Salti incondizionati

- Jump and link register
  - jal e jalr
- Costrutto case/switch

Pseudoistruzioni

- Il linguaggio assembler fornisce “pseudoistruzioni”
  - Una pseudoistruzione esiste solo in linguaggio assembler e non in linguaggio macchina
  - l’assemblatore traduce le pseudoistruzioni nel linguaggio macchina delle corrispondenti istruzioni
  - Quando si effettua per es. il debugging è necessario ricordare quali siano le istruzioni reali
  - L’elenco delle pseudoistruzioni è presente nel manuale di riferimento
- Esempio

```
mv x5,x6   ⟶  addi x5,x6,0

not x5,x6  ⟶  xori x5,x6,-1
```

Aritmetica con segno e senza segno

- Le istruzioni aritmetiche fin qui esaminate operano su interi con segno
  - Gli operandi (a 64 bit, nei registri) sono rappresentati in complemento a due: i valori vanno da -2^{63} a $2^{63}-1$
  - Anche i byte-offset di `ld` e `sd` e il numero di istruzioni di cui saltare nella `bne`/`beq` sono interi con segno da -2^{11} a $2^{11}-1$
- È possibile utilizzare anche operandi senza segno
  - In tal caso i valori vanno da 0 a $2^{64}-1$
  - Invece di `slt`, `slti` si useranno `sltu`, `sltiu`
    - `slt` e `sltu` forniranno un risultato diverso se un operando è negativo (es. -4 < 3 a 1 con `slt`, ma 0 con `sltu`)
      - Infatti il -4 verrebbe interpretato come un numero molto grande (es. 7 se gli operandi fossero solo di 3 bit), quindi 7 < 3 risulterebbe falso

Istruzioni aritmetiche: moltiplicazione

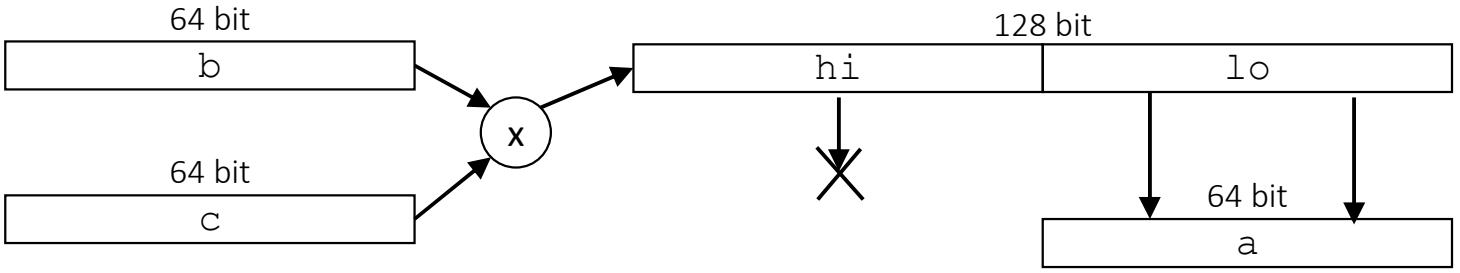
1

```
a = b*c
```

Linguaggio C

```
mul a, b, c
```

RISC-V assembler



- Viene utilizzato un registro interno nascosto all'utente
- I 64 bit più significativi possono essere ottenuti con l'istruzione

```
mulh a, b, c
```

2

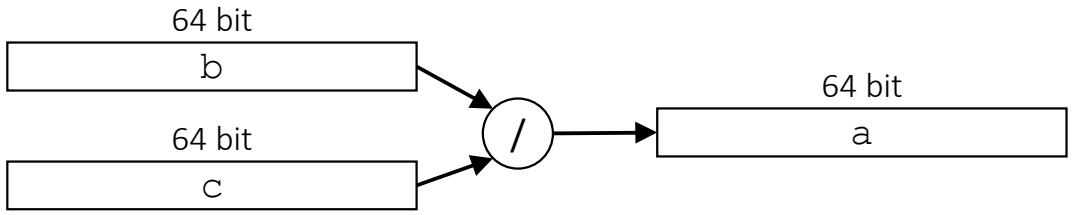
Istruzioni aritmetiche: divisione e resto

```
a = b/c
```

Linguaggio C

```
div a, b, c
```

RISC-V assembler

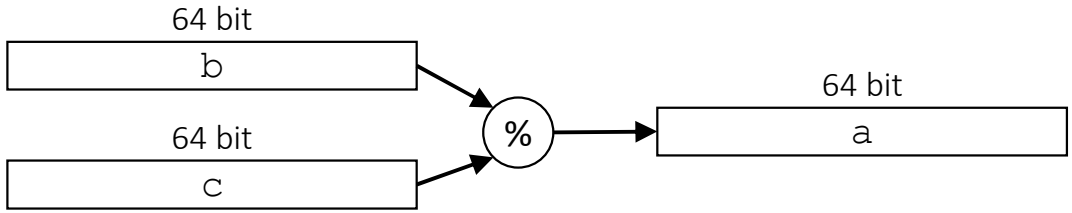



```
a = b%c
```

Linguaggio C

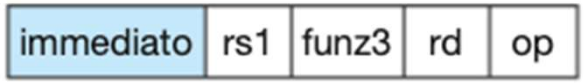
```
rem a, b, c
```

RISC-V assembler



Modalità di indirizzamento

- Indirizzamento immediato

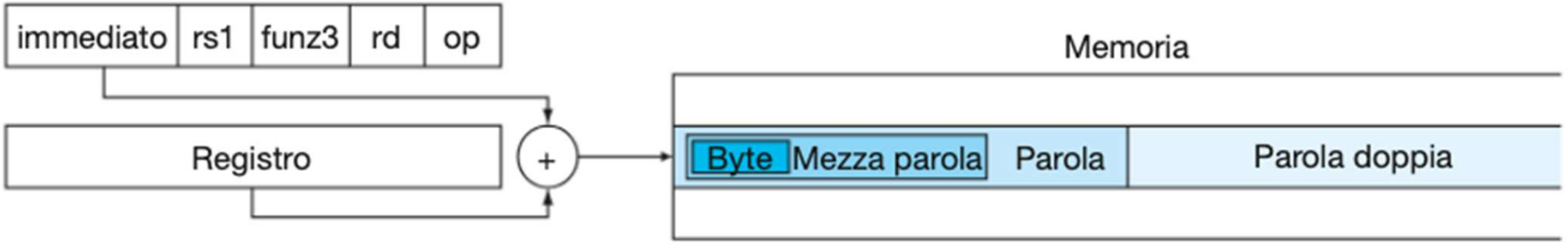


- Indirizzamento tramite registro

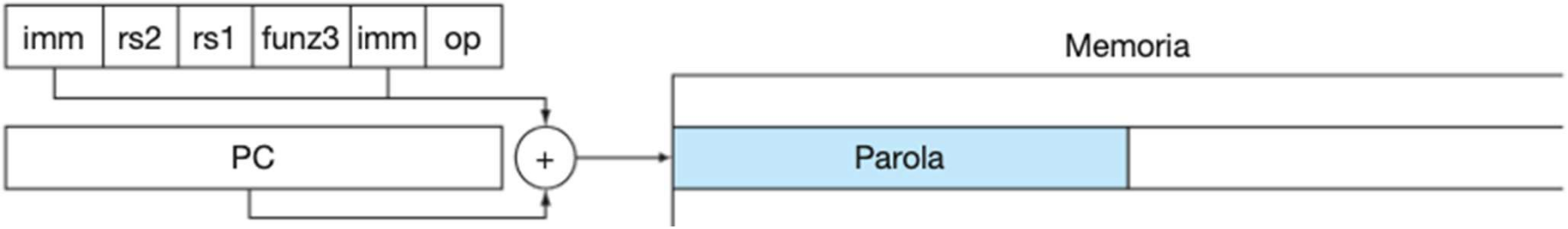
Modalità di indirizzamento

offset

- Indirizzamento tramite base e spiazzamento



- Indirizzamento relativo al program counter



Le procedure

- Porzioni di codice associati ad un **nome** che possono essere invocati più volte e che eseguono un compito specifico, avendo come input una lista di **parametri** e come output un **valore di ritorno**
- Vantaggi
  - Astrazione
  - Riusabilità del codice (librerie)
  - Maggiore organizzazione del codice
  - Testing più agevole

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

es in C/Java

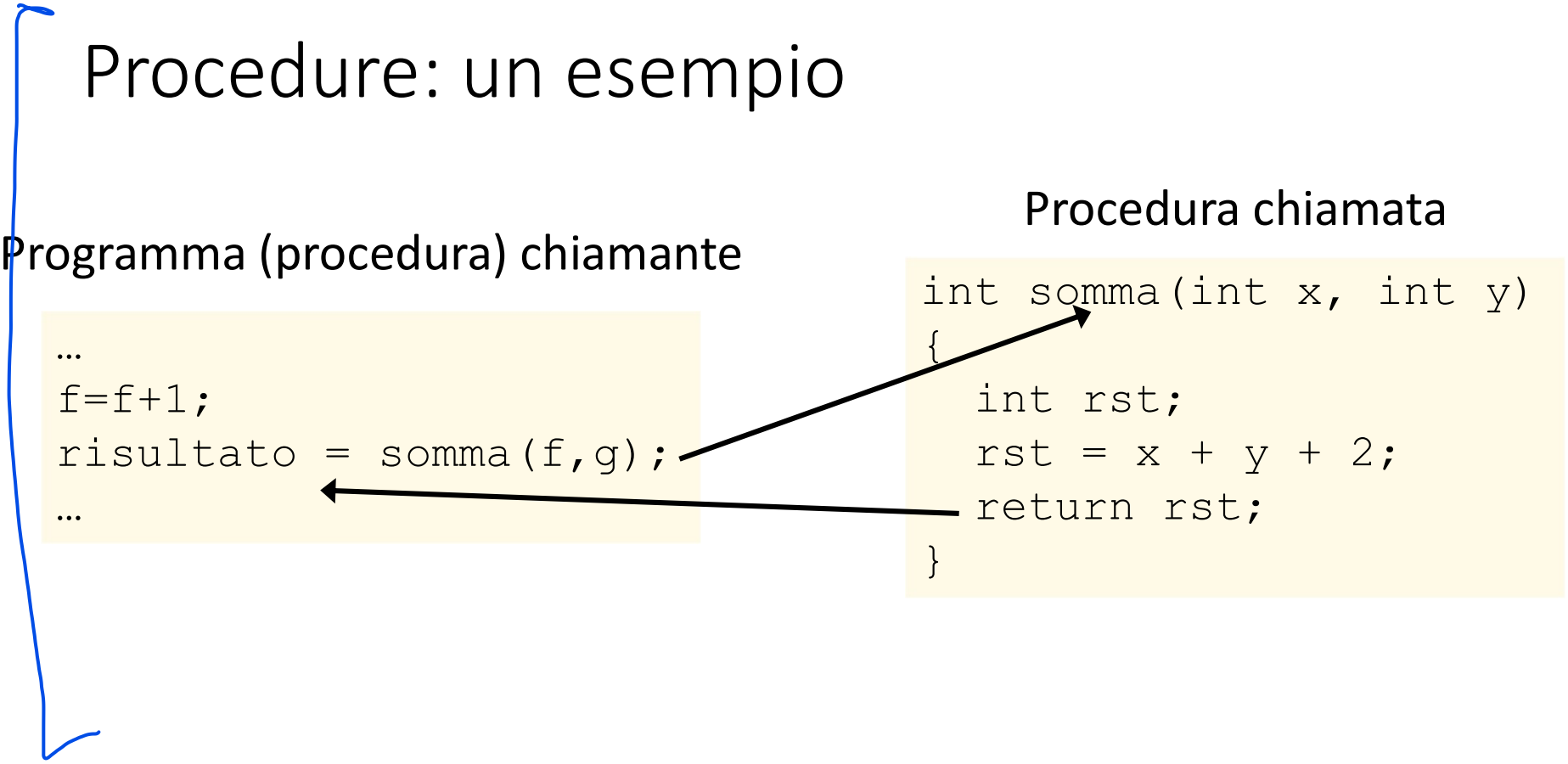
Procedure: un esempio

Programma (procedura) chiamante

```
…
f=f+1;
risultato = somma(f,g);
…
```

Procedura chiamata

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



Le procedure: passi da seguire

- Chiamante
  - Mettere i parametri in un luogo accessibile alla procedura
  - Trasferire il controllo alla procedura
- Chiamato
  - Acquisire le risorse necessarie per l'esecuzione della procedura
  - Eseguire il compito richiesto
  - Mettere il risultato in un luogo accessibile al programma chiamante
  - Restituire il controllo al punto di origine (la stessa procedura può essere chiamata in differenti punti di un programma)

Modifica del flusso di programma: invocazione

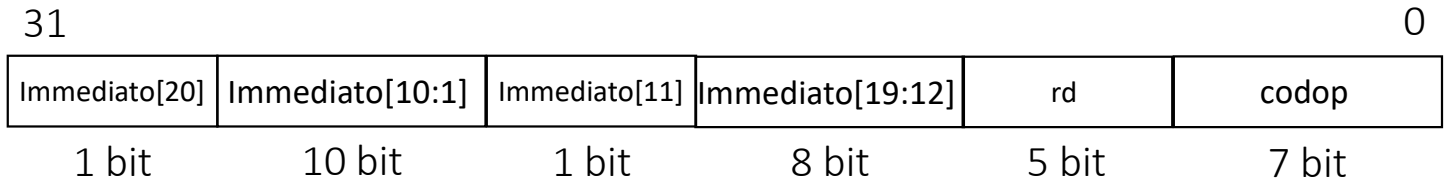
`jal IndirizzoProcedura` Jump-and-link

- Salta all'indirizzo (offset) con etichetta `IndirizzoProcedura`
- Memorizza il valore dell'istruzione successiva `PC+4` nel registro `x1` (return address)
- Pseudo-istruzione, abbreviazione di
  - `jal x1, IndirizzoProcedura`

vedi in 06

JAL e linguaggio macchina

- Viene introdotto un nuovo tipo: J



Salti con offset più grandi

- Come per le costanti, anche i salti possono essere eseguiti anche ad istruzioni più lontane
- RISC-V introduce la possibilità di salto in un intervallo pari a 2^{32}
- Nuova istruzione

```
auipc rd offset
```

Add Upper Immediate PC

Tipo U in linguaggio macchina

  - Inserisce nel registro `rd` l'indirizzo di PC + (offset << 12)

Salti con offset più grandi

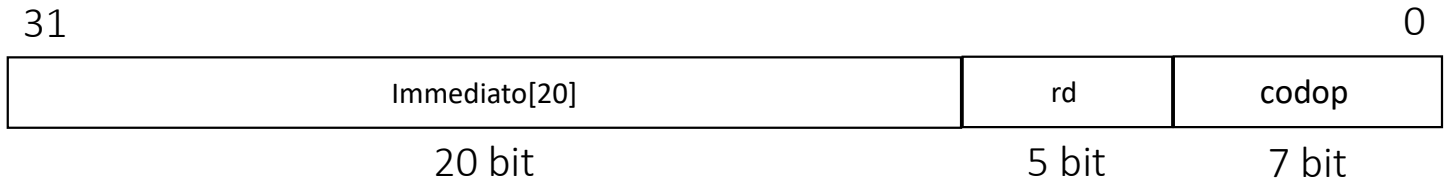
- Se usiamo `auipc` con i 20 bit più significativi dell'offset, allora possiamo aggiungere questa istruzione una istruzione che calcola
  - PC = rd + offset[]
- Otteniamo come risultato un salto incondizionato con offset più esteso
- L'istruzione da considerare è `jalr`! Ricapitolando:

```
auipc rd offset[31..12]
jalr x0, offset[0..11](rd)
```

- Realizza un salto incondizionato a PC+offset[31..0]

AUIPC e linguaggio macchina

- Viene usato il tipo U



Modifica del flusso di programma: ritorno al chiamante

```
jalr rd, offset(rs1)
```

- Salta all'indirizzo (offset) con etichetta `IndirizzoProcedura`
  - `x[rd]=PC+4; PC=x[rs1]+sext(offset)&~1`

Segno esteso a 64bit

Bit 0 azzerato

- La procedura chiamata come ultima istruzione esegue

```
jalr x0, 0(x1)
```

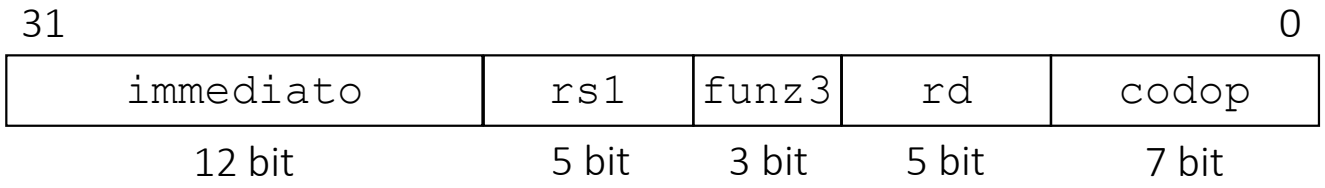
Tipo I in linguaggio macchina

```
x0=PC+4; PC=x1+0
```

Operazione nulla

Return Address

Formato di tipo I (immediato)



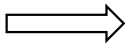
- Permette di codificare anche `jalr`
- Il campo immediato
  - Rappresentato in complemento a due
  - Valori possibili: da `-2048` a `+2047`

Side effect – sovrascrittura dei registri (1)

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

….
f=f+1
risultato=somma(f,g)
…
```

x → x10
y → x11
rst → x20
f → x6

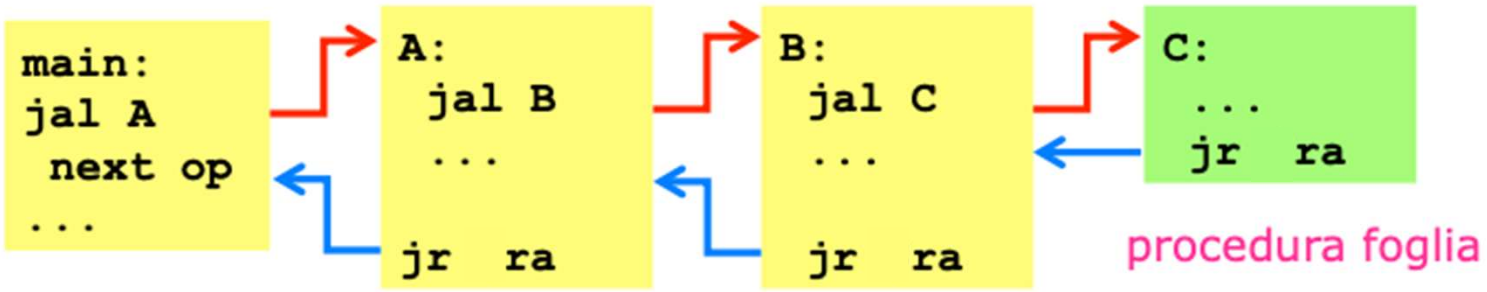


```
SOMMA: add x5,x10,x20
       addi x20,x5,2
       jalr x0,0(x1)
…..
…..
       addi x6,x6,1
       jal SOMMA
```

- **Problema**: che cosa succede se il registro x5 contiene un valore usato dalla procedura chiamante?
- **Soluzione**: nella procedura, **salvare** il valore di x5 **in memoria** prima di utilizzarlo (e ripristinarlo prima del ritorno al chiamante)

Side effect – procedure annidate (2)

- Problema
  - Nel caso di procedure annidate, il return address (`x1`) viene sovrascritto
- Soluzione:
  - La procedura chiamata, deve **salvare in memoria** il valore di `x1` prima di chiamare la procedura annidata con l'istruzione `jal`



Side effect – parametri numerosi (3)

- Problema:
  - che cosa succede se i parametri e le variabili di una procedura superano il numero di registri disponibili?
- Soluzione:
  - **Salvare** temporaneamente i dati **in memoria** per caricarli nei registri prima del loro utilizzo all'interno della procedura