Architettura degli elaboratori - lezione 4

Appunti di Davide Vella 2024/2025

Claudio Schifanella
claudio.schifanella@unito.it
Link al moodle :
https://informatica.i-learn.unito.it/course/view.php?id=3106

26/02/2025

Contenuti

Note sulla scorsa lezione

- lh : load halfword, normale, legge il segno
- lhu : load halfword unsigned, non conta l'estensione del segno ()

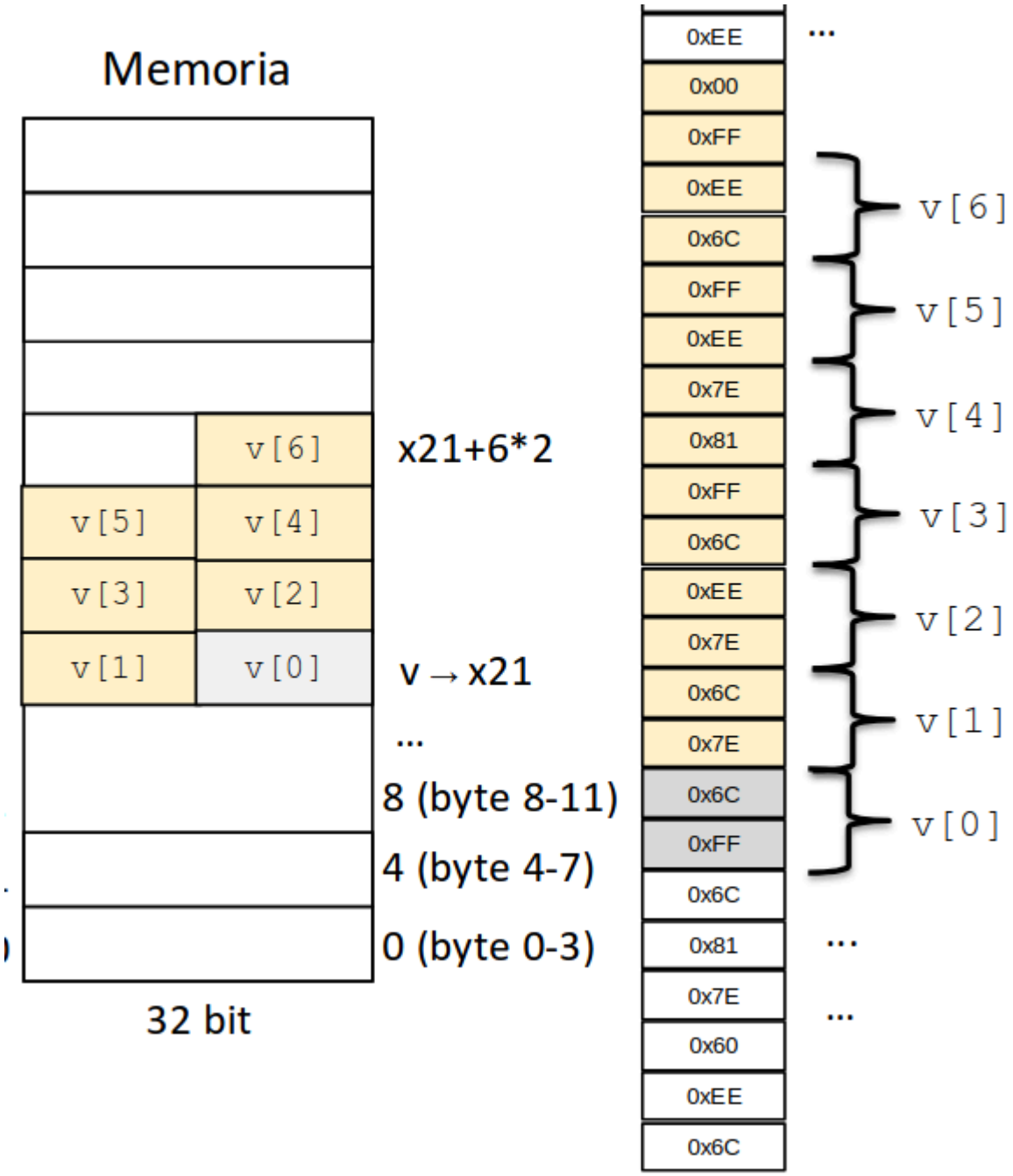
Allineamento degli indirizzi

La memroia è classicamente "indirizzata al byte", quindi le istruzioni di load e store usano l'indirizzo di byte, però :

- lw, lwu (64 bit), sw trasferiscono 32 bit
- lh, lhu, sh trasferiscono 16 bit
- lb, lbu, sb trasferiscono 8 bit
  È conveniente pertanto che l'indirizzo sia opportunamente allineato :
- per lw, lwu, sw dovrebbe essere allineanto ad un multiplo di 4
- per lh, lhu, sh dovrebbe essere allineanto ad un multiplo di 2
  Esempio con variabili a 16 bit (short)

```
short g, f, h;
short v[10];
g = h + v[3];
v[6] = g - f;
```

```
lh x10, 6(x21)
add x5, x9, x10
sub x6, x5, x19
sh x6, 12(x21)
```



Operandi immediati e costanti

Ci si ritrova molto spesso ad dover sommare una costante a qualcosa, si usa quindi l'istruzione "addi" :

Example

```
c = c + 4;
```

```
addi, x5, x5, 4
```

Il valore della costante può variare da -2048/+2047. Non esiste "subi" per la sottrazione, quindi si usa la "addi" con la costante negativa :

Example

```
c = c - 4;
```

```
addi, x5, x5, -4
```

Linguaggio macchina

Ogni istruzione RISC-V richiede esattamente 32 bit per la sua reppresentazione in linguaggio macchina (si 32 bit, che 64). RISC-V definisce diversi formati di istruzione che consentono di codificare in binario ogni istruzione assembly. Abbiamo un codice operazione per riferirci all'istruzione che vogliamo usare che occupa 7 bit. Ogni registro occupa 5 bit (i registri sono 32, per rappresentare 32 bastano 5 bit ($2^5 = 32$)).

Formato di tipo R (registro)

Example

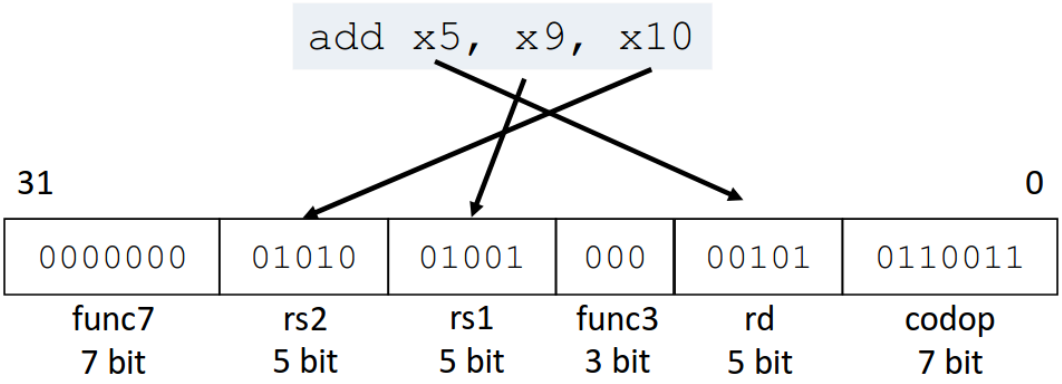
In 32 bit :

```
add x4, x6, x21
```

Da qui vediamo che "add" occupa 7 bit, x4 5 bit, x6 5 bit, x21 5 bit, in totale usiamo 22 bit su 32 disponibili.

Formato di tipo R (registro)

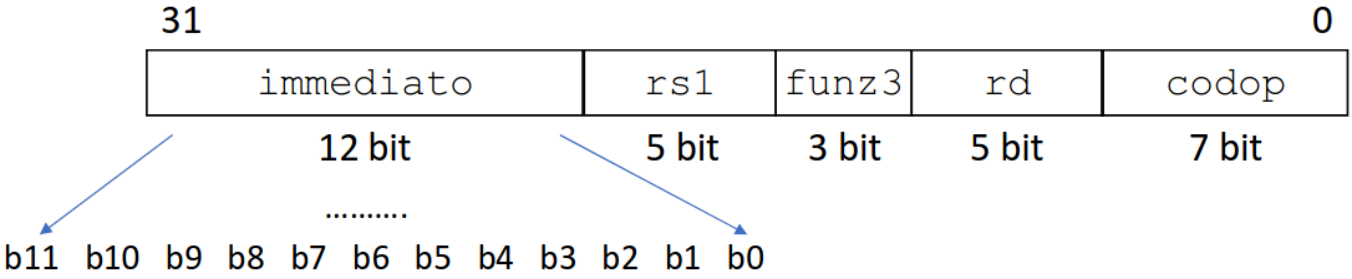
- Esempio



func7 e func3 sono codici operativi aggiuntivi utili in alcuni casi

Formato di tipo I (immediato)

Ci riferiamo alle istruzioni load, addi, andi, ori, ... Queste istruzioni (avendo la possibilità di usare le costanti) occupano 12 bit in più in quei 32 disponibili, di seguito lo schema :



Dove rsl e rd sono i due registri usati nell'istruzione.

Example

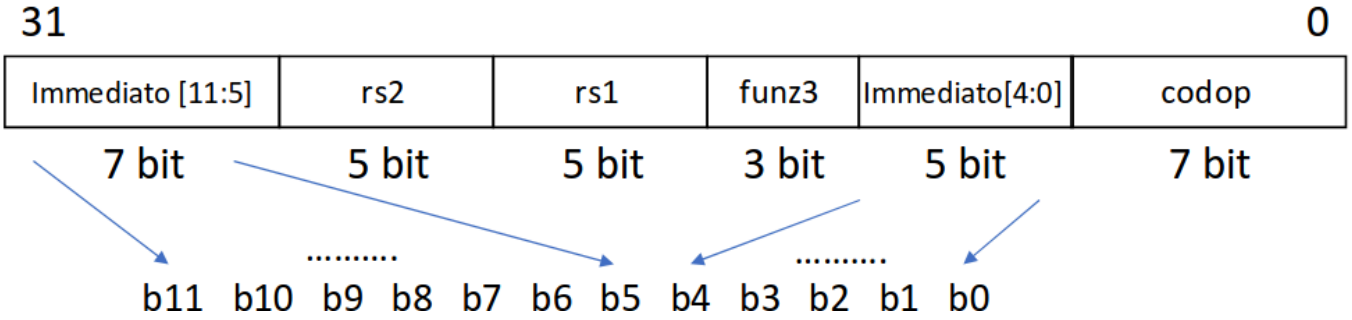
```
load x5, 12(x21)
```

Qui :

- x5 = rd (register destination)
- x21 = rs1 (register source 1)
- immediato = 12
- load = codop

Formato di tipo S

Permette di codificare le istruzioni che richiedono il salvataggio in memoria o una costante, come "store" (sw, sh, shu, ...).



Il motivo per cui il campo "immediato" sono divisi (una da 7 bit e uno da 5) è il seguente : le istruzioni "store" non hanno un output, hanno solo input. Per cercare di mantenere uno standard, **che aiuta la costruzione fisica del circuito** (importante), dove a destra di funz3 ci sono gli output (in questo caso appunto non ci sono ed è occupato da immediato) e a sinistra troviamo gli input (che sono due, quindi prende 5 bit da immediato, ma quei 5 bit vengono ripresi dal altro immediato).
Dato che l'immediato in realtà occupa sempre 12 bit, i valori possibili vanno sempre da -2048/+2047.

Operazioni logiche

Shift logico

sll : shift left logical

```
sll x9, x22, x19
```

Qui :

- x9 : destinazione dello shift, dove salviamo il risultato
- x22 : il registro che shiftiamo
- x19 : il valore per cui shiftiamo
- $x9 = x22 << x19$

slli : shift left logical immediate

```
sll x9, x22, 5
```

- $x9 = x22 << 5$

sra : shift right logical

```
sra x9, x22, x19
```

- $x9 = x22 >> x19$

srai : shift right logical immediate

```
sra x9, x22, 5
```

- $x9 = x22 >> 5$

Formato dei registri

Queste funzioni dette sopra, usato tutte il formato registro R.

Esempio

Passa da C a assembly :

```c
int i, j;
int v[10];
v[i] = v[j];
```

i = x9
j = x21
v = x19

```
addi x6,x21,0
slli x6,x6,2 //x6 ora contiene l'indirizzo del primo byte che rapp. v[j]
//sopra, stiamo moltiplicando per 4, perché un int occupa 4 byte. Stiamo
//calcolando l'offset
add x6,x6,x19 //arriviamo al punto v[j]
lw x6,0(x6) //prendiamo il valore di v[j]
addi x7,x9,0
slli x7,x7,2 //idem sopra per x6, ma di v[i]
add x7,x7,x19 //arriviamo al punto di v[i]
sw x6,0(x7) //scriviamo il valore v[j] in v[i]
```

Nota

Per maggiori approfondimenti (magari su registri, formati, codop...), si può andare a vedere il manuale di riferimento di RISC-V, ottenibile anche sul libro del corso oppure online.

Domande poste durante la lezione (forse utili per esame)

- Perché non esiste lwu (load word unsigned) (nelle architetture a 32 bit, nella 64 esiste)?
- Perchè non esiste shu (store halfword unsigned)?
- Perché non usiamo la moltiplicazione anzichè lo shift nell'ultimo esempio? Perché non è detto che l'istruzione di moltiplicazione sia nel set di istruzioni base supportate dall'hardware.