Εργαστήριο Μικροϋπολογιστών – 4η Σειρά Ασκήσεων
Κυριακόπουλος Γιώργος – el18153

1η Άσκηση:

```
.include "m16def.inc"
.def leds = r16
.def flags = r17                ; flags(0) = correct_team
                                ; flags(1) = gas_error
                                ; flags(2) = blinker

.DSEG
_tmp_: .byte 2                  ; initialize _tmp_ for RAM

.CSEG
.org 0x00
rjmp main                       ; main program
.org 0x10
rjmp ISR_TIMER1_OVF             ; TIMER1 overflow interrupt routine
.org 0x1c
rjmp ISR_ADC                    ; ADC convertion interrupt routine

main:
    ldi r24, low(RAMEND)        ; initialize stack pointer
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24

    clr leds                    ; initialize leds
    clr flags                   ; initialize flags

    ser r24
    out DDRB, r24               ; set PORTB as output
    out DDRD, r24               ; set PORTD as output
    ldi r24, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
    out DDRC, r24               ; set PORTC[7:4] as output

    rcall lcd_init_sim          ; initialize lcd screen
    rcall ADC_init              ; initialize ADC

    ldi r24, (1 << TOIE1)       ; enable overflow interrupt 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 so calculate
    out TCNT1H, r24             ; initial value as 65536 - 0.1 *
    ldi r24, 0xf3               ; 8000000 / 1024 = 65536 - 781.25 =
    out TCNT1L, r24             ; 64754.75 -> 64755 = 0xfcf3
    sei                         ; enable interrupts

first:
    rcall scan_keypad_rising_edge_sim ; scan keypad
    clr r20                     ; add 2 registers with button values
    or r20, r24                 ; to r20 and check if it is 0
    or r20, r25                 ; if it is 0 no button was pressed
    cpi r20, 0                  ; so repeat reading
    breq first                  ; else, continue

    mov r19, r25                ; store buttons pressed (r25r24)
    mov r18, r24                ; to r1918 for later check

second:
    rcall scan_keypad_rising_edge_sim; scan keypad
    clr r20                     ; add 2 registers with button values
    or r20, r24                 ; to r20 and check if it is 0
    or r20, r25                 ; if it is 0 no button was pressed
    cpi r20, 0                  ; so repeat reading
    breq second                 ; else, continue

    cpi r19, 0x40               ; compare r19r18 to right value 3
    brne wrong_team             ; that is 0x40 on r19 and 0x00 on r18
    cpi r18, 0                  ; if any register doesn't have the
    brne wrong_team             ; right value, go to wrong_team
    cpi r25, 0x40               ; since the 2 digits are not the
    brne wrong_team             ; right ones, else continue to
    cpi r24, 0                  ; correct team
    brne wrong_team             ; also, do the same for r25r24

correct_team:
    ori flags, 0x01             ; set correct_team to 1
    rcall welcome               ; display welcome message

    ori leds, 0x80              ; turn PB7 on
    ldi r21, 0x50               ; initialize counter to 80 for delay
    rcall delay_between         ; call 50ms delay routine (4000ms)
    andi leds, 0x7f             ; turn PB7 off
```

```
    ldi r24, 0x01
    rcall lcd_command_sim       ; clear display after welcome message
    sbrc flags, 1               ; if gas_error is set
    rcall gas                   ; re-display gas message
    andi flags, 0xfe            ; set correct_team to 0
    rjmp first                  ; restart from the beginning

wrong_team:
    ldi r22, 0x04               ; initialize counter for 4 blinks
led_blink:
    ori leds, 0x80              ; turn PB7 on
    ldi r21, 0x0a               ; initialize counter to 10 for delay
    rcall delay_between         ; call 50ms delay routine (500ms)
    andi leds, 0x7f             ; turn PB7 off
    ldi r21, 0x0a               ; initialize counter to 10 for delay
    rcall delay_between         ; call 50ms delay routine (500ms)

    dec r22                     ; decrement blinking counter
    cpi r22, 0                  ; if it is not 0 repeat
    brne led_blink              ; else continue
    rjmp first                  ; restart from the beginning

delay_between:
    rcall scan_keypad_rising_edge_sim; scan keypad (15ms)
    ldi r24, low(35)            ; set registers for 35ms delay
    ldi r25, high(35)           ; so 15+35ms is equal to 50ms delay
    rcall wait_msec             ; call the msec delay routine
    dec r21                     ; decrement delay counter
    cpi r21, 0                  ; if it is not 0 repeat
    brne delay_between          ; else continue
    ret

welcome:
    rcall lcd_init_sim          ; initialize lcd screen
    ldi r24,'W'                 ; print the required message
    rcall lcd_data_sim          ; 'WELCOME' character by character
    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

gas:
    rcall lcd_init_sim          ; initialize lcd screen
    ldi r24,'G'                 ; print the required message
    rcall lcd_data_sim          ; 'GAS DETECTED' character by
    ldi r24,'A'                 ; character
    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

clear:
    rcall lcd_init_sim          ; initialize lcd screen
    ldi r24,'C'                 ; print the required message
    rcall lcd_data_sim          ; 'CLEAR' character by character
    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

; ============================================================== ;

ISR_TIMER1_OVF:
    push r24                    ; push r24 to stack
    in r24, ADCSRA              ; get ADCSRA register status
    ori r24, (1 << ADSC)        ; start the conversion by changing
    out ADCSRA, r24             ; only the ADSC bit to 1.
    ldi r24, 0xfc               ; reset the 100 ms interrupt timer
    out TCNT1H, r24
    ldi r24, 0xf3
    out TCNT1L, r24
    pop r24                     ; restore r24
    reti

; For the following routine we have used these equations:
;
; Vin = ADC * Vref / 1024 = ADC * 5 / 1024 = ADC / 204.8
; M = 129 * 10^2 * 10^(-9) * 10^3 = 129 * 10^(-4) = 129 / 10^4
; Cx = (1 / M) * (Vgas - Vags0) = (10^4 / 129) * (Vin - 0.1) =
; = (10^4 / 129) * ((ADC / 204.8) - 0.1) =>
; ((ADC / 204.8) - 0.1) = Cx * 129 / 10^4 =>
; ADC = ((Cx * 129 / 10^4) + 0.1) * 204.8 =>
; ADC = ((Cx / 77.52) + 0.1) * 204.8
;
; By solving the final equation for various Cx (ppm), we find the
; corresponding ADC values for the comparisons.
;
; Also we have 8 levels for the ADC 10 bit value (0 to 1023):
; 0 to 127 (0 leds), 128 to 255 (1 led), 256 to 383 (2 leds),
; 384 to 511 (3 leds), 512 to 639 (4 leds), 640 to 767 (5 leds),
; 768 to 895 (6 leds) and 896 to 1023 (7 leds).
;
; For Cx = 70 ppm we get ADC = 205.41 which rounds down to
; ADC = 205. So for ADC > 205 or ADC >= 206 we need to blink the
; LEDs and print GAS DETECTED to the LCD Screen.
ISR_ADC:
    push r24                    ; push r24 to stack
    push r25                    ; push r25 to stack
    push r26                    ; push r26 to stack
    andi leds, 0x80             ; keep MSB of LEDs (PB7)
    in r24, ADCL                ; read ADC value from register
    in r25, ADCH                ; low first, then high.
    andi r25, 0x03              ; keep 2 LSBs from ADCH (10 bit ADC)
```

```
    cpi r25, 0x00               ; check if r25 = 0 meaning ADC < 256
    brne two_plus               ; if it's not, then lowest level is 2
    cpi r24, 0x80               ; compare r24 with threshold
    brlo controller             ; if r24 < 128, level is 0
    rjmp level_one              ; if r24 >= 128, level is 1
two_plus:
    cpi r25, 0x01               ; check if r25 = 01 meaning ADC < 512
    brne four_plus              ; if it's not, then lowest level is 4
    cpi r24, 0x80               ; compare r24 with threshold
    brlo level_two              ; if r24 < 128, level is 2
    rjmp level_three            ; if r24 >= 128, level is 3
four_plus:
    cpi r25, 0x02               ; check if r25 = 10 meaning ADC < 768
    brne six_plus               ; if it's not, then lowest level is 6
    cpi r24, 0x80               ; compare r24 with threshold
    brlo level_four             ; if r24 < 128, level is 4
    rjmp level_five             ; if r24 >= 128, level is 5
six_plus:
    cpi r24, 0x80               ; compare r24 with threshold
    brlo level_six              ; if r24 < 128, level is 6
level_seven:
    ori leds, 0x7f              ; turn on all 7 LEDs (PB0 to PB6)
    rjmp controller
level_six:
    ori leds, 0x3f              ; turn on 6 LEDs (PB0 to PB5)
    rjmp controller
level_five:
    ori leds, 0x1f              ; turn on 5 LEDs (PB0 to PB4)
    rjmp controller
level_four:
    ori leds, 0x0f              ; turn on 4 LEDs (PB0 to PB3)
    rjmp controller
level_three:
    ori leds, 0x07              ; turn on 3 LEDs (PB0 to PB2)
    rjmp controller
level_two:
    ori leds, 0x03              ; turn on 2 LEDs (PB0 and PB1)
    rjmp controller
level_one:
    ori leds, 0x01              ; turn on 1 LED (PB0)
controller:
    sbrc flags, 0               ; if correct_team = 1 then expert
    rjmp all_on                 ; team will enter, so don't alarm
```

```
    cpi r25, 0x00               ; check if r25 >= 0 (ADC > 256)
    brne gasly                  ; or Cx > 70, if yes go to gasly
    cpi r24, 0xce               ; check if r24 >= 206 (r25 = 0)
    brsh gasly                  ; or Cx > 70, if yes go to gasly
tsunoda:
    sbrs flags, 1               ; if gas_error = 0 then
    rjmp all_on                 ; don't display clear

    andi flags, 0xfd            ; else, set gas_error to 0
    rcall clear                 ; display clear message
    rjmp all_on
gasly:
    sbrc flags, 2               ; if the blinker = 1
    rjmp all_on                 ; turn all LEDs on

    sbrc flags, 1               ; if gas_error = 0 before then
    rjmp only_pb7               ; don't jump to output, first
    ori flags, 0x02             ; set gas_error to 1 and then
    rcall gas                   ; display gas message
only_pb7:
    andi leds, 0x80             ; keep only PB7
    out PORTB, leds             ; output only PB7
    ori flags, 0x04             ; set blinker to 1
    rjmp exit
all_on:
    out PORTB, leds             ; output PB7 + gas LEDs
    andi flags, 0xfb            ; set blinker to 0
exit:
    pop r26                     ; restore r26
    pop r25                     ; restore r25
    pop r24                     ; restore r24
    reti

; ============================================================= ;

ADC_init:
    ldi r24,(1<<REFS0) ; Vref: Vcc
    out ADMUX,r24 ;MUX4:0 = 00000 for A0.
    ;ADC is Enabled (ADEN=1)
    ;ADC Interrupts are Enabled (ADIE=1)
    ;Set Prescaler CK/128 = 62.5Khz (ADPS2:0=111)
    ldi r24,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
    out ADCSRA,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 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 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 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
    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,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:
    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)
```

2η Άσκηση:

```
#define F_CPU 8000000                   // frequency of atmega16
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

unsigned char memory[2], keypad[2], first, second, leds;
int correct_team, blinker;

// 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);
}
```

```
// timer1 interruption service routine
ISR(TIMER1_OVF_vect) {
    ADCSRA |= (1 << ADSC);              // set ADSC bit to 1
    TCNT1 = 64755;                      // reset the 100 ms timer
}

// update LEDs based on new ADC value
void update_leds() {
    leds &= 0x80;                       // keep only PB7
    if (ADC < 128) leds |= 0x00;        // level 0, 0 LEDs
    else if (ADC < 256) leds |= 0x01;   // level 1, 1 LED
    else if (ADC < 384) leds |= 0x03;   // level 2, 2 LEDs
    else if (ADC < 512) leds |= 0x07;   // level 3, 3 LEDs
    else if (ADC < 640) leds |= 0x0f;   // level 4, 4 LEDs
    else if (ADC < 768) leds |= 0x1f;   // level 5, 5 LEDs
    else if (ADC < 896) leds |= 0x3f;   // level 6, 6 LEDs
    else leds |= 0x7f;                  // level 7, 7 LEDs
}

// // ADC interruption service routine
// ISR(ADC_vect) {
//  update_leds();                      // update LEDs

//  if (correct_team) {                 // if correct_team = 1
//      blinker = 1;                    // turn on PB7 + gas LEDs
//  }

//  if (ADC >= 206 && !blinker) {       // if Cx > 70 and blinker = 0
//      PORTB = (leds & 0x80);          // output only PB7
//      blinker = 1;                    // next, turn on gas LEDs
//  }
//  else {                              // if Cx <= 70 or blinker = 1
//      PORTB = leds;                   // output PB7 + gas LEDs
//      blinker = 0;                    // next, turn off gas LEDs
//  }
// }

// ADC interruption service routine
ISR(ADC_vect) {
    update_leds();                      // update LEDs

    // if correct_team = 1 or Cx <= 70 or blinker = 1
    if (correct_team || ADC < 206 || (ADC >= 206 && blinker)) {
        PORTB = leds;                   // output PB7 + gas LEDs
        blinker = 0;                    // set blinker to 0
```

```
    }
    else if (ADC >= 206 && !blinker) {  // if Cx > 70 and blinker = 0
        PORTB = (leds & 0x80);          // output only PB7
        blinker = 1;                    // next, turn on gas LEDs
    }
}

// turn on LEDs for 4 secs
void correct() {
    correct_team = 1;                   // set correct_team to 1

    leds |= 0x80;                       // turn PB7 on
    for(int i = 0; i < 80; i++) {       // 4000ms divided in 80*50ms
        scan_keypad_rising_edge();      // read/ignore keypad (15ms)
        _delay_ms(35);                  // 80*50ms is 80*(15+35)
    }
    leds &= 0x7f;                       // turn PB7 off
    correct_team = 0;                   // set correct_team to 0
}

// blink LEDs every 0.5 sec for 4 secs
void wrong() {
    for(int i = 0; i < 8; i++) {        // loop 8 times (4 on/off)
        if(i % 2) {                     // if i is odd (1, 3, 5, 7)
            leds &= 0x7f;               // turn PB7 off
        }
        else {                          // if i is even (0, 2, 4, 6)
            leds |= 0x80;               // turn PB7 on
        }

        for(int j = 0; j < 10; j++) {   // 500ms divided in 10*50ms
            scan_keypad_rising_edge();  // read/ignore keypad (15 ms)
            _delay_ms(35);              // 10*50ms is 10*(15+35)
        }
    }
}

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

    ADC_init();                         // initialize ADC

    TIMSK = (1 << TOIE1);               // overflow interrupt TIMER1
    TCCR1B = (1 << CS12) | (0 << CS11) | (1 << CS10);// CK/1024
```

```
    TCNT1 = 64755;                          // interrupt every 100 ms
    sei();                                  // enable interrupts

    leds = 0x00;                            // initialize leds
    correct_team = 0;                       // initialize correct_team
    blinker = 0;                            // initialize blinker

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

    while(1) {
        while(1) {
            if(scan_keypad_rising_edge()) {// scan for button pressed
                first = keypad_to_ascii(); // get its ascii and break
                break;
            }
        }

        while(1) {
            if(scan_keypad_rising_edge()) {// scan for button pressed
                second = keypad_to_ascii();// get its ascii and break
                break;
            }
        }

        if(first != '3' || second != '3') {
            wrong();                        // call wrong() if wrong
        }
        else {
            correct();                      // call correct() if correct
        }
    }
    return 0;
}
```