Architettura dei Calcolatori

Giuseppe Pozzi

Impianti di Elaborazione
Facoltà di Ingegneria dell'Informazione
Politecnico di Milano

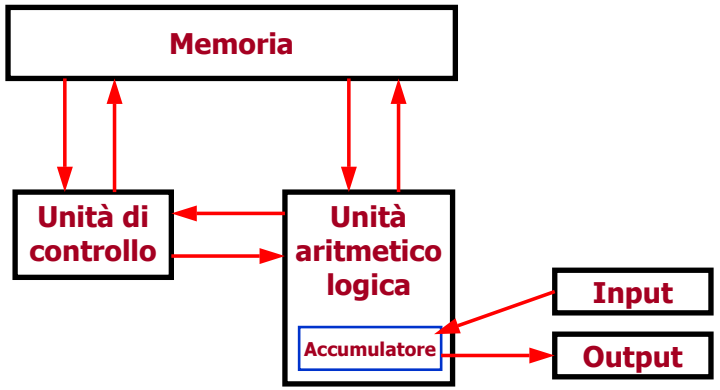
giuseppe.pozzi@polimi.it
- versione del 20 settembre 2002 -

Architettura dei Calcolatori

- **Bibliografia**:

  Tanenbaum A. S., Goodman J. R,
  "**Architettura dei computer** - Un approccio strutturato",
  Prentice Hall International, 2000 - paragrafo 2.1
- **Sommario**:
  - Organizzazione della CPU.
  - Esecuzione delle istruzioni.
  - Differenze fra RISC e CISC.
  - Principi di progettazione per i moderni calcolatori.
  - Parallelismo a livello delle istruzioni.
  - Parallelismo a livello di processore.

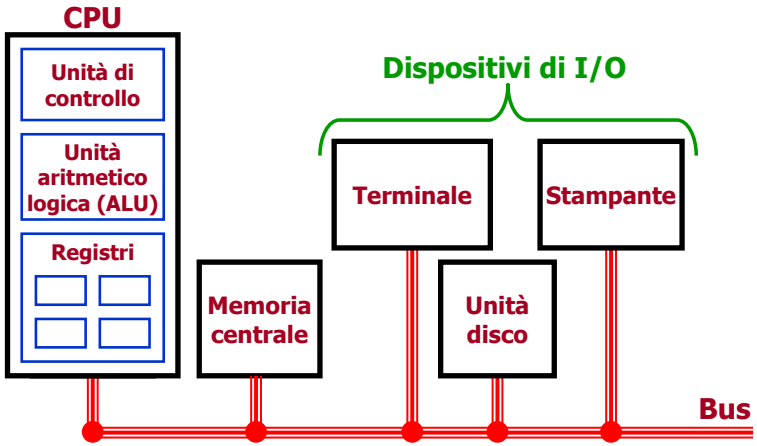
Macchina di von Neumann /1



Macchina di von Neumann /2

- Progetto effettivo (**macchina IAS**)
  - Memoria composta da **4096 parole** di **40 bit** contenenti
    - **due istruzioni di 20 bit** ciascuna (8 bit per indicare il tipo d'istruzione e 12 bit per indirizzare una delle 4096 parole), oppure
    - un numero **intero con segno di 40 bit**;
  - Unità aritmetico logica dotata di un registro **accumulatore** di **40 bit**
  - Esempi di istruzioni:
    - aggiungi il contenuto di una cella di memoria all'accumulatore;
    - trasferisci il contenuto dell'accumulatore in una cella di memoria;
    - … …

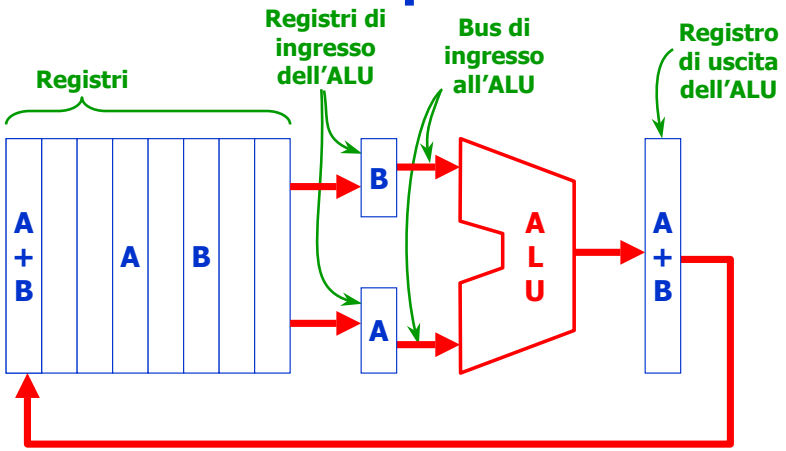
Organizzazione tipica di un calcolatore "bus oriented"



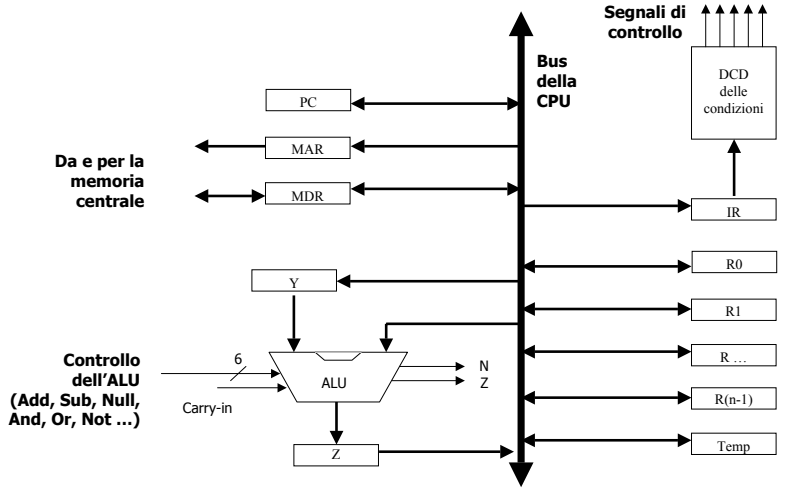
Elementi di una CPU

- **Unità di controllo**
  - legge le istruzioni dalla memoria e ne determina il tipo.
- **Unità aritmetico–logica**
  - esegue le operazioni necessarie per eseguire le istruzioni.
- **Registri**
  - **memoria ad alta velocità** usata per risultati temporanei e informazioni di controllo;
  - il **valore massimo** memorizzabile in un registro è determinato dalle **dimensioni** del registro;
  - esistono registri di uso generico e registri specifici:
    - **Program Counter** (**PC**) – qual è l'istruzione successiva;
    - **Instruction Register** (**IR**) – istruzione in corso d'esecuzione;
    - **…**

Struttura generica di un "data path"



Un "data path"



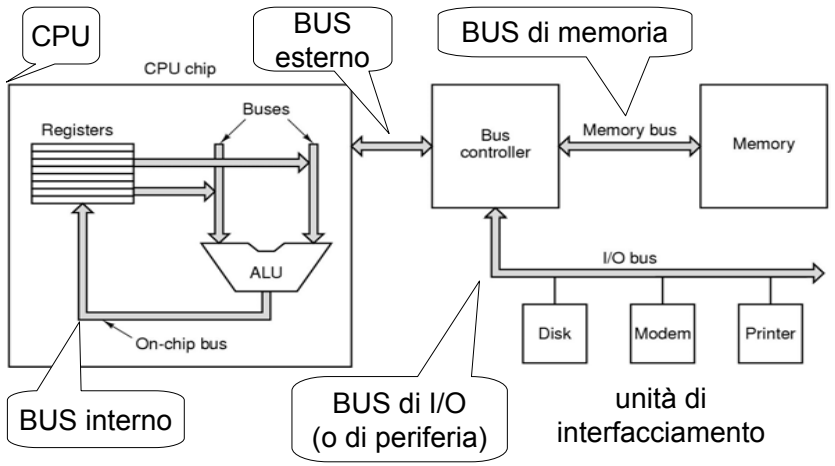
Alcune istruzioni assembler

- Mov R1, R2
  - Possibile codifica in linguaggio binario:
    0000 000 001 000 010
- Add R1, R3
  - Possibile codifica in linguaggio binario:
    0001 000 001 000 011
- Sub R2, R4
  - Possibile codifica in linguaggio binario:
    0010 000 010 000 100

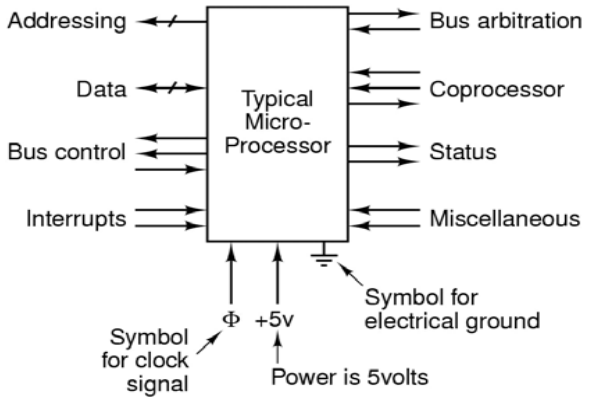
A esercitazione

- Approfondire le modalita' di indirizzamento.
- Individuare quali sono i segnali di controllo che devono essere emessi dall'unita' di controllo per eseguire alcune istruzioni.

Sistema con diversi BUS



Piedinatura di una CPU tipica



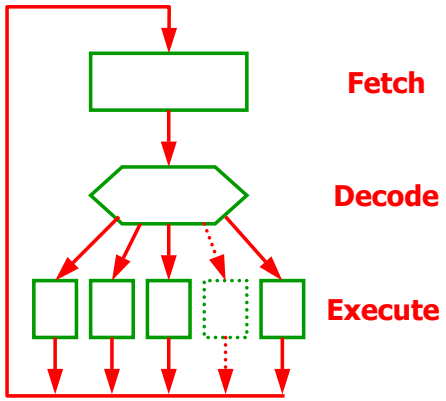
Categorie di istruzioni

- **Registro–Memoria**
  - gestiscono il **trasferimento dati** tra i registri della CPU e la memoria centrale;
  - l'unità di informazione trasferita tra memoria e registri prende il nome convenzionale di **parola** (**word**);
- **Registro–Registro**
  - utilizzano il contenuto dei registri per svolgere operazioni (tramite l'ALU) e memorizzano il risultato in un registro (**ciclo del data path**);
  - il **ciclo del data path** è al centro del funzionamento di quasi tutte le CPU, **quanto più piccolo è il tempo di ciclo tanto più veloce è il calcolatore**.

Esecuzione delle istruzioni

- Ciclo **Fetch–Decode–Execute** (**leggi–decodifica–esegui**)
  1. Prendi l'**istruzione corrente** dalla memoria e mettila nel **registro istruzioni** (**IR**).
  2. **Incrementa** il **program counter** (**PC**) in modo che contenga l'indirizzo dell'istruzione successiva.
  3. Determina il tipo dell'istruzione corrente (**decodifica**).
  4. Se l'istruzione usa una parola in memoria, determina dove si trova.
  5. Carica la parola, se necessario, in un registro della CPU.
  6. **Esegui** l'istruzione.
  7. Torna al punto 1 e inizia a eseguire l'istruzione successiva.

Ciclo Fetch–Decode–Execute



Le architetture RISC

- Sviluppo della **tecnologia** degli anni '80:
  - maggior **complessità** dei **circuiti integrati** (VLSI);
  - memoria centrale di **velocità** pari alla control store.
- Possibilità di realizzare CPU **cablate**
  - obiettivo: ottimizzare le **prestazioni complessive**;
  - fase 1: istruzioni semplici, eseguibili in **poco tempo**;
  - fase 2: istruzioni che possano essere **messe in esecuzione velocemente**;
  - risultato: CPU con un numero ristretto di istruzioni chiamate **RISC** (**Reduced Instruction Set Computer**) in contrasto con le **CISC** (**Complex ISC**) che dominavano il mercato;
  - esempi: SPARC, MIPS, …

RISC vs CISC

- RISC ha **prestazioni migliori**
  - istruzioni semplici, eseguite in un solo ciclo del data path;
  - 4/5 istruzioni RISC equivalgono a una istruzione CISC;
  - assenza di interpretazione dà un vantaggio di circa 10 volte.
- CISC ha garantito la **compatibilità** col passato
  - i clienti non hanno perso l'investimento SW;
  - modularità e flessibilità dell'evoluzione.
- CISC si è evoluto verso un **approccio misto**
  - processore combinato: una parte (core) RISC e una CISC;
  - approccio RISC per le istruzioni più semplici (e più comuni) eseguite in un ciclo del data path;
  - approccio CISC per le istruzioni più complesse.

Alcuni principi di progettazione

- Eseguire direttamente in **hardware** le istruzioni (soprattutto quelle **più comuni**).
- Ottimizzare la velocità di "lancio" delle istruzioni:
  - cercare di **far partire tante istruzioni** insieme;
  - sviluppare il **parallelismo a livello hardware**;
  - gestire l'esecuzione "**out of order**" delle istruzioni.
- **Semplificare la** fase di **decodifica**.
- Progettare architetture **load/store** in cui l'accesso alla memoria avviene solo tramite queste 2 istruzioni.
- Aumentare il **numero di registri** disponibili.

Parallelismo

- La **frequenza di clock**
  - influenza direttamente il tempo di ciclo del data path e quindi le prestazioni di un calcolatore;
  - è limitata dalla tecnologia disponibile.
- Il **parallelismo** permette di migliorare le prestazioni senza modificare la frequenza di clock. Esistono due forme di parallelismo:
  - **parallelismo a livello delle istruzioni** (architetture **pipeline** o architetture **superscalari**);
  - **parallelismo a livello di processori** (**Array computer**, **multiprocessori** o **multicomputer**).

Architettura pipeline

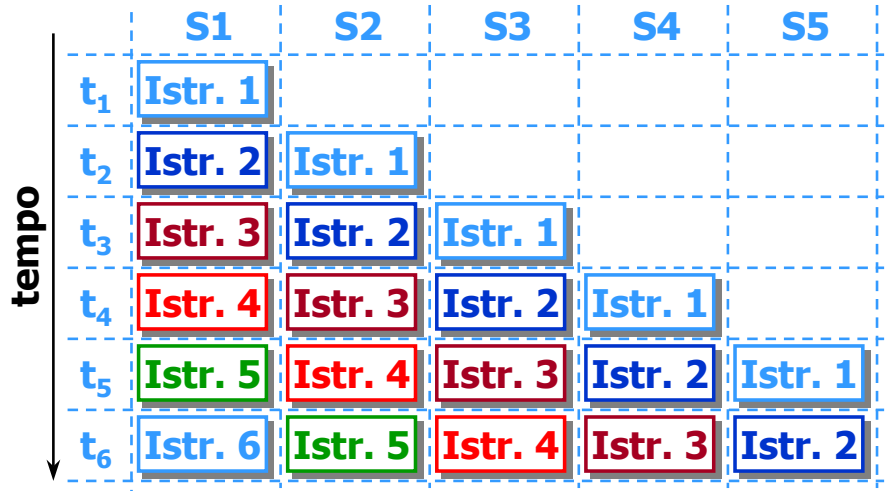
- Organizzazione della CPU come una "**catena di montaggio**"
  - la CPU viene suddivisa in "**stadi**", ognuno dedicato all'esecuzione di un compito specifico;
  - l'esecuzione di un'istruzione richiede il **passaggio attraverso** (tutti o quasi tutti) **gli stadi della pipeline**;
  - in un determinato istante, **ogni stadio esegue la parte di sua competenza di una istruzione**;
  - in un determinato istante, esistono **diverse istruzioni contemporaneamente in esecuzione**, una per ogni stadio.

Esempio di pipeline /1

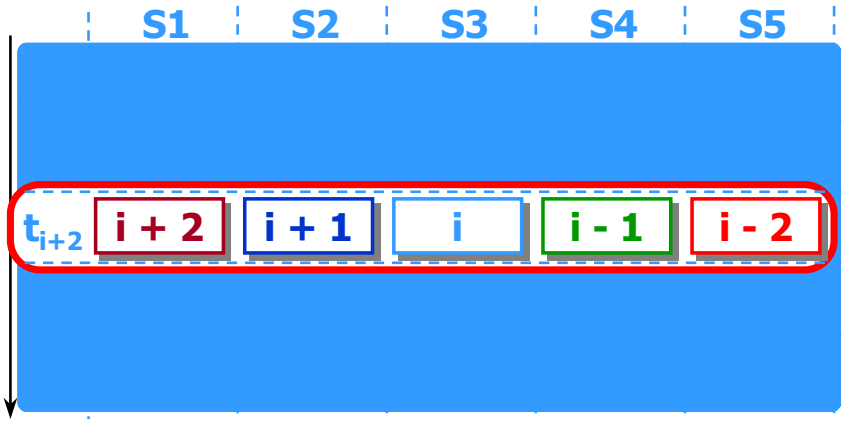
Pipeline in **cinque stadi**:

S1. **lettura istruzioni** dalla memoria e loro caricamento in un apposito buffer;
S2. **decodifica** dell'istruzione per determinarne il tipo e gli operandi richiesti;
S3. individuare e **recuperare gli operandi** dai registri o dalla memoria;
S4. **esecuzione** dell'istruzione, tipicamente facendo passare gli operandi per il data path;
S5. **invio dei risultati** al registro appropriato.

Esempio di pipeline /2

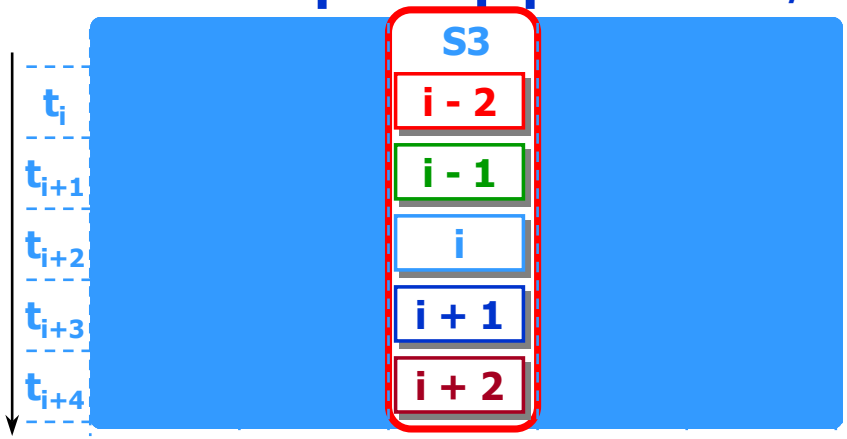


Esempio di pipeline /3



All'istante t_{i+2} ci sono 5 istruzioni in esecuzione

Esempio di pipeline /4



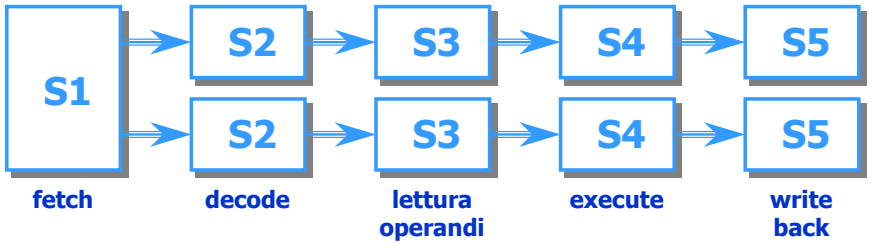
Lo stadio S3 esegue la parte di sua competenza di istruzioni successive l'una all'altra.

Prestazioni di una pipeline

- Il tempo di esecuzione (**latenza**) della singola istruzione non diminuisce, anzi **aumenta**
  - il tempo di attraversamento (latenza) della pipeline corrisponde al numero degli stadi (**N**) moltiplicato per il tempo di ciclo (**T**);
  - il tempo di ciclo è limitato dallo stadio più lento!
- **Aumenta** il numero di istruzioni completate nell'unità di tempo (**throughput**)
  - si completa **un'istruzione a ogni ciclo di clock**;
  - l'**incremento** di throughput è quasi **proporzionale al numero degli stadi!**

Architetture superscalari

- Vista la disponibilità di un maggior numero di transistor si inseriscono **più pipeline** nella stessa CPU
  - aumenta il parallelismo perché è possibile eseguire contemporaneamente diversi **flussi di istruzioni**;
  - è necessario garantire che non ci siano **conflitti** tra le istruzioni che vengono eseguite insieme; di solito il controllo è affidato al compilatore.



Architetture superscalari

In alternativa si possono **replicare le unità funzionali**

- rappresentano lo stadio più lento della pipeline (in genere richiedono diversi cicli di clock);
- è più semplice evitare i **conflitti** tra le diverse istruzioni.

Array computer

- Pensati per la soluzione di problemi che richiedono l'esecuzione delle **stesse operazioni su tanti dati**
  - **algoritmi** facilmente **parallelizzabili**, cioè possono essere suddivisi in modo efficiente tra diversi esecutori;
  - vettori o **matrici come operandi**.
- Due tipi di architetture
  - **array processor:** composti da tante unità funzionali uguali che eseguono le stesse istruzioni su dati diversi;
  - **vector processor:** le operazioni vengono eseguite da una sola unità funzionale, dotata di una pipeline profonda con tempo di ciclo molto breve. I dati stanno in **registri vettoriali** (un insieme di registri tradizionali che si possono leggere dalla memoria in una sola istruzione).