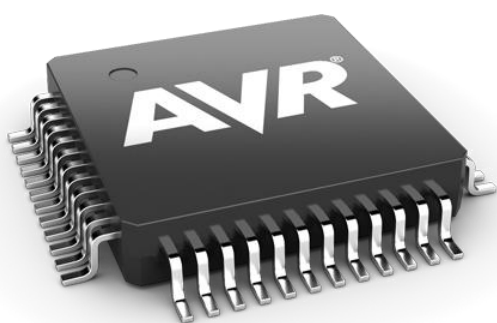




ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΜΟΥΣΙΚΗΣ ΚΑΙ ΟΠΤΟΑΚΟΥΣΤΙΚΩΝ ΤΕΧΝΟΛΟΓΙΩΝ
ΤΜΗΜΑ ΜΟΥΣΙΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ & ΑΚΟΥΣΤΙΚΗΣ

Συστήματα Μικροεπεξεργαστών

(ΤΑ 6018)
Σημειώσεις Εργαστηρίου



Συγγραφική Ομάδα

Δρ. Ορφανός Γιάννης, ΕΔΙΠ

Δρ. Παναγιώτης Γρ. Ζέρβας, Καθηγητής Εφαρμογών

Βενιέρης Μανώλης, Ηλεκτρολόγος Μηχανικός

Παπαδάκης Γιάννης, Msc Τεχνολόγος Ηλεκτρονικός

Κασελούρης Βαγγέλης, Msc Φυσικός Εφαρμογών

Φεβρουάριος 2020

Περιεχόμενα

Εργαστήριο 1ο - Εισαγωγή στον AVR.....	5
1. Εισαγωγή.....	6
2. Κατηγορίες μικροελεγκτών AVR	6
3. Εξοικείωση με το περιβάλλον AVR Studio 4.....	10
4. Περιγραφή οθόνη εργασίας	11
5. Εισαγωγή προγράμματος και αποσφαλμάτωση (Debug)	12
6. Ανάλυση κώδικα γραμμή προς γραμμή	13
7. Παραδείγματα.....	14
Εργαστήριο 2ο - Εντολές Μεταφοράς Δεδομένων σε Καταχωρητές και Μνήμη	17
1. Εισαγωγή.....	18
2. Καταχωρητής.....	18
3. Διαφορετικοί καταχωρητές	19
4. Τρόποι Διευθυνσιοδότησης.....	19
5. Παραδείγματα:.....	21
Παράδειγμα 1ο - Γράψιμο/Διάβασμα από/προς μνήμη με άμεσο τρόπο	21
Παράδειγμα 2ο - Χρήση καταχωρητών διεύθυνσης	22
6. Ασκήσεις 2ης Εργαστηριακής Άσκησης:	23
Εργαστήριο 3ο - Πρόσθεση και Αφαίρεση δεκαεξαδικών (8-bit/16-bit) αριθμών.....	25
1. Εισαγωγή.....	26
2. Εντολές Πρόσθεσης	26
3. Εντολές Αφαίρεσης	27
4. Καταχωρητής Κατάστασης - Status REGister (SREG).	27
5. Έλεγχος SREG - Μνήμης & Καταχωρητών.....	27
6. Παραδείγματα:.....	29
Παράδειγμα 1ο - Πρόσθεση 8bit αριθμών	29
Παράδειγμα 2ο - Πρόσθεση 16bit αριθμών	30
7. Ασκήσεις 3ης Εργαστηριακής Άσκησης:	31
Εργαστήριο 4ο - Λογικές Πράξεις	33
1. Εισαγωγή.....	34
2. Εντολές λογικών πράξεων.....	34
3. Παραδείγματα:.....	35
Παράδειγμα 1ο - Λογικές πράξεις στον AVR (AND, OR).....	35
Παράδειγμα 2ο - Αλλαγή τιμής συγκεκριμένων bit του περιεχομένου μιας θέσης μνήμης	36
4. Ασκήσεις 4ης Εργαστηριακής Άσκησης:	37
Εργαστήριο 5ο - Διαγράμματα Ροής – Διακλαδώσεις – Λήψη αποφάσεων	39
1. Εισαγωγή.....	40
2. Διαγράμματα Ροής.....	40
Λήψη αποφάσεων.....	41
3. α) Διακλάδωση χωρίς συνθήκη	41
4. β) Διακλάδωση υπό συνθήκη	42
• Διακλαδώσεις που γίνεται έλεγχος byte:	42
• Διακλαδώσεις που γίνεται έλεγχος ενός bit:	43
Βοηθητικές εντολές.....	43
• Αύξηση ή μείωση του περιεχομένου του καταχωρητή	43
• Αύξηση ή μείωση της διεύθυνσης μνήμης κατά 1	43
Παραδείγματα:.....	44

Παράδειγμα 1 ^ο - Αντιγραφή αριθμού σε συνεχόμενες θέσεις μνήμης	44
Παράδειγμα 2 ^ο - Σύγκριση 2 αριθμών και αποθήκευση σε συγκεκριμένες θέσεις μνήμης	
SRAM.....	44
Ασκήσεις 5 ^{ης} Εργαστηριακής Άσκησης:	47
Εργαστήριο 6 ^ο – Υποπρογράμματα - Λειτουργίες I/O (Εισόδου-Εξόδου).....	49
1. Εισαγωγή.....	50
2. Κλήση υπορουτινών - υποπρογραμμάτων	50
3. Δείκτης Στοιβάς (Stack Pointer - SP)	50
4. Εντολές εισόδου-εξόδου (I/O)	51
5. Παραδείγματα:.....	52
• Παράδειγμα 1 ^ο - Έλεγχος LED με χρήση θυρών & Υποπρόγραμμα καθυστέρησης	52
Αποσύνθεση (disassembly) του προγράμματος από τον μεταφραστή (compiler)	53
Δεκαεξαδική μορφή του προγράμματος (γλώσσα μηχανής)	54
6. Ασκήσεις 6 ^{ης} Εργαστηριακής Άσκησης:	55
Εργαστήριο 7 ^ο – Πλατφόρμα ανάπτυξης Arduino.....	57
1. Εισαγωγή.....	58
2. Πλακέτα ανάπτυξης Arduino Uno.....	58
3. Προγραμματισμός του Arduino	60
4. Ανάπτυξη προγραμμάτων στο περιβάλλον Arduino	61
5. Παραδείγματα:.....	65
• Παράδειγμα 1 ^ο - Αναβόσβημα LED	65
• Παράδειγμα 2 ^ο - Ανάγνωση Ψηφιακής Εισόδου	65
Ασκήσεις 7 ^{ης} Εργαστηριακής Άσκησης:	67
Εργαστήριο 8 ^ο – Αναλογική επικοινωνία & Δομές ελέγχου στον Arduino	69
1. Εισαγωγή.....	70
2. Αναλογική επικοινωνία.....	70
3. Συγκρίσεις	72
4. Δομές ελέγχου.....	73
5. Παραδείγματα:.....	74
Παράδειγμα 1 ^ο – Διάβασμα τιμής ποτενσιόμετρου	74
Παράδειγμα 2 ^ο – Λειτουργία RGB Led	75
Ασκήσεις 8 ^{ης} Εργαστηριακής Άσκησης:	77
Εργαστήριο 9 ^ο –Δομές ελέγχου στον Arduino II & Είσοδοι/Εξοδοι	79
1. Εισαγωγή.....	80
2. Ορίσματα & Βιβλιοθήκες	80
3. Τελεστές λογικών πράξεων.....	83
4. Αυξήσεις & Μειώσεις μεταβλητών (++ / --)	83
5. Δομή ελέγχου “for”	84
6. Συναρτήσεις συχνότητων.....	85
7. Παραδείγματα:.....	85
Παράδειγμα 1 ^ο – Παραγωγή μελωδίας	85
Παράδειγμα 2 ^ο – Ένδειξη Θερμοκρασίας	86
Ασκήσεις 9 ^{ης} Εργαστηριακής Άσκησης:	89

Εργαστήριο 1ο - Εισαγωγή στον AVR

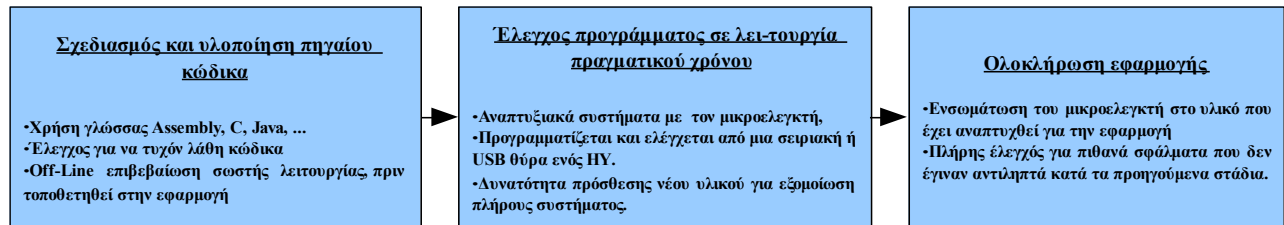
Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Κατηγορίες μικροελεγκτών AVR
3. Εξοικείωση με το περιβάλλον AVR Studio 4
 - Βήμα 1ο: Δημιουργία νέου έργου (project)
 - Βήμα 2ο: Ορισμός των παραμέτρων του νέου έργου
 - Βήμα 3ο: Ορισμός των παραμέτρων προσομοίωσης
4. Περιγραφή οθόνης εργασίας
5. Εισαγωγή προγράμματος και αποσφαλμάτωση (Debug)
6. Ανάλυση κώδικα γραμμή προς γραμμή
7. Παραδείγματα

1. Εισαγωγή

Για την υλοποίηση μιας εφαρμογής σε έναν μικροελεγκτή λαμβάνει χώρα η παράλληλη ανάπτυξη υλικού και λογισμικού. Έτσι κατά την σχεδίαση κάποιας εφαρμογής σε αυτά ο σχεδιαστής επεμβαίνει στο λογισμικό κομμάτι της εφαρμογής (software) και έτσι δεν χρειάζεται η επανασχεδίαση του υλικού (hardware). Αυτό αποτελεί και ένα σημαντικό πλεονέκτημα της ανάπτυξης εφαρμογών σε ένα μικροελεγκτή σε σχέση με τα άλλα συμβατικά κυκλώματα.

Έχοντας σαν στόχο την επίτευξη υψηλής απόδοσης κατά την ανάπτυξη μιας εφαρμογής σε έναν μικροελεγκτή είθισται να ακολουθούμε τη διαδικασία που φαίνεται στην εικόνα 1.



Εικόνα 1. Βήματα που ακολουθούμε για την ανάπτυξη μιας εφαρμογής στον AVR

2. Κατηγορίες μικροελεγκτών AVR

Η οικογένεια των μικροελεγκτών AVR χωρίζεται σε τρεις ομάδες:

- **tinyAVR**
- **AVR (Classic AVR)**
- **megaAVR**

Η διαφορά μεταξύ τους είναι τα διαθέσιμα χαρακτηριστικά κάθε μικροελεγκτή. Οι tinyAVR είναι συνήθως μικροελεγκτές με μικρότερο αριθμό ακροδεκτών (pin-count) ή περιορισμένες δυνατότητες συγκρινόμενοι με τους megaAVR's. Όλοι οι AVR έχουν το ίδιο ρεπερτόριο εντολών (instruction set) και οργάνωση μνήμης έτσι ώστε να διευκολύνεται η αλλαγή του μικροελεγκτή μιας εφαρμογής όταν αυτό απαιτηθεί.

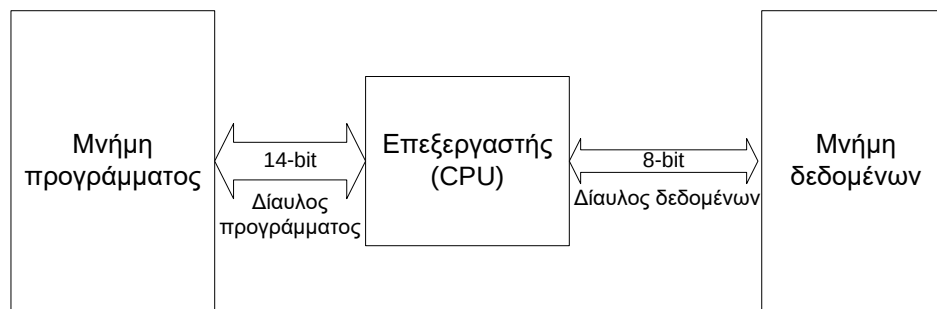
Οι μικροελεγκτές AVR εκτός από την CPU περιλαμβάνουν ένα σύνολο περιφερειακών μονάδων όπως στατική RAM (SRAM), EEPROM, διαύλους διασύνδεσης με εξωτερική SRAM μετατροπέα αναλογικού σήματος σε ψηφιακό (Analog to Digital Converter), μονάδα πολλαπλασιασμού (Hardware Multiplier), μονάδες σύγχρονης ή/και ασύγχρονης επικοινωνίας (UART, USART) και πολλά άλλα περιφερειακά.

Αν από οποιονδήποτε μικροελεγκτή AVR (Tiny, Classic ή Mega αφαιρέσουμε όλα τα περιφερειακά θα μείνει η κεντρική μονάδα επεξεργασίας (AVR Core). Η κεντρική αυτή μονάδα είναι ίδια για όλα τα μέλη της οικογένειας των AVR. Όταν θέλουμε να επιλέξουμε τον κατάλληλο AVR για μία εφαρμογή πρέπει να έχουμε υπόψη ότι ο χαρακτηρισμός tinyAVR, Classic AVR και megaAVR δεν είναι χαρακτηρισμός επιδόσεων αλλά ένδειξη της πολυπλοκότητας του μικροελεγκτή:

- Πολλά περιφερειακά → megaAVR,
- Περιορισμένος αριθμός περιφερειακών → tinyAVR.

Ο κλασσικός AVR (Classic AVR) είναι κάπου στη μέση αν και οι διακρίσεις μεταξύ των ομάδων γίνονται όλο και πιο συγκεχυμένες. Επομένως, αν πρέπει να επιλέξουμε έναν AVR με βάση το κόστος πρέπει να επιλέξουμε αυτόν με τα λιγότερα περιφερειακά που ικανοποιεί τις ανάγκες της εφαρμογής. Αν όμως δεν υπάρχει περιορισμός κόστους καλό είναι να χρησιμοποιήσουμε κάποιον με περισσότερες δυνατότητες ώστε να μπορέσουμε ευκολότερα στο μέλλον να βελτιώσουμε τα χαρακτηριστικά της εφαρμογής. Για την ανάπτυξη εφαρμογών με τους μικροελεγκτές AVR η ATMEL μεταξύ άλλων προσφέρει το περιβάλλον ανάπτυξης προγραμμάτων AVR Studio και το αναπτυξιακό σύστημα STK500, τα οποία χρησιμοποιούνται στα πλαίσια του εργαστηρίου. Στόχος της πρώτης άσκησης είναι η γνωριμία των σπουδαστών με τα προαναφερόμενα συστήματα.

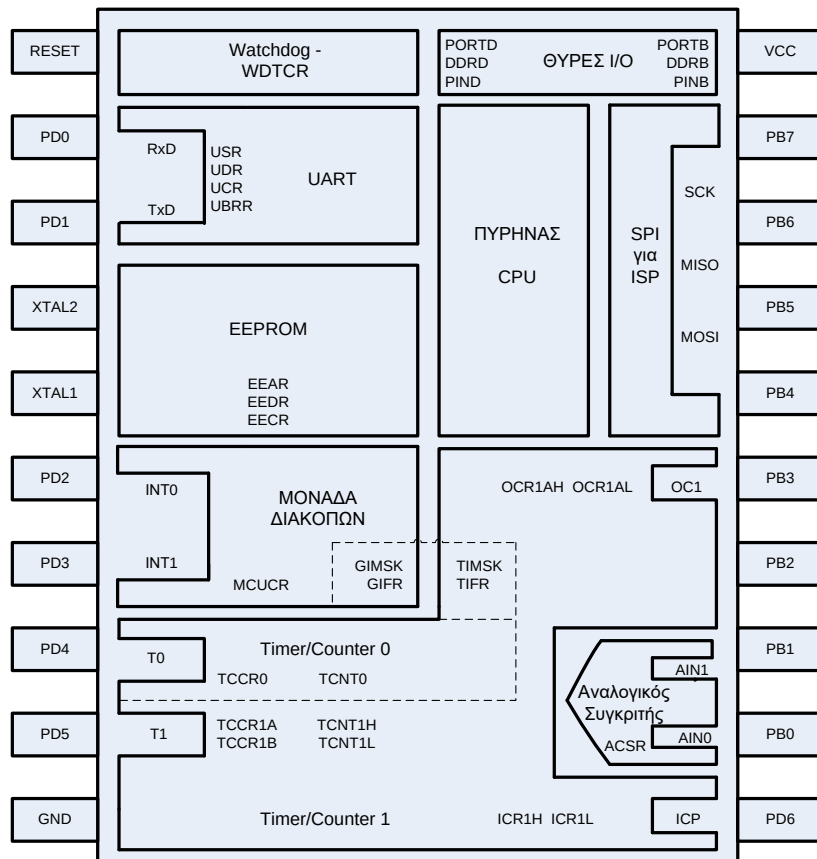
Σκοπός του κεφαλαίου είναι η περιγραφή των μικροελεγκτών AVR στο πεδίο της αρχιτεκτονικής και των περιφερειακών μονάδων. Ο μικροελεγκτής AVR περιέχει έναν επεξεργαστή RISC (Reduced Instruction Set Computer) ο οποίος έχει σχεδιαστεί με βάση την αρχιτεκτονική Harvard. Όπως φαίνεται από το σχήμα, στην αρχιτεκτονική Harvard υπάρχει διαφορετικός δίαυλος για τη μεταφορά δεδομένων (data bus) και διαφορετικός δίαυλος 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 της εντολής σε ένα μόνο χρόνο μηχανής.



Σχήμα 1.1 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(In System Programmable)- σαν μνήμη προγράμματος. Δηλαδή δεν είναι απαραίτητες εξωτερικές μνήμες τύπου rom ή erom που να περιέχουν τον κώδικα του προγράμματος.
- β.** Οι μικροελεγκτές διαθέτουν 32 καταχωρητές εργασίας των 8-bit. Συνεπώς, οι διάφορες μεταβλητές μπορούν να αποθηκεύονται εντός της CPU 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.** Λειτουργία σε συχνότητες χρονισμού από 0 έως 10MHz. Καθώς οι περισσότερες εντολές ολοκληρώνονται σε μια περίοδο του κεντρικού σήματος χρονισμού, η επίδοση είναι βελτιωμένη σε σύγκριση με τους κλασσικούς μικροελεγκτές που λειτουργούν στην ίδια συχνότητα χρονισμού.
- ε.** Εσωτερικό κύκλωμα που επανατοποθετεί το σύστημα κατά την εφαρμογή της τάσης τροφοδοσίας (Power On Reset-POR).
- στ.** Οι μικροελεγκτές διαθέτουν ενσωματωμένο προγραμματιζόμενο χρονιστή με μονάδα διαίρεσης συχνότητας(prescaler) . Η μονάδα αυτή χρησιμοποιείται σε εφαρμογές κρίσιμου χρονισμού.
- ζ.** Εσωτερικές και εξωτερικές διακοπές.
- η.** Προγραμματιζόμενο χρονιστή επιτήρησης (Watch Dog Timer-WDT). Ο συνήθης ρόλος του είναι η αποφυγή εσφαλμένων λειτουργιών σε περίπτωση κατάρρευσης του προγράμματος.
- θ.** Λειτουργίες αποκοπής (Power down) και ηρεμίας (Sleep) .Ο επεξεργαστής τίθεται στις καταστάσεις αυτές όταν δεν υπάρχει δραστηριότητα, ούτως ώστε να μειωθεί η κατανάλωση ισχύος.
- ι.** Διάφορα μοντέλα διαθέτουν ενσωματωμένο ταλαντωτή τύπου RC.
- κ.** Πληθώρα περιφερειακών, όπως UART, SPI, αναλογικός συγκριτής, μετατροπέας adc, κλπ



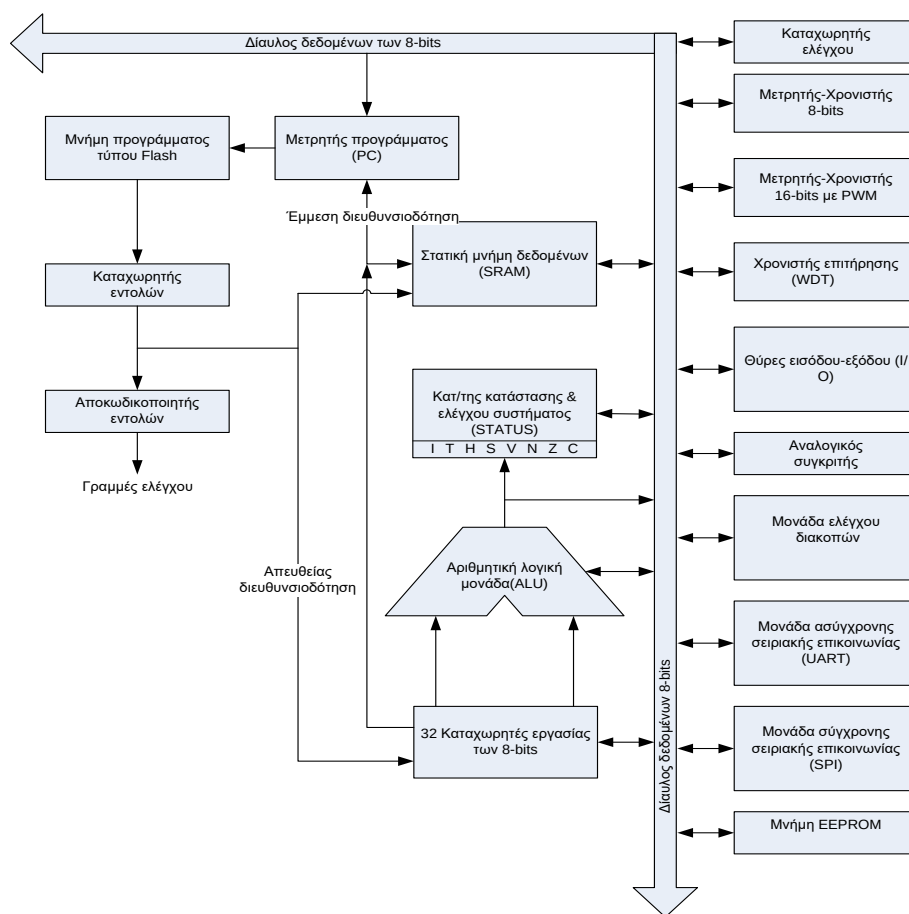
Σχήμα 1.2 Μικροελεγκτής και περιφερειακά

Διάφορα μοντέλα μικροελεγκτών και τα χαρακτηριστικά τους:

Πίνακας 1.1 Διάφορα μοντέλα μικροελεγκτών AVR

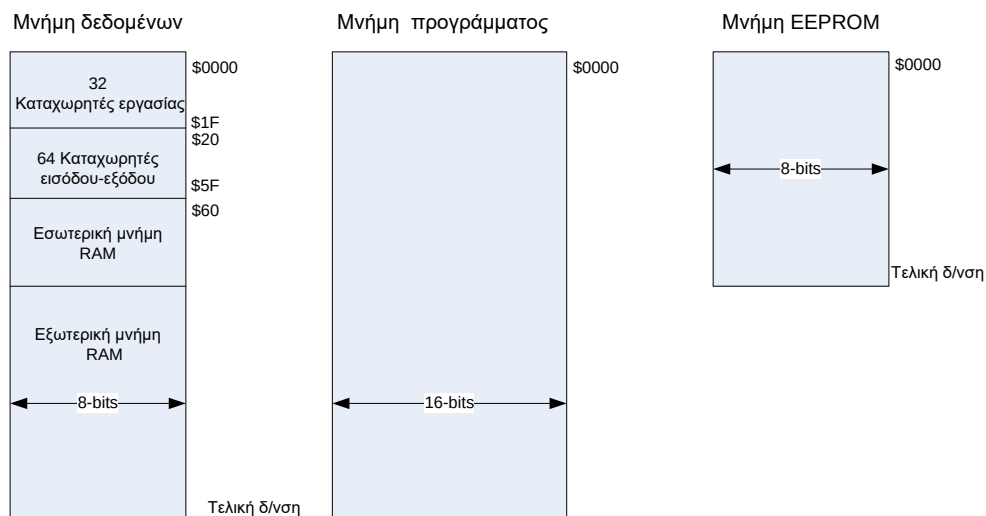
Μοντέλο	Ακροδέκτες	Flash	EEPROM	RAM	UART	ADC
90S1200	20	1K	64 bytes	0	Όχι	Όχι
90S2313	20	2K	128	128	Ναι	Όχι
90S2323	8	2K	128	128	Όχι	Όχι
90S2333	28	2K	128	128	Ναι	Ναι
90S4433	28	4K	256	128	Ναι	Ναι
90S4414	40	4K	256	256	Ναι	Όχι
90S8515	40	8K	512	512	Ναι	Όχι
90S4434	40	4K	256	256	Ναι	Ναι
90S2343	8	2K	128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

Η αρχιτεκτονική των επεξεργαστών AVR με τις διάφορες λειτουργικές μονάδες φαίνεται στο Σχήμα 1.3.



Σχήμα 1.3 Αρχιτεκτονική των μικροελεγκτών AVR

Όπως φαίνεται στο σχήμα, ένας μικροελεγκτής διαθέτει τα εξής είδη μνήμης: (α) μνήμη δεδομένων, (β) μνήμη προγράμματος τύπου flash και (γ) μνήμη EEPROM με αρχική διεύθυνση \$0000 και χωρητικότητα από 64bytes ως 4Kbytes.



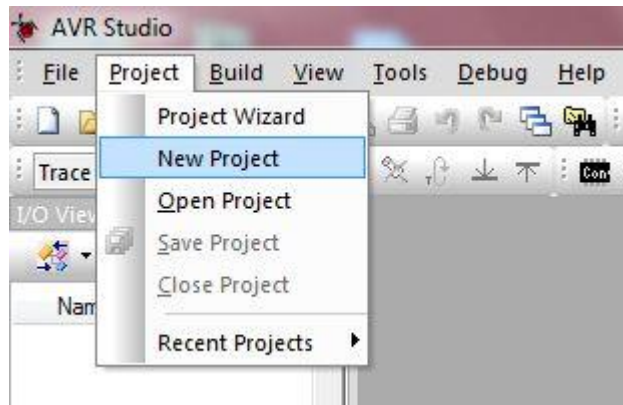
Σχήμα 1.4 Τύποι μνήμης

3. Εξοικείωση με το περιβάλλον AVR Studio 4

Όπως είδαμε στην εισαγωγή το πρώτο βήμα για την ανάπτυξη μιας εφαρμογής στον μικροελεγκτή είναι ο σχεδιασμός και η υλοποίηση του πηγαίου κώδικα. Αυτό μπορεί να γίνει με την χρήση του προγράμματος AVR Studio αφού πρώτα το εγκαταστήσουμε στον ΗΥ μας. Το πρόγραμμα αυτό μπορεί να ληφθεί από τον δικτυακό τόπο της εταιρίας ATMEL (<http://www.atmel.com/products/avr/>).

Παρακάτω θα αναφέρουμε βήμα προς βήμα την διαδικασία εκείνη που πρέπει να ακολουθηθεί για την συγγραφή και εκτέλεση ενός προγράμματος για τον AVR.

Βήμα 1ο: Δημιουργία νέου έργου (project)



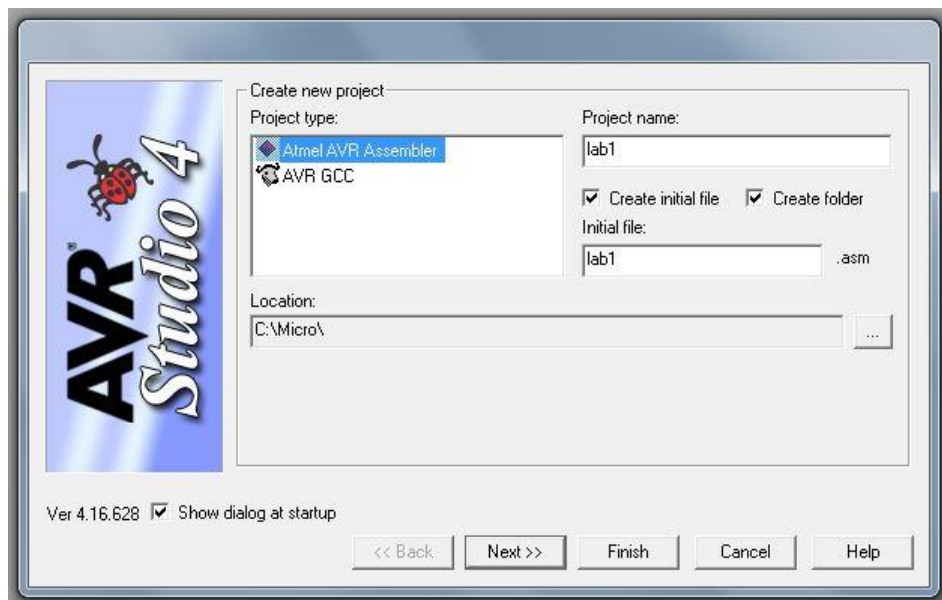
Εικόνα 2. Επιλογή για δημιουργία νέου ή άνοιγμα ήδη υπάρχοντος project

Η φιλοσοφία λειτουργίας του AVR Studio είναι ότι κάθε πρόγραμμα αποθηκεύεται σε ένα φάκελο μαζί με όλα τα αρχεία που προκύπτουν κατά την μεταγλώττιση του. Στην εικόνα 2 βλέπουμε την επιλογή **"Project"** του μενού του AVR Studio όπου περιέχονται όλες οι λειτουργίες για την διαχείριση των project. Στην περίπτωση μας αφού θέλουμε να δημιουργήσουμε ένα νέο project θα επιλέξουμε την επιλογή **"New Project"**. Στο παράθυρο αυτό μπορείτε να επιλέξετε να δουλέψετε ένα έργο που ήδη υπάρχει με το πλήκτρο "Open" ή να δημιουργήσετε ένα νέο έργο. Επιλέξτε το **"New Project"** ώστε να εμφανιστεί η οθόνη επιλογών του project

Βήμα 2ο: Ορισμός παραμέτρων του νέου project

Επόμενο βήμα είναι ο ορισμός παραμέτρων για το project στο οποίο θα δουλέψουμε. Η διαδικασία που ακολουθήσαμε στο 1ο βήμα έχει σαν αποτέλεσμα την εμφάνιση του παραθύρου της εικόνας 3. Τα μέρη από τα οποία απαρτίζεται το παράθυρο αυτό είναι:

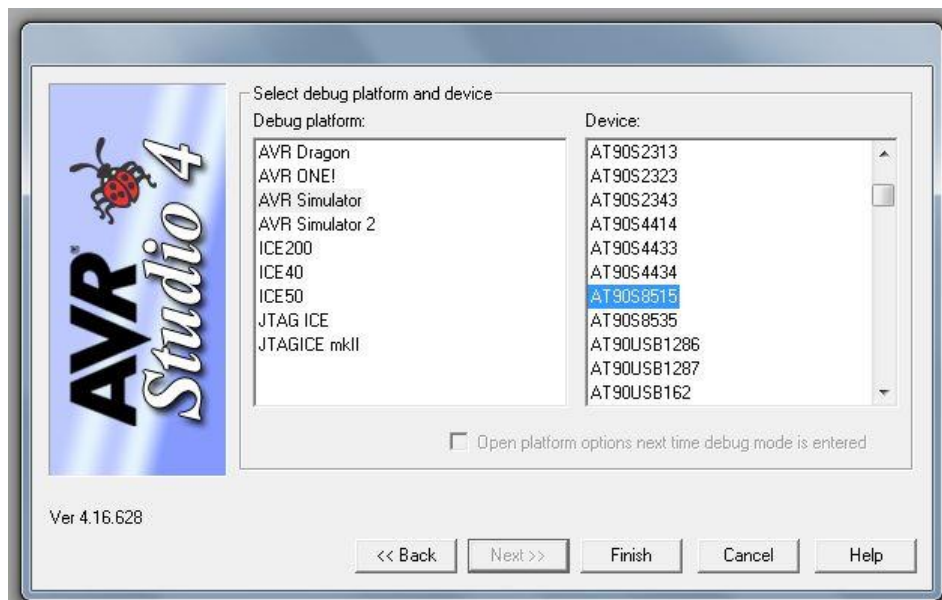
1. **Project Type:** Σε αυτό το σημείο θα πρέπει να ορίσουμε το είδος του project που θέλουμε να δημιουργήσουμε. Στην περίπτωση μας θα επιλέγουμε πάντα project για συμβολομεταφραστή, δηλαδή **"Atmel AVR Assembler"**.
2. **Project Name:** Εδώ δίνουμε το όνομα του project που δημιουργήσαμε. Στα πλαίσια των εργαστηριακών ασκήσεων σαν όνομα project θα δίνουμε την λέξη *lab* και τον αριθμό της άσκησης που κάνουμε. π.χ. Αν κάνουμε την άσκηση 1 το όνομα του project που θα δημιουργήσουμε θα είναι **"lab1"**.
3. **Initial File:** Ως **"initial file"** καλείται το αρχείο που θα περιέχει τον κώδικα που θα αναπτύξουμε. Αν θέλουμε να δημιουργήσει το AVR Studio αυτόματα το αρχείο αυτό θα πρέπει να έχουμε επιλέξει το κουτάκι **"Create Initial File"**.
4. **Location:** Εδώ ορίζουμε τον φάκελο στον οποίο θα δημιουργηθεί το νέο project. Για την διεξαγωγή των εργαστηρίων οι φάκελοι των project θα έχουν όνομα την λέξη *lab* και το νούμερο της άσκησης. π.χ. Για την άσκηση 1 το όνομα του φακέλου θα είναι *lab1* και θα βρίσκεται πάντα στον φάκελο **"D:\Micro"**. Για να αναζητήσουμε τον φάκελο θα πατάμε το κουμπί με τις τρεις τελείες. **Προσοχή!** Μη χρησιμοποιείτε ελληνικούς χαρακτήρες στο όνομα του έργου.



Εικόνα 3. Παράθυρο δημιουργίας νέου project

Βήμα 3ο: Ορισμός παραμέτρων προσομοίωσης

Αφού συμπληρώσουμε σωστά όλα τα στοιχεία μπορούμε να πατήσουμε το κουμπί **“NEXT”** για να εμφανιστεί η οθόνη επιλογών αποσφαλμάτωσης και μοντέλου μικροελεγκτή (debug platform and device), εικόνα 4. Το AVR Studio 4 μπορεί να υποστηρίξει και να χρησιμοποιηθεί από ένα ευρύ φάσμα εργαλείων αποσφαλμάτωσης. Επειδή όμως χρησιμοποιούμε τη βασική του έκδοση, επιλέξτε σαν εξομοιωτή τον **“AVR simulator”**. Στην συνέχεια σαν συσκευή **“Device”** εξομοίωσης, επιλέξτε το μοντέλο AVR με το οποίο θα δουλέψετε. (Στα πλαίσια του εργαστηρίου επιλέξτε τον **AT90S8515**).



Εικόνα 4. Επιλογή οικογένειας AVR

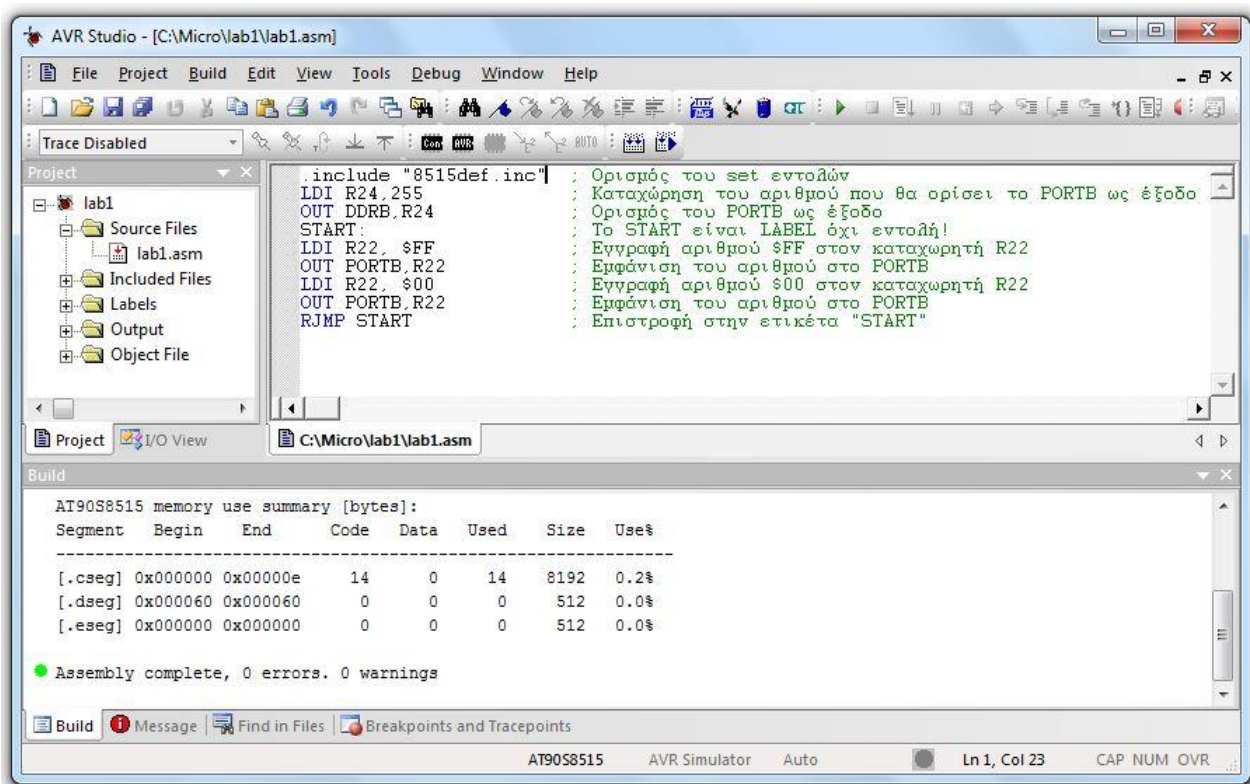
Αφού επαληθεύσουμε όλες τις επιλογές μας και πατήσουμε το κουμπί **“Finish”** δημιουργείται το project και εμφανίζεται η οθόνη εργασίας του AVR Studio4 όπου μπορείτε να γράψετε το πρόγραμμα για το συμβολομεταφραστή, εικόνα 5.

4. Περιγραφή οθόνη εργασίας

Η οθόνη εργασίας του AVR Studio 4 περιλαμβάνει γραμμή των menus όπου υπάρχουν ομαδοποιημένες όλες οι παρεχόμενες λειτουργίες του. Έτσι λοιπόν κάτω από την επιλογή **“File”** υπάρχουν λειτουργίες σχετικές με την

διαχείριση των αρχείων (new, open, save, print, close κ.α.), στην **“Project”** λειτουργίες για την διαχείριση των project, στην **“Edit”** για την επεξεργασία του κειμένου (Cut, Paste, κ.α.), στην **“Tools”** για την ενεργοποίηση εργαλείων εξομοίωσης του AVR όπως έλεγχος των register κ.α. και την επιλογή **“Debug”** όπου έχουμε λειτουργίες όπως η έναρξη μεταγλώττισης (Build, Run, Debug κ.α.). Κάτω από το μενού έχουμε τις μπάρες εργασίας **“Toolbars”** (οι οποίες είναι συντομεύσεις για εργασίες που εκτελούνται συχνά, (π.χ. Build & Run, memory window).

Πέραν των παραπάνω έχουμε και την εμφάνιση παραθύρων για την εισαγωγή του προγράμματος και τη λειτουργία του μικροελεγκτή. Συγκεκριμένα, αριστερά του περιβάλλοντος εργασίας βλέπουμε την στήλη με το όνομα **“Workspace”** η οποία αποτελείται από τρεις καρτέλες. Η καρτέλα με το όνομα **“project”** περιέχει πληροφορίες για τα αρχεία του έργου. Το κεντρικό κομμάτι της επιφάνειας εργασίας καταλαμβάνεται από το παράθυρο που έχει το όνομα του αρχείου assembly. Εκεί λαμβάνει χώρα η εισαγωγή του προγράμματος (Text editor). Στο κάτω μέρος υπάρχει το παράθυρο εξόδου **“output window”** όπου δίνονται διάφορες πληροφορίες για την κατάσταση του AVR Studio ανάλογα με την εργασία που εκτελείται κάθε φορά και ανάλογα με την επιλογή που έχουμε κάνει (build, message κλπ). **Προσοχή!** Εκεί θα εμφανίζονται τα λάθη του συμβολομεταφραστή (Assembler) και με διπλό κλικ πάνω στο μήνυμα λάθους ο δρομέας (cursor) πηγαίνει αυτόματα στο παράθυρο εισαγωγής κειμένου στη γραμμή που παρουσιάζεται το σφάλμα.



Εικόνα 5. Περιβάλλον επεξεργασίας πηγαίου κώδικα

Τέλος η καρτέλα **“I/O View”** δίνει πληροφορίες για την κατάσταση του μικροελεγκτή κατά τη διάρκεια εκτέλεσης του προγράμματος, μας δίνει τη δυνατότητα να δούμε με λεπτομέρεια τις τιμές όλων των καταχωρητών, όπως επίσης και των θυρών του μικροελεγκτή, έτσι μπορούμε να προσδιορίσουμε τυχόν σφάλματα. Στην τελευταία γραμμή της οθόνης εμφανίζονται μερικές επιπλέον πληροφορίες για τη λειτουργία του AVR Studio και την εκτέλεση του προγράμματος.

5. Εισαγωγή προγράμματος και αποσφαλμάτωση (Debug)

Ακολουθώντας τα βήματα που περιγράψαμε προηγουμένως δημιουργήστε έργο (project) με όνομα lab1 το οποίο θα εναλλάσσει την τιμή FF και 00 στον καταχωρητή R22 και στην συνέχεια θα την εμφανίζει στο PORTB.

Στο παράθυρο του editor, εικόνα 5, γράψτε το παρακάτω πηγαίο κώδικα και σώστε τον.

Προσοχή! Η γλώσσα assembly δεν κάνει διάκριση μεταξύ μικρών και κεφαλαίων γραμμάτων (δεν είναι case sensitive). Για την εισαγωγή σχολίων χρησιμοποιούμε τον χαρακτήρα **“;”** στην αρχή αυτών.

```

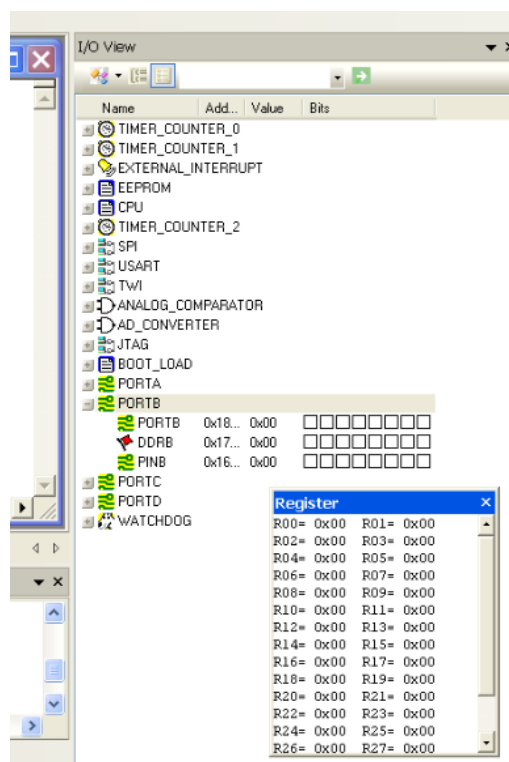
1.  .include "8515def.inc"      ; Ορισμός του set εντολών
2.  LDI R24,255                ; Καταχώρηση του αριθμού που θα ορίσει το PORTB ως έξοδο
3.  OUT DDRB,R24               ; Ορισμός του PORTB ως έξοδος
4.  START:                    ; Προσοχή! Το START είναι LABEL όχι εντολή!
5.  LDI R22, $FF               ; Εγγραφή αριθμού $FF στον καταχωρητή R22
6.  OUT PORTB,R22              ; Εμφάνιση του αριθμού στο PORTB
7.  LDI R22, $00               ; Εγγραφή αριθμού $00 στον καταχωρητή R22
8.  OUT PORTB,R22              ; Εμφάνιση του αριθμού στο PORTB
9.  RJMP START                 ; Επιστροφή στην ετικέτα "START"

```

6. Ανάλυση κώδικα γραμμή προς γραμμή

1. Η γραμμή αυτή περιέχει μια εντολή η οποία απευθύνεται στον assembler και όχι στον μικροελεγκτή. Παρατηρήστε ότι η γραμμή αυτή ξεκινάει με τελεία (.). Αυτή η εντολή (assembler directive) ενσωματώνει στον πηγαίο μας κώδικα πληροφορίες σχετικά με ορισμούς ονομάτων των καταχωρητών (στην προκειμένη περίπτωση μας, του AT90S8515). Αυτό το αρχείο έχει γραφεί από την Atmel. **Προσοχή!** σε περίπτωση που χρησιμοποιούμε άλλο μικροελεγκτή το όνομα του αρχείου αυτού πρέπει να αλλάξει.
2. Με την γραμμή αυτή και την παρακάτω, καταχωρώντας δηλαδή στον καταχωρητή R24 τον αριθμό 255,
3. και έπειτα στέλνοντας τον αριθμό που περιέχει ο R24 στο DDRB, ορίζουμε το PORTB ως έξοδο.
4. Η γραμμή αυτή είναι μια ετικέτα που χρησιμοποιούμε όταν θέλουμε να στείλουμε την ροή του προγράμματος μας σε αυτό το σημείο.
5. Με την εντολή της γραμμής αυτής “φορτώνουμε” τον καταχωρητή R22 με τον αριθμό FF.
6. Η εντολή της γραμμής αυτής στέλνει το περιεχόμενο του R22 στο PORTB.
7. Με την εντολή της γραμμής αυτής “φορτώνουμε” τον καταχωρητή R22 με τον αριθμό 00.
8. Η εντολή της γραμμής αυτής στέλνει το περιεχόμενο του R22 στο PORTB.
9. Η εντολή αυτή στέλνει την ροή του προγράμματος στην ετικέτα START.

Αφού περάσουμε το πρόγραμμα στον editor του AVR Studio από την επιλογή “Project” επιλέξτε την εντολή “Build” για να δημιουργηθεί το εκτελέσιμο αρχείο του προγράμματός σας. Προσοχή! Αν ο κώδικας μας έχει συντακτικά σφάλματα αυτά θα φανούν στο παράθυρο εξόδου του μεταγλωττιστή όπως φαίνεται στο κάτω μέρος της εικόνας 5.



Εικόνα 6. Παράθυρο καταχωρητών

Από την επιλογή “Debug” επιλέξτε “Start debugging”. Στο παράθυρο “I/O view” επιλέξτε να βλέπετε τις λεπτομέρειες του PORTB και από την επιλογή “View” επιλέξτε το “Register” ώστε να βλέπετε και τις τιμές των καταχωρητών γενικού σκοπού R16-R31, όπως φαίνεται στην διπλανή εικόνα 6.

7. Παραδείγματα

Παράδειγμα 1ο

Αντιγράψτε το παρακάτω πρόγραμμα. Οι εντολές δεν πρέπει να ξεκινούν από τη πρώτη στήλη (τέρμα αριστερά). Χρησιμοποιείστε τουλάχιστον ένα TAB.

```
.include "8515def.inc"
.cseg
.org 0
ldi r18,$FF
out ddrb,r18
RESET:
    ldi r16,$30
    out portb,r16
    mov r17,r16          ; r17 ← r16
    ldi r17,$40
    sub r17,r16          ; r17 ← r17 - r16
    out portb,r17
rjmp RESET
```

Το παραπάνω πρόγραμμα αφαιρεί τον αριθμό 30H από τον αριθμό 40H και “βγάζει” το αποτέλεσμα στο portb του μικροελεγκτή (*υπάρχει ένα συντακτικό λάθος στον παραπάνω κώδικα!).

Αφού γράψουμε το πρόγραμμα θα πρέπει να το μεταφράσουμε. Καλό θα είναι να το σώσουμε με την επιλογή **Save** του μενού **File**.

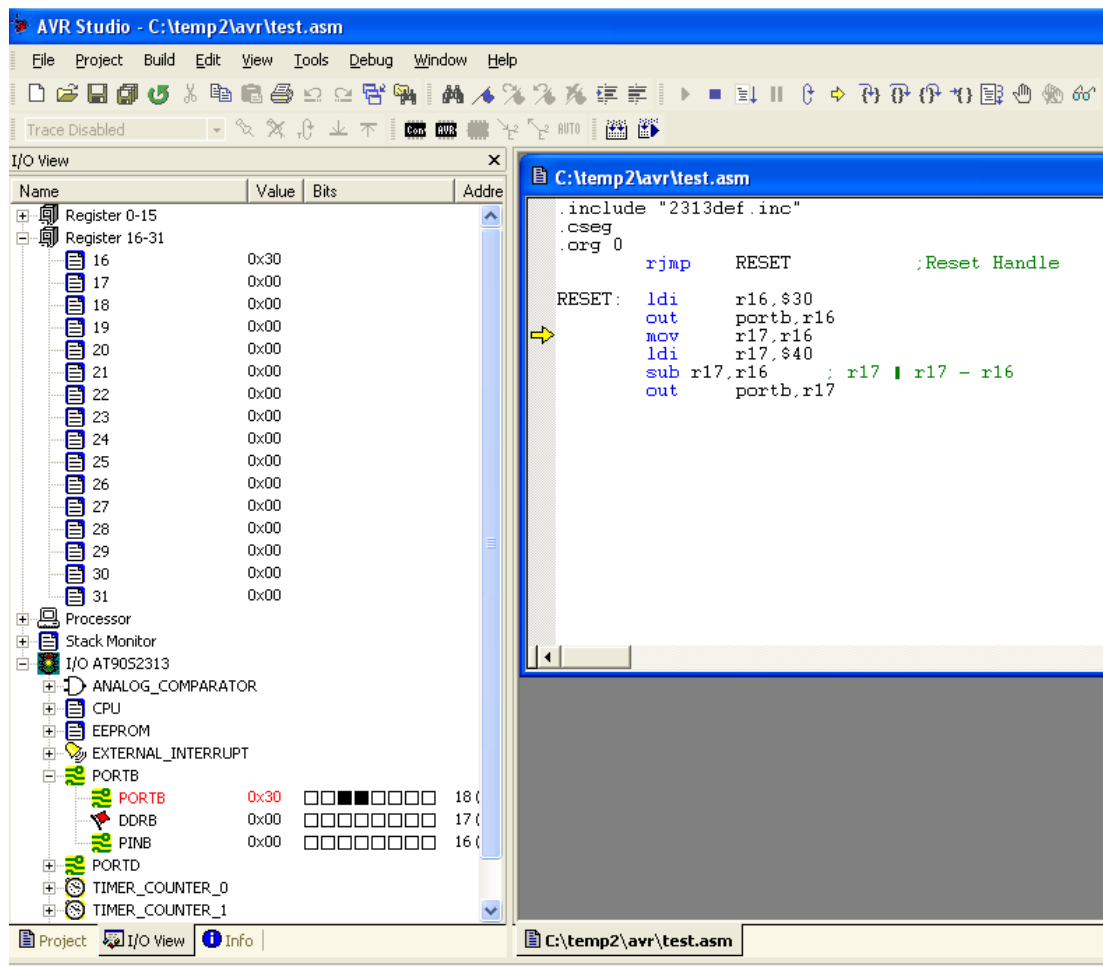
* Για να **μεταφράσουμε** το πρόγραμμα πάμε στο μενού “**Build**” και επιλέγουμε “**Build**”.

Αν η διαδικασία της μετάφρασης είναι επιτυχής (εμφάνιση μηνύματος: “**Assembly complete, 0 errors. 0 warnings**”), μπορούμε να εκτελέσουμε το πρόγραμμα στον εξομοιωτή, ή να προγραμματίσουμε τη μνήμη ROM του μικροελεγκτή με το αναπτυξιακό του εργαστηρίου. Το πέρασμα από τον εξομοιωτή βοηθά στην αποσφαλμάτωση (Debugging) του προγράμματος αφού ο εξομοιωτής μιμείται τη συμπεριφορά του επεξεργαστή κατά την εκτέλεση του προγράμματος αυτού.

Τον εξομοιωτή καλούμε από το μενού **Debug** και την επιλογή **Start Debugging**.

Στην οθόνη μπορούμε να δούμε τα εξής κατά περιοχή όπως φαίνονται στο παρακάτω σχήμα (εικόνα 7):

1. Τους καταχωρητές και τις σημαίες
2. Τμήμα του προγράμματος
3. Την κατάσταση των χρονιστών (Timer)
4. Την κατάσταση των διακοπών
5. Απεικόνιση της μνήμης
6. Την κατάσταση της σειριακής θύρας
7. Τις διευθύνσεις και τα περιεχόμενα των θυρών εισόδου εξόδου
8. Την περιοχή των εντολών
9. Την κατάσταση του εξομοιωτή



Εικόνα 7. Παράθυρο καταχωρητών

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

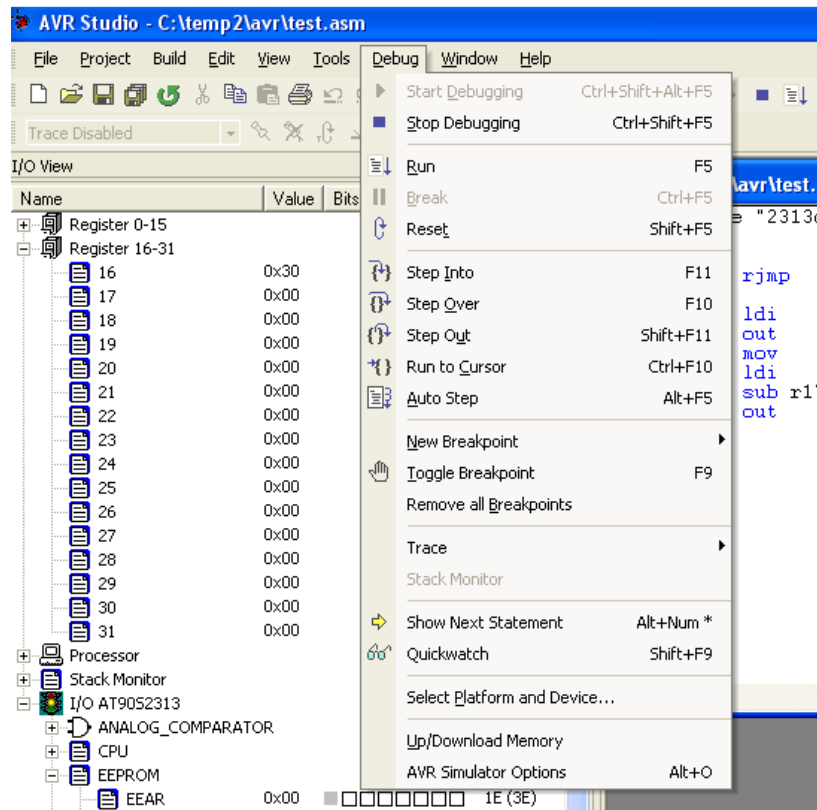
Πίνακας 1. Οι 32 καταχωρητές γενικής χρήσης του AVR

	7	0	Addr.	
General Purpose Working Registers	R0		\$00	
	R1		\$01	
	R2		\$02	
	...			
	R13		\$0D	
	R14		\$0E	
	R15		\$0F	
	R16		\$10	
	R17		\$11	
	...			
	R26		\$1A	X-register low byte
	R27		\$1B	X-register high byte
	R28		\$1C	Y-register low byte
	R29		\$1D	Y-register high byte
	R30		\$1E	Z-register low byte
	R31		\$1F	Z-register high byte

Για την ώρα να σημειώσουμε ότι όλες οι εντολές εφαρμόζονται στους Upper Registers, ενώ στους Lower Registers κάποιες δεν μπορούν να εφαρμοστούν.

Η εκτέλεση του προγράμματος

Για να τρέξουμε το πρόγραμμα δεν έχουμε παρά να χρησιμοποιήσουμε τις επιλογές του μενού “Debug”, όπως φαίνεται στην εικόνα 8.



Εικόνα 8. Παράθυρο καταχωρητών

Αναλυτικά τις επιλογές αυτές θα τις βλέπουμε κατά τη διάρκεια των μαθημάτων.

Παράδειγμα 2ο

Τρέξτε τον εξομοιωτή (simulator) με εκτέλεση βήμα – βήμα (F11). Καταγράψτε ποιοι καταχωρητές αλλάζουν περιεχόμενο και σε ποια εντολή συμβαίνει αυτό.

Εργαστήριο 2^ο - Εντολές Μεταφοράς Δεδομένων σε Καταχωρητές και Μνήμη

Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Καταχωρητής
3. Διαφορετικοί καταχωρητές
4. Τρόποι Διευθυνσιοδότησης
5. Παραδείγματα
 - Παράδειγμα 1ο - Γράψιμο/Διάβασμα από/προς μνήμη με άμεσο τρόπο 25
 - Παράδειγμα 2ο - Χρήση καταχωρητών διεύθυνσης 26
6. Ασκήσεις 2ης Εργαστηριακής Άσκησης

1. Εισαγωγή

Σκοπός του παρόντος εργαστηρίου είναι η μελέτη του τρόπου με τον οποίο μπορούμε να μεταφέρουμε δεδομένα σε θέσεις μνήμης. Η διαδικασία αυτή μπορεί να γίνει άμεσα ή έμμεσα.

2. Καταχωρητής

Οι καταχωρητές είναι ειδικές αποθηκευτικές θέσεις μνήμης με χωρητικότητα 8bit που μοιάζουν ως εξής:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
------	------	------	------	------	------	------	------

Εικόνα 9. Περιεχόμενο bit ενός καταχωρητή

Παρατηρήστε την αρίθμηση των bit: το Least Significant Bit (LSB) ξεκινάει με μηδέν ($2^0 = 1$)

Ένας καταχωρητής μπορεί να αποθηκεύσει:

- αριθμούς από 0-255 (θετικούς αριθμούς, μη αρνητικές τιμές),
- ή αριθμούς από -128 έως +127 (αριθμός με το bit πρόσημου στην θέση bit 7),
- ή τιμή που αντιστοιχεί σε έναν χαρακτήρα του κώδικα ASCII (π.χ. 'A'),
- ή απλά οκτώ bit τα οποία καμία σχέση δεν έχουν το ένα με το άλλο (π.χ. οκτώ δείκτες σε ένα καταχωρητή κατάσταση).

Ο ιδιαίτερος χαρακτήρας των καταχωρητών, συγκρινόμενος με άλλες θέσεις μνήμης, είναι ότι:

- μπορούν να χρησιμοποιηθούν απευθείας σε εντολές του assembler,
- οι εντολές απαιτούν μόνο μια δεσμευμένη λέξη στην μνήμη του προγράμματος,
- συνδέονται απευθείας με την κεντρική μονάδα επεξεργασίας (accumulator),
- είναι πηγή και δέκτης των υπολογισμών.

Η αρχιτεκτονική του AVR περιλαμβάνει 32 καταχωρητές, η ονομασία των οποίων είναι R0 σε R31, έχουμε όμως την δυνατότητα να τους αλλάξουμε όνομα δίνοντας τους νόημα χρησιμοποιώντας συγκεκριμένες εντολές (assembler directive) του assembler. Για παράδειγμα:

```
.DEF MyPreferredRegister = R16 ; Αλλαγή ονόματος καταχωρητή
```

Προσοχή: Οι *assembler directives* πάντα ξεκινούν με τελεία "." στην στήλη 1 του επεξεργαστή κειμένου. Οι εντολές (commands) δεν ξεκινούν ποτέ στην στήλη 1, πάντα προηγείται ένας στηλοθέτης (tab) ή ένας χαρακτήρας κενού (space) από αυτές. Εδώ πρέπει να σημειώσουμε ότι οι assembler directives, όπως αυτή του παραπάνω παραδείγματος, έχουν νόημα μόνο για τον assembler και δεν παράγουν κανένα κώδικα ο οποίος εκτελείται από τον AVR. Αντί λοιπόν να χρησιμοποιούμε το όνομα R16 μπορούμε τώρα να αναφερόμαστε σε αυτόν με το όνομα που του δώσαμε, δηλ. MyPreferredRegister, κάθε φορά που θα αναφερόμαστε σε αυτόν σε μια εντολή. Έτσι μπορεί να γράφουμε λίγο παραπάνω κείμενο όταν αναφερόμαστε σε αυτόν αλλά έχουμε πλέον μια ένδειξη για την χρήση του καταχωρητή αυτού.

Κάνοντας χρήση της παρακάτω εντολής:

```
LDI MyPreferredRegister, 150
```

έχει σαν αποτέλεσμα: την φόρτωση του αριθμού 150 άμεσα στον καταχωρητή R16, **Load Immediate**. Έτσι φορτώνουμε μια συγκεκριμένη τιμή ή μια σταθερά στον καταχωρητή. Ακολουθώντας την assembly ή μεταφράζοντας την γραμμή κώδικα στην μνήμη του προγράμματος του AVR θα είναι κάτι σαν:

```
000000 E906
```

Η εντολή **LDI** καθώς και ο καταχωρητής (R16) και η τιμή (150) είναι τμήματα της δεκαεξαδικής τιμής E906, αν και αυτό μπορεί να μην είναι φανερό με την πρώτη ματιά. Ευτυχώς για εμάς δεν χρειάζεται να θυμόμαστε αυτήν την κωδικοποίηση αφού ο assembler ξέρει να το κάνει ώστε να πάρει την πληροφορία που περιέχεται στο E906.

Σε κάποιες εντολές δύο διαφορετικοί καταχωρητές μπορεί να λειτουργούν ως ορίσματα. Η πιο απλή εντολή όπου φαίνεται καθαρά αυτό είναι η εντολή MOV. Με την εντολή αυτή γίνεται αντιγραφή των περιεχομένων ενός καταχωρητή σε έναν άλλο. Παράδειγμα:

```
.DEF MyPreferredRegister = R16
.DEF AnotherRegister = R15
LDI MyPreferredRegister, 150
MOV AnotherRegister, MyPreferredRegister
```

Στις δύο πρώτες γραμμές αυτού του προγράμματος έχουμε τον ορισμό ονομάτων στους καταχωρητές R16 και R15. Όπως είπαμε και προηγουμένως αυτές οι δύο εντολές δεν παράγουν κάποιο κώδικα για τον AVR.

Οι εντολές LDI και MOV παράγουν τον κώδικα:

```
000000 E906
000001 2F01
```

Οι εντολές αυτές γράφουν το 150 στον καταχωρητή R16 και στη συνέχεια το περιεχόμενο του μεταφέρτε στον R15.

Προσοχή! Ο δεύτερος καταχωρητής είναι πάντα η **πηγή** (source) και ο πρώτος ο **προορισμός** (target) που θα γραφούν τα δεδομένα (αντίθετα από ότι θα περίμενε κάποιος).

3. Διαφορετικοί καταχωρητές

Θα μπορούσε κάποιος να πει ότι το παραπάνω πρόγραμμα θα μπορούσε να γραφεί και ως εξής:

```
.DEF AnotherRegister = R15
LDI AnotherRegister, 150
```

και θα ήταν λάθος!!!!.. **Μόνο οι καταχωρητές από R16 έως και R31 μπορούν να φορτωθούν με μια τιμή άμεσα χρησιμοποιώντας την LDI, οι καταχωρητές από R0-R15 δεν μπορούν να χρησιμοποιηθούν με αυτό τον τρόπο.** Ο περιορισμός αυτός δεν είναι κάτι καλό, αλλά δεν μπορούσε να αποφευχθεί κατά τον σχεδιασμό του σετ εντολών των AVR.

Υπάρχει μια εξαίρεση σε αυτό και αυτή είναι όταν θέλουμε να μηδενίσουμε τα περιεχόμενα ενός καταχωρητή. Αυτό γίνεται με την εντολή CLR ως εξής:

```
CLR MyPreferredRegister
```

και μπορεί να χρησιμοποιηθεί για όλους τους καταχωρητές.

Εκτός από την εντολή LDI θα δούμε ότι ο περιορισμός αυτός ισχύει και για τις ακόλουθες εντολές:

- ANDI Rx,K; Λογική πράξη AND, μεταξύ του καταχωρητή Rx και μιας σταθερής τιμής K,
- CBR Rx,M; Καθαρισμός όλων των bits του καταχωρητή Rx που έχουν τιμή 1 μέσα στην τιμή (μάσκα) M,
- CPI Rx,K ; Σύγκριση του περιεχομένου του Rx με την σταθερή τιμή K,
- SBCI Rx,K; Αφαίρεση της σταθεράς K και της τρέχουσας τιμής του carry flag από το περιεχόμενο του Καταχωρητή Rx, αποθηκεύοντας το αποτέλεσμα στον Καταχωρητή Rx,
- SBR Rx,M; Ορισμός όλων των bits του καταχωρητή Rx σε 1, που έχουν τιμή 1 μέσα στην τιμή (μάσκα) M,
- SER Rx; Ορισμός όλων των bits του καταχωρητή Rx σε 1 (αντίστοιχο του LDI Rx,255),
- SUBI Rx,K; Αφαίρεση της σταθεράς K από το περιεχόμενο του Καταχωρητή Rx, αποθηκεύοντας το αποτέλεσμα στον Καταχωρητή Rx

Για όλες αυτές τις εντολές ο καταχωρητής θα πρέπει να είναι κάποιος μεταξύ των R16 έως και R31. Αν λοιπόν σχεδιάζουμε να χρησιμοποιήσουμε κάποια από αυτές τις εντολές θα πρέπει να έχουμε κατά νου ότι θα πρέπει να γίνει χρήση κάποιων από τους καταχωρητές από R16 έως και R31. Λόγω αυτού του περιορισμού κάποιος μπορεί να αντιληφθεί και τον λόγο για τον οποίο θα πρέπει να χρησιμοποιούμε κάποιον από αυτούς τους καταχωρητές. Επίσης λόγω αυτού του περιορισμού γίνεται φανερό και η χρησιμότητα της δυνατότητας να αναφερόμαστε στους καταχωρητές με ονόματα που μπορούμε να ορίζουμε εμείς με την εντολή .DEF.

4. Τρόποι Διευθυνσιοδότησης

Σαν τρόπους διευθυνσιοδότησης (Addressing modes) ορίζουμε τους τρόπους εκείνους με τους οποίους καθορίζεται η διεύθυνση των δεδομένων που παίρνουν μέρος στις εντολές. Στο παρακάτω σχήμα (Εικόνα 10) φαίνεται η δομή της μνήμης και των καταχωρητών του AVR AT90S8515. Για να διαχειριστούμε δεδομένα στις θέσεις μνήμης και τους καταχωρητές που φαίνονται στην Εικόνα 10 χρησιμοποιούμε τους λεγόμενους τρόπους διευθυνσιοδότησης.

Διακρίνουμε τέσσερις τρόπους διευθυνσιοδότησης:

1. **Άμεσος (Immediate).** Στον τρόπο αυτό τα δεδομένα βρίσκονται μέσα στην εντολή του προγράμματος αμέσως μετά το μνημονικό της εντολής. Π.χ.:

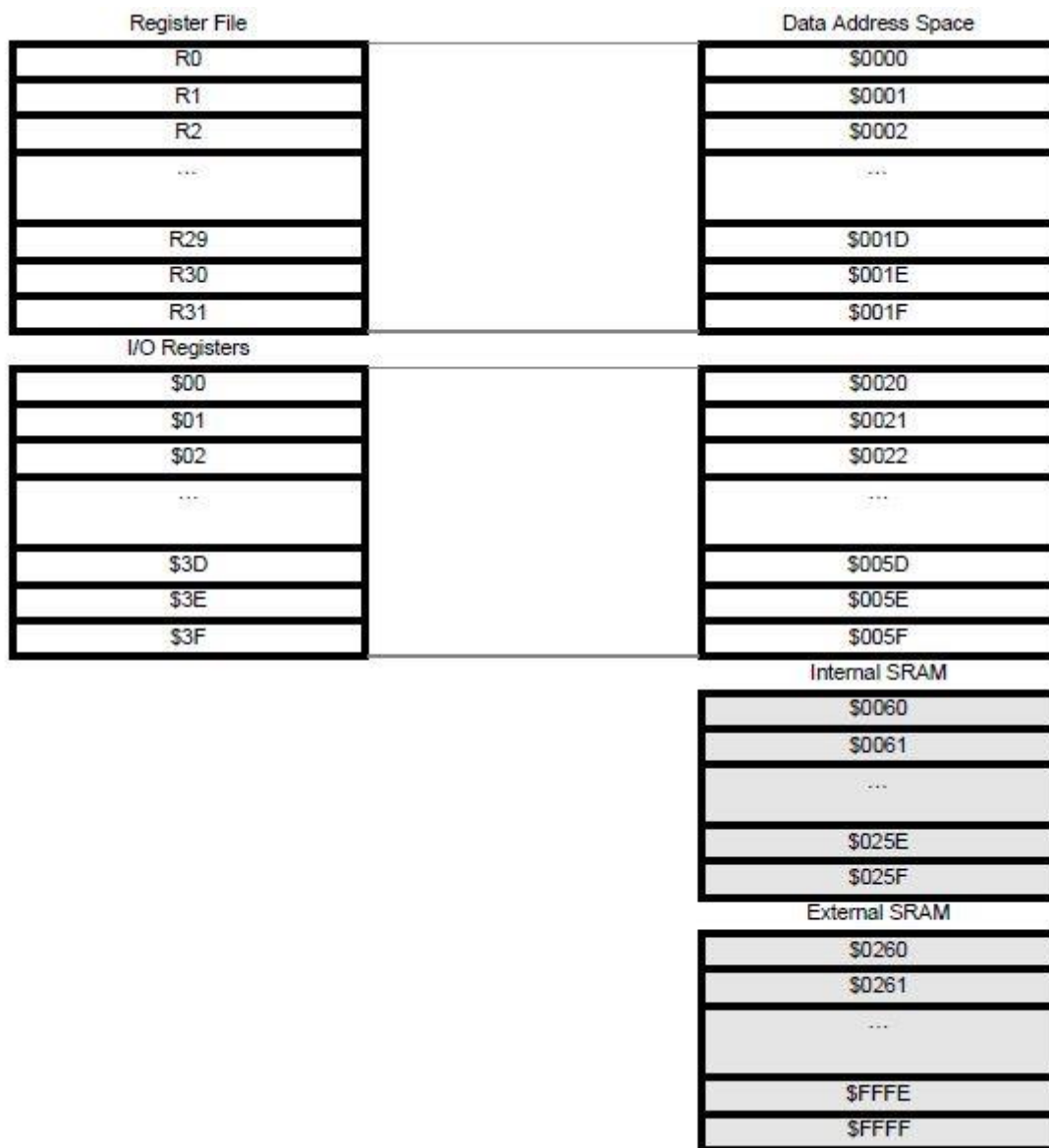
```
LDI R16, $30
```

Μετά την εκτέλεση της εντολής, ο R16 θα περιέχει τον αριθμό 30H.

2. **Καταχωρητή (Register).** Στη διευθυνσιοδότηση καταχωρητή, τόσο η πηγή όσο και ο προορισμός, είναι καταχωρητές της CPU. Π.χ.:

```
MOV R1, R16
```

Μετά την εκτέλεση της εντολής ο καταχωρητής R1 θα έχει περιεχόμενο αυτό του καταχωρητή R16 δηλαδή αντιγράφεται το περιεχόμενο του καταχωρητή R16 (καταχωρητής πηγής) στον καταχωρητή R1 (καταχωρητής προορισμού).



Εικόνα 10. Η δομή της μνήμης και των καταχωρητών του AVR AT90S8515

3. **Απ' ευθείας (Direct).** Η διεύθυνση μνήμης του δεδομένου αναγράφεται αμέσως μετά από το μνημονικό της εντολής. Π.χ.:

LDS R16, \$62

Μετά από την εκτέλεση της εντολής, ο καταχωρητής R16 θα έχει περιεχόμενο το περιεχόμενο της διεύθυνσης μνήμης SRAM 62H. Αν δηλαδή η θέση μνήμης 62H έχει περιεχόμενο 1AH τότε ο R16 θα έχει και αυτός περιεχόμενο την ίδια τιμή 1AH.

Ομοίως:

STS \$65, R17

Μετά από την εκτέλεση της εντολής, η θέση μνήμης 65H θα έχει περιεχόμενο το περιεχόμενο του R17.

4. **Έμμεσος καταχωρητής (register indirect).** Η διεύθυνση μνήμης του δεδομένου δίνεται από την τιμή κάποιου καταχωρητή δείκτη. Οι καταχωρητές δείκτες είναι οι X, Y και Z. Αυτοί οι καταχωρητές λειτουργούν σαν να είναι των 16-bit (2 byte) αλλά στην πραγματικότητα αποτελούνται από δύο καταχωρητές των 8-bit με:

- XH = R27 και XL = R26
- YH = R29 και YL = R28
- ZH = R31 και ZL = R30

Π.χ. :

LD R16, X

Αν ο X έχει την τιμή 63H μετά την εκτέλεση της εντολής ο R16 θα φορτωθεί με το περιεχόμενο της θέσης μνήμης 63H. Ομοίως:

ST X, R18

5. Παραδείγματα:

Παράδειγμα 1^ο - Γράψιμο/Διάβασμα από/προς μνήμη με άμεσο τρόπο

Χρησιμοποιώντας τον άμεσο τρόπο μεταφοράς δεδομένων από/προς την μνήμη, γράψτε ένα πρόγραμμα για να φορτώσετε άμεσα τους αριθμούς \$FF και \$AA σε δύο καταχωρητές και στην συνέχεια να τους μεταφέρετε στις θέσεις μνήμης \$64 και \$65 αντίστοιχα. Σαν τελικό βήμα να αποθηκεύσετε το περιεχόμενο της διεύθυνσης μνήμης \$64 στον καταχωρητή R1. Τρέξτε το πρόγραμμα και διαπιστώστε την ορθή μεταφορά όλων των δεδομένων (μνήμης και καταχωρητών).

1ος Τρόπος

```
.include "8515def.inc"
    LDI R24, $FF
    LDI R25, $AA
    STS $64, R24      ; Από καταχωρητή σε διεύθυνση μνήμης
    STS $65, R25
    LDS R1, $64       ; Από διεύθυνση μνήμης σε καταχωρητή
```

2ος Τρόπος – Χρήση αλλαγής ονόματος καταχωρητών

```
.include "8515def.inc"
.def target = R1
.def buffer1 = R24
.def buffer2 = R25
    LDI buffer1, $FF
    LDI buffer2, $AA
    STS $64, buffer1 ; Από καταχωρητή σε διεύθυνση μνήμης
    STS $65, buffer2
    LDS target, $64  ; Από διεύθυνση μνήμης σε καταχωρητή
```

Παράδειγμα 2ο - Χρήση καταχωρητών διεύθυνσης

Χρησιμοποιώντας την άμεση διευθυνσιοδότηση να φορτώσετε τον καταχωρητή R20 με τον αριθμό \$FE και έπειτα με τον έμμεσο τρόπο να το μεταφέρετε στη θέση μνήμης \$1E6.

```
.include "8515def.inc"
.org 10
    LDI R20, $FE
    LDI XL, $E6      ; Αρχικοποίηση του LowByte του ζεύγους καταχωρητών X
    LDI XH, $01      ; Αρχικοποίηση του HighByte του ζεύγους καταχωρητών X
    ST X, R20        ; Μεταφορά του περιεχομένου του καταχωρητή R20 στη
                     ; διεύθυνση μνήμης που δείχνει ο X
```

6. Ασκήσεις 2^{ης} Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

1. Γράψτε πρόγραμμα σε γλώσσα assembly που να θέτει στους αντίστοιχους καταχωρητές τα παρακάτω δεδομένα:

R16 = 200D
R17 = 45H
R1 = 23H
R2 = 00110101b
Y = 8040H

2. Γράψτε πρόγραμμα που να βάζει στη θέση μνήμης 60H τον αριθμό 89D με απευθείας διευθυνσιοδότηση και στις θέσεις μνήμης 61H και 62H τον αριθμό D9H με έμμεση διευθυνσιοδότηση.
3. Γράψτε πρόγραμμα που να αντιγράφει τα περιεχόμενα του καταχωρητή R3 στους καταχωρητές R1, R2 και R4. Να κάνετε χρήση της δυνατότητας του AVR αλλαγής ονόματος καταχωρητών.

Εργαστήριο 3^ο - Πρόσθεση και Αφαίρεση δεκαεξαδικών (8-bit/16-bit) αριθμών

Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Εντολές Πρόσθεσης
3. Εντολές Αφαίρεσης
4. Καταχωρητής Κατάστασης – Status Register (SREG)
5. Έλεγχος SREG – Μνήμης & Καταχωρητών
6. Παραδείγματα
 - Παράδειγμα 1^ο – Πρόσθεση 8bit αριθμών
 - Παράδειγμα 2^ο – Πρόσθεση 16bit αριθμών
7. Ασκήσεις 3^{ης} Εργαστηριακής Άσκησης

1. Εισαγωγή

Οι αριθμητικές πράξεις όπως πρόσθεση και αφαίρεση είναι θεμελιώδης σε πολλά υπολογιστικά συστήματα. Οι περισσότερες γλώσσες προγραμματισμού υποστηρίζουν διάφορους αριθμητικούς τύπους δεδομένων τα οποία χρησιμοποιούνται για την πραγματοποίηση αριθμητικών υπολογισμών.

Για την εκτέλεση αριθμητικών και λογικών πράξεων οι μικροελεγκτές AVR διαθέτουν ένα ευρύ φάσμα εντολών. Οι εντολές αυτές έχουν ένα ή δύο ορίσματα (operands) τα οποία είναι πάντοτε οι καταχωρητές γενικού σκοπού R0 έως R31. **Δεν υπάρχει δυνατότητα να γίνει απευθείας πράξη μεταξύ του περιεχομένου δύο θέσεων μνήμης.** Επομένως για να γίνει μία αριθμητική ή λογική πράξη πρέπει πρώτα να μεταφερθούν τα ορίσματα από τις θέσεις μνήμης που είναι αποθηκευμένα να γίνει η πράξη και το αποτέλεσμα να γραφεί σε μία άλλη θέση μνήμης.

Στην άσκηση αυτή για λόγους απλότητας χρησιμοποιείται για αποθήκευση των δεδομένων και των αποτελεσμάτων μόνο η εσωτερική SRAM. 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.

Δεκαεξαδική (hex) αναπαράσταση του Byte	Δυαδική αναπαράσταση του Byte
00	0000 0000
01	0000 0001
02	0000 0010
03	0000 0011
04	0000 0100
05	0000 0101
06	0000 0110
07	0000 0111
08	0000 1000
09	0000 1001
0A	0000 1010
0B	0000 1011
0C	0000 1100
0D	0000 1101
0E	0000 1110
0F	0000 1111

2. Εντολές Πρόσθεσης

ADD Rd,Rr : $Rd \leftarrow Rd + Rr$

Περιγραφή: Πρόσθεση 2 καταχωρητών **χωρίς χρήση κρατουμένου** και αποθήκευση στον καταχωρητή προορισμού Rd.

ADC Rd,Rr : $Rd \leftarrow Rd + Rr + C$

Περιγραφή: Πρόσθεση 2 καταχωρητών **με χρήση κρατουμένου** και αποθήκευση στον καταχωρητή προορισμού Rd.

3. Εντολές Αφαίρεσης

SUB Rd,Rr : $Rd \leftarrow Rd - Rr$

Περιγραφή: Αφαίρεση 2 καταχωρητών χωρίς χρήση κρατούμενου και αποθήκευση στον καταχωρητή προορισμού Rd

SUBI Rd,K : $Rd \leftarrow Rd - K$

Περιγραφή: Αφαίρεση μιας τιμής χωρίς χρήση κρατούμενου από έναν καταχωρητή και αποθήκευση στον καταχωρητή

SBCI Rd,K : $Rd \leftarrow Rd - K - C$

Περιγραφή: Αφαίρεση μιας τιμής με χρήση κρατούμενου από έναν καταχωρητή και αποθήκευση στον καταχωρητή

SBC Rd,Rr : $Rd \leftarrow Rd - Rr - C$

Περιγραφή: Αφαίρεση 2 καταχωρητών με χρήση κρατούμενου και αποθήκευση στον καταχωρητή προορισμού Rd.

4. Καταχωρητής Κατάστασης - Status REGISTER (SREG).

Στις εντολές πρόσθεσης και αφαίρεσης συμμετέχει και το **κρατούμενο (C flag)** το οποίο είναι ένα bit του καταχωρητή κατάστασης **SREG (Status REGISTER)**. Το κρατούμενο, κάθε φορά που εκτελείται μια αριθμητική εντολή, ανανεώνεται αυτόματα. Έτσι αν το αποτέλεσμα μιας πράξης βγάλει κρατούμενο τότε θα έχουμε $C = "1"$ ενώ αν δε βγάλει τότε θα έχουμε $C = "0"$, ανεξάρτητα από την τιμή που είχε πριν την εκτέλεση της πράξης. Το κρατούμενο μπορούμε να το κάνουμε και κατ' απαίτηση "1" ή "0" χρησιμοποιώντας τις εντολές SEC και CLC αντίστοιχα.

Bit	7	6	5	4	3	2	1	0	
\$3F (\$5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Εικόνα 11. Καταχωρητής Κατάστασης - Status REGISTER (SREG)

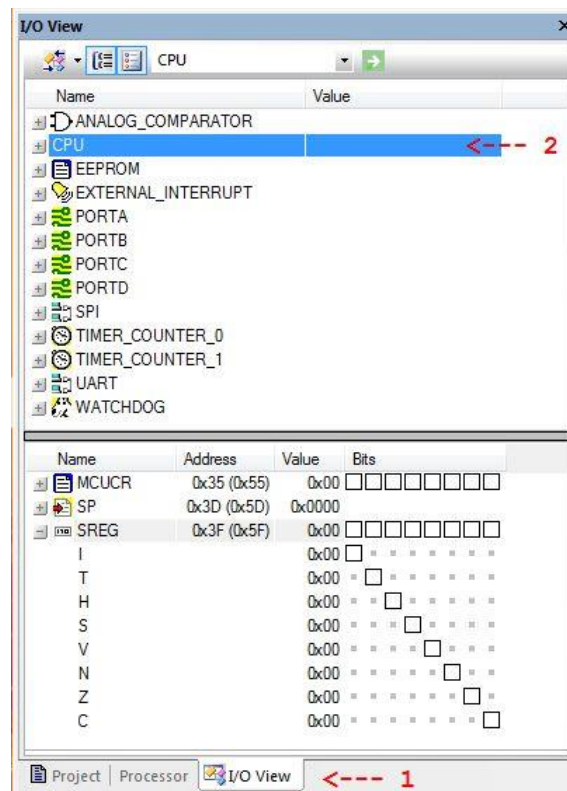
Ο καταχωρητής κατάστασης (**SREG**) είναι ένας 8-bit καταχωρητής που κάθε φορά που εκτελούνται αριθμητικές πράξεις, ανανεώνεται (refresh) με σημαίες που έχουν να κάνουν με το αποτέλεσμα της πράξης.

- **I (Global Interrupt Enable):** Επιτρέπει τις διακοπές (Interrupts) στον επεξεργαστή.
- **T (Target):** Οι εντολές αντιγραφής ενός Bit από μια θέση μνήμης έχουν σαν θέση αποθήκευσης το συγκεκριμένο πεδίο.
- **H (Half Carry):** Κρατούμενο που προκύπτει από τα τέσσερα λιγότερο σημαντικά ψηφία.
- **S (Sign):** Δείκτης πρόσημου, αποτέλεσμα του «αποκλειστικού OR» EXOR μεταξύ των πεδίων V και N.
- **V (Overflow):** Δείχνει υπερχείλιση σε πράξεις συμπληρώματος ως προς 2
- **N (Negative):** Δηλώνει ότι ο αριθμός που προκύπτει από μια αριθμητική ή λογική πράξη είναι αρνητικός
- **Z (Zero):** Δηλώνει ότι το αποτέλεσμα μιας πράξης είναι 0
- **C (Carry):** Δείκτης που δηλώνει ότι προέκυψε κρατούμενο από μια αριθμητική ή λογική πράξη.

Στην πράξη οι περισσότερο χρησιμοποιούμενες - λαμβανόμενες υπ όψιν σημαίες είναι οι: **I , N , Z , C** ενώ οι: **T , H , S , V** έχουν δευτερεύουσα σημασία.

5. Έλεγχος SREG - Μνήμης & Καταχωρητών

Ο έλεγχος του καταχωρητή κατάστασης SREG γίνεται από την περιοχή **"I/O View"** του περιβάλλοντος εργασίας του AVR Studio (Εικόνα 12). Στον ίδιο χώρο φαίνονται συνοπτικά όλες οι τιμές των bit του SREG.



Εικόνα 12. Έλεγχος του καταχωρητή κατάστασης SREG

Ο έλεγχος των καταχωρητών γενικής χρήσης 0-31 γίνεται επιλέγοντας το εικονίδιο **“Toggle Register Window”** ή **“Alt+0”** (Εικόνα 13). Στο παράθυρο που ενεργοποιείται (Εικόνα 14), φαίνονται οι τιμές των καταχωρητών γενικής χρήσης 0-31.



Εικόνα 13. Εικονίδιο για την εμφάνιση του παραθύρου ελέγχου των καταχωρητών γενικής χρήσης 0-31.



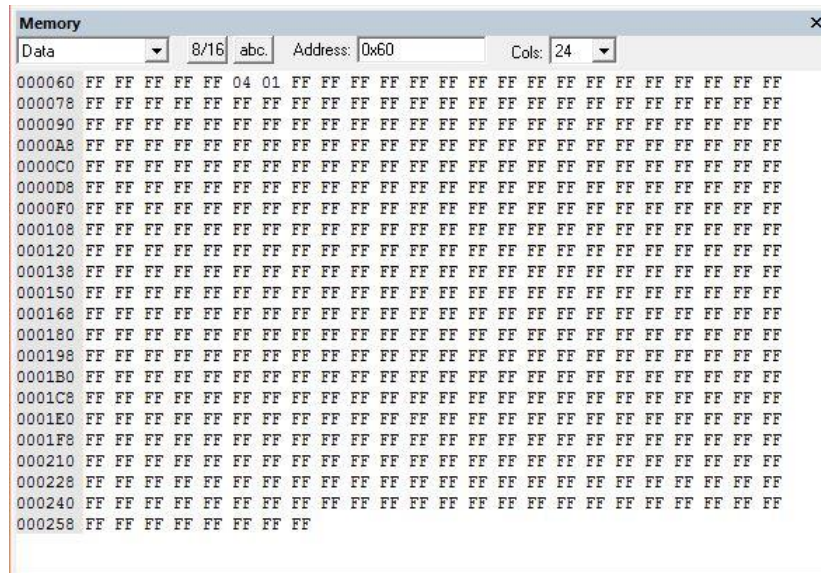
Εικόνα 14. Έλεγχος των καταχωρητών γενικής χρήσης 0-31.

Ο έλεγχος της μνήμης SRAM του μικροεπεξεργαστή, γίνεται επιλέγοντας το εικονίδιο **“Toggle Memory Window”** ή **“Alt+4”** (Εικόνα 15). Στο παράθυρο που ενεργοποιείτε (Εικόνα 16), φαίνονται οι τιμές της μνήμης SRAM.



υ αποθήκευση στην θέση μνήμης \$65

Εικόνα 15. Εικονίδιο για την εμφάνιση του παραθύρου ελέγχου της μνήμης SRAM.



Εικόνα 16. Έλεγχος της μνήμης SRAM

6. Παραδείγματα:

Παράδειγμα 1^ο - Πρόσθεση 8bit αριθμών

Το αποτέλεσμα μιας πρόσθεσης 2 αριθμών 8bit (1 byte) είναι πιθανόν να είναι 2 bytes. Έτσι όταν προσθέτουμε δυο αριθμούς του ενός byte θα το κάνουμε σύμφωνα με το παρακάτω παράδειγμα:

Γράψτε ένα πρόγραμμα για να προσθέσετε τους αριθμούς \$EE και \$B5. Τοποθετήστε το άθροισμά τους στην θέση μνήμης \$65 (Low Byte) και \$64 (High Byte). Υπολογίστε το άθροισμα μόνοι σας και δείτε στον Status Register (SREG) αν δημιουργήθηκε κρατούμενο. Αιτιολογήστε την απάντησή σας.

```
.include "8515def.inc"
; Πρόσθεση των δύο 8bit αριθμών και αποθήκευση στην θέση μνήμης $65
LDI R24,$EE
LDI R25,$B5
ADD R24,R25
STS $65,R24
; Ενδεχομένως να έχουμε κρατούμενο μετά την πρόσθεση των 4 high Byte των
; προσθετέων για να καταφέρουμε να αποθηκεύσουμε το κρατούμενο που είναι
; σε ένα bit θα πρέπει να προσθέσουμε το κρατούμενο με δύο καταχωρητές που
; έχουν περιεχόμενο 0.
; Δηλαδή να γίνει η πράξη 0 + 0 + C έτσι ώστε αν η προηγούμενη πράξη έχει
; δώσει κρατούμενο να προκύψει ο αριθμός 01
; διαφορετικά ο 00.
; Μετατροπή του τυχόν κρατούμενου σε Byte και μεταφορά στον καταχωρητή R26
LDI R26,$00
LDI R27,$00
ADC R26,R27
; Αποθήκευση του κρατούμενου στη θέση μνήμης $64
STS $64,R26
```

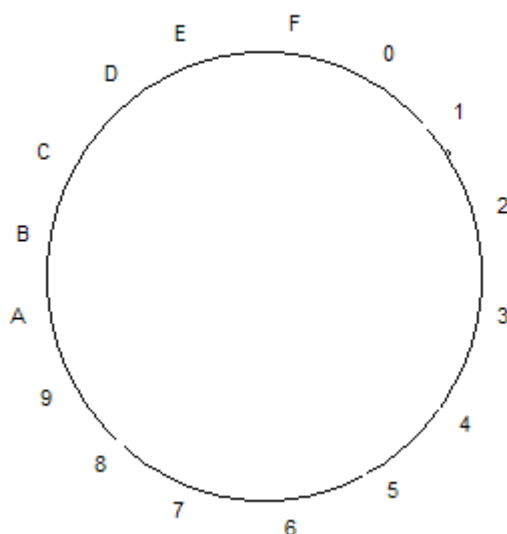
Παράδειγμα 2^ο - Πρόσθεση 16bit αριθμών

Το αποτέλεσμα μιας πρόσθεσης 2 αριθμών 16bit (2 byte) είναι πιθανόν να είναι 3 bytes. Έτσι όταν προσθέτουμε δυο αριθμούς των δυο bytes θα το κάνουμε σύμφωνα με το παρακάτω παράδειγμα:

Γράψτε ένα πρόγραμμα για να αποθηκεύσετε τον 16-bit αριθμό F1A2H στις θέσεις μνήμης 130H, 131H. Στη συνέχεια να προσθέσετε στον αριθμό που αποθηκεύσατε τον 16-bit αριθμό 1284H και τοποθετήστε το αποτέλεσμα στις θέσεις μνήμης 140H, 141H και το κρατούμενο (αν υπάρχει) στην διεύθυνση 142H. Υπολογίστε το άθροισμα μόνοι σας και δείτε στον Status Register (SREG) αν δημιουργήθηκε κρατούμενο. Αιτιολογείστε την απάντησή σας.

```
.include "8515def.inc"
; Αποθήκευση του αριθμού F1A2H στις θέσεις 130H και 131H
    LDI R19,$A2
    LDI R18,$F1
    STS $131,R19
    STS $130,R18
; Πρόσθεση των δύο λιγότερο σημαντικών Bytes των δύο αριθμών με ADD για να
; αγνοηθεί τυχόν προηγούμενο κρατούμενο και αποθήκευση του αποτελέσματος
; στη θέση $140
    LDS R20,$131
    LDI R21,$84
    ADD R20,R21
    STS $142,R20
; Πρόσθεση των δύο περισσότερο σημαντικών Bytes των δύο αριθμών με ADC για να
; ληφθεί υπόψη τυχόν κρατούμενο από την προηγούμενη πρόσθεση και
; αποθήκευση του αποτελέσματος στη θέση $141
    LDS R20,$130
    LDI R21,$12
    ADC R20,R21
    STS $141,R20
; Μετατροπή του τυχόν τελικού κρατούμενου σε Byte και αποθήκευση του στη
; θέση μνήμης $142
    LDI R24,$00
    LDI R25,$00
    ADC R24,R25
    STS $140,R24
```

Προσοχή: Το παρακάτω σχήμα περιγράφει έναν μνημονικό κανόνα για την πρόσθεση και αφαίρεση στο 16δικό σύστημα.



7. Ασκήσεις 3ης Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

1. Γράψτε ένα πρόγραμμα για να αφαιρεί από **\$0A** τον αριθμό **\$5F**. Τοποθετήστε το αποτέλεσμα τους στην θέση μνήμης **\$67**. Αν υπάρχει δανεικό από την αφαίρεση να αποθηκευτεί στην θέση μνήμης **\$68**. Υπολογίστε το αποτέλεσμα μόνοι σας και δείτε στον SREG αν δημιουργήθηκε κρατούμενο. Αιτιολογείστε την απάντησή σας.
2. Γράψτε ένα πρόγραμμα με το οποίο να προσθέτετε τους αριθμούς **3426h** και **AB3Ch**. Στη συνέχεια το αποτέλεσμα της πρόσθεσης των **Low Byte** των παραπάνω 16bit αριθμών να αφαιρείται από τον αριθμό **92h**. Να γίνει επαλήθευση των πράξεων στο χαρτί σας. Δείτε στον SREG αν δημιουργήθηκε κρατούμενο. Αιτιολογείστε την απάντησή σας.

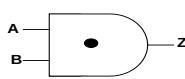
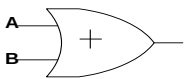
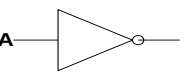
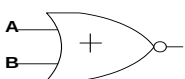
Εργαστήριο 4^ο - Λογικές Πράξεις

Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Εντολές λογικών πράξεων
3. Παραδείγματα
 - Παράδειγμα 1ο - Γράψιμο/Διάβασμα από/προς μνήμη με άμεσο τρόπο
 - Παράδειγμα 2ο - Χρήση καταχωρητών διεύθυνσης
4. Ασκήσεις 4ης Εργαστηριακής Άσκησης

1. Εισαγωγή

Όπως είδαμε και στο προηγούμενο εργαστήριο (3) έτσι και εδώ οι μικροελεγκτές AVR διαθέτουν ένα μεγάλο ρεπερτόριο εντολών και για την εκτέλεση λογικών πράξεων. Στην παρακάτω εικόνα υπάρχουν οι βασικές λογικές πράξεις με τα σύμβολα και τους πίνακες αληθείας τους (Εικόνα 17).

Ονομασία	Σύμβολο	Σχέση	Πίνακας αληθείας		
			A	B	Z
AND		$Z = A \bullet B$	0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		$Z = A + B$	0	0	0
			0	1	1
			1	0	1
			1	1	1
NOT		$Z = \overline{A}$	0	1	
			1	0	
NAND		$Z = \overline{A \bullet B}$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$Z = \overline{A + B}$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XOR		$Z = A \oplus B$	0	0	0
			0	1	1
			1	0	1
			1	1	0
XNOR		$Z = \overline{A \oplus B}$	0	0	1
			0	1	0
			1	0	0
			1	1	1

Εικόνα 17. Βασικές Λογικές Πράξεις, τα σύμβολα και οι πίνακες αληθείας τους

Η γενική μορφή σύνταξης των εντολών για την εκτέλεση λογικών πράξεων είναι η εξής:

ONOMA_ENTOLΗΣ Ορσμ1 (ή Ορσμ1, Ορσμ2)

Τα ορίσματα είναι πάντοτε οι καταχωρητές γενικού σκοπού R0 έως R31. **Δεν υπάρχει δυνατότητα να γίνει απευθείας πράξη μεταξύ του περιεχομένου δύο θέσεων μνήμης.** Επομένως για να γίνει μία λογική πράξη πρέπει πρώτα να μεταφερθούν τα ορίσματα από τις θέσεις μνήμης που είναι αποθηκευμένα να γίνει η πράξη και το αποτέλεσμα να γραφεί σε μία άλλη θέση μνήμης.

2. Εντολές λογικών πράξεων

Με τις εντολές μπορούν να γίνουν όλες οι βασικές λογικές πράξεις είτε σε επίπεδο byte, ή σε επίπεδο bit. Οι εντολές αυτές είναι:

AND Rd,Rr $Rd \leftarrow Rd \bullet Rr$

($0 \leq d \leq 31, 0 \leq r \leq 31$)

Περιγραφή: Εκτελεί ένα λογικό AND ανάμεσα στα περιεχόμενα του καταχωρητή Rd και του καταχωρητή Rr και τοποθετεί το αποτέλεσμα στο καταχωρητή προορισμού Rd.

$$\mathbf{ANDI\ Rd, K\ Rd} \leftarrow \mathbf{Rd} \bullet \mathbf{K}$$

$$(16 \leq d \leq 31, 0 \leq K \leq 255)$$

Περιγραφή: Εκτελεί ένα λογικό AND ανάμεσα στα περιεχόμενα του καταχωρητή Rd και της σταθεράς και τοποθετεί το αποτέλεσμα στον καταχωρητή προορισμού Rd.

Αξίζει να σημειωθεί ότι, με τη χρήση της εντολής AND μπορούμε να μηδενίσουμε συγκεκριμένα bit σε ένα καταχωρητή χωρίς να επηρεαστούν τα υπόλοιπα.

$$\mathbf{OR\ Rd, Rr\ Rd} \leftarrow \mathbf{Rd} \text{ OR } \mathbf{Rr}$$

$$(0 \leq d \leq 31, 0 \leq r \leq 31)$$

Περιγραφή: Εκτελεί ένα λογικό OR ανάμεσα στα περιεχόμενα των καταχωρητών Rd και Rr και τοποθετεί το αποτέλεσμα στον καταχωρητή προορισμού Rd.

$$\mathbf{ORI\ Rd, K\ Rd} \leftarrow \mathbf{Rd} \text{ OR } \mathbf{K}$$

$$(16 \leq d \leq 31, 0 \leq K \leq 255)$$

Περιγραφή: Εκτελεί ένα λογικό OR ανάμεσα στα περιεχόμενα του καταχωρητή Rd και της σταθεράς και τοποθετεί το αποτέλεσμα στον καταχωρητή προορισμού Rd.

Αξίζει να σημειωθεί ότι, με τη χρήση της εντολής OR μπορούμε να θέσουμε σε λογικό "1" συγκεκριμένα bit σε ένα καταχωρητή χωρίς να επηρεαστούν τα υπόλοιπα.

$$\mathbf{EOR\ Rd, Rr\ Rd} \leftarrow \mathbf{Rd} \oplus \mathbf{Rr}$$

$$(0 \leq d \leq 31, 0 \leq r \leq 31)$$

Περιγραφή: Εκτελεί το λογικό EOR ανάμεσα στα περιεχόμενα των καταχωρητών Rd και Rr και τοποθετεί το αποτέλεσμα στον καταχωρητή προορισμού Rd.

Αξίζει να σημειωθεί ότι, με τη χρήση της εντολής EOR μπορούμε να συμπληρώσουμε (αντιστρέψουμε) συγκεκριμένα bit σε ένα καταχωρητή χωρίς να επηρεαστούν τα υπόλοιπα.

$$\mathbf{COM\ Rd\ Rd} \leftarrow \mathbf{\$FF} - \mathbf{Rd}$$

$$(0 \leq d \leq 31)$$

Περιγραφή: Αυτή η εντολή επιστρέφει στον καταχωρητή Rd το συμπληρωματικό ως προς 1 του περιεχομένου του.

$$\mathbf{NEG\ Rd\ Rd} \leftarrow \mathbf{\$00} - \mathbf{Rd}$$

$$(0 \leq d \leq 31)$$

Περιγραφή: Αντικαθιστά το περιεχόμενο του καταχωρητή Rd με τον συμπληρωματικό του ως προς το 0.

3. Παραδείγματα:

Παράδειγμα 1^ο - Λογικές πράξεις στον AVR (AND, OR)

1. Γράψτε πρόγραμμα, το οποίο να τοποθετεί στις θέσεις μνήμης \$60, \$61 και \$62 τους αριθμούς EEh, AAh και 11h. Κατόπιν να εκτελούνται οι εξής λογικές πράξεις με τη σειρά:

- α) κάντε αντιστροφή του περιεχομένου της \$61 (χρήση της EOR με το FFh) και τοποθετείστε τον αντίστροφο στην \$63.
- β) κάντε λογικό AND μέσω της αντίστοιχης εντολής, ανάμεσα στο περιεχόμενο της \$60 και το περιεχόμενο της \$62. Τοποθετείστε το αποτέλεσμα στην \$64.
- γ) κάντε λογικό OR, ανάμεσα στο περιεχόμενο της \$61 και το περιεχόμενο της \$62. Τοποθετείστε το αποτέλεσμα στην \$65. Διαπιστώστε τα αποτελέσματα των λογικών πράξεων στις θέσεις μνήμης από \$60 έως \$65.

Απάντηση:

```
.include "8515def.inc"
; Τοποθέτηση των αριθμών στις διευθύνσεις μνήμης
LDI R16,$EE
STS $60,R16
LDI R16,$AA
STS $61,R16
LDI R16,$11
STS $62,R16
; Απάντηση Ερωτήματος α
LDS R18,$61
LDI R19,$FF
EOR R18,R19
STS $63,R18
; Απάντηση Ερωτήματος β
LDS R18,$60
LDS R19,$62
AND R18,R19
STS $64,R18
; Απάντηση Ερωτήματος γ
LDS R18,$61
LDS R19,$62
OR R18,R19
STS $65,R18
```

Παράδειγμα 2^ο - Αλλαγή τιμής συγκεκριμένων bit του περιεχομένου μιας θέσης μνήμης

Στη θέση μνήμης \$7B να γίνουν οι παρακάτω λογικές πράξεις, χωρίς να επηρεαστούν τα bit 1 και 5:

- α) να μηδενιστούν τα bit 0 και 7
- β) να τεθούν σε λογικό "1" τα bit 2 και 4
- γ) να αντιστραφούν τα bit 3 και 6

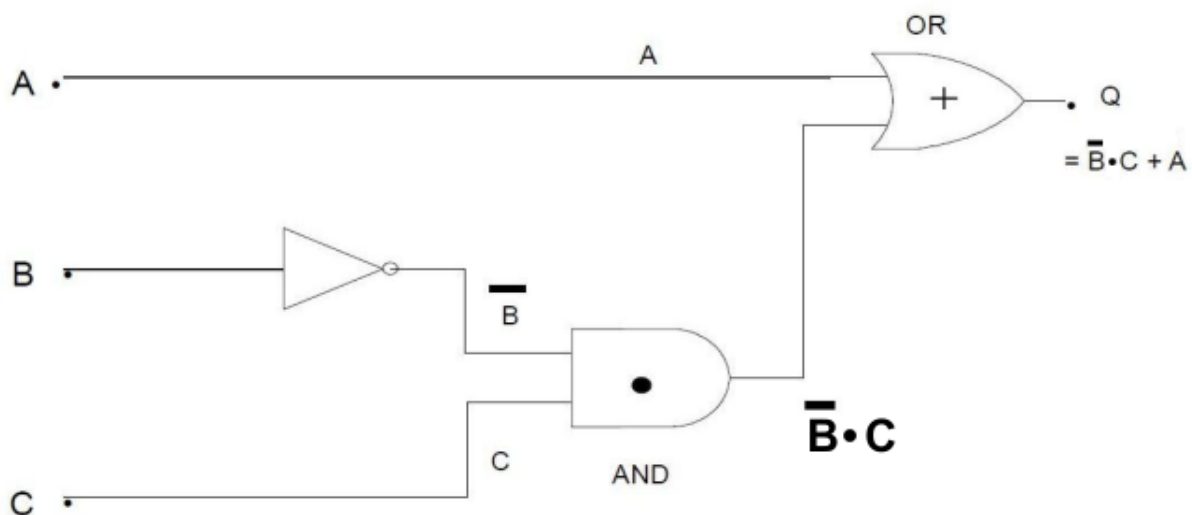
Απάντηση:

```
.include "8515def.inc"
LDS R20,$7B
; Μηδενισμός των bit 0 και 7
ANDI R20,0b01111110
; Τα bit 2 και 4 σε λογικό "1"
ORI R20,0b00010100
; Αντιστροφή των bit 3 και 6
LDI R21,0b01001000
EOR R20,R21
; Αποθήκευση νέου περιεχομένου της $7B
STS $7B,R20
```

4. Ασκήσεις 4^{ης} Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

1. Θεωρήστε το παρακάτω κύκλωμα όπου οι είσοδοι **A**, **B** και **C** αντιστοιχούν στα περιεχόμενα των μνημών **\$60**, **\$61** και **\$62** αντίστοιχα. Γράψτε ένα πρόγραμμα, το οποίο να τοποθετεί στις θέσεις μνήμης **\$60**, **\$61** και **\$62** τους αριθμούς **3Ch**, **15h** και **FAh**. Υλοποιήστε στον AVR την λογική πρόταση του παρακάτω σχήματος. Το τελικό αποτέλεσμα να αποθηκευτεί στην θέση μνήμης **\$65**.



2. Γράψτε πρόγραμμα το οποίο με τη χρήση λογικών εντολών να αντιστρέφει τα τέσσερα λιγότερο σημαντικά bit της θέσης μνήμης **\$85** και να μηδενίζει τα υπόλοιπα. Στην συνέχεια να θέτει σε λογικό **"1"** όλα τα bit της θέσης μνήμης **\$95**.

Εργαστήριο 5^ο - Διαγράμματα Ροής – Διακλαδώσεις – Λήψη αποφάσεων

Περίγραμμα Εργαστηριακής Άσκησης

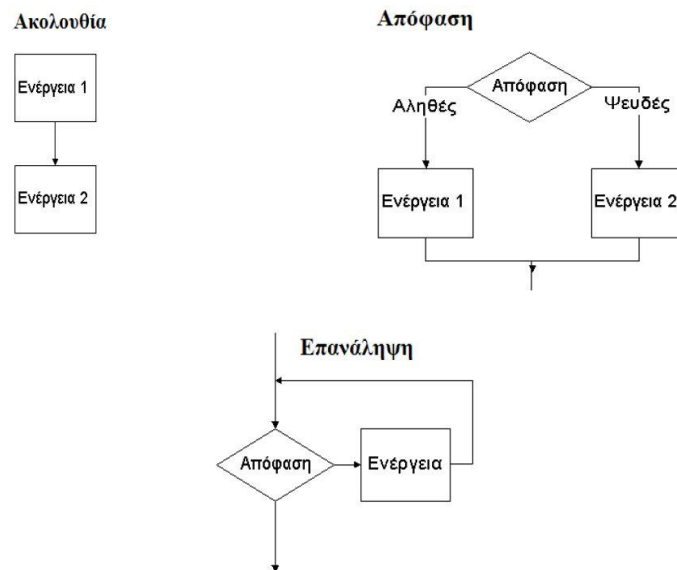
1. Εισαγωγή
2. Διαγράμματα Ροής
3. Λήψη αποφάσεων
 - α) Διακλάδωση χωρίς συνθήκη
 - β) Διακλάδωση υπό συνθήκη
 - με έλεγχο byte
 - με έλεγχο ενός bit
4. Παραδείγματα
 - Παράδειγμα 1^ο – Σύγκριση 2 αριθμών & αποθήκευση σε συγκεκριμένες θέσεις SRAM
 - Παράδειγμα 2^ο – Αντιγραφή αριθμού σε συνεχόμενες θέσεις μνήμης
5. Ασκήσεις 5ης Εργαστηριακής Άσκησης

1. Εισαγωγή

Η επίλυση ενός προβλήματος πολλές φορές ανάγεται στην επίλυση άλλων απλούστερων προβλημάτων, τα οποία με τη σειρά τους μπορούν να ανάγονται σε επίσης απλούστερα προβλήματα. Η τεχνική αυτή είναι γνωστή ως τεχνική του τμηματικού προγραμματισμού και αποτελεί ένα από τα βασικότερα χαρακτηριστικά του δομημένου προγραμματισμού.

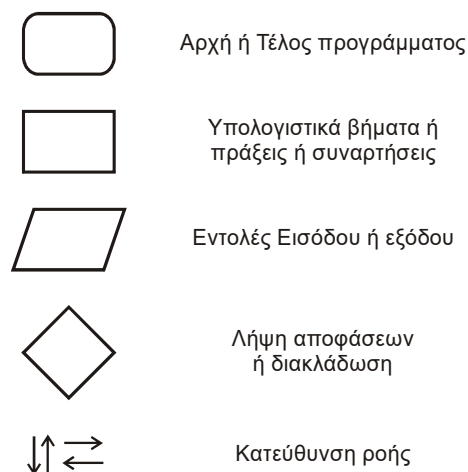
2. Διαγράμματα Ροής

Το διάγραμμα ροής (*flow chart*) μας επιτρέπει να παραστήσουμε με λεπτομέρεια τη ροή του προγράμματος. Όταν προγραμματίζουμε δομημένα πρέπει τα τμήματα του προγράμματος να απαρτίζονται από τα παρακάτω στοιχεία (Εικόνα 18), χωρίς ενδιάμεσες ενώσεις.



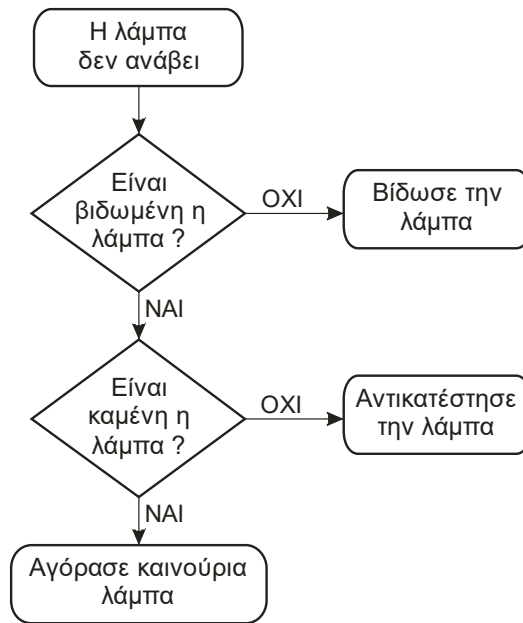
Εικόνα 18. Βασικά στοιχεία των λογικών διαγραμμάτων

Χρησιμοποιώντας τα βασικά σχήματα των λογικών διαγραμμάτων (Εικόνα 19), μπορούμε να περιγράψουμε κάθε λογική διαδικασία.



Εικόνα 19. Βασικά σχήματα λογικών διαγραμμάτων

Στο παράδειγμα που ακολουθεί δίνεται η διαδικασία για τον έλεγχο μιας λάμπας (Εικόνα 20).



Εικόνα 20. Έλεγχος λάμπας

Λήψη αποφάσεων

Οι εντολές διακλάδωσης ή μεταφοράς ελέγχου χρησιμεύουν στη δημιουργία διακλαδώσεων μέσα σ' ένα πρόγραμμα, αν θέλουμε να παρακάμψουμε μια σειρά από εντολές όταν ικανοποιείται ή όταν δεν ικανοποιείται κάποια συνθήκη. Επίσης χρησιμεύουν στις διαδικασίες ανακυκλώσεων και επανεκτελέσεων τμημάτων του προγράμματος. Στις εντολές διακλάδωσης ανήκουν και οι εντολές κλήσης και επιστροφής από υπορουτίνα. Μερικές από τις εντολές αυτές δεν έχουν όρισμα. Όσες έχουν όρισμα, έχουν σαν όρισμα την ετικέτα της εντολής, στην οποία θα γίνει η διακλάδωση. Οι εντολές διακλάδωσης αλλάζουν την ροή του προγράμματος αντικαθιστώντας τα περιεχόμενα του PC (Program Counter) με την διεύθυνση της εντολής που θα γίνει η διακλάδωση.

Οι εντολές διακλάδωσης μπορούν να χωριστούν σε δύο κατηγορίες:

- τις εντολές διακλάδωσης χωρίς συνθήκη και
- τις εντολές διακλάδωσης υπό συνθήκη.

3. α) Διακλάδωση χωρίς συνθήκη

Οι εντολές με τις οποίες μπορεί να γίνει η “άνευ όρων” διακλάδωση. Οι παρακάτω εντολές μας παρέχουν τη δυνατότητα να εκτελέσουμε άλμα σε επιθυμητή διεύθυνση, να καλέσουμε μια ρουτίνα και να επιστρέψουμε από αυτήν.

RJMP k : $PC \leftarrow PC + k + 1$

Περιγραφή: Σχετικό άλμα σε μια διεύθυνση της μνήμης προγράμματος μεταξύ $PC - 2K + 1$ και $PC + 2K$. Το k είναι μία ετικέτα (label) στην αριστερή στήλη του προγράμματος. Σαν ετικέτα μπορούμε να βάλουμε ότι θέλουμε αρκεί **να είναι σε λατινικούς χαρακτήρες, να μην είναι το όνομα κάποιας εντολής και να μην έχει κενά**.

IJMP

Περιγραφή: Άλμα στην διεύθυνση μνήμης flash που ορίζεται από τον δείκτη Z ($Z = R31, R30$). Χρησιμοποιείται σπάνια σε περιπτώσεις που θέλουμε να κάνουμε μεγάλο άλμα μέσα στη μνήμη.

NOP

Περιγραφή: Η εντολή αυτή δεν κάνει τίποτα. Απλά καθυστερεί τον επεξεργαστή για ένα κύκλο μηχανής

4. β) Διακλάδωση υπό συνθήκη

• Διακλαδώσεις που γίνεται έλεγχος byte:

Οι εντολές με τις οποίες γίνεται “υπό συνθήκη” διακλάδωση μπορούν να ελέγχουν είτε ένα bit είτε ένα byte αν ικανοποιεί κάποια συνθήκη που όταν αληθεύει θα λάβει χώρα το άλμα. Στην ουσία εξετάζουν μία από τις επτά σημαίες του καταχωρητή κατάστασης (**SREG**) για να αποφασίσουν αν πρέπει να γίνει το άλμα ή όχι. Οι σημαίες αυτές επηρεάζονται από τις πράξεις και από κάποιες άλλες εντολές που θα δούμε παρακάτω. Πριν εξετάσουμε αυτές τις περιπτώσεις ας δούμε εν συντομία τον SREG:

Bit	7	6	5	4	3	2	1	0	
\$3F (\$5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Εικόνα 21. Καταχωρητής Κατάστασης - Status REGISTER (SREG)

Ο καταχωρητής κατάστασης (**SREG**) είναι ένας 8-bit καταχωρητής που κάθε φορά που εκτελούνται αριθμητικές πράξεις, ανανεώνεται (refresh) με σημαίες που έχουν να κάνουν με το αποτέλεσμα της πράξης.

- **I (Global Interrupt Enable)**: Επιτρέπει τις διακοπές (Interrupts) στον επεξεργαστή.
- **T (Target)**: Οι εντολές αντιγραφής ενός Bit από μια θέση μνήμης έχουν σαν θέση αποθήκευσης το συγκεκριμένο πεδίο.
- **H (Half Carry)**: Κρατούμενο που προκύπτει από τα τέσσερα λιγότερο σημαντικά ψηφία.
- **S (Sign)**: Δείκτης πρόσημου, αποτέλεσμα του «αποκλειστικού OR» EXOR μεταξύ των πεδίων V και N.
- **V (Overflow)**: Δείχνει υπερχείλιση σε πράξεις συμπληρώματος ως προς 2
- **N (Negative)**: Δηλώνει ότι ο αριθμός που προκύπτει από μια αριθμητική ή λογική πράξη είναι αρνητικός
- **Z (Zero)**: Δηλώνει ότι το αποτέλεσμα μιας πράξης είναι 0
- **C (Carry)**: Δείκτης που δηλώνει ότι προέκυψε κρατούμενο από μια αριθμητική ή λογική πράξη.

Στην πράξη οι περισσότερο χρησιμοποιούμενες - λαμβανόμενες υπ όψιν σημαίες είναι οι: **I**, **N**, **Z**, **C** ενώ οι: **T**, **H**, **S**, **V** έχουν δευτερεύουσα σημασία.

Δεν θα δούμε όλες τις εντολές διακλάδωσης υπό συνθήκη, αλλά αυτές που χρησιμοποιούνται πιο συχνά:

CPSE Rd,Rr : Αν $Rd = Rr$ τότε $PC \leftarrow PC + 2$ (ή 3)
αλλιώς $PC \leftarrow PC + 1$

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 μεταξύ του καταχωρητή Rd και μιας σταθεράς.

CP Rd,Rk : $Rd - Rk$

Περιγραφή: Σύγκριση με υπολογισμό της διαφοράς μεταξύ του δυο καταχωρητών

Οι εντολές CP, CPI χρησιμοποιούνται απλά για να κάνουν μια σύγκριση. Για το πώς θα εκμεταλλευτούμε το αποτέλεσμα αυτής της σύγκρισης χρησιμοποιούμε τις εντολές “**BRANCH**”:

BRXX k: όπου **XX** μπορεί να είναι EQ, LO, NE, GE κλπ. Αφού έχουν συγκριθεί 2 Registers, ανάλογα με την κατάσταση που έχουν οι σημαίες στον SREG γίνεται διακλάδωση στο ανάλογο label (στην διεύθυνση $PC+k+1$ της μνήμης flash με $-64 < k < +63$)

BRNE k: Αν $Rd \neq Rr$ ($Z = 0$) τότε $PC \leftarrow PC + k + 1$,
αλλιώς $PC \leftarrow PC + 1$

Περιγραφή: Σχετικό άλμα σε περίπτωση ανισότητας. Όταν η σημαία μηδενός ισούται με 0 (**Z=0**) τότε εκτελείται σχετική διακλάδωση.

BREQ k: Αν $Rd = Rr$ τότε $PC \leftarrow PC + k + 1$,
αλλιώς $PC \leftarrow PC + 1$

Περιγραφή: Σχετικό άλμα σε περίπτωση ισότητας.

BRGE k : Αν $R_d \geq R_r$ ($S = N \oplus V = 0$) τότε $PC \leftarrow PC + k + 1$,
αλλιώς $PC \leftarrow PC + 1$

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση προσημασμένου δώσει μεγαλύτερο ή ίσο (**S = 0**) τότε εκτελείται σχετική διακλάδωση.

BRLO k : Αν $R_d < R_r$ τότε $PC \leftarrow PC + k + 1$,
αλλιώς $PC \leftarrow PC + 1$

Περιγραφή: Σχετικό άλμα υπό συνθήκη. Όταν η σύγκριση προσημασμένου δώσει μικρότερο (**S = 0**) τότε εκτελείται σχετική διακλάδωση.

- **Διακλαδώσεις που γίνεται έλεγχος ενός bit:**

Κατ' ουσία όλες οι παραπάνω εντολές Branch εξετάζουν κάποιο bit στον SREG και ανάλογα μεταβαίνει ο PC. Όμως υπάρχει η δυνατότητα να εξεταστεί ένα bit και να πραγματοποιηθεί διακλάδωση χωρίς τη μεσολάβηση της εντολής CP, CPI κλπ. Γίνεται έλεγχος bit κατ' ευθείαν σε κάποιο από τα 8 bit ενός register ή μιας I/O πόρτας. Χρησιμοποιούνται οι εντολές **SBRS**, **SBRC**, **SBIS** ή **SBIC** (Skip if Bit in Register or I/O is Set or Clear).

Εντολή	Περιγραφή
SBRS Rd, b	Αγνοείται η επόμενη εντολή αν το bit b με $0 \leq b \leq 7$ σε έναν καταχωρητή είναι σε λογικό 1.
SBRC Rd, b	Αγνοείται η επόμενη εντολή αν το bit b με $0 \leq b \leq 7$ σε έναν καταχωρητή είναι σε λογικό 0.
SBIS A, b	Αγνοείται η επόμενη εντολή αν το bit b με $0 \leq b \leq 7$ σε μία I/O θέση μνήμης A είναι σε λογικό 1.
SBIC A, b	Αγνοείται η επόμενη εντολή αν το bit b με $0 \leq b \leq 7$ σε μία I/O θέση μνήμης A είναι σε λογικό 0.

Βοηθητικές εντολές

- **Αύξηση ή μείωση του περιεχομένου του καταχωρητή**

INC Rd : Αύξηση του περιεχομένου του Rd κατά 1

DEC Rd : Μείωση του περιεχομένου του Rd κατά 1

Οι παραπάνω εντολές ισχύουν για όλους τους καταχωρητές (R0 – R31)

- **Αύξηση ή μείωση της διεύθυνσης μνήμης κατά 1**

X+ : Αύξηση της διεύθυνσης μνήμης κατά 1

X- : Μείωση της διεύθυνσης μνήμης κατά 1

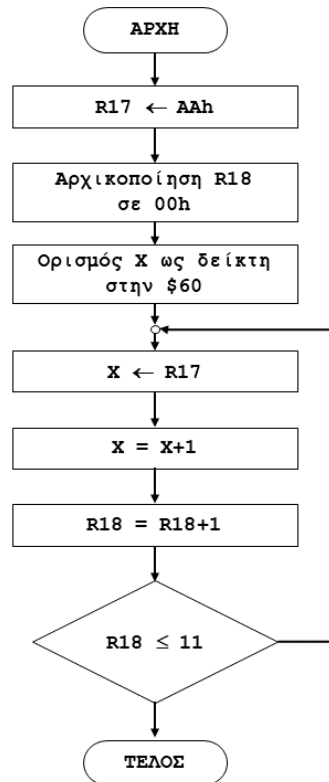
Οι παραπάνω εντολές ισχύουν για όλους του δείκτες διεύθυνσης μνήμης (X,Y,Z)

Παραδείγματα:

Παράδειγμα 1^ο - Αντιγραφή αριθμού σε συνεχόμενες θέσεις μνήμης

Γράψτε ένα πρόγραμμα στο οποίο θα αποθηκεύεται ο αριθμός AAh στις θέσεις μνήμης από \$60 έως και \$6A. Επιβεβαιώστε την εγγραφή στις συγκεκριμένες θέσεις μνήμης. Υπόδειξη: Αρχικά να γίνει διάγραμμα ροής.

Απάντηση:



```

.include "8515def.inc"
; R17 ← $FF, αυτό που θα αποθηκεύεται κάθε φορά
LDI   R17,$AA
; R18 ← $00, χρήση ως μετρητή

LDI   R18,$00
; X = $0060
LDI   XL,$60
LDI   XH,$00

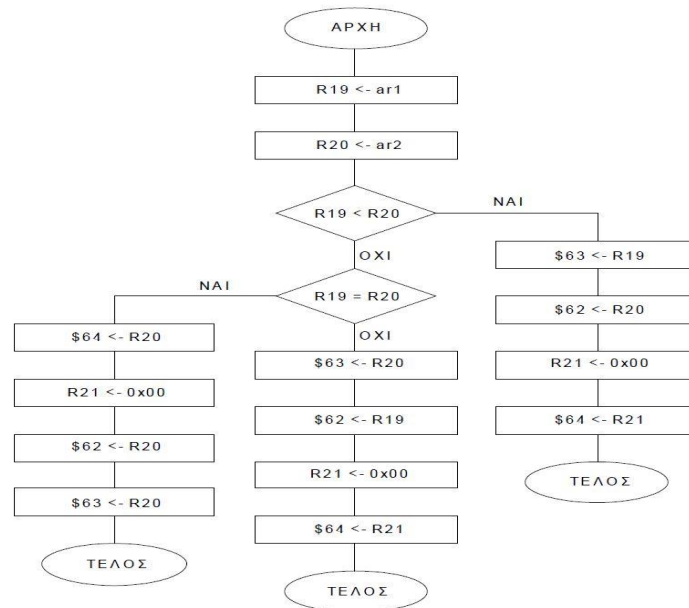
LOOP:
; (X) ← R17 και X = X+1
ST    X+,R17
; R18 = R18 + 1
INC   R18
; Εάν R18 < 11 επιστροφή στο LOOP
CPI   R18,11
BRLO  LOOP
  
```

Παράδειγμα 2^ο - Σύγκριση 2 αριθμών και αποθήκευση σε συγκεκριμένες θέσεις μνήμης SRAM

Έχετε **δύο τυχαίους αριθμούς** στους καταχωρητές εργασίας **R19** και **R20**. Γράψτε το διάγραμμα ροής και το πρόγραμμα, το οποίο να τοποθετεί τον μεγαλύτερο στη θέση μνήμης **62**, τον μικρότερο στην **63** και να τοποθετεί στην **64** το μηδέν (**0x00**). Στην περίπτωση όμως, που είναι ίσοι οι δυο αριθμοί, να τοποθετεί έναν από τους δυο στη θέση μνήμης **64** και τις θέσεις **62** και **63** να τοποθετεί το μηδέν (**0x00**), αφού δεν υπάρχει μέγιστος και ελάχιστος.

Τρέξτε το πρόγραμμα δυο φορές, μια με άνισους αριθμούς για να επιβεβαιώσετε την εύρεση του μέγιστου / ελάχιστου και μια με ίσους, για να επιβεβαιώσετε τη λειτουργία του στην ισότητα.

Απάντηση:



```

.include "8515def.inc"
; Αποθήκευση δύο τυχαίων αριθμών στους καταχωρητές R19 και R20
LDI R19, $25
LDI R20, $25
; Σύγκριση των καταχωρητών R19 και R20
CP R19, r20
; Περίπτωση R19 < R20
BRLO CASE_LESS
; Περίπτωση R19 = R20
BREQ CASE_EQUAL
; Περίπτωση R19 > R20 - Δεν χρειάζεται "ειδική" περίπτωση...
STS $63, R20
STS $62, R19
LDI R21, $00
STS $64, R21
RJMP END
; Εντολές περίπτωσης R19 < R20
CASE_LESS:
STS $63, R19
STS $62, R20
LDI R21, $00
STS $64, R21
RJMP END
; Εντολές περίπτωσης R19 = R20
CASE_EQUAL:
STS $64, R20
LDI R21, $00
STS $62, R21
STS $63, R21
; Προορισμός του rjmp (κενός κύκλος)
END: NOP
  
```


Ασκήσεις 5ης Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

1. Να γίνει πρόγραμμα που να μηδενίζει τις θέσεις μνήμης RAM από 60h μέχρι 8Fh
 2. Να γραφεί πρόγραμμα που να μεταφέρει τον πίνακα θέσεων μνήμης από την 60h μέχρι 6Fh σε μια άλλη περιοχή που αρχίζει από την 100h
 3. Έχετε **δύο τυχαίους αριθμούς** στους καταχωρητές εργασίας **R19** και **R20**. Γράψτε το διάγραμμα ροής και το πρόγραμμα που υλοποιεί τις ακόλουθες διαδικασίες:
 - A. στην περίπτωση που **R19 > R20**, να εκτελεί την λογική πράξη **R19(XOR)R20** και το αποτέλεσμα να αποθηκεύεται στην διεύθυνση μνήμης **\$65**. Στη συνέχεια να φορτώνεται ο καταχωρητής **R25** άμεσα με τον αριθμό **0xB8** και να προστίθεται με το περιεχόμενο της **\$65**. Το αποτέλεσμα της πρόσθεσης να αποθηκεύεται στις θέσεις μνήμης **\$67** (LowByte) και **\$66** (Byte κρατούμενου).
 - B. στην περίπτωση που **R19 < R20**, να εκτελείται η πρόσθεση μεταξύ των 16 bit αριθμών (**R20, R19**) + **0xF37A** και το αποτέλεσμα της πράξης να αποθηκεύεται στις διευθύνσεις μνήμης **\$70** (LowByte), **\$71** (HighByte) και **\$72** (Byte κρατούμενου).
 - C. στην περίπτωση που **R19 = R20**, να φορτώνεται τις θέσεις μνήμης από **\$70** έως και **\$75** με το **0x00**.
- Τρέξτε το πρόγραμμα για να επιβεβαιώσετε την σωστή του λειτουργία για όλες τις παραπάνω περιπτώσεις.

Εργαστήριο 6^ο – Υποπρογράμματα - Λειτουργίες I/O (Εισόδου-Εξόδου)

Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Υποπρογράμματα
3. Δείκτης Στοίβας (Stack Pointer)
4. Εντολές εισόδου – Εξόδου
5. Παραδείγματα
 - Παράδειγμα 1^ο – Έλεγχος LED με χρήση θυρών & Υποπρόγραμμα καθυστέρησης
6. Ασκήσεις 6ης Εργαστηριακής Άσκησης

1. Εισαγωγή

Ο χωρισμός ενός προγράμματος σε υποπρογράμματα προϋποθέτει την ανάλυση του αρχικού προβλήματος σε μικρότερα υπο-προβλήματα, τα οποία μπορούν να αντιμετωπισθούν ανεξάρτητα το ένα από το άλλο. Η ανάλυση αυτή δεν είναι πάντα εύκολη και δεν υπάρχουν συγκεκριμένοι κανόνες για την επιτυχή ανάλυση. Η δυσκολία αυξάνεται όσο πιο μεγάλο και πιο σύνθετο είναι το πρόβλημα. Η σωστή εφαρμογή του τμηματικού προγραμματισμού απαιτεί μελέτη στην ανάλυση του προβλήματος, εμπειρία στο προγραμματισμό, ταλέντο και φυσικά γνώσεις.

Όλοι οι μικροελεγκτές της οικογένειας AVR διαθέτουν μεταξύ άλλων και κάποιο αριθμό ακροδεκτών εισόδου – εξόδου, ξεκινώντας από 3 που συναντούμε στον μικροελεγκτή AT90S2323, μέχρι και 48, που συναντούμε στον μικροελεγκτή Mega103. Όλοι οι ακροδέκτες εισόδου – εξόδου των μικροελεγκτών AVR έχουν τη δυνατότητα να παρέχουν ρεύμα μέχρι 20 mA, ποσότητας ικανής για απευθείας οδήγηση διόδων LED χωρίς την ανάγκη χρήσης κατάλληλης εξωτερικής μονάδας απομόνωσης (buffer).

2. Κλήση υπορουτινών - υποπρογραμμάτων

Η κλήση υπορουτινών γίνεται με τις εντολές που ακολουθούν:

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. Στην ουσία υπορουτίνα (subroutine), καλείται ένα κομμάτι προγράμματος που αρχίζει με μια ετικέτα και τελειώνει με την εντολή RET. Η υπορουτίνα καλείτε (εκτελείται) όταν χρησιμοποιήσουμε την εντολή RCALL και όπου k βάλουμε την ετικέτα με την οποία αρχίζει η υπορουτίνα. Αφού εκτελεστεί η υπορουτίνα, η εκτέλεση του προγράμματος επιστρέφει (**RETurn**) μια εντολή μετά από την RCALL που είχαμε χρησιμοποιήσει.

3. Δείκτης Στοίβας (Stack Pointer - SP)

Ο Δείκτης Στοίβας αρχικοποιεί την στοίβα στην SRAM του μικροεπεξεργαστή μας. Οι λειτουργίες στοίβας είναι απαραίτητες όταν δουλεύουμε με υπορουτίνες ή interrupts. Όταν καλούμε μια υπορουτίνα ή μια ρουτίνα ελέγχου interrupt, η πραγματική διεύθυνση καταχωρείται στην στοίβα, έτσι ώστε, σε περίπτωση επιστροφής, το πρόγραμμα να συνεχίσει την λειτουργία του από εκεί που σταμάτησε.

Η στοίβα βρίσκεται στην πάνω περιοχή της SRAM του μικροεπεξεργαστή μας. Το επάνω αυτό μέρος ονομάζεται RAMEND και ορίζεται στο αρχείο "xxxxdef.inc" για τον κάθε μικροεπεξεργαστή, έτσι εμείς δεν χρειάζεται να θυμόμαστε την πραγματική τιμή του για κάθε μικροεπεξεργαστή.

Ο Δείκτη Στοίβας είναι ένας δείκτης 16bit ο οποίος χρησιμοποιείτε αντίστοιχα με τις θύρες εισόδου/εξόδου (Ports). Ο διπλός καταχωρητής ονομάζεται SPH:SPL. Ο SPH, καταχωρεί το most significant address byte, ενώ ο SPL το least significant. Για να ισχύει το προηγούμενο ο AVR θα πρέπει να έχει SRAM μεγαλύτερη από 256 byte, σε διαφορετική περίπτωση ο SPH μένει χωρίς όρισμα (πράγμα που δεν επιτρέπεται) και έτσι δεν μπορεί να χρησιμοποιηθεί.

Στο παρακάτω παράδειγμα φαίνεται η αρχικοποίηση του Δείκτη Στοίβας (Stack Pointer)

- Αρχικοποίηση του Δείκτη Στοίβας (Stack Pointer)

```
sp_init: ; Ετικέτα αρχικοποίησης του Stack Pointer
LDI R16, LOW(RAMEND)
OUT SPL,R16
LDI R16, HIGH(RAMEND)
OUT SPH,R16
```

4. Εντολές εισόδου-εξόδου (I/O)

Για κάθε μια από τις θύρες εισόδου – εξόδου του συστήματος υπάρχουν τρεις (3) διευθύνσεις στη μνήμη εισόδου – εξόδου, οι οποίες σχετίζονται με κάθε μία από αυτές.

- α) Η μία από τις διευθύνσεις αυτές απαιτείται για τον καθορισμό της κατεύθυνσης των ακροδεκτών, δηλαδή καθορίζει ποιοι από τους ακροδέκτες θα λειτουργούν ως είσοδοι και ποιοι ως εξοδοι και συμβολίζονται ως **DDRx**. Ο καταχωρητής **DDRx** είναι ο λεγόμενος καταχωρητής κατεύθυνσης. Τοποθετώντας ένα bit του καταχωρητή αυτού σε λογικό 1, ο ακροδέκτης της θύρας **PORTx** στον οποίο αντιστοιχεί, θα λειτουργεί ως έξοδος. Μετά από αυτή την ενέργεια μπορούμε να οδηγούμε κατά βούληση την λογική στάθμη του αντίστοιχου ακροδέκτη εξόδου χρησιμοποιώντας τις εντολές **CBI** ή **SBI**, ή ακόμη και τη εντολή **OUT**.
- β) Η δεύτερη διεύθυνση αφορά στα δεδομένα που πρόκειται να εγγραφούν σε εκείνους (ή και όλους) τους ακροδέκτες που έχουν προγραμματιστεί ως εξοδοι και συμβολίζονται ως **PORTx**.
- γ) Τέλος, η τρίτη διεύθυνση αφορά στα δεδομένα που διαβάζονται από εκείνους (ή και όλους) τους ακροδέκτες που έχουν προγραμματιστεί ως είσοδοι συμβολίζονται ως **PINx**.

Οι πιο απλές εντολές εισόδου-εξόδου είναι η **'in'** και **'out'**.

Η εντολή **'in'** διαβάζει την τιμή μιας θύρας ή του καταχωρητή ενός εσωτερικού περιφερειακού (χρονιστή, UART) και τη σώζει σε έναν καταχωρητή.

Σύνταξη:	Καταχωρητές:	Διευθύνσεις:
IN Rd,A	$0 \leq d \leq 31$	$0 \leq A \leq 63$

- Παράδειγμα:

in r25, PINB ; Διάβασε την Port B και μετέφερε τα περιεχόμενα στον R25

Η εντολή **'out'** γράφει την τιμή ενός καταχωρητή σε μια θύρα ή στον καταχωρητή ενός εσωτερικού περιφερειακού.

Σύνταξη:	Καταχωρητές:	Διευθύνσεις:
OUT A,Rr	$0 \leq r \leq 31$	$0 \leq A \leq 63$

- Παράδειγμα:

out PORTB, r16 ; Γράψε τα περιεχόμενα του R16 στην Port B

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.

- Μετατροπή θύρας σε είσοδο:

LDI R12, 0b00000000 ; τα ψηφία της θύρας PORTD
OUT DDRD, R12 ; γίνονται είσοδοι

- Εναλλακτικά η PORTD μετατρέπεται σε είσοδο :

LDI R12, 0x00 ; τα ψηφία της θύρας PORTD
OUT DDRD, R12 ; γίνονται είσοδοι

- ή

CLR R21 ; τα ψηφία της θύρας PORTD γίνονται είσοδοι
OUT DDRD, R21 ; μετατροπή θύρας σε είσοδο
LDI R18, 0b11111111 ; τα ψηφία της θύρας PORTB
OUT DDRB, R18 ; γίνονται εξοδοι

- Μετατροπή θύρας μετατρέπεται σε έξοδο

```
LDI R22, 0xFF ; τα ψηφία της θύρας PORTB
OUT DDRB, R22 ; γίνονται έξοδοι
```

- ή

```
SER R19 ; τα ψηφία της θύρας PORTB
OUT DDRB, R19 ; γίνονται έξοδοι
```

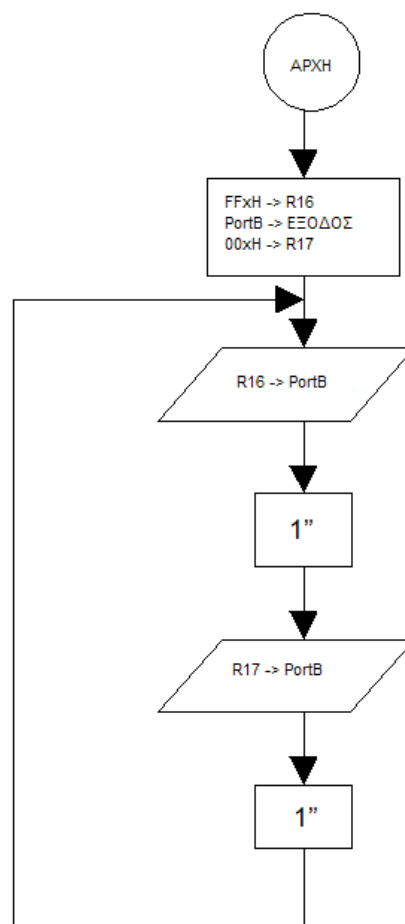
5. Παραδείγματα:

- **Παράδειγμα 1ο - Έλεγχος LED με χρήση θυρών & Υποπρόγραμμα καθυστέρησης**

(α) Σχεδιάστε το διάγραμμα ροής και το πρόγραμμα με το οποίο θα αναβοσβήνουν και τα 8 LED με περιοδικότητα περίπου ένα (1) δευτερόλεπτο. Δηλαδή θα είναι όλα «ON» για 1 δευτερόλεπτο και όλα «OFF» για επίσης 1 δευτερόλεπτο κ.ο.κ.

(β) Κάνετε το ίδιο με τους αριθμούς 55 και AA, με περιοδικότητα 2 δευτερόλεπτα (διπλάσια από το ερώτ. α).

Απάντηση:



```
.include "8515def.inc"
```

```
sp_init: ; Αρχικοποίηση του Stack Pointer
LDI R16, LOW(RAMEND)
OUT SPL,R16
LDI R16, HIGH(RAMEND)
OUT SPH,R16

SER R16 ; Όλα "1" στον r16
OUT DDRB, R16 ; Ορίζουμε την Port B ως έξοδο
CLR R17 ; Όλα "0" στον r17
```

```

loop:
OUT PORTB,R17      ; Γράφουμε στην Port B τα περιεχόμενα του R17
RCALL delay        ; Καλούμε την υπορουτίνα "delay"
OUT PORTB,R16      ; Γράφουμε στην Port B τα περιεχόμενα του R16
RCALL delay        ; Καλούμε την υπορουτίνα "delay"
RJMP loop          ; Καλούμε την υπορουτίνα "loop", επιστροφή στην αρχή!

delay:
LDI R20,$0A        ; Φόρτωση του μέγιστου αριθμού επαναλήψεων
d1:
DEC R20            ; Μείωση (κατά 1) του περιεχομένου του R20
BRNE d1            ; Έως ότου γίνει "0", επιστροφή στο "d1"
RET                ; Επιστροφή στην επόμενη γραμμή (από την κλήση)

```

Αποσύνθεση (disassembly) του προγράμματος από τον μεταφραστή (compiler)

Η αποσύνθεση του προγράμματος γίνεται από τον μεταφραστή και έχει σαν σκοπό να ελεγχθεί κάθε γραμμή για την ορθότητά της (συντακτικά) έτσι ώστε να είναι κατανοητό από τον επεξεργαστή!

Παρακάτω είναι η μετάφραση του προηγούμενου παραδείγματος. Οι αριθμημένες γραμμές είναι ο κώδικας του AVR Studio, ενώ οι υπογραμμισμένες είναι η μετάφραση.

```

4: LDI R16, LOW(RAMEND)
00000000 LDI R16,0x5F          Load immediate
5: OUT SPL,R16
00000001 OUT 0x3D,R16         Out to I/O location
6: LDI R16, HIGH(RAMEND)
00000002 LDI R16,0x02         Load immediate
7: OUT SPH,R16
00000003 OUT 0x3E,R16         Out to I/O location
9: SER R16 ; Όλα "1" στον r16
00000004 SER R16              Set Register
10: OUT DDRB, R16 ; Ορίζουμε την Port B ως έξοδο
00000005 OUT 0x17,R16         Out to I/O location
11: CLR R17 ; Όλα "0" στον r17
00000006 CLR R17              Clear Register
14: OUT PORTB,R17 ; Γράφουμε στην Port B τα περιεχόμενα του R17
00000007 OUT 0x18,R17         Out to I/O location
15: RCALL delay ; Καλούμε την υπορουτίνα "delay"
00000008 RCALL PC+0x0004      Relative call subroutine
16: OUT PORTB,R16 ; Γράφουμε στην Port B τα περιεχόμενα του R16
00000009 OUT 0x18,R16         Out to I/O location
17: RCALL delay ; Καλούμε την υπορουτίνα "delay"
0000000A RCALL PC+0x0002      Relative call subroutine
18: RJMP loop ; Καλούμε την υπορουτίνα "loop", επιστροφή στην αρχή!
0000000B RJMP PC-0x0004      Relative jump
21: LDI R20,$0A ; Φόρτωση του μέγιστου αριθμού επαναλήψεων
0000000C LDI R20,0x0A         Load immediate
23: DEC R20 ; Μείωση (κατά 1) του περιεχομένου του R20
0000000D DEC R20              Decrement
24: BRNE d1 ; Έως ότου γίνει "0", επιστροφή στο "d1"
0000000E BRNE PC-0x01         Branch if not equal
25: RET ; Επιστροφή στην επόμενη γραμμή (από την κλήση)
0000000F RET                  Subroutine return

```

Δεκαεξαδική μορφή του προγράμματος (γλώσσα μηχανής)

Η δεκαεξαδική μορφή του προγράμματος γνωστή και ως «γλώσσα μηχανής» προκύπτει από τον μεταφραστή και είναι αυτό που μεταφέρεται τελικά στον επεξεργαστή!

```
:0200000020000FC
:100000000FE50DBF02E00EBF0FEF07BB112718BBB6
:1000100003D008BB01D0FBCF4AE04A95F1F7089521
:000000001FF
```

- **Παράδειγμα Μικρής Χρονοκαθυστερήσης:**

```
delay:      ldi r26,$FF          ; Καταχώρηση του FF στον R26
d1:         dec r26              ; Μείωση κατά 1 του R26
           brne d1
ret         ; Επιστροφή στο κυρίως πρόγραμμα
```

- **Παράδειγμα Μεγάλης Χρονοκαθυστερήσης:**

```
delay:      ldi r26,$FF          ; Καταχώρηση του FF στον R26
d3:         ldi r27,$FF          ; Καταχώρηση του FF στον R27
d2:         ldi r28,$FF          ; Καταχώρηση του FF στον R28
d1:         dec r28              ; Μείωση κατά 1 του R28
           brne d1
           dec r27              ; Μείωση κατά 1 του R27
           brne d2
           dec r26              ; Μείωση κατά 1 του R26
           brne d3
ret         ; Επιστροφή στο κυρίως πρόγραμμα
```

6. Ασκήσεις 6^{ης} Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

- Σχεδιάστε το διάγραμμα ροής και γράψτε το πρόγραμμα με το οποίο, ανάλογα με την τιμή που περιέχει ο καταχωρητής R20 θα συμβαίνει μια από τις δυο λειτουργίες:
- (α) Η πρώτη λειτουργία θα είναι να αναβοσβήνουν και τα 8 LED με περιοδικότητα 1'' (δηλαδή θα είναι όλα «ON» για 1'' και όλα «OFF» για επίσης 1'' κ.ο.κ.) όταν το περιεχόμενο του R20 είναι μεγαλύτερο από τον αριθμό 128.
- (β) Η δεύτερη λειτουργία θα είναι να εναλλάσσονται ανά 0.5'' οι αριθμοί «10101010» και «01010101» όταν το περιεχόμενο του R20 είναι μικρότερο ή ίσο από τον αριθμό 128.

Εργαστήριο 7^ο – Πλατφόρμα ανάπτυξης Arduino

Περίγραμμα Εργαστηριακής Άσκησης

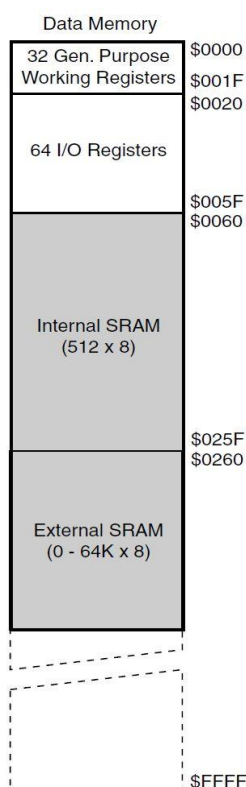
1. Εισαγωγή
2. Πλακέτα ανάπτυξης Arduino Uno
3. Προγραμματισμός του Arduino
4. Ανάπτυξη προγραμμάτων στο περιβάλλον Arduino
5. Παραδείγματα
 - Παράδειγμα 1^ο – Αναβόσβημα LED
 - Παράδειγμα 2^ο – Ανάγνωση Ψηφιακής Εισόδου
6. Ασκήσεις 7ης Εργαστηριακής Άσκησης

1. Εισαγωγή

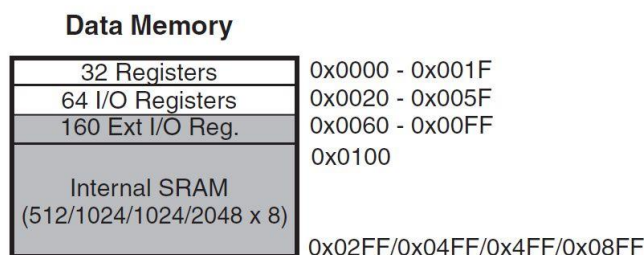
Η πλατφόρμα ανάπτυξης Arduino είναι μια πλατφόρμα φυσικών υπολογισμών (physical computing platform), η οποία βασίζεται σε μία απλή πλακέτα ανάπτυξης μικροελεγκτή (hardware) και σε περιβάλλον ανάπτυξης (software) για την εγγραφή λογισμικού στην πλακέτα. Είναι σχεδιασμένη για να διευκολύνει τη διαδικασία χρήσης ηλεκτρονικών εξαρτημάτων σε πολυποίκιλες εφαρμογές.

Μπορεί να χρησιμοποιηθεί για την ανάπτυξη προγραμμάτων αλληλεπίδρασης, εισάγοντας δεδομένα από μια ποικιλία διακοπτών και αισθητήρων και ελέγχοντας μεγάλη ποικιλία αυτών (φώτα, μοτέρ και άλλες φυσικές εξόδους). Τα προγράμματα (project) αυτά μπορεί να είναι αυτόνομα ή μπορούν να επικοινωνούν με λογισμικό που τρέχει στον υπολογιστή (Flash, Processing, MaxMSP).

Η πλακέτα ανάπτυξης του Arduino αποτελείται από ένα 8-bit Atmel AVR μικροελεγκτή (ATmega328P) με συμπληρωματικές συνδέσεις για τη διευκόλυνση του προγραμματισμού και την ενσωμάτωση σε άλλα κυκλώματα. Οι περισσότερες πλακέτες περιλαμβάνουν ένα 5V γραμμικό ρυθμιστή τάσης (regulator) και ένα 16MHz κρυσταλλικό ταλαντωτή.



Εικόνα 22. Οργάνωση μνήμης AT90S8515



Εικόνα 23. Οργάνωση μνήμης ATmega328P (ενσωματωμένος στην πλακέτα του Arduino)

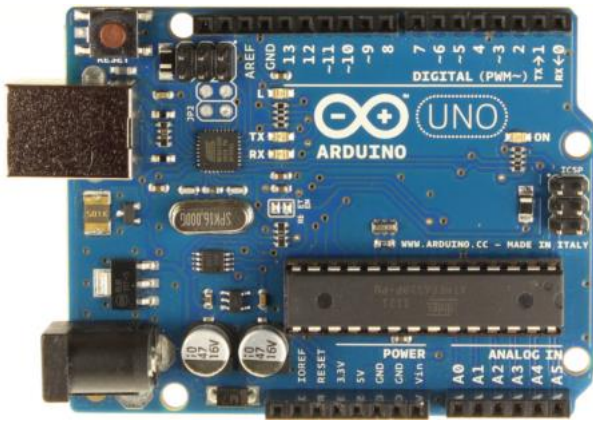
Το λογισμικό (software) Arduino αποτελείται από ένα μεταφραστή σε γλώσσα μηχανής (compiler) και από ένα εκκινητή (boot loader) που τρέχει στην πλακέτα ανάπτυξης. Ο μικροελεγκτής προγραμματίζεται εκ των προτέρων με τον εκκινητή ο οποίος απλοποιεί τη φόρτωση προγραμμάτων στη flash memory του μικροεπεξεργαστή.

Υπάρχει πληθώρα πλακετών ανάπτυξης, η καθεμία από τις οποίες έχει διαφορετικές δυνατότητες. Στα πλαίσια των εργαστηριακών ασκήσεων θα χρησιμοποιηθεί η πλακέτα ανάπτυξης Arduino Uno.

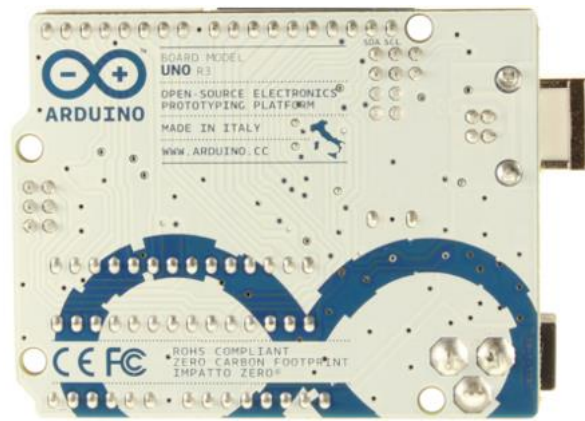
2. Πλακέτα ανάπτυξης Arduino Uno

Η λειτουργία του Arduino Uno βασίζεται στο μικροεπεξεργαστή ATmega328 της εταιρείας Atmel. Διαθέτει 14 ψηφιακούς ακροδέκτες εισόδου/εξόδου (6 από τους οποίους μπορούν να χρησιμοποιηθούν ως 8bit κλιμακωτές ψηφιακές εξόδους PWM), 6 αναλογικές εισόδους, ένα 16MHz κρυσταλλικό ταλαντωτή, σύνδεση μέσω USB θύρας, υποδοχή τροφοδοσίας, ένα ICSP κύκλωμα (In Circuit Serial Programming) και ένα κουμπί επαναφοράς ρυθμίσεων. Περιέχει οτιδήποτε χρειάζεται για τον προγραμματισμό του μικροελεγκτή. Προκειμένου να ξεκινήσει η λειτουργία

του πρέπει απλά να συνδεθεί σε ένα ηλεκτρονικό υπολογιστή μέσω USB θύρας ή να τροφοδοτηθεί με ένα AC-to-DC τροφοδοτικό ή με μπαταρία.



Εικόνα 24. Εμπρός πλευρά του Arduino Uno



Εικόνα 25. Πίσω πλευρά του Arduino Uno

• Πίνακας βασικών χαρακτηριστικών του Arduino Uno

Μικροελεγκτής:	ATmega328P
Τάση λειτουργίας:	5V
Τάση τροφοδοσίας (προτεινόμενη):	7-12V
Τάση τροφοδοσίας (όρια):	6-20V
Ψηφιακοί ακροδέκτες εισόδου εξόδου (I/O pins):	14 (6 από τις οποίες με PWM output)
Αναλογικοί ακροδέκτες εισόδου:	6
DC Ρεύμα για κάθε ακροδέκτη εισόδου εξόδου:	40 mA
DC Ρεύμα για ακροδέκτη 3.3V:	50 mA
Μνήμη Flash:	32 KB (ATmega328) από την οποία 0.5 KB χρησιμοποιείται από τον bootloader
Μνήμη SRAM:	2 KB (ATmega328)
Μνήμη EEPROM:	1 KB (ATmega328)
Διαστάσεις (mm):	68,6mmx55,3mm

• Είσοδοι-έξοδοι (I/O)

Κάθε ένας από τους 14 ψηφιακούς ακροδέκτες του Uno μπορεί να χρησιμοποιηθεί είτε ως είσοδος είτε ως έξοδος, χρησιμοποιώντας τις συναρτήσεις `pinMode`, `digitalWrite` και `digitalRead`. Λειτουργούν στα 5 Volts. Κάθε ακροδέκτης μπορεί να παρέχει ή να λάβει ένα μέγιστο ρεύμα 40mA και διαθέτει εσωτερική pull-up αντίσταση των 20-50kΩ.

Επιπρόσθετα, κάποιοι ακροδέκτες έχουν εξειδικευμένες λειτουργίες:

- **0 (RX) και 1 (TX) – Σειριακή επικοινωνία:** Χρησιμοποιείται για να λαμβάνει (RX) και να μεταδίνει (TX) TTL σειριακά δεδομένα. Αυτοί οι ακροδέκτες συνδέονται με τους αντίστοιχους ακροδέκτες του Atmega8U2 (ενσωματωμένος μικροελεγκτής για τη σειριακή επικοινωνία) USB-to-TTL Serial chip.
- **2 και 3 - Εξωτερικές διακοπές (interrupts):** Οι ακροδέκτες αυτοί μπορούν να ρυθμιστούν για να ενεργοποιήσουν μια διακοπή σε μια χαμηλή τιμή ή μια μεταβολή σε μια τιμή.
- **3, 5, 6, 9, 10 και 11 - PWM:** Παρέχει 8-bit PWM (ψηφιακά κλιμακωτή) έξοδο.
- **10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) - SPI:** Οι ακροδέκτες αυτοί υποστηρίζουν τη σειριακή επικοινωνία SPI (Serial Peripheral Interface) χρησιμοποιώντας τη SPI βιβλιοθήκη.
- **13 - LED:** Εσωτερικό (built-in) LED συνδεδεμένο με το ψηφιακό ακροδέκτη 13. Όταν ο ακροδέκτης έχει τη τιμή HIGH το LED φωτίζει (on), ενώ όταν έχει την τιμή LOW δεν φωτίζει (off).

Ο μικροελεγκτής Uno έχει 6 αναλογικές εισόδους, με ονομασίες από A0 ως A5, η καθεμία από τις οποίες παρέχει 10bit ανάλυση (1024 διαφορετικές τιμές). Από προεπιλογή μετρούν από μηδέν (ground) ως 5V.

Επιπρόσθετα, κάποιοι ακροδέκτες έχουν εξειδικευμένες λειτουργίες:

- **A4 (SDA, Serial Data Line) και A5 (SCL, Serial Clock):** Οι ακροδέκτες αυτοί υποστηρίζουν τη σειριακή επικοινωνία TWI (Two Wire Interface) χρησιμοποιώντας τη βιβλιοθήκη Wire.

Άλλοι ακροδέκτες που υπάρχουν στην πλακέτα:

- **AREF:** Τάση αναφοράς για τις αναλογικές εισόδους.
- **Reset:** Χρησιμοποιείται για να επανεκκινήσει τη λειτουργία του μικροεπεξεργαστή.

3. Προγραμματισμός του Arduino

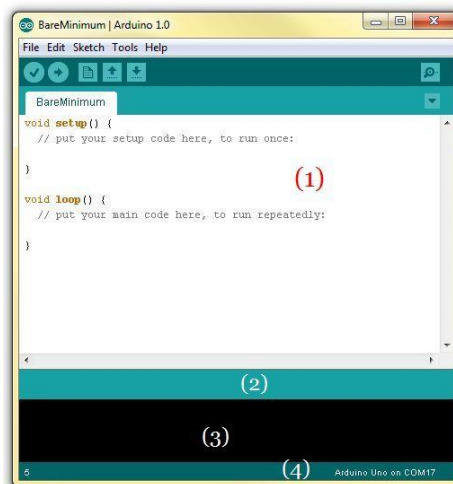
• Γλώσσα προγραμματισμού του Arduino

Η γλώσσα προγραμματισμού του Arduino είναι μια εφαρμογή βασισμένη στην πλατφόρμα Wiring. Η πλακέτα του Arduino, προγραμματίζεται χρησιμοποιώντας μια γλώσσα παρόμοια με τη C++, με κάποιες μικρές απλοποιήσεις και διαφοροποιήσεις. Το περιβάλλον ανάπτυξης του βασίζεται στη γλώσσα προγραμματισμού ανοιχτού κώδικα Processing. Η γλώσσα προγραμματισμού μπορεί να επεκταθεί χρησιμοποιώντας βιβλιοθήκες C++.

• Περιβάλλον προγραμματισμού του Arduino

Το προγραμματιστικό περιβάλλον Arduino είναι γραμμένο σε Java και περιέχει ένα επεξεργαστή κειμένου για συγγραφή κώδικα, μια περιοχή μηνυμάτων, μια κονσόλα κειμένου, μια εργαλειοθήκη και μια σειρά από μενού. Συνδέεται με την πλακέτα Arduino για να φορτώσει προγράμματα και να επικοινωνεί μαζί τους.

Το λογισμικό που γράφεται στον επεξεργαστή κειμένου χρησιμοποιώντας το περιβάλλον του Arduino ονομάζεται Sketch. Το περιβάλλον Arduino χρησιμοποιεί την ιδέα ενός sketchbook: μιας πρότυπης θέσης για την αποθήκευση των προγραμμάτων. Τα αρχεία ανοίγονται από: το μενού **File > Sketchbook** ή από το **Open** στην εργαλειοθήκη. Τα Sketch σώζονται με την επέκταση αρχείου .ino. **Το όνομα του Sketch δε μπορεί να ξεκινάει με νούμερο και πρέπει να είναι μικρότερο από 64 χαρακτήρες, χωρίς να περιέχει κενά.**



Εικόνα 26. Περιβάλλον προγραμματισμού του Arduino

Το προγραμματιστικό περιβάλλον διαθέτει:

1. Επεξεργαστή κειμένου με δυνατότητες για αποκοπή και επικόλληση και για αναζήτηση και αντικατάσταση κειμένου.
2. Περιοχή μηνυμάτων η οποία δίνει πληροφορίες ενώ σώζει, εξάγει και επίσης εμφανίζει λάθη.
3. Κονσόλα κειμένου που περιλαμβάνει πλήρη μηνύματα λάθους και άλλες πληροφορίες.
4. Εμφάνιση της επιλεγμένης πλακέτας και της σειριακής θύρας στην οποία είναι συνδεδεμένη.

Τα κουμπιά της εργαλειοθήκης επιτρέπουν την επαλήθευση και φόρτωση προγραμμάτων, τη δημιουργία, άνοιγμα και αποθήκευση των Sketch και την προβολή των δεδομένων της σειριακής θύρας:

-  **Verify** (Ελέγχει τον κώδικα για πιθανά λάθη)
-  **Upload** (Μεταφράζει τον κώδικα και το φορτώνει στη πλακέτα ανάπτυξης του Arduino)
-  **New** (Δημιουργεί ένα καινούριο Sketch)
-  **Open** (Ανοίγει ένα υπάρχων Sketch)
-  **Save** (Αποθηκεύει το Sketch)
-  **Serial Monitor** (Ανοίγει εξωτερικό παράθυρο για την προβολή δεδομένων της σειριακή θύρας)

Επιπρόσθετες εντολές βρίσκονται μέσα στα μενού: **File, Edit, Sketch, Tools, Help**.

4. Ανάπτυξη προγραμμάτων στο περιβάλλον Arduino

Ένα πρόγραμμα στο περιβάλλον Arduino μπορεί να αναλυθεί σε τρεις κύριες κατηγορίες:

- Δομή
- Τιμές (μεταβλητές και σταθερές)
- Συναρτήσεις

ΠΡΟΣΟΧΗ: Όλα τα στοιχεία των προγραμμάτων που συντάσσονται στο περιβάλλον Arduino (τιμές, συναρτήσεις, κ.λ.π.) είναι «case sensitive» (τα μικρά και τα κεφαλαία προσλαμβάνονται διαφορετικά από τον μεταφραστή!)

• Βασικές συναρτήσεις δομής

α) setup()

Η συνάρτηση setup() καλείται στην εκκίνηση του προγράμματος, όταν ξεκινάει ένα Sketch. Χρησιμοποιείται για να αρχικοποιήσει μεταβλητές (π.χ. τις λειτουργίες του ακροδέκτη (pin modes), για να ξεκινήσει τη χρήση βιβλιοθηκών κ.ά.). Η συνάρτηση setup() εκτελείται μόνο μία φορά, μετά από την έναρξη λειτουργίας της πλακέτας ανάπτυξης του Arduino.

➤ Σύνταξη συνάρτησης:

```
void setup()
{
  // ... Το σύμβολο "//" χρησιμοποιείται πριν τη χρήση σχολίων.
}
```

β) loop()

Μετά την εκτέλεση της συνάρτησης setup(), η οποία αρχικοποιεί και ορίζει τις αρχικές τιμές, η συνάρτηση loop() δημιουργεί βρόχο στο πρόγραμμα, εκτελώντας επαναλαμβανόμενες εντολές. Χρησιμοποιείται για να ελέγχει ενεργά την πλακέτα ανάπτυξης του Arduino. Η συνάρτηση αυτή καλείται επαναλαμβανόμενα μέχρι η πλακέτα ανάπτυξης να απενεργοποιηθεί (power off).

➤ Σύνταξη συνάρτησης:

```
void loop()
{
  // ...
}
```

- **Τιμές: Σταθερές (Constants)**

Οι σταθερές (constants) είναι προκαθορισμένες μεταβλητές στη γλώσσα του Arduino. Χρησιμοποιούνται προκειμένου να κάνουν τα προγράμματα ευκολότερα στην ανάγνωση. Οι σταθερές ταξινομούνται σε ομάδες:

- α) **Οι σταθερές δεδομένων αλήθειας (Boolean) true & false**

Η σταθερά "true", έχει την τιμή ένα «1», η οποία είναι αληθής. Κάθε μη μηδενικός ακέραιος θεωρείται εξ' ορισμού "true". Η σταθερά "false", έχει πάντα την τιμή μηδέν «0».

- β) **Οι ψηφιακές στάθμες HIGH & LOW**

Όταν διαβάζουμε από ή γράφουμε σε ένα ψηφιακό ακροδέκτη υπάρχουν δύο δυνατές τιμές που ο ακροδέκτης μπορεί να λάβει, την τιμή "HIGH" και την τιμή "LOW".

- γ) **Οι ρόλοι των ακροδεκτών INPUT & OUTPUT**

Οι ψηφιακοί ακροδέκτες μπορούν να χρησιμοποιηθούν είτε ως είσοδοι (INPUT) είτε ως έξοδοι (OUTPUT). Η μεταβολή του ακροδέκτη από "INPUT" σε "OUTPUT" πραγματοποιείται με τη συνάρτηση pinMode() και αλλάζει δραστικά την ηλεκτρική συμπεριφορά του ακροδέκτη.

- **Τιμές: Μεταβλητές (Variables) & Τύποι δεδομένων (Data Types)**

Βασικοί τύποι δεδομένων είναι οι: **int, void, boolean, char, long, float, byte** κ.ά.

- α) **int**

Οι ακέραιοι είναι ο βασικός τύπος δεδομένων για αποθήκευση αριθμών και αποθηκεύουν μια τιμή 2 byte.

- **Σύνταξη εντολής:**

```
int var=val;
// όπου var: το όνομα της ακεραίας μεταβλητής
// όπου val: η τιμή που ανατίθεται στη μεταβλητή
```

- ❖ **Παράδειγμα χρήσης εντολής:**

```
int ledpin=13; // η μεταβλητή «ledpin» παίρνει την τιμή «13»
```

- β) **void**

Η λέξη void χρησιμοποιείται μόνο στη δήλωση συναρτήσεων. Υποδηλώνει ότι η συνάρτηση αναμένεται να μην επιστρέψει καμία πληροφορία στη συνάρτηση από την οποία καλείται.

- ❖ **Παράδειγμα χρήσης συνάρτησης:**

```
// Πραγματοποιούνται ενέργειες στις συναρτήσεις "setup" και "loop"
// αλλά καμία πληροφορία δεν θα περάσει στο κυρίως πρόγραμμα
void setup()
{
    // ...
}

void loop()
{
    // ...
}
```

- Ψηφιακές συναρτήσεις εισόδου-εξόδου (I/O)

- α) pinMode()

Ρυθμίζει το συγκεκριμένο ακροδέκτη (pin) να συμπεριφέρεται είτε ως είσοδος (INPUT) είτε ως έξοδος (OUTPUT). Η εκτέλεση της εντολής αυτής δεν επιστρέφει τίποτα.

- Σύνταξη συνάρτησης:

```
pinMode (pin, mode)
// όπου pin: ο αριθμός του ακροδέκτη
// όπου mode: "INPUT" ή "OUTPUT"
```

- ❖ Παράδειγμα χρήσης συνάρτησης:

```
int ledPin = 13;           // LED συνδεδεμένο στο pin 13
pinMode (ledPin, OUTPUT); // θέτει τον ψηφιακό ακροδέκτη ως έξοδο
```

- β) digitalWrite()

Γράφει την τιμή HIGH ή LOW σε ένα ψηφιακό ακροδέκτη.

ΠΡΟΣΟΧΗ:

- Αν ο ακροδέκτης έχει ρυθμιστεί σαν έξοδος OUTPUT με τη συνάρτηση pinMode(), η τάση του θα ρυθμιστεί στην αντίστοιχη τιμή: 5V (ή 3.3V σε πλακέτες ανάπτυξης των 3.3V) για τη τιμή HIGH, και 0V για τη τιμή LOW.
- Αν ο ακροδέκτης έχει ρυθμιστεί σαν είσοδος INPUT, λαμβάνοντας την τιμή HIGH με τη συνάρτηση digitalWrite() θα ενεργοποιήσει μια εσωτερική pull up αντίσταση 20KΩ (*Επομένως η τιμή που θα πάρει θα είναι κάτι μικρότερο από...HIGH*). Λαμβάνοντας την τιμή LOW θα απενεργοποιηθεί η αντίσταση.

- Σύνταξη συνάρτησης:

```
digitalWrite (pin, value)
// όπου pin: ο αριθμός του ακροδέκτη
// όπου value: HIGH ή LOW
```

- ❖ Παράδειγμα χρήσης συνάρτησης:

```
digitalWrite (ledPin, HIGH); // Θέτει το LED "on"
```

- γ) digitalRead()

Διαβάζει την τιμή από ένα συγκεκριμένο ψηφιακό ακροδέκτη, είτε HIGH είτε LOW.

- Σύνταξη συνάρτησης:

```
digitalRead (pin)
// όπου pin: ο αριθμός του ακροδέκτη που πρόκειται να διαβαστεί
```

- ❖ Παράδειγμα χρήσης συνάρτησης:

```
digitalRead (inPin); // διαβάζει την τιμή που ορίζει η μεταβλητή "inPin"
```

- Χρονικές συναρτήσεις

- α) delay()

Η συνάρτηση αυτή καθυστερεί το πρόγραμμα για χρονική διάρκεια (σε millisecond) η οποία δίδεται στην παράμετρο.

- Σύνταξη συνάρτησης:

```
delay (ms)
// όπου ms: η επιθυμητή καθυστέρηση σε milliseconds
```

- ❖ Παράδειγμα χρήσης συνάρτησης:

```
delay (1000) // καθυστερεί το πρόγραμμα για 1 sec
```

- **Συναρτήσεις σειριακής επικοινωνίας**

Χρησιμοποιείται για επικοινωνία μεταξύ της πλακέτας και του υπολογιστή. Όλες οι πλακέτες ανάπτυξης Arduino έχουν το λιγότερο μια σειριακή θύρα (UART). Επικοινωνεί με τους ψηφιακούς ακροδέκτες 0 (RX) και 1 (TX) καθώς και με τον υπολογιστή μέσω USB. Προκειμένου να επιτευχθεί η επικοινωνία με την πλακέτα Arduino, χρησιμοποιείται η ενσωματωμένη στο περιβάλλον Arduino οθόνη σειριακής επικοινωνίας.

- α) **Serial.begin()**

Ορίζεται στην συνάρτηση “setup” και καθορίζει τον ρυθμό μεταφοράς των σειριακών δεδομένων σε bits per second (baud rate). Για επικοινωνία με τον υπολογιστή χρησιμοποιείται μία από τις ακόλουθες τιμές του ρυθμού μεταφοράς: **300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200**.

- **Σύνταξη συνάρτησης:**

```
Serial.begin(speed)
// όπου speed: η τιμή του ρυθμού μεταφοράς
```

- ❖ **Παράδειγμα χρήσης συνάρτησης:**

```
void setup() {
  Serial.begin(9600); // ανοίγει την σειριακή θύρα
                    // και θέτει το ρυθμό μεταφοράς σε 9600 bps
}

void loop() {}
```

- β) **Serial.println()**

Γράφει δεδομένα στη σειριακή θύρα σαν ASCII text ή άλλες μορφές ανάλογα με τις παραμέτρους που θα χρησιμοποιηθούν.

- **Σύνταξη εντολής:**

```
Serial.println(val)
Serial.println(val, format) //διαφορετικά
//όπου val: η τιμή που θα γραφτεί (οποιοδήποτε τύπο δεδομένων)
//όπου format: μορφή τιμής (δεκαδική, δεκαεξαδική, κ.λ.π)
```

- ❖ **Παράδειγμα χρήσης συνάρτησης:**

```
int analogValue = 0; // μεταβλητή που θα κρατήσει την αναλογική τιμή
void setup() {
  Serial.begin(9600);
}

void loop() {
  // διαβάζει την αναλογική τιμή από το pin 0:
  analogValue = analogRead(0);
  // γράφει την τιμή στην σειριακή σε διάφορες μορφές:
  Serial.println(analogValue); // δεκαδική μορφή
  Serial.println(analogValue, DEC); // δεκαδική μορφή
  Serial.println(analogValue, HEX); // δεκαεξαδική μορφή
  Serial.println(analogValue, OCT); // οκταδική μορφή
  Serial.println(analogValue, BIN); // δυαδική μορφή
}
```

- **Η σειριακή οθόνη**

Η σειριακή οθόνη εμφανίζει τα δεδομένα τα οποία στέλνονται στον υπολογιστή από την πλακέτα ανάπτυξης του Arduino (από τη USB ή τη σειριακή θύρα). Ο ρυθμός μεταφοράς (baud rate) από το drop-down μενού πρέπει να ταιριάζει με το ρυθμό που ορίζεται με τη συνάρτηση “Serial.begin” μέσα στο πρόγραμμα.

5. Παραδείγματα:

- **Παράδειγμα 1^ο - Αναβόσβημα LED**

Ανοίγουμε με διπλό κλικ την εφαρμογή Arduino. Στην οθόνη μας βλέπουμε ένα αρχείο Sketch. Στον επεξεργαστή κειμένου του Sketch αρχίζουμε να γράφουμε τον κώδικα μας για το αναβόσβημα του LED.

```
void setup() {  
  // ορίζουμε τον ψηφιακό ακροδέκτη "5" ως έξοδο  
  pinMode(5, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(5, HIGH); // Το LED ανάβει  
  delay(500); // καθυστέρηση ενός δευτερολέπτου  
  digitalWrite(5, LOW); // Το LED σβήνει  
  delay(500); // καθυστέρηση ενός δευτερολέπτου  
}
```

- ♦ **Μετά την εγγραφή του προγράμματος:**

- α) Ελέγχουμε & μεταφράζουμε σε γλώσσα μηχανής πατώντας το κουμπί **"Verify"**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: "Done compiling"
- β) Όταν δεν υπάρχει κάποιο συντακτικό σφάλμα στο πρόγραμμα, μεταφέρουμε το πρόγραμμα στον μικροεπεξεργαστή πατώντας το κουμπί **"Upload"**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: "Done Uploading".
- γ) Μετά από κάποια δευτερόλεπτα και αφού τελειώσει η εκτέλεση της μεταφοράς, θα δούμε ότι το LED στην πλακέτα θα αρχίσει να αναβοσβήνει.

- **Παράδειγμα 2^ο - Ανάγνωση Ψηφιακής Εισόδου**

Στο παράδειγμα αυτό θα διαβαστεί η ψηφιακή είσοδος ενός ακροδέκτη και η τιμή του θα γραφτεί στη σειριακή θύρα του υπολογιστή.

```
void setup() {  
  Serial.begin(9600); // Ορίζουμε το ρυθμό μεταφοράς της σειριακής  
  pinMode(2, INPUT); // Ορίζουμε σαν είσοδο τον ακροδέκτη "2"  
}  
  
void loop() {  
  int sensValue = digitalRead(2); // Διαβάζουμε τον ακροδέκτη  
  Serial.println(sensValue);      // Γράφουμε την τιμή του ακροδέκτη  
                                  // στη σειριακή θύρα  
}
```

Προκειμένου να παρακολουθήσουμε τα δεδομένα που στέλνονται από την πλακέτα ανάπτυξης στον υπολογιστή, μέσω της σειριακής θύρας, πατάμε το κουμπί **"Serial Monitor"** που βρίσκεται τέρμα δεξιά στη γραμμή εργαλείων.

Ασκήσεις 7ης Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

- Σχεδιάστε το διάγραμμα ροής και το γράψτε πρόγραμμα με το οποίο, πατώντας το διακόπτη που βρίσκεται στον ακροδέκτη «2» θα ανάβει το Led που βρίσκεται στον ακροδέκτη «5». Το πρόγραμμα να εκτελείτε επαναλαμβανόμενα.

Εργαστήριο 8^ο – Αναλογική επικοινωνία & Δομές ελέγχου στον Arduino

Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Αναλογική Επικοινωνία
3. Συγκρίσεις
4. Δομές ελέγχου if και if/else
5. Παραδείγματα
 - Παράδειγμα 1^ο – Διάβασμα τιμής ποτενσιόμετρου
 - Παράδειγμα 2^ο – Λειτουργία RGB Led
6. Ασκήσεις 8ης Εργαστηριακής Άσκησης

1. Εισαγωγή

Στο μάθημα αυτό θα δούμε τις βασικές συναρτήσεις αναλογικής επικοινωνίας (I/O) όπως επίσης τις συγκρίσεις και τις δομές ελέγχου if και if/else.

2. Αναλογική επικοινωνία

• Συναρτήσεις αναλογικής επικοινωνίας

α) `analogRead()`

Η συνάρτηση αυτή διαβάζει την τιμή από ένα συγκεκριμένο αναλογικό ή ψηφιακό ακροδέκτη. Η πλακέτα ανάπτυξης Arduino διαθέτει ένα 10-bit αναλογικό σε ψηφιακό μετατροπέα των 6 καναλιών. Αυτό σημαίνει ότι θα διαβαθμίσει τάσεις εισόδου μεταξύ 0 και 5 Volts σε ακέραιες τιμές μεταξύ 0 και 1023. Το γεγονός αυτό αποδίδει ανάλυση μεταξύ των αναγνώσεων των 5 Volts ανά 1024 τιμών ή των 4,9mV ανά τιμή. Απαιτούνται περίπου 100μs προκειμένου να διαβαστεί μια αναλογική είσοδος οπότε ο μέγιστος ρυθμός ανάγνωσης είναι 10000 φορές το δευτερόλεπτο (10kHz).

➤ Σύνταξη συνάρτησης

`analogRead(pin)`

```
// όπου pin ο αριθμός του αναλογικού ή ψηφιακού ακροδέκτη εισόδου  
// από τον οποίο διαβάζεται η τιμή.  
// Η εντολή αυτή επιστρέφει ακέραιο "int" με τιμή από (0 ως 1023).
```

❖ Παράδειγμα (`analogRead` - Ψηφιακός ακροδέκτης):

```
int analogPin = 3;      // Αλλαγή ονόματος του ψηφιακού ακροδέκτη "3"  
int val = 0;           // Ορισμός βοηθητικής μεταβλητής αποθήκευσης  
  
void setup()  
{  
  Serial.begin(9600);  
}  
  
void loop()  
{  
  val = analogRead(analogPin); // ανάγνωση του ψηφιακού ακροδέκτη  
  Serial.println(val);  
}
```

❖ Παράδειγμα (`analogRead` - Αναλογικός ακροδέκτης):

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  int sensorValue = analogRead(A0); // Ανάγνωση του αναλογικού ακροδ. "A0"  
  Serial.println(sensorValue);  
}
```

β) analogWrite()

Γράφει την αναλογική τιμή (PWM wave) σε ένα ακροδέκτη. Μπορεί να χρησιμοποιηθεί για να ανάψει ένα LED σε μεταβλητή φωτεινότητα ή για να οδηγήσει ένα μοτέρ σε διαφορετικές ταχύτητες. Μετά από μια κλήση της συνάρτησης **analogWrite()**, ο ακροδέκτης θα παράγει ένα σταθερό τετραγωνικό κύμα του συγκεκριμένου κύκλου λειτουργίας (duty cycle) μέχρι τη νέα κλήση του **analogWrite()**. Η συχνότητα του PWM σήματος είναι περίπου 490 Hz.

Στην πλακέτα Arduino Uno η συνάρτηση λειτουργεί στους ακροδέκτες 3, 5, 6, 9, 10 και 11. Προκειμένου να τεθεί ο ακροδέκτης ως έξοδος δε χρειάζεται να κληθεί η συνάρτηση **pinMode()** πριν την **analogWrite()** συνάρτηση.

➤ Σύνταξη συνάρτησης

```
analogWrite(pin, value)
// Όπου pin ο ακροδέκτης στον οποίο θα γραφτεί η τιμή
// Όπου value η τιμή του κύκλου λειτουργίας (duty cycle)
// μεταξύ των τιμών 0 και 255.
```

❖ Παράδειγμα (analogWrite)

```
/* Ορίζει ένα LED ως έξοδο και αυξομειώνει
τη τιμή του με ένα ποτενσιόμετρο. */
int ledPin = 9;           // LED συνδεδεμένο στον ψηφιακό ακροδέκτη 9
int analogPin = 3;        // Ποτενσιόμετρο συνδεδεμένο στον αναλ. ακροδ. 3
int val = 0;              // Μεταβλητή για αποθήκευση της τιμής του ποτενσ.

void setup()
{
  pinMode(ledPin, OUTPUT); // θέτει τον ακροδέκτη ως έξοδο
}

void loop()
{
  val = analogRead(analogPin); // διαβάζει τον ακροδέκτη εισόδου
  analogWrite(ledPin, val/4);   // analogRead values go from 0 to 1023,
                                // analogWrite values from 0 to 255
}
```

• Συναρτήσεις κλίμακας**α) constrain(x, a, b)**

Η συνάρτηση αυτή περιορίζει ένα αριθμό να είναι μέσα σε μία συγκεκριμένη κλίμακα.

Οι **παράμετροι** της συνάρτησης είναι:

- x**: ο αριθμός που περιορίζεται
- a**: το κάτω άκρο της κλίμακας
- b**: το πάνω άκρο της κλίμακας

Ενώ η συνάρτηση **επιστρέφει**:

- x**: αν ο x είναι μεταξύ του a και του b
- a**: αν ο x είναι μικρότερος του a
- b**: αν ο x είναι μεγαλύτερος του b

❖ Παράδειγμα (constrain)

```
sensVal = constrain(sensVal, 10, 150);
// περιορίζει την κλίμακα των τιμών της "sensVal" μεταξύ 10 και 150
```

β) map(value, fromLow, fromHigh, toLow, toHigh)

Αλλάζει την κλίμακα σε μια μεταβλητή. Δηλαδή, ένας αριθμός (value) που έχει την τιμή **fromLow** θα αντικατασταθεί από την τιμή **toLow**, μια τιμή του **fromHigh** αριθμού θα αντικατασταθεί από την τιμή **toHigh**, ενώ αναλογικά θα διαμορφωθούν και οι τιμές μεταξύ τους.

Η συνάρτηση map() χρησιμοποιεί ακέραιους αριθμούς οπότε δε θα παράγει κλάσματα. Τα υπόλοιπα από κλάσματα περικόπτονται και δεν στρογγυλοποιούνται.

❖ Παράδειγμα (map)

```
/* Αλλαγή κλίμακας από 10bits σε 8 bits */

void setup() {}

void loop()
{
  int val = analogRead(A0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

3. Συγκρίσεις

- Τελεστές σύγκρισης

Οι κυριότεροι τελεστές σύγκρισης είναι οι παρακάτω:

```
x==y (x ίσο με y)
x!=y (x διαφορετικό από το y)
x<y (x μικρότερο από το y)
x>y (x μεγαλύτερο από το y)
x<=y (x μικρότερο ή ίσο από το y)
x>=y (x μεγαλύτερο ή ίσο από το y)
```

!Προσοχή!:

Το σύμβολο ίσον "=" όταν χρησιμοποιείται μόνο του, αποτελεί τελεστή ανάθεσης, π.χ. αν $x=10$ η μεταβλητή x παίρνει την τιμή 10. Στις δομές ελέγχου προκειμένου να γίνει κάποια σύγκριση για την ισότητα χρησιμοποιείται διπλό ίσον "==", οπότε όταν χρησιμοποιείται ο τελεστής αυτός σύγκρισης, π.χ. $x==10$, γίνεται έλεγχος αν η x είναι ίση ή όχι με την τιμή 10. Η προηγούμενη συνθήκη ($x==10$) δηλαδή είναι αληθής αν η x ισούται με 10, ενώ όταν εκτελούμε την εντολή $x=10$ η δήλωση αυτή θα είναι πάντα αληθής.

4. Δομές ελέγχου

• Δομή ελέγχου if

Η δομή ελέγχου if χρησιμοποιείται πάντα σε συμφωνία με ένα τελεστή σύγκρισης και εξετάζει αν ικανοποιείται μια συγκεκριμένη συνθήκη, όπως για παράδειγμα μια τιμή μεταβλητής να είναι μεγαλύτερη από ένα συγκεκριμένο αριθμό.

➤ Σύνταξη

```
if (someVariable > 50)
{
    // εκτέλεσε κάποιες εντολές
}
```

Αν λοιπόν η συνθήκη που βρίσκεται μέσα στις παρενθέσεις είναι αληθής, εκτελούνται οι εντολές που βρίσκονται μέσα στις αγκύλες { }. Ότι βρίσκεται εκτός αγκυλών, το πρόγραμμα το προσπερνάει!

Εξάιρση: Οι αγκύλες μπορούν να παραληφθούν μετά από μια if δομή ελέγχου, όταν είναι η μοναδική συνθήκη και εφόσον η επόμενη γραμμή τελειώνει με ερωτηματικό ";" όπως στις ακόλουθες περιπτώσεις:

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
    digitalWrite(LEDpin1, HIGH);
    digitalWrite(LEDpin2, HIGH);
}
// όλες οι παραπάνω εντολές είναι ορθές
```

• If/else

Η if/else επιτρέπει μεγαλύτερο έλεγχο πάνω στη ροή του κώδικα από την if, επιτρέποντας πολλές δοκιμές (tests) να μπορούν να γίνουν τον ίδιο χρόνο. Για παράδειγμα, μια αναλογική είσοδος μπορεί να ελεγχθεί και μια ενέργεια θα γίνει αν η τιμή της είναι μικρότερη από 500, ενώ μια άλλη ενέργεια θα γίνει αν η τιμή της είναι μεγαλύτερη από 500.

➤ Σύνταξη

```
if (pinFiveInput < 500)
{
    // ενέργεια A
}
else
{
    // ενέργεια B
}
```

Η else μπορεί να ακολουθεί και άλλη συνθήκη if, έτσι ώστε πολλαπλές, αμοιβαία αποκλειόμενες δοκιμές μπορεί να τρέχουν στον ίδιο χρόνο.

Κάθε συνθήκη θα προχωράει στην επόμενη μέχρι να βρεθεί η αληθής. Όταν βρεθεί η αληθής συνθήκη θα εκτελεστεί ο κώδικας που αντιστοιχεί σε αυτή και το πρόγραμμα στη συνέχεια θα μεταβεί στην επόμενη γραμμή μετά το τέλος της δομής ελέγχου if/else.

Επίσης είναι επιτρεπτός ένας απεριόριστος αριθμός από else if διακλαδώσεις.

❖ Παράδειγμα (if/else)

```
if (pinFiveInput < 500)
{
  // ενέργεια Α
}
else if (pinFiveInput >= 1000)
{
  // ενέργεια Β
}
else
{
  // ενέργεια C
}
```

5. Παραδείγματα:

Παράδειγμα 1^ο – Διάβασμα τιμής ποτενσιόμετρου

Ανοίγουμε με διπλό κλικ την εφαρμογή Arduino. Ανοίγουμε το παράδειγμα “AnalogReadSerial” από την διαδρομή: File→Examples→Basics→AnalogReadSerial.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(5);
}
```

♦ Μετά το άνοιγμα του προγράμματος:

- α) Ελέγχουμε & μεταφράζουμε σε γλώσσα μηχανής πατώντας το κουμπί **“Verify”**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: “Done compiling”
- β) Όταν δεν υπάρχει κάποιο συντακτικό σφάλμα στο πρόγραμμα, μεταφέρουμε το πρόγραμμα στον μικροεπεξεργαστή πατώντας το κουμπί **“Upload”**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: “Done Uploading”.
- γ) Προκειμένου να παρακολουθήσουμε τα δεδομένα που στέλνονται από την πλακέτα ανάπτυξης στον υπολογιστή, μέσω της σειριακής θύρας, πατάμε το κουμπί **“Serial Monitor”** που βρίσκεται τέρμα δεξιά στη γραμμή εργαλείων.
- δ) Γυρνώντας τον διακόπτη του ποτενσιόμετρου θα πρέπει να διαβάζουμε την ένδειξή του στο παράθυρο του **“Serial Monitor”**.

Παράδειγμα 2ο – Λειτουργία RGB Led

Ανοίγουμε με διπλό κλικ την εφαρμογή Arduino. Γράφουμε τον παρακάτω κώδικα:

```
int REDPin = 9;           // Ορισμός του RED LED pin
int GREENPin = 11;        // Ορισμός του GREEN LED pin
int BLUEPin = 10;         // Ορισμός του BLUE LED pin
int brightness = 0;       // Ορισμός της φωτεινότητας του LED
int increment = 5;        // Ορισμός της αύξησης φωτεινότητας

void setup()
{
  pinMode(REDPin, OUTPUT);
  pinMode(GREENPin, OUTPUT);
  pinMode(BLUEPin, OUTPUT);
}

void loop()
{
  brightness = brightness + increment; // αύξηση φωτειν. για το επόμενο loop

  if (brightness <= 0 || brightness >= 255) // αντιστροφή για το fading
  {
    increment = -increment;
  }
  brightness = constrain(brightness, 0, 255);
  analogWrite(REDPin, brightness);
  analogWrite(GREENPin, brightness);
  analogWrite(BLUEPin, brightness);
  delay(20); // αναμονή 20 milliseconds για παρατήρηση του dimming εφέ!
}
```

♦ Μετά την εγγραφή του προγράμματος:

- α) Ελέγχουμε & μεταφράζουμε σε γλώσσα μηχανής πατώντας το κουμπί **“Verify”**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: **“Done compiling”**
- β) Όταν δεν υπάρχει κάποιο συντακτικό σφάλμα στο πρόγραμμα, μεταφέρουμε το πρόγραμμα στον μικροεπεξεργαστή πατώντας το κουμπί **“Upload”**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: **“Done Uploading”**.
- γ) Μετά από κάποια δευτερόλεπτα και αφού τελειώσει η εκτέλεση της μεταφοράς, θα δούμε ότι η ένταση των τριών χρωμάτων του RGB Led θα αρχίσει να αναβοσβήνει.
- δ) Δείτε το κάθε ένα Led ξεχωριστά!

Ασκήσεις 8ης Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

- Σχεδιάστε το διάγραμμα ροής και το γράψτε πρόγραμμα με το οποίο, γυρνώντας το ποτενσιόμετρο, που βρίσκεται στην αναλογική είσοδο “A0” θα αλλάζει η φωτεινότητα του κόκκινου Led που βρίσκεται στην ψηφιακή έξοδο “5”.

Βοήθεια: Κάντε χρήση της συνάρτησης “map()”.

Εργαστήριο 9^ο – Δομές ελέγχου στον Arduino II & Είσοδοι/Έξοδοι

Περίγραμμα Εργαστηριακής Άσκησης

1. Εισαγωγή
2. Ορίσματα & Βιβλιοθήκες
3. Τελεστές λογικών πράξεων
4. Αυξήσεις & Μειώσεις μεταβλητών (++ / --)
5. Δομή ελέγχου “for”
6. Συναρτήσεις συχνότητων
7. Παραδείγματα
 - Παράδειγμα 1^ο – Παραγωγή μελωδίας
 - Παράδειγμα 2^ο – Ένδειξη Θερμοκρασίας
8. Ασκήσεις 9ης Εργαστηριακής Άσκησης

1. Εισαγωγή

Στο μάθημα αυτό θα δούμε ορίσματα σταθερών τιμών, αριθμών κινητής υποδιαστολής, πινάκων, όπως επίσης και την δυνατότητα χρήσης εξωτερικών βιβλιοθηκών. Στην συνέχεια, θα δούμε τους βασικούς τελεστές λογικών πράξεων, την δυνατότητα αύξησης ή μείωσης τιμών μεταβλητών και τη λειτουργία και τη χρήση της δομής ελέγχου “for”. Τέλος θα χρησιμοποιήσουμε τις συναρτήσεις παραγωγής συχνοτήτων.

2. Ορίσματα & Βιβλιοθήκες

α) #define

Η εντολή #define επιτρέπει στον προγραμματιστή να δώσει ένα όνομα σε μια σταθερή τιμή πριν το πρόγραμμα να μεταφραστεί σε γλώσσα μηχανής. Ο μεταφραστής θα αντικαταστήσει αναφορές σε αυτές τις σταθερές με την καθορισμένη τιμή στο χρόνο που γίνεται η μετάφραση. Το σύμβολο “#” είναι απαραίτητο για την εκτέλεση της εντολής, ενώ το σύμβολο “;” δε χρησιμοποιείται στο τέλος της.

➤ Σύνταξη

```
#define constantName value
// Όπου constantName μια σταθερά
// Όπου value η τιμή που παίρνει η σταθερά
```

❖ Παράδειγμα (#define)

```
#define ledPin 3
// Ο μεταφραστής θα αντικαταστήσει οποιαδήποτε αναφορά της
// σταθεράς ledPin με τη τιμή 3, όταν γίνει η μετάφραση.
```

β) #include

Η εντολή #include χρησιμοποιείται προκειμένου να ενσωματώσει εξωτερικές βιβλιοθήκες στο Sketch. Έτσι, είναι εφικτή η πρόσβαση σε μια μεγάλη ομάδα βιβλιοθηκών από την γλώσσα “C”. Το σύμβολο “;” δε χρησιμοποιείται στο τέλος της εντολής.

❖ Παράδειγματα (#include)

```
/* Standard input/output definitions */
#include <stdio.h>
/* String function ..... */
#include <string.h>
/* File control ..... */
#include <fcntl.h>
/* Error number ..... */
#include <errno.h>
```

γ) float

Τύπος δεδομένων για αριθμούς που έχουν δεκαδικό ψηφίο. Οι αριθμοί κινητής υποδιαστολής χρησιμοποιούνται συχνά για να προσεγγίσουν αναλογικές και ψηφιακές τιμές επειδή έχουν μεγαλύτερη ανάλυση από τους ακέραιους. Οι αριθμοί αυτοί μπορούν να είναι μεγάλοι έως 3.4028235E+38 και μικροί έως -3.4028235E+38. Αποθηκεύονται ως 32 bits (4 bytes) πληροφορίας.

Η χρήση των αριθμών κινητής υποδιαστολής σε μαθηματικούς υπολογισμούς είναι πολύ πιο αργή σε σχέση με υπολογισμούς ακέραιων αριθμών, έτσι θα πρέπει να αποφευχθεί όπου είναι δυνατόν. Για παράδειγμα, εάν ένας βρόχος πρέπει να τρέχει στη μέγιστη ταχύτητα για μια κρίσιμη λειτουργία χρονισμού, επιδιώκετε να μετατραπούν υπολογισμοί αριθμών κινητής υποδιαστολής σε υπολογισμούς ακέραιων για την αύξηση της ταχύτητας.

➤ Σύνταξη

```
float var = val;
// Όπου var το όνομα της μεταβλητής και val η τιμή της μεταβλητής.
```

❖ Παράδειγμα (float)

```

int x;
int y;
float z;
x = 1;
y = x / 2;
// ο y έχει την τιμή 0 καθώς οι ακέραιοι δεν συγκρατούν κλάσματα
z = (float)x / 2.0;
// ο z έχει την τιμή 0.5 (πρέπει να χρησιμοποιηθεί 2.0, αντί για 2)

```

δ) Συναρτήσεις - υπορουτίνες

Μια συνάρτηση ή υπορουτίνα είναι μια ομάδα εντολών η οποία λαμβάνει κάποια δεδομένα σαν είσοδο “arguments” και επιστρέφει (αφού εκτελεστούν οι πράξεις/εντολές που βρίσκονται εντός της ομάδας εντολών “function body”), κάποιο αποτέλεσμα “returntype”. Οι υπορουτίνες γράφονται εκτός ορίων του κυρίως loop() και είναι ανεξάρτητες.

➤ Σύνταξη

```

returntype functionName( arguments ){
    // function body
    return returntype;
}

```

❖ Παράδειγμα

```

void setup()
{
    Serial.begin(9600);    // Προετοιμασία σειριακής πόρτας
}

void loop()
{
    int intAns;
    intAns=mySquare(5);    // Κλήση συνάρτησης/υπορουτίνας "mySquare"
    printAns(intAns);      // Κλήση συνάρτησης/υπορουτίνας "printAns"
    delay(500);
}

void printAns(int intTmp)    // Ορισμός συνάρτησης/υπορουτίνας "printAns"
{                            // Η υπορουτίνα δεν επιστρέφει κάτι: "void"
    Serial.print("Square of 5 is: ");
    Serial.println(intTmp);
}

int mySquare(int intInput)    // Ορισμός συνάρτησης/υπορουτίνας "mySquare"
{                            // Η υπορουτίνα επιστρέφει κάτι!
    int intSquare;
    intSquare = intInput * intInput;
    return intSquare;        // Η υπορουτίνα επιστρέφει το... "intSquare"
}

```

ε) Πίνακες

Ένας πίνακας είναι μια συλλογή από παρόμοια στοιχεία. Είναι μια συλλογή από μεταβλητές (στοιχεία) στις οποίες είναι δυνατή η πρόσβαση με ένα δείκτη μέτρησης. Όλες οι μεταβλητές ενός οποιοδήποτε πίνακα πρέπει να είναι του ίδιου τύπου δεδομένων για παράδειγμα int ή float.

- **Δήλωση Πίνακα**

Όλες οι παρακάτω δηλώσεις ισχύουν για τη δημιουργία πίνακα:

```
1. int myInts[6];
2. int myPins[] = {2, 4, 8, 3, 6};
3. int mySensVals[6] = {2, 4, -8, 3, 2};
```

1. Είναι δυνατή η δήλωση ενός πίνακα (myInts[]) χωρίς να γίνεται αρχικοποίηση του.
2. Γίνεται δήλωση του πίνακα myPins χωρίς να επιλέγεται ένα συγκεκριμένο μέγεθος. Ο μεταγλωττιστής μετράει τα στοιχεία και δημιουργεί ένα πίνακα κατάλληλου μεγέθους.
3. Είναι δυνατό να γίνει και η αρχικοποίηση και η επιλογή του μεγέθους του πίνακα (mySensVals[]).

- **Πρόσβαση πίνακα**

Το πρώτο στοιχείο ενός πίνακα έχει δείκτη 0. Για τον πίνακα mySensVals[] της δήλωσης 3 που αναφέρθηκε προηγουμένως:

```
// Το πρώτο στοιχείο του με δείκτη 0 έχει τιμή:
mySensVals[0] == 2
// Το δεύτερο έχει δείκτη 1 και τιμή:
mySensVals[1] == 4
// Ομοίως και για τα επόμενα...
```

Οπότε για ένα πίνακα με 10 στοιχεία ο δείκτης 9 είναι το τελευταίο στοιχείο (μεταβλητή)!

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
// myArray[9] έχει την τιμή 11
// myArray[10] δεν ισχύει και είναι πιθανό να περιέχει τυχαία πληροφορία!
```

Αν χρησιμοποιηθεί ένας δείκτης μεγαλύτερος από το δηλωμένο μέγεθος του πίνακα θα διαβάσει δεδομένα από μνήμη η οποία χρησιμοποιείται για άλλους λόγους. Το γεγονός αυτό θα αποφέρει όχι ορθά δεδομένα. Η γραφή σε τυχαίες τοποθεσίες μνήμης μπορεί να οδηγήσει σε άσχημα αποτελέσματα όπως πάγωμα (κρυστάρισμα) του υπολογιστή και δυσλειτουργία του προγράμματος.

Ο μεταγλωττιστής της C compiler δεν ελέγχει για να δει αν η πρόσβαση στον πίνακα είναι μέσα στα νόμιμα όρια του μεγέθους του πίνακα που έχει δηλωθεί.

- **Ανάθεση τιμής σε ένα πίνακα**

```
mySensVals[0] = 10; // Το στοιχείο 0 του πίνακα έχει τιμή 10
```

- **Ανάκτηση τιμής από ένα πίνακα**

```
x = mySensVals[4]; // Το x έχει την τιμή του στοιχείου 4 του πίνακα
```

- **Πίνακες και η FOR εντολή**

Οι πίνακες χειρίζονται μέσα σε for βρόχους, όπου ο μετρητής του βρόχου χρησιμοποιείται σαν δείκτης για το κάθε στοιχείο του πίνακα. Στο παρακάτω παράδειγμα τα δεδομένα ενός πίνακα τυπώνονται στη σειριακή πόρτα:

```
int i;
for (i = 0; i < 5; i = i + 1) {
    Serial.println(myPins[i]);
}
```

3. Τελεστές λογικών πράξεων

Οι κυριότεροι τελεστές λογικών πράξεων (Boolean) που χρησιμοποιούνται σε δομές ελέγχου είναι οι παρακάτω:

α) && (λογικό and)

Είναι αληθής μόνο όταν και τα δύο ορίσματα της δομής ελέγχου είναι αληθείς.

❖ Παράδειγμα (&&)

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) {  
    // διαβάζει δύο διακόπτες  
}  
// Είναι αληθής μόνο όταν και οι δύο είσοδοι έχουν την τιμή HIGH.
```

β) || (λογικό or)

Είναι αληθής όταν ένας από τα δύο ορίσματα είναι αληθής.

❖ Παράδειγμα (||)

```
if (x > 0 || y > 0) {  
    // ...  
}  
// Είναι αληθής αν είτε ο x είτε ο y είναι μεγαλύτερος του 0.
```

γ) ! (λογικό not)

Αληθής αν το όρισμα δεν είναι αληθής.

❖ Παράδειγμα (!)

```
if (!x) {  
    // ...  
}  
// Είναι αληθής αν το x είναι ψευδής. (π.χ. αν το x ισούται με 0).
```

4. Αυξήσεις & Μειώσεις μεταβλητών (++ / --)

Αυξάνει (++) ή μειώνει (--) μια μεταβλητή.

➤ Σύνταξη

```
x++;  
// αυξάνει το x κατά μια τιμή και επιστρέφει την παλιά τιμή του x  
++x;  
// αυξάνει το x κατά μια τιμή και επιστρέφει τη νέα τιμή του x  
x--;  
// μειώνει το x κατά μια τιμή και επιστρέφει την παλιά τιμή του x  
--x;  
// μειώνει το x κατά μια τιμή και επιστρέφει τη νέα τιμή του x  
// Όπου x: ακέραιος int ή long.
```

❖ Παραδείγματα (++ / --)

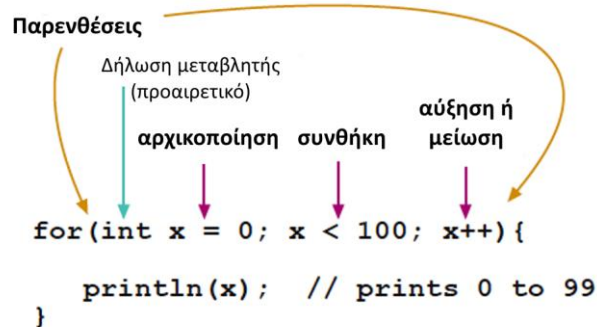
```
x = 2;  
y = ++x;    // ο x έχει τώρα την τιμή 3, ενώ και ο y έχει την 3  
y = x--;    // ο x έχει τώρα την τιμή 2, ενώ και ο y έχει την 3
```

5. Δομή ελέγχου “for”

Η δομή ελέγχου “for” είναι χρήσιμη για κάθε επαναλαμβανόμενη διαδικασία. Χρησιμοποιεί ένα καταμετρητή (αύξησης ή μείωσης) για να αυξάνει ή να μειώνει και να τελειώνει το βρόχο (loop). Υπάρχουν τρία τμήματα για τη δομή “for” (Εικόνα 27).

➤ Σύνταξη

```
for (αρχικοποίηση; συνθήκη; αύξηση ή μείωση) {  
  //ενέργεια;  
}
```



Εικόνα 27. Τμήματα δομής “for”

Η αρχικοποίηση συμβαίνει στην αρχή και για μια φορά. Κάθε φορά μέσα στο βρόχο:

- Ελέγχεται η συνθήκη και εφόσον είναι αληθής:
 - I. Εκτελείται η αύξηση ή η μείωση και το υποπρόγραμμα
 - II. Όταν η συνθήκη γίνει ψευδής, ο βρόχος τελειώνει

❖ Παράδειγμα (for)

```
/* Αύξηση της φωτεινότητας σε ένα LED χρησιμοποιώντας ένα PWM ακροδέκτη */  
int PWMpin = 10; // Ορισμός ονόματος LED που βρίσκεται στον ακροδέκτη 10  
void setup()  
{  
  // Δε χρειάζεται η setup  
}  
  
void loop()  
{  
  for (int i=0; i <= 255; i++){  
    analogWrite(PWMpin, i); // Αλλαγή της φωτεινότητας του LED  
    delay(10);  
  }  
}
```

❖ Παράδειγμα (for)

```
void setup(){  
  // Δε χρειάζεται η setup  
}  
void loop()  
{  
  for(int x = 2; x < 100; x = x * 1.5){  
    Serial.println(x);  
  }  
}  
// Αποτέλεσμα: 2,3,4,6,9,13,19,28,42,63,94
```

6. Συναρτήσεις συχνότητων

α) tone()

Περιγραφή

Παράγει τετραγωνική παλμοσειρά συγκεκριμένης συχνότητας και 50% του κύκλου λειτουργίας (duty cycle) σε ένα ακροδέκτη. Η χρονική του διάρκεια του μπορεί να οριστεί, αλλιώς η παλμοσειρά συνεχίζει μέχρι να κληθεί η συνάρτηση noTone(). Ο ακροδέκτης μπορεί να συνδεθεί σε ένα πιεζοηλεκτρικό κουδούνι (piezo buzzer) ή σε ηχείο για να παίξει τόνους.

Μόνο ένας τόνος μπορεί να παράγεται σε μια χρονική στιγμή. Αν ένας τόνος έχει ξεκινήσει να παίζει σε διαφορετικό ακροδέκτη η κλήση της συνάρτησης tone() δε θα έχει κανένα αποτέλεσμα. Αν ο τόνος παίζει ήδη στον ίδιο ακροδέκτη η κλήση της συνάρτησης θα ρυθμίσει τη συχνότητα του.

➤ Σύνταξη

```
tone(pin, frequency)
tone(pin, frequency, duration)
// Όπου pin, ο ακροδέκτης στον οποίο να παραχθεί ο τόνος
// Όπου frequency, η συχνότητα του τόνου σε hertz
// Όπου duration, η διάρκεια του τόνου σε milliseconds (μη υποχρεωτικό)
```

β) noTone()

Σταματά την παραγωγή της παλμοσειράς η οποία ξεκίνησε από τη συνάρτηση tone(). Δεν έχει κανένα αποτέλεσμα αν δεν παράγεται κάποιος τόνος τη στιγμή αυτή.

➤ Σύνταξη

```
noTone(pin)
// όπου pin, ο ακροδέκτης στον οποίο θα σταματήσει ο παραγόμενος τόνος
```

7. Παραδείγματα:

Παράδειγμα 1^ο – Παραγωγή μελωδίας

Ανοίγουμε με διπλό κλικ την εφαρμογή Arduino. Ανοίγουμε το παράδειγμα “toneMelody” από την διαδρομή: File→Examples→Digital→toneMelody.

```
#include "pitches.h"
// Νότες για την μελωδία:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4};
// Διάρκεια νότας: 4 = τέταρτο, 8 = όγδοο, κλπ:
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4 };
#define tonePin 6
void setup() {
  // Επανάληψη των νοτών της μελωδίας:
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    // Υπολογισμός της διάρκειας της κάθε νότας
    // Χωρίζουμε ένα δευτερόλεπτο ανάλογα με την διάρκεια της νότας
    // π.χ. τέταρτο νότας = 1000 / 4, όγδοο νότας = 1000/8, κλπ.
    int noteDuration = 1000/noteDurations[thisNote];
    tone(tonePin, melody[thisNote], noteDuration);
    // Για να ξεχωρίζουμε τις νότες, ορίζουμε μια καθυστέρηση μεταξύ τους
    // Η καθυστέρηση είναι + 30% η οποία φαίνεται να δουλεύει σωστά:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
```

```
// Σταματάει η συνάρτηση "tone":
noTone(tonePin);
}

void loop() {
// Δε χρειάζεται να επαναλαμβάνεται η μελωδία.
}
```

♦ Μετά το άνοιγμα του προγράμματος:

α) Προσθέτουμε κι αντικαθιστούμε τα υπογραμμισμένα σημεία:

- Προσθέτουμε την γραμμή **"#define tonePin 6"**, πριν από την διαδικασία "setup"
- Αλλάζουμε στις συναρτήσεις "tone()" και "noTone()", τον αριθμό του ακροδέκτη με την σταθερά "tonePin"

β) Ελέγχουμε & μεταφράζουμε σε γλώσσα μηχανής πατώντας το κουμπί **"Verify"**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: "Done compiling"

γ) Όταν δεν υπάρχει κάποιο συντακτικό σφάλμα στο πρόγραμμα, μεταφέρουμε το πρόγραμμα στον μικροεπεξεργαστή πατώντας το κουμπί **"Upload"**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: "Done Uploading".

δ) Ακούμε τη μελωδία από το ηχείο!

Παράδειγμα 2^ο – Ένδειξη Θερμοκρασίας

Ανοίγουμε με διπλό κλικ την εφαρμογή Arduino. Γράφουμε τον παρακάτω κώδικα:

```
/* Είσοδος: τάση από Thermistor,
 * Έξοδος: Θερμοκρασία σε βαθμούς Κελσίου
 * Απαραίτητο: include <math.h>
 * Χρησιμοποιεί την εξίσωση Steinhart-Hart για Thermistor:
 * Θερμοκρασία σε βαθμούς Κέλβιν = 1 / {A + B[ln(R)] + C[ln(R)]3}
 * Όπου A = 0.001129148, B = 0.000234125 και C = 8.76741E-08
 */
#define thermSensor A1 // Αναλογικός αισθητήρας @ A1
#define thermRes aaa // Αντίσταση Thermistor @ 25oC
void setup()
{
// start serial port at 9600 bps:
Serial.begin(9600);
}
void loop()
{

// Διάβασμα της αναλογικής εισόδου
float readTemp = analogRead(thermSensor);
Serial.print("Analog Read = ");
Serial.println(readTemp); // Εκτύπωση τιμής readTemp

// Υπολογισμός τάσης από την αναλογική είσοδο
float voltage = (readTemp / 1024) * 5.0;
Serial.print("Voltage = "); // Εκτύπωση τιμής Voltage
Serial.println(voltage);
```

```
// Υπολογισμός της αντίστασης του Thermistor από την τάση
float resistance=((5.0 * thermRes / voltage) - thermRes);
Serial.print("Resistance = ");          // Εκτύπωση τιμής Resistance
Serial.println(resistance);

// Υπολογισμός της θερμοκρασίας σε Kelvin, από την αντίσταση
float rescubed = log(resistance);
rescubed = rescubed * rescubed * rescubed;
float Kelvin = 1.0 / ((0.001129148 + (0.000234125 * log(resistance))) +
(0.0000000876741 * (rescubed)));
Serial.print("Kelvin = ");              // Εκτύπωση τιμής Kelvin
Serial.println(Kelvin);

float Celsius = Kelvin - 273.15;        // Μετατροπή Kelvin σε Celsius

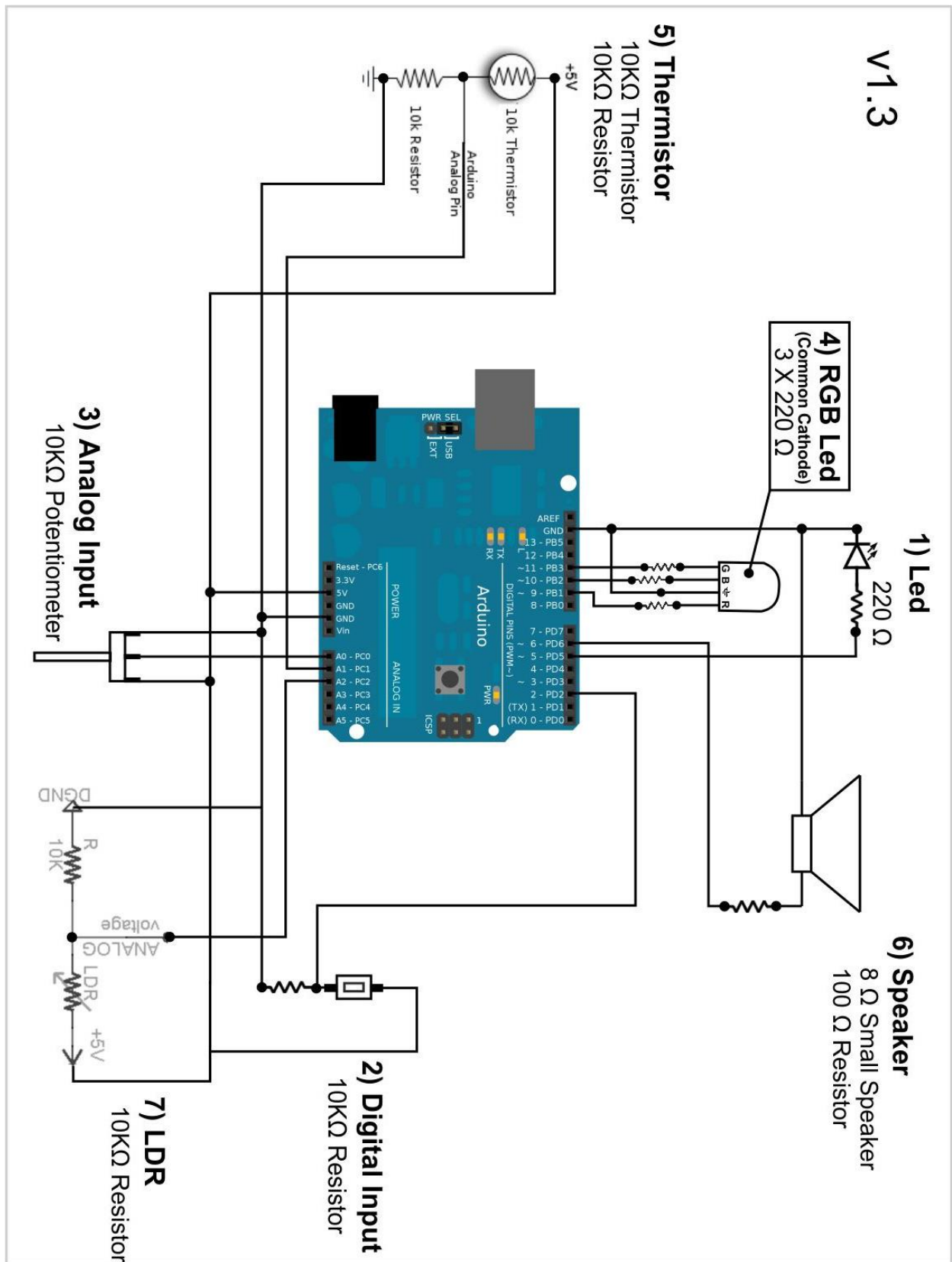
Serial.print("Temp = ");
Serial.println(Celsius);                // Εκτύπωση τιμής Celsius
Serial.println("~~~~~");
delay(1000);
}
```

♦ Μετά την εγγραφή του προγράμματος:

- α) Αντικαθιστούμε στο πρόγραμμα, όπου “aaa” (υπογραμμισμένο στο πρόγραμμα), την αντίσταση από τον παρακάτω πίνακα ανάλογα με τον υπολογιστή στον οποίο βρίσκεστε:

Υπολογιστής	COM Port	aaa(Ω)
PC1	3	5100
PC2	5	107500
PC3		
PC4	3	101500
PC5	5	4650
PC6	4	4530
PC7	5	90500
PC8	4	109500
PC9	5	5100

- β) Ελέγχουμε & μεταφράζουμε σε γλώσσα μηχανής πατώντας το κουμπί **“Verify”**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: “Done compiling”
- γ) Όταν δεν υπάρχει κάποιο συντακτικό σφάλμα στο πρόγραμμα, μεταφέρουμε το πρόγραμμα στον μικροεπεξεργαστή πατώντας το κουμπί **“Upload”**, που βρίσκεται στην εργαλειοθήκη. Εφόσον δεν υπάρχουν λάθη, στην περιοχή μηνυμάτων θα βγει το μήνυμα: “Done Uploading”.
- δ) Προκειμένου να παρακολουθήσουμε τα δεδομένα που στέλνονται από την πλακέτα ανάπτυξης στον υπολογιστή, μέσω της σειριακής θύρας, πατάμε το κουμπί **“Serial Monitor”** που βρίσκεται τέρμα δεξιά στη γραμμή εργαλείων.
- ε) Αγγίζοντας ελαφρά το thermistor, θα πρέπει να διαβάζουμε στο παράθυρο του **“Serial Monitor”**, τις ενδείξεις του (Αναλογική είσοδο, τάση, αντίσταση, θερμοκρασία σε Kelvin και σε Celsius).



Εικόνα 28. Σχεδιάγραμμα συνδέσεων για στην πλακέτα ανάπτυξης Arduino.

Ασκήσεις 9ης Εργαστηριακής Άσκησης:

Όνοματεπώνυμο	
Τμήμα	
Ημερομηνία Διεξαγωγής	

- Σχεδιάστε το διάγραμμα ροής και το γράψτε πρόγραμμα με το οποίο, όταν η θερμοκρασία του thermistor (Analog Input 1) ξεπεράσει τους 30°C, τότε να ανάβει το κόκκινο LED (Digital Pin 5) και να ακούγεται ένας ήχος (Digital Pin 2).

Βοήθεια: Εκτυπώστε στην σειριακή θύρα τις τιμές κρίσιμων μεταβλητών για να ελέγξετε τη σωστή λειτουργία του προγράμματος.

