ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (MICROLAB)

4η Εργαστηριακή Αναφορά στο μάθημα "**ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ**" του 7ου Εξαμήνου

των φοιτητών της **ομάδας 17**,

Εμμανουήλ Αναστάσιου Σερλή, Α.Μ. 03118125
Ευάγγελου Περσόπουλου, Α.Μ.: 03121701

1η Άσκηση: -> micro_lab04_ex01.asm και micro_lab04_ex01.c

Το ζητούμενο πρόγραμμα υλοποιεί την μετατροπή της τάσης ADC στην αναλογική είσοδο A2, και τυπώνει στην οθόνη LCD το αποτέλεσμα με ακρίβεια 2 δεκαδικών ψηφίων. Στην assembly υλοποίηση, η μέτρηση της τιμής του ADC γίνεται εντός του αντίστοιχου interrupt, ενώ σε C υλοποίηση αυτό γίνεται εντός του κυρίως προγράμματος. Η σχέση που χρησιμοποιείται για την αναλογική τιμή τάσης είναι η εξής:

$$V_{IN} = (ADC/1024) * V_{REF}$$

,με V_{IN}: $Αναλογικα\ ταση\ από\ 0\ ως\ 5V$
V_{REF}: $Tαση\ αναφορας\ που\ εχει\ οριστει\ στα\ 5V$
ADC: Τιμή στον καταχωρητή ADC (από 0 ως 1023)

Assembly Program:

```
.include "m328PBdef.inc" ;running

.def VinL=r18
.def VinH=r19
.def counter=r20
.def ADC_L=r21
.def ADC_H=r22
.def temp=r23

.org 0x00
    jmp reset
.org 0x02A
    jmp ISR_ADC

reset:
    ldi r24, high(RAMEND)
    out SPH, r24
    ldi r24, low(RAMEND)
    out SPL, r24

    ;io configuration
    clr r24
    out DDRC, r24   ;input for ADC
    ser r24
    out DDRB, r24   ;output for counter
    out DDRD, r24   ;output for LCD

    ;ADC configuration
    ldi r24, 0b01000010
    sts ADMUX, r24
    ldi r24, 0b10001111
```

```
    sts ADCSRA, r24
    sei

main:
    clr r24
    rcall lcd_init ;initialize lcd
    ldi r24, LOW(16*20)
    ldi r25, HIGH(16*20)
    rcall wait_msec

loop1: ;for PORTB counter
    clr counter
loop2:
    out PORTB, counter
    ldi r24, low(16*300)
    ldi r25, high(16*300)
    rcall wait_msec
    inc counter
    cpi counter, 63

    ;enable ADC
    lds r24, ADCSRA
    ori r24, (1<<ADSC)
    sts ADCSRA, r24

    breq loop1
    rjmp loop2




ISR_ADC:
    push r24
    out SREG, r24
    push r24
    push r25

    ldi r24, 0x01   ;clear lcd
    rcall lcd_command
    ldi r24, LOW(5)
    ldi r25, HIGH(5)
    rcall wait_msec

    lds ADC_L, ADCL
    lds ADC_H, ADCH

    ;Vin=(ADC*5)/2^10
    mov VinL, ADC_L
    mov VinH, ADC_H
    lsl VinL
    rol VinH
    lsl VinL
```

```
rol VinH
add VinL, ADC_L
adc VinH, ADC_H ;Vin=ADC*5

;for /1024 => shift right Vin 10 times
;but for integer of /1024 we just need bit 10-13
mov temp, VinH
lsr temp
lsr temp
andi temp, 0x0F

mov r24, temp
ldi temp, 0x30  ;add '0' for lcd (ASCII code)
add r24, temp
rcall lcd_data
//ldi r24, LOW(16*200)
//ldi r25, HIGH(16*200)
//rcall wait_msec

ldi r24, '.'
rcall lcd_data
//ldi r24, LOW(16*200)
//ldi r25, HIGH(16*200)
//rcall wait_msec

;for first demical Vin*10 and take bit 10-13
andi VinH, 0x03 ;we dont need the bits that we used for integer
mov ADC_L, VinL
mov ADC_H, VinH
lsl VinL ;for Vin*10 we shift left Vin 3 times (2^3=8)
rol VinH ;and we add 2 times the original Vin to the shifted
lsl VinL ;
rol VinH ;
lsl VinL ;
rol VinH ;
add VinL, ADC_L
adc VinH, ADC_H
add VinL, ADC_L
adc VinH, ADC_H

mov temp, VinH
lsr temp
lsr temp
andi temp, 0x0F

mov r24, temp
ldi temp, 0x30  ;add '0' for lcd (ASCII code)
add r24, temp
rcall lcd_data
//ldi r24, LOW(16*200)
//ldi r25, HIGH(16*200)
//rcall wait_msec
```

```
    ;same procedure for the second demical
    andi VinH, 0x03
    mov ADC_L, VinL
    mov ADC_H, VinH
    lsl VinL
    rol VinH
    lsl VinL ;
    rol VinH ;
    lsl VinL ;
    rol VinH ;
    add VinL, ADC_L
    adc VinH, ADC_H
    add VinL, ADC_L
    adc VinH, ADC_H

    mov temp, VinH
    lsr temp
    lsr temp
    andi temp, 0x0F

    mov r24, temp
    ldi temp, 0x30  ;add '0' for lcd (ASCII code)
    add r24, temp
    rcall lcd_data
    //ldi r24, LOW(16*200)
    //ldi r25, HIGH(16*200)
    //rcall wait_msec

    //ldi r24, 0x01   ;clear lcd
    //rcall lcd_command
    //ldi r24, LOW(16*200)
    //ldi r25, HIGH(16*200)
    //rcall wait_msec

    pop r25
    pop r24
    in r24, SREG
    pop r24
    reti


;----------------------------------------------------

write_2_nibbles:  ;write data (first 4MSB and next 4LSB)
    push r24
    in r25 ,PIND
    andi r25 ,0x0f
    andi r24 ,0xf0
    add r24 ,r25
    out PORTD ,r24
    sbi PORTD ,3
```

```
    cbi PORTD ,3
    pop r24
    swap r24
    andi r24 ,0xf0
    add r24 ,r25
    out PORTD ,r24
    sbi PORTD ,3
    nop
    nop
    cbi PORTD ,3
    nop
    nop
    ret
;------------------------------------------------
lcd_data: ;send one byte of data
    sbi PORTD ,2
    rcall write_2_nibbles
    ldi r24 ,43
    ldi r25 ,0
    rcall wait_usec
    ret
;-------------------------------------------------
lcd_command: ;send instruction to lcd controller
    cbi PORTD ,2
    rcall write_2_nibbles
    ldi r24 ,43
    ldi r25 ,0
    rcall wait_usec
    ret
 ;-------------------------------------------------
lcd_init: ;initialize lcd
    ldi r24 ,40
    ldi r25 ,0
    rcall wait_msec
    ldi r24 ,0x30
    out PORTD ,r24
    sbi PORTD ,3
    cbi PORTD ,3
    ldi r24 ,100
    ldi r25 ,0
    rcall wait_usec
    ldi r24 ,0x30
    out PORTD ,r24
    sbi PORTD ,3
    cbi PORTD ,3
    ldi r24 ,100
    ldi r25 ,0
    rcall wait_usec
    ldi r24 ,0x20 ; 4-bit mode
    out PORTD ,r24
    sbi PORTD ,3
    cbi PORTD ,3
```

```
    ldi r24 ,100
    ldi r25 ,0
    rcall wait_usec
    ldi r24 ,0x28
    rcall lcd_command
    ldi r24 ,0x0c
    rcall lcd_command
    ldi r24 ,0x01
    rcall lcd_command
    ldi r24 ,low(5000)
    ldi r25 ,high(5000)
    rcall wait_usec
    ldi r24 ,0x06 ;
    rcall lcd_command
    ret
;-------------------------------------
wait_msec: ;delay routine in ms
    ldi r23,249
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24,1
    brne wait_msec

    ret
;---------------------------------
wait_usec: ;delay routine in us
    ldi r23,200
loop_inn2:
    dec r23
    nop
    brne loop_inn2

    sbiw r24,1
    brne wait_usec

    ret
```

C Program:

```
#define F_CPU 16000000UL //running
#include "avr/io.h"
#include<avr/interrupt.h>
#include<util/delay.h>

//given lcd functions written in C
void write_2_nibbles(char x)
```

```
{
      char y=PIND & 0x0f;
      char x1=x & 0xf0;
      x1=x1+y;
      PORTD=x1;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      x=x<<4 | x>>4;
      x=x & 0xf0;
      PORTD=x+y;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
}

void lcd_data(char x)
{
      PORTD=PORTD | (1<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_command(char x)
{
      PORTD=PORTD | (0<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_init (void)
{
      _delay_ms(40);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x20;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      lcd_command(0x28);
      lcd_command(0x0c);
      lcd_command(0x01);
      _delay_ms(500);
      lcd_command(0x06);
}
void lcd_print(unsigned char a, unsigned char b, unsigned char c)
{
    lcd_command(0x01);
```

```
    _delay_ms(3);
    lcd_data(a);
    lcd_data('.');
    lcd_data(b);
    lcd_data(c);
}


int main()
{
    DDRD |= 0b11111111;  // output for LCD
    DDRC |= 0b00000000;  // input for ADC
    DDRB |= 0b11111111;  // output for counter

    ADMUX |= 0b01000010; //ADLAR=0 -> left adjusted, ADC2 INPUT
    ADCSRA |= 0b10000111; // disable adc interrupt
    unsigned int temp, counter=0;
    unsigned char akeraios, prwto, deutero;
    ADCSRA |= (1<<ADSC); //start ADC read
    lcd_init();
    while(1)
    {

        PORTB = counter;
        _delay_ms(20);
        if (counter > 63)
        {
            counter = 0;
        }
        counter++;
        while ((ADCSRA & (1 << ADSC)) != 0) //stuck here till
convertion ends (ADSC = 0)
        {

        }
        temp = (ADC*5)/1024; //get value with 2-decimal accuracy
        akeraios = temp + '0';
        temp  = (((ADC*5)%1024)*10)/1024;
        prwto = temp +'0';
        temp  = ((((ADC*5)%1024)*10)%1024)*10/1024;
        deutero = temp +'0';
        lcd_print(akeraios, prwto, deutero);
        _delay_ms(500);
        ADCSRA |= (1<<ADSC);   //enable new conversion

    }

    return 0;

}
```

<u>2η Άσκηση:</u> -> micro_lab04_ex02.asm και micro_lab04_ex02.c

Το ζητούμενο πρόγραμμα αναγνωρίζει τα επίπεδα CO που υπάρχουν στην ατμόσφαιρα, μέσω κατάλληλου αισθητήρα ο οποίος είναι συνδεδεμένος στην αναλογική είσοδο A3. Συγκεκριμένα, αρχικά υπολογίσθηκε η τιμή στην είσοδο του ADC που αντιστοιχεί σε συγκέντρωση CO ίση με 70ppm:

$$C_x * M = V_{gas} - V_{gas0}, με:$$

$M = 12.9 * 10^{-3}\ V/ppm$

V_{gas0}=0.1V

, ενώ χρησιμοποιούμε και την σχέση για την τάση ADC:

$$V_{IN} = (ADC/1024) * V_{REF}$$

για να λάβουμε για $\boldsymbol{C_x = 70ppm \Rightarrow V_{gas} = 1.004V \Rightarrow ADC = 215}$

Τα διαφορετικά επίπεδα αερίου > 70ppm και η αντίστοιχη έξοδος στο PORTB και στο LCD δίνονται στον κάτωθι πίνακα:

Level	ADC Range	PORTB	LCD Message
0	0-214	0x00	CLEAR
1	215-255	0x01	GAS DETECTED
2	256-511	0x03	GAS DETECTED
3	512-767	0x07	GAS DETECTED
4	768-782	0x15	GAS DETECTED
5	783-830	0x31	GAS DETECTED
6	831-1023	0x63	GAS DETECTED

Σημειώνεται ότι-όπως και στην άσκηση 4.1-στην assembly υλοποίηση, η μέτρηση της τιμής του ADC γίνεται εντός του αντίστοιχου interrupt, ενώ σε C υλοποίηση αυτό γίνεται εντός του κυρίως προγράμματος.

<u>Assembly Program:</u>

```
.include "m328PBdef.inc" ;running

.def temp = r18
.def ADC_L = r16
.def ADC_H = r17
.def flag = r19

.org 0x00
    rjmp reset
.org 0x2A ;ADC Conversion Complete Interrupt
    rjmp ADC_inter

reset:
    ldi temp, LOW(RAMEND)
    out SPL,temp
    ldi temp, HIGH(RAMEND)
    out SPH,temp

    clr temp
    ldi temp, 0xFF
    out DDRD, temp ;Set PORTD as output

    clr temp
    ldi temp, 0x00
    out DDRC, temp ;Set PORTC as input

    clr temp
    ldi temp, 0xFF ;Set PORTB as output
    out DDRB, temp

    clr temp
    out PORTB,temp ;set PORTB leds to zero


    ; REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0011 => select
ADC3(pin PC3),
    ; ADLAR=1 => Left adjust the ADC result
    clr temp
    ori temp, 0b01000011
    sts ADMUX, temp

    ; ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
    ; ADIE=1 => enable adc interrupt, ADPS[2:0]=111 =>
fADC=16MHz/128=125KHz
    nop
    clr temp
    ori temp, 0b10001111
    sts ADCSRA, temp

    bset 7 ;set I flag
    sei
```

```
    clr r24
    rcall lcd_init

    nop
    ldi ADC_L,0x00 ;initialize CO registers
    ldi ADC_H,0x01
main:
    clr temp
    sts ADCSRA, temp
    ori temp, 0b11001111 ;enable interrupt
    sts ADCSRA,temp
    nop
    cpi ADC_H, 0x01 ;definetely over 70ppm
    brge GAS
    nop
    cpi ADC_L, 0xCD ;over 70ppm (205 ADC output)
    brlo NO_GAS
    rcall GAS
main_end:
    push r24
    push r25
    ldi r24,low(100) ;0.1sec delay for next counter
    ldi r25,high(100)
    rcall wait_msec
    pop r25
    pop r24

    ldi temp,0b11000111 ; enable ADC interrupt
    sts ADCSRA,temp
    rjmp main


ADC_inter:
    in temp,SREG
    nop
    push temp
    clr ADC_L
    clr ADC_H
    lds ADC_L,ADCL ;get ADC value
    lds ADC_H,ADCH
    nop
    pop temp
    out SREG,temp
    nop
    reti
NO_GAS:
    rcall LCD_WRITE_GOOD
    nop
    ldi flag,0
    rjmp BLINK
GAS:
    rcall LCD_WRITE_BAD
```

```
    nop
    cpi ADC_H,0 ;find level of gas
    breq LEVEL_1
    cpi ADC_H,1
    breq LEVEL_2
    cpi ADC_H,2
    breq LEVEL_3
    cpi ADC_H,3
    breq GAS_2
GAS_2:
    cpi ADC_L,63
    brge LEVEL_6
    cpi ADC_L, 15
    brge LEVEL_5
    cpi ADC_L,0
    brge LEVEL_4
    rjmp LEVEL_6
BLINK:
    out PORTB,flag
    nop
    push r24
    push r25
    ldi r24,low(16*200) ;0.2 sec delay for blink delay
    ldi r25,high(16*200)
    rcall wait_msec
    pop r25
    pop r24

    clr temp
    out PORTB,temp
    ldi r24,low(16*200) ;0.2 sec delay for blink delay
    ldi r25,high(16*200)
    rcall wait_msec
    rjmp main_end
LEVEL_1: ;blink different
    ldi flag,0x01
    rjmp BLINK
LEVEL_2:
    ldi flag,3
    rjmp BLINK
LEVEL_3:
    ldi flag,7
    rjmp BLINK
LEVEL_4:
    ldi flag,15
    rjmp BLINK
LEVEL_5:
    ldi flag,31
    rjmp BLINK
LEVEL_6:
    ldi flag,63
    rjmp BLINK
```

```
LCD_WRITE_GOOD:
    ldi r24 ,0x01 ; //clear screen
    rcall lcd_command
    //ldi r24,low(5) ;0.1sec delay for next counter
    //ldi r25,high(5)
    //rcall wait_msec


    ldi r24, 'C'
    rcall lcd_data
    ldi r24, 'L'
    rcall lcd_data
    ldi r24, 'E'
    rcall lcd_data
    ldi r24, 'A'
    rcall lcd_data
    ldi r24, 'R'
    rcall lcd_data
    ret
LCD_WRITE_BAD:
    ldi r24 ,0x01 ; //clear screen
    rcall lcd_command
    //ldi r24,low(5) ;0.1sec delay for next counter
    //ldi r25,high(5)
    //rcall wait_msec

    ldi r24, 'G'
    rcall lcd_data
    ldi r24, 'A'
    rcall lcd_data
    ldi r24, 'S'
    rcall lcd_data
    ldi r24, ' '
    rcall lcd_data
    ldi r24, 'D'
    rcall lcd_data
    ldi r24, 'E'
    rcall lcd_data
    ldi r24, 'T'
    rcall lcd_data
    ldi r24, 'E'
    rcall lcd_data
    ldi r24, 'C'
    rcall lcd_data
    ldi r24, 'T'
    rcall lcd_data
    ldi r24, 'E'
    rcall lcd_data
    ldi r24, 'D'
    rcall lcd_data
    ret
```

```
write_2_nibbles:  ;write data (first 4MSB and next 4LSB)
    push r24
    in r25 ,PIND
    andi r25 ,0x0f
    andi r24 ,0xf0
    add r24 ,r25
    out PORTD ,r24
    sbi PORTD ,3
    cbi PORTD ,3
    pop r24
    swap r24
    andi r24 ,0xf0
    add r24 ,r25
    out PORTD ,r24
    sbi PORTD ,3
    nop
    nop
    cbi PORTD ,3
    nop
    nop
    ret
;-----------------------------------------------
lcd_data: ;send one byte of data
    sbi PORTD ,2
    rcall write_2_nibbles
    ldi r24 ,43
    ldi r25 ,0
    rcall wait_usec
    ret
;-----------------------------------------------
lcd_command: ;send instruction to lcd controller
    cbi PORTD ,2
    rcall write_2_nibbles
    ldi r24 ,43
    ldi r25 ,0
    rcall wait_usec
    ret
 ;-----------------------------------------------
lcd_init: ;initialize lcd
    ldi r24 ,40
    ldi r25 ,0
    rcall wait_msec
    ldi r24 ,0x30
    out PORTD ,r24
    sbi PORTD ,3
    cbi PORTD ,3
    ldi r24 ,100
    ldi r25 ,0
    rcall wait_usec
    ldi r24 ,0x30
    out PORTD ,r24
```

```
    sbi PORTD ,3
    cbi PORTD ,3
    ldi r24 ,100
    ldi r25 ,0
    rcall wait_usec
    ldi r24 ,0x20 ; 4-bit mode
    out PORTD ,r24
    sbi PORTD ,3
    cbi PORTD ,3
    ldi r24 ,100
    ldi r25 ,0
    rcall wait_usec
    ldi r24 ,0x28
    rcall lcd_command
    ldi r24 ,0x0c
    rcall lcd_command
    ldi r24 ,0x01
    rcall lcd_command
    ldi r24 ,low(5000)
    ldi r25 ,high(5000)
    rcall wait_usec
    ldi r24 ,0x06 ;
    rcall lcd_command
    ret
;-------------------------------------
wait_msec: ;delay routine
    ldi r23,249
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24,1
    brne wait_msec

    ret
;---------------------------------
wait_usec: ;delay routine
    ldi r23,200
loop_inn2:
    dec r23
    nop
    brne loop_inn2

    sbiw r24,1
    brne wait_usec

    ret
```

C Program:

```
#define F_CPU 16000000UL // running
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>

//given lcd functions written in C
void write_2_nibbles(char x)
{
      char y=PIND & 0x0f;
      char x1=x & 0xf0;
      x1=x1+y;
      PORTD=x1;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      x=x<<4 | x>>4;
      x=x & 0xf0;
      PORTD=x+y;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
}

void lcd_data(char x)
{
      PORTD=PORTD | (1<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_command(char x)
{
      PORTD=PORTD | (0<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_init (void)
{
      _delay_ms(40);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x20;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
```

```
      lcd_command(0x28);
      lcd_command(0x0c);
      lcd_command(0x01);
      _delay_ms(500);
      lcd_command(0x06);
}

void LCD_WRITE_GOOD(){
    lcd_command(0x01);
    _delay_ms(3);
    lcd_data('C');
    lcd_data('L');
    lcd_data('E');
    lcd_data('A');
    lcd_data('R');
}

void LCD_WRITE_BAD() {
    lcd_command(0x01);
    _delay_ms(3);
    lcd_data('G');
    lcd_data('A');
    lcd_data('S');
    lcd_data(' ');
    lcd_data('D');
    lcd_data('E');
    lcd_data('T');
    lcd_data('E');
    lcd_data('C');
    lcd_data('T');
    lcd_data('E');
    lcd_data('D');
}

int main() {
    DDRD |=0b11111111; //I/O set
    DDRC |=0b00000000;
    DDRB |=0b11111111;

    ADMUX |= 0b01000011; //ADLAR=0 -> left adjusted
    ADCSRA |=0b10000111; // ADC enable + enable interrupt

    lcd_init(); //initialize lcd screen
    unsigned int temp=0;
    unsigned int lvl=0;

    while(1){
        //enable interrupt
        while ((ADCSRA & (1 << ADSC)) != 0) //stuck here till
conversion ends (ADSC = 0)
        {
```

```
        }
        temp=ADC;
        if((temp>=0) & (temp<215)){ //find different levels of gas
            lvl=0;
            LCD_WRITE_GOOD();
        }
        if((temp>=215) & (temp<256)){
            lvl=1;
            LCD_WRITE_BAD();
        }
        if((temp>=256) & (temp<512)){
            lvl=3;
            LCD_WRITE_BAD();
        }
        if((temp>=512) & (temp<768)){
            lvl=7;
            LCD_WRITE_BAD();
        }
        if((temp>=768) & (temp<783)){
            lvl=15;
            LCD_WRITE_BAD();
        }
        if((temp>=783) & (temp<831)){
            lvl=31;
            LCD_WRITE_BAD();
        }
        if((temp>=831) & (temp<1024)){
            lvl=63;
            LCD_WRITE_BAD();
        }
        PORTB = lvl; //blink according to the measured gas level
        _delay_ms(200);
        PORTB = 0x00;
        _delay_ms(200);
        ADCSRA |= (1<<ADSC);   //enable new conversion
    }
}
```

<u>3η Άσκηση:</u> -> micro_lab04_ex03.c

Το ζητούμενο πρόγραμμα παράγει μία PWM κυματομορφή στον ακροδέκτη PB1, σε διαφορετικό duty cycle, ανάλογα με το PB2-PB5 πλήκτρο που είναι πατημένο. Οι αναμενόμενες τιμές τάσης δίνονται στον κάτωθι πίνακα:

ΠΛΗΚΤΡΟ	D.C.	ΤΑΣΗ ADC
PB2	20%	1V
PB3	40%	2V
PB4	60%	3V

PB5	80%	4V

Η λήψη της μέτρησης από τον ADC A1 γίνεται εντός του κυρίως προγράμματος, γράφοντας λογικό 1 στον καταχωρητή ADSC και αναμένοντας να μηδενισθεί η εν λόγω τιμή.

C Program:

```
#define F_CPU 16000000UL  //running
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>

//given lcd functions written in C
void write_2_nibbles(char x)
{
      char y=PIND & 0x0f;
      char x1=x & 0xf0;
      x1=x1+y;
      PORTD=x1;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      x=x<<4 | x>>4;
      x=x & 0xf0;
      PORTD=x+y;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
}

void lcd_data(char x)
{
      PORTD=PORTD | (1<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_command(char x)
{
      PORTD=PORTD | (0<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_init (void)
{
      _delay_ms(40);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
```

```
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x20;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      lcd_command(0x28);
      lcd_command(0x0c);
      lcd_command(0x01);
      _delay_ms(500);
      lcd_command(0x06);
}

void lcd_init_2 (void) { //skips screen clear with 0x01 instruction
      _delay_ms(2);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x20;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      lcd_command(0x28);
      lcd_command(0x0c);
      _delay_ms(1);
      lcd_command(0x06);
}


void lcd_print_2(unsigned char a, unsigned char b, unsigned char c){
    lcd_command(0xC0); //start writing from initial address of second
lcd line
    _delay_ms(3);
    lcd_data(a);
    lcd_data('.');
    lcd_data(b);
    lcd_data(c);
}

void adc_display(void)
{
    unsigned int temp2;
    unsigned char akeraios, prwto, deutero;
```

```
    temp2 = (ADC*5)/1024; //ADC 2-decimal conversion
    akeraios = temp2 + '0';
    temp2  = (((ADC*5)%1024)*10)/1024;
    prwto = temp2 +'0';
    temp2  = ((((ADC*5)%1024)*10)%1024)*10/1024;
    deutero = temp2 +'0';
    lcd_print_2(akeraios, prwto, deutero); //to print on the second
row
    _delay_ms(700);
}
int main(){

    //fast PWM mode and pre-scaler at 8
    TCCR1A = (1<<WGM11)| (0<<WGM10) | (1<<COM1A1);
    TCCR1B = (1<<WGM12) | (1<<CS11) | (1<<WGM13);

    DDRD |= 0b11111111;  // output for LCD
    DDRC |= 0b00000000;  // input for ADC
    DDRB |= 0b00000010;  // output for counter

    ADMUX |= 0b01000001; //ADLAR=0 -> left adjusted
    ADCSRA |=0b10000111; // ADC enable + enable interrupt

    ICR1=399;
    lcd_init();
    unsigned int temp=0;

    while(1){
        asm("NOP");
        temp=~PINB;
        /*while ((ADCSRA & (1 << ADSC)) != 0) //stuck here till
converion ends (ADSC = 0)
        {

        }*/
        temp=PINB;
        asm("NOP");
        if(temp==0xB9){ //cases
            asm("NOP");
            OCR1A=51+20; //duty
            ICR1=399; //frequency
            lcd_command(0x01);
             _delay_ms(3);
            lcd_data(2+'0');
            lcd_data(0+'0');
            lcd_data('%');
            while ((ADCSRA & (1 << ADSC)) != 0){}
            adc_display();
            temp=PINB;

        }
        if(temp==0xB5){
```

```
            OCR1A=102+40;
            ICR1=399;
            lcd_command(0x01);
             _delay_ms(3);
            lcd_data(4+'0');
            lcd_data(0+'0');
            lcd_data('%');
            while ((ADCSRA & (1 << ADSC)) != 0){}
            adc_display();
        }
        if(temp==0xAD){
             OCR1A=153+70;
             ICR1=399;
             lcd_command(0x01);
             _delay_ms(3);
             lcd_data(6+'0');
             lcd_data(0+'0');
             lcd_data('%');
             while ((ADCSRA & (1 << ADSC)) != 0){}
             adc_display();
        }
        if(temp==0x9D){
            OCR1A=320;
            ICR1=400;
            lcd_command(0x01);
            _delay_ms(3);
            lcd_data(8+'0');
            lcd_data(0+'0');
            lcd_data('%');
            while ((ADCSRA & (1 << ADSC)) != 0){}
            adc_display();
        }
        if(temp==0xBD){
            OCR1A=1;
            lcd_command(0x01);
            _delay_ms(3);
            lcd_data(0+'0');
            lcd_data(0+'0');
            lcd_data('%');
            while ((ADCSRA & (1 << ADSC)) != 0){}
            adc_display();
        }
        lcd_command(0x01);
        _delay_ms(3);

        ADCSRA |= (1<<ADSC);   //enable new conversion
}
    return 0;
}
```