ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΟΣ 2: Μικροελεγκτές AVR

Κ. ΠΕΚΜΕΣΤΖΗ
ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.

ΑΘΗΝΑ 2012

ΠΕΡΙΕΧΟΜΕΝΑ

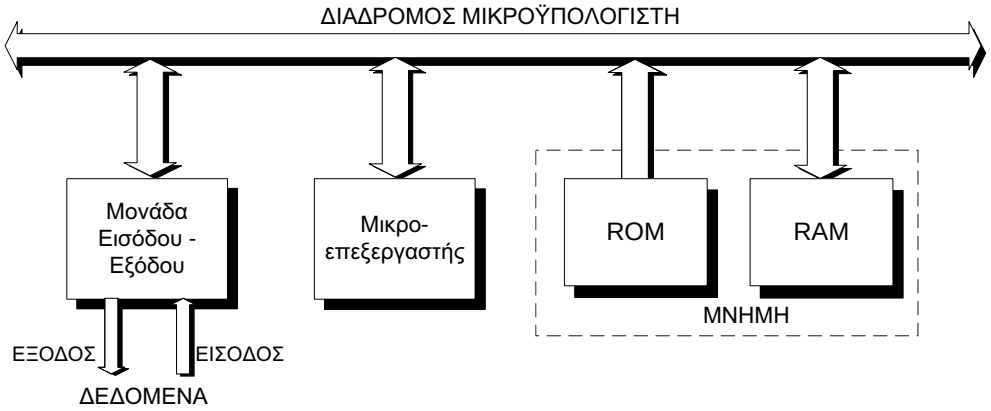
1

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ ΜΙΚΡΟΕΛΕΓΚΤΩΝ AVR

1.1 Εισαγωγή στους Μικροελεγκτές και Μικροεπεξεργαστές

Συστήματα Μικροϋπολογιστών - Μικροεπεξεργαστές

Οι Μικροϋπολογιστές είναι το προϊόν της τεχνολογίας VLSI. Το 1971 η τεχνολογία αυτή που αφορά στην κατασκευή ολοκληρωμένων κυκλωμάτων επέτρεψε την υλοποίηση μεγάλου αριθμού λογικών πυλών ή τρανζίστορ (ξεκίνησε από ~1.000 τρανζίστορ) σε μια επίπεδη ψηφίδα πυριτίου επιφάνειας της τάξεως του mm^2 (περίπου όσο η κεφαλή μιας καρφίτσας). Άμεση ήταν η προσπάθεια αξιοποίησής της για την υλοποίηση τμημάτων ενός υπολογιστή όπως μιας απλής Κεντρικής Μονάδας Επεξεργασίας (Κ.Μ.Ε), της απαραίτητης μνήμης και μονάδων Εισόδου/Εξόδου δεδομένων σε ξεχωριστές ψηφίδες πυριτίου. Η Κ.Μ.Ε ονομάστηκε Μικροεπεξεργαστής (Microprocessor) και το σύστημα, που συντέθηκε με βάση τις ψηφίδες αυτές Μικροϋπολογιστής (ή Μικροϋπολογιστικό Σύστημα) λόγω του μεγέθους και της απλότητάς του.



Σχήμα 1.1 Το βασικό διάγραμμα ενός Μικροϋπολογιστικού Συστήματος

Η απλότητά τους σχετιζόταν με το μικρό ρεπερτόριο εντολών (τάξεως μερικών δεκάδων) και επίσης με το μικρό μήκος λέξης δεδομένων (4-8 δυαδικών ψηφίων) που μπορούσαν να επεξεργαστούν ταυτόχρονα. Από τότε όμως πέρασε πολύς καιρός και το μόνο που έμεινε μικρό ήταν το φυσικό τους μέγεθος. Η βελτίωση της τεχνολογίας επέτρεψε τη γρήγορη αύξηση της εσωτερικής πολυπλοκότητάς τους. Έτσι όχι μόνο αυξήθηκε το ρεπερτόριο εντολών (τάξη μερικών εκατοντάδων) και το μήκος των δεδομένων και των διευθύνσεων (32-64 δυαδικών ψηφίων) αλλά υλοποιήθηκαν και προχωρημένες αρχιτεκτονικές, που υπήρχαν παλιότερα μόνο σε υπολογιστές υψηλών επιδόσεων (τεχνικές παραλληλίας, διοχέτευση εντολών κλπ.). Η πολυπλοκότητά τους τείνει προς το 1 δισεκατομμύριο τρανζίστορ. Ταυτόχρονα έχουμε και μια πολύ μεγάλη αύξηση στην ταχύτητα λειτουργίας τους (μεγαλύτερη των 3GHz). Πολύ μεγάλη αύξηση έχουμε όμως και στη χωρητικότητα των μνημών. Επίσης και οι περιφερειακές μονάδες των Μικροϋπολογιστών ακολούθησαν σε αύξηση πολυπλοκότητας και ταχύτητας.

Η εκρηκτική αυτή ανάπτυξη δημιούργησε μια μεγάλη ποικιλία διαφορετικών τύπων Μικροϋπολογιστών. Το ερώτημα βέβαια που προκύπτει είναι γιατί να κατασκευάσουμε χιλιάδες διαφορετικούς τύπους Μικροϋπολογιστών - Μικροεπεξεργαστών αλλά το κυριότερο εκατοντάδες εκατομμύρια αντίτυπά τους. Η απάντηση είναι απλή. Χρειάζονται σχεδόν σε κάθε ηλεκτρονική συσκευή. Μπορούμε να διακρίνουμε τρεις κατηγορίες Μικροεπεξεργαστών. Αυτούς που χρησιμοποιούνται σε υπολογιστές γενικής χρήσης, στη βιομηχανία και στην επεξεργασία σήματος.

Οι Μικροεπεξεργαστές που χρησιμοποιούνται στους προσωπικούς υπολογιστές για εφαρμογές γραφείου, όπως επεξεργασία κειμένου, ή για εφαρμογές που απαιτούν αριθμητικές πράξεις έχουν προχωρημένα αρχιτεκτονικά χαρακτηριστικά. Στους Μικροεπεξεργαστές αυτούς η επιδίωξη είναι να έχουμε υψηλές επιδόσεις σε ένα ευρύ φάσμα εφαρμογών (Pentium, Επεξεργαστές RISC κλπ.).

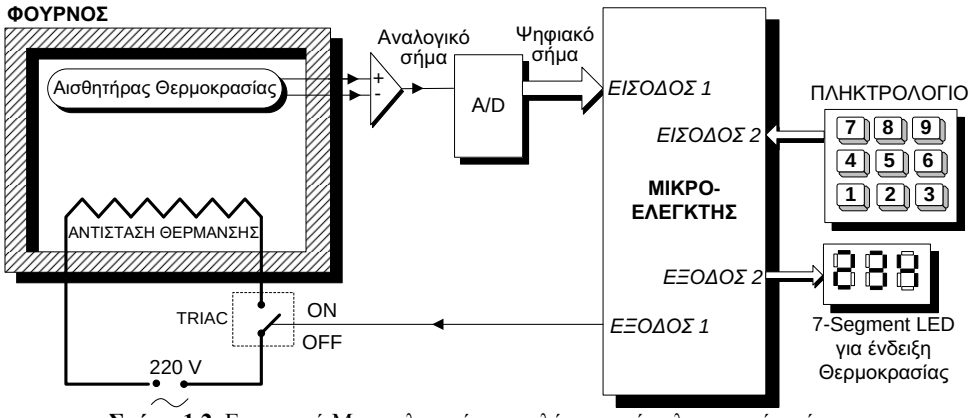
Μικροελεγκτές

Η δεύτερη κατηγορία Μικροεπεξεργαστών βρίσκει εφαρμογή στις ηλεκτρονικές συσκευές, στις τηλεπικοινωνίες και στη βιομηχανία. Κάθε συσκευή, μηχάνημα ή ηλεκτρονικό σύστημα εκτός από τον έλεγχο της σωστής και αποδοτικής λειτουργίας του χρειάζεται επίσης χειρισμό, ρύθμιση και παρακολούθηση. Το ζητούμενο στις εφαρμογές αυτές είναι να βρίσκονται στην ίδια ψηφίδα με το Μικροεπεξεργαστή και οι περιφερειακές μονάδες. Τα ολοκληρωμένα κυκλώματα που περιλαμβάνουν στην ίδια ψηφίδα εκτός από το Μικροεπεξεργαστή και θύρες για είσοδο - έξοδο δεδομένων (σε παράλληλη και σειριακή μορφή), μετατροπείς αναλογικών σημάτων σε ψηφιακή μορφή και αντίστροφα (A/D και D/A), χρονιστές και μνήμες (ROM και RAM) ονομάζονται Μικροελεγκτές (Microcontrollers). Ο Μικροελεγκτής είναι υπεύθυνος για την είσοδο και έξοδο, την επεξεργασία, αποθήκευση και μετάδοση των αναλογικών και ψηφιακών

σημάτων μιας εφαρμογής. Πιο συγκεκριμένα η χρήση ενός Μικροελεγκτή στον έλεγχο, για παράδειγμα ενός ηλεκτρικού φούρνου, φαίνεται στο Σχήμα 1.2. Στο σύστημα αυτό ο Μικροελεγκτής λαμβάνει από το πληκτρολόγιο την επιθυμητή θερμοκρασία, το χρόνο παραμονής σε αυτήν και την εντολή εκκίνησης.

Στη συνέχεια με τη χρήση της ΕΞΟΔΟΥ 1 ανοίγει την τροφοδοσία και ταυτόχρονα μέσω ενός αισθητήρα και του ενσωματωμένου μετατροπέα A/D που διαθέτει (ΕΙΣΟΔΟΣ 1) παρακολουθεί τη θερμοκρασία. Κάθε φορά που φτάνει στην επιθυμητή τιμή διακόπτει την τροφοδοσία της αντίστασης θέρμανσης (με τη χρήση ενός ηλεκτρονικού διακόπτη ισχύος π.χ. TRIAC) και όταν είναι κάτω από την τιμή αυτή να την ανοίγει. Επίσης, κάθε δευτερόλεπτο ανανεώνει την ένδειξη της θερμοκρασίας τα τρία 7-segment LED που διαθέτει. Υπάρχει μια πληθώρα από τέτοιου είδους εφαρμογές (μεγαλύτερης ή και μικρότερης πολυπλοκότητας) που έχουν σημαντικό πρακτικό ενδιαφέρον.

Είναι φανερό ότι οι μικροελεγκτές (microcontrollers) και οι μικρο-επεξεργαστές (microprocessors) αποτελούν απαραίτητο μέρος των σύγχρονων τεχνολογικών εφαρμογών.



Σχήμα 1.2 Εφαρμογή Μικροελεγκτή στον ελέγχου ενός ηλεκτρικού φούρνου

Ωστόσο, οι δύο αυτές συσκευές διαφέρουν μεταξύ τους σε αρκετά σημεία. Μια βασική τους διαφορά, και η πιο σημαντική, βρίσκεται στη λειτουργικότητά τους. Προκειμένου να λειτουργήσει ένας μικροεπεξεργαστής, θα πρέπει να συνδεθεί και με άλλες συσκευές, όπως η μνήμη (memory) ή μια συσκευή αποστολής και λήψης δεδομένων. Αυτό σημαίνει ότι ένας μικροεπεξεργαστής είναι η καρδιά του συστήματος. Αντίθετα, ένας μικροελεγκτής σχεδιάζεται με τέτοιο τρόπο ώστε να περιέχει όλες τις παραπάνω συσκευές. Συνεπώς, δε χρειάζονται άλλες συσκευές για τη λειτουργία του, εφόσον όλα τα απαραίτητα περιφερειακά (peripherals) είναι ενσωματωμένα στη δομή του. Με τον τρόπο αυτό, εξοικονομούμε χώρο και χρόνο, κατά την κατασκευή ενός συστήματος που βασίζεται σε μικροελεγκτή.

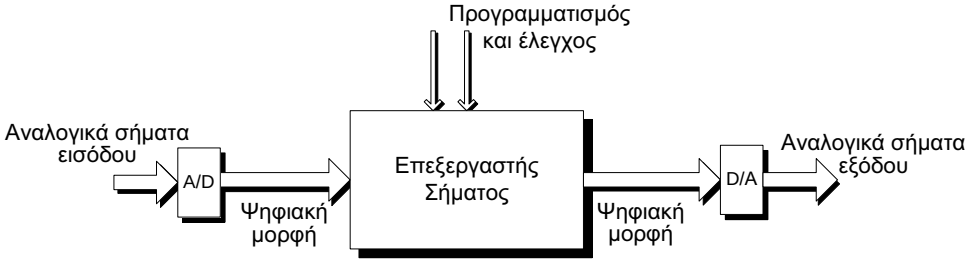
Επεξεργαστές Σήματος

Πέρα όμως από τις εφαρμογές ελέγχου και αυτοματισμών, οι Μικροϋπολογιστές χρησιμοποιούνται ευρύτατα στο χειρισμό και στην επεξεργασία σημάτων. Στη σημερινή ψηφιακή εποχή, κάθε είδος σήματος όπως, ήχος, εικόνα κλπ. δειγματοληπτείται και η αντίστοιχη στιγμιαία τιμή του μετατρέπεται σε ένα δυαδικό αριθμό. Στη νέα του αυτή μορφή το λεγόμενο ψηφιακό σήμα προσφέρει πολλά πλεονεκτήματα. Τα πλεονεκτήματα της ψηφιακής τεχνολογίας στην επεξεργασία, αποθήκευση και μετάδοση της πληροφορίας φαίνονται στον πίνακα 1.1.

Πίνακας 1.1 Πλεονεκτήματα της ψηφιακής μορφής

Επεξεργασία	Αποθήκευση	Μετάδοση
▪ Ευελιξία ▪ Υλοποίηση αλγορίθμων με λογισμικό	▪ Μεγάλη χωρητικότητα ▪ Ακρίβεια ▪ Ελαχιστοποίηση σφαλμάτων ▪ Ευκολία και ταχύτητα ενταμίευσης/ανάκτησης	▪ Υψηλές ταχύτητες ▪ Ευκολία και Ευελιξία ▪ Υλοποίηση με λογισμικό πρωτοκόλλων, δρομολόγησης και ελέγχου σφαλμάτων ▪ Ακρίβεια ▪ Ελαχιστοποίηση σφαλμάτων

Χωρίς τη χρήση των Μικροεπεξεργαστών δε θα ήταν τεχνολογικά και οικονομικά εφικτή η υλοποίηση της ψηφιακής τεχνολογίας.



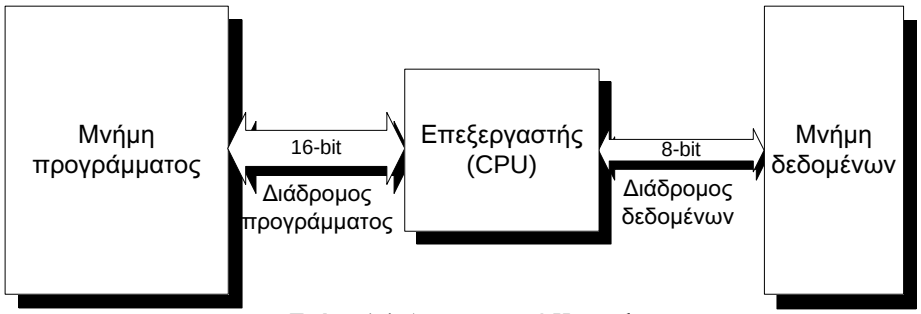
Σχήμα 1.3 Η αρχή λειτουργίας των Συστημάτων Επεξεργασίας Σήματος

Ακόμα και στις επικοινωνίες, όπου απαιτείται ηλεκτρονική επεξεργασία σε πολύ υψηλές ταχύτητες (μικροκύματα, ραδιοσυχνότητες, ραδιόφωνο, τηλεόραση, κινητές επικοινωνίες) προβλέπεται σύντομα τα αντίστοιχα συστήματα να υλοποιούνται με τη χρήση νέας γενιάς γρήγορων A/D και D/A μετατροπέων και γρήγορων Μικροεπεξεργαστών.

Στις εφαρμογές αυτές, η έμφαση είναι στην ταχύτητα και στην βέλτιστη υλοποίηση ειδικών αλγορίθμων όπως ψηφιακών φίλτρων, του γρήγορου μετασχηματισμού Fourier (FFT) κλπ. Οι Μικροεπεξεργαστές που είναι εφοδιασμένοι με αυτές τις ιδιότητες ονομάζονται Επεξεργαστές Σήματος (Signal Processors). Στο σχήμα 1.3 φαίνεται η αρχή λειτουργίας των συστημάτων Επεξεργασίας Σήματος.

1.2 Εισαγωγή στους Μικροελεγκτές AVR

Σκοπός του κεφαλαίου είναι η περιγραφή της αρχιτεκτονικής και των εντολών των μικροελεγκτών AVR. Ο μικροελεγκτής AVR περιέχει έναν πυρήνα (KME) επεξεργαστή RISC (**R**educed **I**nstruction **S**et **C**omputer) ο οποίος έχει σχεδιαστεί με βάση την αρχιτεκτονική Harvard. Όπως φαίνεται από το σχήμα, στην αρχιτεκτονική Harvard υπάρχει διαφορετικός διάδρομος για τη μεταφορά δεδομένων (data bus) και διαφορετικός διάδρομος για τη μεταφορά των εντολών (instruction bus). Η ύπαρξη δύο διαφορετικών μνημών, μνήμη δεδομένων (data memory) και μνήμη προγράμματος (program memory), καθιστά την αρχιτεκτονική Harvard αποδοτική, αφού μπορεί να εκτελείται κάποια εντολή και παράλληλα να εγγράφεται ή να διαβάζεται η μνήμη. Με τον τρόπο αυτό, επιτυγχάνεται η εκτέλεση των περισσότερων εντολών σε ένα μόνο κύκλο μηχανής.



Σχήμα 1.4 Αρχιτεκτονική Harvard

Τα γενικά χαρακτηριστικά των μικροελεγκτών AVR είναι τα παρακάτω:

α. Οι μικροελεγκτές διαθέτουν ενσωματωμένη μνήμη flash με δυνατότητα προγραμματισμού εντός του συστήματος – ISP (**I**n **S**ystem **P**rogrammable)- ως μνήμη προγράμματος. Δηλαδή δεν είναι απαραίτητες εξωτερικές μνήμες τύπου ROM ή EPROM που να περιέχουν τον κώδικα του προγράμματος και δε χρειάζεται για τον προγραμματισμό τους να μεταφερθούν σε κάποια εξωτερική συσκευή.

β. Οι μικροελεγκτές διαθέτουν 32 καταχωρητές εργασίας των 8 bit. Συνεπώς υπάρχουν αρκετές θέσεις ώστε, οι διάφορες μεταβλητές να μπορούν να αποθηκεύονται εντός της KME αντί των θέσεων μνήμης, όπου η διαδικασία πρόσβασης απαιτεί περισσότερο χρόνο.

γ. Οι μικροελεγκτές διαθέτουν ενσωματωμένη μνήμη δεδομένων (data memory) τύπου EEPROM και SRAM στα περισσότερα μοντέλα της σειράς. Οι μνήμες αυτές χρησιμοποιούνται για την αποθήκευση σταθερών τιμών και μεταβλητών.

δ. Διαθέτουν ενσωματωμένο προγραμματιζόμενο χρονιστή με μονάδα διαίρεσης συχνότητας (prescaler). Η μονάδα αυτή χρησιμοποιείται σε εφαρμογές όπου η ακρίβεια του χρονισμού είναι κρίσιμη.

ε. Προγραμματιζόμενο χρονιστή επιτήρησης (Watch Dog Timer-WDT). Ο συνήθης ρόλος του είναι η αποφυγή εσφαλμένων λειτουργιών σε περίπτωση κατάρρευσης του προγράμματος.

στ. Έως τέσσερις παράλληλες θύρες (των 8-bit) και πληθώρα περιφερειακών, όπως UART, SPI, αναλογικός συγκριτής, μετατροπέας ADC, κλπ. Για την υποστήριξη των θυρών εισόδου-εξόδου διαθέτει έως και 64 καταχωρητές (δεδομένων, ελέγχου και ρύθμισης λειτουργίας).

ζ. Δυνατότητα για εσωτερικές και εξωτερικές διακοπές.

η. Λειτουργία σε συχνότητες χρονισμού από 0 έως 10MHz. Καθώς οι περισσότερες εντολές ολοκληρώνονται σε μια περίοδο του κεντρικού σήματος χρονισμού (το ρολόι της ΚΜΕ), η επίδοση είναι ανώτερη σε σύγκριση με τους κλασσικούς μικροελεγκτές που λειτουργούν στην ίδια συχνότητα χρονισμού αλλά απαιτούν για την εκτέλεση μιας εντολής περισσότερους κύκλους ρολογιού.

θ. Διάφορα μοντέλα διαθέτουν ενσωματωμένο ταλαντωτή τύπου RC για επιλογή της συχνότητας χωρίς τη χρήση εξωτερικού κρυστάλλου.

ι. Εσωτερικό κύκλωμα που επανατοποθετεί το σύστημα κατά την εφαρμογή της τάσης τροφοδοσίας (Power On Reset-POR).

κ. Λειτουργίες αποκοπής (Power down) και ηρεμίας (Sleep). Ο επεξεργαστής τίθεται στις καταστάσεις αυτές όταν δεν υπάρχει δραστηριότητα, ούτως ώστε να μειωθεί η κατανάλωση ισχύος.

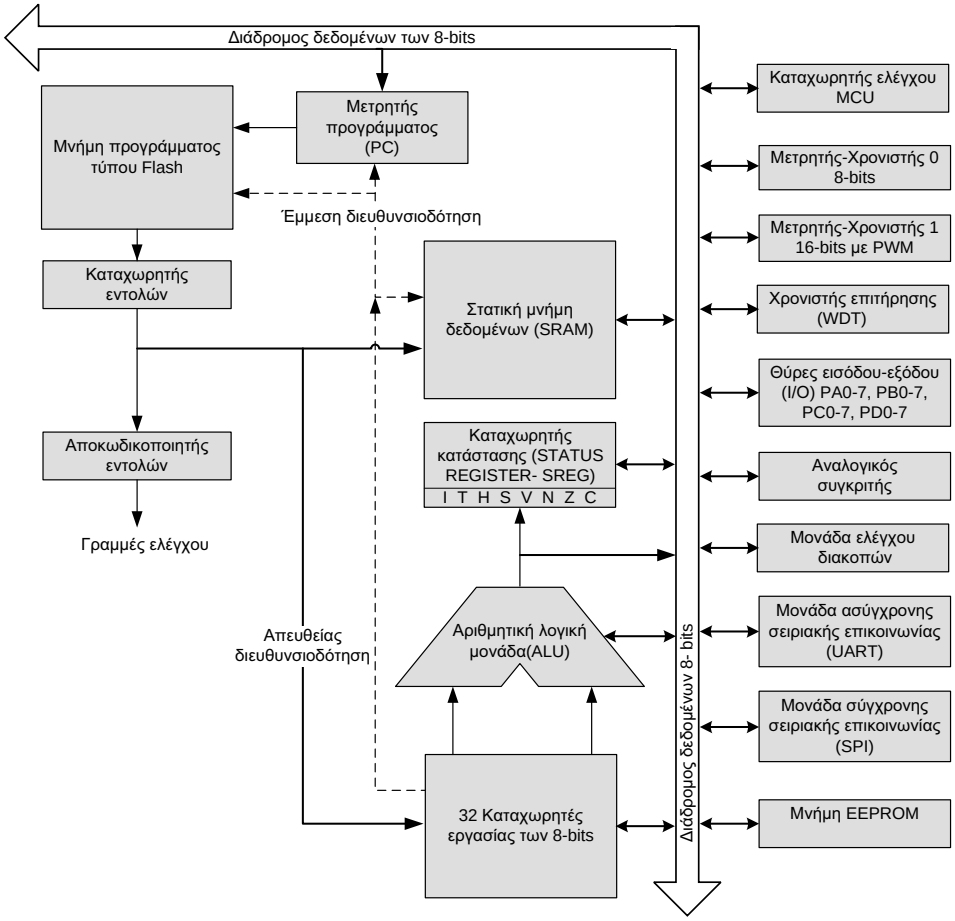
Στον Πίνακα 1.2, παρουσιάζονται διάφοροι τύποι μικροελεγκτών και τα χαρακτηριστικά τους.

Πίνακας 1.2 Διάφοροι τύποι μικροελεγκτών AVR

Μοντέλο	Ακροδέκτες	Flash	EEPROM	RAM	UART	ADC
90S1200	20	1K	64 bytes	0	Όχι	Όχι
90S2313	20	2K	128	128	Ναι	Όχι
90S2323	8	2K	128	128	Όχι	Όχι
90S2333	28	2K	128	128	Ναι	Ναι
90S4433	28	4K	256	128	Ναι	Ναι
90S4414	40	4K	256	256	Ναι	Όχι
90S8515	40	8K	512	512	Ναι	Όχι
90S4434	40	4K	256	256	Ναι	Ναι
90S2343	8	2K	128	128	Όχι	Όχι
Mega103	64	128K	4096	4096	Ναι	Ναι
Mega603	64	64K	2048	4096	Ναι	Ναι
Mega16	**40**	**16K**	**512**	**1024**	**Ναι**	**Ναι**
Tiny10	8	1K	64	0	Όχι	Όχι
Tiny12	8	1K	64	0	Όχι	Όχι
Tiny13	8	2K	128	128	Όχι	Όχι

Στην παρουσίαση αυτή θα επικεντρωθούμε στο μικροελεγκτή ATMega16, που μολονότι διαθέτει όλα τα παραπάνω χαρακτηριστικά, εντούτοις παραμένει ένας σχετικά προσιτός, από πλευράς κόστους, μικροελεγκτής, και για το λόγο αυτό αποτελεί μια συνήθη επιλογή σε μεσαίας πολυπλοκότητας κατασκευές.

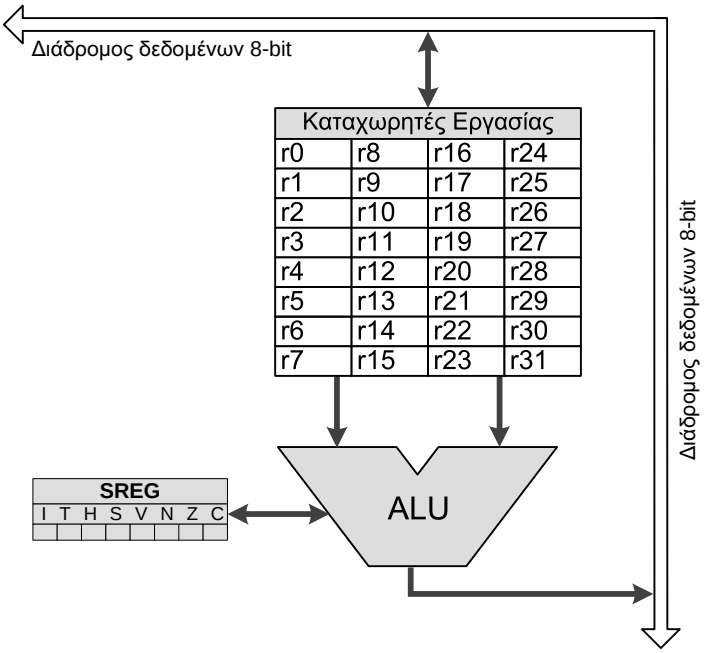
Το σημαντικό σε όλους αυτούς τους τύπους μικροελεγκτών AVR είναι ότι ο πυρήνας της αρχιτεκτονικής τους είναι ίδιος και έχουν το ίδιο σύνολο εντολών. Διαφέρουν μόνο στα περιφερειακά και στο μέγεθος της μνήμης που παρέχουν. Συνεπώς, το λογισμικό είναι μεταφέρσιμο μεταξύ των διαφορετικών τύπων. Φυσικά, αν έχουμε ένα τύπο μικροελεγκτή που διαθέτει περισσότερα περιφερειακά, θα απαιτηθεί η αντίστοιχη προσθήκη κώδικα, που αφορά στο χειρισμό τους.
Η αρχιτεκτονική των επεξεργαστών AVR, με τις διάφορες λειτουργικές μονάδες τους, φαίνεται στο σχήμα 1.5.



Σχήμα 1.5 Αρχιτεκτονική των Μικροελεγκτών AVR και τα περιφερειακά τους.

Αριθμητική λογική μονάδα (ALU)

Είναι η μονάδα αριθμητικών και λογικών πράξεων του μικροελεγκτή. Αυτή εκτελεί αριθμητικές, λογικές και σε επίπεδο bit εντολές μεταξύ Καταχωρητών Εργασίας ή και Καταχωρητή Εργασίας με σταθερά και αποθηκεύει το αποτέλεσμα στον καταχωρητή εργασίας που δείχνει η εντολή στη διάρκεια μιας περιόδου.
Κατά την εκτέλεση των πράξεων ενημερώνονται οι σημαίες του καταχωρητή κατάστασης (Status register). Όπως φαίνεται στο σχήμα 1.6 η ALU συνδέεται απευθείας με όλους τους καταχωρητές εργασίας, τον καταχωρητή κατάστασης και το διάδρομο δεδομένων:



Σχήμα 1.6 Αριθμητική λογική μονάδα (ALU)

Καταχωρητές εργασίας

Οι μικροελεγκτές διαθέτουν 32 καταχωρητές εργασίας των 8-bit και συμβολίζονται R0-R31. Οι περισσότερες εντολές που χρησιμοποιούν τους καταχωρητές εργασίας (πλην των καταχωρητών δείκτη) ολοκληρώνονται σε μια περίοδο, εκτός από τις εντολές όπως ADIW, SBIW, MUL, FMUL και LD, ST, LDD, STD, LPM, SPM. Αυτές οι τελευταίες εντολές χρησιμοποιούν μόνο τους R26-R31 που είναι καταχωρητές δείκτες. Επίσης οι εντολές ANDI, ORI, LDI χρησιμοποιούν μόνο τους R16-R31.

Καταχωρητές δείκτη

Τα ζεύγη καταχωρητών R27:R26, R29:R28 και R31:R30 ή αντίστοιχα X,Y,Z χρησιμοποιούνται ως 16-bit καταχωρητές δείκτη προς θέσεις μνήμης SRAM ή προς θέσεις μνήμης προγράμματος (μονάχα ο Z).

Καταχωρητής κατάστασης (Status register)

Ο καταχωρητής κατάστασης είναι 8-bits, τα οποία αντιστοιχούν στις σημαίες του συστήματος. Οι σημαίες δίνουν πληροφορίες για την κατάσταση του συστήματος και μηδενίζονται κατά την εκκίνηση ή επανατοποθέτηση του συστήματος (reset). Η διεύθυνση στη μνήμη εισόδου-εξόδου είναι $3F.

Καταχωρητής κατάστασης (Status Register - SREG)

Bit	7	6	5	4	3	2	1	0
Σημαία	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**

Η σημασία των σημαιών είναι η εξής:

- Bit7 (I): καθολική ενεργοποίηση διακοπών (Global Interrupt Enable-GIE). Θέτοντας το bit αυτό ενεργοποιούμε τις διακοπές.
- Bit6 (T): σημαία αντιγραφής-αποθήκευσης (bit Copy Storage). Με χρήση των εντολών BLD και BST επιτυγχάνουμε την ανάγνωση και αποθήκευση συγκεκριμένων bits.
- Bit5 (H): σημαία δεκαδικού κρατουμένου (Half Carry flag). Ενημερώνει για την ύπαρξη δεκαδικού κρατουμένου μετά από αριθμητικές πράξεις.
- Bit4 (S): σημαία προσήμου (Sign flag). Ενημερώνει για το πρόσημο ενός καταχωρητή και ισούται με το λογικό OR των σημαιών αρνητικού προσήμου και υπερχείλισης.(S=N or V)
- Bit3 (V): σημαία υπερχείλισης (Overflow flag) για αριθμητική συμπληρώματος του δύο.
- Bit2 (N): σημαία αρνητικού προσήμου (negative flag).
- Bit1 (Z): σημαία μηδενισμού (Zero flag). Τίθεται όταν το αποτέλεσμα μιας πράξης είναι μηδέν.
- Bit0 (C): σημαία κρατουμένου (Carry flag). Ενημερώνει για την ύπαρξη ή όχι κρατουμένου μετά από αριθμητικές ή λογικές πράξεις.

Οι σημαίες τίθενται ή μηδενίζονται μέσω αριθμητικών, λογικών, σύγκρισης, επιπέδου bit, ολίσθησης και άλλων εντολών.

1.3 Μνήμη δεδομένων και μνήμη προγράμματος

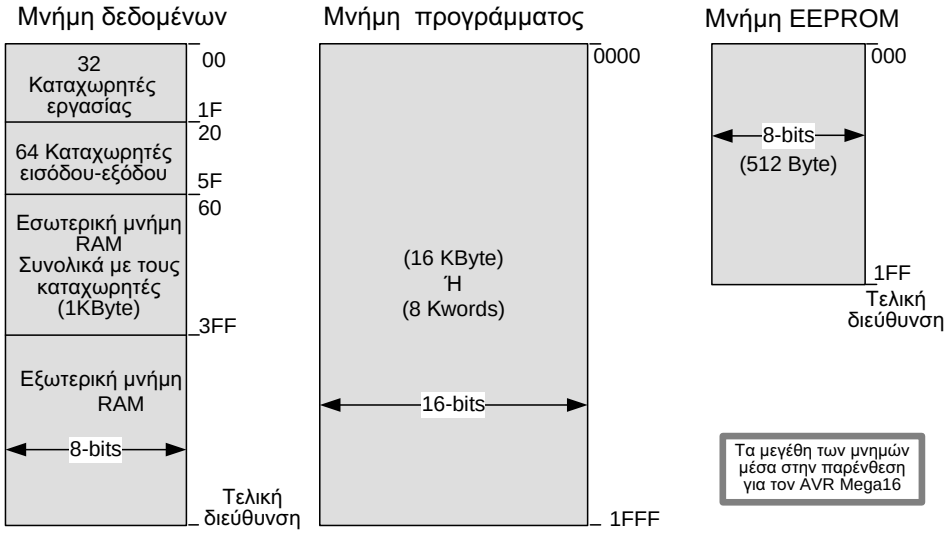
Όπως φαίνεται στο σχήμα 1.7, ένας μικροελεγκτής διαθέτει τα εξής είδη μνήμης:

- μνήμη δεδομένων
- μνήμη προγράμματος τύπου flash
- μνήμη EEPROM με αρχική διεύθυνση $0000 και χωρητικότητα από 64bytes ως 4Kbytes.

Η **μνήμη προγράμματος** είναι τύπου flash, επιτυγχάνει ταχεία αποθήκευση-φόρτωση δεδομένων, και συνδέεται με διάδρομο των 16-bit για την τροφοδοσία του καταχωρητή εντολών. Στη μνήμη προγράμματος αποθηκεύονται οι εντολές και τα διανύσματα διακοπών (interrupt vectors). Όλες οι εντολές έχουν μήκος 16 bits (δηλ. 2 bytes=1 λέξη) ή 32 bits, άρα καταλαμβάνουν μία ή δύο θέσεις της μνήμης προγράμματος, αφού η μνήμη flash είναι οργανωμένη σε διάταξη 8Kx16. Ο μετρητής προγράμματος του ATmega16 έχει μήκος 13 bits ώστε να επιτυγχάνει πρόσβαση σε 8K θέσεις μνήμης προγράμματος.

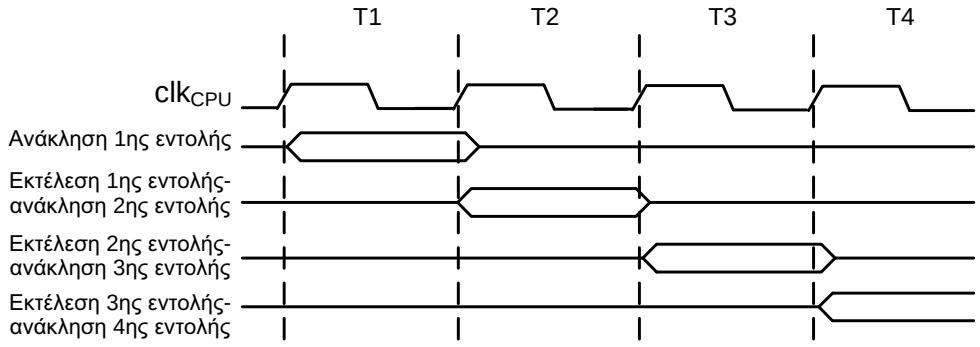
Η πρόσβαση στη μνήμη προγράμματος γίνεται σε μία περίοδο του σήματος χρονισμού. Η εκτέλεση των περισσοτέρων εντολών γίνεται σε μία περίοδο του κεντρικού ρολογιού, γεγονός που οφείλεται στη συνεχή διοχέτευση εντολών (instruction **pipeline**).

Δηλαδή, κατά την εκτέλεση μιας εντολής γίνεται ανάκληση από τη μνήμη της επόμενης εντολής. Ανάκληση και εκτέλεση πραγματοποιούνται ταυτόχρονα. Το σχήμα 1.8 απεικονίζει την παράλληλη διαδικασία ανάκλησης (από τη μνήμη προγράμματος) και εκτέλεσης εντολών (γίνεται με τη χρήση της μνήμης δεδομένων) που καθίσταται δυνατή χάρη στην αρχιτεκτονική Harvard και το ταχείας πρόσβασης Register File.

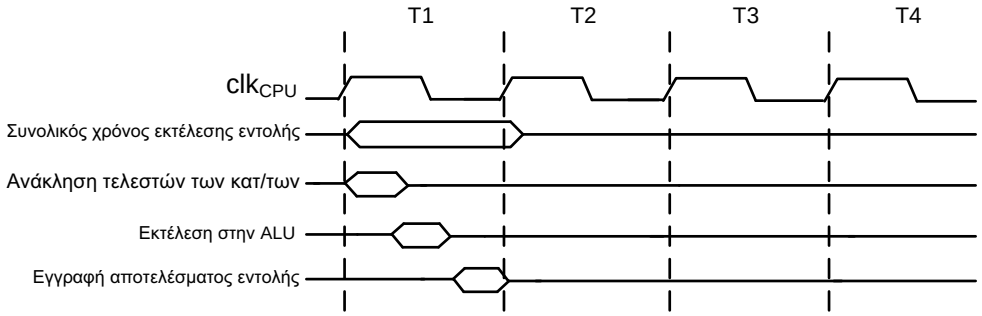


Σχήμα 1.7 Τύποι μνημών μικροελεγκτών AVR

Έπειτα, αφού η εντολή ανακληθεί από τη μνήμη φορτώνεται στον καταχωρητή εντολών. Αυτός με τη σειρά του συνδέεται με τον αποκωδικοποιητή εντολών, όπου παράγονται τα κατάλληλα σήματα ελέγχου για την εκτέλεση της εντολής με τη βοήθεια της αριθμητικής λογικής μονάδας (ALU). Τα τρία βήματα που απαιτούνται: ανάκληση δεδομένων από τους καταχωρητές, εκτέλεση στην ALU, αποθήκευση του αποτελέσματος στους καταχωρητές γίνονται σε μια περίοδο του ρολογιού. Το σχήμα 1.9 απεικονίζει τη διαδικασία, που ως επί το πλείστον απαιτεί μια περίοδο του σήματος χρονισμού.
Ο επεξεργαστής ΚΜΕ καθοδηγείται από το σήμα ρολογιού clk_{CPU}, που παράγεται από την επιθυμητή (κρύσταλλο ή RC) πηγή χρονισμού του chip. Δε χρησιμοποιείται εσωτερική διαίρεση του σήματος ρολογιού.

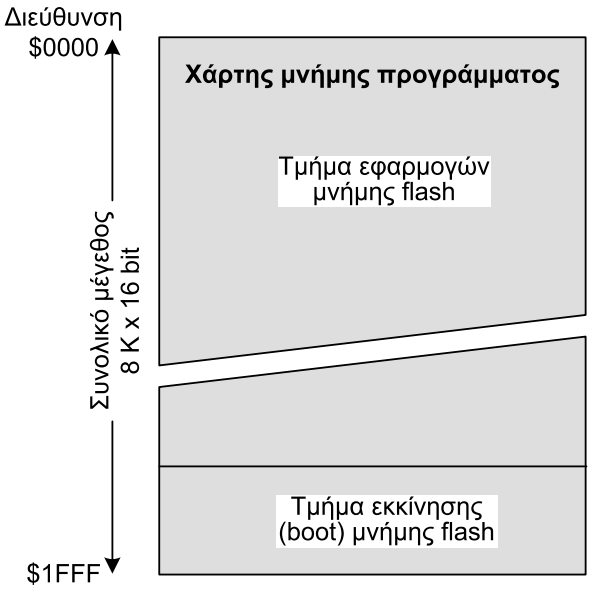


Σχήμα 1.8 Συνεχής διοχέτευση εντολών (Παράλληλη ανάκληση και εκτέλεση εντολών)



Σχήμα 1.9 Βήματα διαδικασίας εκτέλεσης εντολής στην ALU

Η διάρκεια της μνήμης είναι τουλάχιστον 10000 κύκλοι εγγραφής / διαγραφής. Για προστασία του λογισμικού, η μνήμη χωρίζεται σε δύο τμήματα, το τμήμα εκκίνησης (Boot Program section) και το τμήμα εφαρμογών (Application Program section). Τα δύο τμήματα έχουν bits κλειδώματος για ασφάλεια εγγραφής και ανάγνωσης/εγγραφής. (Η εντολή **SPM** που γράφει στο τμήμα εφαρμογών πρέπει να βρίσκεται εντός του τμήματος εκκίνησης).



Σχήμα 1.10 Χάρτης μνήμης προγράμματος

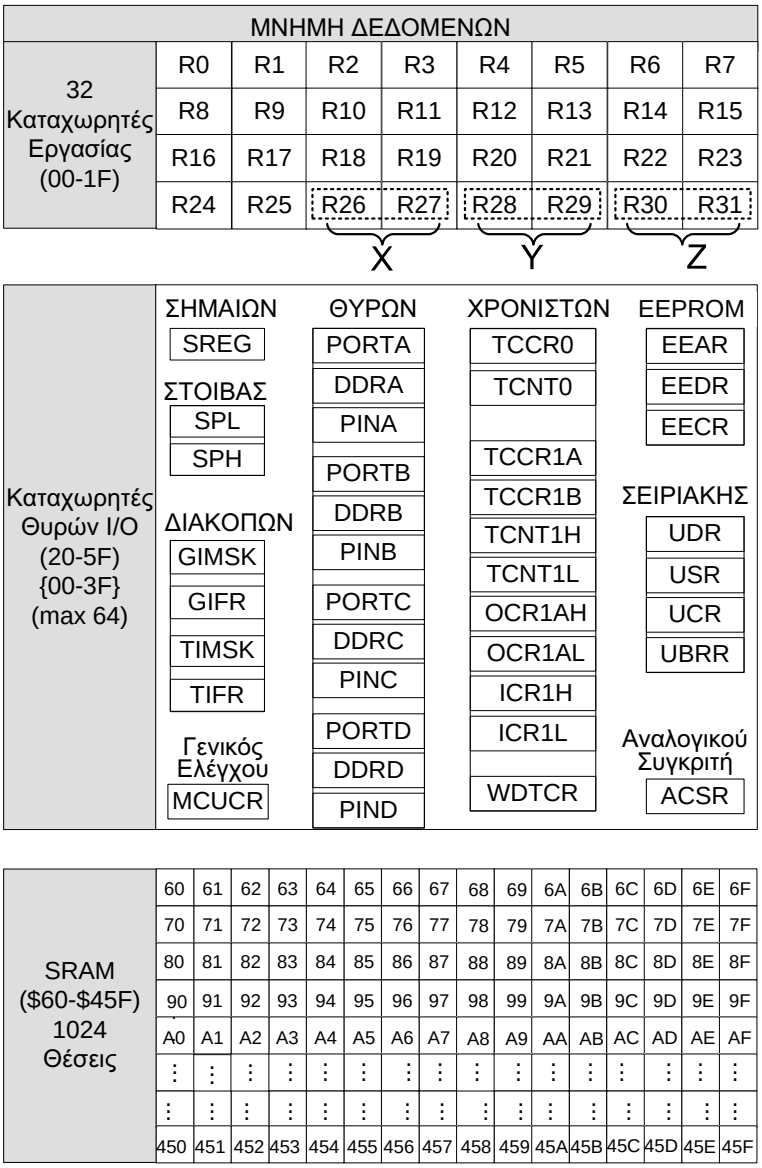
Η μνήμη προγράμματος έχει δυνατότητα προγραμματισμού εντός του συστήματος (In-System Programmable Flash memory) μέσω μιας σειριακής διεπαφής SPI ή με ένα συμβατικό προγραμματιστή. Ενδεικτικά ο μικροελεγκτής ATmega16 περιέχει 8K bytes ενσωματωμένης In-System Programmable Flash μνήμης.

Η **μνήμη δεδομένων** συνδέεται με ένα διάδρομο των 8-bit για την επικοινωνία των περιφερειακών μονάδων με τους καταχωρητές ελέγχου. Η μνήμη δεδομένων, όπως φαίνεται στο Σχ. 1.11, χωρίζεται στα εξής τμήματα:

- Στην ομάδα καταχωρητών εργασίας (register file) των 8-bits.
- Στους καταχωρητές εισόδου-εξόδου των 8-bits.
- Στην εσωτερική στατική μνήμη SRAM. Η μνήμη αυτή χρησιμοποιείται για αποθήκευση μεταβλητών και ως στοίβα (stack).
- Στην εξωτερική στατική μνήμη SRAM. Σε ελεγκτές που διαθέτουν θύρες πρόσβασης σε εξωτερικό διάδρομο μπορούμε να συνδέσουμε εξωτερική μνήμη.

Οι θέσεις της μνήμη δεδομένων που αποτελείται από τους καταχωρητές εργασίας (32), τους καταχωρητές εισόδου–εξόδου (64 θέσεις) και τη μνήμη SRAM (1KB), είναι συνεχόμενες και μπορούν να θεωρηθούν ως μια ενιαία μνήμη. Η πρόσβαση σε αυτές επιτυγχάνεται εύκολα μέσω των διαφορετικών τρόπων διευθυνσιοδότησης που υποστηρίζει η αρχιτεκτονική AVR.

Εφόσον στους καταχωρητές εργασίας ανατίθενται οι 32 χαμηλότερες διευθύνσεις της μνήμης δεδομένων ($00-$1F), μπορούμε να τους χειριστούμε σαν να πρόκειται για μνήμη και όχι μόνο ως ανεξάρτητους καταχωρητές. Άρα, λοιπόν, η πρόσβαση επιτυγχάνεται και με τους συμβατικούς τρόπους (εντολές) αναφοράς στη μνήμη.



Σχήμα 1.11 Μνήμη δεδομένων

Οι θέσεις *μνήμης καταχωρητών εισόδου-εξόδου* περιέχουν 64 διευθύνσεις για λειτουργίες περιφερειακών της ΚΜΕ, όπως καταχωρητές ελέγχου, μονάδα USART, χρονιστές/μετρητές, A/D μετατροπείς, αναλογικό συγκριτή και άλλες λειτουργίες I/O. Η πρόσβαση σε αυτές τις θέσεις μπορεί να είναι μέσω εντολών *εισόδου-εξόδου* (π.χ. IN UDR, r1 ή OUT r1, UDR) και διευθύνσεις για τις θύρες από (00-$3F) ή ως θέσεις μνήμης μέσω της αντίστοιχης διεύθυνσης ($20-$5F). Για

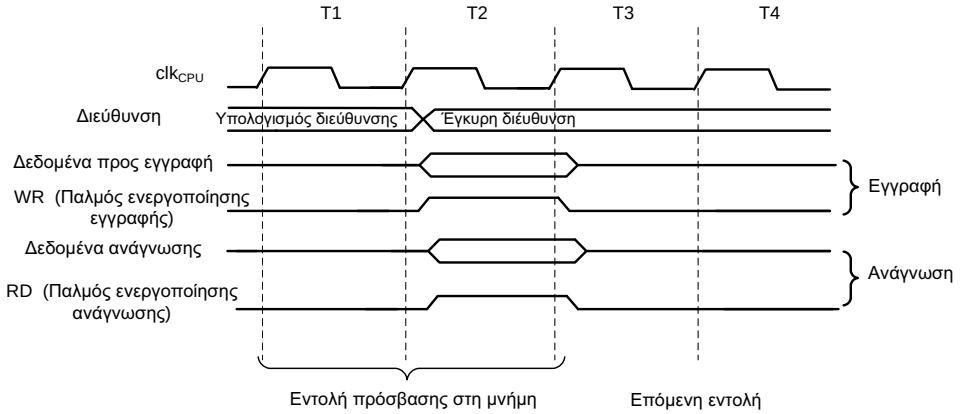
παράδειγμα, ισοδύναμες με τις προηγούμενες είναι οι εντολές LDS Rd, Address και ST Address, Rd αντίστοιχα, όπου Address=UDR+$20.
Ενδεικτικά για το μικροελεγκτή ATmega16, οι πρώτες 96 θέσεις μνήμης δεδομένων αντιστοιχούν σε 32 καταχωρητές του Register File (00-$1F) και 64 θύρες I/O ($20-$5F) και οι επόμενες 1024 ($60-$45F) αντιστοιχούν στη μνήμη SRAM, η οποία εξηγείται αναλυτικά στο επόμενο κεφάλαιο.

1.4 Μνήμη SRAM

Τα περισσότερα μοντέλα μικροελεγκτών διαθέτουν μνήμη δεδομένων SRAM (static RAM). Η χρήση της μπορεί να αποφευχθεί μόνο σε απλά προγράμματα, όπου τα δεδομένα μπορούν να αποθηκευτούν σε καταχωρητές.

Η SRAM είναι μνήμη η οποία δεν είναι απευθείας προσβάσιμη από την κεντρική μονάδα επεξεργασίας (Arithmetic and Logical Unit ALU ή συσσωρευτής), σε αντίθεση με τους καταχωρητές. Για την πρόσβαση σε θέσεις μνήμης συνήθως χρησιμοποιούμε κάποιον καταχωρητή για προσωρινή ενδιάμεση αποθήκευση. Το μέγεθος της κυμαίνεται από 128 bytes (π.χ. στα μοντέλα AT90S2323, AT Tiny13) έως 4096 bytes (στα μοντέλα AT Mega103, AT Mega603).

Προφανώς οι λειτουργίες μέσω της SRAM είναι πιο αργές, σε σχέση με τους καταχωρητές. Ο χρόνος πρόσβασης στη μνήμη απαιτεί δυο περιόδους του σήματος χρονισμού. Το γεγονός αυτό οφείλεται στο ότι η πρόσβαση στην SRAM χρησιμοποιεί έναν καταχωρητή δείκτη (ζεύγος δύο καταχωρητών), ο οποίος καθορίζει τη διεύθυνση στη μνήμη. Κατά την πρώτη περίοδο του κεντρικού ρολογιού εκτελούνται οι απαιτούμενες λειτουργίες των καταχωρητών για τον καθορισμό της διεύθυνσης. Στη δεύτερη περίοδο λαμβάνει χώρα η πρόσβαση στη μνήμη (εγγραφή ή ανάγνωση). Οι φάσεις ανάγνωσης-εγγραφής φαίνονται στο σχήμα 1.12.



Σχήμα 1.12 Φάσεις ανάγνωσης-εγγραφής στην εσωτερική μνήμη SRAM

Από την άλλη πλευρά η ελάχιστη χωρητικότητα είναι 128 bytes, πολύ περισσότερο από τη χωρητικότητα των 32 καταχωρητών. Τα μοντέλα όπως το AT Mega16 προσφέρουν τη δυνατότητα για σύνδεση εξωτερικής μνήμης SRAM (το συγκεκριμένο 1KB) για εφαρμογές με μεγάλες απαιτήσεις χωρητικότητας.
Η πρόσβαση επιτυγχάνεται όπως και στην ενσωματωμένη SRAM χωρίς τη χρήση επιπλέον εντολών. Εκτός από την απευθείας πρόσβαση με χρήση διευθύνσεων, υπάρχει και η έμμεση πρόσβαση με χρήση καταχωρητή δείκτη με ή χωρίς αύξηση, μείωση και μετατόπιση.

Η πιο σημαντική χρήση της SRAM είναι η λειτουργία στοίβας. Στη στοίβα μπορούμε να αποθηκεύουμε (εντολή push) πληροφορίες και έπειτα να τις ανακτήσουμε (εντολή pop). Οι πληροφορίες μπορεί να είναι τιμές, το περιεχόμενο ενός καταχωρητή, η διεύθυνση επιστροφής πριν την κλήση υπορουτίνας ή διεύθυνση επιστροφής από διακοπή του υλικού.

Σύνδεση εξωτερικής μνήμης SRAM

Σε εφαρμογές υψηλών απαιτήσεων σε μνήμη είναι δυνατή η σύνδεση εξωτερικής μνήμης SRAM. Ο τρόπος σύνδεσης γίνεται συνήθως μέσω των θυρών PORTA, PORTC και του σήματος ALE. Το σήμα ALE (Address Latch Enable) πολυπλέκει το διάδρομο δεδομένων (PORTC) με το διάδρομο διευθύνσεων (PORTA). Η θύρα PORTA γίνεται το τμήμα AD0-7 του διάδρομου δ/νσεων της εξωτερικής μνήμης και η θύρα PORTC γίνεται το τμήμα AD8-15 του διάδρομου διευθύνσεων.
Η σύνδεση εξωτερικής μνήμης ενεργοποιείται θέτοντας το bit7 (SRE) του καταχωρητή MCUCR. Ο χρόνος πρόσβασης στην εξωτερική μνήμη διαρκεί τρείς περιόδους του σήματος χρονισμού. Όταν όμως τίθεται και το bit6 (SRW) του καταχωρητή MCUCR, εισάγεται ένας επιπλέον κύκλος αναμονής, οπότε η διαδικασία πρόσβασης διαρκεί τέσσερις περιόδους.

1.5 Εντολές

Οι εντολές της γλώσσας των μικροελεγκτών AVR μπορούν να ομαδοποιηθούν στις παρακάτω τέσσερις κατηγορίες :

α) Εντολές μεταφοράς δεδομένων
β) Εντολές αριθμητικών και λογικών πράξεων
γ) Εντολές σε επίπεδο bit και ελέγχου bit
δ) Εντολές ελέγχου ροής του προγράμματος και διακλάδωσης

Οι εντολές περιλαμβάνουν 2 τμήματα. Το πρώτο (operation code ή opcode) αποτελεί το λειτουργικό κώδικα που πληροφορεί τον επεξεργαστή για τις ενέργειες που πρέπει να εκτελεστούν. Το δεύτερο τμήμα προσδιορίζει το όρισμα (operand) για τον οποίο θα λειτουργήσει ο κώδικας.

1.5.1 Εντολές μεταφοράς δεδομένων

Πρόκειται για εντολές μεταφοράς δεδομένων μεταξύ καταχωρητών ή μεταξύ καταχωρητών και μνήμης ή μεταξύ καταχωρητή και σταθεράς. Η εκτέλεση των εντολών αυτών δεν επηρεάζει καμία σημαία. Ο καταχωρητής είναι ένας από τους 32 καταχωρητές εργασίας R0-R31 που διαθέτουν το αντίστοιχο τμήμα μνήμης (register file). Με Rd συμβολίζεται ο καταχωρητής εργασίας που αποτελεί τη θέση προορισμού του αποτελέσματος και με Rr συμβολίζεται ο καταχωρητής εργασίας που αποτελεί τη θέση προέλευσης των δεδομένων.
Το k είναι μια σταθερή διεύθυνση και K ένα σταθερό δεδομένο (8 bit). Οι καταχωρητές X, Y, Z λέγονται καταχωρητές δείκτη και αντιστοιχούν στα ζεύγη X=R27:R26, Y=R29:R28 και Z=R31:R30. Χρησιμοποιούνται ως 16-bit καταχωρητές δείκτη για πρόσβαση προς θέσεις μνήμης SRAM ή προς θέσεις μνήμης προγράμματος (μονάχα ο Z). Με το P συμβολίζεται ένας καταχωρητής εισόδου-εξόδου. Τέλος, q (6-bit) είναι η μετατόπιση για έμμεση διευθυνσιοδότηση. Οι εντολές LD και ST, με όλες τις παραλλαγές τους, εφαρμόζονται και για τους τρεις καταχωρητές δείκτη. Αντίθετα, οι εντολές LDD, STD εφαρμόζονται μόνο για τους καταχωρητές δείκτη Y και Z (δεν εφαρμόζονται για τον X).

Εντολές μεταφοράς δεδομένων

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
MOV	Rd, Rr	Move Between Registers	Rd← Rr	-	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd← Rr+1:Rr	-	1
LDI	Rd, K	Load Immediate	Rd ← K	-	1
LD	Rd, X	Load Indirect	Rd ← (X)	-	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X← X+ 1	-	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X← X – 1, Rd← (X)	-	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	-	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y← Y+ 1	-	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y← Y – 1, Rd← (Y)	-	2
LDD	Rd, Y+q	Load Indirect + Displacement	Rd ← (Y+ q)	-	2

LD	Rd, Z	Load Indirect	Rd ← (Z)	-	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z← Z+ 1	-	2
LD	Rd, - Z	Load Indirect and Pre-Dec.	Z←Z – 1, Rd← (Z)	-	2
LDD	Rd, Z+q	Load Indirect + Displacement	Rd ← (Z+ q)	-	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	-	2
ST	X, Rr	Store Indirect	(X)←Rr	-	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X)←Rr, X← X+ 1	-	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X – 1, (X)←Rr	-	2
ST	Y, Rr	Store Indirect	(Y)←Rr	-	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y)←Rr, Y← Y+ 1	-	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y – 1, (Y)←Rr	-	2
STD	Y+q,Rr	Store Indirect + Displacement	(Y + q) ← Rr	-	2
ST	Z, Rr	Store Indirect	(Z)←Rr	-	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z)←Rr, Z← Z+ 1	-	2
ST	- Z, Rr	Store Indirect and Pre-Dec.	Z ← Z – 1, (Z)←Rr	-	2
STD	Z+q,Rr	Store Indirect + Displacement	(Z + q) ← Rr	-	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	-	2
LPM		Load Program Memory	R0 ← (Z)	-	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	-	3
LPM	Rd, Z+	Load Program Memory	Rd ← (Z), Z← Z+ 1	-	3
SPM		Store Program Memory	(Z) ← R1:R0	-	-
IN	Rd, P	In Port	Rd ← P	-	1
OUT	P, Rr	Out Port	P ← Rr	-	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	-	2
POP	Rd	Pop Register from Stack	Rd ← STACK	-	2

Πίνακας 1.3 Πρόσβαση μέσω δεικτών

Δείκτης	**Λειτουργία**	**Παράδειγμα**
Z	Ανάγνωση από διεύθυνση Z, Z αμετάβλητος Εγγραφή προς διεύθυνση Z, Z αμετάβλητος	LD R3,Z ST Z,R3
Z+	Ανάγνωση από διεύθυνση Z, και αύξηση κατά 1 Εγγραφή προς διεύθυνση Z, και αύξηση κατά 1	LD R3,Z+ ST Z+,R3
-Z	Ανάγνωση από διεύθυνση Z, και μείωση κατά 1 Εγγραφή προς διεύθυνση Z, και μείωση κατά 1	LD R3,-Z ST -Z,R3

Παρόμοια ισχύουν και για τους καταχωρητές X και Y.

Κανόνες για τη χρήση καταχωρητών:

1. Ορίζετε ονόματα καταχωρητών με την οδηγία .DEF, αντί να τους χρησιμοποιείται απευθείας με το όνομά τους Rx.
2. Κρατήστε τους R26-R31 για πρόσβαση μέσω καταχωρητή δείκτη.
3. Ως 16-bit μετρητής προτιμάται το ζεύγος R25:R24.
4. Για πρόσβαση σε μεμονωμένα bits εντός συγκεκριμένων καταχωρητών (π.χ. για έλεγχο σημαιών), χρησιμοποιήστε τους R16-R23.
5. Για ανάγνωση από τη μνήμη προγράμματος, π.χ. ενός πίνακα δεδομένων, διαφυλάξτε τον R0 και τον Z (R31:R30).

Μεταφορά δεδομένων μεταξύ καταχωρητών:
MOV Rd, Rr **:** Rd ← Rr
Περιγραφή: Το περιεχόμενο του καταχωρητή Rr αντιγράφεται στον Rd. Ο Rr παραμένει αναλλοίωτος
MOVW Rd, Rr **:** Rd+1:Rd ← Rr+1:Rr
Περιγραφή: Το περιεχόμενο του ζεύγους καταχωρητών Rr+1:Rr αντιγράφεται στο ζεύγος καταχωρητών Rd+1:Rd.

Άμεση φόρτωση σταθεράς σε καταχωρητή:
LDI Rd, K **:** Rd ←K
Περιγραφή: Άμεση φόρτωση μιας σταθεράς 8-bit στον καταχωρητή Rd. Ο καταχωρητής Rd είναι ένας από τους R16-R31.

Απευθείας φόρτωση/ αποθήκευση από/προς θέση μνήμης SRAM:
LDS Rd, k **:** Rd ←(k)
Περιγραφή: Τα περιεχόμενα της θέσης μνήμης (k) φορτώνονται στον καταχωρητή Rd.
STS k, Rr **:** (k) ←Rr
Περιγραφή: Απευθείας αποθήκευση καταχωρητή. Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (k).

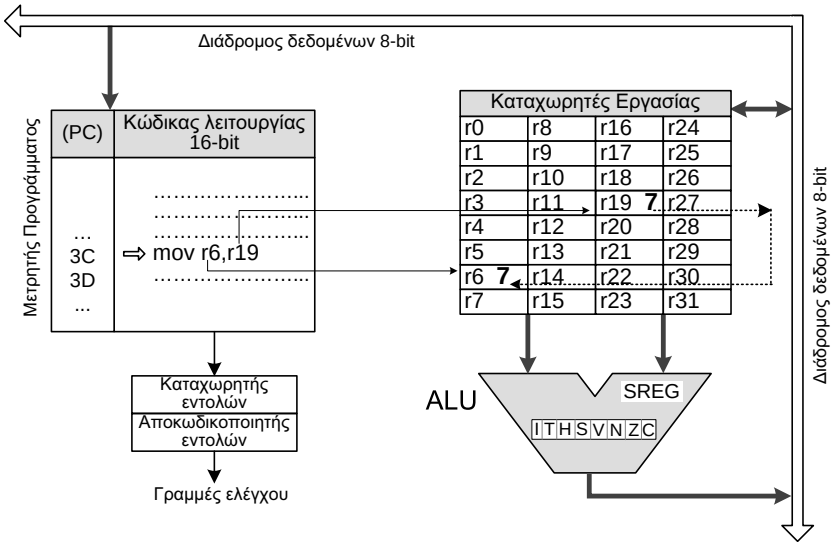
Έμμεση φόρτωση/αποθήκευση καταχωρητή μέσω ενός καταχωρητή δείκτη (X, Y, Z). Οι επόμενες εντολές εφαρμόζονται αντίστοιχα και για τους τρεις καταχωρητές δείκτη. Επίσης, είναι δυνατή η αύξηση ή μείωση του καταχωρητή δείκτη κατά μία μονάδα, καθώς και η χρήση μετατόπισης κατά την πρόσβαση σε θέσεις μνήμης μέσω καταχωρητή δείκτη:

LD Rd, X **:** Rd ←(X)
Περιγραφή: Έμμεση φόρτωση του καταχωρητή Rd με τα περιεχόμενα της θέσης μνήμης (X). Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27.

ST X, Rr **:** (X) ←Rr
Περιγραφή: Έμμεση αποθήκευση καταχωρητή. Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (X).
Επίσης, είναι δυνατή η αύξηση ή μείωση του καταχωρητή δείκτη κατά μία μονάδα, καθώς και η χρήση μετατόπισης κατά την πρόσβαση σε θέσης μνήμης μέσω καταχωρητή δείκτη. Αντίστοιχες εντολές υπάρχουν και για τους καταχωρητές δείκτη Y και Z.

Χειρισμός θέσης μνήμης προγράμματος:
LPM : R0 ←(Z)
Περιγραφή: Φόρτωση των περιεχομένων μιας θέσης μνήμης. Το περιεχόμενο της θέσης μνήμης προγράμματος (Z) φορτώνεται στον καταχωρητή R0

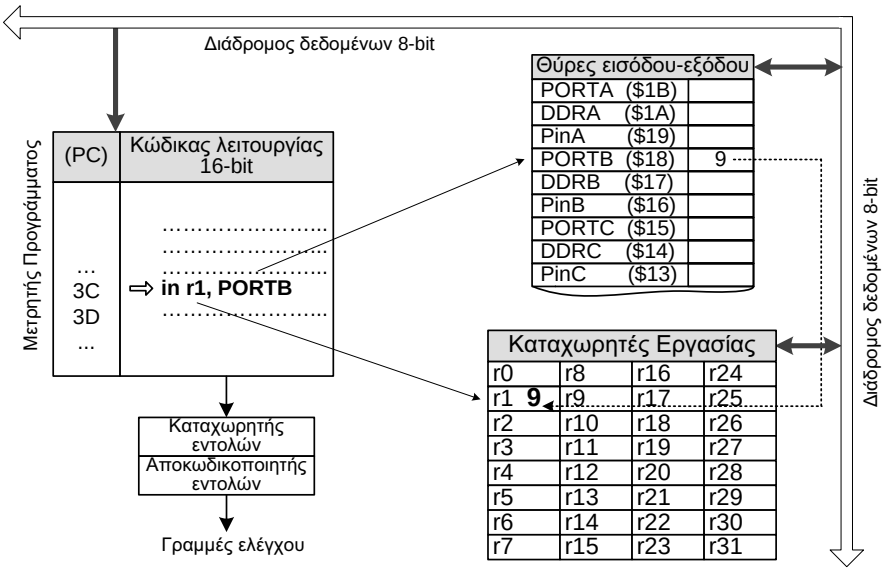
SPM : (Z) ←R1:R0
Περιγραφή: (Store Program Memory) Έμμεση αποθήκευση ζεύγους καταχωρητών. Το περιεχόμενο των καταχωρητών R1:R0 αποθηκεύεται στη θέση μνήμης προγράμματος που δείχνει ο καταχωρητής δείκτη Z. Μέσω της εντολής αυτής, είναι δυνατός ο προγραμματισμός εντός κυκλώματος (in-circuit programming) της μνήμης προγράμματος flash.



Σχήμα 1.13 Παράδειγμα μεταφοράς μεταξύ καταχωρητών

Η πρόσβαση στις θέσεις μνήμης που αντιστοιχούν οι μονάδες εισόδου-εξόδου (I/O memory) επιτυγχάνεται με 2 εντολές: την IN Rd, PortAddress και την OUT PortAddress, Rr όπου PortAddress είναι η διεύθυνση του καταχωρητή εισόδου-εξόδου.

IN Rd, A **:** Rd ← I/O(A) και **OUT** A, Rr **:** I/O(A) ← Rr
Περιγραφή: Εισαγωγή ή Εξαγωγή δεδομένων από/προς καταχωρητή εισόδου-εξόδου I/O(A) (θύρες, χρονιστές, κλπ.) προς/από καταχωρητή Rd.
Παραδείγματα τέτοιων εντολών είναι: IN Rd,PINB και OUT PORTD,Rr



Σχήμα 1.14 Παράδειγμα ανάγνωσης μιας θύρας

Για αποθήκευση δεδομένου στη στοίβα κατά την κλήση κάποιας υπορουτίνας, διακοπής κλπ. ή για ανάκτηση δεδομένου από τη στοίβα πριν την επιστροφή από υπορουτίνα, χρησιμοποιούνται οι δυο επόμενες εντολές:

PUSH Rr **:** STACK ←Rr

Περιγραφή: Αποθήκευση περιεχομένων καταχωρητή Rr στη στοίβα. Μείωση του δείκτη στοίβας κατά μία μονάδα.

POP Rd **:** Rd ←STACK

Περιγραφή: Αύξηση του δείκτη στοίβας κατά μία μονάδα. Αποθήκευση περιεχομένων της στοίβας στον καταχωρητή Rd.

1.5.2 Εντολές αριθμητικών και λογικών πράξεων

Οι εντολές της ομάδας αυτής προϋποθέτουν τη χρήση της αριθμητικής λογικής μονάδας σε αντίθεση με τις εντολές μεταφοράς δεδομένων. Εκτελούνται μόνο μεταξύ Καταχωρητών Εργασίας ή Καταχωρητή Εργασίας με σταθερά. Διαβάζονται τα περιεχόμενα των καταχωρητών, εκτελείται η πράξη με αυτά και αποθηκεύεται το αποτέλεσμα στον ίδιο ή σε άλλον καταχωρητή (μόνο στον πολλαπλασιασμό).

Εντολές αριθμητικών και λογικών πράξεων

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Regs	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 Regs	Rd ← Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Const from Reg	Rd ←Rd – K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract & C two Regs	Rd ← Rd – Rr – C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract & C Const from Reg	Rd ← Rd – K – C	Z,C,N,V,H	1
SBIW	Rdl, K	Subtract Imm from Word	Rdh:Rdl ←Rdh:Rdl – K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd Λ Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Reg & Const	Rd ← Rd Λ K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Reg & Const	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 ← $FF ⊕ Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd V K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd Λ ($FF – 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 Λ Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	-	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd × Rr	Z,C	2

Οι εντολές αριθμητικών πράξεων διακρίνονται σε εντολές πρόσθεσης και αφαίρεσης με ή χωρίς κρατούμενο μεταξύ δύο καταχωρητών ή ενός καταχωρητή και μιας σταθεράς ή ζεύγους καταχωρητών και μιας σταθεράς, και εντολές πολλαπλασιασμού προσημασμένων ή μη.

Εντολές πρόσθεσης:

ADD Rd,Rr : Rd ← Rd + Rr
Περιγραφή: Πρόσθεση 2 καταχωρητών χωρίς χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd.
ADC Rd,Rr **:** Rd ← Rd + Rr+C
Περιγραφή: Πρόσθεση 2 καταχωρητών με χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd.
ADIW Rd,K **:** Rd+1:Rd ←Rd+1:Rd + K
Περιγραφή: Πρόσθεση μιας τιμής K(0-63) σε ένα ζεύγος καταχωρητών και αποθήκευση στο ζεύγος καταχωρητών.

Εντολές αφαίρεσης:

SUB Rd,Rr **:** Rd ← Rd - Rr
Περιγραφή: Αφαίρεση 2 καταχωρητών χωρίς χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd
SUBI Rd,K **:** Rd ←Rd - K
Περιγραφή: Αφαίρεση μιας τιμής χωρίς χρήση κρατουμένου από έναν καταχωρητή και αποθήκευση στον καταχωρητή
SUBCI Rd,K **:** Rd ←Rd – K-C
Περιγραφή: Αφαίρεση μιας τιμής με χρήση κρατουμένου από έναν καταχωρητή και αποθήκευση στον καταχωρητή
SBC Rd,Rr **:** Rd ← Rd – Rr-C
Περιγραφή: Αφαίρεση 2 καταχωρητών με χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd.
SBIW Rd,K **:** Rd+1:Rd ←Rd+1:Rd - K
Περιγραφή: Αφαίρεση μιας τιμής από ένα ζεύγος καταχωρητών και αποθήκευση στο ζεύγος καταχωρητών.

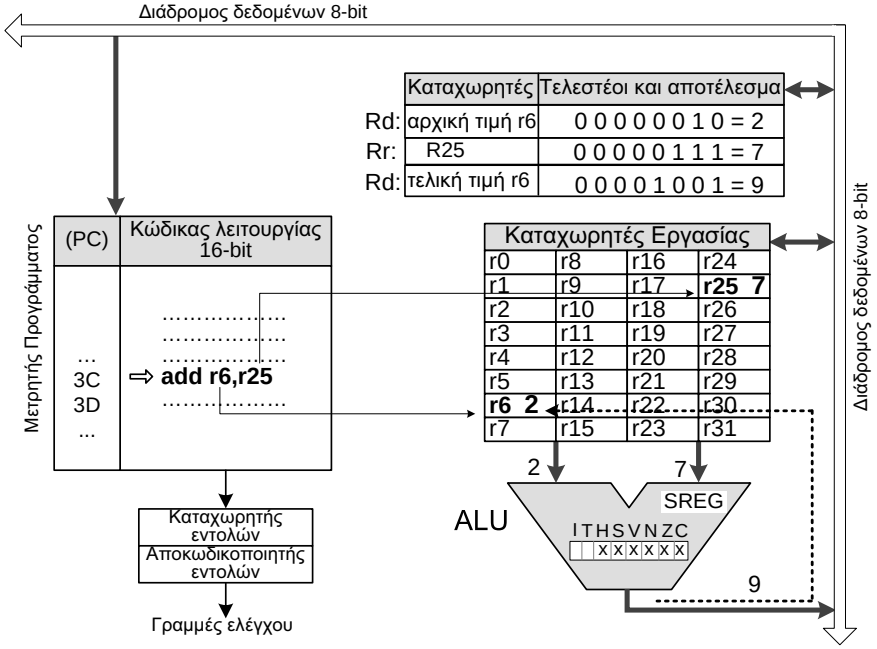
Υπάρχουν οι εντολές πολλαπλασιασμού MUL, MULS, MULSU, μεταξύ προσημασμένων ή μη αριθμών, καθώς και οι εντολές FMUL, FMULS, FMULSU πολλαπλασιασμού κλασματικών αριθμών (σταθερής υποδιαστολής σε μορφή 1.7 και το αποτέλεσμα σε μορφή 1.15) επίσης προσημασμένων ή μη αριθμών που παρατίθενται αναλυτικότερα στο παράρτημα.

Εντολές πολλαπλασιασμού:

MUL Rd,Rr **:** R1, R0 ←Rd × Rr
Περιγραφή: Μη προσημασμένος πολλαπλασιασμός μεταξύ των περιεχομένων των καταχωρητών Rd, Rr. Το αποτέλεσμα αποθηκεύεται στο ζεύγος καταχωρητών R1, R0. Η εντολή **MULS** αφορά σε πολλαπλασιασμό μεταξύ δύο προσημασμένων ενώ η εντολή

MULSU σε πολλαπλασιασμό μεταξύ προσημασμένου (Rd) και μη-προσημασμένο (Rr) αριθμού.

FMUL Rd,Rr **:** R1, R0 ←Rd × Rr
Περιγραφή: Μη Προσημασμένος κλασματικός πολλαπλασιασμός μεταξύ των περιεχομένων των καταχωρητών Rd (μορφοποίηση 1.7) και Rr (μορφοποίηση 1.7). Το αποτέλεσμα αποθηκεύεται στο ζεύγος καταχωρητών R1, R0 (μορφοποίηση 1.15). Η εντολή **FMULS** αφορά σε πολλαπλασιασμό μεταξύ δύο προσημασμένων ενώ η εντολή **FMULSU** σε πολλαπλασιασμό μεταξύ προσημασμένου (Rd) και μη-προσημασμένο (Rr) αριθμού.



Σχήμα 1.15 Παράδειγμα απευθείας διευθυνσιοδότησης δύο καταχωρητών

Επίσης, στην ομάδα αυτή ανήκουν οι εντολές λογικών πράξεων, εντολές αύξησης, ελάττωσης, συμπληρώματος ως προς ένα και ως προς δύο ενός καταχωρητή. Ακόμα συμπεριλαμβάνονται και εντολές μηδενισμού ή ανάθεσης λογικού 1 στα bit ενός καταχωρητή.

Εντολές λογικών πράξεων (AND, OR και XOR)**:**

AND Rd, Rr **:** Λογικό AND Rd ←Rd Λ Rr
Περιγραφή: Λογικό AND μεταξύ των περιεχομένων των καταχωρητών Rd και Rr. Το αποτέλεσμα αποθηκεύεται στον καταχωρητή Rd.

OR Rd, Rr **:** Λογικό OR Rd ←Rd v Rr
Περιγραφή: Λογικό OR μεταξύ των περιεχομένων των καταχωρητών Rd και Rr.
Το αποτέλεσμα αποθηκεύεται στον καταχωρητή Rd.

EOR Rd, Rr **:** Λογικό EXOR Rd ←Rd ⊕ Rr
Περιγραφή: Λογικό EXOR (αποκλειστικό ‘ή’) μεταξύ των περιεχομένων των καταχωρητών Rd και Rr. Το αποτέλεσμα αποθηκεύεται στον καταχωρητή Rd.

Αντίστοιχα, υπάρχουν οι εντολές λογικού ΚΑΙ (AND) και Η (OR) μεταξύ του περιεχομένου ενός καταχωρητή Rd και ενός σταθερού αριθμού Κ. Το αποτέλεσμα αποθηκεύεται στον καταχωρητή Rd.:

ANDI Rd, K
ORI Rd, K.

Εντολές συμπληρώματος ως προς 1 και ως προς 2:

COM Rd **:** Rd ←$FF - Rd
Περιγραφή: Αντιστροφή των περιεχομένων του καταχωρητή Rd με χρήση αριθμητικής συμπληρώματος του 1.

NEG Rd **:** Rd ←$00 - Rd
Περιγραφή: Αντιστροφή των περιεχομένων του καταχωρητή Rd με χρήση αριθμητικής συμπληρώματος του 2.

Τοποθέτηση λογικού ‘1’ ή ‘0’ στα bits καταχωρητή:
SBR Rd,K **:** Rd ←Rd v K
Περιγραφή: Τοποθέτηση λογικού ‘1’ στα bits του καταχωρητή Rd εκτελώντας λογικό Η (OR) μεταξύ των περιεχομένων του καταχωρητή Rd και της σταθεράς Κ

CBR Rd,K **:** Rd ← Rd Λ ($FFh -K)
Περιγραφή: Τοποθέτηση λογικού ‘0’ στα bits του καταχωρητή Rd εκτελώντας λογικό ΚΑΙ (AND) μεταξύ των περιεχομένων του καταχωρητή Rd και της σταθεράς Κ.

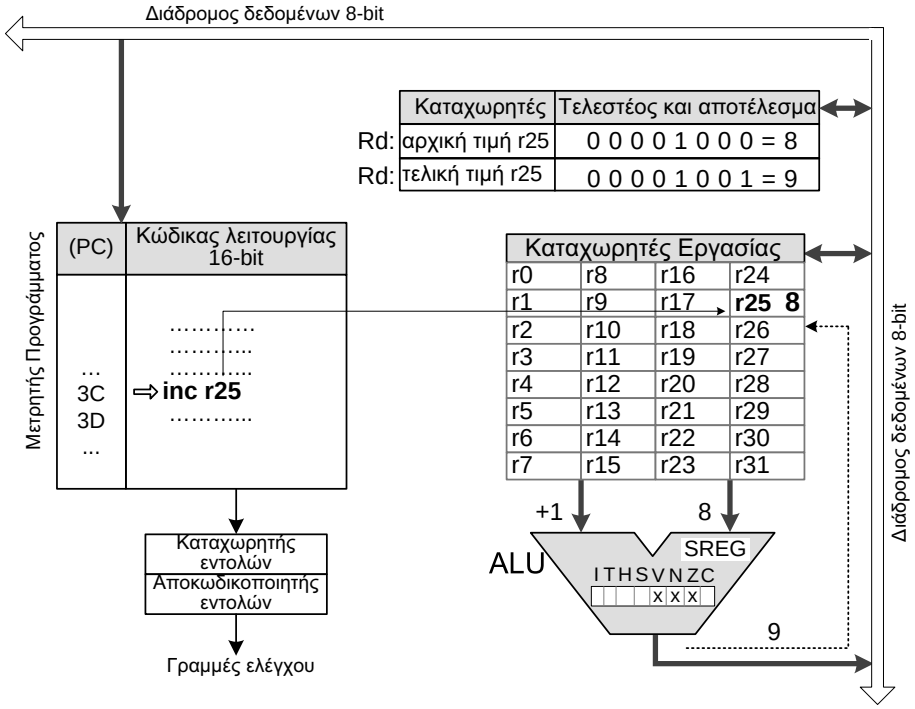
Εντολές αύξησης/μείωσης καταχωρητή:

INC Rd **:** Rd ←Rd + 1
Περιγραφή: Αύξηση του περιεχομένου του καταχωρητή Rd κατά μία μονάδα.

DEC Rd**:** Rd ←Rd - 1
Περιγραφή: Μείωση του περιεχομένου του καταχωρητή Rd κατά μία μονάδα.
Μηδενισμός/ θέσιμο καταχωρητή:

CLR Rd **:** Rd ←Rd ⊕ Rd
Περιγραφή: Μηδενισμός του καταχωρητή Rd με εκτέλεση αποκλειστικού ‘ή’ μεταξύ του καταχωρητή και του εαυτού του.

SER Rd**:** Rd ←$FF
Περιγραφή: Τοποθέτηση ‘1’ σε όλα τα bits του καταχωρητή με φόρτωση της τιμής $FF.



Σχήμα 1.16 Παράδειγμα απευθείας διευθυνσιοδότησης ενός καταχωρητή

Παράδειγμα πρόσβασης σε θέση μνήμης προγράμματος

Η πρόσβαση σε θέσεις μνήμης προγράμματος επιτυγχάνεται με την εντολή LPM (Load from Program Memory) που ορίζεται για τον καταχωρητή Z. Η εντολή αντιγράφει το περιεχόμενο της θέσης μνήμης προγράμματος (Z) στον καταχωρητή R0. Η μνήμη προγράμματος οργανώνεται κατά λέξη (2 bytes ή 16 bits), οπότε το LSB αντιστοιχεί στο κάτω byte και το MSB στο άνω byte. Η αρχική διεύθυνση πρέπει να πολλαπλασιαστεί επί δύο και η πρόσβαση περιορίζεται σε 15-bit ή 32 kB μνήμης προγράμματος. Δηλαδή:

```
LDI ZH,HIGH(2*Address)
 LDI ZL,LOW(2*Address)
 LPM
```

Στη συνέχεια, για να διαβάσουμε το επόμενο byte της μνήμης προγράμματος, η διεύθυνση πρέπει να αυξηθεί ως εξής:

```
ADIW ZL,1
LPM
```

όπου ADIW Rd,K (ADd Immediate Word) σημαίνει πρόσθεση μιας τιμής K(0-63) σε ένα ζεύγος καταχωρητών και αποθήκευση στο ζεύγος καταχωρητών.

Ανάλογα, υπάρχει η εντολή SBIW ZL,1 για να γυρίσουμε στο προηγούμενο byte, όπου SBIW Rd,K (SuBtract Immediate Word) σημαίνει αφαίρεση μιας τιμής K(0-63) από ένα ζεύγος καταχωρητών και αποθήκευση του αποτελέσματος στο ζεύγος καταχωρητών. Αυτές οι δυο εντολές εφαρμόζονται στα ζεύγη καταχωρητών X,Y,Z και R25:R24. Ο τελευταίος δε χρησιμοποιείται για πρόσβαση σε θέση μνήμης, αλλά μόνο για χειρισμό 16-bit τιμών.
Οι οδηγίες .DB και .DW προς το μεταφραστή χρησιμοποιούνται για την εισαγωγή ενός πίνακα δεδομένων στη μνήμη προγράμματος.

Εισαγωγή κατά byte γίνεται με την ετικέτα .DB, ως εξής:

```
.DB 64,28,255,4          ; λίστα 4 bytes
.DB "This is a text."    ; λίστα byte χαρακτήρων
```

Για την αποδοτικότερη αξιοποίηση της μνήμης προγράμματος που έχει πλάτος 16 bit, bytes ή χαρακτήρες αποθηκεύονται ανά ζεύγη και η εισαγωγή γίνεται κατά λέξη με την .DW, ως εξής:

```
.DW 23013, 8515                ; λίστα 2 λέξεων
```

Αντί σταθερών τιμών, μπορούμε να τοποθετήσουμε ετικέτες:

```
Ετικέτα1:
[... κώδικας ... ]
Ετικέτα2:
[... επιπλέον κώδικας ... ]
Table:
.DW Ετικέτα1,Ετικέτα2    ; λίστα ετικετών σε λέξεις
```

Επίσης, συχνή είναι η χρήση δεικτών για πρόσβαση σε πίνακα δεδομένων εντός της μνήμης προγράμματος. Στο επόμενο παράδειγμα παρουσιάζεται ένας πίνακας με 10 τιμές-λέξεις (16-bit), όπου η έκτη τιμή διαβάζεται και αποθηκεύεται στους καταχωρητές R25:R24.

```
                                ; οι τιμές του πίνακα είναι οργανωμένες κατά λέξη
Table:
 .DW 0x1234,0x2345,0x3456,0x4567,0x5678
 .DW 0x6789,0x789A,0x89AB,0x9ABC,0xABCD
                                ; ακολουθεί η ανάγνωση της 6ης λέξης
    LDI ZH,HIGH(Table*2)        ; η διεύθυνση του πίνακα στον καταχωρητή Z.
    LDI ZL,LOW(Table*2)         ; πολλαπλασιασμός επί 2 για πρόσβαση κατά byte
    ADIW ZL,10                  ; δείχνουμε στο έκτο στοιχείο
    LPM                         ; ανάγνωση LSB από μνήμη προγράμματος (0x89)
    MOV R24,R0                  ; αντιγραφή LSB στον καταχωρητή R24 (0x89)
    ADIW ZL,1                   ; δείχνουμε στο MSB στη μνήμη προγράμματος
    LPM                         ; ανάγνωση του MSB (0x67)
    MOV R25,R0                  ; αντιγραφή MSB στον καταχωρητή R25
```

Παραδείγματα χρήσης εντολών μεταφοράς δεδομένων και αριθμητικών πράξεων:

Παράδειγμα 1.1:
Ένας 16-bit αριθμός, που βρίσκεται στις θέσεις μνήμης 0x60-0x61, προστίθεται με έναν 16-bit αριθμό, που βρίσκεται στις θέσεις μνήμης 0x62-0x63. Το αποτέλεσμα σώζεται στη διεύθυνση 0x64-0x65 της μνήμης δεδομένων.

```
.include "m16def.inc"   ; δήλωση μικροελεγκτή
.org 0x000
main:           ; κυρίως πρόγραμμα
    ldi xl, 0x60  ; θέτουμε διεύθυνση 0x60 στον καταχωρητή X
    clr xh        ; όπου xl=R26, xh=R27

    ld r1,x+      ; ανάγνωση θέσης μνήμης, αποθήκευση σε καταχωρητή
                  ; και αύξηση δείκτη για επόμενη θέση
    ld r2,x+      ; ανάγνωση από τη θέση 0x61
    ld r3,x+      ; ανάγνωση από τη θέση 0x62
    ld r4,x+      ; ανάγνωση από τη θέση 0x63

    add r1,r3     ; πρόσθεση λιγότερο σημαντικών byte
    adc r2,r4     ; πρόσθεση περισσότερο σημαντικών byte με κρατούμενο

    st x+,r1      ; αποθήκευση αποτελέσματος στη διεύθυνση 0x64 και 0x65
    st x,r2
```

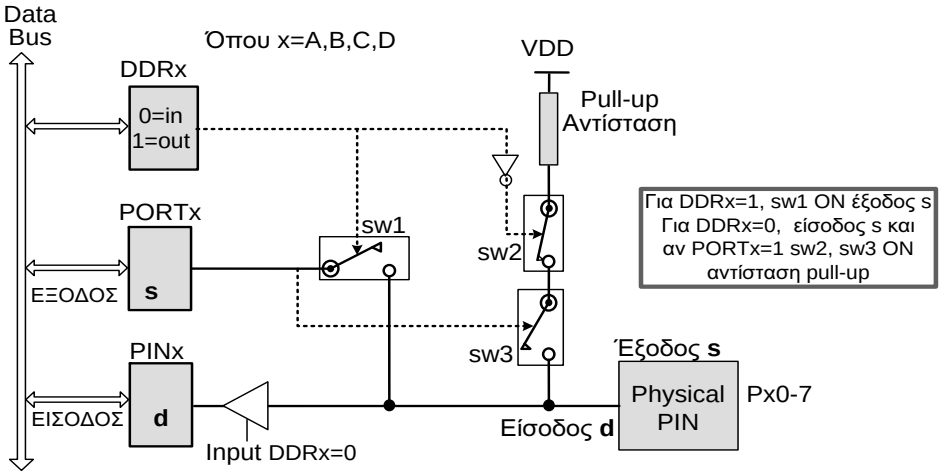
1.5.3 Θύρες εισόδου-εξόδου

Οι θύρες ενός μικροελεγκτή είναι οι πύλες επικοινωνίας της κεντρικής μονάδας επεξεργασίας με εσωτερικές και εξωτερικές μονάδες υλικού και λογισμικού. Ο επεξεργαστής επικοινωνεί με τις διάφορες μονάδες (χρονιστές, σειριακή θύρα, παράλληλη θύρα κ.ο.κ.) γράφοντας ή διαβάζοντας στις αντίστοιχες θύρες-καταχωρητές.
Έτσι εκτός από τους 32 καταχωρητές εργασίας υπάρχουν 64 θύρες-καταχωρητές, όχι όλοι διαθέσιμοι στους διάφορους τύπους μικροελεγκτών. Έχουν μια δεδομένη διεύθυνση ανεξάρτητη του μοντέλου AVR για την προσπέλασή τους από την κεντρική μονάδα επεξεργασίας. Τη διεύθυνση αυτή δεν είναι απαραίτητο να τη θυμόμαστε (π.χ. .EQU PORTB, 0x18) καθώς η πληροφορία αυτή περιέχεται στα header files (.include "m16def.inc" στην assembly ή #include <mega16.h> στη C). Ο μικροελεγκτής ATmega16 διαθέτει τέσσερις θύρες των 8 bit (PORTA, PORTB, PORTC και PORTD) που αντιστοιχούν στα pins PA0-7, PB0-7, PC0-7, PD0-7. Για την κάθε μία από αυτές τις θύρες υπάρχουν τρεις καταχωρητές που αφορούν στη λειτουργία τους και είναι οι εξής:

Καταχωρητής δεδομένων της θύρας: PORTx
Πρόκειται για έναν καταχωρητή των 8-bit με δυνατότητα εγγραφής και ανάγνωσης. Όταν η θύρα ρυθμιστεί ως έξοδος μπορούμε να γράψουμε στους ακροδέκτες της θύρας μέσω του καταχωρητή PORTx. Ενώ όταν η θύρα ρυθμιστεί ως είσοδος μπορούμε να διαβάσουμε τα δεδομένα του καταχωρητή PORTx. Η ρύθμιση των ακροδεκτών είτε ως εισόδων είτε ως εξόδων γίνεται με τη βοήθεια του καταχωρητή DDRx. Η αρχική τιμή του καταχωρητή δεδομένων είναι $00.

Καταχωρητής κατεύθυνσης της θύρας: DDRx
Ο καταχωρητής κατεύθυνσης δεδομένων (Data Direction Register) ρυθμίζει την κατεύθυνση των ακροδεκτών της θύρας PORTx. Εάν γράψουμε την τιμή ‘0’ σε κάποια από τα bits του καταχωρητή DDRx, τότε οι αντίστοιχοι ακροδέκτες της θύρας PORTx γίνονται είσοδοι. Αντίστοιχα, αν γράψουμε την τιμή ‘1’, τότε οι αντίστοιχοι ακροδέκτες της θύρας PORTx γίνονται έξοδοι.
Η αρχική τιμή του καταχωρητή κατεύθυνσης είναι $00. Επίσης, κάθε bit μπορεί να τροποποιηθεί με τις εντολές SBI, CBI.

Διεύθυνση ακροδεκτών εισόδου της θύρας (PINx)
Η διεύθυνση ακροδεκτών εισόδου (PINx) δεν είναι καταχωρητής. Αυτή η διεύθυνση επιτρέπει την πρόσβαση στις λογικές στάθμες κάθε ακροδέκτη της θύρας PORTx. Δηλαδή όταν διαβάζουμε το περιεχόμενο του PINx διαβάζουμε τις λογικές στάθμες κάθε ακροδέκτη της θύρας PORTx.
Οι πιο απλές εντολές εισόδου-εξόδου είναι η ‘in’ και ‘out’. Η ‘in’ διαβάζει την τιμή μιας θύρας ή του καταχωρητή ενός εσωτερικού περιφερειακού (χρονιστή, UART) και τη σώζει σε έναν καταχωρητή (in r16, PinD). Η ‘out’ γράφει την τιμή ενός καταχωρητή σε μια θύρα ή στον καταχωρητή ενός εσωτερικού περιφερειακού (out PortD, r16).



Σχήμα 1.17 Καταχωρητές θυρών εισόδου-εξόδου

Επίσης, υπάρχουν οι εντολές ‘sbi’ και ‘cbi’ για χειρισμό μεμονωμένων ψηφίων μιας θύρας και οι sbic , sbis για τον έλεγχο των ψηφίων μιας θύρας I/O. Αυτές οι τέσσερις εντολές δεν εφαρμόζονται σε όλους τους καταχωρητές εισόδου- εξόδου. Ο ακόλουθος κώδικας εξηγεί την εγγραφή και ανάγνωση σε μια θύρα:

1. Μετατροπή θύρας σε είσοδο
```
LDI  R12, 0b00000000        ; ή απλούστερα CLR R12
OUT  DDRD, R12              ; τα ψηφία της θύρας PORTD γίνονται είσοδοι
```

2. Μετατροπή θύρας σε έξοδο
```
LDI  R18, 0b11111111        ; ή απλούστερα SER R18
OUT  DDRB, R19              ; τα ψηφία της θύρας PORTB γίνονται έξοδοι
```

Είσοδος Δεδομένου: Ανάγνωση από θύρα εισόδου- εξόδου με την εντολή ‘in’
```
CLR R12
OUT DDRD, R12               ; Τα ψηφία της θύρας PORTD γίνονται είσοδοι.
LDI R12, 0b11111111         ; Ενεργοποίηση των  αντιστάσεων πρόσδεσης στην
OUT PORTD,R12               ; τάση τροφοδοσίας όλων των pin της θύρας PORTD
IN R12, PIND                ; Ανάγνωση των τιμών στις εισόδους της θύρας
                            ; PORTD και αποθήκευση αποτελέσματος στον R12.
```

Έξοδος Δεδομένου: Εγγραφή σε θύρα με την εντολή ‘out’
```
LDI R12, 0b00001111         ; τα bit 1ο - 4ο της θύρας PORTD γίνονται έξοδοι
OUT DDRD, R12
LDI R12, 0b00001010         ; οδήγηση των pin PD1 και PD3 σε λογική στάθμη 1
OUT PORTD, R12              ; και PD0= PD2=0
LDI R12, 0b00000101         ; οδήγηση των pin PD0 και PD2 σε λογική στάθμη 1
OUT PORTD, R12              ; και PD1= PD3=0
IN R13, PIND                ; ανάγνωση των pin εισόδου PD4 – PD7 της θύρας
                            ; PORTD και αποθήκευση αποτελέσματος στον R13
```

Παράδειγμα 1.2:
Υπολογίζουμε το τετράγωνο ενός αριθμού με βάση έναν πίνακα δεδομένων εντός της μνήμης προγράμματος. Ο αριθμός ανήκει στο διάστημα [0,15]. Ο πίνακας περιέχει τα τετράγωνα των αριθμών σε δεκαεξαδική μορφή. Η εντολή **lpm** χρησιμοποιείται για πρόσβαση στη μνήμη προγράμματος.

```
.INCLUDE "m16def.inc"
.DEF   temp=R16
start:                      ; η διεύθυνση του πίνακα στον καταχωρητή Z
    LDI ZH,HIGH(Table*2)    ; πολλαπλασιασμός επί δύο για πρόσβαση
    LDI ZL,LOW(Table*2)     ; κατά byte, όπου ZL=R30, ZH=R31
    clr temp                ; PORTD ως είσοδος
    out DDRD, temp          ; ο assembler δεν είναι case sensitive
    in temp,PIND            ; ανάγνωση αριθμού, έστω από θύρα D
    add zl,temp             ; πρόσβαση στο σωστό στοιχείο
```

```
    lpm                 ; Το περιεχόμενο της θέσης μνήμης προγράμματος που
                        ; δείχνει ο Z, φορτώνεται στον καταχωρητή R0:
                        ; Δηλαδή. R0 ←(Z)
    mov r22,r0          ; μεταφορά αποτελέσματος στον r22
    rjmp start          ; άλμα στη διεύθυνση start για επανάληψη
            ; Οι οδηγίες .DB και .DW προς το μεταφραστή χρησιμοποιούνται
            ; για την εισαγωγή ενός πίνακα δεδομένων στη μνήμη προγράμματος.
Table:                  ; Οι τιμές του πίνακα είναι οργανωμένες κατά λέξη
.DW 0x0100,0x0904,0x1910,0x3124,0x5140
.DW 0x7964,0xA990,0xE1C4
```

1.5.4 Εντολές σε επίπεδο bit και ελέγχου bit

Οι εντολές αυτές μας παρέχουν τη δυνατότητα να θέσουμε ή να μηδενίσουμε σημαίες του καταχωρητή κατάστασης, διακοπές και συγκεκριμένα bit καταχωρητών ή θυρών. Επίσης, να ολισθήσουμε κάποιον καταχωρητή μέσω κρατουμένου ή όχι. Πρόκειται για τις εξής εντολές:

Ολίσθηση καταχωρητή αριστερά ή δεξιά μέσω κρατουμένου ή όχι:

LSL Rd **:** Rd(n+1) ←Rd(n) , Rd(0) ← 0 ,C ←Rd(7)
Περιγραφή: Ολίσθηση των bits του καταχωρητή Rd μία θέση προς τα αριστερά. Το bit0 μηδενίζεται και το bit7 φορτώνεται στη σημαία C του SREG.

ROL Rd **:** Rd(0) ← C , Rd(n+1) ←Rd(n) , C ←Rd(7)
Περιγραφή: Περιστροφή των bits του καταχωρητή Rd μία θέση προς τα αριστερά μέσω της σημαίας κρατουμένου. Η σημαία C ολισθαίνει στο bit0 και το bit7 ολισθαίνει στη σημαία C.
Αντίστοιχα γίνεται η ολίσθηση και περιστροφή προς τα δεξιά με τις εντολές:
LSR Rd, **ROR** Rd.

Χειρισμός bit (σημαίας) καταχωρητή κατάστασης:
BSET s **:** SREG(s) ←1
Περιγραφή: Ενεργοποίηση σημαίας του καταχωρητή κατάστασης.

BCLR s **:** SREG(s) ←0
Περιγραφή: Μηδενισμός σημαίας του καταχωρητή κατάστασης.

Χειρισμός bit καταχωρητή εισόδου-εξόδου:
SBI A,s **:** I/O(A,s) ←1
Περιγραφή: Ενεργοποίηση του bit s ενός καταχωρητή εισόδου-εξόδου.
CBI A,s **:** I/O(A,s) ←0
Περιγραφή: Μηδενισμός του bit s ενός καταχωρητή εισόδου-εξόδου.

Χειρισμός bit καταχωρητή μέσω σημαίας T:
BST Rd,b **:** T ←Rd(b)
Περιγραφή: Αποθήκευση του bit b του καταχωρητή Rd στη σημαία T του SREG.

BLD Rd,b **:** Rd(b) ←T
Περιγραφή: Αποθήκευση της σημαίας T του SREG στο bit b του καταχωρητή Rd.

Χειρισμός σημαίας καταχωρητή κατάστασης:
SEx : x ←1
Περιγραφή: Ενεργοποίηση της σημαίας x του καταχωρητή κατάστασης.

CLx : x ←0
Περιγραφή: Μηδενισμός της σημαίας x του καταχωρητή κατάστασης.

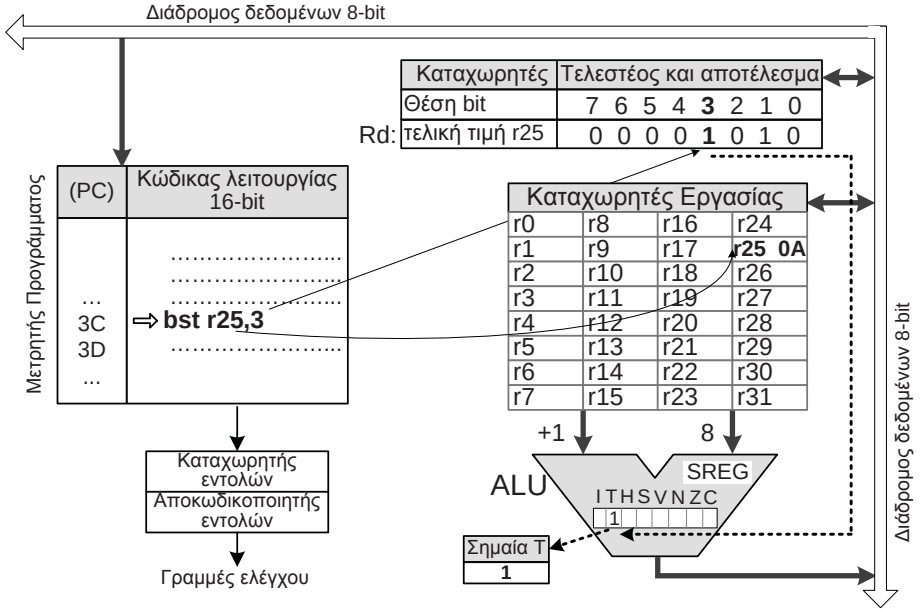
Εντολές σε επίπεδο bit και ελέγχου bit

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
SBI	P,b	Set Bit in I/O Reg	I/O(P,b) ← 1	-	2
CBI	P,b	Clear Bit in I/O Reg	I/O(P,b) ← 0	-	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 + Carry	Rd(0)←C, Rd(n+1)←Rd(n), C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right + 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)	-	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 Reg. to T	T ← Rr(b)	T	1
BLD	Rd,b	Bit load from T to Reg.	Rd(b) ← T	-	1
SEx		Set Flag	x ← 1	x	1
CLx		Clear Flag	x ← 0	x	1

Όπου x μια από τις σημαίες του καταχωρητή κατάστασης (C-N-Z-I-S-V-T-H), b το bit θέση με τιμή 0-7 (3-bit) ενός καταχωρητή ή μιας θύρας, s το bit θέσης 0-7 (3-bit) του καταχωρητή κατάστασης SREG, k μια σταθερή διεύθυνση και P διεύθυνση καταχωρητή θύρας εισόδου-εξόδου.

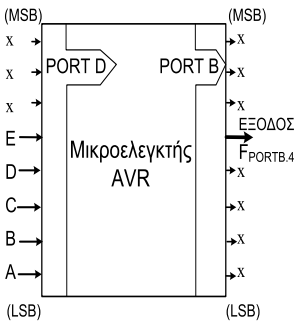
Παράδειγμα εγγραφής σε θύρα με τις εντολές ‘sbi’ και ‘cbi’:

```
SBI DDRD,2      ; το 2ο ψηφίο της θύρας PORTD γίνεται έξοδος
SBI PORTD,2     ; οδήγηση του 2ου ψηφίου σε λογική στάθμη 1
CBI PORTD,2     ; οδήγηση του 2ου ψηφίου σε λογική στάθμη 0
```



Σχήμα 1.18 Παράδειγμα εντολής σε επίπεδο bit

Παράδειγμα 1.3 : Υλοποίηση λογικών συναρτήσεων με τη χρήση εντολών σε επίπεδο bit και θυρών εισόδου – εξόδου.



Στο παράδειγμα αυτό γίνεται η παρουσίαση εντολών που επιτρέπουν το χειρισμό σε επίπεδο bit και οι οποίες είναι χρήσιμες για την επίλυση προβλημάτων συνδυαστικής λογικής. Πιο συγκεκριμένα, υποθέτουμε ότι οι λογικές μεταβλητές A, B, C, D και E βρίσκονται συνδεδεμένες στα 5 πρώτα LSB της θύρας PORTD. Να δοθεί το assembly πρόγραμμα που υλοποιεί τη λογική εξίσωση:

$F_{PORTD.4} = (A+B')\cdot(C\cdot D+E)$

Το ένα bit της εξόδου παρέχεται από το 5^{ο} bit της θύρας PORTB.

```
.include "m16def.inc"
.DEF A=r16              ; δήλωση καταχωρητών
.DEF BN=r17              ; συμπλήρωμα
.DEF C=r18
.DEF D=r19
.DEF E=r20
.DEF temp=r21
.cseg
.org 0                  ; διεύθυνση εκκίνησης
    clr temp            ; θύρα D ως είσοδος
    out DDRD,temp
```

```
    ser temp
    out PORTD,temp    ; pull-up θύρας D
    out DDRB,temp     ; θύρα B ως έξοδος
again:
    in temp, PIND     ; ανάγνωση ακροδεκτών PORTD
    mov A, temp       ; το A στο LSB του καταχωρητή A
    lsr temp
    mov BN, temp      ; το B στο LSB του καταχωρητή BN
    com BN            ; συμπλήρωμα B
    lsr temp
    mov C, temp       ; το C στο LSB του καταχωρητή C
    lsr temp
    mov D, temp       ; το D στο LSB του καταχωρητή D
    lsr temp
    mov E, temp       ; το E στο LSB του καταχωρητή E
    or A, BN                ; A=A+B΄
    and C,D                 ; C=C·D
    or C,E                  ; C=C·D+E
    and A,C           ; υλοποίηση συνδυαστικής λογικής A=(A+B΄)·(C·D+E)
    andi A, 1         ; απομόνωση του LSB
    lsl  A (×4)       ; 4 ολισθήσεις αριστερά για να έρθει το αποτέλεσμα στη
    out PORTB,A       ; σωστή θέση. Έξοδος αποτελέσματος
    rjmp again        ; άλμα στη διεύθυνση again για επανάληψη
```

Β΄ Τρόπος: Εναλλακτικά η υλοποίηση με βάση τη σχέση $F_{PORTD.4}$ ={A΄·B +C΄·E΄+ D΄·E΄}΄ που είναι ισοδύναμη με την προηγούμενη μορφή βάσει του θεωρήματος De Morgan, μπορεί να γίνει με τον εξής τρόπο:

```
AGAIN:
    clr r22
    in temp, PIND     ; ανάγνωση ακροδεκτών PORTD
    mov r16, temp     ; το A στο LSB του καταχωρητή A
    andi r16, 0x03    ; απομόνωση των A και B
    cpi r16,0x02      ; συγκρίνεται για A=0 και B=1. Μόνο τότε A΄·B=1
    breq ZERO         ; Αν ισχύει F=0, άλμα στη διεύθυνση ZERO
    mov r16, temp     ; Αλλιώς ελέγχω τον επόμενο όρο C΄·E΄
    andi r16, 0x14    ; απομόνωση των C και E. Αν C=E=0, C΄·E΄=1
    breq ZERO         ; Αν ισχύει F=0, άλμα στη διεύθυνση ZERO
    mov r16, temp     ; Αλλιώς ελέγχω αν ο τελευταίος όρος D΄·E΄=1
    andi r16, 0x18    ; απομόνωση των D και E. Αν D=E=0, D΄·E΄=1
    breq ZERO         ; Αν ισχύει F=0, άλμα στη διεύθυνση ZERO
    ldi r22, 0x10     ; F=1. Έξοδο αποτελέσματος στο PORT.4 =1
ZERO:                 ; Αν είναι από άλμα F=0. Ακολουθεί το κοινό τμήμα
    out PORTB,r22     ; έξοδος αποτελέσματος
    rjmp AGAIN        ; άλμα στη διεύθυνση AGAIN για επανάληψη
```

Η υλοποίηση αυτή είναι αποδοτικότερη, όταν στη λογική συνάρτηση Boole έχουμε λίγους όρους (λογικά AND ή OR) και πολλές μεταβλητές.

1.5.5 Εντολές ελέγχου ροής προγράμματος και διακλάδωσης

Σ' αυτήν την κατηγορία ανήκουν οι εντολές άλματος, παράκαμψης, διακλάδωσης και κλήσης ρουτινών. Πρόκειται για εντολές που αλλάζουν την κανονική ακολουθιακή ροή του προγράμματος. Δηλαδή, όταν εκτελείται μια τέτοια εντολή ο έλεγχος μεταφέρεται σε άλλο σημείο του προγράμματος, αντί να εκτελεστεί η επόμενη εντολή. Το σημείο αυτό μπορεί να είναι σε μια διεύθυνση της μνήμης προγράμματος, η οποία καθορίζεται από το όρισμα της εντολής. Συγκεκριμένα, ο μετρητής προγράμματος (PC-Program Counter) θα πάρει την επιθυμητή τιμή είτε από τον καταχωρητή Z είτε από μια ετικέτα (label).

Οι εντολές παράκαμψης και διακλάδωσης είναι υπό συνθήκη. Αυτό σημαίνει ότι εξετάζεται η τιμή κάποιου bit για να καθοριστεί αν πρέπει να μεταφερθεί ο έλεγχος ή όχι. Συγκεκριμένα, οι εντολές παράκαμψης εξετάζουν την τιμή ενός bit κάποιου καταχωρητή ή μιας θύρας εισόδου-εξόδου, ενώ οι εντολές διακλάδωσης εξετάζουν μία από τις επτά σημαίες του καταχωρητή κατάστασης. Ο μικροελεγκτής έχει μια πληθώρα εντολών διακλάδωσης, γεγονός που συμβάλλει στην αποδοτικότητα και ευελιξία του κώδικα. Οι εντολές παράκαμψης εξετάζουν την τιμή ενός bit κάποιου καταχωρητή ή μιας θύρας εισόδου-εξόδου, για να καθορίσουν αν πρέπει να εκτελεστεί άλμα ή όχι. Οι σημαίες δεν επηρεάζονται.

Παρακάτω, παρατίθενται οι προαναφερθείσες εντολές ανά κατηγορία:

Εντολές παράκαμψης

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) *	-	1/ 2/ 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) *	-	1/ 2/ 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) *	-	1/ 2/ 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) *	-	1/ 2/ 3

* Όταν ισχύει η συνθήκη φορτώνεται η νέα τιμή του Μετρητή Προγράμματος: PC ← PC + 2 or 3

SBRC Rr,b **:** Αν Rr(b) = 0 τότε PC ←PC + 2 (ή 3) αλλιώς PC ←PC + 1

Περιγραφή: Αν το bit (b) του καταχωρητή είναι 0 (Rr(b) = 0) τότε παρακάμπτεται η επόμενη εντολή.

SBIC A,b **:** Αν I/O(A,b) = 0 τότε PC←PC + 2 (ή 3) αλλιώς PC ←PC + 1

Περιγραφή: Αν το bit (b) του καταχωρητή εισόδου-εξόδου είναι 0 (A(b) = 0) τότε παρακάμπτεται η επόμενη εντολή.

Αντίστοιχα, οι εντολές SBRS, SBIS παρακάμπτουν την επόμενη εντολή αν το bit (b) είναι μονάδα.

Οι εντολές διακλάδωσης εξετάζουν μία από τις επτά σημαίες του καταχωρητή κατάστασης για να αποφασίσουν αν πρέπει να εκτελεστεί άλμα ή όχι. Οι σημαίες δεν επηρεάζονται.

BRBC s,k **:** Αν SREG(s) = 0 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Αν το bit s του καταχωρητή κατάστασης είναι 0 (SREG(s) = 0) τότε εκτελείται άλμα στη διεύθυνση της μνήμης προγράμματος που βρίσκεται k θέσεις μετά σε σχέση με το μετρητή προγράμματος.

BRNE k**:** Αν Rd ≠ Rr (Z = 0) τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1

Περιγραφή: Σχετικό άλμα σε περίπτωση ανισότητας. Όταν η σημαία μηδενός ισούται με 0 (Z = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 1,k.

BRCC k **:** Αν C = 0 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία κρατουμένου ισούται με 0 (C = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 0,k.

BRSH k **:** Αν Rd>=Rr (C = 0) τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση δώσει μεγαλύτερο ή ίσο (C = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 0,k.

BRPL k **:** Αν N = 0 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση δώσει (N = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 2, k.

BRGE k **:** Αν Rd>=Rr (S=N⊕V = 0) τότε PC ←PC + k + 1, αλλιώς PC←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση προσημασμένου δώσει μεγαλύτερο ή ίσο (S = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 4,k.

BRHC k **:** Αν H = 0 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία δεκαδικού κρατουμένου ισούται με 0 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 5,k.

BRTC k **:** Αν T = 0 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία T ισούται με 0 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 6, k.

Εντολές διακλάδωσης

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
BRBS	s,k	Branch if Status Flag Set	if (SREG(s) = 1) *	-	1/2
BRBC	s,k	Branch if Status Flag Cleared	if (SREG(s) = 0) *	-	1/2
BREQ	k	Branch if Equal	if (Z = 1) then *	-	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then *	-	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then *	-	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then *	-	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then *	-	1/2
BRLO	k	Branch if Lower	if (C = 1) then *	-	1/2
BRMI	k	Branch if Minus	if (N = 1) then *	-	1/2
BRPL	k	Branch if Plus	if (N = 0) then *	-	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N⊕V= 0) then *	-	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N⊕V= 1) then *	-	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then *	-	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then *	-	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then *	-	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then *	-	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then *	-	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then *	-	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then *	-	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then *	-	1/2

* Όταν ισχύει η συνθήκη φορτώνεται η νέα τιμή του Μετρητή Προγράμματος: PC ← PC + k +1

BRVC k **:** Αν V = 0 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία υπερχείλισης V ισούται με 0 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 3,k.

BRID k **:** Αν I=0 τότε PC← PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία ολικών διακοπών ισούται με 0(I=0), δηλαδή οι διακοπές είναι απενεργοποιημένες, τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 7, k.
Ανάλογα, οι εντολές BRBS s,k , BREQ k , BRLO k , BRMI k , BRLT k , BRHS k , BRVS k , BRIE k , BRCS k όταν η αντίστοιχη σημαία είναι μονάδα.

Εντολές σύγκρισης

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	-	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 Reg with Immediate	Rd < K	Z,N,V,C,H	1

Οι εντολές σύγκρισης μας παρέχουν τη δυνατότητα να συγκρίνουμε δύο καταχωρητές ή έναν καταχωρητή και μια σταθερά.
CPSE Rd,Rr **:** Αν Rd = Rr τότε PC←PC + 2 (ή 3) αλλιώς PC←PC + 1
Περιγραφή: Σύγκριση μεταξύ των καταχωρητών Rd,Rr. Σε περίπτωση ισότητας παρακάμπτεται η επόμενη εντολή.
CPI Rd,K **:** Rd - K
Περιγραφή: Σύγκριση με υπολογισμό της διαφοράς και χρήση κρατουμένου μεταξύ του καταχωρητή Rd και μιας σταθεράς.
Ανάλογα υπάρχει δυνατότητα σύγκρισης με χρήση κρατουμένου (CPC).

Εντολές άλματος- ρουτινών

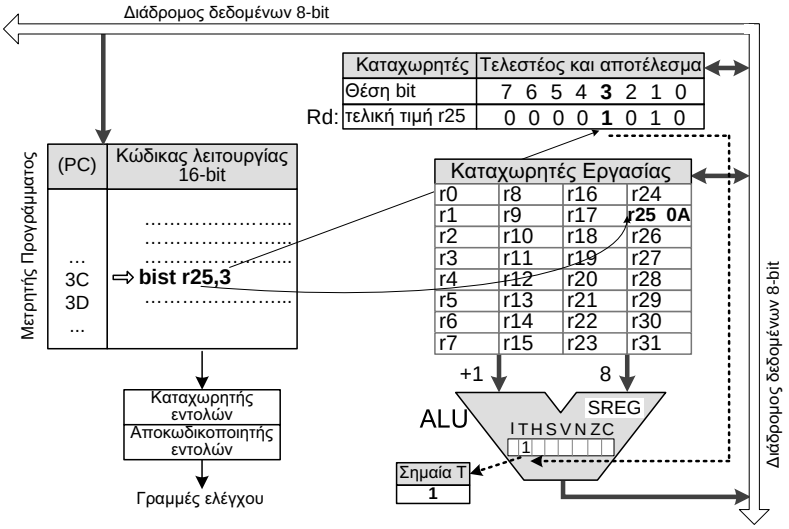
Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
RJMP	k	Relative Jump	PC ← PC + k + 1	-	2
IJMP		Indirect Jump to (Z)	PC ← Z	-	2
JMP	k	Direct Jump	PC ← k	-	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	-	3
ICALL		Indirect Call to (Z)	PC ← Z	-	3
CALL	k	Direct Subroutine Call	PC ← k	-	4
RET		Subroutine Return	PC ← STACK	-	4
RETI		Interrupt Return	PC ← STACK	I	4

Οι επόμενες εντολές μας παρέχουν τη δυνατότητα να εκτελέσουμε άλμα σε επιθυμητή διεύθυνση, να καλέσουμε μια ρουτίνα και να επιστρέψουμε από αυτήν.

RJMP k **:** PC←PC + k + 1
Περιγραφή: Σχετικό άλμα σε μια διεύθυνση της μνήμης προγράμματος μεταξύ PC – 2k +1 και PC + 2k .

RCALL k **:** PC ←PC + k + 1
Περιγραφή: Σχετική κλήση ρουτίνας σε μια διεύθυνση μεταξύ PC – 2k +1 και PC + 2k. Η διεύθυνση επιστροφής αποθηκεύεται στη στοίβα.

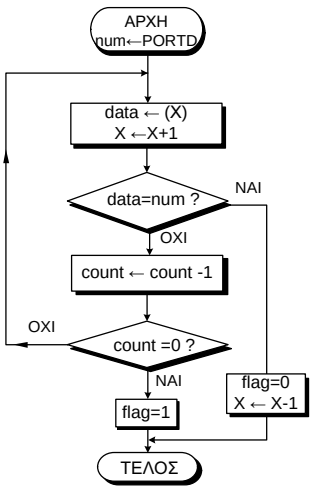
RET: PC ←Stack **Περιγραφή**: Επιστροφή από υπορουτίνα. Η διεύθυνση επιστροφής φορτώνεται από τη στοίβα.
Στο σχήμα 1.19 απεικονίζεται η διαδικασία κλήσης ρουτίνας.



Σχήμα 1.19 Παράδειγμα εντολής κλήσης ρουτίνας

Παραδείγματα εντολών ελέγχου ροής προγράμματος και διακλάδωσης

Παράδειγμα 1.4:
Το πρόγραμμα βρίσκει αν η τιμή (num) που δίνεται από το PORTD είναι σε περιοχή της μνήμης δεδομένων SRAM που η πρώτη διεύθυνση βρίσκεται στον καταχωρητή X= r27:r26 και το πλήθος τους, στον καταχωρητή r21 (count). Αν βρει τη τιμή, θέτει στον καταχωρητή r22 τιμή 1 και τη διεύθυνση που βρήκε το δεδομένο (μέσω του καταχωρητή δείκτη X), αλλιώς θέτει 0 στον καταχωρητή r22 (flag).

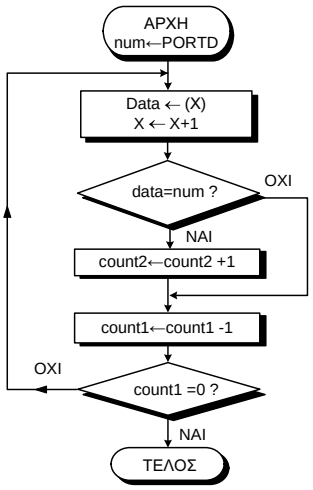



```
.INCLUDE "m16def.inc"
.DEF   flag=r22
.DEF   count=r21
.DEF   num=r20
.DEF   data=r19
```

```
start:
    clr num             ; το PORTD ορίζεται ως είσοδος
    out DDRD,num
    in num,PIND         ; ανάγνωση αριθμού από PORTD
loop:
    ld data,x+          ; φόρτωση θέσης X μνήμης δεδομένων και αύξηση
                        ; δείκτη για πρόσβαση στον επόμενο κύκλο σε επόμενη θέση
    cp data,num         ; σύγκριση του εισαγόμενου αριθμού με θέση μνήμης
    breq found          ; η τιμή βρέθηκε στον πίνακα

not_found:
    dec count           ; μειώνω το μετρητή
    brne loop           ; έλεγχος για να μην περάσουμε το δοσμένο πλήθος
    clr flag            ; αν η τιμή δεν βρέθηκε μηδενίζουμε τον r22
    rjmp end
found:                  ; η τιμή βρέθηκε στον πίνακα οπότε
    ldi flag,1          ; θέτουμε τον r22 και επιστρέφουμε την αντίστοιχη
    sbiw r26,1          ; διεύθυνση στον X= r27 : r26 αφού το μειώσουμε κατά 1
 end:
```

Παράδειγμα 1.5:

Να υπολογιστεί πόσες φορές εμφανίζεται ένας χαρακτήρας, που η τιμή του δίνεται από το PORTD, σε περιοχή της μνήμης δεδομένων SRAM που η πρώτη διεύθυνση βρίσκεται στον καταχωρητή X=r27:r26 και το πλήθος τους, στον καταχωρητή r21 (count1). Το πλήθος των εμφανίσεων του χαρακτήρα επιστρέφεται μέσω του καταχωρητή r22 (count2).



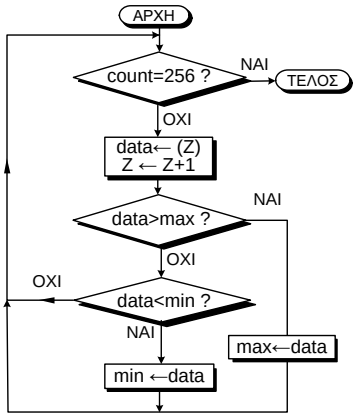

```
.INCLUDE "m16def.inc"
.DEF   count2=r22
.DEF   count1=r21
.DEF   num=r20
.DEF   data=r19
```

```
start:
    clr num             ; το PORTD ορίζεται ως είσοδος
    out DDRD,num
    in num,PIND         ; ανάγνωση αριθμού από PORTD
loop:
    ld data,x+          ; φόρτωση θέσης X μνήμης δεδομένων και

; αύξηση δείκτη (X←X+1) για πρόσβαση στον επόμενο κύκλο σε επόμενη θέση
```

```
    cp data,num          ; σύγκριση του εισαγόμενου αριθμού με θέση μνήμης
    brne not_found       ; η τιμή δεν βρέθηκε στον πίνακα
    inc count2

not_found:
    dec count1           ; μειώνω το μετρητή
    brne loop            ; έλεγχος για να μην περάσουμε το δοσμένο πλήθος

end:
```

Παράδειγμα 1.6:

Το πρόγραμμα εντοπίζει το μέγιστο και τον ελάχιστο σε περιοχή της μνήμης προγράμματος όπου έχουν αποθηκευθεί συνολικά 256 δεδομένα (των 8 bit).




```
.INCLUDE "m16def.inc"
.DEF   count=R18
.DEF   data=R17
.DEF   max=R16
.DEF   min=R15
```

```
begin:
    ldi ZH,HIGH(Array *2)
    ldi ZL,LOW(Array *2) ; η διεύθυνση του πίνακα στον καταχ. Z
    lpm data,z+          ; φόρτωση μνήμης προγράμματος
    mov max, data
    mov min, data
    clr count            ; το count γίνεται 0
search:
 inc count               ; έλεγχος αν το πλήθος είναι 256
    breq end

    lpm data,z+          ; πρόσβαση σε επόμενη θέση και αύξηση δείκτη Z
    cp data,max          ; σύγκριση με μέγιστο
    brge new_max         ; άλμα εφόσον βρεθεί μεγαλύτερη τιμή

    cp data,min          ; σύγκριση με ελάχιστο
    brlo new_min         ; άλμα εφόσον βρεθεί μικρότερη τιμή
    rjmp search          ; επιστροφή στην αναζήτηση
new_max:
    mov max, data        ; σώσιμο στο max της νέας μέγιστης τιμής
    rjmp search          ; επιστροφή στην αναζήτηση
```

```
new_min:
      mov min, data       ; σώσιμο στο min της νέας ελάχιστης τιμής
      rjmp search         ; επιστροφή στην αναζήτηση
end:
Array:                    ; εισαγωγή πίνακα δεδομένων στη μνήμη προγράμματος
.DW 0x0908,0x0706,0x1713,0x3326
.DW 0x3042,0x7061,0x7205,0x7803
```

Παράδειγμα 1.7:
Στο παράδειγμα αυτό επιδεικνύεται η χρήση της LPM για ανάγνωση bytes από πίνακα δεδομένων στο τμήμα κώδικα (code segment). Συγκεκριμένα, διαβάζονται διακόπτες που υποθέτουμε ότι είναι συνδεδεμένοι στο PORTD και ανάβουν τα leds (PORTB ως έξοδος των leds) ως εξής: sw0 ανάβει led0, sw1ανάβει led0 και led1, sw2 ανάβει led0 led1 led2 κοκ. Η αντιστοιχία αυτή υλοποιείται με τη δημιουργία κατάλληλου πίνακα δεδομένων στο τμήμα κώδικα. Μια απλούστερη υλοποίηση είναι να κάνουμε την πράξη led=(sw-1)·2.

```
.INCLUDE "m16def.inc"  ; δήλωση μικροελεγκτή
.LIST
.DEF   reg=R0          ; η εντολή LPM χειρίζεται τον καταχωρητή R0
.DEF   temp=R16
start:
    clr temp           ; PORTD ως είσοδος των διακοπτών
    out DDRD,temp
    dec temp           ; φόρτωση τιμής 0xFF στον καταχωρητή temp
    out DDRB,temp      ; PORTB ως έξοδος των Leds
    out PORTD,temp     ; ενεργοποίηση εσωτερικών αντιστάσεων πρόσδεσης
; χρήση καταχωρητή δείκτη Z: {ZL (R30) και ZH (R31)}
; που έχει οριστεί στο αρχείο m16def.inc

loop:
    ldi ZL,LOW(array)  ; ο Z δείχνει στο 1ο byte (FF) της λίστας
    ldi ZH,HIGH(array)
    in   temp,PIND     ; ανάγνωση διακοπτών
    cpi temp,0xFF      ; αν όλοι οι διακόπτες off, τότε όλα τα Leds off
    breq diabasma      ; άλμα στο diabasma αν όλοι οι διακόπτες είναι off
inc_pointer:           ; Προσοχή στην αρνητική λογική των διακοπτών!
    adiw ZL, 1         ; Αυξάνεται το Z μέχρι να βρεθεί η πρώτη μονάδα
    ror temp           ; καθώς ολισθαίνει δεξιά η τιμή των διακοπτών.
    brlo inc_pointer   ; Όσο το C=1 έχουμε άλμα και αύξηση του Z.
diabasma:              ; Όταν C=0 γίνεται ανάγνωση του byte της λίστας.
    lpm                ; Η θέση που δείχνει ο Z  εγγράφεται στον R0.
    out PORTB,reg      ; Μεταφορά του δεδομένου στα leds.
    rjmp loop
; Η χρήση του  .DB xx στη μνήμη προγράμματος πάντα προσθέτει ένα μηδενικό byte.
; Γι' αυτό προτιμάται η χρήση του .DW ;xx,yy (2 δεδομένα ενώνονται σε μια λέξη).
```

```
; Αντίστοιχα ισχύει σε κείμενο χαρακτήρων, δηλ. .DB "atmel" αποδοτικότερα
; εγγράφεται ως  .DW.  Αν ο αριθμός των bytes ή χαρακτήρων είναι περιττός θα
; πρέπει να συμπληρωθεί η λίστα με ένα μηδενικό.

 ; Πίνακας με τους συνδυασμούς των leds, όπου κάθε byte αντιστοιχεί σε ένα
 ; διακόπτη. Οι τιμές τοποθετούνται κατά λέξεις για αποδοτικότερη χρήση της μνήμης
table:
.DW   0xFEFF ; 1 Led, 0 Leds
.DW   0xF8FC ; 3 Leds, 2 Leds
.DW   0xE0F0 ; 5 Leds, 4 Leds
.DW   0x80C0 ; 7 Leds, 6 Leds
; Όταν προσθέτουμε μια λέξη όπως FEFFh σε λίστα, το 1ο byte της λίστας που
; λαμβάνεται μέσω της εντολής LPM είναι το LSB (FFh), και όχι το MSB (FEh).
; Η πρόσβαση στη λίστα γίνεται κατά byte, ενώ οι διευθ. οργανώνονται κατά λέξεις
.EQU   array=table*2
```

Παράδειγμα 1.8:
Το παρακάτω πρόγραμμα αναβοσβήνει τα leds παρεμβάλλοντας χρονική καθυστέρηση σε κάθε κατάσταση, για να γίνουν ορατές οι αλλαγές:

```
.INCLUDE "m16def.inc"   ; δηλώνουμε μικροελεγκτή
.DEF reg = R16
.def Delay = r17        ; καταχωρητής μεταβλητής Delay
.def Delay2 = r18       ; καταχωρητής μεταβλητής Delay2

main:
    ldi reg,0b11111111
    out DDRB,reg        ; ορίζουμε το PORTB ως έξοδο

DLY:
    dec Delay           ; Καθυστέρηση για να γίνουν ορατές οι αλλαγές
    brne DLY
    dec Delay2
    brne DLY
    ldi reg,0x00        ; (0=on, 1=off).
    out PORTB,reg       ; άναμμα των leds
DLY2:
    dec Delay           ; Καθυστέρηση για να γίνουν ορατές οι αλλαγές
    brne DLY2
    dec Delay2
    brne DLY2
    ldi reg,0xFF
    out PORTB,reg       ; σβήσιμο των leds και ατέρμων βρόχος
    rjmp main           ; για συνεχή επανάληψη της διαδικασίας
```

Παρατηρήσεις:

1. Αν υποθέσουμε ότι το ρολόι του επεξεργαστή είναι 4 MHz να υπολογιστεί η μέγιστη χρονοκαθυστέρηση (για Delay= Delay2=0) που προκαλείται από το τμήμα κώδικα (ο διπλός βρόχος από DLY έως brne DLY) του παραπάνω προγράμματος. Να παρατηρηθεί ότι μετά τον τερματισμό της εκτέλεσης του, οι δυο παράμετροι Delay, Delay2 γίνονται 0, γι' αυτό και στο πρόγραμμα δεν υπάρχουν εντολές αρχικοποίησής τους.
2. Για να αποφύγουμε τον παραπάνω διπλό βρόχο χρονοκαθυστέρησης μπορούμε να αξιοποιήσουμε την εντολή sbiw R_d, 1 που αντιστοιχεί σε εντολή μείωσης κατά 1 του διπλού καταχωρητή R_{d+1}:R_d. Με κατάλληλη αρχική τιμή του διπλού καταχωρητή να δημιουργηθεί ρουτίνα χρονοκαθυστέρησης π.χ. 50msec απλού βρόχου.
3. Για να είναι πιο δομημένα τα προγράμματά μας είναι χρήσιμο να δημιουργηθεί κατάλληλη μακροεντολή ή ρουτίνα καθυστέρησης που μέσω παραμέτρων να ρυθμίζεται ο χρόνος της.
4. Η παραπάνω τεχνική δεν είναι ο καλύτερος τρόπος δημιουργίας καθυστέρησης, γιατί εξαιτίας της σπαταλάται, χωρίς να χρειάζεται, επιπλέον υπολογιστική ισχύς. Αποδοτικότερος τρόπος είναι η χρήση των χρονιστών/μετρητών που παρουσιάζονται στο Κεφ. 2, το οποίο αφορά στις περιφερειακές μονάδες του μικροελεγκτή AVR.

Παράδειγμα 1.9:

Στο παράδειγμα γίνεται χειρισμός στα leds με βάση τη τιμή των switches που συνδέονται στις θύρες PORTB και PORTD αντίστοιχα. Η συμπεριφορά των leds καθορίζεται με βάση ποιος διακόπτης πιέζεται ως εξής:

PORTD-switches	**PORTB - LEDS**
pin0=1	Μέτρηση LEDS κάτω
pin1=1	Μέτρηση LEDS πάνω
pin2=1	Ολίσθηση LEDS μία θέση δεξιά
pin3=1	Ολίσθηση LEDS μία θέση αριστερά
pin4=1	Αντιστροφή LEDS
pin5=1	Συμπλήρωμα ως προς 2 των LEDS
pin6 =1	Εναλλαγή 4 LSB με τα 4 MSB
pin7= 1	Τα 4 LSB ON και τα 4 MSB OFF

Υποθέτουμε ότι δεν πιέζονται ταυτόχρονα περισσότεροι του ενός διακόπτες. Επίσης έχει τεθεί αρχική κατάσταση OFF για όλα τα LEDS.

```
.include "m16def.inc"
.def Temp =r16          ; προσωρινός καταχωρητής
.def Delay =r17         ; καταχωρητής μεταβλητής Delay
.def Delay2 =r18        ; καταχωρητής μεταβλητής Delay2

RESET:
    clr Temp            ; αρχικοποίηση του PORTD
    out DDRD,Temp       ; ως θύρας εισόδου
    ser Temp            ; αρχικοποίηση του PORTB
    out DDRB,Temp       ; ως θύρας εξόδου
    out PORTD,Temp      ; ενεργοποίηση των αντιστάσεων πρόσδεσης
```

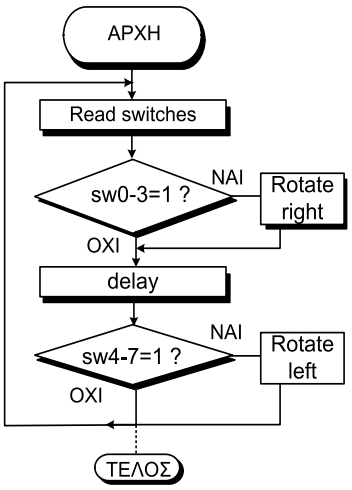
```
                    ; Έλεγχος input/output
LOOP:
    sbic PIND,0x00  ; Αν (Port D, pin0 = 1)
    inc Temp                ; τότε μέτρηση LEDS μια μονάδα προς τα κάτω
    sbic PIND,0x01  ; Αν (Port D, pin1 = 1)
    dec Temp                ; τότε μέτρηση LEDS μια μονάδα προς τα πάνω
    sbic PIND,0x02  ; Αν (Port D, pin2 = 1)
    ror Temp                ; τότε ολίσθηση LEDS μία θέση δεξιά
    sbic PIND,0x03  ; Αν (Port D, pin3 = 1)
    rol Temp                ; τότε ολίσθηση LEDS μία θέση αριστερά
    sbic PIND,0x04  ; Αν (Port D, pin4 = 1)
    com Temp                ; τότε αντιστροφή LEDS
    sbic PIND,0x05  ; Αν (Port D, pin5 = 1)
    neg Temp                ; τότε αντιστροφή LEDS και πρόσθεση 1
    sbic PIND,0x06  ; Αν (Port D, pin6 = 1) τότε
    swap Temp               ; εναλλαγή των τμημάτων  high και low των LEDS
    sbic PIND,0x07  ; Αν (Port D, pin7 = 1)
    ldi Temp,0xF0
    out PORTB,Temp  ; ενημέρωση LEDS

DLY:
    dec Delay       ; καθυστέρηση για να γίνουν ορατές οι αλλαγές
    brne DLY
    dec Delay2
    brne DLY
    rjmp LOOP       ; επανάληψη βρόχου
```

Ερώτημα 1ο: Να συμπληρωθεί το προηγούμενο πρόγραμμα έτσι ώστε να γίνεται αναγνώριση ότι αφήνουμε το εκάστοτε πλήκτρο για να γίνει νέο πάτημα.

Ερώτημα 2ο: Αν επιτρέψουμε τη δυνατότητα να πατηθούν δυο διακόπτες ταυτόχρονα (χωρίς να αποκλείουμε μια μικρή χρονική διαφορά < 20ms), πώς μπορούμε να αναγνωρίσουμε το συνδυασμό για να γίνει κάποια ενέργεια π.χ. μέτρηση πάνω στα LEDS; Η υλοποίηση πρέπει να επιτρέπει την ασφαλή επανάληψη της διαδικασίας μετά από χρόνο μεγαλύτερο του 1 sec. Η ενέργεια να γίνεται στο πάτημα και όχι όταν αφήνουμε τα πλήκτρα. Υποθέστε 2 διαφορετικές περιπτώσεις:
α. Να γίνεται αναγνώριση ότι αφήνουμε τα πλήκτρα για να αναγνωριστεί νέο πάτημα.
β. Δε γίνεται αναγνώριση ότι αφήνουμε τα πλήκτρα οπότε αν τα κρατάμε πατημένα μετά από χρόνο 1 sec θα πρέπει να θεωρηθεί ως $2^{ο}$ πάτημα.

Παράδειγμα 1.10:

Με το πάτημα ενός διακόπτη (ή και περισσότερων) από τα 4 δεξιότερα switches (PORTD) ανάβει το αντίστοιχο led (PORTB) και όσο είναι πατημένο να το περιστρέφει κυκλικά προς τα δεξιά.
Αντίστοιχα, πάτημα ενός διακόπτη από τα 4 αριστερότερα να προκαλεί αριστερή περιστροφή.
Όταν δεν είναι πατημένο κανένα switch, να σβήνουν όλα τα leds.




```
.include "m16def.inc"
.def temp=r16
.def tempN=r17
.def Delay1=r18
.def Delay2=r19
```

```
RESET:
    ldi temp,LOW(RAMEND) ; αρχικοποίηση δείκτη στοίβας
    out SPL,temp
    ldi temp,HIGH(RAMEND)
    out SPH,temp
    clr temp
    out DDRD,temp      ; DDRD ως θύρα εισόδου
    ser temp
    out DDRB,temp      ; DDRB ως θύρα εξόδου
    out PORTD,temp     ; τίθενται οι αντιστάσεις πρόσδεσης (pull-up)

ARXH:
    in temp,PIND
    mov tempN,temp     ; αντίγραφο των διακοπτών σε θετική λογική
    com tempN          ; στον καταχωρητή tempN
RIGHT:
    andi tempN, 0x0F   ; Αν οποιοσδήποτε εκ των διακοπτών sw0-3
    breq CONT          ; πατηθεί έχουμε ≠0 και δεν εκτελειται άλμα

 ; Αν θέλουμε ο έλεγχος να γίνει σε αρνητική λογική τότε έχουμε τον κώδικα:
 ; ori temp, 0xF0     ; Αν πατηθεί ένα από τα sw0-3 έχουμε
 ; cpi temp, 0xFF     ; κάποιο 0 και η σύγκριση τότε δε δίνει
 ; breq CONT          ; ισότητα οπότε δεν εκτελείται άλμα
                      ; αλλά έχουμε περιστροφή δεξιά.
    ror temp
    out PORTB,temp
CONT:
   rcall DELAY
```

```
LEFT:
    andi tempN, 0xF0    ; Αν οποιωσδήποτε των διακοπτών sw4-7
    breq RIGHT          ; πατηθεί έχουμε ≠0 και δεν εκτελείται άλμα
    rol temp            ; αλλά έχουμε περιστροφή αριστερά.
    out PORTB,temp
    rjmp ARXH

DELAY:
d1:                     ; Ρουτίνα χρονοκαθυστέρησης όπου
    dec Delay2          ; οι 2 πρώτες εντολές εκτελούνται
    brne d1             ; Delay2* Delay1 φορές
    dec Delay1
    brne d1
    ret
```

Ζητούμενο: Η παραπάνω άσκηση να συμπληρωθεί, έτσι ώστε αν είναι πατημένοι διακόπτες και από τις δύο τετράδες, να ανάβουν όλα τα leds. Να δοθεί προσοχή στο γεγονός ότι είναι πρακτικά αδύνατο να πατηθούν δυο διακόπτες ταυτόχρονα. Συνεπώς, πρέπει να δοθεί ένας χρόνος της τάξεως των 10-20 msec για τον έλεγχο των δυο τετράδων.

1.5.6 Εντολές ελέγχου μικροελεγκτή

Πρόκειται για εντολές που ελέγχουν τη λειτουργία του μικροελεγκτή, π.χ. προκαλώντας επανεκκίνηση του watchdog timer, ή κάποια λειτουργία ηρεμίας, ανενεργής κατάστασης ή διακοπή. Οι σημαίες μένουν ανεπηρέαστες.

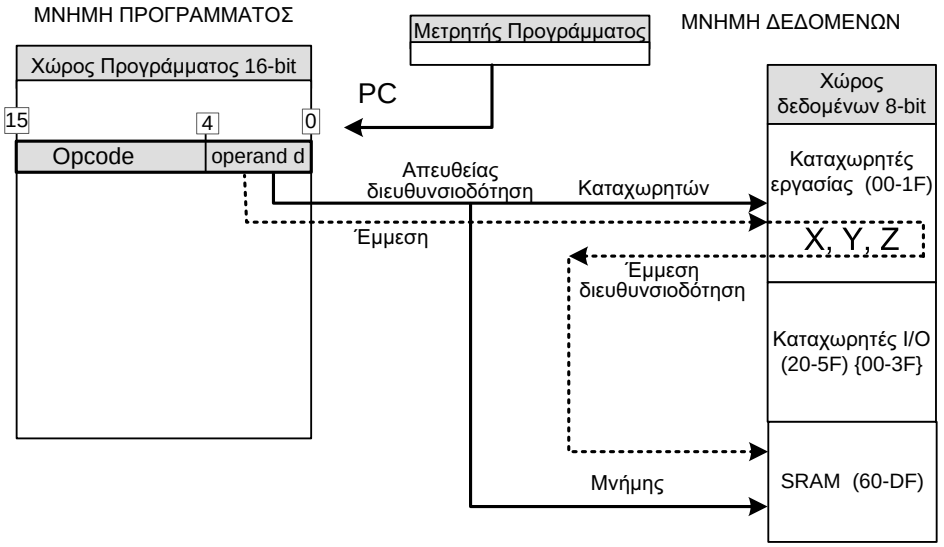
Εντολές ελέγχου MCU

Μνημονικό και Ορίσματα		Περιγραφή	Λειτουργία	Σημαίες και Κύκλοι	
NOP	-	No Operation		-	1
SLEEP	-	Sleep	Sleep function	-	1
WDR	-	Watchdog Reset	WDR/timer function	-	1
BREAK	-	Break	For On-chip Debug Only	-	N/A

NOP: Δεν εκτελείται καμία λειτουργία. Χρησιμοποιείται στη δημιουργία τεχνητών καθυστερήσεων εντός του προγράμματος. Οι καταχωρητές και οι σημαίες μένουν ανεπηρέαστες.
SLEEP: Θέτει το κύκλωμα σε ανενεργό κύκλο που προσδιορίζεται από τον καταχωρητή ελέγχου της MCU.
WDR: Επαναθέτει το χρονιστή επιτήρησης.
BREAK: Θέτει τη ΚΜΕ σε λειτουργία Stopped.

1.6 Διευθυνσιοδότηση

Οι εντολές της γλώσσας assembly των μικροελεγκτών μπορούν να ταξινομηθούν και με βάση τον τρόπο που επιτυγχάνεται πρόσβαση στα δεδομένα. Πρόκειται για τους τρόπους διευθυνσιοδότησης του προγράμματος και των δεδομένων. Η διευθυνσιοδότηση διακρίνεται σε απευθείας και έμμεση, όπως φαίνεται στο Σχήμα 1.20.



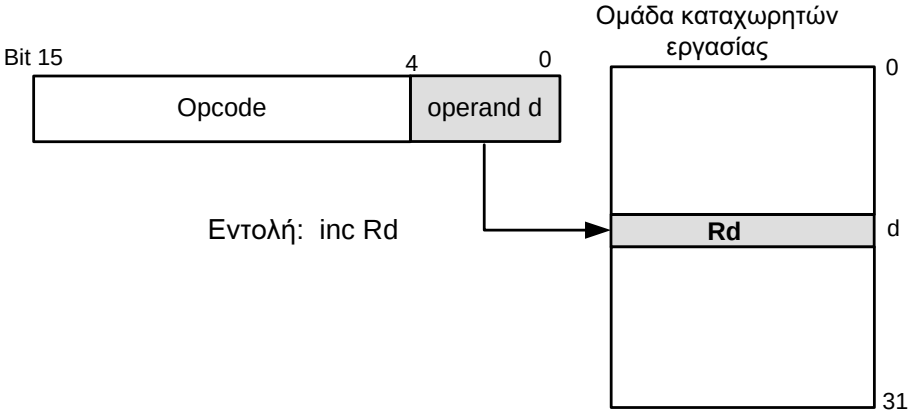
Σχήμα 1.20 Τρόποι διευθυνσιοδότησης

Οι εντολές περιλαμβάνουν 2 τμήματα. Το πρώτο (operation code ή opcode) αποτελεί το λειτουργικό κώδικα που πληροφορεί τον επεξεργαστή για τις ενέργειες που πρέπει να εκτελεστούν. Το δεύτερο τμήμα προσδιορίζει το όρισμα (operand) ή τα ορίσματα πάνω στο οποίο (ή στα οποία) θα λειτουργήσει ο κώδικας.

1.6.1 Απευθείας διευθυνσιοδότηση ενός καταχωρητή εργασίας

Η εντολή διαβάζει τα περιεχόμενα του καταχωρητή, εκτελεί τις πράξεις με αυτά και αποθηκεύει το αποτέλεσμα στον ίδιο καταχωρητή. Ο καταχωρητής είναι ένας από τους 32 καταχωρητές εργασίας R0-R31 που διαθέτουν το αντίστοιχο τμήμα μνήμης (register file). Στο σχήμα 1.21 φαίνονται οι θέσεις προέλευσης και προορισμού των δεδομένων. Με Rd συμβολίζεται ο καταχωρητής εργασίας που αποτελεί τη θέση προέλευσης των δεδομένων και τη θέση προορισμού του αποτελέσματος. Παράδειγμα τέτοιας εντολής είναι:

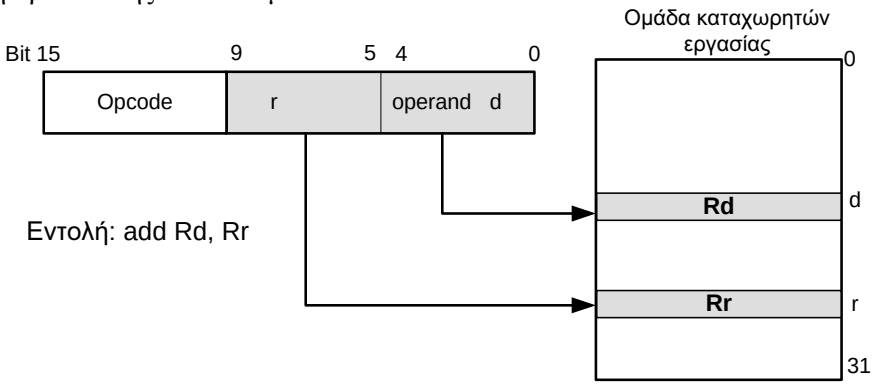
INC Rd: Αύξηση των περιεχομένων του καταχωρητή Rd κατά μία μονάδα.



Σχήμα 1.21 Απευθείας διευθυνσιοδότηση καταχωρητή

1.6.2 Απευθείας διευθυνσιοδότηση δύο καταχωρητών εργασίας

Η εντολή διαβάζει τα περιεχόμενα των δυο καταχωρητών, εκτελεί την πράξη μεταξύ των δεδομένων και αποθηκεύει το αποτέλεσμα στον καταχωρητή προορισμού Rd. Στο σχήμα 1.22 φαίνονται οι θέσεις προέλευσης και προορισμού των δεδομένων. Με Rr συμβολίζεται ο καταχωρητής εργασίας που αποτελεί τη θέση προέλευσης των δεδομένων.

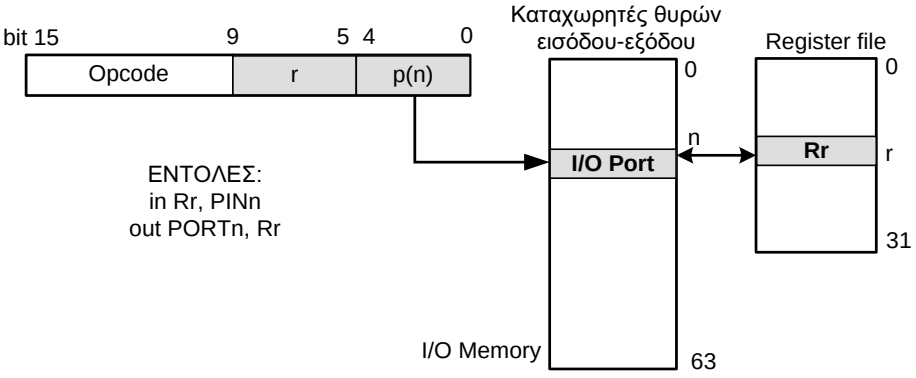


Σχήμα 1.22 Απευθείας διευθυνσιοδότηση δύο καταχωρητών

Παραδείγματα τέτοιων εντολών είναι:
ADD Rd, Rr
MOV Rd, Rr

Απευθείας διευθυνσιοδότηση των θυρών εισόδου-εξόδου

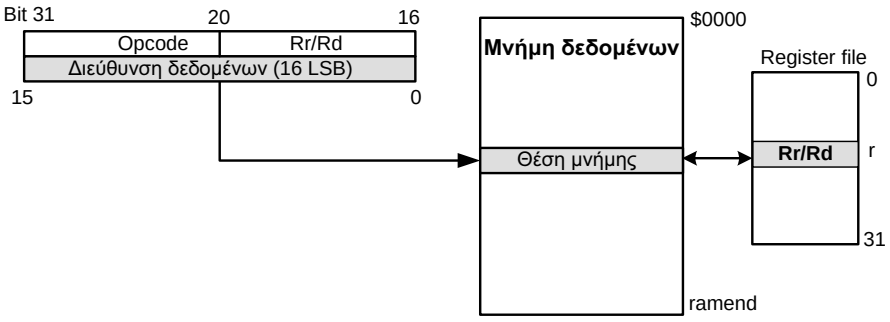
Η πρόσβαση στις θέσεις μνήμης που αντιστοιχούν οι θύρες εισόδου-εξόδου (I/O memory) επιτυγχάνεται με τις εντολές:
IN Rd, PortAddress και OUT PortAddress, Rr όπου PortAddress είναι η διεύθυνση του καταχωρητή εισόδου/εξόδου. Παραδείγματα τέτοιων εντολών είναι:
IN Rr, PINB
OUT PORTD, Rr



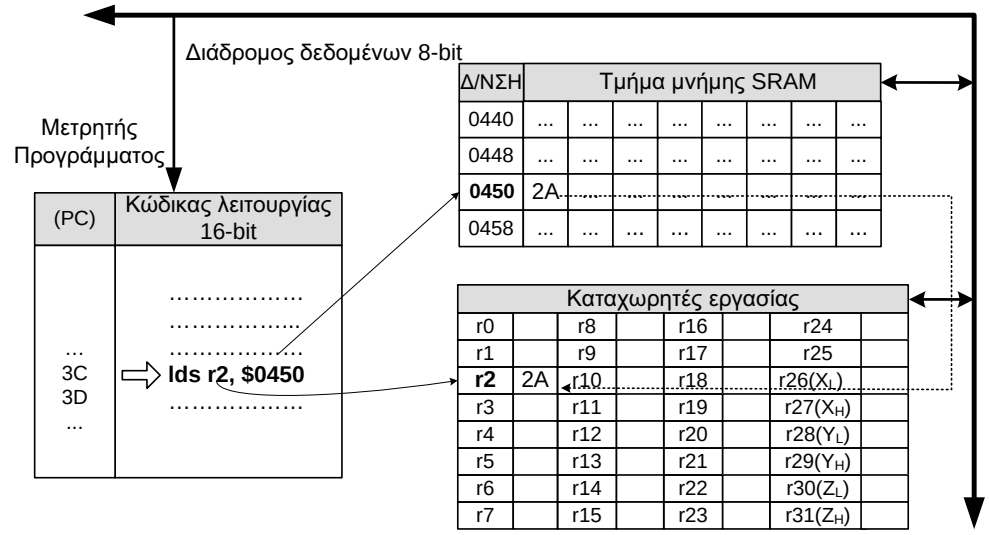
Σχήμα 1.23 Απευθείας διευθυνσιοδότηση των θυρών εισόδου-εξόδου

Απευθείας διευθυνσιοδότηση μνήμης δεδομένων

Η πρόσβαση στις θέσεις μνήμης επιτυγχάνεται με εντολές μήκους 2 λέξεων, όπου το όρισμα που αποτελεί τη μία λέξη (16 bits) αντιστοιχεί στη διεύθυνση της μνήμης δεδομένων. Άρα, ο χώρος μνήμης όπου επιτυγχάνεται πρόσβαση μπορεί να είναι μέχρι 64Kbyte (εξαρτώμενο από το τύπο του AVR μικροελεγκτή).
Παραδείγματα τέτοιων εντολών είναι:
STS K, Rr και
LDS Rd, K ; όπου K διεύθυνση μήκους 16 bits.



Σχήμα 1.24 Απευθείας διευθυνσιοδότηση της μνήμης δεδομένων



Σχήμα 1.25 Παράδειγμα απευθείας διευθυνσιοδότησης της μνήμης δεδομένων

Παράδειγμα απευθείας διευθυνσιοδότησης

Για την αντιγραφή μιας τιμής στην SRAM με απευθείας διευθυνσιοδότηση, πρέπει να προσδιορίσουμε τη διεύθυνση. Οι διευθύνσεις ξεκινούν από $60 μέχρι το τέλος της φυσικής μνήμης (στο μοντέλο AT90S8515 η τελευταία προσβάσιμη θέση είναι $025F - SRAM 512 Bytes και στον ATMega16 είναι η $045F – SRAM 1024 Bytes).

Με χρήση της εντολής: STS 0x0060, R1

αντιγράφουμε το περιεχόμενο του καταχωρητή R1 στην $1^{η}$ θέση μνήμης SRAM ($0060).

Αντίθετα, με την εντολή: LDS R1, 0x0060

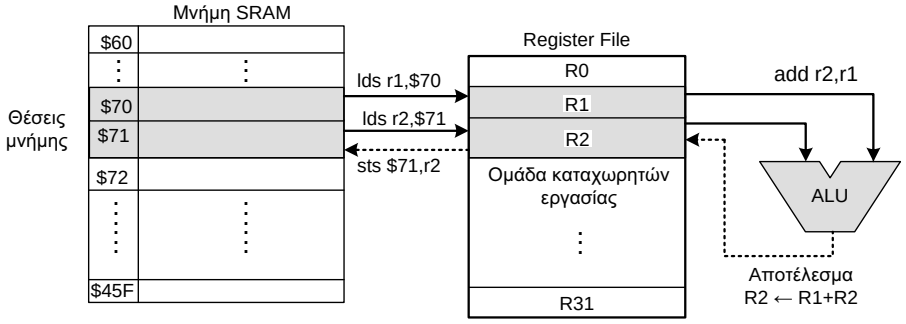
το περιεχόμενο της διεύθυνσης $0060 αντιγράφεται στον καταχωρητή R1. Πρόκειται για απευθείας πρόσβαση σε διεύθυνση που δίνει ο χρήστης. Για να αποφύγουμε τη χρήση αριθμητικών διευθύνσεων μπορούμε να αντιστοιχίσουμε συμβολικά ονόματα στις διευθύνσεις. Έτσι αποφεύγουμε τη χρήση δεκαεξαδικών διευθύνσεων και διευκολύνουμε την εργασία μας σε περίπτωση που θελήσουμε να αλλάξουμε τη δομή των δεδομένων στη μνήμη SRAM. Π.χ.

```
.EQU Location = 0x0060
 STS Location, R1
```

Στο παρακάτω παράδειγμα απευθείας πρόσβασης στη μνήμη, μεταφέρουμε τα περιεχόμενα δυο θέσεων μνήμης της SRAM στους καταχωρητές R1 και R2 και

τους προσθέτουμε (το αποτέλεσμα αποθηκεύεται στον καταχωρητή R2). Έπειτα, σώζουμε το αποτέλεσμα πίσω στη $2^{η}$ θέση μνήμης.

```
lds r1,$70      ; φόρτωση θέσης μνήμης $70 στον καταχωρητή r1
lds r2,$71      ; φόρτωση θέσης μνήμης $71 στον καταχωρητή r2
add r2,r1       ; πρόσθεση καταχωρητή r1 με r2
sts $71,r2      ; αποθήκευση r2 στη θέση μνήμης $72
```



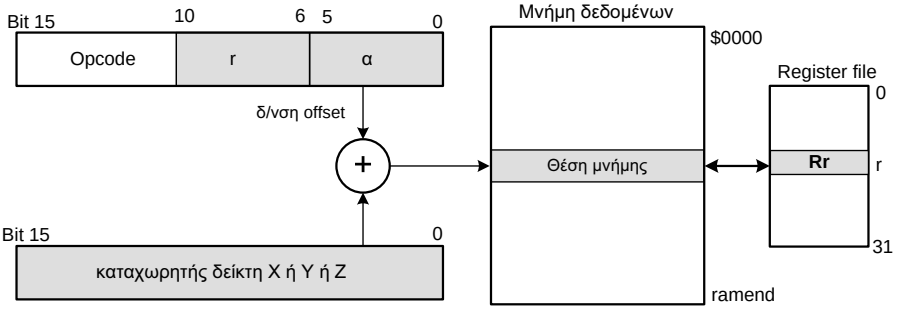
Σχήμα 1.26 Πρόσβαση (φόρτωση και αποθήκευση καταχωρητή) στην SRAM

Εκτός από την απευθείας πρόσβαση με χρήση διευθύνσεων, υπάρχει και η έμμεση πρόσβαση με χρήση καταχωρητή δείκτη, που παρουσιάζεται στη συνέχεια.

1.6.3 Έμμεση διευθυνσιοδότηση μνήμης δεδομένων

Οι εντολές αυτές προσπέλασης της μνήμης δεδομένων έχουν μήκος μιας λέξης και χρησιμοποιούν έναν καταχωρητή δείκτη (έναν εκ των X, Y, Z), που το περιεχόμενο του προσδιορίζει τη διεύθυνση της μνήμης. Η EEPROM έχει ξεχωριστό χώρο διευθύνσεων. Παραδείγματα τέτοιων εντολών είναι:

```
ST X, Rr        ; έμμεση αποθήκευση καταχωρητή σε θέση μνήμης και
LD Rr, Y        ; έμμεση ανάκληση θέσης μνήμης σε καταχωρητή.
```



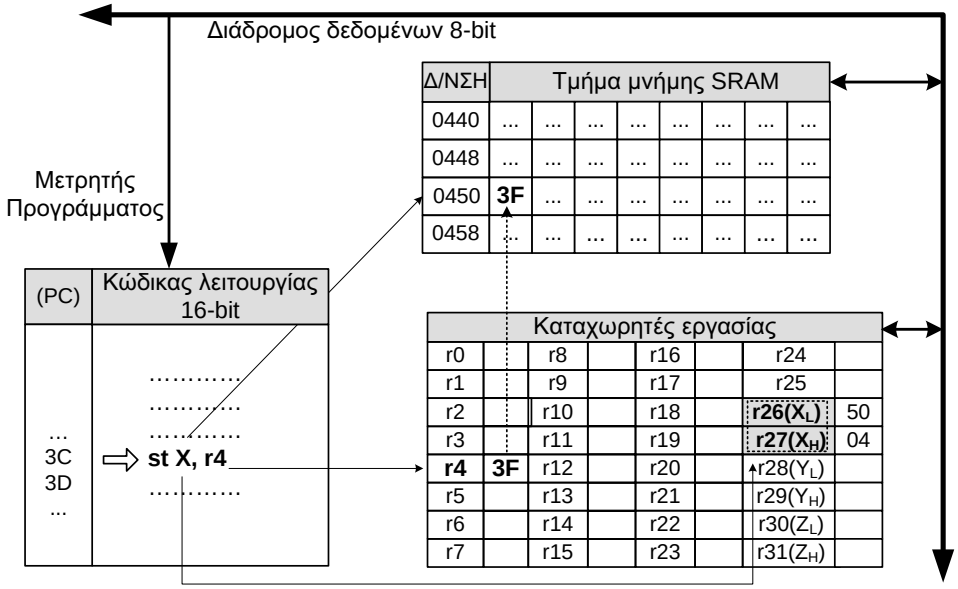
Σχήμα 1.27 Έμμεση διευθυνσιοδότηση της μνήμης δεδομένων

Σ' αυτή την κατηγορία εντολών, οι καταχωρητές δείκτη X, Y και Z μπορούν να αυξηθούν κατά μία μονάδα μετά την ανάθεση της τιμής στον καταχωρητή Rr, ώστε να δείχνουν σε επόμενη θέση μνήμης.
Παραδείγματα τέτοιων εντολών είναι:

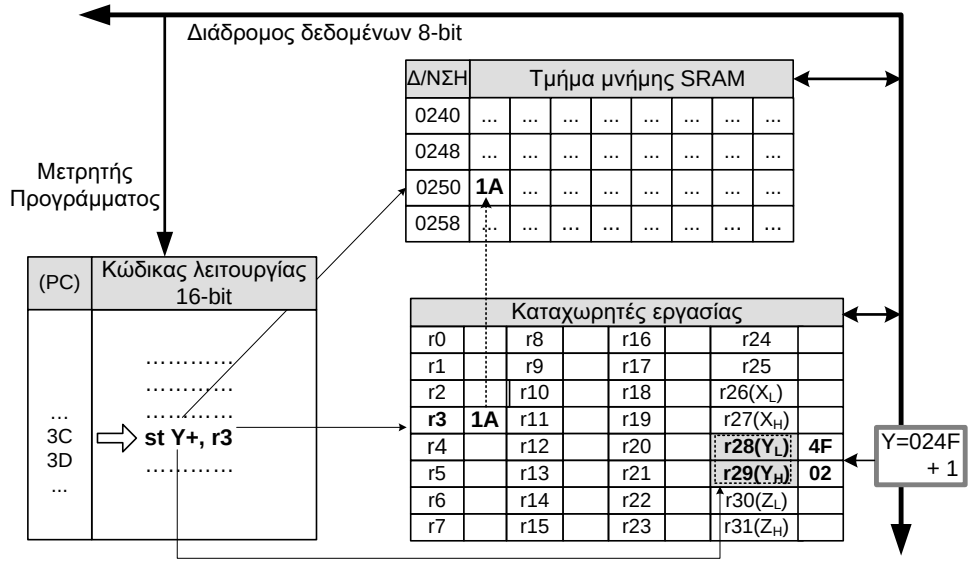
ST X+, Rr
LD Rr, X+
LD Rr, Y+

Αντίστοιχα, οι καταχωρητές δείκτη X, Y και Z μπορούν να μειωθούν κατά μία μονάδα πριν την ανάθεση της τιμής στον καταχωρητή Rr, ώστε να δείχνουν σε προηγούμενη θέση μνήμης.
Παραδείγματα τέτοιων εντολών είναι:

LD Rr, -X
LD Rr, -Z
ST –X, Rd



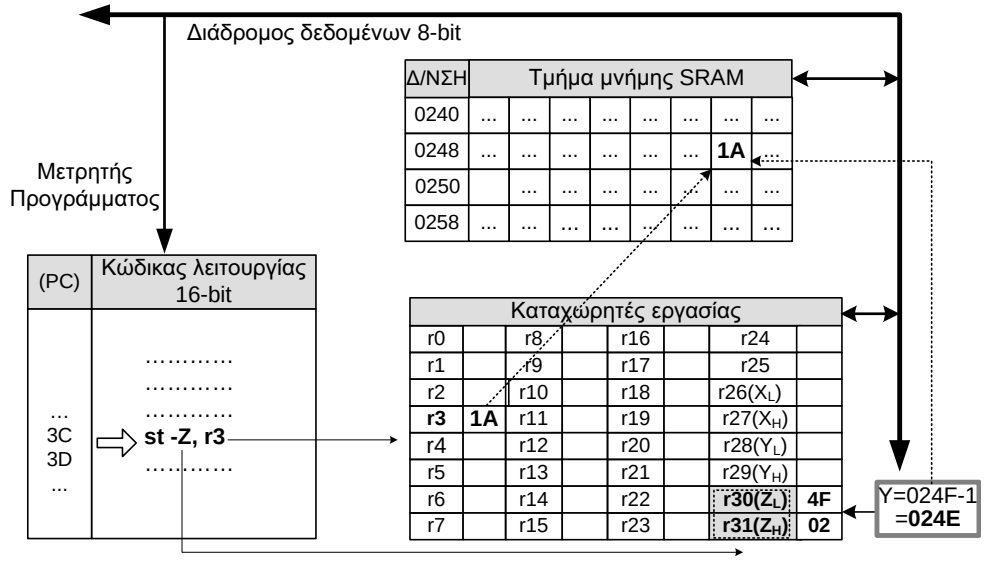
Σχήμα 1.28 Παράδειγμα έμμεσης διευθυνσιοδότησης της μνήμης δεδομένων



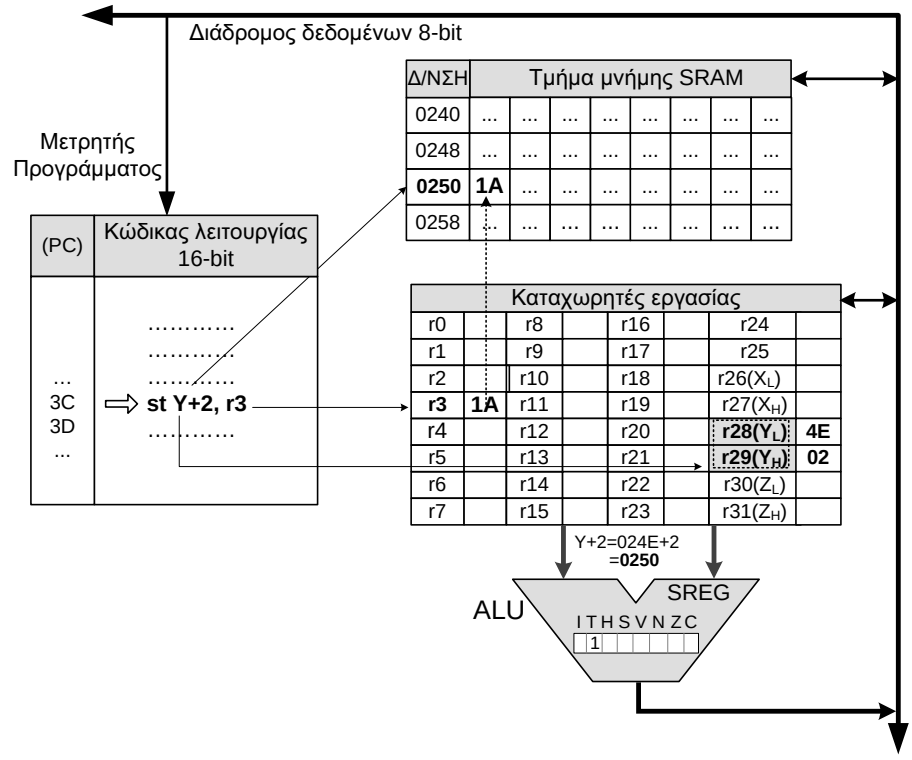
Σχήμα 1.29 Παράδειγμα έμμεσης διευθυνσιοδότησης της μνήμης δεδομένων με αύξηση δείκτη

Τέλος, μπορούμε να έχουμε έμμεση φόρτωση του καταχωρητή Rd με τα περιεχόμενα της θέσης μνήμης (Y+q), ώστε να επιτύχουμε πρόσβαση σε παραπέρα θέση μνήμης. Πρόκειται δηλαδή για μετατόπιση q (εύρους 6-bit) για έμμεση διευθυνσιοδότηση.
Παραδείγματα τέτοιων εντολών είναι:

LDD Rd,Y+2 , LDD Rd,Z+4 , ST Y+6,Rd.



Σχήμα 1.30 Παράδειγμα έμμεσης διευθυνσιοδότησης της μνήμης δεδομένων με μείωση δείκτη



Σχήμα 1.31 Παράδειγμα έμμεσης διευθυνσιοδότησης της μνήμης δεδομένων με μετατόπιση

Παραδείγματα έμμεσης διευθυνσιοδότησης

Παράδειγμα έμμεσης διευθυνσιοδότησης αποτελεί η πρόσβαση σε θέση μνήμης με χρήση καταχωρητή δείκτη. Χρειάζονται 2 καταχωρητές που περιέχουν την 16-bit διεύθυνση της μνήμης. Οι καταχωρητές δείκτη είναι τα ζεύγη X (XH:XL, R27:R26), Y (YH:YL, R29:R28) και Z (ZH:ZL, R31:R30). Επιτρέπουν πρόσβαση στη θέση που δείχνουν (π.χ. ST X, R1), μετά από μείωση της διεύθυνσης (π.χ. ST -X, R1) ή μετέπειτα αύξηση της διεύθυνσης (π.χ. ST X+, R1).
Η χρήση καταχωρητή δείκτη για εγγραφή και ανάγνωση στην SRAM επιτυγχάνεται ως εξής :

```
ldi r28,$60     ; η αρχική διεύθυνση είναι $60 και τοποθετείται στον
clr r29         ; καταχωρητή δείκτη Y ως offset διεύθυνση
st y+,data      ; αποθήκευση περιεχομένων του καταχωρητή data
                ; στην SRAM και  αύξηση της διεύθυνσης ανάγνωση
ldi r28,$60     ; η αρχική διεύθυνση είναι $60 και τοποθετείται στον
clr r29         ; καταχωρητή δείκτη Y ως offset διεύθυνση
ld data,y+      ; ανάγνωση περιεχομένων καταχωρητή data από την
                ; SRAM κα αύξηση διεύθυνσης
```

Παράδειγμα αποτελεί η πρόσβαση σε 3 διαδοχικές θέσεις, για μεταφορά δεδομένων με χρήση καταχωρητή δείκτη:

```
.EQU Location1 = 0x0060
.EQU Location2 = 0x0072
    LDI XH, HIGH(Location1)
    LDI XL, LOW(Location1)

    LD R1, X+                   ; Ανάγνωση τριών διαδοχικών θέσεων
    LD R2, X+                   ; από τη διεύθυνση 0x0060
    LD R3, X

    LDI XH, HIGH(Location2) ; καθορισμός διεύθυνσης αποθήκευσης
    LDI XL, LOW(Location2)

    ST X+, R1                   ; Αποθήκευση σε τρεις διαδοχικές
    ST X+, R2                   ; θέσεις με αρχή τη διεύθυνση 0x0072
    ST X, R3
```

Έτσι, επιτυγχάνεται πρόσβαση σε διαδοχικές θέσεις μνήμης, π.χ. σε πίνακα δεδομένων.
Επίσης, σημαντική είναι η πρόσβαση στη μνήμη χρησιμοποιώντας μια συγκεκριμένη διεύθυνση έναρξης σε έναν καταχωρητή δείκτη. Στην περίπτωση αυτή, η διεύθυνση αποθηκεύεται στον καταχωρητή δείκτη και μια τιμή μετατόπισης προστίθεται κάθε φορά στη διεύθυνση, ώστε να επιτυγχάνουμε πρόσβαση στη θέση αυτή για εγγραφή/ανάγνωση.

Παράδειγμα έμμεσης διευθυνσιοδότησης αποτελεί και η πρόσβαση σε θέση μνήμης με χρήση καταχωρητή δείκτη, σε συνδυασμό με μια μετατόπιση q (6-bit), 0<q<63.
Η διεύθυνση υπολογίζεται προσθέτοντας την τιμή μετατόπισης q σε έναν καταχωρητή δείκτη. Κατά την πρόσβαση αυτή, η τιμή του καταχωρητή δεν αλλάζει. Για την πρόσβαση στη διεύθυνση $0062, όπως έγινε στο παραπάνω παράδειγμα, τώρα θα θέσουμε στον καταχωρητή δείκτη την αρχική τιμή $0060:

```
.EQU Location = 0x0060
.DEF Register_1 = R1
    LDI YH, HIGH(Location)
    LDI YL, LOW(Location)
```

Οπότε, για πρόσβαση στη διεύθυνση $0062 εκτελούμε:

```
    STD Y+2, Register_1    ; εγγραφή στην SRAM
    LDD Register_1, Y+2    ; ανάγνωση
```

Η τιμή 2 δεν προστίθεται στον καταχωρητή Y, παρά μόνο αλλάζει η διεύθυνση προορισμού. Η λειτουργία γίνεται με τους καταχωρητές Y, Z αλλά όχι με τον X!

Μ' αυτό τον τρόπο, το πρώτο παράδειγμα (μεταφορά δεδομένων τριών διαδοχικών διευθύνσεων σε θέσεις μνήμης με έμμεση διευθυνσιοδότηση) με τη χρήση μετατόπισης διαμορφώνεται ως εξής:

```
.EQU Location1 = 0x0060
.EQU Location2 = 0x0072
    LDI YH, HIGH(Location1)
    LDI YL, LOW(Location1)
    LD R1, Y+           ; Ανάγνωση τριών διαδοχικών θέσεων από διεύθυνση 0x0060
    LD R2, Y+
    LD R3, Y
    STD Y+10, R1        ; Αποθήκευση σε τρεις διαδοχικές θέσεις με αρχή
    STD Y+11, R2        ; τη διεύθυνση 0x0072
    STD Y+12, R3
```

Παράδειγμα 1.11:
Στο πρόγραμμα αυτό γίνεται συνεχής ανάγνωση της θύρας PortD. Όταν διαπιστώνεται ότι άλλαξε η τιμή της, τότε κάθε νέα τιμή αποθηκεύεται σε διαδοχικές θέσεις μνήμης SRAM, αρχίζοντας από τη διεύθυνση $0060 ως την $0150.

```
.include "m16def.inc"
.def temp=r16           ; καταχωρητής για προσωρινή αποθήκευση
.def port_value=r17     ; αποθήκευση τιμής θύρας
.def reg = r18
.org 0x000
  rjmp main             ; παράκαμψη διανυσμάτων διακοπών
.org 0x100
main:                   ; κυρίως πρόγραμμα
    ldi reg,LOW(RAMEND)     ; αρχικοποίηση δείκτη στοίβας
    out SPL,reg
    ldi reg,HIGH(RAMEND)
    out SPH,reg
    clr temp
    out DDRD,temp       ; Port D ως είσοδος
    ser temp
    out PORTD,temp          ; Port D ( ενεργοποίηση pull-ups)
    clr xh              ; δήλωση διεύθυνσης αποθήκευσης στην SRAM
    ldi xl, 0x60        ; θέτουμε διεύθυνση 0x60 στον καταχωρητή X
    in port_value,PIND  ; 1η ανάγνωση θύρας
diavasma:               ; βρόχος ανάγνωσης
    in temp,PIND        ; διάβασμα τρέχουσας τιμής
    cp temp,port_value  ; έλεγχος εάν έχει αλλάξει η τιμή
; Για να εφαρμοστεί πρακτικά θα πρέπει να παρεμβάλλουμε χρονοκαθυστέρηση
    breq diavasma       ; ίδιες τιμές, επιστροφή στο diavasma
    nop
    st x+,temp          ; αλλιώς αποθήκευση τιμής & αύξηση καταχωρητή X
```

```
    mov port_value,temp          ; ώστε να δείχνει στην επόμενη θέση μνήμης
    cpi xl,0x51          ; έλεγχος να μην περάσουμε τη διεύθ. 0x150
    brne diavasma
    ldi xl,0x60          ; αν τη φτάσουμε, αρχίζουμε αποθηκεύσεις πάλι από 0x60
    clr xh
    rjmp diavasma
```

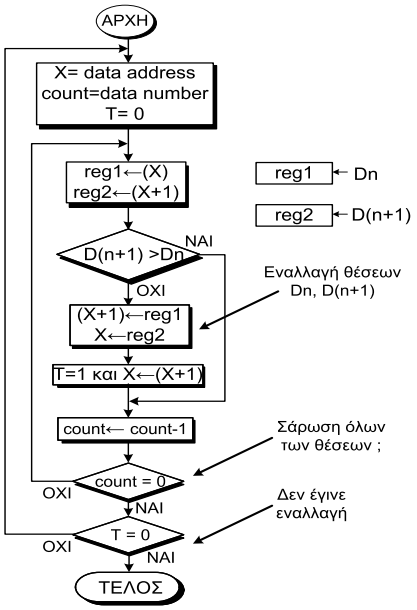
Παράδειγμα 1.12:

Μεταφορά ενός πίνακα των 20 bytes από το τμήμα κώδικα (cseg) στο τμήμα δεδομένων (dseg). Εξετάζεται κάθε στοιχείο του πίνακα. Εάν πρόκειται για περιττό, αποθηκεύεται στη διεύθυνση $0060, ενώ αν πρόκειται για άρτιο, στη διεύθυνση $0074 του τμήματος δεδομένων.

```
.INCLUDE "m16def.inc"
.DEF   count=R18          ; Ορίζουμε όνομα καταχωρητή-μετρητή
.EQU Location1 = 0x0060          ; Διεύθυνση αποθήκευσης περιττών
.EQU Location2 = 0x0074          ; Διεύθυνση αποθήκευσης άρτιων

start:
    ldi count,21                 ; Μετρητής για μεταφορά 20 στοιχείων
    ldi ZH, HIGH(Table*2)        ; Η διεύθυνση του πίνακα  δεδομένων
    ldi ZL,LOW(Table*2)          ; στη μνήμη προγράμματος στον καταχωρητή Z
    ldi XH,HIGH(Location1)       ; Η διεύθυνση του πίνακα περιττών στον καταχ. X
    ldi XL,LOW(Location1)
    ldi YH,HIGH(Location2)       ; Η διεύθυνση του πίνακα άρτιων στον καταχ.Y
    ldi YL,LOW(Location2)

loop:
    lpm r20,z+           ; Φόρτωση μνήμης προγράμματος και
                         ; αύξηση δείκτη πρόσβασης επόμενης θέσης.
   dec count             ; Μείωση μετρητή και άλμα
    breq end             ; όταν μεταφερθούν όλα τα στοιχεία.
    sbrs r20,0           ; Αν πρόκειται για περιττό (R20.0=1) skip
    rjmp even                    ; Αλλιώς άρτιος και άλμα στην ετικέτα even.
odd:                     ; Ο αριθμός είναι περιττός.
    ST X+, R20           ; Αποθήκευση περιττών και
    rjmp loop            ; επανάληψη με φόρτωση νέου αριθμού.
even:
    ST Y+, R20           ; Αποθήκευση άρτιων και επανάληψη.
     rjmp loop

.cseg                    ; Στη μνήμη προγράμματος εισάγεται πίνακας δεδομένων.
Table:
.DW 0x0100,0x0706,0x1713,0x3326,0x27C6
.DW 0x5042,0x7A61,0xA2F1,0xE0D7,0x89FD
end: .exit
```

Παράδειγμα 1.13:

Κατάταξη αριθμών που βρίσκονται στη μνήμη δεδομένων (διεύθυνση $0060) σε αύξουσα σειρά. Εφαρμόζουμε τη μέθοδο των φυσαλίδων σύμφωνα με την οποία πραγματοποιούμε διαδοχικά περάσματα στην ακολουθία των αριθμών και αντιστρέφουμε τους διαδοχικούς αριθμούς που δεν βρίσκονται σε αύξουσα σειρά. Η διαδικασία επαναλαμβάνεται έως ότου να μη χρειαστεί εναλλαγή.

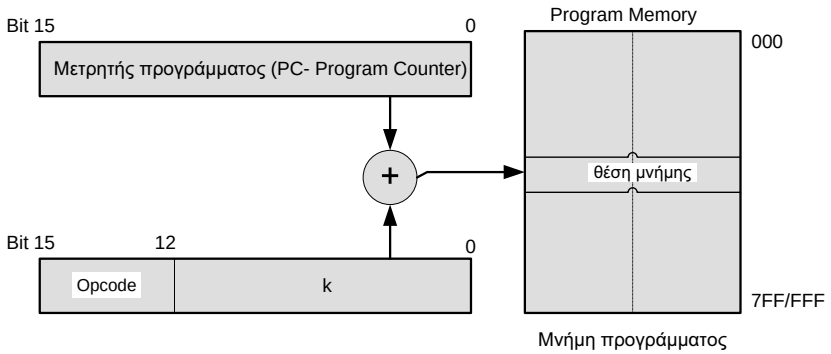



```
.INCLUDE "m16def.inc"
.DEF   reg1=R18
.DEF   reg2=R20
.EQU Location1 = 0x0060
.DEF  count=r16
.EQU numb= 0x100
```

```
sort:
    LDI XH, HIGH(Location1)   ; διεύθυνση μνήμης δεδομένων
    LDI XL, LOW(Location1)
    ldi count, numb       ; πλήθος αριθμών
    clt                   ; αρχικοποίηση δείκτης αλλαγής (T=0)

loop:
    ld reg1,x+            ; φόρτωση διαδοχικών τιμών από μνήμη
    ld reg2,x             ; δεδομένων σε δυο καταχωρητές
    cp reg2,reg1          ; σύγκριση διαδοχικών τιμών
    brge no_change

    st x,reg1             ; εναλλαγή για αύξουσα διάταξη
    st -x,reg2
    set                   ; θέτουμε δείκτη αλλαγής (T=1)

no_change:
    dec count             ; σάρωση όλου του πίνακα
    brne loop
    brbs 6,sort           ; επανάληψη σε περίπτωση εναλλαγής (T=1)
.exit
```

1.6.4 Σχετική διευθυνσιοδότηση μνήμης προγράμματος

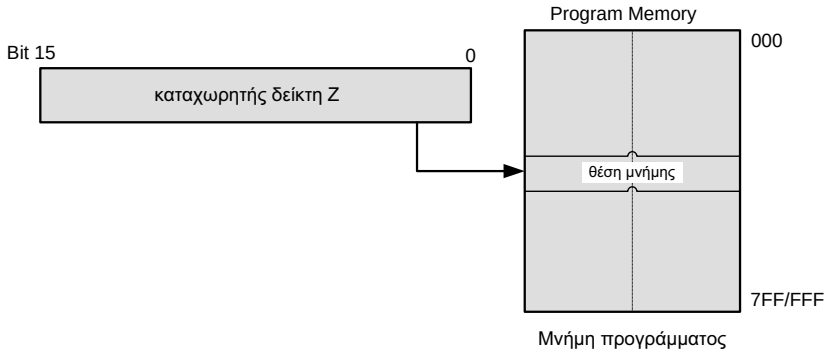
Οι εντολές αυτές επιτυγχάνουν πρόσβαση σε χώρο μνήμης προγράμματος και είναι του τύπου **RCALL, RJMP**, όπου χρησιμοποιείται μια μετατόπιση +/- 2K στο περιεχόμενο του μετρητή προγράμματος.



Σχήμα 1.32 Σχετική διευθυνσιοδότηση της μνήμης προγράμματος

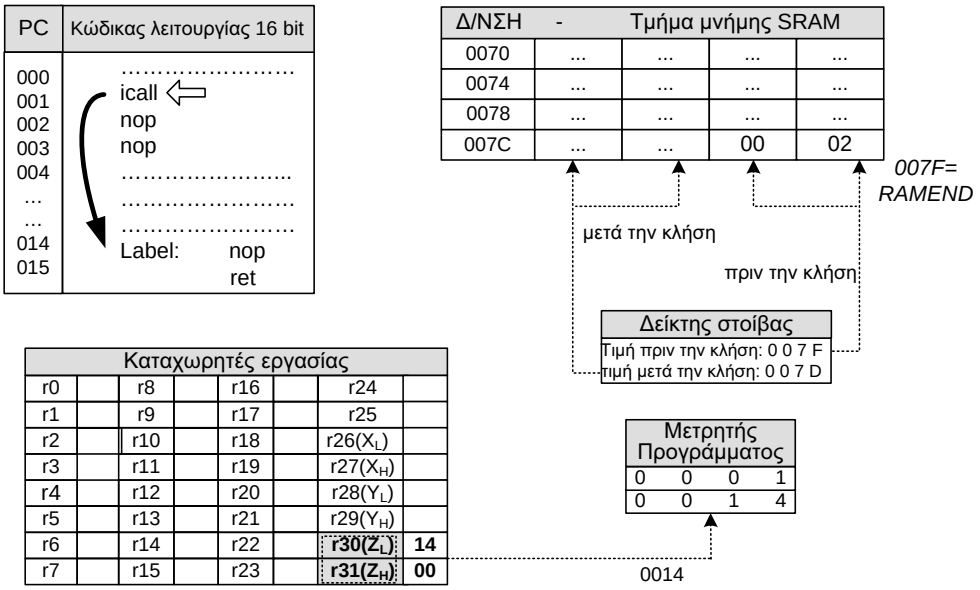
Έμμεση διευθυνσιοδότηση μνήμης προγράμματος

Οι εντολές αυτές επιτυγχάνουν πρόσβαση σε χώρο μνήμης προγράμματος έως 64Kbytes με χρήση του καταχωρητή Z, ως δείκτη μιας θέσης μνήμης προγράμματος. Παραδείγματα τέτοιων εντολών είναι οι **ICALL**, **IJMP**.



Σχήμα 1.33 Έμμεση διευθυνσιοδότηση της μνήμης προγράμματος

Η εντολή **IJMP** καλεί έμμεσα μια υπορουτίνα η διεύθυνση της οποίας αντιστοιχεί στο περιεχόμενο του καταχωρητή δείκτη Z. Δηλαδή, το περιεχόμενο του Z φορτώνεται στο μετρητή προγράμματος (PC ←Z(15:0)), όπως φαίνεται στο σχήμα 1.34.



Σχήμα 1.34 Παράδειγμα έμμεσης διευθυνσιοδότησης της μνήμης προγράμματος

Παρόμοια με την εντολή **ICALL** εκτελείται έμμεσο άλμα σε διεύθυνση της μνήμης προγράμματος που δίνεται από τον καταχωρητή δείκτη Z [PC ←Z(15:0)]. Η τυπική μορφή (format) των εντολών του μικροελεγκτή AVR αποτελείται από τα παρακάτω τέσσερα πεδία. Ενδέχεται ένα ή και περισσότερα πεδία να είναι κενά.

επιγραφή	λειτουργία	όρισμα(τα)	σχόλια

Τμήμα επιγραφής (label): Η επιγραφή αντιστοιχεί σε μια θέση του προγράμματος και είναι καθοριστικής σημασίας για τη ροή του προγράμματος. Κάθε ετικέτα πρέπει να αρχίζει με αλφαβητικό χαρακτήρα και να ακολουθείται από άνω και κάτω τελεία. Δεν επιτρέπεται να χρησιμοποιηθεί η ίδια ετικέτα πάνω από μια φορά.

Τμήμα λειτουργίας (mnemonic): Σ' αυτό το πεδίο βρίσκεται το μνημονικό όνομα της εντολής (ή της ψευδοεντολής). Τα γράμματα του μνημονικού ονόματος παραμπέμπουν στη λειτουργία της εντολής. Το όνομα της εντολής πρέπει να ακολουθείται από ένα τουλάχιστον κενό. Παραδείγματα format εντολών δίνονται στον πίνακα 1.4.

Τμήμα ορισμάτων (operands): Τα ορίσματα καθορίζουν την προέλευση και τον προορισμό των δεδομένων, τη ροή του προγράμματος,κλπ. Το πλήθος και η σειρά των ορισμάτων εξαρτάται από την εκάστοτε εντολή και διαχωρίζονται μεταξύ τους με κόμμα.

Πίνακας 1.4 Τυπική μορφή (format) εντολών

Συμβολική Μορφή	**Κώδικας Λειτουργίας**			
ldi Rd, K	1111	KKKK	DDDD	KKKK
Τα D=ορίζουν τον Rd (16-31), K=8 bit σταθερά. Λειτουργία: (Rd) ←K				
inc Rd	1001	010D	DDDD	0011
Τα D=ορίζουν τον Rd (0-31). Λειτουργία: (Rd) ← (Rd) + 1				
add Rd, Rr	0000	011R	DDDD	RRRR
Τα D=ορίζουν τον Rd (0-31) και τα R τον Rr (0-31). Λειτουργία: (Rd)← (Rd) + (Rr)				
sbi PORT, b	1001	1010	AAAA	Abbb
A: διεύθυνση καταχωρητή I/O, b: θέση bit καταχωρητή I/O (3-bit) Λειτουργία: Γίνεται 1 το τάξης (bbb) bit του καταχωρητή I/O				
brbs s, K	1111	00KK	KKKK	Ksss
s: θέση bit καταχωρητή κατάστασης (3-bit). Λειτουργία: Αν το bit αυτό είναι 1 τότε: (PC) ← (PC) + K +1				
in Rd, PORT	1011	0AAD	DDDD	AAAA
Τα D=ορίζουν τον Rd (0-31) και τα A=6-bit διεύθυνση καταχωρητή I/O (PORT). Λειτουργία: (Rd) ← (PORT 0-63)				
out PORT, Rr	1011	1AAR	RRRR	AAAA
Τα R=ορίζουν τον Rr (0-31) και τα A=6-bit διεύθυνση καταχωρητή I/O (PORT). Λειτουργία: (PORT 0-63) ←(Rr)				
sbrs Rd, b	1111	111 R	RRRR	0bbb
Τα R=ορίζουν τον Rr (0-31) και το b: θέση bit. Λειτουργία: Αν το bit αυτό είναι 1 τότε: (PC) ← (PC) +2 αλλιώς (PC)= (PC) +1				
rjmp loop	1100	KKKK	KKKK	KKKK
Εδώ K=12 bit, προσημασμένος. Λειτουργία: (PC) ← (PC) + K +1				
jmp loop	1001	010K	KKKK	110K
	KKKK	KKKK	KKKK	KKKK K=22 bit (έως 4MB)
Άλμα στη διευθ. loop. Καταλαμβάνει 2 λέξεις της μνήμης προγράμματος				
adiw Rd, K	1001	011**0**	KKDD	KKKK (η εντολή sbiw Rd,K με 011**1**)
Τα D=ορίζουν τον Rd (24, 26,28,30), K=0-63 σταθερά Λειτουργία: (Rd:Rd+1)←(Rd:Rd+1)+K				
lds Rd, K	1001	00**0**D	DDDD	0000 (η εντολή sts K, Rd, με 00**1**D)
	KKKK	KKKK	KKKK	KKKK
Τα D=ορίζουν τον Rd (0-31), K=16 bit σταθερά. Λειτουργία: (Rd:Rd+1) ←K				
ldd Rd, Y+q	10Q0	QQ0D	DDDD	1QQQ
Τα D=ορίζουν τον Rd (0-31), q=6 bit σταθερά. Λειτουργία: (Rd) ← ((Y)+q) όπου Y=R28:R29				
st Y, Rr	1000	001R	RRRR	1000
Τα R=ορίζουν τον Rr (0-31). Λειτουργία: ((Y)) ← (Rr) όπου Y=R28:R29				

Τμήμα σχολίων: Η χρήση σχολίων βοηθά στην κατανόηση του προγράμματος και είναι προαιρετική. Τα σχόλια πρέπει να έπονται του ελληνικού ερωτηματικού (;) και δε λαμβάνονται υπόψη από το μεταφραστή.

1.7 Στοίβα

Η στοίβα χρησιμοποιείται από την αριθμητική λογική μονάδα (ALU) για αποθήκευση διευθύνσεων επιστροφής από υπορουτίνες. Η στοίβα χρειάζεται ένα δείκτη στοίβας (SP) και χώρο μνήμης στην SRAM.

Καταχωρητής δείκτης στοίβας (stack pointer- SPH/SPL)

Σε περίπτωση εξυπηρέτησης μιας ρουτίνας διακοπής ή κατά την κλήση μιας υπορουτίνας, η διεύθυνση επιστροφής του κυρίου προγράμματος αποθηκεύεται στη στοίβα. Το περιεχόμενο του καταχωρητή δείκτη στοίβας είναι η διεύθυνση της μνήμης SRAM που αντιστοιχεί στην κορυφή της στοίβας. Ο καταχωρητής έχει μήκος 1 byte για μικροελεγκτές μεγέθους μνήμης ως 256 bytes και μήκος 2 bytes για μνήμες άνω των 256 bytes με συμβολισμό SPH-SPL. Κατά την εισαγωγή μιας τιμής στη στοίβα (διαδικασία push) η τιμή του δείκτη στοίβας ελαττώνεται κατά μια μονάδα. Αντίστροφα, η ανάκτηση μιας αποθηκευμένης τιμής από τη στοίβα (διαδικασία pop), αυξάνει κατά μια μονάδα το δείκτη στοίβας.
Λειτουργίες στοίβας απαιτούνται όταν πρόκειται να κληθεί υπορουτίνα ή διακοπή. Τότε η πραγματική διεύθυνση φυλάσσεται στη στοίβα για επιστροφή στο σημείο του προγράμματος μετά την εξυπηρέτηση της ρουτίνας.
Η στοίβα βρίσκεται στο τέλος της ενσωματωμένης SRAM, ονόματι RAMEND και ορίζεται στο αρχείο "xxxxdef.inc" για τον αντίστοιχο τύπο επεξεργαστή, οπότε δε μας απασχολεί η πραγματική της τιμή. Αν ένα byte σταλεί στη στοίβα, εγγράφεται σε θέση της μνήμης SRAM και ο 16-bit καταχωρητής δείκτη στοίβας SPH:SPL μειώνεται στην επόμενη χαμηλότερη θέση στοίβας.
Περαιτέρω αποστολή bytes φέρνει το δείκτη πλησιέστερα προς την αρχή της SRAM. Αν ένα byte ληφθεί από τη στοίβα, τότε ο δείκτης αυξάνει προτού διαβαστεί η τιμή. Η τελευταία τιμή που αποθηκεύεται θα διαβαστεί πρώτη. Πρόκειται, λοιπόν, για δομή τύπου Last-In-First-Out (LIFO).
Μια διεύθυνση SRAM έχει μήκος 16 bits, οπότε ο καταχωρητής SPL κρατά τα 8 LSB και ο SPH τα 8 MSB της διεύθυνσης.

Η στοίβα αποτελεί την πλέον συνηθισμένη χρήση της SRAM.
Για να χρησιμοποιήσουμε την SRAM ως στοίβα πρέπει να θέσουμε τον καταχωρητή δείκτη στοίβας (με συμβολισμό SPH-SPL) ίσο με την υψηλότερη διεύθυνση SRAM, όπως δείχνει το ακόλουθο παράδειγμα:

```
.DEF Register = R16              ; αρχικοποίηση στοίβας
    LDI Register, HIGH(RAMEND)   ; άνω byte
    OUT SPH,Register             ; στον stack pointer
    LDI Register, LOW(RAMEND)    ; κάτω byte
    OUT SPL,Register             ; στον stack pointer
```

Η τιμή RAMEND εξαρτάται από το μοντέλο μικροελεγκτή και περιέχεται στο INCLUDE file. Το αρχείο 8515def.inc έχει τη γραμμή:

```
.equ RAMEND =$25F                ; τελευταία θέση μνήμης
```

Προφανώς, το αρχείο 8515def.inc πρέπει να εισαχθεί από το μεταφραστή με χρήση της επόμενης εντολής στην αρχή του πηγαίου κώδικα:

```
.INCLUDE  "C:\location\8515def.inc"
```

Αφού αρχικοποιήσαμε τη στοίβα, μένει να μάθουμε πως υλοποιούνται οι διαδικασίες αποθήκευσης και ανάκτησης δεδομένων. Το περιεχόμενο ενός καταχωρητή σώζεται στη στοίβα με την εντολή: PUSH Register
Δε μας απασχολεί η διεύθυνση προορισμού και το γεγονός ότι ο καταχωρητής δείκτη μειώθηκε κατά μία μονάδα. Το μόνο που μας ενδιαφέρει είναι ότι η ανάκτηση των δεδομένων γίνεται με την εντολή: POP Register

Σχόλιο: Η στοίβα είναι χρήσιμη όταν δεν υπάρχουν διαθέσιμοι καταχωρητές, όταν τα δεδομένα ζητούνται επανειλημμένως στο πρόγραμμα και όταν εκτελούνται υπορουτίνες. Διαφορετικά η αποθήκευση καταχωρητών στη στοίβα είναι άχρηστη και σπαταλά πόρους του επεξεργαστή.
Σε περίπτωση κλήσης μιας υπορουτίνας, η διεύθυνση επιστροφής στο κυρίως πρόγραμμα (δηλαδή η τιμή του μετρητή προγράμματος) σώζεται στη στοίβα (διαδικασία PUSH) και έπειτα εκτελείται το άλμα στην υπορουτίνα. Στο τέλος της εκτέλεση της υπορουτίνας, η διεύθυνση επιστροφής ανακτάται από τη στοίβα (διαδικασία POP) και φορτώνεται στο μετρητή προγράμματος. Συνεπώς συνεχίζεται η εκτέλεση του προγράμματος με την επόμενη εντολή μετά την RCALL:

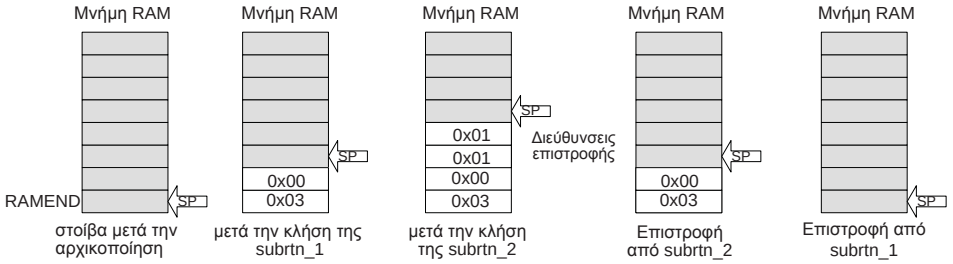
```
      [...] εντολές κύριου πρόγραμματος
   RCALL subroutine              ; άλμα στην υπορουτίνα
     [...] υπόλοιπο πρόγραμμα.
       ….
subroutine:                      ; διεύθυνση άλματος
    [...] εκτέλεση υπορουτίνας
   RET                           ; Επιστροφή στο υπόλοιπο πρόγραμμα
```

Τέλος, η στοίβα εξυπηρετεί τις διακοπές υλικού (hardware interrupts). Οι διακοπές σταματούν την κανονική εκτέλεση του προγράμματος και καλούνται ειδικές ρουτίνες για την εξυπηρέτηση τους. Έπειτα το πρόγραμμα επιστρέφει στη θέση όπου έγινε η διακοπή για την εκτέλεση του υπόλοιπου προγράμματος. Η διεύθυνση επιστροφής αποθηκεύεται στη στοίβα.

Για να εξηγηθεί πως διαμορφώνεται η στοίβα μέσω των εντολών rcall και ret όταν έχουμε φωλιασμένες κλήσεις παραθέτουμε το παρακάτω πρόγραμμα, όπου το κύριο πρόγραμμα καλεί μια ρουτίνα που και αυτή με τη σειρά της καλεί μια δεύτερη :

```
org 0x00
    ldi SPL, low(RAMEND)      ; αρχικοποίηση στοίβας στην τελευταία θέση
    ldi SPH, high(RAMEND)     ; μνήμης
    rcall subrtn_1            ; κλήση 1ης ρουτίνας
.org 0x100
subrtn_1:                     ; διεύθυνση 1ης ρουτίνας
    rcall subrtn_2            ; κλήση 2ης ρουτίνας
    ret
.org 0x140
subrtn_2:                     ; διεύθυνση 2ης ρουτίνας
    ret
```

Οι διευθύνσεις επιστροφής κατά την κλήση των ρουτινών αποθηκεύονται στη στοίβα όπως φαίνεται στο Σχήμα 1.35.



Σχήμα 1.35 Διαμόρφωση στοίβας

Όπου RAMEND αντιστοιχεί στη τελευταία διαθέσιμη θέση μνήμης SRAM και χρησιμοποιείται για την αρχικοποίηση της στοίβας. Όταν επιστρέφουμε από τη στοίβα, η διεύθυνση επιστροφής ανακτάται και ο δείκτης SP αυξάνει. Δηλαδή κατά την επιστροφή από την subrtn_2, εκτελείται άλμα στη διεύθυνση 0x101 (εντολή ret στη subrtn_1) και ο SP δείχνει ξανά στο επίπεδο 1.

Επίσης η στοίβα χρησιμεύει στο πέρασμα ορισμάτων σε υπορουτίνες μέσω των εντολών push και pop. Η διαδικασία φαίνεται στο ακόλουθο παράδειγμα:

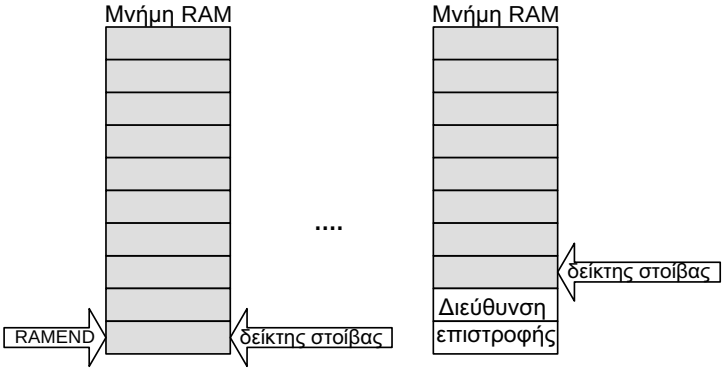
```
    push r16
    push r17              ; push 16-bit ορίσματος των r17 : r16
    rcall set_TCNT1       ; Κλήση ρουτίνας
set_TCNT1:                ; Η ρουτίνα γράφει το όρισμα στο μετρητή του Timer1.
    pop r15               ; Αποθηκεύουμε προσωρινά τη διεύθυνση επιστροφής
    pop r14               ; στους καταχωρητές r15 : r14
    pop r17
    pop r16               ; Ανάκτηση ορίσματος από στοίβα
    out TCNT1H, r17       ; (με ανάστροφη σειρά!) και η χρήση του.
    out TCNT1L, r16
    push r14              ; Ανάκτηση της διεύθυνσης επιστροφής από τη στοίβα
    push r15
    ret                   ; Επιστροφή
```

Θα μπορούσαμε να χρησιμοποιήσουμε απευθείας τους καταχωρητές r17: r16 αντί της στοίβας, για να περάσουμε τα ορίσματα. Όμως, μέσω της στοίβας έχουμε μεγαλύτερη ευελιξία, χωρίς να περιοριζόμαστε από τους συγκεκριμένους καταχωρητές για να περάσουμε τιμές. Επίσης, οι καταχωρητές αυτοί μπορούν να χρησιμοποιηθούν σε επεξεργασία-υπολογισμούς (στη ρουτίνα) αφού η τιμή τους από το κύριο πρόγραμμα είναι αποθηκευμένη στη στοίβα.
Γενικά, όμως, όταν θέλουμε να διατηρηθούν (ως προς το κύριο πρόγραμμα), πριν και μετά την εκτέλεση μίας ρουτίνας, οι τιμές των καταχωρητών που χρησιμοποιούνται τόσο από το κύριο πρόγραμμα όσο και από τη ρουτίνα, χρησιμοποιούμε εντολές push και pop για τους καταχωρητές αυτούς μέσα στη ρουτίνα, οι οποίες όμως πρέπει να είναι ίσες στο πλήθος. Αυτό σημαίνει ότι κάθε push θα αντιστοιχεί σε μια pop. Διαφορετικά, δε μπορεί να βρεθεί η διεύθυνση επιστροφής (η διεύθυνση της εντολής του προγράμματος που έπεται αυτής που έκανε την κλήση) που είναι αποθηκευμένη στη στοίβα.

1.8 Ρουτίνες

Οι ρουτίνες είναι ανεξάρτητα τμήματα κώδικα τα οποία καλούμε κατά τη διάρκεια του κυρίου προγράμματος για την εκτέλεση κάποιας συγκεκριμένης λειτουργίας. Έπειτα επιστρέφουμε από αυτές μέσω της εντολής RET, στην επόμενη εντολή (από αυτήν που κάλεσε τη ρουτίνα) του κυρίου προγράμματος, ενώ μπορούμε να τις καλούμε στη συνέχεια όποτε χρειάζεται και όσες φορές θέλουμε. Έτσι, κάνουμε οικονομία στη μνήμη προγράμματος και δίνουμε πιο απλή και κατανοητή δομή στο πρόγραμμά μας.

Μία ρουτίνα ξεκινά με μια ετικέτα που αντιστοιχεί στο όνομα της και τελειώνει με την εντολή επιστροφής (RET). Η κλήση της γίνεται με την εντολή RCALL. Αυτή η εντολή εκτελεί άλμα σε μια σχετική διεύθυνση, έχει μήκος 2 bytes και απαιτεί 3 περιόδους ρολογιού. Το μειονέκτημα είναι ότι η ρουτίνα πρέπει να βρίσκεται εντός +/- 2k λέξεων. Άλλη εντολή είναι η CALL. Αυτή εκτελεί άλμα σε απόλυτη διεύθυνση, γι'αυτό έχει μήκος 4 bytes, και απαιτεί 4 περιόδους ρολογιού. Όμως επιτυγχάνει πρόσβαση σε όλο το χώρο κώδικα, γεγονός απαραίτητο σε μοντέλα με μνήμη προγράμματος άνω των 8kB. Οι 8k AVRs χρειάζονται μόνο τις RJMP και RCALL, αφού όλες οι διευθύνσεις είναι προσβάσιμες με άλματα +/- 2k λέξεων. Επίσης, υπάρχει η ICALL για έμμεση κλήση μέσω της διεύθυνσης στο ζεύγος καταχωρητών Z.
Για να γνωρίζουμε πού βρίσκονται οι διευθύνσεις κλήσης και επιστροφής μιας ρουτίνας πρέπει να θέσουμε το δείκτη στοίβας (SP). Ο SP συνήθως τίθεται στη τελευταία θέση μνήμης (RAMEND), κατά την αρχικοποίηση. Όταν η διεύθυνση επιστροφής αποθηκεύεται, ο SP δείχνει τη θέση μνήμης που προηγείται της αποθηκευμένης διεύθυνσης. Τα παραπάνω φαίνονται στο σχήμα 1.36.



Σχήμα 1.36 Λειτουργία στοίβας

Σε περίπτωση σύνδεσης εξωτερικής μνήμης SRAM, ο δείκτης τίθεται στη τελευταία θέση της εσωτερικής μνήμης, διότι η πρόσβαση στην εσωτερική μνήμη είναι ταχύτερη. Ακολουθεί ένα παράδειγμα που δείχνει πώς να θέτουμε τη στοίβα, να καλούμε ρουτίνες, να διαβάζουμε μια θύρα και να εκτελούμε άλματα συνθήκης.

Παράδειγμα 1.14: Δίνεται πρόγραμμα που όταν ένα από τα switches 0 και 1 είναι πατημένο ανάβουν τα αντίστοιχα leds, ενώ όταν είναι πατημένο ένα από τα switches 2-6 ανάβουν τα υπόλοιπα (leds 2 -7). Το switch 7 σβήνει όλα τα leds.

```
.INCLUDE "m16def.inc"          ; δήλωση ελεγκτή
.DEF   reg = R16               ; ορίζουμε όνομα καταχωρητή
    rjmp   main                ; εντολή άλματος στη διεύθυνση 0:
.org 0                         ; κυρίως πρόγραμμα
main:
    ldi reg,LOW(RAMEND)        ; αρχικοποίηση δείκτη στοίβας στη
    out SPL,reg
    ldi reg,HIGH(RAMEND)       ; τελευταία θέση της SRAM (RAMEND)
    out SPH,reg

    clr reg
    out DDRD,reg               ; PortD ως είσοδος
    ser reg
    out PORTD,reg              ; Port D (pull-ups)
    out DDRB,reg               ; PortB ως έξοδος
    out PORTB,reg              ; σβήσιμο leds (ανάστροφη λογική)

LOOP:
; ανάγνωση switch 0, δηλαδή portD (PIND 0) και εντολή παράκαμψης (SBIS)
; επόμενης εντολής αν το  bit 0 είναι 1 που λόγω της ανάστροφης λογικής των
; διακοπτών σημαίνει ότι δεν πατήθηκε. Διαφορετικά κλήση της ρουτίνας Light0.
    sbis   PIND,0
```

```
    rcall   Light0
; Ανάγνωση switch 1 με ένα διαφορετικό τρόπο:
; Οι θύρες I/O αντιστοιχούν επίσης και σε χώρο της μνήμης SRAM.
; Η διεύθυνση SRAM είναι 32 bytes υψηλότερα από την αντίστοιχη διεύθυνση θύρας.
; Οπότε η πρόσβαση σε μια θύρα μπορεί να επιτευχθεί με εντολές διευθυνσιοδότησης
; της SRAM.
.EQU   mem_name=PIND + $20  ; διεύθυνση στην sram

; Ορίζουμε το ζεύγος καταχωρητών R27:R26 ως καταχωρητή δείκτη αυτής της θύρας.
; Με την εντολή LoaD φορτώνουμε την τιμή της θύρας σε καταχωρητή σαν να είναι
; ένα SRAM byte.
    ldi R26,LOW(mem_name)
    ldi R27,HIGH(mem_name)
    ld reg,X                    ; φόρτωσε καταχωρητή reg από καταχωρητή δείκτη
    sbrs reg,1                  ; Έλεγχος Pin1 (switch 1- Bit 1 ). Αν είναι  0 που
    rcall Light1                ; σημαίνει ότι πατήθηκε, γίνεται κλήση της Light1.
 ; Αλλιώς παρακάμπτεται για να γίνει ο έλεγχος των Switches 2 ως 6.
    in reg,PIND         ; Ανάγνωση port D (Προσοχή στην ανάστροφη
    ori reg,0b10000011  ; λογική των διακοπτών).
    cpi reg,0b11111111   ; Εφαρμογή μάσκας στα switches 0, 1 και 7
    breq sw7            ; αν κανένας διακοπτής 2-6 δεν είναι ON τότε
                        ; άλμα στην ετικέτα sw7
    in reg,PIND         ; Αλλιώς ανάγνωση τρέχουσας κατάστασης LEDs
    andi reg,0b00000011         ; άναμμα των LED 2 ως 7
    out PORTB,reg       ; χωρίς να πειραχθούν τα LED 0 και 1
sw7:
    in reg,PIND         ; ανάγνωση θύρας διακοπτών
    rol reg             ; ολίσθηση 7ου bit στο κρατούμενο
    brcs endloop        ; αν 7ο bit είναι 1 (BRanch Carry Set) καμία
    ldi reg,0xFF        ; αλλαγή. Αλλιώς (C=0) σβήσιμο όλων των LEDs.
    out PORTB,reg
endloop:
    rjmp LOOP

Light0:                 ; υπορουτίνα Light0: LED 0 ON
    in reg,PIND         ; ανάγνωση τρέχουσας κατάστασης port B
    andi reg,0b11111110    ; άναμμα του 1ου LED τα υπόλοιπα μένουν
    out PORTB,reg       ; στην ίδια κατάσταση.
    ret

Light1:                 ; Υπορουτίνα Light1: LED 1 ON
    in reg,PIND         ; ανάγνωση κατάστασης port B
    cbr reg,0b00000010 ; θέτει bit 2 στο μηδέν (η εντολή αυτή ισοδυναμεί
    out PORTB,reg       ; με andi reg, 0b11111101). Ανάβει το 2ο LED και τα
    ret                 ; υπόλοιπα LED μένουν στην ίδια κατάσταση.
```

1.9 Μακροεντολές

Οι μακροεντολές είναι τμήματα κώδικα που γράφονται και ελέγχονται μια φορά και εισάγονται στο κυρίως πρόγραμμα μέσω του ονόματος τους. Οι μακροεντολές, όπως και οι υπορουτίνες, είναι χρήσιμες όταν χρειάζεται να γράψουμε πανομοιότυπα ή παρόμοια τμήματα κώδικα μέσα στο κύριο πρόγραμμα. Συνεπώς, γλιτώνουμε χρόνο, δίνουμε πιο απλή και κατανοητή μορφή στο πρόγραμμα μας και αποφεύγουμε τα άλματα που απαιτεί μια υπορουτίνα. Όμως, σε αντίθεση με τις υπορουτίνες, αυξάνει η απαίτηση σε μνήμη προγράμματος.

Επομένως, για μεγάλα τμήματα κώδικα ή όταν υπάρχει έλλειψη μνήμης προγράμματος προτιμούνται οι ρουτίνες. Μια μακροεντολή δέχεται ως δέκα παραμέτρους, οι οποίες συμβολίζονται ως @0-@9 και βρίσκονται εντός του ορισμού της. Όταν καλούμε μια μακροεντολή οι παράμετροι διαχωρίζονται με κόμμα.

Ορισμός μακροεντολής:

```
.MACRO Delay1          ; όνομα
    NOP                ; σώμα
    NOP
    NOP
    NOP
.ENDMACRO              ; τέλος
; Ο ορισμός της μακροεντολής δεν παράγει κώδικα. Ο κώδικας θα παραχθεί όταν
; καλέσουμε τη μακροεντολή με το όνομα της εντός του πηγαίου κώδικα:
   Delay1 ; όνομα μακροεντολής (σε αυτό το σημείο εισάγονται 4 εντολές NOP)
```

Παράδειγμα 1.15 : Άναμμα των LEDs 0, 1, 2 με χρήση μακροεντολής.

```
.INCLUDE "m16def.inc"
.LIST
.DEF temp=R18
.MACRO LedsOn          ; δήλωση μακροεντολής
    inc temp
    inc temp
    inc temp
    inc temp
.ENDMACRO
                       ; κύριο πρόγραμμα
    ser temp           ; ο καταχωρητής temp=0xFF
    out DDRB,temp      ; PortB (LEDs) ως έξοδος
    clr temp           ; μηδενισμός καταχωρητή temp
    LedsOn             ; εισαγωγή μακροεντολής (4 INCs)
    LedsOn             ; ξανά εισαγωγή μακροεντολής (άλλα 4 INCs)
    dec temp
    com temp           ; αναστροφή για απεικόνιση στα LEDs 0,1,2
    out PORTB,temp     ; άναμμα των leds
LOOP: rjmp LOOP
```

Παράδειγμα 1.16 : Επίδειξη χρήσης ετικέτας εντός μακροεντολής και άλματος σε ετικέτα έξω από μακροεντολή.

```
.INCLUDE "m16def.inc"
.DEF temp=R18

.MACRO TestMacro      ; δήλωση μακροεντολής
    inc  temp         ; αύξηση καταχωρητή temp
    brne macro_jump   ; παράκαμψη επόμενης εντολής αν δεν προκύψει υπερχείλιση
    rjmp overflow     ; άλμα σε περίπτωση υπερχείλισης σε ετικέτα εκτός
                      ; μακροεντολής
macro_jump:           ; Ετικέτα εντός μακροεντολής
.ENDMACRO
.LISTMAC              ; κύριο πρόγραμμα
    ser temp          ; PortB (LEDs) ως έξοδος
    out DDRB,temp
    ldi temp,0xFE     ; θέτουμε καταχωρητή=254
    testmacro         ; εισαγωγή μακροεντολής (ένα INC)
    testmacro         ; ξανά εισαγωγή μακροεντολής (άλλο ένα INC)
; Σωστή λειτουργία των μακροεντολών θα προκαλέσει υπερχείλιση (λόγω INC),
; οπότε η επόμενη εντολή δε θα εκτελεστεί. Εάν εκτελείτο θα άναβε όλα τα LEDs.
outp:
    out PORTB,temp    ; άναμμα των leds
LOOP:                 ; λόγω υπερχείλισης θα εκτελεστεί ο παρακάτω κώδικας και
    rjmp LOOP         ; τα LEDs PB.0 - PB.7 θα σβήσουν
overflow:
    ser temp          ; σβήσιμο των LEDs
    out PORTB,temp
    rjmp loop
```

Παράδειγμα 1.17 : Το επόμενο πρόγραμμμα δείχνει τη χρήση παραμέτρων σε μακροεντολή (ίδιο με προηγούμενο, αλλά με πέρασμα παραμέτρου).

```
.INCLUDE "m16def.inc"
.DEF temp=R18

.MACRO TestMacro      ; δήλωση μακροεντολής θέτουμε την
    ldi temp,@0       ; πρώτη παράμετρο(@0 ως τιμή του καταχωρητή temp)
    inc temp          ; αύξηση καταχωρητή temp
    brne macro_jump   ; παράκαμψη επόμενης εντολής αν δεν προκύψει υπερχείλιση
    rjmp overflow     ; Αλλιώς άλμα σε περίπτωση υπερχείλισης σε ετικέτα εκτός
macro_jump:           ; μακροεντολής
.ENDMACRO
.LISTMAC              ; κύριο πρόγραμμα
    ser temp          ; PortB (LEDs) ως έξοδος
    out DDRB,temp
```

```
; πέρασμα τιμής 0xFF στη μακροεντολή, αύξηση κατά μια μονάδα και άλμα στην
; overflow αφού προέκυψε υπερχείλιση και εισαγωγή μακροεντολής
    testmacro(0xff)
; Σωστή λειτουργία των μακροεντολών θα προκαλέσει υπερχείλιση (λόγω INC),
; οπότε η επόμενη εντολή δε θα εκτελεστεί. Εάν εκτελείτο θα άναβε όλα τα LEDs.
outp:
    out PORTB,temp      ; άναμμα leds
LOOP:
    rjmp LOOP

overflow:               ; λόγω υπερχείλισης θα εκτελεστεί ο παρακάτω
    ser temp            ; κώδικας και τα LEDs PB.0 - PB.7 θα σβήσουν
    out PORTB,temp
    rjmp LOOP
```

1.10 Καταχωρητές θυρών Ε/Ε

Ο παρακάτω πίνακας περιλαμβάνει τις πιο συνηθισμένες θύρες-καταχωρητές των μικροελεγκτών AVR:

Πίνακας 1.5 Θύρες-καταχωρητές των μικροελεγκτών AVR

Μονάδα - Σύμβολο	**Καταχωρητής**	**Σύμβολο**
Accumulator - SREG	Status Register	SREG
Stack - SPH/SPL	Stackpointer	SPH/SPL
Ext.SRAM/ Ext.Interrupt-MCUCR	MCU General Control Register	MCUCR
External Interrupt - INT	Interrupt Mask Register	GIMSK
	Flag Register	GIFR
Timer Interrupts	Timer Int Mask Register	TIMSK
	Timer Interrupt Flag Register	TIFR
Timer-0	Timer/Counter-0 Control Register	TCCR0
	Timer/Counter-0	TCNT0
Timer-1	Timer/Counter Control Register1A	TCCR1A
	Timer/Counter Control Register1B	TCCR1B
	Timer/Counter-1	TCNT1
	Output Compare Register 1 A	OCR1A
	Output Compare Register 1 B	OCR1B
	Input Capture Register	ICR1L/H
Watchdog Timer - WDT	Watchdog Timer Control Register	WDTCR
EEPROM	EEPROM Adress Register	EEAR
	EEPROM Data Register	EEDR
	EEPROM Control Register	EECR
SPI	Serial Peripheral Control Register	SPCR
	Serial Peripheral Status Register	SPSR
	Serial Peripheral Data Register	SPDR
UART	UART Register	UDR
	UART Status Register Control Data	USR
	UART Register	UCR
	UART Baud Rate Register	UBRR
Analog Comparator	Analog Comparator Control and Status Register	ACSR
I/O-Ports	PORTx	Port Output Register
	DDRx	Port Direction Register
	PINx	Port Input Register

Καταχωρητής κατάστασης του μικροελεγκτή (MCUSR ή MCUCSR)

Παρέχει πληροφορίες για την πηγή επανατοποθέτησης (reset) που προκάλεσε την επανατοποθέτηση της MCU.

Bit	7	6	5	4	3	2	1	0
Bits κατ/σης	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF

Η σημασία των bits του καταχωρητή είναι η εξής:

- Bit4 (JTRF: JTAG Reset Flag): Αυτό το bit τίθεται όταν προκύπτει επανατοποθέτηση από λογικό '1' στον καταχωρητή JTAG Reset μέσω της JTAG εντολής AVR_RESET. Μηδενίζεται με εμφάνιση ενός Power-on Reset ή με εγγραφή μηδενικού στη σημαία.
- Bit3 (WDRF: Watchdog Reset Flag): Αυτό το bit τίθεται όταν προκύπτει επανατοποθέτηση του χρονιστή επιτήρησης. Μηδενίζεται με εμφάνιση ενός Power-on Reset ή με εγγραφή μηδενικού στη σημαία.
- Bit2 (BORF: Brown-out Reset Flag): Αυτό το bit τίθεται όταν προκύπτει Brown-out Reset. Μηδενίζεται με εμφάνιση ενός Power-on Reset ή με εγγραφή μηδενικού στη σημαία.
- Bit1 (EXTRF: External Reset Flag): Αυτό το bit τίθεται όταν προκύπτει External Reset. Μηδενίζεται με εμφάνιση ενός Power-on Reset ή με εγγραφή μηδενικού στη σημαία.
- Bit0 (PORF: Power-on Reset Flag): Αυτό το bit τίθεται όταν προκύπτει Power-on Reset. Μηδενίζεται με εγγραφή μηδενικού στη σημαία.

Για να κάνουμε χρήση των παραπάνω σημαιών, ώστε να προσδιορίσουμε την αιτία επανατοποθέτησης, πρέπει να διαβάσουμε και να μηδενίσουμε τον MCUCSR όσο το δυνατόν νωρίτερα μέσα στο πρόγραμμα. Εάν ο καταχωρητής μηδενιστεί προτού συμβεί άλλη επανατοποθέτηση, τότε εξετάζοντας τις σημαίες συμπεραίνουμε αν και ποια επανατοποθέτηση συνέβη.

Τιμές των EXTRF - PORF μετά από επανεκκίνηση, Y=αμετάβλητη τιμή		
EXTRF	PORF	Λόγος επανεκκίνησης
X	1	Εφαρμογή τάσης τροφοδοσίας
1	Y	Εξωτερικός χειρισμός
Y	1	Υπερχείλιση του χρονιστή επιτήρησης WDT

1.11 Διακοπές

Η εμφάνιση μιας διακοπής σηματοδοτεί ότι προέκυψε ένα γεγονός που ο επεξεργαστής θα πρέπει να το χειριστεί ξεχωριστά, διακόπτοντας την κανονική ροή του προγράμματος. Το γεγονός αυτό μπορεί να είναι ένα σήμα σε κάποιες ειδικές -για το σκοπό αυτό- ακίδες του μικροελεγκτή ή σε κάποια περιφερειακή μονάδα, η υπερχείλιση ενός καταχωρητή ή ενός μετρητή - χρονιστή, το πάτημα ενός διακόπτη, η λήψη δεδομένων σε μια θύρα κλπ., που η διαχείρισή του γίνεται μέσω της κλήσης και εκτέλεσης μιας ρουτίνας που ονομάζεται ρουτίνα εξυπηρέτησης διακοπής (ISR-Interrupt Service Routine). Τα βήματα κατά την εμφάνιση μιας διακοπής είναι τα εξής:

- Μια περιφερειακή μονάδα (μέσω μιας σημαίας) ειδοποιεί ότι προέκυψε διακοπή.
- Η εντολή που βρίσκεται υπό εκτέλεση στον καταχωρητή εντολών (IR) ολοκληρώνεται.
- Η διεύθυνση της επόμενης εντολής του κυρίου προγράμματος σώζεται στη στοίβα.
- Η διεύθυνση της ρουτίνας εξυπηρέτησης διακοπής φορτώνεται στο μετρητή προγράμματος (PC-Program Counter) μέσω μιας εντολής icall, rcall κλπ., ώστε να εκτελεστεί η $1^{η}$ εντολή της ρουτίνας εξυπηρέτησης διακοπής.
- Οι εντολές της ρουτίνας εξυπηρέτησης διακοπής εκτελούνται διαδοχικά μέχρι και την τελευταία εντολή τύπου reti, όπου η διεύθυνση επιστροφής φορτώνεται από τη στοίβα και η σημαία ολικών διακοπών τίθεται.
- Η εκτέλεση του κυρίου προγράμματος συνεχίζεται κανονικά.

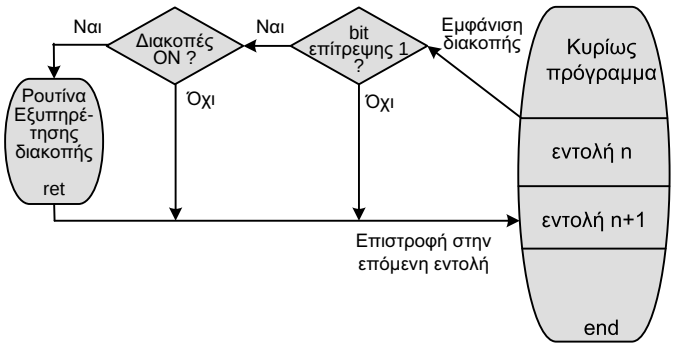
Όταν προκύπτει μια συγκεκριμένη διακοπή, με βάση το αριθμό του διανύσματος διακοπής εκτελείται η αντίστοιχη σε σειρά εντολή άλματος στη ρουτίνα εξυπηρέτησής της. Ο αριθμός και η ονομασία του κάθε διανύσματος διακοπής βρίσκεται στους Πίνακες 1.8 και 1.9.

Η διαδικασία της διακοπής απαιτεί 4 κύκλους ρολογιού που στη διάρκειά τους φορτώνεται ο μετρητής προγράμματος στη στοίβα (1 κύκλος) και γίνεται άλμα στη ρουτίνα εξυπηρέτησης (3 κύκλοι). Η επιστροφή στο κύριο πρόγραμμα επίσης απαιτεί 4 κύκλους ρολογιού (ανάκτηση από τη στοίβα του PC, αύξηση του δείκτη στοίβας κατά 2 και το I-bit του καταχωρητή SREG γίνεται ON).

Οι μικροελεγκτές AVR διαθέτουν ένα ολοκληρωμένο σύνολο διακοπών με δυνατότητα ενεργοποίησης-απενεργοποίησης. Αυτό σημαίνει ότι μπορούμε να ενεργοποιούμε ή να απενεργοποιούμε κάποιες λειτουργίες των περιφερειακών μονάδων κατά βούληση.

Κατά την εκτέλεση μιας ρουτίνας εξυπηρέτησης διακοπής ενδέχεται να αλλοιωθούν οι σημαίες του καταχωρητή κατάστασης (SREG) καθώς επίσης και κάποιοι καταχωρητές εργασίας. Το γεγονός αυτό ίσως επηρεάσει την ομαλή λειτουργία του κυρίου προγράμματος. Για το λόγο αυτό οι προηγούμενοι

καταχωρητές αποθηκεύονται στην αρχή της ρουτίνας και ανακτώνται στο τέλος της (διαδικασίες PUSH και POP).
Επίσης, ενδέχεται κατά τη διάρκεια εκτέλεσης μιας ρουτίνας εξυπηρέτησης διακοπής να προκύψει νέα διακοπή. Για να αποφύγουμε μια τέτοια κατάσταση απενεργοποιούμε τις καθολικές διακοπές του συστήματος μέσω της εντολής CLI στην αρχή της ρουτίνας και τις ενεργοποιούμε στο τέλος μέσω της εντολής SEI. Δηλαδή, μηδενίζουμε ή θέτουμε το bit GIE (Global Interrupt Enable) του SREG αντιστοίχως. Το επόμενο απλοποιημένο διάγραμμα ροής απεικονίζει την εξυπηρέτηση μιας διακοπής:



Σχήμα 1.37 Διάγραμμα ροής για εξυπηρέτηση διακοπής

Γενικός καταχωρητής μάσκας διακοπών (GIMSK)

Ο γενικός καταχωρητής μάσκας διακοπών (General Interrupt MaSK register- GIMSK) ενεργοποιεί ή απενεργοποιεί τις εξωτερικές διακοπές θέτοντας ή μηδενίζοντας τα bits ελέγχου του. Προϋπόθεση αποτελεί η ενεργοποίηση της σημαίας καθολικών διακοπών (I) του καταχωρητή SR (Status Register).

Bit	7	6	5	4	3	2	1	0	**GIMSK**
Bits ελέγχου	INT1	INT0	INT2	-	-	-	-	-	

Γενικός καταχωρητής σημαίας διακοπών (GIFR)

Ο γενικός καταχωρητής σημαίας διακοπών (General Interrupt Flag register- GIFR) ενημερώνει το σύστημα αν συνέβη κάποια διακοπή. Τότε τίθεται το bit του καταχωρητή. Μετά την εξυπηρέτηση της διακοπής η σημαία μηδενίζεται.

Bit	7	6	5	4	3	2	1	0	**GIFR**
Bits ελέγχου	INTF1	INTF0	INTF2	-	-	-	-	-	

Γενικός καταχωρητής ελέγχου (MCUCR)
Ο γενικός καταχωρητής ελέγχου (MiCroprocessor Unit Control Register MCUCR) επιτρέπει τον έλεγχο κάποιων λειτουργιών του επεξεργαστή.

Bit	7	6	5	4	3	2	1	0	**MCUCR**
Λειτουργία	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	

Η σημασία των bits του καταχωρητή είναι η εξής:

- Bit7, 5 και 4 (SM2, SM1, SM0): Τα bit αυτά καθορίζουν ένα από τα επόμενα έξι (6) Sleep Mode που μπορεί να τεθεί ο μικροελεγκτής όταν εκτελείται η εντολή **SLEEP**.

Πίνακας 1.6 Καθορισμός των 6 Sleep Mode στους μικροελεγκτές AVR

SM2	**SM1**	**SM0**	**Sleep Mode**
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Φυλάσσεται για μελλοντική χρήση
1	0	1	Φυλάσσεται για μελλοντική χρήση
1	1	0	Standby
1	1	1	Extended Standby

1. Idle Mode: Στην κατάσταση αυτή σταματάει η CPU αλλά επιτρέπεται η λειτουργία των SPI, USART, Analog Comparator, ADC, Twowire Serial Interface, Timer/Counters, Watchdog, και του συστήματος διακοπών. Αυτό το sleep mode σταματάει clkCPU and clkFLASH, ενώ αφήνει τα άλλα ρολόγια να τρέχουν.
Το Idle mode επιτρέπει στο Μικροελεγκτή να ξυπνήσει με εξωτερική διακοπή καθώς και με εσωτερική διακοπή όπως Timer Overflow, USART Transmit Complete interrupts.
Αν δεν απαιτείται ξύπνημα από Analog Comparator αυτό μπορεί να τεθεί εκτός τροφοδοσίας μέσω του bit ACD (=1) του καταχωρητή Analog Comparator Control and Status Register – ACSR.
Αν το ADC είναι ενεργοποιημένο, μια μετατροπή αρχίζει αυτόματα όταν μπαίνουμε στην κατάσταση αυτή.

2. ADC Noise Reduction Mode: Στην κατάσταση αυτή σταματάει η CPU αλλά επιτρέπεται η λειτουργία των ADC, External Interrupts, Two-wire Serial Interface address watch, Timer/Counter2 και του Watchdog (αν είναι ενεργοποιημένα). Στην κατάσταση αυτή σταματούν τα ρολόγια clkI/O, clkCPU και clk-FLASH, ενώ τα άλλα λειτουργούν. Αυτό βελτιώνει τις συνθήκες θορύβου του ADC, επιτρέποντας έτσι μετρήσεις υψηλότερης ακρίβειας.
Αν το ADC είναι ενεργοποιημένο, μια μετατροπή αρχίζει αυτόματα όταν μπαίνουμε στην κατάσταση αυτή. Εκτός από τη διακοπή ADC Conversion Complete, επίσης: εξωτερικό Reset, Watchdog Reset, Brown-out Reset, Two-wire Serial Interface address match interrupt, Timer/Counter2 interrupt, SPM/EEPROM ready interrupt, εξωτερική επιπέδου

τάσης διακοπή INT0 ή INT1, ή και εξωτερική διακοπή INT2 μπορούν να ξυπνήσουν το Μικροελεγκτή από το ADC Noise Reduction mode.

3. Power-down Mode: Στην κατάσταση αυτή ο Εξωτερικός Ταλαντωτής σταματάει ενώ οι εξωτερικές διακοπές, το Two-wire Serial Interface (I^2C) address watch, και ο χρονιστής Watchdog συνεχίζουν να λειτουργούν (αν είναι ενεργοποιημένα). Μόνο εξωτερικό Reset, Watchdog Reset, Brown-out Reset, Two-wire Serial Interface address match interrupt, εξωτερική επιπέδου τάσης διακοπή INT0 ή INT1, ή και εξωτερική διακοπή INT2 μπορούν να ξυπνήσουν το Μικροελεγκτή. Στην κατάσταση αυτή σταματούν όλα τα παραγόμενα ρολόγια και αφήνεται η λειτουργία μόνο των ασύγχρονων μονάδων. Για να ξυπνήσει απαιτείται χρόνος κάποιων κύκλων ρολογιού.

4. Power-save Mode: Είναι ίδιο με το Power- down mode με τη διαφορά ότι ο Timer/Counter2 οδηγείται με ασύγχρονο ρολόι, π.χ. όταν το bit AS2 του καταχωρητή ASSR είναι 1, ο Timer/Counter2 συνεχίζει να λειτουργεί. Στην κατάσταση αυτή ο Μικροελεγκτής μπορεί να ξυπνήσει από διακοπή του Timer /Counter2.
Αν δεν έχουμε ασύγχρονη λειτουργία συνιστάται η χρήση της Power-down mode.

5. Standby Mode: Είναι ίδιο με το Power- down mode με τη διαφορά ότι ο ταλαντωτής δε σταματάει. Ξυπνάει σε 6 κύκλους ρολογιού.

6. Extended Standby Mode: Είναι ίδιο με το Power-save mode με τη διαφορά ότι ο ταλαντωτής δε σταματάει. Ξυπνάει σε 6 κύκλους ρολογιού.

Η σημασία των υπολοίπων bits του καταχωρητή MCUCR είναι η εξής:

- Bit6 (SE): Ενεργοποίηση ανενεργού κύκλου (Sleep mode Enable). Όταν το bit έχει τεθεί τότε μπορεί να εκτελεστεί η εντολή sleep.
- Bit3, Bit2 (ISC11, ISC10): Αυτά τα 2 bits προσδιορίζουν τον τρόπο διέγερσης από την εξωτερική διακοπή INT1.
- Bit1, Bit0 (ISC01, ISC00): Αυτά τα 2 bits προσδιορίζουν τον τρόπο διέγερσης από την εξωτερική διακοπή INT0.

Πίνακας 1.7 Μορφή σήματος για πρόκληση διακοπής στους μικροελεγκτές AVR

ISC01 - ISC00 ή ISC11 – ISC10	**ΠΕΡΙΓΡΑΦΗ : Προκαλείται αίτηση διακοπής όταν το σήμα στην ακίδα INT0 ή INT1 έχει:**
0 0	Χαμηλή Στάθμης
0 1	Υψηλή Στάθμης
1 0	Κατερχόμενη Ακμή
1 1	Ανερχόμενη Ακμή

Οι εξωτερικές διακοπές προκαλούνται με κατάλληλα σήματα στις ακίδες INT0, INT1, και INT2. Αξίζει να σημειωθεί ότι ακόμα και αν οι ακίδες INT0, INT1, και INT2 οριστούν ως έξοδοι, διακοπή μπορεί να προκληθεί. Η ακίδα INT2 είναι ακμοπυροδότητη (στο θετικό ή στο αρνητικό μέτωπο και ορίζεται μέσω των καταχωρητών MCUCR και MCUCSR) ενώ τα άλλα δυο προκαλούν διακοπή σε χαμηλή στάθμη σήμα εισόδου.

Διανύσματα διακοπών μικροελεγκτών AVR

Διακοπή μπορεί να προκληθεί από εξωτερικό σήμα αλλά και εσωτερικά από κάποια περιφερειακή συσκευή του μικροελεγκτή. Τα διανύσματα όλων αυτών των διακοπών ενός μικροελεγκτή ξεκινούν από τη διεύθυνση 0x0000, με πρώτο το διάνυσμα επανατοποθέτησης (reset).

Τα διανύσματα διακοπών που αφορούν στους μικροελεγκτές AVR με μικρό χώρο μνήμης π.χ. AT8515 δίνονται στην αρχή του προγράμματος, όπως παρακάτω:

```
.org 0x0000
    rjmp reset              ; $0000 Reset
    rjmp EXT_INT0           ; $0001 IRQ0
    rjmp EXT_INT1           ; $0002 IRQ1
    rjmp TIMER1_CAPT        ; $0003 Σύλληψη Timer1
    rjmp TIMER1_COMPA       ; $0004 Σύγκριση Timer1
    rjmp TIMER1_COMPB       ; $0005     >>
    rjmp TIMER1_OVF         ; $0006 Υπερχείλιση Timer1
    rjmp TIMER0_OVF         ; $0007 Υπερχείλιση Timer0
    rjmp SPI_STC            ; $0008 Μονάδα SPI
    rjmp UART_RXC           ; $0009 Ολοκλήρωση λήψης UART
    rjmp UART_DRE           ; $000A UDR άδειος
    rjmp UART_TXC           ; $000B Ολοκλήρωση εκπομπής UART
    rjmp ANA_COMP           ; $000C αναλογικός συγκριτής
    rjmp start
```

Το γενικό πρόγραμμα εγκατάστασης Reset και διευθύνσεων διανυσμάτων διακοπών στο Μικροελεγκτή ATmega16 δίνεται παρακάτω:

```
$000  jmp RESET             ; Reset Handler
$002  jmp EXT_INT0          ; IRQ0 Handler
$004  jmp EXT_INT1          ; IRQ1 Handler
$006  jmp TIM2_COMP         ; Timer-2 Compare Handler
$008   jmp TIM2_OVF         ; Timer-2 Overflow Handler
$00A  jmp TIM1_CAPT         ; Timer-1 Capture Handler
$00C  jmp TIM1_COMPA        ; Timer-1 CompareA Handler
$00E  jmp TIM1_COMPB        ; Timer-1 CompareB Handler
$010   jmp TIM1_OVF         ; Timer-1 Overflow Handler
$012   jmp TIM0_OVF         ; Timer-0 Overflow Handler
$014   jmp SPI_STC          ; SPI Transfer Complete Handler
$016   jmp USART_RXC        ; USART RX Complete Handler
$018   jmp USART_UDRE       ; UDR Empty Handler
$01A  jmp USART_TXC         ; USART TX Complete Handler
$01C  jmp ADC               ; ADC Conversion Complete Handler
$01E  jmp EE_RDY            ; EEPROM Ready Handler
$020   jmp ANA_COMP         ; Analog Comparator Handler
$022   jmp TWSI             ; Two-wire Serial Interface Handler
$024   jmp EXT_INT2         ; IRQ2 Handler
$026   jmp TIM0_COMP        ; Timer-0 Compare Handler
$028   jmp SPM_RDY          ; Store Program Memory Ready Handler
```

Ο επόμενος πίνακας περιλαμβάνει τα διανύσματα διακοπών και τη λειτουργία τους για το μικροελεγκτή ATmega16. Στη σειρά αυτή (Mega) που διαθέτει μεγαλύτερο χώρο μνήμης προγράμματος (16KB) χρησιμοποιείται υποχρεωτικά η εντολή JMP η οποία καταλαμβάνει 2 θέσεις (της μνήμης προγράμματος) ενώ στον προηγούμενο τύπο (AT8515) χρησιμοποιείται η εντολή RJMP (που καταλαμβάνει μια μόνο θέση).

Πίνακας 1.8 Διανύσματα διακοπών στη μνήμη προγράμματος του μικροελεγκτή ATmega16

Αρ. Διαν.	**Διεύθυνση**	**Όνομα**	**Ορισμός διακοπής**
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

Φυσικά, στο εκάστοτε πρόγραμμα θα περιλαμβάνονται μόνο οι εντολές άλματος των διακοπών που χρησιμοποιούνται. Στις υπόλοιπες θέσεις μπαίνει η εντολή ret, ώστε να είναι φυλαγμένες για πιθανή συμπλήρωση ή και λόγους προστασίας, ώστε αν ενεργοποιηθούν από σφάλμα, να μην έχουν καμία επίπτωση (αποτελούν εκφυλισμένη μορφή ρουτίνας εξυπηρέτησης που απλά επιστρέφει στο κύριο πρόγραμμα).

Πίνακας 1.9 Διανύσματα διακοπών στη μνήμη προγράμματος ενός μικροελεγκτή AVR

Όνομα	Διεύθυνση μνήμης προγράμματος		Σχόλια
	AT90S 8515	ATmega16	
Reset	$0000	$000	Διαχείριση επανεκκίνησης
EXT_INT0	$0001	$002	Διαχείριση εξωτερικής διακοπής IRQ0
EXT_INT1	$0002	$004	Διαχείριση εξωτερικής διακοπής IRQ1
TIMER2 COMP	-	$006	Timer/Counter2 Compare Match
TIMER2 OVF	-	$008	Timer/Counter2 Overflow
TIMER1_CAPT	$0003	$00A	Διαχείριση διακοπής σε λειτουργία σύλληψης του χρονιστή timer1
TIMER1_COMPA	$0004	$00C	Διαχείριση διακοπής σε λειτουργία συγκριτή A του χρονιστή timer1
TIMER1_COMPB	$0005	$00E	Διαχείριση διακοπής σε λειτουργία συγκριτή B του χρονιστή timer1
TIMER1_OVF	$0006	$010	Διαχείριση διακοπής υπερχείλισης του χρονιστή timer1
TIMER0_OVF	$0007	$012	Διαχείριση διακοπής υπερχείλισης του χρονιστή timer0
SPI_STC	$0008	$014	Διαχείριση διακοπής κατά την ολοκλήρωση της μετάδοσης από τη μονάδα SPI
UART_RXC	$0009	$016	Διαχείριση διακοπής κατά την ολοκλήρωση της λήψης από τη μονάδα UART
UART_DRE	$000A	$018	Διαχείριση διακοπής κατά την εκκένωση του καταχ/τή UDR της μονάδας UART
UART_TXC	$000B	$01A	Διαχείριση διακοπής κατά την ολοκλήρωση της εκπομπής από τη μονάδα UART
ADC ADC	-	$01C	Conversion Complete
EE_RDY	-	$01E	EEPROM Ready
ANA_COMP	$000C	$020	Διαχείριση διακοπής του αναλογικού συγκριτή
TWI	-	$022	Two-wire Serial Interface
INT2	-	$024	External Interrupt Request 2
TIMER0 COMP	-	$026	Timer/Counter0 Compare Match
SPM_RDY	-	$028	Store Program Memory Ready

Παράδειγμα 1.18: Στο πρόγραμμα αυτό, κάθε φορά που προκαλείται εξωτερικής διακοπής INT0 (ακροδέκτης PD2) αναστρέφεται ο φωτισμός των led εξόδου (που θεωρούμε ότι συνδέονται στη θύρα εξόδου PORTB).

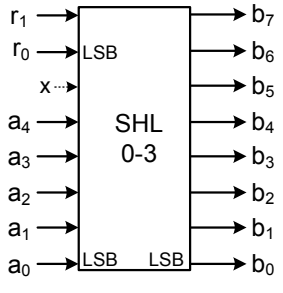
```
.include "m16def.inc"
.def temp = r16
.def leds = r17
    jmp reset                   ; Reset Handler
    jmp interrupt0              ; IRQ0 Handler
    jmp interrupt1              ; IRQ1 Handler
    reti                        ; Υπόλοιποι Handlers

reset:
    ldi temp,high(RAMEND)       ; κύριο πρόγραμμα
    out SPH,temp                ; θέτουμε δείκτη στοίβας στην RAM
    ldi temp,low(RAMEND)
    out SPL,temp

    ldi temp, 0xFF
    out DDRB, temp              ; PORTB  ως έξοδο των leds

    ldi leds, 0b11111011        ; άναμμα led2
    out PORTB, leds

    ldi temp, 1<<INT0           ; 0100 0000 (INT0=6)
    out GIMSK, temp             ; ενεργοποίηση εξωτερικής διακοπής 0

    ldi temp, 0b00000010        ; ορίζουμε η εξωτερική διακοπή INT0 να
    out MCUCR, temp             ; προκαλείται στην ακμή πτώσης (βλ. Πίνακα 1)
    sei                         ; ενεργοποίηση συνολικά των διακοπών
LOOP:
    rjmp LOOP                   ; αναμονή εξωτερικής διακοπής (πάτημα διακόπτη 2)

interrupt0:                     ; ρουτίνα εξυπηρέτησης διακοπής INT0
    com leds                    ; προέκυψε διακοπή, τίθεται INTF0
    out PORTB, leds             ; επίδειξη στα LEDs
    reti
```

Ασκήσεις προς λύση

1.1 Να γραφτεί πρόγραμμα σε assembly το οποίο θα ανάβει για 2 δευτερόλεπτα 8 leds (που θεωρούμε ότι είναι συνδεδεμένα με τα bit της θύρας PortB) και θα τα κρατάει σβηστά για 1 δευτερόλεπτο. Το πρόγραμμα θα είναι συνεχούς λειτουργίας. Να υπολογίσετε τη χρονο-καθυστέρηση υποθέτοντας 4MHz το ρολόι του μικροελεγκτή AVR;

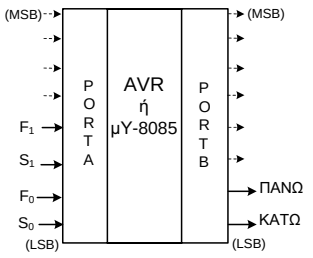
1.2 Να υλοποιηθούν σε μικροελεγκτή AVR οι λογικές συναρτήσεις:
F0=AB'CD' , F1=AB'CD'E'F' , F2= F0+F1 και να απεικονιστούν με θετική λογική (αναμμένο λογικό 1, σβηστό λογικό 0) στα led0, led1 και led2 της θύρας PortB αντίστοιχα. Οι μεταβλητές εισόδου (A, B, C, D, E, F) υποθέτουμε ότι αντιστοιχούν στα bit της θύρας PortD (0-5).

1.3 Σε ένα Μικροελεγκτή AVR να γραφεί πρόγραμμα σε Assembly που να προσομοιώνει τη λειτουργία ενός ολισθητή από 0 έως 3 βήματα. Τα σήματα a0, a1, a2, a3, a4, r0 και r1 υποθέτουμε ότι αντιστοιχούν σε 7 bit μιας θύρας εισόδου (PORTA) και το αποτέλεσμα $B=A\times 2^R$ (όπου A=a4a3a2a1a0, B=b7b6b5b4b3b2b1b0, R=r1r0) εξάγεται από τη θύρα εξόδου (PORTB) με ολίσθηση αριστερά κατά R θέσεις. Υποθέτουμε ότι η προσομοίωση λειτουργεί συνεχώς.



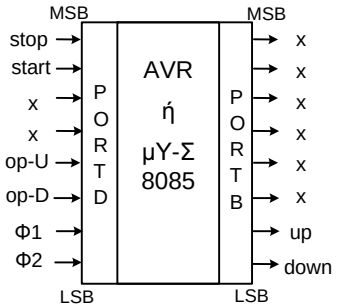
1.4 Να γραφεί πρόγραμμα σε Assembly του Μικροελεγκτή AVR για τη μέτρηση των εξωτερικών διακοπών τύπου INT1 (pin PD3 της θύρας PortD) που συνδέεται στο διακόπτη sw3. Να ληφθεί πρόνοια ώστε πιθανοί σπινθηρισμοί του διακόπτη να μην επηρεάζουν την ορθότητα της καταμέτρησης των διακοπών. **Υπόδειξη:** Να περιληφθεί ρουτίνα χρονοκαθυστέρησης της τάξεως των 20-30 msec για να αποκλειστούν τα μεταβατικά φαινόμενα.

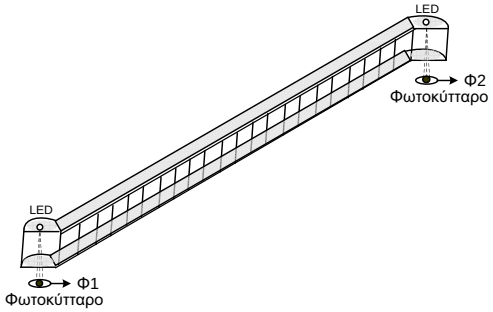
1.5 Σε ένα Μικροελεγκτή AVR να γραφεί πρόγραμμα σε Assembly που να επιτρέπει τον έλεγχο ενός ανελκυστήρα δυο θέσεων (ισογείου και 1ου ορόφου). Η κίνηση προς το ισόγειο ή τον 1ου όροφο ελέγχεται από τους διακόπτες (Push-Buttons) F0 και F1 αντίστοιχα με την προϋπόθεση ότι είναι σταματημένος. Όταν κινείται σταματάει από τους διακόπτες S0 και S1 που είναι τερματικοί διακόπτες και οι οποίοι



δίνουν λογικό 1 όταν ο θάλαμος φτάσει στο ισόγειο ή τον $1^{ου}$ όροφο αντίστοιχα. Υποθέτουμε ότι κατά την εκκίνηση του συστήματος ο θάλαμος πρέπει να βρίσκεται στο ισόγειο αλλιώς πριν δεχτεί οποιαδήποτε εντολή να μεταφέρεται σε αυτή τη θέση αυτόματα. Να δοθεί επίσης και το Διάγραμμα Ροής του συστήματος.

1.6 Σε ένα Μικροελεγκτή AVR που διαθέτει πόρτα εισόδου PORTD και πόρτα εξόδου PORTB να υλοποιηθεί ένα σύστημα οδήγησης κυλιόμενης σκάλας και η οποία θα ενεργοποιείται από δύο φωτοκύτταρα (Φ1, Φ2). Συγκεκριμένα, θεωρούμε πως όταν ένας επιβάτης εισέρχεται στη σκάλα, όταν είναι ακίνητη, διακόπτει δέσμη φωτός (γίνεται Φ1 ή Φ2=0) και τότε τίθεται σε κίνηση η σκάλα με τα σήματα εξόδου up ή down ανάλογα με το φωτοκύτταρο. Το φωτοκύτταρο που διακόπηκε αν παραμένει 1 δηλ. δεν περάσει άλλος επιβάτης για χρονικό διάστημα 10 sec (χρόνος για να αδειάσει η σκάλα) τότε η κίνηση να σταματάει. Να δοθεί ρουτίνα χρονοκαθυστέρησης DSEC που να δημιουργεί καθυστέρηση 100 msec. Επίσης ένας χειριστής να μπορεί να το σταματάει με το stop και να το οδηγεί πάνω ή κάτω με τα op-U, op-D και θα επανέρχεται σε κανονική λειτουργία με το πάτημα του start. Να δοθεί και το Διάγραμμα Ροής του συστήματος.





ΠΑΡΑΡΤΗΜΑ 1

Περιγραφή ATMEGA16

Χαρακτηριστικά

• 8-bit AVR μικροελεγκτής αρχιτεκτονική RISC
– 131 εντολές, 32 x 8 καταχωρητές, Έως 16 MIPS απόδοση. Ενσωματωμένος πολλαπλασιαστής

• Μνήμη προγράμματος και δεδομένων
– 16K Bytes από In-System Self-Programmable Flash, Αντοχή: 10,000 κύκλοι εγγραφής/ανάγνωσης
– Προαιρετικό τμήμα Boot Code με ανεξάρτητα bits κλειδώματος (Lock Bits)
In-System Programming από πρόγραμμα εκκίνησης (ενσωματωμένο πρόγραμμα Boot)
– 512 Bytes EEPROM, Αντοχή: 100,000 κύκλοι εγγραφής/ανάγνωσης
– 1K Byte εσωτερικής μνήμης SRAM
– Programming Lock για ασφάλεια λογισμικού

• JTAG διεπαφή (IEEE std. 1149.1 συμβατή)
–JTAG Standard, υποστήριξη αποσφαλμάτωσης (Debug),
– Προγραμματισμός των Flash, EEPROM, ασφαλειών, και Bits κλειδώματος μέσω JTAG

• Περιφεριακά
– Δύο 8-bit Timer/Counters, Ένα 16-bit Timer/Counter, Μετρητή πραγματικού χρόνου
– χρονιστής επιτήρησης (Watchdog Timer), 4 PWM κανάλια, 8-κάναλο, Αναλογικός συγκριτής
–10-bit ADC, 8 απλά κανάλια, 2 διαφορικά κανάλια με προγραμματιζόμενο κέρδος 1x, 10x, ή 200x
–Two-wire σειριακή διεπαφή, Προγραμματιζόμενη σειριακή USART, Master/Slave SPI σειριακή

• Ειδικά χαρακτηριστικά
– Power-on Reset, εσωτερικός RC ταλαντωτής, Εξωτερικές και εσωτερικές πηγές διακοπών,
–6 λειτουργίες ηρεμίας: Idle, ADC Noise Reduction, Power-save, Power-down, Standby και Extended Standby

• I/O and Packages
– 32 Programmable I/O Lines, 40-pin PDIP, Ταχύτητα λειτουργίας: 0 - 16 MHz

• Κατανάλωση ισχύος για 1 MHz, 3V

– Δραστήρια κατάσταση: 1.1 mA, Power-down κατάσταση: < 1 μA, άεργη κατάσταση: 0.35 mA

Ο ATmega16 είναι ένας 8-bit μικροελεγκτής τύπου CMOS βασισμένος σε αρχιτεκτονική μειωμένου συνόλου εντολών (RISC). Εκτελεί εντολές εντός ενός κύκλου ρολογιού, επιτυγχάνοντας απόδοση 1 MIPS ανά MHz, ενώ η κατανάλωση ισχύος του διατηρείται σε χαμηλά επίπεδα.
Ο πυρήνας AVR συνδυάζει ένα πλήρες σύνολο εντολών με 32 γενικούς καταχωρητές εργασίας. Οι καταχωρητές αυτοί συνδέονται άμεσα στην αριθμητική λογική μονάδα, επιτρέποντας πρόσβαση σε 2

ανεξάρτητους καταχωρητές με μια απλή εντολή διάρκειας ενός παλμού ρολογιού. Κατά συνέπεια, η αρχιτεκτονική λειτουργεί με αποδοτικό κώδικα, έως δεκαπλάσιας απόδοσης από τους συμβατικούς μικροελεγκτές CISC.
Ο ATmega16 παρέχει τα ακόλουθα χαρακτηριστικά: 16K bytes In-System Programmable Flash Program μνήμη με Read-While-Write δυνατότητα, 512 bytes EEPROM, 1K byte SRAM, 32 γενικές γραμμές εισόδου-εξόδου I/O, 32 γενικούς καταχωρητές εργασίας, μια διεπαφή JTAG για λειτουργία Boundary-scan, υποστήριξη αποσφαλμάτωσης και προγραμματισμού, τρεις εύελικτους μετρητές/χρονιστές με λειτουργία σύγκρισης, εσωτερικές και εξωτερικές διακοπές, σειριακά προγραμματιζόμενη USART, μία σειριακή διεπαφή Two-wire, έναν 8-κάναλο των 10-bit μετατροπέα αναλογικής τάσης σε ψηφιακή μορφή με προαιρετική διαφορική είσοδο (A/D Converter-ADC), έναν προγραμματιζόμενο χρονιστή επιτήρησης με εσωτερικό ταλαντωτή, μια σειριακή θύρα SPI και 6 τρόπους χαμηλής κατανάλωσης που επιλέγονται μέσω λογισμικού.
Η άεργη κατάσταση (Idle mode) σταματά τη ΚΜΕ ενώ επιτρέπει τη λειτουργία των USART, διεπαφή Two-wire, μετατροπέα A/D, SRAM, Timer/Counters, θύρα SPI, και συστήματος διακοπών. Η κατάσταση Power-down αποθηκεύει τα περιεχόμενα των καταχωρητών, αλλά παγώνει τον ταλαντωτή απενεργοποιώντας κάθε άλλη λειτουργία, ωσότου προκύψει εξωτερική διακοπή ή επανατοποθέτηση (Hardware Reset). Σε κατάσταση Power-save, ο ασύγχρονος χρονιστής εξακολουθεί να τρέχει, επιτρέποντας στο χρήστη να διατηρεί μια βάση χρόνου ενώ η υπόλοιπη συσκεύη αδρανεί. Η κατάσταση μείωσης θορύβου ADC σταματά τη ΚΜΕ και όλες τις I/O εκτός από τον ασύγχρονο χρονιστή και τον ADC, ώστε να ελαχιστοποιήσει το θόρυβο μεταγωγής κατά τις μετατροπές ADC. Σε κατάσταση Standby, ο κρύσταλλος του ταλαντωτή (crystal/resonator Oscillator) τρέχει ενώ η υπόλοιπη συσκεύη αδρανεί, επιτρέποντας γρήγορες εκκινήσεις με χαμηλή κατανάλωση ισχύος. Σε Extended Standby κατάσταση, ο κύριος ταλαντωτής και ο ασύγχρονος χρονιστής συνεχίζουν να τρέχουν.
Το On-chip ISP Flash επιτρέπει στη μνήμη προγράμματος να επαναπρογραμματίζεται εντός-συστήματος μέσω της σειριακής διεπαφής SPI, από ένα συμβατικό προγραμματιστή μνήμης είτε από το On-chip Boot πρόγραμμα που τρέχει στον πυρήνα. Το boot πρόγραμμα μπορεί να χρησιμοποιήσει οποιαδήποτε διεπαφή ώστε να φορτώσει την εκάστοτε εφαρμογή στη μνήμη Application Flash. Το λογισμικό στο τμήμα Boot Flash θα συνεχίσει να τρέχει όσο ενημερώνεται το τμήμα Application Flash, παρέχοντας Read-While-Write λειτουργία. Συνδυάζοντας έναν 8-bit RISC ΚΜΕ με In-System Self-Programmable Flash, ο Atmel ATmega16 καθίσταται ένας ισχυρός μικροελεγκτής, παρέχοντας εύελικτες και φθηνές λύσεις σε ενσωματωμένες εφαρμογές.
Ο ATmega16 υποστηρίζεται από μια πλήρη σουίτα εργαλείων ανάπτυξης, όπως C compilers, macro assemblers, program debugger/simulators (περιλαμβάνονται στη πλατφόρμα AVR Studio), in-circuit emulators, και evaluation kits (όπως το STK 500).

Ο παρακάτω πίνακας περιλαμβάνει όλες τις θύρες-καταχωρητές του μικροελεγκτή AVR ATmega16 (σε παρένθεση είναι η τιμή της διεύθυνσης όταν θεωρείται ως θέση μνήμης SRAM):

Πίνακας Π1.1 Διευθύνσεις καταχωρητών I/O

Address	**Name**	**Address**	**Name**
$3F ($5F)	SREG	$1F ($3F)	EEARH-EEPROM Addr register high byte
$3E ($5E)	SPH	$1E ($3E)	EEARL-EEPROM Addr register low byte
$3D ($5D)	SPL	$1D ($3D)	EEDR - EEPROM Data Register
$3C ($5C)	OCR0 Timer/Counter0 Output Compare Register	$1C ($3C)	EECR - EEPROM Control Register
$3B ($5B)	GICR Gener. Interrupt Control	$1B ($3B)	PORTA
$3A ($5A)	GIFR General Interrupt Flags	$1A ($3A)	DDRA
$39 ($59)	TIMSK Timers Interrupt Mask	$19 ($39)	PINA
$38 ($58)	TIFR Timers Interrupt Flags	$18 ($38)	PORTB
$37 ($57)	SPMCR Store Prog Con. Reg.	$17 ($37)	DDRB
$36 ($56)	TWCR - TWI Control Register	$16 ($36)	PINB
$35 ($55)	MCUCR Processor General Control	$15 ($35)	PORTC

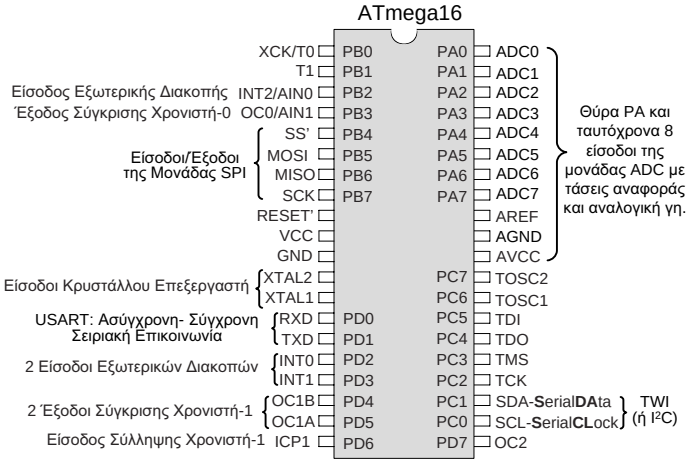
	Register		
$34 ($54)	MCUCSR (Pr Status Register)	$14 ($34)	DDRC
$33 ($53)	TCCR0 Timer0 Control Reg.	$13 ($33)	PINC
$32 ($52)	TCNT0Timer/Counter0	$12 ($32)	PORTD
$31 ($51)	OSCCAL Osc. Calibration Reg	$11 ($31)	DDRD
$30 ($50)	SFIOR	$10 ($30)	PIND
$2F ($4F)	TCCR1A Timer1 Control Register A	$0F ($2F)	SPDR - SPI Data Register
$2E ($4E)	TCCR1B Timer1 Control Register B	$0E ($2E)	SPSR - SPI Status Register
$2D ($4D)	TCNT1H Timer/Counter1 Counter Register High Byte	$0D ($2D)	SPCR - SPI Control Register
$2C ($4C)	TCNT1L Timer/Counter1 Counter Register Low Byte	$0C ($2C)	UDR USART I/O Data Register
$2B ($4B)	OCR1AH Timer/Counter1 Output Compare Register A	$0B ($2B)	UCSRA-USARTControl Status Register A
$2A ($4A)	OCR1AL Timer/Counter1 Output Compare Register A	$0A ($2A)	UCSRB-USARTControl Status Register B
$29 ($49)	OCR1BH Timer/Counter1 Output Compare Register B	$09 ($29)	UBRRL - USART Baud Rate Register Low Byte
$28 ($48)	OCR1BL Timer/Counter1 Output Compare Register B	$08 ($28)	ACSR - Analog Control Status Register
$27 ($47)	ICR1H Timer/Counter1 Input Capture Register High Byte1	$07 ($27)	ADMUX - ADC Multiplexer Selection Register
$26 ($46)	ICR1L Timer/Counter1 Input Capture Register Low Byte1	$06 ($26)	ADCSRA - ADC Control Status Register
$25 ($45)	TCCR2 Timer2 Control Reg.	$05 ($25)	ADCH - ADC Data Register High Byte
$24 ($44)	TCNT2 Timer/Counter2 (8 bit)	$04 ($24)	ADCL - ADC Data Register Low Byte
$23 ($43)	OCR2Timer/Counter2 Output Compare Register	$03 ($23)	TWDR Two-wire Serial Interface Data Register
$22 ($42)	ASSR Asynchr. Status Reg.	$02 ($22)	TWAR Two-wire Address Register
$21 ($41)	WDTCR- WatchDog Con. Reg	$01 ($21)	TWSR Two-wire Status Register
$20 ($40)	UBRRH/URSEL (USART Baud Rate Register High Byte)	$00 ($20)	TWBR Two-wire bit Rate Register

Περιγραφή ακροδεκτών

VCC Ψηφιακή τροφοδοσία τάσης. **GND** Γείωση.

Port A (PA7..PA0) Η Port A λειτουργεί είτε ως αναλογική είσοδος του μετατροπέα A/D, είτε ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8bit. (Οι ακροδέκτες της θύρας παρέχουν εσωτερικές pull-up αντιστάσεις. Οι buffers εξόδου της Port A έχουν συμμετρικά χαρακτηριστικά οδήγησης με δυνατότητα υψηλής βύθισης τάσης και άντλησης έντασης.
Όταν οι ακροδέκτες PA0..PA7 χρησιμοποιούνται ως είσοδοι και είναι εξωτερικά pulled low, τότε θα αντλήσουν ρεύμα αν οι εσωτερικές pull-up αντιστάσεις είναι ενεργοποιημένες. Οι ακροδέκτες της Port A είναι τριών καταστάσεων, όταν ενεργοποιείται κατάσταση επανατοποθέτησης, ακόμα και αν το ρολόι δεν τρέχει.)
Port B (PB7..PB0) Η Port B λειτουργεί ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8 bit (με κοινά χαρακτηριστικά με την Port A). Επίσης, η Port B εκτελεί κάποιες ιδιαίτερες λειτουργίες του μικροελεγκτή.
Port C (PC7..PC0) Η Port C λειτουργεί ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8-bit (με κοινά χαρακτηριστικά με την Port A). Αν η διεπαφή JTAG είναι ενεργοποιημένη, οι pull-up αντιστάσεις στους ακροδέκτες PC5(TDI), PC3(TMS) and PC2(TCK) θα ενεργοποιηθούν σε περίπτωση επανατοποθέτησης (reset). Επίσης, η Port C εκτελεί κάποιες ιδιαίτερες λειτουργίες του μικροελεγκτή.

Port D (PD7..PD0) Η Port D λειτουργεί ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8-bit (με κοινά χαρακτηριστικά με την Port A). Επίσης, η Port D εκτελεί κάποιες ιδιαίτερες λειτουργίες του μικροελεγκτή.



Σχήμα Π.1 Ακροδέκτες του μικροελεγκτή ATmega16

RESET Έμφάνιση χαμηλού επιπέδου στον ακροδέκτη αυτό για χρόνο μεγαλύτερο από τον ελάχιστο μήκος παλμού, θα παράγει reset, ακόμα και αν το ρολόι δε λειτουργεί. Ελάχιστο μήκος παλμού 2μs.
XTAL1 Είσοδος στον ταλαντωτή-ενισχυτή και στο κύκλωμα εσωτερικού ρολογιού.
XTAL2 Έξοδος από ταλαντωτή-ενισχυτή.
AVCC είναι ο ακροδέκτης τροφοδοσίας τάσης για την Port A και το μετατροπέα A/D. Πρέπει να συνδέεται εξωτερικώς με τον ακροδέκτη VCC, ακόμη και αν ο ADC δε χρησιμοποιείται. Αν ο ADC χρησιμοποιείται, πρέπει να συνδεθεί στον VCC μέσω ενός βαθυπερατού φίλτρου.
AREF είναι ο *analog reference* ακροδέκτης για το μετατροπέα A/D.

ΠΑΡΑΡΤΗΜΑ 2

Πίνακας χαρακτήρων ASCII

Ο πίνακας χαρακτήρων ASCII χρησιμεύει για τη μετατροπή χαρακτήρων ascii σε δεκαεξαδικούς ή δεκαδικούς αριθμούς και αντιστρόφως.

Πίνακας Π2.1 Χαρακτήρες ASCII

Dec	Hex	Name	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
00	00	**NUL**	32	20	**Space**	64	40	**@**	96	60	**`**
01	01	**SOH**	33	21	**!**	65	41	**A**	97	61	**a**
02	02	**STX**	34	22	**“**	66	42	**B**	98	62	**b**
03	03	**ETX**	35	23	**#**	67	43	**C**	99	63	**c**
04	04	**EOT**	36	24	**$**	68	44	**D**	100	64	**d**
05	05	**ENQ**	37	25	**%**	69	45	**E**	101	65	**e**
06	06	**ACK**	38	26	**&**	70	46	**F**	102	66	**f**
07	07	**BEL**	39	27	**‘**	71	47	**G**	103	67	**g**
08	08	**BS**	40	28	**(**	72	48	**H**	104	68	**h**
09	09	**HT**	41	29	**)**	73	49	**I**	105	69	**i**
10	A	**LF**	42	2A	*	74	4A	**J**	106	6A	**j**
11	B	**VT**	43	2B	+	75	4B	**K**	107	6B	**k**
12	C	**FF**	44	2C	**,**	76	4C	**L**	108	6C	**l**
13	D	**CR**	45	2D	**-**	77	4D	**M**	109	6D	**m**
14	E	**SO**	46	2E	**.**	78	4E	**N**	110	6E	**n**
15	F	**SI**	47	2F	/	79	4F	**O**	111	6F	**o**
16	10	**DLE**	48	30	**0**	80	50	**P**	112	70	**p**
17	11	**DC1**	49	31	**1**	81	51	**Q**	113	71	**q**
18	12	**DC2**	50	32	**2**	82	52	**R**	114	72	**r**
19	13	**DC3**	51	33	**3**	83	53	**S**	115	73	**s**
20	14	**DC4**	52	34	**4**	84	54	**T**	116	74	**t**
21	15	**NAK**	53	35	**5**	85	55	**U**	117	75	**u**
22	16	**SYN**	54	36	**6**	86	56	**V**	118	76	**v**
23	17	**ETB**	55	37	**7**	87	57	**W**	119	77	**w**
24	18	**CAN**	56	38	**8**	88	58	**X**	120	78	**x**
25	19	**EM**	57	39	**9**	89	59	**Y**	121	79	**y**
26	1A	**SUB**	58	3A	**:**	90	5A	**Z**	122	7A	**z**
27	1B	**ESC**	59	3B	**;**	91	5B	**[**	123	7B	**{**
28	1C	**FS**	60	3C	<	92	5C	\	124	7C	**|**
29	1D	**GS**	61	3D	=	93	5D	**]**	125	7D	**}**
30	1E	**RS**	62	3E	>	94	5E	^	126	7E	~
31	1F	**US**	63	3F	**?**	95	5F	**_**	127	7F	**del**

Πίνακας Π2.2 Χαρακτήρες ελέγχου

Hex	**ΣΗΜΑΣΙΑ**	**Hex**	**ΣΗΜΑΣΙΑ**
01	SOH - Start Of Header	11	DC1 - Device Control 1
02	STX - Start Of teXt	12	DC2 - Device Control 2
03	ETX - End Of teXt	13	DC3 - Device Control 3
04	EOT - End Of Transmission	14	DC4 - Device Control 4
05	ENQ - ENQuiry	15	NAK - Negative AcKnowledge
06	ACK - ACKnowledge	16	SYN - SYNchronous idle
07	BEL - BELl	17	ETB - End of Transmission Block
08	BS - BackSpace	18	CAN - CANcel
09	HT - Horizontal Tabulation	19	EM - End of Medium
A	LF - Line Feed	1A	SUB - SUBstitute
B	VT - Vertical Tabulation	1B	ESC - ESCape
C	FF - Form Feed	1C	FS - File Separator
D	CR - Carriage Return	1D	GS - MainForm.Group Separator
E	SO - Shift Out	1E	RS - Record Separator
F	SI - Shift In	1F	US - Unit Separator
10	DLE - Data Link Escape		

ΠΑΡΑΡΤΗΜΑ 3

Οι εντολές του Μικροελεγκτή AVR

Στον πίνακα παρουσιάζονται οι συμβάσεις που χρησιμοποιούνται στην περιγραφή των εντολών.

Συμβολισμοί περιγραφής εντολών:

Καταχωρητής κατάστασης (SREG)

SREG: Καταχωρητής κατάστασης, C: Σημαία κρατουμένου, Z: Σημαία μηδενισμού, N: Σημαία αρνητικού αποτελέσματος
V: Σημαία για υπερχείλιση σε συμπλήρωμα ως προς 2
S: N xorV, για έλεγχο προσημασμένων
H: Σημαία ενδιάμεσου κρατουμένου
T: Μεταφερόμενο bit, χρησιμοποιείται από τις εντολές BLD and BST
I: Καθολική Σημαία ενεργοποίησης/απενεργοποίησης Διακοπών

Καταχωρητές και Ορίσματα

Rd, Rr: Καταχωρητής Εργασίας Προορισμού και Προέλευσης
R: Result after instruction is executed
K: Σταθερό Δεδομένο, k: Σταθερή Διεύθυνση
b: Θέση Bit Καταχωρητή Εργασίας ή Καταχωρητή θυρών I/O (3-bit)
s: Θέση Bit στον Καταχωρητής κατάστασης (3-bit)
X, Y, Z: Καταχωρητές έμμεσης διευθυνσιοδότησης (X=R27:R26, Y=R29:R28 and Z=R31:R30)
A: Διεύθυνση θυρών I/O
q: Μετατόπιση σε έμμεση διευθυνσιοδότηση (6-bit)

Μνημονικό	Ορίσματα	Περιγραφή	Λειτουργία	Σημαίες	Κύκλοι
Εντολές μεταφοράς δεδομένων					
MOV	Rd, Rr	Move Between Registers	Rd← Rr	-	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd← Rr+1:Rr	-	1
LDI	Rd, K	Load Immediate	Rd ← K	-	1
LD	Rd, X	Load Indirect	Rd ← (X)	-	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X← X+ 1	-	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X← X – 1, Rd← (X)	-	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	-	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y← Y+ 1	-	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y← Y – 1, Rd← (Y)	-	2
LDD	Rd, Y+q	Load Indirect + Displacement	Rd ← (Y+ q)	-	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	-	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z← Z+ 1	-	2
LD	Rd, - Z	Load Indirect and Pre-Dec.	Z←Z – 1, Rd← (Z)	-	2
LDD	Rd, Z+q	Load Indirect + Displacement	Rd ← (Z+ q)	-	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	-	2
ST	X, Rr	Store Indirect	(X)←Rr	-	2
ST	X+, Rr	Store Indirect and Post-Inc.	X)←Rr, X← X+ 1	-	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X – 1, (X)←Rr	-	2
ST	Y, Rr	Store Indirect	(Y)←Rr	-	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y)←Rr, Y← Y+ 1	-	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y – 1, (Y)←Rr	-	2
STD	Y+q,Rr	Store Indirect + Displacement	(Y + q) ← Rr	-	2
ST	Z, Rr	Store Indirect	(Z)←Rr	-	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z)←Rr, Z← Z+ 1	-	2
ST	- Z, Rr	Store Indirect and Pre-Dec.	Z ← Z – 1, (Z)←Rr	-	2
STD	Z+q,Rr	Store Indirect + Displacement	(Z + q) ← Rr	-	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	-	2
LPM		Load Program Memory	R0 ← (Z)	-	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	-	3
LPM	Rd, Z+	Load Program Memory	Rd ← (Z), Z← Z+ 1	-	3
SPM		Store Program Memory	(Z) ← R1:R0	-	-
IN	Rd, P	In Port	Rd ← P	-	1
OUT	P, Rr	Out Port	P ← Rr	-	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	-	2
POP	Rd	Pop Register from Stack	Rd ← STACK	-	2

*** Μεταφορά μεταξύ Καταχωρητών Εργασίας ή και Καταχωρητή Εργασίας με θέση μνήμης (δεδομένων και προγράμματος).**

Μνημονικό	Ορίσματα	Περιγραφή	Λειτουργία	Σημαίες	Κύκλοι

Εντολές αριθμητικών και λογικών πράξεων

ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Regs	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 Regs	Rd ← Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Const from Reg	Rd ←Rd – K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract & C two Regs	Rd ← Rd – Rr – C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract & C Const from Reg	Rd ← Rd – K – C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Imm from Word	Rdh:Rdl ←Rdh:Rdl – K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd Λ Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Reg & Const	Rd ← Rd Λ K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Reg & Const	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 ← $FF ⊕ Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd← Rd ($FF – 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 Λ Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	Καμία	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd × Rr	Z,C	2

*** Εκτελούνται μόνο μεταξύ Καταχωρητών Εργασίας ή και Καταχωρητή Εργασίας με σταθερά. Το αποτέλεσμα πάντα πάει σε Καταχωρητή Εργασίας.**

Εντολές άλματος, κλήσης ρουτινών, παράκαμψης, σύγκρισης και διακλάδωσης

Μνημονικό	Ορίσματα	Περιγραφή	Λειτουργία	Σημαίες	Κύκλοι

Εντολές άλματος και κλήσης ρουτινών

RJMP	k	Relative Jump	PC ← PC + k + 1	-	2
IJMP		Indirect Jump to (Z)	PC ← Z	-	2
JMP	k	Direct Jump	PC ← k	-	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	-	3
ICALL		Indirect Call to (Z)	PC ← Z	-	3
CALL	k	Direct Subroutine Call	PC ← k	-	4
RET		Subroutine Return	PC ← STACK	-	4
RETI		Interrupt Return	PC ← STACK	I	4

Μνημονικό	Ορίσματα	Περιγραφή	Λειτουργία	Σημαίες	Κύκλοι

Εντολές παράκαμψης

SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) *	-	1/ 2/ 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) *	-	1/ 2/ 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) *	-	1/ 2/ 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) *	-	1/ 2/ 3

* Σε όλες τις περιπτώσεις που ισχύει η συνθήκη φορτώνεται η νεα τιμή του μετρητή Προγράμματος: PC ← PC + 2 or 3.
Ισχύουν για τους Γενικούς Καταχωρητές και για τους Καταχωρητές Θυρών I/O.

Εντολές σύγκρισης

CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	-	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 Reg with Immediate	Rd < K	Z,N,V,C,H	1

*** Μεταξύ Καταχωρητών Εργασίας ή και Καταχωρητή Εργασίας με σταθερά.**

Εντολές διακλάδωσης

BRBS	s,k	Branch if Status Flag Set	if (SREG(s) = 1) *	-	1/2
BRBC	s,k	Branch if Status Flag Cleared	if (SREG(s) = 0) *	-	1/2
BREQ	k	Branch if Equal	if (Z = 1) then *	-	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then *	-	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then *	-	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then *	-	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then *	-	1/2
BRLO	k	Branch if Lower	if (C = 1) then *	-	1/2
BRMI	k	Branch if Minus	if (N = 1) then *	-	1/2
BRPL	k	Branch if Plus	if (N = 0) then *	-	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N⊕V= 0) then *	-	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N⊕V= 1) then *	-	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then *	-	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then *	-	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then *	-	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then *	-	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then *	-	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then *	-	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then *	-	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then *	-	1/2

* Σε όλες τις περιπτώσεις που ισχύει η συνθήκη φορτώνεται η νεα τιμή του μετρητή Προγράμματος: PC ← PC + k +1

Εντολές ελέγχου MCU

NOP		No Operation		-	1
SLEEP		Sleep	Sleep function	-	1
WDR		Watchdog Reset	WDR/timer function	-	1
BREAK		Break	For On-chip Debug Only	-	N/A

Μνημονικό	Ορίσματα	Περιγραφή	Λειτουργία	Σημαίες	Κύκλοι

Εντολές σε επίπεδο bit και ελέγχου bit

SBI	P,b	Set Bit in I/O Reg	I/O(P,b) ← 1	-	2
CBI	P,b	Clear Bit in I/O Reg	I/O(P,b) ← 0	-	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 + Carry	Rd(0)←C, Rd(n+1)←Rd(n), C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right + 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)	-	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	-	1
SEx		Set Flag	x ← 1	x	1
CLx		Clear Flag	x ← 0	x	1

*Ισχύουν για Καταχωρητές Εργασίας και οι 2 πρώτες για τους Καταχωρητές Θυρών I/O.

Οι σημαίες που περιλαμβάνει ο καταχωρητής κατάστασης θέτονται ή μηδενίζονται μέσω αριθμητικών, λογικών, σύγκρισης, επιπέδου bit, ολίσθησης και άλλων εντολών, όπως φαίνεται στον επόμενο πίνακα.

Σημαία	Αριθμητικές	Λογικές	Σύγκρι-σης	Επιπέδου bit	Ολίσθησης κλπ.
Zero	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI OR, ORI EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR Z, BSET Z, CLZ, SEZ, TST	ASR, LSL, LSR, ROL, ROR, **CLR**
Carry	ADD, ADC, ADIW, SUB, SUBI, SBC, SBCI, SBIW	COM, NEG	CP, CPC, CPI	BCLR C, BSET C, CLC, SEC	ASR, LSL, LSR, ROL, ROR
Nega-tive	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI OR, ORI EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR N, BSET N, CLN, SEN, TST	ASR, LSL, LSR, ROL, ROR, **CLR**
oVer-flow	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI OR, ORI EOR, COM, NEG, SBR CBR	CP, CPC, CPI	BCLR V, BSET V, CLV, SEV, TST	ASR, SL, LSR, ROL, ROR, **CLR**
Sign	SBIW			BCLR S, BSET S, CLS, SES	
Half Carry	ADD, ADC, SUB, SUBI, SBC, SBCI	NEG	CP, CPC, CPI	BCLR H, BSET H, CLH, SEH	
Trans-fer				BCLR T, BSET T, BST, CLT, SET	
Inter-rupt				BCLR I, BSET I, CLI, SEI	**RETI**

ΠΕΡΙΓΡΑΦΗ ΕΝΤΟΛΩΝ

Στις επόμενες σελίδες υπάρχει λεπτομερής περιγραφή των εντολών του μικροελεγκτή AVR. Κάθε εντολή περιγράφεται με τον εξής τρόπο:

1. Η περιγραφή της εντολής αποτελείται από το **μνημονικό** της, τυπωμένο σε BOLDFACE και από τα πεδία ορισμάτων.

2. Ακολουθεί μια συμβολική περιγραφή της **λειτουργίας** της εντολής.

3. Στη συνέχεια έχουμε μια **περιγραφή** της λειτουργίας της εντολής.

4. Στην ένδειξη **Ενημέρωση σημαιών** δηλώνονται οι **σημαίες** του καταχωρητή κατάστασης που επηρεάζονται από την εκτέλεση της εντολής. Όταν δεν υπάρχει σημαίνει ότι δεν επηρεάζεται καμία σημαία.

5. Δίνεται ένα **παράδειγμα** για να γίνει κατανοητή η σύνταξη της εντολής. Δε δίνεται παράδειγμα αν η εφαρμογή της είναι μονοσήμαντη και δε διαθέτει ορίσματα.

6. Η τελευταία γραμμή περιέχει τον αριθμό των **περιόδων ρολογιού** (cycles) που απαιτούνται για την επεξεργασία της εντολής. Δεδομένου ότι οι περισσότερες εντολές απαιτούν 1 κύκλο ρολογιού, σε αυτές παραλήπεται και αναφέρεται όταν έχουμε περισσότερους από 1 κύκλο Αν η εντολή έχει δύο δυνατούς χρόνους εκτέλεσης, όπως σε μια εντολή άλματος υπό συνθήκη, τότε οι δύο χρόνοι διαχωρίζονται από "/".

ΕΝΤΟΛΕΣ ΑΡΙΘΜΗΤΙΚΩΝ ΚΑΙ ΛΟΓΙΚΩΝ ΠΡΑΞΕΩΝ

ADD Rd, Rr

Λειτουργία: Rd ← Rd + Rr
Περιγραφή: Πρόσθεση 2 καταχωρητών χωρίς χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd.
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: ADD R4,R5

ADC Rd, Rr

Λειτουργία: Rd ← Rd + Rr+C
Περιγραφή: Πρόσθεση 2 καταχωρητών με χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd.
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: ADD R5,R6

ADIW Rd, K

Λειτουργία: Rd+1:Rd ←Rd+1:Rd + K
Περιγραφή: Πρόσθεση μιας τιμής K (0-63) σε ένα ζεύγος καταχωρητών και αποθήκευση στο ζεύγος καταχωρητών
Ενημέρωση σημαιών: Z, C, N, S, V
Παράδειγμα: ADD R24:R23,5
Περίοδοι ρολογιού: 2

SUB Rd, Rr

Λειτουργία: Rd ← Rd - Rr
Περιγραφή: Αφαίρεση 2 καταχωρητών χωρίς χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: SUB R4,R5

SUBI Rd, K

Λειτουργία: Rd ←Rd - K
Περιγραφή: Αφαίρεση μιας τιμής χωρίς χρήση κρατουμένου από έναν καταχωρητή και αποθήκευση στον καταχωρητή
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: SUBI R4,15

SUBCI Rd, K

Λειτουργία: Rd ←Rd – K-C
Περιγραφή: Αφαίρεση μιας τιμής με χρήση κρατουμένου από έναν καταχωρητή και αποθήκευση στον καταχωρητή
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: SUB R9,2

SBC Rd, Rr

Λειτουργία: Rd ← Rd – Rr-C
Περιγραφή: Αφαίρεση 2 καταχωρητών με χρήση κρατουμένου και αποθήκευση στον καταχωρητή προορισμού Rd
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: SBC R4,R5

SBIW Rd, K

Λειτουργία: Rd+1:Rd ←Rd+1:Rd - K
Περιγραφή: Αφαίρεση μιας τιμής από ένα ζεύγος καταχωρητών και αποθήκευση στο ζεύγος καταχωρητών
Ενημέρωση σημαιών: Z, C, N, S, V
Παράδειγμα: SBIW R26,8
Περίοδοι ρολογιού: 2

AND Rd, Rr

Λειτουργία: Λογικό AND Rd ←Rd Λ Rr
Περιγραφή: Λογικό AND μεταξύ των περιεχομένων των καταχωρητών Rd, Rr
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: AND R26,R8

ANDI Rd, K

Λειτουργία: Λογικό AND Rd ←Rd Λ K
Περιγραφή: Λογικό AND μεταξύ των περιεχομένων του καταχωρητή Rd και της σταθεράς K .Ο καταχωρητής Rd είναι ένας από τους R16-R31. Σταθερά K είναι 8bits
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: AND R26,8

OR Rd, Rr

Λειτουργία: Λογικό OR Rd ←Rd ∨ Rr
Περιγραφή: Λογικό OR μεταξύ των περιεχομένων των καταχωρητών Rd, Rr
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: OR R2,R3

ORI Rd, K

Λειτουργία: Λογικό OR Rd ←Rd ∨ K
Περιγραφή: Λογικό OR μεταξύ των περιεχομένων του καταχωρητή Rd και της σταθεράς K . Ο καταχωρητής Rd είναι ένας από τους R16-R31. Η σταθερά K είναι 8 bit
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: ORI R16,0b10011101

EOR Rd, Rr

Λειτουργία: Λογικό XOR Rd ←Rd □ Rr
Περιγραφή: Λογικό XOR (αποκλειστικό ‘ή’) μεταξύ των περιεχομένων των καταχωρητών Rd, Rr.
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: EOR R2,R3

COM Rd

Λειτουργία: Rd ←$FF - Rd
Περιγραφή: Αντιστροφή των περιεχομένων του καταχωρητή Rd με χρήση αριθμητικής συμπληρώματος 1.
Ενημέρωση σημαιών: Z, C, N, S,V
Παράδειγμα: COM R1

NEG Rd

Λειτουργία: Rd ←$00 - Rd
Περιγραφή: Αντιστροφή των περιεχομένων του καταχωρητή Rd με χρήση αριθμητικής συμπληρώματος 2.
Ενημέρωση σημαιών: Z,C, N, S,V, H
Παράδειγμα: NEG R10

SBR Rd, K

Λειτουργία: Rd ←Rd ∨ K
Περιγραφή: Τοποθέτηση λογικού ‘1’ στα bits του καταχωρητή Rd εκτελώντας λογικό OR μεταξύ των περιεχομένων του καταχωρητή Rd και της σταθεράς K. Ο καταχωρητής Rd είναι ένας από τους R16-R31.
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: SBR R18,4

CBR Rd, K

Λειτουργία: Rd←Rd Λ ($FFh -K)
Περιγραφή: Τοποθέτηση λογικού ‘0’ στα bits του καταχωρητή Rd εκτελώντας λογικό AND μεταξύ των περιεχομένων του καταχωρητή Rd και της σταθεράς K. Ο καταχωρητής Rd είναι ένας από τους R16-R31.
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: CBR R16,5

INC Rd

Λειτουργία: Rd ←Rd + 1
Περιγραφή: Αύξηση των περιεχομένων του καταχωρητή Rd κατά μία μονάδα.
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: INC R1

DEC Rd

Λειτουργία: Rd ←Rd - 1
Περιγραφή: Μείωση των περιεχομένων του καταχωρητή Rd κατά μία μονάδα.
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: DEC R5

TST Rd

Λειτουργία: Rd ←Rd Λ Rd
Περιγραφή: Έλεγχος μηδενικής ή αρνητικής τιμής των περιεχομένων του καταχωρητή Rd.
Ενημέρωση σημαιών: Z, N,S,V
Παράδειγμα: TST R3

CLR Rd

Λειτουργία: Rd ←$00
Περιγραφή: Μηδενισμός καταχωρητή Rd.
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: CLR R3

SER Rd

Λειτουργία: Rd ←$FF
Περιγραφή: Τοποθέτηση '1' σε όλα τα bits του καταχωρητή (τιμή $FF).
Ενημέρωση σημαιών: Z, N, S,V
Παράδειγμα: SER R3

MUL Rd, Rr

Λειτουργία: R1, R0 ←Rd×Rr
Περιγραφή: Μη προσημασμένος πολλαπλασιασμός μεταξύ των περιεχομένων των καταχωρητών Rd, Rr. Το αποτέλεσμα αποθηκεύεται στο ζεύγος καταχωρητών R1, R0.
Ενημέρωση σημαιών: Z, C
Παράδειγμα: MUL R3,R2
Περίοδοι ρολογιού: 2

MULS Rd, Rr

Λειτουργία: R1, R0 ←Rd×Rr
Περιγραφή: Προσημασμένος πολλαπλασιασμός Rd, Rr. Το αποτέλεσμα στο ζεύγος καταχωρητών R1, R0.
Ενημέρωση σημαιών: Z, C
Παράδειγμα: MULS R23,R21
Περίοδοι ρολογιού: 2

MULSU Rd, Rr

Λειτουργία: R1, R0 ←Rd×Rr
Περιγραφή: Προσημασμένος πολλαπλασιασμός μεταξύ των καταχωρητών Rd (προσημασμένου), Rr(μη-προσημασμένου). Το αποτέλεσμα στο ζεύγος καταχωρητών R1, R0.
Ενημέρωση σημαιών: Z, C
Παράδειγμα: MULSU R21, R20
Περίοδοι ρολογιού: 2

FMUL Rd, Rr

Λειτουργία: R1, R0 ←Rd×Rr
Περιγραφή: Μη προσημασμένος κλασματικός πολλαπλασιασμός μεταξύ των περιεχομένων των καταχωρητών Rd (μορφοποίηση 1.7) και Rr (μορφοποίηση 1.7). Το αποτέλεσμα αποθηκεύεται στο ζεύγος καταχωρητών R1, R0 (μορφοποίηση 1.15).
Ενημέρωση σημαιών: Z, C
Παράδειγμα: FMUL R22, R20
Περίοδοι ρολογιού: 2

FMULS Rd, Rr

Λειτουργία: R1, R0 ←Rd×Rr
Περιγραφή: Προσημασμένος κλασματικός πολλαπλασιασμός μεταξύ των περιεχομένων των καταχωρητών Rd (μορφοποίηση 1.7) και Rr (μορφοποίηση 1.7). Το αποτέλεσμα αποθηκεύεται στο ζεύγος καταχωρητών R1, R0 (μορφοποίηση 1.15).
Ενημέρωση σημαιών: Z, C
Παράδειγμα: FMULS R23, R22
Περίοδοι ρολογιού: 2

FMULSU Rd, Rr

Λειτουργία: R1, R0 ←Rd×Rr
Περιγραφή: Προσημασμένος κλασματικός πολλαπλασιασμός μεταξύ των περιεχομένων των καταχωρητών Rd (προσημασμένος με μορφοποίηση 1.7) και Rr (μη-προσημασμένος με μορφοποίηση 1.7). Το αποτέλεσμα αποθηκεύεται στο ζεύγος καταχωρητών R1, R0 (μορφοποίηση 1.15).
Ενημέρωση σημαιών: Z, C
Παράδειγμα: FMULSU R23, R22
Περίοδοι ρολογιού: 2

ΕΝΤΟΛΕΣ ΕΛΕΓΧΟΥ ΡΟΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ ΚΑΙ ΔΙΑΚΛΑΔΩΣΗΣ

RJMP k

Λειτουργία: PC←PC + k + 1
Περιγραφή: Σχετικό άλμα σε μια διεύθυνση της μνήμης προγράμματος μεταξύ PC - 2K +1 και PC + 2K .
Παράδειγμα: rjmp ok
error: add r16, r17
inc r16
ok: nop
Περίοδοι ρολογιού: 2

IJMP

Λειτουργία: PC ←Z(15:0)
Περιγραφή: Έμμεση εκτέλεση άλματος σε θέση της μνήμης προγράμματος η διεύθυνση της οποίας αντιστοιχεί στο περιεχόμενο του καταχωρητή δείκτη Z .
Παράδειγμα: ldi r30, low
Ldi r31, high
ijmp
Περίοδοι ρολογιού: 2

JMP k

Λειτουργία: PC ←k
Περιγραφή: Απλό άλμα σε μια θέση της μνήμης προγράμματος.

Παράδειγμα:
jmp farplc ; άλμα χωρίς συνθήκη
;
farplc: nop
Περίοδοι ρολογιού: 3

RCALL k

Λειτουργία: PC ←PC + k + 1
Περιγραφή: Σχετική κλήση ρουτίνας σε μια διεύθυνση μεταξύ PC - 2K +1 και PC + 2K. Η διεύθυνση επιστροφής αποθηκεύεται στη στοίβα.
Παράδειγμα:
rcall routine ; κλήση ρουτίνας

routine: push r14 ; r14 στη στοίβα
; … εντολές τις ρουτίνας …
pop r14 ; ανάκτηση r14 από τη στοίβα
ret ; επιστροφή από υπορουτίνα
Περίοδοι ρολογιού:3/4

ICALL k

Λειτουργία: PC ←Z(15:0)
Περιγραφή: Έμμεση κλήση ρουτίνας η διεύθυνση της οποίας αντιστοιχεί στο περιεχόμενο του καταχωρητή δείκτη Z .
Παράδειγμα: ldi r30, low
ldi r31, high
icall
Περίοδοι ρολογιού:3/4

CALL k

Λειτουργία: PC ←k
Περιγραφή: Κλήση υπορουτίνας εντός της μνήμης προγράμματος . Η διεύθυνση επιστροφής αποθηκεύεται στη στοίβα.
Παράδειγμα: call delay
Περίοδοι ρολογιού: 4/5

RET

Λειτουργία: PC ←Stack
Περιγραφή: Επιστροφή από υπορουτίνα. Η διεύθυνση επιστροφής φορτώνεται από τη στοίβα.
Περίοδοι ρολογιού: 4/5

RETI

Λειτουργία: PC ←Stack
Περιγραφή: Επιστροφή από ρουτίνα εξυπηρέτησης διακοπής. Η διεύθυνση επιστροφής φορτώνεται από τη στοίβα και η σημαία ολικών διακοπών τίθεται.
Ενημέρωση σημαιών: I
Περίοδοι ρολογιού: 4/5

CPSE Rd, Rr

Λειτουργία: Αν Rd = Rr τότε PC←PC + 2 (ή 3) αλλιώς PC←PC + 1
Περιγραφή: Σύγκριση μεταξύ των καταχωρητών Rd,Rr. Σε ισότητα παρακάμπτεται η επόμενη εντολή.
Παράδειγμα: cpse r4,r0
neg r4 ; εκτέλεση όταν r4≠r0
Περίοδοι ρολογιού: 1/2/3

CP Rd, Rr

Λειτουργία: Rd - Rr
Περιγραφή: Σύγκριση με υπολογισμό της διαφοράς μεταξύ των καταχωρητών Rd, Rr.
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: cp r4, r19 ; σε περίπτωση
brne noteq ; ανισότητας διακλάδωση

CPC Rd, Rr

Λειτουργία: Rd – Rr-C
Περιγραφή: Σύγκριση με υπολογισμό της διαφοράς και χρήση κρατουμένου μεταξύ των καταχωρητών Rd,Rr.
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: cp r2, r0
cpc r3,r1 ; σε περίπτωση ισότητας
breq equal ; διακλάδωση

CPI Rd, K

Λειτουργία: Rd - K
Περιγραφή: Σύγκριση με υπολογισμό της διαφοράς και χρήση κρατουμένου μεταξύ του καταχωρητή Rd και μιας σταθεράς. Ο καταχωρητής Rd είναι ένας από τους R16-R31
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: cpi r19, 3
brne error ; διακλάδωση σε περίπτωση
; ανισότητας

SBRC Rr, b

Λειτουργία: Αν Rr(b) = 0 τότε PC ←PC + 2 (ή 3) αλλιώς PC ←PC + 1
Περιγραφή: Αν το bit (b) του καταχωρητή είναι 0 (Rr(b) = 0) τότε παρακάμπτεται η επόμενη εντολή.
Παράδειγμα: loop: in r4, PINB
sbrc r4, 1
rjmp loop
ser r4
Περίοδοι ρολογιού: 1/2/3

SBRS Rr, b

Λειτουργία: Αν Rr(b) = 1 τότε PC ←PC + 2 (ή 3) αλλιώς PC ←PC + 1
Περιγραφή: Αν το bit (b) του καταχωρητή είναι 1 (Rr(b) = 1) τότε παρακάμπτεται η επόμενη εντολή.
Παράδειγμα: loop: in r4, PINB
sbrs r4, 2
rjmp loop
clr r4
Περίοδοι ρολογιού: 1/2/3

SBIC A, b

Λειτουργία: Αν I/O(A,b) = 0 τότε PC←PC + 2 (or 3) αλλιώς PC ←PC + 1
Περιγραφή: Αν το bit (b) του καταχωρητή εισόδου-εξόδου είναι 0 (A(b) = 0) τότε παρακάμπτεται η επόμενη εντολή.
Παράδειγμα:
e2wait: sbic $1C,1 ; Αν το EEWE είναι 0
; παράκαμψε την επόμενη εντολή
rjmp e2wait ; η εγγραφή της EEPROM
nop ; δεν ολοκληρώθηκε
Περίοδοι ρολογιού: 1/2/3

SBIS A, b

Λειτουργία: Αν I/O(A,b) = 1 τότε PC←PC + 2 (ή 3) αλλιώς PC ←PC + 1
Περιγραφή: Αν το bit (b) του καταχωρητή εισόδου-εξόδου είναι 1 (A(b) = 1) τότε παρακάμπτεται η επόμενη εντολή.
Παράδειγμα: wait: sbis PINB,0
rjmp wait
Περίοδοι ρολογιού: 1/2/3

BRBS s, k

Λειτουργία: Αν SREG(s) = 1 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Αν το bit s του καταχωρητή SREG(s) = 1 τότε εκτελείται άλμα στη διεύθυνση της μνήμης προγράμματος που βρίσκεται k θέσεις μετά σε σχέση με το μετρητή προγράμματος.
Παράδειγμα:
bst r0,3 ;Φόρτωση του bit 3 του r0 στο T
brbs 6,bitset ; διακλάδωση αν bit T =1
...
bitset: nop ; προορισμός διακλάδωσης
Περίοδοι ρολογιού: 1/2

BRBC s, k

Λειτουργία: Αν SREG(s) = 0 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Αν το bit s του καταχωρητή κατάστασης είναι 0 (SREG(s) = 0) τότε εκτελείται άλμα στη διεύθυνση της μνήμης προγράμματος που βρίσκεται k θέσεις μετά σε σχέση με το μετρητή προγράμματος.
Παράδειγμα: bst r0, 4 ;Φόρτωση του bit 4
brbs 6, bit0 ; του r0 στο T, διακλάδωση
... ; αν το bit T=0
bit0: nop ; προορισμός διακλάδωσης
Περίοδοι ρολογιού: 1/2

BREQ k

Λειτουργία: Αν Rd = Rr (Z = 1) τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα σε περίπτωση ισότητας. Όταν η σημαία μηδενός ισούται με 1 (Z = 1) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 1, k. **Παράδειγμα:** cp r1, r0
breq equal ; άλμα αν Z=1
…
equal: nop ; προορισμός διακλάδωσης
Περίοδοι ρολογιού: 1/2

BRNE k

Λειτουργία: Αν Rd <> Rr (Z = 0) τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα σε περίπτωση ανισότητας. Όταν η σημαία μηδενός ισούται με 0 (Z = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 1, k.
Παράδειγμα: cp r1, 5
brne notequal ; άλμα αν Z=0
…
notequal: nop ; προορισμός διακλάδωσης
Περίοδοι ρολογιού: 1/2

BRCS k

Λειτουργία: Αν C = 1 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία κρατουμένου ισούται με 1 (C = 1) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 0, k.
Παράδειγμα: cpi r26, $56
brcs carry ; έλεγχος και άλμα αν C=1
..
carry: nop ; προορισμός διακλάδωσης
Περίοδοι ρολογιού: ½

BRCC k

Λειτουργία: Αν C = 0 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία κρατουμένου ισούται με 0 (C = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 0,k.
Παράδειγμα: add r22, r23
brcc nocarry ; έλεγχος και άλμα αν C=0
...
nocarry: nop
Περίοδοι ρολογιού: 1/2

BRSH k

Λειτουργία: Αν Rd >=Rr (C = 0) τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση δώσει μεγαλύτερο ή ίσο (C = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 0, k.
Παράδειγμα: subi r19, 4 ; r19 - 4
brsh higher ; άλμα αν C=0
...
higher: nop
Περίοδοι ρολογιού: 1/2

BRLO k

Λειτουργία: Αν Rd<Rr (C = 1) τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση δώσει μικρότερο (C = 1) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 0,k.
Παράδειγμα: loop: inc r19
cpi r19, $10 ;
brlo loop ; όταν r19 τερμτίζεται ο βρόχος
Περίοδοι ρολογιού: 1/2

BRMI k

Λειτουργία: Αν N = 1 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση δώσει (N = 1) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 2, k.
Παράδειγμα: subi r18, 4
brmi negative
...
negative: nop
Περίοδοι ρολογιού: 1/2

BRPL k

Λειτουργία: Αν N = 0 τότε PC ←PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση δώσει (N = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 2, k.
Παράδειγμα: subi r26, $50
brpl positive
...
positive: nop
Περίοδοι ρολογιού: 1/2

BRGE k

Λειτουργία: Αν Rd>=Rr (S=N⊕V= 0) τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση προσημασμένου δώσει μεγαλύτερο ή ίσο (S = 0) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 4,k.
Παράδειγμα: cp r21, r19 ;
brge greateq
...
greateq:
Περίοδοι ρολογιού: 1/2

BRLT k

Λειτουργία: Αν Rd<Rr (S=N⊕V=1) τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση προσημασμένου δώσει μικρότερο (S = 1) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 4,k.
Παράδειγμα: cp r21, r19 ;
brlt less ;
...
less: nop
Περίοδοι ρολογιού: 1/2

BRHS k

Λειτουργία: Αν H = 1 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία δεκαδικού κρατουμένου ισούται με 1 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 5, k.
Παράδειγμα: brhs hset
Περίοδοι ρολογιού: 1/2

BRHC k

Λειτουργία: Αν H = 0 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία δεκαδικού κρατουμένου ισούται με 0 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 5,k.
Παράδειγμα: brhc hclr
Περίοδοι ρολογιού: ½

BRTS k

Λειτουργία: Αν T = 1 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία T ισούται με 1 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 6, k.
Παράδειγμα: bst r3, 5
brts tset
...
tset: nop
Περίοδοι ρολογιού: 1/2

BRTC k

Λειτουργία: Αν T = 0 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία T ισούται με 0 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 6,k.
Παράδειγμα: bst r3, 5
brtc tclear
...
tclear:
Περίοδοι ρολογιού: 1/2

BRVS k

Λειτουργία: Αν V = 1 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία υπερχείλισης V ισούται με 1 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 3, k.
Παράδειγμα: add r3, r4
brvs overfl ; άλμα σε υπερχείλιση
...
overfl: nop
Περίοδοι ρολογιού: 1/2

BRVC k

Λειτουργία: Αν V = 0 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία υπερχείλισης V ισούται με 0 τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 3, k.
Παράδειγμα: add r3, r4
brvc nooverfl ; V=1 σχετική διακλάδωση
...
nooverfl: nop

Περίοδοι ρολογιού: 1/2

BRIE k

Λειτουργία: Αν I=1 τότε PC ←PC + k + 1, αλλιώς PC←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία ολικών διακοπών ισούται με 1(I=1) τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBS 7,k.
Παράδειγμα:

```
brie inten      ; άλμα όταν οι διακοπές είναι
                ; ενεργοποιημένες
...
inten: nop
```

Περίοδοι ρολογιού: 1/2

BRID k

Λειτουργία: Αν I=0 τότε PC← PC + k + 1, αλλιώς PC ←PC + 1
Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σημαία ολικών διακοπών ισούται με 0(I=0), δηλαδή οι διακοπές είναι απενεργοποιημένες, τότε εκτελείται σχετική διακλάδωση. Ισοδυναμεί στην εντολή BRBC 7, k.
Παράδειγμα:

```
brid intdis  ; σχετική διακλάδωση όταν οι
       ; διακοπές είναι απενεργοποιημένες
...
intdis: nop
```

Περίοδοι ρολογιού: 1/2

ΕΝΤΟΛΕΣ ΜΕΤΑΦΟΡΑΣ ΔΕΔΟΜΕΝΩΝ
(δεν ενημερώνεται καμία σημαία)

MOV Rd, Rr

Λειτουργία: Rd← Rr
Περιγραφή:Το περιεχόμενο του καταχωρητή Rr αντιγράφεται στον Rd. Ο Rr παραμένει αναλλοίωτος.
Παράδειγμα: mov r7, r4

MOVW Rd, Rr

Λειτουργία: Rd+1:Rd← Rr+1:Rr
Περιγραφή: Το περιεχόμενο του ζεύγους καταχωρητών Rr+1:Rr αντιγράφεται στο ζεύγος καταχωρητών Rd+1:Rd.
Παράδειγμα: mov r26,r28

LDI Rd, K

Λειτουργία: Rd ←K
Περιγραφή: Άμεση φόρτωση μιας σταθεράς 8-bit στον καταχωρητή Rd. Ο καταχωρητής Rd είναι ένας από τους R16-R31.
Παράδειγμα: ldi r20, $8

LDS Rd, k

Λειτουργία: Rd ←(k)
Περιγραφή: Τα περιεχόμενα της θέσης μνήμης (k) φορτώνονται στον καταχωρητή Rd.
Παράδειγμα: lds r20,$FA00
Περίοδοι ρολογιού: 2

LD Rd, X (επίσης **LD** Rd, Y ή Z)

Λειτουργία: Rd ←(X)
Περιγραφή: Έμμεση φόρτωση του καταχωρητή Rd με τα περιεχόμενα της θέσης μνήμης (X). Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27 και Y=R28:R29,Z= R30:R31.
Παράδειγμα: ld r2, X
Περίοδοι ρολογιού: 2

LD Rd, X+ (επίσης **LD** Rd, Y+ ή Z+)

Λειτουργία: Rd ←(X) ,X ←X +1
Περιγραφή: Έμμεση φόρτωση του καταχωρητή Rd με τα περιεχόμενα της θέσης μνήμης (X). Αύξηση του καταχωρητής X κατά μία μονάδα. Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27 και Y=R28:R29,Z= R30:R31.
Παράδειγμα: ld r2,X+
Περίοδοι ρολογιού: 2

LD Rd, -X (επίσης **LD** Rd, -Y ή -Z)

Λειτουργία: X ←X - 1, Rd ←(X)
Περιγραφή: Μείωση του καταχωρητής X κατά μία μονάδα. Έμμεση φόρτωση του καταχωρητή Rd με τα περιεχόμενα της θέσης μνήμης (X). Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27 και Y=R28:R29,Z= R30:R31.
Παράδειγμα: ld r6,-X
Περίοδοι ρολογιού: 2

LDD Rd, Y+q (επίσης **LDD** Rd, Z+q)

Λειτουργία: Rd ←(Y+q)
Περιγραφή: Έμμεση φόρτωση του καταχωρητή Rd με τα περιεχόμενα της θέσης μνήμης (Y+q). Ο καταχωρητής Y είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R28:R29 και Z= R30:R31.
Παράδειγμα: ld r6,Y+4
Περίοδοι ρολογιού: 2

STS k,Rr

Λειτουργία: (k) ←Rr
Περιγραφή: Άμεση αποθήκευση καταχωρητή. Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (k).
Παράδειγμα: sts $4A00, r20
Περίοδοι ρολογιού: 2

ST X, Rr

Λειτουργία: (X) ←Rr
Περιγραφή: Έμμεση αποθήκευση καταχωρητή. Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (X). Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27.
Παράδειγμα: st X, r3
Περίοδοι ρολογιού: 2

ST X+, Rr (επίσης **ST** Rd, Y+ ή Z+)

Λειτουργία: (X) ←Rr , X ←X+1
Περιγραφή: Έμμεση αποθήκευση καταχωρητή. Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (X). Αύξηση του καταχωρητή X κατά μία μονάδα. Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27.
Παράδειγμα: st X+, r2
Περίοδοι ρολογιού: 2

ST -X, Rr (επίσης **ST** Rd, -Y ή -Z)

Λειτουργία: X ←X-1 , (X) ←Rr
Περιγραφή: Έμμεση αποθήκευση καταχωρητή. Μείωση του καταχωρητής X κατά μία μονάδα .Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (X). Ο καταχωρητής X είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R26:R27.
Παράδειγμα: st -X, r12
Περίοδοι ρολογιού: 2

STD Y+q, Rr (επίσης **STD** Rd, Z+q)

Λειτουργία: (Y+q) ←Rr
Περιγραφή: Έμμεση αποθήκευση καταχωρητή με χρήση μετατόπισης. Το περιεχόμενο του καταχωρητή Rr αποθηκεύεται στη θέση μνήμης (Y+q). Ο καταχωρητής Y είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R28:R29 και Z= R30:R31.
Παράδειγμα: std Y+4,r2
Περίοδοι ρολογιού: 2

LPM

Λειτουργία: R0 ←(Z)
Περιγραφή: Φόρτωση των περιεχομένων μιας θέσης μνήμης. Το περιεχόμενο της θέσης μνήμης προγράμματος (Z) φορτώνεται στον καταχωρητή R0. Ο καταχωρητής Z είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R30:R31.
Περίοδοι ρολογιού: 3

LPM Rd, Z

Λειτουργία: Rd ←(Z)
Περιγραφή: Φόρτωση των περιεχομένων μιας θέσης μνήμης. Το περιεχόμενο της θέσης μνήμης προγράμματος (Z) φορτώνεται στον καταχωρητή Rd. Ο καταχωρητής Z είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R30:R31.
Παράδειγμα: lpm r4, Z
Περίοδοι ρολογιού: 3

LPM Rd, Z+

Λειτουργία: Rd ←(Z), Z ←Z + 1
Περιγραφή: Το περιεχόμενο της θέσης μνήμης προγράμματος (Z) φορτώνεται στον καταχωρητή Rd. Αύξηση του καταχωρητή Z κατά μία μονάδα. Ο καταχωρητής Z είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R30:R31.
Παράδειγμα: lpm r24, Z+
Περίοδοι ρολογιού: 3

SPM

Λειτουργία: (Z) ←R1:R0
Περιγραφή: (Store Program Memory) Έμμεση αποθήκευση ζεύγους καταχωρητών. Το περιεχόμενο των καταχωρητών R1:R0 αποθηκεύεται στη θέση μνήμης προγράμματος (Z). Ο καταχωρητής Z είναι ένας καταχωρητής δείκτη που αντιστοιχεί στο ζεύγος καταχωρητών εργασίας R30:R31.
Περίοδοι ρολογιού: -

IN Rd, A

Λειτουργία: Rd ← I/O(A)
Περιγραφή: Εισαγωγή δεδομένων από καταχωρητή εισόδου-εξόδου (θύρες, χρονιστές κλπ) σε καταχωρητή Rd.
Παράδειγμα: in r3, PIND

OUT A, Rr

Λειτουργία: I/O(A) ←Rr
Περιγραφή: Εξαγωγή δεδομένων από καταχωρητή Rr σε καταχωρητή εισόδου-εξόδου (θύρες, χρονιστές, καταχ/τες κλπ).
Παράδειγμα: out PORTC, r6

PUSH Rr

Λειτουργία: STACK ←Rr
Περιγραφή: Αποθήκευση περιεχομένων καταχωρητή Rr στη στοίβα. Μείωση του δείκτη στοίβας κατά μία μονάδα.
Παράδειγμα: push r3
Περίοδοι ρολογιού: 2

POP Rd

Λειτουργία: Rd ←STACK
Περιγραφή: Αύξηση του δείκτη στοίβας κατά μία μονάδα. Αποθήκευση περιεχομένων της στοίβας στον καταχωρητή Rd.
Παράδειγμα: pop r3
Περίοδοι ρολογιού: 2

ΕΝΤΟΛΕΣ ΕΛΕΓΧΟΥ ΚΑΙ ΕΝΤΟΛΕΣ ΣΕ ΕΠΙΠΕΔΟ BIT

LSL Rd

Λειτουργία: Rd(n+1) ←Rd(n), Rd(0) ← 0, C ←Rd(7)
Περιγραφή: Ολίσθηση των bits του καταχωρητή Rd μία θέση προς τα αριστερά. Το bit0 μηδενίζεται και το bit7 φορτώνεται στη σημαία C του SREG.
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: lsl r1

LSR Rd

Λειτουργία: Rd(n) ←Rd(n+1), Rd(7) ← 0 , C ←Rd(0)
Περιγραφή: Ολίσθηση των bits του καταχωρητή Rd μία θέση προς τα δεξιά. Το bit7 μηδενίζεται και το bit0 φορτώνεται στη σημαία C του SREG.
Ενημέρωση σημαιών: Z, C, N, S, V
Παράδειγμα: lsr r21

ROL Rd

Λειτουργία: Rd(0) ← C , Rd(n+1) ←Rd(n) , C ←Rd(7)
Περιγραφή: Περιστροφή των bits του καταχωρητή Rd μία θέση προς τα αριστερά μέσω της σημαίας κρατουμένου. Η σημαία C ολισθαίνει στο bit0 και το bit7 ολισθαίνει στη σημαία C.
Ενημέρωση σημαιών: Z, C, N, S, V, H
Παράδειγμα: rol r2

ROR Rd

Λειτουργία: Rd(7) ← C , Rd(n) ←Rd(n+1) , C ←Rd(0)
Περιγραφή: Περιστροφή των bits του καταχωρητή Rd μία θέση προς τα δεξιά μέσω της σημαίας κρατουμένου. Η σημαία C ολισθαίνει στο bit7 και το bit0 ολισθαίνει στη σημαία C.
Ενημέρωση σημαιών: Z, C, N, S, V
Παράδειγμα: ror r2

ASR Rd

Λειτουργία: Rd(n) ←Rd(n+1) n=0..6, C←Rd(0)
Περιγραφή: Ολίσθηση των bits του καταχωρητή Rd μία θέση προς τα δεξιά. Το bit7 διατηρείται σταθερό και το bit0 ολισθαίνει στη σημαία C.
Ενημέρωση σημαιών: Z, C, N, S, V
Παράδειγμα: asr r5

SWAP Rd

Λειτουργία: R(7:4) ←Rd(3:0), R(3:0) ←Rd(7:4)
Περιγραφή: Εναλλαγή των τμημάτων high και low ενός byte.
Παράδειγμα: swap r1

BSET s

Λειτουργία: SREG(s) ←1
Περιγραφή: Ενεργοποίηση σημαίας του καταχωρητή κατάστασης.
Ενημέρωση σημαιών: SREG(s)
Παράδειγμα: bset 5

BCLR s

Λειτουργία: SREG(s) ←0
Περιγραφή: Μηδενισμός σημαίας του καταχωρητή κατάστασης.
Ενημέρωση σημαιών: SREG(s)
Παράδειγμα: bclr 7

SBI A, s

Λειτουργία: I/O(A,b) ←1
Περιγραφή: Ενεργοποίηση του bit s ενός καταχωρητή εισόδου-εξόδου.
Παράδειγμα: sbi portc,0

CBI A, s

Λειτουργία: I/O(A,b) ←0
Περιγραφή: Μηδενισμός του bit s ενός καταχωρητή εισόδου-εξόδου.
Παράδειγμα: cbi portc, 0

BST Rd, b

Λειτουργία: T ←Rd(b)
Περιγραφή: Αποθήκευση του bit b του καταχωρητή Rd στη σημαία T του SREG.
Ενημέρωση σημαιών: T
Παράδειγμα: bst r2,1

BLD Rd, b

Λειτουργία: Rd(b) ←T
Περιγραφή: Αποθήκευση της σημαίας T του SREG στο bit b του καταχωρητή Rd.
Παράδειγμα: bld r2,1

SEC Λειτουργία: C ←1
Περιγραφή: Ενεργοποίηση της σημαίας κρατουμένου του SREG.
Ενημέρωση σημαιών: C

CLC Λειτουργία: C ←0
Περιγραφή: Μηδενισμός της σημαίας κρατουμένου του SREG.
Ενημέρωση σημαιών: C

SEN Λειτουργία: N ←1
Περιγραφή: Ενεργοποίηση της σημαίας αρνητικού προσήμου του καταχωρητή κατάστασης. **Ενημέρωση σημαιών**: N

CLN Λειτουργία: N ←0
Περιγραφή: Μηδενισμός της σημαίας αρνητικού προσήμου.
Ενημέρωση σημαιών: N

SEZ Λειτουργία: Z ←1
Περιγραφή: Ενεργοποίηση της σημαίας μηδενισμού του καταχωρητή κατάστασης.
Ενημέρωση σημαιών: Z

CLZ Λειτουργία: Z ←0
Περιγραφή: Απενεργοποίηση της σημαίας μηδενισμού.
Ενημέρωση σημαιών: Z

SEI Λειτουργία: I ←1
Περιγραφή: Ενεργοποίηση της σημαίας διακοπών του καταχωρητή κατάστασης.
Ενημέρωση σημαιών: I

CLI Λειτουργία: I ←0
Περιγραφή: Απενεργοποίηση της σημαίας διακοπών. **Ενημέρωση σημαιών**: I

SES Λειτουργία: S ←1
Περιγραφή: Ενεργοποίηση της σημαίας προσήμου του SREG.
Ενημέρωση σημαιών: S

CLS Λειτουργία: S ←0
Περιγραφή: Απενεργοποίηση της σημαίας προσήμου.

SEV Λειτουργία: V ←1
Περιγραφή: Ενεργοποίηση της σημαίας υπερχείλισης στην αριθμητική συμπληρώματος 2.
Ενημέρωση σημαιών: V

CLV Λειτουργία: V ←0
Περιγραφή: Απενεργοποίηση της σημαίας υπερχείλισης.
Ενημέρωση σημαιών: V

SET Λειτουργία: T ←1
Περιγραφή: Ενεργοποίηση της σημαίας αντιγραφής και αποθήκευσης.
Ενημέρωση σημαιών: T

CLT Λειτουργία: T ←0
Περιγραφή: Απενεργοποίηση της σημαίας αντιγραφής και αποθήκευσης.
Ενημέρωση σημαιών: T

SEH Λειτουργία: H ←1
Περιγραφή: Ενεργοποίηση της σημαίας δεκαδικού κρατουμένου.
Ενημέρωση σημαιών: H

CLH Λειτουργία: H ←0
Περιγραφή: Απενεργοποίηση της σημαίας δεκαδικού κρατουμένου.
Ενημέρωση σημαιών: H

SLEEP
Περιγραφή: Θέτει το κύκλωμα σε ανενεργό κύκλο που προσδιορίζεται από τον καταχωρητή ελέγχου της MCU.

WDR
Περιγραφή: Επαναθέτει το χρονιστή επιτήρησης.

NOP
Περιγραφή: Καμία λειτουργία.