

## Parte 3

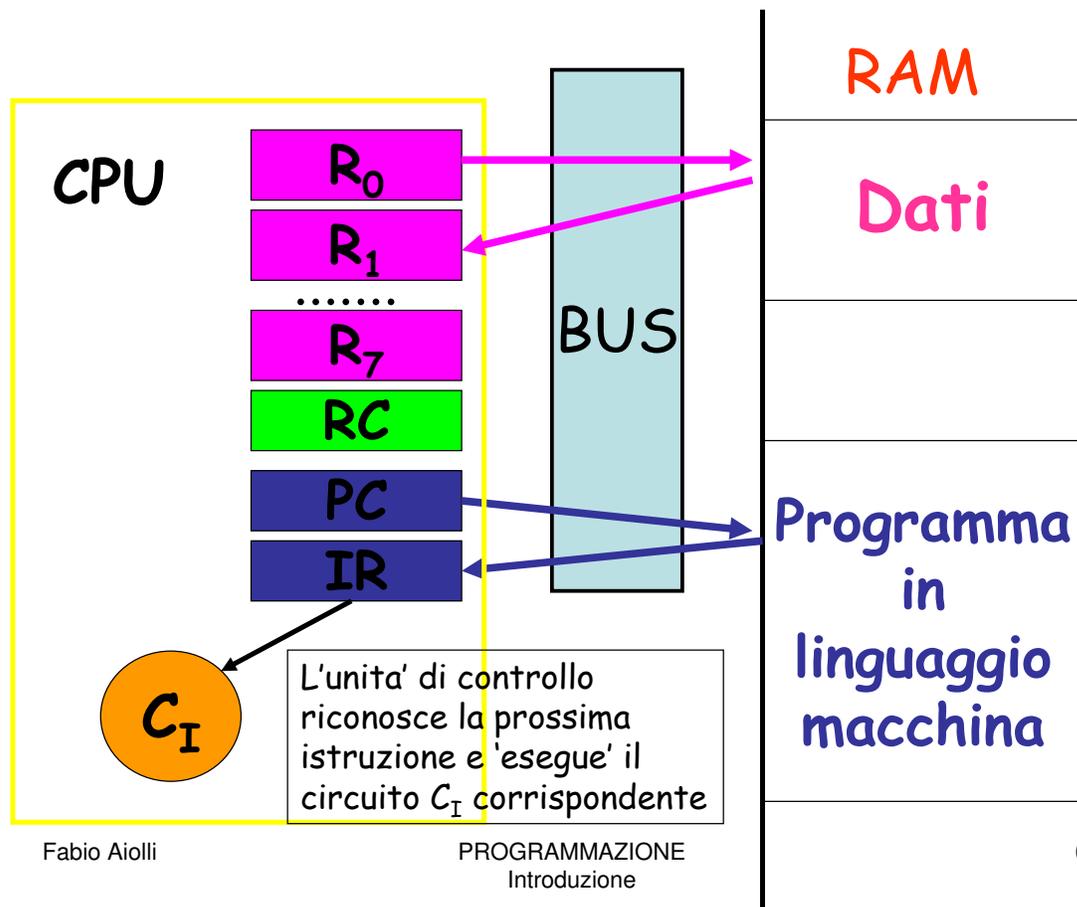
### Linguaggio Macchina e Assembler

#### LINGUAGGIO MACCHINA

Descriveremo una CPU "MINIMA" dotata di un certo insieme di istruzioni  $I$  ciascuna realizzata da un corrispondente circuito  $C_I$ .

Questo insieme di istruzioni della CPU "MINIMA" costituisce il linguaggio macchina di "MINIMA".

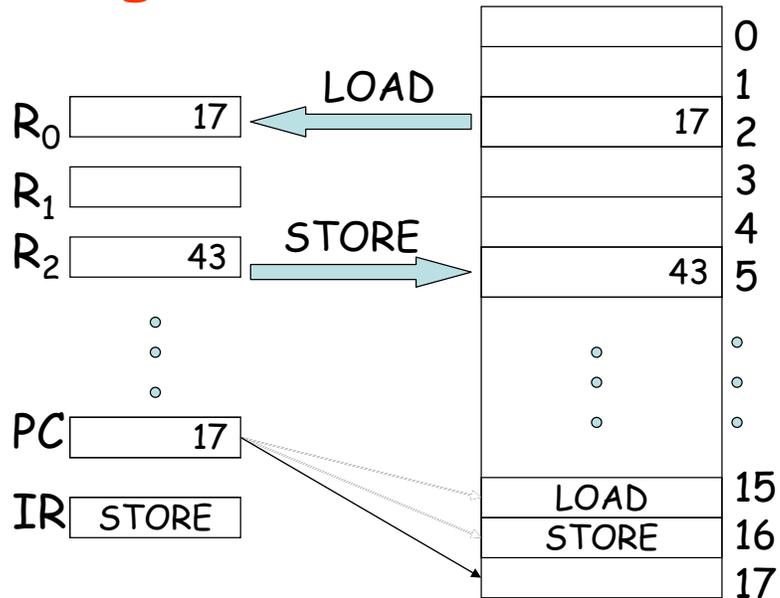
Semplificheremo l'approccio considerando che ogni istruzione sia memorizzata in una parola di memoria di 32 bit.



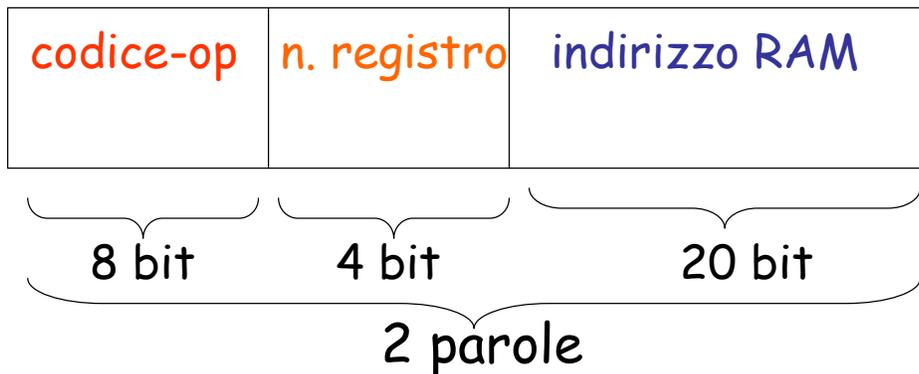
## 3 tipi di istruzioni macchina

- 1) **trasferimento** tra RAM e registri di calcolo della CPU
- 2) **operazioni aritmetiche**: somma, differenza, moltiplicazione e divisione
- 3) **operazioni di controllo**: confronto, salto e stop

# Istruzioni di trasferimento: registri $\Leftrightarrow$ RAM



## Formato delle istruzioni di trasferimento



**Codici:**

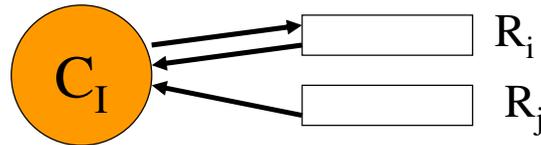
LOAD	00000000
STORE	00000001

**Esempio:**

00000000	0000	0000000000000000000010
00000001	0010	00000000000000000000101

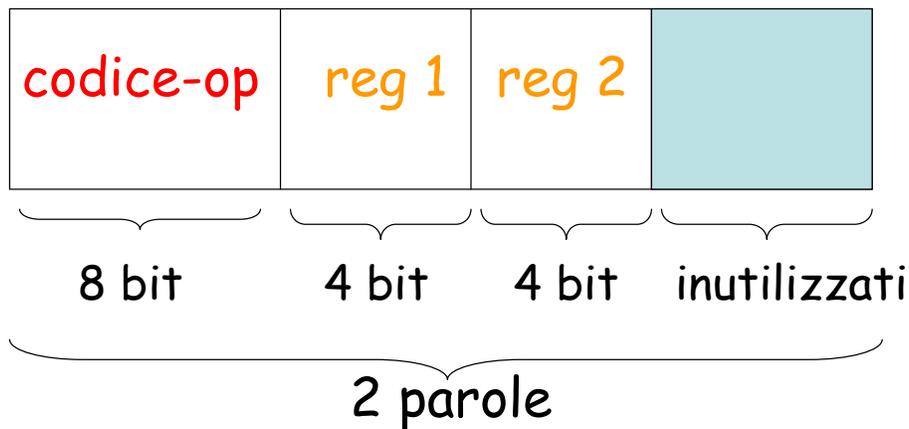
# Istruzioni aritmetiche

Eseguono somma, differenza, moltiplicazione e divisione usando i registri come operandi. Memorizzano il risultato nel primo registro argomento.



<b>ADD</b>	00000010	<b>FADD</b>	00000011
<b>SUB</b>	00000100	<b>FSUB</b>	00000101
<b>MULT</b>	00000110	<b>FMULT</b>	00000111
<b>DIV</b>	00001000	<b>FDIV</b>	00001001
<b>MOD</b>	00001010		

## Formato delle istruzioni aritmetiche

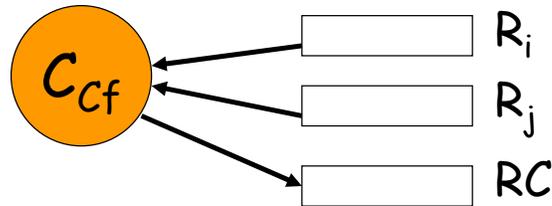


Esempio: 00000010 0011 0001 xxxxxxxxxxxxxxxxxxxxxx  
00000011 0011 0001 xxxxxxxxxxxxxxxxxxxxxx

# Istruzione di confronto

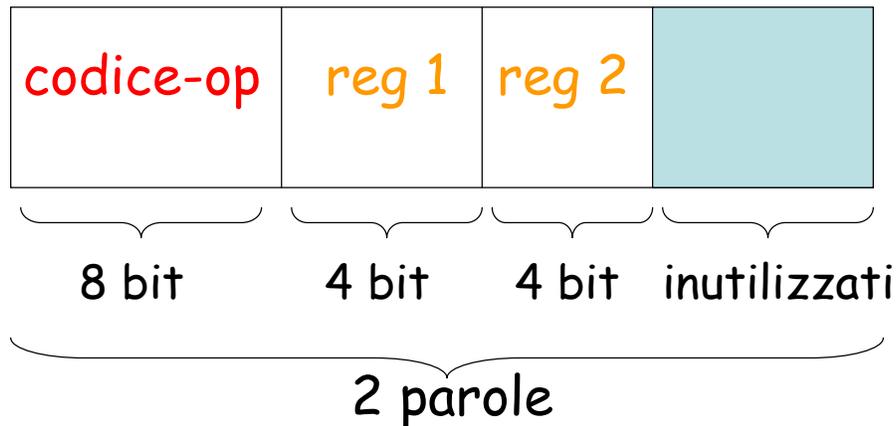
Paragona il contenuto di 2 registri  $R_i$  e  $R_j$

- se  $R_i < R_j$  memorizza -1 nel registro RC
- se  $R_i = R_j$  memorizza 0 in RC
- se  $R_i > R_j$  memorizza 1 in RC



Codici:            COMP    00100000  
                  FCOMP    00100001

## Formato dell'istruzione di confronto



Esempio: 00100000 0010 0101 xxxxxxxxxxxxxxxxxxxx  
          00100001 0010 0101 xxxxxxxxxxxxxxxxxxxx

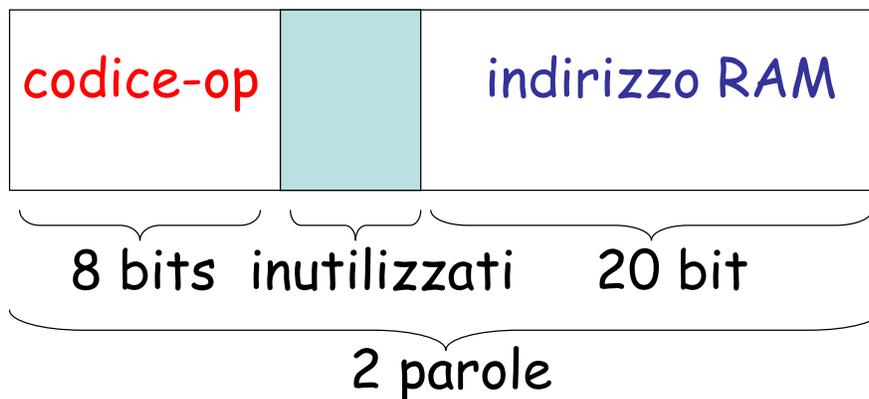
# Istruzione di salto

Permette di "saltare" ad un'altra istruzione del programma a seconda del contenuto del registro RC (cioè a seconda del risultato di un confronto)

<b>BRLT</b> (RC=-1)	01000001	<b>BRNE</b> (RC≠0)	01000100
<b>BRLE</b> (RC<1)	01000010	<b>BRGE</b> (RC>-1)	01000110
<b>BREQ</b> (RC=0)	01000011	<b>BRGT</b> (RC=1)	01000101
	<b>BRANCH</b>		10000000

↑  
**Salto incondizionato!**

## Formato dell'istruzione di salto



Esempio: **01000001** **xxxx** 000000000000000000001001  
**10000000** **xxxx** 000000000000000000001010

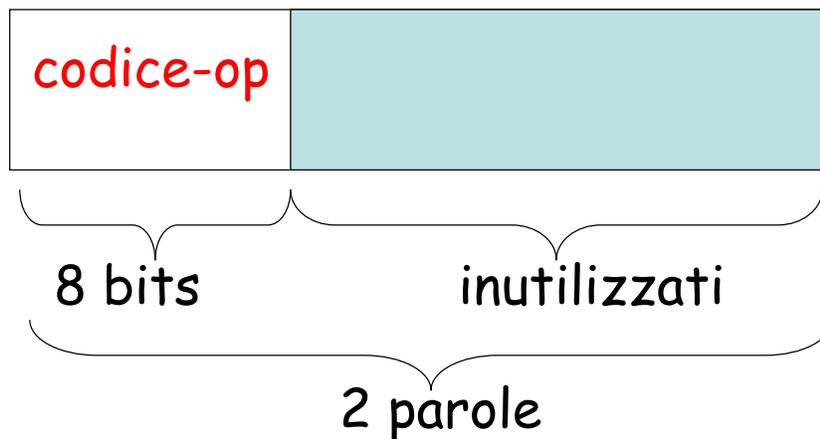
L'effetto di una istruzione di salto ad un indirizzo  $M$  è quindi quello di memorizzare  $M$  nel registro PC se si verifica la relativa condizione nel registro RC

# Istruzione di stop

Semplicemente termina il programma

Codice:    STOP    10000001

## Formato istruzione di stop

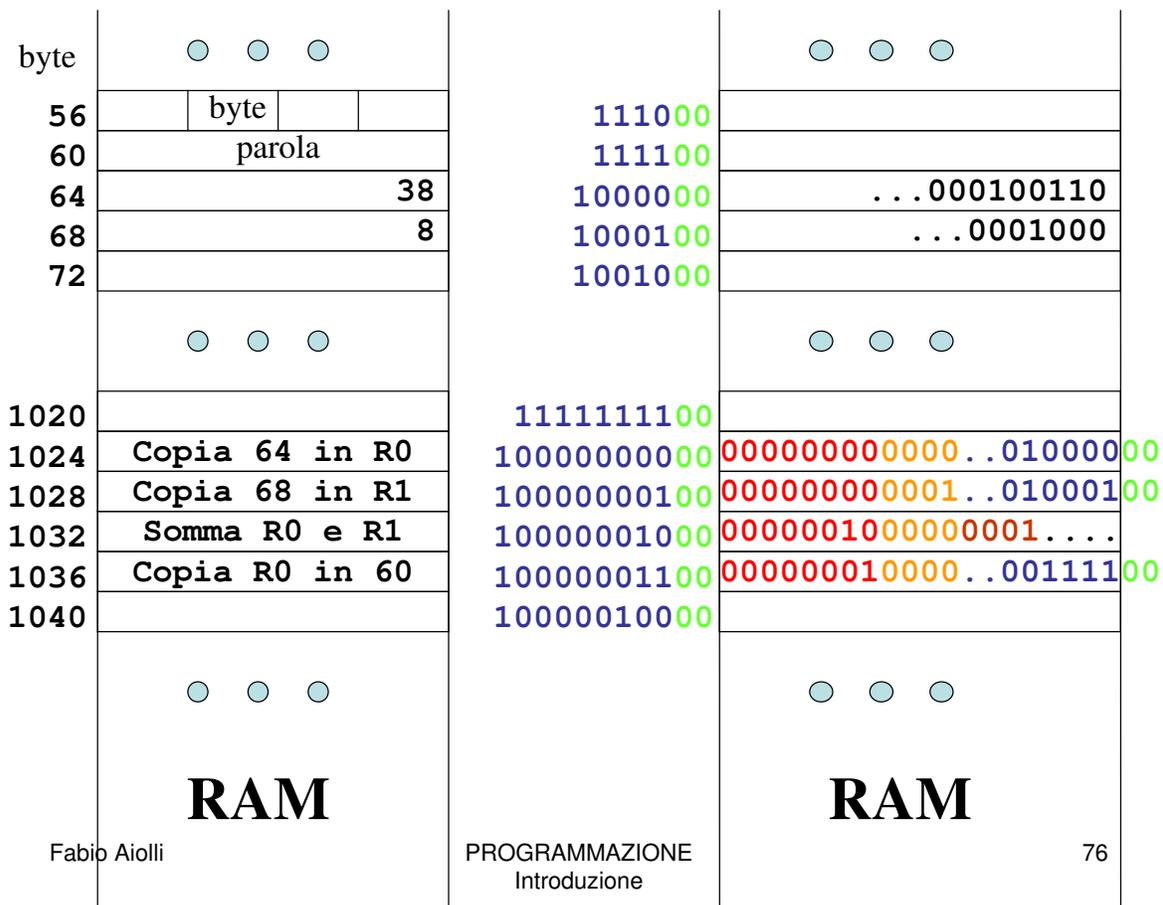


Esempio:    10000001    xx

# Esempio

Scriviamo un programma in linguaggio macchina che:

- trasferisce il contenuto delle 2 parole della RAM di indirizzi 64 e 68 nei registri  $R_0$  e  $R_1$
- somma i contenuti dei registri  $R_0$  ed  $R_1$
- trasferisce il risultato nella parola della RAM all'indirizzo 60



# Svantaggi del linguaggio macchina

- programmi in binario sono difficili da scrivere, capire e modificare
- il programmatore deve occuparsi della gestione degli indirizzi RAM

primo passo  $\Rightarrow$  Assembly

## Caratteristiche dell'Assembly

L'Assembly e' un rudimentale linguaggio di programmazione

- codici mnemonici per le operazioni
- nomi mnemonici, detti *variabili*, al posto degli indirizzi RAM dei dati
- nomi mnemonici, detti *etichette*, al posto degli indirizzi RAM delle istruzioni usati nei salti
- categorie di valori: si tratta dei cosiddetti *tipi di dato* INT e FLOAT

## Codici mnemonici di operazioni

- **trasferimento**: **LOAD** (RAM → CPU) e **STORE** (CPU → RAM)
- **aritmetiche**: **ADD, SUB, MULT, DIV, MOD, FADD, FSUB, FMULT, FDIV**
- **test**: **COMP, FCOMP**
- **salto**: **BREQ, BRGT, BRLT, BRGE, BRLE, BRANCH**
- **terminazione**: **STOP**

## Esempio precedente in Assembly

```
Z : INT ;  
X : INT 38 ;  
Y : INT 8 ;  
LOAD R0 X ;  
LOAD R1 Y ;  
ADD R0 R1 ;  
STORE R0 Z ;
```

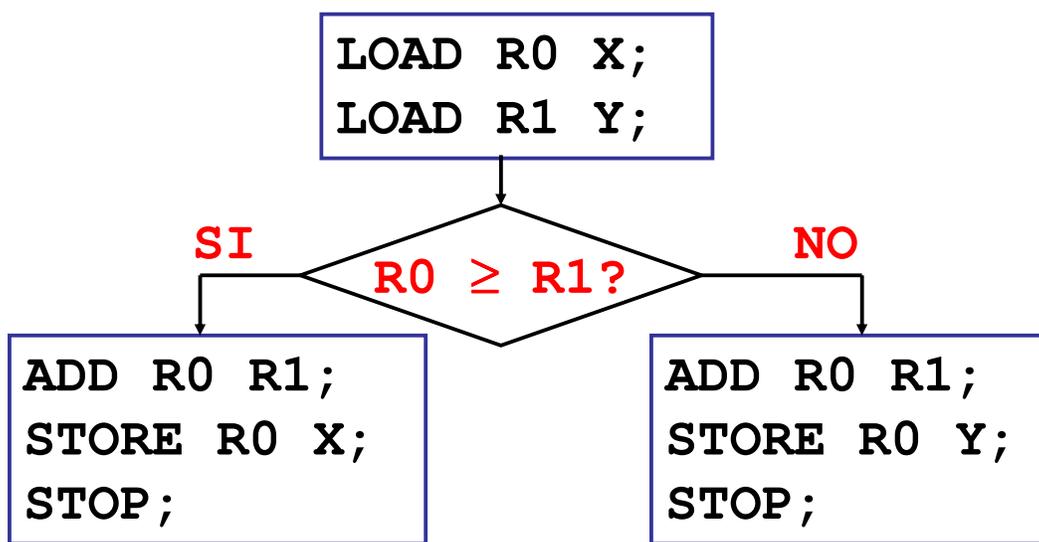
dichiarazioni di  
variabili

istruzioni

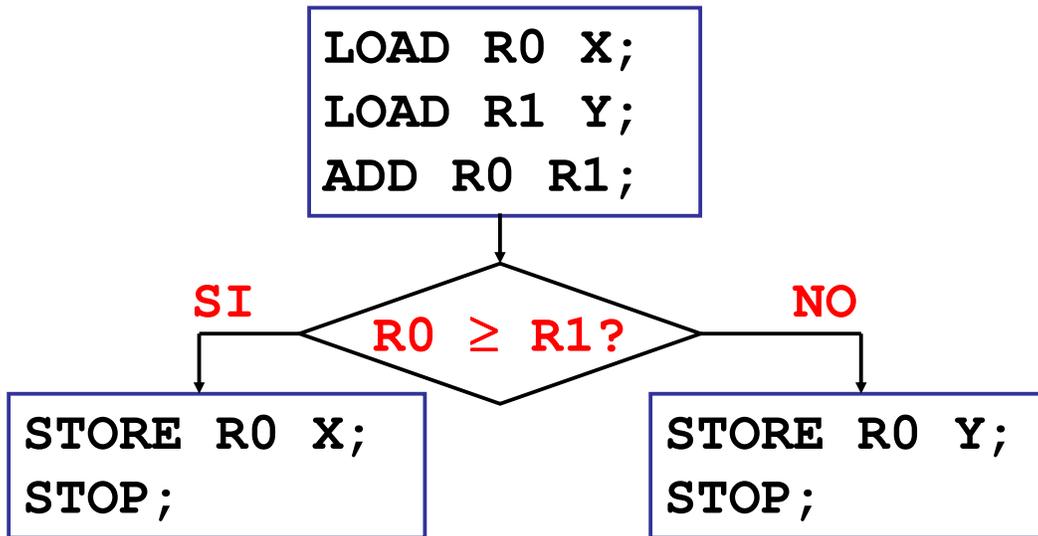
## Altro esempio

Carica due valori dalla RAM, li somma e memorizza il risultato al posto del maggiore dei 2 numeri sommati (nel caso siano uguali, in uno qualsiasi dei 2 indirizzi RAM)

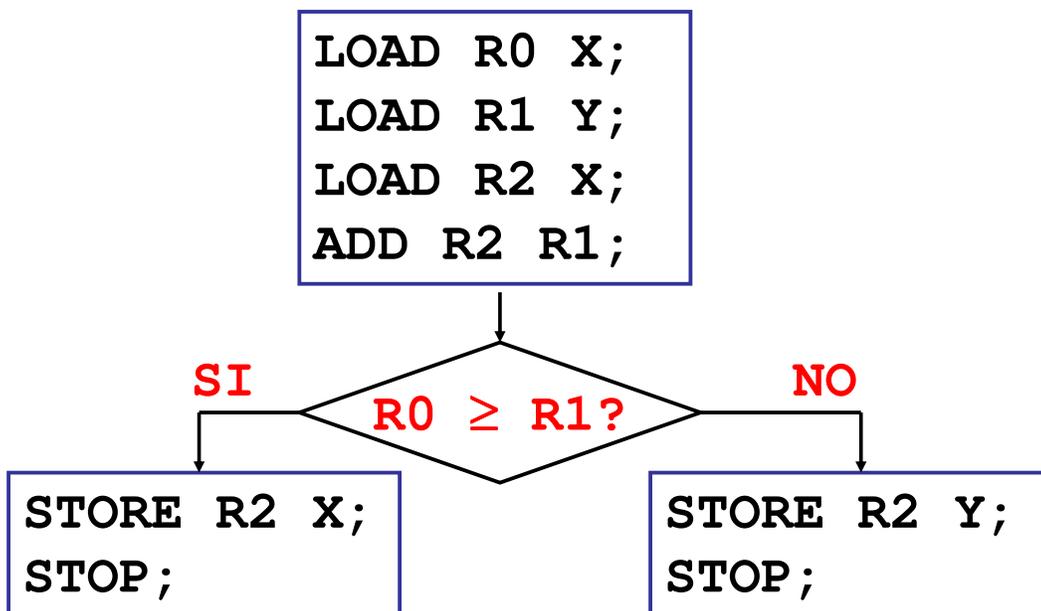
## Algoritmo 1



# Questo algoritmo funziona?



## Algoritmo 2



## Codice assembly algoritmo 1

```
X: INT 38;
Y: INT 8;
    LOAD R0 X;
    LOAD R1 Y;
    COMP R0 R1;
    BRGE maggiore;
    ADD R0 R1;
    STORE R0 Y;
    STOP;
maggiore:
    ADD R0 R1;
    STORE R0 X;
    STOP;
```

## Codice assembly algoritmo 2

```
X: INT 38;
Y: INT 8;
    LOAD R0 X;
    LOAD R1 Y;
    LOAD R2 X;
    ADD R2 R1;
    COMP R0 R1;
    BRGE maggiore;
    STORE R2 Y;
    STOP;
maggiore:
    STORE R2 X;
    STOP;
```

## Conclusioni

3 costrutti fondamentali:

- 1)  $X \leftarrow \text{valore}$  **ASSEGNAMENTO**
- 2) **TEST** di una condizione che determina quale azione intraprendere successivamente
- 3) **CICLO** che si ripete finchè una data condizione è vera

**SONO COSTRUTTI FONDAMENTALI in ogni LINGUAGGIO di PROGRAMMAZIONE**

## Esercizi su Assembler

### ESERCIZIO I

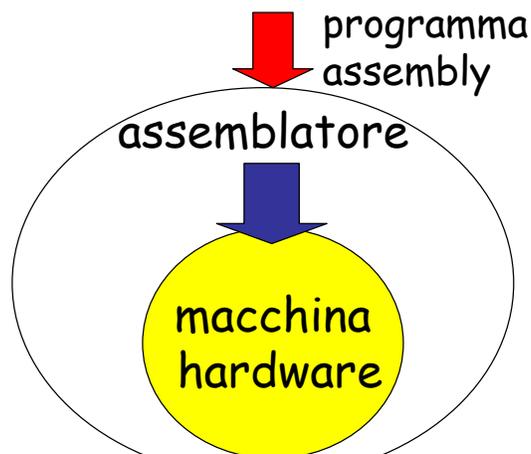
- Legge un dato da una variabile N
- Calcola la somma dei numeri naturali fino a N
- Il risultato viene scritto nella variabile X

### ESERCIZIO II

- Legge due dati A e N
- Se  $N \leq 0$  termina il programma
- Calcola  $A^N$
- Il risultato viene scritto nella variabile X

# La CPU non "capisce" l'assembly !!

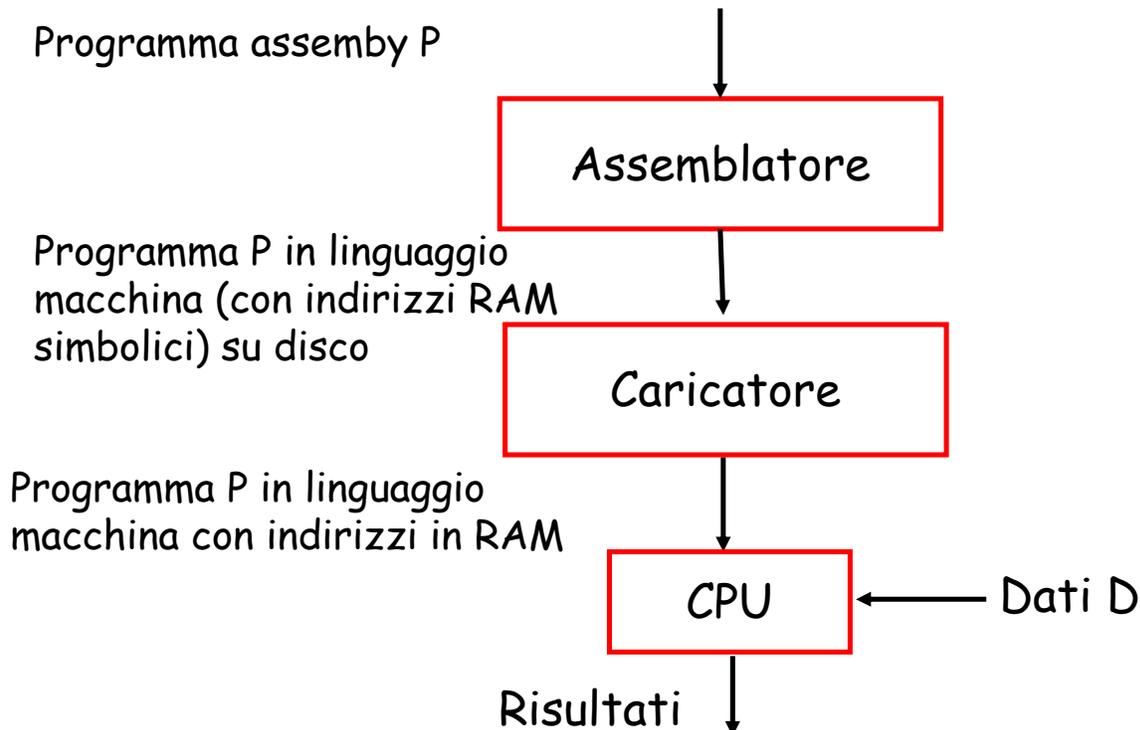
Il programma assembly deve essere tradotto in un programma macchina



Fabio Aioli

PROGRAMMAZIONE  
Introduzione

89



Fabio Aioli

PROGRAMMAZIONE  
Introduzione

90