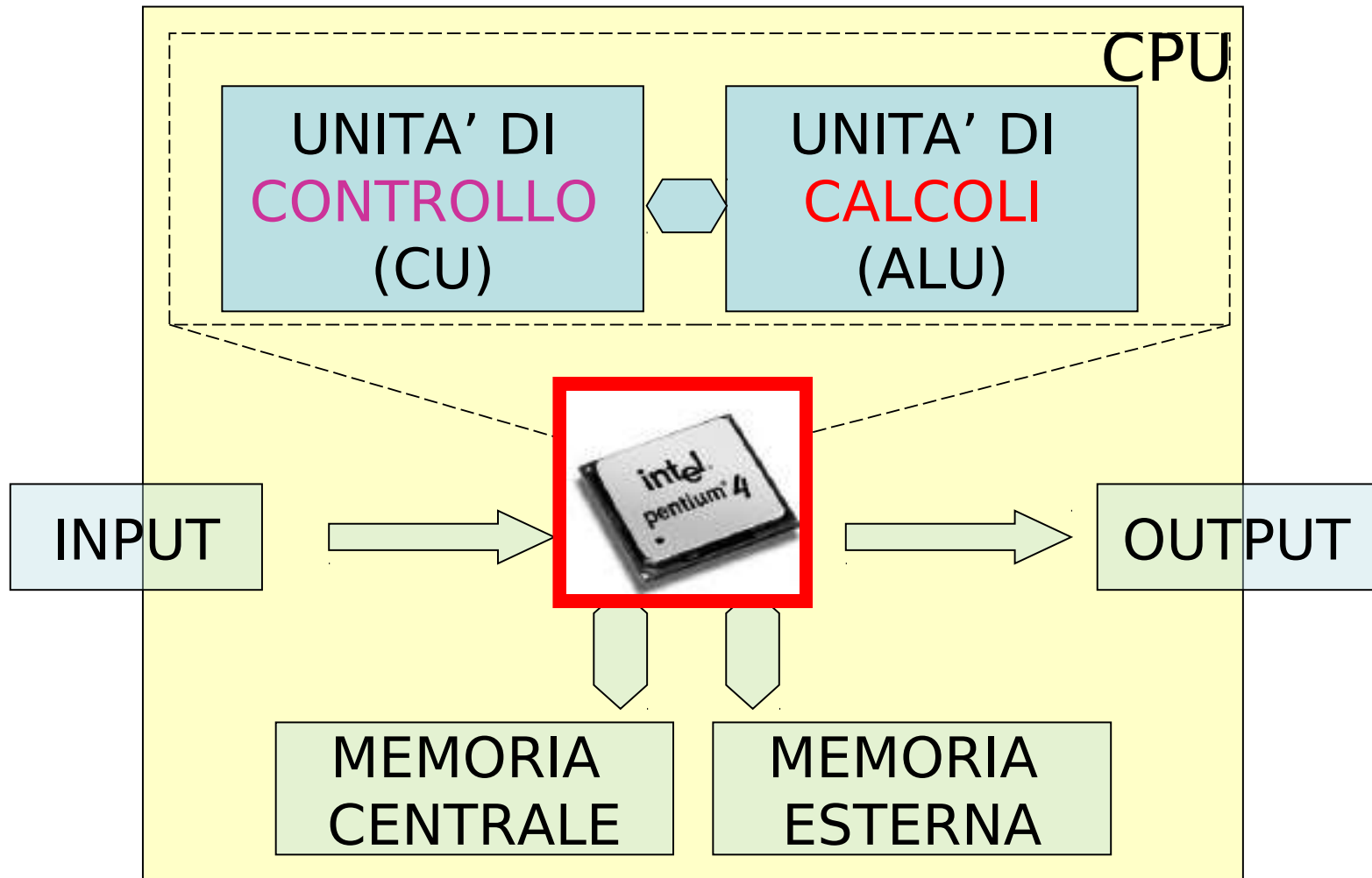


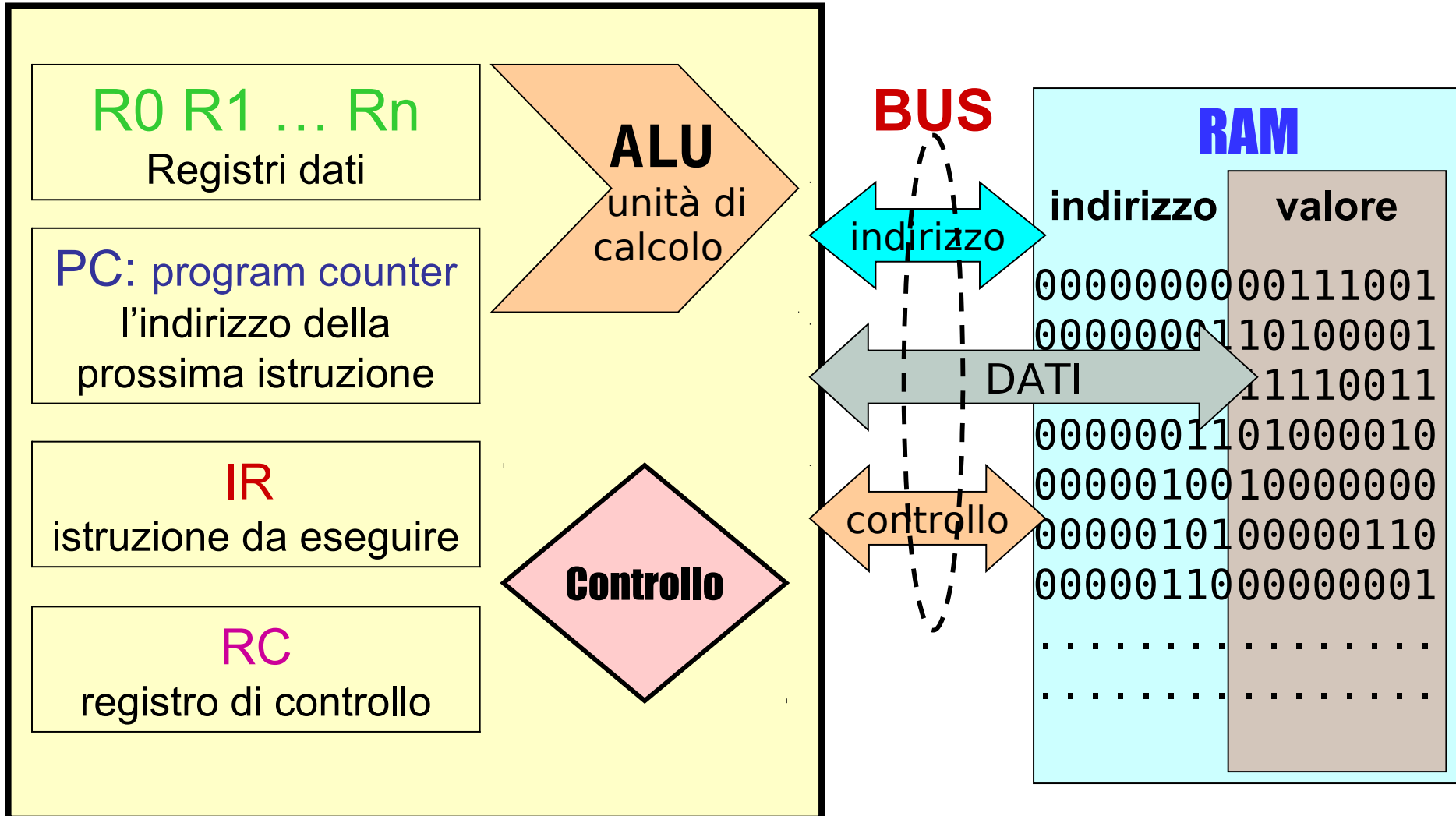
# II CPU

**Ivlin Stoianov**  
[www.stoianov.it](http://www.stoianov.it)

# La struttura del Calcolatore



# CPU



# Il CPU esegue istruzioni

- i) *Preleva* un'istruzione dalla memoria, dall'indirizzo indicato nel **PC** e ne pone nel **IR**;
- ii) *Decodifica* l'istruzione nel **IR**, individuando un circuito hardware da attivare.
- iii) Il circuito selezionato *esegue* l'istruzione, recuperando eventuali dati (dalla memoria o registri), poi depositando il risultato (nella memoria o un registro) e quindi aumentando il **PC**.

# Categorie di istruzioni

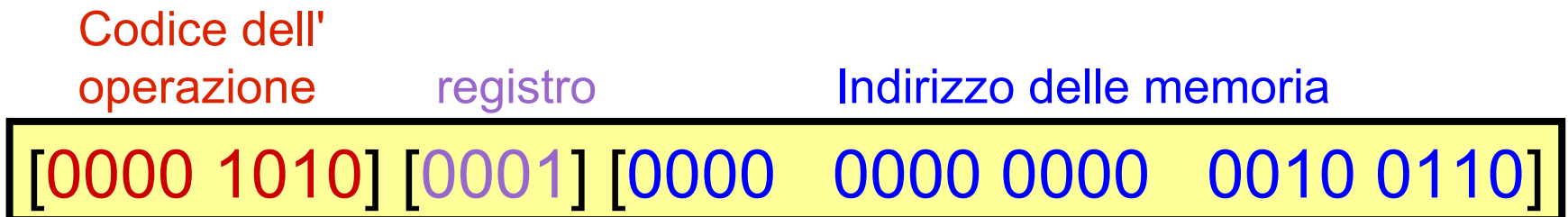
- **Trasferimento dati** (tra memoria e registri)
- **Operazioni aritmetiche** (per es., somma, ...)
- **Controllo del flusso** (confronto, salto, ...)
- **Input / Output** dai dispositivi I/O

# Codifica delle istruzioni

Una sequenza di bit (ad es., 4 byte) che contiene:

- 1) Il codice binario di un operazione da eseguire
- 2) parametri dell'operazione:  
(registri dati, valori, indirizzi nella memoria)

Esempio:



# Istruzioni

## Trasferimento dati



*Codice operazione*    *indice-registro*    *Indirizzo della memoria*

[0000 0000] [rrrr] [aaaa aaaa aaaa aaaa aaaa]

- carica in un registro-dati **rrrr** il contenuto della cella di memoria con indirizzo **a..a**

[1000 0000] [rrrr] [xxxx xxxx xxxx xxxx xxxx]

- carica nel registro dati **rrrr** il valore **xx..xx**

[0000 0001] [rrrr] [aaaa aaaa aaaa aaaa aaaa]

- carica nella cella di memoria con indirizzo **a..a** il contenuto del registro dati **rrrr**



# Istruzioni Aritmetiche

**0000 0010** somma i valori binari di tipo *integer* contenuti nei registri *iiii* e *jjjj*. Il risultato resta nel registro *iiii*

Operazione    registro1    registro2                    bit non-utilizzati

[0000 0010] [iiii] [jjjj] [xxxxxxxx xxxxxxxx]

**0000 0100** sottrazione (integer)

**0000 0110** moltiplicazione (integer)

**0000 1000** divisione (integer)

Operazioni aritmetiche di dati tipo *float* (numeri reali)

**0000 0011** somma (float)

**0000 0101** sottrazione (float)

**0000 0111** moltiplicazione (float)

**0000 1001** divisione (float)





# Istruzioni Aritmetiche tra registri e costanti

**1000 0010** aggiunge al contenuto del registro *iii* una costante contenuta nella istruzione stesa

Operazione    registro dati                    valore (20 bit)

[1000 0010] [iii] [ xxxx xxxx xxxx xxxx xxxx ]

**1000 0100** sottrazione (integer)

**1000 0110** moltiplicazione (integer)

**1000 1000** divisione (integer)

Operazioni aritmetiche di dati tipo **float** (numeri reali)

**1000 0011** somma (float)

**1000 0101** sottrazione (float)

**1000 0111** moltiplicazione (float)

**1000 1001** divisione (float)

# Un programmino nel codice binario

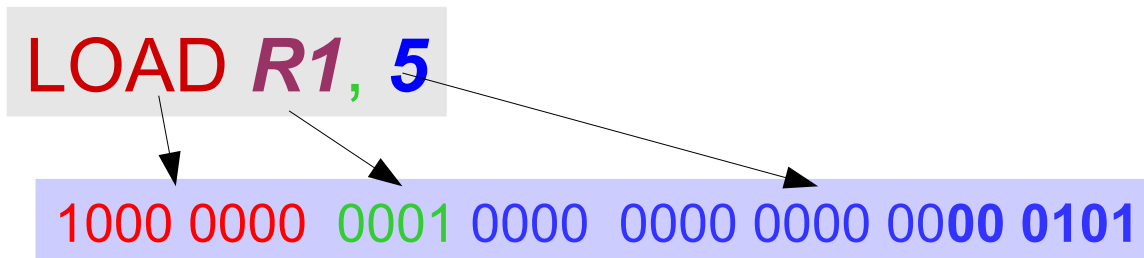
```
0000 0000 0000 0000 0000 0000 0100 0000
0000 0000 0001 0000 0000 0000 0100 0001
0000 0110 0000 0001 0000 0000 0000 0000
0000 0001 0000 0000 0000 0000 0100 0010
```

Che cosa fa ? ... :-)

# Codice Mnemonico

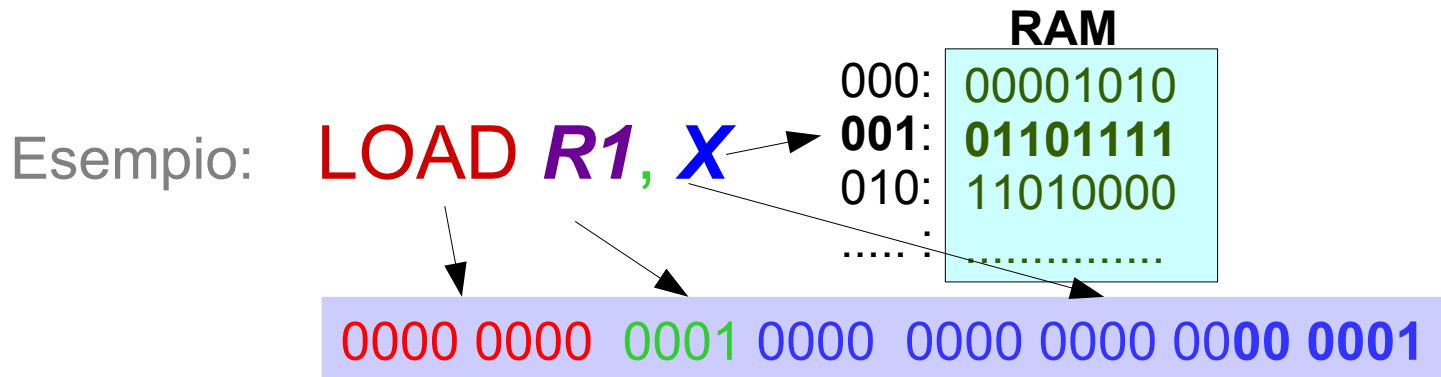
- Programmare nel codice binario è disumano.
- Molto più facile se usare codici mnemonici:
  - istruzioni (ad es., “0000 0001”=STORE )
  - Registri (0001 = R1)
- Il CPU capisce solo il codice binario ... ?
- Un programma – **compilatore** – traduce il codice mnemonico nel codice binario del CPU.

Esempio:

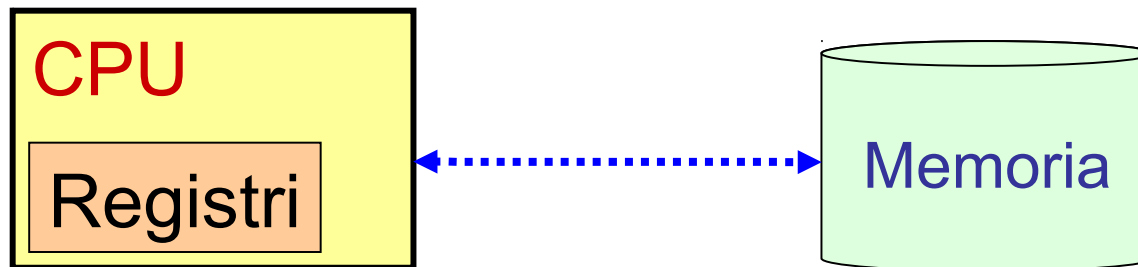


# Riferimenti mnemonici degli indirizzi nella RAM

- Nei codici mnemonici, anche gli *indirizzi nella RAM* dei dati e degli istruzioni sono riferiti per nome:
  - I riferimenti mnemonici degli indirizzi della RAM che contengono dati si chiamano **variabili**.
  - I riferimenti mnemonici degli indirizzi nella RAM che contengono istruzioni del CPU si chiamano **etichette**.
- I **compilatori** traducono l'insieme di codici mnemonici di { **istruzioni**, **variabili** e **etichette** } in codice binario.



# Istruzioni mnemonici per trasferimento dati



- **LOAD  $R$ ,  $Var$**  carica nel registro-dati  $R$  il contenuto della variabile  $Var$  (una cella di memoria)
- **LOAD  $R$ ,  $cost$**  carica nel registro-dati  $R$  la costante  $cost$
- **LOAD  $R$ ,  $R1$**  carica nel registro-dati  $R$  il contenuto del registro dati  $R1$
  
- **STORE  $R$ ,  $var$**  copia il contenuto del registro dati  $R$  nella variabile  $Var$  (una cella di memoria)



# Istruzioni mnemonici per l'Aritmetica binaria

- **ADD R1, R2** somma i valori di tipo *integer* (*interi*) contenuti nei registri **R1** e **R2**. Il risultato resta nel registro **R1**



- **ADD R, cost** aggiunge al contenuto del registro **R** la costante **cost**

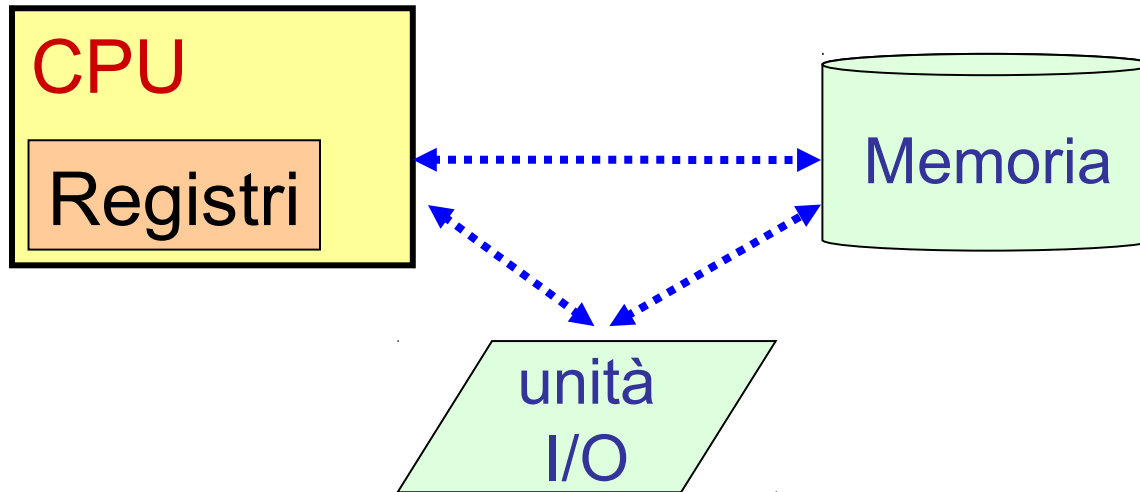


- **SUB R1, R2** e **SUB R1, cost** sottrazione (integer)
- **MUL R1, R2** e **MUL R1, cost** moltiplica (integer)
- **DIV R1, R2** e **DIV R1, cost** divide (integer)
- **FADD, FSUB, FMUL, FDIV:** operazioni su dati *float*

# Il “programmino” nel codice Mnemonico



# Input / Output



- **READ  $u$ ,  $Var$**  legge un byte dall'unità  $u$  e lo memorizza nella variabile  $Var$
- **WRITE  $u$ ,  $Var$**  scrive il contenuto della variabile  $Var$  all'unità  $u$ .
- Unità I/O: (0 stdin: tastiera), (1 stdout: video), ecc.



# Un programma completo: input, elaborazione, output

dati nella RAM  
riferiti per nome

```
X: INT
Y: INT

READ  STINP, X
READ  STINP, Y
LOAD  R0, X
LOAD  R1, Y
ADD   R0, R1
STORE R0, X
WRITE STOUT, X
```



# Confronto

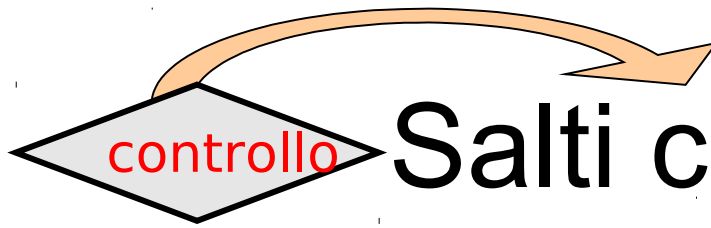
- **CMP *R1*, *R2*** confronta i valori-*integer* dei registri *R1* e *R2*  
**Risultato:** Se  $Rr1 < Rr2$  , memorizza -1 nel registro *RC*  
Se  $Rr1 = Rr2$  , memorizza 0  
Se  $Rr1 > Rr2$  , memorizza +1
- **CMP *R1*, *cost*** confronta il registro *r1* con *la costante cost*
- **FCMP *R1*, *R2*** confronta valori di tipo *float*

**CMP** Operazione registro1 registro2 bit non-utilizzati

[0010 0000] [iiii] [jjjj] [xxxxxxxx xxxxxxxx]

**CMP** Operazione registro costante

[1010 0000] [iiii] [xxxx xxxx xxxx xxxx xxxx]



# Salti condizionali

- **BRLT *addr*** se  $RC = -1$ , spostare il PC all'*addr*
- **BRLE *addr*** se  $RC \leq 0$ , spostare il PC all'*addr*
- **BREQ *addr*** se  $RC = 0$ , spostare il PC all'*addr*
- **BRNE *addr*** se  $RC \neq 0$ , spostare il PC all'*addr*
- **BRGT *addr*** se  $RC = 1$ , spostare il PC all'*addr*
- **BRGE *addr*** se  $RC \geq 0$ , spostare il PC all'*addr*

**BRLT** Operazione bit non-utilizzati nuovo indirizzo per il PC

[0100 0001] [xxxx] [aaaa aaaa aaaa aaaa aaaa]

# Branch e STOP

- BRANCH *addr* spostamento del PC all'indirizzo *addr*
- STOP termina l'esecuzione del programma

**Branch** Operazione nuovo indirizzo per il PC

```
[1001 0001] [xxxx] [aaaa aaaa aaaa aaaa aaaa]
```

**STOP** Operazione bit non-utilizzati

```
[1001 0001] [xxxx xxxx xxxx xxxx xxxx xxxx]
```

# Esempio: Elaborazione condizionale

## Controllo del flusso

dati nella RAM  
riferiti per nome

X: INT  
Y: INT

READ STINP, X

READ STINP, Y

LOAD R0, X

LOAD R1, Y

COMP R0, R1

BRLE PRINT

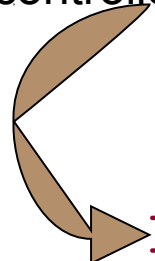
STORE R1, X

STORE R0, Y

PRINT:WRITE STOUT, X

WRITE STOUT, Y

(registro di controllo) RC



ETICHETTA: indirizzo di un  
istruzione riferito per nome