ΗΜΜΥ ΕΜΠ

ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ

4η (ΕΡΓΑΣΤΗΡΙΑΚΗ) ΑΝΑΦΟΡΑ

Φοιτητές: Αθανασίου Ιωάννης, 03117041, 6ο εξάμηνο
Καραβαγγέλης Αθανάσιος, 03117022, 6ο εξάμηνο
Τσιτσής Αντώνιος, 03117045, 6ο εξάμηνο

Ημερομηνία: 06/06/2020

ΑΣΚΗΣΗ 1

Στην άσκηση αυτή μας ζητείται να απεικονίζουμε το led της PORTB είτε να κινείται συνεχώς από το LSB στο MSB και αντίστροφα, είτε να μένει ακίνητο, ανάλογα με την κατάσταση του Push Button PC2. Η λογική που ακολουθούμε είναι να "σπάσουμε" την πορεία αυτή σε δύο υποτμήματα, ένα με την κίνηση από το LSB προς το MSB και ένα με την αντίστροφη. Χρησιμοποιούμε ένα τμήμα κώδικα για κάθε τέτοιο υποτμήμα κίνησης (ετικέτες: poreia1, poreia2). Για να υλοποιήσουμε τον περιορισμό το led να είναι ακίνητο για όσο είναι πατημένο το push button PC2, χρησιμοποιούμε τους επαναληπτικούς βρόχους ελέγχου check_push_button1 και check_push_button2 (έναν για κάθε πορεία), αφού απομονώσουμε το 3ο bit της PORTC με χρήση των εντολών BST, BLD.

Αναλύτικα ο κώδικας που γράψαμε :

```
.include "m16def.inc"

.def counter = r26
.def C_input = r27
.def push_button = r28

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 r24
                    out DDRC, r24       ; initialize PORTC for input

                    ldi counter, 1      ; initialize counterBREAKPOINT


poreia1:            out PORTB , counter     ;kinhsh pros aristera   BREAKPOINT

                    jmp check_push_button1  ;

poreia1_continue:   ldi r24 , low(1000)      ;
                    ldi r25 , high(1000)     ;  delay 1 sec
                    ; rcall wait_msec        ;

                    lsl counter     ; counter <- counter*2
                    cpi counter, 128    ; compare with 128 (8o bit)
                    brlo poreia1        ; if lower goto poreia1,
                                ; else go to poreia2(kinhsh pros dejia)

poreia2:            out PORTB, counter      ; kinhsh prow dejia BREAKPOINT

                    jmp check_push_button2  ;

poreia2_continue:   ldi r24, low(1000)
                    ldi r25, high(1000)
                    ; rcall wait_msec
                    lsr counter             ; counter <- counter / 2
                    cpi counter, 1
                    breq poreia1
                    jmp poreia2

check_push_button1: in C_input, PINC
                    BST C_input, 2      ; T <- C_input(2) apomonwsh tou 3ou bit
tou                                     PINC BREAKPOINT
```

```
                    BLD push_button, 0  ; push_button(0) <- T, metafora tou sto
1o                                          bit tou kataxwrhth push button

                    cpi push_button, 1

                    breq check_push_button1 ; push_button==1 -> again
BREAKPOINT

                    jmp poreia1_continue; else return to poreia1



check_push_button2: in C_input, PINC

                    BST C_input, 2      ; T <- C_input(2)   BREAKPOINT

                    BLD push_button, 0  ; push_button(0) <- T

                    cpi push_button, 1

                    breq check_push_button2     ; BREAKPOINT

                    jmp poreia2_continue


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)

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)
```

ΑΣΚΗΣΗ 2

Στην άσκηση αυτή μας ζητείται η απεικόνιση 2 λογικών συναρτήσεων σε ένα σύστημα Μικροελεγκτή AVR(προσομοίωση με το Atmel Studio 7). Οι λογικές συναρτήσεις αυτές είναι οι εξής **F0 = (AB' + BC'D)'** και **F1= (A+C)·(B+D)** .Για τις εισόδους A,B,C,D χρησιμοποιούμε αντίστοιχα τα LSB 0-3 του PORT B και για τις εξόδους F0-F1 αντίστοιχα τα LSB 0-1 του PORT A . Για την υλοποίηση ,απομονώνουμε και με κατάλληλη περιστροφή φέρνουμε στη θέση του LSB το κάθε bit εισόδου ώστε να λάβει την πραγματική του τιμή 0 ή 1. Για την αναπαράσταση του NOT χρησιμοποιούμε XOR για να πάρουμε το συμπλήρωμα. Έπειτα με κατάλληλη χρήση των τελεστών &,| αλλά και παρενθέσεων παίρνουμε το αποτέλεσμα των εξόδων F0,F1. Τέλος ,περιστρέφουμε αριστερά μία θέση την έξοδο της F1 ώστε να αναπαρίσταται στο 2^{ο} LSB ,την προσθέτουμε στην F0 και το αποτέλεσμα των 2 bit πλέον το αναθέτουμε στην PORT A.

Με κάθε πάτημα του κουμπιού “RUN” μέσω του Atmel Studio 7 , ανάλογα με την είσοδο, οδηγούμαστε κατευθείαν στην εντολή της εξόδου αφού έχουμε αναθέσει breakpoint στην εντολή αυτή .Συγκεκριμένα, υπάρχουν 16 διαφορετικές καταστάσεις εισόδου ανάλογα με τις εισόδους A,B,C,D και 4 διαφορετικές έξοδοι 00,01,10 ή 11.

```
#include <avr/io.h>
char A,B,NOTB,C,NOTC,D,F0,F1;

int main(void)
{
    DDRA=0xFF; // Αρχικοποίηση PORTA ως output
    DDRB=0x00; // Αρχικοποίηση PORTB ως input
    while (1)
    {
        A = PINB & 0x01; //Απομόνωση των 4 LSB για τα A,B,C,D αντίστοιχα
        B = PINB & 0x02;
        B=B>>1;          //1 περιστροφή δεξιά για να έχουμε την πραγματική τιμή
        C = PINB & 0x04;
        C=C>>2;          //2 περιστροφές δεξιά     -//-         -//-
        D = PINB & 0x08;
        D=D>>3;          //3-//-           -//-      -//-

        NOTC = C^0x01;   //XOR για να πάρουμε το συμπλήρωμα
        NOTB = B^0x01;   //          -//-

        F0 = ((A & NOTB)|(B & NOTC & D))^0x1; //F0=(AB'+BC'D)'

        F1 = (A | C)&(B | D) ;               //F1=(A+C)(B+D)
        F1=F1<<1;        //1 περιστροφή αριστέρα αφού θέλουμε
                         //να αναπαρίσταται στο 2ο LSB
        F0 = F0 + F1;    //εμφανίζουμε ουσιαστικά και τα 2 LSB με την πρόσθεση
αυτή
     •  PORTA = F0;      //στην PORT A ,όπου έχουμε ορίσει και breakpoint
    }
    return 0;
}
```

ΑΣΚΗΣΗ 3

Το παραπάνω ζήτημα λύθηκε με χρήση C σε AVR Atmega16. Το πρόγραμμα διαβάζει αρχικά την είσοδο απο PORTA και ανάλογα με τις τιμες 1,2,4,8 το LED στην PORTB κάνει κυκλική κίνηση δεξιά, κυκλική κίνηση αριστερά, μετακίνηση στο MSB και LSB αντίστοιχα. Με την χρήση while επιτυγχάνουμε να λαμβάνουμε τις παραπάνω κινήσεις του LED όταν κλείσει το αντίστοιχο switch δηλαδή έχει ανοιγοκλείσει. Γι' αυτόν τον λόγο σε κάθε ένα από τα 4 while τοποθετούμε και ένα breakpoint για καλύτερη παρουσίαση της διαδικασίας:

```
#include <avr/io.h>
int main(void)

{   DDRB=0xFF;
    DDRA=0x00;

    int x = 1;          //or char doesn't matter

    while(1){           //always on

        if ((PINA & 0x01) == 1){                //if first switch opens

            while((PINA & 0x01) == 1);          //wait to be unpressed
            if (x==1)
               x = 128;                 //check for overflow
            else
               x = x>>1;                    //shift right
        }

        if ((PINA & 0x02) == 2){            //if second switch opens

            while((PINA & 0x02) == 2);      //wait to be unpressed
            if (x==128)
               x = 1;                       //check for overflow
```

```
            else
               x = x<<1;                    //shift left
        }

        if ((PINA & 0x04) == 4){            //if third switch opens

           while((PINA & 0x04) == 4);       //wait to be unpressed
           x = 1;                           //1st LSB
        }

        if ((PINA & 0x08) == 8){            //if fourth switch opens

           while((PINA & 0x08) == 8);       //wait to be unpressed
           x = 128;                     //4rth LSB
        }

        PORTB = x;                          //output
    }

return 0;
}
```