Μικροελεγκτές AVR

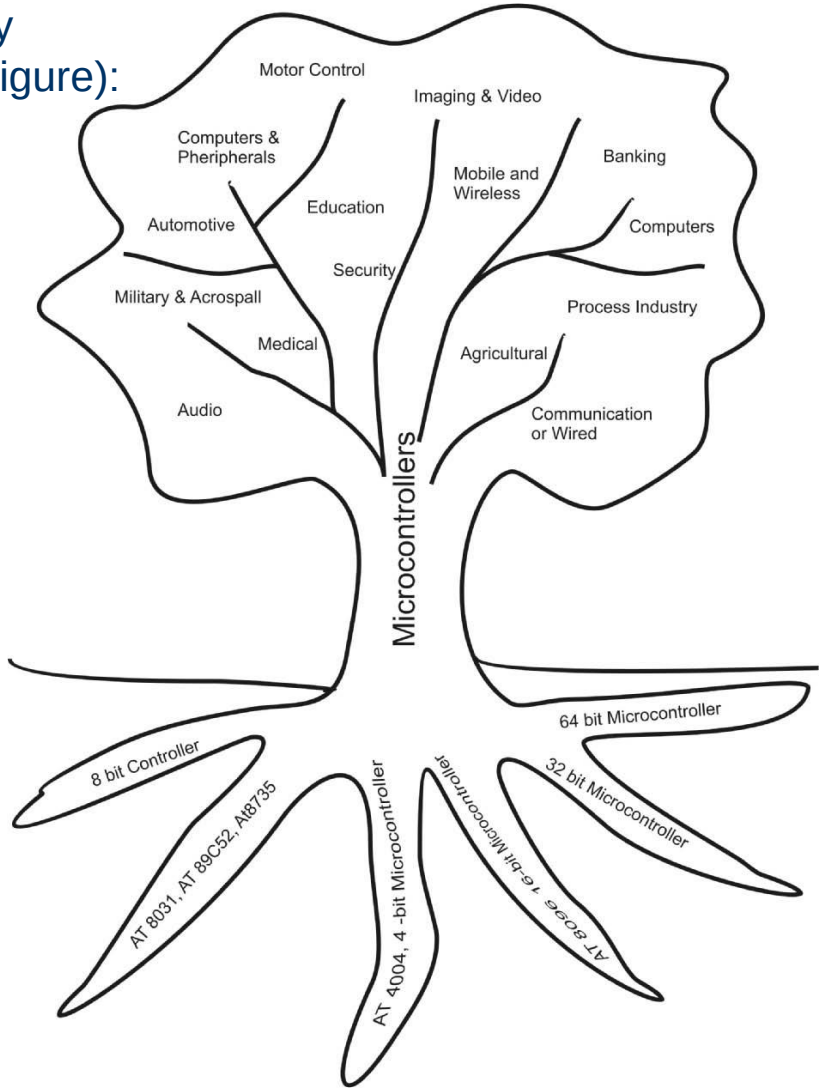
2η ΕΝΟΤΗΤΑ:
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ ΓΛΩΣΣΑ C ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

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

Microcontroller application tree

The microcontroller applications are mainly categorized into the following types (see Figure):

- Audio
- Automotive
- Communication/wired
- Computers and peripherals
- Consumer
- Industrial
- Imaging and video
- Medical
- Military/aerospace
- Mobile/wireless
- Motor control
- Security
- General Purpose
- Miscellaneous



Γλώσσα C για Προγραμματισμό Ενσωματωμένων Συστημάτων

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

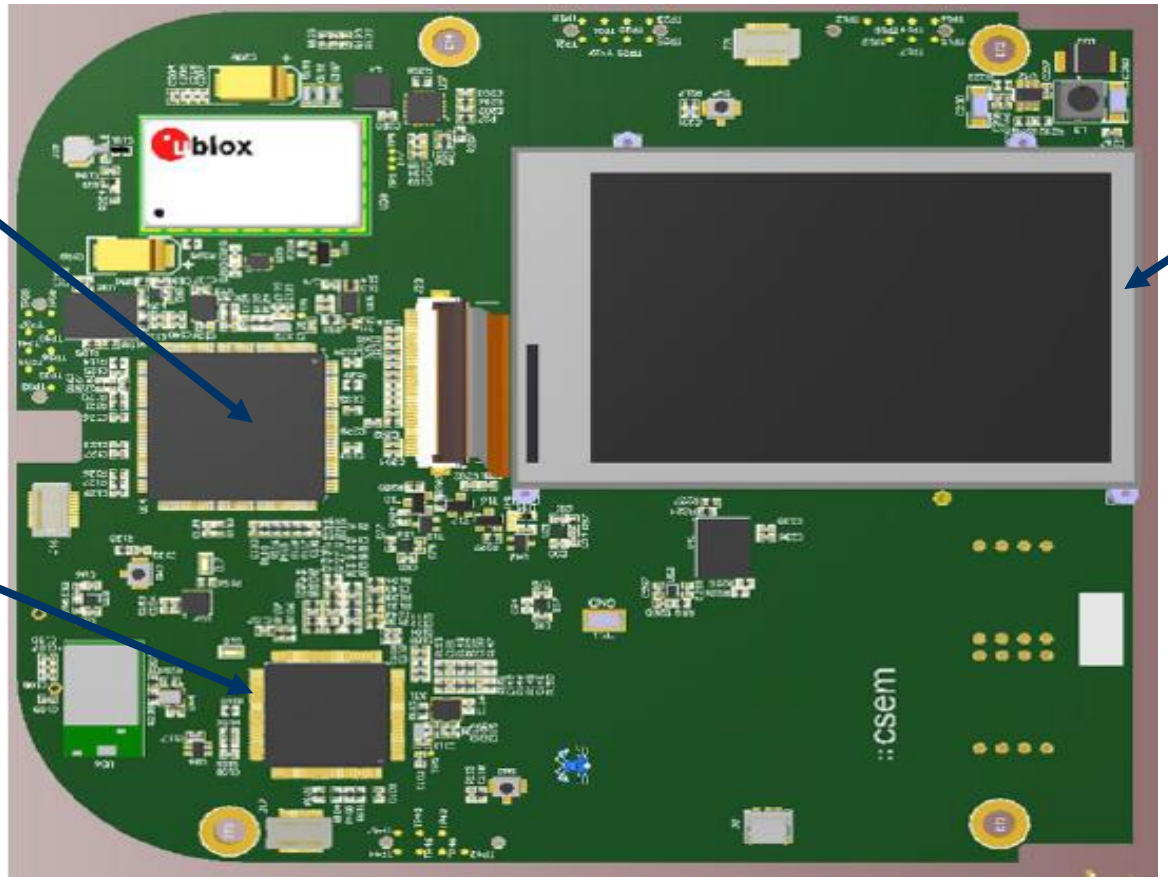
Τα προγράμματα που αφορούν σε εφαρμογές πραγματικού χρόνου συχνά δεν περιλαμβάνουν κάποιο λειτουργικό σύστημα όπως αυτά των υπολογιστών γενικού σκοπού για τους παρακάτω λόγους:

- Συνήθως είναι συστήματα μικρού ή μεσαίου μεγέθους, δεδομένου ότι (όπως έγινε σαφές και από τα προηγούμενα) τα ενσωματωμένα συστήματα επιτελούν μια συγκεκριμένη εργασία.
- Η παρουσία ενός κλασικού λειτουργικού συστήματος αποτελεί σοβαρή χρονική επιβάρυνση. Έτσι για να αποκρίνεται 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 της ενσωματωμένης εφαρμογής.

Proof of concepts: Smart Negative Pressure Medical Device Programming in Microlab

Main Microcontroller (ARM-M4)

Supervisory microcontroller (ARM-M0+)



User Interface microcontroller (under ePaper Display)

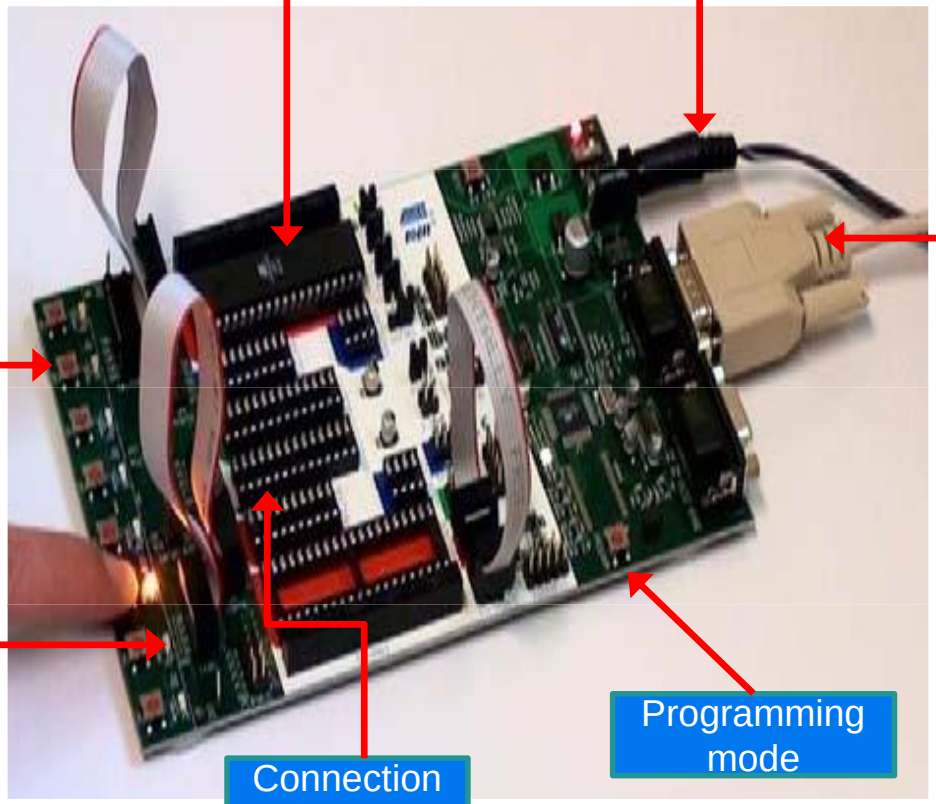
SNPD (Smart Negative Pressure Device)

- Software stack from drivers up to user application
- No RTOS (Real Time Operating System)
- C and assembly programming

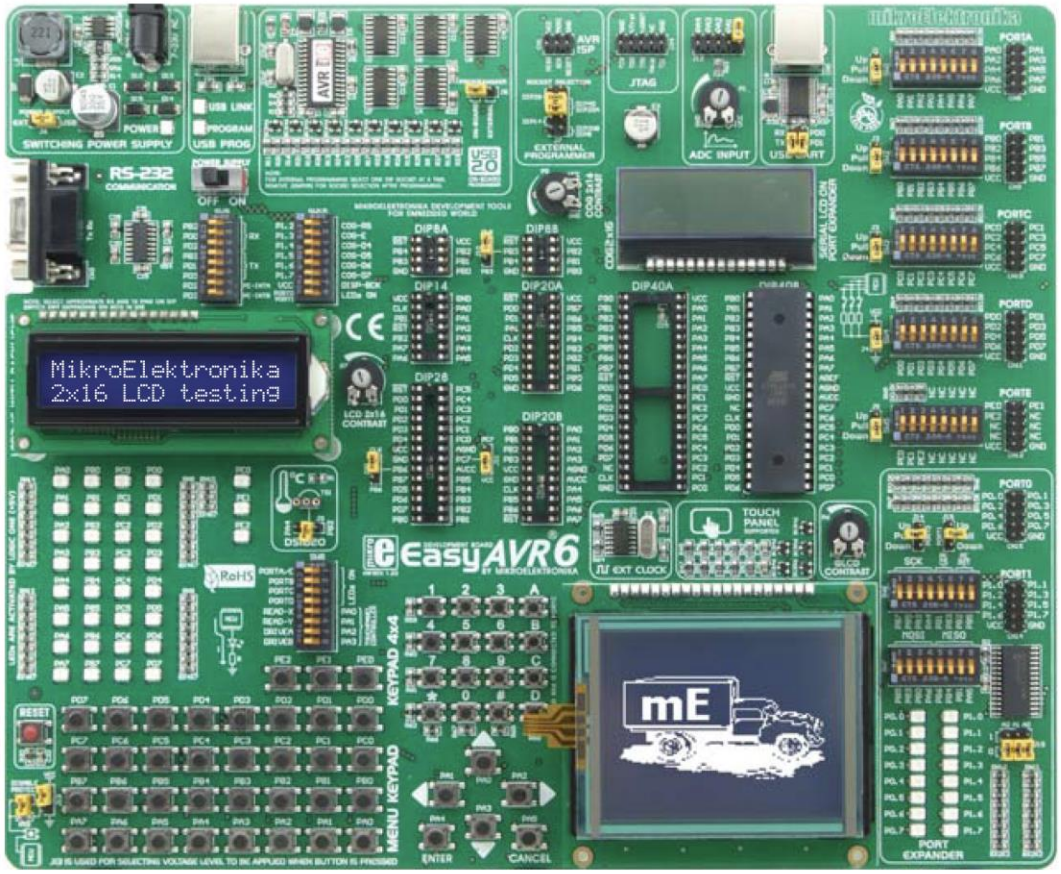
SNPD real case: Pump control

Verification in complex lab tubing system with pressure meter and on healthy volunteer

AVR development kit



EasyAVR6 Development System (περισσότερα το επόμενο εξάμηνο…)



AVR DEVELOPMENT BOARD — Full-featured and user-friendly development board for AVR microcontrollers

USB 2.0 IN-CIRCUIT PROGRAMMER — High-Performance USB 2.0 On-Board Programmer

PORT EXPANDER — Port Expander provides easy I/O expansion (2 additional ports) using data format conversion

COG 2x16 SERIAL LCD — Alphanumeric On-Board 2x16 LCD with Serial Communication

GRAPHIC LCD CONNECTOR ON-BOARD — Graphic LCD with backlight

Χρήσιμα ‘tips’ για Embedded AVR-C

Θέματα που θα μελετηθούν:

- AVR I/O
- Παραδείγματα Προγραμματισμού σε AVR-C
- Volatile variables
- AVR Memory space
- Stack and function call

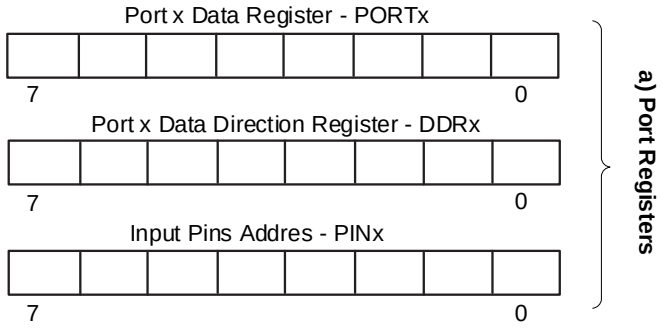
I/O από AVR θύρες

Output:

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

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

Input:

```
Unsigined char new_PORTB;   // PORTB new value
:
:
:
new_PORTB = PINB;          // read PORTB
```



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

Ανάγνωση σε και εγγραφή από θύρες

```
#include <mega16.h>  // Αρχείο επικεφαλίδας για τον AVR ATmega16
// το περιεχόμενο του αρχείο επικεφαλίδας περιλαμβάνει δηλώσεις του τύπου
// sfrb PORTB = 0x16, sfrb DDRB = 0x17, sfrb DDRD = 0x17,(special function register byte)
// δηλ. όλες τις διευθύνσεις των I/O καταχωρητών των περιφερειακών
// και τα διανύσματα διακοπών όλων των περιφερειακών

void main(void)
{

DDRB=0xFF;                  // Αρχικοποίηση της θύρας B σαν έξοδο
DDRD=0x00;                  // Αρχικοποίηση της θύρας D σαν είσοδο
while (1)
     {                      // Τα bits της θύρας PIND (switches) διαβάζονται και
     PORTB=PIND + 5;        // αφού προστεθούν με το 5 γράφονται στην PORTB (leds)
                            //  Υποθέτουμε απεικόνιση με θετική λογική.
     };
}
```

Όλες τις διευθύνσεις των I/O καταχωρητών των περιφερειακών

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

ΠΑΡΑΔΕΙΓΜΑ 1ο: I/O και Χειρισμός bit
Τα 3 LSB της θύρας PORTD στα 3 MSB της θύρας PORTB

```
#include <mega16.h>  // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char z;
void main(void)
{
DDRB = 0xFF;                    // Αρχικοποίηση της θύρας B ως έξοδο
DDRD = 0x00;                    // Αρχικοποίηση της θύρας D ως είσοδο
while (1)
    {                           // Τα bits της θύρας PIND (switches) διαβάζονται
        z = PIND & 0x7;         // και απομονώνονται τα 3 τελευταία bits (masking)
        z << 5;                 // ολίσθηση 5 θέσεις αριστερά και
                                // τα 3 LSB πάνε στα 3 MSB
PORTB = PORTB & 0x1F;           // Τα bits της θύρας PORTB (leds) διαβάζονται,
                                // μηδενίζονται τα 3 MSB και στη συνέχεια μπαίνουν τα
PORTB = PORTB | z;              //  αντίστοιχα του PORTD αφού γίνουν  OR με το z,
    };                          // γράφονται ξανά πίσω στην PORTB (leds).
}                               // Υποθέτουμε απεικόνιση με θετική λογική.
```

ΠΑΡΑΔΕΙΓΜΑ 2ο: Διαφορά πλήθους μονάδων από το πλήθος των μηδενικών της θύρας PORTD

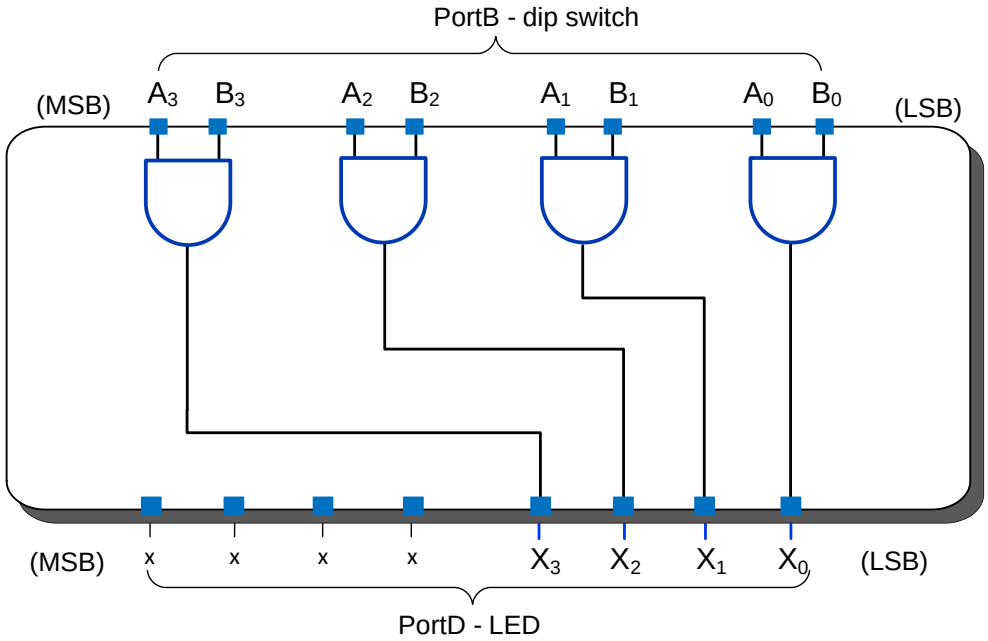
```
#include <mega16.h>    // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char x, y, z;
void main(void)
{
int i;
DDRB=0xFF;          // Αρχικοποίηση της θύρας B ως έξοδο
DDRD=0x00;          // Αρχικοποίηση της θύρας D ως είσοδο
while (1)           // Ατέρμων βρόγχος. Η συνθήκη είναι πάντα αληθής.
    {
        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
        {                   // Απομονώνεται το LSB του z.
            y = z & 0x1;    // Η εντολή αυτή μπορεί να παραληφθεί και να πάει στην if.
            z >> 1;         // ολίσθηση του καταχωρητή z, 1 θέση δεξιά
            if (y == 1)     // Δομή ελέγχου για διακλάδωση της ροής
                {           // προγράμματος με βάση την τιμή του y
                    x = x + 1;
                }
            else
                {
                    x = x - 1;
                }
        }
    PORTB = x ^ 0xFF;   // Τα bits της θύρας PORTB (leds)
                        // γράφονται με το συμπλήρωμα ως προς 1 του z
    };                  // Υποθέτουμε απεικόνιση με αρνητική λογική.
}
```

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

```
#include <mega16.h>    // Αρχείο επικεφαλίδας για τον AVR ATmega16
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. Η συνθήκη είναι πάντα αληθής.
    {
        z = PIND & 0x7;      // Διαβάζονται τα bits της θύρας PIND (switches)
        x = 1;               // Αρχικοποίηση του καταχωρητή x.
        if(z > 0)
        {
            for(i = 0; i < z; i++)    // Βρόγχος με z επαναλήψεις
            {
            x << 1;     // ολίσθηση του καταχωρητή x, 1 θέση αριστερά
            }
        }
    PORTB = x ^ 0xFF;       // Τα bits της θύρας PORTB (leds)
                            // γράφονται με το συμπλήρωμα ως προς 1 του z
    };                      // Υποθέτουμε απεικόνιση με αρνητική λογική.
}
```

ΠΑΡΑΔΕΙΓΜΑ 4ο: Εξομοίωση πυλών

Να εξομοιωθεί σε C η λειτουργία ενός υποθετικού I.C. που περιλαμβάνει 4 πύλες AND όπως φαίνεται στο σχήμα. Τα bits εισόδου πρέπει να αντιστοιχούν ακριβώς όπως φαίνονται στο σχήμα με τα dip switches της πόρτας εισόδου PortB, και οι έξοδοι με τα LEDs που πρέπει να είναι τα τέσσερα LSB της πόρτας εξόδου PortD ενός Μικροελεγκτή AVR. Υποθέτουμε θετική λογική.



ΠΑΡΑΔΕΙΓΜΑ 4ο: Εξομοίωση πυλών

```
#include <mega16.h>    // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char x, y, z;
void main(void)
{
int i;
DDRB=0x00;          // Αρχικοποίηση της θύρας B ως είσοδο
DDRD=0xFF;          // Αρχικοποίηση της θύρας D ως έξοδο
while (1)           // Ατέρμων βρόγχος. Η συνθήκη είναι πάντα αληθής.
    {
    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.
}
```

Volatile variables

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.

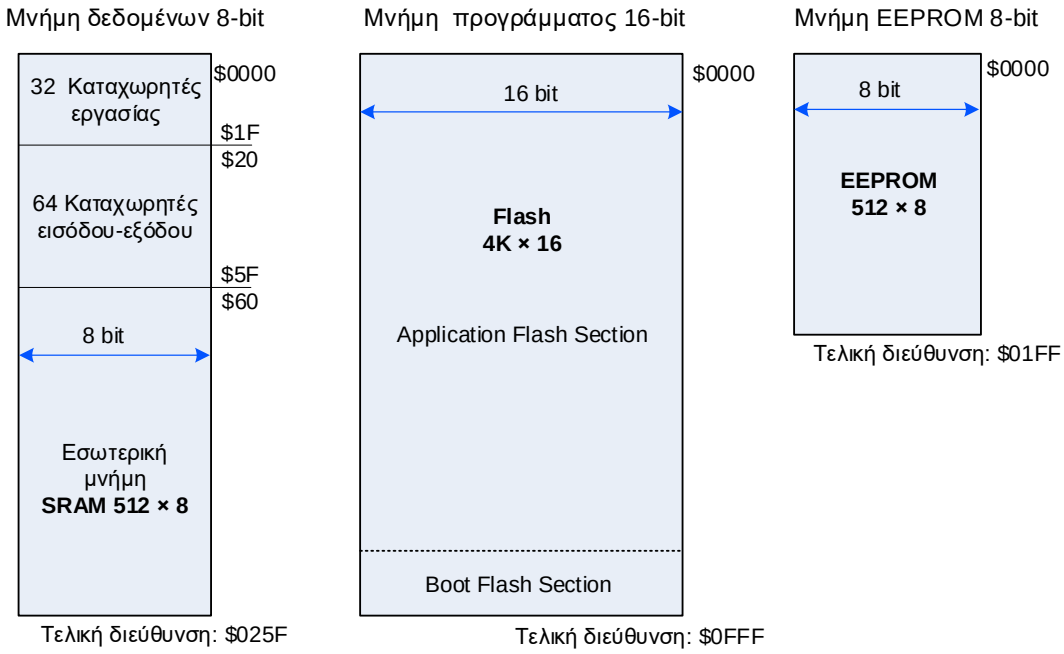
```
#define PER_PORT 0x1775    // διεύθυνση θύρα εισόδου
                           // ενός περιφερειακού με 16-bit είσοδο
…
volatile int *port = PER_PORT;  // Δείκτης χειρισμού της θύρας
int data_a, data_b;

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

Η PORT θα 'διαβαστεί' δυο φορές σε περίπτωση όπου η τιμή της έχει αλλάξει στο ενδιάμεσο.

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**

AVR memory space



AVR Memory Segments

- Data Memory Segment (.DSEG)
- Program Memory Segment (.CSEG)
- EEPROM Memory (.ESEG)

Data/Variable Allocation Directives:

- **Variables in .DSEG**
- **Constants in .CSEG or .ESEG** (προκειμένου η τιμή να παραμένει και μετά το power off)

ΠΙΝΑΚΑΣ ΜΕ ΤΑ AVR ASSEMBLER DIRECTIVES

DIRECTIVES για μετάφραση και υλοποίηση τύπων δεδομένων σε AVR ASSEMBLY

ΠΕΡΙΣΣΟΤΕΡΕΣ ΠΛΗΡΟΦΟΡΙΕΣ:
[http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_directives.html]

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define ...nich device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

C to AVR Assembly: allocation and placements of constants and variables

C Source Code

```
int a;
const chat b[ ]="COMP9032";
const int c=9032;
```

AVR Assembly

```
.dseg
.org 0x100
a: .byte 4

.cseg
b: .DB 'C', 'O', 'M', 'P', '9', '0', '3', '2', 0
c: .DW 9032
```

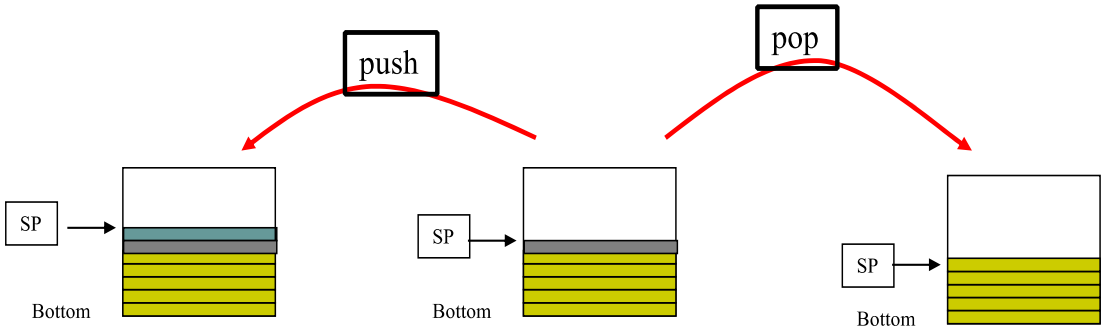
- **Μεταβλητές → .dseg**
- **σταθερές → .cseg (ΓΙΑΤΙ ?)**

GCC Assembler vs Atmel AVR Assembler

GCC ASSEMBLER	ATMEL AVR ASSEMBLER
#INCLUDE	.INCLUDE
ENABLE DATA INITIALIZATION IN .DATA SEGMENT	DATA INITIALIZATION NOT ALLOWED IN .DSEG
HI8()	HIGH()
LO8()	LOW()
.ASCIZ	.DB
.SECTION .DATA	.DSEG
.SECTION .TEXT	.CSEG
<AVR/IO.H>	“M16DEF.INC”

Avr Stack

- Στους επεξεργαστές AVR, η στοίβα (stack) υλοποιείται από ένα σύνολο συνεχομένων BYTES στην μνήμη SRAM.
- Stack pointer (SP): I/O register SPH:SPL
- Η στοίβα χρησιμοποιείται στην υλοποίηση κλήσεων συναρτήσεων.



POP instruction

- Syntax: *pop Rd*
- Operands: Rd∈{r0, r1, …, r31}
- Operation: SP ← SP +1
  Rd ← (SP)
- Words: 1
- Cycles: 2

PUSH instruction

- Syntax: *push Rr*
- Operands: Rr∈{r0, r1, …, r31}
- Operation: (SP) ← Rr
  SP ← SP –1
- Words: 1
- Cycles: 2

Παράδειγμα κλήσης συνάρτησης στη γλώσσα C

```
// int parameters b & e,
// returns an integer

unsigned int pow(unsigned int b, unsigned int e)
{
        unsigned int i, p;          // local variables
        p = 1;
        for (i = 0;  i  < e; i++)   // p = b^e
                p = p * b;
        return p;                   // return value of the function
}

int main(void)
{
        unsigned int m, n;
        m = 2;
        n = 3;
        m = pow(m, n);
        return 0;
}
```

Παράδειγμα κλήσης συνάρτησης στη γλώσσα C

```
// int parameters b & e,
// returns an integer
                                        Parameter Type → Passing By Value
unsigned int pow(unsigned int b, unsigned int e)
{
        unsigned int i, p;          // local variables
        p = 1;
        for (i = 0;  i  < e; i++)   // p = b^e
                p = p * b;
        return p;                   // return value of the function
}                       Return Value
    CALLER
int main(void)
{
        unsigned int m, n;
        m = 2;
        n = 3;     CALLEE
        m = pow(m, n);
        return 0;
}
```

Calling conventions in WINAVR C compiler

- Οι registers **r31:r30:r27:r26** → Επιστροφή αποτελέσματος συνάρτησης ανάλογα με τον τύπο δεδομένων (char, short, int).

char => r30, short => r31:r30, int => r31:r30:r27:r26

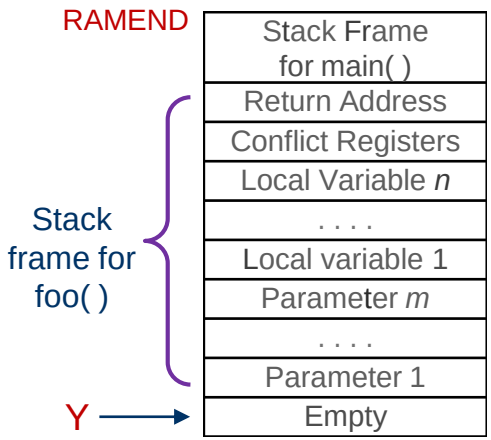
- Register **y=r29:r28** → stack frame pointer (δείκτης στο τοπικό stack frame κάθε συνάρτησης).
- Ο SP του AVR χρησιμεύει στις κλίσεις-επιστροφές των ρουτινών όταν το πρόγραμμα μεταφράζεται σε assembly.
- Οι conflict registers του CALLER αποθηκεύονται στη στοίβα αυτή.
  - Register conflicts → τουλάχιστον ένας από τους καταχωρητές του AVR χρησιμοποιείται και στη συνάρτηση CALLER και στη συνάρτηση CALLEE.

Stack frame

Stack frame:

- Return address
- Conflict registers
- Local variables
- Parameters (arguments)

```
int main(void)
{ …
     foo(arg1, arg2, …, arg m);
}

void foo(arg1, arg2, …, arg m)
{
     int var1, var2, …, var n;
…
}
```



Κλήση Συναρτήσεων- Μετάφραση σε Assembly

- ✓ Σε αυτήν την Παράγραφο θα μελετήσουμε πώς γίνεται η μετάφραση σε assembly, με τον compiler της C, μιας συνάρτησης.
- ✓ Για να μπορεί να υποστηρίζει κλήση συνάρτησης μέσα από άλλη συνάρτηση διαμορφώνεται μια στοίβα ορισμάτων που χρησιμοποιούνται από την τρέχουσα συνάρτηση. Όταν αυτή τερματίζεται επιστρέφει ο **δείκτης της στοίβας** αυτής στην αρχική της τιμή. Στο συγκεκριμένο compiler για τους Μικροελεγκτής AVR είναι ο διπλός καταχωρητής δείκτη **Y=R29:R28**.
- ✓ Οι μεταβλητές του Προγράμματος αποθηκεύονται στους καταχωρητές R16, R17, … Ανάλογα με τον τύπο τους καταλαμβάνουνε 1, 2 ή 4 καταχωρητές.
- ✓ Πριν την κλήση της κάθε συνάρτησης τα ορίσματα της αποθηκεύονται στη στοίβα ορισμάτων. Η καλουμένη συνάρτηση κάνει ανάκτηση των ορισμάτων (σε καταχωρητές), εκτελείται η επεξεργασία που περιγράφεται από το σώμα της συνάρτησης και **το αποτέλεσμα επιστρέφεται μέσω των καταχωρητών R31:R30:R27:R26 ανάλογα με το τύπο του (char, short, int).**

```
#include <mega16.h>
short function(char x, char y)
{
      return (short)(x+y);
}
void main()
{
      char x='z';
      char y='b';
      short z;
      z=function(x,y);
}
```

Μετάφραση σε Assembly

Αυτό το πρόγραμμα περιέχει δύο μεταβλητές των 8 bit (τις μεταβλητές x και y) και καλεί την 16 bit συνάρτηση function, η οποία επιστρέφει ως αποτέλεσμα το άθροισμα (x+y). Ο Disassembler του AVR Studio δίνει ως αποτέλεσμα τον ακόλουθο κώδικα :

```
   JMP main              ; άλμα στην κύρια συνάρτηση
function:                ; η ρουτίνα που υλοποιεί τη συνάρτηση z=function(x,y)
   LDD R26,Y+0           ; φόρτωση ορίσματος x από τη στοίβα
   LDD R30,Y+1           ; φόρτωση ορίσματος y από τη στοίβα
   ADD R30,R26           ; εκτέλεση πρόσθεσης
   LDI R31,0x00          ; επέκταση αποτελέσματος στα 16 bit
   ADIW R28,0x02         ; επαναφορά δείκτη στοίβας ορισμάτων Y=R29:R28
   RET                                ; επιστροφή
; σημείο εισόδου του προγράμματος, φόρτωση μεταβλητών και προετοιμασία
; στοίβας ορισμάτων για την κλίση της συνάρτησης: z=function(x,y);
main:
   LDI R16,0x7A          ; ανάθεση τιμής στη μεταβλητή x
   LDI R17,0x62          ; και στη μεταβλητή y
   ST  -Y,R16            ; αποθήκευση ορισμάτων της συνάρτησης
   ST  -Y,R17            ; στη στοίβα Y
   RCALL function        ; κλίση της συνάρτησης z. Ο SP του AVR χρησιμεύει στις κλίσεις των ρουτινών
   MOV R18,R30           ; μεταφορά του αποτελέσματος z από τους R31:R30
   ; (που αποτελούν τους καταχωρητές που επιστρέφουν το αποτέλεσμα – short)
  ; στον καταχωρητή της μεταβλητής z=R19:R18
wait:
   RJMP wait                          ; ατέρμων βρόχος αναμονής
```

2ο Παράδειγμα

Ένα δεύτερο παράδειγμα που δείχνει καλύτερα το πώς διαμορφώνεται η εικονική στοίβα των ορισμάτων που χρησιμοποιεί ο compiler του AVR είναι το ακόλουθο πρόγραμμα C :

```
#include <mega16.h>
short function(char x, char y, char w)
{
    return (short)((x+w)*y);
}
void main()
{
    char x='z';
    char y='b';
    char w='e';
    short z;
    z=function(x,y,w);
}
```

Μετάφραση σε Assembly της συνάρτησης z

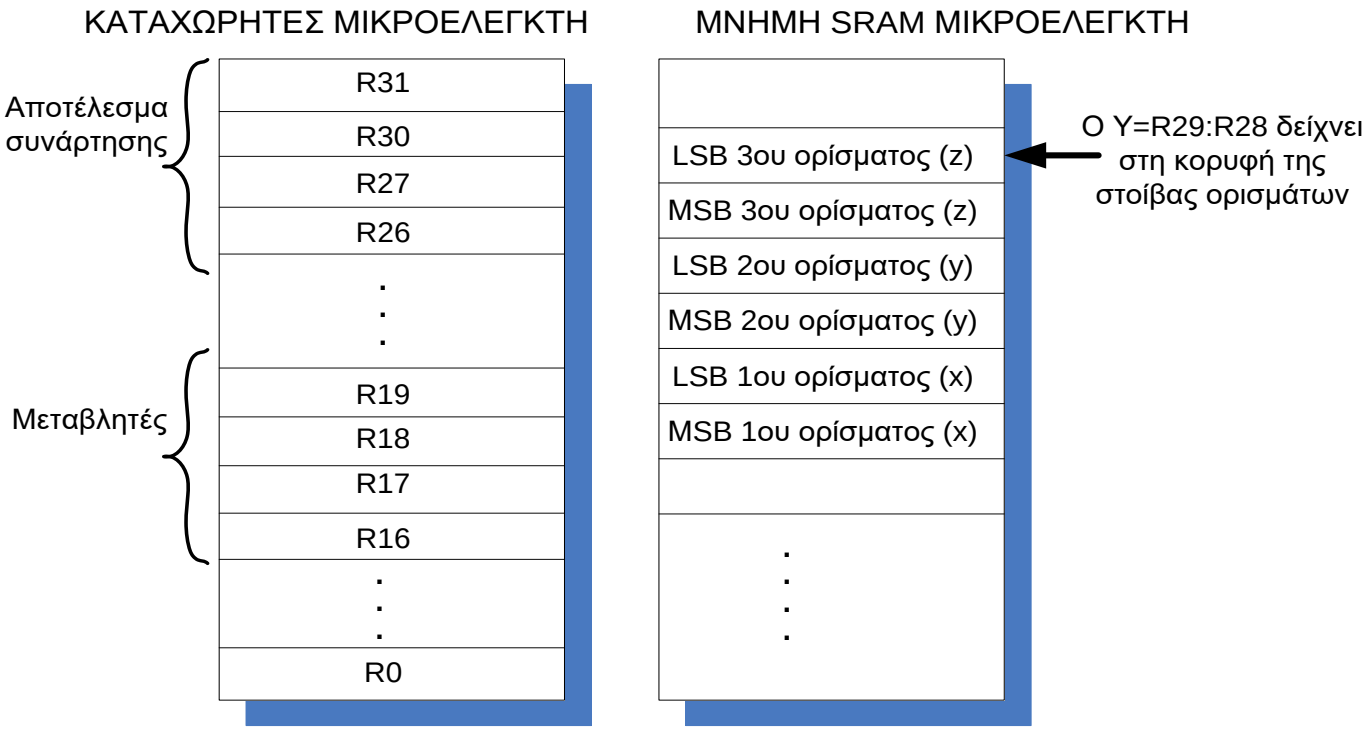
Εδώ η συνάρτηση δέχεται τρία ορίσματα των 8 bit, τα x, y και w, και επιστρέφει ως 16 bit αποτέλεσμα την έκφραση (x+w)*y. Ο Dissasembler παράγει τον ακόλουθο κώδικα σε assembly:

```
   JMP main                ; Άλμα στην κυριά συνάρτηση
function:                  ; η ρουτίνα που υλοποιεί τη συνάρτηση z=function(x,y)

LDD R26,Y+0                ; φόρτωση ορίσματος LOW(x) από τη στοίβα
   LDD R30,Y+2             ; φόρτωσης ορίσματος LOW(w) από τη στοίβα
   ADD R30,R26             ; εκτέλεση Πρόσθεσης
   MOV R26,R30
   LDD R30,Y+1             ; φόρτωσης ορίσματος y από τη στοίβα
   MUL R30,R26             ; Εκτέλεση Πολλαπλασιασμού, από το γινόμενο
   MOV R30,R0              ; κρατάει τα 8 LSB (μεταβλητές τύπου char)
   LDI R31,0x00            ; επέκταση αποτελέσματος στα 16 bit
   ADIW R28,0x03           ; επαναφορά δείκτη στοίβας ορισμάτων Y=R29:R28
   RET                     ; επιστροφή
```

Μετάφραση σε Assembly της κύριας συνάρτησης

```
; σημείο εισόδου του προγράμματος, φόρτωση μεταβλητών και προετοιμασία
; στοίβας ορισμάτων για την κλίση της συνάρτησης: z=function(x,y)

main:
   LDI R16,0x7A                ; ανάθεση τιμής στη μεταβλητή x
   LDI R17,0x62                ; στη μεταβλητή y
   LDI R16,0x7A                ; και στη μεταβλητή w
   ST  -Y,R16                  ; αποθήκευση ορισμάτων της συνάρτησης
   ST  -Y,R17                  ; στη στοίβα Y
   ST  -Y,R18
RCALL function          ; κλίση της συνάρτησης z
   MOVW R19,R30                ; μεταφορά του αποτελέσματος z από τους R31:R30
                        ; (αποτελούν τους καταχωρητές επιστροφής του αποτελέσματος – short)
                               ; στον καταχωρητή της μεταβλητής z=R20:R19
wait:
   RJMP wait                   ; ατέρμων βρόχος αναμονής
```

Η στοίβα πριν την κλήση της συνάρτησης και η χρήση των καταχωρητών



Φωλιασμένες Κλήσεις Συναρτήσεων

Ένα πρόγραμμα C, το οποίο έχει τέσσερις 8 bit μεταβλητές (τύπου char), τις x, y, z και w, καλεί την συνάρτηση madd, η οποία δέχεται τέσσερα 8 bit ορίσματα. Αυτή η συνάρτηση καλεί δύο φορές την συνάρτηση fadd, η οποία είναι μία συνάρτηση που επιτελεί την πράξη της πρόσθεσης για δύο 8 bit μεταβλητές (τύπου char), για να υπολογίσει τα επιμέρους αθροίσματα (x+y) και (z+w). Η συνάρτηση στο τέλος επιστρέφει το συνολικό άθροισμα των τεσσάρων 8 bit μεταβλητών με μία νέα κλήση της fadd, με ορίσματα τα αποτελέσματα των δύο προηγούμενων κλήσεων της fadd. Ακολουθεί ο κώδικας του προγράμματος στη γλώσσα C :

```
#include <mega16.h>
char fadd(char x,char y)
{
      return x+y;
}
char madd(char x,char y,char z,char w)
{
      return fadd(fadd(x,y),fadd(z,w));
}
void main()
{
      char x='f';
      char y='d';
      char z='b';
      char w='z';
      char result;
      result= madd(x,y,z,w);
 }
```

Μετάφραση σε Assembly (1/3)

Ο Disassembler του AVRStudio δίνει ως αποτέλεσμα τον ακόλουθο κώδικα assembly :

```
   JMP main          ; άλμα στην κύρια συνάρτηση
fadd:                ; η ρουτίνα που υλοποιεί τη συνάρτηση fadd(x,y)=x+y
   LDD R30,Y+0       ; φόρτωση ορίσματος (x) από τη στοίβα
   LDD R26,Y+1       ; φόρτωση ορίσματος (y) από τη στοίβα
   ADD R30,R26       ; εκτέλεση πρόσθεσης, αποτέλεσμα στον R30
   ADIW R28,0x02     ; επαναφορά δείκτη στοίβας ορισμάτων
   Y=R29:R28
   RET               ; επιστροφή
```

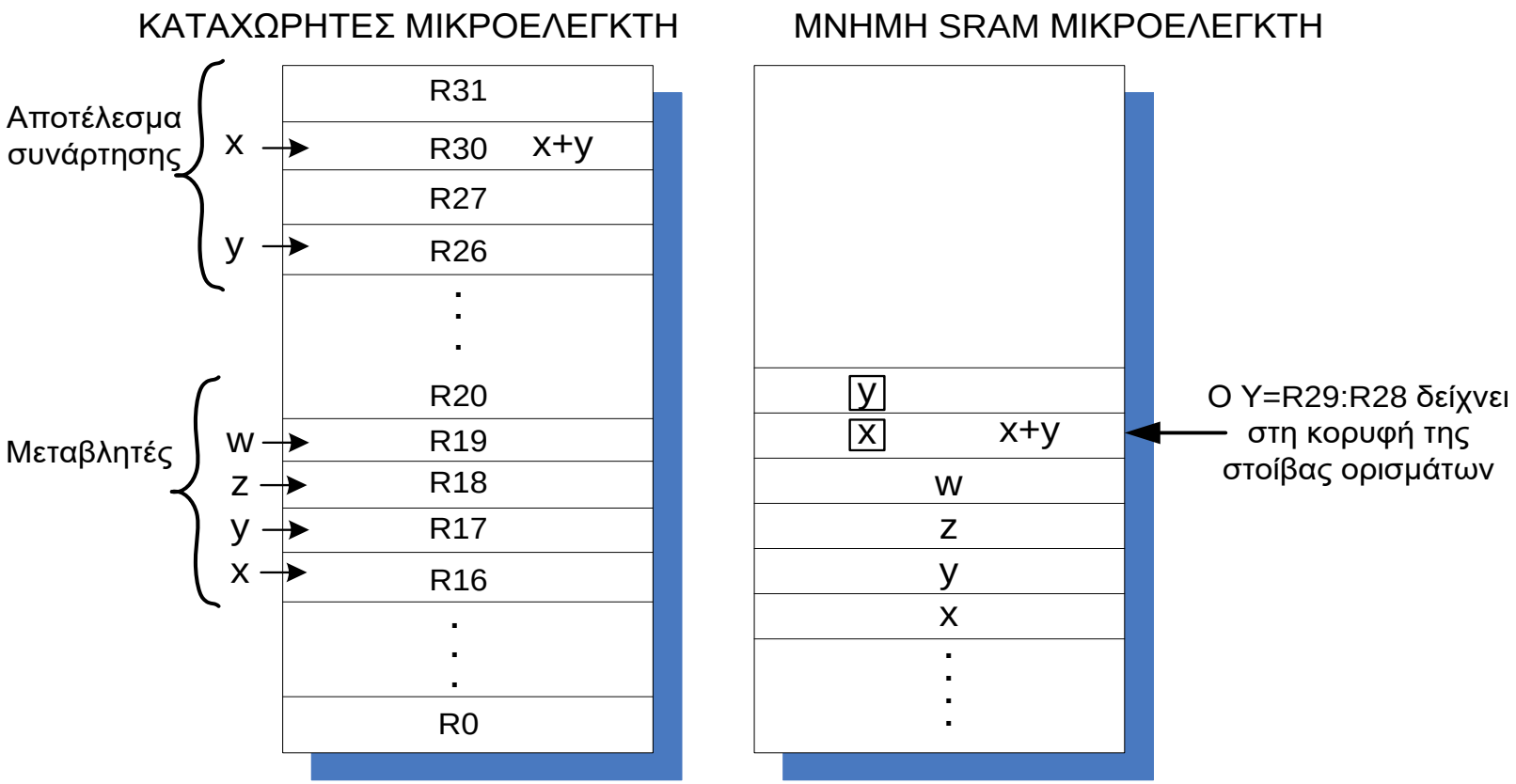
Μετάφραση σε Assembly (2/3)

```
madd:                ; η ρουτίνα που υλοποιεί τη συνάρτηση madd(x,y,z,w)
   LDD R30,Y+3       ; φόρτωση ορίσματος x από τη στοίβα
   ST  -Y,R30        ; αποθήκευση ορίσματος x της συνάρτησης fadd
   LDD R26,Y+3       ; φόρτωση ορίσματος y από τη στοίβα
   ST  -Y,R30        ; αποθήκευση ορίσματος y της συνάρτησης fadd
   RCALL fadd                ; κλίση της συνάρτησης fadd

bp1:ST  -Y,R30       ; αποθήκευση (x+y) στη στοίβα ορισμάτων
   LDD R30,Y+2       ; φόρτωση ορίσματος z από τη στοίβα
   ST  -Y,R30        ; αποθήκευση ορίσματος z της fadd
   LDD R26,Y+2       ; φόρτωση ορίσματος w από τη στοίβα
   ST  -Y,R30        ; αποθήκευση ορίσματος w της fadd
   RCALL fadd                ; κλίση της συνάρτησης fadd

bp2:ST  -Y,R30       ; αποθήκευση αποτελέσματος-ορίσματος z+w της fadd
   RCALL fadd        ; κλίση της συνάρτησης fadd
   ADIW R28,0x04     ; επαναφορά δείκτη στοίβας ορισμάτων Y=R29:R28
   RET                       ; επιστροφή
```

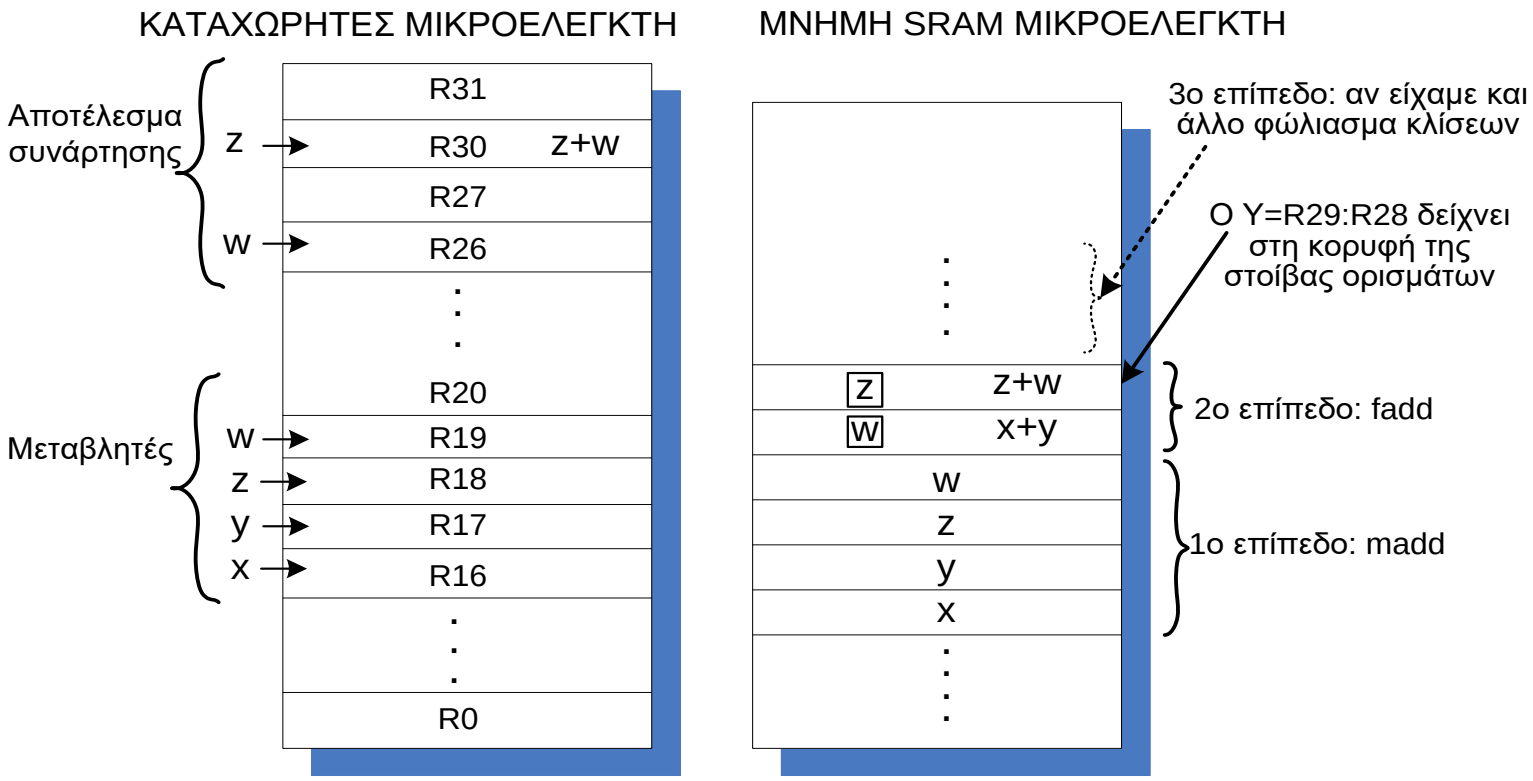
Μετάφραση σε Assembly (3/3)

```
; σημείο εισόδου του προγράμματος, φόρτωση μεταβλητών και προετοιμασία
; στοίβας ορισμάτων για την κλίση της συνάρτησης: madd(x,y,z,w);
main:
   LDI R16,0x66              ; ανάθεση τιμής στη μεταβλητή x='f'
   LDI R17,0x64              ; στη μεταβλητή y='d'
   LDI R18,0x62              ; στη μεταβλητή z='b'
   LDI R18,0x7A              ; και στη μεταβλητή w='z'
   ST  -Y,R16                ; αποθήκευση ορισμάτων x, y, z, w
   ST  -Y,R17                ; της συνάρτησης madd(x,y,z,w)
   ST  -Y,R18                ; στη στοίβα Y
   ST  -Y,R19
   RCALL madd       ; κλίση της συνάρτησης
   MOV R20,R30               ; μεταφορά του αποτελέσματος madd από τον
                    ; καταχωρητή R30 αποτελέσματος στον καταχωρητή R20
wait:   RJMP wait            ; ατέρμων βρόχος αναμονής
```

Στοίβα και καταχωρητές στο σημείο bp1 του προγράμματος



Στοίβα και καταχωρητές στο σημείο bp2 του προγράμματος



Πρόγραμμα διαχείρισης διακοπών σε 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 <mega16.h>
unsigned char chLed;
                              // External Interrupt 0 service routine. Η EXT_INT0 αντιστοιχεί στη θέση 02-03
interrupt [EXT_INT0] void ext_int0_isr(void)
{
  chLed = chLed ^ 0xFF;
  PORTB=chLed;
}

void main(void)
{
  PORTB=0x00;           // Η PORTB τίθεται ως έξοδος
  DDRB=0xFF;
  GIMSK=0x40;           // Ενεργοποίηση εξωτερικής διακοπής 0,  INT0: On
  MCUCR=0x02;           // INT0 Mode: στην ακμή πτώσης
  #asm("sei")           // ενεργοποίηση συνολικά των διακοπών
  chLed=0x00;
  PORTB=chLed;
  while (1)
      {                 // Μπορούμε τοποθετώντας πρόσθετο κώδικα στο σημείο αυτό,
                        // το κύριο πρόγραμμα να κάνει επιπλέον λειτουργίες εκτός του ON – OFF των led.
      };
}
```

Αναφορές

- Κ. Πεκμεστζή, “Συστήματα Μικροϋπολογιστών – Μικροελεγκτής AVR και PIC”
- William Barnekow, “Mixing C and assembly language programs”, Lecture Notes, Cornel University
- R. Barnett, S. Cox and L. O’Cull, “Embedded C Programming and the Atmel AVR”, Delmar, Cengage 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/