2. Δήλωση μεταβλητών

Οι μεταβλητές είναι τα βασικά στοιχεία που διαπραγματεύεται ένα πρόγραμμα. Περιέχουν τιμές ή δεδομένα και μπορεί να έχουν μια σταθερή τιμή ή το περιεχόμενό τους να εξαρτάται από τις διεργασίες. Ανάλογα με τον τρόπο που θα δηλωθούν μπορούν να δεσμεύσουν μια μόνιμη ή προσωρινή περιοχή στη μνήμη ή να δεσμεύσουν έναν καταχωρητή του μικροελεγκτή. Δηλώνονται είτε στην αρχή του κώδικα μαζί με τις εντολές **#include**, είτε μέσα σε κάποιο μπλοκ ή ρουτίνα και αν απαιτείται είναι δυνατό να λάβουν αρχική τιμή:

```
#include <avr/io.h>
#include <util/delay.h>
unsigned char i=0xA0;
```

Με την παραπάνω εντολή δηλώνεται στην αρχή του κώδικα η μεταβλητή **i** με αρχική τιμή **0xA0**. Οι μεταβλητές που δηλώνονται στην αρχή του κώδικα δεσμεύουν μόνιμες σταθερές θέσεις στη μνήμη και είναι ενεργές για όλη τη διάρκεια λειτουργίας του προγράμματος. Είναι ορατές από όλες τις ρουτίνες και το περιεχόμενό τους μπορεί να τροποποιηθεί από αυτές οποτεδήποτε.

Αν αντί για θέση στη μνήμη απαιτηθεί να δεσμεύσουν συγκεκριμένο καταχωρητή πρέπει να δηλωθούν ως **register**:

```
register unsigned char i asm("r3");
```

Με την παραπάνω εντολή δηλώνεται η μεταβλητή με ονομασία **i** στον καταχωρητή **r3** του μικροελεγκτή. Οι καταχωρητές που είναι διαθέσιμοι για αυτή τη λειτουργία είναι οι **r2** ως **r15**, αλλά οι **r8** ως **r15** κάποιες φορές είναι απαραίτητοι για χρήση από τον compiler. Επίσης υπάρχει πιθανότητα κάποιος καταχωρητής να χρησιμοποιείται από βιβλιοθήκες που είναι φορτωμένες. Για τους παραπάνω λόγους χρειάζεται προσοχή η δήλωση μεταβλητής σε καταχωρητή.

Όταν κάποια ρουτίνα χρησιμοποιεί μια μεταβλητή και συμβεί διακοπή, δεν είναι δυνατό να μεταβληθεί το περιεχόμενο της μεταβλητής από ρουτίνα της διακοπής. Αν όμως οι ανάγκες του προγράμματος απαιτούν να μεταβληθεί το περιεχόμενό της, τότε πρέπει να δηλωθεί ως **volatile**:

```
volatile unsigned char i;
```

Όταν οι μεταβλητές δηλώνονται μέσα σε κάποια διεργασία, είτε αυτή είναι συνάρτηση, είτε ρουτίνα ή μπλοκ, είναι τοπικές μεταβλητές και είναι προσωρινά ενεργές, για όσο διαρκεί η συγκεκριμένη διεργασία. Δεσμεύουν μια θέση στη μνήμη η οποία απελευθερώνεται μετά

το τέλος της διεργασίας. Η χρήση τοπικών μεταβλητών, όπου είναι δυνατόν, είναι προτιμότερη γιατί ο compiler δημιουργεί κώδικα με λιγότερες απαιτήσεις σε μνήμη.

```
for (unsigned char i1 = 0; i1 < 16; i1++) { /* Your code here */ }
```

Με την παραπάνω εντολή δηλώνεται η μεταβλητή **i1** στη ρουτίνα μιας εντολής **for**. Η μεταβλητή αυτή είναι ενεργή για όσο διαρκεί η ρουτίνα και στη συνέχεια απελευθερώνεται ο χώρος που καταλαμβάνει καθώς και η ονομασία της.

Μια τοπική μεταβλητή μπορεί να παραμείνει ενεργή και να διατηρεί το περιεχόμενό της αν δηλωθεί ως **static**. Η μεταβλητή αυτή είναι αόρατη από το υπόλοιπο πρόγραμμα και το όνομά της μπορεί να χρησιμοποιηθεί και σε άλλες διεργασίες αλλά σε διαφορετικά μπλοκ ή ρουτίνες χωρίς να προκύψει σύγχυση.

```
static unsigned char i;
```

Οι μεταβλητές είναι δυνατό να εμφανίζονται και ως πίνακες οι οποίοι περιέχουν δεδομένα. Οι πίνακες όταν δηλώνονται γράφεται και η διάστασή τους, διαφορετικά ο compiler θα τους αποδώσει διάσταση 1. Όταν αρχικοποιείται ένας πίνακας κατά τη δήλωσή του, δε χρειάζεται να γραφεί η διάστασή του αλλά λαμβάνεται ως διάσταση το πλήθος των δεδομένων της αρχικοποίησης. Σε έναν πίνακα έχουμε τη δυνατότητα να εισάγουμε λιγότερα ή περισσότερα δεδομένα από τη διάστασή του. Σε περίπτωση που εισάγουμε λιγότερα, στις θέσεις που δεν έχουμε παρέμβει παραμένουν τα προηγούμενα δεδομένα ή μηδενικά αν δεν υπήρχαν δεδομένα. Σε περίπτωση που εισάγουμε περισσότερα, δίνοντας δείκτη μεγαλύτερο από τη διάσταση του πίνακα, τότε τα δεδομένα που εισάγουμε καταλαμβάνουν θέσεις στη μνήμη που ενδέχεται να είναι δεσμευμένες από δεδομένα του προγράμματος, με αποτέλεσμα να τα τροποποιήσουν και να δημιουργηθούν προβλήματα στη λειτουργία του προγράμματος.

Πιο συνηθισμένη μορφή είναι οι μονοδιάστατοι πίνακες οι οποίοι μπορούν να αποτελέσουν και μια σειρά δεδομένων (string):

```
unsigned char name[] =
     {'D','i','m','i','t','r','i','o','s',' ','P','o','r','l','i','d','a','s'};
     // 18 characters string with initial values
unsigned char name[10];                  // 10 characters string
```

Οι πολυδιάστατοι πίνακας δηλώνονται ως εξής:

```
int mat[2][2]={{3,7},{-6,4}};            // 2x2 matrix with initial values
int mat[10][4];                          // 10x4 matrix
```

Το μέγεθος των μεταβλητών μπορεί να είναι 8bit, 16bit, 32bit ή 64bit και μπορεί να είναι προσημασμένες ή μη. Όταν είναι μη προσημασμένες χαρακτηρίζονται ως **unsigned**. Οι προσημασμένες μεταβλητές έχουν τη μορφή του συμπληρώματος ως προς 2. Οι μεταβλητές ανάλογα με το μέγεθός τους δηλώνονται ως: **char** (8bit), **int** (16bit ή 32bit, ανάλογα με τον

επεξεργαστή, στους AVR8 είναι 16bit), **short** (16bit), **long** (32bit). Το Atmel Studio 7.0 υποστηρίζει επίσης τη βιβλιοθήκη **stdint.h** για μεταβλητές συγκεκριμένου πλάτους με ονοματολογία **int8_t**, **int16_t**, **int32_t**, **in64_t** για προσημασμένες μεταβλητές 8, 16, 32, 64 bit και **uint8_t**, **uint16_t**, **uint32_t**, **uin64_t** αντίστοιχα, για μη προσημασμένες, καθώς και ολόκληρο το σετ της ίδιας βιβλιοθήκης για μεταβλητές και συναρτήσεις (http://www.cplusplus.com/reference/cstdint/).

3. Αριθμητικοί και λογικοί τελεστές

Οι τελεστές χρησιμοποιούνται για απόδοση τιμής, σύγκριση, αριθμητικές και λογικές πράξεις. Οι κυριότεροι τελεστές που χρησιμοποιούμε είναι οι εξής:

=, -, +, *, /, %, <, >, !, ~, &, |, ^, +=, -=, *=, /=, %=, ==, >=, <=, --, ++, //, <<, >>, !=, &&, ||

Οι **αριθμητικοί τελεστές =**, **-**, **+**, *****, **/**, **%** χρησιμοποιούνται για αριθμητικές πράξεις μεταξύ τελεστέων (καταχωρητών – μεταβλητών – σταθερών). Με τη σειρά που αναφέρονται είναι οι αριθμητικές πράξεις ισότητας, αφαίρεσης, πρόσθεσης, πολλαπλασιασμού, διαίρεσης και υπόλοιπου διαίρεσης.

```
i1 = 0;                    // i1 will be i1=0
i1 = i2 + i3;              // i2=45, i3=10, then i1 will be i1=55
i1 = i1 + 6;               // Initially i1=55, then i1 will be i1=61
i1 += 6;                   // Initially i1=55, then i1 will be i1=61
```

Στο πρώτο από τα παραπάνω παραδείγματα η μεταβλητή **i1** παίρνει τιμή **0**, στο δεύτερο η μεταβλητή **i1** παίρνει την τιμή του αθροίσματος των **i2** και **i3**, στο τρίτο και στο τέταρτο η **i1** παίρνει την τιμή που προκύπτει από το άθροισμα της τιμής που είχε αρχικά με το 6. Μολονότι δίνουν το ίδιο αποτέλεσμα και οι δύο τρόποι σύνταξης, ο τελευταίος είναι προτιμότερος γιατί μπορεί να βοηθήσει τον compiler να παράγει αποτελεσματικότερο κώδικα. Οι τελεστές **+=**, **-=**, ***=**, **/=, %=** ονομάζονται τελεστές αντικατάστασης.

Κατά τη διαίρεση ως αποτέλεσμα λαμβάνουμε μόνο το ακέραιο μέρος, για το υπόλοιπο πρέπει να εκτελέσουμε την πράξη του υπολοίπου διαίρεσης:

```
i1 = i2 / i3;              // i2=45, i3=10, then i1 will be i1=4
i1 = i2 % i3;              // i2=45, i3=10, then i1 will be i1=5
i1 %= 6;                   // Initially i1=55, then i1 will be i1=1
```

Στις αριθμητικές πράξεις μπορούμε να συμπεριλάβουμε τη μοναδιαία αύξηση **i1++** ή **++i1** και τη μοναδιαία ελάττωση **i1--** ή **--i1** όπου η μεταβλητή αυξάνει ή ελαττώνει την τιμή της αντίστοιχα. Η διαφορά της χρήσης του τελεστή αύξησης **++** ή ελάττωσης **- -** μετά ή πριν το όνομα της μεταβλητής είναι ότι, όταν βρίσκεται μετά, χρησιμοποιείται πρώτα η τρέχουσα τιμή της μεταβλητής στη συνθήκη που εμφανίζεται και μετά αυξάνει η μεταβλητή, ενώ όταν βρίσκεται πριν, αυξάνει πρώτα η μεταβλητή και μετά χρησιμοποιείται η τιμή της στη συνθήκη.

Οι **τελεστές σύγκρισης >**, **<**, **>=**, **<=,** χρησιμοποιούνται, όπως ορίζεται από το σύμβολο, για σύγκριση αριθμητικών τιμών μεταξύ τελεστέων (καταχωρητών – μεταβλητών – σταθερών) και επιστρέφουν Αληθές ή Ψευδές[1] στη συνθήκη που χρησιμοποιούνται:

```
if (i1 >= 4) { /* Your code here */ }
```

Στο παραπάνω παράδειγμα θα εκτελεστεί ο κώδικας που βρίσκεται μέσα στις αγκύλες αν η συνθήκη στην παρένθεση είναι Αληθής, δηλαδή η **i1** είναι μεγαλύτερη ή ίση του 4.

Με παρόμοιο τρόπο λειτουργούν **οι τελεστές σύγκρισης ==** και **!=** όπου ο πρώτος χρησιμοποιείται για το ίσο και ο δεύτερος για το διάφορο:

```
if (i1 == 4) { /* Your code here */ }
```

Στο παραπάνω παράδειγμα θα εκτελεστεί ο κώδικας που βρίσκεται μέσα στις αγκύλες αν η **i1** είναι ίση του 4.

Οι **λογικοί τελεστές &**, **|**, **^**, **~**, **<<**, **>>**, χρησιμοποιούνται για λογικές πράξεις με bit μεταξύ τελεστέων (καταχωρητών – μεταβλητών – σταθερών). Με τη σειρά που αναφέρονται είναι οι λογικές πράξεις AND, OR, XOR, συμπλήρωμα ως προς 1, αριστερή ολίσθηση και δεξιά ολίσθηση[2]:

```
i1 = i2 & i3;    // i2=0b00101101, i3=0b00001010, then i1 will be i1=0b00001000
i1 = i2 | i3;    // i2=0b00101101, i3=0b00001010, then i1 will be i1=0b00101111
i1 = i3 << 4;    // i3=0b00001010, then i1 will be i1=0b10100000
i1 = ~i3;        // i3=0b00001010, then i1 will be i1=0b11110101
```

Το συμπλήρωμα ως προς 1 μπορεί να χρησιμοποιηθεί και στον ίδιο τον τελεστή:

```
i1 = ~i1;        // Initially i1=0b00110110, then i1 will be i1=0b11001001
```

Στις λογικές πράξεις με bit μεταξύ τελεστέων μπορούμε να χρησιμοποιήσουμε επίσης τελεστές αντικατάστασης **&=**, **|=**, **^=**, **<<=**, **>>=**

```
i1 &= 0b00001010;       // Initially i1=0b00110110, then i1 will be i1=0b00000010
```

τους οποίους και σε αυτή την περίπτωση είναι προτιμότερο να χρησιμοποιούμε αντί της πιο ανεπτυγμένης μορφής:

```
i1 = (i1 & 0b00001010);
```

Οι **λογικοί τελεστές &&**, **||**, **!**, χρησιμοποιούνται για λογικές πράξεις μεταξύ των καταστάσεων Αληθούς ή Ψευδούς των συνθηκών κάποιου γενικότερου όρου. Με τη σειρά

---

[1] *Αληθές ή Ψευδές (True or False). Το Αληθές αντιστοιχεί σε οποιαδήποτε μη μηδενική τιμή (non-zero value), ενώ το Ψευδές αντιστοιχεί σε μηδέν (zero value) (0).*

[2] *Κατά την αριστερή ολίσθηση το LSB συμπληρώνεται με **0** και κάθε ολίσθηση ισοδυναμεί με πολλαπλασιασμό με 2. Κατά τη δεξιά ολίσθηση το MSB συμπληρώνεται με **0** αν η μεταβλητή είναι μη προσημασμένη. Αν είναι προσημασμένη, συμπληρώνεται με **1** αν είναι αρνητικός αριθμός (το MSB είναι **1** και ο αριθμός έχει τη μορφή συμπληρώματος ως προς 2) ενώ αν είναι θετικός αριθμός (το MSB είναι **0** και ο αριθμός έχει κανονική μορφή) με **0** και ισοδυναμεί με διαίρεση με 2.*

που αναφέρονται είναι οι λογικές πράξεις: AND, OR, NOT. Επιστρέφουν και αυτές με τη σειρά τους κατάσταση αληθούς ή ψευδούς.

```
if ((i1 == 0xA0) && (i2 == 0xB0) && (i3 == 0xC0)) { /* Your code here */ }
if (!((i1 == 0xA0) && (i2 == 0xB0) && (i3 == 0xC0))) { /* Your code here */ }
```

Στο πρώτο από τα παραπάνω παραδείγματα θα εκτελεστεί ο κώδικας που βρίσκεται μέσα στις αγκύλες αν ισχύουν και οι 3 συνθήκες της παρένθεσης, δηλαδή **i1=A0**, **i2=B0** και **i3=C0**. Αυτό ερμηνεύεται ως εξής: κάθε μια συνθήκη όταν ισχύει επιστρέφει Αληθές, το AND Αληθών είναι και αυτό Αληθές συνεπώς ο όρος της παρένθεσης στην **if** καθίσταται Αληθής και η εντολή εκτελείται. Στο δεύτερο παράδειγμα ο κώδικας θα εκτελείται πάντα εκτός από την περίπτωση που θα ισχύσουν οι τρεις συνθήκες, γιατί το λογικό αποτέλεσμα των τριών συνθηκών αντιστρέφεται.

4. Εντολές Embedded C for AVR MCUs

Το σύνολο των εντολών που χρησιμοποιούνται στο **Atmel Studio 7.0** για σύνταξη κώδικα σε γλώσσα C περιέχει, εκτός από τις εντολές της ANSI C και μια σειρά εντολών **Embedded C for AVR MCUs**, οι οποίες αναφέρονται σε λειτουργίες και καταχωρητές των μικροελεγκτών. Οι εντολές αυτές ενδέχεται να διαφοροποιούνται ανάλογα με τον μικροελεγκτή και τις δυνατότητές του. Για παράδειγμα, δεν είναι δυνατό να χρησιμοποιήσουμε εντολές που αφορούν καταχωρητές της **USART** σε μικροελεγκτές που δεν έχουν ενσωματωμένη θύρα, ο compiler θα βγάλει σφάλμα κατά τη μεταγλώττιση. Στα data sheet των μικροελεγκτών παρουσιάζονται οι εντολές που τους υποστηρίζουν καθώς και παραδείγματα σύνταξης. Επίσης υπάρχουν συναρτήσεις και ρουτίνες που καλούνται από βιβλιοθήκες που υπάρχουν ενσωματωμένες στο **Atmel Studio 7.0** εφόσον συμπεριλάβουμε το κατάλληλο header file στον κώδικα. Στη συνέχεια παρουσιάζονται παραδείγματα σύνταξης των εντολών που θα χρησιμοποιήσουμε στον κώδικα για τα πρώτα προγράμματα που θα συνθέσουμε.

```
DDRA = 0b10110110;          // port A 7, 5, 4, 2, 1 outputs, 6, 3, 0 inputs
```

Η εντολή **DDRx** Data Direction Register x (όπου x η αντίστοιχη θύρα [port] του μικροελεγκτή A, B, C, ….) μας δίνει πρόσβαση στον αντίστοιχο καταχωρητή του μικροελεγκτή, ο οποίος είναι ανάγνωσης/εγγραφής (R/W) και καθορίζει τη χρήση των ακροδεκτών (pin) του port ως είσοδο ή έξοδο. Κάθε pin προγραμματίζεται ξεχωριστά από τα υπόλοιπα έτσι ώστε να είναι δυνατό στο ίδιο port να έχουμε pin εισόδων και pin εξόδων. Με **1** προγραμματίζεται το pin ως έξοδος, ενώ με **0** ως είσοδος. Η αρχική κατάσταση, αμέσως μετά την επαναφορά από RESET, είναι όλα **0**, δηλαδή είσοδοι. Στο παράδειγμα γίνεται εγγραφή στον καταχωρητή και προγραμματίζονται τα pin 7, 5, 4, 2, 1 του port A ως έξοδοι και τα 6, 3, 0 ως είσοδοι.

```
PORTA = 0b11111111;         //enable pull up resistors to inputs, set 1 to outputs
```

Η εντολή **PORTx** (όπου x το αντίστοιχο port του μικροελεγκτή A, B, C, ….) μας δίνει πρόσβαση στον αντίστοιχο καταχωρητή του μικροελεγκτή, ο οποίος είναι ανάγνωσης/εγγραφής (R/W) και προσδίδει τιμή στο port αν είναι προγραμματισμένο ως έξοδος ή ενεργοποιεί την εσωτερική αντίσταση πρόσδεσης (pull up) αν είναι προγραμματισμένο ως είσοδος. Η αντίσταση πρόσδεσης μπορεί να είναι μόνο pull up και είναι της τάξης των 40kΩ. Αν δεν ενεργοποιηθεί η pull up τότε η είσοδος εμφανίζει υψηλή σύνθετη αντίσταση. Η αρχική κατάσταση, αμέσως μετά την επαναφορά από RESET, είναι όλα **0.** Στο παράδειγμα παραπάνω γίνεται εγγραφή στον καταχωρητή. Είναι δυνατόν να κάνουμε

εγγραφή σε καθορισμένα bit του καταχωρητή με τη σύνταξη **PORTxy** (όπου x το αντίστοιχο port του μικροελεγκτή A, B, C, …. και y το αντίστοιχο bit, αρχίζοντας από **0** για το LSB έως το **7** για το MSB):

```
PORTA = (1<<PORTA2);
              /* Initially PORTA=0b00000000, then i1 will be PORTA=0b00000100*/
PORTA = (3<<PORTA2);
              /* Initially PORTA=0b00000000, then i1 will be PORTA=0b00001100*/
```

Στο πρώτο από τα παραπάνω παραδείγματα εγγράφουμε **1** στο bit 2 του **PORTA**, ενώ στο δεύτερο παράδειγμα εγγράφουμε **3 (0b11)** στο ίδιο bit με αποτέλεσμα να επηρεάζεται και το αμέσως σημαντικότερο.

```
while(PINA == 0b11111110) { /* Your code here */ }
```

Ο **PINx** είναι ο καταχωρητής κατάστασης του port x (όπου x το αντίστοιχο port του μικροελεγκτή A, B, C, ….). Είναι καταχωρητής μόνο ανάγνωσης (R) και δηλώνει την κατάσταση των pin του port είτε αυτά είναι προγραμματισμένα ως είσοδοι είτε ως έξοδοι. Δεν μπορεί να γίνει απόδοση τιμής στον καταχωρητή και για αυτό το λόγο συναντάμε τον συμβολισμό του μέσα σε εντολές όπως η **while** ή η **if**, όπως στο παράδειγμα, όπου γίνεται ανάγνωση του καταχωρητή.

```
_delay_ms(10);              //Delay 10ms
```

Η εντολές *_delay_ms*() και *_delay_us*() χρησιμοποιούνται για καθυστέρηση στη ροή του προγράμματος σε ms ή μs αντίστοιχα ανάλογα με τον αριθμό της παρένθεσης. Στην πραγματικότητα πρόκειται για μια εξωτερική ρουτίνα η οποία καλείται ως συνάρτηση και προκειμένου να εκτελεστεί σωστά, πρέπει να έχει δηλωθεί η συχνότητα λειτουργίας του επεξεργαστή και να έχει οριστεί βελτιστοποίηση κώδικα (όπως αναφέρεται στο μάθημα 1). Για τη χρήση της χρειάζεται να συμπεριληφθεί το header file delay.h **(#include <util/delay.h>**).

```
if (bit_is_set(PINA,0)) { /* Your code here */ }
```

Η εντολή **bit_is_set()** χρησιμοποιείται προκειμένου να ανιχνεύσουμε αν κάποιο bit ενός καταχωρητή ή μιας μεταβλητής είναι **1**. Μέσα στην παρένθεση της εντολής βάζουμε το όνομα του καταχωρητή ή της μεταβλητής και μετά το κόμμα τον αριθμό του bit, αρχίζοντας από **0** για το LSB έως το MSB-1. Σε περίπτωση που το bit είναι **1** μας επιστρέφει αληθές στη συνθήκη της εντολής που είναι ενσωματωμένη η **bit_is_set()** και στο παράδειγμα που προηγείται θα εκτελεστεί ο κώδικας που βρίσκεται μέσα στις αγκύλες. Κατά τη σύνταξη του

κώδικα το Atmel Studio 7.0 θα μας υπογραμμίσει την εντολή ως συντακτικό λάθος, όμως ο compiler θα τη μεταγλωττίσει κανονικά χωρίς error, warning ή message.

Εκτός από τις εντολές για τους καταχωρητές και τις συναρτήσεις που εξετάσαμε στα προηγούμενα παραδείγματα, υπάρχουν αντίστοιχες εντολές για όλους τους καταχωρητές και τις λειτουργίες των μικροελεγκτών, τις οποίες θα αναφέρουμε με παραδείγματα καθώς θα συνθέτουμε προγράμματα που τις υποστηρίζουν.