Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

ΑΘΗΝΑ, 12 Νοεμβρίου 2022

4η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"
Χρήση ADC και Οθόνης 2×16 Χαρακτήρων στον AVR

Αναφορά 4ης Εργαστηριακής Άσκησης

Ραπτόπουλος Πέτρος (el19145)
Σαφός Κωνσταντίνος (el19172)

Ζήτημα 4.1: Παρακάτω φαίνεται ο κώδικας σε Assembly και C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

(α) Κώδικας σε Assembly

```
.include "m328PBdef.inc"

.equ FOSC_MHZ = 16     ; Microcontroller operating frequency in MHz
.EQU PD3 = 3
.EQU PD2 = 2
.DEF temp2=r18
.DEF counter=r21
.DEF temp=r22
.DEF argL=r24          ; used as argument
.DEF argH=r25          ; used as argument
.def ADC_L = r19
.def ADC_H = r20

.org 0x0
rjmp reset
.org 0x2A                 ;ADC Conversion Complete Interrupt
rjmp ISR_ADC

reset:
    ldi temp, LOW(RAMEND) ;Initialize stack pointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ser temp
    out DDRD, temp ; init PORTD (connected to lcd) as output
    out DDRB, temp ; set PORTB as output

    clr temp
    out DDRC, temp ;Set PORTC as input

    clr counter    ; set counter to zero
    ; REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0010 => select ADC2(pin PC2),
    ; ADLAR=1 => Left adjust the ADC result
    ldi temp, 0b01100010 ;
    sts ADMUX, temp

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

    rcall lcd_init ; init lcd display
    ldi argL, low(2*FOSC_MHZ)
    ldi argH, high(2*FOSC_MHZ) ; wait 2 msec
    rcall wait_msec

main:
    lds temp, ADCSRA
    ori temp, (1<<ADSC)     ; Set ADSC flag of ADCSRA
    sts ADCSRA, temp        ; in order to start conversion
    inc counter             ; increase counter
    out PORTB, counter      ; counter to PORTB

    ldi r24, low(1000*FOSC_MHZ)    ; wait 1 sec
    ldi r25, high(1000*FOSC_MHZ)
    rcall wait_msec

    ldi temp ,0x01                 ; clear display
    rcall lcd_command
    ldi r24 ,low(5000*FOSC_MHZ)
    ldi r25 ,high(5000*FOSC_MHZ)
    rcall wait_usec
    rjmp main                      ; loop forever
```

```
; delay of 4*F1 + 6 cycles (almost equal to F1 cycles)
wait_usec:
    sbiw r24, 1          ; 2 cycles
    brne wait_usec       ; 1 or 2 cycles
    ret                  ; 4 cycles

; delay of 1000*F1 + 6 cycles (almost equal to 1000*F1 cycles)
wait_msec:
; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
    ldi r23, 249
loop_inn:
    dec r23          ; 1 cycle
    nop              ; 1 cycle
    brne loop_inn    ; 1 or 2 cycles
    sbiw r24, 1      ; 2 cycles
    brne wait_msec   ; 1 or 2 cycles
    ret              ; 4 cycles

; send one byte divided into 2 (4 bit) parts
write_2_nibbles:     ; uses register r24 as argument
    push r24         ; send the 4 MSB
    in r25 ,PIND     ; read 4 LSB and resend them
    andi r25 ,0x0f   ; in order not to alter any previous state
    andi r24 ,0xf0   ; isolate the 4 MSB
    add r24 ,r25     ; combine them with the preexisting 4 LSB
    out PORTD ,r24   ; send them to output
    sbi PORTD ,PD3   ; create enable pulse at pin PD3
    cbi PORTD ,PD3   ; PD3=1 and after PD3=0
    nop              ; do nothing (for delay purposes)
    nop              ; do nothing
    pop r24          ; send the 4 LSB. restore saved byte
    swap r24         ; swap 4 MSB with 4 LSB
    andi r24 ,0xf0   ; send them
    add r24 ,r25
    out PORTD ,r24
    sbi PORTD ,PD3   ; new Enable pulse
    cbi PORTD ,PD3
    nop              ; do nothing
    nop              ; do nothing
    ret

; send one byte of data to the lcd display. uses register r24 as argument
lcd_data:
    sbi PORTD ,PD2               ; select data register (PD2=1)
    rcall write_2_nibbles        ; send byte
    ldi r24, low(100*FOSC_MHZ)   ; wait 100?s in order the lcd to receive
    ldi r25 ,high(100*FOSC_MHZ) ; the data successfully
    rcall wait_usec
    ret

; send one byte of instuction to the lcd display. uses register r24 as argument
lcd_command:
    cbi PORTD ,PD2                ; select instuction register (PD2=0)
    rcall write_2_nibbles         ; send byte
    ldi r24 ,low(100*FOSC_MHZ)    ;  wait 100?s in order the lcd to implement
    ldi r25 ,high(100*FOSC_MHZ)   ; the instruction successfully
    rcall wait_usec               ; Note: "clear display" and "return home" instructions
    ret                           ; need more time

lcd_init:
    ldi r24 ,low(40*FOSC_MHZ)     ; when we power on the lcd, it begins
    ldi r25 ,high(40*FOSC_MHZ)    ; its own initialization sequence
    rcall wait_msec               ; wait 40 msec until the initialization is complete

    ldi r24 ,0x30                 ; instruction to transition to 8 bit mode
    out PORTD ,r24                ; send the instruction twice
    sbi PORTD ,PD3                ; because we can not be sure whether at first the
    cbi PORTD ,PD3                ; input mode of the lcd is 4 bit or 8 bit.
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)   ; if lcd controller was at 8-bit mode
    rcall wait_usec               ; nothing happens, but if it had input mode 4-bit
                                  ; it will transition to 8 bit
```

```
    ldi r24 ,0x30
    out PORTD ,r24
    sbi PORTD ,PD3
    cbi PORTD ,PD3
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_usec

    ldi r24 ,0x20    ; transition to 4-bit mode
    out PORTD ,r24
    sbi PORTD ,PD3
    cbi PORTD ,PD3
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_usec

    ldi r24 ,0x28     ; select character size 5x8 dots
    rcall lcd_command ; and two line display

    ldi r24 ,0x0c     ; enable lcd, hide cursor
    rcall lcd_command

    ldi r24 ,0x01     ; clear display
    rcall lcd_command
    ldi r24 ,low(5000*FOSC_MHZ)
    ldi r25 ,high(5000*FOSC_MHZ)
    rcall wait_usec

    ldi r24 ,0x06     ; enable auto increment of address
    rcall lcd_command ; disable shift of the display
    ret

ISR_ADC:
    ; interrupt routine for ADC
    push r24      ; save r24
    push r25      ; save r25
    push temp     ; save temp
    in temp, SREG ; save SREG
    push temp

    lds ADC_L,ADCL ; Read ADC result(Left adjusted)
    lds ADC_H,ADCH

    mov temp, ADC_H ; hold copy of ADC data
    mov temp2, ADC_L

    ; we want to multiply ADC data with 5
    ; in order to do this shift ADC 2 times to the right
    ; and add that to the original ADC data
    ; keep the 3 MSB (the carry and the 2MSB of temp)

    ror temp         ; rotate ADC_H, ADC_H(0)->C
    ror temp2        ; rotate ADC_L, C->ADC_L(7)
    ror temp         ; rotate ADC_H, ADC_H(0)->C
    ror temp2        ; rotate ADC_L, C->ADC_L(7)
    andi temp, 0x3F  ; isolate desired bits
    andi temp2, 0xF0
    add temp2, ADC_L
    adc temp, ADC_H
    ; now the carry and the 2 MSB of temp holds
    ; the integer part of the Vin
    mov ADC_L, temp2 ; hold copy of the values
    mov ADC_H, temp
    rol temp
    rol temp
    rol temp
    andi temp, 0x07  ; now the 3LSB of temp hold
                     ; the integer part of the division
    ; lcd code for 0: 0x30, for 1:0x31...
    subi temp, -(0x30) ; addi temp, 0x30
    mov r24, temp      ; argument to lcd_data
    rcall lcd_data     ; send one byte of data to the lcd's controller
```

```
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 0x2E  ; display dot
    rcall lcd_data

    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_msec

     ; ---------------- 1st decimal ----------------------

    andi ADC_H, 0x3F ; remove the integer part of Vin
    mov temp2, ADC_L ; fetch copy of the values
    mov temp, ADC_H

    ror temp         ; rotate ADC_H, ADC_H(0)->C
    ror temp2        ; rotate ADC_L, C->ADC_L(7)
    ror temp         ; rotate ADC_H, ADC_H(0)->C
    ror temp2        ; rotate ADC_L, C->ADC_L(7)
    andi temp, 0x0F  ; isolate desired bits
    andi temp2, 0xFC
    add temp2, ADC_L
    adc temp, ADC_H
    mov ADC_L, temp2 ; hold copy of the values
    mov ADC_H, temp

    ror temp
    ror temp
    ror temp
    andi temp, 0x0F
    subi temp, -(0x30)  ; addi temp, 0x30
    mov r24, temp       ; argument to lcd_data
    rcall lcd_data      ; send one byte of data to the lcd's controller

    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_msec

    ; ---------------- 2nd decimal ----------------------
    andi ADC_H, 0x07 ; remove the integer part of Vin
    mov temp2, ADC_L ; fetch copy of the values
    mov temp, ADC_H

    ror temp         ; rotate ADC_H, ADC_H(0)->C
    ror temp2        ; rotate ADC_L, C->ADC_L(7)
    ror temp         ; rotate ADC_H, ADC_H(0)->C
    ror temp2        ; rotate ADC_L, C->ADC_L(7)
    andi temp, 0x0F  ; isolate desired bits
    andi temp2, 0xFF
    add temp2, ADC_L
    adc temp, ADC_H
    mov ADC_L, temp2 ; hold copy of the values
    mov ADC_H, temp

    andi temp, 0x0F
    subi temp, -(0x30) ; addi temp, 0x30
    mov r24, temp      ; argument to lcd_data
    rcall lcd_data     ; send one byte of data to the lcd's controller

    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_msec

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

    pop temp
    out SREG, temp  ; restore SREG
    pop temp         ; restore temp
    pop r25          ; restore r25
    pop r24          ; restore r24
    reti             ; return to callee
```

(β) Κώδικας σε C

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

// send one byte divided into 2 (4 bit) parts
void write_2_nibbles(char data) {
    char pinState = PIND; // read 4 LSB and resend them
                          // in order not to alter any previous state
    PORTD = (pinState & 0x0F) | (data & 0xF0) | (1<<PD3); // send 4MSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    PORTD = (pinState & 0x0F) | ((data<<4) & 0xF0) | (1<<PD3); // send 4LSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
}

// send one byte of data to the lcd display
void lcd_data(char data) {
    PORTD |= (1<<PD2);  // select data register
    write_2_nibbles(data);
    _delay_us(100);
}

// send one byte of instuction to the lcd display
void lcd_command(char data) {
    PORTD &= (0xFF) & (0<<PD2);  // select command register
    write_2_nibbles(data);
    _delay_us(100);
}

void lcd_init() {
    _delay_ms(40);                // lcd init procedure
    PORTD = 0x30 | (1<<PD3);      // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x30 | (1<<PD3);      // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x20 | (1<<PD3);      // change to 4 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    lcd_command(0x28); // select character size 5x8 dots and two line display
    lcd_command(0x0c); // enable lcd, hide cursor
    lcd_command(0x01); // clear display
    _delay_us(5000);
    lcd_command(0x06); // enable auto increment of address, disable shift of the display
}

void read_adc () {
    int adc = ADC;
    double vin = adc*5.0/1024;

    char temp = vin; // integer part of vin
    temp += 0x30;
    lcd_data(temp);
    _delay_ms(100);

    lcd_data(0x2E); // dot
    _delay_ms(100);

    temp = ((int)(vin*10)%10);  // 1st decimal
    temp += 0x30;
    lcd_data(temp);
    _delay_ms(100);

    temp = ((int)(vin*100)%10); // 2nd decimal
    temp += 0x30;
    lcd_data(temp);
    _delay_ms(100);
}
```

```c
int main() {
    int counter = 0;
    DDRD = 0xFF;    // set portD as output
    DDRB = 0xFF;    // set portB as output
    DDRC = 0x00;    // set portB as input

    // REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0010 => select ADC2(pin PC2),
    // ADLAR=0 => Right adjust the ADC result
    ADMUX = 0b01000010;
    // ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
    // ADIE=0 => disable adc interrupt, ADPS[2:0]=111 => fADC=16MHz/128=125KHz
    ADCSRA = 0b10000111;
    lcd_init();     // init lcd
    _delay_ms(2);   // wait for lcd init

    while(1) {
        ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
                             // enable conversion
        counter++;           // increase counter
        PORTB = counter;     // output counter
        while((ADCSRA & (1<<ADSC)) == (1<<ADSC));
        // wait until flags become zero
        // that means that the conversion is complete
        read_adc();

        _delay_ms(1000);
        lcd_command(0x01);  // clear display
        _delay_us(5000);
    }

}
```

Ζήτημα 4.2: Παρακάτω φαίνεται ο κώδικας σε Assembly και C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

(α) Κώδικας σε Assembly

```
.include "m328PBdef.inc"

.equ FOSC_MHZ = 16   ; Microcontroller operating frequency in MHz
.EQU PD3 = 3
.EQU PD2 = 2

.DEF temp2=r18
.DEF counter=r21
.DEF temp=r22
.DEF argL=r24          ; used as argument
.DEF argH=r25          ; used as argument
.def ADC_L = r19
.def ADC_H = r20

.org 0x0
rjmp reset
.org 0x2A                   ;ADC Conversion Complete Interrupt
rjmp ISR_ADC

reset:
    ldi temp, LOW(RAMEND)   ;Initialize stack pointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ser temp
    out DDRD, temp ; init PORTD (connected to lcd) as output
    out DDRB, temp ; set PORTB as output

    clr temp
    out DDRC, temp ;Set PORTC as input

    ; REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0011 => select ADC3(pin PC3),
    ; ADLAR=1 => Left adjust the ADC result
    ldi temp, 0b01100011 ;
    sts ADMUX, temp

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

    rcall lcd_init ; init lcd display
    ldi argL, low(2*FOSC_MHZ)
    ldi argH, high(2*FOSC_MHZ)
    rcall wait_msec

main:

    lds temp, ADCSRA
    ori temp, (1<<ADSC)             ; Set ADSC flag of ADCSRA
    sts ADCSRA, temp                ; in order to start conversion

    ldi r24, low(100*FOSC_MHZ)      ; wait 100 msec
    ldi r25, high(100*FOSC_MHZ)
    rcall wait_msec

    ldi temp ,0x01 ; clear display
    rcall lcd_command
    ldi r24 ,low(5000*FOSC_MHZ)
    ldi r25 ,high(5000*FOSC_MHZ)
    rcall wait_usec

    rjmp main
```

```
; delay of 4*F1 + 6 cycles (almost equal to F1 cycles)
wait_usec:
    sbiw r24, 1          ; 2 cycles
    brne wait_usec       ; 1 or 2 cycles
    ret                  ; 4 cycles

; delay of 1000*F1 + 6 cycles (almost equal to 1000*F1 cycles)
wait_msec:
; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
    ldi r23, 249
loop_inn:
    dec r23         ; 1 cycle
    nop             ; 1 cycle
    brne loop_inn   ; 1 or 2 cycles
    sbiw r24, 1     ; 2 cycles
    brne wait_msec  ; 1 or 2 cycles
    ret             ; 4 cycles

; send one byte divided into 2 (4 bit) parts
write_2_nibbles:    ; uses register r24 as argument
    push r24        ; send the 4 MSB
    in r25 ,PIND    ; read 4 LSB and resend them
    andi r25 ,0x0f  ; in order not to alter any previous state
    andi r24 ,0xf0  ; isolate the 4 MSB
    add r24 ,r25    ; combine them with the preexisting 4 LSB
    out PORTD ,r24  ; send them to output
    sbi PORTD ,PD3  ; create enable pulse at pin PD3
    cbi PORTD ,PD3  ; PD3=1 and after PD3=0
    nop              ; do nothing (for delay purposes)
    nop              ; do nothing
    pop r24          ; send the 4 LSB. restore saved byte
    swap r24         ; swap 4 MSB with 4 LSB
    andi r24 ,0xf0  ; send them
    add r24 ,r25
    out PORTD ,r24
    sbi PORTD ,PD3   ; new Enable pulse
    cbi PORTD ,PD3
    nop                  ; do nothing
    nop                  ; do nothing
    ret

; send one byte of data to the lcd display. uses register r24 as argument
lcd_data:
    sbi PORTD ,PD2                  ; select data register (PD2=1)
    rcall write_2_nibbles           ; send byte
    ldi r24, low(100*FOSC_MHZ)      ; wait 100?s in order the lcd to receive
    ldi r25 ,high(100*FOSC_MHZ)     ; the data successfully
    rcall wait_usec
    ret

; send one byte of instuction to the lcd display. uses register r24 as argument
lcd_command:
    cbi PORTD ,PD2                  ; select instuction register (PD2=0)
    rcall write_2_nibbles           ; send byte
    ldi r24 ,low(100*FOSC_MHZ)      ; wait 100micros in order the lcd to implement
    ldi r25 ,high(100*FOSC_MHZ)     ; the instruction successfully
    rcall wait_usec                 ; Note: "clear display" and "return home" instructions
    ret                             ; need more time

lcd_init:
    ldi r24 ,low(40*FOSC_MHZ)       ; when we power on the lcd, it begins
    ldi r25 ,high(40*FOSC_MHZ)      ; its own initialization sequence
    rcall wait_msec                 ; wait 40 msec until the initialization is complete

    ldi r24 ,0x30                   ; instruction to transition to 8 bit mode
    out PORTD ,r24                  ; send the instruction twice
    sbi PORTD ,PD3                  ; because we can not be sure whether at first the
    cbi PORTD ,PD3                  ; input mode of the lcd is 4 bit or 8 bit.
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)     ; if lcd controller was at 8-bit mode
    rcall wait_usec                 ; nothing happens, but if it had input mode 4-bit
                                    ; it will transition to 8 bit
```

```
    ldi r24 ,0x30
    out PORTD ,r24
    sbi PORTD ,PD3
    cbi PORTD ,PD3
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_usec

    ldi r24 ,0x20           ; transition to 4-bit mode
    out PORTD ,r24
    sbi PORTD ,PD3
    cbi PORTD ,PD3
    ldi r24 ,low(100*FOSC_MHZ)
    ldi r25 ,high(100*FOSC_MHZ)
    rcall wait_usec

    ldi r24 ,0x28           ; select character size 5x8 dots
    rcall lcd_command       ; and two line display

    ldi r24 ,0x0c           ; enable lcd, hide cursor
    rcall lcd_command

    ldi r24 ,0x01           ; clear display
    rcall lcd_command
    ldi r24 ,low(5000*FOSC_MHZ)
    ldi r25 ,high(5000*FOSC_MHZ)
    rcall wait_usec

    ldi r24 ,0x06           ; enable auto increment of address
    rcall lcd_command       ; disable shift of the display
    ret

ISR_ADC:
    ; interrupt routine for ADC
    push r24       ; save r24
    push r25       ; save r25
    push temp      ; save temp
    in temp, SREG ; save SREG
    push temp

    lds ADC_L,ADCL ; Read ADC result(Left adjusted)
    lds ADC_H,ADCH
    mov temp, ADC_H ; hold copy of ADC data
    mov temp2, ADC_L

    ; we want to multiply ADC data with 5
    ; in order to do this shift ADC 2 times to the right
    ; and add that to the original ADC data
    ; keep the 3 MSB (the carry and the 2MSB of temp)
    ror temp      ; rotate ADC_H, ADC_H(0)->C
    ror temp2     ; rotate ADC_L, C->ADC_L(7)
    ror temp      ; rotate ADC_H, ADC_H(0)->C
    ror temp2     ; rotate ADC_L, C->ADC_L(7)
    andi temp, 0x3F  ; isolate desired bits
    andi temp2, 0xF0
    add temp2, ADC_L
    adc temp, ADC_H
    ; now the carry and the 2 MSB of temp holds
    ; the integer part of the Vin
    mov ADC_L, temp2 ; hold copy of the values
    mov ADC_H, temp
    rol temp
    rol temp
    rol temp
    andi temp, 0x07 ; now the 3LSB of temp hold the integer part of the division
    ; Cx: gas concentration
    ; Cx = (Vgas - Vgas0)/M, Vgas0 = 0.1V
    ; M = Sensitivity code*TIA Gain*10^(-9)*10^(3)
    ; Sensitivity code = 129nA/ppm
    ; TIA Gain = 100(kV/A)
    ; So for Cx = 70ppm we have approx. Vgas = 1V
    ; 1st Gas Level: Vgas in [0.1...1), 2nd Gas Level: Vgas in [1...2), 3rd Gas Level: Vgas in [2...3)
    ; 4th Gas Level: Vgas in [3...4), 5th Gas Level: Vgas in [4...4.5), 6th Gas Level: Vgas in [4.9...5]
```

```
Level_1:
    cpi temp, 0x01
    brsh Level_2    ; branch if same or higher
    ldi temp2, 0x01
    out PORTB, temp2 ; first led on

    ldi r24, 'C'
    rcall lcd_data ; send one byte of data to the lcd's controller
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'L'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'E'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'A'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'R'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec
    rjmp end_routine

Level_2:
    cpi temp, 0x02
    brsh Level_3    ; branch if same or higher
    ldi temp2, 0x02
    out PORTB, temp2 ; 2nd led on
    rjmp display

Level_3:
    cpi temp, 0x03
    brsh Level_4    ; branch if same or higher
    ldi temp2, 0x04
    out PORTB, temp2 ; 3rd led on
    rjmp display

Level_4:
    cpi temp, 0x04
    brsh Level_5    ; branch if same or higher
    ldi temp2, 0x08
    out PORTB, temp2 ; 4th led on
    rjmp display

Level_5:
    ; ---------------- 1st decimal -----------------------
    andi ADC_H, 0x3F ; remove the integer part of Vin
    mov temp2, ADC_L ; fetch copy of the values
    mov temp, ADC_H

    ror temp     ; rotate ADC_H, ADC_H(0)->C
    ror temp2    ; rotate ADC_L, C->ADC_L(7)
    ror temp     ; rotate ADC_H, ADC_H(0)->C
    ror temp2    ; rotate ADC_L, C->ADC_L(7)
    andi temp, 0x0F  ; isolate desired bits
    andi temp2, 0xFC
    add temp2, ADC_L
    adc temp, ADC_H
```

```
    ror temp
    ror temp
    ror temp
    andi temp, 0x0F ; temp holds the 1st decimal
    ; -------------------------------------------------------
    cpi temp, 0x09
    brsh Level_6    ; branch if same or higher
    ldi temp2, 0x10
    out PORTB, temp2 ; 5th led on
    rjmp display

Level_6:
    ldi temp2, 0x20
    out PORTB, temp2 ; 6th led on

display:
    ldi r24, 'G'
    rcall lcd_data ; send one byte of data to the lcd's controller
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'A'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'S'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, ' '
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'D'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'E'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'T'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'E'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'C'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'T'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
```

```asm
    rcall wait_msec

    ldi r24, 'E'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi r24, 'D'
    rcall lcd_data
    ldi r24 ,low(10*FOSC_MHZ) ; delay from 100 to 10
    ldi r25 ,high(10*FOSC_MHZ)
    rcall wait_msec

    ldi temp2, 0x00
    out PORTB, temp2 ; lights off

end_routine:
    pop temp
    out SREG, temp  ; restore SREG
    pop temp       ; restore temp
    pop r25        ; restore r25
    pop r24        ; restore r24
    reti           ; return to callee
```

(β) Κώδικας σε C

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

// send one byte divided into 2 (4 bit) parts
void write_2_nibbles(char data) {
    char pinState = PIND; // read 4 LSB and resend them
                          // in order not to alter any previous state
    PORTD = (pinState & 0x0F) | (data & 0xF0) | (1<<PD3); // send 4MSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    PORTD = (pinState & 0x0F) | ((data<<4) & 0xF0) | (1<<PD3); // send 4LSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
}

// send one byte of data to the lcd display
void lcd_data(char data) {
    PORTD |= (1<<PD2);  // select data register
    write_2_nibbles(data);
    _delay_us(100);
}

// send one byte of instuction to the lcd display
void lcd_command(char data) {
    PORTD &= (0xFF) & (0<<PD2);  // select command register
    write_2_nibbles(data);
    _delay_us(100);
}

void lcd_init() {
    _delay_ms(40);  // lcd init procedure
    PORTD = 0x30 | (1<<PD3);     // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x30 | (1<<PD3);     // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x20 | (1<<PD3);     // change to 4 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    lcd_command(0x28); // select character size 5x8 dots and two line display
    lcd_command(0x0c); // enable lcd, hide cursor
    lcd_command(0x01); // clear display
    _delay_us(5000);
    lcd_command(0x06); // enable auto increment of address, disable shift of the display
}
```

```c
void read_adc () {
    int adc = ADC;
    double vin = adc*5.0/1024;

    // Cx: gas concentration
    // Cx = (Vgas - Vgas0)/M, Vgas0 = 0.1V
    // M = Sensitivity code*TIA Gain*10^(-9)*10^(3)
    // Sensitivity code = 129nA/ppm
    // TIA Gain = 100(kV/A)
    // So for Cx = 70ppm we have approx. Vgas = 1V
    // 1st Gas Level: Vgas in [0.1...1)
    // 2nd Gas Level: Vgas in [1...2)
    // 3rd Gas Level: Vgas in [2...3)
    // 4th Gas Level: Vgas in [3...4)
    // 5th Gas Level: Vgas in [4...4.5)
    // 6th Gas Level: Vgas in [4.9...5]

    if (vin < 1) {
        PORTB = 0x01;  // first led on
        lcd_data('C');
        _delay_ms(10); // delay time decreased from 100 to 10
        lcd_data('L');
        _delay_ms(10);
        lcd_data('E');
        _delay_ms(10);
        lcd_data('A');
        _delay_ms(10);
        lcd_data('R');
        _delay_ms(10);
    } else {
        // light up the leds
        if (vin < 2) PORTB = 0x02;
        else if (vin < 3) PORTB = 0x04;
        else if (vin < 4) PORTB = 0x08;
        else if (vin < 4.9) PORTB = 0x10;
        else PORTB = 0x20;
        lcd_data('G');
        _delay_ms(10); // delay time decreased from 100 to 10
        lcd_data('A');
        _delay_ms(10);
        lcd_data('S');
        _delay_ms(10);
        lcd_data(' ');
        _delay_ms(10);
        lcd_data('D');
        _delay_ms(10);
        lcd_data('E');
        _delay_ms(10);
        lcd_data('T');
        _delay_ms(10);
        lcd_data('E');
        _delay_ms(10);
        lcd_data('C');
        _delay_ms(10);
        lcd_data('T');
        _delay_ms(10);
        lcd_data('E');
        _delay_ms(10);
        lcd_data('D');
        _delay_ms(10);
        PORTB = 0x00; // lights off
    }
}
```

```c
int main() {
    DDRD = 0xFF;     // set portD as output
    DDRB = 0xFF;     // set portB as output
    DDRC = 0x00;     // set portB as input

    // REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0011 => select ADC3(pin PC3),
    // ADLAR=0 => Right adjust the ADC result
    ADMUX = 0b01000011;
    // ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
    // ADIE=0 => disable adc interrupt, ADPS[2:0]=111 => fADC=16MHz/128=125KHz
    ADCSRA = 0b10000111;
    lcd_init(); // init lcd
    _delay_ms(2);    // wait for lcd init

    while(1) {
        ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
                             // enable conversion
        while((ADCSRA & (1<<ADSC)) == (1<<ADSC));
        // wait until flags become zero
        // that means that the conversion is complete
        read_adc();

        _delay_ms(100);
        lcd_command(0x01); // clear display
        _delay_us(5000);
    }

}
```

Ζήτημα 4.3:

Παρακάτω φαίνεται ο κώδικας σε C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε C

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

// send one byte divided into 2 (4 bit) parts
void write_2_nibbles(char data) {
    char pinState = PIND; // read 4 LSB and resend them
                          // in order not to alter any previous state
    PORTD = (pinState & 0x0F) | (data & 0xF0) | (1<<PD3); // send 4MSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    PORTD = (pinState & 0x0F) | ((data<<4) & 0xF0) | (1<<PD3); // send 4LSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
}

// send one byte of data to the lcd display
void lcd_data(char data) {
    PORTD |= (1<<PD2);  // select data register
    write_2_nibbles(data);
    _delay_us(100);
}

// send one byte of instruction to the lcd display
void lcd_command(char data) {
    PORTD &= (0xFF) & (0<<PD2);  // select command register
    write_2_nibbles(data);
    _delay_us(100);
}
```

```c
void lcd_init() {
    _delay_ms(40);  // lcd init procedure
    PORTD = 0x30 | (1<<PD3); // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x30 | (1<<PD3); // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x20 | (1<<PD3); // change to 4 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    lcd_command(0x28);      // select character size 5x8 dots and two line display
    lcd_command(0x28);
    lcd_command(0x0c);      // enable lcd, hide cursor
    lcd_command(0x01);      // clear display
    _delay_us(5000);
    lcd_command(0x06);      // enable auto increment of address, disable shift of the display
}

ISR (ADC_vect) {
    int adc = ADC;
    double vin = adc*5.0/1024;

    lcd_command(0b11000000); //write to second line ADC result (DDRAM Address 0x40)
    _delay_us(5000);
    lcd_command(0b11000000); //write to second line ADC result (DDRAM Address 0x40)
    _delay_us(5000);

    char temp = vin; // integer part of vin
    temp += 0x30;
    lcd_data(temp);
    _delay_ms(100);

    lcd_data(0x2E); // dot
    _delay_ms(100);

    temp = ((int)(vin*10)%10);            // 1st decimal
    temp += 0x30;
    lcd_data(temp);
    _delay_ms(100);

    temp = ((int)(vin*100)%10);           // 2nd decimal
    temp += 0x30;
    lcd_data(temp);
    _delay_ms(100);

}

int main() {
    DDRB = 0b00000010;    // set PORTB0,2-7 as input, PORTB1 output
    DDRD = 0xFF;    // Set PORTD as output
    TCCR1A = (0<<WGM10) | (1<<WGM11) | (1<<COM1A1); // Init control register A of Timer 1
    TCCR1B = (1<<WGM12) | (1<<WGM13);               // Init control register B of Timer 1
    // with the above values we have fast PWM mode with TOP = ICR1

    // REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0001 => select ADC1(pin PC1),
    // ADLAR=0 => Right adjust the ADC result
    ADMUX = 0b01000001;

    // ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
    // ADIE=1 => enable adc interrupt, ADPS[2:0]=111 => fADC=16MHz/128=125KHz
    ADCSRA = 0b10001111;
    sei();  // enable interrupts
    lcd_init();      // init lcd
    _delay_ms(2);    // wait for lcd init

    while(1) {                                        // loop forever
        TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (0<<CS11) | (0<<CS10); // when no buttons are pressed
        // we stop PWM setting timer's frequency to zero
        // we want to change only the CS12, CS11, CS10 bits of the register

        lcd_command(0x01);      //clear screen
        _delay_us(5000);
```

```
    // formula: top(fpwm, N) = fclk/(N*fpwm) - 1, fpwm = 5k, N = 64, top = 49
    if ((PINB & 0x04) == 0x00) { // check if PB2 pressed (logical 0)
        TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (1<<CS11) | (1<<CS10);    // N = 64
        ICR1H = 0;                                  //ICR=TOP
        ICR1L = 49;
        OCR1AH = 0x00;
        OCR1AL = 10;                             // DC=20%

        lcd_command(0b10000000); //write to first line DC%
        _delay_us(5000);
        lcd_data(0x32);        // 2
        _delay_ms(100);
        lcd_data(0x30);        // 0
        _delay_ms(100);
        lcd_data(0b00100101);  // %
        _delay_ms(100);

        while((PINB & 0x04) == 0x00) {
            ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
            _delay_ms(10);
        }

    }
    else if ((PINB & 0x08) == 0x00) {           // check if PB3 pressed (logical 0)
        TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (1<<CS11) | (1<<CS10);
        ICR1H = 0;
        ICR1L = 49;
        OCR1AH = 0x00;
        OCR1AL = 20;                             // DC=40%

        lcd_command(0b10000000); //write to first line DC%
        _delay_us(5000);
        lcd_data(0x34);
        _delay_ms(100);
        lcd_data(0x30);
        _delay_ms(100);
        lcd_data(0b00100101);
        _delay_ms(100);

        while((PINB & 0x08) == 0x00) {
            ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
            _delay_ms(10);
        }
    }
    else if ((PINB & 0x10) == 0x00) {           // check if PB4 pressed (logical 0)
        TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (1<<CS11) | (1<<CS10);
        ICR1H = 0;                                  //ICR=TOP
        ICR1L = 49;
        OCR1AH = 0x00;
        OCR1AL = 30;                             // DC=60%

        lcd_command(0b10000000); //write to first line DC%
        _delay_us(5000);
        lcd_data(0x36);
        _delay_ms(100);
        lcd_data(0x30);
        _delay_ms(100);
        lcd_data(0b00100101);
        _delay_ms(100);

        while((PINB & 0x10) == 0x00) {
            ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
            _delay_ms(10);
        }
    }
    else if ((PINB & 0x20) == 0x00) {           // check if PB5 pressed (logical 0)
        TCCR1B = (1<<WGM12) | (1<<WGM13) | (0<<CS12) | (1<<CS11) | (1<<CS10);
        ICR1H = 0;
        ICR1L = 49;
        OCR1AH = 0x00;                            // values from 0 to 255
        OCR1AL = 40;                             // DC=80%

        lcd_command(0b10000000); //write to first line DC%
```

```
            _delay_us(5000);
            lcd_data(0x38);
            _delay_ms(100);
            lcd_data(0x30);
            _delay_ms(100);
            lcd_data(0b00100101);
            _delay_ms(100);

            while((PINB & 0x20) == 0x00) {
                ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
                _delay_ms(10);
            }
        }
    }
}
```