Architettura degli Elaboratori Elettronici

Esercitazioni

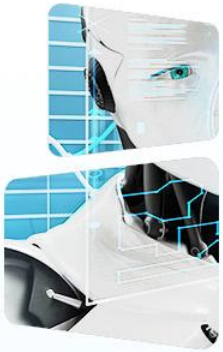
Linguaggi di programmazione

Franco Liberati

liberati@di.uniroma1.it

LINGUAGGI DI PROGRAMMAZIONE

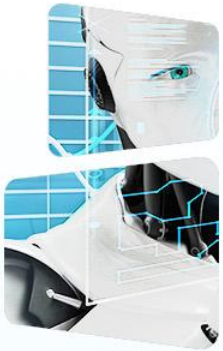
Dal codice sorgente al codice eseguibile



Linguaggi di programmazione

Definizione

- Un **linguaggio di programmazione** è un linguaggio formale che specifica un insieme di istruzioni le quali sono usate per eseguire un algoritmo e produrre un risultato

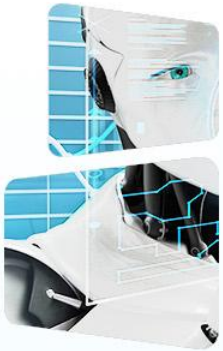


Linguaggi di programmazione

Storia

- Il primo linguaggio di programmazione della storia è il «linguaggio meccanico» adoperato da **Ada Byron Lovelace** per la programmazione della macchina di Charles Babbage nel 1842, si trattava di una sorta di linguaggio assemblativo in grado di mascherare istruzioni e i comandi binari utili per attivare la macchina di Babbage (studi analoghi furono condotti precedentemente da Luigi Federico Menabrea)

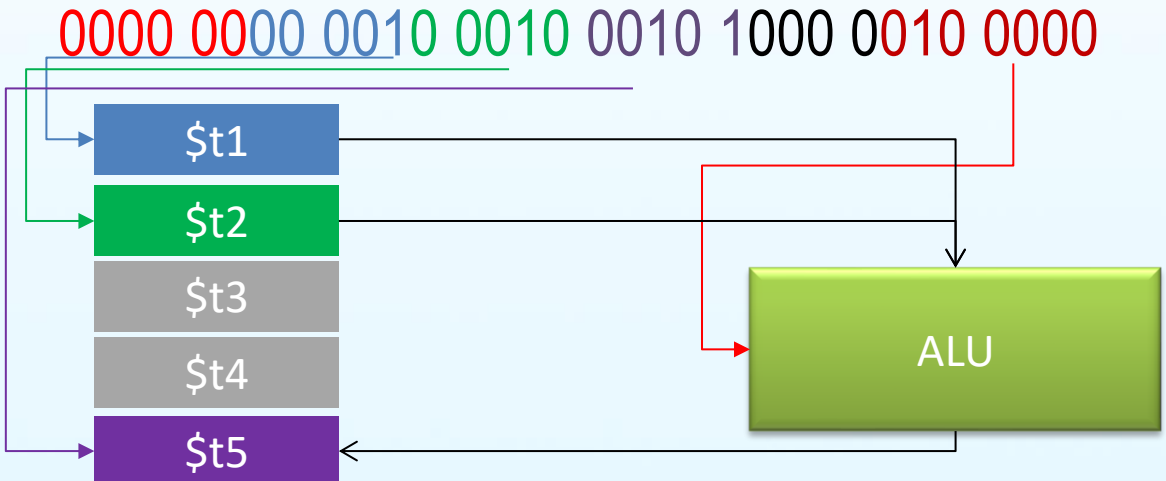
- I sistemi di calcolo automatico degli anni Venti e Trenta del XX secolo impiegano **il linguaggio macchina**, in cui è fondamentale la conoscenza dell'architettura della macchina

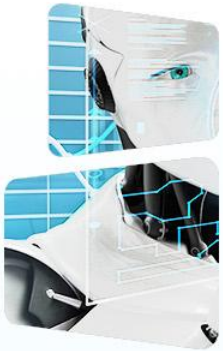


Linguaggi di programmazione

Linguaggio Macchina

- Il **linguaggio macchina** è basato su codici (stringhe binarie), con il significato di istruzioni, operandi e indirizzi ed è utilizzato dall'elaboratore per eseguire programmi

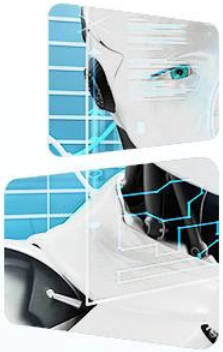




Linguaggi di programmazione

Storia

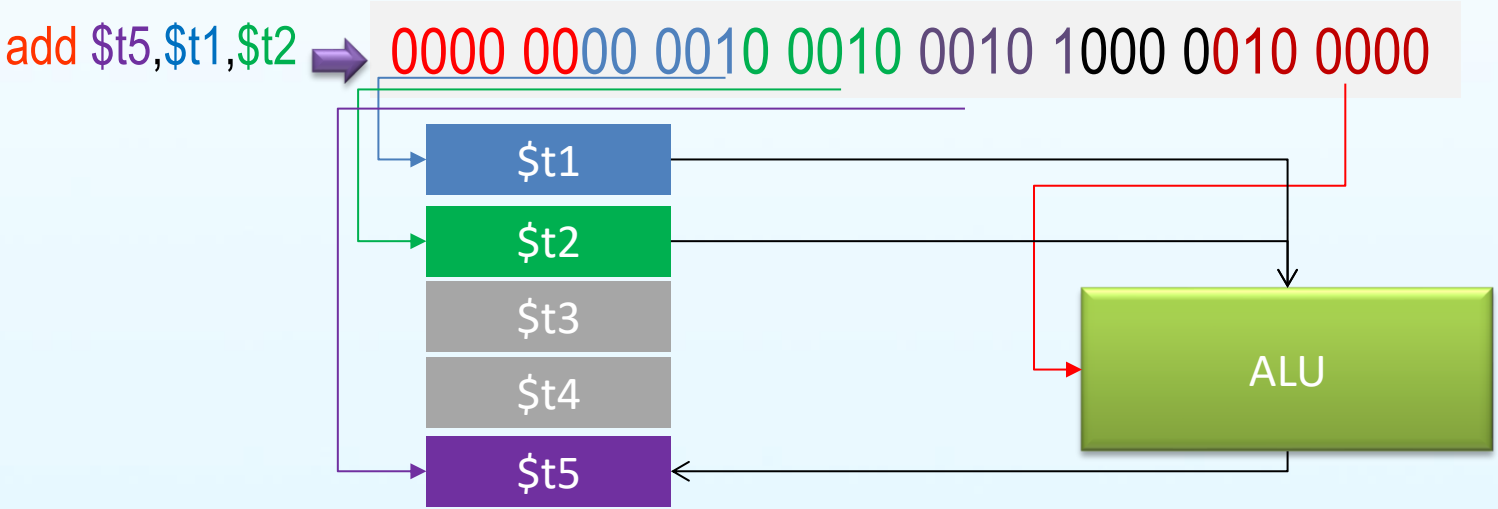
- Dopo la seconda Guerra Mondiale con lo sviluppo dell'informatica si usa il **linguaggio assemblativo** (assembly) il quale introduce una rappresentazione simbolica del linguaggio macchina

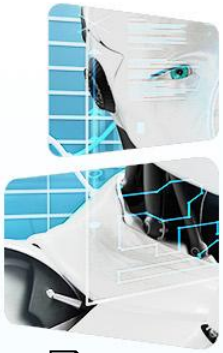


Linguaggi di programmazione

Linguaggio Assemblativo

- Il linguaggio assemblativo prevede una rappresentazione dei codici binari del linguaggio macchina con dei **mnemonici**: si utilizzano simboli invece di stringhe binarie per rappresentare istruzioni, operandi, indirizzi e registri



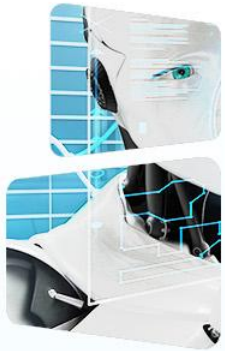


Linguaggi di programmazione

Linguaggio Assemblativo – Struttura di una istruzione

- Una **istruzione** di un generico linguaggio assemblativo è strutturata in campi:
  - **INDIRIZZO**: l'indirizzo dove è memorizzata l'istruzione
  - **ETICHETTA**: un identificatore che individua la locazione di memoria corrente (opzionale)
  - **ISTRUZIONE**: l'istruzione da eseguire
    - OPCODE: è un codice che identifica l'istruzione da eseguire
    - MODO DI INDIRIZZAMENTO: è un codice che esplicita il modo in cui devono essere reperiti i valori con cui si deve operare
  - **COMMENTO**

INDIRIZZO	ETICHETTA	ISTRUZIONE	COMMENTO
100:		addi.b d0,d1,#8	#somma in 68000
	salto:		
100:		add eax, ebx	#somma in Intel x86

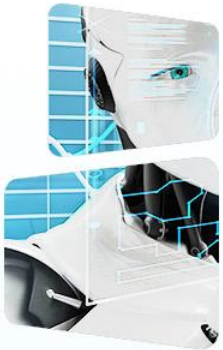


Linguaggi di programmazione

Linguaggio Assemblativo – Struttura di una istruzione: esempio

```
int a=0;
int b=1;
while (a!=10){
    a=a+b;
}
```

INDIRIZZO	ETICHETTA	ISTRUZIONE	COMMENTO
100:		li $t0,0	# inserimento valore 0 in $t0
104:		li $t1,1	# inserimento valore 1 in $t1
108:		li $t6,10	# inserimento valore 10 in $t6
112:	salto:		
116:		add $t0,$t0,$t1	#somma di $t1+$t0 in $t1
120:		bne $t0,$t6,salto	# se il contenuto di $t0 è diverso dal #contenuto di $t6 allora vai a salto



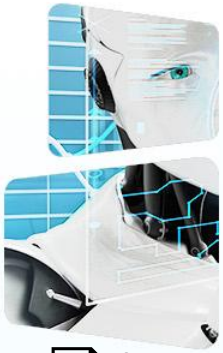
Linguaggi di programmazione

Linguaggio Assemblativo – Sottodivisione di una istruzione

- Una **istruzione** è strutturata in campi:
  - **OPCODE**: è un codice che identifica l'istruzione da eseguire
  - **MODO DI INDIRIZZAMENTO**: è un codice che esplicita come devono essere reperiti i valori con cui si deve operare

OPCODE	Modo di indirizzamento

lb $t0,a #caricamento di un valore nel registro $t0	1000 00dd dddi iiii iiii iiii iiii iiii
sb $t1,b #caricamento di un valore in memoria	1010 00ss sssi iiii iiii iiii iiii iiii
add $t5,$t0,$t1 #somma di due operandi	0000 00dd dddt tttt ssss sXXX X10 0000
sub $t5,$t0,$t1 #sottrazione di due operandi	0000 00dd dddt tttt ssss sXXX X10 0010

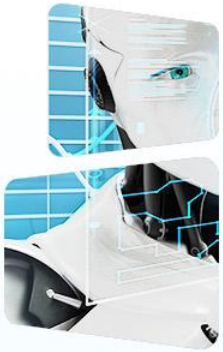


Linguaggi di programmazione

Storia

- In seguito si cercò di astrarre la conoscenza della architettura della macchina mediante i **linguaggi ad alto livello**
  - I primi a raggiungere una certa popolarità furono il Fortran (1957); il BASIC (1964); il Lisp (1959); l'ALGOL (1960); Simula (1967); il Pascal (1970) e il C (1972)

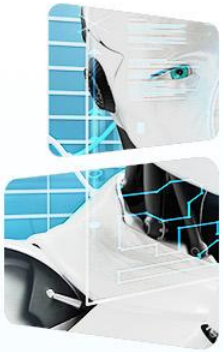
- Nel 1983 vede la luce Smalltalk, il primo linguaggio realmente e completamente **ad oggetti**; poi lo Eiffel (1986), il C++ (1986) e Java (1995)



Linguaggi di programmazione

Linguaggio ad alto livello

- I linguaggi di programmazione ad alto livello sono caratterizzati dalla presenza di **astrazioni** che permettono al programmatore di non specificare certi tipi di dettagli implementativi della macchina



Linguaggi di programmazione

Confronto

```
#include “stdio.h”
void main()
{
    int a, b, c;
                a=4;
                b=6;
                c=a+b;
}
```

```
.text
lb $t3,a
lb $t4,b
add $t5,$t3,$t4
sb $t5,c
.data
a:.byte 4
b:.byte 6
c:.byte 0
```


```
1000 0000 0110 0000 0000  0000 0000 1000
1000 0000 1000 0000 0000  0000 0000 1010
0000 0000 1010 0011 0010  0000 0010 0000
1010 0000 1010 0000 0000  0000 0000 1100
```

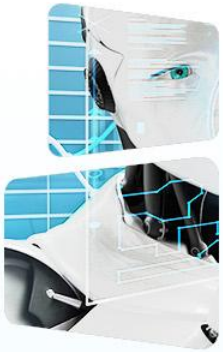
MIPS

Progettazione di un programma

Progettazione di un programma

Introduzione

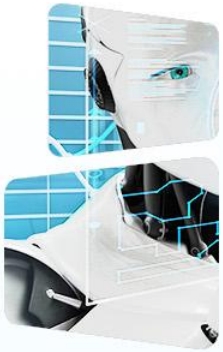
- L'**esecuzione di un programma** è il punto di arrivo di una sequenza di azioni che nella maggior parte dei casi iniziano con la scrittura di un programma in un linguaggio simbolico di alto livello
- Le azioni principali che compongono tale sequenza nel caso in cui si parte da un linguaggio ad alto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



Progettazione di un programma

Linguaggio di modellazione

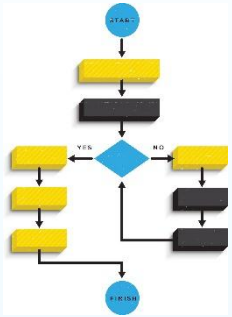
- Un **linguaggio di modellazione** (*modelling language*) è un linguaggio formale che può essere utilizzato per descrivere (e modellare) un sistema di qualche natura o formalizzare la risoluzione di un problema
- Esistono diversi linguaggi di modellazione:
  - Grafici o testuali
  - Interpretabili o non interpretabili
  - Oggetto della modellazione

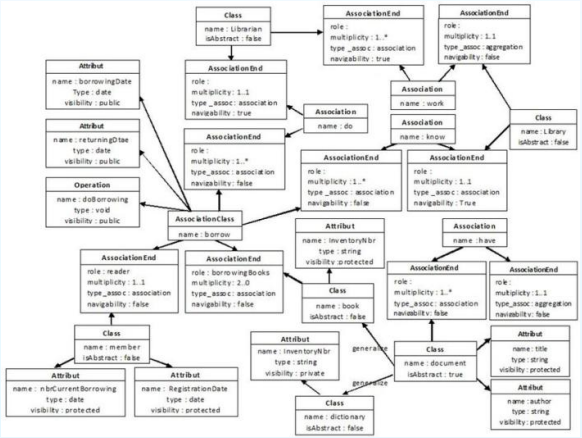


Progettazione di un programma

Linguaggio di modellazione - esempi

- Esempi:
  - I diagrammi di flusso
  - Unified Modeling Language (UML)
  - Systems Modeling Language (SysML)
  - EXPRESS-G
  - Interaction Flow Modeling Language (IFML)

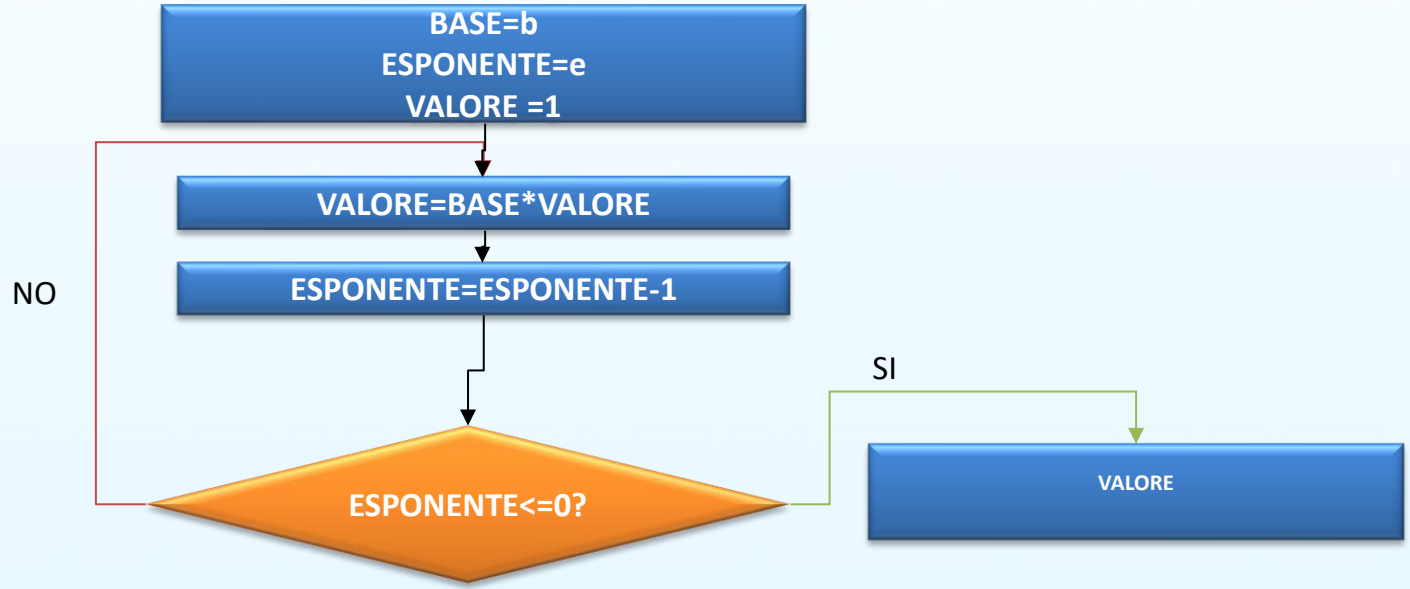


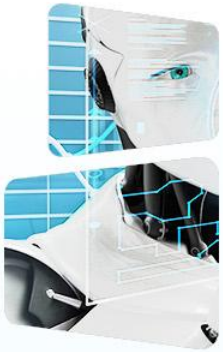


Progettazione di un programma

Linguaggio di modellazione – *diagramma di flusso*

Realizzare un programma che calcola l'elevamento a potenza b^e con b>0 e e>0



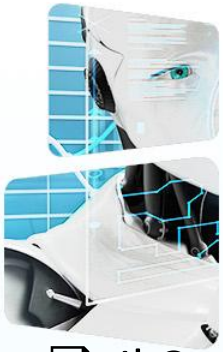


Progettazione di un programma

Scrittura in un linguaggio ad alto livello

- Programma scritto sfruttando un **linguaggio ad alto livello** (o in assembly)

```
void main()
{
    int e, b, v, t;
    b=2;
    e=10;
    v=1;
repeat
    v=v*b
    e=e-1;
   until (e>0)
}
```



Progettazione di un programma

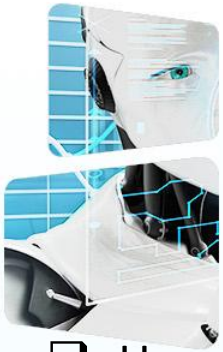
Scrittura in un linguaggio ad alto livello

- Il **Compilatore** traduce un codice scritto in un linguaggio ad alto livello in un:
  - programma equivalente scritto in linguaggio assemblativo
  - oppure, direttamente in un file oggetto
- Il compilatore svolge una analisi sintattica del programma, un riposizionamento delle istruzioni e una ottimizzazione del codice

```
void main()
{
   int e, b, v, t;
   b=2;
   e=10;
   v=1;
repeat
   v=v*b
   e=e-1;
  until (e>0)
}
```

```
.text
.globl main
main:
                lw  $t1,b
                lw  $t2,e
                lw  $t3,v
salto:
                mul $t3,$t3,$t1
                sub $t2,$t2,1
                bgtz $t2,salto
                li $v0,10
                syscall
.data
e:   .word 10
b:   .word 2
v: .word 1
```

t non è riportato perché
il compilatore ottimizza il codice



Progettazione di un programma

Modulo oggetto

- Un **modulo oggetto**, ottenuto dall'**assemblatore** (con un **procedimento a doppia passata**), è un codice che può contenere:
  - Istruzioni
  - Dati
  - Riferimenti a indirizzi interni (salto a funzioni,…)

- Un modulo oggetto, se richiede collegamenti a librerie esterne o fa uso di variabili globali tra moduli oggetti non è direttamente eseguibile

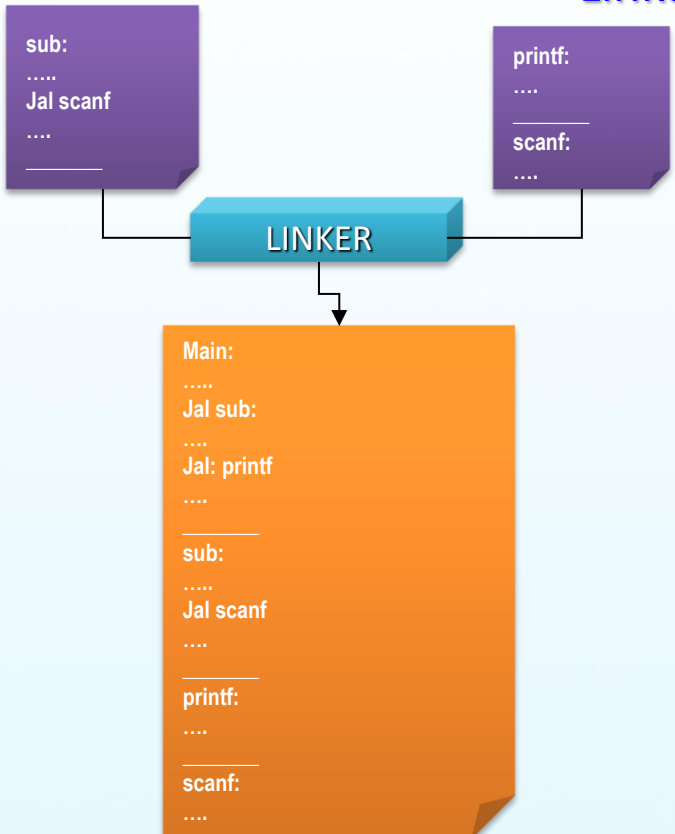
```
.text
.globl main
main:
                lw  $t1,b
                lw  $t2,e
                lw  $t3,v
salto:
                mul $t3,$t3,$t1
                sub $t2,$t2,1
                bgtz $t2,salto
                li $v0,10
                syscall
.data
e:   .word 10
b:   .word 2
v: .word 1
```

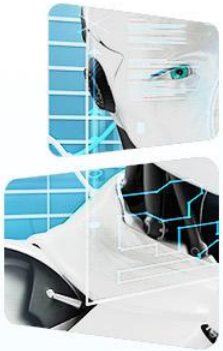
```
…
00010101010
01010111110
(riferimento ad altro modulo)
01111111110
11111000010
11111111011
…
```

Progettazione di un programma

Linker

- Il **Linker** combina un insieme di moduli e librerie in un *programma eseguibile*
- Il linker ha tre compiti:
  - Ricercare nei file libreria le routine di libreria utilizzate dal programma (*es. printf: routine per la stampa a video di dati*)
  - Determinare le locazioni di memoria che il codice di ogni modulo andrà ad utilizzare e aggiornare i riferimenti assoluti in modo opportuno
  - **Risolvere i riferimenti tra i diversi moduli** e le **variabili globali**





Progettazione di un programma

Linker

- Il **Linker** determina le locazioni di memoria che il codice di ogni modulo andrà ad utilizzare, aggiorna i riferimenti assoluti in modo opportuno e fa riferimento a variabili globali che coinvolgono più moduli

RIFERIMENTI RELATIVI

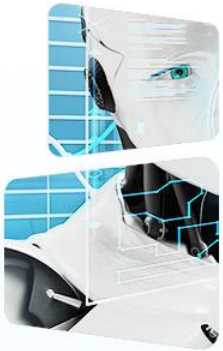
```
Main:
.....
Jal sub:
....
Jal: printf
....
_____
sub:
.....
Jal scanf
....
_____
printf:
....
_____
scanf:
....
```

```
Main:
.....
Jal (a0)
....
Jal: (a1)
....
_____
sub:
.....
Jal (a2)
....
_____
printf:
....
_____
scanf:
....
```

RIFERIMENTI ASSOLUTI

```
0
.....
100
....
124
....
_____
132
.....
140
....
_____
164
....
_____
200
....
```

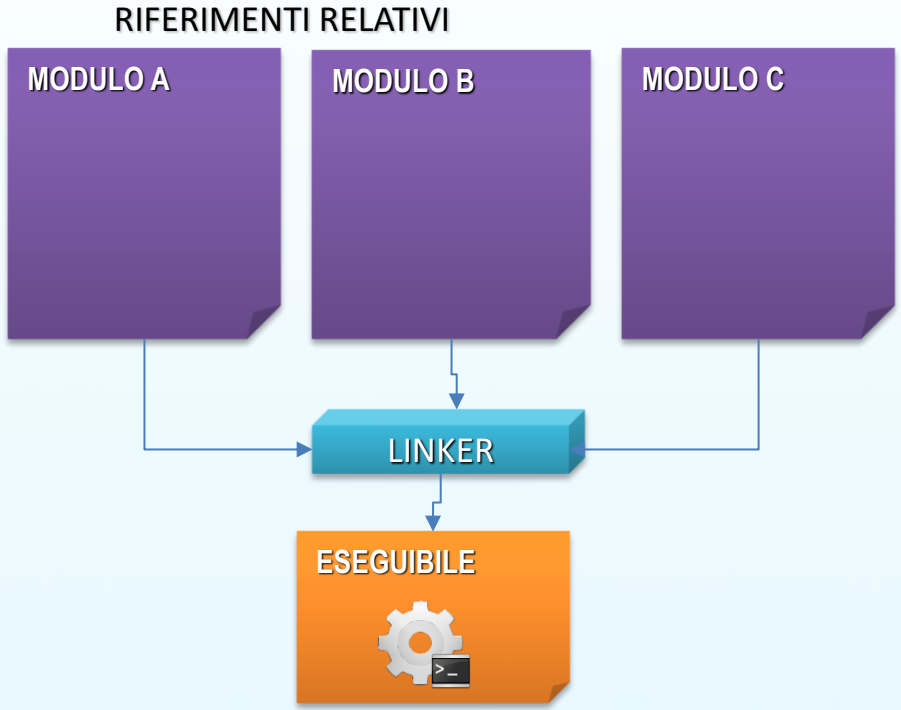
```
Main:
.....
Jal 132
....
Jal: 164
....
_____
sub:
.....
Jal 200
....
_____
printf:
....
_____
scanf:
....
```

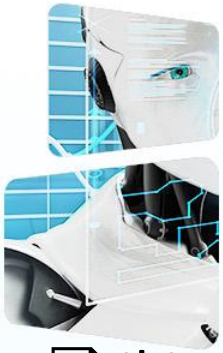


Progettazione di un programma

Linker- produzione dell'seguibile

- Il **Linker** combina un insieme di moduli e file libreria in un *programma eseguibile*
- Il programma eseguibile non deve contenere riferimenti non risolti (*unresolved references*)
- Solamente il programma eseguibile può essere elaborato su una macchina

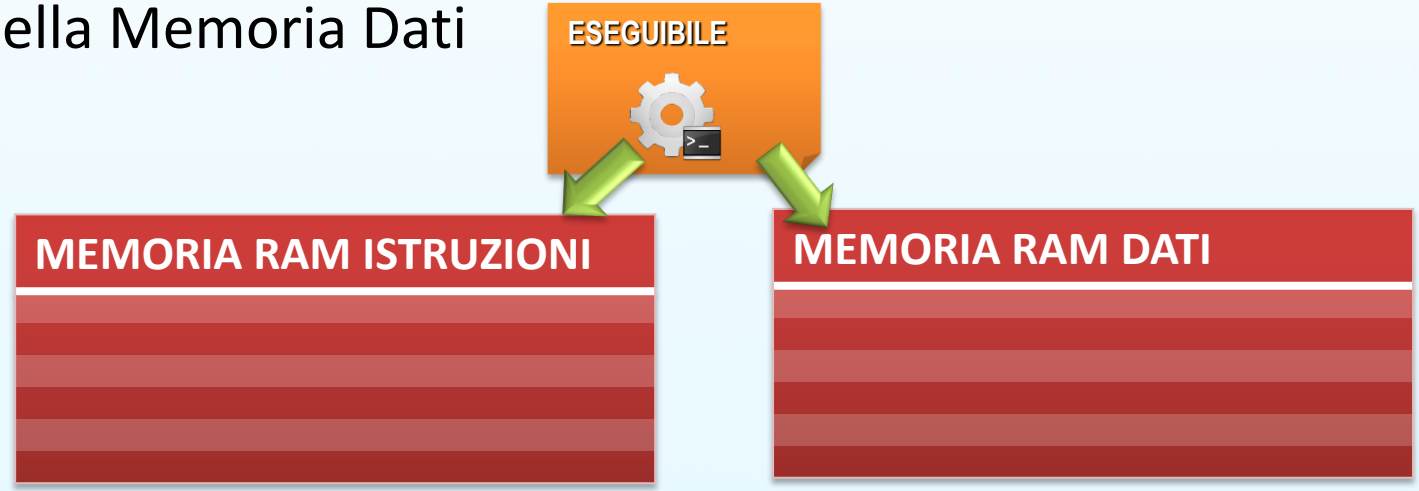


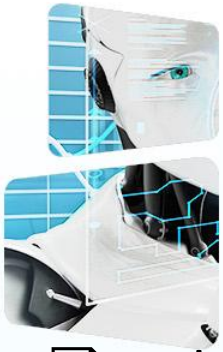


Progettazione di un programma

Loader

- Il Loader (o caricatore) sposta il file eseguibile in Memoria Centrale. Nel MIPS la parte relativa al programma è sita nella Memoria Istruzione viceversa quella che archivia gli operandi nella Memoria Dati

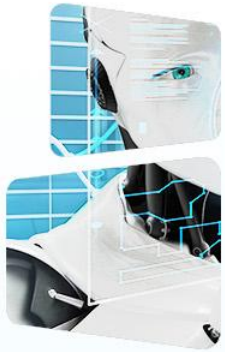




Progettazione di un programma

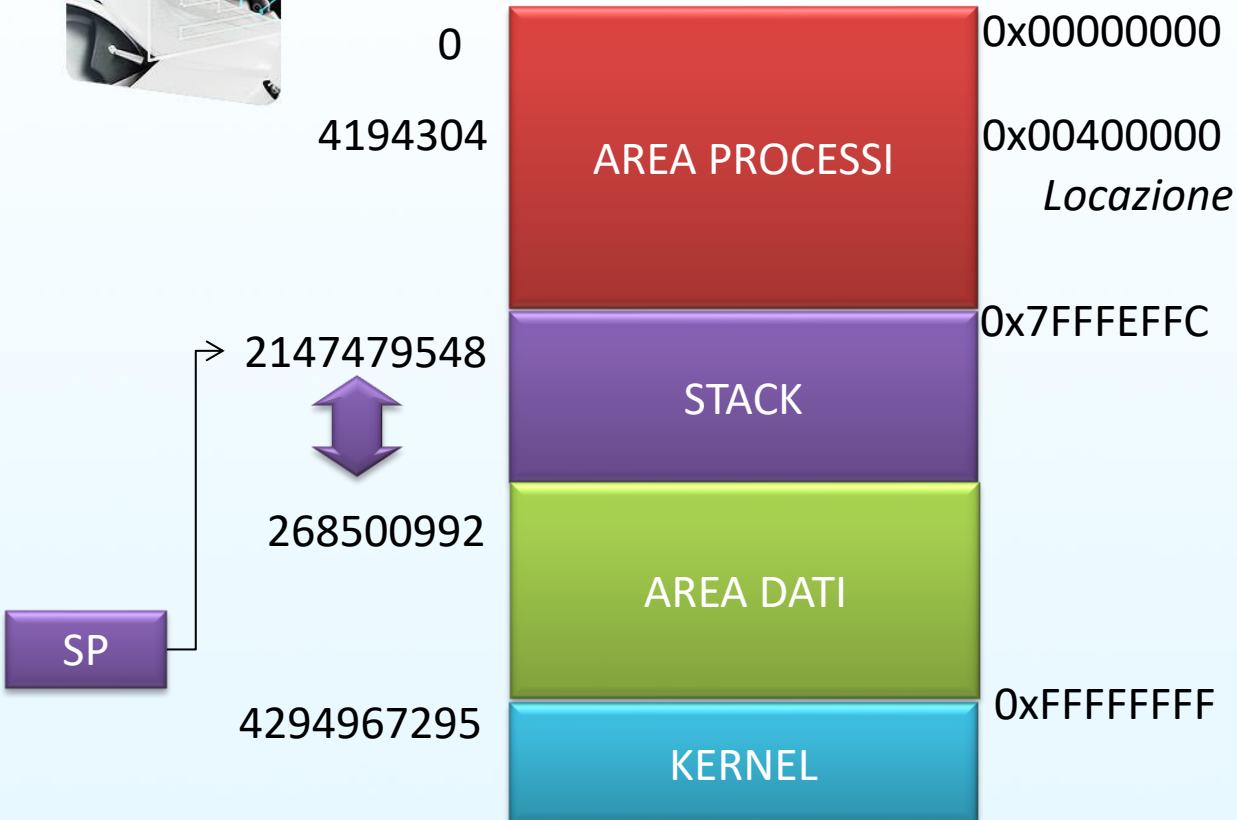
Memoria MARS

- Nel simulatore MARS la memoria è suddivisa in segmenti
- Ogni segmento è utilizzato per un particolare scopo
- **Segmenti principali:**
  - ***Text***: Contiene il codice dei programma
  - ***Data***: Contiene i dati “globali” dei programmi
  - ***Stack***: Contiene i dati “locali” delle funzioni



Progettazione di un programma

Memoria MARS



Fine