Architettura 1

Il cervello del calcolatore è la **CPU** (Central Processing Unit) può svolegere tutte le operazione purchè abbia delle istruzioni per farlo.

Un programma eseguibile è espresso come una *sequenza di operazioni*.

Normalmente usiamo linguaggi ad alto livello.

Ha due obiettivi:

- Funzioni **evolute** vicine al problema.
- Funzioni **semplici** da realizzare con hardware poco costoso.

Chiamiamo **macchina** M0 un'entità capace di eseguire un insieme definito di funzioni L0. Per esempio la parte hardware è chiamata M0 con linguaggio L0 (cioè l'insieme di istruzioni), il software è chiamato M1 con un linguaggio L1, può creare programmi chiamati P1.

> Il **Traduttore** è un programma P0 che partendo da P1, traduce ogni istruzione di linguaggio L1 in istruzioni di linguaggio L0.

> L'**Interprete** è programma che legge le istruzioni del P1 e le associa alle istruzioni riconosciute da M0.

Vari strati della macchina:

- Livello **0**: *logico* digitale
  - Sono i segnali tra i componenti fisici.
- Livello **1**: *microarchitettura*
  - Tutto quello che avviene a livello logico in un circuito, tutta la parte hardware.
- Livello **2**: architettura e *istruzioni*
  - Linguaggio macchina **ISA** (Instruction Set Architecture) il livello in cui sono specificati tutte le istruzioni del computer.
- Livello **3**: macchina e *sistema operativo*
  - Il nucleo (detto anche **kernel**) permette l'attivazione di un processo e l'interconnessione con altri, può anche far gestire le risorse di sistema dai processi.
- Livello **4**: liguaggio *assemblativo*
  - Livello di macchina virtuale più basso per un programmatore, il linguaggio assembly ne è un esempio.
- Livello **5**: linguaggi *alto livello* (C, Java ecc...)
  - Livello della macchia virtuale utilizzato dai programmatori.

Ognuno di questi livelli ha una Mn (**Macchina**), Ln (**Linguaggio**) e Pn (**Programma**).

Struttura di un calcolatore

- **Processore** (Central Process Unit) è il cervello.
  - Alu (Aritmethic Logic Unit)

    - Registri
    - Unità di controllo
- Memoria **Ram** (Random Access Memory).
- Periferiche Input/Output.

  Sono costrutti connessi tramite *bus.* All'interno della cpu vengono lette informazioni dalla cache, passate all'**Alu** sarà salvato di nuovo nei registri tutto controllato dall'unità di controllo, ogni istruzione è interpretata e poi quest'ultima indirizzerà alla giusta locazione.

Attraverso i transitor si possono realizzare porte logiche come **AND**, **OR**, **NOT**, **XOR** e molte altre. Sul circuito stampato della motherboard vi sono piste di materiale conduttore che trasmettono segnali.
Ci sono diversi tipi di registri: a uso
speciale e **generali.** Della seconda categoria troviamo **PC** (**Program Counter**) che contiene l'indirizzo della prossima istruzione e l'**IR** (**Instruction Registrer**) che contiene l'indirizzo dell'attuale istruzione in esecuzione.

Esecuzione di un'istruzione

1. Si preleva l'istruzione dal **PC** e si inserisce **nell'IR**. (**FETCH**)
2. Si aggiorna il **PC** che punterà alla **prossima istruzione**.
3. Si determina il tipo si istruzione caricata. (**DECODE**)
4. Se l'istruzione usa dati presenti in memeoria, si individua l'indirizzo e si prelevano i dati.

Esecuzione tramite **interprete**: uno dei vantaggi è quello di mantenere più semplice l'architettura hardware sottostante e di estendere l'insieme delle istruzioni messe a disposizione dal processore.

1. Si esegue l'istruzione. (**EXECUTE**)
2. Si torna al passo 1.

Il microinterprete viene memoriazzato in una **CONTROL STORE**: una memoria di sola lettura **ROM** molto veloce per ridurre il ritardo dell'interprete rispetto all'esecuzione diretta.

- **RISC** (Reducex Istruction Set Computer) → permette di eseguire più velocemente la istruzioni frequenti senza interprete.
- **CISC** (Complex Istruction Set Computer) → permette di eseguire più azioni in una istruzione.

In entrambi i casi è stata introdotta una **PIPELINE** per mettere in parallelo le istruzioni.

Spiegazione della PIPELINE:

- Stadio **1**: Unità di fetch istruzione.
- Stadio **2**: Decodifica istruzione.
- Stadio **3**: Unità di fetch degli operandi.
- Stadio **4**: Esecuzione dell'istruzione.
- Stadio **5**: Unità di salvataggio del risultato.

Principi di progettazione RISC

Le istruzioni (di linguaggio macchina, livello 2 **ISA**) devono essere realizzate in hardware, le istruzioni devono essere esguite in parallelo, istruzioni facili da decodificare, solo il load/store operano sulla memoria, molti registri disponibili grazie al **control store**.

Rappresentazione interna dei dati

Tutte le informazioni all'interno del computer sono rappresentate sotto forma di **sequenze di bit.**
Bit = Binary DIgit
che è una cifra rappresentata da 0 o 1. Per convenineza possono essere raggruppati. Esempio **8 bit = 1 byte.**

Organizzazione della Ram

La memoria RAM viene organizzata in celle grandi 1byte o più. Ogni cella ha un suo indirizzo che la identifica univocamente e può contenere un dato. Il tipo di **memoria è volatile** ciò significa che una volta tolta la carica i dati verranno persi. La cpu può svolgere due funzioni: **read** o **write** su di essa un dato nell cella con un determinato indirizzo.

Conversione da base 10 a base 2

Metodo 1.

- Trovare tutte le potenze di 2 contenute in N, sottrarre il numero della potenza dal numero in base 10, prendiamo il risultato e continuiamo fino a che non abbiamo 0.

Esempio con 49: la potenza più vicina è 32 quindi 2^5, 5 caselle, otteniamo 17, il 16 è il più vicino quindi 2^4, abbiamo 1, 2^0 è 1.

1	1	0	0	1

Metodo 2.

- Eseguiamo divisioni a catena per 2, ogni volta teniamo a mente il R nella tabella e dal basso verso l'altro scriviamo il numero in cifra binaria fin quando il quoziente (risultato della divisione) non è 0.

Numero	Divisone	Resto
49:2	24	1
24:2	12	0
12:2	6	0
6:2	3	0
3:2	1	1
1:2	0	1

Dal basso verso l'alto è 11001

Virgola fissa

Un numero binario con parte frazionaria è comunemente rappresentato come un base decimale un esempio è 11,0101 per trasformalo in decimale dobbiamo moltiplicare la parte dopo la virgola per la relativa potenza di due negativa.

Esempio con 11,0101 che diventa che come risultato genera: 3,3125.

Virgola mobile

Dato un determinato numero dobbiamo usare il formato IEEE754 con il quale abbiamo a disposizione una griglia nella quale abbiamo una cella per il segno (1 bit), una per esponente (8 bit) e poi per la mantissa (23 bit).

Esempio con -13,25 che diventa 1101,01 che diventa 1,10101.

Segno	**Esponente + 127**	**Parte Frazionaria**
1	10000010	10101000000000000000000

Per calcolare cosa va affiancato al 127 dobbiamo prendere l'esponente del numero che moltiplica la cifra binaria, per ricavarlo all'origine dobbiamo scalare di tante posizioni fin quando non otteniamo 1,xxxxxxx e il numero di spostamenti aumenta di 1 l'esponente.

Se invece volessimo fare il contrario dobbiamo avere una tabella del tipo:

Segno	**Esponente + 127**	**Parte Frazionaria**
1	01111101	10010000000000000000

1,1001 * l'esponente lo ricaviamo dalla conversione di 01111101 che è 125 e per trovare l'esponente dobbiamo togliere 127, otteniamo -2. Se convertissimo il risultato in forma normale otteniamo: 0,011001 che convertito in decimale genera:

0,390625.

Precisione singola e doppia

La singola prevede 1 bit di segno, 8 per l'esponente e 23 per la mantissa con un totale di 32 bit. La precisione doppia prevede 1 bit di segno, 11 per l'esponente e

Numeri denormalizzati

Servono per recuperare i numeri che danno underflow, questi numeri hanno esponente uguale a 0, e 23 bit di parte frazionaria.

Aritmetica dei floating point

- **Moltiplicazione**: gli esponenti vengono sommati alle mantisse e le mantisse vengono moltiplicate, il numero viene normalizzato.
- **Divisione**: gli esponenti vengono sottratti e le mantisse divise e se serve normalizzare il risultato alla fine.
- **Addizione**: I numeri devono essere trasformati in una rappresentazione con lo stesso esponente e dopo possono essere sommate le mantisse.
- **Sottrazione**: i numeri devono essere trasformati in una rappresentazione con lo stesso esponente e dopo possono essere sottratti.

Porte logiche

Un calcolatore è un insieme di porte logiche opportunamente connesse, capaci di operare su segnali binari che sono i livelli di tensione, 0 **basso** e 1 **alto.** L'algebra booleana serve a gestire le variabili booleane ovvero quelle che hanno valori 0 o 1, ci sono tre costrutti: variabili, operatori e funzioni. Per indicare le **variabili booleane** usiamo lettere dell'alfabeto $x = 0$ oppure $y = 1$. Gli **operatori booleani** invece sono **not, and** e **or.** Definiamo la loro tavola di verità:

x	$\overline{x}$
0	1
1	0

Come possiamo osservare il valore del bit si **inverte**.

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

L'operazione dell'and genera **true** solo se **entrambi** i valori sono **true.**

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

L'operatore or invece genera **true** quando **almeno uno** dei due è **true.**

Possiamo anche costruire delle espressioni booleane quindi concatenando più operatori con variabili, possiamo anche esprimerle grazie a delle tavole di verità per verificare i loro valori. Con queste conoscenze di base possiamo capire al

meglio le porte logiche le quali per la maggior parte delle volte hanno **due** ingressi chiamti **input** (ovvero due bit da calcolare alla volta) e **un'uscità** chiamata **output**.

Transistor

Un transistor è dotato di tre componenti: la base, il collettore e l'emettitore. I transistor sono interrutori ognuno con una operazioni logica differente. Per poter generare qualcune porta è necessario disporre di un **NAND** o di un **NOR.** Grazie a questi due possiamo generare un **NOT**, **AND** e anche **OR.** Tutto grazie ai teoremi di **De Morgan.**

L'operatore XOR (Exclusive Or) si compone in questo modo:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Grazie a questo possiamo dimostrare $\overline{x}y + x\overline{y}$. Grazie a questo operatore booleano possiamo realizzare: **invertitore controllato**, **comparatore** e un **verificatore di parità**.

Minterm e Maxterm

Quando creiamo la tavola di verità possiamo stabilire dei **minterm** e dei **maxterm**, i primi si ricavano quando abbiamo uno 0 invertiamo la variabile e se è 1 lo lasciamo così al contrario i maxterm rimangono così se 0 e si negano se 1, vediamo un esempio:

x	y	$minterm$
0	0	$\overline{x} * \overline{y}$
0	1	$\overline{x} * y$
1	0	$x * \overline{y}$
1	1	$x * y$

La somma di prodotti **SP è il minterm**, dobbiamo verificare quando la funzione è uguale a 1 e sono il quel caso possiamo prendere i valori delle variabili e poi successivamente creare la somma di prodotti, presa la riga con risultato uguale a 1, dobbiamo vedere il valore della variabile dove se la variabile è 0 dobbiamo invertirla altrimenti lasciarla così. Il prodotto di somme **PS è il maxterm,** viene ricavato quando la funzione è 0 e dobbiamo prendere i valori nella relativa riga, se abbiamo 0 lasciamo il valore così altrimenti lo invertiamo.

x	y	$maxterm$
0	0	$x + y$
0	1	$x + \overline{y}$
1	0	$\overline{x} + y$
1	1	$\overline{x} + \overline{y}$

Esempio di minterm (SP): $(\overline{x} * y) + (\overline{y} * x) + (\overline{x} * \overline{y})$.

Esempio di maxterm (PS): $(\overline{x} + y) * (\overline{y} + x) * (\overline{x} + \overline{y})$.

Esercizi architettura

a,b,c	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0
F(a, b, c)	0	1	1	1	1	1	0

a. Ricavare l'espressione in forma normale somma di prodotti che descrive la funzione.

b. Ricavare l'espressione tramite mappa di **Karnaugh**.

c. Disegnare il circuito con le porte adeguate.

Per poter svolgere la funzione in minterm, prendiamo dove ci sono gli 1 in $F(x)$, se c'è 1 teniamo normale altrimenti neghiamo: $\overline{a}\overline{b}c + \overline{a}b\overline{c} + \overline{a}bc + a\overline{b}\overline{c} + a\overline{b}c$.

Disegniamo ora la mappa di **Karnaugh,** andiamo a mettere 1 dove la nostra configurazione è presente**:**

ab/c	0 0	0 1	1 1	1 0
0		1		1
1	1	1		1

La distanza di Hamming dice quanti bit differiscono per esempio 110 e 101 ha distanza di 2 dato che hanno 2 bit differenti.

Possiamo raggruppare le colonne 01 con la 10 e la righa 00 con 01, associamo ora una formula. Costruiamo il nuovo **minterm** semplificati, dobbiamo prendere l'ultima colonna che è 100 o 101 dobbiamo vedere quello cge rimane uguale ovvero $a\overline{b}$. Prendiamo ora la seconda colonna 010 e 011 e vediamo che rimane uguale 01 ovvero e l'ultima è 001 e 011 le variabili che rimangono uguali sono la a la c quindi 01 che diventa $\overline{a}c$. La nuova formula ricavata è $a\overline{b} + \overline{a}b + \overline{a}c$.

Per quanto riguarda il maxterm abbiamo 000 110 e 111 che devono essere invertiti se 1 e lasciati così se 0: $(a + b + c) * (\overline{a} + \overline{b} + c) * (\overline{a} + \overline{b} + \overline{c})$. Possiamo anche semplificare:

ab/c	0 0	0 1	1 1	1 0
0	0		0	
1			0	

Da qui ricaviamo 000 che non può essere ragruppato, 110 e 111 sono uguali i due 1 quindi ricaviamo: $(\overline{a} + \overline{b}) * (a + b + c)$.

x	y	$minterm$
0	0	$\overline{x} * \overline{y}$
0	1	$\overline{x} * y$
1	0	$x * \overline{y}$
1	1	$x * y$

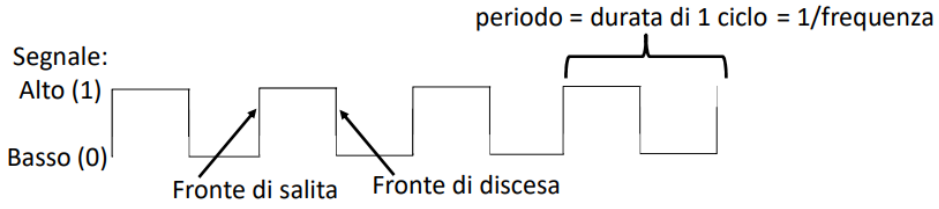
Il Clock

Il clock è un generatore di impulsi che serve a gestire gli eventi nel computer. Per esempio nella CPU si compie un percorso dati completo (una fase del ciclo di FETCH) a ogni cambio di stato, **alto** (1) o **basso** (0).

Il periodo è uguale alla frequenza ed è il lasso di tempo va ci mette il clock a tornare alto da uno stato alto.

Durante un ciclo di clock nella CPU avvengono varie fasi: si **preparano** gli operandi nei registri di input della ALU, la ALU **esegue** l'operazione e si **salva** il risultato in un registro di output.

Alcuni eventi impegano più cicli di clock e possono essere iniziati al fronte di salita o al fronte di discesa.

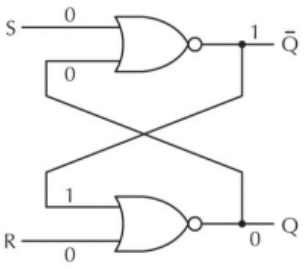


Il clock può anche generare dei segnali con ritardi differenti, così da creare onde con fronti di salita e discesa diversi.

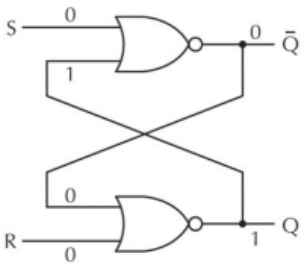
Circuiti sequenziali

Latch

Prendiamo per esempio il circuito **LATCH:** una memoria a 1 bit, è un circuito che è composto nel seguente modo:



(a)



(b)

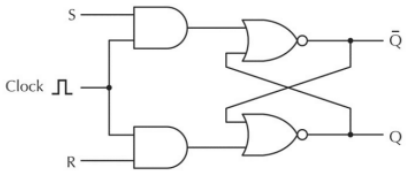
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

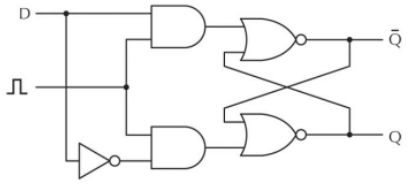
(c)

La nomenclatura è la seguente: R (Reset), S (Set) e Q (Output). Se entrambe le due uscite sono a 1 lo **stato non è stabile.** Ricordiamo in oltre che S e R possono essere entrambi 0 oppure uno dei due 0 a l'altro 1 ma NON entrambi a 1.

Il caso instabile viene generato quando entrambi sono a 1 dato che il valore del prossimo sarà indeterminato, l'obiettivo di questo circuito è mantenere il risultato precedente. Se entrambi sono a 0 si mantiene, se S è a 1 e R è a 0 si usa il SET, viceversa il RESET.

Per risolvere il proble di S a 1 e R a 1 possiamo usare il seguente circuito:

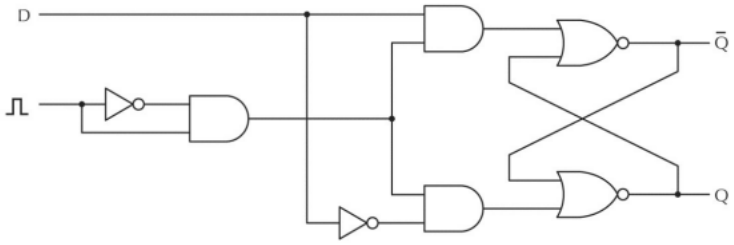




Utilizzando un latch D si risolve quello dell latch SR.

Flip Flop D

Anziché attivarsi quando il clock è alto, il flip flop d viene attivato quando è basso grazie a una porta logica not.



Si presenta nel seguente formato. I due diversi tipi di circuito generano diversi tipi di memorie RAM, le attuali diffuse in commercio sono: SRAM, DRAM, SDRAM.

Organizzazione della memoria

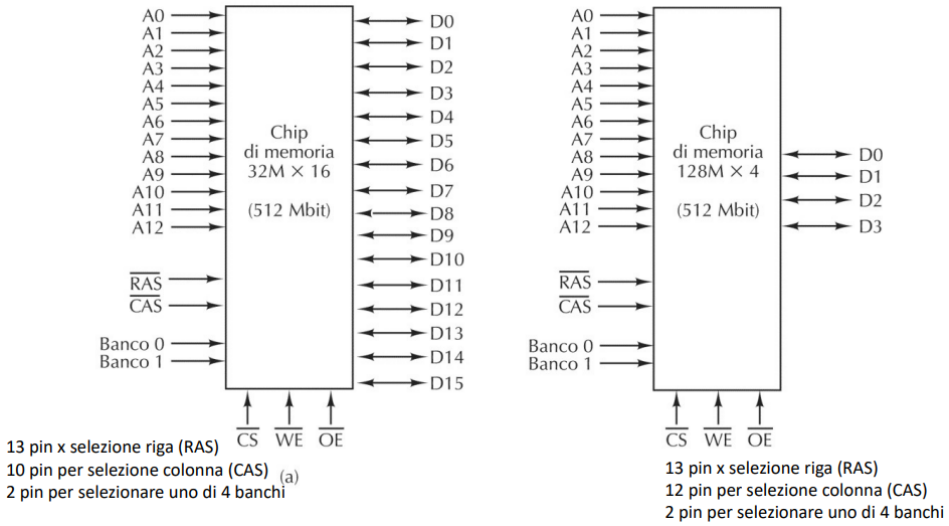
RAS (Row Address Strobe) e **CAS** (Column Address Strobe) indicano che una memoria è organizzata a matrice, vi possono anche essere presenti due pin per indicare quale dei 4 banchi scegliere.

Nel primo caso abbiamo 32 Megabyte di celle suddivise in celle da 16 bit. Sappiamo che sono suddivisi in questo modo:

- 2^4 bit sono per le celle (16 bit).
- 2^{13} bit per le RAS.
- 2^{10} bit per le CAS.
- 2^2 bit per la scelta del banco.

Nel secondo caso abbiamo la memoria di 128 Megabyte le cui celle sono da 4 bit. Sappiamo che sono suddivisi in questo modo:

- 2^2 bit sono per le celle (4 bit).
- 2^{13} bit per le RAS.
- 2^{12} bit per le CAS.
- 2^2 bit per la scelta del banco.



Esercizi architettura

Scheda composta da 8 chip con una capacità da 4 Gbyte. Celle di dimensione 1 byte.

a. Capacità di ciascun chip? Dobbiamo dividere 4 Gbtye con gli 8 chip trasformiamo in potenze di 2 quindi 4 Gbyte = $2^2 * 2^{30}$ mentre 8 chip sono 2^3 quindi abbiamo 2^{29} bit per ogni cella ovvero 512 Mbyte per chip.

b. Numero di bit per ogni indirizzo? Quale è la dimensione della memoria.

Gerarchia delle memorie

Le memorie si classificano in base alla loro velocità, dalle più vicine alla CPU a quelle più esterne. La prima che analizziamo è la **cache** ovvero una memoria ad alta velocità che serve per restringere il tempo di accesso alla RAM. A livello fisico si trova esternamente alla CPU. Il motivo per cui la cache non può avere le dimensioni della RAM è perché costa molto produrla.

Analizzando le sequenze di accesso alla memoria generaste dai programmi osserviamo due località: **spaziali** e **temporali**. La località è le previsione dell'accesso alla prossima sequenza, se si ripresenta nella **stessa cella** in un periodo è

temporale, se si presenta nelle celle **vicine spaziale**.

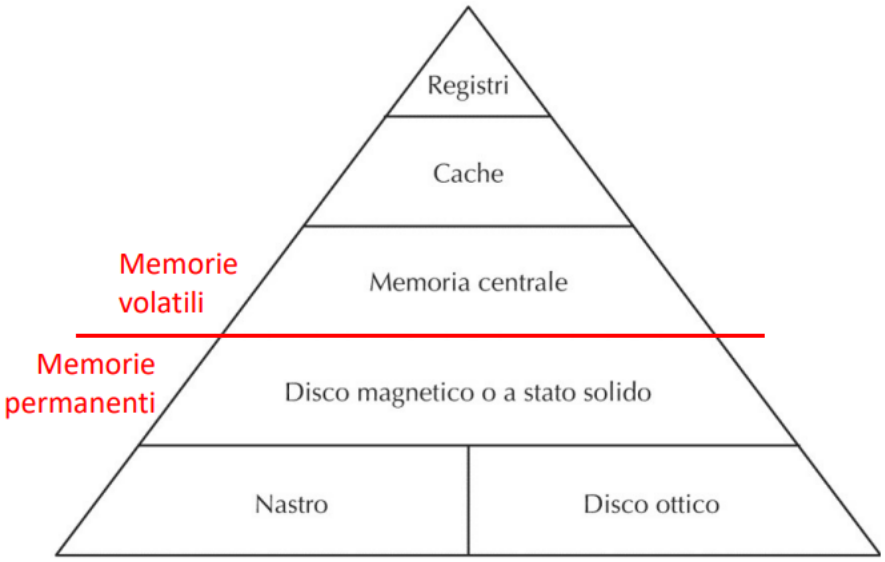
- Località spaziale
  - Quando l'esecuzione avviene in sequenza prelevando istruzioni collocate in posizioni contigue.
  - Le variabili locali di una funzione sono nello stack o le celle di un vettore.
- Località temporale
  - Quando in un ciclo while o for andiamo a cambiare la stessa variabile per esempio incrementandola, quindi accesso alla stessa cella più volte in un periodo.

L'**hit rate** è la frequenza di accessi della CPU alla cache e si divide in due. Se avviene un **cache hit** allora il dato è presente nella cache, se avviene un **cache miss** vuole dire che il dato non era presente in cache e deve essere prelevato dalla RAM.

Tipologia	Tempo
h	tc
$1-h$	$t_{ram}+tc$

Il tempo di accesso è indicato dalla seguente formula: $t_{acc}=t_c+(1-h)*t_m$.

La gerarchia è la seguente: **registri**, **cache**, **ram**, **ssd** e **hdd**.



> Pagina nuovo argomento: **microarchitettura**.
>
> Microarchitettura