6 - Componenti dei computer e tipologie di istruzioni

DATAPATH E FLUSSO DI DATI

📖 Il **datapath** è l'insieme delle componenti della **CPU** che svolgono operazioni aritmetiche.

I dati in input vengono immessi dalle **periferiche d'entrata**, mentre gli output sono mostrati dalle **periferiche di uscita**.

Il **datapath** si compone, in particolare, di ALU, registri e Register File.

- I **registri** sono memorie ad alta velocità che consentono di immagazzinare i dati prossimi ad essere processati. Fra di essi troviamo:
  - Il **Program Counter** è un registro della CPU che conserva l'address della prossima istruzione da eseguire. Esso è un reigstro "puntatore", anche detto instruction pointer.
  - Il **registro istruzione** contiene l'istruzione in esecuzione.
- La **ALU** recupera i dati dai registri della CPU, processa i dati contenuti nel registro istruzione e decodificati dalla CU, per poi salvare il risultato nel **registro di uscita**.
- Il **Register File** contiene i registri.

📖 Il **ciclo fetch-execute** è la dinamica fondamentale del funzionamento di un calcolatore e viene ripetuto all'infinito:

1. Si carica l'istruzione a cui si riferisce il Program Counter nel registro istruzione della CPU (**fetch**).
2. Il Program Counter recupera l'indirizzo dell'istruzione successiva (*MIPS: salta di 4 in 4 byte poiché tale è la lunghezza della parola di memoria*).
3. L'istruzione viene decodificata.
4. L'istruzione viene eseguita (**execute**).

La **CU** (Control Unit) ha la funzione di **controllare l'esecuzione** delle istruzioni da parte del datapath, mediante i *segnali di controllo*. Ha anche un ruolo nella **decodifica** e nell'**interpretazione** delle istruzioni.

I trasferimenti dei dati avvengono attraverso i **bus**:

- I **control bus** determinano l'operazione da svolgere (solitamente read o write).
- I **data bus** trasportano i dati.
- Gli **address bus** trasportano gli indirizzi.

Se i dati sono allineati, i trasferimenti degli stessi avvengono in blocco; altrimenti, ognuno dei 4 byte (MIPS) dell'istruzione avrà diverso indirizzo.

STRUTTURA DELLE ISTRUZIONI

Ogni istruzione (*MIPS: 32 bit, come in ogni architettura RISC*) riserva i 6 bit più significativi al **codice operativo (***opcode***)**, che indica la categoria dell'istruzione.

Esempio: le istruzioni aritmetico-logiche di tipo R hanno opcode 000000.

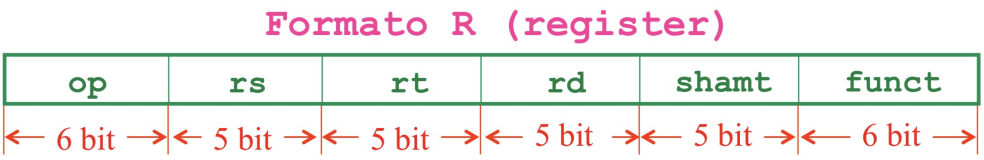
In generale, le istruzioni sono di diversi tipi:

ISTRUZIONI R

📖 Le istruzioni R operano sui **registri** e, spesso, sono operazioni di tipo aritmetico logico.

Le funzioni R si differenziano in base al campo *func* (6 bit meno significativi), che hanno lo scopo di identificare le specifiche operazioni di uguale opcode.

Le funzioni R sono anche deputate alla gestione di registri speciali (*mflo*, *mfhi*) ed al salto di registri (*jr*).



I registri *rs* e *rt* indicano i due **operandi sorgente**, mentre *rd* indica il **registro di destinazione**.

shamt indica lo **shift amount** ed è diverso da zero solo per le operazioni di shift.

*Esempio: Considerando come registro di destinazione **$t0**, e come addendi **$s1** e **$s2**,*

***Comando assembly:** add $t0, $s1, $2*

***Valori del registro R:** 0 17 18 8 0 32*

*(17, 18 e 8 sono i codici dei registri, inventati per ragion d'esempio. 32 è il **func** per la somma).*

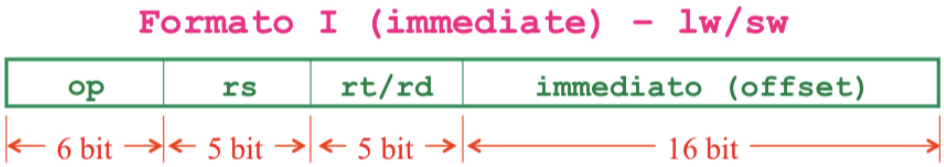
***Stringa in binario:** 000000 10001 10010 01000 00000 100000*

***Stringa in esadecimale:** 0x02324020*

ISTRUZIONI I

📖 Le **istruzioni di tipo I** contengono un **immediato di 16 bit**. Include le operazioni che si svolgono senza coinvolgere altri registri.

- **load-word (*lw*), store-word (*sw*)**: servono per recuperare o caricare dati in memoria.



L'***offset*** è il valore aggiunto all'indirizzo base per identificare l'indirizzo di memoria interessato: mentre *rs* è la parte alta dell'indirizzo di memoria, l'*offset* ne indica la parte bassa.

> ⚠️ **base register + offset → indirizzo ricercato in memoria.**

rt/*rd* indicano il registro in cui viene caricato il dato (*lw*)/il registro da cui si ottiene il dato (*sw*).

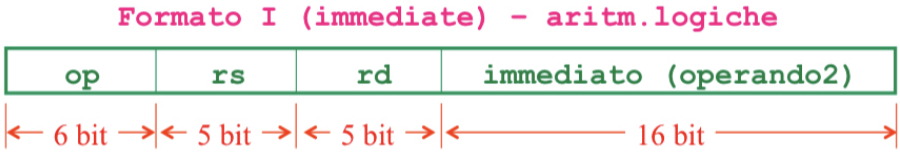
Esempio:

Considerando come registro rt $t1,

***Comando assembly**: lw $t1, 32($s3).*

In questo caso, 32 è l'offset mentre $s3 è l'indirizzo base. Il comando sw ha la stessa sintassi.

- **operazioni aritmetico-logiche immediate (*addi*, *subi*)**: effettuano un'operazione unaria fra un registro e una costante. In particolare **Set-Less-Than-Immediate** (*slti*) assegna valore 1 al registro destinazione se il registro sorgente contiene un valore minore della costante.



rs indica il registro contenente l'operando, *rd* è il registro destinazione.

Esempio:

Considerando come registro rd $t1 e come registro rs $s3,

***Comando assembly**: addi $t1, $s3, 4.*

Il comando slti ha la stessa sintassi.

- **salti condizionati (*beq*, *bne*)**: consentono il salto (***offset***) di n istruzioni (a partire dalla posizione del Program Counter → **salto relativo**) se le condizioni risultano vere (**Branch-On-Equal** o **Branch-On-Not-Equal**).

  L'offset è riportato nell'istruzione in **complemento a due**, per consentire anche i salti indietro.

  Il formato dell'istruzione è uguale a quello delle altre istruzioni I, ma i due registri indicano i **termini del confronto** e, nell'immediato, si ritrova il **numero di word** (4 byte) **da saltare**. Il Program Counter proseguirà a partire dall'istruzione successiva ai byte saltati.

  Esempio:

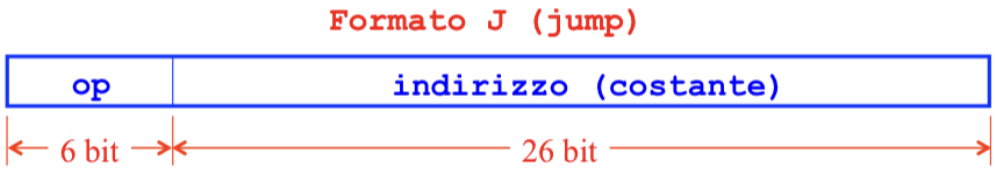
  Considerando come registri da confrontare rd $t1 e come registro rs $s3,

  ***Comando assembly**: beq $t1, $s3, 8.*

  Si noti che, mentre in linguaggio macchina l'offset è misurato in word, in assembly esso è misurato in byte. In assembly, i due bit meno significativi di tale salto valgono sempre 00. Il comando bne ha la stessa sintassi.

ISTRUZIONI J

📖 **Le istruzioni di tipo J (*j*)** sono relative al salto incondizionato.



I 26 bit meno significativi contengono **parte dell'indirizzo di parola** di destinazione.

Il **Program Counter** viene modificato: I 4 bit più significativi rimangono tali, mentre il resto viene sostituito dall'indirizzo di parola riportato nell'istruzione. I restatnti 2 bit meno significativi vengono posti a *00*.

Esempio:

Comando assembly: j L1

L1 è un'etichetta che indica il word address contenuto nell'istruzione.

PROCEDURE

Le **procedure** sono l'equivalente dei metodi nei linguaggi di programmazione di alto livello.

Per eseguire una procedura, è necessario cambiare il flusso di dati per saltare alla procedura e salvare l'indirizzo di ritorno ***$ra***.

Sintassi:

```
jal Indirizzo_procedura // JUMP AND LINK
...

jr $ra // JUMP TO REFERENCE
```

Ad ogni procedura viene dedicata un'area di memoria nello stack: lo **stack frame**.

Nel caso di **procedure annidate o ricorsive**, è necessario salvare l'indirizzo di ritorno *$ra* nello stack, poiché la procedura chiamata modificherà tale indirizzo di rientro.

Nel caso di **procedure "foglia"** (ovvero, che non richiamano altre procedure), non è necessario salvare *$ra* nello stack: è possibile lasciarlo salvato nel registro dato che nessuno lo modificherà.

CONVENZIONI SULL'USO DELLA MEMORIA

Dato che nell'assembly la gestione della memoria è **manuale**, è necessario adottare alcune **convenzioni sul suo utilizzo**.

In particolare, **heap e stack** occupano dinamicamente la stessa zona di memoria (da 0x10008000 a 0x7FFFFFFF). Il registro ***$sp*** (**stack pointer**) indica quanta

memoria stack è ancora disponibile. Esso contiene l'indirizzo della cima dello stack (che cresce verso il basso, contrariamente allo heap) e viene aggiornato ogni qualvolta un elemento viene aggiunto o eliminato.

Sotto l'indirizzo 0x10008000 si ritrovano, in ordine decrescente, la **static data memory,** la **text memory** e una **zona riservata** da 0x00000000 a 0x00400000.