GERARCHIE DI MEMORIA
mercoledì 28 dicembre 2022 10:29

GERARCHIE DI MEMORIA

PER RISOLVERE IL GAP SI INTRODUCE IL CONCETTO DI **GERARCHIE DI MEMORIA**!

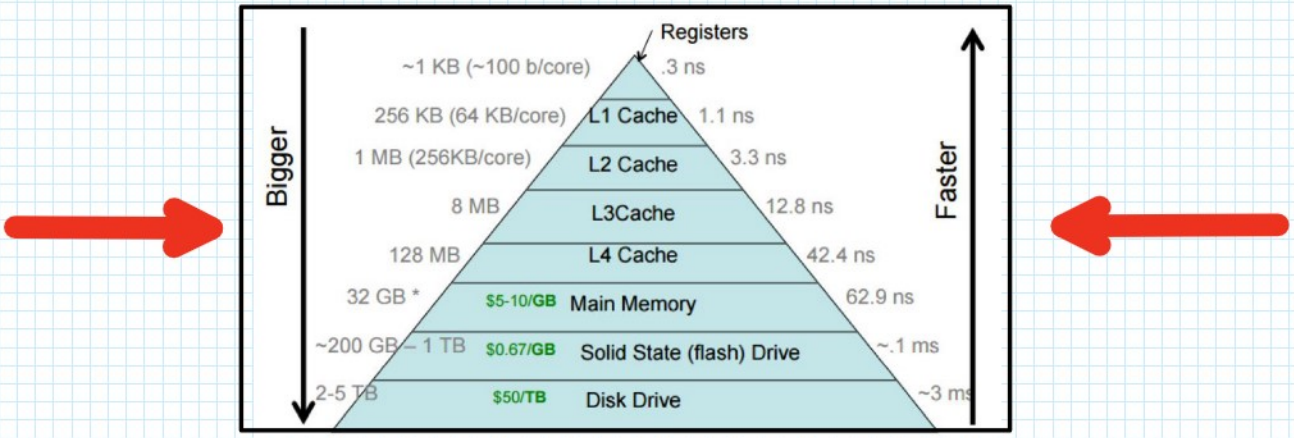
↓

TRA CPU E MEMORIA

IL PROCESSORE NON DIALOGA PIÙ DIRETTAMENTE CON LA MEMORIA PRINCIPALE, MA TRA CPU E MEMORIA, INSERIAMO DEI COMPONENTI CHE CERCANO DI ESSERE UN PO' PIÙ VELOCI.

I Registri sono degli elementi di memoria che vanno alla stessa velocità del precessore e la capacità non è massima: il registro è composto da tanti flip flop che a sua volta sono transistor e non ho un costo contenuto.

SI ANDRÀ A REALIZZARE UNA MEMORIA TAMPONE!



Queste cache sono altre memorie che mantengono copie dei dati e queste copie dei dati possono essere utilizzate per servire delle richieste di memoria, sia in lettura che in scrittura;
Andando verso su ci avviciniamo al processore, e man mano che saliamo usiamo tecnologie più costose e tecnologie più costose ci permettono di avere latenze di accesso minori;
Andando verso giù, i tempi di accesso e le dimensioni aumentano!

CERCHEREMO DI FAR TROVARE AL PROCESSORE TUTTI QUANTI I DATI CHE GLI SERVONO, PIÙ IN ALTO POSSIBILE NELLA GERARCHIA DI MEMORIA;
E I TEMPI SARANNO RIDOTTI PERCHÈ VIAGGEREMO ALLA VELOCITÀ DEL LIVELLO DI CACHE IN CUI OPERIAMO.

Il processore chiederà di accedere ai dati soltanto alla CACHE DI PRIMO LIVELLO.
SE IL DATO NON È DISPONIBILE LA CACHE di primo livello andrà di prelevarlo dalla cache di secondo livello... e così via fino ad arrivare alla memoria PRINCIPALE!

CERCHIAMO DI BUFFERIZZARE I DATI RENDENDOLI PIÙ VICINI AL NOSTRO PROCESSORE;

ESEMPIO

- Intel Core i9-10900K (10th gen), Q2 2020
- Tempi e frequenze *indicativi* per accedere ad un blocco di dati in modo casuale

	Reg.	L1i	L1d	L2	L3	RAM	Disco
Dimensione	1K	32 kB	32 kB	256 kB	20 MB	128GB	TB
Latenza (cicli)	1	4	4	12	57	253	10^7
Latenza (sec)	0,2ns	0,8ns	0,8ns	2,4ns	11,4ns	50,5ns	0,45ms - 2ms
Hz / IOPS	3,70G	1,25G	1,25G	416M	87M	19M	50-100000

Ho due cache di primo livello;

L1i e L1d

L'ARCHITETTURA HARVARD CERCAVA DI RISOLVERE IL COLLO DI BOTTIGLIA CREANDO UNA MEMORIA PER I DATI E PER LE ISTRUZIONI!
MA È UN'ARCHITETTURA CHE È STATA ABBANDONATA perché NON POSSO SOVRASCRIVERE la MEMORIA DATI! PERÒ È UNA SOLUZIONE PER ABBATTERE IL COLLO DI BOTTIGLIA;

ESISTONO DUE CACHE DI PRIMO LIVELLO: UNA PER LE ISTRUZIONI E UNA PER I DATI;
RIDUCO L'ARCHITETTURA HARVARD SOLO AL LIVELLO DI CACHE!
TORNERÀ UTILE NEL MODELLO PIPELINE!
Se riusciamo ad avere i dati che ci servono più in alto la fetch più veloce!
LA MEMORIA DELLE CACHE È MOLTO PIÙ PICCOLA DELLA RAM, SIGNIFICA CHE LE ISTRUZIONI E I DATI CHE POSSO METTERE SONO MINORI RISPETTO ALLE ISTRUZIONI CHE POSSONO ESSERE MESSE NELLA RAM!
Le cache, così come i registri, mantengono un sottoinsieme dei dati!
IL DATO IN CACHE È UNA COPIA DEI DATI IN MEMORIA PRINCIPALE, NON VIENE SOLO TRASFERITO.
Con l'unico obiettivo quello di massimizzare la velocità!

I DATI E LE ISTRUZIONI, che ANDIAMO AD UTILIZZARE IN UNA CERTA FINESTRA DI TEMPO, IN UN PROGRAMMA SONO QUELLI: PRINCIPIO DI LOCALITÀ!

Immaginiamo le istruzioni assembly che implementano un ciclo, continuiamo a girare sempre sulle stesse istruzioni. Ci sarà sicuramente un momento in cui queste istruzioni non ce l'ho nella mia gerarchia di memoria perché chiaramente all'inizio le istruzioni le ho solo in RAM, quindi ci sarà un momento in cui dovrò prelevare queste istruzioni ed effettuare una copia nella cache di primo livello istruzioni; poi deve MANTENERE queste istruzioni nella cache di primo livello così se sto ancora girando riesco ad accedere facilmente alle istruzioni!

Nella normale vita di un programma, noi eseguiamo il programma e la fase di fetch fa passare all'istruzione successiva.
Quindi c'è questa sequenzialità nell'accesso di istruzioni!
ACCEDO ALLE ISTRUZIONI IN FORMA SEQUENZIALE O RIPETO LE ISTRUZIONI!
QUESTO FUNZIONA ANCHE PER I DATI, PERCHÉ SE HO UN VETTORE LO ACCEDO IN MANIERA

ACCEDO ALLE ISTRUZIONI IN FORMA SEQUENZIALE O RIPETO LE ISTRUZIONI!
QUESTO FUNZIONA ANCHE PER I DATI, ANCHE SE HO UN VETTORE IO ACCEDO IN MANIERA SEQUENZIALE AI SUOI DATI.

- **Località temporale:** se si è effettuato un accesso ad un elemento, è *probabile* che nel breve tempo si effettuerà un altro accesso allo stesso elemento (esempio: le istruzioni in un ciclo)
- **Località spaziale:** se si è effettuato un accesso ad un elemento, è *probabile* che nel breve tempo si effettuerà un accesso ad un elemento vicino (esempio: accesso sequenziale ad un vettore)

DOBBIAMO ORGANIZZARE LE NOSTRE ARCHITETTURE IN BASE AI NOSTRI PRINCIPI!
COSÌ MIGLIORIAMO LE PRESTAZIONI DELLE NOSTRE APPLICAZIONI!