Εργαστήριο Μικροϋπολογιστών – 5η Σειρά Ασκήσεων

Κυριακόπουλος Γιώργος – el18153
Τζελέπης Σεραφείμ – el18849

1η Άσκηση:

C file

```c
#define F_CPU 8000000UL
#include "avr/io.h"
#include <util/delay.h>
#include <avr/interrupt.h>

// extern is used to link the assembly functions that will be used
// here, declared as global, on the .s file
extern void lcd_data_sim(uint8_t);
extern void lcd_init_sim();

// global variables
uint8_t value;
unsigned char memory[2], keypad[2], duty = 0, digit = 0;
int counter = 0;

// scan a keyboard row defined by i
unsigned char scan_row(int i) {
    unsigned char r = (1 << (i + 3));   // set r to 1 shifted row + 3
    PORTC = r;                          // r bit of PORTC is output
    _delay_us(500);                     // delay 500 us for remote
    ;                                   // nop
    ;                                   // nop
    return PINC & 0x0F;                 // return the 4 isolated LSBs
}

// swap 4 LSBs with 4 MSBs
unsigned char swap(unsigned char x) {
    return ((x & 0x0F) << 4) | ((x & 0xF0) >> 4);
}

// scan all the keypad rows and store result in keypad
void scan_keypad() {
    unsigned char i;
```

```
    i = scan_row(1);                    // scan 1st row (PC4)
    keypad[1] = swap(i);                // store in 4 keypad[1] MSBs

    i = scan_row(2);                    // scan 2nd row (PC5)
    keypad[1] += i;                     // store in 4 keypad[1] LSBs

    i = scan_row(3);                    // scan 3rd row (PC6)
    keypad[0] = swap(i);                // store in 4 keypad[0] MSBs

    i = scan_row(4);                    // scan 4th row (PC7)
    keypad[0] += i;                     // store in 4 keypad[0] LSBs

    PORTC = 0x00;                       // remote
}

// scan keypad the right way
int scan_keypad_rising_edge() {
    scan_keypad();                      // scan and store keypad

    unsigned char temp[2];              // temporary register
    temp[0] = keypad[0];                // store the keypad data
    temp[1] = keypad[1];                // store the keypad data

    _delay_ms(15);                      // delay 15 ms for flashover

    scan_keypad();                      // scan and store keypad

    keypad[0] &= temp[0];               // keep pressed buttons
    keypad[1] &= temp[1];               // keep pressed buttons

    temp[0] = memory[0];                // get old buttons from RAM
    temp[1] = memory[1];                // get old buttons from RAM

    memory[0] = keypad[0];              // store new buttons in RAM
    memory[1] = keypad[1];              // store new buttons in RAM

    keypad[0] &= ~temp[0];              // keep new pressed buttons
    keypad[1] &= ~temp[1];              // keep new pressed buttons

    return (keypad[0] || keypad[1]);    // return new pressed buttons
}

// button pressed hex to ascii
unsigned char keypad_to_ascii() {
```

```
if (keypad[0] & 0x01) {              // check every bit and if it
    return '*';                      // is 1 return the
}                                    // corresponding ascii code
if (keypad[0] & 0x02) {
    return '0';
}
if (keypad[0] & 0x04) {
    return '#';
}
if (keypad[0] & 0x08) {
    return 'D';
}
if (keypad[0] & 0x10) {
    return '7';
}
if (keypad[0] & 0x20) {
    return '8';
}
if (keypad[0] & 0x40) {
    return '9';
}
if (keypad[0] & 0x80) {
    return 'C';
}
if (keypad[1] & 0x01) {
    return '4';
}
if (keypad[1] & 0x02) {
    return '5';
}
if (keypad[1] & 0x04) {
    return '6';
}
if (keypad[1] & 0x08) {
    return 'B';
}
if (keypad[1] & 0x10) {
    return '1';
}
if (keypad[1] & 0x20) {
    return '2';
}
if (keypad[1] & 0x40) {
    return '3';
}
```

```
    if (keypad[1] & 0x80) {
        return 'A';
    }
    return 0;                           // if none pressed return 0
}

// initialize ADC
void ADC_init(void) {
    // Vref: Vcc
    // MUX4:0 = 00000 for A0
    ADMUX = (1 << REFS0);
    // ADC is Enable (ADEN=1)
    // ADC Interrupts are Enabled (ADIE=1)
    // Set Prescaler CK/128 = 62.5Khz (ADPS2:0=111)
    ADCSRA = (1 << ADEN) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1)
| (1 << ADPS0);
}

// TMR0 overflow interruption service routine
ISR(TIMER0_OVF_vect) {
    // increment counter with every overflow
    counter++;

    // if counter reaches 1000, start the ADC conversion
    // and reset the counter to 0
    if(counter == 1000){
        ADCSRA |= (1 << ADSC);
        counter = 0;
    }
}

// ADC interruption service routine
ISR(ADC_vect) {
    // read the ADC value and get the integer digit (v_one)
    // plus the first 2 decimal digits (v_two, v_three)
    double V = ADC * 5.0 / 1024.0;
    int v_one = V / 1;
    int v_two = (int) (V * 10) % 10;
    int v_three = (int) (V * 100) % 10;

    // clear the screen by initializing it again and output
    // the voltage according to the specified format
    lcd_init_sim();
    value = 'V';
    lcd_data_sim(value);
```

```
    value = 'o';
    lcd_data_sim(value);
    value = '1';
    lcd_data_sim(value);
    value = '\n';
    lcd_data_sim(value);
    value = v_one + '0';
    lcd_data_sim(value);
    value = '.';
    lcd_data_sim(value);
    value = v_two + '0';
    lcd_data_sim(value);
    value = v_three + '0';
    lcd_data_sim(value);
}

// PWM init function of TMR0 and OC0 is connected to pin PB3
void PWM_init() {
    // set TMR0 in fast PWM 8 bit mode with non-inverted output
    // prescale = 8, since f_pwm = f_clk/(N(1+TOP)) => N = 8
    TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS01);

    // set initial duty cycle compare value to 0
    OCR0 = 0;

    // set PB3 pin as output
    DDRB |= (1 << PB3);
}

int main () {
    memory[0] = 0;                      // initialize array for RAM
    memory[1] = 0;                      // initialize array for RAM

    DDRC = 0xF0;                        // [7:4] output [3:0] input
    DDRD = 0xFF;                        // PORTD is output

    lcd_init_sim();                     // initialize LCD
    ADC_init();                         // initialize ADC

    TIMSK = (1 << TOIE0);               // enable overflow interrupt
    PWM_init();                         // initialize PWM
    sei();                              // enable interrupts

    while(1) {
        while(1) {
```

```
            // scan for button pressed, get its ascii and break
            if(scan_keypad_rising_edge()) {
                digit = keypad_to_ascii();
                break;
            }
        }

        // if digit is 1 then increase duty value and update OCR0
        if(digit == '1') {
            if (duty < 255) {
                duty++;
                OCR0 = duty;
            }
            _delay_ms(8);
        }
        // if digit is 2 then decrease duty value and update OCR0
        else if(digit == '2') {
            if (duty > 0) {
                duty--;
                OCR0 = duty;
            }
            _delay_ms(8);
        }
    }

    return 0;
}
```

Assembly file

```
; AVR GCC compiler is used so there are a few changes on the code,
like
; high -> hi8, low -> lo8 and .include "m16def.inc" -> #include
<avr/io.h>

; following lines are needed to use the IN/OUT ports
#define _SFR_ASM_COMPAT 1
#define __SFR_OFFSET 0
#include <avr/io.h>

; global declaration of functions that are used in the c program
.global lcd_data_sim
.global lcd_init_sim

write_2_nibbles_sim:
```

```
    push r24                    ; τμήμα κώδικα που προστίθεται για τη
σωστή
    push r25                    ; λειτουργία του προγράμματος
απομακρυσμένης
    ldi r24 ,lo8(6000)          ; πρόσβασης
    ldi r25 ,hi8(6000)
    rcall wait_usec
    pop r25
    pop r24                     ; τέλος τμήμα κώδικα
    push r24                    ; στέλνει τα 4 MSB
    in r25, PIND                ; διαβάζονται τα 4 LSB και τα
ξαναστέλνουμε
    andi r25, 0x0f              ; για να μην χαλάσουμε την όποια
προηγούμενη κατάσταση
    andi r24, 0xf0              ; απομονώνονται τα 4 MSB και
    add r24, r25                ; συνδυάζονται με τα προϋπάρχοντα 4
LSB
    out PORTD, r24              ; και δίνονται στην έξοδο
    sbi PORTD, PD3              ; δημιουργείται παλμός Enable στον
ακροδέκτη PD3
    cbi PORTD, PD3              ; PD3=1 και μετά PD3=0
    push r24                    ; τμήμα κώδικα που προστίθεται για τη
σωστή
    push r25                    ; λειτουργία του προγράμματος
απομακρυσμένης
    ldi r24 ,lo8(6000)          ; πρόσβασης
    ldi r25 ,hi8(6000)
    rcall wait_usec
    pop r25
    pop r24                     ; τέλος τμήμα κώδικα
    pop r24                     ; στέλνει τα 4 LSB. Ανακτάται το
byte.
    swap r24                    ; εναλλάσσονται τα 4 MSB με τα 4 LSB
    andi r24 ,0xf0              ; που με την σειρά τους αποστέλλονται
    add r24, r25
    out PORTD, r24
    sbi PORTD, PD3              ; Νέος παλμός Enable
    cbi PORTD, PD3
    ret

lcd_data_sim:
    push r24
    push r25
    sbi PORTD,PD2
    rcall write_2_nibbles_sim
```

```
    ldi r24,43
    ldi r25,0
    rcall wait_usec
    pop r25
    pop r24
    ret

lcd_command_sim:
    push r24                    ; αποθήκευσε τους καταχωρητές r25:r24
γιατί τους
    push r25                    ; αλλάζουμε μέσα στη ρουτίνα
    cbi PORTD, PD2              ; επιλογή του καταχωρητή εντολών
(PD2=0)
    rcall write_2_nibbles_sim   ; αποστολή της εντολής και αναμονή
39μsec
    ldi r24, 39                 ; για την ολοκλήρωση της εκτέλεσης
της από τον ελεγκτή της lcd.
    ldi r25, 0                  ; ΣΗΜ.: υπάρχουν δύο εντολές, οι
clear display και return home,
    rcall wait_usec             ; που απαιτούν σημαντικά μεγαλύτερο
χρονικό διάστημα.
    pop r25                     ; επανάφερε τους καταχωρητές r25:r24
    pop r24
    ret

lcd_init_sim:
    push r24                    ; αποθήκευσε τους καταχωρητές r25:r24
γιατί τους
    push r25                    ; αλλάζουμε μέσα στη ρουτίνα

    ldi r24, 40                 ; Όταν ο ελεγκτής της lcd
τροφοδοτείται με
    ldi r25, 0                  ; ρεύμα εκτελεί την δική του
αρχικοποίηση.
    rcall wait_msec             ; Αναμονή 40 msec μέχρι αυτή να
ολοκληρωθεί.
    ldi r24, 0x30               ; εντολή μετάβασης σε 8 bit mode
    out PORTD, r24              ; επειδή δεν μπορούμε να είμαστε
βέβαιοι
    sbi PORTD, PD3              ; για τη διαμόρφωση εισόδου του
ελεγκτή
    cbi PORTD, PD3              ; της οθόνης, η εντολή αποστέλλεται
δύο φορές
    ldi r24, 39
```

```
    ldi r25, 0                  ; εάν ο ελεγκτής της οθόνης βρίσκεται
σε 8-bit mode
    rcall wait_usec             ; δεν θα συμβεί τίποτα, αλλά αν ο
ελεγκτής έχει διαμόρφωση
                                ; εισόδου 4 bit θα μεταβεί σε
διαμόρφωση 8 bit
    push r24                    ; τμήμα κώδικα που προστίθεται για τη
σωστή
    push r25                    ; λειτουργία του προγράμματος
απομακρυσμένης
    ldi r24,lo8(1000)           ; πρόσβασης
    ldi r25,hi8(1000)
    rcall wait_usec
    pop r25
    pop r24                     ; τέλος τμήμα κώδικα
    ldi r24, 0x30
    out PORTD, r24
    sbi PORTD, PD3
    cbi PORTD, PD3
    ldi r24,39
    ldi r25,0
    rcall wait_usec
    push r24                    ; τμήμα κώδικα που προστίθεται για τη
σωστή
    push r25                    ; λειτουργία του προγράμματος
απομακρυσμένης
    ldi r24 ,lo8(1000)          ; πρόσβασης
    ldi r25 ,hi8(1000)
    rcall wait_usec
    pop r25
    pop r24                     ; τέλος τμήμα κώδικα
    ldi r24,0x20                ; αλλαγή σε 4-bit mode
    out PORTD, r24
    sbi PORTD, PD3
    cbi PORTD, PD3
    ldi r24,39
    ldi r25,0
    rcall wait_usec
    push r24                    ; τμήμα κώδικα που προστίθεται για τη
σωστή
    push r25                    ; λειτουργία του προγράμματος
απομακρυσμένης
    ldi r24 ,lo8(1000)          ; πρόσβασης
    ldi r25 ,hi8(1000)
    rcall wait_usec
```

```
    pop r25
    pop r24                     ; τέλος τμήμα κώδικα
    ldi r24,0x28                ; επιλογή χαρακτήρων μεγέθους 5x8
κουκίδων
    rcall lcd_command_sim       ; και εμφάνιση δύο γραμμών στην οθόνη
    ldi r24,0x0c                ; ενεργοποίηση της οθόνης, απόκρυψη
του κέρσορα
    rcall lcd_command_sim
    ldi r24,0x01                ; καθαρισμός της οθόνης
    rcall lcd_command_sim
    ldi r24, lo8(1530)
    ldi r25, hi8(1530)
    rcall wait_usec
    ldi r24 ,0x06               ; ενεργοποίηση αυτόματης αύξησης κατά
1 της διεύθυνσης
    rcall lcd_command_sim       ; που είναι αποθηκευμένη στον μετρητή
διευθύνσεων και
                                ; απενεργοποίηση της ολίσθησης
ολόκληρης της οθόνης
    pop r25                     ; επανάφερε τους καταχωρητές r25:r24
    pop r24
    ret

wait_msec:
    push r24                    ; 2 κύκλοι (0.250 μsec)
    push r25                    ; 2 κύκλοι
    ldi r24 , lo8(998)          ; φόρτωσε τον καταχ. r25:r24 με 998
(1 κύκλος - 0.125 μsec)
    ldi r25 , hi8(998)         ; 1 κύκλος (0.125 μsec)
    rcall wait_usec             ; 3 κύκλοι (0.375 μsec), προκαλεί
συνολικά καθυστέρηση 998.375 μsec
    pop r25                     ; 2 κύκλοι (0.250 μsec)
    pop r24                     ; 2 κύκλοι
    sbiw r24 , 1                ; 2 κύκλοι
    brne wait_msec              ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
    ret                         ; 4 κύκλοι (0.500 μsec)

wait_usec:
    sbiw r24 ,1                 ; 2 κύκλοι (0.250 μsec)
    nop                         ; 1 κύκλος (0.125 μsec)
    nop                         ; 1 κύκλος (0.125 μsec)
    nop                         ; 1 κύκλος (0.125 μsec)
    nop                         ; 1 κύκλος (0.125 μsec)
    brne wait_usec              ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
    ret                         ; 4 κύκλοι (0.500 μsec)
```