

 **POLITECNICO DI MILANO**



# Introduzione al linguaggio macchina

Fondamenti di Informatica

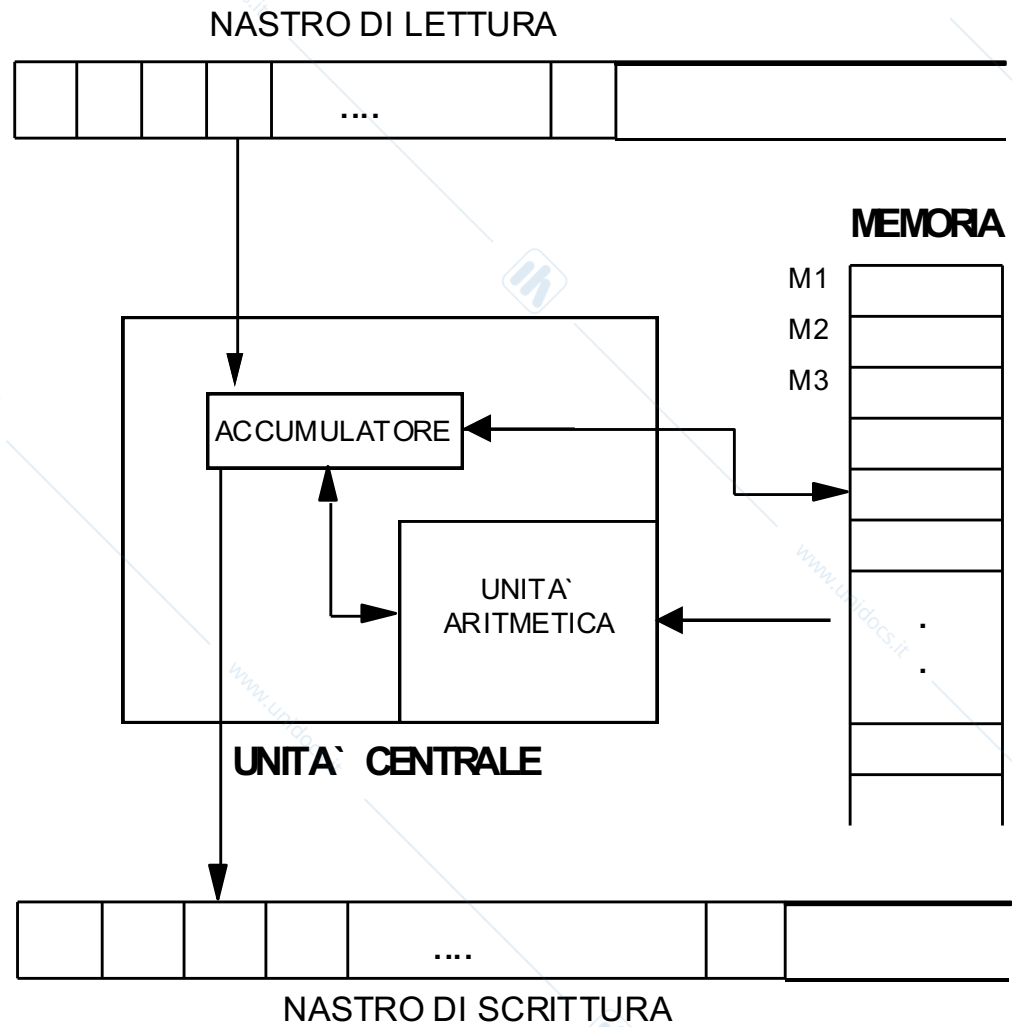
**Daniele Loiacono**

# Come è fatto il linguaggio macchina?

# Progettare il linguaggio macchina

- ❑ Il linguaggio macchina è un linguaggio di basso livello che viene progettato *attorno* al calcolatore:
  - ▶ richiede quindi di conoscere esattamente la struttura del calcolatore (la sua architettura);
  - ▶ è specifico per ogni calcolatore (in genere per ogni famiglia di calcolatori);
  - ▶ permette di sfruttare al meglio le risorse fisiche della macchina.
  
- ❑ Non studieremo l'architettura e il linguaggio macchina di un reale calcolatore, ma ci *limiteremo* ad un modello astratto di calcolatore (seppur funzionante) e a progettare un linguaggio macchina *essenziale* per questo modello.

# Un modello semplificato di calcolatore



Quale istruzioni ci servono?

# Istruzioni I/O

READ

- ▶ legge un dato dal nastro di input e lo memorizza nell'accumulatore

WRITE

- ▶ scrive il contenuto dell'accumulatore sul nastro di uscita

# Gestione della memoria

LOAD  $x$

- ▶ carica il dato nella cella di memoria all'indirizzo  $x$  nell'accumulatore

STORE  $x$

- ▶ scrive l'informazione presente nell'accumulatore nella cella di memoria all'indirizzo  $x$

# Operazioni aritmetiche

ADD x

- ▶ somma il contenuto della cella di memoria all'indirizzo x all'accumulatore

SUB x

- ▶ sottrae il contenuto della cella di memoria all'indirizzo x all'accumulatore

MULT x

- ▶ moltiplica il contenuto della cella di memoria all'indirizzo x all'accumulatore

DIV x

- ▶ divide il contenuto della cella di memoria all'indirizzo x all'accumulatore

# Controllo di flusso

BR x

- ▶ salta direttamente all'istruzione x

BEQ x / BNE x

- ▶ salta all'istruzione x se il valore dell'accumulatore è uguale/diverso da 0

BL x / BLE x / BG x / BGE x

- ▶ salta all'istruzione x se il valore dell'accumulatore è minore/minore o uguale/maggiore/maggiore o uguale a 0

END

- ▶ termina il programma

Come traduciamo un semplice programma?

# Esempio

```
scanf ("%d", &x) ;  
scanf ("%d", &y) ;  
printf ("%d", x+y) ;
```

- ❑ Come lo traduciamo in linguaggio macchina (ipotizzando che `scanf` e `printf` lavorino con i nastri di lettura e scrittura) ?

# Esempio

```
scanf ("%d", &x) ;  
scanf ("%d", &y) ;  
printf ("%d", x+y) ;
```

- ❑ Come lo traduciamo in linguaggio macchina (ipotizzando che `scanf` e `printf` lavorino con i nastri di lettura e scrittura) ?

- |              |   |
|--------------|---|
| 1. READ      | [Primo numero nell'accumulatore]        |
| 2. STORE 101 | [Primo numero nella cella 101]          |
| 3. READ      | [Secondo numero nell'accumulatore]      |
| 4. ADD 101   | [Somma l'accumulatore alla cella 101]   |
| 5. WRITE     | [Scrive il contenuto dell'accumulatore] |
| 6. END       |   |

Ci servono altre istruzioni?

# Indirizzamento

- ❑ Tutte le istruzioni che interagiscono con la memoria possono sfruttare differenti approcci per accedere allo spazio di memorizzazione.
- ❑ Si definiscono tali approcci come :
  - ▶ Indirizzamento DIRETTO
  - ▶ Indirizzamento INDIRECTO
  - ▶ Indirizzamento ESPLICITO
- ❑ Se assumiamo l'istruzione generica ISTR otteniamo :
  - ▶ Indirizzamento **DIRETTO** ISTR 13 (LOAD 13)  
«carico direttamente la cella di memoria numero13»
  - ▶ Indirizzamento **INDIRETTO** ISTR@ 11 (LOAD@ 11)  
«carico il valore della cella all'indirizzo memorizzato nella cella di memoria numero11»
  - ▶ Indirizzamento **ESPLICITO** ISTR=14 (LOAD= 14)  
«carico direttamente il valore 14»

# Esempio: sommatoria N numeri

```
scanf ("%d", &n) ;  
s=0;  
for (i=0; i<n; i++)  
{  
    scanf ("%d", &x) ;  
    s = s + x;  
}  
printf ("%d", s) ;
```

# Esempio: sommatoria N numeri (traduz)

1. READ
2. STORE 101
3. LOAD= 0
4. STORE 102
5. LOAD 101
6. BEQ 13
7. SUB= 1
8. STORE 101
9. READ
10. ADD 102
11. STORE 102
12. BR 5
13. LOAD 102
14. WRITE
15. END

# Esempio: sommatoria N numeri (traduz)

1.	READ	
2.	STORE 101	[1,2 → scanf("%d",&n)]
3.	LOAD= 0	
4.	STORE 102	[3,4 → s=0]
5.	LOAD 101	
6.	BEQ 13	[Salta se n=0]
7.	SUB= 1	
8.	STORE 101	[7,8 → n = n-1]
9.	READ	[scanf("%d",&x)]
10.	ADD 102	
11.	STORE 102	[10,11 → s = s+x]
12.	BR 5	
13.	LOAD 102	
14.	WRITE	[printf("%d",s)]
15.	END	

# Esempio: invertire una sequenza

```
n = 0;  
do  
{  
    scanf ("%d", &x[n]);  
    n++;  
} while (x[n-1] != 0);  
  
for (i=n-2; i>=0; i--)  
    printf ("%d", x[i]);
```

# Esempio: invertire una sequenza (traduz)

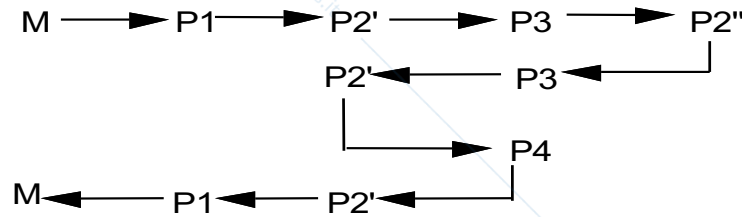
- ❑ Come organizzare la memoria?
  - ▶ Mi serve una cella per memorizzare il contatore degli elementi letti
  - ▶ Ciascun elemento verrà memorizzato all'indirizzo successivo a quello usato per il numero precedente
- ❑ Come posso memorizzare gli elementi della sequenza in un indirizzo sempre diverso?
  - ▶ Uso l'indirizzamento **indiretto**
  - ▶ Avrò bisogno di una cella di memoria che contenga l'indirizzo da utilizzare per l'elemento *corrente*
- ❑ Una possibile soluzione
  - ▶ Indirizzo contatore elementi: 101
  - ▶ Indirizzo elemento corrente: 102
  - ▶ Indirizzi elementi: 103, 104, 105, ...

# Esempio: invertire una sequenza (traduz)

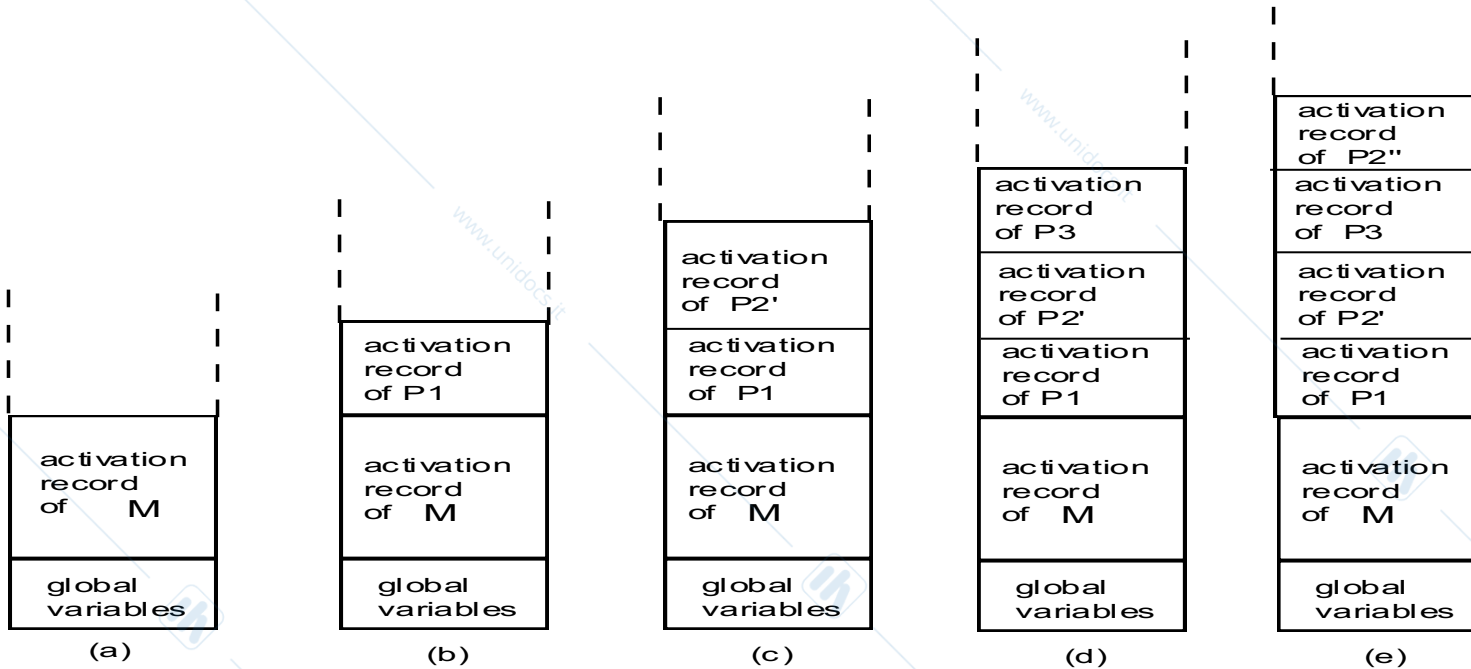
```
1.  LOAD= 0
2.  STORE 101
3.  LOAD= 103
4.  STORE 102
5.  READ
6.  BEQ 15
7.  STORE@ 102
8.  LOAD 102
9.  ADD= 1
10. STORE 102
11. LOAD 101
12. ADD= 1
13. STORE 101
14. BR 5
15. LOAD 101
16. BEQ 25
17. SUB= 1
18. STORE 101
19. LOAD 102
20. SUB= 1
21. STORE 102
22. LOAD@ 102
23. WRITE
24. BR 15
25. END
```

# Gestione della memoria con sottoprogrammi

# La gestione a pila o stack

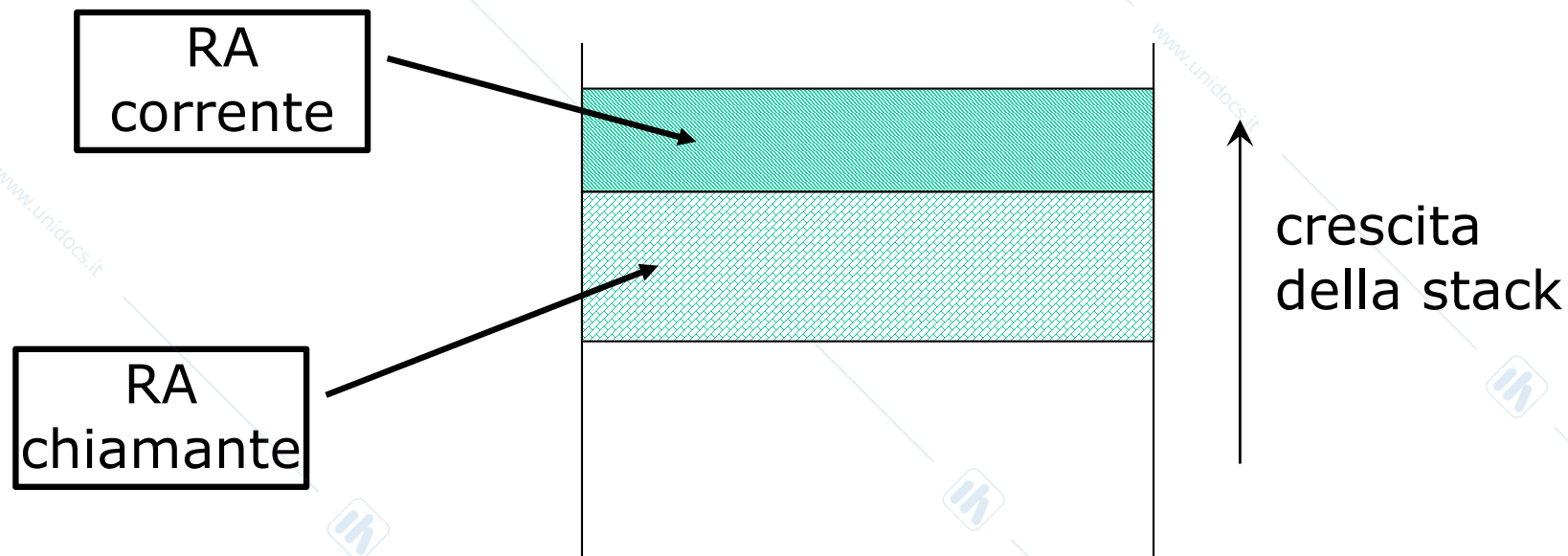


**Legend**  
 A rightwards arrow denotes a call  
 A leftwards arrow denotes the return to the calling subprogram  
 The different activations of P2 are marked by apexes

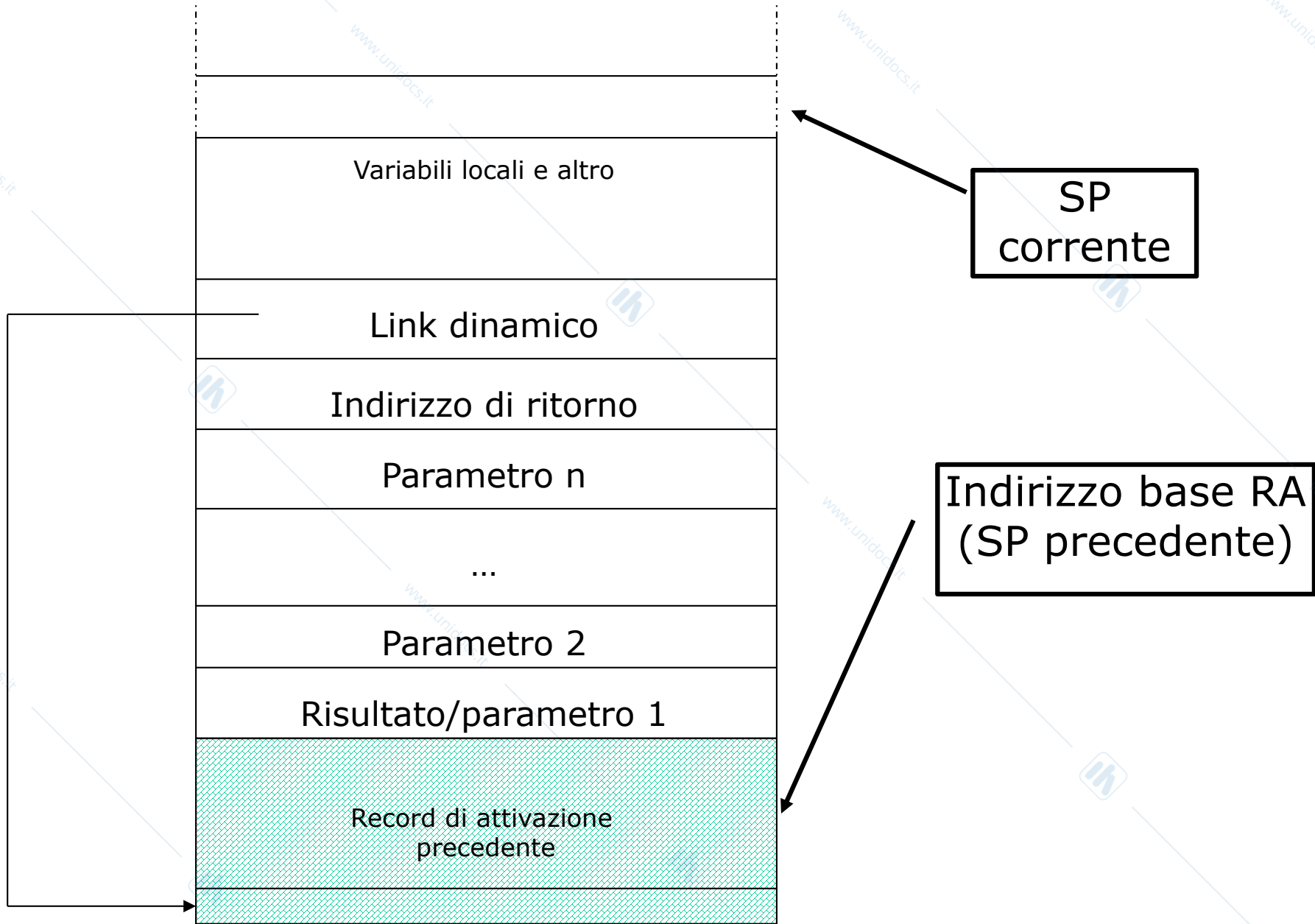


# Dettagli su gestione stack

- Il record di attivazione (RA):
  - ▶ Parametri attuali
  - ▶ Variabili locali
  - ▶ Indirizzo di ritorno (RetAdd)
  - ▶ (Valore precedente dello) **stack pointer (SP)**



# Il record di attivazione



# La chiamata

## Codice del chiamante

- Riserva memoria per il risultato (se previsto)
- Assegna valore ai parametri attuali
- Assegna l'indirizzo di ritorno
- Assegna il link dinamico
- Aggiorna l'indirizzo di base del nuovo record di attivazione
- Assegna il nuovo valore allo SP
- Salta alla prima istruzione del chiamato

## Codice del chiamato

# Il ritorno

## Codice del chiamato

- Riporta il valore di SP al valore precedente
- Riporta il valore dell'indirizzo di base al valore precedente
- Salta all'indirizzo di ritorno

## Codice del chiamante

- Codice della chiamata
- [Indirizzo di ritorno]
- Recupera eventuale risultato



Ma il linguaggio macchina non dovrebbe essere binario?

# Linguaggio assemblativo e linguaggio macchina

- ❑ Quello visto finora è un linguaggio assemblativo (*assembly*), che è un linguaggio macchina *simbolico* per facilitare la lettura e scrittura dei programmi.
- ❑ La sua traduzione in linguaggio macchina (operata da un programma chiamato *assembler*) è molto semplice.
- ❑ Le regole di traduzione richiedono tuttavia diverse scelte progettuali:
  - ▶ ad ogni tipo di istruzione viene associato un codice operativo binario ;
  - ▶ viene definita la dimensione massima degli operandi (che dipende, tra le altre cose, dalla dimensione massima della memoria indirizzabile);
  - ▶ viene quindi definita la dimensione finale che occuperà la traduzione di ogni possibile istruzione.

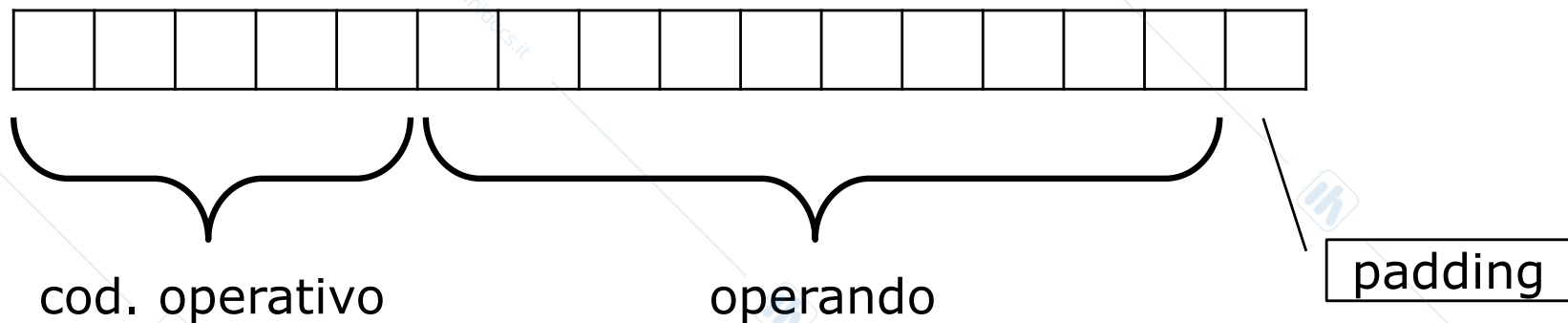
Progettiamo il *nostro* linguaggio macchina

# Codice operativo istruzioni

ISTR	Codice Op	ISTR	Codice Op
READ	00000	DIV	01110
WRITE	00001	LOAD=	01111
BR	00010	STORE=	10000
BEQ	00011	ADD=	10001
BNE	00100	SUB=	10010
BL	00101	MULT=	10011
BLE	00110	DIV=	10100
BG	00111	LOAD@	10101
BGE	01000	STORE@	10110
LOAD	01001	ADD@	10111
STORE	01010	SUB@	11000
ADD	01011	MULT@	11001
SUB	01100	DIV@	11010
MULT	01101	END	11111

# Dimensioni operandi e istruzioni

- ❑ Per semplicità, ipotizziamo che la memoria indirizzabile nel nostro sistema sia  $2^{10}$  celle (o parole).
- ❑ Sempre per semplicità ipotizziamo che anche le costanti numeriche nei programmi siano comprese tra 0 e 1023
- ❑ Tutti gli operandi sono quindi rappresentabili con 10 bit
- ❑ Il codice operativo richiede 5 bit
- ❑ Le istruzioni richiedono 15 bit, ma noi useremo 16 bit



# Esempio

1.	00000000000000000000	READ
2.	0101000011001010	STORE 101
3.	0111100000000000	LOAD= 0
4.	0101000011001100	STORE 102
5.	0100100011001010	LOAD 101
6.	0001100000011010	BEQ 13
7.	1001000000000010	SUB= 1
8.	0101000011001010	STORE 101
9.	0000000000000000	READ
10.	0101100011001100	ADD 102
11.	0101000011001100	STORE 102
12.	000100000001010	BR 5
13.	0100100011001100	LOAD 102
14.	0000100000000000	WRITE
15.	1111100000000000	END