

architettura degli elaboratori lezione 6


Appunti di Davide Scarlata 2024/2025

 **Prof:** Claudio Schifanella

 **Mail:** claudio.schifanella@unito.it

 **Corso:** C

 **Moodle** [Unito](#)

 **Data:** 11/03/2025

Ciclo `while`

```
int v[10], k, i;
while (v[i] == k) {
    ...
    i = i + 1;
}
```

```
LOOP:
    slli x10, x22, 2 # Calcola l'indirizzo della word v[i]
    add x10, x10, x25
    lw x9, 0(x10)    # Carica v[i] in x9
    bne x9, x24, ENDLLOOP # Se v[i] != k, esce dal ciclo
    ...
    addi x22, x22, 1  # Incrementa i
    beq x0, x0, LOOP  # Torna alla valutazione della condizione
ENDLOOP:
```

Salti condizionati e confronto

L'istruzione `slt` scrive 1 in `rd` se `rs1 < rs2`, altrimenti scrive 0.

```
if (i < j) {
    k = 1;
} else {
```

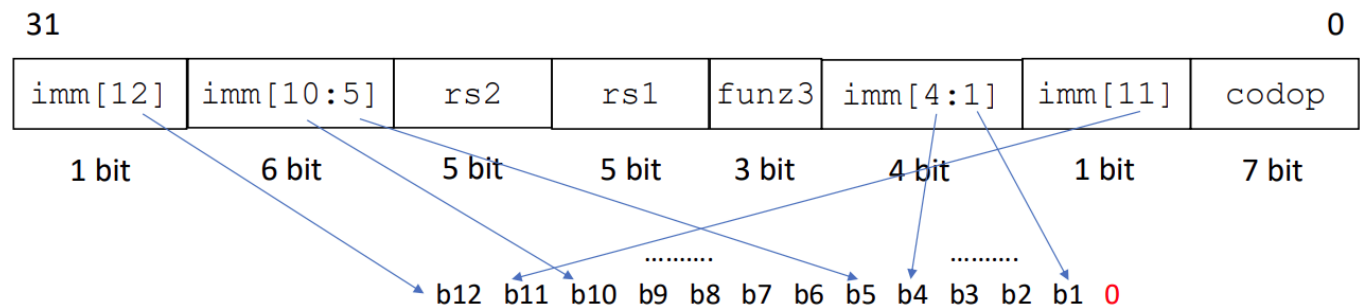
```
k = 0;
}
```

```
slt x21, x19, x20 # x21 = (x19 < x20) ? 1 : 0
```

Possiamo combinare `slt` con `beq` per implementare il salto condizionato.

Salti condizionati e linguaggio macchina

Le istruzioni di salto condizionato utilizzano il formato **SB**. Questo formato può rappresentare indirizzi di salto da -4096 a 4094 (multipli di 2).



Pseudoistruzioni

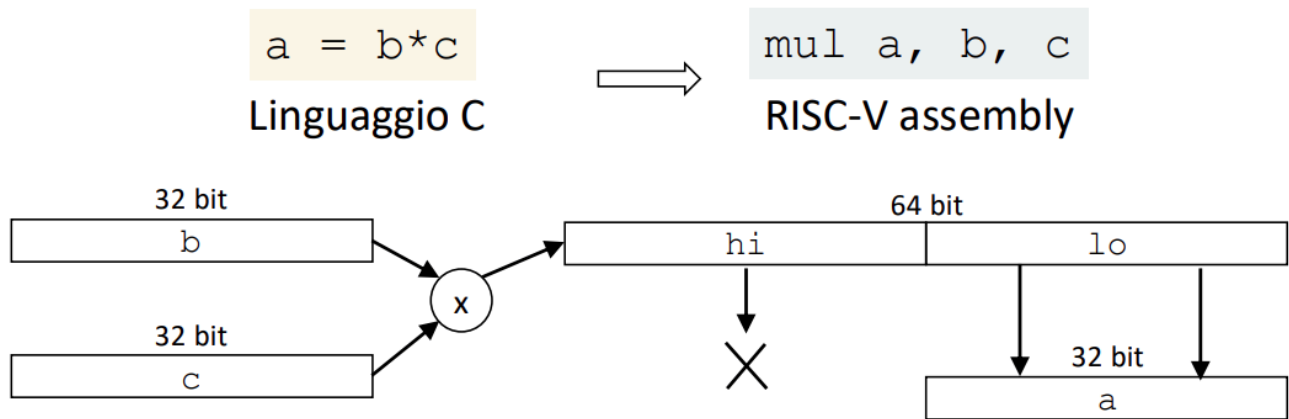
Le pseudoistruzioni esistono solo in assembly e vengono tradotte in istruzioni reali dall'assemblatore.

• Esempio

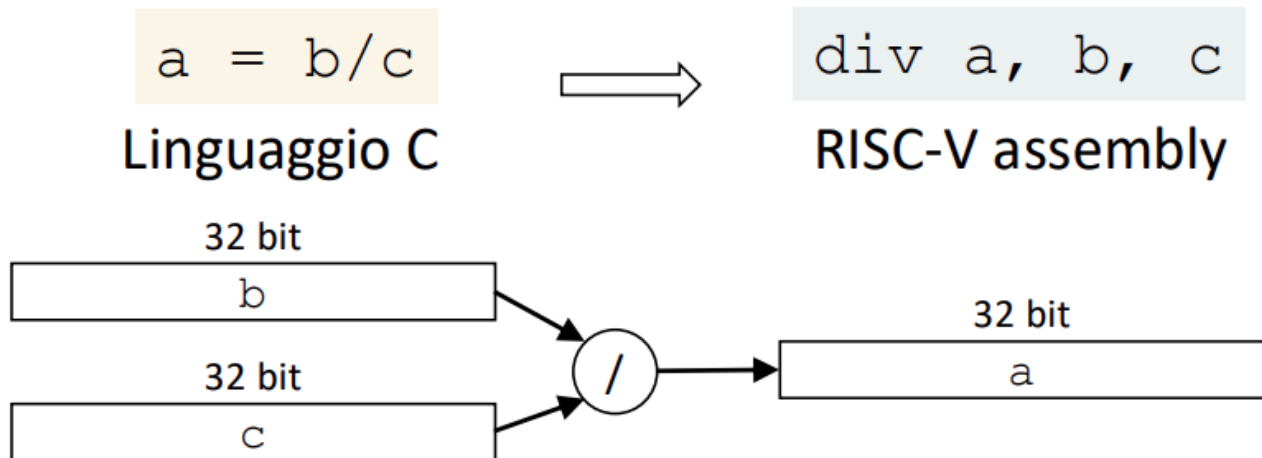
```
mv x5, x6 → addi x5, x6, 0
```

```
not x5, x6 → xori x5, x6, -1
```

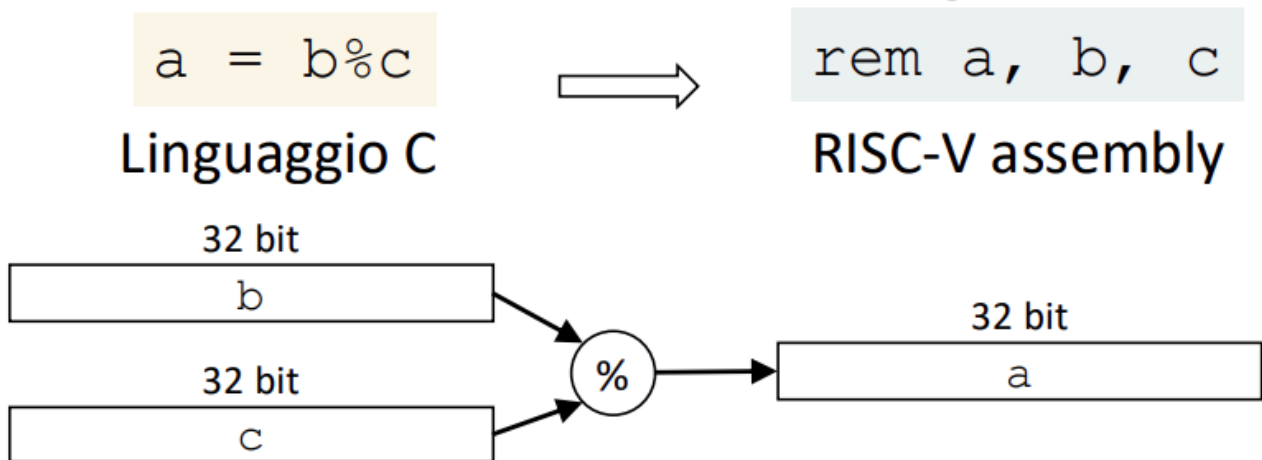
Istruzioni aritmetiche: moltiplicazione



Istruzioni aritmetiche: divisione e resto



- Ci sono le versioni unsigned



Procedure

Definizione

Le procedure sono blocchi di codice riutilizzabili che eseguono un compito specifico.

Vantaggi:

- Astrazione
- Riutilizzabilità del codice
- Maggiore organizzazione
- Testing più agevole

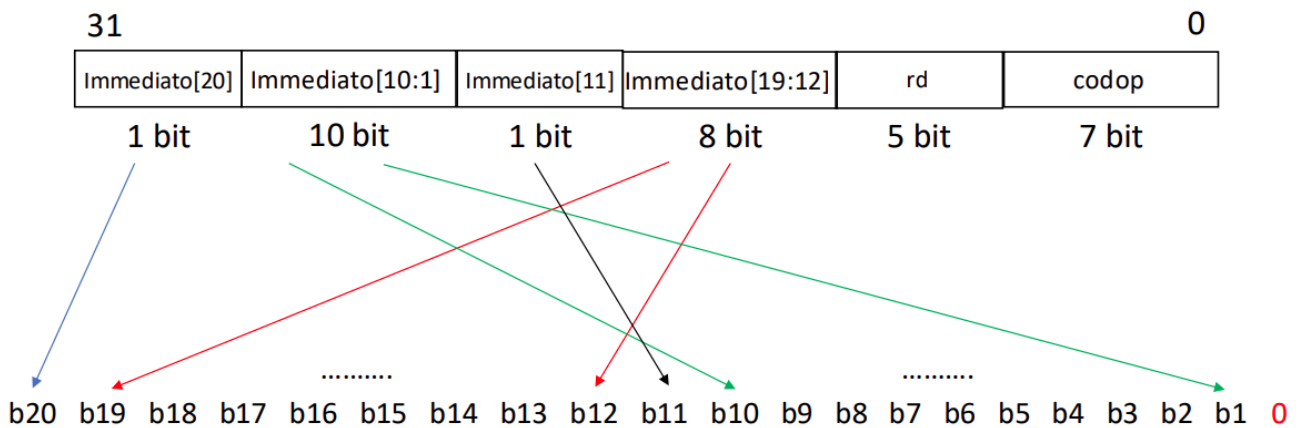
Chiamante e chiamata

```
int somma(int x, int y) {  
    int rst;  
    rst = x + y + 2;  
    return rst;  
}
```

```
jal IndirizzoProcedura    # Jump and Link
```

- Salta all'indirizzo della procedura
- Memorizza il valore dell'istruzione successiva in `x1`

- Viene introdotto un nuovo tipo: J



Ritorno al chiamante

```
jalr rd, offset(rs1)    # Salto a un indirizzo qualsiasi
```

Side Effects – Sovrascrittura dei registri

```
int somma(int x, int y) {
    int rst = x + y + 2;
    return rst;
}
```

```
SOMMA: add x5, x10, x11
       addi x20, x5, 2
       jalr x0, 0(x1)    # Ritorna al chiamante
```

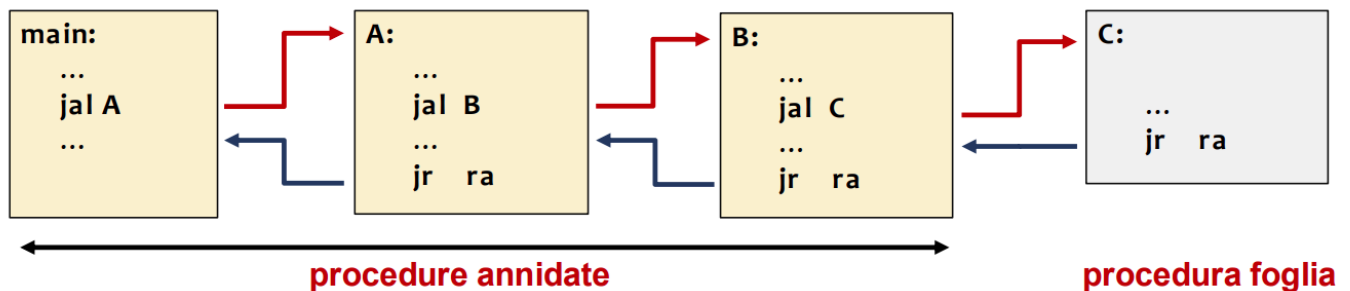
Se un registro contiene un valore usato dalla procedura chiamante, occorre salvarlo in memoria prima di utilizzarlo.

- **Problema**

- Nel caso di procedure annidate, il return address (x1 o ra) viene sovrascritto

- **Soluzione:**

- La procedura chiamata, deve **salvare in memoria** il valore di x1 **prima** di chiamare la procedura annidata con l'istruzione jal



Lo Stack

- Lo stack segue la logica **LIFO (Last In First Out)**.
- Il **Stack Pointer (SP)** indica l'ultima cella occupata nello stack (registro x2).

PUSH

```
addi sp, sp, -4    # Decrementa SP
sw x20, 0(sp)      # Salva in memoria
```

POP

```
lw x20, 0(sp)    # Carica il valore  
addi sp, sp, 4    # Incrementa SP
```
