Il **Linguaggio** macchina è il linguaggio nativo del computer su cui un certo programma vien eseguito.
Esso consiste di sequenze di bit che sono interpretate da un meccanismo interno al computer, tale sequenza può esser espressa in binario o in esadecimale.

Un programma di linguaggio **assembly** è tradotto in linguaggio macchina da un programma chiamato **assembler** che traduce una alla volta le istuzioni in assembly in codice macchina.

Il **Compilatore** è un programma che legge un programma in un linguaggio (**programma sorgente**) e lo traduce in un programma equivalente in un altro linguaggio (**programma target**).
Durante la fase di traduzione, il compilatore, riporta eventuali errori nel programma sorgente.

L'**Interprete** , a differenza del compilatore, non produce il programma target come traduzione; esso esegue direttamente le operazione specificate nel programma sorgente su input forniti dall'utente.

DIFFERENZE FRA COMPILATORE E INTERPRETE
Il programma target prodotto da un compilatore di solito è 10 volte più veloce di quello prodotto dall'interprete.

Invece l'interprete può eseguire una diagnostica degli errori più accurata poiché esegui il programma sorgente step dopo step

La **Macchina Virtuale** è un software che crea un ambiente di lavoro in cui l'utente può eseguire qualsiasi applicazione indipendentemente dal sistema operativo sottostante.

LINGUAGGI COMPILATI: DAL SORGENTE ALL'ESEGUIBILE
Il programma sorgente può essere diviso in più file separati. Il compito di unire questi file spetta al **Preprocessore**, il quale crea un programma sorgente modificato.
Il programma sorgente modificato viene dato in pasto al **compilatore**, il quale produce un codice in **linguaggio assembly** che è più facile da produrre e più facile per fare il debug.
Questo programma **assembly** viene processato dall'**assembler** il quale produce un **codice macchina rilocabile**.
Di solito i programmi lunghi sono compilati a pezzi, in questo modo il codice macchina rilocabile deve essere linkato con altri file in codice macchina rilocabile e file di libreria al fine di produrre un codice che giri su macchina.
Il **linker** ha il compito di risolvere gli indirizzi di memoria per evitare che un file faccia riferimento ad un altro file. Il **loader** ha il compito di mettere assieme tutti questi file rilocabili in memoria per l'esecuzione, calcolandone gli indirizzi fisici.

Ogni linguaggio simbolico è formato da:
- **Alfabeto:** insieme di simboli usati
- **Lessico**: insieme di parole che formano un linguaggio
- **Sintassi**: insieme di regole che definiscono la formazione delle frasi
- **Semantica**:significato da associare ad ogni parole e istruzione del linguaggio

Il processo di compilazione si può suddividere in due fasi:

Analisi: Analisi Lessicale, Analisi Sintattica, Analisi Semantica, Generazione del codice intermedio
Sintesi: Ottimizzazione, Generazione del codice target

La fase di analisi viene detta anche “front end” del compilatore e dipende dal tipo di linguaggio che deve essere tradotto, mentre la fase di sintesi viene detta anche “back and” del compilatore perchè dipende dalla macchina

ANALISI LESSICALE
In questa fase l'analizzatore lessicale (**scanner**) suddivide il codice sorgente in **lessemi** che memorizza in una **symble table** dove saranno contenute tutte le indicazione necessarie per identificare i singoli lessemi. Per ogni lessema viene prodotto un **token**.
Un token può essere pensato come una coppia <nome_token, attributo> dove il nome rappresenta la descrizione astratta di un'unità lessicale del linguaggio
I compiti dello scanner sono la costuzione della symble table e la trasformazione del codice sorgente in forma semplificata.

ANALISI SINTATTICA
In questa fase l'analizzatore sintattico (parser) usa i token per definire la struttura grammaticale del linguaggio, ovvero controlla la struttura di ogni singola istruzione e verifica se sono state rispettate le regole di sintassi del linguaggio.
Il parser adopera il codice semplificato e la symble table e genera per ogni istruzione un albero sintattico (**syntax three**). L'output dell'analisi sintattica è costituito dall'insieme degli alberi sintattici che descrivono il programma

ANALISI SEMANTICA
L'analizzatore semantico usa il syntax three e la symble table per controllare la semantica con le definizioni del linguaggio, ovvero interpreta il significato delle strutture prodotte nelle fasi precedenti e controlla se sono legali e significative.
Una parte importante dell'analisi semantica è il **type checking** in cui l'analizzatore semantico controlla che ogni operatore abbia gli operandi giusti.
L'output dell'analisi semantica è l'albero sintattico decorato.

La semantica può essere:
- **Statica** se è indipendente dai dati su cui opera il programma sorgente (compilatore)
- **Dinamica** se dipende dai dati su cui opera il programma sorgente (interprete)

GENERAZIONE DEL CODICE INTERMEDIO
In questa fase si provvede alla generazione del codice intermedio , che è solitamente una rappresentazione di bassi livello.
Esistono vari modelli per tale rappresentazione,che deve rispettare due proprieta: facile da produrre e facile da tradurre in codice target.
Uno dei più usati è il three-address-code, in cui tutte le istruzioni hanno la forma delle istruzioni assembly con tre operandi per istruzione.

OTTIMIZZAZIONE DEL CODICE
In questa fase il codice intermedio viene analizzato e trasformato in codice ad esso equivalente ma più efficiente. Questa fase può essere molto complessa e lenta, proprio per questo molti compilatori permette di disattivare tale fase.

GENERAZIONE DEL CODICE TARGET O OGGETTO
In questa fase si provvede alla traduzione del codice intermedio, eventualmente ottimizzato, nel linguaggio della target-machine.

ANALIZZATORE LESSICALE

In questa fase lo scanner scandisce la sequenza di caratteri che costituisce il codice sorgente, li raggruppa in lessemi e produce una sequenza di token corrispondenti a tali lessemi (tokenizzazione)
A meno che non venga gestito da un precompilatore, lo scanner si preoccupa di:

- Rimuovere i commenti
- Case Conversion: in cui tutte le lettere vengono convertite in maiuscolo (alcuni linguaggi)
- Rimuovere gli spazi
- Tener traccia del numero di linea

Token:è una coppia costituita da un nome del token e da un attributo opzionale.
Il nome del token rappresenta un tipo di unità lessicale del programma sorgente.

Lessema:é una sequenza di caratteri nel programma sorgente identificati dallo scanner, posssiamo anche definirla coma un'istanza del token.

Patter:è la descrizione compatta della forma che il lessema di un token può assumere.
Nel caso delle keyword, il pattern è proprio la sequenza dei caratteri della keyword.
Nel caso degli identificatori, il pattern è la struttura in grado di matchare con molte stringhe.

Attributi:sono particolari informazioni relative ad un specifico lessema di un particolare token.
E' necessario quando più lessemi possono matchare con un pattern.

IMPLEMENTAZIONE DELLA SYMBLE TABLE
Lo scanner comunica con la symble table con operazioni del tipo:

- insert(s,t) restituisce l'indice di una nuova entry per la stringa s,token t
- lookup(s), restituisce l'indice della entry s, 0 se non la trova.

Le regole lessicali di un linguaggio di programmazione vengono descritte attraverso le espressioni regolari.

Un lessico può essere definito mediante espressioni regolari multiple, le quali possono produrre delle ambiguità. Per eliminare questo problema introduciamo:

- Longest best match in cui viene considerata come lessema la più lunga stringa che ha un match con espressione regolare.
- Regola di priorità in cui se due espressioni regolari hanno entrambe un match, allora il match viene determinato dalla prima espressione regolare.

FLEX

Funzionamento flex:
Vengono descritte tutte le espressioni regolari che definiscono i token.
Per ogni espressione regolare viene costruito un NFA, li combina in un unico NFA tramite epslon-transizioni.Infine converte tale NFA in DFA, lo ottimizza e produce il codice per la simulazione del DFA.
Per consentire il riconoscimento del longest best match e delle regole di priorità, durante la subset

construction ogni stato accettante del DFA conterrà le informazioni dell'NFA di provenienza