ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

Πτυχιακή εργασία

Υλοποίηση εργαστηριακών ασκήσεων του μαθήματος «Αρχιτεκτονικής Η/Υ» με το μΕ AVR και χρήση του μΕ στην κατασκευή χρονισμού προειδοποιητικού λαμπτήρα

Τσιρογιάννης Π. Νικόλαος, ΑΕΜ: 1403

Μαργαρίτης Κ. Βασίλειος, ΑΕΜ: 1314

Επιβλέποντες καθηγητές: Σπυρίδων Καζαρλής, Θωμάς Μάντζου

Σέρρες, Μάιος 2014

Δηλώνουμε υπεύθυνα ότι η παρούσα πτυχιακή εργασία αποτελεί προϊόν προσωπικής μας μελέτης και εργασίας και πως όλες οι πηγές που χρησιμοποιήθηκαν για τη συγγραφή της δηλώνονται σαφώς στη βιβλιογραφία – δικτυογραφία. Επίσης, γνωρίζουμε πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαστε ενήμεροι για την επέλευση των νομίμων συνεπειών.

Ευχαριστίες

Από τη θέση μας, θα θέλαμε να ευχαριστήσουμε θερμά τους δύο επιβλέποντες καθηγητές της συγκεκριμένης πτυχιακής εργασίας. Συγκεκριμένα, τον κ. Θωμά Μάντζου για την εμπιστοσύνη που έδειξε στο πρόσωπό μας με την ανάθεση της πτυχιακής εργασίας και για τις πολύτιμες συμβουλές του κατά το αρχικό στάδιο της εκπόνησής της. Επίσης, θα θέλαμε να ευχαριστήσουμε και τον κύριο Σπυρίδωνα Καζαρλή, το νέο επιβλέποντα καθηγητή, για την πολύτιμη βοήθειά του στη συμπλήρωση και ολοκλήρωση της εργασίας. Τους ευχαριστούμε για το χρόνο που αφιέρωσαν, καθώς και για την υπομονή που επέδειξαν κατά τη διάρκεια της εκπόνησης και της ολοκλήρωσης της εργασίας.

Πίνακας περιεχομένων

Πρόλογος

Ο κλάδος των μικροελεγκτών – μικροϋπολογιστών είναι ένα πεδίο το οποίο αναπτύσσεται με πολύ μεγάλη ταχύτητα και έχει πληθώρα εφαρμογών στην καθημερινή ζωή λόγω του χαμηλού κόστους, του μικρού όγκου και, κυρίως, της υψηλής ευελιξίας τους.

Στόχος της συγκεκριμένης πτυχιακής εργασίας είναι η γνωριμία με τους μικροελεγκτές AVR, η περιγραφή της πλακέτας EvB 4.3 v4, η οποία αποτελείται από το μικροελεγκτή ATMega644P, και η αναλυτική περιγραφή των στοιχείων της πλακέτας. Επίσης, επιδιώκει την περιγραφή του μικροελεγκτή ATMega644P, την υλοποίηση ασκήσεων του εργαστηριακού μαθήματος «Αρχιτεκτονική Η/Υ», καθώς και τη σχεδίαση και εφαρμογή παραδειγμάτων με τους λαμπτήρες με την ταυτόχρονη εμφάνιση μηνυμάτων στην οθόνη.

Στην παρούσα πτυχιακή εργασία τα εργαλεία που χρησιμοποιήσαμε χρησιμοποιήθηκαν είναι τα εξής: ένας ηλεκτρονικός υπολογιστής, η πλακέτα EvB 4.3 v4, δέκα καλώδια για τη σύνδεση των ακροδεκτών της πλακέτας και ένα καλώδιο USB για τη σύνδεση της πλακέτας με τον ηλεκτρονικό υπολογιστή.

Για την εφαρμογή των εργαστηριακών ασκήσεων χρησιμοποιήθηκε το λογισμικό της Atmel, το Atmel Studio 6.1 και οι ασκήσεις πραγματοποιήθηκαν σε γλώσσα Assembly. Επίσης, για την εξαγωγή των αποτελεσμάτων των εργαστηριακών ασκήσεων στους λαμπτήρες της πλακέτας, χρησιμοποιήθηκε και το λογισμικό avrdude, ενώ η εφαρμογή του δεύτερου παραδείγματος πραγματοποιήθηκε στο περιβάλλον ανάπτυξης του Arduino, το οποίο ως γλώσσα προγραμματισμού χρησιμοποιεί τη γλώσσα Wiring, που αποτελεί μία παραλλαγή της C/C++.

Κεφάλαιο 1

Εισαγωγή στους μικροελεγκτές

1.1 Ορισμός του «μικροελεγκτή»

Ο «μικροελεγκτής» (microcontroller) αποτελεί ένα είδος επεξεργαστή και πρόκειται για παραλλαγή του μικροεπεξεργαστή. Διαθέτει πολλά ενσωματωμένα υποσυστήματα και η λειτουργία του μπορεί να πραγματοποιηθεί με πολύ λίγα εξωτερικά εξαρτήματα. Συνήθως, χρησιμοποιείται στα ενσωματωμένα συστήματα (embedded systems) ελέγχου χαμηλού και μεσαίου κόστους, όπως είναι αυτά που χρησιμοποιούνται σε ηλεκτρικές συσκευές, σε ηλεκτρονικά προϊόντα (από τις ψηφιακές φωτογραφικές μηχανές έως σε παιχνίδια), αυτοματισμούς και κάθε είδους αυτοκινούμενα τροχοφόρα οχήματα. Πρόκειται για ένα ολοκληρωμένο κύκλωμα το οποίο προγραμματίζεται, διαθέτει μνήμη, διαφόρων ειδών περιφερειακά κυκλώματα επεξεργαστή και, προκειμένου να επικοινωνήσει με εξωτερικές συσκευές, διαθέτει και θύρες εισόδου – εξόδου.

Επιπλέον, όπως ένας μικροϋπολογιστής έχει τη δυνατότητα να εκτελεί διάφορα προγράμματα και να διαθέτει περιφερειακές συσκευές, επεξεργαστή και μνήμη, έτσι και ο μικροελεγκτής διαθέτει τα χαρακτηριστικά που προαναφέρθηκαν και, μάλιστα, σε ένα μόνο chip. Τέλος, η αποθήκευση των προγραμμάτων που εκτελούν οι μικροελεγκτές γίνεται πάντα στη μνήμη προγράμματος.

1.2 AVR Μικροελεγκτές

Χρήση των μικροελεγκτών γίνεται, κυρίως, στα ηλεκτρονικά. Ο σχεδιασμός πλακετών στις εφαρμογές γίνεται πιο απλός και πιο εύκολος, διότι, κατά τη διάρκειά του, ενσωματώνονται πολλές και διάφορες λειτουργίες. Επίσης, για τον προγραμματισμό και την ανάπτυξη διαφόρων εφαρμογών υπάρχουν ποικίλα εργαλεία, καθώς και αρκετοί διαφορετικοί τύποι μικροελεγκτών. Όποιος ενδιαφέρεται να ασχοληθεί με τους μικροελεγκτές AVR έχει τη συγκεκριμένη δυνατότητα, καθώς, εκτός του ότι οι μικροελεγκτές της Atmel έχουν χαμηλό κόστος, υπάρχουν και πολλά εργαλεία που διατίθενται για το συγκεκριμένο σκοπό και τα οποία είναι ελεύθερα στην αγορά.

1.3 Αρχιτεκτονική μικροελεγκτών AVR

- Data RAM:

  Οι μνήμες EEPROM, SRAM και Flash ενσωματώνονται σε ένα chip. Αυτό έχει ως αποτέλεσμα να μην απαιτείται επιπλέον εξωτερική μνήμη. Επίσης, για την προσθήκη συμπληρωματικών chip μνήμης, ορισμένες συσκευές διαθέτουν και έναν εξωτερικό δίαυλο.

- Μνήμη προγράμματος (Flash):

  Στη Flash memory αποθηκεύονται όλες οι εντολές του προγράμματος. Η ονοματολογία του μικροελεγκτή περιέχει το μέγεθος της μνήμης του προγράμματος. Για παράδειγμα, ο μικροελεγκτής ATmega64x έχει μνήμη προγράμματος 64kb. Για τα προγράμματα τα οποία βρίσκονται σε κάποια εξωτερική μνήμη δεν υπάρχει κάποια πρόβλεψη. Αντίθετα, οι κώδικες εκτελούνται από τον πυρήνα του μικροελεγκτή και μεταφέρονται στην ενσωματωμένη μνήμη flash.

- Εσωτερική μνήμη δεδομένων:

  Η διευθυνσιοδότηση των δεδομένων αποτελείται από τους καταχωρητές εισόδου – εξόδου (I/O), τη μνήμη SRAM και από το αρχείο καταχωρητών.

1.4 Κατασκευαστές μικροελεγκτών

Όσον αφορά του κατασκευαστές μικροελεγκτών, αυτοί είναι οι εξής:

- Atmel
- Microchip
- Hitachi
- Texas Instruments
- NEC
- Toshiba
- Freescale Semiconductor (πρώην Motorola)
- Maxim (μετά την εξαγορά της Dallas)
- Epson
- ARM (απλά παραχωρεί τα δικαιώματα της χρήσης του πυρήνα, δεν κατασκευάζει μικροελεγκτές)

1.5 Διαφορές από το μικροεπεξεργαστή

Στους σύγχρονους μικροεπεξεργαστές για μη ενσωματωμένα συστήματα, όπως για παράδειγμα στους μικροεπεξεργαστές των προσωπικών ηλεκτρονικών υπολογιστών, αυτό το οποίο συμβαίνει είναι το εξής: αποδίδεται βαρύτητα στην υπολογιστική ισχύ και υπάρχει μεγάλη ευελιξία για τη δημιουργία πολλών και διαφορετικών εφαρμογών, αφού καθορίζεται η λειτουργικότητα του τελικού συστήματος από τα εξωτερικά περιφερειακά, τα οποία συνδέονται με την κεντρική μονάδα, δηλαδή με το μικροεπεξεργαστή.

Αντιθέτως, αναφορικά με τους μικροεπεξεργαστές που αφορούν ενσωματωμένα συστήματα, όπως για παράδειγμα τους μικροελεγκτές, τα χαρακτηριστικά γνωρίσματα είναι:

- ✓ η δυνατότητα για συνεργασία των μικροεπεξεργαστών με εξωτερικά περιφερειακά είναι πολύ μικρότερη έως και μηδαμινή
- ✓ υπάρχει λιγότερη ευελιξία
- ✓ υπάρχει περιορισμένη υπολογιστική ισχύς
- ✓ δίνεται βαρύτητα από τους μικροελεγκτές στο χαμηλό κόστος και στην εξειδίκευση
- ✓ προκειμένου μία συσκευή να λειτουργήσει, δίνεται έμφαση από τους μικροελεγκτές στον αριθμό των ολοκληρωμένων κυκλωμάτων που απαιτούνται και ο αριθμός αυτών των κυκλωμάτων είναι μικρός

Αναλυτικότερα, τα πλεονεκτήματα των μικροελεγκτών είναι:

- ➢ Αυτονομία. Η αυτονομία επιτυγχάνεται μέσα από την ενσωμάτωση ορισμένων σύνθετων περιφερειακών υποσυστημάτων, όπως είναι οι θύρες επικοινωνίας και οι μνήμες. Επομένως, για τη λειτουργία αρκετών μικροελεγκτών δεν απαιτείται κάποιο άλλο ολοκληρωμένο κύκλωμα.
- ➢ Η υλοποίηση των εφαρμογών καθίσταται περισσότερο εύκολη. Επίσης, με την ενσωμάτωση περιφερειακών επιτυγχάνονται πιο απλές διασυνδέσεις.
- ➢ Απαιτείται χαμηλότερη κατανάλωση ισχύος.
- ➢ Μειώνεται το κόστος της συσκευής στην οποία ενσωματώνεται ο μικροελεγκτής.
- ➢ Εξαιτίας της ύπαρξης λιγότερων διασυνδέσεων, υπάρχει μεγαλύτερη αξιοπιστία.

- Οι εκπομπές ηλεκτρομαγνητικών παρεμβολών είναι μειωμένες, όπως επίσης μειωμένη είναι και η ευαισθησία σε ανάλογες παρεμβολές που προέρχονται από άλλες ηλεκτρονικές αλλά και ηλεκτρικές συσκευές. Οι ταχύτητες λειτουργίας, οι οποίες είναι χαμηλές, το μικρό μήκος των εξωτερικών διασυνδέσεων, αλλά και ο μικρός αριθμός των εξωτερικών διασυνδέσεων οδηγούν στο συγκεκριμένο πλεονέκτημα.
- Για ψηφιακές εισόδους – εξόδους (αν το μέγεθος του ολοκληρωμένου κυκλώματος είναι δεδομένο), διατίθενται περισσότεροι ακροδέκτες κι αυτό συμβαίνει, επειδή οι ακροδέκτες δεν δεσμεύονται σε περίπτωση σύνδεσης με εξωτερικά περιφερειακά.
- Για το συνολικό υπολογιστικό σύστημα το μέγεθος είναι μικρό.

Οι μικροελεγκτές διαθέτουν παρόμοια βασική αρχιτεκτονική μ' αυτή των κοινών μικροεπεξεργαστών. Συνεπώς, η αρχιτεκτονική τους δεν διαφέρει, αν και στους μικροελεγκτές συχνά υπάρχει η αρχιτεκτονική μνήμης τύπου Harvard. Αντίθετα, στους κοινούς μικροεπεξεργαστές η διάταξη μνήμης που εφαρμόζεται συχνότερα είναι τύπου Von Neumann.

1.6 Συνήθη υποσυστήματα

Το ολοκληρωμένο κύκλωμα που συγκροτεί το μικροεπεξεργαστή περιλαμβάνει: στοιχειώδεις καταχωρητές (registers), προσωρινή μνήμη RAM μεγάλης ταχύτητας (cache memory), τη Λογική και Αριθμητική Μονάδα (ALU) και, σε ορισμένες περιπτώσεις, τον ελεγκτή μνήμης (memory controller). Προκειμένου να λειτουργήσει ένα πλήρως ενσωματωμένο υπολογιστικό σύστημα, προϋποτίθενται αρκετά εξωτερικά υποσυστήματα και περιφερειακά, όπως:

- Κύκλωμα συνδετικής λογικής (glue logic) για την επίτευξη σύνδεσης των εξωτερικών μνημών και άλλων περιφερειακών παράλληλης σύνδεσης στην αρτηρία δεδομένων (bus) του επεξεργαστή.
- Μνήμη προγράμματος (τύπου EPROM, FLASH, ROM κ.ά.), που περιλαμβάνει το λογισμικό του συστήματος. Σε ορισμένα μοντέλα, μετά την εγγραφή της μνήμης, υπάρχει η δυνατότητα να κλειδώνει η μνήμη για την προστασία του περιεχομένου της από περιπτώσεις αντιγραφής.
- Μεγάλο μέγεθος μνήμης RAM.

- ❖ Μόνιμη μνήμη για την αποθήκευση των παραμέτρων λειτουργίας (τύπου EEPROM ή NVRAM). Η συγκεκριμένη μνήμη να μπορεί να εγγράφεται στον πυρήνα του μικροελεγκτή. Επίσης, υπάρχει η δυνατότητα να διαγράφεται και να εγγράφεται οποιοδήποτε μεμονωμένο byte. Αυτή η δυνατότητα δεν παρέχεται από τη μνήμη Flash.
- ❖ Reset – κύκλωμα αρχικοποίησης.
- ❖ Interrupt request controller – διαχειριστής αιτήσεων διακοπής από τα περιφερειακά.
- ❖ Brown – out detection – κύκλωμα επιτήρησης τροφοδοσίας. Το συγκεκριμένο κύκλωμα επιβλέπει την τροφοδοσία και, σε περίπτωση που η τροφοδοσία πέσει σε κατώτερο σημείο από τα ανεκτά όρια, πραγματοποιείται αρχικοποίηση όλου του συστήματος. Με την αρχικοποίηση αυτή δεν αλλοιώνονται τα δεδομένα.
- ❖ Watch dog timer – κύκλωμα επιτήρησης λειτουργίας. Το συγκεκριμένο κύκλωμα αρχικοποιεί το σύστημα, αν λόγω κολλήματος (hang) εμφανιστούν σημάδια δυσλειτουργίας.
- ❖ Για την παροχή παλμών χρονισμού (clock) υπάρχει ο τοπικός ταλαντωτής.
- ❖ Ένας ή περισσότεροι χρονιστές – απαριθμητές υψηλής ταχύτητας (hard ware timer – counter). Οι χρονιστές – απαριθμητές χρησιμεύουν στη μέτρηση διάρκειας γεγονότων, στη δημιουργία καθυστερήσεων, στην απαρίθμηση γεγονότων και άλλων λειτουργιών ακριβούς χρονισμού.
- ❖ Real Time Clock, RTC – ρολόι πραγματικού χρόνου. Η τροφοδοσία του παρέχεται από μία ανεξάρτητη μπαταρία και γι' αυτό η κατανάλωση ρεύματος πρέπει να είναι αρκετά χαμηλή.
- ❖ Parallel Input – Output, PIO – σειρά ανεξάρτητων ψηφιακών εισόδων και εξόδων.

Όλα τα είδη των μικροελεγκτών περιέχουν τα περισσότερα από τα περιφερειακά που προαναφέρθηκαν. Υπάρχουν, όμως, ορισμένες διαφορές οι οποίες αφορούν την ύπαρξη ή μη εσωτερικής μνήμης προγράμματος και το είδος της συγκεκριμένης μνήμης. Επομένως, υπάρχουν:

- ✓ Μικροελεγκτές οι οποίοι δεν περιέχουν μνήμη προγράμματος και χαρακτηρίζονται ως ROM – less. Οι συγκεκριμένοι μικροελεγκτές παρέχουν μία παράλληλη αρτηρία (bus) δεδομένων, όπου πραγματοποιείται η σύνδεση εξωτερικών μνημών προγράμματος και RAM. Αυτά τα είδη μικροελεγκτών

προορίζονται, συνήθως, για υπολογιστικά συστήματα ελέγχου τα οποία είναι ισχυρότερα και οι απαιτήσεις της μνήμης μεγαλύτερες.

- ✓ Μικροελεγκτές οι οποίοι περιέχουν μνήμη ROM. Η συγκεκριμένη μνήμη κατασκευάζεται με το λογισμικό της (Mask ROM) ή γράφεται μόνο μία φορά (One Time Programmable, OTP). Όταν αγοράζονται σε μεγάλες ποσότητες, παρέχεται η δυνατότητα διάθεσής τους σε ιδιαίτερα χαμηλό κόστος.
- ✓ Μικροελεγκτές οι οποίοι διαθέτουν μνήμη FLASH και μπορούν να προγραμματιστούν πολλές φορές. Αυτή η κατηγορία μικροελεγκτών είναι και η πιο διαδεδομένη. Στις περισσότερες περιπτώσεις, ο προγραμματισμός της μνήμης πραγματοποιείται ακόμη και πάνω στο κύκλωμα της ίδιας της ενσωματωμένης (embedded) εφαρμογής (δυνατότητα In Circuit Programming, ISP). Οι συγκεκριμένοι μικροελεγκτές έχουν αντικαταστήσει τους παλαιότερους τύπους EPROM, οι οποίοι έσβηναν με υπεριώδη ακτινοβολία.

1.7 Πρόσθετες λειτουργίες

Το περιεχόμενο των μικροελεγκτών εξαρτάται από τον τρόπο χρήσης τους. Επομένως, ανάλογα με τις εφαρμογές τους, οι μικροελεγκτές είναι δυνατό να περιλαμβάνουν:

i. Universal Asynchronous Receiver Transmitter, UART – μία ή περισσότερες ασύγχρονες σειριακές θύρες επικοινωνίας.
ii. Σύγχρονες σειριακές θύρες επικοινωνίας (I2C, SPI, Ethernet).
iii. Ολόκληρα υποσυστήματα για την άμεση υποστήριξη από υλικό λογισμικό (firmware) των πιο σύνθετων πρωτοκόλλων επικοινωνίας, όπως CAN, HDLC, ISDN, ADSL.
iv. Floating Point Processing Unit, FPU – μονάδα άμεσης εκτέλεσης πράξεων κινητής υποδιαστολής, η οποία είναι πιο γρήγορη από τη Λογική και Αριθμητική μονάδα (ALU) του επεξεργαστή. Μικροελεγκτές που έχουν δυνατότητες ψηφιακής επεξεργασίας σήματος (Digital Signal Processing, DSP) χαρακτηρίζονται από μονάδες τέτοιου είδους.
v. Digital to Analog converter, DAC – μετατροπέα ψηφιακού σήματος σε αναλογικό σήμα.
vi. Εισόδους για τη μετατροπή του αναλογικού σήματος σε ψηφιακό σήμα (Analog to Digital converter, ADC).

vii. Liquid Crystal Display, LCD – ελεγκτή οθόνης υγρών κρυστάλλων.

viii. Υποσύστημα προγραμματισμού πάνω στο κύκλωμα (τύπου ISP). Εξαιτίας του συγκεκριμένου κυκλώματος, υπάρχει η δυνατότητα του επαναπρογραμματισμού (αναβάθμιση λογισμικού) της εφαρμογής, συνδέοντας στη συσκευή μία εξωτερική συσκευή προγραμματισμού (συνήθως σε θύρα UART RS-232) ή ακόμη και από το διαδίκτυο. Αυτή η δυνατότητα δεν είναι δυνατό να πραγματοποιηθεί σε άδεια μνήμη, καθώς προϋποτίθεται η ύπαρξη λογισμικού υποδοχής (boot strap) μέσα στη μνήμη προγράμματος.

ix. Υποσύστημα προγραμματισμού (τύπου ISP) και διάγνωσης (συνήθως είναι το καθιερωμένο πρότυπο JTAG). Εξαιτίας του συγκεκριμένου υποσυστήματος προγραμματισμού, υπάρχει η δυνατότητα η μνήμη προγράμματος να προγραμματίζεται, χωρίς να χρειάζεται κάποιο πρόγραμμα υποδοχής. Γι' αυτό και είναι χρήσιμο κατά τη διάρκεια του αρχικού προγραμματισμού, όπως για παράδειγμα κατά τη διάρκεια της συναρμολόγησης, ή σε περίπτωση που υπάρχει σφάλμα (bug) στο λογισμικό υποδοχής, που να καθιστά ανέφικτη την αναβάθμιση με κανονικό τρόπο.

1.8 Διαδεδομένες κατηγορίες μικροελεγκτών

Εξαιτίας του ισχυρού ανταγωνισμού που υπάρχει, αλλά και της τάσης που επικρατεί για την ενσωμάτωση των μικροελεγκτών σε ηλεκτρικές και ηλεκτρονικές συσκευές, οι κατασκευαστές μικροελεγκτών δημιουργούν ανταγωνιστικά μοντέλα μαζικής παραγωγής, καθώς και μικροελεγκτές οι οποίοι χρησιμοποιούνται σε εφαρμογές που είναι περισσότερο εξειδικευμένες. Οι κατηγορίες στις οποίες διακρίνονται οι μικροελεγκτές είναι:

- Μικροελεγκτές των 4-bit αλλά, συνήθως, των 8-bit. Αυτοί οι μικροελεγκτές έχουν χαμηλό κόστος, είναι γενικής χρήσης και οι ακροδέκτες που διαθέτουν είναι λίγοι (υπάρχει περίπτωση οι ακροδέκτες να είναι και λιγότεροι από 8). Ο σχεδιασμός των συγκεκριμένων μικροελεγκτών βασίζεται στη χαμηλή κατανάλωση ισχύος και στην αυτάρκεια, ώστε να μην χρειάζονται πολλά εξωτερικά εξαρτήματα (σε μερικές περιπτώσεις μπορεί να μην χρειάζονται και καθόλου εξωτερικά εξαρτήματα) και το εσωτερικό λογισμικό τους να μην μπορεί να αντιγραφεί με ευκολία. Επιπλέον, δεν παρέχεται η δυνατότητα για περαιτέρω επέκταση της μνήμης τους και ορισμένα μοντέλα είναι ιδιαίτερα

γνωστά σε ερασιτέχνες ηλεκτρονικούς, όπως είναι οι περισσότεροι μικροελεγκτές των σειρών PIC (της Microchip), AVR (της Atmel) και 8051 (της Intel, της Atmel, της Dallas κ.ά.).

- Μικροελεγκτές οι οποίοι μπορεί να είναι των 8-bit, των 16-bit ή ακόμη και των 32-bit. Οι συγκεκριμένοι μικροελεγκτές είναι χαμηλού κόστους, γενικής χρήσης και με μέτριο έως σχετικά μεγάλο αριθμό ακροδεκτών. Επίσης, διαθέτουν μεγάλο αριθμό κοινών περιφερειακών, όπως είναι οι θύρες UART, I2C, SPI ή CAN, και μετατροπείς για τη μετατροπή του σήματος από αναλογικό σε ψηφιακό σήμα και το αντίστροφο. Στους κατασκευαστές της Άπω Ανατολής (Ιαπωνία, Κορέα) είναι συνηθισμένη και η ενσωμάτωση ελεγκτών οθόνης υγρών κρυστάλλων και πληκτρολογίου. Σε ορισμένες περιπτώσεις, υπάρχει και η δυνατότητα επέκτασης της μνήμης τους.
- Μικροελεγκτές των 32-bit. Οι συγκεκριμένοι μικροελεγκτές είναι μικροελεγκτές μέσου κόστους, γενικής χρήσης και διαθέτουν αρκετά μεγάλο αριθμό ακροδεκτών. Σ' αυτούς τους μικροελεγκτές δίνεται έμφαση στην ταχύτητα εκτέλεσης εντολών, στην υψηλή αυτάρκεια περιφερειακών και στις μεγάλες δυνατότητες εσωτερικής ή εξωτερικής μνήμης προγράμματος (FLASH) και RAM. Στην κατηγορία αυτή εμφανίζονται οι αρχιτεκτονικές που διαθέτουν υψηλή μεταφερσιμότητα λογισμικού (port ability) από τον ένα κατασκευαστή στον άλλο, όπως για παράδειγμα ανάμεσα στους μικροελεγκτές τύπου ARM ή MIPS, όπου το σύνολο των βασικών εντολών που αναγνωρίζει η Λογική και Αριθμητική μονάδα (ALU) είναι το ίδιο. Αυτό έχει ως αποτέλεσμα να μειώνονται οι πολλές αλλαγές στο λογισμικό, όταν στο μέλλον κάποιος θελήσει να χρησιμοποιήσει ένα διαφορετικό μικροελεγκτή κάποιου άλλου κατασκευαστή, με την απαραίτητη προϋπόθεση και ο επόμενος μικροελεγκτής να υποστηρίζει εξίσου και αντίστοιχα το σύνολο εντολών ARM ή MIPS.
- Μικροελεγκτές εξειδικευμένων εφαρμογών. Οι συγκεκριμένοι μικροελεγκτές ενσωματώνουν ένα εξειδικευμένο πρωτόκολλο επικοινωνίας, που υλοποιείται πάντα σε hardware. Χρήση τέτοιων μικροελεγκτών γίνεται σε τηλεπικοινωνιακές συσκευές, όπως είναι τα μόντεμ.

Οι μικροελεγκτές που πωλούνται περισσότερο είναι οι μικροελεγκτές των 8-bit, αφού κοστίζουν ελάχιστα και διαθέτουν μικρότερο μέγεθος λογισμικού για την

επίτευξη του ίδιου αποτελέσματος. Αυτό συμβαίνει, διότι τα σύγχρονα μοντέλα των μικροελεγκτών των 8-bit αποδίδουν καλύτερα συγκριτικά με ό,τι ίσχυε στο παρελθόν.

1.9 Γλώσσες προγραμματισμού και εργαλεία ανάπτυξης

Το πόσο επιτυχημένη θα είναι μία κατηγορία μικροελεγκτών εξαρτάται κατά ένα πολύ μεγάλο ποσοστό από τα διαθέσιμα εργαλεία ανάπτυξης, αλλά και από το πόσο εύχρηστα είναι αυτά. Τέτοιου είδους εργαλεία ανάπτυξης είναι οι μεταφραστές από γλώσσες υψηλού επιπέδου σε μία γλώσσα κατανοητή από το μικροελεγκτή (assembly), τα εργαλεία εκσφαλμάτωσης (debuggers) και οι προγραμματιστές της εσωτερικής μνήμης. Τα συγκεκριμένα εργαλεία δεν περιέχουν μόνο το λογισμικό και η αιτία είναι ότι δεν υπάρχει κάποιος τυποποιημένος τρόπος επικοινωνίας με τους μικροελεγκτές. Για τη δημιουργία εργαλείων ανάπτυξης ασχολούνται όχι μόνο εταιρείες που κατασκευάζουν μικροελεγκτές, αλλά και ορισμένες που εξειδικεύονται στον τομέα αυτό.

Οι γλώσσες που χρησιμοποιούνται περισσότερο, και οι οποίες είναι και οι περισσότερο διαδεδομένες, για τον προγραμματισμό των μικροελεγκτών είναι η C, η C++ και οι διάφορες παραλλαγές τους. Επίσης, μπορεί να χρησιμοποιηθεί και η γλώσσα προγραμματισμού assembly. Η χρήση της assembly γίνεται ειδικότερα σε τμήματα του λογισμικού όπου χρειάζεται μεγάλη ταχύτητα ή μικρό μέγεθος της μνήμης που χρησιμοποιείται. Γενικά, όμως, στις περισσότερες εφαρμογές η assembly έχει αντικατασταθεί από τη C, εξαιτίας των μεγαλύτερων απαιτήσεων που υπάρχουν σε λειτουργικότητα, της ευκολίας με την οποία προγραμματίζεται η C, αλλά και λόγω της επαρκέστατης μνήμης που διαθέτουν, πλέον, οι σύγχρονοι μικροελεγκτές.

Κεφάλαιο 2

Η πλακέτα EvB 4.3 v4

2.1 Περιγραφή της πλακέτας

Ο μικροελεγκτής EvB 4.3 είναι ένα σύνολο δρομολόγησης, βασισμένο στους δύο πιο διάσημους μικροεπεξεργαστές της ATML, τον ATMega16 και τον ATMega32. Η πλακέτα είναι εξοπλισμένη με μία σειρά από περιφερειακά στοιχεία, των οποίων οι καταλήξεις είναι συνδεδεμένες με τον ακροδέκτη, κάτι που επιτρέπει στο χρήστη τη γρήγορη υλοποίηση οποιασδήποτε εργασίας. Όλες οι κεφαλές έχουν επισημανθεί, βρίσκονται κοντά και είναι στενά συνδεδεμένες με τα περιφερειακά. Επίσης, παρέχει τη δυνατότητα για διαισθητική σύνδεση των στοιχείων, χωρίς την ανάγκη να διαβάζει κανείς την τεκμηρίωση.

Η πλακέτα EvB 4.3 έχει σχεδιαστεί και για μη έμπειρους χρήστες, οι οποίοι κάνουν τα πρώτα τους βήματα στο χώρο των μικροεπεξεργαστών, αλλά και για επαγγελματίες προγραμματιστές, οι οποίοι αναζητούν μία καθολική πλατφόρμα για τις εργασίες τους.

Οι παλαιότερες εκδόσεις από την πλακέτα EvB 4.3 είχαν εφαρμοστεί με επιτυχία σε μία σειρά μεγάλων εργασιών στα Πανεπιστήμια της Πολωνίας, κατά τη διάρκεια διπλωματικών επιστημονικών ερευνών. Στη σημερινή εποχή, οι πλακέτες χρησιμοποιούνται στα Πανεπιστήμια της περιοχής της Σιλεσίας.

2.2 Χαρακτηριστικά της πλακέτας

- Η πλακέτα EvB 4.3 v4 περιέχει τα παρακάτω χαρακτηριστικά:
  - Επεξεργαστή AVR ATMega644p με σώμα DIP40
  - Ρολόι πραγματικού χρόνου (Real time clock) PCF8583
  - Μνήμη EEPROM AT24C02
  - Δέκτη υπερύθρων TSOP4836
  - Ανιχνευτή θερμοκρασίας DS18B20
  - Μετατροπέα RS485
  - Υποδοχή κάρτας MMC/SD
  - 5 κουμπιά πίεσης
  - 8 leds

- 2 τρανζίστορ με εξόδους 1A το καθένα
- 3 τρανζίστορ με εξόδους 500mA το καθένα
- 2 αναλογικά ποτενσιόμετρα
- μία οθόνη διαστάσεων 4x7
- USB θύρα
- ISP θύρα
- 5 ακροδέκτες της τάσης των +5V
- 5 ακροδέκτες της γείωσης
- οθόνη LCD των 2x16 χαρακτήρων
- ένα σετ με εργαλεία σύνδεσης (10 απλά καλώδια – 10 εκατοστών)

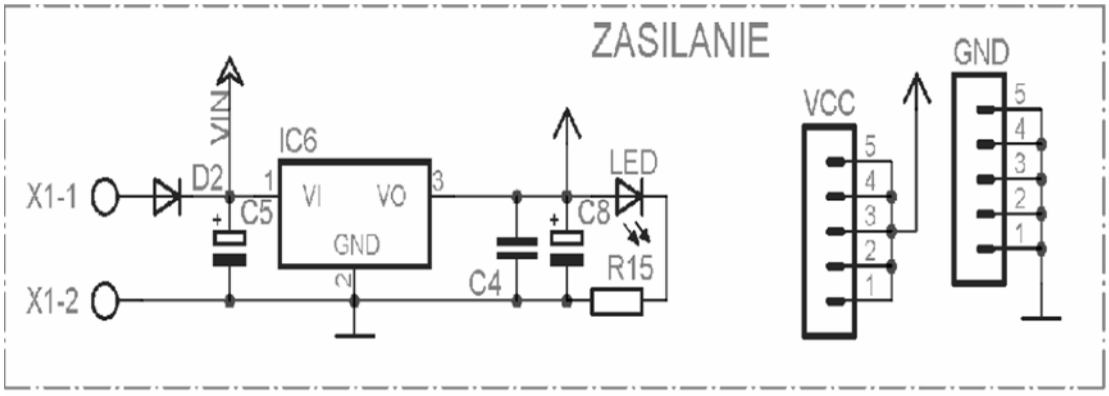
2.3 Τροφοδοτικό

Η πλακέτα EvB 4.3 v4 μπορεί να τροφοδοτείται από:

- τη θύρα USB, χρησιμοποιώντας το USB-Vcc jumper
- το εξωτερικό AC τροφοδοτικό, με ελάχιστη τάση των 9V. Θα πρέπει να είναι συνδεδεμένο με την υποδοχή τροφοδοσίας, ρυθμίζοντας την πόλωση που περιγράφεται στην πλακέτα (σ' αυτή την περίπτωση το USB-Vcc jumper πρέπει να είναι ανοικτό).

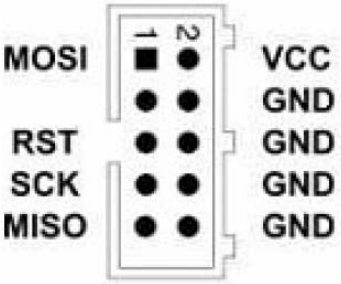
Το πράσινο λαμπάκι (LED), που βρίσκεται πάνω από τη γείωση και τις υποδοχές των +5V, σηματοδοτεί τη σωστή σύνδεση της πηγής ενέργειας.

Η γείωση και οι υποδοχές των +5V που βρίσκονται πάνω στην πλακέτα συνδέονται με το σύνολο και την τάση των +5V.

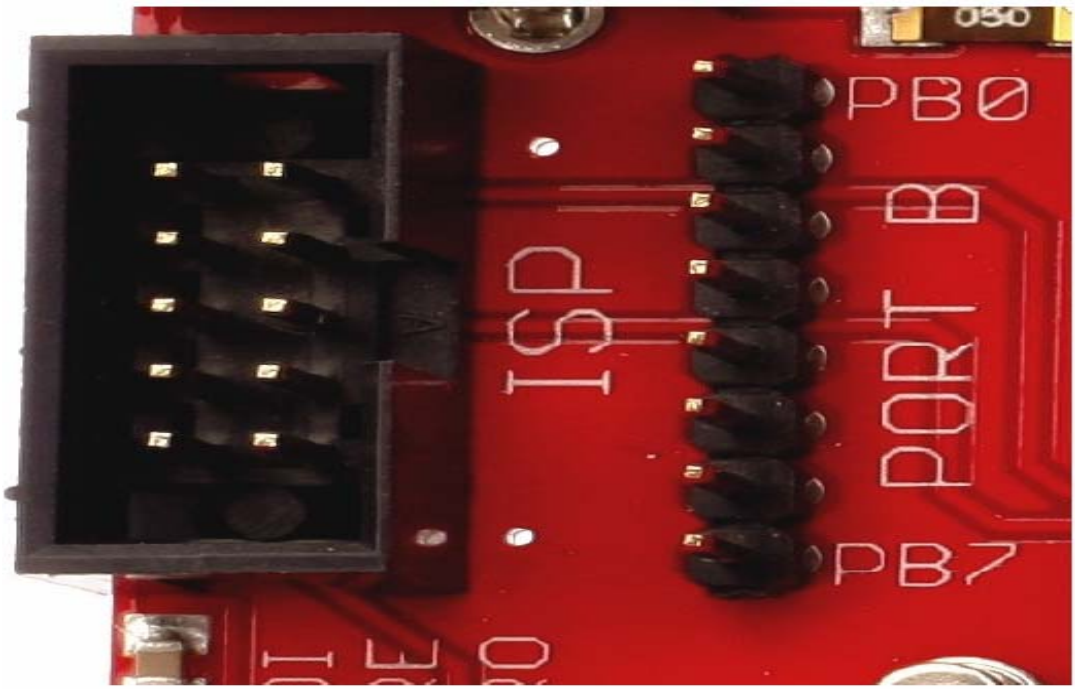


2.4 Υποδοχή προγραμματιστή

Στην πλακέτα EvB 4.3 v4, στο ISP KANDA standard βρίσκεται μία υποδοχή 10 ακροδεκτών, η οποία χρησιμοποιείται για τον προγραμματισμό της πλακέτας. Η υποδοχή αυτή είναι συμβατή με τα περισσότερα προγράμματα προγραμματισμού που διατίθενται στην αγορά, συμπεριλαμβανομένων και των STK200 και AVRProg.



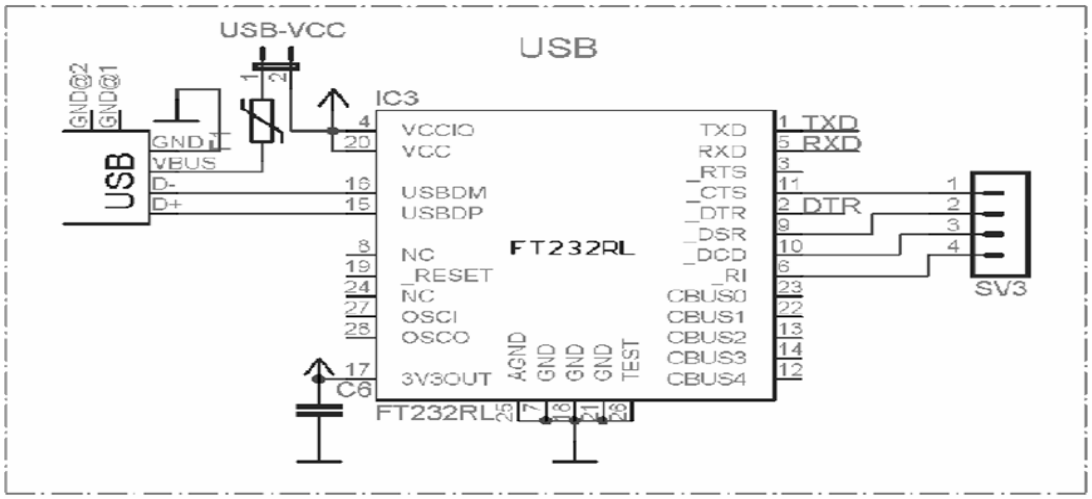
MOSI, MISO, SCK	ISP data way signals
RST	Destination system reset
NC	Not connected
VCC	Destination system voltage



2.5 Θύρα USB

Η επικοινωνία της πλακέτας EvB 4.3 v4 με έναν υπολογιστή έχει σχεδιαστεί με τέτοιο τρόπο, ώστε να χρησιμοποιεί ένα μετατροπέα (USB to UART), το FT232RL [δημιουργεί μία εικονική παράλληλη θύρα (COM)]. Το σύστημα FT232RL είναι συνδεδεμένο με τις γραμμές TXD και RXD του επεξεργαστή. Οι οδηγοί (drivers) για την εικονική παράλληλη θύρα είναι διαθέσιμοι στη διεύθυνση: http://www.ftdichip.com/Drivers/VCP.htm.

Επίσης, το σύστημα FT232RL είναι ένα ολοκληρωμένο κύκλωμα, το οποίο παρέχει σειριακή επικοινωνία με τον υπολογιστή για προγραμματισμό πάνω από τη θύρα USB με τη βοήθεια των ανάλογων FTDI drivers. Οι drivers αυτοί παρέχουν μία ιδεατή θύρα επικοινωνίας με τον υπολογιστή για τους σκοπούς της επικοινωνίας.

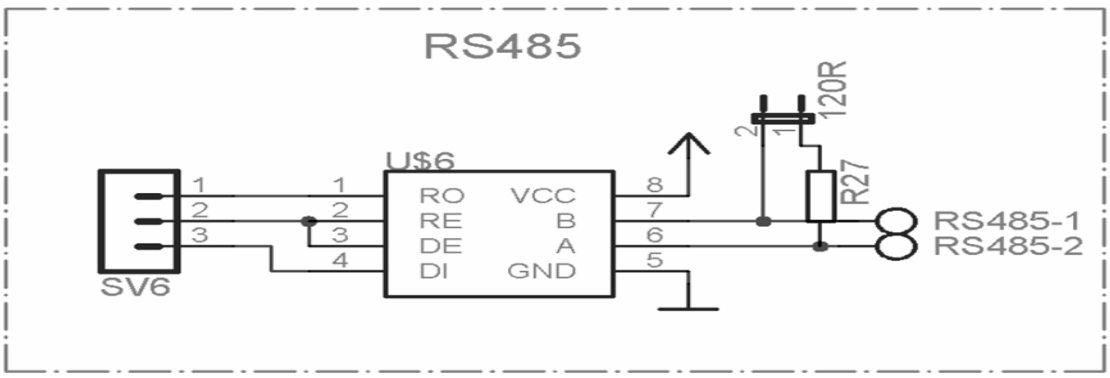


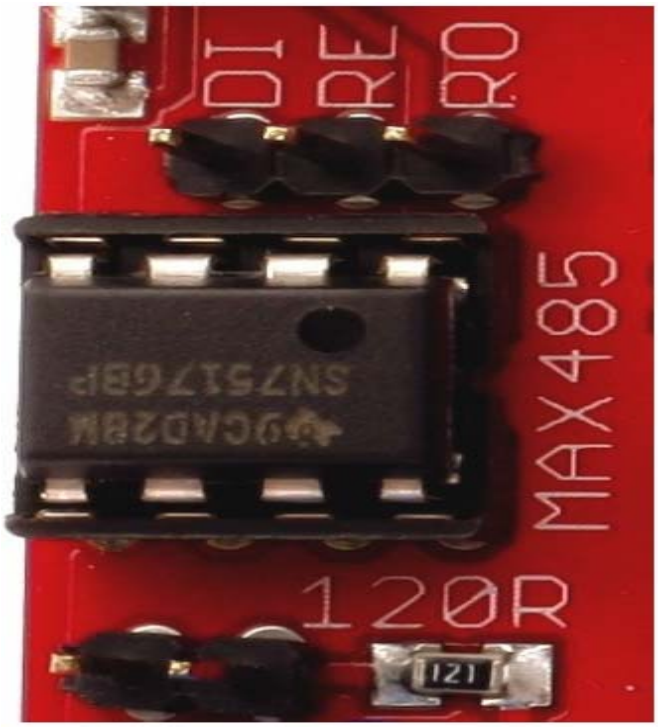
Επιπλέον, οι γραμμές του συστήματος FT232RL προήλθαν από τους ακροδέκτες CTS, DSR, DCD και RI. Αυτές οι γραμμές χρησιμοποιούνται για τον προγραμματισμό του μικροεπεξεργαστή.

Ακροδέκτης	Πλήρης ονομασία – Λειτουργία
TXD	Transmit data – Μεταφορά δεδομένων από τον υπολογιστή
RXD	Receive Data – Λήψη δεδομένων από τον υπολογιστή
CTS	Clear To Send – Έλεγχος ροής
DSR	Data Set Ready – Έτοιμο για επικοινωνία
DCD	Data Carrier Detect – Σύνδεση του modem με άλλη συσκευή
RI	Ring indicator – Κλήση τηλεφωνικής γραμμής

2.6 Θύρα RS485

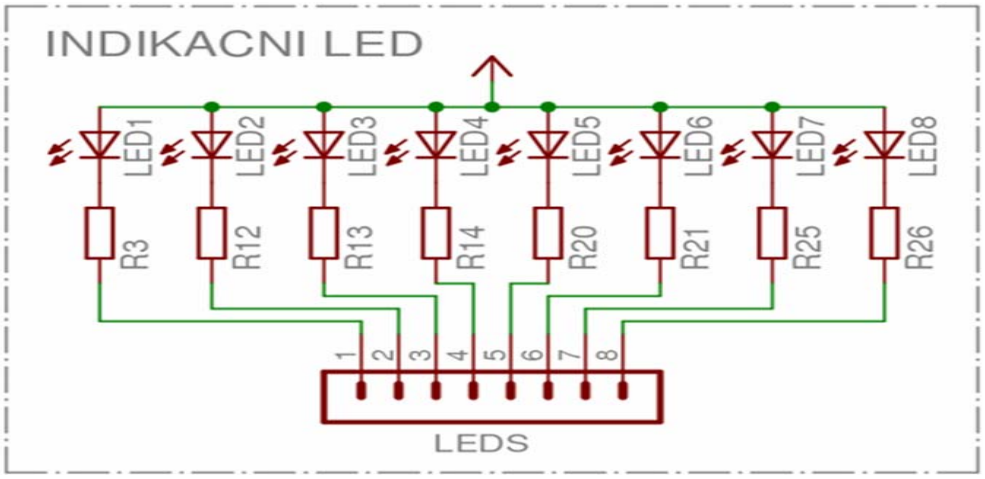
Η πλακέτα EvB 4.3 v4 είναι εξοπλισμένη με μία θύρα RS485, η οποία της επιτρέπει να χρησιμοποιηθεί σε βιομηχανικές εφαρμογές. Οι γραμμές δεδομένων (A και B) βρίσκονται στην κάτω αριστερή γωνία της πλακέτας. Ο 120R jumper είναι υπεύθυνος για τη σύνδεση της γραμμής τερματισμού. Επιπλέον, υπάρχουν οι ακροδέκτες RO, DI και RE.

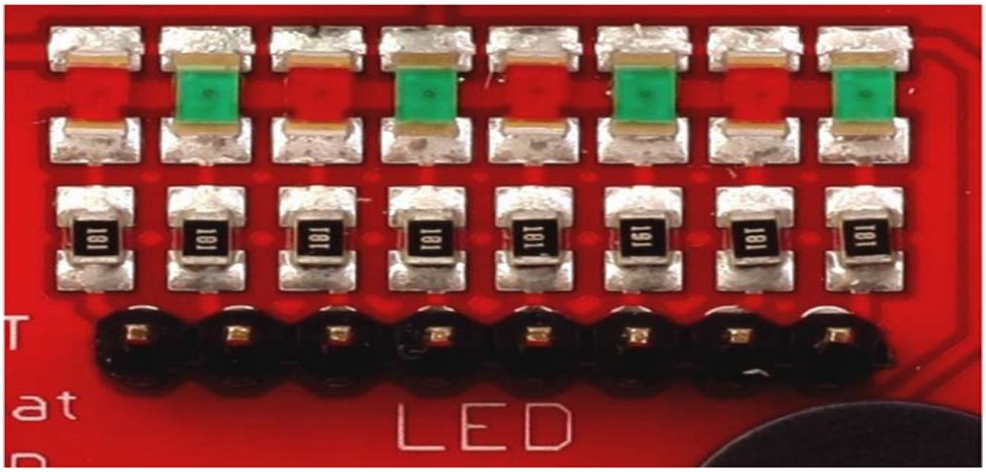




2.7 LEDS

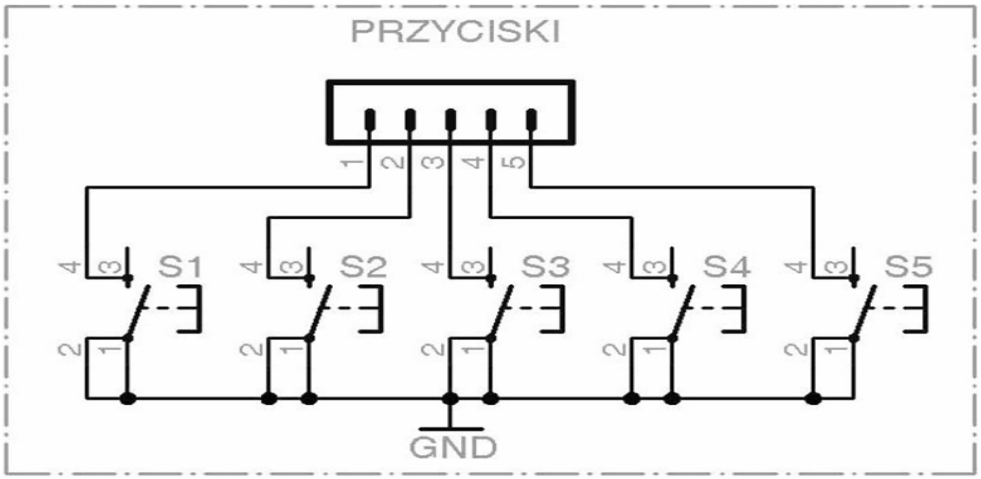
Για να ανάψει ένα LED που είναι εγκατεστημένο στην πλακέτα, πρέπει το σύνολο των ακροδεκτών να είναι σε λογικό «0». Οι ακροδέκτες είναι 8 και ο καθένας από αυτούς αντιστοιχεί σε ένα led.

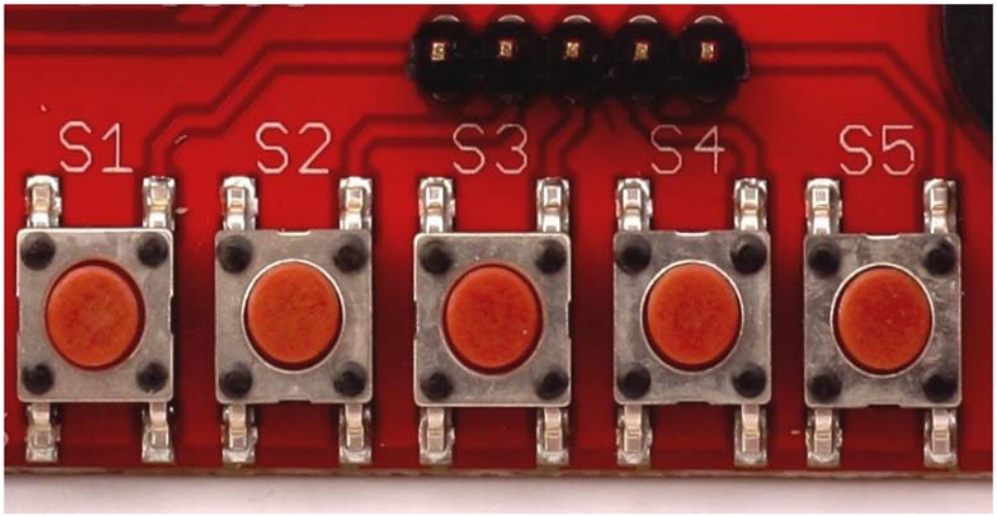




2.8 Κουμπιά πίεσης (Push Buttons)

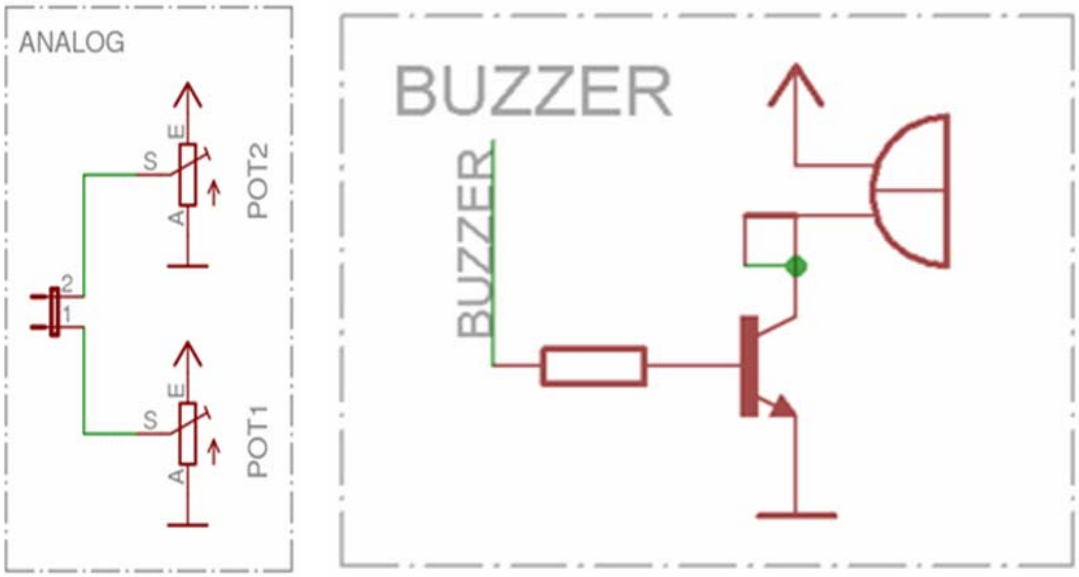
Τα κουμπιά πίεσης της πλακέτας, όταν πατηθούν, συνδέουν τους ακροδέκτες τους με το σύνολο. Ο επεξεργαστής καταλαβαίνει ότι πιέζεται ένα κουμπί, όταν αποσυνδέεται μία αντίσταση. Τα κουμπιά πίεσης είναι 5.

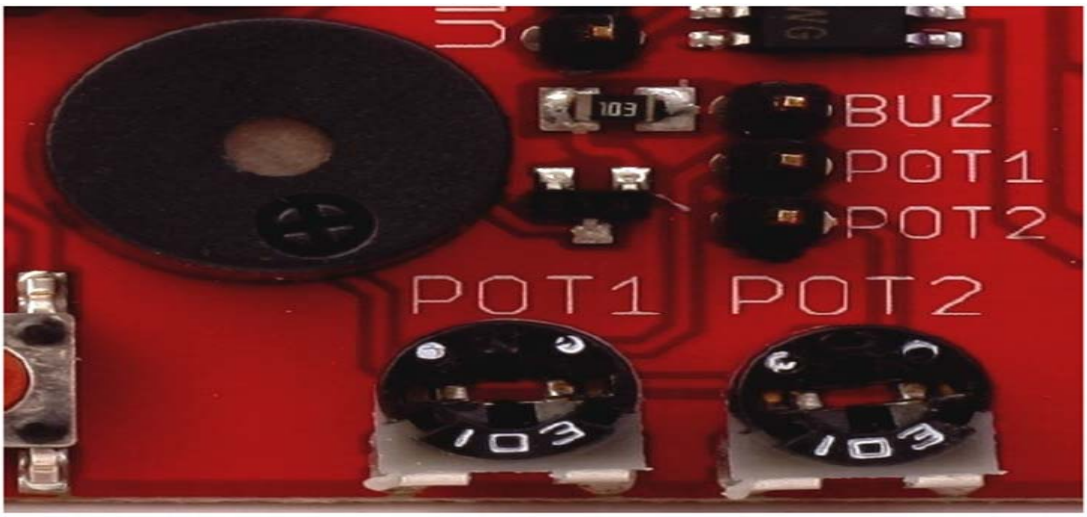




2.9 Ποτενσιόμετρα και βομβητής

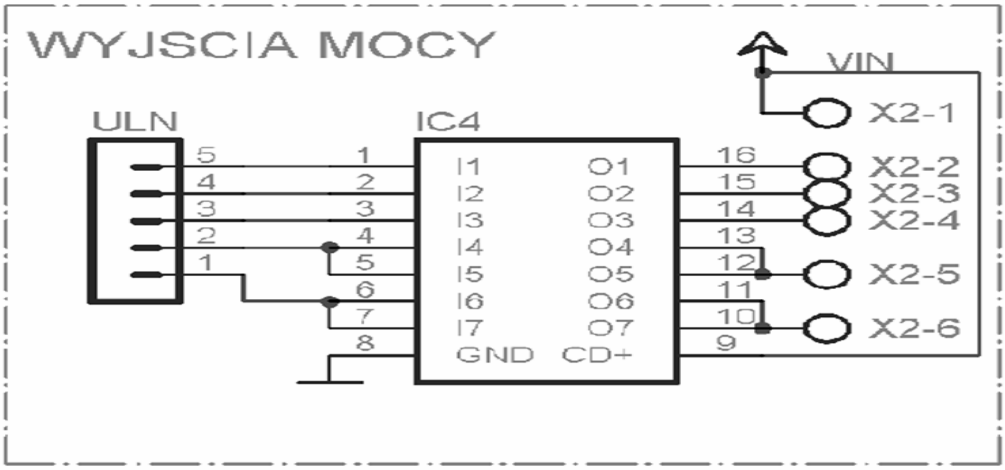
Τα ποτενσιόμετρα που βρίσκονται στην πλακέτα είναι ικανά να παράγουν οποιαδήποτε διαφορά δυναμικού μεταξύ 0 και 5V. Όσον αφορά το βομβητή, είναι τοποθετημένος δίπλα στα ποτενσιόμετρα, έτσι, ώστε να παράγει ηχητικά σήματα, όταν αυτά παράγουν διαφορά δυναμικού +5V.

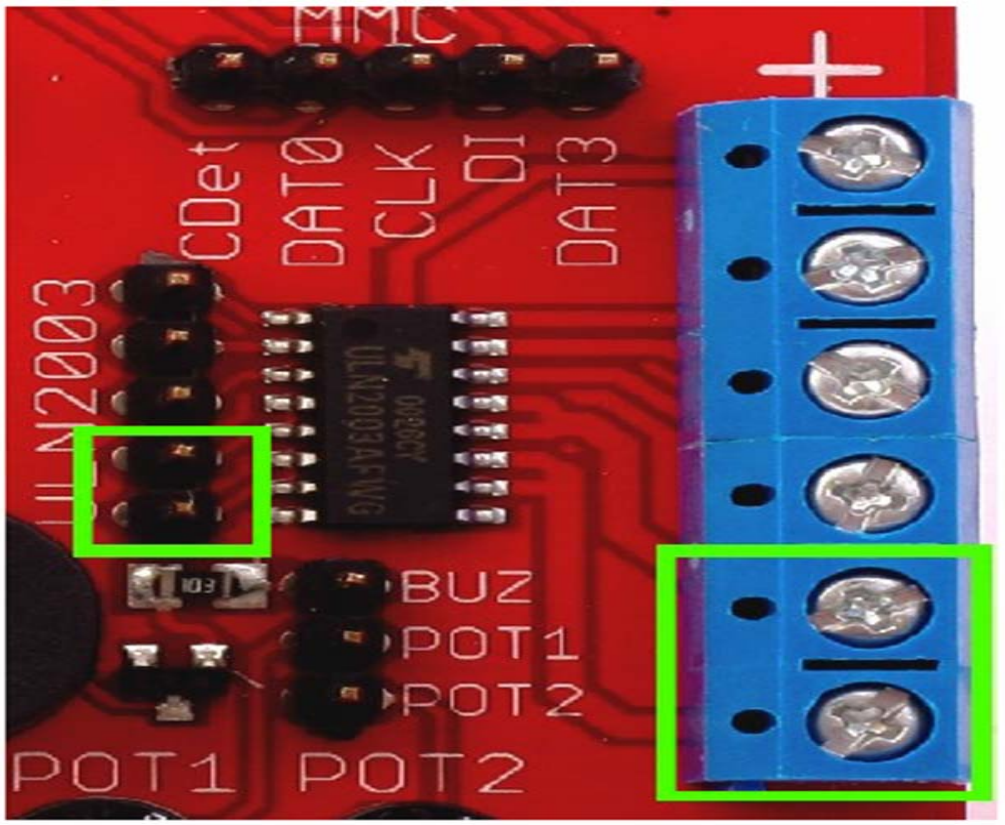




2.10 Ενεργειακές έξοδοι

Η πλακέτα είναι εξοπλισμένη με 5 εξόδους, 2 από τις οποίες έχουν 1A και 3 500mA. Η υψηλότερη έξοδος, που είναι μαρκαρισμένη με το σύμβολο +, αποτελεί τη διαφορά δυναμικού από το εξωτερικό ενεργειακό πακέτο (αν αυτό συνδέεται). Οι άλλες θέσεις του υποδοχέα συνδέονται με το σύνολο μετά τα 5V (λογικό 1), καθώς και με τις αναφερόμενες εξόδους.

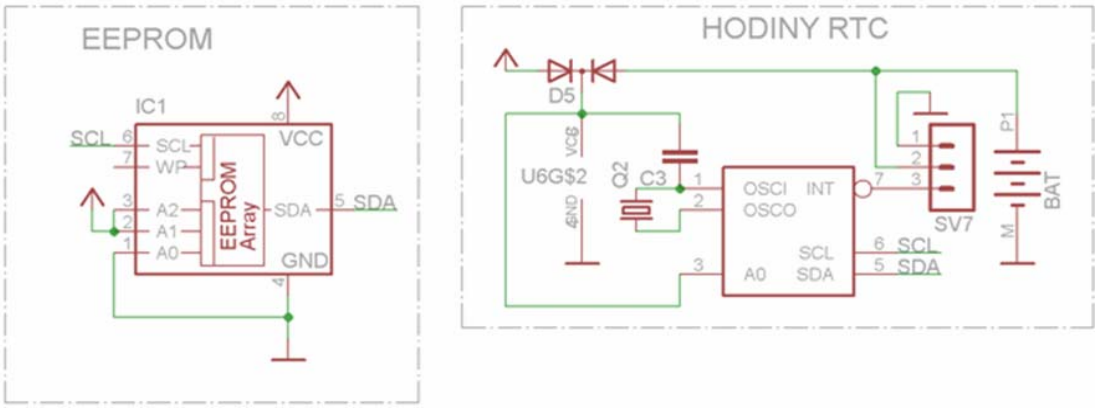


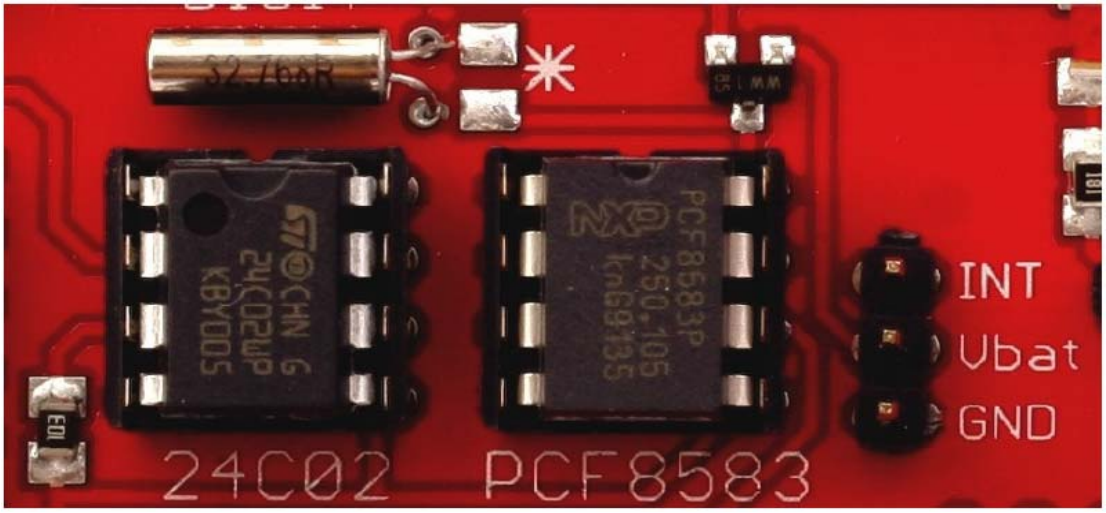


2.11 RTC ρολόι και μνήμη

Η πλακέτα είναι εξοπλισμένη με δύο κυκλώματα χρησιμοποιώντας απλό τρόπο πληροφοριών 12C. Συγκεκριμένα, περιλαμβάνει τα 2 kBits της EEPROM μνήμης [της διεύθυνσης 170 (0*AD) για ανάγνωση και της 173 (0*AC) για εγγραφή] και το ρολόι πραγματικού χρόνου PCF8583 [της διεύθυνσης 162 (0*A2) για ανάγνωση και της 163 (0*A3) για εγγραφή]. Το κύκλωμα PCF8583 είναι συνδεδεμένο με τον ακροδέκτη INT, ο οποίος είναι υπεύθυνος για τις διακοπές που προκαλούνται από το συναγερμό, και με έναν υποδοχέα για την τροφοδοσία της

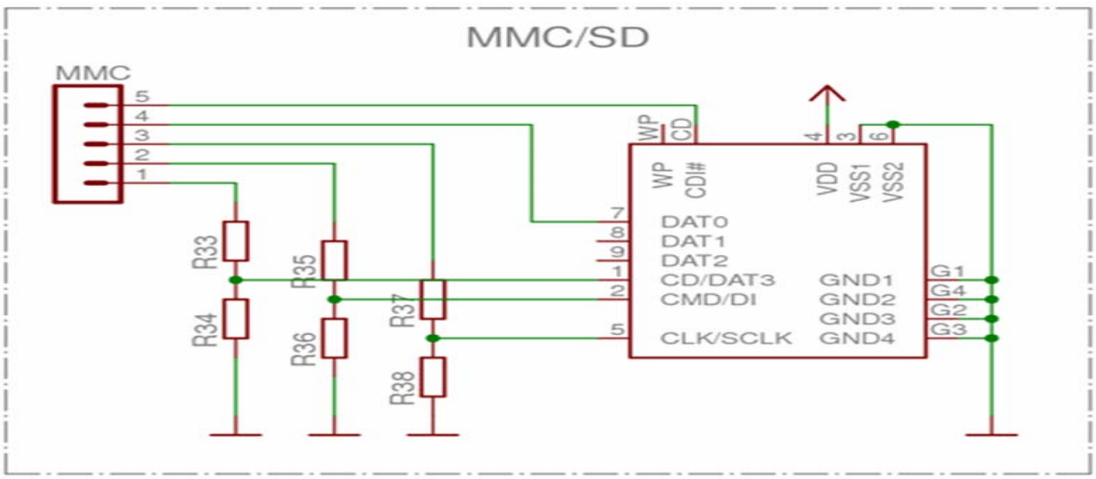
μπαταρίας του ρολογιού. Η μπαταρία μπορεί, επίσης, να εισέλθει σε μία υποδοχή στη βάση της πλακέτας.

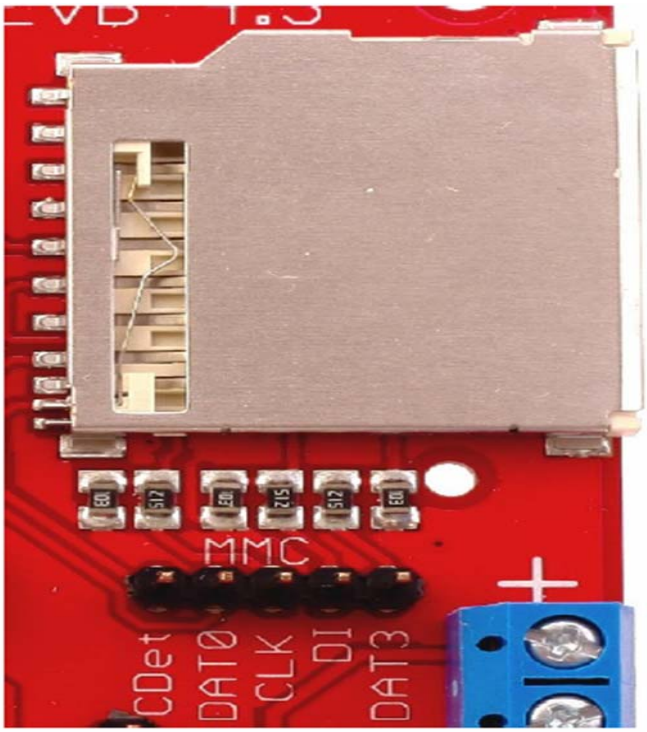




2.12 MMC/SD κάρτα

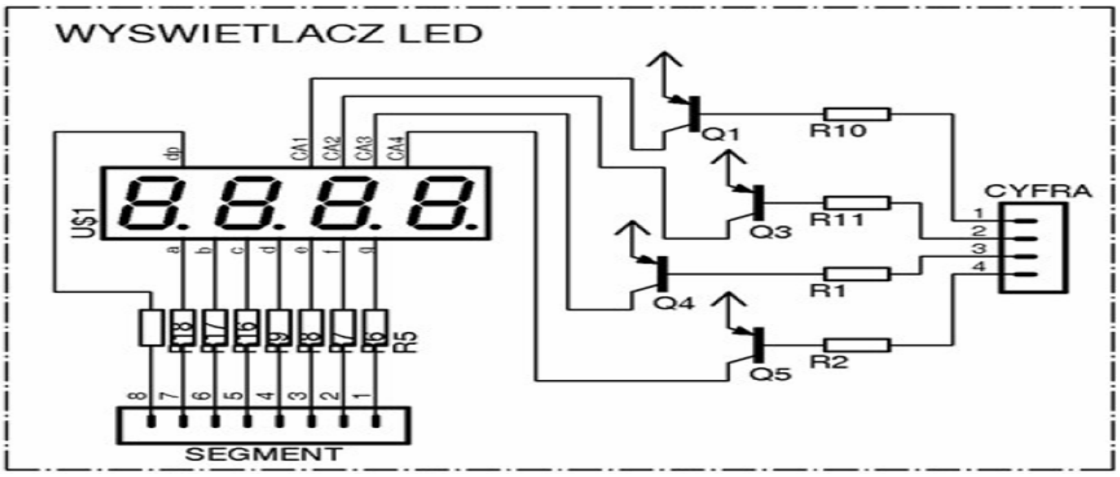
Η πλακέτα είναι εξοπλισμένη με μία υποδοχή για την εξωτερική κάρτα μνήμης (MMC και SD). Η πηγή ενέργειας γι' αυτές τις κάρτες είναι 3,3V και ο σταθεροποιητής βρίσκεται στο σύστημα FT232RL. Τα σήματα προσαρμόζονται στα 5V από τους διαιρέτες αντίστασης. Ο MMC υποδοχέας συνδέεται με μία υποδοχή, χρησιμοποιώντας τα είδη σήματος CLK, DI, DATA0, DATA3 και μία κάρτα επαφής εισόδου.

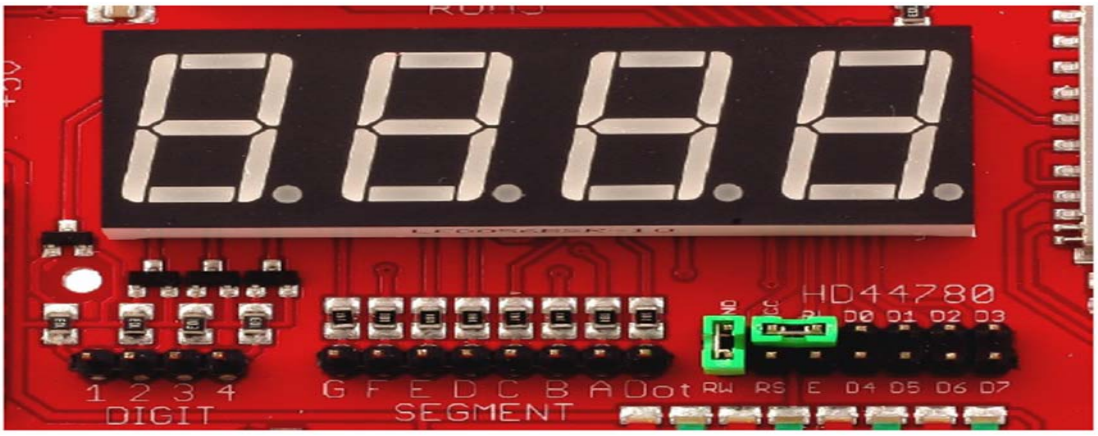




2.13 Οθόνη LED

Προκειμένου να ανάψει ένα τμήμα της οθόνης, ακολουθεί ένα λογικό μηδέν στη βάση του τρανζίστορ (ακροδέκτες DIGI), ενώ ο ακροδέκτης είναι υπεύθυνος για την ακριβή παρουσίαση της οθόνης (ακροδέκτες SEGMENT).

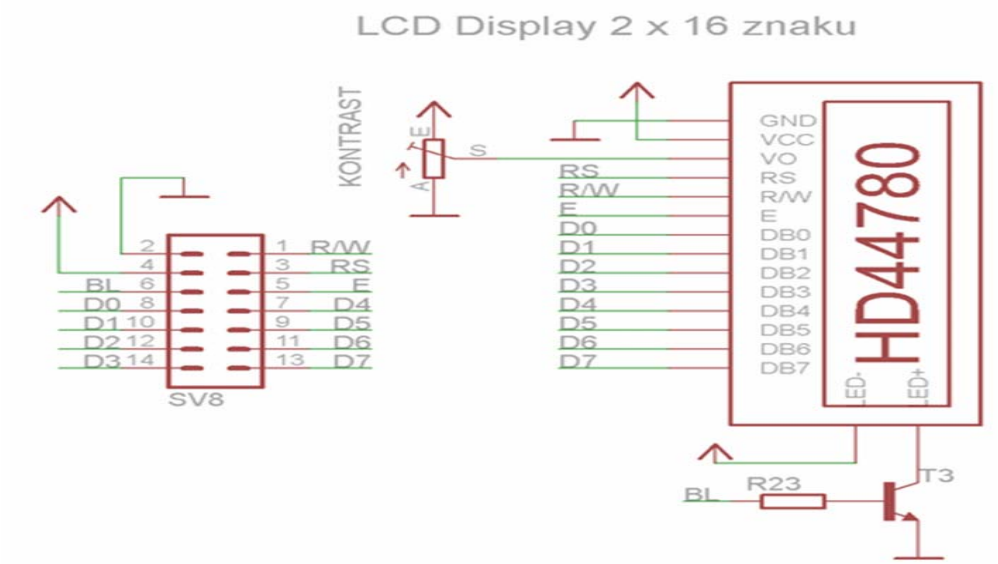


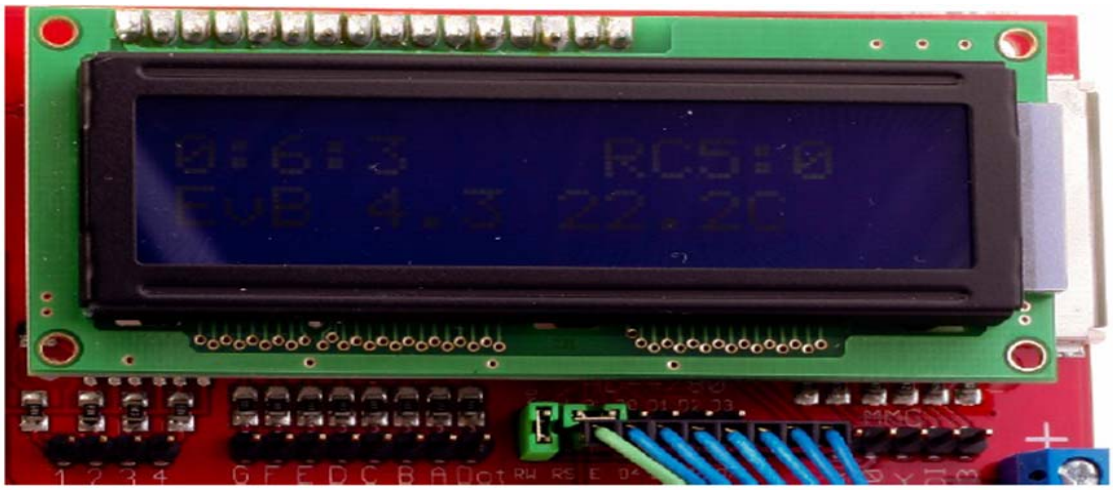


2.14 Οθόνη LCD

Ένας υποδοχέας 16-ακροδεκτών για τη σύνδεση της οθόνης είναι τοποθετημένος στον ελεγκτή HD44780. Η ηλεκτρική σύνδεση της οθόνης ελέγχεται με τη χρήση 4 ή 8 bits λέξεων. Τα σήματα της οθόνης είναι συνδεδεμένα με τον ελεγκτή HD44780 κάτω από την ετικέτα. Η αντίθεση της οθόνης μπορεί να προσαρμοστεί στο ποτενσιόμετρο (σημειώνεται στο παρακάτω σχήμα).

Το σήμα αναπηδά, καθώς ο BL ενεργοποιεί το LCD Back Light (ενεργό στα 5V).

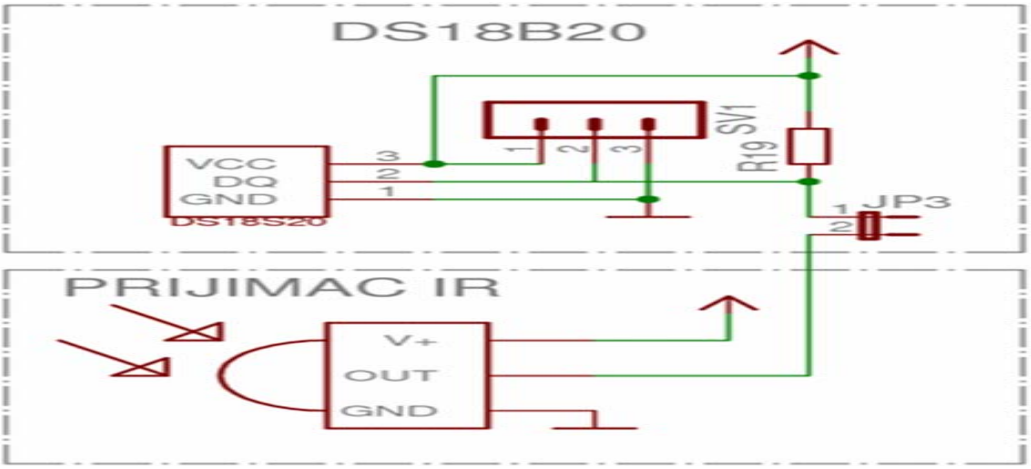


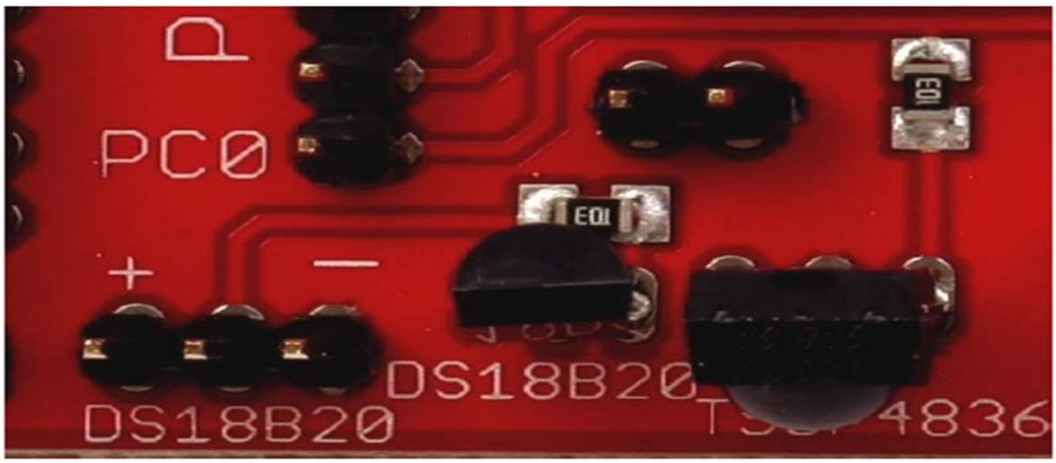


2.15 Αποδέκτης IR και ανιχνευτής θερμοκρασίας

Ο υπέρυθρος ανιχνευτής σήματος TSOP4836 και ο ανιχνευτής θερμοκρασίας DS18B20 είναι διαθέσιμοι στον υποδοχέα που βρίσκεται πάνω από τις αναφερόμενες συσκευές. Ο αριστερός ακροδέκτης με τον ανιχνευτή θερμοκρασίας και ο δεξιός ακροδέκτης συνδέονται με το δέκτη.

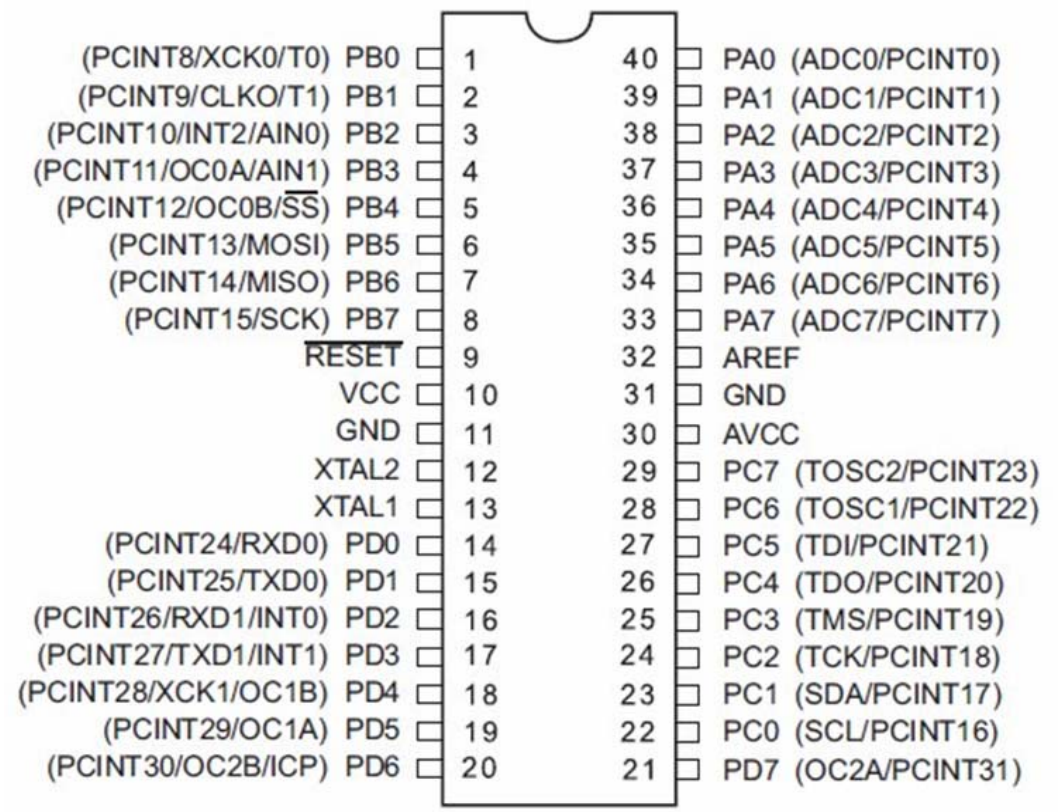
Επίσης, ανιχνευτές θερμοκρασίας μπορούν να περιλαμβάνονται και να συνδέονται με τον υποδοχέα από την αριστερή πλευρά του ανιχνευτή. Όταν εγκατασταθεί ένας επιπλέον ανιχνευτής, πρέπει να είναι στους 180 βαθμούς έναντι του ήδη συγκολλημένου ανιχνευτή (ο αριστερός ακροδέκτης είναι στα +5V, ο μεσαίος είναι υπεύθυνος για το εξωτερικό σήμα και ο δεξιός είναι υπεύθυνος για τη λειτουργία).

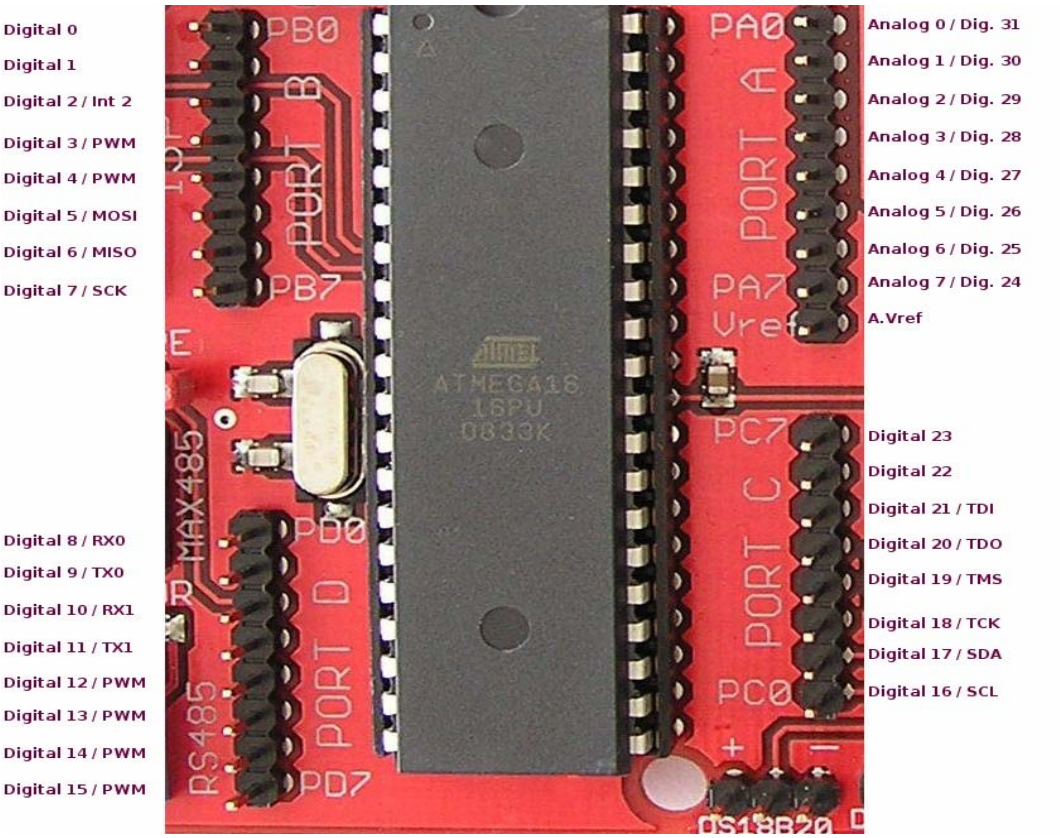




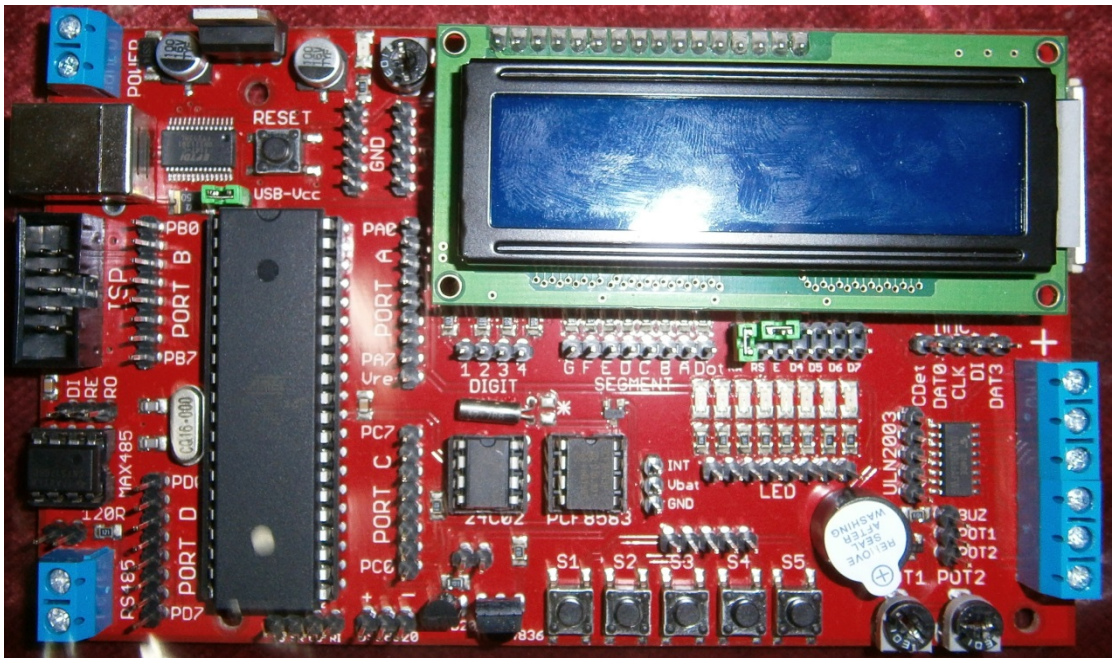
2.16 Ακροδέκτες uC

Το σύνολο των θυρών και ο ακροδέκτης AREF είναι συνδεδεμένα στους ακροδέκτες του επεξεργαστή. Όλες οι περιγραφές των συνδέσεων είναι διαθέσιμες στο παρακάτω σχήμα. Το κουμπί RESET βρίσκεται δίπλα στη θύρα USB και είναι σχεδιασμένο για ολική επανεκκίνηση του κυκλώματος.


(PCINT8/XCK0/T0) PB0 1
(PCINT9/CLKO/T1) PB1 2
(PCINT10/INT2/AIN0) PB2 3
(PCINT11/OC0A/AIN1) PB3 4
(PCINT12/OC0B/SS) PB4 5
(PCINT13/MOSI) PB5 6
(PCINT14/MISO) PB6 7
(PCINT15/SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(PCINT24/RXD0) PD0 14
(PCINT25/TXD0) PD1 15
(PCINT26/RXD1/INT0) PD2 16
(PCINT27/TXD1/INT1) PD3 17
(PCINT28/XCK1/OC1B) PD4 18
(PCINT29/OC1A) PD5 19
(PCINT30/OC2B/ICP) PD6 20
40 PA0 (ADC0/PCINT0)
39 PA1 (ADC1/PCINT1)
38 PA2 (ADC2/PCINT2)
37 PA3 (ADC3/PCINT3)
36 PA4 (ADC4/PCINT4)
35 PA5 (ADC5/PCINT5)
34 PA6 (ADC6/PCINT6)
33 PA7 (ADC7/PCINT7)
32 AREF
31 GND
30 AVCC
29 PC7 (TOSC2/PCINT23)
28 PC6 (TOSC1/PCINT22)
27 PC5 (TDI/PCINT21)
26 PC4 (TDO/PCINT20)
25 PC3 (TMS/PCINT19)
24 PC2 (TCK/PCINT18)
23 PC1 (SDA/PCINT17)
22 PC0 (SCL/PCINT16)
21 PD7 (OC2A/PCINT31)


Digital 0
Digital 1
Digital 2 / Int 2
Digital 3 / PWM
Digital 4 / PWM
Digital 5 / MOSI
Digital 6 / MISO
Digital 7 / SCK
Digital 8 / RX0
Digital 9 / TX0
Digital 10 / RX1
Digital 11 / TX1
Digital 12 / PWM
Digital 13 / PWM
Digital 14 / PWM
Digital 15 / PWM
Analog 0 / Dig. 31
Analog 1 / Dig. 30
Analog 2 / Dig. 29
Analog 3 / Dig. 28
Analog 4 / Dig. 27
Analog 5 / Dig. 26
Analog 6 / Dig. 25
Analog 7 / Dig. 24
A.Vref
Digital 23
Digital 22
Digital 21 / TDI
Digital 20 / TDO
Digital 19 / TMS
Digital 18 / TCK
Digital 17 / SDA
Digital 16 / SCL
PB0
PB7
PD0
PD7
PA0
PA7
PC7
PC0
PORT B
PORT D
PORT A
PORT C
MAX485
RS485
ATMEGA16
16PU
0833K
DS18B20

Εικόνα: η πλακέτα EvB 4.3 v4



Κεφάλαιο 3

Χαρακτηριστικά του ATmega644P – Αρχιτεκτονική

3.1 Χαρακτηριστικά του ATmega644P

Ο μικροελεγκτής AVR ATmega644P κατασκευάζεται από την εταιρεία ATML και έχει τα εξής χαρακτηριστικά:

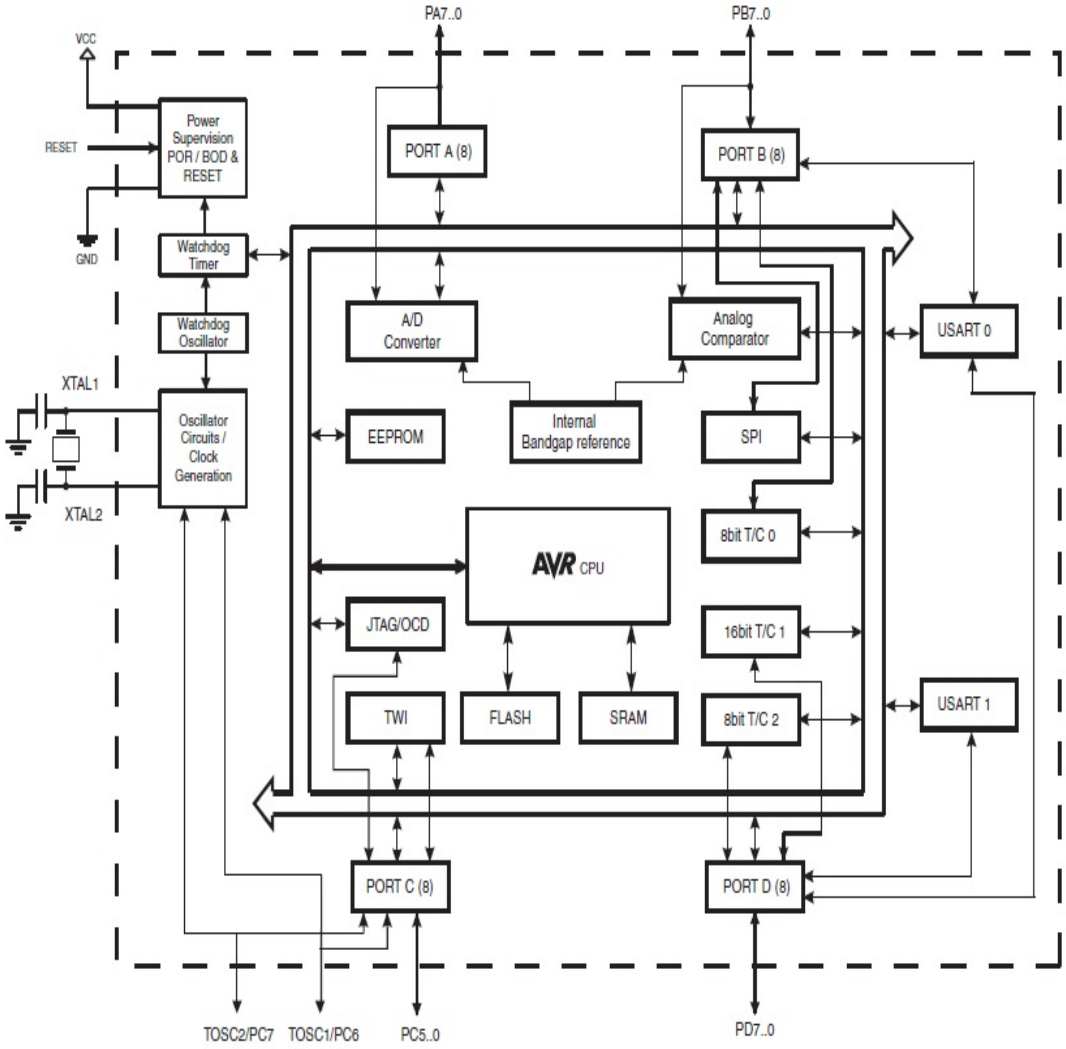
- ✓ Βασίζεται στην αρχιτεκτονική RISC
- ✓ Επεξεργαστή AVR των 8bit
- ✓ 131 εντολές, εκ των οποίων οι περισσότερες εκτελούνται σε ένα μόνο κύκλο του ρολογιού
- ✓ 32*8 καταχωρητές γενικής χρήσης
- ✓ 3 χρονιστές
- ✓ Δύο χρονιστές/μετρητές (timer/counters) των 8-bit, ένα χρονιστή/μετρητή των 16bit και ένα Watchdog timer
- ✓ Ένα μετρητή πραγματικού χρόνου (Real Time Counter), συχνότητας 32khz.
- ✓ Εσωτερικό ταλαντωτή τύπου RC
- ✓ 6 κανάλια PMW (Pulse Width Modulation)
- ✓ Power on reset (σύστημα επανεκκίνησης του μικροελεγκτή κατά την έναρξη της τροφοδοσίας)
- ✓ 10.000 κύκλους εγγραφής/διαγραφής για τη flash memory και 100.000 κύκλους εγγραφής/διαγραφής για την EEPROM memory
- ✓ 8 κανάλια Αναλογικού/Ψηφιακού μετατροπέα (ADC) των 10bits και ταχύτητας 15kbps
- ✓ Έναν αναλογικό συγκριτή
- ✓ 6 κανάλια εξόδου του συγκριτή
- ✓ 3 σειριακές θύρες SPI
- ✓ Μέγιστη συχνότητα λειτουργίας 20MHz
- ✓ 32 εξωτερικές διακοπές
- ✓ 32 ακροδέκτες εισόδου/εξόδου γενικής χρήσης
- ✓ 2 θύρες UART
- ✓ 64Kbytes FLASH Memory
- ✓ 4Kbytes SRAM Memory

- ✓ 2048bytes EEPROM Memory
- ✓ Θερμοκρασία λειτουργίας -40 έως 85 βαθμούς Κελσίου
- ✓ Διατήρηση δεδομένων για 20 χρόνια σε θερμοκρασία 85^{o}C και για 100 χρόνια σε θερμοκρασία 25^{o}C
- ✓ Τάση λειτουργίας 1.8 – 5.5V

3.2 Περιγραφή του ATmega644P

Ο μικροελεγκτής ATmega644P βασίζεται στην αρχιτεκτονική RISC. Εκτελώντας ισχυρές εντολές σε έναν απλό κύκλο ρολογιού, ο ATmega644P επιτυγχάνει διαμεταγωγή, η οποία πλησιάζει το 1 MIPS ανά MHz, επιτρέποντας, έτσι, στο σχεδιαστή του συστήματος να βελτιστοποιήσει την κατανάλωση ενέργειας σε σχέση με την ταχύτητα επεξεργασίας.

Εικόνα: Block Diagram ATmega644P



Ο πυρήνας του μικροελεγκτή AVR συνδυάζει ένα πλούσιο σετ εντολών με 32 καταχωρητές γενικού σκοπού. Και οι 32 καταχωρητές συνδέονται άμεσα με τη Λογική και Αριθμητική Μονάδα (ALU), επιτρέποντας σε δύο ανεξάρτητους καταχωρητές να είναι προσβάσιμοι με μία μόνο εντολή, η οποία εκτελείται σε έναν κύκλο ρολογιού. Η αρχιτεκτονική που προκύπτει είναι περισσότερο αποτελεσματική σε ό,τι αφορά τον κώδικα, επιτυγχάνοντας, παράλληλα, διαμεταγωγή η οποία είναι μέχρι και δέκα φορές πιο γρήγορη συγκριτικά με τους συμβατικούς μικροελεγκτές CISC.

Όσον αφορά τα χαρακτηριστικά γνωρίσματα του μικροελεγκτή ATmega644P, αυτά είναι τα εξής: 64Kbytes μιας προγραμματιζόμενης μνήμης Flash η οποία έχει δυνατότητες ανάγνωσης και εγγραφής, 2Kb EEPROM μνήμη, 4Kb SRAM μνήμη, 32 γραμμές εισόδου – εξόδου (I/O) γενικής χρήσεως, 32 καταχωρητές γενικής χρήσεως, Real Time Counter (RTC), 3 Timer/Counters και PMW, 2 USARTs, μία σειριακή διεπαφή προσανατολισμένη σε byte 2 καλωδίων, έναν 8κάναλο μετατροπέα σημάτων ADC των 10 bits με προαιρετικό το στάδιο διαφορικής εισόδου με προγραμματιζόμενο κέρδος, προγραμματιζόμενο Watch dog Timer με εσωτερικό Oscillator, μία σειριακή θύρα SPI, IEEE std 1149.1 συμβατή με JTAG test interface το επίσης χρησιμοποιείται για την πρόσβαση στο σύστημα On – chip debug και για τον προγραμματισμό του λογισμικού χρησιμοποιούνται έξι τρόποι εξοικονόμησης ενέργειας. Η κατάσταση αναμονής σταματά τη CPU, καθώς επιτρέπει τη SRAM, το Timer/counters, τη θύρα SPI, και διακόπτει το σύστημα, προκειμένου να συνεχίσουν να λειτουργούν.

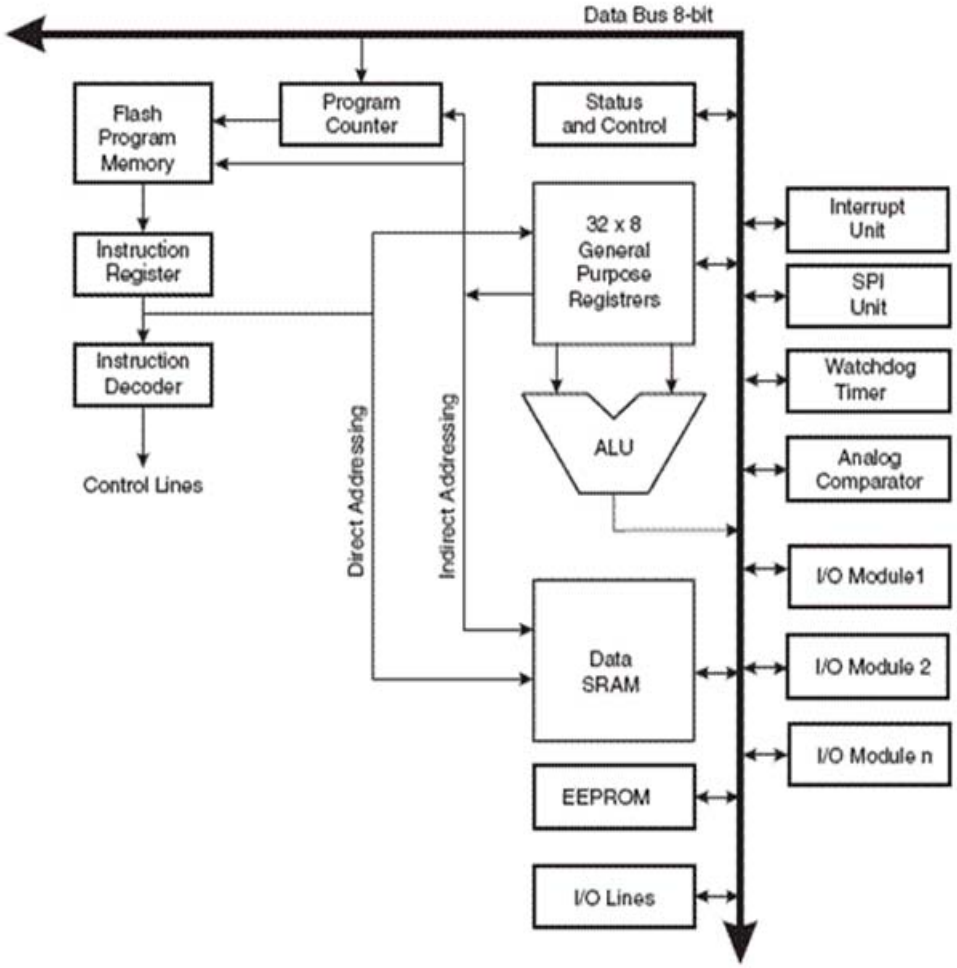
Τέλος, ο μικροελεγκτής ATmega644P υποστηρίζει ένα πλήρες σετ εργαλείων ανάπτυξης και προγραμματισμού. Συγκεκριμένα, περιλαμβάνει μεταγλωττιστές της C, macro assemblers, προγράμματα εντοπισμού σφαλμάτων και προσομοίωσης, κύκλωμα εξομοίωσης και kits αξιολόγησης.

3.3 Ο πυρήνας της CPU του AVR

Η κύρια λειτουργία της CPU είναι να διασφαλίζει τη σωστή εκτέλεση των προγραμμάτων. Συγκεκριμένα, η CPU θα πρέπει να προσπελαύνει μνήμες, να εκτελεί υπολογισμούς, να ελέγχει περιφερειακές συσκευές και να διαχειρίζεται διακοπές (Interrupts).

Στο παρακάτω block διάγραμμα φαίνεται η αρχιτεκτονική AVR.

Εικόνα: Διάγραμμα αρχιτεκτονικής AVR



Προκειμένου να αυξηθεί η απόδοση, ο AVR χρησιμοποιεί αρχιτεκτονική Harvard με ξεχωριστές μνήμες και ξεχωριστούς διαύλους για τα προγράμματα και τα δεδομένα. Όταν μία εντολή εκτελείται, η επόμενη εντολή έχει ήδη αναζητηθεί από τη μνήμη του προγράμματος. Η έννοια αυτή επιτρέπει τις εντολές να εκτελούνται σε κάθε κύκλο ρολογιού. Η μνήμη προγράμματος είναι In – System Reprogrammable Flash memory (αυτοπρογραμματιζόμενη μνήμη Flash).

Επιπλέον, έξι από τους 32 καταχωρητές (οι καταχωρητές R26, R27, R28, R29, R30, R31) μπορούν να χρησιμοποιηθούν σε ζευγάρια ως δείκτες έμμεσης διευθυνσιοδότησης των 16 bits, διευκολύνοντας τους υπολογισμούς διευθύνσεων. Ένας από αυτούς τους καταχωρητές μπορεί, επίσης, να χρησιμοποιηθεί και ως δείκτης διεύθυνσης για πρόσβαση σε πίνακες δεδομένων που είναι αποθηκευμένοι στη μνήμη Flash του προγράμματος. Οι καταχωρητές R26 και R27 αντιστοιχούν στον καταχωρητή X, οι καταχωρητές R28 και R29 στον καταχωρητή Y και οι καταχωρητές R30 και R31 στον καταχωρητή Z.

Όσον αφορά τη Λογική και Αριθμητική μονάδα (ALU), αυτή υποστηρίζει αριθμητικές και λογικές πράξεις μεταξύ των καταχωρητών και μεταξύ μίας τιμής και ενός καταχωρητή. Μετά από μία αριθμητική πράξη, ο καταχωρητής κατάστασης (Status Register) ενημερώνεται, ώστε οι πληροφορίες να αντιστοιχούν στο αποτέλεσμα της πράξης.

Τέλος, κατά τη διάρκεια διακοπών και καλεσμάτων ρουτινών, η διεύθυνση επιστροφής του μετρητή προγράμματος (Program Counter – PC) είναι αποθηκευμένη στη στοίβα (Stack).

3.4 Περιγραφή ακροδεκτών

- **VCC**: Ψηφιακή τάση τροφοδοσίας.
- **GND**: Γείωση.
- **Port A (PA7:PA0)**: Η θύρα Α χρησιμεύει ως αναλογική είσοδος στον Αναλογικό/Ψηφιακό μετατροπέα. Επίσης, η θύρα Α χρησιμεύει και ως 8-μπιτη αμφίδρομη θύρα εισόδου/εξόδου (I/O) με εσωτερικές pull-up αντιστάσεις (που έχουν επιλεγεί για κάθε bit). Ως είσοδοι, οι ακροδέκτες της θύρας Α μπορούν να απάγουν ρεύμα, ακόμα και όταν οι pull-up αντιστάσεις είναι ενεργοποιημένες. Οι ακροδέκτες της θύρας Α είναι τριών καταστάσεων, όταν η κατάσταση επανεκκίνησης (reset) γίνεται ενεργή, ακόμα και σε περίπτωση που το ρολόι δεν τρέχει.
- **Port B (PB7:PB0)**: Η θύρα Β χρησιμεύει και ως 8-μπιτη αμφίδρομη θύρα εισόδου/εξόδου (I/O) με εσωτερικές pull-up αντιστάσεις (που έχουν επιλεγεί για κάθε bit). Ως είσοδοι, οι ακροδέκτες της θύρας Β μπορούν να απάγουν ρεύμα, ακόμα και όταν οι pull-up αντιστάσεις είναι ενεργοποιημένες. Οι ακροδέκτες της θύρας Β είναι τριών καταστάσεων, όταν η κατάσταση επανεκκίνησης (reset) γίνεται ενεργή, ακόμα και σε περίπτωση που το ρολόι δεν τρέχει.
- **Port C (PC7:PC0)**: Η θύρα C χρησιμεύει και ως 8-μπιτη αμφίδρομη θύρα εισόδου/εξόδου (I/O) με εσωτερικές pull-up αντιστάσεις (που έχουν επιλεγεί για κάθε bit). Ως είσοδοι, οι ακροδέκτες της θύρας C μπορούν να απάγουν ρεύμα, ακόμα και όταν οι pull-up αντιστάσεις είναι ενεργοποιημένες. Οι ακροδέκτες της θύρας C είναι τριών καταστάσεων, όταν η κατάσταση

επανεκκίνησης (reset) γίνεται ενεργή, ακόμα και σε περίπτωση που το ρολόι δεν τρέχει.

- **Port D (PD7:PD0)**: Η θύρα D χρησιμεύει και ως 8-μπιτη αμφίδρομη θύρα εισόδου/εξόδου (I/O) με εσωτερικές pull-up αντιστάσεις (που έχουν επιλεγεί για κάθε bit). Ως είσοδοι, οι ακροδέκτες της θύρας D μπορούν να απάγουν ρεύμα, ακόμα και όταν οι pull-up αντιστάσεις είναι ενεργοποιημένες. Οι ακροδέκτες της θύρας D είναι τριών καταστάσεων, όταν η κατάσταση επανεκκίνησης (reset) γίνεται ενεργή, ακόμα και σε περίπτωση που το ρολόι δεν τρέχει.
- **RESET**: Είσοδος Reset. Πρόκειται για ένα χαμηλού επιπέδου ακροδέκτη που για περισσότερο από την ελάχιστη διάρκεια παλμού παράγει reset, ακόμα και αν το ρολόι δεν λειτουργεί. Αντίθετα, μικρότεροι παλμοί δεν εγγυώνται reset.
- **XTAL1**: Πρόκειται για είσοδο στον αναστρέφοντα ταλαντωτή ενίσχυσης του κεντρικού ρολογιού (Oscillator) και είσοδο στο εσωτερικό ρολόι λειτουργίας του κυκλώματος.
- **XTAL2**: Είναι η έξοδος από τον αναστρέφοντα ταλαντωτή ενίσχυσης (Oscillator).
- **AVCC**: Είναι ο ακροδέκτης για την τάση τροφοδοσίας της θύρας F και τον Analog-to-digital Converter. Πρέπει να είναι εξωτερικά συνδεδεμένος με τον ακροδέκτη Vcc, ακόμα και αν δεν χρησιμοποιείται το ADC. Σε περίπτωση που το ADC χρησιμοποιείται, τότε πρέπει να είναι συνδεδεμένο με το Vcc μέσω ενός χαμηλοπερατού φίλτρου.
- **AREF**: Είναι ο αναλογικός ακροδέκτης για τον Analog-to-digital Converter.

Κεφάλαιο 4

Καταχωρητές – Μνήμες

4.1 Καταχωρητής κατάστασης – Status REGister (SREG)

Ο καταχωρητής κατάστασης περιέχει πληροφορίες σχετικά με τα αποτελέσματα των αριθμητικών πράξεων που εκτελέστηκαν πρόσφατα. Οι πληροφορίες αυτές μπορούν να χρησιμοποιηθούν για την αλλαγή της ροής του προγράμματος, προκειμένου να εκτελέσουν ορισμένες εργασίες υπό όρους. Επιπλέον, ο συγκεκριμένος καταχωρητής κατάστασης ενημερώνεται μετά από όλες τις πράξεις της ALU. Ακόμη, ο καταχωρητής κατάστασης περιλαμβάνει 8 bits, το καθένα από τα οποία αντιστοιχεί σε μία σημαία (flags). Οι σημαίες μας ενημερώνουν για την κατάσταση στην οποία βρίσκεται ο επεξεργαστής.

Επίσης, όπως θα αποδειχθεί αργότερα στις εργαστηριακές ασκήσεις, στις πράξεις της πρόσθεσης και της αφαίρεσης συμμετέχει και η σημαία του κρατουμένου (C – Carry flag). Εάν μετά από μία πράξη προκύψει κρατούμενο, τότε θα παρατηρηθεί ότι η σημαία κρατουμένου θα ενημερωθεί αυτόματα και θα είναι ίση με ένα (C = 1), ενώ, αν η πράξη δεν προκαλέσει κρατούμενο, θα παρατηρηθεί ότι η σημαία κρατουμένου δεν θα ανανεωθεί, οπότε θα είναι ίση με μηδέν (C = 0). Τέλος, χρησιμοποιώντας τις εντολές CLC και SEC, μπορούμε να θέσουμε εμείς το κρατούμενο να είναι ίσο με μηδέν ή ίσο με ένα αντίστοιχα.

SREG – Status Register:

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable –** Καθολική Ενεργοποίηση Διακοπών: Το συγκεκριμένο bit καθορίζει την ενεργοποίηση των διακοπών του συστήματος.
- **Bit 6 – T (Target): Bit Copy Storage –** Σημαία αντιγραφής – αποθήκευσης: Οι εντολές [BLD (BitLoaD – ανάγνωση ενός bit) και BST (BitSTore – αποθήκευση ενός bit)] του Bit Copy χρησιμοποιούν το T-bit ως πηγή ή ως προορισμό για τη λειτουργία του συγκεκριμένου bit. Ένα bit από έναν

καταχωρητή του Register File μπορεί να αντιγραφεί σε T-bit με την εντολή BST και ένα bit σε T μπορεί να αντιγραφεί σε ένα bit ενός καταχωρητή του Register File με την εντολή BLD.

- **Bit 5 – H: Half Carry Flag –** Σημαία δεκαδικού κρατουμένου: Το συγκεκριμένο bit ανανεώνεται, όταν υπάρχει δεκαδικό κρατούμενο μετά την εκτέλεση μερικών αριθμητικών πράξεων. Αυτό το bit είναι χρήσιμο στην αριθμητική BCD. Επίσης, προκύπτει από τα 4 λιγότερο σημαντικά ψηφία.

- **Bit 4 – S: Sign Bit, S = N ⊕V –** Σημαία προσήμου: Το συγκεκριμένο bit αντιστοιχεί στη σημαία που προκύπτει από τη λογική πράξη Ή (OR) ανάμεσα στη σημαία αρνητικού προσήμου N και τη σημαία υπερχείλισης στην αριθμητική συμπληρώματος ως προς δύο.
- **Bit 3 – V: Two's Complement Overflow Flag –** Σημαία υπερχείλισης: Είναι η σημαία υπερχείλισης στην αριθμητική συμπληρώματος ως προς δύο.
- **Bit 2 – N: Negative Flag:** Σημαία που δηλώνει ότι ο αριθμός ο οποίος προκύπτει από μία αριθμητική ή από μία λογική πράξη είναι αρνητικός.
- **Bit 1 – Z: Zero Flag –** Σημαία μηδενισμού: Η συγκεκριμένη σημαία δηλώνει ότι το αποτέλεσμα μιας αριθμητικής ή λογικής πράξης είναι 0.
- **Bit 0 – C: Carry Flag –** Σημαία κρατουμένου: Η σημαία κρατουμένου δηλώνει ότι μετά από μία αριθμητική ή λογική πράξη έχει προκύψει κρατούμενο.

4.2 General Purpose Register File

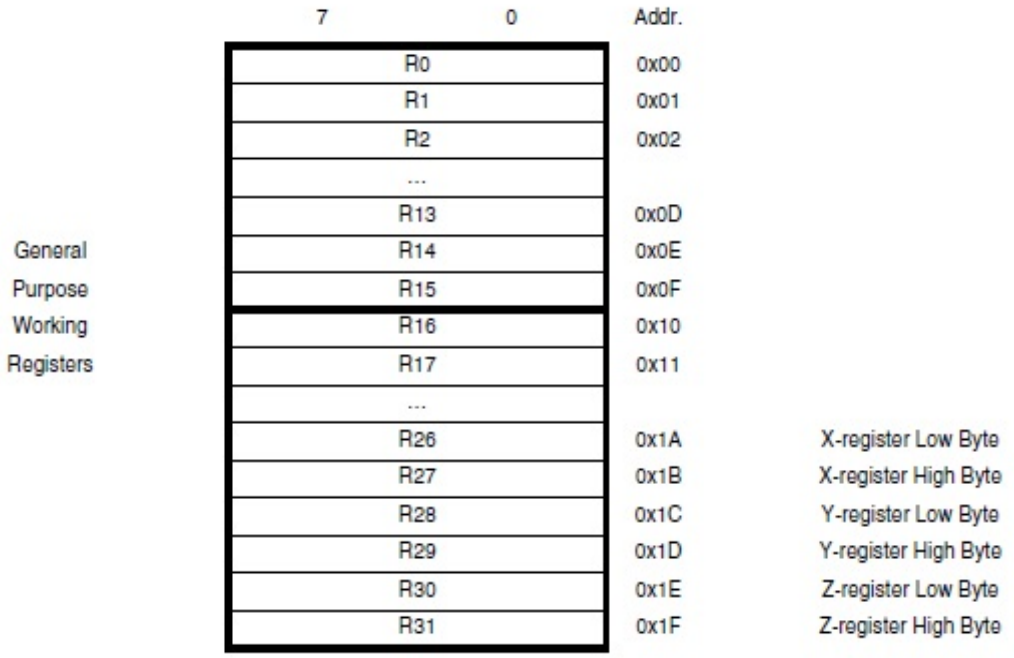
Το Register File έχει βελτιστοποιηθεί για τους μικροελεγκτές AVR με ένα σύνολο εντολών της ενισχυμένης αρχιτεκτονικής RISC. Οι εντολές αυτές έχουν ένα ή δύο ορίσματα (operands), τα οποία είναι πάντοτε οι καταχωρητές γενικού σκοπού R0 έως R31. Προκειμένου να επιτευχθεί η απαιτούμενη απόδοση και ευελιξία στα ακόλουθα συστήματα εισόδου/εξόδου που υποστηρίζονται από το Register File, έχουμε:

- ένα όρισμα εξόδου 8bit και ένα 8bit εισόδου αποτελέσματος
- δύο ορίσματα εξόδου 8bit και ένα 8bit εισόδου αποτελέσματος

- δύο ορίσματα εξόδου 8bit και ένα 16bit εισόδου αποτελέσματος
- ένα όρισμα εξόδου 16bit και ένα 16bit εισόδου αποτελέσματος

Ο μικροελεγκτής AVR έχει 32 καταχωρητές γενικής χρήσης, οι οποίοι φαίνονται στον παρακάτω πίνακα:

Εικόνα: Οι 32 καταχωρητές γενικής χρήσης

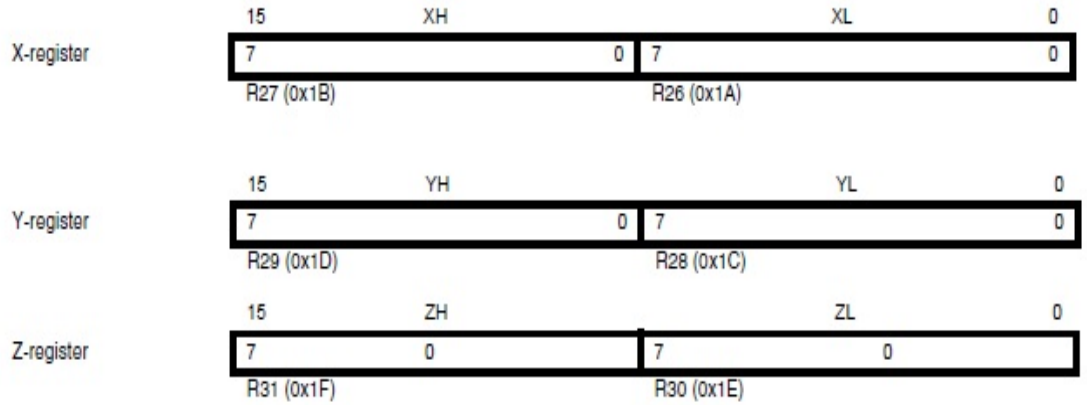


Επίσης, σε κάθε καταχωρητή έχει δοθεί μία διεύθυνση μνήμης δεδομένων, η χαρτογράφηση των οποίων δείχνει απευθείας στις 32 πρώτες θέσεις του χώρου δεδομένων του χρήστη. Η οργάνωση της μνήμης παρέχει μεγάλη ευελιξία όσον αφορά την πρόσβαση των καταχωρητών, όπως τα X-, Y- και Z- pointer registers, τα οποία μπορούν να ρυθμιστούν σε κάθε καταχωρητή μέσα στο αρχείο.

4.3 Καταχωρητές X, Y, Z

Οι καταχωρητές R26, R27, R28, R29, R30 και R31 έχουν μερικές επιπλέον λειτουργίες. Οι συγκεκριμένοι καταχωρητές είναι δείκτες διεύθυνσης εύρους 16-bit για την έμμεση διευθυνσιοδότηση του χώρου δεδομένων. Οι καταχωρητές αυτοί αντιστοιχίζονται (ανά δύο) σε 3 άλλους καταχωρητές στο X, Y και Z.

Εικόνα: Καταχωρητές X,Y, Z



Ο καταχωρητής X ορίζεται από τους καταχωρητές R27, R26.

Ο καταχωρητής Y ορίζεται από τους καταχωρητές R29, R28.

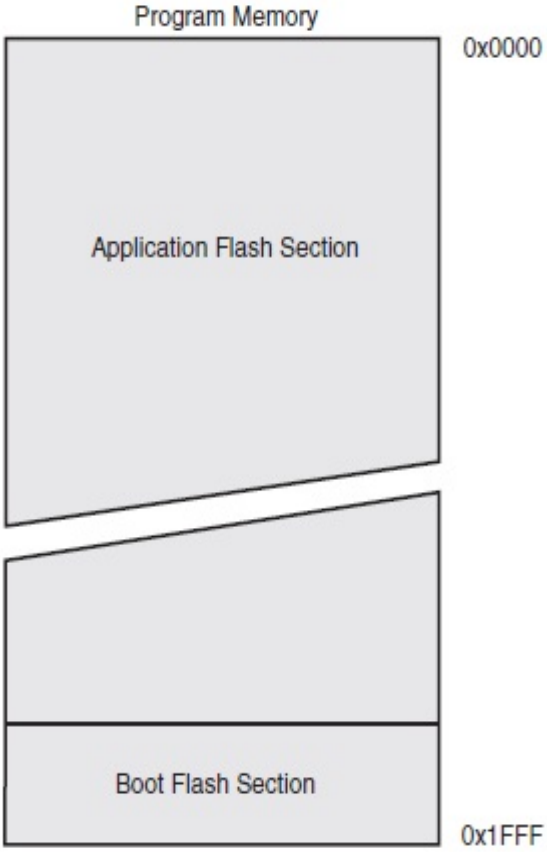
Ο καταχωρητής Z ορίζεται από τους καταχωρητές R31, R30.

Οι καταχωρητές X, Y και Z έχουν εύρος 16bit και χρησιμοποιούνται και ως δείκτες (δείκτης X, δείκτης Y, δείκτης Z).

4.4 Μνήμη προγράμματος Flash

Ο μικροελεγκτής ATmega644P περιέχει 64Kbytes On-chip In-System Reprogrammable Flash memory για την αποθήκευση των προγραμμάτων. Καθώς όλες οι εντολές του AVR είναι εύρους 16 ή 32 bits, η Flash memory είναι οργανωμένη ως 32/64 x 16. Για την ασφάλεια του λογισμικού, ο χώρος μνήμης προγράμματος χωρίζεται σε δύο τμήματα: στο τμήμα εκκίνησης προγράμματος (Boot Program section) και στο τμήμα εφαρμογής προγράμματος (Application Program section). Επίσης, η μνήμη είναι αμετάβλητου τύπου, διατηρεί τα περιεχόμενά της αναλλοίωτα, ακόμα και όταν δεν υπάρχει τάση τροφοδοσίας, και το περιεχόμενό της μπορεί να γραφεί ξανά μέσω του προγραμματιστή για περίπου 10.000 φορές.

Εικόνα: Χαρτογράφηση της Flash memory



4.5 Μνήμη EEPROM

Ο μικροελεγκτής ATmega644P περιέχει μία EEPROM μνήμη των 2Kbytes. Η μνήμη είναι οργανωμένη ως ξεχωριστός χώρος δεδομένων στο μικροελεγκτή, όπου απλά bytes μπορούν να διαβαστούν και να γραφούν. Η EEPROM έχει δυνατότητα για τουλάχιστον 100.000 write/erase κύκλους.

Οι μνήμες EEPROM (ή E^2PROM) αποτελούν εξέλιξη των προγενέστερων μνημών ROM. Οι συγκεκριμένες μνήμες, σε αντίθεση με τις αντίστοιχες RAM, διατηρούν τα περιεχόμενά τους και μετά τη διακοπή της τροφοδοσίας τους με ηλεκτρική ισχύ. Μπορούν, όμως, να διαγραφούν και να επαναπρογραμματιστούν με νέες πληροφορίες με ηλεκτρικό τρόπο, ακόμη και πάνω στο κύκλωμα στο οποίο είναι τοποθετημένες.

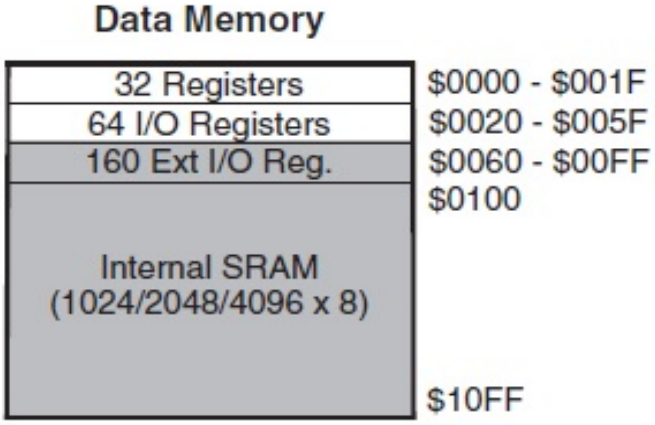
4.6 SRAM

Μνήμη τυχαίας προσπέλασης (RAM, Random access memory) είναι ο όρος που χρησιμοποιούμε για ηλεκτρονικές διατάξεις προσωρινής αποθήκευσης ψηφιακών

δεδομένων (μνήμης υπολογιστή), οι οποίες επιτρέπουν πρόσβαση στα δεδομένα που έχουν αποθηκευτεί στον ίδιο χρόνο, οπουδήποτε και αν βρίσκονται αυτά, δηλαδή με «τυχαία πρόσβαση».

Στο μικροελεγκτή ATmega644P οι 4.352 θέσεις μνήμης δεδομένων αφορούν: το αρχείο καταχωρητών (Register File), τη μνήμη εισόδου – εξόδου (I/O), την επεκτεταμένη μνήμη εισόδου – εξόδου (Extended I/O Memory) και την εσωτερική μνήμη δεδομένων SRAM. Οι πρώτες 32 θέσεις αντιστοιχούν στους καταχωρητές (στο αρχείο των καταχωρητών, Register File), οι επόμενες 64 θέσεις είναι για τη μνήμη καταχωρητών εισόδου – εξόδου (I/O Memory), οι 160 που ακολουθούν αφορούν την επεκτεταμένη μνήμη καταχωρητών εισόδου – εξόδου (Extended I/O memory) και οι επόμενες 4.096 θέσεις καλύπτουν την εσωτερική SRAM μνήμη.

Εικόνα: Χαρτογράφηση SRAM



Στη SRAM, για τον επεκτεταμένο χώρο εισόδου – εξόδου από $060 έως $FF χρησιμοποιούνται μόνο οι εντολές ST, STS, STD, LD, LDS και LDD.

Εντολή ST (ST Y, X ή Z, καταχωρητής): Η εντολή ST (Store indirect) μεταφέρει το περιεχόμενο του καταχωρητή στη θέση μνήμης της SRAM.

Σύνταξη: ST Y, Rr

Παράδειγμα: ST X, R15

Εντολή ST (ST Y+, X+ ή Z+, καταχωρητής): Η εντολή ST (Store indirect and post–inc.) μεταφέρει το περιεχόμενο του καταχωρητή στην επόμενη θέση μνήμης της SRAM. Δηλαδή, αν το Y δείχνει στην 0200, το Y+ θα δείχνει στην 0201.

Σύνταξη: ST Y+, Rr

Παράδειγμα: ST Y+, R15

Εντολή ST (ST -Y, -X ή -Z, καταχωρητής): Η εντολή ST (Store indirect and pre-dec.) μεταφέρει το περιεχόμενο του καταχωρητή στην προηγούμενη θέση μνήμης της SRAM από αυτήν που δείχνει το Y.

Σύνταξη: ST –Y, Rr

Παράδειγμα: ST –Y, R15

Εντολή STD (ST Y+q, ή Z+q, καταχωρητής): Η εντολή STD (Store indirect with displacement) μεταφέρει το περιεχόμενο του καταχωρητή στη θέση μνήμης της SRAM η οποία είναι αυξημένη κατά q θέσεις.

Σύνταξη: STD Y+q, Rr

Παράδειγμα: STD Y+10, R15

Εντολή LD (LD καταχωρητής, X, Y ή Z): Η εντολή LD (Load indirect) τοποθετεί το περιεχόμενο της θέσης μνήμης της SRAM σε έναν καταχωρητή.

Σύνταξη: LD Rd, Y

Παράδειγμα: LD R20, Y

Εντολή LD (LD καταχωρητής, X+, Y+, Z+): Η εντολή LD (Load indirect and post-inc) μεταφέρει το περιεχόμενο της θέσης μνήμης SRAM, η οποία θέση μνήμης είναι αυξημένη κατά ένα, σε έναν καταχωρητή.

Σύνταξη: LD Rd, Y+

Παράδειγμα: LD R20, Y+

Εντολή LD (LD καταχωρητής, -X, -Y, -Z): Η εντολή LD (Load indirect and pre-dec) μεταφέρει το περιεχόμενο της θέσης μνήμης SRAM, η οποία θέση μνήμης είναι μειωμένη κατά ένα, σε έναν καταχωρητή.

Σύνταξη: LD Rd, -Y

Παράδειγμα: LD R20, -Y

Εντολή LDD (LDD καταχωρητής, Y+q, Z+q): Η εντολή LDD (Load indirect with displacement), μεταφέρει το περιεχόμενο της θέσης μνήμης SRAM, η οποία θέση μνήμης είναι αυξημένη κατά q θέσεις, σε έναν καταχωρητή.

Σύνταξη: LDD Rd, Y+q

Παράδειγμα: LDD R20, Y+10

Σημείωση: Οι εντολές LDS και STS αναλύονται στο πρώτο κεφάλαιο των εργαστηριακών ασκήσεων, στις σελίδες 64 και 65.

Κεφάλαιο 5

Θύρες εισόδου/εξόδου (I/O)

5.1 Οι θύρες εισόδου/εξόδου (I/O)

Όλες οι θύρες του AVR έχουν τρεις πραγματικές λειτουργίες Ανάγνωση – Τροποποίηση – Εγγραφή (Read – Modify – Write), όταν χρησιμοποιούνται ως γενικές ψηφιακές θύρες εισόδου – εξόδου (I/O ports). Αυτό σημαίνει ότι ο κάθε ακροδέκτης μιας θύρας μπορεί να μετατραπεί από ακροδέκτης εισόδου σε ακροδέκτη εξόδου και το αντίστροφο, χωρίς ακούσια μετατροπή άλλων ακροδεκτών και με τη χρήση των εντολών SBI και CBI. Για κάθε θύρα εισόδου – εξόδου υπάρχουν συνολικά τρεις διευθύνσεις στη μνήμη εισόδου – εξόδου. Πρόκειται για τις διευθύνσεις DDxn, PORTxn και PINxn.

Η πρώτη διεύθυνση, η DDxn, αναφέρεται και ως καταχωρητής DDxn και χρησιμοποιείται για τον ορισμό της κατεύθυνσης των ακροδεκτών, δηλαδή ορίζει ποιος ακροδέκτης θα είναι είσοδος και ποιος ακροδέκτης θα είναι έξοδος. Αν ο καταχωρητής DDxn έχει την τιμή ένα, ο ακροδέκτης Pxn λειτουργεί ως ακροδέκτης εξόδου. Αν ο DDxn έχει την τιμή 0, ο Pxn λειτουργεί ως ακροδέκτης εισόδου.

Η δεύτερη διεύθυνση, η PORTxn, αναφέρεται και ως καταχωρητής PORTxn και αφορά τα δεδομένα που πρόκειται να εγγραφούν στους ακροδέκτες που έχουν οριστεί ως έξοδοι. Αν ο καταχωρητής PORTxn έχει την τιμή ένα και ο ακροδέκτης έχει οριστεί ως ακροδέκτης εισόδου, τότε ενεργοποιείται η αντίσταση pull – up. Για να απενεργοποιηθεί η αντίσταση pull – up, θα πρέπει στον καταχωρητή PORTxn να γραφεί λογικό μηδέν ή ο ακροδέκτης να μετατραπεί σε ακροδέκτη εξόδου. Επίσης, οι ακροδέκτες όλων των θυρών είναι τρισταθείς (tristated).

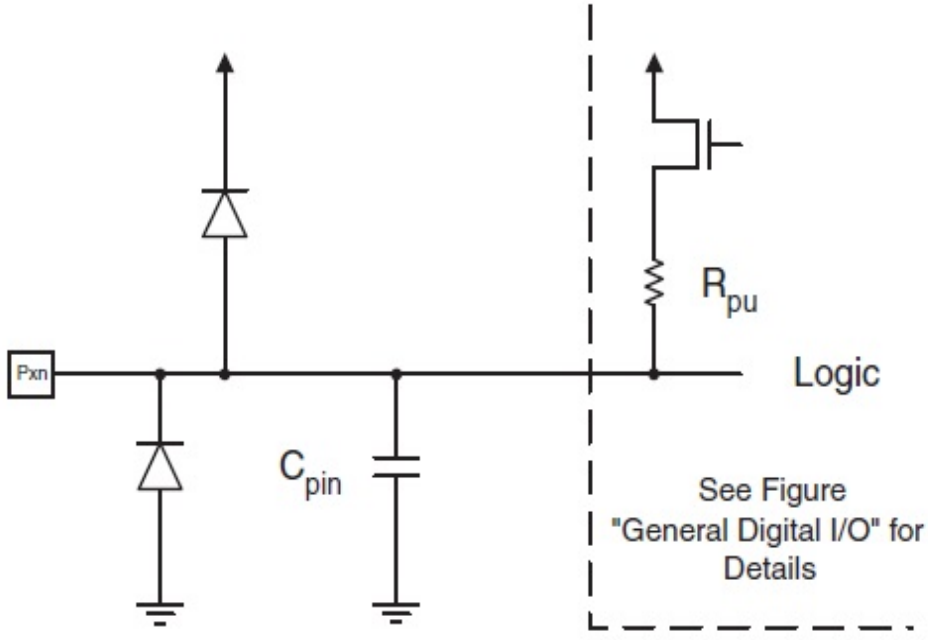
Επιπλέον, σε περίπτωση που ο καταχωρητής PORTxn έχει την τιμή ένα και ο ακροδέκτης είναι ακροδέκτης εξόδου, τότε ο ακροδέκτης της θύρας οδηγείται στο λογικό υψηλό (ένα). Αντίθετα, αν ο καταχωρητής PORTxn έχει την τιμή μηδέν και ο ακροδέκτης έχει οριστεί ως ακροδέκτης εξόδου, τότε ο ακροδέκτης οδηγείται στο λογικό χαμηλό (μηδέν).

Τέλος, η τρίτη διεύθυνση (είναι διεύθυνση ανάγνωσης), η PINxn, αναφέρεται και ως καταχωρητής PINxn. Αφορά τα δεδομένα που διαβάζονται από τους ακροδέκτες που έχουν οριστεί ως είσοδοι.

Ο παρακάτω πίνακας συνοψίζει τα σήματα ελέγχου για την τιμή των ακροδεκτών:

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Εικόνα: Ισοδύναμο κύκλωμα θυρών εισόδου/εξόδου:



Ο μικροελεγκτής ATmega644P διαθέτει 32 ακροδέκτες εισόδου – εξόδου, όπου οι 32 καταχωρητές είναι οργανωμένοι σε 4 θύρες εισόδου – εξόδου, η καθεμία από τις οποίες περιλαμβάνει 8 ακροδέκτες: Port A, Port B, Port C και Port D. Οι θύρες αυτές χρησιμοποιούνται είτε ως γενικές ψηφιακές είσοδοι – έξοδοι είτε με βάση τις εναλλακτικές τους λειτουργίες.

5.2 Οι καταχωρητές προγραμματισμού των PORTs

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

  When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01).

PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINA – Port A Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINB – Port B Input PIns Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

PORTC – Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRC – Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINC – Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

PORTD – Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRD – Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PIND – Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

5.3 Εναλλακτικές λειτουργίες των ακροδεκτών

Εκτός από τις παραπάνω λειτουργίες, στους ακροδέκτες των τεσσάρων θυρών αντιστοιχούν και εναλλακτικές λειτουργίες.

Εναλλακτικές λειτουργίες των ακροδεκτών της θύρας Α:

Ακροδέκτης θύρας	Εναλλακτική λειτουργία
PA7	ADC7 (Κανάλι εισόδου 7 του ADC) PCINT7 (Ακροδέκτης της αλλαγής διακοπής 7)
PA6	ADC6 (Κανάλι εισόδου 6 του ADC) PCINT6 (Ακροδέκτης της αλλαγής διακοπής 6)
PA5	ADC5 (Κανάλι εισόδου 5 του ADC) PCINT5 (Ακροδέκτης της αλλαγής διακοπής 5)
PA4	ADC4 (Κανάλι εισόδου 4 του ADC) PCINT4 (Ακροδέκτης της αλλαγής διακοπής 4)
PA3	ADC3 (Κανάλι εισόδου 3 του ADC) PCINT3 (Ακροδέκτης της αλλαγής διακοπής 3)
PA2	ADC2 (Κανάλι εισόδου 2 του ADC) PCINT2 (Ακροδέκτης της αλλαγής διακοπής 2)
PA1	ADC1 (Κανάλι εισόδου 1 του ADC) PCINT1 (Ακροδέκτης της αλλαγής διακοπής 1)
PA0	ADC0 (Κανάλι εισόδου 0 του ADC) PCINT0 (Ακροδέκτης της αλλαγής διακοπής 0)

5.4 Εναλλακτικές λειτουργίες των ακροδεκτών της θύρας Β

Ακροδέκτης θύρας	Εναλλακτική λειτουργία
PB7	SCK (Σειριακό ρολόι του διαύλου SPI) PCINT15 (Ακροδέκτης της αλλαγής διακοπής 15)
PB6	MISO (Δίαυλος SPI Master Εισόδου/Slave Εξόδου) PCINT14 (Ακροδέκτης της αλλαγής διακοπής 14)
PB5	MOSI (Δίαυλος SPI Master Εξόδου/Slave Εισόδου) PCINT13 (Ακροδέκτης της αλλαγής διακοπής 13)
PB4	SS (Δίαυλος SPI, είσοδος επιλογής slave) OC0B (Έξοδος Β του συγκριτή του Timer/Counter 0) PCINT12 (Ακροδέκτης της αλλαγής διακοπής 12)
PB3	AIN1 (Αρνητική είσοδος του αναλογικού συγκριτή) OC0A (Έξοδος Α του συγκριτή του Timer/Counter 0) PCINT11 (Ακροδέκτης της αλλαγής διακοπής 11)
PB2	AIN0 (Θετική είσοδος του αναλογικού συγκριτή) INT2 (Είσοδος της εξωτερικής διακοπής 2) PCINT10 (Ακροδέκτης της αλλαγής διακοπής 10)
PB1	T1 (Timer/Counter1 εξωτερική είσοδος μετρητή) CLKO (Σύστημα εξόδου ρολογιού) PCINT9 (Ακροδέκτης της αλλαγής διακοπής 9)
PB0	T0 (Timer/Counter0 εξωτερική είσοδος μετρητή) XCK0 (Εξωτερικό ρολόι εισόδου/εξόδου USART) PCINT8 (Ακροδέκτης της αλλαγής διακοπής 8)

5.5 Εναλλακτικές λειτουργίες των ακροδεκτών της θύρας C

Ακροδέκτης θύρας	Εναλλακτική λειτουργία
PC7	TOSC2 (Ακροδέκτης 2 του ταλαντωτή του Timer) PCINT23 (Ακροδέκτης της αλλαγής διακοπής 23)
PC6	TOSC1 (Ακροδέκτης 1 του ταλαντωτή του Timer) PCINT22 (Ακροδέκτης της αλλαγής διακοπής 22)
PC5	TDI (Έλεγχος δεδομένων εισόδου του JTAG) PCINT21 (Ακροδέκτης της αλλαγής διακοπής 21)
PC4	TDO (Έλεγχος δεδομένων εξόδου του JTAG) PCINT20 (Ακροδέκτης της αλλαγής διακοπής 20)
PC3	TMS (Έλεγχος επιλογής της κατάστασης λειτουργίας του JTAG) PCINT19 (Ακροδέκτης της αλλαγής διακοπής 19)
PC2	TCK (Έλεγχος ρολογιού του JTAG) PCINT18 (Ακροδέκτης της αλλαγής διακοπής 18)
PC1	SDA (Δίαυλος δεδομένων του I2C) PCINT17 (Ακροδέκτης της αλλαγής διακοπής 17)
PC0	SCL (Δίαυλος ρολογιού του I2C) PCINT16 (Ακροδέκτης της αλλαγής διακοπής 16)

5.6 Εναλλακτικές λειτουργίες των ακροδεκτών της θύρας D

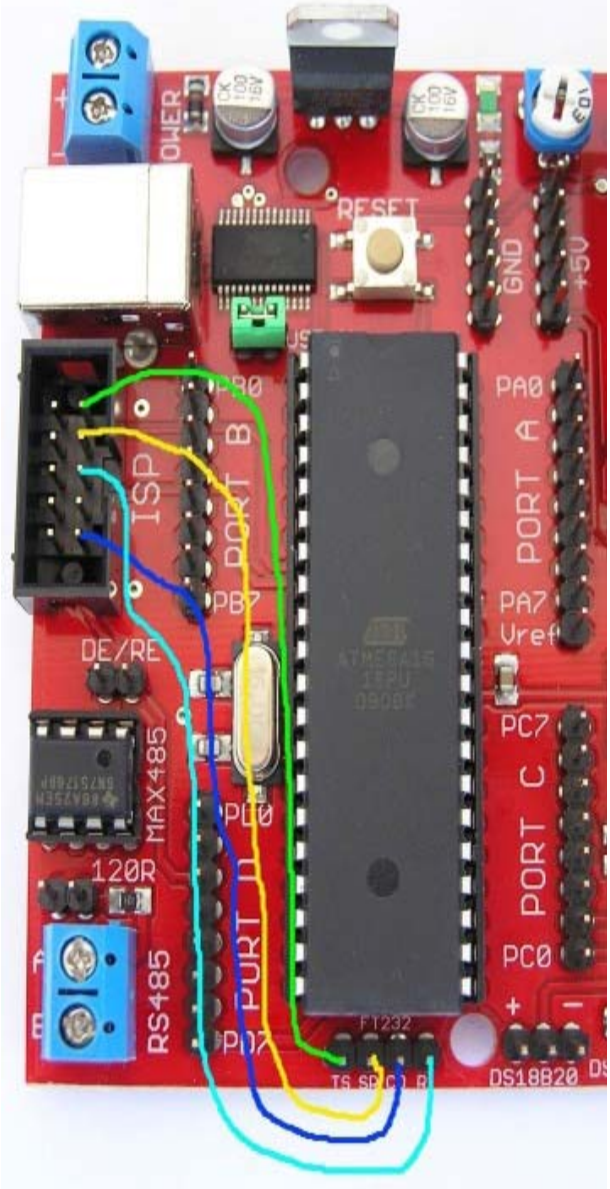
Ακροδέκτης θύρας	Εναλλακτική λειτουργία
PD7	OC2A (Έξοδος Α του συγκριτή του Timer/Counter2) PCINT31 (Ακροδέκτης της αλλαγής διακοπής 31)
PD6	ICP1 (Είσοδος του Timer/Counter1) OC2B (Έξοδος Β του συγκριτή του Timer/Counter2) PCINT30 (Ακροδέκτης της αλλαγής διακοπής 30)
PD5	OC1A (Έξοδος Α του συγκριτή του Timer/Counter1) PCINT29 (Ακροδέκτης της αλλαγής διακοπής 29)
PD4	OC1B (Έξοδος Β του συγκριτή του Timer/Counter1) PCINT28 (Ακροδέκτης της αλλαγής διακοπής 28)
PD3	INT1 (Είσοδος της εξωτερικής διακοπής 1) PCINT27 (Ακροδέκτης της αλλαγής διακοπής 27)
PD2	INT0 (Είσοδος της εξωτερικής διακοπής 0) PCINT26 (Ακροδέκτης της αλλαγής διακοπής 26)
PD1	TXD0 (Ακροδέκτης (εξόδου) της USART) PCINT25 (Ακροδέκτης της αλλαγής διακοπής 25)
PD0	RXD0 (Ακροδέκτης (εισόδου) της USART) PCINT24 (Ακροδέκτης της αλλαγής διακοπής 24)

Κεφάλαιο 6

Σύνδεση της πλακέτας με ηλεκτρονικό υπολογιστή

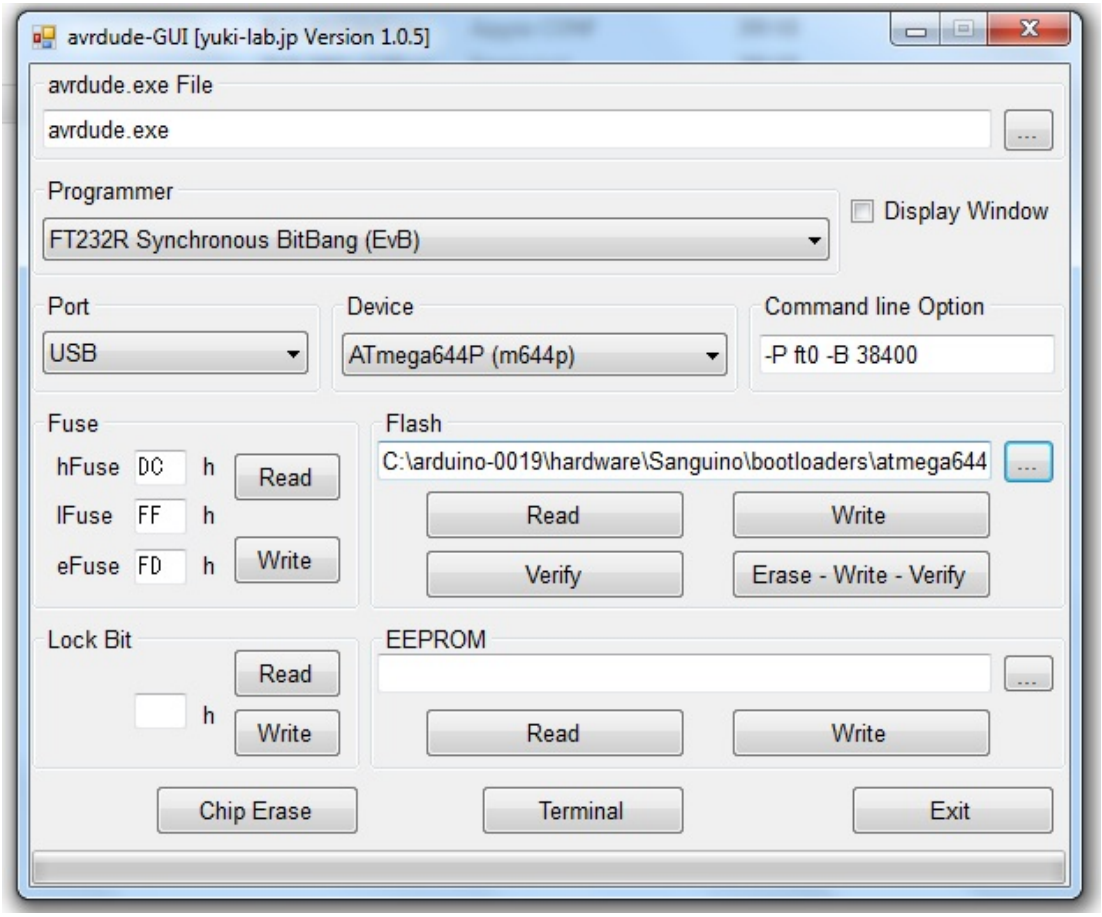
6.1 Σύνδεση της πλακέτας

1. Συνδέουμε το μικροεπεξεργαστή με την ISP σύνδεση.



Programmer	ISP port
TS	MISO
SR	SCK
CD	MOSI
RI	RESET

2. Συνδέουμε την πλακέτα με τη θύρα USB του υπολογιστή μας μέσω ενός καλωδίου. Την πρώτη φορά που γίνεται η σύνδεση της πλακέτας με τον υπολογιστή, το σύστημα μας ζητά τους drivers, προκειμένου να πραγματοποιηθεί η εγκατάσταση. Οι drivers υπάρχουν στο site της FTDI: http://www.ftdichip.com/Drivers/VCP.htm
3. Κατεβάζουμε και εγκαθιστούμε το λογισμικό AVRDUDE: www.and-tech.pl/files/EvBISP.zip.
4. Ανοίγουμε την εφαρμογή avrdude-GUI.exe.
5. Στο παράθυρο που μας ανοίγει, στο πλαίσιο Programmer, επιλέγουμε FT232R Synchronous BitBang (EvB).
6. Στο πλαίσιο Port, επιλέγουμε την εικονική θύρα (στη δική μας περίπτωση επιλέγουμε USB, η οποία αντιστοιχεί στην COM3).
7. Στη συνέχεια, στο πλαίσιο Device επιλέγουμε τη συσκευή μας (ATmega644p).
8. Στο πλαίσιο Command line Option, γράφουμε -P ft0 -B 38400.
9. Στο πλαίσιο Fuse, οι τιμές στα πεδία είναι: hFuse: DC, IFuse: FF και eFuse: FD και στη συνέχεια, για να ελέγξουμε τη σύνδεση, πατάμε Write.
10. Στο πλαίσιο Flash, επιλέγουμε το αρχείο bootloader, που αντιστοιχεί στη δική μας συσκευή (ATmegaBOOT_644P.hex), και στο τέλος πατάμε Erase - Write - Verify. Το αρχείο bootloader είναι διαθέσιμο στη σελίδα http://and-tech.pl/files/bootloader/bootloader_m644p.zip.



6.2 Αποσύνδεση της πλακέτας από τον υπολογιστή

Εφόσον όλα έχουν γίνει σωστά και δεν εμφανιστεί κάποιο μήνυμα σφάλματος, κλείνουμε την εφαρμογή πατώντας Exit. Τέλος, για την αποσύνδεση της συσκευής μας από τον υπολογιστή, πρέπει πρώτα να αποσυνδέσουμε το καλώδιο USB και στη συνέχεια τα υπόλοιπα καλώδια από την πλακέτα. Πλέον, η συσκευή είναι έτοιμη για προγραμματισμό και για τη σύνδεση της πλακέτας με τον υπολογιστή θα χρειάζεται μόνο το καλώδιο USB[1].

[1] Αργότερα, προκειμένου να τρέξουμε τα παραδείγματα που είναι γραμμένα σε γλώσσα Assembly στο Atmel Studio 6.1, είναι αναγκαίο να γίνει ξανά η σύνδεση όπως στο βήμα 1. Παράλληλα, στο λογισμικό AVRDUDE θα πρέπει να φορτώσουμε το hex αρχείο, που δημιουργείται, μόλις τρέξουμε τον κώδικα στο Atmel Studio 6.1.

Κεφάλαιο 7

Atmel Studio 6.1

7.1 Εισαγωγικά

Στο συγκεκριμένο κεφάλαιο θα γίνει αναφορά στο λογισμικό το λογισμικό που χρησιμοποιήθηκε για την υλοποίηση των εργαστηριακών ασκήσεων του μαθήματος «Αρχιτεκτονική Η/Υ», το Atmel Studio 6.1. Πρόκειται για το Atmel Studio, το οποίο βρίσκεται στην ιστοσελίδα της Atmel http://www.atmel.com/tools/atmelstudio.aspx[2], διατίθεται δωρεάν και η εγκατάστασή του στον υπολογιστή πραγματοποιείται με ιδιαίτερη ευκολία. Το λογισμικό αυτό περιλαμβάνει ένα συντάκτη – διορθωτή (editor), ένα συμβολομεταφραστή (assembler) και έναν αποσφαλματωτή (debugger). Επίσης, επιτρέπει την αξιοποίηση κάθε προσομοιωτή AVR.

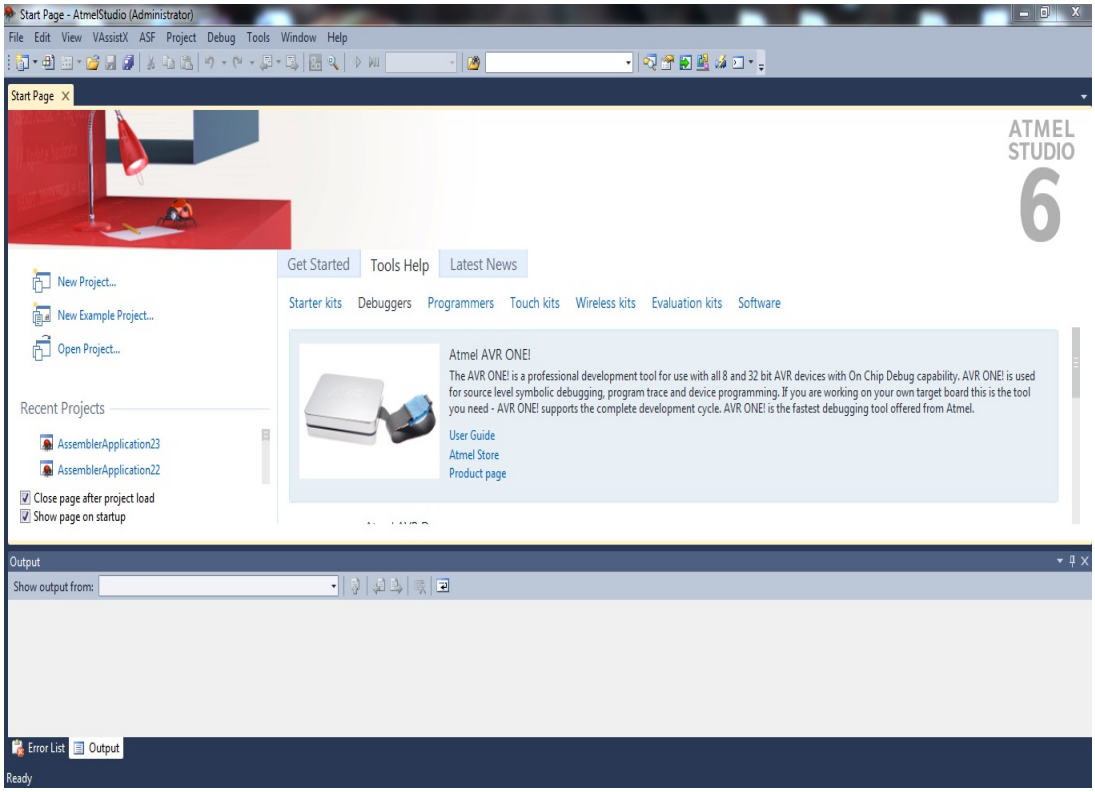
Η γλώσσα που χρησιμοποιήθηκε για την υλοποίηση των εργαστηριακών ασκήσεων είναι η assembly. Πρόκειται για μία γλώσσα χαμηλού επιπέδου, που επιτρέπει πρόσβαση στις λειτουργίες του επεξεργαστή. Είναι, δηλαδή, μία γλώσσα που βρίσκεται πολύ κοντά στη γλώσσα μηχανής και στο υλικό του υπολογιστή. Άλλωστε, κάθε συγκεκριμένη αρχιτεκτονική συνόλου εντολών, δηλαδή κάθε οικογένεια επεξεργαστών, έχει τη δική της συμβολική γλώσσα, η οποία δίνεται, συνήθως, από τον κατασκευαστή της.

7.2 Περιβάλλον λογισμικού του Atmel Studio 6.1

Μετά την εγκατάσταση του λογισμικού, ανοίγοντάς το παρατηρείται η αρχική σελίδα του λογισμικού, όπως φαίνεται στην παρακάτω εικόνα.

[2] Από το συγκεκριμένο link, αφού επιλέξουμε το software **Atmel Studio 6.1 update 2.0 (build 2730) Installer**, οδηγούμαστε στην επόμενη σελίδα http://www.atmel.com/System/BaseForm.aspx?target=tcm:26-49768. Σ' αυτή τη σελίδα συμπληρώνουμε τα πεδία με τα στοιχεία μας, πατάμε Submit και στο επόμενο link, http://www.atmel.com/forms/software-download.aspx?target=tcm:26-49768, επιλέγοντας το software, θα αρχίσει να «κατεβαίνει» το λογισμικό στον υπολογιστή μας.

Εικόνα 1: αρχική σελίδα Atmel Studio 6.1

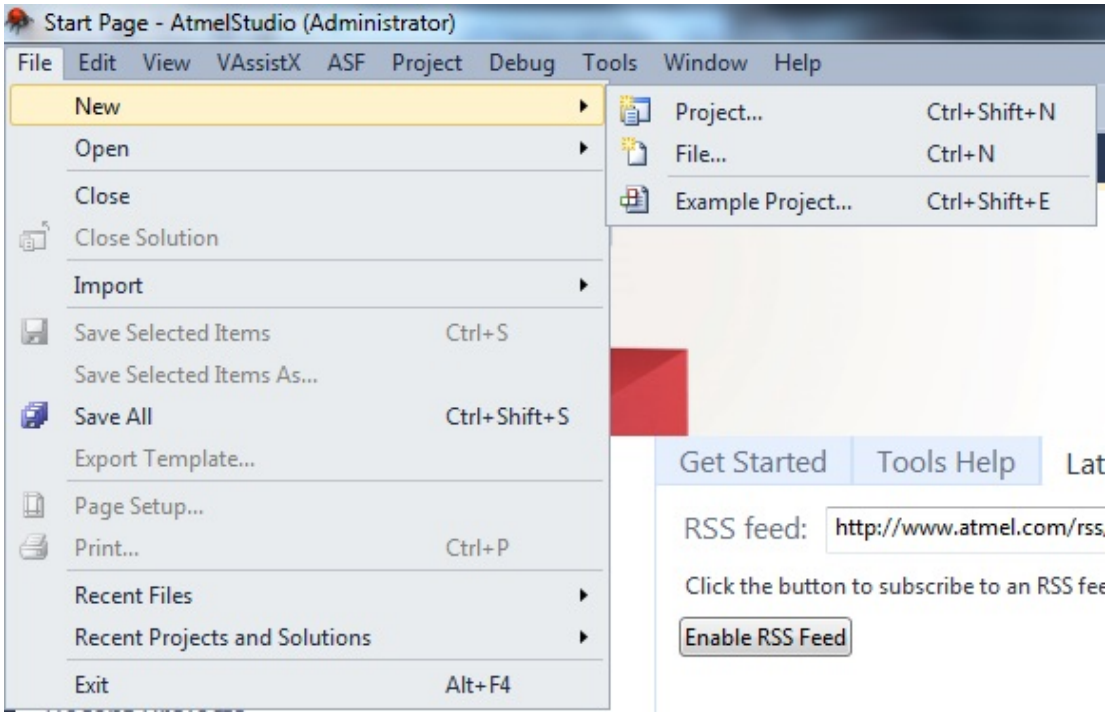


Στην επιλογή Get Started υπάρχουν πληροφορίες για το λογισμικό μας. Η επιλογή Tools Help περιέχει πληροφορίες σχετικά με τις πλακέτες που υποστηρίζει το λογισμικό, τους programmers, τους debuggers αλλά και το software. Η επιλογή Latest News περιλαμβάνει τις τελευταίες ενημερώσεις του λογισμικού.

1. Δημιουργία νέου project:

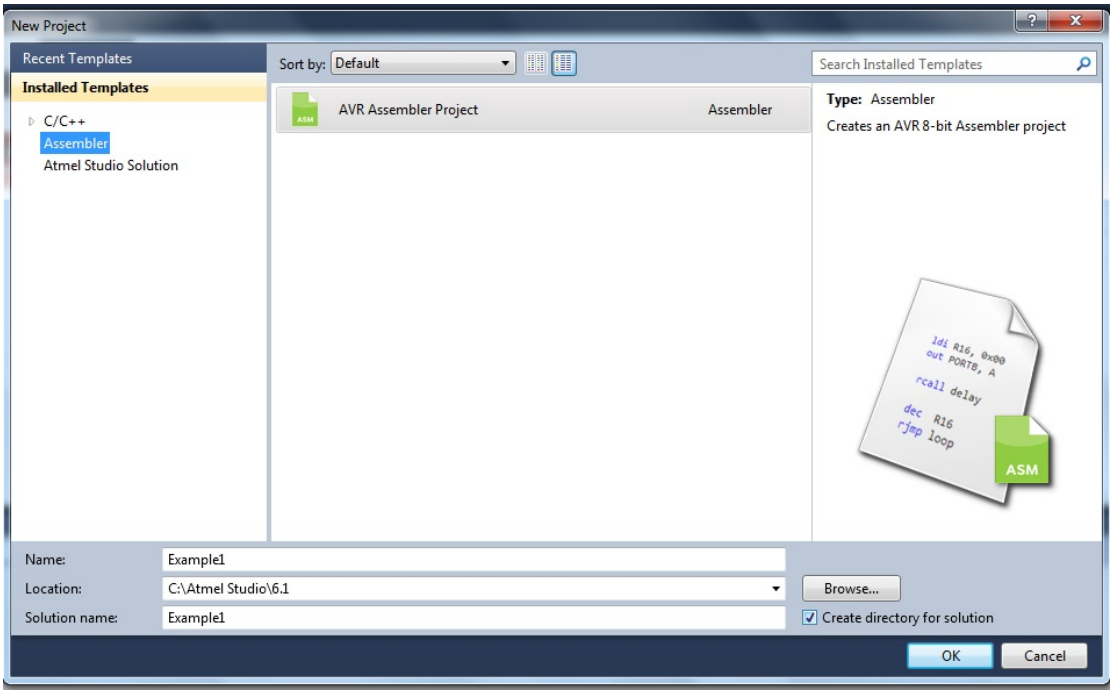
   Προκειμένου να δημιουργήσουμε νέο Project, από τη γραμμή μενού του λογισμικού επιλέγουμε File ⇨ New ⇨ Project (εικόνα 2). Επίσης, η επιλογή File περιλαμβάνει τις επιλογές της εκτύπωσης, της αποθήκευσης, του ανοίγματος μιας υπάρχουσας εργασίας, του κλεισίματος του υπάρχοντος project και της εξόδου από το λογισμικό.

Εικόνα 2: δημιουργία νέου project



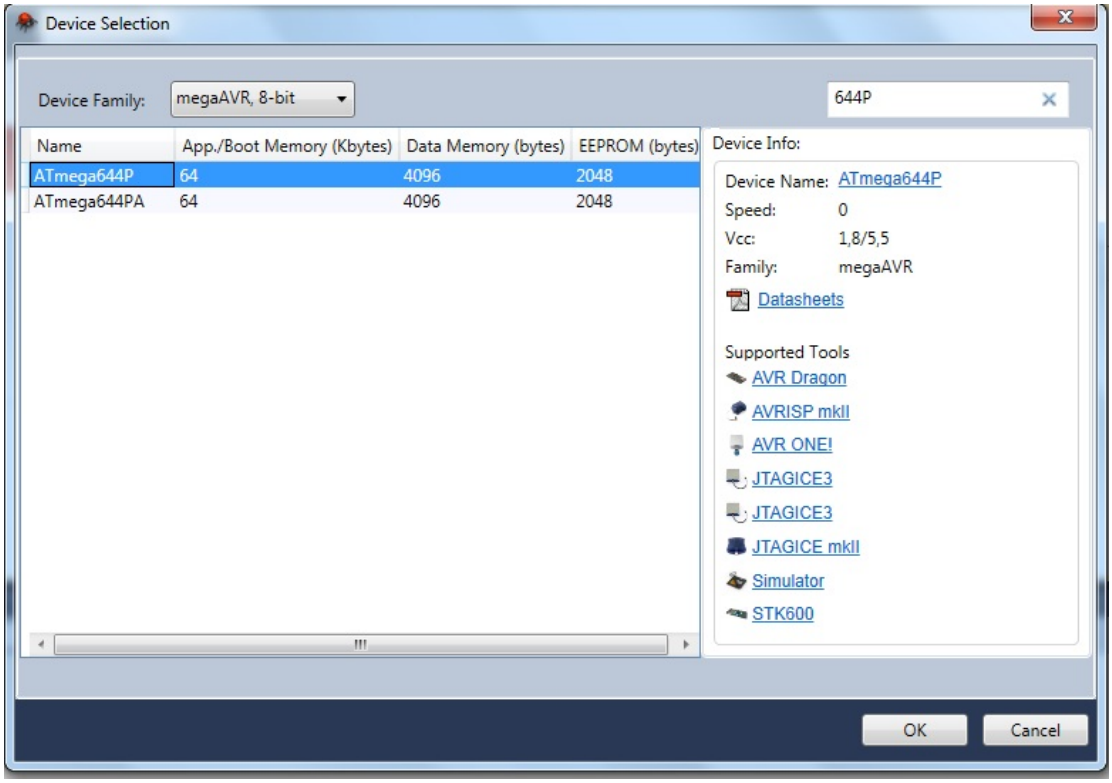
2. Στη συνέχεια, επιλέγουμε τη γλώσσα που θα χρησιμοποιήσουμε στις ασκήσεις μας (assembler), δίνουμε το όνομα της άσκησής μας (π.χ. Example1) και ορίζουμε την τοποθεσία όπου θα αποθηκεύονται οι ασκήσεις μας [(π.χ. C:\Atmel Studio\6.1), εικόνα 3].

Εικόνα 3: επιλογή γλώσσας, όνομα αρχείου, τοποθεσία αποθήκευσης



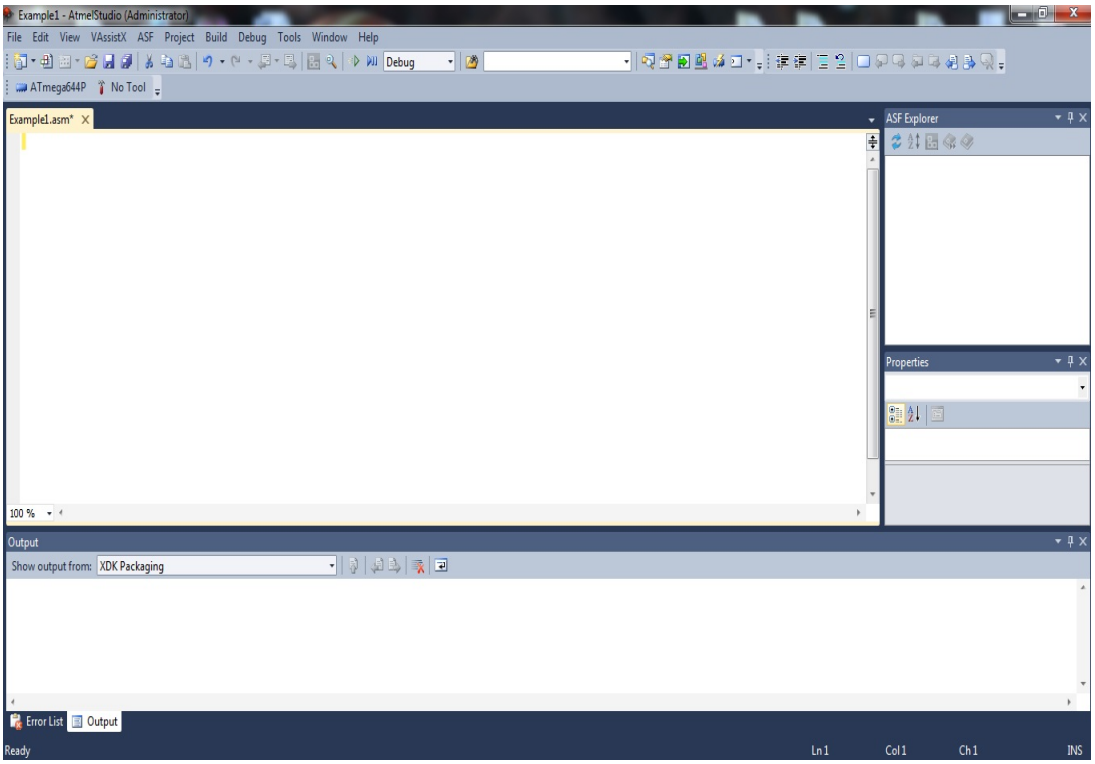
3. Αφού ολοκληρώσουμε το προηγούμενο βήμα, πατάμε OK και οδηγούμαστε στο επόμενο παράθυρο, όπου επιλέγουμε το μικροελεγκτή τον οποίο θέλουμε να προγραμματίσουμε (ATmega644P). Από το πλαίσιο Device Family, μπορούμε να επιλέξουμε την κατηγορία στην οποία ανήκει ο μικροελεγκτής μας (megaAVR, 8 – bit). Εξαιτίας των πολλών και διαφόρων μικροελεγκτών που υπάρχουν, για λόγους συντομίας μπορούμε να γράψουμε στο πλαίσιο αναζήτησης του συγκεκριμένου παραθύρου 644P, όπου θα μας εμφανίσει 2 επιλογές, και από αυτές τις 2 επιλέγουμε το δικό μας μικροελεγκτή (εικόνα 4). Αφού επιλέξουμε το μικροελεγκτή, εμφανίζονται πληροφορίες που σχετίζονται μ' αυτόν.

Εικόνα 4: επιλογή μικροελεγκτή



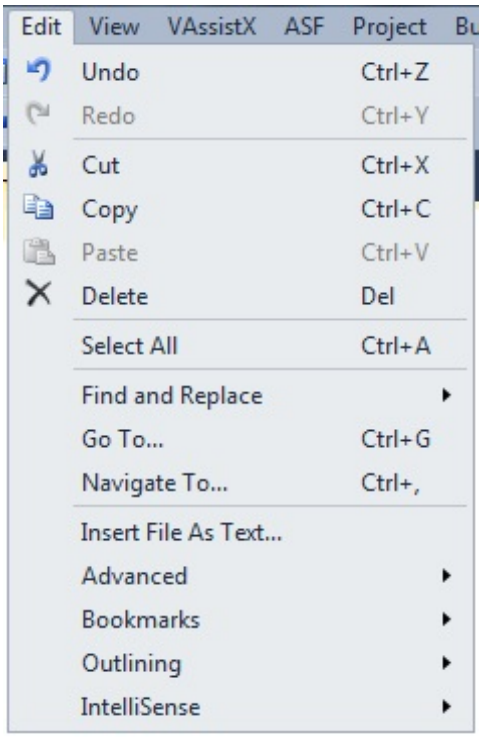
4. Στη συνέχεια, αφού έχουμε επιλέξει και το μικροελεγκτή, πατάμε OK και, πλέον, βλέπουμε την οθόνη εργασίας του λογισμικού (εικόνα 5).

Εικόνα 5: οθόνη εργασίας



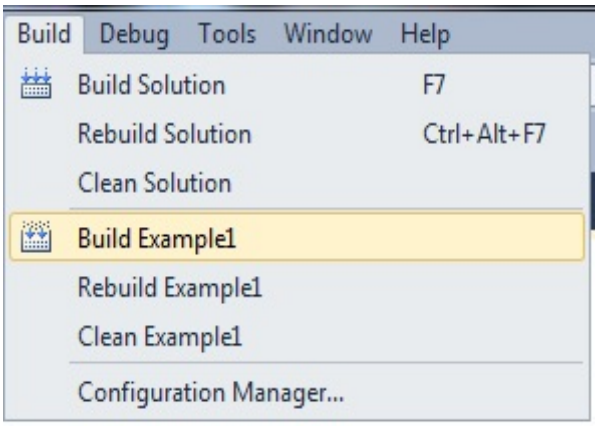
Η γραμμή μενού του λογισμικού περιλαμβάνει ορισμένες επιλογές. Η επιλογή File έχει αναφερθεί παραπάνω. Η επιλογή Edit περιλαμβάνει λειτουργίες επεξεργασίας του κειμένου [(cut, copy, paste, delete, select all, undo, redo κ.ά.), εικόνα 6].

Εικόνα 6: επιλογή Edit

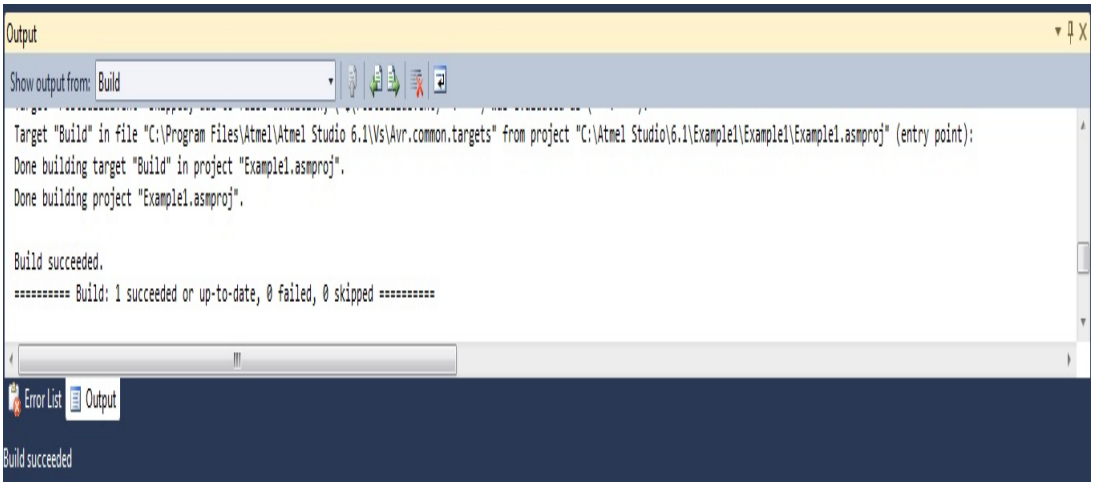


Αφού γράψουμε το πρόγραμμα στον editor του Atmel Studio, από την επιλογή Build (εικόνα 7) επιλέγουμε την εντολή Build Example1, για να δημιουργηθεί το εκτελέσιμο αρχείο του προγράμματός μας. Αν ο κώδικάς μας έχει συντακτικά σφάλματα, αυτά θα φανούν στο παράθυρο εξόδου του μεταγλωττιστή, όπως φαίνεται στην εικόνα 7.

Εικόνα 7: επιλογή Build



Εικόνα 8: παράθυρο εξόδου

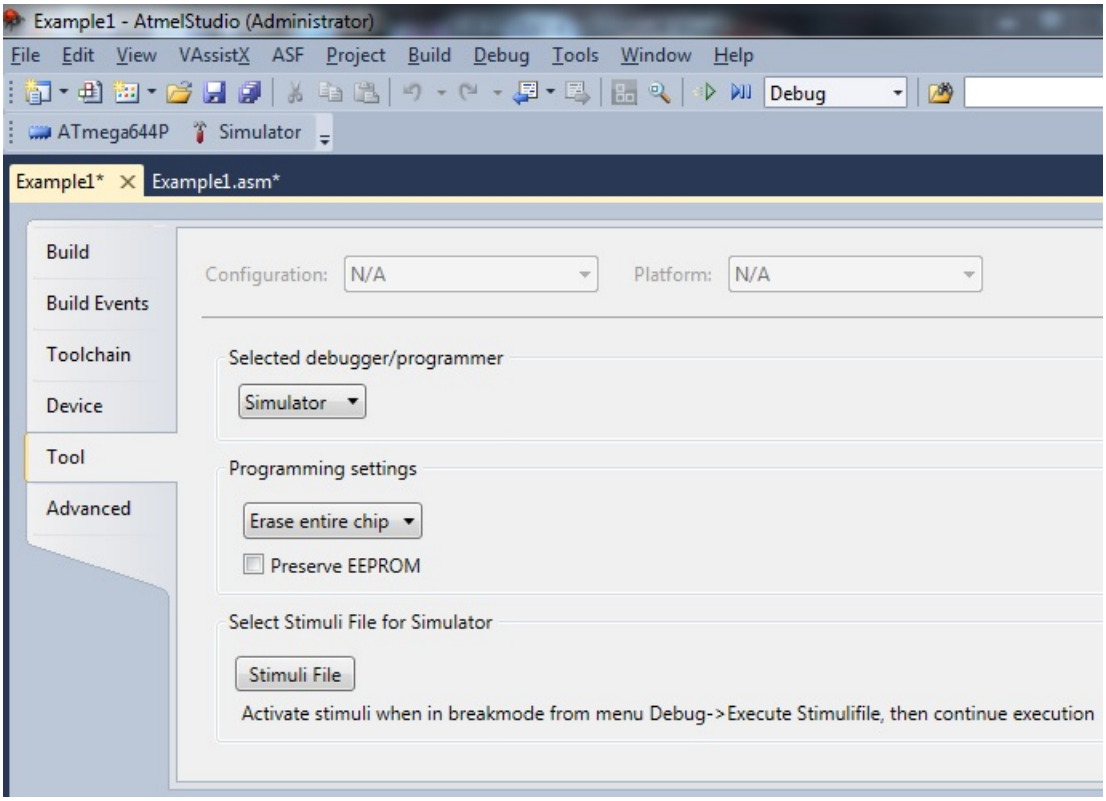


Αφού έχει δημιουργηθεί το εκτελέσιμο αρχείο του προγράμματος, μπορούμε να τρέξουμε το πρόγραμμά μας. Πριν από αυτό, όμως, θα πρέπει να επιλέξουμε τον προσομοιωτή (simulator), ώστε να μπορέσουμε να ελέγξουμε τα αποτελέσματα του προγράμματος. Ακριβώς πάνω από τον editor, πατάμε το No tool (εικόνα 9).

Εικόνα 9: επιλογή No tool

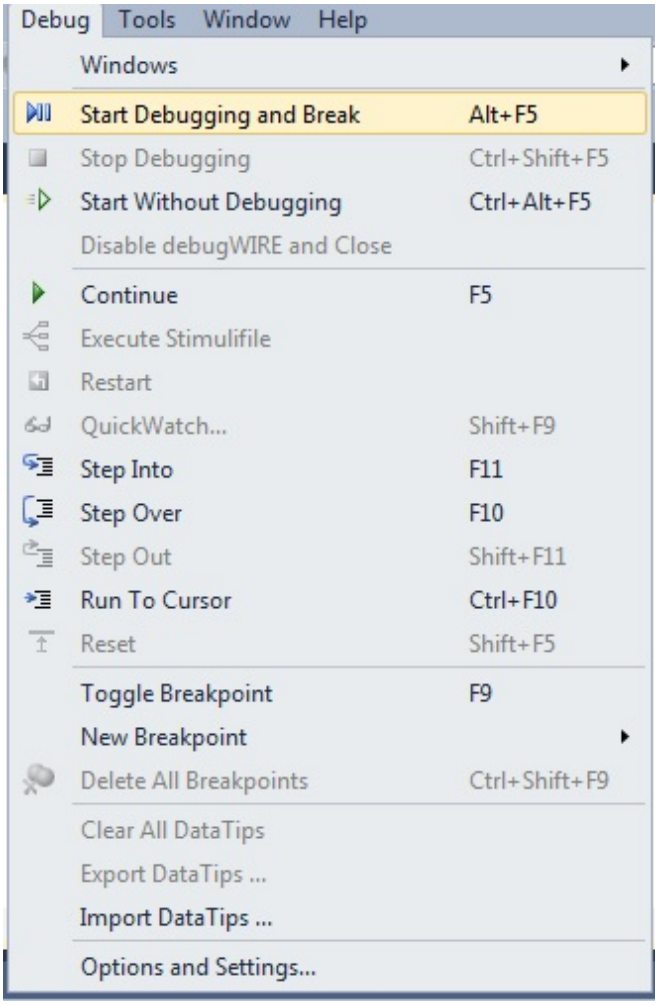
Στη συνέχεια, στο νέο παράθυρο, στην επιλογή Selected debugger/programmer, επιλέγουμε το Simulator (εικόνα 10).

Εικόνα 10: επιλογή Simulator



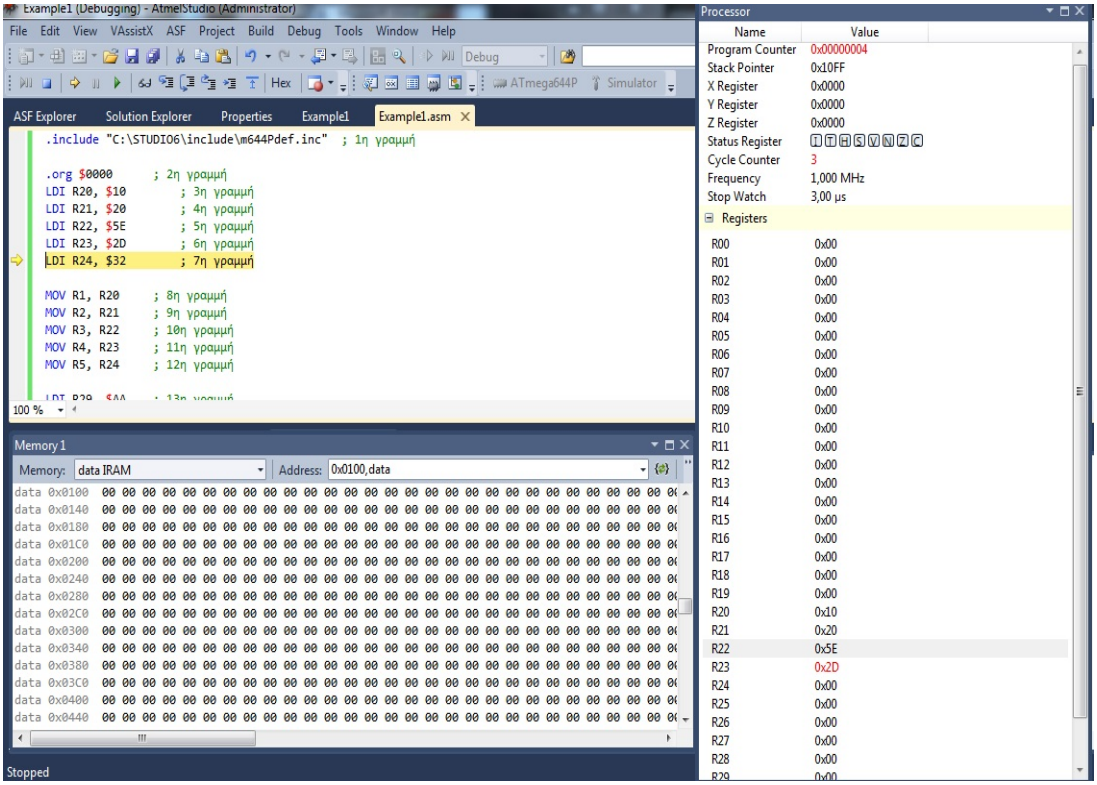
Αφού επιλέξουμε το Simulator, επιστρέφουμε στο αρχείο Example1.asm. Πλέον, είμαστε έτοιμοι, για να τρέξουμε το πρόγραμμά μας. Από την επιλογή Debug, πατάμε Start Debugging and break ή Alt+F5 (εικόνα 11) και αρχίζει η εκτέλεση του προγράμματος. Στην ίδια εικόνα, μπορούμε να δούμε ότι για τη βηματική εκτέλεση του προγράμματος πατάμε F11, προκειμένου να σταματήσουμε το πρόγραμμα, πατάμε Stop Debugging, για τη συνέχιση του προγράμματος επιλέγουμε Continue ή F5 και για την επανεκκίνησή του επιλέγουμε Restart.

Εικόνα 11: επιλογή Debug



Κατά τη διάρκεια της εκτέλεσης του προγράμματος, εμφανίζονται δύο παράθυρα (εικόνα 12). Το πρώτο παράθυρο (Processor) δείχνει την κατάσταση του επεξεργαστή και το δεύτερο (Memory 1) δείχνει τις τιμές της μνήμης του προγράμματος που εκτελούμε. Στο παράθυρο Processor παρατηρούμε ότι υπάρχει ο μετρητής προγράμματος (Program Counter), ο Stack Pointer, οι καταχωρητές X, Y, και Z, οι σημαίες (Status Register), ο κύκλος ρολογιού (Cycle Counter), η συχνότητα (Frequency) και οι 32 καταχωρητές (R00 – R31). Στο παράθυρο της μνήμης, προκειμένου να δούμε τις τιμές που έχουν οι θέσεις μνήμης του προγράμματός μας στο πλαίσιο Memory, επιλέγουμε data IRAM.

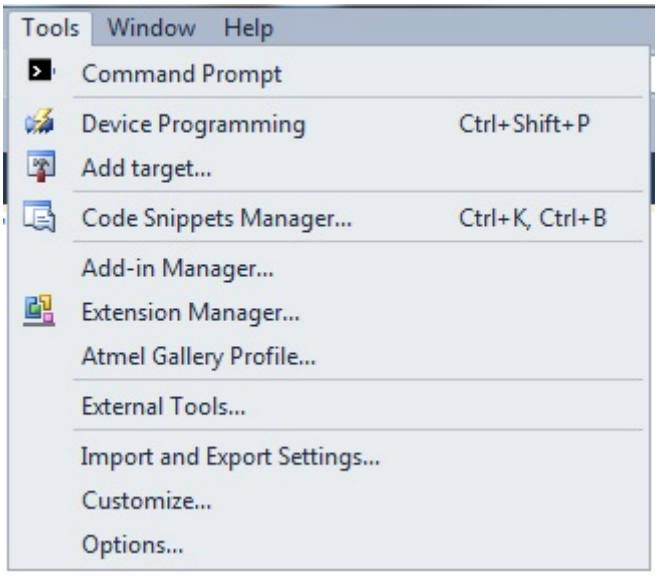
Εικόνα 12: παράθυρα Processor – Memory 1



Επίσης, όπως μπορούμε να παρατηρήσουμε στην παραπάνω εικόνα, όταν έχουμε νέες τιμές στους καταχωρητές ή σε μία θέση μνήμης (το ίδιο ισχύει για τις σημαίες, για το Program Counter αλλά και για όλα τα υπόλοιπα), οι τιμές αυτές επισημαίνονται με κόκκινο χρώμα (στην εικόνα 12 έχει αλλάξει η τιμή του καταχωρητή R23 και, πλέον, έχει την τιμή $2D).

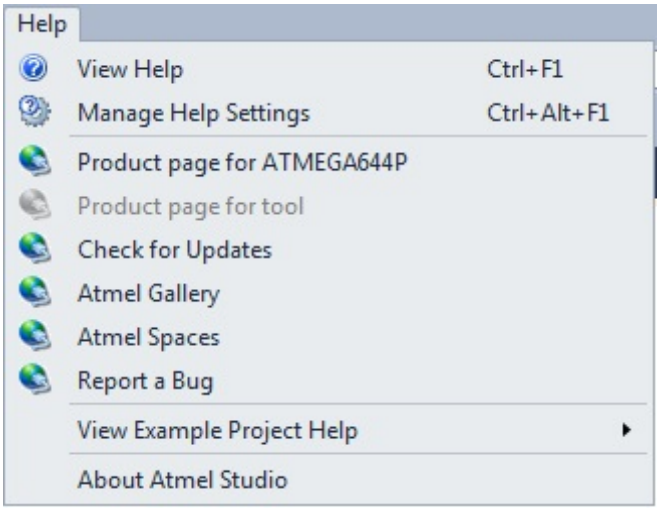
Ακόμη, στην επιλογή Tools υπάρχουν λειτουργίες για την ενεργοποίηση των εργαλείων εξομοίωσης του AVR (εικόνα 13).

Εικόνα 13: επιλογή Tools



Επίσης, η επιλογή Help (εικόνα 14) περιέχει θέματα βοήθειας που αφορούν το μικροελεγκτή και το Atmel Studio, ενώ ελέγχει για τυχόν ενημερώσεις κ.ά.

Εικόνα 14: επιλογή Help



Κεφάλαιο 8

Ασκήσεις

8.1 Εισαγωγικά

Σ' αυτό το κεφάλαιο θα εφαρμοστούν εργαστηριακές ασκήσεις του μαθήματος «Αρχιτεκτονική Η/Υ[3]», χρησιμοποιώντας τη γλώσσα assembly. Σε πολλές ασκήσεις θα ζητηθεί το τελικό αποτέλεσμα να εμφανιστεί στους λαμπτήρες της πλακέτας. Ο ένας τρόπος είναι να χρησιμοποιούμε την τεχνική συμπληρώματος ως προς ένα (εντολή COM) και ο δεύτερος τρόπος είναι να χρησιμοποιούμε τη λογική πράξη XOR (εντολή EOR). Και στις δύο περιπτώσεις δίνουμε στον καταχωρητή διεύθυνσης DDRxn την τιμή 1 (FF), ώστε να ορίζονται οι ακροδέκτες της θύρας ως έξοδοι. Επιπλέον, προκειμένου να δούμε τους λαμπτήρες να λειτουργούν, θα πρέπει να φορτώσουμε το .hex αρχείο που δημιουργεί το Atmel Studio στο λογισμικό avrdude.

8.2 Εργαστήριο 1

Εισαγωγικά: Στο πρώτο εργαστηριακό μάθημα γίνεται μία γνωριμία με τις εντολές MOV, LDI, LDS και STS.

Θεωρία:

Εντολή MOV (MOV καταχωρητής 1, καταχωρητής 2): Με την εντολή MOV (Move between registers) γίνεται μεταφορά (αντιγραφή) των περιεχομένων μεταξύ δύο καταχωρητών.

Σύνταξη: MOV Rd, Rr

Παράδειγμα: MOV R1, R10 (το περιεχόμενο του καταχωρητή R10 μεταφέρεται στον καταχωρητή R1)

Εντολή LDS (LDS καταχωρητής, θέση μνήμης): Με την εντολή LDS (Load direct from SRAM) γίνεται μετακίνηση του περιεχομένου της θέσης μνήμης στον καταχωρητή.

Σύνταξη: LDS Rd, k

[3] Σπύρος Καζαρλής, *Σημειώσεις Εργαστηρίου Αρχιτεκτονικής Υπολογιστών, Εκπαιδευτικό Σύστημα BGG-8088, Επίσημες Σημειώσεις για το Εργαστηριακό μάθημα «Αρχιτεκτονική Η/Υ» του Δ Εξαμήνου του Τμήματος Πληροφορικής & Επικοινωνιών της Σχολής Σ.Τ.Ε.Φ. του Τ.Ε.Ι. Σερρών*, Τ.Ε.Ι. Σερρών, Σεπτέμβριος 2004

Παράδειγμα: LDS R1, $0200

Εντολή STS (STS θέση μνήμης, καταχωρητής): Με την εντολή STS (Store direct to SRAM) γίνεται μετακίνηση του περιεχομένου του καταχωρητή στη θέση μνήμης.

Σύνταξη: STS k, Rr

Παράδειγμα: STS $20, R1

Εντολή LDI (LDI καταχωρητής, αριθμός): Με την εντολή LDI (Load immediate) δίνεται στον καταχωρητή ένας αριθμός.

Σύνταξη: LDI Rd, K

Παράδειγμα: LDI R30, $30

Σημείωση 1: Η εντολή LDI μπορεί να χρησιμοποιήσει καταχωρητές από R16 έως R31.

Σημείωση 2**:** Ο καταχωρητής Rd ονομάζεται καταχωρητής «προορισμού» (και «πηγή») και ο καταχωρητής Rr ονομάζεται καταχωρητής «πηγή».

Άσκηση 1.1: Να γράψετε ένα πρόγραμμα το οποίο να πραγματοποιεί χρήση των εντολών LDI, MOV, STS και LDS. Το πρόγραμμα να αρχίζει από τη διεύθυνση 0000.

Πρόγραμμα:

```
.include "C:\STUDIO6\include\m644Pdef.inc"        ; 1η γραμμή

.org $0000              ; 2η γραμμή
LDI R20, $10            ; 3η γραμμή
LDI R21, $20            ; 4η γραμμή
LDI R22, $5E            ; 5η γραμμή
LDI R23, $2D            ; 6η γραμμή
LDI R24, $32            ; 7η γραμμή

MOV R1, R20             ; 8η γραμμή
MOV R2, R21             ; 9η γραμμή
MOV R3, R22             ; 10η γραμμή
MOV R4, R23             ; 11η γραμμή
MOV R5, R24             ; 12η γραμμή
```

```
LDI R29, $AA        ; 13η γραμμή
LDI R30, $BB        ; 14η γραμμή
LDI R31, $77        ; 15η γραμμή
STS $0200, R29      ; 16η γραμμή
STS $0240, R30      ; 17η γραμμή
STS $0280, R31      ; 18η γραμμή

LDS R10, $0280      ; 19η γραμμή
LDS R11, $0300      ; 20η γραμμή
LDS R15, $03C0      ; 21η γραμμή
LDS R16, $0440      ; 22η γραμμή
```

Επεξήγηση: Η εντολή LDI δίνει μία τιμή στους καταχωρητές. Η εντολή MOV μεταφέρει (αντιγράφει) το περιεχόμενο ενός καταχωρητή σε άλλον καταχωρητή. Η εντολή STS μεταφέρει το περιεχόμενο ενός καταχωρητή σε μία θέση μνήμης της SRAM και η εντολή LDS μεταφέρει το περιεχόμενο της μνήμης σε έναν καταχωρητή.

8.3 Εργαστήριο 2

Εισαγωγικά: Στο συγκεκριμένο εργαστηριακό μάθημα θα δούμε τα συστήματα αρίθμησης δεκαδικό, δυαδικό και δεκαεξαδικό, πώς γίνονται οι μετατροπές αριθμών από το ένα σύστημα σε κάποιο άλλο, την πρόσθεση αριθμών και τις εντολές ADD, ADC, ADIW, SEC και CLC.

Θεωρία:

Δεκαδικό σύστημα αρίθμησης: Στο συγκεκριμένο σύστημα η «βάση αρίθμησης» είναι το 10, επομένως για την αναπαράσταση των αριθμών χρησιμοποιούνται δέκα διαφορετικά σύμβολα (οι αριθμοί από 0 έως 9). Επίσης, το κάθε ψηφίο ενός αριθμού πολλαπλασιάζεται με αυξανόμενες δυνάμεις του 10, π.χ. ο αριθμός $348_{10} = 3 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$.

Δυαδικό σύστημα αρίθμησης: Το δυαδικό σύστημα αρίθμησης χρησιμοποιείται στους ηλεκτρονικούς υπολογιστές, η «βάση αρίθμησης» είναι το 2 και χρησιμοποιεί δύο μόνο διαφορετικά σύμβολα, το 0 και το 1. Η αναπαράσταση των δύο αυτών συμβόλων ηλεκτρονικά μπορεί να γίνει είτε με διακόπτες on – off είτε

με 0V ή 5V σε επίπεδο τάσης. Κάθε ψηφίο ενός δυαδικού αριθμού πολλαπλασιάζεται με αυξανόμενες δυνάμεις του 2, π.χ. $0011_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10}$.

Δεκαεξαδικό σύστημα αρίθμησης: Ένα άλλο σύστημα αρίθμησης που χρησιμοποιείται στους υπολογιστές είναι το δεκαεξαδικό σύστημα και αυτό συμβαίνει, διότι οι δεκαεξαδικοί αριθμοί μετατρέπονται με ευκολία σε δυαδικούς αριθμούς και αντίστροφα και, επίσης, Επίσης, το συγκεκριμένο σύστημα αρίθμησης χρησιμοποιείται για την αναπαράσταση μεγάλων αριθμών με λίγα ψηφία. Το σύστημα αυτό έχει ως «βάση αρίθμησης» το 16 και για την αναπαράσταση των αριθμών διατίθενται 16 διαφορετικά σύμβολα (οι αριθμοί από 0 έως 9 και τα γράμματα A, B, C, D, E, F). Στη συγκεκριμένη περίπτωση, το κάθε ψηφίο ενός τέτοιου αριθμού πολλαπλασιάζεται με αυξανόμενες δυνάμεις του 16, π.χ. $3BD_{16} = 3 \times 16^2 + B \times 16^1 + D \times 16^0 = 3 \times 256 + 11 \times 16 + 13 = 768 + 176 + 13 = 957_{10}$.

Μετατροπή ενός δυαδικού αριθμού σε δεκαδικό αριθμό: Για τη μετατροπή ενός δυαδικού αριθμού σε δεκαδικό, έχουμε έναν πίνακα όπου σε κάθε θέση bit αντιστοιχεί και μία δύναμη του 2, η οποία, όταν το αντίστοιχο bit είναι 1, θα προστίθεται, ενώ, όταν το αντίστοιχο bit θα είναι 0, δεν θα προστίθεται.

Θέση bit	7	6	5	4	3	2	1	0
Δύναμη του 2	128	64	32	16	8	4	2	1
Δυαδικός αριθμός	0	0	1	1	1	1	0	1

Επομένως, ο δυαδικός αριθμός 00111101_2 είναι: $32 + 16 + 8 + 4 + 1 = 61_{10}$.

Μετατροπή ενός δεκαδικού αριθμού σε δυαδικό αριθμό: Η μετατροπή ενός δεκαδικού αριθμού σε δυαδική μορφή γίνεται διαιρώντας το δεκαδικό αριθμό με το 2, ενώ μετά από κάθε διαίρεση, διαιρούμε το πηλίκο που προκύπτει (από την προηγούμενη διαίρεση) ξανά με το 2. Η διαδικασία της διαίρεσης συνεχίζεται, μέχρι το πηλίκο να γίνει 0. Ακόμη, σε κάθε διαίρεση διατηρούμε το υπόλοιπο που προκύπτει και το υπόλοιπο μπορεί να είναι ή 0 ή 1.

Διαίρεση	Πηλίκο	Υπόλοιπο
93/2	46	1
46/2	23	0
23/2	11	1
11/2	5	1
5/2	2	1
2/2	1	0
1/2	0	1

Επομένως, ο αριθμός 93_{10} σε δυαδική μορφή είναι: 1011101_2 (παίρνουμε τα υπόλοιπα των διαιρέσεων, ξεκινώντας από κάτω προς τα πάνω).

Μετατροπή ενός δεκαεξαδικού αριθμού σε δυαδικό αριθμό: Κάθε ψηφίο ενός δεκαεξαδικού αριθμού στη δυαδική μορφή αντιστοιχεί σε τέσσερα ψηφία. Για παράδειγμα, ένας δυαδικός αριθμός των 8bit στο δεκαεξαδικό σύστημα αρίθμησης αναπαρίσταται με 2 ψηφία, π.χ. $255_{10} = 1111\ 1111_2 = FF_{16}$. Επομένως, ο αριθμός B5 μετατρέπεται σε δυαδική μορφή με την εξής διαδικασία:

Δεκαεξαδικός αριθμός	$B5_{16}$	
Σε 16δικό σύστημα/ψηφίο	B	5
Σε 10δικό σύστημα/ψηφίο	11	5
Σε 2δικό σύστημα/ψηφίο	1011	0101
Δυαδικός αριθμός	1011 0101	

Επομένως, ο δεκαεξαδικός αριθμός $B5_{16}$ στο δυαδικό σύστημα είναι ο αριθμός $1011\ 0101_2$.

Μετατροπή ενός δυαδικού αριθμού σε δεκαεξαδικό: Η μετατροπή ενός δυαδικού αριθμού σε δεκαεξαδικό γίνεται με τον αντίστροφο τρόπο. Έστω, ότι έχουμε το δυαδικό αριθμό 11100011_2, η μετατροπή του θα είναι:

Δυαδικός αριθμός	11100011_2	
Χωρισμός ανά 4 ψηφία	1110	0011
Σε 10δικό σύστημα/ψηφίο	14	3
Σε 16δικό σύστημα/ψηφίο	E	3
Δεκαεξαδικός αριθμός	E3	

Επομένως, ο δυαδικός αριθμός 11100011_2 στο δεκαεξαδικό σύστημα είναι ο αριθμός $E3_{16}$.

Εντολές που χρησιμοποιούνται:

Εντολή ADD (ADD καταχωρητής 1, καταχωρητής 2): Με την εντολή ADD (ADD two registers) πραγματοποιείται η πρόσθεση του περιεχομένου του ενός καταχωρητή με το περιεχόμενο του άλλου καταχωρητή, χωρίς κρατούμενο, και το αποτέλεσμα της πρόσθεσης τοποθετείται στον πρώτο καταχωρητή.

Σύνταξη: ADD Rd, Rr

Παράδειγμα: ADD R2, R31 (το αποτέλεσμα της πρόσθεσης θα τοποθετηθεί στον καταχωρητή R1).

Εντολή ADC (ADC καταχωρητής 1, καταχωρητής 2): Με την εντολή ADC (ADC with carry two registers) πραγματοποιείται πρόσθεση του περιεχομένου του ενός καταχωρητή με το περιεχόμενο του άλλου καταχωρητή, χρησιμοποιώντας κρατούμενο, και το αποτέλεσμα της πρόσθεσης τοποθετείται στον πρώτο καταχωρητή.

Σύνταξη: ADC Rd, Rr

Παράδειγμα: ADC R2, R31 (το αποτέλεσμα της πρόσθεσης μεταφέρεται στον καταχωρητή R1).

Εντολή ADIW (ADIW καταχωρητής 1:καταχωρητής 2, τιμή: Με την εντολή ADIW (ADD immediate to word) γίνεται η πρόσθεση μεταξύ μίας τιμής και ενός ζεύγους καταχωρητών, με το αποτέλεσμα της πρόσθεσης να τοποθετείται στους δύο καταχωρητές. Η τιμή μπορεί να είναι από 0 έως 63.

Σύνταξη: ADIW Rd:Rr, K

Παράδειγμα: ADIW R1:R2, 10 (το αποτέλεσμα τοποθετείται στο ζεύγος των καταχωρητών).

Εντολή SEC: Η εντολή SEC (Set carry) θέτει το κρατούμενο, δηλαδή το κάνει ίσο με 1 (C = 1).

Σύνταξη: SEC

Εντολή CLC: Η εντολή CLC (Clear carry) μηδενίζει το κρατούμενο, δηλαδή το κάνει ίσο με 0 (C = 0).

Σύνταξη: CLC

Άσκηση 2.1: Να γράψετε ένα πρόγραμμα που θα προσθέτει τους αριθμούς 09_{16} (9_{10} σε δεκαδική μορφή) και $5D_{16}$ (93_{10} σε δεκαδική μορφή), οι οποίοι

βρίσκονται στις θέσεις μνήμης 0300 και 0301 αντίστοιχα, και το αποτέλεσμα της πρόσθεσης να αποθηκεύεται στη θέση μνήμης 0302. Οι αριθμοί είναι των 8bit και το τελικό αποτέλεσμα να αποτυπωθεί στους λαμπτήρες της πλακέτας.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R25, $09
STS $0300, R25
LDI R26, $5D
STS $0301, R26

LDS R20, $0300
LDS R21, $0301
ADD R20, R21
STS $0302, R20

LDI R30, $FF
OUT DDRA, R30      ; ορίζουμε τους ακροδέκτες της θύρας A ως εξόδους
COM R20            ; τεχνική συμπληρώματος ως προς ένα στο τελικό αποτέλεσμα
OUT PORTA, R20     ; εξαγωγή αποτελέσματος στους ακροδέκτες της θύρας A
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι η μοναδική σημαία που ενημερώνεται μετά την πρόσθεση είναι η H (Half Carry), η οποία δηλώνει ότι έχει προκύψει κρατούμενο από τα 4 λιγότερο σημαντικά ψηφία. Επίσης, το αποτέλεσμα είναι σωστό, διότι μας δίνει 66_{16}, και σε δεκαδική μορφή αντιστοιχεί στον αριθμό 102. Ακόμη, ορίζουμε τους ακροδέκτες της θύρας A ως εξόδους, έχοντας δώσει την τιμή FF, εφαρμόζουμε την τεχνική συμπληρώματος ως προς ένα στον καταχωρητή που περιέχει το τελικό αποτέλεσμα και το εξάγουμε στους ακροδέκτες της θύρας A. Ο λόγος που χρησιμοποιούμε την εντολή COM είναι ότι στα led εφαρμόζεται η λογική active low. Δηλαδή, τα led τα οποία θα έχουν την τιμή μηδέν θα είναι αναμμένα και τα led τα οποία θα έχουν την τιμή ένα θα είναι σβηστά.

Άσκηση 2.2: Να γράψετε ένα πρόγραμμα που να προσθέτει τους 8μπιτους αριθμούς $9F_{16}$ (159_{10}) και 90_{16} (144_{10}), οι οποίοι βρίσκονται στις θέσεις μνήμης 0140 και 0141 αντίστοιχα. Το αποτέλεσμα της πρόσθεσης να αποθηκεύεται στη θέση

μνήμης 0142 και να αποτυπώνεται στους λαμπτήρες της πλακέτας. Αν μετά την πρόσθεση υπάρχει κρατούμενο, να αποθηκευτεί στη θέση μνήμης 0143.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $9F
STS $0140, R20
LDI R21, $90
STS $0141, R21
LDS R24, $0140
LDS R25, $0141
ADD R24, R25
STS $0142, R24

ADC R18, R19
STS $0143, R18

LDI R30, $FF
OUT DDRB, R30
EOR R24, R30
OUT PORTB, R24
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, θα παρατηρήσουμε ότι ενημερώνεται η σημαία C (Carry Flag), κάτι που σημαίνει ότι από την πρόσθεση των δύο αριθμών προκύπτει κρατούμενο. Επιπλέον, ενημερώνεται και η σημαία S (Sign Bit – σημαία προσήμου), η οποία προκύπτει από τη λογική πράξη OR ανάμεσα στη σημαία αρνητικού προσήμου και τη σημαία υπερχείλισης. Ακόμη, ενημερώνεται και η σημαία V (Two's Complement Overflow Flag), που είναι η σημαία υπερχείλισης στην αριθμητική συμπληρώματος ως προς 2, ενώ το αποτέλεσμα της πράξης είναι $\$2F_{16}$. Ο λόγος που χρησιμοποιούμε τη λογική εντολή EOR είναι ότι στα led εφαρμόζεται η λογική active low. Δηλαδή, τα led τα οποία θα έχουν την τιμή μηδέν θα είναι αναμμένα και τα led τα οποία θα έχουν την τιμή ένα θα είναι σβηστά.

Άσκηση 2.3: Να γράψετε ένα πρόγραμμα το οποίο θα πραγματοποιεί την αφαίρεση δύο αριθμών (προσημασμένων), χωρίς να έχουμε στη διάθεσή μας την

εντολή της αφαίρεσης. Θα προσθέσουμε τους αριθμούς A3 και A2, αφού πρώτα βρούμε τους αντίθετους αυτών. Οι αριθμοί να βρίσκονται αποθηκευμένοι στις θέσεις μνήμης 0140 και 0141 αντίστοιχα. Το αποτέλεσμα να αποθηκευτεί στη θέση μνήμης 0142 και να αποτυπωθεί στους λαμπτήρες της πλακέτας.

Υπόδειξη: Προκειμένου να μετατραπούν οι αριθμοί σε αντίθετους, να χρησιμοποιηθεί η τεχνική συμπληρώματος ως προς 2, δηλαδή η εντολή NEG. Η σύνταξη της συγκεκριμένης εντολής είναι: NEG Rd.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $A3
NEG R20          ; τεχνική συμπληρώματος ως προς 2
STS $0140, R20
LDI R21, $A2
NEG R21          ; τεχνική συμπληρώματος ως προς 2
STS $0141, R21

LDS R25, $0140
LDS R26, $0141
ADD R25, R26
STS $0142, R25

LDI R30, $FF
OUT DDRB, R30
COM R25
OUT PORTB, R25
```

Παρατηρήσεις: Ο αντίθετος αριθμός του A3 είναι ο 5D και ο αντίθετος αριθμός του A2 είναι ο 5E. Για να βρούμε τους αντίθετους αριθμούς, θα χρησιμοποιήσουμε την τεχνική συμπληρώματος ως προς 2, δηλαδή την εντολή NEG. Αμέσως μετά την πρόσθεση των αριθμών, ενημερώνεται η σημαία V (σημαία υπερχείλισης) και η σημαία N, που σημαίνει ότι ο αριθμός που έχει προκύψει (BB_{16}) είναι αρνητικός. Το αποτέλεσμα της πρόσθεσης είναι BB_{16}.

Άσκηση 2.4: Να γράψετε ένα πρόγραμμα το οποίο να προσθέτει δύο αριθμούς των 16bit. Ο ένας αριθμός να είναι ο 3945_{16} (14661_{10}) και να είναι αποθηκευμένος στις θέσεις μνήμης 0180 (byte χαμηλής τάξης) και 0181 (byte υψηλής τάξης) και ο δεύτερος αριθμός να είναι ο $E5C2_{16}$ (58818_{10}), που να είναι αποθηκευμένος στις θέσεις μνήμης 0182 (byte χαμηλής τάξης) και 0183 (byte υψηλής τάξης). Τα αποτελέσματα της πρόσθεσης να τοποθετούνται στις θέσεις μνήμης 0184 (byte χαμηλής τάξης) και 0185 (byte υψηλής τάξης).

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $45        ; χαμηλής τάξης byte
STS $0180, R20
LDI R21, $39        ; υψηλής τάξης byte
STS $0181, R21
LDI R22, $C2        ; χαμηλής τάξης byte
STS $0182, R22
LDI R23, $E5        ; υψηλής τάξης byte
STS $0183, R23

LDS R16, $0180
LDS R18, $0182
ADD R16, R18
STS $0184, R16

LDS R17, $0181
LDS R19, $0183
ADC R17, R19        ; πρόσθεση με κρατούμενο
STS $0185, R17
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι μετά από την πρώτη πρόσθεση, όπου το αποτέλεσμα είναι 07_{16}, ενημερώνεται η σημαία κρατούμενου (C = 1). Με την εντολή ADC λαμβάνουμε υπόψη τυχόν κρατούμενο της προηγούμενης πρόσθεσης. Μετά από την πρόσθεση με την εντολή ADC, η σημαία κρατουμένου παραμένει ίση με ένα, ενώ το αποτέλεσμα είναι $1F_{16}$.

Άσκηση 2.5: Να γράψετε ένα πρόγραμμα το οποίο να προσθέτει δύο αριθμούς των 32bit. Ο ένας αριθμός να είναι ο 3512 1002 και να είναι αποθηκευμένος από τη θέση μνήμης 0300 έως 0303 και ο δεύτερος να είναι ο αριθμός AB27 C53E και να είναι αποθηκευμένος από τη θέση μνήμης 0304 έως 0307. Τα αποτελέσματα των προσθέσεων να μεταφέρονται στις θέσεις μνήμης από 0308 έως 030B.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $02
STS $0300, R16
LDI R17, $3E
STS $0304, R17

LDI R18, $10
STS $0301, R18
LDI R19, $C5
STS $0305, R19

LDI R20, $12
STS $0302, R20
LDI R21, $27
STS $0306, R21

LDI R22, $35
STS $0303, R22
LDI R23, $AB
STS $0307, R23

LDS R1, $0300
LDS R2, $0304
ADD R1, R2          ; πρώτη πρόσθεση
STS $0308, R1

LDS R3, $0301
```

```
LDS R4, $0305
ADC R3, R4          ; δεύτερη πρόσθεση με κρατούμενο
STS $0309, R3

LDS R5, $0302
LDS R6, $0306
ADC R5, R6          ; τρίτη πρόσθεση με κρατούμενο
STS $030A, R5

LDS R7, $0303
LDS R8, $0307
ADC R7, R8          ; τέταρτη πρόσθεση με κρατούμενο
STS $030B, R7
```

Παρατηρήσεις: Μετά την πρώτη πρόσθεση, ενημερώνεται η σημαία H, οπότε προέκυψε κρατούμενο από τα λιγότερο σημαντικά bytes. Το αποτέλεσμα της πρώτης πρόσθεσης είναι 40_{16}. Στη συνέχεια, στις επόμενες προσθέσεις χρησιμοποιούμε την εντολή ADC, προκειμένου να λάβουμε υπόψη τυχόν κρατούμενο των προηγούμενων προσθέσεων. Ύστερα από τη δεύτερη πρόσθεση, ενημερώνονται οι σημαίες S και N. Οπότε, ο αριθμός που προέκυψε ($D5_{16}$) είναι αρνητικός. Μετά την τρίτη πρόσθεση, δεν ενημερώνεται καμία σημαία και το αποτέλεσμα είναι 39_{16}. Τέλος, όταν πραγματοποιείται και η τέταρτη πρόσθεση, ενημερώνονται οι σημαίες H, S και N και το αποτέλεσμα είναι $E0_{16}$.

8.4 Εργαστήριο 3

Εισαγωγικά: Στις ασκήσεις του τρίτου εργαστηριακού μαθήματος θα ασχοληθούμε με την αφαίρεση και τον πολλαπλασιασμό των αριθμών. Οι εντολές οι οποίες χρησιμοποιούνται στην αφαίρεση είναι οι: SUB, SUBI, SBC και SBCI. Στον πολλαπλασιασμό, οι εντολές που χρησιμοποιούνται είναι η MUL και η MULS. Επίσης, θα δούμε και την εντολή NEG.

Εντολές που χρησιμοποιούνται:

Εντολή SUB (SUB καταχωρητής 1, καταχωρητής 2): Η εντολή αυτή (Subtract two registers) πραγματοποιεί αφαίρεση μεταξύ του περιεχομένου δύο

καταχωρητών. Συγκεκριμένα, αφαιρεί το περιεχόμενο του δεύτερου καταχωρητή από το περιεχόμενο του πρώτου, χωρίς να χρησιμοποιεί κάποιο κρατούμενο.

Σύνταξη: SUB Rd, Rr

Παράδειγμα: SUB R1, R2 (το αποτέλεσμα της αφαίρεσης θα τοποθετηθεί στον καταχωρητή R1)

Εντολή SUBI (SUBI καταχωρητής, τιμή): Η εντολή SUBI (Subtract constant from register) αφαιρεί μία τιμή από το περιεχόμενο ενός καταχωρητή, χωρίς να χρησιμοποιεί κάποιο κρατούμενο.

Σύνταξη: SUBI Rd, K

Παράδειγμα: SUBI R25, $30 (το αποτέλεσμα της αφαίρεσης θα τοποθετηθεί στον καταχωρητή R25)

Σημείωση: Η εντολή SUBI μπορεί να χρησιμοποιήσει έναν οποιοδήποτε καταχωρητή από τους R16 έως R31 και όχι από τους καταχωρητές R0 έως R15.

Εντολή SBC (SBC καταχωρητής 1, καταχωρητής 2): Η εντολή SBC (Subtract with carry two registers) χρησιμοποιείται για την αφαίρεση του περιεχομένου δύο καταχωρητών. Συγκεκριμένα, αφαιρεί το περιεχόμενο του δεύτερου καταχωρητή από το περιεχόμενο του πρώτου. Η αφαίρεση σ' αυτή την περίπτωση γίνεται με τη χρήση κρατουμένου.

Σύνταξη: SBC Rd, Rr

Παράδειγμα: SBC R11, R22 (το αποτέλεσμα της αφαίρεσης θα τοποθετηθεί στον καταχωρητή R11)

Εντολή SBCI (SBCI καταχωρητής, τιμή): Η εντολή SBCI (Subtract with carry constant from register) αφαιρεί μία τιμή από το περιεχόμενο ενός καταχωρητή με τη χρήση κρατουμένου.

Σύνταξη: SBCI Rd, K

Παράδειγμα: SBCI R31, $15 (το αποτέλεσμα της αφαίρεσης θα τοποθετηθεί στον καταχωρητή R31)

Σημείωση: Η εντολή SBCI μπορεί να χρησιμοποιήσει έναν οποιοδήποτε καταχωρητή από τους R16 έως R31 και όχι από τους καταχωρητές R0 έως R15.

Εντολή MUL (MUL καταχωρητής 1, καταχωρητής 2): Η εντολή MUL (Multiply unsigned) χρησιμοποιείται για τον πολλαπλασιασμό του περιεχομένου δύο καταχωρητών και ο πολλαπλασιασμός είναι για μη προσημασμένους αριθμούς.

Σύνταξη: MUL Rd, Rr

Παράδειγμα: MUL R10, R19 [το αποτέλεσμα του πολλαπλασιασμού θα τοποθετηθεί στον καταχωρητή R0 (byte χαμηλής τάξης) και στον καταχωρητή R1 (byte υψηλής τάξης)].

Εντολή MULS (MULS καταχωρητής 1, καταχωρητής 2): Η εντολή MULS (Multiply signed) χρησιμοποιείται για τον πολλαπλασιασμό του περιεχομένου δύο καταχωρητών και στη συγκεκριμένη περίπτωση ο πολλαπλασιασμός είναι για προσημασμένους αριθμούς.

Σύνταξη: MULS Rd, Rr

Παράδειγμα: MULS R20, R30 [το αποτέλεσμα του πολλαπλασιασμού θα τοποθετηθεί στον καταχωρητή R0 (byte χαμηλής τάξης) και στον καταχωρητή R1 (byte υψηλής τάξης)].

Εντολή NEG: Η εντολή αυτή αντικαθιστά το περιεχόμενο ενός καταχωρητή με το συμπληρωματικό του ως προς 2.

Σύνταξη: NEG Rd

Παράδειγμα: NEG R20

Άσκηση 3.1: Να γράψετε ένα πρόγραμμα το οποίο να αφαιρεί τον αριθμό 3D από τον αριθμό 65. Οι αριθμοί να βρίσκονται στις θέσεις μνήμης 0301 και 0300 αντίστοιχα και το αποτέλεσμα να αποθηκεύεται στη θέση μνήμης 0302. Επίσης, το τελικό αποτέλεσμα να αποτυπώνεται στους λαμπτήρες της πλακέτας.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $65
STS $0300, R16
LDI R17, $3D
STS $0301, R17

LDS R1, $0300
LDS R2, $0301
SUB R1, R2
STS $0302, R1

LDI R20, $FF
```

```
OUT DDRD, R20
COM R1
OUT PORTD, R1
```

Παρατηρήσεις: Μετά την αφαίρεση των δύο αριθμών, η σημαία που ενημερώνεται είναι η σημαία H και δηλώνει ότι έχει προκύψει δεκαδικό κρατούμενο (ή αλλιώς κρατούμενο από τα 4 λιγότερο σημαντικά ψηφία). Το τελικό αποτέλεσμα είναι το 28_{16}.

Άσκηση 3.2: Να γράψετε ένα πρόγραμμα το οποίο να αφαιρεί τους προσημασμένους αριθμούς 13 και BE με χρήση της εντολής SUB, αφού πρώτα ο αριθμός BE μετατραπεί στον αντίθετό του. Οι αριθμοί να βρίσκονται στις θέσεις μνήμης 0202 και 0201 αντίστοιχα και το αποτέλεσμα να αποθηκεύεται στη θέση μνήμης 0203. Επίσης, το τελικό αποτέλεσμα να αποτυπώνεται στους λαμπτήρες της πλακέτας.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $BE
NEG R16
STS $0201, R16
LDI R17, $13
STS $0202, R17

LDS R1, $0201
LDS R2, $0202
SUB R1, R2
STS $0203, R1

LDI R25, $FF
OUT DDRB, R25
COM R1
OUT PORTB, R1
```

Παρατηρήσεις: Ο αντίθετος αριθμός του BE_{16} είναι ο αριθμός 42_{16}. Μετά τη μετατροπή του αριθμού, ενημερώνονται οι σημαίες H και C. Μετά την αφαίρεση, η σημαία H είναι ίση με 1, ενώ το τελικό αποτέλεσμα είναι το $2F_{16}$.

Άσκηση 3.3: Να γράψετε ένα πρόγραμμα το οποίο να αφαιρεί την τιμή 1207_{16} (4615_{10}) από την τιμή 3548_{16} (13640_{10}). Η τιμή 3548_{16} να βρίσκεται στις θέσεις μνήμης 0300 (byte χαμηλής τάξης) και 0301 (byte υψηλής τάξης), ενώ η τιμή 1207_{16} να βρίσκεται στις θέσεις μνήμης 0302 (byte χαμηλής τάξης) και 0303 (byte υψηλής τάξης). Το αποτέλεσμα της αφαίρεσης της χαμηλής τάξης bytes να τοποθετείται στη θέση μνήμης 0304 και το αποτέλεσμα της αφαίρεσης της υψηλής τάξης bytes να τοποθετείται στη μνήμη 0305. Οι αριθμοί είναι των 16bit.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $48        ; χαμηλής τάξης byte
STS $0300, R16
LDI R18, $35        ; υψηλής τάξης byte
STS $0301, R18
LDI R17, $07        ; χαμηλής τάξης byte
STS $0302, R17
LDI R19, $12        ; υψηλής τάξης byte
STS $0303, R19

LDS R1, $0300
LDS R2, $0302
SUB R1, R2
STS $0304, R1

LDS R3, $0301
LDS R4, $0303
SBC R3, R4          ; αφαίρεση με κρατούμενο
STS $0305, R3
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι δεν ενημερώνεται καμία σημαία. Το αποτέλεσμα της πρώτης αφαίρεσης είναι 41_{16} (65_{10})

και της δεύτερης είναι 23_{16} (35_{10}). Στη δεύτερη αφαίρεση χρησιμοποιούμε την εντολή SBC, προκειμένου να λάβουμε υπόψη τυχόν κρατούμενο από την προηγούμενη αφαίρεση.

Άσκηση 3.4: Να γράψετε ένα πρόγραμμα το οποίο να αφαιρεί την τιμή 0008_{16} (8_{10}) από την τιμή 6600_{16} (26112_{10}). Η τιμή 6600_{16} να βρίσκεται στις θέσεις μνήμης $0300 (byte χαμηλής τάξης) και 0301 (byte υψηλής τάξης), ενώ η τιμή 0008_{16} να βρίσκεται στις θέσεις μνήμης 0302 (byte χαμηλής τάξης) και 0303 (byte υψηλής τάξης). Το αποτέλεσμα της αφαίρεσης της χαμηλής τάξης bytes να τοποθετείται στη θέση μνήμης 0304 και το αποτέλεσμα της αφαίρεσης της υψηλής τάξης bytes να τοποθετείται στη θέση μνήμης 0305. Οι αριθμοί είναι των 16bit.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $00        ; χαμηλής τάξης byte
STS $0300, R16
LDI R18, $66        ; υψηλής τάξης byte
STS $0301, R18
LDI R17, $08        ; χαμηλής τάξης byte
STS $0302, R17
LDI R19, $00        ; υψηλής τάξης byte
STS $0303, R19

LDS R1, $0300
LDS R2, $0302
SUB R1, R2
STS $0304, R1

LDS R3, $0301
LDS R4, $0303
SBC R3, R4
STS $0305, R3
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι μετά την πρώτη αφαίρεση ενημερώνονται οι σημαίες H, S, N και C. Αυτό σημαίνει ότι μετά

την αφαίρεση των λιγότερο σημαντικών bytes προκύπτει κρατούμενο (σημαία H), υπάρχει δείκτης προσήμου (σημαία S), ο αριθμός που προκύπτει ($F8_{16}$) είναι αρνητικός (σημαία N) και υπάρχει κρατούμενο (σημαία C). Αντίθετα, μετά τη δεύτερη αφαίρεση δεν ενημερώνεται καμία σημαία. Το αποτέλεσμα της πρώτης αφαίρεσης είναι $F8_{16}$ (-248_{10}), ενώ της δεύτερης είναι 65_{16} (101_{10}).

Άσκηση 3.5: Να γράψετε ένα πρόγραμμα στο οποίο να πολλαπλασιάζονται οι αριθμοί 03_{16} (3_{10}) και 13_{16} (19_{10}). Ο αριθμός 03_{16} να είναι τοποθετημένος στη θέση μνήμης 0200 και ο αριθμός 13_{16} να βρίσκεται στη θέση μνήμης 0201. Το αποτέλεσμα να αποθηκεύεται στη θέση μνήμης 0202 και να αποτυπώνεται και στους λαμπτήρες της οθόνης.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $03
STS $0200, R16
LDI R17, $13
STS $0201, R17

LDS R5, $0200
LDS R6, $0201
MUL R5, R6
STS $0202, R0

LDI R20, $FF
OUT DDRA, R20
COM R0
OUT PORTA, R0
```

Παρατηρήσεις: Παρατηρούμε ότι, αφού εκτελέσουμε το πρόγραμμα, δεν ενημερώνεται καμία σημαία του συστήματος. Εξορισμού, το αποτέλεσμα του πολλαπλασιασμού αποθηκεύεται στον καταχωρητή R0. Το συγκεκριμένο αποτέλεσμα είναι το 39_{16} (57_{10}) και είναι σωστό.

Άσκηση 3.6: Να γράψετε πρόγραμμα το οποίο να πολλαπλασιάζει τους αριθμούς 00_{16} (0_{10}) και 10_{16} (16_{10}). Ο αριθμός 00_{16} να βρίσκεται στη θέση μνήμης

0300 και ο αριθμός 10_{16} στη θέση μνήμης 0301. Το αποτέλεσμα να αποθηκευτεί στη θέση μνήμης 0302 και να αποτυπωθεί στα led της πλακέτας.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $00
STS $0300, R16
LDI R17, $10
STS $0301, R17

LDS R5, $0300
LDS R6, $0301
MUL R5, R6
STS $0302, R0

LDI R20, $FF
OUT DDRA, R20
COM R0
OUT PORTA, R0
```

Παρατηρήσεις: Παρατηρούμε ότι η μοναδική σημαία που ενημερώνεται είναι η σημαία Z (Zero), η οποία δηλώνει ότι το αποτέλεσμα (που αποθηκεύεται στον R0) είναι μηδέν.

Άσκηση 3.7: Να γράψετε ένα πρόγραμμα το οποίο να πολλαπλασιάζει τους αριθμούς FF_{16} (255_{10}) και 34_{16} (52_{10}). Ο αριθμός FF_{16} να βρίσκεται στη θέση μνήμης 0300 και ο αριθμός 34_{16} στη θέση μνήμης 0301. Τα αποτελέσματα να αποθηκευτούν στις θέσεις μνήμης 0302 και 0303.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $FF
STS $0300, R16
LDI R17, $34
STS $0301, R17
```

```
LDS R5, $0300
LDS R6, $0301
MUL R5, R6
STS $0302, R0
STS $0303, R1
```

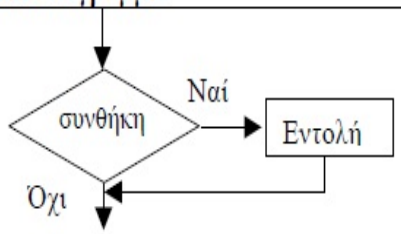
Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι ο καταχωρητής R0 (χαμηλής τάξης byte) έχει το αποτέλεσμα CC (204_{10}) και ο καταχωρητής R1 (υψηλής τάξης byte) έχει το αποτέλεσμα 33 (52_{10}). Εξορισμού, το αποτέλεσμα του πολλαπλασιασμού αποθηκεύεται στους δύο αυτούς καταχωρητές.

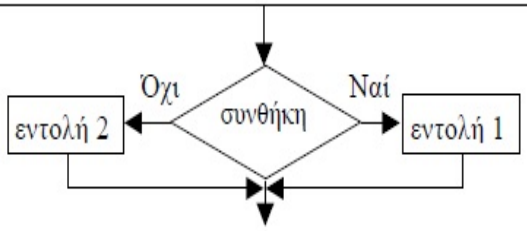
8.5 Εργαστήριο 4

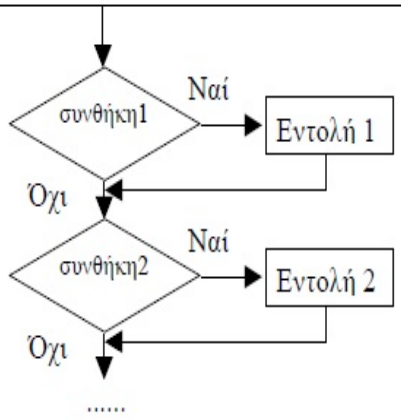
Εισαγωγικά: Στο συγκεκριμένο εργαστηριακό μάθημα θα δούμε τη σύγκριση δύο αριθμών, τη διακλάδωση προγράμματος με συνθήκη, τη διακλάδωση προγράμματος χωρίς συνθήκη, καθώς και τις αντίστοιχες εντολές τους.

Θεωρία: Οι γλώσσες προγραμματισμού διαθέτουν μία πολύ σημαντική δυνατότητα. Πρόκειται για τη δυνατότητα δημιουργίας δομών «ελέγχου ροής προγράμματος». Οι συγκεκριμένες δομές, στην περίπτωση που ισχύει κάποια συνθήκη, πραγματοποιούν αλλαγή στη ροή των προγραμμάτων, με αποτέλεσμα να επιτρέπουν να υλοποιηθούν εμβόλιμα τμήματα κώδικα. Αυτό έχει ως συνέπεια τη δημιουργία προγραμμάτων που εφαρμόζουν διαφορετικές εντολές ανάλογα με τις συνθήκες ή η εφαρμογή τους να εξαρτάται από τις παραμέτρους που εισάγει ο προγραμματιστής. Οι δομές, οι οποίες είναι κοινές, στις γλώσσες προγραμματισμού είναι οι εξής:

Α/Α	Δομή	Λογικό Διάγραμμα	Παράδειγμα σε C

1	If - then		…
			if (k>10)
			{
			k=k-10;
			reset=true;
			}
			…

2	If – then - else		…
			if (A[i]==Search)
			{ Found=true;
			Position=i; }
			else i=i+1;
			…

3	Case / Switch		…
			switch (key)
			{
			case 'U' : y--;break;
			case 'D' : y++; break;
			case 'L' : x--; break;
			case 'R' : x++; break;
			}
			…

Εντολές που χρησιμοποιούνται:

1) **Εντολές σύγκρισης:**

- Εντολή CP (CP καταχωρητής 1, καταχωρητής 2): Η εντολή CP (Compare) συγκρίνει το περιεχόμενο δύο καταχωρητών.

  Σύνταξη: CP Rd, Rr

  Παράδειγμα: CP R20, R22

- Εντολή CPC (CPC καταχωρητής 1, καταχωρητής 2): Η εντολή CPC (Compare with carry) συγκρίνει το περιεχόμενο δύο καταχωρητών. Επίσης, λαμβάνει υπόψη και το προηγούμενο κρατούμενο.

  Σύνταξη: CPC Rd, Rr

  Παράδειγμα: CPC R25, R18

- Εντολή CPI (CPI καταχωρητής, τιμή): Η εντολή CPI (Compare register with immediate) πραγματοποιεί σύγκριση μεταξύ του περιεχομένου ενός καταχωρητή και μίας τιμής.

Σύνταξη: CPI Rd, K

Παράδειγμα: CPI R19, $08

- Εντολή CPSE (CPSE καταχωρητής 1, καταχωρητής 2): Η εντολή CPSE (Compare skip, if equal) συγκρίνει το περιεχόμενο δύο καταχωρητών και, αν το περιεχόμενό τους είναι ίσο (π.χ. R1 = R5), τότε παραβλέπει την επόμενη εντολή.

  Σύνταξη**:** CPSE Rd, Rr

  Παράδειγμα: CPSE R1, R5

2) **Εντολές, προκειμένου να πραγματοποιηθεί διακλάδωση προγράμματος χωρίς συνθήκη:**
   - ✓ **Εντολή JMP:** Η εντολή JMP (Jump) πραγματοποιεί μεταφορά της εκτέλεσης του προγράμματος σε μία εντολή η οποία είναι σε άλλη θέση μνήμης του προγράμματος.

     Σύνταξη: JMP ARXH
   - ✓ **Εντολή IJMP (Indirect Jump):** Η εντολή IJMP (Indirect Jump) πραγματοποιεί μεταφορά της εκτέλεσης του προγράμματος σε μία διεύθυνση, η οποία διεύθυνση ορίζεται από το δείκτη Z. Επίσης, η μεταφορά του προγράμματος είναι έμμεση. Ακόμη, ο συγκεκριμένος δείκτης Z αντιστοιχεί στους καταχωρητές R31 και R30 (Z = R31, R30).

     Σύνταξη: IJMP
   - ✓ **Εντολή RJMP (Relative Jump):** Η εντολή RJMP πραγματοποιεί σχετική μεταπήδηση σε μία διεύθυνση προγράμματος ανάμεσα σε PC – 2K και PC + 2K. Επίσης, στον assembler αντί για αριθμούς χρησιμοποιούνται ετικέτες, καθώς το K είναι ετικέτα.

     Σύνταξη: RJMP K

3) **Εντολές, προκειμένου να πραγματοποιηθεί διακλάδωση προγράμματος με συνθήκη:** Σημείωση: Οι εντολές που ακολουθούν αφορούν μη προσημασμένους αριθμούς.
   - **Εντολή BRLO:** Η εντολή BRLO (Branch if lower) ελέγχει αν η σημαία κρατουμένου είναι ίση με 1 [if (C = 1)]. Αν C = 1, τότε πραγματοποιείται διακλάδωση στο μετρητή προγράμματος. Επίσης, η εντολή BRLO εκτελείται μετά από τις εντολές αφαίρεσης (SUB, SUBI) και σύγκρισης (CP, CPI) και η διακλάδωση θα

πραγματοποιηθεί, μόνο στην περίπτωση που ο πρώτος αριθμός είναι μικρότερος από το δεύτερο αριθμό.

Σύνταξη: BRLO K

- **Εντολή BRSH:** Η εντολή BRSH (Branch if same or higher) πραγματοποιεί έλεγχο στη σημαία κρατουμένου σχετικά με το αν είναι ίση με 0 [if (C = 0)]. Σε περίπτωση που C = 0, δημιουργεί διακλάδωση στο μετρητή προγράμματος. Επίσης, η συγκεκριμένη εντολή εκτελείται μετά από τις εντολές αφαίρεσης (SUB, SUBI) και σύγκρισης (CP, CPI) και η διακλάδωση στο πρόγραμμα θα γίνει, μόνο στην περίπτωση κατά την οποία ο πρώτος αριθμός είναι ίσος ή μεγαλύτερος από το δεύτερο αριθμό.

  Σύνταξη: BRSH K

4) **Εντολές, προκειμένου να πραγματοποιηθεί διακλάδωση με συνθήκη:**

Σημείωση: Οι εντολές που ακολουθούν αφορούν και προσημασμένους αλλά και μη προσημασμένους αριθμούς.

- **Εντολή BREQ**: Η εντολή BREQ (Branch if equal) ελέγχει αν η σημαία μηδενισμού είναι ίση με 1 [if (Z = 1)]. Αν Z = 1, τότε πραγματοποιείται διακλάδωση στο μετρητή προγράμματος. Ακόμη, η εντολή BREQ εκτελείται μετά από τις εντολές αφαίρεσης (SUB, SUBI) και σύγκρισης (CP, CPI) και η διακλάδωση στο πρόγραμμα θα γίνει, μόνο σε περίπτωση που οι δύο αριθμοί είναι ίσοι.

  Σύνταξη: BREQ K

- **Εντολή BRNE:** Η εντολή BRNE (Branch if not equal) ελέγχει αν η σημαία μηδενισμού είναι ίση με 0 [if (Z = 0)] και, σε περίπτωση που Z = 0, πραγματοποιείται διακλάδωση στο μετρητή προγράμματος. Επιπλέον, όπως οι προηγούμενες εντολές, έτσι κι αυτή εκτελείται μετά από τις εντολές αφαίρεσης (SUB, SUBI) και σύγκρισης (CP, CPI) και η διακλάδωση στο πρόγραμμα θα γίνει, μόνο σε περίπτωση που ο πρώτος αριθμός ΔΕΝ είναι ίσος με το δεύτερο αριθμό.

  Σύνταξη: BRNE K

5) **Εντολές, προκειμένου να πραγματοποιηθεί διακλάδωση με συνθήκη:**

Σημείωση: Οι εντολές που ακολουθούν αφορούν προσημασμένους αριθμούς.

- **Εντολή BRGE:** Η εντολή BRGE (Branch if greater or equal) πραγματοποιεί έλεγχο στη σημαία προσήμου και εξετάζει αν είναι ίση με μηδέν [if (S = 0)]. Σε περίπτωση που S = 0, τότε δημιουργείται διακλάδωση στο μετρητή προγράμματος. Επιπλέον, η εντολή BRGE εκτελείται μετά από τις εντολές αφαίρεσης (SUB, SUBI) και σύγκρισης (CP, CPI) και η διακλάδωση προγράμματος θα γίνει, εφόσον ο πρώτος αριθμός είναι μεγαλύτερος ή ίσος από το δεύτερο αριθμό.

  Σύνταξη: BRGE K
- **Εντολή BRLT:** Η εντολή BRLT (Branch if less than) ελέγχει αν η σημαία προσήμου είναι ίση με 1 [if (S = 1)]. Αν S = 1, τότε πραγματοποιεί σχετική διακλάδωση στο μετρητή προγράμματος. Επίσης, η εντολή εκτελείται μετά από τις εντολές αφαίρεσης (SUB, SUBI) και σύγκρισης (CP, CPI) και η διακλάδωση προγράμματος γίνεται, μόνο σε περίπτωση που ο πρώτος αριθμός είναι μικρότερος από το δεύτερο αριθμό.

  Σύνταξη: BRLT K

Άσκηση 4.1: Να γράψετε ένα πρόγραμμα που θα συγκρίνει δύο αριθμούς οι οποίοι βρίσκονται στις θέσεις μνήμης $0300 και $0301 αντίστοιχα. Αν ο πρώτος αριθμός είναι μεγαλύτερος από το δεύτερο ή οι δύο αριθμοί είναι ίσοι, να παραμένουν όπως είναι, και, αν ο πρώτος αριθμός είναι μικρότερος από το δεύτερο, τότε να εναλλάσσονται στις θέσεις μνήμης. Επίσης, στη θέση μνήμης 0200 να περιέχονται οι αριθμοί: 01, αν ο πρώτος αριθμός είναι μεγαλύτερος από το δεύτερο, 00, αν οι δύο αριθμοί είναι ίσοι, και 02, αν ο πρώτος αριθμός είναι μικρότερος από το δεύτερο. Το τελικό αποτέλεσμα της μνήμης 0200 να αποτυπώνεται στους λαμπτήρες της οθόνης. Επίσης, να γράψετε κώδικα για τους αριθμούς 25 – 15, EE – EE και 11 – 12.

Λύση (για το ζεύγος αριθμών 25 – 15):

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $25
STS $0300, R20
```

```
LDI R21, $15
STS $0301, R21

LDS R25, $0300
LDS R26, $0301
CP   R25, R26

BREQ EQUAL
BRSH GREATER
BRLO LESS

EQUAL:
LDI R16, $00
STS $0200, R16
RJMP TELOS

GREATER:
LDI R16, $01
STS $0200, R16
RJMP TELOS

LESS:
STS $0300, R25
STS $0301, R26
LDI R16, $02
STS $0200, R16
RJMP TELOS

TELOS:
LDI R30, $FF
OUT DDRA, R30
COM R16
OUT PORTA, R16
RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι η εντολή BRSH ελέγχει τη σημαία κρατουμένου. Αν είναι ίση με μηδέν, η σημαία δεν ενημερώνεται, οπότε C = 0 και υλοποιείται η διακλάδωση προγράμματος. Με την εντολή BRSH GREATER το πρόγραμμα μεταφέρεται στο σημείο GREATER:, το οποίο αφορά την περίπτωση κατά την οποία ο πρώτος αριθμός είναι μεγαλύτερος από το δεύτερο. Επίσης, μετά τη σύγκριση των δύο αριθμών δεν ενημερώνεται καμία σημαία.

Λύση (για το ζεύγος αριθμών EE – EE):

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $EE
STS $0300, R20
LDI R21, $EE
STS $0301, R21

LDS R25, $0300
LDS R26, $0301
CP   R25, R26

BREQ EQUAL
BRSH GREATER
BRLO LESS

EQUAL:
LDI R16, $00
STS $0200, R16
RJMP TELOS

GREATER:
LDI R16, $01
STS $0200, R16
RJMP TELOS

LESS:
```

```
STS $0300, R25
STS $0301, R26
LDI R16, $02
STS $0200, R16
RJMP TELOS

TELOS:
LDI R30, $FF
OUT DDRA, R30
COM R16
OUT PORTA, R16
RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι η εντολή BREQ ελέγχει τη σημαία μηδενισμού, η οποία είναι όντως Z = 1, οπότε και πραγματοποιείται η αλλαγή στο πρόγραμμα. Με την εντολή BREQ EQUAL το πρόγραμμα μεταφέρεται στο σημείο EQUAL:, το οποίο αφορά την περίπτωση κατά την οποία οι δύο αριθμοί είναι ίσοι. Αντίθετα με την προηγούμενη περίπτωση, μετά τη σύγκριση ενημερώνεται η σημαία Z.

Λύση (για το ζεύγος αριθμών 11 – 12):

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $11
STS $0300, R20
LDI R21, $12
STS $0301, R21

LDS R25, $0300
LDS R26, $0301
CP   R25, R26

BREQ EQUAL
BRSH GREATER
BRLO LESS
```

```
EQUAL:
LDI R16, $00
STS $0200, R16
RJMP TELOS

GREATER:
LDI R16, $01
STS $0200, R16
RJMP TELOS

LESS:
STS $0300, R26
STS $0301, R25
LDI R16, $02
STS $0200, R16
RJMP TELOS

TELOS:
LDI R30, $FF
OUT DDRA, R30
COM R16
OUT PORTA, R16
RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι η εντολή BRLO ελέγχει τη σημαία κρατούμενου, η οποία είναι όντως C = 1, οπότε και εκτελείται η διακλάδωση στο πρόγραμμα. Με την εντολή BRLO LESS το πρόγραμμα μεταφέρεται στο σημείο LESS:, το οποίο αφορά την περίπτωση κατά την οποία ο πρώτος αριθμός είναι μικρότερος από το δεύτερο, ενώ μετά τη σύγκριση των αριθμών ενημερώνονται οι σημαίας H, S, N και C.

Άσκηση 4.2: Να γράψετε ένα πρόγραμμα το οποίο από τη θέση μνήμης 0300 έως 0309 θα τοποθετεί με αυτόματο τρόπο τους αριθμούς από 20 έως 29.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"
```

```
LDI R20, $20
LDI R21, $00        ; μετρητής επαναλήψεων
LDI R29, $03
LDI R28, $00        ; ορίζεται η θέση μνήμης 0300

LOOP:
ST Y, R20           ; αποθήκευση του περιεχομένου του R20 στη θέση μνήμης
                    ; ξεκινώντας από την 0300
INC R20             ; αύξηση αριθμού
INC R28             ; αύξηση θέσης μνήμης
INC R21             ; αύξηση μετρητή επαναλήψεων
CPI R21, $0A
BRNE LOOP

RET
```

Παρατηρήσεις: Μετά τη σύγκριση του μετρητή επαναλήψεων, ενημερώνονται οι σημαίες H, S, N και C. Το πρόγραμμα τρέχει, όσο ο μετρητής δεν είναι ίσος με τον αριθμό $0A_{16.}$ Όταν γίνει ίσος, ενημερώνεται η σημαία Z και το πρόγραμμα αρχίζει από την αρχή. Ο καταχωρητής Y χρησιμοποιείται ως δείκτης και ορίζεται από τους καταχωρητές R28:R29. Ο δείκτης ST Y δείχνει στη θέση μνήμης που έχουμε ορίσει, δηλαδή στην 0300, επομένως η εντολή ST Y, R20 μεταφέρει το περιεχόμενο του καταχωρητή R20 στην αντίστοιχη θέση μνήμης, η οποία θέση μνήμης με την εντολή INC R28 αυξάνεται κατά ένα.

8.6 Εργαστήριο 5

Εισαγωγικά: Στο συγκεκριμένο εργαστήριο θα δούμε τους βρόχους επανάληψης και δύο νέες εντολές, την εντολή INC και την εντολή DEC.

Θεωρία: Οι βρόχοι επανάληψης, ύστερα από τις δομές ελέγχου ροής του προγράμματος, που αναλύσαμε στο προηγούμενο εργαστήριο, αποτελούν κι αυτοί πολύ σημαντικές δομές στον προγραμματισμό. Οι βρόχοι παρέχουν τη δυνατότητα να δημιουργηθούν προγράμματα τα οποία εφαρμόζουν λειτουργίες επανάληψης,

προκειμένου να πραγματοποιηθούν σύνθετοι αλγόριθμοι και προγράμματα, που ενεργούν με τον ίδιο ακριβώς τρόπο σε έναν υψηλό αριθμό δεδομένων.

Στο πλαίσιο αυτό, Ένας βρόχος επανάληψης μεταφέρει την εκτέλεση του προγράμματος σε μία προηγούμενη διεύθυνσή του και η μεταφορά αυτή γίνεται με συνθήκη. Η μεταφορά πραγματοποιείται, αφού οριστεί πριν από το βρόχο επανάληψης μία αρχική τιμή σε μία μεταβλητή, οπότε, στη συνέχεια, μέσα στη δομή του βρόχου επανάληψης η τιμή της μεταβλητής αλλάζει. Η μεταβολή της μεταβλητής γίνεται ανάλογα με το τι έχουμε ορίσει να συμβεί σ' αυτήν. Επομένως, η μεταβλητή μπορεί να αυξάνεται κατά ένα ή να μειώνεται κατά ένα. Επίσης, όταν τελειώνει ο βρόχος επανάληψης, πραγματοποιείται έλεγχος σχετικά με το αν η συγκεκριμένη μεταβλητή ισούται με το όριο των επαναλήψεων που έχουμε ορίσει.

Εντολές που χρησιμοποιούνται:

Χρησιμοποιούνται όλες οι εντολές που συναντήσαμε στο προηγούμενο εργαστήριο. Οι δύο νέες εντολές τις οποίες θα δούμε είναι η εντολή INC και η εντολή DEC.

Εντολές αύξησης και μείωσης των περιεχομένων των καταχωρητών

- **Εντολή INC (INC καταχωρητής):** Η εντολή INC (Increment) αυξάνει το περιεχόμενο ενός καταχωρητή κατά 1.

  Σύνταξη: INC Rd

  Παράδειγμα: INC R1

  Εντολή DEC (DEC καταχωρητής): Η εντολή DEC (Decrement) μειώνει το περιεχόμενο ενός καταχωρητή κατά 1.

  Σύνταξη: DEC Rd

  Παράδειγμα: DEC R5

Άσκηση 5.1: Δίνεται ένας πίνακας, ο οποίος από τη θέση μνήμης 0310 έως 0319 περιέχει 10 αριθμούς των 8bit. Να γράψετε ένα πρόγραμμα που να βρίσκει το μεγαλύτερο αριθμό και να τον τοποθετεί στη θέση μνήμης 0400. Επίσης, ο αριθμός να αποτυπωθεί και στους λαμπτήρες της οθόνης.

Οι αριθμοί του πίνακα είναι στο δεκαεξαδικό σύστημα.

Θέση μνήμης	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
Αριθμός	10	15	05	26	1C	2E	11	3A	0B	3F

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $10
STS $0310, R20

LDI R20, $15
STS $0311, R20

LDI R20, $05
STS $0312, R20

LDI R20, $26
STS $0313, R20

LDI R20, $1C
STS $0314, R20

LDI R20, $2E
STS $0315, R20

LDI R20, $11
STS $0316, R20

LDI R20, $3A
STS $0317, R20

LDI R20, $0B
STS $0318, R20
```

```
LDI R20, $3F
STS $0319, R20

LDI R29, $03
LDI R28, $10        ; αρχικοποίηση του δείκτη Y στη θέση μνήμης 0310

LDI R25, $00        ; μηδενισμός του μετρητή επαναλήψεων

LDI R16,$00

LOOP:
LD R18, Y           ; μέσω του δείκτη Y φορτώνουμε στον R18 την επόμενη κάθε φορά θέση
                    του πίνακα
CP R16, R18         ; σύγκριση των αριθμών που περιέχουν οι δύο καταχωρητές

BRSH NEXT           ; αν ο καταχωρητής R16 είναι μεγαλύτερος από τον
                    R18, συνέχισε στο NEXT, αλλιώς συνέχισε στην
                    επόμενη εντολή
STS $0400, R18      ; βρέθηκε μεγαλύτερη τιμή από την τρέχουσα και ενημερώθηκε ο
μεγαλύτερος αριθμός της μνήμης 0400

NEXT:
LD R16, Y           ; μέσω του δείκτη Y φορτώνουμε στον R16 κάθε φορά την επόμενη θέση
                    του πίνακα
STS $0400, R16      ; μεταφορά του περιεχομένου του καταχωρητή R16 στη μνήμη 0400
INC R28             ; αύξηση της θέσης μνήμης κατά ένα
INC R25             ; αύξηση του αριθμού των επαναλήψεων
CPI R25, $0A        ; σύγκριση του αριθμού των επαναλήψεων
BRNE LOOP           ; όσο ο μετρητής επαναλήψεων είναι μικρότερος του
                    0A, συνέχισε στη LOOP

LDI R30, $FF
OUT DDRB, R30
```

```
COM R18
OUT PORTB, R18

RET
```

Παρατηρήσεις: Όσο ο αριθμός των επαναλήψεων είναι μικρότερος από το 0A, το πρόγραμμα συνεχίζει να λειτουργεί. Μόλις ο μετρητής γίνει ίσος με τον αριθμό 0A, ενημερώνεται η σημαία Z και το πρόγραμμα αρχίζει από την αρχή. Επίσης, πραγματοποιείται σύγκριση μεταξύ των αριθμών που περιέχουν οι δύο καταχωρητές (CP R16, R18) και, αν ο πρώτος αριθμός είναι ίσος ή μεγαλύτερος από το δεύτερο, το πρόγραμμα μεταφέρεται στο σημείο NEXT:. Σε διαφορετική περίπτωση, συνεχίζει με την επόμενη εντολή. Τέλος, συνολικά πραγματοποιούνται 10 επαναλήψεις και ο μεγαλύτερος αριθμός είναι ο αριθμός $3F_{16}$.

Άσκηση 5.2: Να γράψετε ένα πρόγραμμα το οποίο με τη χρησιμοποίηση βρόχου επανάληψης να κάνει την πράξη της πρόσθεσης στους αριθμούς 1+2+3+4+5+6+7. Το τελικό αποτέλεσμα να μεταφέρεται κάθε φορά στη θέση μνήμης $0300. Τέλος, το τελικό αποτέλεσμα να αποτυπωθεί στους λαμπτήρες της πλακέτας.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $07
LDI R21, $00        ; αρχικοποίηση με τον τελευταίο αριθμό (7)

LOOP:
ADD R21, R20        ; πρόσθεση των αριθμών που περιέχουν οι 2 καταχωρητές
STS $0300, R21      ; μεταφορά του αποτελέσματος στη θέση μνήμης 0300
DEC R20             ; μείωση του αριθμού που περιέχει ο R20
CPI R20, 00         ; σύγκριση του αριθμού που έχει ο R20 με το μηδέν
BRNE LOOP           ; όσο ο καταχωρητής R20 είναι διάφορος του 00,
                    συνέχισε στη LOOP

LDI R16, $FF
OUT DDRA, R16
COM R21
```

```
OUT PORTA, R21

RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι, μέχρι να μηδενιστεί ο καταχωρητής R20, πραγματοποιούνται 7 επαναλήψεις. Η πρόσθεση ξεκινά από τον τελευταίο αριθμό (7) και μειώνεται κάθε φορά κατά ένα, μέχρι να φτάσει στο (0), έχοντας προσθέσει όλους τους προηγούμενους αριθμούς. Επίσης, μετά την τρίτη πρόσθεση των περιεχομένων των δύο καταχωρητών, ενημερώνεται η σημαία H, καθώς προκύπτει κρατούμενο από τα λιγότερο σημαντικά ψηφία. Ακόμη, στην τελευταία επανάληψη, όταν και μηδενίζεται ο καταχωρητής R20, ενημερώνεται η σημαία μηδενισμού (Z). Τέλος, το αποτέλεσμα που προκύπτει είναι το $1C_{16}$.

Άσκηση 5.3: Να χρησιμοποιήσετε βρόχο επανάληψης και να γράψετε ένα πρόγραμμα το οποίο να προσθέτει τους αριθμούς 1+2+3+4+5+6+….+N. Το N είναι μία μεταβλητή και βρίσκεται στη θέση μνήμης $0300. Το αποτέλεσμα να τοποθετείται κάθε φορά στη διεύθυνση μνήμης $0400 και να αποτυπώνεται στους λαμπτήρες της οθόνης.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $01
LDI R21, $00
LDI R25, $11          ; μεταβλητή N
STS $0300, R25

LOOP:
ADD R21, R20
STS $0400, R21
INC R20
CP R20, R25
BRNE LOOP

LDI R30, $FF
OUT DDRB, R30
```

```
COM R21
OUT PORTB, R21

RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι πραγματοποιεί 16 επαναλήψεις. Στον καταχωρητή R25 έχουμε εισάγει τη μεταβλητή N και είναι ο αριθμός 11_{16}. Ακόμη, μετά από κάθε σύγκριση μεταξύ των περιεχομένων των καταχωρητών R20 και R25, ενημερώνονται οι εξής σημαίες: η σημαία προσήμου (S), η σημαία αρνητικού αριθμού (N) και η σημαία κρατουμένου (C), εκτός από την τελευταία επανάληψη, όπου το περιεχόμενο των δύο καταχωρητών γίνεται ίσο και μετά από αυτή τη σύγκριση ανανεώνεται μόνο η σημαία μηδενισμού (Z). Επιπλέον, μετά από την $6^η$, $8^η$, $10^η$, $11^η$, $13^η$, $14^η$ και $15^η$ πρόσθεση, ενημερώνεται η σημαία H, που σημαίνει ότι προκύπτει κρατούμενο από τα λιγότερο σημαντικά ψηφία, ενώ, ύστερα από την τελευταία πρόσθεση, ενημερώνεται η σημαία υπερχείλισης (V). Τέλος, το αποτέλεσμα είναι το 88_{16}.

8.7 Εργαστήριο 6

Εισαγωγικά: Αντικείμενο του συγκεκριμένου εργαστηρίου είναι οι υπορουτίνες, η κλήση τους, η επαναφορά τους και οι αντίστοιχες εντολές τους.

Θεωρία: Οι υπορουτίνες ουσιαστικά αποτελούν ανεξάρτητα τμήματα κώδικα, τα οποία ολοκληρώνουν μία συγκεκριμένη εργασία. Οι υπορουτίνες μπορούν να κληθούν όσες φορές απαιτείται και η κλήση τους γίνεται από το κυρίως πρόγραμμα. Επίσης, διαθέτουν και ορισμένα πλεονεκτήματα. Συγκεκριμένα, στην περίπτωση που υπάρχουν ίδια τμήματα κώδικα τα οποία επαναλαμβάνονται, οι υπορουτίνες βοηθούν στη σημαντική μείωση του μεγέθους του κώδικα. Ακόμη, κάνουν πιο εύκολη τη διόρθωση, διότι με το να διορθωθεί ο κώδικας που ανήκει στην υπορουτίνα, στη συνέχεια, η διόρθωση θα ισχύσει και για όλο το υπόλοιπο τμήμα του προγράμματος. Τέλος, οι υπορουτίνες παρέχουν τη δυνατότητα, ώστε ο κώδικας να χρησιμοποιηθεί ξανά, ενώ η χρήση των υπορουτινών βοηθά, ώστε ο κώδικας να γίνεται πιο ευκολονόητος.

Εντολές που χρησιμοποιούνται:

Εντολή CALL (CALL μνήμη): Η εντολή CALL (Direct Subroutine Call) πραγματοποιεί κλήση μιας υπορουτίνας μέσα από τη μνήμη του προγράμματος.

Επίσης, προκαλεί μεταφορά του προγράμματος σε μία υπορουτίνα, η οποία βρίσκεται σε μία θέση μνήμης και έχει δηλωθεί ως παράμετρος.

Σύνταξη: CALL K

Παράδειγμα: CALL LOOP

Εντολή ICALL: Η εντολή ICALL [(Indirect Call to (Z)] καλεί μία υπορουτίνα, η οποία ορίζεται από το δείκτη Z. Η κλήση της υπορουτίνας είναι έμμεση.

Σύνταξη: ICALL

Εντολή RCALL (RCALL μνήμη): Η εντολή RCALL (Relative Subroutine Call) υλοποιεί κλήση μιας υπορουτίνας, η οποία είναι στη διεύθυνση ανάμεσα σε PC – 2K + 1 και PC + 2K.

Σύνταξη: RCALL K

Παράδειγμα: RCALL LOOP

Εντολή RET: Η εντολή RET (Subroutine Return) υλοποιεί επιστροφή από μία υπορουτίνα. Η συγκεκριμένη εντολή επιστρέφει στην αντίστοιχη που βρίσκεται μετά από την εντολή CALL, η οποία προκάλεσε κλήση της υπορουτίνας. Επίσης, η RET είναι η τελευταία εντολή της υπορουτίνας και επιστρέφει την εκτέλεση στο κυρίως πρόγραμμα αμέσως μετά την εκτέλεση των εντολών της υπορουτίνας.

Σύνταξη: RET

Εντολή RETI: Η εντολή RETI (Interrupt Return) πραγματοποιεί επιστροφή από μία διακοπή.

Σύνταξη: RETI

Εντολή PUSH (PUSH καταχωρητής): Η εντολή PUSH (Push Register on stack) μεταφέρει το περιεχόμενο ενός καταχωρητή στη στοίβα.

Σύνταξη: PUSH Rr

Παράδειγμα: PUSH R1

Εντολή POP (POP καταχωρητής): Η εντολή POP (Pop Register from stack) επαναφέρει από τη στοίβα το περιεχόμενο ενός καταχωρητή.

Σύνταξη: POP Rr

Παράδειγμα: POP R1

Άσκηση 6.1: Δίνονται δύο πίνακες δέκα θέσεων, από τους οποίους ο καθένας περιέχει δέκα αριθμούς των 8bits. Τα δεδομένα του πρώτου πίνακα είναι από τις θέσεις μνήμης 0110 έως 0119 και τα δεδομένα του δεύτερου πίνακα είναι από τις

θέσεις μνήμης 0210 έως 0219. Να γράψετε ένα πρόγραμμα το οποίο να πραγματοποιεί σύγκριση μεταξύ των δεδομένων των δύο αυτών πινάκων και η σύγκριση να είναι ανά δύο. Δηλαδή, ο αριθμός της θέσης μνήμης 0110 να συγκρίνεται με τον αριθμό της θέσης μνήμης 0210, ο αριθμός που βρίσκεται στη μνήμη 0111 να συγκρίνεται με τον αριθμό που βρίσκεται στη μνήμη 0211 κτλ. Μετά από κάθε σύγκριση των αριθμών, στις ανάλογες θέσεις ενός τρίτου πίνακα που θα βρίσκεται στις θέσεις μνήμης από 0310 έως 0319 να τοποθετείται DF, αν οι δύο αριθμοί διαφορετικοί, ενώ, αν είναι ίδιοι, να τοποθετείται ο αριθμός 55.

Θέσεις μνήμης πρώτου πίνακα	Δεδομένα πρώτου πίνακα	Θέσεις μνήμης δεύτερου πίνακα	Δεδομένα δεύτερου πίνακα
0110	13	0210	11
0111	35	0211	35
0112	AD	0212	AD
0113	C8	0213	5D
0114	EE	0214	EE
0115	54	0215	77
0116	FA	0216	AF
0117	85	0217	85
0118	BF	0218	BF
0119	EF	0219	CB

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $13
STS $0110, R20
LDI R22, $11
STS $0210, R22

LDI R20, $35
STS $0111, R20
LDI R22, $35
STS $0211, R20
```

```
LDI R20, $AD
STS $0112, R20
LDI R22, $AD
STS $0212, R22

LDI R20, $C8
STS $0113, R20
LDI R22, $5D
STS $0213, R22

LDI R20, $EE
STS $0114, R20
LDI R22, $EE
STS $0214, R22

LDI R20, $54
STS $0115, R20
LDI R22, $77
STS $0215, R22

LDI R20, $FA
STS $0116, R20
LDI R22, $AF
STS $0216, R22

LDI R20, $85
STS $0117, R20
LDI R22, $85
STS $0217, R22

LDI R20, $BF
STS $0118, R20
LDI R22, $BF
STS $0218, R22
```

```
LDI R20, $EF
STS $0119, R20
LDI R22, $CB
STS $0219, R22

LDI R16, $DF        ; αν οι αριθμοί είναι διαφορετικοί
LDI R17, $55        ; αν οι αριθμοί είναι ίδιοι
LDI R25, $00        ; μετρητής επαναλήψεων

LDI R27, $01
LDI R26, $10        ; θέση μνήμης 0110
LDI R29, $02
LDI R28, $10        ; θέση μνήμης 0210
LDI R31, $03
LDI R30, $10        ; θέση μνήμης 0310

LOOP:
LD R1, X+     ; μεταφορά του περιεχομένου της θέσης μνήμης στον καταχωρητή R1
LD R5, Y+     ; μεταφορά του περιεχομένου της θέσης μνήμης στον καταχωρητή R5
CP R5, R1
BREQ EQUAL
ST Z+, R16    ; αποθήκευση του DF στη θέση μνήμης που δείχνει ο Z (ξεκινά από 0310)
JMP NEXT

EQUAL:
ST Z+, R17    ; αποθήκευση του 55 στη θέση μνήμης που δείχνει ο Z
JMP NEXT

NEXT:
INC R25       ; αύξηση κατά ένα του μετρητή επαναλήψεων
CPI R25, $0A ; σύγκριση του μετρητή επαναλήψεων με τον αριθμό 0A
BRNE LOOP
```

RET

Παρατηρήσεις: Η αύξηση των δεικτών X, Y και Z κατά 1 γίνεται λόγω της προσθήκης του χαρακτήρα "+". Οι δείκτες διεύθυνσης X, Y και Z δείχνουν τις θέσεις μνήμης 0110, 0210 και 0310, δηλαδή τους καταχωρητές R26:R27, R28:R29 και R30:R31 αντίστοιχα. Με τις εντολές LD R1, X+ και LD R5, Y+ γίνεται μεταφορά των δεδομένων από τις θέσεις μνήμης (ξεκινώντας από 0110 και 0210 αντίστοιχα) στους καταχωρητές R1 και R5, ενώ, κάθε φορά που το πρόγραμμα θα μπαίνει στη LOOP, οι θέσεις μνήμης θα αυξάνονται κατά ένα και θα μεταφέρουν στους καταχωρητές τα αντίστοιχα δεδομένα. Ο δείκτης Z+, ανάλογα με το αποτέλεσμα που προκύπτει από τη σύγκριση των περιεχομένων των δύο καταχωρητών, δέχεται και το αντίστοιχο περιεχόμενο του καταχωρητή R16 ή R17. Επίσης, ο συγκεκριμένος δείκτης διεύθυνσης στην πρώτη επανάληψη δείχνει τη θέση μνήμης 0310 και, στη συνέχεια, με κάθε επανάληψη που υλοποιείται αυξάνεται κατά ένα. Εκτελώντας το πρόγραμμα, παρατηρούμε ότι το πλήθος των επαναλήψεων είναι 10. Όταν πραγματοποιείται σύγκριση μεταξύ των περιεχομένων των πινάκων, ενημερώνονται οι σημαίες H, S, N και C. Αντίθετα, όταν οι δύο αριθμοί είναι ίσοι, ενημερώνεται μόνο η σημαία Z. Όταν πραγματοποιείται η σύγκριση του μετρητή των επαναλήψεων με τον αριθμό 10, ενημερώνονται οι σημαίες S και N και, όταν γίνει ίσος με το όριο των επαναλήψεων, το πρόγραμμα τερματίζει. Η τελική μορφή του πίνακα παρουσιάζεται παρακάτω:

Τελική μορφή πίνακα:

Θέσεις μνήμης πρώτου πίνακα	Δεδομένα πρώτου πίνακα	Θέσεις μνήμης δεύτερου πίνακα	Δεδομένα δεύτερου πίνακα	Θέσεις μνήμης τρίτου πίνακα	Αποτελέσματα
0110	13	0210	11	0310	DF
0111	35	0211	35	0311	55
0112	AD	0212	AD	0312	55
0113	C8	0213	5D	0313	DF
0114	EE	0214	EE	0314	55
0115	54	0215	77	0315	DF
0116	FA	0216	AF	0316	DF
0117	85	0217	85	0317	55
0118	BF	0218	BF	0318	55
0119	EF	0219	CB	0319	DF

Άσκηση 6.2: Να γράψετε ένα πρόγραμμα το οποίο να πραγματοποιεί την πράξη της πρόσθεσης ανάμεσα σε δύο πίνακες. Ο κάθε πίνακας θα έχει δέκα θέσεις. Ο πρώτος πίνακας θα είναι στη θέση μνήμης 0110 και ο δεύτερος θα είναι στη θέση μνήμης 0210. Το αποτέλεσμα να αποθηκεύεται σε έναν τρίτο πίνακα, ο οποίος να είναι στη θέση μνήμης 0310.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R25, $00          ; αριθμός επαναλήψεων

LDI R27, $01
LDI R26, $10
LDI R29, $02
LDI R28, $10
LDI R31, $03
LDI R30, $10

LOOP:
LD R1, X+     ; μεταφορά του περιεχομένου της θέσης μνήμης στον καταχωρητή R1
LD R5, Y+     ; μεταφορά του περιεχομένου της θέσης μνήμης στον καταχωρητή R5
ADD R5, R1
ST Z+, R5     ; αποθήκευση του αποτελέσματος της πρόσθεσης στην ανάλογη θέση
              ; μνήμης
INC R25       ; αύξηση του μετρητή επαναλήψεων
CPI R25, $0A ; σύγκριση του μετρητή επαναλήψεων με τον αριθμό 0A
BRNE LOOP

RET
```

Παρατηρήσεις: Η αύξηση των δεικτών X, Y και Z κατά 1 γίνεται λόγω της προσθήκης του χαρακτήρα "+". Οι δείκτες διεύθυνσης X, Y και Z δείχνουν τις θέσεις μνήμης 0110, 0210 και 0310, δηλαδή τους καταχωρητές R26:R27, R28:R29 και R30:R31 αντίστοιχα. Με τις εντολές LD R1, X+ και LD R5, Y+ γίνεται μεταφορά των δεδομένων από τις θέσεις μνήμης (ξεκινώντας από 0110 και 0210 αντίστοιχα)

στους καταχωρητές R1 και R5, ενώ, κάθε φορά που το πρόγραμμα θα μπαίνει στη LOOP, οι θέσεις μνήμης θα αυξάνονται κατά ένα. Στο δείκτη Z αποθηκεύεται το αποτέλεσμα της πρόσθεσης. Επίσης, ο συγκεκριμένος δείκτης διεύθυνσης στην πρώτη επανάληψη δείχνει τη θέση μνήμης 0310 και, στη συνέχεια, με κάθε επανάληψη που πραγματοποιείται αυξάνεται κατά ένα. Μετά τη σύγκριση του μετρητή επαναλήψεων με το όριο που έχουμε θέσει για τις επαναλήψεις, ανανεώνονται οι σημαίες H, S, N και V. Όταν ο μετρητής γίνει ίσος με 10, τότε το πρόγραμμα τερματίζεται.

Άσκηση 6.3: Να τροποποιήσετε το προηγούμενο πρόγραμμα, ώστε να χρησιμοποιήσετε μία υπορουτίνα για την πρόσθεση κάθε στοιχείου των δύο αυτών πινάκων.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R25, $00            ; μετρητής επαναλήψεων

LDI R27, $01
LDI R26, $10
LDI R29, $02
LDI R28, $10
LDI R31, $03
LDI R30, $10

LOOP:
CALL ROUTINE
INC R25
CPI R25, $0A
BRNE LOOP

ROUTINE:
LD R1, X+     ; μεταφορά του περιεχομένου της μνήμης στον R1
LD R5, Y+     ; μεταφορά του περιεχομένου της μνήμης στον R5
ADD R5, R1
```

```
ST Z+, R5       ; αποθήκευση του αποτελέσματος στη θέση μνήμης που δείχνει ο δείκτης Z

RET
```

Παρατηρήσεις: Οι δείκτες διεύθυνσης X, Y και Z δείχνουν τις θέσεις μνήμης 0110, 0210 και 0310 αντίστοιχα. Με τις εντολές LD R1, X+ και LD R5, Y+ γίνεται μεταφορά των δεδομένων από τις θέσεις μνήμης (ξεκινώντας από 0110 και 0210 αντίστοιχα) στους καταχωρητές R1 και R5, ενώ, κάθε φορά που το πρόγραμμα θα μπαίνει στη LOOP, οι θέσεις μνήμης θα αυξάνονται κατά ένα. Στο δείκτη Z αποθηκεύεται το αποτέλεσμα της πρόσθεσης. Επίσης, ο συγκεκριμένος δείκτης διεύθυνσης στην πρώτη επανάληψη δείχνει τη θέση μνήμης 0310 και, στη συνέχεια, με κάθε επανάληψη που υλοποιείται αυξάνεται κατά ένα. Μετά τη σύγκριση του μετρητή επαναλήψεων με τον αριθμό 10, ανανεώνονται οι σημαίες H, S, N και V. Όσο δεν γίνεται ο μετρητής ίσος με 10, το πρόγραμμα συνεχίζεται και μετά τη σύγκριση επιστρέφει στο βρόχο. Όταν ο μετρητής γίνει ίσος με 10, τότε το πρόγραμμα τερματίζεται.

Άσκηση 6.4: Να τροποποιήσετε την προηγούμενη άσκηση, ώστε να πραγματοποιείται πρόσθεση μεταξύ αριθμών των 16bit. Δίνονται οι αριθμοί AA23 και 54C5.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $23
STS $0200, R20      ; χαμηλής τάξης byte
LDI R20, $C5
STS $0300, R20      ; χαμηλής τάξης byte
LDI R20, $AA
STS $0201, R20      ; υψηλής τάξης byte
LDI R20, $54
STS $0301, R20      ; υψηλής τάξης byte

LDI R27, $02
LDI R26, $00
```

```
LDI R29, $03
LDI R28, $00
LDI R31, $04
LDI R30, $00
CLC                 ; μηδενισμός του κρατουμένου
LDI R16, $00        ; μηδενισμός του μετρητή επαναλήψεων

LOOP:
LD R18, X+          ; μεταφορά των αριθμών στον καταχωρητή R18
LD R19, Y+          ; μεταφορά των αριθμών στον καταχωρητή R19
ADC R18, R19        ; πρόσθεση των δύο αριθμών με κρατούμενο
ST Z+, R18    ; αποθήκευση του αποτελέσματος στην ανάλογη θέση μνήμης
INC R16       ; αύξηση του μετρητή επαναλήψεων
CPI R16, $02  ; σύγκριση του μετρητή επαναλήψεων με τον αριθμό 02
BRNE LOOP

RET
```

Παρατηρήσεις: Οι δείκτες X, Y και Z δείχνουν τις θέσεις μνήμης 0200, 0300 και 0400 αντίστοιχα. Όταν οι θέσεις μνήμης αυξάνονται κατά ένα, δείχνουν τη 0201 και 0301 αντίστοιχα. Αντίθετα, με την εντολή STD Z+, R18 το αποτέλεσμα της πρόσθεσης τοποθετείται στη θέση μνήμης 0401. Με την πρόσθεση των δύο αριθμών 16 bits με κρατούμενο (αφού πρώτα το μηδενίζουμε με την CLC) ενημερώνονται οι σημαίες S και N. Το αποτέλεσμα της πρώτης πρόσθεσης είναι $E8_{16}$ και της δεύτερης πρόσθεσης είναι FF_{16}.

8.8 Εργαστήριο 7

Εισαγωγικά: Στο συγκεκριμένο εργαστήριο θα δούμε πίνακες αναφοράς, ένθετους βρόχους, διδιάστατους πίνακες και το πώς ταξινομείται ένας πίνακας.

Θεωρία: Όταν σε έναν πίνακα έχουμε δύο σύνολα, από τα οποία το πρώτο σύνολο περιέχει ορισμένους ακέραιους αριθμούς και το δεύτερο έχει ορισμένα σύμβολα ή αριθμητικές πληροφορίες και τα δύο αυτά σύνολα αντιστοιχίζονται μεταξύ τους, τότε ο πίνακας ονομάζεται πίνακας αναφοράς. Ο δείκτης σε έναν πίνακα

είναι ένας ακέραιος αριθμός και χρησιμοποιείται, ώστε να μπορούμε να παίρνουμε το αντίστοιχο στοιχείο του πίνακα.

Οι ένθετοι βρόχοι είναι δύο βρόχοι επανάληψης, από τους οποίους ο ένας βρίσκεται μέσα στον άλλον και η εφαρμογή τους πραγματοποιείται με δύο διακλαδώσεις σε προηγούμενες διευθύνσεις, ενώ η μία διακλάδωση περικλείεται μέσα στην άλλη. Οι ένθετοι βρόχοι είναι χρήσιμοι στη δημιουργία αρκετών προγραμμάτων, όπως είναι οι αλγόριθμοι ταξινόμησης, η επεξεργασία διδιάστατων, τριδιάστατων και γενικά πολυδιάστατων πινάκων.

Έστω ότι έχουμε έναν πίνακα ο οποίος αποτελείται από τέσσερις γραμμές και η κάθε γραμμή περιέχει τρία στοιχεία. Η αποθήκευση στη μνήμη ενός διδιάστατου πίνακα γίνεται με σειριακό τρόπο. Αυτό σημαίνει ότι πρώτα αποθηκεύονται τα στοιχεία της πρώτης γραμμής, στη συνέχεια τα στοιχεία της δεύτερης γραμμής, κατόπιν τα στοιχεία της τρίτης γραμμής και ακολουθούν τα στοιχεία της τέταρτης γραμμής.

A_{11}	A_{12}	A_{13}
A_{21}	A_{22}	A_{23}
A_{31}	A_{32}	A_{33}
A_{41}	A_{42}	A_{43}

Η αποθήκευση των στοιχείων στη μνήμη θα είναι:

0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	0310	0311
A_{11}	A_{12}	A_{13}	A_{21}	A_{22}	A_{23}	A_{31}	A_{32}	A_{33}	A_{41}	A_{42}	A_{43}

Προκειμένου να πραγματοποιηθεί ταξινόμηση πινάκων, έχουν δημιουργηθεί διάφοροι αλγόριθμοι ταξινόμησης. Ο κάθε αλγόριθμος έχει τη δική του απόδοση και σε μεγάλο βαθμό η απόδοση ενός αλγορίθμου εξαρτάται από το κατά πόσο είναι ταξινομημένα τα δεδομένα ενός πίνακα, αλλά και από το πλήθος των δεδομένων που πρέπει να επεξεργαστεί ο αλγόριθμος.

Ένας από τους πιο γνωστούς αλγόριθμους ταξινόμησης είναι ο αλγόριθμος Bubblesort, στον οποίο γίνεται ταξινόμηση κατά αύξουσα σειρά. Πιο συγκεκριμένα, πραγματοποιείται σύγκριση μεταξύ των δύο πρώτων στοιχείων του πίνακα (του

πρώτου και του δεύτερου). Σε περίπτωση που το πρώτο στοιχείο του πίνακα είναι μεγαλύτερο από το δεύτερο, τότε τα δύο στοιχεία αλλάζουν θέσεις (το δεύτερο στοιχείο μεταφέρεται στη θέση του πρώτου και το πρώτο στοιχείο μεταφέρεται στη θέση του δεύτερου). Εν συνεχεία, το δεύτερο στοιχείο συγκρίνεται με το τρίτο στοιχείο του πίνακα και, αν από τη μεταξύ τους σύγκριση προκύψει ότι το δεύτερο στοιχείο είναι μεγαλύτερο από το τρίτο, τότε τα δύο αυτά στοιχεία αλλάζουν θέσεις. Η διαδικασία συνεχίζεται, μέχρι να πραγματοποιηθεί η σύγκριση όλων των αριθμών. Στο τέλος του πρώτου ελέγχου, το μεγαλύτερο στοιχείο θα βρίσκεται στην τελευταία θέση του πίνακα. Οι έλεγχοι που θα πραγματοποιήσει ο αλγόριθμος στον πίνακα θα είναι τόσοι όσοι χρειάζονται, ώστε στον τελευταίο έλεγχο να μην γίνει καμία αλλαγή μεταξύ των στοιχείων.

Έστω ότι έχουμε τον εξής πίνακα:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αριθμός	18	15	16	5	6	3

Μετά τον πρώτο έλεγχο, ο πίνακας θα έχει τη μορφή:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αριθμός	15	16	5	6	3	18

Μετά το δεύτερο έλεγχο:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αριθμός	15	5	6	3	16	18

Μετά τον τρίτο έλεγχο:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αριθμός	5	6	3	15	16	18

Μετά τον τέταρτο έλεγχο, ο πίνακας θα έχει τη μορφή:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αριθμός	5	3	6	15	16	18

Μετά τον πέμπτο έλεγχο, ο πίνακας θα είναι:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αριθμός	3	5	6	15	16	18

Μετά το τέλος του πέμπτου ελέγχου, ο πίνακας θα έχει πάρει την τελική του μορφή. Όμως, θα πραγματοποιηθεί και έκτος έλεγχος στον πίνακα, κατά τον οποίο δεν θα συμβεί καμία αλλαγή, αφού τα στοιχεία έχουν ταξινομηθεί, οπότε το πρόγραμμα θα φτάσει στο τέλος του.

Άσκηση 7.1: Χρησιμοποιώντας τον αλγόριθμο Bubblesort, να γράψετε ένα πρόγραμμα το οποίο να ταξινομεί τους 10 αριθμούς (οι αριθμοί είναι των 8bits) που δίνονται στον παρακάτω πίνακα. Η ταξινόμηση να είναι κατά αύξουσα σειρά. Επίσης, οι αριθμοί να βρίσκονται στις θέσεις μνήμης 0301 έως 030A.

Θέση μνήμης	0301	0302	0303	0304	0305	0306	0307	0308	0309	030A
Αριθμός	EE	B7	FF	AB	C0	54	99	A0	23	C2

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

 LDI R16, $EE
STS $0301, R16

LDI R16, $B7
STS $0302, R16

LDI R16, $FF
STS $0303, R16

LDI R16, $AB
STS $0304, R16

LDI R16, $C0
STS $0305, R16
```

```
LDI R16, $54
STS $0306, R16

LDI R16, $99
STS $0307, R16

LDI R16, $A0
STS $0308, R16

LDI R16, $23
STS $0309, R16

LDI R16, $C2
STS $030A, R16

 ARXH:
 LDI R31, $03
 LDI R30, $01
LDI R28, $00            ; αριθμός αλλαγών

 LOOP:
 LD R21, Z
 LDD R23, Z+1
 CP R23, R21
 BRSH NO_CHANGE
 ST Z, R23
 STD Z+1, R21
 INC R28

 NO_CHANGE:
 INC R30
 CPI R30, $0A
 BRNE LOOP
```

```
CPI R28, $00
BRNE ARXH

RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι, όταν συγκρίνεται το περιεχόμενο των δύο καταχωρητών, ενημερώνονται οι σημαίες H, S, N και C. Όταν συγκρίνεται η μνήμη με τον αριθμό 0A, ενημερώνονται οι σημαίες S και N. Το πρόγραμμα τερματίζεται, όταν δεν θα πραγματοποιηθεί στον πίνακα καμία αλλαγή.

Τελική μορφή πίνακα:

Θέση μνήμης	0301	0302	0303	0304	0305	0306	0307	0308	0309	030A
Αριθμός	23	54	99	A0	AB	B7	C0	C2	EE	FF

Άσκηση 7.2: Να γράψετε ένα πρόγραμμα το οποίο να προσθέτει όλα τα στοιχεία σε ένα διδιάστατο πίνακα. Ο πίνακας να είναι τεσσάρων γραμμών και τεσσάρων στηλών (4X4). Τα στοιχεία του πίνακα να βρίσκονται στις θέσεις μνήμης από 0300 έως 030F, όπου από τη θέση 0300 έως 0303 να είναι η πρώτη γραμμή, από 0304 έως 0307 να είναι η δεύτερη, από 0308 έως 030B να είναι η τρίτη και από 030C έως 030F η τέταρτη γραμμή. Επίσης, το πρόγραμμα να περιέχει μετρητές γραμμών, στηλών και επαναλήψεων.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R16, $00  ; μετρητής γραμμών
LDI R17, $00  ; μετρητής στηλών
LDI R18, $00  ; μετρητής επαναλήψεων

LDI R31, $03
LDI R30, $00

LDI R19, $00
```

```
LOOP:
LD R2, Z+
ADD R1, R2
INC R18        ; αύξηση μετρητή επαναλήψεων
INC R17        ; αύξηση μετρητή στήλης
CPI R17, $04   ; σύγκριση του αριθμού στήλης με τον αριθμό 04
BRNE LOOP
LDI R17, $00
INC R16        ; αύξηση μετρητή γραμμής
CPI R16, $04   ; σύγκριση του αριθμού γραμμής με τον αριθμό 04
BRNE LOOP

RET
```

Παρατηρήσεις: Εκτελώντας το πρόγραμμα, παρατηρούμε ότι, κάθε φορά που γίνονται οι συγκρίσεις των δύο μετρητών των γραμμών και των στηλών με τον αριθμό τέσσερα, ενημερώνονται οι σημαίες H, S, N και C. Όταν οι μετρητές των γραμμών και των στηλών δεν είναι ίσοι με το τέσσερα, το πρόγραμμα μεταφέρεται στην αρχή του βρόχου. Στην περίπτωση που ο μετρητής της στήλης γίνεται ίσος με τέσσερα, τότε το πρόγραμμα συνεχίζεται προς τα κάτω, με αποτέλεσμα να αυξάνεται ο μετρητής της γραμμής και ο μετρητής της στήλης να μηδενίζεται. Όταν και ο μετρητής της γραμμής γίνει ίσος με το τέσσερα, τότε το πρόγραμμα τελειώνει. Επίσης, όταν οι δύο μετρητές γίνουν ίσοι με τον αριθμό τέσσερα, ενημερώνεται η σημαία Z.

8.9 Εργαστήριο 8

Εισαγωγικά: Στο συγκεκριμένο εργαστήριο θα ασχοληθούμε με τα αλφαριθμητικά, την αναζήτηση χαρακτήρων και την εκτέλεση προγραμμάτων διαχείρισης αλφαριθμητικών.

Θεωρία: Στους μικροελεγκτές AVR δεν υπάρχουν εντολές για τα αλφαριθμητικά, όμως μπορούν να τα διαχειριστούν με δεικτοδοτούμενη διευθυνσιοδότηση. Τα αλφαριθμητικά αποτελούν πίνακες αριθμών, όπου, μέσω της κωδικοποίησης ASCII, ο κάθε αριθμός αντιστοιχεί σε ένα χαρακτήρα. Για

παράδειγμα, έστω ότι έχουμε τη λέξη JANUARY. Τότε, ο κάθε χαρακτήρας αντιστοιχεί στον αριθμό που δείχνει ο παρακάτω πίνακας:

Θέση μνήμης	0200	0201	0202	0203	0204	0205	0206	0207
Αλφαριθμητικό	J	A	N	U	A	R	Y	
Δεκαεξαδικός αριθμός	4A	41	4E	55	41	52	59	00

Στην τελευταία θέση μνήμης υπάρχει ο αριθμός μηδέν. Με το συγκεκριμένο αριθμό τερματίζει ένα οποιοδήποτε αλφαριθμητικό. Για τη χρησιμοποίηση των αλφαριθμητικών ο συγκεκριμένος κανόνας αποτελεί μία σύμβαση και ισχύει σε πολλές γλώσσες προγραμματισμού και, κυρίως, στη γλώσσα C. Τέλος, στους μικροελεγκτές AVR τα αλφαριθμητικά είναι σαν μονοδιάστατοι πίνακες αριθμών.

Άσκηση 8.1: Δίνεται στη θέση μνήμης 0140 η τιμή AE. Να γράψετε ένα πρόγραμμα το οποίο θα αναζητά τη συγκεκριμένη τιμή στον πίνακα (παρακάτω δίνεται ο πίνακας με τις τιμές) που είναι αποθηκευμένος στις θέσεις μνήμης από 0300 έως 030A. Από τις θέσεις μνήμης 0400 και μετά, θα τοποθετείται το σημείο του πίνακα στο οποίο βρίσκεται η τιμή που αναζητείται.

Πίνακας:

Θέση μνήμης	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	030A
Τιμή	18	BB	AE	F2	AA	AE	EA	33	AE	66	AE

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $18
STS $0300, R20

LDI R20, $BB
STS $0301, R20

LDI R20, $AE
STS $0302, R20
```

```
LDI R20, $F2
STS $0303, R20

LDI R20, $AA
STS $0304, R20

LDI R20, $AE
STS $0305, R20

LDI R20, $EA
STS $0306, R20

LDI R20, $33
STS $0307, R20

LDI R20, $AE
STS $0308, R20

LDI R20, $66
STS $0309, R20

LDI R20, $AE
STS $030A, R20

LDI R20, $AE
STS $0140, R20

LDI R29, $03
LDI R28, $00        ; ορίζεται η θέση μνήμης 0300
LDI R31, $04
LDI R30, $00        ; ορίζεται η θέση μνήμης 0400

LDI R25, $00
```

```
LOOP:
LD R1, Y+       ; μεταφορά του περιεχομένου της μνήμης στον R1
CP R20, R1
BRNE NEXT
ST Z+, R25      ; αποθήκευση του αποτελέσματος στην ανάλογη θέση μνήμης


NEXT:
INC R25         ; αύξηση του μετρητή επαναλήψεων
CPI R25, $0B    ; σύγκριση του μετρητή επαναλήψεων με τον αριθμό 0B
BRNE LOOP

RET
```

Παρατηρήσεις: Στην αρχή του προγράμματος καταχωρούνται συγκεκριμένες τιμές στον πίνακα στις διευθύνσεις 0300 ως 030A. Στο πρόγραμμα αναζητείται η τιμή AE. Εκτελώντας το πρόγραμμα, παρατηρούμε ότι στη θέση μνήμης 0400 αντιστοιχεί ο αριθμός 02, στην 0401 ο αριθμός 05, στην 0402 ο αριθμός 08 και στην 0403 ο αριθμός 0A. Επομένως, το στοιχείο που αναζητούμε βρίσκεται στην $3^{η}$, στην $6^{η}$, στην $9^{η}$ και στην $11^{η}$ θέση του πίνακα.

Άσκηση 8.2: Να γράψετε ένα πρόγραμμα το οποίο θα πραγματοποιεί αντιγραφή ενός αλφαριθμητικού που βρίσκεται στις θέσεις μνήμης από 0300 έως 0308, ενώ το μέγιστο μήκος του αλφαριθμητικού να είναι οκτώ χαρακτήρων. Επίσης, το τέλος του αλφαριθμητικού να συμβολίζεται με τον αριθμό 00 και η αντιγραφή του αλφαριθμητικού να ξεκινά από τη θέση μνήμης 0400.

Αλφαριθμητικό:

Θέση μνήμης	0300	0301	0302	0303	0304	0305	0306
Αλφαριθμητικό	F	R	I	D	A	Y	
Δεκαεξαδικός αριθμός	46	52	49	44	41	59	00

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"
```

```
LDI R20, $46
STS $0300, R20

LDI R20, $52
STS $0301, R20

LDI R20, $49
STS $0302, R20

LDI R20, $44
STS $0303, R20

LDI R20, $41
STS $0304, R20

LDI R20, $59
STS $0305, R20

LDI R20, $00
STS $0306, R20

LDI R29, $03
LDI R28, $00          ; ορισμός της θέσης μνήμης 0300
LDI R31, $04
LDI R30, $00          ; ορισμός της θέσης μνήμης 0400

LOOP:
LD R16, Y       ; μεταφορά του περιεχομένου της μνήμης στον R16
ST Z+, R16      ; αποθήκευση του αποτελέσματος στη θέση μνήμης που δείχνει ο δείκτης Z
CPI R16, 00
BREQ TELOS
INC R28       ; αύξηση της θέσης μνήμης που δείχνει ο Y κατά ένα
CPI R28, $08
```

```
BRNE LOOP

TELOS:
RET
```

Παρατηρήσεις: Ο δείκτης διεύθυνσης Y δείχνει τη θέση μνήμης 0300. Στη συνέχεια, αντιγράφεται κάθε ψηφίο του αλφαριθμητικού στη θέση μνήμης 0400 και μετά εκεί που δείχνει ο δείκτης Z. Εκτελώντας το πρόγραμμα, παρατηρούμε ότι, όταν δίνεται ο αριθμός 00, που συμβολίζει και το τέλος του αλφαριθμητικού, το πρόγραμμα τερματίζει.

Άσκηση 8.3: Δίνεται ένα αλφαριθμητικό που είναι γραμμένο με κεφαλαία γράμματα. Να γράψετε ένα πρόγραμμα το οποίο θα μετατρέπει τα κεφαλαία γράμματα σε μικρά. Το αλφαριθμητικό να ξεκινά από τη θέση μνήμης 0300, το τέλος του να συμβολίζεται με τον αριθμό 00 και να έχει μέγιστο μήκος οκτώ χαρακτήρες. Επίσης, το αλφαριθμητικό να αποθηκευτεί στη θέση μνήμης 0400.

Υπόδειξη: Στον κώδικα ASCII τα αλφαριθμητικά περιλαμβάνουν γράμματα-αριθμούς και σύμβολα. Το κάθε γράμμα ενός αλφαριθμητικού αντιστοιχεί σε έναν αριθμό (από 0 έως 255). Για παράδειγμα, για τα κεφαλαία γράμματα ισχύει: $A = 41_{16}$, $B = 42_{16}$, $C = 43_{16}$, $D = 44_{16}$, $E = 45_{16}$, $F = 46_{16}$, ……., $X = 58_{16}$, $Y = 59_{16}$, $Z = 5A_{16}$ και για τα μικρά γράμματα ισχύει: $a = 61_{16}$, $b = 62_{16}$, $c = 63_{16}$, $d = 64_{16}$, $e = 65_{16}$, $f = 66_{16}$, ……., $x = 78_{16}$, $y = 79_{16}$, $z = 7A_{16}$. Επομένως, τα κεφαλαία και τα πεζά γράμματα είναι κωδικοποιημένα στη σειρά και δύο αντίστοιχα γράμματα έχουν διαφορά μεταξύ τους κατά 20_{16}. Τέλος, στην αρχή του βρόχου να πραγματοποιείται έλεγχος του αλφαριθμητικού, διότι από την αρχή μπορεί να περιλαμβάνει τον αριθμό 0. Επίσης, το αλφαριθμητικό να αποθηκευτεί στη θέση μνήμης 0400.

Αλφαριθμητικό:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αλφαριθμητικό	M	O	N	T	H	
Δεκαεξαδικός αριθμός	4D	4F	4E	54	48	00

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $4D
```

```
STS $0300, R20

LDI R20, $4F
STS $0301, R20

LDI R20, $4E
STS $0302, R20

LDI R20, $54
STS $0303, R20

LDI R20, $48
STS $0304, R20

LDI R20, $00
STS $0305, R20

LDI R29, $03
LDI R28, $00          ; ορισμός της θέσης μνήμης 0300
LDI R31, $04
LDI R30, $00          ; ορισμός της θέσης μνήμης 0400

LDI R17, $20

LOOP:
LD R16, Y     ; μεταφορά των δεδομένων της θέσης μνήμης στον καταχωρητή R16
CPI R16, 00
BREQ TELOS
ADD R16, R17
ST Z+, R16    ; αποθήκευση του αποτελέσματος στη μνήμη που δείχνει ο Z
INC R28               ; αύξηση της θέσης μνήμης που δείχνει ο Y κατά ένα
CPI R28, $08
BRNE LOOP
```

```
TELOS:
LDI R17, $00
ST Z, R17

RET
```

Παρατηρήσεις: Ο δείκτης διεύθυνσης Y δείχνει τη θέση μνήμης 0300 και στη συνέχεια η θέση μνήμης αυξάνεται κατά ένα. Ο δείκτης Z ξεκινά δείχνοντας τη θέση μνήμης 0400 και κατά τη διάρκεια της εκτέλεσης του προγράμματος το αποτέλεσμα αποθηκεύεται στις επόμενες θέσεις μνήμης (0401, 0402 κτλ). Εκτελώντας το πρόγραμμα, παρατηρούμε ότι, όταν δίνεται ο αριθμός 00, που συμβολίζει και το τέλος του αλφαριθμητικού, το πρόγραμμα τερματίζει. Τα αποτελέσματα παρατίθενται στον παρακάτω πίνακα.

Αποτελέσματα:

Θέση μνήμης	0300	0301	0302	0303	0304	0305
Αλφαριθμητικό	M	O	N	T	H	
Δεκαεξαδικός αριθμός	4D	4F	4E	54	48	00
Θέση μνήμης	0400	0401	0402	0403	0404	
Αλφαρηθμιτικό	m	o	n	t	h	
Δεκαεξαδικός Αριθμός	6D	6F	6E	74	68	00

8.10 Εργαστήριο 9

Εισαγωγικά: Στο συγκεκριμένο εργαστήριο θα δούμε τις λογικές εντολές, τον έλεγχο της τιμής ενός bit, το πώς καθορίζουμε τιμή σε ένα bit, καθώς και τις εντολές ολίσθησης.

Θεωρία: Εκτός από τις αριθμητικές πράξεις, υπάρχει και η δυνατότητα να πραγματοποιούνται λογικές πράξεις στα δεδομένα των καταχωρητών, τα οποία δεδομένα είναι δυαδικής μορφής. Τα περιεχόμενα που υπάρχουν στους καταχωρητές και στις μνήμες αντιμετωπίζονται από τις λογικές πράξεις ως πίνακες των bits και εφαρμόζουν πράξεις με τα αντίστοιχα bits των δύο τελεσταίων που συμμετέχουν στην πράξη αυτή. Με τις λογικές πράξεις υπάρχει η δυνατότητα να αλλάξει η τιμή ενός bit, είτε σε κάποιον καταχωρητή είτε σε κάποια θέση μνήμης, χωρίς, μάλιστα, να

αλλάξουν τα υπόλοιπα bits. Επίσης, υπάρχει η δυνατότητα να γίνει έλεγχος στην τιμή ενός bit σε έναν καταχωρητή ή σε μία θέση μνήμης. Όλα αυτά χρησιμεύουν στον προγραμματισμό των περιφερειακών συσκευών, διότι οι πληροφορίες που στέλνονται στις θύρες εξόδου ή λαμβάνονται στις θύρες εισόδου είναι από bits. Στο πλαίσιο αυτό, το κάθε bit έχει τη δική του σημασία για την κατάσταση στην οποία βρίσκεται η περιφερειακή συσκευή αλλά και για τη λειτουργία της.

I. Αλλαγή της τιμής σε ένα bit

1. Μηδενισμός ενός bit:

Αν θέλουμε να αλλάξουμε το bit ενός αριθμού που βρίσκεται σε δυαδική μορφή και να το μετατρέψουμε σε μηδέν, τότε πρέπει να χρησιμοποιήσουμε τη λογική πράξη AND (ΚΑΙ) μεταξύ του αριθμού και μιας «μάσκας», η οποία στη θέση του ψηφίου που θέλουμε να γίνει μηδέν θα έχει το μηδέν. Επίσης, εφόσον θέλουμε τα υπόλοιπα bits του αριθμού να παραμείνουν αμετάβλητα, η μάσκα στις υπόλοιπες θέσεις της θα έχει τον αριθμό ένα.

Έστω ότι έχουμε τον αριθμό $C9_{16}$

Δυαδική μορφή $C9_{16}$	1	1	0	0	1	0	0	1
Bits	B7	B6	B5	B4	B3	B2	B1	B0

και θέλουμε να μηδενίσουμε το Bit7 και όλα τα υπόλοιπα bits να παραμείνουν αμετάβλητα. Σ' αυτή την περίπτωση, το πρόγραμμα θα περιλαμβάνει το λογικό AND μεταξύ του δυαδικού αριθμού και μιας μάσκας που θα έχει περιέχει έναν αριθμό, του οποίου όλα τα ψηφία θα είναι ένα οποιοδήποτε εκτός από το Bit7, το οποίο θα είναι μηδέν.

Δυαδική μορφή $C9_{16}$	**1**	1	0	0	1	0	0	1
Αριθμός μάσκας	**0**	1	1	1	1	1	1	1
Νέος αριθμός	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
Bits	B7	B6	B5	B4	B3	B2	B1	B0

Επομένως, ο νέος αριθμός που προκύπτει μετά από τη μετατροπή είναι ο αριθμός 01001001_2, που σε δεκαεξαδική μορφή είναι ο αριθμός 49_{16}.

2. **Τοποθέτηση του αριθμού ένα σε ένα bit:**

Αν θέλουμε να αλλάξουμε το bit ενός αριθμού που βρίσκεται σε δυαδική μορφή και να μετατρέψουμε το συγκεκριμένο bit σε ένα, τότε πρέπει να χρησιμοποιήσουμε τη λογική πράξη OR (Ή) μεταξύ του αριθμού και μιας «μάσκας», η οποία στη θέση του ψηφίου που θέλουμε να γίνει ένα θα έχει το ένα. Επίσης, εφόσον θέλουμε τα υπόλοιπα bits του αριθμού να παραμείνουν αμετάβλητα, η μάσκα στις υπόλοιπες θέσεις της θα έχει τον αριθμό μηδέν.

Έστω ότι έχουμε τον αριθμό 49_{16}

Δυαδική μορφή 49_{16}	0	1	0	0	1	0	0	1
Bits	B7	B6	B5	B4	B3	B2	B1	B0

και θέλουμε να τοποθετήσουμε τον αριθμό ένα στο Bit4, ενώ τα υπόλοιπα ψηφία του αριθμού θέλουμε να παραμείνουν αμετάβλητα. Σ' αυτή την περίπτωση, η μάσκα θα είναι της μορφής 00010000 και το πρόγραμμα θα περιλαμβάνει το λογικό OR μεταξύ του αριθμού και της μάσκας.

Δυαδική μορφή 49_{16}	0	1	0	**0**	1	0	0	1
Μάσκα	0	0	0	**1**	0	0	0	0
Νέος αριθμός	0	1	0	**1**	1	0	0	1
Bits	B7	B6	B5	B4	B3	B2	B1	B0

Επομένως, ο νέος αριθμός είναι ο 01011001_2 και σε δεκαεξαδική μορφή είναι ο αριθμός 59_{16}.

II. Έλεγχος της τιμής σε ένα bit:

Προκειμένου να ελέγξουμε εάν κάποιο bit ενός αριθμού είναι μηδέν ή ένα, χωρίς να μας ενδιαφέρουν οι τιμές των υπόλοιπων bits, πρέπει να κάνουμε τη λογική πράξη AND μεταξύ του αριθμού και μιας «μάσκας», η οποία σε όλες τις θέσεις της θα έχει το μηδέν εκτός από μία, τη θέση που θέλουμε να ελέγξουμε, όπου θα έχει την τιμή ένα. Εξαιτίας των μηδενικών, όλα τα bits θα γίνουν μηδέν και το αποτέλεσμα θα εξαρτάται από την τιμή του bit που

ελέγχουμε και είτε θα είναι μηδέν είτε θα είναι ένας αριθμός ο οποίος θα είναι διάφορος του μηδενός.

Έστω ότι έχουμε τον αριθμό $C2_{16}$

Δυαδική μορφή $C2_{16}$	1	1	0	0	0	0	1	0
Bits	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

και θέλουμε να ελέγξουμε αν το bit1 είναι 0 ή 1. Θα κάνουμε τη λογική πράξη AND και η μάσκα θα έχει τις τιμές 00000010.

Δυαδική μορφή $C2_{16}$	1	1	0	0	0	0	**1**	0
Μάσκα	0	0	0	0	0	0	**1**	0
Νέος αριθμός	0	0	0	0	0	0	**1**	0
Bits	B7	B6	B5	B4	B3	B2	B1	B0

Επομένως, το τελικό αποτέλεσμα είναι διάφορο του μηδενός, άρα το bit1 είναι μονάδα.

Εντολές που χρησιμοποιούνται:

Εντολή AND (AND καταχωρητής 1, καταχωρητής 2): Η εντολή AND (ΚΑΙ) πραγματοποιεί τη λογική πράξη ΚΑΙ ανάμεσα στα περιεχόμενα δύο καταχωρητών.

Σύνταξη: AND Rd, Rr

Παράδειγμα: AND R10, R20 (το αποτέλεσμα μεταφέρεται στον πρώτο καταχωρητή, δηλαδή στον R10)

Εντολή ANDI (ANDI καταχωρητής1, τιμή): Η συγκεκριμένη εντολή υλοποιεί τη λογική πράξη ΚΑΙ ανάμεσα στο περιεχόμενο ενός καταχωρητή και μιας τιμής.

Σύνταξη: ANDI Rd, K

Παράδειγμα: ANDI R25, $BB (το αποτέλεσμα μεταφέρεται στον καταχωρητή)

Εντολή OR (OR καταχωρητής 1, καταχωρητής 2): Η εντολή OR (Ή) υλοποιεί τη λογική πράξη Ή ανάμεσα στα περιεχόμενα δύο καταχωρητών.

Σύνταξη: OR Rd, Rr

Παράδειγμα: OR R1, R2 (το αποτέλεσμα μεταφέρεται στον πρώτο καταχωρητή)

Εντολή ORI (ORI καταχωρητής, τιμή): Η εντολή ORI (Ή) υλοποιεί τη λογική πράξη Ή ανάμεσα στο περιεχόμενο ενός καταχωρητή και μιας τιμής.

Σύνταξη: ORI Rd, K

Παράδειγμα: ORI R16, $10 (το αποτέλεσμα μεταφέρεται στον καταχωρητή)

Εντολή EOR (EOR καταχωρητής 1, καταχωρητής 2): Με την εντολή EOR υλοποιείται η λογική πράξη XOR ανάμεσα στο περιεχόμενο δύο καταχωρητών.

Σύνταξη: EOR Rd, Rr

Παράδειγμα: EOR R5, R15 (το αποτέλεσμα μεταφέρεται στον πρώτο καταχωρητή)

Εντολή COM (COM καταχωρητής): Η εντολή COM (One's Complement) υλοποιεί τη λογική πράξη NOT στο περιεχόμενο ενός καταχωρητή. Αποτελεί το συμπληρωματικό ως προς ένα του περιεχομένου του καταχωρητή.

Σύνταξη: COM Rd

Παράδειγμα: COM R30

Εντολή TST (TST καταχωρητής): Η συγκεκριμένη εντολή εξετάζει αν το περιεχόμενο ενός καταχωρητή είναι μηδέν ή αρνητικός αριθμός. Ουσιαστικά, υλοποιεί μία λογική πράξη ΚΑΙ ανάμεσα στον καταχωρητή και τον ίδιο.

Σύνταξη: TST Rd

Παράδειγμα: TST R18

Εντολή LSR (LSR καταχωρητής): Η εντολή LSR (Logical Shift Right – λογική μετακίνηση των bits δεξιά) προκαλεί τη μετακίνηση των bits του καταχωρητή κατά μία θέση προς τα δεξιά. Το bit7 γίνεται μηδέν και το bit0 μεταφέρεται στη σημαία κρατουμένου του καταχωρητή κατάστασης. Επίσης, η συγκεκριμένη εντολή διαιρεί έναν αριθμό, μη προσημασμένο, με το 2.

Σύνταξη: LSR Rd

Παράδειγμα: LSR R13

Εντολή LSL (LSL καταχωρητής): Η εντολή LSL (Logical Shift Left – λογική μετακίνηση των bits αριστερά) προκαλεί τη μετακίνηση των bits του καταχωρητή κατά μία θέση προς τα αριστερά. Αυτό έχει ως αποτέλεσμα το bit0 να γίνεται μηδέν και το bit7 να μεταφέρεται στη σημαία κρατουμένου του καταχωρητή κατάστασης. Επίσης, η συγκεκριμένη εντολή πολλαπλασιάζει έναν αριθμό, μη προσημασμένο, με το 2.

Σύνταξη: LSL Rd

Παράδειγμα: LSL 13

Εντολή ASR (ASR καταχωρητής): Η εντολή ASR (Arithmetic Shift Right – αριθμητική μετακίνηση των bits δεξιά) προκαλεί τη μετακίνηση των bits του καταχωρητή κατά μία θέση προς τα δεξιά. Ωστόσο, το bit7 παραμένει αμετάβλητο, ενώ το bit0 φορτώνεται στον καταχωρητή κρατουμένου. Επίσης, με το συγκεκριμένο τρόπο, πραγματοποιείται αποτελεσματικά η διαίρεση με το 2.

Σύνταξη: ASR Rd

Παράδειγμα: ASR R13

Εντολή ROL (ROL καταχωρητής): Η εντολή ROL (Rotate Left Through Carry – περιστροφή των bits αριστερά) πραγματοποιεί περιστροφή όλων των bits του καταχωρητή. Η περιστροφή υλοποιείται κατά μία θέση προς τα αριστερά. Αποτέλεσμα της συγκεκριμένης περιστροφής είναι στο bit0 να μετακινείται η σημαία κρατουμένου και στη σημαία κρατούμενου να μετακινείται το bit7.

Σύνταξη: ROL Rd

Παράδειγμα: ROL R13

Εντολή ROR (ROR καταχωρητής): Η εντολή ROR (Rotate Right Through Carry – περιστροφή των bits δεξιά) υλοποιεί περιστροφή όλων των bits του καταχωρητή. Η περιστροφή των bits γίνεται κατά μία θέση προς τα δεξιά. Αυτό έχει ως αποτέλεσμα στο bit7 να μετακινείται η σημαία κρατουμένου και στη σημαία κρατουμένου να μετακινείται το bit0.

Άσκηση 9.1: Δίνεται ο αριθμός $D2_{16}$ και είναι αποθηκευμένος στη θέση μνήμης 0140. Να γράψετε ένα πρόγραμμα το οποίο θα πραγματοποιεί μετατροπή του συγκεκριμένου αριθμού σε δυαδικό αριθμό και το κάθε ψηφίο θα τοποθετείται στις θέσεις μνήμης 0300 έως 0307. Για τη μετατροπή του αριθμού να χρησιμοποιήσετε ως μάσκα τον αριθμό 1000 0000, ο οποίος να βρίσκεται στη θέση μνήμης 0180. Επίσης, σε κάθε επανάληψη στον αριθμό της μάσκας να πραγματοποιείται ολίσθηση των bits κατά μία θέση προς τα δεξιά και σε κάθε επανάληψη να πραγματοποιείται η κατάλληλη λογική πράξη μεταξύ του αριθμού της μάσκας και του αρχικού αριθμού και ανάλογα με το αποτέλεσμα να αποθηκεύεται το ψηφίο μηδέν ή το ψηφίο ένα στην αντίστοιχη θέση μνήμης.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"
```

```
LDI R25, $D2
STS $0140, R25
LDI R20, $80        ; αριθμός μάσκας
STS $0180, R20

LDI R16, $00        ; μετρητής επαναλήψεων
LDI R17, $00        ; bit 0
LDI R18, $01        ; bit 1

LDI R31, $03
LDI R30, $00

LOOP:
LDI R25, $D2
AND R25, R20
BRNE BIT_ENA

BIT_MHDEN:
ST Z+, R17
JMP EPOMENO_BIT

BIT_ENA:
ST Z+, R18

EPOMENO_BIT:
INC R16
LSR R20
CPI R16, $08
BRNE LOOP

RET
```

Παρατηρήσεις: Στο πρόγραμμά μας χρησιμοποιούμε τη λογική πράξη AND (ΚΑΙ). Εκτελώντας το πρόγραμμα, παρατηρούμε ότι, όταν εκτελείται την πρώτη

φορά η λογική πράξη ΚΑΙ, ενημερώνονται οι σημαίες S και N. Όταν πραγματοποιείται η σύγκριση του μετρητή επαναλήψεων με το πλήθος των επαναλήψεων, ενημερώνονται οι σημαίες H, S, N, C και, όταν ο μετρητής γίνει ίσος με 8, το πρόγραμμα τερματίζει. Επίσης, μετά τη λογική πράξη ΚΑΙ, αν το bit είναι μηδέν, ενημερώνεται η σημαία Z και μεταβαίνει στο αντίστοιχο τμήμα του κώδικα, ενώ, αν είναι ένα, δεν ενημερώνεται κάποια σημαία και το πρόγραμμα μεταβαίνει στο βρόχο BIT_ENA. Τέλος, ο τελικός δυαδικός αριθμός που δημιουργείται είναι ο αριθμός 1101 0010, ο οποίος, πράγματι, αντιστοιχεί στο δεκαεξαδικό αριθμό D2, που δώσαμε στην αρχή.

Δεκαεξαδικός αριθμός	Δυαδική μορφή αριθμών							
D2	1	1	0	1	0	0	1	0
(μάσκα)	1	0	0	0	0	0	0	0
Θέση μνήμης	0300	0301	0302	0303	0304	0305	0306	0307
Τελικός αριθμός	1	1	0	1	0	0	1	0

Άσκηση 9.2: Δίνεται ο δυαδικός αριθμός 0011 1101 στις θέσεις μνήμης 0300 έως 0307. Να γράψετε ένα πρόγραμμα το οποίο να υλοποιεί τη μετατροπή του συγκεκριμένου δυαδικού αριθμού σε ακέραιο. Ο τελικός αριθμός να αποθηκευτεί στη θέση μνήμης 0400 και να αποτυπωθεί στους λαμπτήρες της πλακέτας. Επίσης, να χρησιμοποιήσετε τον αριθμό 80_{16} ως μάσκα και να βρίσκεται στη θέση μνήμης 0180. Σε κάθε επανάληψη, στον αριθμό της μάσκας να πραγματοποιείται μετακίνηση των bits κατά μία θέση προς τα δεξιά.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R17, $00
STS $0300, R17

LDI R17, $00
STS $0301, R17

LDI R17, $01
STS $0302, R17
```

```
LDI R17, $01
STS $0303, R17

LDI R17, $01
STS $0304, R17

LDI R17, $01
STS $0305, R17

LDI R17, $00
STS $0306, R17

LDI R17, $01
STS $0307, R17

LDI R25, $00
STS $0400, R25
LDI R20, $80          ; αριθμός μάσκας
STS $0180, R20

LDI R31, $03
LDI R30, $00

LOOP:
LD R21, Z
CPI R21, $00
BREQ EPOMENO
OR R25, R20           ; λογική πράξη Ή (OR)
STS $0400, R25

EPOMENO:
INC R30
LSR R20               ; μετακίνηση των bits κατά μία θέση δεξιά
```

```
CPI R30, $08
BRNE LOOP

LDI R22, $FF
OUT DDRB, R22
COM R25
OUT PORTB, R25

RET
```

Παρατηρήσεις: Η λογική πράξη που υλοποιούμε είναι η πράξη OR (Ή). Το πρόγραμμα συγκρίνει αν το περιεχόμενο των θέσεων μνήμης από 0300 έως 0307 είναι μηδέν ή ένα και ανάλογα με το αποτέλεσμα μεταφέρεται στο αντίστοιχο σημείο του κώδικα. Ακόμη, το πρόγραμμα έχει έναν καταχωρητή (R25), όπου, αν η αντίστοιχη θέση μνήμης έχει την τιμή ένα, τότε το bit του αριθμού της μάσκας προστίθεται στο συγκεκριμένο καταχωρητή. Επίσης, αν η σύγκριση του περιεχομένου των θέσεων μνήμης ισούται με μηδέν, ανανεώνεται η σημαία Z (Z = 1). Μετά την εκτέλεση της εντολής OR, ανανεώνονται οι σημαίες S, N, ενώ, μετά τη σύγκριση του μετρητή επαναλήψεων με τον αριθμό 8, οι σημαίες που ανανεώνονται είναι οι H, S, N και C και, όταν ο μετρητής επαναλήψεων γίνει ίσος με το 8, το πρόγραμμα τερματίζεται. Τέλος, ο δυαδικός αριθμός που δώσαμε αντιστοιχεί στον αριθμό $3D_{16}$.

Άσκηση 9.3: Πρόγραμμα επίδειξης πράξεων των λογικών πράξεων AND, OR, EOR, NOT, COM και ANDI.

Να γράψετε ένα πρόγραμμα το οποίο να έχει τις τιμές 09, 45, D5, BA, 12 και να είναι αποθηκευμένες στις θέσεις μνήμης 0200, 0201, 0202, 0203 και 0204 αντίστοιχα. Ανάμεσα στις δύο πρώτες τιμές να γίνεται η λογική πράξη Ή και το αποτέλεσμα της πράξης να μεταφέρεται στη θέση μνήμης 0300. Ανάμεσα στη μνήμη 0202 και 0203 να γίνεται η λογική πράξη ΚΑΙ και το αποτέλεσμα να μεταφέρεται στη μνήμη 0301. Ανάμεσα στη μνήμη 0301 και τη μνήμη 0204 να πραγματοποιείται η λογική πράξη EOR και το αποτέλεσμα να μεταφέρεται στη θέση μνήμης 0302. Μεταξύ της τιμής 12 και της τιμής FF να εφαρμόζεται η λογική πράξη ΚΑΙ και το αποτέλεσμα να μεταφέρεται στη μνήμη 0303. Στη συνέχεια, σε έναν καταχωρητή να δίνεται η τιμή CE, να υλοποιείται η λογική πράξη NOT και το αποτέλεσμα να μεταφέρεται στη θέση μνήμης 0304. Το αποτέλεσμα της μνήμης 0304 να δίνεται σε

έναν καταχωρητή, στο περιεχόμενο του συγκεκριμένου καταχωρητή να πραγματοποιείται μετακίνηση των bits κατά μία θέση προς τα δεξιά και το αποτέλεσμα να τοποθετηθεί στη μνήμη 0305. Επίσης, να δοθεί η τιμή 99, η οποία θα τοποθετείται στη θέση μνήμης 0400, όπου στο bit6, bit5 και bit2 να τοποθετείται η μονάδα, και το αποτέλεσμα να μεταφερθεί στη μνήμη 0401. Στη συνέχεια, τα bit7, bit4 και bit0 της μνήμης 0401 να μηδενίζονται και το αποτέλεσμα να μεταφερθεί στη μνήμη 0402, όπου θα υλοποιείται περιστροφή των bits κατά μία θέση προς τα αριστερά, και το νέο αποτέλεσμα να μεταφερθεί στη θέση μνήμης 0403.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"

LDI R20, $09
LDI R21, $45
LDI R22, $D5
LDI R23, $BA
LDI R24, $12

STS $0200, R20
STS $0201, R21
STS $0202, R22
STS $0203, R23
STS $0204, R24

OR R20, R21
STS $0300, R20

AND R22, R23
STS $0301, R22

LDS R25, $0301
EOR R25, R24
STS $0302, R25

ANDI R24, $FF
```

```
STS $0303, R24

LDI R26, $CE
COM R26
STS $0304, R26

LDS R27, $0304
LSR R27
STS $0305, R27

LDI R28, $99
STS $0400, R28
ORI R28, 0b01100100
STS $0401, R28

LDS R29, $0401
ANDI R29, 0b01101110
STS $0402, R29

LDS R30, $0402
ROL R30
STS $0403, R30

RET
```

Παρατηρήσεις: Όταν πραγματοποιούνται οι λογικές πράξεις OR και EOR, δεν ενημερώνεται καμία σημαία. Όταν εφαρμόζεται η λογική πράξη AND, ενημερώνονται οι σημαίες S και N. Όταν εκτελείται η λογική πράξη COM, ενημερώνεται η σημαία C. Όταν πραγματοποιείται μετακίνηση των bits προς τα δεξιά, ενημερώνονται οι σημαίες S και V. Όταν εκτελείται η εντολή ORI, ενημερώνεται η σημαία N. Επίσης, όταν πραγματοποιείται η περιστροφή των bits προς τα αριστερά, ενημερώνονται οι σημαίες H, V και N. Τέλος, όταν εφαρμοστεί η λογική πράξη ANDI, δεν ενημερώνεται καμία σημαία.

8.11 Εργαστήριο 10

Εισαγωγικά: Στο συγκεκριμένο εργαστήριο θα δούμε τον προγραμματισμό των θυρών εισόδου – εξόδου του μικροελεγκτή ATmega644P, τις εντολές in – out και τις εντολές SBI και CBI.

Θεωρία: Ένας ηλεκτρονικός υπολογιστής δεν διαθέτει μόνο επεξεργαστή και μνήμη. Διαθέτει επίσης, και θύρες εισόδου – εξόδου, που του επιτρέπουν να επικοινωνήσει και με άλλες περιφερειακές συσκευές. Η επικοινωνία των θυρών εισόδου – εξόδου γίνεται μέσω του Address Bus (δίαυλος διευθύνσεων) και μέσω του Data Bus (δίαυλος δεδομένων).

Ο μικροελεγκτής ATmega644P διαθέτει 32 ακροδέκτες θυρών εισόδου – εξόδου, από τους οποίους ο καθένας έχει τη δυνατότητα να μπορεί να μετατραπεί. Αν κάποιος ακροδέκτης είναι ακροδέκτης εξόδου, μπορεί να μετατραπεί σε ακροδέκτη εισόδου, και ένας ακροδέκτης εισόδου μπορεί να μετατραπεί σε ακροδέκτη εξόδου. Αυτό πραγματοποιείται με τις εντολές CBI και SBI αντίστοιχα. Επίσης, η κάθε θύρα διαθέτει 3 διευθύνσεις, τη διεύθυνση DDxn, τη διεύθυνση PORTxn και τη διεύθυνση PINxn. Η διεύθυνση DDxn ορίζει τους ακροδέκτες, ποιος ακροδέκτης θα είναι είσοδος και ποιος θα είναι έξοδος, η διεύθυνση PORTxn αφορά τα δεδομένα που γράφονται στους ακροδέκτες που έχουν οριστεί ως έξοδοι και η διεύθυνση PINxn αφορά εκείνα τα δεδομένα τα οποία διαβάζονται από τους ακροδέκτες εισόδου.

Εντολές που χρησιμοποιούνται:

Εντολή OUT (αριθμός θύρας, καταχωρητή): Η εντολή OUT (OUT Port) εξάγει τα περιεχόμενα ενός καταχωρητή σε μία θύρα εξόδου.

Σύνταξη: OUT P, Rr

Παράδειγμα: OUT PORTA, R20

Εντολή IN (καταχωρητής, αριθμός θύρας): Η εντολή IN (IN Port) διαβάζει τα δεδομένα που υπάρχουν στη θύρα εισόδου που έχουμε ορίσει και τα μεταφέρει στον καταχωρητή.

Σύνταξη: IN Rd, P

Παράδειγμα: IN R20, PORTA

Εντολή SBI (καταχωρητής, αριθμός ακροδέκτη): Η εντολή SBI (Set bit in I/O register) εισάγει τη μονάδα σε ένα συγκεκριμένο bit ενός καταχωρητή εισόδου-εξόδου.

Σύνταξη: SBI P, b

Παράδειγμα: SBI PORTB, PB2

Εντολή CBI (καταχωρητής, αριθμός ακροδέκτη): Η εντολή CBI (Clear bit in I/O register) θέτει μηδέν ένα συγκεκριμένο bit ενός καταχωρητή εισόδου-εξόδου.

Σύνταξη: CBI P, b

Παράδειγμα: SBI PORTB, PB2

Στη συνέχεια, θα δούμε με παραδείγματα τι συμβαίνει στην πλακέτα μας, ανάλογα με το πώς έχουν οριστεί οι ακροδέκτες μιας θύρας:

Περίπτωση 1: Ο καταχωρητής DDRxn ορίζεται ως είσοδος (έχει την τιμή 0). Σ' αυτή την περίπτωση, ό,τι τιμή και να έχει ο καταχωρητής PORTxn, δεν ανάβει κανένα led.

```
.include "C:\STUDIO6\include\m644Pdef.inc"
LDI R30, $00
OUT DDRB, R30
LDI R31, $FF
OUT PORTB, R31

RET
```

Περίπτωση 2: Ο καταχωρητής DDRxn ορίζεται ως έξοδος (έχει την τιμή 1). Σ' αυτή την περίπτωση, αν ο καταχωρητής PORTxn έχει την τιμή ένα, τα led θα αναβοσβήνουν για ένα μικρό χρονικό διάστημα και στη συνέχεια θα είναι σβηστά, αλλά το φως που θα βγάζουν θα είναι εξαιρετικά μικρό.

```
.include "C:\STUDIO6\include\m644Pdef.inc"
LDI R30, $FF
OUT DDRB, R30
LDI R31, $FF
OUT PORTB, R31

RET
```

Περίπτωση 3: Ο καταχωρητής DDRxn ορίζεται ως έξοδος και ο καταχωρητής PORTxn έχει την τιμή μηδέν. Τότε, όλοι οι λαμπτήρες θα είναι αναμμένοι (είναι η λογική active low).

```
.include "C:\STUDIO6\include\m644Pdef.inc"
LDI R30, $FF
OUT DDRB, R30
```

```
LDI R31, $00
OUT PORTB, R31

RET
```

Προκειμένου να αποτυπώσουμε στους λαμπτήρες της πλακέτας το τελικό αποτέλεσμα, είτε χρησιμοποιούμε την τεχνική συμπληρώματος ως προς ένα (την εντολή COM) είτε τη λογική πράξη EOR.

Άσκηση 10.1: Να γράψετε ένα πρόγραμμα το οποίο να προσθέτει τους αριθμούς 00 και 0F. Οι αριθμοί να βρίσκονται στις θέσεις μνήμης 0300 και 0301 αντίστοιχα, το αποτέλεσμα να τοποθετηθεί στη μνήμη 0302 και να αποτυπωθεί στους λαμπτήρες της πλακέτας.

Λύση:

```
.include "C:\STUDIO6\include\m644Pdef.inc"
LDI R20, $00
LDI R21, $0F
STS $0300, R20
STS $0301, R21

LDS R16, $0300
LDS R17, $0301
ADD R16, R17
STS $0302, R16

LDI R25, $FF
OUT DDRB, R25
EOR R16, R25
OUT PORTB, R16
```

2ος τρόπος αποτύπωσης του αποτελέσματος στους λαμπτήρες:

```
.include "C:\STUDIO6\include\m644Pdef.inc"
LDI R20, $00
LDI R21, $0F
STS $0300, R20
STS $0301, R21
```

```
LDS R16, $0300
LDS R17, $0301
ADD R16, R17
STS $0302, R16

LDI R25, $FF
OUT DDRB, R25
COM R16
OUT PORTB, R16
```

Κεφάλαιο 9

Προγραμματισμός της πλακέτας – Περιβάλλον ανάπτυξης Arduino – Γλώσσα προγραμματισμού

9.1 Προγραμματισμός

Στο συγκεκριμένο κεφάλαιο θα δούμε τον τρόπο με τον οποίο προγραμματίζεται η πλακέτα μας σε περιβάλλον ανάπτυξης Arduino. Ο Arduino είναι ένας μικροελεγκτής, που βασίζεται στην τεχνολογία ATmega328. Το περιβάλλον ανάπτυξης του Arduino είναι μία εφαρμογή και χρησιμοποιεί τη γλώσσα Wiring, η οποία αποτελεί παραλλαγή της C/C++.

Όταν ανοίγουμε το περιβάλλον ανάπτυξης του Arduino, πρέπει να φορτώσουμε τον κατάλληλο μικροεπεξεργαστή. Έχει διάφορες επιλογές: ATmega8, ATmega168, ATmega328 κ.ά. Δεν διαθέτει τον ATmega644P. Για να φορτώσουμε το δικό μας μικροεπεξεργαστή, χρησιμοποιούμε την επιλογή Sanguino. Το Sanguino, επίσης, είναι ένας μικροελεγκτής, ο οποίος βασίζεται στο μικροεπεξεργαστή ATmega644p. Προκειμένου το περιβάλλον ανάπτυξης του Arduino να αναγνωρίσει την πλακέτα μας, τη φορτώνουμε ως Sanguino. Μ’ αυτό τον τρόπο, μπορούμε να προγραμματίσουμε το εξάρτημά μας και η εικονική θύρα που αντιστοιχεί στη δική μας περίπτωση είναι η COM 3.

9.2 Arduino

Το Arduino είναι ένα εργαλείο, που χρησιμοποιείται για την κατασκευή ενός υπολογιστικού συστήματος, με την έννοια ότι αυτό θα έχει τον έλεγχο των συσκευών του φυσικού κόσμου, κάτι που δεν πραγματοποιεί ο ηλεκτρονικός υπολογιστής. Πρόκειται περί ενός ανοιχτού υλικού και λογισμικού, που βασίζεται σε μία αναπτυξιακή πλακέτα, η οποία ενσωματώνει ένα μικροελεγκτή και συνδέεται με τον ηλεκτρονικό υπολογιστή, προκειμένου να τον προγραμματίσουμε μέσα από ένα απλό περιβάλλον ανάπτυξης.

Επιπλέον, ο μικροελεγκτής Arduino μπορεί να χρησιμοποιηθεί, για να την ανάπτυξη διαδραστικών αντικειμένων, να δεχτεί εισόδους από αισθητήρια όργανα και διακόπτες, αλλά και να ελέγχει διάφορα φώτα, κινητήρες, καθώς και ποικίλες άλλες συσκευές εξόδου που αφορούν το φυσικό κόσμο. Οι εργασίες στο

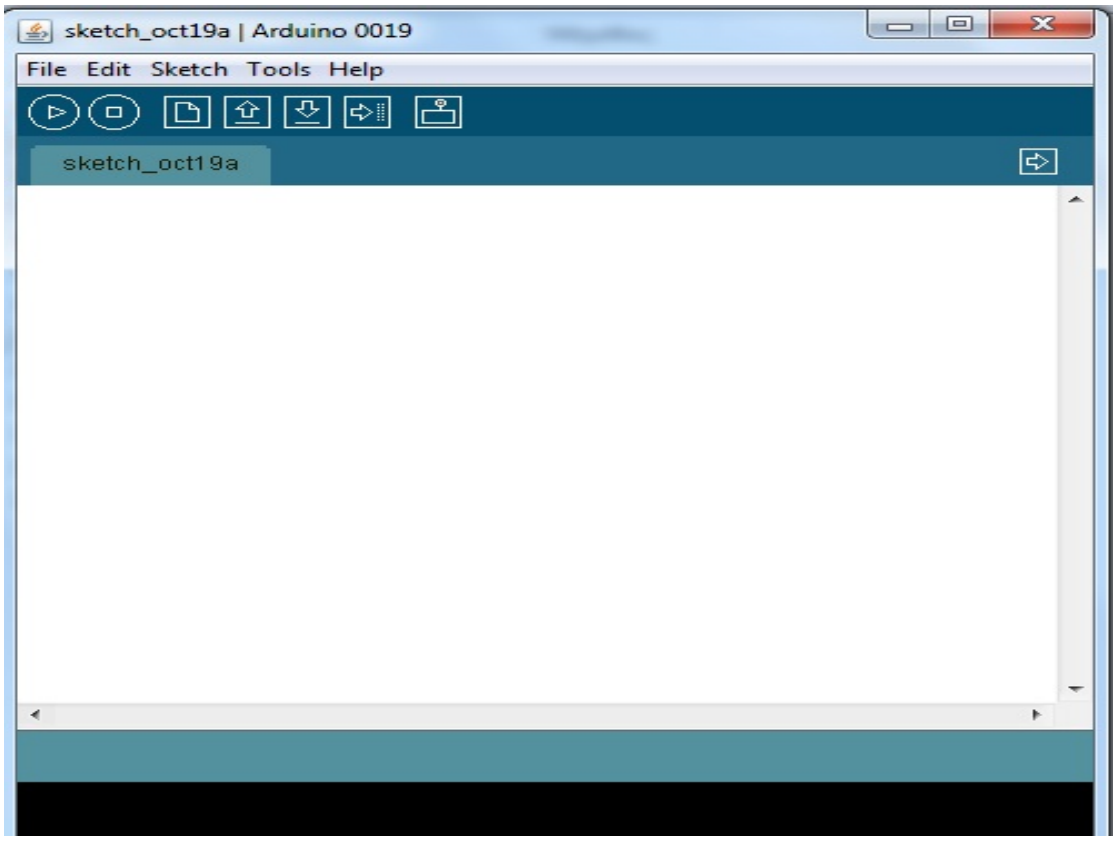
συγκεκριμένο μικροελεγκτή μπορούν να είναι ανεξάρτητες, σε επίπεδο υλικού, ή μπορούν να επικοινωνήσουν με κάποιο λογισμικό στον ηλεκτρονικό υπολογιστή του χρήστη. Η συναρμολόγηση των πλακετών μπορεί να πραγματοποιηθεί με ευκολία ακόμα και από κάποιον μη έμπειρο ή μπορούν οι πλακέτες να αγοραστούν έτοιμες και να μην χρειάζονται καμία συναρμολόγηση.

Επίσης, το περιβάλλον ανάπτυξης του λογισμικού βασίζεται σε δύο γλώσσες προγραμματισμού: στη γλώσσα Wiring και στη γλώσσα Processing. Και οι δύο γλώσσες προγραμματισμού είναι ανοιχτού κώδικα (open source) και μπορεί κάποιος να τις «κατεβάσει δωρεάν».

9.3 Περιβάλλον ανάπτυξης (IDE) Arduino

Το περιβάλλον ανάπτυξης (IDE) του Arduino είναι μία εφαρμογή, που έχει γραφεί σε Java και βασίζεται στο περιβάλλον της γλώσσας προγραμματισμού Processing (http://processing.org/).

Εικόνα 9.1: Περιβάλλον λογισμικού Arduino



Βασικές λειτουργίες του περιβάλλοντος ανάπτυξης:

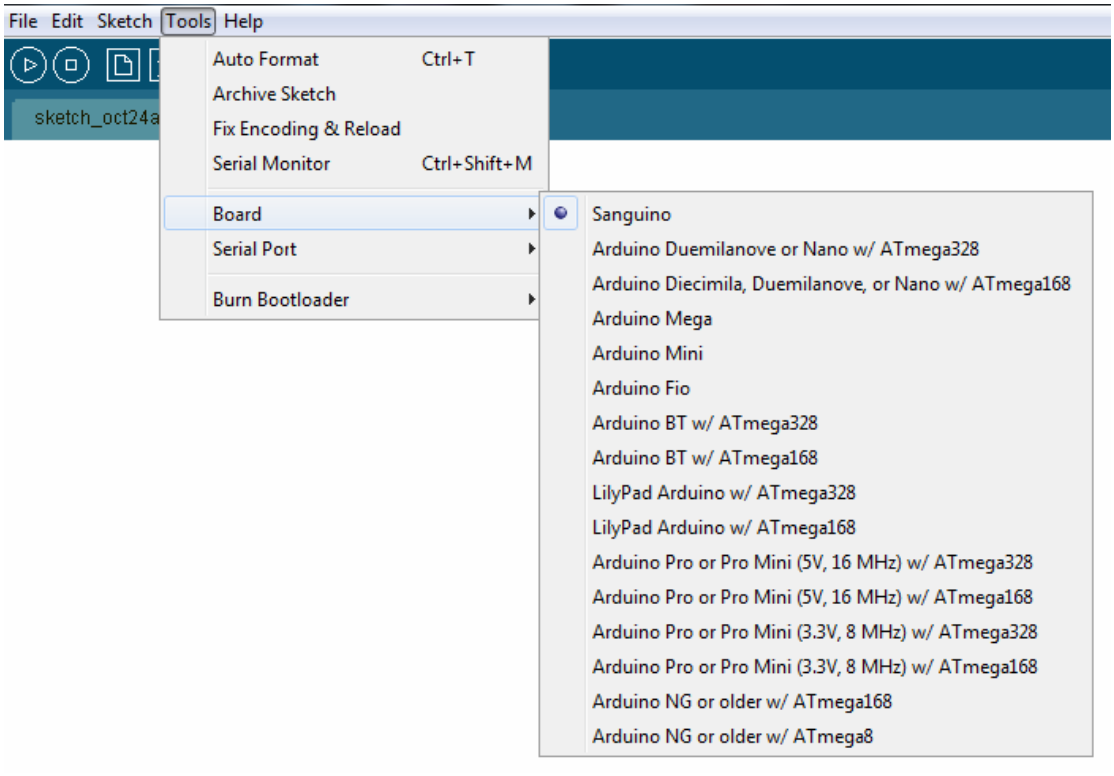
	Έλεγχος του κώδικα για λάθη.
	Τερματισμός της σειριακής κονσόλας.
	Δημιουργία νέου έργου (sketch).
	Παρουσίαση μενού με όλα τα αποθηκευμένα έργα. Πατώντας σε ένα από αυτά, ανοίγει για επεξεργασία.
	Αποθήκευση του έργου.
	Μεταγλώττιση του κώδικα και ανέβασμά του στο Arduino.
	Εμφάνιση της σειριακής κονσόλας. Αποστολή και λήψη δεδομένων που στάλθηκαν μέσω της σειριακής θύρας.

9.4 Ρυθμίσεις του περιβάλλοντος ανάπτυξης

Οι βασικές ρυθμίσεις που πρέπει να γίνουν από τη στιγμή κατά την οποία ενωθεί το Arduino (αλλά και η δική μας πλακέτα) στο σύστημά μας είναι:

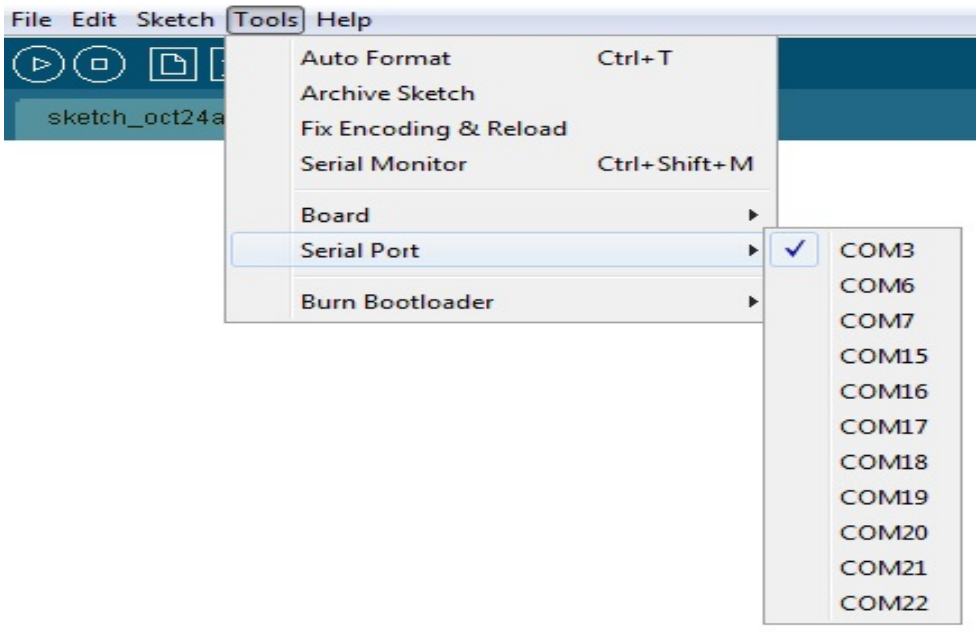
1) Επιλογή της πλακέτας. Από το μενού Tools, επιλέγουμε στη συνέχεια Board και, κατόπιν, την πλακέτα που έχουμε στη διάθεσή μας. Στη δική μας περίπτωση, επιλέγουμε Sanguino.

Εικόνα 9.2: επιλογή πλακέτας



2) Επιλογή της θύρας. Από το μενού Tools, στη συνέχεια επιλέγουμε Serial Port και, κατόπιν, τη θύρα. Στην περίπτωσή μας, επιλέγουμε την COM3.

Εικόνα 9.3: επιλογή θύρας



9.5 Γλώσσα προγραμματισμού

Η γλώσσα του Arduino βασίζεται στη γλώσσα προγραμματισμού Wiring, η οποία αποτελεί παραλλαγή της C και της C++ και αφορά μικροελεγκτές AVR, όπως

είναι ο μικροελεγκτής ATmega. Επίσης, υποστηρίζει όλες τις βασικές δομές της γλώσσας προγραμματισμού C, αλλά και ορισμένα χαρακτηριστικά της γλώσσας C++. Ο μεταγλωττιστής που χρησιμοποιεί είναι ο AVR gcc και η βιβλιοθήκη AVR libc χρησιμοποιείται ως η βασική βιβλιοθήκη C.

Επομένως, η γλώσσα του Arduino μπορεί να χρησιμοποιήσει τις ίδιες βασικές εντολές και συναρτήσεις, με την ίδια σύνταξη, τους ίδιους τύπων δεδομένων και τους ίδιους τελεστές (αριθμητικούς και σύγκρισης), όπως ακριβώς και η γλώσσα προγραμματισμού C. Επιπλέον, και οι πίνακες δηλώνονται όπως στη C. Ωστόσο, υπάρχουν και ορισμένες ειδικές εντολές, συναρτήσεις και σταθερές, οι οποίες παρέχουν σημαντική βοήθεια στη διαχείριση του ειδικού υλικού (hardware) του Arduino. Στη συνέχεια, παρατίθενται μερικές από αυτές τις εντολές:

Όρισμα	**Είδος**	**Τύπος**	**Σύνταξη**	**Περιγραφή**
LOW	σταθερά	int	-	Έχει την τιμή 0, αντίστοιχη του λογικού false.
HIGH	σταθερά	int	-	Έχει την τιμή 1, αντίστοιχη του λογικού true.
INPUT	σταθερά	int	-	Οι ακροδέκτες ορίζονται ως είσοδοι.
OUTPUT	σταθερά	int	-	Οι ακροδέκτες ορίζονται ως έξοδοι.
pinMode	εντολή	-	pinMode(pin, mode)	Ορίζει αν το συγκεκριμένο ψηφιακό pin θα είναι ακροδέκτης εισόδου ή εξόδου, ανάλογα με την τιμή που έχει δοθεί στην παράμετρο mode (input ή output αντίστοιχα).
digitalWrite	εντολή	-	digitalWrite(pin, pinstatus)	Θέτει την κατάσταση pinstatus (high ή low) στο συγκεκριμένο ψηφιακό pin.
digitalRead	συνάρτηση	int	digitalRead(pin)	Επιστρέφει την κατάσταση του συγκεκριμένου ψηφιακού pin (0 για low και 1 για high),

				εφόσον αυτό είναι pin εισόδου.
millis	συνάρτηση	unsigned long	millis()	Μετρητής που επιστρέφει το χρονικό διάστημα σε ms από την στιγμή που άρχισε η εκτέλεση του προγράμματος.
delay	εντολή	-	delay(ms)	Σταματά προσωρινά τη ροή του προγράμματος. Ο χρόνος είναι σε ms.

9.6 Δομή προγράμματος

Δομή προγράμματος Arduino:

```
// δηλώσεις μεταβλητών
void setup() {
// αρχικοποιήσεις
}
void loop() {
// ...
}
```

Παρατηρούμε ότι υπάρχουν δύο βασικές συναρτήσεις:

Η συνάρτηση setup(), η οποία εκτελείται στην αρχή του προγράμματος και η εκτέλεσή της γίνεται για μία μόνο φορά. Η συγκεκριμένη συνάρτηση χρησιμοποιείται για τις αρχικοποιήσεις των μεταβλητών, τις δηλώσεις των ακροδεκτών (αν θα είναι είσοδος ή έξοδος) και τις αρχικοποιήσεις των βιβλιοθηκών.

Στη συνάρτηση loop() ο κώδικας που έχει γραφεί μέσα σ' αυτήν τη συνάρτηση επαναλαμβάνεται διαρκώς και δίνει τη δυνατότητα στο πρόγραμμα να αλλάζει τιμές και με βάση τις αλλαγές των τιμών η πλακέτα μας να ανταποκρίνεται ανάλογα.

Κεφάλαιο 10

Σταθερές τιμές – Ψηφιακοί ακροδέκτες

10.1 False - True

False: Είναι το λογικό 0.

True: Είναι το λογικό 1.

10.2 HIGH–LOW

Όταν διαβάζουμε ή γράφουμε σε έναν ψηφιακό ακροδέκτη, υπάρχουν δύο πιθανές τιμές που μπορεί να έχει ο ακροδέκτης: HIGH και LOW.

HIGH: Η έννοια του HIGH (σε σχέση με έναν ακροδέκτη) είναι διαφορετική και εξαρτάται από το εάν ένας ακροδέκτης έχει οριστεί ως είσοδος (INPUT) ή ως έξοδος (OUTPUT). Όταν ο ακροδέκτης ορίζεται ως είσοδος με την εντολή pinMode και σε συνδυασμό με την εντολή digitalRead, ο μικροελεγκτής θα δώσει HIGH, αν στον ακροδέκτη υπάρχει τάση ίση ή και μεγαλύτερη των 3V.

Επίσης, ένας ακροδέκτης μπορεί να οριστεί ως είσοδος με την εντολή pinMode και με την εντολή digitalWrite να έχει την τιμή HIGH. Σ' αυτή την περίπτωση, ενεργοποιούνται οι εσωτερικές pull-up αντιστάσεις των 20K, οι οποίες οδηγούν τον ακροδέκτη εισόδου σε κατάσταση HIGH, εκτός κι αν έλκεται από κάποιο εξωτερικό κύκλωμα, οπότε θα έχει την τιμή LOW.

Αντίθετα, όταν ένας ακροδέκτης έχει ρυθμιστεί ως έξοδος με την εντολή pinMode και έχει τεθεί σε κατάσταση HIGH με την εντολή digitalWrite, τότε είναι στα 5V. Σ' αυτή την περίπτωση, μπορεί να δίνει ρεύμα, π.χ. να ανάβει ένα LED που συνδέεται μέσω μιας αντίστασης σε σειρά με τη γείωση ή με άλλον ακροδέκτη, ο οποίος έχει οριστεί ως έξοδος και είναι ρυθμισμένος στο LOW.

LOW: Η έννοια του LOW έχει, επίσης, διαφορετική σημασία και εξαρτάται από το αν ένας ακροδέκτης έχει οριστεί ως είσοδος (INPUT) ή ως έξοδος (OUTPUT). Όταν ένας ακροδέκτης ορίζεται ως είσοδος με την εντολή pinMode, τότε σε συνδυασμό με την εντολή digitalRead ο μικροελεγκτής θα δώσει LOW, εάν η τάση στον ακροδέκτη είναι μικρότερη ή ίση των 2V.

Επιπλέον, όταν ο ακροδέκτης ορίζεται ως έξοδος με την εντολή pinMode και έχει τεθεί σε κατάσταση LOW με την εντολή digitalWrite, τότε είναι στα 0V. Σ' αυτή

την περίπτωση, μπορεί να απάγει ρεύμα, π.χ. να ανάβει ένα LED που συνδέεται μέσω μίας αντίστασης σε σειρά με τα +5V ή με έναν άλλο ακροδέκτη, που έχει οριστεί ως έξοδος και έχει τεθεί στο HIGH.

10.3 Καθορισμός ψηφιακών ακροδεκτών, εισόδων και εξόδων

Οι ψηφιακοί ακροδέκτες μπορούν να χρησιμοποιηθούν ως είσοδοι (INPUT) ή ως έξοδοι (OUTPUT). Αλλάζοντας έναν ακροδέκτη με την εντολή pinMode(), αλλάζει η ηλεκτρική συμπεριφορά του.

10.4 Ακροδέκτες καθορισμένοι ως είσοδοι (INPUTS)

Οι ακροδέκτες του ATmega που ορίζονται ως είσοδοι (INPUT) με την εντολή pinMode() τίθενται σε κατάσταση υψηλής αντίστασης, καθιστούν εξαιρετικά μικρές τις απαιτήσεις πάνω στο κύκλωμα και ισοδυναμούν με μία αντίσταση των 100MΩ μπροστά από τον ακροδέκτη. Αυτός ο τρόπος είναι χρήσιμος για την ανάγνωση αισθητήρων, αλλά όχι για την ενεργοποίηση ενός LED.

Αν έχουμε ρυθμίσει τον ακροδέκτη μας ως είσοδο, θα θέλουμε ο ακροδέκτης να έχει μία αναφορά στη γείωση. Αυτό συχνά επιτυγχάνεται με μία pull-up αντίσταση (αντίσταση που «πηγαίνει» στη γείωση).

10.5 Ακροδέκτες καθορισμένοι ως έξοδοι (OUTPUTS)

Οι ακροδέκτες που ορίζονται ως έξοδοι (OUTPUTS) με την εντολή pinMode() τίθενται σε κατάσταση χαμηλής αντίστασης. Αυτό σημαίνει ότι μπορούν να παρέχουν αρκετό ρεύμα σε άλλα κυκλώματα. Σε άλλες συσκευές ή σε άλλα κυκλώματα, οι ακροδέκτες του ATmega μπορούν να δώσουν (source) ρεύμα ή να πάρουν (sink) ρεύμα μέχρι 40mA. Αυτός ο τρόπος είναι χρήσιμος για την ενεργοποίηση ενός LED, αλλά δεν είναι χρήσιμος για την ανάγνωση αισθητήρων. Επίσης, οι ακροδέκτες που έχουν οριστεί ως έξοδοι μπορούν να καταστραφούν, αν συνδεθούν στη γείωση ή στην τάση. Ωστόσο, το παραπάνω ρεύμα δεν είναι αρκετό για την τροφοδοσία ρελαί ή μοτέρ και γι' αυτό το λόγο στις συγκεκριμένες περιπτώσεις χρειάζονται ενδιάμεσα κυκλώματα.

Κεφάλαιο 11

Υλοποίηση παραδείγματος

11.1 Στόχος – υλοποίηση παραδείγματος

Με βάση τα μέσα που διαθέτουμε, στόχος του παραδείγματός μας είναι να προγραμματίσουμε την πλακέτα μας και να αναδείξουμε μερικές από τις δυνατότητες που αυτή έχει. Για τον προγραμματισμό της πλακέτας χρησιμοποιήσαμε έναν ηλεκτρονικό υπολογιστή, το περιβάλλον ανάπτυξης του Arduino, τα τέσσερα από τα οκτώ λαμπάκια που διαθέτει η πλακέτα μας, ένα από τα πέντε κουμπιά πίεσης που αυτή διαθέτει, την οθόνη LCD, εννιά καλώδια για τη σύνδεση των ακροδεκτών εισόδου με τους ακροδέκτες της οθόνης και με τους ακροδέκτες των λαμπτήρων, καθώς και ένα καλώδιο για τη σύνδεση ενός κουμπιού πίεσης με ένα λαμπτήρα.

11.2 Περιγραφή παραδείγματος

Στο παράδειγμά μας, όταν συνδέουμε την πλακέτα και φορτώνουμε τον κώδικα, στο περιβάλλον ανάπτυξης αυτό που συμβαίνει είναι να ανάβει το πρώτο το λαμπάκι, ενώ τα άλλα δύο να παραμένουν σβηστά. Στη συνέχεια, ανάβει το δεύτερο λαμπάκι και κατόπιν ανάβει και το τρίτο. Μετά την πάροδο ενός συγκεκριμένου χρόνου, ανάβουν ξανά το πρώτο και το δεύτερο λαμπάκι, με αποτέλεσμα για ένα ορισμένο χρονικό διάστημα να είναι αναμμένα συγχρόνως και τα τρία λαμπάκια.

Ακολούθως, το δεύτερο και το τρίτο λαμπάκι σβήνουν και παραμένει αναμμένο μόνο το πρώτο. Όταν σβήνουν τα δύο λαμπάκια, στην οθόνη εμφανίζεται ο χρόνος που δείχνει πόση ώρα είναι σε λειτουργία η πλακέτα μας. Επίσης, την πρώτη φορά που θα σβήσουν το δεύτερο και το τρίτο λαμπάκι, στην οθόνη θα εμφανιστούν και δύο μηνύματα. Παράλληλα, έχουμε συνδέσει ένα κουμπί πίεσης με ένα τέταρτο λαμπάκι.

11.3 Επεξήγηση των τιμών HIGH και LOW

Όπως θα δούμε παρακάτω στον κώδικα, όταν έχουμε την τιμή HIGH στην εντολή digitalWrite(), το λαμπάκι είναι σβηστό. Όταν, όμως, στην εντολή digitalWrite() έχουμε την τιμή LOW, το λαμπάκι ανάβει. Αυτό συμβαίνει, διότι η ηλεκτρική συμπεριφορά των ακροδεκτών του ATmega αλλάζει και εξαρτάται από το

πώς έχει οριστεί ένας ακροδέκτης (είσοδος ή έξοδος) και από το ποια εντολή χρησιμοποιείται [digitalRead() ή digitalWrite()][4].

11.4 Κώδικας

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(16, 17, 18, 19, 24, 25);
int ledPin1 =  5;
int ledPin2 =  10;
int ledPin3 =  11;
int ledPin4;
int buttonPin;
int buttonMode = 0;

void setup()   {

lcd.begin(16, 2);
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
pinMode(ledPin4, OUTPUT);
pinMode(buttonPin, INPUT);
}

void loop()
{
        lcd.setCursor(0, 1);
        lcd.print(millis()/1000);
        digitalWrite(ledPin2, HIGH);
        digitalWrite(ledPin3, HIGH);
        digitalWrite(ledPin1, LOW);
        delay(20000);
```

[4] Περισσότερες πληροφορίες σχετικά με τη συμπεριφορά των ακροδεκτών μπορούν να αναζητηθούν στο κεφ. 10, σελ. 142.

```
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(7000);
digitalWrite(ledPin2, LOW);
delay(15000);
digitalWrite(ledPin2, HIGH);
digitalWrite(ledPin3, HIGH);
delay(3000);
digitalWrite(ledPin3, LOW);
delay(5000);
digitalWrite(ledPin1, LOW);
delay(5000);
digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, LOW);
delay(5000);
lcd.setCursor(0,0);
lcd.print("Ptyxiakh Ergasia");
lcd.setCursor(5, 1);
lcd.print("Nick-Bill");
buttonMode = digitalRead(buttonPin);
if (buttonMode == HIGH) {
        digitalWrite(ledPin4, HIGH);
}
        else {
                digitalWrite(ledPin4, LOW);
                }

}
```

11.5 Επεξήγηση του κώδικα

#include <LiquidCrystal.h> // περιλαμβάνει τον κώδικα της βιβλιοθήκης LiquidCrystal.h, η οποία βιβλιοθήκη είναι έτοιμη από το περιβάλλον ανάπτυξης του Arduino

LiquidCrystal lcd(16, 17, 18, 19, 24, 25); // προετοιμάζει τη βιβλιοθήκη με τους αριθμούς από τους ακροδέκτες διασύνδεσης

int ledPin1 = 5; // το πρώτο λαμπάκι (ledPin1) συνδέεται με τον ψηφιακό ακροδέκτη 5

int ledPin2 = 10; // το δεύτερο λαμπάκι (ledPin2) συνδέεται με τον ψηφιακό ακροδέκτη 10

int ledPin3 = 11; // το τρίτο λαμπάκι (ledPin3) συνδέεται με τον ψηφιακό ακροδέκτη 11

int ledPin4; // ορίζουμε το τέταρτο λαμπάκι (ledPin4)

int buttonPin; // ορίζουμε ένα κουμπί πίεσης

int buttonMode = 0; // ορίζουμε τη μεταβλητή για την ανάγνωση της κατάστασης του κουμπιού

// η συνάρτηση setup() τρέχει μία φορά, όταν φορτωθεί το πρόγραμμα

void setup() {

lcd.begin(16, 2); // θέτουμε τον αριθμό των στηλών και των σειρών της LCD οθόνης (16 στήλες, 2 σειρές)

pinMode(ledPin1, OUTPUT); // ορίζουμε το ledPin1 ως έξοδο

pinMode(ledPin2, OUTPUT); // ορίζουμε το ledPin2 ως έξοδο

pinMode(ledPin3, OUTPUT); // ορίζουμε το ledPin3 ως έξοδο

pinMode(ledPin4, OUTPUT); // ορίζουμε το ledPin4 ως έξοδο

pinMode(buttonPin, INPUT); // ορίζουμε το κουμπί πίεσης ως είσοδο

}

// η συνάρτηση loop() τρέχει συνεχώς, όσο η πλακέτα μας είναι συνδεδεμένη

void loop()

{

lcd.setCursor(0, 1); // θέτουμε τον κέρσορα στην πρώτη στήλη και στη δεύτερη γραμμή (σημείωση: η γραμμή 1 είναι η δεύτερη γραμμή, καθώς η καταμέτρηση αρχίζει από το μηδέν)

lcd.print(millis()/1000); // εκτυπώνει το χρόνο (σε δευτερόλεπτα) λειτουργίας της πλακέτας

digitalWrite(ledPin2, HIGH); // θέτουμε το LED2 σε κατάσταση off

digitalWrite(ledPin3, HIGH); // θέτουμε το LED3 σε κατάσταση off

digitalWrite(ledPin1, LOW); // θέτουμε το LED1 σε κατάσταση on

delay(20000); // ανάβει για 20 δευτερόλεπτα

digitalWrite(ledPin1, HIGH); // θέτουμε το LED1 σε κατάσταση off

digitalWrite(ledPin2, HIGH); // το LED2 παραμένει σε κατάσταση off

delay(7000); // για 7 δευτερόλεπτα

digitalWrite(ledPin2, LOW); // θέτουμε το LED2 σε κατάσταση on

delay(15000); // παραμένει αναμμένο για 15 δευτερόλεπτα

digitalWrite(ledPin2, HIGH); // θέτουμε το LED2 σε κατάσταση off

digitalWrite(ledPin3, HIGH); // το LED3 παραμένει σε κατάσταση off

delay(3000); // για 3 δευτερόλεπτα

digitalWrite(ledPin3, LOW); // θέτουμε το LED3 σε κατάσταση on

delay(5000); // για 5 δευτερόλεπτα είναι αναμμένο μόνο του και μετά από 5 δευτερόλεπτα ανάβει το LED1

digitalWrite(ledPin1, LOW); // θέτουμε το LED1 σε κατάσταση on

delay(5000); // για 5 δευτερόλεπτα είναι αναμμένο μαζί με το LED3 και μετά από 5 δευτερόλεπτα ανάβει το LED2

digitalWrite(ledPin2, LOW); // θέτουμε το LED2 σε κατάσταση on

digitalWrite(ledPin3, LOW); // το LED3 παραμένει σε κατάσταση on

delay(5000); // για 5 δευτερόλεπτα είναι αναμμένα και τα 3 λαμπάκια μαζί, στη συνέχεια το LED2 και το LED3 σβήνουν και παραμένει αναμμένο το LED1

lcd.setCursor(0,0); // θέτουμε τον κέρσορα στην πρώτη στήλη και στην πρώτη γραμμή της LCD οθόνης

lcd.print("Ptyxiakh Ergasia"); // στη θέση που έχουμε θέσει τον κέρσορα προηγουμένως εμφανίζεται στην οθόνη το μήνυμα «Ptyxiakh Ergasia»

lcd.setCursor(5, 1); // θέτουμε τον κέρσορα στην πέμπτη στήλη και στη δεύτερη γραμμή της LCD οθόνης

lcd.print("Nick-Bill"); // στη θέση που έχουμε θέσει τον κέρσορα προηγουμένως εμφανίζεται στην οθόνη το μήνυμα «Nick-Bill»

buttonMode = digitalRead(buttonPin); // διαβάζει την κατάσταση του κουμπιού πίεσης

if (buttonMode == HIGH) // Ελέγχουμε αν το κουμπί είναι πατημένο. Αν είναι, η κατάστασή του είναι HIGH.

{

digitalWrite(ledPin4, HIGH); // ανάβει το τέταρτο λαμπάκι ledPin4

}

else {

digitalWrite(ledPin4, LOW); // διαφορετικά, αν δεν είναι πατημένο, είναι LOW και το λαμπάκι είναι σβηστό

}

}

11.6 Σχόλια – Συμπεράσματα

Αρχικά, ανάβει το πρώτο το λαμπάκι για είκοσι δευτερόλεπτα, ενώ τα άλλα δύο παραμένουν σβηστά. Αφού σβήσει το λαμπάκι μετά από επτά δευτερόλεπτα, ανάβει το δεύτερο για χρονικό διάστημα δεκαπέντε δευτερολέπτων. Στη συνέχεια, σβήνει το δεύτερο και μετά από τρία δευτερόλεπτα ανάβει το τρίτο. Το τρίτο λαμπάκι παραμένει αναμμένο (μόνο του) για πέντε δευτερόλεπτα. Μετά την πάροδο πέντε δευτερολέπτων, ανάβει το πρώτο λαμπάκι (το τρίτο συνεχίζει να είναι αναμμένο). Για πέντε δευτερόλεπτα είναι αναμμένα το πρώτο και το τρίτο. Μετά από πέντε δευτερόλεπτα, ανάβει και το δεύτερο (το πρώτο και το τρίτο συνεχίζουν να είναι αναμμένα). Αφού ανάψει και το δεύτερο, για πέντε δευτερόλεπτα είναι αναμμένα συγχρόνως και τα τρία λαμπάκια.

Στη συνέχεια, και μετά από πέντε δευτερόλεπτα, το δεύτερο και το τρίτο λαμπάκι σβήνουν και παραμένει αναμμένο το πρώτο για είκοσι δευτερόλεπτα και συνεχίζεται η ίδια διαδικασία. Όταν, όμως, σβήσουν τα δύο λαμπάκια, στη δεύτερη γραμμή της οθόνης θα εμφανιστεί ο χρόνος (σε δευτερόλεπτα) που δείχνει πόση ώρα είναι σε λειτουργία η πλακέτα μας. Κάθε φορά που θα σβήνουν τα δύο λαμπάκια, αμέσως μετά τη στιγμή που θα είναι και τα τρία ταυτόχρονα αναμμένα, ο χρόνος θα ανανεώνεται.

Επίσης, την πρώτη φορά που θα σβήσουν το δεύτερο και το τρίτο λαμπάκι, στην οθόνη θα εμφανιστούν και δύο μηνύματα. Το πρώτο μήνυμα θα εμφανιστεί στην πρώτη γραμμή της οθόνης και θα γράφει «Ptyxiakh Ergasia»[5] και το δεύτερο μήνυμα θα εμφανιστεί στη δεύτερη γραμμή της οθόνης και θα γράφει «Nick-Bill»[6].

Τέλος, το τέταρτο λαμπάκι, που έχουμε συνδέσει με το κουμπί πίεσης, ανάβει, όποτε πατάμε το κουμπί. Όταν το κουμπί δεν είναι πατημένο, το λαμπάκι παραμένει σβηστό.

[5] Το μήνυμα θα είναι γραμμένο με αγγλικούς χαρακτήρες, διότι η πλακέτα μας δεν μπορεί να διαβάσει ελληνικούς.
[6] Το μήνυμα θα είναι γραμμένο με αγγλικούς χαρακτήρες, διότι η πλακέτα μας δεν μπορεί να διαβάσει ελληνικούς.

Επίλογος

Τα αποτελέσματα που προέκυψαν από τις εργαστηριακές ασκήσεις στους λαμπτήρες, καθώς και το παράδειγμα που δείξαμε στο περιβάλλον ανάπτυξης Arduino αποτελούν ένα μικρό μέρος των δυνατοτήτων που περιλαμβάνουν η πλακέτα μας και ο μικροελεγκτής ATmega644P. Η πλακέτα EvB 4.3 v4 μπορεί να χρησιμοποιηθεί για τη μέτρηση της θερμοκρασίας του περιβάλλοντος, η οποία θερμοκρασία θα εμφανίζεται στην οθόνη, την εμφάνιση της ώρας και την ημερομηνίας και την παραγωγή ηχητικών σημάτων μέσω του βομβητή. Ακόμη, με τη χρήση του υπέρυθρου ανιχνευτή σήματος και κατάλληλου τηλεχειριστηρίου έχει τη δυνατότητα να παράγει σήματα, για παράδειγμα την επίτευξη αλλαγής σταθμών σε ένα ραδιόφωνο και την αλλαγή καναλιών σε μία τηλεόραση. Επίσης, μπορεί να απαριθμεί πόσες φορές πατιέται ένα κουμπί, όταν ανάβει ένας λαμπτήρας.

Όσον αφορά το μικροελεγκτή AVR ATmega644P και γενικά όλους τους μικροελεγκτές, αυτοί έχουν πολύ μεγάλες δυνατότητες. Ειδικότερα, αυτό ισχύει, όταν συνδυάζονται και με επιπλέον μέσα, όπως είναι περισσότερα καλώδια για τη σύνδεση των ακροδεκτών της πλακέτας, το ράστερ, οι αντιστάσεις, οι δίοδοι και οι αισθητήρες.

Επιπλέον, οι μικροελεγκτές μπορούν να χρησιμοποιηθούν και σε άλλες περιπτώσεις. Τέτοιες είναι η κατασκευή φωτοβολταϊκού ηλιοστάτη, η μέτρηση της θερμοκρασίας ενός κινητήρα αυτοκινήτου, η κατασκευή και ο έλεγχος ενός ρομπότ, η κατασκευή και ο έλεγχος ενός δικτύου δεξαμενών κ.ά.

Παράρτημα

Εντολές ATmega644P

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ←PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ←Z	None	2
JMP	k	Direct Jump	PC ←k	None	3
RCALL	k	Relative Subroutine Call	PC ←PC + k + 1	None	4
ICALL		Indirect Call to (Z)	PC ←Z	None	4
CALL	k	Direct Subroutine Call	PC ←k	None	5
RET		Subroutine Return	PC ←STACK	None	5
RETI		Interrupt Return	PC ←STACK	I	5
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ←PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd –Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd –Rr –C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd –K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ←PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ←PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ←PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ←PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ←PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ←PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ←PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ←PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ←PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ←PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ←PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ←PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ←PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ←PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ←PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ←PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ←PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ←PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ←PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ←PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ←PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ←PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ←1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ←0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ←Rd(n), Rd(0) ←0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ←Rd(n+1), Rd(7) ←0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)←Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)←Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ←Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ←1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ←0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ←Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ←T	None	1
SEC		Set Carry	C ←1	C	1
CLC		Clear Carry	C ←0	C	1
SEN		Set Negative Flag	N ←1	N	1
CLN		Clear Negative Flag	N ←0	N	1
SEZ		Set Zero Flag	Z ←1	Z	1
CLZ		Clear Zero Flag	Z ←0	Z	1
SEI		Global Interrupt Enable	I ←1	I	1
CLI		Global Interrupt Disable	I ←0	I	1
SES		Set Signed Test Flag	S ←1	S	1
CLS		Clear Signed Test Flag	S ←0	S	1
SEV		Set Twos Complement Overflow.	V ←1	V	1
CLV		Clear Twos Complement Overflow	V ←0	V	1
SET		Set T in SREG	T ←1	T	1
CLT		Clear T in SREG	T ←0	T	1
SEH		Set Half Carry Flag in SREG	H ←1	H	1
CLH		Clear Half Carry Flag in SREG	H ←0	H	1

DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ←Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ←Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ←K	None	1
LD	Rd, X	Load Indirect	Rd ←(X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ←(X), X ←X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ←X - 1, Rd ←(X)	None	2
LD	Rd, Y	Load Indirect	Rd ←(Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ←(Y), Y ←Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ←Y - 1, Rd ←(Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ←(Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ←(Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ←(Z), Z ←Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ←Z - 1, Rd ←(Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ←(Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ←(k)	None	2
ST	X, Rr	Store Indirect	(X) ←Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ←Rr, X ←X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ←X - 1, (X) ←Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ←Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ←Rr, Y ←Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ←Y - 1, (Y) ←Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ←Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ←Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ←Rr, Z ←Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ←Z - 1, (Z) ←Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ←Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ←Rr	None	2
LPM		Load Program Memory	R0 ←(Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ←(Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ←(Z), Z ←Z+1	None	3
SPM		Store Program Memory	(Z) ←R1:R0	None	-
IN	Rd, P	In Port	Rd ←P	None	1
OUT	P, Rr	Out Port	P ←Rr	None	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
PUSH	Rr	Push Register on Stack	STACK ←Rr	None	2
POP	Rd	Pop Register from Stack	Rd ←STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

ΠΑΡΑΓΓΕΛΙΑ ΕΞΑΡΤΗΜΑΤΟΣ – ΒΙΒΛΙΟΓΡΑΦΙΑ – ΙΣΤΟΓΡΑΦΙΑ:

Η αγορά του εξαρτήματος έγινε από την ιστοσελίδα: http://www.acdcshop.gr/developmentkitavrusb-p-10301.html (online Αγορά) Evaluation Kit EvB 4.3 v4.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Καζαρλής Σπύρος, *Σημειώσεις Εργαστηρίου Αρχιτεκτονικής Υπολογιστών, Εκπαιδευτικό Σύστημα BGG-8088, Επίσημες Σημειώσεις για το Εργαστηριακό μάθημα «Αρχιτεκτονική Η/Υ» του Δ Εξαμήνου του Τμήματος Πληροφορικής & Επικοινωνιών της Σχολής Σ.Τ.Ε.Φ. του Τ.Ε.Ι. Σερρών*, Τ.Ε.Ι. Σερρών, Σεπτέμβριος 2004.

2. Καλόμοιρος Ιωάννης, *Σημειώσεις του μαθήματος «Προηγμένα Ψηφιακά Συστήματα»*, Α.Τ.Ε.Ι. Σερρών.

ΙΣΤΟΓΡΑΦΙΑ

1) http://el.wikipedia.org/wiki/Μικροελεγκτής
2) Πορλιδάς Δημήτριος, «Μικροελεγκτές AVR: Τα πρώτα βήματα», στο: http://www.porlidas.gr/Micro/AVR.htm
3) http://www.microplanet.gr/tutorials/microcontrollers/atmel/atmel_avr
4) http://www.eln.teilam.gr/sites/default/files/Lesson03.pdf
5) http://www.rlx.sk/pdf/EvB43v4_manual.pdf (πρόκειται για το manual του εξαρτήματος)
6) http://www.atmel.com/devices/atmega644p.aspx?tab=parameters
7) http://www.atmel.com/images/doc7674.pdf (datasheet ATmega644P)
8) http://www.atmel.com/images/doc0856.pdf (AVR Instruction Set)
9) el.wikipedia.org/wiki/Μνήμη_τυχαίας_προσπέλασης
10) http://and-tech.pl/wp-content/download/zestaw%20evb%204.3/bootloader_evb_4.3_eng.pdf
11) http://www.atmel.com/System/BaseForm.aspx?target=tcm:26-49768

12) http://www.elec.uow.edu.au/avr/guides/Getting-started-C-programming-Atmel-Studio-6.pdf
13) http://www.pololu.com/docs/0J36/3.b
14) http://www.avr-asm-tutorial.net/avr_en/
15) http://www.avr-asm-tutorial.net/avr_en/beginner/index.html
16) http://www.avrbeginners.net/
17) Χατζηκυριάκου Γιώργος, «Μάθετε το Arduino, Μέρος πρώτο, Εισαγωγή», στο: t-h.wikispaces.com/file/view/ΕισαγωγήστοArduino.pdf
18) http://deltahacker.gr/2009/08/01/arduino-intro/
19) http://and-tech.pl/
20) http://www.ftdichip.com/Drivers/VCP.htm
21) www.and-tech.pl/files/EvBISP.zip
22) http://and-tech.pl/files/bootloader/bootloader_m644p.zip
23) http://arduino.cc/en/Reference/HomePage
24) http://arduino.cc/en/Reference/Constants
25) http://arduino.cc/en/Main/Software
26) https://sites.google.com/site/funlw65/evb-projects/projects-using-arduino-language
27) http://www.hlektronika.gr/