

# Il linguaggio del calcolatore

## Fondamenti di Informatica A

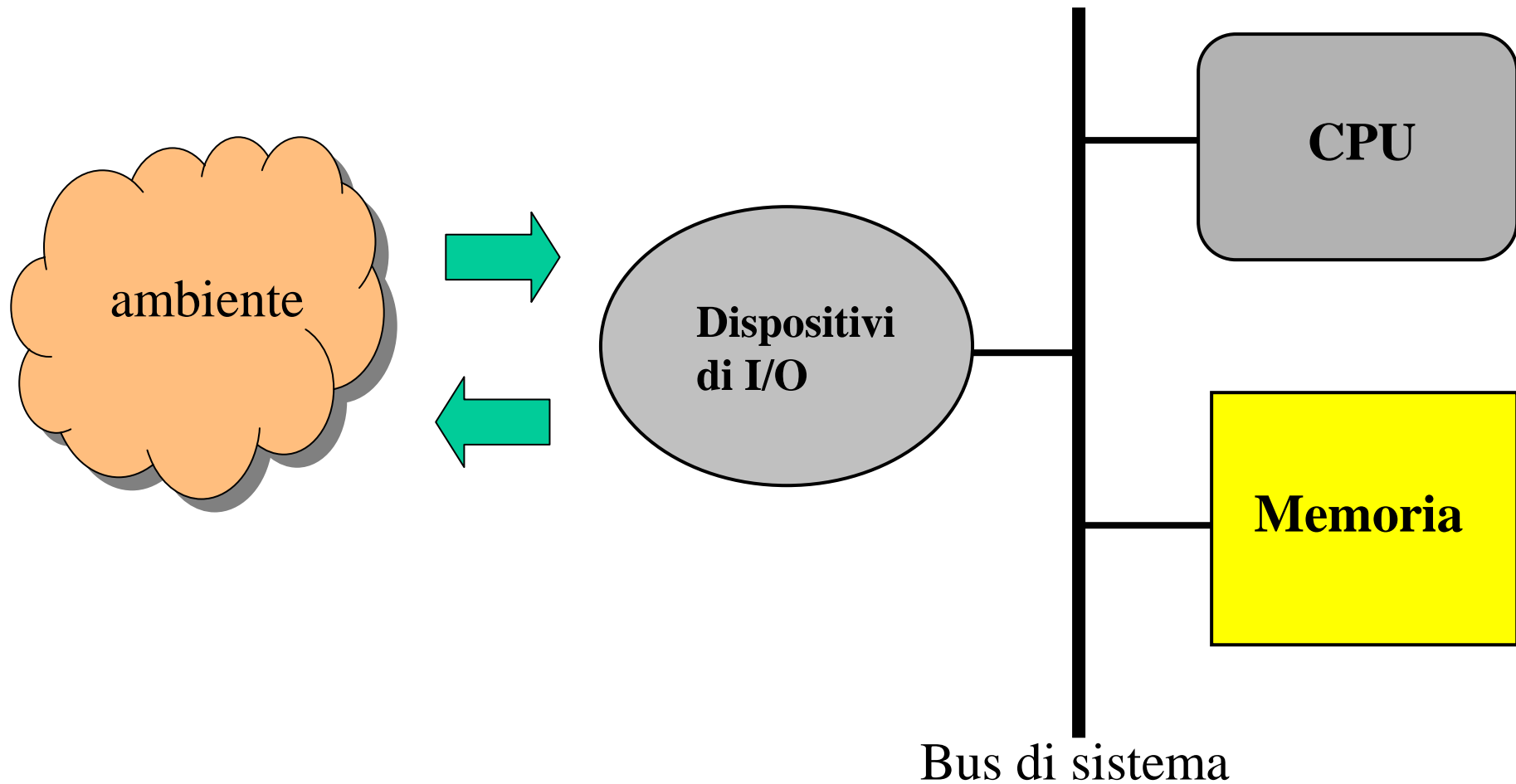
*Percorso di Preparazione agli Studi di Ingegneria*

Università degli Studi di Brescia

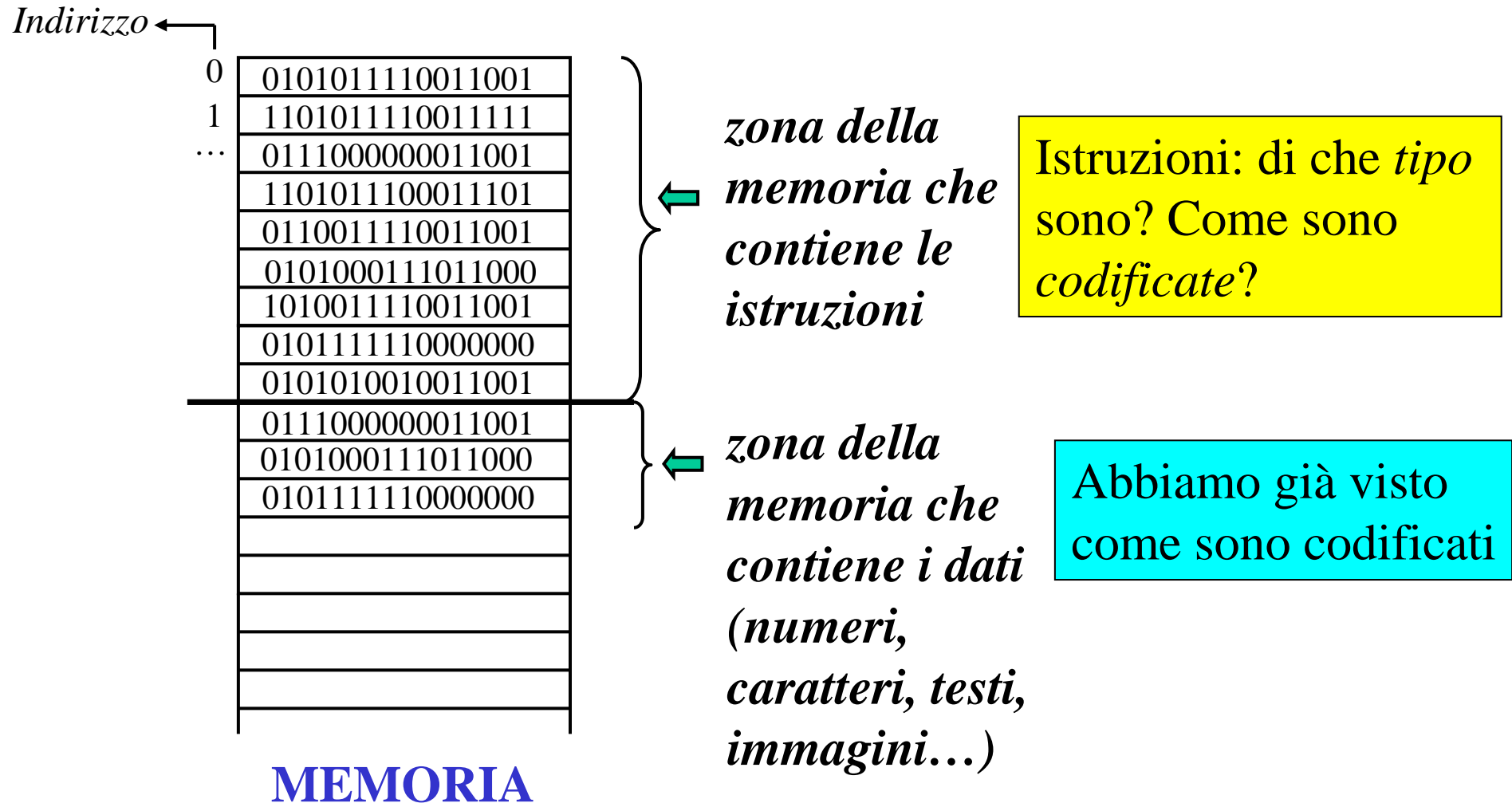
*Docente: Massimiliano Giacomini*

## Richiamo sull'architettura del calcolatore

**CPU** = *Central Processing Unit (Unità centrale)*  
detta oggi Microprocessore o processore



# Programma e dati in memoria: rivisitazione



# Il linguaggio macchina

- Linguaggio macchina: costituito da **istruzioni macchina**, eseguite dalla CPU
- Ogni CPU ha un proprio linguaggio macchina (**ISA – Instruction Set Architecture**): per esempio, le istruzioni dei processori Intel X86 sono diverse da quelle del processore MIPS
  - esistono CPU di marca diversa con diversa struttura fisica che risultano **compatibili** (es. Intel e AMD)
- Le istruzioni del linguaggio macchina sono costituite da stringhe di bit, suddivise in:
  - **Codice operativo** → tipo istruzione
  - **Operandi** → indicano i dati su cui l'istruzione opera (sorgenti) e dove memorizzare il risultato (destinazione)

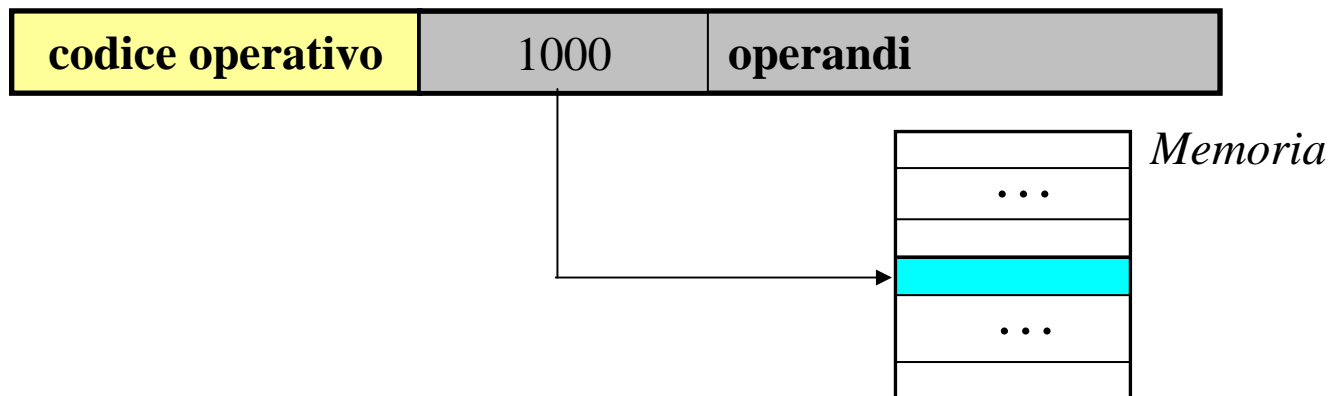


# Operandi

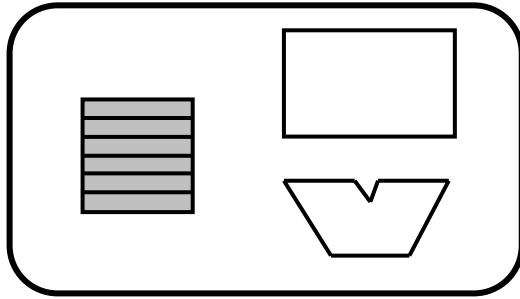
- Ogni istruzione specifica operandi sorgente e destinazione
- Un operando può essere:
  - Un valore indicato nel campo dell'istruzione (*immediato*)



- Una parola della memoria centrale (*operando in memoria*)



- Un registro dell'unità centrale (*operando registro*)



## I registri della CPU

- Sono **celle di memoria a n bit**

0 000000001010000100000000000011000  
1 000011001010000100000000000011000

Identificati da  
un numero

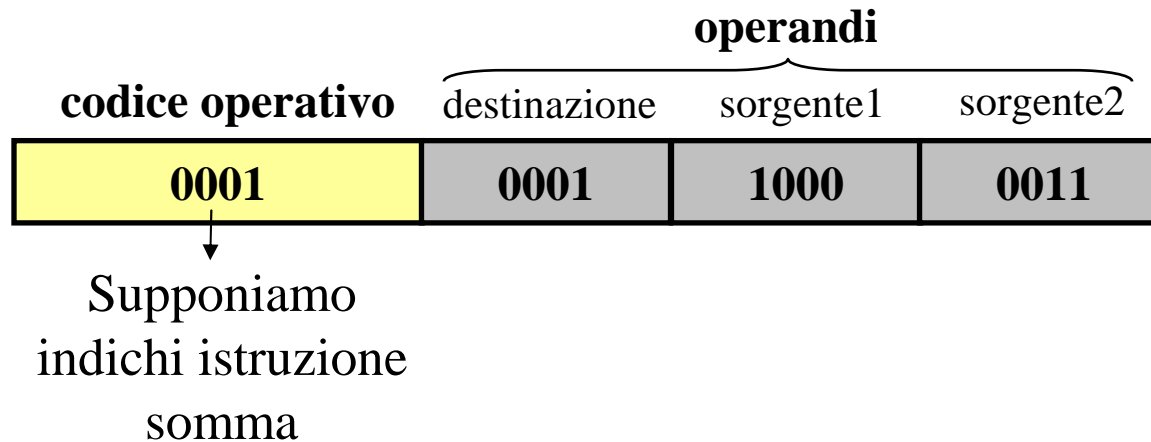
...

- Quanti bit per registro?: esistono processori a 16, 32, 64 bit...
- Quanti registri? Dipende dal processore

POSSONO MEMORIZZARE OPERANDI DELLE ISTRUZIONI

➡ Costituiscono la “memoria a breve termine” del calcolatore

## Un esempio ipotetico (con operandi registro)



⇒ Somma i valori dei registri 8 e 3, mettendo il risultato nel registro 1  
(il precedente valore del registro 1 è sovrascritto)

- Quanti bit servono per il campo degli operandi registro?

Esempio: CPU con 32 registri, occorrono 5 bit per identificare uno dei registri ⇒ nelle istruzioni i campi per gli operandi registro saranno di 5 bit

# Il linguaggio assembler


- Usare direttamente il formato binario per scrivere (e leggere) programmi sarebbe impraticabile  $\Rightarrow$  si usa il *linguaggio assembly* (o *assembler*)
- Il linguaggio assembler è la *rappresentazione simbolica* della codifica binaria usata dal calcolatore (linguaggio macchina)
- L'assembler è più leggibile:
  - utilizza *codici operativi simbolici* (anziché bit) che richiamano direttamente il significato di una istruzione (p.es. ADD al posto di 0001)
  - permette l'utilizzo di *etichette* per identificare gli indirizzi di parole di memoria che contengono istruzioni o dati (in questo caso le etichette possono essere indicate come “nomi di variabili”)
- *Assemblatore*: traduce linguaggio assembler in linguaggio macchina



# Linguaggio macchina vs. Linguaggio assembler

**Codice macchina di una procedura  
che calcola e stampa la somma dei  
quadrati degli interi fra 0 e 100**

```
0010011110111101111111111111100000
10101111110111111100000000000010100
1010111111010010000000000000100000
1010111111010010100000000000100100
101011111101000000000000000011000
101011111101000000000000000011100
100011111101011100000000000011100
100011111101110000000000000011000
000000011100111000000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
101011111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
101011111101110010000000000011000
00111100000001000001000000000000
100011111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
100011111101111110000000000010100
001001111101111010000000000100000
0000001111100000000000000001000
00000000000000000001000000100001
```

  
Traduzione:  
programma  
chiamato  
“assemblatore”

**Codice assembler di una procedura  
che calcola e stampa la somma dei  
quadrati degli interi fra 0 e 100**

```
addiu    $29, $29, -32
sw        $31, 20($29)
sw        $4, 32($29)
sw        $5, 36($29)
sw        $0, 24($29)
sw        $0, 28($29)
lw        $14, 28($29)
lw        $24, 24($29)
multu    $14, $14
addiu    $8, $14, 1
slti     $1, $8, 101
sw        $8, 28($29)
mflo     $15
addu     $25, $24, $15
bne      $1, $0, -9
sw        $25, 24($29)
lui      $4, 4096
lw        $5, 24($29)
jal      1048812
addiu    $4, $4, 1072
lw        $31, 20($29)
addiu    $29, $29, 32
jr        $31
move     $2, $0
```

# Tipologie di istruzioni

- *Istruzioni aritmetico-logiche e di manipolazione di bit* (Elaborazione dati)
  - Somma, Sottrazione, Divisione, ...
  - And, Or, Xor, ...
  - Maggiore, Minore, Uguale, Minore o uguale, ...
- *Istruzioni di controllo*
  - Salti condizionati
  - Salti incondizionati
- *Istruzioni di trasferimento*
  - Trasferimento dati e istruzioni tra CPU e memoria
- *Istruzioni di ingresso e uscita*
  - Trasferimento dati e istruzioni tra CPU e dispositivi di ingresso/uscita (attraverso le relative interfacce)

## Esempi di istruzioni aritmetiche

add     \$r1, \$r3, \$r4     # somma il contenuto di \$r3 col contenuto  
                                  # di \$r4 e poni il risultato in \$r1

sub     \$r0, \$r1, \$r2     # sottrai dal contenuto di \$r1 il contenuto di  
                                  # \$r2 e poni il risultato in \$r0

*Stato registri: prima  
dell'esecuzione*

\$r0	4
\$r1	3
\$r2	5
\$r3	10
\$r4	2

...

*Stato registri: dopo la  
prima istruzione*

\$r0	4
\$r1	12
\$r2	5
\$r3	10
\$r4	2

...

*Stato registri: dopo la  
seconda istruzione*

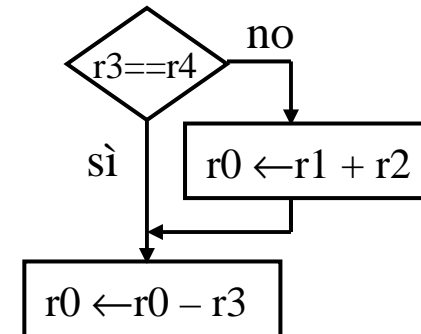
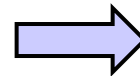
\$r0	7
\$r1	12
\$r2	5
\$r3	10
\$r4	2

...

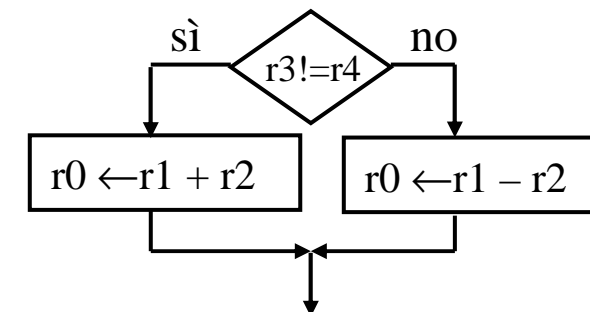
## Esempi di istruzioni di controllo

- Salto incondizionato *j* (jump)
- Salto condizionato *beq* (branch if equal) e *bne* (branch if not equal)

**beq**     **\$r3, \$r4, L1**  
add     \$r0, \$r1, \$r2  
L1: sub     \$r0, \$r0, \$r3



**bne**     **\$r3, \$r4, Allora**  
sub     \$r0, \$r1, \$r2  
**j**     **Esci**  
Allora: add     \$r0, \$r1, \$r2  
Esci: ...

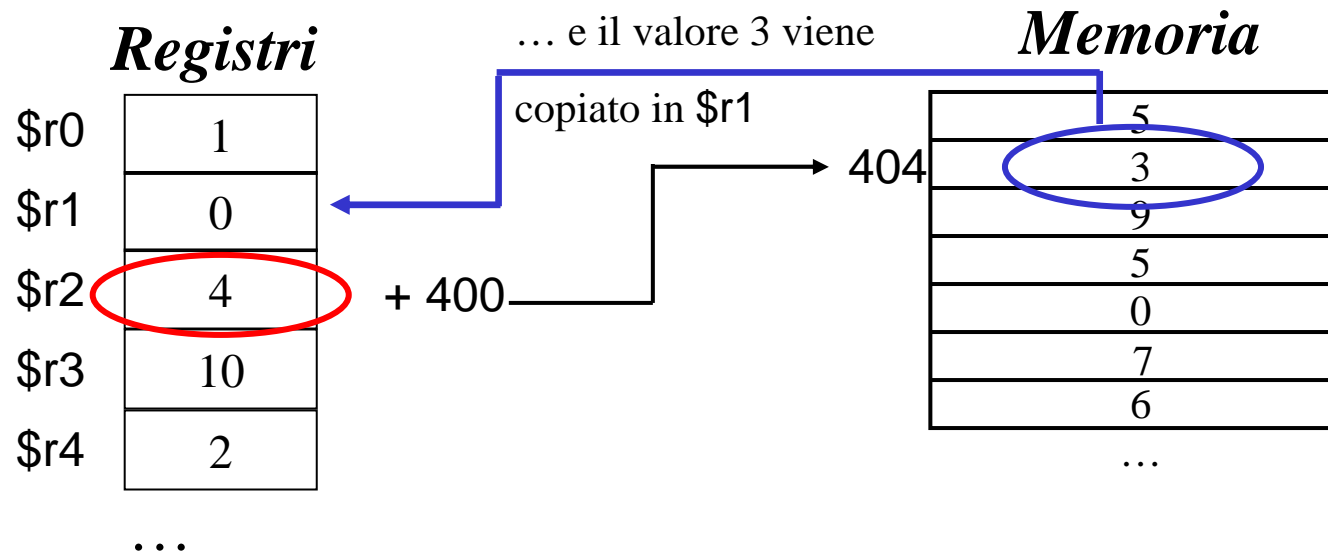


## Esempi di istruzioni di trasferimento

- Istruzione lw (*load word*)

lw \$r1, \$r2, 400

significa: carica nel registro \$r1 il contenuto della cella di memoria il cui indirizzo si trova sommando 400 al contenuto di \$r2



- Istruzione sw (store word):  
sw \$r1, \$r2, 400 salva \$r1 in memoria (indirizzo \$r2+400)

# La traduzione in linguaggio macchina binario

- Traduzione dei codici simbolici in corrispondenti codici binari:  
mediante una tabella dei codici  
(es: ad *add* corrisponde il codice operativo 00..1)
- Traduzione degli indirizzi simbolici (etichette) in indirizzi effettivi:
  - decisione di dove memorizzare il programma e i dati
  - individuazione, per ogni etichetta e nome di variabile, del corrispondente indirizzo effettivo
  - traduzione delle etichette simboliche nei corrispondenti indirizzi in binario