4η εργαστηριακή άσκηση

Ιωάννης Μπουφίδης 03120162
Φωτεινή Κυριακοπούλου 03120182

Ζήτημα 4.1

```
.include "m328PBdef.inc" ; ATmega328P microcontroller definition

.equ PD0=0
.equ PD1=1
.equ PD2=2
.equ PD3=3
.equ PD4=4
.equ PD5=5
.equ PD6=6
.equ PD7=7

.def quotient = r23

.org 0x00
rjmp main
.org 0x2A                       ; ADC interrupt vector
rjmp ADC_int

main:
 ldi r24, low(RAMEND)
 out SPL, r24
 ldi r24, high(RAMEND)
 out SPH, r24                   ; init stack pointer

 ser r24
 out DDRD, r24                  ; set PORTD as output

 ldi r24, 0b01000010            ; right adjusted, ADC2 channel, Avcc voltage ref.
 sts ADMUX, r24
 ldi r24, 0b10001111            ; ADEN = 1 (enable ADC circuit)
 sts ADCSRA, r24                ; ADIE =1 (enable interrupt), ADPS = 128

 clr r24
 rcall lcd_init                 ; initialize display
 ldi r24, low(100)
 ldi r25, high(100)             ; delay 100 mS
 rcall wait_msec
 sei
main1:
 rcall lcd_clear_display        ; clear display

 lds r24, ADCSRA
 ori r24, (1<<ADSC)             ; set ADCS flag
 sts ADCSRA, r24                ; start ADC conversion

 ldi r24, low(1000)
```

```
 ldi r25, high(1000)
 rcall wait_msec              ; delay 1 Sec

 rjmp main

ADC_int:
 ; adc conversion result (right-adjusted)
 lds r30, ADCL
 lds r31, ADCH

 ; Vadc = (5*ADC/1024) * 100 to get two decimals

 ; calculate in r26:r27 -> 12*ADC/1024 = 3*ADC/256
 mov r26, r30
 mov r27, r31
 clr r17
 rcall multiplier             ; multiply ADC * 3
 ldi r17, 7                   ; divide by 256 - shift right 8 times

shift:
    lsr r27
    ror r26
    dec r17
    cpi r17,0
    breq exit
    rjmp shift
exit:

                              ; now we calculate 512*ADC/1024
 lsr r31
 ror r30                      ; divide ADC / 2 - shift right one time

 ; Vadc = 512*ADC/1024 - 12*ADC/1024 ;
 sub r30, r26
 sbc r31, r27

 ldi quotient, 0
 ldi r18, 100
 ldi r17, 0
 rcall divide                 ; divide Vadc by 100 to get whole part
                              ; fractional part in remainder r30

 ldi r24, '0'
 add r24, quotient
 rcall lcd_data               ; display data

 ldi r31,0
 ldi r18,10
 ldi quotient, 0
 rcall divide                ; divide fractional part by 10 to get first decimal digit
                             ; second digit is the remainder r30

 ldi r24, '.'
```

```
 rcall lcd_data

 ldi r24, '0'
 add r24, quotient
 rcall lcd_data

 ldi r24, '0'
 add r24, r30
 rcall lcd_data

 reti


; multiply by 3 a 16-bit value in registers r26:r27
multiplier:
    ldi r28, 3
loop2:
    dec r28
    cpi r28,0
    breq exit3
    clc
    add r26, r26
    adc r27, r17
    rjmp loop2
exit3:
    ret

; divide a 16-bit value in registers r30:r31
divide:
    cp r30, r18
    brsh loop
    cpi r31,0
    brne loop
    ldi quotient,0
    rjmp exit_now
loop:
    clc
    sub r30, r18
    sbc r31, r17
    cpi r31, 0
    breq check
    inc quotient
    rjmp loop
check:
    cp r30, r18
    brcc exit1
    rjmp end
exit1:
    inc quotient
    rjmp loop
end:
    inc quotient
exit_now:
```

```
   ret


lcd_init:
ldi r24 ,low(200)
ldi r25 ,high(200)          ; Wait 200 mSec
rcall wait_msec

ldi r24 ,0x30               ; command to switch to 8 bit mode
out PORTD ,r24
sbi PORTD ,PD3              ; Enable Pulse
nop
nop
cbi PORTD ,PD3
ldi r24 ,250                ;
ldi r25 ,0                  ; Wait 250uSec
rcall wait_usec             ;

ldi r24 ,0x30               ; command to switch to 8 bit mode
out PORTD ,r24              ;
sbi PORTD ,PD3              ; Enable Pulse
nop
nop
cbi PORTD ,PD3
ldi r24 ,250                ;
ldi r25 ,0                  ; Wait 250uSec
rcall wait_usec             ;

ldi r24 ,0x30               ; command to switch to 8 bit mode
out PORTD ,r24              ;
sbi PORTD ,PD3              ; Enable Pulse
nop
nop
cbi PORTD ,PD3
ldi r24 ,250                ;
ldi r25 ,0                  ; Wait 250uSec
rcall wait_usec

ldi r24 ,0x20               ; command to switch to 4 bit mode
out PORTD ,r24
sbi PORTD ,PD3              ; Enable Pulse
nop
nop
cbi PORTD ,PD3
ldi r24 ,250                ;
ldi r25 ,0                  ; Wait 250uSec
rcall wait_usec

ldi r24 ,0x28               ; 5x8 dots, 2 lines
rcall lcd_command
ldi r24 ,0x0c               ; display on, cursor off
rcall lcd_command
```

```
rcall lcd_clear_display
ldi r24 ,0x06                ; Increase address, no display shift
rcall lcd_command            ;
ret

lcd_clear_display:

ldi r24 ,0x01                ; clear display command
rcall lcd_command

ldi r24 ,low(5)              ;
ldi r25 ,high(5)             ; Wait 5 mSec
rcall wait_msec              ;
ret

lcd_command:
cbi PORTD ,PD2               ; LCD_RS=0(PD2=0), Instruction
rcall write_2_nibbles        ; send Instruction
ldi r24 ,250                 ;
ldi r25 ,0                   ; Wait 250uSec
rcall wait_usec
ret

lcd_data:
sbi PORTD ,PD2               ; LCD_RS=1(PD2=1), Data
rcall write_2_nibbles        ; send data
ldi r24 ,250                 ;
ldi r25 ,0                   ; Wait 250uSec
rcall wait_usec
ret

write_2_nibbles:
push r24                     ; save r24(LCD_Data)

in r25 ,PIND                 ; read PIND

andi r25 ,0x0f               ;
andi r24 ,0xf0               ; r24[3:0] Holds previus PORTD[3:0]
add r24 ,r25                 ; r24[7:4] <-- LCD_Data_High_Byte
out PORTD ,r24               ;

sbi PORTD ,PD3               ; Enable Pulse
nop
nop
cbi PORTD ,PD3               ; Disable Pulse

pop r24                      ; Recover r24(LCD_Data)
swap r24                     ;
andi r24 ,0xf0               ; r24[3:0] Holds previus PORTD[3:0]
add r24 ,r25                 ; r24[7:4] <-- LCD_Data_Low_Byte
out PORTD ,r24

sbi PORTD ,PD3               ; Enable Pulse
```

```
 nop
 nop
 cbi PORTD ,PD3

 ret

 wait_msec:
 push r24                    ; 2 cycles
 push r25                    ; 2 cycles
 ldi r24 , low(999)          ; 1 cycle
 ldi r25 , high(999)         ; 1 cycle
 rcall wait_usec             ; 998.375 usec
 pop r25                     ; 2 cycles
 pop r24                     ; 2 cycles
 nop                         ; 1 cycle
 nop                         ; 1 cycle
 sbiw r24 , 1                ; 2 cycles
 brne wait_msec              ; 1 or 2 cycles
 ret                         ; 4 cycles


wait_usec:
 sbiw r24 ,1                 ; 2 cycles (2/16 usec)
 call delay_8cycles          ; 4+8=12 cycles
 brne wait_usec              ; 1 or 2 cycles
 ret
delay_8cycles:
 nop
 nop
 nop
 ret
```

Ζήτημα 4.2

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <util/delay.h>

// swap macro
uint8_t swap(uint8_t x) {
    return (uint8_t)((x >> 4) | (x << 4));
}

void write_2_nibbles(uint8_t command) {
    // write MSB nibble
    PORTD = ((PIND & 0X0f) | (command & 0xf0));

    PORTD |= (1 << PD3);  // Enable pulse
    _delay_us(1);
    PORTD &= ~(1 << PD3); // Disable pulse

    // write LSB nibble
    PORTD = ((PIND & 0X0f) | (swap(command) & 0xf0));

    PORTD |= (1 << PD3);
    _delay_us(1);
    PORTD &= ~(1 << PD3);
}

// send a command to lcd's controller (4-bit mode)
void lcd_command(uint8_t command) {
    PORTD &= ~(1 << PD2);  // LCD_RS = 0 => instruction
    write_2_nibbles(command);  // send instruction
    _delay_us(250);  // wait 250us
}

// send data to lcd's controller (4-bit mode)
void lcd_data(uint8_t data) {
    PORTD |= (1 << PD2);  // LCD_RS = 1 => data
    write_2_nibbles(data);  // send instruction
    _delay_us(250);  // wait 250us
}

void lcd_clear_display() {
    lcd_command(0x01);
    _delay_ms(5);
}

// lcd monitor set-up
void lcd_init() {
```

```
    _delay_ms(200);  // wait 200ms

    // set microcontroler in 4-bit mode

    for (int i=0; i<3; i++) {  // 3 times
        PORTD = 0x30;  // switch to 8-bit mode
        PORTD |= (1 << PD3);  // Enable pulse
        _delay_us(1);
        PORTD &= ~(1 << PD3); // Disable pulse
        _delay_us(250);
    }

    PORTD = 0x20;  // switch to 4-bit mode
    PORTD |= (1 << PD3);  // Enable pulse
    _delay_us(1);
    PORTD &= ~(1 << PD3); // Disable the enable pulse
    _delay_us(250);

    lcd_command(0x28);   // 5x8 dots, 2 lines
    lcd_command(0x0c);   // display on, cursor-blinking off
    lcd_clear_display(); // clear display
    lcd_command(0x06);   // increase address, no display shift
}

int main() {
    char Vdisp[4];
    uint16_t adc_value;
    float Vin;

    // set voltage reference to AVCC and select ADC2 channel (0010)
    ADMUX = (1 << REFS0) | (1 << MUX1);
    // enable ADC and set prescaler to 128 for maximum resolution
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

    // initialize PORTD as output
    DDRD = 0xff;

    lcd_init();
    _delay_ms(100);

    while (1) {
        lcd_clear_display();

        // start an ADC conversion and wait for it to finish
        // if ADSC is set, a conversion is in progress
        ADCSRA |= (1 << ADSC);
        while (ADCSRA & (1 << ADSC));
        // collect the conversion's 10-bit result
        adc_value = ADC;

        // scale the digital ADC to a voltage level with 2 decimal places
        Vin = (adc_value*5.0)/1023.0;
```

```
        // extracting individual digits
        int digit1 = (int)Vin;
        int digit2 = (int)((Vin - digit1) * 10);
        int digit3 = (int)((Vin * 100) - (digit1 * 100 + digit2 * 10));

        // convert digits to characters and store in the array
        Vdisp[0] = '0' + digit1;
        Vdisp[1] = '.';
        Vdisp[2] = '0' + digit2;
        Vdisp[3] = '0' + digit3;

        // display the result character-by-character
        for (int i = 0; i < 4; i++) {
            lcd_data((uint8_t)(Vdisp[i]));
        }
        // wait for the next second
        _delay_ms(1000);
    }
    return 0;
}
```

Ζήτημα 4.3

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <util/delay.h>

// messages to be displayed
char gas_msg[] = {'G', 'A', 'S', ' ', 'D', 'E', 'T', 'E', 'C', 'T', 'E', 'D'};
char clr_msg[] = {'C', 'L', 'E', 'A', 'R'};

// swap macro
uint8_t swap(uint8_t x) {
    return (uint8_t)((x >> 4) | (x << 4));
}

void write_2_nibbles(uint8_t command) {
    // write MSB nibble
    PORTD = ((PIND & 0X0f) | (command & 0xf0));

    PORTD |= (1 << PD3);  // Enable pulse
    _delay_us(1);
    PORTD &= ~(1 << PD3); // Disable pulse

    // write LSB nibble
    PORTD = ((PIND & 0X0f) | (swap(command) & 0xf0));

    PORTD |= (1 << PD3);
    _delay_us(1);
    PORTD &= ~(1 << PD3);
}

// send a command to lcd's controller (4-bit mode)
void lcd_command(uint8_t command) {
    PORTD &= ~(1 << PD2);  // LCD_RS = 0 => instruction
    write_2_nibbles(command);  // send instruction
    _delay_us(250);  // wait 250us
}

// send data to lcd's controller (4-bit mode)
void lcd_data(uint8_t data) {
    PORTD |= (1 << PD2);  // LCD_RS = 1 => data
    write_2_nibbles(data);  // send instruction
    _delay_us(250);  // wait 250us
}

void lcd_clear_display() {
    lcd_command(0x01);
```

```
    _delay_ms(5);
}

// lcd monitor set-up
void lcd_init() {
    _delay_ms(200);  // wait 200ms

    // set microcontroler in 4-bit mode

    for (int i=0; i<3; i++) {  // 3 times
        PORTD = 0x30;  // switch to 8-bit mode
        PORTD |= (1 << PD3);  // Enable pulse
        _delay_us(1);
        PORTD &= ~(1 << PD3); // Disable pulse
        _delay_us(250);
    }

    PORTD = 0x20;  // switch to 4-bit mode
    PORTD |= (1 << PD3);  // Enable pulse
    _delay_us(1);
    PORTD &= ~(1 << PD3); // Disable the enable pulse
    _delay_us(250);

    lcd_command(0x28);   // 5x8 dots, 2 lines
    lcd_command(0x0c);   // display on, cursor-blinking off
    lcd_clear_display(); // clear display
    lcd_command(0x06);   // increase address, no display shift
}

// display the messages
void gas_detected() {
    for (int i=0; i<12; i++) {
        lcd_data((uint8_t)gas_msg[i]);
    }
}

void gas_clear() {
    for (int i=0; i<5; i++) {
        lcd_data((uint8_t)clr_msg[i]);
    }
}

int main() {
    uint16_t adc_value;
    float CO;
    unsigned int normal_level = 0;
    unsigned int first_det=0, first_clr=0;

    // set voltage reference to AVCC and select ADC2 channel (0010)
    ADMUX = (1 << REFS0) | (1 << MUX1);
    // enable ADC and set prescaler to 128 for maximum resolution
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
```

```
// initialize PORTD and PORTB as output
DDRD = 0xff;
DDRB = 0xff;

// initialize PORTB
PORTB = 0;

// set up lcd and wait 100ms
lcd_init();
_delay_ms(100);

while (1) {
    // start an ADC conversion and wait for it to finish
    // if ADSC is set, a conversion is in progress
    ADCSRA |= (1 << ADSC);
    while (ADCSRA & (1 << ADSC));
    // collect the conversion's 10-bit result
    adc_value = ADC;

    // scale the digital ADC to a voltage level with 2 decimal places
    CO = (adc_value*90.0)/1023.0;
    // 70ppm is a intermediate to high value considering CO level
    // so we set 90ppm as max value

    // turn on the corresponding LEDs
    if (CO <= 15) { PORTB = 0b1; }
    else if (CO <= 30) { PORTB = 0b11; }
    else if (CO <= 45) { PORTB = 0b111; }
    else if (CO <= 60) { PORTB = 0b1111; }
    else if (CO <= 75) { PORTB = 0b11111; }
    else { PORTB = 0b111111; }

    // wait 50ms
    _delay_ms(50);


    // if levels are high
    if (CO >= 70) {
        // turn off and on the LEDs every 50ms - 'blinking'
        PORTB = 0;
        // display "GAS DETECTED" message on lcd
        if (first_det == 0) {
            lcd_clear_display();
            gas_detected(); normal_level = 10;
            // don't display the message again
            first_det = 1;
        }
    }
    // when CO levels come back to normal
    // for the next 10 iterations of while-loop (10*(~100ms) = 1s)
    // display "CLEAR" message on lcd
    else if (normal_level > 0) {
        first_det = 0;
```

```
            if (first_clr == 0) {
                lcd_clear_display();
                gas_clear();
                // don't display the message again
                first_clr = 1;
            }
            normal_level--;
        }
        // else clear lcd
        else {
            lcd_clear_display();
            first_clr = 0;
        }

        // wait another 50ms (total of 100ms)
        _delay_ms(50);

    }
    return 0;
}
```