Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μ Α Θ Η Μ Α
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ (ΡΟΗ Υ)

Ε Ρ Γ Α Σ Ι Α
4Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Σ Π Ο Υ Δ Α Σ Τ Ε Σ
ΚΑΡΑΜ ΚΩΝΣΤΑΝΤΙΝΟΣ
ΚΟΛΙΟΣ ΑΠΟΣΤΟΛΟΣ

2022–23

Οι πλήρεις κώδικες όλων των ασκήσεων περιέχονται στο zipped αρχείο.

Ζήτημα 4.1 *(Lab4_1.asm και Lab4_1.c)*

Δίνεται ο κώδικας σε assembly για την μετατροπή του ADC και την εκτύπωση του αποτελέσματος στην οθόνη με την χρήση polling:

```
.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_int

write_2_nibbles:
push r24
in r25 ,PIND
andi r25 ,0x0f
andi r24 ,0xf0
add r24 ,r25
out PORTD ,r24
sbi PORTD ,PD3
cbi PORTD ,PD3
nop
```

```
nop
pop r24
swap r24
andi r24 ,0xf0
add r24 ,r25
out PORTD ,r24
sbi PORTD ,PD3
cbi PORTD ,PD3
nop
nop
ret

lcd_data:
sbi PORTD ,PD2
rcall write_2_nibbles
ldi r24 ,100
ldi r25 ,0
rcall wait_usec
ret

lcd_command:
cbi PORTD ,PD2
rcall write_2_nibbles
ldi r24 ,100
ldi r25 ,0
rcall wait_usec
ret

lcd_init:
ldi r24 ,100
ldi r25 ,0
rcall wait_msec
ldi r24 ,0x30
out PORTD ,r24
sbi PORTD ,PD3
cbi PORTD ,PD3
ldi r24 ,100
ldi r25 ,0
rcall wait_usec
ldi r24 ,0x30
out PORTD ,r24
sbi PORTD ,PD3
cbi PORTD ,PD3
ldi r24 ,100
ldi r25 ,0
rcall wait_usec
ldi r24 ,0x20
out PORTD ,r24
sbi PORTD ,PD3
cbi PORTD ,PD3
```

```
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

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

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

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

sei

; REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0000 => select ADC0(pin PC0),
; ADLAR=0 => RIGHT adjust the ADC result
ldi temp, 0b01000010
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, 0b11101111
sts ADCSRA, temp

ldi temp, 0b00000000
sts ADCSRB, temp

ldi temp,0x00

main:
out PORTB,temp
inc temp
ldi r24, low(1000)
```

```
ldi r25, high(1000)
rcall wait_msec
jmp main

wait_msec:                  ; 1 msec delay per call
push r24                    ; 2 cycles
push r25                    ; 2 cycles
ldi r24,low(125)            ; 1 cycle
ldi r25,high(125)           ; 1 cycle
rcall wait_usec             ; 3 cycles
pop r25                     ; 2 cycles
pop r24                     ; 2 cycles
sbiw r24,1                  ; 2 cycles
brne wait_msec              ; 1 or 2 cycles
ret                         ; 4 cycles

wait_usec:                  ; Called 125 times
sbiw r24,1                  ; 2 cycles (2 usec)
nop                         ; 1 cycle (1 usec)
nop                         ; 1 cycle (1 usec)
nop                         ; 1 cycle (1 usec)
nop                         ; 1 cycle (1 usec)
brne wait_usec              ; 1 or 2 cycles (1 or 2 usec)
ret                         ; 4 cycles (4 usec)

adc_int:
rcall lcd_init
ldi r24, low(2)
ldi r25, high(2)
rcall wait_msec
lds r17,ADCL
lds r18,ADCH
mov r22,r18
mov r21,r17
lsl r17
rol r18
lsl r17
rol r18
add r17,r21
adc r18,r22
mov r22,r18
andi r18,0b00011100
sub r22,r18
lsr r18
lsr r18
ori r18,0b00110000
mov r26,r18                 ; MSB -> r26

;gia to 2o psifio
mov r21,r17
```

```
mov r18,r22
lsl r17
rol r18
lsl r17
rol r18
lsl r17
rol r18
add r17,r21
adc r18,r22
add r17,r21
adc r18,r22
mov r22,r18
andi r18,0b00111100
sub r22,r18
lsr r18
lsr r18
ori r18,0b00110000
mov r27, r18                ; 1st decimal -> r27

;gia to 3o psifio
mov r21,r17
mov r18,r22
lsl r17
rol r18
lsl r17
rol r18
lsl r17
rol r18
add r17,r21
adc r18,r22
add r17,r21
adc r18,r22
mov r22,r18
andi r18,0b00111100
sub r22,r18
lsr r18
lsr r18
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(500)
ldi r25, high(500)
rcall wait_msec

reti
```

Για να εκτυπώσουμε το αποτέλεσμα στην LCD θα πρέπει να απομονώσουμε τα bit των καταχωρητών r21, r22 στους οποίους είναι αποθηκευμένη η ADC τάση. Μετατοπίζοντας την θέση των bits των r21, r22 και χρησιμοποιώντας masks για την απομόνωση των κατάλληλων bits(για το πρώτο, δεύτερο και τρίτο ψηφίο), καταφέρνουμε να αποθηκεύσουμε τα ψηφία σε καταχωρητές και έπειτα να τα εκτυπώσουμε σειριακά στην LCD.

Με ανάλογο τρόπο η ίδια διαδικασία σε C. Εδώ για για την διακοπή χρησιμοποιούμε το Polling method.

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

void Write_2_Nibbles(uint8_t in) {
    uint8_t temp = in;
    uint8_t p = PIND;
    p &= 0x0F;
    in &= 0xF0;
    in |= p;
    PORTD = in;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    in = temp;
    in &= 0x0F;
    in = in << 4;
    in |= p;
    PORTD = in;
    PORTD |= 0x08;
    PORTD &= 0xF7;

    return;
}

void LCD_data(uint8_t c) {
    PORTD |= 0x04;
    Write_2_Nibbles(c);
    _delay_us(100);
    return;
}
```

```
void LCD_command(uint8_t c) {
    PORTD &= 0xFB;
    Write_2_Nibbles(c);
    _delay_us(100);
    return;
}

void LCD_init(void) {
    _delay_ms(40);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    PORTD = 0x30;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    PORTD = 0x20;
    PORTD |= 0x08;
    PORTD &= 0xF7;
    _delay_us(100);

    LCD_command(0x28);
    LCD_command(0x0C);
    LCD_command(0x01);
    _delay_us(5000);

    LCD_command(0x06);
}


int main()
{
    DDRB = 0xFF;
    DDRD = 0xFF;
    DDRC = 0x00;

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

    while(1) {
        LCD_init();
        ADCSRA |= (1 << ADSC);
        while ((ADCSRA & (1 << ADSC)) == (1 << ADSC));
        float adc = ADC;
```

```
        adc = adc * 5 / 1024;
        int adc_akeraio = (uint8_t)(adc);
        int adc_1_dec = (adc - adc_akeraio) * 10;
        adc_1_dec = (uint8_t)(adc_1_dec);
        int adc_2_dec = (((adc - adc_akeraio) * 10) - adc_1_dec) * 10;
        adc_2_dec = (uint8_t)(adc_2_dec);

        adc_akeraio |= 0x30;
        adc_1_dec |= 0x30;
        adc_2_dec |= 0x30;
        LCD_data(adc_akeraio);
        LCD_data('.');
        LCD_data(adc_1_dec);
        LCD_data(adc_2_dec);
    }
}
```

Ζήτημα 4.3 *(Lab4_3.c)*

Δίνεται ο κώδικας σε γλώσσα C:

```
int main()
{
    TCCR1A = (0<<WGM10) | (1<<WGM11) | (1<<COM1A1);
    TCCR1B = (1<<WGM12) | (1<<CS11) | (1<<WGM13);

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

    OCR1AL = 0x80;

    DDRB = 0b00000010;
    DDRC = 0x00;
    DDRD = 0xFF;

    LCD_init();

    float adc;
    int adc_akeraio, adc_1_dec, adc_2_dec;

    while(1)
    {
        ICR1 = 0x00;

        if (PINB == 0b10111011) {
```

```
            while(PINB == 0b10111011) {
                ICR1 = 0x018F;
                OCR1A = ICR1 * 0.2;

                ADCSRA |= (1 << ADSC);
                while ((ADCSRA & (1 << ADSC)) == (1 << ADSC));
                adc = ADC;
                adc = adc * 5 / 1024 * 0.2;
                adc_akeraio = (uint8_t)(adc);
                adc_1_dec = (adc - adc_akeraio) * 10;
                adc_1_dec = (uint8_t)(adc_1_dec);
                adc_2_dec = (((adc - adc_akeraio) * 10) - adc_1_dec) * 10;
                adc_2_dec = (uint8_t)(adc_2_dec);

                adc_akeraio |= 0x30;
                adc_1_dec |= 0x30;
                adc_2_dec |= 0x30;

                LCD_init();
                LCD_data('2');
                LCD_data('0');
                LCD_data('%');
                LCD_command(0b11000000);    // Next line
                LCD_data(adc_akeraio);
                LCD_data('.');
                LCD_data(adc_1_dec);
                LCD_data(adc_2_dec);
            }
        }
        if (PINB == 0b10110111) {
            while(PINB == 0b10110111) {
                ICR1 = 0x018F;
                OCR1A = ICR1 * 0.4;

                ADCSRA |= (1 << ADSC);
                while ((ADCSRA & (1 << ADSC)) == (1 << ADSC));
                adc = ADC;
                adc = adc * 5 / 1024 * 0.4;
                adc_akeraio = (uint8_t)(adc);
                adc_1_dec = (adc - adc_akeraio) * 10;
                adc_1_dec = (uint8_t)(adc_1_dec);
                adc_2_dec = (((adc - adc_akeraio) * 10) - adc_1_dec) * 10;
                adc_2_dec = (uint8_t)(adc_2_dec);

                adc_akeraio |= 0x30;
                adc_1_dec |= 0x30;
                adc_2_dec |= 0x30;

                LCD_init();
                LCD_data('4');
```

```
            LCD_data('0');
            LCD_data('%');
            LCD_command(0b11000000);
            LCD_data(adc_akeraio);
            LCD_data('.');
            LCD_data(adc_1_dec);
            LCD_data(adc_2_dec);
        }
    }
    if (PINB == 0b10101111) {
        while(PINB == 0b10101111) {
            ICR1 = 0x018F;
            OCR1A = ICR1 * 0.6;

            ADCSRA |= (1 << ADSC);
            while ((ADCSRA & (1 << ADSC)) == (1 << ADSC));
            adc = ADC;
            adc = adc * 5 / 1024 * 0.6;
            adc_akeraio = (uint8_t)(adc);
            adc_1_dec = (adc - adc_akeraio) * 10;
            adc_1_dec = (uint8_t)(adc_1_dec);
            adc_2_dec = (((adc - adc_akeraio) * 10) - adc_1_dec) * 10;
            adc_2_dec = (uint8_t)(adc_2_dec);

            adc_akeraio |= 0x30;
            adc_1_dec |= 0x30;
            adc_2_dec |= 0x30;

            LCD_init();
            LCD_data('6');
            LCD_data('0');
            LCD_data('%');
            LCD_command(0b11000000);
            LCD_data(adc_akeraio);
            LCD_data('.');
            LCD_data(adc_1_dec);
            LCD_data(adc_2_dec);
        }
    }
    if (PINB == 0b10011111) {
        while(PINB == 0b10011111) {
            ICR1 = 0x018F;
            OCR1A = ICR1 * 0.8;

            ADCSRA |= (1 << ADSC);
            while ((ADCSRA & (1 << ADSC)) == (1 << ADSC));
            adc = ADC;
            adc = adc * 5 / 1024 * 0.8;
            adc_akeraio = (uint8_t)(adc);
            adc_1_dec = (adc - adc_akeraio) * 10;
```

```
                adc_1_dec = (uint8_t)(adc_1_dec);
                adc_2_dec = (((adc - adc_akeraio) * 10) - adc_1_dec) * 10;
                adc_2_dec = (uint8_t)(adc_2_dec);

                adc_akeraio |= 0x30;
                adc_1_dec |= 0x30;
                adc_2_dec |= 0x30;

                LCD_init();
                LCD_data('8');
                LCD_data('0');
                LCD_data('%');
                LCD_command(0b11000000);
                LCD_data(adc_akeraio);
                LCD_data('.');
                LCD_data(adc_1_dec);
                LCD_data(adc_2_dec);
            }
        }
    }
}
```

Ανάλογα με το κουμπί που είναι πατημένο αρχικοποιείται ο OCR1A. Έπειτα χρησιμοποιούμε τον κώδικα για την απομόνωση των ψηφίων για τη τιμή του ADC. Τέλος, εκτυπώνουμε την τιμή του ανάλογου level στην γραμμή κάτω από την τιμή του duty cycle.