La memoria si organizza in modo gerarchico. Questa organizzazione permette di trovare un buon compromesso tra capacità di acceso (quindi velocità all'acesso al dato, che può essere il dato o l'istruzione di memoria) ed il costo. Quindi questi 3 elementi principali.

Il nostro desiderato ottimale è quello di avere un costo minimo, una grandezza elevata e tempo di accessi minimi. Questa soluzione non è realizzabile perchè le memorie che hanno un elevata frequenza di accesso sono memorie molto costose e con capacità di memorizzazione bassa. Quello che si fa è cercare di "approssimare" questa soluzione ideale con la tecnica delle gerarchie di memoria.

A seconda del livello di memoria possiamo utilizzare tecnologie diverse per soddisfare al meglio i 3 requisiti.

Un programma non accede mai a tutte le istruzioni e a tutti i dati contemporaneamente. I nostri dati e le nostre istruzioni hanno probabilità diverse di essere richieste in memoria. Se abbiamo quindi un programma che utilizza diversi dati in diversi momenti con probabilità diverse; quindi possiamo usarr questo principio per progettare una gerarchia di memoria organizzata in livelli in modo da caratterizzare ogni livello da una velocità, dimensione e costo.

A parità di capacità, quindi le memorie più veloci hanno un costo più elevato per singolo bit rispetto quelle più lente.

L'hardware e il SIstema Operativo sono le due parti del calcolatpre che gesticono l'insieme di memorie rganizzate gerarchicamente che contengono delle repliche dei dati in mood che la CPu trovi quel dato nella memoria più vicina. L'obiettivo è definire delle politiche per accedere ai dati più utili e li vogliamo trovare ai livelli più alti, nelle memorie più veloci.

Memoria interna alla cpu: caratterizzata da altà velocità e limitate dimensioni
Memoria centrale: dimensioni maggiori e tempi di accesso più elevati. Accessibile in modo diretto tramite indirizzi.
Nei sistemi attuali le cache si trovano tra questi due livelli.
Le memorie secondarie sono basso costo, lente e molto capienti e non volatili.

Per garantire l'efficienza sfruttiamo il principio di località, ovvero il principio tale per cui un programma in qualunque istante può usare una specifica porzione di memoria limitata nello spazio di indirizzamento. La cpu richiede un dato, per questo principio sono definite due politiche in modo tale che i dati vengono posizionati cosi che sono piu facili da recuperare.
Località temporale: predilige l'accesso ad una informazione che è stata utilizzata recentemente
Località Spaziale: quando ci riferiamo ad un elemento c'è la tendenza a fare riferimento ad altri elementi vicino ad esso.

Un livello di memoria vicino al processore contiene un sottoinsieme di dati memorizzati in ogni livello sottostante e tutti i dati si trovano nel livello più basso.

Le memorie possono essere organizzati in blocchi. Sono unità di informazione minimo che viene scambiato tra le gerarchie.
Hit: l'informazione richiesta si trova nel livello immediatamente inferiore di memoria.
Miss: l'informazione richiesta non si trova nel liv. immediatamente inferiore ed occorre andare nel livello più lontano dalla cpu.

Hit Rate: frazione degli acecssi in memoria nei quali l'informazione è stata trovata nel liv. necessario
Miss Rate: frazione degli accessi in memoria nei quali l'informazione non è stata trovata nel liv. immediatamente inferiore. (1- Hit rate)

Tempo di hit: tempo necessario per prelevare il dato nel livello più vicino+ il tempo di verifica se è un hit/miss

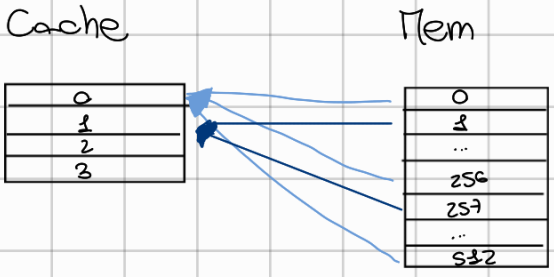
Tempo di miss: Tempo necessario per sostituire un blocco del livello in cui ci troviamo con un nuovo blocco del livello inferiore della gerarchia+ il trasferimento del dato al processore.

L'algoritmo di Caching si basa sui due principi:
Mantiene i blocchi richiesti recentemente vicino alla cpu (temporale)
muove blocchi contigui di memoria che contendon i dati richiesti (spaziale)

Ci sono 3 tipi di cache:

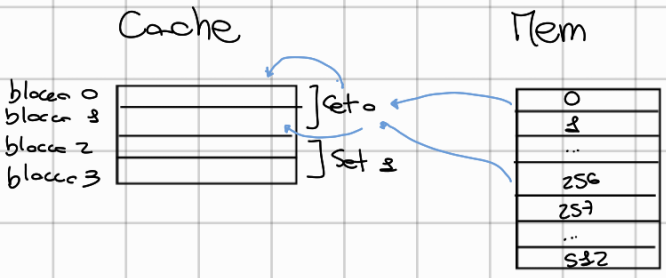
Mappaggio diretto: a ciascun blocco della memoria corrisponde una specifica locazione nella cache
Fully Associative: ogni blocco può essere collocato in qualunque posizione.
Set Associative: Soluzione intermedia, la caratteristica principale che ogni blocco di memori ha un numero fisso di specifice locazioni nella cache.

Il problema della direct mapping che a diverse locazioni in memoria può corrispondere lo stesso blocco in cache.



Per risolvere ciò è stato introdotto la fully associative, dove ogni blocco della memoria viene inserito in qualsiasi locazione , in modo che non avrem più il "conflitto".
Il problema però è che essendo un blocco della memoria può essere in qualsiasi locazione della cache, quando la cpu deve cercarlo una ricerca potrebbe richiedere troppo tempo e non avremmo efficienza. Per migliorare si potrebbe pensare ad una ricerca parallela invece di una sequenziale, ma questa soluzione potrebbe risultare troppo costosa.

Nelle set associative avremo i blocchi della cache divisi in set, e dunque non abbiamo più il conflitto della direct così frequente, perchè possiamo inserirli in blocchi diversi, però restando sempre nel set, e quindi ricerca più semplice della fully.



Partendo dalla mappatura diretta: per trovare il blocco che corrisponde ad un indirizzo della mem. centrale bisogna fare:
(indirizzo blocco) modulo (numero blocchi cache)

Esempio: Se ho 64 blocchi da 16 byte, l'indirizzo 1200 si ricava come:

ind.blocco= 1200/16=75
num blocchi cache = 64

75 mod 64 = 11 .

1200 quindi sarà nell'11 blocco della cache.

Ho 8 blocchi di cache. Per sapere dove andrà un blocco faccio il log: 3.
Se devo cercare dove si trova un indirizzo dovrò guardare i suoi ultimi 3 bit, perchè quei bit indicheranno la posizione all'interno della cache. Infatti 3 bit generano le 8 possibili combinazioni della cache (000, 001, 010, 011, 100, 101, 110, 111)

Il tag contiene le informazioni necessarie per verificare se una parola corrisponde alla parola cercata.
L'indice seleziona il blocco di cache in cui andare a cercare. (log di num blocchi)
Bit di validità: bit che indica se il dato è valido o meno.
Bit di Offset: 2 bit per selezionare il byte richiesto all'interno della parola.

Se abbiamo un indirizzo da 32 bit, per calcolare i bit di tag bisogna sottrare 32 -indice-offset.
Per avere un hit dobbiamo avere corrispondenza tra il tag del nostor indirizzo e il tag del campo della cache e il bit di validità deve essere true, altrimenti abbiamo miss.

Se abbiamo più word (es. 2 word: 64 bit) Ci serve anche un bit chiamato Word Selector, per selezionare la word che ci interessa.

Un ampia dimensione del blocco permette di sfruttare meglio la località spaziale, ma presenta molti difetti.
Se la dim del blocco è troppo grossa rispetto alla dim cache, aumenta miss rate.
Definendo la miss penalty come la misura quanto tempo, quando un dato non esiste, speso per accedere alla cache di livello inferiore; esso tende ad aumentare all'aumento della dimensione di blocchi grandi.

Nelle set associative, una volta stabilito il set, per determinare se un indirizzo è present in un blocco ; bisogna confrontare in parallelo tutti i tag ti tutti i blocchi appartenenti al set

Nella DIrect mapped non ci poniamo il problema della sostituzione del blocco perchè esiste una sola locazione in cui esso potrà andare. Nelle set e fully invece abbiamo o un certo numero o in tutta la cache bisogno di sostituire un blocco quando dobbiamo inserirne uno nuovo.
Ci sono diverse politiche di sostituzione di blocchi:
Random-> semplice da implementare ma potremmo rimuovere blocchi importanti
LRU: sfruttando la località temporale andiamo a sostituire quello che non si utilizza da più tempo.
FIFO; viene rimosso il più vecchio.

In ogni caso stiamo facendo delle previsioni sul futuro, non sappiamo davvero chi ci servirà in futuro, ma possiamo fare delle previsioni di chi potrebbe servirci meno.

I Miss in lettura possono essere di due tipi: istruzione e dato.
Nel momento in cui un istruzione presente in un blocco non è nella cache, la cpu va in stallo e un'altra unità di controllo va nei livelli inferiori della cache e lo porta su fino alla cache.
Se il miss non riguarda un istruzione ma un dato , la cpu continua fino a quando il dato non deve essere utilizzato.

Quando invece dobbiamo scrivere un dato in memoria abbiamo 2 tecniche principali :

Write through per mantenere equilibrio e coerenza tra i livelli della cache scrive il dato nel livello e nel livello inferiore. Abbiamo coerenza ed è semplice ma lavoriamo all'efficienza del livello inferiore e c'è traffico sui bus di sistema.
Write back vengono scritt nel solo livello della cache e quando deve essere sostituito viene scritto nel liv. inferiore. I Vantaggi sono più velocità e solo scritture successive alterano la cache lo svantaggio è che non c'è coerenza tra i livelli.

Abbiamo anche il write through con write buffer dove vengono scritto oltre al livello anche ad un cache che ha tipicamente 4 elemeneti, gestito in FIFO. per garantire efficienze la velocità di aggiornamento del buffer deve essere molto poiu piccolo alla vrelocità in cui si sostituiscono i blocchi, altrimenti perdiamo tempo.

Un write miss è un tentativo di scrittura in un blocco che non è presente in cache: Abbiamo 2 soluzione
- Write allocate: il blocco vine trasferito in cache e si effettua la scrittura con write back
- No Write Allocate: Il blocco viene scritto nella memoria di livello inferiore, senza trasferimento alla Cache.