Introduzione alla Programmazione Assembly

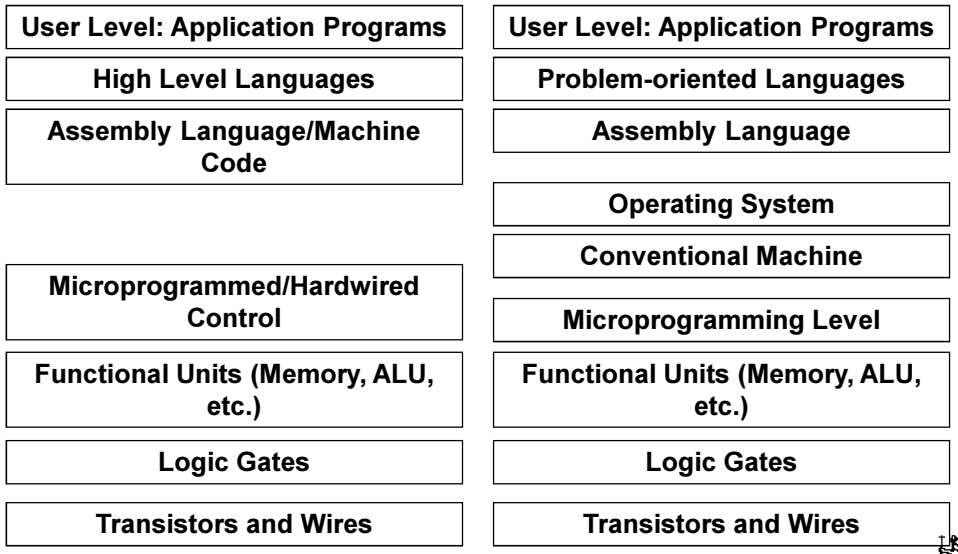
Nuovo Corso di Calcolatori Elettronici I

Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli "Federico II"

Roadmap

- Il livello Assembly
- Ciclo di sviluppo
- Esempi di codice sorgente
- Modello di programmazione del processore MC68K
- L'ambiente di riferimento per il corso
  - L'editor
  - L'assemblatore
  - Il simulatore ASIM

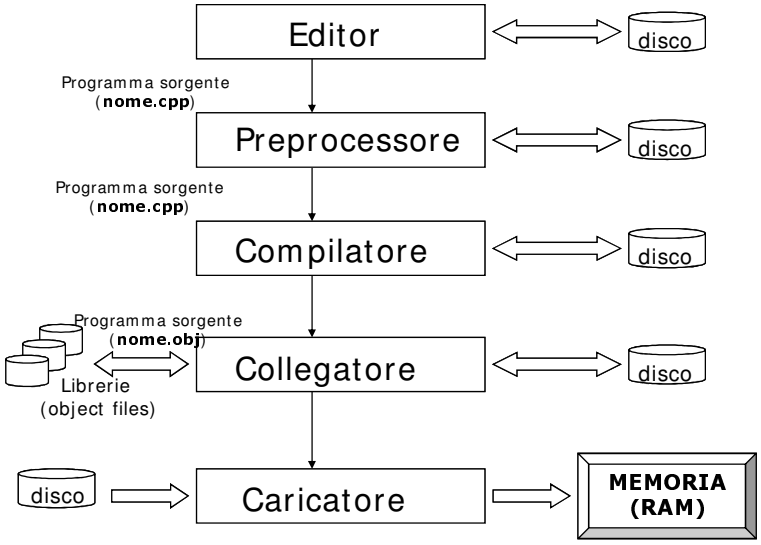
Livelli di macchine



Il linguaggio Assembly

- È funzionalmente equivalente al linguaggio macchina, ma usa "nomi" più intuitivi (mnemonics).
- Definisce l'Instruction Set Architecture (ISA) della macchina.
- Un compilatore traduce un linguaggio di alto livello, che è indipendente dall'architettura, in linguaggio assembly, che è dipendente dall'architettura.
- Un assemblatore traduce programmi in linguaggio assembly in codice binario eseguibile.
- Nel caso di linguaggi compilati (es. C) il codice binario viene eseguito direttamente dalla macchina target.
- Nel caso di linguaggi interpretati (es. Java) il bytecode viene interpretato dalla Java Virtual Machine, che – in questo senso – è al livello Assembly language.

Ciclo di Sviluppo di un programma in C++



Fase 1

preparazione del testo origine

… il testo origine viene digitato mediante un editor e viene memorizzato in un file con estensione ***cpp***

Fase 2

precompilazione

… il testo origine (sorgente) contenuto nel file " ***.cpp***" viene elaborato da un programma detto preprocessore (o precompilatore) che sostanzialmente esegue l'espansione del codice (inclusioni, macro, etc.)

Fase 3

compilazione delle unità

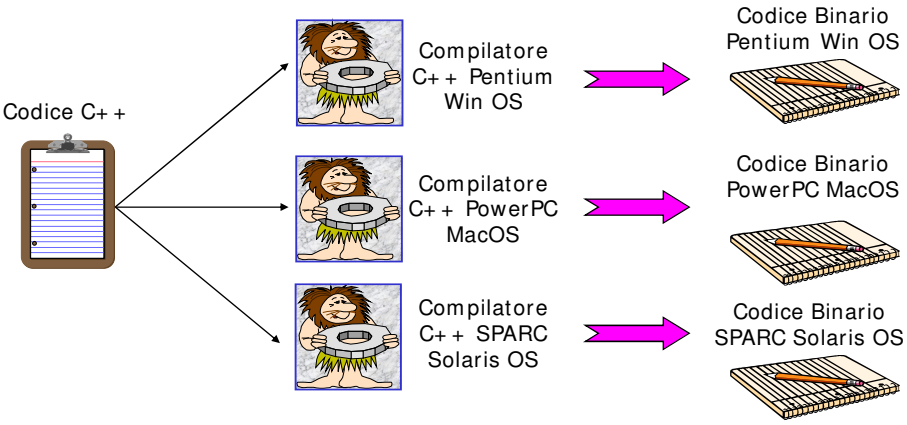
… le unità del programma vengono compilate mediante l'attivazione del compilatore; il testo origine verrà trasformato in testo **oggetto** e memorizzato in un file con estensione ***.obj***

Fase 4

collegamento (linkage)

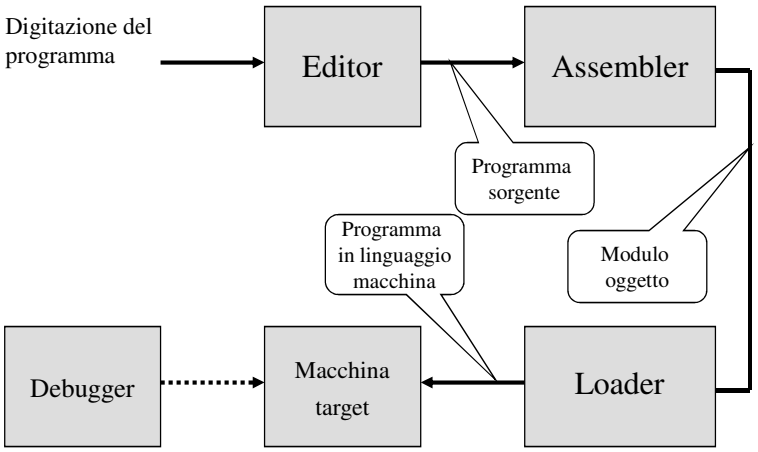
… i diversi moduli oggetto costituenti il programma eseguibile vengono collazionati fra loro (*p1.obj,…, pn.obj*) e con il **supporto al tempo di esecuzione** mediante un modulo collegatore (***linker***). Il tutto viene memorizzato in un file che costituisce il programma eseguibile (*p.exe*)

Ciclo di sviluppo di un programma in C++



@@@ I programmi C++ (compilati) funzionano su tutti i sistemi di una determinata architettura

Ciclo di sviluppo di un programma in ASM



Esempio – Assembly X86 a 32 bit

```
DES_std_crypt:
        movl 4(%esp),%edx
        pushl %ebx
        movl DES_count,%ecx
        xorl %ebx,%ebx
        movq (%edx),K1
        movq 32(%edx),K2
        movq K1,tmp1
        movq 8(%edx),K3
        movq 16(%edx),K4
        DES_copy(24, 40)
        …
        DES_copy(112, 120)
        movq DES_IV,R
        xorl %edx,%edx
        movq DES_IV+8,L
DES_loop:
        …
```


Esempio – Assembly Alpha

```
DES_std_crypt:
        ldgp $29,0($27)
DES_std_crypt..ng:
        subq $30,56,$30
        lda tmp1,DES_IV
        lda tmp2,DES_count
        lda SPE,DES_SPE_F
        ldq R,0(tmp1)
        ldq L,8(tmp1)
        ldq count,0(tmp2)
        ldq K1,0(kp)
        ldq K2,8(kp)
        ldq K3,16(kp)
        ldq K4,24(kp)
        xor K1,R,D
        ldq K5,32(kp)
        ldq K6,40(kp)
        ldq K7,48(kp)
        …
        ldq K8,56(kp)
        stq K9,0($30)
        stq K10,8($30)
        stq K11,16($30)
        stq K12,24($30)
        stq K13,32($30)
        stq K14,40($30)
        stq K15,48($30)
        ldq K9,64(kp)
        ldq K10,72(kp)
        ldq K11,80(kp)
        ldq K12,88(kp)
        ldq K13,96(kp)
        ldq K14,104(kp)
        ldq K15,112(kp)
        ldq K16,120(kp)
DES_loop:
        DES_2_ROUNDS(K2, K3)
        …
```

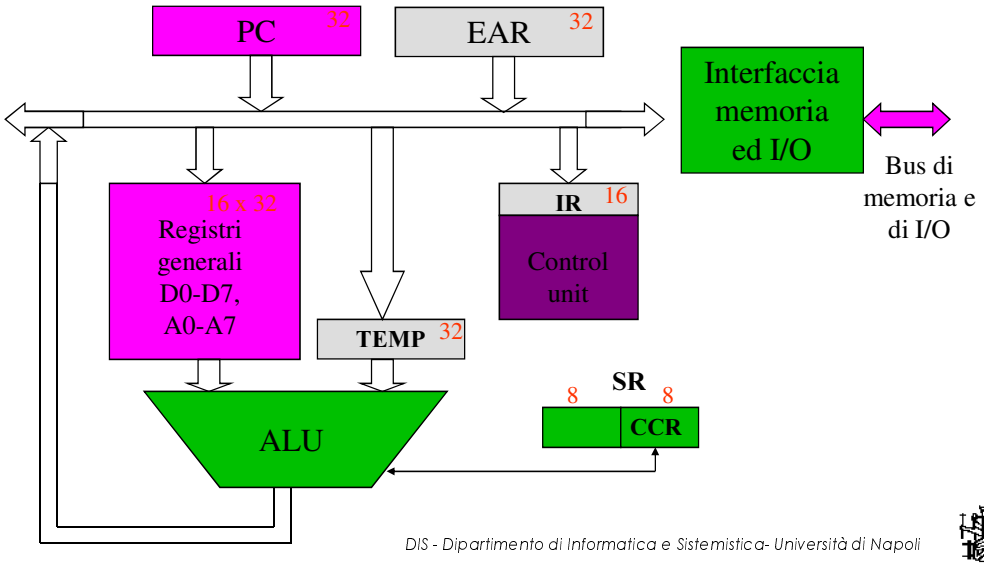

Esempio – Assembly Sparc

```
DES_std_crypt:
        …
        save %sp,-120,%sp
        st %i7,[%fp-24]
        sethi %hi(DES_SPE_L),SPE_L_0
        sethi %hi(DES_SPE_L+0x400),SPE_L_4
        add SPE_L_0,0x808,SPE_H_0
        …
        ldd [kp],D1
        ldd [SPE_L_4+0xC08],R1
        …
        ld  [SPE_L_4+0xC18],count
DES_loop:
        DES_2_ROUNDS(kp)
        …
        std R1,[out]
        std L1,[out+8]
        ret
        restore
        …
```


Linguaggi Assembly

- Per una data macchina, esiste sempre **almeno** il linguaggio assembly definito dal costruttore
- In aggiunta, possono esistere linguaggi assembly forniti da terze parti
- Quando si definisce un linguaggio assembly
  - » Si ha libertà di scelta per quanto riguarda:
    - ♦ Gli mnemonics
    - ♦ Il formato delle linee del sorgente
    - ♦ I formati per specificare modi di indirizzamento, varianti delle istruzioni, costanti, label, pseudo-operatori, etc.
  - » Non si ha libertà di scelta per quanto riguarda:
    - ♦ L'effetto finale di ogni singola istruzione macchina

Architettura del processore MC 68000



Caratteristiche del processore MC68000 – 1/2

- Dati:
  - All'esterno:
    parola di 16 bit (16 pin per i dati)
  - All'interno:
    registri di 32 bit
- Indirizzi:
  - All'esterno:
    Di 24 bit (spazio di indirizzamento fisico 2^{24} = 16M)
    - 512 pagine (2^9) da 32K (2^{15})
  - All'interno:
    Di 32 bit

Caratteristiche del processore MC68000 – 2/2

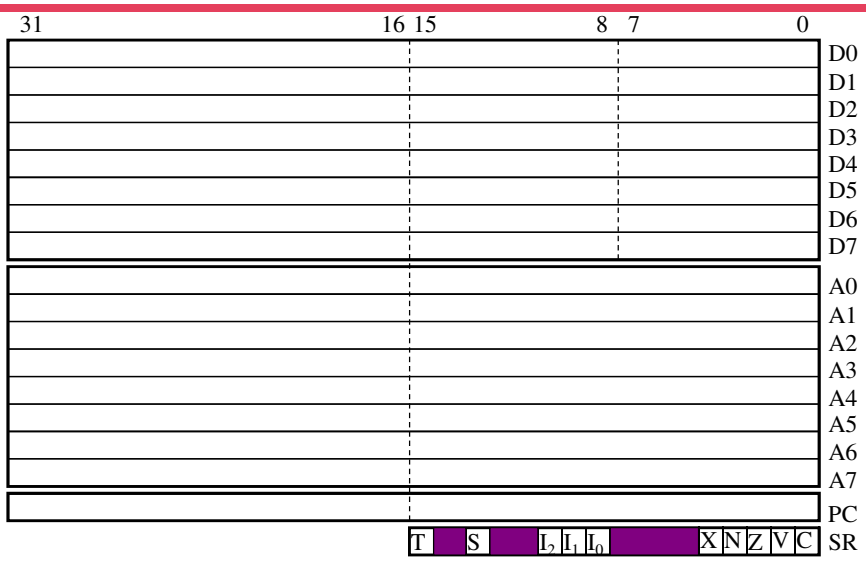
Parallelismo della memoria:

- Parole di 16 bit, ognuna costituita da due byte con indirizzi distinti (memoria byte addressable)

Convenzioni della memoria:

- Una parola deve essere allineata ad un indirizzo pari (even boundary)
- Convenzione big-endian

Modello di programmazione del MC68000



Formato del codice sorgente

- Una linea di codice sorgente Assembly è costituita da quattro campi:
  - LABEL
    - Stringa alfanumerica
    - Definisce un nome simbolico per il corrispondente indirizzo
  - OPCODE
    - Codice mnemonico o pseudo-operatore
    - Determina la generazione di un'istruzione in linguaggio macchina o la modifica del valore corrente del Program Location Counter
  - OPERANDS
    - Oggetti dell'azione specificata dall'OPCODE
    - Variano a seconda dell'OPCODE e del modo di indirizzamento
  - COMMENTS
    - Testo arbitrario inserito dal programmatore

etichetta	**cod. operativo**	**operando/i**	**commento**
(label)	*(opcode)*	*(operand(s))*	*(remark)*

Esempio di generazione di codice assemblato

```
PLC       contenuto              label    opcode     operands    comments
00000000                          1  * Programma per sommare i primi 17 interi
00000000                          2  *
00008000                          3           ORG        $8000
00008000  4279 00008032           4  START    CLR.W      SUM
00008006  3039 00008034           5           MOVE.W     ICNT,D0
0000800C  33C0 00008030           6  ALOOP    MOVE.W     D0,CNT
00008012  D079 00008032           7           ADD.W      SUM,D0
00008018  33C0 00008032           8           MOVE.W     D0,SUM
0000801E  3039 00008030           9           MOVE.W     CNT,D0
00008024  0640 FFFF              10           ADD.W      #-1,D0
00008028  66E2                   11           BNE        ALOOP
0000802A  4EF9 00008008          12           JMP        SYSA
00008030  =00008008              13  SYSA     EQU        $8008
00008030                         14  CNT      DS.W       1
00008032                         15  SUM      DS.W       1
00008034  =00000011              16  IVAL     EQU        17
00008034  0011                   17  ICNT     DC.W       IVAL
```

Symbol Table

ALOOP	800C	CNT	8030	IVAL	0011
START	8000	SUM	8032	ICNT	8034

Convenzioni

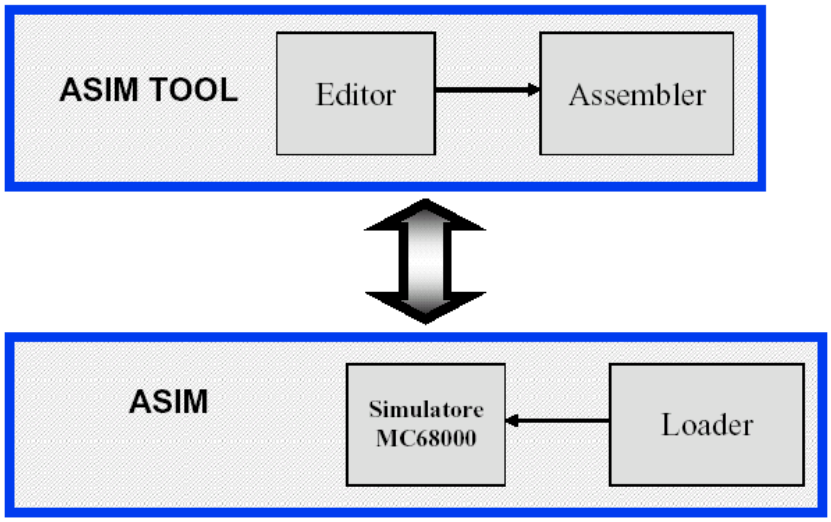
- Gli spazi bianchi tra i diversi campi fungono esclusivamente da separatori (vengono ignorati dall'assemblatore)
- Una linea che inizi con un asterisco (*) è una linea di commento
- Nelle espressioni assembly, gli argomenti di tipo numerico si intendono espressi
  - In notazione decimale, se non diversamente specificato
  - In notazione esadecimale, se preceduti dal simbolo "$"
- Nell'indicazione degli operandi, il simbolo "#" denota un indirizzamento immediato

Il Program Location Counter (PLC)

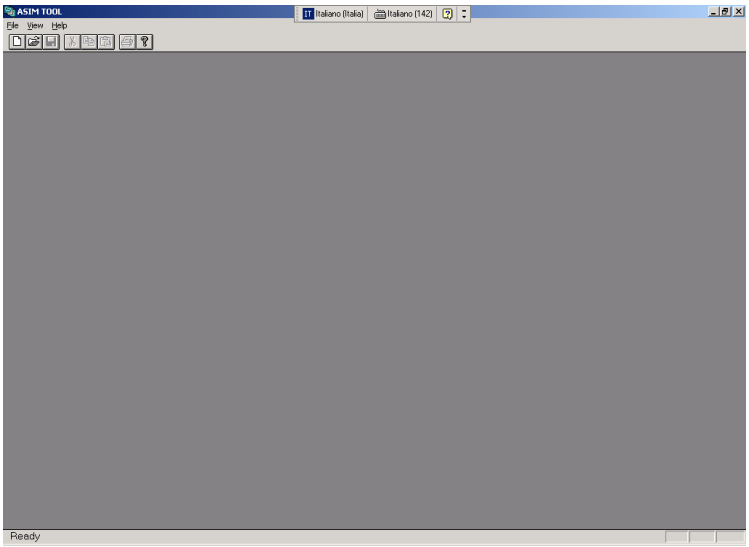
- **E' una variabile interna dell'assemblatore**
- **Punta alla locazione di memoria in cui andrà caricata – a run time – l'istruzione assemblata**
- **Viene inizializzato dallo pseudo-operatore "origin" (ORG)**
- **Durante il processo di assemblaggio, il suo valore è aggiornato sia in funzione degli operatori, sia in funzione degli pseudo-operatori**
- **E' possibile, all'interno di un programma, fare riferimento al suo valore corrente, mediante il simbolo "*"**

Ambiente di Sviluppo

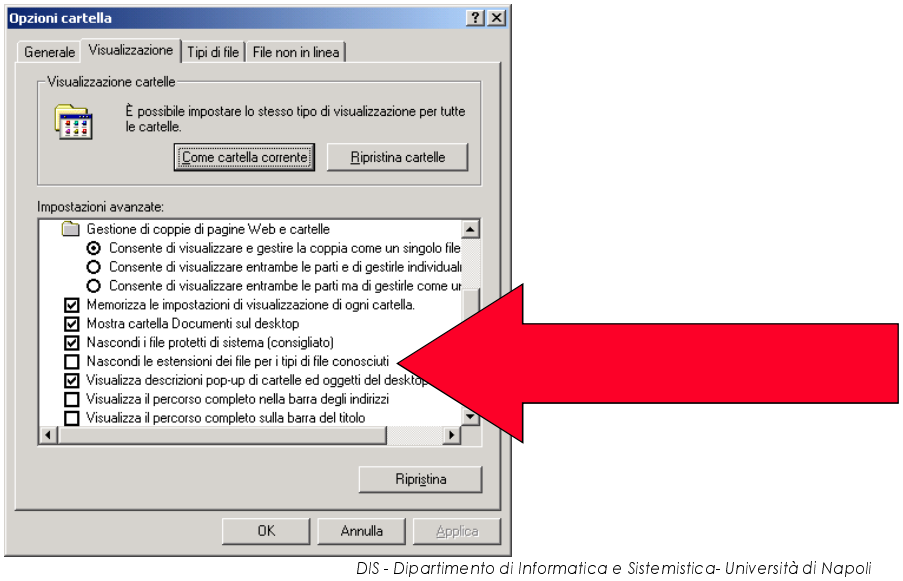
Ciclo di sviluppo di un programma in ASM



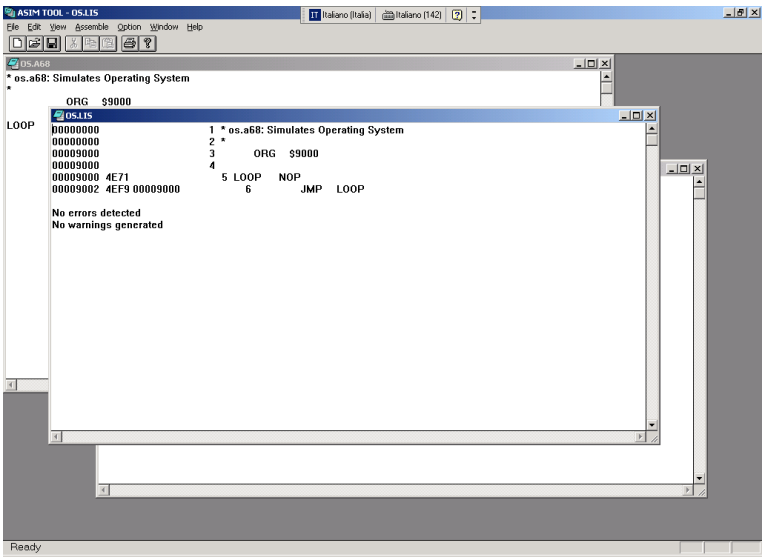
AsimTool – 1/ 3



AsimTool – 2/ 3



AsimTool – 3/ 3

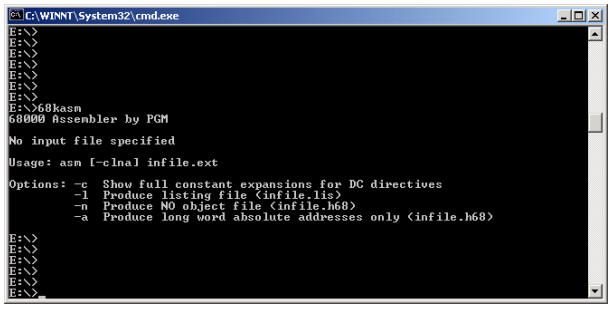


AsimTool

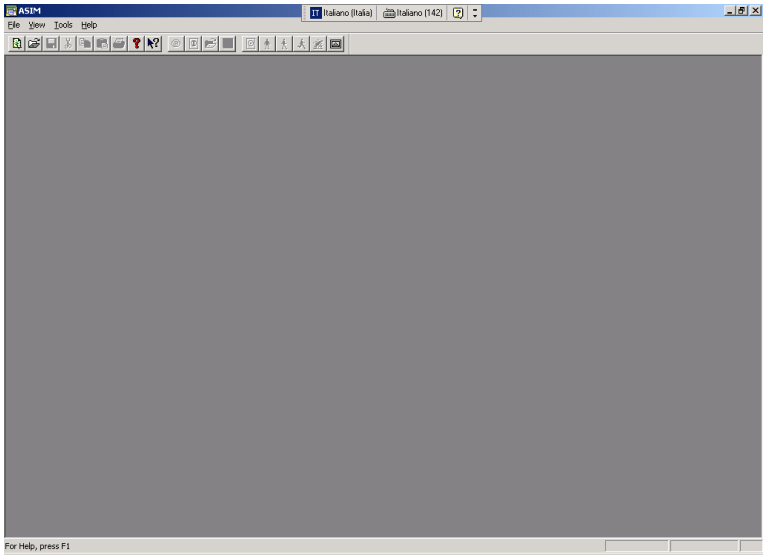
- Produce il file .lis che può essere caricato in ASIM
- Dopo aver assemblato il file sorgente, ASIMTOOL mostra il file .lis prodotto

L'assemblatore

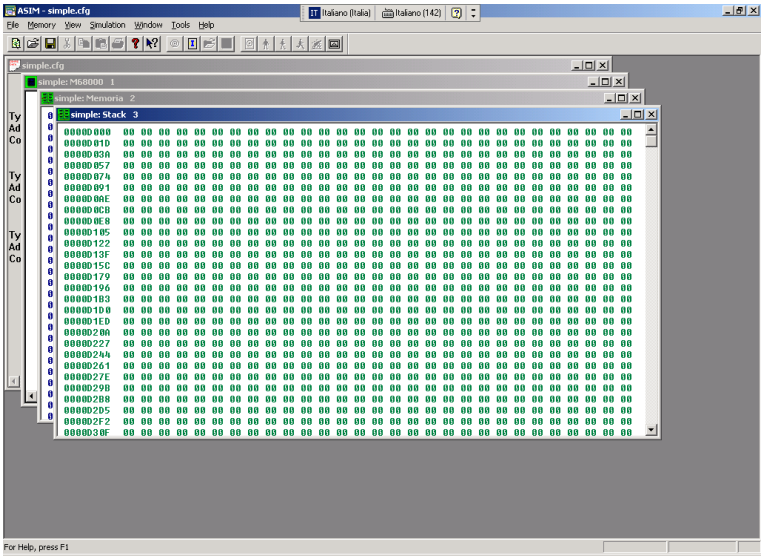
- Genera codice non rilocabile
- Si può lanciare:
  - da AsimTool
    - Assemble
  - da linea di comando
    - 68kasm.exe -l srcfile.asm



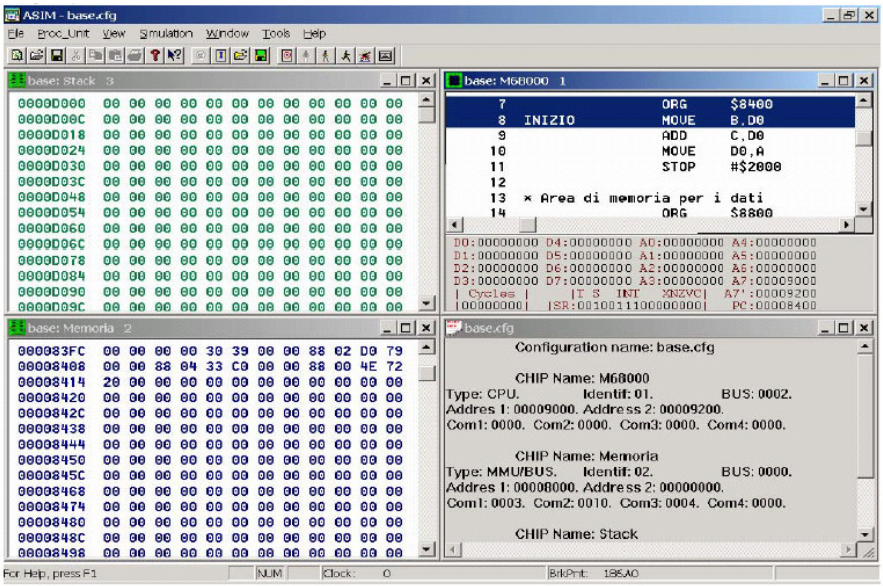
Il Simulatore ASIM – 1/ 3



Il Simulatore ASIM – 2/ 3



Il Simulatore ASIM – 3/ 3



Toolchain

Istruzioni per l'uso del simulatore ASIM

- 1 - Copiare asim.exe
- 2 - Lanciarlo
- 3 - Aprire il file di setup "base.cfg"(menu File-Open)
- 4 - Esplodere i componenti (tasto del "tiro al bersaglio")
- 5 - Resettare la simulazione (menu Simulation-OnReset)
- 6 - Evidenziare la finestra MC68000
- 7 - Caricare il programma da eseguire (filename.lis)(menu ProcUnit-LoadAssembler)
- 8 - Inizializzare il PC all'entry point (ricavabile dal file filename.lis)
- 9 - Procedere step-by-step

Istruzioni per l'assemblaggio dei programmi

- 1 - Eseguire il comando:
- 68kasm.exe -l filename.a68 (assembla.bat)
- 2 - Verificare che siano stati generati i file filename.h68 e filename.lis