

Corso di Calcolatori Elettronici I

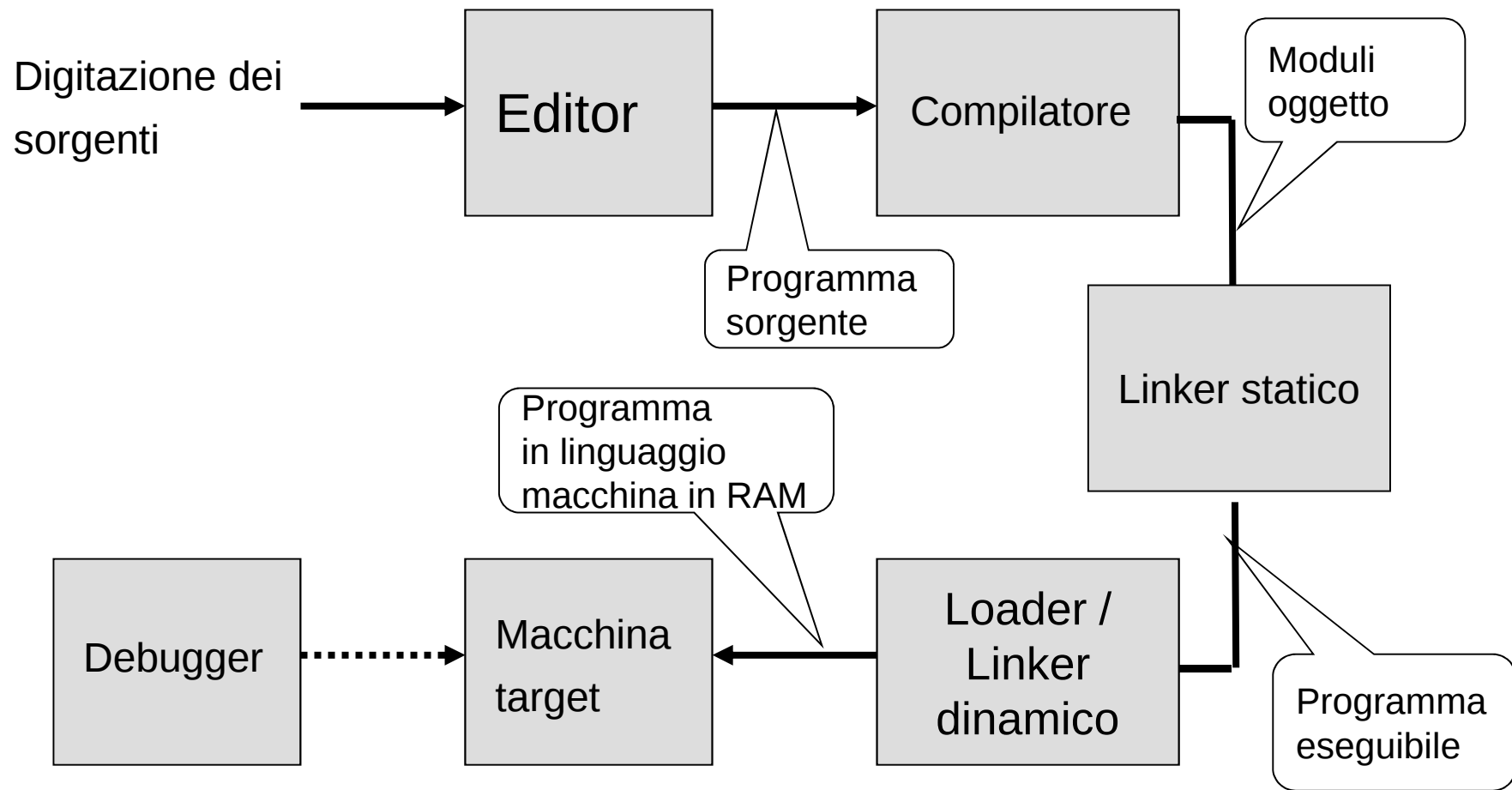
Introduzione al linguaggio assembly

Prof. Roberto Canonico

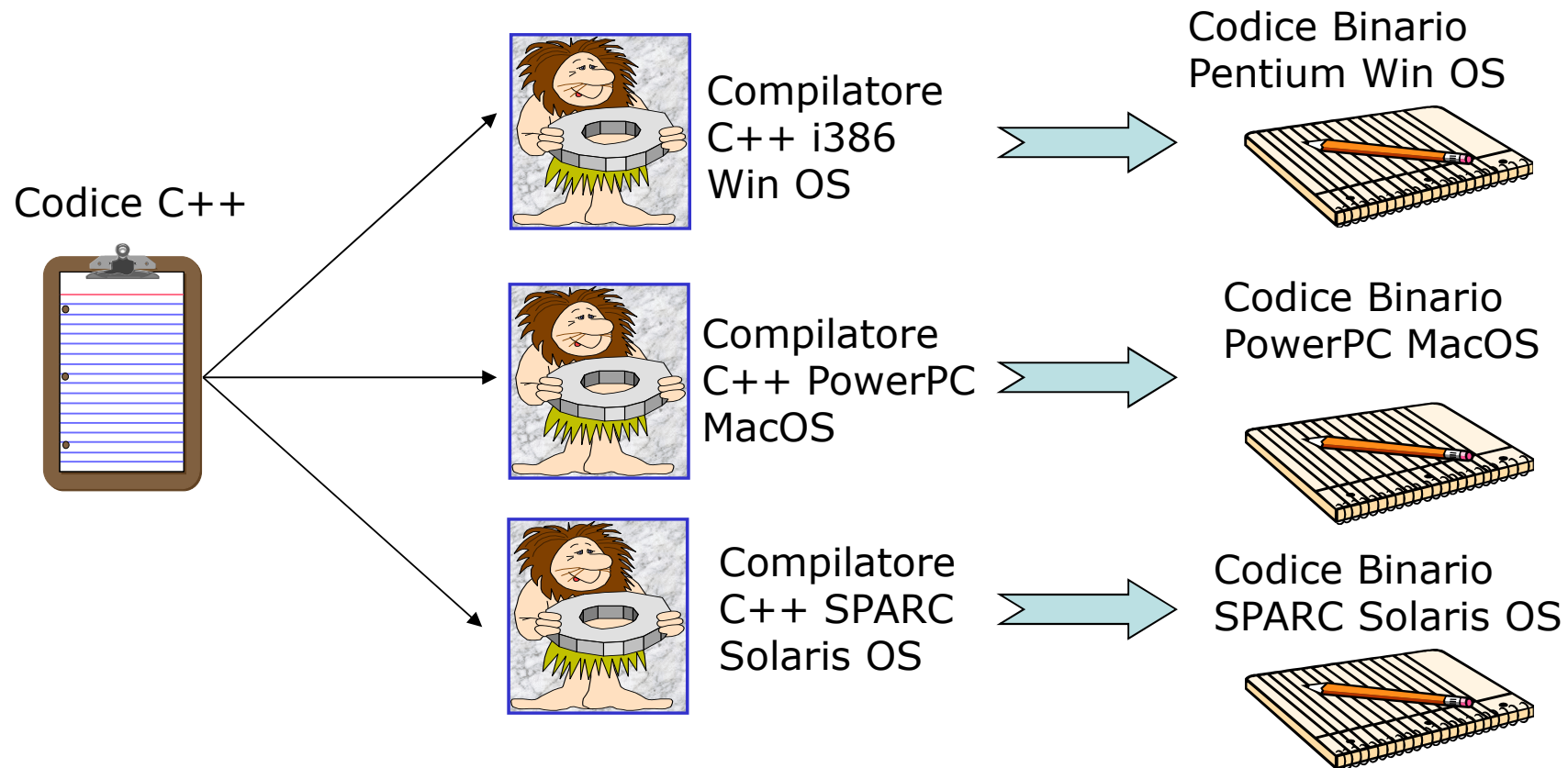


Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica
e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica
Corso di Laurea in Ingegneria dell'Automazione

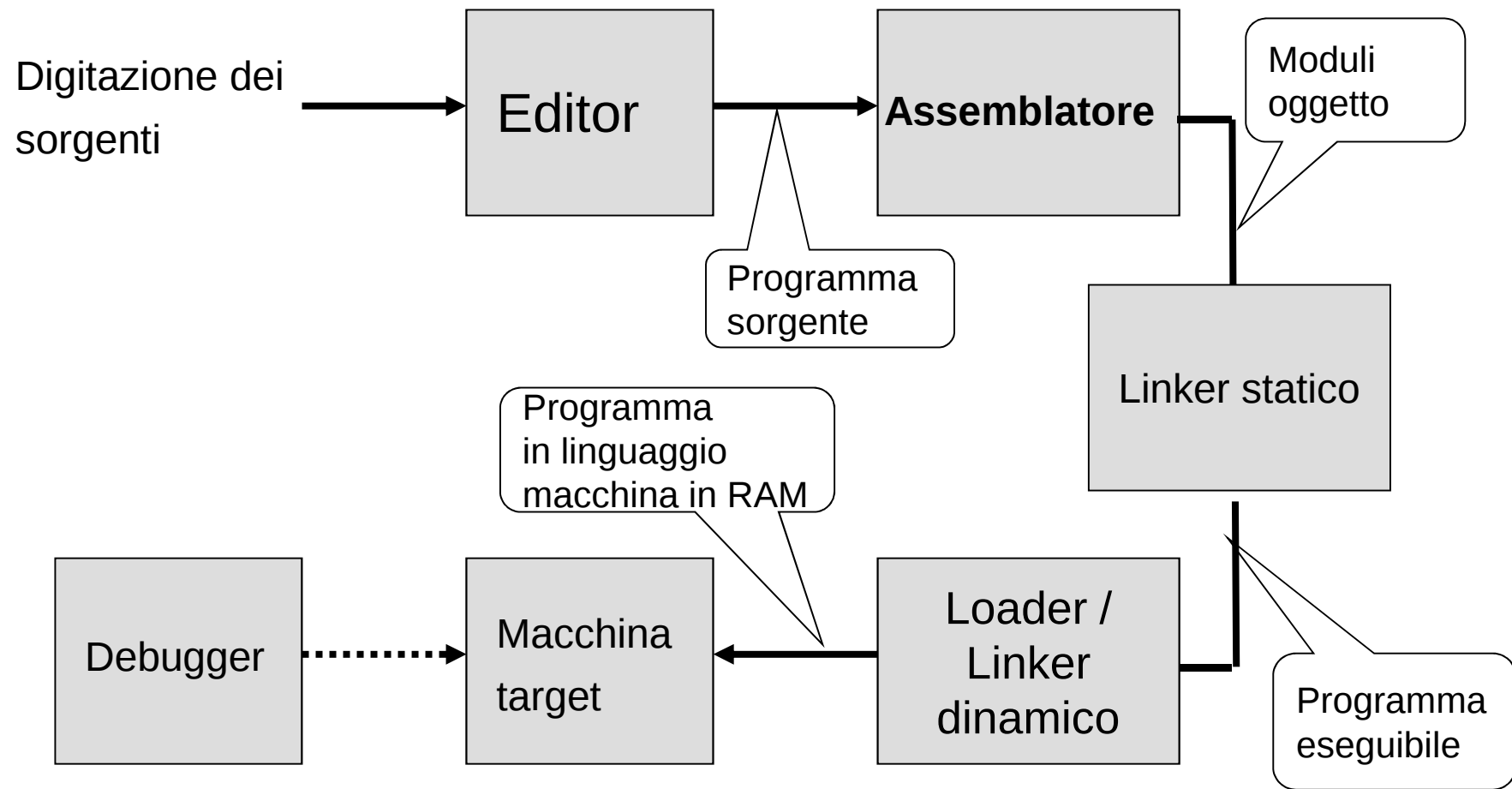
Ciclo di sviluppo/esecuzione per programmi in linguaggio di alto livello



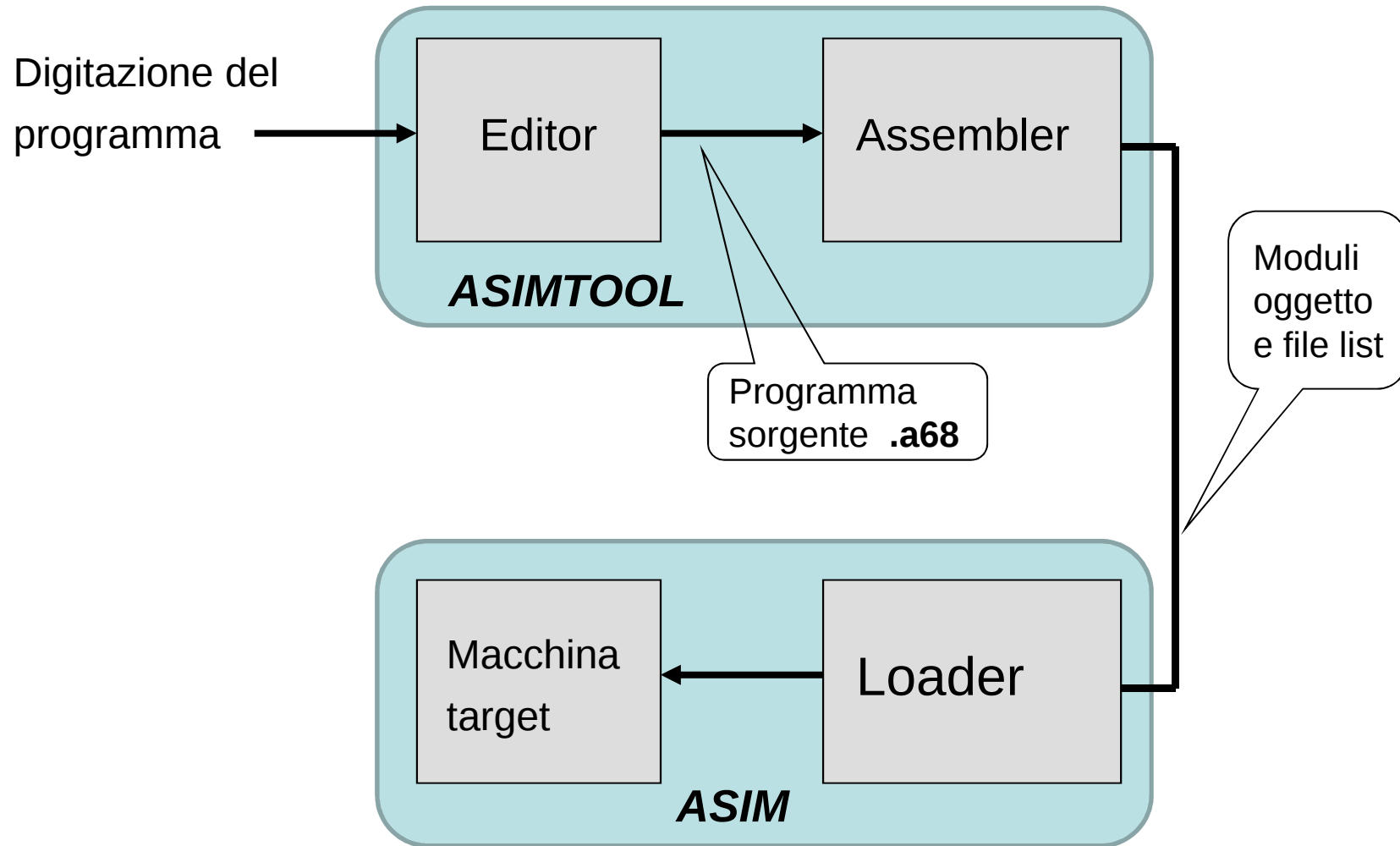
Linguaggi e dipendenza dalla piattaforma di esecuzione



Ciclo di sviluppo/esecuzione per programmi in linguaggio assembly



Ciclo di sviluppo semplificato di programmi assembly MC68000 nel sistema didattico ASIM



Assembly:

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
 - OP CODE
 - 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

Assembly: caratteristiche generali

- Di regola, una linea di codice assembly corrisponde ad una istruzione I/m
 - Eccezioni:
 - Macro: 1 linea assembler → n istruzioni I/m
 - Pseudo istruzioni: 1 linea assembler → 0 istr. I/m
 - Variabili interamente gestite dal programmatore
 - Allocazione: memoria o registri CPU
 - No dichiarazione
-

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
-

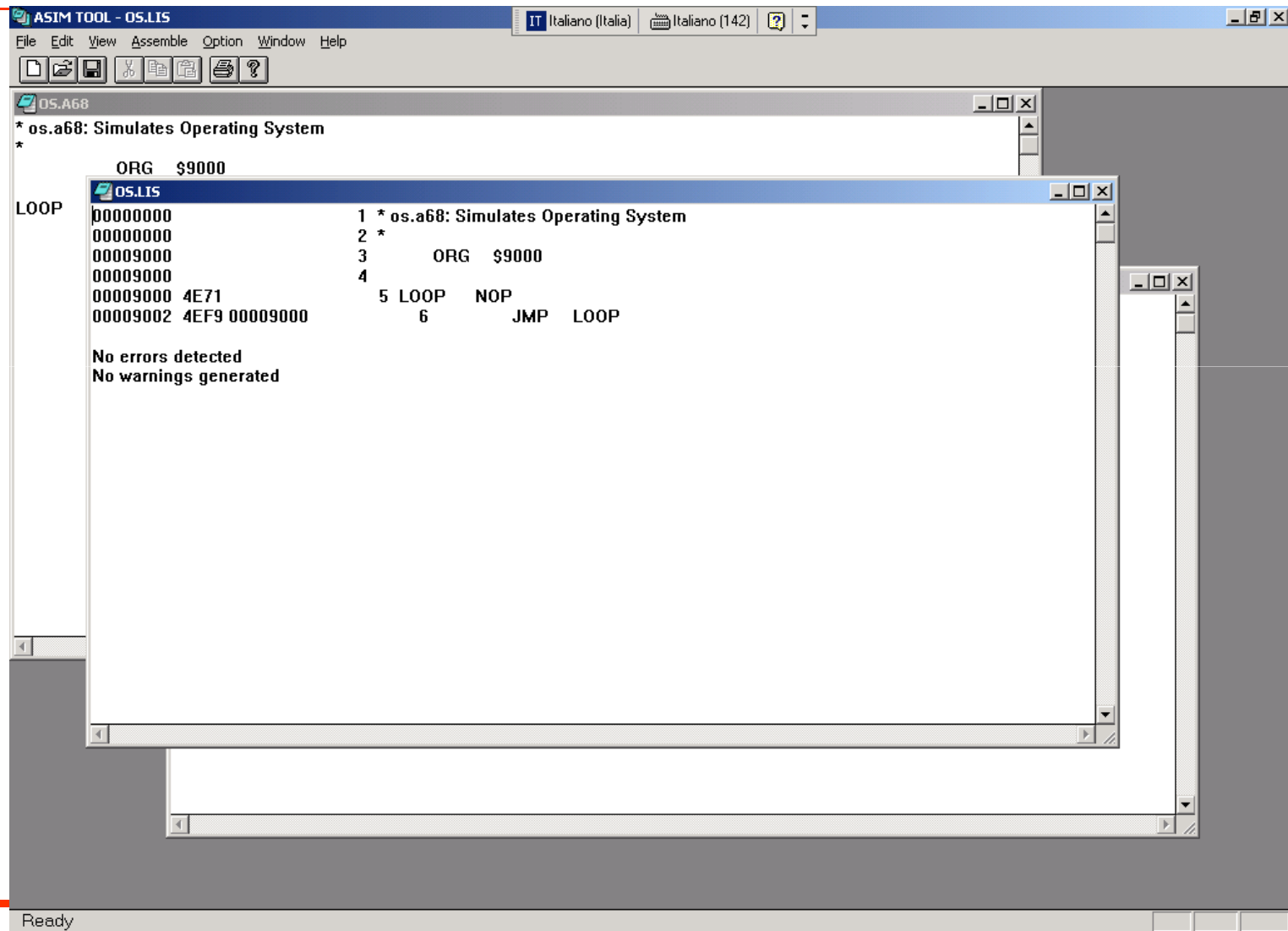
Convenzioni

- 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
-

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 “*”
-

AsimTool



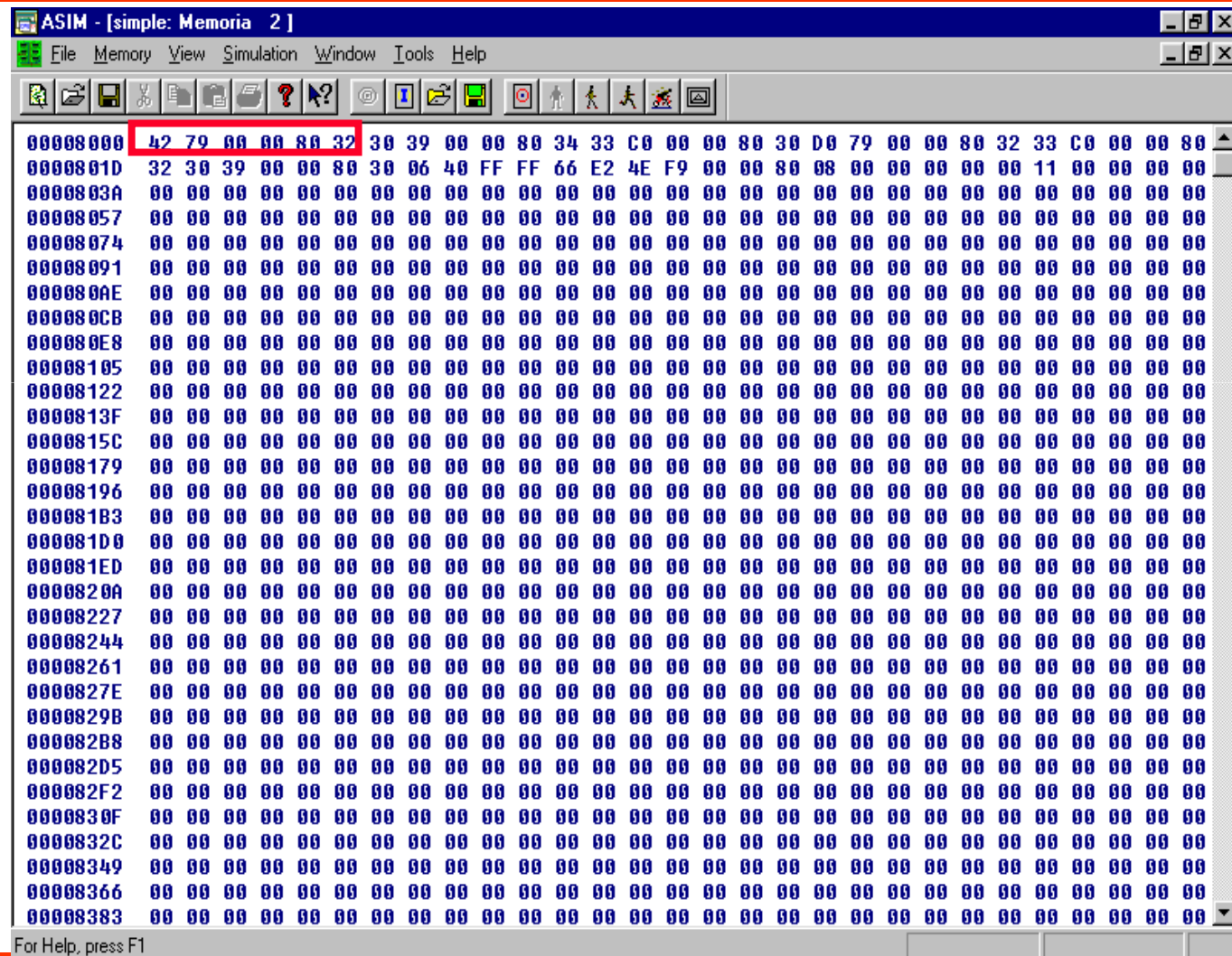
AsimTool: esempio di file list

PLC	contenuto	label	opcode	operands	comments
00000000		1	* Somma 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

ASIM: programma caricato in memoria



Pseudo-operatori

- NON sono istruzioni eseguite dal processore
 - sono direttive che regolano il processo di traduzione del programma assembler in programma eseguibile
 - Lo pseudo-operatore ORG
 - Viene usato per inizializzare il Program Location Counter (PLC), ovvero per indicare a quale indirizzo sarà posta la successiva sezione di codice o dati
 - **Esempio:** ORG \$8100
 - Lo pseudo-operatore END
 - Viene usato per terminare il processo di assemblaggio ed impostare l'entry point (prima istruzione da eseguire) nel programma
 - **Esempio:** END TARGETLAB
-

Pseudo-operatori

- Lo pseudo-operatore DS
 - Viene usato per incrementare il Program Location Counter (PLC), in modo da riservare spazio di memoria per una variabile
 - **Esempio:** LABEL DS.W NUMSKIPS
 - Lo pseudo-operatore DC
 - Viene usato per inizializzare il valore di una variabile
 - **Esempio:** LABEL DC.W VALUE
 - Lo pseudo-operatore EQU
 - Viene usato per definire una costante usata nel sorgente assembler
 - **Esempio:** LABEL EQU VALUE
-

Etichette (label)

- Sono stringhe di testo arbitrarie (opzionali) anteposte ad una istruzione o ad un dato all'interno del programma assembler
- Servono a riferirsi al particolare indirizzo che contiene quella istruzione o dato
 - usati per gestire i salti
 - usati per gestire variabili (manipolate nel programma assembler attraverso le loro etichette in maniera simile alle variabili di un linguaggio di programmazione di alto livello)
- Ad esempio:
 - ALOOP è un'etichetta usata per riferirsi all'istruzione MOVE, SUM è una etichetta usata per gestire una variabile, mentre IVAL è una costante

ALoop	MOVE .W	D0, CNT
	ADD .W	SUM, D0
...	...	...
SUM	DS .W	1
IVAL	EQU	17
...	...	...

Esempio assembly MC68000 - Moltiplicazione di due interi

* Programma per moltiplicare MCND e MPY

*

ORG \$8000

*

MULT	CLR.W	D0	D0 accumula il risultato
	MOVE.W	MPY,D1	D1 e' il contatore di ciclo
	BEQ	DONE	Se il contatore e' zero e' finito
LOOP	ADD.W	MCND,D0	Aggiunge MCND al prodotto parziale
	ADD.W	#-1,D1	Decrementa il contatore
	BNE	LOOP	e ripete il giro
DONE	MOVE.W	D0,PROD	Salva il risultato
PROD	DS.W	1	Riserva spazio di memoria per PROD
MPY	DC.W	3	Definisce il valore di MPY
MCND	DC.W	4	Definisce il valore di MCND
	END	MULT	
