CODIFICA PROGRAMMI
E
AMBIENTE DI PROGRAMMAZIONE

GERARCHIE DI LINGUAGGI DI PROGRAMMAZIONE

Linguaggi di macchina:

Ogni processore ha il proprio linguaggio macchina

È strettamente correlato alla progettazione dell'hardware

Linguaggi simbolici o assembler:

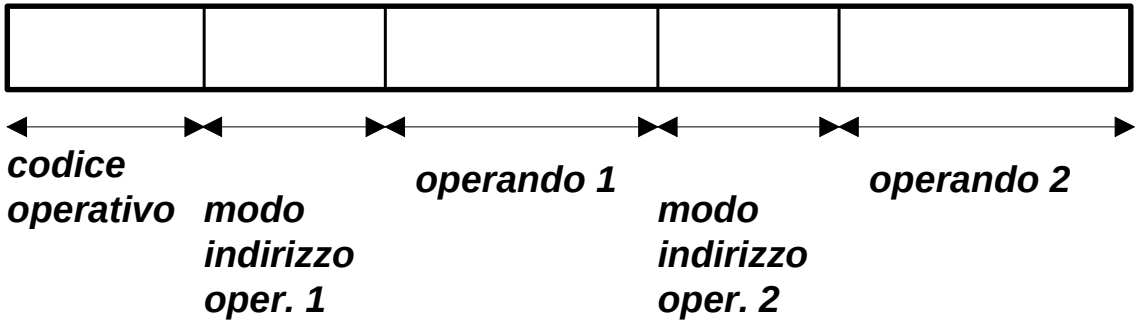
Linguaggio simbolico che fa corrispondere al codice binario del linguaggio macchina un nome simbolico

Linguaggi di alto livello:

denotano una minore dipendenza dalla macchina effettivamente usata

LINGUAGGI DI MACCHINA

IL FORMATO DELLE ISTRUZIONI



Il codice operativo è sempre presente ed identifica l'operazione da svolgere.

Gli operandi, che possono essere anche 3 o mancare del tutto, specificano la dislocazione (in memoria centrale o in un registro generale della CPU) dei dati cui l'operazione si riferisce. Se un dato è in un registro, l'operando indica il numero di tale registro. Se un dato è in memoria, l'operando ne indica l'indirizzo.

La lunghezza di un'istruzione in bit (l_{istr}) dipende, oltre che dalla lunghezza del codice operativo (in genere 8 bit), dal numero e dal modo d'indirizzamento dei dati-operandi.

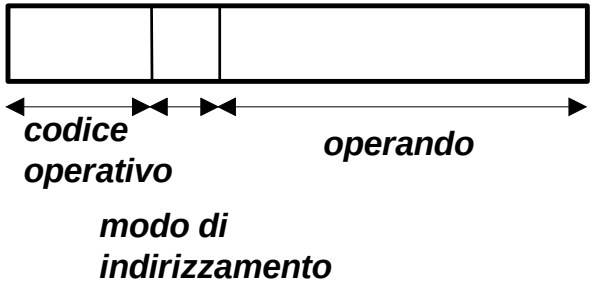
LINGUAGGI DI MACCHINA

IL SET DELLE ISTRUZIONI

L'insieme delle istruzioni eseguibili, rappresentate dai rispettivi codici operativi, viene detto set di istruzioni.

I MODI DI INDIRIZZAMENTO

- diretto, quando l'indirizzo indicato come operando è quello assoluto della parola di memoria interessata dall'operazione
- indiretto, quando l'indirizzo indicato come operando è quello di una parola di memoria che a sua volta contiene l'indirizzo assoluto della parola di memoria interessata dall'operazione. Può essere indirizzata una memoria più grande di quella indirizzabile dal solo operando
- relativo (tramite un registro indice), quando l'indirizzo della parola di memoria interessata dall'operazione si ottiene sommando il numero contenuto in un registro (registro indice o registro base) al valore dell'indirizzo-operando. Può essere indirizzata una memoria più grande di quella indirizzabile dal solo operando
- immediato, quando l'operando non indirizza la memoria, ma fornisce direttamente il valore interessato dall'operazione



LINGUAGGI DI MACCHINA

ESEMPIO DI ISTRUZIONI DI MACCHINA

Codice Operativo	*Operando*	*Significato*
00000000	indirizzo binario	carica operando in reg. A
00000001	indirizzo binario	carica operando in reg. B
00000010	indirizzo binario	scarica reg.A in operando
00000011	indirizzo binario	scarica reg.B in operando
00000100	indirizzo binario	leggi da I in operando
00000101	indirizzo binario	scrivi da operando in O
00000110		aggiungi reg. B a reg. A
00000111		sottrai reg. B da reg. A
00001000		moltiplica reg. A e reg.B
00001001		dividi reg.A per reg. B
00001010	indirizzo binario	salta a istr. in operando
00001011	indirizzo binario	c.s. se reg. A = 0 (*)
00001100		pausa di una istruzione
00001101		termina esecuzione

LINGUAGGI SIMBOLICI O ASSEMBLER

GLI OPERANDI SIMBOLICI

Ogni dato (costante o variabile) è identificato da un nome. Con tale nome viene etichettata la cella di memoria che contiene il dato.

I CODICI SIMBOLICI

Codice Operativo	*Simbolo*	*Operando*	*Significato*
00000000	LOADA	dato con nome	carica operando in reg. A
00000001	LOADB	dato con nome	carica operando in reg. B
00000010	STOREA	dato con nome	scarica reg.A in operando
00000011	STOREB	dato con nome	scarica reg.B in operando
00000100	READ	dato con nome	leggi da I in operando
00000101	WRITE	dato con nome	scrivi da operando in O
00000110	ADD		aggiungi reg. B a reg. A
00000111	DIF		sottrai reg. B da reg. A
00001000	MUL		moltiplica reg. A e reg.B
00001001	DIV		dividi reg.A per reg. B
00001010	JUMP	istruzione con nome	salta a istr. in operando
00001011	JUMPZ	istruzione con nome	c.s. se reg. A = 0 (*)
00001100	NOP		pausa di una istruzione
00001101	HALT		termina esecuzione

GLI OPERANDI SIMBOLICI

Ogni dato (costante o variabile) è identificato da un nome. Con tale nome viene etichettata la cella di memoria che contiene il dato.

Nome (etichetta o label)

Indirizzo | ***valore***

Le istruzioni possono essere identificate da un nome.

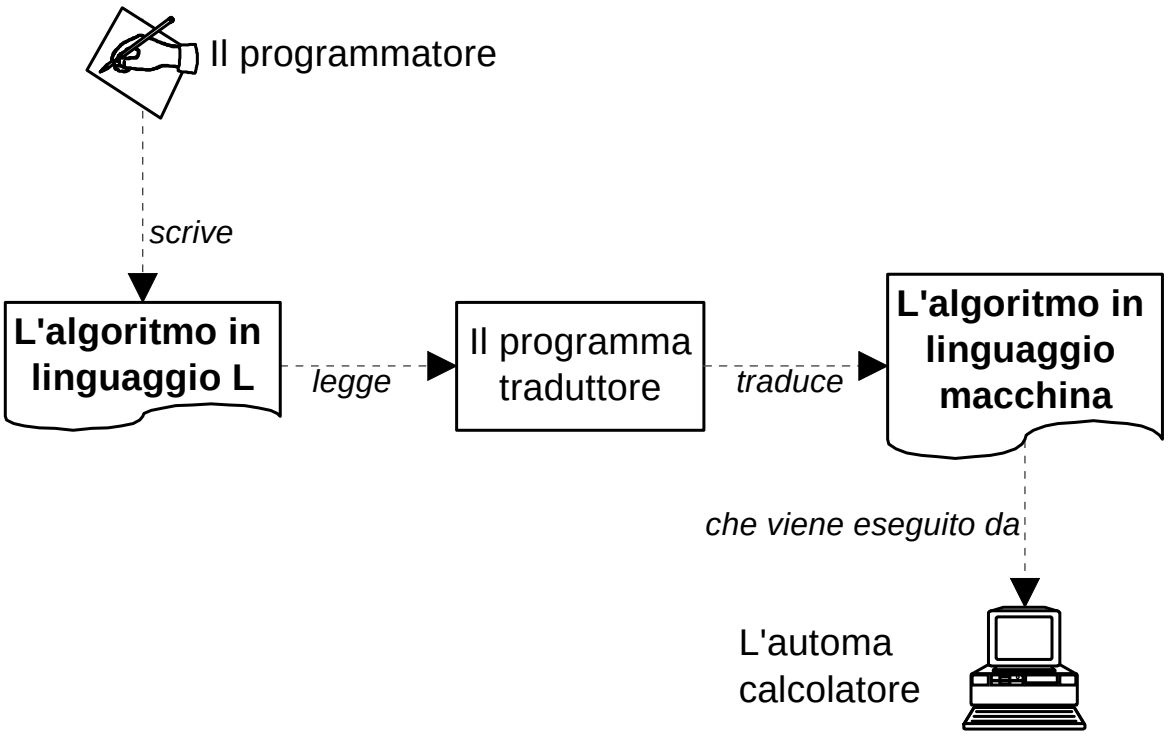
LINGUAGGI SIMBOLICI O ASSEMBLER

UN ESEMPIO DI PROGRAMMA ASSEMBLER

```
LOADA     UNO
          STOREA    N
CICLO1    READ      NUM
          LOADA     NUM
          JUMPZ     CICLO2
          LOADI     N
          STOREA    A(I)
          LOADA     N
          LOADB     UNO
          ADD
          STOREA    N
          JUMP      CICLO1
CICLO2    LOADA     N
          LOADB     UNO
          DIF
          JUMPZ     FINE
          STOREA    N
          LOADI     N
          LOADA     A(I)
          STOREA    NUM
          WRITE     NUM
          JUMP      CICLO2
FINE      HALT

UNO       1
N         INT
NUM       INT
A1        INT
A2        INT
.....
A10       INT
```

LINGUAGGI E TRADUTTORI



Il programma traduttore può:

- tradurre ed eseguire frase per frase (***interprete***)
- tradurre tutte le frasi e solo successivamente eseguire (***assemblatore o compilatore***)

Interpretazione

- Un ***Interprete*** è un programma che legge il programma sorgente e, per ogni istruzione
  1. Verifica la correttezza sintattica
  2. Effettua la traduzione nella corrispondente sequenza di istruzioni in linguaggio macchina
  3. Esegue direttamente la sequenza di istruzioni in linguaggio macchina

- SVANTAGGIO: Istruzioni eseguite più volte (es. ciclo), vengono verificate e tradotte più volte
- VANTAGGIO: Facile sviluppo e correzione dei programmi

Compilazione

- Un ***Compilatore*** è un programma che legge il programma sorgente e lo **traduce interamente** in un programma scritto in linguaggio macchina (programma oggetto)
  - Verifica la correttezza sintattica di ciascuna istruzione
  - Il programma oggetto è generato solo se non ci sono errori sintattici
  - La correttezza semantica è effettuata solo in fase di esecuzione

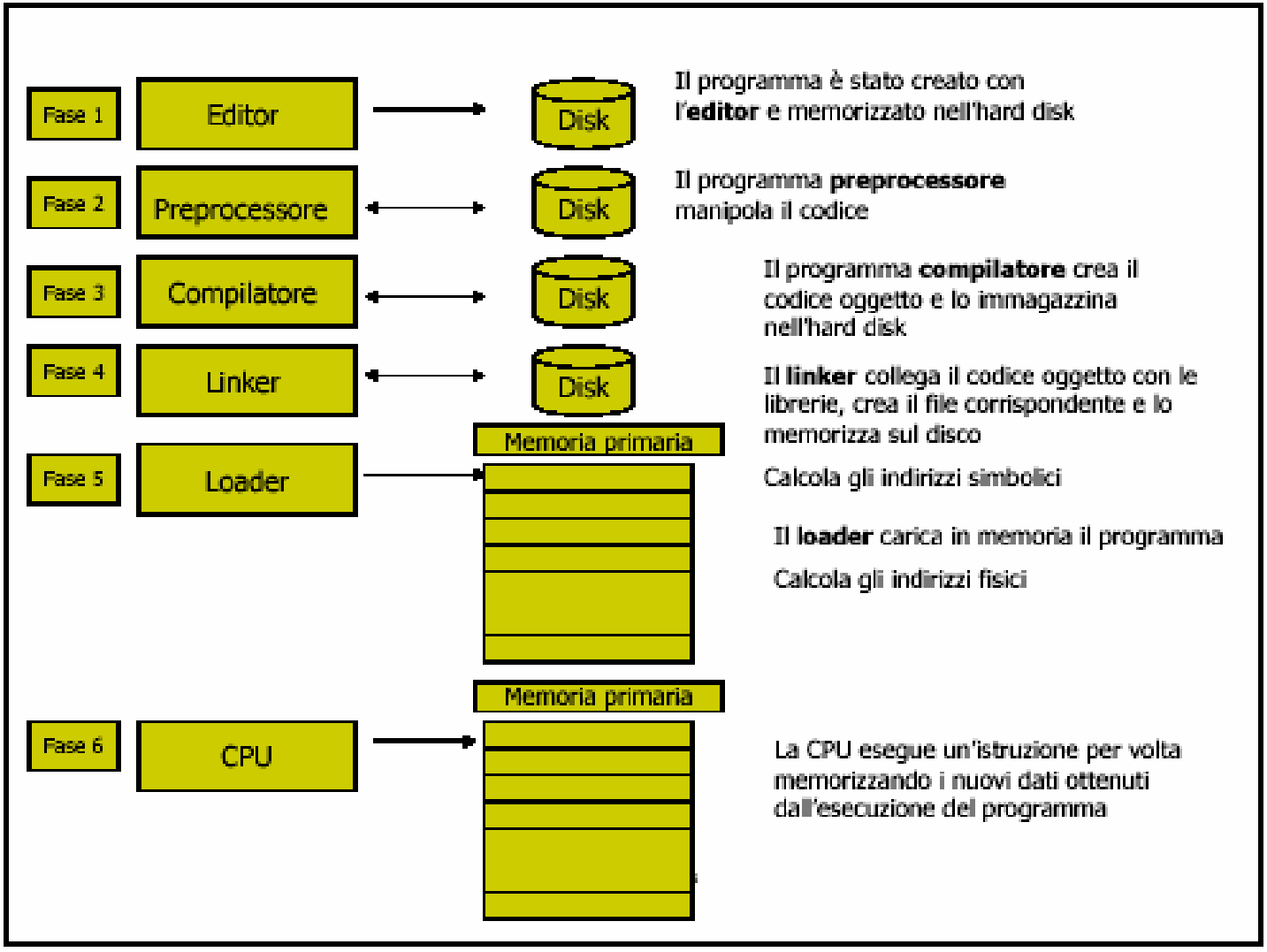
Interpretazione vs Compilazione

- Velocità di esecuzione:
  - Bassa per i linguaggi interpretati
  - Alta per i linguaggi compilati
- Facilità di messa a punto dei programmi:
  - Alta per i linguaggi interpretati
  - Bassa per i linguaggi compilati

L'AMBIENTE DI PROGRAMMAZIONE C

LE FASI

- Editazione
- Preelaborazione
- Compilazione
- Linking
- Caricamento
- Esecuzione



EDITAZIONE DEL PROGRAMMA

La prima fase consiste nella editazione del codice in un file

Il file dovrà terminare con l'estensione .c

Il programma che consente di eseguire questa fase è detto editor

Il programma editato è memorizzato in un dispositivo di memoria secondaria e viene detto programma sorgente (source program)

```
#include<stdio.h>
void main()
{ int n1,n2,sum;
printf("inserire un numero intero");
scanf("%d",&n1);
printf("inserire un numero intero");
scanf("%d",&n2);
sum=n1+n2;
}
```

Nell'esempio il source file è somma.c

PRECOMPILAZIONE DEL PROGRAMMA ED ESPANSIONE DELLE MACRO

Il programma che esegue la precompilazione è detto preprocessore e viene eseguito in maniera automatica prima che avvenga la compilazione.

Il preprocessore obbedisce a comandi detti direttive del preprocessore o macro

- ☞ Indicano che sul programma devono essere effettuate delle manipolazioni prima della compilazione
  - Inclusione di altri file in quello da compilare
  - Sostituzione di simboli speciali con un testo del programma

Le linee per il preprocessore di macro iniziano con il simbolo #
Ad esempio:

```
#define max 40
#include<stdio.h>
```

Prima della compilazione del programma ogni occorrenza di max è sostituita con il valore 40

La riga **`#include<stdio.h>`** viene sostituita dal contenuto del file stdio.h. Ciò consente al programma di usare la funzione di libreria **`Printf`**

LA LIBRERIA STANDARD DEL C

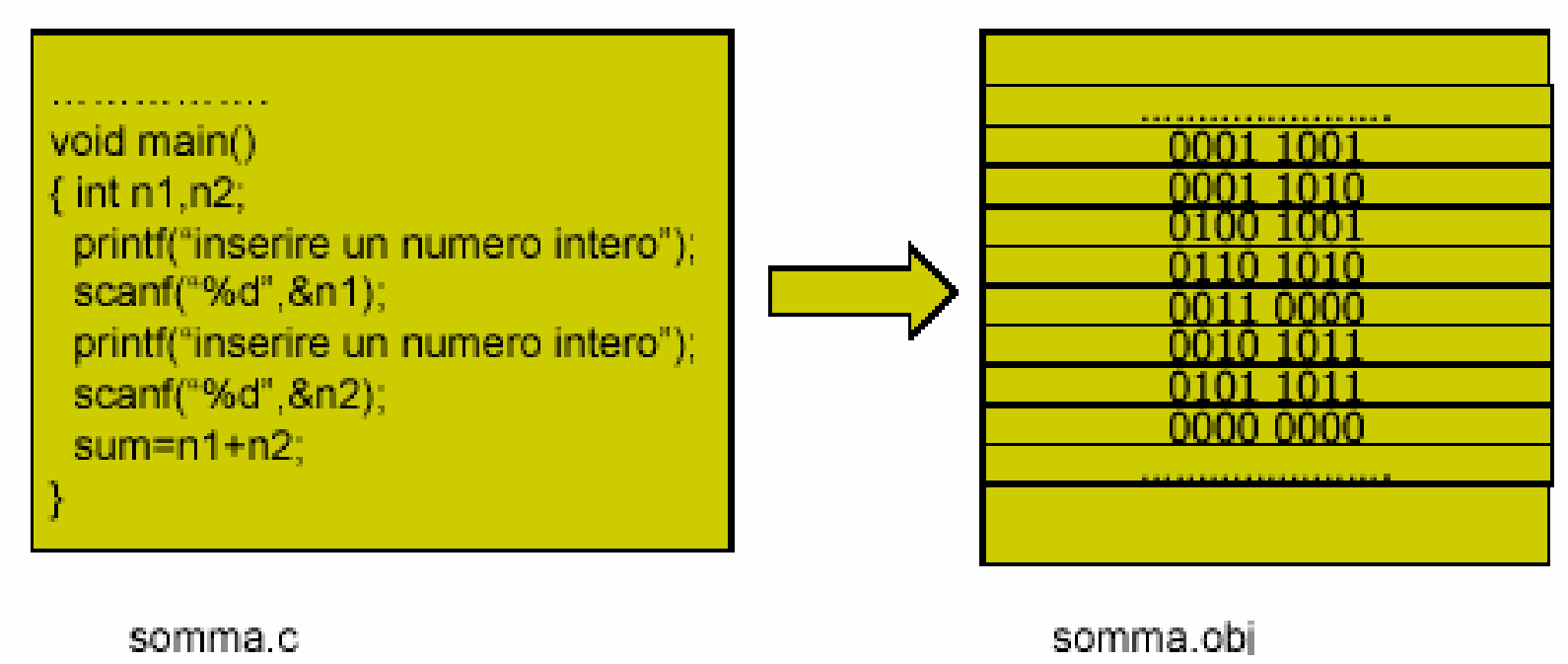
Comprende principalmente:

<stdef.h>	definizioni comuni
<assert.h>	diagnostica
<ctype.h>	gestione di caratteri
<local.h>	localizzazione
<math.h>	funzioni matematiche
<stdio.h>	I/O
<stdlib.h>	utilità generiche
<string.h>	gestione di stringhe

TRADUZIONE (COMPILAZIONE O INTERPRETAZIONE) DEL PROGRAMMA

Il traduttore (compilatore o interprete) trasforma le istruzioni del programma in istruzioni in linguaggio di macchina

In particolare, il compilatore traduce il programma C nel codice in linguaggio macchina, detto codice oggetto



CODICE OGGETTO

Un programma può essere suddiviso in parti separate, dette moduli. Ciascun modulo può essere compilato separatamente
La compilazione separata dei vari moduli genera moduli oggetto separati

Nel codice oggetto i nomi delle variabili definite nel modulo di programma (indirizzi simbolici) sono trasformati in indirizzi rilocabili

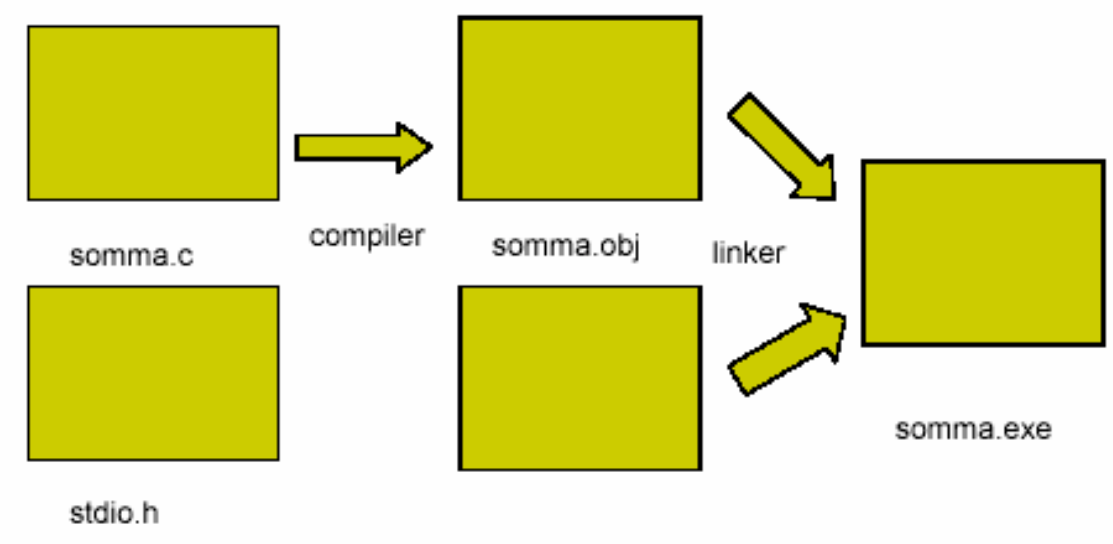
- espressi cioè in forma logica
- indipendenti dall'allocazione del programma in memoria
- calcolati come se il programma fosse caricato a partire dall'indirizzo zero

LINKING DEL PROGRAMMA

Il linking è eseguito da un programma detto linker, che:

- trasforma i vari moduli oggetto in un unico programma eseguibile
- collega il programma dell'utente a librerie di programmi di utilità disponibili nell'ambiente di programmazione di ciascun linguaggio. Infatti i programmi scritti in C contengono riferimenti a funzioni definite altrove:
  - ☞ nelle librerie standard
  - ☞ nelle librerie definite dal programmatore
- risolve riferimenti a variabili definite esternamente a ciascun modulo trasformando gli indirizzi simbolici in indirizzi rilocabili

Il linker collega il codice oggetto con quello delle funzioni mancanti eproduce il codice eseguibile o programma eseguibile



CARICAMENTO IN MEMORIA DEL PROGRAMMA

Prima di essere eseguito un programma deve essere caricato nella memoria

Il caricamento è effettuato da un programma detto loader

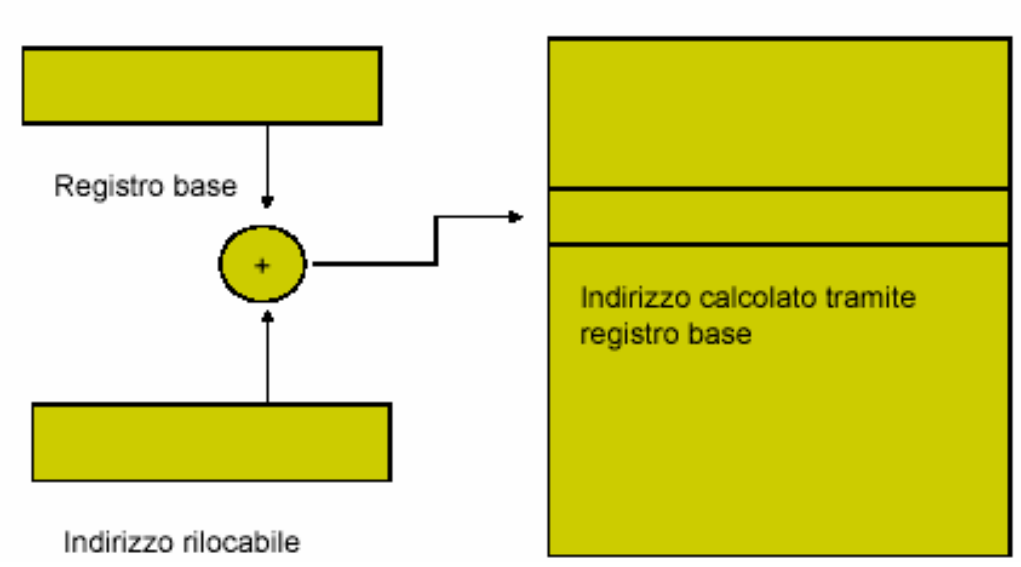
Il loader preleva dal disco il programma eseguibile e lo trasferisce nella memoria

Indirizzi di un programma

in formato assoluto: calcolati a partire da una specifica cella di memoria; si verifica se:

- ☞ la memoria è partizionata staticamente e i programmi vengono allocati alla memoria in modo rigido
- ☞ tutta la memoria è assegnata ad un unico programma utente

in formato rilocabile: calcolati come se il programma dovesse essere caricato in memoria a partire dalla cella zero



IL LOADER

Carica il programma eseguibile in memoria centrale

- Se il programma ha già indirizzi assoluti il loader deve caricare il programma in una specifica zona di memoria
- Se il programma oggetto ha istruzioni in formato rilocabile il loader effettua la rilocazione

  È necessario pertanto trasformare gli indirizzi logici presenti nei programmi in indirizzi fisici

  La rilocazione può essere:
  - Statica: avviene nel momento del caricamento in memoria modificando istruzione per istruzione gli indirizzi che compaiono nel programma
  - Dinamica: non comporta una modifica del codice: gli indirizzi sono calcolati utilizzando il registro base

ESECUZIONE DEL PROGRAMMA

L’esecuzione del programma eseguibile (in linguaggio macchina) è effettuata nella CPU