

Il linguaggio del calcolatore

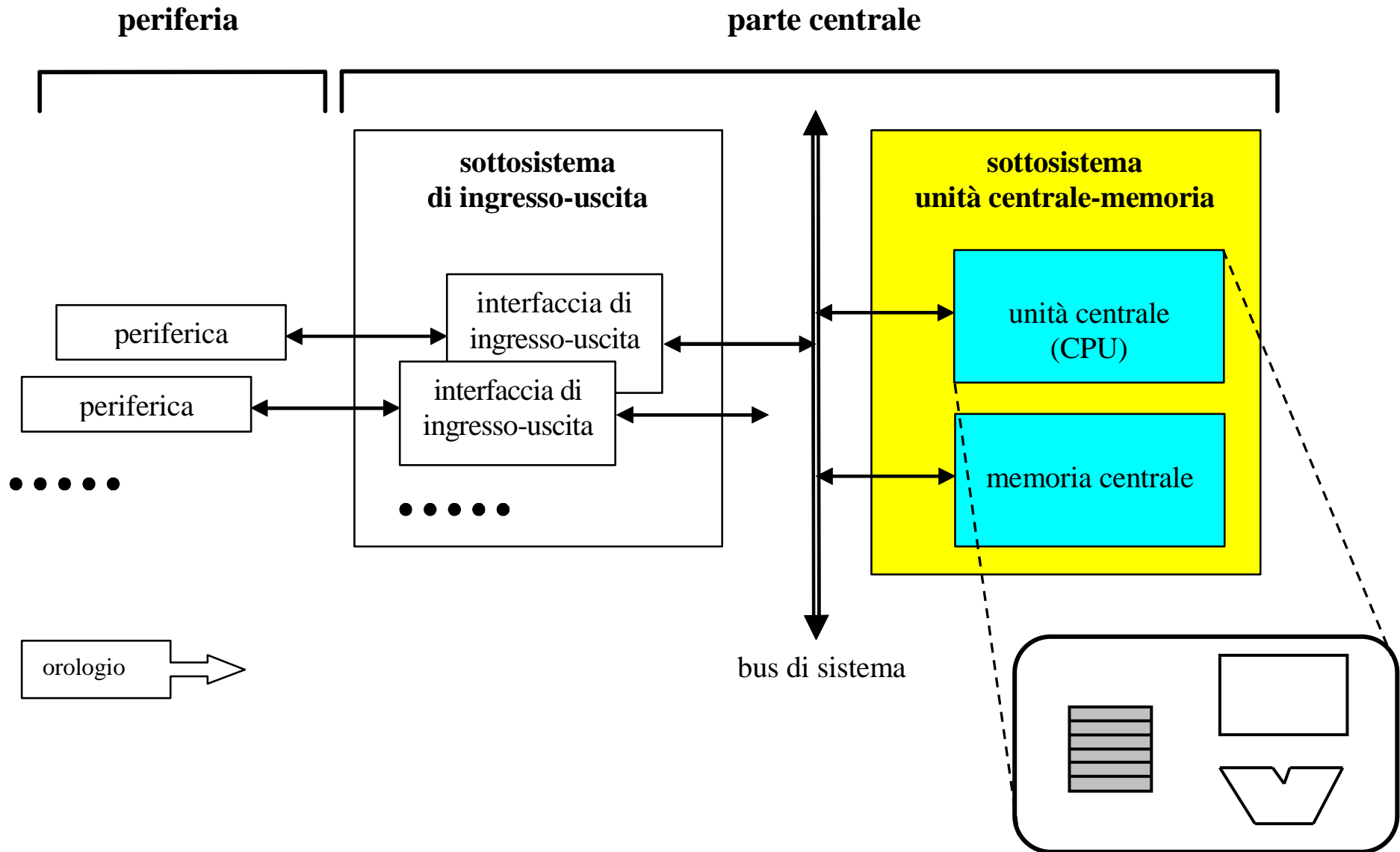
Fondamenti di Informatica A

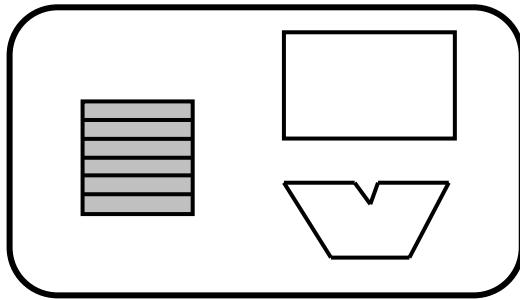
Percorso di Preparazione agli Studi di Ingegneria

Università degli Studi di Brescia

Docente: Massimiliano Giacomini

Architettura elementare: i registri della CPU

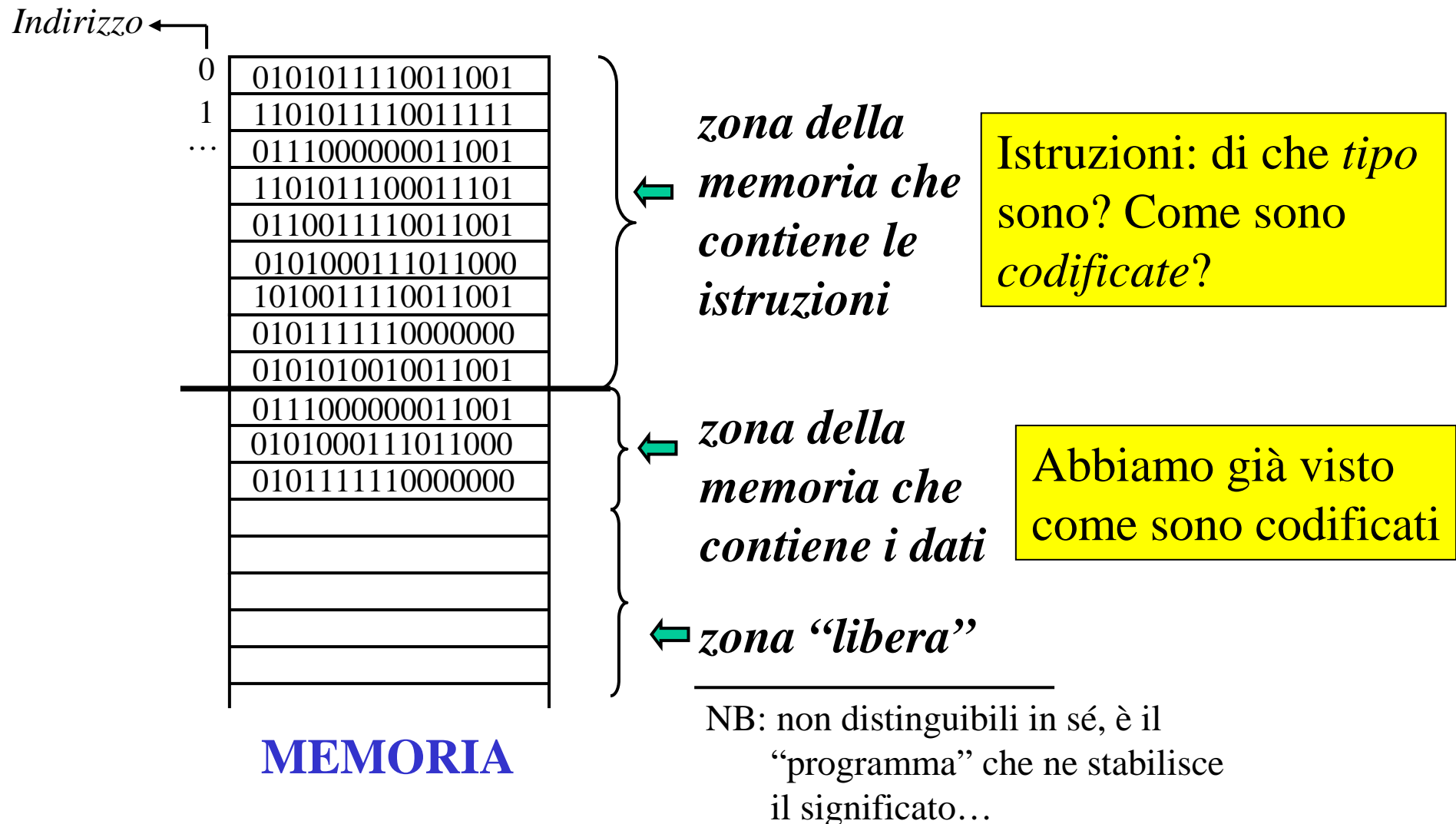




I registri della CPU

- Memoria a **breve termine** del calcolatore
- Sono **celle di memoria** utilizzate per immagazzinare le informazioni necessarie per l'esecuzione delle istruzioni
- **Registri per memorizzare gli operandi** delle operazioni da eseguire
- Il loro numero dipende dal tipo di CPU
- Gli operandi delle istruzioni aritmetico/logiche possono contenere un **indirizzo di registro**
- **Esempio**: CPU con 32 registri, occorrono 5 bit per identificare uno dei registri, nelle istruzioni ci saranno gruppi di 5 bit per gli operandi

Programma e dati in memoria: rivisitazione



Linguaggio macchina

- Il linguaggio macchina è il linguaggio per cui la CPU si comporta da **esecutore**
- Ogni CPU è caratterizzata funzionalmente dal suo 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 (**istruzioni macchina**) sono costituite da stringhe di bit, suddivise in:
 - **Codice operativo** → tipo istruzione
 - **Operandi** → indirizzi dove recuperare i dati

Istruzioni e tipologie



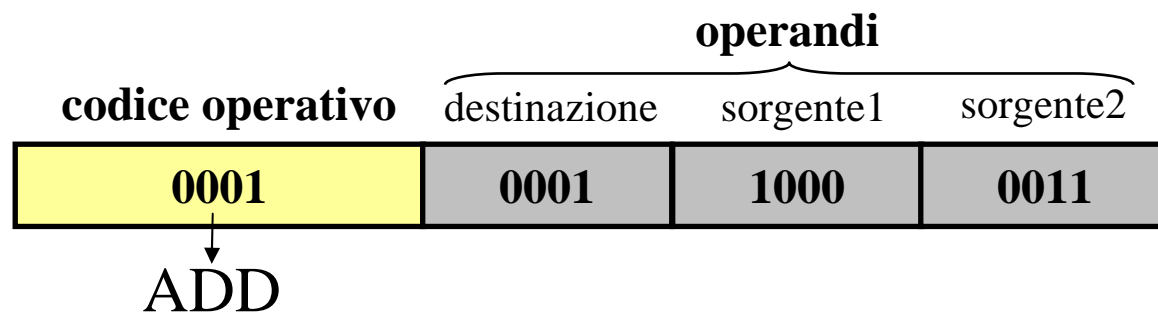
Formato
generale delle
istruzioni

- *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)

Operandi

- Possono essere:
 - Un valore indicato nel campo dell'istruzione (immediato)
 - Un registro dell'unità centrale (operando registro)
 - Una parola della memoria centrale (operando in memoria)
- Ogni istruzione specifica operandi sorgente e destinazione

Esempio ipotetico con operandi registro



Somma i valori dei registri 8 e 3, mettendo il risultato nel registro 1

NB: usare direttamente il formato binario per scrivere (e leggere) programmi sarebbe impraticabile...

Linguaggio 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

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*

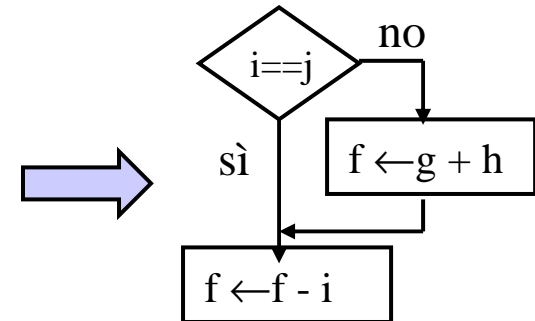
\$r0	7
\$r1	12
\$r2	5
\$r3	10
\$r4	2

...

Esempio: salto condizionato

(beq = branch if equal)

- 1 Se $(i = j)$ allora vai al passo 3
- 2 $f \leftarrow g + h$;
- 3 $f \leftarrow f - i$;

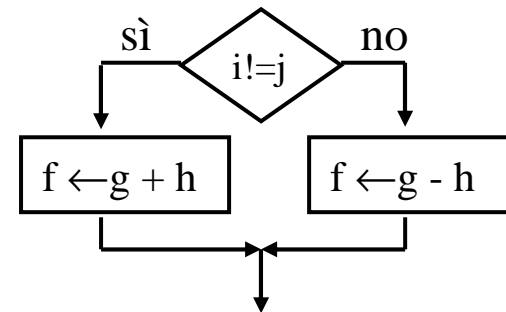
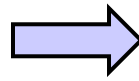


Supponendo che f, g, h, i, j corrispondano ai registri $\$r0, \$r1, \$r2, \$r3, \$r4$, il corrispondente programma in linguaggio assembler potrebbe essere il seguente

```
beq    $r3, $r4, L1    # va a L1 se i è uguale a j
add    $r0, $r1, $r2    # f = g + h
L1:    sub   $r0, $r0, $r3  # f = f - i
```

Esempio: salto condizionato e salto incondizionato

Se $(i \neq j)$ allora $f \leftarrow g + h$
altrimenti $f \leftarrow g - h$



Supponendo che f, g, h, i, j corrispondano ai registri $\$r0, \$r1, \$r2, \$r3, \$r4$, il programma potrebbe essere il seguente

```

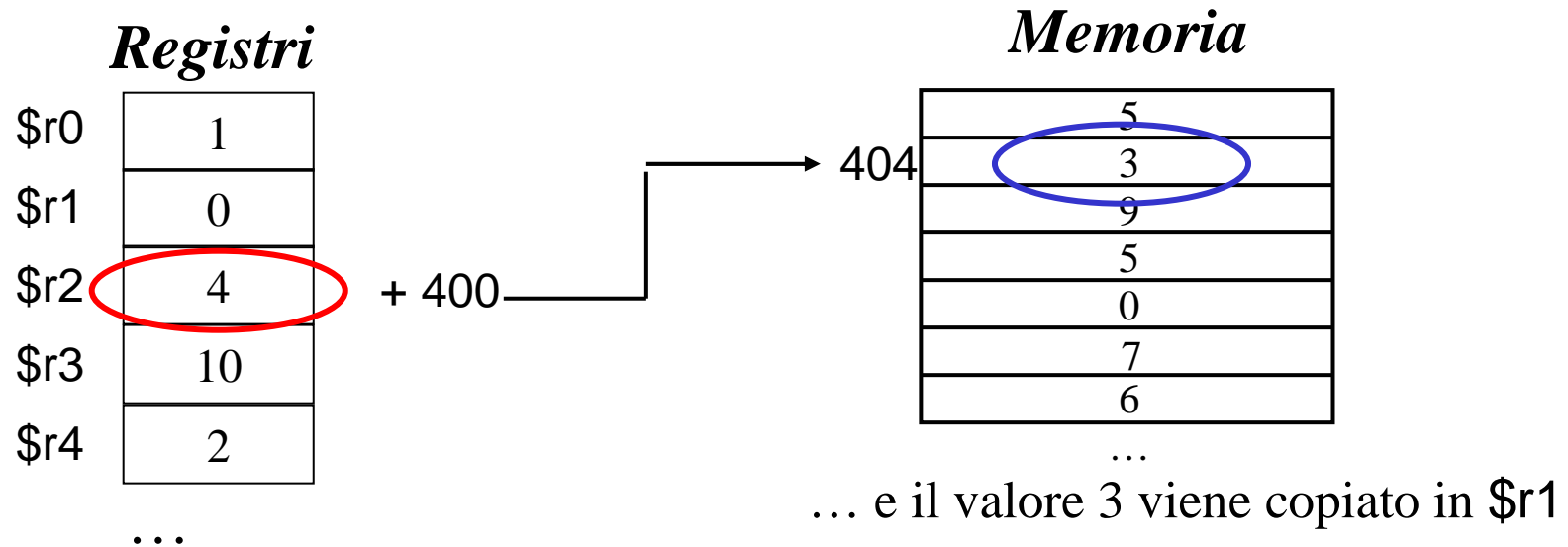
    bne    $r3, $r4, Allora # branch if not equal
    sub    $r0, $r1, $r2
    j      Esci    # salto incondizionato (jump)
Allora: add    $r0, $r1, $r2
Esci:  ...
```

Esempio: lettura dato da memoria

- 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`



Linguaggio macchina vs. Linguaggio assembler

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

```
001001111011110111111111111100000
101011111011111110000000000010100
1010111110100100000000000000100000
1010111110100101000000000000100100
101011111010000000000000000011000
101011111010000000000000000011100
100011111010111000000000000011100
100011111011100000000000000011000
000000011100111000000000000011001
001001011100100000000000000000001
00101001000000010000000001100101
101011111010100000000000000011100
000000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
101011111011100100000000000011000
00111100000001000001000000000000
100011111010010100000000000011000
000011000001000000000000011101100
00100100100001000000010000110000
100011111011111100000000000010100
001001111011111010000000000010000
00000011111000000000000000001000
000000000000000000001000000100001
```


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
```

La traduzione in linguaggio macchina binario

- Traduzione dei codici simbolici in corrispondenti codici binari: mediante una tabella dei codici
- Traduzione degli indirizzi simbolici (etichette) in indirizzi effettivi:
 - tabella dei simboli (elenca etichette e nomi di variabili)
 - decisione di dove memorizzare il programma e i dati
 - individuazione, per ogni etichetta e nome di variabile, del corrispondente indirizzo effettivo
- L'assegnazione degli indirizzi effettivi in memoria è detta *rilocalizzazione* e può essere determinata dal programmatore mediante opportune direttive all'assemblatore (es. ORGP e ORGD)