Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2016-2017

ΑΘΗΝΑ 9 Νοεμβρίου 2016

6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

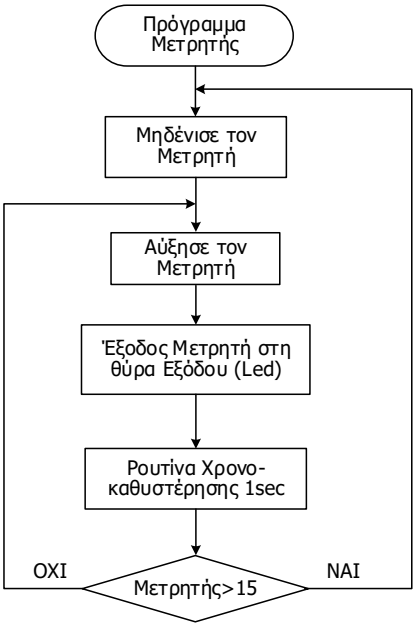
1η Εργ. Άσκ. στον Μικροελεγκτή AVR - Χρήση υπορουτινών και χρονοκαθυστερήσεων
(υλοποίηση στο εκπαιδευτικό σύστημα easyAVR6)

Εξέταση – επίδειξη 23/11/2016

Χρονοκαθυστερήσεις

Μια χρήσιμη εφαρμογή συστημάτων μικροελεγκτών είναι η ανταπόκριση σε εξωτερικές συνθήκες σε τακτά χρονικά διαστήματα. Για το σκοπό αυτό είναι πολύ χρήσιμη η ανάπτυξη σχετικού λογισμικού (υπορουτίνες) που να δημιουργεί ακριβείς και συγκεκριμένες χρονοκαθυστερήσεις και να χρησιμοποιήται από οποιαδήποτε χρονικά εξαρτώμενη εφαρμογή. Βοήθεια για την ανάπτυξη αυτού του κώδικα δίνουν τα τεχνικά χαρακτηριστικά του εκάστοτε μικροελεγκτή και συγκεκριμένα η περίοδος ρολογιού και οι κύκλοι εκτέλεσης κάθε εντολής, από τα οποία προκύπτει ο χρόνος εκτέλεσης κάθε εντολής. Η δημιουργία κώδικα χρονοκαθυστέρησης συνήθως επιτυγχάνεται με τη διαδοχική εκτέλεση μιας σειράς εντολών που δεν παράγουν κανένα χρήσιμο αποτέλεσμα (συνηθίζεται η εντολή nop). Το μέγεθος της σειράς μαζί με κατάλληλους πολλαπλασιαστικούς βρόχους δημιουργούν την επιθυμητή χρονοκαθυστέρηση. Η τεχνική αυτή φαίνεται στην παρακάτω υπορουτίνα **wait_usec**, που για τον μικροελεγκτή AVR ATmega16 και την αναπτυξιακή πλακέτα EasyAVR6 (συχνότητα ρολογιού 8MHz, περίοδος ρολογιού 0.125μsec), είναι μια χρονοκαθυστέρηση τόσων μsec, όση η δυαδική τιμή του καταχωρητή r25:r24 κατά την κλήση. Επίσης παρακάτω δίνεται η ρουτίνα **wait_msec** που αξιοποιεί την προηγούμενη και αυτή προκαλεί χρονοκαθυστέρηση τόσων msec, όση η τιμή του καταχωρητή r25:r24. Οι ρουτίνες αυτές αξιοποιούνται στο επόμενο παράδειγμα

Παράδειγμα 1.1 Να προγραμματίσετε και να επιδείξετε στο εκπαιδευτικό σύστημα easyAVR6 χρονόμετρο δευτερολέπτων που απεικονίζει το χρόνο σε δυαδική μορφή πάνω στα LED PA3-PA0. Το χρονόμετρο όταν φτάνει στην τιμή 15_{10}, στο επόμενο βήμα ξαναρχίζει από την αρχή. Όλο το πρόγραμμα σας δίνετε και το ζητούμενο είναι να περάσει από το AVRStudio5 αρχικά για προσομοίωση και στη συνέχεια την παραγωγή του εκτελέσιμου κώδικα που πρέπει να κατέβει στην πλακέτα για την επίδειξη της ορθής λειτουργίας στο πραγματικό σύστημα. Ακολουθούν τα αναγκαία προγράμματα και οι ρουτίνες assembly:



Σχήμα 1. 1. Πρόγραμμα μετρητής modulo 15.

```
.include "m16def.inc"

reset:  ldi r24 , low(RAMEND)   ; initialize stack pointer
        out SPL , r24
        ldi r24 , high(RAMEND)
        out SPH , r24
        ser r24                 ; initialize PORTA for output
        out DDRA , r24
        clr r26                  ; clear time counter

main:   out PORTA , r26
        ldi r24 , low(1000)     ; load r25:r24 with 1000
        ldi r25 , high(1000)    ; delay 1 second
        rcall wait_msec
        inc r26                 ; increment time counter, one second passed
        cpi r26 , 16            ; compare time counter with 16
        brlo main               ; if lower goto main, else clear time counter
        clr r26                 ; and then goto main
        rjmp main

.include "wait.asm"
```

Ρουτίνα: wait_μsec
Προκαλεί καθυστέρηση τόσων μsec, όση η τιμή του καταχωρητή r25:r2
Είσοδος: Ο χρόνος (1 - 65535 μs) μέσω του καταχωρητή r25:r24
Καταχωρητές: r25:r24

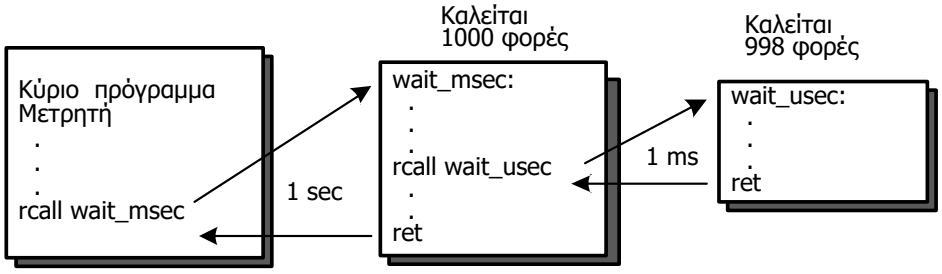
```
wait_usec:
        sbiw r24 ,1         ; 2 κύκλοι (0.250 μsec)
        nop                 ; 1 κύκλος (0.125 μsec)
        nop                 ; 1 κύκλος (0.125 μsec)
        nop                 ; 1 κύκλος (0.125 μsec)
        nop                 ; 1 κύκλος (0.125 μsec)
        brne wait_usec      ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
        ret                 ; 4 κύκλοι (0.500 μsec)
```

Από τα σχόλια φαίνεται οτι ο παραπάνω κώδικας, όταν εκτελείται ο επαναληπτικός βρόχος, απαιτεί 8 κύκλους ρολογιού ή 1μsec. Άρα, όσες φορές εκτελεστεί ο βρόχος, τόσα μsec καθυστέρησης απαιτούνται. Η μικροδιαφορές που προκύπτουν από την μια φορά που θα εκτελεστεί η έξοδος από το βρόχο και η εντολή επιστροφής (ret), μπορούν αν απαιτηθεί να συνυπολογιστούν στον κώδικα που καλεί την υπορουτίνα wait_usec. (αναλυτικά, η υπορουτίνα wait_usec με είσοδο r25:r24=n καθυστερεί n-1+0.875+0.500=n+0.375 μsec). Για παράδειγμα, η παρακάτω υπορουτίνα για τον μικροελεγκτή AVR ATmega16 και την αναπτυξιακή πλακέτα EasyAVR6 είναι μια χρονοκαθυστέρηση τόσων msec, όση η δυαδική τιμή που περιέχεται στο ζευγάρι καταχωρητών r25:r24 κατά την κλήση και βασίζεται στην προηγούμενη (wait_usec).

Ρουτίνα: wait_msec
Προκαλεί καθυστέρηση τόσων msec, όση η τιμή του καταχωρητή r25:r2
Είσοδος: Ο χρόνος (1 - 65535 ms) μέσω του καταχωρητή r25:r24
Καταχωρητές: r25:r24
Καλούμενες υπορουτίνες: wait_usec

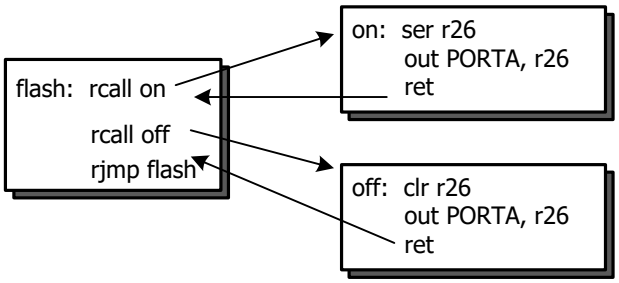
```
wait_msec:
        push r24            ; 2 κύκλοι (0.250 μsec)
        push r25            ; 2 κύκλοι
        ldi r24 , low(998)  ; φόρτωσε τον καταχ.  r25:r24 με 998 (1 κύκλος - 0.125 μsec)
        ldi r25 , high(998) ; 1 κύκλος (0.125 μsec)
        rcall wait_usec     ; 3 κύκλοι (0.375 μsec), προκαλεί συνολικά καθυστέρηση 998.375 μsec
        pop r25             ; 2 κύκλοι (0.250 μsec)
        pop r24             ; 2 κύκλοι
        sbiw r24 , 1        ; 2 κύκλοι
        brne wait_msec      ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
        ret                 ; 4 κύκλοι (0.500 μsec)
```

Από τα σχόλια φαίνεται ότι η παραπάνω υπορουτίνα wait_msec, όταν εκτελείται ο επαναληπτικός βρόχος, απαιτεί 17 κύκλους ρολογιού ή 2.125μsec και μαζί με τη χρονοκαθυστέρηση της υπορουτίνας wait_usec, που με είσοδο 998 είναι 998.375μsec, συνολικά 1000.5μsec ή 1.0005msec.



Σχήμα 1. 2. Κλήσεις υπορουτινών στο πρόγραμμα του μετρητή *modulo* 15.

Παράδειγμα 1.2 Δίνεται ένα παράδειγμα προγράμματος που αναβοσβήνει συνεχώς τα LEDs εξόδου του συστήματος easyAVR6. Το κύριο πρόγραμμα έχει μόνο 3 βασικές εντολές: μια που καλεί την ρουτίνα ON, μια που καλεί την ρουτίνα OFF και μια που ξαναγυρνά στην αρχή. Το σχήμα 1.3 δείχνει πως χρησιμοποιεί υπορουτίνες για να αναβοσβήνει τα LEDs της θύρας PORTA.



Σχήμα 1.3. Πρόγραμμα που αναβοσβήνει τα LEDs.

Στη συνέχεια παρουσιάζουμε αναλυτικά την εφαρμογή.

Πίνακας 1.1. Πρόγραμμα που αναβοσβήνει τα LEDs

Ετικέτα	Εντολή	Σχόλια
	ser r26	; αρχικοποίηση της PORTA
	out DDRA , r26	; για έξοδο
flash:	**rcall on**	; Άναψε τα LEDs
	nop	; Για προσθήκη εντολών 200 ms
	nop	
	rcall off	; Σβήσε τα LEDs
	nop	; Για προσθήκη εντολών
	nop	
	rjmp flash	; Επανέλαβε
	; Υπορουτίνα για να ανάβουν τα LEDs	
on:	ser r26	; θέσε τη θύρα εξόδου των LED
	out PORTA , r26	
	ret	; Γύρισε στο κύριο πρόγραμμα
	; Υπορουτίνα για να σβήνουν τα LEDs	
off:	clr r26	; μηδένισε τη θύρα εξόδου των LED
	out PORTA , r26	
	ret	; Γύρισε στο κύριο πρόγραμμα

Τα ζητούμενα της 6ης εργαστηριακής άσκησης (AVR-1)

Ζήτημα 1.1 Να προγραμματίσετε σε assembly και να επιδείξετε στο εκπαιδευτικό σύστημα easyAVR6 πρόγραμμα που να απεικονίζει ένα αναμμένο led το οποίο να κινείται πάνω στα bit της θύρας PA0-PA7 από αριστερά προς τα δεξιά και αντίστροφα όταν φτάσει σε ένα άκρο. Κάθε led θα μένει αναμμένο 0.5 sec. Η κίνηση του led θα ελέγχεται από το push button PB0. Όταν αυτό είναι πατημένο η κίνηση να σταματά, ενώ διαφορετικά να συνεχίζεται.

Ζήτημα 1.2 Τροποποιήστε το παράδειγμα του Πίνακα 1.1, ώστε η καθυστέρηση στο άναμμα και το σβήσιμο των led (PA0-PA7) να καθορίζεται από τις τιμές (0 - 15) των dip switches PB0-PB3 και PB4-PB7 αντίστοιχα. Η αντίστοιχη διάρκεια να καθορίζεται με βάση τη σχέση $D=x*100$ msec όπου $x \in (0\text{-}15)$ είναι η δεκαεξαδική τιμή των dip switches PB0-PB3 ή PB4-PB7. Έτσι η μικρότερη καθυστέρηση είναι 100 msec και η μεγαλύτερη 1500 msec. Αν έχουμε την τιμή $x=0$ στα dip switches PB0-PB3 ή PB4-PB7 τα led να είναι μόνιμα σβησμένα ή αναμμένα αντίστοιχα. Αν και τα δύο έχουν τιμή 0 τότε τα led να είναι μόνιμα σβησμένα. Το πρόγραμμα να δοθεί σε assembly.

Ζήτημα 1.3 Να γραφτεί πρόγραμμα σε C για το σύστημα easyAVR6 το οποίο αρχικά να ανάβει το led0 που είναι συνδεδεμένο στο bit0 της θύρας εξόδου PortB (απεικόνιση με θετική λογική - αναμμένο λογικό 1, σβηστό λογικό 0 - αντίστοιχα και για τα υπόλοιπα ledx => bitx PortB). Στην συνέχεια με το πάτημα των διακοπτών (Push-buttons) SW0-4 που υποθέτουμε ότι είναι συνδεδεμένα στα αντίστοιχα bit της θύρας εισόδου PortD να συμβαίνουν τα εξής:

- SW0 μετακίνηση του led μια θέση αριστερά (κυκλικά).
- SW1 μετακίνηση του led μια θέση δεξιά (κυκλικά).
- SW2 μετακίνηση του led δυο (2) θέσεις αριστερά (κυκλικά).
- SW3 μετακίνηση του led δυο (2) θέσεις δεξιά (κυκλικά).
- SW4 μετακίνηση του αναμμένου led στην αρχική του θέση (LSB - led0).

Όλες οι αλλαγές να γίνονται αφήνοντας (επανερχόμενα) τα Push-buttons SWx (bitx PortD), οι εντολές έχουν προτεραιότητα με μεγαλύτερη αυτή του SW0 και μικρότερη αυτή του SW4. Έτσι αν είναι πατημένο το SW1 και το SW2 και αφεθούν ταυτόχρονα, τότε θα πραγματοποιηθεί η εντολή που αντιστοιχεί στο SW1. Σημείωση: οι αλλαγές γίνονται στο άφημα του push button, δηλαδή αν είναι πατημένο το SW1 και το SW2 και αφεθεί μόνο το SW2 τότε πρέπει να πραγματοποιηθεί η εντολή που αντιστοιχεί στο SW2. Επίσης υποθέτουμε ότι οι διακόπτες είναι συνδεδεμένοι με θετική λογική (για πάτημα δίνουν λογικό'1').