Εργαστήριο Μικρουπολογιστών

4η Άσκηση

Μοίρας Αλέξανδρος

A.M.: el18081

Για να αναπαριστούμε το επίπεδο του μονοξειδίου του άνθρακα με 7 leds ορίζουμε 8 επίπεδα αερίου [0-14ppm), [14ppm, 28ppm) , [28ppm, 42ppm) , [42ppm, 56ppm) , [56ppm, 70ppm) , [70ppm, 84ppm) , [84ppm, 98ppm) , [98ppm, +∞). Τα κόκκινα αποτελούν επίπεδα συναγερμού.

Θα υπολογίσουμε το M του αισθητήρα με βάση τον τύπο που δίνεται στο φύλλο δεδομένων

$$M = Sensitivity\ Code\left(\frac{nA}{ppm}\right) \cdot TIA\ Gain\left(\frac{kV}{A}\right) \cdot 10^{-9} \cdot 10^{3} \Rightarrow$$

$$M = 129 \cdot 100 \cdot 10^{-6} = 129 \cdot 10^{-4}\,\frac{V}{ppm}$$

όπου μας δίνεται Sensitivity Code = 129nA/ppm και από το φύλλο δεδομένων βλέπουμε ότι TIA Gain = 100 kV/A για το CO.

Για τον υπολογισμό της τιμής της τάσης που θα δώσει ο αισθητήρας για τα διάφορα επίπεδα CO θα χρησιμοποιήσουμε τον τύπο:

$$C_x = \frac{1}{M}\left(V_{gas} - V_{gas_0}\right) \Rightarrow V_g = M \cdot C_x + V_{gas_0} \Rightarrow$$

$$V_g = 129 \cdot 10^{-4} \cdot C_x + 0.1\text{V}$$

για τον οποίο δεν χρειαζόμαστε κάποιον συντελεστή αφού βρισκόμαστε σε ονομαστικές συνθήκες θερμοκρασίας (20°C) και υγρασίας (40%) με βάση το φύλλο δεδομένων.

Αφού υπολογίσουμε την τάση που θα δει ο ADC για κάθε ένα από τα επίπεδα CO που ορίσαμε πρέπει να υπολογίσουμε τον αριθμό που θα δούμε εμείς στους καταχωρητές ADCL, ADCH (ψηφιακή έξοδος του ADC). Θα χρησιμοποιήσουμε τον παρακάτω τύπο για $V_{REF} = 5V$.

$$V_{IN} = \frac{ADC}{1024} V_{REF}$$

Χρησιμοποιώντας τους παραπάνω τύπους υπολογίζουμε τον παρακάτω πίνακα για τα επίπεδα που υπολογίσαμε:

C_x	V_{gas}	ADC	HEX
14	0.2806	57	39
28	0.4612	94	5E
42	0.6418	131	83
56	0.8224	168	A8
70	1.003	205	CD
84	1.1836	242	F2
98	1,3642	279	117

Οι HEX τιμές θα χρησιμοποιηθούν στο πρόγραμμα για να ελέγχεται το επίπεδο του αερίου κάθε 0.1s και να ανάβουν τα κατάλληλα leds.

Ζήτημα 5.1:

Ο κώδικας του προγράμματος (main.asm) εμπλουτισμένος με σχόλια:

```
.DSEG
_tmp_: .byte 2

.CSEG

rjmp main ; go to the start of the program
.org 0x10
rjmp ISR_TIMER1_OVF
.org 0x1C
rjmp ADC_INT

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
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)

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,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 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

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 ,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 γιατι τους
push r27 ; αλλάζουμε μέσα στη ρουτίνα
movw r26 ,r24   ; λογικό ‘1’ στις θέσεις του καταχωρητή r26 δηλώνουν
ldi r24 ,'*'    ; τα παρακάτω σύμβολα και αριθμούς
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'
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 προγραμματος 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 προγραμματος 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 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
```

```
print_clear:
ldi r24,0x01 ;clear the lcd
rcall lcd_command_sim
ldi r24, 'C'
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

print_gas_detected:
ldi r24,0x01 ;clear the lcd
rcall lcd_command_sim
ldi r24, 'G'
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

main:
```

```
ldi r24, low(RAMEND) ;initialize stack pointer
out SPL, r24
ldi r24, high(RAMEND)
out SPH, r24
ldi r24, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4) ; 4 MSB of PORTC as
outputs
out DDRC, r24
clr r24
out PORTC, r24 ;disable pull-ups
out DDRA, r24 ; PINA as input
ser r24
out DDRB, r24 ;PORTB as output
out DDRD, r24 ;PORTD as output
sts _tmp_, r24 ; initialize _tmp_
clr r24
ldi r18, 0x00
rcall lcd_init_sim ;initialize the lcd
rcall ADC_init

ldi r24 ,(0<<CS12) | (1<<CS11) | (1<<CS10) ; CK/64=125KHz
out TCCR1B ,r24

ldi r24,0xCF ; initialize TCNT1 for overflow after 0.1s
out TCNT1H ,r24 ; 0.1s = 12.500 cycles 65536-12500=53036=0xCF2C
ldi r24 ,0x2C
out TCNT1L ,r24

ldi r24 ,(1<<TOIE1) ; enable overflow interrupt of TCNT1
out TIMSK ,r24

ldi r18, 0x00 ; initialize led state
ldi r28, 0x00 ; initialize PB7
ldi r19, 0x02 ; previous state (gas(1)-no gas(0)) initialized at 2 so that we get
an initial print
ldi r30, 0x00 ; 1 if a special team has entered. no speacial team has entered

sei ;enable all interrupts

digit1:
rcall scan_keypad_rising_edge_sim ;read the first number
rcall keypad_to_ascii_sim ;convert the button pressed to ascii
cpi r24,0x00 ;check if a button was pressed otherwise read again
breq digit1
mov r20,r24 ;Store the ascii code in r20
subi r20,0x30 ;then convert it to an integer
```

```
digit2:
rcall scan_keypad_rising_edge_sim ;same for the second number
rcall keypad_to_ascii_sim
cpi r24,0x00
breq digit2
mov r21,r24 ;Store the ascii code in r21
subi r21,0x30 ;then convert it to an integer

cpi r20,0x04 ;Check if both digits of the password are correct
brne wrong
cpi r21,0x05
brne wrong

correct: ;if they are
ldi r30 , 0x01 ; a special team has entered
ldi r19, 0x02 ; invalid previous state so that a new message is printed after 4s
ldi r24,0x01 ;clear the lcd
rcall lcd_command_sim
ldi r20, 0x80
or r20, r18
out PORTB,r20 ;PB7 on alongside led state
ldi r24, 'W' ;Print "WELCOME" on the lcd
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
ldi r24, ' '
rcall lcd_data_sim
ldi r20,0xBE  ; each call of scan_keypad_rising_edge_sim takes longer than 19ms
(19ms is the total delay time from delay routines).
            ;we call it 190 times for a total delay ~4s

loop1:
ldi r31, 0x80
or r31, r18
```

```
out PORTB, r31 ;keep updating gas levels
dec r20
rcall scan_keypad_rising_edge_sim
cpi r20,0x00
brne loop1 ;keep reading from keypad and ignoring until 4s have passed
;clr r20
mov r20, r18 ; turn PB7 off
out PORTB,r20 ;turn off the leds
ldi r24,0x01 ;clear the lcd
rcall lcd_command_sim
ldi r30, 0x00 ;special team has exited
rjmp digit1 ;read a new code

wrong: ;if the wrong password was inserted
ldi r20,0x04 ;total of 4 blinks
outerloop:
dec r20
ldi r28, 0x80 ; PB7 on for 0.5s
ldi r21,0x18 ;each call of scan_keypad_rising_edge_sim takes longer than 19ms
(19ms is the total delay time from delay routines).
               ;we call it 24 times for a total delay of ~0.5s

inner1:
sbi PORTB, 7 ; Turn PB7 on for 0.5s
dec r21
rcall scan_keypad_rising_edge_sim ;keep reading digits but ignoring them for 0.5s
cpi r21,0x00
brne inner1

ldi r21,0x18 ;each call of scan_keypad_rising_edge_sim takes longer than 19ms
(19ms is the total delay time from delay routines).
               ;we call it 24 times for a total delay of ~0.5s

inner2:
cbi PORTB, 7 ; Then PB7 back off
dec r21
rcall scan_keypad_rising_edge_sim ;keep reading digits but ignoring them for 0.5s
cpi r21,0x00
brne inner2

cpi r20,0x00
brne outerloop ;total of four blinks
rjmp digit1 ;read a new password

ISR_TIMER1_OVF: ;every 0.1s
```

```
cpi r30, 0x01 ;check if a special team has entered
breq read_adc ;if yes don't blink the lights
ldi r16, 0x10
and r16, r18 ;isolate bit4 of r18 which indicates gas presence
cpi r16, 0x10;
breq gas

no_gas:
cpi r19, 0x00 ; if previous state was clear don't print clear again
breq skip_print_clear
rcall print_clear
skip_print_clear:
ldi r19,0x00 ; previous state = no gas
in r29, PORTB
andi r29, 0x80 ;Consider the state in which PB7 is
or r29, r18
out PORTB, r29 ;Print gas level alongside PB7
ldi r17,0x00 ;reset blink timer for gas alarm
jmp read_adc

gas:
cpi r19, 0x01 ; if previous state was gas_detected don't print gas detected again
breq skip_print_gas_detected
rcall print_gas_detected
skip_print_gas_detected:
ldi r19, 0x01 ; previous state = gas
inc r17
cpi r17,0x06 ;for the first 5*0.1=0.5s led's on then off for another 0.5s
brsh blink_off

blink_on:
;mov r29, r28
;or r29, r18
;out PORTB, r29
in r29, PORTB ;Consider the state in which PB7 is
andi r29, 0x80
or r29, r18
out PORTB, r29 ;Print gas level for 0.5s alongside PB7
;out PORTB, r18
jmp read_adc

blink_off:
;out PORTB, r28
;ldi r16, 0x00
in r29, PORTB ;Consider the state in which PB7 is
```

```
andi r29, 0x80
out PORTB, r29 ;Blink all leds off keeping PB7 as is
;out PORTB, r16
cpi r17,0x0A
brne read_adc
ldi r17,0x00

read_adc:
SBI ADCSRA, 6 ;read from ADC
sei ; re-enable interrupts
ret

ADC_INT:
push r16 ;Store registers r16, r17 that will be used
push r17
in r16, ADCL ;retrieve the value read from ADC
in r17, ADCH
andi r17,0x03

ldi r18, 0b01111111
cpi r17, 0x01 ;if value read is higher than 90ppm
brsh go_back

ldi r18, 0b00111111
cpi r16, 0xF2 ;if value read is higher than 84ppm
brsh go_back

ldi r18, 0b00011111
cpi r16, 0xCD ;if value read is higher than 70ppm
brsh go_back

ldi r18, 0b00001111
cpi r16, 0xA8 ;if value read is higher than 56ppm
brsh go_back

ldi r18, 0b00000111
cpi r16, 0x83 ;if value read is higher than 42ppm
brsh go_back

ldi r18, 0b00000011
cpi r16, 0x5E ;if value read is higher than 28ppm
brsh go_back

ldi r18, 0b00000001
cpi r16, 0x39 ;if value read is higher than 14ppm
```

```
brsh go_back

ldi r18, 0x00 ;if value read is lower than 14ppm
go_back:
ldi r24,0xCF ; reset TCNT1
out TCNT1H ,r24 ; for overflow after 0.1s
ldi r24 ,0x2C
out TCNT1L ,r24
sei ; re-enable interrupts
pop r17
pop r16
ret
```

Ζήτημα 5.2:

Όπως και στην προηγούμενη άσκηση χρησιμοποιήθηκαν δύο αρχεία στο ίδιο Project το Assemblyfunctions.S και το main.c με το Microchip Studio να αναλαμβάνει το compilation. Εδώ επιλέξαμε τη μέθοδο polling αντί των διακοπών για να διαβάσουμε τον ADC. Αναγνωρίζουμε ότι είναι κακή ιδέα το polling μέσα στη ρουτίνα εξυπηρέτησης της διακοπής χρονιστή και θα ήταν προτιμότερο να χρησιμοποιούνταν διακοπές ωστόσο χρησιμοποιήσαμε polling απλώς για λόγους δοκιμής και επειδή δεν επηρέασε τη λειτουργικότητα του προγράμματος (πιθανότατα επειδή ο χρόνος για το polling είναι πολύ μικρός) το κρατήσαμε. Η ίδια λειτουργία θα μπορούσε να υλοποιηθεί και με διακοπές ακριβώς αντίστοιχα με το assembly πρόγραμμα του προηγούμενου ερωτήματος. Ο κώδικας των δύο αρχείων εμπλουτισμένος με σχόλια:

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

    .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 ; 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,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 clear_lcd
    clear_lcd:
    push r24
    ldi r24,0x01 ;clear the lcd
    rcall lcd_command_sim
    pop r24
    ret

    .global print_gas_detected
    print_gas_detected:
    push r24
    ldi r24, 'G'
    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
    pop r24
    ret

    .global print_clear
    print_clear:
    push r24
    ldi r24, 'C'
    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
    pop r24
    ret

    .global print_welcome
    print_welcome:
    push r24
    ldi r24, 'W' ;Print "WELCOME" on the lcd
    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
    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)

    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

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

    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
```

main.c:

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

int led_state=0; //which leds are on
int previous_state;
int led7; // PB7 state
int blink_timer=0;
int value_read; //variable to store value read from adc
int special_team; //to check whether a special team has entered or not
int scan_keypad_rising_edge_sim(); //declaration of assembly functions. 16bit
return value must be store in r25:r24
//void wait_msec(int msecs);
//void wait_usec(int usecs);
void initialize_variable();
void lcd_init_sim();
void clear_lcd();
```

```
void print_gas_detected();
void print_clear();
void print_welcome();

char keypad_to_ascii(int btn) { //returns the ascii character that corresponds to
the first bit 1 found
    if ((btn & 0x0001)==0x0001) {
        return '*';
    }
    if ((btn & 0x0002)==0x0002) {
        return '0';
    }
    if ((btn & 0x0004)==0x0004) {
        return '#';
    }
    if ((btn & 0x0008)==0x0008) {
        return 'D';
    }
    if ((btn & 0x0010)==0x0010) {
        return '7';
    }
    if ((btn & 0x0020)==0x0020) {
        return '8';
    }
    if ((btn & 0x0040)==0x0040) {
        return '9';
    }
    if ((btn & 0x0080)==0x0080) {
        return 'C';
    }
    if ((btn & 0x0100)==0x0100) {
        return '4';
    }
    if ((btn & 0x0200)==0x0200) {
        return '5';
    }
    if ((btn & 0x0400)==0x0400) {
        return '6';
    }
    if ((btn & 0x0800)==0x0800) {
        return 'B';
    }
    if ((btn & 0x1000)==0x1000) {
        return '1';
    }
```

```
    if ((btn & 0x2000)==0x2000) {
        return '2';
    }
    if ((btn & 0x4000)==0x4000) {
        return '3';
    }
    if ((btn & 0x8000)==0x8000) {
        return 'A';
    }
    return 0;
}

ISR(TIMER1_OVF_vect) {
    if(special_team==0) { // if a special team has entered don't blink the leds
        if((led_state & 0x10) == 0x10) { //gas detected
            if(previous_state!=1) { //if previous state was gas detected don't
print gas detected again
                clear_lcd();
                print_gas_detected();
                previous_state=1;
            }
            if(blink_timer<5) { //blink gas level leds for 0.5s
                led7=PORTB & 0x80;
                PORTB = led7 | led_state; //without interfering with PB7
                blink_timer++;
            }
            else { //then blink them off for another 0.5s
                led7 = PORTB & 0x80;
                PORTB = led7; //without interfering with PB7
                blink_timer++;
                if(blink_timer==10) { //reset blink timer for the next blink
                    blink_timer=0;
                }
            }
        }
        else { //clear
            if(previous_state!=0) { //if previous state was clear don't print
clear again
                clear_lcd();
                print_clear();
                previous_state=0;
            }
            led7 = PORTB & 0x80;
            PORTB = led7 | led_state; //print gas level without interfering with
PB7
```

```
            blink_timer=0; //reset blink timer for gas alarm
        }
    }
    ADCSRA|=(1<<ADSC); //start conversion

    while(ADCSRA & (1<<ADSC)); //wait until conversion is done
    value_read = ADCW; //store value read from ADCW in value_read
    if(value_read>=0x117){ //if value read is higher than 90ppm
        led_state=0b01111111;
    }
    else if(value_read>=0xF2){ //if value read is higher than 84ppm
        led_state=0b00111111;
    }
    else if(value_read>=0xCD){ //if value read is higher than 70ppm
        led_state=0b00011111;
    }
    else if(value_read>=0xA8){ //if value read is higher than 56ppm
        led_state=0b00001111;
    }
    else if(value_read>=0x83){ //if value read is higher than 42ppm
        led_state=0b00000111;
    }
    else if(value_read>=0x5E){ //if value read is higher than 28ppm
        led_state=0b00000011;
    }
    else if(value_read>=0x39){ //if value read is higher than 14ppm
        led_state=0b00000001;
    }
    else{ //if value read is lower than 14ppm
        led_state=0;
    }


    TCNT1H=0xCF;
    TCNT1L=0x2C; //reset TCNT1 for overflow after 0.1s
}

int main(void)
{
    DDRC = 0xF0; //4 MSBs of PORTC as outputs 4LSBs as inputs
    PORTC = 0x00; //disable pull-up resistors
    DDRB = 0xFF; //PORTB as output
    DDRA = 0x00; //PORTA as input
    DDRD = 0xFF; //PORTD as output
```

```
    ADMUX = 0x40; //Vref: Vcc
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //no interrupts

    TCCR1B = 0x03; //CK/64
    TCNT1H = 0xCF;
    TCNT1L = 0x2C; //initialize TCNT1 for overflow after 0.1s
    TIMSK = 0x04; //enable overflow interrupt for TCNT1

    int btn;
    int digit1, digit2;
    //scan_keypad_rising_edge_sim(); //just to instantiate _tmp_ assembly
variable
    initialize_variable(); //initialize _tmp_
    lcd_init_sim(); //initialize lcd
    previous_state=2; //invalid previous state so that we get a print at the
start
    special_team=0; //no special team has entered at the start
    sei(); //enable interrupts
    while (1)
    {
        btn=0;
        while(btn==0){
            btn=scan_keypad_rising_edge_sim(); //scan the keypad until a key is
pressed
        }
        digit1=keypad_to_ascii(btn)-48; //translate the first key pressed to an
ascii character then to an integer
        btn=0;
        while(btn==0){
            btn=scan_keypad_rising_edge_sim(); //wait until the second key is
pressed
        }
        digit2=keypad_to_ascii(btn)-48; //translate the second key pressed to an
ascii character then to an integer

        if((digit1==4)&&(digit2==5)) { //if password is correct
            //cli(); //disable interrupts
            special_team=1;
            PORTB=0x80|led_state; //light PB7 and gas indicator constantly
            clear_lcd();
            print_welcome();
            for(int i=0; i<190; ++i) {          //we need to keep scanning the
keypad. Each scan takes longer than 19ms (19ms is the total delay time from delay
routines)
```

```
                scan_keypad_rising_edge_sim();  //so we call it 190 times so that
the leds stay on for ~4secs
                PORTB=0x80|led_state; //keep updating the gas level indicator
            }
            previous_state=2; //invalid previous state so that we get a print
            PORTB=led_state; //PB7 off
            special_team=0;
            //sei(); //re-enable interrupts
        }
        else { //if password is incorrect
            for(int i=0; i<4; ++i) { //total of four blinks
                PORTB|=(1<<PB7); //light PB7 without interfering with the rest
leds for 0.5s
                for(int i=0; i<24; ++i) { //we need to keep scanning the keypad.
Each scan takes longer than 19ms (19ms is the total delay time from delay
routines)
                    scan_keypad_rising_edge_sim(); //so we call it 24 times for a
total of ~0.5secs
                }
                PORTB&=~(1<<PB7); //turn off PB7 without interfering with the
rest leds for 0.5s
                for(int i=0; i<24; ++i) {
                    scan_keypad_rising_edge_sim();
                }
            }
        } //read two numbers from the keypad again
    }
}
```