La Produzione dei Programmi

1 Il processo di Traduzione

- 1.1 compilazione
- 1.2 collegamento
- 1.3 caricamento

2. Il processo di Interpretazione

Introduzione

- La creazione di un programma è una attività complessa che va dalla scelta di un algoritmo alla sua realizzazione in un linguaggio di programmazione.
- Un linguaggio di programmazione costituisce per l'utente una macchina virtuale che complessivamente *analizza il testo* del programma e *lo esegue* secondo le regole del linguaggio.

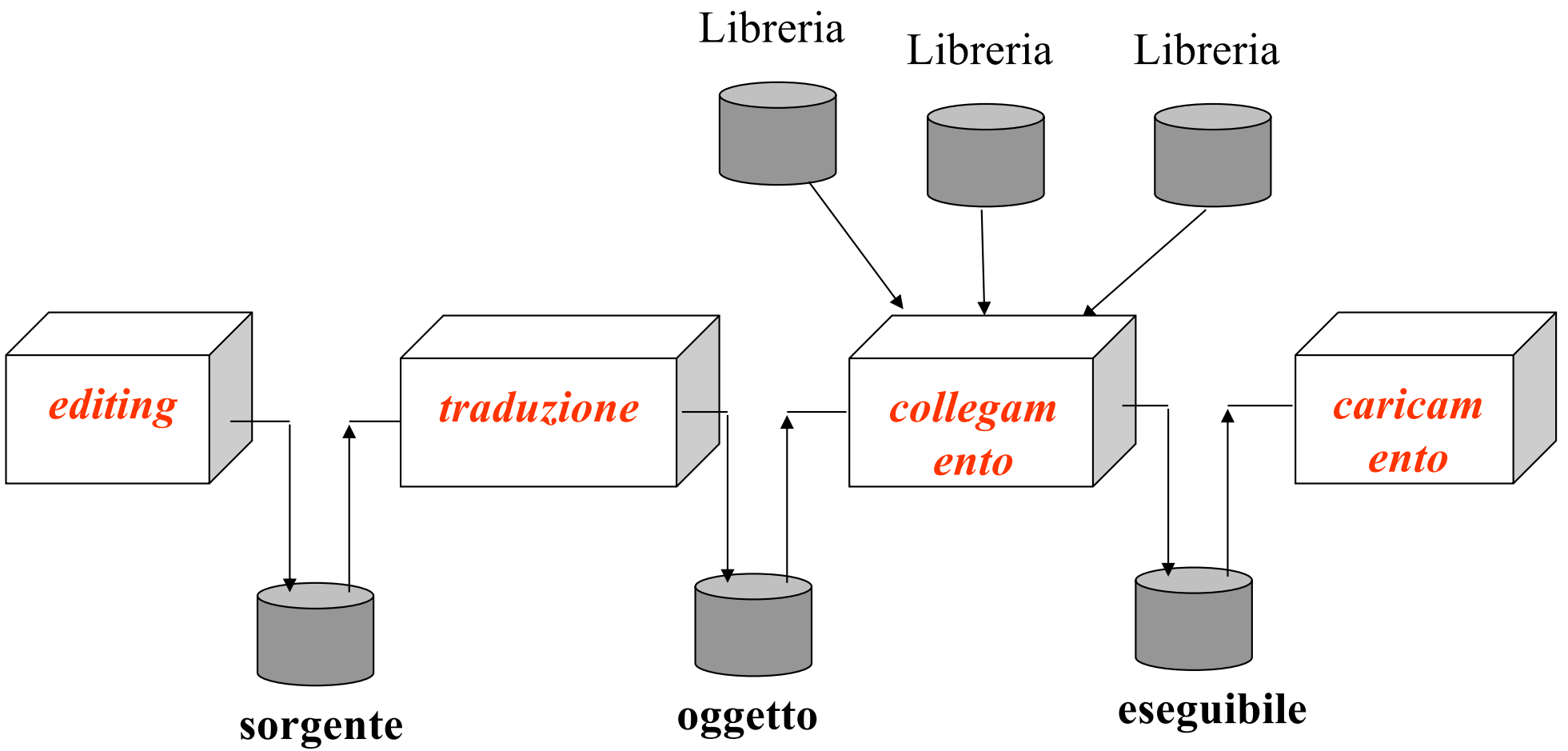
Traduzione e Interpretazione

- In realtà, la traduzione del testo del programma - detto anche *testo sorgente* o origine - in operazioni eseguibili sulla macchia reale, avviene con tecniche diverse:
  -) preventiva **traduzione** del testo origine nel linguaggio della macchina reale
  -) **interpretazione** del testo origine e sua esecuzione.

Diversi tipi di traduttori

- I traduttori assumono nomi diversi a seconda del tipo di linguaggio sorgente e del linguaggio oggetto:
  - *compilatori* (traduttori da un linguaggio algoritmico a linguaggio macchina),
  - *precompilatori* (traduttori sorgente/sorgente)
  - gli *assemblatori* (traduttori da linguaggio assemblativo a linguaggio macchina).

Il processo di Traduzione



Editor

- E’ Lo strumento che permette di introdurre il testo di un programma, di visionarlo ed eventualmente di modificarlo.
- Il testo sorgente prodotto viene memorizzato su un file.
  - Il file prodotto è un file testo con una estensione che ‘ricorda’ il linguaggio con cui è stato scritto: nome.c, nome.cpp, nome.pas sono un esempio di programma sorgente scritto in linguaggio C, C++ o pascal.

Compilatore

- Il compilatore è lo strumento che permette la traduzione da sorgente a oggetto.
  - Il risultato del processo di compilazione è un *file oggetto*.
- Il compilatore tuttavia non è in grado di mettere assieme le diverse componenti di un programma.

Linker e Loader

- Il Collegatore o Linker permette di
  - aggregare componenti scritti in momenti diversi da persone diverse ed eventualmente in linguaggi diversi
  - effettuare l'aggancio tra la definizione di una procedura scritta a parte e il suo richiamo.
- Per essere eseguito deve però essere caricato in memoria centrale.
  - Questa operazione è affidata ad un altro programma detto caricatore o **loader** che legge da disco *l'immagine di memoria* del programma, decide dove posizionarlo in memoria e gli trasferisce il controllo.

Il processo di Compilazione

- Il processo di traduzione è tanto più complesso quanto più vicino al linguaggio naturale è il linguaggio da tradurre.
  - Quanto più il linguaggio ad alto livello svincola dai dettagli di macchina, tanto più aumenta il divario esistente tra la macchina virtuale che l'utente vede e la macchina reale sottostante (il cosiddetto *gap semantico*).

Fase di Analisi

- Scopo della fase di analisi è:
  - riconoscere le frasi appartenenti al linguaggio e attribuirvi un significato;
    - *nel caso il riconoscimento fallisca, determinarne i motivi, rilevare gli errori ossia le violazioni delle regole grammaticali*
  - costruire un insieme di informazioni sulle variabili e sulle procedure usate, sui tipi definiti dall'utente e sullo spazio necessario per contenere i dati.

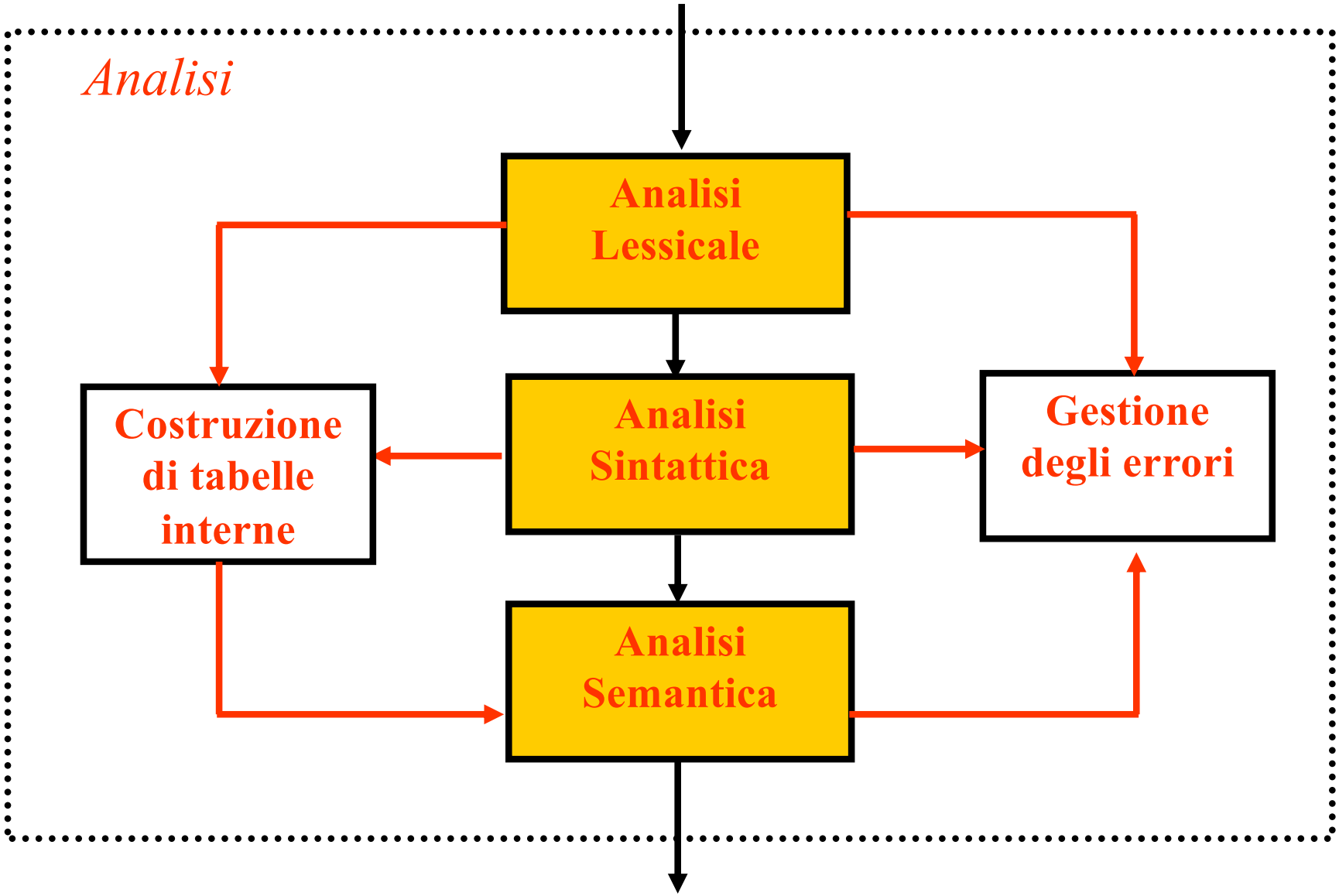
Cosa è un Linguaggio

- Elemento fondamentale di ogni linguaggio è ***l'alfabeto***.
  - L'alfabeto è definito come un insieme di *simboli* che chiameremo caratteri.
  - L'aggregazione di caratteri dell'alfabeto permette di costruire le *parole*.
  - Le parole possono essere a loro volta aggregate in *frasi* del linguaggio.
- Considereremo valide le parole o le frasi costruite secondo una ***grammatica***.

Lessico, Sintassi, Semantica

- Le regole di costruzione delle parole a partire dall'alfabeto sono dette anche regole **lessicali**.
- Il modo con cui le parole sono concatenate tra loro per costruire frasi corrette nel linguaggio costituisce la **sintassi** del linguaggi.
- Il significato da attribuire alla frase riguarda la **semantica** del linguaggio.

Fase di Analisi (2)



Analisi Lessicale

- L'analizzatore lessicale o **scanner**
  - verifica le regole di costruzione delle parole
  - estrae i cosiddetti token (parole chiave, identificatori, costanti, operatori, etc.).
- Lo scanner trasforma il testo sorgente in una sequenza di token.
  - Consideriamo ad esempio l'istruzione

    x:=100; viene trasformata nella sequenza

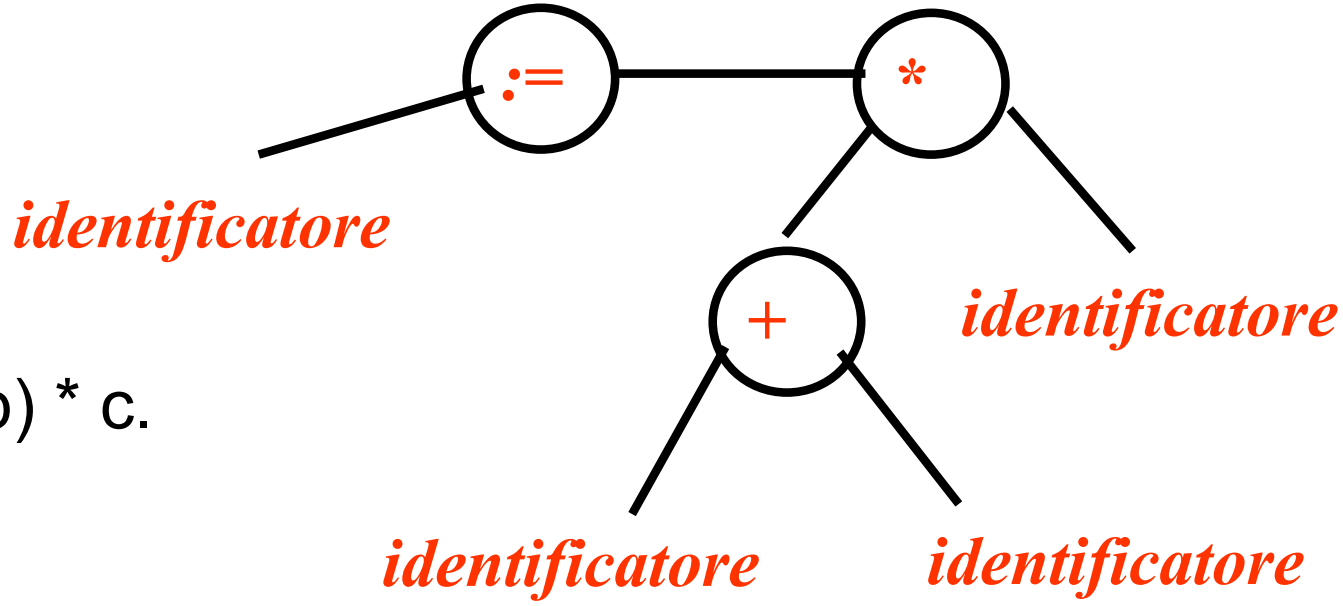
    ***identificatore, operatore di assegnamento, costante, punto e virgola*.**

Analisi Sintattica

- L'analizzatore sintattico, o **parser**, raggruppa i token in strutture sintattiche.
- Le regole sintattiche vengono definite attraverso opportuni metalinguaggi,
  - come la BNF (Backus_Naur Form)
  - le carte sintattiche.

Alberi Sintattici

- Gli alberi sintattici sono alberi i cui **nodi** sono gli operatori del linguaggio e le **foglie** gli operandi.



x := (a + b) * c.

Analisi Semantica

- L'analizzatore semantico stabilisce se una frase ha qualche significato.
  - Nel caso dei linguaggi di programmazione costituiscono violazioni semantiche rilevabili a compilation time
    - operazioni tra operandi di tipo **non compatibile** (esempio: a = b, con a di tipo carattere e b di tipo reale)
    - **operatori illegali** per il tipo di oggetti a cui sono applicati
    - uso di **oggetti non dichiarati**
    - il **richiamo di funzioni** con un numero di parametri (*ordine e tipo*) diverso da quello della dichiarazione

Tabella dei Simboli

- Alle variabili dichiarate è necessario associare ***al loro nome***
  - ***il loro tipo***

    per poter riservare lo spazio destinato a contenere il valore che assumeranno durante l'esecuzione,
  - l'indirizzo di memoria

    in cui collocare il valore
  - le modalità di accesso,

    che saranno diverse a seconda che si tratti di una variabile globale, locale, di un parametro o di una funzione.

La Sintesi del Codice

- L'obiettivo di questa fase è la trasformazione della forma intermedia - prodotta dalle fasi di analisi - in codice effettivamente eseguibile da una macchina reale
  - L'approccio più semplice: scandire la forma intermedia e le tabelle e di procedere con una generazione quasi uno ad uno delle istruzioni in linguaggio macchina.

Un Esempio (1)

- Ad esempio, dato il blocco:

```
x = a+b ;
y = a+b ;
```

- un eventuale codice prodotto:

```
LOAD    a
ADD     b
STORE   x
LOAD    a
ADD     b
STORE   y
```

Un Esempio (2)

- Ma la soluzione è inefficiente! R
  - Ricordando, infatti, l'uso dei registri e il valore che via via contengono, è possibile trasformare il codice precedente in

```
LOAD    a
ADD     b
STORE   x
STORE   y
```

Ottimizzazione del Codice

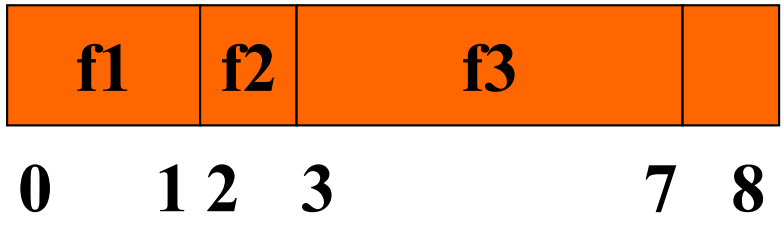
- Il generatore di codice deve in generale non solo **rispettare la semantica** del testo originale, ma anche **gestire efficientemente** le risorse della macchina reale per ottenere un programma dalle **prestazioni ottimali**.

Spazio/Velocità

- Una ottimizzazione del codice si ottiene ...
  - **attraverso una riduzione dell'occupazione in memoria** in quanto meno codice significa in genere esecuzione di un numero minore di istruzioni
- Una ottimizzazione spinta dell'area dati provoca spesso un aumento dei tempi di accesso alle strutture stesse.
- Riduzione di spazio di memoria e velocità di esecuzione diventano spesso conflittuali.
  - E' proprio nel compromesso che viene raggiunto nel rapporto spazio/velocità che si valuta la qualità di un compilatore

esempio

```
Type rec= record
        f1:0..3;
        f2: boolean;
        f3:0..15;
end;
```



Ma che succede se si vuole accedere al campo f3?

Sintesi del codice: fase 1

- Generazione di un codice intermedio, non orientato a nessuna macchina reale.
  - Si genera, cioè, del codice di livello molto basso, ma ancora largamente indipendente dalla CPU per la quale verrà generato il codice finale. Questo tipo di codice, in genere, **ignora l'esistenza** dei registri di macchina, **la loro natura**, **il loro numero** e **le reali modalità** di indirizzamento della memoria.

Sintesi del codice: fase 2

- Il codice intermedio viene tradotto nel codice macchina **ottimizzato** per un determinato processore.
  - La forma intermedia ha in genere delle caratteristiche che permettono di effettuare delle operazioni rivolte ad ottimizzare il codice che verrà prodotto.

Linker

- Il Linker o Collegatore fonde gli spazi di indirizzi separati dei moduli oggetto in un unico indirizzo lineare.
  - Per poter aggregare in un unico file eseguibile le diverse unità di compilazione prodotte, occorre che il compilatore generi delle informazioni opportune nel modulo oggetto.

Modulo Oggetto

IDENTIFICAZIONE
TABELLA DEI SIMBOLI IMPORTATI
TABELLA DEI SIMBOLI ESPORTATI
ISTRUZIONI MACCHINA
DIZIONARIO DI RILOCAZIONE
FINE MODULO

Come funziona un Linker (1)

- 1. Costruisce una tabella di tutti i moduli oggetto e delle loro lunghezze

Name	Length	Starting
A	400	100
B	600	500
C	500	1100
D	300	1600

- 2.Assegna un indirizzo di caricamento ad ogni modulo oggetto

Come funzione un linker (2)

- 3. Individua tutte le istruzioni che contengono un indirizzo di memoria e aggiunge una costante, detta **costante di rilocazione**, uguale all'indirizzo di partenza del modulo in cui sono contenute
- 4. Trova tutte le istruzioni che fanno riferimento ad altre procedure e vi inserisce l'indirizzo di queste procedure.

Esempio (1)

1. ***Costruzione della tebella dei simboli***
2. ***Fusione dello Spazio di indirizzi***

	module A		module C
0	JMP 200	0	JMP 200
200	Move #0,D1	200	Move #3,D1
300	JSR B		
400		400	JSR D
		500	
	module B		
0	JMP 300		module D
		0	JMP 200
200	Move #2,D1		
		200	Move #4,D1
		300	
500	JSR C		
600			

Esempio (2)

0	JMP 200	interrupt
100		A
300	Move #0,D1	
400	JSR B	
500	JMP 300	B
800	Move #2,D1	
1000	JSR C	
1100	JMP 200	C
1300	Move #3,D1	
1500	JSR D	
1600	JMP 200	D
1800	Move #4,D1	
1900		

Linker Symbol Table

Name	Length	Starting
A	400	100
B	600	500
C	500	1100
D	300	1600

Esempio (3)

0	JMP 300	interrupt
100		A
300	Move #0,D1	
400	JSR 500	
500	JMP 800	B
800	Move #2,D1	
1000	JSR 1100	
1100	JMP 1300	C
1300	Move #3,D1	
1500	JSR 1600	
1600	JMP 1800	D
1800	Move #4,D1	
1900		

- **Modulo Assoluto**

L'indirizzo di caricamento (LA) è specificato ad Assembly Time

- **Rilocabile**

LA è specificato a Load Time

Caricamento

- Nei sistemi multiprogrammati, non è possibile, prevedere a priori da parte del linker l'esatta porzione di memoria libera per il caricamento.
- E' necessario, quindi, ricorrere ad una forma di rappresentazione dei programmi più flessibile. Una via percorsa è quella del **codice binario rilocabile**.

Indirizzi Relativi

- Gli indirizzi prodotti dai traduttori e dal linker non sono indirizzi assoluti,
  - cioè indirizzi che corrispondono uno a uno con gli indirizzi della memoria fisica,
- ma indirizzi **relativi ad una origine** di indirizzo zero.

Rilocazione

- In fase di caricamento il caricatore può rilocare tutti gli indirizzi spiazzandoli di una certa quantità rispetto all'origine: occorre cioè che in fase di caricamento, ad ogni indirizzo venga sommato, a opera del caricatore, questo valore iniziale, detto anche **base di rilocazione**.

Marche di rilocazioni

- Informazioni ausiliarie prodotte dal linker che ...
  - marcando esplicitamente le istruzioni che possono essere modificate,
  - permettono che diversi segmenti di codice siano allocati non contiguamente accessibili mediante **diverse basi** di rilocazione.
- Scopo: usare frammenti liberi non contigui di memoria, ottimizzando cosi' lo sfruttamento della stessa.

Interpreti

- Un interprete è un esecutore che definisce una **macchina astratta** attraverso il linguaggio che riconosce
  - per un interprete il linguaggio sorgente diventa il suo linguaggio macchina.
- Un interprete costituisce un sistema chiuso
  - esso **esegue** un certo codice e **non deve generare** nessun eseguibile per una macchina reale.

Compilatori vs Interpreti

- Interpreti
  - Indipendenza dalla Macchina Ospite
  - Indipendenza dal Sistema Operativo Ospite

  - Lo stesso programma può girare su macchine eterogenee
  - es. INTERNET
- Compilatori
  - Dipendenza dal processore e dal Sistema Operativo
  - Maggiore efficienza
    - velocità
    - occupazione di risorse