4^{η} Εργαστηριακή Άσκηση στο Εργαστήριο Μικροεπεξεργαστών

Ομάδα Β 3

Αλέξανδρος Κυριακάκης (03112163),
Ιωάννης Αλεξόπουλος (03117001)

Δεκέμβρης 2020

1^{η} Άσκηση

Όσον αφορά τους υπολογισμούς που έγινα στον Assembly Code παραπέμπουμε στον C Code Function: calc_cx (line 198).

```
1  .include "m16def.inc"
2  .def flag = r16
3  .def leds = r17
4  .def entered_correct = r18
5  .DSEG
6  _tmp_: .BYTE 2
7  .CSEG
8  .org 0x0
9  jmp reset
10 .org 0x10
11 rjmp ISR_TIMER1_OVF
12 .org 0x1c
13 rjmp ADC_ISR
14 reset:
15  ldi r24, low(RAMEND) ; initialize stack pointer
16  out SPL, r24
17  ldi r24, high(RAMEND)
18  out SPH, r24
19  ldi r26, low(_tmp_)  ; initialize _tmp_ variable
20  ldi r27, high(_tmp_)
21  clr r24
22  st X+,r24
23  st X,r24
24  clr flag    ;initialize flag
25  clr leds    ;initialize leds
26  ser r24     ; initialize PORTB for output
```

```
 out DDRB, r24
 out DDRD, r24           ; initialize PORTD for lcd display as output
 clr r24
 ldi r24, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4) ; initialize PORTC
  ↪ for keypad
  out DDRC, r24
 rcall lcd_init_sim  ;initialize lcd display
;--------------Νέος
  ↪ Κώδικας-------------------------------------------------------------------------
 rcall ADC_init   ;initialize adc converter
 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

 sei      ;enable interrupts


first_digit:
 ldi r24, 0xf0 ; pernaw asso se ola ta pliktra
 rcall scan_keypad_rising_edge_sim ; elegxw tis eksodous
 clr r22 ; arxikopoiw sto 0
 or r22, r24 ; ta grafw ola ekei gia na dv an exw allages
 or r22, r25 ;
 cpi r22, 0
 breq first_digit
 ; password  = 03 -> r25 = 0 + r24 = 2 and then r24 = 0 r25 = 0b1000000
 cpi r25, 0
 brne wrong_first
 cpi r24, 2
 brne wrong_first
 rjmp second_digit
wrong_first:
 ldi r21, 1      ; flag that indicates first digit was incorrect

second_digit:
 ldi r24, 0xf0 ; pernaw asso se ola ta pliktra
 rcall scan_keypad_rising_edge_sim ; elegxw tis eksodous
 clr r22 ; arxikopoiw sto 0
 or r22, r24 ; ta grafw ola ekei gia na dv an exw allages
 or r22, r25
 cpi r22,0
 breq second_digit
 ;r24 = 0 r25 = 0b1000000
```

```
 cpi r21,1
 breq wrong_passwd
 cpi r24,0
 brne wrong_passwd
 cpi r25,0x40
 brne wrong_passwd


right_passwd:
 rcall scan_keypad_rising_edge_sim
 ldi entered_correct,0x01   ;set 3rd 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
 sbrc flag, 0
 rcall display_gas
 ldi entered_correct,0x00
 ; WARNING: ori flag,0x10   ; set 5th bit of flag to indicate that
  ↪  correct_password process is over
 rjmp first_digit  ; repeat

wrong_passwd:
 rcall scan_keypad_rising_edge_sim
 ldi r21, 0              ;reset first digit wrong flag
 ldi r23, 0x04   ;counter for blinking 4 times
leds_loop:
 rcall leds_on   ; 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
```

```
166
167 display_welcome:
168  rcall lcd_init_sim
169  clr r24
170  ldi r24, 'W'  ; welcome message
171  rcall lcd_data_sim
172  ldi r24, 'E'
173  rcall lcd_data_sim
174  ldi r24, 'L'
175  rcall lcd_data_sim
176  ldi r24, 'C'
177  rcall lcd_data_sim
178  ldi r24, 'O'
179  rcall lcd_data_sim
180  ldi r24, 'M'
181  rcall lcd_data_sim
182  ldi r24, 'E'
183  rcall lcd_data_sim
184  ret
185
186 leds_on:
187  ori leds,0x80   ;set 8th bit of leds
188  out PORTB, leds
189  ret
190
191 leds_off:
192  andi leds,0x7F   ;clear 8th bit of leds
193  out PORTB, leds
194  ret
195
196 scan_row_sim:
197  out PORTC, r25 ; η αντίστοιχη γραμμή τίθεται στο λογικό '1'
198  push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
199  push r25 ; λειτουργία του προγραμματος απομακρυσμένης
200  ldi r24,low(500) ; πρόσβασης
201  ldi r25,high(500)
202  rcall wait_usec
203  pop r25
204  pop r24 ; τέλος τμήμα κώδικα
205  nop
206  nop ; καθυστέρηση για να προλάβει να γίνει η αλλαγή κατάστασης
207  in r24, PINC ; επιστρέφουν οι θέσεις (στήλες) των διακοπτών που είναι πιεσμένοι
208  andi r24 ,0x0f ; απομονώνονται τα 4 LSB όπου τα '1' δείχνουν που είναι πατημένοι
209  ret ; οι διακόπτες
210
211  scan_keypad_sim:
212  push r26 ; αποθήκευσε τους καταχωρητές r27:r26 γιατι τους
```

```
213   push r27 ; αλλάζουμε μέσα στην ρουτίνα
214   ldi r25 , 0x10 ; έλεγξε την πρώτη γραμμή του πληκτρολογίου (PC4: 1 2 3 A)
215   rcall scan_row_sim
216   swap r24 ; αποθήκευσε το αποτέλεσμα
217   mov r27, r24 ; στα 4 msb του r27
218   ldi r25 ,0x20 ; έλεγξε τη δεύτερη γραμμή του πληκτρολογίου (PC5: 4 5 6 B)
219   rcall scan_row_sim
220   add r27, r24 ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r27
221   ldi r25 , 0x40 ; έλεγξε την τρίτη γραμμή του πληκτρολογίου (PC6: 7 8 9 C)
222   rcall scan_row_sim
223   swap r24 ; αποθήκευσε το αποτέλεσμα
224   mov r26, r24 ; στα 4 msb του r26
225   ldi r25 ,0x80 ; έλεγξε την τέταρτη γραμμή του πληκτρολογίου (PC7: * 0 # D)
226   rcall scan_row_sim
227   add r26, r24 ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r26
228   movw r24, r26 ; μετέφερε το αποτέλεσμα στους καταχωρητές r25:r24
229   clr r26 ; προστέθηκε για την απομακρυσμένη πρόσβαση
230   out PORTC,r26 ; προστέθηκε για την απομακρυσμένη πρόσβαση
231   pop r27 ; επανάφερε τους καταχωρητές r27:r26
232   pop r26
233   ret
234
235   scan_keypad_rising_edge_sim:
236   push r22 ; αποθήκευσε τους καταχωρητές r23:r22 και τους
237   push r23 ; r26:r27 γιατι τους αλλάζουμε μέσα στην ρουτίνα
238   push r26
239   push r27
240   rcall scan_keypad_sim ; έλεγξε το πληκτρολόγιο για πιεσμένους διακόπτες
241   push r24 ; και αποθήκευσε το αποτέλεσμα
242   push r25
243   ldi r24 ,15 ; καθυστέρησε 15 ms (τυπικές τιμές 10-20 msec που καθορίζεται από τον
244   ldi r25 ,0 ; κατασκευαστή του πληκτρολογίου ( χρονοδιάρκεια σπινθηρισμών)
245   rcall wait_msec
246   rcall scan_keypad_sim ; έλεγξε το πληκτρολόγιο ξανά και απόρριψε
247   pop r23 ; όσα πλήκτρα εμφανίζουν σπινθηρισμό
248   pop r22
249   and r24 ,r22
250   and r25 ,r23
251   ldi r26 ,low(_tmp_) ; φόρτωσε την κατάσταση των διακοπτών στην
252   ldi r27 ,high(_tmp_) ; προηγούμενη κλήση της ρουτίνας στους r27:r26
253   ld r23 ,X+
254   ld r22 ,X
255   st X ,r24 ; αποθήκευσε στη RAM τη νέα κατάσταση
256   st -X ,r25 ; των διακοπτών
257   com r23
258   com r22 ; βρες τους διακόπτες που έχουν μόλις πατηθεί
259   and r24 ,r22
260   and r25 ,r23
```

```
261  pop r27 ; επανάφερε τους καταχωρητές r27:r26
262  pop r26 ; και r23:r22
263  pop r23
264  pop r22
265  ret
266
267  keypad_to_ascii_sim:
268  push r26 ; αποθήκευσε τους καταχωρητές r27:r26 γιατι τους
269  push r27 ; αλλάζουμε μέσα στη ρουτίνα
270  movw r26 ,r24 ; λογικό '1' στις θέσεις του καταχωρητή r26 δηλώνουν
271  ; τα παρακάτω σύμβολα και αριθμούς
272  ldi r24 ,'*'
273  ; r26
274  ;C 9 8 7 D # 0 *
275  sbrc r26 ,0
276  rjmp return_ascii
277  ldi r24 ,'0'
278  sbrc r26 ,1
279  rjmp return_ascii
280  ldi r24 ,'#'
281  sbrc r26 ,2
282  rjmp return_ascii
283  ldi r24 ,'D'
284  sbrc r26 ,3 ; αν δεν είναι '1'παρακάμπτει την ret, αλλιώς (αν είναι '1')
285  rjmp return_ascii ; επιστρέφει με τον καταχωρητή r24 την ASCII τιμή του D.
286  ldi r24 ,'7'
287  sbrc r26 ,4
288  rjmp return_ascii
289  ldi r24 ,'8'
290  sbrc r26 ,5
291  rjmp return_ascii
292  ldi r24 ,'9'
293  sbrc r26 ,6
294  rjmp return_ascii ;
295  ldi r24 ,'C'
296  sbrc r26 ,7
297  rjmp return_ascii
298  ldi r24 ,'4' ; λογικό '1' στις θέσεις του καταχωρητή r27 δηλώνουν
299  sbrc r27 ,0 ; τα παρακάτω σύμβολα και αριθμούς
300  rjmp return_ascii
301  ldi r24 ,'5'
302  ;r27
303  ;A 3 2 1 B 6 5 4
304  sbrc r27 ,1
305  rjmp return_ascii
306  ldi r24 ,'6'
307  sbrc r27 ,2
```

```
308   rjmp return_ascii
309   ldi r24 ,'B'
310   sbrc r27 ,3
311   rjmp return_ascii
312   ldi r24 ,'1'
313   sbrc r27 ,4
314   rjmp return_ascii ;
315   ldi r24 ,'2'
316   sbrc r27 ,5
317   rjmp return_ascii
318   ldi r24 ,'3'
319   sbrc r27 ,6
320   rjmp return_ascii
321   ldi r24 ,'A'
322   sbrc r27 ,7
323   rjmp return_ascii
324   clr r24
325   rjmp return_ascii
326   return_ascii:
327   pop r27 ; επανάφερε τους καταχωρητές r27:r26
328   pop r26
329   ret
330
331   write_2_nibbles_sim:
332   push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
333   push r25 ; λειτουργία του προγραμματος απομακρυσμένης
334   ldi r24 ,low(6000) ; πρόσβασης
335   ldi r25 ,high(6000)
336   rcall wait_usec
337   pop r25
338   pop r24 ; τέλος τμήμα κώδικα
339   push r24 ; στέλνει τα 4 MSB
340   in r25, PIND ; διαβάζονται τα 4 LSB και τα ξαναστέλνουμε
341   andi r25, 0x0f ; για να μην χαλάσουμε την όποια προηγούμενη κατάσταση
342   andi r24, 0xf0 ; απομονώνονται τα 4 MSB και
343   add r24, r25 ; συνδυάζονται με τα προϋπάρχοντα 4 LSB
344   out PORTD, r24 ; και δίνονται στην έξοδο
345   sbi PORTD, PD3 ; δημιουργείται παλμός Enable στον ακροδέκτη PD3
346   cbi PORTD, PD3 ; PD3=1 και μετά PD3=0
347   push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
348   push r25 ; λειτουργία του προγραμματος απομακρυσμένης
349   ldi r24 ,low(6000) ; πρόσβασης
350   ldi r25 ,high(6000)
351   rcall wait_usec
352   pop r25
353   pop r24 ; τέλος τμήμα κώδικα
354   pop r24 ; στέλνει τα 4 LSB. Ανακτάται το byte.
355   swap r24 ; εναλλάσσονται τα 4 MSB με τα 4 LSB
```

```
356  andi r24 ,0xf0 ; που με την σειρά τους αποστέλλονται
357  add r24, r25
358  out PORTD, r24
359  sbi PORTD, PD3 ; Νέος παλμός Enable
360  cbi PORTD, PD3
361  ret
362
363  lcd_data_sim:
364  push r24
365  push r25
366  sbi PORTD,PD2
367  rcall write_2_nibbles_sim
368  ldi r24,43
369  ldi r25,0
370  rcall wait_usec
371  pop r25
372  pop r24
373  ret
374
375  lcd_command_sim:
376  push r24 ; αποθήκευσε τους καταχωρητές r25:r24 γιατί τους
377  push r25 ; αλλάζουμε μέσα στη ρουτίνα
378  cbi PORTD, PD2 ; επιλογή του καταχωρητή εντολών (PD2=0)
379  rcall write_2_nibbles_sim ; αποστολή της εντολής και αναμονή 39μsec
380  ldi r24, 39 ; για την ολοκλήρωση της εκτέλεσης της από τον ελεγκτή της lcd.
381  ldi r25, 0 ; ΣΗΜ.: υπάρχουν δύο εντολές, οι clear display και return home,
382  rcall wait_usec ; που απαιτούν σημαντικά μεγαλύτερο χρονικό διάστημα.
383  pop r25 ; επανάφερε τους καταχωρητές r25:r24
384  pop r24
385  ret
386
387  lcd_init_sim:
388  push r24 ; αποθήκευσε τους καταχωρητές r25:r24 γιατί τους
389  push r25 ; αλλάζουμε μέσα στη ρουτίνα
390  ldi r24, 40 ; Όταν ο ελεγκτής της lcd τροφοδοτείται με
391  ldi r25, 0 ; ρεύμα εκτελεί την δική του αρχικοποίηση.
392  rcall wait_msec ; Αναμονή 40 msec μέχρι αυτή να ολοκληρωθεί.
393  ldi r24, 0x30 ; εντολή μετάβασης σε 8 bit mode
394  out PORTD, r24 ; επειδή δεν μπορούμε να είμαστε βέβαιοι
395  sbi PORTD, PD3 ; για τη διαμόρφωση εισόδου του ελεγκτή
396  cbi PORTD, PD3 ; της οθόνης, η εντολή αποστέλλεται δύο φορές
397  ldi r24, 39
398  ldi r25, 0 ; εάν ο ελεγκτής της οθόνης βρίσκεται σε 8-bit mode
399  rcall wait_usec ; δεν θα συμβεί τίποτα, αλλά αν ο ελεγκτής έχει διαμόρφωση
400  ; εισόδου 4 bit θα μεταβεί σε διαμόρφωση 8 bit
401  push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
402  push r25 ; λειτουργία του προγραμματος απομακρυσμένης
403  ldi r24,low(1000) ; πρόσβασης
```

```
404   ldi r25,high(1000)
405   rcall wait_usec
406   pop r25
407   pop r24 ; τέλος τμήμα κώδικα
408   ldi r24, 0x30
409   out PORTD, r24
410   sbi PORTD, PD3
411   cbi PORTD, PD3
412   ldi r24,39
413   ldi r25,0
414   rcall wait_usec
415   push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
416   push r25 ; λειτουργία του προγραμματος απομακρυσμένης
417   ldi r24 ,low(1000) ; πρόσβασης
418   ldi r25 ,high(1000)
419   rcall wait_usec
420   pop r25
421   pop r24 ; τέλος τμήμα κώδικα
422   ldi r24,0x20 ; αλλαγή σε 4-bit mode
423   out PORTD, r24
424   sbi PORTD, PD3
425   cbi PORTD, PD3
426   ldi r24,39
427   ldi r25,0
428   rcall wait_usec
429   push r24 ; τμήμα κώδικα που προστίθεται για τη σωστή
430   push r25 ; λειτουργία του προγραμματος απομακρυσμένης
431   ldi r24 ,low(1000) ; πρόσβασης
432   ldi r25 ,high(1000)
433   rcall wait_usec
434   pop r25
435   pop r24 ; τέλος τμήμα κώδικα
436   ldi r24,0x28 ; επιλογή χαρακτήρων μεγέθους 5x8 κουκίδων
437   rcall lcd_command_sim ; και εμφάνιση δύο γραμμών στην οθόνη
438   ldi r24,0x0c ; ενεργοποίηση της οθόνης, απόκρυψη του κέρσορα
439   rcall lcd_command_sim
440   ldi r24,0x01 ; καθαρισμός της οθόνης
441   rcall lcd_command_sim
442   ldi r24, low(1530)
443   ldi r25, high(1530)
444   rcall wait_usec
445   ldi r24 ,0x06 ; ενεργοποίηση αυτόματης αύξησης κατά 1 της διεύθυνσης
446   rcall lcd_command_sim ; που είναι αποθηκευμένη στον μετρητή διευθύνσεων και
447   ; απενεργοποίηση της ολίσθησης ολόκληρης της οθόνης
448   pop r25 ; επανάφερε τους καταχωρητές r25:r24
449   pop r24
450   ret
451
```

```
452
453  wait_msec:
454  push r24     ; 2 κύκλοι (0.250 μsec)
455  push r25     ; 2 κύκλοι
456  ldi r24 , low(998)  ; φόρτωσε τον καταχ. r25:r24 με 998 (1 κύκλος - 0.125 μsec)
457  ldi r25 , high(998)  ; 1 κύκλος (0.125 μsec)
458  rcall wait_usec   ; 3 κύκλοι (0.375 μsec), προκαλεί συνολικά καθυστέρηση 998.375
     ↪ μsec
459  pop r25      ; 2 κύκλοι (0.250 μsec)
460  pop r24      ; 2 κύκλοι
461  sbiw r24 , 1   ; 2 κύκλοι
462  brne wait_msec   ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
463  ret       ; 4 κύκλοι (0.500 μsec)
464
465  wait_usec:
466  sbiw r24 ,1   ; 2 κύκλοι (0.250 μsec)
467  nop      ; 1 κύκλος (0.125 μsec)
468  nop      ; 1 κύκλος (0.125 μsec)
469  nop      ; 1 κύκλος (0.125 μsec)
470  nop      ; 1 κύκλος (0.125 μsec)
471  brne wait_usec  ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
472  ret      ; 4 κύκλοι (0.500 μsec)
473
474
475 ;--------------Νέος
     ↪ Κώδικας-------------------------------------------------------------------------
476 ADC_init:
477  ldi r24,(1<<REFS0) ; Vref: Vcc
478  out ADMUX,r24 ;MUX4:0 = 00000 for A0.
479  ;ADC is Enabled (ADEN=1)
480  ;ADC Interrupts are Enabled (ADIE=1)
481  ;Set Prescaler CK/128 = 62.5Khz (ADPS2:0=111)
482  ldi r24,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
483  out ADCSRA,r24
484  reti
485
486 ISR_TIMER1_OVF:
487  push r24
488  in r24,ADCSRA  ;begin adc conversion
489  ori r24,(1<<ADSC)
490  out ADCSRA, r24
491  ldi r24,0xfc  ;reset timer
492  out TCNT1H ,r24
493  ldi r24 ,0xf3
494  out TCNT1L ,r24
495  pop r24
496  reti
497
```

```
; We have calculated the ADC value according to specific values of CO
  ↪ concentration
; formula : ADC = ((Cx/77.5) + 1) * 204.8 with rounding
ADC_ISR:
 push r24
 push r25
 push r26
 clr r26
 in r24, ADCL  ;the result of the ADC is stored in 25:24
 in r25, ADCH
 andi r25,0x03  ;keep the 2 lsbs of r25 since result is 8 bits
 cpi r25, 0x02
 brsh seven_leds
 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?
 rjmp four_leds
five_plus:
 cpi r24, 0x29   ; 80 < ppm < 105?
 brlo five_leds
 cpi r24, 0x86   ; 105 < ppm < 140?
 brlo six_leds
seven_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
```

```
545 fixed_leds:
546  andi leds,0x80   ;keep the 8th bit
547  or leds,r26   ;add the level of CO concentration
548  out PORTB,leds   ;display it in PORTB
549  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)
550  jmp exit   ;even if CO was above level we think it is good practice to keep it
     ↪ that way
551  cpi r26,0x08   ;else, check if level below 4th bit indicating < 70 ppm
552  brlo clear   ;if yes it's clear
553  cpi flag,0x01   ;else check if the previous check indicated danger and the led
     ↪ was on
554  breq on_off
555  cpi flag,0x03   ;check if the previous check indicated danger and the led was off
556  breq on_ex
557  ldi flag,0x01   ;else if the previous check was ok, set 1st bit of flag
558  ldi r24,0x01
559  rcall lcd_command_sim  ;clean display
560  ldi r24, low(1530)  ;clean display delay
561  ldi r25, high(1530)
562  rcall wait_usec
563  rcall display_gas  ;display gas message
564  jmp exit
565 on_off:
566  ori flag,0x02   ;set 2nd bit of flag so on next interrupt it is turned on
567  andi leds,0x80   ;keep only 8th bit of leds
568  out PORTB, leds  ;in order to turn off led of level of gas
569  jmp exit
570 on_ex:
571  andi flag,0xFD   ;clear 2nd bit of flag to display concentration level on next
     ↪ interrupt
572  jmp exit
573 clear:
574  cpi flag,0x00   ;check if previous check was also clear so as not to display
     ↪ clear message
575  breq exit
576  sbrs flag, 0   ;check if previous check indicated dangerous level of gas
577  jmp exit
578  ldi flag,0x00   ;clear flag to show safe level of gas
579  ldi r24,0x01
580  rcall lcd_command_sim  ;clean display
581  ldi r24, low(1530)  ;clean display delay
582  ldi r25, high(1530)
583  rcall wait_usec
584  rcall display_clear  ;display clear message
585 exit:
586  pop r26
587  pop r25
```

```
 pop r24
 reti
```

2^{η} Άσκηση

```
#define F_CPU 8000000   // FREQUENCY OF ATMEGA16

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


// MACRO => SET LEDS ON FOR 4sec
#define SUCCESS do {\
 HUGE_flag = 1;\
 flag_MSB = 1;\
 PORTB = 0x80 | leds;\
 _delay_ms(4000);\
 PORTB = 0x00 | leds;\
 flag_MSB = 0;\
 HUGE_flag = 0;\
} while (0)

// MACRO => BLINK LEDS ON-OFF FOR 500ms EACH STATE x 4 TIMES = 4 SECONDS
#define BLINK_FAIL do {\
 flag_MSB = 1;\
 PORTB = 0x80 | leds;\
 _delay_ms(500);\
 flag_MSB = 0;\
 PORTB = 0x00 | leds;\
 _delay_ms(500);\
 flag_MSB = 1;\
 PORTB = 0x80 | leds;\
 _delay_ms(500);\
 flag_MSB = 0;\
 PORTB = 0x00 | leds;\
 _delay_ms(500);\
 flag_MSB = 1;\
 PORTB = 0x80 | leds;\
 _delay_ms(500);\
 flag_MSB = 0;\
 PORTB = 0x00 | leds;\
 _delay_ms(500);\
 flag_MSB = 1;\
```

```
 PORTB = 0x80 | leds;\
 _delay_ms(500);\
 flag_MSB = 0;\
 PORTB = 0x00 | leds;\
 _delay_ms(500);\
} while (0)

// GLOBAL VARIABLES
unsigned char mem[2],
key_reg[2],
first,second,    // x: 1ST KEY, y: 2ND KEY
flag, flag_MSB = 0x00, leds = 0x00, Blink_flag = 0, HUGE_flag = 0; // USED TO
↪  CHECK IF FIRST KEY WAS CORRECT
int counter = 0;

// SCAN ROW(x)
unsigned char scan_row(int i) {   // i = 1,2,3,4
 unsigned char a = ( 1 << 3 ); // SKIP 3 LSB
 a = (a << i);   // SELECT ROW ACCORDING TO FUNCTION INPUT i
 PORTC = a;    // WE SELECT ROW BY SETTING CORRESPONDING BIT TO 1
 _delay_us(500);   // DELAY FOR REMOTE USAGE
 return PINC & 0x0F;  // WE READ THE 4 LSB, '1' INDICATES SWITCH PUSHED
}

/* FUNCTION TO SWAP LO WITH HO BITS */
unsigned char swap(unsigned char x) {
 return ((x & 0x0F) << 4 | (x & 0xF0) >> 4);
}

/* SCAN ROWS(1..4) *DIFFERENT ORDER FROM EXERSISE DOCUMENT*
* FIRST ROW: PC4->PC0: 1,  PC4->PC1: 2,  PC4->PC2: 3, PC4->PC3: A
* SECOND ROW: PC5->PC0: 4,  PC5->PC1: 5,  PC5->PC2: 6, PC5->PC3: B
* THIRD ROW: PC6->PC0: 7,  PC6->PC1: 8,  PC6->PC2: 9, PC6->PC3: C
* FOURTH ROW: PC7->PC0: *,  PC7->PC1: 0,  PC7->PC2: #, PC7->PC3: D
*/
void scan_keypad() {
 unsigned char i;

 // check row 1, 0b0001-ROW CORRESPONDING TO PC4
 i = scan_row(1);
 key_reg[1] = swap(i); //key_reg[1] = first_row(4 MSB)-0000

 // check row 2, 0b0010-ROW CORRESPONDING TO PC5
 i = scan_row(2);
 key_reg[1] += i;  //key_reg[1] = first_row(4 MSB)-second_row(4 LSB)

 // check row 3, 0b0100-ROW CORRESPONDING TO PC6
 i = scan_row(3);
```

```
 key_reg[0] = swap(i); //key_reg[0] = third_row(4 MSB) -0000

 // check row 4, 0b1000-ROW CORRESPONDING TO PC7
 i = scan_row(4);
 key_reg[0] += i;  //key_reg[0] = third_row(4 MSB)-fourth_row(4 LSB)
 PORTC = 0x00;   // added for remote usage
}

int scan_keypad_rising_edge() {
 // CHECK KEYPAD
 scan_keypad();      // RETURNS RESULTS IN key_reg
 // ADD TEMPORARY VARIABLES
 unsigned char tmp_keypad[2];
 tmp_keypad[0] = key_reg[0];   //tmp_keypad HOLD ACQUIRED DATA FROM SCAN_KEYPAD()
 tmp_keypad[1] = key_reg[1];

 _delay_ms(0x15);     // APOFYGH SPINTHIRISMOU


 scan_keypad();
 key_reg[0] &= tmp_keypad[0];  // APPORIPSE TIS TIMES POU EMFANISAN SPINTHIRISMO
 key_reg[1] &= tmp_keypad[1];

 tmp_keypad[0] = mem[0];    // BRING LAST STATE OF SWITCHES FROM RAM TO tmp_keypad
 tmp_keypad[1] = mem[1];

 mem[0] = key_reg[0];    // STORE NEW KEYPAD STATE IN RAM FOR FUTURE CALL
 mem[1] = key_reg[1];


 key_reg[0] &= ~tmp_keypad[0];  // FIND KEYPAD SWITCHES THAT HAVE JUST BEEN
  ↪  PRESSED
 key_reg[1] &= ~tmp_keypad[1];

 return (key_reg[0] || key_reg[1]); // 16 BIT VALUE INDICATING FRESHLY PRESSED
  ↪  SWITCHES - RETURNS 0 IF NO SWITCH PRESSED
}

/* CONVERT VALUE TO ASCII CODE *CHECK COMMENT ABOVE SCAN_KEYPAD FOR CORRESPONDENCE
* key_reg[0] = third_row(4 MSB)-fourth_row(4 LSB)
* key_reg[1] = first_row(4 MSB)-second_row(4 LSB)
* LSB -> MSB == LEFT -> RIGHT IN KEYPAD */
unsigned char keypad_to_ascii() {
 if (key_reg[0] & 0x01)
 return '*';

 if (key_reg[0] & 0x02)
 return '0';
```

```
133
134   if (key_reg[0] & 0x04)
135   return '#';
136
137   if (key_reg[0] & 0x08)
138   return 'D';
139
140   if (key_reg[0] & 0x10)
141   return '7';
142
143   if (key_reg[0] & 0x20)
144   return '8';
145
146   if (key_reg[0] & 0x40)
147   return '9';
148
149   if (key_reg[0] & 0x80)
150   return 'C';
151
152   if (key_reg[1] & 0x01)
153   return '4';
154
155   if (key_reg[1] & 0x02)
156   return '5';
157
158   if (key_reg[1] & 0x04)
159   return '6';
160
161   if (key_reg[1] & 0x08)
162   return 'B';
163
164   if (key_reg[1] & 0x10)
165   return '1';
166
167   if (key_reg[1] & 0x20)
168   return '2';
169
170   if (key_reg[1] & 0x40)
171   return '3';
172
173   if (key_reg[1] & 0x80)
174   return 'A';
175
176   // Nothing Found
177   return 0;
178 }
179
180 // -------- START of New code added for LAB Exercise 4 ------
```

```
181
182 int Cx = 0; // Concentration of CO in ppm
183 unsigned char Led_ON(void){
184  if (Cx < 30) return 0x01; // if   0 <= CO < 30 ppm LEDS_PORTB ->   X0000001
185  if (Cx < 50) return 0x03; // if  30 <=  CO < 50 ppm LEDS_PORTB ->  X0000011
186  if (Cx < 70) return 0x07; // if  50 <=  CO < 70 ppm LEDS_PORTB ->  X0000111
187  if (Cx < 80) return 0x0F; // if  70 <=  CO < 80 ppm LEDS_PORTB ->  X0001111
188  if (Cx < 105) return 0x1F;// if  80 <=  CO < 105 ppm LEDS_PORTB ->  X0011111
189  if (Cx < 140) return 0x3F;// if 105 <=  CO < 140 ppm LEDS_PORTB ->  X0111111
190  return 0x7F;       // if CO >= 140 ppm LEDS_PORTB ->    X1111111
191 }
192 void ADC_init(void) // Initialize ADC
193 {
194  ADMUX = 0x40;
195  ADCSRA = (1<<ADEN | 1<<ADIE | 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0 );
196 }
197
198 int calc_cx (void) // Calculate Cx where Vin  = (ADC/5)/1024 and Cx = (1/M) * (Vin
    ↪  - Vgas0)
199 {
200  volatile float sensitivity = 129.0, Vgas0 = 0.1;
201  volatile float Vin = (ADC*5.0)/1024.0; // Vin  = (ADC/5)/1024
202  volatile float M = sensitivity * 0.0001; // Cx = (1/M) * (Vin - Vgas0)
203  return (int)((1/M) * (Vin - Vgas0));
204 }
205
206 ISR(ADC_vect) // ADC Interuption routine
207 {
208  Cx = calc_cx(); // Calculate Cx
209  leds = Led_ON(); // Tell me which leds should i Turn on
210  if (Cx > 70) // blink every 200ms (alarm) Using timer
211  {
212   if (Blink_flag == 0) PORTB = flag_MSB << 7; // Leds OFF
213   else PORTB = leds | (flag_MSB << 7); // Leds ON
214  }
215  else if (Cx <= 70)
216  {
217   PORTB = leds | (flag_MSB << 7); // else Just Leds ON
218  }
219
220 }
221
222 ISR(TIMER1_OVF_vect) // Timer Interuption routine
223 {
224
225  ADCSRA |= (1<<ADSC);          // Start the next conversion
226  TCNT1 = 64755;       //Timer set to overflow in 100 msec
227  TCCR1B = (1<<CS12) | (0<<CS11) | (1<<CS10); // Start again.
```

```
228
229     if (counter == 2) // Here we change the flag for the alarm (Blink_flag) every 2
        ↪  Timer Interrupts
230     {
231      Blink_flag = !Blink_flag;
232      if (HUGE_flag == 1) Blink_flag = 1; // HUGE_flag Turns ON only whenn we are in
         ↪  SUCCESS Mode (Correct passwd Typed) So we by pass Blink_flag to carry leds
         ↪  ON.
233      counter = 0;
234     }
235     counter++;
236    }
237
238    // -------- END of New code added for LAB Exercise 4 ------
239
240    int main(void) {
241
242     DDRB = 0xFF;         // PORTB => OUTPUT
243     DDRC = 0xF0;         // KEYPAD: PORTC[7:4] => OUTPUT, PORTC[3:0] => INPUT
244
245    // -------- START of New code added for LAB Exercise 4 ------
246
247     ADC_init();    // Initialize ADC
248
249     TIMSK = (1 << TOIE1);          //Timer1 ,interrupt enable
250     TCCR1B = (1<<CS12) | (0<<CS11) | (1<<CS10);      //frequency of Timer1 8MHz/1024
251     TCNT1 = 64755;       //Timer set to overflow in 100 msec
252
253     sei();    // enable interupts
254    // -------- END of New code added for LAB Exercise 4 ------
255     while (1) {
256      MAIN_L:
257
258      mem[0] = 0;    // INITIALIZE RAM
259      mem[1] = 0;
260      PORTB = 0;
261      flag = 0;
262
263      while (1) {
264
265       // GET FIRST DIGIT
266       if (scan_keypad_rising_edge()) {
267        first = keypad_to_ascii();
268        break;
269       }
270      }
271
272      // IF INPUT EQUAL WITH EXPECTED KEY SET FLAG
```

```
273     if (first == '0')
274     flag = 1;
275
276     // GET SECOND DIGIT
277     while (1) {
278      if (scan_keypad_rising_edge()) {
279       second = keypad_to_ascii();
280       scan_keypad_rising_edge();  // EXTRA CALL ADDED FOR REMOTE USAGE
281       break;
282      }
283     }
284
285     // IF INPUT NOT EQUAL WITH EXPECTED KEY OR FLAG NOT SET INDICATING FIRST DIGIT
        ↪  WRONG ->  WRONG_INPUT
286     if (second != '3' || (!flag)) { goto WRONG_INPUT; }
287
288     // SUCCESSFUL
289
290     SUCCESS;
291     goto MAIN_L;
292
293
294     WRONG_INPUT:
295     BLINK_FAIL;
296    }
297    return 0;
298   }
```