Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

4η Σειρά Ασκήσεων

Μάθημα: Εργαστήριο Μικροϋπολογιστών

Εξάμηνο: 7ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Ζήτημα 4.1

Να γραφεί πρόγραμμα σε assembly για τον ATmega328PB το οποίο θα ξεκινάει μια μετατροπή του ADC κάθε 1Sec. Η ανάγνωση των δεδομένων του ADC πρέπει να γίνεται μέσα στην ρουτίνα εξυπηρέτησης της διακοπής ολοκλήρωσης μετατροπής του ADC. Η διακοπή αυτή (ADC) μεταφέρει τον έλεγχο στην διεύθυνση 0x02A, αν είναι ενεργοποιημένη η αντίστοιχη διακοπή (από το bit ADIE του ADCSRA) καθώς και το flag επίτρεψης όλων των διακοπών. Η τιμή μέτρησης του ADC να μετατρέπεται σε τάση και να εκτυπώνονται στην LCD οθόνη, αρχίζοντας κάθε φορά από τον πρώτο χαρακτήρα της πρώτης γραμμής, με ακρίβεια δύο δεκαδικών ψηφίων (δεν χρειάζεται στρογγυλοποίηση).
Η τάση δίνεται από τον τύπο:

$$Vin = \frac{ADC}{1024} Vref$$

Όπου:

- VIN η τάση στην αναλογική είσοδο A2 του μικροελεγκτή.
- ADC η τιμή που διαβάζεται από τον ADC (10bit, από 0-1023)
- VREF η τάση αναφοράς που έχει οριστεί στα 5V.

Ο κώδικας για την υλοποίηση του ζητήματος:

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

.equ PD3 = 3
.equ PD2 = 2

.def temp = r16
.def counter = r17
.def ADC_L = r21
.def ADC_H = r22

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


reset:

;Initialize Stack Pointer
    ldi r24, low(RAMEND)
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24

    ser r24
    out DDRD, r24               ;Set PORTD as output
    out DDRB, r24               ;Set PORTD as output

    clr r24
    out DDRC, r24               ;Set PORTC as input

    sei                         ;Enable global interrupts

    ;REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0010 => select
ADC2(pin PC2)
    ;ADLAR=0 => RIGHT adjust the ADC result

    ldi r16, 0b01000010
    sts ADMUX, r16
```

```
    ;ADEN=1 => ADC Enable

    ldi r16, 0b11101111
    sts ADCSRA, r16

    ldi r16, 0b00000000
    sts ADCSRB, r16


main:
    jmp main


adc_interrupt:

    ; Initialize LCD and wait for 200 ms
    rcall lcd_init
    ldi r24, low(200)
    ldi r25, high(200)
    rcall wait_msec

    lds ADC_L, ADCL            ;Load the low byte of ADC result into
ADC_L
    lds ADC_H, ADCH            ;Load the high byte of ADC result into
ADC_H


    ; Copy the values of ADC_L and ADC_H to r17 and r18, respectively
    mov r17, ADC_L
    mov r18, ADC_H

    lsl r17
    rol r18
    lsl r17
    rol r18
    add r17, ADC_L
    adc r18, ADC_H
```

```
;Isolation of the MSB
mov ADC_H, r18
andi r18, 0b00011100
sub    ADC_H, r18
lsr r18
lsr r18

; Add 0x30 for the display output
ori r18, 0b00110000
mov r26, r18          ;Store the MSB in r26

;deftero psifio
mov ADC_L, r17
mov r18, ADC_H

lsl r17
rol r18
lsl r17
rol r18
lsl r17
rol r18
add r17, ADC_L
adc r18, ADC_H
add r17, ADC_L
adc r18, ADC_H

;Isolation of the first decimal
mov ADC_H, r18
andi r18,0b00111100
sub ADC_H, r18
lsr r18
lsr r18

; Add 0x30 for the display output
ori r18,0b00110000
mov r27, r18            ;Store the 1st decimal in r27


;trito  psifio
```

```
mov ADC_L, r17
mov r18, ADC_H
lsl r17
rol r18
lsl r17
rol r18
lsl r17
rol r18
add r17, ADC_L
adc r18, ADC_H
add r17, ADC_L
adc r18, ADC_H

;Isolation of the second decimal
mov ADC_H, r18
andi r18, 0b00111100
sub ADC_H, r18
lsr r18
lsr r18

; Add 0x30 for the display output
ori r18, 0b00110000

mov r24,r26
rcall lcd_data

ldi r24,'.'
rcall lcd_data

mov r24,r27
rcall lcd_data

mov r24,r18
rcall lcd_data
ldi r24, low(10)
ldi r25, high(10)
rcall wait_msec

reti
```

```
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

    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

 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
```

```
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_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_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               ; dislay on, cursor off
    rcall lcd_command

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

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

wait_msec:
    ; Delay function using nested loops
    ldi r23, 249
    loop_inn:
        dec r23
        nop
        brne loop_inn
        sbiw r24, 1
        brne wait_msec
    ret
```

Ο κώδικας εξηγείται περιληπτικά στα σχόλια.
Η ανάγνωση των δεδομένων του ADC γίνεται μέσα στην ρουτίνα εξυπηρέτησης της διακοπής ολοκλήρωσης μετατροπής του ADC (`adc_interrupt`).
Στη ρουτίνα adc_interrupt, η οποία καλείται όταν ολοκληρωθεί η μετατροπή ADC, διαβάζει το αποτέλεσμα ADC, το επεξεργάζεται σε μια τριψήφια δεκαδική αναπαράσταση και στέλνει αυτά τα δεδομένα στην οθόνη LCD. Το αποτέλεσμα ADC χωρίζεται στα μεμονωμένα ψηφία του και δημιουργούνται κωδικοί ASCII για να εμφανιστεί η δεκαδική τιμή στην οθόνη LCD.
Χρησιμοποιούμε τις ρουτίνες που μας έχουν δοθεί για την αρχικοποίηση, την εκκαθάριση της οθόνης και την εκτύπωση των χαρακτήρων σε αυτή.

Ζήτημα 4.2

Να γραφεί σε γλώσσα C, πρόγραμμα για τον ATmega328PB το οποίο θα ξεκινάει μια ADC μετατροπή, όπως στο Ζήτημα 4.1. Δεν θα κάνει χρήση της διακοπής ολοκλήρωσης μετατροπής του ADC και θα περιμένει μέχρι να ολοκληρωθεί η ADC μετατροπή ελέγχοντας το bit ADSC του ADCSRA το οποίο γίνεται 0 μόλις ολοκληρωθεί η μετατροπή (Polling). Θα εκτυπώνει την τάση στην LCD οθόνη με ακρίβεια δύο δεκαδικών ψηφίων.

Ο κώδικας για την υλοποίηση του ζητήματος:

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

void write_2_nibbles(uint8_t input){

    uint8_t temp = input;
    uint8_t pin = PIND;
    pin &= 0x0F;
    input &=0xF0;
    input |= pin;
    PORTD = input;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    input = temp;
    input &= 0x0F;
    input = input << 4;
    input |= pin;
    PORTD = input;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    return;
}

void LCD_data(uint8_t x){

    PORTD |= 0x04;
    write_2_nibbles(x);
    _delay_us(250);
    return;

}

void LCD_command(uint8_t x){

    PORTD &= 0xFB;
```

```
    write_2_nibbles(x);
    _delay_us(250);
    return;

}

void LCD_clear_display(){

    uint8_t x = 0x01;
    LCD_command(x);
    _delay_ms(5);

}


void LCD_init(void){

    _delay_ms(200);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(250);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(250);

    PORTD = 0x20;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(250);

    LCD_command(0x28);
    LCD_command(0x0C);

    LCD_clear_display();
```

```
    LCD_command(0x06);

}

int main(){

    DDRB = 0xFF;                    //Set PORTB as output
    DDRD = 0xFF;                    //Set PORTD as output
    DDRC = 0x00;                    //Set PORTC as input

    // Initialize the ADC for analog input
    ADMUX = (1 << REFS0) | (1 << MUX1);
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 <<
ADPS0);
    ADCSRB = 0x00;
    DIDR0 = ~(1 << ADC2D);

    while(1){

        // Initialize the LCD
        LCD_init();

        // Start ADC conversion
        ADCSRA |= (1 << ADSC);

        // Wait for ADC conversion to complete
        while ((ADCSRA & (1 << ADSC)) == (1 << ADSC));

        // Calculate voltage from ADC value
        float adc = ADC;
        float voltage = adc * 5 / 1024;

         // Extract individual digits of the voltage
        int voltage_msb = (uint8_t)(voltage);
        int voltage_first_dec = (voltage - voltage_msb)*10;
        voltage_first_dec = (uint8_t)(voltage_first_dec);
        int voltage_second_dec = (((voltage - voltage_msb)*10) -
voltage_first_dec)*10;
        voltage_second_dec = (uint8_t)(voltage_second_dec);
```

```
            // Convert digits to ASCII characters
            voltage_msb |= 0x30;
            voltage_first_dec |= 0x30;
            voltage_second_dec |= 0x30;

            // Display voltage on the LCD
            LCD_data(voltage_msb);
            LCD_data('.');
            LCD_data(voltage_first_dec);
            LCD_data(voltage_second_dec);

        }
    }
```

Ο κώδικας εξηγείται περιληπτικά στα σχόλια. Για να εκτυπώσουμε το αποτέλεσμα στην οθόνη LCD πρέπει να απομονώσουμε το κάθε ψηφίο.

Πρώτον, στο voltage_msb εκχωρείται το ακέραιο μέρος της τιμής της τάσης μετατρέποντας την τιμή τάσης σε έναν μη-προσημασμένο 8-bit ακέραιο (με χρήση του `uint8_t`), περικόπτοντας ουσιαστικά το δεκαδικό μέρος.

Στη συνέχεια, το voltage_first_dec υπολογίζεται αφαιρώντας το ακέραιο τμήμα (το οποίο ήταν αποθηκευμένο σε voltage_msb) από την αρχική τάση και πολλαπλασιάζοντας το δεκαδικό μέρος με το 10. Αυτή η λειτουργία έχει ως αποτέλεσμα το πρώτο δεκαδικό ψηφίο.

Στη συνέχεια, επαναλαμβάνετε μια παρόμοια διαδικασία για να λάβουμε το δεύτερο δεκαδικό ψηφίο, υπολογίζοντας τη διαφορά μεταξύ της αρχικής τάσης και του πρώτου δεκαδικού και πολλαπλασιάζοντας το αποτέλεσμα επί 10.

Τέλος, τα τρία ψηφία μετατρέπονται στις αναπαραστάσεις τους ASCII προσθέτοντας 0x30, που αντιστοιχεί στην τιμή ASCII του '0'. Αυτή η μετατροπή είναι απαραίτητη για την εμφάνιση των ψηφίων στην οθόνη LCD, καθώς η οθόνη LCD συνήθως αναμένει χαρακτήρες ASCII για εμφάνιση.

Ζήτημα 4.3

Να δημιουργηθεί κώδικας σε C για την επιτήρηση ενός χώρου όπου υπάρχει αυξημένος κίνδυνος ύπαρξης μονοξειδίου του άνθρακα (CO). Ο αισθητήρας CO είναι συνδεδεμένος στην αναλογική είσοδο A3 του μικροελεγκτή.

Καθ’ όλη την διάρκεια πρέπει να διαβάζεται η τιμή του αισθητήρα ανά 100 ms (μικρές αποκλίσεις είναι αποδεκτές) και να εμφανίζεται μια ένδειξη του επιπέδου του αερίου στα LED PB0-PB5. Αν οποιαδήποτε στιγμή η συγκέντρωση του CO ξεπεράσει τα 70ppm να τυπώνεται στην LCD το μήνυμα GAS DETECTED και να αναβοσβήνουν τα αντίστοιχα LED στα PB0-PB5 αναλόγως του επίπεδου του αερίου. Το επίπεδο θα πρέπει να συνεχίζει να διαβάζεται (και να εμφανίζεται στα LED) και αν επανέλθει σε επίπεδο χαμηλότερο των 70ppm τα LEDs να σταματούν να αναβοσβήνουν και να τυπώνεται στην LCD το μήνυμα CLEAR. Είναι καλή πρακτική να διατηρείτε τις ρουτίνες εξυπηρέτησης διακοπών όσο το δυνατόν μικρότερες σε χρονική διάρκεια. Παρόλα αυτά αν το πρόγραμμα σας είναι λειτουργικό δεν θα υπάρξει αρνητική επίπτωση στην βαθμολογία.

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

unsigned char level, flag;

void write_2_nibbles(uint8_t input){

    uint8_t temp = input;
    uint8_t pin = PIND;
    pin &= 0x0F;
    input &=0xF0;
    input |= pin;
    PORTD = input;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    input = temp;
    input &= 0x0F;
    input = input << 4;
    input |= pin;
    PORTD = input;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    return;
}
```

```
void LCD_data(uint8_t x){

    PORTD |= 0x04;
    write_2_nibbles(x);
    _delay_us(250);
    return;

}

void LCD_command(uint8_t x){

    PORTD &= 0xFB;
    write_2_nibbles(x);
    _delay_us(250);
    return;

}

void LCD_clear_display(){

    uint8_t x = 0x01;
    LCD_command(x);
    _delay_ms(5);

}


void LCD_init(void){

    _delay_ms(200);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(250);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
```

```
    _delay_us(250);

    PORTD = 0x20;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(250);

    LCD_command(0x28);
    LCD_command(0x0C);

    LCD_clear_display();

    LCD_command(0x06);

}

ISR(TIMER1_OVF_vect) {

     ADCSRA =0xEF;
}

ISR(ADC_vect) { // check the CO level

     if(21<=ADC && ADC<=45.2) { //Cx = [0,14] ppm
          level = 0x01;
          PORTB = level;

          TCNT1H = 0xfc;
          TCNT1L = 0xf3;        //Timer1 overflows in 100 msec
     }

    if(45.2<ADC && ADC<=90.4) { //Cx = [15,38] ppm
        level = 0x03;
        PORTB = level;

        TCNT1H = 0xfc;
          TCNT1L = 0xf3;        //Timer1 overflows in 100 msec
    }
```

```
    if(90.4<ADC && ADC<=135.6) { //Cx = [29,42]
        level = 0x07;
        PORTB = level;

        TCNT1H = 0xfc;
           TCNT1L = 0xf3;        //Timer1 overflows in 100 msec
    }

    if(135.6<ADC && ADC<=180) { //Cx = [29,42]
        level = 0x0F;
        PORTB = level;

        TCNT1H = 0xfc;
           TCNT1L = 0xf3;        //Timer1 overflows in 100 msec
    }

     if(180<ADC && ADC<=205) {   // Cx = [36,70] ppm
           level = 0x1F;
           PORTB = level;

           TCNT1H = 0xfc;
           TCNT1L = 0xf3;        //Timer1 overflows in 100 msec
     }

      if(205<ADC && ADC<=298 ) { // Cx = [71,105] ppm
            level = 0xc;

            TCNT1H = 0xfc;
            TCNT1L = 0xf3;

     }

     if(298<ADC && ADC<=614) {   // Cx = [105, ] ppm
           level = 0x30;

           TCNT1H = 0xfc;
           TCNT1L = 0xf3;        //Timer1 overflows in 100 msec
     }
}
```

```
int main(){

    DDRB = 0xFF;
    DDRD = 0xFF;
    DDRC = 0x00;

    ADMUX = (1 << REFS0) | (1 << MUX1); //| (1 << MUX0);
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 <<
ADPS0);
    ADCSRB = 0x00;
    DIDR0 = ~(1 << ADC2D);

    TIMSK1 = (1 << TOIE1);       //Timer1 ,interrupt enable
    TCCR1B =(1<<CS12) | (0<<CS11) | (1<<CS10); //frequency of Timer1
8MHz/1024

     sei();

    while(1){

        LCD_init();
        ADCSRA |= (1 << ADSC);

        if(flag==0x01) {
            LCD_data('C');
            LCD_data('L');
            LCD_data('E');
            LCD_data('A');
            LCD_data('R');
            _delay_ms(5000);
            LCD_clear_display();
            flag=0x00;
        }

        if((level==0x01) || (level==0x03) || (level==0x07) ||
(level==0x0F) || (level==0x1F)){
                PORTB = level;
          }
```

```
            if ((level==0xc) || (level==0x30)){
             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');

             flag = 0x01;

             while ((level==0xc) || (level==0x30)){ // Cx = [71,105]
ppm ALARM
                        PORTB = 0xff;
                        _delay_ms(500);
                        PORTB = 0x00;
                        _delay_ms(500);

                    }
            }
    }
}
```

Στον παραπάνω κωδικά χρησιμοποιούνται οι ίδιες βασικές συναρτήσεις που εμφανίζονται και στο *Ζήτημα 4.2*, με την προσθήκη της συνάρτησης διακοπής `ISR(ADC_vect)`. Η διακοπή αυτή ενεργοποιείται όταν ολοκληρωθεί μια μετατροπή ADC. Ελέγχει το επίπεδο CO, ενημερώνοντας κατάλληλα την τιμή της μεταβλητής `level`, με βάση τη μετρούμενη τιμή και ορίζει τον Timer1 για έλεγχο υπερχείλισης.
Στη main συνάρτηση ο κωδικός αρχικοποιεί τη διαμόρφωση του μικροελεγκτή, η οποία περιλαμβάνει τη ρύθμιση του μετατροπέα αναλογικού σε ψηφιακό (ADC) και του Timer1 για την παρακολούθηση των επιπέδων μονοξειδίου του άνθρακα (CO).
Μέσα στον κύριο βρόχο, το πρόγραμμα προετοιμάζει την οθόνη LCD και ξεκινά μια μετατροπή ADC για τη μέτρηση του επιπέδου CO. Ανάλογα με το μετρούμενο επίπεδο CO και το συγκεκριμένο εύρος

του, ο κώδικας ενημερώνει τη μεταβλητή `level`. Εάν το επίπεδο CO πέφτει εντός ενός συγκεκριμένου εύρους (0xc ή 0x30), εμφανίζει ένα προειδοποιητικό μήνυμα στην οθόνη LCD και ρυθμίζει τη «σημαία» ώστε να ενεργοποιεί πρόσθετες ενέργειες.
Σε περίπτωση συνθήκης συναγερμού, ειδικά όταν το επίπεδο CO είναι στο εύρος 0xc ή 0x30, το πρόγραμμα ανάβει και σβήνει συνεχώς μια λυχνία LED. Αυτό το LED αντιπροσωπεύεται από τη μεταβλητή «PORTB». Αυτό το LED που αναβοσβήνει χρησιμεύει ως οπτική ειδοποίηση, σηματοδοτώντας μια δυνητικά επικίνδυνη συγκέντρωση CO.