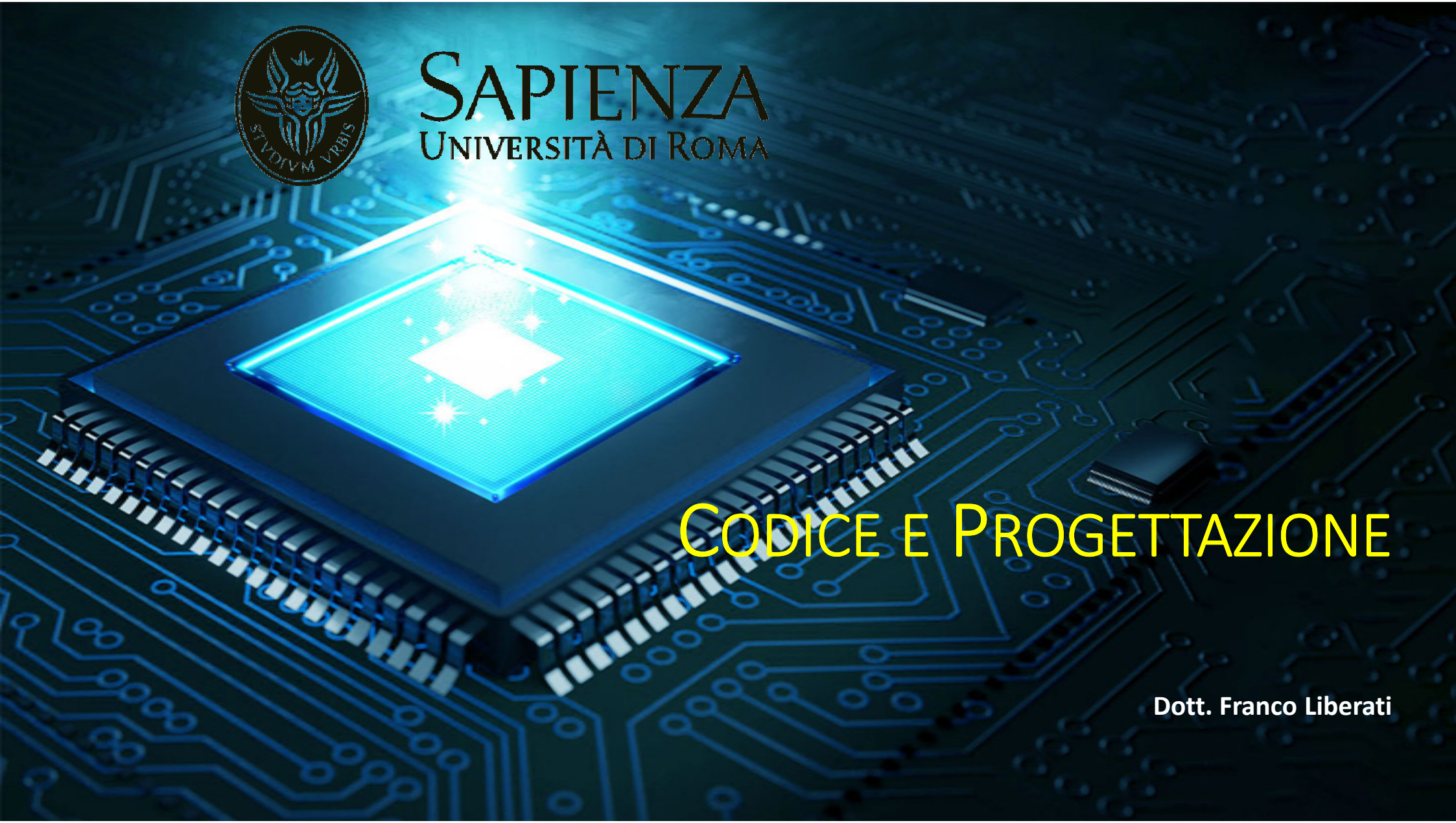

SAPIENZA
UNIVERSITÀ DI ROMA
CODICE E PROGETTAZIONE
Dott. Franco Liberati

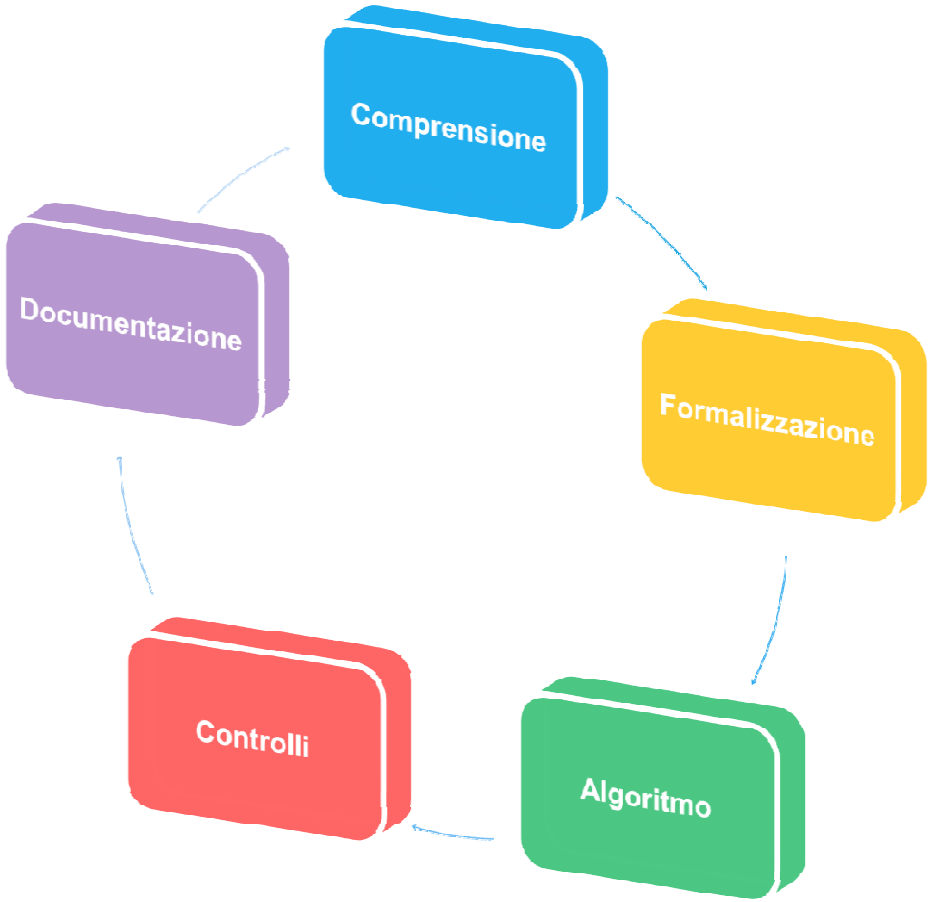

Argomenti
01
Strutture di controllo
02
Diagramma di flusso
03
Programma
04
Processo


Strutture di controllo

Strutture di controllo

Il primo passo da compiere quando si deve realizzare un programma informatico è la **comprensione del problema**, la sua **formalizzazione**, la definizione di quali operazioni compiere e l'ordine in cui devono essere svolte le istruzioni da impartire all'elaboratore

A questo è necessario aggiungere un approccio alla soluzione che preveda la minimizzazione delle istruzioni; degli scenari di uso; dei controlli; delle verifiche e una documentazione completa ed esaustiva



Strutture di controllo

Una procedura per la risoluzione di un problema, nei termini di azioni da eseguire, e l'ordine in cui questi atti devono essere svolti realizzano un **algoritmo**

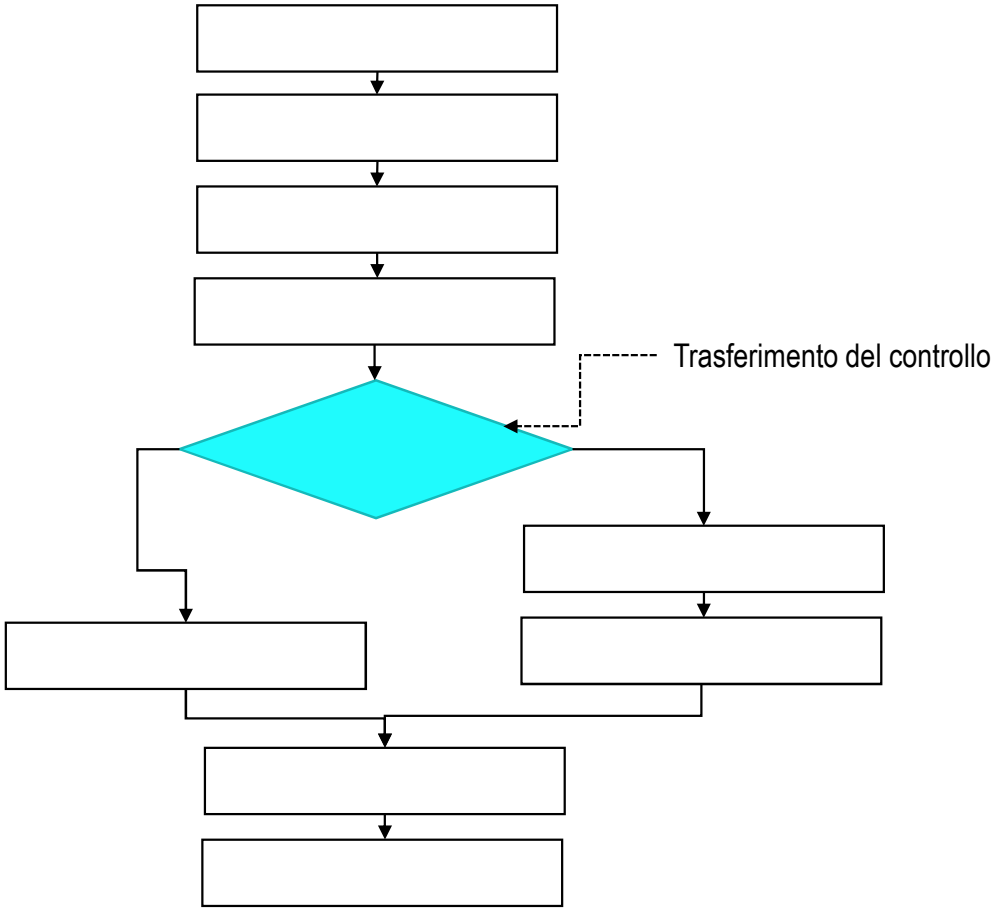
Un algoritmo può essere espresso sfruttando uno **pseudocodice**, ovvero un linguaggio artificiale e informale che consente di riflettere su come risolvere il problema prima di scrivere il codice informatico. Lo pseudocodice consiste in istruzioni di azione e di decisioni nonché la definizione delle variabili e delle costanti. Una formalizzazione di un problema utilizzando uno pseudocodice, se preparato con cura, può essere facilmente convertito in un programma

COMPUTO DEL NUMERO PARI DI CINQUE VALORI IMMESSI DA TASTIERA

```
i=5;
contatore=0;
WHILE (i>0)
 {
        READ(X);
        IF ((X%2)==0) THEN contatore=contatore+1;
         i=i-1;
}
PRINT(contatore);
```

Strutture di controllo

Un **programma** è definito come una serie di istruzioni sequenziali, mentre l'ordine in cui le istruzioni sono elaborate è detto il **controllo del programma**. Alcune istruzioni, quelle appartenenti alla classe dei salti, consentono di variare il normale andamento sequenziale e la loro presenza nel programma implica il **trasferimento del controllo**.

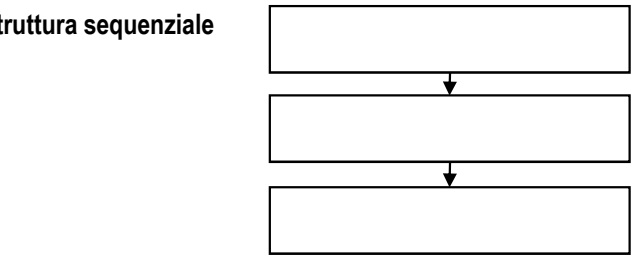


Strutture di controllo

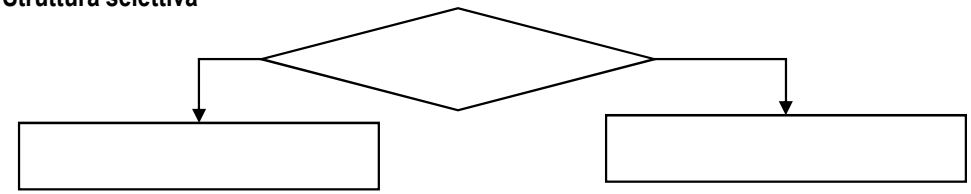
I lavori svolti da due professori italiani, Corrado Böhm e Giuseppe Jacopini, hanno dimostrato che tutti i programmi possono essere scritti in termini di sole tre **strutture di controllo**: sequenziale, selettiva e di iterativa.
La struttura **sequenziale** è l'elencazione di istruzioni eseguite una dopo l'altra nell'ordine in cui sono scritte (il blocco).
La struttura **selettiva** consente l'esecuzione sequenziale di un gruppo di istruzioni rispetto ad un altro in accordo alla veridicità di una condizione.
La struttura **iterativa** permette lo svolgimento sequenziale di un gruppo di istruzioni una o più volte fino al verificarsi di una condizione che ne stabilisca la terminazione e il passaggio a quanto scritto dopo

Un algoritmo organizzato secondo questa logica è detto a **programmazione strutturata**. Utilizzando tale approccio è possibile produrre programmi complessi, ma facili da capire (rispetto a quelli non strutturati) e quindi più comodi da verificare, correggere, modificare e perfino dimostrare in senso matematico come corretti.

Struttura sequenziale



Struttura selettiva



Struttura iterativa

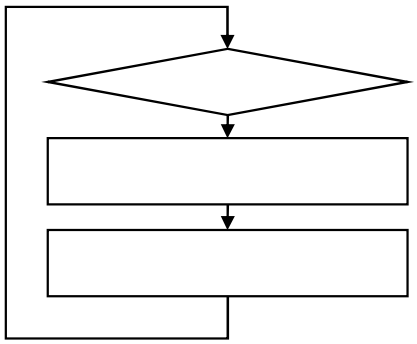



Diagramma di flusso

Diagramma di flusso

Un programma, e le relative strutture, possono avere anche una descrizione grafica: il **diagramma di flusso**. Come lo pseudocodice, questa raffigurazione è utile per descrivere un algoritmo ed offrire una rappresentazione chiara ed immediata di come e dove operano le strutture di controllo.

Un diagramma di flusso è disegnato usando dei **simboli** specifici: il rettangolo con i vertici arrotondati, il cerchietto, il rettangolo e il rombo. I simboli sono collegati da frecce chiamate **linee di flusso**. Le linee di flusso indicano l'ordine in cui sono eseguite le istruzioni ed intraprese delle azioni.

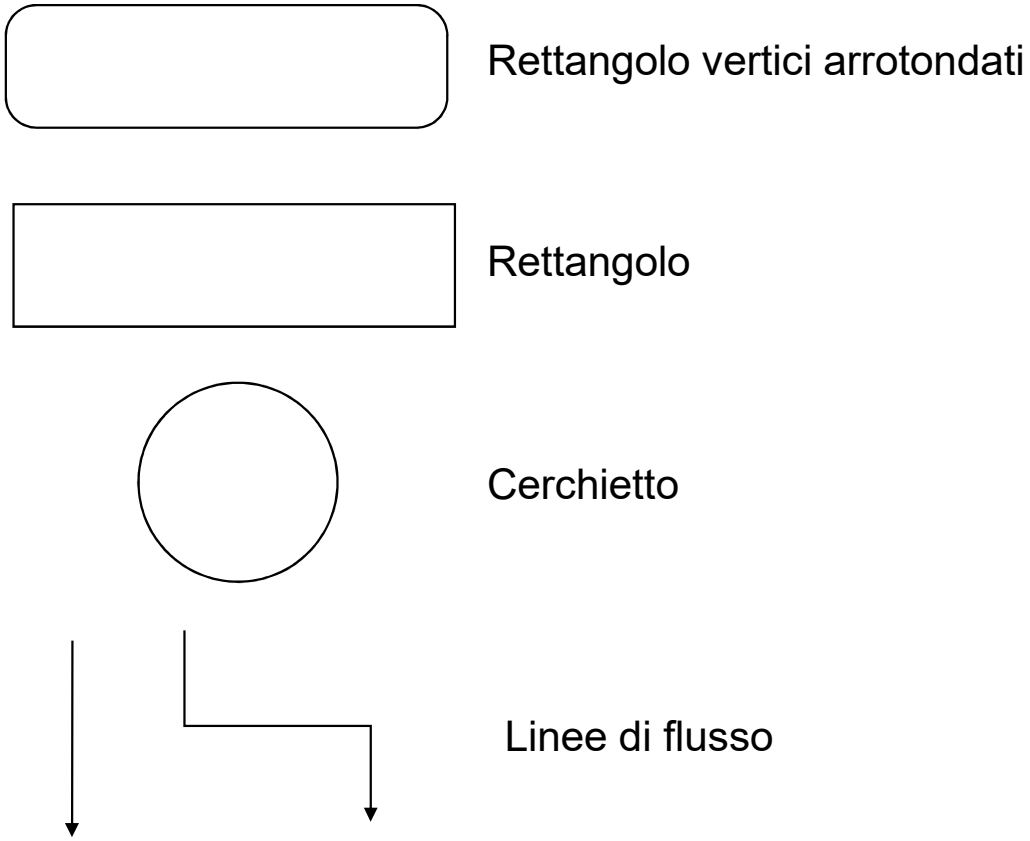
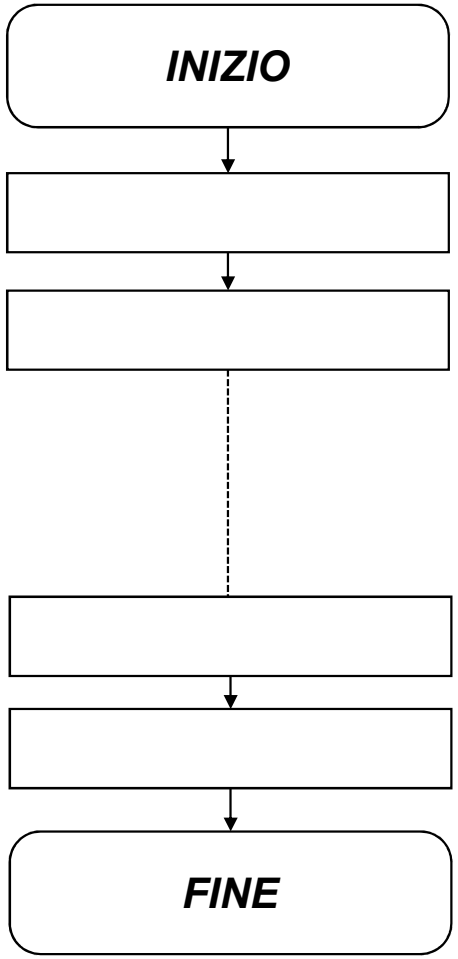


Diagramma di flusso

I **rettangoli** con i vertici arrotondati, da cui si sviluppa il resto del diagramma, individuano il **punto di inizio** e il **punto di terminazione** del programma. Il primo contiene la parola *Inizio* e non prevede alcuna linea di flusso entrante; l'altro riporta il termine *Fine* e non ha alcuna linea di flusso uscente.

Quando si disegna solo una porzione di un diagramma, magari la parte più interessante dell'algoritmo, si usano dei **cerchietti**, chiamati anche **simboli connettori**



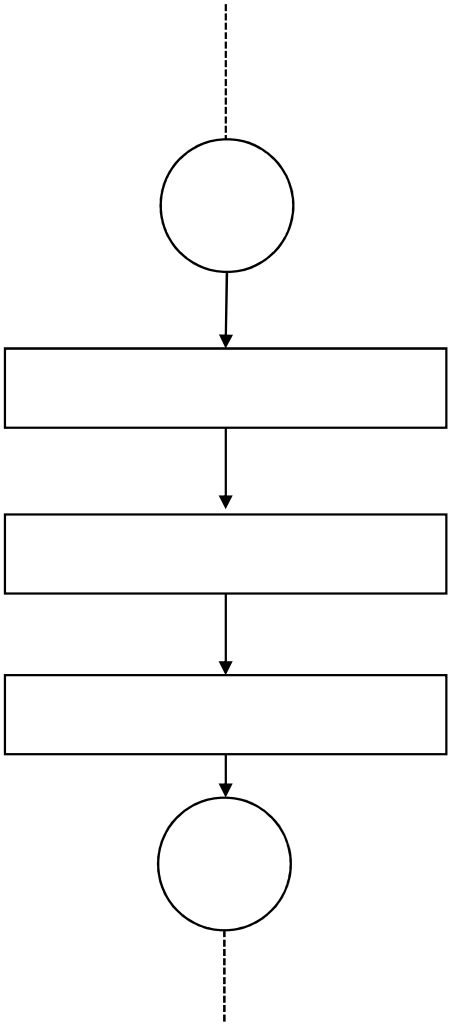
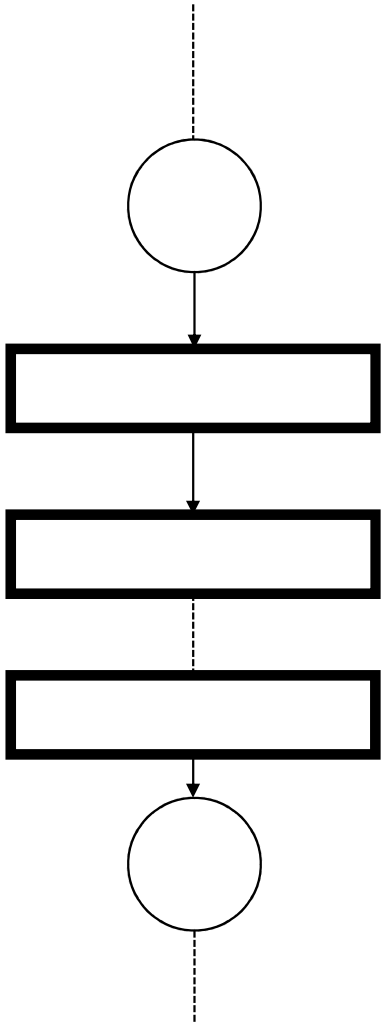


Diagramma di flusso

Il **rettangolo**, o **simbolo di azione** (*functional box*), esprime un calcolo o una operazione di interazione con una periferica (stampante, videoterminale, tastiera)

Il **rombo**, o **simbolo di decisione** (*predicative box*), è la forma geometrica più interessante di un diagramma di flusso perché indica la necessità di effettuare una decisione e, in accordo all'esito (vero, *true*, o falso, *false*), indirizzare il seguito dell'elaborazione verso una struttura sequenziale o un'altra



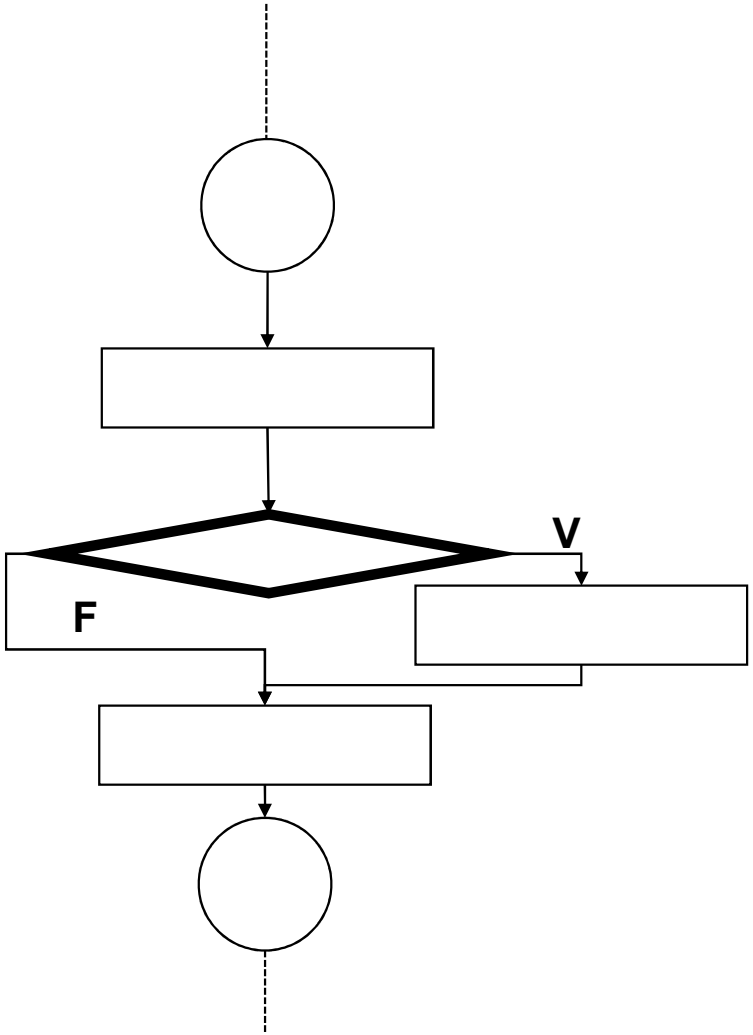


Diagramma di flusso

Esercizio
Confronto per determinare il maggiore tra due numeri o la loro uguaglianza

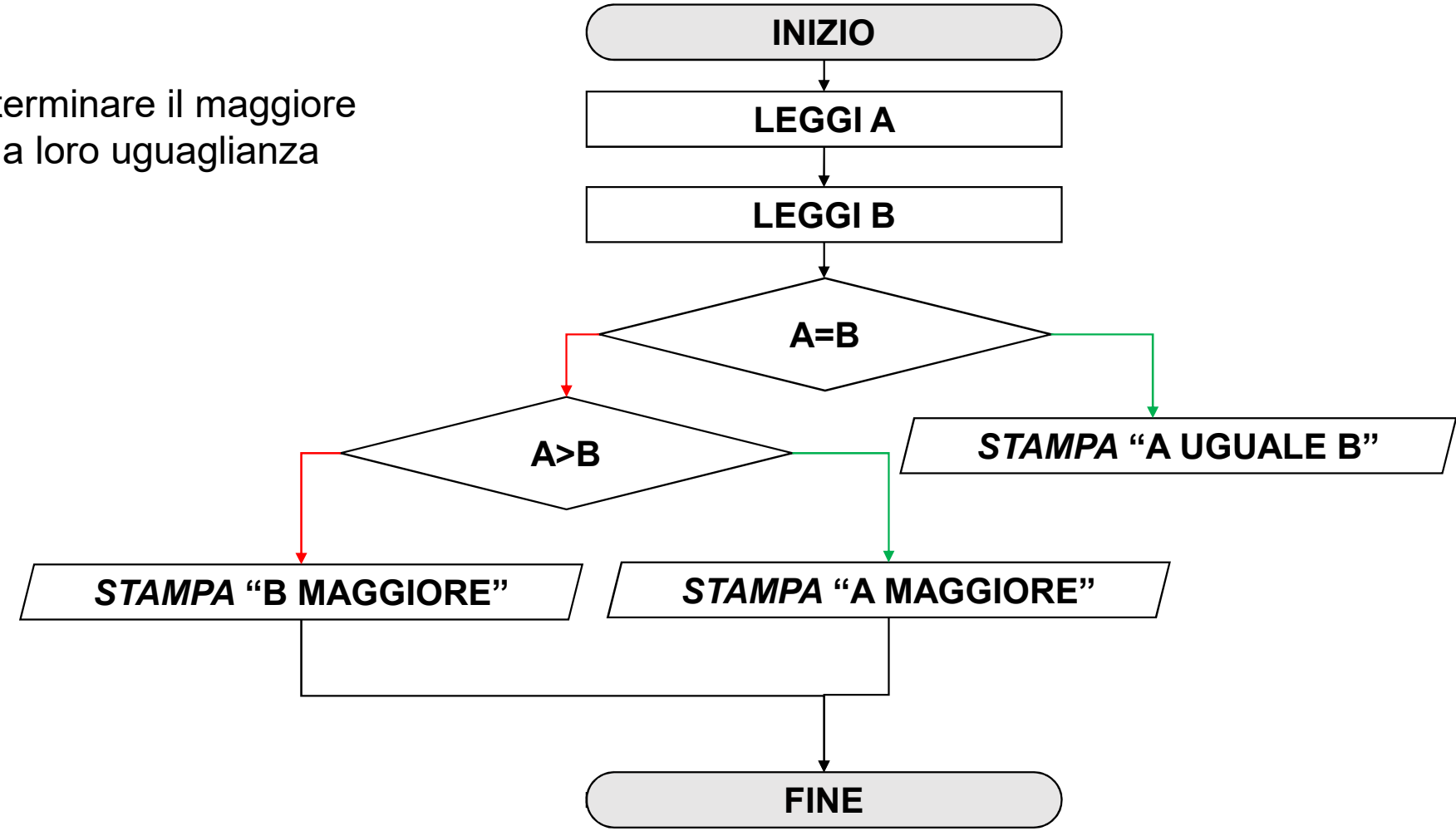


Diagramma di flusso

Esercizio proposto
Acquisire da tastiera cinque interi positivi e visualizzarne la somma.

Esempio
Input: 3,5,7,15,10
Output:**40**

Diagramma di flusso

Esercizio proposto
Acquisire da tastiera cinque interi positivi e visualizzarne la somma.

Esempio
Input: 3,5,7,15,10
Output:**40**

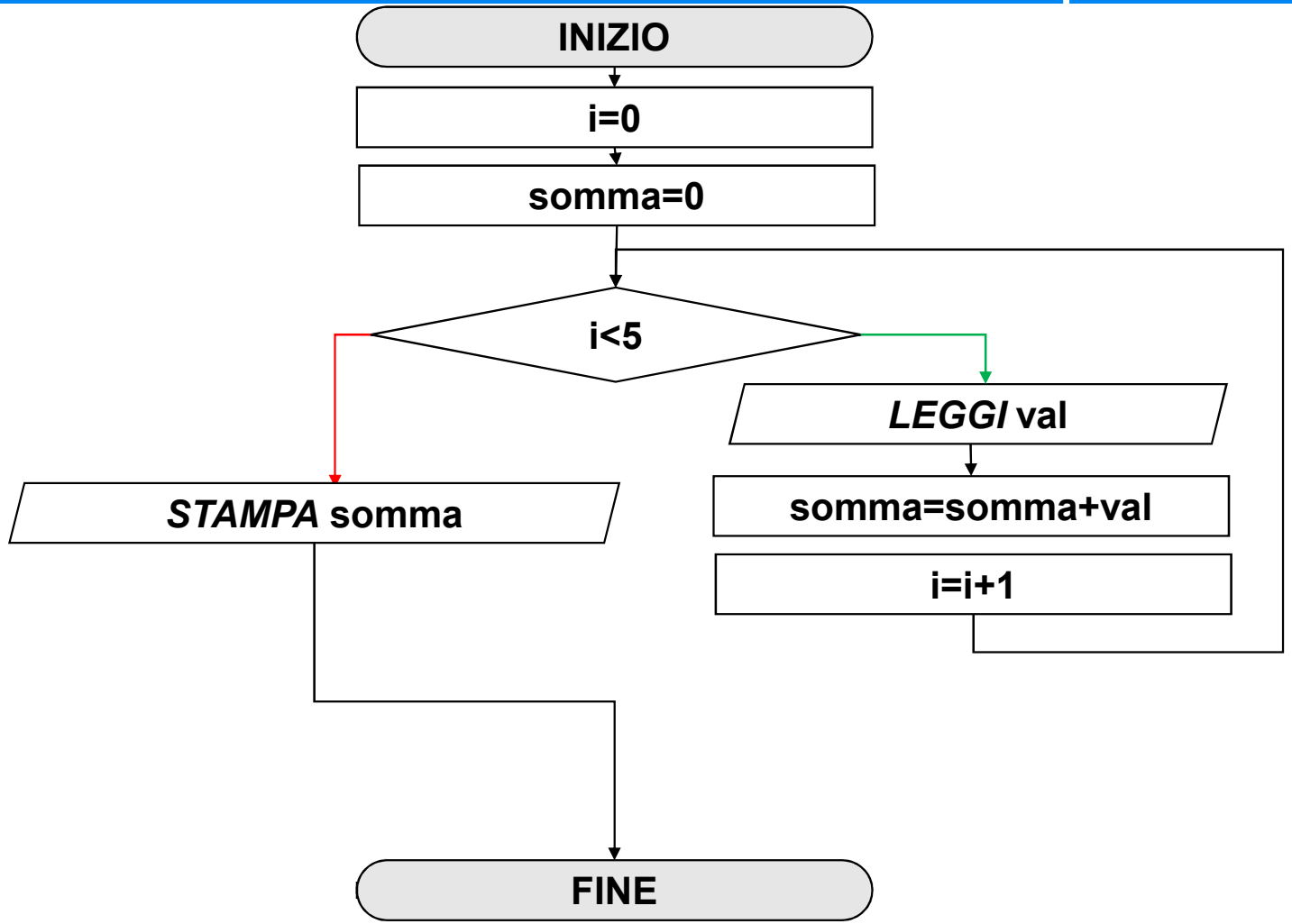
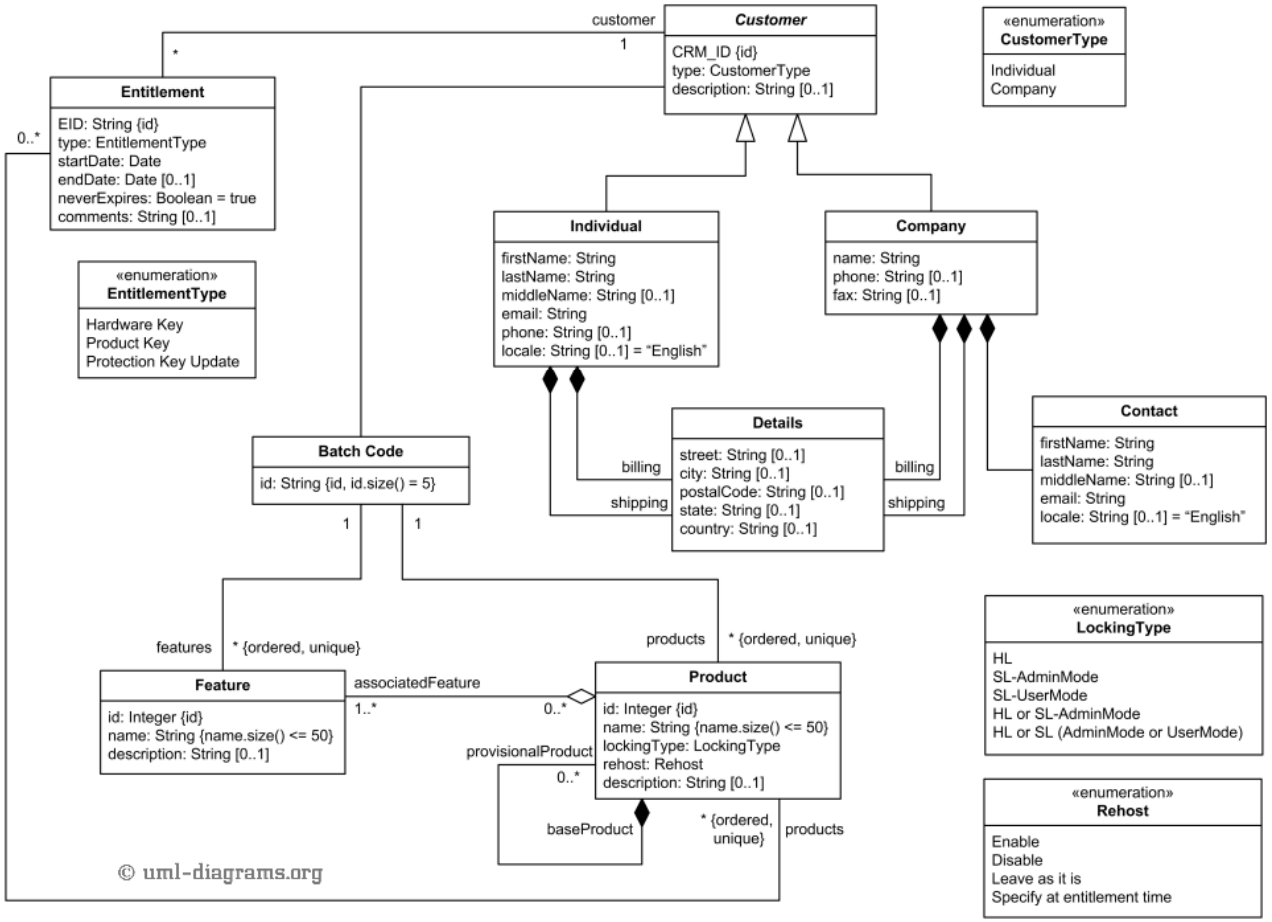
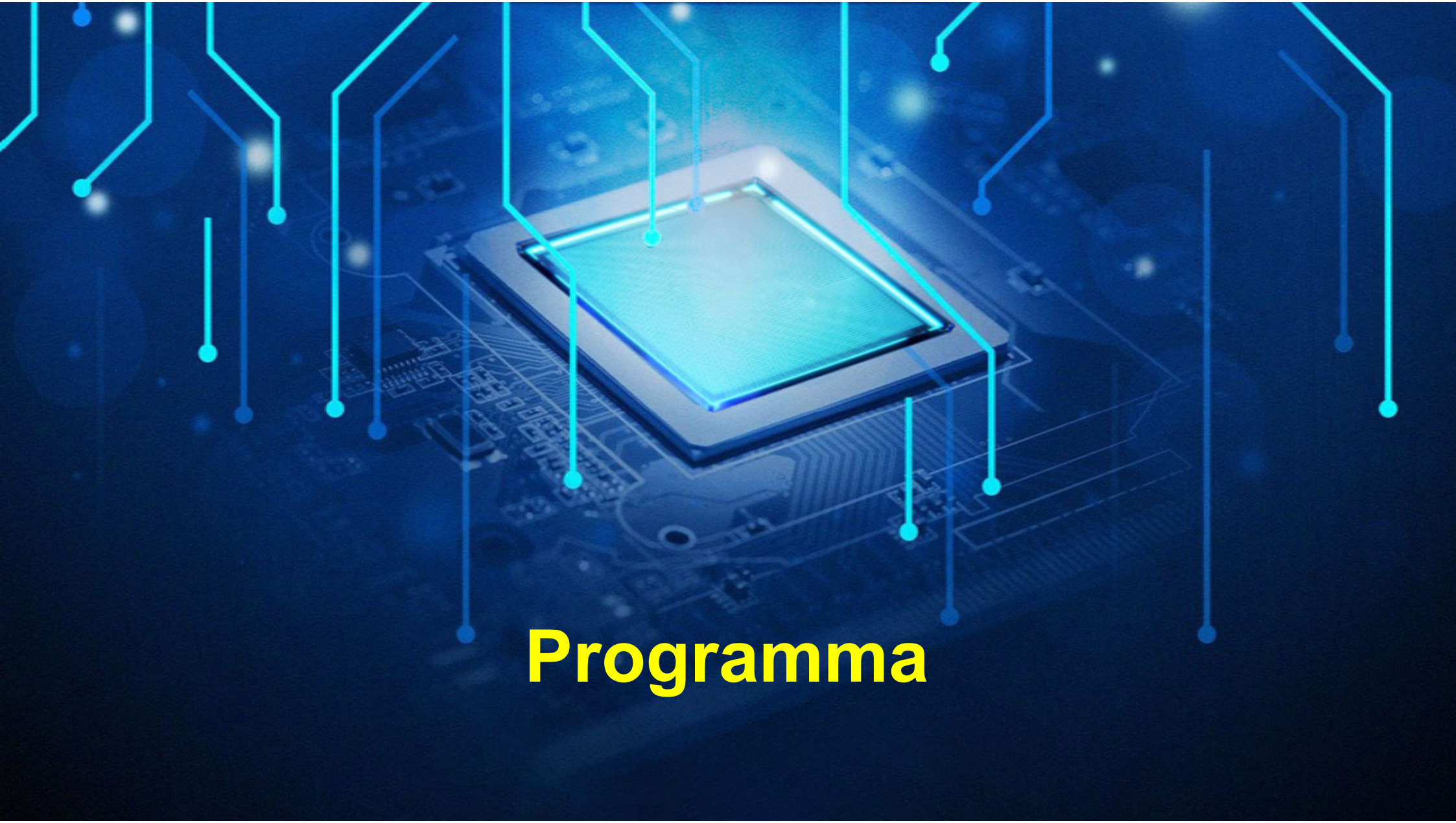


Diagramma di flusso

Un diagramma di flusso, come anche la descrizione in pseudocodice, sono **linguaggi di modellazione** (*modelling language*) ovvero espressioni formali che possono essere utilizzati per descrivere un qualsiasi sistema (anche di natura diversa da quella matematica ed informatica). Attualmente, con lo sviluppo della programmazione ad oggetti, ce ne sono di più sviluppati (*Unified Modeling Language*, UML; *Systems Modeling Language*, SysML; EXPRESS-G; *Interaction Flow Modeling Language*, IFML), ma sono trattati in altri corsi

.



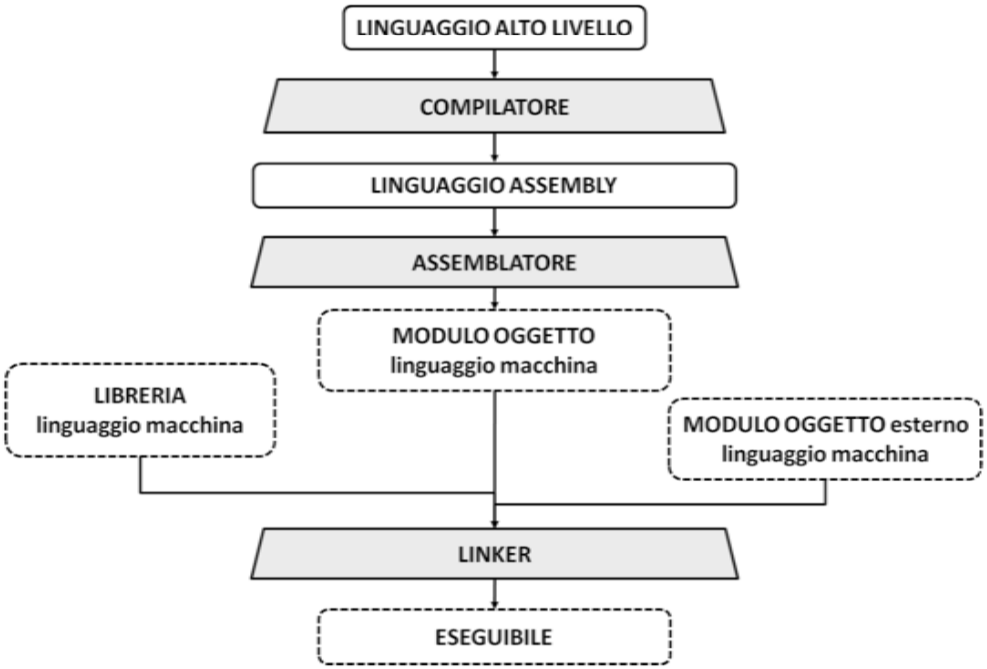

Programma

Programma

Dal linguaggio ad alto livello al codice eseguibile

Un programma, come visto, è una sequenza di istruzioni in grado di risolvere un determinato algoritmo. Il primo passo per realizzare un programma è la sua formalizzazione schematica e successivamente la scrittura delle istruzioni in un **linguaggio di alto livello** (es.: C, C+, Pascal), in cui la sintassi ha aspetti familiari con il lessico umano. In seguito questo codice è analizzato e tradotto fino ad arrivare ad un **linguaggio eseguibile dalla macchina**.

In alcuni casi queste azioni sono raggruppate per ridurre il tempo di traduzione, ma, escluso i linguaggi interpretabili, i programmi, per essere eseguiti, sono soggetti all'azione di tre procedimenti: compilazione, assemblaggio e collegamento

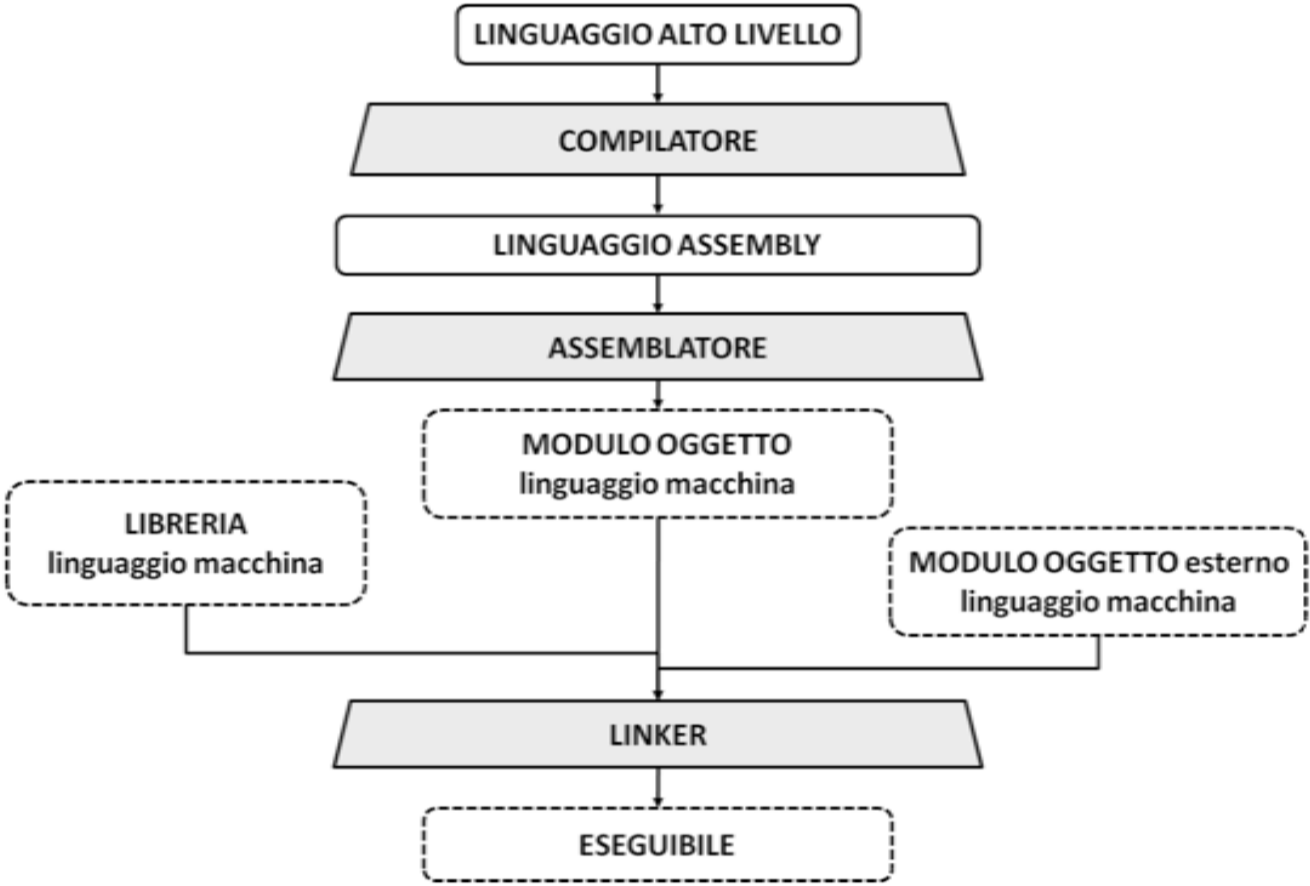


Programma

Compilatore

Durante la compilazione, il programma **compilatore** (*compiler*) trasforma, dopo un controllo sintattico, il programma scritto in un linguaggio ad alto livello in uno assemblativo, cioè in una forma simbolica che l'elaboratore è in grado di utilizzare ma, ancora, non eseguire.

Il compilatore può svolgere anche un'ottimizzazione del codice, eliminando le istruzioni inutili, o riordinando le istruzioni per mitigare le criticità sui dati e sul controllo nei sistemi canalizzati



Programma

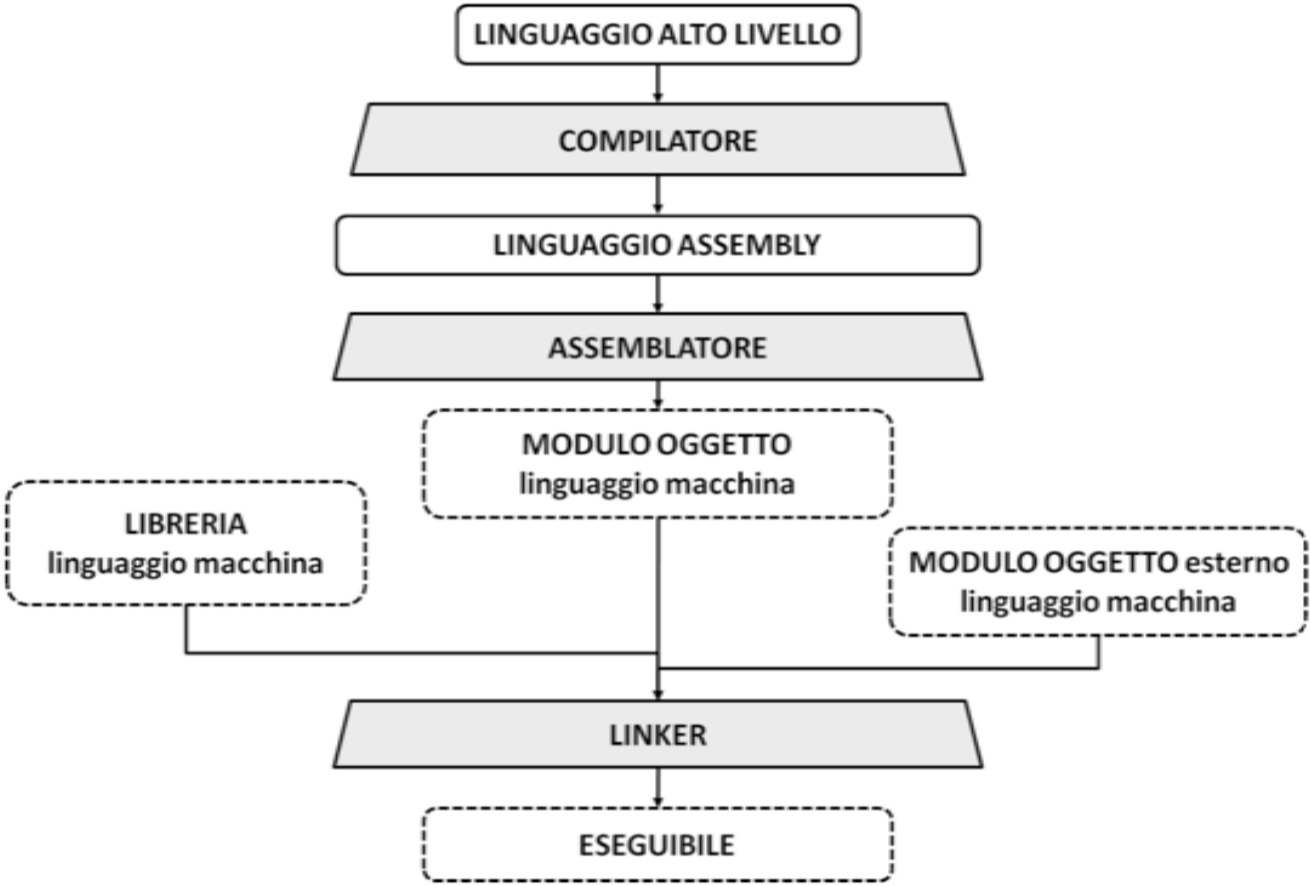
Assemblatore

Durante il processo di assemblaggio interviene il programma **assemblatore** (*assembler*) che converte un programma scritto in linguaggio assemblativo in un **file oggetto**

Un file oggetto è una raccolta d'istruzioni in linguaggio macchina, di dati e di informazioni necessarie a collocare le istruzioni in memoria in una determinata posizione (indirizzo iniziale, *start address*).

Il ruolo principale dell'assemblatore è di 'risolvere', cioè di sostituire, le etichette con gli indirizzi locali ovvero quelli interni al programma, supponendo di posizionare il programma a partire da una locazione prestabilita.

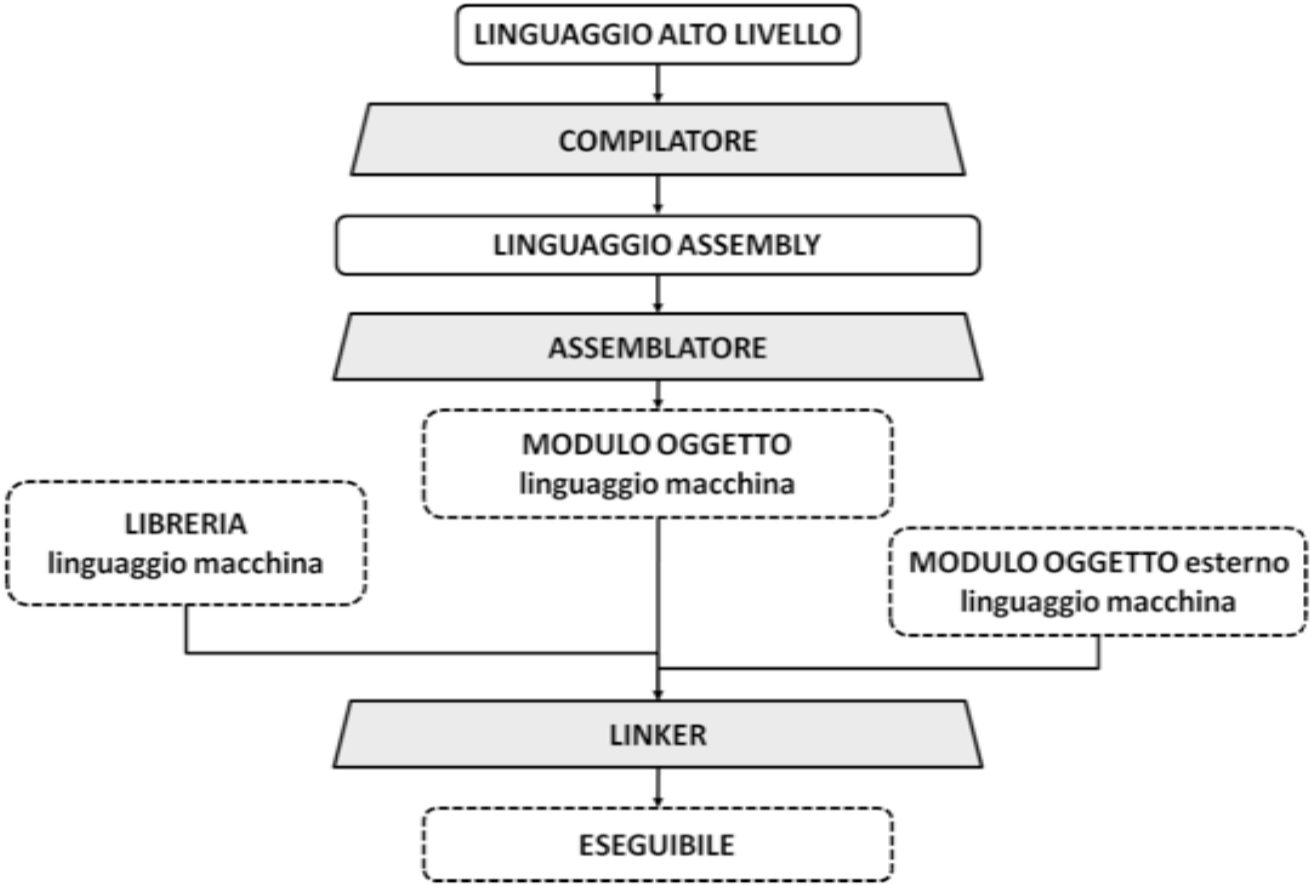
Inoltre **un pre-assemblatore riscrive le macro e risolve le pseudoistruzioni**



Programma

Assemblatore

L'assemblatore compie due passi logici sequenziali e in parte indipendenti; per questo motivo si parla di **assemblatore a doppia passata**.
Nel **primo passaggio** il programma in linguaggio assemblativo è letto sequenzialmente, si identificano, le direttive, le istruzioni e gli operandi, si calcola la lunghezza del codice (in base al numero e alla dimensione delle istruzioni) e si assegna un indirizzo a ciascuna istruzione (in base all'indirizzo iniziale preimpostato); inoltre, quando è letto un indirizzo simbolico (cioè una etichetta o una variabile) questo è inserito nella **Tabella dei Simboli** (*symbol table*) con uno schema <etichetta,indirizzo>.



Programma

Assemblatore

Le righe di una Tabella dei Simboli sono popolate in momenti diversi perché un simbolo può essere usato prima di essere definito (c'è l'etichetta, ma manca l'indirizzo).
Durante il **secondo passaggio** il programma in linguaggio assemblativo è ancora letto a partire dall'inizio e a tutte le etichette non ancora 'risolte' si sostituisce l'indirizzo corrispondente che in questa fase è sicuramente presente nella Tabella dei Simboli. Inoltre tutte le istruzioni e i relativi operandi ancora in forma simbolica (OPCODE, registri, etichette) sono tradotti in stringhe in linguaggio macchina.

Ogni istruzione è tradotta in una stringa binaria, cioè in linguaggio macchina con un rapporto uno a uno

PASSO 1

Etichetta	Indirizzo	Luogo di conservazione
n	Non risolto	
k	Non risolto	
somma	Non risolto	
ciclo	020	Memoria Istruzioni
fine	Non risolto	

PASSO 2

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
somma	308	Memoria Dati
ciclo	020	Memoria Istruzioni
fine	040	Memoria Istruzioni

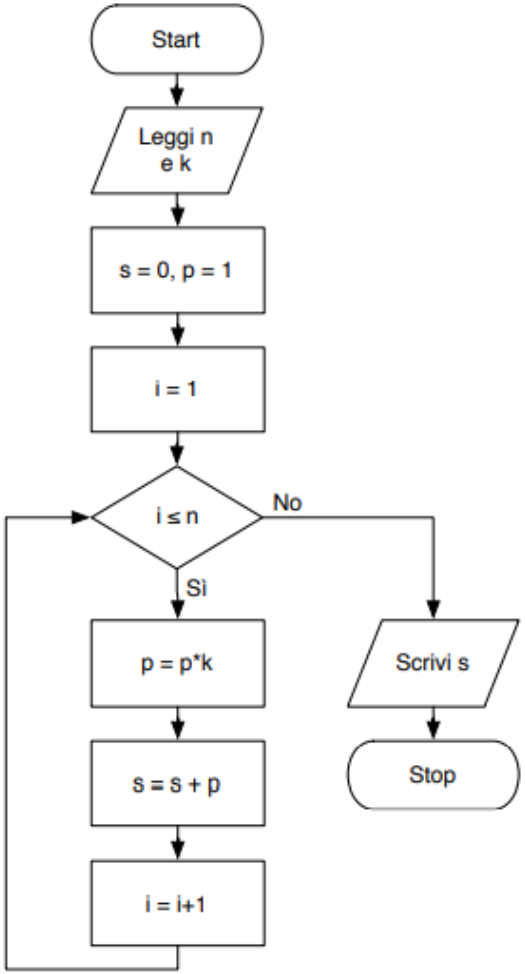
Programma

Esempio del processo di assemblamento a doppia passata

Calcolo della sommatoria di
$k+k^2+k^3+\ldots+k^n$
con n>0 e >0

Pseudo-codifica dell'algoritmo:

```
0: Start
1:          leggi n,k
2:          s=0
3:          p=1
4:          i=1
5:          fintanto che i ≤ n ripeti
6:                      p=p ·k
7:                      s=s + p
8:                      i=i +1
9:          fine-ciclo
10:         Memorizzare s
11: Stop
```



Programma

Esempio del processo di assemblamento a doppia passata

Calcolo della sommatoria di
$k+k^2+k^3+\ldots+k^n$
con n>0 e >0

Pseudo-codifica dell'algoritmo:
0: inizio
1: leggi n,k
2: s = 0
3: p = 1
4: i = 1
5: fintanto che i ≤ n ripeti
6: p = p ·k
7: s = s + p
8: i = i +1
9: fine-ciclo
10: Memorizzare s
11: stop

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020	ciclo:	bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j ciclo
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020	ciclo:	bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j ciclo
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020	ciclo:	bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j ciclo
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	
k	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020	ciclo:	bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j ciclo
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	
k	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020	*ciclo:*	bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j ciclo
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	
k	Non definito	
ciclo	020	Memoria Istruzioni
fine_ciclo	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	
k	Non definito	
ciclo	020	Memoria Istruzioni
fine_ciclo	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040	*fine_ciclo:*	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	
k	Non definito	
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	Non definito	
k	Non definito	
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	Non definito	
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	Non definito	

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	308	Memoria Dati

PASSO 1

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,300
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	308	Memoria Dati

PASSO 2

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,300
004		lb $t1,304
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	308	Memoria Dati

PASSO 2

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,300
004		lb $t1,304
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,040
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	308	Memoria Dati

PASSO 2

Programma

Esempio del processo di assemblamento a doppia passata

Indirizzo	Etichetta	Istruzione
000		lb $t0,300
004		lb $t1,304
008		li $t2,0
012		li $t3,1
016		li $t4,1
020		bgt $t4,$t0,040
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j 020
040		
044		sw $t2,308
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Etichetta	Indirizzo	Luogo di conservazione
n	300	Memoria Dati
k	304	Memoria Dati
ciclo	020	Memoria Istruzioni
fine_ciclo	040	Memoria Istruzioni
somma	308	Memoria Dati

PASSO 2

Programma

File oggetto

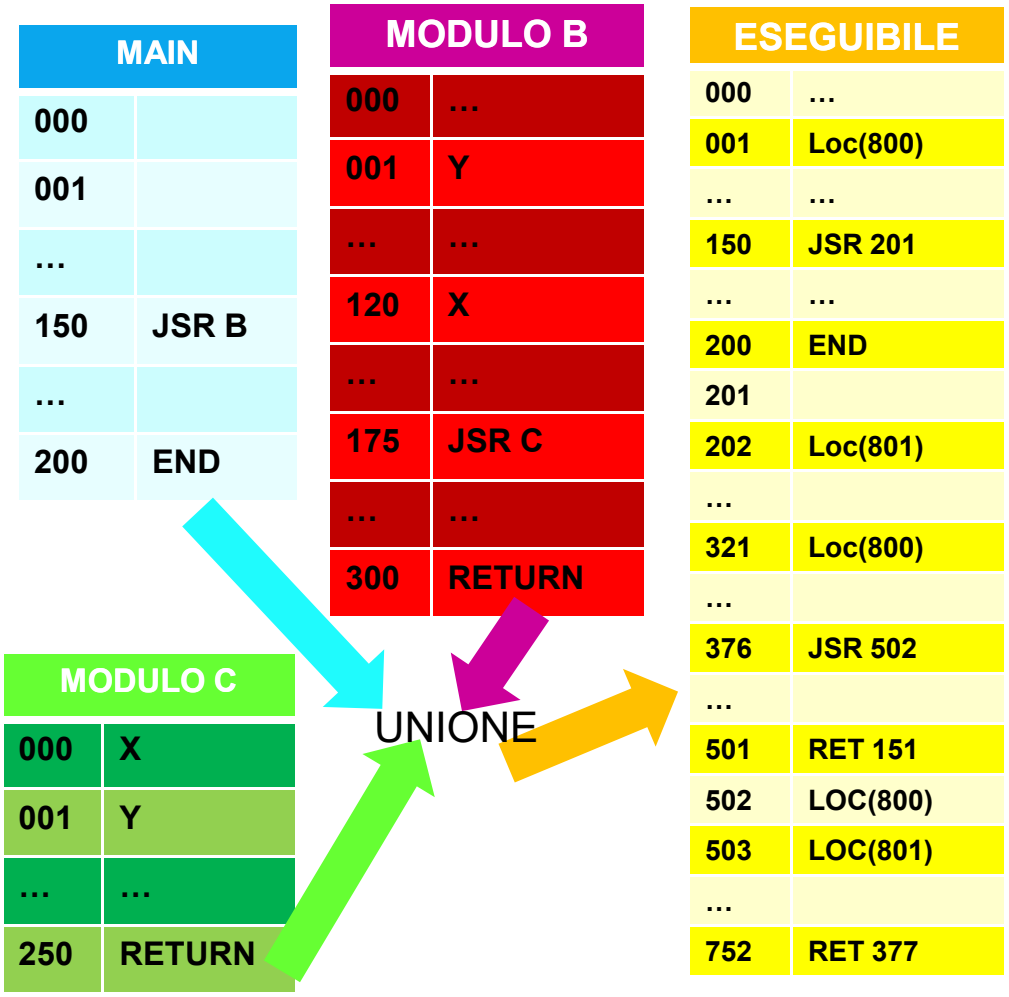
In generale il file oggetto è suddiviso in sezioni distinte. La separazione del codice prevede l'area *object file header*, in cui sono presenti informazioni sulla dimensione e sulla posizione delle altre sezioni del file oggetto; il *text segment*, nel quale sono contenute le istruzioni in linguaggio macchina del programma; il *data segment*, che ha la dichiarazione delle variabili e delle strutture dati; il *relocation information*, dove risiedono le istruzioni e i dati che dipendono da indirizzi assoluti e che devono essere rilocati dal linker.
Elemento comune per tutti i linguaggi assemblativi è la *symbol table* al cui interno sono presenti anche i simboli che non sono ancora risolti come le etichette che fanno riferimento a moduli esterni o alle variabili globali. Infine c'è il *debugging information*, che ha informazioni utili per l'analisi e il controllo delle anomalie durante l'elaborazione (*debugger*)

FILE OGGETTO
OBJECT FILE HEADER
TEXT SEGMENT
DATA SEGMENT
RELOCATION INFORMATION
SYMBOL TABLE
DEBUGGING INFORMATION

Programma

Linker

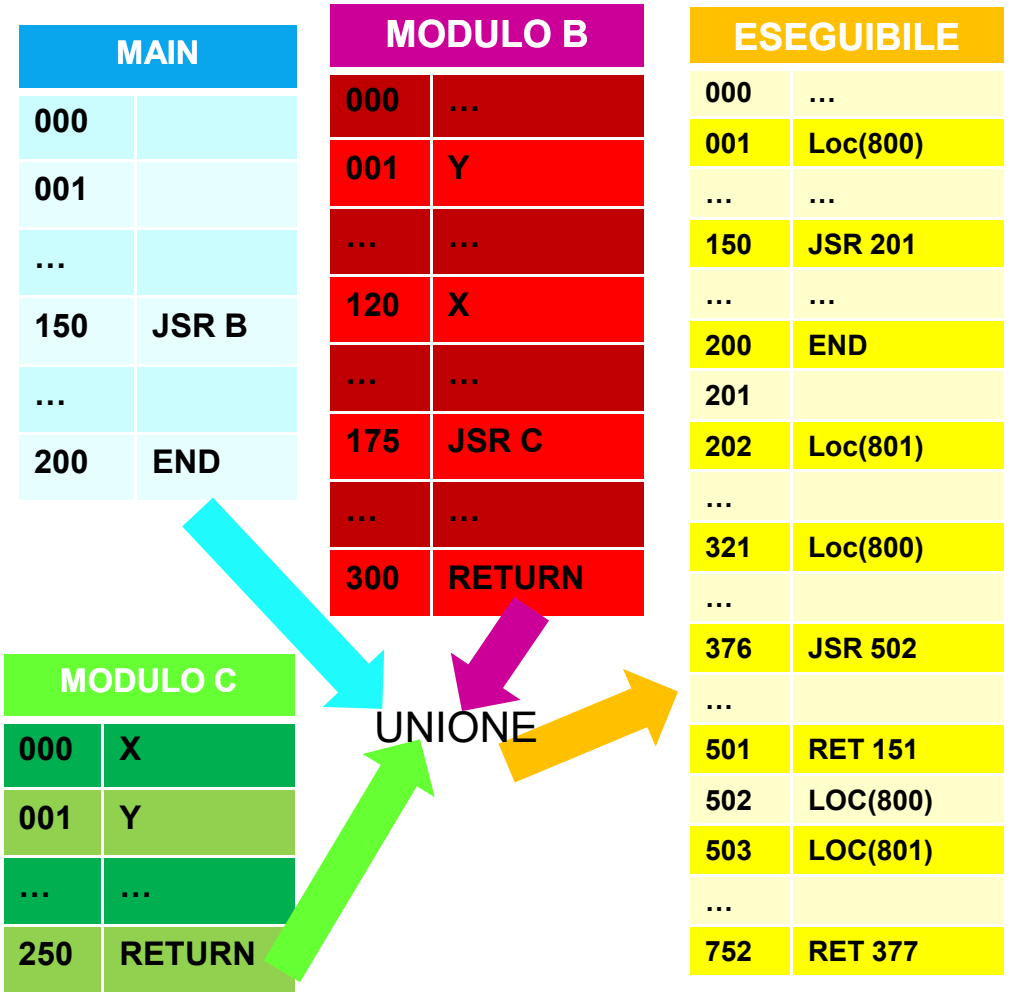
Il file oggetto potrebbe essere direttamente eseguito, ma se un programma è costituito da più moduli, il processo di compilazione e di assemblaggio deve essere ripetuto per ciascun modulo.
I differenti file oggetto sono messi in relazione (*linked*) tra di loro all'interno di unico **file eseguibile** grazie al **collegatore** (*linker*). Infine, il file eseguibile è caricabile in memoria ed elaborabile
Per ogni modulo tradotto separatamente l'indirizzo iniziale è lo stesso: è compito del linker modificare gli indirizzi di ciascun modulo affinché non ci siano sovrapposizioni.
Un eseguibile ha sempre un modulo principale, noto come *main*, mentre gli altri; se non hanno dipendenze particolari possono essere posizionati senza particolari accortezze (di solito si antepongono al *main*).



Programma

Linker: file eseguibile

La traslazione dell'indirizzo di ogni istruzione in ciascun modulo permette di unire tutti i moduli ma non è sufficiente: bisogna modificare in maniera consistente anche tutti gli indirizzi assoluti. In altre parole se un modulo ha un indirizzo di collegamento ad un altro modulo (riferimento esterno, *external reference*) è necessario ricalcolare quest'ultimo, in maniera coerente, in accordo alla nuova posizione del file eseguibile. Esempi di riferimenti esterni sono le variabili globali (che devono essere viste da tutti i moduli) e le chiamate tra subroutine appartenenti a moduli diversi. Per questo, il linker costruisce una **Tabella dei Moduli** grazie alla quale si procede alla rilocazione e al calcolo dei riferimenti esterni a ciascun modulo.



Programma

Linker file eseguibile

Il linker, alla fine, produce un **file eseguibile** che di norma ha la stessa struttura e rappresentazione binaria di un file oggetto, ma non contiene riferimenti non risolti, non ha informazioni di rilocazione ed è privo della tabella dei simboli

Dal linguaggio ad alto livello a quello macchina

Linguaggio ad alto livello	Linguaggio assembly (MIPS)	Linguaggio Macchina

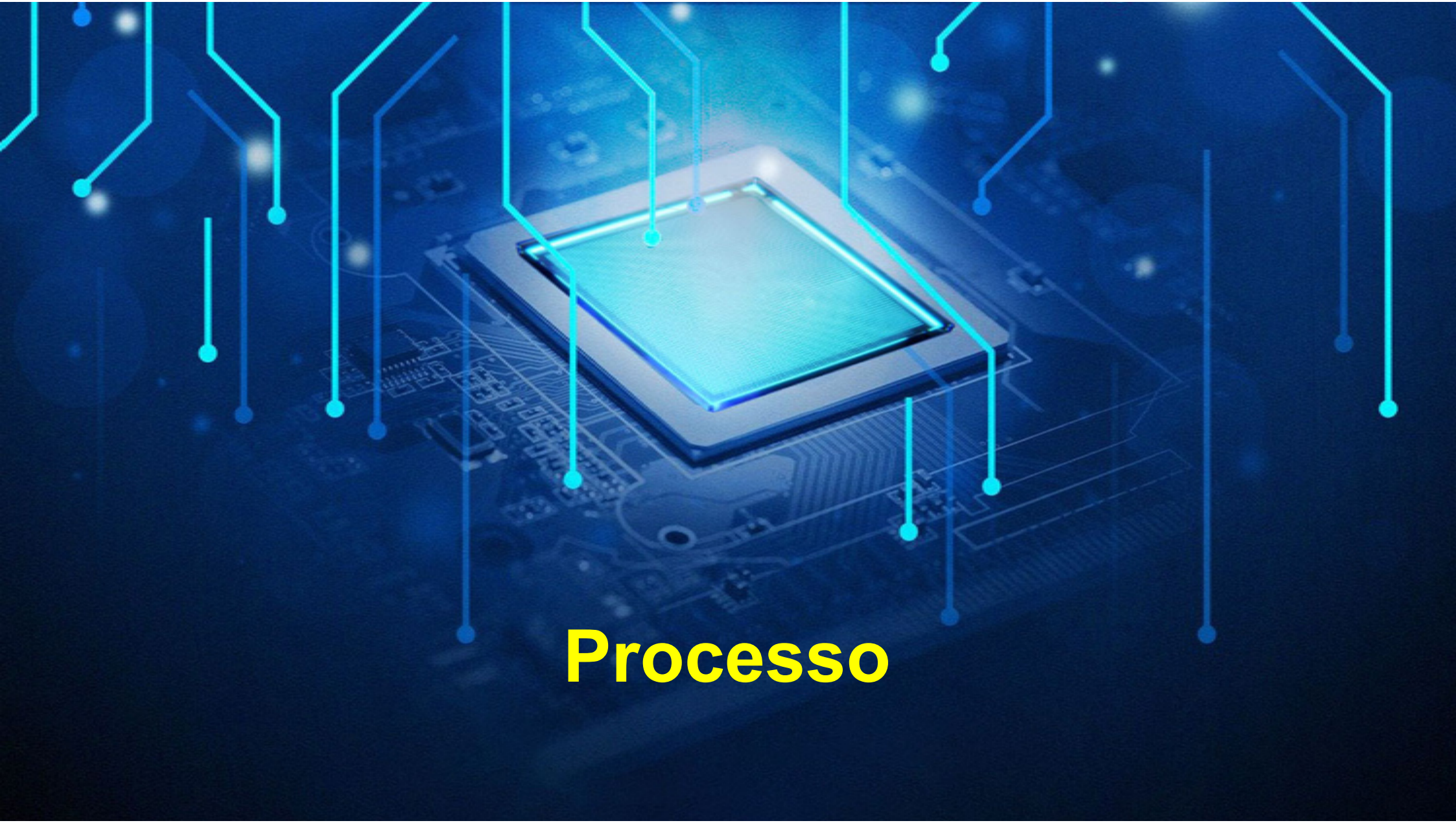
Main ()
{
int ris=Pow (2,3);
}
int Pow(int b,int e)
{
int t=1;
for(i=0;i<e;i++)
{
t=t·b;
}
return(t);
}
	.text
	.globl main
	main:
	lw $a0,base #caricamento base
	lw $a1,espo #caricamento esponente
	jal pow #salto a funzione
	sw $v0,ris
	li $v0,10
	syscall
	#FUNZIONE
	pow:
	li $t0,0 #inizializzazione contatore
	li $t1,1 #inizializzazione risultato
	move $t3,$a0
	ciclo:
	bge$t0,$a1,fine
	mul $t1,$t1,$t3
	add $t0,$t0,1
	j ciclo
	fine:
	move $v0,$t1
	jr $ra
	.data
	base: .word 2
	espo: .word 3
	ris: .word 0
		00000000000001001001100000101100
		10001100001001000000000000000000
		00111100000000010001000000000001
		10001100001001010000000000000100
		00001100000100000000000000001001
		00111100000000010001000000000001
		10101100001001100000000000001000
		00110100000000100000000000001010
		00000000000000000000000000001100
		00000010000010000000010100000000
		00110100000010010000000000000001
		00000000000001000101100000100001
		00000001000001010000100000101010
		00010000001000000000000000000100
		01110001001010110100100000000010
		00100001000010000000000000000001
		00001000000100000000000000001100
		00000000000010010011000000100001
		00000011111000000000000000001000

Programma

Caricatore

Il file eseguibile è memorizzato su un supporto di massa (generalmente il disco magnetico o la memoria a stato solido) e per essere eseguito deve essere trasferito in memoria. Questo compito è svolto dal **caricatore** (o *loader*), un sottoprogramma del Sistema Operativo. Il caricatore, oltre a creare le informazioni sul processo, può sfruttare lo spazio in memoria a seconda del tipo di allocazione: monoprogrammazione o multiprogrammazione.

Il loader del MIPS compie: la lettura dell'intestazione del file eseguibile, per determinare la dimensione del *text segment* e del *data segment*; lo spostamento della porzione *text segment* nella Memoria delle Istruzioni e quella del *data segment* nella Memoria dei Dati; la creazione di un'area riservata allo stack; la copia nello stack di eventuali argomenti iniziali passati al programma (*argv*, *argc*); l'inizializzazione dei registri della CPU; e, infine, il richiamo alla **direttiva di inizio** (BEGIN), da cui si comincia ad eseguire il programma fino alla **direttiva di terminazione** (END). Ciascuna istruzione in formato binario MIPS è ospitata in una unica locazione di memoria del text segment perché a dimensione fissa e quindi ad ogni istruzione è associato un indirizzo univoco di residenza. Come mostrato più avanti, la direttiva di inizio può essere modificata dal programmatore (facendo riferimento alla funzione principale).


Processo

Processo

Un file eseguibile, presente nella memoria (nel caso del MIPS nella Memoria delle Istruzioni) e in corso di esecuzione è detto **processo**. Ogni processo ha un **blocco di informazioni** associate (*process control block*, PCB) che lo descrive. I campi principali sono:

- Stato del Processo: *new*, *ready*, *running*, *waiting* e *halted*.
- Contenuto dei registri della CPU: sono le informazioni salvate quando si verifica una interruzione;
- Informazioni sull'uso (*scheduling*) della CPU, come, ad esempio, la priorità del processo;
- Informazioni inerenti al modulo di gestione della memoria (i registri base e limite, la tabelle delle pagine, ed altro);
- Informazioni di contabilità e statistiche (es.: il tempo di uso, l'intervallo temporale assegnato, ed altro);
- Informazioni di interazione con le Unità di Ingresso e di Uscita (es.: l'elenco dei file aperti, le periferiche attive, le richieste di I/O, i supporti di memorizzazione).

Organizzazione Memoria MARS

I MARS ha una Memoria suddivisa logicamente in segmenti. A partire dalla locazione 4194304 è presente il **Text Segment**; con inizio 268500992 c'è il **Data Segment**. Dalla locazione 268697600 inizia l'area **heap**, ovvero la parte riservata per le strutture dati dinamiche; mentre lo **stack** comincia da 2147479529; l'area in mezzo è occupata in base alle necessità. Oltre a questo ci sono anche delle locazioni per la memorizzazione temporanea dei dati del **kernel** (da 2415919104) e per lo scambio di informazioni con le periferiche **MMIO** (da 4294902016).

MEMORIA MARS
RESERVED
TEXT
DATA
HEAP
FREE
STACK
KDATA
MMIO


FINE