

# ATmega103 Assembly I

---

2/20/2004

Costas Foudas, Imperial College,  
Rm: 508, x47590

1



# Outline:

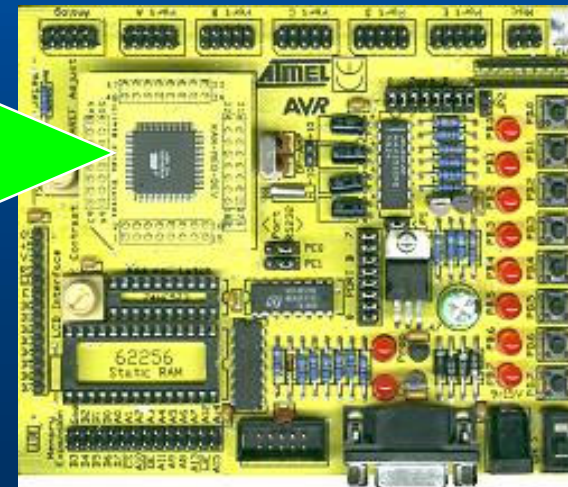
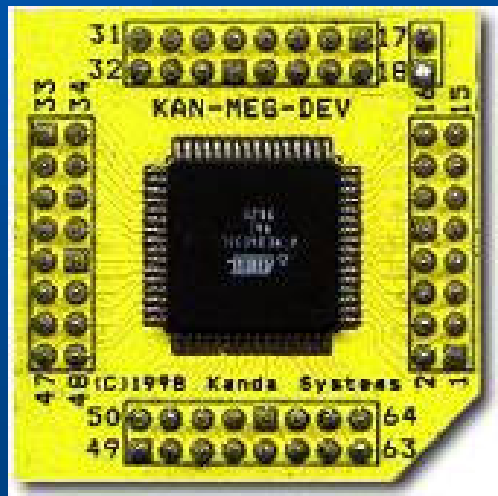
---

- ATmega103 architecture
- AVR assembly language
- Elementary example program
- AVR Assembler
- Using the **STUDIO3.52** simulator
- Downloading with **PonyProg**



# The ATmega103 Microprocessor

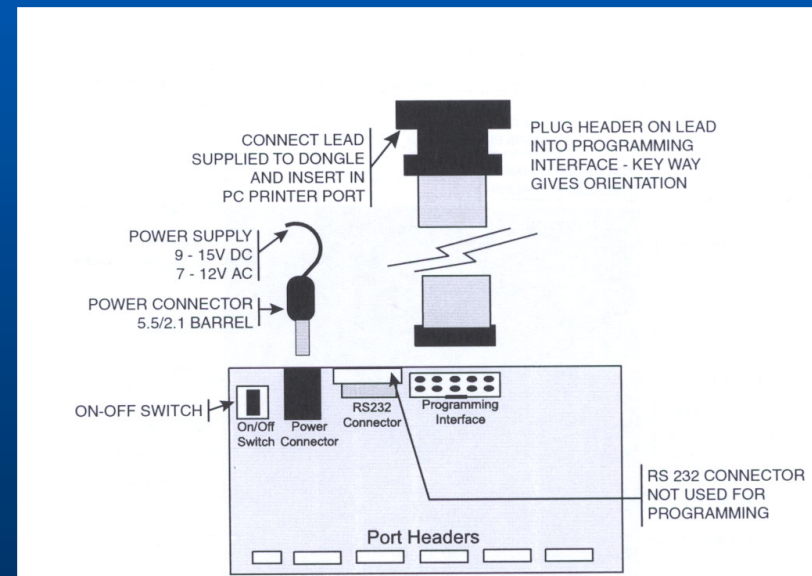
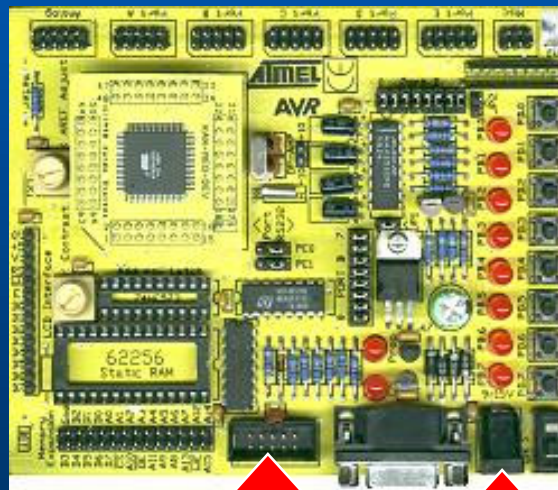
- In this course you will be using the ATmega103 processor mounted on an ATMEL programming board*





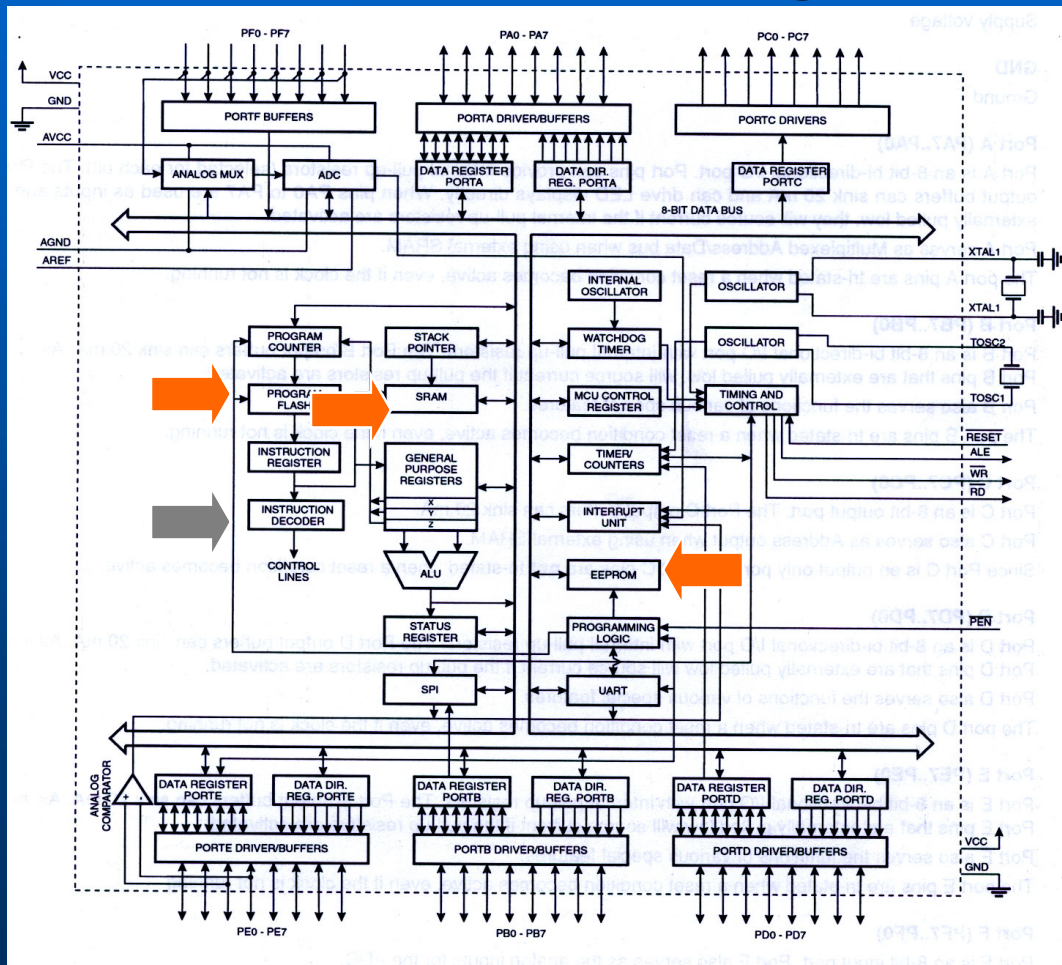
# The ATMEL BOARD

- *Connecting the board with your PC*





# ATmega103 diagram



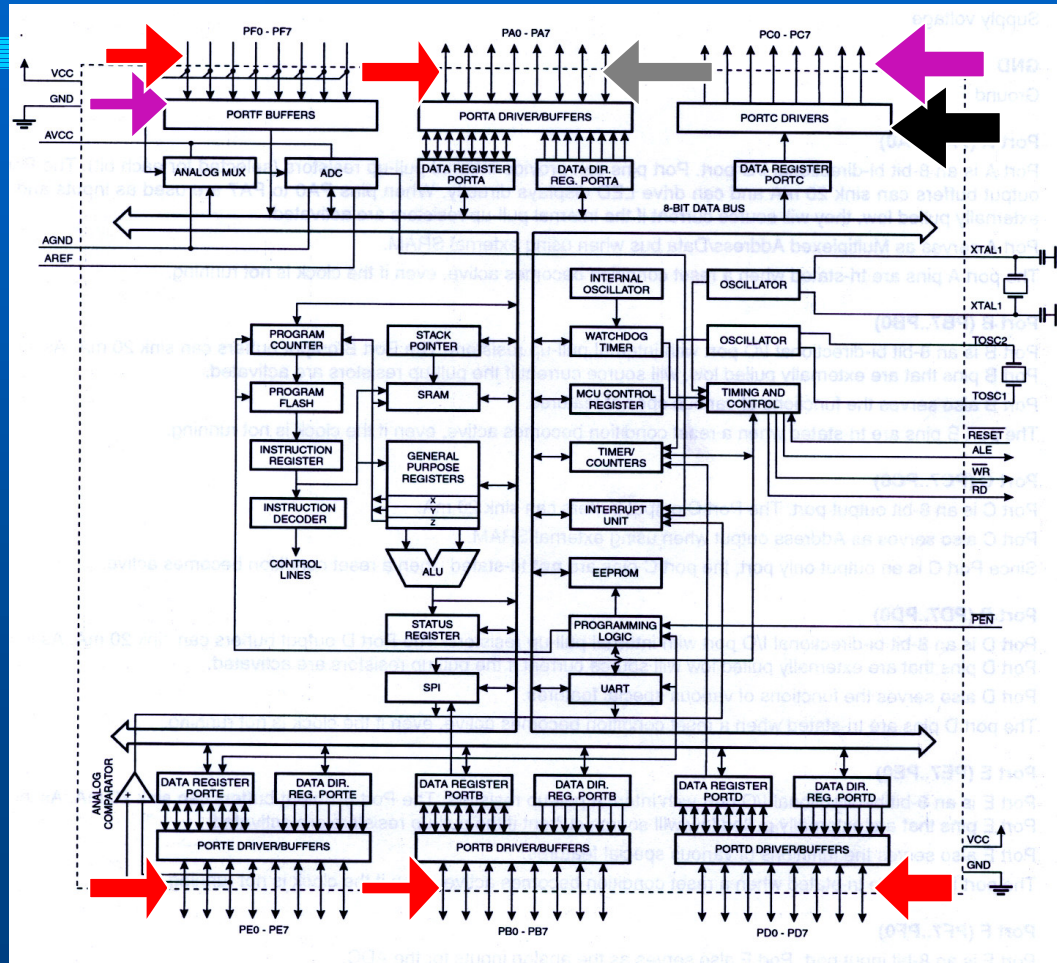
- ♣ RISC Architecture
- ♣ 121 Instructions
- ♣ 32x8 Registers
- ♣ 4 MIPS @ 4 MHz

- ♣ 128 Kbytes In-System  
Prog. Flash Memory
- ♣ 4 Kbytes SRAM
- ♣ 4 Kbytes In-System  
EEPROM





# ATmega103 Peripherals I



- ♣ 6 PORTS;
- ♣ 4 Bi-Directional

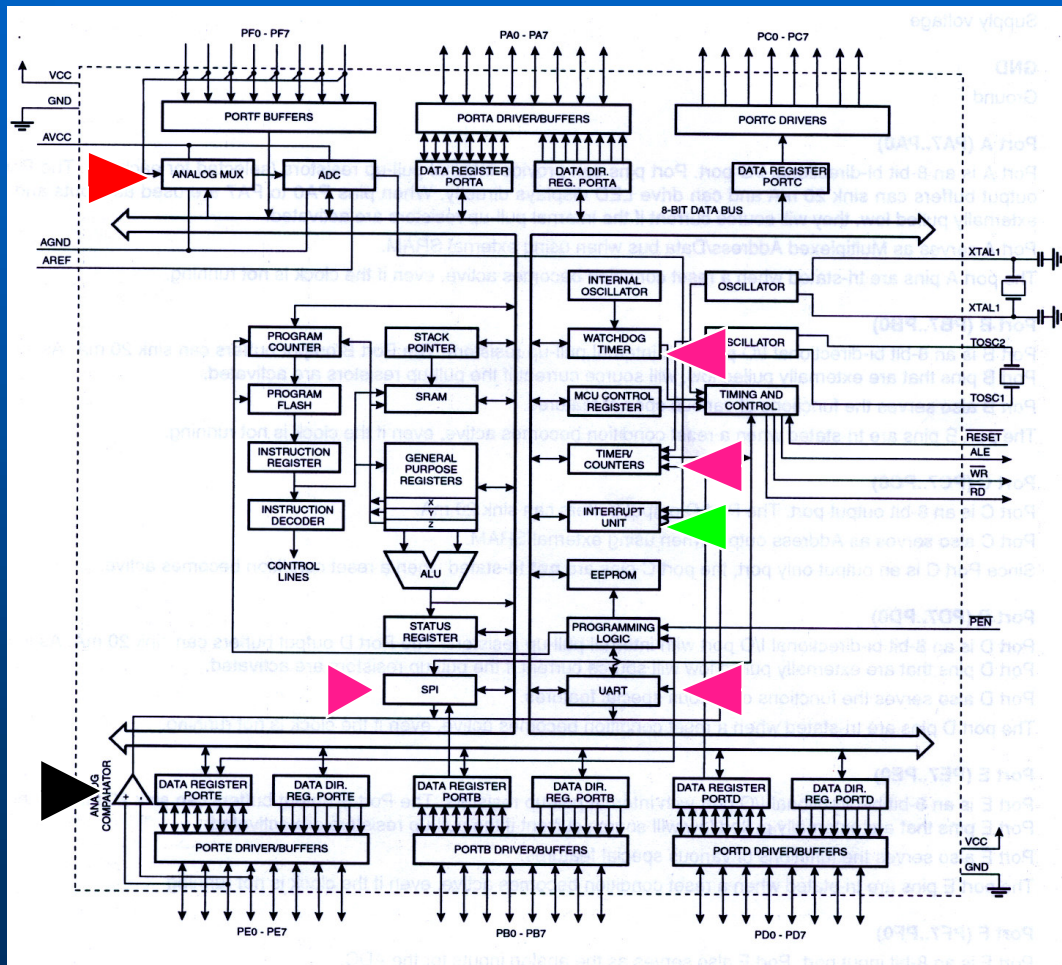
- ♣ 1 Output
- ♣ 1 Input

♣ Port A 8-bit I/O  
SRAM ADDR/DATA

♣ Port C SRAM  
♣ ADDRESS



# ATmega103 Peripherals II



## Interrupt Unit

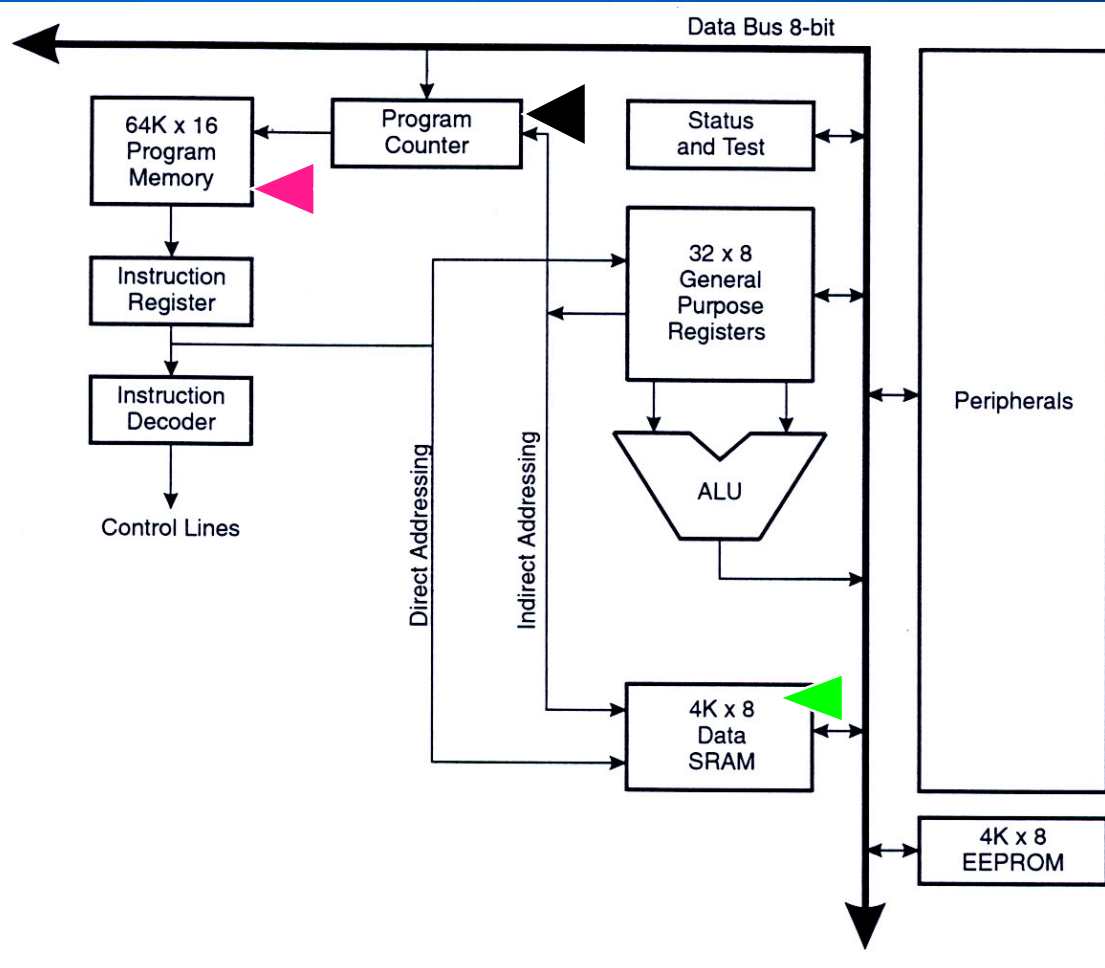
**8-Channel 10 Bit ADC  
with interrupt**

**On Chip analog  
Comparator with  
Interrupt.**

- ♣ Prog. Watchdog Timer
- ♣ Real Time Counter
- ♣ SPI Interface
- ♣ UART



# The ATmega103 Architecture I



♣ Your Program resides in this 128 Kbytes memory in .hex Intel format (hex numbers )

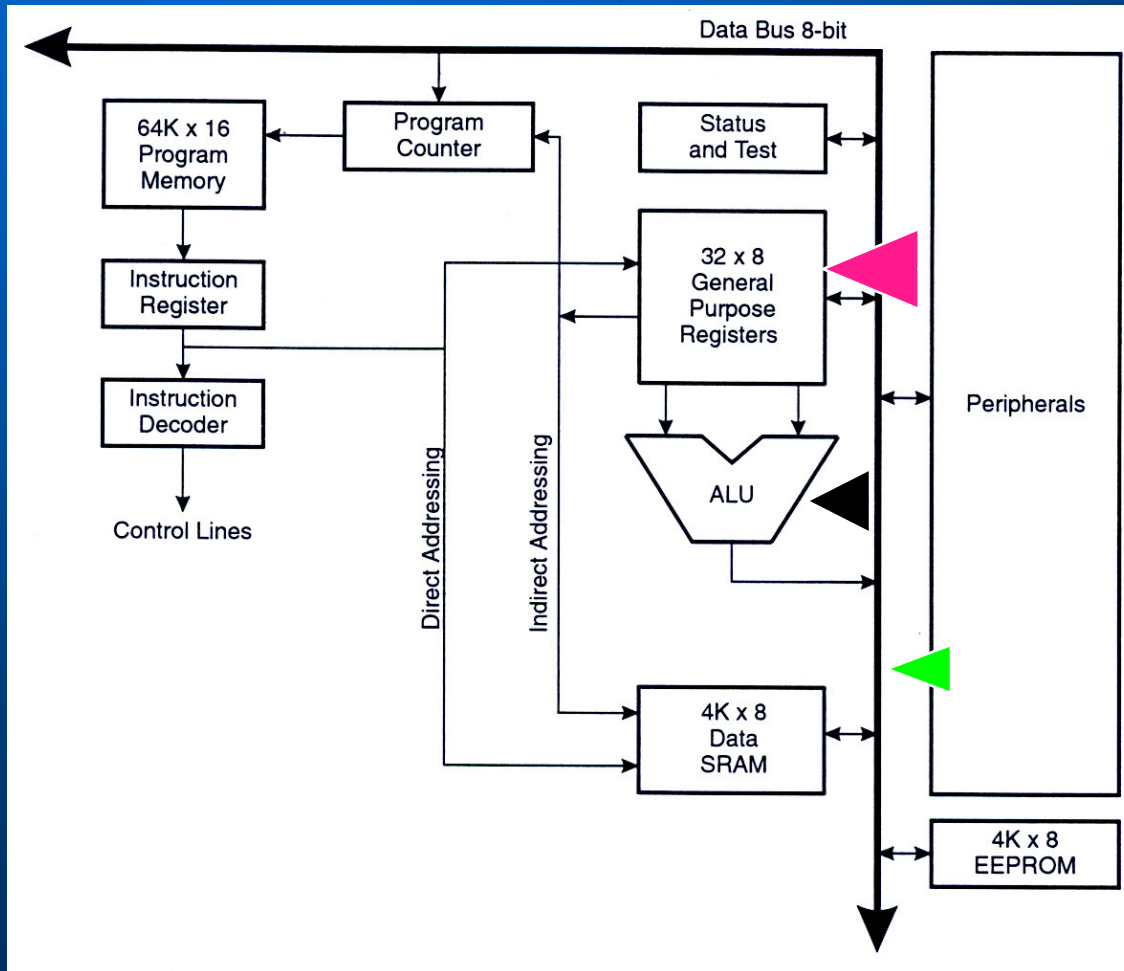
**The Program Counter Keeps track which instruction is to be executed next**

The PC value at a subroutine call is stored in SRAM and after the subroutine execution (return) it is increased by 1 and loaded back.





# The ATmega103 Architecture II



**32 General Purpose Registers**

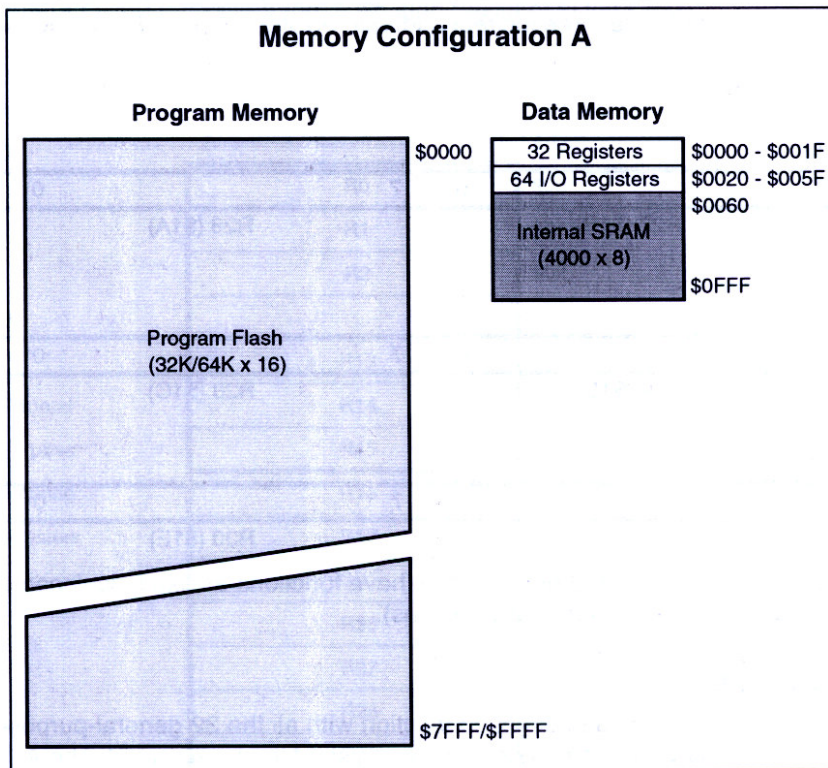
**The Arithmetic Logic Unit operates on the 32 registers**

**8-bit Data Bus**

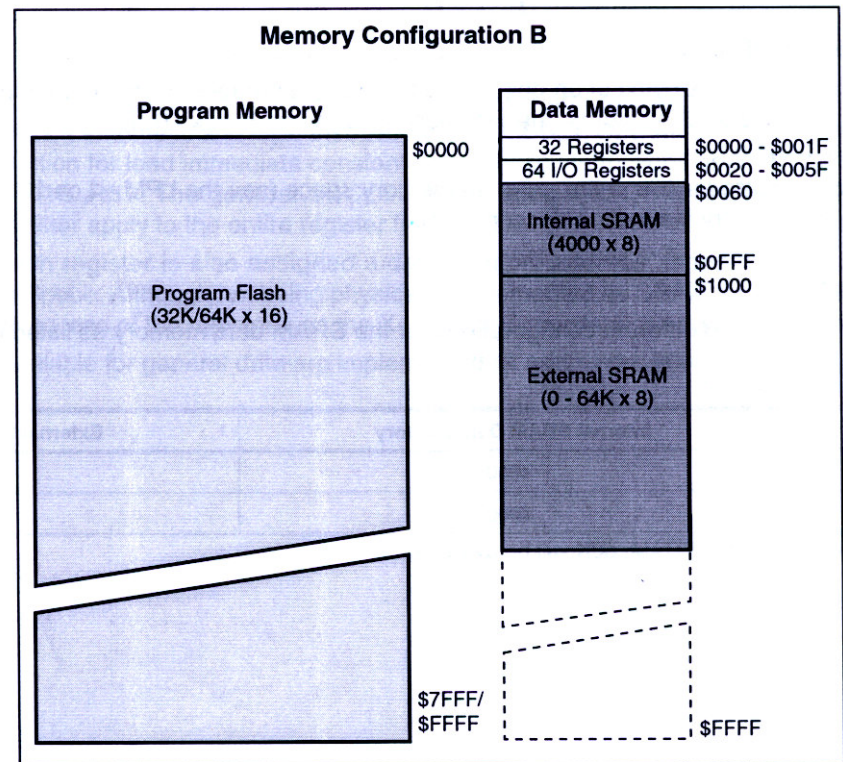


# The ATmega103 Memory Map

**Memory Configuration A**



**Memory Configuration B**





# AVR Assembly Language

## Why Assembly ?

- *Direct access to the architecture of the processor*
- *Direct use of the machine registers memory and stack*
- *Full control of the processor*
- *Faster*



# Registers

- There are 32 registers **r0-r31** on ATmega103. You may name them in a way that you can remember:

Example: **.def InPutRegister = r16**

- You can zero (clear) them by:

Example: **clr r16 ; This would load \$00 on r16**



# Registers I

- You can set them to ones by:

Example: **ser r16** ; *This would load \$FF on r16*

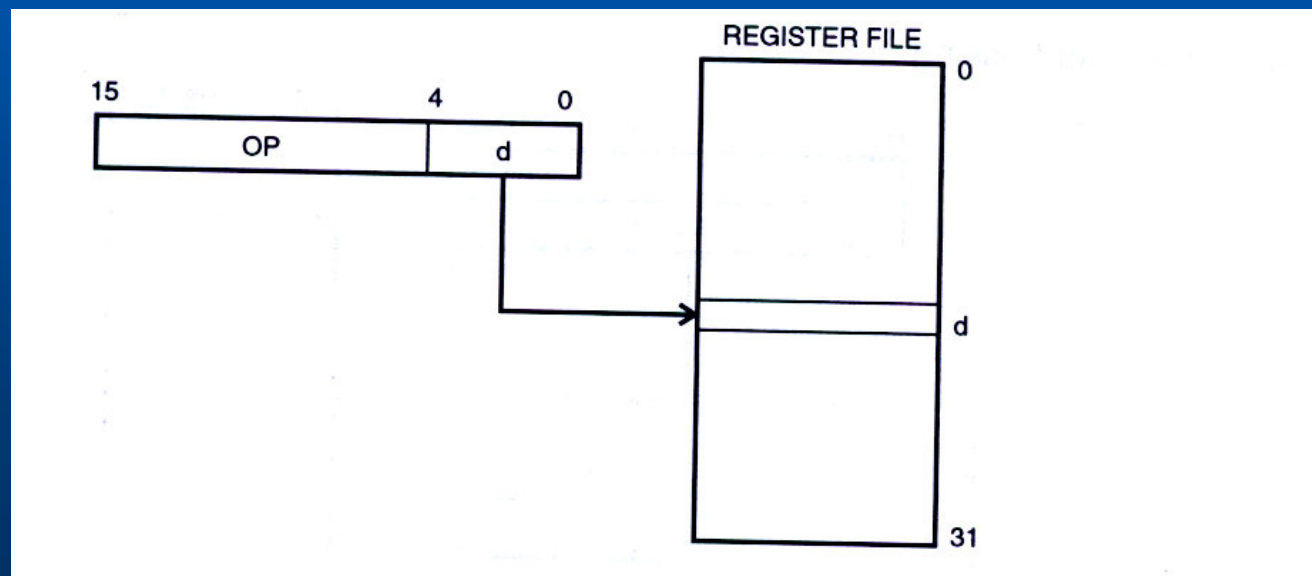
**Remember the D-Flip-Flop ??**





# Registers II

- Both CLR and SER are **Direct Single Register Addressing** because:





# Loading a register

- You can set the contents of r16 by: **ldi InPutRegister, \$AA**

The *ldi* command will load with the HEX value \$AA to register *InPutRegister* which is just *r16*.



# Two register commands

- Introduce one more register r15:

```
.def RegisterTwo = r15
```

- The following command:

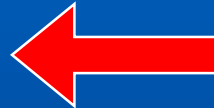
```
mov RegisterTwo, InPutRegister
```

**will transfer the contents of  
r16 (\$AA) to r15**

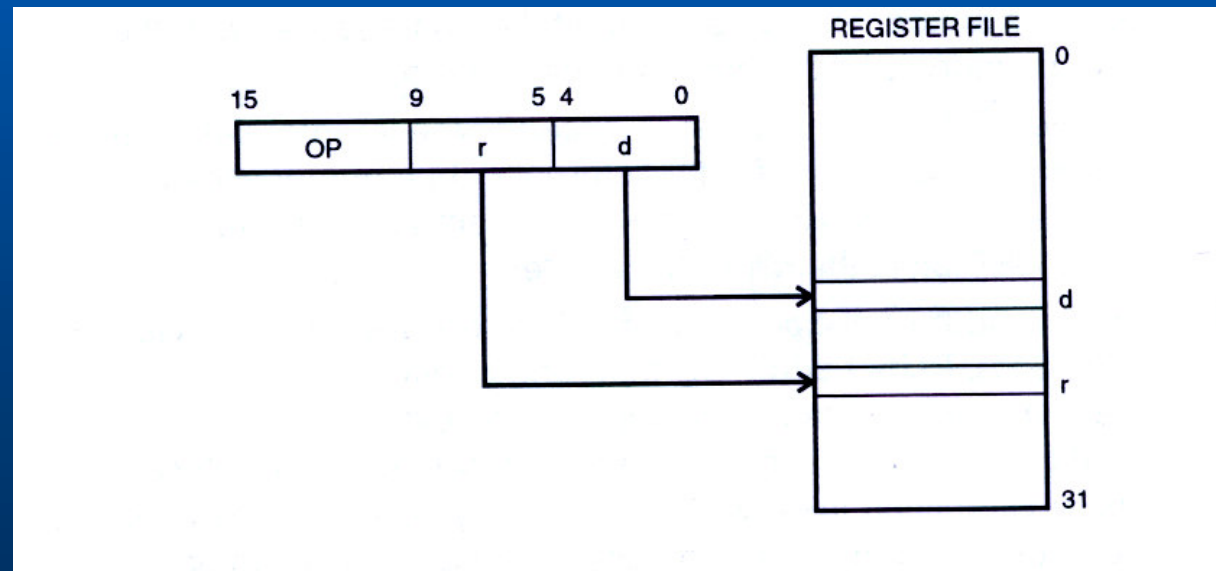


# Direct two register

**MOV Rd, Rx**



**Note: that the direction goes  
Against intuition:**

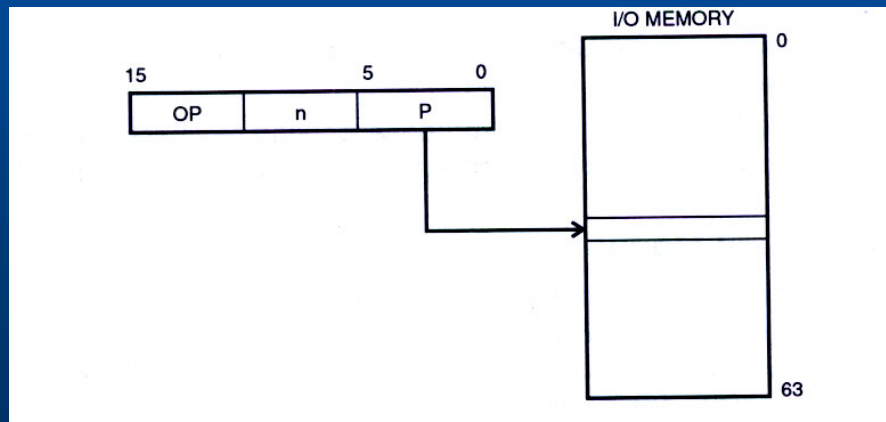




# I/O Direct Addressing

To read or write to the ATmega103 ports use the commands:

**IN Rd, PINX** ←  
**OUT PORTX, Rx ;** X is A-F









# Setting up a port:

## PORTB : OUTPUT PORT

```
; ***** Port B Setup Code *****  
    ldi r16, $FF ;  
    out DDRB, r16 ; Port B Direction Register  
    ldi r16, $FF ; Init value  
    out PORTB, r16 ; Port B value  
  
; ***** Port D Setup Code *****  
    ldi r16, $00 ; I/O:  
    out DDRD, r16 ; Port D Direction Register  
    ldi r16, $FF ; Init value  
    out PORTD, r16 ; Port D value
```

## PORTD : INPUT PORT



# Program Header I:

```
; ** ATmega103(L) Assembly Language File - IAR Assembler Syntax **
```

```
.ORG 0
```

```
.include "m103def.inc" ; Add required path to IAR Directory
```

```
RJMP Init
```

```
; ** Author : Costas Foudas
```

```
; ** Company : Imperial College
```

```
; ** Comment : Program Header; PORTB=OUT, PORTD=IN
```



# Program Header II:

**Init:**

**; \*\*\*\*\* Stack Pointer Setup Code**

```
ldi r16, $0F           ; Stack Pointer Setup
out SPH,r16            ; Stack Pointer High Byte
ldi r16, $FF           ; Stack Pointer Setup
out SPL,r16            ; Stack Pointer Low Byte
```

**; \*\*\*\*\* RAMPZ Setup Code \*\*\*\*\***

```
ldi r16, $00           ; 1 = EPLM acts on upper 64K
out RAMPZ, r16         ; 0 = EPLM acts on lower 64K
```

**; \*\*\*\*\* Comparator Setup Code \*\*\*\*\***

```
ldi r16,$80            ; Comparator Disabled, Input
                        ; Capture Disabled
out ACSR, r16          ; Comparator Settings
```



# Example LED Program:

Main:

```
IN r16, PIND
```

```
OUT PORTB, r16
```

```
rjmp Main
```

Read in PortD

Write the PortD  
input to the PortB  
output register

Go back to Main





# Setting up a code directory:

Open a directory where you will store you code:

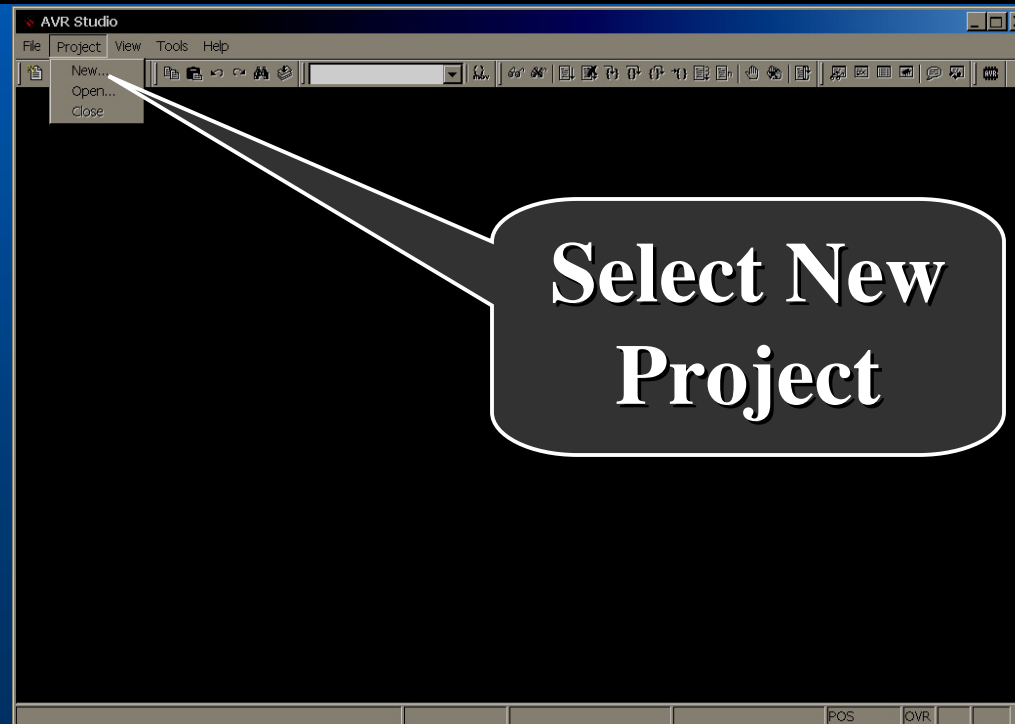
Down-load in this directory the files:  
LEDIO.ASM and m103def.inc from the course  
web page (Lecture 2b):

Make sure you have : LEDIO.ASM,  
m103def.inc  
in your code directory



# Getting Started with STUDIO 3.52 :

Go to **Start** → **Programs** → **ATMEL AVR Tools**  
→ **AVR Studio 3.52**





# Getting Started with STUDIO 3.52 :

You should be getting now the window:

**Select new project**

Project name: [?]

Location: H:\My\_directories\Teaching\Micro-Course\Mega

Project type: AVR Assembler, Generic 3rd party C [?]

**Select new project**

Project name: Project1

Location: H:\My\_directories\Teaching\Micro-Course\Mega

Project type: AVR Assembler, Generic 3rd party C

**Pick AVR Assembler**

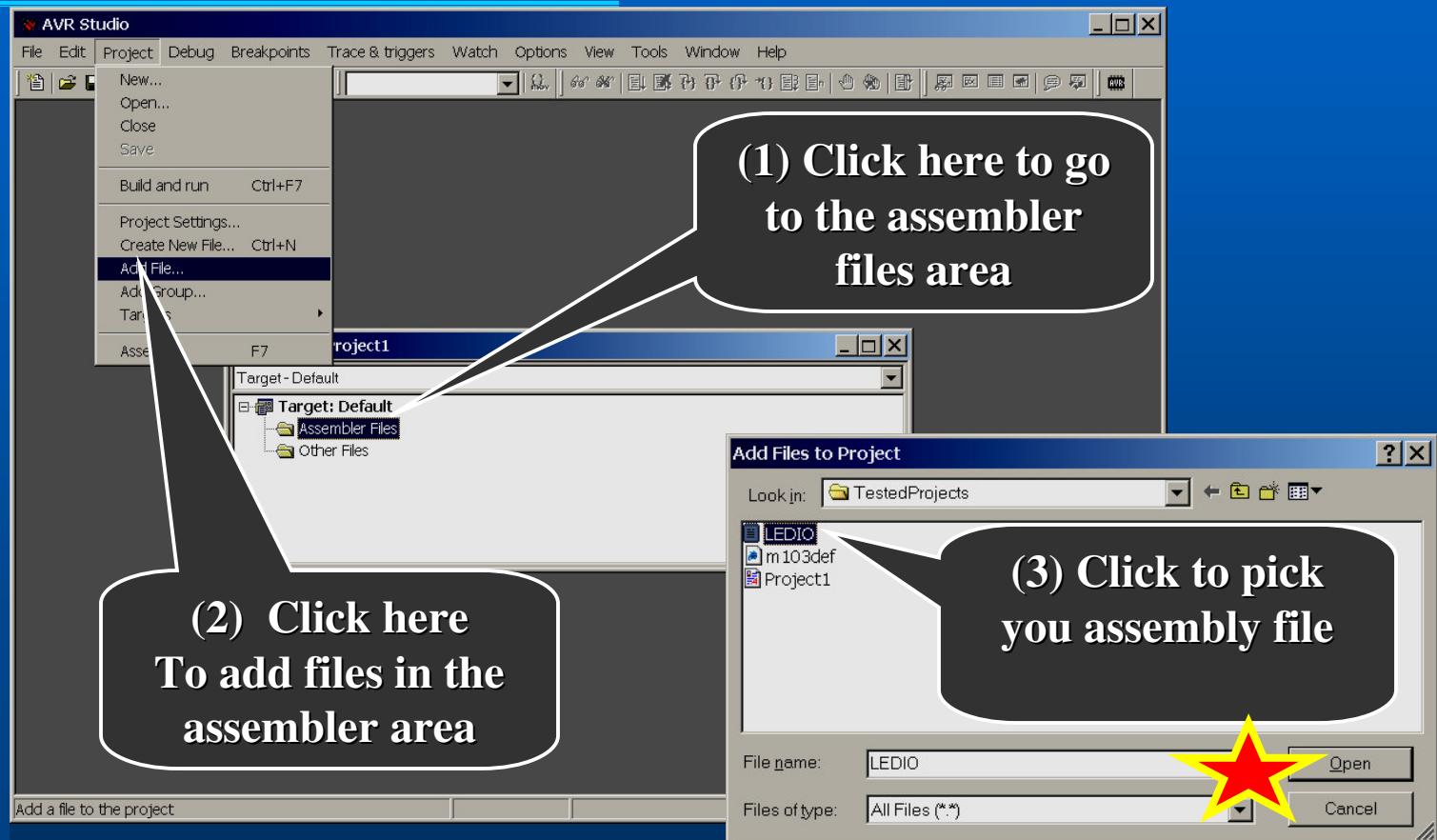
**This is your code directory**

**At the end**

**Pick a name for your project**



# Entering files in STUDIO3.52 (I):





# Entering files in STUDIO3.52 (II):

(2) Click here to add files in the 'other files' area

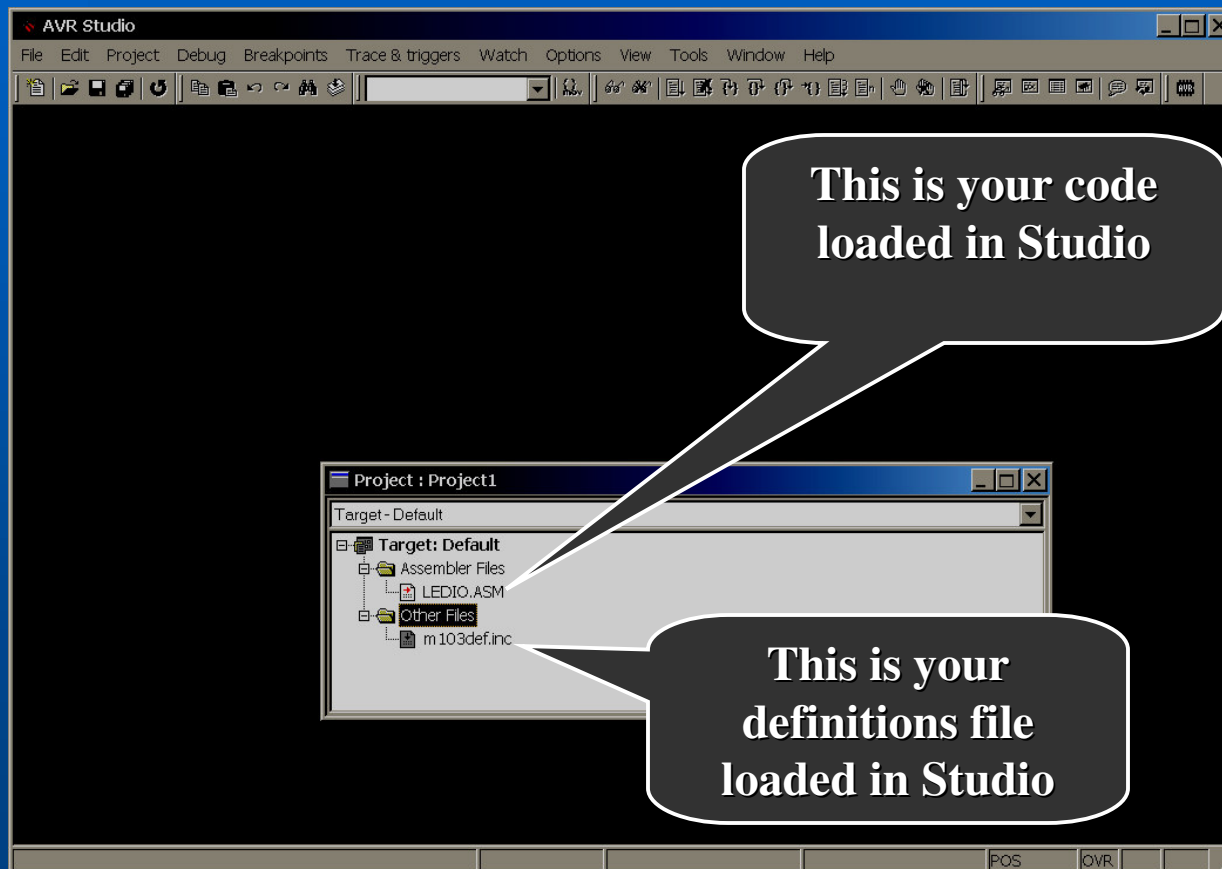
(1) Click here to go to the 'other files' area

(3) Click here to go to the 'other files' area





# Entering files in STUDIO3.52 (III):





# Select the output file format :

The image shows the AVR Studio interface. The 'Project' menu is open, with 'Project Settings...' selected. A callout bubble points to this menu item with the text 'Click on Project Settings'. Below the menu, the 'AVR Assembler Options' dialog box is shown. A callout bubble points to the 'Output file format' dropdown, which is currently set to 'Object format for AVR Studio'. The text 'Change the format of the output file' is written next to this bubble. To the right, another 'AVR Assembler Options' dialog box is shown, with the 'Output file format' dropdown set to 'Intel Intellec 8/MDS (Intel Hex)'. A yellow star is placed next to this dialog box with the text 'Select Intel HEX format'.

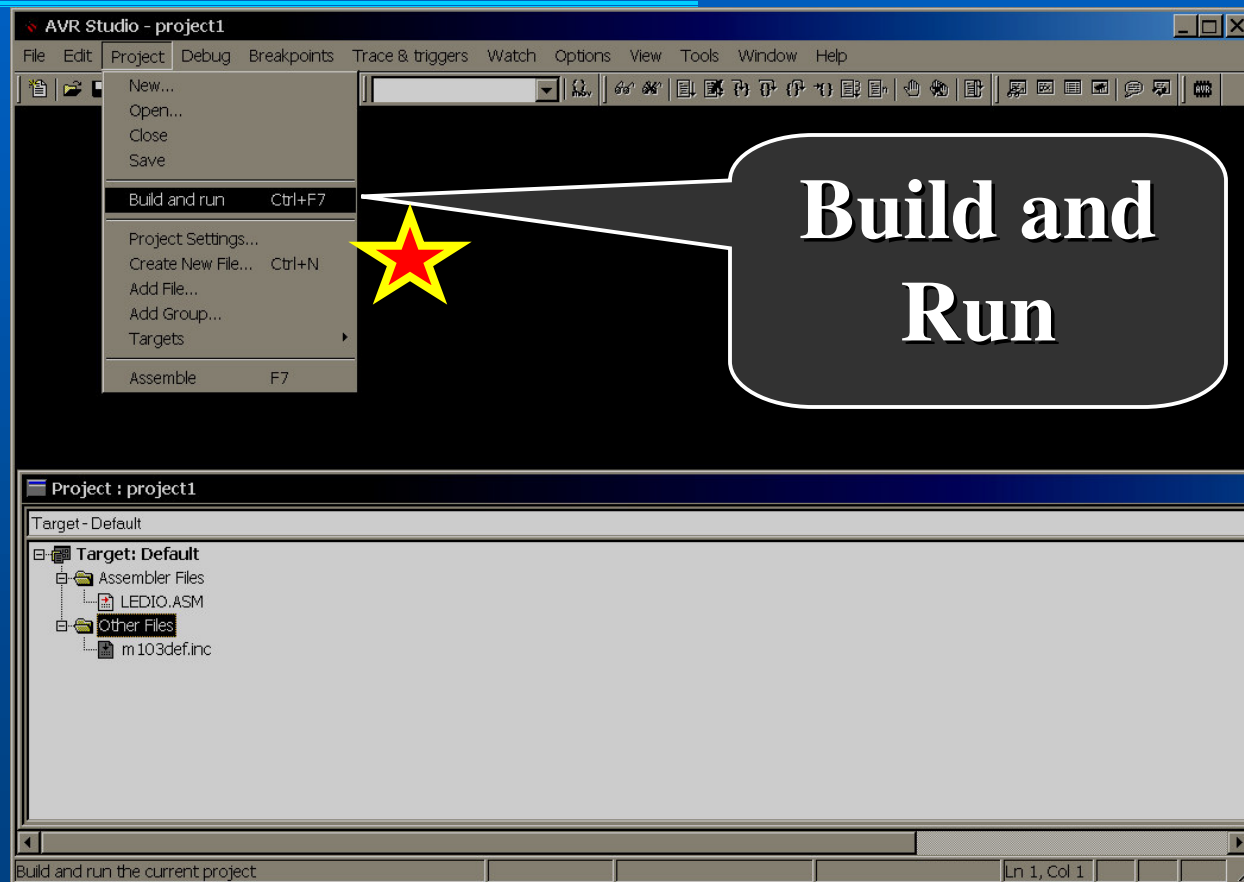
Click on Project Settings

Change the format of the output file

Select Intel HEX format



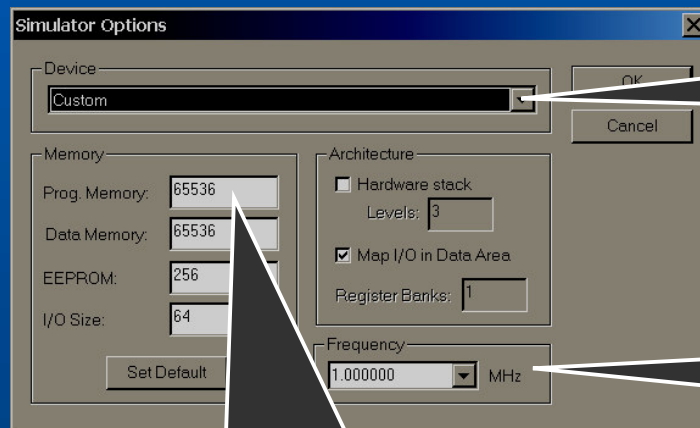
# Running your Program in Studio :





# Processor Settings:

ATmega103,  
Program memory 131072 (128 Kbytes)  
Frequency = 4 MHz



ATmega103

Frequency

Program Flash



# Processor Settings:

**Program  
Flash**

Simulator Options

Device: ATmega103

Memory:

Prog. Memory:	131072
Data Memory:	4096
EEPROM:	4096
I/O Size:	64

Architecture:

☐ Hardware stack  
Levels: 3

☒ Map I/O in Data Area

Register Banks: 1

Frequency: 4.000000 MHz

Set Default

OK  
Cancel

**ATmega103**

**Frequency**









# Watch your Program Running:

AVR Studio - LEDIO

File Edit Project Debug Breakpoints Trace & triggers Watch Options View Tools Window Help

LEDIO

**IO: 1 (Standard)**

Name	Value	Location
Port B		
Port B Data	0x18	0x18
Data Direct...	0x17	0x17
Input Pins	0x1F	0x1F
Port C		
Port D		
Port D Data	0x12	0x12
Data Direct...	0x11	0x11
Input Pins	0x10	0x10
Port E		
Port F		

**InPut OutPut**

**Registers**

Register	Value
R0	0x00
R1	0x00
R2	0x00
R3	0x00
R4	0x00
R5	0x00
R6	0x00
R7	0x00
R8	0x00
R9	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x09
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0x00
R29	0x00

**Processor**

Program Counter: 0x00000012 X-Register: 0x0000

Stack Pointer: 0x00000FFF Y-Register: 0x0000

Cycle Counter: 00000139 Z-Register: 0x0000

Time Elapsed: 34.75 us Frequency: 4.0 MHz

Flags: I T H S V N Z C

StopWatch: Clear 34.75 us

**LEDIO**

```
; inc r16
; lsl r16
IN r16, PIND
OUT PORTB, r16
rjmp Main
```

**Memory:2**

Data: 0x0060

**Memory:1**

Program Mem: 0x000000

**Project Output**

Including 'm103def.inc'

Program memory usage:

Category	Words
Code	21
Constants (dw/db)	0
Unused	0
Total	21

Assembly complete with no errors.

**Project : Project1**

Target: Default

Target: Default

- Assembler Files
  - LEDIO.ASM
- Other Files
  - m103def.inc

2/2/2004

Rm: 508, x47590



# Exercising the ATmega103 commands:

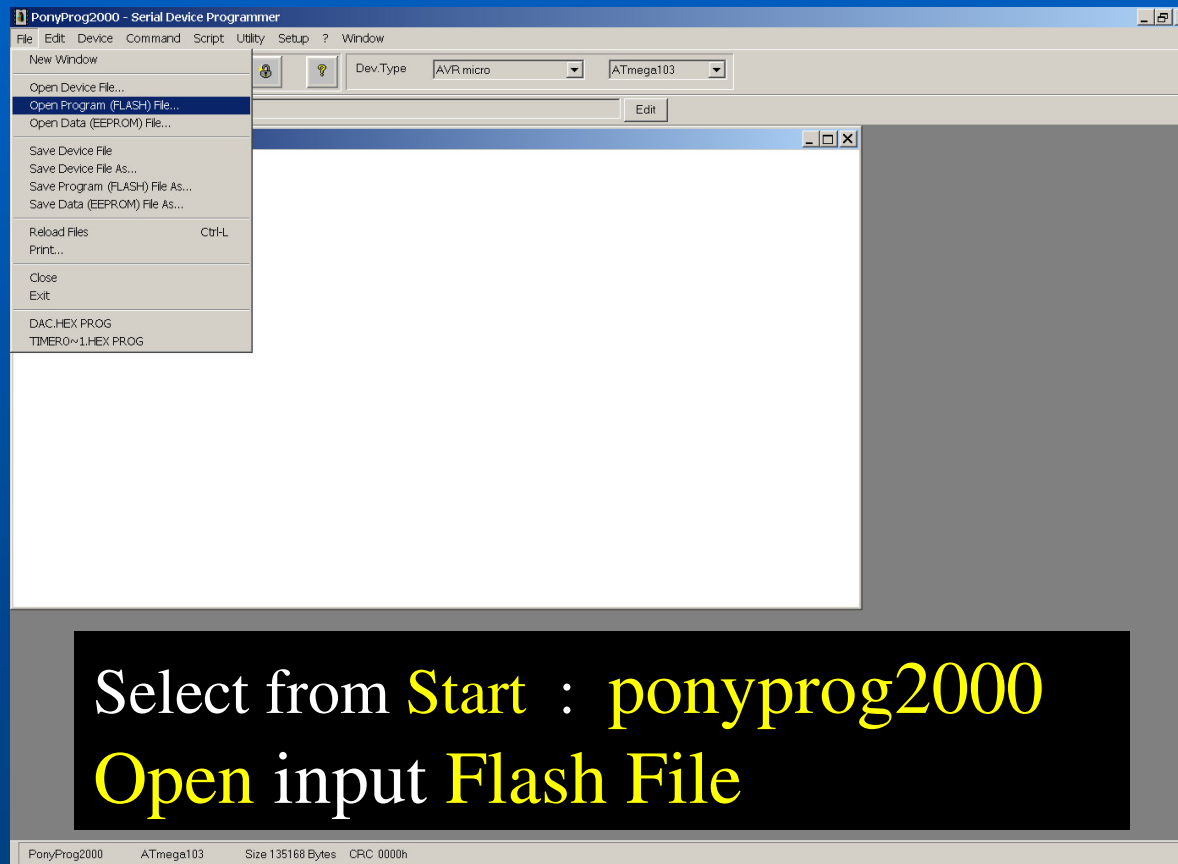
**Look at the processor assembly commands in the Web page and try writing other programs.**

**Try the commands SER, CLR, MOV, ADD, INC, LSL,... and see what do they do.....**

**Write some programs that use these commands and Output the results on PORTB so you can see them on Using the POTRB LEDs when you eventually down load for real to the ATmega103.**

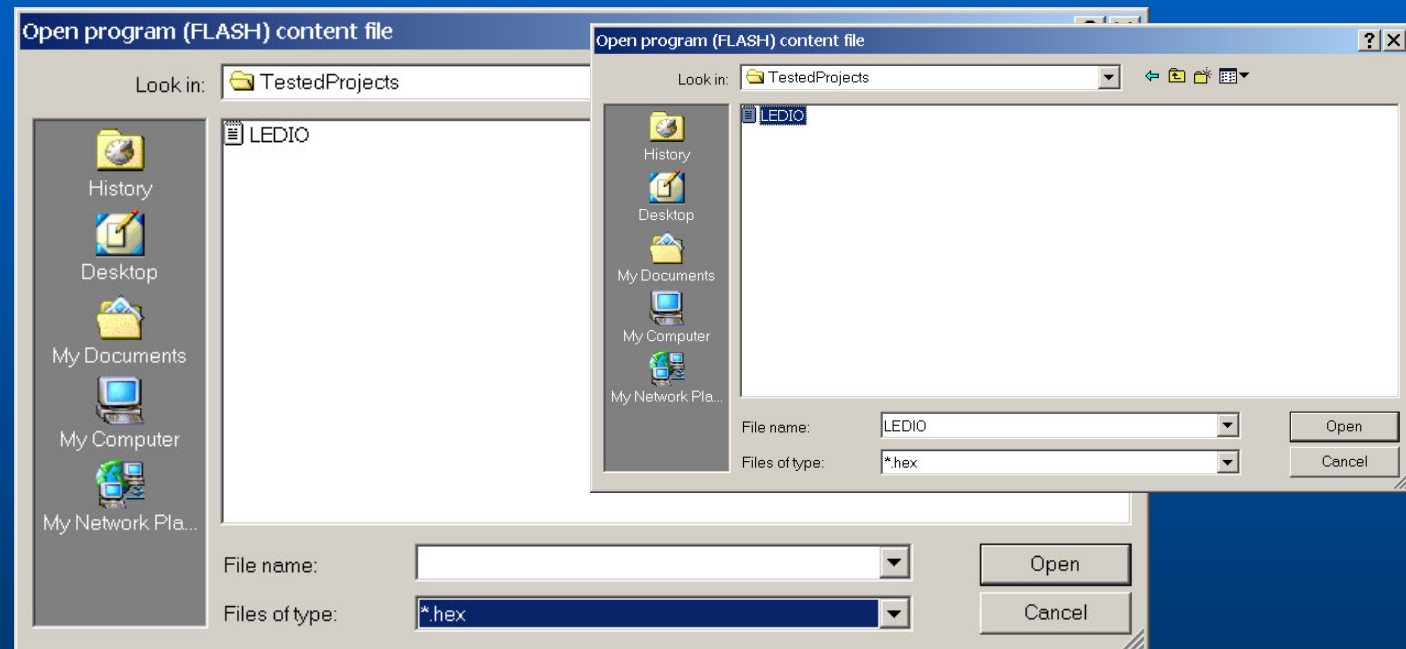


# Downloading with Ponyprog I:





# Downloading with Ponyprog II:



Select from **\*.hex** and **LEDIO**



# Downloading with Ponyprog III:

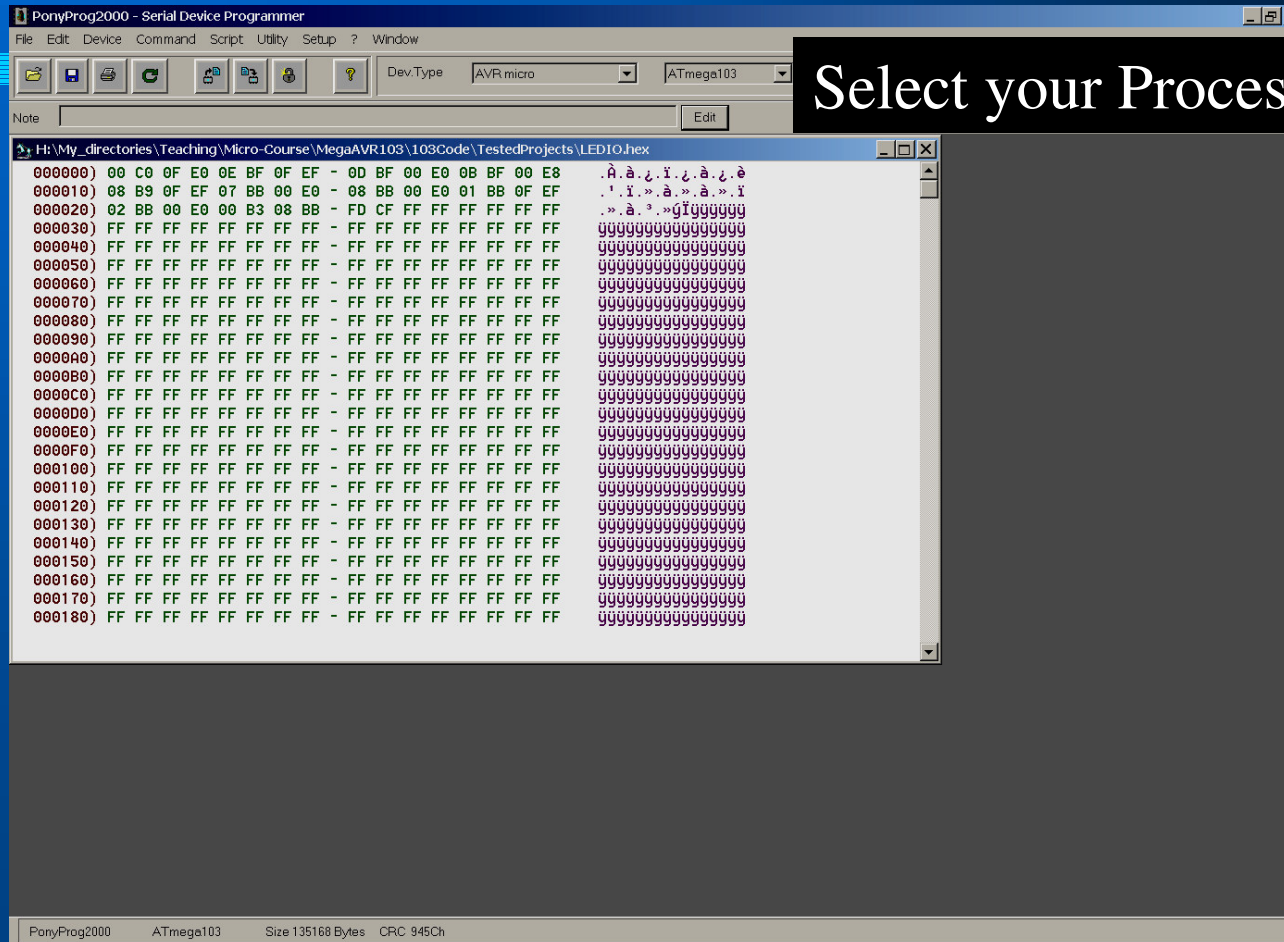
The screenshot shows the PonyProg2000 - Serial Device Programmer window. The 'Dev Type' is set to 'AVR micro' and the device is 'ATmega103'. The file path is 'H:\My\_directories\Teaching\Micro-Course\MegaAVR103\103Code\TestedProjects\LED10.hex'. The main window displays the hex code in a table format, with the first column showing the address (000000 to 000180) and the second column showing the hex data. The data is mostly 'FF' (unprogrammed) and '00' (programmed). The status bar at the bottom shows 'PonyProg2000 ATmega103 Size 135168 Bytes CRC 945Ch'.

Address	Hex Data
000000	00 C0 0F E0 0E BF 0F EF - 0D BF 00 E0 0B BF 00 E8
000010	08 B9 0F EF 07 BB 00 E0 - 08 BB 00 E0 01 BB 0F EF
000020	02 BB 00 E0 00 B3 08 BB - FD CF FF FF FF FF FF FF
000030	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000040	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000050	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000060	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000070	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000080	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000090	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
0000A0	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
0000B0	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
0000C0	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
0000D0	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
0000E0	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
0000F0	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000100	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000110	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000120	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000130	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000140	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000150	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000160	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000170	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
000180	FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF

This is your machine code to be downloaded to the ATmega103 chip



# Downloading with Ponyprog IV:

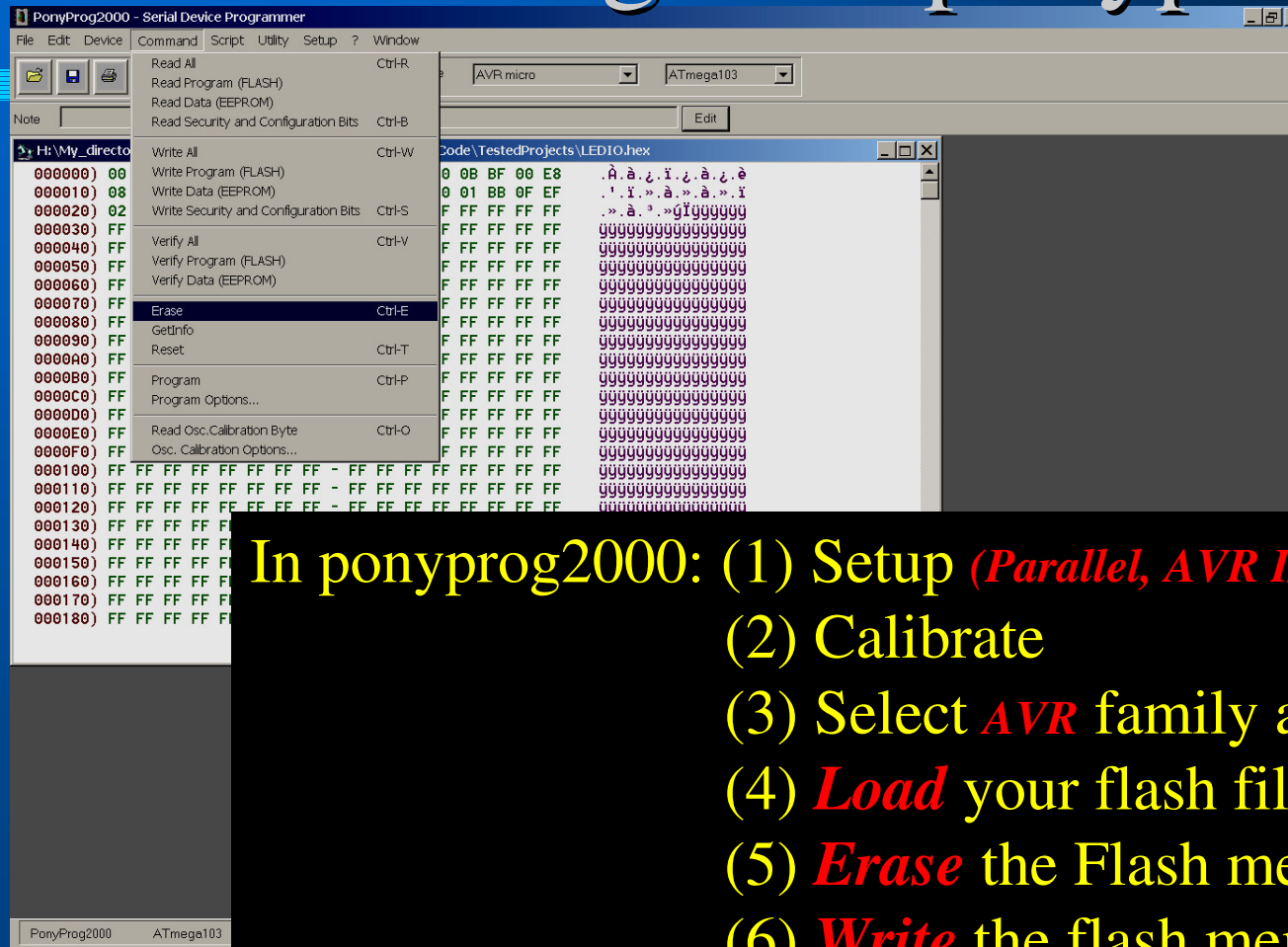


Select your Processor again





# Downloading with ponyprog V:



- In ponyprog2000:
- (1) Setup (*Parallel, AVR I/O, LP1*)
  - (2) Calibrate
  - (3) Select *AVR* family and *ATmega103*
  - (4) *Load* your flash file *LEDIO.hex*
  - (5) *Erase* the Flash memory
  - (6) *Write* the flash memory





# Programs to write I:

- (1) Download from the Web Page the program LEDIO.asm, assemble it and run it. What do you see if you press the buttons on PORTD ?? Do you know why this happens (dark LED  $\rightarrow$  1) ???
- (2) Write a program that adds 2 numbers and outputs the result at PORTB. Read the result using the LEDs.



# Programs to write II:

- (3) Make a counter from 0 – FF and look with your scope probe at the LSB. How long does it take to make an addition ?? Why does the D0 bit toggle with a frequency that is twice that of D1 ?  
*(use two scope probes one on D0 and another on D1)*
- (4) In the documentation you will find how many clock counts are required to perform an instruction in your program. Given that the ATmega103 has a 4 MHz clock you can predict the time it takes to do an addition. Does it agree with what you measure using the scope probes ?