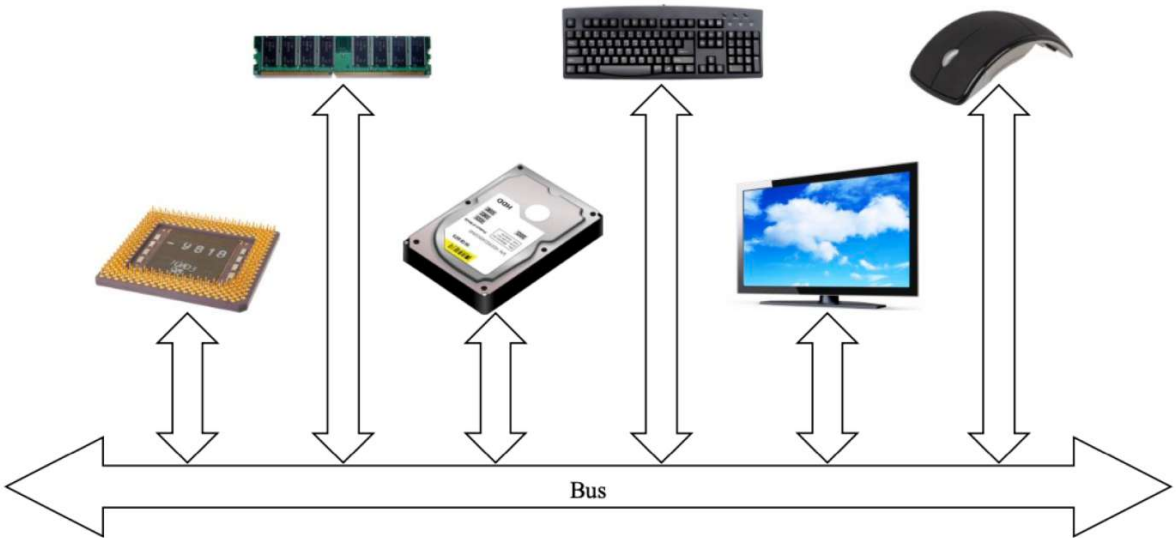
Architetture di calcolo 2.1.m4a

I sistemi di calcolo che utilizziamo al giorno d'oggi sono costituiti da massa, cioè una serie di cavi di collegamento a cui sono connessi diversi componenti, che sono suddivise in: componenti di eleborazione come la cpu, che è il cuore pulsante del computer esegue tutte le operazioni di calcolo; ci sono le componenti di memoria in particolare quella che noi chiamiamo ram (Random Access Memory) poi abbiamo i dischi e le periferiche d'ingresso/uscita o di input/output, che sono tutti i dispositivi tramite cui inseriamo dei dati come la tastiera o il mouse o tramite cui vediamo un risultato come ad esempio il monitor le stampanti.

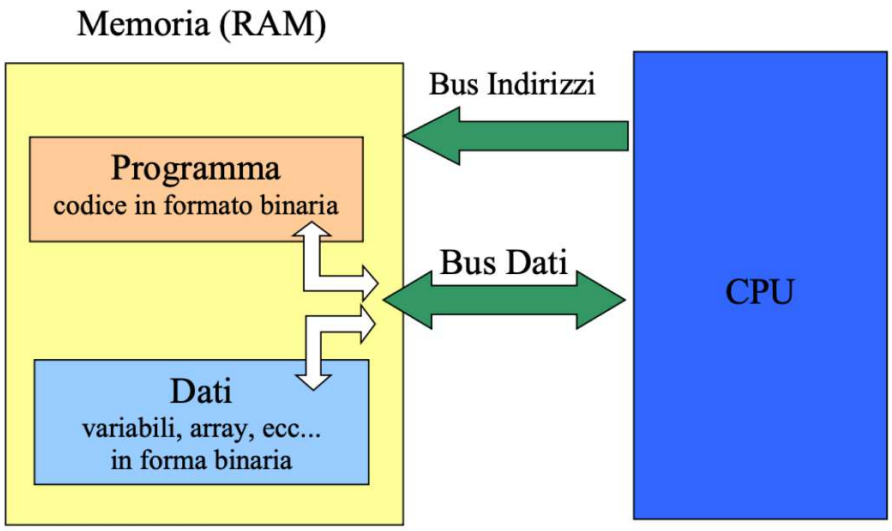
Un calcolatore digitale è un sistema composto da processori, memorie e dispositivi di input/output (I/O) collegati tra loro.



Ovviamente esistevano precedentemente questo modello o modelli differenti ma questo modello prende il nome di architettura bus orient, cioè orientato all'uso di un bus; quindi il focus di quest'architettura è proprio di disporre di questo bus ovvero quest'insieme di cavi che scorrono parallelamente gli uni con gli altri per trasferimento dei dati dei diversi componenti. Non sono esattamente dei cavi, ma dovete immaginare una scheda madre con dei circuiti elettronici e su di essa sono designati dei percorsi di rame, tante linee di rame parallele le une con le altre molto vicine e fitte queste linee create sono quelle che vengono chiamate cavi.

- Questa organizzazione del calcolatore digitale è detta "*bus oriented*".
- I componenti sono connessi fra loro mediante un **bus,** cioè un insieme di cavi paralleli sui quali vengono trasmessi indirizzi, dati e segnali di controllo.
- I bus possono essere esterni alla CPU, per connetterla alla memoria e ai dispositivi di I/O, oppure interni alla CPU stessa.

I bus possono essere sia esterni alla cpu e quindi sarebbero i bus esterni sono quelli che collegano tra di loro i diversi componenti, mentre quelli interni ai componenti quindi esistono dei bus interni che fanno collegare tra loro i componenti interni della cpu che ovviamente questa è divisa al suo interno da vari componenti.



- Nell'architettura di Von Neumann, **la memoria viene usata non solo per i dati ma anche per i programmi**.
- Grazie a questa architettura, i programmi (su schede perforate) potevano essere caricati in memoria con un lettore di schede evitando complesse configurazioni/programmazioni con interruttori e cavi.
- **Programmi e Dati** al tempo di esecuzione sono caricati in memoria (codificati in forma binaria).
- Programmi e dati sono **trasferiti entrambi attraverso il bus dati**. Il Bus indirizzi è utilizzato dalla CPU per indicare alla memoria le locazioni dove risiedono le informazioni da trasferire.

La storia dell'evoluzione dei computer è rappresentata dall architettura di Thom Holman (thomas holman credo) il quale definì una macchina che era rivoluzionaria rispetto agli approcci precedenti. Era rivoluzionaria holman intuì che era più conveniente memorizzare nella ram non solamente i dati da elaborare ma in realtà è conveniente memorizzare anche i programmi. Quindi la ram contiene programmi e dati, entrambe le cose (prima di holman conteneva solo i dati).
RAM (Random Acess Memory) chiamata così perché possiamo accedere in maniera e in tempo costante in un qualsiasi, quindi casuale, punto della memoria stessa.
Prima i programmi venivano cablati dai programmatori nel computer, cioè per far fare al computer qualcosa di specifico cioè per eseguire un particolare programma, nei tempi antichi bisognava collegare gli spinotti in varie posizioni e creare dei circuiti che facessero quello che si voleva fare, in epoca successiva furono inventate le schede perforate, che erano dei fogli di carta in cui venivano creati dei buchi in determinate posizioni, queste schede venivano inserite in dei lettori di schede del computer e il lettore leggendo le schede capiva cosa doveva fare (un'istruzione si codificava facendo un buco in un certo punto) e quindi il programma era costituito da uno i più schede, mazzi di schede, quindi il programma si trovava fuori dalla ram si trovava nelle schede e il lettore di schede eseguiva un'istruzione dopo l'altra, chiaramente è un modello poco efficiente per diversi motivi, di fatto leggere i dati dalla scheda è molto più lento che leggerli dalla ram, e poi programmare con le schede significava che al minimo errore si doveva buttare la scheda e rifarla.
È proprio dalle schede che nasce il concetto di bug (gli insetti si intrufolavano tra i cablaggi della macchina modificando la lettura delle schede e quindi portavano errori che da quel momento furono chiamati bug).

Quindi il modello di Holman semplificò di molto il modo di programmare, grazie al fatto che programmi e dati si trovavano in forma binaria sulla ram, ciò fu importante perché la cpu deve leggere sia i dati sia i programmi per capire cosa fare.
Sia i programmi codificati in forma binaria sia i dati vengono copiati nella cpu usando il cosiddetto bus dati.
Quindi il bus dati è quella parte di bas che trasferisce i dati ma ovviamente non è sufficiente trasferire i dati sul bus, perché bisogna trasferire anche i gli indirizzi.
La memoria è costituita da celle, ogni cella avrà un numero identificativo che è il suo indirizzo, la cpu per dire alla MAR quale cella vuole deve dire quale indirizzo vuole, e poi riceve tramite il bas dati il contenuto della cella questo vale sia in lettura che in scrittura.
Se voglio leggere devo dire alla ram dammi il contenuto della cella x, mentre se voglio scrivere dirò scrivi questo nella cella x, quindi la cpu dovrà dire alla ram leggi o scrivi, questo implica che c'è un altro componente, ovvero il bus di controllo con il quale la cpu dice alla ram leggi o scrivi, quindi controlla quale tipo di operazione deve essere attuata.
La cpu è l'unica centrale di elaborazione, quindi il cuore pulsante della macchina, è costituita da 3 componenti fondamentali che sono: • Unita di controllo, •la ALU ed • i registri.
Il compito della cpu è quello di eseguire i programmi che si trovano in ram, seguendo un'istruzione per volta fino ad arrivare al completamento del programma, quindi questi tre componenti cooperano per portare al termine questo incarico.
L'Unità di controllo è la parte di cpu che si occupa di leggere le istruzioni della memoria centrale, per determinare il tipo leggere il codice e cosa fare.
La ALU si occupa invece delle operazioni aritmetiche logiche.
I registri invece sono delle piccole memorie, che si trovano dentro la cpu e che servono principalmente a mantenere i dati su cui la cpu sta lavorando.
I dati stanno nella ram però la cpu non può lavorare direttamente sui dati che sono nella ram, deve prendere i dati della ram copiarli nei registri e solo dopo elaborarli.
Ci sono inoltre tra i registri registri speciali i principali sono due **Program counter** e **Instruction register**.

Il compito della CPU è quello di eseguire i programmi immagazzinati nella memoria centrale leggendo le loro istruzioni ed eseguendole in sequenza.

Una CPU è composta da:

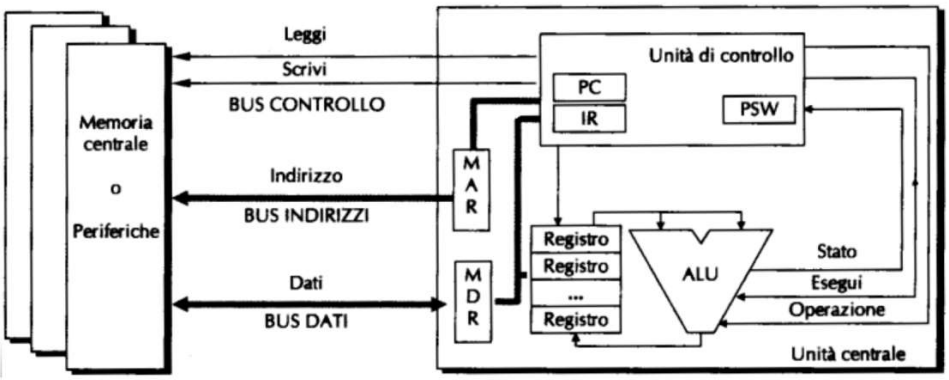
Unità di controllo: legge le istruzioni dalla memoria centrale e ne determina il tipo.

ALU: esegue le operazioni necessarie all'esecuzione delle istruzioni (AND,OR, addizione binaria).

Registri: sono una piccola memoria ad alta velocità utilizzata per memorizzare i risultati temporanei e le informazioni di controllo necessarie al funzionamento dell'ALU. I registri risiedono all'interno della CPU.

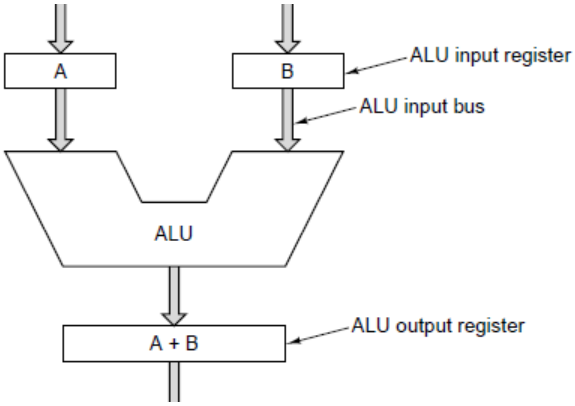
- Normalmente i registri hanno tutti le stesse dimensioni, alcuni vengono utilizzati per compiti specifici altri sono general purpose.
- Il registro più importante è il **Program Counter** (PC) che indica la prossima istruzione da eseguire.
- L'**Instruction Register** (IR) è il registro che memorizza l'istruzione che si sta per eseguire.

Program counter tradotto il Contatore di programma, non è esattamente un contatore ma è un registro che continente l'indirizzo della prossima istruzione da eseguire, l'indirizzo della ram in cui si trova la prossima istruzione da eseguire; mentre **instruction register** è un altro registro che memorizza l'istruzione che stiamo eseguendo adesso, non memorizza l'indirizzo dell istruzione che stiamo eseguendo adesso ma memorizza il codice dell'istruzione; quindi il primo memorizza l'indirizzo e ci dice quale sarà la prossima istruzione che andremo a calcolare, il secondo contiene il codice dell indirizzo e ci dice quale istruzione stiamo eseguendo in questo momento.



Descrive un immagine
La memoria è collegata all'unita centrale.
La cpu è costituita da diversi componenti, parla con la ram attraverso tre bus, un unico bus diviso in tre parti: bas indirizzi che trasferisce degli indirizzi di dati della ram, il bas dati che trasferisce dei dati ad esempio dalla ram ai registri o dai registri alla ram, se questi dati sono messi nei registri devono tornare nella ram sotto forma di altri dati che viaggiano sul bus dati; invece il bus di controllo comunica alla ram che si vuole leggere o che si vuole scrivere quindi sui fili mandiamo un codice(es. 00 per dire leggi 01 per dire scrivi).
Il bus di controllo è collegato a un componente che si chiama unità di controllo, che è quella unità che determina le operazioni che si devono fare.
All'interno dell'unità di controllo troviamo i due registri citati prima, program counter instruction register.
Uscendo dall'unita di controllo si trovano due registri che si chiamano MAR (memory address register) registro di memoria degli indirizzi e MDR (Memory data register) il quale è un registro che contiene dati della memoria.
Il mar contiene indirizzi che dobbiamo mandare alla ram, di fatto è attaccato alla ram tramite un bus indirizzi; quindi se volessi dire alla ram leggi la cella 1000 dovrò scrivere in parole 1000 nel mar e poi questo 1000 lo mando alla ram tramite il bus indirizzi, mentre se voglio scrivere 1000 in valore nella ram dovrò scrivere 1000 nel MDR, perché è un dato, e mandare questo dato tramite il bus di dati.
Quindi esistono due registri separati, mar ed mdr, sostanzialmente perché servono a contenere valori concettualmente differenti, sono sempre numero binari però alcuni rappresentano indirizzi altri dati quindi come tali vengono messi in registro differenti.
Mdr ->come funziona -> serve a contenere i dati che voglio scrivere sulla ram o che leggo dalla ram e che mi arrivano dalla ram è quindi un registro a due direzioni.
Es. Se voglio scrivere un dato nella ram devo dire alla ram cosa voglio scrivere, dove voglio scrivere e cosa, quindi bus di controllo in cui dico scrivi, bus indirizzi in cui dico dove scrivere e poi il bus di dati per dire cosa scrivere; quindi cosa farò ? se voglio scrivere il valore 10 nella cella 1 dovrei scrivere nel mar il valore 1 nell mdr il valore 10 e poi nel bus di controllo dovrò dire scrivi, la ram a quel punto riceverà il comando di scrivere, andrà nel mar e si prenderà il numero della cella che sarà 1, andrà nell mdr si prende il valore che è 10, andrà nella cella 1 e metterà il valore 10.
Quindi tramite questo tre bus si può leggere e scrivere tutto ciò che serve per eseguire il programma.
Si nota nell'immagine che il mar è collegato all'unita di controllo che va a scrivere questi registri, la stessa unità di controllo fa uscire i messaggi di lettura e scrittura verso la ram.

L’ALU rappresentato con una forma pseudo-trapezoidale con in alto i dati in ingresso (normalmente due operandi ma dipendente dal tipo di operazione da attuare che riceve) e in basso i dati in uscita (risultato dell’ operazione logica) che poi finiscono nei registri .
Che fine fa il valore che l’ALU mette nel registro ?
O finisce nel registro e poi ritorna verso l’ALU per essere processato di nuovo però senza passare per la ram oppure se si avrà il risultato finale per cui non verrà ri-processato e finirà nella ram.



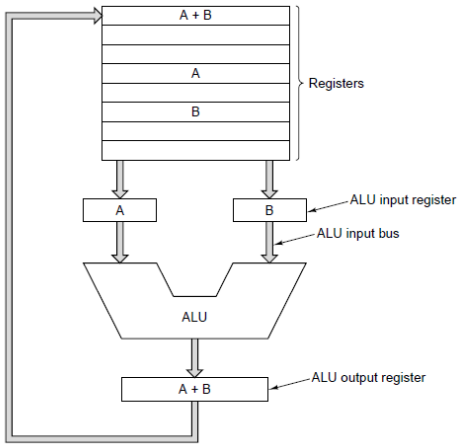
I passi sono sostanzialmente sempre i seguenti: 1) la cpu mette il valore del program counter (contiene l’indirizzo della prossima istruzione da eseguire) su mar e poi attiva linea “Leggi” (ovvero leggi il valore di quella cella); 2) la memoria accede al mar e scrive il contenuto della cella in mdr, ciò avviene mediante i bus di dati; 3) la cpu accede alla mdr e copia il valore sull’ir e decodifica il significato in base ai codici dell’operazione da svolgere; 4) l’istruzione viene passata alla ALU per essere eseguita; 5) se l’operazione da svolgere ha bisogno di operandi di memoria, per ciascun operando vengono richiamati i registri e vengono caricati all’interno, il numero di registri presenti è sufficiente per accogliere il numero di operandi presenti per ogni tipo di operazione possibile (sono qualche decina), recuperati gli operandi viene effettuata l’operazione specifica e si può andare alla eventuale struttura di memoria in cui il dato deve essere eventualmente allocato se non necessita di essere iterato nuovamente dalla ALU,supponendo che siano risultati finali e che questi debbano tornare alla ram, ciò avviene in più passaggi: •mettiamo in mar l’indirizzo di destinazione in cui si trova il dato che deve essere allocato • la memoria accede al mar e reperisce il dato richiesto, e attraverso i bus di dati lo riscrive su mdr • la cpu ricopia sul registro di destinazione il valore presente sul mdr; 6) terminata l’esecuzione la CPU copia sul registro di destinazione il valore estrapolato dalla ALU e se è previsto scrive in memoria il valore tramite l’istruzione ”Scrivi”; 7) terminate l’esecuzione del processo una volta aggiornato il valore di PC il ciclo ricomincia da capo.

1) la CPU mette il valore di PC (indirizzo della prossima istruzione da leggere dalla memoria) su MAR e attiva la linea Leggi;
2) la memoria attraverso il bus indirizzi accede a MAR e, una volta reperito quanto richiesto, lo scrive su MDR attraverso il bus dati;
3) la CPU copia su IR il valore di MDR e decodifica l'istruzione;
4) l'istruzione passa in esecuzione sulla ALU;

6) terminata l'esecuzione la CPU copia sul registro destinazione il valore prodotto dalla ALU; se è prevista scrittura in memoria del valore calcolato:
   6.1) la CPU mette l'indirizzo della cella di destinazione su MAR e il risultato su MDR e attiva la linea Scrivi;
   6.2) la memoria attraverso il bus indirizzi accede a MAR, attraverso il bus dati a MDR e, una volta reperito il valore in MDR, lo scrive sulla propria cella interna indicata da MAR;
7) Si ritorna al punto 1 dopo aver aggiornato il valore di PC (prossima istruzione da eseguire).

Questo è sostanzialmente il percorso che fanno le istruzioni nelle strutture.
Il concetto fondamentale da ricordare è che grazie a questi 3 BUS la CPU può dire dove andare o dire cosa spostare, gli indirizzi nei BUS vengono messi attraverso i REGISTRI.
I bus di dati usano i filtri … per ricevere i dati dai registri, mentre i bus di indirizzi usano filtri … per ricevere i dati dai registri.

In questo scenario che abbiamo descritto esiste quello che viene chiamato il percorso dei dati (DATA PATH, descrive sostanzialmente quale percorso seguono i dati che entrano in input ed escono in output dalla ALU) che vengono estrapolati da un registro e successivamente quando vengono manipolari rientrano in un altro registro, i risultati finali di questi dati vengono a loro volta vendono rispediti in registro generali i Registri General Purpose.
Questo è il data path che sostanzialmente troviamo in tutte le macchine, si usa il termine ciclo di data path per indicare il percorso dei dati verso la ALU e la successiva memorizzazione, si parla di ciclo proprio perché sono una serie di operazioni che vanno dall'estrapolazione del dato alla sua memorizzazione, quindi partono dalla memoria e tornano alla memoria.



In figura è rappresentato il classico **data path** per una CPU di Von Neumann. Il data path comprende l'ALU e i registri.

Il passaggio di due operandi attraverso la ALU e la memorizzazione del risultato in un nuovo registro viene detto **ciclo di data path**..

Il ciclo di data path avviene con una frequenza che dipende dalla frequenza della macchina: il CLOCK.
Il Clock presente nella macchina è uno script che genera impulsi con una frequenza altissima (Ghz) e determina con quante operazioni al secondo la macchina esegue.
Le operazioni del linguaggio Assembly vengono eseguite in uno o più cicli di data path, il numero di cicli dipende dalla distanza dell'operazione rispetto all'operazione semplice eseguibile dal linguaggio macchina per cui man mano che la distanza aumenta aumentano anche il numero di cicli necessari (calcoli più complessi non sono nient'altro che ripetizione di calcoli più semplici) (SOMMA ——> SOMME ——> MOLTIPLICAZIONI E DIVISIONI ——> ESPONENZIALI ——> CALCOLI COMPLESSI).

Ogni istruzione ISA (assembly) viene eseguita in uno o più cicli di data-path; diversi cicli sono necessari per istruzioni complesse (es. divisione).

In architetture non parallele il ciclo di data path corrisponde al **ciclo di clock** (misurato in nanosecondi) ossia l'intervallo di tempo utilizzato per sincronizzare le diverse operazioni del processore.

La velocità con cui viene compiuto un ciclo di data-path contribuisce significativamente a determinare la velocità della CPU.

NON TUTTE LE OPERAZIONI SEMPLICI RICHIEDONO LO STESSO TEMPO !!! (numero di cicli di data path X tempo di esecuzione di un ciclo [1/Clock]).
Il ciclo di data path ha una durata che corrisponde al ciclo di Clock ovvero l'intervallo di tempo tra due istanti di Clock, nell'istante di Clock c'è un chip che genera un impulso su cui si sincronizzano tutte le operazioni, le macchine nel tempo hanno sempre aumentato esponenzialmente il numero di Clock quindi il numero di operazioni al secondo, esso si misura in multipli del Hz (1 Hz = 1 operazione semplice al secondo, 1 MHz = 1 milione di operazioni semplici al secondo, 1 GHz = 1 miliardo di operazioni semplici al secondo).

durata ciclo di data path = durata ciclo di clock = 1/F
(dove F è la frequenza di lavoro della CPU)

durata istruzione ISA = n × durata ciclo di data path
(*n* variabile per istruzioni diverse, ma anche per architetture diverse)

Istruzioni ISA per sec. = 1/durata istruzione ISA = F / n

Siccome le operazioni di media non sono operazioni semplici e che quindi richiedono 10/100/1000 e più e più cicli di Clock, capiamo bene che una cosa è poter fare 5 miliardi di operazioni al secondo e una cosa è poterne fare 500.
Possiamo in effetti calcolare la durata dei cicli di data path come l'inverso della frequenza di Clock (1/Clock), la durata di un'operazione assembly è pari a n(numero di operazioni semplici da effettuare) * durata data path (1/Clock).
Dato che non è semplice capire n a quanto corrisponde si utilizzano dei benchmark che hanno delle operazioni note da svolgere e si calcola la n media eseguita in un unita di tempo e così si può estrapolare la capacità della macchina di effettuare n operazioni complesse al secondo.
(per capire meglio possiamo fare riferimento alla tabella) da qui si può notare l'anno di uscita, il MIPS (milioni di istruzioni assembly al secondo) e il Clock della macchina, tipo la prima macchina rappresentata ovvero la 4004 aveva meno di un MHz (0,74) è un MIPS di un decimo del Clock (0,07).
Se andiamo avanti osserviamo che i processori hanno avuto un enorme aumento di velocità di processazione, di fatto oggi giorno si può notare che si arriva a verte di 5 GHz ed un MIPS di 412000 (milioni di operazioni).

CPU	Anno	Frequenza	Registri	Transistor	MIPS
4004	1971	0,74 MHz	4 bit	2.300 (10 μm)	0,07
8008	1972	0,5 MHz	8 bit	3.500 (10 μm)	0.05
8080	1974	2 MHz	8 bit	6.000 (6 μm)	0,29
8086	1978	8 MHz	16 bit	29.000 (3 μm)	0,66
80286	1982	12,5 MHz	16 bit	134.000 (1,5 μm)	2,66
Intel386	1985	33 MHz	32 bit	275.000 (1 μm)	10
Intel486	1989	50 MHz	32 bit	$1{,}2 \cdot 10^6$ (0,8 μm)	41
Pentium	1993	66 MHz	32 bit	$3{,}1 \cdot 10^6$ (0,8 μm)	112
Pentium Pro	1995	200 MHz	32 bit	$5{,}5 \cdot 10^6$ (0,35 μm)	541
Pentium II	1997	300 MHz	32 bit	$7{,}5 \cdot 10^6$ (0,35 μm)	813
Pentium III	1999	600 MHz	32 bit	$9{,}5 \cdot 10^6$ (0,25 μm)	1.105
Pentium 4	2000	1.5 GHz	32 bit	$42 \cdot 10^6$ (0,18 μm)	2.262
Pentium D	2005	3.2 GHz	64 bit	$230 \cdot 10^6$ (90 nm)	7.145
Core i7 (gen. 5)	2014	3.8 GHz	64 bit	$1{,}3 \cdot 10^9$ (14 nm)	298.190
Core i9 (gen. 9)	2018	4.7 GHz	64 bit	$3 \cdot 10^9$ (14 nm)	412.090

Questa operazione ciclica che abbiamo descritto prima prende in gergo tecnico il nome di CICLO FETCH-EXECUTE, fetch significa sostanzialmente acquisire dalla memoria l'istruzione da eseguire e caricarla (di fatto la prima cosa da fare è proprio prendere dalla memoria l'istruzione da dover eseguire), per caricarla dobbiamo però prima comprendere cosa l'istruzione ci dice di fare quindi deve essere DECODIFICATA; EXECUTE significa invece proprio eseguire l'operazione caricata nella fase precedente.

La CPU opera in modo ciclico, ripetendo le seguenti operazioni fino al termine dell'esecuzione del programma:

- **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.

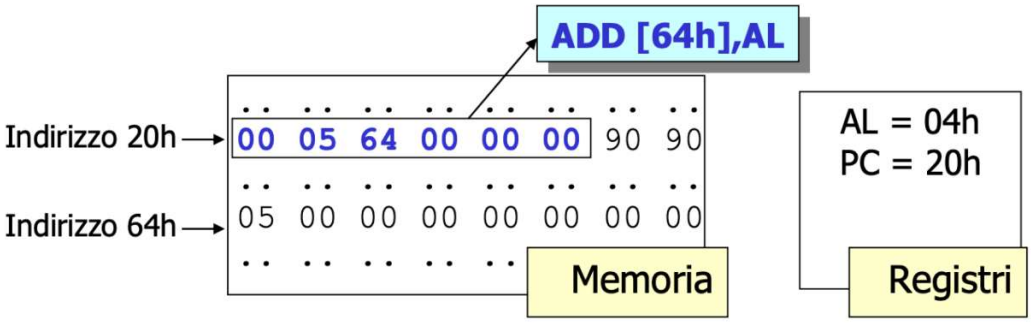
Possiamo vedere nell'immagine da cosa sono composte le tre Macro fasi di FETCH, DECODE e EXECUTE.
Si parla di ciclo proprio perché si torna sempre all'inizio (il punto 8 torna al punto 1) (il punto 6 può essere costituito da una serie di sottopassi qualora l'istruzione non sia una istruzione semplice ma una complessa [complesso di istruzioni semplici]); la parte di fetch quindi estrapola il comando, aumenta il program counter e successivamente fa si che venga decodificato il comando stesso; l'esecuzione delle operazioni dipende da una interpretazione, questa interpretazione viene effettuata da quello che in gergo prende il nome di Microprogramma, questo programma si trova direttamente sopra al livello macchina ed è in grado di capire dato un particolare codice che cosa fare.

Più in dettaglio, l'esecuzione di ogni istruzione da parte della CPU richiede una serie di passi che, in linea di massima, possono essere così riassunti:

1) Leggi l'istruzione seguente dalla memoria e mettila nel IR
2) Incrementa il PC per indicare l'istruzione seguente
3) Decodifica l'istruzione appena letta
4) Se l'istruzione utilizza degli operandi (parole) determina dove si trovano (memoria/registri)
5) Se necessario metti gli operandi in registri della CPU
6) Esegui l'istruzione
7) Salva il risultato in un registro
8) Torna al punto 1

A sua volta il punto (6) può richiedere un insieme di sotto-passi a seconda della complessità dell'Instruction Set. La traduzione di una istruzione nei suoi passi elementari è effettuata da un **interprete** (microprogramma).

Supponiamo di avere questa situazione: slide 17 … la h cosa rappresenta ??? È un numero esadecimale e la h (hex) sta proprio per farci capire che è un numero esadecimale visto che non fa parte della rappresentazione del sistema; i numeri esadecimale vengono usati in quanto sono molto ma molto più compatti dei numeri binari, tramite l'uso degli esadecimali spesso si rappresentano alcuni dati molto grandi come gli indirizzi che altrimenti richiederebbero molto spazio per essere espressi in binario o anche in decimale.
PC che vuol dire ? Program counter e pc =20h vuol dire che la prossima istruzione da eseguire si trova nella cella 20h, questa dicitura presente nella cella esplicitata dall'indirizzo del pc una volta tradotta vuol dire ADD [64H] AL ovvero : aggiungi il valore della cella 64h al registro della cpu chiamato AL; ma in AL è già presente un valore per cui in realtà aggiungiamo al valore presente nel registro il valore della cella di indirizzo 64h.
Nella fase di FETCH ovvero la prima, avviene il caricamento nell'IR (instruction register) il dato puntato dall'indirizzo presente in Program Counter, subito dopo PC viene incrementato; nella seconda fase ovvero di DECODIFICA L'Unità di controllo decodifica il valore della cella e capisce il significato dell'operazione da effettuare; nella terza fase ovvero quella di EXECUTE dopo aver recuperato dalla memorano valori delle celle e il significato dell'operazione avviene la vera e propria esecuzione dell'istruzione e il salvataggio in memoria del valore risultante.



Fetch:
viene letta in IR la sequenza 00 05 64 00 00 00 dalla memoria all'indirizzo correntemente puntato da PC (20h); PC viene incrementato.

Decode:
la sequenza 00 05 64 00 00 00 viene decodificata in ADD [64h], AL

Execute:
viene recuperato dalla memoria il contenuto all'indirizzo 64h; questo viene sommato con il valore di AL e nuovamente scritto all'indirizzo 64h. L'operazione richiede sicuramente più cicli di data-path, se non altro per la necessità di recuperare operandi dalla memoria.

Sin da quando furono realizzati i primi sistemi ci si accorse che, eseguire in modo sequenziale la fetch, la decodifica e l'execute determinava un rallentamento dell'esecuzione stessa, per ovviare ciò fu inventata una tecnica che si chiama PREFETCHING in cui al posto di caricare una singola istruzione ne vengono caricate più di una in modo da tenerle già pronte, queste istruzioni vengono caricate in quello che si chiama BUFFER DI PREFETCH, in questo modo si tende a eliminare il tempo morto tra le esecuzioni.

- Uno dei maggiori colli di bottiglia nella velocità di esecuzione delle istruzioni è rappresentato dal prelievo delle istruzioni dalla memoria.
- Una prima soluzione a questo problema fu la tecnica del **prefetching**: prelevare in anticipo le istruzioni dalla memoria, in modo da averle già a disposizione nel momento in cui dovessero rendersi necessarie.

- Le istruzioni venivano memorizzate in un insieme di registri chiamati **buffer di prefetch,** dai quali potevano essere prese nel momento in cui venivano richieste, senza dover attendere che si completasse una lettura della memoria.

- In pratica la tecnica di prefetching divide l'esecuzione dell'istruzione in due parti: il prelievo dell'istruzione e la sua esecuzione effettiva.

Possiamo dunque dire che il PREFETCHING consiste nel caricarsi in anticipo le istruzioni da dover eseguire successivamente, ma ciò successivamente è stato ulteriormente migliorato andando a formare quella che è la tecnica di PIPELINING, tecnica in cui invece di dividere l'esecuzione di un'istruzione solamente in due fasi (estrapolazione ed esecuzione dell'istruzione), la si divide in un numero maggiore di parti che possono essere eseguite in parallelo da componenti hardware dedicati. (***non confondere con pipeline***)

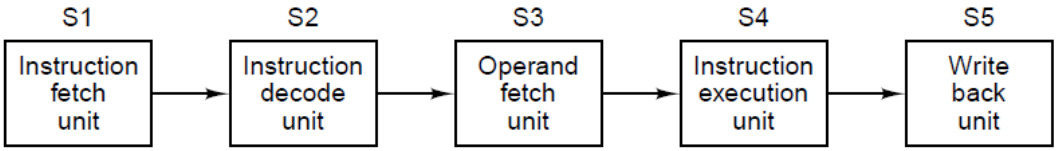
- Il concetto di **pipeline** è una evoluzione della strategia del prefetching: invece di dividere l'esecuzione di un'istruzione solamente in due fasi, la si divide in un numero maggiore di parti che possono essere eseguite in parallelo da componenti hardware dedicati.

Esempio posso dividere il tutto in 5 stadi come riportato nella slide in cui: S1) è l'unita di fetch che si occupa di estrarre dalla memoria l'operazione, S2) è l'unita di decodifica dell'istruzione, S3) è l'unita di fetch che preleva gli operandi, S4) è l'unita di esecuzione dell'operazione ed S5) è L'Unità di scrittura del risultato dell'operazione.
Cercando di velocizzare le tempistiche di esecuzione attraverso il concetto di pipelining ci troveremo in una situazione di parallelismo tra più operazioni ad istanti di Clock successivi, è così descrivibile:

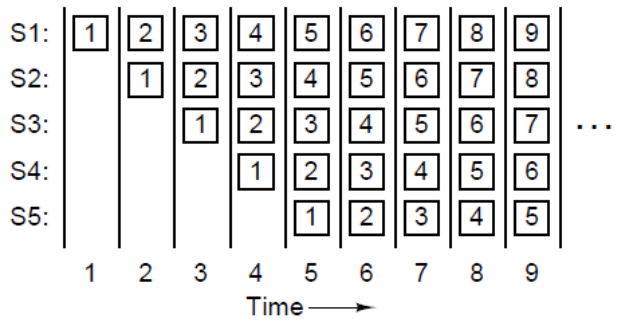
- Istante Clock 1: lo stadio S1 sta lavorando sull'istruzione 1, prelevandola dalla memoria.
- Istante Clock 2: S2 decodifica l'istruzione 1, mentre S1 preleva l'istruzione 2.
- Istante Clock 3: S3 preleva gli operandi per l'istruzione 1, S2 decodifica l'istruzione 2 e S1 preleva l'istruzione 3.
- Istante Clock 4: S4 esegue l'istruzione 1, S3 preleva gli operandi per l'istruzione 2, S2 decodifica l'istruzione 3 e S1 preleva l'istruzione 4.
- Istante Clock 5: S5 scrive il risultato dell'istruzione 1, mentre gli altri componenti lavorano sulle istruzioni successive.

Con il pipelining se abbiamo delle unità di elaborazione specializzate possiamo parallelizzate l'esecuzioni senza fare dei cicli sequenziali come visti prima.
(Fa un paragone con un autolavaggio con più persone specializzate in un compito specifico messe in catena tra di loro).

- Esempio di pipeline a 5 stadi:

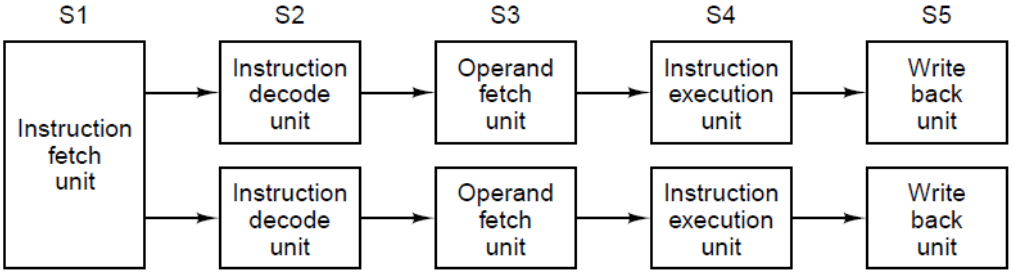


- **Clock 1**: lo stadio S1 sta lavorando sull'istruzione 1, prelevandola dalla memoria.
- **Clock 2**: S2 decodifica l'istruzione 1, mentre S1 preleva l'istruzione 2.
- **Clock 3**: S3 preleva gli operandi per l'istruzione 1, S2 decodifica l'istruzione 2 e S1 preleva l'istruzione 3.
- **Clock 4**: S4 esegue l'istruzione 1, S3 preleva gli operandi per l'istruzione 2, S2 decodifica l'istruzione 3 e S1 preleva l'istruzione 4.
- **Clock 5**: S5 scrive il risultato dell'istruzione 1, mentre gli altri componenti lavorano sulle istruzioni successive.

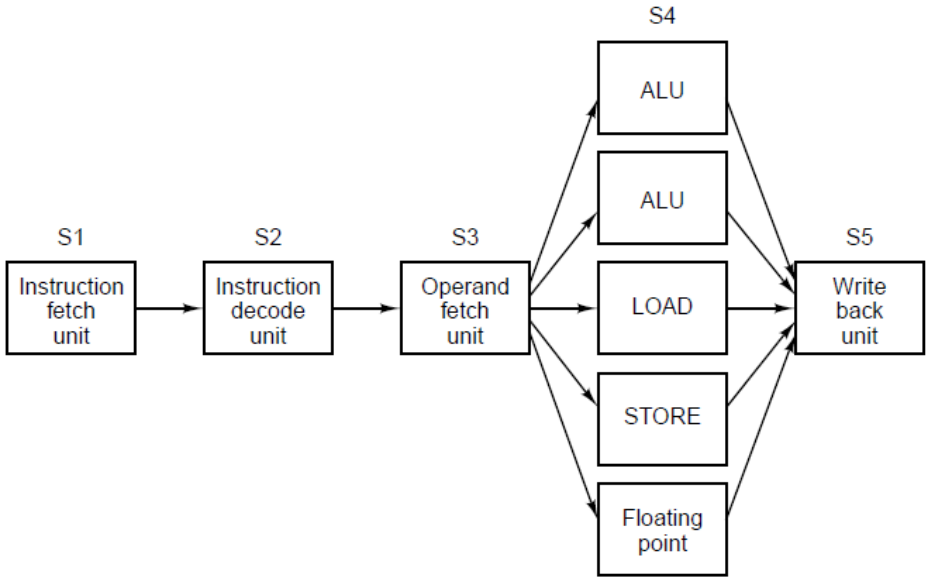


Abbiamo la possibilità di lavorare in più pipelining consecutivi per sveltire ulteriormente il processo; esempio nella doppia pipeline una singola unità di fetch preleva due istruzioni alla volta e le inserisce in due pipeline, ognuna delle quali è dotata di una ALU.
Da notare che affinchè le due istruzioni possano essere eseguite in parallelo, non devono esserci conflitti nell'uso delle risorse e nessuna delle due istruzioni deve dipendere dal risultato dell'altra; di fatto il compilatore deve occuparsi di gestire correttamente questa situazione, in quanto l'hardware non effettua alcun controllo e se le istruzioni sono incompatibili restituisce un risultato errato.
L'idea della doppia pipeline è quella di avere due linee di pipeline che si occupano in parallelo anche dell'esecuzione dell'istruzione, l'unica fetch a rimanere realmente sequenziale è quella delle istruzioni della ram che in realtà non rappresenta un collo di bottiglia, e ciò rappresenta potenzialmente un raddoppiamento della velocità di esecuzione (riuscendo però ad evitare tutti i conflitti di registri e di operazioni [ciò si può evitare in due modi: o il programmatore scrive codice già parallelo o il compilatore evita il conflitto]) rispetto alla già aumentata velocità della pipelining.

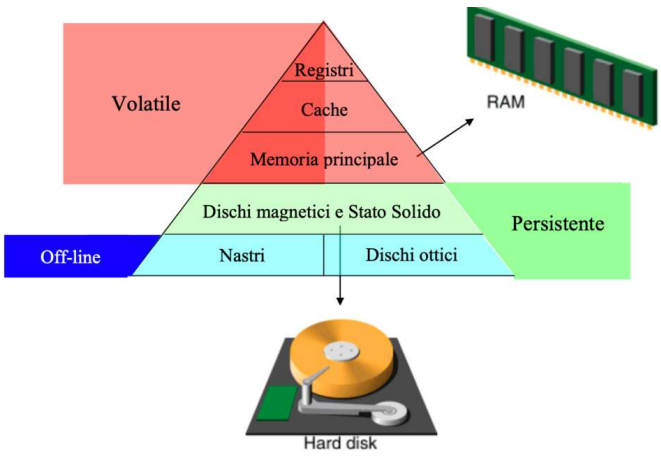
- Nell'esempio seguente, una singola unità di *fetch* preleva due istruzioni alla volta e le inserisce in due pipeline, ognuna delle quali è dotata di una ALU.
- Affinché le due istruzioni possano essere eseguite in parallelo, non devono però esserci conflitti nell'uso delle risorse (i registri) e nessuna delle due istruzioni deve dipendere dal risultato dell'altra.
- Il compilatore deve occuparsi di gestire correttamente questa situazione, in quanto l'hardware non effettua alcun controllo e se le istruzioni sono incompatibili restituisce un risultato errato.



Un ulteriore evoluzione che troviamo anche oggi giorno nelle architetture moderne è quella delle ARCHITETTURE SUPER-SCALARI, l'idea alla base delle architetture superscalari consiste nell'avere una singola pipeline, ma di associarle più unità funzionali nella catena esecutiva (presenta anche ALU separate).



Un altro componente importante del calcolatore/ computer è rappresentato dalla memoria, il termine memoria in realtà è molto generale poiché sotto questo termine si racchiudono diversi tipi di memoria.
In generale una memoria è uno spazio che ci consente di allocare e salvare dei dati e programmi ed è quindi fondamentale per il funzionamento della macchina.
La ram è fondamentale per caricare l'SO i programmi ecc, ma non è l'unica fondamentale per il corretto funzionamento.
In questa piramide troviamo rappresentati i vari tipi di memoria presenti, l'idea è che più ci avviciniamo alla punta, l'apice della piramide più la memoria che rappresenta è piccola, quindi più andiamo verso la base più la memoria sarà grande; le memorie più piccole in termini di quantità di dati che complessivamente memorizzo nella macchina sono più costosi.
Subito dopo i registri troviamo la cache si trova tra la cpu e la ram, successivamente troviamo la ram, la più importante.
Queste memorie qui in rosso sono memorie volatili ovvero memorie che allo spegnimento della macchina perdono il dato, e di conseguenza non sono idonee al mantenere dati in maniera costante nel tempo.
Sotto le memorie volatili troviamo le memorie persistenti che a loro volta si distinguono in due categorie principali: dischi allo stato solido e dischi magnetici (sotto questi ultimi troviamo anche i nastri magnetici).
Questi dispositivi alla base sono dispositivi offline caratterizzati da un tempo di lettura dei dati dipendente anche dal tempo impiegato per essere montato e letto dalla macchina.



Questa piramide non solamente si caratterizza per il fatto che alla cima abbiamo i supporti più piccolo mentre alla base abbiamo quelli più grandi, ma anche per il fatto che il costo per byte cresce salendo la piramide, ciò è dovuto dal luogo in cui è integrata la memoria e dal costo di produzione della tecnologia stessa, anche se le memorie a costo maggiore hanno tipicamente una maggiore velocità di accesso ai dati.

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

- 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 dimensione delle memorie cresce scendendo la piramide

I dati quando li memorizziamo non abbiamo la garanzia che rimanga sempre lì , sappiamo infatti che il dato può essere soggetto a corruzione, ovvero si può rovinare per volontà di terzi, danni accidentali, danni per superamento di cicli di scrittura, o anche per problematiche ambientali (raggi uv, onde elettromagnetiche, calore ecc); ciò avviene con qualsiasi dispositivo di memoria (IN AMERICA ESISTE UN DETTO CHE IN ITALIANO SI TRADUCE : CI SONO DUE COSE CERTE LA MORTE E LE TASSE , AGGIUNGE IL PROFESSORE ANCHE CHE IL DISCO SI ROMPERÀ !).
Il modo migliore per mantenere i dati è quello di averli nel cloud in quanto vengono costantemente ricopiati e controllati in caso di errori.
Come facciamo a capire se un dato si è modificato o deteriorato ?
Sostanzialmente andiamo a controllare i bit del dato nella forma originale e i bit nella forma successiva e vedere se si sono modificati.
Esiste anche un modo per quantificare il cambiamento di un dato , una misura, che prende il nome di DISTANZA DI HAMMING la quale indica il numero di bit corrispondenti che differiscono in due parole/stringhe/byte.

La memorizzazione possono occasionalmente commettere errori a causa, ad esempio, di picchi di tensione elettrica (o difetti). Questi errori possono essere prevenuti utilizzando dei codici di **correzione degli errori**.

Distanza di Hamming: indica il numero di bit corrispondenti che differiscono in due parole.

D. Hamming=2	D. Hamming=4
10011100	11111110
11010100	01100111

Se due parole di codice hanno distanza di Hamming H, saranno necessari H errori (cambiamenti di stato di 1 bit) per convertire una nell'altra.

Dato che dei picchi di tensione potrebbero portare alla generazione di errore nel dato (alta tensione =1 e bassa tensione =0) esistono delle tecniche che mirano a correggere gli errori, esse si dividono sostanzialmente in di due tipi: TECNICHE DI PREVENZIONE e TECNICHE DI CORREZIONE.
L'identificazione dell'errore è molto più semplice della correzione dell'errore, ci sono tecniche semplicissime che in molti casi permettono di identificare l'errore ma non di correggerlo; poi ci sono le tecniche di correzione che si basano sulla ridondanza, esse aumentano l'espressione di un dato è così sono capaci di correggere l'errore, il vantaggio di queste tecniche è molto ovvio ovvero quello di correggere l'errore trovato ma lo svantaggio è altrettanto evidente, si aumenta la memoria necessaria alla rappresentazione del dato stesso; per questo in molti casi ci si accontenta di identificare l'errore in maniera tale da richiedere una copia o nuovamente il dato stesso, ma in casi basta anche applicare una semplice tecnica: IL BIT DI PARITÀ.
Bit di parità: semplice tecnica che non permette di correggere alcun errore, ma permette di identificare errori di un bit, funziona in 3 semplici passaggi:

- Ad ogni parola (WORD sequenza di bit) viene aggiunto un bit di controllo
- Il bit di controllo vale 1 se il numero di bit con valore 1 della parola è dispari
- Il bit di controllo vale 0 se il numero di bit con valore 1 della parola è pari

Questo bit aggiunto permette di fare un semplice controllo, ovvero contare il numero di bit pari a 1 e verificare che successivamente siano ancora pari, questo perché una corruzione del dato potrebbe sfasare il valore dei bit e non si troverebbe più (avendo lo 0 o l'1 alla fine sappiamo perfettamente se la regione a sinistra aveva già o meno un numero di bit pari; ciò ci consente quindi di renderci conto se è cambiato un bit nella word (ma solo uno in caso ne cambino due questo controllo non sarà più valido [è molto raro che cambi più di un bit per cui la tecnica risulta essere molto efficiente]).

Bit di parità: semplice tecnica che non permette di correggere alcun errore, ma permette di identificare errori di un bit.

Funzionamento:

- Ad ogni parola viene aggiunto un bit di controllo
- Il bit di controllo vale 1 se il numero di bit a 1 della parola è dispari
- Il bit di controllo vale 0 se il numero di bit a 1 della parola è pari

10001110	100011100
10001010	100010101

Il **bus** (da una contrazione del latino *omnibus*[1]), in elettronica e informatica, è un canale di comunicazione che permette a periferiche e componenti di un sistema elettronico - come ad esempio un computer - di interfacciarsi tra loro scambiandosi informazioni o dati di vario tipo attraverso la trasmissione e la ricezione di segnali.
In ogni transazione sul bus:

- un dispositivo prende il controllo del bus;
- invia una richiesta (I/O) a un secondo dispositivo;
- svolta la richiesta, il bus viene liberato per un'altra comunicazione.

Il ruolo di un dispositivo può cambiare nel tempo; un dispositivo può comportarsi da master o da slave in contesti differenti. Lo standard che definisce il bus deve fornire le regole per gestire tali condizioni o vietarle. Esistono due diversi meccanismi di temporizzazione dei segnali:

- Protocollo sincrono: è previsto un segnale di sincronizzazione (clock) che permette di gestire la temporizzazione delle comunicazioni.

- Protocollo asincrono: tutta la temporizzazione della comunicazione è gestita dal protocollo stesso attraverso lo scambio dei messaggi.

La scelta dipende da:

- Interfaccia del dispositivo;
- adattare interfacce con differenti velocità;
- tempo totale richiesto per il trasferimento;
- possibilità di rilevare errori;
- Uno schema completamente asincrono è affidabile e flessibile ma le interfacce e la logica di controllo sono molto più complicate da realizzare.

Modalità di connessione

- A stella, in cui ogni nodo della rete è collegato agli altri passando per uno o più concentratori, detti hub, ed esiste un solo percorso che colleghi un nodo a un altro. Ogni nodo ha un solo ramo, collegato a un hub, mentre gli hub hanno almeno due rami di connessione verso altri nodi e altri hub.
- *Daisy-chain*, in cui i nodi sono collegati uno di seguito all'altro, e quindi ogni nodo ha due rami (connessioni) con l'eccezione dei 2 nodi posti alle estremità.
- *Ring*, simile alla rete *daisy-chain* in cui i punti estremi sono anch'essi connessi fra loro, creando quindi un anello.

I bus si suddividono in tre grandi famiglie :

- bus dati

È il bus sul quale transitano le informazioni. È usufruibile da tutti i componenti del sistema, sia in scrittura sia in lettura. È bidirezionale (permette il passaggio dati in più direzioni contemporaneamente).

- bus indirizzi

È il bus (unidirezionale) attraverso il quale la CPU decide in quale indirizzo andare a scrivere o a leggere informazioni; è fruibile anche in lettura dagli altri componenti, mediante l'utilizzo del DMA controllato solo parzialmente dalla CPU e poi dalla DMAC (controller DMA) anche in modo bidirezionale, per dare accesso a dischi, schede grafiche ed eventuali altre risorse.

- bus controllo

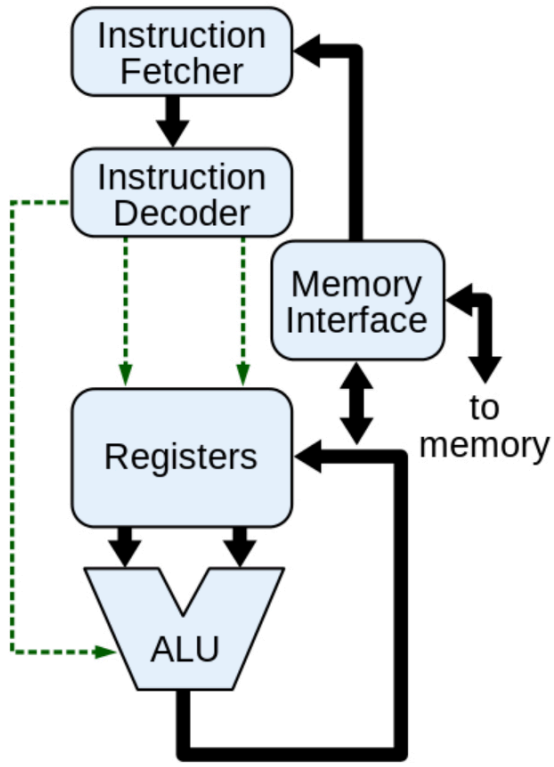
È un insieme di collegamenti il cui scopo è coordinare le attività del sistema; tramite esso, la CPU può decidere quale componente deve scrivere sul bus dati in un determinato momento, quale indirizzo leggere sul bus indirizzi, quali celle di memoria devono essere scritte e quali invece lette, etc.

Il termine **CLOCK**, in elettronica, indica un segnale periodico, generalmente un'onda quadra, utilizzato per sincronizzare il funzionamento dei dispositivi elettronici digitali. Può essere generato da qualsiasi oscillatore, si usa generalmente il tipo a quarzo per la sua alta stabilità di oscillazione. Il segnale è costituito da un livello di tensione che periodicamente in modo regolare fa una rapida transizione dal valore zero ad un valore che generalmente coincide con la tensione di alimentazione del circuito, resta a questo livello per un certo tempo e poi in modo altrettanto rapido ritorna a livello zero, rimane a livello zero per un determinato tempo e poi il ciclo si ripete. La velocità o frequenza di clock è il numero di commutazioni tra i due livelli logici "0" e "1" che i circuiti all'interno di un'unità di calcolo o di un microprocessore sono in grado di eseguire nell'unità di tempo di un secondo, ed è espressa in cicli al secondo, o hertz, e suoi multipli; normalmente per eseguire un'istruzione o una semplice somma sono necessari più cicli di clock.

ciclo di fetch-execute (letteralmente in italiano *ciclo di prelevamento ed esecuzione*) si riferisce alla dinamica generale di funzionamento logico dei processori dei computer, quando devono eseguire un programma software, con le sue singole istruzioni, risiedente nella memoria del calcolatore. In termini generali, un processore esegue iterativamente tre operazioni: *preleva* (*fetch*) un'istruzione dalla memoria primaria, in seguito avviene la decodifica (*decode*) con cui interpreta

l'istruzione, infine la *esegue* (*execute*) combinandola coi dati relativi all'istruzione stessa. In questo modo il processore esegue sequenzialmente istruzioni che danno vita a thread e processi, sotto la supervisione del sistema operativo attraverso lo scheduler. L'estrema velocità di elaborazione dei processori moderni rispetto alla velocità di accesso intrinseca alla memoria rende quest'ultima un collo di bottiglia in fase di progettazione dell'intero sistema di elaborazione.

Nell'architettura dei calcolatori, il **registro istruzione** (spesso abbreviato in **IR**, in inglese ***instruction register***) è un registro della CPU che immagazzina l'istruzione in fase di elaborazione. Ogni istruzione viene caricata dentro il *registro istruzione* che la deposita mentre viene decodificata, la prepara per l'esecuzione e quindi la elabora. Questo processo può necessitare di molti passaggi. Altre CPU più complesse usano il *registro istruzione* della pipeline dati dove ogni fase viene preparata per la successiva operazione, sia essa di decodifica o di esecuzione. Il "registro istruzione", insieme al "contatore di programma" (*program counter*) e al "registro degli indirizzi"(*Address Register*) costituiscono i principali registri di controllo presenti nell'unità di controllo (CU) di ogni CPU.



Schema di un processore

MICROPROGRAMMA

Il processo di scrittura del microcodice per la memoria di controllo dell'unità di elaborazione centrale di un computer è chiamato microprogrammazione. Il microcodice per la memoria di controllo viene generato dopo la configurazione di un computer e la relativa unità di controllo micro programmata. La memoria di controllo è parte dell'unità di controllo che memorizza tutti i micro programmi che non possono essere modificati frequentemente. Ogni riga del microprogramma rappresenta una microistruzione che specifica una o più micro operazioni. Esistono due modi distinti di organizzare le microistruzioni: orizzontale e verticale. Le

microistruzioni orizzontali rappresentano diverse micro-operazioni che vengono eseguite contemporaneamente. Tuttavia, in casi estremi, ogni orizzontalela microistruzione controlla tutte le risorse hardware del sistema. Al contrario, la microistruzione verticale assomiglia al formato del linguaggio macchina convenzionale comprendente un'operazione e pochi operandi. A differenza delle microistruzioni orizzontali, la microistruzione verticale rappresenta singole micro-operazioni.

PROGRAM COUNTER

Nell'architettura dei calcolatori, il ***program counter*** (spesso abbreviato in **PC** e, nelle architetture prive di segmentazione, detto ***instruction pointer***) è un registro della CPU la cui funzione è quella di conservare l'indirizzo di memoria della prossima istruzione (in linguaggio macchina) da eseguire. È un registro *puntatore* cioè punta a un dato che si trova in memoria all'indirizzo corrispondente al valore contenuto nel registro stesso. Su alcune architetture il *program counter* conserva invece l'indirizzo dell'istruzione in via di esecuzione. Il *program counter* è utilizzato nel ciclo *fetch-execute* che costituisce la dinamica fondamentale nel funzionamento di un computer; tale ciclo è una ripetizione infinita dei seguenti passi:

1. caricamento dell'istruzione riferita dal program counter;
2. aggiornamento (incremento) del program counter, in modo che contenga l'indirizzo dell'istruzione successiva;
3. esecuzione dell'istruzione caricata.

Nel normale ciclo *fetch-execute*, quindi, il *program counter* viene incrementato automaticamente. In aggiunta, tutti i linguaggi macchina forniscono una o più istruzioni che *modificano* esplicitamente il *program counter* se vale una certa condizione (per esempio se l'accumulatore ha tutti i bit impostati a 0). Queste istruzioni consentono al programma di "saltare" a una istruzione di programma che non sia quella immediatamente successiva a quella appena eseguita, e forniscono quindi lo strumento fondamentale sul quale sono realizzate le strutture di controllo dei linguaggi di programmazione.

Il **prefetch** (dal latino *pre*, "prima" e *fetch*, in inglese "andare a prendere") è una tecnica usata nei microprocessori per accelerare l'esecuzione dei programmi riducendo gli stati di attesa (in inglese *wait states*).
I moderni microprocessori sono infatti molto più veloci della memoria RAM che contiene i programmi e i dati, perciò le istruzioni dei programmi non possono essere lette in modo sufficientemente veloce da tenere il processore occupato.
L'attività di prefetch potrebbe consistere nel precaricare la prossima istruzione del programma durante l'esecuzione dell'istruzione corrente; in effetti a questo si limitava nelle sue prime implementazioni (come nel caso dell'Intel 8086). Quando però l'istruzione da eseguire era un'istruzione di salto, il lavoro compiuto si rivelava del tutto inutile e la coda di prefetch veniva svuotata, con perdita di efficienza.
Nei processori più evoluti l'attività di prefetch è realizzata mediante un complesso algoritmo di predizione, con il quale il processore cerca di individuare la giusta istruzione successiva da caricare. Nel caso della famiglia Intel, questa tecnica prende il nome di *Branch Target Buffer* e può arrivare a predizioni esatte nel 90% dei casi.

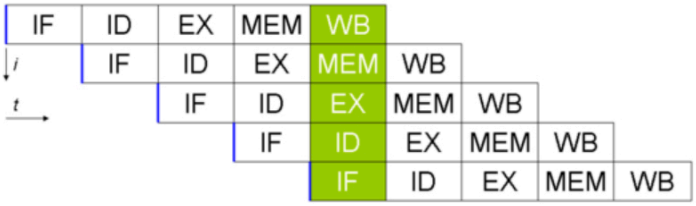
La ***pipeline* dati**, in Informatica, è una tecnologia utilizzata nell'architettura hardware dei microprocessori dei computer per incrementare il *throughput*, ovvero la quantità di istruzioni eseguite in una data quantità di tempo, parallelizzando i flussi di elaborazione di più istruzioni.
L'elaborazione di un'istruzione da parte di un processore si compone di cinque passaggi fondamentali:

1. IF (*instruction fetch*): lettura dell'istruzione da memoria;
2. ID (*instruction decode*): decodifica istruzione e lettura operandi da registri;
3. EX (*execution*): esecuzione dell'istruzione;
4. MEM (*memory*): attivazione della memoria (solo per certe istruzioni);
5. WB (*write back*): scrittura del risultato nel registro opportuno;

Praticamente ogni CPU in commercio è gestita da un clock centrale e ogni operazione elementare richiede almeno un ciclo di clock per poter essere eseguita. Le prime CPU erano formate da

un'unità polifunzionale che svolgeva in rigida sequenza tutti e cinque i passaggi legati all'elaborazione delle istruzioni. Una CPU classica richiedeva quindi almeno cinque cicli di clock per eseguire una singola istruzione. Con il progresso della tecnologia si è potuto integrare un numero maggiore di transistor in un microprocessore e quindi si sono potute parallelizzare alcune operazioni riducendo i tempi di esecuzione. La pipeline dati rappresenta la massima parallelizzazione del lavoro di un microprocessore.
Una CPU con *pipeline* è composta da cinque stadi specializzati, capaci di eseguire ciascuno un'operazione elementare di quelle sopra descritte. La CPU lavora come in una catena di montaggio cioè ad ogni stadio provvede a svolgere in maniera sequenziale un solo compito specifico per l'elaborazione di una certa istruzione. Quando la catena è a regime, ad ogni ciclo di clock dall'ultimo stadio esce un'istruzione completata. Nello stesso istante ogni unità sta però elaborando in parallelo i diversi stadi di successive altre istruzioni. In sostanza quindi si guadagna una maggior velocità di esecuzione a prezzo di una maggior complessità circuitale del microprocessore, che non deve essere più composto da una sola unità, ma da cinque unità che devono collaborare tra loro.

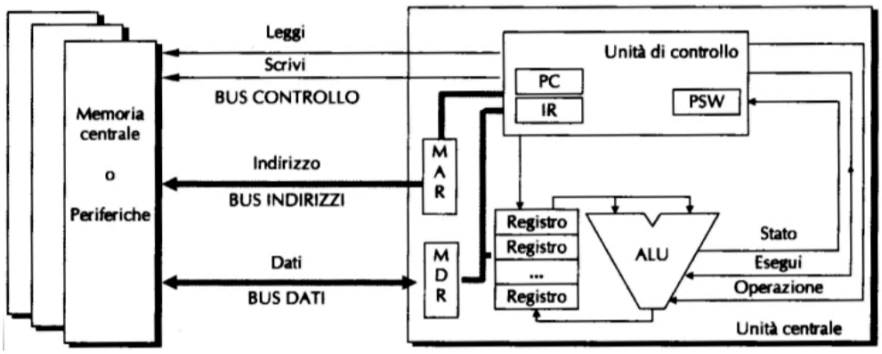
Esecuzione delle istruzioni in un microprocessore senza *pipeline*



Esecuzione delle istruzioni in un microprocessore con *pipeline*

Il concetto di **pipeline** (in inglese, *tubatura* — composta da più elementi collegati — o *condotto*) viene utilizzato per indicare un insieme di componenti software collegati tra loro in cascata, in modo che il risultato prodotto da uno degli elementi (output) sia l'ingresso di quello immediatamente successivo (input).

Esempio: esecuzione di una istruzione



1) la CPU mette il valore di PC (indirizzo della prossima istruzione da leggere dalla memoria) su MAR e attiva la linea Leggi;
2) la memoria attraverso il bus indirizzi accede a MAR e, una volta reperito quanto richiesto, lo scrive su MDR attraverso il bus dati;
3) la CPU copia su IR il valore di MDR e decodifica l'istruzione;
4) l'istruzione passa in esecuzione sulla ALU;
5) se l'istruzione prevede la lettura di operandi dalla memoria, questi devono essere caricati sui registri; per ciascun operando da reperire:
    5.1) la CPU mette l'indirizzo dell'operando su MAR e attiva la linea Leggi;
    5.2) la memoria attraverso il bus indirizzi accede a MAR e, una volta reperito quanto richiesto, lo scrive su MDR attraverso il bus dati;
    5.3) la CPU copia sul registro destinazione il valore dell'operando che è in MDR;
6) terminata l'esecuzione la CPU copia sul registro destinazione il valore prodotto dalla ALU; se è prevista scrittura in memoria del valore calcolato:
    6.1) la CPU mette l'indirizzo della cella di destinazione su MAR e il risultato su MDR e attiva la linea Scrivi;
    6.2) la memoria attraverso il bus indirizzi accede a MAR, attraverso il bus dati a MDR e, una volta reperito il valore in MDR, lo scrive sulla propria cella interna indicata da MAR;
7) Si ritorna al punto 1 dopo aver aggiornato il valore di PC (prossima istruzione da eseguire).