ATmega103 Assembly I

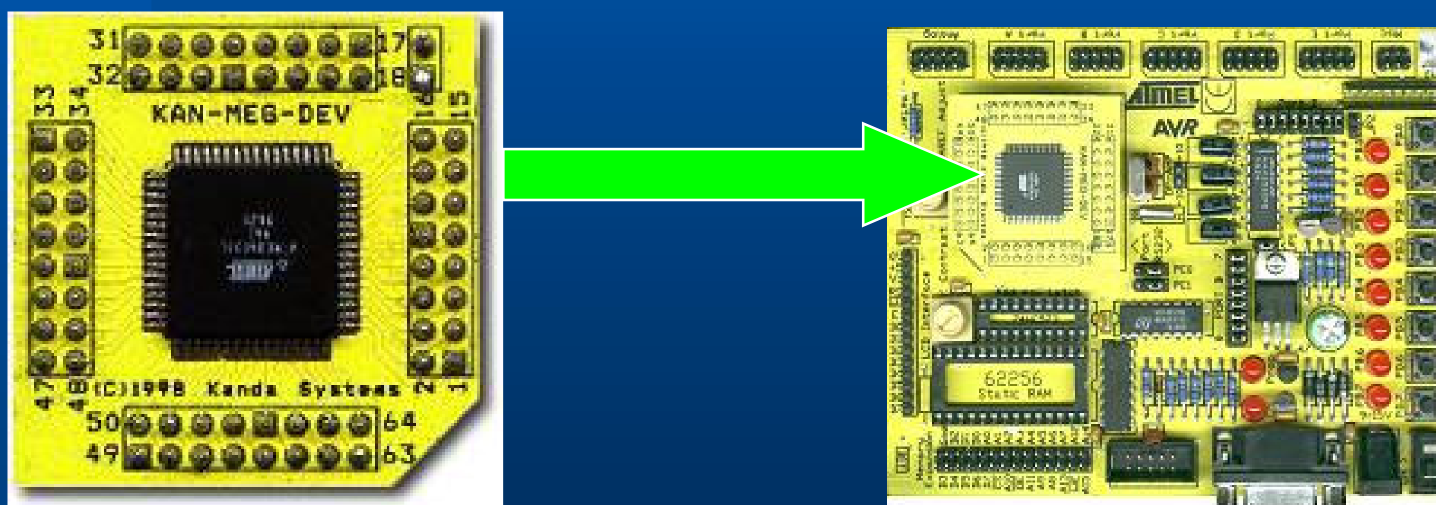
Costas Foudas, Imperial College,
Rm: 508, x47590

Outline:

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

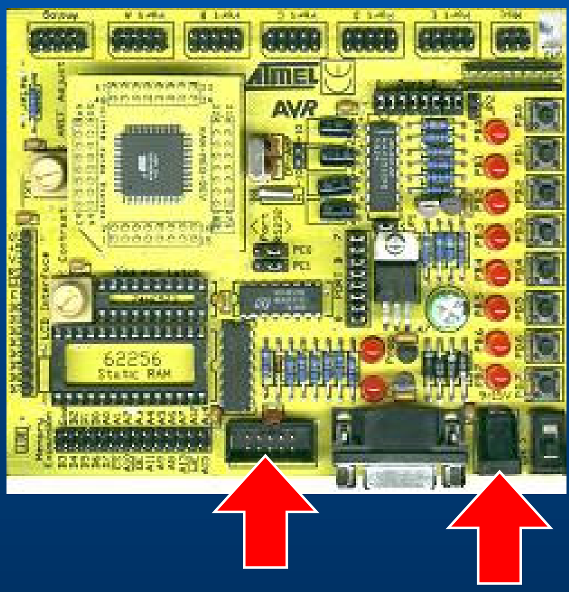
The ATmega103 Microprocessor

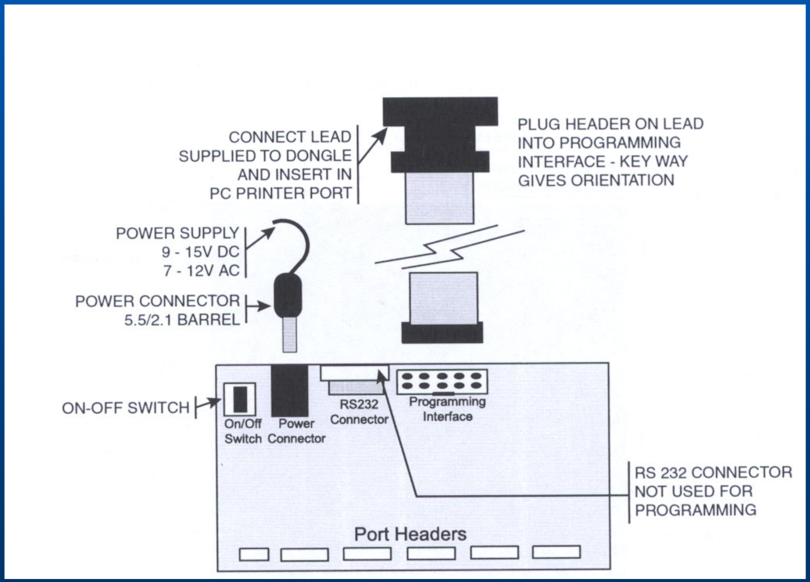
- *In this course you will be using the ATmega103 processor mounted on an ATMEL programming board*



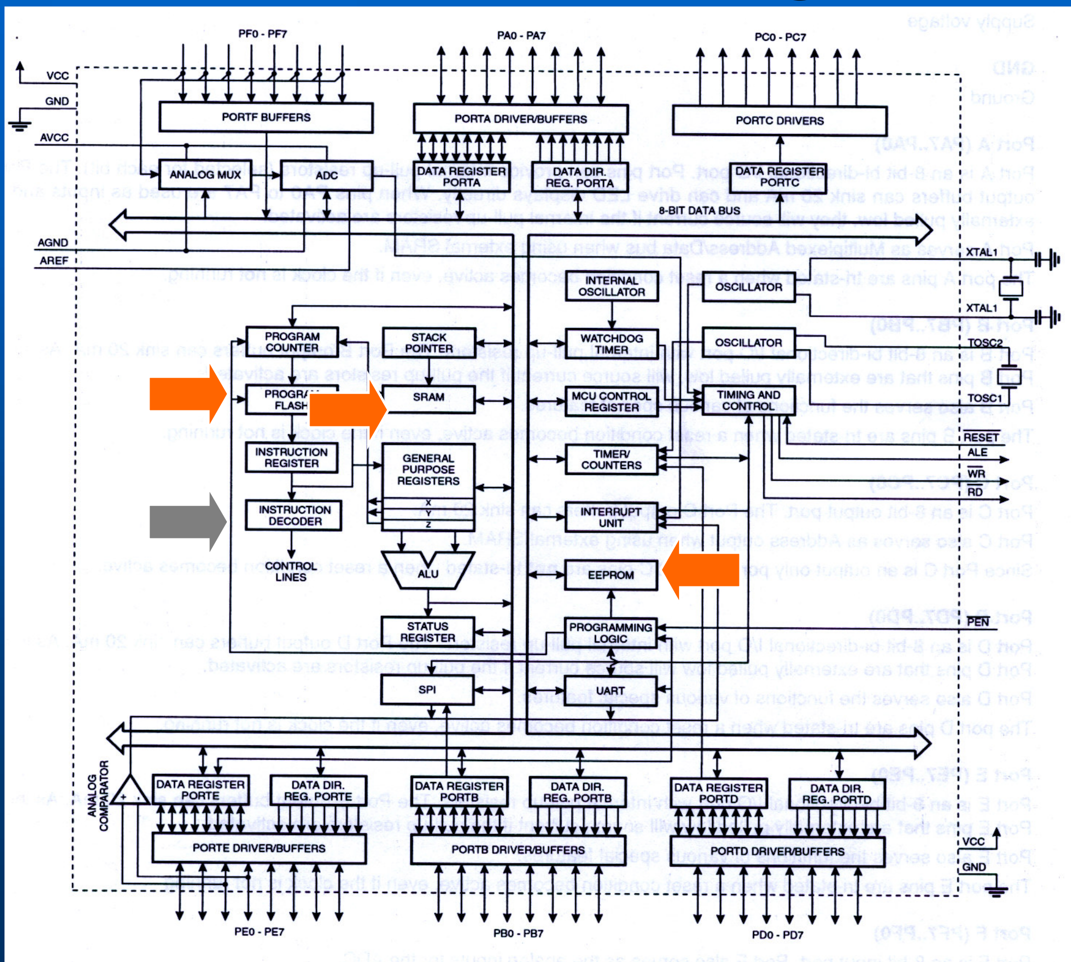
The ATMEL BOARD

- *Connecting the board with your PC*





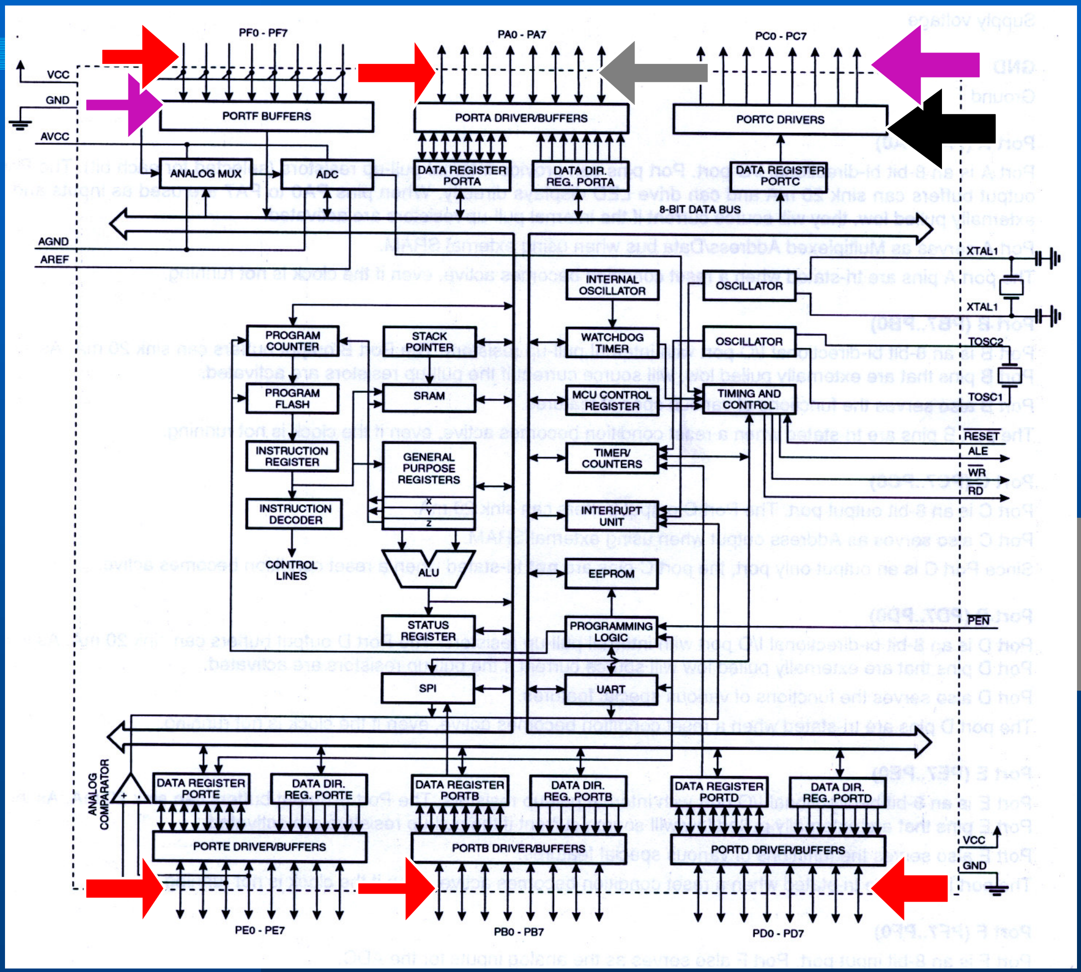
ATmega103 diagram



- 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



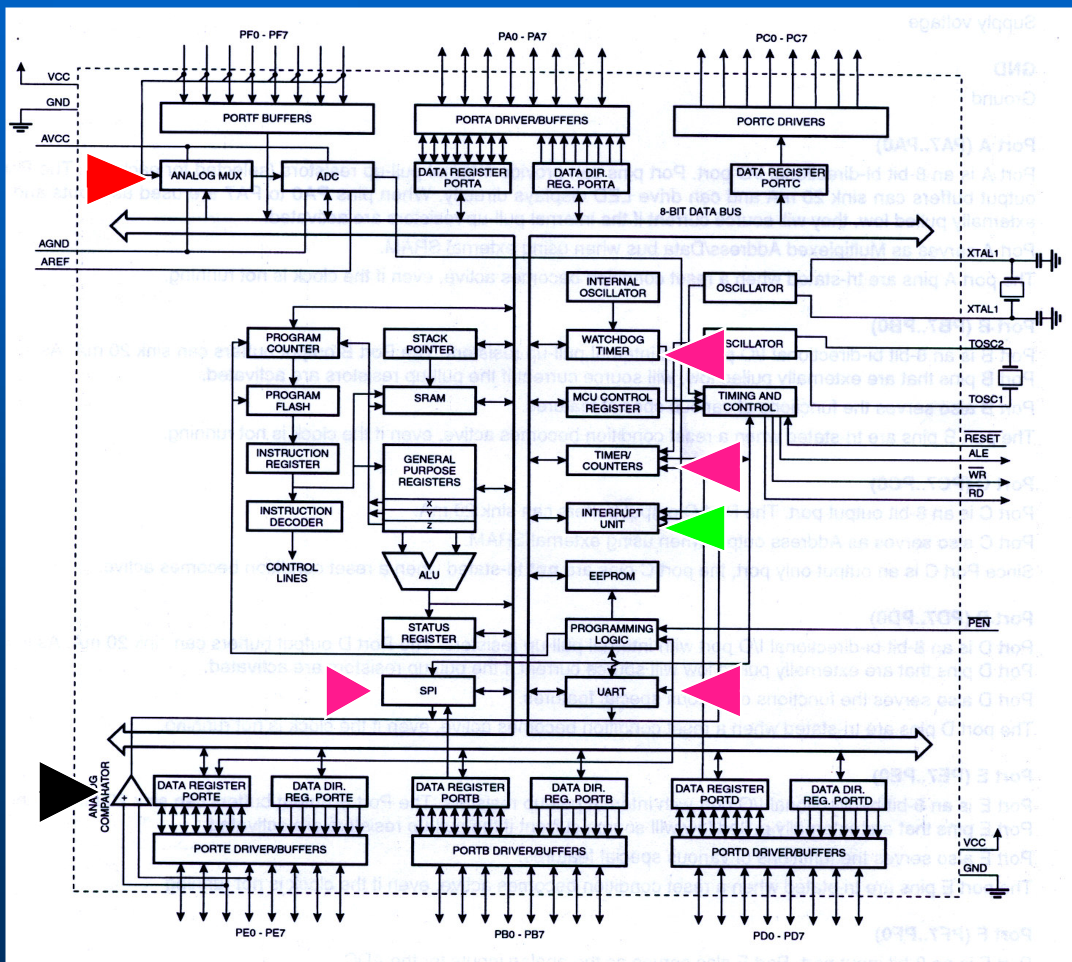
- 6 PORTS;
- 4 Bi-Directional

- 1 Output
- 1 Input

- Port A 8-bit I/O SRAM ADDR/DATA

- Port C SRAM
- ADDRESS

ATmega103 Peripherals II



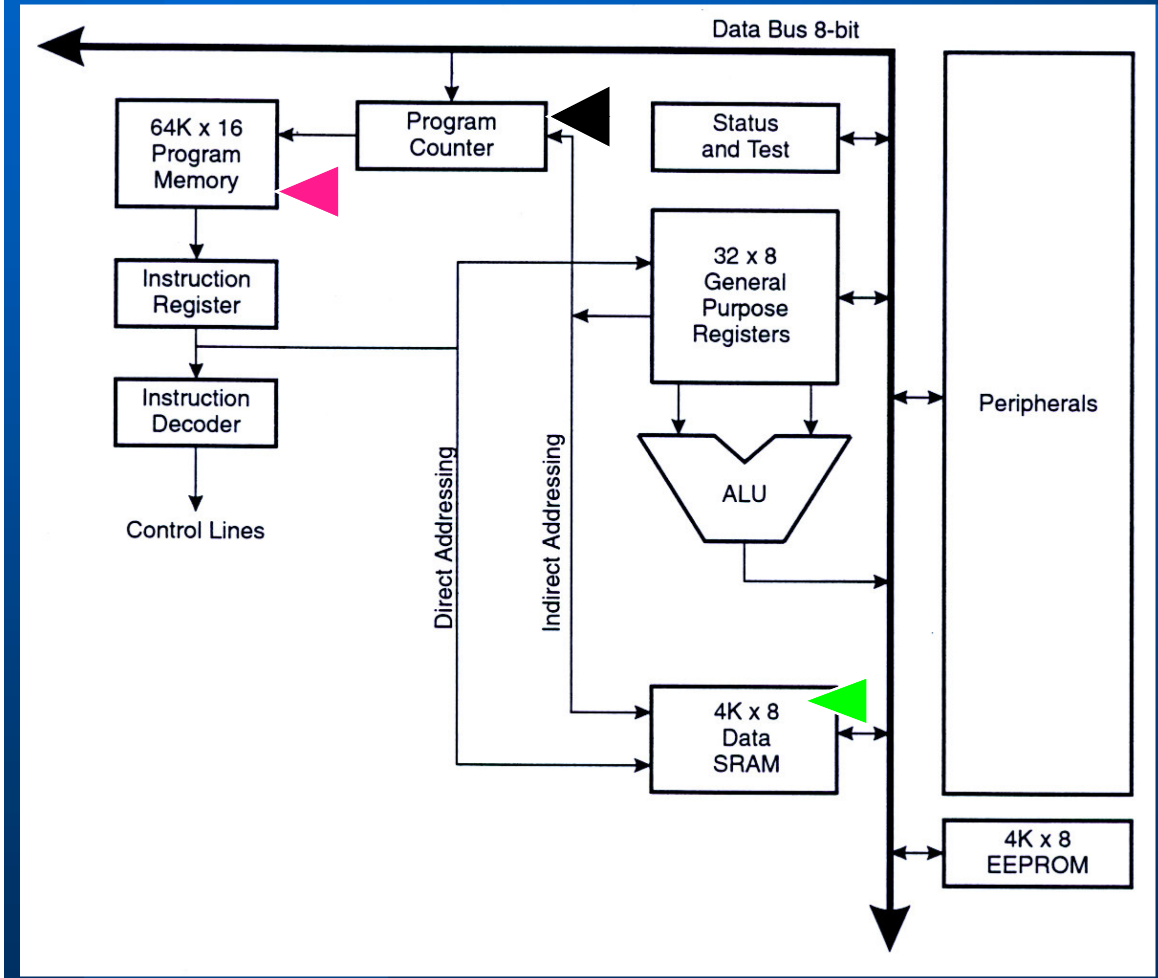
Interrupt Unit

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

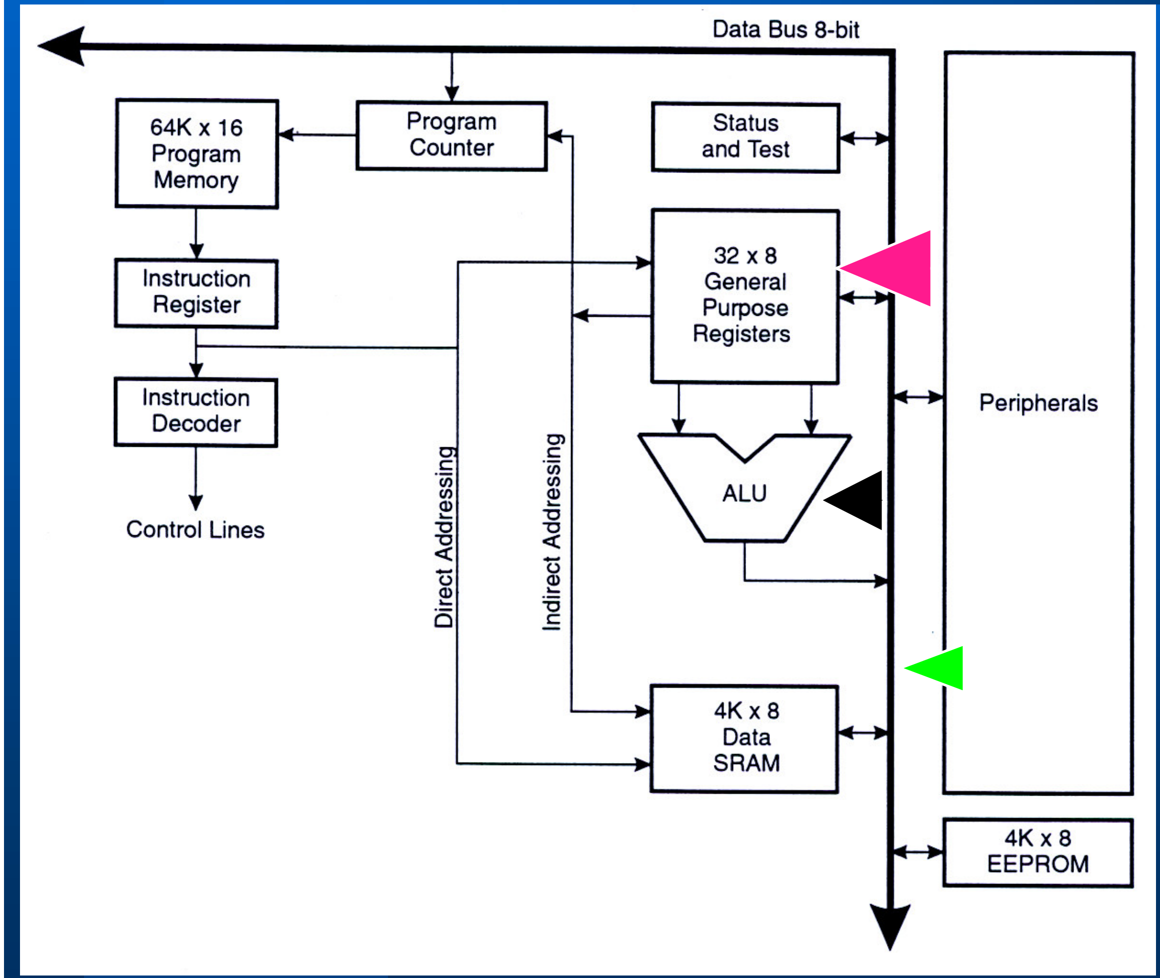


- Your Program resides in this 128 Kbytes memory in .hex Intel format (hex numbers)

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

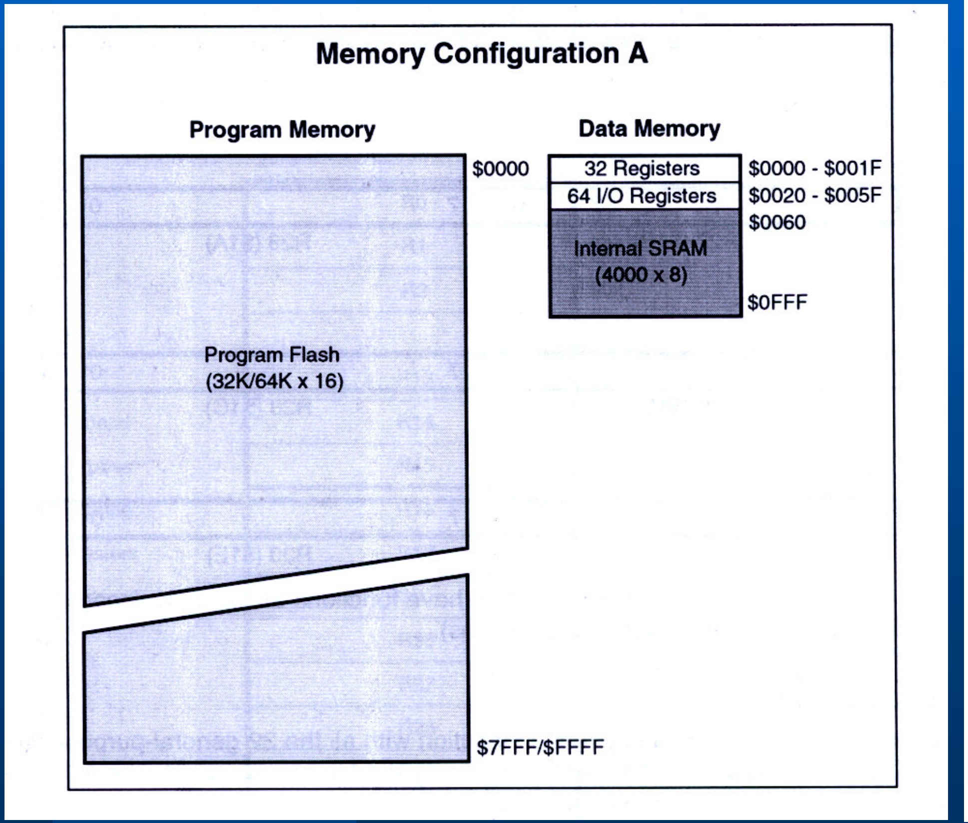


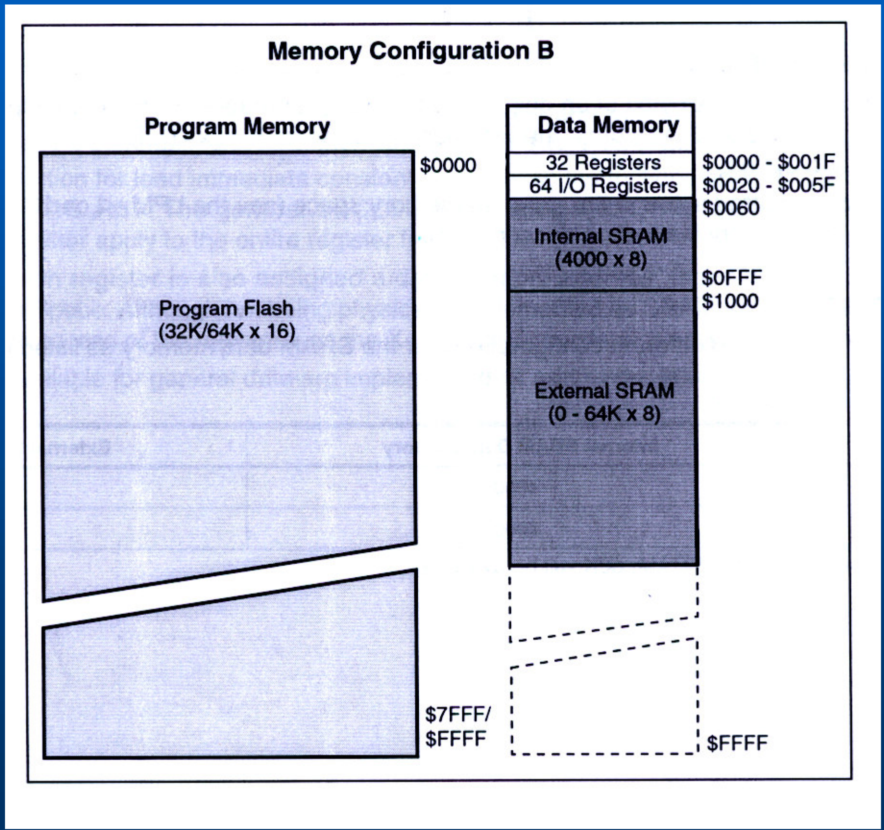
32 General Purpose Registers

The Arithmetic Logic Unit operates on the 32 registers

8-bit Data Bus

The ATmega103 Memory Map





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

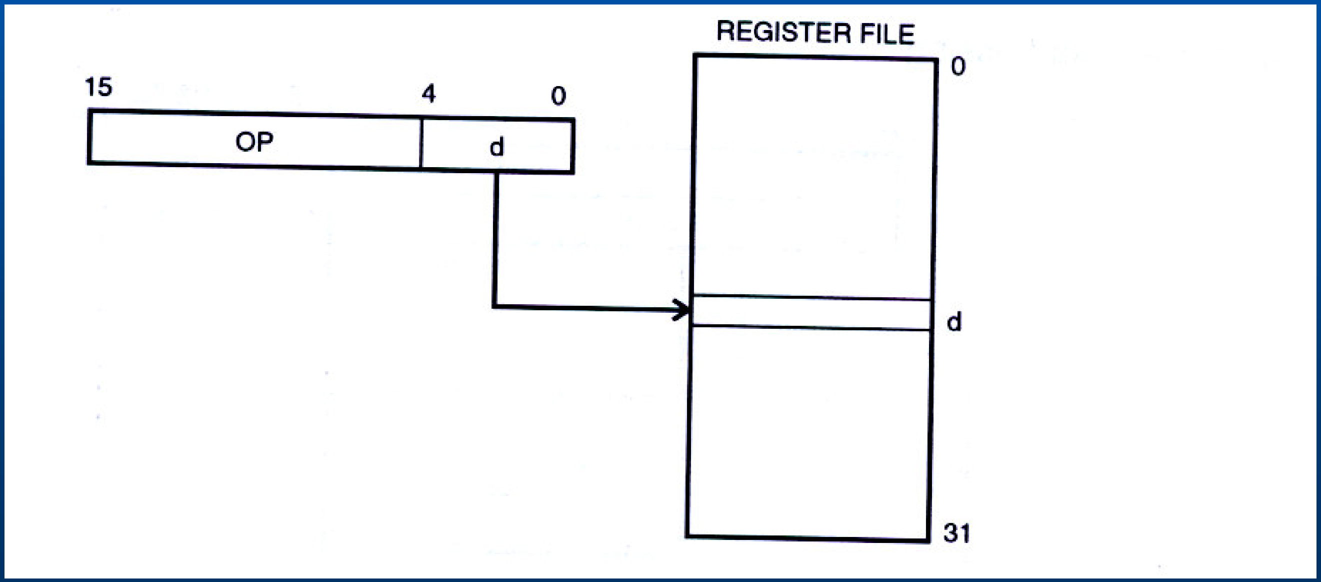
- **You can set them to ones by:**

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:

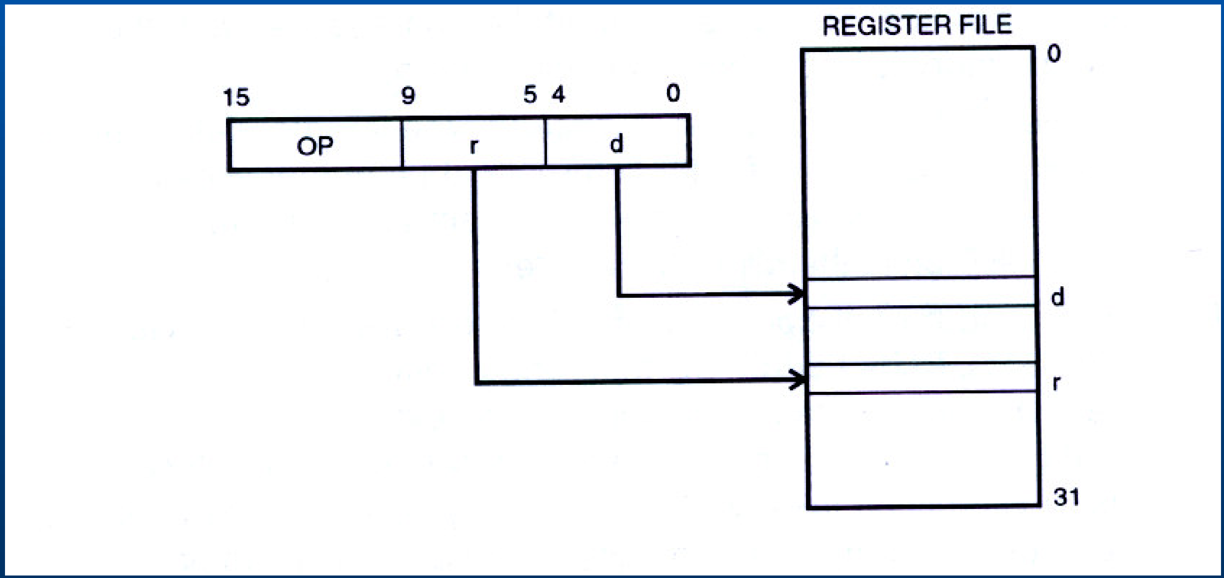
```
mov  RegisterTwo,   InPutRegister
```

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

Direct two register

MOV Rd, Rx

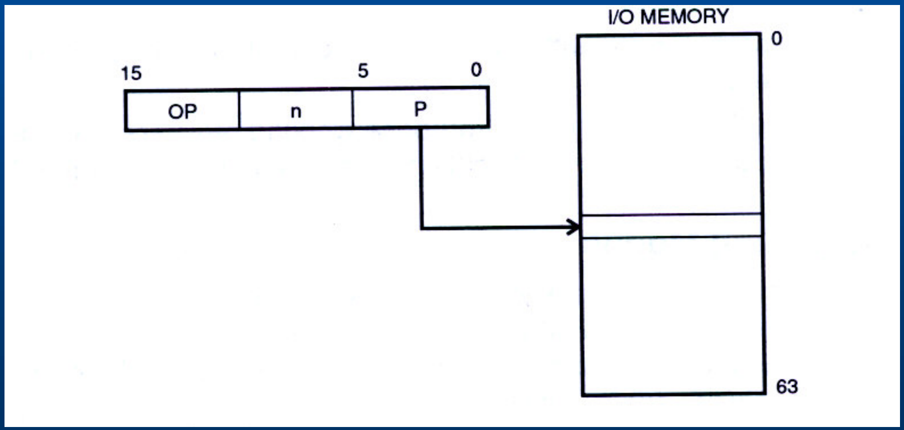
Note: that the direction goes Against intuition:



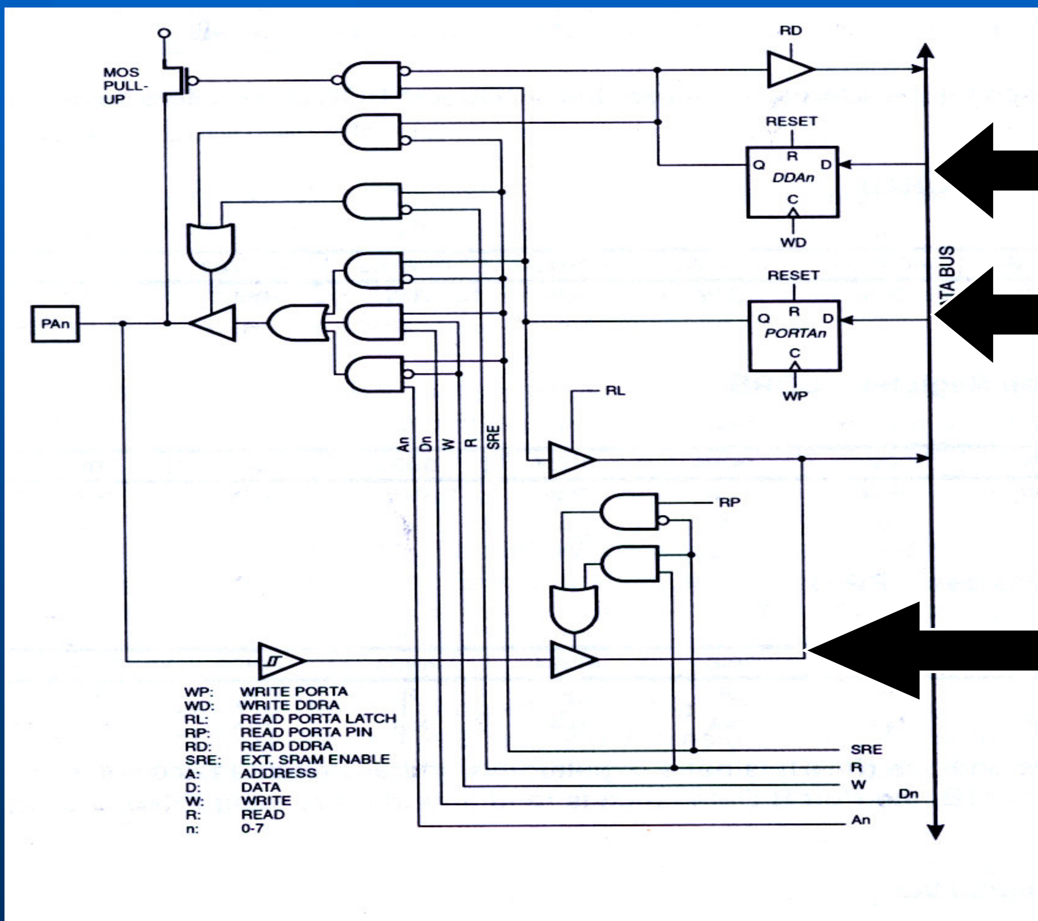
I/O Direct Addressing

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

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



Port Example:



DDRA : Direction Reg.

PORTA : OUTPUT Reg.

PINA : INPUT (no Reg.)

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:

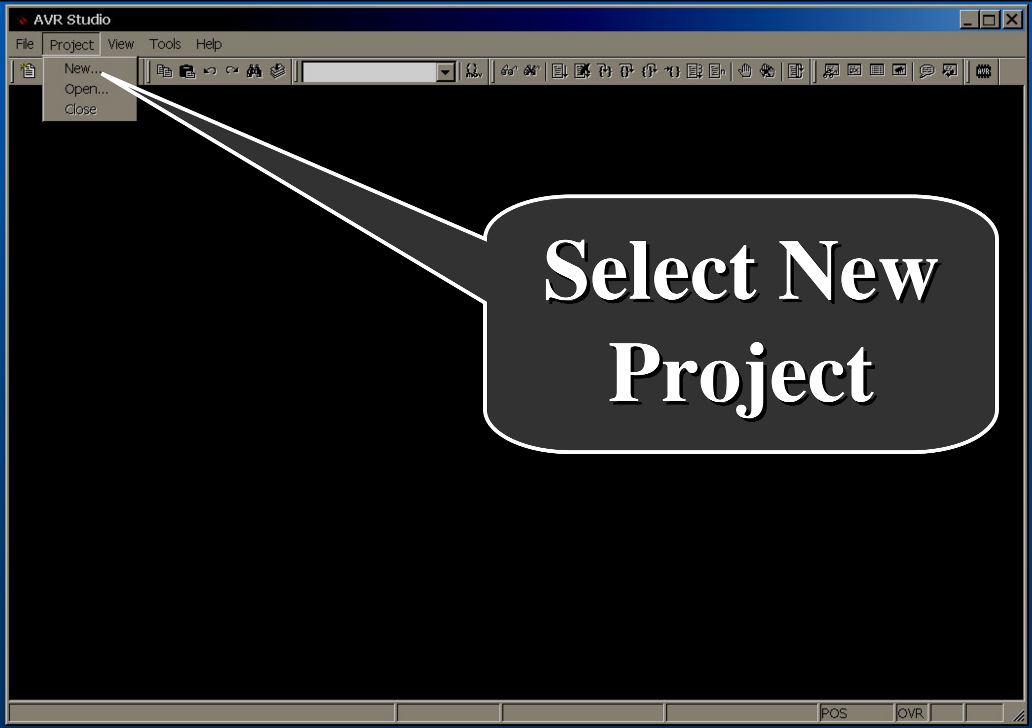
Open a directory where you will store you code:

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

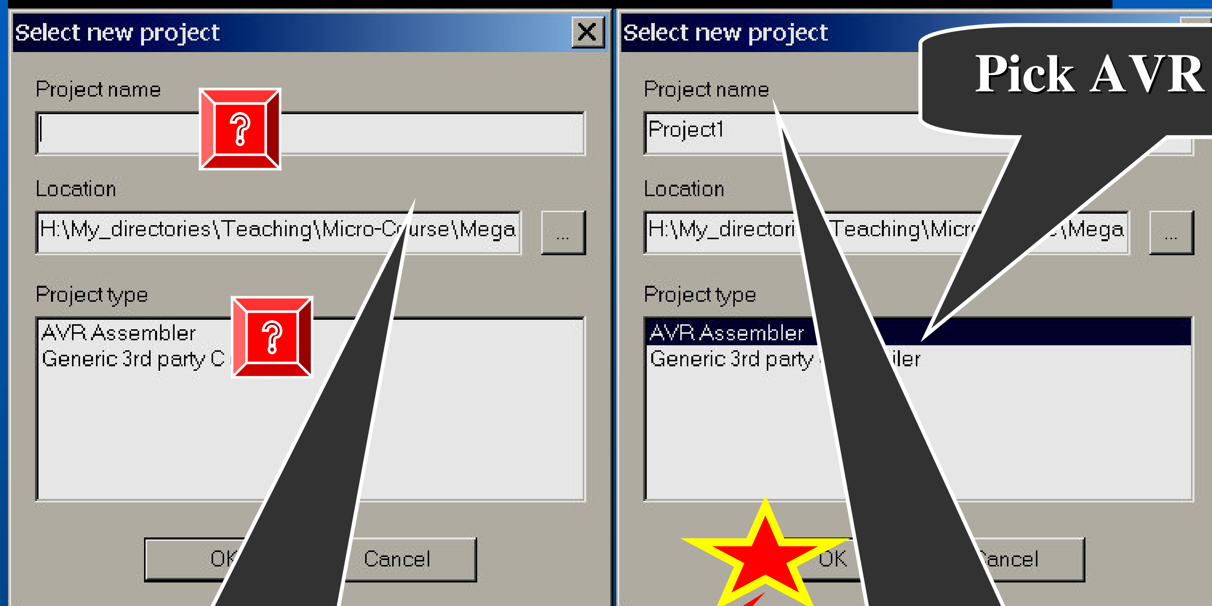
Getting Started with STUDIO 3.52 :

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



Getting Started with STUDIO 3.52 :

You should be getting now the window:



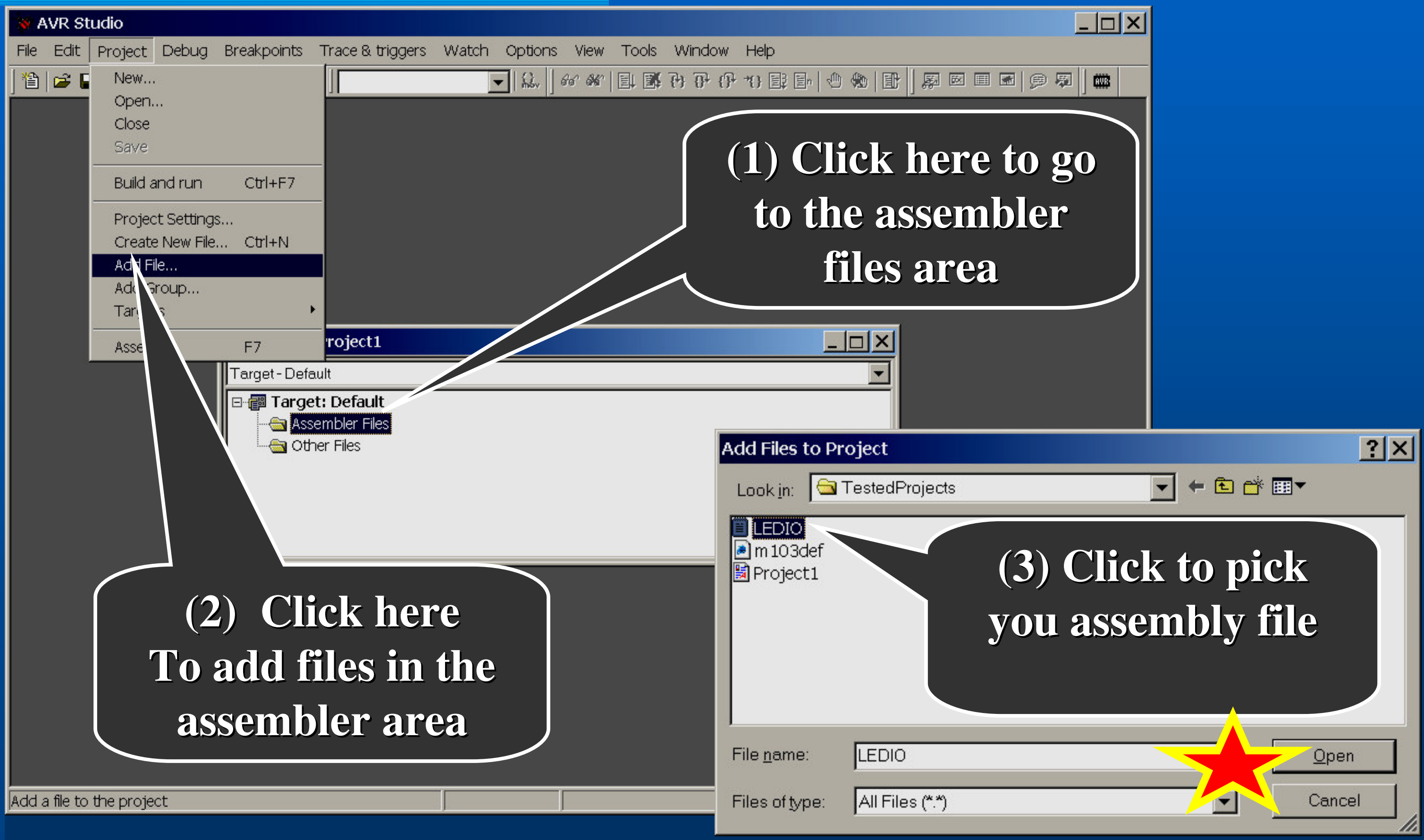
Pick AVR Assembler

This is your code directory

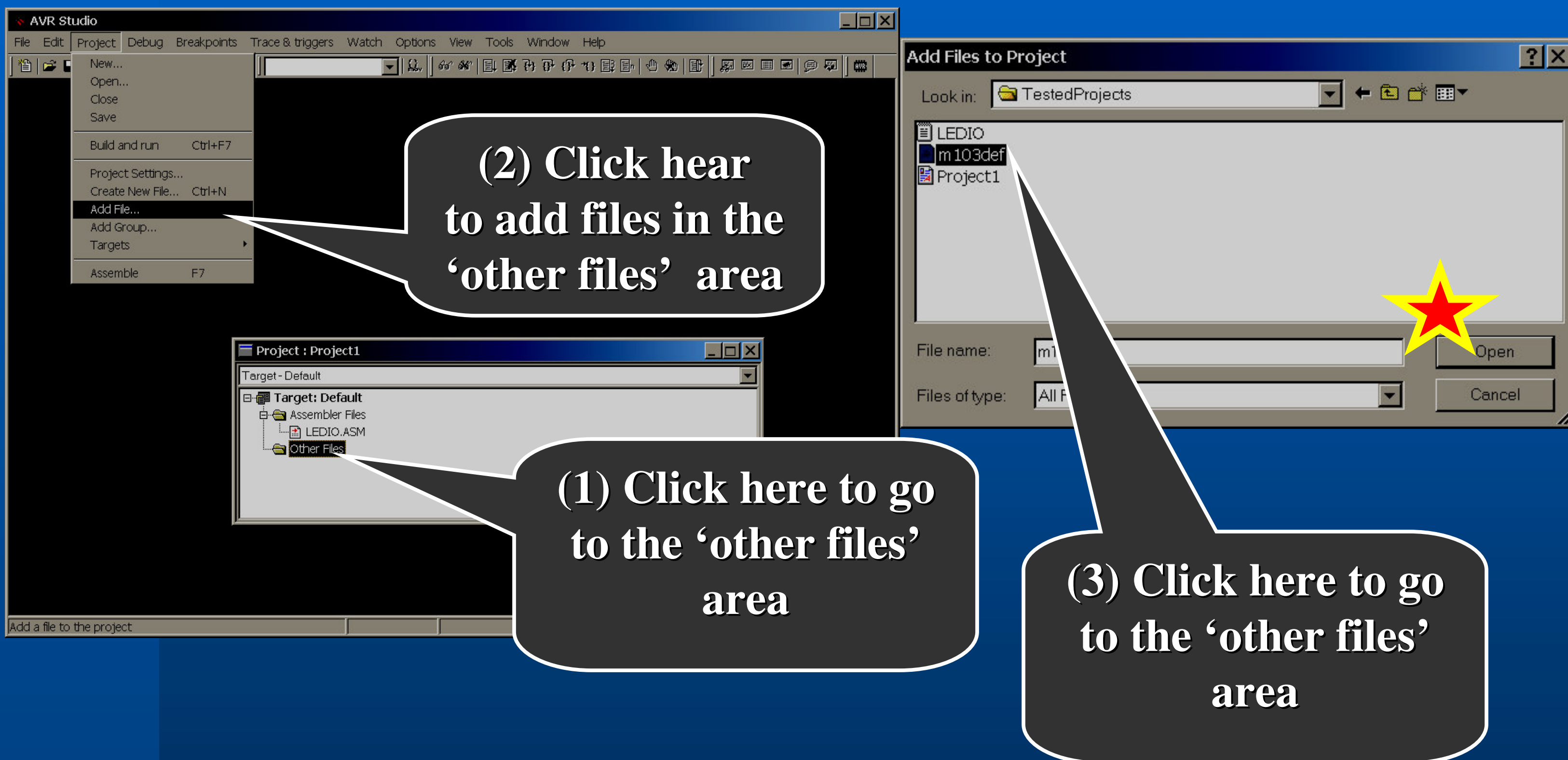
At the end

Pick a name for your project

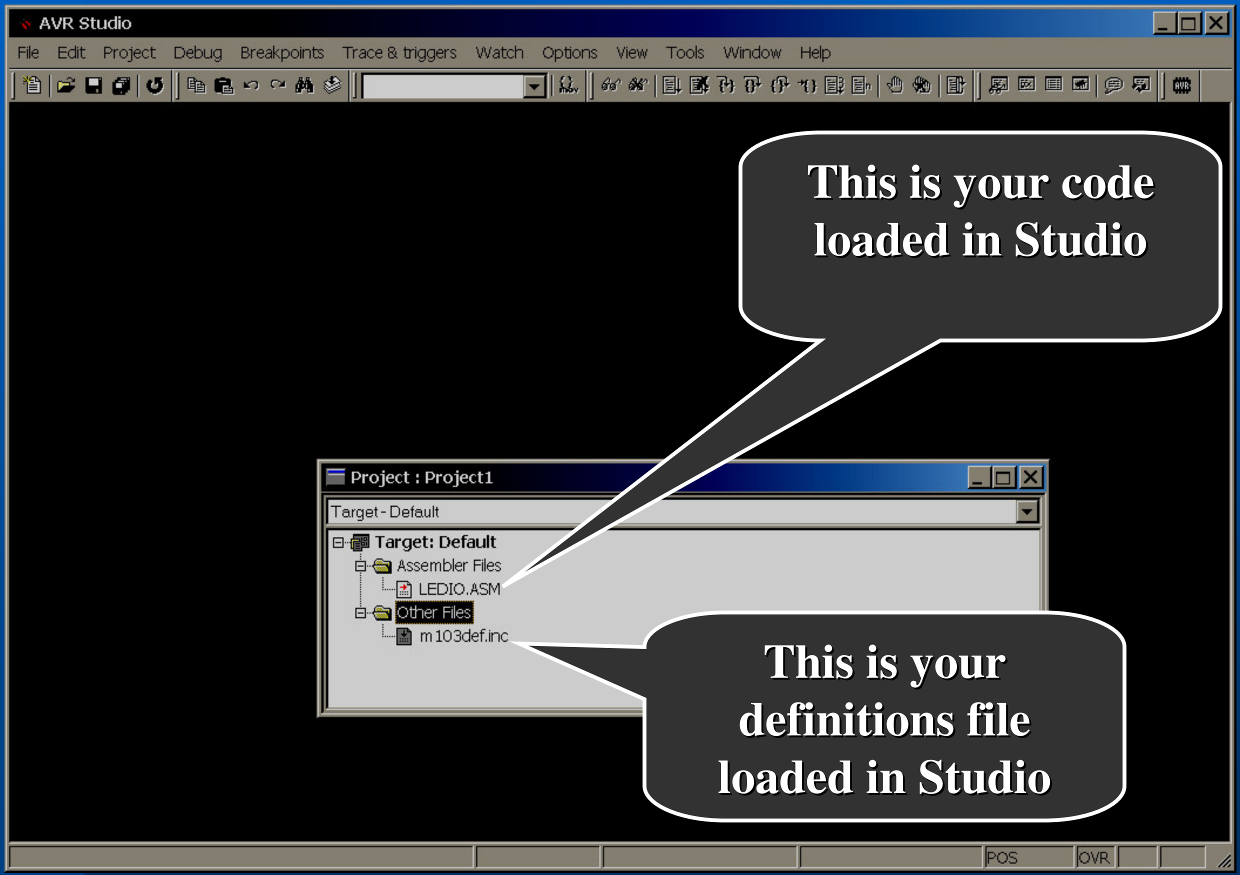
Entering files in STUDIO3.52 (I):



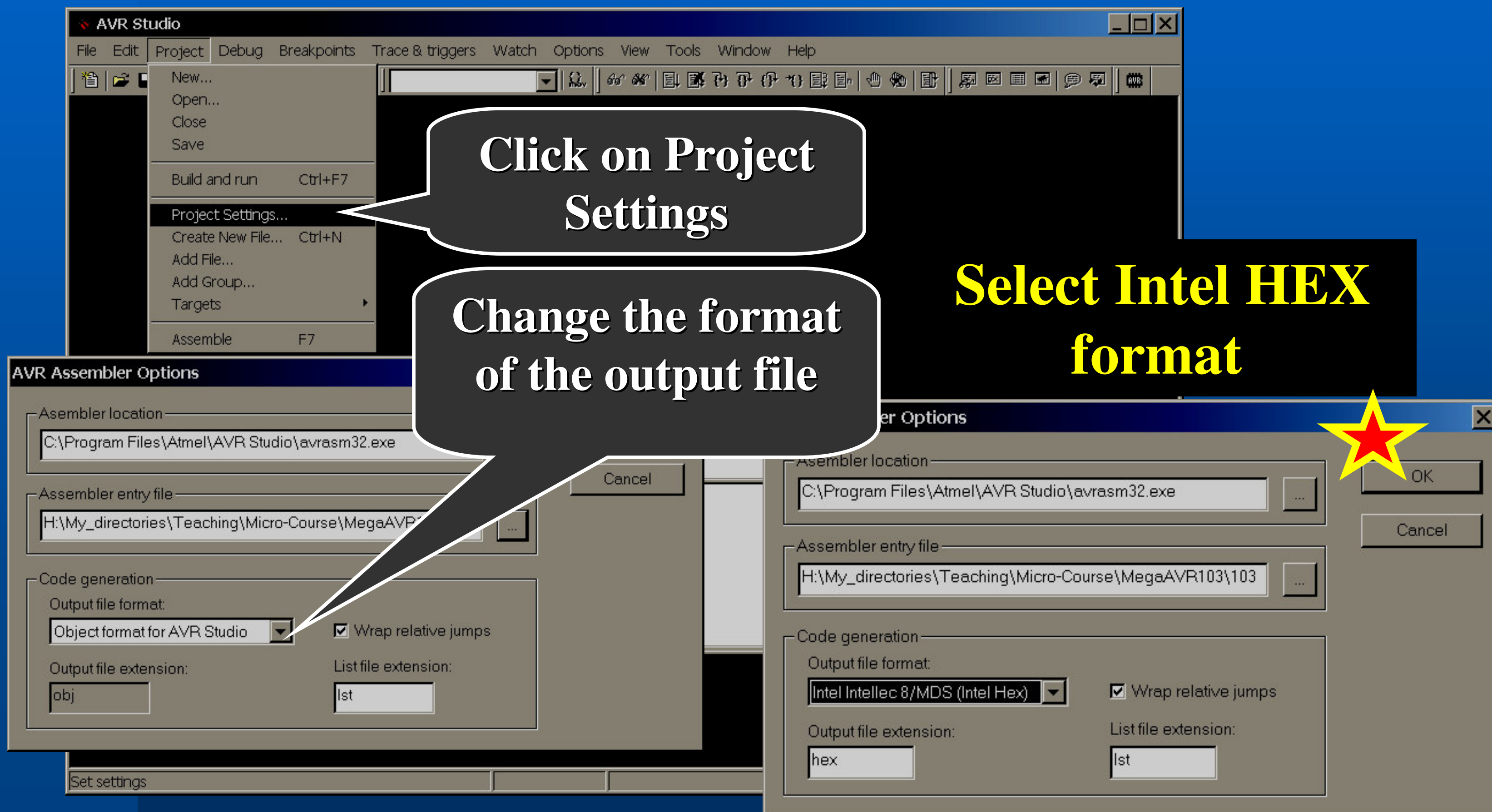
Entering files in STUDIO3.52 (II):



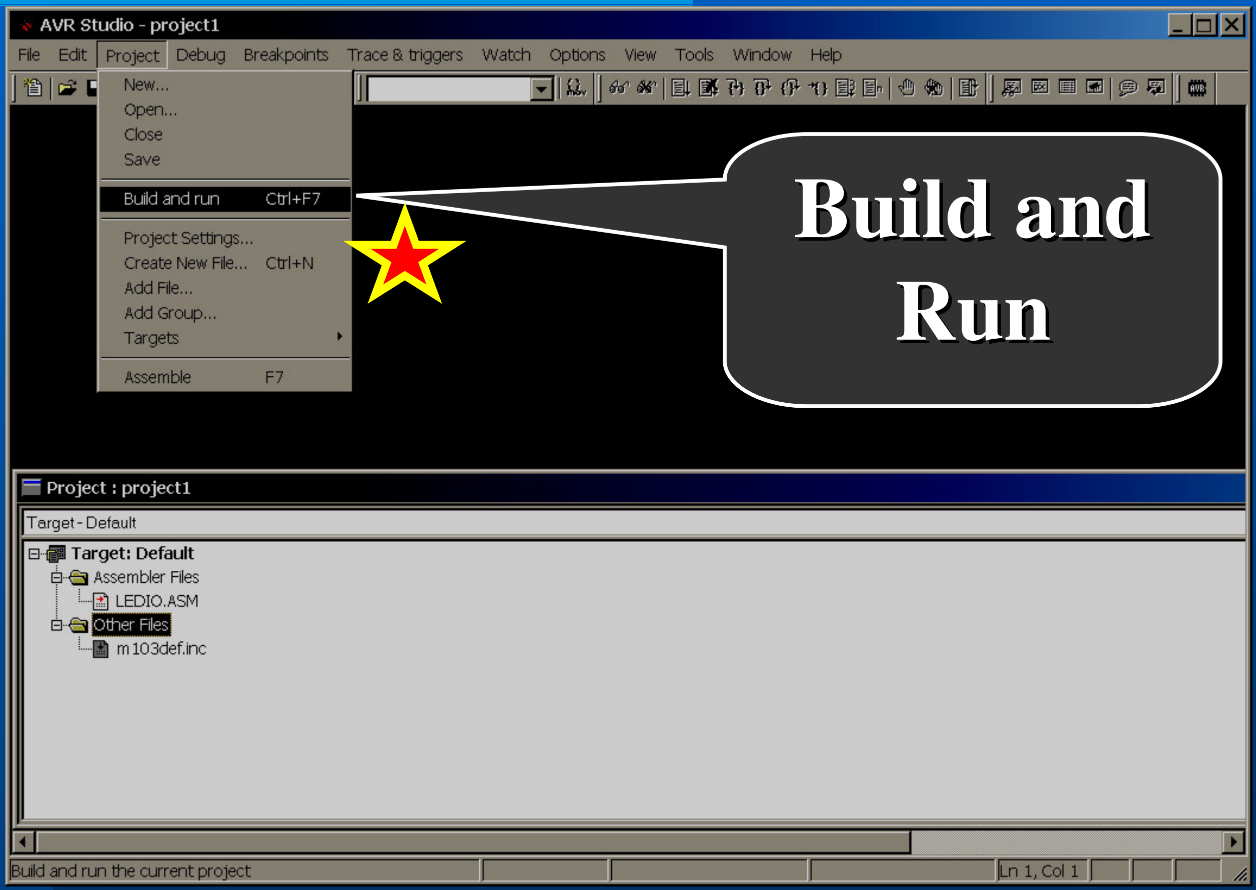
Entering files in STUDIO3.52 (III):



Select the output file format :

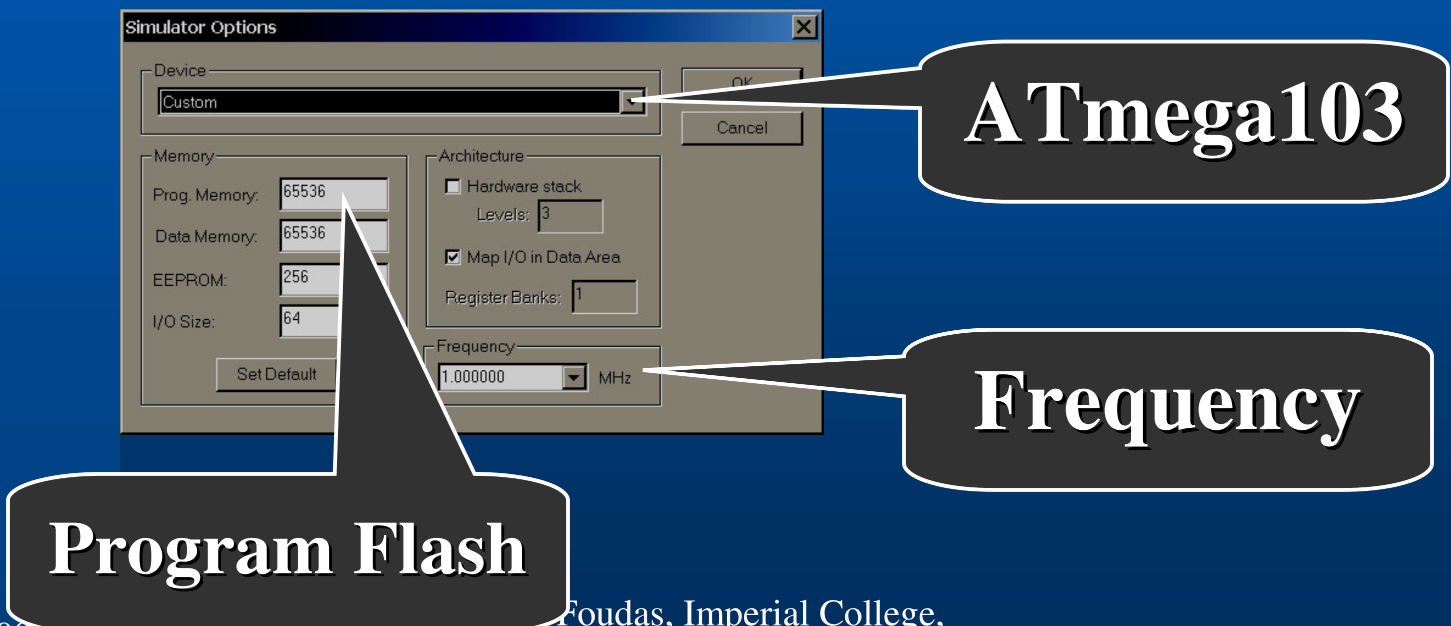


Running your Program in Studio :

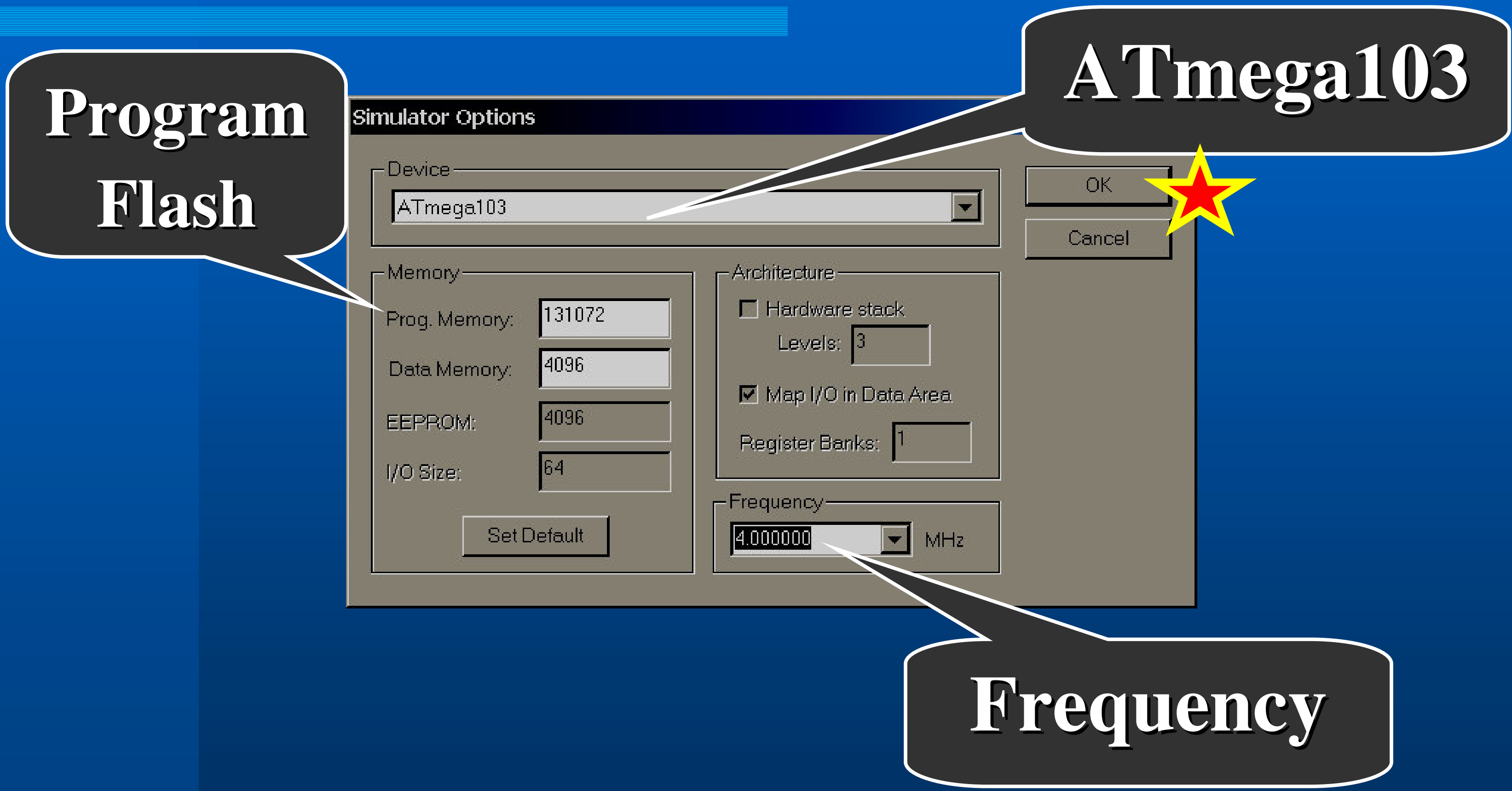


Processor Settings:

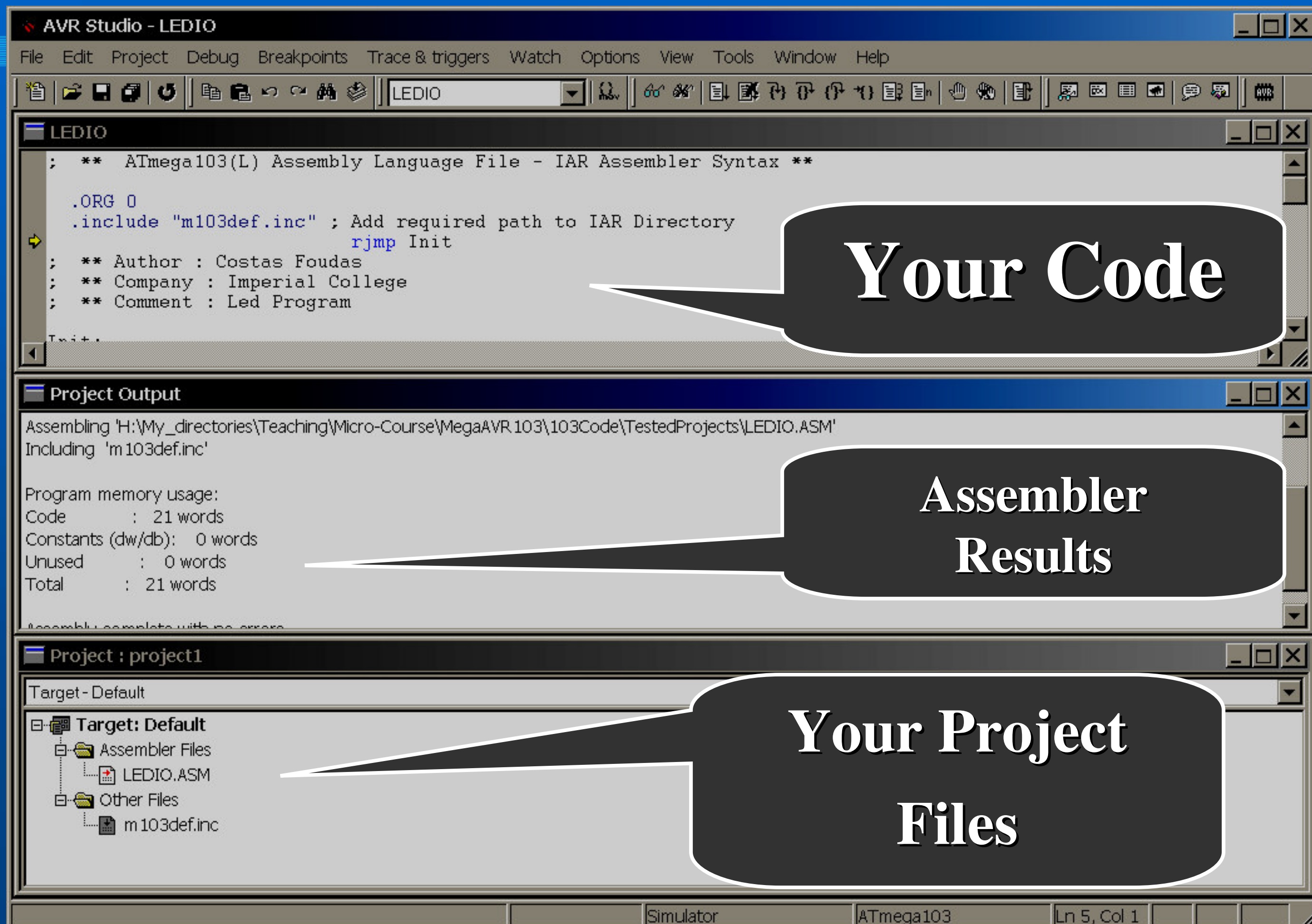
ATmega103,
Program memory 131072 (128 Kbytes)
Frequency = 4 MHz



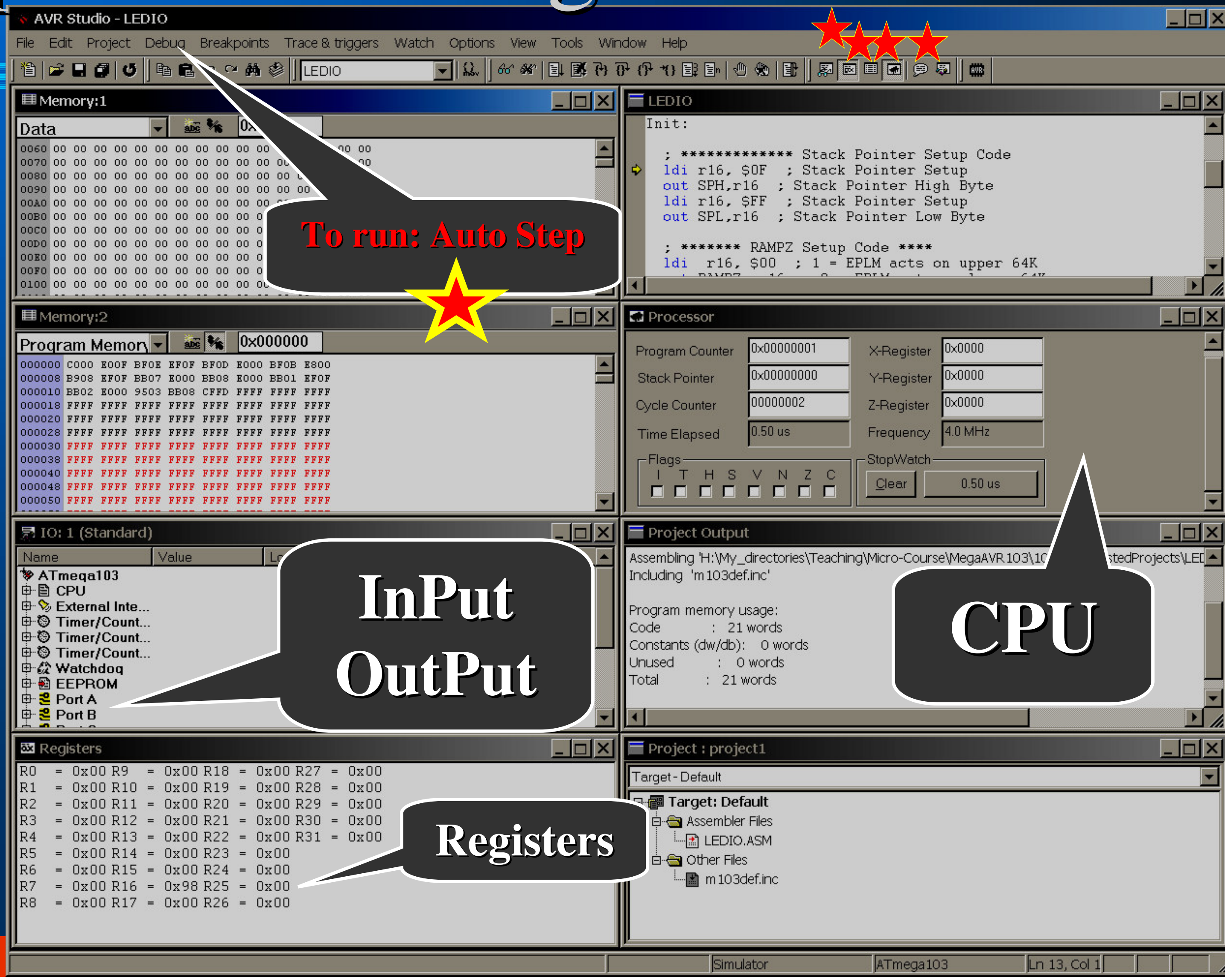
Processor Settings:



Ready to Run :

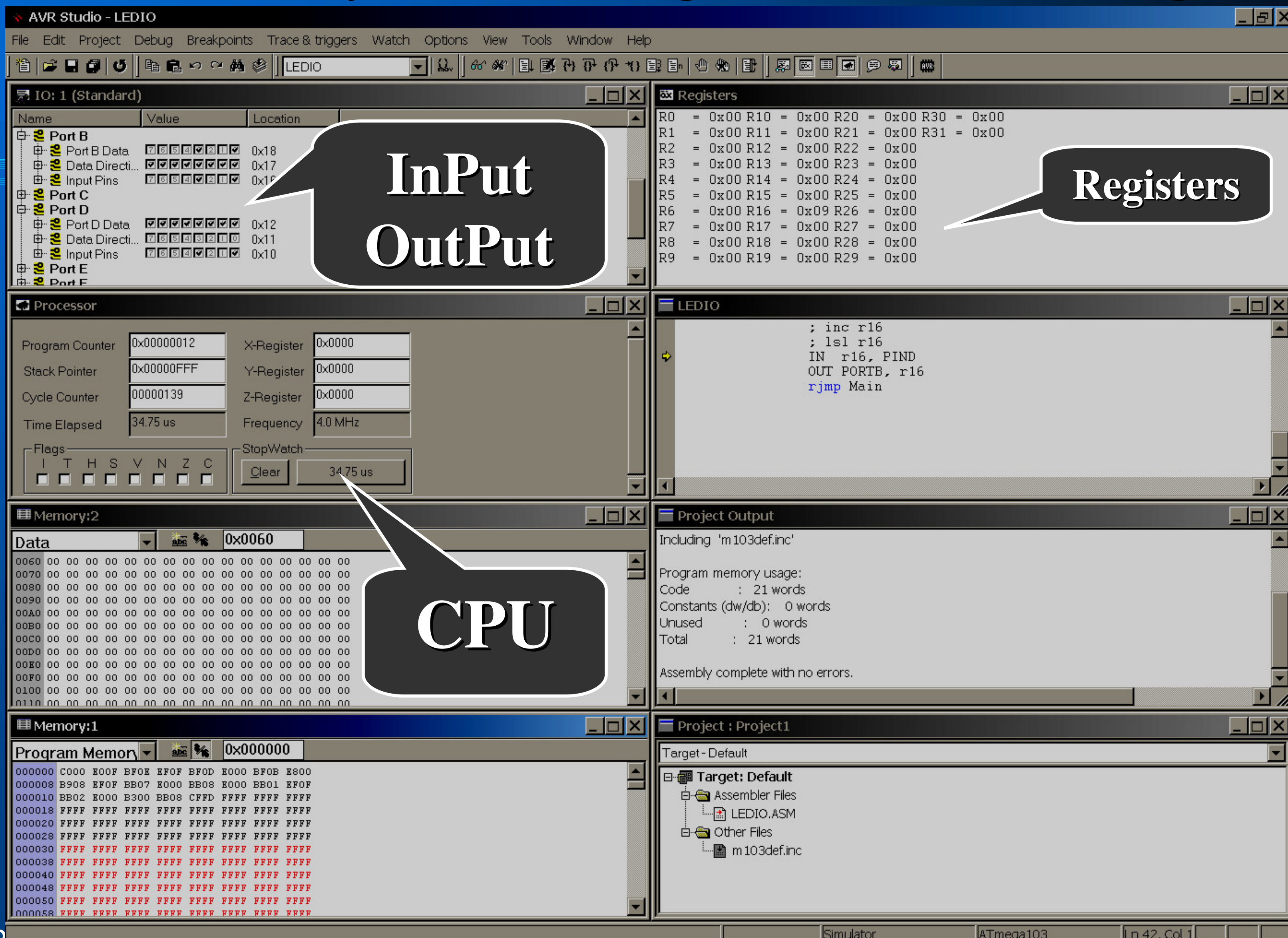

AVR Studio - LEDIO
File Edit Project Debug Breakpoints Trace & triggers Watch Options View Tools Window Help
LEDIO
; ** ATmega103(L) Assembly Language File - IAR Assembler Syntax **
.ORG 0
.include "m103def.inc" ; Add required path to IAR Directory
rjmp Init
; ** Author : Costas Foudas
; ** Company : Imperial College
; ** Comment : Led Program
Your Code
Project Output
Assembling 'H:\My_directories\Teaching\Micro-Course\MegaAVR103\103Code\TestedProjects\LEDIO.ASM'
Including 'm103def.inc'
Program memory usage:
Code : 21 words
Constants (dw/db): 0 words
Unused : 0 words
Total : 21 words
Assembler Results
Project : project1
Target - Default
Target: Default
Assembler Files
LEDIO.ASM
Other Files
m103def.inc
Your Project Files
Simulator
ATmega103
Ln 5, Col 1

Open monitoring Windows:



2/20/2004

Watch your Program Running:



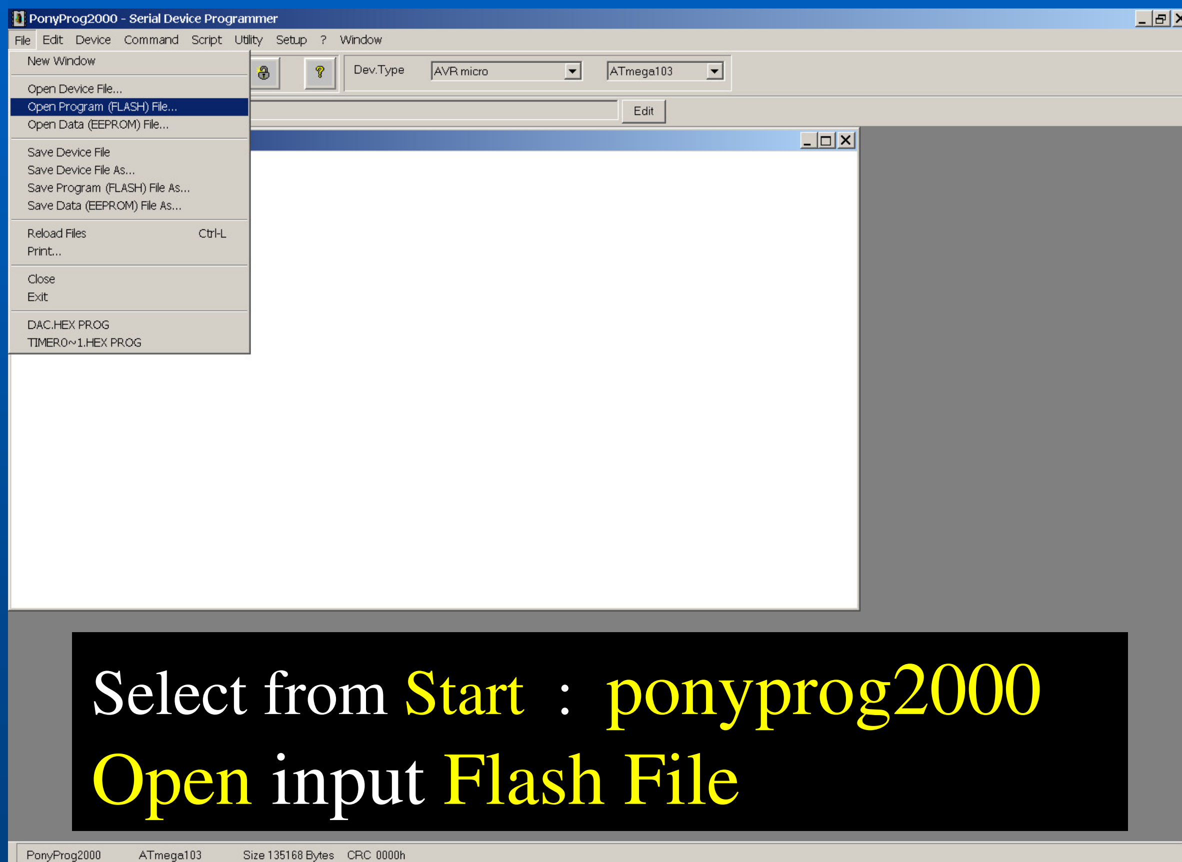
Exercising the ATmega103 commands:

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.

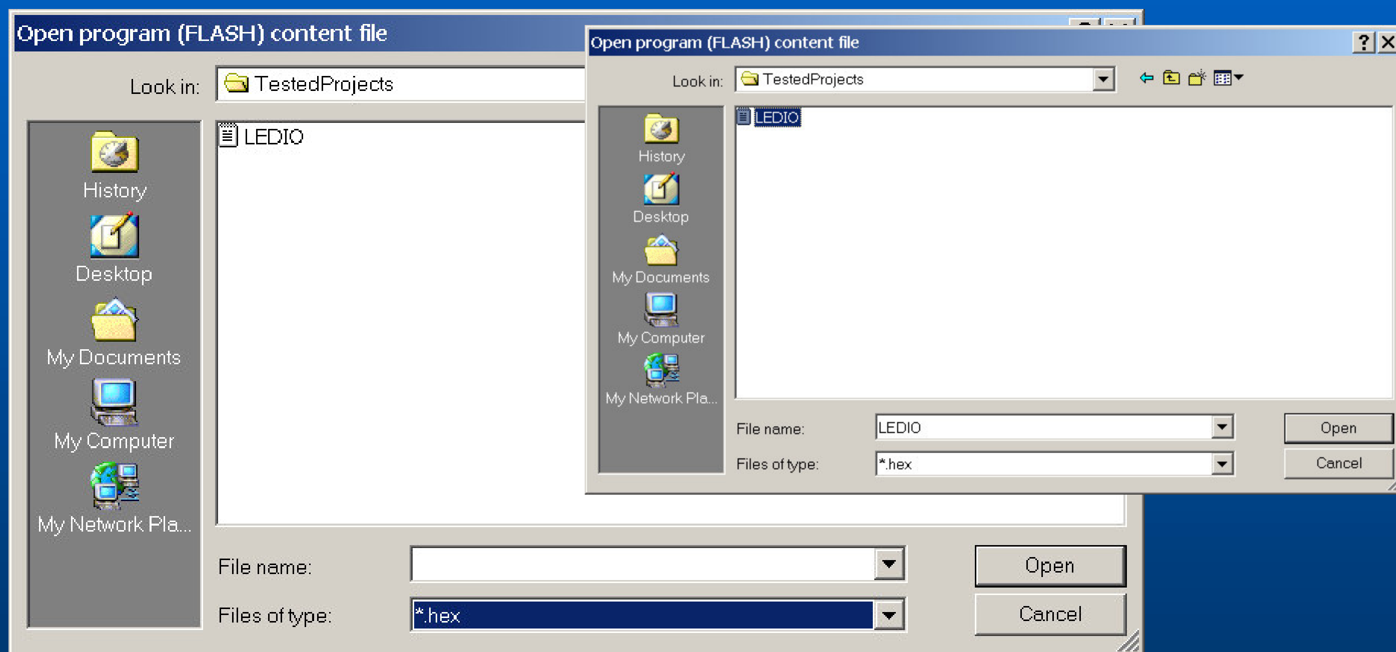
Downloading with Ponyprog I:



Select from Start : ponyprog2000

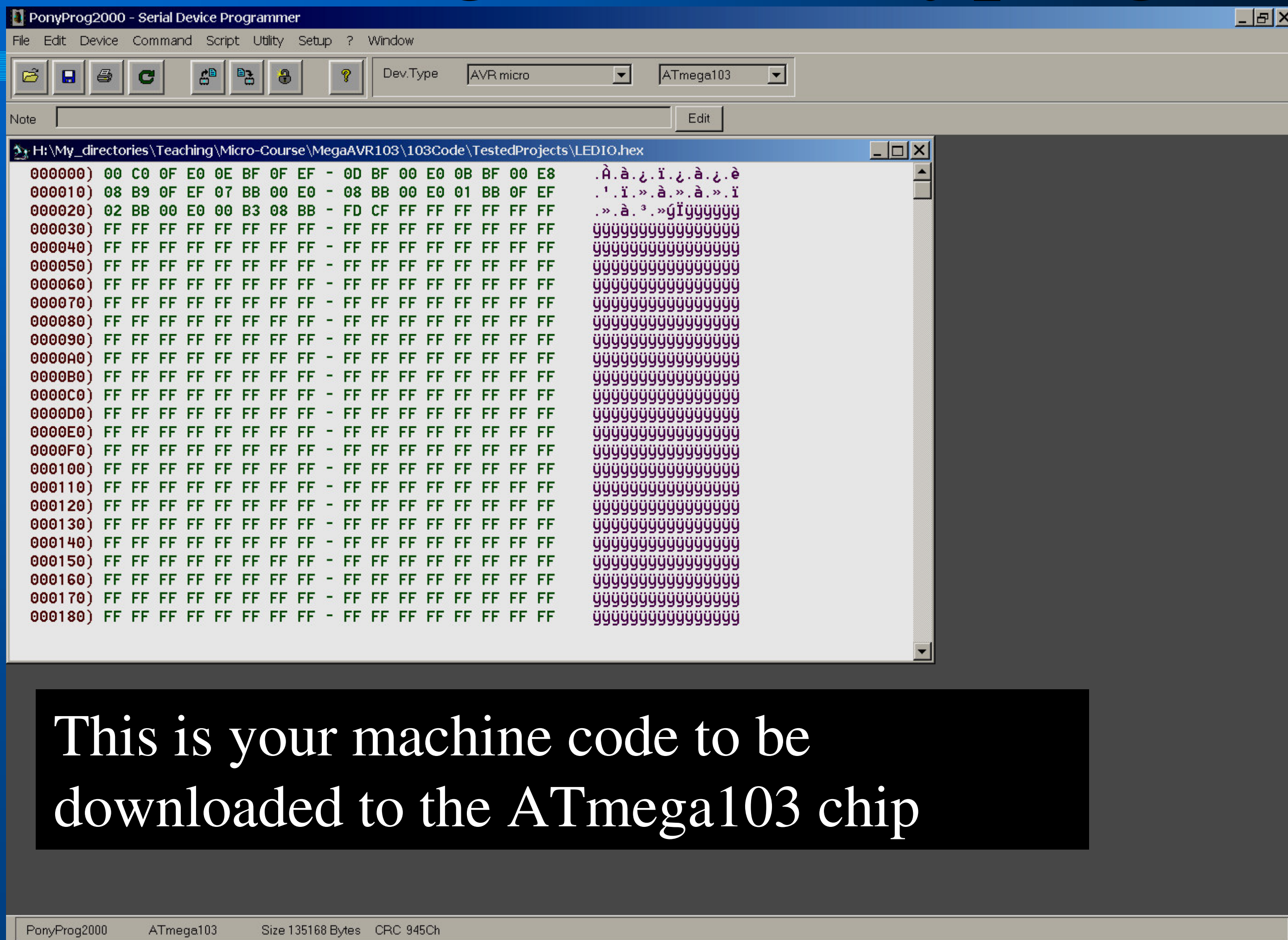
Open input Flash File

Downloading with Ponyprog II:



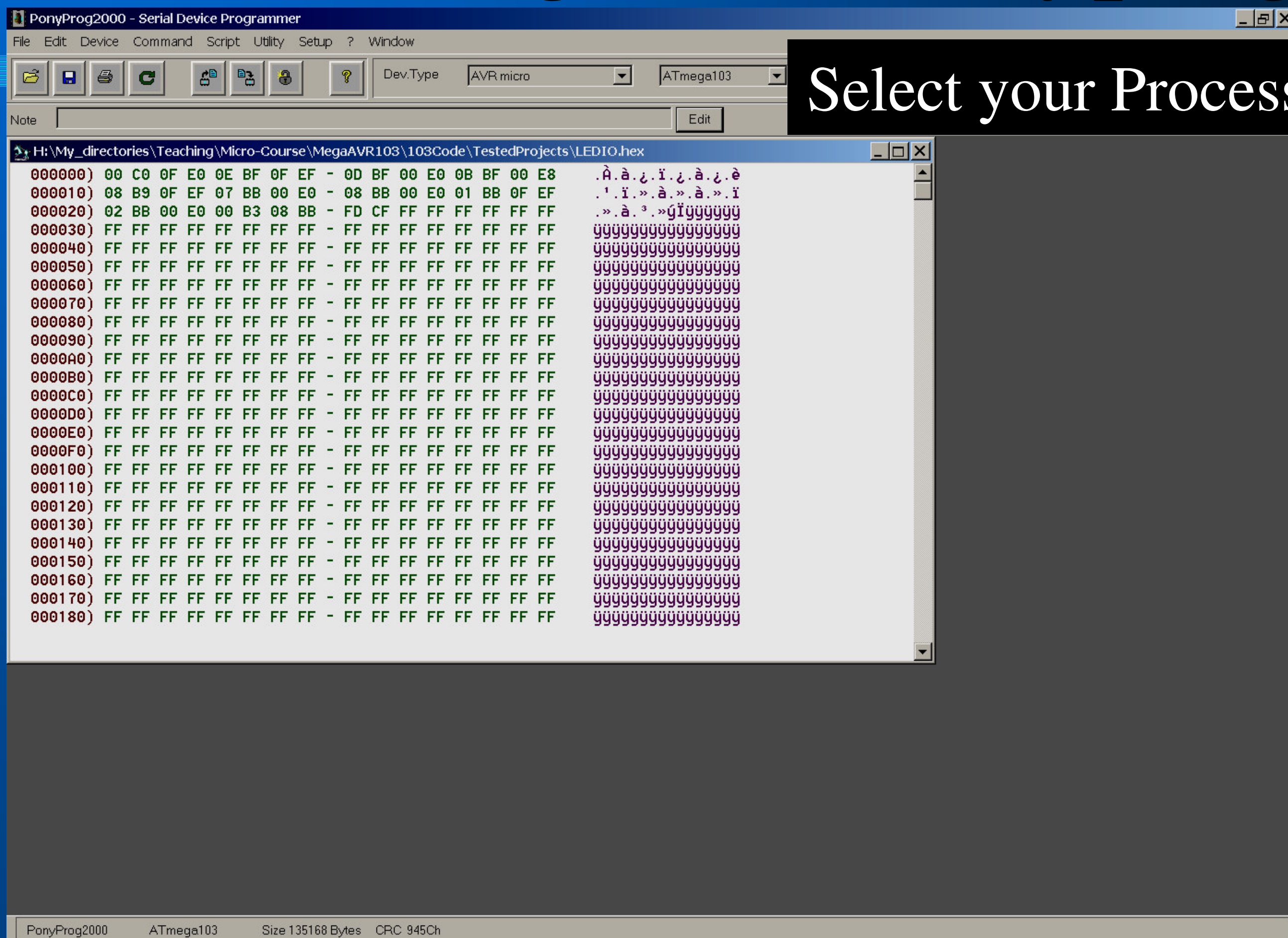
Select from *.hex and LEDIO

Downloading with Ponyprog III:



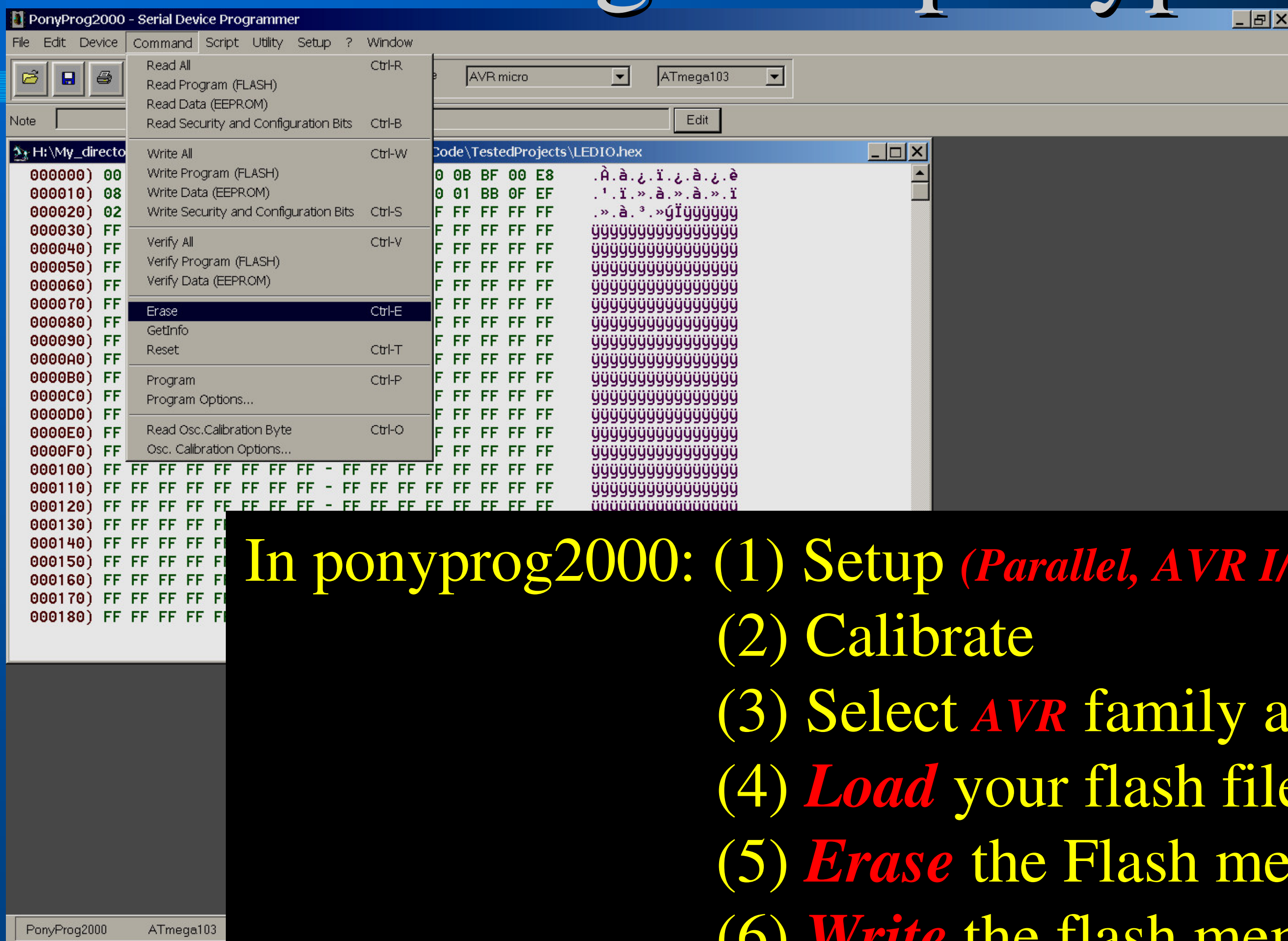
This is your machine code to be downloaded to the ATmega103 chip

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 ?