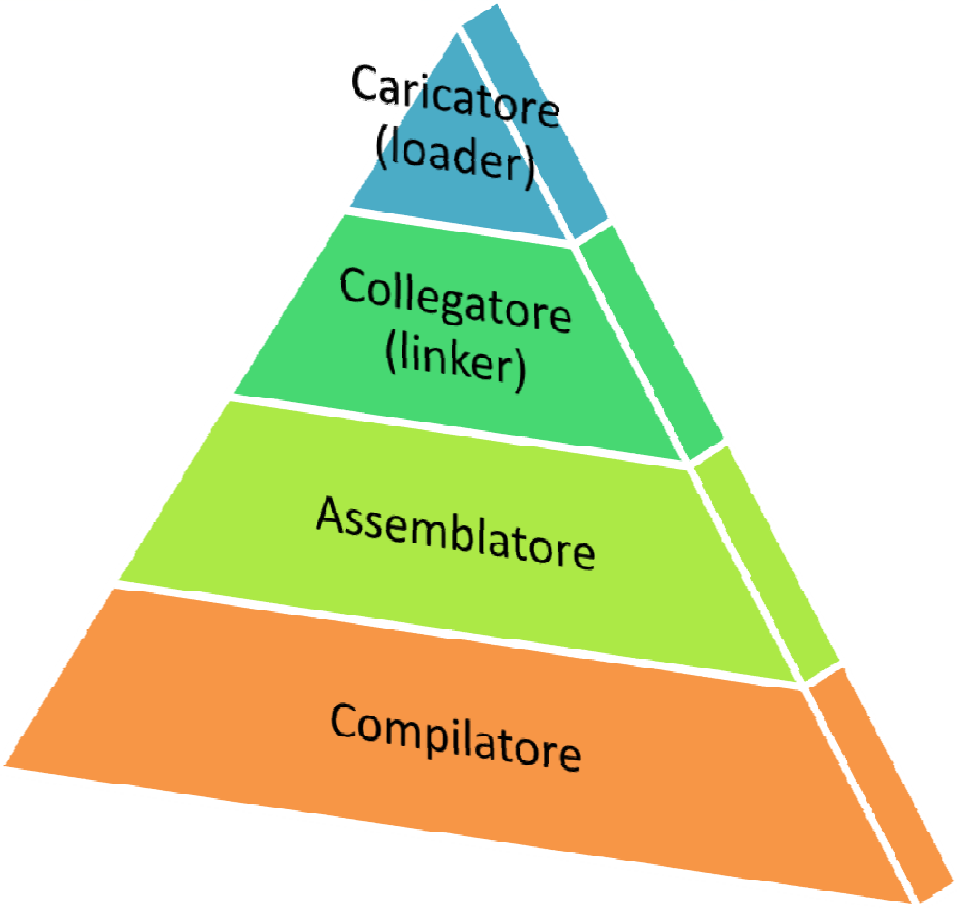
Architettura degli elaboratori

Assemblatore

Dott. Franco Liberati

ARGOMENTI DELLA LEZIONE

- Compilatore
- Assemblatore
- Collegatore (linker)
- Caricatore (loader)



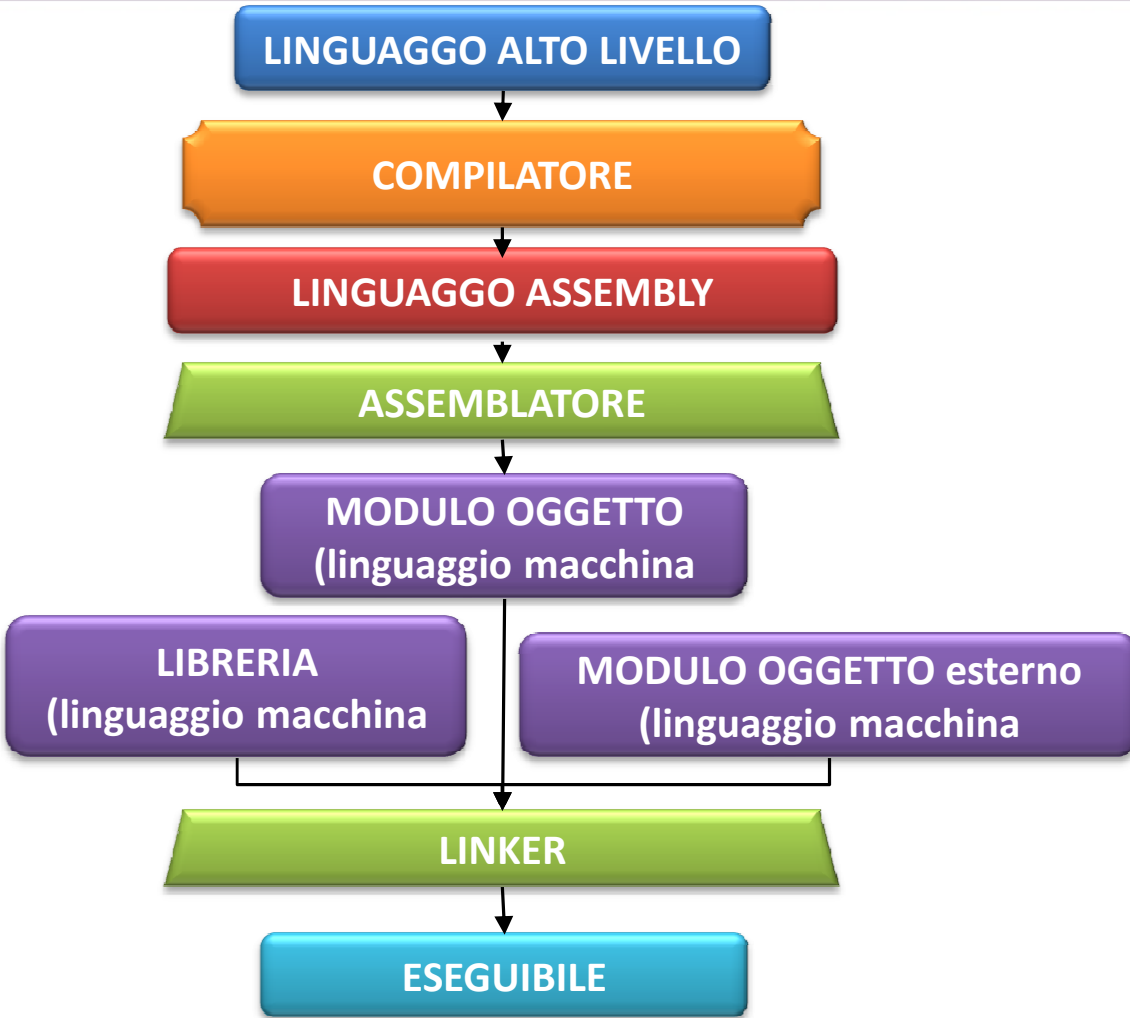
Architettura degli elaboratori

Assemblatore

PROGRAMMA

Generalità

- L'esecuzione di un programma è il punto di arrivo di una sequenza di azioni che nella maggior parte dei casi iniziano con la scrittura di un programma in un linguaggio simbolico di alto livello
- Le azioni principali che compongono tale sequenza nel caso in cui si parta da un linguaggio ad alto livello sono quelle che vedono in gioco il compilatore, l'assemblatore e il collegatore (linker)
- A volte queste azioni per ridurre il tempo di traduzione, ma concettualmente tutti i programmi passano sempre attraverso le fasi mostrate



PROGRAMMA

Compilatore

- Il **compilatore** trasforma, dopo un controllo sintattico, il programma scritto in un linguaggio ad alto livello in uno in linguaggio assembly, cioè in una forma simbolica che il calcolatore è in grado di capire ma, ancora, non eseguire
- Il compilatore inoltre ottimizzare il codice: ritardo dei salti, nel caso di macchine canalizzate; elimina le istruzioni inutili; stabilisce una visibilità delle locazioni di memoria in cui risiedono gli operandi (variabili private, pubbliche,….)

PROGRAMMA

Compilatore

Linguaggio ad alto livello (linguaggio C)

```
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);
}
```

Linguaggio assembly (SPIM)

```
.text
.globl main
main:
                lw $a0,base     #caricamento valore
                lw $a1,espo     #caricamento valore
                jal POW         #salto a funzione
                sw $a2,ris      #spostamento risultato in memoria
                li $v0,10
                syscall

POW:
                li $t0,0        #inizializzazione contatore
                li $t1,1        #inizializzazione risultato temporaneo
                move $t3,$a0
ciclo:
                bge $t0,$a1,fine #confronto contatore-esponente
                mul $t1,$t1,$t3  #moltiplicazione per la base
                addi $t0,1          #incremento contatore
                j ciclo             #salto
fine:
move $a2,$t1
jr $ra                          #ritorno a funzione
```

```
.data
base: .word 2
espo: .word 3
ris: .word 0
```

PROGRAMMA

Assemblatore

- L'**assemblatore** converte un programma assembly in un **file oggetto**, che è una combinazione di istruzioni in linguaggio macchina, di dati e di informazioni necessarie a collocare le istruzioni in memoria nella posizione opportuna
- Come prima fase, il **pre-assembler** procede a riscrivere le macro e a risolvere le pseudoistruzioni
- Un programma assembly è tradotto in linguaggio oggetto attraverso il **processo di assemblaggio** (assembler) costituito da due passi logici successivi ed in parte indipendenti:
  1. il programma assembly è letto sequenzialmente, si identificano le istruzioni e i loro operandi, si calcola la lunghezza e si assegna un indirizzo (relativo) a ciascuna istruzione; inoltre, quando è letto un simbolo (un indirizzo simbolico quale una label o una variabile), nome e indirizzo vengono inseriti in una **tabella dei simboli** (symbol table): nome e indirizzo di un simbolo possono essere inseriti nella symbol table in momenti diversi se un simbolo viene usato prima di essere definito
  2. il programma assembly è letto sequenzialmente, a tutti i simboli è sostituito il valore numerico corrispondente presente nella symbol table, a tutte le istruzioni e ai relativi operandi ancora in forma simbolica viene sostituito il valore numerico corrispondente (opcode, ecc.).

PROGRAMMA

Assemblatore doppio passaggio (esempio)

Indirizzo	Istruzione
100	lw $t0,Val_x
101 ciclo:	beqz $t0,salto
102	add $t0,$t0,1
103	j ciclo
104 salto:	sw $t0,Val_y

300 Val_x: .word 56
304 Val_y:.word 67

PASSO I

Indirizzo	Istruzione
100	lw $t0,Val_x
101 ciclo:	beqz $t0,salto
102	add $t0,$t0,1
103	j 101
104 salto:	sw $t0,Val_y

300 Val_x: .word 56
304 Val_y:.word 67

TABELLA SIMBOLI

Etichetta	Locazione
Val_x	300
ciclo	101
salto	104
Val_y	304

PASSO II

Indirizzo	Istruzione
100	lw $t0,300
101	beqz $t0,104
102	add $t0,$t0,1
103	j 101
104	sw $t0,304

300 Val_x: .word 56
304 Val_y:.word 67

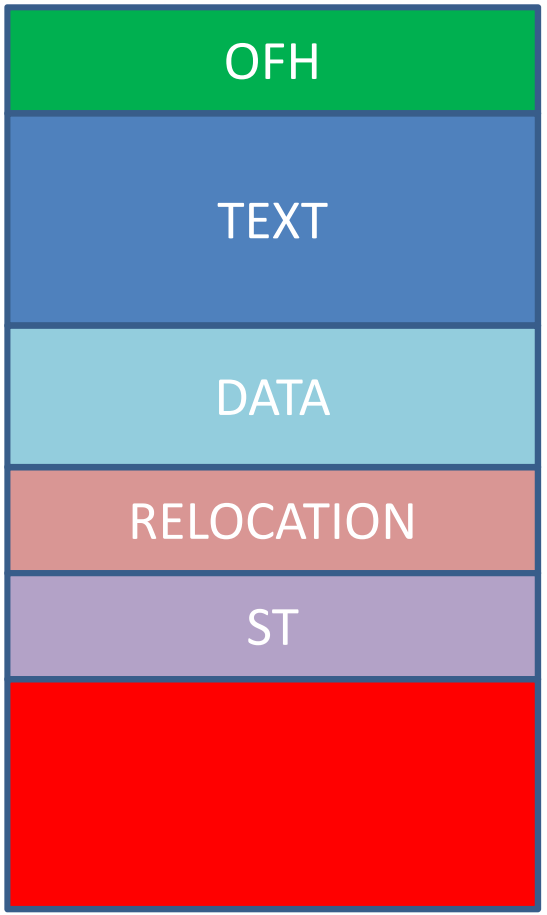
TABELLA SIMBOLI

Etichetta	Locazione
Val_x	300
ciclo	101
salto	104
Val_y	304

PROGRAMMA

Disposizione dei file oggetto in memoria

- I file oggetto sono suddivisi in sei sezioni distinte:
  1. ***object file header*** (OFH): descrive la dimensione e la posizione delle altre sezioni del file oggetto;
  2. ***text segment*** (TEXT): contiene le istruzioni in linguaggio macchina;
  3. ***data segment*** (DATA): contiene tutti i dati che fanno parte del programma;
  4. ***relocation information*** (RELOCATION): identifica le istruzioni e i dati che dipendono da indirizzi assoluti e che dovranno essere rilocati dal linker
  5. ***symbol table*** (ST): contiene ancora i simboli che non sono ancora definiti, ad esempio le etichette che fanno riferimento a moduli esterni;
  6. ***debugging information*** (DEBUG): contiene informazioni per il debugger.



PROGRAMMA

Collegatore (Linker)

- Quando un programma simbolico è costituito da più moduli contenuti in diversi file sorgenti, il processo di traduzione (compilazione e assemblaggio) è ripetuto per ciascun modulo
- I **file oggetto** (object) risultanti devono essere **collegati** (linked) opportunamente tra di loro all'interno di unico file eseguibile, che solo allora può essere caricato in memoria
- Per ogni modulo tradotto separatamente l'indirizzo iniziale è lo stesso: è compito del linker modificare gli indirizzi di ciascun modulo in modo che non ci siano sovrapposizioni

Modulo A	
000	...
001	x
...	...
150	Jsr B
...	...
200	...

Modulo C	
000	x
001	y
...	...
250	Ret B

Modulo B	
000	...
001	y
...	...
120	x
...	...
175	Jsr C
...	...
300	Ret A

Eseguibile	
000	...
001	x
...	...
150	Jsr 201
...	...
200	...
201	...
202	y
...	...
321	Loc (001)
...	...
376	Jsr 502
...	...
501	Ret 151
502	Loc (001)
503	Loc (202)
...	...
752	Ret 377

PROGRAMMA
Collegatore (Linker)

- La traslazione dell'indirizzo di ogni istruzione in ciascun modulo permette di unire tutti i moduli ma non è sufficiente, infatti: è necessario traslare in maniera consistente anche tutti gli indirizzi (assoluti) che compaiono come operandi
- Per ogni riferimento da parte di un modulo a un indirizzo di un altro modulo è necessario calcolare coerentemente l'**indirizzo esterno** (riferimento esterno o *external reference*). Esempi in questo senso sono le **variabili globali** (che devono essere viste da tutti i moduli) e le chiamate tra procedura appartenenti a moduli diversi
- Per questo, il linker costruisce una **tabella dei moduli** grazie alla quale è possibile procedere alla rilocazione e al calcolo dei riferimenti esterni a ciascun modulo
- **Il linker produce un file eseguibile che di norma ha la stessa struttura di un file oggetto, ma non contiene riferimenti non risolti**

Modulo A	
000	...
001	x
...	...
150	Jsr B
...	...
200	...

Modulo C	
000	x
001	y
...	...
250	Ret B

Modulo B	
000	...
001	y
...	...
120	x
...	...
175	Jsr C
...	...
300	Ret A

Eseguibile	
000	...
001	x
...	...
150	Jsr 201
...	...
200	...
201	...
202	y
...	...
321	Loc (001)
...	...
376	Jsr 502
...	...
501	Ret 151
502	Loc (001)
503	Loc (202)
...	...
752	Ret 377

PROGRAMMA

Caricatore (loader)

- Una volta che il file eseguibile è memorizzato sul supporto di massa (generalmente il disco magnetico), il loader (attraverso il sistema operativo) può caricarlo in memoria per l'esecuzione e quindi effettuare:
  - la lettura dell'intestazione del file eseguibile per determinare la dimensione dei segmenti testo (istruzioni) e dati
  - la creazione un nuovo spazio di indirizzamento, grande a sufficienza per contenere istruzioni, dati e stack
  - la copia delle istruzioni e i dati dal file al nuovo spazio di indirizzamento
  - la copia sullo stack degli eventuali argomenti del programma
  - l'inizializzazione dei registri della CPU
  - l'inizio dell'esecuzione a partire da una direttiva di inizio che copia gli argomenti del programma dallo stack agli opportuni registri e che chiama la funzione main() o la **direttiva di inizio** (BEGIN); fino ad una **direttiva di terminazione** (END)

PROGRAMMA

Eseguibile

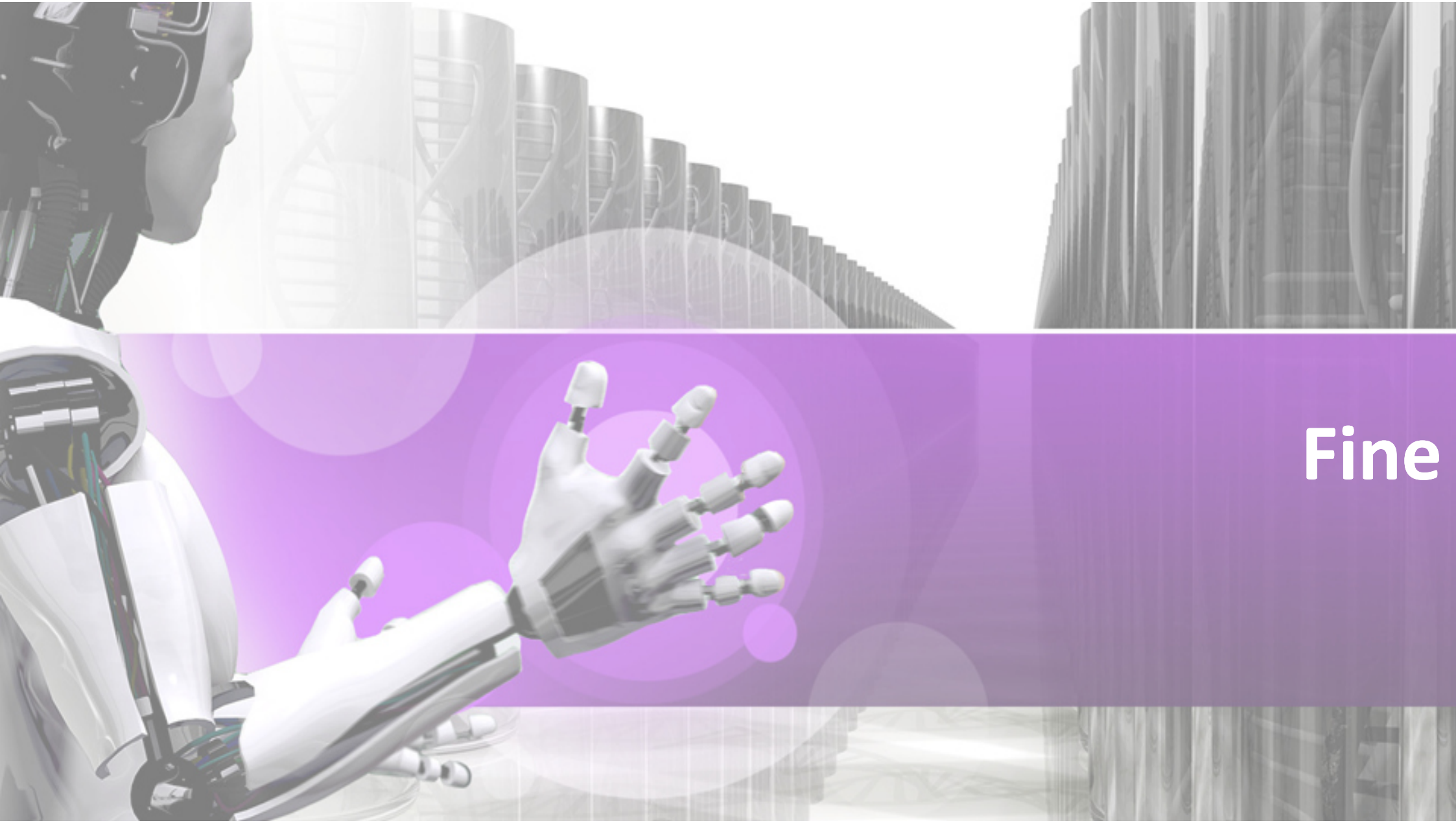
Linguaggio assembly (MIPS)

```
.text
.globl main
main:
lw $a0,base
lw $a1,espo
jal POW
sw $a2,ris
li $v0,10
syscall
POW:
li $t0,0
li $t1,1
move $t3,$a0
ciclo:
                bge $t0,$a1,fine
                mul $t1,$t1,$t3
                j ciclo
fine:
move $a2,$t1
jr $ra
.data
base: .word 2
espo: .word 3
ris: .word 0
```

Linguaggio macchina (MIPS)

Area istruzioni (al netto delle syscall)

```
0x40000000  00000000000001001001100000101100
0x40000004  10001100001001000000000000000000
0x40000008  00111100000000010001000000000001
0x4000000c  10001100001001010000000000000100
0x4000000e  00001100000100000000000000001001
0x40000010  00111100000000010001000000000001
0x40000014  10101100001001100000000000001000
0x40000018  00110100000000100000000000001010
0x4000001c  00000000000000000000000000001100
0x4000001e  00000010000010000000010100000000
0x40000020  00110100000010010000000000000001
0x40000024  00000000000001000101100000100001
0x40000028  00000001000001010000100000101010
0x4000002c  00010000001000000000000000000100
0x4000002e  01110001001010110100100000000010
0x40000030  00100001000010000000000000000001
0x40000034  00001000000100000000000000001100
0x40000038  00000000000001001001100000100001
0x4000003c  00000011111000000000000000001000
```


Fine