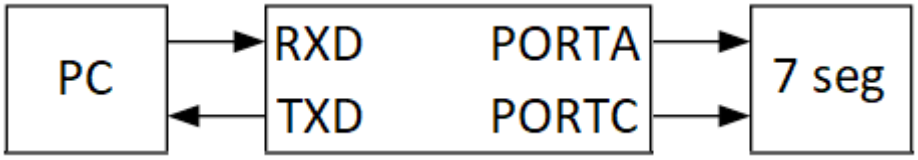
HPY411- Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Εργαστήριο 4

LAB41145851 **31/10/2020**

Εμμανουήλ Πετράκος ΑΜ 2014030009

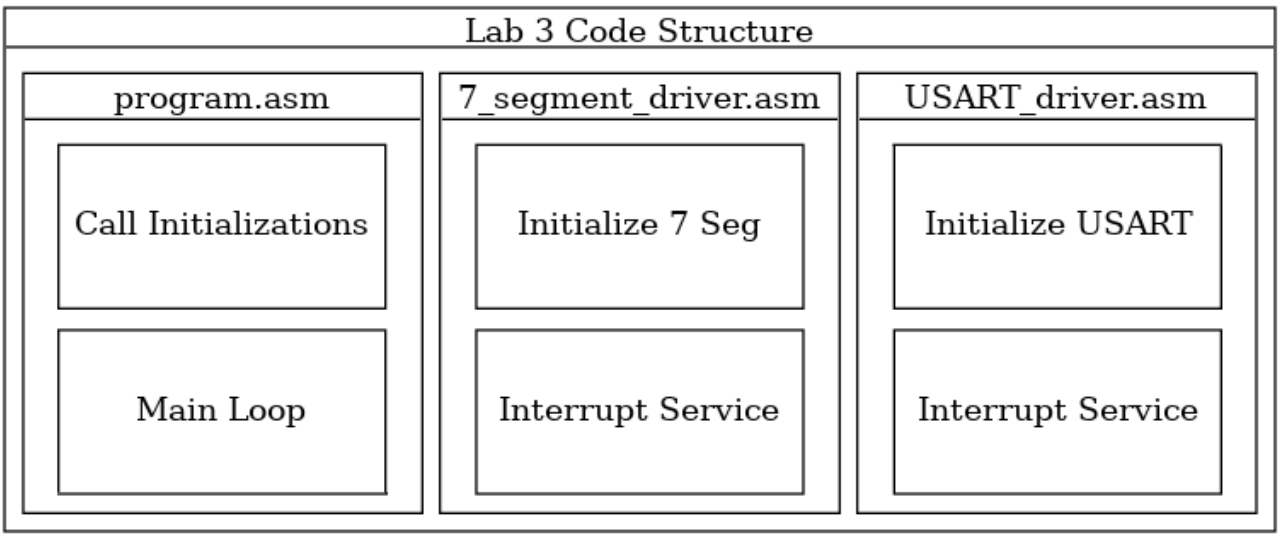
Στο τέταρτο εργαστήριο δεν αλλάζουν οι προδιαγραφές του συστήματος, αλλά το κυρίως πρόγραμμα και οι αρχικοποιήσεις υλοποιούνται σε C. Και πάλι, ως είσοδος θεωρούνται τα δεδομένα που φτάνουν στον δέκτη του USART, ενώ ως έξοδος οι απαντήσεις στον πομπό του USART και τα 7 segment LEDs που ελέγχονται από τις θύρες A και C.

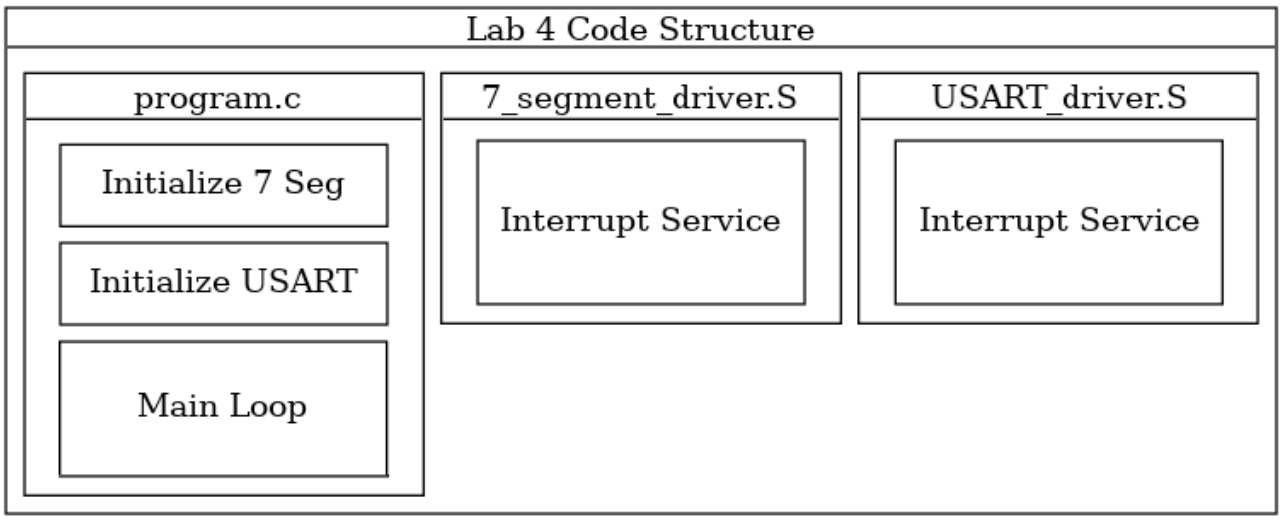


Top level schematic

Επεξήγηση προσέγγισης

Στο προηγούμενο εργαστήριο, ο κώδικας ήταν χωρισμένος σε τρία αρχεία assembly, ένα για κάθε driver με τις αρχικοποιήσεις του και την εξυπηρέτηση των interrupts του, και ένα για την main ρουτίνα. Στο παρόν εργαστήριο, η δομή έχει αλλάξει και αποτελείται από ένα αρχείο C που περιέχει την main ρουτίνα μαζί με όλες τις αρχικοποιήσεις και δύο αρχεία assembly με τους κώδικες εξυπηρέτησης των interrupts. Πρακτικά, ο κώδικας ήταν ήδη χωρισμένος σε Initialize και Service blocks και η αλλαγή ήταν αρκετά εύκολη. Στις παρακάτω εικόνες φαίνεται η δομή του κώδικα πριν και μετά την αλλαγή.





Μέχρι το προηγούμενο εργαστήριο, η παραγωγή του εκτελέσιμου γινόταν μέσω του avr assembler (AVRASM). Για να χρησιμοποιηθεί κώδικας C, είναι αναγκαίος ο avr-gcc compiler/ assembler και πρέπει να γίνουν αλλαγές στον κώδικα assembly.

- Ο AVRASM δεν έχει linker. Αν ο κώδικας βρίσκεται σε πολλά αρχεία, χρησιμοποιείται η ντιρεκτίβα "`.include`" και ο assembler τα ενώνει σε ένα. Αντίθετα, ο avr-gcc έχει linker και για να χρησιμοποιηθούν ρουτίνες σε άλλα αρχεία πρέπει να δηλωθούν "`.global`", ενώ για να χρησιμοποιηθούν μεταβλητές από άλλα αρχεία πρέπει να δηλωθούν εξωτερικές με την ντιρεκτίβα "`.extern`".
- Για να καταλάβει ο avr-gcc ότι ένα όνομα αναφέρεται σε I/O καταχωρητή, χρησιμοποιείται το macro "`_SFR_IO_ADDR()`".

  Πχ, η εντολή "`in r18, UDR`" γίνεται "`in r18, _SFR_IO_ADDR( UDR )`".
- Ντιρεκτίβες HIGH(), LOW() αντικαθιστώνται με τις αντίστοιχες hi8() , low8().
- Οι καταχωρητές που χρησιμοποιούνται έχουν αλλάξει ώστε να ακολουθούνται οι συμβάσεις του avr-gcc που φαίνονται στον παρακάτω πίνακα. Αν και δεν επηρεάζεται η λειτουργικότητα του παρόν προγράμματος, αυτό έγινε ώστε ο κώδικας να είναι προετοιμασμένος για τα επόμενα εργαστήρια.

Table 5-1. Summary of the register interfaces between C and assembly.

Register	Description	Assembly code called from C	Assembly code that calls C code
r0	Temporary	Save and restore if using	Save and restore if using
r1	Always zero	Must clear before returning	Must clear before calling
r2-r17	"call-saved"	Save and restore if using	Can freely use
r28			
r29			
r18-r27	"call-used"	Can freely use	Save and restore if using
r0			
r31			

Στις αρχικοποιήσεις των driver, εξαιτίας της μετατροπής τους σε C, υπάρχουν δυο σημαντικές αλλαγές σε σχέση με το προηγούμενο εργαστήριο. Πρώτον, δεν χρειάζεται να γίνει αρχικοποίηση του stack pointer καθώς ο compiler εισάγει αυτόματα τον αντίστοιχο κώδικα. Δεύτερον, η διαχείριση της μνήμης γίνεται αυτόματα από τον compiler. Αντί να χρησιμοποιούνται pointers με τις πραγματικές διευθύνσεις, δηλώνονται με τον παρακάτω τρόπο και χρησιμοποιούνται με τα αντίστοιχα ονόματα.

```
volatile unsigned char data[8] __attribute__ ((section (".noinit")));
```

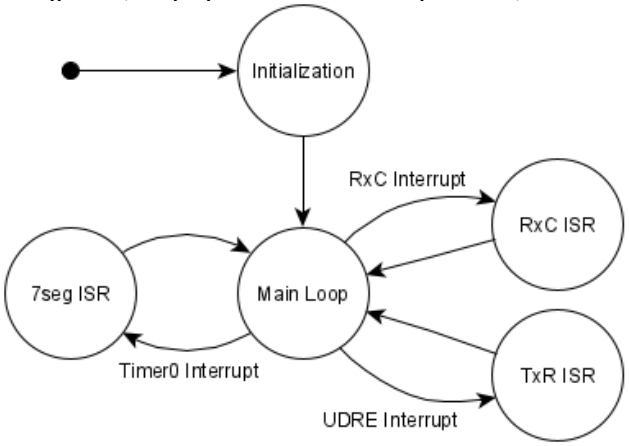
Το "`volatile`" χρησιμοποιείται για να καταλάβει ο compiler ότι η μεταβλητή μπορεί να αλλάξει εκτός κανονικής ροής του προγράμματος, πχ σε interrupt, και να αποφευχθούν αθέμιτες βελτιστοποιήσεις. Επίσης, παρατηρήθηκε ότι ο compiler εισήγαγε κώδικα για να αρχικοποιήσει τις μεταβλητές με την τιμή 0, κάτι αχρείαστο καθώς όλες αρχικοποιούνται στις αντίστοιχες συναρτήσεις. Για να μην γίνει αυτό, υπάρχει το macro "`__attribute__ ((section (".noinit")))`".

Παρακολουθώντας την SRAM μέσω του simulator, φαίνεται ότι ο compiler χαρτογραφεί την μνήμη με τον παρακάτω τρόπο.

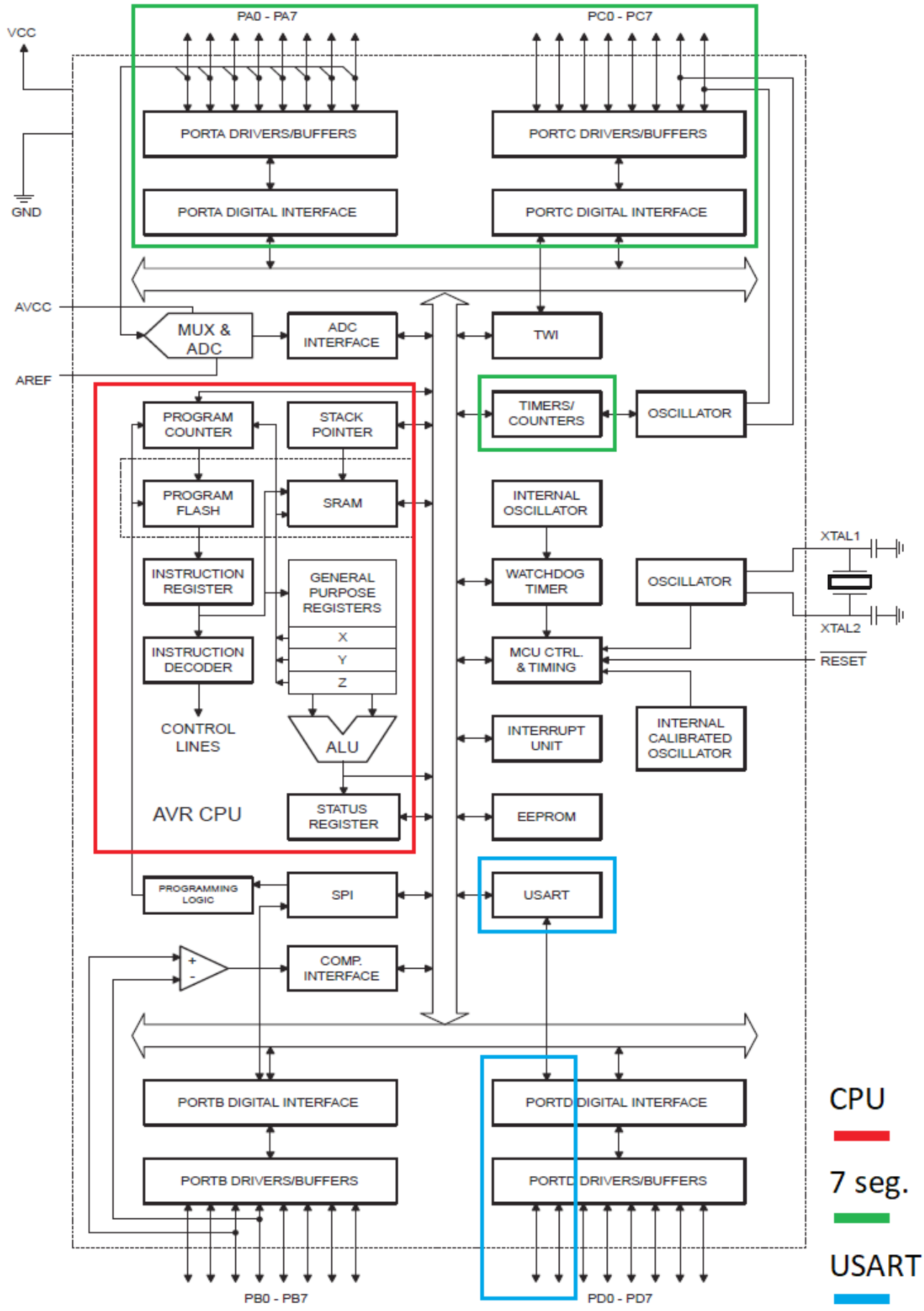
0x0000 ... 0x005F	Register address space
0x0060	Remaining transmits
0x0061	Transmitter state
0x0062 ... 0x006c	7 segment format 0 ... 7 segment format A
0x006d ... 0x0074	Data LBS ... Data MSB
0x0075 ...	Free Memory

Χάρτης μνήμης

Η λειτουργικότητα του συστήματος παραμένει ίδια και παρουσιάζεται στο παρακάτω διάγραμμα.



Τέλος, παρουσιάζεται ο χάρτης χρησιμοποιούμενων πόρων.



Χάρτης πόρων μικροελεγκτή

Πειραματική Διαδικασία

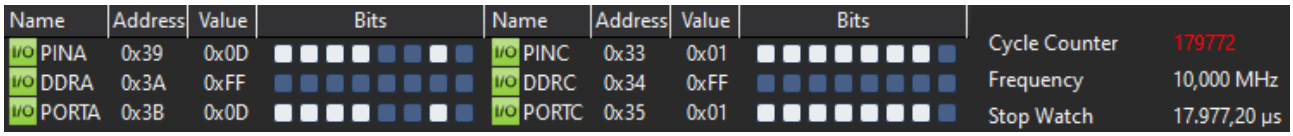
Ο έλεγχος της λειτουργίας του προγράμματος γίνεται με τον ίδιο τρόπο με τα προηγούμενα εργαστήρια. Για τον έλεγχο του driver του USART παρακολουθούνται η μνήμη και οι απαντήσεις μετά από τις εντολές που βρίσκονται στο αρχείο usart.stim, ενώ για τον έλεγχο του driver των 7 segment παρακολουθούνται οι PORTA και PORTC.

Εντολή	Κατάσταση μνήμης δεδομένων, 0x006d – 0x0074 (LSB → MSB)							
-	0a	0a	0a	0a	0a	0a	0a	0a
N1<CR><LF>	01	0a	0a	0a	0a	0a	0a	0a
N123<CR><LF>	03	02	01	0a	0a	0a	0a	0a

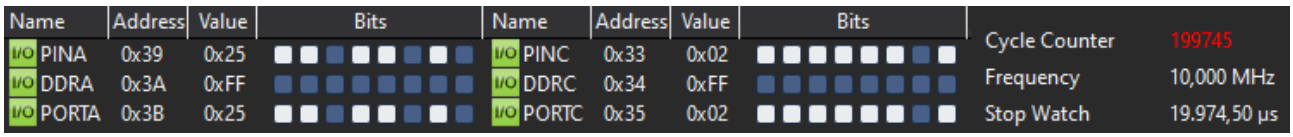
Στο lab.log φαίνονται οι απαντήσεις προς το PC και η καθυστέρηση μεταξύ των χαρακτήρων:

```
TCNT2 = 0x4f
#25
TCNT2 = 0x4b
#10399
TCNT2 = 0x0d
#10401
TCNT2 = 0x0a
#25175
TCNT2 = 0x4f
#25
TCNT2 = 0x4b
#10419
TCNT2 = 0x0d
#10401
TCNT2 = 0x0a
```

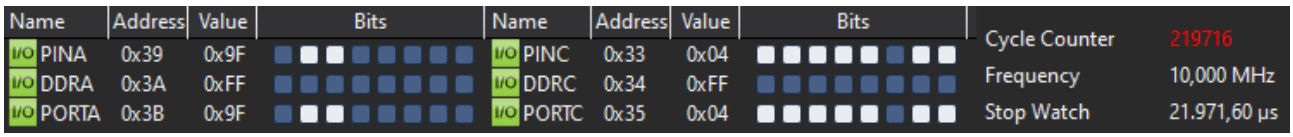
Ακολουθούν οι έξοδοι που ελέγχουν τα 7 segment LED μετά την αποδοχή των μηνυμάτων.



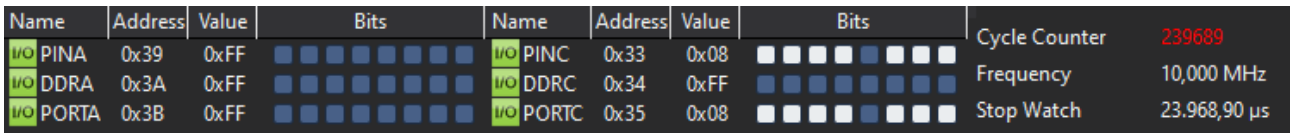
PORTA = 0b00001101 (3), PORTC = AN0



PORTA = 0b00100101 (2), PORTC = AN1



PORTA = 0b10011111 (1), PORTC = AN2



PORTA = 0b11111111 (τίποτα), PORTC = AN3

Ανάλυση & Παρατηρήσεις

Κοιτώντας τον κώδικα assembly που παράγει ο compiler, παρατηρούνται μερικές διαφορές σε σχέση με τον κώδικα του προηγούμενου εργαστηρίου. Όπως αναφέρθηκε πριν, παράγει κώδικα για την αρχικοποίηση του stack pointer και της μνήμης. Επίσης, μετά τον κώδικα του προγράμματος κλείνει τα interrupt και επιστρέφει στην αρχή, πιθανότατα για να μην αφήσει το πρόγραμμα να κρεμάσει αν για κάποιο λόγο η ροή του ξεφύγει από το main loop.

Ο παραγόμενος κώδικα assembly των αρχικοποιήσεων δεν διαφέρει ιδιαίτερα με τον χειρόγραφο του προηγούμενου εργαστηρίου, η μόνη εμφανής διαφορά είναι το πλήθος κλήσης συναρτήσεων. Στο προηγούμενο εργαστήριο, οι αρχικοποιήσεις ήταν χωρισμένες σε αρκετές υπορουτίνες για λόγους αναγνωσιμότητας. Αντίθετα, όταν είναι υλοποιημένες σε C δεν υπάρχει τέτοια ανάγκη καθώς ο κώδικας είναι πιο συνοπτικός. Σαν αποτέλεσμα, και η παραγόμενη assembly είναι χωρισμένη σε λιγότερες υπορουτίνες σε σχέση με την χειρόγραφη.

Η κατανάλωση πόρων, επεξεργαστικής δύναμης και μνήμης είναι ίδια με το προηγούμενο εργαστήριο, καθώς δεν έχει αλλάξει ο κώδικας των ρουτινών εξυπηρέτησης των interrupts.

Πηγές

Atmel AT1886: Mixing Assembly and C with AVRGCC
http://ww1.microchip.com/downloads/en/appnotes/doc42055.pdf

avr-lib: Combining C and assembly source files
https://www.nongnu.org/avr-libc/user-manual/group__asmdemo.html