**Architettura degli elaboratori**

Sommario

[Introduzione 1](#_Toc178244100)

[Storia dei calcolatori 2](#_Toc178244101)

[Organizzazione di un calcolatore 4](#_Toc178244102)

[Esempi 7](#_Toc178244103)

[Porte logiche e circuiti combinatori 13](#_Toc178244104)

# Introduzione

Durante il corso di Architettura andremo a studiare i principi alla base del funzionamento dei calcolatori, a partire dalle porte logiche NAND fino alla realizzazione completa (in un simulatore) di un calcolatore in grado di eseguire il videogioco pong.

Utilizzeremo la porta NAND perché essa è in primo luogo universale, ovvero ci permette, combinandola in vari modi con altre porte NAND, di realizzare tutte le logiche booleane (che elaborano due valori: 0,1 - chiamati anche vero e falso); in secondo luogo la porta NAND è facile da realizzare utilizzando i transistor (presenti in tutte le odierne CPU).

Il progetto che andremo a realizzare, pong, è estremamente complesso; al termine esso sarà composto da centinaia, se non migliaia, di porte NAND, quindi per poterlo affrontare seguiremo il principio di *astrazione/implementazione*.

Il principio di astrazione/implementazione è uno dei concetti base dell’informatica. Esso si basa su una semplice operazione: ogni volta che vado a creare un componente che mi serve (implementazione), esso diventa un “mattoncino” che posso poi andare ad utilizzare, senza dovermi ricordare come l’ho costruito.

LIVELLO HARDWARE

Livello 0

È l’unico livello interamente fisico, su cui si trovano le porte logiche (XOR, NOT; AND ecc), i circuiti combinatori, i circuiti sequenziali (hanno una memoria).

Questi sono tutti fatti alla fine da porte NAND

Livello 1: microarchitettura.

Questo livello governa il flusso di dati fra i vari componenti del livello logico digitale, può essere hardware, ma più comunemente è software.

Livello 2: livello ISA

Dà l’insieme di istruzioni eseguibili dalla microarchitettura.

È considerato il livello di interfaccia tra hardware e software

LIVELLO SOFTWARE

Livello3-4: Livello del sistema operativo

linguaggio assemblativo.

È molto semplice è strettamente correlato al processore. Utilizza il linguaggio Assembler, dà istruzioni riguardo a come accedere al processore ecc. Il S.O. aggiunge funzionalità al livello ISA (i programmi), ad esempio i servizi per gestire I/O, la memoria, la scheda di rete, ecc. Il linguaggio Assemblativo permette di programmare i livelli sottostanti (traducendo l’Assembly in linguaggio macchina). Il confine tra S.O. e Assembly è un po’ sfumato, per questo si parla di livello ibrido.

Livello 5: linguaggio di programmazione ad alto livello

Immagine che contiene testo, elettronica, schermata, Carattere

Descrizione generata automaticamente

# Storia dei calcolatori

Sviluppo nelle potenzialità dei calcolatori 🡪 spinto dalla necessità di avere calcolatori più veloci e dalla necessità economica; si voleva rendere il calcolatore più efficiente in rapporto qualità-prezzo. Queste due forze sono tutt’ora al lavoro e rimarranno al lavoro nel futuro. Nel corso della storia sono anche stati sviluppate molte nuove componenti che sono state implementate e li hanno resi più complessi, rendendoli quelli che sono oggi.

**Generazione 0**: computer meccanici, senza elettricità

1642 – PASCALINA: prima macchina calcolatrice per somme e sottrazioni realizzata da Blaise Pascal.

La prima necessità fu far eseguire dei conti matematici alla macchina. Questa macchina faceva solo somme e sottrazioni.

1672 – Leibniz costruì una macchina calcolatrice capace di eseguire anche moltiplicazioni e divisioni.

1834 – Analytical Engine: primo computer progettato da Charles Babbage (la costruzione non venne completata, era troppo complicato da realizzare il progetto):

* Magazzino: memoria con 1000 parole da 50 cifre decimali (ci si rese conto in fretta che erano meglio le binarie)
* Mulino: l’unità computazionale, in grado di eseguire programmi (era un oggetto programmabile)
* Input/Output: tramite schede perforate (tessere con dei buchi)

Il primo programma venne scritto da Ada Lovelace, considerata la prima programmatrice della storia. (In foto il precedente “difference engine”)

1936 – Z1: la prima macchina calcolatrice basata su relè – tasto che si apre o chiude se passa o non passa corrente (elettromeccanici) realizzata da Konrad Zuse.

**Generazione 1: Computer basati su valvole (1943-1955)** 🡪 Le valvole sono simili ai transistor, ma sono oggetti molto più grossi e simili a lampadine. Questi oggetti vanno quindi con la corrente elettrica

1945 – Colossus: il governo britannico realizzò un calcolatore per decodificare i messaggi trasmessi dai tedeschi codificati tramite la macchina Enigma. Alan Turing, uno dei fondatori della moderna “computer science”, lavorò a tale progetto. L’architettura di Colossus rimase sotto segreto militare per più di 30 anni. Questi calcolatori occupavano stanze intere.

1946 – ENIAC (Electronic Numerical Integrator and Computer): costituito da 18000 valvole e 3000 relè. Realizzato negli USA per calcolare le tabelle per il puntamento dell’artiglieria pesante. Pesava 30 tonnellate, era basato su aritmetica decimale, possedeva 20 registri ed era programmabile tramite 6000 interruttori multi-posizione e una moltitudine di cavi. L’input/output era possibile attraverso schede perforate.

1952 – IAS: John von Neumann progetta un calcolatore seguendo una architettura su cui si basa la maggior parte degli elaboratori che verranno realizzati in seguito, inclusi quelli odierni. È fatta da una memoria, e la principale innovazione fu la collocazione del programma in memoria insieme ai dati: se i programmi sono in memoria possiamo elaborare i programmi stessi, quindi possono essere letti e scritti da altri programmi (es. linguaggio assembly che diventa linguaggio macchina).

\*\*\* la memoria RAM contiene i programmi in esecuzione al momento. Ha due difetti: se spengo il pc perde tutto, ed è relativamente piccola: non può starci tutto e sempre. Utilizzando le memorie di massa come l’hard disk i dati rimangono anche quando spengo il pc e sono molto più grandi.

Unità di controllo: prende dalla memoria istruzioni in linguaggio macchina, le interpreta e manda segnali al resto del calcolatore. L’ unità aritmetico-logica fa operazioni aritmetiche e logiche (and, or, not). L’accumulatore è la prima istanza di registro; la memoria è veloce sotto certi standard ma non così tanto veloce; per questo si usano memorie molto piccole dentro la CPU. Ogni registro contiene numeri a 64 bi (o a 32 ad esempio). I registri sono memorie molto piccole e contengono dati dentro la CPU; in questo modo si velocizza il calcolatore. La RAM è un po’ più lenta dei registri; l’hard disk ancora più lento.

Su quanti bit il calcolatore riesce a lavorare in un’unica operazione? Ad esempio i calcolatori odierni lavorano a 64 bit. I computer attuali hanno alcune decine di registri.

Infine, è necessario l’input e l’output. Il monitor è generalmente output ma può essere input se è touch.

**Generazione 2: Computer basati su transistor (1955-1965)** 🡪 a parità di spazio posso fare molte più cose

1961 – PDP-1: prodotto dalla DEC, rappresenta il primo minicalcolatore. Il progetto basava il suo punto di forza non tanto sulla potenza ma sul prezzo. Il PDP-1 era oltre 10 volte meno costoso (120K$) della macchina più potente di quel periodo (IBM 7090, alcuni M$), ma solo due volte più lento. Ne furono vendute decine. Tra le principali innovazioni usava un display grafico 512 x 512 (prima era basato su schede perforate o stampava). Ispirò la realizzazione di uno dei primi videogame della storia: spacewar.

1964 - CDC 6600. Rappresenta la prima macchina parallela della storia, destinata al calcolo. Al suo interno coesistevano diverse unità funzionali preposte a compiti diversi (addizioni, sottrazioni, divisioni, ecc.). Al momento della sua uscita sul mercato CDC 6600 era dieci volte più veloce della macchina più potente di quei tempi. Allo sviluppo di tale calcolatore partecipò Seymour Cray che successivamente contribuì allo sviluppo di vari supercomputer. Nella foto un Cray-1 degli anni ’70.

Diverse parti della macchina erano in grado di svolgere compiti diversi nello stesso tempo; allo stesso tempo per fare due cose allo stesso tempo devo avere abbastanza circuiti e non posso combinare i risultati. In questo periodo nascono degli accorgimenti per ovviare a questi problemi e migliorare le prestazioni.

**Generazione 3: Computer basati su circuiti integrati (1965-1980)**

1964 - IBM 360: prima famiglia di calcolatori, esistevano infatti 4 modelli con prestazioni via via crescenti. Un’innovazione molto importante fu la possibilità di avere più programmi in memoria (multiprogrammazione): mentre si aspettava il completamento dell’input/output, si poteva eseguire un altro programma. Inoltre, la famiglia IBM 360 permetteva l’emulazione dei modelli precedenti IBM tramite microprogrammazione.

**Generazione 4: Computer basati su VLSI (1980- ?)**

VLSI è una sigla che indica: Very Large Scale Integration. Identifica la possibilità di inserire grandi quantità di transistor in un unico chip.

Questa evoluzione ha seguito un andamento conosciuto come “legge di Moore”: la quantità di transistor che può essere inserita in un chip aumenta ogni anno del 60%

1981 – IBM 5150: è il primo personal computer. Raggiungerà velocemente una diffusione enorme e sarà seguito da un’ampia famiglia di processori. IBM decide di pubblicare schemi e dettagli dell’architettura per semplificare il progetto di schede di espansione: questo diede il via all’industria dei cloni compatibili IBM. Anche il sistema operativo MS-DOS (Microsoft Disk Operating System) non venne realizzato da IBM, ma dalla Microsoft Corporation (allora piccolissima software house).

**Generazione 5: Computer ubiqui e computer invisibili**

Tablet, Smartphone, Smartwatch, Smartglasses,… sono strumenti allo stesso tempo “economici” e contenenti architetture di calcolo estremamente potenti e sofisticate. Usati oramai ovunque ed in tutte le attività quotidiane. Questo fenomeno viene chiamato con il nome di “ubiquitous computing”.

Inoltre, esistono una quantità enorme di oggetti che contengono elaboratori (elettrodomestici, giocattoli, carte di credito, …). Tali oggetti sono conosciuti per le loro funzionalità e non per le architetture di calcolo che contengono. Per questo motivo tali dispositivi vengono detti “computer invisibili”.

# Organizzazione di un calcolatore

Un calcolatore è un sistema composto da processori, memorie e dispositivi di input/output (I/O).

Questa organizzazione (con l’unica differenza del bus) è uguale a quella della macchina di Von Neumann, introdotto per gestire meglio il crescente numero di dispositivi I/O che si interfacciano al PC.

Organizzazione “bus oriented”: Un BUS è un insieme di connessioni elettriche parallele utilizzato per trasportare informazioni da un componente all’altro. Generalmente si utilizzano due BUS: un BUS indirizzi, che viene usato dalla CPU per comunicare alla RAM quale indirizzo di memoria leggere, ed un BUS dati, che viene usato per lo scambio di informazioni.

**Architettura di Von Neumann**

* Usa la memoria non solo per i dati ma anche per i programmi. Quando fu proposta era rivoluzionaria: evitava complesse configurazioni con interruttori e cavi
* Programmi e dati sono trasferiti entrambi attraverso il (sotto)bus dati. Il (sotto)bus indirizzi è utilizzato dalla CPU per indicare alla memoria le locazioni delle informazioni da trasferire

**Parti della macchina di Von Neumann**

**La CPU**

Esegue i programmi nella memoria centrale

È composta da:

* Control Unit (CU): legge e interpreta le istruzioni
* ALU: esegue le operazioni (AND,OR, addizione, …)
* Registri: memorizzano i risultati temporanei e le informazioni necessarie al funzionamento

**Alcuni registri speciali**

Abbiamo sia registri generici che registri dedicati; i principali sono

* Program Counter (PC): indica la prossima istruzione in memoria
* Instruction Register (IR): contiene l’istruzione che si sta per eseguire
* Memory Address Register (MAR): indirizzo della cella di memoria da usare nella prossima lettura/scrittura in MDR
* Memory Data Register (MDR): registro che accede al Bus dei Dati (sia in lettura che in scrittura), contiene sia le istruzioni appena lette dalla RAM (linea legge), sia i risultati di istruzioni appena eseguite e che devono essere scritti sulla RAM (linea scrive).
* Program Status Word (PSW): indica informazioni sull’ultima operazione eseguita (zero, overflow, …)

Tipica esecuzione di un’istruzione

1. contenuto di PC posto su MAR e attivazione linea Leggi
2. il contenuto in memoria all’indirizzo indicato da MAR viene scritto su MDR attraverso il bus dati

3. contenuto di MDR copiato su IR e relativa decodifica

4. l'istruzione passa in esecuzione sulla ALU

5. se ci sono operandi da prelevare in memoria, si collocano in registri (usando come sopra MAR e MDR)

6. terminata l'esecuzione il risultato va su registro destinazione; se bisogna scrivere in memoria il valore calcolato si usano MAR / MDR attivando linea Scrivi

7. Si ritorna al punto 1 dopo aver aggiornato il valore di PC

Il ciclo di esecuzione descritto nella slide precedente è conosciuto come ciclo “Fetch – Decode – Execute”:

* Caricamento (Fetch): acquisizione dalla memoria di un’istruzione del programma
* Decodifica (Decode): identificazione del tipo di operazione da eseguire
* Esecuzione (Execute): effettuazione delle operazioni corrispondenti all’istruzione

In particolare: i passi 1-2 corrispondono a Fetch, il passo 3 a Decode, i passi 4-5-6 ad Execute

**CPU**

Come accennato prima, la Control Unit è la parte del processore che detta legge. Tutte le altre parti, quando non stanno eseguendo delle istruzioni, aspettano che la CU le dica cosa fare. Il modo più semplice di realizzare una CU è costruendo un circuito hardware con un insieme fisicamente fissato di istruzioni. Andando a inserire molte istruzioni, però, realizzare l’unità di controllo in questo modo diventerebbe troppo costoso, oltre che complesso; per ovviare al problema, solitamente si permette una (micro)programmazione del comportamento della CPU, si crea cioè il livello della microarchitettura. In questo modo il processore può:

* eseguire più istruzioni a partire dagli stessi componenti di base;
* cambiare il proprio comportamento, cambiando il microprogramma (patch di errori, aggiunta di nuove istruzioni, ecc).

**ALU**

L’Unità Aritmetica e Logica (ALU) è la parte del processore predisposta alle operazioni aritmetiche e logiche. Un processore può avere una o più ALU.

Il Data Path è la parte della CPU che comprende la ALU, i suoi input ed i suoi output (solitamente registri) ed è il percorso che fanno i dati e le istruzioni all’interno del processore.

Il flusso di dati, dai registri alla ALU e viceversa, avviene nell’arco del cosiddetto ciclo di clock, la cui durata è decisa appunto dal clock.

**Clock**

Il clock è un meccanismo che è stato introdotto per ovviare al fatto che i processori non sono macchine perfette.

Ogni operazione si svolge nell’arco di un ciclo di clock (o semplicemente clock). Anche se l’operazione e la scrittura del risultato terminano prima della fine del ciclo, la ALU o gli altri componenti non potranno leggere, scrivere o eseguire altre istruzioni prima del nuovo ciclo di clock.

In un processore ideale, ogni qual volta si passa da una situazione di non passaggio di corrente ad una in cui passa della corrente e viceversa (ricordiamo che il non-passaggio corrisponde a 0 e il passaggio ad 1), l’aumento e il calo di corrente sarebbero netti e i valori salvati (nei registri, oppure nella RAM, oppure in altri posti) potrebbero essere letti i immediatamente.

In un ciclo di clock succede tutto: quando il segnale del clock passa da 0 a 5v (valore a caso per esemplificare) avviene il recupero dalla memoria; questo deve avvenire necessariamente in un ciclo, che è sufficientemente lungo per far avvenire il tutto. In diversi elaboratori gli stessi processi possono richiedere uno o più cicli; tutto è regolato dalla control unit.

La control unit riceve le istruzioni nei fili d’ingresso, le fa passare dalle porte logiche e manda i segnali ai registri. Il segnale 0 non viene considerato; il segnale 1 viene sovrascritto nei registri. Guardando un circuito elettrico, vedo che invia sempre un valore (0 oppure 1).

Quando la corrente varia da 0 a 5 volt, non fa uno stacco netto ma oscilla fino a che non si avvicina maggiormente al valore 1. Il ciclo di clock serve a dare al segnale il tempo di stabilizzarsi, facendo in modo che quando le varie parti della CPU andranno a leggere i dati, leggeranno il valore giusto. Il valore intermedio tra 0 e 1 in genere non viene considerato; va comunque tenuto a mente perché a volte approssimando possono esserci degli errori.

La frequenza del clock è l’inverso del periodo di clock (1/T)

In base alla frequenza, capiamo quanto è veloce il calcolatore, ma non è un valore preciso e utile a confrontare diversi calcolatori perché non dice veloce a fare che cosa; è influente per il confronto solo a parità di architettura.

La velocità del processore dipende quindi in gran parte dalla durata del clock. Un clock solitamente dura pochi microsecondi (se non nanosecondi) e solitamente si parla di frequenza di clock, che indica quanti cicli vengono effettuati in un secondo. La durata del ciclo di clock corrisponde anche alla durata del ciclo di data path.

Dalla durata del data path possiamo risalire alla durata di un istruzione ISA, che è uguale al prodotto di n per la durata del ciclo di path (n \* durata del ciclo di data path), dove n varia da istruzione ad istruzione, a seconda della specifica architettura.

Uno degli aspetti principali dell’evoluzione dei computer è stato l’aumento della velocità di calcolo

Se aumentiamo troppo il ciclo, i circuiti non fanno in tempo a stabilizzarsi prima della prossima istruzione e il calcolatore inizia a mettere valori a caso.

I costruttori fanno una stima, sapendo come è fatto il calcolatore, di quanto ci mettono a stabilizzarsi, e mettono un ciclo di clock leggermente più lungo per compensare le imprecisioni

Overclock 🡪 aumento eccessivo del ciclo di clock. Se rimango nell’intervallo in cui le frequenze sono stabili ho vinto, altrimenti il calcolatore inizia a dare i numeri. Se la CPU non ha impurità particolari sfrutto il margine di tranquillità messo dal costruttore per farlo funzionare; altrimenti devo tornare al clock precedente.

Un modo per migliorare la tecnologia è usare circuiti più veloci; altrimenti, se i circuiti rimangono quelli, posso usare il parallelismo, cioè far eseguire più istruzioni in parallelo al computer.

## Esempi

1.

Registri

ADD %EAX, %EBX %EAX = %%EAX + %EBX

2345: 101110011...111100 🡪 questa è l’istruzione da inviare contenuta nella cella 2345

PC: 2345 (indirizzo della cella di memoria)

PC -> MAR -> address bus

Per eseguire un’istruzione si mette l’impulso sul MAR, poi da lì si attiva il segnale sull’address bus. La memoria mette sul data bus il contenuto della cella (ad es cella 2345)

Dal data bus il segnale arriva al registro MDR. A questo punto la control unit decide cosa fare del registro MDR. La control unit fa in modo che il valore vada dall’MDR all’IR.

IR: 101110011...111100

Questa è la stessa procedura qualunque sia l’istruzione

La control unit dice alla ALU di fare, in questo caso, ADD.

Uso dei multiplexer 🡪 prendono tanti dati in ingresso (ad es a 64 bit) e ne fanno uscire uno solo. La Control Unit manda questi segnali all’ALU. L’ALU e i registri non sanno cosa sta succedendo è il multiplexer che ha 4 ingressi che decide che cosa far passare in base all’istruzione.

4 registri (ognuno a 64 bit)

%EAX -

%EBX - >>>>

%ECX -

%EDX –

L’uscita della ALU va a tutti i registri su cui posso scrivere: come faccio a capire su quale registro devo andare a scrivere? Ogni registro ha un ulteriore filo; solo il filo del registro corretto viene attivato dalla control unit.

Quindi, l’informazione parte dai registri e va alla ALU (che operazione fare) e ai multiplexer (che informazione far passare), e poi la CU decide su quale registro memorizzare l’output.

%EAX (5) + %EBX (3) 🡪 ADD 🡪 8 🡪 viene sovrascritto in %EAX

2.

Registro + valore in memoria del registro

ADD %EAX, [%EBX] %EAX = %EAX + il valore della cella di RAM di indirizzo %EBX

2346: 010110011...1011100

PC -> MAR -> address bus -> MDR -> IR

La IR capisce che deve recuperare il secondo operando dalla memoria.

Manda su MAR il valore di %EBX (locazione 3 in RAM con valore 82) mandando ai multiplexer indicazioni di mandare l’uscita di %EBX su MAR attivando la scrittura su MAR.

Attivo la linea leggi

La memoria si vede arrivare un 3 e manda il valore contenuto nella locazione 3 sul bus dati

82 arriva su MDR

La CU indica ai multiplexer di mandare il valore di %EAX al primo ingresso della ALU

La CU indica ad altri multiplexer di mandare il valore di MDR al secondo ingresso della ALU

La CU indica alla ALU di fare la somma

La CU indica a %EAX di memorizzare il risultato

La PSW azzera il flag di ZERO 🡪 assumiamo che se l’ultima operazione ha dato risultato 0 il bit è 1, se ha dato risultato non zero il bit è 0

La CU dice a PC di prendere PC+1

3.

JZ 123 Jump if zero 123

È un comando che controlla il valore del flag ZERO della PSW. Se l’operazione precedente ha dato come risultato 0, la PSW marca con valore 1; di conseguenza il programma salta a una data posizione di memoria che dipende dall’istruzione successiva. Altrimenti, se l’operazione ha dato un risultato diverso da 0, la marco con 0 (=falso)

La CU controlla il valore del flag ZERO della PSW. La CU vede che il flag è zero, quindi l’operazione non ha dato zero.

2347: 01011100011...100000

PC: 2347

**CISC & RISC**

Alla fine degli anni ‘70, la microprogrammazione permise di realizzare elaboratori con istruzioni molto complesse: CISC - Complex Instruction Set Computer (CPU più complesse)

In contrapposizione a questa tendenza nacque l’idea di realizzare architetture con set di istruzioni più semplici: RISC – Reduced Instruction Set Computer 🡪 architetture molto complesse e grandi possono rimetterci a livello tecnico (ad esempio si surriscalda più facilmente, il sistema è impreciso a causa dell’overclock ecc.)

In CISC dò una singola istruzione molto complessa, in RISC la stessa istruzione è scomposta in più istruzioni più semplici. In RISC è più facile mettere le istruzioni in parallelo e riorganizzarle, perché sono più semplici.

A volte i processori hanno sia parti CISC che RISC per combinare i vantaggi di entrambe.

* Istruzioni più semplici possono essere eseguite più velocemente (con un ciclo di clock ridotto), possibilmente evitando la microprogrammazione
* Ecco perché il livello che abbiamo chiamato “microarchitettura” può essere implementato tramite software (microprogrammazione) oppure hardware

**Pipelining**

Un modo per migliorare le prestazioni di un processore è eseguire contemporaneamente più cicli FDE, usando per ognuno di essi parti diversi della CPU.

Il pipeline è come una catena di montaggio. I pezzi vengono assemblati in momenti diversi lungo la catena di montaggio, e solo alla fine della catena l’output è concluso. Abbiamo quindi non un unico circuito che prende le istruzioni in memoria, decodifica, prende gli operandi, calcola ALU e scrive il risultato; ma abbiamo tanti pezzetti che vanno a creare il risultato finale, e ognuna fa qualcosa.

Immagine che contiene testo, diagramma, Piano, schermata

Descrizione generata automaticamente

La prima unità fa il fetch. La seconda unità fa decode; nel frattempo inizio già a fare il fetch della seconda istruzione perché il primo circuito è libero; quando la prima istruzione è al terzo stadio ho già iniziato la terza istruzione. Se tutto va a regime, in 9 cicli di clock finisco un’istruzione. Il ciclo di clock può essere più veloce di 3 o 4 volte.

Siccome ogni stadio mi fa una sola cosa, anche se vado più veloce posso avere dei pezzi hardware singoli. Ad esempio solo lo stadio S4 richiede la ALU; la CU sta solo in S2; ecc.

Il pipeline si trova in qualunque calcolatore di fascia medio-alta perché va veloce e costa poco. Il pipeline è limitato; non può avere più di sette stadi.

Immagine che contiene testo, schermata, bianco e nero, tabellone

Descrizione generata automaticamente

**Calcolatori multicore** 🡪 hanno più processori, i quali a loro volta hanno più pipeline

Il core contiene quasi tutte le cose che contiene il processore; il core però non contiene la memoria o al massimo una piccola cache. Il multicore ha una memoria condivisa e condivide anche le unità input/output. Ogni core esegue un’istruzione; ad esempio se abbiamo 6 core, fanno 6 istruzioni alla volta.

**Parallelismo**

**Array computer (SIMD)**

Ho una sola Control Unit con più processori ALU che svolgono tutte la stessa funzione, ma con registri diversi (e quindi dati diversi). Ad esempio con un processore del genere posso fare molto velocemente lo schiarimento di un’immagine: posso fare la stessa operazione su ogni singolo pixel che però contengono tutti dati diversi (un colore diverso).

L’altro esempio è la modifica di figure 3D: ogni singolo triangolino che compone la figura può essere modificato con dati diversi. È utile anche per operazioni con i bitcoin o il machine learning.

I calcolatori che vengono venduti normalmente non sono di questo tipo.

La scheda grafica in realtà ha questo tipo di struttura, perché svolge molto bene il tipo di lavoro a livello grafico.

**Multiprocessore (MIMD)**

Alternativamente alla SIMD, se devo eseguire pochi calcoli ma molto complessi, posso replicare tutta la CPU e mettere tutti questi processori in parallelo, ognuno che si occupa del suo set di istruzioni sui suoi dati, permettendomi di fare più operazioni contemporaneamente.

Ogni processore può essere multicore o pipeline. Accedono a una shared memory e collaborano tutti insieme, non eseguono necessariamente le stesse istruzioni.

Posso anche implementare le memorie locali.

Se ho troppe CPU che fanno riferimento a una sola memoria, posso trovarmi di fronte a un collo di bottiglia che rallenta tutto;

Posso quindi implementare un **multicomputer**, la forma più estrema di parallelismo, che fa in modo che moltissime CPU siano collegate.

Più i problemi sono parallelizzabili (= eseguire diverse informazioni in contemporanea che non sono troppo collegate tra di loro) più questo meccanismo è utile.

**Memorie**

Le memorie sono le componenti del calcolatore in grado di memorizzare le informazioni: dati, programmi e risultati indispensabili per il suo funzionamento

Immagine che contiene testo, schermata, design

Descrizione generata automaticamente

Se ho un po’ di registri e un po’ di RAM posso dividere i dati tra le due e in questo modo combino grandi quantità di dati all’interno della RAM con la velocità dei registri. I registri sono più piccolini della RAM ma più veloci.

La cache è una memoria intermedia tra registri e RAM che sta dentro al core. La differenza principale tra cache e registri è che i secondi vengono trattati a livello assembler. Chi programma in assembler deve avere presente quali sono i registri. Il programmatore assembler invece non vede la cache, la quale si ricorda i segnali che ha visto passare; se ha un segnale nuovo lo recupera dalla RAM.

Quando abbiamo dei dati elaborati che vogliamo salvare e mantenere, usiamo i dischi rigidi o ottici. Sono molto lenti, ma sono grandi memorie che funzionano anche offline.

Si usano vari tipi di memoria diversi per scopi diversi:

* Volatile: l’informazione rimane memorizzata fino a che il calcolatore è alimentato
* Persistente: l’informazione rimane memorizzata anche quando il calcolatore non è alimentato (spento)
* On-line: i dati sono sempre accessibili
* Off-line: il supporto deve essere montato per poter accedere ai dati
* Il costo di memorizzazione per byte cresce salendo la piramide
* La capacità (q.tà di byte) cresce scendendo la piramide

Le memorie organizzano i dati in celle. Una cella è una sequenza di bit con un suo specifico indirizzoImmagine che contiene testo, Carattere, schermata, numero

Descrizione generata automaticamente

La grandezza dei registri determina la grandezza dei dati che la CPU può gestire (8bit, 16bit, 32bit, ecc.), questi blocchi di byte vengono detti parole (in inglese word). Esistono 2 modi per memorizzare le “word” in celle di memoria di dimensione standard (1 byte):

* big endian (indirizzi assegnati da sx a dx, byte più significativo a indirizzo più basso)
* litte endian (opposto)

Immagine che contiene testo, schermata, diagramma, numero

Descrizione generata automaticamente

La **memoria cache** è una memoria poco capiente ma molto veloce, che intercetta le chiamate che vanno dal processore alla memoria. Quando c’è una richiesta dal processore alla RAM la cache la intercetta e vede se conosce il contenuto della memoria a cui vuole arrivare il processore. Se la conosce, bene; altrimenti la manda alla RAM e tiene traccia della risposta. In questo caso siamo più lenti, perché abbiamo un tempo più piccolo per verificare se la cache conosce la risposta, e in più dobbiamo arrivare alla RAM; molto spesso la cache riesce comunque a velocizzare il processo.

La maggior parte dei programmi esibiscono il cosiddetto principio di località:

* località temporale: se accediamo a una cella di memoria è molto probabile che nel futuro immediato ci accederemo di nuovo. Gli accessi alla RAM non sono fatti a caso: molto spesso fanno accessi o alla stessa locazione o a locazioni vicine. Quando scrivo un programma in genere lo eseguo in modo sequenziale dall’alto verso il basso. Questo è l’ideale per la cache: la CPU copia in cache anche i blocchi di locazioni contigue, aumentando la probabilità di trovare il dato necessario nei successivi accessi alla cache, evitando quindi di andare a prenderli dalla memoria.

Prestazioni di una cache

* *c* è il tempo di accesso alla memoria cache
* *m* è il tempo di accesso alla memoria centrale
* *h* (hit-ratio) probabilità di successo del recupero dei dati. In genere è un valore tra 0 e 1; più è vicino a 1, più è efficiente.

*c<<m* 🡪 *c* è molto più piccolo di *m*

Tempo medio di accesso alla cache

🡪 caso di successo

🡪 caso di fallimento

Osserviamo che quando h si avvicina ad 1, il tempo medio t si avvicina a c, mentre quando h si avvicina a 0, t si avvicina a

Il termine cache si usa in tanti contesti: esistono altre cache in cui il concetto è simile. Ad esempio, la cache del disco usa meccanismi un po’ diversi rispetto alla cache della RAM. La cache della RAM si pulisce da sola, mentre quella del disco ad esempio ogni tanto può essere pulita dall’utente. Anche online possiamo avere tante cache, ovvero sistemi che intercettano le richieste provando a rispondere più velocemente.

**Dischi magnetici**

Un hard disk (HD) è un dispositivo elettro-meccanico per la conservazione di informazioni sotto forma magnetica.

* Testina 🡪 legge il disco
* Traccia 🡪 sequenza circolare di bit
* Settore 🡪 porzione di traccia che contiene una quantità prefissata di bit (uguale per tutti i settori)

Il driver relativo da un lato comunica con il disco dicendogli cosa fare a livello fisico (gira per 1 secondo e mezzo… ecc), dall’altro parla con il S.O. che si occupa di recuperare i file giusti.

**Memorie allo stato solido (SSD)**

Somigliano alle chiavette USB o alle memorie ROM, ma fanno un lavoro più simile a quello del disco. È un circuito elettrico che contiene file, che ha lo stesso identico impiego dei dischi rigidi. È più resistente agli urti e alle vibrazioni

**Dischi ottici**

Il disco viene letto da un laser che riflette le zone piane e di avvallamenti che codificano zeri ed uni.

* CD-ROM: scritti dal costruttore (Read Only Memory)
* CD-R o CD Registrabili: scrivibili una sola volta
* CD-RW: ri-scrivibili più volte
* DVD: Digital Video Disk o Digital Versatile Disk
* Blu Ray: laser blu e non rosso (diversa frequenza)

Oltre le memorie, i calcolatori includono anche altri componenti: tastiera, mouse, stampanti, schede di rete, monitor (il quale richiede una specifica scheda video che contiene un processore detto GPU (Graphical Processing Unit). Lo schermo è una griglia di punti detti pixel, che possono assumere svariati colori. La GPU permette di evitare di mandare uno alla volta il colore di ogni pixel, ma manda informazioni compresse che velocizzano il processo. Le GPU sono oggetti molto sofisticati e potenti in questo tipo di calcolo.

Dal punto di vista tecnico, abbiamo tanti dispositivi input/output, ognuno con un controller, che gestisce il passaggio dei dati. I dati viaggiando devono andare alla CPU. Posso mandarli direttamente al processore (direct memory access). Per parlare direttamente con la memoria il processore ha bisogno di un BUS, ma anche dalle periferiche per parlare con il processore o dalle periferiche per parlare con la memoria. Il BUS può essere o un circuito a sé stante, oppure integrato nel processore. I controller collegano la periferica al bus. I bus sono insiemi di fili che possono avere tecnologie diverse. Ci sono vari standard per descrivere come sono fatti i BUS a livello fisico.

# Porte logiche e circuiti combinatori

Alla base del funzionamento dei calcolatori ci sono le porte logiche e i circuiti combinatori. A partire dalla porta logica **NAND**, siamo in grado di creare i circuiti logici digitali alla base del funzionamento del computer: le porte NOT, NAND, NOR, AND e OR.

Un circuito combinatorio agisce solo con i valori dati istantaneamente e non conserva memoria (ad es. la ALU). I circuiti sequenziali invece tengono traccia delle operazioni avvenute nel passato (ad. es. le memorie).

Tabelle di verità:

sono uno dei modi possibili per descrivere un circuito. Immagine che contiene testo, diagramma, Piano, linea

Descrizione generata automaticamente

Ho un certo numero di ingressi nelle porte, a seconda del circuito; una colonna per ogni ingresso e una per ogni uscita; e abbiamo una colonna che dà gli output possibile.

Lo standard dice che le righe vanno messe in questo ordine anche per essere sicuri di non scordarsi di ogni possibile combinazione.

**Algebra di Boole**

Abbiamo degli operatori (AND, OR, NOT) , delle costanti (solo 0 e 1) e delle variabili. Mi serve a scrivere relazioni tra le variabili e semplificare espressioni complicate.

OR 🡪 addizione

Ha come risultato 1 se almeno uno dei due parametri è 1.

* 0+0=0;
* 0+1=1;
* 1+0=1;
* 1+1=1

AND 🡪 Moltiplicazione

Ha come risultato 1 solo se entrambi solo 1; altrimenti viene 0

* 0・0=0;
* 0・1=0;
* 1・0=0;
* 1・1=1

NOT 🡪 negazione

Ha come risultato l’opposto del parametro in input

Con quest’algebra possiamo ragionare sui circuiti digitali.

corrisponde al circuito

Immagine che contiene Elementi grafici, Carattere, simbolo, bianco

Descrizione generata automaticamente

OR, AND sono operazioni sia commutative che associative

Se in un circuito ABC faccio l’AND tra AB e poi con C; oppure se lo faccio tra BC e poi con A, i circuiti sono equivalenti.

**Proprietà dell’algebra di Boole**

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

\*\*\*idempotent law 🡪 non ha corrispondenza nell’algebra degli interi.

Queste proprietà permettono di semplificare le forme e di conseguenza semplificare i circuiti.

La legge di De Morgan dice che se abbiamo una negazione di una AND, possiamo negare i singoli elementi e poi fare una OR.

La negazione della OR è come negare i singoli elementi e poi fare una AND

Se ho una negazione in un’espressione complicata fatta di OR ed AND nego i singoli elementi e scambio gli OR con gli AND e viceversa.

Usando De Morgan possiamo

[Negazione applicata ad una AND]

Nego i singoli elementi e metto una OR

[definisco A negato come B]

Sono partito da un’equazione valida, usando le proprietà dell’algebra di Boole ho derivato un’equazione più semplice.

Le proprietà dell’algebra di Boole And e Or sono simmetriche perché grazie a De Morgan posso passare da un lato all’altro

Esercizio. passare da inverse law OR 🡪 AND [ 🡪 ]

[commutativa ]

Dato un circuito, possiamo operarci sopra con l’algebra di Boole, in modo da renderlo più semplice.

L’altro modo per rappresentare i circuiti è tramite le tabelle di verità; è comodo però solo con numeri di input relativamente piccoli

**Tabella di verità**

La tabella di verità ci consente di derivare un circuito in forma canonica.

Un letterale è

* una variabile (ad esempio A)
* oppure una variabile negata (ad esempio )

Un minitermine su n variabili è l’AND fra n letterali corrispondenti alle n variabili (ogni riga della tabella)

Ogni combinazione delle variabili di una funzione booleana ha un corrispondente mintermine (vero per quella specifica combinazione)

La variabile è negata se ho 0 nella tabella, è non negata se ho 1

Alla fine devo fare l’OR dei mintermini.

Immagine che contiene numero, quadrato, testo, Carattere

Descrizione generata automaticamente

La tabella di verità è già data, e i risultati output sono già forniti ma non mi dice come calcolarli. L’operazione interna ad ogni circuito è AND, ma non corrisponde ai risultati forniti nelle tabelle di verità.

Per fare l’OR di questi mintermini, posso considerare solo quelli che in uscita hanno valore 1 e otteniamo una funzione booleana che corrisponde esattamente alla funzione di verità.

* questa funzione corrisponde alla tabella di verità sopra.