

# Architettura degli elaboratori - lezione 4

## Appunti di Davide Scarlata 2024/2025

 **Prof:** Claudio Schifanella

 **Mail:** [claudio.schifanella@unito.it](mailto:claudio.schifanella@unito.it)

 **Corso:** C

 **Moodle** [Unito](#)

 **Data:** 26/02/2025

---

## Load Halfword (LH) e Load Halfword Unsigned (LHU)

- `lhu` ❌ non esegue l'estensione del segno, mentre `lh` ✅ sì.

### Esempio:

```
lh x5, 8(x21)
```

➡ Per `x5`, 2 byte saranno occupati dal numero e 2 dal segno ( $-2^{15}$ ,  $2^{15}-1$ ).

```
lhu x5, 8(x21)
```

➡ Per `x5`, 2 byte (**16 bit**) composti da **0** a ( $2^{16}-1$ ).

⚠ **Non esiste la `lwu` in architettura 32-bit** perché quando carico un numero lo carico già con il segno corretto.

⚠ **Non abbiamo `shu` e `sbu`** perché un byte viene preso dal registro e memorizzato così com'è.

---

## Restrizioni sull'allineamento degli indirizzi

 La memoria è classicamente "indirizzata al byte".

 Le istruzioni di **load** e **store** usano l'indirizzo di un byte.

✓ Tuttavia:

- `lw`, `lwu`, `sw` 📦 **trasferiscono 32 bit.**
- `lh`, `lhu`, `sh` 📦 **trasferiscono 16 bit.**
- Solo `lb`, `lbu`, `sb` 📦 **trasferiscono 8 bit.**

♦ **Allineamento consigliato:**

- Per `lw`, `lwu`, `sw` ➡ multiplo di 4.
- Per `lh`, `lhu`, `sh` ➡ multiplo di 2.

⚠ **Nota:** Se si specifica un indirizzo non allineato, il RISC-V impiegherà più tempo per accedere al dato.

---

## 🎯 Operandi immediati e costanti

L'operazione `b = b + 5` in assembly:

```
lw x9, indirizzoCostante5(x3)
add x5, x5, x9
```

✗ Questa operazione è **lenta**. Possiamo usare invece:

```
addi x5, x5, 5
```

- ✓ `addi` (**add immediate**) permette di sommare un valore immediato.
- ✓ Il valore immediato può essere compreso tra **-2048 e +2047**.
- ✗ **Nota:** La sottrazione immediata non esiste, si usano valori negativi.






---

## 🏗 Il linguaggio macchina

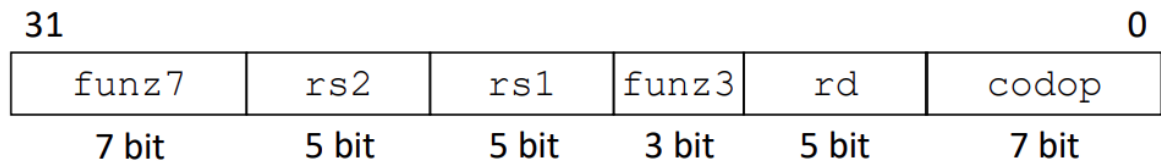
- 💻 Ogni istruzione RISC-V richiede esattamente **32 bit**.
- 💻 RISC-V definisce diversi formati di istruzioni per codificare in binario ogni operazione assembly.
- 💻 Una sequenza di istruzioni in linguaggio macchina è chiamata **codice macchina**.

♦ **Struttura di un'istruzione RISC-V**

Esempio: `add x5, x6, x21`

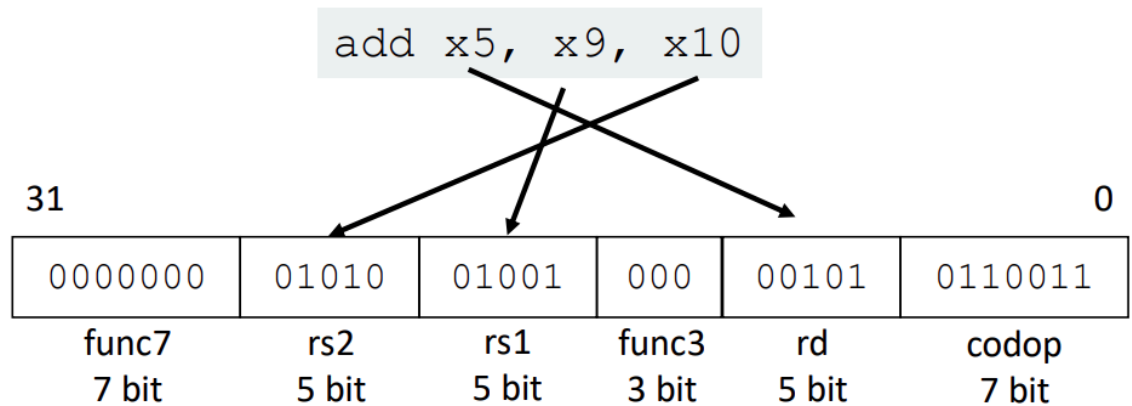
-  **Opcode:** identifica il tipo di istruzione.
-  **rd:** registro di destinazione.
-  **rs1:** primo operando sorgente.
-  **rs2:** secondo operando sorgente.
-  **funct3, funct7:** codici operativi aggiuntivi.

## Formato di tipo R (registro)

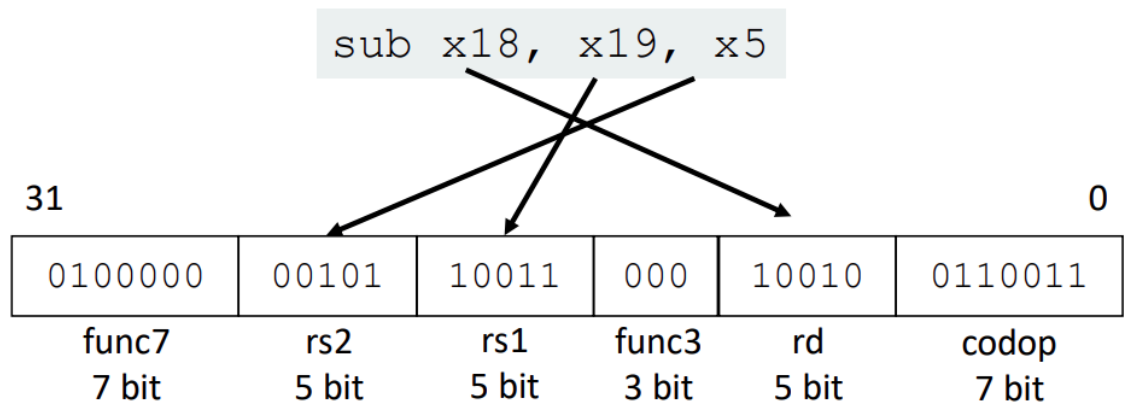


## Formato di tipo R (registro)

### • Esempio



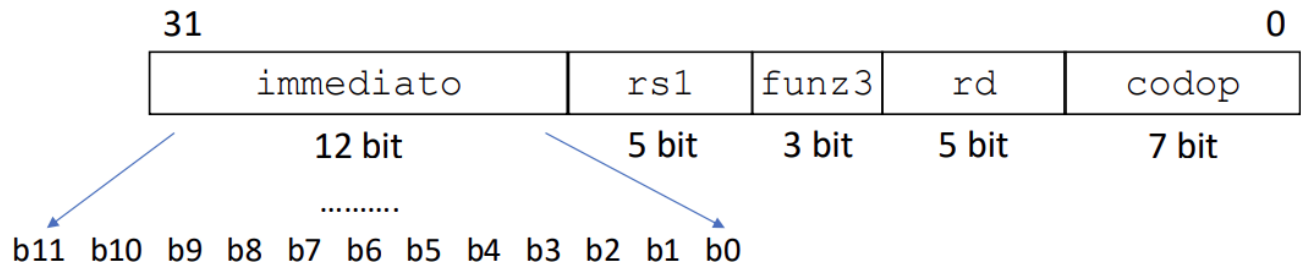
### • Esempio



⚠ **Nota:** La differenza tra `add` e `sub` sta nel valore di `funct7`.

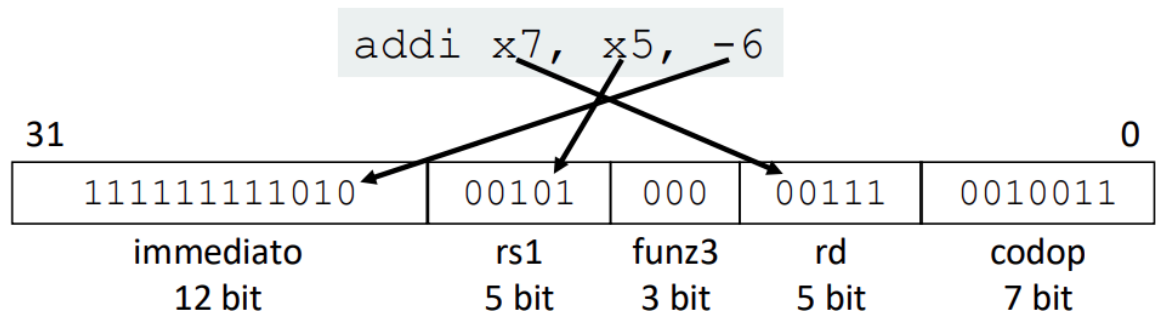
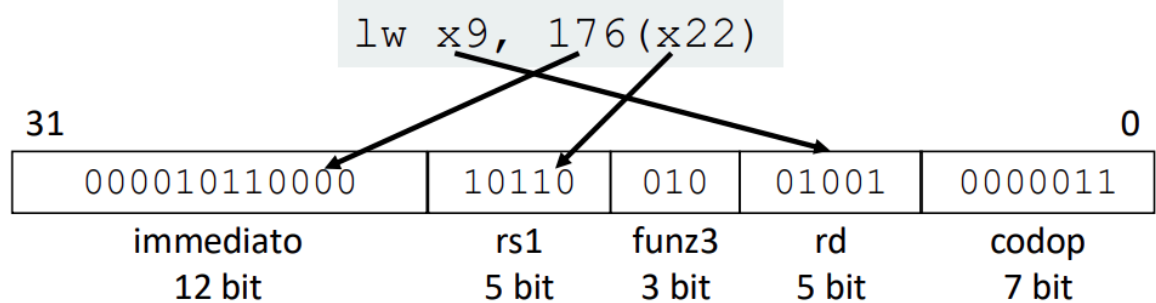
## Formato immediato

# Formato di tipo I (immediato)



- ✓ Permette di codificare le istruzioni che richiedono il caricamento dalla memoria o una costante ( `load` , `addi` , `andi` , `ori` , ... ).
- ✓ Sono presenti **12 bit** perché con **5 bit** l'intervallo di rappresentazione per costanti e offset sarebbe troppo ridotto.
- ✓ **Campo immediato**: rappresentato in complemento a due (**-2048 a +2047**).

## • Esempi

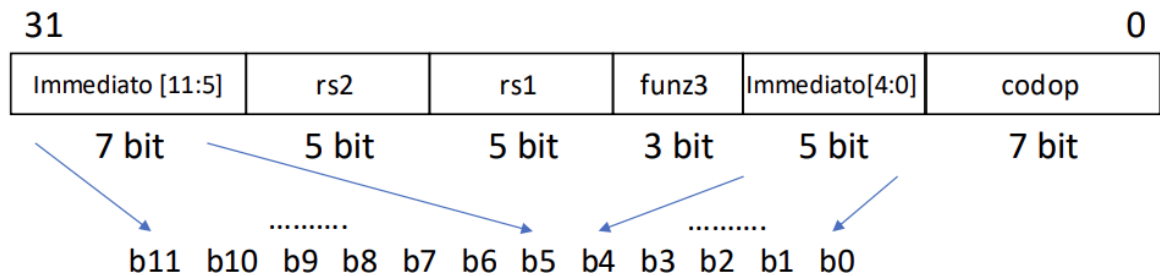


## Store Word ( `sw` ) e Formati di Istruzioni

Esempio: `sw x5, 8(x20)`

- ✓ Non utilizza il formato I, ma il formato S.
- ✓ Il campo immediato è diviso in due parti per semplificare il circuito hardware.
- ✓ Intervallo del campo immediato: da -2048 a +2047.

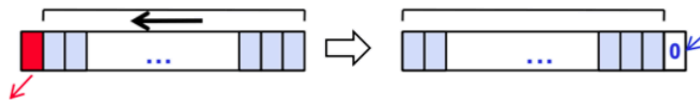
# Formato di tipo S



## 🔧 Operazioni Logiche

### • Shift logico

#### • A sinistra



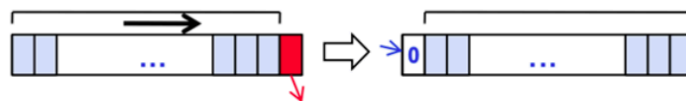
Shift Left Logical

```
sll x9, x22, x19
x9 = x22 << x19
```

```
slli x9, x22, 5
x9 = x22 << 5
```

Shift Left Logical Immediate

#### • A destra



Shift Right Logical

```
srl x9, x22, x19
x9 = x22 >> x19
```

```
srli x9, x22, 5
x9 = x22 >> 5
```

Shift Right Logical Immediate<sub>73</sub>

RISC-V Instruction Set

📄 Le istruzioni `slli` e `srli` utilizzano una costante di **5 bit**.

### • Shift aritmetico

#### • A destra

##### • Positivo



##### • Negativo



Shift Right Arithmetic

```
sra x9, x22, x19
x9 = x22 >> x19
```

```
srai x9, x22, 5
x9 = x22 >> 5
```

Shift Right Arithmetic Immediate

#### • A sinistra

- Non esiste perché non ha senso: identico a `sll`



## Esempio in linguaggio C

```
v[i] = v[j];
```



Traduzione in Assembly:

```
addi x6, x21, 0    # Calcola indice in x6
slli x6, x6, 2      # Moltiplica per dim in byte
add x6, x6, x19     # Somma l'indirizzo base
lw x6, 0(x6)

addi x7, x9, 0      # Calcola indice in x7
slli x7, x7, 2      # Moltiplica per dim in byte
add x7, x7, x19     # Somma l'indirizzo base
sw x6, 0(x7)
```



**Nota:** x7 contiene l'indirizzo del primo byte che rappresenta v[i] x6 contiene l'indirizzo del primo byte che rappresenta v[j] .

uso lo shift logico perchè l'operazione di moltiplicazione non è detto che ci sia in architetture più elementari

laboratorio 1:

qual'è la codifica in binario dell'istruzione mv?

quello che decide l'assemblatore (è una pseudo istruzione )