας B του μικροελεγκτή. Η διακοπή αυτή ενεργοποιείται με τη βοήθεια του bit INTE του καταχωρητή INTCON.

```
#include "P16F877.inc"

TEMP equ 20h
        Org 0
        goto start              ;Πήγαινε στο κυρίως πρόγραμμα
        Org 4                   ;Τοποθέτησε την επόμενη εντολή στη θέση 4 του προγράμματος

        incf TEMP, F            ;Αύξησε τον καταχωρητή TEMP κατά 1
        movf TEMP,W
        movwf PORTC             ;Μετέφερε τον TEMP στην θύρα C
        bcf INTCON, INTF        ;Μηδένισε την σημαία της διακοπής
        retfie                  ;Επέστρεψε από την διακοπή
start
        clrf TEMP               ;Μηδένισε τον TEMP
        bsf STATUS, RP0         ;Πήγαινε στην σελίδα μνήμης 1
        movlw b'00000000'
        movwf TRISC             ;Κάνε την θύρα C έξοδο
        bcf STATUS, RP0         ;Επέστρεψε στη σελίδα μνήμης 0
        bsf INTCON, INTE        ;Ενεργοποίησε την διακοπή RB0
        bcf INTCON, INTF        ;Μηδένισε την σημαία της διακοπής RB0
        bsf INTCON, GIE         ;Ενεργοποίησε τις διακοπές
loop    goto loop               ;Περίμενε για διακοπή
        END
```

Κώδικας 5.1 Κώδικας που κάνει χρήση σήματος διακοπής από τον ακροδέκτη INT

5.4 Μετάβαση σε υπορουτίνα διακοπής – Context saving

Ένα τελευταίο σημείο προσοχής, που πρέπει να λαμβάνεται υπόψη όταν συμβαίνει μία διακοπή είναι ότι κατά τη διάρκεια της διακοπής πιθανότατα θα μεταβληθούν οι τιμές σημαντικών καταχωρητών, γεγονός που μπορεί να δημιουργήσει προβλήματα μετά τη επιστροφή στο κυρίως πρόγραμμα. Για το λόγο αυτό είναι σκόπιμο οι τιμές των σημαντικών καταχωρητών να αποθηκεύονται κατά την εκκίνηση της υπορουτίνας της διακοπής, ώστε να ανακτώνται μετά την εκτέλεση της υπορουτίνας. Οι σημαντικότεροι καταχωρητές που είναι σκόπιμο να αποθηκεύονται και να ανακτώνται είναι ο καταχωρητής εργασίας **W** και ο καταχωρητής κατάστασης **STATUS**.
Παρακάτω, παρουσιάζεται ο κώδικας για την αποθήκευση των βασικών καταχωρητών, καθώς και ο κώδικας για την ανάκτησή τους, κατά το πέρας της υπορουτίνας διακοπής.

```
;Στο κυρίως πρόγραμμα:
SaveWReg equ OC      ;Ορίζουμε έναν καταχωρητή με όνομα SaveWReg
SaveStatus equ 0D    ;Ορίζουμε έναν καταχωρητή με όνομα SaveStatus

;Στην αρχή της υπορουτίνας διακοπής γράφουμε τις παρακάτω εντολές αποθήκευσης
movwf SaveWReg
swapf STATUS, w      ;Ανταλλαγή των τετράδων του καταχωρητή STATUS
movwf SaveStatus
```

Κώδικας 5.2 Αποθήκευση του περιεχομένου των βασικών καταχωρητών κατά τη μετάβαση σε υπορουτίνα εξυπηρέτησης διακοπής

Η εντολή **SWAPF**, που χρησιμοποιείται στο σημείο αυτό του προγράμματος, ανταλλάσσει τα τέσσερα κατώτερα bits (b0, b1, b2 και b3) με τα τέσσερα ανώτερα bits (b4, b5, b6 και b7). Ο λόγος που χρησιμοποιούμε αυτήν την εντολή για τη μεταφορά δεδομένων είναι ότι ειδικά αυτή η εντολή αφήνει αμετάβλητο τον καταχωρητή κατάστασης. Έτσι είμαστε βέβαιοι ότι δεν θα μεταβληθεί το περιεχόμενο του καταχωρητή **STATUS** κατά τη διαδικασία αποθήκευσης και ανάκτησης των σημαντικών καταχωρητών.

```
swapf SaveStatus, W  ;Ανταλλαγή τετράδων του καταχωρητή SaveStatus
                     ;και μεταφορά στον W
movwf STATUS         ;Αντιγραφή των περιεχομένων του καταχωρητή W στον
                     ;καταχωρητή STATUS
swapf SaveWReg       ;Ανταλλαγή των τετράδων του καταχωρητή SaveWReg
swapf SaveWReg, w    ;Επαναφορά στον W της αρχικής του τιμής
```

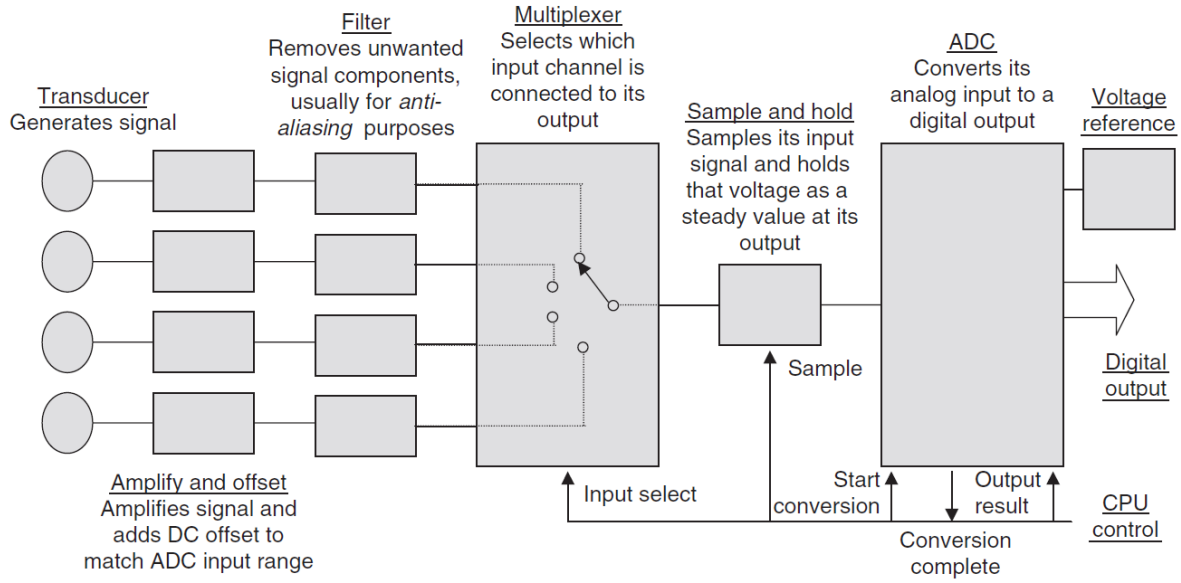
Κώδικας 5.3 Ανάκτηση του περιεχομένου των βασικών καταχωρητών κατά την επιστροφή από υπορουτίνα εξυπηρέτησης διακοπής

6. Μετατροπή αναλογικού σήματος σε ψηφιακό στους μικροελεγκτές PIC16F

6.1 Εισαγωγή

Στο σχήμα 6.1 φαίνεται ένα τυπικό σύστημα συλλογής δεδομένων (data acquisition). Αποτελείται από τους αισθητήρες, που μετατρέποουν τη φυσική ποσότητα που μετράται σε αναλογικό ηλεκτρικό σήμα, τα κυκλώματα ρύθμισης (signal conditioning), που ενισχύουν και φιλτράρουν το σήμα, και τέλος το σύστημα μετατροπής του αναλογικού σήματος σε ψηφιακό (ADC – analog to digital converter).

Το τελευταίο αυτό μέρος περιλαμβάνει συνήθως 1. έναν πολυπλέκτη, που επιλέγει το σήμα που θα υποστεί τη μετατροπή, 2. ένα κύκλωμα δειγματοληψίας και συγκράτησης (sample and hold) το οποίο λαμβάνει ένα στιγμιαίο δείγμα της αναλογικής τάσης και με τη βοήθεια ενός πυκνωτή συγκρατεί την τιμή του όσο χρειάζεται προκειμένου να γίνει η μετατροπή του δείγματος σε δυαδικό κώδικα και τέλος 3. ένα κύκλωμα που παράγει ένα δυαδικό κώδικα με *n* bits, που η δεκαδική στάθμη του είναι ανάλογη με την αρχική αναλογική τιμή. Δηλαδή, όσο μεγαλύτερη η αναλογική τάση του σήματος εισόδου, τόσο υψηλότερη είναι η κβαντική στάθμη του δυαδικού κώδικα που παράγεται στην έξοδο. Προφανώς, ένας ADC με *n* bits υποστηρίζει στάθμες από 0 μέχρι 2^n-1.



Σχ. 6.1 Ένα τυπικό σύστημα συλλογής δεδομένων

Ένας ADC λαμβάνει μια τάση αναφοράς (Voltage reference - V_{ref}) η οποία ρυθμίζει ποια αναλογική τάση εισόδου θα αντιστοιχίζεται στην πρώτη, τη δεύτερη, μέχρι και τη μέγιστη ψηφιακή κβαντική στάθμη. Για παράδειγμα, ένας μετατροπέας 8-bit, θα υποστηρίζει συνολικά 256 κβαντικές στάθμες. Όταν η στάθμη αναφοράς V_{ref} είναι 5V, τότε η τάση αυτή μοιράζεται σε όλες τις διαθέσιμες κβαντικές στάθμες, οπότε από τη μια στάθμη στην επόμενη αντιστοιχεί ένα μικρό βήμα αναλογικής τάσης V_{step}, ως εξής:

$$V_{step} = \frac{V_{ref}}{256} = \frac{5V}{256} = 0{,}0195V \qquad (6.1)$$

Αν η αναλογική τάση της εισόδου είναι V_{an}, τότε η κβαντική στάθμη που αντιστοιχεί είναι

$$\kappa\beta\alpha\nu\tau\iota\kappa\acute{\eta}\ \sigma\tau\acute{\alpha}\theta\mu\eta = \frac{V_{an}}{V_{step}} \qquad (6.2)$$

Έτσι, ο μετατροπέας αντιστοιχίζει κάθε αναλογική τάση της εισόδου του σε κβαντικές στάθμες, σύμφωνα με τον Πίνακα 6-1.

Πίνακας 6-1
Αντιστοιχία αναλογικής τάσης και δυαδικής τιμής σε ADC 8-bit

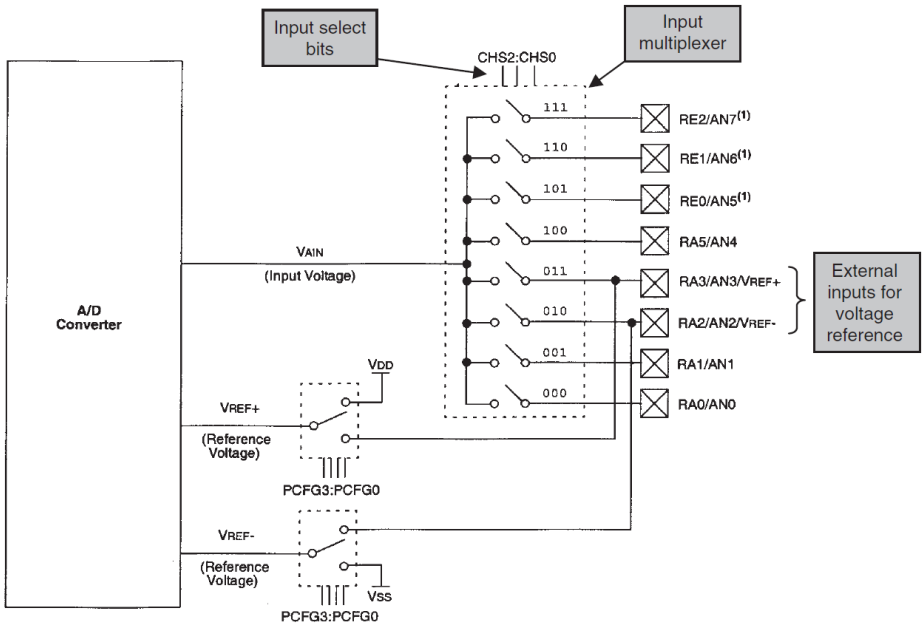
Αναλογική είσοδος (V_{an})	**Κβαντική στάθμη εξόδου**	**Δυαδική τιμή εξόδου**
0 V	0	00000000
0,9945 V	51	00110011
2,496 V	128	10000000
3,49 V	179	10110011
4,98 V	255	11111111

6.2 Κύκλωμα ADC στον μικροελεγκτή PIC16F877

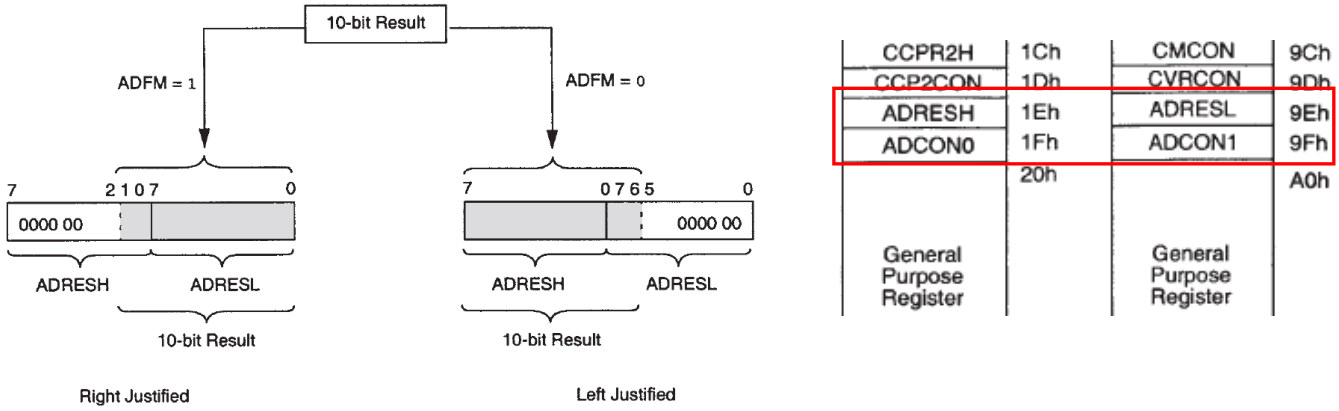
Το σχήμα 6.2 παρουσιάζει το κύκλωμα μετατροπής αναλογικού σήματος σε ψηφιακό στους μικροελεγκτές της Microchip. Ένας αριθμός ακροδεκτών εισόδου προορίζεται να λάβει είσοδο αναλογικού σήματος. Για να λειτουργήσουν αυτοί οι ακροδέκτες ως αναλογικές είσοδοι θα πρέπει να γίνει η κατάλληλη επιλογή, μέσω ενός καταχωρητή που ρυθμίζει τη λειτουργία του ADC. Στην είσοδο υπάρχει το κύκλωμα πολυπλεξίας που επιλέγει ένα αναλογικό σήμα εισόδου. Η επιλογή του καναλιού εισόδου που μετατρέπεται κάθε φορά, γίνεται πάλι με τη βοήθεια ειδικού καταχωρητή (input select

bits). Η τάση αναφοράς τίθεται αυτόματα στα 5V που είναι η τάση τροφοδοσίας, αλλά μπορεί και να ρυθμιστεί.

Ο μετατροπέας του ADC του PIC16F877 έχει ανάλυση 10 bit. Αυτό σημαίνει ότι υποστηρίζει 1024 κβαντικές στάθμες. Όταν ολοκληρώνεται μια μετατροπή, το αποτέλεσμα τοποθετείται σε δύο καταχωρητές, τους ADRESH και ADRESL. Το αποτέλεσμα μπορεί να τοποθετηθεί με δεξιά ή αριστερή στοίχηση, όπως φαίνεται στο σχήμα 6.3, ανάλογα με τις ρυθμίσεις που θα γίνουν στον καταχωρητή ADCON1.



Σχ. 6.2 Κύκλωμα ADC στους μικροελεγκτές PIC.



Σχ. 6.3 Καταχώρηση αποτελέσματος μετατροπής στους καταχωρητές ADRESH και ADRESL, με δεξιά και αριστερή στοίχηση. Δεξιά στο σχήμα, φαίνονται οι βασικοί καταχωρητές που σχετίζονται με τον ADC.

6.3 Οι βασικοί καταχωρητές του μετατροπέα ADC

Όπως όλα τα περιφερειακά του PIC, έτσι και ο ADC ρυθμίζεται ώστε να επιτελεί τις επιθυμητές λειτουργίες, με τη βοήθεια ορισμένων καταχωρητών ειδικού σκοπού. Οι βασικοί καταχωρητές είναι ο ADCON0 και ADCON1. Όπως αναφέρθηκε, οι καταχωρητές ADRESL και ADRESH αποθηκεύουν το αποτέλεσμα της μετατροπής. Με αριστερή στοίχηση, ο ADRESH αποθηκεύει τα 8 πιο σημαντικά bits της μετατροπής, άρα μπορεί να χρησιμοποιηθεί αν επιθυμούμε να απλουστεύσουμε τον μετατροπέα και να τον χρησιμοποιήσουμε σαν να είχε ανάλυση 8-bits.
Όλες τις λεπτομέρειες για τις λειτουργίες που ρυθμίζουν τα διάφορα bits των καταχωρητών, ο ενδιαφερόμενος χρήστης μπορεί να τις βρει στα εγχειρίδια χρήσης του μικροελεγκτή που δημοσιεύονται διαδικτυακά από την εταιρία Microchip. Οι απαραίτητες ρυθμίσεις περιγράφονται συνοπτικά παρακάτω.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

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

bit 0 **ADON:** A/D On bit
1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Σχ. 6.4 Ο καταχωρητής ADCON0 και οι λειτουργίες που επιτελούν τα bits που αυτός διαθέτει.

1. Επιλογή συχνότητας ρολογιού ADC: Ο μετατροπέας χρονίζεται με τη βοήθεια γεννήτριας παλμών, που λαμβάνεται με υποδιαίρεση της συχνότητας του εξωτερικού κρυστάλλου. Η υποδιαίρεση ρυθμίζεται με τα bits ADCS2, ADCS1, ADCS0, από τα οποία τα δύο λιγότερα σημαντικά ανήκουν στον καταχωρητή ADCON0, ενώ το πιο σημαντικό (ADCS2) ανήκει στον ADCON1 (βλέπε σχήματα 6.4 και 6.5). Μια κατάλληλη υποδιαίρεση για το ρολόϊ του μετατροπέα είναι Fosc/16, που αντιστοιχεί σε bits επιλογής 101.
2. Τα bits CHS2:CHS0 του ADCON0 επιλέγουν κανάλι εισόδου, μέσω του πολυπλέκτη εισόδου. Έτσι, για να γίνει η μετατροπή του καναλιού Ch0 (RA0) αυτά πρέπει να λάβουν την τιμή 000.
3. Ο καταχωρητής ADCON1 διαθέτει τέσσερα bits PCFG3:PCFG0 που ρυθμίζουν τη διαμόρφωση των εισόδων AN0 έως AN7, δηλαδή ποιές θα λειτουργούν ως αναλογικές και ποιες ως ψηφιακές. Επίσης, κάποιες μπορούν να διαμορφωθούν ώστε να δεχτούν είσοδο αναφοράς.
4. Το bit ADFM του καταχωρητή ADCON1 ρυθμίζει τη στοίχηση της λέξης 10-bit που παράγει ο ADC. Συνιστάται αριστερή στοίχηση (ADFM=0).

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O

Σχ. 6.5 Ο καταχωρητής ADCON1 και οι λειτουργίες που επιτελούν τα bits που αυτός διαθέτει.

Τέλος, για να λειτουργήσει ο ADC πρέπει να θέσουμε το bit ADON του ADCON0 σε λογικό 1 (power up). Για να γίνει η δειγματοληψία μιας νέας αναλογικής τάσης, η σημαία ADIF πρέπει να μηδενιστεί. Είναι απαραίτητο να προβλέψουμε μια μικρή περίοδο αδράνειας, ώστε να σταθεροποιηθεί η τάση στην είσοδο του κυκλώματος δειγματοληψίας και συγκράτησης. Αυτό επιτυγχάνεται με δύο ή τρεις εντολές nop.
Για να ξεκινήσει μια νέα μετατροπή θέτουμε σε λογικό 1 το bit GO του ADCON0.
Όταν ολοκληρωθεί η μετατροπή η σημαία ADIF γίνεται 1. Η σημαία αυτή βρίσκεται στον καταχωρητή PIR1, που ρυθμίζει τα σήματα διακοπής περιφερειακών συσκευών. Όταν η σημαία ADIF γίνει 1, το αποτέλεσμα βρίσκεται στους καταχωρητές ADRESH, ADRESL, και μπορεί να διαβαστεί. Παρακάτω δίνεται ένα πλήρες παράδειγμα κώδικα για τη μετατροπή ADC.

```
        #include “p16f877.inc”
        Org 00

        ;initialize ADC
        bsf STATUS, RP0
        movlw b'00011111'    ;5 first bits of PORTA inputs
        movwf TRISA
        movlw b'00000000'
        movwf TRISB
        movlw b'01000010'    ;left justified, ADCS2=1, 3 Dig 5 Analog ch
        movwf ADCON1
        bcf STATUS, RP0
        movlw b'01000001'    ;101 Fosc/16, ch0, ADON
        movwf ADCON0
        clrf PORTB

        ;conversion
loop1   bcf PIR1, ADIF
        nop                  ;wait for the output of S&H circuit to stabilize
        nop
        nop
        bsf ADCON0, GO
wait    btfss PIR1, ADIF
        goto wait

        ;read ADC
        movf ADRESH, w
        movwf PORTB
        goto loop1
        end
```

Κώδικας 6.1 Μετατροπή αναλογικού σήματος σε ψηφιακό (χρήση μόνον 8-bit)

7. Ασύγχρονη σειριακή μετάδοση –το κύκλωμα UART

7.1 Σύγχρονη και ασύγχρονη σειριακή μετάδοση

Συχνά μια εφαρμογή πραγματικού χρόνου χρειάζεται να επικοινωνήσει με άλλα μικροϋπολογιστικά συστήματα ή συσκευές, ειδικά σε κατανεμημένα συστήματα, όπου η πληροφορία παράγεται τοπικά, μεταδίδεται μέσω καναλιών και συλλέγεται ή υφίσταται επεξεργασία κεντρικά. Η μετάδοση της πληροφορίας μπορεί να γίνει παράλληλα, με ταυτόχρονη μετάδοση όλων των bits κάθε ψηφιολέξης ή σειριακά. Στη σειριακή μετάδοση η πληροφορία μεταδίδεται ως μια ακολουθία από bits από τον πομπό προς το δέκτη. Γενικά διακρίνουμε τη σύγχρονη και την ασύγχρονη σειριακή μετάδοση.

Στη *σύγχρονη* σειριακή επικοινωνία πομπός και δέκτης πρέπει να είναι συγχρονισμένοι για την εκπομπή και λήψη των δεδομένων. Ταυτόχρονα με την εκπομπή των δεδομένων εκπέμπεται και το clock, για τον συγχρονισμό της επικοινωνίας.

Στην *ασύγχρονη* μετάδοση, τα δεδομένα μπορεί να εμφανιστούν στη γραμμή οποιαδήποτε χρονική στιγμή, χωρίς κανένα συγχρονισμό με ωρολογιακά σήματα. Ο μόνος απαραίτητος χρονισμός υλοποιείται με το start bit, ενώ ο συγχρονισμός πομπού και δέκτη επιτυγχάνεται διατηρώντας σταθερό το ρυθμό εκπομπής και λήψης (baud rate).

Στο κεφάλαιο αυτό θα μελετήσουμε τη χρήση της σειριακής θύρας ως μονάδας ασύγχρονης σειριακής επικοινωνίας στον μικροελεγκτή PIC16F877.

7.2 Ασύγχρονη Σειριακή Επικοινωνία

Η ασύγχρονη σειριακή επικοινωνία εξυπηρετεί τη μετάδοση χαρακτήρων που εκπέμπονται από κάποιο πομπό χωρίς κανένα συγχρονισμό, όπως συμβαίνει με τους αλφαριθμητικούς χαρακτήρες που δημιουργούνται όταν πιέζουμε τα πλήκτρα ενός πληκτρολογίου.

Κάθε τέτοιος χαρακτήρας μετατρέπεται σε ψηφιολέξη μέσω του κώδικα ASCII και η ακολουθία bits που αντιστοιχεί σε αυτόν εμφανίζεται στη γραμμή μετάδοσης. Ο δέκτης πρέπει να είναι σε θέση να αναγνωρίζει ότι έφτασε ένας χαρακτήρας και να δέχεται τα bits του χαρακτήρα με τη σωστή σειρά και χωρίς απώλειες.

Για τον παραπάνω σκοπό, τα bits της ασύγχρονης σειριακής μετάδοσης οργανώνονται σε ομάδες των εννέα έως δώδεκα bits συνολικά, οι οποίες περιέχουν κάποιους χαρακτήρες έναρξης και λήξης (Σχήμα 7.1). Το πρώτο bit κάθε πλαισίου είναι το λεγόμενο START BIT, το οποίο αντιστοιχεί σε λογικό μηδέν. Ακολουθεί η σειρά των ψηφίων του χαρακτήρα που αποστέλλεται. Για παράδειγμα, εάν αποστέλλεται το κεφαλαίο γράμμα Α, τότε η ακολουθία των ψηφίων θα είναι 01001011. Μετά από τα bits του χαρακτήρα ακολουθεί ένα bit άρτιας ή περιττής *ισοτιμίας* (*parity*), το οποίο ενεργοποιεί μία διαδικασία ελέγχου σφαλμάτων, για να ανιχνεύσει τυχόν λάθη που

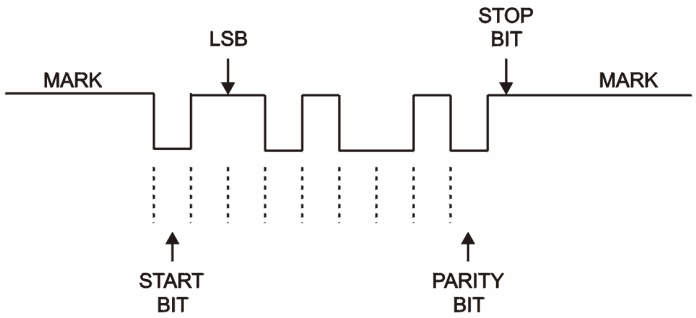
συνέβησαν κατά τη μετάδοση. Το πλαίσιο κλείνει με ένα ή δύο STOP BITS, που υποδηλώνουν το τέλος του χαρακτήρα και την κατάσταση αναμονής για τον επόμενο. Η λογική κατάσταση των ψηφίων λήξης (STOP BITS) είναι το λογικό ένα.

Όταν δεν μεταδίδεται κάποιος χαρακτήρας, τότε λέμε ότι η σύνδεση είναι *ανενεργή*, οπότε η γραμμή ευρίσκεται σε λογικό ένα. Η κατάσταση αυτή ονομάζεται «*συνθήκη MARK*». Όταν μεταδοθεί το bit έναρξης (START BIT) και φθάσει στο δέκτη, ο δέκτης καταλαβαίνει ότι ακολουθούν τα bits του χαρακτήρα που αποστέλλεται, οπότε ενεργοποιεί το σύστημα χρονισμού του και διαβάζει με τη σειρά τα επόμενα bits μέχρι τα bits λήξης (STOP BITS). Στη συνέχεια τίθεται και πάλι στην κατάσταση αναμονής.

Είναι φανερό ότι η διάρκεια του κάθε εκπεμπόμενου bit στην ασύγχρονη σειριακή μετάδοση πρέπει να είναι αυστηρά η ίδια, ώστε να μπορεί ο δέκτης, με βάση κάποιο σύστημα χρονισμού, να διακρίνει τα bits μεταξύ τους. Συνεπώς ο πομπός και ο δέκτης πρέπει να συμφωνούν ως προς την ταχύτητα της σειριακής μετάδοσης των bits. Η ταχύτητα αυτή ορίζει τον λεγόμενο «*ρυθμό μετάδοσης*» (*baud rate*), που μετριέται σε *bits ανά δευτερόλεπτο* (*bits/sec* ή *bps*). Συνήθεις ρυθμοί στις ασύγχρονες σειριακές επικοινωνίες είναι 2400, 4800, 9600, 14400, 19200, 28800 και 33600 bits/sec. Η μέγιστη ταχύτητα που υποστηρίζει μια θύρα UART κατά την αποστολή χαρακτήρων από έναν υπολογιστή προς μία συσκευή επικοινωνίας είναι 115.2 kbps. Τυπική ασύγχρονη σειριακή συσκευή είναι το modem, που διασυνδέει τον υπολογιστή ή κάποιο τερματικό με την τηλεφωνική γραμμή.

Σαν παράδειγμα αναφέρουμε ότι, για να έχουμε ρυθμό μετάδοσης 9600 bps, η διάρκεια του κάθε bit πρέπει να είναι 104 μs. Κάθε χαρακτήρας θα διαρκεί στη γραμμή 1.14 ms. *Ας σημειωθεί ότι η ακρίβεια στη διάρκεια του κάθε bit είναι σημαντική, ώστε να υπάρχει συγχρονισμός πομπού και δέκτη*.

Το βασικό μειονέκτημα της ασύγχρονης σειριακής μετάδοσης είναι η ανάγκη που προκύπτει για START και STOP bits στην αρχή και στο τέλος κάθε χαρακτήρα. Με τον τρόπο αυτό επιβαρύνεται σημαντικά η διαδικασία της μετάδοσης με επιπλέον bits που δεν αντιπροσωπεύουν χρήσιμη πληροφορία.



Σχήμα 7.1 Μορφή του σήματος στην ασύγχρονη σειριακή μετάδοση χαρακτήρα

7.3 Το Πρωτόκολλο RS-232C

Το πρωτόκολλο RS-232C επιτρέπει την ασύγχρονη σειριακή επικοινωνία ανάμεσα σε δύο συσκευές. Εάν επιθυμούμε να συνδέσουμε περισσότερες από δύο συσκευές σε έναν υπολογιστή, χρειαζόμαστε περισσότερες από μία σειριακές θύρες. Υπάρχουν, βέβαια, και άλλα σειριακά πρωτόκολλα, όπως το πρωτόκολλο σύγχρονης επικοινωνίας I^2C, που επιτρέπουν τη διασύνδεση πολλών συσκευών σε ένα σειριακό κύκλωμα.

Το πρωτόκολλο RS-232C χρησιμοποιεί αρνητική ψηφιακή λογική και μεγάλες στάθμες, ώστε να επιτρέπει τη διάδοση του σήματος σε μεγάλες αποστάσεις χωρίς απώλειες. Αυτά έχουν σαν αποτέλεσμα οι τάσεις του πρωτοκόλλου RS-232C να μην είναι συμβατές με τις στάθμες TTL. Τα επίπεδα τάσεων του πρωτοκόλλου RS-232C, σύμφωνα με τις προδιαγραφές που θέσπισε η Ένωση EIA, φαίνονται στον πίνακα 7.1.

Αν και σύμφωνα με το πρωτόκολλο ο μέγιστος ρυθμός μετάδοσης (baud rate) δεν ξεπερνά τα 19.2 kbps, οι σημερινές ταχύτητες μπορεί να είναι πολύ μεγαλύτερες.

Με χρήση του πρωτοκόλλου RS-232C, ένα τερματικό χαρακτήρων (ASCII terminal) μπορεί να αποστείλει μέσω μιας γραμμής επικοινωνίας δεδομένα, σύμφωνα με τους κανόνες της ασύγχρονης σειριακής μετάδοσης που περιγράψαμε στην παραπάνω παράγραφο.

Όταν η γραμμή είναι ανενεργή, ευρίσκεται σε συνθήκη MARK, δηλαδή −12 V περίπου, που αντιστοιχούν σε λογικό ένα. Η γραμμή ενεργοποιείται με τη συνθήκη SPACE (λογικό μηδέν ή +12V). Ακολουθεί η μετάδοση επτά ή οκτώ bits για τον αποστελλόμενο χαρακτήρα, ένα προαιρετικό bit άρτιας ή περιττής ισοτιμίας (parity) και ένα ή δύο STOP bits (συνθήκη MARK), που σηματοδοτούν το τέλος του χαρακτήρα (βλέπε Σχήμα 7.1).

Πίνακας 7.1 Προδιαγραφές τάσεων και ρευμάτων πρωτοκόλλου RS232

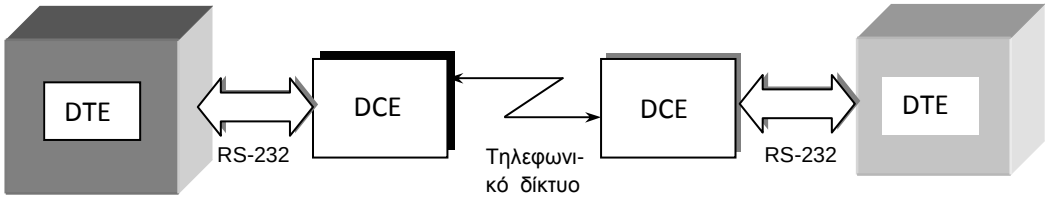
1. Το λογικό 0, που λέγεται και **SPACE**, βρίσκεται μεταξύ +3 και +25 V (στην πράξη λαμβάνονται και εκπέμπονται από +5 έως +15 V).
2. Το λογικό 1, που λέγεται και **MARK**, βρίσκεται μεταξύ −3 και −25V (πρακτικά από −5 έως −15V).
3. Η περιοχή από −3V έως +3V δεν αντιπροσωπεύει καθορισμένη λογική στάθμη.
4. Κανένας από τους ακροδέκτες της σειριακής θύρας δεν μπορεί να δεχτεί δυναμικό μεγαλύτερο από 25V σε σχέση με τη γη.
5. Το μέγιστο ρεύμα δεν μπορεί να ξεπερνά τα 500mA.

7.4 Τύποι Ασύγχρονων Σειριακών Συσκευών (DTE/DCE)

Το πρωτόκολλο RS-232 καθορίζει δύο τύπους συσκευών. Ο πρώτος τύπος συσκευής ονομάζεται *Τερματική Συσκευή Δεδομένων* (*Data Terminal Equipment* ή *DTE*). Ο δεύτερος τύπος συσκευής ονομάζεται *Συσκευή Επικοινωνίας Δεδομένων* (*Data Communications Equipment* ή *DCE*).

Κάθε προσωπικός υπολογιστής η μικροϋπολογιστής σε τσιπ που διαθέτει θύρα RS-232 είναι συσκευή DTE.

Για τη διασύνδεση δύο τερματικών σταθμών σε μεγάλες αποστάσεις μέσω σειριακής θύρας παρεμβάλλονται συσκευές επικοινωνίας, όπως εικονίζεται στο Σχήμα 7.2. Τυπικές τέτοιες συσκευές είναι τα modem, που αναφέρονται ως *συσκευές DCE*.

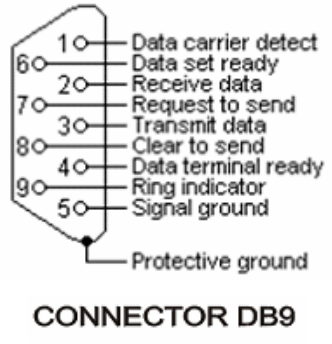


Σχήμα 7.2 Τυπικές διασυνδέσεις DTE/DCE και DCE/DCE

Εκτός από τα modem, στη σειριακή θύρα ενός υπολογιστή μπορεί να συνδεθούν κι άλλες συσκευές που δεν προορίζονται για επικοινωνία, αλλά για μετρήσεις, έλεγχο κυκλωμάτων κ.λπ. Πολλά πολύμετρα, πηγές, ελεγκτές (controllers), αισθητήρες, ακόμη και οικιακές ηλεκτρονικές συσκευές όπως video, δορυφορικοί δέκτες κλπ. έρχονται εξοπλισμένα με σειριακή θύρα RS-232. Οι συσκευές αυτές συνήθως είναι διαμορφωμένες σαν DCE και το καλώδιο που τις συνδέει με τον υπολογιστή καθώς και τα σήματα ελέγχου που ανταλλάσσουν μαζί του είναι ίδια με αυτά ενός modem.

7.5 Ακροδέκτες της σειριακής θύρας

Ανάμεσα στους ακροδέκτες της σειριακής θύρας διακρίνουμε τους *ακροδέκτες* ή *γραμμές δεδομένων* (*data lines*) και τους *ακροδέκτες* ή *γραμμές ελέγχου* (*control lines*). Οι σπουδαιότεροι ακροδέκτες είναι αυτοί που μεταφέρουν τα δεδομένα εκπομπής και λήψης, δηλαδή την πληροφορία προς την μία ή την άλλη κατεύθυνση. Όλοι οι υπόλοιποι ακροδέκτες είναι βοηθητικοί αλλά απαραίτητοι για την επικοινωνία ενός DTE με ένα DCE. Ο σύνδεσμος της σειριακής θύρας είναι εννέα ακροδεκτών, τύπου DB-9.



Σχ. 7.3 Σύνδεσμος DB-9 για τη σύνδεση της σειριακής θύρας

Οι γραμμές δεδομένων ονομάζονται TXD, RXD και SGND, και αντιστοιχούν στους ακροδέκτες 2 και 3, (Πίνακας 7-2). Η γραμμή TXD προορίζεται για την εκπομπή των σειριακών δεδομένων, η γραμμή RXD για τη λήψη των δεδομένων, ενώ η γραμμή SGND είναι ο αγωγός αναφοράς των τάσεων που διαδίδονται στις πρώτες δύο γραμμές. Το σχήμα 7.3 παρουσιάζει τη μορφή και τους ακροδέκτες του τυπικού συνδέσμου.

Οι σπουδαιότερες από τις γραμμές ελέγχου είναι οι RTS, CTS, DSR και DTR. Όταν ο υπολογιστής (DTE) θέλει να στείλει δεδομένα σε μια συσκευή DCE, ενεργοποιεί τη γραμμή RTS (*Request To Send*) θέτοντας στη γραμμή τη συνθήκη SPACE. Εάν η συσκευή DCE έχει χώρο για να δεχθεί δεδομένα, τότε απαντά ενεργοποιώντας τη γραμμή CTS (*Clear To Send*).

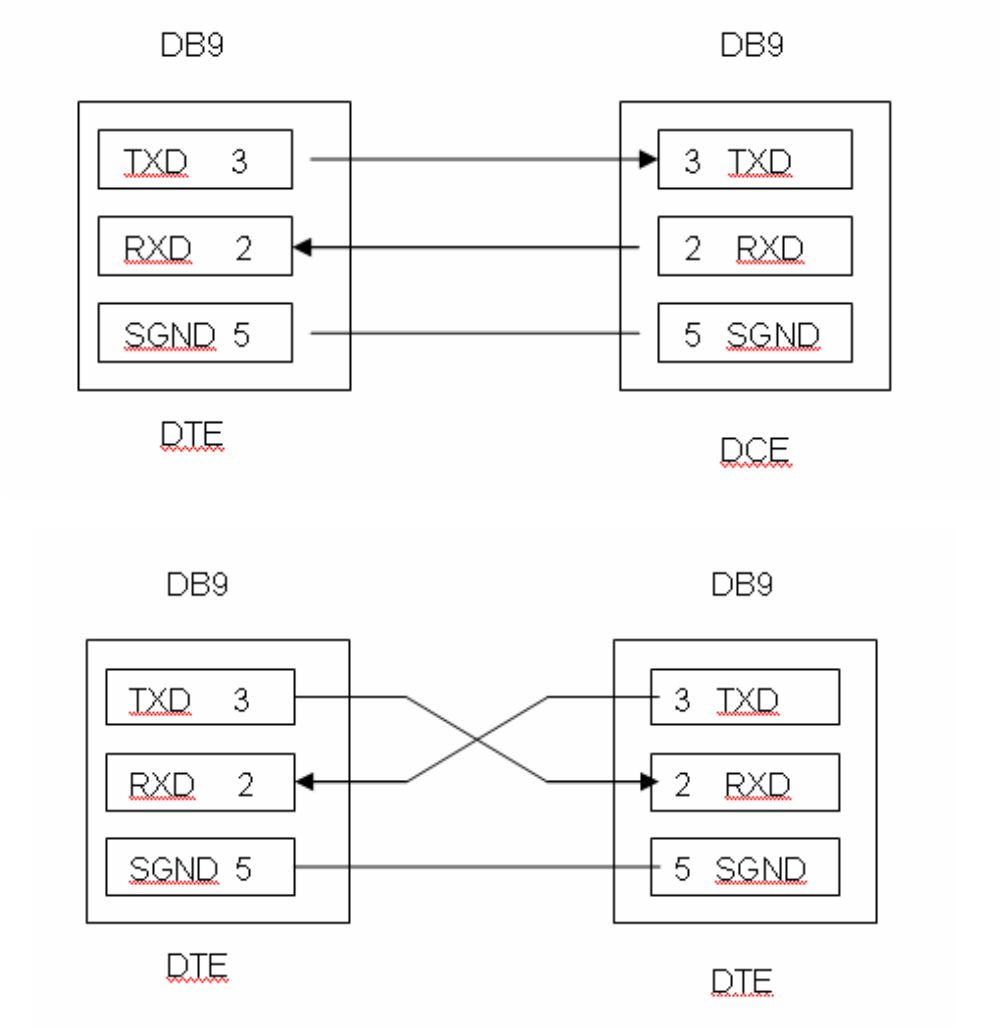
Πίνακας 7-2
Ονομασία και αντιστοιχία των ακροδεκτών της σειριακής θύρας

Ακροδέκτες σε σύνδεσμο D-9	Κωδική ονομασία	Όνομα
3	TXD	Transmit Data
2	RXD	Receive Data
5	SGND	Signal Ground
7	RTS	Request To Send
8	CTS	Clear To Send
6	DSR	Data Set Ready
4	DTR	Data Terminal Ready
1	CD	Carrier Detect
9	RI	Ring Indicator

Αντίστοιχα, όταν το modem θέλει να στείλει δεδομένα, πληροφορεί τη θύρα UART ενεργοποιώντας τη γραμμή DSR (*Data Set Ready*). Εάν ο υπολογιστής από την πλευρά του είναι έτοιμος να λάβει δεδομένα, απαντά θέτοντας την κατάσταση SPACE στον ακροδέκτη DTR (*Data Terminal Ready*). Εάν, πάλι, δεν είναι έτοιμος να λάβει δεδομένα, τότε θέτει συνθήκη MARK στη γραμμή DTR.

Από τους παραπάνω ακροδέκτες, οι απαραίτητοι είναι αυτοί που αντιστοιχούν σε γραμμές επικοινωνίας (2,3,5). Οι γραμμές ελέγχου χρησιμοποιούνται προαιρετικά, μόνον αν επιθυμούμε να υλοποιήσουμε το αντίστοιχο πρωτόκολλο.

Τυπική σύνδεση ανάμεσα σε τερματική συσκευή DTE και σε συσκευή επικοινωνίας DCE φαίνεται στο σχήμα 7.4. Στο ίδιο σχήμα φαίνεται και η σύνδεση ανάμεσα σε τερματικές συσκευές (DTE/DTE), για παράδειγμα ανάμεσα σε δύο προσωπικούς υπολογιστές ή ανάμεσα σε δύο μικροελεγκτές, που επικοινωνούν μέσω σειριακής θύρας.



Σχ. 7.4 Τυπικές συνδέσεις DTE/DCE και DTE/DTE

7.4 Μονάδα ασύγχρονης επικοινωνίας στον PIC16F877

Η Σύγχρονη-Ασύγχρονη μετάδοση (USART) είναι μία από τις δύο σειριακές μονάδες εισόδου – εξόδου που διαθέτει ο μικροελεγκτής PIC16F877. Αυτή μπορεί να διαμορφωθεί ως ένα ασύγχρονο σύστημα το οποίο μπορεί να επικοινωνεί με περιφερειακές συσκευές, όπως LCD displays και προσωπικούς υπολογιστές ή μπορεί να διαμορφωθεί ως ένα σύγχρονο σύστημα το οποίο μπορεί να επικοινωνεί με κυκλώματα όπως A/D ή D/A, μνήμες EEPROM και άλλα. Για τη σειριακή επικοινωνία ο PIC16F877 χρησιμοποιεί τις γραμμές TXD και RXD και SGND του πίνακα 7.2. Οι ακροδέκτες αυτοί είναι οι 25, 26 και 31 αντίστοιχα (βλέπε σχ. 2.5, κεφ. 2). Οι δύο πρώτοι αντιστοιχούν στους ακροδέκτες RC6 και RC7 της θύρας C.

Όλοι οι ακροδέκτες του κυκλώματος UART είναι συμβατοί με τα επίπεδα TTL. Άρα, ανάμεσα στο UART και τον D-τύπου συνδετήρα της σειριακής θύρας παρεμβάλλονται μετατροπείς στάθμης, ώστε τα σήματα του UART να γίνουν συμβατά με τα επίπεδα της λογικής του πρωτοκόλλου RS-232. Οι περισσότεροι Η/Υ χρησιμοποιούν για το σκοπό αυτό τα ολοκληρωμένα κυκλώματα DS1489 για τη λήψη και DS1488 για την εκπομπή σημάτων. Στην παράγραφο 7.9 θα γίνει αναφορά στον μετατροπέα στάθμης MAX232.

Κάθε κύκλωμα UART περιέχει ένα κύκλωμα χρονισμού, που ονομάζεται *γεννήτρια ρυθμού* (*baud rate generator*). Για τη λειτουργία του κυκλώματος χρονισμού απαιτείται ένας εξωτερικός κρύσταλλος, που στην περίπτωση του PIC είναι ο εξωτερικός κρύσταλλος χρονισμού. Η συχνότητα του κρυστάλλου μετατρέπεται εσωτερικά σε ρυθμό εκπομπής και λήψης (baud rate), με τη βοήθεια ενός προγραμματιζόμενου διαιρέτη συχνότητας (baud reate generator).

Ο τρόπος λειτουργίας του κυκλώματος UART προγραμματίζεται με τη βοήθεια ορισμένων καταχωρητών, τους οποίους ο υπολογιστής βλέπει σαν θέσεις μνήμης. Με τη βοήθεια των καταχωρητών αυτών μπορεί ο χρήστης να έχει πλήρη έλεγχο της διαδικασίας εισόδου/εξόδου μέσω της σειριακής θύρας.

Η Ασύγχρονη σειριακή θύρα αποτελείται από τα ακόλουθα σημαντικά κυκλώματα:

- Γεννήτορας ρυθμού μετάδοσης.
- Το κύκλωμα.
- Ασύγχρονος μεταδότης.
- Ασύγχρονος λήπτης.

7.5 Καταχωρητές της ασύγχρονης σειριακής θύρας

Για να επικοινωνήσει ο PIC μέσω της σειριακής θύρας, με περιφερειακές συσκευές ή υπολογιστές χρησιμοποιεί τους εξής καταχωρητές ειδικού σκοπού:

- **TXSTA**: Transmit Status and Control Register- Καταχωρητής κατάστασης και ελέγχου της αποστολής των δεδομένων.
- **RCSTA**: Receive Status and Control Register- Καταχωρητής κατάστασης και ελέγχου της λήψης των δεδομένων.

- **SPBRG**: Baud Rate Generation Register – Καταχωρητής παραγωγής του ρυθμού μετάδοσης baud rate.
- **TXREG**: Καταχωρητής αποστολής δεδομένων.
- **RCREG**: Καταχωρητής λήψης δεδομένων.

Οι πρώτοι τρεις καταχωρητές ρυθμίζουν την λειτουργία της μονάδας USART, ενώ τους άλλους δύο τους χρησιμοποιούμε ως καταχωρητές αποθήκευσης, για να μεταφέρουμε δεδομένα από και προς την USART.

7.6 Αρχικοποίηση των καταχωρητών της σειριακής θύρας

Για την ενεργοποίηση της σειριακής θύρας, το bit SPEN (RCSTA<7>) πρέπει να τεθεί σε λογικό '1'.
Για την ενεργοποίηση της ασύγχρονης επικοινωνίας το bit SYNC του καταχωρητή TXSTA (TXSTA<4>) τίθεται σε λογικό 1.
Τα bits TRISC < 7:6 > πρέπει να τεθούν σε κατάσταση "10", για να διαμορφωθούν τα pins RC6/TX/CX και RC7/RX/DT της USART ως σειριακή έξοδος και είσοδος αντίστοιχα. Το πρώτο (RC6) εκπέμπει δεδομένα, ενώ το δεύτερο (RC7) λαμβάνει δεδομένα.
Για τον καθορισμό του baud rate εγγράφουμε τον καταχωρητή SPBRG με μια τιμή που προκύπτει από τις παρακάτω σχέσεις. Σημειώνεται ότι σε περίπτωση που επιθυμούμε χαμηλούς ρυθμούς μετάδοσης, το bit BRGH (TXSTA<2>) είναι μηδέν, οπότε χρησιμοποιούμε την πρώτη σχέση, ενώ αν επιθυμούμε υψηλούς ρυθμούς μετάδοσης το bit αυτό τίθεται σε λογικό 1 και η τιμή SPBRG προκύπτει από τη δεύτερη σχέση:

$$\text{For } \mathbf{BRGH} = 0 \qquad \text{Baud rate} = \frac{f_{\text{osc}}}{64([\mathbf{SPBRG}] + 1)}$$

$$\text{For } \mathbf{BRGH} = 1 \qquad \text{Baud rate} = \frac{f_{\text{osc}}}{16([\mathbf{SPBRG}] + 1)}$$

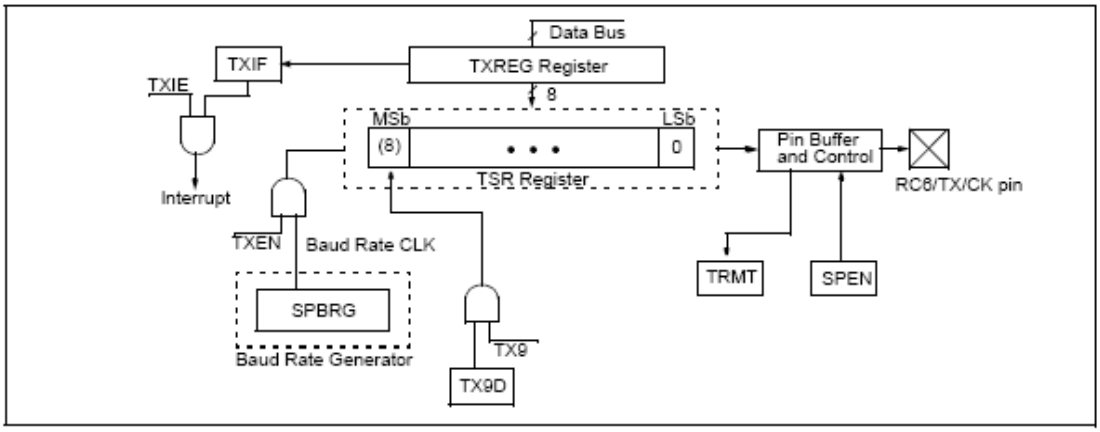
Η Ασύγχρονη επικοινωνία χρησιμοποιεί τη μορφή μετάδοσης που περιγράψαμε στην παράγραφο 7.1: ένα bit START, οχτώ ή εννιά bit δεδομένων και ένα STOP bit. Η USART στέλνει και λαμβάνει το LSB (Least Significant Bit) πρώτα. Η αποστολή και η λήψη είναι λειτουργικά ανεξάρτητες, όμως χρησιμοποιούν την ίδια μορφή δεδομένων και τον ίδιο ρυθμό μετάδοσης.

7.7 Ασύγχρονη Αποστολή

Ο πυρήνας της μετάδοσης είναι ένας εσωτερικός καταχωρητής, ο καταχωρητής TSR. Ο TSR καταχωρητής περιέχει τα δεδομένα του buffer μετάδοσης TXREG (όπου ο TXREG περιέχει τα δεδομένα αποστολής που φορτώνουμε εμείς από το Software). Ο TSR δεν φορτώνεται με δεδομένα έως ότου έρθει το STOP bit από την προηγούμενη φόρτωση. Μόλις έρθει το STOP bit τα δεδομένα (εάν υπάρχουν) μεταφέρονται από τον TXREG

στον TSR. Μόλις ο TXREG στείλει τα δεδομένα στον TSR, υψώνεται σημαία ότι ο TXREG είναι άδειος. Αυτό δηλώνεται από το bit TXIF (PIR1<4>) με την ένδειξη '1'. Αυτή η διακοπή μπορεί να ενεργοποιηθεί θέτοντας '1' το bit TXIE (PIE1<4>). Αν το TXIE είναι '0' το TXIF θα γίνει '1',αλλά δεν θα παραχθεί σήμα διακοπής. Ο TXIF γίνεται '0' μόλις φορτωθούν καινούργια δεδομένα στον TXREG.

Η μετάδοση μπορεί να ενεργοποιηθεί θέτοντας '1' το bit TXEN (TXSTA<5>). Η μετάδοση στην πραγματικότητα δεν θα πραγματοποιηθεί μέχρι ο TXREG να φορτωθεί με δεδομένα και ο Γεννήτορας Ρυθμού Μετάδοσης να παράγει τον χρονισμό.



Σχήμα 7.5: Διάγραμμα Ασύγχρονης Αποστολής Δεδομένων

Πίνακας 7.3: Καταχωρητές που σχετίζονται με την Ασύγχρονη Αποστολή

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Για να καθορίσουμε μια Ασύγχρονη Αποστολή ακολουθούμε τα εξής βήματα:

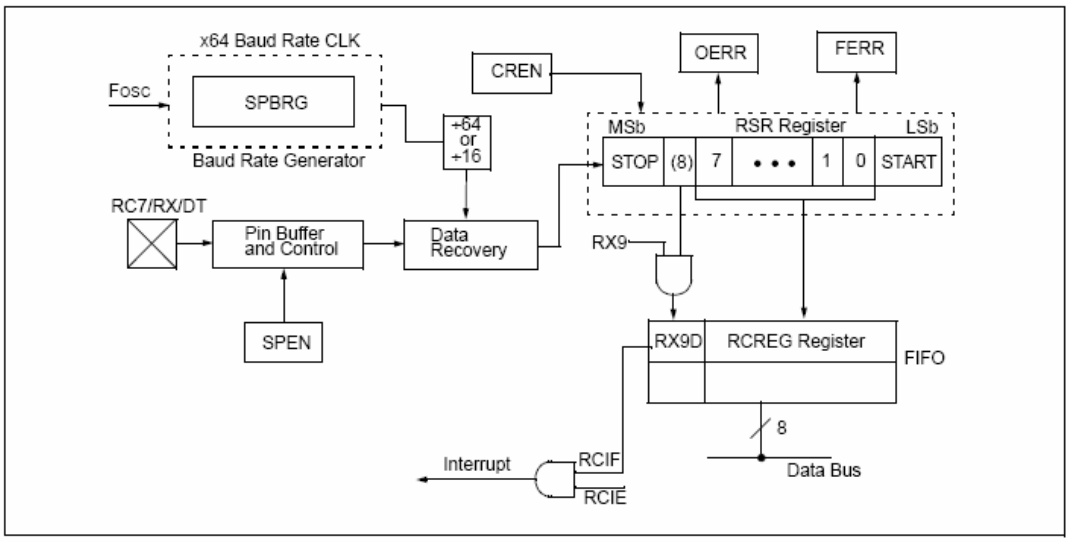
1. Αρχικοποιούμε τον καταχωρητή SPBRG με το κατάλληλο ρυθμό μετάδοσης. Εάν επιθυμούμε υψηλούς ρυθμούς μετάδοσης, θέτουμε '1' το bit BRGH
2. Ενεργοποιούμε την Ασύγχρονη Σειριακή πόρτα θέτοντας '0' το bit SYNC (TXSTA<4>) και '1' το bit SPEN (RCSTA<7>)

3. Εάν επιθυμούμε διακοπές, τότε ενεργοποιούμε το bit TXIE (PIE1<4>)
4. Ενεργοποιούμε την μετάδοση θέτοντας '1' το bit TXEN (TXSTA<5>), το οποίο θα θέσει με την σειρά του '1' το bit TXIF (PIR1<4>)
5. Φορτώνουμε στον TXREG τα δεδομένα (ξεκινάει η μετάδοση)
6. Εάν χρησιμοποιούμε διακοπές, σιγουρευόμαστε ότι τα bits (6 και 7) του καταχωρητή INTCON είναι '1'

Στο Σχήμα 7.5 παρουσιάζεται το διάγραμμα της ασύγχρονης αποστολής των δεδομένων της σειριακής επικοινωνίας.

7.8 Ασύγχρονη Λήψη

Αντίστοιχα με όσα συμβαίνουν στην ασύγχρονη σειριακή εκπομπή συμβαίνουν και κατά την σειριακή λήψη. Τα δεδομένα λαμβάνονται από το pin RC7/RX/DT και μεταφέρονται στον Data Recovery. Ο πυρήνας της λήψης είναι ο εσωτερικός καταχωρητής RSR. Μετά τη λήψη του STOP bit τα δεδομένα λήψης τα οποία βρίσκονται στον RSR, μεταφέρονται στον καταχωρητή RCREG (αν αυτός είναι άδειος). Αν η μεταφορά ολοκληρωθεί η σημαία του καταχωρητή RCIF (PIR1<5>), γίνεται '1'. Η πραγματική διακοπή μπορεί να ενεργοποιηθεί θέτοντας '1' το bit RCIE, του καταχωρητή PIE<5>.
Η διαδικασία της ασύγχρονης σειριακής λήψης των δεδομένων παρουσιάζεται στο Σχήμα 7.6.



Σχήμα 7.6: Διάγραμμα Ασύγχρονης Λήψης Δεδομένων

Για να καθορίσουμε μια Ασύγχρονη Λήψη ακολουθούμε τα εξής βήματα:

1.Φόρτωση του SPBRG για το κατάλληλο ρυθμό μετάδοσης. Αν θέλουμε υψηλό ρυθμό μετάδοσης τότε το bit BRGH πρέπει να τεθεί '1'

2. Ενεργοποιούμε την ασύγχρονη σειριακή πόρτα με το μηδενισμό του SYNC bit και θέτοντας ‘1’ το bit SPEN

3. Αν η διακοπή είναι επιθυμητή θέτουμε ‘1’ το bit RCIE

4.Ενεργοποιούμε την λήψη θέτοντας ‘1’ το bit CREN

5. Η σημαία του bit RCIF θα είναι ‘1’ όταν η λήψη είναι ολοκληρωμένη και μια διακοπή θα λάβει χώρα εφόσον το RCIE bit θα είναι ‘1’

6. Διαβάζουμε τα 8-bit δεδομένων που λήφθηκαν διαβάζοντας τον καταχωρητή RCREG

7. Εάν οποιοδήποτε σφάλμα λάβει χώρα, μηδενίζουμε το σφάλμα, μηδενίζοντας το bit CREN

8. Αν χρησιμοποιούμε διακοπή σιγουρευόμαστε ότι το GIE και το PEIE (7^{o} και 6^{o} bit αντίστοιχα) του καταχωρητή INTCON είναι ‘1’

Πίνακας 7.4: Καταχωρητές που σχετίζονται με την Ασύγχρονη Λήψη

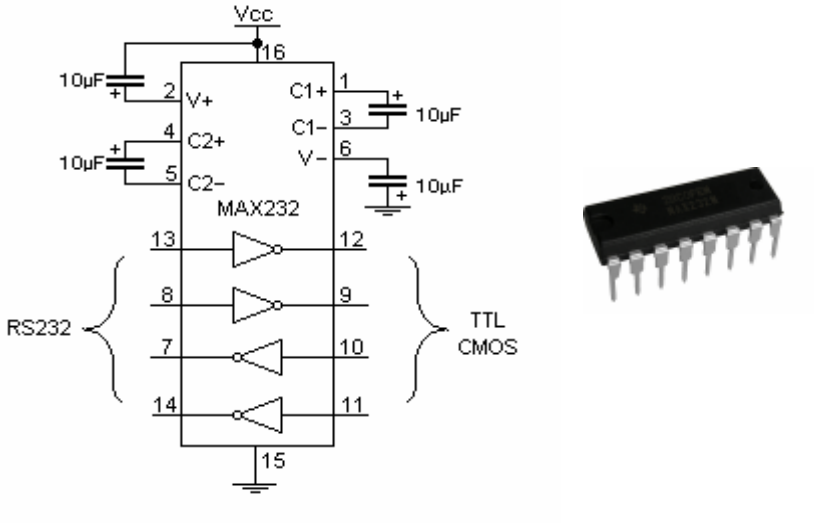
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

7.9 Το κύκλωμα MAX232

Τα διάφορα ψηφιακά ολοκληρωμένα κυκλώματα που χρησιμοποιούμε στις κατασκευές μας δέχονται στάθμες TTL ή CMOS, στις οποίες συνήθως το λογικό ‘0’ αντιστοιχεί σε 0 Volts και το λογικό ΄1΄σε 5 Volts. Για να διασυνδέσουμε επομένως κάποια εφαρμογή με τη σειριακή θύρα, θα πρέπει να παρεμβάλουμε κάποια κυκλώματα που θα μετατρέπουν τις στάθμες του πρωτοκόλλου RS-232 (Πίνακας 7.1) στις παραπάνω απλές στάθμες TTL.

Ένα κύκλωμα που χρησιμοποιείται ευρύτατα για τέτοιο σκοπό είναι το ολοκληρωμένο κύκλωμα MAX232 και τα συμβατά με αυτό. Το κύκλωμα αυτό περιέχει δύο γραμμές εκπομπής και δύο γραμμές λήψης. Λειτουργεί με μια απλή τροφοδοσία +5V και παράγει τα -10 και +10 Volts που απαιτούνται για τις στάθμες της θύρας RS-232 με τη βοήθεια μιας αντλίας φορτίσεως.

Άλλοι τυπικοί μετατροπείς στάθμης για το πρωτόκολλο RS-232 είναι τα ολοκληρωμένα κυκλώματα DS1488 και DS1489. Το πρώτο εκτελεί εκπομπή δεδομένων προς τη θύρα RS-232 και το δεύτερο εκτελεί λήψη δεδομένων από την θύρα RS-232. Κάθε κύκλωμα περιέχει τέσσερις αντιστροφείς ενός μόνο τύπου, δηλαδή εκπομπούς ή

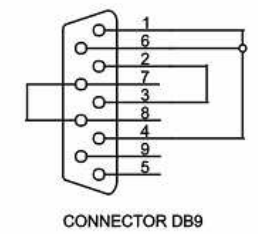


Σχήμα 7.7: Τυπική εφαρμογή του ολοκληρωμένου κυκλώματος MAX232

δέκτες. Στο Σχήμα 7.7 φαίνεται η τυπική εφαρμογή του ολοκληρωμένου κυκλώματος MAX232.

7.10 Παράδειγμα κώδικα για την ασύγχρονη σειριακή επικοινωνία

Παρακάτω παρατίθεται παράδειγμα κώδικα που στέλνει έναν χαρακτήρα και στη συνέχεια λαμβάνει έναν χαρακτήρα. Ο πιο πρόσφορος τρόπος για να χρησιμοποιηθεί ο κώδικας αυτούσιος είναι μια εφαρμογή “echo”, όπου η σειριακή θύρα στέλνει χαρακτήρα και αμέσως λαμβάνει πίσω τον χαρακτήρα που μόλις έστειλε. Αυτό μπορεί να γίνει με κατάλληλη επιστροφή του σήματος TX στον ακροδέκτη RX, δηλαδή βραχυκυκλώνοντας του ακροδέκτες 2 και 3 του σειριακού συνδέσμου (σχήμα 7.8).
Ο παρακάτω κώδικας είναι δομημένος με υπορουτίνες. Η υπορουτίνα initialize επιτελεί την αρχικοποίηση των καταχωρητών της σειριακής θύρας. Η υπορουτίνα send στέλνει έναν χαρακτήρα και η υπορουτίνα receive λαμβάνει. Ανάμεσα σε διαδοχικές αποστολές-λήψεις το σύστημα αναμένει την ενεργοποίηση ενός διακόπτη πιέσεως (push-button). Η υπορουτίνα καθυστέρησης delay_ms χρησιμοποιείται για τον αποκλυδωνισμό (debouncing) του διακόπτη.



Σχήμα 7.8 Σύνδεση βρόχου αυτοακρόασης (τάπα) για λειτουργία echo

```
#include "p16f877.inc"
__CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF & _CPD_OFF &
_WRT_ENABLE_ON & _BODEN_ON & _LVP_OFF

msec equ 20h
                Org 0
                call initialize
                movlw 0xAA              ;test pattern
                movwf PORTB
main            btfss PORTC, 0          ;is button pressed?
                goto $-1
                movlw d'35'             ;wait 25ms to debounce
                call delay_ms
                btfsc PORTC, 0          ;is button released?
                goto $-1
                movlw d'35'             ;wait 25ms to debounce
                call delay_ms
                movf PORTD,w
                call send
                call receive
                movwf PORTB
                goto main

;initialize serial port
initialize      bsf STATUS,RP0
                 movlw 0x00
                movwf TRISB             ;PORTB output
                movlw b'10000001'       ;RC7 (RX) input, RC0 input
                 movwf TRISC
                movlw b'00100100'        ;Transmit enable, high speed baud rate
                movwf TXSTA
                movlw d'103'            ;2404 bps
                movwf SPBRG
                bcf STATUS,RP0
                movlw b'10010000' ;port is on, 8-bit transfer, no address detect
                movwf RCSTA
                return

;subroutine for transmit
send            btfss PIR1,TXIF         ;Edw perimenei mexri to TXIF na ginei '1', pou
                                        ;shmainei adeios TXREG
```

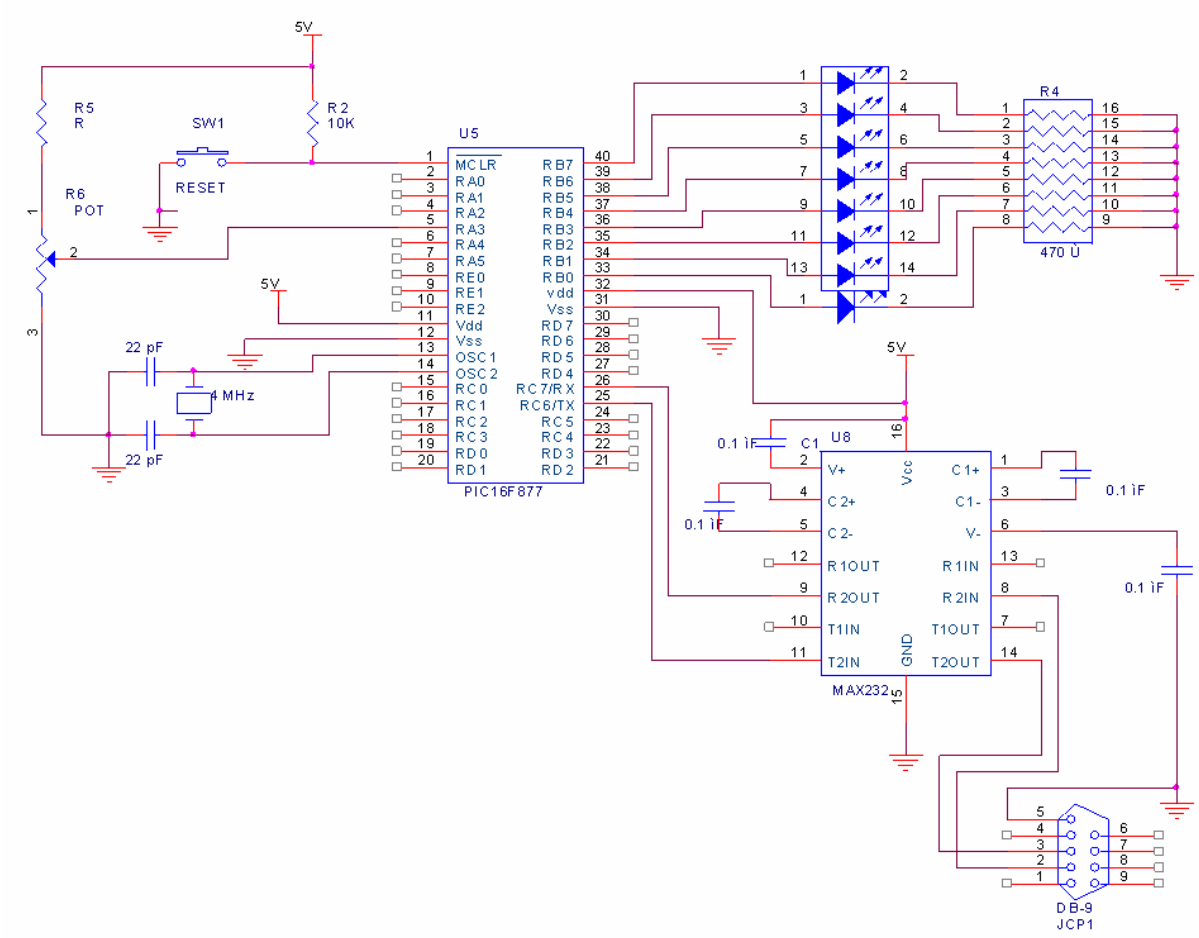
```
                goto $-1                ;wste na fortwsw ta dedomena ston TXREG.
                movwf TXREG
                return

;subroutine for receive
receive         btfss PIR1,RCIF         ;otan ginei 1 exei ftasei xarakthras
                goto receive             ;wait
                movf RCREG,w
                return

;delay subroutine
delay_ms         movwf msec
loop            movlw 0xF8
                call micro4
                nop
                nop
                decfsz msec, f
                goto loop
                return
micro4          addlw 0FF               ;add -1 in 2's complement
                btfss STATUS,Z          ;no skip 1μs. With skip 2μ
                goto micro4
                return

end
```

7.11 Ολοκληρωμένη εφαρμογή συλλογής και αποστολής δεδομένων

Στο σχήμα 7.9 παρουσιάζεται μια ολοκληρωμένη εφαρμογή με βάση τον μικροελεγκτή PIC16F877. Ο μικροελεγκτής συλλέγει τιμές μέσω του μετατροπέα αναλογικού σήματος σε ψηφιακό, διαβάζοντας τις τάσεις στην μεσαία λήψη ενός ποτενσιομέτρου. Στη θέση του ποτενσιομέτρου μπορεί να βρίσκεται ένα αισθητήρας που μετρά κάποιο φυσικό μέγεθος (π.χ. θερμοκρασία, πίεση, φωτεινή στάθμη κλπ.). Στο κύκλωμα του σχήματος 7.9 χρησιμοποιούμε το κανάλι 3 (RA3) του μετροπέα ADC. Στο ίδιο κύκλωμα φαίνονται και οι υπόλοιπες απαραίτητες τυπικές συνδέσεις, όπως του κρυσταλλικού ταλαντωτή και της τροφοδοσίας. Η θύρα B χρησιμοποιείται ως έξοδος και στους ακροδέκτες της συνδέονται LEDs για την απεικόνιση των μετρήσεων. Έτσι, κάθε τιμή που συλλέγει ο ADC μπορεί να απεικονίζεται στη θύρα B. Οι ακροδέκτες 25 και 26 χρησιμοποιούνται ως TX και RX της σειριακής θύρας και συνδέονται με το MAX232 για τη μετατροπή της στάθμης στα επίπεδα που προβλέπει το πρωτόκολλο RS232C. Για να ολοκληρωθεί η εφαρμογή χρειάζεται να συνδεθεί ο συνετήρας DB-9 με τη σειριακή θύρα ενός προσωπικού υπολογιστή ή με έναν μετατροπέα USB-to-serial. Από τη μεριά του ξενιστή

(host) υπολογιστή πρέπει να εκτελείται μια εφαρμογή επικοινωνίας με τη σειριακή θύρα. Για το σκοπό αυτό ο χρήστης μπορεί να χρησιμοποιήσει το περιβάλλον προγραμματισμού Matlab ή το περιβάλλον συλλογής και επεξεργασίας δεδομένων LabVIEW.

Ο ενδιαφερόμενος χρήστης μπορεί να δημιουργήσει τις εφαρμογές του μικροελεγκτή και του ξενιστή υπό μορφή άσκησης, λαμβάνοντας υπόψη όσα αναπτύχθηκαν στο παρόν και στο προηγούμενο κεφάλαιο. Για την ανάπτυξη εφαρμογών σειριακής επικοινωνίας στον ξενιστή μπορεί να ανατρέξει και στο λήμμα 8 της βιβλιογραφίας.



Σχήμα 7.9 Ολοκληρωμένη εφαρμογή για τη λήψη δεδομένων μέσω του μετατροπέα ADC και αποστολή των δεδομένων σε υπολογιστή μέσω της σειριακής θύρας.

8. Έννοιες συστημάτων πραγματικού χρόνου

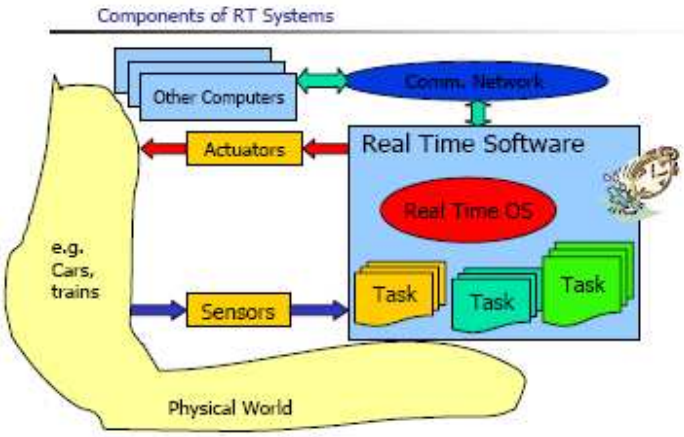
8.1 Τα μέρη ενός συστήματος πραγματικού χρόνου

Στο πρώτο κεφάλαιο δόθηκαν ορισμένα παραδείγματα συστημάτων πραγματικού χρόνου, που μας βοηθούν να κατανοήσουμε τα ιδιαίτερα χαρακτηριστικά τους και τις βασικές διαφορές που εμφανίζουν μεταξύ τους. Στα προηγούμενα κεφάλαια παρουσιάστηκε η αρχιτεκτονική και ο προγραμματισμός ενός μικροελεγκτή, που αποτελεί τυπικό ενσωματωμένο σύστημα. Με βάση τα παραπάνω, θα προσεγγίσουμε τις έννοιες των συστημάτων πραγματικού χρόνου, από τη σκοπιά των μικρών ενσωματωμένων συστημάτων.

Ένα σύστημα πραγματικού χρόνου αποτελείται γενικά από τα παρακάτω μέρη:

- Το υλικό μέρος (hardware), όπου διακρίνουμε τις μονάδες επεξεργασίας (CPUs), τις περιφερειακές μονάδες εισόδου/εξόδου και ένα απαραίτητο σύστημα χρονισμού. Οι περιφερειακές μονάδες μπορούν να συνδέονται με ηλεκτρομηχανικούς ενεργοποιητές (ηλεκτρονόμους, μοτέρ κλπ), αποθηκευτικά μέσα, αισθητήρες, κανάλια επικοινωνίας κλπ. Μέρος του υλικού μπορεί να υλοποιείται ως «σύστημα σε τσιπ» (SoC). Όλα αυτά είναι οι πόροι του συστήματος.
- Ένα λειτουργικό σύστημα πραγματικού χρόνου (RTOS) ή απλά έναν πυρήνα πραγματικού χρόνου (Real-Time Kernel), με προβλέψιμη συμπεριφορά και καλά καθορισμένη λειτουργικότητα (βλέπε κεφάλαιο 10).
- Ένα σύνολο από «**διεργασίες**» (tasks) πραγματικού χρόνου. Οι «διεργασίες» είναι τμήματα κώδικα σε συνδυασμό με μηχανισμούς που δημιουργεί το λειτουργικό σύστημα, για να διαχειρίζεται την εκτέλεση των διεργασιών σε μια εφαρμογή. Οι διεργασίες μοιράζονται τους πόρους του συστήματος, επικοινωνούν και συγχρονίζονται μεταξύ τους και με το περιβάλλον.

Στο σχ. 8.1 φαίνονται οι βασικές συνιστώσες ενός συστήματος πραγματικού χρόνου.



Σχ. 8.1 Τα μέρη ενός συστήματος πραγματικού χρόνου

8.2 Χρονικοί περιορισμοί

Όπως είδαμε στο προηγούμενο κεφάλαιο ένα σύστημα πραγματικού χρόνου πρέπει να παράγει αποτελέσματα της λειτουργίας του μέσα σε συγκεκριμένα χρονικά όρια, που αλλιώς τα χαρακτηρίσαμε «παράθυρο ευκαιρίας» για τη σωστή λειτουργία. Αν το σύστημα είναι περιοδικό, περιέχει δηλαδή επαναλαμβανόμενες διεργασίες, τότε η περίοδος του βρόγχου επανάληψης αντιπροσωπεύει μια μορφή χρονικού περιορισμού.

Ας υποθέσουμε ότι ένα βιομηχανικό σύστημα έχει σκοπό να συλλέγει δεδομένα της θερμοκρασίας τριών φούρνων, από ισάριθμους αισθητήρες θερμοκρασίας (π.χ. θερμοζεύγη) και κατόπιν να ρυθμίζει την θερμοκρασία των φούρνων με τη βοήθεια θερμαντικών αντιστάσεων, σύμφωνα με έναν νόμο ελέγχου. Ο έλεγχος μπορεί να γίνεται περιοδικά. Έστω ότι θέτουμε τον περιορισμό ο έλεγχος και η ρύθμιση για κάθε φούρνο να γίνεται τέσσερις φορές το δευτερόλεπτο. Αυτό αποτελεί μια «διεργασία» (task) πραγματικού χρόνου. Την χρονική στιγμή 0 ξεκινά η εκτέλεση της διεργασίας, και μέσα στα πρώτα 20 ms το σύστημα κάνει αρχικοποίηση (initialization). Σε 20 ms από την αρχή μπορεί να λάβει δεδομένα από τα τρία θερμοζεύγη και έχει στη διάθεσή του 250 ms για να ρυθμίσει όλους τους φούρνους. Αυτό χαρακτηρίζεται ως το πρώτο έργο (job) ή στιγμιότυπο (instance) της διεργασίας. Κατόπιν μέσα στα επόμενα 250 ms επαναλαμβάνει τον βρόγχο ελέγχου κ.ο.κ. Προφανώς, η αρχή κάθε διαδικασίας ελέγχου είναι $20+k\times250$ ms, όπου k=0,1,…

Οι παραπάνω χρόνοι κατά τους οποίους μπορεί να αρχίσει να εκτελείται ο βρόγχος ελέγχου, καλούνται «**χρόνοι αποδέσμευσης**» (release times) του έργου. Στο παράδειγμά μας οι χρόνοι αποδέσμευσης είναι 20, 270, 520,… ms.

Ανάλογα με την φύση της εφαρμογής, υπάρχει μια «**προθεσμία**» (deadline) από τη στιγμή που αποδεσμεύεται ένα έργο, μέσα στην οποία αυτό πρέπει να ολοκληρωθεί. Στο παραπάνω παράδειγμα, το τέλος της προθεσμίας συμπίπτει με τον χρόνο αποδέσμευσης του επόμενου έργου, οπότε οι προθεσμίες είναι διαδοχικά 270 ms, 520 ms, 770 ms κ.λπ. Όμως, θα μπορούσαμε να ορίσουμε σαν περιορισμό ότι ο έλεγχος των φούρνων πρέπει να ολοκληρώνεται νωρίτερα, ώστε να μας μένει ένα περιθώριο ασφαλείας. Σ' αυτή την περίπτωση, οι προθεσμίες θα μπορούσε να είναι 170 ms, 420 ms, 670 ms κ.ο.κ. Τότε, θα είχαμε στη διάθεσή μας 100 ms αναμονής, μέχρι την έναρξη του επόμενου κύκλου.

Στα παραπάνω, οι χρονικοί περιορισμοί ορίζονται σε μια απόλυτη κλίμακα χρόνου, με βάση την αρχή εκτέλεσης της διεργασίας. Συχνά, προτιμούμε να αναφερόμαστε στον «χρόνο απόκρισης» (response time) του κάθε έργου, ο οποίος είναι ο χρόνος από την αποδέσμευση του έργου μέχρι την ολοκλήρωσή του. Η μέγιστος επιτρεπόμενος χρόνος απόκρισης του έργου αποτελεί την «σχετική προθεσμία» του έργου. Στο παραπάνω παράδειγμα, κατά την πρώτη εκδοχή η σχετική προθεσμία είναι 250 ms, ενώ κατά την δεύτερη και αυστηρότερη εκδοχή η σχετική προθεσμία είναι 150 ms.

Το χρονικό διάστημα ανάμεσα σε διαδοχικές δειγματοληψίες (ανάμεσα στην έναρξη διαδοχικών έργων) είναι προφανώς 250ms. Αυτό το διάστημα αποτελεί και την **περίοδο** T της διεργασίας.

Στο παράδειγμα με την μηχανή εμφιάλωσης, που είδαμε στο πρώτο κεφάλαιο (παράγραφος 1.1), μπορούμε πάλι να ορίσουμε τους χρονικούς περιορισμούς που προκύπτουν από την φύση της εφαρμογής. Έστω ότι οι φιάλες περνούν κάτω από την μηχανή με ρυθμό πέντε φιάλες το δευτερόλεπτο. Άρα, κάθε 200 ms μια νέα φιάλη εμφανίζεται στην ακτίνα δράσης της μηχανής. Η **περίοδος** της διεργασίας είναι 200ms. Ας υποθέσουμε ότι η μηχανή έχει ένα παράθυρο ευκαιρίας να βάλει το πώμα, μέσα σε 150 ms από την στιγμή της εμφάνισης της φιάλης. Αν αργήσει περισσότερο, η φιάλη θα έχει απομακρυνθεί πολύ. Άρα, η σχετική προθεσμία εδώ είναι 150 ms, ενώ κάθε 200 ms έχουμε έναν νέο χρόνο αποδέσμευσης.

8.3 Αυστηρά και χαλαρά συστήματα πραγματικού χρόνου

Ανάλογα με την φύση της εφαρμογής πραγματικού χρόνου, οι χρονικοί περιορισμοί που επιβάλλονται μπορεί να νοούνται περισσότερο ή λιγότερο αυστηρά. Για παράδειγμα, μπορεί να επιμένουμε ότι σε μια ουρά στο ταμείο μιας τράπεζας ο ρυθμός εξυπηρέτησης πρέπει να είναι περίπου ένας πελάτης κάθε ενάμισι λεπτό, όμως συνήθως αρκεί ο περιορισμός αυτός να τηρείται με βάση κάποιο γενικό στατιστικό μέτρο. Κανείς δεν θα μας αναγκάσει να αποδείξουμε πέραν αμφιβολίας ότι αυτός ο περιορισμός τηρείται πάντα. Αντίθετα, σε ένα σύστημα αερόσακκου, η υποψία ότι η προθεσμία που τέθηκε από τις προδιαγραφές μπορεί να μην ισχύει πάντα, καθιστά το σύστημα άχρηστο.

Όταν μία διεργασία έχει λειτουργικούς χρονικούς περιορισμούς που πρέπει να τηρηθούν ώστε να αποφευχθούν ανεπιθύμητες συνέπειες ή και καταστροφές, τότε αναφερόμαστε σε αυστηρή διεργασία πραγματικού χρόνου (hard real-time task). Όταν ένα σύστημα πραγματικού χρόνου περιλαμβάνει κατά το μεγαλύτερο μέρος του αυστηρές διεργασίες, τότε ονομάζεται αυστηρό σύστημα πραγματικού χρόνου (hard real-time system). Σε τέτοια συστήματα είναι απαραίτητο να μην χαθεί ποτέ καμία προθεσμία.

Ο χρήστης μιας αυστηρής εφαρμογής πραγματικού χρόνου απαιτεί επιπλέον την επαλήθευση της τήρησης των χρονικών περιορισμών. Ο σχεδιασμός τέτοιων συστημάτων πρέπει να γίνεται με τρόπο που να επιτρέπει την επαλήθευση της ακρίβειας στην τήρηση των προθεσμιών.

Τυπικό παράδειγμα αυστηρού συστήματος πραγματικού χρόνου είναι ο έλεγχος των πτητικών μηχανισμών σε ένα αεροσκάφος. Αν τα διάφορα υποσυστήματα δεν ανταποκριθούν εγκαίρως σε νέα συμβάντα, μέσα σε προκαθορισμένες προθεσμίες, τότε οδηγούμαστε σε ένα ασταθές αεροσκάφος, με όλες τις επακόλουθες συνέπειες.

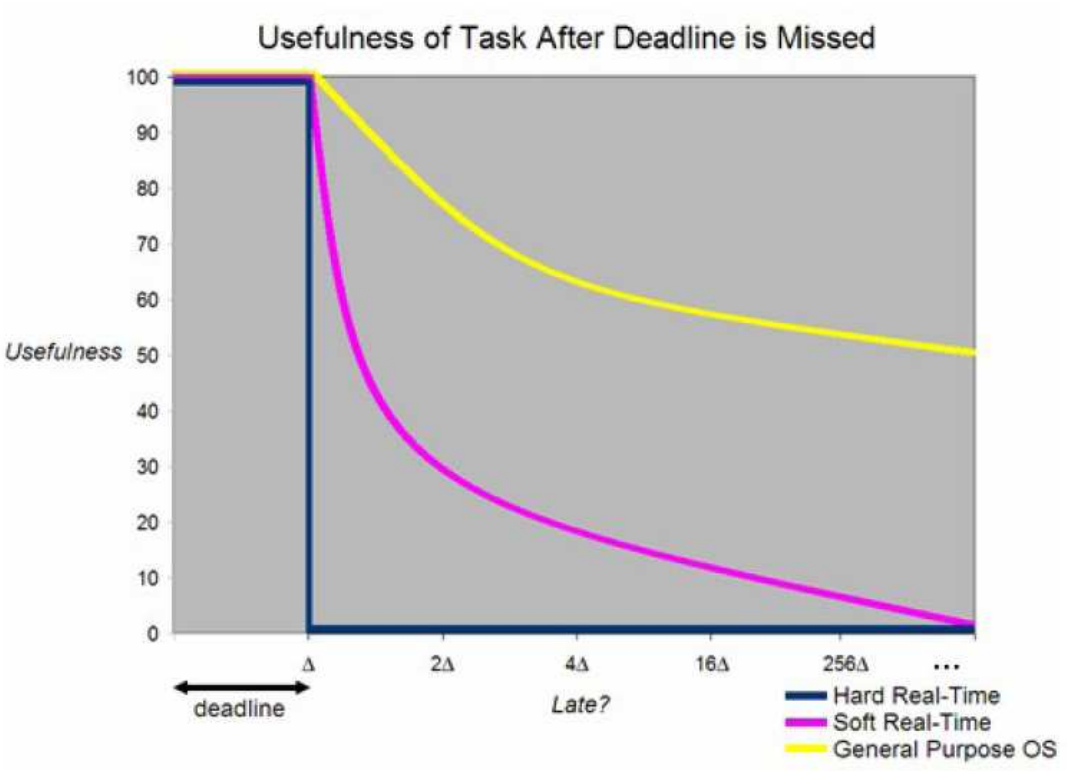
Από την άλλη μεριά, αν η τήρηση των προθεσμιών είναι επιθυμητή, αλλά όχι αναγκαστική, τότε μιλούμε για χαλαρό χρονικό περιορισμό και συνακόλουθα για χαλαρά συστήματα πραγματικού χρόνου (soft real-time systems). Ο αυτόματος έλεγχος της ταχύτητας ενός οχήματος (cruise control) απαιτεί την μέτρηση της ταχύτητας και την επανάληψη του βρόγχου ελέγχου, περίπου κάθε 20ms. Αν το λογισμικό αποτύχει να συλλέξει ένα τρέχον δείγμα της τιμής της ταχύτητας, είναι δυνατό να κάνει τον

υπολογισμό των σημάτων οδήγησης χρησιμοποιώντας την παλιά τιμή, αφού προφανώς η ταχύτητα δεν άλλαξε πολύ από την τελευταία μέτρηση. Από την άλλη μεριά αν το σύστημα απωλέσει αρκετά διαδοχικά δείγματα, τότε παύει πλέον να εκπληρώνει τις προδιαγραφές.

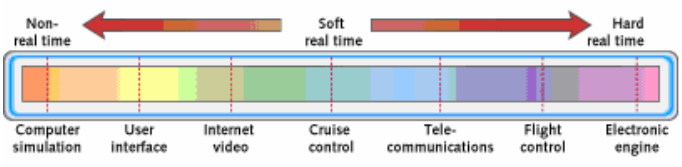
Ένα άλλο παράδειγμα χαλαρού συστήματος πραγματικού χρόνου είναι μια πολυμεσική εφαρμογή ροής βίντεο, όπως ένα σύστημα τηλεδιάσκεψης. Προφανώς, υπάρχει η απαίτηση ώστε το σύστημα να αποδίδει ένα νέο πλαίσιο κάθε τριακοστό του δευτερολέπτου, ενώ η εικόνα και ο ήχος πρέπει να συγχρονίζονται μέσα σε μια προθεσμία 80 ms. Ωστόσο, κάποιες απώλειες στη ροή των πλαισίων ή κάποια απώλεια συγχρονισμού εικόνας και ήχου μπορεί να θεωρηθεί αποδεκτή, μέσα σε κάποια όρια. Φυσικά, όσο περισσότερες προθεσμίες χάνονται, τόσο μειώνεται η χρησιμότητα του συστήματος. Πολλά χαμένα πλαίσια καθιστούν την εφαρμογή άχρηστη.

Με βάση την παραπάνω παρατήρηση είναι δυνατό να περιγράψουμε την έννοια των αυστηρών και χαλαρών συστημάτων πραγματικού χρόνου, με τη βοήθεια μιας συνάρτησης χρησιμότητας (usefulness function). Στο διάγραμμα που ακολουθεί, φαίνεται η χρησιμότητα μιας διεργασίας πραγματικού χρόνου, πέρα από τα όρια της προθεσμίας της. Στα αυστηρά συστήματα η χρησιμότητα μηδενίζεται αμέσως μετά το όριο της προθεσμίας.

Στο Σχ. 8.3 φαίνεται ένα φάσμα εφαρμογών, από τις χαλαρές, που δεν θέτουν πολύ αυστηρούς χρονικούς περιορισμούς, μέχρι και τις πιο αυστηρές.



Σχ. 8.2 Χρησιμότητα μιας διεργασίας, μετά την απώλεια της προθεσμίας



Σχ. 8.3 Φάσμα εφαρμογών πραγματικού χρόνου

Ας σημειωθεί ότι σε ένα σύστημα πραγματικού χρόνου είναι δυνατό να υπάρχουν παράλληλα αυστηρά και χαλαρά υποσυστήματα. Για παράδειγμα, τα υποσυστήματα στα διαφορετικά επίπεδα ιεραρχικού ελέγχου ενός συστήματος πτήσης (βλέπε Κεφ. 1), υπακούουν σε λιγότερο ή περισσότερο αυστηρές προθεσμίες.

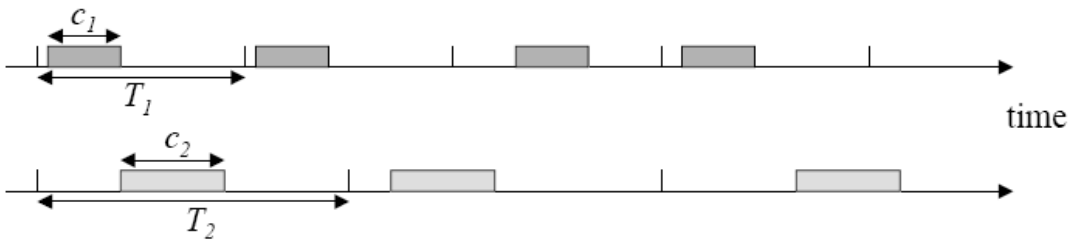
8.4 Άλλες χρονικές παράμετροι διεργασιών πραγματικού χρόνου

Στην παράγραφο 8.3 είδαμε το νόημα των χρονικών περιορισμών που διέπουν τις διεργασίες πραγματικού χρόνου. Κάθε έργο σε μια διεργασία χαρακτηρίζεται από έναν *χρόνο αποδέσμευσης* και μια *προθεσμία* (απόλυτη ή χαλαρή).

Μια άλλη σημαντική χρονική παράμετρος είναι ο **χρόνος εκτέλεσης** c_i (execution time) ενός έργου J_i. Το c_i είναι ο χρόνος που απαιτείται ώστε να ολοκληρωθεί η εκτέλεση του έργου J_i όταν εκτελείται μόνο του και έχει στη διάθεσή του όλους τους πόρους που χρειάζεται. Άρα η παράμετρος αυτή εξαρτάται από την πολυπλοκότητα του έργου και την ταχύτητα του επεξεργαστή.

Ο χρόνος εκτέλεσης μιας διεργασίας δεν είναι σταθερός, καθώς μπορεί (για παράδειγμα) να εξαρτάται από διακλαδώσεις που εκτελούνται υπό συνθήκη. Για τον λόγο αυτό, η παράμετρος που πραγματικά μας ενδιαφέρει είναι ο *μέγιστος χρόνος εκτέλεσης* μιας διαδικασίας (worst case execution time).

Το σχήμα 8.4 παρουσιάζει δύο διεργασίες με περίοδο T_1 και T_2, σε μια χρονογραμμή (timeline). Οι μικρές κατακόρυφες γραμμές αντιστοιχούν στην έναρξη κάθε περιόδου, όπου αποδεσμεύεται κάθε έργο. Θεωρούμε ότι η εκτέλεση κάθε έργου γίνεται σε χρόνο ίσο με τον μέγιστο χρόνο εκτέλεσης. Η απόδοση του έργου γίνεται μέσα στα όρια της σχετικής προθεσμίας που θέτουν οι προδιαγραφές του συστήματος.



Σχ. 8.4 Χρονικές παράμετροι για δύο διαφορετικές περιοδικές διεργασίες

Σε πολλές περιοδικές διεργασίες, όπως στο σχ. 8.4, η προθεσμία D_i ταυτίζεται με την περίοδο του έργου, αφού κάθε έργο αποδεσμεύεται στην αρχή της περιόδου του και πρέπει να έχει ολοκληρωθεί μέχρι το πέρας της περιόδου. Σε ορισμένες άλλες περιπτώσεις, είναι δυνατό η προθεσμία να είναι μικρότερη από την περίοδο, ώστε να υπάρχει χρόνος για συμπληρωματικές ενέργειες, πριν αρχίσει το επόμενο έργο.

Σε ορισμένες περιπτώσεις, μπορεί να αποδεσμεύεται κάποιο έργο, αλλά να απαιτεί κάποιο χρόνο προετοιμασίας μέχρι να αρχίσουν να εκτελούνται οι υπολογισμοί. Συχνά, χρειάζεται να μεταφερθούν δεδομένα στην μνήμη ή από μια μνήμη σε άλλη. Στο σχήμα 8.4 φαίνεται ότι η εκτέλεση κάθε έργου μπορεί να αρχίζει σε διαφορετική στιγμή μέσα στην περίοδο, ανάλογα με τον χρόνο προετοιμασίας.

Τέλος, ας υποθέσουμε ότι έχουμε ένα σύνολο από *n* περιοδικές διεργασίες, που η κάθε μια έχει τη δική της περίοδο T_i, τον δικό της μέγιστο χρόνο εκτέλεσης c_i και την δική της σχετική προθεσμία D_i (i=1,2,…,*n*). Οι διεργασίες αυτές πρέπει να οργανωθούν σε έναν κοινό χρονικό προγραμματισμό, ώστε να εκτελούνται παράλληλα. Ένας τρόπος είναι να εκτελούνται εναλλάξ, μέχρι να ολοκληρωθεί η ακολουθία. Στην παράγραφο 8.6 δίνεται ένα παράδειγμα.

Το ελάχιστο κοινό πολλαπλάσιο των περιόδων T_i συμβολίζεται με H και ονομάζεται υπερπερίοδος (hyperperiod) των περιοδικών διεργασιών. Ο μέγιστος αριθμός έργων που μπορεί να χωρέσει σε μια υπερπερίοδο ισούται με

$$N = \sum_{i=1}^{n} H / T_i \qquad (8.1)$$

Για παράδειγμα, αν έχουμε τρεις περιοδικές διεργασίες με περιόδους 3,4 και 10, η υπερπερίοδος H είναι 60. Ο συνολικός αριθμός N των έργων που μπορούν να εκτελεστούν μέσα στην υπερπερίοδο είναι 41.

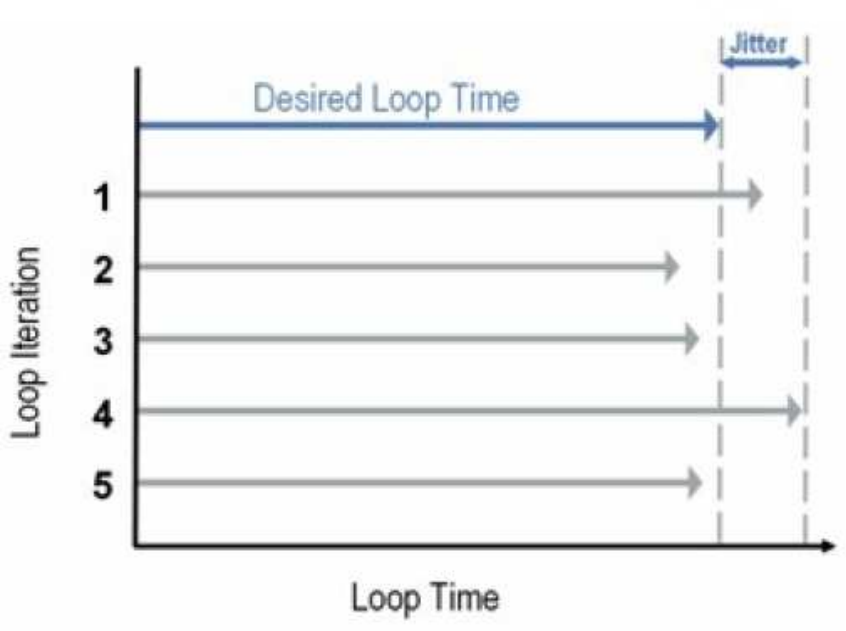
8.5 Χρονικές διαταραχές (Jitter)

Είδαμε ότι ο χρόνος εκτέλεσης μιας διεργασίας μπορεί να μην είναι απολύτως σταθερός, αλλά να μεταβάλλεται μέσα σε κάποια όρια, ανάλογα με τον υπολογιστικό φόρτο σε κάθε περίοδο του έργου.

Σε ορισμένες άλλες περιπτώσεις μπορεί ο χρόνος αποδέσμευσης να μεταβάλλεται μέσα σε κάποια όρια, να εμφανίζει δηλαδή μια διαταραχή, που ονομάζεται jitter.

Τέλος, ο χρόνος προετοιμασίας της εκτέλεσης ενός έργου, αφού αυτό αποδεσμευτεί, είδαμε ότι μπορεί να μεταβάλλεται. Όλες αυτές οι διαταραχές έχουν σαν αποτέλεσμα ο χρόνος ολοκλήρωσης ενός έργου να μην είναι πάντα σταθερός, αλλά να μεταβάλλεται μέσα σε κάποια όρια. Έτσι, ένας επαναλαμβανόμενος βρόγχος δεν ολοκληρώνεται πάντα στον επιθυμητό προκαθορισμένο χρόνο μέσα στην περίοδο, αλλά μέσα σε κάποια όρια σφάλματος, που γενικά χαρακτηρίζεται ως διαταραχή του χρόνου απόδοσης (response time jitter).

Το σχήμα 8.5 δίνει μια εικόνα της διαταραχής αυτής για διαδοχικές επαναλήψεις του βρόγχου ελέγχου.



Σχ. 8.5 Παράδειγμα χρονικής διαταραχής (jitter)

8.6 Χρονικοί περιορισμοί σε ελεγκτή αυτοκινήτου

Ας φανταστούμε ένα σύνθετο σύστημα, για παράδειγμα έναν ελεγκτή αυτοκινήτου. Το σύστημα, αποτελείται από πολλές διεργασίες, όπως για παράδειγμα την έγχυση καυσίμου, τον έλεγχο των φρένων ABS, τον έλεγχο της ταχύτητας και τον έλεγχο λιγότερο αυστηρών συστημάτων, όπως το σύστημα ήχου ή το σύστημα κλιματισμού.

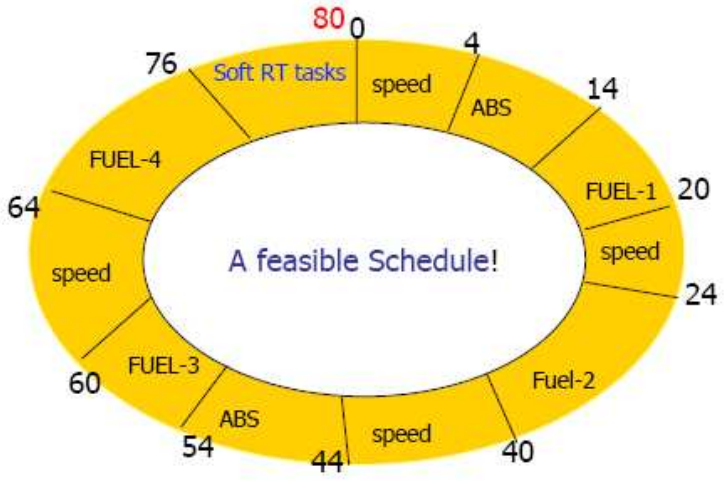
Η διεργασία (task) του ελέγχου της ταχύτητας (cruise control), όταν έχουμε ρυθμίσει σταθερή ταχύτητα π.χ. 80 Km/h, πραγματοποιείται με μια διαδοχική δειγματοληψία τιμών της ταχύτητας, με τη βοήθεια ενός αισθητήρα ταχύτητας. Σε κάθε εκτέλεση του βρόγχου, η απόκλιση από την επιθυμητή ταχύτητα διορθώνεται ελέγχοντας κατάλληλα τις στροφές του κινητήρα. Κάθε διαδοχική δειγματοληψία και διόρθωση θεωρείται ένα επαναλαμβανόμενο έργο, που υπακούει σε σχετική προθεσμία, όπως και στο παράδειγμα με τη ρύθμιση των φούρνων, στην παράγραφο 8.2. Η ελάχιστος χρόνος ανάμεσα σε διαδοχικές δειγματοληψίες είναι η περίοδος Τ της διεργασίας.

Προκειμένου να προγραμματίσουμε τις παραπάνω διεργασίες, ώστε να εκτελούνται σε πραγματικό χρόνο από τον ελεγκτή του αυτοκινήτου, χρειαζόμαστε για την κάθε μία την γνώση των βασικών χρονικών παραμέτρων. Έστω, λοιπόν, ότι μας πληροφορούν ότι ισχύει:

Έλεγχος ταχύτητας: Μέγιστος χρόνος εκτέλεσης e_1=4 ms, Περίοδος T_1=20 ms, και Προθεσμία D_1=5 ms.

Έλεγχος φρένων ABS: e_2=10 ms, T_2=40 ms, D_2=40 ms.

Έγχυση καυσίμου: e_3=40 ms, T_3=80 ms, D_3= 80 ms.



Σχ. 8.6 Αποδοτική χρονοδρομολόγηση διεργασιών του ελεγκτή αυτοκινήτου

Για να επιτελούνται οι παραπάνω διεργασίες μέσα στα όρια των χρονικών τους περιορισμών, χρειάζεται να υπολογιστεί ένα προγραμματισμός των εργασιών (schedule), με βάση τον οποίο κάθε διεργασία θα καλείται, θα εκτελείται και θα επαναλαμβάνεται, χωρίς να παραβιάζει τα όρια των άλλων διεργασιών. Αυτή η **χρονοδρομολόγηση** (scheduling) είναι ένα βασικό μέρος των εργασιών που εκτελεί ένα λειτουργικό σύστημα πραγματικού χρόνου. Αναφορά στα λειτουργικά συστήματα πραγματικού χρόνου, στις ιδιότητες και στις λειτουργίες τους θα γίνει σε επόμενο κεφάλαιο.

Ένα παράδειγμα οργάνωσης της διαδοχικής εκτέλεσης των λειτουργιών ενός ελεγκτή αυτοκινήτου δίνεται στο σχήμα 8.6. Ας σημειωθεί ότι η υπερπερίοδος Η είναι 80 ms.

8.7 Απεριοδικές και σποραδικές διεργασίες

Εκτός από τις περιοδικές διεργασίες, όπως αυτές που περιγράψαμε στα παραπάνω παραδείγματα, ένα σύστημα πραγματικού χρόνου πρέπει να είναι σε θέση να ανταποκριθεί σε εξωτερικά γεγονότα, που συμβαίνουν σε τυχαίες χρονικές στιγμές. Τότε, το σύστημα πρέπει να εκτελέσει ένα σύνολο έργων, των οποίων ο χρόνος αποδέσμευσης δεν είναι εκ των προτέρων γνωστός και επέρχεται όταν συμβεί το εξωτερικό γεγονός. Για παράδειγμα, ο ελεγκτής αυτοκινήτου, που θεωρήσαμε στην προηγούμενη παράγραφο, είναι δυνατό να δεχτεί ένα σήμα που να αποσυνδέει τον αυτόματο έλεγχο της ταχύτητας και να περνά τον έλεγχο στον οδηγό. Το σύστημα ασφαλείας ενός εργοστασίου πρέπει να διακόψει τους περιοδικούς ελέγχους του, για να ανταποκριθεί σε μια επικίνδυνη κατάσταση, που ανιχνεύουν οι ανιχνευτές του. Ένα σύστημα μετρήσεων μπορεί να δεχτεί ένα σήμα, που μεταφέρει τον έλεγχό του από τοπικό επίπεδο σε απομακρυσμένο (remote control). Τέτοιες διεργασίες ονομάζονται

απεριοδικές, αν έχουν χαλαρές προθεσμίες. Αν όμως έχουν απόλυτες προθεσμίες οι διεργασίες ονομάζονται *σποραδικές*.

8.8 Προβλεψιμότητα και καθοριστικότητα

Ένα σύστημα πραγματικού χρόνου χαρακτηρίζεται **προβλέψιμο** (predictable), όταν η χρονική του συμπεριφορά είναι πάντα μέσα σε αποδεκτά όρια. Η συμπεριφορά αυτή αφορά στο σύνολο των διεργασιών που εκτελούνται στο σύστημα, ώστε θα πρέπει να ισχύει ότι όλες οι διεργασίες ολοκληρώνονται μέσα στις προθεσμίες τους. Γενικά, για να δημιουργήσουμε ένα προβλέψιμο σύστημα πραγματικού χρόνου πρέπει να γνωρίζουμε την περίοδο, την προθεσμία και τον μέγιστο χρόνο εκτέλεσης, για κάθε διεργασία του συστήματος. Για τη δημιουργία ενός τέτοιου συστήματος χρειαζόμαστε έναν κατάλληλο αλγόριθμο χρονοπρογραμματισμού (scheduling).

Ένα **καθοριστικό** σύστημα (deterministic system) είναι μια ειδική περίπτωση ενός προβλέψιμου συστήματος. Όχι μόνον η συμπεριφορά του είναι μέσα σε αποδεκτά όρια, αλλά αυτή η χρονική συμπεριφορά είναι προκαθορισμένη. Κάθε διεργασία εκτελείται μόνον σε προκαθορισμένες χρονοθυρίδες (time-slots). Ο χρόνος εκτέλεσης κάθε διεργασίας πρέπει να είναι γνωστός και να μην υπάρχουν ανωμαλίες που μπορεί να προκαλέσουν απόκλιση από την προκαθορισμένη συμπεριφορά. Η χρονικές διαταραχές (jitter) που είδαμε στην παράγραφο 8.5 προκαλούν απόκλιση από την καθοριστικότητα. Έτσι, η δημιουργία ενός καθοριστικού συστήματος πραγματικού χρόνου είναι πολύ δύσκολη υπόθεση. Πάντως, η καθοριστικότητα δεν είναι απαραίτητη προκειμένου το σύστημα να είναι προβλέψιμο.

8.9 Υλοποίηση συστημάτων πραγματικού χρόνου

Προκειμένου να υλοποιήσουμε συστήματα πραγματικού χρόνου, ικανά να αποδίδουν το αποτέλεσμα των διαφόρων διεργασιών μέσα στις προβλεπόμενες προθεσμίες, μπορούμε να ακολουθήσουμε μία από τις παρακάτω προσεγγίσεις.

Η πρώτη είναι ο προγραμματισμός του συστήματος σε χαμηλό επίπεδο, χρησιμοποιώντας γλώσσα Assembly, ή αντίστοιχα κάποιον μεταγλωττιστή C. Ο προγραμματισμός του συστήματος γίνεται με χρήση χρονιστών, διακοπών (interrupts), επαναλαμβανόμενων βρόγχων κλπ. Οι ρουτίνες σε assembly πρέπει να διαχειρίζονται τις θύρες I/O, τους μετατροπείς αναλογικού σήματος σε ψηφιακό και τα κανάλια επικοινωνίας, όπως οι UART, οι θύρες USB, οι δίαυλοι CAN (Controller Area Network) κ.λπ. Μια τέτοια προσέγγιση χαμηλού επιπέδου είναι κατάλληλη για μικρά ενσωματωμένα συστήματα, όπως αυτά που συναντήσαμε στα προηγούμενα κεφάλαια. Στην επόμενη παράγραφο θα αναφερθούμε σε τεχνικές πολυεπεξεργασίας, που επιτρέπουν προγραμματισμό πραγματικού χρόνου, σύμφωνα με αυτή την προσέγγιση.

Μια άλλη προσέγγιση, πιο κατάλληλη για μεγάλα και πολύπλοκα συστήματα, είναι ο προγραμματισμός με χρήση παράλληλων (concurrent) γλωσσών πραγματικού χρόνου. Τέτοιες γλώσσες είναι η Ada 95, η Real-time Java, η RTL/2 και άλλες. Περιβάλλοντα

όπως το LabVIEW και το Matlab διαθέτουν, επίσης, ενσωματωμένες δυνατότητες προγραμματισμού πραγματικού χρόνου. Οι γλώσσες αυτές έχουν το χαρακτηριστικό ότι μπορούν να διαχειρίζονται παραπάνω από μία διεργασία (task), καθώς ένα πρόγραμμα μπορεί να αποτελείται από ένα σύνολο αυτόνομων σειριακών διεργασιών, που η λογική τους εκτέλεση γίνεται παράλληλα. Η δυνατότητα αυτή στηρίζεται στην έννοια των πολλών διεργασιών μέσα σ' ένα πρόγραμμα (multithreads), η οποία δεν υπάρχει στις συμβατικές γλώσσες, όπως η PASCAL, η FORTRAN, η C, που είναι καθαρά σειριακές γλώσσες προγραμματισμού.

Βέβαια, πραγματική παραλληλία στην εκτέλεση των διαφόρων διεργασιών μέσα σ' ένα πρόγραμμα μπορεί να γίνει μόνον σ' ένα σύστημα με πολλούς επεξεργαστές. Σε σύστημα ενός επεξεργαστή, οι διαδικασίες εκτελούνται με την τεχνική της πολυπλεξίας. Γι' αυτό και η έννοια της ταυτόχρονης εκτέλεσης αναφέρεται στη βιβλιογραφία ως «εν δυνάμει» παραλληλισμός (potential parallelism).

Η διαχείριση των διαδικασιών σε μια παράλληλη γλώσσα γίνεται με τη βοήθεια του Run-Time Support System (συστήματος υποστήριξης κατά τον χρόνο εκτέλεσης) ή RTSS. Επίσης, αναφέρεται ως Real-Time Kernel (πυρήνας πραγματικού χρόνου). Το RTSS ουσιαστικά ενεργεί σαν τον χρονοπρογραμματιστή (scheduler) ενός λειτουργικού συστήματος και εδράζεται ανάμεσα στο υλικό και στο λογισμικό της εφαρμογής. Το σύστημα συνήθως παράγεται από το μεταγλωττιστή (compiler) μαζί με τον εκτελέσιμο κώδικα (object code) του προγράμματος. Αυτή η προσέγγιση, τουλάχιστον, ακολουθείται στα προγράμματα της Ada και της Java, που χαρακτηρίζονται ως γλώσσες πραγματικού χρόνου.

Μια τρίτη προσέγγιση είναι να χρησιμοποιήσουμε μια συμβατική σειριακή γλώσσα προγραμματισμού (π.χ. PASCAL, C) κι ένα λειτουργικό σύστημα, που παρέχει υποστήριξη για εφαρμογές πραγματικού χρόνου. Τέτοια λειτουργικά συστήματα αναφέρονται ως λειτουργικά συστήματα πραγματικού χρόνου (Real Time Operating Systems) και παρέχουν ευκολίες για πολυδιεργασία (multitasking) και πολυνημάτωση (multithreading). Στην περίπτωση αυτή οι λειτουργίες του παραλληλισμού περνούν στο λειτουργικό σύστημα και όχι στην γλώσσα προγραμματισμού. Όπως θα δούμε, μικροί πυρήνες RTOS εγκαθίστανται πολλές φορές και σε μικρά ενσωματωμένα συστήματα, καθώς αυτά πλέον έχουν αρκετή μνήμη. Όμως, η δημιουργία ενός RTOS kernel είναι αρκετά εξειδικευμένο έργο και για το λόγο αυτό τις περισσότερες φορές χρησιμοποιούνται εμπορικές λύσεις.

8.10 Τεχνικές πολυδιεργασίας στα ενσωματωμένα συστήματα

Όπως περιγράφηκε στα προηγούμενα κεφάλαια, όπου αναπτύχθηκαν τεχνικές προγραμματισμού ενσωματωμένων συστημάτων, ο συνήθης τρόπος προγραμματισμού ενός μικροελεγκτή είναι η επαναλαμβανόμενη εκτέλεση του βρόχου ελέγχου (control loop). Σε απλές εφαρμογές, όλα τα επιμέρους τμήματα του κώδικα ενσωματώνονται σε μια ενιαία λίστα εντολών, που εκτελούνται σειριακά. Τέτοιες απλές εφαρμογές

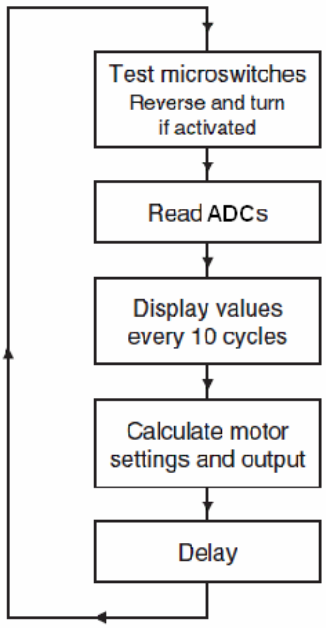
συνιστούν μια μόνο διεργασία. Όταν ο βρόχος ελέγχου εκτελείται αρκετά γρήγορα, τότε από τη σκοπιά του χρήστη εκπληρώνονται οι βασικές απαιτήσεις.

Η παραπάνω τεχνική προγραμματισμού αφήνει ανοιχτά πολλά προβλήματα εκτέλεσης σε πραγματικό χρόνο. Έτσι, η εκτέλεση ενός τμήματος κώδικα μπορεί να γίνεται σε βάρος ενός άλλου, αν το πρώτο έχει ιδιαίτερα μεγάλες υπολογιστικές απαιτήσεις και χρησιμοποιεί διαρκώς τους υπολογιστικούς πόρους.

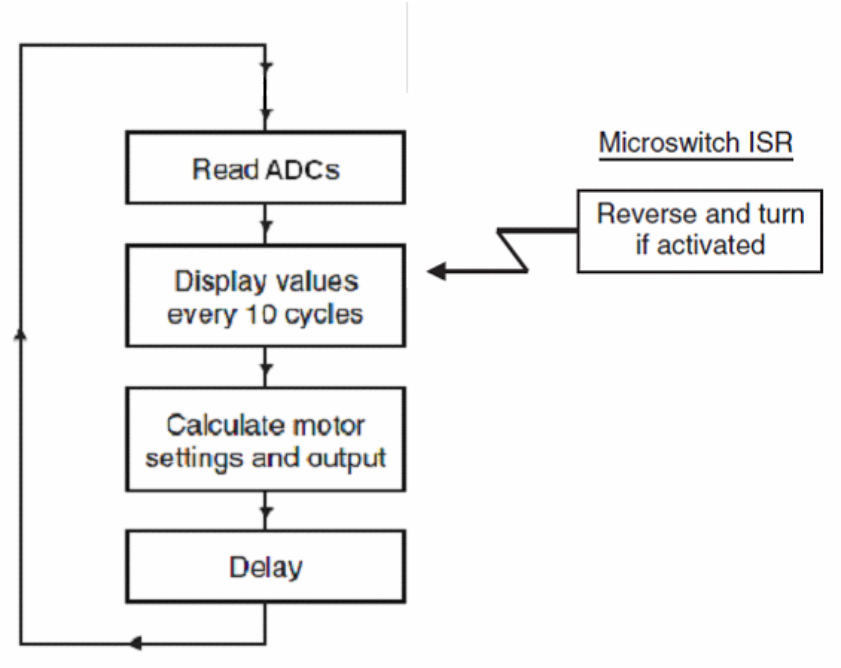
Σε πιο σύνθετες εφαρμογές, το υπολογιστικό έργο χωρίζεται αρχικά σε ένα σύνολο από διακριτές ***διεργασίες***. Με μια απλή έννοια, η διεργασία είναι ένα τμήμα κώδικα, που μπορεί να εκτελεστεί ανεξάρτητα από άλλα τμήματα κώδικα και να παράγει τα δικά του διακριτά αποτελέσματα. Σε ένα μικρό αυτόνομο όχημα, που κινείται στο χώρο αποφεύγοντας τα εμπόδια, μπορούμε να διακρίνουμε τις εξής διεργασίες: α. Συλλογή δεδομένων από τον αισθητήρα απόστασης για τη χαρτογράφηση του χώρου στη γειτονιά του οχήματος β. Αντίδραση του οχήματος σε πιθανές συγκρούσεις με εμπόδια, οι οποίες καταγράφονται μέσω μικροδιακοπτών γ. προβολή μηνυμάτων προς το χρήστη σε οθόνη LCD γ. ρύθμιση των παλμών οδήγησης των κινητήρων του οχήματος.

Η εκτέλεση των διεργασιών πρέπει να γίνει με τρόπο που να επιτρέπει σε όλες να παράγουν το υπολογιστικό έργο τους μέσα στις προβλεπόμενες προθεσμίες. Χαρακτηριστικές τεχνικές που μπορεί να χρησμοποιηθούν για το σκοπό αυτό είναι οι εξής:

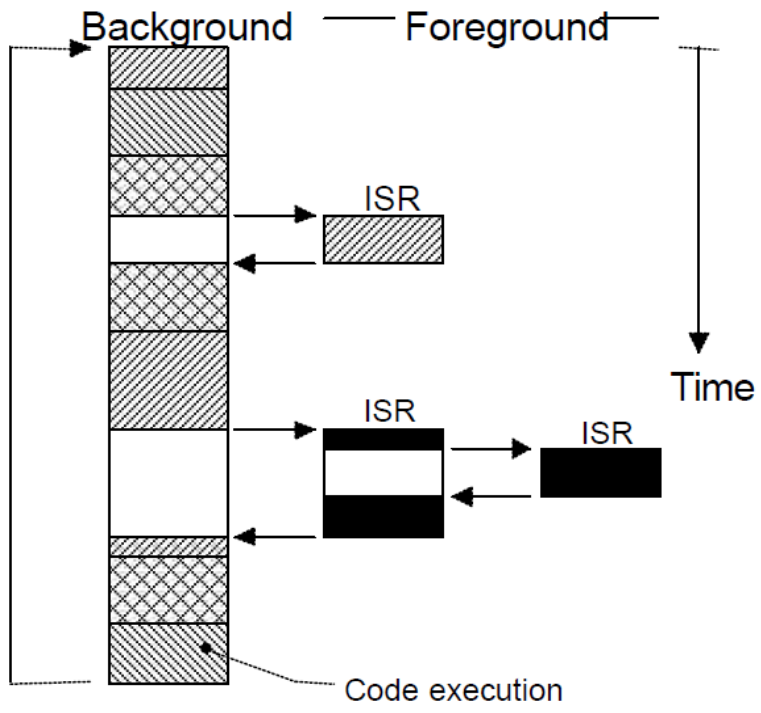
1. **Κυκλική εκτέλεση των διεργασιών**, μέσα σε βρόχο επανάληψης. Σύμφωνα με την τεχνική αυτή, οι διεργασίες εκτελούνται η μία μετά την άλλη. Ένας μετρητής μετρά τις επαναλήψεις του βρόχου. Ανάλογα με τις ανάγκες, κάποιες διεργασίες μπορεί να μην εκτελούνται κάθε φορά, αλλά μόνον κάθε ορισμένο αριθμό επαναλήψεων του βρόχου,



Σχ. 8.7 Κυκλική επανάληψη διεργασιών μέσα στο βρόχο ελέγχου (background loop) για το σύστημα του αυτόνομου οχήματος.



(α)

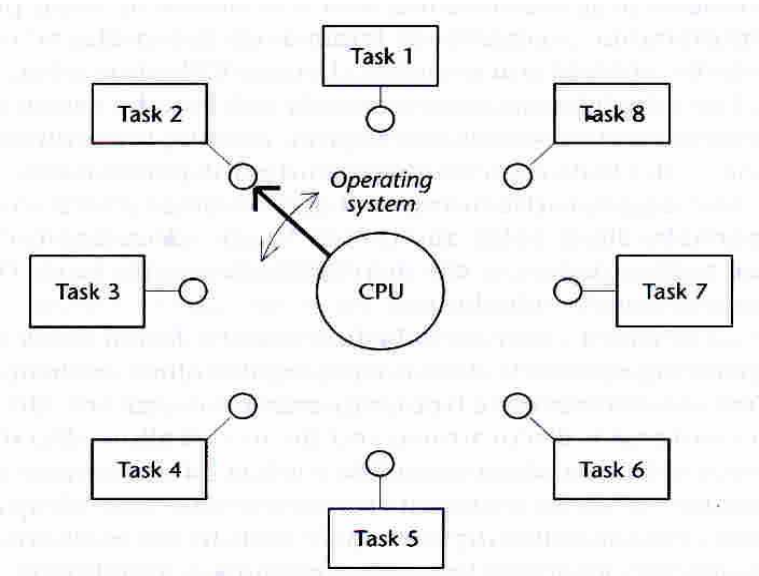


(β)

Σχήμα 8.8 Εκτέλεση υποβάθρου-προσκηνίου (background-forground loop) για το σύστημα του οχήματος (α) και στη γενική περίπτωση (β), όπου προβλέπονται και εμφωλευμένες διακοπές.

π.χ. κάθε πέντε ή δέκα επαναλήψεις. Με τον τρόπο αυτό, οι διεργασίες υψηλής προτεραιότητας ή αυτές που έχουν μικρότερη περίοδο επανάληψης εκτελούνται συχνότερα. Οι λιγότερο κρίσιμες μπορούν να εκτελούνται πιο αραιά. Στο σχήμα 8.7 φαίνεται ένας τέτοιος βρόχος κυκλικής επανάληψης των διεργασιών. Μια καθυστέρηση στο τέλος της κάθε επανάληψης μπορεί να ρυθμίσει την περίοδο επανάληψης του βρόχου και κατά συνέπεια την περίοδο με την οποία εκτελείται η κάθε διεργασία. Σύμφωνα με όσα περιγράψαμε, κάθε διεργασία θα εκετελείται με περίοδο που είναι ακέραιο πολλαπλάσιο της περιόδου επανάληψης του βρόχου. Η τεχνική αυτή αντιστοιχεί σ' αυτό που η βιβλιογραφία αναφέρει ως εκτέλεση βρόχου υποβάθρου (background loop).

2. **Εκτέλεση στο υπόβαθρο και στο προσκήνιο** (background-foreground loop). Η τεχνική αυτή εξασφαλίζει ότι κάποιες διεργασίες υψηλής προτεραιότητας θα ενεργοποιούνται μέσω σημάτων διακοπής διακόπτοντας όσες άλλες διεργασίες εκτελούνται στο βρόχο υποβάθρου. Στο παράδειγμα του συστήματος που περιγράφεται στο σχήμα 8.7, η αντίδραση σε συγκρούσεις που ανιχνεύονται από τους μικροδιακόπτες του οχήματος μπορεί να εκτελείται ως ρουτίνα διακοπής, ώστε να έχει υψηλή προτεραιότητα. Η διακοπή ενεργοποιείται από τα σήματα των μικροδιακοπτών, που κλείνουν όταν έρχονται σε επαφή με εμπόδια. Διεργασίες χαμηλότερης προτεραιότητας θα εκτελούνται στο βρόχο υποβάθρου, εφόσον δεν έχουν ενεργοποιηθεί οι διακοπές.
Στο σχήμα 8.8 εμφανίζεται η ιδέα της εκτέλεσης στο υπόβαθρο και στο προσκήνιο (background-foreground loop). Όπως φαίνεται στο σχήμα 8.8 (β) είναι δυνατό να προβλέπονται εμφωλευμένες διακοπές, ανάλογα με την προτεραιότητα κάθε πηγής διακοπής. Η δυνατότητα χρήσης εμφωλευμένων διακοπών είναι δυνατή στα πιο προχωρημένα συστήματα μικροελεγκτών.



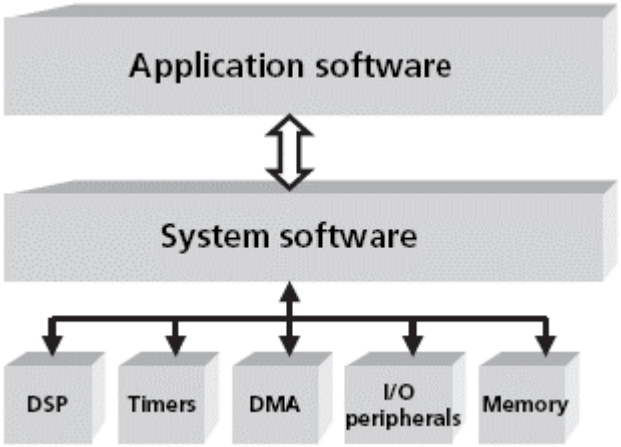
Σχήμα 8.9 Πυρήνας λειτουργικού συστήματος για την εκτέλεση διεργασιών

3. **Υλοποίηση λειτουργικού συστήματος πραγματικού χρόνου** (RTOS). Σε πολλές περιπτώσεις οι παραπάνω τεχνικές δεν έχουν πάντα τα επιθυμητά αποτελέσματα εκτέλεσης σε πραγματικό χρόνο. Στην περίπτωση αυτή, η χρονοδρομολόγηση (scheduling) των διεργασιών ανατίθεται σε ειδικό κώδικα, που αποτελεί έναν πυρήνα λειτουργικού συστήματος. Στη βάση κάθε χρονοδρομολογητή είναι ένας χρονιστής που παράγει ένα περιοδικό σήμα διακοπής. Ως αποτέλεσμα της διακοπής εκτελείται περιοδικά η ρουτίνα της χρονοδρομολόγησης (scheduler), η οποία λαμβάνει την απόφαση για το ποιά θα είναι η επόμενη διεργασία που θα εκτελεστεί. Ανάλογα με τη φύση του λειτουργικού συστήματος, ο scheduler μπορεί να διακόπτει μια διεργασία και να εκτελεί μία άλλη υψηλότερης προτεραιότητας. Ένας τέτοιος χρονοδρομολογητής ονομάζεται *preemptive scheduler*. Αντίθετα, είναι επίσης δυνατό ο χρονοδρομολογητής απλά να προετοιμάζει την επόμενη διεργασία προς εκτέλεση, με βάση την προτεραιότητα και να επιστρέφει στην τρέχουσα διεργασία. Αφού εκτελέσει την τρέχουσα διεργασία μέχρι την ολοκληρωσή της, τότε μεταβαίνει και εκτελεί την επόμενη διεργασία. Ένας τέτοιος δρομολογητής αναφέρεται ως *non-preemptive scheduler*. Στο σχήμα 8.9 φαίνεται η κεντρική ιδέα της εκτέλεσης διεργασιών με βάση τον χρονοδρομολογητή ενός λειτουργικού συστήματος.

9. Λειτουργικά Συστήματα πραγματικού χρόνου

9. 1 Πολυδιεργασία και πολυνημάτωση

Στο κεφάλαιο αυτό θα αναφερθούμε σε ορισμένα βασικά χαρακτηριστικά των λειτουργικών συστημάτων πραγματικού χρόνου. Το σχήμα 9.1 είναι μια υπενθύμιση για το ρόλο του λειτουργικού συστήματος (system software) να παρέχει έτοιμους μηχανισμούς προς το χρήστη για το χειρισμό των υπολογιστικών πόρων.

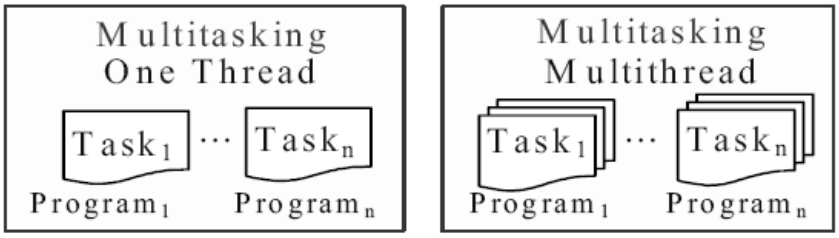


Σχ. 9.1 Το λειτουργικό σύστημα είναι μέρος του λογισμικού και διαχειρίζεται τους πόρους του συστήματος

Κατ' αρχάς ας ξεκαθαρίσουμε ότι η έννοια του «συστήματος πραγματικού χρόνου» και του «λειτουργικού συστήματος πραγματικού χρόνου» δεν είναι ταυτόσημες. Ένα λειτουργικό σύστημα πραγματικού χρόνου παρέχει μηχανισμούς που διευκολύνουν την υλοποίηση συστημάτων πραγματικού χρόνου.

Τα λειτουργικά συστήματα γενικού σκοπού, διαχειρίζονται διεργασίες, που η κάθε μια είναι ένα απλό πρόγραμμα που τρέχει στον δικό του χώρο μνήμης. Ένα λειτουργικό σύστημα που κάνει πολυδιεργασία, εφαρμόζει την τεχνική της κατανομής χρόνου (time-sharing) προκειμένου να πολυπλέξει τις διάφορες διεργασίες σε ισότιμη βάση.

Τα τελευταία χρόνια υπάρχει η τάση στα λειτουργικά συστήματα να υποστηρίζουν τη δημιουργία υπο-διεργασιών μέσα στο ίδιο πρόγραμμα, που η κάθε μια ελέγχεται ανεξάρτητα από τις άλλες. Επίσης, υποστηρίζουν απεριόριστη πρόσβαση σε κοινή μνήμη ανάμεσα στις διεργασίες, και διευκολύνουν τον συγχρονισμό και την μεταξύ τους επικοινωνία. Τέτοιες υπο-διεργασίες ονομάζονται «νήματα» (threads) και η δυνατότητα του λειτουργικού συστήματος να υποστηρίζει αυτή την «νημάτωση» των προγραμμάτων σε παράλληλες υπο-διεργασίες ονομάζεται «πολυνημάτωση» (multithreading). Το σχήμα 9.2 παρουσιάζει τη διαφορά ανάμεσα στα συστήματα πολυδιεργασίας (multitasking) και πολυνημάτωσης.



Σχ. 9.2 Οι έννοιες της πολυδιεργασίας και της πολυνημάτωσης στα λειτουργικά συστήματα

Ο «εν δυνάμει» παραλληλισμός που εισάγει η πολυνημάτωση είναι μια βασική ιδιότητα των λειτουργικών συστημάτων πραγματικού χρόνου. Μια άλλη βασική ιδιότητα των λειτουργικών συστημάτων πραγματικού χρόνου είναι η προβλεψιμότητα (predictability) και η καθοριστικότητα (determinism), όπως αυτές ορίστηκαν στην παράγραφο 8.8. Γενικά, ένα RTOS χαρακτηρίζεται *προβλέψιμο*, αν ο χρόνος για να αναγνωρίσει ένα αίτημα από ένα εξωτερικό γεγονός είναι γνωστός εκ των προτέρων. Όσο πιο ακριβής είναι αυτή η εκ των προτέρων γνώση, τόσο πιο *καθοριστικό* θεωρείται το σύστημα.

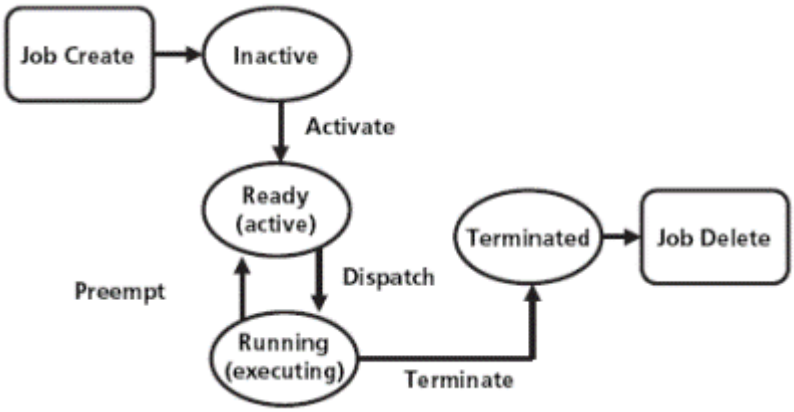
Στα μικρά ενσωματωμένα συστήματα εκτελείται κατά βάση μια εφαρμογή. Άρα, εδώ η έννοια της πολυδιεργασίας αντιστοιχεί βασικά στο νήμα των διεργασιών που εκτελούνται στο πλαίσιο αυτής της εφαρμογής.

9.2 Η έννοια της διεργασίας σε ένα RTOS

Ένα λειτουργικό σύστημα πραγματικού χρόνου διαχειρίζεται τα διάφορα προγράμματα χωρίζοντάς τα σε νήματα διεργασιών, τις οποίες και ελέγχει κατάλληλα. Σε πολλές περιπτώσεις, η διαίρεση μιας εφαρμογής σε μικρότερα τμήματα κώδικα που θα εκτελούνται παράλληλα, είναι έργο του προγραμματιστή.

Κάθε διεργασία αντιπροσωπεύει ένα τμήμα κώδικα που μπορεί να διεκδικεί τους πόρους του συστήματος, παράλληλα με άλλες διεργασίες. Οι πόροι είναι η μνήμη, η κεντρική μονάδα, οι περιφερειακές μονάδες. Κάποιο τμήμα κώδικα μπορεί να χρειάζεται συχνά τις περιφερειακές μονάδες, ενώ μια άλλη διεργασία μπορεί να χρειάζεται σε μεγάλο βαθμό την CPU. Ωστόσο, σε ένα σύστημα πραγματικού χρόνου δεν είναι δυνατό κάποιες διεργασίες να μονοπωλούν τους πόρους του συστήματος, ειδικά όταν μια πιο κρίσιμη διεργασία χρειάζεται τους ίδιους πόρους.

Έτσι, μια διεργασία, στο πλαίσιο ενός RTOS, αποκτά επιπλέον χαρακτηριστικά σε σχέση με αυτά μιας απλής εφαρμογής. Εδώ, διεργασία είναι ένα σύνολο μηχανισμών, μια οντότητα λογισμικού, που δημιουργείται από το λειτουργικό σύστημα προκειμένου να διαχειρίζεται την εκτέλεση ενός τμήματος κώδικα, με προβλέψιμο τρόπο. Κάθε διεργασία είναι ανεξάρτητη από τις άλλες διεργασίες και μπορεί να συγχρονιστεί και να επικοινωνήσει μαζί τους. Αφού δημιουργηθεί, έχει μια διάρκεια ζωής, κατά την οποία μπορεί να βρεθεί σε διάφορες «καταστάσεις», ανάλογα με το ποιες καταστάσεις διεργασιών υποστηρίζει το λειτουργικό σύστημα. Μια διεργασία μπορεί να μεταβαίνει

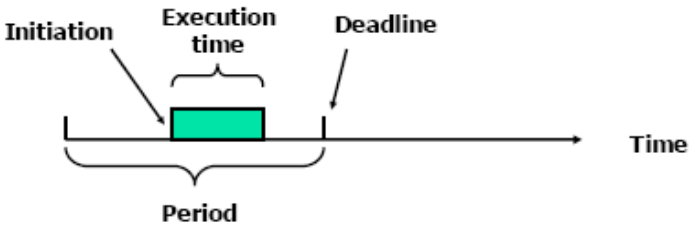


Σχ. 9.3 Τυπικό μοντέλο καταστάσεων διεργασίας για ένα RTOS

από τη μια κατάσταση στην άλλη σύμφωνα με το διάγραμμα καταστάσεων. Μπορεί, λοιπόν, να δημιουργείται, να βρίσκεται σε ανενεργή κατάσταση (sleeping), σε κατάσταση ετοιμότητας (ready), να εκτελείται και να τερματίζεται. Η τρέχουσα κατάστασή της καθορίζει ποια θα είναι η επόμενη κατάσταση στην οποία θα περιέλθει, με απόφαση που λαμβάνει ο χρονοδρομολογητής (scheduler). Η τρέχουσα κατάσταση μιας διεργασίας αποθηκεύεται σε έναν καταχωρητή κατάστασης, που μαζί με άλλα στοιχεία που αφορούν στη διεργασία, περιλαμβάνεται στο «τμήμα ελέγχου της διεργασίας» (task control block).

Το σχήμα 9.3 δείχνει διαγραμματικά τις βασικές καταστάσεις (states) που χαρακτηρίζουν μια διεργασία.

Το σχήμα 9.4 δείχνει τις βασικές χρονικές παραμέτρους μιας διεργασίας, στις οποίες αναφερθήκαμε και στο προηγούμενο κεφάλαιο. Από χρονική σκοπιά, κάθε διεργασία αποδεσμεύεται, ετοιμάζεται εκτελείται και τερματίζεται στα όρια της προθεσμίας. Ο χρονοδρομολογητής του RTOS αναλαμβάνει να διαχειριστεί την διεργασία μέσα στους χρονικούς της περιορισμούς.

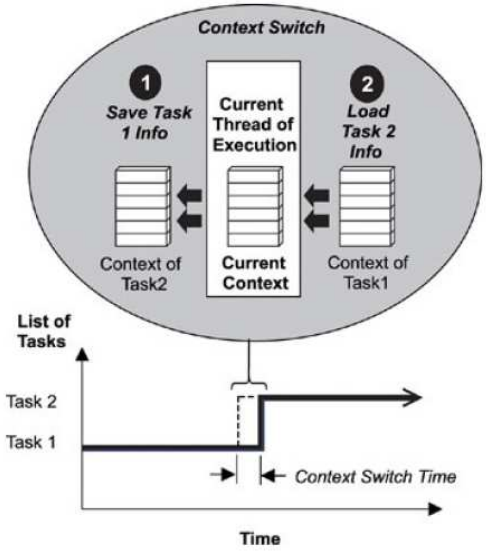


Σχ. 9.4 Χρονικές παράμετροι μιας διεργασίας. Στην αρχή της περιόδου έχουμε τον χρόνο αποδέσμευσης, στη συνέχεια η διεργασία είναι έτοιμη (initiation) και εκτελείται. Το RTOS την προγραμματίζει να εκτελεστεί μέσα στα όρια της προθεσμίας της (deadline).

9.3 Βασικές απαιτήσεις πολυδιεργασίας

Όταν ένα σύστημα σχεδιάζεται με σκοπό την πολυδιεργασία (multitasking) πρέπει να είναι σε θέση να επιτελέσει τα εξής:

1. **Μετάβαση περιεχομένου** (context switching). Κάθε διεργασία χρησιμοποιεί έναν αριθμό από καταχωρητές, που σχετίζονται με την εκτέλεση του κώδικα. Αυτοί περιλαμβάνουν τους καταχωρητές εργασίας, τον απαριθμητή προγράμματος, τον καταχωρητή κατάστασης, το δείκτη της στοίβας, καθώς και τις τιμές που αποθηκεύονται στη στοίβα, όσο εκτελείται η διεργασία. Οι τιμές αυτών των καταχωρητών είναι κρίσιμες και πρέπει να διατηρούνται όσο διαρκεί η εκτέλεση του κώδικα. Επιπρόσθετα, μπορεί να υπάρχουν κι άλλες κρίσιμες μεταβλητές, που σχετίζονται με την εκτέλεση αριθμητικών πράξεων ή και κρυμμένες μεταβλητές που χρησιμοποιούνται από συναρτήσεις ανώτερων γλωσσών προγραμματισμού. Όλες αυτές οι πληροφορίες αποτελούν το «περιεχόμενο» (context) της διεργασίας. Όταν σε ένα σύστημα πολυδιεργασίας η εκτέλεση του προγράμματος μεταβαίνει από τη μια διεργασία στην άλλη, το περιεχόμενο πρέπει να αποθηκεύεται κατάλληλα, ώστε να μπορεί να ξαναφορτώνεται στους κατάλληλους καταχωρητές, όταν επαναλαμβάνεται η εκτέλεση της διεργασίας. Κάθε αλλοίωση των τιμών των κρίσιμων καταχωρητών που προαναφέρθηκαν οδηγεί σε σοβαρά προβλήματα. Για παράδειγμα, αλλοίωση των τιμών της στοίβας θα οδηγήσει σε λάθος διεύθυνση επιστροφής από μια υπορουτίνα.
Για τους παραπάνω λόγους, κάθε φορά που διακόπτεται η εκτέλεση μιας διεργασίας και ξεκινά η εκτέλεση μιας άλλης, το σύστημα πρέπει να αναλαμβάνει να καλεί μια συνάρτηση που θα αποθηκεύει το περιεχόμενο της διεργασίας που διακόπτεται, καθώς και μια συνάρτηση που θα ανακαλεί το περιεχόμενο της διεργασίας που αρχίζει να εκτελείται. Η διαδικασία αυτή αναφέρεται ως «μετάβαση περιεχομένου» ή context switching και παριστάνεται διαγραμματικά στο σχήμα 9.5.



Σχ. 9.5 Η διαδικασία της μετάβασης (context switch) κατά την εναλλαγή διεργασιών

Είναι προφανές ότι για να επιτευχθεί η αποθήκευση περιεχομένου, κάθε διεργασία πρέπει να διαθέτει ένα χώρο της μνήμης του συστήματος (block μνήμης) που να προορίζεται για τον παραπάνω σκοπό. Τέλος, η μετάβαση χρειάζεται κάποιο χρόνο, που αναφέρεται ως context switch time.

2. **Επικοινωνία ανάμεσα σε διεργασίες** (task communication). Σε μια εφαρμογή, όπου ο κώδικας αποτελεί μια ενότητα, δηλαδή μια ενιαία διεργασία, τα αποτελέσματα που παράγονται από ένα τμήμα κώδικα μπορούν να μεταβιβάζονται και να χρησιμοποιούνται από άλλο τμήμα κώδικα, με μια φυσική ροή αποτελεσμάτων ανάμεσα σε καταχωρητές ή μεταβλητές. Όταν, όμως, πολλές διαφορετικές διεργασίες συνιστούν αυτόνομες οντότητες, τότε η εκτέλεση ή η διακοπή τους ρυθμίζεται από το λειτουργικό σύστημα. Στην περίπτωση αυτή, δεν διαδέχονται η μία την άλλη με φυσικό τρόπο, οπότε γεννιέται το ερώτημα, πώς θα μεταβιβάσει αποτελέσματα και δεδομένα η μία στην άλλη, όταν χρειάζεται. Αν η διεργασία Β πρέπει να λάβει αποτελέσματα από τη διεργασία Α, τότε αν η Β κληθεί προς εκτέλεση πριν εκτελεστεί η Α, τότε η Β ουσιαστικά εμποδίζεται (blocked). Επίσης είναι δυνατό η διεργασία Α να παράγει δεδομένα με διαφορετικό ρυθμό από τον ρυθμό με τον οποίο η Β μπορεί να δεχτεί τα δεδομένα. Άρα, χρειάζεται ένα σύστημα επικοινωνίας ανάμεσα στις διεργασίες, που να εξασφαλίζει ότι η κάθε μια μπορεί να στέλνει και να λαμβάνει δεδομένα όποτε χρειάζεται.

3. **Διαχείριση προτεραιοτήτων.** Σε κάθε σύστημα πραγματικού χρόνου κεντρικό ρόλο παίζει η απόδοση προτεραιοτήτων στις διάφορες διεργασίες. Με βάση τις σχετικές προτεραιότητες των διεργασιών λαμβάνεται η απόφαση για το ποιά διεργασία είναι πιο σημαντικό να εκτελεστεί κάθε στιγμή. Η απόδοση προτεραιοτήτων γίνεται με τη βοήθεια ειδικών αλγορίθμων, ειδικά στα μεγάλα συστήματα. Στα μικρά ενσωματωμένα συστήματα συνήθως γίνεται χρήση σταθερών προτεραιοτήτων που αποδίδονται στις διεργασίες εκ των προτέρων.

4. **Έλεγχος του χρονισμού**. Κάθε σύστημα πραγματικού χρόνου πρέπει να διαθέτει ένα σύστημα που να καθορίζει κάθε πότε θα εκτελείται η κάθε διεργασία. Φυσικά, ένα τέτοιο σύστημα συμπλέκεται και με τις προτεραιότητες των διεργασιών. Τις περισσότερες φορές ο χρονισμός των διεργασιών στηρίζεται σε χρονιστές, που πρέπει να προγραμματίζονται κατάλληλα.

9.4 Τεχνικές πολυδιεργασίας στα λειτουργικά συστήματα

Ένα λειτουργικό σύστημα μπορεί να ακολουθεί διάφορες στρατηγικές για την επιτέλεση της πολυδιεργασίας. Υπάρχουν διαφορετικές τεχνικές που καθορίζουν πως μια διεργασία *παραδίδει τον έλεγχο* σε άλλη. Έτσι, μπορεί να έχουμε

- **Συνεργαζόμενη πολυδιεργασία**

Με την συνεργαζόμενη πολυδιεργασία (cooperative multitasking) μια διεργασία παραδίδει τον έλεγχο σε άλλη μόνον «εθελοντικά». Για να χάσει η διεργασία τον έλεγχο

πρέπει να καλέσει η ίδια το λειτουργικό σύστημα (RTOS). Με τον τρόπο αυτό οι διεργασίες μοιράζονται δίκαια τον επεξεργαστή και τους άλλους πόρους.

- **Ολοκληρωμένη εκτέλεση**

Μια διεργασία παραδίδει τον έλεγχο μόνον όταν έχει ολοκληρώσει το έργο της (run to completion). Από πρακτική άποψη αυτή η τεχνική μπορεί να χρησιμοποιηθεί σε ένα RTOS μόνον όταν όλες οι διεργασίες είναι σχετικά σύντομες.

- **Διακοπή από τον προγραμματιστή διεργασιών (preemption)**

Στην περίπτωση αυτή μια διεργασία παραδίδει τον έλεγχο όποτε υπαγορεύει ο χρονοπρογραμματιστής (scheduler). Η τεχνική αναφέρεται ως preemption και θα την δούμε πιο αναλυτικά παρακάτω. Γενικά, οι schedulers που στηρίζονται σ' αυτή την τεχνική μπορούν να ανταποκριθούν σε συγκεκριμένες χρονικές απαιτήσεις καλύτερα από άλλους.

9. 5 Χαρακτηριστικά λειτουργικών συστημάτων πραγματικού χρόνου

Με βάση όσα περιγράψαμε παραπάνω, τα σύγχρονα λειτουργικά συστήματα πραγματικού χρόνου έχουν συνήθως τα εξής χαρακτηριστικά:

- Δυνατότητα χρονοδρομολόγησης των διεργασιών με βάση προτεραιότητες (preemptive scheduling based on priorities)
- Γρήγορη μετάβαση ανάμεσα σε διεργασίες (switch context)
- Μικρό μέγεθος
- Επικοινωνία και συγχρονισμό ανάμεσα σε διεργασίες
- Ύπαρξη χρονιστών πραγματικού χρόνου, για τον έλεγχο χρονισμού

Τέλος, εξυπακούεται ότι ένα λειτουργικό σύστημα πραγματικού χρόνου υποστηρίζει όλες τις βασικές λειτουργίες ενός συμβατικού λειτουργικού συστήματος, όπως η διαχείριση διακοπών (interrupt handling), η διαχείριση διεργασιών (task manager), η διαχείριση της μνήμης (εκχώρηση, απελευθέρωση και προστασία της μνήμης), η διαχείριση εισόδου/εξόδου μέσω οδηγών συσκευών (device drivers) και η επικοινωνία ανάμεσα σε διεργασίες.

9.6 Πυρήνας RTOS

Το κέντρο ενός λειτουργικού συστήματος ονομάζεται **πυρήνας** (kernel). Στα λειτουργικά συστήματα πραγματικού χρόνου ο πυρήνας αποτελείται από τον χρονοδρομολογητή διεργασιών (scheduler), τον διαχειριστή διακοπών (interrupt handler), και τον διαχειριστή πόρων (resource manager), που διαχειρίζεται τη μνήμη και τον διαμοιρασμό της στις διάφορες διεργασίες.

Στις επόμενες παραγράφους θα αναφερθούμε με συντομία στις βασικότερες λειτουργίες των λειτουργικών συστημάτων πραγματικού χρόνου.

9.7 Χρονοδρομολόγηση πραγματικού χρόνου

Η βασική διάκριση ανάμεσα στα λειτουργικά συστήματα πραγματικού χρόνου και στα συμβατικά λειτουργικά συστήματα βρίσκεται στον διαχειριστή διεργασιών (task manager). Αυτός αποτελείται από τον αποστολέα (dispatcher) και το χρονοδρομολογητή (scheduler).

Ο αποστολέας, γενικά, ελέγχει τη ροή εκτέλεσης του προγράμματος, η οποία αποφασίζεται από τον scheduler. Έτσι, ο αποστολέας αναλαμβάνει τη μετάβαση περιεχομένου (switch context), δηλαδή την μετάβαση από την μια διεργασία στην άλλη ή τη μετάβαση σε ρουτίνες εξυπηρέτησης διακοπής. Όπως αναφέρθηκε (βλέπε παράγραφο 9.3), κατά την μετάβαση αυτή αποθηκεύεται η κατάσταση της διεργασίας που διακόπτεται και φορτώνεται η κατάσταση της διεργασίας που πρόκειται να εκτελεστεί.

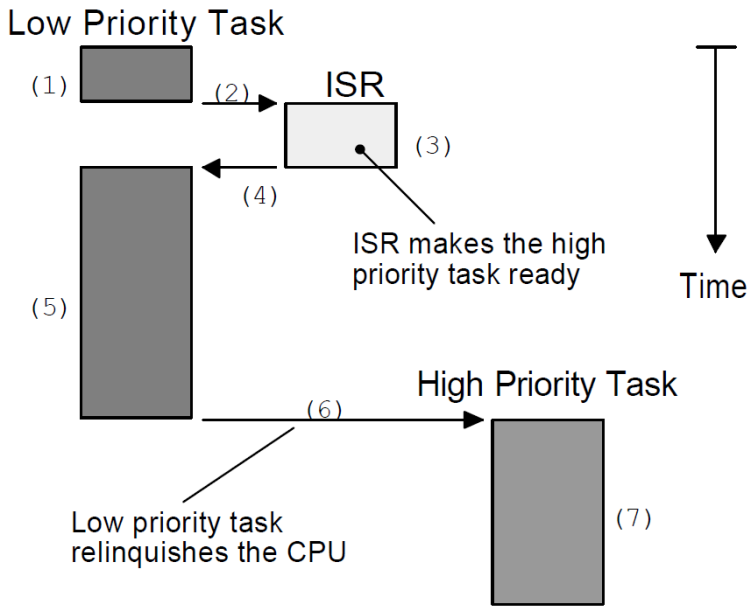
Ο χρονοδρομολογητής (scheduler) είναι το μέρος του λειτουργικού συστήματος που αποφασίζει κάθε φορά ποια θα είναι η επόμενη διεργασία που θα εκτελεστεί από τον επεξεργαστή, ώστε το σύστημα να είναι προβλέψιμο, μέσα στις προδιαγραφές. Η επιλογή αυτή γίνεται με τη βοήθεια ειδικών αλγορίθμων και τεχνικών χρονοδρομολόγησης. Στη συνεργαζόμενη πολυδιεργασία η κλήση του χρονο-δρομολογητή γίνεται από την ίδια τη διεργασία, που με τον τρόπο αυτό παραδίδει τον έλεγχο εθελοντικά. Σε άλλες περιπτώσεις, ο χρονοδρομολογητής εκτελείται περιοδικά, ως διακοπή που προκαλεί ένας χρονιστής. Όπως αναφέρθηκε και στην παράγραφο 8.10.3, διακρίνουμε γενικά δύο μορφές χρονοδρομολόγησης, χωρίς διακοπή διεργασιών (non-preemptive) και με διακοπή διεργασιών (preemptive).

9.7.1 Χρονοδρομολόγηση χωρίς διακοπή διεργασιών (non-preemptive)

Χρησιμοποιείται συνήθως σε εφαρμογές, όπου οι διεργασίες έχουν παρόμοιες απαιτήσεις για τη χρήση των πόρων του συστήματος και ο χρόνος εκτέλεσής τους είναι σχετικά μικρός. Στην περίπτωση αυτή, ο αλγόριθμος χρονοδρομολόγησης επιτρέπει την ολοκλήρωση της τρέχουσας διεργασίας, πριν μεταβεί σε μια άλλη, υψηλότερης προτεραιότητας. Μια μορφή ενός τέτοιου συστήματος χρονοδρομολόγησης φαίνεται στο σχήμα 9.6. Όταν καλείται ο scheduler (στο σχήμα αυτό γίνεται με ένα σήμα διακοπής, που μπορεί να παράγει ένας χρονιστής), η υπορουτίνα ISR φέρνει σε κατάσταση ετοιμότητας (ready) τη διεργασία ύψιστης προτεραιότητας, αλλά δεν την εκτελεί. Η διεργασία υψηλής προτεραιότητας έρχεται σε κατάσταση εκτέλεσης μόνον μετά την ολοκλήρωση της τρέχουσας διεργασίας.

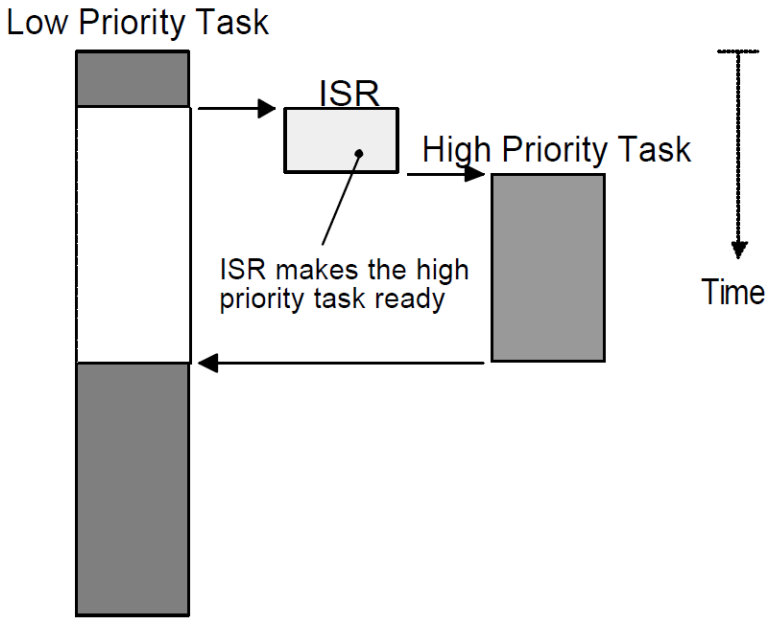
9.7.2 Χρονοδρομολόγηση με διακοπή διεργασιών (preemptive)

Όπως αναφέρθηκε και στα προηγούμενα, οι διάφορες διεργασίες δεν έχουν πάντα την ίδια σπουδαιότητα, ή αλλιώς δεν έχουν την ίδια κρισιμότητα (criticality). Έτσι, συχνά είναι ανάγκη να υπάρχει ένας μηχανισμός ώστε μια διεργασία χαμηλής κρισιμότητας να

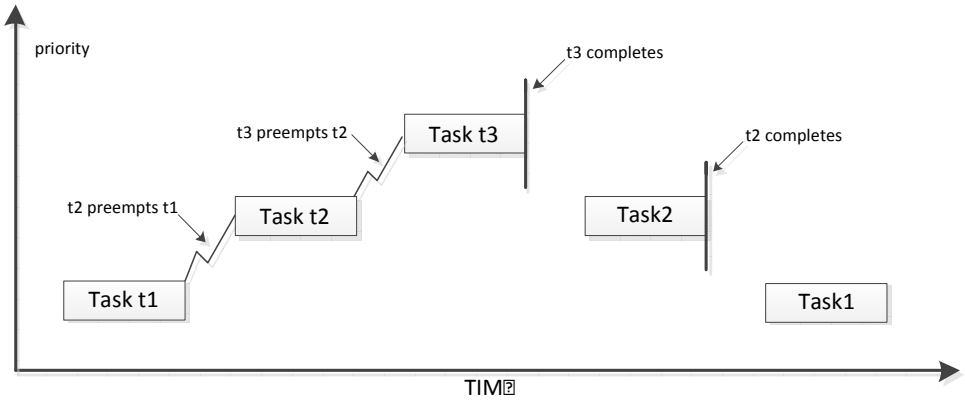


Σχ. 9.6 Παράδειγμα χρονοδρομολόγησης χωρίς διακοπή (non-preemptive)

διακόπτεται και να παραδίδει τους πόρους σε πιο κρίσιμες διεργασίες. Η διαδικασία αυτή ονομάζεται **διακοπή της διεργασίας** (preemption). Αν μια διεργασία μπορεί να διακοπεί και να παραχωρήσει τη σειρά της σε άλλη, τότε λέμε ότι επιδέχεται διακοπή (preemtable). Όταν ο scheduler υποστηρίζει διακοπή διεργασιών με βάση προτεραιότητες, τον χαρακτηρίζουμε ως preemptive scheduler, priority-based.



Σχ. 9.7 Παράδειγμα χρονοδρομολόγησης με διακοπή διεργασιών (preemptive scheduler)



Σχ. 9.8 Διαδοχική διακοπή διεργασιών σε preemptive scheduler, με βάση προτεραιότητες

Στο σχήμα 9.7 φαίνεται η λειτουργία ενός τέτοιου συστήματος χρονο-δρομολόγησης. Αρχικά εκτελείται μια διεργασία χαμηλής προτεραιότητας. Όταν καλείται ο scheduler, για παράδειγμα, μέσω διακοπής, τότε η διεργασία υψηλότερης προτεραιότητας μεταβαίνει στην κατάσταση ετοιμότητας (ready) και στη συνέχεια ο αποστολέας (dispatcher) εκτελεί τη μετάβαση (switch context) και τρέχει τη διεργασία υψηλής προτεραιότητας. Στη συνέχεια, η εκτέλεση επιστρέφει στη διεργασία χαμηλότερης προτεραιότητας.

Στο σχήμα 9.8 παρουσιάζεται η διαδοχική διακοπή διεργασιών, κάθε φορά που καλείται ο scheduler, ώστε πάντα να εκτελείται η διεργασία υψηλότερης προτεραιότητας. Στον κατακόρυφο άξονα βρίσκεται η προτεραιότητα των διεργασιών. Η διεργασία t2 διακόπτει την t1, ενώ η t3 διακόπτει με τη σειρά της την t2. Επειδή δεν προκύπτει διεργασία υψηλότερης προτεραιότητας από την t3, αυτή ολοκληρώνεται. Στη συνέχεια ολοκληρώνεται η t2 και τέλος η t1.

9.8 Τεχνικές χρονοδρομολόγησης

Υπάρχουν τρεις σημαντικές στρατηγικές που καθορίζουν πώς ο χρονοδρομολογητής (scheduler) καθορίζει τη σειρά με την οποία οι εκτελούνται οι διάφορες διεργασίες. Η δρομολόγηση των διεργασιών μπορεί να στηρίζεται σε μια από τις εξής τεχνικές:

- Οδήγηση από χρονιστή (clock-driven)
- Περιστρεφόμενη εκτέλεση (round-robin)
- Οδήγηση από προτεραιότητες (priorities)

Στις επόμενες παραγράφους θα εξετάσουμε τις τρεις παραπάνω τεχνικές χρονοδρομολόγησης.

9.8.1 Οδήγηση από χρονιστή

Στην τεχνική αυτή ο χρονοδρομολογητής λαμβάνει αποφάσεις για το ποιες διεργασίες θα εκτελεστούν και πότε, σε συγκεκριμένες χρονικές στιγμές. Οι στιγμές

αυτές είναι γνωστές εκ των προτέρων, πριν αρχίσει η εκτέλεση του συστήματος. Τυπικά, σε ένα σύστημα που χρησιμοποιεί χρονοδρομολόγηση με βάση την οδήγηση από χρονιστή όλες οι παράμετροι των διεργασιών πραγματικού χρόνου (περίοδος, χρόνος εκτέλεσης, προθεσμία, προτεραιότητα κλπ.) είναι γνωστές εκ των προτέρων. Το σύστημα υπολογίζει έναν πρόγραμμα εργασιών off-line, το αποθηκεύει και το χρησιμοποιεί κατά τον χρόνο εκτέλεσης. Ο χρονοδρομολογητής χρησιμοποιεί έναν χρονιστή, δηλαδή ένα προγραμματιζόμενο κύκλωμα παλμών, ώστε να παίρνει αποφάσεις δρομολόγησης διεργασιών σε καθορισμένα χρονικά διαστήματα.

Ας υποθέσουμε ότι έχουμε δύο διεργασίες T_1 και T_2, με περιόδους $p_1=p_2=4$ ms και χρόνο εκτέλεσης c=2ms η κάθε μια. Το πρόγραμμα που υπολογίζεται εκ των προτέρων, είναι ένας πίνακας που περιλαμβάνει απλούστατα τις δύο διεργασίες διαδοχικά, και τους χρόνους λήψης της απόφασης κατά την εκτέλεση του προγράμματος, που προκύπτουν κάθε 2 ms. Ο χρονιστής προγραμματίζεται να παράγει έναν παλμό λήψης απόφασης κάθε 2 ms, όταν δηλαδή θα έχει ολοκληρωθεί η εκτέλεση κάθε προηγούμενης διεργασίας. Σε κάθε παλμό θα εναλλάσσει τις διεργασίες.

Ένα περιοδικό στατικό πρόγραμμα, όπως το παραπάνω, το καλούμε “κυκλική εκτέλεση”. Ένα πρόγραμμα που ελέγχεται από παλμούς ρολογιού κατά την εκτέλεσή του, παρουσιάζει ομαλή και αυστηρά προβλέψιμη συμπεριφορά, και άρα είναι κατάλληλο για αυστηρά (hard) συστήματα πραγματικού χρόνου.

Ας σημειωθεί ότι σε ένα πιο περίπλοκο πρόγραμμα από το παραπάνω, με ακανόνιστους χρόνους εκτέλεσης και περιόδους, ο χρονιστής μπορεί να προγραμματίζεται να παράγει ένα σήμα διακοπής μόλις τελειώνει ο χρόνος εκτέλεσης της προηγούμενης διεργασίας. Τότε, ο scheduler αναλαμβάνει να παραδώσει τον έλεγχο στην επόμενη διεργασία που έχει στον πίνακα και να θέσει τον χρονιστή ώστε να παράγει σήμα διακοπής μόλις ολοκληρωθεί η εκτέλεση της νέας διεργασίας. Τη στιγμή της νέας διακοπής προγραμματίζεται η επόμενη διεργασία κ.ο.κ. Το σύστημα είναι πάλι κυκλικό, με την έννοια ότι χαρακτηρίζεται από μια υπερπερίοδο επανάληψης του προγράμματός του, σύμφωνα με όσα αναφέρθηκαν σε προηγούμενο κεφάλαιο.

Η ύπαρξη χρονιστών είναι κεντρική και στους επόμενους δύο τρόπους χρονοδρομολόγησης, όμως σ’ αυτούς κυριαρχούν και άλλα στοιχεία στη λήψη της απόφασης για την εκτέλεση ή όχι μιας διεργασίας.

9.8.2 Περιστρεφόμενη εκτέλεση (Round-Robin)

Πρόκειται για μια συνήθη τεχνική χρονοδρομολόγησης, που εφαρμόζεται κυρίως σε συστήματα κατανομής χρόνου (time-sharing). Σύμφωνα με την τεχνική αυτή, όταν μια διεργασία είναι «έτοιμη» προς εκτέλεση, εισέρχεται σε μια ουρά καταχώρησης τύπου First-In-First-Out (FIFO). Οι διεργασίες εκτελούνται με βάση μια θυρίδα χρόνου (time-slice), η οποία τυπικά έχει διάρκεια μερικές δεκάδες χιλιοστά του δευτερολέπτου. Η πρώτη στην ουρά εκτελείται πρώτη, ακολουθεί η δεύτερη κ.ο.κ. Εάν μια διεργασία δεν έχει ολοκληρωθεί κατά το πέρας της χρονοθυρίδας, τότε διακόπτεται από τον scheduler (preempted) και τοποθετείται στο τέλος της ουράς για να αναμένει την

επόμενη σειρά της. Αν έχουμε *n* διεργασίες στην ουρά, τότε κάθε διεργασία εκτελείται κάθε *n* χρονοθυρίδες, δηλαδή σε κάθε «γύρο». Κάθε διεργασία έχει μερίδιο κατά το ένα νιοστό του χρόνου του επεξεργαστή, δηλαδή όλες έχουν ίσο μερίδιο.

Μια παραλλαγή της παραπάνω τεχνικής χρησιμοποιήθηκε στον προγραμματισμό της κυκλοφορίας σε πραγματικό χρόνο, σε δίκτυα υψηλής ταχύτητας. Εκεί, η χρονοθυρίδα δεν δίνεται ισότιμα στις διεργασίες, αλλά με βάση παράγοντες βαρύτητας (weights). Ανάλογα με τον παράγοντα βαρύτητας που την χαρακτηρίζει, μια διεργασία λαμβάνει ανάλογο μερίδιο σε χρονοθυρίδες, ώστε χρησιμοποιεί ανάλογα και τον επεξεργαστή.

9.8.3 Οδήγηση με προτεραιότητες

Ο προγραμματισμός *περιοδικών* διεργασιών σε σύστημα ενός επεξεργαστή είναι από τα πιο κλασσικά προβλήματα χρονοπρογραμματισμού σε συστήματα πραγματικού χρόνου. Έχουν προταθεί δύο εναλλακτικές προσεγγίσεις, βασισμένες στην απόδοση μιας σταθερής ή δυναμικής προτεραιότητας σε κάθε διεργασία. Στην περίπτωση της **σταθερής προτεραιότητας** (fixed priority), η τιμής της προτεραιότητας κάθε διεργασίας υπολογίζεται μια φορά, προσδίδεται στην διεργασία και παραμένει αναλλοίωτη σε όλη τη διάρκεια ζωής της διεργασίας. Στην άλλη περίπτωση, η προτεραιότητα υπολογίζεται κατά τη διάρκεια του χρόνου εκτέλεσης και συνδέεται με την προθεσμία που δίνουμε στην κάθε διεργασία. Η δεύτερη αυτή περίπτωση ονομάζεται επίσης «**οδηγούμενη από προθεσμία**» (deadline driven).

Η τεχνική της διακοπής διεργασιών (preemption) που περιγράψαμε στις προηγούμενες παραγράφους, στηρίζεται στην λογική των προτεραιοτήτων. Μία διεργασία διακόπτει κάποια άλλη, όταν έχει υψηλότερη προτεραιότητα. Φυσικά, είναι δυνατό να έχουμε προγραμματισμό με βάση προτεραιότητες, χωρίς να εφαρμόζεται η τεχνική της διακοπής. Κάθε διεργασία, με σειρά προτεραιότητας, μπορεί να εκτελείται μέχρι το τέλος (run to completion). Αυτή είναι η περίπτωση της χρονοδρομολόγησης non-preemptive, που αναφέρθηκε παραπάνω.

9.9 Απόδοση προτεραιοτήτων

Οι αλγόριθμοι χρονοδρομολόγησης με βάση τις προτεραιότητες μπορούν να χωριστούν σε δύο κατηγορίες: τους *στατικούς* και τους *δυναμικούς*. Για την εφαρμογή στατικών αλγορίθμων πρέπει να είναι γνωστές εκ των προτέρων όλες οι πληροφορίες για τον προγραμματισμό των διαδικασιών (αριθμός διεργασιών, προθεσμίες, προτεραιότητες, περίοδοι κλπ.) Έτσι, το πρόβλημα του προγραμματισμού των διεργασιών λύνεται πριν εκτελεστεί (off-line). Στους δυναμικούς αλγόριθμους το πρόγραμμα των διεργασιών καθορίζεται κατά τη διάρκεια της εκτέλεσης. Έτσι, αλλαγές στη διαμόρφωση του χρονικού προγραμματισμού μπορούν να συμβούν on-line.

Προφανώς, οι στατικοί αλγόριθμοι χαρακτηρίζονται από καλύτερη προβλεψιμότητα σε σχέση με τους δυναμικούς, αλλά ταυτόχρονα είναι πολύ πιο

άκαμπτοι και αδύναμοι να αντιμετωπίσουν νέες καταστάσεις που προκύπτουν κατά την διάρκεια της εκτέλεσης (run time).

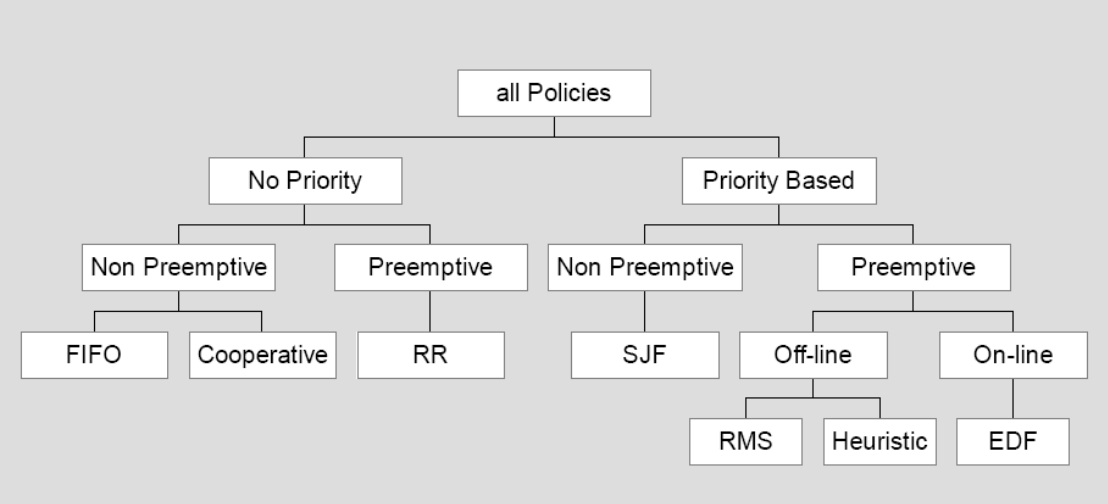
Σε λειτουργικά συστήματα που υποστηρίζουν διακοπή εργασιών με βάση προτεραιότητες (priority-based, preemptive RTOS), χρησιμοποιούνται διάφορες τεχνικές αλγορίθμων, από τις οποίες οι πιο σημαντικές είναι οι εξής:

Μονοτονικού ρυθμού (Rate Monotonic – RM). Η τεχνική αυτή αντιστοιχεί μια σταθερή προτεραιότητα σε κάθε διεργασία, σύμφωνα με την εξής αρχή: όσο πιο μικρή είναι η περίοδος της διεργασίας, τόσο μεγαλύτερη είναι η προτεραιότητα. Έχει αποδειχτεί ότι η τεχνική αυτή παρουσιάζει μια βέλτιστη συμπεριφορά ανάμεσα σε άλλες παρόμοιες τεχνικές που λειτουργούν με βάση σταθερές προτεραιότητες.

Προήγηση συντομότερης προθεσμίας (Earliest Deadline First – EDF). Όσο συντομότερη είναι η προθεσμία μιας διεργασίας, τόσο υψηλότερη είναι η προτεραιότητά της.

Προήγηση ελάχιστου χρόνου χαλάρωσης (Least Slack-time First – LSTF). Ο χρόνος χαλάρωσης (slack-time) ορίζεται ως η διαφορά του διαστήματος από την τρέχουσα χρονική στιγμή μέχρι το πέρας της προθεσμίας μιας διεργασίας και του χρόνου εκτέλεσης της διεργασίας: (απόλυτη προθεσμία - τρέχουσα χρονική στιγμή) - χρόνος εκτέλεσης. Σύμφωνα με την τεχνική αυτή η διεργασία που χαρακτηρίζεται την τρέχουσα στιγμή από τον ελάχιστο χρόνο χαλάρωσης έχει και την μεγαλύτερη προτεραιότητα.

Οι δύο παραπάνω τεχνικές αποδίδουν στις διεργασίες δυναμικές προτεραιότητες, με βάση κριτήρια που σχετίζονται με τις απόλυτες προθεσμίες. Δηλαδή όσο πιο κοντά στη λήξη της προθεσμίας βρισκόμαστε τόσο μεγαλύτερη προτεραιότητα αποκτά η διεργασία.



Σχ. 9.9 Κατηγορίες τεχνικών χρονοπρογραμματισμού. RR: *Round Robin*, SJF: *Shortest Job First*, RMS: *Rate Monotonic Scheduling*, EDF: *Earliest Deadline First*.

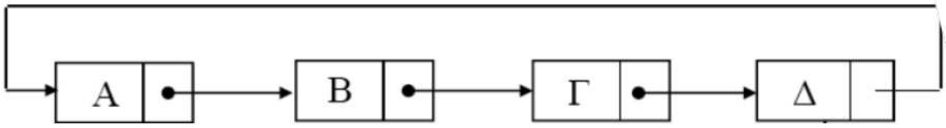
9.10 Χρονοδρομολόγηση απεριοδικών διεργασιών

Εκτός από τις παραπάνω τεχνικές που προγραμματίζουν περιοδικές διεργασίες με βάση προτεραιότητες, έχουμε και τις απεριοδικές και σποραδικές διεργασίες (βλέπε παράγραφο 8.7). Μια πολιτική χρονοδρομολόγησης απεριοδικών διεργασιών είναι να εκτελούνται μόνον όταν δεν υπάρχουν έτοιμες προς εκτέλεση (ready) περιοδικές διεργασίες. Μια άλλη τεχνική είναι να δημιουργήσουμε μια περιοδική διεργασία με συγκεκριμένη προτεραιότητα, που ο σκοπός της είναι να ελέγχει αν υπάρχουν απεριοδικές διεργασίες και να τις εξυπηρετεί (τεχνική περιοδικού ελέγχου – polling).

9.11 Μηχανισμοί υποστήριξης διεργασιών

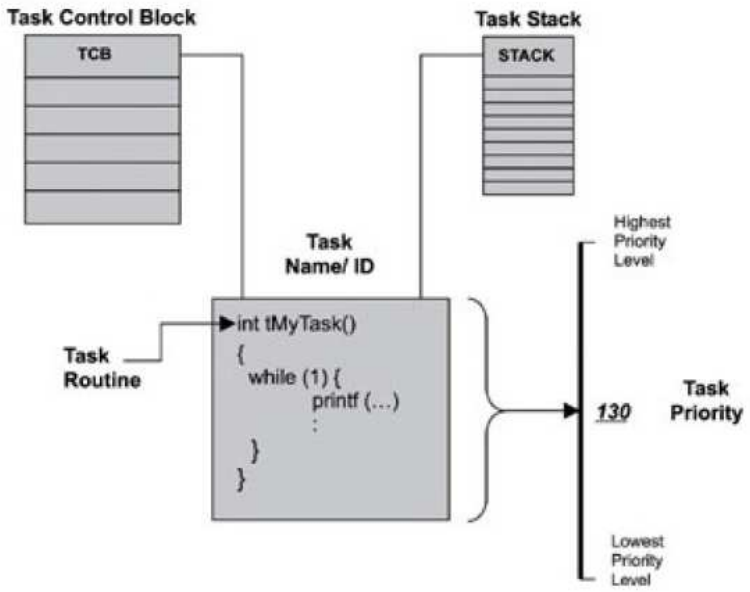
9.11.1 Task Control Block

Κάθε φορά που δημιουργείται μια διεργασία, παράγεται και συσχετίζεται με αυτήν μια δομή δεδομένων (data structure) που ονομάζεται τμήμα ελέγχου της διεργασίας ή task control block (TCB). Εκεί φυλλάσσονται όλες οι πληροφορίες που πρέπει να γωρίζει το λειτουργικό σύστημα για τη συγκεκριμένη διεργασία. Τα TCBs υλοποιούνται ως κυκλικά συνδεδεμένη λίστα, όπου ο αριθμός των κόμβων ισούται με τον αριθμό των διεργασιών (σχήμα 9.10).



Σχ. 9.10 Κυκλικά συνδεδεμένη λίστα για την υλοποίηση των task control blocks

Στο TCB μιας διεργασίας αποθηκεύονται μεταβλητές, όπως η προτεραιότητα της διεργασίας, η κατάστασή της (ready, running, paused κλπ.), η καθυστέρηση προκειμένου να μεταβεί η διεργασία από την κατάσταση παύσης στην κατάσταση ετοιμότητας κ. ά. Επίσης, στο TCB περιέχονται δείκτες (pointers). Ένας τυπικός τέτοιος δείκτης είναι αυτός που δείχνει πάντα την τιμή που πρέπει να λάβει ο program counter προκειμένου να εκτελέσει τη συγκεκριμένη διεργασία. Ένας άλλος δείκτης είναι αυτός που δείχνει στο επόμενο TCB της κυκλικά συνδεδεμένης λίστας. Με τον τρόπο αυτό ο scheduler μπορεί να διατρέχει τη συνδεδεμένη λίστα και να ελέγχει την κατάσταση της κάθε διεργασίας, την καθυστέρησή της, την προτεραιότητα κλπ. προκειμένου να αποφασίσει για την επόμενη προς εκτέλεση διεργασία. Προφανώς, ο δείκτης του τελευταίου κόσμβου TCB δείχνει στον πρώτο. Ένας άλλος πολύ σημαντικός δείκτης είναι αυτός που χρησιμοποιείται για την ανταλλαγή δεδομένων ανάμεσα σε διεργασίες.

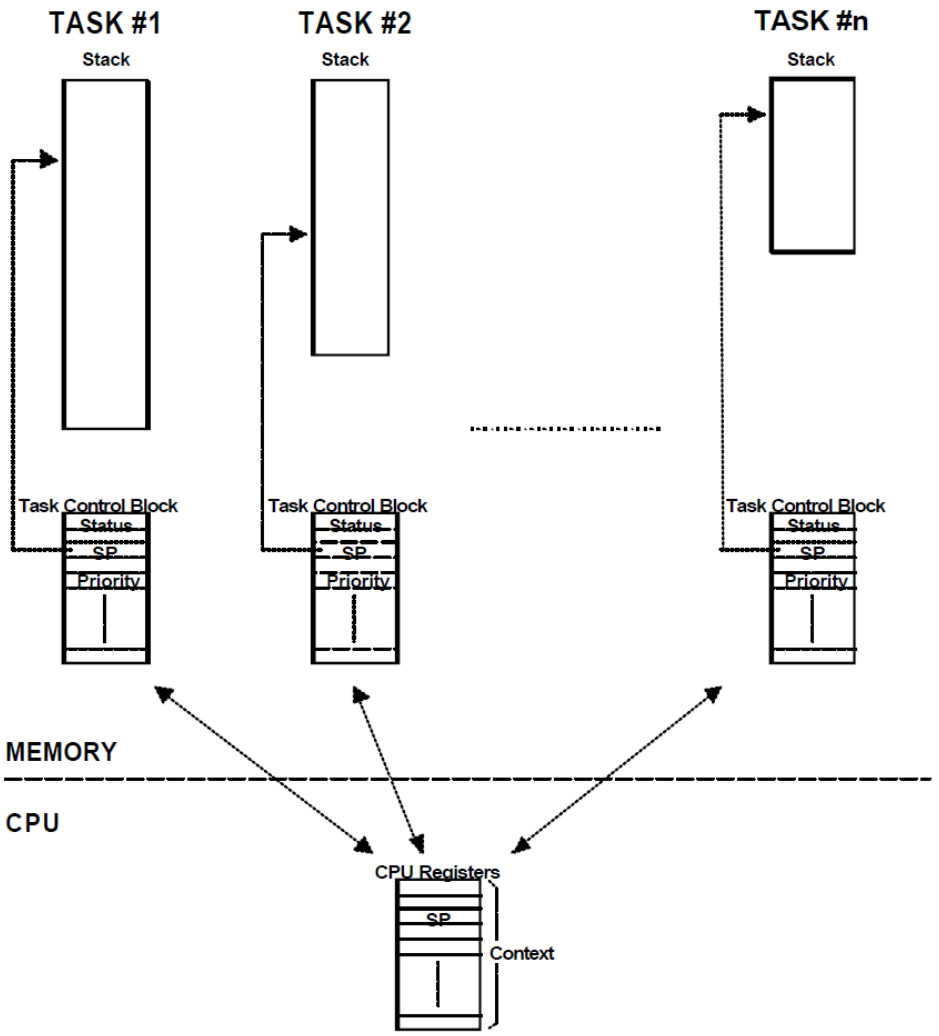


Σχ. 9. 11 Τμήμα ελέγχου διεργασίας (TCB) και στοίβα διεργασίας (task stack)

Έτσι, μια διεργασία μπορεί να αποστείλει δεδομένα σε μια άλλη, αποθηκεύοντας τα δεδομένα στη μνήμη, όπου δείχνει ο συγκεκριμένος δείκτης. Ο μηχανισμός που χρησιμοποιείται για την ανταλλαγή μηνυμάτων ανάμεσα σε διεργασίες ονομάζεται «ταχυδρομική θυρίδα» (mailbox) και θα σχολιαστεί παρακάτω. Τέλος, κάποιοι δείκτες ορίζουν το χώρο της μνήμης όπου αποθηκεύεται το «περιεχόμενο» της διεργασίας κατά τη μετάβαση (context switching). Αυτός ο χώρος της μνήμης είναι η «στοίβα» της διεργασίας (task stack). Εκεί αποθηκεύονται οι καταχωρητές εργασίας, καταχωρητές κατάστασης, ο απαριθμητής προγράμματος και όποιοι άλλοι βασικοί καταχωρητές είναι απαραίτητοι για την ομαλή εκτέλεση της διεργασίας. Οι παραπάνω ιδέες παριστάνονται διαγραμματικά στο σχήμα 9.11.
Αφού η εκτέλεση των διεργασιών ελέγχεται από το λειτουργικό σύστημα μέσω των παραπάνω μηχανισμών, το σχήμα 8.9 του προηγούμενου κεφαλαίου μπορεί να τώρα να παρασταθεί όπως στο σχήμα 9.12. Ο αλγόριθμος χρονοδρομολόγησης χρησιμοποιεί κάθε φορά το TCB και τη στοίβα κάθε διεργασίας, ώστε να της αναθέσει περιοδικά τον έλεγχο των πόρων του συστήματος.

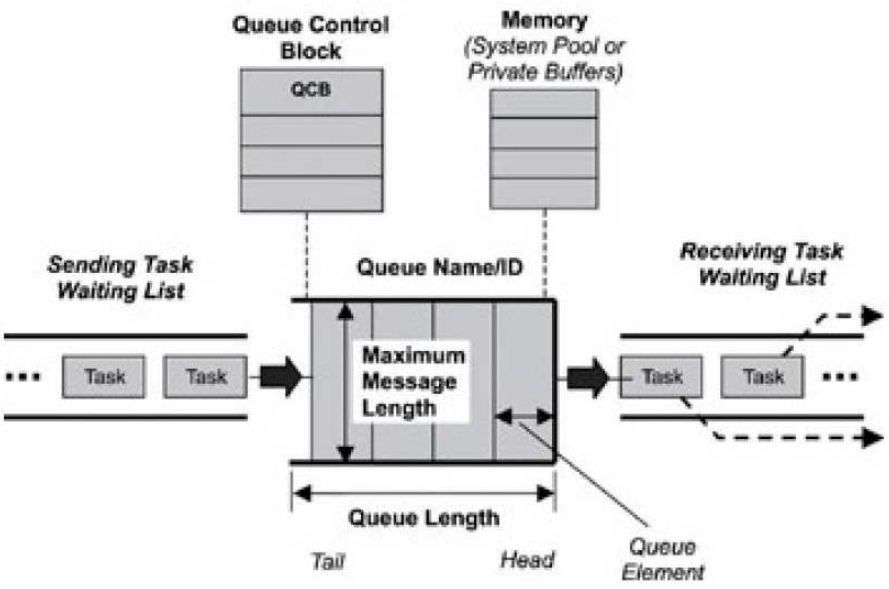
9.11.2 Ταχυδρομικές θυρίδες

Η ταχυδρομική θυρίδα (mailbox) είναι ένας μηχανισμός επικοινωνίας ανάμεσα στις διεργασίες. Χρησιμοποιείται όταν μια διεργασία Α παράγει δεδομένα που θέλει να τα στείλει σε άλλη διεργασία Β ή όταν η διεργασία Β θέλει να λάβει τα δεδομένα που της έχουν αποστείλει άλλες διεργασίες. Η ανάγκη της επικοινωνίας σχολιάστηκε και στην παράγραφο 9.3. Το δεδομένο προς αποστολή μπορεί να ειναι μια τιμή φυσικού μεγέθους (θερμοκρασία, πίεση κλπ.) που έχει μετατρέψει ο ADC, ενώ το δεδομένο που



Σχ. 9.12 Κάθε διεργασία χρησιμοποιεί τους κοινόχρηστους πόρους, με τη βοήθεια του TCB και της στοίβας που της αντιστοιχεί.

λαμβάνεται μπορεί να σταλεί για απεικόνιση σε μια οθόνη LCD. Έτσι, η διεργασία-αποστολέας μπορεί να είναι αυτή που ελέγχει τον ADC ενώ η διεργασία-δέκτης μπορεί να είναι αυτή που ελέγχει την οθόνη LCD. Στην απλή περίπτωση, μια ταχυδρομική θυρίδα είναι μια μεταβλητή που περιέχει έναν δείκτη. Μέσω μια συνάρτησης του λειτουργικού συστήματος μια διεργασία-αποστολέας μπορεί να αποθέσει έναν δείκτη στη θυρίδα, ενώ αντίστοιχα, μια διεργασία-δέκτης μπορεί να λάβει τον δείκτη προκειμένου να διαβάσει το μήνυμα-δεδομένο. Τόσο η διεργασία-αποστολέας, όσο και η διεργασία-δέκτης πρέπει να συμφωνούν ως προς τον τύπο του δεδομένου.
Αν περισσότερες από μία διεργασίες θέλουν να λάβουν ή να στείλουν δεδομένα, τότε τοποθετούνται σε λίστες αναμονής (waiting lists). Γενικά, το mailbox υλοποιείται ως μια ουρά και έχει συγκεκριμένο μήκος, ενώ τα μηνύματα που αποθηκεύονται μπορούν να έχουν μέχρι ένα μέγιστο μέγεθος. Η βασική ιδέα των ταχυδρομικών θυρίδων παρουσιάζεται στο σχήμα 9.13.



Σχ. 9.13 Υλοποίηση ταχυδρομικής θυρίδας (mailbox)

9.11.3 Σημαιοφόροι (semaphores)

Ο σημαιοφόρος είναι ένας μηχανισμός πρωτοκόλλου, που χρησιμοποιείται από τα περισσότερα λειτουργικά συστήματα. Είναι ένα αντικείμενο του λειτουργικού συστήματος που δεσμεύεται όταν μια διεργασία Α χρησιμοποιεί ένα κοινόχρηστο πόρο (π.χ. τον εκτυπωτή) και αποδεσμεύεται όταν η διεργασία απελευθερώνει τον πόρο. Αν μια άλλη διεργασία Β διεκδικεί τον ίδιο πόρο, όσο ο σημαιοφόρος είναι δεσμευμένος, τότε αυτή πρέπει να αναμένει την απαλευθέρωση του πόρου. Όταν αυτό συμβεί, τότε η διεργασία Β κατέχει τον σημαιοφόρο και κάθε άλλη πρέπει να περιμένει. Έτσι, η πρώτη διεργασία που θα διεκδικήσει τον πόρο θα βρει τον σημαιοφόρο σε κατάσταση GO (1) ενώ η δεύτερη θα τον βρει σε κατάσταση WAIT (0).

Αν ο πόρος μπορεί να κατέχεται κάθε φορά μόνον από μία διεργασία, τότε είναι δυαδικός σημαιοφόρος (binary semaphore). Αν ο πόρος μπορεί να δεσμευτεί πολλές φορές, μέχρι έναν μέγιστο αριθμό, από ισάριθμες διεργασίες, τότε είναι counting semaphore και μπορεί να πάρει τιμές από 0 μέχρι 2^n, ανάλογα με τον αριθμό n των bits που χρησιμοποιούνται για την υλοποίηση του σημαιοφόρου. Αυτό μπορεί να συμβεί αν το σύστημα περιλαμβάνει περισσότερους από έναν ίδιους πόρους (π.χ. πολλούς εκτυπωτές).

Οι σημαιοφόροι χρησιμοποιούνται για τους παρακάτω λόγους:

- Για να ελέγχουν την πρόσβαση σε κοινόχρηστους πόρους
- Για να σηματοδοτούν ένα συμβάν
- Για να επιτρέπουν σε δύο διεργασίες να συγχρονίζουν τις δραστηριότητές τους (π.χ. εάν η μία έχει στείλει δεδομένο σε θυρίδα, τότε η άλλη μπορεί να το λάβει. Ο σχετικός σημαιοφόρος είναι σε κατάσταση GO).

9.12 Αδιέξοδα

Ένας συνηθισμένος κίνδυνος της πολυδιεργασίας σε συστήματα πραγματικού χρόνου είναι τα λεγόμενα αδιέξοδα (deadlocks). Πρόκειται για την κατάσταση, όπου δύο προγράμματα που μοιράζονται τους ίδιους πόρους, εμποδίζουν το ένα το άλλο να προσπελάσουν τους κοινούς πόρους, έτσι ώστε τελικά δεν λειτουργεί κανένα. Ο παρακάτω σενάριο δείχνει πως δημιουργείται ένα αδιέξοδο:

Το πρόγραμμα 1 ζητά τον πόρο Α και τον λαμβάνει.

Το πρόγραμμα 2 ζητά τον πόρο Β και τον λαμβάνει.

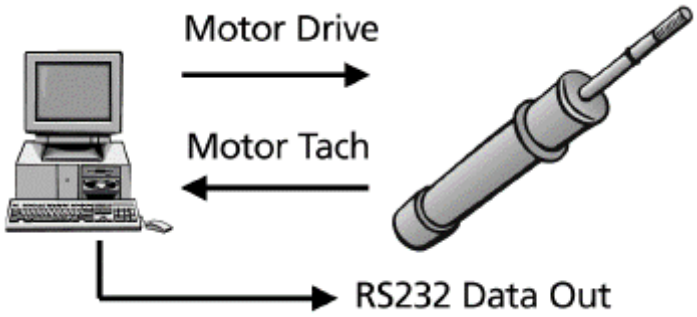
Το πρόγραμμα 1 ζητά τον πόρο Β και μπαίνει στην ουρά περιμένοντας την απελευθέρωση του Β.

Το πρόγραμμα 2 ζητά τον πόρο Α και μπαίνει στην ουρά, περιμένοντας την απελευθέρωση του Α.

Τώρα, κανένα πρόγραμμα δεν μπορεί να προχωρήσει, μέχρι το άλλο πρόγραμμα να απελευθερώσει τον πόρο που καταλαμβάνει. Το λειτουργικό σύστημα πρέπει να διακόψει το ένα από τα δύο προγράμματα.

9.13 Μελέτη περίπτωσης (Case Study) RTOS

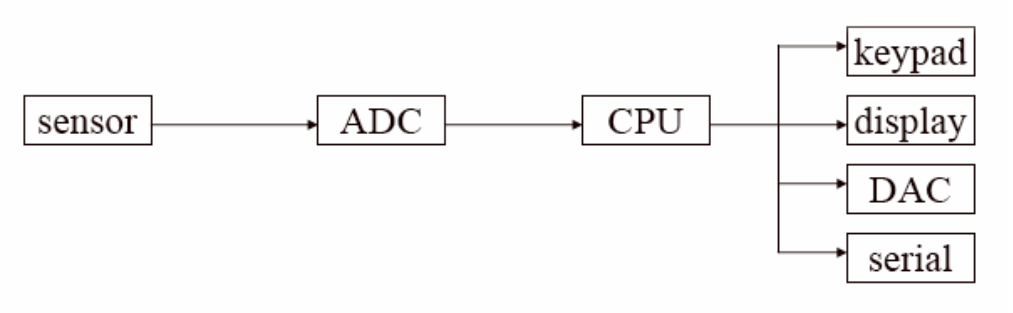
Ως ένα παράδειγμα των τεχνικών και των εννοιών που παρουσιάστηκαν σ' αυτό το κεφάλαιο, θα μελετήσουμε ένα απλό παράδειγμα ελέγχου ενός κινητήρα. Για τον έλεγχο του κινητήρα θα χρησιμοποιηθεί ένας DSP ελεγκτής, που αποτελεί και παράδειγμα ενσωματωμένου συστήματος (embedded system). Ο DSP ελέγχει επίσης ένα πληκτρολόγιο, μέσω του οποίου ένας χειριστής μπορεί να στείλει δεδομένα ελέγχου στο σύστημα, εμφανίζει κάποια δεδομένα σε μια απλή διάταξη απεικόνισης, όπως μια μικρή οθόνη και στέλνει δεδομένα προς άλλες συσκευές, μέσω μιας θύρας του DSP. Οι λειτουργίες αυτές υποστηρίζονται από απλό λειτουργικό σύστημα που τρέχει στον ελεγκτή. Στο σχ. 3.9 φαίνεται ένα γενικό διάγραμμα της εφαρμογής. Στη θέση του DSP εμφανίζεται ένας υπολογιστής γενικής χρήσης.



Σχ. 9.14 Απλό παράδειγμα ελέγχου κινητήρα

Σύμφωνα με το γενικό σχήμα ελέγχου, που παρουσιάσαμε στο κεφάλαιο 1, ένας αισθητήρας πρέπει να λαμβάνει δείγματα της ταχύτητας του συστήματος, με ρυθμό 1KHz και ένας ADC μετατρέπει το δείγμα σε ψηφιακή τιμή. Ένας χρονιστής θα παράγει ένα σήμα διακοπής με ρυθμό 1KHz, δηλαδή κάθε ένα χιλιοστό του δευτερολέπτου. Σε κάθε διακοπή ο DSP θα διαβάσει την ταχύτητα του μοτέρ με τη βοήθεια του αισθητήρα. Με βάση την τιμή μέτρησης, ο DSP εκτελεί μερικούς υπολογισμούς και ρυθμίζει την ταχύτητα του κινητήρα ανάλογα. Για την μετατροπή του ψηφιακού σήματος σε αναλογικό, προκειμένου να ελεγχθεί το μοτέρ, χρησιμοποιείται ένας DAC. Όταν ο χειριστής στείλει μέσω του πληκτρολογίου το κατάλληλο σήμα ελέγχου, ο DAC στέλνει μέσω της σειριακής θύρας (RS232) διαγνωστικά δεδομένα. Στο σχ. 9.15 φαίνεται το σχήμα ελέγχου.

Ορισμός των νημάτων-διεργασιών: Το πρώτο βήμα στην ανάπτυξη ενός συστήματος πολυδιεργασίας είναι να δομήσουμε την εφαρμογή με απομονωμένα και ανεξάρτητα μεταξύ τους νήματα διεργασίας. Αυτό μπορεί να γίνει και με τη βοήθεια



Σχ. 9.15 Σχήμα ελέγχου για την εφαρμογή του ελεγκτή κινητήρα

ειδικών εργαλείων. Από αυτή τη φάση σχεδίασης, θα προκύψουν λογικά διαγράμματα και διαγράμματα βαθμίδων του συστήματος. Για την εφαρμογή που περιγράψαμε, ένα παράδειγμα ανεξάρτητων διεργασιών φαίνεται στον πίνακα 9.1.

Στο σύστημά μας υπάρχουν τέσσερις ανεξάρτητες διεργασίες: 1. Η διεργασία που εκτελεί τον αλγόριθμο ελέγχου του μοτέρ, η οποία είναι περιοδική, με συχνότητα 1KHz (περίοδο 1ms). 2. Η διεργασία που επικοινωνεί με το πληκτρολόγιο, που είναι μια απεριοδική διεργασία κάτω από τον έλεγχο του χειριστή. 3. Η διεργασία ανανέωσης της οθόνης, που είναι περιοδική, με συχνότητα 2 Hz (περίοδος 0,5 s). 4. Η διεργασία αποστολής δεδομένων μέσω της σειριακής θύρας, η οποία τρέχει στο υπόβαθρο και αποστέλλει δεδομένα όταν δεν υπάρχει άλλη ανάγκη επεξεργασίας.

- Έλεγχος της ταχύτητας του μοτέρ (1KHz ρυθμός δειγματοληψίας)
- Λήψη εντολών ελέγχου από το πληκτρολόγιο, και εκτέλεση του βρόγχου ελέγχου
- Οδήγηση της οθόνης και ανανέωση δύο φορές το δευτερόλεπτο.
- Αποστολή δεδομένων μέσω της σειριακής θύρας όταν δεν υπάρχει άλλη επεξεργασία

Οι απαιτήσεις πραγματικού χρόνου αυτής της εφαρμογής φαίνονται περιληπτικά στο παραπάνω πλαίσιο.

Ορισμός προτεραιοτήτων: Αφού οριστούν οι βασικές διεργασίες, πρέπει να οριστεί η σχετική προτεραιότητα των διεργασιών. Επειδή πρόκειται για σύστημα πραγματικού χρόνου, με αυστηρές προθεσμίες και κρίσιμες λειτουργίες, πρέπει να οριστεί μια προτεραιότητα για την εκτέλεση κάθε διεργασίες. Η πιο αυστηρή διεργασία είναι ο αλγόριθμος ελέγχου, που πρέπει να εκτελείται με ρυθμό 1 KHz. Αυτή έχει και την ψηλότερη προτεραιότητα. Επίσης, υπάρχουν διεργασίες με χαλαρές προθεσμίες, όπως η ανανέωση της οθόνης με ρυθμός 2Hz. Προφανώς, η διεργασία είναι χαλαρή, με την έννοια ότι αν και ζητούμε να ανανεώνεται η οθόνη με ρυθμό 2Hz, το σύστημα δεν θα αχρηστευθεί αν δεν τηρηθεί ακριβώς αυτή η προθεσμία. Ο έλεγχος του πληκτρολογίου είναι επίσης χαλαρή διεργασία, αλλά έχει υψηλότερη προτεραιότητα από την ανανέωση της οθόνης, αφού αντιπροσωπεύει την βασική είσοδο δεδομένων προς το σύστημα. Η επικοινωνία με την σειριακή θύρα, πάλι, μπορεί να εκτελείται χωρίς αυστηρή προθεσμία και έχει την χαμηλότερη προτεραιότητα.

Χρήση σημάτων διακοπής: Το σύστημα ελέγχου του μοτέρ θα σχεδιαστεί με χρήση σημάτων διακοπής. Τα σήματα διακοπής χαρακτηρίζονται από ταχεία μετάβαση περιεχομένου και μπορούν να παραχθούν από τον χρονιστή του συστήματος. Στον πίνακα 9.1 φαίνονται οι προτεραιότητες των τεσσάρων διεργασιών. Πρόκειται για προτεραιότητες μονοτονικού ρυθμού (Rate Monotonic): όσο μικρότερη η περίοδος της διεργασίας, τόσο υψηλότερη η προτεραιότητα.

Στον ίδιο πίνακα περιγράφεται και ο μηχανισμός ενεργοποίησης της κάθε διεργασίας. Η διεργασία ελέγχου του μοτέρ ενεργοποιεί την εκτέλεσή της με σήμα διακοπής. Τα σήματα διακοπής, γενικά, στα λειτουργικά συστήματα πραγματικού

Διεργασία	**Ρυθμός**	**Προτεραιότητα**	**Περιοδική/απεριοδική**	**Μηχανισμός ενεργοποίησης**
Βρόγχος ελέγχου κινητήρα	1KHz	1	Περιοδική	Σήμα διακοπής
Έλεγχος πληκτρολογίου	5 Hertz	2	Απεριοδική	Σήμα διακοπής
Έλεγχος οθόνης	2 Hertz	3	Περιοδική	Διακοπή λογισμικού
Σειριακή θύρα	Στο υπόβαθρο	4	Απεριοδική	Ατέρμων βρόγχος

Πίνακας 9.1: Απόδοση προτεραιότητας στις διεργασίες του ελέγχου κινητήρα και μηχανισμοί ενεργοποίησης διεργασιών

χρόνου αντιπροσωπεύουν τους μηχανισμούς χρονοπρογραμματισμού με την υψηλότερη προτεραιότητα. Ο έλεγχος του πληκτρολογίου επιτυγχάνεται επίσης με σήμα διακοπής, αλλά μικρότερης προτεραιότητας. Δηλαδή, κάθε φορά που ο χειριστής χρησιμοποιεί το πληκτρολόγιο, ενεργοποιείται μια ρουτίνα διακοπής, που ελέγχει για χαρακτήρες πέντε φορές το δευτερόλεπτο. Επειδή ενεργοποιείται από τον χειριστή, η υπορουτίνα αυτή είναι απεριοδική. Η διεργασία ανανέωσης της οθόνης είναι περιοδική και ενεργοποιείται κάθε μισό δευτερόλεπτο, με τη βοήθεια μια υπορουτίνας διακοπής που ενεργοποιείται με εντολή λογισμικού (software interrupt). Τέλος, η σειριακή αποστολή δεδομένων τρέχει διαρκώς στο υπόβαθρο, ως ατέρμων βρόγχος, με χαμηλή προτεραιότητα και όταν δεν εκτελείται καμιά άλλη διεργασία.

ΠΑΡΑΡΤΗΜΑ Α

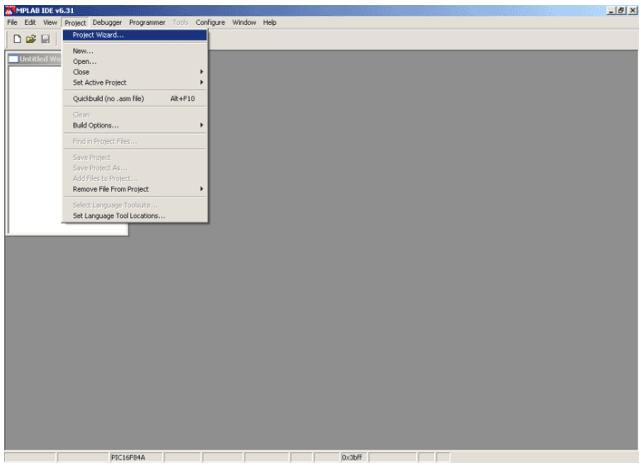
Εισαγωγή στον εξομοιωτή MPLAB

Η εργαστηριακή άσκηση που ακολουθεί έχει σκοπό να εξοικειώσει τους σπουδαστές με τις βασικές εντολές προγραμματισμού των μικροελεγκτών PIC και με το περιβάλλον συμβολομετάφρασης και προσομοίωσης του κώδικα σε γλώσσα Assembly. Θα χρησιμοποιήσουμε το περιβάλλον MPLAB, που αποτελεί ελεύθερο λογισμικό της εταιρίας Microchip.

Μπορείτε να καταφορτώσετε το λογισμικό MPLAB v.8 (legacy) από το site της εταιρίας Microchip (www.microchip.gr).

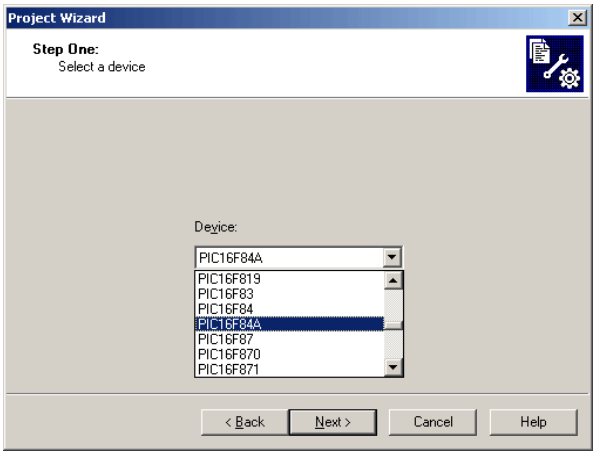
Βήμα 1^{o}: Δημιουργία Σχεδίου Εργασίας

Ξεκινείστε το λογισμικό MPLAB, ρυθμίζουμε τις λεπτομέρειες του παραθύρου έργου επιλέγοντας από το μενού PROJECT είτε την επιλογή NEW είτε την επιλογή PROJECT WIZARD. Στην δεύτερη περίπτωση θα εμφανιστεί ένα παράθυρο όπως αυτό στην εικόνα Π.1



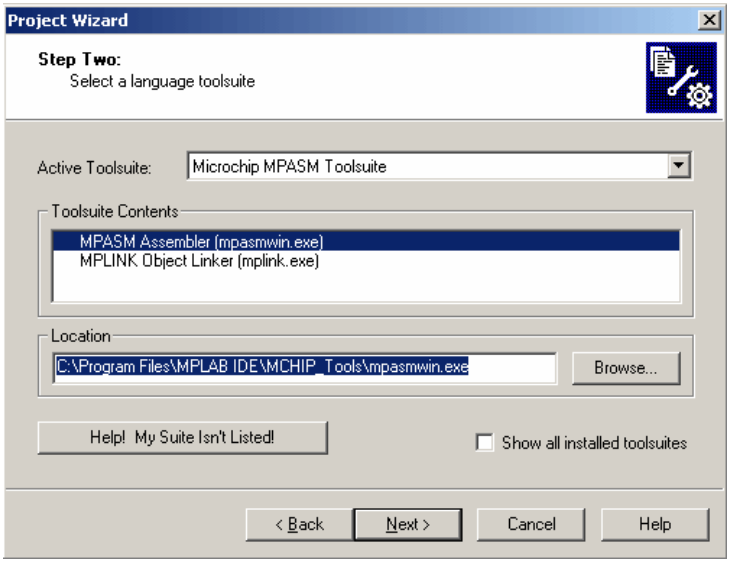
Εικόνα Π.1: Το παράθυρο έναρξης project στο MPLAB

Στη συνέχεια θα πρέπει να επιλέξουμε τον τύπο του μικροελεγκτή που θα χρησιμοποιήσουμε όπως φαίνεται στην εικόνα Π.2



Εικόνα Π.2: Το παράθυρο επιλογής μικροελεγκτή

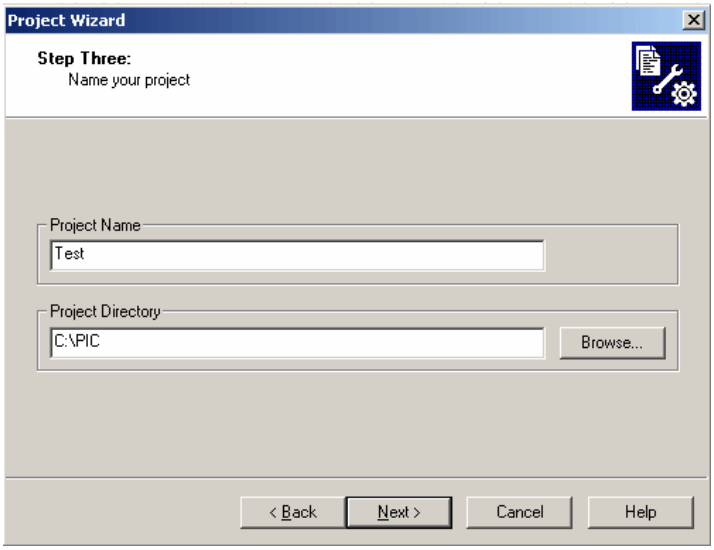
Με το επόμενο βήμα, καθορίζουμε την γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε που στην περίπτωσή μας είναι η Συμβολική γλώσσα Assembly για τους μικροελεγκτές PIC και για το λόγο αυτό επιλέγουμε να φορτωθεί ο συμβολομεταφραστής MPASMWIN.EXE με καθορισμένη διαδρομή αναζήτησης που οδηγεί στο φάκελο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



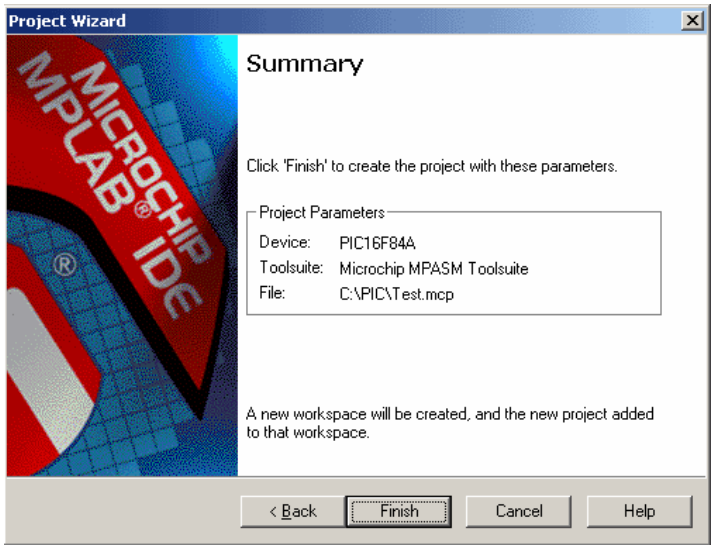
Εικόνα Π.3: Το παράθυρο επιλογής της γλώσσας προγραμματισμού

Τέλος, δίνουμε όνομα στο έργο που θα δημιουργήσουμε καθώς και τον φάκελο στο οποίο το έργο θα αποθηκευτεί σύμφωνα με το παράθυρο της εικόνας Π.4.

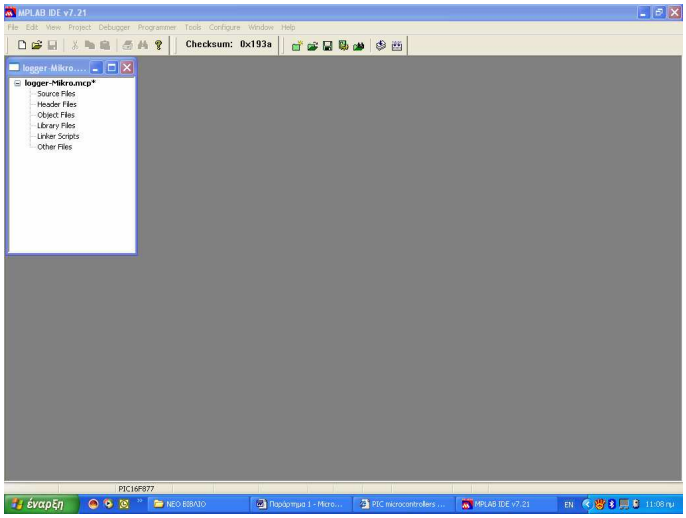
Στον παραπάνω φάκελο αυτό, θα αποθηκεύουμε και όλα τα αρχεία με τους κώδικες assembly που θα δημιουργήσουμε στην συνέχεια. Η διαδικασία αυτή ολοκληρώνεται με την διαδοχική επιλογή των πλήκτρων NEXT και FINISH οπότε εμφανίζεται ένα παράθυρο με την περίληψη όλων των ρυθμίσεων που δώσαμε έως τώρα.



Εικόνα Π.4: Το παράθυρο απόδοσης ονόματος στο έργο.



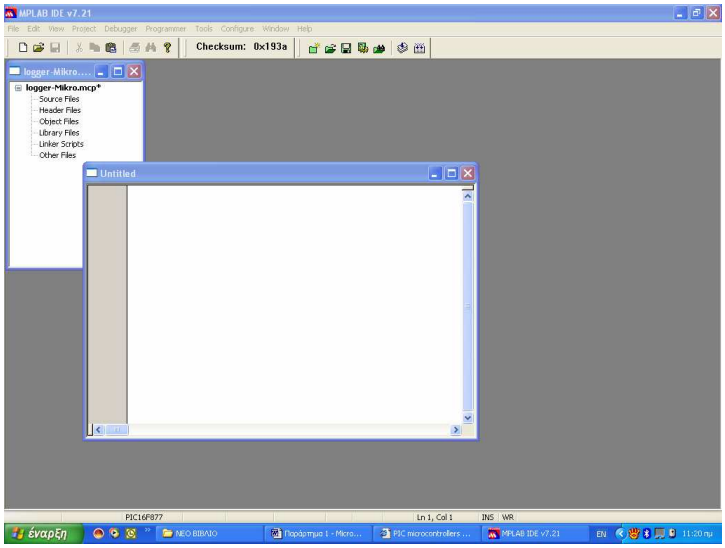
Εικόνα Π.5: Το παράθυρο περίληψης των ρυθμίσεων του έργου



Εικόνα Π.6: Το παράθυρο του συντάκτη (editor) για την συγγραφή κώδικα

Βήμα 2^{ο} Δημιουργία .ASM Αρχείου

Μετά την ολοκλήρωση της δημιουργίας του έργου, εμφανίζεται ένα παράθυρο όμοιο με εκείνο της εικόνας Π.6. Για να γράψουμε τον κώδικα assembly 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 assembly που μας ενδιαφέρει.



Εικόνα Π.7: Το παράθυρο έργου (project)

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 .mcp)) όσο και για τα αρχεία assembly (με προέκταση .asm)που συνδέονται με το έργο.

Επιλέγουμε File-New. Στον Editor που ανοίγει πληκτρολογούμε το αρχείο σε γλώσσα Assembly που περιγράφει την εφαρμογή μας. Το παρακάτω πρόγραμμα μεταφέρει τιμές στον καταχωρητή εργασίας και σε θέσεις μνήμης και εκτελεί απλές αριθμητικές πράξεις.

```
#include "p16F877.inc"

        ;Initialization

        Org 0                   ;Start from program memory address 0

        movlw b'00001010'       ;Load in w decimal value '10'

        movwf 22h               ;Trasfer w value to memory 22h

        movlw b'00000101’       ;Load in w decimal '5'

        movwf 23h               ;Trasfer w value to memory 23h

        ;main

        movf 22h,w              ;Transfer content of memory 22h to w

        addwf 23h,w             ;Add w with memory 23h

        movwf 24h               ;Transfer result to memory 24h

        end
```

Μετά την συγγραφή του κώδικα και έχοντας επιλεγμένο το παράθυρο του συντάκτη (η γραμμή τίτλου του παραθύρου να είναι έντονη) από την επιλογή Αποθήκευση (Save) αποθηκεύουμε το αρχείο του κώδικα assembly με την προέκταση .asm. στον ίδιο φάκελο που έχει αποθηκευτεί και το αρχείο έργου (project).

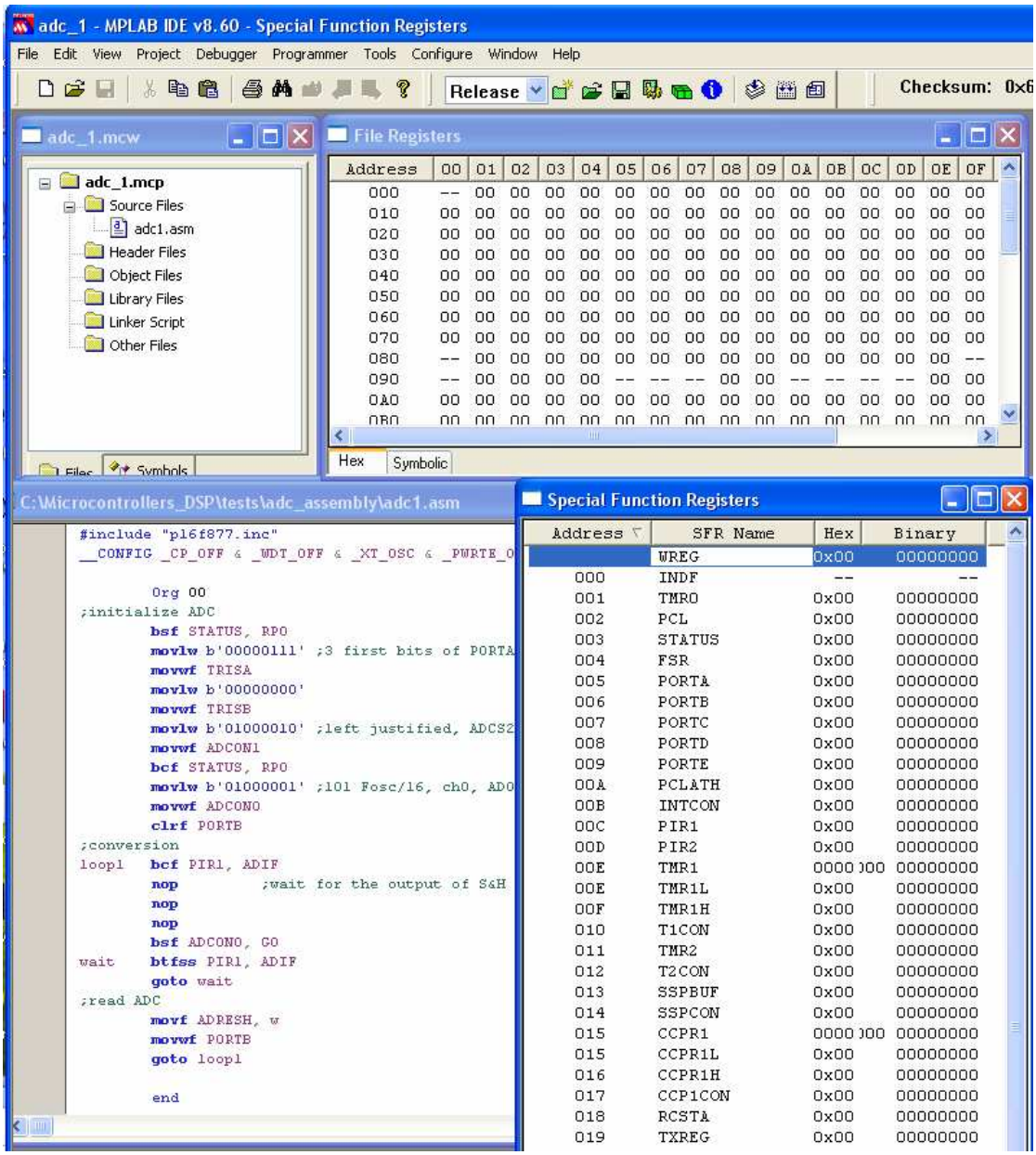
Το επόμενο βήμα είναι να ενημερώσουμε το αρχείο έργου με τα αρχεία κώδικα που θα περιλαμβάνει. Αυτό γίνεται κάνοντας διαδοχικά δεξί κλίκ πάνω στην επιλογή Source Files του παραθύρου project και στην συνέχεια επιλέγοντας την επιλογή Add Files. Στο παράθυρο που εμφανίζεται επιλέγουμε το ήδη αποθηκευμένο αρχείο κώδικα (.asm) που έχουμε από πριν δημιουργήσει σύμφωνα με το παραπάνω βήμα.

Βήμα 3ο Δημιουργία δεκαεξαδικού αρχείου (.hex)

Επιλέξτε Project-Build all. Θα προκύψουν πιθανά λάθη. Όταν διορθωθούν τα λάθη θα δημιουργηθεί το τελικό αρχείο προγραμματισμού.

Βήμα 4ο. Προσομοίωση

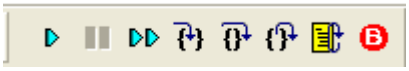
Από την επιλογή View, ανοίξτε τα παράθυρα File Registers και Special Function Registers, όπως φαίνεται στην παρακάτω οθόνη.



Εικόνα Π.8 Παράθυρα για την προσομοίωση του κώδικα

Το παράθυρο File Registers εμφανίζει τα περιεχόμενα της μνήμης RAM του μικροελεγκτή. Το επιλέγουμε πατώντας το πλήκτρο RAM στη γραμμή εργαλείων. Το παράθυρο SFR εμφανίζει τις τιμές των βασικών καταχωρητών ειδικού σκοπού. Κάθε φορά που αλλάζει τιμή ένας καταχωρητής ειδικού σκοπού το περιεχόμενό του εμφανίζεται με κόκκινο.

Επιλέξτε Debugger-Select tool-MPLAB SIM. Στη συνέχεια επιλέγοντας Debugger-Step into, η εκτέλεση του κώδικα προσομοιώνεται εντολή προς εντολή. Το ίδιο αποτέλεσμα,



πετυχαίνουμε με τα εργαλεία προσομοίωσης. Κάθε φορά μπορούμε να μεταβούμε στην εκτέλεση της επόμενης εντολής πατώντας το εικονίδιο 

Παρατηρείστε πως αλλάζουν τιμές οι καταχωρητές και οι θέσεις μνήμης.

Εναλλακτικά, μπορείτε να δημιουργήσετε ένα παράθυρο παρακολούθησης (Watch Window), από το μενού View. Εκεί, επιλέξτε τους καταχωρητές (SFR) που θέλετε να παρακολουθήσετε και προσθέστε τους (add) στο παράθυρο. Με δεξί κλικ στις τιμές των καταχωρητών (values) μπορείτε να επιλέξετε να παρακολουθείτε και τις δυαδικές τιμές των καταχωρητών (binary values).

ΒΙΒΛΙΟΓΡΑΦΙΑ

Στη συγγραφή των παραπάνω σημειώσεων χρησιμοποιήθηκε η παρακάτω βιβλιογραφία, που προτείνεται και για περαιτέρω μελέτη:

1. Jane Liu, *Real-time Systems*, Prentice-Hall, Upper Saddle River, NJ, 2000.

2. Ι. Καλόμοιρος, Σ. Μπουλταδάκης, Ι. Πεταλάς, *Έλεγχος Κυκλωμάτων και Μετρήσεων με Η/Υ*, Εκδόσεις Τζιόλα, Θεσσαλονίκη, 2000.

3. Tim Wilmhurst, *An introduction to the design of small-scale embedded systems*, Palgrave, 2001.

4. Tim Wilmhurst, *Designining Embedded Systems with PIC Microcontrollers, Principles and Applications*, Elsevier-Newnes, 2007.

5. Keith E. Curtis, *Embedded Multitasking with small microcontrollers*, Elsevier-Newnes, 2006

6. David E. Simon, *An Embedded Software Primer*, Addison-Wesley, 1999.

7. Παύλος Χατζηγιαννάκογλου, *Ανάπτυξη Λειτουργικού Συστήματος*, Μεταπτυχιακή Εργασία, Παν. Μακεδονίας, 2009.

8. Σ. Μπουλταδάκης, Ι. Καλόμοιρος, *Υλικό και Λογισμικό Μετρήσεων, Παραδείγματα & Εφαρμογές*, Εκδόσεις Τζιόλα, Θεσσαλονίκη, 2009.