Architetture di calcolo 1

C'erano le calcolatrici che a differenza del computer aveva bisogno di un umano a guidarlo attraverso delle istruzioni di controllo.
Il computer una volta impostato non ha più bisogno di avere dei controlli.
Il computer è realizzato mediante semplici circuiti elettronici, che però non sono in grado di eseguire qualsiasi tipo di istruzione, per capirci non possiamo installarci python e eseguirlo … per cui come fanno i computer a eseguire programmi se i circuiti non sono in grado di farlo ?
Il computer esegue il programma ad alto livello (come programmi python) mediante scomposizione in istruzioni più semplici fino ad arrivare a istruzioni elementari che la macchina è quindi in grado di capire, di fatto la macchina è in grado di capire solo quelle istruzioni esplicate in LINGUAGGIO MACCHINA (sequenza molto limitata di istruzioni semplici in cui tutte le istruzioni complesse sono ricondotte [indica il linguaggio in cui sono scritti i programmi eseguibili per computer: può venire classificato come linguaggio di programmazione, sebbene basato su un alfabeto detto *binario* in quanto comprende due soli simboli, generalmente indicati con 0 e 1]).
(!!!*** la macchina per eseguire i comandi necessita che questi siano esplicati in linguaggio macchina, per cui avviene una vera e propria traduzione dal programma all'istruzione, ciò avviene grazie ai linguaggi di basso livello più nello specifico grazie all'assembly, un linguaggio un pelo sopra l'architettura della macchina che grazie a delle semplici istruzioni riesce a impartire ordini alle diverse componenti della macchina, le quali prima di essere attuate vengono tradotte tramite l'assembler in linguaggio macchina; la catena è per cui LINGUAGGIO ALTO LIVELLO (in cui programmiamo [esempio python {linguaggio interpretato passo passo in istruzioni più semplici}] ———> LINGUAGGIO BASSO LIVELLO [fino ad arrivare all'assembly] ———> TRADUZIONE [tramite assembler] ———> LINGUAGGIO MACCHINA ———> ESECUZIONE ***!!!).

- **Computer**: una macchina che può risolvere problemi eseguendo le istruzioni che le vengono assegnate.
- **Programma**: sequenza di istruzioni che descrive come portare a termine un dato compito.
- I circuiti elettronici di un computer possono riconoscere ed eseguire direttamente soltanto un *insieme limitato di istruzioni elementari* in cui tutti i programmi devono essere convertiti prima di poter essere eseguiti.

Quali sono le istruzioni elementari che una macchina può eseguire ?
Sono istruzioni molto elementari come la somma di due numeri, controllare se un numero è 0 oppure copiare i dati da una parte all'altra della macchina; quindi sono operazioni davvero molto elementari che di solito non sono presenti nei programmi di più alto livello che andiamo a compilare, di fatto noi non ci limitiamo a queste semplici operazioni ma facciamo cose molto più complicate che poi vengono tradotte in queste istruzioni più semplici.
Il LINGUAGGIO MACCHINA è l'unico linguaggio con cui si può comunicare con il computer.

- Esempi di istruzioni elementari:
  - sommare due numeri
  - controllare se un numero vale zero
  - copiare una porzione di dati da una parte all'altra della memoria.

- L'insieme di queste istruzioni elementari forma il cosiddetto **linguaggio macchina**, attraverso il quale è possibile comunicare con il computer.

Coloro che progettano e realizzano il computer devono innanzi tutto decidere quali sono le istruzioni che il computer dovrà implementare, esempio chi ha progettato il computer x86 (progenitore dei computer odierni) ha dovuto decidere in primis quali erano le istruzioni semplici che la macchina doveva supportare, quindi quale era il linguaggio macchina di quella architettura; stabilito il set di istruzioni (dipendente anche dalle componenti della macchina) si è definito sostanzialmente il comportamento della macchina, quindi come si può chiedere alla macchina di lavorare ovvero con quali istruzioni in linguaggio macchina si chiede al computer di funzionare.
Il linguaggio macchina viene però utilizzato molto raramente e da una cerchia molto piccola di programmatori nel mondo, in nessun corso di laurea si spiega nemmeno il linguaggio macchina perché non è l'obbiettivo di questi corsi.

- Chi progetta un computer deve decidere quali istruzioni costituiranno il suo linguaggio macchina.

- Per ridurre la complessità e il costo dei dispositivi elettronici, si cerca di progettare le più semplici istruzioni primitive che siano compatibili con il tipo di utilizzo e i requisiti prestazionali del computer.

- Poiché quasi tutti i linguaggi macchina sono semplici ed elementari, risulta difficile e noioso utilizzarli.

Esiste però un linguaggio simbolico che è quasi a livello del linguaggio macchina: l'ASSEMBLY, che è una rappresentazione simbolica del linguaggio macchina, anche quest'ultimo però viene utilizzato poco è molto raramente, è però molto utile capirlo poiché così si capisce veramente come funziona la macchina; principalmente in assembly scrivono i programmatori che si occupano di cose molto specifiche come programmare i driver dei dispositivi, il programmatore normale utilizza linguaggi ad alto livello.
Visto che è noioso scrivere e programmare in linguaggio macchina, sin da subito ovvero nei primi anni 60 si è capito che i computer dovevano essere progettati sulla base dell'APPROCCIO STRUTTURALE.
L'approccio strutturale prevede che i computer non siano progettati come un hardware su cui sopra è installato un software, ma come un insieme di livelli (LAYER) uno sull'altro che partono dal

livello fisico (hardware) ed introducendo man mano delle astrazioni ci si avvicina sempre più all'utente finale che poi utilizzerà il computer.
Questo approccio prevede un uso variabile di più livelli (esempio 3-4-5 o 6).

- Per gestire più agevolmente la complessità dei computer e progettarli in modo sistematico e organizzato, i computer sono strutturati come una serie di *livelli di astrazione*, ciascuno dei quali è costruito sulla base di quello sottostante.

- Questo modo di concepire l'architettura dei computer è detto **approccio strutturale.**

Per capire meglio questa idea, partiamo da una semplice considerazione: i computer possono fare qualcosa x (x rappresenta la possibilità di eseguire un istruzione in linguaggio macchina) ma non vogliamo progettare questa cosa x in linguaggio macchina, quindi come si fa a ovviare a questo problema, ovvero rispondere alla differente necessità degli utenti rispettando le possibilità del computer ?
La soluzione è quella di definire un nuovo insieme di istruzioni che sia più comodo da usare rispetto a quello previsto dal linguaggio macchina (che chiameremo L1) che offre un insieme di istruzioni più completo e semplice da usare rispetto al linguaggio macchina (L0).
Supponiamo che io abbia scritto un programma in L1 e voglia eseguirlo, come posso fare visto che la macchina capisce solo L0?
Ci sono in realtà due approcci per farlo: la TRADUZIONE (ovvero prendiamo il programma scritto in L1 e traduciamo ogni istruzione in un insieme di istruzioni scritte in L0 [tipicamente necessita di tante istruzioni] così che queste istruzioni possano poi essere eseguite nella macchina{*** prima di essere eseguito il programma deve essere prima interamente tradotto***}) e l'INTERPRETAZIONE (per interpretare un programma scritto in L1 devo prima scrivere un programma in L0 chiamato INTERPRETE che accetta come input il programma scritto in L1, l'interprete sarà quindi capace di leggere le istruzioni e poi eseguirle sulla macchina in L0 {*** il programma inizia a essere eseguito subito ma con una velocità diversa poiché c'è un secondo programma che traduce in tempo reale istruzione per istruzione ***}).
È più efficiente la traduzione o l'interpretazione ?
Di solito la traduzione prende anche il nome di compilazione risulta essere più efficiente, d'altra parte l'interpretazione risulta essere più comoda in molte situazioni; entrambi gli approcci sono validi e si usano in differenti situazioni.
La traduzione è quanto più difficile quanto più lontani sono i linguaggi.
Per consentire quindi una traduzione o una interpretazione il quanto più efficiente possibile è necessario che i due linguaggi siano molto simili tra loro.
Potrebbe capitare che il linguaggio L1 quindi per quanto sia più ad alto livello di L0 sia comunque a un livello troppo lontano dall'uomo, ovvero è ancora troppo astratto per essere letto con facilità, ecco il motivo per cui vengono aggiunti ulteriori livelli che mirano progressivamente a aumentare la facilità di leggibilità.
Viene successivamente creato quindi un linguaggio L2 che questa volta è basato sul linguaggio L1 ma è a sua volta ancora di più alto livello, in questo modo sarà facile passare da L2 a L1 e da L1 a L0 anche se passare da L2 a L0 sarebbe stato impossibile.
(esempio: prendiamo 3 lingue ovvero italiano, spagnolo e portoghese, la strada più semplice per passare dall'italiano che rappresenta il mio L0 al portoghese che rappresenta il mio L2 sarà quella di passare prima per lo spagnolo che rappresenta il mio L1 proprio perché lo spagnolo rappresenta una lingua intermedia tra le due e che discostandosi meno renderà la traduzione più semplice).
(*** ogni volta che passiamo da un linguaggio di livello superiore ad un linguaggio di livello inferiore dobbiamo necessariamente o tradurlo o interpretarlo***)

Continuando con questo ragionamento immetto una serie di linguaggi, ciascuno più pratico di quelli sottostanti, arrivo alla fine ad un linguaggio di livello n (dipendente dal numero di passi necessari) che è utilizzabile dall'utente finale.
In questo modo andremo a definire una serie di strati o livelli che andranno a formare l'approccio alla macchina in cui : il linguaggio più alto è quello più sofisticato e vicino all'uomo mentre il linguaggio più basso, di livello 0 è quello più semplice e vicino alla macchina.
I programmi di livello 0 possono essere direttamente eseguiti sui circuiti digitali cioè sull'hardware, al livello L1 abbiamo la macchina virtuale L1 che è lievemente più astratta rispetto al hardware e qui i programmi possono essere interpretati sulla macchina L0 o tradotti, a seguire abbiamo L2 in cui c'è una macchina virtuale L2 e il codice scritto in L2 può essere eseguito da una qualsiasi macchina di livello sottostante (non obbligatoriamente il subito sottostante [di solito però non si arriva direttamente in L0 ma si effettuano una serie di passaggi intermedi un po' più rigidi]) … e così via fino ad arrivare al livello n.

- **Problema**: Gli utenti vogliono fare *X,* ma i computer possono fare soltanto *Y*

- **Soluzione**: Definizione di un *nuovo insieme d'istruzioni* che sia più comodo da utilizzare rispetto a quanto non lo siano le istruzioni predefinite del linguaggio macchina.

- Anche questo nuovo insieme d'istruzioni forma un linguaggio (che chiamiamo **L1**), allo stesso modo in cui le istruzioni macchina formavano a loro volta un linguaggio (che chiamiamo **L0**).

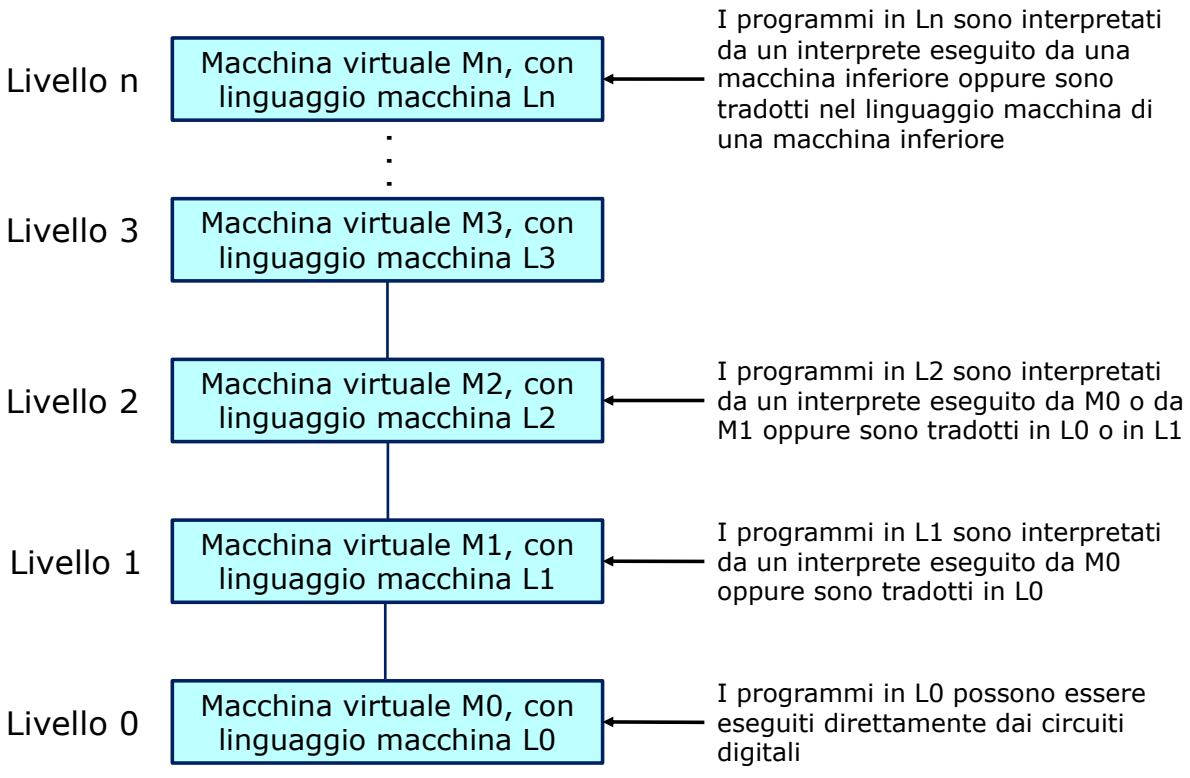
- Come eseguire un programma scritto in L1?

  - **Traduzione**: Sostituire, in una fase iniziale, ogni istruzione del programma scritto in L1 con un'equivalente sequenza di istruzioni in L0. Il programma che ne risulta è costituito interamente da istruzioni di L0 e può essere eseguito dal computer al posto del programma L1 originale.

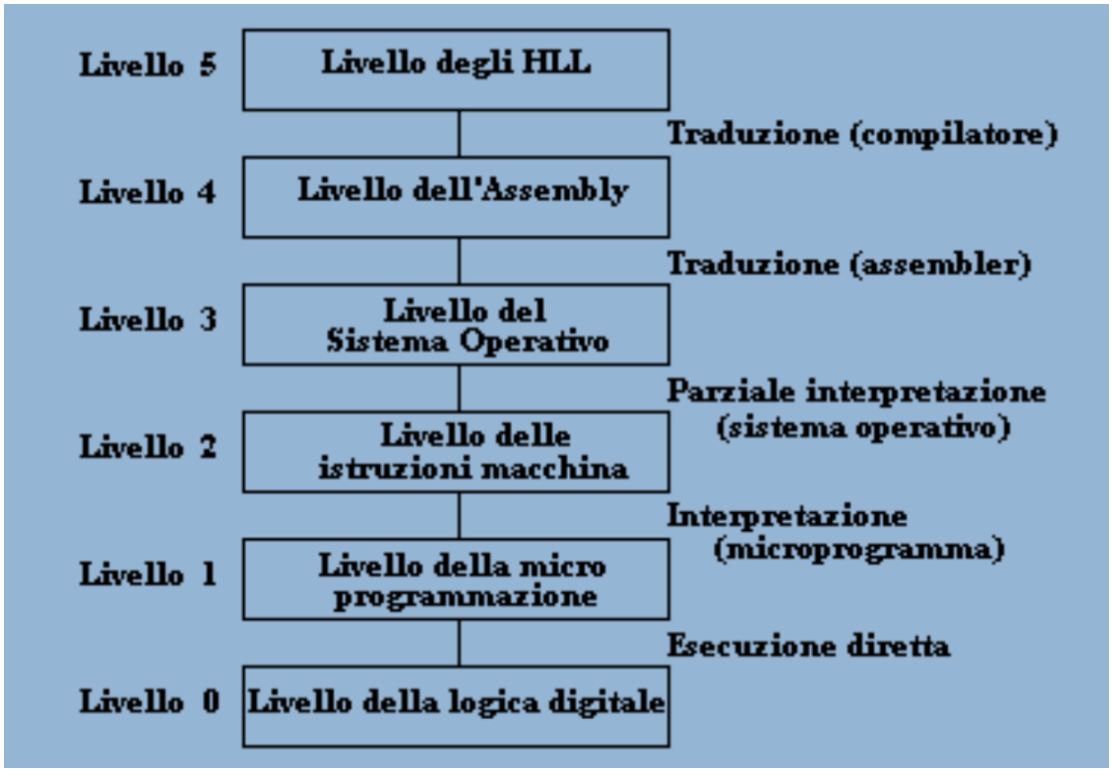
  - **Interpretazione**: Scrivere un programma in L0 (detto *interprete*) che accetta come dati d'ingresso programmi in L1. Tale *interprete* esegue un programma in L1 esaminando un'istruzione alla volta e sostituendola direttamente con l'equivalente sequenza di istruzioni L0.

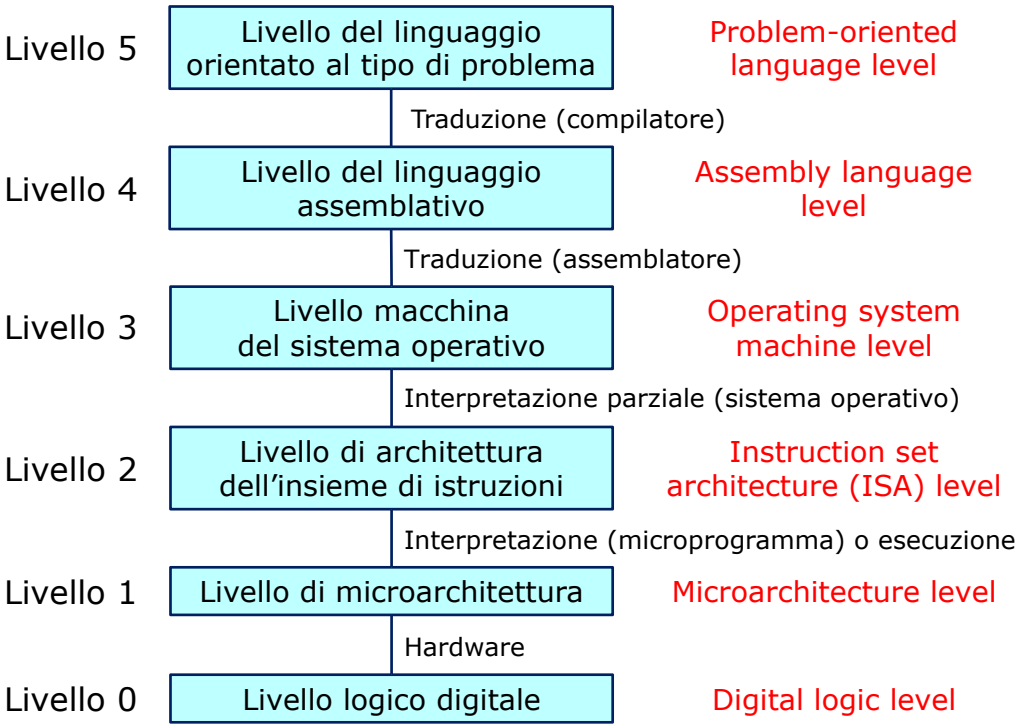
- Si può immaginare l'esistenza di un ipotetico computer, o **macchina virtuale**, il cui linguaggio macchina sia L1. Chiamiamo questa macchina virtuale M1, e chiamiamo M0 la macchina virtuale corrispondente al linguaggio L0.

- Per rendere la traduzione o l'interpretazione utilizzabili in pratica, i linguaggi L0 e L1 non devono essere troppo diversi fra loro. Per la maggior parte delle applicazioni questo vincolo fa sì che L1, pur essendo migliore di L0, sia spesso ancora lontano dal linguaggio ideale.
- Per risolvere questo problema, si può definire un nuovo insieme d'istruzioni che sia, rispetto a L1, maggiormente orientato agli utenti piuttosto che alle macchine.
- La definizione di una serie di linguaggi, ciascuno dei quali più pratico da utilizzare rispetto al precedente, può continuare indefinitamente finché non se ne ottenga uno sufficientemente adeguato.
- Ciascun linguaggio utilizza il precedente come base; un computer che usa questa tecnica può quindi essere immaginato come una serie di **strati** o **livelli** disposti l'uno sopra l'altro.
- Il livello, o linguaggio, che si trova più in basso è il più semplice, mentre quello più in alto è il più sofisticato.



Quanti livelli ci sono in una macchina vera ?
Mediamente le macchine moderne sono macchine a 6 livelli (anche se possono essere anche di diverso numero ma sicuramente l'architettura rimane stratificata in maniera tale che ogni livello aggiunge un DELTA [che rappresenta una sorta di avvicinamento dalla macchina verso l'utente che però rispetti la fattibilità dell'operatività da parte della macchina stessa {non deve allontanarsi troppo dalla macchina e rischiare di rendere impossibile la traduzione a linguaggio macchina}]), i 6 livelli sono:





Il livello 0 (LOGICO DIGITALE) rappresenta e si occupa del hardware ed è composto dagli oggetti più semplici che si possano trovare sul computer, essi prendono il nome di porte logiche (booleane)[AND, OR e NOT], sono dei gate caratterizzati dalla capacità di trasformare valori di input in valori di output (tipicamente 2 valori di input è uno di output).
Questi gate rappresentano le funzioni booleane ovvero funzioni che operano su valori 0 (falso/non passa corrente) e 1(vero/passa corrente), ovvero i numeri binari, e generare un output in funzione di questi ultimi [AND genera 1 se e solo se ha in ingresso due valori pari a 1, OR genera 0 se e solo se ha in ingresso due valori pari a 0, la porta NOT è un inverter ovvero restituisce come output il complementare dell'input; QUESTE PORTE POSSONO ESSERE COMBINATE TRA DI LORO PER FORMARE OMPONENTI PIÙ COMPLESSI che ad esempio siano capaci di salvare un dato in memoria e mettendo assieme 8 bit di memoria formiamo un byte oppure combinare tante memorie da bit per formare i REGISTRI oppure confrontare dei dati tra loro].
Dall'utilizzo di più porte logiche formiamo degli operatori più complessi che comunque si trovano nel livello 0 perché formati sempre da operatori elementari ma che siano capaci di gestire delle operazioni un attimino piu avanzate.
Questo livello logico è caratterizzato dal fatto che permette di eseguire le operazioni in un linguaggio il più basso possibile il cui senso è legato strettamente al hardware della macchina e quindi alla sua architettura.

- **Livello logico digitale** (**livello 0**): rappresenta l'hardware della macchina, i cui circuiti eseguono i programmi scritti nel linguaggio macchina del livello 1.
- Gli elementi di base del livello 0 sono le **porte** (**gate**).
- Ciascuna porta è dotata di uno o più input digitali (segnali corrispondenti ai valori 0 o 1) e calcola in output una semplice funzione dei valori d'ingresso, per esempio AND od OR.
- E possibile combinare un piccolo numero di porte per formare una memoria a 1 bit, che può memorizzare i valori 0 e 1.
- Combinando le memorie a 1 bit in gruppi, per esempio di 16, 32 o 64, è possibile formare i cosiddetti **registri**.
- Ogni registro può contenere un numero il cui valore può variare fino a un certo limite.

Livello 1 di micro architettura, di fatto permette l'esecuzione di micro programmi che ci permettono di gestire il flusso di dati nella macchina (GPU, CPU, RAM ecc).
Introduce qui il discorso della macchina di von Neumann che riprenderà poi nella prossima lezione.
Tra le componenti abbiamo l'ALU (arithmetic Logic unit)quella parte del processore che si occupa di fare i calcoli, esso caratterizza il livello 1 ed e deputato a operazioni logico matematiche e la gestione dei dati secondo uno schema detto DATA PATH.
In questo livello si spiga come si spostano i dati da un componente all'altro (esempio: se io devo sommare due numeri, a questo livello vengono gestiti i processi di estrazione dei dati, ovvero viene gestita l'area di memoria in cui si deve accedere per estrarre il dato e anche come accedervi, viene gestito il salvataggio del dato della somma e la preparazione dell'area di memoria che dovrà accoglierlo).
Il percorso dei dati è guidato da un mini programma, ovvero un insieme di istruzioni native della macchina che gestiscono e stabiliscono come il flusso di dati e le istruzioni semplici devono essere eseguite.
In alcuni casi però questo mini programma non esiste nella macchina, per cui tutti questi processi sono controllati direttamente dal hardware.

- **Livello di microarchitettura** (**livello 1**)**:** contiene una memoria locale, formata da un gruppo di registri (in genere da 8 a 32), e un circuito chiamato ALU (*Arithmetic Logic Unit,* unità aritmetico-logica), capace di effettuare semplici operazioni aritmetiche e logiche.

- I registri sono connessi alla ALU per formare un **percorso dati** (**data path**) lungo il quale si spostano i dati. L'operazione base del percorso dati consiste nel selezionare uno o due registri, permettere alla ALU di operare su di loro (per esempio sommandoli) e memorizzare infine il risultato in uno dei registri.

- In alcune macchine, le operazioni del percorso dati sono controllate da un programma chiamato **microprogramma,** mentre su altre il percorso dati è controllato direttamente dall'hardware.

Livello 2 (ISA) si occupa di fornire al programmatore della macchina le istruzioni fondamentali di basso livello utili alla macchina stessa.
È l'aspettò che il computer assume agli occhi di un programmatore di linguaggio macchina.
Spesso si dice che il livello ISA corrisponde al livello di programma in assembly, ciò non è necessariamente vero però di fatto il linguaggio assembly è un linguaggio che è capace di esprimere istruzioni quasi corrispondenti 1 a 1 alle istruzioni di livello ISA; di fatto molto spesso le istruzioni di questo livello sono rappresentate in linguaggio assembly poiché meno astratto e più facile da comprendere e ricordare.
Per programmare in questo livello dobbiamo conoscere l'architettura del computer e delle sue componenti fondamentali come: MEMORIE , PROCESSORE e REGISTRI.

- **Livello di architettura dell'insieme d'istruzioni** (**livello 2**): Più noto con l'acronimo inglese di **livello ISA** *(Instruction Set Architecture level)* si può definire come l'aspetto che il computer assume agli occhi di un programmatore in linguaggio macchina.

- Ogni diversa CPU ha un proprio ISA e quindi istruzioni diverse spesso non compatibili tra loro.

- Il termine **assembly** si riferisce ad un linguaggio costituito da codici mnemonici corrispondenti alle istruzioni ISA.

- Al fine di produrre codice di livello ISA, si deve conoscere il modello di memoria, quali registri ci sono, quali sono i tipi di dati e di istruzioni disponibili, e così via; l'insieme di tutte queste informazioni definisce il livello ISA.

Livello 3 abbiamo il Sistema Operativo abbiamo un'astrazione ulteriore ovvero la gestione della memoria virtuale, il parallelismo dei processi, il multi-treath, il multi-processing, lo scarpino dei processi (astrazioni che permettono di far girare programmi complessi).
Spesso questo livello viene visto come un livello più basso, poiché le istruzioni che questo livello contiene possono essere eseguite anche direttamente a livello sottostante, questo perché un

linguaggio scritto in un certo linguaggio può essere tradotto o interpretato in un qualsiasi linguaggio di livello sottostante, ciò avviene spesso a livello dei sistemi operativi.
Questo livello viene fondamentalmente sfruttato per aggiungere un livello di “semplicità” alle istruzioni ma sopratutto per lanciare più programmi in esecuzione contemporaneamente; di fatto si aggiungono dei concetti fondamentali che però nelle macchine più vecchie non esistevano e che nelle macchine moderne è diventato prassi, questi concetti si basano sulla MULTI-PROGRAMMAZIONE, la GESTIONE DELLA CONCORRENZA (più processi devono evitare di scrivere contemporaneamente sullo stesso file e generare conflitti), la GESTIONE DELLA MEMORIA VIRTUALE (fa credere alla macchina che la quantità di ram presente è molto maggiore rispetto a quella presente veramente) e altri concetti che studieremo successivamente.
Questo livello è stato di fatto aggiunto successivamente per dare alla macchina una maggiore flessibilità d’uso ed efficienza.
Come fa il computer a eseguire contemporaneamente più programmi ?
Il concetto della multiprogrammazione ha rivoluzionato il concetto del computer, ciò avviene sfruttando al meglio le risorse della macchina, tutti i programmi fanno sostanzialmente due cose (CALCOLI E INPUT/OUTPUT ovvero calcolano tramite la cpu i/o e scrivono i risultati [nel momento in cui sovrascrive il disco in realtà la cpu è ferma, ed è in questo momento che si “sovrappongono” i processi da svolgere, ciò ottimizza lo sfruttamento della cpu ***CONTEXT SWITCH***]).
Il numero ideali di programmi da eseguire è dipendente dalla macchina e dai tipi di processi (per intenderci è un lievemente meno del numero di processi capace di mandare in crush il computer di fatto in questo momento si raggiunge il massimo di produttività della macchina che poi aumentando ancora sovraccarica di processi la cpu e le componenti e per questo va in crush).
I livelli fino al livello 3 sono per programmatori di sistema, chi programma fino a qui deve conoscere i linguaggi specifici della macchina.

- Il **livello macchina del sistema operativo (livello 3)** è generalmente un livello *ibrido*. La maggior parte delle istruzioni nel suo linguaggio fa parte anche del livello ISA.
- Inoltre, vi è un insieme di nuove istruzioni, una diversa organizzazione della memoria e la capacità di eseguire programmi in modo concorrente, oltre a varie altre funzionalità.
- I nuovi servizi aggiunti nel livello 3 sono realizzati da un interprete eseguito al livello 2, storicamente chiamato *sistema operativo*. Le istruzioni del livello 3, identiche a quelle del livello 2, sono eseguite direttamente dal microprogramma (o dai circuiti elettronici) e non dal sistema operativo.
- In altre parole, alcune delle istruzioni del livello 3 sono interpretate dal sistema operativo, mentre altre sono interpretate in modo diretto dal microprogramma; per questo motivo tale livello viene detto ibrido.

- Tra i livelli 3 e 4 vi è una spaccatura fondamentale. I tre livelli inferiori sono concepiti principalmente per eseguire interpreti e traduttori necessari come supporto per i livelli più alti.
- Questi interpreti e traduttori sono scritti dai cosiddetti **programmatori di sistema,** specializzati nella progettazione e nell’implementazione di nuove macchine virtuali.

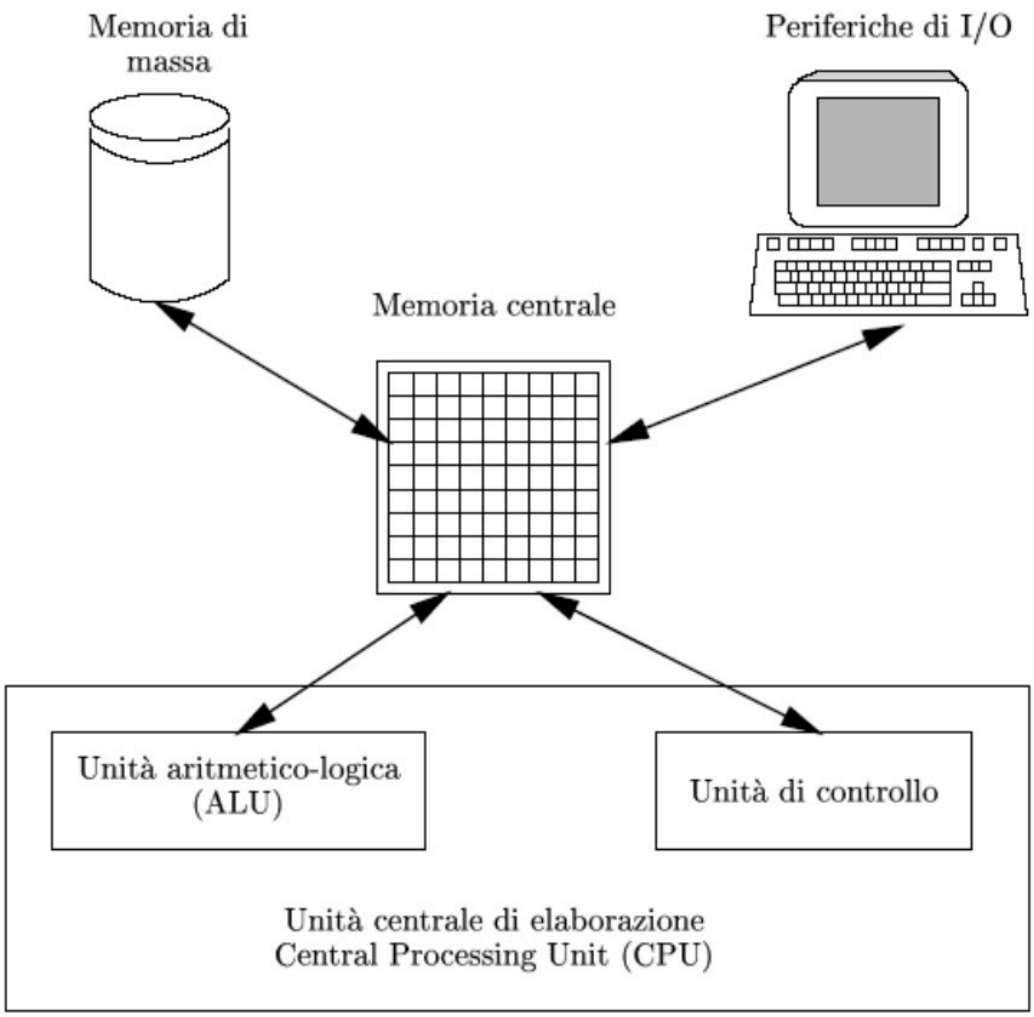
- Tra i livelli 3 e 4 vi è una spaccatura fondamentale. I tre livelli inferiori sono concepiti principalmente per eseguire interpreti e traduttori necessari come supporto per i livelli più alti.
- Questi interpreti e traduttori sono scritti dai cosiddetti **programmatori di sistema,** specializzati nella progettazione e nell'implementazione di nuove macchine virtuali.
- Il livello 4 e quelli superiori sono invece pensati per i programmatori che devono risolvere problemi applicativi.
- I linguaggi macchina dei livelli 1, 2 e 3 sono numerici; i loro programmi consistono quindi in lunghe serie di numeri, certamente più adatte alle macchine che agli esseri umani.
- A partire dal livello 4 i linguaggi contengono invece parole e abbreviazioni umanamente comprensibili.

Livello 4 abbiamo l'assembly, usando traduzione o interpretazione è in grado di generare codice utilizzabile direttamente sulla macchina.
Qui si inizia a programmare la macchina per scopi specifici delle applicazioni a differenza dei livelli precedenti.
Già da qui si può usare il linguaggio assembly per programmare applicativi o applicazioni, anche se molto raro che ciò accada infatti si usa principalmente solo quando l'efficienza diventa un fattore fondamentale poiché scrivere direttamente in questo linguaggio bypassando l'assembler (ASSEMBLATORE programma di traduzione in assembly) si aumenta notevolmente l'efficienza del programma (assembly scritto a mano è circa 3 volte più veloce di un assembly scritto tramite l'assembler).
Di base questo livello fornisce strumenti per scrivere programmi in livelli sottostanti in un modo più intuitivo e semplice.
Si sottolinea che di solito in questo livello raramente si scrive a mano ma si sfrutta l'assembler per generare il codice per questo livello da un codice di livello superiore.

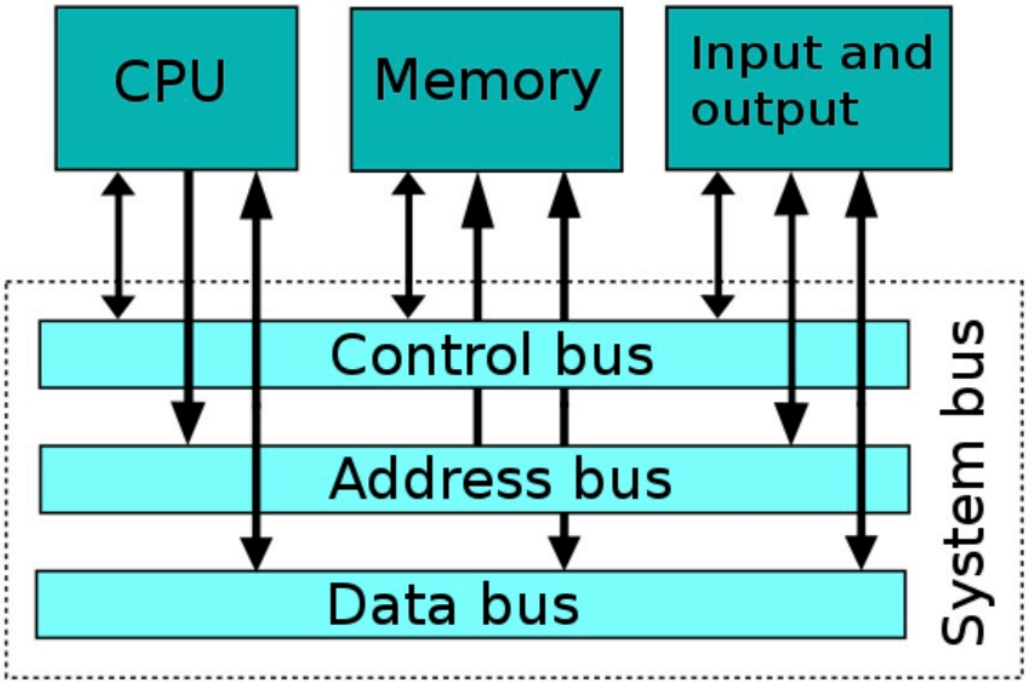
- **Livello del linguaggio assemblativo** (**livello 4**): fornisce ai programmatori un modo per scrivere programmi per i livelli 1, 2 e 3 in una forma meno difficile rispetto a quella dei linguaggi delle rispettive macchine virtuali.
- I programmi in linguaggio assemblativo sono inizialmente tradotti nei linguaggi dei livelli 1, 2 o 3 e successivamente interpretati dall'appropriata macchina virtuale o reale.
- Il programma che esegue la traduzione è chiamato **assemblatore.**

Livello 5 è il livello dei linguaggi di programmazione.
I linguaggi presenti in questo livello vengono eseguiti in assembly mediante traduzione da parte dell'assembler.
Qui sono presenti tutti i linguaggi di alto livello.

- **Livello del linguaggio orientato al tipo di problema** (**livello 5**): consiste generalmente di linguaggi, spesso chiamati **linguaggi ad alto livello,** definiti per essere utilizzati dai programmatori di applicazioni.
- Ne esistono letteralmente a centinaia; alcuni fra i più conosciuti sono C, C++, Python e Java.
- I programmi scritti in questi linguaggi sono generalmente tradotti al livello 3 o al livello 4 da un traduttore detto **compilatore;** in casi meno frequenti è anche possibile che essi siano interpretati.
- I programmi in Java, per esempio, di solito sono tradotti inizialmente in un linguaggio di tipo ISA chiamato *Java byte code,* che viene successivamente interpretato.
- Il concetto chiave è che i computer sono progettati come una serie di livelli, ciascuno costruito su quelli che lo precedono.
- Ciascun livello rappresenta una diversa astrazione, caratterizzata dalla presenza di oggetti e operazioni diversi.
- Progettando e analizzando i computer in questo modo è possibile tralasciare temporaneamente i dettagli irrilevanti, riducendo di conseguenza un soggetto complesso in qualcosa di più facile comprensione.



Architettura di von Neumann



L'**architettura di Von Neumann** è una tipologia di architettura *hardware* per *computer* digitali programmabili a programma memorizzato la quale condivide i dati del programma e le istruzioni del programma nello stesso spazio di memoria, contrapponendosi all'architettura Harvard nella quale invece i dati del programma e le istruzioni del programma sono memorizzati in spazi di memoria distinti.
Lo schema si basa su cinque componenti fondamentali:

1. **Unità centrale di elaborazione (CPU)**, che si divide a sua volta in unità aritmetica e logica (ALU o unità di calcolo) e unità di controllo;
2. **Unità di memoria**, intesa come memoria di lavoro o memoria principale (RAM, Random Access Memory);
3. **Unità di *input***, tramite la quale i dati vengono inseriti nel calcolatore per essere elaborati;
4. **Unità di *output***, necessaria affinché i dati elaborati possano essere restituiti all'operatore;
5. **Bus**, un canale che collega tutti i componenti fra loro.

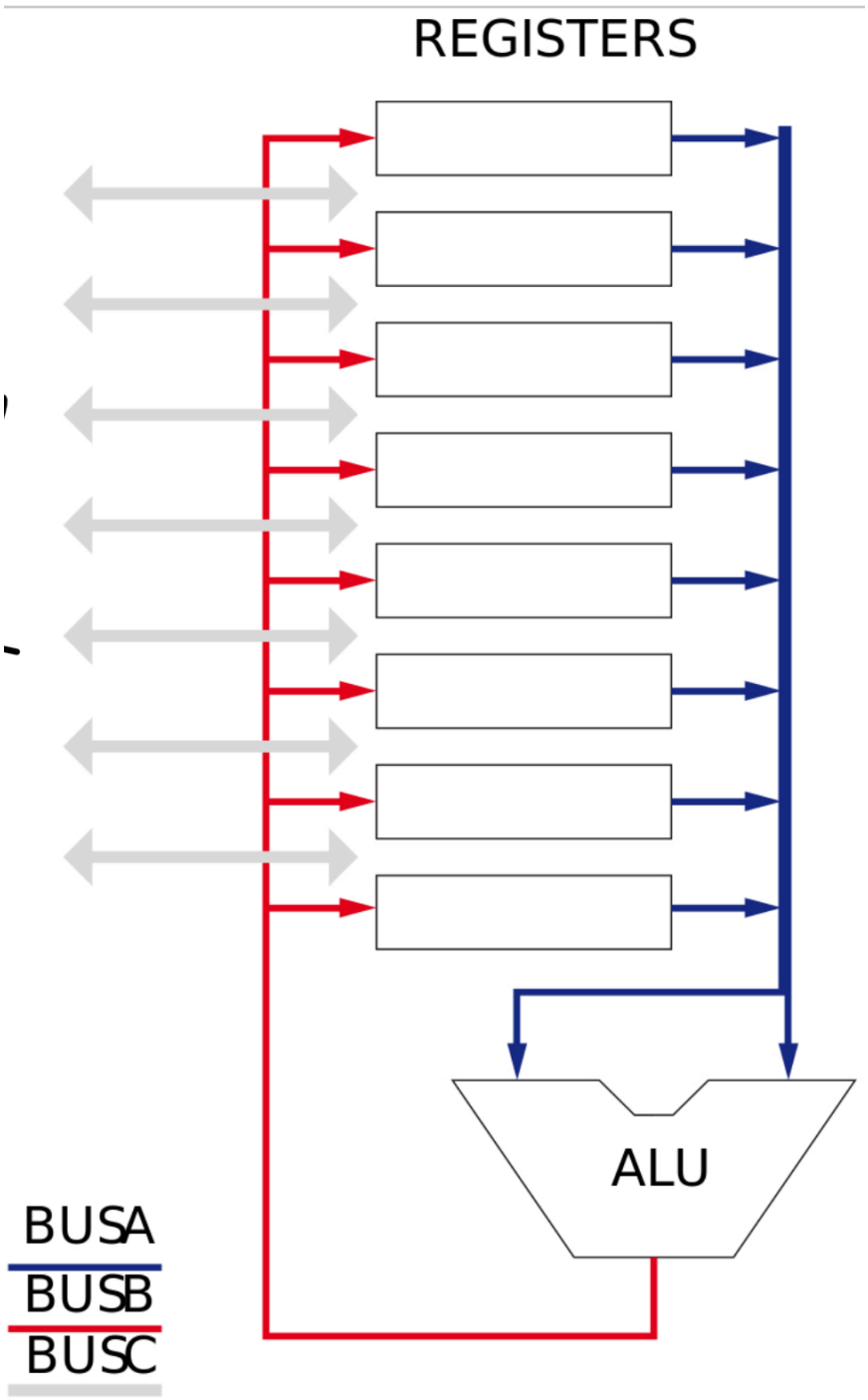
Queste cinque unità fondamentali vengono raggruppate a loro volta in quattro categorie o sottosistemi: DI INTERFACCIA, DI MEMORIZZAZIONE, DI ELABORAZIONE e BUS.

La **MACCHINA VIRTUALE** è una ipotetica macchina in cui il linguaggio macchina è L1 per cui quest'ultimo non è un linguaggio da dover più tradurre ma il linguaggio con cui questa macchina virtuale lavora.

I REGISTRI (in inglese: *processor register*), in informatica e nell'architettura dei calcolatori, è una piccola parte di memoria utilizzata per velocizzare l'esecuzione dei programmi fornendo un accesso rapido ai valori usati più frequentemente e/o tipicamente, i valori correntemente in uso in una determinata parte di un calcolo.
Il termine è usato spesso per riferirsi esclusivamente al gruppo di registri che possono essere direttamente indirizzati dalle istruzioni di input e output del microprocessore. Più propriamente, questi registri sono definiti "architected registers".
Per esempio, nell'architettura x86 è disponibile un set di otto registri utilizzabili dalle istruzioni del linguaggio macchina, ma la CPU conterrà molti più registri per uso interno o con funzioni speciali.
I registri costituiscono il punto più alto della gerarchia della memoria, e sono il meccanismo più rapido per il sistema di manipolare i dati. ***I registri sono normalmente misurati in base al numero di bit che possono contenere*** (ad esempio, registri a 8 bit o registri a 32 bit).
Le combinazioni sono dipendenti dal numero di bit in quanto esse sono numero di simboli ^ posizioni disponibili ——> 2 ^ 32 nel caso di 32 bit.

Il **data path** è un insieme di unità di calcolo, come ad esempio le unità di elaborazione (ALU), i registri e i moltiplicatori necessari per l'esecuzione delle istruzioni nella CPU.[1]
Il passaggio di due operandi attraverso la ALU e la memorizzazione del risultato in un nuovo registro viene detto ciclo di data path. Tale ciclo definisce ciò che è in grado di fare una macchina: più veloce è il ciclo del data path, più veloce sarà la macchina. Ogni istruzione ISA (assembly) viene eseguita in uno o più cicli di data path; diversi cicli sono necessari per istruzioni complesse come, per esempio, la 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.
Il data path, insieme all'unità di controllo, costituisce il nucleo stesso della CPU (core). Altri tre componenti che completano la CPU sono: la memoria, l'unità di ingresso (input) e l'unità di uscita (output).



LA MULTI PROGRAMMAZIONE in informatica, modo di funzionamento di un calcolatore elettronico digitale ad accessi multipli che dà luogo all'esecuzione contemporanea di diversi programmi applicativi, indipendenti tra loro; ciò consente di sfruttare appieno la velocità operativa dell'unità centrale e di utilizzare in modo intelligente i tempi morti di attesa che si verificano specie durante il trasferimento di dati e di comandi con le unità periferiche. A tal fine, il calcolatore seziona in modo opportuno i programmi in esecuzione e li sviluppa intervallandoli tra loro; la cadenza degli intervalli di tempo destinati a ciascun programma è gestita autonomamente dal sistema di elaborazione stesso che ne determina anche la successione in modo ottimale; tra i programmi è generalmente prefissata una scala di importanza crescente che condiziona la suddivisione del tempo in base alle richieste, con la priorità prestabilita.

La **commutazione di contesto** (in inglese **context switch**), in informatica, indica una particolare operazione del sistema operativo che conserva lo stato del processo o thread, in modo da poter essere ripreso in un altro momento. Questa attività permette a più processi di condividere la CPU, ed è anche una caratteristica essenziale per i sistemi operativi multitasking.
Un altro significato del context switch è quello dovuto agli interrupt, ovvero quando un processo richiede l'uso di un disco o di attività di I/O, si richiede alla CPU di fermare il processo in esecuzione, salvare il suo stato con tutte le informazioni dei registri e altre varie informazioni, e di occupare la CPU nel soddisfare la richiesta di interrupt.

Discussione su Python e Java (linguaggio del web), Java è un esempio molto importante di linguaggio interpretato, in realtà Java viene prima tradotto in un suo standard ISA java bitecode (passando subito da livello 5 a livello 3) che viene poi interpretato ed eseguito dalla Java virtual machine.

https://www.dmi.unict.it/barba/FONDAMENTI/PROGRAMMI-TESTI/READING-MATERIAL/MA-HTML/6.htm