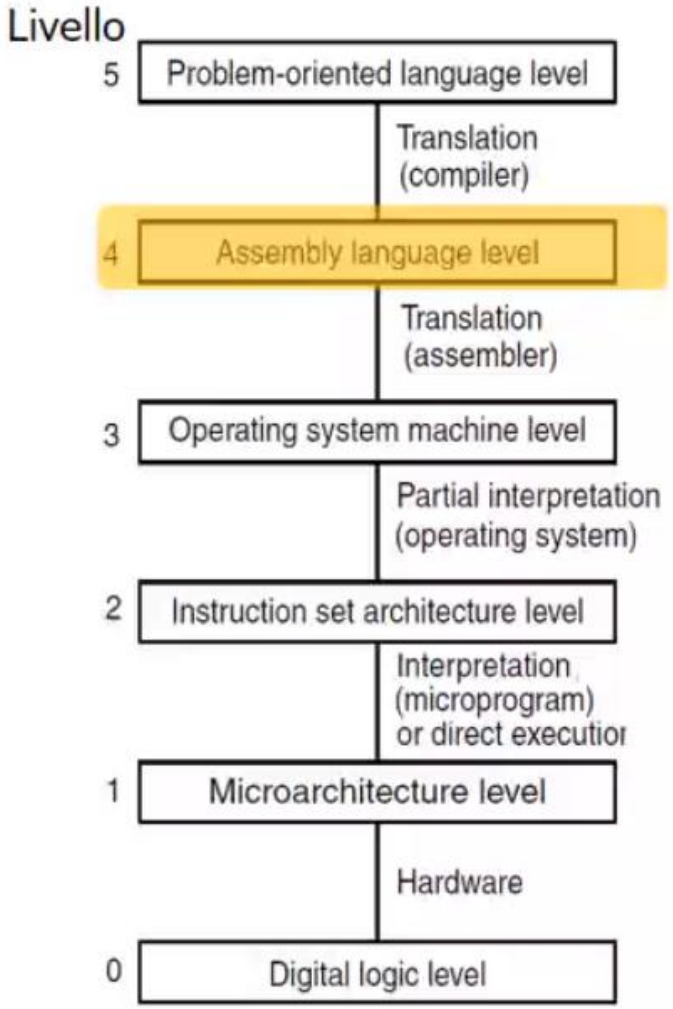
Il livello del linguaggio assemblativo



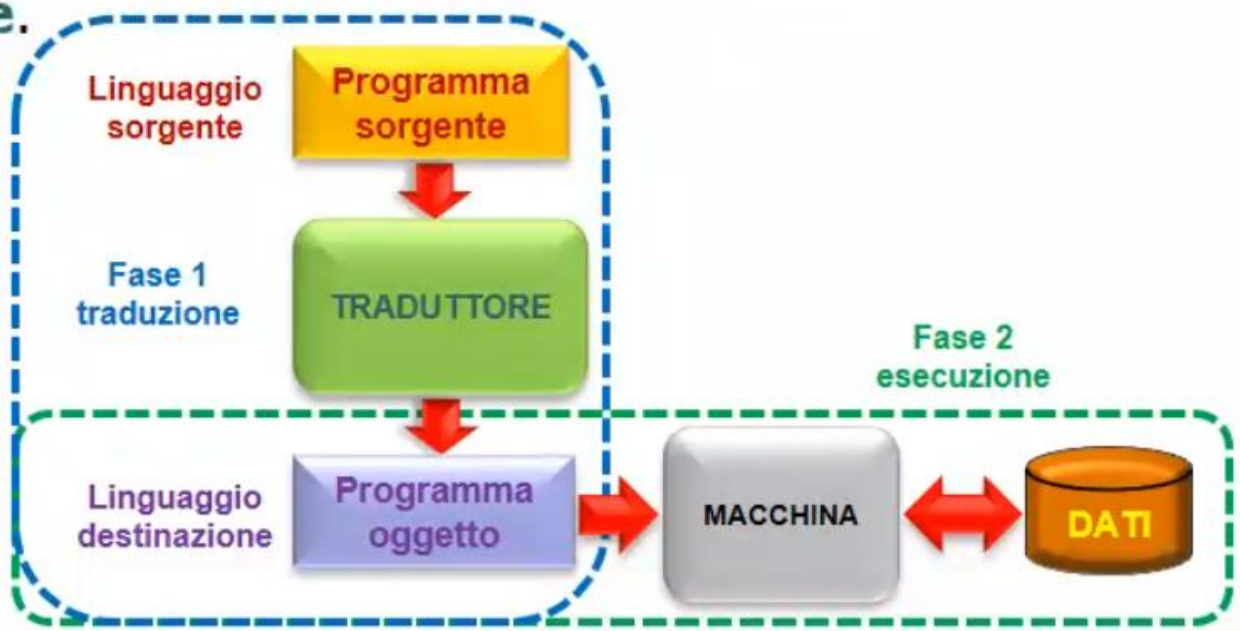
Ricordiamo che:

Assembly è sinonimo di assemblativo e fa riferimento al linguaggio.

Assembler è sinonimo di assemblatore e fa riferimento al traduttore.

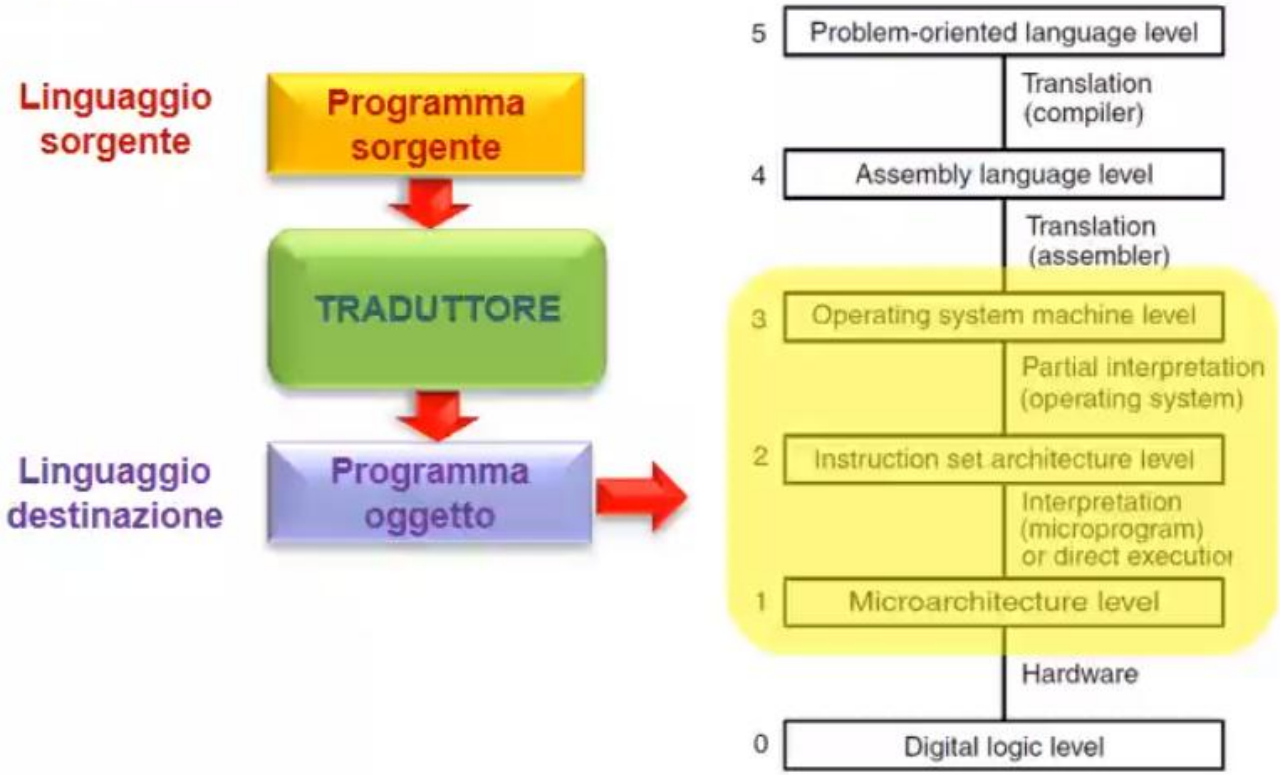
Introduzione

- Mentre il livello di micro e macro-architettura, del sistema operativo sono **interpretati**, il livello del linguaggio assembly è realizzato mediante **traduzione**.
- Il programma che converte un programma sorgente nel corrispondente programma destinazione è detto **traduttore**.



Introduzione

- Mentre si esegue il programma oggetto sono coinvolti i livelli di micro e macro architettura ed il livello macchina del SO.



Introduzione al linguaggio assemblativo

- Il linguaggio sorgente è una rappresentazione simbolica del linguaggio macchina (linguaggio assemblativo)
- Simboli mnemonici (ADD, SUB, MUL,...) sono più semplici da ricordare rispetto ad una sequenza di bit ...010100...
- Il linguaggio assemblativo ha una corrispondenza **uno ad uno** rispetto a quello macchina.
- Il programmatore assembler ha accesso a tutte le risorse della macchina, differentemente da uno che lavora con linguaggi ad alto livello (es. non può vedere i registri).
- Un programma è scritto per una specifica famiglia di macchine, programma ad alto livello è "*machine independent*".

Perché usare il linguaggio assemblativo?

- Programmare in Assembler è complesso.
- Per alcune categorie di applicazioni ove servono ridotte dimensioni del programma, velocità e completo sfruttamento della macchina non esiste alternativa!
- N = I + J eseguito su architettura x86:

Etichetta	Opcode	Operandi	Commenti
FORMULA:	MOV	EAX,I	; EAX = I
	ADD	EAX,J	; EAX = I + J
	MOV	N,EAX	; N = I + J
I	DD	3	; I=3 (I occupa 4 byte)
J	DD	4	; J=4 (J occupa 4 byte)
N	DD	0	; N=0 (N occupa 4 byte)

Define Double (nell'8088 la parola era a 16 bit!)

Pseudoistruzioni

- I comandi che l'assemblatore utilizza nel codice sono dette **pseudoistruzioni** o **direttive dell'assemblatore**.

Esempi

- Per definire un nuovo simbolo pari al valore di una espressione:

```
BASE EQU 1000
```

- Per allocare 3 byte con dei valori fissati:

```
TABLE DB 11,23,49
```

- Assemblaggio condizionale:

```
WORDSIZE EQU 32
IF WORDSIZE GT 32
   WSIZE DD 64
ELSE
   WSIZE DD 32
ENDIF
```

Pseudoistruzioni MASM x86

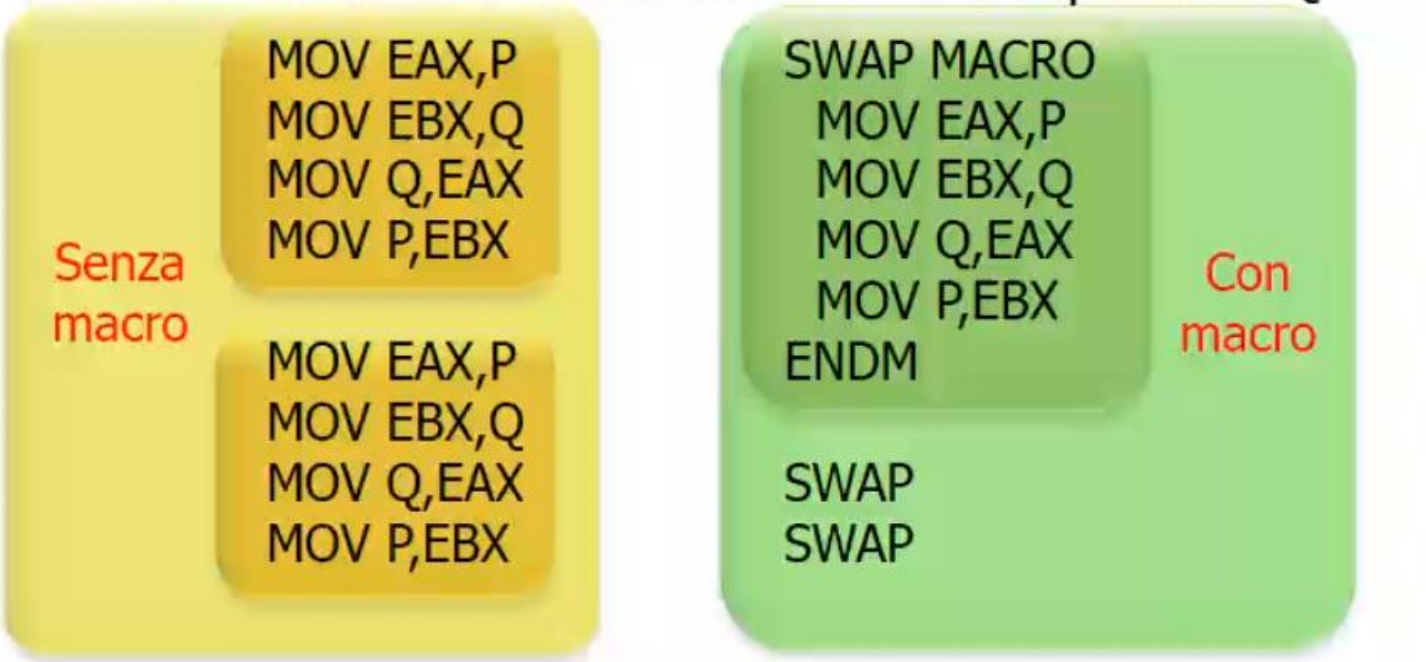
SEGMENT	Indica l'inizio di un segmento (testo,dati,..) con certi attributi.
ENDS	Indica la fine del segmento corrente.
ALIGN	Controlla l'allineamento della prossima istruzione o dei successivi dati.
EQU	Definisce un nuovo simbolo assegnandogli una data espressione.
DB	Allocata spazio per uno o più byte, inizializzandoli.
DW	Allocata spazio per uno o più word (16-bit), inizializzandoli.
DD	Allocata spazio per uno o più doubleword (32-bit), inizializzandoli.
DQ	Allocata spazio per uno o più quadword (64-bit), inizializzandoli.
PROC	Indica l'inizio di una procedura.
ENDP	Indica la fine di una procedura.
MACRO	Indica l'inizio della definizione di macro.
ENDM	Indica la fine della definizione di macro.
PUBLIC	Esporta un nome di identificatore definito nel modulo corrente.
EXTERN	Importa un nome di identificatore da un altro modulo.
INCLUDE	Preleva ed include un altro file.
IF	Inizio assemblaggio condizionale: se la condizione è vera assembla le istruzioni che seguono.
ELSE	Se la condizione IF è falsa assembla le istruzioni che seguono.
ENDIF	Fine assemblaggio condizionale
COMMENT	Definisce un nuovo simbolo per il commento.
PAGE	Genera una interruzione di pagina nel listato.
END	Indica il termine del programma assemblativo.

LE MACROISTRUZIONI

- La definizione di una macro è un modo per assegnare ad un nome una porzione di testo in modo che l'assembler esegua poi la macrosostituzione.

Esempio

- Si vuole scambiare il contenuto di P con quello di Q.



Confronto tra macro e procedure

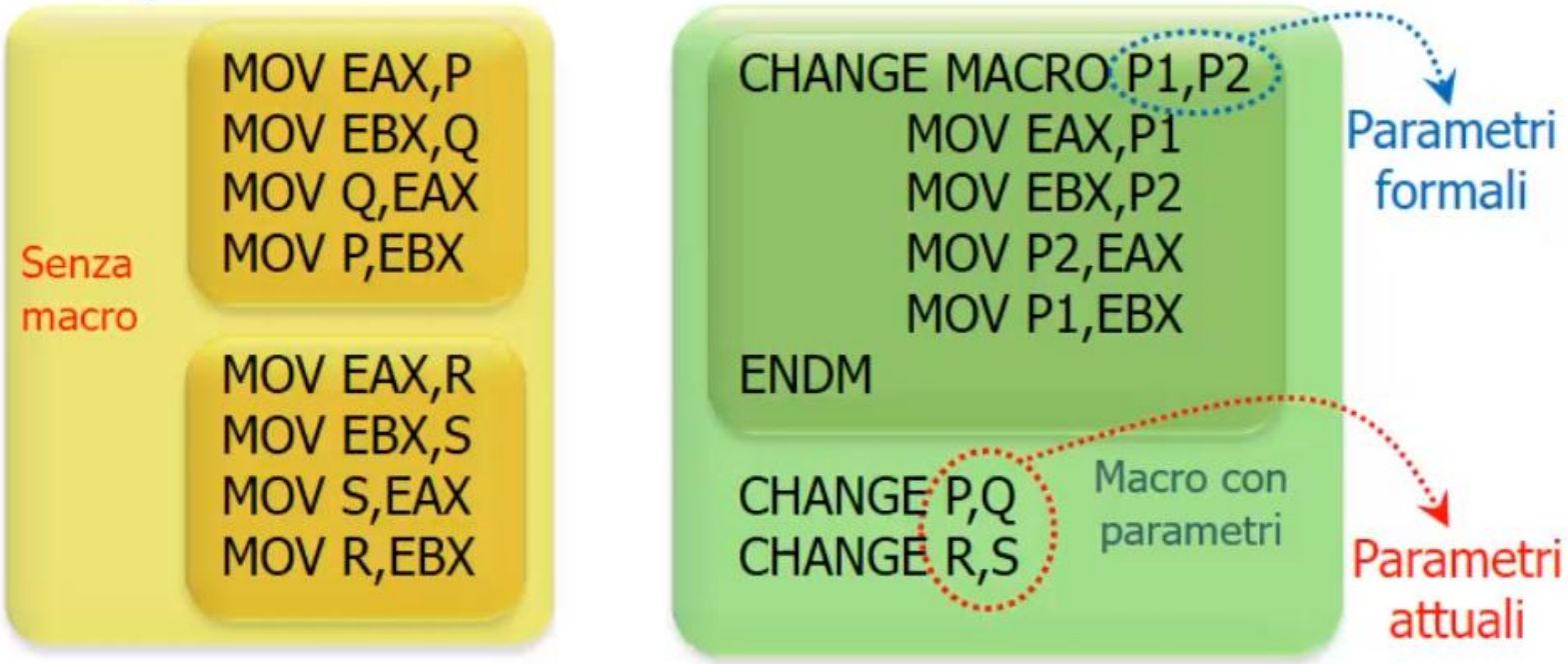
Le macro non devono essere confuse con le procedure!

Questione	chiamata alla Macro	chiamata alla Procedura
Quando è fatta la chiamata?	Durante l'assemblaggio (compile-time)	durante l'esecuzione (run-time)
Il corpo è inserito nel programma oggetto ogni volta che appare la chiamata?	Sì	No
L'istruzione per la chiamata di procedura è inserita nel programma oggetto e successivamente eseguita?	No	Sì
Occorre utilizzare una istruzione di ritorno dopo aver eseguito una chiamata?	No	Sì
Quante copie del corpo appaiono nel programma oggetto?	Tante quante sono le chiamate	Una

Macro con parametri

- La definizione di una macro può includere dei parametri formali che verranno poi sostituiti dai corrispondenti valori attuali.

Esempio



PROCESSO DI ASSEMBLAGGIO

- Poiché un programma si compone di istruzioni che possono avere dei "salti" in avanti l'assemblatore non può conoscere in anticipo la posizione dell'istruzione richiamata (**problema dei riferimenti in avanti**).
- Esistono due soluzioni:
  - Leggere il programma sorgente due volte costruendo la prima volta una tabella dei simboli, etichette ed istruzioni.
  - Leggere il programma sorgente una volta e convertirlo in un formato intermedio in una tabella in memoria eseguendo poi un secondo passaggio sulla tabella.
- In entrambe le soluzioni il primo passaggio ha il compito di espandere tutte le macro.

Primo passaggio

- Il primo passaggio costruisce la **tabella dei simboli**.
- Durante questa fase l'assembler utilizza una variabile **ILC** (*Instruction Location Counter*) che memorizza l'indirizzo dell'istruzione che sta assemblando.
- La maggior parte degli assembler utilizza tre tabelle interne per memorizzare:
  - I simboli.
  - Le pseudoistruzioni.
  - I codici operativi.

Esempio

Etichetta	Opcode	Operandi	Commenti	Lung.	+	ILC
MARIA	MOV	EAX,I	;EAX = I	5		100
	MOV	EBX,J	;EBX = J	6		105
ROBERTA:	MOV	ECX,K	;ECX = K	6		111
	IMUL	EAX, EAX	;EAX = I*I	2		117
	IMUL	EBX, EBX	;EBX = J*J	3		119
	IMUL	ECX, ECX	;ECX = K*K	3		122
MARILYN:	ADD	EAX, EBX	;EAX = I*I+J*J	2		125
	ADD	EAX, ECX	;EAX = I*I+J*J+K*K	2		127
STEPHANY:	JMP	DONE	;branch to DONE	5		129

Tabella dei simboli

Symbol	Value
MARIA	100
ROBERTA	111
MARILYN	125
STEPHANY	129

Secondo passaggio

- Durante il secondo passaggio è generato il codice oggetto.

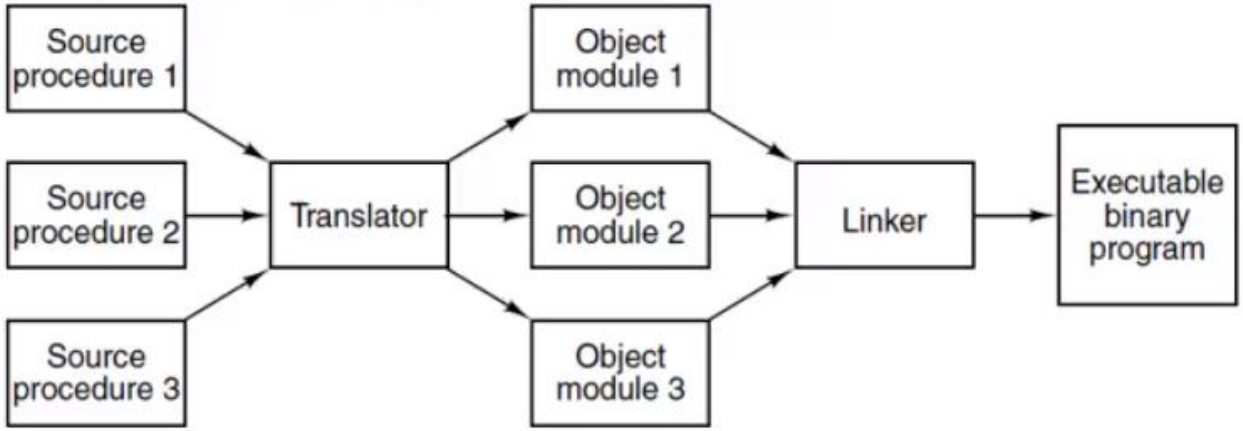
- Deve generare le informazioni utili al **linker** per collegare in un unico file eseguibile tutte le procedure assemblate in momenti distinti.

La tabella dei simboli

- La tabella dei simboli è una tabella associativa: un insieme di coppie <simbolo, valore> accessibili tramite il simbolo.
- Esistono varie tecniche per realizzarla:
  1. Utilizzare una struttura ordinata ed accedervi in modo dicotomico (O(log n)).
     - Mantenere ordinata una struttura di dati costa dal punto di vista computazionale.
  2. Usare una codifica **hash** che mappa i simboli nell'intervallo da 0 a k-1 (O(1)).
     - Problema dell'uniforme distribuzione della funzione e delle collisioni.

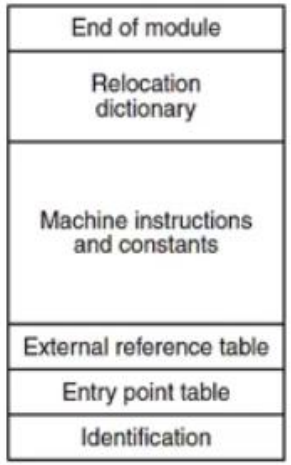
LINKER e LOADER

- Tutti i programmi hanno più procedure.
- Gli assembler traducono una procedura alla volta in linguaggio oggetto e salvano il risultato sul disco.
- Prima di poter eseguire il programma occorre collegare in modo appropriato tutte le procedure oggetto (**linking**).
- In assenza di memoria virtuale occorre caricare, attraverso il **loader**, il programma generato in memoria centrale prima di eseguirlo.



La struttura di un codice oggetto

- Inizialmente troviamo il codice identificativo e le lunghezze delle singole parti del modulo (dati utili al linker).
- Segue l'insieme dei punti di ingresso a cui possono fare riferimento altri moduli (direttiva PUBLIC).
- Troviamo poi la lista dei riferimenti utilizzati all'esterno del modulo (es. richiami a procedure esterne attraverso la direttiva EXTERN).
- A questo punto troviamo il codice assemblato e le costanti (l'unica parte che verrà caricata in memoria al momento dell'esecuzione).
- Segue il dizionario di rilocazione che fornisce gli indirizzi che dovranno essere rilocati.



- Infine troviamo l'identificativo di fine modulo, l'indirizzo ove iniziare l'esecuzione, un eventuale *checksum* per rilevare gli errori che possono avvenire durante la lettura del modulo