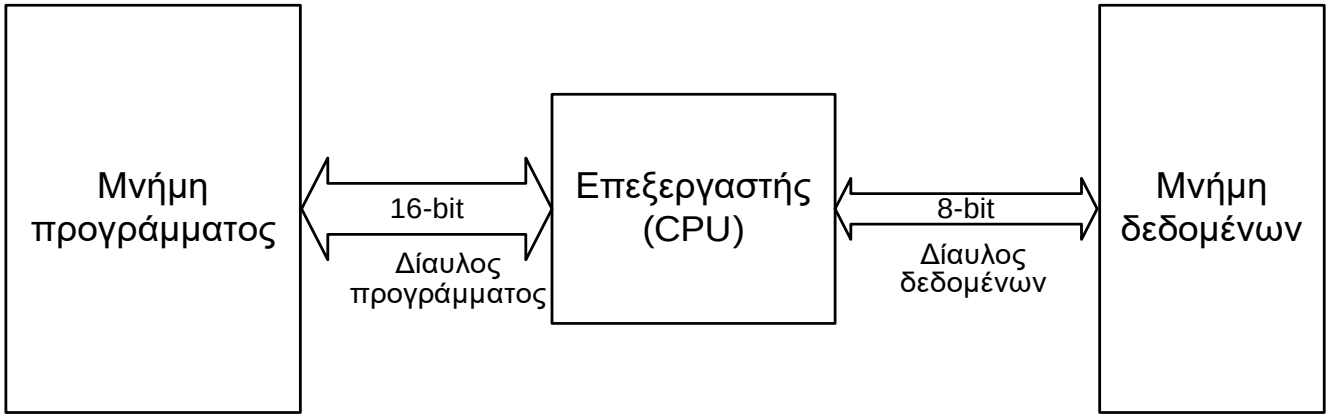
Μικροελεγκτές AVR

1η ΕΝΟΤΗΤΑ:

Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΩΝ ΜΙΚΡΟΕΛΕΓΚΤΩΝ AVR

Ε.Μ.Π.
Εργ. Μικροϋπολογιστών & Ψηφιακών Συστημάτων
Υπεύθυνος: Κ. ΠΕΚΜΕΣΤΖΗ Καθ.

Αρχιτεκτονική HARVARD



Ο μικροελεγκτής AVR περιέχει έναν επεξεργαστή RISC (Reduced Instruction Set Computer) ο οποίος έχει σχεδιαστεί με βάση την αρχιτεκτονική Harvard. Στην αρχιτεκτονική Harvard υπάρχει διαφορετικός δίαυλος για τη μεταφορά δεδομένων (data bus) και διαφορετικός δίαυλος για τη μεταφορά των εντολών (instruction bus).

Αρχιτεκτονική HARVARD

Διαφορετικός δίαυλος για τη μεταφορά:

- εντολών (instruction bus) και
- δεδομένων (data bus)

Περιλαμβάνει δύο διαφορετικές μνήμες:

- μνήμη προγράμματος (program memory) και
- μνήμη δεδομένων (data memory)

Επιτρέπει οι εντολές να έχουν διαφορετικό μήκος από τα δεδομένα ανάλογα με το πλήθος των εντολών
Επιτρέπει επικάλυψη λειτουργιών ανάκλησης εντολής, ανάγνωσης-εγγραφής δεδομένων, εκτέλεσης εντολών.

Οικογένειες του μΕ AVR

Μοντέλο	Ακροδέκτες	Flash	EEPROM	RAM	UART	ADC
90S1200	20	1K	64 bytes	0	Όχι	Όχι
90S2313	20	2K	128	128	Ναι	Όχι
90S4433	28	4K	256	128	Ναι	Ναι
90S4414	40	4K	256	256	Ναι	Όχι
90S2343	8	2K	128	128	Όχι	Όχι
Mega103	64	128K	4096	4096	Ναι	Ναι
Mega16	40	16K	512	1024	Ναι	Ναι
Mega603	64	64K	2048	4096	Ναι	Ναι
Tiny10	8	1K	64	0	Όχι	Όχι
Tiny13	8	2K	128	128	Όχι	Όχι

Η οικογένεια των μικροελεγκτών AVR έχει παρόμοιο ρεπερτόριο εντολών και βασική αρχιτεκτονική. Η διαφοροποίηση μεταξύ των διάφορων μελών της οικογένειας είναι στον αριθμό των ακροδεκτών, στο μέγεθος της μνήμης (FLASH, EEPROM, SRAM) καθώς και στα περιφερειακά τα οποία χρησιμοποιούνται.

Αρχιτεκτονική του μικροελεγκτή AVR (ATmega16)

Πυρήνας

• υψηλής απόδοσης, χαμηλής κατανάλωσης επεξεργαστής **CPU**

• αρχιτεκτονική **RISC**

– 131 ισχυρές εντολές

– 32 x 8 καταχωρητές

– Έως 16 MIPS απόδοση στα 16 MHz

▪ Μονάδα συνεχούς διοχέτευσης εντολών (Instruction Pipelining)

▪Αριθμητική Λογική Μονάδα (**A**rithmetic **L**ogic **U**nit, ALU)

• μνήμη προγράμματος και δεδομένων

– 16K Bytes από **Flash** (In-System Self-Programmable)

– Προαιρετικό τμήμα Boot Code με ανεξάρτητα bits κλειδώματος (Lock Bits)

– 512 Bytes **EEPROM** (Αντοχή: 100,000 κύκλοι εγγραφής/ανάγνωσης)

– 1K Byte εσωτερικής μνήμης **SRAM** και **στοίβα**

– Programming Lock για ασφάλεια λογισμικού

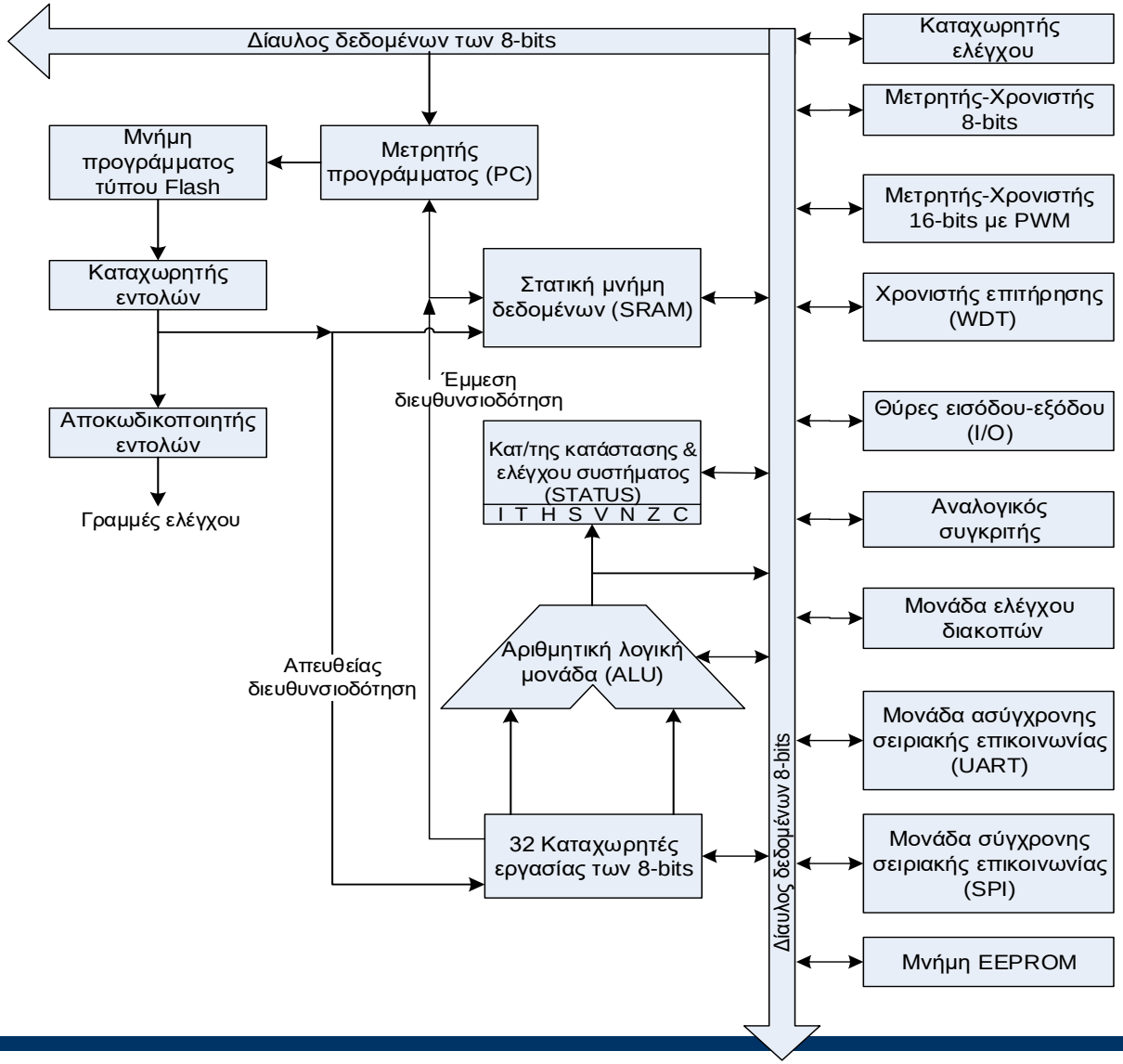
• **JTAG διεπαφή**

– Προγραμματισμός των Flash, EEPROM, ασφαλειών, και Bits κλειδώματος μέσω της διεπαφής JTAG

Αρχιτεκτονική του μικροελεγκτή AVR (ATmega16)

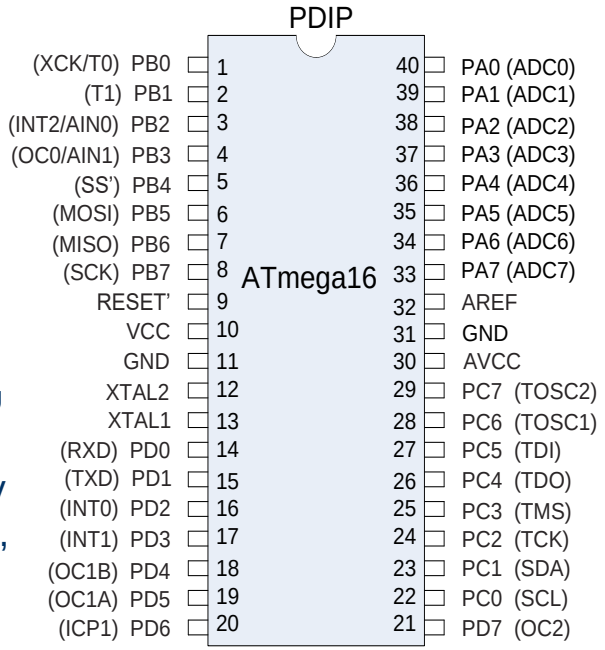
• **Περιφερειακά**
– Δύο 8-bit Timer/Counters με ξεχωριστές μονάδες διαίρεσης χρόνου (Prescalers) και τρόπους σύγκρισης (Compare Modes)
– Ένα 16-bit Timer/Counter με ξεχωριστή μονάδα διαίρεσης χρόνου (Prescaler), τρόπο σύγκρισης, και σύλληψης (Capture Mode)
– Μετρητή πραγματικού χρόνου με ανεξάρτητο ταλαντωτή
– 4 PWM κανάλια
– 8-κάναλο, 10-bit ADC
–Two-wire σειριακή διεπαφή TWI (IIC)
– Προγραμματιζόμενη σειριακή USART
– Master/Slave SPI σειριακή διεπαφή
– Προγραμματιζόμενος χρονιστής επιτήρησης (Watchdog Timer)
– Αναλογικός συγκριτής
• **Ειδικά χαρακτηριστικά**
– Power-on Reset και προγραμματιζόμενο κύκλωμα ανίχνευσης βύθισης τάσης τροφοδοσίας(Brown-out)
– εσωτερικός RC ταλαντωτής
– Εξωτερικές και εσωτερικές πηγές διακοπών
– 6 λειτουργίες ηρεμίας: Idle, ADC Noise Reduction, Power-save, Power-down, Standby
• Τάση λειτουργίας (4.5 - 5.5V για ATmega16)
• Ταχύτητα λειτουργίας (0 - 16 MHz για ATmega16)
• Κατανάλωση ισχύος για 1 MHz, 3V, και 25°C του ATmega16L
– δραστήρια κατάσταση: 1.1 mA– Power-down κατάσταση: < 1 μA– άεργη κατάσταση:0.35 mA

Αρχιτεκτονική του μικροελεγκτή AVR



Ακροδέκτες του μικροελεγκτή AVR (ATmega16)

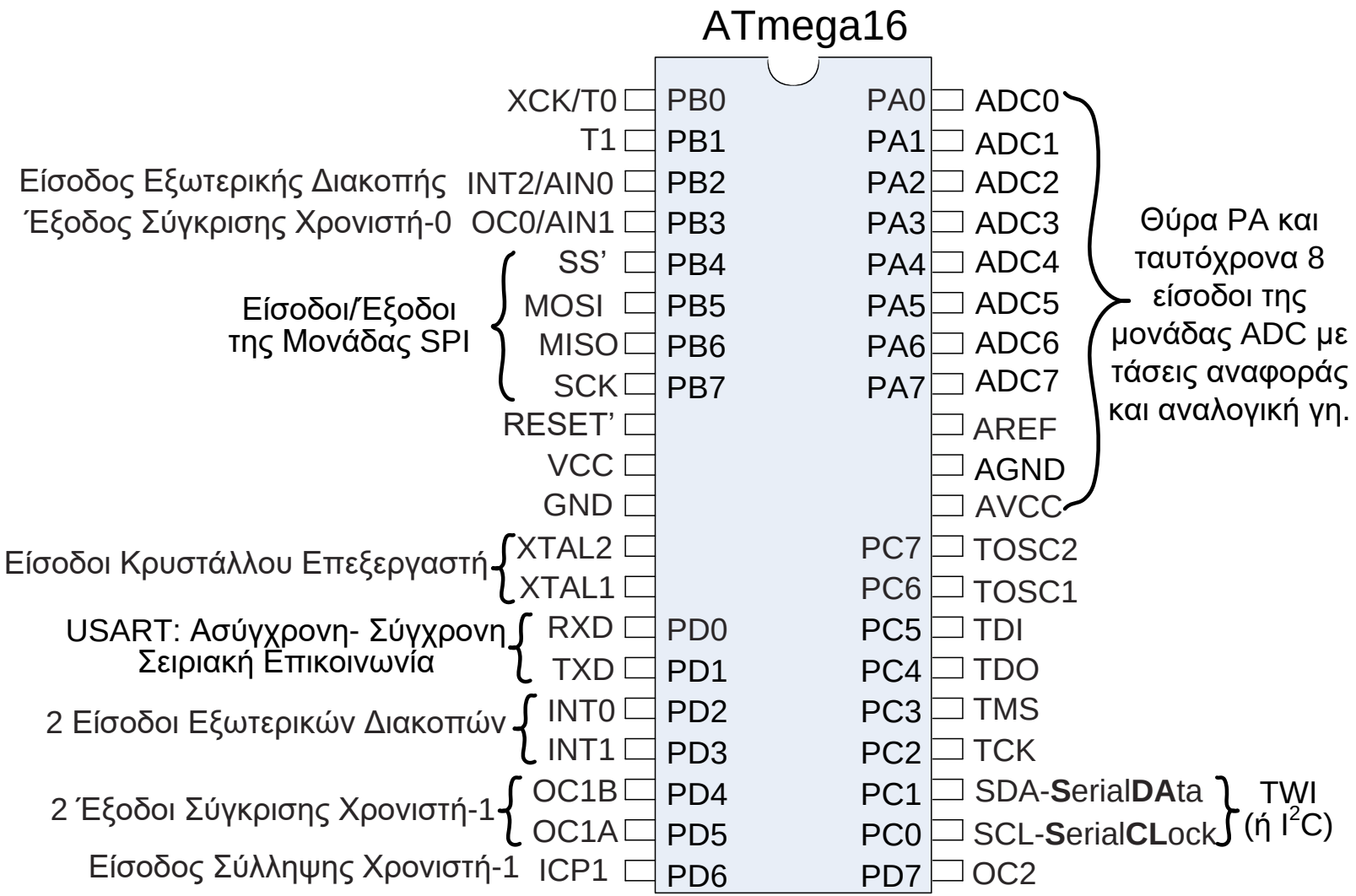
VCC Ψηφιακή τροφοδοσία τάσης.
GND Γείωση.
Port A (PA7..PA0) Η Port A λειτουργεί είτε ως αναλογική είσοδος του μετατροπέα A/D είτε
ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8-bit.
Port B (PB7..PB0) Η Port B λειτουργεί ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8-bit
Port C (PC7..PC0) Η Port C λειτουργεί ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8-bit
Port D (PD7..PD0) Η Port D λειτουργεί ως θύρα εισόδου-εξόδου διπλής κατεύθυνσης των 8-bit).
Επίσης, οι θύρες εκτελούν κάποιες ιδιαίτερες λειτουργίες του μικροελεγκτή.
RESET Έμφάνιση χαμηλού επιπέδου στον ακροδέκτη αυτόν για χρόνο μεγαλύτερο από τον ελάχιστο μήκος παλμού(2μs), θα παράγει reset.
XTAL1 Είσοδος στον ταλαντωτή-ενισχυτή και στο κύκλωμα εσωτερικού ρολογιού.
XTAL2 Έξοδος από ταλαντωτή-ενισχυτή.
AVCC είναι ο ακροδέκτης τροφοδοσίας τάσης για την Port A και τον μετατροπέα A/D.
AREF είναι ο *analog reference* ακροδέκτης για τον μετατροπέα A/D.



Περιφερειακά του μΕ AVR

- Σειριακή μονάδα USART
- 8-bit Timer/Counter με (Prescaler) και λειτουργία σύγκρισης (Compare Mode)
- 16-bit Timer/Counter με (Prescaler), λειτουργία σύγκρισης, σύλληψης (Capture Mode), 4 PWM κανάλια
- Χρονιστής επιτήρησης (Watchdog Timer)
- Μνήμη EEPROM
- Master/Slave SPI σειριακή διεπαφή
- Two-wire σειριακή διεπαφή TWI
- 8-κάναλο, 10-bit ADC
- Αναλογικός συγκριτής

Ακροδέκτες περιφερειακών ATmega16



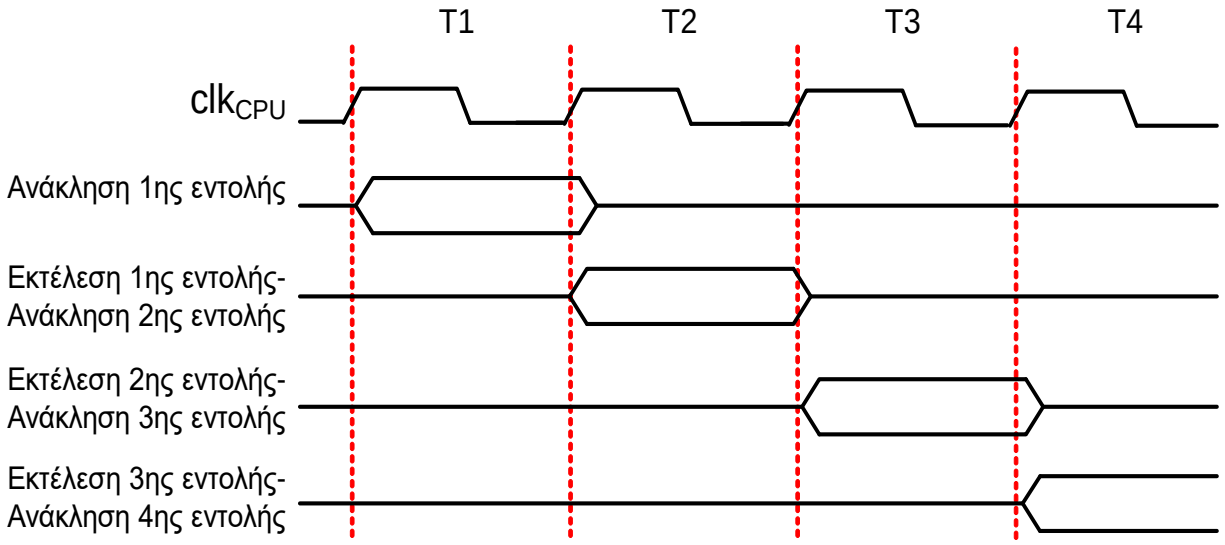
Θύρες- καταχωρητές των μικροελεγκτών AVR

Μονάδα	Καταχωρητής	Σύμβολο
Timer 0	Timer/Counter 0 Control Register	TCCR0
	Timer/Counter 0	TCNT0
Timer 1	Timer/Counter Control Register1A	TCCR1A
	Timer/Counter Control Register1B	TCCR1B
	Timer/Counter 1	TCNT1
	Output Compare Register 1 A	OCR1A
	Output Compare Register 1 B	OCR1B
	Input Capture Register	ICR1L - H
Timer 2	Timer/Counter2 (8 Bits)	TCNT2
	Timer/Counter Control Register	TCCR2
	Timer/Counter2 Output Compare Register	OCR2
	Asynchronous Status Register	ASSR
Watchdog Timer	Watchdog Timer Control Register	WDTCR
UART	UART Data Register	UDR
	UART Status Register	USR
	UART Control Register	UCR
	UART Baud Rate Register	UBRR

Θύρες-καταχωρητές των μικροελεγκτών AVR

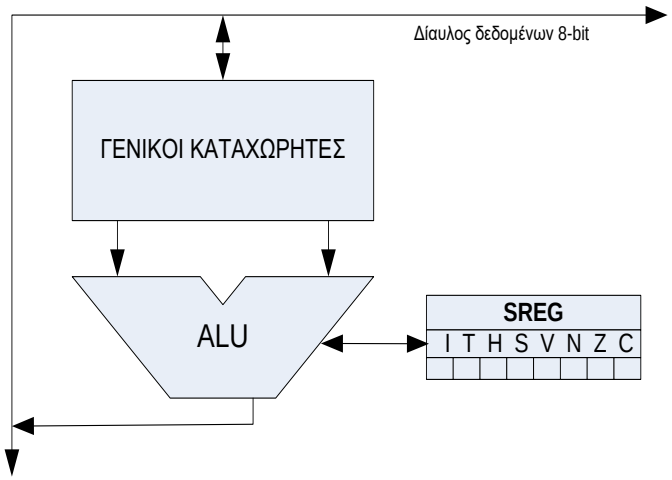
Μονάδα	Καταχωρητής	Σύμβολο
EEPROM	EEPROM Adress Register	EEAR
	EEPROM Data Register	EEDR
	EEPROM Control Register	EECR
SPI	Serial Peripheral Control Register	SPCR
	Serial Peripheral Status Register	SPSR
	Serial Peripheral Data Register	SPDR
Analog Comparator	Analog Comparator Control and Status Register	ACSR
Two-wire Serial (I^2C) Interface TWI	TWI Data Register	TWDR
	TWI Address Register	TWAR
	TWI Bit Rate Register	TWBR
	TWI Control Register	TWCR
	TWI Status Register	TWSR
ADC	Πολυπλέκτης επιλογής 8 γραμμών εισόδου ADC Multiplexer Selection Register	ADCMUX
	ADC Control and Status Register A	ADCSRA
	ADC Data Register (high and low)	ADCH - L

Παράδειγμα λειτουργίας του PIPELINE

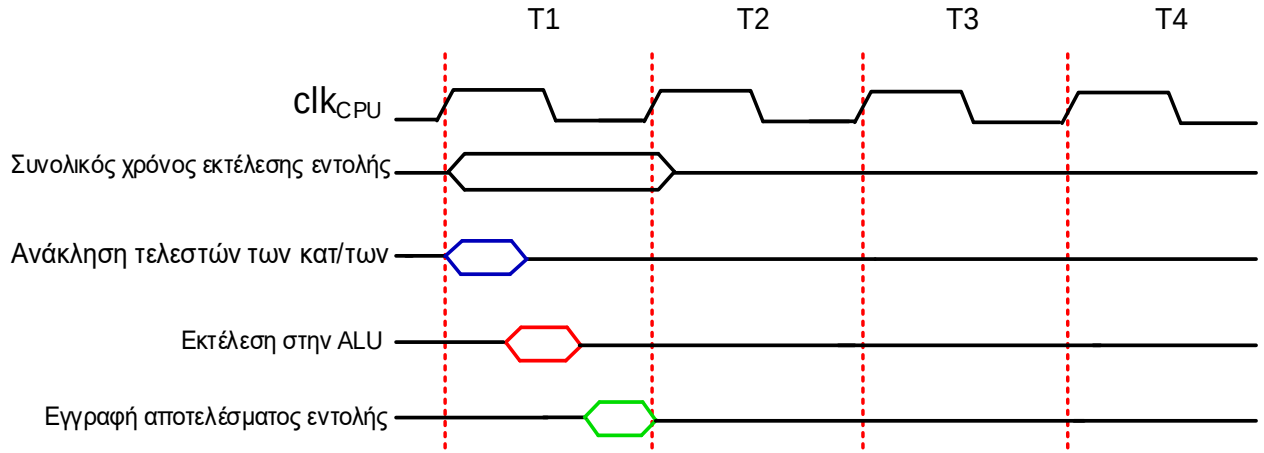


Στη λειτουργία Pipeline έχουμε συνεχή διοχέτευση εντολών με αποτέλεσμα την παράλληλη ανάκληση και εκτέλεση εντολών. Οπότε, η εκτέλεση των περισσοτέρων εντολών γίνεται σε μία περίοδο του κεντρικού ρολογιού με αποτέλεσμα απόδοση έως 1 MIPS ανά MHz.

Αριθμητική Λογική Μονάδα (ALU) του AVR



Είναι η κεντρική μονάδα του επεξεργαστή. Αυτή εκτελεί αριθμητικές, λογικές και σε επίπεδο bit εντολές στη διάρκεια μιας περιόδου και αποθηκεύει το αποτέλεσμα στον καταχωρητή που δείχνει η εντολή. Κατά την εκτέλεση των πράξεων ενημερώνονται οι σημαίες του καταχωρητή κατάστασης (Status register).



Διαδικασία εκτέλεσης εντολής στην ALU

Καταχωρητής κατάστασης (SREG) STATUS REGISTER

Bit	7	6	5	4	3	2	1	0
Σημαία	I	T	H	S	V	N	Z	C

Bit7 **(I):** καθολική **ενεργοποίηση διακοπών** (Global Interrupt Enable-GIE). Θέτοντας το bit αυτό ενεργοποιούμε τις διακοπές

Bit6 **(T):** σημαία **αντιγραφής-αποθήκευσης (bit** Copy-Storage). Με χρήση των εντολών BLD και BST επιτυγχάνουμε την ανάγνωση και αποθήκευση συγκεκριμένων bits

Bit5 **(H):** σημαία **δεκαδικού κρατουμένου** (Half Carry flag). Ενημερώνει για την ύπαρξη δεκαδικού κρατουμένου μετά από αριθμητικές πράξεις

Bit4 **(S):** σημαία **προσήμου** (Sign flag). Ενημερώνει για το πρόσημο ενός καταχωρητή και ισούται με το XOR των σημαιών αρνητικού προσήμου και υπερχείλισης.(S=N xor V) Όταν έχουμε υπερχείλιση αποτέλεσμα που εμφανίζεται >0 είναι στην πραγματικότητα <0

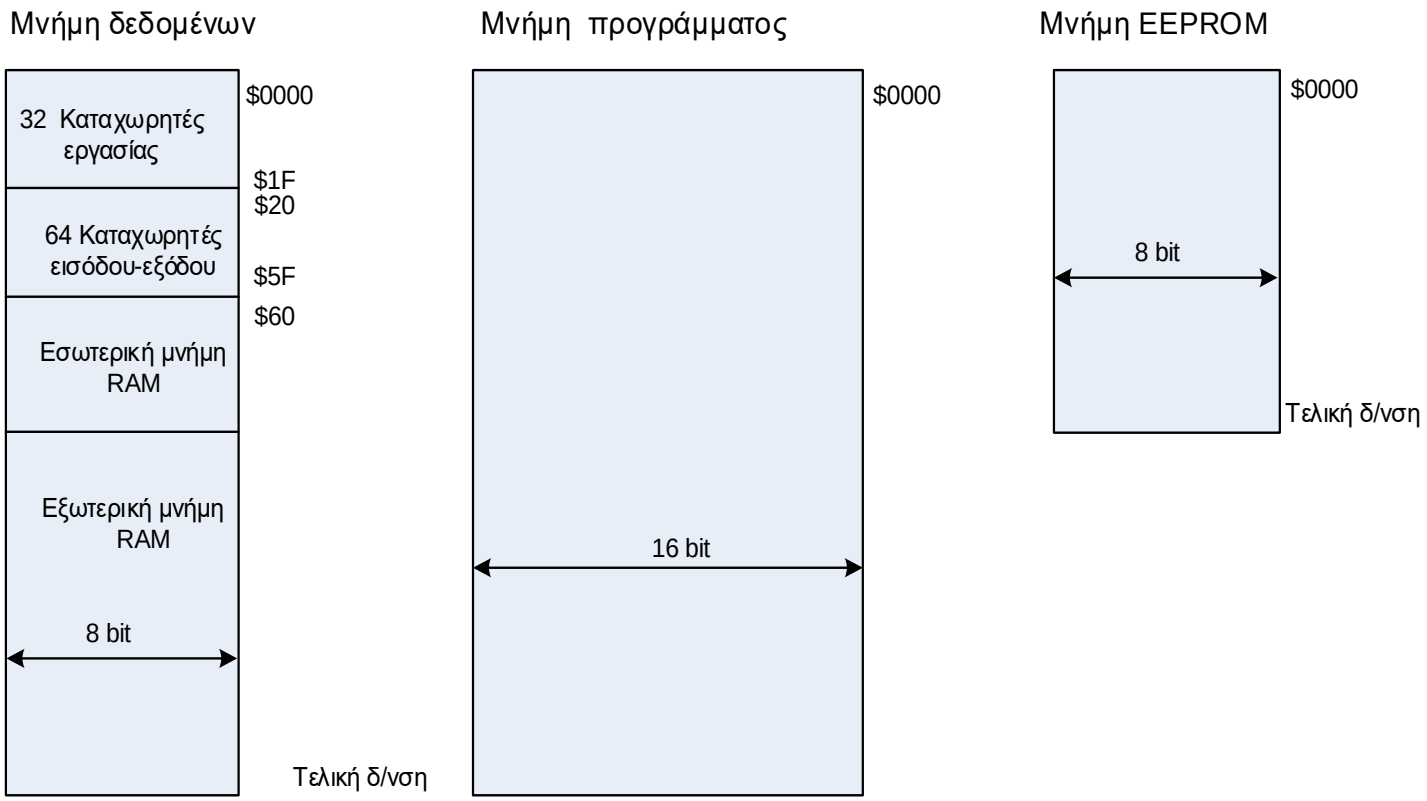
Bit3 **(V):** σημαία **υπερχείλισης** (Overflow flag) για αριθμητική συμπληρώματος του δύο

Bit2 **(N):** σημαία **αρνητικού προσήμου** (negative flag)

Bit1 **(Z):** σημαία **μηδενισμού** (Zero flag).Τίθεται όταν το αποτέλεσμα μιας πράξης είναι 0

Bit0 **(C):** σημαία **κρατουμένου** (Carry flag).

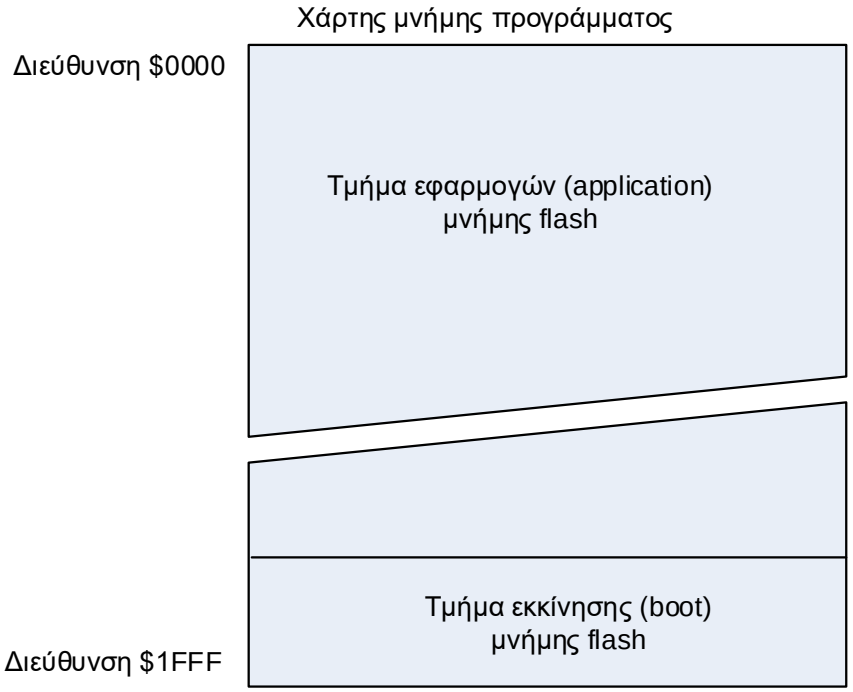
Τύποι μνήμης του μικροελεγκτή AVR



Ένας μικροελεγκτής διαθέτει τα εξής είδη μνήμης:

- μνήμη δεδομένων (1 Kbyte)
- μνήμη προγράμματος τύπου flash (16KByte)
- μνήμη EEPROM με αρχική διεύθυνση $0000 και χωρητικότητα από 64bytes ως 4Kbytes.

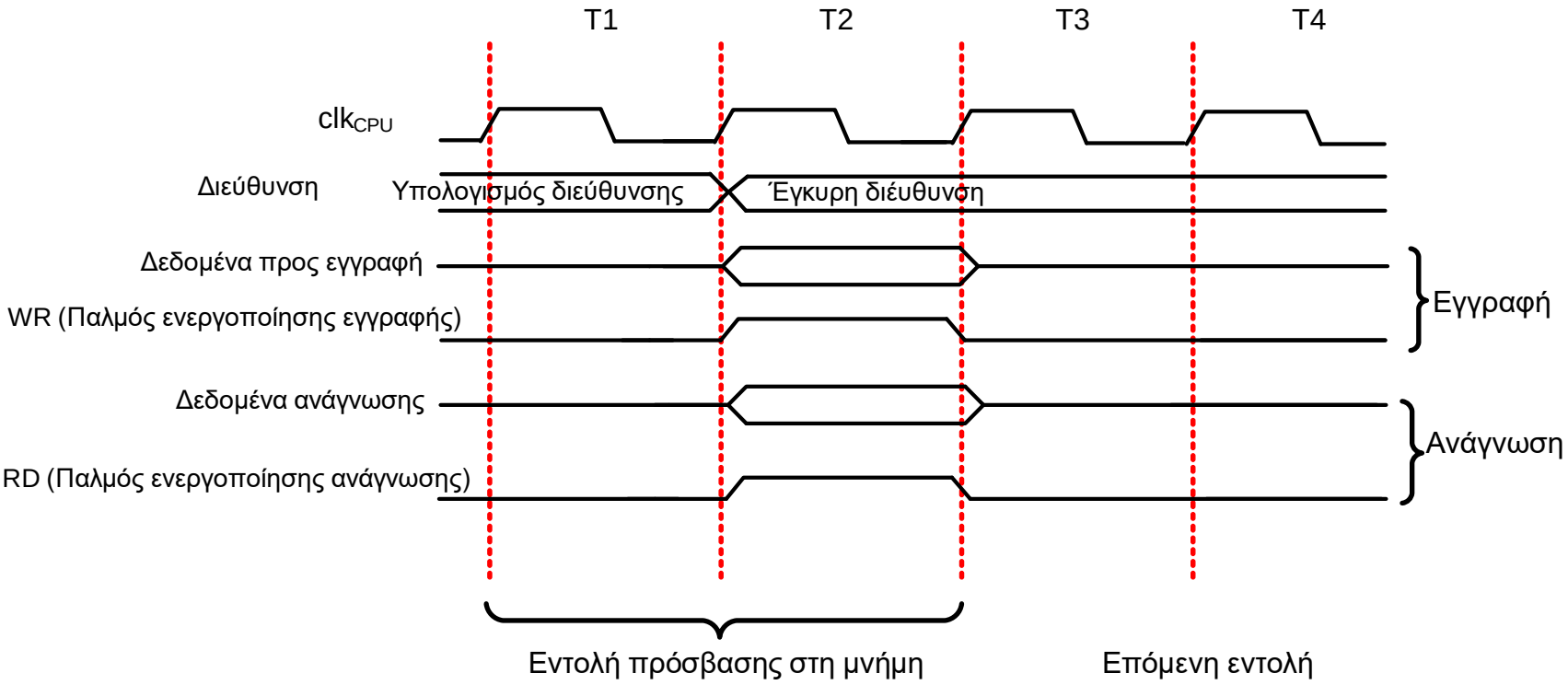
Μνήμη προγράμματος του μΕ AVR



- τύπου flash για ταχεία αποθήκευση-φόρτωση δεδομένων
- σύνδεση με δίαυλο των 16-bit για την τροφοδοσία του καταχωρητή εντολών.
- αποθήκευση εντολών και διανυσμάτων διακοπών (interrupt vectors).

Όλες οι εντολές έχουν μήκος 16 bits ή 32 bits (2 bytes=1 λέξη), άρα καταλαμβάνουν μία ή δύο θέσεις της μνήμης προγράμματος, αφού η μνήμη flash είναι οργανωμένη σε διάταξη 8Kx16. Ο μετρητής προγράμματος του ATmega16 έχει μήκος 13 bits ώστε να επιτυγχάνει πρόσβαση σε 8K θέσεις μνήμης προγράμματος.

Φάσεις ανάγνωσης-εγγραφής στην εσωτερική μνήμη SRAM



Μνήμη δεδομένων του μΕ AVR

ΜΝΗΜΗ ΔΕΔΟΜΕΝΩΝ								
Καταχ/τές εργασίας (00-1F)	R0	R1	R2	R3	R4	R5	R6	R7
	R8	R9	R10	R11	R12	R13	R14	R15
	R16	R17	R18	R19	R20	R21	R22	R23
	R24	R25	R26	R27	R28	R29	R30	R31
Καταχ/τές I/O (20-5F) {00-3F}	SREG	PORTA	TCCR0	EEAR				
	SPL	DDRA	TCNT0	EEDR				
	SPH	PINA		EECR				
		PORTB	TCCR1A					
	GIMSK	DDRB	TCCR1B					
	GIFR	PINB	TCNT1H	UDR				
		PORTC	TCNT1L	USR				
	TIMSK	DDRC	OCR1AH	UCR				
	TIFR	PINC	OCR1AL	UBRR				
		PORTD	ICR1H					
	MCUCR	DDRD	ICR1L	ACSR				
		PIND	WDTCR					
SRAM ($60-$45F)	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF

Η μνήμη δεδομένων συνδέεται με ένα δίαυλο των 8-bit για την επικοινωνία των περιφερειακών με τους κατ/τές ελέγχου.

Η μνήμη δεδομένων χωρίζεται στα τμήματα:

- 32 καταχωρητές εργασίας (register file) των 8-bits (R0-R31)
- 64 καταχωρητές εισόδου-εξόδου των 8-bits
- εσωτερική μνήμη SRAM (για αποθήκευση μεταβλητών και ως στοίβα (stack))
- εξωτερική SRAM (1KB).

Εντολές του μΕ AVR

Οι εντολές της γλώσσας των μικροελεγκτών μπορούν να ομαδοποιηθούν στις παρακάτω τέσσερις κατηγορίες :

- μεταφοράς δεδομένων
- αριθμητικών και λογικών πράξεων
- σε επίπεδο bit και ελέγχου bit
- ελέγχου ροής του προγράμματος και διακλάδωσης

Οι εντολές περιλαμβάνουν 2 τμήματα. Το πρώτο (operation code ή opcode) αποτελεί το λειτουργικό κώδικα που πληροφορεί τον επεξεργαστή για τις ενέργειες που πρέπει να εκτελεστούν. Το δεύτερο τμήμα προσδιορίζει τους τελεστές (r, operand) για τους οποίους θα λειτουργήσει ο κώδικας.

Καταχωρητές X, Y, Z έμμεσης διευθυνσιοδότησης

X (XH:XL, R27:R26), **Y** (YH:YL, R29:R28) και **Z** (ZH:ZL, R31:R30). Επιτρέπουν πρόσβαση στη θέση που δείχνουν (π.χ. ST X, R1), μετά από μείωση της διεύθυνσης (π.χ. ST -X, R1) ή μετέπειτα αύξηση της διεύθυνσης (π.χ. ST X+, R1)

Εντολές μεταφοράς δεδομένων

Πρόκειται για εντολές μεταφοράς δεδομένων μεταξύ καταχωρητών ή μεταξύ καταχωρητών και μνήμης ή σταθεράς σε καταχωρητή. Οι σημαίες δεν επηρεάζονται. Οι Rd, Rr ανήκουν στους 32 καταχωρητές εργασίας R0-R31.

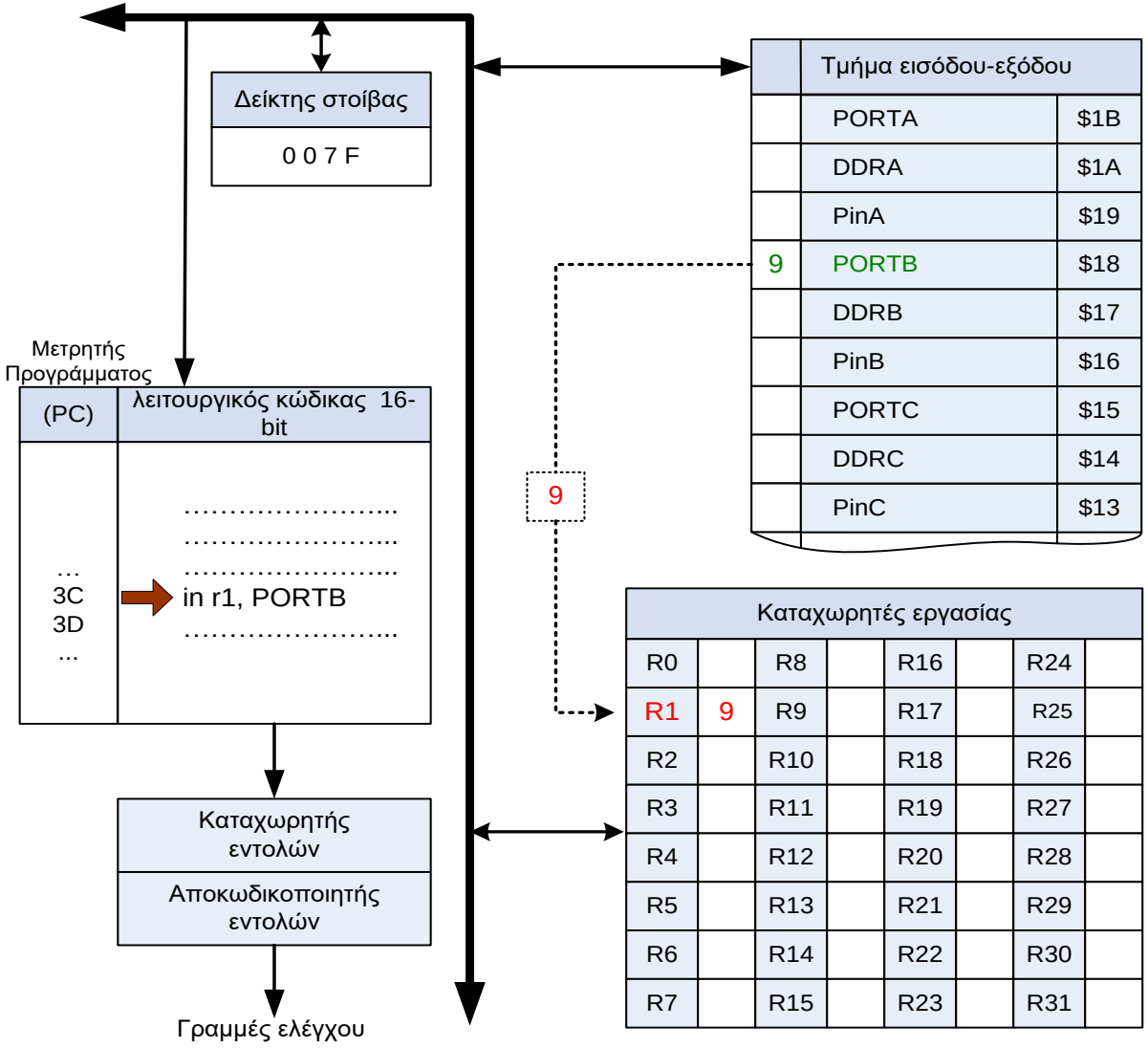
MOV	Rd, Rr	Move Between Registers	Rd← Rr
LDI	Rd, K	Load Immediate	Rd ← K
LD	Rd, X	Load Indirect	Rd ← (X)
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)
ST	X, Rr	Store Indirect	(X) ← Rr
STS	k, Rr	Store Direct to SRAM	(k) ← Rr
LPM		Load Program Memory	R0 ← (Z)
LPM	Rd, Z	Load Program Memory	Rd ← (Z)
SPM		Store Program Memory	(Z) ← R1:R0
IN	Rd, P	In Port	Rd ← P
OUT	P, Rr	Out Port	P ← Rr
PUSH	Rr	Push Register on Stack	STACK ← Rr
POP	Rd	Pop Register from Stack	Rd ← STACK

Παραδείγματα έμμεσης διευθυνσιοδότησης

X (XH:XL, R27:R26), Y (YH:YL, R29:R28) και Z (ZH:ZL, R31:R30). Επιτρέπουν πρόσβαση στη θέση που δείχνουν (π.χ. ST X, R1), μετά από μείωση της διεύθυνσης (π.χ. ST -X, R1) ή αύξηση της διεύθυνσης μετά την εκτέλεση της εντολής (π.χ. ST X+, R1).

```
; εγγραφή
    ldi r28,$60     ; Η αρχική διεύθυνση είναι $0060 και τοποθετείται στον
    clr r29         ; καταχωρητή Y σαν δείκτης διεύθυνσης.
    st y+,data      ; Αποθήκευση περιεχομένων του καταχωρητή data
                    ; στην SRAM και αύξηση της διεύθυνσης.
; ανάγνωση
    ldi r28,$60     ; Η αρχική διεύθυνση είναι $0060 και τοποθετείται στον
    clr r29         ; καταχωρητή Y σαν δείκτης διεύθυνσης.
    ld data,y+      ; Ανάγνωση περιεχομένων καταχωρητή data από την SRAM
                    ; και αύξηση της διεύθυνσης.
```

Παράδειγμα μεταφοράς δεδομένων

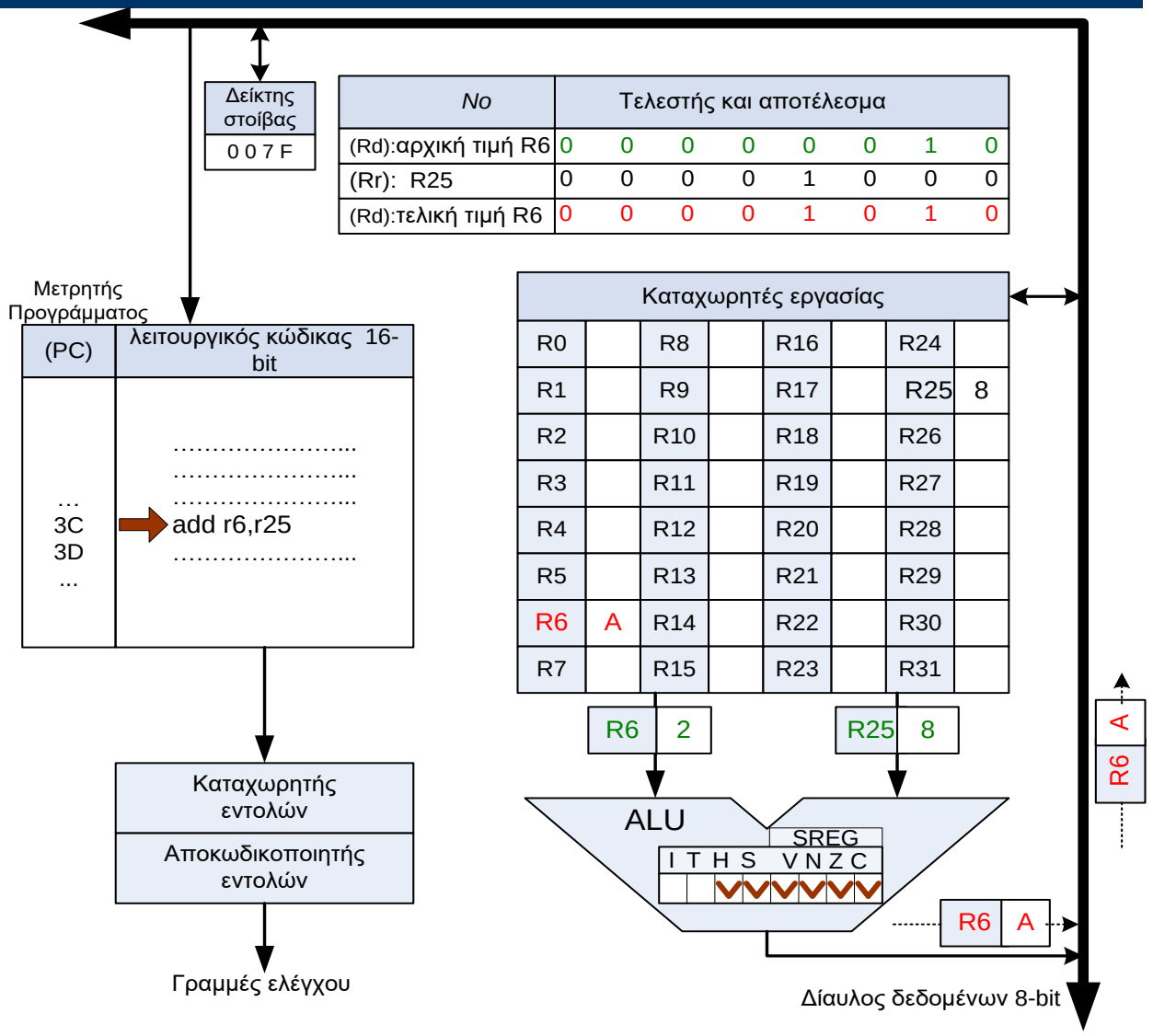


Εντολές αριθμητικών και λογικών πράξεων

Προϋποθέτουν τη χρήση της αριθμητικής λογικής μονάδας σε αντίθεση με τις εντολές μεταφοράς. Η εντολή διαβάζει τα περιεχόμενα του καταχωρητή, εκτελεί τις πράξεις με αυτά και αποθηκεύει το αποτέλεσμα στον ίδιο ή σε άλλον καταχωρητή.

ADD /SUB	Rd, Rr	Add/ Subtract two Registers	Rd ← Rd ± Rr
ADC /SBC	Rd, Rr	Add with Carry two Regs	Rd ← Rd ± (Rr + C)
SUBI	Rd, K	Subtract Constant from Reg	Rd ←Rd – K
ADIW/ SBIW	Rd, K	add/subtract Immediate to word	Rd+1:Rd←Rd+1:Rd ±K(0,63)
AND/OR/ EOR	Rd, Rr	AND/OR/XOR Regs	Rd ← Rd ∩/v/⊕ Rr
ANDI/ ORI	Rd, K	AND/OR Reg & Constant	Rd ← Rd ∩/v K
COM	Rd	One's Complement	Rd ← $FF ⊕ Rd
SBR/CBR	Rd,K	Set/ Clear Bit(s) in Reg	Rd ← Rd v K / ∩($FF – K)
INC /DEC	Rd	Increment /Decrement	Rd ← Rd ± 1
TST	Rd	Test for Zero or Minus	Rd ← Rd ∩ Rd
CLR /SER	Rd	Clear /Set Reg	Rd ← Rd ⊕ Rd / Rd ←$FF
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd × Rr

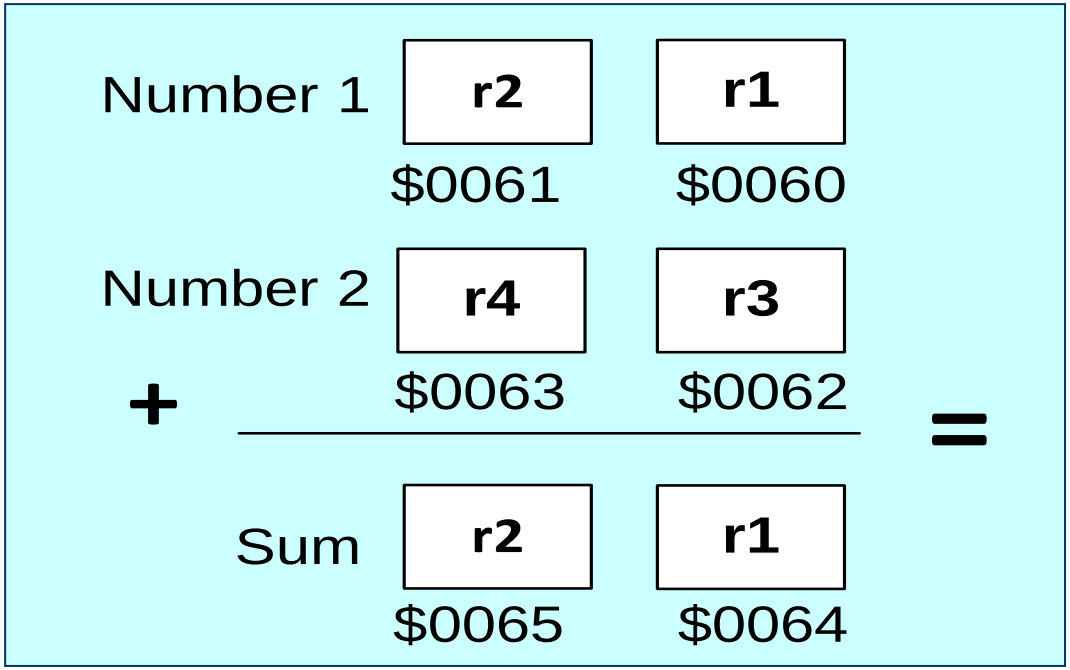
Παράδειγμα αριθμητικών και λογικών πράξεων



Παράδειγμα 1ο : Να προστεθούν δυο 16-bit αριθμοί, που βρίσκονται στις θέσεις μνήμης 0x0060-0x0061 και 0x0062-0x0063. Το αποτέλεσμα σώζεται στη διεύθυνση 0x0064-0x0065 της μνήμης δεδομένων.

```
.include "m16def.inc"        ; δήλωση μικροελεγκτή

.org 0x000
main:                        ; κυρίως πρόγραμμα
  ldi xl, 0x60               ; θέτουμε διεύθυνση 0x0060 στον καταχωρητή X
  clr xh                     ; όπου xl=R26, xh=R27
  ld r1,x+                   ; ανάγνωση θέσης μνήμης, αποθήκευση σε κατ/τη
                             ; και αύξηση δείκτη για επόμενη θέση
  ld r2,x+                   ; ανάγνωση 0x0061
  ld r3,x+                   ; ανάγνωση 0x0062
  ld r4,x+                   ; ανάγνωση 0x0063
  add r1,r3                  ; πρόσθεση λιγότερο σημαντικών byte
  adc r2,r4                  ; πρόσθεση περισσότερο σημαντικών byte με κρατούμενο
  st x+,r1                   ; αποθήκευση αποτελέσματος στη διεύθυνση
  st x,r2                    ; 0x0064 και 0x0065
 .exit
```

Πρόσθεση δυο 16-bit αριθμών

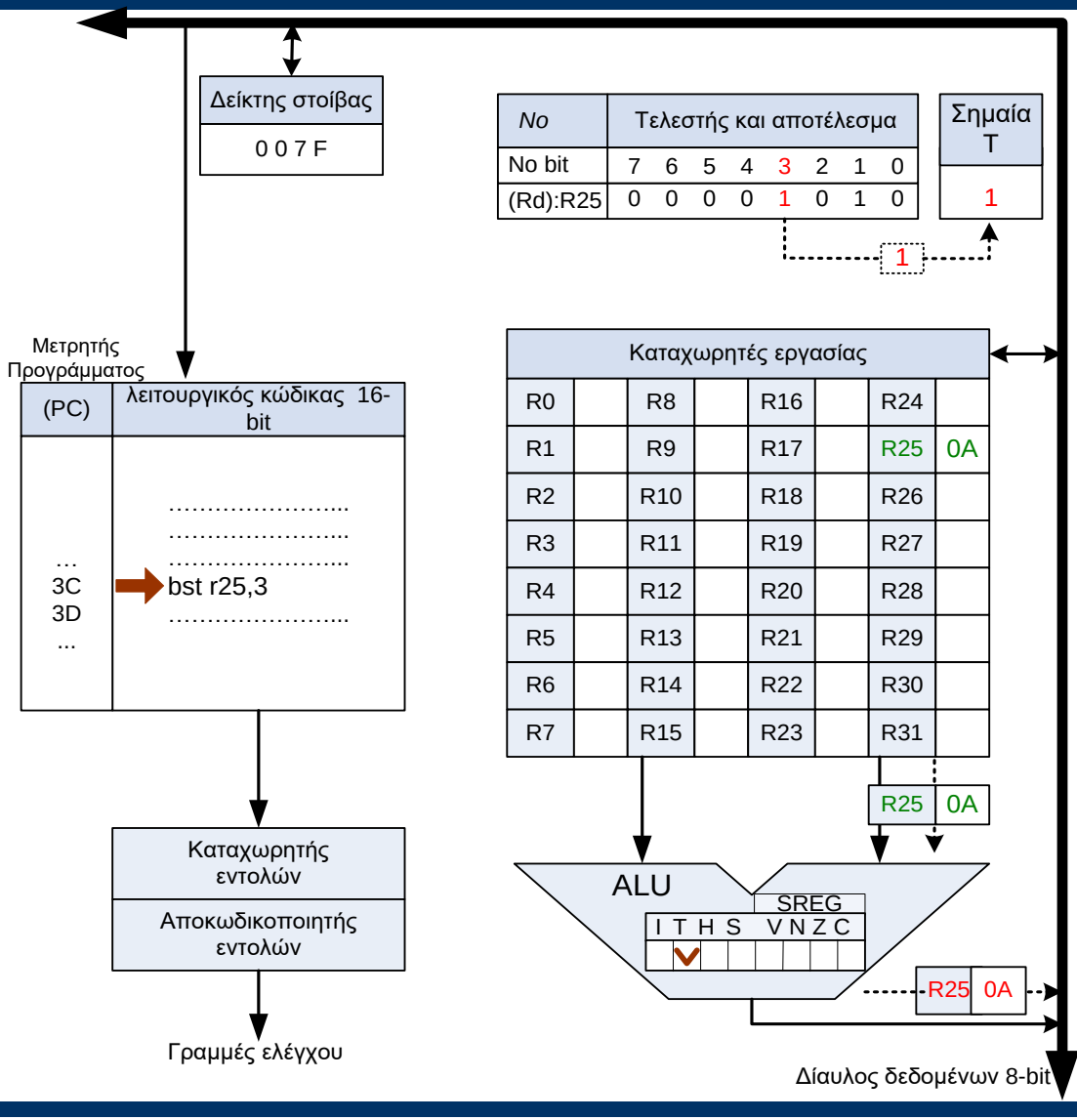


Εντολές σε επίπεδο bit και ελέγχου bit

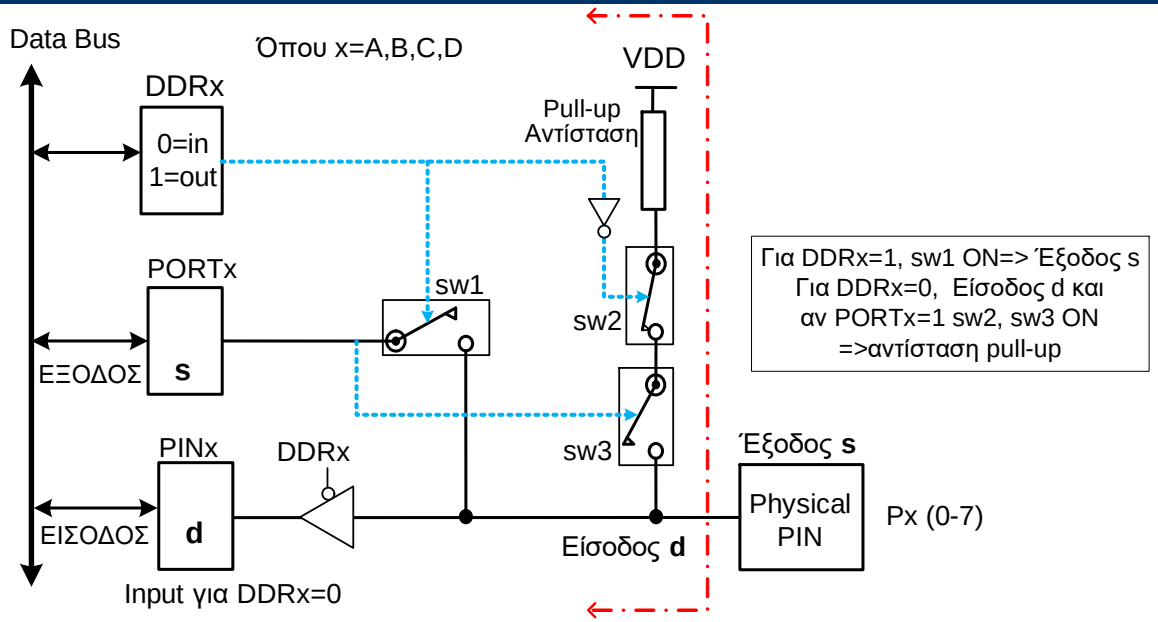
Οι εντολές αυτές μας παρέχουν τη δυνατότητα να θέσουμε ή να μηδενίσουμε σημαίες του καταχωρητή κατάστασης, διακοπές και συγκεκριμένα bit καταχωρητών ή θυρών. Επίσης, να περιστρέψουμε κάποιον καταχωρητή μέσω κρατουμένου ή όχι.

SBI/ CBI	P,b	Set/Clear Bit in I/O Reg	I/O(P,b) ← 1/ 0
LSL/ LSR	Rd	Logical Shift Left/ Right	Rd(n+1) ← Rd(n), Rd(0) ← 0
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)←Rd(n),C←Rd(7)
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)←Rd(n+1),C←Rd(0)
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6, Rd(7)=Rd(7)
SWAP	Rd	Swap Nibbles	Rd(3-0)←Rd(7-4) Rd(7-4)←Rd(3-0)
BSET/BCLR	s	Flag Set/Clear (s=0-7)	SREG(s) ← 1/ 0
BST	Rr, b	Bit Store from Reg to T	T ← Rr(b)
BLD	Rd, b	Bit load from T to Reg	Rd(b) ← T
SEx (π.χ. SEC)		Set Flag (x=I,T,H,S,V,N,Z,C)	x ← 1 όπου x=σημαία (SREG)
CLx (π.χ. CLT)		Clear Flag (x=I,T,H,S,V,N,Z,C)	x ← 0 όπου x=σημαία (SREG)

Παράδειγμα σε επίπεδο bit και ελέγχου bit



Θύρες εισόδου-εξόδου



Ο επεξεργαστής επικοινωνεί με τις διάφορες μονάδες γράφοντας ή διαβάζοντας στις θύρες-καταχωρητές: PORTA, PORTB, PORTC και PORTD.

Οι ακροδέκτες είναι διπλής κατεύθυνσης και ελέγχονται από το αντίστοιχο bit του καταχωρητή κατεύθυνσης που καλείται *DDRx* .

Ο καταχωρητής *PORTx* περιέχει τα δεδομένα της αντίστοιχης θύρας.

Η διεύθυνση ακροδεκτών εισόδου *PINx* δεν είναι καταχωρητής. Αυτή η διεύθυνση επιτρέπει την πρόσβαση στις λογικές στάθμες κάθε ακροδέκτη της θύρας PORTx.

Για οικονομία ακροδεκτών, οι είσοδοι-έξοδοι των περιφερειακών όπως: A/D μετατροπέας, οι σειριακές θύρες κλπ., χρησιμοποιούν τους ίδιους ακροδέκτες με τις θύρες.

Μετατροπή θύρας σε είσοδο/έξοδο

1. Μετατροπή θύρας σε είσοδο

```
LDI  R12, 0b00000000        ; ή απλούστερα CLR R12
OUT  DDRD, R12              ; τα ψηφία της θύρας PORTD γίνονται είσοδοι
```

2. Μετατροπή θύρας σε έξοδο

```
LDI  R18, 0b11111111        ; ή απλούστερα SER R19
OUT  DDRB, R19              ; τα ψηφία της θύρας PORTB γίνονται έξοδοι
```

Είσοδος Δεδομένου: Ανάγνωση από θύρα εισόδου- εξόδου με την εντολή 'in'

```
CLR R12
OUT DDRD, R12               ; τα ψηφία της θύρας PORTD γίνονται είσοδοι
LDI R12, 0b11111111         ; ενεργοποίηση των  αντιστάσεων πρόσδεσης σε
OUT PORTD,R12               ; υψηλή τάση όλων των pin της θύρας
IN R12, PIND                ; ανάγνωση των λογικών σταθμών στις εισόδους
    ; της θύρας PORTD και αποθήκευση αποτελέσματος στον R12
```

Έξοδος Δεδομένου: Εγγραφή σε θύρα με την εντολή 'out'

```
LDI R12, 0b00001111         ; τα bit 1ο - 4ο της θύρας PORTD γίνονται έξοδοι
OUT DDRD, R12
LDI R12, 0b00001010         ; οδήγηση των pin PD1 και PD3 σε λογική στάθμη 1
OUT PORTD, R12              ; και PD0= PD2=0
LDI R12, 0b00000101         ; οδήγηση των pin PD0 και PD2 σε λογική στάθμη 1
OUT PORTD, R12              ; και PD1= PD3=0
IN R13, PIND                ; ανάγνωση των pin εισόδου PD4 – PD7
; της θύρας PORTD και αποθήκευση αποτελέσματος στον R13
```

Πρόσβαση σε δεδομένα εντός της μνήμης προγράμματος

Συχνή είναι η χρήση δεικτών για πρόσβαση σε πίνακα δεδομένων εντός της μνήμης προγράμματος. Στο επόμενο παράδειγμα παρουσιάζεται ένας πίνακας με 8 τιμές-λέξεις (16-bit), όπου η έκτη τιμή διαβάζεται και αποθηκεύεται στους καταχωρητές R25:R24.

```
                                ; οι τιμές του πίνακα είναι οργανωμένες κατά λέξη
Table:
 .DW 0x1201,0x3423,0x5645,0x7867,0x9A89
 .DW 0xBCAB,0xDECD,0xF0EF
                                ; ακολουθεί η ανάγνωση της 6ης λέξης
    LDI ZH,HIGH(Table*2)        ; η διεύθυνση του πίνακα στον καταχωρητή Z.
    LDI ZL,LOW(Table*2)         ; πολλαπλασιασμός επί 2 για πρόσβαση κατά byte
    ADIW ZL,10                  ; δείχνουμε στο έκτο στοιχείο
    LPM                         ; ανάγνωση LSByte από μνήμη προγράμματος (0xAB)
    MOV R24,R0                  ; αντιγραφή LSByte στον καταχωρητή R24 (0xAB)
    ADIW ZL,1                   ; δείχνουμε στο MSByte στη μνήμη προγράμματος
    LPM                         ; ανάγνωση του MSByte (0xBC)
    MOV R25,R0                  ; αντιγραφή MSByte στον καταχωρητή R25
```

Οι τιμές του Πίνακα

Οι διευθύνσεις του πίνακα των 16bit δεδομένων στη *Μνήμη Προγράμματος*

	Διεύθυνση	Τιμές
Table	Table*2	01
	Table*2+1	12
Table+1	Table*2+2	23
	Table*2+3	34
Table+2	Table*2+4	45
	Table*2+5	56
Table+3	Table*2+6	67
	Table*2+7	78
Table+4	Table*2+8	89
	Table*2+9	9A
Table+5	Table*2+10	AB
	Table*2+11	BC
Table+6	Table*2+11	CD
	Table*2+11	DE
Table+7	Table*2+11	EF
	Table*2+15	F0

2. Πρόγραμμα που υπολογίζει το τετράγωνο ενός αριθμού με βάση έναν πίνακα δεδομένων, που περιέχει τα τετράγωνα των αριθμών, εντός της μνήμης προγράμματος. Ο αριθμός ανήκει στο διάστημα [0,15].

```
start:
    clr     temp                ; Όταν ένα bit του καταχωρητή κατεύθυνσης DDRD
    out     DDRD, temp          ; είναι 0, αυτή η θέση γίνεται είσοδος.
                                ; ο assembler δεν είναι case sensitive
    LDI ZH, HIGH(Table*2)       ; η διεύθυνση του πίνακα στον καταχωρητή Z
    LDI ZL, LOW(Table*2)        ; πολλαπλασιασμός επί δύο για πρόσβαση κατά byte
                                ; όπου zl=R30, zh=R31
    in r21,PIND                 ; ανάγνωση αριθμού, έστω από θύρα D
    add zl, r21                 ; πρόσβαση στο σωστό στοιχείο
    adc zh, temp
                                ; χρήση lpm για πρόσβαση στη μνήμη προγράμματος.
    lpm                         ; Το περιεχόμενο της θέσης μνήμης προγράμματος που
                                ; δείχνει ο Z φορτώνεται στον καταχωρητή R0. Δηλ. R0 ← (Z)
    mov r22,r0                  ; μεταφορά αποτελέσματος στον r22
    rjmp start                  ; Άλμα στη διεύθυνση start για ατέρμονα λειτουργία

; Οι οδηγίες .DB και .DW προς μεταφραστή χρησιμοποιούνται για εισαγωγή πίνακα
; δεδομένων στη μνήμη προγράμματος.
Table:                          ; οι τιμές του πίνακα οργανωμένες κατά λέξη
.DW 0x0100,0x0904,0x1910,0x3124,0x5140
.DW 0x7964,0xA990,0xE1C4
```

Οι τιμές του Πίνακα των τετραγώνων

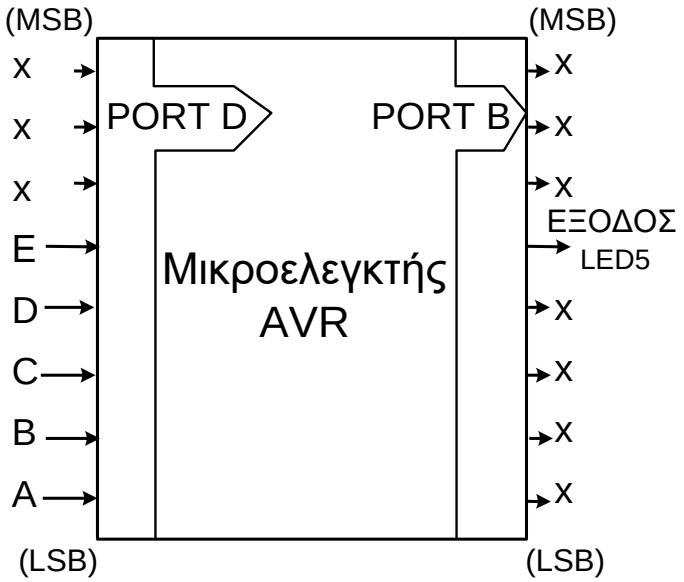
	Διεύθυνση	Τιμές
Table		00
		01
Table+1		04
		09
Table+2		10
		19
Table+3		24
		31
Table+4		40
		51
Table+5		64
		79
Table+6		90
		A9
Table+7		C4
		E1

3. Παράδειγμα εντολών σε επίπεδο bit

Στο παράδειγμα αυτό γίνεται η παρουσίαση εντολών που επιτρέπουν τον χειρισμό σε επίπεδο bit, χρήσιμα στην επίλυση προβλημάτων συνδυαστικής λογικής. Πιο συγκεκριμένα υποθέτουμε ότι οι λογικές μεταβλητές A, B, C, D και E βρίσκονται συνδεδεμένες στα 5 πρώτα LSB της θύρας PORTD. Να δοθεί το assembly πρόγραμμα που υλοποιεί τη λογική εξίσωση:

ΕΞΟΔΟΣ(LED5) = (A+B΄)*(C*D+E)

Το ένα bit της εξόδου παρέχεται από το 5ο bit της θύρας PORTB.




```
.include "m16def.inc"
.DEF A=r16              ; δήλωση καταχωρητών
.DEF BN=r17             ; συμπλήρωμα
.DEF C=r18
.DEF D=r19
.DEF E=r20
.DEF temp=r21
.cseg
.org 0                  ; διεύθυνση εκκίνησης
```

3. Πρόγραμμα που υλοποιεί τη λογική συνάρτηση: ΕΞΟΔΟΣ(LED5) = (A+B΄)*(C*D+E)

```
start:  clr temp            ; θύρα D ως είσοδος
   out DDRD,temp
   ser temp
   out PORTD,temp           ; pull-up θύρας D
   out DDRB,temp            ; θύρα B ως έξοδος
again:  in temp, PIND       ; ανάγνωση ακροδεκτών PORTD
   mov A, temp              ; το A στο LSB του καταχωρητή A
   lsr temp
   mov BN, temp             ; το B στο LSB του καταχωρητή BN
   com BN                   ; συμπλήρωμα B
   lsr temp
   mov C, temp              ; το C στο LSB του καταχωρητή C
   lsr temp
   mov D, temp              ; το D στο LSB του καταχωρητή D
   lsr temp
   mov E, temp              ; το E στο LSB του καταχωρητή E
   or A, BN                 ; A=A+B΄
   and C,D                  ; C=C·D
   or C,E                   ; C=C·D+E
   and A,C                  ; υλοποίηση συνδυαστικής λογικής A=(A+B΄)·(C·D+E)
   andi A, 1                ; απομόνωση του LSB
   lsl  A (x4)          ; 4 ολισθήσεις αριστερά για να έρθει το αποτέλεσμα στη σωστή θέση
   out PORTB,A              ; έξοδος αποτελέσματος
   rjmp again               ; άλμα στη διεύθυνση again για επανάληψη
```

3. Εναλλακτική υλοποίηση της λογικής συνάρτησης: ΕΞΟΔΟΣ(LED5) = {A΄·B +C΄·E΄+ D΄·E΄}΄ ή A·C·D+A·E +B΄·C·D+B΄·E

```
AGAIN:
   in temp, PIND       ; ανάγνωση ακροδεκτών PORTD
   mov r16, temp       ; το A στο LSB του καταχωρητή A
   andi r16, 0x03      ; απομόνωση των A και B
   cpi r16,0x02        ; συγκρίνεται για A=0 και B=1. Μόνο τότε A΄·B=1
   breq ZERO           ; Αν ισχύει F=0, άλμα στη διεύθυνση ZERO
   mov r16, temp       ; Αλλιώς ελέγχω τον επόμενο όρο C΄·E΄
   andi r16, 0x14      ; απομόνωση των C και E. Αν C=E=0, C΄·E΄=1
   breq ZERO           ; Αν ισχύει F=0, άλμα στη διεύθυνση ZERO
   mov r16, temp       ; Αλλιώς ελέγχω αν ο τελευταίος όρος D΄·E΄=1
   andi r16, 0x18      ; απομόνωση των D και E. Αν D=E=0, D΄·E΄=1
   breq ZERO           ; Αν ισχύει F=0, άλμα στη διεύθυνση ZERO
   ldi r16, 0x10       ; Αλλιώς F=1.
   rjmp ONE
ZERO:
   clr r16             ; Αν είναι από άλμα F=0. Ακολουθεί το κοινό τμήμα
ONE:
   out PORTB, r16      ; Έξοδο αποτελέσματος στο PortB.4
   rjmp AGAIN          ; άλμα στη διεύθυνση AGAIN για επανάληψη
```

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.

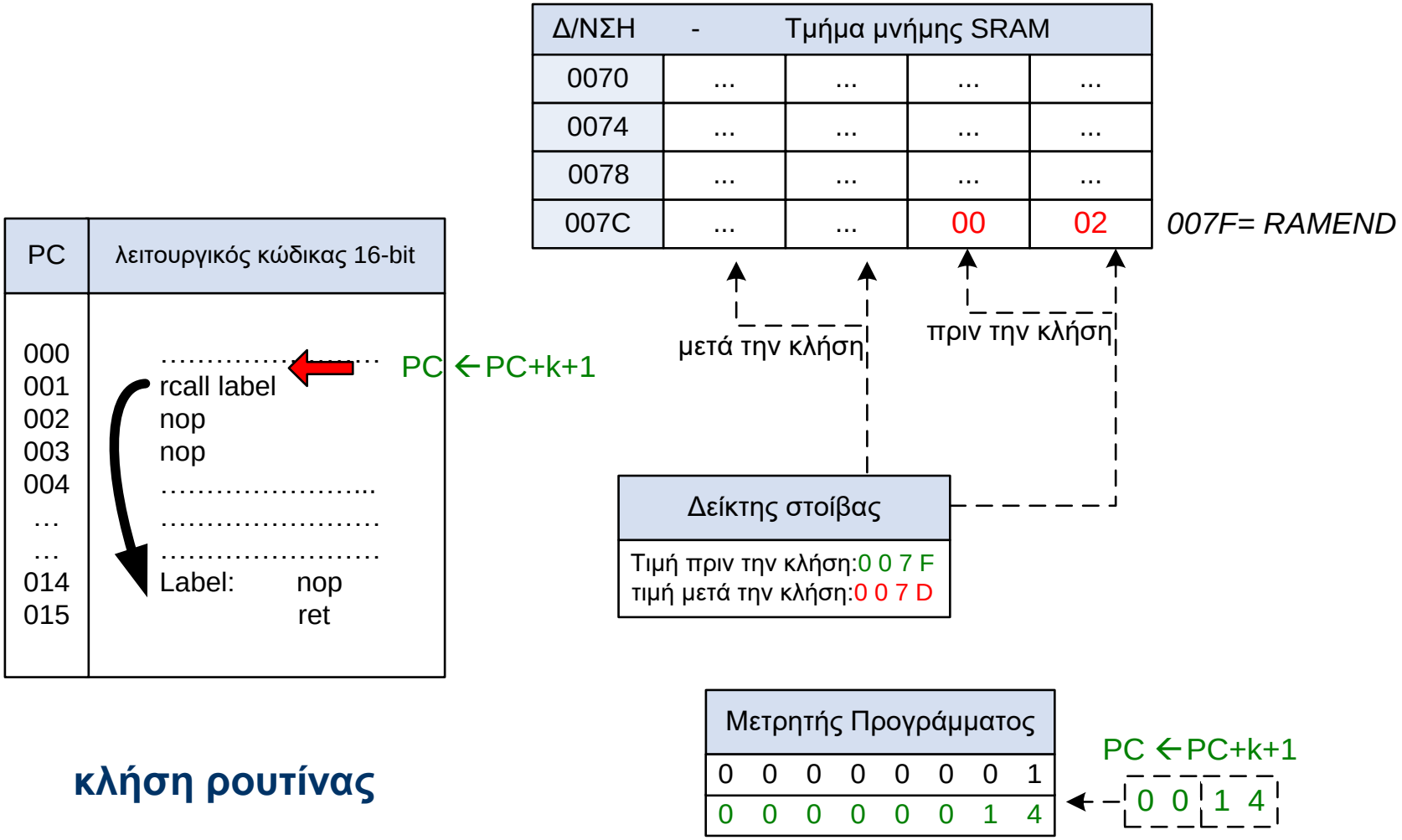
εντολές διακλάδωσης & παράκαμψης			
BRBS/BRBC	s,k	Branch if Status Flag 1/0	If (SREG(s)=1) then PC←PC+k+1
BREQ/BRNE	k	Branch if Equal/ Not Equal	if (Z = 1) then PC ← PC + k + 1
BRCS/BRCC	k	Branch if Carry 1/0	if (C = 1) then PC ← PC + k + 1
BRSH/BRLO	k	Branch if Same or >/ <	if (C = 0) then PC ← PC + k + 1
BRGE/BRLT	k	Branch if > or =, Signed	if (NV= 0) then PC ← PC + k+ 1
BRHS/BRHC	k	Branch if Half_Carry 1/0	if (H = 1) then PC ← PC + k + 1
BRTS/ BRTC	k	Branch if T Flag 1/0	if (T = 1) then PC ← PC + k + 1
BRVS/BRVC	k	Branch if Overflow Flag1/0	if (V = 1) then PC ← PC + k + 1
BRIE/ BRID	k	Branch if Int 1/0 Disabled	if (I = 1) then PC ← PC + k + 1
SBRC/SBRS	Rr, b	Skip if Bit in Reg 0/1	if (Rr(b)=0) PC ← PC + 2 or 3
SBIC/ SBIS	P, b	Skip if Bit in I/O Reg 0/1	if (P(b)=0) PC ← 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.

RJMP	k	Relative Jump	PC ← PC+ k+1
IJMP		Indirect Jump to (Z)	PC ← Z
JMP	k	Direct Jump	PC ← k
RCALL	k	Relative Subroutine Call	PC ← PC+k+1
ICALL		Indirect Call to (Z) - Η διεύθυνση επιστροφής (PC) αποθηκεύεται στη στοίβα.	PC ← Z STACK ← PC
CALL	k	Direct Subroutine Call - Η διεύθυνση επιστροφής (PC) αποθηκεύεται στη στοίβα.	PC ← k, STACK ← PC
RET		Subroutine Return	PC ← STACK
RETI		Interrupt Return	PC ← STACK
CP	Rd, Rr	Compare	Rd < Rr
CPI	Rd, K	Compare Reg with Immediate	Rd < K

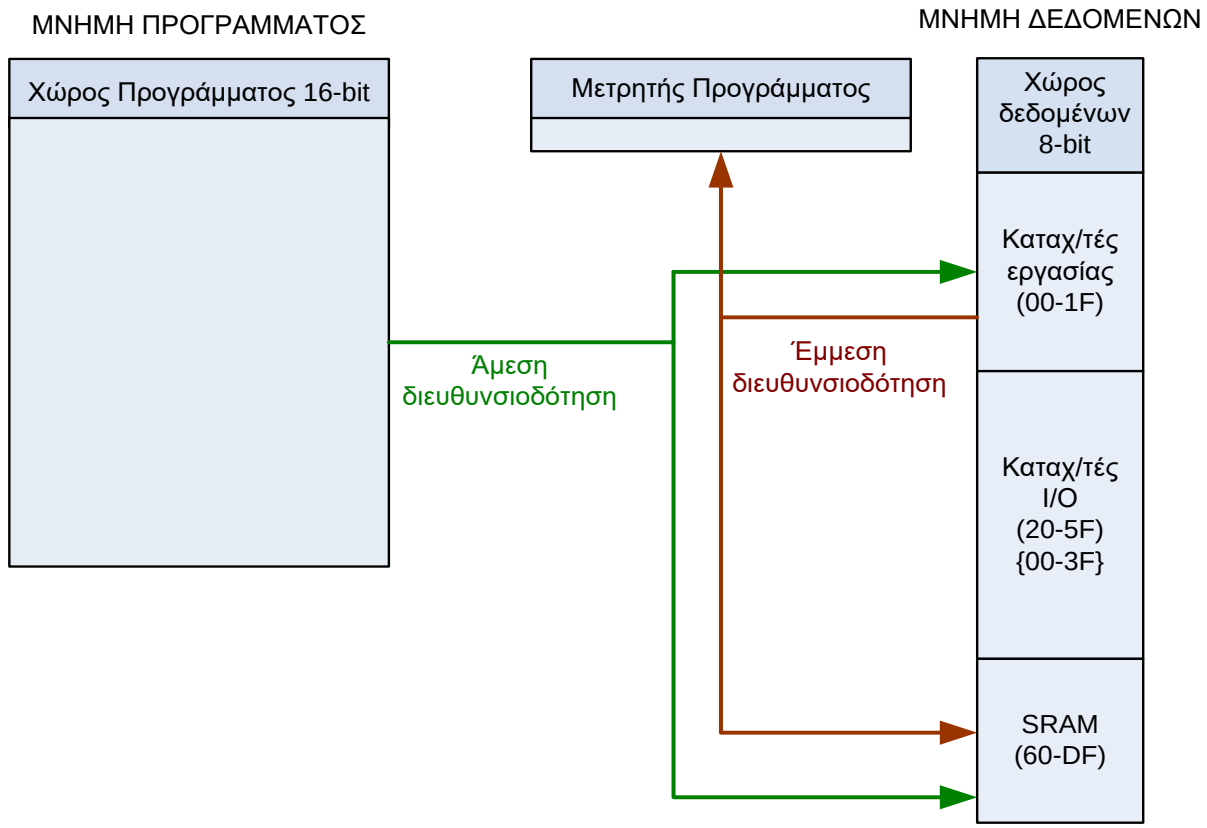
Παράδειγμα άλματος-ρουτινών



κλήση ρουτίνας

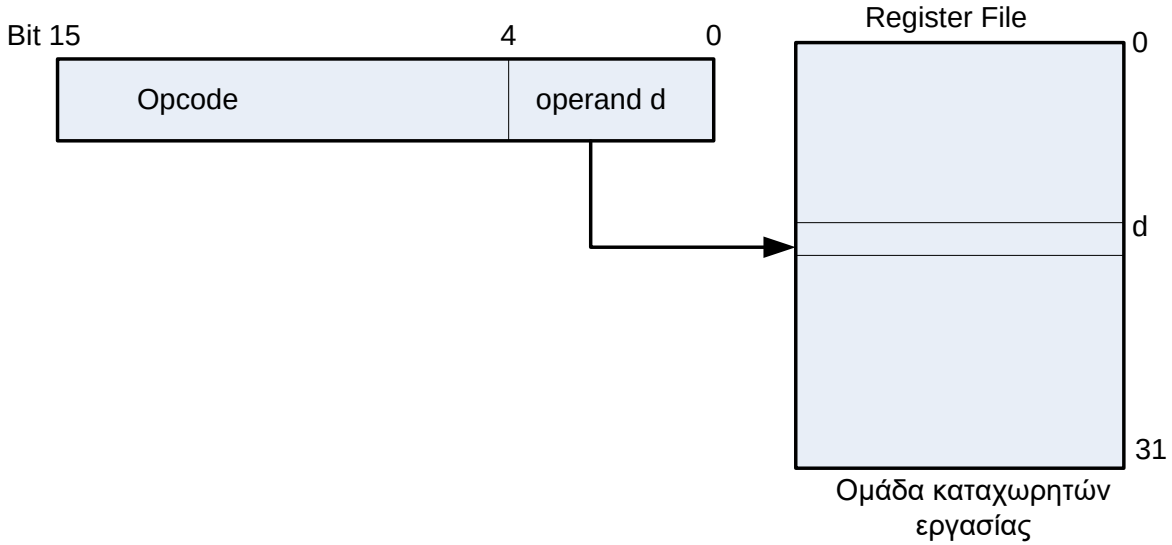
Διευθυνσιοδότηση μΕ AVR

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

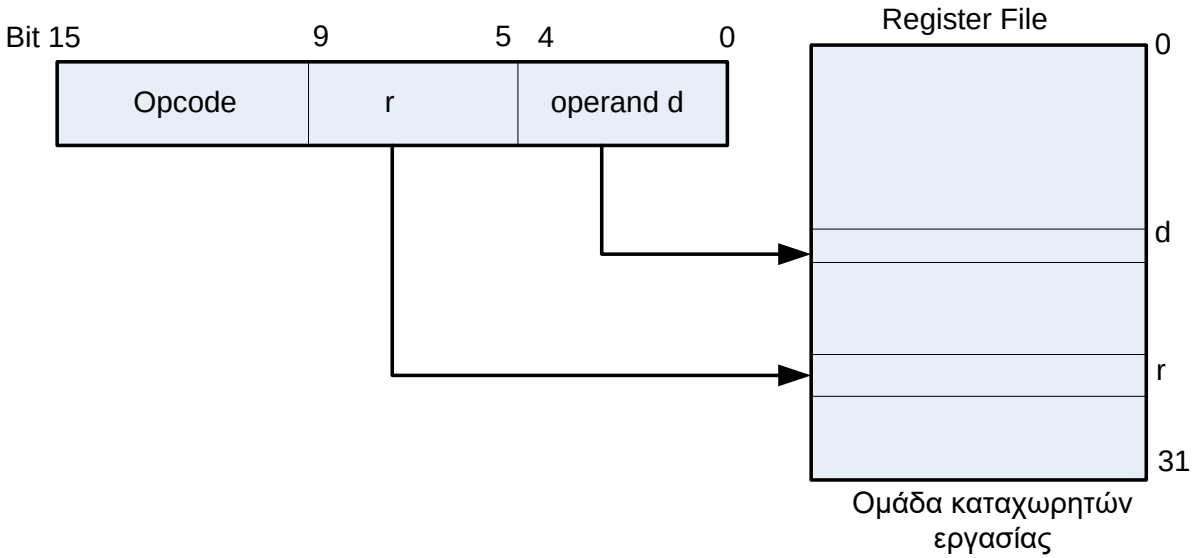
Η εντολή διαβάζει τα περιεχόμενα του καταχωρητή, εκτελεί τις πράξεις με αυτά και αποθηκεύει το αποτέλεσμα στον ίδιο καταχωρητή. Ο καταχωρητής είναι ένας από τους 32 καταχωρητές εργασίας R0-R31 που διαθέτουν το αντίστοιχο τμήμα μνήμης (register file).



Παραδείγματα τέτοιων εντολών είναι:
INC Rd: Αύξηση των περιεχομένων του καταχωρητή Rd κατά μία μονάδα.

Άμεση διευθυνσιοδότηση δύο καταχωρητών

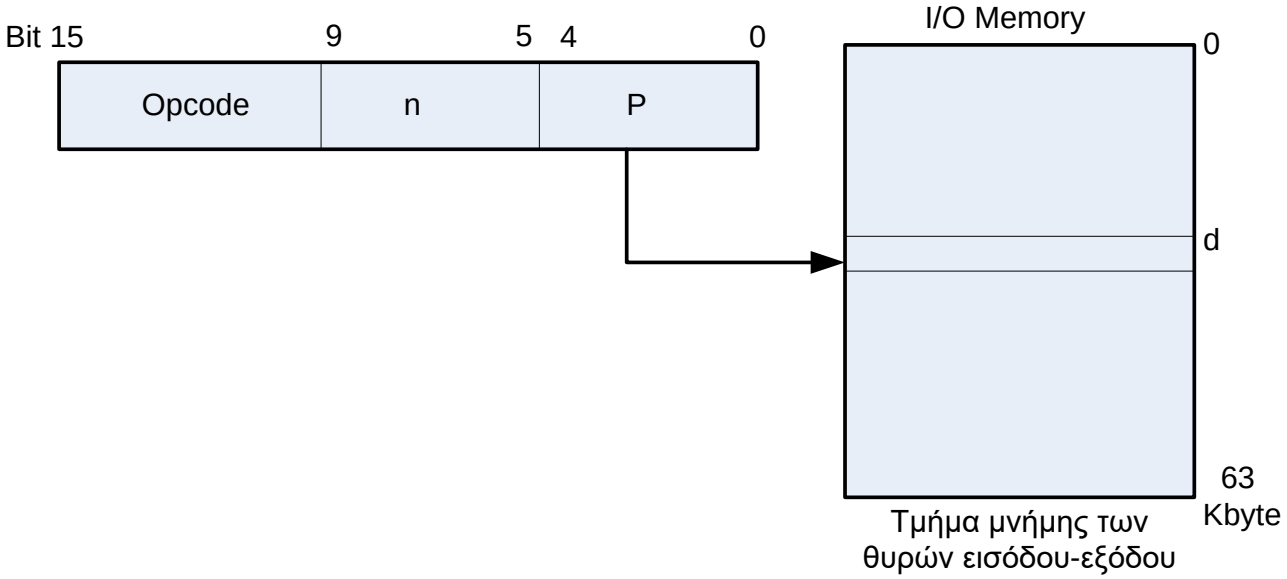
Η εντολή διαβάζει τα περιεχόμενα των δυο καταχωρητών, εκτελεί την πράξη μεταξύ των δεδομένων και αποθηκεύει το αποτέλεσμα στον καταχωρητή προορισμού Rd.



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

Απευθείας διευθ/τηση των θυρών εισόδου-εξόδου

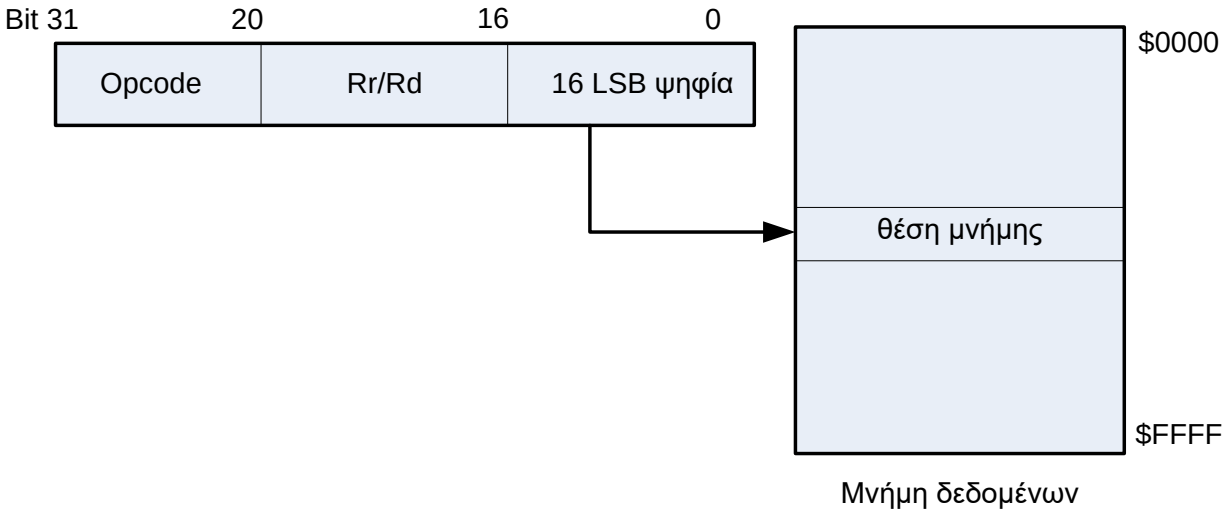
Η πρόσβαση στις θέσεις μνήμης που αντιστοιχούν οι θύρες εισόδου-εξόδου (I/O memory) επιτυγχάνεται με 2 εντολές: της IN Rd, PortAddress και της OUT PortAddress, Rr όπου PortAddress είναι η διεύθυνση του καταχωρητή εισόδου-εξόδου.



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

Άμεση διευθυνσιοδότηση μνήμης δεδομένων

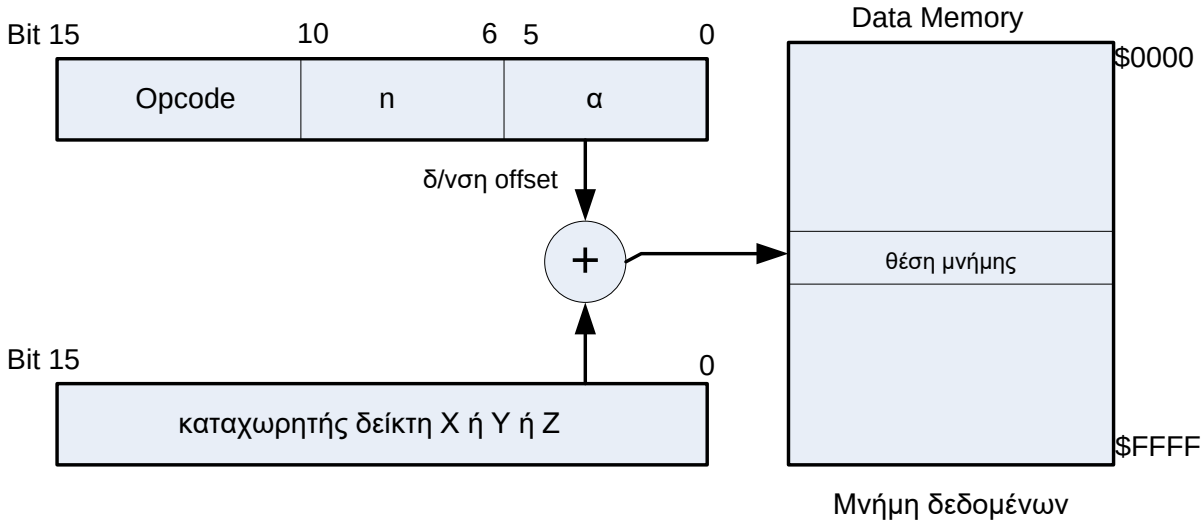
Η πρόσβαση στις θέσεις μνήμης επιτυγχάνεται με εντολές μήκους 2 λέξεων, όπου η μία λέξη (16 bits) αντιστοιχεί στη διεύθυνση της μνήμης δεδομένων. Άρα, ο χώρος μνήμης όπου επιτυγχάνεται πρόσβαση είναι 64Kbyte.



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

Έμμεση διευθυνσιοδότηση μνήμης δεδομένων

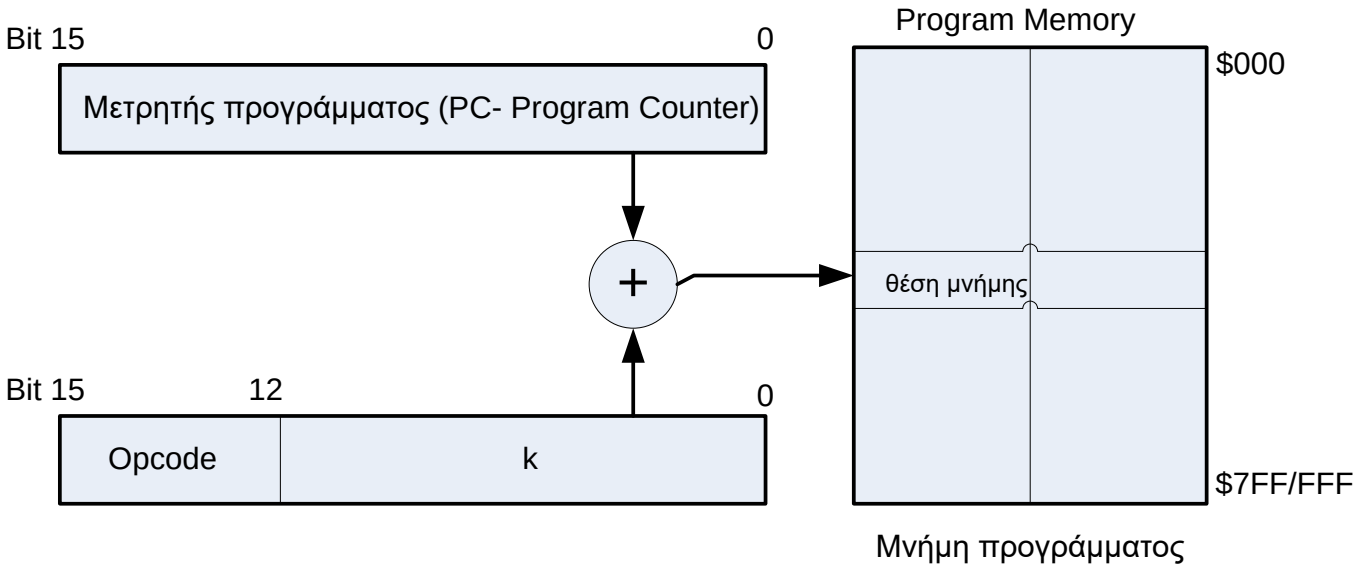
Οι εντολές αυτές έχουν μήκος 1 λέξης και χρησιμοποιούν έναν κατ/τή δείκτη (X,Y,Z), το περιεχόμενο του οποίου αντιστοιχεί στη διεύθυνση της μνήμης δεδομένων. Ο κατ/τής δείκτη X,Y,Z μπορεί να *αυξηθεί κατά μία μονάδα* μετά την ανάθεση της τιμής στον κατ/τη Rr, ώστε να δείχνει σε επόμενη θέση μνήμης ή να *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+ ST X+,Rr LD Rr,-Z ST –X,Rd LDD Rd,Y+2 ST Y+6,Rd

Σχετική διευθυνσιοδότηση μνήμης προγράμματος

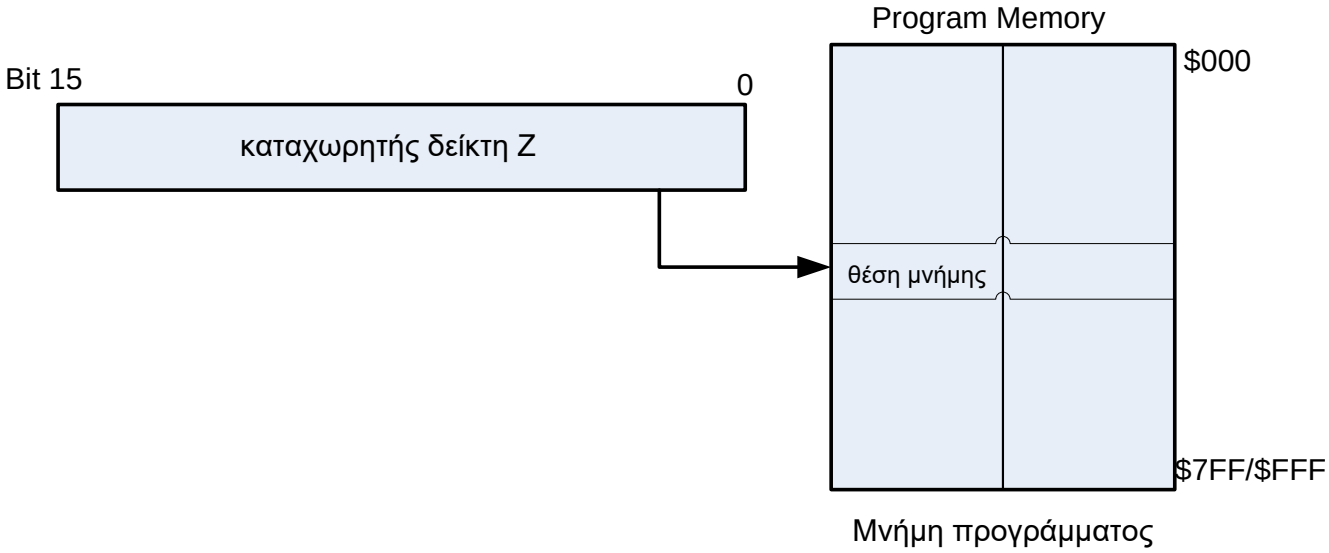
Οι εντολές αυτές επιτυγχάνουν πρόσβαση σε χώρο μνήμης προγράμματος και είναι του τύπου RCALL, RJMP όπου χρησιμοποιείται μια μετατόπιση +/- 2K στο περιεχόμενο του μετρητή προγράμματος.



Παραδείγματα : RCALL, RJMP

Έμμεση διευθυνσιοδότηση μνήμης προγράμματος

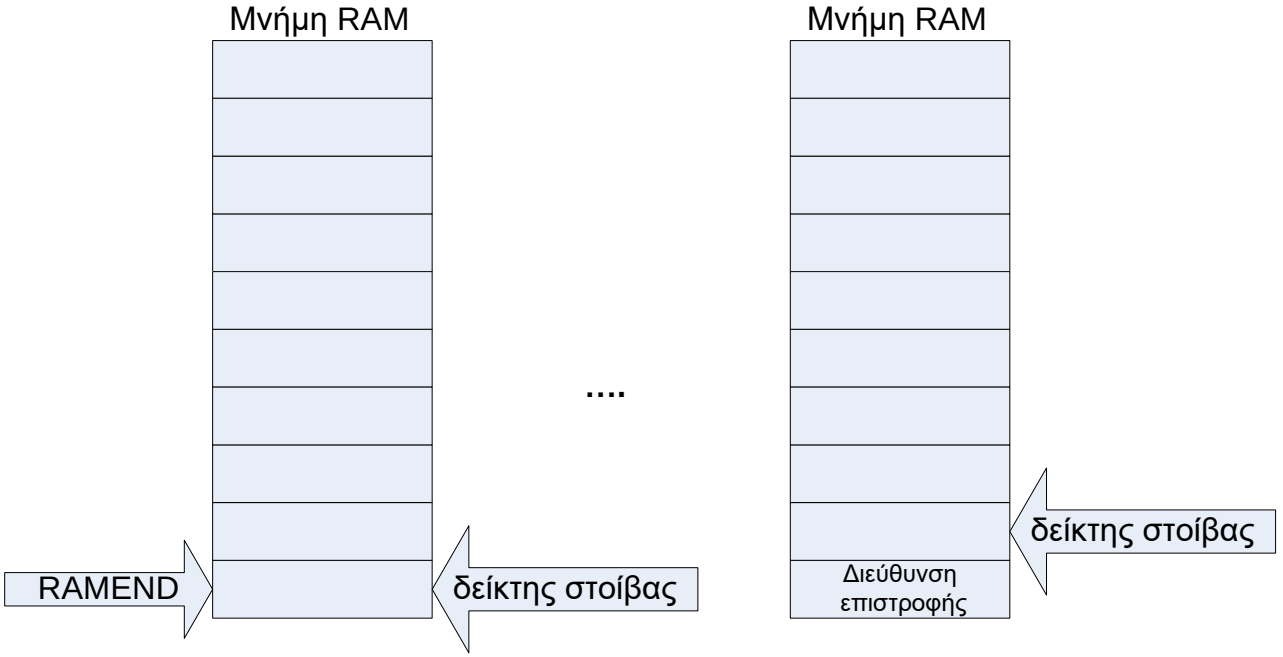
Οι εντολές αυτές επιτυγχάνουν πρόσβαση σε χώρο μνήμης προγράμματος έως 64Kbytes με χρήση του καταχωρητή Z, ως δείκτη μιας θέσης μνήμης προγράμματος.



Παραδείγματα : ICALL , IJMP

Στοίβα

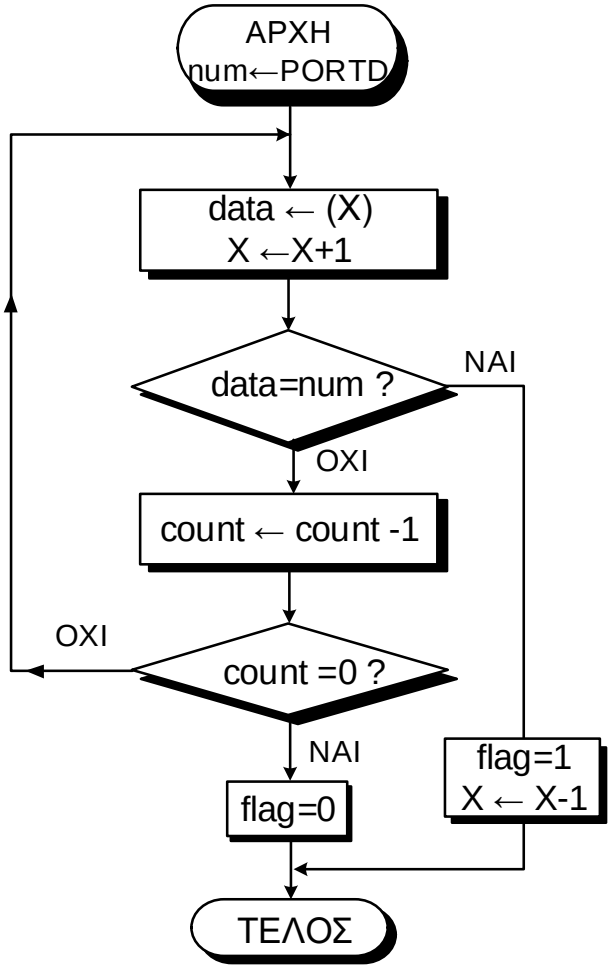
Η στοίβα χρησιμοποιείται από την αριθμητική λογική μονάδα (ALU) για αποθήκευση διευθύνσεων επιστροφής από ρουτίνες διακοπής-υπορουτίνες. Η στοίβα χρειάζεται έναν δείκτη στοίβας (SP) και χώρο μνήμης στην SRAM. Πρόκειται, λοιπόν, για δομή τύπου Last-In-First-Out (LIFO). Μια διεύθυνση SRAM έχει μήκος 16 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 4ο : 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).

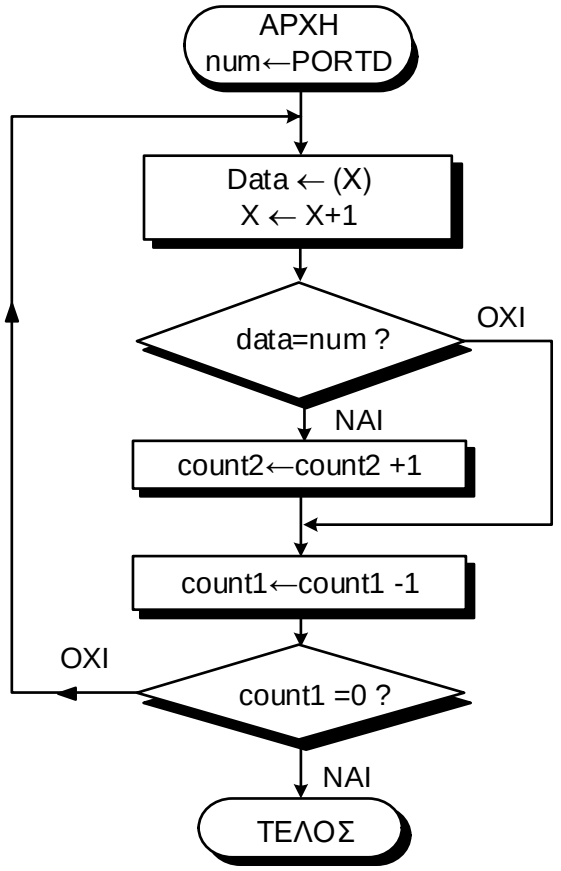


Το πρόγραμμα βρίσκει αν η τιμή που δίνεται από τον χρήστη από το PORTD είναι σε περιοχή της μνήμης προγράμματος. Αν ναι, επιστρέφει τη διεύθυνση, αλλιώς θέτει 0 στον r22.

```
.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:   ld data,x+           ; φόρτωση θέσης X μνήμης δεδομένων και αύξηση
                             ; δείκτη για πρόσβαση στον επόμενο κύκλο σε επόμενη θέση
        cp data,num          ; σύγκριση του εισαγόμενου αριθμού με θέση μνήμης
        breq found           ; 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:                       ; η τιμή βρέθηκε στον πίνακα οπότε
        ldi flag,1           ; θέτουμε τον r22 και επιστρέφουμε την αντίστοιχη
        sbiw r26,1           ; διεύθυνση στον X= r27 : r26 αφού το μειώσουμε κατά 1
 end:
```

Παράδειγμα 5ο

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).

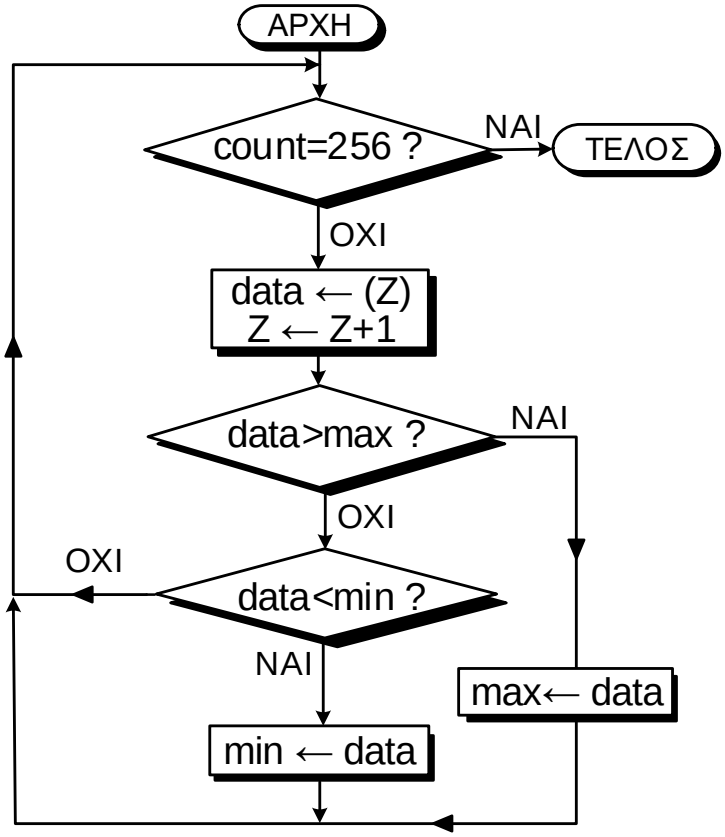


Παράδειγμα 5ο: Αναζήτηση ενός χαρακτήρα

```
.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 6ο

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



Παράδειγμα 6ο : Το πρόγραμμα εντοπίζει τον μέγιστο και τον ελάχιστο

```
.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 6ο - 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: Χρήση της εντολής LPM

Στο παράδειγμα αυτό επιδεικνύεται η χρήση της εντολής LPM για ανάγνωση bytes από πίνακα δεδομένων στο τμήμα κώδικα (code segment). Συγκεκριμένα, διαβάζονται διακόπτες που υποθέτουμε ότι είναι συνδεδεμένοι στο PORTD και ανάβουν τα leds (PORTB ως έξοδος των Leds) που αντιστοιχούν στα δεδομένα του πίνακα δεδομένων στο τμήμα κώδικα (δηλαδή: sw0 ανάβει led0, sw1 ανάβει led0 και led1, sw2 ανάβει τα led0, led1 και led2, κοκ) :

0 => 0

1 => 0,1

2 => 0,1,2

3 => 0,1,2,3

κλπ.

7=> 0,1,2,3,4,5,6,7

Απλούστερα θα μπορούσε να υπολογιστεί από τη σχέση: led=2*sw-1

Παράδειγμα 7ο: Χρήση της εντολής LPM (1)

```
.INCLUDE "m16def.inc"      ; δήλωση μικροελεγκτή
.LIST
.DEF   reg=R0              ; η εντολή LPM χειρίζεται τον καταχωρητή R0
.DEF   temp=R16
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
                           ; χρήση καταχωρητή δείκτη Z: {ZL (R30) και ZH (R31)}
                           ; που έχει οριστεί στο αρχείο m16def.inc

loop:
      ldi ZL,LOW(array)    ; ο Z δείχνει στο 1ο byte (FF) της λίστας
      ldi ZH,HIGH(array)
      in   temp,PIND       ; ανάγνωση διακοπτών
      cpi temp,0xFF        ; αν όλοι οι διακόπτες off, τότε όλα τα Leds off
      breq diavasma        ; άλμα στο diavasma αν όλοι οι διακόπτες είναι off
inc_pointer:               ; Προσοχή στην αρνητική λογική των διακοπτών!
      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  (επίσης η εντολή brcs) .
```

Παράδειγμα 7ο: Χρήση της εντολής LPM (2)

```
diavasma:               ; Όταν C=0 γίνεται ανάγνωση του byte της λίστας.
      lpm               ; Η θέση που δείχνει ο Z  εγγράφεται στον R0.
      out PORTB,reg     ; Μεταφορά του δεδομένου στα leds.
      rjmp loop
       ; Η χρήση του  .DB xx στη μνήμη προγράμματος πάντα προσθέτει ένα μηδενικό byte.
       ; Γι' αυτό προτιμάται η χρήση του .DW ;xx,yy (2 δεδομένα ενώνονται σε μια λέξη).
       ; Αντίστοιχα ισχύει σε κείμενο χαρακτήρων, δηλ. .DB "atmel" αποδοτικότερα
       ; εγγράφεται ως  .DW.  Αν ο αριθμός των bytes ή χαρακτήρων είναι περιττός θα
       ; πρέπει να συμπληρωθεί η λίστα με ένα μηδενικό.

       ; Πίνακας με τους συνδυασμούς των leds, όπου κάθε byte αντιστοιχεί σε ένα
       ; διακόπτη. Οι τιμές τοποθετούνται κατά λέξεις για αποδοτικότερη χρήση της μνήμης
table:
.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
       ; Όταν προσθέτουμε μια λέξη όπως FEFFh σε λίστα, το 1ο byte της λίστας που
       ; λαμβάνεται μέσω της εντολής LPM είναι το LSB (FFh), και όχι το MSB (FEh).
       ; Η πρόσβαση στη λίστα γίνεται κατά byte, ενώ οι διευθ. οργανώνονται κατά λέξεις
.EQU   array=table*2
```

8ο Αναβοσβήνει τα leds παρεμβάλλοντας χρονική καθυστέρηση σε κάθε κατάσταση, για να γίνουν ορατές οι αλλαγές:

```
.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 9ο: Χειρισμός στα leds με βάση τη τιμή των switches

Στο παράδειγμα γίνεται χειρισμός στα leds με βάση τη τιμή των switches που συνδέονται στις θύρες PORTB και PORTD αντίστοιχα. Η συμπεριφορά των leds καθορίζεται με βάση ποιος διακόπτης πιέζεται ως εξής:
Υποθέτουμε ότι δεν πιέζονται ταυτόχρονα περισσότεροι του ενός διακόπτες. Επίσης έχει τεθεί αρχική κατάσταση OFF για όλα τα LEDS.

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

Παράδειγμα 9ο : 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
                            ; Έλεγχος input/output
```

Παράδειγμα 9ο : Κύριο πρόγραμμα

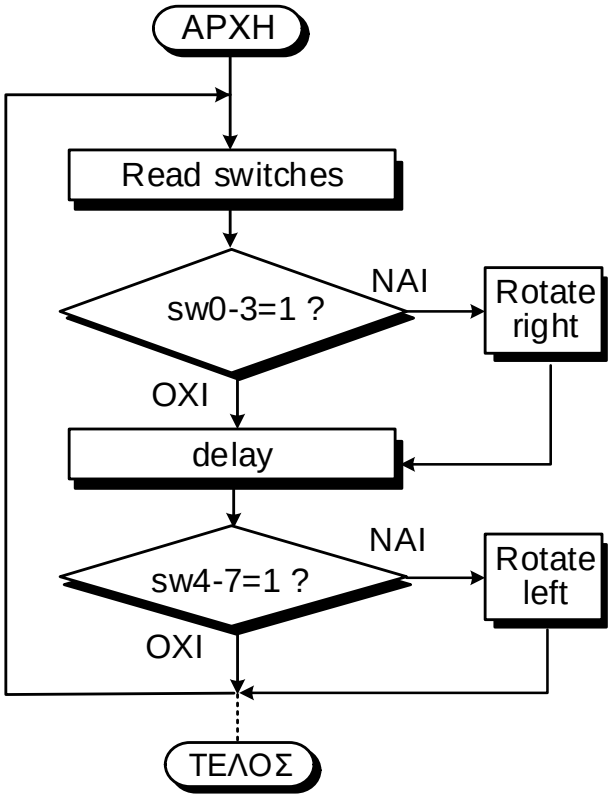
```
LOOP:
    sbic PIND, 0x00        ; Αν (Port D, pin0 = 1) τότε
    inc Temp                    ; μέτρηση LEDS μια μονάδα προς τα κάτω
    sbic PIND, 0x01        ; Αν (Port D, pin1 = 1) τότε
    dec Temp                    ; μέτρηση LEDS μια μονάδα προς τα πάνω
    sbic PIND, 0x02        ; Αν (Port D, pin2 = 1) τότε
    ror Temp                    ; ολίσθηση LEDS μία θέση δεξιά
    sbic PIND, 0x03        ; Αν (Port D, pin3 = 1) τότε
    rol Temp                    ; ολίσθηση LEDS μία θέση αριστερά
    sbic PIND, 0x04        ; Αν (Port D, pin4 = 1) τότε
    com Temp                    ; αντιστροφή LEDS
    sbic PIND, 0x05        ; Αν (Port D, pin5 = 1) τότε
    neg Temp                    ; αντιστροφή LEDS και πρόσθεση 1
    sbic PIND, 0x06        ; Αν (Port D, pin6 = 1) τότε
    swap Temp                   ; εναλλαγή των τμημάτων  high και low των LEDS
    sbic PIND, 0x07        ; Αν (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 10ο : Χειρισμός διακοπτών εισόδου και led εξόδου - 2

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 περιστρέφει 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 10ο : Χειρισμός διακοπτών εισόδου και led εξόδου - 2

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

Παράδειγμα 11ο : 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 11ο : Χειρισμός θυρών I/O (1)

```
.include "m16def.inc"
                                ; ορίζουμε ονόματα καταχωρητών
.def temp=r16                   ; καταχωρητής για προσωρινή αποθήκευση
.def port_value=r17             ; αποθήκευση τιμής θύρας
.def reg = r18
.org 0x000
     rjmp main                  ; παράκαμψη διανυσμάτων διακοπών
.org 0x100
main:                           ; κυρίως πρόγραμμα
     ldi  reg,LOW(RAMEND)       ; αρχικοποίηση δείκτη στοίβας
     out  SPL,reg
     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 11ο : Χειρισμός θυρών I/O (2)

```
    clr xh                  ; δήλωση διεύθυνσης αποθήκευσης στην SRAM
    ldi xl, 0x60            ; θέτουμε διεύθυνση 0x60 στον καταχωρητή X
    in port_value, PIND     ; 1η ανάγνωση θύρας

diavasma:                   ; βρόχος ανάγνωσης
    in temp,PIND            ; διάβασμα τρέχουσας τιμής
    cp temp,port_value      ; έλεγχος εάν έχει αλλάξει η τιμή
    breq diavasma           ; ίδιες τιμές, επιστροφή στο diavasma για
                            ; ανάγνωση
    nop
    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 12ο: Μεταφορά δεδομένων

Μεταφορά ενός πίνακα των 20 bytes από το τμήμα κώδικα (cseg) στο τμήμα δεδομένων (dseg). Εξετάζεται κάθε στοιχείο του πίνακα. Εάν πρόκειται για περιττό αποθηκεύεται στη διεύθυνση $0060, ενώ αν πρόκειται για άρτιο 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)      ; Η διεύθυνση του πίνακα  δεδομένων
                              ; στη μνήμη προγράμματος στον Z
   ldi ZL,LOW(Table*2)
   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)
```

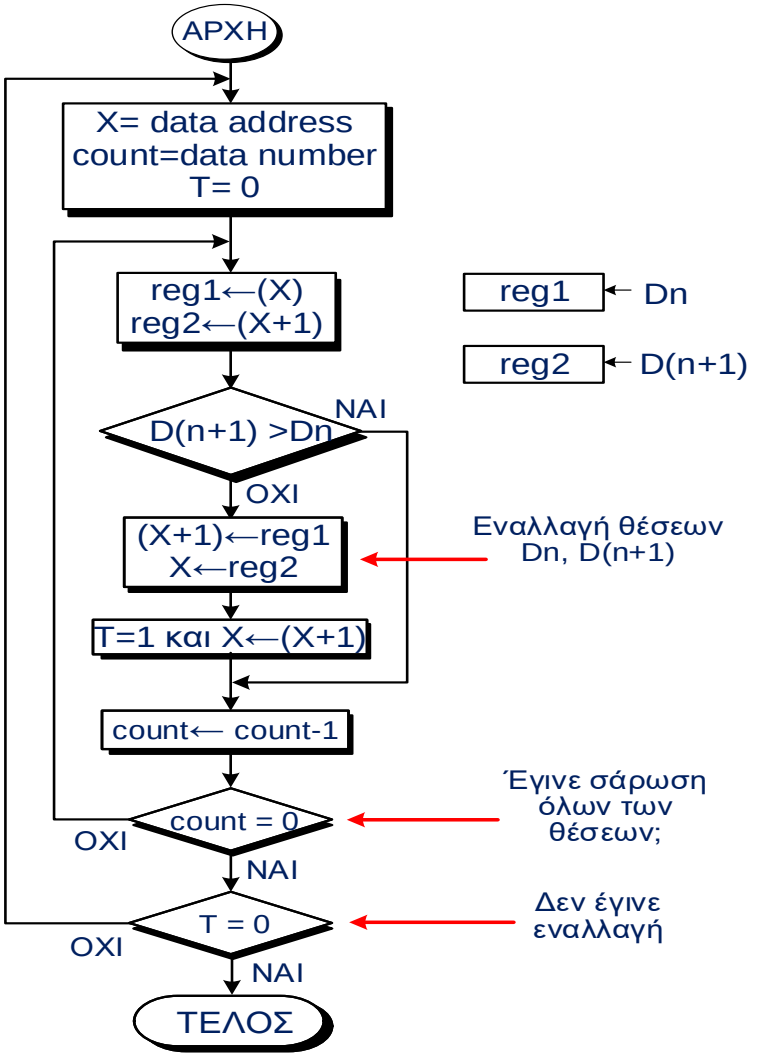
Παράδειγμα 12ο : Μεταφορά δεδομένων

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


.cseg                         ; Στη μνήμη προγράμματος εισάγεται
Table:                        ; πίνακας δεδομένων.
.DW 0x0100,0x0706,0x1713,0x3326,0x27C6
.DW 0x5042,0x7A61,0xA2F1,0xE0D7,0x89FD
end: .exit
```

Παράδειγμα 13ο : 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 13ο : Κατάταξη αριθμών

```
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
       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)
.exit
```

Παράδειγμα 14ο : Χειρισμός διακοπτών εισόδου και led εξόδου

Δίνεται πρόγραμμα που όταν ένα από τα switches 0 και 1 είναι πατημένο ανάβουν τα αντίστοιχα leds, ενώ όταν είναι πατημένο ένα από τα switches 2-6 ανάβουν όλα τα υπόλοιπα (leds 2 -7). Το switch 7 σβήνει όλα τα leds.

```
.INCLUDE "m16def.inc"        ; δήλωση ελεγκτή
.DEF   reg = R16             ; ορίζουμε όνομα καταχωρητή
rjmp   main                  ; εντολή άλματος στη διεύθυνση 0:
.org 0                       ; κυρίως πρόγραμμα
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
   out PORTD,reg             ; Port D (pull-ups)
   out DDRB,reg              ; PortB ως έξοδος
   out PORTB,reg             ; σβήσιμο leds (ανάστροφη λογική)
```

Παράδειγμα 14ο : Χειρισμός διακοπτών εισόδου και led εξόδου

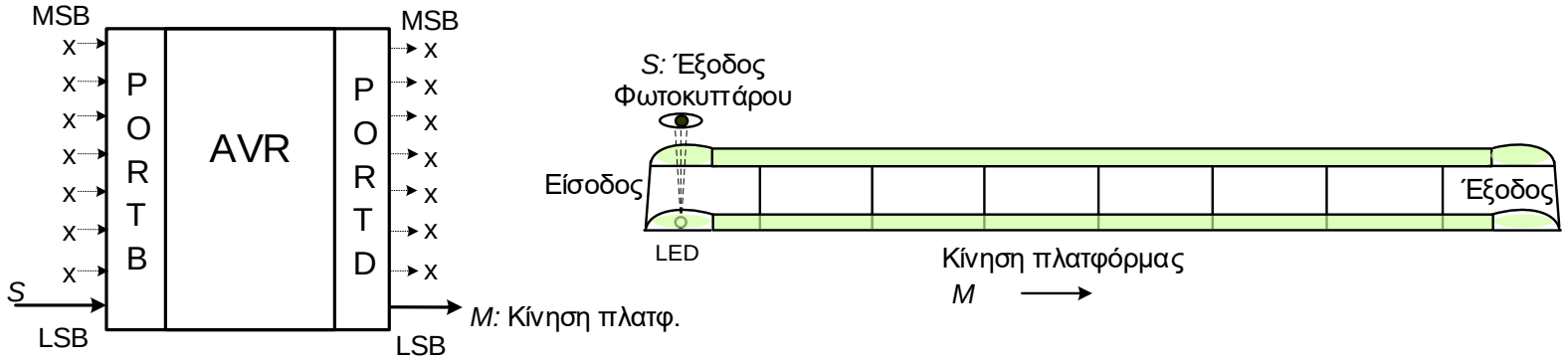
```
loop:
; Ανάγνωση switch 0, δηλαδή portD (PIND 0) και εντολή παράκαμψης (SBIS) της επόμενης
; εντολής  αν το  bit 0 είναι 1 που λόγω της ανάστροφης λογικής των διακοπτών
; σημαίνει ότι δεν πατήθηκε. Διαφορετικά κλήση (RCALL) της ρουτίνας Light0.

   sbis    PIND,0
   rcall    Light0
; Ανάγνωση switch 1 με ένα διαφορετικό τρόπο:
; Οι θύρες I/O αντιστοιχούν επίσης και σε χώρο της μνήμης SRAM.
; Η διεύθυνση SRAM είναι 32 bytes υψηλότερα από την αντίστοιχη διεύθυνση θύρας.
; Οπότε η πρόσβαση σε μια θύρα μπορεί να γίνει με εντολές διευθυνσιοδότησης της SRAM.

.EQU   mem_name=PIND + $20              ; διεύθυνση στην sram
; Ορίζουμε το ζεύγος καταχωρητών R27:R26 ως καταχωρητή δείκτη αυτής της θύρας.
; Με την LoaD φορτώνουμε την τιμή της θύρας σε καταχωρητή σαν να είναι ένα SRAM byte.

   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.
                        ; Αλλιώς παρακάμπτεται για να γίνει ο έλεγχος των Switches 2 ως 6
```

Παράδειγμα 14ο : Χειρισμός διακοπτών εισόδου και led εξόδου

```
   in reg, PIND              ; Ανάγνωση port D (Προσοχή στην ανάστροφη λογική των διακοπτών).
   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               ; Αλλιώς ανάγνωση τρέχουσας κατάστασης LEDs
   andi reg,0b00000011       ; άναμμα των LED 2 ως 7
   out PORTB, reg            ; χωρίς να πειραχθούν τα LED 0 και 1
sw7:
   in reg,PIND               ; ανάγνωση θύρας διακοπτών
   rol reg                   ; ολίσθηση 7ου bit στο κρατούμενο
   brcs endloop              ; αν 7ο bit είναι 1 (BRanch Carry Set) καμία αλλαγή.
   ldi reg, 0xFF             ; Αλλιώς (C=0) σβήσιμο όλων των LEDs.
   out PORTB, reg
endloop:    rjmp loop
Light0:                      ; υπορουτίνα Light0: LED 0 ON
   in reg,PIND               ; ανάγνωση τρέχουσας κατάστασης port B
   andi reg,0b11111110       ; άναμμα του 1ου LED τα υπόλοιπα μένουν
   out PORTB,reg             ; στην ίδια κατάσταση.
   ret
 Light1:                     ; Υπορουτίνα Light1: LED 1 ON
   in reg,PIND               ; ανάγνωση κατάστασης port B
   cbr reg,0b00000010        ; θέτει bit 2 στο μηδέν (η εντολή αυτή ισοδυναμεί
   out PORTB,reg             ; με andi reg, 0b11111101). Ανάβει το 2ο LED και τα
   ret                       ; υπόλοιπα LED μένουν στην ίδια κατάσταση.
```

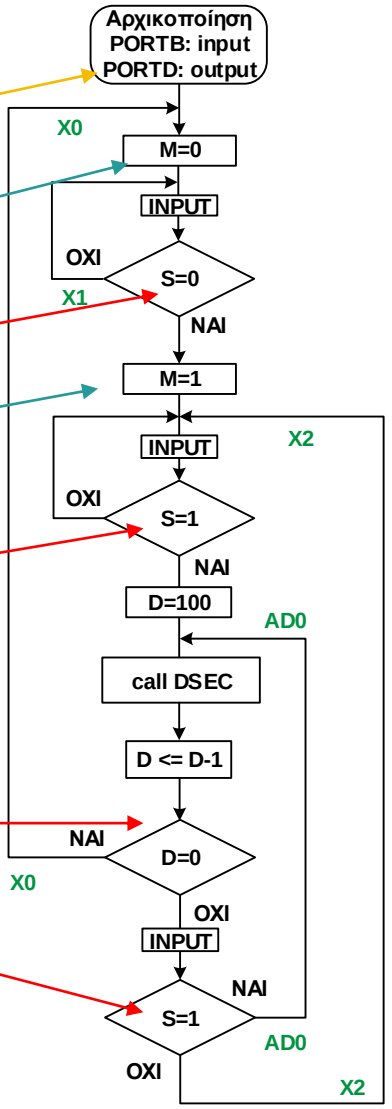
Παράδειγμα 15ο: Οδήγηση κυλιόμενης πλατφόρμας

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 ~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 15ο: Οδήγηση κυλιόμενης πλατφόρμας

```
        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 15ο : *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           ; ξανά εισαγωγή μακροεντολής (άλλα 4 INCs)
   Delay4             ; temp=0000 1000
   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 16ο : Χρήσης ετικέτας εντός και άλματος σε ετικέτα έξω από μακροεντολή

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

Παρ. 17ο : Χρήση παραμέτρων σε μακροεντολή (ίδιο με προηγούμενο αλλά με πέρασμα παραμέτρου)

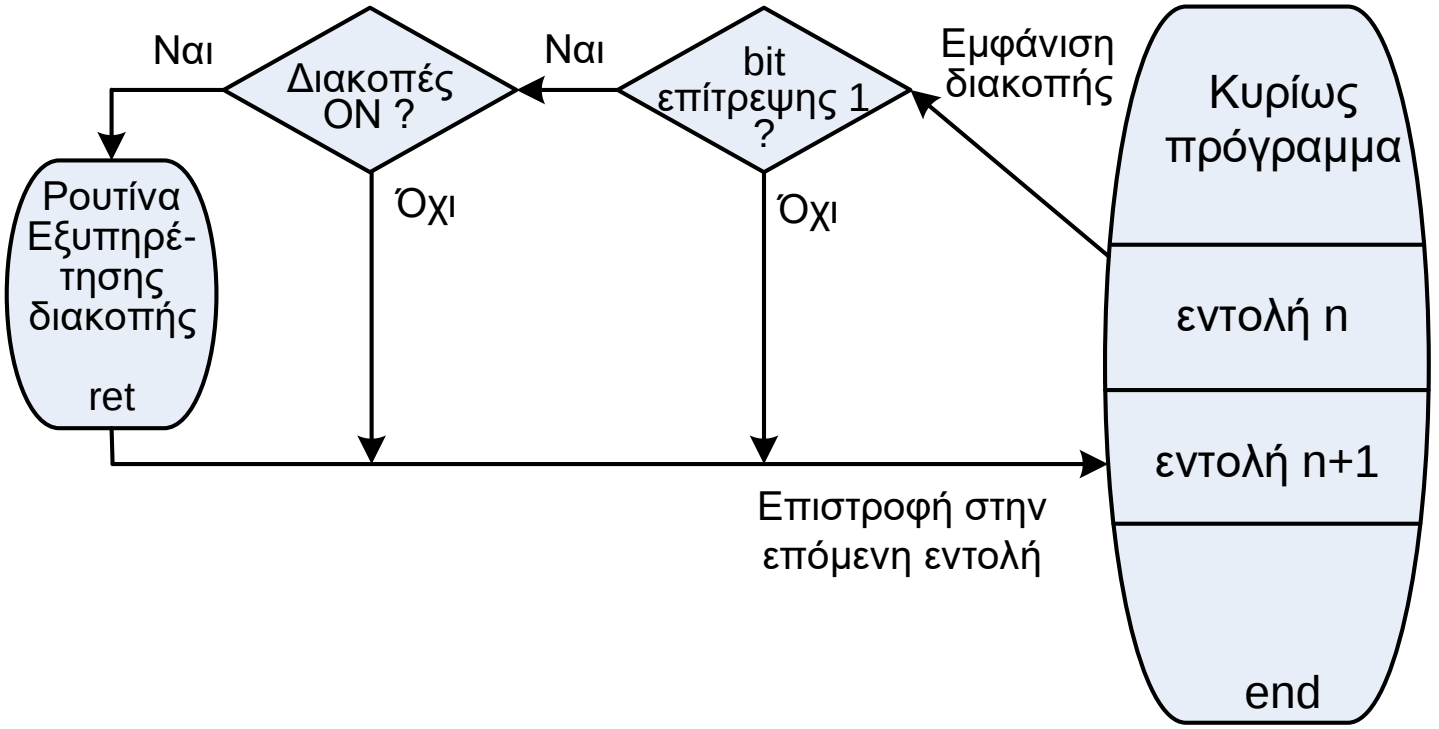
```
.INCLUDE "m16def.inc"
.DEF temp=R18

.MACRO TestMacro          ; δήλωση μακροεντολής
    ldi temp,@0           ; πρώτη παράμετρο @0 ως τιμή του καταχωρητή temp
    inc  temp             ; αύξηση καταχωρητή temp
    brne macro_jump       ; παράκαμψη επόμενης εντολής αν δεν προκύψει υπερχείλιση
    rjmp OVERFLOW         ; Αλλιώς άλμα σε περίπτωση υπερχείλισης σε ετικέτα εκτός μακροεντολής
macro_jump:               ; Ετικέτα εντός μακροεντολής
.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
    ser temp          ; τα LEDs PB.0 - PB.7 θα σβήσουν
    out PORTB,temp
    rjmp LOOP
```

Διακοπές

Οι διακοπές διακόπτουν την κανονική ροή του προγράμματος λόγω ενός ιδιαίτερου γεγονότος στον επεξεργαστή ή σε κάποια περιφερειακή μονάδα, όπως η υπερχείλιση ενός καταχωρητή, το πάτημα ενός διακόπτη, η λήψη δεδομένων σε μια θύρα, κλπ., που πρέπει να αντιμετωπιστεί μέσω της εκτέλεσης κάποιου τμήματος κώδικα. Αυτός ο κώδικας συνιστά την ρουτίνα εξυπηρέτησης διακοπής (ISR-Interrupt Service Routine):

- Μια περιφερειακή μονάδα (μέσω μιας σημαίας) ειδοποιεί το σύστημα ότι προέκυψε διακοπή.
- Η εντολή που βρίσκεται υπό εκτέλεση στον καταχωρητή εντολών (IR) ολοκληρώνεται.
- Η διεύθυνση της επόμενης εντολής του κυρίου προγράμματος σώζεται στη στοίβα.
- Η διεύθυνση της ρουτίνας εξυπηρέτησης διακοπής φορτώνεται στον μετρητή προγράμματος μέσω μιας εντολής icall, rcall κλπ., ώστε να εκτελεστεί η 1η εντολή της ISR.
- Οι εντολές της ρουτίνας εξυπηρέτησης διακοπής εκτελούνται διαδοχικά μέχρι και την τελευταία εντολή τύπου reti, όπου η διεύθυνση επιστροφής φορτώνεται από τη στοίβα και η σημαία ολικών διακοπών τίθεται.
- Η εκτέλεση του κυρίου προγράμματος συνεχίζεται κανονικά.

Διακοπές του AVR



Διακοπές του AVR

Τα διανύσματα διακοπών ενός μικροελεγκτή ξεκινούν στη διεύθυνση 0x0000, με πρώτο το διάνυσμα επανατοποθέτησης reset.

- επανατοποθέτηση (reset)
- εξωτερική διακοπή IRQ0 στον ακροδέκτη PD2
- εξωτερική διακοπή IRQ1 στον ακροδέκτη PD3
- εξωτερική διακοπή IRQ2 στον ακροδέκτη PB2
- διακοπή σε λειτουργία σύλληψης του χρονιστή timer1
- διακοπή του αναλογικού συγκριτή
- διακοπή σε λειτουργία συγκριτή B του χρονιστή timer1
- διακοπή υπερχείλισης του χρονιστή timer1
- διακοπή υπερχείλισης του χρονιστή timer0
- κατά την ολοκλήρωση της μετάδοσης από τη μονάδα SPI
- διακοπή κατά την ολοκλήρωση της λήψης από τη μονάδα UART
- διακοπή κατά την εκκένωση του καταχ/τή UDR της μονάδας UART
- διακοπή κατά την ολοκλήρωση της εκπομπής από τη μονάδα UART
- διακοπή σε λειτουργία συγκριτή A του χρονιστή timer1

Καταχωρητές διακοπών

Γενικός καταχωρητής **μάσκας** διακοπών (General Interrupt MaSK register-**GIMSK**)

Bit	7	6	5	4	3	2	1	0
Bits ελέγχου	INT1	INT0	INT2					

Γενικός καταχωρητής **σημαίας** διακοπών (General Interrupt Flag register - **GIFR**)

Bit	7	6	5	4	3	2	1	0
Bits ελέγχου	INTF1	INTF0	INTF2					

Γενικός καταχωρητής **ελέγχου** (MiCroprocessor Unit Control Register - **MCUCR**)

Bit	7	6	5	4	3	2	1	0
Λειτουργία	**SM2**	**SE**	**SM1**	**SM0**	ISC11	ISC10	ISC01	ISC00

SE= Sleep Mode Enable (εντολή **`sleep`**)

Καθορισμός των 6 Sleep Mode στους μΕ AVR

SM2	**SM1**	**SM0**	**Sleep Mode**
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Φυλάσσεται για μελλοντική χρήση
1	0	1	Φυλάσσεται για μελλοντική χρήση
1	1	0	Standby
1	1	1	Extended Standby

Μορφή σήματος για πρόκληση διακοπής στους μΕ AVR

ISC01 - ISC00 ή ISC11 – ISC10		**Προκαλείται αίτηση διακοπής όταν το σήμα στην ακίδα INT0 ή INT1 έχει:**
0	0	Χαμηλή Στάθμης
0	1	Υψηλή Στάθμης
1	0	Κατερχόμενη Ακμή
1	1	Ανερχόμενη Ακμή

Διανύσματα διακοπών μΕ AVR ATmega16

Όνομα	Διεύθυνση μνήμης προγράμματος ATmega16	Περιγραφή
Reset	$000	Διαχείριση επανεκκίνησης
EXT_INT0	$002	Διαχείριση εξωτερικής διακοπής IRQ0
EXT_INT1	$004	Διαχείριση εξωτερικής διακοπής IRQ1
TIMER2 COMP	$006	Timer/Counter2 Compare Match
TIMER2 OVF	$008	Timer/Counter2 Overflow
TIMER1_CAPT	$00A	Διαχείριση διακοπής σε λειτουργία σύλληψης του χρονιστή timer1
TIMER1_COMPA	$00C	Διαχείριση διακοπής σε λειτουργία συγκριτή A του χρονιστή timer1
TIMER1_COMPB	$00E	Διαχείριση διακοπής σε λειτουργία συγκριτή B του χρονιστή timer1
TIMER1_OVF	$010	Διαχείριση διακοπής υπερχείλισης του χρονιστή timer1
TIMER0_OVF	$012	Διαχείριση διακοπής υπερχείλισης του χρονιστή timer0
SPI_STC	$014	Διαχείριση διακοπής κατά την ολοκλήρωση της μετάδοσης από τη μονάδα SPI
UART_RXC	$016	Διαχείριση διακοπής κατά την ολοκλήρωση της λήψης από τη μονάδα UART
UART_DRE	$018	Διαχείριση διακοπής κατά την εκκένωση του καταχ/τή UDR της μονάδας UART
UART_TXC	$01A	Διαχείριση διακοπής κατά την ολοκλήρωση της εκπομπής από τη μονάδα UART
ADC ADC	$01C	Conversion Complete
EE_RDY	$01E	EEPROM Ready
ANA_COMP	$020	Διαχείριση διακοπής του αναλογικού συγκριτή
TWI	$022	Two-wire Serial Interface
INT2	$024	External Interrupt Request 2
TIMER0 COMP	$026	Timer/Counter0 Compare Match
SPM_RDY	$028	Store Program Memory Ready

Γενικό πρόγραμμα εγκατάστασης Reset και Διεύθυνσης Διανυσμάτων Διακοπών στον μικροελεγκτή ATmega16

```
$000    jmp RESET       ; Reset Handler
$002    jmp EXT_INT0    ; IRQ0 Handler
$004    jmp EXT_INT1    ; IRQ1 Handler
$006    jmp TIM2_COMP   ; Timer2 Compare Handler
$008    jmp TIM2_OVF    ; Timer2 Overflow Handler
$00A    jmp TIM1_CAPT   ; Timer1 Capture Handler
$00C    jmp TIM1_COMPA  ; Timer1 CompareA Handler
$00E    jmp TIM1_COMPB  ; Timer1 CompareB Handler
$010    jmp TIM1_OVF    ; Timer1 Overflow Handler
$012    jmp TIM0_OVF    ; Timer0 Overflow Handler
$014    jmp SPI_STC     ; SPI Transfer Complete Handler
$016    jmp USART_RXC   ; USART RX Complete Handler
$018    jmp USART_UDRE  ; UDR Empty Handler
$01A    jmp USART_TXC   ; USART TX Complete Handler
$01C    jmp ADC         ; ADC Conversion Complete Handler
$01E    jmp EE_RDY      ; EEPROM Ready Handler
$020    jmp ANA_COMP    ; Analog Comparator Handler
$022    jmp TWSI        ; Two-wire Serial Interface Handler
$024    jmp EXT_INT2    ; IRQ2 Handler
$026    jmp TIM0_COMP   ; Timer0 Compare Handler
$028    jmp SPM_RDY     ; Store Program Memory Ready Handler
```

Παράδειγμα χειρισμού διακοπών

Στο πρόγραμμα αυτό κάθε φορά που προκαλείται εξωτερική διακοπή INT0 (ακροδέκτης PD2) αναστρέφεται ο φωτισμός των led εξόδου (που συνδέονται στην θύρα εξόδου PORTB). Αρχικά θεωρούμε ότι είναι αναμμένο το led2.

```
.include "m16def.inc"
.def temp = r16
.def leds = r17
.org 0
    jmp reset                ; Reset Handler
    jmp interrupt0           ; IRQ0 Handler
    jmp interrupt1           ; IRQ1 Handler
    reti                     ; Υπόλοιποι Handlers

reset:
    ldi temp, high(RAMEND)   ; κύριο πρόγραμμα
    out SPH, temp            ; θέτουμε δείκτη στοίβας στην RAM
    ldi temp, low(RAMEND)
    out SPL, temp
```

Παράδειγμα χειρισμού διακοπών - 2

```
    ldi temp, 0xFF
    out DDRB, temp          ; PORTB  ως έξοδο των leds

    ldi leds, 0b11111011    ; άναμμα led2
    out PORTB, leds
    ldi temp, 1<<INT0       ; 0100 0000 (INT0=6)
    out GIMSK, temp         ; ενεργοποίηση εξωτερικής διακοπής 0
    ldi temp, 0b00000010    ; ορίζουμε η εξωτερική διακοπή INT0 να
    out MCUCR, temp         ; προκαλείται στην ακμή πτώσης (βλ. Πίνακα 1)
    sei                     ; ενεργοποίηση συνολικά των διακοπών
loop:                       ; αναμονή εξωτερικής διακοπής INT0
    rjmp loop               ; (πάτημα διακόπτη 2)

interrupt0:                 ; ρουτίνα εξυπηρέτησης διακοπής INT0
    com leds                ; προέκυψε διακοπή, τίθεται INTF0
    out PORTB, leds         ; επίδειξη στα LEDs
    reti
```