

Calcolatori Elettronici

Esercitazioni Assembler

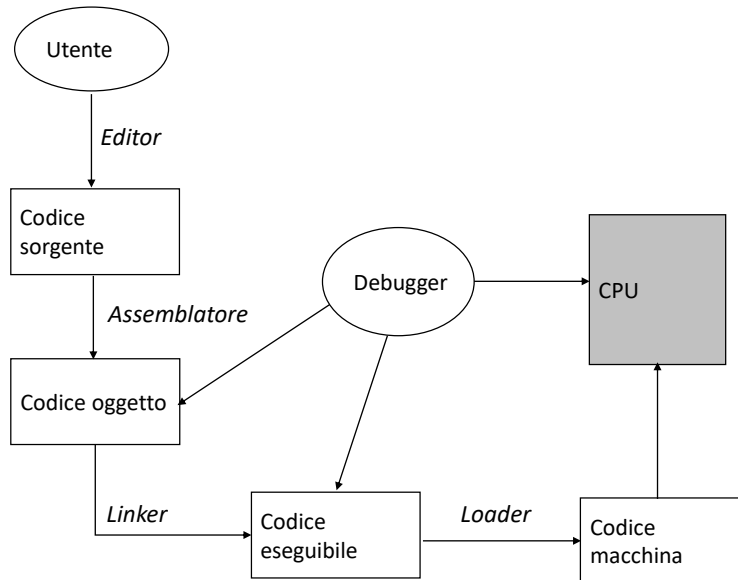
Maurizio Rebaudengo – Renato Ferrero
renato.ferrero@polito.it

Politecnico di Torino
Dipartimento di Automatica e Informatica

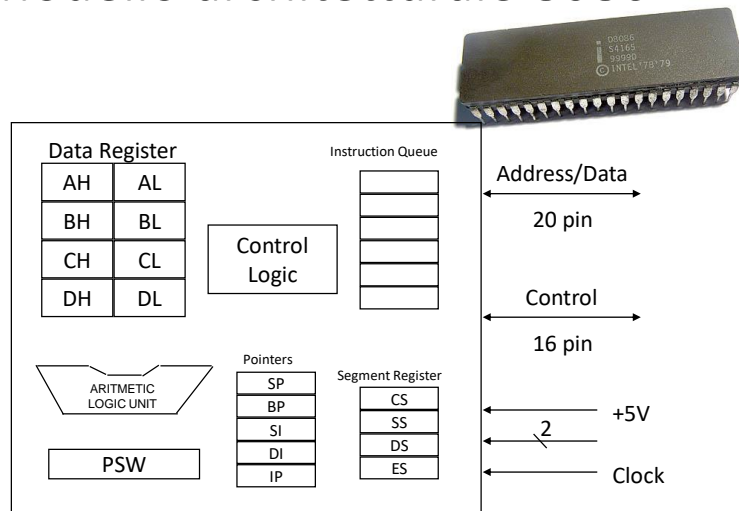
Orari laboratorio

- LABINF
 - 1° piano, ingresso da C.so Castelfidardo lato ovest
- Esercitazioni assistite: 2 squadre
 - Mercoledì h. 11.30-13.00 (L – P)
 - Mercoledì h. 13.00-14.30 (R – Z).

Ciclo di vita di un programma

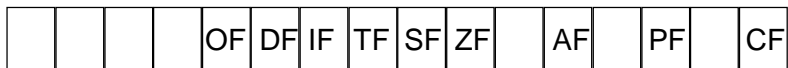


Modello architetturale 8086



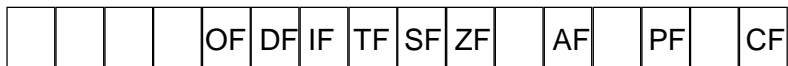
Processor Status Word (PSW)

- È composta da 16 bit, ma solo 9 di questi sono usati. Ogni bit corrisponde ad un flag.
- I flag si dividono in:
 - flag di condizione
 - flag di controllo.



Flag di condizione

- Sono automaticamente scritti al termine di varie operazioni:
 - SF (*Sign Flag*): coincide con il MSB del risultato dopo un'istruzione aritmetica
 - ZF (*Zero Flag*): vale 1 se il risultato è nullo, 0 altrimenti
 - PF (*Parity Flag*): vale 1 se il numero di 1 negli 8 bit meno significativi del risultato è pari, 0 altrimenti
 - CF (*Carry Flag*): dopo le istruzioni aritmetiche vale 1 se c'è stato riporto (somma) o prestito (sottrazione); altre istruzioni ne fanno un uso particolare
 - AF (*Auxiliary Carry Flag*): usato nell'aritmetica BCD; vale 1 se c'è stato riporto (somma) o prestito (sottrazione) dal bit 3
 - OF (*Overflow Flag*): vale 1 se l'ultima istruzione ha prodotto overflow.



Flag di controllo

- Possono venire scritti e manipolati da apposite istruzioni, e servono a regolare il funzionamento di talune funzioni del processore:
 - DF (*Direction Flag*): utilizzato dalle istruzioni per la manipolazione delle stringhe; se vale 0 le stringhe vengono manipolate partendo dai caratteri all'indirizzo minore, se vale 1 a partire dall'indirizzo maggiore
 - IF (*Interrupt Flag*): se vale 1, i segnali di Interrupt mascherabili vengono percepiti dalla CPU, altrimenti questi vengono ignorati
 - TF (*Trap Flag*): se vale 1, viene eseguita una *trap* al termine di ogni istruzione.

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

Informazioni utili

- Utilizzando google riuscirete a trovare informazioni sul linguaggio Assembly 8086
- Link utili:
 - <http://www.giobe2000.it/Tutorial/Schede/Home.asp>
(in particolare la scheda 7 descrive l'istruzione set completo).



EMU8086

- Emulatore del processore 8086 per *MS-Windows*
 - Il codice compilato è eseguito da una *macchina virtuale*; il sistema non è utilizzato in modo diretto, per cui i *crash* sono evitati
 - Memoria, monitor e dispositivi di I/O sono emulati
- Convenzioni di linguaggio come MASM (quasi tutto)
- Permette l'esecuzione in modalità *step-by-step*
- Integra un disassemblatore
- Permette di emulare periferiche e di progettarne di nuove.

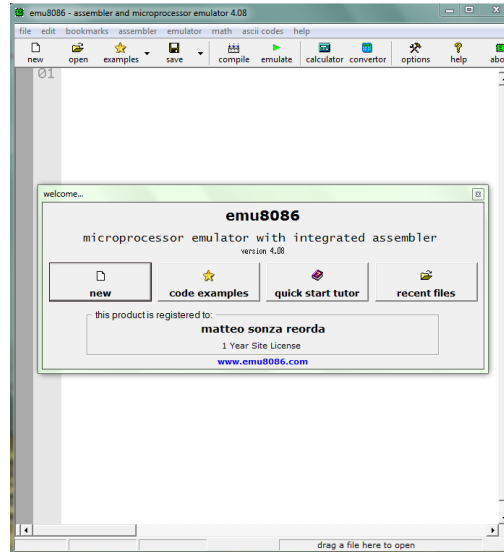


EMU8086 [cont.]

- È possibile scaricare la versione più recente dalle pagine del corso sul portale della didattica
- Per l'anno accademico corrente, gli studenti possono utilizzare la licenza di Istituto del Politecnico:
 - license name: LINO TODESCO
 - license code: 27RX-A747-6I2R-4J2W-1K6O
- NB: in laboratorio il software è già installato!

Main window

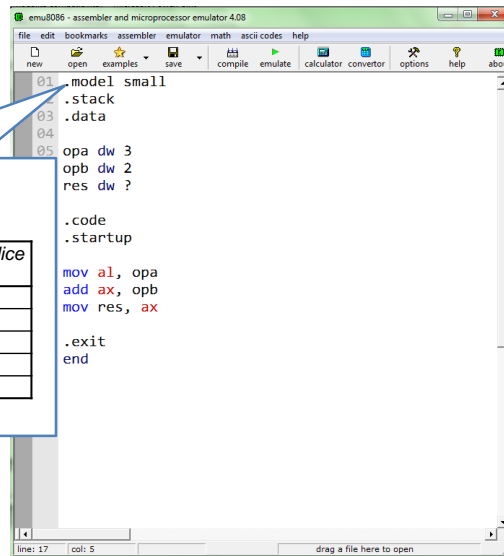
- Per iniziare:
 - *new*
 - *empty workspace.*



Inserimento del codice

.model stabilisce il modello di memoria desiderato

Modello di Memoria	Codice	Dati	Dati e Codice Combinati
TINY	NEAR	NEAR	SI
SMALL	NEAR	NEAR	NO
MEDIUM	FAR	NEAR	NO
COMPACT	NEAR	FAR	NO
LARGE o HUGE	FAR	FAR	NO



Inserimento del codice [cont.]

Diagram illustrating the insertion of code into the emu8086 assembler and microprocessor emulator 4.08. The code is shown in the main window, and callouts explain the directives:

- `.stack {mem}` crea lo stack (dimensione default pari a 1Kbyte)
- `.data` crea il segmento di dato
- `.code` crea il segmento di codice
- `.startup` ed `.exit` producono le istruzioni macchina necessarie all'esecuzione del programma in ambiente MS-DOS virtuale

```
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
```

Salvataggio e compilazione

- Per salvare il file sorgente:
 - **File > Save as...**
- Per compilare:
 - **compile** (pulsante)
 - Attenzione ai **messaggi d'errore!**
 - Alternativa: pulsante **emulate**.

Diagram illustrating the compilation process in the emu8086 assembler and microprocessor emulator 4.08. The code is shown in the main window, and an error message is displayed in the assembler status window:

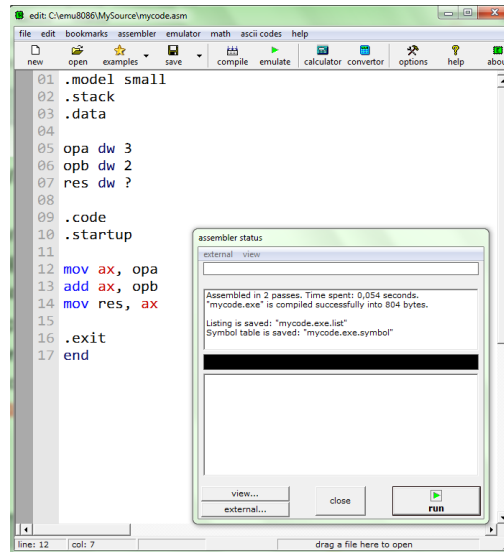
```
01 .model small
02 .stack
03 .data
04
05 opa dw 3
06 opb dw 2
07 res dw ?
08
09 .code
10 startup
11
12 mov al, opa
13 add ax, opb
14 mov res, ax
15
16 .exit
17 end
```

Assembler status window:

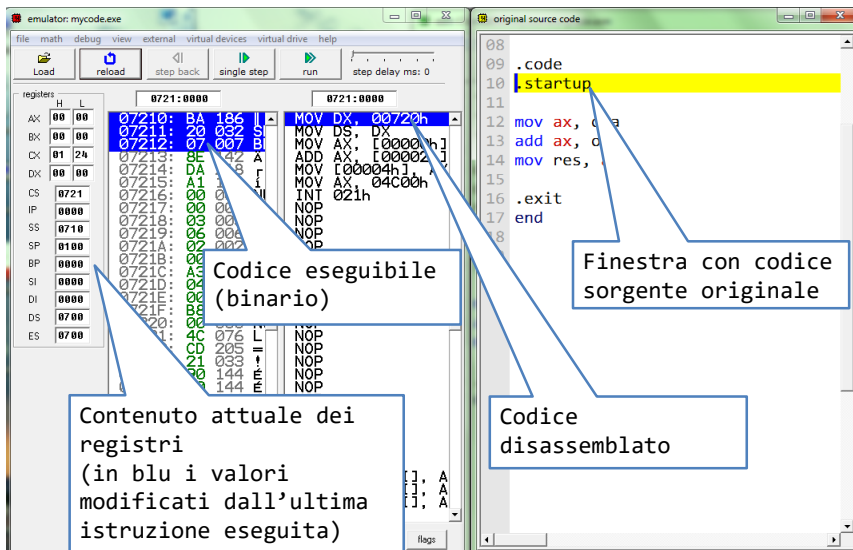
```
There are errors.
(12) wrong parameters: MOV al, opa
(12) operands do not match: 8 bit register and 16 bit address (add)
```

Salvataggio e compilazione [cont.]

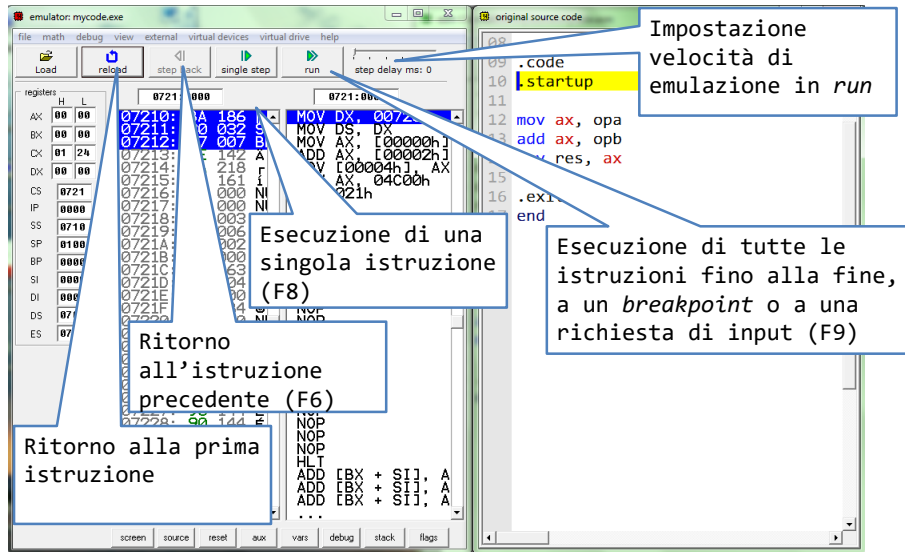
- Se la compilazione va a buon fine...
 - è richiesto dove salvare l'eseguibile
 - si può procedere all'emulazione (pulsante **run**).



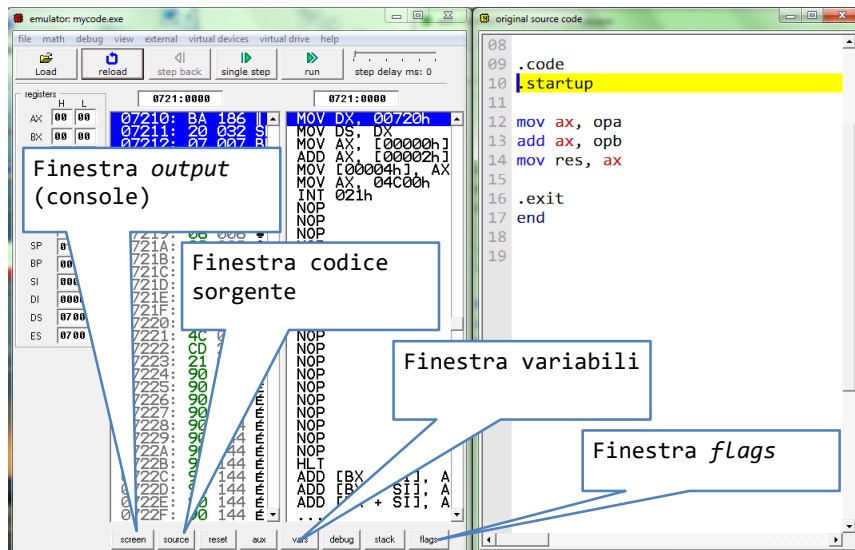
Emulazione e debug



Emulazione e debug [cont.]

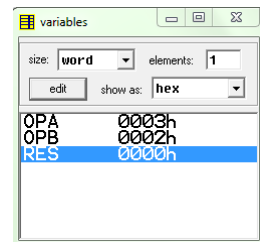
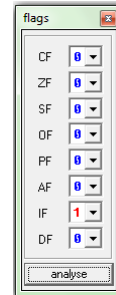


Emulazione e debug [cont.]

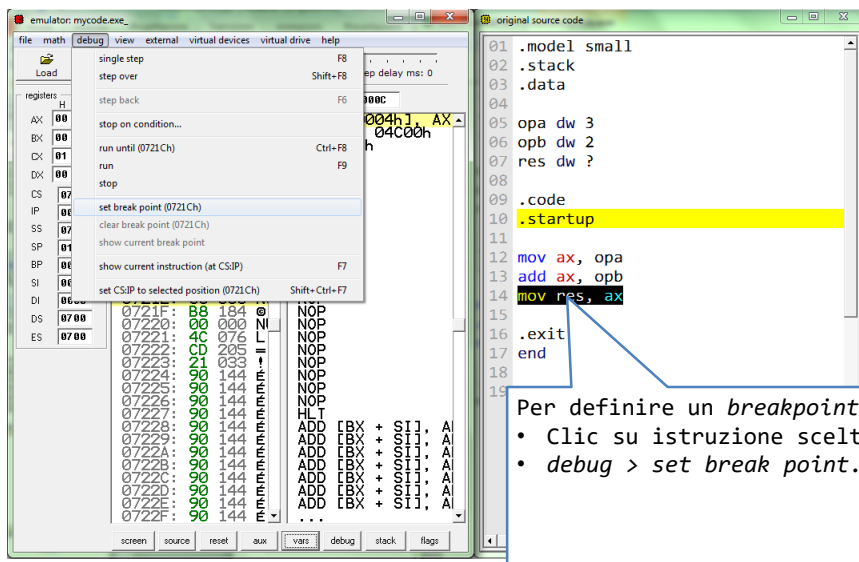


Emulazione e debug [cont.]

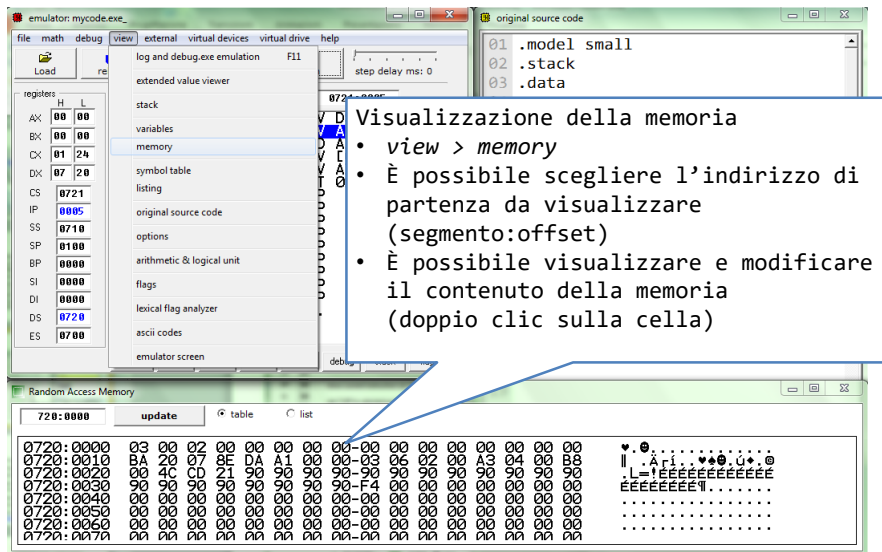
- Finestra *flags*:
 - Le flag modificate dall'ultima istruzione eseguita sono evidenziate in rosso
- Finestra *variables*:
 - È possibile modificare le modalità di visualizzazione (tipo, num. di elementi, formato)
 - È possibile modificare il valore della variabile (*edit*).



Emulazione e debug [cont.]



Emulazione e debug [cont.]



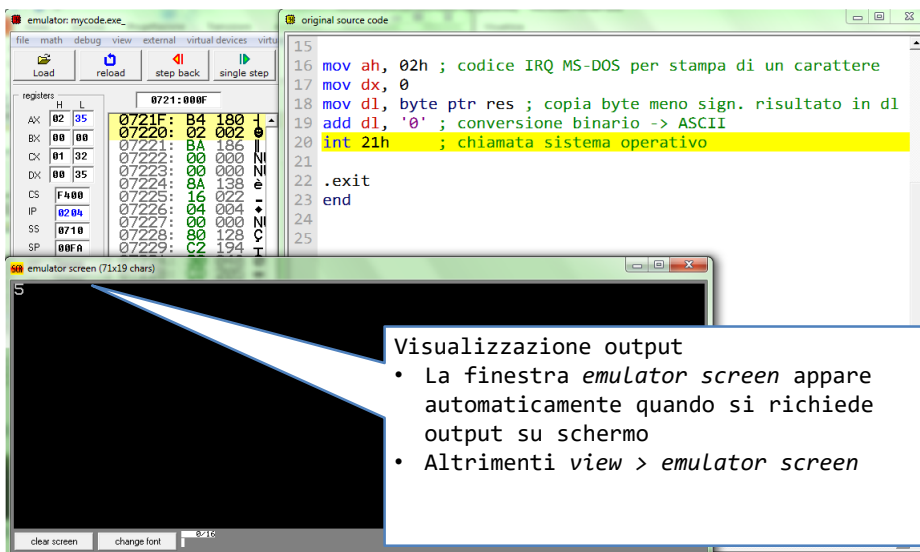
Visualizzazione della memoria

- `view > memory`
- È possibile scegliere l'indirizzo di partenza da visualizzare (segmento:offset)
- È possibile visualizzare e modificare il contenuto della memoria (doppio clic sulla cella)

The screenshot shows the 'emulator: mycode.exe' window. The 'view' menu is open, and 'memory' is selected. The 'Random Access Memory' window is displayed, showing a table of memory addresses and their contents. The address 0720:0000 is highlighted.

Address	Hex	ASCII
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 00 00 00 00 00 00 00 00	..Ari..U..
0720:0020	00 4C CD 21 90 90 90 90 90 90 90 90 90 90 90	..=====
0720:0030	90 90 90 90 90 90 90 90 90 90 90 90 90 90	..=====
0720:0040	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	..=====
0720:0060	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	..=====

Emulazione e debug [cont.]



Visualizzazione output

- La finestra *emulator screen* appare automaticamente quando si richiede output su schermo
- Altrimenti `view > emulator screen`

The screenshot shows the 'emulator: mycode.exe' window. The 'original source code' window is displayed, showing assembly code. The instruction `int 21h` is highlighted. The 'emulator screen' window is also visible, showing the output of the program.

```

15
16 mov ah, 02h ; codice IRQ MS-DOS per stampa di un carattere
17 mov dx, 0
18 mov dl, byte ptr res ; copia byte meno sign. risultato in dl
19 add dl, '0' ; conversione binario -> ASCII
20 int 21h ; chiamata sistema operativo
21
22 .exit
23 end
24
25

```

Assembler 8086 e Linux

- Esistono differenze rispetto ad ambiente DOS/Win
 - INT 21h → INT 0x80 (differenti funzioni per il Kernel Linux)
- Per Linux esistono vari Assemblatori/Linker
 - As86, Nasm, Yasm, Fasm, ...
 - Differiscono per direttive per il compilatore e per sintassi di alcune istruzioni
- Consiglio compilatore e debugger: NASM + GDB
 - Sintassi simile a MASM
 - HOWTO: <http://www.csee.umbc.edu/help/nasm/nasm.shtml>
- Alternativa: macchina virtuale DOS/Windows e EMU8086.

Esercizio

- Si prendano gli esempi di codice presentati nell' "Introduzione all'Assembler 8086" (assem_00.pdf). Si richiede di inserire tali esempi di codice in EMU8086, compilarli, eseguirli e analizzarne il comportamento in modalità di debug.