A. Tips

1. **Τοποθέτηση σημαίας ή bit ενός καταχωρητή ή μεταβλητής**

```
err |= (1<<4);
```

Με την παραπάνω εντολή τοποθετείται (γίνεται **1**) το bit 4 της μεταβλητής[1] **err** ενώ **όλα τα υπόλοιπα bit μένουν ως έχουν**. Είναι ισοδύναμη με την εντολή: **err = err | 0b00010000;** με τη διαφορά ότι είναι συντομότερη κατά τον μεταγλωττισμό και συνεπώς εκτελείται γρηγορότερα. Αν θέλουμε να **μηδενίσουμε όλα τα υπόλοιπα bit** μπορούμε να χρησιμοποιήσουμε την εντολή:

```
err = (1<<4);
```

2. **Καθάρισμα σημαίας ή bit ενός καταχωρητή ή μεταβλητής**

```
err &= ~(1<<4);
```

Με την παραπάνω εντολή καθαρίζει (γίνεται **0**) το bit 4 της μεταβλητής[1] **err** ενώ **όλα τα υπόλοιπα bit μένουν ως έχουν**. Είναι ισοδύναμη με την εντολή: **err = err & 0b11101111;** με τη διαφορά ότι είναι συντομότερη κατά τον μεταγλωττισμό και συνεπώς εκτελείται γρηγορότερα.

3. **Σύγκριση μονοδιάστατων πινάκων (στο παράδειγμα 5 στοιχείων)**

```
#include <avr/io.h>
uint8_t l[5]=;
uint8_t m[5]=;
uint8_t err;
              /* Your code here */
err &= ~(1<<0);
for (uint8_t i=0; i<5; i++)
{
   if (l[i] != m[i])
   {
      err |= (1<<0);
      break;
   }
}
if (bit_is_set(err,0))
{
   /* Your condition here */
}
              /* Your code here */
```

Με τον παραπάνω κώδικα γίνεται σύγκριση μονοδιάστατων πινάκων (string). Αρχικά καθαρίζει το bit 0 της μεταβλητής **err** και στη συνέχεια ξεκινάει η σύγκριση των στοιχείων

[1] *Στο παράδειγμα η μεταβλητή είναι 8bit όμως ισχύει και για 16bit.*

ένα προς ένα. Όταν βρεθεί η πρώτη διαφορά τοποθετείται το bit 0 της μεταβλητής **err** και σταματάει η σύγκριση (ιδιαίτερα χρήσιμο για μεγάλους πίνακες). Με μια μικρή τροποποίηση μπορεί να γίνει αρίθμηση των διαφορετικών στοιχείων των πινάκων:

```
#include <avr/io.h>
uint8_t l[5]=;
uint8_t m[5]=;
uint8_t err;
              /* Your code here */
err = 0;
for (uint8_t i=0; i<5; i++)
{
   if (l[i] != m[i])
   err++;
}
```

4. Έξοδος από κατάσταση αναμονής σε περίπτωση αστοχίας

```
#include <avr/io.h>
uint8_t i;
              /* Your code here */
i=0;
while (bit_is_set(PINA,0))
{
   i++;
   _delay_us(5);
   if (i > 100)
   {
      /* Your condition here */
      break;
   }
}
```

Στον παραπάνω κώδικα το πρόγραμμα εξέρχεται από τον βρόχο της **while** ακόμα και αν δεν ικανοποιηθεί η συνθήκη της. Μπορεί να χρησιμοποιηθεί σε περιπτώσεις που δεν πραγματοποιείται απόκριση κάποιας συσκευής από αστοχία (στο παράδειγμα η συσκευή είναι συνδεδεμένη στο **PORTA0**). Ο χρόνος λήξης της αναμονής (time out) υπολογίζεται από τον αριθμό σύγκρισης της **i** στη συνθήκη της **if** και την καθυστέρηση στον βρόχο της **while** (πρέπει να είναι μεγαλύτερος από τον προβλεπόμενο χρόνο απόκρισης, προτείνεται 10πλάσιος).

5. Μετατροπή εύρους τιμών με διαίρεση

```
#include <avr/io.h>
uint8_t output[2];
uint8_t AdcOut;

output[0] = 0;                    // Reset register
output[1] = 0;
while (AdcOut > 50)
{
   AdcOut -= 51;
   output[0]++;
```

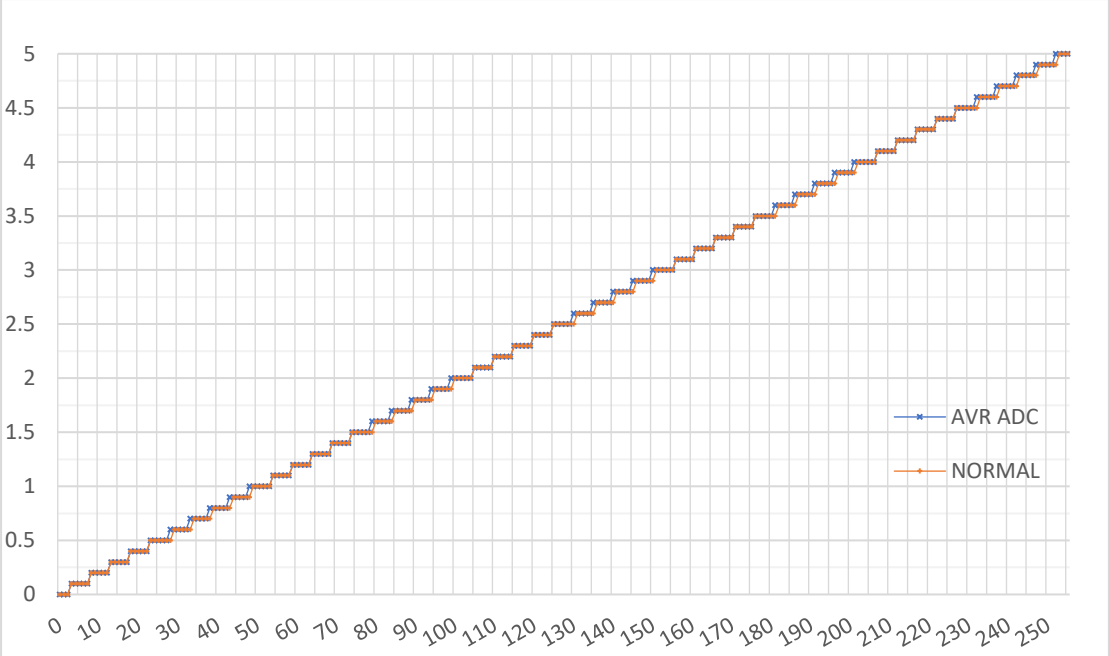
```
}

while (AdcOut > 4)
{
   AdcOut -= 5;
   output[1]++;
}

if (AdcOut > 2)
output[1]++;
if (output[1] > 9)
{
   output[0]++;
   output[1] = 0;
}
```

Με τον παραπάνω κώδικα μπορούμε να μετατρέψουμε το εύρος τιμών μιας μεταβλητής και να αποθηκεύσουμε το αποτέλεσμα σε έναν πίνακα. Η μετατροπή είναι χρήσιμη όταν χρησιμοποιούμε τον ADC του μικροελεγκτή για μέτρηση τάσεων από διάφορες διατάξεις ή αισθητήρες με αναλογική έξοδο. Στο παράδειγμα που παρουσιάζεται γίνεται μετατροπή του εύρους μιας 8bit μη προσημασμένης μεταβλητής **AdcOut** (0 – 255) σε τάση 0.0 – 5.0V όπου μονάδες και δέκατα αποθηκεύονται σε διαφορετικές μεταβλητές **output[0]**, **output[1]**. Για να γίνει η μετατροπή πρέπει να διαιρέσουμε την **AdcOut** με το 51. Η διαίρεση αυτή δεν είναι πάντα τέλεια και απαιτεί αρκετή μνήμη. Εναλλακτικά μπορούμε να κάνουμε τη διαίρεση με διαδοχικές αφαιρέσεις. Το υπόλοιπο της διαίρεσης μπορούμε να το κάνουμε στρογγυλοποίηση ή να το κρατήσουμε ως τιμή.

Αρχικά μηδενίζονται τα στοιχεία του πίνακα **output[]** που θα αποθηκευτεί το αποτέλεσμα και εκτελείται μια ρουτίνα που, όσο η μεταβλητή **AdcOut** είναι μεγαλύτερη από το 50, αφαιρεί συνεχώς από αυτήν το 51 και μετά από κάθε αφαίρεση αυξάνει την τιμή του **output[0]** κατά ένα. Έτσι υπολογίζονται οι μονάδες. Με την επόμενη ρουτίνα υπολογίζονται τα δέκατα αφαιρώντας το 5 από το υπόλοιπο που αφήνει η προηγούμενη. Κανονικά θα έπρεπε να αφαιρείται το 5.1 αλλά η χρήση δεκαδικών αριθμών έχει μεγάλο κόστος σε μνήμη και χρόνο εκτέλεσης. Η εκδοχή που παρουσιάζεται δεν επηρεάζει σημαντικά την ακρίβεια του αποτελέσματος όπως φαίνεται στο διάγραμμα του σχήματος 1. Τα δέκατα αποθηκεύονται στη μεταβλητή **output[1]**. Στη συνέχεια γίνεται στρογγυλοποίηση του αποτελέσματος στο πλησιέστερο δέκατο. Το υπόλοιπο της διαίρεσης αναμένεται να είναι 0 – 4. Αν είναι μεγαλύτερο του 2 η στρογγυλοποίηση γίνεται προς τα πάνω αυξάνοντας τα δέκατα κατά ένα και αν αυτά γίνουν δέκα μηδενίζονται αυξάνοντας τις μονάδες κατά μία.



Σχήμα 1. Κανονική διαίρεση και στρογγυλοποίηση στο πρώτο δεκαδικό (NORMAL) και διαίρεση με στρογγυλοποίηση με τον κώδικα που παρουσιάζεται (AVR ADC).

6. Μετατροπή τιμής μιας μεταβλητής σε χαρακτήρες ASCII με τη συνάρτηση «itoa»

```
#include <avr/io.h>
#include <stdlib.h>
char tmpasc[6];
int16_t tmp;
            /* Your code here */
itoa (tmp, tmpasc, 10);
```

Με τη συνάρτηση ***itoa*** μπορούμε να μετατρέψουμε την τιμή μιας μεταβλητής από δυαδικό αριθμό σε χαρακτήρες ASCII. Στο παράδειγμα ο δυαδικός αριθμός είναι το περιεχόμενο της μεταβλητής **tmp** η οποία πρέπει να είναι προσημασμένη 16bit (τύπου **int16_t**). Οι χαρακτήρες ASCII αποθηκεύονται στον πίνακα **tmpasc** τα στοιχεία του οποίου είναι προσημασμένοι χαρακτήρες (τύπου **char**). Η διάσταση του πίνακα πρέπει να είναι υποχρεωτικά 6 ώστε να αποθηκευτούν τα 5 ψηφία και το πρόσημο[2]. Οι αρνητικοί αριθμοί έχουν τη μορφή του συμπληρώματος ως προς δύο. Το πρόσημο εμφανίζεται ως χαρακτήρας μόνο στους αρνητικούς αριθμούς. Στον πίνακα, στο στοιχείο 0 αποθηκεύεται το πρόσημο, όταν υπάρχει, ή το μεγαλύτερης αξίας ψηφίο. Στο στοιχείο 1 αποθηκεύεται το αμέσως μικρότερης αξίας ψηφίο κοκ. Αν περισσέψουν στοιχεία σε αυτά αποθηκεύεται η τιμή **0x00** που αντιστοιχεί στον ASCII NULL και είναι μη εκτυπώσιμος χαρακτήρας. Η ***itoa*** έχει διάφορες παραμέτρους στη σύνταξή της οι οποίες πρέπει να προσδιοριστούν. Στην πρώτη παράμετρο

[2] *16bit προσημασμένη μεταβλητή μπορεί να πάρει τιμές -32768 – 32767 στο δεκαδικό σύστημα.*

αναφέρεται το όνομα της μεταβλητής που περιέχει τον δυαδικό αριθμό (στο παράδειγμα η **tmp**). Η δεύτερη παράμετρος χωρίζεται με κόμμα (,) από τη πρώτη και αναφέρεται σε αυτήν το όνομα της μεταβλητής που θα αποθηκευτούν οι χαρακτήρες ASCII και στην τρίτη παράμετρο το σύστημα αρίθμησης που θα γίνει η μετατροπή. Για να χρησιμοποιηθεί η ***itoa*** πρέπει να συμπεριληφθεί η βιβλιοθήκη **stdlib.h**.

7. Μετατροπή τιμής μιας μεταβλητής μικρότερης του 10 σε χαρακτήρα ASCII

```
tmp |= (3<<4);
```

Όταν η τιμή μιας μεταβλητής είναι μικρότερη του 10 (0 – 9) δε χρειάζεται να γίνει χρήση της ***itoa*** γιατί ο χαρακτήρας ASCII είναι μόνο ένας αφού ο αριθμός είναι μονοψήφιος. Με την παραπάνω εντολή τοποθετούνται (γίνοται **1**) τα bit 4 και 5 της μεταβλητής **tmp** ενώ **όλα τα υπόλοιπα bit μένουν ως έχουν**. Είναι ισοδύναμη με την εντολή: **tmp = tmp | 0b00110000;** Αν υποθέσουμε ότι το περιεχόμενο της **tmp** αρχικά ήταν **0b00000101 (0x05)**, μετά την εφαρμογή της εντολής θα γίνει **0b00110101 (0x35)**, που αντιστοιχεί στον χαρακτήρα «5» σε κωδικοποίηση ASCII.

8. Μετατροπή τιμής μιας μεταβλητής 16bit σε χαρακτήρες ASCII

```
#include <avr/io.h>
int16_t Int16bit;
uint8_t OutASCII[6];

if (Int16bit<0)                  // If negative
{
   OutASCII[0]='-';              // Write negative sign
   Int16bit=(~Int16bit)+1;       // Convert two's complement
}
else if (Int16bit>=0)            // If positive
OutASCII[0]=' ';                 // Write "space"

OutASCII[1]=0;                   // Reset 10000's digit
while (Int16bit>9999)            // Calculate new 10000's digit
{
   Int16bit -= 10000;            // Subtract 10000
   OutASCII[1]++;                // Increase 10000's digit
}
OutASCII[1] |= (3<<4);           // Convert single digit binary to ASCII

OutASCII[2]=0;                   // Reset 1000 digit
while (Int16bit>999)             // Calculate new 1000 digit
{
   Int16bit -= 1000;             // Subtract 1000
   OutASCII[2]++;                // Increase 1000's digit
}
OutASCII[2] |= (3<<4);           // Convert single digit binary to ASCII

OutASCII[3]=0;                   // Reset 100 digit
while (Int16bit>99)              // Calculate new 100 digit
{
```

```
   Int16bit -= 100;                // Subtract 100
   OutASCII[3]++;                  // Increase 100's digit
}
OutASCII[3] |= (3<<4);             // Convert single digit binary to ASCII

OutASCII[4]=0;                     // Reset 10 digit
while (Int16bit>9)                 // Calculate new 10 digit
{
   Int16bit -= 10;                 // Subtract 10
   OutASCII[4]++;                  // Increase 10's digit
}
   OutASCII[4] |= (3<<4);          // Convert single digit binary to ASCII

OutASCII[5] = ((3<<4) | Int16bit);       // Convert rest (monads) single digit
                                         // binary number to ASCII
```

Με τον κώδικα που παρουσιάζεται μπορούμε να μετατρέψουμε την τιμή μιας προσημασμένης μεταβλητής 16bit από δυαδικό αριθμό σε χαρακτήρες ASCII. Στο παράδειγμα ο δυαδικός αριθμός είναι το περιεχόμενο της μεταβλητής **`Int16bit`** η οποία πρέπει να είναι τύπου **int16_t**. Οι χαρακτήρες ASCII αποθηκεύονται στον πίνακα **`OutASCII`** τα στοιχεία του οποίου είναι μη προσημασμένοι χαρακτήρες (τύπου **uint8_t**). Η διάσταση του πίνακα πρέπει να είναι υποχρεωτικά 6 ώστε να αποθηκευτούν τα 5 ψηφία και το πρόσημο[3].

Στην αρχή εξετάζεται αν ο αριθμός είναι αρνητικός ή θετικός. Σε περίπτωση αρνητικού, αποθηκεύεται το πρόσημο στο στοιχείο 0 του πίνακα και μετατρέπεται ο αριθμός από μορφής συμπληρώματος ως προς 2 σε κανονικό δυαδικό αριθμό. Σε περίπτωση θετικού η μηδέν αποθηκεύεται ο χαρακτήρας του κενού στην ίδια θέση. Στη συνέχεια γίνεται έλεγχος για δεκάδες χιλιάδες. Αρχικά μηδενίζεται το στοιχείο 1 του πίνακα και εκτελείται μια ρουτίνα που, όσο η μεταβλητή **`Int16bit`** είναι μεγαλύτερη από το 9999, αφαιρεί συνεχώς από αυτήν το 10000 και μετά από κάθε αφαίρεση αυξάνει την τιμή του στοιχείου κατά ένα. Αν η **`Int16bit`** είναι μικρότερη από 10000 στο στοιχείο παραμένει η τιμή 0. Με την επόμενη εντολή μετατρέπεται η δυαδική τιμή του συγκεκριμένου στοιχείου[4] σε χαρακτήρα ASCII. Ακολουθεί η ίδια λογική για τον έλεγχο χιλιάδων, εκατοντάδων, δεκάδων ενώ το υπόλοιπο των μονάδων μετατρέπεται απευθείας σε ASCII και αποθηκεύεται στο αντίστοιχο στοιχείο.

Με τον παραπάνω κώδικα αδυνατεί να αποδοθεί η ελάχιστη αρνητική τιμή -32768. Η μετατροπή του συμπληρώματος ως προς δύο για αυτόν τον αριθμό οδηγεί πάλι στον ίδιο αριθμό:

(-32768) `0b1000 0000 0000 0000`

εφαρμόζοντας NOT γίνεται: `0b0111 1111 1111 1111`

---

[3] *16bit προσημασμένη μεταβλητή μπορεί να πάρει τιμές -32768 – 32767 στο δεκαδικό σύστημα. Στο δυαδικό σύστημα οι αρνητικοί αριθμοί έχουν τη μορφή του συμπληρώματος ως προς δύο.*

[4] *Κανένα από τα στοιχεία του πίνακα δε μπορεί να πάρει τιμή μεγαλύτερη του 9 και για αυτό η μετατροπή μπορεί να γίνει χωρίς πρόβλημα.*

και προσθέτοντας 1 γίνεται: `0b1000 0000 0000 0000`

ο οποίος είναι μικρότερος του μηδέν και έτσι το αποτέλεσμα από τη μετατροπή σε ASCII προκύπτει: `-0000`

Το πρόβλημα μπορεί να ξεπεραστεί με τον κώδικα που παρουσιάζεται στη συνέχεια. Σε αυτή την εκδοχή αντιμετωπίζονται ξεχωριστά οι αρνητικοί και οι θετικοί αριθμοί, με την μετατροπή του συμπληρώματος ως προς δύο να γίνεται μόνο για το αρνητικό υπόλοιπο. Το σημείο που υστερεί είναι ότι είναι μεγαλύτερος σε έκταση, χωρίς όμως να σημαίνει ότι θα χρειαστεί περισσότερο χρόνο να εκτελεστεί.

```
if (AdcOut10n<0)
{
   output10[0]='-';
   output10[1]=0;
   while (AdcOut10n<-9999)
   {
      AdcOut10n += 10000;
      output10[1]++;
   }
   output10[1] |= (3<<4);

   output10[2]=0;
   while (AdcOut10n<-999)
   {
      AdcOut10n += 1000;
      output10[2]++;
   }
   output10[2] |= (3<<4);

   output10[3]=0;
   while (AdcOut10n<-99)
   {
      AdcOut10n += 100;
      output10[3]++;
   }
   output10[3] |= (3<<4);

   output10[4]=0;
   while (AdcOut10n<-9)
   {
      AdcOut10n += 10;
      output10[4]++;
   }
   output10[4] |= (3<<4);

   output10[5] = ((3<<4) | ((~AdcOut10n)+1));
}

else if (AdcOut10n>=0)
{
   output10[0]=' ';
   output10[1]=0;
   while (AdcOut10n>9999)
   {
      AdcOut10n -= 10000;
      output10[1]++;
   }
   output10[1] |= (3<<4);
```

```
   output10[2]=0;
   while (AdcOut10n>999)
   {
      AdcOut10n -= 1000;
      output10[2]++;
   }
   output10[2] |= (3<<4);

   output10[3]=0;
   while (AdcOut10n>99)
   {
      AdcOut10n -= 100;
      output10[3]++;
   }
   output10[3] |= (3<<4);

   output10[4]=0;
   while (AdcOut10n>9)
   {
      AdcOut10n -= 10;
      output10[4]++;
   }
   output10[4] |= (3<<4);

   output10[5] = ((3<<4) | AdcOut10n);
}
```

9. Καταμέτρηση εκτυπώσιμων χαρακτήρων σε μονοδιάστατο πίνακα

```
#include <avr/io.h>
#include <string.h>
char tmpasc[6];
              /* Your code here */
for (uint8_t idht = 0; idht < (unsigned)strlen(tmpasc); idht++)
     /* Your condition here */
```

Η καταμέτρηση των εκτυπώσιμων χαρακτήρων ενός μονοδιάστατου πίνακα μπορεί να γίνει με τη συνάρτηση ***strlen*** η οποία επιστρέφει κατά την κλήση της το πλήθος αυτών. Στη σύνταξή της αναφέρεται το όνομα του πίνακα στην παρένθεση που τη συνοδεύει δεξιά. Στο παράδειγμα καλείται μέσα σε εντολή **for** και επιστρέφει έναν αριθμό με τον οποίο γίνεται σύγκριση με την **idht** για τη λειτουργία της εντολής. Για να χρησιμοποιηθεί η ***strlen*** πρέπει να συμπεριληφθεί η βιβλιοθήκη **string.h**.