

# Εντολές ελέγχου ροής και διακλάδωσης

Πρόκειται για εντολές άλματος, παράκαμψης, διακλάδωσης και κλήσης ρουτινών. Δηλ. αλλάζει η κανονική ακολουθιακή ροή του προγράμματος. Οι εντολές παράκαμψης και διακλάδωσης είναι υπό συνθήκη.

εντολές διακλάδωσης & παράκαμψης			
<b>BRBS/BRBC</b>	s,k	Branch if Status Flag 1/0	If (SREG(s)=1) then $PC \leftarrow PC + k + 1$
<b>BREQ/BRNE</b>	k	Branch if Equal/ Not Equal	if ( $Z = 1$ ) then $PC \leftarrow PC + k + 1$
<b>BRCS/BRCC</b>	k	Branch if Carry 1/0	if ( $C = 1$ ) then $PC \leftarrow PC + k + 1$
<b>BRSH/BRLO</b>	k	Branch if Same or $>$ / $<$	if ( $C = 0$ ) then $PC \leftarrow PC + k + 1$
<b>BRGE/BRLT</b>	k	Branch if $>$ or $=$ , Signed	if ( $NV = 0$ ) then $PC \leftarrow PC + k + 1$
<b>BRHS/BRHC</b>	k	Branch if Half_Carry 1/0	if ( $H = 1$ ) then $PC \leftarrow PC + k + 1$
<b>BRTS/ BRTC</b>	k	Branch if T Flag 1/0	if ( $T = 1$ ) then $PC \leftarrow PC + k + 1$
<b>BRVS/BRVC</b>	k	Branch if Overflow Flag 1/0	if ( $V = 1$ ) then $PC \leftarrow PC + k + 1$
<b>BRIE/ BRID</b>	k	Branch if Int 1/0 Disabled	if ( $I = 1$ ) then $PC \leftarrow PC + k + 1$
<b>SBRC/SBRS</b>	Rr, b	Skip if Bit in Reg 0/1	if ( $Rr(b) = 0$ ) $PC \leftarrow PC + 2$ or 3
<b>SBIC/ SBIS</b>	P, b	Skip if Bit in I/O Reg 0/1	if ( $P(b) = 0$ ) $PC \leftarrow PC + 2$ or 3

# Εντολές άλματος-ρουτινών & σύγκρισης

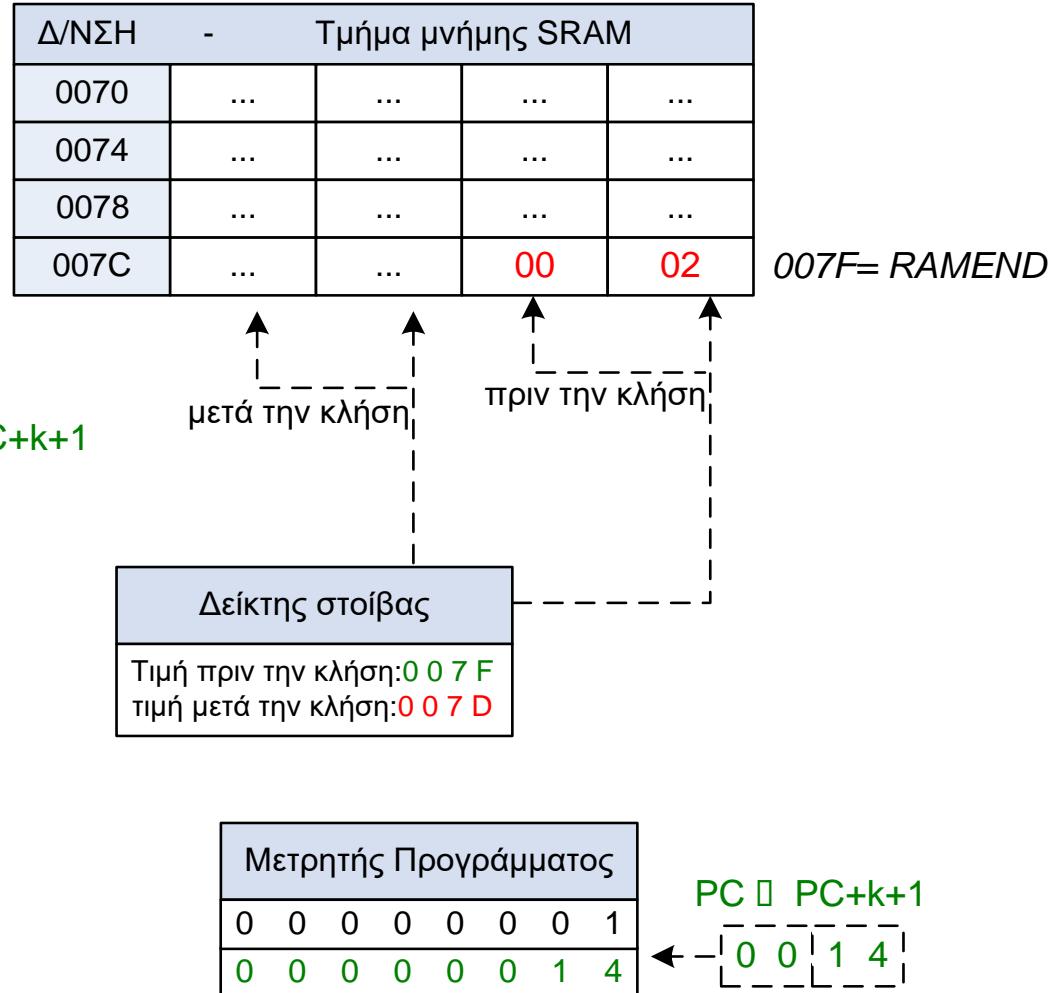
Παρέχουν τη δυνατότητα να εκτελέσουμε άλμα σε επιθυμητή διεύθυνση, να καλέσουμε μια ρουτίνα και να επιστρέψουμε από αυτήν. Οι εντολές σύγκρισης παρέχουν τη δυνατότητα να συγκρίνουμε δύο καταχωρητές ή έναν καταχωρητή και μια σταθερά.

<b>RJMP</b>	k	Relative Jump, W(1), C(2), - 2K +1 to 2K	$PC \leftarrow PC + k+1$
<b>IJMP</b>		Indirect Jump to (Z), W(1), C(2), 64K	$PC \leftarrow Z$
<b>JMP</b>	k	Direct Jump, W(2), C(3), 4M	$PC \leftarrow k$
<b>RCALL</b>	k	Relative Subroutine Call	$PC \leftarrow PC+k+1$
<b>ICALL</b>		Indirect Call to (Z) - Η διεύθυνση επιστροφής (PC) αποθηκεύεται στη στοίβα.	$PC \leftarrow Z$ $STACK \leftarrow PC$
<b>CALL</b>	k	Direct Subroutine Call - Η διεύθυνση επιστροφής (PC) αποθηκεύεται στη στοίβα.	$PC \leftarrow k,$ $STACK \leftarrow PC$
<b>RET</b>		Subroutine Return	$PC \leftarrow STACK$
<b>RETI</b>		Interrupt Return, I flag is set	$PC \leftarrow STACK$
<b>CP</b>	Rd, Rr	Compare	$Rd < Rr$
<b>CPI</b>	Rd, K	Compare Reg with Immediate	$Rd < K$

# Παράδειγμα áλματος-ρουτινών

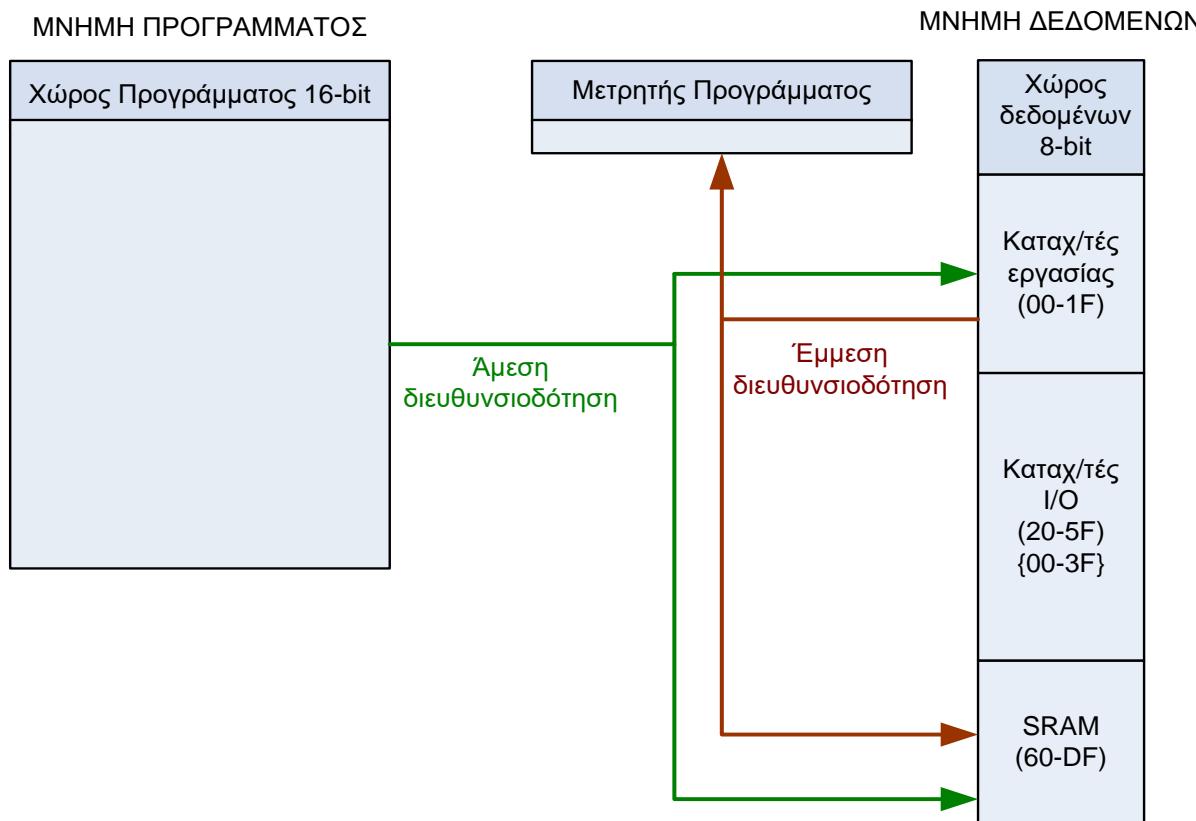
PC	λειτουργικός κώδικας 16-bit
000	.....
001	rcall label
002	nop
003	nop
004	.....
...	.....
014	Label:      nop
015	ret

κλήση ρουτίνας



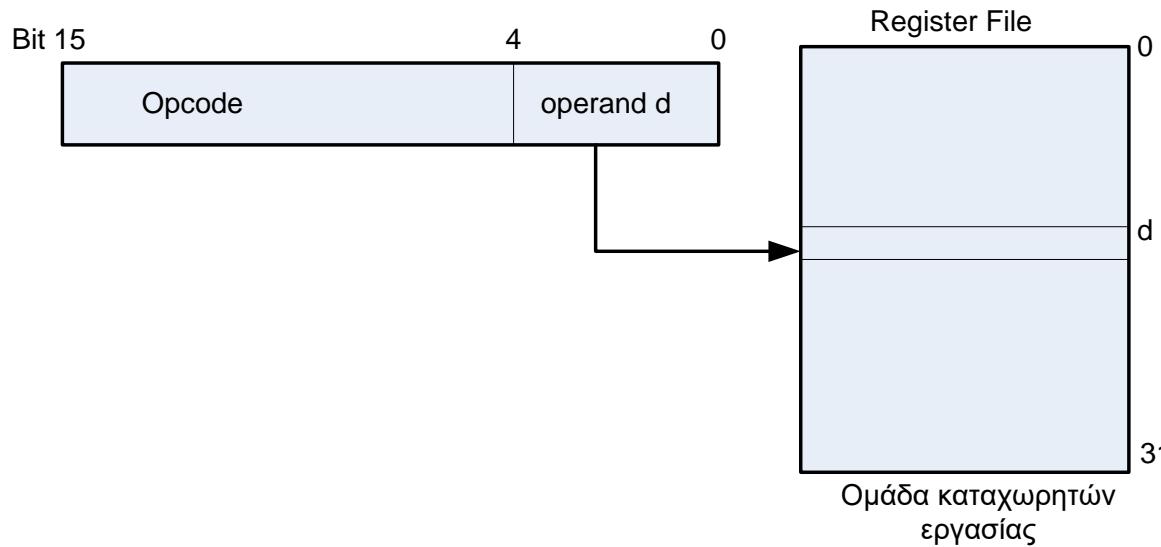
# Διευθυνσιοδότηση μΕ AVR

Επιπλέον οι εντολές ταξινομούνται με βάση τον τρόπο που επιτυγχάνεται πρόσβαση στα δεδομένα και τις πράξεις που εκτελούνται με αυτά.  
Πρόκειται για τους τρόπους διευθυνσιοδότησης του προγράμματος και των δεδομένων. Η διευθυνσιοδότηση διακρίνεται σε άμεση και έμμεση:



# Άμεση διευθυνσιοδότηση ενός καταχωρητή

Η εντολή διαβάζει τα περιεχόμενα του καταχωρητή, εκτελεί τις πράξεις με αυτά και αποθηκεύει το αποτέλεσμα στον ίδιο καταχωρητή. Ο καταχωρητής είναι ένας από τους 32 καταχωρητές εργασίας R0-R31 που διαθέτουν το αντίστοιχο τμήμα μνήμης (register file).

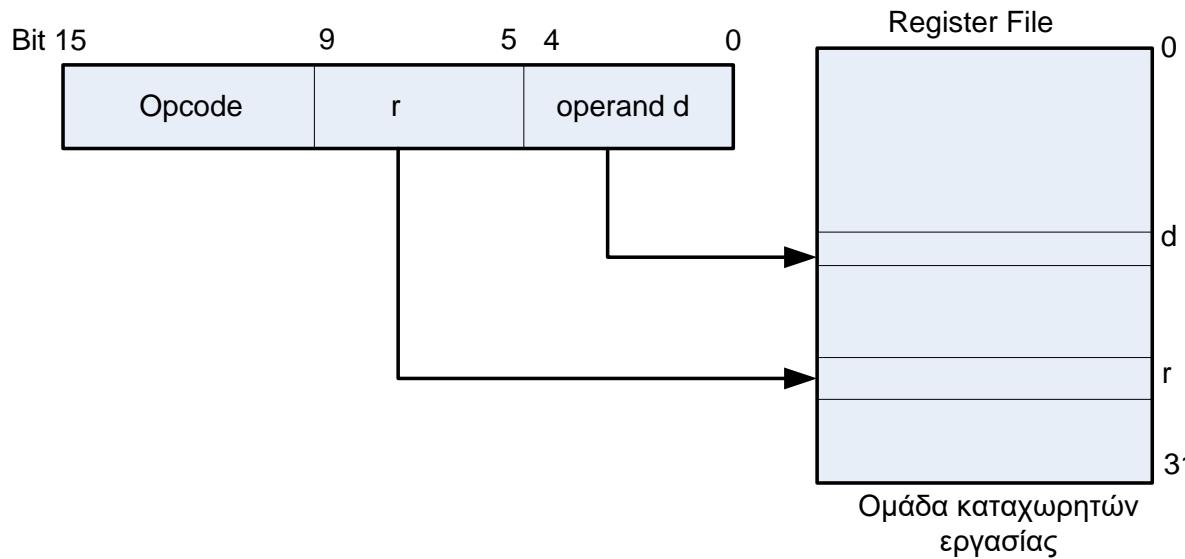


Παραδείγματα τέτοιων εντολών είναι:

**INC Rd:** Αύξηση των περιεχομένων του καταχωρητή Rd κατά μία μονάδα.

# Άμεση διευθυνσιοδότηση δύο καταχωρητών

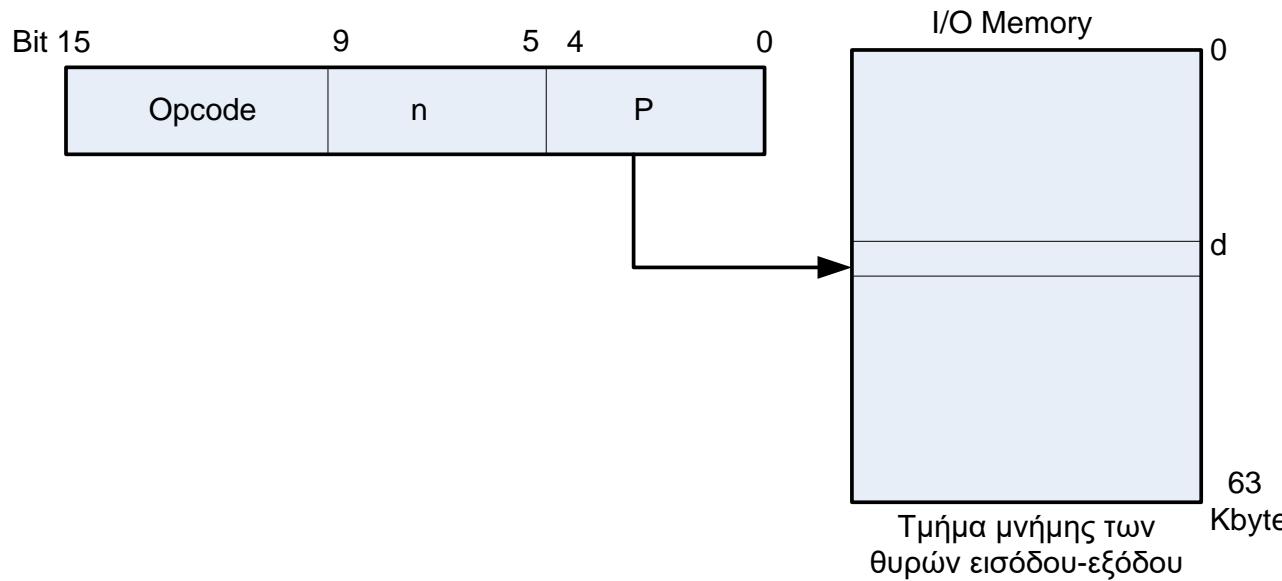
Η εντολή διαβάζει τα περιεχόμενα των δυο καταχωρητών, εκτελεί την πράξη μεταξύ των δεδομένων και αποθηκεύει το αποτέλεσμα στον καταχωρητή προορισμού Rd.



Παραδείγματα τέτοιων εντολών είναι: **ADD Rd,Rr** και **MOV Rd,Rr**

# Άμεση διευθ/τηση των θυρών εισόδου-εξόδου

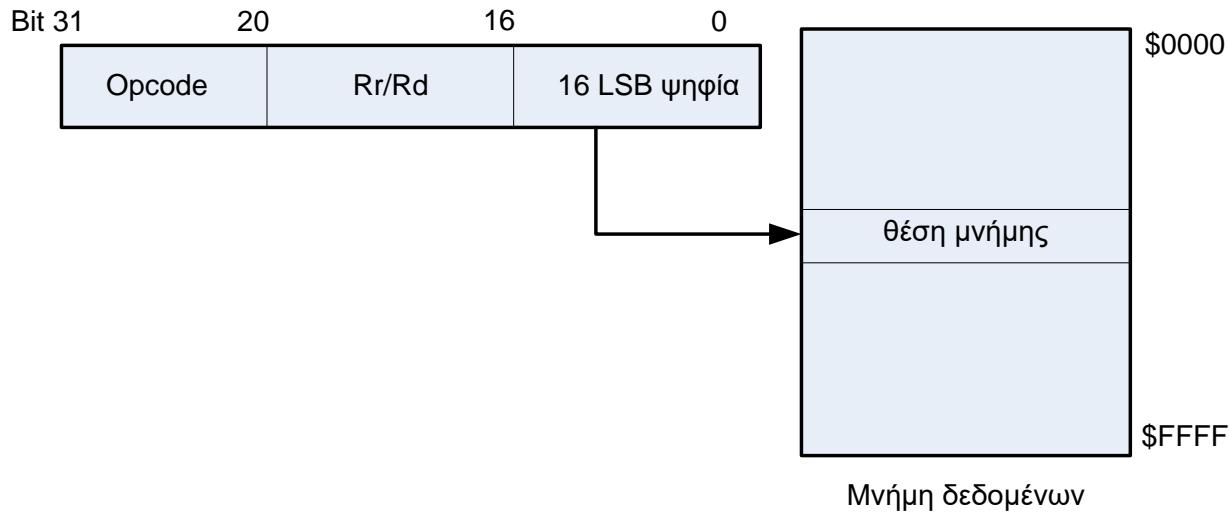
Η πρόσβαση στις θέσεις μνήμης που αντιστοιχούν οι θύρες εισόδου-εξόδου (I/O memory) επιτυγχάνεται με 2 εντολές: της IN Rd, PortAddress και της OUT PortAddress, Rr όπου PortAddress είναι η διεύθυνση του καταχωρητή εισόδου-εξόδου.



Παραδείγματα τέτοιων εντολών είναι: IN Rd, PINB και OUT PORTD, Rr

# Άμεση διεύθυνσιοδότηση μνήμης δεδομένων

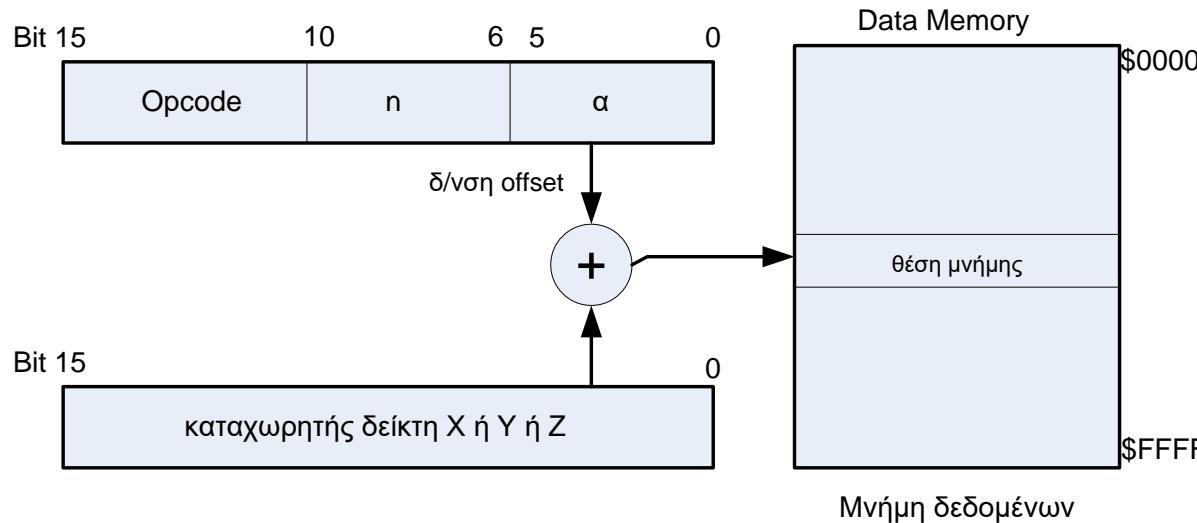
Η πρόσβαση στις θέσεις μνήμης επιτυγχάνεται με εντολές μήκους 2 λέξεων, όπου η μία λέξη (16 bits) αντιστοιχεί στη διεύθυνση της μνήμης δεδομένων. Άρα, ο χώρος μνήμης όπου επιτυγχάνεται πρόσβαση είναι 64Kbyte.



Παραδείγματα τέτοιων εντολών: **STS K,Rr / LDS Rd,K** όπου **K** διεύθυνση 16 bits

# Έμμεση διεύθυνσιο δότηση μνήμης δεδομένων

Οι εντολές αυτές έχουν μήκος 1 λέξης και χρησιμοποιούν έναν κατ/τή δείκτη ( $X, Y, Z$ ), το περιεχόμενο του οποίου αντιστοιχεί στη διεύθυνση της μνήμης δεδομένων. Ο κατ/τής δείκτη  $X, Y, Z$  μπορεί να αυξηθεί κατά μία μονάδα μετά την ανάθεση της τιμής στον κατ/τη  $Rr$ , ώστε να δείχνει σε επόμενη θέση μνήμης ή να μειωθεί κατά μία μονάδα πριν την ανάθεση της τιμής στον κατ/τη  $Rr$ , ώστε να δείχνει σε προηγούμενη θέση μνήμης ή να έχουμε έμμεση φόρτωση του κατ/τή  $Rd$  με τα περιεχόμενα της θέσης μνήμης ( $Y+q$ ), ώστε να επιτύχουμε πρόσβαση σε παραπέρα θέση μνήμης. Πρόκειται δηλαδή για μετατόπιση.

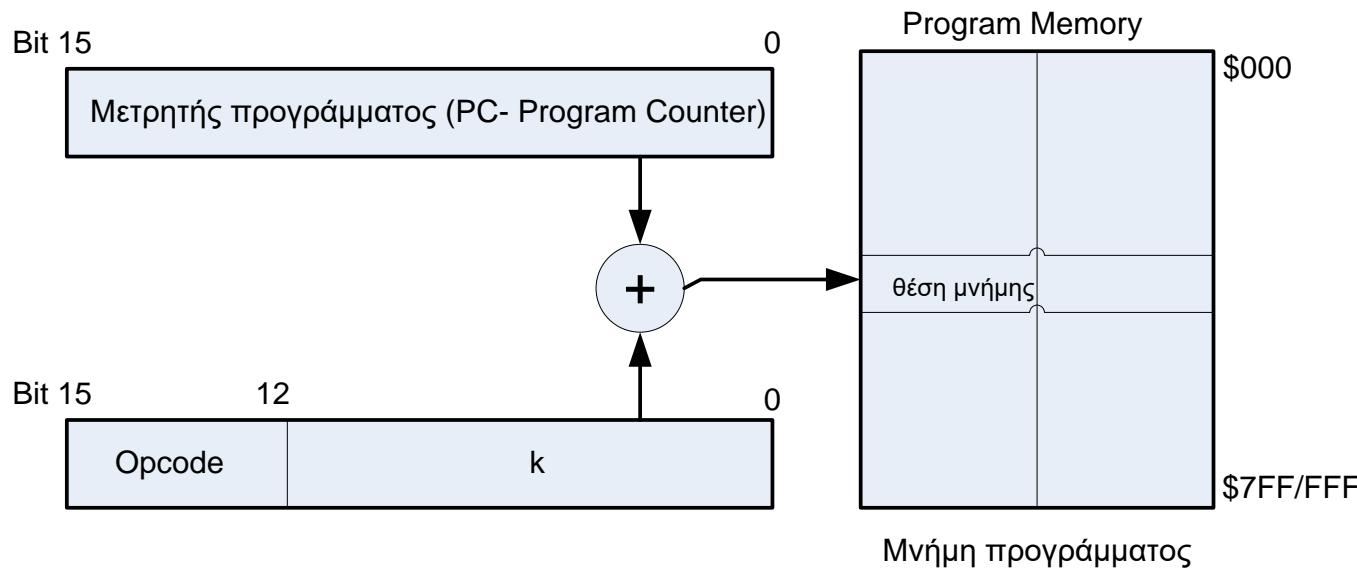


Παραδείγματα :**ST X,Rr**   **LD Rr,Y**   **LD Rr,X+**  
**LDD Rd,Y+2**   **ST Y+6,Rd**

**ST X+,Rr**   **LD Rr,-Z**   **ST -X,Rd**

# Σχετική διευθυνσιοδότηση μνήμης προγράμματος

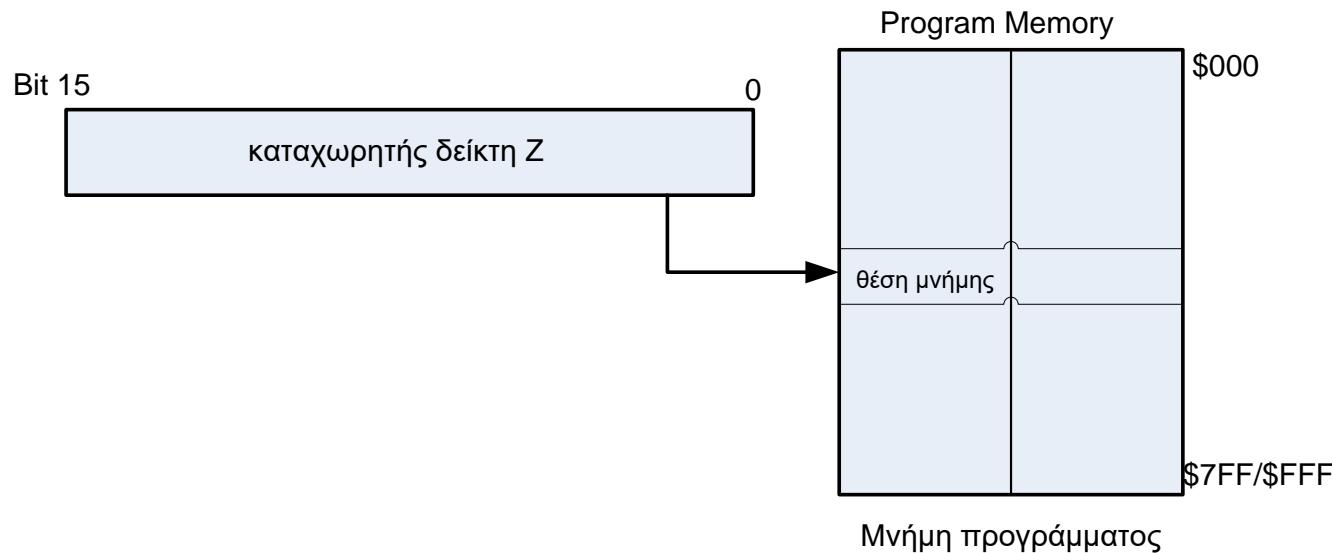
Οι εντολές αυτές επιτυγχάνουν πρόσβαση σε χώρο μνήμης προγράμματος και είναι του τύπου RCALL, RJMP όπου χρησιμοποιείται μια μετατόπιση +/- 2K στο περιεχόμενο του μετρητή προγράμματος.



Παραδείγματα : **RCALL, RJMP**

# Έμμεση διευθυνσιοδότηση μνήμης προγράμματος

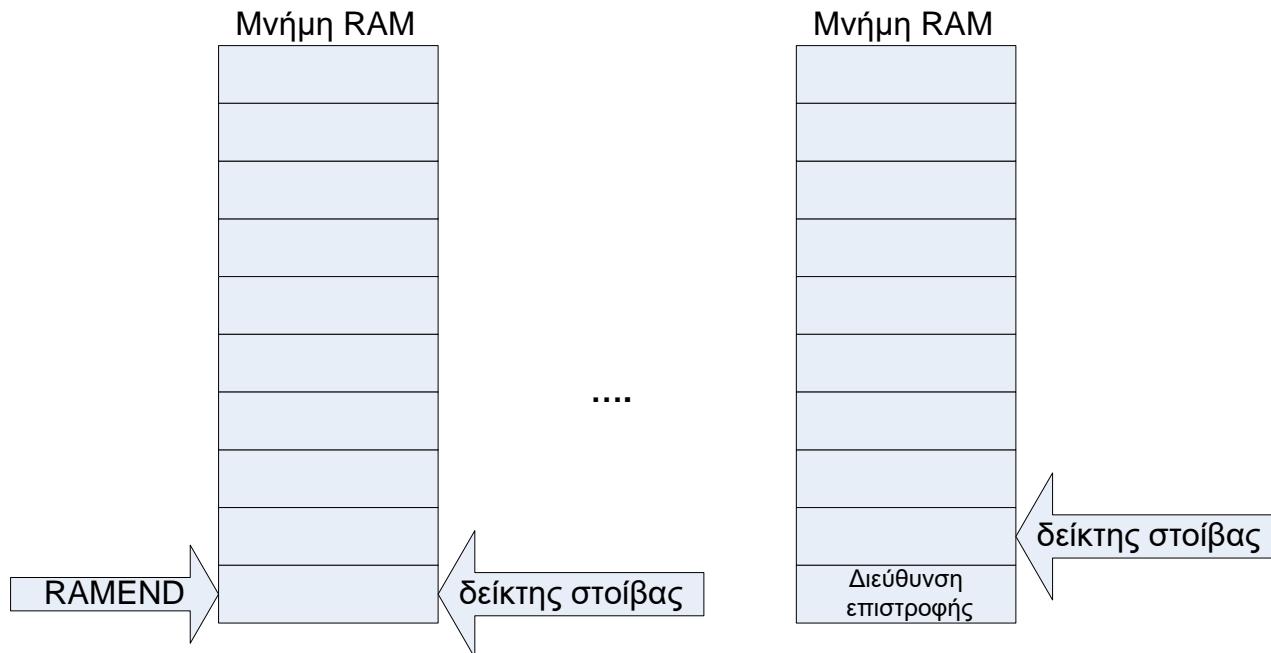
Οι εντολές αυτές επιτυγχάνουν πρόσβαση σε χώρο μνήμης προγράμματος έως 64Kbytes με χρήση του καταχωρητή Z, ως δείκτη μιας θέσης μνήμης προγράμματος.



Παραδείγματα : ICALL , IJMP

# Στοίβα

Η στοίβα χρησιμοποιείται από την αριθμητική λογική μονάδα (ALU) για αποθήκευση διευθύνσεων επιστροφής από ρουτίνες διακοπής-υπορουτίνες. Η στοίβα χρειάζεται έναν δείκτη στοίβας (SP) και χώρο μνήμης στην SRAM. Πρόκειται, λοιπόν, για δομή τύπου Last-In-First-Out (LIFO). Μια διεύθυνση SRAM έχει μήκος 16 bits, οπότε ο καταχωρητής SPL κρατά τα 8 LSB bits και ο SPH τα 8 MSB bits της διεύθυνσης.

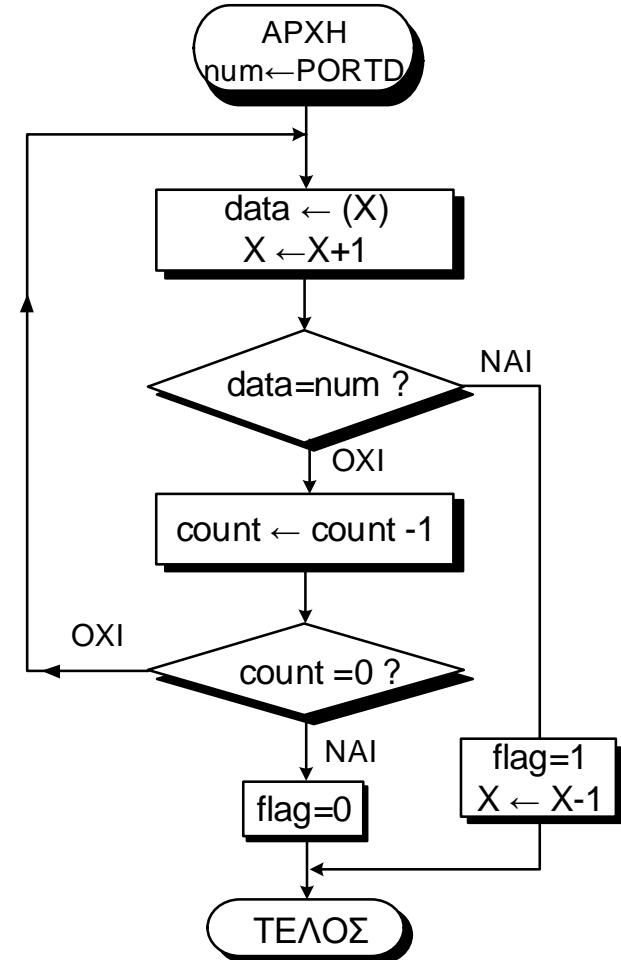


Ο SP τίθεται στη τελευταία θέση μνήμης (RAMEND) κατά την αρχικοποίηση

# Παράδειγμα 4° : Εντολές ελέγχου ροής του προγράμματος και διακλάδωσης

Το πρόγραμμα βρίσκει αν η τιμή στον καταχωρητή r20 (num) που δίνεται από το PORTD είναι σε περιοχή της μνήμης δεδομένων SRAM που η πρώτη διεύθυνση βρίσκεται στον καταχωρητή X= r27:r26 και το πλήθος τους, στον καταχωρητή r21 (count).

Αν βρει τη τιμή, θέτει στον καταχωρητή r22 τιμή 1 και τη διεύθυνση που βρήκε το δεδομένο (μέσω του καταχωρητή δείκτη X), αλλιώς θέτει 0 στον καταχωρητή r22 (flag).

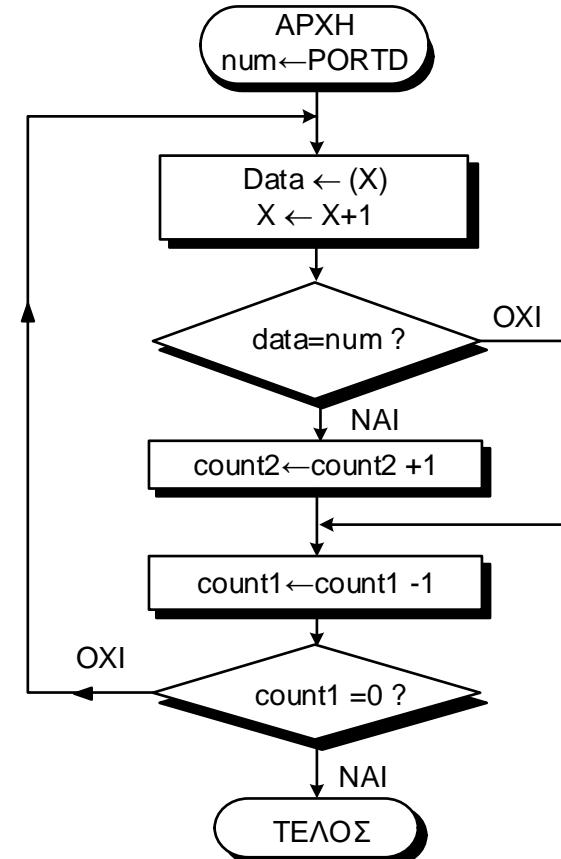


Το πρόγραμμα βρίσκει αν η τιμή που δίνεται από τον χρήστη από το PORTD είναι σε περιοχή της μνήμης προγράμματος. Αν ναι, επιστρέφει τη διεύθυνση, αλλιώς θέτει 0 στον r22.

```
.INCLUDE "m16def.inc"
.DEF flag=r22
.DEF count=r21
.DEF num=r20
.DEF data=r19
start: clr num          ; το PORTD ορίζεται ως είσοδος
       out DDRD, num
       in num,PIND      ; ανάγνωση αριθμού από PORTD
loop: ldi data,x+        ; φόρτωση θέσης X μνήμης δεδομένων και αύξηση
      cp data,num        ; δείκτη για πρόσβαση στον επόμενο κύκλο σε επόμενη θέση
      breq found         ; σύγκριση του εισαγόμενου αριθμού με θέση μνήμης
      ; η τιμή βρέθηκε στον πίνακα
not_found:
      dec count           ; μειώνω τον μετρητή
      brne loop            ; έλεγχος για να μην περάσουμε το δοσμένο πλήθος
      clr flag             ; αν η τιμή δεν βρέθηκε μηδενίζουμε τον r22
      rjmp end
found:
      ldi flag,1           ; η τιμή βρέθηκε στον πίνακα οπότε
      sbiw r26,1            ; θέτουμε τον r22 και επιστρέφουμε την αντίστοιχη
      ; διεύθυνση στον X= r27 : r26 αφού το μειώσουμε κατά 1
end:
```

# Παράδειγμα 5<sup>ο</sup> : Αναζήτηση ενός χαρακτήρα

Να υπολογιστεί πόσες φορές εμφανίζεται ένας χαρακτήρας, που η τιμή του δίνεται από το PORTD, σε περιοχή της μνήμης δεδομένων SRAM που η πρώτη διεύθυνση βρίσκεται στον καταχωρητή  $X=r27:r26$  και το πλήθος τους, στον καταχωρητή r21 (count1). Το πλήθος των εμφανίσεων του χαρακτήρα επιστρέφεται μέσω του καταχωρητή r22 (count2).

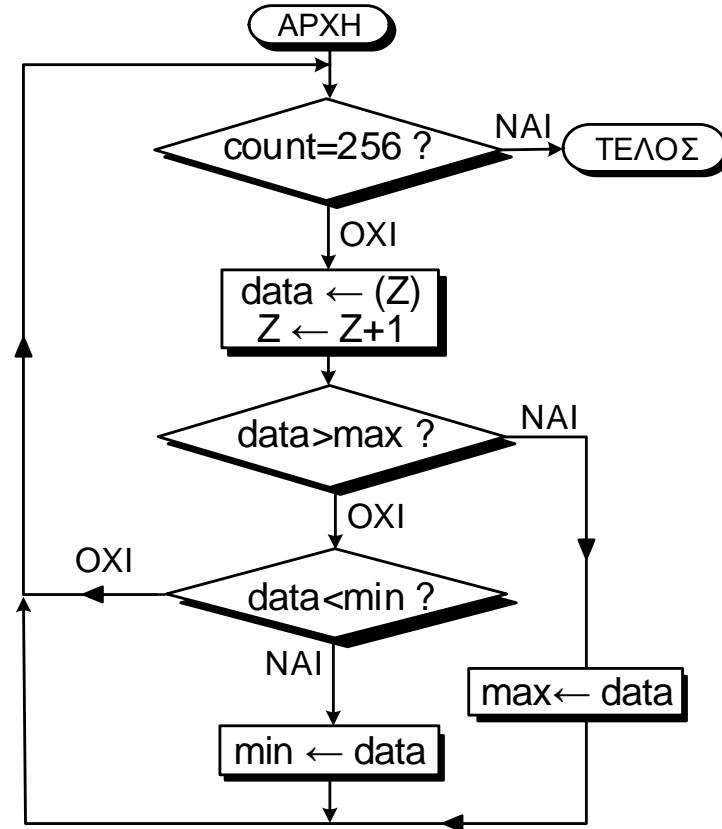


# Παράδειγμα 5<sup>ο</sup> - συνέχεια

```
.INCLUDE "m16def.inc"
.DEF count2=r22
.DEF count1=r21
.DEF num=r20
.DEF data=r19
start: clr num          ; το PORTD ορίζεται ως είσοδος
       out DDRD, num
       in num, PIND      ; ανάγνωση αριθμού από PORTD
loop:
       ld data,x+        ; φόρτωση θέσης X μνήμης δεδομένων και
; αύξηση δείκτη (X←X+1) για πρόσβαση στον επόμενο κύκλο σε επόμενη θέση
       cp data, num      ; σύγκριση του εισαγόμενου αριθμού με θέση μνήμης
       brne not_found    ; η τιμή δεν βρέθηκε στον πίνακα
       inc count2
not_found:
       dec count1         ; μειώνω τον μετρητή
       brne loop          ; έλεγχος για να μην περάσουμε το δοσμένο πλήθος
end:
```

# Παράδειγμα 6<sup>ο</sup>: Πρόγραμμα εντοπισμού μέγιστου και ελάχιστου

Το πρόγραμμα εντοπίζει τον μέγιστο και τον ελάχιστο σε περιοχή της μνήμης προγράμματος όπου έχουν αποθηκευθεί συνολικά 256 δεδομένα (των 8 bit).



# Παράδειγμα 6<sup>ο</sup> - συνέχεια

```
.INCLUDE "m16def.inc"
```

```
.DEF count=R18
```

```
.DEF data=R17
```

```
.DEF max=R16
```

```
.DEF min=R15
```

begin:

```
ldi ZH,HIGH(Array *2)
```

```
ldi ZL,LOW(Array *2)
```

; η διεύθυνση του πίνακα στον Z

```
lpm data,z+
```

; φόρτωση μνήμης προγράμματος

```
mov max, data
```

```
mov min, data
```

```
clr count
```

; το count γίνεται 0

search:

```
inc count
```

; Έλεγχος αν μηδενίστηκε ο count

```
breq end
```

; Συνολικά 255 φορές θα εκτελεστεί

; ο βρόχος που ακολουθεί

# Παράδειγμα 6<sup>ο</sup> - συνέχεια

```
lpm data,z+          ; πρόσβαση σε επόμενη θέση και αύξηση δείκτη Z
cp data,max          ; σύγκριση με μέγιστο
brge new_max          ; άλμα εφόσον βρεθεί μεγαλύτερη τιμή
cp data,min          ; σύγκριση με ελάχιστο
brlo new_min          ; άλμα εφόσον βρεθεί μικρότερη τιμή
rjmp search           ; επιστροφή στην αναζήτηση

new_max:
    mov max, data      ; σώσιμο στο max της νέας μέγιστης τιμής
    rjmp search         ; επιστροφή στην αναζήτηση

new_min:
    mov min, data      ; σώσιμο στο min της νέας ελάχιστης τιμής
    rjmp search         ; επιστροφή στην αναζήτηση

end:

Array:    ; εισαγωγή πίνακα δεδομένων στη μνήμη προγράμματος
.DW 0x0908,0x0706,0x1713,0x3326
.DW 0x3042,0x7061,0x7205,0x7803
```

## Παράδειγμα 7ο: Διάβασμα/Εγγραφή των Ports

Στο παράδειγμα αυτό διαβάζονται διακόπτες που είναι συνδεδεμένοι στο PORTD και ανάβουν leds που είναι συνδεδεμένα στο PORTB ως με την εξής λογική:

Αν ο διακόπτης sw0 είναι ON τότε ανάβει το led0.

Αν ο διακόπτης sw1 είναι ON τότε ανάβουν τα led0 και led1.

Αν ο διακόπτης sw2 είναι ON τότε ανάβουν τα led0, led1 και led2.

...

Αν ο διακόπτης sw7 είναι ON τότε ανάβουν τα led0, led1, led2 , led , led , led , led .

Μόνο ένας διακόπτης είναι ON κάθε φορά.

Η υλοποίηση του κώδικα βασίζεται στη δημιουργία ενός πίνακα στη μνήμη προγράμματος (code segment) που περιέχει τα δεδομένα για την έξοδο PORTB .

(Ενναλακτικά η έξοδος PORTB θα μπορούσε να υπολογιστεί με απλούστερο τρόπο από τη σχέση  $led=2^{*}sw-1$ )

# Παράδειγμα 7ο - συνέχεια

```
.INCLUDE "m16def.inc"      ; δήλωση μικροελεγκτή
.LIST                      ; ενεργοποιεί την εμφάνιση του listing του κώδικα κατά τη μεταγλώττιση
.DEF reg=R0                ; η εντολή LPM χειρίζεται τον καταχωρητή R0
.DEF temp=R16
```

table:

- ; Πίνακας με τους συνδυασμούς των leds(κάθε byte αντιστοιχεί σε ένα συνδυασμό).
- ; Για τα led υποθέτουμε ότι ισχύει αρνητική λογική(είναι αναμμένα με έξοδο 0)
- ; Η χρήση του .DB xx στη μνήμη προγράμματος πάντα προσθέτει ένα μηδενικό byte.
- ; Προτιμάται η χρήση του .DW xxxy όπου 2 byte ενώνονται σε μια λέξη.
- ; Όταν προσθέτουμε μια λέξη όπως FEFFh σε πίνακα, το 1o byte του πίνακα που ; λαμβάνεται μέσω της εντολής LPM είναι το LSB (FFh), και όχι το MSB (FEh).

.DW 0xFEFF	; 1 Led, 0 Leds
.DW 0xF8FC	; 3 Leds, 2 Leds
.DW 0xE0F0	; 5 Leds, 4 Leds
.DW 0x80C0	; 7 Leds, 6 Leds
.DW 0x0000	; 8 Leds, 8 Leds

- ; Η πρόσβαση στον πίνακα μέσω της εντολής lpm γίνεται κατά byte, ενώ οι διευθύνσεις ; οργανώνονται κατά λέξεις
- .EQU array=table\*2

# Παράδειγμα 7ο - συνέχεια

start:

```
clr temp ; PORTD ως είσοδος των διακοπτών  
out DDRD,temp  
dec temp ; φόρτωση τιμής 0xFF στον καταχωρητή temp  
out DDRB,temp ; PORTB ως έξοδος των Leds  
out PORTD,temp ; ενεργοποίηση εσωτερικών αντιστάσεων πρόσδεσης
```

loop:

```
; χρήση καταχωρητή δείκτη Z(ZL(R30) και ZH(R31)) που έχει οριστεί στο αρχείο m16def.inc  
ldi ZL,LOW(array) ; ο Z δείχνει στο 1o byte (FF) του πίνακα.  
ldi ZH,HIGH(array)  
in temp,PIND ; ανάγνωση διακοπτών  
cpi temp,0xFF ; αν όλοι οι διακόπτες είναι off(αρνητική λογική διακοπτών), τότε  
breq diavasma ; άλμα στο diavasma (πρώτη θέση του πίνακα - όλα τα Leds off)
```

inc\_pointer:

```
adiw ZL, 1 ; Αυξάνεται το Z μέχρι να βρεθεί η πρώτη μονάδα  
ror temp ; καθώς ολισθαίνει δεξιά η τιμή των διακοπτών.  
brlo inc_pointer ; Όσο το C=1 έχουμε άλμα και αύξηση του Z (ενναλακτικά brcs) .
```

diavasma:

```
lpm ; Το περιεχόμενο της θέσης που δείχνει ο Z εγγράφεται στον reg(R0).  
out PORTB,reg ; Μεταφορά του δεδομένου στα leds.  
rjmp loop
```

# Παράδειγμα 8ο: Χειρισμός leds με χρονική καθυστέρηση

.INCLUDE "m16def.inc" ; δηλώνουμε μικροελεγκτή

.DEF reg = R16

.def Delay = r17 ; καταχωρητής μεταβλητής Delay

.def Delay2 = r18 ; καταχωρητής μεταβλητής Delay2

main:

ldi reg,0b11111111

out DDRB,reg ; ορίζουμε το PORTB ως έξοδο

DLY:

dec Delay ; Καθυστέρηση για να γίνουν ορατές οι αλλαγές

brne DLY

dec Delay2

brne DLY

ldi reg,0x00 ; (0=on, 1=off).

out PORTB,reg ; άναμμα των leds

DLY2:

dec Delay ; Καθυστέρηση για να γίνουν ορατές οι αλλαγές

brne DLY2

dec Delay2

brne DLY2

ldi reg,0xFF

out PORTB,reg ; σβήσιμο των leds και ατέρμων βρόχος

rjmp main ; για συνεχή επανάληψη της διαδικασίας

# Παράδειγμα 9°: Χειρισμός στα leds με βάση τη τιμή των switches

Στο παράδειγμα γίνεται χειρισμός στα leds που συνδέονται στη θύρα PORTB με βάση τη τιμή των switches που συνδέονται στις θύρα PORTD.

Η συμπεριφορά των leds καθορίζεται με βάση ποιος διακόπτης πιέζεται σύμφωνα με τον παρακάτω πίνακα:

PORTD-switches	PORTB - LEDs
pin0=1	Μέτρηση στα LEDS κάτω
pin1=1	Μέτρηση στα LEDS πάνω
pin2=1	Περιστροφή των LEDS μία θέση δεξιά
pin3=1	Περιστροφή των LEDS μία θέση αριστερά
pin4=1	Αντιστροφή LEDS
pin5=1	Συμπλήρωμα ως προς 2 των LEDS
pin6 =1	Εναλλαγή 4 LSB με τα 4 MSB
pin7= 1	Τα 4 LSB ON και τα 4 MSB OFF

- Δεν πιέζονται ταυτόχρονα περισσότεροι του ενός διακόπτες.
- Αρχική κατάσταση OFF για όλα τα LEDS.
- Αρνητική λογική για τα led.

# Παράδειγμα 9<sup>ο</sup> : Αρχικοποιήσεις

```
.include "m16def.inc"
.def Temp =r16           ; προσωρινός καταχωρητής
.def Delay =r17          ; καταχωρητής μεταβλητής Delay
.def Delay2 =r18         ; καταχωρητής μεταβλητής Delay2
```

RESET:

```
clr Temp                 ; αρχικοποίηση του PORTD
out DDRD,Temp            ; ως θύρας εισόδου
ser Temp                 ; αρχικοποίηση του PORTB
out DDRB,Temp            ; ως θύρας εξόδου
out PORTD,Temp           ; ενεργοποίηση των αντιστάσεων πρόσδεσης
```

# Παράδειγμα 9<sup>ο</sup> : Κύριο πρόγραμμα

LOOP:

sbic PIND, 0x00	; Av (Port D, pin0 = 1) τότε
inc Temp	; μέτρηση LEDS μία μονάδα προς τα κάτω
sbic PIND, 0x01	; Av (Port D, pin1 = 1) τότε
dec Temp	; μέτρηση LEDS μία μονάδα προς τα πάνω
sbic PIND, 0x02	; Av (Port D, pin2 = 1) τότε
ror Temp	; ολίσθηση LEDS μία θέση δεξιά
sbic PIND, 0x03	; Av (Port D, pin3 = 1) τότε
rol Temp	; ολίσθηση LEDS μία θέση αριστερά
sbic PIND, 0x04	; Av (Port D, pin4 = 1) τότε
com Temp	; αντιστροφή LEDS
sbic PIND, 0x05	; Av (Port D, pin5 = 1) τότε
neg Temp	; αντιστροφή LEDS και πρόσθεση 1
sbic PIND, 0x06	; Av (Port D, pin6 = 1) τότε
swap Temp	; εναλλαγή των τμημάτων high και low των LEDS
sbic PIND, 0x07	; Av (Port D, pin7 = 1) τότε
ldi Temp, 0xF0	; Τα 4 LSB ON και τα 4 MSB OFF
out PORTB, Temp	; ενημέρωση των LEDS

DLY:

dec Delay	; καθυστέρηση για να γίνουν ορατές οι αλλαγές
brne DLY	
dec Delay2	
brne DLY	
rjmp LOOP	; επανάληψη βρόχου

# Παράδειγμα 10<sup>ο</sup> : Χειρισμός διακοπτών και led

Με το πάτημα ενός διακόπτη (ή και περισσότερων) από τα 4 δεξιότερα switches (PORTD) ανάβει το αντίστοιχο led (PORTB) και όσο είναι πατημένο να το περιστρέψει κυκλικά προς τα δεξιά.

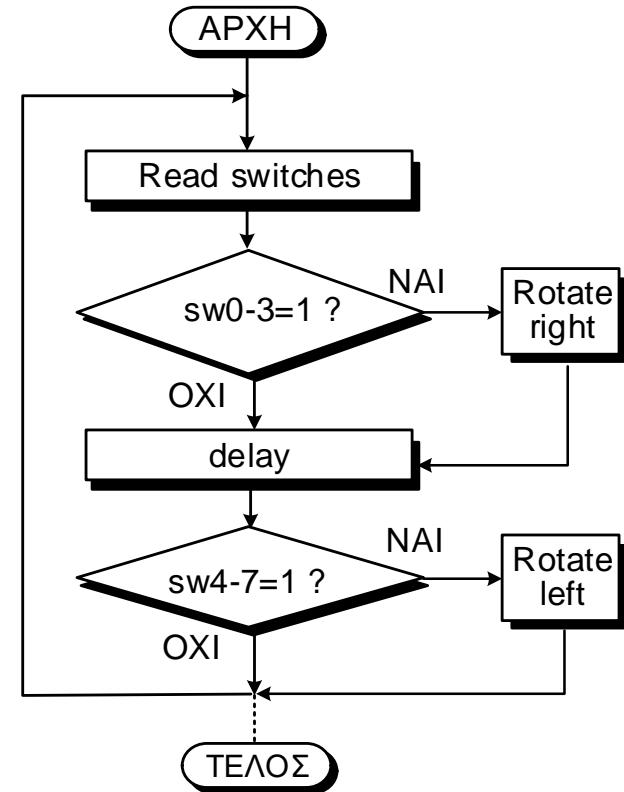
Αντίστοιχα, πάτημα ενός διακόπτη από τα 4 αριστερότερα να προκαλεί αριστερή περιστροφή.

Όταν δεν είναι πατημένο κανένα switch να σβήνουν όλα τα leds.

```
.include "m16def.inc"  
.def temp=r16  
.def tempN=r17  
.def Delay1=r18  
.def Delay2=r19
```

RESET:

```
ldi temp, LOW(RAMEND) ; αρχικοποίηση  
out SPL, temp ; δείκτη στοίβας  
ldi temp, HIGH(RAMEND)  
out SPH, temp  
clr temp  
out DDRD, temp ; DDRD ως θύρα εισόδου  
ser temp  
out DDRB, temp ; DDRB ως θύρα εξόδου  
out PORTD,temp ; τίθενται οι αντιστάσεις πρόσδεσης pull-up
```



# Παράδειγμα 10<sup>ο</sup> - συνέχεια

ARXH:

```
in temp, PIND      ; Αρχικά, οι διακόπτες είναι με αρνητική λογική  
mov tempN, temp    ; αντίγραφο των διακοπτών  
com tempN          ; σε θετική λογική στον καταχωρητή tempN
```

RIGHT:

```
andi tempN, 0x0F    ; Αν οποιοσδήποτε των διακοπτών sw0-3  
breq CONT           ; πατηθεί έχουμε ≠0 και δεν εκτελείται άλμα  
; Αν θέλουμε ο έλεγχος να γίνει απευθείας σε αρνητική τότε έχουμε τον κώδικα:  
; ori temp, 0xF0     ; Αν πατηθεί ένα από τα sw0-3 έχουμε  
; cpi temp, 0xFF     ; κάποιο 0 και η σύγκριση τότε δεν δίνει  
; breq CONT           ; ισότητα οπότε δεν εκτελείται άλμα  
ror temp             ; αλλά έχουμε περιστροφή δεξιά.  
out PORTB, temp
```

CONT:

```
rcall DELAY
```

LEFT:

```
andi tempN, 0xF0    ; Αν οποιοσδήποτε εκ των διακοπτών sw4-7 πατηθεί  
breq ARXH            ; έχουμε ≠0 και δεν εκτελείται άλμα  
rol temp              ; αλλά έχουμε περιστροφή αριστερά.  
out PORTB, temp
```

rjmp ARXH

DELAY:

```
dec Delay2            ; Ρουτίνα χρονοκαθυστέρησης όπου  
brne DELAY           ; οι 2 πρώτες εντολές εκτελούνται  
dec Delay1            ; Delay2* Delay1 φορές  
brne DELAY  
ret
```

# Παράδειγμα 11<sup>ο</sup> : Χειρισμός Θυρών I/O

Συνεχής ανάγνωση θύρας PortD.

Όταν διαπιστώνεται ότι άλλαξε η τιμή της, τότε κάθε νέα τιμή αποθηκεύεται σε διαδοχικές θέσεις μνήμης SRAM αρχίζοντας από τη διεύθυνση \$0060 ως την \$0150.

# Παράδειγμα 11<sup>ο</sup> - συνέχεια

```
.include "m16def.inc"

.def temp=r16           ; ορίζουμε ονόματα καταχωρητών
.def port_value=r17      ; καταχωρητής για προσωρινή αποθήκευση
.def reg = r18           ; αποθήκευση τιμής θύρας

.org 0x000
    rjmp main           ; παράκαμψη διανυσμάτων διακοπών
.org 0x100
main:
    ldi reg,LOW(RAMEND) ; κυρίως πρόγραμμα
    out SPL,reg          ; αρχικοποίηση δείκτη στοίβας
    ldi reg,HIGH(RAMEND)
    out SPH,reg
    clr temp
    out DDRD, temp       ; Port D ως είσοδος
    ser temp
    out PORTD, temp      ; Port D ( ενεργοποίηση pull-ups)
```

# Παράδειγμα 11<sup>ο</sup> - συνέχεια

clr xh	; δήλωση διεύθυνσης αποθήκευσης στην SRAM
ldi xl, 0x60	; θέτουμε διεύθυνση 0x60 στον καταχωρητή X
in port_value, PIND	; 1η ανάγνωση θύρας
diavasma:	
in temp,PIND	; βρόχος ανάγνωσης
cp temp,port_value	; διάβασμα τρέχουσας τιμής
breq diavasma	; έλεγχος εάν έχει αλλάξει η τιμή
	; ίδιες τιμές, επιστροφή στο diavasma για
nop	; ανάγνωση
st x+,temp	; αλλιώς αποθήκευση τιμής & αύξηση καταχωρητή X
mov port_value,temp	; ώστε να δείχνει στην επόμενη θέση μνήμης
nop	
cpi xl,0x51	; έλεγχος για να μην περάσουμε τη διεύθυνση 0x150
brne diavasma	
ldi xl,0x60	; αν τη φτάσουμε, αρχίζουμε αποθηκεύσεις πάλι από 0x60
clr xh	
rjmp diavasma	

# Παράδειγμα 12<sup>ο</sup>: Μεταφορά δεδομένων

Μεταφορά ενός πίνακα των 20 bytes από το τμήμα κώδικα (cseg) στο τμήμα δεδομένων (dseg).

Εξετάζεται κάθε στοιχείο του πίνακα και αν πρόκειται για περιττό αποθηκεύεται σε πίνακα στη διεύθυνση \$0060, ενώ αν πρόκειται για άρτιο σε πίνακα στη διεύθυνση \$0074 του τμήματος δεδομένων.

```
.INCLUDE "m16def.inc"
.DEF count=R18          ; Ορίζουμε όνομα καταχωρητή-μετρητή
.EQU Location1 = 0x0060 ; Διεύθυνση αποθήκευσης περιττών
.EQU Location2 = 0x0074 ; Διεύθυνση αποθήκευσης άρτιων
start:
    ldi count,21        ; Μετρητής για μεταφορά 20 στοιχείων
    ldi ZH, HIGH(Table*2) ; Φόρτωμα του Z με τη διεύθυνση του πίνακα
    ldi ZL,LOW(Table*2)  ; από το τμήμα κώδικα

    ldi XH,HIGH(Location1) ; Η διεύθυνση του πίνακα περιττών στον X
    ldi XL,LOW(Location1)

    ldi YH,HIGH(Location2) ; Η διεύθυνση του πίνακα άρτιων στον Y
    ldi YL,LOW(Location2)
```

# Παράδειγμα 12<sup>ο</sup> - συνέχεια

loop:

lpm r20,z+ ; Φόρτωση μνήμης προγράμματος και  
; αύξηση δείκτη πρόσβασης επόμενης θέσης.  
dec count ; Μείωση μετρητή και άλμα όταν  
; μεταφερθούν όλα τα στοιχεία.  
breq end ; Αν πρόκειται για περιπτώ (R20.0=1) skip  
sbrs r20,0 ; Άλλιώς άρτιος και άλμα στην ετικέτα even.  
rjmp even ; Ο αριθμός είναι περιπτώς.  
odd:  
ST X+, R20 ; Αποθήκευση περιπτών και  
rjmp loop ; επανάληψη με φόρτωση νέου αριθμού.

even:

ST Y+, R20 ; Αποθήκευση άρτιων και επανάληψη.  
rjmp loop ; επανάληψη με φόρτωση νέου αριθμού.

.cseg ; Επιλογή μνήμης κώδικα

Table: ; πίνακας δεδομένων.

.DW 0x0100,0x0706,0x1713,0x3326,0x27C6

.DW 0x5042,0x7A61,0xA2F1,0xE0D7,0x89FD

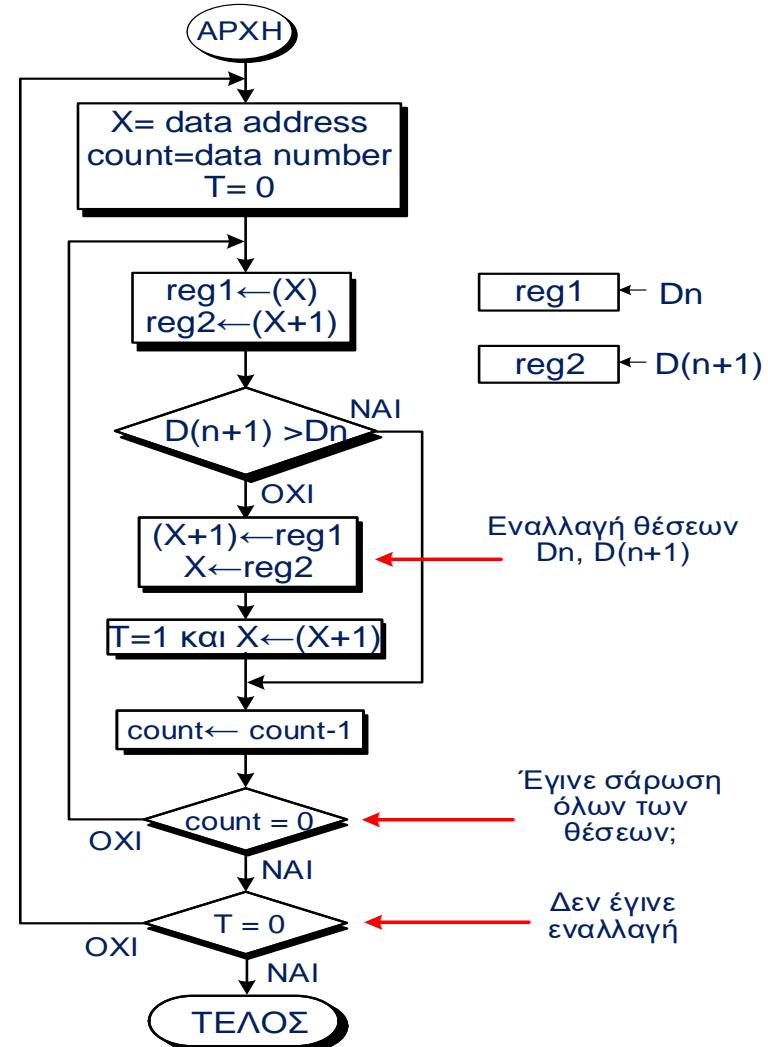
end: .exit

# Παράδειγμα 13<sup>ο</sup> : Κατάταξη αριθμών

Κατάταξη 256 αριθμών που βρίσκονται στη μνήμη δεδομένων (διεύθυνση \$0060) να τεθούν σε σειρά αύξοντος μεγέθους.

Εφαρμόζουμε τη μέθοδο των φυσαλίδων σύμφωνα με την οποία πραγματοποιούμε διαδοχικά περάσματα στην ακολουθία των αριθμών και αντιστρέφουμε τους διαδοχικούς αριθμούς που δεν βρίσκονται σε αύξουσα σειρά. Η διαδικασία επαναλαμβάνεται έως ότου να μην χρειαστεί εναλλαγή.

```
.INCLUDE "m16def.inc"
.DEF reg1=R18
.DEF reg2=R20
.EQU Location1 = 0x0060
.DEF count=r16
.EQU numb= 0x100
```



# Παράδειγμα 13<sup>ο</sup> - συνέχεια

sort:

```
LDI XH, HIGH(Location1) ; διεύθυνση μνήμης δεδομένων  
LDI XL, LOW(Location1)  
ldi count, numb ; πλήθος αριθμών  
clt ; αρχικοποίηση δείκτης αλλαγής (T=0)
```

loop:

```
ld reg1,x+ ; φόρτωση διαδοχικών τιμών από μνήμη  
ld reg2,x ; δεδομένων σε δυο καταχωρητές
```

```
cp reg2,reg1 ; σύγκριση διαδοχικών τιμών(reg2 - reg1)  
brge no_change
```

```
st x,reg1 ; εναλλαγή για αύξουσα διάταξη
```

```
st -x,reg2
```

```
adiw xl,1 ; x+
```

```
set ; θέτουμε δείκτη αλλαγής (T=1)
```

no\_change:

```
dec count ; σάρωση όλου του πίνακα
```

```
brne loop
```

```
brbs 6,sort ; επανάληψη σε περίπτωση εναλλαγής (T=1)  
;(εναλλακτικά brts sort)
```

.exit

## Παράδειγμα 14° : Χειρισμός διακοπτών εισόδου και led εξόδου

Αν ένα από τα switch 0 ή 1 είναι πατημένο, τότε ανάβει τα αντίστοιχο led.

Αν ένα από τα switch 2-6 είναι πατημένο, τότε ανάβουν τα leds 2 - 7.

Αν το switch 7 είναι πατημένο, τότε σβήνουν όλα τα leds.

Ανάστροφη λογική στα switch (0 για πατημένο switch)

Ανάστροφη λογική στα Led (0 για αναμμένο led)

```
.INCLUDE "m16def.inc"      ; δήλωση ελεγκτή  
.DEF reg = R16            ; ορίζουμε όνομα καταχωρητή
```

main:

```
ldi reg,LOW(RAMEND)      ; αρχικοποίηση δείκτη στοίβας στη  
out SPL,reg  
ldi reg,HIGH(RAMEND)     ; τελευταία θέση της SRAM (RAMEND)  
out SPH,reg  
clr reg                  ; PortD ως είσοδος  
out DDRD,reg  
ser reg                  ; Port D (pull-ups)  
out PORTD,reg             ; PortB ως έξοδος  
out DDRB,reg  
out PORTB,reg             ; σβήσιμο leds (ανάστροφη λογική)
```

# Παράδειγμα 14° - συνέχεια

loop:

```
sbis PIND,0      ; Ανάγνωση switch 0  
rcall Light0
```

; Ανάγνωση switch 1 με διαφορετικό τρόπο:

; Η πρόσβαση σε μια θύρα μπορεί να γίνει με εντολές διευθυνσιοδότησης της SRAM.

; Η διεύθυνση SRAM είναι 32 bytes υψηλότερα από την αντίστοιχη διεύθυνση θύρας.

; Ορίζουμε το X (XH:XL, R27:R26) ως δείκτη αυτής της θύρας.

```
.EQU mem_name=PIND + $20      ; διεύθυνση στην sram
```

```
ldi R26,LOW(mem_name)
```

```
ldi R27,HIGH(mem_name)
```

```
ld reg,X      ; φόρτωσε καταχωρητή reg από καταχωρητή δείκτη
```

```
sbrs reg,1      ; Έλεγχος Pin1 (switch 1- Bit 1 ). Αν είναι 0 που
```

```
rcall Light1      ; σημαίνει ότι πατήθηκε, γίνεται κλήση της Light1.
```

; Άλλιώς παρακάμπτεται για να γίνει ο έλεγχος των Switches 2 ως 6

```
in reg, PIND      ; Ανάγνωση port D (Ανάστροφη λογική στα switch).
```

```
ori reg, 0b10000011      ; Εφαρμογή μάσκας στα switches 0, 1 και 7
```

```
cpi reg, 0b11111111      ;
```

```
breq sw7      ; αν κανένας διακόπτης 2-6 δεν είναι ON τότε άλμα στην ετικέτα sw7
```

```
in reg,PIND      ; Άλλιώς ανάγνωση τρέχουσας κατάστασης LEDs
```

```
andi reg,0b00000011      ; άναμμα των LED 2 ως 7
```

```
out PORTB, reg      ; χωρίς να γίνει αλλαγή στα LED 0 και 1
```

# Παράδειγμα 14° - συνέχεια

sw7:

```
in reg,PIND          ; ανάγνωση θύρας διακοπών
rol reg              ; ολίσθηση 7ου bit στο C του SREG
brcs loop           ; αν 7ο bit είναι 1 (BRanch Carry Set) καμία αλλαγή.
ldi reg, 0xFF        ; Άλλιως (C=0) σβήσιμο όλων των LEDs.
out PORTB, reg
rjmp loop
```

Light0:

```
in reg,PIND          ; υπορουτίνα Light0: LED 0 ON
andi reg,0b11111110  ; ανάγνωση τρέχουσας κατάστασης port B
out PORTB,reg        ; άναμμα του 1ου LED τα υπόλοιπα μένουν
ret                  ; στην ίδια κατάσταση.
```

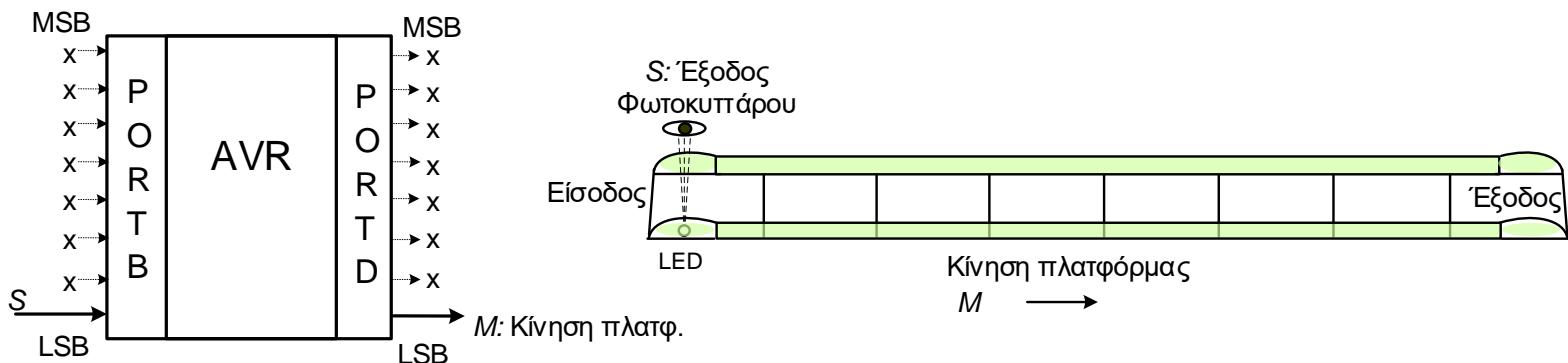
Light1:

```
in reg,PIND          ; Υπορουτίνα Light1: LED 1 ON
cbr reg,0b00000010   ; ανάγνωση κατάστασης port B
out PORTB,reg        ; Το bit 2 γίνεται μηδέν (η εντολή αυτή ισοδυναμεί
ret                  ; με andi reg, 0b11111101). Ανάβει το 2o LED και τα
                      ; υπόλοιπα LED μένουν στην ίδια κατάσταση.
```

# Παράδειγμα 15<sup>ο</sup>: Οδήγηση κυλιόμενης πλατφόρμας

Σε ένα Μικροελεγκτή AVR που το PORTB ορίζεται ως θύρα εισόδου και το PORTD ως θύρα εξόδου να υλοποιηθεί ένα σύστημα οδήγησης κυλιόμενης πλατφόρμα μονής κατεύθυνσης η οποία να ενεργοποιείται από το φωτοκύτταρο S.

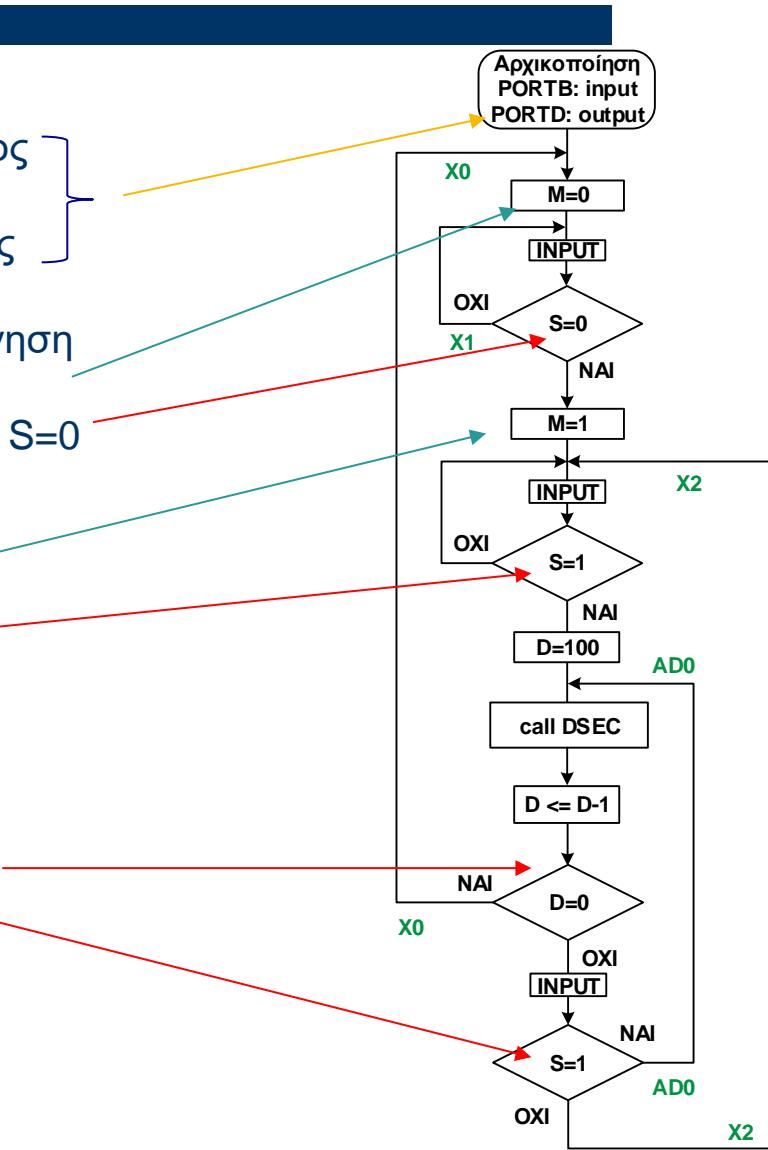
Συγκεκριμένα, αν ένας επιβάτης εισέρχεται στην πλατφόρμα, όταν είναι ακίνητη, διακόπτει δέσμη φωτός (γίνεται  $S=0$ ) και τότε τίθεται σε κίνηση η πλατφόρμα με το σήμα εξόδου M (για  $M=1$  έχουμε κίνηση). Η κίνηση να σταματά  $\sim 10$  sec μετά την τελευταία διακοπή του φωτοκυττάρου S (χρόνος για να αδειάσει η πλατφόρμα από επιβάτες). Δίνεται ρουτίνα χρονοκαθυστέρησης DSEC των 100 msec.



# Παράδειγμα 15° - συνέχεια

```

clr R16
out DDRB, R16      ; Θύρα B ως είσοδος
ser R16
out DDRD, R16      ; Θύρα D ως έξοδος
out PORTB, R16
X0: ldi R16, 0x00    ; σταματάει την κίνηση
        out PORTD, R16 ; της πλατφόρμας
X1: sbic PINB, 0     ; έλεγχος αν S=0
        rjmp X1
        ldi R16, 0x01    ; θέτει σε κίνηση
        out PORTD, R16 ; την πλατφόρμα
X2: sbis PINB, 0     ; έλεγχος S=1
        rjmp X2
        ldi R17, 0x64    ; D=100
AD0: call DSEC
      dec R17
      breq X0           ; έλεγχος αν D=0
      sbis PINB, 0       ; έλεγχος αν S=1
      rjmp X2
      rjmp AD0
  
```



# Μακροεντολές

**Παράδειγμα 15<sup>ο</sup> : Άναμμα των LEDs 0, 1, 2 με χρήση μακροεντολής**

```
.INCLUDE "m16def.inc"
.LIST
.DEF temp=R18

.MACRO FOUR_INC ; δήλωση μακροεντολής
    inc temp
    inc temp
    inc temp
    inc temp
.ENDMACRO
```

```
; Κύριο πρόγραμμα
ser temp           ; ο καταχωρητής temp=0xFF
    out DDRB,temp ; PortB (LEDs) ως έξοδος
    clr temp        ; μηδενισμός καταχωρητή temp
    FOUR_INC         ; εισαγωγή μακροεντολής (4 INCs)
    FOUR_INC         ; άλλα 4 INCs, temp=0000 1000
    Delay4
    Delay4
    Delay4          ; Συνολικά καθυστέρηση 12 NOP
    dec temp         ; temp=0000 1000 =>0000 0111
    com temp         ; αναστροφή για απεικόνιση στα LEDs
    out PORTB,temp  ; άναμμα των leds
LOOP: rjmp LOOP
```

**Ορισμός μακροεντολής:**

.MACRO Delay4	; όνομα
NOP	; σώμα
NOP	
NOP	
NOP	
.ENDMACRO	; Τέλος

# Παράδειγμα 16<sup>ο</sup> : Χρήσης ετικετών και μακροεντολών

```
.INCLUDE "m16def.inc"
.DEF temp=R18

.MACRO TestMacro      ; δήλωση μακροεντολής
    inc temp          ; αύξηση καταχωρητή temp
    brne macro_jump   ; παράκαμψη επόμενης εντολής αν δεν προκύψει υπερχείλιση
    rjmp OVERFLOW      ; άλλα σε περίπτωση υπερχείλισης σε ετικέτα εκτός μακροεντολής
macro_jump:            ; ετικέτα εντός μακροεντολής
.ENDMACRO

        ser temp          ; κύριο πρόγραμμα
        out DDRB,temp       ; PortB (LEDs) ως έξοδος
        ldi temp,0xFE        ; θέτουμε καταχωρητή=254
        TestMacro           ; εισαγωγή μακροεντολής (ένα INC), temp=255
        TestMacro           ; ξανά εισαγωγή μακροεντολής (άλλο ένα INC), temp=0 και overflow
; Η λειτουργία των μακροεντολών θα προκαλέσει υπερχείλιση (λόγω INC),
; οπότε η επόμενη εντολή δεν θα εκτελεστεί. Εάν εκτελείτο θα άναβε όλα τα LEDs.
outp:
        out PORTB,temp      ; άναμμα των leds
LOOP:   rjmp LOOP

OVERFLOW:
        ser temp          ; λόγω υπερχείλισης θα εκτελεστεί ο παρακάτω κώδικας και
        out PORTB,temp      ; τα LEDs PB.0 - PB.7 θα σβήσουν
        rjmp LOOP
```

# Παράδειγμα 17<sup>ο</sup> : Χρήση παραμέτρων σε μακροεντολή

```
.INCLUDE "m16def.inc"
.DEF temp=R18

.MACRO TestMacro
    ldi temp,@0
    inc temp
    brne macro_jump
    rjmp OVERFLOW
macro_jump:
.ENDMACRO

    ser temp           ; κύριο πρόγραμμα
    out DDRB,temp      ; PortB (LEDs) ως έξοδος

; πέρασμα τιμής 0xFF στην μακροεντολή, αύξηση κατά μια μονάδα και άλμα στην overflow
; αφού προέκυψε υπερχείλιση και εισαγωγή μακροεντολής

TestMacro(0xff)      ; Η λειτουργία της μακροεντολής θα προκαλέσει υπερχείλιση (λόγω INC),
                      ; και η επόμενη εντολή δεν θα εκτελεστεί. Εάν εκτελείτο θα άναβε όλα τα LEDs.

outp:
    out PORTB,temp   ; άναμμα των leds
LOOP:   rjmp LOOP

OVERFLOW:   ; λόγω υπερχείλισης θα εκτελεστεί ο παρακάτω κώδικας και
            ; τα LEDs PB.0 - PB.7 θα σβήσουν
    ser temp
    out PORTB,temp
    rjmp LOOP
```