**4η Σειρά Ασκήσεων (Εργαστήριο)**

**Συστήματα Μικροϋπολογιστών**

Βερέμης Αλέξανδρος (03115063)  
Μπαρουξή Ευδοκία (03116658)  
Ταράση Στελίνα (03116442)

**Άσκηση 2.1 : Σκοπός της Άσκησης:**Έπρεπε να προγραμματίσουμε σε assembly πρόγραμμα που να απεικονίζει ένα αναμμένο led που να αντιστοιχεί στα bit της θύρας PB0-PB7. Το αναμμένο led έπρεπε να κινείται συνεχώς ξεκινώντας από τα LSB προς τα MSB και αντίστροφα όταν φτάνει στο άλλο άκρο. Κάθε φορά το led έπρεπε να μένει αναμμένο για ~0.5 sec. Η κίνηση του αναμμένου led ελέγχεται από το push button PA0. Όταν αυτό είναι πατημένο η κίνηση σταματάει, ενώ διαφορετικά συνεχίζεται.

**Επεξήγηση του Κώδικα:**

Αρχικά στον κώδικα κάνουμε την τυπική διαδικασία για να έχουμε ως έξοδο το PORTB ,καθαρίζουμε τους καταχωρητές που θα χρησιμοποιήσουμε και έχουμε 2 βασικές συναρτήσεις: τη main και main2.

Στη main έχουμε ολίσθηση του καταχωρητή μια θέση αριστερότερα ,ώστε να ανάβει το επόμενο LED ,σε κάθε επανάληψη μέχρι να φτάσει στο 128 όπου τότε περνάει στη main2 και εκεί εχουμε shift του καταχωρητή μια θέση δεξιότερα, ώστε να ανάβει το προηγούμενο LED ,σε κάθε επανάληψη μέχρι να φτάσει στο 2 ,από όπου και ξαναρχίζουμε από την αρχή γιατί στον καταχωρητή r26 βάζουμε 1 και πηγαίνουμε ξανά στην main.

Μέσα στις main και main2 υπάρχουν οι συναρτήσεις kathisterisi και kathisterisi2 αντίστοιχα, όπου κάνουν αυτό ακριβώς που λέει το όνομά τους δηλαδή καθυστερούν την συνέχιση της κίνησης για όση ώρα είναι πατημένο το PA0. Αυτό το καταφέρνουμε με την επανάληψη kathisterisi(2) η οποία συνεχίζεται μέχρι να σταματήσουμε να πατάμε το PA0.  
Την συνάρτηση wait\_msec την καλούμε για 0.5 δευτερόλεπτα αφού έχουμε φορτώσει στους καταχωρητές r24,r25 τον αριθμό 500 που αντιστοιχεί σε 500 msec δηλαδή 0.5 sec.

Ο κώδικας που χρησιμοποιήσαμε:  
  
.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 DDRB , r24

clr r26 ; clear time counter

clr r16

inc r26

main: out PORTB , r26

kathisterisi:

in r16,PINA

andi r16,0x01

cpi r16,0

brne kathisterisi

ldi r24 , low(500) ; load r25:r24 with 500

ldi r25 , high(500) ; delay 0.5 second

rcall wait\_msec

lsl r26 ; shift left time counter, half a second passed

cpi r26 , 128 ; compare time counter with 129 >128

brlo main ; if lower goto main, else main2

main2: out PORTB , r26

kathisterisi2:

in r16,PINA

andi r16,0x01

cpi r16,0

brne kathisterisi2

ldi r24 , low(500) ; load r25:r24 with 500

ldi r25 , high(500) ; delay half second

rcall wait\_msec

lsr r26

cpi r26 , 2

brsh main2

clr r26 ; theloyme na valoume to 1 ston r26

inc r26

rjmp main

wait\_msec:

push r24

push r25

ldi r24 , low(998)

ldi r25 , high(998)

rcall wait\_usec

pop r25

pop r24

sbiw r24 , 1

brne wait\_msec

ret

wait\_usec:

sbiw r24 ,1

nop

nop

nop

nop

brne wait\_usec

ret

**Άσκηση 2.2 : Σκοπός της Άσκησης:**Μας ζητήθηκε να υλοποιηθούν σε ένα σύστημα Μικροελεγκτή AVR οι λογικές συναρτήσεις: F0=(ABC+ C'D)'  
F1=(A+B)⋅(C+D)  
Οι τιμές των F0-F1 εμφανίζονται αντίστοιχα στα δύο LSB της θύρας εξόδου PORTB (0-1).  
Οι μεταβλητές εισόδου (A,B,C,D) αντιστοιχούν στα 4bit της θύρας εισόδου PORTA(0-3), με το A να βρίσκεται στο LSB. Το πρόγραμμα έπρεπε να δοθεί σε assembly και σε C.

**Assembly: Επεξήγηση του Κώδικα:**

Στον κώδικα αρχικά κάνουμε την τυπική διαδικασία για να έχουμε ως έξοδο το PORTB και καθαρίζουμε τους καταχωρητές που θα χρησιμοποιήσουμε.

Στη main: σε κάθε επανάληψη καθαρίζουμε τους r26,r20,21 γιατί στη συνέχεια θα συμμετέχουν σε μερικά and και δε θέλουμε να έχει ξεμείνει μέσα τους κάποιος άσσος.  
Στη συνέχεια, παίρνουμε από το PORTA 1 ή 0 ανάλογα αν είναι ή όχι πατημένο το αντίστοιχο PIN. Κάνουμε τα αντίστοιχα shift right όπου χρειάζεται ώστε ο άσσος να πάει στο LSB αν υπάρχει. Μετέπειτα σώζουμε A,C γιατί θα τα ξαναχρειαστούμε.   
Έπειτα κάνουμε τις απαραίτητες λογικές πράξεις όπως ζητείται στην εκφώνηση, πρώτα για το F1 που είναι πιο εύκολο να υπολογιστεί και μετά για το F0. Υπάρχουν και τα αντίστοιχα σχόλια στον κώδικα. Για τo F1 κάνουμε ενα shift left για να μπει στην κατάλληλη θέση στον καταχωρητή r26 που θα βγάλουμε στην έξοδο PORTB, ενώ το F0 απλά το προσθέτουμε στον καταχωρητή r26.

Ακόμη , για να πάρουμε το αντίθετο ενός bit στο τελευταίο μόνο bit με τη χρήση καταχωρητών στην Assembly χρειάστηκε να κάνουμε:  
com r18 ; 1’s complement   
andi r18,0x01  
Kαι στο τέλος όλης αυτής της διαδικασίας κάνουμε rjmp main , δηλαδή ξαναγυρνάμε πίσω στη main για να επαναλάβουμε την ίδια διαδικασία αενάως.

Ο κώδικας που χρησιμοποιήσαμε:  
  
.include "m16def.inc"

reset: ldi r24 , low(RAMEND) ; initialize stack pointer

out SPL , r24

ldi r24 , high(RAMEND)

out SPH , r24

ser r24 ; initialize PORTB for output

out DDRB , r24

clr r16

clr r17

clr r18

clr r19

clr r20

clr r21

clr r26 ; clear teliko register me apotelesma

main: out PORTB , r26

clr r26

clr r20

clr r21

in r16,PINA

in r17,PINA

in r18,PINA

in r19,PINA

andi r16,01 ;a

andi r17,0x02 ;b

lsr r17

andi r18,0x04 ;c

lsr r18

lsr r18

andi r19,0x08 ;d

lsr r19

lsr r19

lsr r19

or r20,r16 ;sozoyme a,c

or r21,r18

or r18,r19 ;c=c or d

or r16,r17 ; a=a or b

and r16,r18 ; a= (a or b) and (c or d), o r1 exei tin F1.

lsl r16 ;r1=r1 \*2

or r26,r16 ; ston r26 mpainei sti sosti 8esi to F1.

ori r16,0x01

ori r18,0x01

and r16,r20 ;epanaferoyme ta a,c poy eixame swsei

and r18,r21

and r16,r17; a=a&b

and r16,r18; a=a&b&c

com r18; !c

andi r18,0x01 ; gia na paroyme to anti8eto mono sto teleytaio bit akolou8oyme ayti ti diadikasia

and r19,r18; d=!c&d

or r16,r19; r1= oli i parastasi i esoteriki

com r16; r1= sosto twra

andi r16,0x01

add r26,r16 ;pleon r26 exei to swsto

rjmp main

**C: Επεξήγηση του Κώδικα:**

Στον κώδικα της C, ξεκινάμε δηλώνοντας τις μεταβλητές A,B,C,D,F0,F1 που θα χρησιμοποιήσουμε και αρχικοποιώντας το PORTB ως output και το PORTA ως input.

Έχουμε μια διαρκή επανάληψη μέσω της while(1) ,στην οποία γίνονται τα εξής :  
Στο Α καταχωρείται το LSB της εισόδου PORTA με την εντολή A= PINA & 0x01; και αντίστοιχα τα υπόλοιπα bit στα B,C,D με τις εντολές B= PINA & 0x02; B= B>>1; C= PINA & 0x04; C= C>>2; D= PINA & 0x08 D= D>>3; Κάνουμε τα απαραίτητα shift right ώστε το bit που μας ενδιαφέρει να είναι στο LSB όλων των καταχωρητών.  
Στη συνέχεια υπολογίζουμε τα F0, F1 για τις εκφράσεις που μας ζητούνται και κάνουμε shift left για το F1 ώστε να βρίσκεται στην κατάλληλη θέση για την έξοδο.

Τέλος στην έξοδο θέτουμε την τιμή (F0+F1).

#include <avr/io.h>

char A,B,C,D,F0,F1;

int main(void)

{

DDRB=0xFF;

DDRA=0x00;

while(1)

{

A= PINA & 0x01;

B= PINA & 0x02;

B= B>>1; //shift right

C= PINA & 0x04;

C= C>>2; //shift right

D= PINA & 0x08;

D= D>>3; //shift right

F0=!((A&B&C)|(!C&D));

F1=(A|B)&(C|D);

F1=F1<<1; //shift left

//OR

//F1=F1\*2;

PORTB=(F0 + F1);

}

return 0;

}

**Άσκηση 2.3 : Σκοπός της Άσκησης:**Στην επεξήγηση του κώδικα γίνεται αναλυτική αποσαφήνιση του σκοπού της άσκησης.

**Επεξήγηση του Κώδικα:**

Στη 3η άσκηση θεωρούμε πως το led0 είναι συνδεδεμένο με το bit0 της θύρας εξόδου PORTA και με το πάτημα των διακοπτών SW0-3, που είναι συνδεδεμένα με τα αντίστοιχα bit της θύρας εισόδου PORTC, εκτελούνται συγκεκριμένες διαδικασίες.  
Πιο συγκεκριμένα, ξεκινάμε το πρόγραμμα μας όπως προηγουμένως δηλώνοντας το x που θα χρησιμοποιήσουμε και με αρχικοποίηση των PORTA ως output και PORTC ως input. Θέτουμε το x=1 και το PORTA να ισούται με x γιατί στην αρχή μας ζητείται να ανάβει το led0.   
Σε μια διαρκή επανάληψη εκτελούμε τα εξής:   
Αρχικά με τη συνθήκη if ((PINC & 0x01) == 1) ελέγχουμε πρώτα αν είναι πατημένο το SW0, και αντίστοιχα με τις else if ((PINC & 0x02) == 2), else if ((PINC & 0x04) == 4), else if ((PINC & 0x08) == 8) αν είναι πατημένο το SW1, το SW2 ή το SW3.

Στην περίπτωση που ((PINC & 0x01) == 1) θέλουμε να εκτελέσουμε ολίσθηση-περιστροφή του led μια θέση αριστερά. Αρχικά ελέγχουμε αν x==128 ώστε να μη συμβεί υπερχείλιση , αν είναι αληθής τότε θέτουμε x=1 ,για να έχουμε κυκλική κίνηση ,διαφορετικά κάνουμε : x = x<<1;

Αν ((PINC & 0x02) == 2) θέλουμε να κάνουμε ολίσθηση-περιστροφή του led μια θέση δεξιά, αντίστοιχα με την 1η περίπτωση ελέγχουμε αν x==1, αν είναι αληθής τότε θέτουμε x=128 , για να έχουμε κυκλική κίνηση ,διαφορετικά κάνουμε : x =x>>1;

Αν ((PINC & 0x04) == 4) γίνεται μετακίνηση του αναμμένου led στην θέση ΜSB θέτοντας στο x   
την τιμή 128.   
  
Aν ((PINC & 0x08) == 8) : μετακινούμε το αναμμένο led στην αρχική του θέση θέτοντας στο x την τιμή 1.  
  
O λόγος που σε όλους τους ελέγχους if που κάνουμε , μέσα υπάρχει μια εντολή:  
 while ((PINC & 0x0(1/2/4/8)) == 1); είναι διότι θέλουμε όλες οι αλλαγές που κάνουμε να συμβαίνουν με το άφημα (δηλαδή στον επανερχομό) των διακοπτών SW0-3

Τέλος θέτουμε στην έξοδο την τιμή του x.

Ο κώδικας που χρησιμοποιήσαμε:  
  
#include <avr/io.h>

char x;

int main(void)

{

DDRA=0xFF;

DDRC=0x00;

x = 1;

PORTA = x;

while(1)

{

if ((PINC & 0x01) == 1){

while ((PINC & 0x01) == 1);

if (x==128)

x = 1;

else

x = x<<1;

}

else if ((PINC & 0x02) == 2) {

while ((PINC & 0x02) == 2);

if (x==1)

x=128;

else

x =x>>1;

}

else if ((PINC & 0x04) == 4) {

while ((PINC & 0x04) == 4);

x=128;

}

else if ((PINC & 0x08) == 8) {

while ((PINC & 0x08) == 8);

x=1;

}

PORTA = x;

}

return 0;

}