1 Introduzione e componenti PC

Un'**algoritmo** è un insieme finito di istruzioni usate per la risoluzione di un certo lavoro. Quest'ultimo molte volte viene definito come una particolare funzione parziale di una macchina di Turing (Alan Turing, matematico britannico) oppure come un programma di una macchina di Von Neumann (matematico statunitense).

Le funzioni di base di un'elaboratore possono essere riassunte in 4 macro aree:

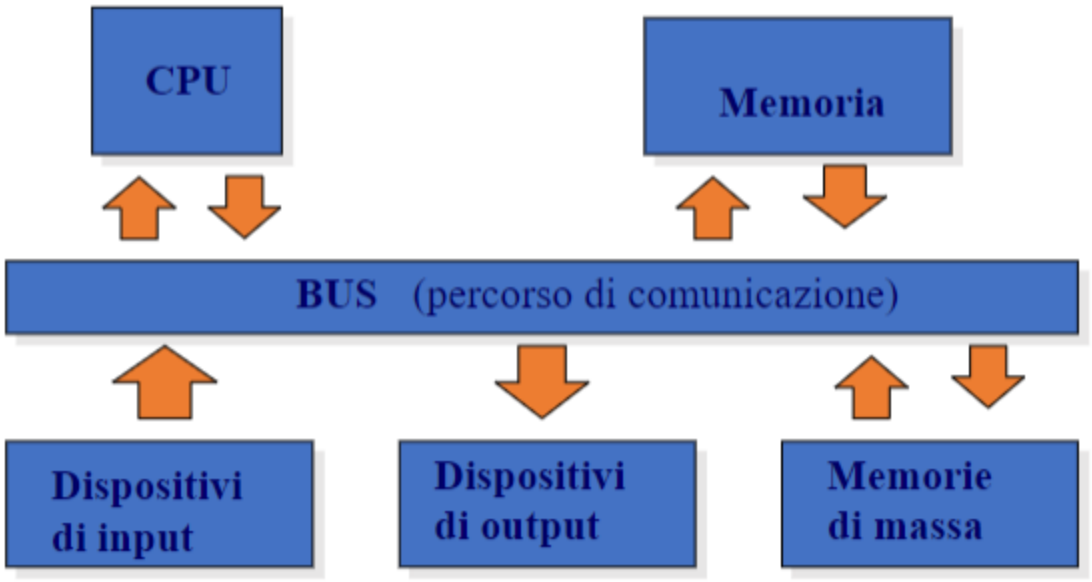
- memorizzazione dei dati
- elaborazione dei dati
- trasferimento dei dati
- controllo.

Un'esempio di elaboratore è il **computer**, il computer è una macchina che computa ovvero che esegue un certo algoritmo (ovviamente scritto in modo che la macchina stessa lo possa interpretare). Esistono vari tipi di computer:

- Laptop
- Server
- Desktop
- Tablet/Smartphone

Un qualsiasi computer moderno segue la seguente architettura:

Macchina di Von Neumann



Questo tipo di architettura viene chiamata Macchina di Von Neumann (questa è l'architettura alla quale fa riferimento la definizione di algoritmo sopracitata). Di seguito una descrizione di tutti i componenti:

- La **CPU**, o **Central Processing Unit**, è l'unità centrale di elaborazione di un computer. È il componente principale che esegue le istruzioni dei programmi, gestisce le operazioni logiche e aritmetiche, e coordina il funzionamento delle altre parti del sistema. Come memoria di lavoro usa i registri e la cache. La sua velocita si misura in numero di cicli al secondo (MHz o GHz). La CPU è formata da 4 parti fondamentali:
  - il **program counter**: una locazione di memoria che contiene l'indirizzo dell'istruzione da eseguire
  - il **registro delle istruzioni**: una locazione di memoria contenente l'istruzione da eseguire
  - **ALU (Arithmetic logic unit)**: un sistema che esegue le operazioni aritmetiche e logiche
  - **CU (Control Unit)**: Il sistema di controllo gestisce il flusso di esecuzione delle istruzioni. Fa sì che ogni parte del processore faccia il suo lavoro nel momento giusto, attraverso una serie di cambiamenti di stato.
- La **memoria**, è un deposito di dati e di istruzioni da eseguire, ne esistono di 3 tipi:
  - **ROM**, o **Read Only Memory**, è una memoria di sola lettura non volatile dove si trovano tutte varie informazioni come le istruzioni usate per l'avvio del pc, o dei parametri necessari per il corretto funzionamento del dispositivo
  - **RAM**, o **Random Access Memory**, è una memoria volatile utilizzata per immagazzinare tutte quelle istruzioni che poi vengono eseguite dal processore

    - La **cache** è una memoria ad alta velocità che si trova all'interno o vicino alla CPU e viene utilizzata per immagazzinare temporaneamente i dati e le istruzioni più frequentemente utilizzate. La sua funzione principale è quella di ridurre i tempi di accesso alla RAM migliorando così le prestazioni del sistema e quindi riducendo il Bottleneck tra CPU e RAM.
    -
- **Dispositivi di input**: come un tastiera ed un mouse
- **Dispositivi di output**: come un monitor o una stampante
- **Memoria di massa**: un tipo di memoria non volatile che viene usata per immagazzinare grandi file, e ovviamente molto più lenta di qualsiasi altro tipo di memoria sopracitata. La memoria di massa è fondamentale per garantire l'archiviazione stabile di software, documenti, immagini e altri dati digitali, rendendoli accessibili nel tempo. Degli esempi sono:
    - **Hard disk drive (HDD)**: basato su dischi magnetici rotanti.
    - **Solid State Drive (SSD)**: usa chip di memoria flash, più veloce e resistente rispetto agli HDD.
- **BUS**, tutte queste componenti comunicano tra di loro attraverso i bus. I principali tipi di bus vengono usati per inviare dati o segnali di controllo

La macchina di von Neumann viene definita logicamente come una terna, ovvero un'insieme di 3 elementi:

- **N** = {0,1,2,3} ovvero l'insieme dei numeri naturali (l'alfabeto della macchina)
- **IS** = {ZERO, INC, SOM, SOT, MOL, DIV, UGUALE, MINORE, SALCOND, ALT} è l'Instruction Set set ovvero un'insieme di istruzioni che la macchina può usare
- **P** = {I0, I1, I2, I3, … , I|P| – 1} è una sequenza finita di istruzioni prese dall'insieme IS, questo insieme si chiama programma
  Come possiamo ben notare un programma eseguibile dalla macchina von Neumann consiste in una lista di istruzioni che devono essere eseguite dal processore. Ogni istruzione viene sottoposta al ciclo macchina ovvero una serie di passaggi impiegati per l'esecuzione dell'istruzione stessa, di seguito i passaggi appena citati:

1. **Legge il contenuto del program counter**: ovvero l'indirizzo della prossima istruzione da eseguire.

2. **Caricamento nel registro delle istruzioni (fetch)**: Il processore va a recuperare l'istruzione dalla memoria, utilizzando l'indirizzo letto da program counter. Questa istruzione viene quindi inserita nel registro delle istruzioni.

3. **Decodifica dell'istruzione**: Una volta che l'istruzione è stata caricata, il processore la decodifica, per capire di che tipo di istruzione si tratta

4. **Invio all'ALU**: Se l'istruzione richiede un'operazione aritmetica o logica, l' unità logico-aritmetica (ALU) riceve l'istruzione e i dati necessari per eseguire l'operazione.

5. **Accesso ai dati**: Se l'istruzione da eseguire richiede dei dati la control unit li recupera dalla memoria. Se l'istruzione è del tipo SOM(M1, M2) ad esempio, significa che M1 e M2 sono indirizzi in memoria da cui vengono presi i dati.

6. **Esecuzione**: L'ALU effettua l'operazione richiesta (ad esempio, somma o confronto) utilizzando i dati forniti.

7. **Memorizzazione del risultato**: Una volta ottenuto il risultato, viene registrato nella locazione di memoria specificata dall'istruzione.

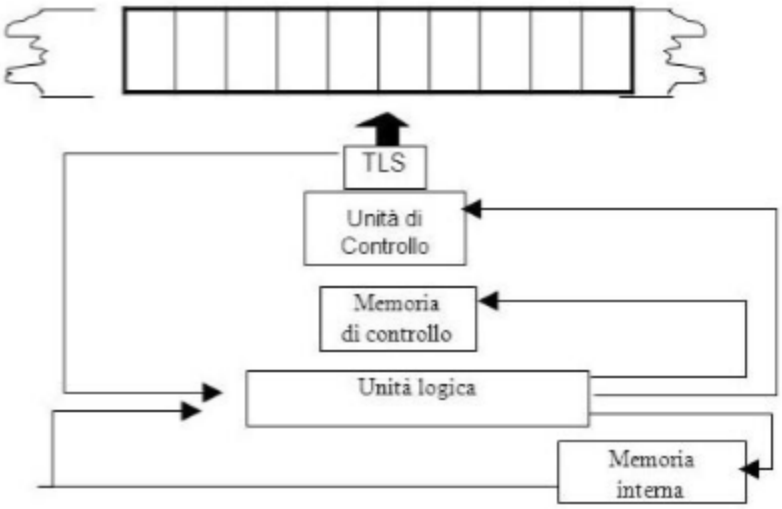
8. **Aggiornamento del contatore**: Il program counter viene incrementato per puntare all'istruzione successiva.

9. **Ripetizione del ciclo**: Questo ciclo continua fino a quando non viene incontrata un'istruzione speciale che ferma o altera l'esecuzione, come un'istruzione ALT o un salto condizionato che modifica il flusso del programma.

Il modello di elaboratore dalla quale Von Neumann prese spunto fu:

Macchina di Turing

Una **macchina di Turing (o MdT)** è stata inventata da Alan Turing nel 1936. Questo modello è fondamentale nella teoria della computabilità e fornisce una rappresentazione astratta di come funzionano i calcolatori. Formalmente viene definita in questo modo:



Di seguito una descrizione di tutti i componenti:

- **Nastro:** Un'unità di memoria esterna virtualmente infinita, suddivisa in celle, ogni cella contiene un simbolo oppure è vuota.
- **Testina di lettura/scrittura (TLS) :** Un dispositivo che interagisce direttamente con il nastro.
- **Unità di memoria interna:** Una struttura che memorizza lo stato interno della macchina.
- **Unità di calcolo:** Un componente che esegue le operazioni di base.
- **Unità di controllo:** Il "cervello" della macchina, che coordina le altre unità.
- **Unità di logica:** Un componente che si occupa delle operazioni logiche.
  Il comportamento di una MdT può essere programmata definendo un'insieme di regole, o quintuple di questo tipo:
- (stato-interno-corrente, simbolo-letto, prossimo-stato-interno, simbolo-scritto, direzione)
  di seguito degli esempi:
- (0, A, 1, B, -) se la macchina si trova nello stato 0 e legge il simbolo A passa allo stato 1 e scrive sul nastro B e sta ferma
- (1, B, 0, A, >) se si trova nello stato 1 e legge il simbolo B passa allo stato 0 e scrive sul nastro A e si muove di una posizione a destra

È importante sottolineare come l'attenzione di Turing sia rivolta al processo di calcolo, **indipendentemente da come esso avviene fisicamente**. Una M.d.T è un dispositivo ideale, cioè indipendente da ogni sua possibile realizzazione fisica.

Una funzione (**parziale**)si dice **Tuing-computabile** se almeno una MdT è in grado di computarla con un numero finito di passi.

FUNZIONE PARZIALE
Una funzione parziale è una funzione che dato un input non necessariamente restituirà un output definito, infatti potrebbe non terminare (loop) o non essere definita
Le funzioni parziali sono importanti perché rappresentano la **computabilità reale** ovvero non tutte le funzioni che possiamo eseguire sono calcolabili in tempo finito.
Quindi le funzioni parziali sono in grado di definire sia le funzioni che terminano con successo (totali) sia quelle che non lo fanno.

Condizioni di finitezza

Una MdT per essere tale deve rispettare le condizioni di finitezza che sono:

- il **numero di simboli** che usa deve essere fissato e **finito**
- il **numero di caselle** del nastro **osservabili in una volta è finito**
- è possibile **ricordare solo un numero finito di stadi precedenti**

- le operazioni che può compiere sono:
    1. Cambiare il contenuto di alcune caselle osservate
    2. Cambiare le caselle osservate
    3. Cambiare il proprio stato
    4. Osservare nuove caselle che si trovano ad una distanza prefissata dalla casella osservata

Macchina di Turing universale

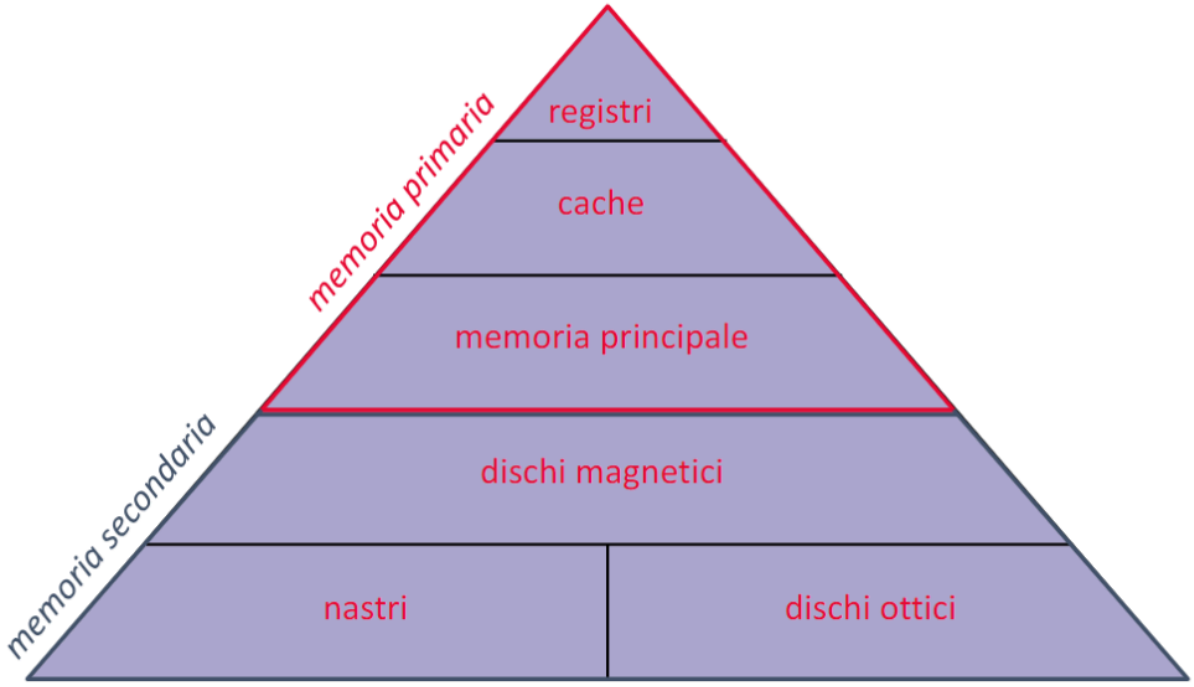
Se supponiamo di avere una macchina di Turing senza limiti di spazio, di tempo e che non possa commettere errori quest'ultima sarà in grado di calcolare tutte le funzioni calcolabili in ogni singola macchina di Turing, questa macchina la chiamiamo **Macchina di Turing Universale (MdTU)**, inoltre deve rispettare sia la condizione di finitezza sopracitata ma anche la condizione di determinatezza spiegata di seguito

Condizione di determinatezza

le azioni di una MdTU devono dipendere solo dal simbolo contenuto nella casella osservata in quell'instante e dallo "Stato mentale" corrente, cioè da quello che ricorda dei calcoli precedenti

Da tutto questo Turing formula la seguente Tesi: "*ogni funzione parziale calcolabile con un algoritmo è una funzione parziale calcolabile da una macchina di Turing*."
(appunto per questo all'inizio si parla dell'algoritmo come una funzione parziale della macchina di Turing). Questa tesi ci dice che **ogni funzione calcolabile da un algoritmo può essere calcolata da una macchina di Turing**, poiché ogni algoritmo può essere descritto in termini di un insieme finito di regole.

MEMORIA



• La memoria, detta normalmente memoria principale (per distinguerla da altri tipi di memorie dette secondarie), è un contenitore di celle ordinato.

• Nelle celle di memoria vengono immesse o prelevate le istruzioni del software e i dati di input e di output. Ogni cella è ampia un byte e ogni cella possiede un indirizzo (address).

• Gli indirizzi delle celle partono da zero e l'indirizzo dell'ultima cella coincide con il numero totale di celle della memoria (meno uno, dato che gli indirizzi partono da zero). l'insieme di tutte le celle di una memoria è detto spazio degli indirizzi o spazio di indirizzamento della memoria.

(lo spazio di indirizzamento dipende dall' ABUS vedi capitolo BUS)

La RAM (Random Access Memory) per non perdere il suo contenuto quando il pc è acceso, viene "refreshata" con un segnale elettrico con frequenza costante.

le celle di memoria sono dei micro condensatori infatti si chiama DRAM (Dynamic RAM)

Per la fase di **BOOTSTRAP** (avvio) il computer ha bisogno di caricare dei dati per il corretto avviamento e questi dati sono contenuti nel **BIOS** che è una **ROM** (Read Only Memory) ovvero una memoria di sola lettura che non perde i dati allo spegnimento del computer. I programmi contenuti nelle ROM sono detti Firmware.

Esiste una tipologia di RAM **più veloce** della DRAM (questa è la RAM normale,8,16,32 GB ecc..) **chiamata SRAM (Static RAM)** che è molto più veloce ma anche più costosa, i

micro condensatori vengono sostituiti dai micro flip-flop. **Queste sono le MEMORIE CACHE**

Comparing memory types

	SRAM	DRAM	NAND FLASH	NOR FLASH
NON-VOLATILE	No	No	Yes	Yes
PRICE PER GB	High	Low	Very low	Low
READ SPEED	Very fast	Fast	Slow	Fast
WRITE SPEED	Very fast	Fast	Slow	Slow
SMALLEST WRITE	Byte	Byte	Page	Byte
SMALLEST READ	Byte	Page	Page	Byte
POWER	High	High	Medium	Medium

UNDESIRABLE/LEAST DESIRABLE MIDDLE MOST DESIRABLE

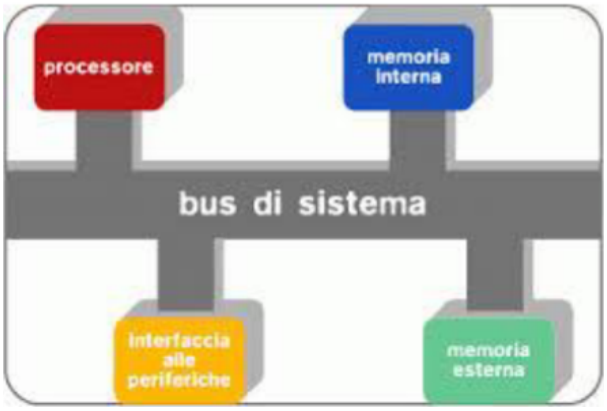
```
                                                    (La NOR FLASH è la ROM del
BIOS)
```

BUS

Il bus è l'unità di interconnessione tra i moduli del modello di von Neumann.

Esso si presenta come un fascio ordinato di linee, ognuna delle quali **può assumere il significato di un bit**, cioè di un valore binario.

Si dice che i moduli processore, memoria e input/output si «affacciano» sul bus, ovvero essendovi collegati, possono impostare, prelevare o modificare i valori presenti sulle linee che lo compongono.



molta dell'attività di un calcolatore si basa sul trasferimento dati attraverso i vari componenti attraverso i bus, **questo modello è di tipo master/slave** (processore -

master , memoria, I/O - slave)

Un operazione che **trasporta bit dalla CPU alla memoria (o I/O) si dice di WRITE** mentre se i bit vanno **verso la CPU** si dice che è un operazione di **READ**

il BUS è scomponibile in **3 sottoinsiemi (LINEE)** -AddressBUS, DataBUS, ControlBUS

1. CBUS ha una linea che specifica(controlla) la direzione (CPU--I/O oppure I/O CPU) ha un altra linea per specificare il verso del trasferimento (WRITE/READ)
2. DBUS da qui passano i dati da trasferire.
3. ABUS : viene usato dalla CPU per trasmettere gli indirizzi di memoria ad altri componenti (RAM)

le **CPU vanno a diversi GHz mentre i bus al massimo a alcune centinaia di MHz** (bottleneck)

- la linea **WAIT** del CBUS indica il trasferimento completato (1) o in corso (0), infatti il trasferimento dei dati da CPU a memoria è più lento dei tempi di operazione della CPU.

LINEA CBUS	VALORE	SIGNIFICATO
WAIT	1	trasferimento completato
WAIT	0	trasferimento in corso
I/O-MEM	1	trasferimento CPU-I/O
I/O-MEM	0	trasferimento CPU-MEM
READ/WRITE	1	lettura
READ/WRITE	0	scrittura

La quantità di linee di ABUS e DBUS dipendono (anche se non sempre) dall'architettura della **CPU (64 bit 64 linee, 32 bit 32 linee)**
la **dimensione dell' ABUS specifica la quantità di memoria raggiungibile dai programmi.** 2 elevato al numero di linee dell' ABUS quindi un' architettura a 32 bit = 2^{32} indirizzi (4 GB di RAM al massimo ciascun processo) 64 bit = 2^{64}

la dimensione del **DBUS** rappresenta il grado di parallelismo del processore ovvero la massima quantità di dati che è in grado di elaborare in un solo trasferimento di bus

(domanda d' esame)** se ho 4 processi e un architettura a 32 bit (32 linee di ABUS) ognuno dei 4 processi può indirizzare 4 gb di RAM?
si 4 gb massimo (2^32) ogni processo

Il chipset è il termine complessivo per descrivere tutte le linee e i bus di sistema. Il chipset è formato da northBridge (dedicato alla connessione memoria-CPU)e southBridge (dedicato alla connessione CPU-I/O)

I/O

La sezione di input/output di un calcolatore è dedicata all'acquisizione dei dati e dei programmi, ed alla rappresentazione degli stessi in varie forme, dal video alla stampa a valori memorizzati su memorie secondarie (per esempio tutti i tipi di memorie di massa, dal DVD all'hard disk al pendrive).

Concettualmente la sezione di **I/O è ancora rappresentabile come un contenitore di celle del tutto analogo alla memoria,** anche se dotato di uno spazio di indirizzamento (spazio degli indirizzi di I/O molto più ridotto.

Ogni dispositivo periferico, di input o di output, possiede un proprio range di indirizzi di I/0 riservato (ogni tanto se hanno bisogno di tanti indirizzi possono usare la memoria normale), detti anche registri di I/O o porte di I/O all'interno dello spazio di indirizzamento di I/O.

La sezione di I/O dispone tuttavia di almeno un'altra modalità fondamentale per consentire la gestione delle attività di I/O con le attività generali del bus, **ovvero linee di sincronizzazione dedicate denominate linee di interruzione (Interrupt).**

il **CBUS** ha 2 segnali da mandare (INTR, INTA) per gestire gli interrupt che esprimono la richiesta di interruzione e il suo completamento.

Con un segnale di interruzione, il dispositivo periferico chiede al processore di sospendere temporaneamente la sua esecuzione per eseguire una parte di programma che lo riguarda, sotto forma di procedura associata a quella interruzione (ISR, Interrupt Service Routine).
• Questo è molto importante per quell'I/0 denominato **asincrono, ovvero che può intervenire senza preavviso e in ogni momento** (per esempio il movimento del mouse).

In realtà i calcolatori prevedono anche speciali modalità di trasferimento di I/O che evitano di occupare il processore, veicolando i valori di I/O direttamente verso (e dalla) memoria, mediante tecniche denominate DMA Direct Memory Access) o di bus Mastering. In questi casi ampie quantità di informazioni vengono spostate sul bus senza impegnare il processore.
La circuiteria di una periferica, dedicata ad affacciarsi sul bus di sistema, a rendere disponibili i propri indirizzi di 1/0, a sincronizzarsi con i trasferimenti tramite il CBus e a condividere le proprie linee dedicate di interruzione e/o DMA, è detta **scheda controller della periferica.**

Molti dispositivi di I/O sono interni al calcolatore, cioè hanno la scheda controller integrata nel Chipset della scheda madre (per esempio tastiera, scheda di rete e scheda video).
Anche speciali elementi di I/0 dedicati al controllo dei trasferimenti di I/O sono integrati nel Chipset, come ad esempio il dispositivo di controllo delle interruzioni (Interrupt Controller) e di gestione del DMA (DMA Controller)

Un tempo indirizzi canali e linee dovevano essere impostati manualmente dall'utente o da un tecnico, ma con le nuove tecnologie **bus Plug&Play**, BIOS, sistema operativo e firmware si mettono d'accordo all'avvio del PC o quando viene montato un nuovo dispositivo e gestiscono tutto in automatico.

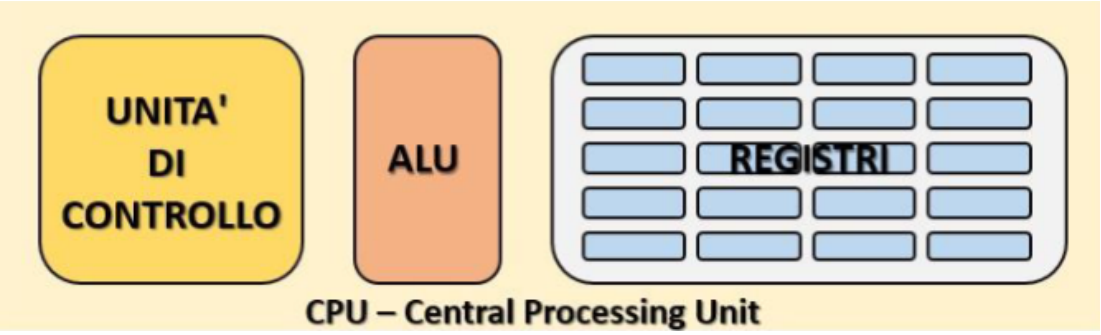
Tra gli standard di connessione più diffusi ricordiamo lo standard USB (Universal Serial Bus), il Firewire (o IEEE 1394) e l'Ethernet 802.x.

PROCESSORE

Un processore è un singolo circuito integrato in grado di effettuare operazioni decisionali, di calcolo o di elaborazione dell'informazione; spesso il processore è indicato con la sigla CPU (Central Processor Unit).

Il processore può essere **suddiviso in tre unità funzionali**, la **CU** (unità di controllo), l'**area dei registri** e l'**ALU** (unità aritmetico-logica).
(Nella realtà ci sono più componenti come l FPU Float Point Unit, per il calcoli in virgola mobile che sono trattati diversamente dai numeri naturali, e ci sono anche varie unità di calcolo per istruzioni più complesse).

La CU si affaccia sul bus, lo arbitra impostando i valori sulle linee ABus, DBus e CBus, legge il DBus e il CBus, legge dalla memoria (e dall'I/O) i dati o li aggiorna in memoria (o nell'I/O) dopo aver compiuto operazioni



I registri contengono i dati letti dalla CU sul bus per predisporli all'esecuzione delle istruzioni che avverranno nell'ALU**; oppure contengono i risultati delle operazioni compiute dall'ALU** in attesa di essere passati alla CU e quindi sul bus.

L'ALU è l'unità di esecuzione effettiva del processore, all'interno della quale si trovano microprogrammi, scritti nel cosiddetto microcodice con relative microistruzioni.

Instruction set e funzionamento operazioni

Ogni processore viene progettato con un set di istruzioni specifico denominato ISA (Instruction Set Architecture o Instruction Set), in corrispondenza di ognuna delle quali è implementato un preciso microprogramma.

• **Ogni istruzione dell'ISA è contraddistinta da un numero specifico, denominato Op. Code** (**Operation Code**, contraddistingue in **modo univoco ogni IS**) e ogni istruzione dotata di Op.Code necessita di un numero preciso e definito di parametri (operandi) che, assieme all'Op. Code, determinano la lunghezza dell'istruzione (in byte).

• Ad ogni Op. Code si associa anche una breve descrizione in lingua naturale che ne ricorda la funzione, detta codice mnemonico. **Un registro speciale del processore, detto PC (Program Counter), si incrementa automaticamente della lunghezza dell'istruzione appena eseguita.**

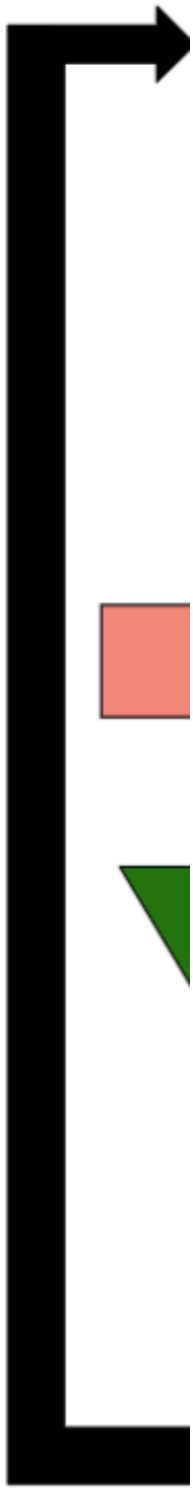
Il data path

L'ALU, i registri e molti bus costituiscono il data path.

L'ALU puo' avere 2 registri di input e 1 di output.

I registri sono collegati ai registri di input.

Il registro di output e' collegato ai registri.

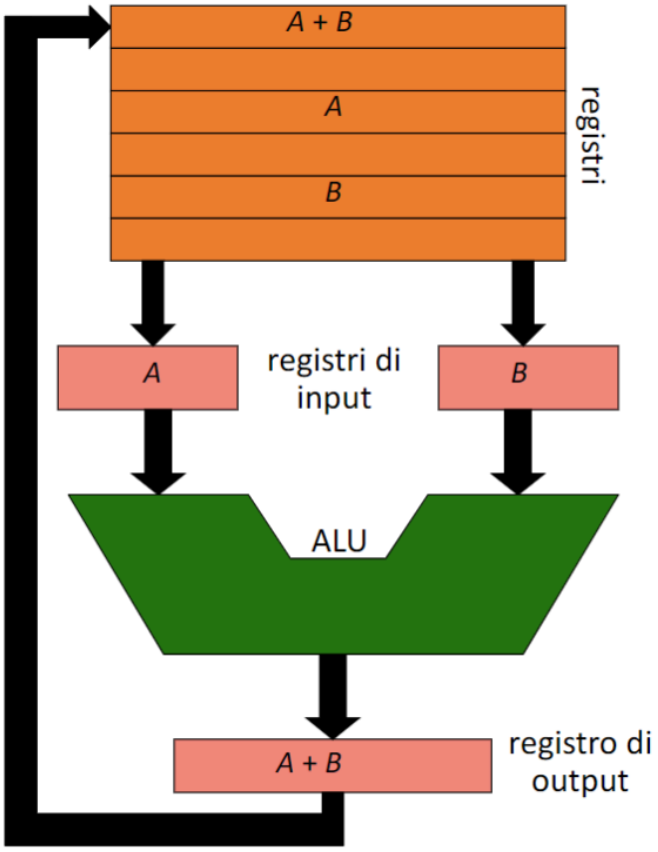


Le istruzioni

registro-memoria: trasferisce un dato dalla memoria (esterna) ai registri, o viceversa

registro-registro: trasferisce un dato da un registro a un altro

Ciclo del data path: passaggio di due operandi attraverso la ALU e memorizzazione del risultato nei registri.



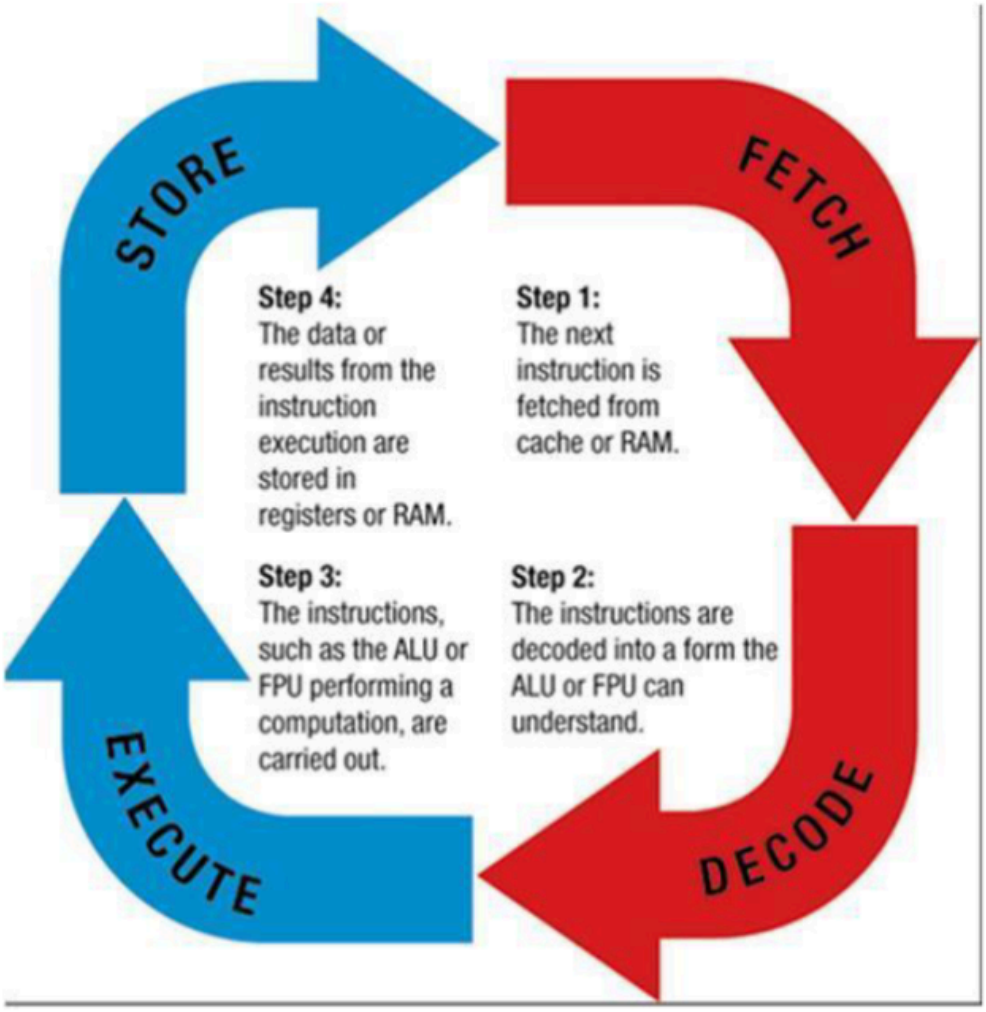
- **Esistevano le architetture CISC** (Complex Instruction Set Computer), sostenute dai colossi IBM, VAX, Intel, … **basate su molte e complicate istruzioni, interpretate**
- **Nascono negli anni '80 le architetture RISC** (Reduced Instruction Set Computer), come il DEC Alpha, **basate su poche e semplici istruzioni, non interpretate**
  Le architetture moderne sono ibride.

(Studia bene Fetch-decode-excute-store)

Fetch-Decode-Execute-Store

Il processore tipico quindi agisce secondo una rigida sequenza di passi che si ripetono fino all'arresto della macchina:

- **Fetch**: l'UC pone sull'ABus il valore del registro Program Counter, imposta lettura da memoria e carica l'Op. Code ivi memorizzato
- **Decode**: l'UC, a partire dall'Op.Code appena letto, determina la lunghezza dell'istruzione, cioè la quantità di parametri di cui necessita, quindi attiva una fase intermedia di caricamento degli operandi (Operand Fetch) che si trovano necessariamente e in modo ordinato agli indirizzi adiacenti a quello dell'Op. Code. Gli operandi caricati andranno a depositarsi nei registri.
- **Execute**: viene avviato il microprogramma relativo all'Op. Code attuale, che usa i propri parametri correttamente memorizzati nei registri. La frequenza in base alla quale vengono eseguiti i microprogrammi è regolata dal clock di CPU (frequenza del microprocessore).
- **Store**: al termine della fase di Execute gli eventuali risultati, posti nei registri, vengono scritti attraverso il bus dall'UC, o verso la memoria, o verso IO.



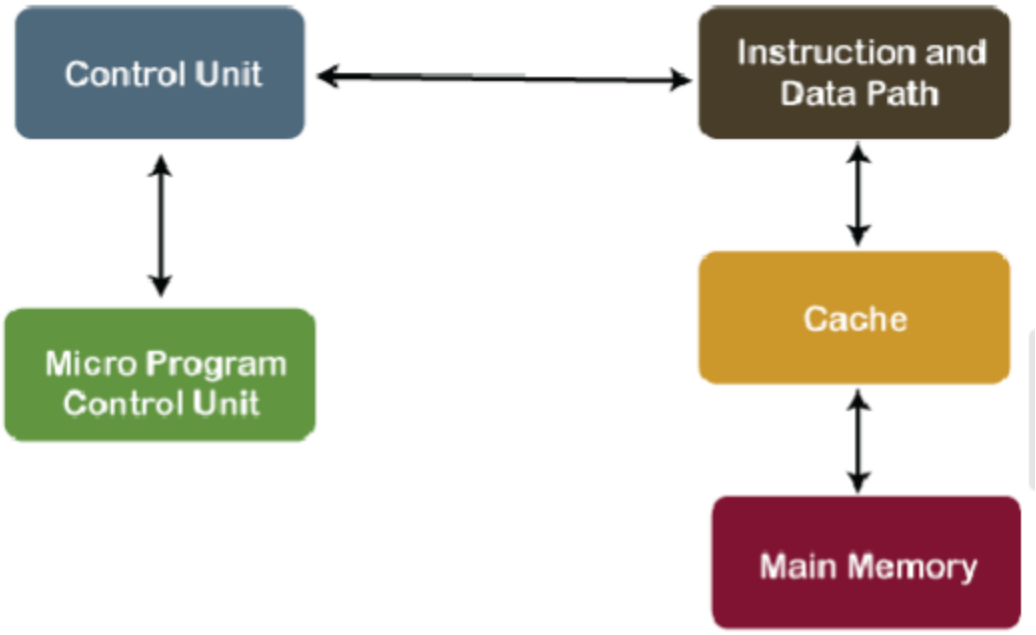
Questo ciclo può interrompersi, infatti il segnale **INTR** sospende il ciclo CPU ed è un interrupt per una periferica (la CPU si dedica alle richieste della periferica).

Ogni **IS è caratterizzata da un proprio Op. code, una determinata quantità di operandi e un preciso numero di cicli di bus** per il suo completamento (compresi tra il fetch, il decode e lo store). Questo ne determina il tempo di esecuzione di ogni microprogramma, anche se influisce poco dato che ogni ciclo CPU è almeno 10 volte più veloce di un ciclo di bus.
Questo ne determina un bottleneck di sistema.

(Studia bene Cisc e Risc)

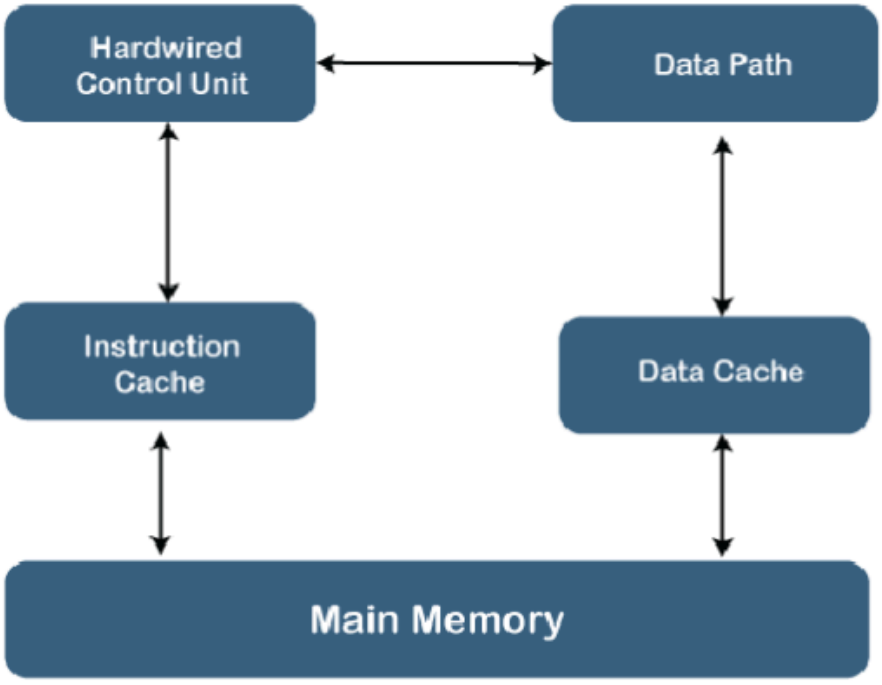
CISC (Complex Instruction Set Code).
Queste architetture utilizzano IS molto complessi e quindi per ogni istruzione fanno più cicli CPU.
Le architetture CISC possiedono un set di istruzioni numeroso; le ampiezze delle istruzioni sono molto variabili.
Si tratta di architetture che facilitano la portabilità del software, dato che l'insieme dei

microprogrammi (o interprete del processore) può essere trasportato su processori più recenti, e quindi sono processori adatti per essere programmati anche in Assembly.



CISC Architecture

RISC (Reduced Instruction Set Code)
Il set di istruzioni di una architettura RISC è limitato, contiene istruzioni di lunghezza costante (con un numero di operandi fisso), con fase di decode breve e senza microprogrammi da eseguire nel processore: ogni istruzione è eseguita direttamente in hardware con pochi cicli di clock.
In questo modo una elaborazione RISC appare nettamente più veloce (almeno di un ordine 10).



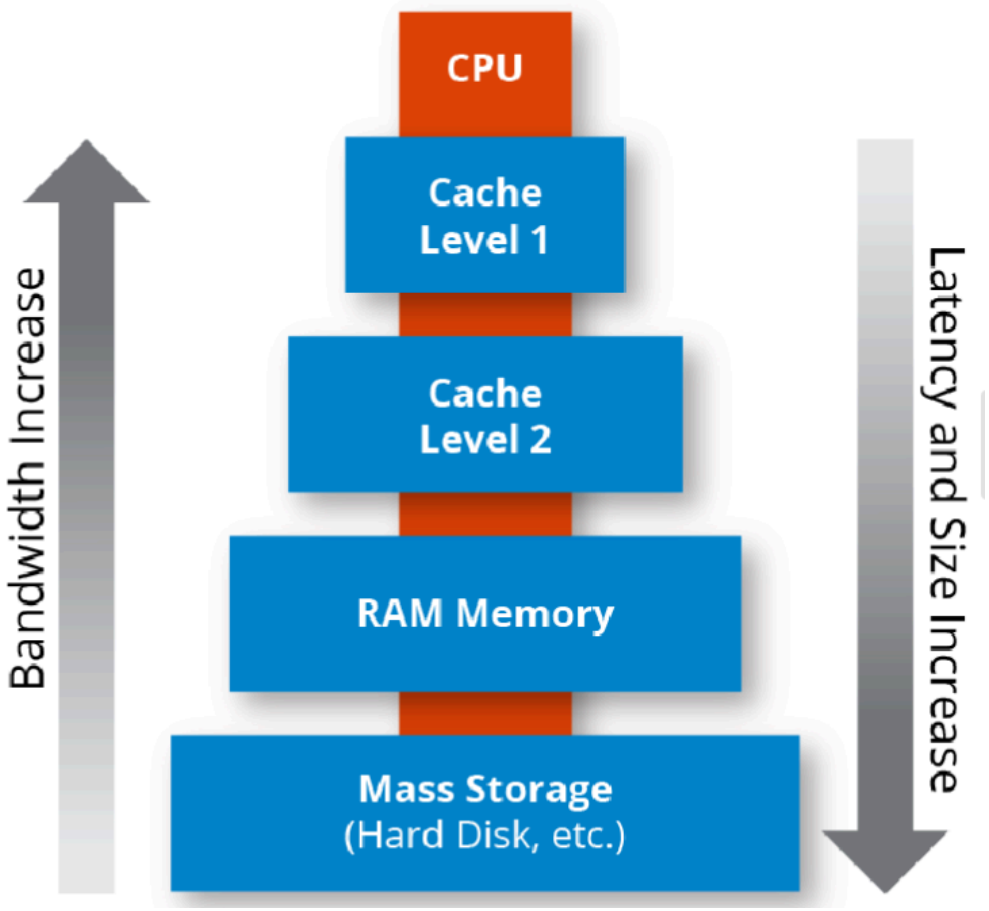
RISC Architecture

In sostanza un'istruzione CISC - con molti passi nel data path - equivale a numerose istruzioni RISC con data path singolo.

Architettura	Vantaggi	Svantaggi	Uso
CISC	Portabilità dei programmi, Facilità di programmazione in Assembly	Tempi di esecuzione lenti	Si utilizzavano nei vecchi Personal computer
RISC	Tempi di esecuzione veloci	Non Portabilità dei programmi, Difficoltà di programmazione in Assembly	Si utilizzavano nei vecchi server

Nessuna delle due è ideale.
Oggi i processori sono Ibridi, ovvero basati su CISC ma con sottosistemi RISC, cercando di ottenere i vantaggi di uno e dell'altro.
Queste architetture moderne si chiamano **CRISC** (Complex Reduced Instruction Set Code).

CACHE
Un modo ingegnoso per diminuire gli accessi al bus e alla memoria, è quello di dotare il calcolatore di una memoria «tampone» (Cache Memory) tra il processore e il bus.
in parole povere serve per non passare dal bus e rimanere dentro la CPU evitando il battleneck.



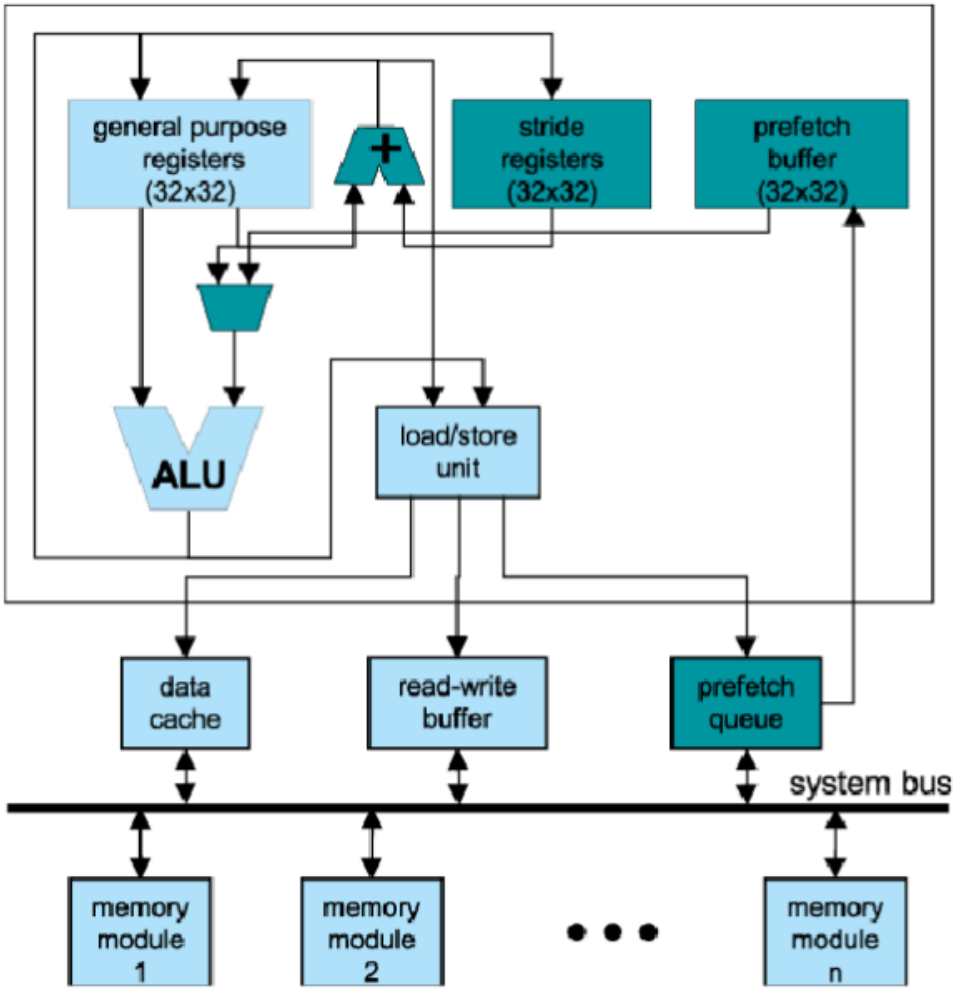
La cache è una memoria di tipo SRAM (static ram) ed è quindi velocissima, ma non molto capiente (i migliori processori del 2024 hanno al massimo 32 mega di cache).

Man mano che il processore legge dalla memoria, ad un **determinato indirizzo, molte locazioni di memoria con indirizzi prossimi a quello, vengono spostati nella memoria cache (nello stesso tempo di bus). Queste sono le linee di cache**

Quindi, **ad ogni operazione di lettura** del processore, l'informazione viene cercata prima di tutto nella cache; se è presente (hit), non è necessario accedere al bus; se non è presente (miss) si accede alla memoria e, oltre all'informazione richiesta, si carica una nuova linea in cache, sovrascrivendo la linea di cache meno usata di recente.
Esistono **3 livelli di cache**
• Livello 1, all'interno del processore; (man mano che i livelli crescono aumenta lo
• Livello 2, collegato al processore; spazio nella cache ma diminuisce la velocità)
• Livello 3 sulla motherboard (anni 2000), oggi nel processore.

PREFETCH

Un modo per aumentare il parallelismo d'esecuzione, fu quello di caricare nel processore più istruzioni oltre a quella richiesta.
Fin dagli esordi, per esempio, i processori erano dotati di una coda di Prefetch, ovvero di un buffer interno in cui il processore memorizzava i successivi 6 o 8 byte consecutivi a quello appena letto dalla memoria. In questo modo, con un solo accesso alla memoria, si aveva a disposizione una serie di valori che potevano essere usati successivamente (come istruzioni ed operandi) senza dover accedere di nuovo al bus.



Esistono **4 tipi di prefetch**

1. **PREFETCHING HARDWARE**: le CPU hanno un componente HW al loro interno che rileva automaticamente i data pattern (dati che servono molte volte e seguono quindi una sorta di pattern nel loro utilizzo) nella memoria e li carica preventivamente nella cache.
2. **PREFETCHING SOFTWARE**: nel prefetching software si usano delle variabili per suggerire alla CPU che certi dati saranno utilizzati molto spesso, quindi cosi la CPU li caricherà nella cache senza passare ogni volta dal BUS.

3. **TECNICA DI OTTIMIZZAZIONE IN FASE DI COMPILAZIONE** : è un metodo che in fase di compilazione identifica le linee di codice e i dati che vanno utilizzati di più. Questo è un modo per ottimizzare il codice. Funziona con compilatori moderni (C,C++ ecc...)
4. **PREFETCHING CON ACCESSO ALLE LINEE DI CACHE**: utilizza le linee di cache per l'accesso alla memoria, quindi trasporta dei blocchi fissi, da 64 byte solitamente, direttamente alla cache.

PIPELINE

Spiegazione Pipeline completa

Ben presto, **alla coda di Prefetch, fu affiancato un sistema a Pipeline** che ha lo scopo di sfruttare il concetto di **catena di montaggio:** invece di eseguire un'istruzione completamente e solo al termine la sua successiva, si può avviare l'istruzione subito dopo che la precedente è stata inserita nel data path.
Per esempio, basta che la **prima istruzione si trovi in fase di decode, e la successiva può essere posta in stato di fetch**. Così come in una catena di montaggio, un nuovo pezzo può essere lavorato anche se il precedente non è ancora stato completato: **basta che le fasi (dette anche stadi della pipeline) non si sovrappongano.**

Le fasi della pipeline sono: **fetch-decode-execute-store/writeback**

Writeback permette di scrivere il risultato di un operazione nei registri, apporta un beneficio nella velocità della CPU, aumentando il throughput di calcolo.

Così, una Pipeline a 5 stadi trasporta cinque istruzioni in catena di montaggio.
• La Pipeline sopperisce così alle attese di CPU veloci nei confronti di memorie lente (collo di bottiglia di von Neumann).
• Una volta dotato di Pipeline, in un processore si è notato che lo stadio di esecuzione è il più lento: lo stadio precedente fornisce più valori di quanto lo stadio di esecuzione, implementato nell'ALU, può elaborare.
• Ecco allora che sui processori sono montate più ALU in modo da servire velocemente (più cores = "più ALU")

Ci sono anche dei pericoli usando le pipeline Tipi di conflitti in dettaglio

1. **DATA HAZARDS**: si verifica quando un istruzione richiede dei dati che vengono forniti da un altra istruzione che non ha ancora finito (ritardi di elaborazione o blocchi)
2. **CONTROL HAZARDS** si verifica quando la pipeline deve gestire dei salti condizionali (tipo il go-to)
3. **STRUCTURAL HAZARDS** si verifica quando più istruzioni competono e cercano di accedere alla stessa risorsa (problemi dei filosofi a cena)

• Ecco allora che sui processori sono montate più ALU in modo da servire velocemente ogni istruzione che arriva allo stadio di execute. In questo caso il processore è detto Superscalare.
• In questo modo è possibile dotare i processori anche di due o quattro pipeline differenti.

La pipeline viene vanificata se:

- c'è un istruzione di salto (go-to) - qui la pipeline viene persa.
- Dipendenza dei dati tra le istruzioni - qui la pipeline potrebbe dover' essere interrotta

L'esecuzione predicativa (Speculative execution dettagliata), cerca di prevenire la perdita delle pipeline a causa delle istruzioni di salto, attraverso degli algoritmi che usano tabelle simili a memorie cache.
Il problema di questa tecnica, che in realtà è molto efficiente, si ha quando la previsione è sbagliata: le istruzioni eseguite inutilmente devono essere gettate e lo stato della macchina ripristinato.
L'esecuzione predicativa è anche nota come esecuzione speculativa, intendendosi quella elaborazione che computa anche il codice che potrebbe non essere mai utilizzato.

• **Quando il processore individua una dipendenza in una pipeline, invece «di buttarla» e attendere l'operando mancante, la salta e prosegue con istruzioni "future"** che, in teoria, dovrebbero essere eseguite solo dopo quella interrotta.
• **In questo caso la pipeline viene quasi del tutto conservata (un solo stadio rimane bloccato, fino all'arrivo dell'operando mancante) ma, una volta risolta la dipendenza, saranno già state eseguite altre istruzioni (quelle che avevamo chiamato istruzioni «future»)**, con conseguente risalita delle prestazioni.

Condizioni critiche
• Naturalmente le istruzioni "future" possono essere eseguite solo se non hanno, a loro volta, dipendenze con istruzioni in corso.
• Non appena le istruzioni con dipendenze terminano, il processore continuerà l'esecuzione in ordine (in order execution).
• La condizione più critica per questa tecnica si presenta quando il processore deve essere interrotto a causa di un interrupt; se il processore si trova in fase di fuori ordine, lo stato del sistema potrebbe non essere coerente. In questi casi il processore deve ripristinare lo stato della CPU ritirando «in ordine» tutte le fasi fuori ordine.

Per poter riordinare il giusto flusso di esecuzione dopo aver saltato e ricalcolato una istruzione con dipendenza, i processori utilizzano una serie di registri d'appoggio (interni e invisibili al programmatore) su cui memorizzare i calcoli temporanei delle istruzioni fuori ordine.

All'atto del riordinamento, **per evitare di spostare i valori dai registri interni a quelli effettivamente usati nel data path**, i processori sono in grado di **rinominare i registri interni nei nomi dei registri effettivi**, **risparmiando il tempo del trasferimento (register renaming).**
(in pratica anziché spostare i dati dei registri d'appoggio a quelli normali (che richiederebbe del tempo) rinominano quei registri d'appoggio con i nomi dei normali registri e fanno la stesso quelle operazioni).

VLIW
Anche se non esplicitamente, tutte queste innovazioni (pipeline, super- scalarità, predicazione, esecuzione fuori ordine) cercano di implementare un modello di esecuzione parallelo molto studiato nei centri di calcolo, e denominato VLIW (Very Long Instruction Word).

I compilatori moderni prendono tutte le **istruzioni a-sincrone e le raggruppano formando il VLIW cosi la CPU può parallelizzare** almeno quelle.

Vantaggio: Migliora la velocità di calcolo della CPU perché parallelizza le operazioni a-sincrone.

Limite: VLIW funziona bene solo quando le operazioni sono veramente indipendenti (ovvero può capitare che il compilatore sbagli), inoltre se nel codice non ci sono abbastanza operazioni parallelizzabili questo sistema non funziona.

Ulteriori informazioni:

Bottleneck

La CPU è progettata per operare a velocità estremamente elevate, elaborando milioni di istruzioni al secondo. Tuttavia, la RAM, sebbene veloce, ha tempi di accesso più lunghi quando la CPU richiede dati o istruzioni deve attendere che questi vengano recuperati dalla RAM. Questo processo di attesa genera un rallentamento, poiché la CPU rimane inattiva in attesa dei dati necessari per continuare l'elaborazione. Per mitigare questo problema, le CPU fanno uso della cache, una memoria più veloce e più vicina al processore, dove vengono conservati i dati e le istruzioni più frequentemente utilizzati. Tuttavia, anche con l'uso della cache, il bottleneck rimane una preoccupazione, specialmente in scenari di carico elevato o quando vengono eseguiti più processi contemporaneamente. In questi casi, se la RAM non è in grado di tenere il passo con le richieste della CPU, si crea un rallentamento significativo, influenzando negativamente l'efficienza complessiva del sistema.

Il test di Turing

Turing cercò di rispondere alla domanda "can machines think?", per fare ciò formulò quello che viene chiamato test di Turing ovvero un giudice umano comunica con due interlocutori nascosti: uno è un essere umano, e l'altro è una macchina. Se il giudice non riesce a distinguere chi è la macchina e chi è l'umano basandosi solo sulle risposte fornite, allora si dice che la macchina ha **superato il test** e può essere considerata "intelligente". Fino a qualche mese fa nessuno era riuscito a superarlo tranne ChatGPT-4 di recente (2024)

Processi sincroni / a-sincroni, latenza

- **La latenza** è il tempo che il dato impiega a passare per il bus a seguito del comando READ,
  Il comando viene lanciato (READ) - il dato passa per il bus - il dato viene effettivamente letto dalla CPU.
- **I processi sincroni** hanno bisogno di essere "sincronizzati" tra loro perché ad esempio il primo è propedeutico per il secondo (quindi il secondo non può iniziare se ancora il primo non ha finito) questi processi vengono eseguiti dalla CPU che esegue sia processi sincroni che a-sincroni.
- **I processi a-sincroni** non hanno bisogno di essere sincronizzati perché "lavorano da soli" infatti le GPU hanno moltissimi core rispetto alle CPU proprio perché loro devono svolgere solo processi a-sincroni
  (lo schermo è formato da delle matrici (1920×1080 è la matrice del full HD, quindi la risoluzione indica anche quanto è grande la matrice dello schermo) quindi ogni processo si occupa di una sottomatrice e svolge il suo compito dentro un core della scheda video, alla fine la scheda video mette insieme tutti i risultati di questi processi per avere una matrice completa e renderizzare lo schermo correttamente. (lo schermo a colori è dato da 3 matrici RED,GREEN,BLUE [RGB]))

Memorie

- L' ultima tecnologia di **DRAM** (fino al 12/10/24) è la DDR5 uscita nel 2020.
- Le **ROM** anche se hanno l'acronimo Read Only Memory attraverso procedure speciali si possono effettuare operazioni di scrittura.

Throughput = portata di calcolo

Deadlock: quando 2 istruzioni rimangono bloccate a causa di dipendenze reciproche

Codifica Bit-Pair

è una tecnica di compressione dei dati utilizzata per ridurre la lunghezza di una sequenza di bit senza perdere informazioni(raggruppa i bit in coppie (bit-pair))

questa tecnica e utile per ottimizzare l'uso della memoria