Laboratory 0

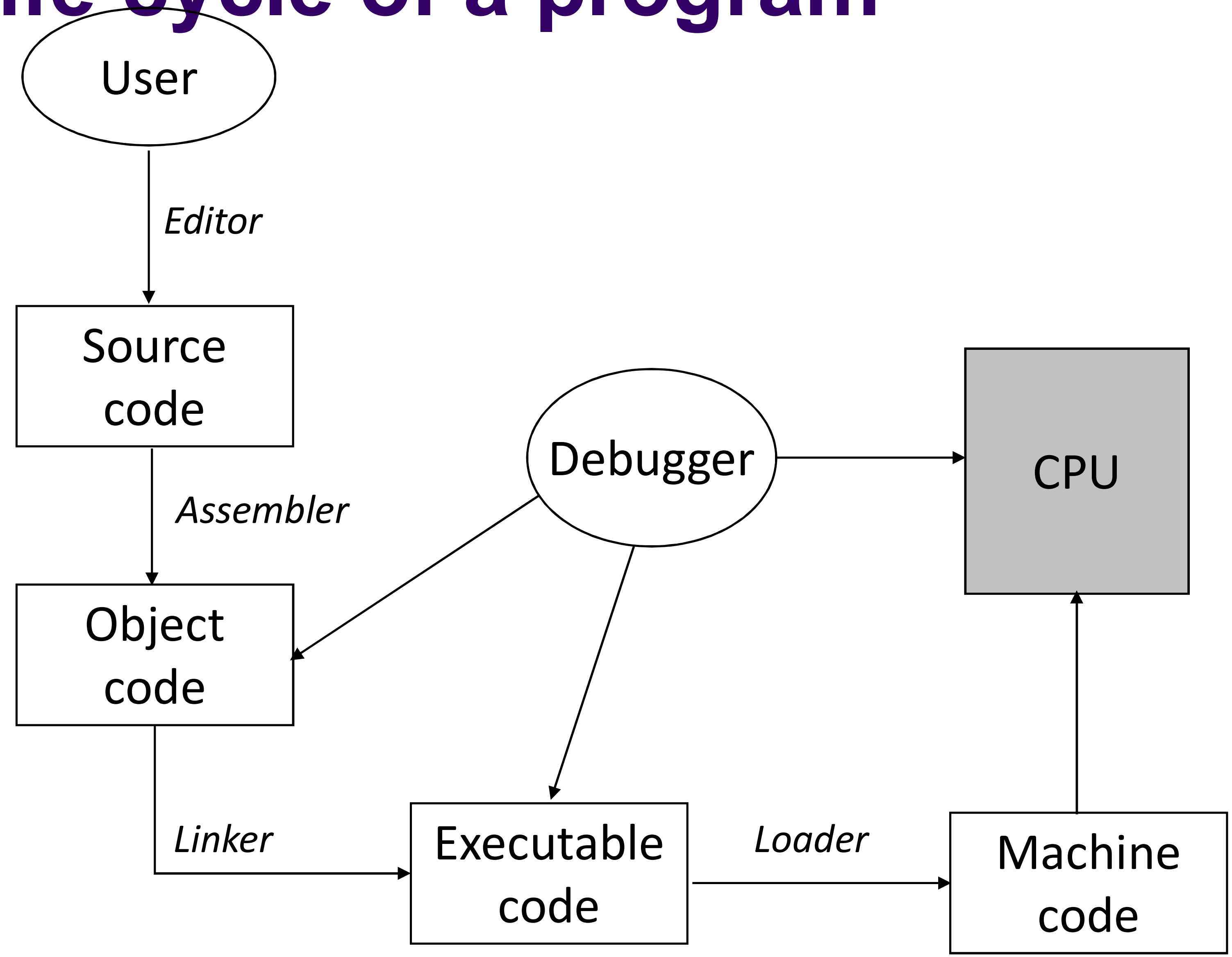
R. Ferrero

Politecnico di Torino

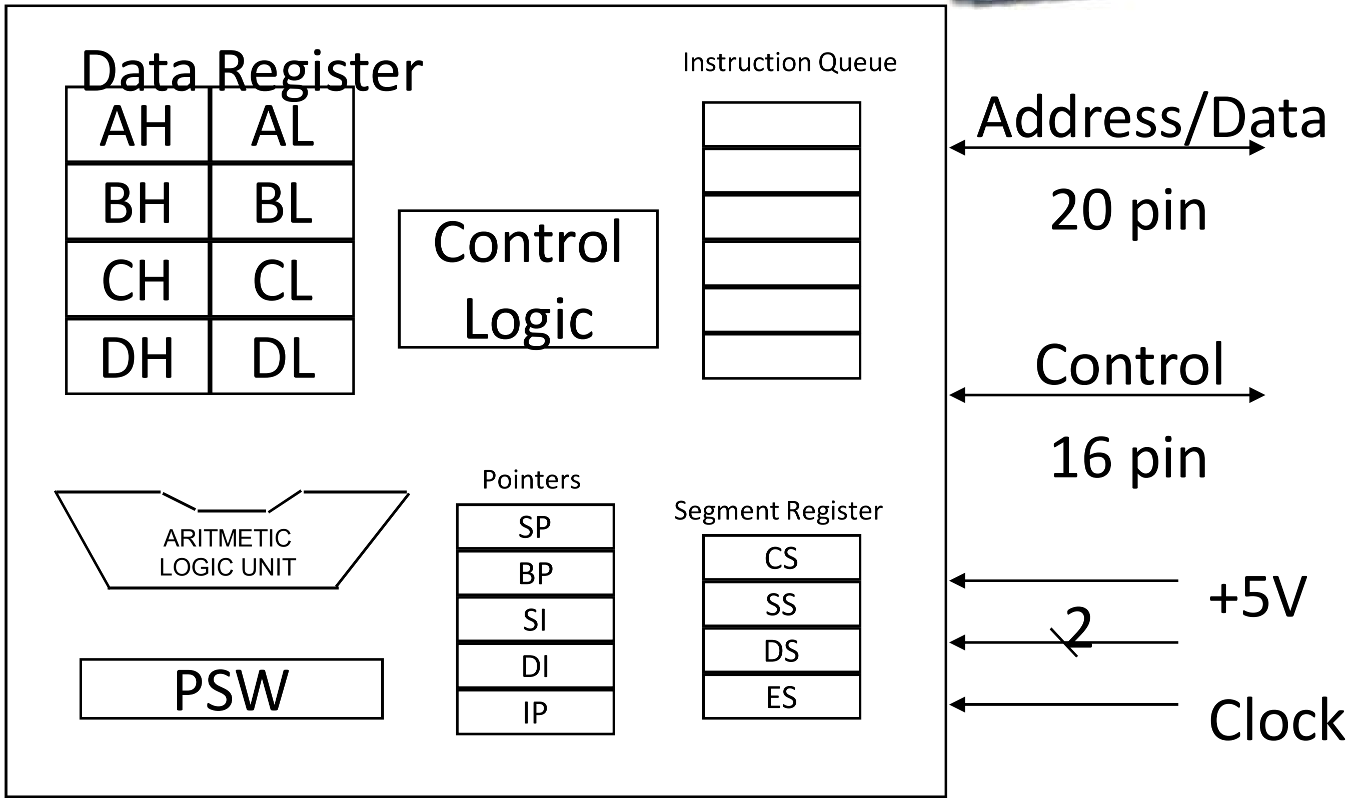
Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

Life cycle of a program



8086 architectural model



Processor Status Word (PSW)

- It is a 16-bit register, but only 9 bits are used.
- Every bit is a flag. Flags can be either:
  - condition flag
  - control flag.

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Condition flags

- They are automatically set at the end of some instructions
  - SF (*Sign Flag*): MSB of the result after an aritmethic instruction
  - ZF (*Zero Flag*): it is 1 is the result is zero, 0 otherwise
  - PF (*Parity Flag*): it is 1 if the result has an even number of bits set to 1, 0 otherwise
  - CF (*Carry Flag*): it is 1 in presence of an arithmetic carry or borrow with unsigned arithmetic instructions
  - AF (*Auxiliary Carry Flag*): in BCD arithmetic, it is 1 with a carry or borrow of the third bit
  - OF (*Overflow Flag*): it is 1 in presence of an overflow with signed arithmetic instructions

				OF	DF	IF	TF	**SF**	**ZF**		**AF**		**PF**		**CF**

Control flag

- They can be written and manipulated by specific instructions, and are used to regulate the functioning of certain processor functions:
  - DF (*Direction Flag*): used by the instructions for string manipulation; if it is 0, the strings are manipulated starting from the characters at the lower address, if it is 1 starting from the largest address
  - IF (*Interrupt Flag):* if it is 1, the maskable Interrupt signals are managed by the CPU, otherwise these are ignored
  - TF (*Trap Flag*): if it is 1, a trap is executed at the end of each instruction.

				OF	**DF**	**IF**	**TF**	SF	ZF		AF		PF		CF

EMU8086

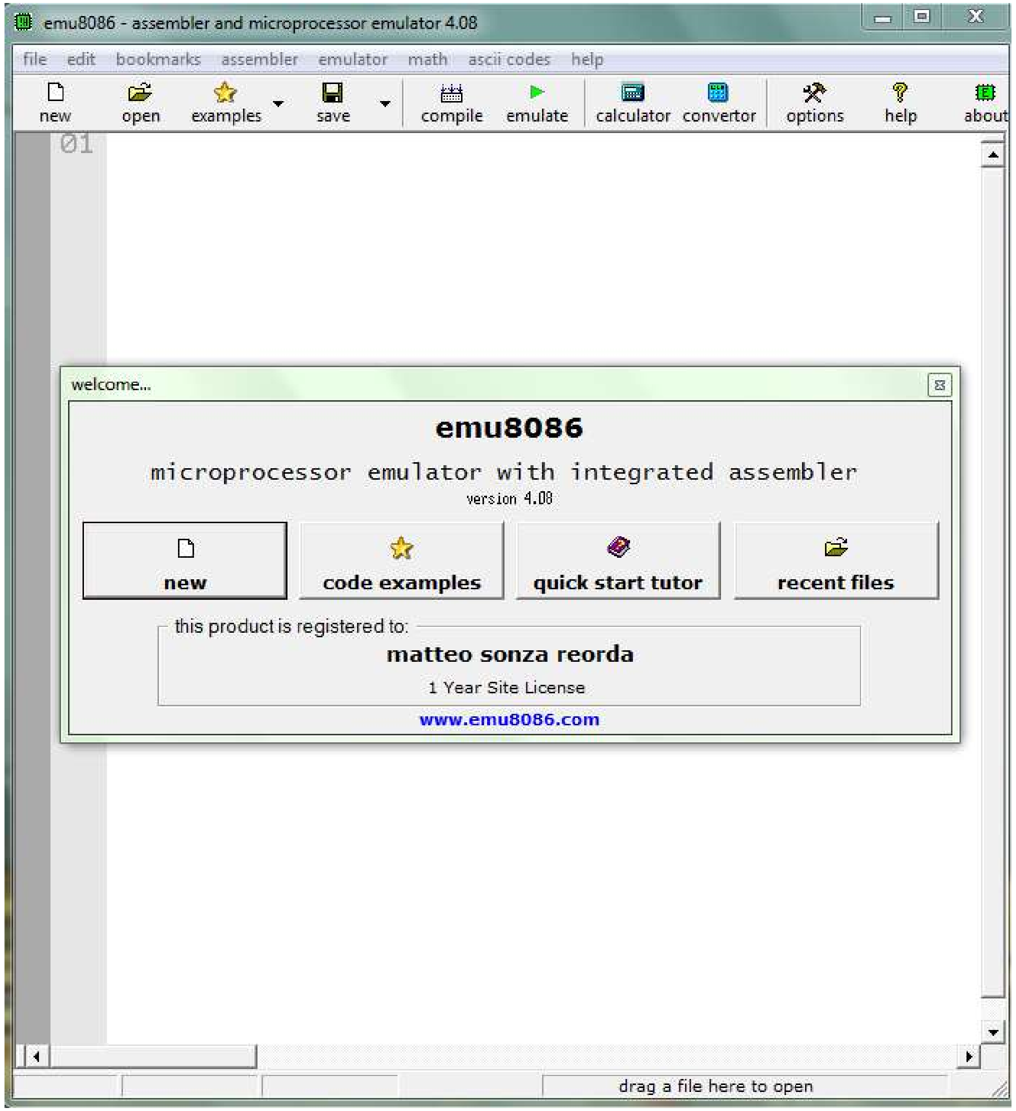
- Emulator of the 8086 processor for *Windows*
  - The compiled code is executed by a virtual machine; the system is not used directly, so crashes are avoided
  - Memory, monitors and I / O devices are emulated
- It allows execution in step-by-step mode
- It integrates a disassembler
- Peripherals can be emulated and new ones can be designed.

EMU8086 [cont.]

- You can download the latest version from the course pages on the portal
- For the current academic year, students can use the license of the Politecnico di Torino:
  - license name: LINO TODESCO
  - license code: 27RX-A747-6I2R-4J2W-1K6O
- The software is already installed in the laboratory.

Main window

- To begin:
  - ***new***
  - ***empty workspace***.



emu8086

microprocessor emulator with integrated assembler

version 4.08

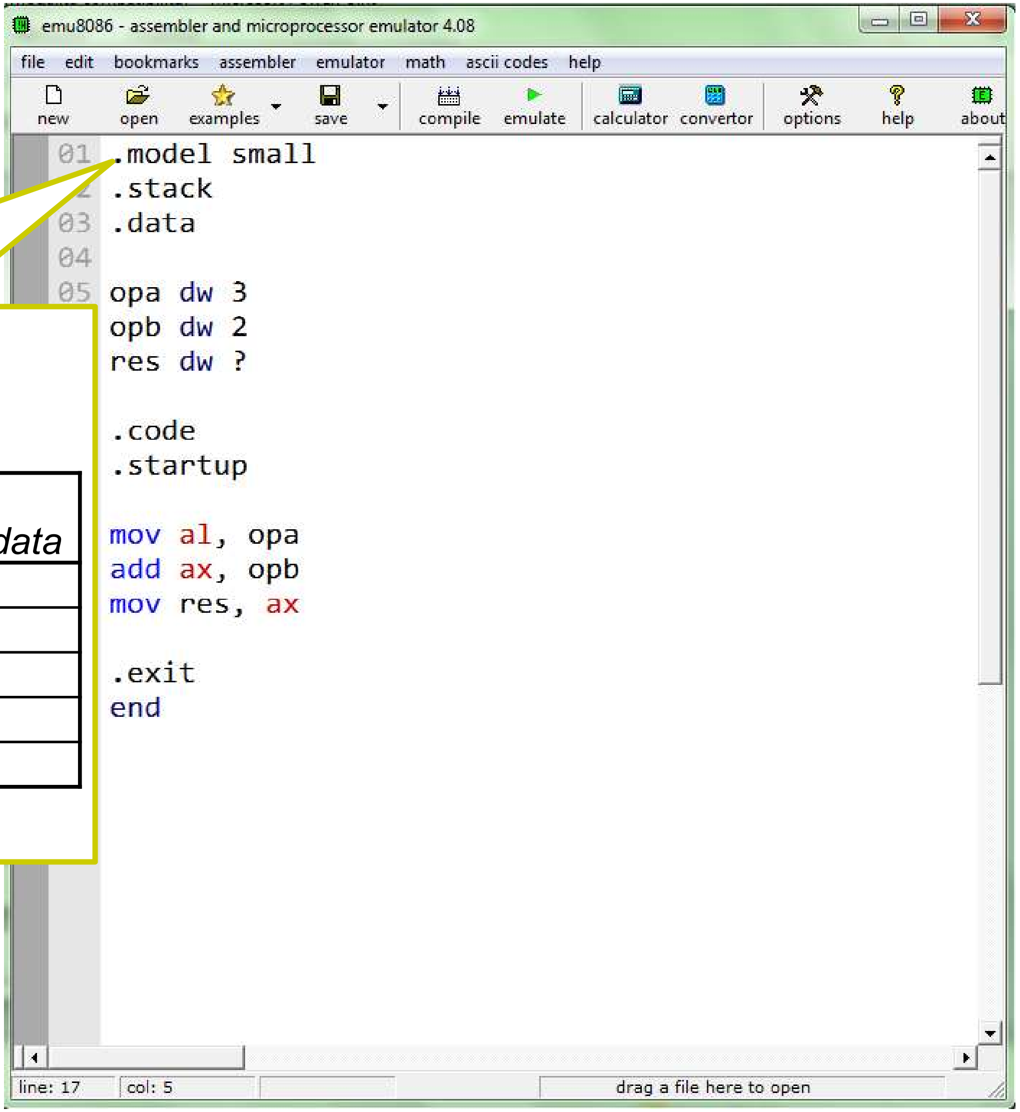
this product is registered to:

matteo sonza reorda

1 Year Site License

www.emu8086.com

Entering the code



```
.model small
.stack
.data

opa dw 3
opb dw 2
res dw ?

.code
.startup

mov al, opa
add ax, opb
mov res, ax

.exit
end
```

`.model` indicates which memory model is used

Memory model	*Code*	*Data*	*Combined code and data*
TINY	NEAR	NEAR	YES
SMALL	NEAR	NEAR	NO
MEDIUM	FAR	NEAR	NO
COMPACT	NEAR	FAR	NO
LARGE or **HUGE**	FAR	FAR	NO

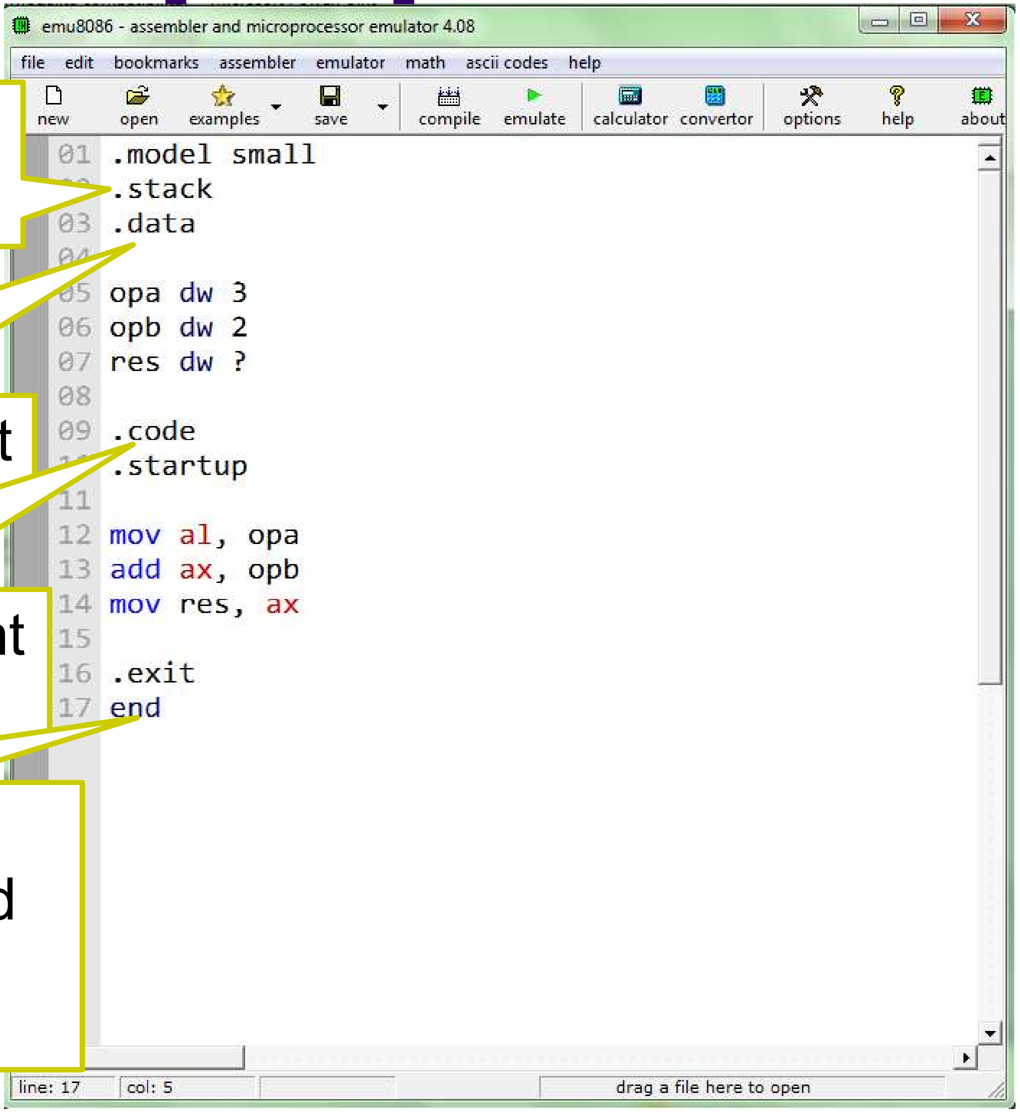
Entering the code [cont.]

`.stack {mem}` creates the stack (default size is 1Kbyte)

`.data` creates the data segment

`.code` creates the code segment

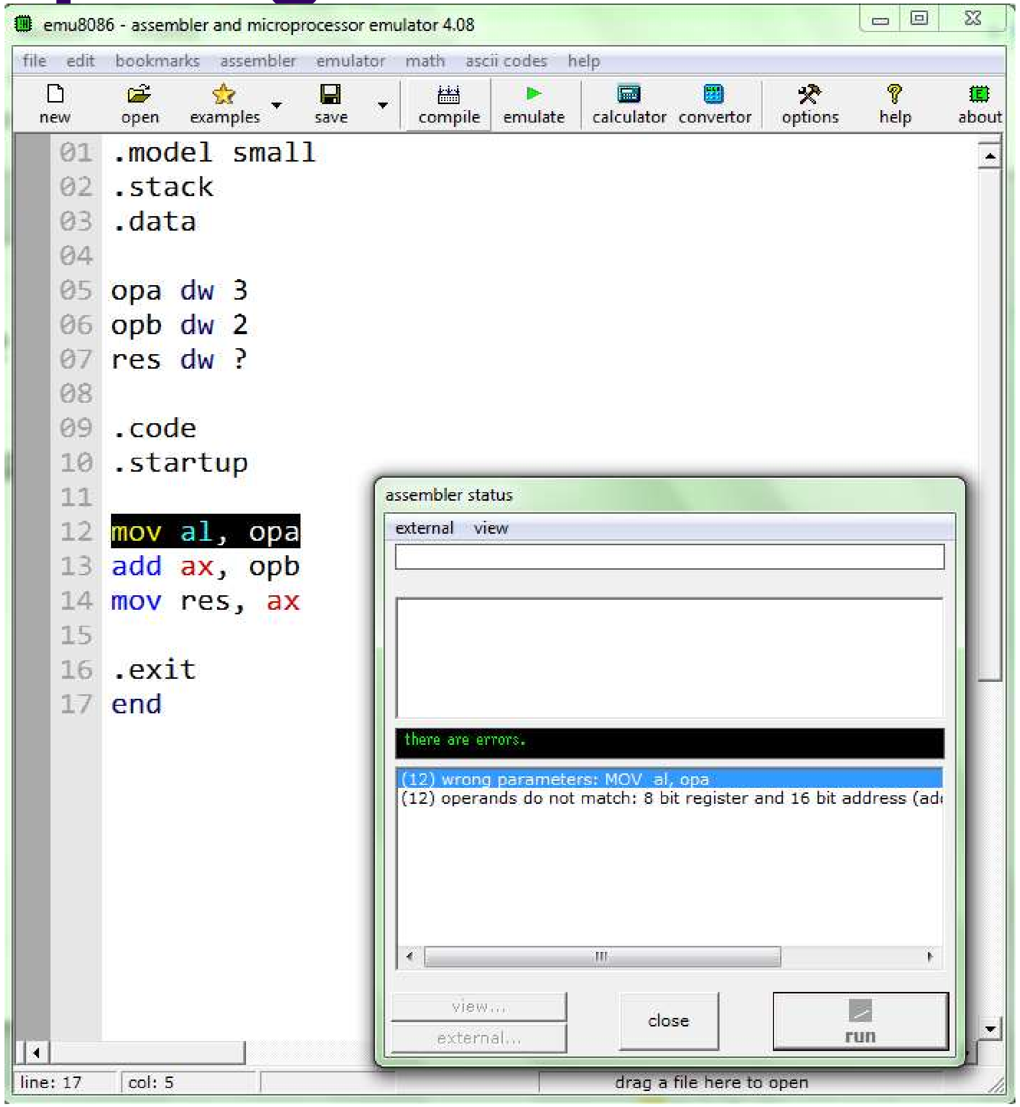
`.startup` and `.exit` produce the machine instructions needed for executing the program in a virtual MS-DOS environment.



```
.model small
.stack
.data

opa dw 3
opb dw 2
res dw ?

.code
.startup

mov al, opa
add ax, opb
mov res, ax

.exit
end
```

Saving and compiling

- To save the source file:
  - ***File > Save as…***
- To compile, either one of:
  - ***compile*** (button)
  - **emulate** (butto)
- Beware of **error messages!**

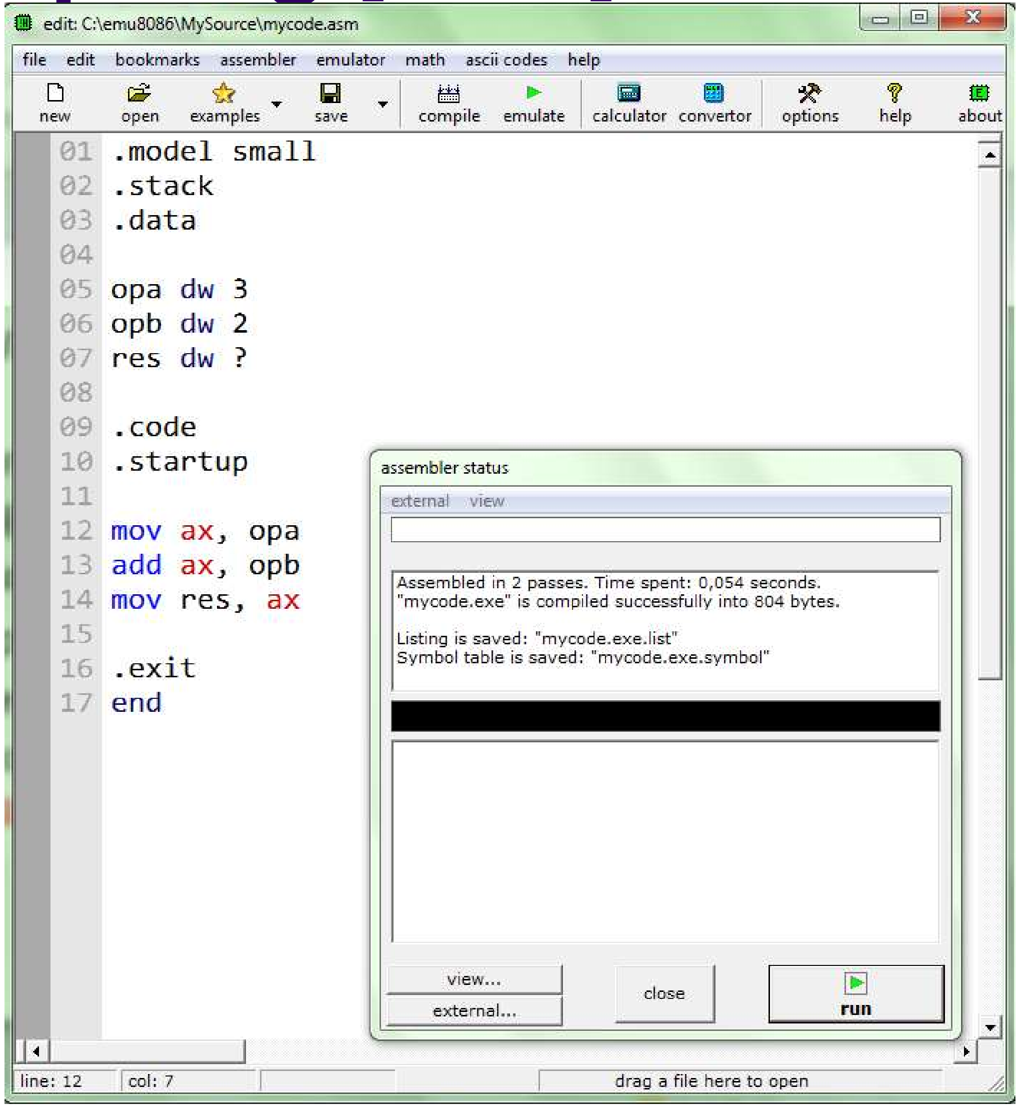


```
01 .model small
02 .stack
03 .data
04
05 opa dw 3
06 opb dw 2
07 res dw ?
08
09 .code
10 .startup
11
12 mov al, opa
13 add ax, opb
14 mov res, ax
15
16 .exit
17 end
```

```
there are errors.
(12) wrong parameters: MOV  al, opa
(12) operands do not match: 8 bit register and 16 bit address (ad
```

Saving and compiling [cont.]

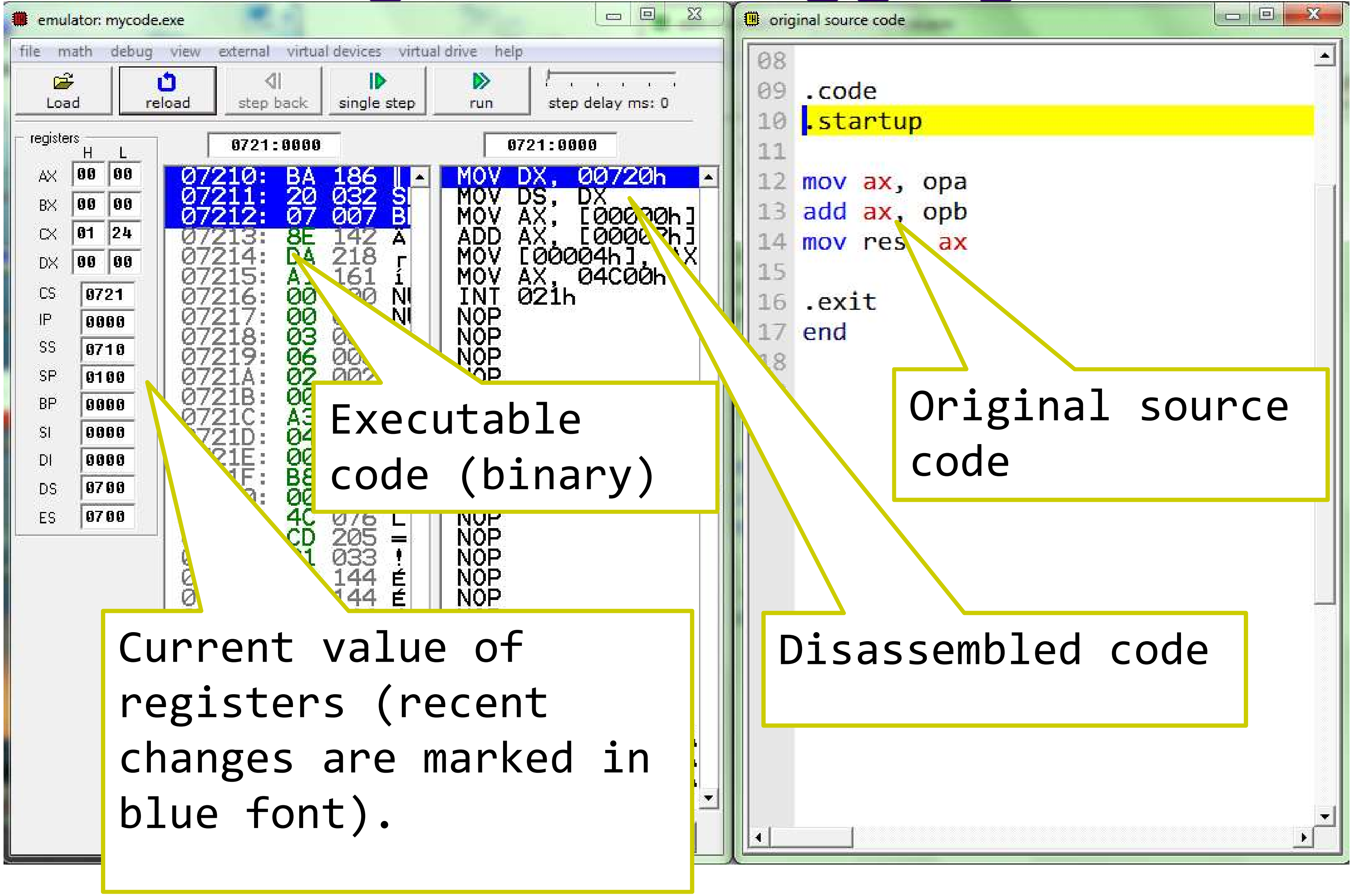
- If the compilation is successful, you can:
  - indicate where to save the executable file
  - emulate the executable file (***run*** button).



```
.model small
.stack
.data

opa dw 3
opb dw 2
res dw ?

.code
.startup

mov ax, opa
add ax, opb
mov res, ax

.exit
end
```

```
Assembled in 2 passes. Time spent: 0,054 seconds.
"mycode.exe" is compiled successfully into 804 bytes.

Listing is saved: "mycode.exe.list"
Symbol table is saved: "mycode.exe.symbol"
```

Emulating and debugging



	H	L
AX	00	00
BX	00	00
CX	01	24
DX	00	00

CS	0721
IP	0000
SS	0710
SP	0100
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

```
0721:0000
07210: BA 186 ║
07211: 20 032 SP
07212: 07 007 BEL
07213: 8E 142 Ä
07214: DA 218 ┌
07215: A1 161 í
07216: 00 000 NULL
07217: 00 000 NULL
07218: 03 00
07219: 06 00
0721A: 02 002
0721B: 00
0721C: A3
0721D: 04
0721E: 00
0721F: B8
07220: 00
       4C 076 L
       CD 205 ═
       21 033 !
       90 144 É
       90 144 É
```

```
0721:0000
MOV DX, 00720h
MOV DS, DX
MOV AX, [00000h]
ADD AX, [00002h]
MOV [00004h], AX
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
```

```
.code
.startup

mov ax, opa
add ax, opb
mov res, ax

.exit
end
```

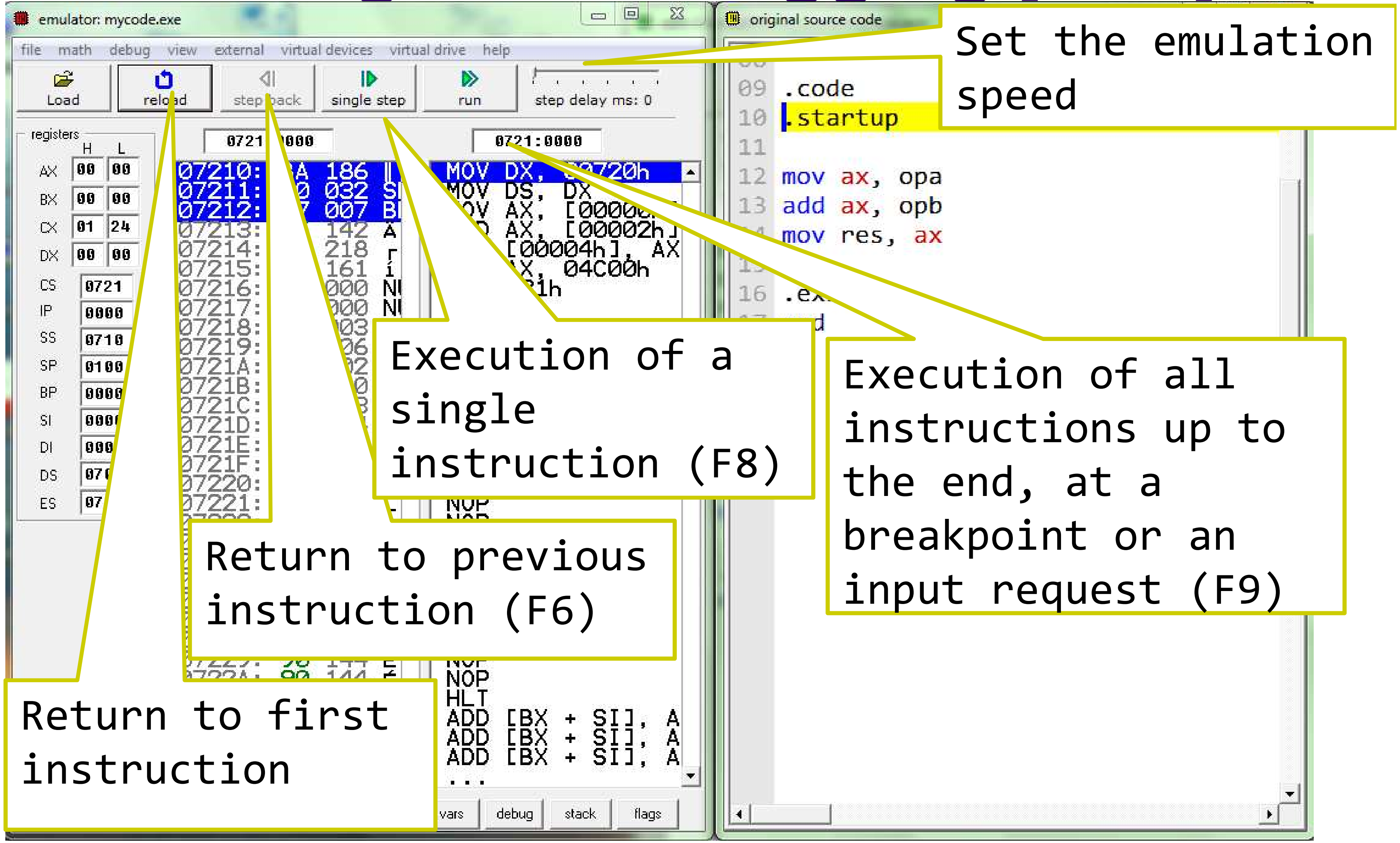
Executable code (binary)

Original source code

Current value of registers (recent changes are marked in blue font).

Disassembled code

Emulating and debugging [cont.]



	H	L
AX	00	00
BX	00	00
CX	01	24
DX	00	00
CS	0721	
IP	0000	
SS	0710	
SP	0100	
BP	0000	
SI	000	
DI	000	
DS	07	
ES	07	

```
0721:0000                     0721:0000
07210: A  186 ║               MOV DX, 0720h
07211: 0  032 S               MOV DS, DX
07212: 7  007 B               OV AX, [0000
07213:    142 A               D AX, [00002h]
07214:    218 ┌                  [00004h], AX
07215:    161 í                  X, 04C00h
07216:    000 NU                 1h
07217:    000 NU
07218:    03
07219:    06
0721A:    02
0721B:    0
0721C:    3
0721D:
0721E:
0721F:
07220:
07221:                        NOP
                              NOP
                              NOP
                              HLT
                              ADD [BX + SI], A
                              ADD [BX + SI], A
                              ADD [BX + SI], A
                              ...
```

```
.code
.startup

mov ax, opa
add ax, opb
mov res, ax

.ex
  d
```

Set the emulation speed

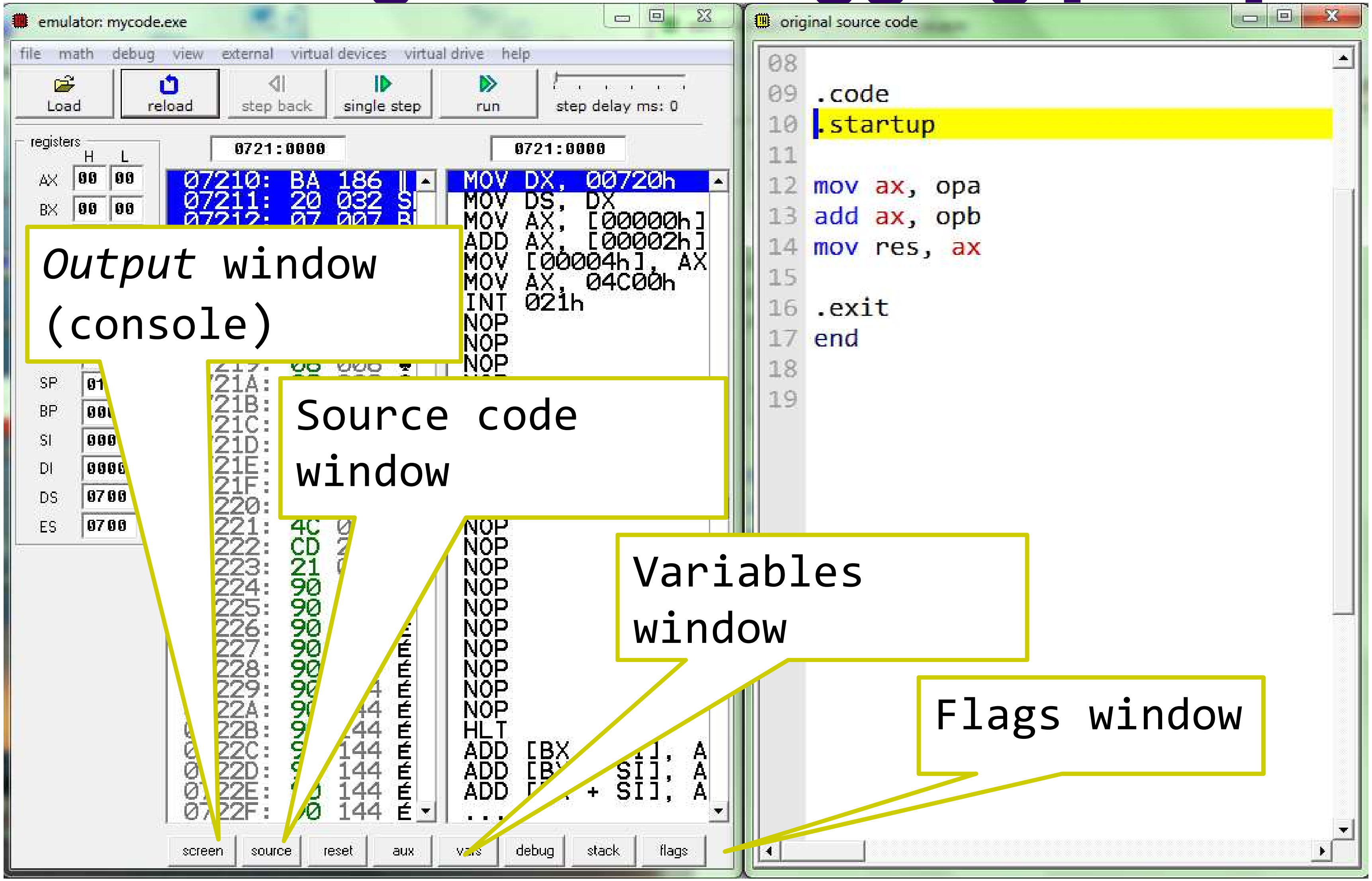
Execution of a single instruction (F8)

Execution of all instructions up to the end, at a breakpoint or an input request (F9)

Return to previous instruction (F6)

Return to first instruction

Emulating and debugging [cont.]



```
MOV DX, 00720h
MOV DS, DX
MOV AX, [00000h]
ADD AX, [00002h]
MOV [00004h], AX
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
```

```
.code
.startup

mov ax, opa
add ax, opb
mov res, ax

.exit
end
```

Output window (console)

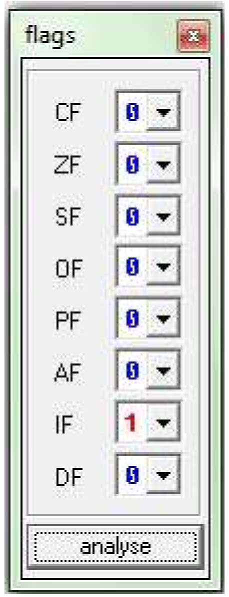
Source code window

Variables window

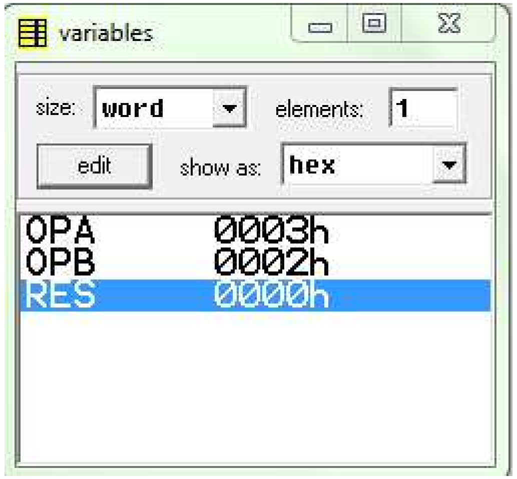
Flags window

Emulating and debugging [cont.]

- Flags window
  - flags modified by the last executed instruction are highlighted in red
- Variables window:
  - You can change the display modes (type, number of items, format)
  - You can change the value of the variable (edit).



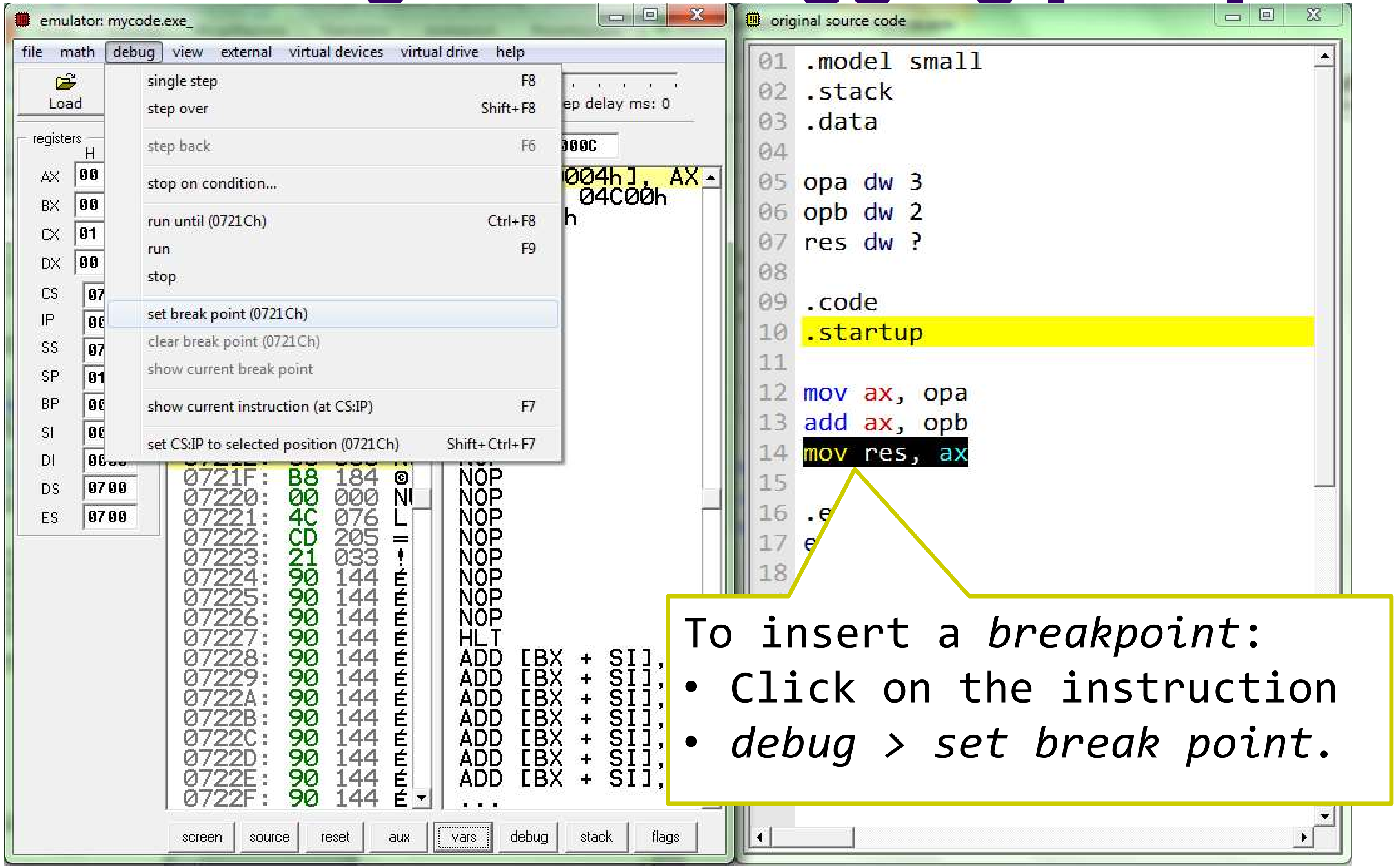
CF 0
ZF 0
SF 0
OF 0
PF 0
AF 0
IF 1
DF 0



size: word
elements: 1
show as: hex

```
OPA         0003h
OPB         0002h
RES         0000h
```

Emulating and debugging [cont.]



```
004h], AX
 04C00h
h
```

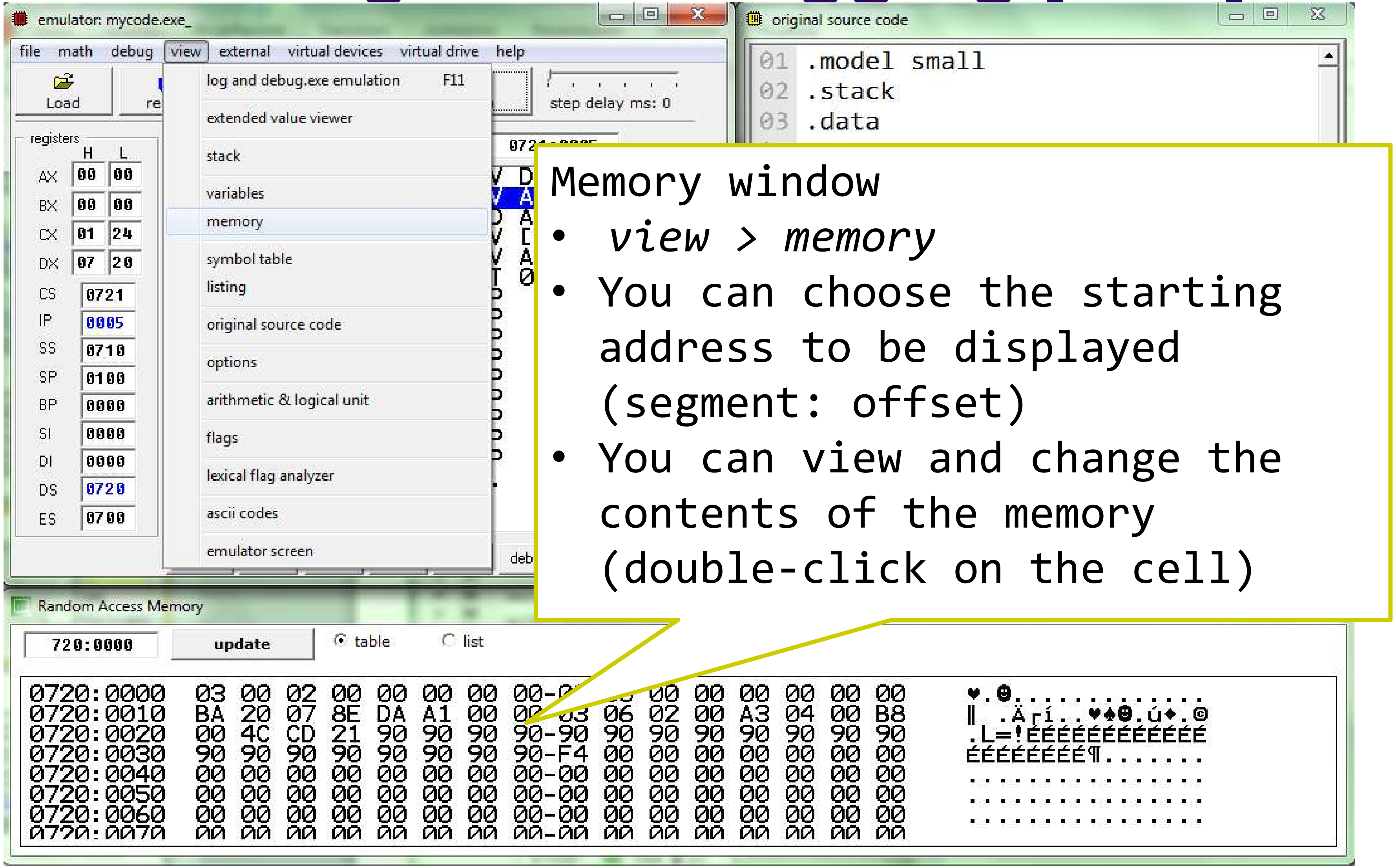
```
0721F: B8 184 ©     NOP
07220: 00 000 NUL   NOP
07221: 4C 076 L     NOP
07222: CD 205 =     NOP
07223: 21 033 !     NOP
07224: 90 144 É     NOP
07225: 90 144 É     NOP
07226: 90 144 É     NOP
07227: 90 144 É     HLT
07228: 90 144 É     ADD [BX + SI],
07229: 90 144 É     ADD [BX + SI],
0722A: 90 144 É     ADD [BX + SI],
0722B: 90 144 É     ADD [BX + SI],
0722C: 90 144 É     ADD [BX + SI],
0722D: 90 144 É     ADD [BX + SI],
0722E: 90 144 É     ADD [BX + SI],
0722F: 90 144 É     ...
```

```
.model small
.stack
.data

opa dw 3
opb dw 2
res dw ?

.code
.startup

mov ax, opa
add ax, opb
mov res, ax

.e
e

```

To insert a *breakpoint*:

- Click on the instruction
- *debug > set break point.*

Emulating and debugging [cont.]



registers

	H	L
AX	00	00
BX	00	00
CX	01	24
DX	07	20

CS	0721
IP	0005
SS	0710
SP	0100
BP	0000
SI	0000
DI	0000
DS	0720
ES	0700

```
01 .model small
02 .stack
03 .data
```

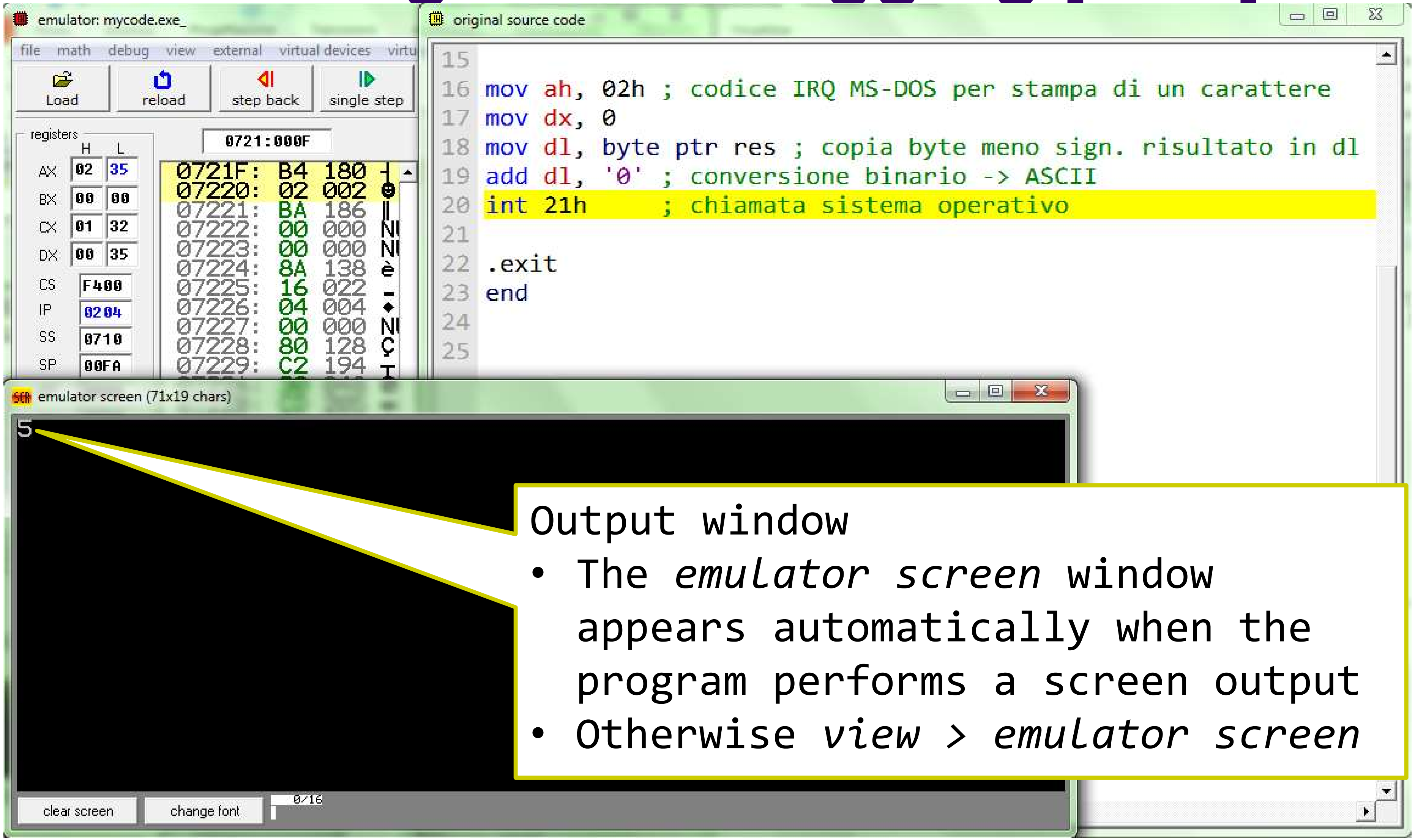
720:0000

```
0720:0000  03 00 02 00 00 00 00 00-00 00 00 00 00 00 00 00   ♥.☻.............
0720:0010  BA 20 07 8E DA A1 00 00-03 06 02 00 A3 04 00 B8   ║ .Äΰí..♥♠☻.ú♦.©
0720:0020  00 4C CD 21 90 90 90 90-90 90 90 90 90 90 90 90   .L=!ÉÉÉÉÉÉÉÉÉÉÉÉ
0720:0030  90 90 90 90 90 90 90 90-F4 00 00 00 00 00 00 00   ÉÉÉÉÉÉÉÉ¶.......
0720:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0720:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Memory window

- *view > memory*
- You can choose the starting address to be displayed (segment: offset)
- You can view and change the contents of the memory (double-click on the cell)

Emulating and debugging [cont.]



```
    H  L
AX  02 35
BX  00 00
CX  01 32
DX  00 35
CS  F400
IP  0204
SS  0710
SP  00FA
```

```
0721:000F
0721F:  B4 180 ┤
07220:  02 002 ☻
07221:  BA 186 ║
07222:  00 000 NU
07223:  00 000 NU
07224:  8A 138 è
07225:  16 022 ▬
07226:  04 004 ♦
07227:  00 000 NU
07228:  80 128 Ç
07229:  C2 194 ┬
```

```
mov ah, 02h ; codice IRQ MS-DOS per stampa di un carattere
mov dx, 0
mov dl, byte ptr res ; copia byte meno sign. risultato in dl
add dl, '0' ; conversione binario -> ASCII
int 21h      ; chiamata sistema operativo

.exit
end
```

```
5
```

Output window
- The *emulator screen* window appears automatically when the program performs a screen output
- Otherwise *view > emulator screen*

Exercises

- The following slides show some code examples.
- It is required to insert these code examples into EMU8086, compile them, run them and analyze their behavior in debug mode.

Writing a value in a register

```
.MODEL small
.STACK
.DATA
.CODE
.STARTUP
MOV    AX, 0
.EXIT
END
```

Writing a value in a memory cell

```
.MODEL small
.STACK
.DATA
VAR    DW     ?
.CODE
.STARTUP
MOV    VAR, 0
.EXIT
END
```

Sum of two values

```
.MODEL small
.STACK
.DATA
OPD1  DW    10
OPD2  DW    24
RESULT      DW     ?
.CODE
.STARTUP
MOV   AX, OPD1
ADD   AX, OPD2
MOV   RESULT, AX
.EXIT
END
```

Sum of the elements of an array (I)

```
        .MODEL SMALL
        .STACK
        .DATA
VETT    DW    5, 7, 3, 4, 3
RESULT  DW    ?
        .CODE
        .STARTUP
        MOV   AX, 0
        ADD   AX, VETT
        ADD   AX, VETT+2
        ADD   AX, VETT+4
        ADD   AX, VETT+6
        ADD   AX, VETT+8
        MOV   RESULT, AX
        .EXIT
        END
```

Sum of the elements of an array (II)

```
DIM     EQU   15
        .MODEL       small
        .STACK
        .DATA
VETT    DW 2, 5, 16, 12, 34, 7, 20, 11, 31, 44, 70, 69, 2,
        4, 23
RESULT  DW    ?
        .CODE
        .STARTUP
        MOV   AX, 0
        MOV   CX, DIM        ; array size now stored in CX
        MOV   DI, 0
```

```
lab:    ADD   AX, VETT[DI] ; add i-th element to AX
        ADD   DI, 2        ; go to next element
        DEC   CX
        CMP   CX, 0        ; compare array index with 0
        JNZ   lab          ; jump if not equal
        MOV   RESULT, AX   ; otherwise, write the result
        .EXIT
        END
```

Read and display a character array

```
DIM     EQU   20
        .MODEL      small
        .STACK
        .DATA
VETT    DB    DIM DUP(?)
        .CODE
        .STARTUP
        MOV   CX, DIM
        MOV   DI, 0
        MOV   AH, 1          ; set AH for reading
```

```
lab1:  INT   21H          ; read a character
       MOV   VETT[DI], AL ; store the character
       INC   DI           ; go to next element
       DEC   CX
       CMP   CX, 0        ; compare array index with 0
       JNZ   lab1         ; jump if not equal
       MOV   CX, DIM
       MOV   AH, 2        ; set AH for writing
lab2:  DEC   DI           ; go to next element
       MOV   DL, VETT[DI]
       INT   21H          ; display the character
       DEC   CX
       CMP   CX, 0
       JNZ   lab2
       .EXIT
       END
```

Search for the minimum character

```
        .MODEL      small
        .STACK
DIM     EQU    20
        .DATA
TABLE   DB     DIM DUP(?)
        .CODE
        .STARTUP
        MOV    CX, DIM
        LEA    DI, TABLE
        MOV    AH, 1              ; reading
lab1:   INT    21H
        MOV    [DI], AL
        INC    DI
        DEC    CX
        CMP    CX, 0
        JNE    lab1               ; loop 20 times
        MOV    CL, 0FFH
        MOV  DI, 0
ciclo:  CMP   CL, TABLE[DI]; compare with current minimum
        JB    dopo
        MOV   CL, TABLE[DI]; store new minimum
dopo:   INC   DI
        CMP   DI, DIM
        JB    ciclo
output: MOV   DL, CL
        MOV   AH, 2
        INT   21H               ; display
        .EXIT
        END
```