Architettura del Computer

Corso di Sistemi e Reti

October 15, 2025

Contents

1 Introduzione

L'architettura del computer descrive l'organizzazione e il funzionamento dei componenti di un computer, dalla CPU alla memoria, dai bus ai registri, in modo che possano elaborare informazioni.

Per un programmatore, comprendere l'architettura del computer è fondamentale per:

- Scrivere codice più efficiente e ottimizzato
- Capire le prestazioni e i limiti dei programmi
- Diagnosticare problemi di performance
- Comprendere come il codice ad alto livello viene tradotto in operazioni hardware

In questa introduzione vedremo i concetti fondamentali del **modello di Von Neumann**, della **CPU**, dell'**ALU**, dei **registri**, dei **bus** e del **ciclo macchina**.

2 Modello di Von Neumann

Il modello di Von Neumann, proposto nel 1945 da John von Neumann, è l'architettura base di quasi tutti i computer moderni. È caratterizzato da:

- **Memoria centrale** unica per dati e istruzioni (stored-program concept).
- **Unità di elaborazione centrale (CPU)** che esegue istruzioni sequenzialmente.
- **Bus di comunicazione** che trasferisce dati e istruzioni tra memoria e CPU.
- **Unità di input/output** per comunicare con l'esterno.

2.1 Il concetto di Stored Program

Una delle innovazioni più importanti del modello di Von Neumann è che le istruzioni del programma sono memorizzate nella stessa memoria dei dati. Questo significa che:

- I programmi possono essere facilmente caricati e modificati
- Un programma può modificare se stesso (self-modifying code)
- La CPU non distingue tra dati e istruzioni: entrambi sono sequenze di bit

2.2 Il collo di bottiglia di Von Neumann

Un limite intrinseco di questa architettura è il "Von Neumann bottleneck": CPU e memoria comunicano attraverso un unico bus, limitando la velocità di elaborazione. Le moderne architetture usano cache e parallelismo per mitigare questo problema.

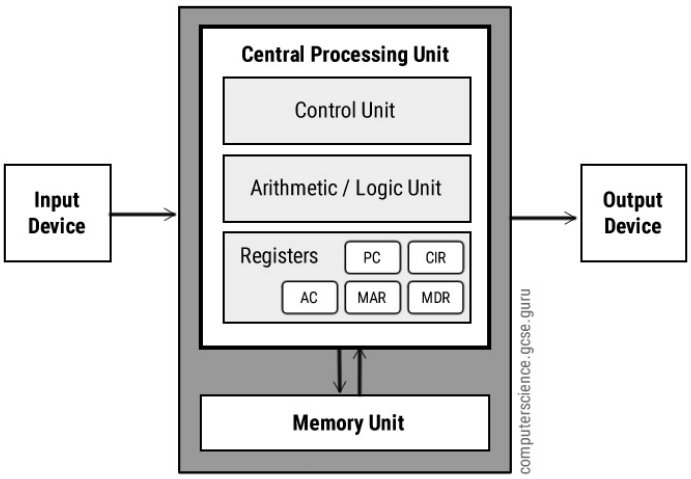

Figure 1: Schema del modello di Von Neumann

3 CPU (Central Processing Unit)

La CPU è il cervello del computer e si occupa di:

- Prelevare le istruzioni dalla memoria (fetch).
- Interpretarle ed eseguirle (decode ed execute).
- Gestire il flusso di dati tra memoria e periferiche.
- Coordinare tutte le operazioni del sistema.

La CPU è composta principalmente da:

- **ALU (Arithmetic Logic Unit)**: esegue operazioni aritmetiche e logiche.
- **Unità di controllo (Control Unit)**: coordina il funzionamento della CPU e del ciclo macchina, genera i segnali di controllo.
- **Registri**: memorie interne velocissime usate per conservare dati temporanei.
- **Cache**: memoria intermedia tra CPU e RAM per accelerare gli accessi.

3.1 Clock e frequenza

La CPU opera sincronizzata da un clock, un segnale periodico che scandisce le operazioni. La frequenza di clock (misurata in GHz) indica quanti cicli al secondo la CPU può eseguire. Tuttavia, più GHz non significa sempre più prestazioni: conta anche l'architettura e il numero di operazioni per ciclo (IPC - Instructions Per Cycle).

4 ALU (Arithmetic Logic Unit)

L'ALU è il componente che esegue tutte le operazioni di calcolo:

4.1 Operazioni aritmetiche

- Somma, sottrazione
- Moltiplicazione, divisione
- Operazioni con numeri interi e in virgola mobile (floating point)

4.2 Operazioni logiche

- AND, OR, NOT, XOR (porte logiche)
- Shift (spostamento bit a sinistra o destra)
- Rotazioni

4.3 Operazioni di confronto

- Maggiore, minore, uguale
- Impostazione di flag (Zero, Carry, Overflow, Sign)

Per il programmatore: ogni operazione nel codice (come `a + b`, `x > y`, `flag && condition`) viene tradotta in operazioni ALU. Comprendere questo aiuta a capire perché alcune operazioni sono più veloci di altre (es. moltiplicare per 2 con shift vs moltiplicazione).

5 Registri

I registri sono memorie ad altissima velocità (tempo di accesso ¡ 1 ns) integrate nella CPU. Sono la memoria più veloce ma anche la più limitata in capacità.

5.1 Registri principali

- **Registro accumulatore (ACC)**: memorizza temporaneamente risultati intermedi delle operazioni ALU.
- **Registro istruzioni (IR - Instruction Register)**: contiene l'istruzione corrente da eseguire.
- **Program Counter (PC)**: contiene l'indirizzo della prossima istruzione da eseguire. Viene incrementato automaticamente dopo ogni fetch.
- **Stack Pointer (SP)**: punta alla cima dello stack, usato per chiamate a funzioni e variabili locali.
- **Registri general purpose**: registri usati per memorizzare dati temporanei durante l'esecuzione (es. EAX, EBX, ECX, EDX in architetture x86).
- **Status Register (FLAGS)**: contiene i flag che indicano il risultato delle operazioni (zero, carry, overflow, segno).

5.2 Perché i registri sono importanti per i programmatori

- I compilatori (coloro che traducono il nostro codice in bits) ottimizzano il codice cercando di usare il più possibile i registri invece della RAM
- Le variabili locali frequentemente usate vengono mappate in registri
- Capire i registri aiuta a leggere codice assembly e debuggare a basso livello

6 Gerarchia di memoria

La memoria di un computer è organizzata in una gerarchia basata su velocità e capacità:

1. **Registri** (più veloce, 0.5 ns, 1 KB)
2. **Cache** (1-20 ns, 64 KB - 32 MB)
3. **RAM** (50-100 ns, 4-64 GB)
4. **SSD/HDD** (0.1-10 ms, 256 GB - 4 TB)

Principio di località: i programmi tendono ad accedere ripetutamente agli stessi dati (località temporale) e a dati vicini in memoria (località spaziale). Le cache sfruttano questo principio per migliorare le prestazioni.

7 Bus di dati e indirizzi

I bus sono canali di comunicazione tra i vari componenti del computer. Ogni bus è caratterizzato da una larghezza (numero di bit trasmessi contemporaneamente).

7.1 Tipi di bus

- **Bus dati**: trasferisce i dati effettivi tra CPU, memoria e periferiche. La larghezza del bus dati (es. 32 bit, 64 bit) determina quanti dati possono essere trasferiti in un ciclo.
- **Bus indirizzi**: trasporta l'indirizzo di memoria da leggere o scrivere. Un bus indirizzi di n bit può indirizzare 2^n locazioni di memoria (es. 32 bit = 4 GB, 64 bit = 16 exabyte teorici).
- **Bus di controllo**: trasmette segnali che coordinano le operazioni (read/write, interrupt, clock, reset).

7.2 Sistemi a 32 bit vs 64 bit

Un sistema a 64 bit significa che:

- I registri sono da 64 bit (possono gestire numeri più grandi)
- Il bus indirizzi è da 64 bit (può indirizzare più di 4 GB di RAM)
- Le istruzioni operano su parole da 64 bit

8 Ciclo macchina (Fetch-Decode-Execute)

Il ciclo macchina descrive il processo attraverso cui la CPU esegue le istruzioni. È un ciclo continuo e ripetitivo:

1. **Fetch (prelievo)**:
   - La CPU legge l'istruzione dalla memoria all'indirizzo indicato dal PC
   - L'istruzione viene caricata nel registro IR
   - Il PC viene incrementato per puntare alla prossima istruzione
2. **Decode (decodifica)**:
   - L'unità di controllo interpreta l'istruzione
   - Identifica l'operazione da compiere e gli operandi necessari
   - Prepara i segnali di controllo per l'ALU e altri componenti
3. **Execute (esecuzione)**:
   - L'ALU esegue l'operazione richiesta (calcolo, confronto, ecc.)
   - Vengono letti i dati dai registri o dalla memoria
   - Vengono aggiornati i flag di stato
4. **Store (memorizzazione)**:
   - Il risultato viene scritto in un registro o nella memoria
   - Lo stato della CPU viene aggiornato

8.1 Esempio pratico

Consideriamo l'istruzione: `ADD R1, R2, R3` (R1 = R2 + R3)

1. **Fetch**: preleva l'istruzione dalla memoria
2. **Decode**: riconosce ADD e identifica i registri R1, R2, R3
3. **Execute**: l'ALU somma i valori di R2 e R3
4. **Store**: il risultato viene scritto in R1

8.2 Pipeline e parallelismo

Le CPU moderne usano la pipeline per eseguire più istruzioni contemporaneamente in fasi diverse:

- Mentre un'istruzione è in Execute, la successiva è in Decode e un'altra in Fetch
- Questo aumenta il throughput (istruzioni al secondo)
- Complicazioni: hazard, branch prediction, out-of-order execution

9 Linguaggi di programmazione e hardware

9.1 Livelli di astrazione

1. **Linguaggio macchina**: codice binario eseguito direttamente dalla CPU
2. **Assembly**: rappresentazione simbolica del linguaggio macchina
3. **Linguaggi di basso livello**: C, C++ (controllo diretto su memoria e hardware)
4. **Linguaggi di alto livello**: Python, Java, JavaScript (forte astrazione dall'hardware)

9.2 Compilazione vs Interpretazione

- **Linguaggi compilati** (C, C++, Rust): tradotti completamente in codice macchina prima dell'esecuzione. Più veloci ma meno portabili.
- **Linguaggi interpretati** (Python, JavaScript): eseguiti riga per riga da un interprete. Più lenti ma più flessibili.
- **Linguaggi intermedi** (Java, C#): compilati in bytecode ed eseguiti da una macchina virtuale (JVM, CLR).

10 Concetti essenziali per programmatori

10.1 Rappresentazione dei dati

- **Bit e byte**: tutte le informazioni sono rappresentate in binario
- **Tipi di dati**: int (32 bit), long (64 bit), float (32 bit), double (64 bit)
- **Endianness**: ordine dei byte (big-endian vs little-endian)
- **Overflow**: cosa succede quando un calcolo supera la capacità del tipo

10.2 Memoria: unità di misura e conversioni

10.2.1 Unità di misura

La memoria del computer si misura in potenze di 2. Le unità principali sono:

- **Bit (b)**: l'unità elementare, può valere 0 o 1
- **Byte (B)**: 8 bit = 1 Byte
- **Kilobyte (KB)**: 2^{10} Byte = 1.024 Byte $\approx$ 1.000 Byte
- **Megabyte (MB)**: 2^{20} Byte = 1.048.576 Byte $\approx$ 1.000.000 Byte
- **Gigabyte (GB)**: 2^{30} Byte = 1.073.741.824 Byte $\approx$ 1.000.000.000 Byte
- **Terabyte (TB)**: 2^{40} Byte = 1.099.511.627.776 Byte $\approx$ 1.000.000.000.000 Byte

Nota importante: esiste una differenza tra unità binarie (KB, MB, GB) e unità decimali (kB, MB, GB secondo lo standard SI). Nelle specifiche dei produttori di storage si usano spesso le unità decimali (1 GB = 10^9 byte), mentre i sistemi operativi usano quelle binarie.

10.2.2 Formule di conversione

- Da bit a Byte: dividere per 8
- Da Byte a bit: moltiplicare per 8
- Da KB a Byte: moltiplicare per 1.024
- Da MB a KB: moltiplicare per 1.024
- Da GB a MB: moltiplicare per 1.024
- In generale: per salire di livello moltiplicare per 1.024, per scendere dividere per 1.024

10.2.3 Esercizi svolti

Esercizio 1: Converti 4096 bit in Byte.
Soluzione:

$$4096 \text{ bit} \div 8 = 512 \text{ Byte}$$

Esercizio 2: Converti 2 GB in MB.
Soluzione:

$$2 \text{ GB} \times 1.024 = 2.048 \text{ MB}$$

Esercizio 3: Converti 500 MB in bit.
Soluzione:

$$\begin{aligned} 500 \text{ MB} &= 500 \times 1.024 \text{ KB} = 512.000 \text{ KB} \\ &= 512.000 \times 1.024 \text{ Byte} = 524.288.000 \text{ Byte} \\ &= 524.288.000 \times 8 \text{ bit} = 4.194.304.000 \text{ bit} \end{aligned}$$

Esercizio 4: Un file occupa 8.388.608 bit. Quanti MB sono?
Soluzione:

$$\begin{aligned} 8.388.608 \text{ bit} &\div 8 = 1.048.576 \text{ Byte} \\ &\div 1.024 = 1.024 \text{ KB} \\ &\div 1.024 = 1 \text{ MB} \end{aligned}$$

Esercizio 5: Una memoria RAM è da 16 GB. Quanti byte può contenere?
Soluzione:

$$16 \text{ GB} = 16 \times 1.024^3 \text{ Byte} = 16 \times 1.073.741.824 = 17.179.869.184 \text{ Byte}$$

10.2.4 Esercizi da svolgere

Esercizio 1: Converti 2048 Byte in KB.

Esercizio 2: Converti 5 GB in Byte.

Esercizio 3: Un video occupa 750 MB. Quanti bit sono?

Esercizio 4: Una chiavetta USB ha capacità di 32 GB. Quanti file da 4 MB ciascuno può contenere?

Esercizio 5: Un disco rigido da 2 TB quanti GB contiene? E quanti MB?

10.2.5 Soluzioni degli esercizi

Soluzione Esercizio 1:

$$2048 \text{ Byte} \div 1.024 = 2 \text{ KB}$$

Soluzione Esercizio 2:

$$5 \text{ GB} = 5 \times 1.024^3 = 5.368.709.120 \text{ Byte}$$

Soluzione Esercizio 3:

$$\begin{aligned} 750 \text{ MB} &= 750 \times 1.024 \times 1.024 \text{ Byte} = 786.432.000 \text{ Byte} \\ &= 786.432.000 \times 8 = 6.291.456.000 \text{ bit} \end{aligned}$$

Soluzione Esercizio 4:

$$\begin{aligned} 32 \text{ GB} &= 32 \times 1.024 \text{ MB} = 32.768 \text{ MB} \\ 32.768 \text{ MB} &\div 4 \text{ MB} = 8.192 \text{ file} \end{aligned}$$

Soluzione Esercizio 5:

$$\begin{aligned} 2 \text{ TB} &= 2 \times 1.024 \text{ GB} = 2.048 \text{ GB} \\ 2.048 \text{ GB} &= 2.048 \times 1.024 \text{ MB} = 2.097.152 \text{ MB} \end{aligned}$$

10.3 Puntatori e indirizzi di memoria

- Un puntatore contiene l'indirizzo di memoria di un dato
- Fondamentali in C/C++ per gestire memoria dinamica
- Permettono di passare parametri per riferimento
- Errori comuni: segmentation fault, memory leak, dangling pointer

10.4 Stack e Heap

- **Stack**: memoria automatica per variabili locali e chiamate a funzione. LIFO (Last In First Out). Veloce ma limitata.
- **Heap**: memoria dinamica allocata con malloc/new. Più lenta ma flessibile. Va gestita manualmente (free/delete).

10.5 Performance e ottimizzazione

- **Cache-friendly code**: accedere ai dati in modo sequenziale sfrutta la cache
- **Branch prediction**: evitare if complessi in loop critici
- **Allineamento dati**: struct e array allineati sono più veloci
- **SIMD**: Single Instruction Multiple Data per parallelismo

11 Conclusione

Questi concetti costituiscono la base per comprendere come funziona un computer. Per un programmatore, capire l'architettura hardware significa:

- Scrivere codice più efficiente
- Comprendere i limiti e le possibilità dell'hardware
- Debuggare problemi complessi
- Fare scelte architetturali migliori nei propri progetti

Il modello di Von Neumann, il ruolo della CPU e dei registri, l'importanza dei bus e il ciclo macchina sono alla base di ogni sistema di calcolo moderno. Anche se i linguaggi di alto livello astraggono molto dall'hardware, conoscere questi fondamenti distingue un buon programmatore da uno eccellente.