Εργαστήριο Μικρουπολογιστών

5η Άσκηση

Παντελαίος Δημήτριος el18049

Μοίρας Αλέξανδρος el18081

Χρησιμοποιήθηκαν 2 αρχεία -main.c και Assemblyfunctions.S- με το main.c να περιέχει το κυρίως πρόγραμμα και το Assemblyfunctions.S να περιέχει τους οδηγούς του keypad, της LCD, τις συναρτήσεις χρονοκαθυστερήσεων, μία συνάρτηση που εκτυπώνει τον χαρακτήρα που παίρνει ως όρισμα (αυτός ο χαρακτήρας θα τοποθετεί στον r24 βάσει των calling conventions), μία συνάρτηση που στέλνει την κατάλληλη εντολή ώστε να αλλάξουμε γραμμή στην LCD (που δεν θα χρησιμοποιηθεί καθώς δεν υποστηρίζεται από το πρόγραμμα προσομοίωσης) καθώς και μία συνάρτηση που αρχικοποιεί τη μεταβλητή _tmp_ που βρίσκεται στη RAM. Το compilation το αναλαμβάνει το microchip studio το οποίο κατά το build κάνει αυτόματα link τα παράγωγα του κώδικα που περιέχεται στα δύο αρχεία. Κατά τη σχεδίαση σεβαστήκαμε τα function call conventions μεταξύ c και assembly (πχ το 16bit return value της scan_keypad_rising_edge_sim() αποθηκεύεται στους r25:r24 ώστε να ανακτηθεί από το c πρόγραμμα κατά την επιστροφή της συνάρτησης). Ο κώδικας των δύο αρχείων εμπλουτισμένος με σχόλια:

main.c:

```c
#define F_CPU 8000000UL
#include <avr/io.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <avr/interrupt.h>

int counter;
float value_read, last_value;
int scan_keypad_rising_edge_sim();
void initialize_variable();
void lcd_init_sim();
void lcd_clear();
//void changeline();
//void changeline2();
```

```
void print();

//all possible ascii characters that can be read from the keyboard
char Symbol1[16] = {'*', '0', '#', 'D', '7', '8', '9', 'C','4', '5', '6', 'B',
'1', '2','3', 'A'};
char value_to_print[20]; //string buffer to convert float voltage value to string

char our_toascii(int myr2524){
    counter = 0;
    if(myr2524 != 0x0000){ //rotate until you find a set bit and return the
corresponding character
        while((myr2524 & 0x0001) != 0x0001){
            myr2524 = myr2524 >> 1;
            counter += 1;
        }
        return Symbol1[counter];
    }
    else
        return 0;
}

void PWM_init()
{
    //set TMR0 in fast PWM mode with non-inverted output, prescale=8
    TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS01); // WGM for fast
PWM, COM0 for non-inverting mode and CS for prescaler=8
    DDRB|=(1<<PB3); //set PB3 pin as output
}

void ADC_init(void){
    ADMUX = 0x40; //PINA as input for the ADC
    ADCSRA = (1<<ADEN | 1<<ADIE | 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0); //Enable ADC,
enable ADC conversion complete interrupt, division factor 128
}

ISR(TIMER1_OVF_vect){
    ADCSRA = ADCSRA | 0x40; // ADC Enable, ADC Interrupt Enable and f = CLK/128
}

ISR(ADC_vect) {
    value_read=ADC*5.0; //convert digital ADC Value to analog voltage value
    value_read/=1024;

    if(!((value_read-last_value<0.01)&&(value_read-last_value>-0.01))) { //if the
reading has changed since the last reading
```

```
        last_value=value_read; //update last value read
        lcd_clear(); //clear the lcd to print the new value
        sprintf(value_to_print, "%.2f", value_read); //convert float read to a
string in order to print it
        cli(); //disable interrupts to ensure printing will be performed normally
        print('V'); //print "Vo1" and change line (normally by sending command
0xC0)
        print('o');
        print('1');
        //changeline2()
        print('\n'); //for the simulation we send endl character
        //for(int m=0; m<22-3; ++m)
            //print('a');
        //wait_msec(30);
        for(int m=0; m<strlen(value_to_print); ++m) //print voltage
            print(value_to_print[m]);
        sei(); //re-enable interrupts
    }
    TCNT1 = 61630; //reset counter for interrupt after 0.5s
}

int main ()
{
    DDRC = 0xF0; //4 MSBs of PORTC as outputs 4 LSBs as inputs
    DDRB = 0xFF; //PORTB as output
    DDRD = 0xFF; //PORTD as output
    int read, ascii, timer0;
    PWM_init(); //initialize PWM
    initialize_variable(); //initialize _tmp_ used for reading from keyboard
    TIMSK = (1 << TOIE1); //Enable TCNT1 overflow interrupt
    TCCR1B = (1<<CS12) | (0<<CS11) | (1<<CS10); //CLK/1024
    TCNT1 = 61630; //interrupt after 0.5s


    lcd_init_sim(); //initialize the lcd
    ADC_init();  //initialize ADC
    value_read=0.0;
    last_value=6.0; //invalid last value to get a print at the start of the
program
    sei(); //enable interrupts

    timer0 = 0; //PWM duty cycle control variable. Its value will be assigned to
OCR0

    while(1){
```

```
        OCR0 = timer0; //update duty cycle

        read = 0; //scan the keyboard until a key is pressed
        while(read == 0x0000){
            read = scan_keypad_rising_edge_sim();
        }
        ascii = our_toascii(read) - 48; //convert from ascii to the actual number
        switch(ascii){
            case 1: //if 1 was pressed increase timer 0 therefore increasing duty
cycle
                timer0++;
                break;
            case 2: //if 2 was pressed decrease timer 0 therefore decreasing duty
cycle
                timer0--;
                break;
        }
    }
}
```

AssemblyFunctions.S:

```
#include <avr/io.h>
#define _SFR_ASM_COMPAT 1
#define __SFR_OFFSET 0 //required in order to use the I/O ports in the assembly
.DATA
_tmp_: .byte 2

.TEXT
.global scan_keypad_rising_edge_sim
scan_keypad_rising_edge_sim:
push r22 ; αποθήκευσε τους καταχωρητές r23:r22 και τους
push r23 ; r26:r27 γιατι τους αλλάζουμε μέσα στην ρουτίνα
push r26
push r27
rcall scan_keypad_sim ; έλεγξε το πληκτρολόγιο για πιεσμένους διακόπτες
push r24 ; και αποθήκευσε το αποτέλεσμα
push r25
ldi r24 ,15 ; καθυστέρησε 15 ms (τυπικές τιμές 10-20 msec που καθορίζεται από τον
ldi r25 ,0 ; κατασκευαστή του πληκτρολογίου – χρονοδιάρκεια σπινθηρισμών)
rcall wait_msec
rcall scan_keypad_sim ; έλεγξε το πληκτρολόγιο ξανά και απόρριψε
pop r23 ; όσα πλήκτρα εμφανίζουν σπινθηρισμό
pop r22
```

```
and r24 ,r22
and r25 ,r23
ldi r26 ,lo8(_tmp_) ; φόρτωσε την κατάσταση των διακοπτών στην
ldi r27 ,hi8(_tmp_) ; προηγούμενη κλήση της ρουτίνας στους r27:r26
ld r23 ,X+
ld r22 ,X
st X ,r24 ; αποθήκευσε στη RAM τη νέα κατάσταση
st -X ,r25 ; των διακοπτών
com r23
com r22 ; βρες τους διακόπτες που έχουν «μόλις» πατηθεί
and r24 ,r22
and r25 ,r23
pop r27 ; επανάφερε τους καταχωρητές r27:r26
pop r26 ; και r23:r22
pop r23
pop r22
ret

.global initialize_variable
initialize_variable:
ldi r24, 0xFF
sts _tmp_, r24 ; initialize _tmp_ to 0xFF
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)
```

```
scan_row_sim:
out PORTC, r25 ; η αντίστοιχη γραμμή τίθεται στο λογικό ‘1’
push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25 ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24,0xF4 ; πρόσβασης
ldi r25,0x01
rcall wait_usec
pop r25
pop r24 ; τέλος τμήμα κώδικα
nop
nop ; καθυστέρηση για να προλάβει να γίνει η αλλαγή κατάστασης
in r24, PINC ; επιστρέφουν οι θέσεις (στήλες) των διακοπτών που είναι πιεσμένοι
andi r24 ,0x0f ; απομονώνονται τα 4 LSB όπου τα ‘1’ δείχνουν που είναι πατημένοι
ret ; οι διακόπτες.

scan_keypad_sim:
push r26 ; αποθήκευσε τους καταχωρητές r27:r26 γιατι τους
push r27 ; αλλάζουμε μέσα στην ρουτίνα
ldi r25 , 0x10 ; έλεγξε την πρώτη γραμμή του πληκτρολογίου (PC4: 1 2 3 A)
rcall scan_row_sim
swap r24 ; αποθήκευσε το αποτέλεσμα
mov r27, r24 ; στα 4 msb του r27
ldi r25 ,0x20 ; έλεγξε τη δεύτερη γραμμή του πληκτρολογίου (PC5: 4 5 6 B)
rcall scan_row_sim
add r27, r24 ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r27
ldi r25 , 0x40 ; έλεγξε την τρίτη γραμμή του πληκτρολογίου (PC6: 7 8 9 C)
rcall scan_row_sim
swap r24 ; αποθήκευσε το αποτέλεσμα
mov r26, r24 ; στα 4 msb του r26
ldi r25 ,0x80 ; έλεγξε την τέταρτη γραμμή του πληκτρολογίου (PC7: * 0 # D)
rcall scan_row_sim
add r26, r24 ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r26
movw r24, r26 ; μετέφερε το αποτέλεσμα στους καταχωρητές r25:r24
clr r26 ; προστέθηκε για την απομακρυσμένη πρόσβαση
out PORTC,r26 ; προστέθηκε για την απομακρυσμένη πρόσβαση
pop r27 ; επανάφερε τους καταχωρητές r27:r26
pop r26
ret


.global lcd_init_sim
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
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 προγραμματος 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 προγραμματος 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 προγραμματος 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

.global lcd_clear
lcd_clear:
ldi r24,0x01 ; καθαρισμός της οθόνης
rcall lcd_command_sim


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

write_2_nibbles_sim:
push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25 ; λειτουργία του προγραμματος 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 προγραμματος 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 ; αποθήκευσε τους καταχωρητές r25:r24 γιατί τους
push r25 ; αλλάζουμε μέσα στη ρουτίνα
sbi PORTD, PD2 ; επιλογή του καταχωρητή δεδομένων (PD2=1)
rcall write_2_nibbles_sim ; αποστολή του byte
ldi r24 ,43 ; αναμονή 43μsec μέχρι να ολοκληρωθεί η λήψη
ldi r25 ,0 ; των δεδομένων από τον ελεγκτή της lcd
rcall wait_usec
pop r25 ;επανάφερε τους καταχωρητές r25:r24
pop r24
ret



.global print
print:
rcall lcd_data_sim

.global changeline
changeline:
ldi r24,0xC0
rcall lcd_command_sim
ldi r24,0xC1
rcall lcd_command_sim
```

```
.global changeline2
changeline2:
ldi r24,0xC0
rcall lcd_command_sim
```