ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Μικροϋπολογιστών

3η Εργαστηριακή Άσκηση

Ονοματεπώνυμα: **Μπέτζελος Χρήστος, Γιαννιός Γεώργιος-Ταξιάρχης**

Α.Μ. : **031 16 067, 031 16 156**

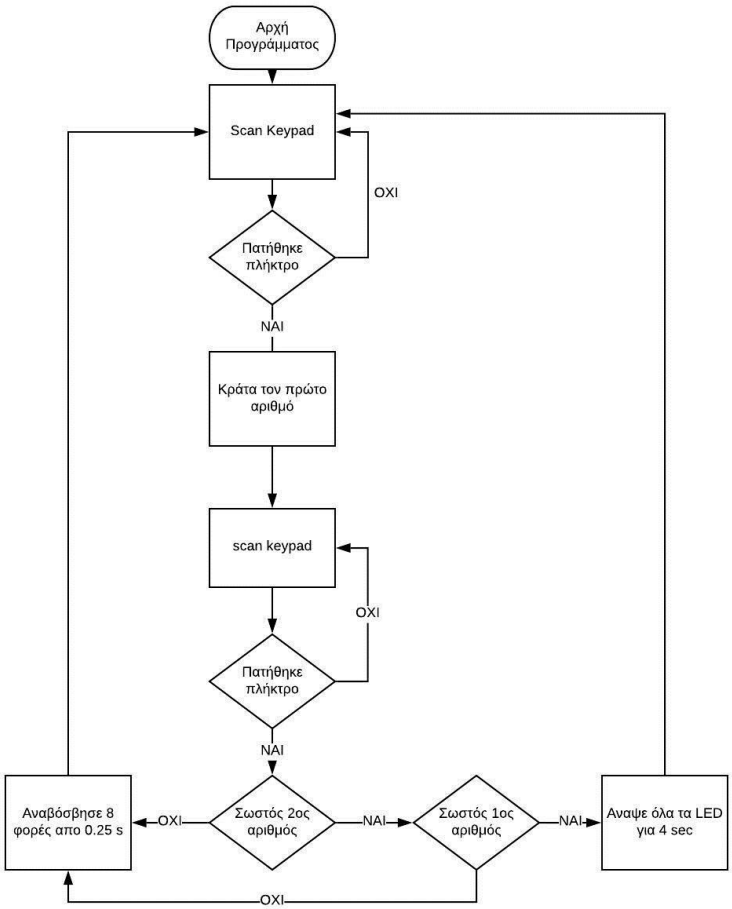
Ομάδα : **Α 16**

Εξάμηνο: **7ο**

1η Άσκηση

Γράψαμε ένα πρόγραμμα «ηλεκτρονικής κλειδαριάς» το οποίο ανάβει όλα τα leds PB0-7 για 4 sec συνολικά, μόνο όταν πατηθούν στη σειρά τα δύο πλήκτρα στο keypad 4×4 που αντιστοιχούν στο διψήφιο αριθμό της ομάδας μας (16). Αν δεν έχουν δοθεί οι δύο σωστοί αριθμοί αναβοσβήνει (χρόνος ~0.25 sec αναμμένο και ~0.25 sec σβησμένο) τα leds PB0-7 επίσης για 4 sec. Ανεξάρτητα από το χρονικό διάστημα για το οποίο θα μείνει πατημένο ένα πλήκτρο, το πρόγραμμά θεωρεί ότι πατήθηκε μόνο μια φορά. Μετά το πάτημα δύο αριθμών το πρόγραμμα δεν δέχεται για 4 sec άλλον αριθμό. Το πρόγραμμα αυτό είναι συνεχόμενης λειτουργίας. Παρακάτω δίδεται το διάγραμμα ροής και το πρόγραμμα σε assembly και σε C. Επίσης όλες οι ρουτίνες δίνονται σε μορφή συνάρτησης C για να αξιοποιηθούν από το πρόγραμμα σε C.

Διάγραμμα ροής



Πρόγραμμα σε assembly

```
#include "m16def.inc"
.DSEG
 _tmp_: .byte 2
.CSEG
reset:
 ;Αρχικοποίηση δείκτη στοίβας
 ldi r24 , low(RAMEND)
 out SPL , r24
 ldi r24 , high(RAMEND)
 out SPH , r24

 ser r24
 out DDRB, r24          ;PORTB έξοδος

;Θέτουμε ως έξοδο τα 4 MSB του PORTC για το keybpad
 ldi r24 ,(1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
 out DDRC ,r24
first_key:
 ldi r24,0x05          ;5ms σπινθηρισμού
 rcall scan_keypad_rising_edge
 rcall keypad_to_ascii
 cpi r24,0x00          ;Αν είναι μηδέν τότε δεν πατήθηκε τίποτα και
 breq first_key               ;ξαναπήγαινε στο first_key
 mov r28, r24          ;Αποθήκευσε προσωρινά στον r28
```

```
second_key:
 ldi r24,0x05          ;5ms σπινθηρισμού
 rcall scan_keypad_rising_edge
 rcall keypad_to_ascii
 cpi r24,0x00          ;Αν είναι μηδέν τότε δεν πατήθηκε τίποτα και
 breq second_key       ;ξαναπήγαινε στο second_key
 ldi r16,8                    ;Φόρτωση μετρητή άναψε/σβήσε
 cpi r24,'6'           ;Σύγκρινε τον 2ο αριθμό εισόδου με της ομάδας
 brne wrong                   ;Αν δεν είναι ίσοι πήγαινε στο wrong
 cpi r28,'1'           ;Σύγκρινε τον 1ο αριθμό εισόδου με της ομάδας
 brne wrong                   ;Αν δεν είναι ίσοι πήγαινε στο wrong

alright:
 ldi r26, 0xFF
 out PORTB, r26               ;Άναμμα όλων των LED
 ldi r24, low(4000) ;delay 4sec
 ldi r25, high(4000)
 rcall wait_msec
 ldi r26, 0x00
 out PORTB, r26        ;Σβήνουν όλα τα LED και
 rjmp reset                   ;ξαναπηγαίνουμε στην αρχή να πάρουμε νέα είσοδο

wrong:
 ldi r26, 0xFF
 out PORTB, r26               ;Άναμμα όλων των LED για 0.25 sec
 ldi r24, low(250)     ;delay 0.25sec
 ldi r25, high(250)
 rcall wait_msec
 ldi r26, 0x00
 out PORTB, r26               ;Σβήσιμο όλων των LED για 0.25 sec
 ldi r24, low(250)
 ldi r25, high(250)
 rcall wait_msec
 dec r16
 breq reset                   ;Όταν γίνει αυτό 8 φορές (4sec)
 rjmp wrong                   ;επιστρέφουμε στην αρχή να πάρουμε νέα είσοδο

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

```
 ldi r24 ,'C'
 sbrc r26 ,7
 ret
 ldi r24 ,'4'  ;λογικό '1' στις θέσεις του καταχωρητή r27 δηλώνουν
 sbrc r27 ,0   ;τα παρακάτω σύμβολα και αριθμούς
 ret
 ldi r24 ,'5'
 sbrc r27 ,1
 ret
 ldi r24 ,'6'
 sbrc r27 ,2
 ret
 ldi r24 ,'B'
 sbrc r27 ,3
 ret
 ldi r24 ,'1'
 sbrc r27 ,4
 ret
 ldi r24 ,'2'
 sbrc r27 ,5
 ret
 ldi r24 ,'3'
 sbrc r27 ,6
 ret
 ldi r24 ,'A'
 sbrc r27 ,7
 ret
 clr r24              ;αν δεν πατήθηκε τίποτα επιστρέφει μηδέν
 ret

scan_keypad_rising_edge:
 mov r22 ,r24
 rcall scan_keypad
 push r24
 push r25
 mov r24 ,r22
 ldi r25 ,0
 rcall wait_msec
 rcall scan_keypad
 pop r23
 pop r22
 and r24 ,r22
 and r25 ,r23
 ldi r26 ,low(_tmp_)
 ldi r27 ,high(_tmp_)
 ld r23 ,X+
 ld r22 ,X
 st X ,r24
 st -X ,r25
 com r23
 com r22
 and r24 ,r22
 and r25 ,r23
 ret

;Ρουτίνα ελέγχου όλου του πληκτρολογίου
scan_keypad:
 ldi r24 , 0x01       ; έλεγξε την πρώτη γραμμή του πληκτρολογίου
 rcall scan_row
 swap r24             ; αποθήκευσε το αποτέλεσμα
 mov r27 , r24        ; στα 4 msb του r27
```

```
 ldi r24 ,0x02 ; έλεγξε τη δεύτερη γραμμή του πληκτρολογίου
 rcall scan_row
 add r27 , r24         ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r27
 ldi r24 , 0x03 ; έλεγξε την τρίτη γραμμή του πληκτρολογίου
 rcall scan_row
 swap r24              ; αποθήκευσε το αποτέλεσμα
 mov r26 , r24         ; στα 4 msb του r26
 ldi r24 ,0x04         ; έλεγξε την τέταρτη γραμμή του πληκτρολογίου
 rcall scan_row
 add r26 , r24         ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r26
 movw r24 , r26 ; μετέφερε το αποτέλεσμα στους καταχωρητές r25:r24
 ret

;Ρουτίνα ελέγχου μιας γραμμής του πληκτρολογίου
scan_row:
 ldi r25 , 0x08        ; αρχικοποίηση με '0000 1000'
back_:
 lsl r25               ; αριστερή ολίσθηση του '1' τόσες θέσεις
 dec r24               ; όσος είναι ο αριθμός της γραμμής
 brne back_
 out PORTC , r25; η αντίστοιχη γραμμή τίθεται στο λογικό '1'
 nop                   ; καθυστέρηση για να προλάβει να γίνει η αλλαγή κατάστασης
 nop
 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.

;Ρουτίνες χρονοκαθυστέρησης
wait_msec:
 push r24
 push r25
 ldi r24 , low(998)
 ldi r25 , high(998)
 rcall wait_usec
 pop r25
 pop r24
 sbiw r24 , 1
 brne wait_msec
 ret

wait_usec:
 sbiw r24 ,1
 nop
 nop
 nop
 nop
 brne wait_usec
 ret
```

Πρόγραμμα σε C

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

char scan_row(int);
void scan_keypad(char *);
void scan_keypad_rising_edge(char*, char*);
char keypad_to_ascii(char *);

void main(void)
{
	SP = RAMEND;
	char prev[2],next[2],c,pressed[2];
	DDRB=0xFF; // Initiating PORT B as output

	while(1)
	{
		DDRC=(1<<PC7)|(1<<PC6)|(1<<PC5)|(1<<PC4); // Initiating PORT C4-C7 as output
		scan_keypad_rising_edge(prev,next);
		while((c=keypad_to_ascii(next))=='@')
		{
			scan_keypad_rising_edge(prev,next);
		}
		pressed[0]=c;

		scan_keypad_rising_edge(prev,next);
		while((c=keypad_to_ascii(next))=='@')
		{
			scan_keypad_rising_edge(prev,next);
	    }
		pressed[1]=c;
		if((pressed[0]=='1') && (pressed[1]=='6'))
		{
			PORTB=0xFF;
			_delay_ms(4000);
			PORTB=0x00;
		}
		else {
			for(int i=0; i<8; i++){
				PORTB=0xFF;
				_delay_ms(250);
				PORTB=0x00;
				_delay_ms(250);
			}
		}
	}
}
char scan_row(int row){
	char x=0x08,a;
	x=x<<row;
	PORTC =x;
	_delay_us(1);
	a=PINC & 0x0F;
	return a;
}
void scan_keypad(char next_st[2])
{
	next_st[0]=0x00;
```

```
        next_st[1]=0x00;

        char line=scan_row(1) & 0x0f;
        char temp = line<<4;
        next_st[0]=temp;

        line=scan_row(2) & 0x0f;
        next_st[0]=next_st[0]|line;

        line=scan_row(3) & 0x0f;
        temp = line<<4;
        next_st[1]=temp;

        line=scan_row(4) & 0x0f;
        next_st[1]=next_st[1]|line;
        return;
}
void scan_keypad_rising_edge(char prev_st[2], char next_st[2]) {

        scan_keypad(next_st);
        char temp[2];

        temp[0] = next_st[0];
        temp[1] = next_st[1];
        _delay_ms(15);

        scan_keypad(next_st);

        next_st[0] = next_st[0] & temp[0];
        next_st[1] = next_st[1] & temp[1];

        temp[0] = ~prev_st[0];
        temp[1] = ~prev_st[1];

        prev_st[0] = next_st[0];
        prev_st[1] = next_st[1];

        next_st[0] = next_st[0] & temp[0];
        next_st[1] = next_st[1] & temp[1];
        return;
}
char keypad_to_ascii(char * keys)
{
        if ((keys[1]&0x01)==0x01) return '*';
        if ((keys[1]&0x02)==0x02) return '0';
        if ((keys[1]&0x04)==0x04) return '#';
        if ((keys[1]&0x08)==0x08) return 'D';
        if ((keys[1]&0x10)==0x10) return '7';
        if ((keys[1]&0x20)==0x20) return '8';
        if ((keys[1]&0x40)==0x40) return '9';
        if ((keys[1]&0x80)==0x80) return 'C';
        if ((keys[0]&0x01)==0x01) return '4';
        if ((keys[0]&0x02)==0x02) return '5';
        if ((keys[0]&0x04)==0x04) return '6';
        if ((keys[0]&0x08)==0x08) return 'B';
        if ((keys[0]&0x10)==0x10) return '1';
        if ((keys[0]&0x20)==0x20) return '2';
        if ((keys[0]&0x40)==0x40) return '3';
        if ((keys[0]&0x80)==0x80) return 'A';
        return '@';
}
```

2η Άσκηση

Γράψαμε πρόγραμμα σε assembly και C που απεικονίζει στο LCD display διψήφιο δεκαεξαδικό αριθμό (υποθέτουμε ότι είναι σε συμπλήρωμα ως προς 2) που δίνεται από το keypad 4×4 με την χρήση της ρουτίνα keypad_to_hex. Ο αριθμός μετατρέπεται σε δεκαδική μορφή τριών ψηφίων με το πρόσημο (δηλαδή αν από το πληκτρολόγιο δοθεί π.χ. ο αριθμός 6E, τότε ξεκινώντας από την πάνω αριστερή θέση του display εμφανίζεται «6E=+110» ενώ αν δοθεί ο αριθμός 80 εμφανίζεται το «80=- 128»). Η διαδικασία είναι συνεχόμενη. Έγινε μετατροπή στην ρουτίνα keypad_to_ascii ώστε να αναγνωρίζει το '#' ως 'F' και το '*' ως 'E' και να μπορεί το πληκτρολόγιο να χρησιμοποιηθεί ως δεκαεξαδικό. Η νέα ρουτίνα keypad_to_hex επιστρέφει την δεκαεξαδική τιμή των πλήκτρων συμπεριλαμβανομένων των '#' και '*' που θα θεωρούνται πλέον ως 'F' και 'E' αντίστοιχα. Όλες οι ρουτίνες δίνονται σε μορφή συνάρτησης C για να αξιοποιηθούν από το πρόγραμμα σε C. Ακολουθήσαμε το διάγραμμα ροής που μας δόθηκε για τη μετατροπή ενός προσημασμένου δυαδικού αριθμού σε BCD μορφή.

Πρόγραμμα σε assembly

```
#include "m16def.inc"
.DSEG
 _tmp_: .byte 2
.CSEG
reset:
 ;Αρχικοποίηση δείκτη στοίβας
 ldi r24 , low(RAMEND)
 out SPL , r24
 ldi r24 , high(RAMEND)
 out SPH , r24

 ldi r24, 0xfc          ;αρχικοποίηση PORTD που συνδέεται
 out DDRD, r24          ;η οθόνη ως έξοδος

;Θέτουμε ως έξοδο τα 4 MSB του PORTC
 ldi r24 ,(1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)
 out DDRC ,r24

first_key:
 ldi r24,0x05           ;5ms σπινθηρισμού
 rcall scan_keypad_rising_edge
 rcall keypad_to_hex
 cpi r24,0x10           ;αν είναι 0x10 τότε δεν πατήθηκε τίποτα και
 breq first_key         ;ξαναπήγαινε στο first_key
 andi r24,0x0F          ;απομώνωσε τα LSB
 mov r17,r24            ;r17=1ο ψηφίο
 swap r24               ;LSB -> MSB
 mov r16, r24           ;αποθήκευσε προσωρινά στον r16

second_key:
 ldi r24,0x05           ;5ms σπινθηρισμού
 rcall scan_keypad_rising_edge
 rcall keypad_to_hex
 cpi r24,0x10           ;αν είναι 0x10 τότε δεν πατήθηκε τίποτα και
 breq second_key        ;ξαναπήγαινε στο second_key
 andi r24,0x0F          ;απομώνωσε τα LSB
```

```
 mov r18,r24          ;r18=2ο ψηφίο
 add r24,r16          ;πρόσθεσε τον προηγούμενο αριθμό
;Έλεγχος μηδενός
cpi r24,0x00
brne sign

zero:
 rcall lcd_init
 ldi r24, '0'
 rcall lcd_data
 ldi r24, '0'
 rcall lcd_data
 ldi r24, '='
 rcall lcd_data
 ldi r24, '0'
 rcall lcd_data
 rjmp reset

;Έλεγχος προσήμου
sign:
 sbrs r24,7           ;Αν r24(7)=1 έχουμε αρνητικό αριθμό
 rjmp positive

negative:             ;Βρίσκουμε το μέτρο του αρνητικού αριθμού
 ldi r29,'-'
 andi r24,0x7F
 ldi r25,0x80
 sub r25,r24
 mov r24,r25
 rjmp bcd

positive:
 ldi r29,'+'

bcd:
 cpi r24,0x64         ;Σύγκριση μέτρου με το 100
 brcc ekat            ;Αν είναι >= 100 πήγαινε στο ekat
 ldi r28,'0'          ;Εκατοντάδες (r28) = 0
 rjmp deci

ekat:
 ldi r28,'1'          ;Εκατοντάδες (r28) = 1
 subi r24,0x64

deci:
 ldi r27,0x00         ;Αρχικοποίηση δεκάδων (r27) στο 0
 cpi r24,0x0A         ;Σύγκριση μέτρου με το 10
 brcc mon             ;Αν είναι >= 10 πήγαινε στο mon
 rjmp end

mon:
 inc r27
 subi r24,0x0A
 cpi r24,0x0A         ;Σύγκριση μέτρου με το 10
 brcc mon             ;Αν είναι >= 10 πήγαινε στο mon

end:
 ldi r16,0x30
 add r27,r16          ;Μετατροπή σε ascii
 add r24,r16          ;Μετατροπή σε ascii
 mov r26,r24          ;Μετατροπή σε ascii
```

```
 add r17,r16          ;Μετατροπή σε ascii
 add r18,r16          ;Μετατροπή σε ascii
 cpi r17,0x3A
 brcc greater1
 cpi r18,0x3A
 brcc greater2
 rjmp lcd

greater1:
 ldi r16,0x07
 add r17, r16
 cpi r18,0x3A
 brcc greater2
 rjmp lcd

greater2:
 ldi r16,0x07
 add r18, r16

;Οπότε τελικά έχουμε στον r29:πρόσημο, r28:ekat, r27:dec, r26:mon και αρχικός διψήφιος στους
r17-r18
lcd:
rcall lcd_init        ;Αρχικοποίηση οθόνης
mov r24, r17
rcall lcd_data        ;Εμφάνιση 1ου ψηφίου
mov r24, r18
rcall lcd_data        ;Εμφάνιση 2ου ψηφίου
ldi r24,'='
rcall lcd_data        ;Εμφάνιση του ίσου
mov r24, r29          ;Εμφάνιση προσήμου
rcall lcd_data        ;στην οθόνη

cpi r28,'0'                 ;Αν ekat!=0
brne triple                 ;πήγαινε στο triple

double:
 cpi r27,'0'          ;Αν dec==0
 breq single          ;πήγαινε στο single
 mov r24, r27
 rcall lcd_data              ;Τύπωσε δεκάδες
 mov r24, r26
 rcall lcd_data              ;Τύπωσε μονάδες
 rjmp reset

single:
 mov r24, r26
 rcall lcd_data              ;Τύπωσε μονάδες
 rjmp reset

triple:
 mov r24, r28
 rcall lcd_data              ;Τύπωσε εκατοντάδες
 mov r24, r27
 rcall lcd_data              ;Τύπωσε δεκάδες
 mov r24, r26
 rcall lcd_data              ;Τύπωσε μονάδες
 rjmp reset


;ρουτίνα αντιστοίχισης διακοπτών σε hex
keypad_to_hex:
```

```
 movw r26 ,r24
 ldi r24 ,0x0E
 sbrc r26 ,0
 ret
 ldi r24 ,0x00
 sbrc r26 ,1
 ret
 ldi r24 ,0x0F
 sbrc r26 ,2
 ret
 ldi r24 ,0x0D
 sbrc r26 ,3
 ret
 ldi r24 ,0x07
 sbrc r26 ,4
 ret
 ldi r24 ,0x08
 sbrc r26 ,5
 ret
 ldi r24 ,0x09
 sbrc r26 ,6
 ret
 ldi r24 ,0x0C
 sbrc r26 ,7
 ret
 ldi r24 ,0x04
 sbrc r27 ,0
 ret
 ldi r24 ,0x05
 sbrc r27 ,1
 ret
 ldi r24 ,0x06
 sbrc r27 ,2
 ret
 ldi r24 ,0x0B
 sbrc r27 ,3
 ret
 ldi r24 ,0x01
 sbrc r27 ,4
 ret
 ldi r24 ,0x02
 sbrc r27 ,5
 ret
 ldi r24 ,0x03
 sbrc r27 ,6
 ret
 ldi r24 ,0x0A
 sbrc r27 ,7
 ret
 ldi r24, 0x10         ;Αν δεν πατήθηκε τίποτα φόρτωσε το 0x010
 ret

scan_keypad_rising_edge:
 mov r22 ,r24
 rcall scan_keypad
 push r24
 push r25
 mov r24 ,r22
 ldi r25 ,0
 rcall wait_msec
 rcall scan_keypad
```

```
 pop r23
 pop r22
 and r24 ,r22
 and r25 ,r23
 ldi r26 ,low(_tmp_)
 ldi r27 ,high(_tmp_)
 ld r23 ,X+
 ld r22 ,X
 st X ,r24
 st -X ,r25
 com r23
 com r22
 and r24 ,r22
 and r25 ,r23
 ret

;Ρουτίνα ελέγχου όλου του πληκτρολογίου
scan_keypad:
 ldi r24 , 0x01        ; έλεγξε την πρώτη γραμμή του πληκτρολογίου
 rcall scan_row
 swap r24              ; αποθήκευσε το αποτέλεσμα
 mov r27 , r24         ; στα 4 msb του r27
 ldi r24 ,0x02 ; έλεγξε τη δεύτερη γραμμή του πληκτρολογίου
 rcall scan_row
 add r27 , r24         ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r27
 ldi r24 , 0x03 ; έλεγξε την τρίτη γραμμή του πληκτρολογίου
 rcall scan_row
 swap r24              ; αποθήκευσε το αποτέλεσμα
 mov r26 , r24         ; στα 4 msb του r26
 ldi r24 ,0x04         ; έλεγξε την τέταρτη γραμμή του πληκτρολογίου
 rcall scan_row
 add r26 , r24         ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r26
 movw r24 , r26 ; μετέφερε το αποτέλεσμα στους καταχωρητές r25:r24
 ret

;Ρουτίνα ελέγχου μιας γραμμής του πληκτρολογίου
scan_row:
 ldi r25 , 0x08        ; αρχικοποίηση με '0000 1000'
back_:
 lsl r25               ; αριστερή ολίσθηση του '1' τόσες θέσεις
 dec r24               ; όσος είναι ο αριθμός της γραμμής
 brne back_
 out PORTC , r25; η αντίστοιχη γραμμή τίθεται στο λογικό '1'
 nop                   ; καθυστέρηση για να προλάβει να γίνει η αλλαγή κατάστασης
 nop
 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.

;ρουτίνα αρχικοποίσης και ρυθμίσεων LCD
lcd_init:
 ldi r24 ,40
 ldi r25 ,0
 rcall wait_msec
 ldi r24 ,0x30
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
```

```
 ldi r24 ,0x30
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ldi r24 ,0x20
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ldi r24 ,0x28
 rcall lcd_command
 ldi r24 ,0x0c
 rcall lcd_command
 ldi r24 ,0x01
 rcall lcd_command
 ldi r24 ,low(1530)
 ldi r25 ,high(1530)
 rcall wait_usec
 ldi r24 ,0x06
 rcall lcd_command
 ret

;ρουτίνα αποστολής ενός byte δεδομένων στην LCD
lcd_data:
 sbi PORTD ,PD2
 rcall write_2_nibbles
 ldi r24 ,43
 ldi r25 ,0
 rcall wait_usec
 ret

;ρουτίνα αποστολής μιας εντολής στην LCD
lcd_command:
 cbi PORTD ,PD2
 rcall write_2_nibbles
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ret

;ρουτίνα αποστολής ενός byte στην LCD
write_2_nibbles:
 push r24
 in r25 ,PIND
 andi r25 ,0x0f
 andi r24 ,0xf0
 add r24 ,r25
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 pop r24
 swap r24
 andi r24 ,0xf0
 add r24 ,r25
 out PORTD ,r24
 sbi PORTD ,PD3
```

```
 cbi PORTD ,PD3
 ret
;Ρουτίνες χρονοκαθυστέρησης
wait_msec:
 push r24
 push r25
 ldi r24 , low(998)
 ldi r25 , high(998)
 rcall wait_usec
 pop r25
 pop r24
 sbiw r24 , 1
 brne wait_msec
 ret

wait_usec:
 sbiw r24 ,1
 nop
 nop
 nop
 nop
 brne wait_usec
 ret
```

Πρόγραμμα σε C

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

char scan_row(int);
void scan_keypad(char *);
void scan_keypad_rising_edge(char*, char*);
int keypad_to_hex(char *);

void lcd_init (void);
void write_2_nibbles(char);
void lcd_data(char);
void lcd_command(char);

void main(void)
{
	SP = RAMEND;
	char prev[2],next[2],c,pressed[2];
	DDRD=0xFC;
	while(1)
	{
		char sign,metro=0,ekat=0,dec=0,mon=0;
		DDRC=(1<<PC7)|(1<<PC6)|(1<<PC5)|(1<<PC4); // Initiating PORT C4-C7 as output
		scan_keypad_rising_edge(prev,next);
		while((c=keypad_to_hex(next))==0x10)
		{
			scan_keypad_rising_edge(prev,next);
		}
		pressed[0]=c & 0x0f;

		scan_keypad_rising_edge(prev,next);
		while((c=keypad_to_hex(next))==0x10)
		{
			scan_keypad_rising_edge(prev,next);
```

```
                }

                pressed[1]=c & 0x0f;
                lcd_init();
                if(pressed[0]>=10){
                        lcd_data(pressed[0]+0x37);
                }
                else
                        lcd_data(pressed[0]+0x30);
                }
                if(pressed[1]>=10){
                        lcd_data(pressed[1]+0x37);
                }
                else{
                        lcd_data(pressed[1]+0x30);
                }
                lcd_data('=');
                c=(pressed[0]<<4)|pressed[1];
                if(c==0){
                        lcd_data('0');
                }
                else{
                        if(c>=0x80){
                                sign='-';
                                metro=(0x80-(c & 0x7f));
                        }
                        else{
                                sign='+';
                                metro=c;
                        }
                        if(metro>=100){
                                ekat=1;
                                metro=metro-100;
                        }
                        while(metro>=10){
                                dec++;
                                metro=metro-10;
                        }
                        mon=metro;
                        lcd_data(sign);
                        if(ekat==0){
                                if(dec==0){
                                        lcd_data((mon & 0x0f)+0x30);
                                }
                                else{
                                        lcd_data((dec & 0x0f)+0x30);
                                        lcd_data((mon & 0x0f)+0x30);
                                }
                        }
                        else{
                                lcd_data((ekat & 0x0f)+0x30);
                                lcd_data((dec & 0x0f)+0x30);
                                lcd_data((mon & 0x0f)+0x30);
                        }
                }
        }
}
void write_2_nibbles(char x)
{
        char y=PIND & 0x0f;
        char x1=x & 0xf0;
```

```
        x1=x1+y;
        PORTD=x1;
        PORTD=PORTD | (1<<PD3);
        PORTD=PORTD & (0<<PD3);
        x=x<<4 | x>>4;
        x=x & 0xf0;
        PORTD=x+y;
        PORTD=PORTD | (1<<PD3);
        PORTD=PORTD & (0<<PD3);
}

void lcd_data(char x)
{
        PORTD=PORTD | (1<<PD2);
        write_2_nibbles(x);
        _delay_us(43);
}

void  lcd_command(char x)
{
        PORTD=PORTD | (0<<PD2);
        write_2_nibbles(x);
        _delay_us(43);
}

void lcd_init (void)
{
        _delay_ms(40);
        PORTD=0x30;
        PORTD=PORTD | (1<<PD3);
        PORTD=PORTD & (0<<PD3);
        _delay_us(39);
        PORTD=0x30;
        PORTD=PORTD | (1<<PD3);
        PORTD=PORTD & (0<<PD3);
        _delay_us(39);
        PORTD=0x20;
        PORTD=PORTD | (1<<PD3);
        PORTD=PORTD & (0<<PD3);
        _delay_us(39);
        lcd_command(0x28);
        lcd_command(0x0c);
        lcd_command(0x01);
        _delay_us(1530);
        lcd_command(0x06);
}
char scan_row(int row)
{
        char x=0x08,a;
        x=x<<row;
        PORTC =x;
        _delay_us(1);
        a=PINC & 0x0F;
        return a;
}
void scan_keypad(char next_st[2])
{
        next_st[0]=0x00;
        next_st[1]=0x00;

        char line=scan_row(1) & 0x0f;
```

```
        char temp = line<<4;
        next_st[0]=temp;
        line=scan_row(2) & 0x0f;
        next_st[0]=next_st[0]|line;

        line=scan_row(3) & 0x0f;
        temp = line<<4;
        next_st[1]=temp;

        line=scan_row(4) & 0x0f;
        next_st[1]=next_st[1]|line;

        return;
}
void scan_keypad_rising_edge(char prev_st[2], char next_st[2]) {

        scan_keypad(next_st);
        char temp[2];

        temp[0] = next_st[0];
        temp[1] = next_st[1];

        _delay_ms(15);

        scan_keypad(next_st);

        next_st[0] = next_st[0] & temp[0];
        next_st[1] = next_st[1] & temp[1];

        temp[0] = ~prev_st[0];
        temp[1] = ~prev_st[1];

        prev_st[0] = next_st[0];
        prev_st[1] = next_st[1];

        next_st[0] = next_st[0] & temp[0];
        next_st[1] = next_st[1] & temp[1];

        return;

}
int keypad_to_hex(char * keys)
{
        if ((keys[1]&0x01)==0x01) return 0x0E;
        if ((keys[1]&0x02)==0x02) return 0x00;
        if ((keys[1]&0x04)==0x04) return 0x0F;
        if ((keys[1]&0x08)==0x08) return 0x0D;
        if ((keys[1]&0x10)==0x10) return 0x07;
        if ((keys[1]&0x20)==0x20) return 0x08;
        if ((keys[1]&0x40)==0x40) return 0x09;
        if ((keys[1]&0x80)==0x80) return 0x0C;
        if ((keys[0]&0x01)==0x01) return 0x04;
        if ((keys[0]&0x02)==0x02) return 0x05;
        if ((keys[0]&0x04)==0x04) return 0x06;
        if ((keys[0]&0x08)==0x08) return 0x0B;
        if ((keys[0]&0x10)==0x10) return 0x01;
        if ((keys[0]&0x20)==0x20) return 0x02;
        if ((keys[0]&0x40)==0x40) return 0x03;
        if ((keys[0]&0x80)==0x80) return 0x0A;
        return 0x10;
}
```

3η Άσκηση

Δημιουργήσαμε ένα ψηφιακό χρονόμετρο. Το πρόγραμμα μας (σε assembly και C) απεικονίζει στην οθόνη LCD το χρόνο στη μορφή:

Λεπτά : Δευτερόλεπτα

Τα λεπτά και τα δευτερόλεπτα απεικονίζονται με δύο δεκαδικά ψηφία (έχοντας δίπλα τους τις αντίστοιχες ενδείξεις MIN, SEC) και όταν το χρονόμετρο φτάνει την τιμή (59 MIN:59 SEC) θα πρέπει ξεκινάει πάλι από την αρχή. Το χρονόμετρο ξεκινά όταν πατάμε το πλήκτρο PB0 και όταν το αφήνουμε σταματάει και η ένδειξη παραμένει. Αν ξαναπατήσουμε το πλήκτρο PB0 συνεχίζει από το σημείο που έμεινε. Σταματά και μηδενίζεται όταν πατάμε το πλήκτρο PB7. Μεταξύ των δύο πλήκτρων θεωρούμε ότι PB7 έχει υψηλότερη προτεραιότητα. Κατά την έναρξη λειτουργίας εμφανίζεται στην οθόνη η ένδειξη: 00 MIN:00 SEC.

Πρόγραμμα σε assembly

```
#include "m16def.inc"

reset:
 ;Αρχικοποίηση δείκτη στοίβας
 ldi r24 , low(RAMEND)
 out SPL , r24
 ldi r24 , high(RAMEND)
 out SPH , r24

 ldi r24,0xfc          ;αρχικοποίηση PORTD που συνδέεται
 out DDRD, r24         ;η οθόνη ως έξοδος
 clr r24
 out DDRB,r24

 ;Αρχικοποίηση μετρητή
clear:
 ldi r29, '0'          ;min_decades
 ldi r28, '0'          ;min_monades
 ldi r27, '0'          ;sec_decades
 ldi r26, '0'          ;sec_monades

loopp:
 rcall lcd_init              ;Αρχικοποίηση οθόνης
 mov r24, r29
 rcall lcd_data              ;Τύπωσε min_decades
 mov r24, r28
 rcall lcd_data              ;Τύπωσε min_monades
 ldi r24, 'M'
 rcall lcd_data              ;Τύπωσε το 'M'
 ldi r24, 'I'
 rcall lcd_data              ;Τύπωσε το 'I'
 ldi r24, 'N'
 rcall lcd_data              ;Τύπωσε το 'N'
 ldi r24, ':'
 rcall lcd_data              ;Τύπωσε το ':'
 mov r24, r27
 rcall lcd_data              ;Τύπωσε sec_decades
 mov r24, r26
```

```
 rcall lcd_data               ;Τύπωσε sec_monades
 ldi r24, 'S'
 rcall lcd_data               ;Τύπωσε το 'S'
 ldi r24, 'E'
 rcall lcd_data               ;Τύπωσε το 'E'
 ldi r24, 'C'
 rcall lcd_data               ;Τύπωσε το 'C'

 ldi r24,low(1000)
 ldi r25,high(1000)
 rcall wait_msec      ;Καθυστέρησε 1sec

 cpi r26, '9'
 breq zero_sec
 inc r26

 stop_1:
 sbic PINB,7          ;Αν πατηθεί το PB7, ξεκίνα τη μέτρηση απο
 rjmp clear           ;την αρχή
;Οσο ο διακόπτης ειναι 1
 sbis PINB,0          ;συνέχισε τη μετρηση σου
 rjmp stop_1
 rjmp loopp

zero_sec:
 cpi r27, '5'
 breq one_minute
 inc r27
 ldi r26, '0'

stop_2:
 sbic PINB,7          ;Αν πατηθεί το PB7, ξεκίνα τη μέτρηση απο
 rjmp clear           ;την αρχή
;Οσο ο διακόπτης ειναι 1
 sbis PINB,0          ;συνέχισε τη μετρηση σου
 rjmp stop_2
 rjmp loopp

one_minute:
 cpi r28, '9'
 breq zero_min
 inc r28
 ldi r27, '0'
 ldi r26, '0'

stop_3:
 sbic PINB,7          ;Αν πατηθεί το PB7, ξεκίνα τη μέτρηση απο
 rjmp clear           ;την αρχή
;Οσο ο διακόπτης ειναι 1
 sbis PINB,0          ;συνέχισε τη μετρηση σου
 rjmp stop_3
 rjmp loopp

zero_min:
 cpi r29, '5'
 breq one_hour
 inc r29
 ldi r28, '0'
 ldi r27, '0'
 ldi r26, '0'
```

```
stop_4:
 sbic PINB,7           ;Αν πατηθεί το PB7, ξεκίνα τη μέτρηση απο
 rjmp clear            ;την αρχή
;Όσο ο διακόπτης ειναι 1
 sbis PINB,0           ;συνέχισε τη μετρηση σου
 rjmp stop_4
 rjmp loopp

one_hour:
 ldi r29, '0'
 ldi r28, '0'
 ldi r27, '0'
 ldi r26, '0'

stop_5:
 sbic PINB,7           ;Αν πατηθεί το PB7, ξεκίνα τη μέτρηση απο
 rjmp clear            ;την αρχή
;Όσο ο διακόπτης ειναι 1
 sbis PINB,0           ;συνέχισε τη μετρηση σου
 rjmp stop_5
 rjmp loopp

;ρουτίνα αρχικοποίσης και ρυθμίσεων LCD
lcd_init:
 ldi r24 ,40
 ldi r25 ,0
 rcall wait_msec
 ldi r24 ,0x30
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ldi r24 ,0x30
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ldi r24 ,0x20
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ldi r24 ,0x28
 rcall lcd_command
 ldi r24 ,0x0c
 rcall lcd_command
 ldi r24 ,0x01
 rcall lcd_command
 ldi r24 ,low(1530)
 ldi r25 ,high(1530)
 rcall wait_usec
 ldi r24 ,0x06
 rcall lcd_command
 ret
```

```
;ρουτίνα αποστολής ενός byte δεδομένων στην LCD
lcd_data:
 sbi PORTD ,PD2
 rcall write_2_nibbles
 ldi r24 ,43
 ldi r25 ,0
 rcall wait_usec
 ret

;ρουτίνα αποστολής μιας εντολής στην LCD
lcd_command:
 cbi PORTD ,PD2
 rcall write_2_nibbles
 ldi r24 ,39
 ldi r25 ,0
 rcall wait_usec
 ret

;ρουτίνα αποστολής ενός byte στην LCD
write_2_nibbles:
 push r24
 in r25 ,PIND
 andi r25 ,0x0f
 andi r24 ,0xf0
 add r24 ,r25
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 pop r24
 swap r24
 andi r24 ,0xf0
 add r24 ,r25
 out PORTD ,r24
 sbi PORTD ,PD3
 cbi PORTD ,PD3
 ret

;Ρουτίνες χρονοκαθυστέρησης
wait_msec:
 push r24
 push r25
 ldi r24 , low(998)
 ldi r25 , high(998)
 rcall wait_usec
 pop r25
 pop r24
 sbiw r24 , 1
 brne wait_msec
 ret

wait_usec:
 sbiw r24 ,1
 nop
 nop
 nop
 nop
 brne wait_usec
 ret
```

Πρόγραμμα σε C

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
void lcd_init (void);
void write_2_nibbles(char);
void lcd_data(char);
void lcd_command(char);
void main (void){
	DDRD=0xfc;
	DDRB=0x00;
	char min_dec='0',min_mon='0',sec_dec='0',sec_mon='0';
	while(1){
		while((PINB & 0x01)==0){
			if((PINB & 0x80)==128){
				min_dec='0',min_mon='0',sec_dec='0',sec_mon='0';
				lcd_init();                          //Initiating LCD
				lcd_data(min_dec);            //Print min_dec
				lcd_data(min_mon);            //Print min_mon
				lcd_data('M');                //Print 'M'
				lcd_data('I');                //Print 'I'
				lcd_data('N');                //Print 'N'
				lcd_data(':');                //Print ':'
				lcd_data(sec_dec);            //Print sec_dec
				lcd_data(sec_mon);            //Print sec_mon
				lcd_data('S');                //Print 'S'
				lcd_data('E');                //Print 'E'
				lcd_data('C');                //Print 'C'
			}
		}
		lcd_init();                          //Initiating LCD
		lcd_data(min_dec);            //Print min_dec
		lcd_data(min_mon);            //Print min_mon
		lcd_data('M');                //Print 'M'
		lcd_data('I');                //Print 'I'
		lcd_data('N');                //Print 'N'
		lcd_data(':');                //Print ':'
		lcd_data(sec_dec);            //Print sec_dec
		lcd_data(sec_mon);            //Print sec_mon
		lcd_data('S');                //Print 'S'
		lcd_data('E');                //Print 'E'
		lcd_data('C');                //Print 'C'
		_delay_ms(1000);              //delay 1sec
		if(sec_mon=='9'){
			if(sec_dec=='5'){
				if(min_mon=='9'){
					if(min_dec=='5'){
						min_dec='0';
						min_mon='0';
						sec_dec='0';
						sec_mon='0';
					}
					else{
						min_dec++;
						min_mon='0';
						sec_dec='0';
						sec_mon='0';
					}
				}
```

```
                                else{
                                        min_mon++;
                                        sec_mon='0';
                                        sec_dec='0';
                                }
                        }
                        else{
                                sec_dec++;
                                sec_mon='0';
                        }
                }
                else{
                        sec_mon++;
                }
        }
}
void write_2_nibbles(char x){
        char x2;
        char y=PIND & 0x0f;
        char x1=x & 0xf0;
        x1=x1+y;
        PORTD=x1;
        PORTD=PORTD | 0x08;
        PORTD=PORTD & 0xf7;
        x2=x<<4 ;
        x=x>>4;
        x=x|x2;
        x=x & 0xf0;
        PORTD=x+y;
        PORTD=PORTD | 0x08;
        PORTD=PORTD & 0xf7;
}
void lcd_data(char x){
        PORTD=PORTD | 0x04;
        write_2_nibbles(x);
        _delay_us(43);
}
void  lcd_command(char x){
        PORTD=PORTD & (0xfb);
        write_2_nibbles(x);
        _delay_us(39);
}
void lcd_init (void){
        _delay_ms(40);
        PORTD=0x30;
        PORTD=PORTD | 0x08;
        PORTD=PORTD & 0xf7;
        _delay_us(39);
        PORTD=0x30;
        PORTD=PORTD | 0x08;
        PORTD=PORTD & 0xf7;
        _delay_us(39);
        PORTD=0x20;
        PORTD=PORTD | 0x08;
        PORTD=PORTD & 0xf7;
        _delay_us(39);
        lcd_command(0x28);
        lcd_command(0x0c);
        lcd_command(0x01);
        _delay_us(1530);
        lcd_command(0x06);}
```