Parte 3

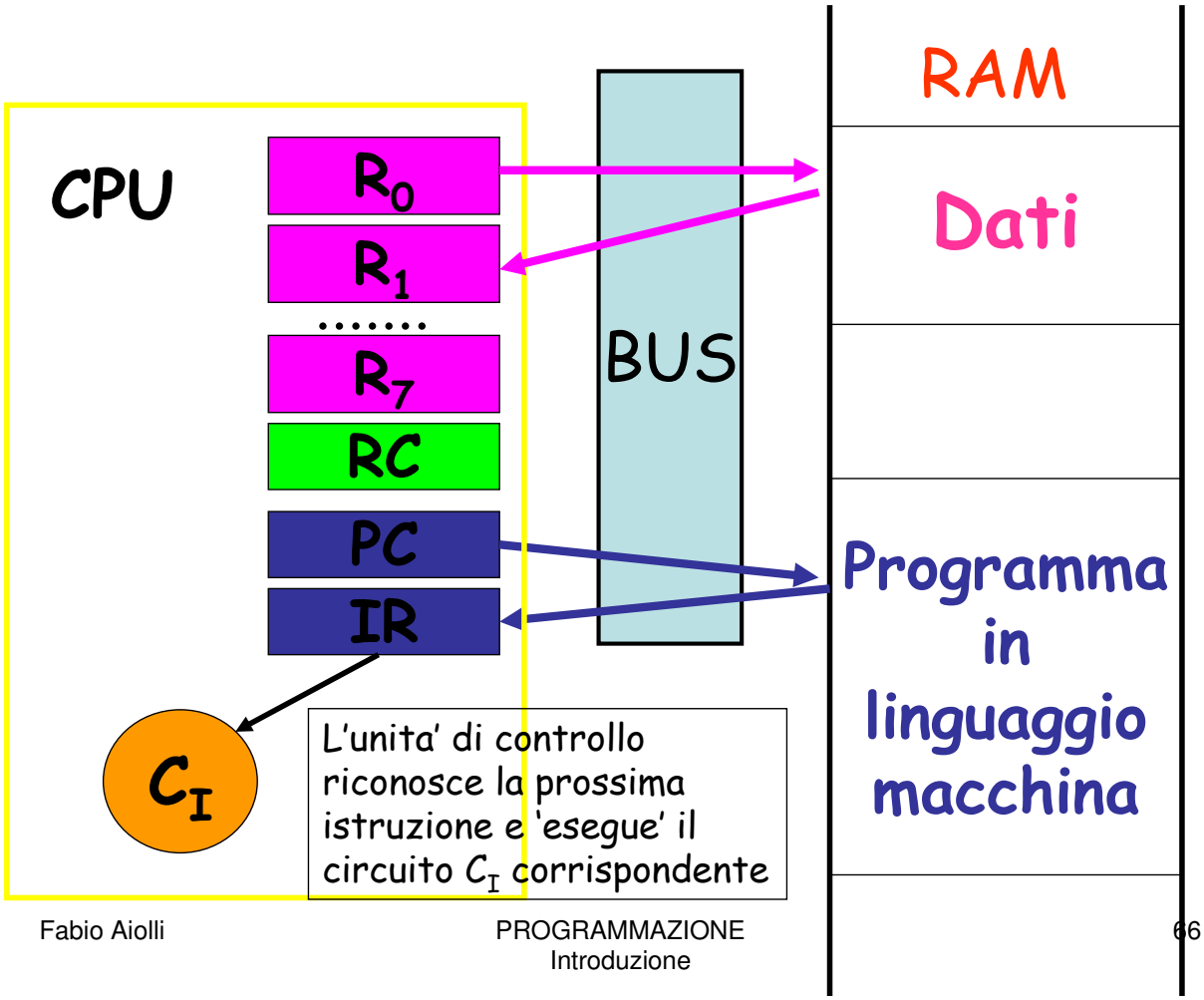
Linguaggio Macchina e Assembler

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".

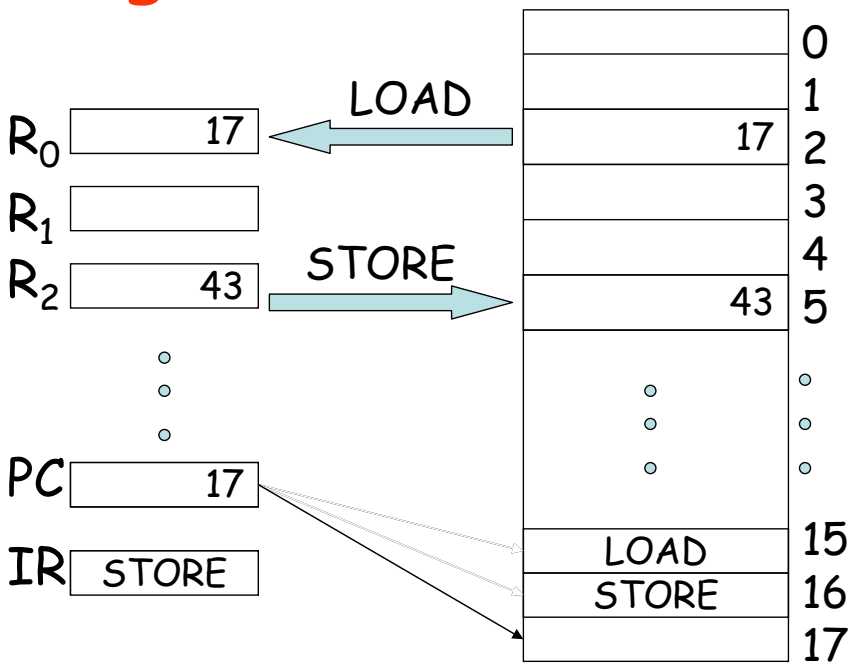
Semplificheremo l'approccio considerando che ogni istruzione sia memorizzata in una parola di memoria di 32 bit.



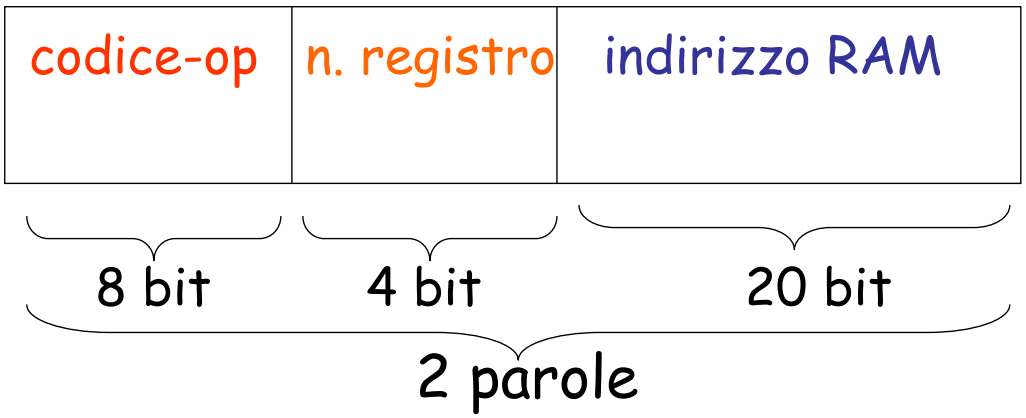
3 tipi di istruzioni macchina

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 ⇔ RAM



Formato delle istruzioni di trasferimento



Codici:

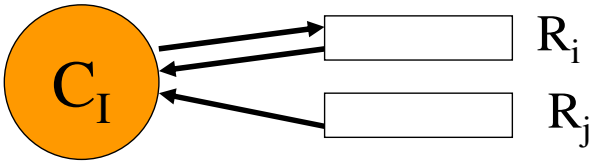
LOAD	**00000000**
STORE	**00000001**

Esempio:

```
00000000 0000 00000000000000000010
00000001 0010 00000000000000000101
```

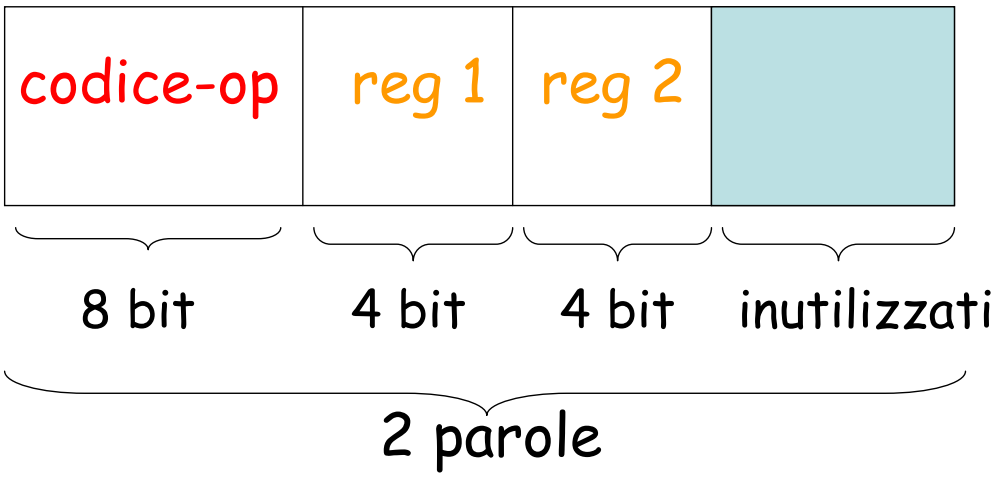
Istruzioni aritmetiche

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



ADD	**00000010**	**FADD**	**00000011**
SUB	**00000100**	**FSUB**	**00000101**
MULT	**00000110**	**FMULT**	**00000111**
DIV	**00001000**	**FDIV**	**00001001**
MOD	**00001010**		

Formato delle istruzioni aritmetiche

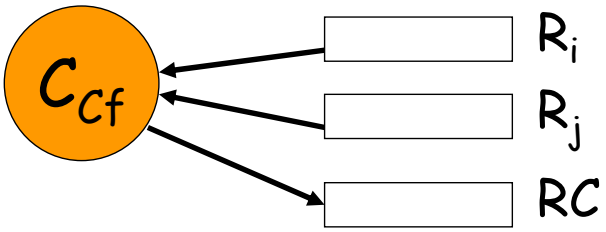


Esempio: **00000010 0011 0001** xxxxxxxxxxxxxxxx
00000011 0011 0001 xxxxxxxxxxxxxxxx

Istruzione di confronto

Paragona il contenuto di 2 registri R_i e R_j

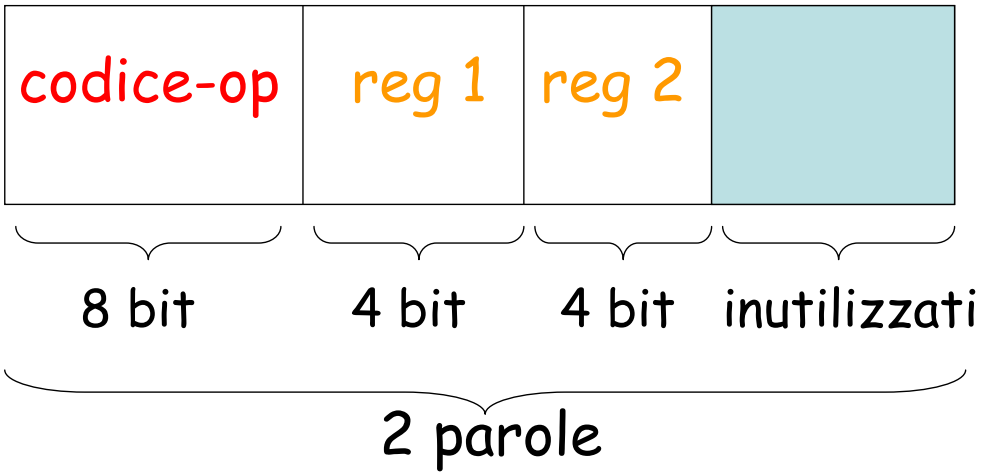
- se $R_i < R_j$ memorizza -1 nel registro RC
- se $R_i = R_j$ memorizza 0 in RC
- se $R_i > R_j$ memorizza 1 in RC



Codici:

COMP	00100000
FCOMP	00100001

Formato dell'istruzione di confronto



Esempio:

```
00100000 0010 0101 xxxxxxxxxxxxxxxx
00100001 0010 0101 xxxxxxxxxxxxxxxx
```

Istruzione di salto

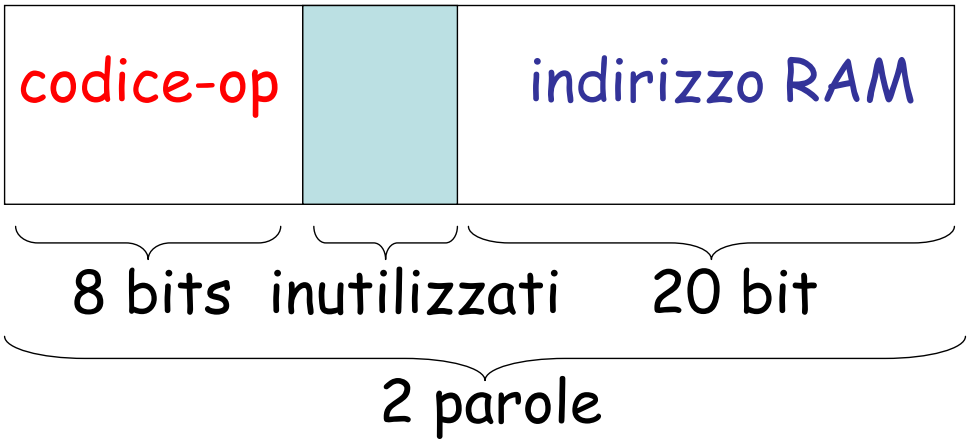
Permette di "saltare" ad un'altra istruzione del programma a seconda del contenuto del registro RC (cioè a seconda del risultato di un confronto)

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

BRANCH 10000000

Salto incondizionato!

Formato dell'istruzione di salto



Esempio:

```
01000001 xxxx 00000000000000001001
10000000 xxxx 00000000000000001010
```

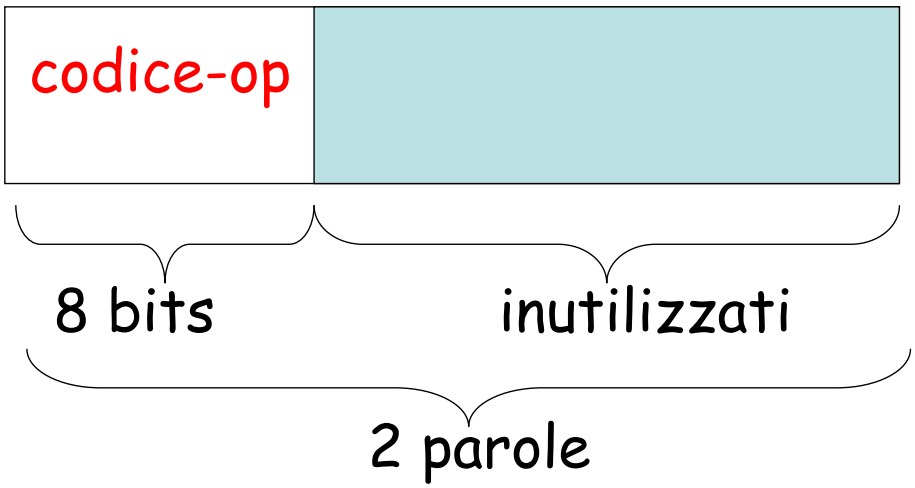
L'effetto di una istruzione di salto ad un indirizzo M e` quindi quello di memorizzare M nel registro PC se si verifica la relativa condizione nel registro RC

Istruzione di stop

Semplicemente termina il programma

Codice: `STOP 10000001`

Formato istruzione di stop

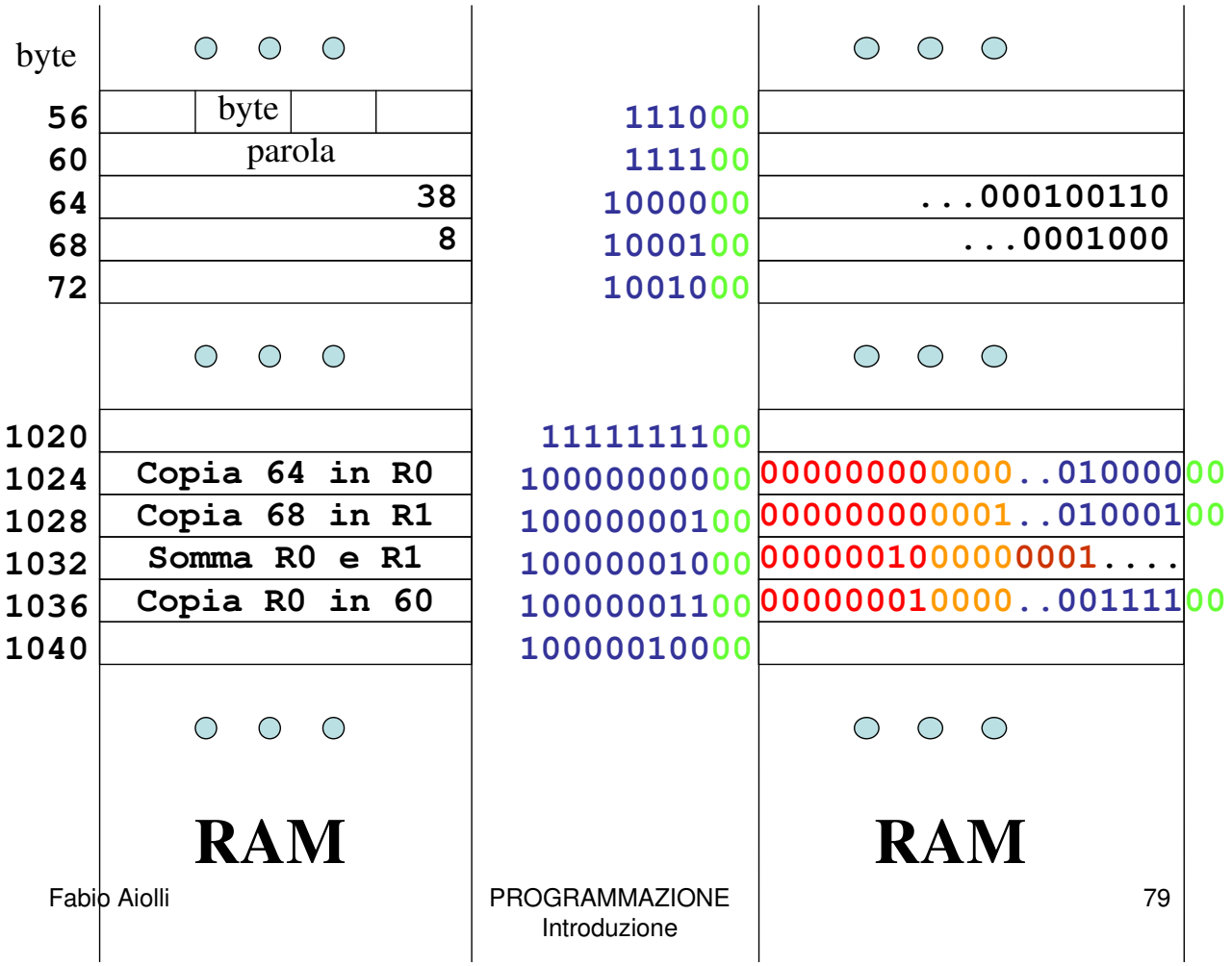


Esempio: `10000001 xxxxxxxxxxxxxxxxxxxxxxxx`

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

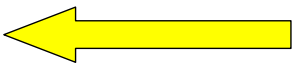
- codici mnemonici per le operazioni
- nomi mnemonici, detti *variabili*, al posto degli indirizzi RAM dei dati
- nomi mnemonici, detti *etichette*, al posto degli indirizzi RAM delle istruzioni usati nei salti
- categorie di valori: si tratta dei cosiddetti *tipi di dato* INT e FLOAT

Codici mnemonici di operazioni

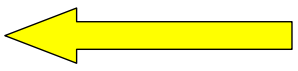
- trasferimento: **LOAD** (RAM → CPU) e **STORE** (CPU → RAM)
- aritmetiche: **ADD,SUB,MULT,DIV,MOD, FADD,FSUB,FMULT,FDIV**
- test: **COMP,FCOMP**
- salto: **BREQ,BRGT,BRLT,BRGE,BRLE, BRANCH**
- terminazione: **STOP**

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

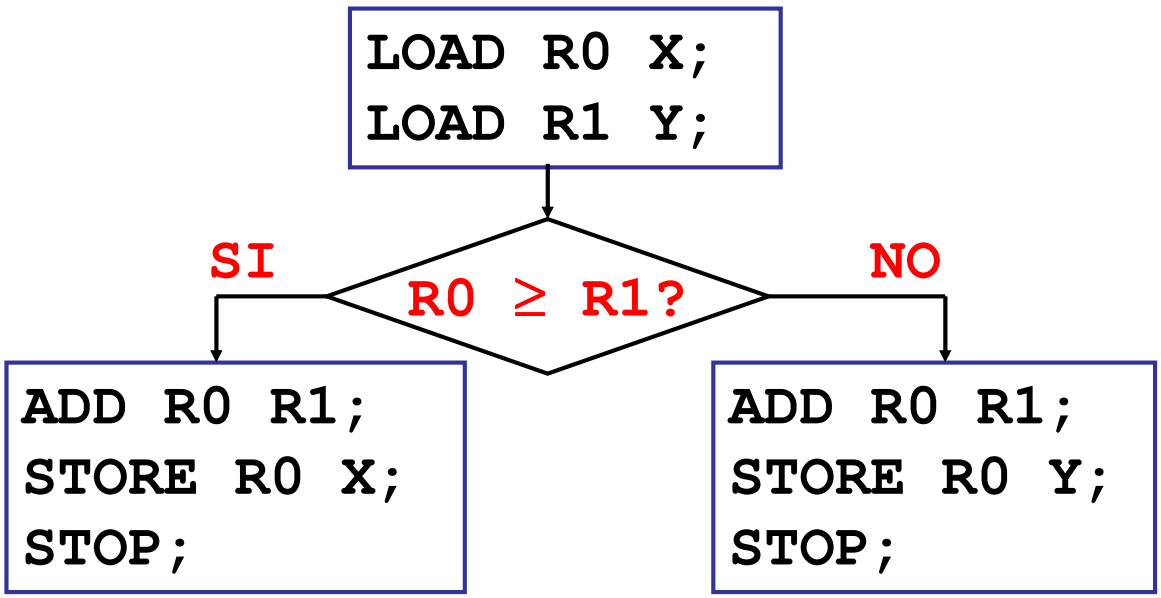


istruzioni

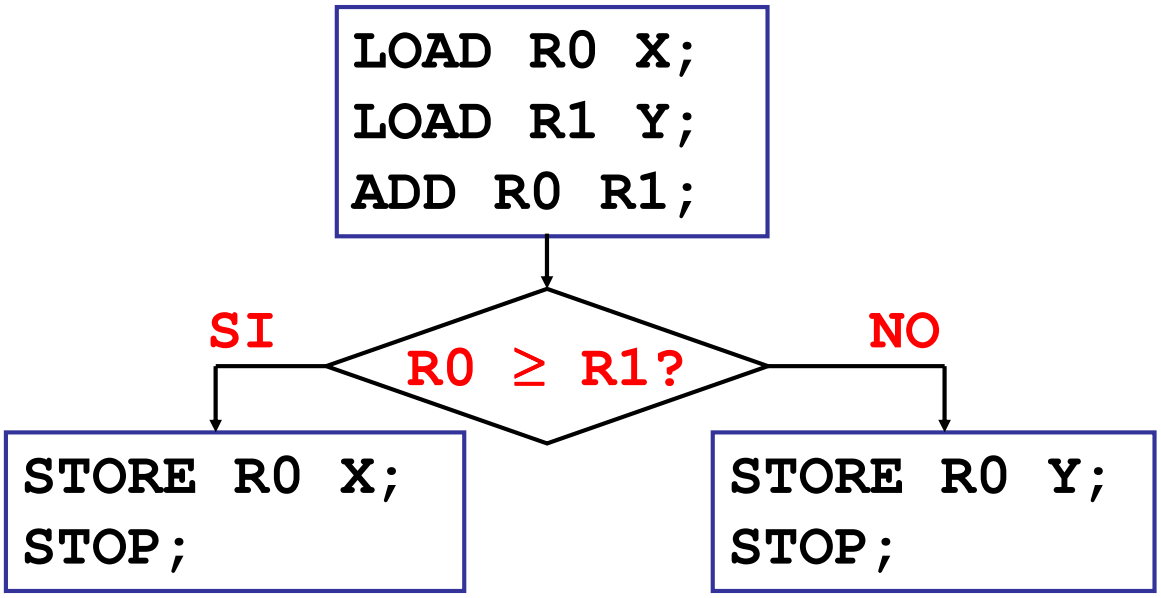
Altro esempio

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)

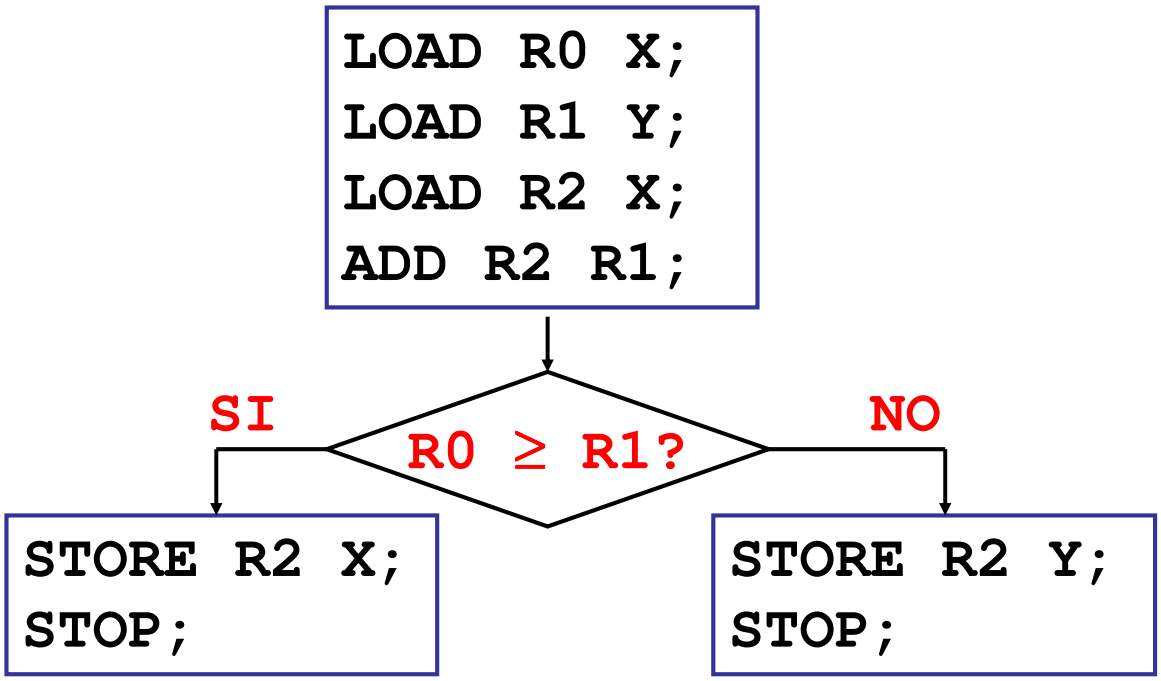
Algoritmo 1



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 ← valore **ASSEGNAMENTO**

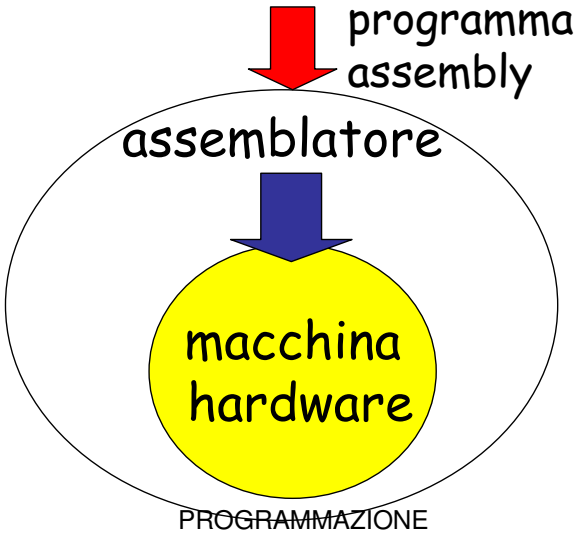
2) TEST di una condizione che determina quale azione intraprendere successivamente

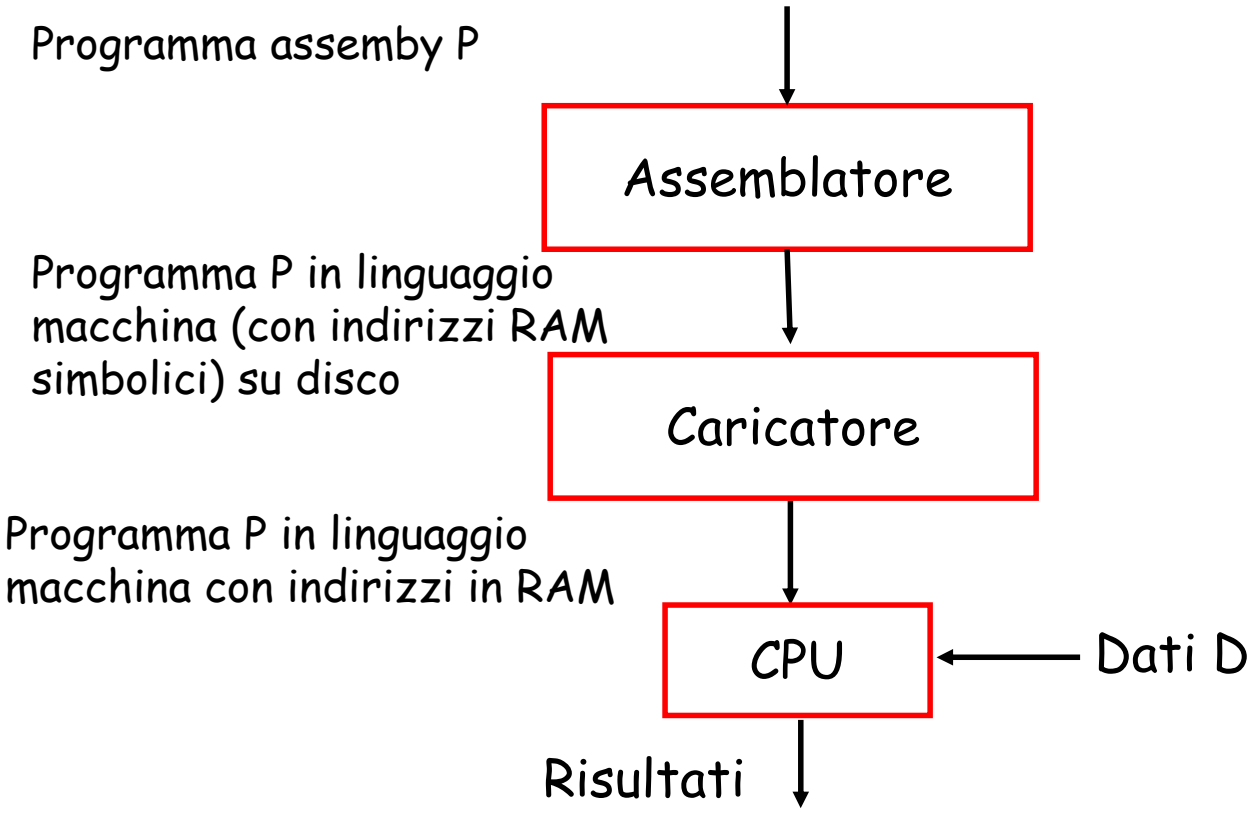
3) CICLO che si ripete finchè una data condizione è vera

SONO COSTRUTTI FONDAMENTALI in ogni LINGUAGGIO di PROGRAMMAZIONE

La CPU non "capisce" l'assembly !!

Il programma assembly deve essere tradotto in un programma macchina




Programma assemby P
Assemblatore
Programma P in linguaggio macchina (con indirizzi RAM simbolici) su disco
Caricatore
Programma P in linguaggio macchina con indirizzi in RAM
CPU
Dati D
Risultati