Architettura dei sistemi di elaborazione

Materials

Introduzione

Il calcolatore digitale è una macchina che svolte dei compiti eseguendo delle istruzioni che gli vengono fornite dall'utente; l'insieme di queste istruzioni prende il nome di **programma**.
I circuiti elettronici dei computer sono in grado di comprendere ed eseguire soltanto alcune istruzioni base (sommare due numeri, riconoscere se un numero è uguale o diverso da 0, spostare dei dati da una parte all'altra della memoria) che prendono il nome di **linguaggio macchina**, in cui tutti i programmi devono essere convertiti per essere utilizzati.
Da qui la necessità di creare un nuovo insieme di istruzioni che sia più comodo da utilizzare rispetto al linguaggio macchia.

Approccio strutturale

Questa necessità di dover creare più insiemi di istruzioni prende il nome di **approccio strutturale**. Globalmente queste nuove istruzioni definiscono un **linguaggio**, che chiameremo **L1**, mentre indicheremo con **L0** il **linguaggio macchina.**
Tenendo sempre in mente che il computer è in grado di riconoscere ed eseguire soltanto istruzioni in **L0**, il problema che affrontiamo con l'approccio strutturale è quello di convertire istruzioni **L1** in istruzioni **L0**. Questa conversione può essere affrontata in due modi distinti:

- **Traduzione:** Le istruzioni del programma **L1** vengono convertite in istruzioni analoghe **L0**. Il programma risultante avrà solo istruzioni **L0**; di conseguenza il

computer salverà in memoria il nuovo programma che verrà usato al posto di quello **L1**.

- **Interpretazione:** Viene creato un programma **L0** che accetta in ingresso istruzioni **L1**. In questo caso le istruzioni **L1** non sono altro che dati che vengono interpretati e sostituiti con l'equivalente in **L0**. Il computer avrà quindi il controllo soltanto del programma **interprete**.

Per rendere la traduzione e l'interpretazione realmente efficaci, è necessario che il linguaggio **L1** non sia troppo diverso dal linguaggio **L0**. Questo, però, fa sì che il linguaggio **L1**, seppur superiore e più facile da usare del linguaggio macchina, è ancora lontano dall'essere un linguaggio ideale.
La soluzione più ovvia è quella di creare un linguaggio **L2** ancora più rivolto agli utenti, che verrà poi tradotto in o interpretato in **L1**. Questa ricerca di un linguaggio sempre più indirizzato agli utenti può continuare in modo indefinito, finché non si arrivi ad un linguaggio realmente ideale. Ciò ci porta a pensare al computer come ad una struttura a **livelli**, o **strati**.

Macchine multilivello

Gli attuali computer sono formati da due o più livelli. Il livello 0 rappresenta il vero e proprio hardware della macchina, i cui circuiti eseguono programmi scritti nel linguaggio macchina del livello 1.
Uno degli elementi più importanti del livello 0, detto anche **livello logico digitale**, sono le **porte** (che sono costruite con componenti analogici come i **transistor**). Ogni **porta** è in grado di ricevere uno o più **input digitali** (ovvero segnali **0** e **1**) e calcola in output una semplice funzione dei valori in ingresso, come una funzione **AND** o **OR**.
È possibile combinare un piccolo numero di porte per creare una **memoria da 1bit**. Mettendo insieme più memorie, in gruppi da **16**, **32** o **64**, si vengono a creare quelli che vengono definiti **registri**. I **registri** sono delle memorie molto piccole in grado di memorizzare un valore fino ad un massimo numero di cifre digitali.

Subito dopo troviamo il **livello di micro-architettura**, **livello 1**. Qui troviamo una memoria locale, formata da un gruppo di registri (di solito da 8 o 32), e un circuito chiamato **ALU** (Arithmetic Logic Unit, Unità Aritmetico Logica) capace di effettuare delle semplici operazioni aritmetiche. I registri sono collegati alla **ALU** formando un **percorso dati** che permette di selezionare uno o più registri, permettere alla **ALU** di effettuare

delle operazioni su di loro (ad esempio sommandoli), e registrare il risultato su uno dei registri.

- In alcune macchine questo passaggio avviene tramite un **microprogramma**, mentre in altre avviene direttamente a livello hardware. Nelle macchine in cui questo passaggio avviene a livello software, il **microprogramma** non è altro che un interprete del linguaggio macchina del livello 2 che sfrutta il percorso dati per analizzare le istruzioni ed eseguirle una alla volta.

Il livello 2 viene comunemente chiamato il **livello ISA** (Architettura dell'insieme delle istruzioni). In questo livello troviamo il **sistema operativo**, che non è altro che un'interprete delle istruzioni del livello 3.

Il livello 3, che prende il nome di **livello macchina del sistema operativo**, è solitamente un livello ibrido. Condivide alcune istruzioni con il secondo livello, **Livello ISA**, mentre aggiunge nuove istruzioni che vengono interpretate dal **sistema operativo**. Le funzioni in comune tra **livello ISA** e **livello 3** vengono eseguite direttamente dal microprogramma (o dai circuiti elettronici), mentre le nuove istruzioni vengono interpretate dal sistema operativo.

I livelli che abbiamo visto fino ad ora sono pensati per eseguire interpreti e traduttori di supporto per i livelli più alti (questi traduttori e interpreti vengono scritti da professionisti chiamati **programmatori di sistema**). Mentre a partire dal livello 4 iniziamo a trovare linguaggi utilizzati per risolvere problemi applicativi. Un altro cambiamento è che, mentre i livelli sottostanti erano sempre interpretati, a partire dal livello 4 i linguaggi che vengono utilizzati sono convertiti mediante traduzione.

Il livello 4, **livello del linguaggio assemblativo**, consente ai programmatori un modo per scrivere programmi per i livelli 1, 2 e 3 in una forma più "facile" rispetto alla scrittura nel linguaggio dei livelli precedenti. Questo **linguaggio assemblativo** viene prima tradotto (il traduttore prende il nome di **assemblatore**) nel linguaggio dei livelli sottostanti e poi interpretato dai relativi microprogrammi o dalla componente hardware.

Il livello 5 è formato da tutti quei **linguaggi ad alto livello** che vengono usati per programmi applicativi (C, C++, Java, Python..). Questi linguaggi vengono generalmente tradotti al livello 3 o 4 da un traduttore chiamato **compilatore**.

Riassumendo, il concetto chiave da ricordare è che il computer è una macchina costruita a livelli. Ogni livello viene costruito a partire da quello che lo precede ed ogni livello è caratterizzato dalla presenza di oggetti o operazioni differenti.

- Per **Hardware** intendiamo gli oggetti tangibili del computer: circuiti, cavi, trasformatori, memorie..). Per **Software** intendiamo gli **algoritmi**, ovvero le istruzioni dettagliate su come eseguire un determinato compito, e la loro rappresentazione per i computer, ovvero i **programmi**.
  Nei primissimi computer il confine tra questi due mondi era chiaro, ma nel corso del tempo il confine si è sfocato per via dell'aggiunta, della rimozione e dell'unione di livelli, tanto che oggi è difficile separare questi due mondi.

  Hardware e software sono logicamente equivalenti.

Generazione zero - Computer meccanici (1642 - 1945)

- Computer meccanici in grado di eseguire operazioni aritmetiche di base.
- **1944** - **Mark I** (Howard Aiken): Calcolatore composto da 72 parole e 23 numeri decimali.

Prima generazione - Valvole (1945-1955)

- **1945** - **Enigma**: Macchina codificatrice/decodificatrice di messaggi.
- **Colossus**: Primo elaboratore digitale
- **1946** - **Eniac**: Electronic Numerical Integrator And Computer. COmposto da 18000 valvole termoioniche, 1500 relé e 20 registri. Ogni registro era in grado di memorizzare un numero decimale a 10 cifre.
- **Macchina di von Nuemann (Macchina IAS)**: 4 Componenti principali:

    - La memoria da 4096 locazioni, ogni locazione conteneva 40 bit. Ogni parola poteva occupare due istruzioni da 20 bit oppure un numero da 40 bit. Le istruzioni erano composte da 8 bit, che definivano il tipo di istruzione, e dai restanti bit che specificavano una delle 4096 parole di memoria.
    - Unità aritmetico-logica e l'unità di controllo erano il cervello del computer. Nei computer moderni sono fusi in un unico chip, la **CPU**. Nell'unità aritmetico-logica troviamo uno speciale registro da 40 bit chiamato **accumulatore**. La tipica istruzione sommava una parola di memoria all'accumulatore oppure ne copiava il contenuto in memoria.
    - Unità di controllo
    - Dispositivi di input ed output

Seconda generazione - Transistor (1955-1965)

- **TX-0**: primo computer a transistor
- **PDP-1**: 4096 parole di memoria da 18 bit e poteva eseguire circa 200.000 istruzioni al secondo. Era dotato di un display visuale in grado di disegnare punti in qualsiasi parte dello schermo da 512x512 px.
- **PDP-8**: Per la prima volta un unico bus (omnibus). Un **bus** è un insieme di cavi paralleli utilizzato per connettere i diversi componenti del computer.
- **CDC 6600**: CPU dotata di varie unità funzionali che potevano lavorare contemporaneamente (operazioni matematiche). Si potevano eseguire circa 10 istruzioni contemporaneamente.

Terza generazione - Circuiti integrati (1965-1980)

- L'invenzione dei circuiti integrati consente di realizzare su un unico chip decine di transistor.
- Computer più piccoli, veloci ed economici.
- **IBM System/360**:
    - Famiglia di computer dotati dello stesso linguaggio assemblativo, con dimensioni e potenza delle macchine crescente.

    - Il software scritto per un modello “base” funzionava anche sul modello più grande, ma non valeva il contrario: spostando un programma per un computer più grande su uno più piccolo, poteva non bastare la memoria a disposizione.
    - **Multiprogrammazione**: avere più programmi in memoria nello stesso tempo, mentre si è in attesa di completare un’operazione si possono eseguire calcoli.
    - 2^24 e 2^32 byte.

Quarta generazione - Integrazione a grandissima scala (1980-?)

- Tecnologia **VLSI (Very Large Scale Integration):**
    - Si passa da decina di migliaia, a centinaia di migliaia, a milioni di transistor stampati in un unico chip.
    - Computer più piccoli e più veloci
- Inizio dell’era dei **Personal Computer**:
    - Era compito dell’acquirente montare il computer
    - Il software non era incluso, qualsiasi programma si voleva eseguire andava scritto
    - **Intel 8080**
- **CP/M:** Primo vero sistema operativo con un file system e uno **shell**, un interprete di comandi in grado di eseguire una serie di istruzioni
- **Apple I** e **Apple II**: Primi personal computer più diffusi
- **PC IBM**:
    - Integra il nuovo processore **Intel 8088**
    - Sistema operativo **Microsoft MS/DOS**
- **Macintosh (1984)**:
    - Primo personal computer dotato di **GUI** (Graphical User Interface)
- **Osborne-1**: Primo vero personal computer portatile (peso di 11kg)
- Prime versioni di **Windows**

- **Intel 80386** e **80486**:
  - Processori a 32 bit
  - Le versioni successive diventeranno rispettivamente **Intel Pentium** ed **Intel Core**
  - Architettura x86
- **Progettazione RISC**:
  - Architetture molto più semplici e veloci
  - In grado di eseguire più istruzioni contemporaneamente
- **Circuito FPGA:**
  - Circuito integrato con porte logiche che potevano essere “programmate” per creare sistemi informatici specializzati per applicazioni uniche al servizio di un ristretto numero di utenti (prototipazione, applicazioni di progettazione, istruzione)
- **1992 Prima macchina a 64 bit di tipo RISC**
- **Fine anni 90:**
  - Stallo nella ottimizzazione dei chip: non si riusciva a rendere i programmi più veloci e raffreddare i processori era troppo costoso
  - **Nascono le prime macchine dual core**:
    - **IBM dual-core Power4**: prima macchina con due processori su un unico chip

Quinta generazione - Computer a basso consumo e computer invisibili (1989-?)

- **PDA (Personal Digital Assistant)**:
  - **GridPad** ed **Apple Newton**:
    - Piccolo schermo dove gli utenti potevano scrivere con una speciale penna.
  - L’evoluzione di questi dispositivi porta agli smartphone
  - **Primi anni 90 - IBM Simon**:

        - Primo telefono cellulare touchscreen dotato di PDA
- **Computer Invisibili:**
    - Computer integrati in elettrodomestici, orologi, carte di credito..
    - In futuro i computer saranno integrati in qualsiasi oggetto della vita quotidiana e per questo saranno invisibili.

Famiglie di computer

Architettura x86

Con il nome **Architettura x86** si intende l'**architettura di set di istruzioni** (**ISA**) introdotta da **Intel** con il processore **8086**, processore a 16 bit. I processori successivi erano tutti retrocompatibili con il set di istruzioni originale utilizzato nel processore **8086**, e poiché tutti i processori terminavano con il numero 86, l'**ISA** prende il nome di **x86**.
Con l'introduzione del processore **80386**, il set di istruzioni **x86** viene esteso ad un sistema a 32 bit. Da qui nasceranno tutti i processori **Pentium** (Pentium I, II, III..), anche loro con architettura a 32 bit.
A questo set di istruzioni vengono introdotte, nel tempo, nuove istruzioni speciali tra cui **MMX** e **SSE**. Le istruzioni **MMX** (MultiMedia eXtension) servivano per migliorare i calcoli necessari per l'elaborazione audio e video, rendendo superflua l'esistenza di co-processori dedicati alle attività multimediali. Le istruzioni **SSE** vanno ad ampliare l'**MMX** migliorando tutto ciò che è inerente alla grafica 3D.
All'architettura x86 seguirà l'architettura **x64**, ovvero il set di istruzioni che viene esteso alle macchine a 64 bit.

Architettura ARM

L'**architettura ARM** indica una famiglia di microprocessori a 32-64 bit che ha avuto grande successo nel mercato dei dispositivi mobili, a basso consumo e integrati. I processori con architettura ARM godono di alta velocità e bassissimi consumi, impiegati principalmente in dispositivi come i **PDA**.

Architettura AVR

L’**architettura AVR** viene utilizzata in sistemi integrati di fascia bassa (microcontrollori). Oggi l’interesse maggiore per questo tipo di architettura è dovuto alla sua presenza nella piattaforma **Arduino**.
L’architettura AVR viene realizzata in tre classi.
La classe più bassa, **tinyAVR**, viene progettata per le applicazioni con maggiori vincoli in termini di spazio, potenza e costi. Essa comprende una CPU a 8 bit, il supporto di base per I/O digitale e Il supporto per un ingresso analogico (per esempio, la lettura della temperatura di un termostato).
Il **megaAVR** (presente in **Arduino**), è dotato anche del supporto I/O seriale, di orologi interni e di uscite analogiche programmabili.
Il **AVR XMEGA** incorpora anche un acceleratore per le operazioni di crittografia e il supporto integrato per interfacce USB.

Ogni classe ci processori AVR include generalmente 3 tipi di **memoria**: **flash**. **EEPROM** e **RAM**.
La memoria flash è programmabile con l’ausilio di un’interfaccia esterna ed è qui dove il codice del programma ed i dati vengono memorizzati. Questo tipo di **memoria non è volatile, ovvero i dati in essa contenuti vengono mantenuti anche quando il sistema viene spento**. Ance la memoria **EEPROM** non è volatile, ma a differenza della flash, può essere modificata da l programma durante l’esecuzione. In questa memoria esiste un sistema integrato che mantiene le informazioni di configurazione dell’utente (ad esempio orario visualizzato in 12 o 24 ore).
Infine, la **RAM** è la memoria in cui sono mantenute le variabili del programma in esecuzione. Questa **memoria** è di tipo **volatile**, **quindi i dati contenuti in essa vengono persi quando si toglie alimentazione al sistema.**

Unità metriche

Quando parliamo delle dimensioni di **memorie**, dischi, file, database.. le loro dimensioni sono sempre **potenze di 2**. **1 KB** contiene **1024** e non 1000 byte (**2^10 = 1024**), analogamente 1 MB contiene 2^20 byte, 1 GB 2^30.
Quando parliamo invece di velocità di trasferimento dati, parliamo di **Kbps (10^3)**, **Mbps**, **Gbps**.. e in questo caso parliamo di potenze di 10. **1 Kbps** trasmette 1000 bit al secondo, una Lan a 10 Mbps trasmette 10.000.000 bit/s.

Prefisso	**Valore**		**Prefisso**	**Valore**
KB	2^10		Kbps	10^3

MB	2^20		Mbps	10^6
GB	2^30		Gbps	10^9
TB	2^40		Tbps	10^12

Processori

La **CPU** (**Central Processing Unit**) è il “cervello” del computer e la sua funziona è quella di eseguire i programmi contenuti nella memoria principale prelevando le loro istruzioni ed eseguendole una dopo l’altra. I componenti sono collegati tra loro mediante un **bus** (insieme di cavi paralleli); i **bus** possono essere esterni alla **CPU**, per connetterla alla memoria e ad altri dispositivi, oppure interni.

Tra le parti della quale si compone la **CPU** troviamo l’**unità di controllo** e l’**unità aritmetico logica**. La prima si occupa di prelevare le istruzioni dalla memoria e di determinarne il tipo, mentre la seconda esegue le operazioni necessarie (ad esempio l’addizione e l’AND) per eseguire le istruzioni.

La **CPU** contiene anche una piccola memoria ad alta velocità costituita da **registri**. Questa memoria serve per memorizzare i risultati temporanei e alcune informazioni di controllo. Solitamente i registri che compongono questa memoria hanno una funzione e una dimensione predefinite; ognuno di loro può contenere un numero, il cui valore può variare fino ad un massimo che dipende dalla dimensione del registro (dimensione solitamente uguale per tutti i registri).

I **registri** più importanti sono:

- **Program Counter**: “punta” alla successiva istruzione da prelevare per l’esecuzione
- **Instruction Counter** o **Registro istruzione corrente:** contiene l’istruzione attualmente in esecuzione.

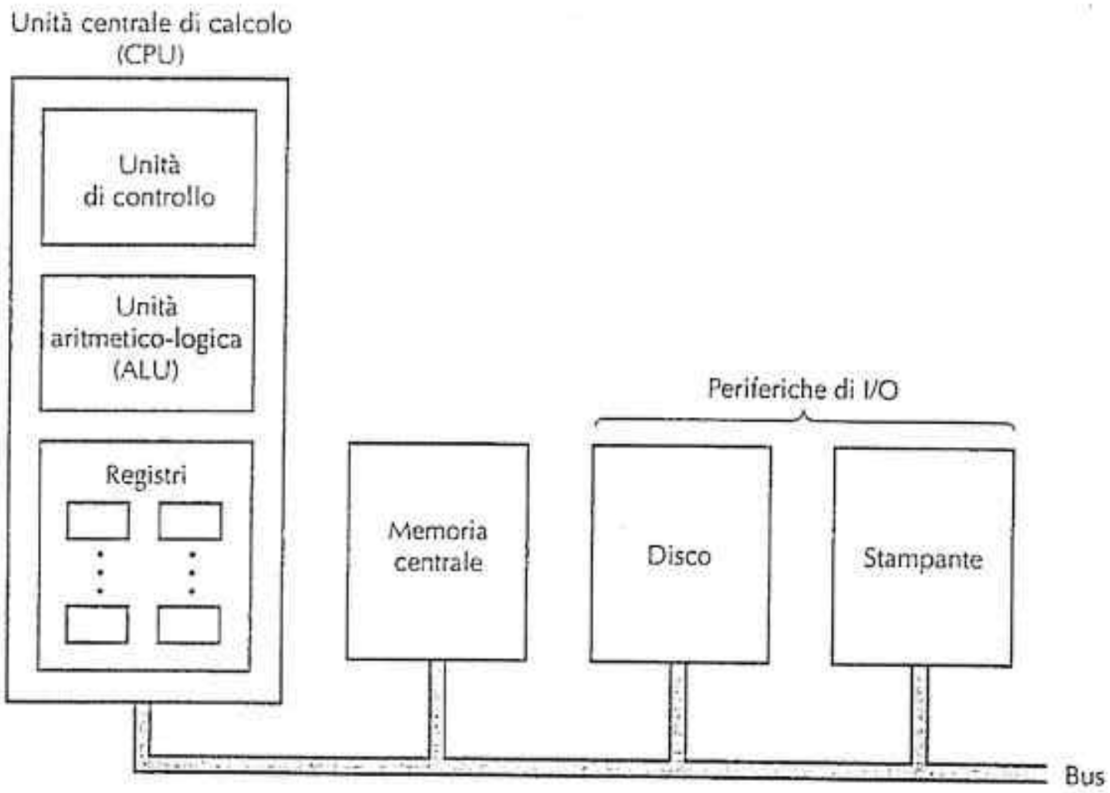


Figura 2.1 Organizzazione di un semplice calcolatore con una CPU e due periferiche di I/O.

Organizzazione della CPU

Il **percorso dati** (**datapath**) di una tipica CPU di **von Neumann** è composto dai **registri** e dalla **ALU** (unità logico aritmetica) collegati tra loro tramite bus. I registri della CPU sono collegati ai due registri input della ALU (indicati in figura con le lettere A e B) che, mentre la ALU è occupata nell'esecuzione di alcune computazioni, mantengono i dati in ingresso finché non posso essere scaricati nell'ALU.

La ALU esegue semplici operazioni sui suoi input (come addizioni e sottrazioni) ed il risultato generato viene memorizzato nell'apposito registro di output. Volendo, questo risultato può essere riportato in uno dei registri della CPU per essere, successivamente, salvato in memoria.

La maggior parte delle **istruzioni** del **percorso dati** si dividono in due principali categorie:

- Istruzioni **registro-memoria**: non fanno altro che mettere in comunicazione i registri della CPU con la memoria. Possiamo prelevare "parole di memoria" per

portarle nei registri, dove sono poi utilizzabili nell’alu, oppure possiamo copiare valori di registri nella memoria.

- Istruzioni **registro-registro**: mettono in comunicazione i vari registri. Una tipica istruzione è prelevare due operandi dai registri della CPU per essere portati nei registri di input dell’ALU.

- Il processo che consiste nel portare i due operandi attraverso la ALU prende il nome di **ciclo del percorso dati** e rappresenta il cuore della maggior parte delle CPU. I computer moderni dispongono di più ALU che operano in parallelo, specializzate su funzioni diverse.
  Più veloce è il ciclo del percorso dati, maggiore sarà la velocità del calcolatore.

Il modo in cui le **CPU** eseguono le istruzioni prende il nome di **ciclo esecutivo delle istruzioni** (**fetch-decode-execute**)**:**

- Viene prelevata la successiva istruzione dalla memoria e viene portata nell’IR
- Viene modificato il PC per farlo puntare alla prossima istruzione
- Viene determinato il tipo di istruzione
- Se l’istruzione usa una parola di memoria, viene localizzata
- Se necessario, viene prelevata la parola e portata in un registro della CPU
- Esegue l’istruzione

- Questo procedimento può essere eseguito anche a livello software con un programma **interprete.**

- Per **architettura** si intende una famiglia di macchine con un set base di istruzioni comune.

RISC e CISC

RISC: computer con un insieme ridotto di istruzioni (Reduced Instruction Set Computer)

CISC: computer con un insieme di istruzioni complesso (Complex Instruction Set Computer)

I processori **RISC** offrono un numero relativamente basso di istruzioni (es. 50), al contrario dei processori **CISC** (es, 200-300). Inoltre, le istruzioni dei processori **RISC** sono semplici e vengono eseguite direttamente in un ciclo di **percorso dati;** anche se i computer **RISC** impiegano 4-5 istruzioni per fare ciò che un computer **CISC** esegue con una sola istruzione, comunque risultano più veloci in quanto non interpretate.

Principi di progettazione dei calcolatori moderni

Esiste un insieme di principi di progettazione, chiamati **principi di progettazione RISC** che vengono sempre seguiti:

- **Tutte le istruzioni vengono eseguite direttamente dall'hardware**: saltanto il livello di interpretazione la velocità di esecuzione delle istruzioni aumenta.
- **Massimizzare la frequenza di emissione delle istruzioni**: e questo è possibile solo se sono anche in grado di eseguire più istruzioni contemporaneamente.
- **Le istruzioni devono essere facili da decodificare**: emettendo milioni di istruzioni in un secondo, non risolvo niente se la macchina impiega molto a decodificare ogni singola istruzione.
- **Solo le istruzioni LOAD e STORE fanno riferimento alla memoria**: dato che l'accesso alla memoria può richiedere un tempo considerevole e non prevedibile, le operazioni di memoria devono essere eseguite sovrapposte ad altre istruzioni (che operano sui registri).
- **Disporre di molti registri**: dato che l'accesso alla memoria è lento, occorre avere molti registri (almeno 32) in modo che la parola di memoria possa essere mantenuta in un registro tutto il tempo necessario. Bisogna ridurre al minimo il rischio di restare senza registri liberi.

Parallelismo: più istruzioni nell'unità di tempo

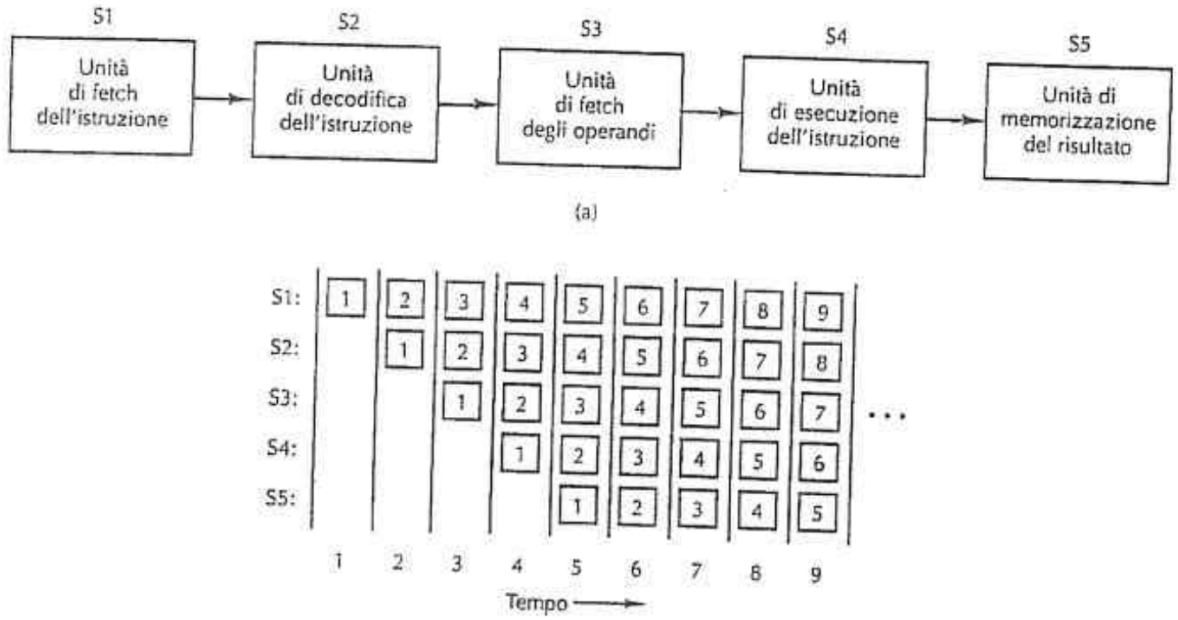
Poiché l'incremento del clock del processore ha raggiunto un limite fisico, i progettisti di CPU guardano al **parallelismo** (**più istruzioni nello stesso tempo**) per incrementare le performance.

Il parallelismo può essere presente in due forme:

- A **livello d'istruzione** in cui il parallelismo viene sfruttato all'interno delle singole istruzioni in modo da elaborarne il più possibile al secondo.
- A **livello di processore** in cui ci sono più CPU che lavorano insieme su uno steso problema.

Pipelining e architetture superscalari (parallelismo a livello d'istruzione)

Uno dei maggiori problemi nella velocità di esecuzione delle istruzioni è rappresentato dal prelievo delle istruzioni dalla memoria. Per aggirare questo problema si fa uso del concetto di **pipelining**; le istruzioni vengono divise in un determinato numero di parti che possono essere eseguite in parallelo (ogni parte viene gestita da una componente hardware dedicata).
Presumiamo di avere una **pipeline** a cinque stadi:



Durante il primo ciclo di clock lo stadio **S1** preleva l'istruzione **1** dalla memoria. Durante il secondo ciclo di clock, lo stadio **S2** decodifica l'istruzione **1** mentre lo stadio **S1** preleva l'istruzione **2** dalla memoria. Nel ciclo di clock successivo, lo stadio **S3** preleva gli operandi per l'istruzione **1**, mentre lo stadio **S2** decodifica l'istruzione **2** e lo stadio **S1** preleva l'istruzione **3**. Successivamente lo stadio **S4** esegue l'istruzione ed infine, al

quinto ciclo di clock, lo stadio **S5** scrive il risultato dell'istruzione.
L'uso della **pipeline** ci consente di bilanciare la **latenza** (il tempo necessario per elaborare un'istruzione) e la **larghezza di banda del processore** (I MIPS della CPU).

- Tuttavia non tutte le istruzioni possono essere eseguite contemporaneamente, perché può succedere che un'istruzione sia dipendente dal risultato dell'altra. Quindi o è il compilatore a gestire questa situazione (l'hardware non effettua nessun controllo e se le istruzioni sono incompatibili restituisce un risultato errato) oppure i conflitti vengono rilevati ed eliminati durante l'esecuzione da componenti hardware ad hoc.

- Anche se le pipeline sono principalmente usate su macchine RISC, Intel inizia ad usare delle pipeline nelle sue CPU a partire dal x486. Il 486 aveva una sola pipeline, mentre il primo Pentium ne aveva due. La pipeline principale, chiamata **pipeline u**, poteva eseguire una qualsiasi istruzione Pentium; la pipeline secondaria **pipeline v**, eseguiva solamente semplici istruzioni su interi.

Con il termine **architettura superscalare** si intende una CPU con singola pipeline ma con più unità funzionali. Le architetture superscalari moderne sono in grado di lanciare più istruzioni durante un solo ciclo di clock (solitamente 4 o 6).

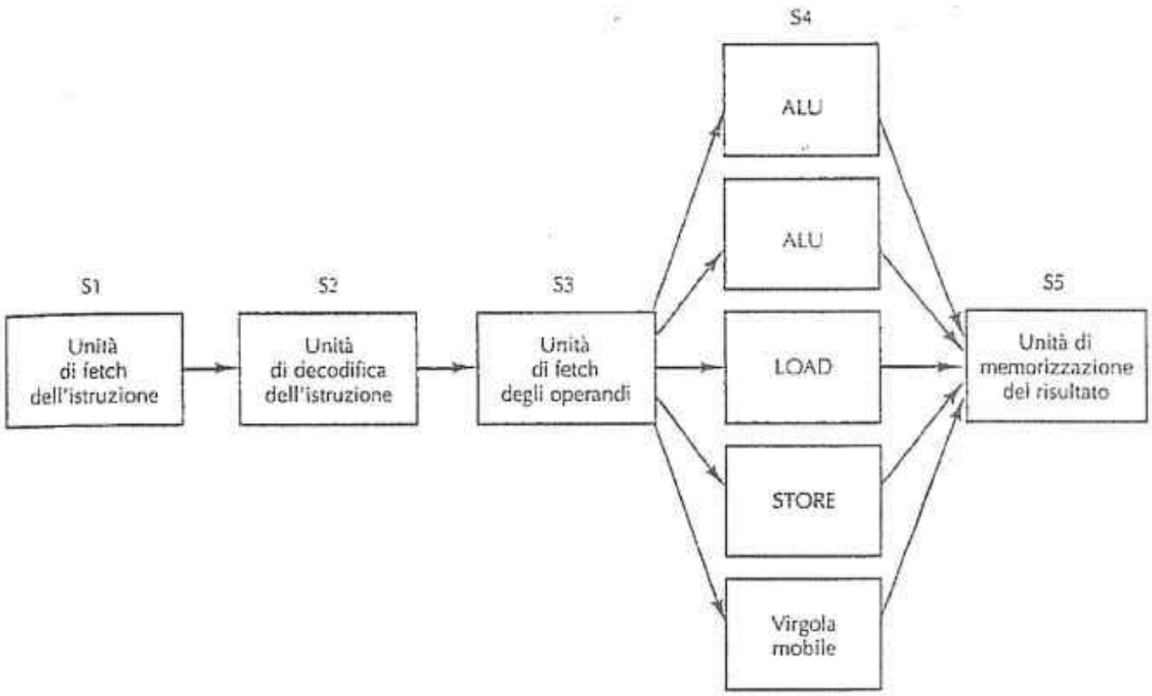


Figura 2.6 Processore superscalare con cinque unità funzionali.

Parallelismo a livello di processore

Per aumentare ancora di più le performance delle macchine occorre progettare sistemi dotati di più CPU. I tre approcci principali a questo tipo di progettazione sono:

- **Computer con parallelismo sui dati**
- **Multiprocessori**
- **Multicomputer**

Computer con parallelismo sui dati

Questo tipo di progettazione risulta molto utile quando abbiamo a che fare con problemi dalla struttura regolare, in cui gli stessi calcoli vengono ripetuti continuamente su insiemi diversi di dati.
Per eseguire questi programmi altamente regolari vengono usati, solitamente, i **processori SIMD**. Questo tipo di processore consiste di un elevato numero di processori identici che eseguono la stessa sequenza di istruzioni su insiemi diversi di dati.
Il modello SIMD è composto da un'unica unità di controllo che esegue una istruzione

alla volta controllando più **ALU** che operano in maniera sincrona. Ad ogni passo, tutti gli elementi eseguono la stessa istruzione scalare, ma ciascuno su un dato differente.

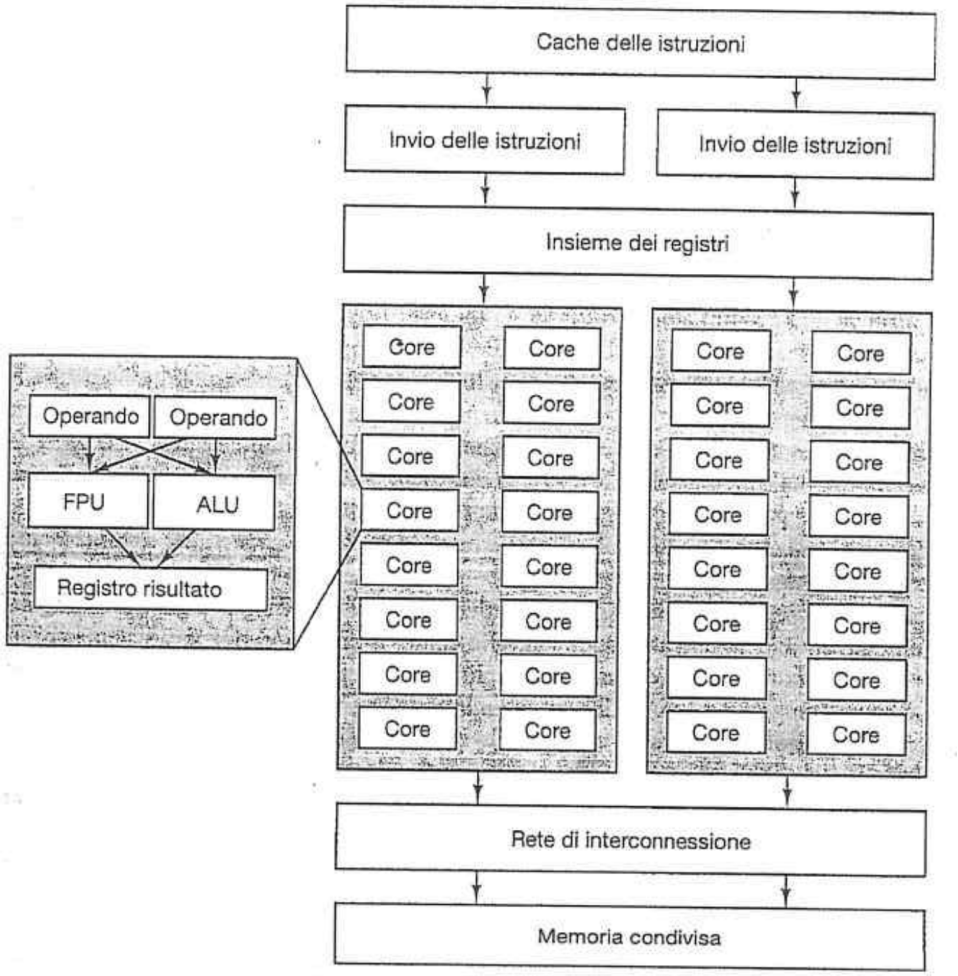


Figura 2.7 Il processore SIMD della GPU Fermi.

- Le moderne **GPU** fanno uso dell'elaborazione SIMD dato che la maggior parte degli algoritmi sono altamente regolari, con operazioni ripetute su pixel, vertici, texture..

- **Thread o processo?** Un **processo** è un programma in **esecuzione**. Un **Thread** è l’insieme di **istruzioni** che caratterizzano il flusso di **esecuzione** di un **processo**.

Multiprocessori

A differenza dei processori SIMD, che hanno un’unica unità di controllo, i **multiprocessori** sono sistemi formati da più CPU distinte, collegate tra loro con un singolo bus, e tutte connesse con una memoria in comune.
Dato che ogni CPU può leggere e scrivere una qualsiasi parte della memoria, esse devono coordinarsi via software per evitare di ostacolarsi. Per evitare problemi, si usa un’architettura in cui ogni processore possiede una memoria locale utilizzabile per contenere il codice del programma e quei dati che non devono essere condivisi.

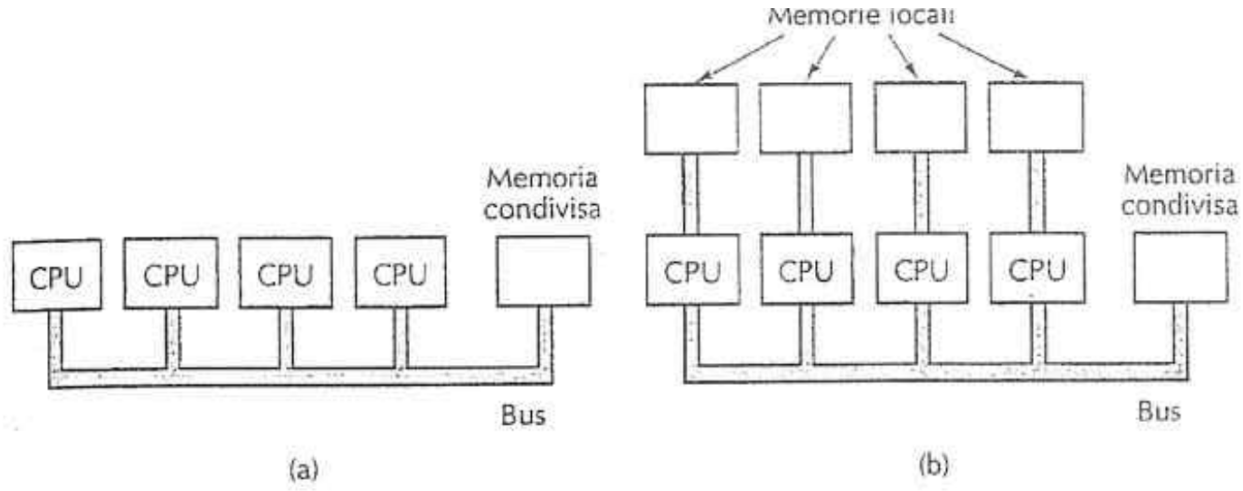


Figura 2.8 (a) Multiprocessore a bus singolo. (b) Multicomputer con memorie locali.

Multicomputer

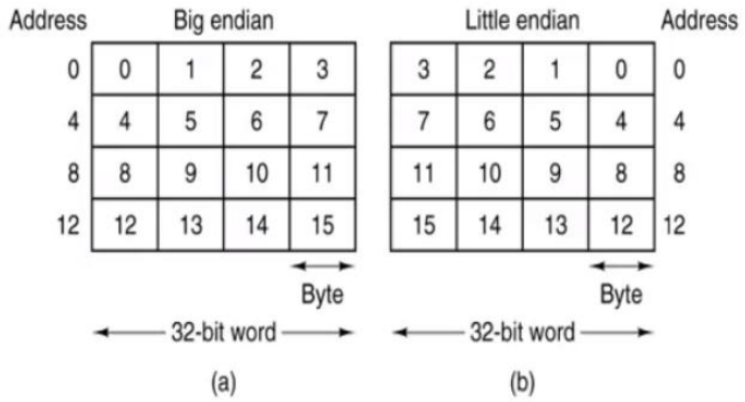
Nei **multicomputer** si abbandona l’idea di avere una memoria comune a favore di un sistema composto da un gran numero di calcolatori interconnessi, ognuno con una memoria privata, che comunicano fra loro inviandosi dei messaggi.

Memoria principale

La **memoria** è quella parte del calcolatore in cui sono depositati programmi e dati. La sua unità base è il **bit**, o unità binaria, che può valere 0 oppure 1. Le informazioni vengono memorizzate in un certo numero di **celle**, o **locazioni**; ogni cella viene indicata con un numero, chiamato **indirizzo**, attraverso il quale può essere richiamata dal programma.

- Se una memoria ha *n* celle, i suoi indirizzi varieranno da 0 a *n* - 1.

Tutte le celle di memoria contengono lo stesso numero di bit, che negli ultimi anni è stato standardizzano ad **8 bit**. Questa dimensione viene denominata **byte** (quindi **1 byte = 8 bit**), ed i **byte** vengono raggruppati in **parole**.
Quindi un calcolatore con parole a 32 bit, avrà 4 byte per parola; un calcolatore con parole a 64 bit, avrà 8 byte per parola.
All'interno di una parola, i **byte** possono essere scritti da destra verso sinistra (*little endian*) o da sinistra verso destra (*big endian*).



Codici di correzione di errore

Al fine di correggere e rilevare eventuali errori commessi dal computer si fa uso del **codice di Hamming**, che va ad aggiungere dei **bit di controllo** in mezzo al **codeword** in base ad un algoritmo prestabilito.

- Per **codeword** (o **parola di codice**) intendiamo una parola, indicata con ***n***, che contiene ***m* bit** di dati **+ *r* bit ridondanti**. (**n = m + r**)

Per di **distanza di Hamming** intendiamo il numero di bit per la quale due parole differiscono.

D(1000,1010) = 1
D(0110,0000) = 2
D(0100,1010) = 3

Ed è importante ricordare che la **distanza minima di Hanning** corrisponde sempre a **K + 1** (K inteso come lunghezza del bit)

Immaginiamo di avere un codice formato da 7 bit, 0000101, di quanti bit di controllo avrò bisogno? Per scoprirlo, ci basta verificare questa disuguaglianza: bit di messaggio + 1 ≤ 2^x - x. In cui ovviamente x = bit di controllo. Nel nostro caso avremo bisogno di 4 bit di controllo, dato che 2^4 = 16 allora: 7 + 1 ≤ 16 - 4 è verificata.
Di conseguenza il nostro messaggio finale sarà composto dal messaggio originale da 8 bit + 4 bit di controllo (12 bit)

Dimensione parola	Bit di controllo	Dimensione totale	Percentuale di overhead
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

Figura 2.13 Numero di bit di controllo per un codice che può correggere errori singoli.

Detto ciò, i bit di controllo vanno inseriti nelle posizioni della potenza del 2; dobbiamo mettere i bit di controllo nelle posizioni di 2^0, $2^1, 2^2, 2^3$.
Quindi, dato 2^0 = 1, noi andiamo ad inserire il primo bit di controllo in posizione 1; 2^1 = 2, quindi secondo bit di controllo in posizione 2; 2^2 = 4, terzo bit di controllo in posizione 4; 2^3 = 8, quarto bit di controllo in posizione 8.
Ipotizziamo di voler trasmettere il messaggio 00101101, Il nostro messaggio di partenza diventerà così: (* = bit di controllo)

bit 1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
*	*	0	*	0	1	0	*	1	1	0	1
pos 1	2	3	4	5	6	7	8	9	10	11	12

Per “trovare” il valore da inserire nei primo bit di controllo, dobbiamo prima capire quali bit controlla ogni bit di controllo, e per questo dobbiamo rifarci sempre alle potenze del 2. Abbiamo detto che il primo bit è 2^0 = 1 e questo vuol dire che il primo bit controlla un bit sì ed un bit no: ovvero controlla i bit 1,3,5,7,9,11.
A seguire, il secondo bit di controllo è 2^1 = 2 e controlla due bit sì e due bit no (da ricordare che il conteggio dei bit parte sempre dal numero del bit stesso, in questo caso partiamo quindi dal bit di controllo in seconda posizione): 2,3,6,7,10,11.

- 2^0 un bit sì e uno no: 1,3,5,7,9,11
- 2^1 due bit sì e due no: 2,3,6,7,10,11
- 2^2 quattro bit sì e quattro no: 4,5,6,7,12
- 2^3 otto bit sì e otto no: 8,9,10,11,12

Ora che abbiamo capito quali bit controlla ogni bit di controllo, possiamo procedere ad inserire il valore corretto. Per fare ciò utilizziamo il **bit di parità**, ovvero andiamo a vedere quanti 1 ci sono in ogni stringa di bit di controllo: se il numero di 1 presenti è pari, inseriamo uno 0, se il numero di 1 è dispari inseriamo un 1 (inseriamo un 1 perché così facendo facciamo tornare la conta dei numeri 1 pari, es: abbiamo 5 volte il valore 1, quindi aggiungiamo il bit di parità 1 per arrivare a 6.)

bit 1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
*	*	0	*	0	1	0	*	1	1	0	1
pos 1	2	3	4	5	6	7	8	9	10	11	12

Rivedendo il codice, procediamo con il calcolo:

- Il primo bit di controllo è formato da: 00010. Il valore 1 compare solo una volta, quindi aggiungiamo il bit di parità 1.

bit 1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
1	*	0	*	0	1	0	*	1	1	0	1
pos 1	2	3	4	5	6	7	8	9	10	11	12

- Il secondo bit di controllo è formato da: 01010. Il valore 1 appare due volte, quindi il bit b2 sarà 0.
- Il terzo bit di controllo è formato da: 0101. Il valore 1 appare due volte, quindi b4 sarà 0.
- il quarto bit di controllo è formato da: 1101. Il valore 1 appare tre volte, quindi b8 sarà 1.

bit 1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
1	0	0	0	0	1	0	1	1	1	0	1
pos 1	2	3	4	5	6	7	8	9	10	11	12