ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ

(2020-2021)

4η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ

Ονοματεπώνυμο:

- Χρήστος Τσούφης

Αριθμός Μητρώου:

- 03117176

Ομάδα Εργαστηρίου:

- B19

Εξέταση – Επίδειξη:

- 20/1/2021

Στοιχεία Επικοινωνίας:

- el17176@mail.ntua.gr

1η Άσκηση

Ο πηγαίος κώδικας, μαζί με τα απαραίτητα σχόλια:

Κώδικας σε **assembly**:

```
.include "m16def.inc"
.def flag = r16
.def leds = r17
.def entered_correct = r18
.DSEG

_tmp_: .BYTE 2
.CSEG
.org 0x0
jmp reset
.org 0x10
rjmp ISR_TIMER1_OVF
.org 0x1c
rjmp ADC_ISR

reset:

ldi r24, low(RAMEND)
out SPL, r24
ldi r24, high(RAMEND)
out SPH, r24                ; αρχικοποίηση stack pointer
ldi r24, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
                            ; θέτει ως εξόδους τα 4 MSB
out DDRC, r24               ; της θύρας PORTC
ser r24                     ; θέτουμε στο 1 τον reg r24
out DDRB, r24
out DDRD, r24               ; αρχικοποίηση PORTD που συνδέεται η οθόνη
                            ; ως έξοδος
clr r24
rcall lcd_init_sim          ; αρχικοποίηση οθόνης
rcall ADC_init              ; αρχικοποίηση adc converter
rcall TIM1_init             ; αρχικοποίηση timer interrupt
sei                         ; enable interrupts

first_digit:

ldi r24, 0xf0
rcall scan_keypad_rising_edge_sim  ; ελέγχω τις εξόδους
clr r22                            ; αρχικοποίηση στο 0
or r22, r24                        ; τα γράφω εκεί για έλεγχο αλλαγών
or r22, r25
cpi r22, 0
breq first_digit
cpi r25, 0x10                      ; δηλ. το ψηφιο στην θεση 16 στον r25
brne wrong_first
cpi r24, 0                         ; δηλ. κανένα ψηφιο στον r24
brne wrong_first
rjmp second_digit

wrong_first:

ldi r21, 1                         ; flag that indicates first digit was
                                   ; incorrect
```

```
second_digit:

ldi r24, 0xf0                       ; περνάω σε όλα τα πλήκτρα άσσο
rcall scan_keypad_rising_edge_sim   ; έλεγχος εξόδου
clr r22                             ; αρχικοποίηση στο 0
or r22, r24                         ; τα γράφω εκεί για έλεγχο αλλαγών
or r22, r25
cpi r22,0
breq second_digit
cpi r21,1
breq wrong_passwd
cpi r24,0x40                        ; δηλ. το ψηφιο στην θεση 32 στον r24
brne wrong_passwd
cpi r25,0                           ; δηλ. κανένα ψηφιο στον r25
brne wrong_passwd

right_passwd:

rcall scan_keypad_rising_edge_sim
ldi entered_correct,0x01            ; set 1st bit of flag in order not to change
                                    ; lcd display during that time
ldi r24,0x01
rcall lcd_command_sim               ; clean display
ldi r24, low(1530)                  ; clean display delay
ldi r25, high(1530)
rcall wait_usec
rcall display_welcome
ldi r24, low(4000)                  ; load r25:r24 with 4000
ldi r25, high(4000)
rcall leds_on                       ; leds_on
rcall wait_msec                     ; delay 4 seconds
rcall leds_off
ldi r24,0x01
rcall lcd_command_sim               ; clean display
ldi r24, low(1530)                  ; clean display delay
ldi r25, high(1530)
rcall wait_usec
ldi entered_correct,0x00
; WARNING: ori flag,0x10            ; set 1st bit of flag to indicate that
                                    ; correct_password process is over
rjmp first_digit                    ; repeat

wrong_passwd:

rcall scan_keypad_rising_edge_sim
ldi r23, 0x04                       ; counter for blinking 4 times

leds_loop:

rcall leds_on
ldi r24, low(500)                   ; load r25:r24 with 500
ldi r25, high(500)
rcall wait_msec                     ; delay 0.5 seconds
rcall leds_off                      ; leds_off
ldi r24, low(500)                   ; load r25:r24 with 500
ldi r25, high(500)
rcall wait_msec                     ; delay 0.5 seconds
dec r23
brne leds_loop
rjmp first_digit                    ; repeat
```

```
display_gas:

rcall lcd_init_sim
clr r24
ldi r24, 'G'                    ; gas message
rcall lcd_data_sim
ldi r24, 'A'
rcall lcd_data_sim
ldi r24, 'S'
rcall lcd_data_sim
ldi r24, ' '
rcall lcd_data_sim
ldi r24, 'D'
rcall lcd_data_sim
ldi r24, 'E'
rcall lcd_data_sim
ldi r24, 'T'
rcall lcd_data_sim
ldi r24, 'E'
rcall lcd_data_sim
ldi r24, 'C'
rcall lcd_data_sim
ldi r24, 'T'
rcall lcd_data_sim
ldi r24, 'E'
rcall lcd_data_sim
ldi r24, 'D'
rcall lcd_data_sim
ret

display_clear:

rcall lcd_init_sim
clr r24
ldi r24, 'C'                    ; clear message
rcall lcd_data_sim
ldi r24, 'L'
rcall lcd_data_sim
ldi r24, 'E'
rcall lcd_data_sim
ldi r24, 'A'
rcall lcd_data_sim
ldi r24, 'R'
rcall lcd_data_sim
ret

display_welcome:

rcall lcd_init_sim
clr r24
ldi r24, 'W'                    ; welcome message
rcall lcd_data_sim
ldi r24, 'E'
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
ret

leds_on:

ori leds,0x80                           ; set 8th bit of leds
out PORTB, leds
ret

leds_off:
andi leds,0x7F                          ; clear 8th bit of leds
out PORTB, leds
ret

ADC_init:

ldi r24,(1<<REFS0)                      ; Vref: Vcc
out ADMUX,r24                           ; MUX4:0 = 00000 for A0.
ldi r24,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
out ADCSRA,r24
reti

TIM1_init:

ldi r24, (1 << TOIE1)                   ; enable interrupt overflow of timer1
out TIMSK,r24
ldi r24,(1<<CS12) | (0<<CS11) | (1<<CS10) ; CK/1024
out TCCR1B,r24
ldi r24,0xfc                            ; interrupt every 100 ms
out TCNT1H,r24
ldi r24,0xf3
out TCNT1L,r24
reti

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
                                        ; 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
                                ; 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
push r23                        ; r26:r27 γιατι τους αλλάζουμε στη 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 διακοπτών 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 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

keypad_to_ascii_sim:

push r26                        ; αποθήκευσε τους καταχωρητές r27:r26 γιατί
push r27                        ; τους 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
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
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
push r25                        ; τους 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
push r25                        ; τους 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 προγραμματος 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 προγραμματος 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 προγραμματος 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:

push r24                        ; 2 κύκλοι (0.250 μsec)
push r25                        ; 2 κύκλοι
ldi r24 , low(998)              ; φόρτωσε τον καταχ. r25:r24 με 998
                                ; (1 κύκλος - 0.125 μsec)
ldi r25 , high(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)
```

```
ISR_TIMER1_OVF:

push r24
in r24,ADCSRA                   ; begin adc conversion
ori r24,(1<<ADSC)
out ADCSRA, r24
ldi r24,0xfc                    ; next interrupt again in 100 ms
out TCNT1H ,r24
ldi r24 ,0xf3
out TCNT1L ,r24
pop r24
reti

ADC_ISR:

push r24
push r25
push r26
clr r26                         ; r26 will have the level of
                                ; gas concentration
in r24, ADCL                    ; keep in r25:r24 the result of
                                ; adc conversion
in r25, ADCH
andi r25,0x03                   ; keep the 2 lsbs since result is 8 bits
cpi r25, 0x01
brsh five_plus                  ; up to 4 leds all bits of r5 are 0

cpi r24, 0x63                   ; 0 < ppm < 30
brlo one_led
cpi r24, 0x98                   ; 30 < ppm < 50
brlo two_leds
cpi r24, 0xCD                   ; 50 < ppm < 70
brlo three_leds
cpi r24, 0xE7                   ; 70 < ppm < 80
brlo four_leds

five_plus:

cpi r24, 0x29                   ; 80 <ppm < 105
brlo five_leds
cpi r24, 0x86                   ; 105 <ppm < 140
brlo six_leds
ldi r26, 0x40                   ; ppm >= 140
rjmp fixed_leds

one_led:

ldi r26, 0x01
rjmp fixed_leds

two_leds:

ldi r26, 0x02
rjmp fixed_leds

three_leds:

ldi r26, 0x04
rjmp fixed_leds

four_leds:
```

```
ldi r26, 0x08
rjmp fixed_leds

five_leds:

ldi r26, 0x10
rjmp fixed_leds

six_leds:

ldi r26, 0x20

fixed_leds:

andi leds,0x80                  ; keep the 8th bit
or leds,r26                     ; add the level of gas concentration
out PORTB,leds                  ; display it in PORTB
sbrc entered_correct,0          ; if lsb of entered_correct is set,
                                ; then we are during welcome message so
                                ; we keep led on
                                ; (without blinking if CO was above level)
jmp exit                        ; even if CO was above level,
                                ; it is good
                                ; practice to keep it that way
cpi r26,0x08                    ; else, check if level below 4th bit
                                ; indicating < 70 ppm
brlo clear                      ; if yes it's clear
cpi flag,0x01                   ; else check if the previous check indicated
                                ; danger and the led was on

breq on_off
cpi flag,0x03                   ; check if the previous check indicated
                                ; danger and the led was off

breq on_ex
ldi flag,0x01                   ; else if the previous check was ok,
                                ; set 1st bit of flag

ldi r24,0x01
rcall lcd_command_sim           ; clean display
ldi r24, low(1530)              ; clean display delay
ldi r25, high(1530)
rcall wait_usec
rcall display_gas               ; display gas message
jmp exit

on_off:

ori flag,0x02                   ; set 2nd bit of flag so on next interrupt
                                ; it is turned on
andi leds,0x80                  ; keep only 8th bit of leds
out PORTB, leds                 ; in order to turn off led of level of gas
jmp exit

on_ex:

andi flag,0xFD                  ; clear 2nd bit of flag in order to have
                                ; led of level of gas turned on

jmp exit

clear:

cpi flag,0x00                   ; check if previous check was ok or we are
                                ; at the beginning

breq exit
```

```
sbrs flag, 0                    ; check if previous check indicated
                                ; dangerous level of gas
jmp exit
ldi flag,0x00                   ; clear flag to show that current check is ok
ldi r24,0x01
rcall lcd_command_sim           ; clean display
ldi r24, low(1530)              ; clean display delay
ldi r25, high(1530)
rcall wait_usec
rcall display_clear

exit:

pop r26
pop r25
pop r24
reti
```

2η Άσκηση

Ο πηγαίος κώδικας, μαζί με τα απαραίτητα σχόλια:

Κώδικας σε **C**:

```
#include<avr/io.h>
#include<avr/interrupt.h>
// Enable global interrupts by setting global interrupt enable bit in SREG

int prev = -1;
int cur  =  0;
int anoixto = 0;

ISR(ADC_vect){
	unsigned short  value = ADCL|(ADCH<<8);  // read the value
	PORTB &= 0x80;
	if (value < 112){
		cur = 1;
		PORTB |= 0x01;
	}
	else if(value<205 && value >= 112){
		cur =2;
		PORTB |= 0x02;
	}
	else if(value<297 && value >= 205){
		cur  = 3;
		if (cur == prev){
			if(anoixto==0){
				PORTB |= 0x04;
				anoixto =1;
				}else{
				anoixto = 0;
			}
			}else{
			PORTB |= 0x04;
			anoixto = 1;
		}
	}
	else if(value<390 && value >=297 ){
		cur  = 4;
		if (cur == prev){
			if(anoixto==0){
				PORTB |= 0x08;
				anoixto = 1;
				}else{
				anoixto = 0;
			}
			}else{
			PORTB |= 0x08;
			anoixto = 1;
		}
	}
	else if(value<482 && value >= 390){
		cur  = 5;
		if (cur == prev){
			if(anoixto==0){
				PORTB |= 0x10;
				anoixto =1;
				}else{
				anoixto = 0;
			}
```

```
                        }else{
                        PORTB |= 0x10;
                        anoixto = 1;
                }
        }
        else if(value<575 && value >= 482){
                cur  = 6;
                if (cur == prev){
                        if(anoixto==0){
                                PORTB |= 0x20;
                                anoixto =1;
                        }
                        else{
                                anoixto = 0;
                        }

                        }else{
                        PORTB |= 0x20;
                        anoixto = 1;
                }
        }
        else if(value>575 ){
                cur  = 7;
                if (cur == prev){
                        if(anoixto==0){
                                PORTB |= 0x40;
                                anoixto =1;
                                }else{
                                anoixto = 0;
                        }
                        }else{
                        PORTB |= 0x40;
                        //an einai diaforetiko prev me cur kane to prwto anama
                        anoixto = 1;
                }
        }
        prev = cur;
        // end of ISR
}
ISR(TIMER1_OVF_vect)    // Timer1 ISR
{
        sei();
        ADMUX = (1<<REFS0);  // MUX4:0 = 00000 for A0.
        ADCSRA  = (1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
        ADCSRA |= (1 << ADSC); // START A2D conversion
        asm("nop");
        // at this point you service the ADC routine ....
        // you are back
        TCNT1 = 0xfcf2;   // for 0.1 reload the timer
}

void wait_usec(unsigned short usec) {//function for usec delay
        while (usec != 0) {//1 loop is 1 usec delay
                asm("nop");
                asm("nop");
                asm("nop");
                asm("nop");
                usec = usec - 1;
        }
}

void wait_msec(unsigned short msec) {//function for msec delay
```

```
	while (msec != 0) {//1 loop is 1msec delay
		wait_usec(999);
		msec = msec - 1;
	}
}

unsigned char scan_row_sim (unsigned char row) {
	PORTC = row;
	//choose keypad row
	wait_usec(500);
	asm("nop");
	asm("nop");
	return (PINC & 0x0F);
	//get the state of the keys
}

unsigned short scan_keypad_sim () {
	unsigned short whole_keypad;
	//variable of 16bit which will contain the state of keypad
	unsigned char i;
	whole_keypad = 0;
	for (i = 0x10; i < 0x80; i = i*2 ) {
		whole_keypad = whole_keypad + scan_row_sim(i);
		//add the row to whole_keypad
		whole_keypad = whole_keypad * 16;
		//left shift 4 bits
	}
	whole_keypad = whole_keypad + scan_row_sim(i);
	//add the last row
	return whole_keypad;
}

unsigned short scan_keypad_rising_edge_sim (unsigned short *prev_key) {
	unsigned short first_keypad;
	first_keypad = scan_keypad_sim();
	wait_msec(15);
	unsigned short second_keypad;
	second_keypad = scan_keypad_sim();
	//first&second keypad to avoid spinthirismos
	unsigned short final_keypad;
	final_keypad = ((first_keypad & second_keypad) & ~(*prev_key));
	//AND previous_keypad to identify the rising edge of keypad
	*prev_key = first_keypad & second_keypad;
	//save the new state to prev_key
	return final_keypad;
}

unsigned char keypad_to_ascii_sim(unsigned short keypad) {
	unsigned char c;
	if (keypad == 1) c = '*';
	if (keypad == 2) c = '0';
	if (keypad == 4) c = '#';
	if (keypad == 8) c = 'D';
	if (keypad == 16) c = '7';
	if (keypad == 32) c = '8';
	if (keypad == 64) c = '9';
	if (keypad == 128) c = 'C';
	if (keypad == 256) c = '4';
	if (keypad == 512) c = '5';
	if (keypad == 1024) c = '6';
	if (keypad == 2048) c = 'B';
	if (keypad == 4096) c = '1';
```

```
        if (keypad == 8192) c = '2';
        if (keypad == 16384) c = '3';
        if (keypad == 32768) c = 'A';
        return c;
}

int main(void)
{
        TCNT1 = 64754;   // for 0.1 sec
        TCCR1A = 0x00;
        TCCR1B = (1<<CS10) |(0<<CS11) | (1<<CS12)  ;
        // Timer mode with 1024 prescler
        TIMSK = (1 << TOIE1) ;   // Enable timer1 overflow interrupt(TOIE1)
        sei();

        DDRB = 0xFF;
        DDRC = 0xF0;
        PORTB = 0x00;
        unsigned char ascii_key1, ascii_key2, ignore,x;
        unsigned short key1, prev_key;
        prev_key = 0x00;
        while (1)
        {
                while ((key1 = scan_keypad_rising_edge_sim(&prev_key)) == 0);
                //scan keypad until a key is pushed
                ascii_key1 = keypad_to_ascii_sim(key1);
                while ((key1 = scan_keypad_rising_edge_sim(&prev_key)) == 0);
                //scan for the second key
                ascii_key2 = keypad_to_ascii_sim(key1);
                if ((ascii_key1 == '1') && (ascii_key2 == '9')) {
                //if "19" is pressed LEDS ON
                        cli();
                        PORTB = 0x80;
                        for (x = 0; x < 16; x++) {
                                wait_msec(246);
                                ignore = scan_keypad_rising_edge_sim(&prev_key);
                        }
                        PORTB = 0x00;
                        sei();
                }
                else {//else LEDS ON-OFF
                        for (x = 0; x < 4; x++) {
                                PORTB |= 0x80 ;
                        //energopoieis to teleutaio alla epitrepeis diakopes
                                wait_msec(246);
                                ignore = scan_keypad_rising_edge_sim(&prev_key);
                                wait_msec(246);
                                ignore = scan_keypad_rising_edge_sim(&prev_key);
                                PORTB ^= 0x80;
                                wait_msec(246);
                                ignore = scan_keypad_rising_edge_sim(&prev_key);
                                wait_msec(246);
                                ignore = scan_keypad_rising_edge_sim(&prev_key);
                                sei();
                        }
                }
        }
}
```