

3 η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Ονοματεπώνυμο: Παγώνης Γιώργος
ΑΜ: 03117030

Ζήτημα 3.1 Γράψτε ένα πρόγραμμα «ηλεκτρονικής κλειδαριάς» το οποίο να ανάβει όλα τα leds PB0-7 για 4 sec συνολικά, μόνο όταν πατηθούν στη σειρά τα δύο πλήκτρα στο keypad 4x4 που αντιστοιχούν στο διψήφιο αριθμό της ομάδας σας (π.χ. 09). Αν δεν έχουν δοθεί οι δύο σωστοί αριθμοί να αναβοσβήνει (χρόνος ~0.5 sec αναμμένο και ~0.5 sec σβησμένο) τα leds PB0-7 επίσης για 4 sec. Μετά το πάτημα δύο αριθμών το πρόγραμμα να συνεχίσει να διαβάζει και άλλους αριθμούς από το keypad αλλά να τους αγνοεί. Το πρόγραμμα αυτό να είναι συνεχόμενης λειτουργίας. Δώστε το διάγραμμα ροής και το πρόγραμμα σε C. Επίσης η ρουτίνα keypad_to_ascii να δοθεί σε μορφή συνάρτησης C για να αξιοποιηθεί από το πρόγραμμα σε C.

```
/*
 * GccApplication12.c
 *
 * Created: 11/21/2020 3:01:57 PM
 * Author : georg
 */

#define F_CPU 8000000UL //needs to be defined before including the avr/delay.h library
#define SPARK_DELAY_TIME 20
#define FIRST_DIGIT 'C'
#define SECOND_DIGIT '3'

#include <avr/io.h>
#include <util/delay.h>

unsigned int previous_keypad_state = 0; //hold the state of the keyboard 0x0000
int ascii[16]; //Is the ascii code for each key on the keyboard

unsigned char scan_row_sim(int row)
```

```

{
    unsigned char temp;
    volatile unsigned char pressed_row;

    temp = 0x08;
    PORTC = temp << row;
    _delay_us(500);
    asm("nop");
    asm("nop");
    pressed_row = PINC & 0x0f;

    return pressed_row;
}
unsigned int scan_keypad_sim(void)
{
    volatile unsigned char pressed_row1, pressed_row2, pressed_row3, pressed_row4;
    volatile unsigned int pressed_keypad = 0x0000;

    pressed_row1 = scan_row_sim(1);
    pressed_row2 = scan_row_sim(2);
    pressed_row3 = scan_row_sim(3);
    pressed_row4 = scan_row_sim(4);

    pressed_keypad = (pressed_row1 << 12 | pressed_row2 << 8) | (pressed_row3 << 4) |
    (pressed_row4);
    PORTC = 0x00;
    return pressed_keypad;
}
unsigned int scan_keypad_rising_edge_sim(void)
{
    unsigned int pressed_keypad1, pressed_keypad2, current_keypad_state, final_keypad_state;

    pressed_keypad1 = scan_keypad_sim();
    _delay_ms(SPARK_DELAY_TIME);
    pressed_keypad2 = scan_keypad_sim();
    current_keypad_state = pressed_keypad1 & pressed_keypad2;
    final_keypad_state = current_keypad_state & (~ previous_keypad_state);
    previous_keypad_state = current_keypad_state;

    return final_keypad_state;
}
unsigned char keypad_to_ascii_sim(unsigned int final_keypad_state)
{
    volatile int j;

```

```

volatile unsigned int temp;

for (j=0; j<16; j++)
{
    temp = 0x01;
    temp = temp << j;
    if (final_keypad_state & temp) //if you find the only pressed key then return
    {
        return ascii[j];
    }
}
//should not reach here
return 1;
}

void initialize_ascii(void)
{
    ascii[0] = '*';
    ascii[1] = '0';
    ascii[2] = '#';
    ascii[3] = 'D';
    ascii[4] = '7';
    ascii[5] = '8';
    ascii[6] = '9';
    ascii[7] = 'C';
    ascii[8] = '4';
    ascii[9] = '5';
    ascii[10] = '6';
    ascii[11] = 'B';
    ascii[12] = '1';
    ascii[13] = '2';
    ascii[14] = '3';
    ascii[15] = 'A';
}

unsigned char read4x4(void)
{
    unsigned int keypad_state;
    unsigned char ascii_code;

    keypad_state = scan_keypad_rising_edge_sim(); // read the state of the keyboard
    if (!keypad_state)
    {
        return 0;
    }
    ascii_code = keypad_to_ascii_sim(keypad_state); // encode it to ascii code

```

```

        return ascii_code;
    }

int main(void)
{
    int i;
    volatile unsigned char first_number, second_number;

    DDRB = 0Xff; // B for output
    DDRC = 0xf0; // c 4 msb for output and 4 lsb for input

    initialize_ascii();

    while (1)
    {
        do
        {
            first_number = read4x4(); // wait for the number to be pushed
        }
        while(!first_number);

        do
        {
            second_number = read4x4(); // wait for the second number to be pushed
        }
        while(!second_number);
        // compare it with the given number (here C3)
        if ((first_number == FIRST_DIGIT) & (second_number == SECOND_DIGIT))
        {
            //if true the just open the leds for 4 sec
            PORTB = 0Xff;
            _delay_ms(4000);
            PORTB = 0X00;
        }
        else
        {
            //if false just open and close the leds with T=0.5 sec
            for (i=0; i<4; i++)
            {
                PORTB = 0Xff;
                _delay_ms(500);
                PORTB = 0X00;
                _delay_ms(500);
            }
        }
    }
}

```

```

        }
    }
}

return 0;
}

```

Ζήτημα 3.2 Να ξαναγραφεί το παραπάνω πρόγραμμα του Ζητήματος 3.1 σε assembly αλλά να προσθέσετε την εξής λειτουργία: Να απεικονίζεται στο LCD display το μήνυμα ALARM ON στην περίπτωση που δοθεί λάθος διψήφιος αριθμός από το πληκτρολόγιο. Σε διαφορετική περίπτωση να αναγράφει WELCOME XX, όπου XX ο αριθμός της ομάδας.

```

;;
; AssemblerApplication10.asm
;
; Created: 11/21/2020 4:45:46 PM
; Author : georg
;

```

```

.include "m16def.inc"

```

```

;-----DATA SEGMENT-----

```

```

.DSEG

```

```

    _tmp_ .byte 2

```

```

;-----CODE SEGMENT-----

```

```

.CSEG

```

```

    .org 0x0

```

```

    rjmp RESET

```

RESET:

.equ FIRST_DIGIT = 'C'

.equ SECOND_DIGIT = '3'

.def temp=r16

.def buttons_pressed=r17

.def first_number=r18

.def second_number=r19

.def loop_error_counter=r20

clr buttons_pressed

clr first_number

clr second_number

ldi loop_error_counter,4

ldi temp,LOW(RAMEND)

out SPL, temp

ldi temp,HIGH(RAMEND)

out SPH, temp ;initialize the stack

ser temp

out DDRB, temp ;PORTB (output)

ldi temp,(1<<PC7)|(1<<PC6)|(1<<PC5)|(1<<PC4)

out DDRC,temp ;PORTC is used by READ4X4

START:

ldi r24,20 ;20 msec delay in READ4X4 for sparks

rcall READ4X4 ;input r22, output r24 with the ascii code of the pressed button

cpi r24,0 ;if a button is pressed-->r24!=0

breq START ;loop here while (no button pressed)

push r24 ;when a button is pressed save its ascii

inc buttons_pressed ;increment the number of pressed buttons

cpi buttons_pressed,2

brne START ;when 2 buttons are pressed stop reading and evaluate

EVALUATE:

pop second_number

pop first_number

cpi first_number,FIRST_DIGIT

brne ERROR

cpi second_number,SECOND_DIGIT

brne ERROR

SUCCESS: ;reached here because both buttons where the right ones

ser r24

out DDRD ,r24

clr r24

```

rcall lcd_init_sim
ldi r24,'W'
rcall lcd_data_sim
ldi r24,'E'
rcall lcd_data_sim
ldi r24,'L'
rcall lcd_data_sim
ldi r24,'L'
rcall lcd_data_sim
ldi r24,'C'
rcall lcd_data_sim
ldi r24,'O'
rcall lcd_data_sim
ldi r24,'M'
rcall lcd_data_sim
ldi r24,'E'
rcall lcd_data_sim
ldi r24,' '
rcall lcd_data_sim
ldi r24,FIRST_DIGIT
rcall lcd_data_sim
ldi r24,SECOND_DIGIT
rcall lcd_data_sim

```

```

clr buttons_pressed ;make number of pressed buttons ZERO for the next check of
numbers

```

```

ldi r24,low(4000)
ldi r25,high(4000)
ser temp
out PORTB,temp
rcall wait_msec
clr temp
out PORTB,temp
clr r24
rcall lcd_init_sim
rjmp START

```

```

ERROR: ;reached here(jumping SUCCESS flag) because one
or two buttons wrong

```

```

clr buttons_pressed ;make number of pressed buttons ZERO for the next check of
numbers

```

```

LOOP_ERROR: ;this loop implements ON-->OFF frequency=1/250 Hz

```

```

ser r24
out DDRD, r24 ; αρχικοποίηση PORTD που συνδέεται η οθόνη, ως έξοδος
clr r24

```

```

rcall lcd_init_sim ; αρχικοποίηση οθόνης
ldi r24, 'A'
rcall lcd_data_sim ; αποστολή ενός byte δεδομένων στον ελεγκτή της οθόνης lcd
ldi r24, 'L'
rcall lcd_data_sim
ldi r24, 'A'
rcall lcd_data_sim
ldi r24, 'R'
rcall lcd_data_sim
ldi r24, 'M'
rcall lcd_data_sim
ldi r24, ''
rcall lcd_data_sim
ldi r24, 'O'
rcall lcd_data_sim
ldi r24, 'N'
rcall lcd_data_sim
ldi r24, low(500)
ldi r25, high(500)
ser temp
out PORTB, temp
rcall wait_msec
ldi r24, low(500)
ldi r25, high(500)
clr temp
out PORTB, temp
rcall wait_msec
dec loop_error_counter
cpi loop_error_counter, 0
brne LOOP_ERROR
ldi loop_error_counter, 4
clr r24
rcall lcd_init_sim
rjmp START

```

READ4X4:

```

push r22                ;save r22
push r23                ;save r23
push r25                ;save r25
push r26                ;save r26
push r27                ;save r27
in r27, SREG
push r27                ;save SREG

```



```
rcall scan_keypad_rising_edge_sim
rcall keypad_to_ascii_sim
```

```
pop r27
out SREG,r27          ;pop SREG
pop r27                ;pop r27
pop r26                ;pop r26
pop r25                ;pop r25
pop r23                ;pop r23
pop r22                ;pop r22
ret
```

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 προγράμματος απομακρυσμένης
ldi r24,low(500) ; πρόσβασης
ldi r25,high(500)
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 διακοπών 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 γιατί 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
```

scan_keypad_rising_edge_sim:

```
push r22 ; αποθήκευσε τους καταχωρητές r23:r22 και τους
push r23 ; r26:r27 γιατί 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, low(_tmp_) ; φόρτωσε την κατάσταση των διακοπών στην
ldi r27, high(_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 διακοπών
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
```

keypad_to_ascii_sim:

```
push r26 ; αποθήκευσε τους καταχωρητές r27:r26 γιατί τους
push r27 ; αλλάζουμε μέσα στη ρουτίνα
movw r26, r24 ; λογικό '1' στις θέσεις του καταχωρητή r26 δηλώνουν
; τα παρακάτω σύμβολα και αριθμούς
ldi r24, '*'
```

```

; r26
; C 9 8 7 D # 0 *
sbrc r26,0
rjmp return_ascii
ldi r24,'0'
sbrc r26,1
rjmp return_ascii
ldi r24,'#'
sbrc r26,2
rjmp return_ascii
ldi r24,'D'
sbrc r26,3 ; αν δεν είναι '1' παρακάμπτει την ret, αλλιώς (αν είναι '1')
rjmp return_ascii ; επιστρέφει με τον καταχωρητή r24 την ASCII τιμή του D.
ldi r24,'7'
sbrc r26,4
rjmp return_ascii
ldi r24,'8'
sbrc r26,5
rjmp return_ascii
ldi r24,'9'
sbrc r26,6
rjmp return_ascii ;
ldi r24,'C'
sbrc r26,7
rjmp return_ascii
ldi r24,'4' ; λογικό '1' στις θέσεις του καταχωρητή r27 δηλώνουν
sbrc r27,0 ; τα παρακάτω σύμβολα και αριθμούς
rjmp return_ascii
ldi r24,'5'
; r27
; A 3 2 1 B 6 5 4
sbrc r27,1
rjmp return_ascii
ldi r24,'6'
sbrc r27,2
rjmp return_ascii
ldi r24,'B'
sbrc r27,3
rjmp return_ascii
ldi r24,'1'
sbrc r27,4
rjmp return_ascii ;
ldi r24,'2'
sbrc r27,5

```

```

rjmp return_ascii
ldi r24,'3'
sbrc r27,6
rjmp return_ascii
ldi r24,'A'
sbrc r27,7
rjmp return_ascii
clr r24
rjmp return_ascii
return_ascii:
pop r27 ; επανάφερε τους καταχωρητές r27:r26
pop r26
ret

```

write_2_nibbles_sim:

```

push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25 ; λειτουργία του προγράμματος απομακρυσμένης
ldi r24,low(6000) ; πρόσβασης
ldi r25,high(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,low(6000) ; πρόσβασης
ldi r25,high(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

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,low(1000) ; πρόσβασης

```

ldi r25,high(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 ,low(1000) ; πρόσβασης
ldi r25 ,high(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 ,low(1000) ; πρόσβασης
ldi r25 ,high(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, low(1530)
ldi r25, high(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:                                ;1msec in total
    push r24                             ;2 cycles (0.250usec)
    push r25                             ;2 cycles (0.250usec)
    ldi r24,low(998)                     ;1 cycle (0.125usec)
    ldi r25,high(998)                   ;1 cycle (0.125usec)
    rcall wait_usec                     ;3 cycles (0.375usec)
    pop r25                             ;2 cycles (0.250usec)
    pop r24                             ;2 cycles (0.250usec)
    sbiw r24,1                          ;2 cycle (0.250usec)
    brne wait_msec                     ;1 or 2 cycles
    ret                                ;4 cycles (0.500usec)
```

```
wait_usec:                                ;998.375usec in total
    sbiw r24,1                          ;2 cycles (0.250usec)
    nop                                ;1 cycle (0.125usec)
    nop                                ;1 cycle (0.125usec)
    nop                                ;1 cycle (0.125usec)
    nop                                ;1 cycle (0.125usec)
    brne wait_usec                     ;1 or 2 cycles (0.125 or 0.250usec)
    ret                                ;4 cycles (0.500usec)
```