Γλώσσα C για Προγραμματισμό Ενσωματωμένων Συστημάτων

- Η γλώσσα προγραμματισμού C είναι υψηλού επιπέδου. Επιτρέπει όμως αποδοτική περιγραφή και διαχείριση λειτουργιών χαμηλού επιπέδου, π.χ. bitwise-operators, pointer-based memory referencing κ.λπ.

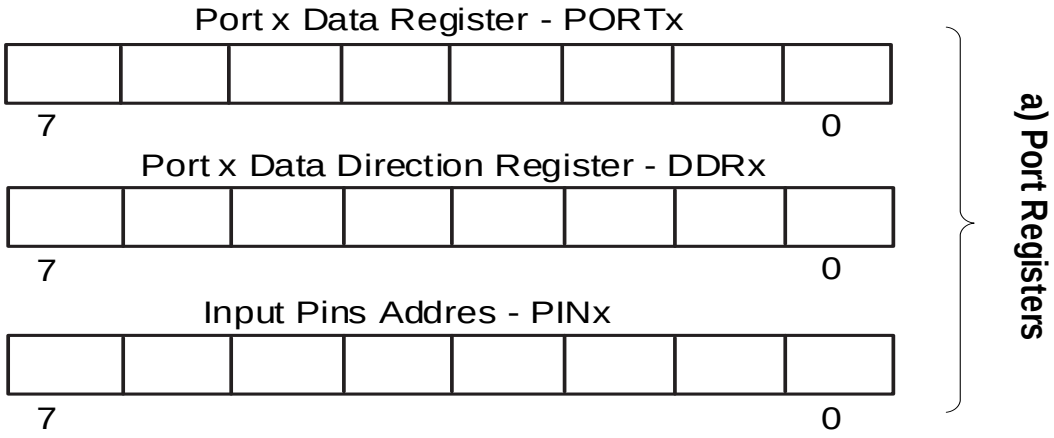
- Τα προγράμματα που αφορούν σε εφαρμογές πραγματικού χρόνου συχνά δεν περιλαμβάνουν κάποιο λειτουργικό σύστημα όπως αυτά των υπολογιστών γενικού σκοπού. Συνήθως είναι συστήματα μικρού ή μεσαίου μεγέθους και η παρουσία ενός κλασικού λειτουργικού συστήματος αποτελεί σοβαρή χρονική επιβάρυνση.

- Σε κρίσιμες εφαρμογές πραγματικού χρόνου μπορούν να χρησιμοποιηθούν ειδικά λειτουργικά συστήματα πραγματικού χρόνου ή να υιοθετηθούν κατά περίπτωση λύσεις.

- Η απλούστερη δομή που υποκαθιστά ένα πλήρες λειτουργικό σύστημα είναι ένας ατέρμονας βρόχος που εκτελεί συνεχώς την εξειδικευμένη εργασία (λειτουργία) και οι χρονικά κρίσιμες λειτουργίες προσαρτώνται σε διακοπές.

- Η παρουσία ενός κλασικού λειτουργικού συστήματος “δυσκολεύει” την αποσφαλμάτωση μιας ενσωματωμένης εφαρμογής.

Καταχωρητές χειρισμού θυρών



DDRxn	PORTxn	I/O	Comments	Pullup
0	0	input	Tri-state (Hi-Z)	No
0	1	input	Pullup	Yes
1	0	output	Output Low	No
1	1	output	Output High	No

x: port (A, B,C,D)
n: pin (0 – 7)

b) port pin configuration

Κώδικας C για AVR I/O θύρες

```
// Αρχείο επικεφαλίδας με δηλώσεις για τις διευθύνσεις των
// I/O  καταχωρητών κλπ.
#include "avr/io.h "
```

Output:

```
DDRB=0xf0;        // PORTB[7:4] as output, PORTB[3:0] as input
PORTB=0x00;       // disable pull-up resistors

DDRC=0xff;        // set PORTC as output
PORTC=0x00;       // initialize low
```

Input:

```
unsigined char new_portb;
new_portb = PINB;          // read PORTB
```

Κώδικας C για ανάγνωση/εγγραφή θυρών

```
#include "avr/io.h «

void main(void)
{

DDRB=0xFF;                  // Αρχικοποίηση της θύρας B σαν έξοδο
DDRD=0x00;                  // Αρχικοποίηση της θύρας D σαν είσοδο

while (1)
    {                               // Διάβασμα θύρας PIND
        PORTB=PIND + 5;  // πρόσθεση  του 5 και
    };                              // εγγραφή στην PORTB
}
```

ΠΑΡΑΔΕΙΓΜΑ 1ο: Κώδικας C για χειρισμό bit

Κώδικας C για τον ATmega16 που διαβάζει τα 3 LSB της θύρας εισόδου PORTD και τα μεταφέρει στα 3 MSB της θύρας εξόδου PORTB.
Υποθέτουμε θετική λογική στην είσοδο και έξοδο.

```
#include "avr/io.h "

int main(void)
{
  DDRD = 0x00;                  // PORTD είσοδος
  DDRB = 0b11100000;            // PORTB, MSB 3 bits  έξοδοι

  while (1)    {
     unsigned char input = PIND & 0x07;  // Διάβασμα των 3 LSB της PORTD
     unsigned char output = input << 5;     // Μεταφορά των bit στην σωστή θέση

     // Τα 5 LSB διατηρούνται ενώ ενημερώνονται τα 3 MSB της PORTB
     PORTB = (PORTB & 0x1F) | output;  }
}
```

ΠΑΡΑΔΕΙΓΜΑ 2ο: Κώδικας C για χειρισμό Led

//Ανάμα – σβήσιμο ενός LED στο bit POPTB,0 με ρυθμό 1Hz

```
#include "avr/io.h "
#include <util/delay.h>

void main()
{
  DDRB = 0XFF;          //PORTB as output
  PORTB = 0x00;         //keep all LEDs off
  while(1)
  {
    PORTB &= 0b11111110;      //turn LED off
    _delay_ms(500);           //wait for 500mS
    PORTB |= 0b00000001;      //turn LED on
    _delay_ms(500);           //wait for 500mS
  };
}
```

ΠΑΡΑΔΕΙΓΜΑ 3ο: Διαφορά πλήθους μονάδων από το πλήθος των μηδενικών της θύρας PORTD

```
#include "avr/io.h "
unsigned char x_p, x_m, y, z;

void main(void){
unsigned char i;
DDRB=0xFF;          // Αρχικοποίηση της θύρας B ως έξοδο
DDRD=0x00;          // Αρχικοποίηση της θύρας D ως είσοδο

while (1)           // Ατέρμων βρόγχος.
    {
        z = PIND;           // Διαβάζονται τα bits της θύρας PIND (switches)
        x = 0;              // Αρχικοποίηση του μετρητή των '1' – '0'.
        for(i = 0; i < 8; i++)   // Βρόγχος με 8 επαναλήψεις
        {
            y = z & 0x1;        // Απομονώνεται το LSB του z.
            z >> 1;             // ολίσθηση 1 θέση δεξιά
            if (y == 1)         // Έλεγχος LSbit για 1 ή 0
                x_p ++;         // Μετρητής ψηφίων 1
            else
                x_m ++;         // Μετρητής ψηφίων 0
        }
        // Χρήση του XOR(^) για συμπλήρωμα ως προς 1(απεικόνιση με ανάστροφη λογική)
        if (x_p > x_m)
            PORTB = (x_p – x_m) ^ 0xFF;
        else
             PORTB = (x_m – x_p) ^ 0xFF;
    }
}
```

ΠΑΡΑΔΕΙΓΜΑ 4ο: Με βάση τον αριθμό (0-7) από τα 3 LSB του PORTD να ανάβει το αντίστοιχης τάξης bit του PORTB

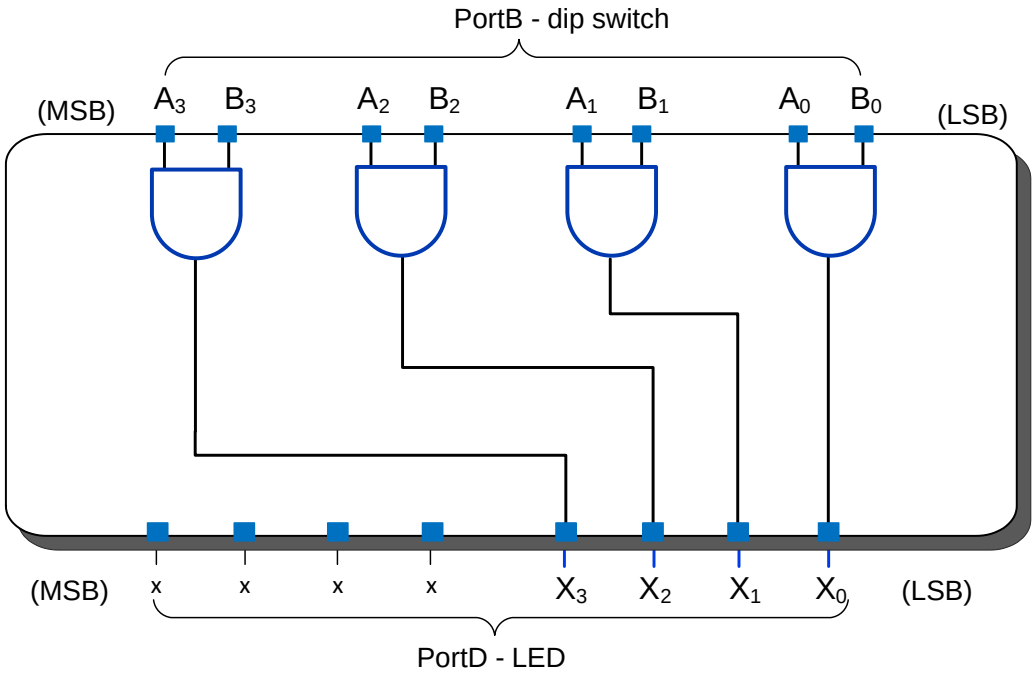
```
#include "avr/io.h "
unsigned char x, y, z;

void main(void) {
int i;
DDRB=0xFF;          // Αρχικοποίηση της θύρας B ως έξοδο
DDRD=0x00;          // Αρχικοποίηση της θύρας D ως είσοδο

while (1)           // Ατέρμων βρόγχος
    {
        z = PIND & 0x7;      // Διαβάζονται τα bits της θύρας PIND (switches)
        x = 1;               // Αρχικοποίηση του καταχωρητή x.

        for(i = 0; i < z; i++)    // Βρόγχος με z επαναλήψεις
        {
            x << 1;      // ολίσθηση του καταχωρητή x, 1 θέση αριστερά
        }

     // Χρήση του XOR(^) για συμπλήρωμα ως προς 1(απεικόνιση με ανάστροφη λογική)
    PORTB = x ^ 0xFF;
    }
}
```

ΠΑΡΑΔΕΙΓΜΑ 5ο: Εξομοίωση πυλών

Να εξομοιωθεί σε C η λειτουργία ενός υποθετικού I.C. που περιλαμβάνει 4 πύλες AND όπως φαίνεται στο σχήμα.

Υπάρχουν dip switches στην πόρτα εισόδου PORTB, και LED στην πόρτα εξόδου PORTD ενός Μικροελεγκτή AVR.

Υποθέτουμε θετική λογική.



ΠΑΡΑΔΕΙΓΜΑ 5ο: - συνέχεια

```
#include "avr/io.h "
unsigned char x, y, z;
void main(void)
{
unsigned char i;
DDRB=0x00;                  // Αρχικοποίηση της θύρας B ως είσοδο
DDRD=0xFF;                  // Αρχικοποίηση της θύρας D ως έξοδο
while (1)                   // Ατέρμων βρόγχος
    {
    x = PINB;               // Διαβάζονται τα 8 bits της θύρας PINB (switches)
    z = 0;                  // Αρχικοποίηση του καταχωρητή z
        for(i = 0; i < 4; i++)  // Βρόγχος με 4 επαναλήψεις
        {
        y=x & 0x3;          // Απομονώνω τα 2 LSB
            if(y == 3)
            {
            z = z | 0x10;   // Θέτει '1' αν η κάθε πύλη δίνει αυτή τη τιμή
            }
            z >> 1;         // ολίσθηση του καταχωρητή z, 1 θέση δεξιά
            x >> 2;         // ολίσθηση του καταχωρητή x, 2 θέσεις δεξιά
        }
    PORTD = z & 0xF;        // Το αποτέλεσμα στα 4 LSbits της θύρας PORTD (leds)
    }                       // Υποθέτουμε απεικόνιση με θετική λογική.
}
```

Μεταβλητές Volatile

Η δεσμευμένη λέξη volatile είναι ένας ANSI-C προσδιοριστής μεταβλητών για κώδικες οδήγησης περιφερειακών και διαχείρισης διακοπών. Κάθε δεδομένο που η τιμή του δύναται να ενημερώνεται με ασύγχρονο τρόπο και όχι μέσα από κώδικα του μικροεπεξεργαστή πρέπει να δηλώνεται ως volatile.

```
#define PER_PORT 0x1775    // διεύθυνση θύρα εισόδου
                           // ενός περιφερειακού με 16-bit είσοδο
…
volatile int *port = PER_PORT;  // Δείκτης χειρισμού της θύρας
int data_a, data_b;
```

Η PORT πρέπει να διαβαστεί δυο φορές για την περίπτωση όπου η τιμή της έχει αλλάξει στο ενδιάμεσο.

```
data_a = *port ;    // 1η Ανάγνωση PER_PORT
data_b = *port ;    // 2η Ανάγνωση PER_PORT
```

Χωρίς τη δήλωση του δείκτη port ως volatile, ο μεταγλωττιστής θα συμπέραινε την εξής συμπεριφορά: data_a = data_ b =*port (εξοικονομώντας έτσι μια θέση μνήμης), κάτι το οποίο δεν ισχύει στην προκειμένη περίπτωση καθώς στη δεύτερη ανάγνωση της θύρας το δεδομένο μπορεί να έχει αλλάξει.

Στην περίπτωση που **data_a** και **data_b** δεν δηλωθούν volatile τότε η PORT θα ‘διαβαστεί’ μόνο μια φορά θεωρώντας οτι: **data_a = data_b**

ΟΔΗΓΙΕΣ(DIRECTIVES) του AVR ASSEMBLER

DIRECTIVES για μετάφραση και υλοποίηση τύπων δεδομένων σε AVR ASSEMBLY

ΠΕΡΙΣΣΟΤΕΡΕΣ ΠΛΗΡΟΦΟΡΙΕΣ: [https://onlinedocs.microchip.com/oxy/GUID-E06F3258-483F-4A7B-B1F8-69933E029363-en-US-2/GUID-844173A6-75A7-4A88-855C-40F4B3151FFF.html]

.ORG (origin) Διεύθυνση μνήμης για τον κώδικα που απολουθεί.
.EQU (equate) Ορίζει μια σταθερά τιμή με ένα όνομα (σύμβολο).
.DEF (define) Ορίζει ένα ψευδώνυμο για έναν καταχωρητή.
.INCLUDE Εισάγει άλλο αρχείο, π.χ. ορισμούς για registers.
.BYTE Δεσμεύει θέσεις στη RAM για μεταβλητές.
.DW (define word) Ορίζει μια λέξη (2 bytes) στη μνήμη προγράμματος.
.DB (define byte) Ορίζει ένα ή περισσότερα bytes δεδομένων.
.CSEG(code segment): Ορίζει την αρχή σε τμήμα κώδικα.
.DSEG(data segment): Ορίζει την αρχή σε τμήμα κώδικα δεδομένων.
.LIST Ενεργοποιεί τη δημιουργία ή τη συνέχιση του αρχείου listing.
.END Δηλώνει το τέλος του αρχείου Assembly.

Θέση αποθήκευσης των μεταβλητών

- Η προεπιλεγμένη θέση αποθήκευσης των μεταβλητών είναι στη μνήμη SRAM.
- Οι σταθερές μπορούν να τοποθετηθούν στη μνήμη FLASH (χώρος προγράμματος) με το χαρακτηρισμό flash ή const.
- Για να τοποθετηθούν μεταβλητές στην EEPROM, χρησιμοποιείται ο χαρακτηρισμός eeprom.

Επιλογή SRAM (C language)

```
char mystring[30] = *This string is placed in SRAM*;
```

Επιλογή FLASH (C language)

```
flash char string_constant1[] = *This is a string constant*;
const char string_constant2[] = *This is also a string constant*;
```

Επιλογή EEPROM (C language)

```
eeprom int cycle_count; // allocates an integer space in EEPROM
eeprom char ee_string[20]; // allocates a 20-byte area in EEPROM
eeprom struct {
        char a;
        int b;
        char c[15];
} se;           // allocates 18-byte structure *se* in EEPROM
```

Διακοπές του AVR

Όταν ενεργοποιηθεί μια διακοπή και είναι επιτρεπόμενη (enabled), ο επεξεργαστής διακόπτει την κανονική ροή του προγράμματος.

Πηγαίνει στη διεύθυνση του αντίστοιχου interrupt vector, εκτελεί τον κώδικα της ISR και επιστρέφει με την εντολή RETI.

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

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

Καταχωρητές διακοπών INT0, INT1

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

Παράδειγμα 6: Χειρισμός διακοπών(Assembly)

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 6 -συνέχεια

```
    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
```

Παράδειγμα 7: Χειρισμός διακοπών(C)

Στη συνέχεια δίνεται ένα παράδειγμα προγράμματος διαχείρισης διακοπών σε C που επιτρέπει εξωτερικές διακοπές στην είσοδο INT0 του Μικροελεγκτή AVR.

Αρχικά τα led της θύρας PORTB είναι ON και στη συνέχεια κάθε φορά που προκαλείται διακοπή αλλάζει η κατάστασή τους.

```
#include "avr/io.h “
#include<avr/interrupt.h>

unsigned char chLed;

// External Interrupt 0 service routine.
ISR(INT0_vect)
{
  chLed = chLed ^ 0xFF;
  PORTB=chLed;
}
```

Παράδειγμα 7: (συνέχεια)

```
void main(void)
{
  PORTB=0x00;
  DDRB=0xFF;          // Η PORTB τίθεται ως έξοδος

  GIMSK=0x40;         // Ενεργοποίηση εξωτερικής διακοπής 0
  MCUCR=0x02;         // INT0 Mode: στην ακμή πτώσης
  sei();              // ενεργοποίηση συνολικά των διακοπών

  chLed=0x00;
  PORTB=chLed;

     while (1)
     {
       // Μπορούμε να τοποθετήσουμε πρόσθετο κώδικα στο σημείο αυτό
     }
}
```

Λίστα με τα ISR ονόματα στη C

```
ISR(INT0_vect)              // Εξωτερική διακοπή από το pin INT0
ISR(INT1_vect)              // Εξωτερική διακοπή από το pin INT1
ISR(INT2_vect)              // Εξωτερική διακοπή από το pin INT2

ISR(TIMER2_COMP_vect)    // Σύγκριση τιμής Timer/Counter2 με OCR2
ISR(TIMER2_OVF_vect)     // Υπερχείλιση του Timer/Counter2
ISR(TIMER1_CAPT_vect)    // Καταγραφή τιμής Timer1 σε Capture
ISR(TIMER1_COMPA_vect)  // Σύγκριση Timer1 με OCR1A
ISR(TIMER1_COMPB_vect)  // Σύγκριση Timer1 με OCR1B
ISR(TIMER1_OVF_vect)     // Υπερχείλιση του Timer/Counter1
ISR(TIMER0_COMP_vect)    // Σύγκριση Timer0 με OCR0
ISR(TIMER0_OVF_vect)     // Υπερχείλιση του Timer/Counter0

ISR(SPI_STC_vect)           // Ολοκλήρωση μεταφοράς SPI
```

Λίστα με τα ISR ονόματα στη C(συνέχεια)

```
ISR(USART_RXC_vect)     // Παραλαβή δεδομένου μέσω USART
ISR(USART_UDRE_vect)    // Έτοιμο το USART για αποστολή νέου byte
ISR(USART_TXC_vect)     // Ολοκλήρωση αποστολής byte μέσω USART

ISR(ADC_vect)           // Ολοκλήρωση μετατροπής ADC

ISR(EE_RDY_vect)        // EEPROM έτοιμη για ανάγνωση/εγγραφή

ISR(ANA_COMP_vect)      // Αναλογικός συγκριτής (Analog Comparator)

ISR(TWI_vect)           // Δραστηριότητα στο TWI (I2C)

ISR(SPM_RDY_vect)       // Αποθήκευση στο Flash (Self Programming)
```

Διευθύνσεις I/O καταχωρητών ATmega16

Address	Name	Address	Name
$3F ($5F)	SREG	$1F ($3F)	EEARH-EEPROM Addr register high byte
$3E ($5E)	SPH	$1E ($3E)	EEARL-EEPROM Addr register low byte
$3D ($5D)	SPL	$1D ($3D)	EEDR - EEPROM Data Register
$3C ($5C)	OCR0 Timer/Counter0 -Output Compare Register	$1C ($3C)	EECR - EEPROM Control Register
$3B ($5B)	GICR Gener. Interrupt Control	$1B ($3B)	PORTA
$3A ($5A)	GIFR General Interrupt Flags	$1A ($3A)	DDRA
$39 ($59)	TIMSK Timers Interrupt Mask	$19 ($39)	PINA
$38 ($58)	TIFR Timers Interrupt Flags	$18 ($38)	PORTB
$37 ($57)	SPMCR Store Prog Con. Reg.	$17 ($37)	DDRB
$36 ($56)	TWCR - TWI Control Register	$16 ($36)	PINB
$35 ($55)	MCUCR Processor General Control Register	$15 ($35)	PORTC
$34 ($54)	MCUCSR (Pr Status Register)	$14 ($34)	DDRC
$33 ($53)	TCCR0 Timer0 Control Reg.	$13 ($33)	PINC
$32 ($52)	TCNT0Timer/Counter0	$12 ($32)	PORTD
$31 ($51)	OSCCAL Osc. Calibration Reg	$11 ($31)	DDRD
$30 ($50)	SFIOR	$10 ($30)	PIND
$2F ($4F)	TCCR1A Timer1 Control Register A	$0F ($2F)	SPDR - SPI Data Register
$2E ($4E)	TCCR1B Timer1 Control Register B	$0E ($2E)	SPSR - SPI Status Register
$2D ($4D)	TCNT1H Timer/Counter1-Counter Register High Byte	$0D ($2D)	SPCR - SPI Control Register
$2C ($4C)	TCNT1L Timer/Counter1-Counter Register Low Byte	$0C ($2C)	UDR USART I/O Data Register
$2B ($4B)	OCR1AH Timer/Counter1-Output Compare Register A	$0B ($2B)	UCSRA-USARTControl Status Register A
$2A ($4A)	OCR1AL Timer/Counter1-Output Compare Register A	$0A ($2A)	UCSRB-USARTControl Status Register B
$29 ($49)	OCR1BH Timer/Counter1-Output Compare Register B	$09 ($29)	UBRRL - USART Baud Rate Register Low Byte
$28 ($48)	OCR1BL Timer/Counter1-Output Compare Register B	$08 ($28)	ACSR - Analog Control Status Register
$27 ($47)	ICR1H Timer/Counter1 Input Capture Register High Byte1	$07 ($27)	ADMUX - ADC Multiplexer Selection Register
$26 ($46)	ICR1L Timer/Counter1 Input Capture Register Low Byte1	$06 ($26)	ADCSRA - ADC Control Status Register
$25 ($45)	TCCR2 Timer2 Control Reg.	$05 ($25)	ADCH - ADC Data Register High Byte
$24 ($44)	TCNT2 Timer/Counter2 (8 bit)	$04 ($24)	ADCL - ADC Data Register Low Byte
$23 ($43)	OCR2Timer/Counter2 Output Compare Register	$03 ($23)	TWDR Two-wire Serial - Interface Data Register
$22 ($42)	ASSR Asynchr. Status Reg.	$02 ($22)	TWAR Two-wire Address Register
$21 ($41)	WDTCR- WatchDog Con. Reg	$01 ($21)	TWSR Two-wire Status Register
$20 ($40)	UBRRH/URSEL (USART Baud Rate Register High Byte)	$00 ($20)	TWBR Two-wire bit Rate Register

Αναφορές

- Κ. Πεκμεστζή, “Συστήματα Μικροϋπολογιστών – Μικροελεγκτής AVR και PIC”
- William Barnekow, “Mixing C and assembly language programs”, Lecture Notes, Cornel University
- “Embedded C Programming and the Atmel AVR”, R. Barnett, Thomson and Delmar Learning.
- Sri Parameswaran, Annie Guo, Hui Wu, “Assembly Programming (iii)”, Lecture Notes on Microprocessors and Interfacing, University of New South Wales
- http://www.atmel.com/webdoc/avrassembler/